# Discovering Compositional Structure

by
Matthew T. Harrison
B.A., University of Virginia, 1998
Sc.M., Brown University, 2000

Doctoral dissertation
Ph.D. Advisor: Stuart Geman

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy
in the Division of Applied Mathematics at Brown University

Providence, Rhode Island
May 2005

This dissertation by Matthew T. Harrison is accepted in its present form by the Division of Applied Mathematics as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date ______________ ________________________________________

Stuart Geman, Advisor

Recommended to the Graduate Council

Date ______________ ________________________________________

Elie Bienenstock, Reader

Date ______________ ________________________________________

David Mumford, Reader

Approved by the Graduate Council

Date ______________ ________________________________________

Karen Newman, Dean of the Graduate School

# Vita

Matthew Harrison was born April 10, 1976, in Atlanta, Georgia, and spent much of his childhood in the Northern Neck area of Virginia. He attended the University of Virginia as a Jefferson Scholar, receiving a B.A. in Mathematics and Cognitive Science in 1998. There he met and married his wife, Marianne. He then attended graduate school at Brown University funded by a National Defense Science and Engineering Graduate Fellowship and a National Science Foundation IGERT Fellowship. He studied under Professor Stuart Geman in the Division of Applied Mathematics and received a Sc.M. in Applied Mathematics in 2000 and (with the completion of this thesis) a Ph.D. in the same field in 2005.

# Acknowledgements

First and foremost, I am grateful to God for this opportunity and for surrounding me with supporting family and friends, especially, my wife, Marianne. I have been heavily influenced by my advisor, Stuart Geman. Many of the ideas in this thesis are directly attributable to him. I would also like to thank the members of my committee – Stuart Geman, Elie Bienenstock, and David Mumford – for taking the time to read my thesis and for many useful comments and discussions throughout my time at Brown. Britt Anderson and Wilson Truccolo (and the neurophysiology labs that they represent) graciously provided data for the jitter experiments and also spent many long hours helping me test and modify the various jitter algorithms.

Naturally, there is a long list of other friends and colleagues who I would like to acknowledge: past and present students of Stu's with whom I have had many useful discussions – Han Amarasingham, Brian Lucena, Ting-Li Chen, Yanchun Wu and Ya Jin; my other officemates – Aaron Hoffman, Jim Zhang, Mario Micheli, and Matt Feiszli; Brown's local Matlab guru – Joe Hicklin; and the incredibly responsive and helpful administrative staff in the Division of Applied Math.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 A hierarchy of reusable parts?

Humans and many animals demonstrate the remarkable ability to successfully learn and generalize from only a few examples, or sometimes just a single example. From a statistical point of view, this indicates the influence of strong and accurate biases or priors [8]. Understanding these priors is crucial for understanding biological learning and for creating machine learning algorithms that can mimic the performance of biological systems.

A highly simplified view of the human (or primate) visual cortex is illustrated in Figure 1.1. This view pertains to the ventral visual pathway which is suspected to be heavily involved in object recognition (see, for example, [20]). It emphasizes certain widely believed generalities about the types of visual stimuli that are likely to elicit a strong neural response in different regions along the pathway. In particular,

- A neuron near the beginning of the pathway, close to the retina, will typically respond well only to a particular simple feature at a highly specific location in the visual field. This feature might be a blob or an oriented edge at a particular scale and location (e.g., [12, 21]).
- A neuron near the end of the pathway, close to inferotemporal cortex (IT), will typically respond well only to a particular complex feature and the response can be invariant to large changes in presentation, such as location, scale or context. The feature might be a particular object or face anywhere within a large portion of the visual field (e.g., [18]).
- There are neurons in the middle of the pathway that respond well to stimuli in between these two extremes (e.g., [13]).

The degree to which this is a useful caricature for understanding human object recognition is still controversial. Nevertheless, it is consistent with the following view about the neural representation underlying object recognition:

- The representation is parts-based and arranged hierarchically.

- Near the bottom of the hierarchy, the parts are simple, localized and not selective for objects. A given part might participate in the representation of many different objects depending on the specific viewing conditions.

- Near the top of the hierarchy, the parts are selective for a specific object, but invariant to many different viewing conditions. A given part can be loosely thought of as an object or an object detector.

- From the bottom of the hierarchy to the top, the parts have increasing selectivity (for specific objects) and increasing invariance (to presentation or viewing conditions).

We refer to this type of representation as a *hierarchy of reusable parts*. Representations that share some or all of these characteristics have found interesting applications in the computer vision literature, especially for object detection and recognition. For example,

- *Hierarchical representations* (in increasing feature complexity) allow coarse-to-fine computation strategies. This improves the efficiency of object detection. Simple features lower in the hierarchy can be quickly computed at many scales and locations. Based on this information, computational resources can be directed to more specific locations and scales, where more diagnostic and more computationally demanding features can be computed (e.g., [2]).

- *Reusable features* allow computations to be shared for multiple object detection and recognition. Instead of having completely different computational routines for each object, in which case computation grows linearly with the number of recognizable objects, feature sharing might allow the computation to grow sublinearly. Indeed, several groups have demonstrated logarithmic-like growth (e.g., [14, 19]). (Note that this is essentially an empirical statement about the nature of the world; *a priori*, objects need not share enough features for sublinear growth.)

- *Parts-based models* provide an explicit background model which might be useful for object recognition in clutter. They explicitly allow an image region to contain the parts of an object without containing the object itself. Many object recognition schemes can be loosely conceptualized as testing an object model versus a no-object model and determining the winner. One common failure mode is when an image region contains several parts of the object, but not the object, that is, the parts are not in the appropriate configuration. Conceptually, at least, parts-based models can remedy this problem. Stuart Geman refers to this principle as "objects define their own background model."

A hierarchy of reusable parts might also provide an effective bias (or prior) for fast visual learning. If the parts of an object have already been learned and can be easily recognized and detected across many viewing conditions and in many different contexts, then learning to recognize the object simply involves learning to identify an appropriate configuration of the constituent parts. Learning effectively takes place in a highly parameterized and low-dimensional space. Most statistical learning paradigms require such spaces in order to have fast and accurate learning.

Under this hypothesis, the hierarchy is learned from the bottom up. At any given time, the existing hierarchical representation supports certain visual tasks, like object recognition and detection, presumably taking advantage of the computational improvements outlined above. In many ways the knowledge embodied in the hierarchy represents what is typically referred to as "the prior" in a Bayesian object recognition framework. So learning the hierarchy is like learning this prior. These ideas are discussed in great detail in the context of cortical organization in Friston (2003) [6].

Although the hierarchy is learned in a bottom-up manner, the computations supported by the hierarchy can allow information to flow in any direction. Indeed, in later chapters we experiment with generative, hierarchical, probabilistic graphical models, for which feedback is an inherent part of any computation. The use of feedback within a hierarchy is strongly consistent with the anatomical organization of (primate) visual cortex [4, 6]. Representations based on hierarchies of reusable parts are also consistent with the (seemingly) compositional nature of human cognition.

## 1.2 Compositionality

Fodor and Pylyshyn (1988) [5] take the closely related principles of productivity, systematicity and compositionality as hallmarks of the symbolic and combinatorial nature of cognition. Of course, these representational principles would be of little use in a world that admitted no such representation. But, at least intuitively, our world is productive – different arrangements of the same parts can create functionally different objects – and it is systematic – there are regular rules governing how these parts can come together and how they can be used – and it is definitely compositional – an object remains the same in a variety of contexts.[1] Attempts at creating computational models that demonstrate these properties often focus on compositionality, since productivity and systematicity tend to arise naturally from compositionality. For this reason and for convenience, we loosely use the term *compositionality* for all three principles.

Compositionality as an overarching principle for computational vision has been pioneered by Geman, Bienenstock and colleagues [1, 9]. Much of the work is theoretical, however, the Ph.D. theses of Potter [17] and Huang [11] describe some computational experiments. Related work includes much of the early work on syntactic pattern recognition, for example, [15, 7, 10], and also Feldman's perceptual theories [3].

The general framework described in the previous section (namely, representations based on stochastic hierarchies of reusable parts increasing in selectivity and invariance) is one instantiation of a compositional representation. The specific ideas discussed in this thesis were developed within the context of this broader framework. Our main focus is on learning: What principles can be used to discover useful compositional representations from data in an unsupervised way?

[1] This could be circular: maybe our world appears productive, systematic and compositional because we perceive it through a system that has these properties. Nevertheless, such a system presumably gives some selective advantage based on the properties of our world.

## 1.3 The organization of this thesis

This thesis is divided into two main parts. The first part focuses on what sorts of learning strategies could be used to create a hierarchy of reusable parts from natural images. To that end, we introduce one such learning strategy and then describe some toy experiments with applying the strategy to natural image data. The second part of the thesis focuses on several statistical methods designed to investigate compositionality within neural systems.

Each Chapter is more or less self contained, especially the four main Chapters 3, 4, 6, and 7, some of which originally appeared as technical reports. Mathematical notation is not entirely consistent across chapters and each chapter has its own bibliography. The figures for each chapter follow its bibliography. The brief concluding remarks in Chapter 8 pertain to the entire thesis. Several chapters use Matlab-like indexing: $x_{i:j} = (x_i, \ldots, x_j)$.

### 1.3.1 Part I: Natural scenes

In Chapter 2 we describe a general unsupervised learning heuristic for incrementally building hierarchical, parts-based models. The main idea is that dependencies are only incorporated into the model through the introduction of new parts.

In Chapter 3 we experiment with this learning heuristic on binary-valued natural images. An important insight from these experiments is that sparse representations are probably crucial for compositional learning. A major defect in the experiments is that the representations have no invariance, only selectivity.

At first glance, the learning heuristic appears designed to only learn selectivity, not invariance. A key observation from the literature, however, is that temporal information from image sequences can be used to learn invariant representations. In Chapter 4 we investigate whether or not the same incremental learning strategy can learn invariant representations when applied to image sequences. Again this involves simple experiments with binary-valued natural images.

We leave for future work the task of simultaneously learning selectivity and invariance. One foreseeable problem with hierarchies of increasing selectivity and invariance is *the Markov dilemma*: invariance at one level in the hierarchy hides information that might be useful for selectivity at higher levels. Some brief thoughts on these topics are collected in Chapter 5. The Markov dilemma, in particular, is used to motivate the jitter methods described in Chapter 7.

### 1.3.2 Part II: Neuroscience

In Chapter 6 we suggest several statistical methods that might be useful for investigating the response properties of neurons, especially neurons in the visual cortex. Agnostic, model-free methods for analyzing neural data are important for evaluating the degree to which the ventral visual pathway looks like a compositional hierarchy. This seems to be especially important in areas corresponding to the middle of the hierarchy, where the representation (if it is indeed compositional) is difficult to anticipate. Unfortunately, agnostic methods tend to require a lot of data, often more data than can be reasonably collected in a physiological experiment. The general theme of Chapter 6 is that the statistics of natural images might be useful for reducing the data collection demands. The methods are tested in simulations, but not in actual physiological experiments.

Chapter 7 explores a certain class of jitter methods that can be evaluated quickly and that can incorporate certain physiological constraints like refractory periods and bursting. Jitter methods are statistical techniques based on the intuition that locally perturbing observed spike times should provide a way to assess the fine temporal structure of a neural spike train while still preserving the classical firing rate. These methods are widely applicable to current questions in neuroscience about spike timing precision. In the context of compositionality within neural systems, certain types of spike timing have been suggested as a solution to the Markov dilemma (see Chapter 5).

## Bibliography

[1] Elie Bienenstock, Stuart Geman, and Daniel Potter. Compositionality, MDL priors and object recognition. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 838–844. MIT Press, Cambridge, 1998.

[2] Gilles Blanchard and Donald Geman. Hierarchical testing designs for pattern recognition. *The Annals of Statistics*, 33(3), June 2005 (to appear).

[3] Jacob Feldman. What is a visual object? *Trends in Cognitive Sciences*, 7(6):252–256, June 2003.

[4] D.J. Felleman and D.C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, 1(1):1–47, 1991.

[5] Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1–2):3–71, March 1988.

[6] Karl Friston. Learning and inference in the brain. *Neural Networks*, 16:1325–1352, 2003.

[7] K.S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, 1982.

[8] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias / variance dilemma. *Neural Computation*, 4:1–58, 1991.

[9] S. Geman, D. Potter, and Z. Chi. Composition systems. *Quarterly of Applied Mathematics*, LX:707–736, 2002.

[10] U. Grenander. *General Pattern Theory: A Study of Regular Structures.* Oxford University Press, 1993.

[11] Shih-Hsiu Huang. *Compositional approach to recognition using multi-scale computations.* PhD thesis, Division of Applied Mathematics, Brown University, 2001.

[12] H.D. Hubel and T.N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[13] E. Kobatake and K. Tanaka. Neuronal selectivities to complex object features in the ventral visual pathway of the macaque cerebral cortex. *Journal of Neurophysiology*, 71:856–867, March 1994.

[14] S. Krempp, D. Geman, and Y. Amit. Sequential learning with reusable parts for object detection. Cis, Johns Hopkins, 2002.

[15] T. Pavlidis. *Structural Pattern Recognition.* Springer-Verlag, 1977.

[16] Tomaso Poggio and Steve Smale. The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society*, 50(5):537–544, May 2003.

[17] Daniel Frederic Potter. *Compositional Pattern Recognition.* PhD thesis, Division of Applied Mathematics, Brown University, 1999.

[18] Keiji Tanaka. Inferotemporal cortex and object vision. *Annual Review of Neuroscience*, 19:109–139, 1996.

[19] Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Sharing visual features for multiclass and multiview object detection. AI Memo 2004-008, MIT, April 2004.

[20] Leslie G. Ungerleider and James V. Haxby. 'what' and 'where' in the human brain. *Current Opinion in Neurobiology*, 4(2):157–165, 1994.

[21] R.L. De Valois, D.G. Albrecht, and L.G. Thorell. Spatial frequency selectivity of cells in macaque visual cortex. *Vision Research*, 22(5):545–559, 1982.

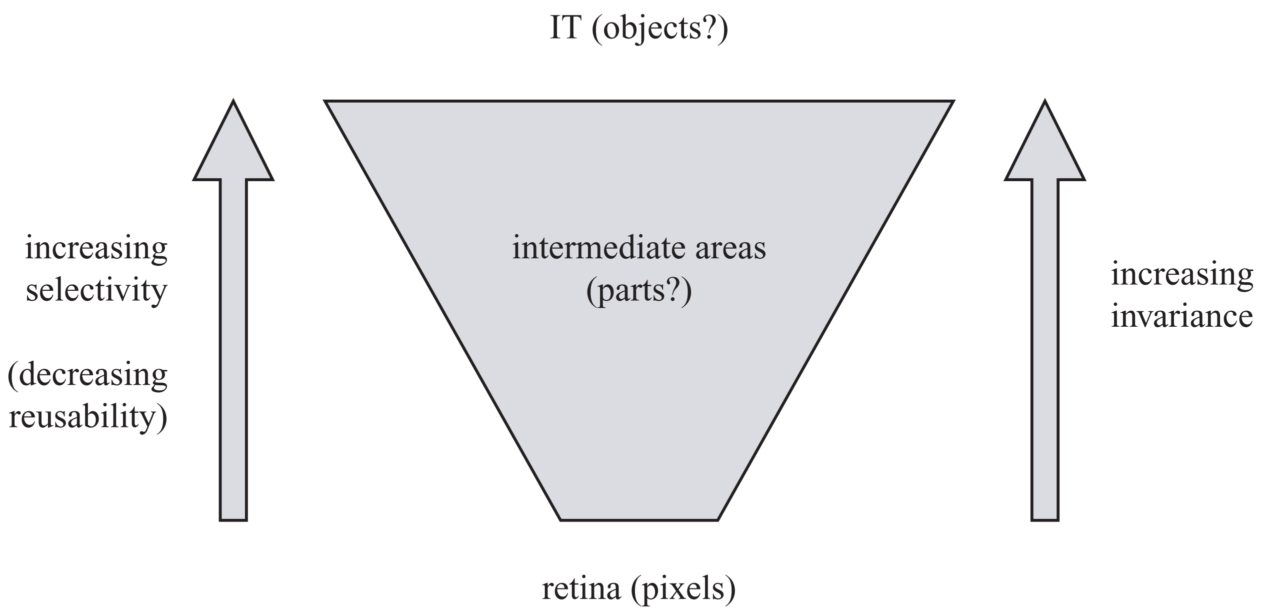

Figure 1.1: A caricature of neural response properties in the ventral visual pathway.

# Part I

# Natural scenes

# Chapter 2

# Model building by perturbation

## 2.1 Introduction

In this chapter we describe an unsupervised learning heuristic for incrementally building hierarchical, parts-based models. The hierarchy is built from the bottom up by adding new parts. In principle, the process of detecting and adding a new part is recursive, i.e., agnostic to the size of the hierarchy. Furthermore, it is not specific to vision problems, but could be potentially useful for other sensory modalities. The main idea is that dependencies are only incorporated into the model through the introduction of new parts.

A major deficiency of the setup here is that we do not address computation, only representation. We partially remedy this by formulating the problem in terms of probabilistic graphical models. Computation using these types of representations has been studied extensively and is still an active area of research. Nevertheless, efficient computation in compositional systems is currently an unsolved problem and it is not clear that the learning heuristic used here will create representations that can actually be used for computation. This becomes evident in Chapters 3 and 4 where we can only partially experiment with the learning heuristic.

The framework focuses on building an internal model that can represent certain aspects of the external world. The hope is that a good statistical model of the world that also happened to be compositional would likely be a useful representation for many visual tasks like segmentation and recognition. We do not directly address this issue here.

## 2.2 Learning heuristic

We want to interpret some data $Y$ in terms of various discrete parts $X$. $Y$ might be an image and $X$ might be lines, T-junctions and L-junctions of various lengths, scales, positions and orientations. $X$ is our internal representation of the external data $Y$.

One way to approach this problem is to define a generative model that specifies a prior distribution $P_X$ over the internal states $X$ and a conditional distribution $P_{Y|X}$ for the data given the internal states. Interpretation then proceeds in Bayesian manner using the posterior distribution $P_{X|Y}$ or some related quantity. We are not concerned here with the specifics of interpretation, but rather how to choose a good generative model.

### 2.2.1 Evaluating the model

The generative model implicitly defines a marginal distribution on the data:

$$P_Y(A) = \sum_x P_{X,Y}(x, A) = \sum_x P_{Y|X}(A|x)P_X(x).$$

This can be quite different from the "world's distribution" $\mathbb{P}_Y$ on the data. From a probabilistic point of view, the ideal goal is to have our distribution on $Y$ match the world's distribution on $Y$, that is

$$P_Y \overset{\text{goal}}{=} \mathbb{P}_Y.$$

This is a difficult goal to affirm; typically $Y$ takes values in a high-dimensional space. Collecting evidence for failure is somewhat easier.

Just as the model implicitly defines a distribution on $Y$, the world implicitly defines a distribution on $X$ by reversing the model:

$$\mathbb{P}_X(x) = \int_y P_{X|Y}(x|y)\mathbb{P}_Y(dy) = \mathbb{E}_Y\big[P_{X|Y}(x|Y)\big],$$

and similarly for any statistic $S = S(X)$ of the internal states:

$$\mathbb{P}_S(s) = \mathbb{E}_Y\big[P_{S|Y}(s|Y)\big],$$

where $\mathbb{E}_Y$ denotes expectation with respect to $\mathbb{P}_Y$. Note that since $X$ is discrete, $S$ is also discrete.

If $P_Y = \mathbb{P}_Y$, then $E_Y = \mathbb{E}_Y$ and

$$\mathbb{P}_X(x) = \mathbb{E}_Y\big[P_{X|Y}(x|Y)\big] \overset{\text{goal}}{=} E_Y\big[P_{X|Y}(x|Y)\big] = P_X(x).$$

Similarly,

$$\mathbb{P}_S(s) \overset{\text{goal}}{=} P_S(s).$$

If $S$ is simple, then this final implication of $P_Y = \mathbb{P}_Y$ might be a good place to look for problems with the model. In particular, approximating $\mathbb{E}_Y$ with an empirical distribution $\hat{\mathbb{E}}_Y$ over a collection of data $y_1, \ldots, y_n$, gives

$$\hat{\mathbb{P}}_S(s) = \hat{\mathbb{E}}_Y\big[P_{S|Y}(s|Y)\big] = \frac{1}{n}\sum_{k=1}^{n} P_{S|Y}(s|y_k) \approx \mathbb{E}_Y\big[P_{S|Y}(s|Y)\big] = \mathbb{P}_S(s) \overset{\text{goal}}{=} P_S(s).$$

A significant departure from this goal, namely

$$\hat{\mathbb{P}}_S \not\approx P_S, \tag{1}$$

suggests that $P_Y \not\approx \mathbb{P}_Y$ and indicates a problem with the model.

### 2.2.2 Improving the model

Suppose that for some statistic $S = S(X)$ we see the failure mode described in (1). One way to possibly improve the model is to modify the distribution of $S$ while leaving the rest of the model unchanged. In particular, since $S = S(X)$ is a function of $X$, we can express $P_X$ as

$$P_X(x) = P_{X,S}(x, S(x)) = P_{X|S}(x|S(x))P_S(S(x))$$

and then modify $P_S$ to get a new prior distribution on $X$. The data model $P_{Y|X}$ remains unchanged.

The particular suggestion here is to perturb $P_S$ by slightly mixing it with another distribution $P_S^1$ to get

$$P_S^\epsilon = (1-\epsilon)P_S + \epsilon P_S^1,$$

This creates new distributions on $X$ and $Y$, namely,

$$P_X^\epsilon(x) = P_{X|S}(x|S(x))P_S^\epsilon(S(x)) \quad \text{and} \quad P_Y^\epsilon(A) = \sum_x P_{Y|X}(A|x)P_X^\epsilon(x),$$

for $\epsilon \in [0,1]$. As long as $P_S^1$ does not put positive probability on any impossible (i.e., $P_S(s) = 0$) values for $S(X)$, everything makes sense. Note that $\epsilon = 0$ corresponds to no perturbation: $P_S^0 = P_S$, $P_X^0 = P_X$ and $P_Y^0 = P_Y$.

The perturbation improves the model if $P_Y^\epsilon$ is an improved approximation of $\mathbb{P}_Y$. We can quantify this with relative entropy:

$$D\big(\mathbb{P}_Y \big\| P_Y^\epsilon\big) \overset{\text{goal}}{<} D\big(\mathbb{P}_Y \big\| P_Y\big). \tag{2}$$

Since $D\big(\mathbb{P}_Y \big\| P_Y^\epsilon\big)$ is convex in $\epsilon$, it should be possible to choose a good $\epsilon$. (For convexity, see Lemma 2.2.2 below. To avoid technicalities, we will always assume that $D\big(\mathbb{P}_Y \big\| P_Y\big) < \infty$.) The hard part is finding an appropriate statistic $S$ and distribution $P_S^1$. The next theorem gives a potential search criterion. A proof can be found at the end of this section.

**Theorem 2.2.1.** Equation (2) is achievable for some $\epsilon$ if and only if

$$\mathbb{E}_S\left[\frac{P_S^1(S)}{P_S(S)}\right] > 1, \tag{3}$$

where $\mathbb{E}_S$ denotes expectation with respect to $\mathbb{P}_S$. If (3) holds, then $\mathbb{P}_S \neq P_S$.

The proof also shows that the larger the left side of (3) then the larger the improvement in relative entropy, at least for small perturbations ($\epsilon$ near 0).

Theorem 2.2.1 essentially says that we can improve the model if we can find a statistic $S$ and a distribution $P_S^1$ such that *on average* $S(X)$ is more likely under $P_S^1$ than $P_S$. The key is that *on average* means we average over internal states $X$ driven by the world's distribution on $Y$ and the current posterior distribution $P_{X|Y}$. Interpretation will typically be based on some approximation to the posterior, so the computations needed for interpretation under the current model should also produce the relevant statistics needed to evaluate potential improvements to the model.

All of this suggests looking for statistics $S$ and distributions $P_S^1$ such that

$$\hat{\mathbb{E}}_S\left[\frac{P_S^1(S)}{P_S(S)}\right] \gg 1, \tag{4}$$

where $\hat{\mathbb{E}}_S$ is expectation with respect to $\hat{\mathbb{P}}_S$.

**Proof of Theorem 2.2.1.** $P_X^\epsilon$ and $P_Y^\epsilon$ are also additive mixtures:

$$P_X^\epsilon = (1-\epsilon)P_X + \epsilon P_X^1 \quad \text{and} \quad P_Y^\epsilon = (1-\epsilon)P_Y + \epsilon P_Y^1.$$

Relative entropy is convex in both arguments, so $D(\epsilon) := D\big(\mathbb{P}_Y \big\| P_Y^\epsilon\big)$ is convex in $\epsilon$ on $[0,1]$. Since $D(0) = D\big(\mathbb{P}_Y \big\| P_Y\big)$, (2) is achievable if and only if $D'(0) < 0$.

Lemmas 2.2.2 and 2.2.3 compute $D'(0)$. The first gives

$$D'(0) = 1 - \mathbb{E}_Y\left[\frac{dP_Y^1}{dP_Y}(Y)\right]$$

and the second gives

$$\frac{dP_Y^1}{dP_Y}(Y) = E_{S|Y}\left[\frac{dP_S^1}{dP_S}(S)\middle| Y\right].$$

To apply the second lemma we take $(X,Y) = (S,Y)$ under the current model and $(X',Y') = (S,Y)$ under the $\epsilon = 1$ altered model with $P_S = P_S^1$. Recall that we assumed that $P_S^1$ puts probability one on the support of $P_S$, so $P_S^1 \ll P_S$, $P_X^1 \ll P_X$ and $P_Y^1 \ll P_Y$.

In the discrete setting here $[dP_S^1/dP_S](s) = P_S^1(s)/P_S(s)$. Also, by definition $\mathbb{E}_S[\cdot] = \mathbb{E}_Y[E_{S|Y}[\cdot|Y]]$. These give

$$D'(0) = 1 - \mathbb{E}_S\big[P_S^1(S)/P_S(S)\big],$$

which we need to be negative. Note that $\mathbb{P}_S = P_S$ gives

$$D'(0) = 1 - E_S\big[P_S^1(S)/P_S(S)\big] = 1 - \textstyle\sum_{s:P_S(s)>0} P_S^1(s) \geq 1 - 1 = 0$$

for any $P_S^1$, so (3) cannot hold. □

**Lemma 2.2.2.** Let $Q$, $P^0$ and $P^1$ be probability measures with $D(Q\|P^0) < \infty$ and $P^1 \ll P^0$. Define $P^\epsilon = (1-\epsilon)P^0 + \epsilon P^1$. Then $D(\epsilon) := D(Q\|P^\epsilon)$ is real valued on $[0,1)$ and convex on $[0,1]$ with $D'(0) = 1 - E_Q[dP^1/dP^0]$. (The derivative is only evaluated from the right.)

**Proof.** It is well known that relative entropy is always nonnegative, so $D(\epsilon) \geq 0$. Let $\tilde{P}^\epsilon$

represent the absolutely continuous component of $P^\epsilon$ with respect to $Q$. We have

$$
\begin{aligned}
D(\epsilon) &= E_Q \log \frac{dQ}{dP^\epsilon} = -E_Q \log \frac{d\tilde{P}^\epsilon}{dQ} = -E_Q \log \left[ (1-\epsilon)\frac{d\tilde{P}^0}{dQ} + \epsilon \frac{d\tilde{P}^1}{dQ} \right] \\
&= -E_Q \log \left[ (1-\epsilon)\frac{d\tilde{P}^0}{dQ} + \epsilon \frac{d\tilde{P}^1}{d\tilde{P}^0}\frac{d\tilde{P}^0}{dQ} \right] = -E_Q \log \left[ 1 + \epsilon \big( \frac{d\tilde{P}^1}{d\tilde{P}^0} - 1 \big) \right] - E_Q \log \frac{d\tilde{P}^0}{dQ} \\
&= D(Q \| P^0) - E_Q \log \left[ 1 + \epsilon \big( \frac{dP^1}{dP^0} - 1 \big) \right] = D(0) - E_Q \log \left[ 1 + \epsilon f \right],
\end{aligned}
$$

where the next to last equality holds because $d\tilde{P}^1/d\tilde{P}^0 = dP^1/dP^0$ a.s. $Q$ and where we define $f = dP^1/dP^0 - 1$. Note that $f \geq -1$, so we have the trivial bound $0 \leq D(\epsilon) \leq D(0) - \log(1-\epsilon) < \infty$ for $\epsilon \in [0,1)$. Note also that the concavity of the logarithm implies that $D(\epsilon)$ is convex on $[0,1]$.

Using the previous calculations gives

$$
D'(0) = \lim_{\epsilon \downarrow 0} \frac{D(\epsilon) - D(0)}{\epsilon} = \lim_{\epsilon \downarrow 0} \frac{-E_Q \log \left[ 1 + \epsilon f \right]}{\epsilon}.
$$

If we move the limit inside the expectation, then we get $D'(0) = -E_Q f = 1 - E_Q[dP^1/dP^0]$ as claimed. So we need only justify exchanging the limit and the integration.

Let $h_\epsilon = \epsilon^{-1} \log(1 + \epsilon f)$. Since $f \geq -1$, $h_\epsilon \geq \epsilon^{-1} \log(1-\epsilon) \uparrow -1$ as $\epsilon \downarrow 0$. On the set $\{f \leq 0\}$, $h_\epsilon \leq 0$ so the dominated convergence theorem can be applied here. On the set $\{f > 0\}$, $h_\epsilon > 0$ and we can write

$$
h_\epsilon = \left[ \frac{\log(1+\epsilon f)}{\epsilon f} \right] f,
$$

which is increasing as $\epsilon \downarrow 0$. The monotone convergence theorem completes the proof (even in the case where $E_Q[dP^1/dP^0] = \infty$). $\square$

**Lemma 2.2.3.** Let $(X,Y)$ and $(X',Y')$ be random elements with regular conditional distributions $P_{Y'|X'} = P_{Y|X}$ and with $P_{X'} \ll P_X$. Then $P_{Y'} \ll P_Y$ and

$$
\frac{dP_{Y'}}{dP_Y}(Y) = E\left[ \frac{dP_{X'}}{dP_X}(X) \middle| Y \right] \text{ a.s.}
$$

**Proof.** For absolute continuity, note that $P_Y(B) = 0$ implies $\int_B P_{Y|X}(dy, x) = 0$ a.s. $P_X$ and thus a.s. $P_{X'}$. Replacing $P_{Y|X}$ with $P_{Y'|X'}$ and integrating w.r.t. $P_{X'}$ shows that $P_{Y'}(B) = 0$.

For the main result, we will show that the left side satisfies the definition of the conditional expectation on the right side. Clearly the left side is $\sigma(Y)$-measurable. If $A \in \sigma(Y)$, then

$A = \{Y \in B\}$ for some $B$ and we can compute

$$\begin{aligned}\int_A \frac{dP_{X'}}{dP_X}(X)dP &= \int_{\mathcal{X}\times B} \frac{dP_{X'}}{dP_X}(x)P_{X,Y}(d(x\times y)) \\ &= \int_{\mathcal{X}} \frac{dP_{X'}}{dP_X}(x)P_X(dx)\int_B P_{Y|X}(x,dy) = \int_{\mathcal{X}} P_{X'}(dx)\int_B P_{Y'|X'}(x,dy) \\ &= \int_{\mathcal{X}\times B} P_{X',Y'}(d(x\times y)) = P_{Y'}(B) = \int_B \frac{dP_{Y'}}{dP_Y}(y)P_Y(dy) = \int_A \frac{dP_{Y'}}{dP_Y}(Y)dP. \qquad \square\end{aligned}$$

### 2.2.3 Updating the model

Once we have found a candidate statistic $S = S(X)$ and a distribution $P_S^1$ that can improve the model via a perturbation, we need to update the model to incorporate this improvement. We will do this by introducing a new part $Z$ to the representation $X$ in such a way that the new distribution on $S$ is exactly $P_S^\epsilon$.

Let $Z \in \{0,1\}$. Define the prior on the new state space $X' = (X,Z)$ as

$$P_{X'}(x,z) = P_{X|S}(x|S(x))\big[(1-z)(1-\epsilon)P_S(S(x)) + z\epsilon P_S^1(S(x))\big],$$

and define the data model as

$$P_{Y|X'}(A|(x,z)) = P_{Y|X}(A|x).$$

The marginal of $X$ under $P_{X'}$ is exactly $P_X^\epsilon$. $Z$ indicates which of the two mixture components is present with $Z = 1$ corresponding to the new component $P_X^1$. The data model remains the same, so the marginal of $Y$ under the new model is $P_Y^\epsilon$ which is the intended perturbation of $P_Y$.

Suppose $X$ has multiple components (parts) $X = (X_1,\ldots,X_N)$ and suppose $P_X$ respects the dependency graph $G$ with vertices corresponding to the $X_i$'s, i.e., $P_X$ factors into a product of functions defined only on the cliques of $G$. Then $X' = (X_1,\ldots,X_N,Z)$ and $P_{X'}$ respects the graph $G'$ which adds a vertex $Z$ to $G$ and adds a clique $(X_S,Z)$, where $X_S$ are those components of $X$ that $S$ depends on, i.e., $S(x) = S(x_S)$. This follows from the alternative representation

$$\begin{aligned}P_{X'}(x,z) &= P_{X|S}(x|S(x))\big[(1-\epsilon)P_S(S(x))\big]^{(1-z)}\big[\epsilon P_S^1(S(x))\big]^z \\ &= \frac{P_X(x)}{P_S(S(x))}\big[(1-\epsilon)P_S(S(x))\big]^{(1-z)}\big[\epsilon P_S^1(S(x))\big]^z \\ &= P_X(x)(1-\epsilon)\left(\frac{\epsilon P_S^1(S(x_S))}{(1-\epsilon)P_S(S(x_S))}\right)^z.\end{aligned}$$

If $G$ facilitates computation under the original model and if $X_S$ is small, then $G'$ will likely facilitate computation under the new perturbed model.

#### 2.2.3.1 Parameter estimation

The perturbed model depends on a parameter $\epsilon = P_Z(1)$. We need to choose an appropriate value of $\epsilon$ in order to guarantee that the perturbed model is better than the original, that is, to guarantee (2). (Recall that $P_Y^\epsilon$ is the perturbed model's distribution on $Y$.) Ideally, we want to choose $\epsilon$ that minimizes $D\big(\mathbb{P}_Y \big\| P_Y^\epsilon\big)$.

The minimizing $\epsilon$ satisfies the fixed point equation

$$\mathbb{P}_Z(1) = P_Z(1) = \epsilon, \tag{5}$$

where $\mathbb{P}_Z(z) = \mathbb{E}_Y\big[P_{Z|Y}(z|Y)\big]$ and where $P_Z$ and $P_{Z|Y}$ refer to the perturbed model with parameter $\epsilon$. Note that each term in (5) depends on $\epsilon$. A proof is given at the end of this section, where we also show that the minimizing $\epsilon$ exists and is unique.

The fixed point (5) makes sense intuitively. When we introduce a new variable $Z$ into the model, we want its distributions under the model and under the world to be the same.

Finding the fixed point (approximately) is more or less straightforward. $\mathbb{P}_Z$ can be approximated by $\hat{\mathbb{P}}_Z$ which should arise naturally from using the model for interpretation. The parameter $\epsilon$ can then be adjusted up or down appropriately, perhaps with a simple neural network-like learning rule or perhaps by recursively setting $\epsilon = \hat{\mathbb{P}}_Z(1)$. This latter strategy is a stochastic version of the EM algorithm.

**Proof of (5).** We assume that $P_Y^1 \neq P_Y$, which, for example, is implied by Theorem 2.2.1 when (3) holds. (If they are equal then $\epsilon$ has no effect on the model, i.e., $P_Y^\epsilon = P_Y$, and (2) cannot hold.) So $D\big(\mathbb{P}_Y \big\| P_Y^\epsilon\big)$ is strictly convex in $\epsilon$ on $[0,1]$ and it has a unique minimizer $\epsilon^*$.

Consider the perturbed model with $\epsilon = \epsilon^*$. In order to derive a contradiction, suppose that $\mathbb{P}_Z(1) \neq P_Z(1) = \epsilon^*$. Fixing the perturbed model, we can use the setup from Section 2.2.2 to imagine perturbing the perturbed model. In particular, we will substitute $Z$ for $S$, $\mathbb{P}_Z$ for $P_S^1$ and $\delta$ for $\epsilon$. We will also use $P_{Y'}$ and $P_{Y'}^\delta$ to denote the new marginals on $Y$. Since $\mathbb{E}_Z\big[\mathbb{P}_Z(Z)/P_Z(Z)\big] > 1$, Theorem 2.2.1 implies that $D\big(\mathbb{P}_Y \big\| P_{Y'}^\delta\big) < D\big(\mathbb{P}_Y \big\| P_{Y'}\big)$. But $P_{Y'} = P_Y^{\epsilon^*}$ and $P_{Y'}^\delta = P_Y^\epsilon$ for some $\epsilon \neq \epsilon^*$, which contradicts the fact that $\epsilon^*$ is the unique minimizer of $D\big(\mathbb{P}_Y \big\| P_Y^\epsilon\big)$.

We can derive a similar contradiction in the other direction by supposing that $\mathbb{P}_Z(1) = P_Z(1) \neq \epsilon^*$. Using the same changes in notation, except now substituting $(1-\epsilon^*)^{(1-z)}\epsilon^{*z}$ for $P_S^1$, Theorem 2.2.1 implies that $D\big(\mathbb{P}_Y \big\| P_{Y'}^1\big) \geq D\big(\mathbb{P}_Y \big\| P_{Y'}\big)$. But this is impossible since $P_{Y'}^1 = P_Y^{\epsilon^*}$ and $P_{Y'} = P_Y^\epsilon$ for $\epsilon \neq \epsilon^*$. □

### 2.2.4 Recursive model building

The above method begins with a candidate statistic $S$ and distribution $P_S^1$. If these can improve the model, say (4) holds, then the model is perturbed slightly by adding a new variable $Z$. $Z$ interacts with and modifies the joint distribution of the components of $X$ that $S$ depends on. The perturbation is local: the state space, its distribution and presumably computation are all slightly modified, but only in the neighborhood of $Z$.

In principle, this model perturbation strategy can be applied recursively to incrementally grow a large graphical model in an unsupervised fashion. We interpret each new categorical variable as a feature, or a part. A new part modifies the distribution of a certain subset of previous parts, which we can think about as the subparts of the new part. This introduces a natural hierarchy. Since the subsets can overlap, the parts are reusable.

Mathematically, each new part adds a new mixture component to the model's implicit distribution on the data. Viewed in this way, the learning strategy approximates the true distribution on the data with a large mixture model. The criterion is likelihood. In practice, all of the typical issues that arise from using large mixture models, such as overfitting and robustness, are likely to be of great importance.

## 2.3 Suspicious coincidences

Another major issue is how to generate candidate statistics $S$ and distributions $P_S^1$. One possibility is to look for departures from independence, that is, each statistic $S$ depends on a subset of components of $X$ that are independent under the current model and each distribution $P_S^1$ introduces dependencies among these components. If these are the only type of statistics entertained by the model, then all dependencies arise from parts. In a sense, dependencies are parts.

Suppose that the current model $X = (X_1, \dots, X_N)$ has two parts $X_i$ and $X_j$ that are independent under the model and suppose that their respective distributions have been tuned to fit the world's distribution: $\mathbb{P}_{X_k} = P_{X_k}$, $k = i, j$. Consider the candidate statistic $S(X) = S(X_i, X_j) = \mathbb{1}\{X_i \in A_i, X_j \in A_j\}$ for $i \neq j$. Since $X_i$ and $X_j$ are independent under the model, the distribution of $S$ is easy to compute, namely,

$$P_S(1) = P_{X_i,X_j}(A_i \times A_j) = P_{X_i}(A_i)P_{X_j}(A_j) = \mathbb{P}_{X_i}(A_i)\mathbb{P}_{X_j}(A_j).$$

If we detect evidence that

$$\mathbb{P}_S(1) = \mathbb{P}_{X_i,X_j}(A_i \times A_j) \neq \mathbb{P}_{X_i}(A_i)\mathbb{P}_{X_j}(A_j) = P_S(1),$$

then $X_i$ and $X_j$ are dependent under the world's distribution and we would like to incorporate this into the model.

When $\mathbb{P}_S(1) > P_S(1)$, the distribution $P_S^1(s) = \mathbb{1}\{s = 1\}$, which is the point mass at 1, satisfies

$$\mathbb{E}_S\left[\frac{P_S^1(S)}{P_S(S)}\right] = \mathbb{P}_S(1)\frac{1}{P_S(1)} + 0 > 1,$$

and Theorem 2.2.1 implies that we can improve the model with $S$ and $P_S^1$. Note that

$$\mathbb{P}_S(1)\frac{1}{P_S(1)} = \frac{\mathbb{P}_{X_i,X_j}(A_i \times A_j)}{\mathbb{P}_{X_i}(A_i)\mathbb{P}_{X_j}(A_j)},$$

so the criterion of interest becomes

$$\frac{\mathbb{P}_{X_i,X_j}(A_i \times A_j)}{\mathbb{P}_{X_i}(A_i)\mathbb{P}_{X_j}(A_j)} > 1. \tag{6}$$

If $X_i$ and $X_j$ are modeled as independent, then (6) is called a *suspicious coincidence.*

Clearly this generalizes to candidate statistics of the form

$$S(X) = S(X_{i_1}, \ldots, X_{i_m}) = \mathbb{1}\{X_{i_1} \in A_{i_1}, \ldots, X_{i_m} \in A_{i_m}\}$$

for independent $X_{i_k}$'s whose marginal distributions are the same under the model and under the world. The criterion then becomes an $m$th-order suspicious coincidence,

$$\frac{\mathbb{P}_{X_{i_1},\ldots,X_{i_m}}(A_{i_1} \times \cdots \times A_{i_m})}{\mathbb{P}_{X_{i_1}}(A_{i_1}) \cdots \mathbb{P}_{X_{i_m}}(A_{i_m})} > 1. \tag{7}$$

The candidate distribution is still $P_S^1(s) = \mathbb{1}\{s = 1\}$. A further generalization is

$$S(X) = S(X_{i_1}, \ldots, X_{i_m}) = \mathbb{1}\{(X_{i_1}, \ldots, X_{i_m}) \in A\},$$

in which case the criterion becomes

$$\frac{\mathbb{P}_{X_{i_1},\ldots,X_{i_m}}(A)}{\left(\mathbb{P}_{X_{i_1}} \times \cdots \times \mathbb{P}_{X_{i_m}}\right)(A)} > 1.$$

If the model has many components that are presumed to be independent, then this provides a large class of potential statistics that can be used to search for ways to improve the model. The search involves looking for suspicious coincidences. When one is detected, a new part, say $Z$, is added to the model that better models the dependency. In particular, if the statistic is $S(X_i, X_j) = \mathbb{1}\{X_i \in A_i, X_j \in A_j\}$, then $Z = 1$ in the updated model implies $X_i \in A_i$ and $X_j \in A_j$. That is, the presence of the new part implies a particular configuration of its constituent subparts. When $Z = 0$, the constituent parts are independent again. During interpretation, if there is evidence that $X_i \in A_i$ and $X_j \in A_j$, then the interpretation algorithm will have to decide if this happened "by chance" or if it happened "because" $Z = 1$. In the latter case, we say that $X_i$ and $X_j$ are composed into $Z$.

Iteratively detecting suspicious coincidences seems like a nice method for growing the hierarchical structures of a composition system. Imagine some sort of compositional algorithm that interprets an image by identifying parts (or features) and their relationships. These are then composed into larger parts and the algorithm iterates. At the highest or final level the algorithm gives an interpretation of the scene and behaves as if there are no more parts that should be composed into larger objects. Looking for suspicious coincidences among these "high-level" objects would indicate whether or not new compositions are required.

For example, suppose the current system only knows about small lines or edges and builds an interpretation of an image out of these basic parts. Let $A$ be the occurrence of a small vertical line in one part of an image and let $B$ be the occurrence of a small vertical line just above $A$. It seems reasonable that $A$ and $B$ are strongly correlated events because they will both occur whenever there is a larger line that encompasses them both. In particular $\mathrm{Prob}(AB) \gg \mathrm{Prob}(A)\,\mathrm{Prob}(B)$. Over time we can detect this as a suspicious coincidence and create a new feature $C$ which is the composition of $A$ and $B$ – a longer line. Now that the algorithm knows about $C$ it can be used to create simpler interpretations of images. This process will iterate to grow larger and larger features.

In Chapters 3 and 4 we will focus exclusively on model building by detecting suspicious coincidences. The notion of using suspicious coincidences to learn about the world is not new.

### 2.3.1 Finding suspicious coincidences

One of the major roles of the brain is detecting associations. There has been some speculation that this is the primary goal of the cerebral cortex. Barlow has advanced the idea that the particular associations of interest are suspicious coincidences [15, 2, 5, 6]. The general idea is nearly identical to the one here: find suspicious coincidences and learn to anticipate them so that they are no longer suspicious. This brings up the issue of how to find suspicious coincidences.

For any collection of $N$ features, there are $N^2$ possible binary compositions and $2^N$ possible compositions of all orders. Already these numbers are unmanageable; we cannot consider every possible composition. Adding in several different types of composition relationships (to the right of, to the left of, etc.) only makes things worse. Barlow and colleagues immediately recognized this [15] and several mechanisms have been postulated for dealing with it [6], including:

- Only look for coincidences that are likely to occur or be of importance, for example, coincidences whose components have similar spatial locations and scales.
- Only look for coincidences whose components occur frequently. If the components rarely occur, then the combination of them will hardly ever happen.
- Use sparse representations. Densely distributed and/or highly repetitive representations make it difficult to determine when a coincidence is suspicious.

The utility of the first two principles is relatively straightforward, although it is certainly not clear exactly how they should be implemented in an actual algorithm. The notion of sparse coding is somewhat less intuitive. Field [7] cites at least three separate considerations that have led people to suggest sparseness as an important coding principle: improved signal-to-noise ratio, simpler and more reliable detection of statistical dependencies, and higher capacity in associative memory networks. In a distributed representation, sparse coding means that a given item is represented by only a few of the many units. Barlow [4, 6] points out that sparsity is important for detecting suspicious coincidences and also uses physiological evidence to argue that visual cortex uses a sparse code [1, 3]. In a densely distributed representation a given feature of interest will be represented by a complex activity pattern over many (neural) elements. Detecting a suspicious coincidence among high-level features requires keeping track of complex higher-order statistical dependencies. This will be memory intensive, inefficient and error-prone. Fortunately, compositionality does not lend itself to densely distributed representations. We think of each element in a representation as being a feature or a part which can stand alone. We do not necessarily need to know the state of all the other elements in order to interpret a single element. In this sense, a compositional representation is ideal for detecting suspicious coincidences.

Sparsity is also important to ensure that a collection of features is well-separated. If it is not, but includes a lot of similar or identical repetitions, then detecting suspicious coincidences will become inefficient. These similar elements will certainly be highly correlated and the new feature created by composing them together will again be quite similar. Nothing much has been gained by this composition. Also, for every composition in which a certain feature plays a role, all of its similar features will play roles in similar compositions. Both of these things cause the number of compositions detected by suspicious coincidences to explode. Avoiding this type of departure from sparsity is crucial for compositional learning.

In Chapter 3 we use some simple heuristics to keep our representations sparse by making sure that features are well separated within each level of the compositional hierarchy. Practically, this helps to alleviate the problems we just discussed. More theoretically, sparsity is important because it helps to justify the independence assumption in the suspicious coincidence criterion. We discuss this further in the next section.

### 2.3.2 Sparse coding and independence

An implicit assumption for the general framework here is that we can generate a large class of statistics whose distributions are well understood under the current model. This seems like a strong assumption. One of the key ideas behind using coincidence detectors is that we only need the current model to have many independent components. But even this is likely to be too strong of an assumption. Or if it is a valid assumption, then it might be difficult to determine which components should be independent under the current model.

A possible remedy is to relax the strict independence requirement. If $X_i$ and $X_j$ are nearly independent under the current, then $\mathbb{P}_{X_i,X_j}(A_i \times A_j) \gg \mathbb{P}_{X_i}(A_i)\mathbb{P}_{X_j}(A_j)$ would still indicate a deficiency in the model. Interestingly, the simple requirement of sparsity goes a long way toward the ideal of independence and may even be better than a truly independent code (a factorial code) because the latter could be densely distributed, which is problematic for detecting associations (not to mention the conceptual difficulties of creating a compositional factorial code).

That sparsity tends towards independence is easy to see from a well-known heuristic argument. If $X = (X_1, \dots, X_n)$ is a distributed signal (random vector), then its entropy $H(X)$ quantifies (inversely) how much statistical structure exists in the signal. If we recode the signal more sparsely, say $Y = f(X)$, $Y = (Y_1, \dots, Y_n)$, then we preserve the total amount of entropy, $H(Y) = H(X)$, but we reduce the entropy of the individual components, $H(Y_k) \leq H(X_k)$, because a sparse random variable has low entropy. This implies that

$$\sum_{k=1}^{n} H(Y_k) - H(Y) \leq \sum_{k=1}^{n} H(X_k) - H(X).$$

Since each side of this equation is a measure of the amount of statistical dependency that exists among the elements, we have moved toward independence by sparse coding. Furthermore, in a certain sense we can nearly achieve independence with the sparsest possible code (sometimes called grandmother cells), which although far from independent will nevertheless have no more than a bit or so of higher-order redundancy. This simple calculation is detailed in the next section.

#### 2.3.2.1 Grandmother cells are nearly independent

$X$ is a signal (random variable) that takes at most $N$ values, labeled $1, \ldots, N$. For example, if $X = (X_1, \ldots, X_n)$ is a distributed signal and each $X_k \in \{0,1\}$ is binary valued, then we can take $N$ to be $2^n$. If we recode the signal as sparsely as possible, say $Y = f(X)$, $Y = (Y_1, \ldots, Y_N)$, $Y_K = \mathbb{1}\{X = K\}$, so that each possible value of $X$ excites a unique element of $Y$ (grandmother cells), then this new representation has at most $\log_2 e \approx 1.44$ bits of higher-order redundancy. In this sense the $Y_K$'s are nearly independent. They cannot be completely independent because only a single $Y_K$ is active at any given time.

Let $p_K := \text{Prob}(X = K)$. The higher order redundancy in $Y$ is

$$\begin{aligned}
\sum_{K=1}^{N} H(Y_K) - H(Y) &= \sum_{K=1}^{N} H(Y_K) - H(X) \\
&= -\sum_{K=1}^{N} \left[p_K \log p_K + (1-p_K)\log(1-p_K)\right] + \sum_{K=1}^{N} p_K \log p_K \\
&= -\sum_{K=1}^{N} (1-p_K)\log(1-p_K) \\
&\le -\sum_{K=1}^{N} \left(1 - \frac{1}{N}\right) \log\left(1 - \frac{1}{N}\right) = \log\left(1 + \frac{1}{N-1}\right)^{N-1} \le \log e.
\end{aligned}$$

The first inequality is easy to derive from the fact that the uniform distribution maximizes entropy once we note that $\sum_{K=1}^{N}(1-p_K) = N-1$ is fixed.

## 2.4 Other related work

Unsupervised learning of hierarchical representations is not a well understood problem [17], although there is an extensive literature describing various supervised (or semi-supervised) hierarchical learning procedures, especially within the neural network community. There is also a variety of work attempting to create learning procedures that can be roughly mapped onto the hierarchical organization of visual cortex, for example, [9, 18]. For a brief review of general (i.e., not necessarily hierarchical) unsupervised learning, see [10].

Perhaps the work most closely related to the ideas here are the various feature pursuit algorithms for sequential learning of additive random field models [16, 21, 20, 22] and several closely related algorithms for projection pursuit density estimation [8, 11] and independent components analysis [12, 19]. These algorithms build increasingly better approximations for a complicated, high-dimensional distribution by successively modeling (or matching) various lower-dimensional statistics. Typically, however, there is no notion of hierarchy. New features are not built out of previous ones, but instead capture statistics that were not captured by previous features. (Although, the higher-order ICA algorithm in [13] seems like it could be iterated hierarchically.) Also, the features tend not to be categorical, i.e., either present or absent, but instead can be present in varying degrees.

A notable exception is the work by Della Pietra, Della Pietra and Lafferty (1997) [16] which has many similarities to the work here. The features are categorical, newer features are composed entirely out of older features and the parameters of the current model are fixed before adding a new feature. They apply their model to unsupervised learning of English spellings. After incorporating 1500 features, sampling from the model produces "words" that show many of the statistical properties of English words. An important difference is that they do not interpret a new feature as adding a new vertex to the underlying graphical model. This keeps the size of the graph small, but allows the connectivity to become quite dense.

## 2.5 Discussion

Motivated by compositionality, we derived an iterative, unsupervised learning heuristic that can build hierarchical, parts-based, probabilistic graphical models. The general theme is that unexplained dependencies in the data become new parts in the generative model. This framework leads naturally to Barlow's notion of detecting suspicious coincidences and to other unsupervised learning principles like sparse coding. It also embeds Barlow's suspicious coincidence ideas firmly within a probabilistic modeling framework, something that they have been criticized for lacking [14].

In the next two chapters we partially experiment with this algorithm. In particular, we investigate whether or not suspicious coincidences can be used to create hierarchies of increasing selectivity and invariance from natural images. As will be evident from these experiments, many details need to be worked out, not the least of which is computation. Another important issue involves sparsity, which is evidently important, but which we have incorporated in an ad hoc manner. We hope to address some of these issues in future work. We also hope to draw tighter connections between this work and existing unsupervised learning algorithms.

## Bibliography

[1] H. B. Barlow. Single units and sensation: A neuron doctrine for perceptual psychology? *Perception*, 1:371–394, 1972.

[2] H. B. Barlow. Cerebral cortex as model builder. In David Rose and Vernon G. Dobson, editors, *Models of the visual cortex*, pages 37–46. John Wiley & Sons, Chichester, 1985.

[3] H. B. Barlow. The Twelfth Bartlett Memorial Lecture: The role of single neurons in the psychology of perception. *The Quarterly Journal of Experimental Psychology*, 37A:121–145, 1985.

[4] H. B. Barlow. Unsupervised learning. *Neural Computation*, 1:295–311, 1989.

[5] H. B. Barlow. Vision tells you more than "what is where". In Andrei Gorea, editor, *Representation of Vision: Trends and tacit assumptions in vision research*, pages 319–329. Cambridge University Press, Cambridge, 1991.

[6] Horace Barlow. What is the computational goal of the neocortex? In Christof Koch and Joel L. Davis, editors, *Large-Scale Neuronal Theories of the Brain*, pages 1–22. MIT Press, Cambridge, 1994.

[7] D. J. Field. What is the goal of sensory coding? *Neural Computation*, 6:559–601, 1994.

[8] J. Friedman, W. Stuetzle, and A. Schroeder. Projection pursuit density estimation. *Journal of the American Statistical Association*, 79:599–608, 1984.

[9] Karl Friston. Learning and inference in the brain. *Neural Networks*, 16:1325–1352, 2003.

[10] Colin Fyfe. Trends in unsupervised learning. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, volume 21–23, pages 319–326, Bruges, Belgium, April 1999.

[11] Peter Huber. Projection pursuit. *Annals of Statistics*, 13(2):435–475, 1985.

[12] A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999.

[13] Yan Karklin and Michael S. Lewicki. Learning higher-order structures in natural images. *Network: Computation in Neural Systems*, 14:483–499, 2003.

[14] David Mumford. Neuronal architectures for pattern-theoretic problems. In Christof Koch and Joel L. Davis, editors, *Large-Scale Neuronal Theories of the Brain*, pages 125–152. MIT Press, Cambridge, 1994.

[15] C. G. Phillips, S. Zeki, and H. B. Barlow. Localization of function in the cerebral cortex: past, present and future. *Brain*, 107(1):327–361, March 1984.

[16] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, April 1997.

[17] Tomaso Poggio and Steve Smale. The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society*, 50(5):537–544, May 2003.

[18] Maximilian Riesenhuber and Tomaso Poggio. Are cortical models really bound by the "binding problem". *Neuron*, 24:87–93, September 1999.

[19] Max Welling, Richard S. Zemel, and Geoffrey E. Hinton. Probabilistic independent components analysis. *IEEE Transactions on Neural Networks*, 15(4):838–849, July 2004.

[20] Song Chun Zhu and David Mumford. Prior learning and gibbs reaction-diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11):1236–1250, November 1997.

[21] Song Chun Zhu, Ying Nian Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural Computation*, 9(8):1627–1660, November 1997.

[22] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.

# Chapter 3

# Learning selectivity

This chapter with minor differences has been circulated and referenced as a technical report in preparation: M. Harrison and S. Geman. *Compositional feature detectors.* August, 2003.

## 3.1 Introduction

We are interested in statistical algorithms that learn compositional representations of images. Compositional representations are hierarchies of reusable parts. The parts are both more invariant and more selective higher in the hierarchy. Here we use some simple heuristics based on the iterative learning scheme described in Chapter 2 to learn low-level, compositional image features. The resulting hierarchy has increasingly selective parts, but not increasingly invariant parts. Invariance is addressed in Chapter 4. A key component of our learning algorithm is Barlow's principle of detecting and removing "suspicious coincidences" [22].

As mentioned in Chapter 2, computation using compositional systems is not well understood (although the Ph.D. theses of Potter [24] and Huang [14] have made progress in this direction). The main point of this chapter is to see whether or not we can identify the first few compositions that are likely to occur in such a system, even though the details of how they would then be used for image interpretation have not been worked out.

We will focus mainly on binary images of natural scenes (but see Section 3.2.2). We begin with a generative, probabilistic model of such an image that treats small, non-overlapping image patches as a hidden mixture model (Section 3.2). The model itself is actually learned from the image data (Section 3.2.1), which is interesting but probably not crucial for our later results. What is more important is that the parameters of the model are then estimated from the data using an EM-like procedure (Section 3.3). Once the model has been trained, we can use the model to probabilistically interpret a new image. In particular, for each image patch we can compute the posterior probability of all of the hidden states (Section 3.4). This lets us detect suspicious coincidences among collections of neighboring hidden states (Section 3.5). We use compositionality and (translation) invariance to facilitate the detection of suspicious coincidences (Section 3.5.5). Sparse coding is crucial for preventing an explosion of suspicious coincidences (Section 3.5.3).

## 3.2 The generative model

The model generates binary $3m\times 3n$-pixel images $I$ by independently generating each of the $mn$ non-overlapping $3\times 3$-pixel patches from the same distribution. We will label these $mn$ patches $Y_{k\ell}$, $1\leq k\leq m, 1\leq \ell\leq n$, preserving the topology of the image and denoting the individual binary pixels as

$$Y_{k\ell}(i,j) := I(3(k-1)+i, 3(\ell-1)+j) \in \{0,1\}, \quad 1\leq i\leq 3,\ 1\leq j\leq 3.$$

We use 0 to denote black and 1 to denote white. The independence of the $Y_{k\ell}$ implies that

$$P(I) = \prod_{k=1}^{m}\prod_{\ell=1}^{n} P(Y_{k\ell}).$$

A given binary 3×3 patch is generated from a hidden mixture model: first a representative patch (or no patch) is chosen from some small collection of $S+1$ hidden states and then (flip) noise is added. Using $X_{k\ell}\in\{0,1,\ldots,S\}$ to denote the hidden state associated with the $(k,\ell)$-th image patch $Y_{k\ell}$ gives

$$P(I) = \prod_{k=1}^{m}\prod_{\ell=1}^{n}\left[\sum_{s=0}^{S} P(X_{k\ell}=s)P(Y_{k\ell}|X_{k\ell}=s)\right].$$

Since our model does not distinguish between different image locations, we can complete the description of the model by specifying the distributions $P(X)$ and $P(Y|X)$ for a generic patch $Y$ and hidden state $X$.

Our model has 11 hidden states with probabilities $P(X=s) := p_s$, $0\leq s\leq 10$, which sum to 1. The states $1,\ldots,10$ correspond to 10 different representative $3\times 3$ binary patches denoted $B_1,\ldots,B_{10}$, where $B_s := \{B_s(i,j) : 1\leq i\leq 3,\ 1\leq j\leq 3\} \in \{0,1\}^{3\times 3}$. The representative patches that we used are shown in Figure 3.4. They were selected from data using heuristics based on suspicious coincidences and sparse coding (see Section 3.2.1). State 0 corresponds to no representative patch.

To generate an observation of a patch $Y$, first, one of the 11 states are chosen with the corresponding probabilities. If state 0 is chosen, then each of the 9 pixels in the $3\times 3$ observation patch is independent and identically distributed (i.i.d.) with probability of white $\beta$ and probability of black $1-\beta$. So

$$P(Y|X=0) := \prod_{i=1}^{3}\prod_{j=1}^{3}\beta^{\mathbb{1}\{Y(i,j)=1\}}(1-\beta)^{\mathbb{1}\{Y(i,j)=0\}} = \left[\beta^{\|Y\|}(1-\beta)^{1-\|Y\|}\right]^9,$$

where $\mathbb{1}\{A\}$ is the indicator of the event $A$ and

$$\|Y\| := \frac{1}{9}\sum_{i=1}^{3}\sum_{j=1}^{3}\big|Y(i,j)\big|.$$

If one of the states $1,\ldots,10$ is chosen, then the corresponding representative patch $B_s$ is selected and the 9 pixels in the observation patch are generated by independently flipping the pixels of the representative patch with some small probability $\alpha$. This gives

$$\begin{aligned} P(Y|X=s) &:= \prod_{i=1}^{3}\prod_{j=1}^{3}\alpha^{\mathbb{1}\{Y(i,j)\neq B_s(i,j)\}}(1-\alpha)^{\mathbb{1}\{Y(i,j)=B_s(i,j)\}} \\ &= \left[\alpha^{\|Y-B_s\|}(1-\alpha)^{1-\|Y-B_s\|}\right]^9 \end{aligned}$$

for $s = 1, \ldots, 10$, where

$$\|Y - B_s\| := \frac{1}{9}\sum_{i=1}^{3}\sum_{j=1}^{3}\left|Y(i,j) - B_s(i,j)\right|.$$

Except for the specific parameter values (see Section 3.3), this is a complete probabilistic description of the generative model for images.

### 3.2.1 Learning the model

The 10 representative binary 3×3-patches used in our generative model and shown in Figure 3.4 were discovered from image data using heuristics based on suspicious coincidences and sparse coding. Under the assumption that every pixel in an image is i.i.d. with probability 1/2 of either black or white, then each of the $2^9 = 512$ possible patches is equally likely. We can easily collect the frequency of occurrence of each of these patches in a collection of images. Patches that occur more frequently than 1/512 are suspicious coincidences.

Using every 3×3 patch from the first 100 images from the image data described in Section 3.3, we computed the frequency of each of the 512 patches and found 44 suspicious coincidences, that is, patches with frequencies greater than 1/512. The histogram of frequencies is shown in Figure 3.2 and the suspicious coincidences are shown in Figure 3.3.

The list of suspicious patches includes many of the features that we would expect, such as the constant patches and the horizontal and vertical edges. Unfortunately, as indicated in Figure 3.3, the collection of suspicious patches is highly redundant. If a certain patch is suspicious, then many of its slight variations (e.g., single pixel flips) will also be suspicious. We cannot keep the entire list of suspicious patches and still maintain a sparse representation of image patches. We need to prune the list.

Ideally, we would like to select a single representative patch for each "feature" and let some sort of noise model take care of the rest. There are many ways to proceed, but the first and simplest thing worked, so that is all that we tried. Each patch has 9 neighboring patches created by flipping the color of a single pixel. For each of the 512 patches, we selected those that were both a suspicious coincidence (frequency > 1/512) and whose frequency was greater than the maximum of its 9 neighbors' frequencies. There were 10 patches that satisfied these criteria. They are shown in Figure 3.4 and were used as the representative patches in the generative model. They are the two constant patches and each of the eight possible horizontal and vertical edges.

### 3.2.2 Possible extensions of the model

All that we need is a generative model with some hidden states that lets us compute the posterior probabilities of the hidden states given an image (see Section 3.4). These hidden states are conceptualized as features. The collection of them should be small and sparse to facilitate looking for suspicious coincidences among their joint probabilities. With this in mind, several extensions of the model are apparent.

Extending the model to patch sizes other than $3\times3$ is trivial. Some care may need to be taken in selecting the representative patches for the hidden states. The method described in Section 3.2.1 found 48 different $4\times4$ patches, composed of the two constant patches, all the horizontal and vertical edges/lines, some diagonal edges/lines and some center-surround patches. These are shown in Figure 3.5. Increasing the patch size much more than this will likely lead to an explosion of representative patches and the sparsening procedure will need to be modified.

Extending the model to images with a few more intensity levels is also straightforward. The noise model will need to be modified slightly. The local maxima sparsening procedure still works on ternary $3\times3$-patches, finding 40 representative patches composed of the three constant patches, vertical and horizontal lines and edges and a few diagonal edges. These are shown in Figure 3.6. Again, increasing the number of intensity levels quickly leads to an explosion of representative patches using this simple sparsening procedure.

Extending the model to gray-scale or color images will require several major changes. It seems likely that the low-frequency component or the mean intensity level of an image patch should be modeled separately from the high-frequency components. The high-frequency components could be treated like our representative patches with a more sophisticated noise model. The low-frequency component might need to be quantized into a few representative intensity levels, again with a more sophisticated noise model.

The projection pursuit [15, 9] methods of finding collections of filters from natural images seem like an attractive option for discovering the representative hidden states in the model. These methods are exemplified by sparse components analysis [21] and independent components analysis [3, 16]. By locally searching for filters with the highest possible kurtosis (or something like it), these methods are in some ways simultaneously looking for suspicious coincidences and sparsity. Closely related work includes products of experts (with sparse experts) [12, 28] and additive random fields / maximum entropy models [4, 23, 29, 30, 31]. Other related work that could be adapted to the situation here includes [11, 26].

There would still be several striking drawbacks of such a hidden mixture model for image patches. The rigidity of the placement of the non-overlapping patches and the complete lack of any invariance will quickly overwhelm any learning algorithm because a given feature can occur in so many different ways (although, see [25]). The model cannot account for the great variety of instantiations of a feature. It will have to learn them each separately. Unfortunately, modifying the model to accommodate these deficiencies prevents us from easily learning the parameters of the model (see Section 3.3) and computing posterior distributions (see Section 3.4), both of which seem crucial for discovering suspicious coincidences.

## 3.3 Estimating the model parameters

The generative model has 12 parameters: $p_1, \ldots, p_{10}, \alpha, \beta$. ($p_0$ is fixed by the probability constraint.) These are easily learned from image data using the expectation-maximization (EM) algorithm (see [5] for details). Given a collection of $T$ binary $3 \times 3$ image patches $Y^1, \ldots, Y^T$, where $Y^t := \{Y^t(i,j) : 1 \le i \le 3,\ 1 \le j \le 3\} \in \{0,1\}^{3\times 3}$, the EM update equations are

$$p_s^{\text{new}} = \frac{1}{T}\sum_{t=1}^{T} w_{ts}, \quad s = 0, \ldots, 10,$$

$$\beta^{\text{new}} = \frac{1}{T}\sum_{t=1}^{T} \frac{w_{t0}\|Y^t\|}{p_0^{\text{new}}} \quad \text{and} \quad \alpha^{\text{new}} = \frac{1}{T}\sum_{t=1}^{T} \frac{\sum_{s=1}^{10} w_{ts}\|Y^t - B_s\|}{\sum_{s=1}^{10} p_s^{\text{new}}}.$$

The weights $w_{ts}$ are just the posterior probabilities of the states given the image patch (2) and can be computed from the noise model $P(Y|X)$ using Bayes' rule and the old parameters.

$$w_{ts} = P(X = s | Y = Y^t) = \frac{p_s^{\text{old}} P(Y = Y^t | X = s; \beta^{\text{old}}, \alpha^{\text{old}})}{\sum_{u=0}^{10} p_u^{\text{old}} P(Y = Y^t | X = u; \beta^{\text{old}}, \alpha^{\text{old}})} \tag{1}$$

for $t = 1, \ldots, T, s = 0, \ldots, 10$. All of these computations are relatively straightforward. Derivations of the EM update equations can be found at the end of this section.

The model was trained on all non-overlapping $3 \times 3$ patches using the first 1000 natural images from a large collection courtesy of Hans van Hateren and described in [27]. The gray-scale images were first reduced in size to $126 \times 192$ pixels (the JPEG thumbnails of van Hateren) and then converted to binary by thresholding each image at its median intensity value. Sample images with enlargements are shown in Figure 3.1.

To avoid having to store all the images in memory simultaneously (relevant for much larger data sets), we ran a single iteration of the EM update equations on each image (2688 images patches per image). This gives a reasonable estimate of the parameters but it weights the final image. To get a better estimate we then repeated this process using a running average update, for example,

$$\beta^{\text{new}} = \frac{1}{t}\beta^{\text{new}} + \frac{t-1}{t}\beta^{\text{old}},$$

where the $\beta^{\text{new}}$ on the right comes from the original EM update equation and $t$ is the current image number. The initial parameters for the whole process were $\alpha = .1$, $\beta = .5$ and $p_1, \ldots, p_{10}$ set to the empirical probabilities of their respective patches (see Section 3.2.1). The fitted model parameters are shown in the next table.

| $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ | $p_7$ | $p_8$ |
|---|---|---|---|---|---|---|---|
| 0.3572 | 0.0127 | 0.0101 | 0.0100 | 0.0092 | 0.0114 | 0.0107 | 0.0130 |
| $p_9$ | $p_{10}$ | $p_0$ | $\alpha$ | $\beta$ | | | |
| 0.0101 | 0.3423 | 0.2133 | 0.0403 | 0.5046 | | | |

The relative entropy between the truth (empirical probabilities of all possible 512 patches) and our fitted model for image patches is 0.1935 bits/patch (or 0.0215 bits/ pixel). This is a simple way of quantifying the fit of the model and can be interpreted as the penalty that we would expect to pay using our model to compress image patches instead of the ideal model. The entropy of the empirical distribution is 4.6505 bits/patch (or 0.5167 bits/pixel), so this penalty only increases code lengths by about 4%. For comparison, the entropy of the model is 5.5033 bits/patch (or 0.6115 bits/pixel).

**Proof of the EM update equations.** For estimating the parameters of a hidden mixture model, the EM equations for the weights $w_{ks}$ (1) are the posterior probabilities of the hidden states given the observations and the previous parameter values. The new (next step) estimates of the mixing probabilities $p_s^{\text{new}}$ are the average of the posterior probabilities (the weights) [5]. The parameters of the noise model $(\alpha^{\text{new}}, \beta^{\text{new}})$ are maximizers of

$$\begin{aligned}\sum_{t=1}^{T}\sum_{s=0}^{10} & w_{ts} \log P(Y^t|\text{state } s; \alpha, \beta) \\ &= \sum_{t=1}^{T} w_{t0} \log \left[\beta^{\|Y^t\|}(1-\beta)^{1-\|Y^t\|}\right]^9 + \sum_{t=1}^{T}\sum_{s=1}^{10} w_{ts} \log \left[\alpha^{\|Y^t-B^s\|}(1-\alpha)^{1-\|Y^t-B^s\|}\right]^9,\end{aligned}$$

over $0 \le \alpha, \beta \le 1$, where the substitutions came from the noise model $P(Y|X)$ in Section 3.2. Differentiating this expression gives

$$\frac{\partial}{\partial \beta}\Big[\cdots\Big] = \frac{9}{\beta}\sum_{t=1}^{T} w_{t0}\|Y^t\| - \frac{9}{1-\beta}\sum_{t=1}^{T} w_{t0}(1-\|Y^t\|),$$

$$\frac{\partial}{\partial \alpha}\Big[\cdots\Big] = \frac{9}{\alpha}\sum_{t=1}^{T}\sum_{s=1}^{10} w_{ts}\|Y^t-B^s\| - \frac{9}{1-\alpha}\sum_{t=1}^{T}\sum_{s=1}^{10} w_{ts}(1-\|Y^t-B^s\|).$$

Solving for zeros to find the maximizers gives the new parameter values

$$\beta^{\text{new}} = \frac{\sum_{t=1}^{T} w_{t0}\|Y^t\|}{\sum_{t=1}^{T} w_{t0}} \quad \text{and} \quad \alpha^{\text{new}} = \frac{\sum_{t=1}^{T}\sum_{s=1}^{10} w_{ts}\|Y^t-B^s\|}{\sum_{t=1}^{T}\sum_{s=1}^{10} w_{ts}},$$

which can be rewritten as in the text. □

## 3.4 Computing the posterior

Once we have specified all the parameters of the model, we can compute the posterior probability of each of the 11 hidden states given an image patch $Y$ using Bayes' rule and the noise model $P(Y|X)$:

$$P(X=s|Y) = \frac{p_s P(Y|X=s)}{\sum_{u=0}^{10} p_u P(Y|X=u)}, \quad s = 0, \ldots, 10. \tag{2}$$

This is the same computation for the weights used by EM when training the model (1).

For an image $I$ we can independently compute the 11 posterior probabilities given each of the $mn$ non-overlapping patches $Y_{k\ell}$. This is an exact computation because of the independence assumption in the model. Preserving the topology of the image grid allows us to arrange the posterior probabilities into a new $m \times n$ grid with 11 values at each point, or equivalently, 11 different $m \times n$ grids (perhaps visualized as stacked on top of one another).

We will denote the posterior probability of state $s$ from the $(k, \ell)$-th patch as

$$Q_{sk\ell}(I) := P(X_{k\ell} = s|I) = P(X_{k\ell} = s|Y_{k\ell}),$$

$k = 1, \ldots, m$, $\ell = 1, \ldots, n$, $s = 0, \ldots, 10$. This is just (2) with extra notation to indicate where the patch is located in the image. Since $Q_{sk\ell}$ is a probability, it has a value between 0 and 1. For $s = 1, \ldots, 10$, $Q_{sk\ell}$ can be viewed as a non-linear filter based on representative patch $B_s$ applied to the $(k, \ell)$-th image patch. We do not need to explicitly compute $Q_{0k\ell}$ because it is fixed by the probability constraint $\sum_{s=0}^{10} Q_{sk\ell} = 1$.

In summary, given an image we compute the values of 10 different non-linear filters centered at each non-overlapping $3 \times 3$ image patch location. These filter values $Q_{sk\ell}$ are the posterior probabilities of the states given the image patches using our generative model with the fitted parameters. We can use $Q$ to look for certain deficiencies in the model, namely suspicious coincidences.

## 3.5 Detecting suspicious coincidences

Our model describes a probability distribution $P$ on images as well as hidden states $X$. The world also has a true probability distribution $\mathbb{P}$ for images but not for hidden states, since these are an invention of the model and do not necessarily correspond to reality. We can, however, combine these distributions to create a "true" distribution for hidden states

$$\mathbb{P}(X_{k\ell} = s) := \mathbb{E}\left[P(X_{k\ell} = s|I)\right],$$

where $\mathbb{E}$ denotes expectation over images $I$ with distribution $\mathbb{P}$. We also have the identity

$$P(X_{k\ell} = s) = E\left[P(X_{k\ell} = s|I)\right],$$

where $E$ denotes expectation over images $I$ with distribution $P$. Using $\mathbb{P}$ to talk about the hidden states of the model is an abuse of notation, but it nicely captures the intuition. If $\mathbb{P}$ and $P$ give the same distribution on images, that is, if $\mathbb{E}$ and $E$ are the same expectation, then $\mathbb{P}$ and $P$ give the same distribution on hidden states. By computing the distribution on hidden states in both cases, we can evaluate our model for images.

Of course, $\mathbb{P}$ is unknown, but we can approximate it by using the empirical distribution $\hat{\mathbb{P}}$ of a large collection of images $I^1, \ldots, I^T$. For example

$$\hat{\mathbb{P}}(X_{k\ell} = s) := \hat{\mathbb{E}}\left[P(X_{k\ell} = s|I)\right] := \frac{1}{T}\sum_{t=1}^{T} P(X_{k\ell} = s|I^t) := \frac{1}{T}\sum_{t=1}^{T} Q_{sk\ell}(I^t).$$

Each of these expressions is just a different way of writing the same thing. The $\hat{\mathbb{P}}$ notation on the left is useful for thinking about suspicious coincidences and other probabilistic considerations. The $Q$ notation on the right shows exactly what would be computed by the algorithm.

Our model does not distinguish among patch or hidden state locations so it makes sense to talk about $P(X)$ for a generic hidden state $X$. If we want to condition on an image, however, $P(X|I)$ is ambiguous and this creates problems for defining $\mathbb{P}(X)$ in the same way that $\mathbb{P}(X_{k\ell})$ was defined. One way to make sense of this is to let $X$ be a patch chosen randomly and uniformly from the possible patch locations in the image, which is $P(X = s|Y = Y_{k\ell})$ averaged over each patch $Y_{k\ell}$ in the image $I$.

$$P(X = s|I) := \frac{1}{mn}\sum_{k=1}^{n}\sum_{\ell=1}^{m} P(X = s|Y = Y_{k\ell}).$$

This now lets us define

$$\mathbb{P}(X = s) := \mathbb{E}\left[P(X = s|I)\right],$$

which is approximated by the empirical distribution

$$\hat{\mathbb{P}}(X = s) := \hat{\mathbb{E}}\left[P(X = s|I)\right] := \frac{1}{T}\sum_{t=1}^{T} P(X = s|I^t) := \frac{1}{T}\sum_{t=1}^{T}\frac{1}{mn}\sum_{k=1}^{m}\sum_{\ell=1}^{n} Q_{sk\ell}(I^t). \tag{3}$$

The main reasons for dealing with a generic $X$ is to make $\hat{\mathbb{P}}$ a better approximation of $\mathbb{P}$ for a given number of images because of the increased amount of averaging and to reduce the number of statistics that we have to measure. The drawback is that we lose the ability to detect statistics for specific locations in the image plane. If the true distribution for images is translation invariant, then we will have lost nothing. In this way our algorithm makes explicit use of an invariance bias.

As previously mentioned, one way to compare our distribution for images $P(I)$ to the true distribution $\mathbb{P}(I)$ is to verify that $P(X = s) = \mathbb{P}(X = s)$ for each $s$. We cannot do this, but we can verify that

$$p_s := P(X = s) \approx \hat{\mathbb{P}}(X = s) := \frac{1}{T}\sum_{t=1}^{T}\frac{1}{mn}\sum_{k=1}^{m}\sum_{\ell=1}^{n} Q_{sk\ell}(I^t), \tag{4}$$

because both sides are either known or easily computable from a collection of images. If this approximation is clearly violated, then there is something wrong with our model. In fact, this particular approximation will be quite good because the EM learning algorithm that we used is designed to enforce this constraint. We will need a different statistic to identify the problems with our model.

One of the striking deficiencies in our model is the assumed independence among patches. Neighboring patches in an image will likely have many of the same statistical properties. For example, the state corresponding to the all black patch is much more likely (than independence would predict) to have all black neighbors because of the presence of large contiguous regions in images. Similarly, the state corresponding to a horizontal edge is much more likely to have left / right neighbors which are also horizontal edges because images have long continuous edges. We can detect these discrepancies using $Q$. This amounts to searching for suspicious coincidences.

The model asserts that $P(X_{k\ell} = s, X_{k'\ell'} = s') = P(X_{k\ell} = s)P(X_{k'\ell'} = s')$ as long as $(k,\ell) \neq (k',\ell')$. Does $\mathbb{P}$ have the same independence?

$$\begin{aligned}\mathbb{P}(X_{k\ell} = s, X_{k'\ell'} = s') &:= \mathbb{E}\left[P(X_{k\ell} = s, X_{k'\ell'} = s'|I)\right]\\ &= \mathbb{E}\left[P(X_{k\ell} = s|I)P(X_{k'\ell'} = s'|I)\right] \overset{?}{=} \mathbb{P}(X_{k\ell} = s)\mathbb{P}(X_{k'\ell'} = s').\end{aligned}$$

We can test this using $\hat{\mathbb{P}}$ and $Q$, but we would first like to incorporate the location invariance of the model and the presumed translation invariance of $\mathbb{P}$ by using a generic hidden states $X$ and $X'$ instead of hidden states $X_{k\ell}$ and $X_{k'\ell'}$ with specific locations in the image. The relative coordinates of $X$ and $X'$ will still need to be preserved; the joint statistics of neighboring patches might be quite different from those of distant patches. We will use the notation $(X,X')_{k_0\ell_0}$ to denote a generic pair of hidden states $X$ and $X'$ with $X'$ offset $(k_0,\ell_0) \neq (0,0)$ from $X$. The location of this pair is chosen randomly and uniformly from all possible locations so that both hidden states fit in the image. For example, $(X,X')_{10}$ means that $X$ is uniformly selected from $\{X_{k\ell} : 1 \le k \le m-1, 1 \le \ell \le m\}$ and that $X'$ is the immediate right neighbor of $X$. We thus have

$$\begin{aligned}&P((X,X')_{k_0\ell_0} = (s,s')|I)\\ &\quad := \frac{1}{(m-k_0)(n-\ell_0)} \sum_{k=1}^{m-k_0} \sum_{\ell=1}^{n-\ell_0} P(X_{k\ell} = s, X_{(k+k_0)(\ell+\ell_0)} = s'|I)\\ &\quad = \frac{1}{(m-k_0)(n-\ell_0)} \sum_{k=1}^{m-k_0} \sum_{\ell=1}^{n-\ell_0} P(X_{k\ell} = s|Y_{k\ell})P(X_{(k+k_0)(\ell+\ell_0)} = s'|Y_{(k+k_0)(\ell+\ell_0)}).\end{aligned}$$

We can now define

$$\mathbb{P}((X,X')_{k_0\ell_0} = (s,s')) := \mathbb{E}\left[P((X,X')_{k_0\ell_0} = (s,s')|I)\right],$$

which is approximated by

$$\begin{aligned}\hat{\mathbb{P}}((X,X')_{k_0\ell_0} = (s,s')) &:= \hat{\mathbb{E}}\left[P((X,X')_{k_0\ell_0} = (s,s')|I)\right]\\ &:= \frac{1}{T}\sum_{t=1}^{T} P((X,X')_{k_0\ell_0} = (s,s')|I^t)\\ &:= \frac{1}{T}\sum_{t=1}^{T} \frac{1}{(m-k_0)(n-\ell_0)} \sum_{k=1}^{m-k_0} \sum_{\ell=1}^{n-\ell_0} Q_{sk\ell}(I^t)Q_{s'(k+k_0)(\ell+\ell_0)}(I^t). \qquad (5)\end{aligned}$$

Does

$$\mathbb{P}((X,X')_{k_0\ell_0} = (s,s')) \overset{?}{=} \mathbb{P}(X = s)\mathbb{P}(X' = s')$$

as the model predicts, or is there some additional dependence? We can test this by verifying that

$$\hat{\mathbb{P}}((X,X')_{k_0\ell_0} = (s,s')) \approx \hat{\mathbb{P}}(X = s)\hat{\mathbb{P}}(X' = s').$$

Both sides are easily computable using (3) and (5). We are specifically interested in situations where

$$\hat{\mathbb{P}}((X,X')_{k_0\ell_0} = (s,s')) \gg \hat{\mathbb{P}}(X=s)\hat{\mathbb{P}}(X'=s'), \tag{6}$$

a suspicious coincidence.

### 3.5.1 Second-order suspicious coincidences

The EM algorithm takes care of first-order suspicious coincidences (departures from the model), in the sense that it makes the (first-order marginal) probabilities of the hidden states match their empirical estimates from $Q$. That is, (4) is a valid approximation.

We can now look for second-order suspicious coincidences – when the joint probability of two states in different locations is higher than predicted by independence – by finding state pairs $(s,s')$ and offsets $(k_0,\ell_0)$ where (6) holds.

An image has $(m-1)(n-1)-1$ different allowable offsets $(k_0,\ell_0)$ (we do not need to consider negative offsets because these are included by switching $s$ and $s'$) and our model has $S+1$ different hidden states. This gives about $mnS^2$ different binary associations for consideration in (6). Since $mn$ can be quite large, this is a significant memory burden. We expect the independence assumption to be most violated by neighboring patches, so we can restrict ourselves to the cases where the offset corresponds to neighboring locations in an image. For example, we can only consider the 2 offsets horizontal $(1,0)$ and vertical $(0,1)$. This gives about $2S^2$ different associations to remember, which is much more manageable.

We also restrict the states to $1,\dots,S=10$, and do not consider state 0. States $1,\dots,S$ represent the presence of a specific feature in the image patch, like a horizontal edge. State 0 represents the absence of any features. Ignoring state 0 maintains some consistency between the setup in this chapter and a more general framework that we are developing in which there will be no state 0. It is also more in the spirit of compositionality, where multiple *present* features are composed into a new high-level feature. In all then, we will only consider $2S^2 = 200$ different possible suspicious coincidences using (6).

For a collection of images $I^1,\dots,I^T$, and each pair of states $(s,s')$ we compute $\hat{\mathbb{P}}(X=s)$, $\hat{\mathbb{P}}(X'=s')$, $\hat{\mathbb{P}}((X,X')_{10}=(s,s'))$ and $\hat{\mathbb{P}}((X,X')_{01}=(s,s'))$ using $Q$ as indicated in (3) and (5). A suspicious coincidence is registered when

$$\hat{\mathbb{P}}((X,X')_{10}=(s,s')) > \hat{\mathbb{P}}(X=s)\hat{\mathbb{P}}(X'=s')$$

and similarly for offset $(0,1)$.

### 3.5.2 A minimum description length (MDL) criterion

This method identifies 89 different second-order suspicious coincidences. We use a minimum description length (MDL) criterion to rank them. Each state pair $(s,s')$ and each offset $(k_0,\ell_0)$ (we only consider two of them) is assigned a measure of suspiciousness

$$r_{k_0\ell_0}(s,s') := \hat{\mathbb{P}}((X,X')_{k_0\ell_0}=(s,s'))\log\frac{\hat{\mathbb{P}}((X,X')_{k_0\ell_0}=(s,s'))}{\hat{\mathbb{P}}(X=s)\hat{\mathbb{P}}(X'=s')}.$$

This is a reasonable measure because $r_{k_0\ell_0}(s, s') > 0$ exactly when we have a suspicious coincidence. It increases as the joint probability becomes proportionally larger than the product of the probabilities, giving a higher rank to larger departures from independence. It also increases as the joint probability increases, giving a higher rank to feature combinations that occur more frequently. From an information theory point of view, we can loosely interpret $r$ as the number of bits (using $\log_2$) that we would save on average by coding with a probability distribution that accounted for this suspicious coincidence as compared with one that did not (our model). MDL-like criteria are quite common for learning and evaluating models [2].

Each of the 89 second-order suspicious coincidences that are discovered are shown in Figure 3.7. They are ranked from highest to lowest using $r$. The constant patches are first, followed by the extended edges, then some other edge or line elements, then some high frequency elements and then some corner or junction configurations. Many of these combinations intuitively make sense when thinking about how the independence assumption might be violated in natural images.

Ideally, we would use this information to create a new model, probably by adding another hierarchical layer of hidden states, that incorporates these dependencies. From the compositionality perspective, these dependencies arise because the low-level features occasionally occur as parts of a higher-level feature. The new layer capturing these suspicious coincidences would thus represent these higher-level features. Although we do not actually build this new model, we can still think about the new higher-level representation. What are the suspicious coincidences among elements of this new level? Is the representation appropriate for detecting suspicious coincidences? Can the process be iterated and where does it break down? These are some of the questions that we try to address in the remainder of this chapter.

### 3.5.3 Using sparse coding

Figure 3.7 has many similar elements. If a given local region of an image "excites" one of these elements, that is, gives it a high posterior probability, then the same region is likely to "excite" another, similar element. The representation is not sparse. As mentioned earlier this will cause a combinatorial explosion of suspicious coincidences in higher levels. We need to prune the representation to make it sparser.

The local maxima procedure described in Section 3.2.1 for finding the representative 3×3 patches also works here. We only keep those elements in Figure 3.7 whose MDL measure of suspiciousness, $r$, is higher than any of its neighboring elements. We use an ad hoc method for determining neighbors. Compositions of the same shape (we only have two shapes: horizontal neighbors or vertical neighbors) are neighbors if their per pixel Hamming distance is less than 1/3. That is,

$$\frac{\|B_{s_1} - B_{s_2}\| + \|B_{s_1'} - B_{s_2'}\|}{2} < \frac{1}{3},$$

where $(s_1, s_1')$ are the state pairs for one element and $(s_2, s_2')$ are the states for another. 1/3 was chosen because it is the minimum per pixel Hamming distance between any two different representative patches $B_s$. For compositions of a different shape we do the same thing where they overlap (at states $s_1$ and $s_2$) and add a penalty of 1/2 for the patches that do not overlap. The criterion is

$$\frac{\|B_{s_1} - B_{s_2}\| + 1/2}{2} < \frac{1}{3}.$$

1/2 was chosen because it is the mean per pixel Hamming distance between any two different representative patches $B_s$. The patches are always aligned by the upper left corner for comparison.

The 16 elements of the sparsened representation are shown in Figure 3.8. Each of the 4 constant intensity patches and the 8 horizontal and vertical edges remain. The remaining 4 patches are 2-pixel width lines. Presumably, these 16 would be the "representative compositions" in the next layer of the model and they would capture some of the dependencies detected by the suspicious coincidences. These are the only 2-patch compositions that are considered in later iterations of this procedure.

The distance function that we use for determining neighboring patches was the first one that we tried. Later investigations showed that the behavior of our algorithm is incredibly sensitive to the parameters 1/3 and 1/2. Changing these constants even slightly can cause the sparsening procedure to drastically over or under prune. A more principled and hopefully more robust approach to pruning would use statistical information to determine neighbors. Two similar compositions in the same image location will be highly correlated and could thus be identified as neighbors. The local maxima procedure or some other clustering algorithm could then be used on this statistical distance. We leave this idea for future implementations. The notion of using statistical dependencies to measure redundancies and then remove them to obtain a sparser representation is nearly as old as the notion of sparseness itself. See Földiák [8] for an early computational example and Hyvärinen et al. [17, 18] for more recent developments.

### 3.5.4 Higher order suspicious coincidences

Up to this point we have only discussed binary associations, but we can also consider higher order suspicious coincidences

$$P(A_1, \ldots, A_N) \gg P(A_1) \cdots P(A_N).$$

In principle, nothing really changes except the notation becomes burdensome. We will use $(X^1, X^2, \ldots, X^N)$ to represent an $N$th-order generic composition of patches. To denote the relative coordinates of the $X^\nu$, we subscript the collection with a list of $N-1$ ordered pairs, each denoting the offset from the position of $X^1$,

$$(X^1, X^2, \ldots, X^N)_{k^2\ell^2, k^3\ell^3, \ldots, k^N\ell^N},$$

so that $X^\nu$ is offset $(k^\nu, \ell^\nu)$ from $X^1$. This is consistent with our previous notation $(X, X')_{k_0\ell_0}$, but now we would prefer to write $(X^1, X^2)_{k^2\ell^2}$. We can define $P(\cdot|I)$ and then $\mathbb{P}(\cdot)$ as before, approximating the latter with

$$\hat{\mathbb{P}}((X^1, \ldots, X^N)_{k^2\ell^2,\ldots,k^N\ell^N} = (s^1, \ldots, s^N))$$
$$:= \frac{1}{T} \sum_{t=1}^{T} \frac{1}{(m-k_0)(n-\ell_0)} \sum_{k=1}^{m-k_0} \sum_{\ell=1}^{n-\ell_0} \prod_{\nu=1}^{N} Q_{s^\nu(k+k^\nu)(\ell+\ell^\nu)}(I^t), \tag{7}$$

where we take $(k^1, \ell^1) := (0, 0)$ and we define

$$k_0 := \max_{\nu \le N} k^\nu, \quad \ell_0 := \max_{\nu \le N} \ell^\nu.$$

As long as $(k^\nu, \ell^\nu) \ne (k^\mu, \ell^\mu)$ for $1 \le \nu \ne \mu \le N$, the model predicts that this distribution should (approximately) factor. An $N$th-order suspicious coincidence is detected when

$$\hat{\mathbb{P}}((X^1, \ldots, X^N)_{k^2\ell^2,\ldots,k^N\ell^N} = (s^1, \ldots, s^N)) \gg \prod_{\nu=1}^{N} \hat{\mathbb{P}}(X = s^\nu). \tag{8}$$

Unfortunately, this approach does not extend very far. The number of $N$th-ordered pairs increases exponentially, not to mention the steadily increasing number of possible spatial arrangements. Even if we restrict ourselves to connected components, there are $6S^3 = 6\,000$ combinations for 3rd-order associations, $19S^4 = 190\,000$ for 4th-order and $55S^5 = 5\,500\,000$ for 5th-order, which is approaching the limits of feasible computation. We partially surmount this problem by making explicit use of the compositionality bias.

### 3.5.5 Using compositionality

Compositionality asserts that the structure found in natural images can be built up hierarchically with reusable parts. If the features that we just detected with suspicious coincidences are, say, level 2 in this hierarchy, then we should be able to build level 3 features by looking for suspicious coincidences among level 2 features. These level 3 features will be high-order suspicious coincidences back in the original data, but they will be low-order suspicious coincidences in the level 2 data. Iterating this process a few times will allow us to detect very high-order structure in the original data, much higher order than would ever be feasible using the exhaustive search techniques of the previous section. If natural images are truly compositional, then we may not be sacrificing much for this incredible gain in efficiency. The high-level features that we find will not only be suspicious coincidences in the pixel statistics, but also suspicious coincidences among reusable parts at many hierarchical levels.

In Section 3.5.1 we described how to find second-order suspicious coincidences. After ranking (Section 3.5.2) and pruning (Section 3.5.3), we were left with a sparse collection of 16 features shown in Figure 3.8. These, along with the 10 original representative patches shown in Figure 3.4 (Section 3.2.1), are the reusable parts that will be composed into features in the next level of the hierarchy.

We will use notation similar to that in Section 3.5.4, adding parentheses to indicate the hierarchical relationship among the components. For example,

$$(X^1, (X^2, X^3)_{k^3\ell^3})_{k^2\ell^2}$$

denotes three generic hidden states with specific relative offsets. $X^1$ can be in any allowable location. Once the position of $X^1$ is fixed, the $(X^2, X^3)_{k^3\ell^3}$ composition unit is offset $(k^2, \ell^2)$ from the position of $X^1$. The position of this unit is its first member's position, in this case, $X^2$. So $X^2$ is offset $(k^2, \ell^2)$ from $X^1$ and $X^3$ is offset $(k^3, \ell^3)$ from $X^2$, which means $X^3$ is offset $(k^2 + k^3, \ell^2 + \ell^3)$ from $X^1$. We can rewrite this as a 3rd-order association

$$(X^1, (X^2, X^3)_{k^3\ell^3})_{k^2\ell^2} \Longleftrightarrow (X^1, X^2, X^3)_{k^2\ell^2,(k^2+k^3)(\ell^2+\ell^3)}, \tag{9}$$

but then we loose the fact that $(X^2, X^3)$ are composed into a 2nd-order feature. The order of binding does not matter, but of course that can modify the relative offset:

$$(X^1, (X^2, X^3)_{k^3\ell^3})_{k^2\ell^2} \Longleftrightarrow ((X^2, X^3)_{k^3\ell^3}, X^1)_{(-k^2)(-\ell^2)}.$$

When a hierarchical association like this is rewritten in the form of a simple higher-order association, as in (9), we always require that the induced offsets are valid, in the sense that none are identically $(0, 0)$ and none are the same. This ensures that each of the hidden states occupies a different, non-overlapping position in the image. The original generative model then asserts that they are all independent (and identically distributed).

A compositional association like this will be a suspicious coincidence if its highest binding is suspicious:

$$\mathbb{P}\left((X^1, (X^2, X^3)_{k^3\ell^3})_{k^2\ell^2} = (s^1, (s^2, s^3))\right) \gg \mathbb{P}(X^1 = s)\mathbb{P}((X^2, X^3)_{k^3\ell^3} = (s^2, s^3)).$$

Rewriting the left side as a simple higher-order suspicious coincidence (9) and using the empirical distribution gives

$$\begin{aligned}\hat{\mathbb{P}}((X^1, X^2, X^3)_{k^2\ell^2,(k^2+k^3)(\ell^2+\ell^3)} &= (s^1, s^2, s^3)) \\ &\gg \hat{\mathbb{P}}(X^1 = s^1)\hat{\mathbb{P}}((X^2, X^3)_{k^3\ell^3} = (s^2, s^3)).\end{aligned} \tag{10}$$

Both sides are easily computable using (3), (5) and their generalization (7). In fact, both terms on the right will have already been computed during the search for 2nd-order suspicious coincidences.

Even though the left side is a 3rd-order suspicious coincidence, we do not search all possible 6 000 such connected components. We demand that $(X^2, X^3)_{k^3\ell^3}$ is one of the 16 allowable 2nd-order associations found previously (Figure 3.8). As usual, we continue to require that $(X^1, X^2, X^3)_{k^2\ell^2,(k^2+k^3)(\ell^2+\ell^3)}$ forms a connected component (diagonals are not allowed). This gives 10 possibilities for $X^1$, 16 for $(X^2, X^3)_{k^3\ell^3}$ and 6 for $(k^2, \ell^2)$ for a total of 960 new associations to consider. While only a moderate reduction in the size of the search space, iterating this idea leads to enormous gains at higher levels.

We still use an MDL ranking (Section 3.5.2)

$$r((s^1,(s^2,s^3)_{k^3\ell^3})_{k^2\ell^2}) := \hat{\mathbb{P}}((X^1,X^2,X^3)_{k^2\ell^2,(k^2+k^3)(\ell^2+\ell^3)} = (s^1,s^2,s^3))$$
$$\times \log \frac{\hat{\mathbb{P}}((X^1,X^2,X^3)_{k^2\ell^2,(k^2+k^3)(\ell^2+\ell^3)} = (s^1,s^2,s^3))}{\hat{\mathbb{P}}(X^1=s^1)\hat{\mathbb{P}}((X^2,X^3)_{k^3\ell^3}=(s^2,s^3))},$$

finding 282 configurations with $r > 0$. Some of these configurations are identical because there can be multiple compositions that lead to the same pixel configuration. We immediately prune any exact (pixel level) repeats, leaving only a single representative composition for each (the one with the highest $r$). This leaves 258 compositions, shown in Figure 3.9 and ranked by decreasing $r$.

There are many similar elements and we use the same local maxima pruning procedure to get a sparser representation (see Section 3.5.3). We compute the distance between two $N$th-order patch configurations ($9N$ pixels), by first aligning the patches into the upper left corner of the same box (so a patch must be touching on the top and on the left), adding the per-pixel Hamming distance of the patches that align, adding 1/2 for each misaligned patch (not double counting) and dividing by $N$ to get a per-pixel measure. If this distance is less than 1/3, the configurations are neighbors. For example, if two $N$-th order configurations with states $(s_1^1,\ldots,s_1^N)$ and $(s_2^1,\ldots,s_2^N)$ have $K$ spatially overlapping patches, indexed by $\nu^1,\ldots,\nu^K$, then they will be neighbors if

$$\frac{\sum_{j=1}^{K}\|B_{s_1^{\nu_j}} - B_{s_2^{\nu_j}}\| + (1/2)(N-K)}{N} < \frac{1}{3}. \tag{11}$$

As discussed in Section 3.5.3, this just happens to work, is quite sensitive to the parameters 1/2 and 1/3 and can probably be accomplished with many added benefits using some sort of statistical distance.

The sparsened list has 28 3rd-order configurations, shown in Figure 3.10. Each of these is a suspicious coincidence between a single patch and a 2nd-order suspicious coincidence, all in a fixed configuration. Each is also a 3rd-order suspicious coincidence among single patches as defined by (8). In fact any compositional suspicious coincidence is also a (stronger) suspicious coincidence at all lower levels. This can be seen by multiplying the likelihood ratios. For example,

$$\begin{aligned}
&\frac{\hat{\mathbb{P}}((X^1,X^2,X^3)_{k^2\ell^2,(k^2+k^3)(\ell^2+\ell^3)} = (s^1,s^2,s^3))}{\hat{\mathbb{P}}(X^1=s^1)\hat{\mathbb{P}}(X^2=s^2)\hat{\mathbb{P}}(X^3=s^3)} \\
&\quad > \frac{\hat{\mathbb{P}}((X^1,X^2,X^3)_{k^2\ell^2,(k^2+k^3)(\ell^2+\ell^3)} = (s^1,s^2,s^3))}{\hat{\mathbb{P}}(X^1=s^1)\hat{\mathbb{P}}(X^2=s^2)\hat{\mathbb{P}}(X^3=s^3)} \frac{\hat{\mathbb{P}}(X^2=s^2)\hat{\mathbb{P}}(X^3=s^3)}{\hat{\mathbb{P}}((X^2,X^3)_{k^3\ell^3})} \\
&\quad = \frac{\hat{\mathbb{P}}((X^1,X^2,X^3)_{k^2\ell^2,(k^2+k^3)(\ell^2+\ell^3)} = (s^1,s^2,s^3))}{\hat{\mathbb{P}}(X^1=s^1)\hat{\mathbb{P}}((X^2,X^3)_{k^3\ell^3})} > 1,
\end{aligned}$$

where the first inequality comes from the fact that $(X^2,X^3)_{k^3\ell^3}$ is a suspicious coincidence and the second from the same fact about $(X^1,(X^2,X^3)_{k^3\ell^3})_{k^2\ell^2}$. This same idea iterates to higher orders, as does the computational procedure.

There are important reasons that we rank a coincidence using the highest level of composition and not the product of all the lowest level elements. A large suspicious coincidence can have a very large likelihood ratio when measured at the lowest level, to the degree that almost anything small can be composed with it and the likelihood ratio will still be much larger than one – a suspicious coincidence. While the new configuration is definitely unusual structure, the only interesting structure comes from the large piece. Nothing is added by composing it with the smaller piece. Moreover, in future work we expect the dependencies discovered at each new level to be incorporated into an updated model. In this new model, the dependencies are accounted for and are no longer suspicious coincidences when compared to the lowest level. The only interesting dependencies are to be found in new compositions which are naturally compared to the marginal statistics of the current level, not the all the way back to the lowest level.

## 3.6 Results

We can easily iterate the procedure described thus far. We look for suspicious coincidences between single patches and 3-patch configurations and also between two 2-patch configurations to discover 4th-order suspicious coincidences. We remove identical (pixel level) elements, rank with an MDL measure that only considers the highest level of composition and sparsen using (11). This gives 26 level 4 configurations shown in Figure 3.11. Now we compose levels 1 and 4 and levels 2 and 3 to search level 5. The sparsened level 5 representation has 26 elements shown in Figure 3.12. We continue in this manner until level 8 (Figures 3.13–3.15), after which we begin to have computational difficulties (mostly because of the inefficiency of our Matlab implementation when generating a list of all possible compositions that need to be considered). The number of elements in each level is shown in the next table.

| order | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| # elements | 10 | 16 | 28 | 26 | 26 | 65 | 81 | 158 |

The higher levels begin to look quite noisy. Indeed, an 8th-order suspicious coincidence is composed of 72 pixels and thus represents unusual statistical structure in a 72-dimensional space. Estimating such structure should require a lot of data. Increasing the number of images for training improves things somewhat. We have experimented with up to 4000 images. Because the representation is less noisy, the local maxima sparsening procedure produces fewer and less noisy elements. This allows the method to continue into the 10th or 11th stage of iteration before things become noisy and the computation breaks down.

We have been able to find 32-order suspicious coincidences by skipping stages and by considering only a small subset of the possible compositions at each stage. At each new level, we only consider compositions created by combining together two configurations from the previous level. This creates suspicious coincidences with orders that are successive powers of 2. We also use a different sparsening procedure, keeping only the best configuration (according to the MDL ranking $r$) that uses each composition found in the previous level. This prevents the number of elements from increasing. This highly constrained search finds a few 32-patch horizontal and vertical perfect edges. These compositions are $96 \times 3$ pixel structures, which is a significant edge in a $126 \times 192$ pixel image. It seems unlikely that any of the training images contain one of these configurations exactly (although we have not checked), but the model allows for noisy perturbations. These detected structures probably correspond to horizon lines, tree trunks, roads, buildings and other such large image features.

### 3.6.1 Other data sets

We have experimented briefly with several other image data sets, including the same images used here but at a much higher resolution ($1023 \times 1536$). The results are all qualitatively similar. Some of the differences between natural images and text images are worth mentioning.

We used the same 10 representative patches, but trained the model parameters and searched for suspicious coincidences with (single author) handwritten text images. The resulting compositions were quite similar to the ones we found here with three notable exceptions. Line-like compositions were more prevalent than edge-like compositions, large all-black compositions were absent and diagonal lines had replaced vertical lines. This last difference is important. It represents the slant of the handwriting and demonstrates that the algorithm is not forced to find vertical and horizontal edges exclusively, even though it is clearly biased in that direction because the of grid nature of the pixel representation and because of the vertical and horizontal (but no diagonal) neighborhood constraint.

Another interesting data set was fixed font (Times New Roman 12pt) text, generated by turning a PDF manuscript into a binary GIF image. Again we used the same 10 representative patches. We had hoped that the algorithm might discover actual letters, but this was not the case. The suspicious coincidences that we discovered mostly represented features characteristic of the typesetting, such as the typical inverted T shape at the bottom of many letters. We also found very long horizontal lines and even long parallel pairs of horizontal lines representing the bottom of a line of text or in the parallel case, two lines of text. These sorts of features quickly came to dominate the representation.

## 3.7 Related Work

Chapter 2 contains several references on the general theme of sequential model building. Section 3.2.2 of this chapter contains some pointers to related work on unsupervised feature selection. There is also a large body of empirical work describing various properties of natural image statistics, including the strong signal for lines and edges that our model naturally detected, for example, [10, 6, 1]. Here we will focus mostly on work related to growing hierarchies of features with increasing selectivity. Note that almost any hierarchical neural network model will loosely fit into this category (for example [20]), but we do not review that literature here.

The general principle behind the feature induction algorithm in [23] is similar to the idea here of growing more complicated features from simpler features that have already been discovered and incorporated into the model. Other comparisons with this work are discussed in Chapter 2.

In Section 3.2.2 we noted that our initial generative model bears some resemblance to sparse coding models, like independent components analysis (ICA). Several groups have experimented with various methods of adding more hidden layers on top of a sparse coding basis in order to extract higher-order dependencies, much like our goal here. Although these methods typically only operate on a single patch, whereas here we are operating on spatially adjacent patches, this difference is perhaps not as big as it seems. Sparse coding bases represent much larger patches (64×64-pixel patches are not uncommon) with the result that many basis elements are confined to a small spatial extent within the large patch. These spatially localized basis functions are analogous to our original 10 ideal patches. Extracting higher-order dependencies within the large patch is then analogous to composing our small patches into larger ones.

Hyvärinen and Hoyer (2001) [17, 18] show that the dependencies among ICA units can be used to define a topographic ordering on the units. This topography can then be used in various ways to (locally) pool the outputs of the ICA units and create a new layer of units with interesting properties. Depending on the (nonlinear) pooling function, these higher-order units can have various properties, including certain types of invariance. Further adding another sparse coding layer on top of these units creates basis functions that often look like longer lines and edges [13], reminiscent of the results here. Presumably this procedure could be iterated to create increasingly complex units. Note that the nonlinear pooling operation is important because simply iterating ICA on the same patch is not effective: all the transformations are linear and can be collapsed into a single linear transformation.

Karklin and Lewicki (2003) [19] model higher-order dependencies among ICA units in a somewhat different fashion. They add a second hidden layer which is essentially another ICA basis, not for the outputs of the first layer, but for the variance (technically, for a dispersion parameter) in the outputs of the first layer. This introduces the nonlinearity between successive applications of ICA in a much more general fashion. It also allows the second layer to capture very coarse information about the image patch. Many of the higher-order basis functions can be interpreted as longer lines and edges, but there are also many units that appear to be capturing certain textural properties. Again, this procedure could presumably be iterated.

Fleuret and D. Geman (2001) [7] use decision trees arranged in a coarse-to-fine hierarchy for face detection in cluttered backgrounds. Interestingly, although the work is motivated by computational concerns and the learning is supervised, their feature selection procedure is quite similar to ours. In particular, they combine low-level features into higher level features exactly when the low-level features are strongly correlated (i.e., when they are a suspicious coincidence) on objects of interest (in this case, faces). They provide a much more rigorous mathematical framework than we do for investigating such hierarchies of suspicious coincidences and they also use the resulting representation for a difficult object detection task. In future work we hope to further investigate connections between the two approaches.

## 3.8 Discussion

We developed some heuristics based on detecting sparse collections of suspicious coincidences which allow us to discover simple hierarchical features in binary natural images. The results are not surprising: there is a strong signal for detecting lines and edges. The method has several problems which prevent us from using it recursively to explore the possibility that larger and more interesting structure could be discovered.

One of the striking deficiencies in the method described here is the lack of invariance. Although we have used some location invariance to speed learning and reduce dimensionality, we are still constrained to the $3 \times 3$ grid. This means that every feature will have at least 9 different copies that need to be learned. Furthermore, the model only operates at a single scale and there is no invariance to illumination or rotation or deformation. These types of invariance are likely to be important in any realistic composition system. The lack of them will necessitate an explosion of features, all slightly different, that would not occur if these differences were captured by the appropriate invariance. Even in the results reported here it is possible to see how the representation is growing too fast and becoming noisy partly because of the lack of invariance. We hope to be able to incorporate more types of invariance in later versions of these ideas. In Chapter 4 we demonstrate that invariant versions of these features can also be learned using a similar learning heuristic.

Another important improvement will be a more robust sparsening procedure. The method used here, which is based on comparing pixel-level representations of higher-order structure, becomes problematic once the high-level representations have a lot of invariance. Using a statistical distance seems like an obvious next step and we will have to investigate this along with invariance. Some type of local maxima procedure will likely still be important. The number of compositions quickly becomes too large to exhaustively search as we have done here. A local gradient method for good suspicious coincidences might simultaneously search the space and create sparseness.

One of the long term goals is model updating. After a new level of structure has been discovered, the model should be updated to take into account these new dependencies. In the updated model, this structure will no longer be unusual and higher-level dependencies can be investigated. This will presumably lead to a better probabilistic description of images and make possible improved image processing algorithms – the whole point of this endeavor. Early indications suggest that incorporating invariance will involve many of the same technical issues as model updating, so we may be trying to solve all of these problems simultaneously.

# Bibliography

[1] Horace Barlow. What is the computational goal of the neocortex? In Christof Koch and Joel L. Davis, editors, *Large-Scale Neuronal Theories of the Brain*, pages 1–22. MIT Press, Cambridge, 1994.

[2] Andrew Barron, Jorma Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, October 1998.

[3] Anthony J. Bell and Terrence J. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338, 1997.

[4] Adam L. Berger, Stephen Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society B*, 46:1–38, 1977.

[6] A. Desolneux, L. Moisan, and J.-M. Morel. Computational gestalts and perception thresholds. *Journal of Physiology - Paris*, 97(2–3):311–322, 2003.

[7] François Fleuret and Donald Geman. Coarse-to-fine face detection. *International Journal of Computer Vision*, 41:85–107, 2001.

[8] P. Földiák. Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics*, 64:165–170, 1990.

[9] J. Friedman, W. Stuetzle, and A. Schroeder. Projection pursuit density estimation. *Journal of the American Statistical Association*, 79:599–608, 1984.

[10] W.S. Geisler, J.S. Perry, B.J. Super, and D.P. Gallogly. Edge co-occurrence in natural images predicts contour grouping performance. *Vision Research*, 41:711–724, 2001.

[11] D. Geman and A. Koloydenko. Invariant statistics and coding of natural microimages. In *Proceedings, IEEE Workshop on Statistical and Computational Theories of Vision*, Fort Collins, CO, June 1999.

[12] Geoffrey E. Hinton. Products of experts. In *Proceedings of the International Conference on Artificial Neural Networks*, volume 1, pages 1–6, Edinburgh, U.K., 1999.

[13] Patrik O. Hoyer and Aapo Hyvärinen. A multi-layer sparse coding network learns contour coding from natural images. *Vision Research*, 42:1593–1605, 2002.

[14] Shih-Hsiu Huang. *Compositional approach to recognition using multi-scale computations.* PhD thesis, Division of Applied Mathematics, Brown University, 2001.

[15] Peter Huber. Projection pursuit. *Annals of Statistics*, 13(2):435–475, 1985.

[16] A. Hyvärinen, P.O. Hoyer, and J. Hurri. Extensions of ICA as models of natural images and visual processing. In *Proceedings of the International Symposium on Independent Component Analysis and Blind Source Separation (ICA2003)*, pages 963–974, Nara, Japan, 2003.

[17] Aapo Hyvärinen and Patrik O. Hoyer. A two-layer sparse coding model learns simple and complex cell receptive fields and topography from natural images. *Vision Research*, 41:2413–2423, 2001.

[18] Aapo Hyvärinen, Patrik O. Hoyer, and Mika Inki. Topographic independent component analysis. *Neural Computation*, 13:1527–1558, 2001.

[19] Yan Karklin and Michael S. Lewicki. Learning higher-order structures in natural images. *Network: Computation in Neural Systems*, 14:483–499, 2003.

[20] Guy Mayraz and Geoffrey E. Hinton. Recognizing handwritten digits using hierarchical products of experts. *IEEE Transactions on Pattern Analysis and Machine Vision*, 24(2):189–197, February 2002.

[21] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[22] C. G. Phillips, S. Zeki, and H. B. Barlow. Localization of function in the cerebral cortex: past, present and future. *Brain*, 107(1):327–361, March 1984.

[23] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, April 1997.

[24] Daniel Frederic Potter. *Compositional Pattern Recognition.* PhD thesis, Division of Applied Mathematics, Brown University, 1999.

[25] Stefan Roth and Michael J. Black. Field of experts: A framework for learning image priors with applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005 (submitted).

[26] M. F. Tappen, B. C. Russell, and W. T. Freeman. Efficient graphical models for processing images. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 673–680, Washington, DC, 2004.

[27] J. H. van Hateren and A. van der Schaaf. Independent component filters of natural images compared with simple cells in primary visual cortex. *Proceedings of the Royal Society of London B*, 265:359–366, 1998.
`http://hlab.phys.rug.nl/archive.html`.

[28] Max Welling, Richard S. Zemel, and Geoffrey E. Hinton. Probabilistic independent components analysis. *IEEE Transactions on Neural Networks*, 15(4):838–849, July 2004.

[29] Song Chun Zhu and David Mumford. Prior learning and gibbs reaction-diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11):1236–1250, November 1997.

[30] Song Chun Zhu, Ying Nian Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural Computation*, 9(8):1627–1660, November 1997.

[31] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.

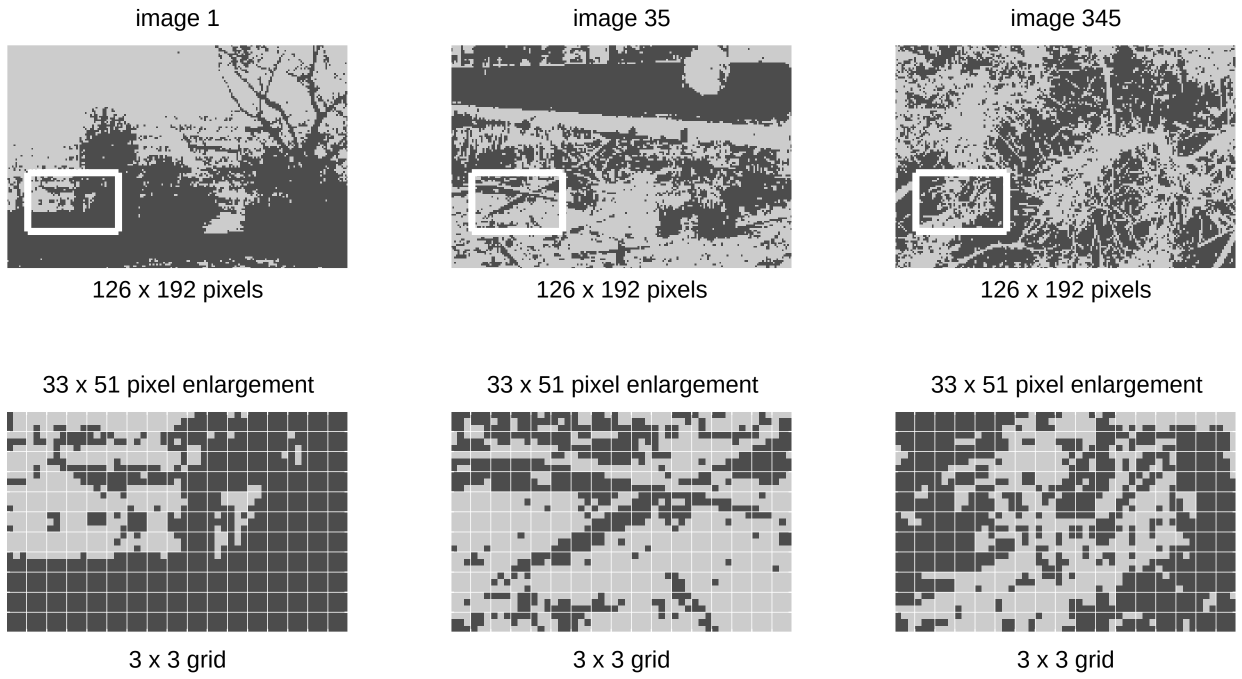

Figure 3.1: Sample images with enlargements [27].

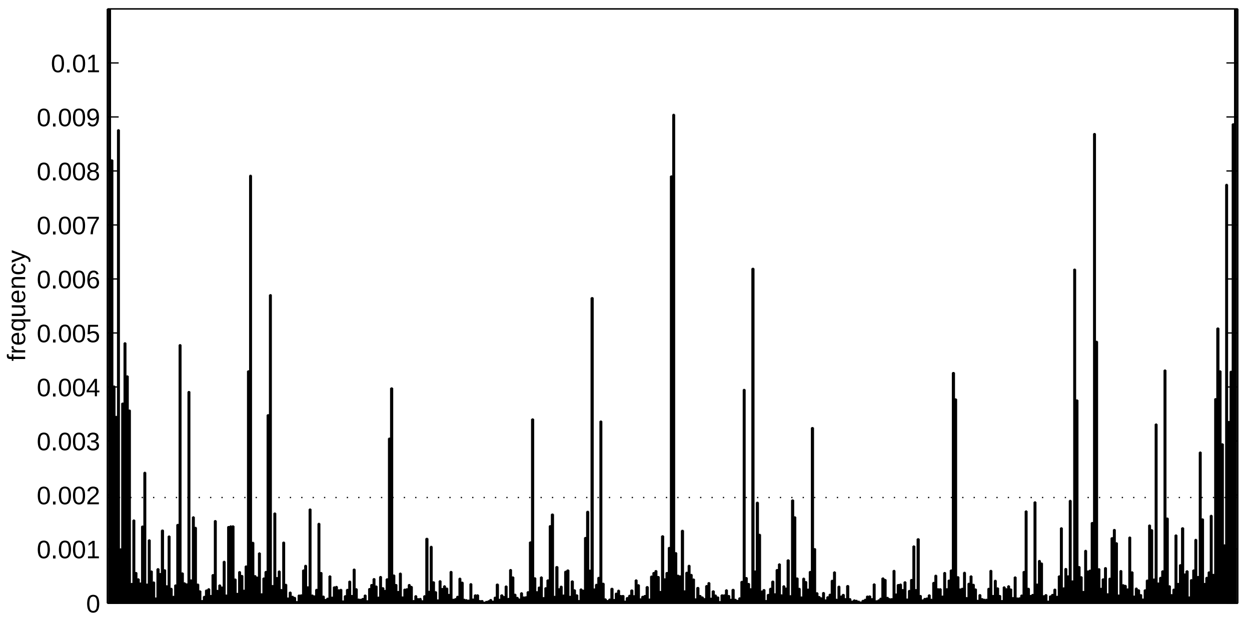

Figure 3.2: The empirical probability distribution of all $3\times3$ binary patches ($2^9 = 512$ total). The dotted line is 1/512. The entropy of this distribution is 4.65 bits (uniform is 9 bits).

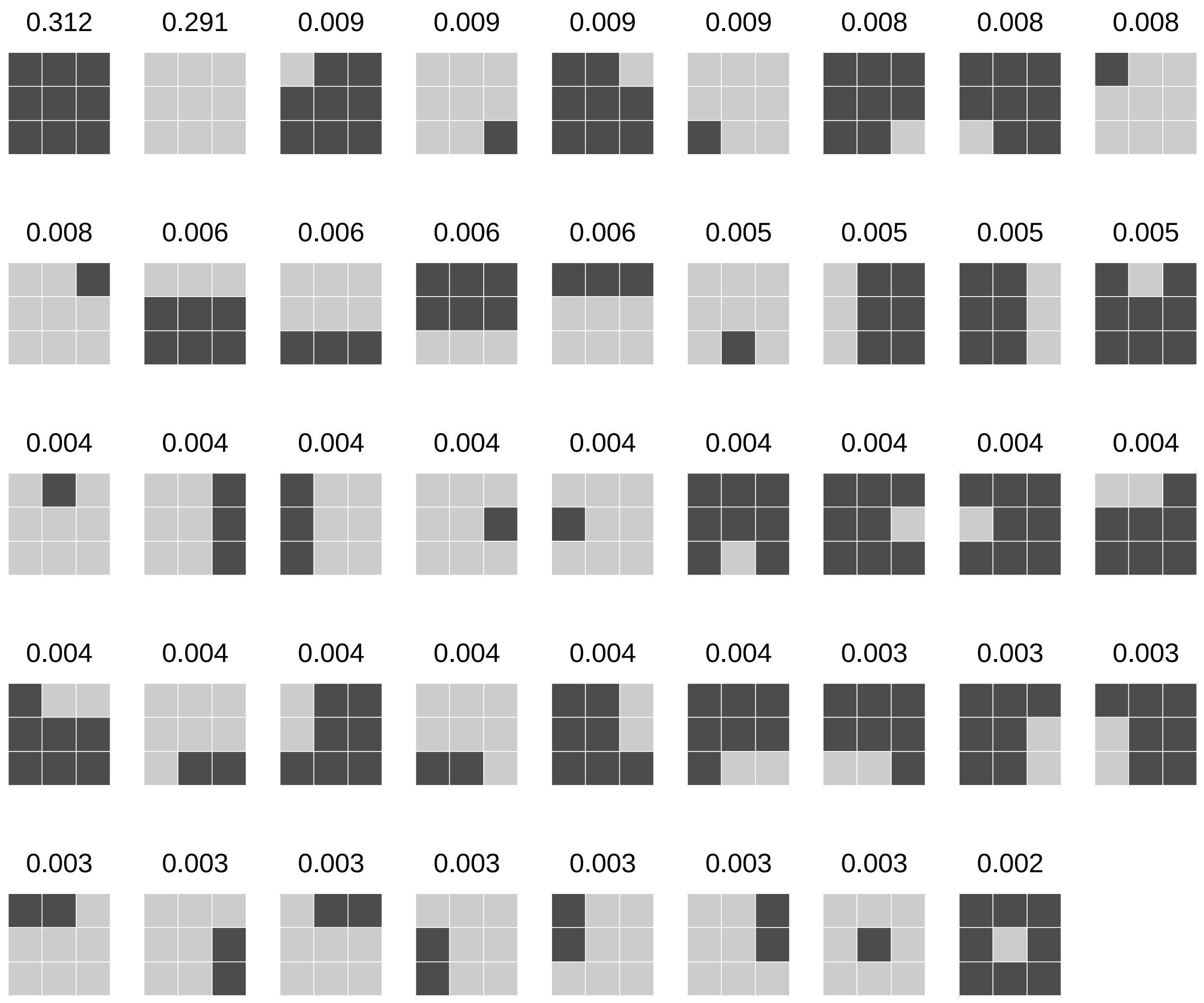

Figure 3.3: The 44 images patches that were suspicious coincidences and their empirical probabilities.

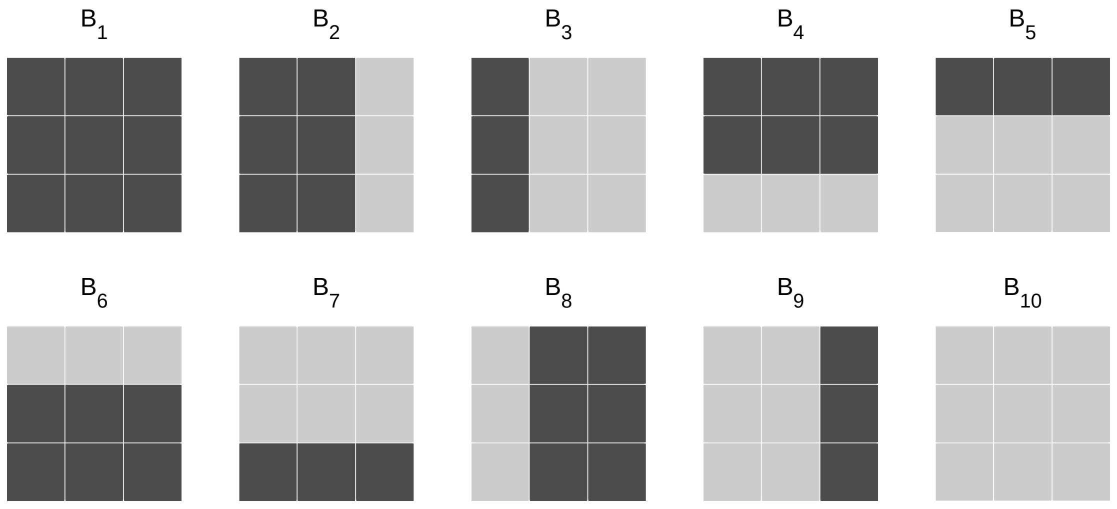

Figure 3.4: The 10 representative patches $B_s$ found by sparsening the suspicious coincidences in Figure 3.3.

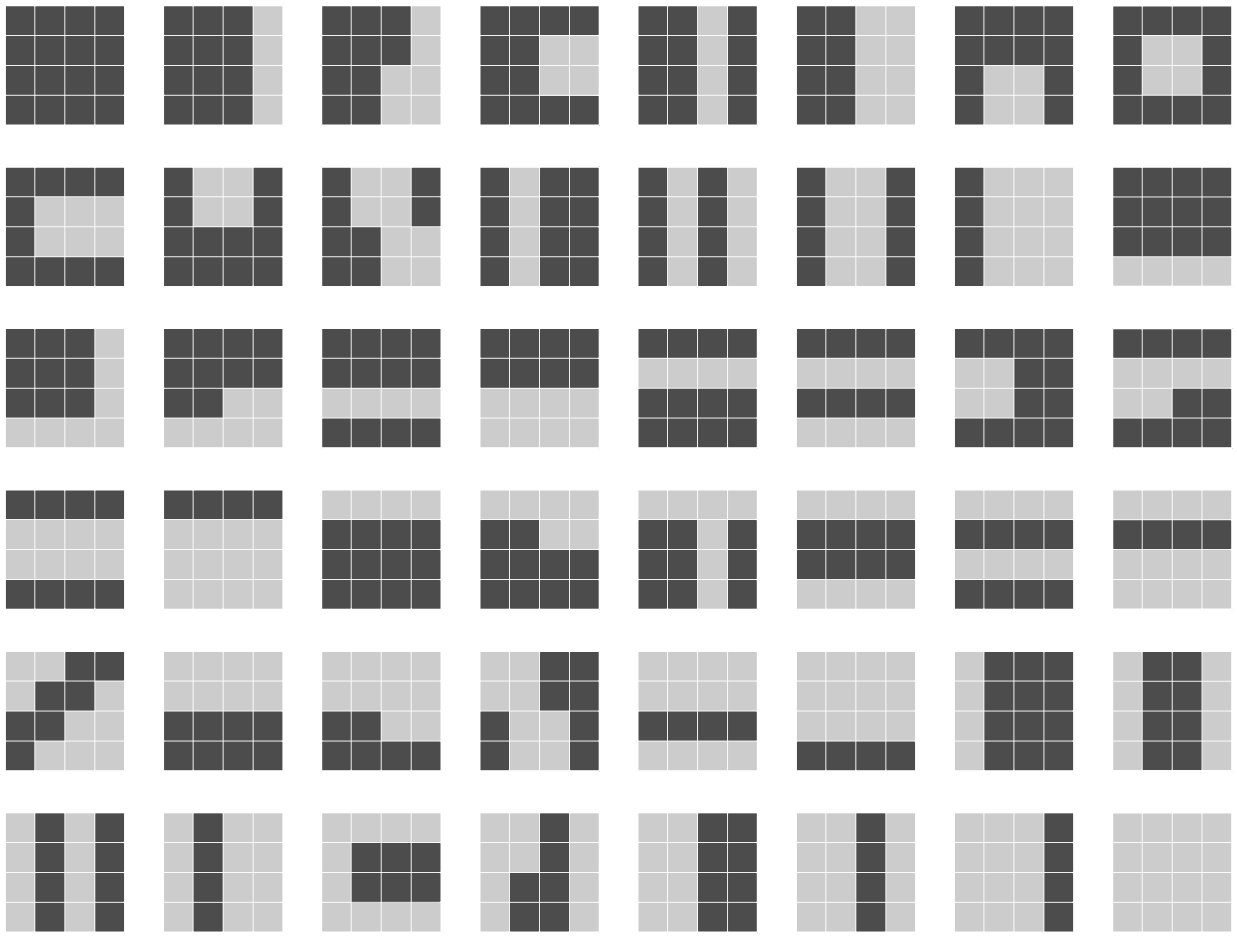

Figure 3.5: The 48 representative $4 \times 4$ patches found in binary images using the methods of Section 3.2.1. These are not used in this chapter, but a similar set derived from a slightly different data set is used in Chapter 4.

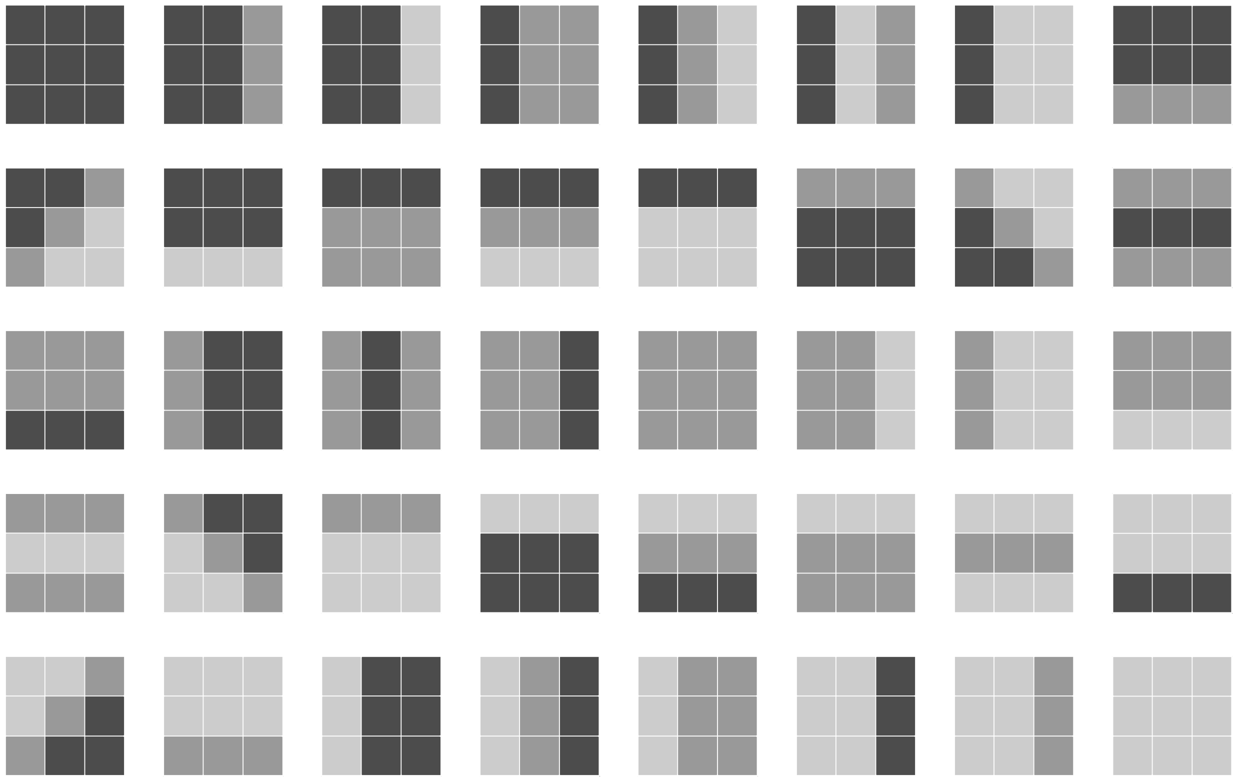

Figure 3.6: The 40 representative $3 \times 3$ patches found in *ternary* images using the methods of Section 3.2.1. These are not used here.

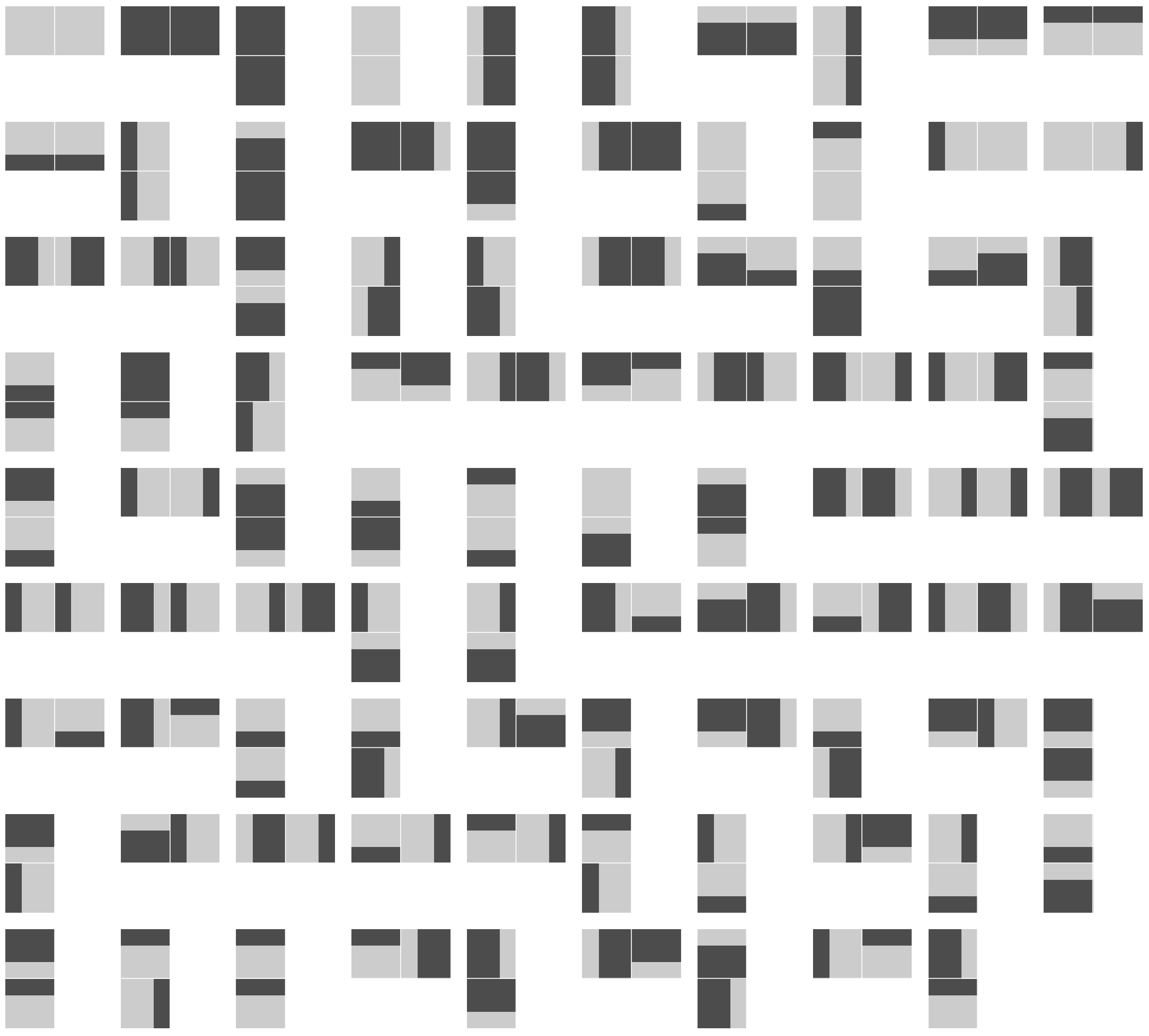

Figure 3.7: The 89 suspicious coincidences composed of two neighboring representative patches $B_s$. MDL ranked in decreasing order from left to right.

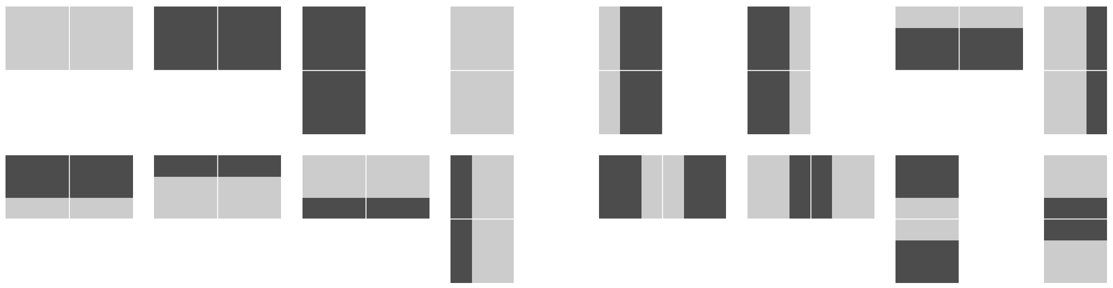

Figure 3.8: The 16 representative 2-patch compositions, found by sparsening the representation in Figure 3.7. MDL ranked in decreasing order from left to right.

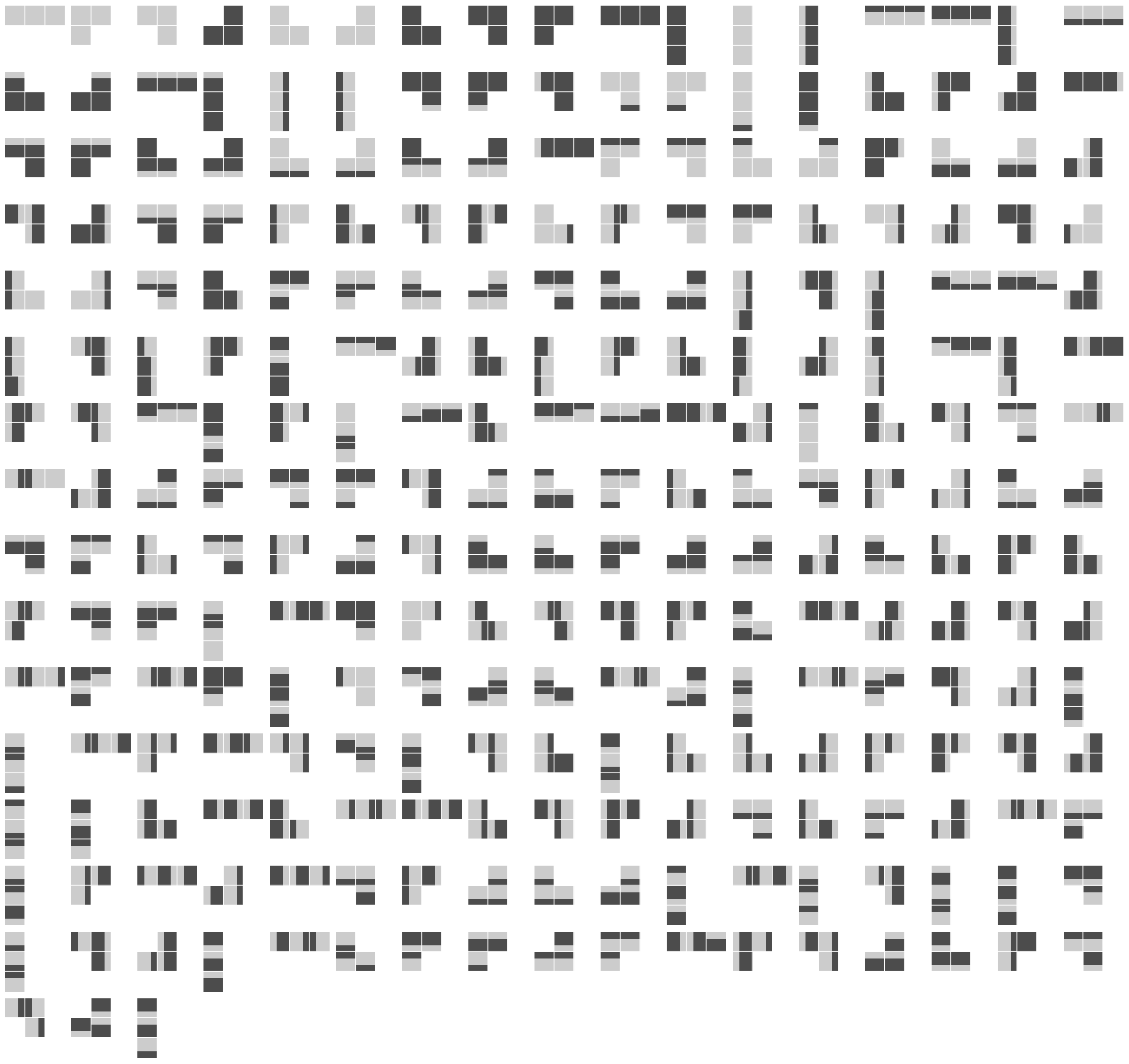

Figure 3.9: The 258 suspicious coincidences (after removing exact pixel repeats) made with three neighboring representative patches $B_s$ and formed by composing one of the single patches in Figure 3.4 with one of the double patches in Figure 3.8. MDL ranked in decreasing order from left to right.

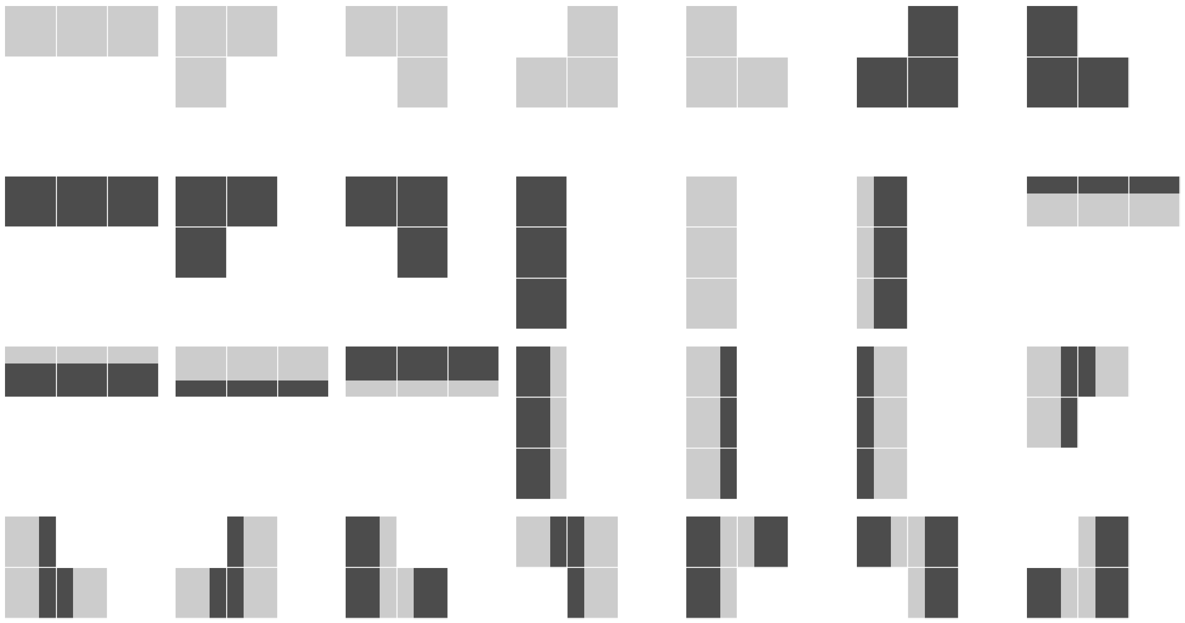

Figure 3.10: The 28 representative 3-patch compositions, found by sparsening the representation in Figure 3.9. MDL ranked in decreasing order from left to right.

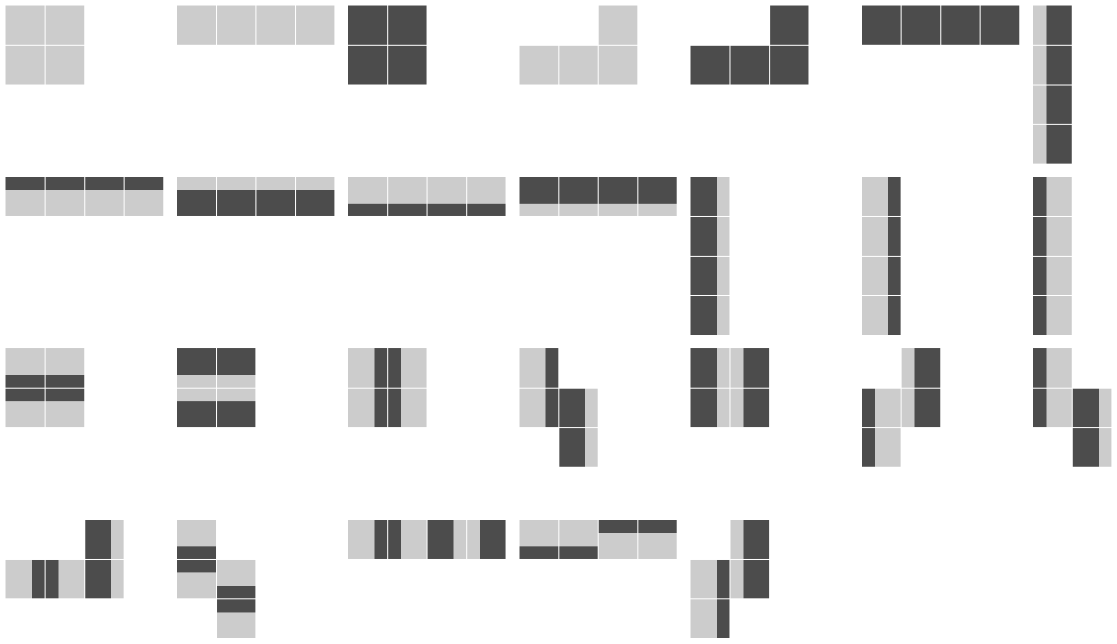

Figure 3.11: The 26 sparsened 4-patch compositions, MDL ranked in decreasing order from left to right.

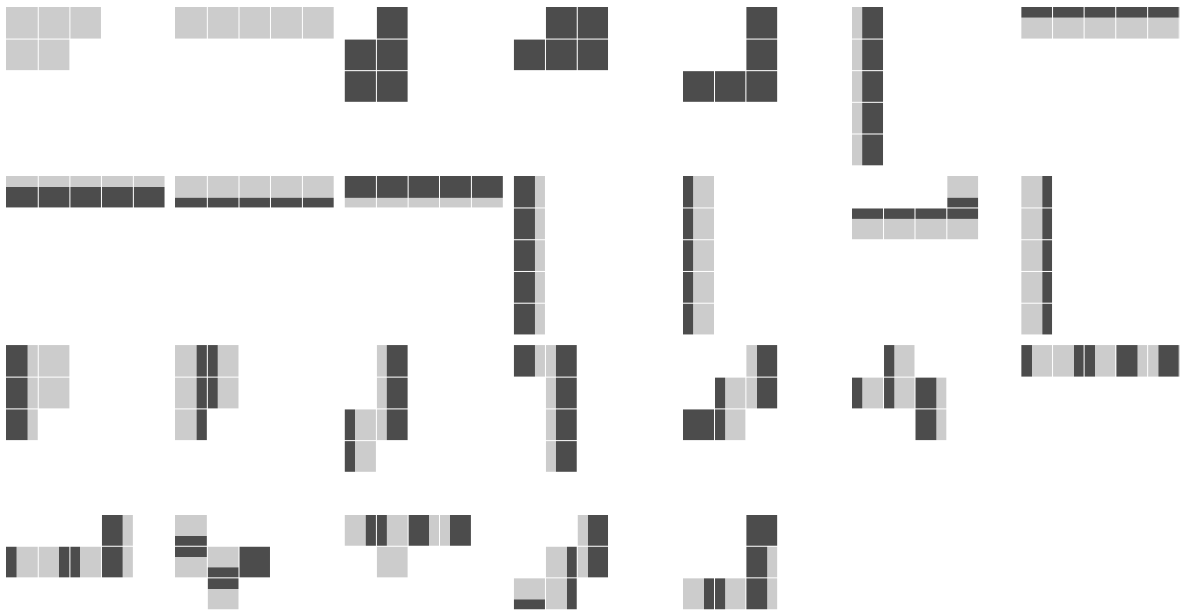

Figure 3.12: The 26 sparsened 5-patch compositions, MDL ranked in decreasing order from left to right.

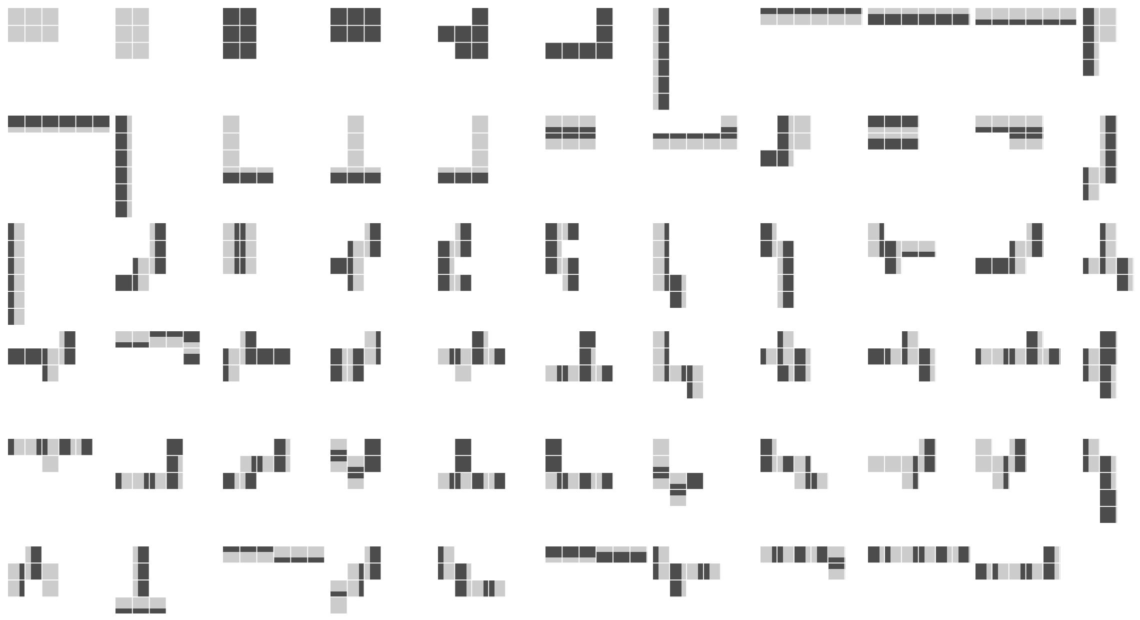

Figure 3.13: The 65 sparsened 6-patch compositions, MDL ranked in decreasing order from left to right. At this level the representation becomes noisy, but this can be remedied with more training examples.

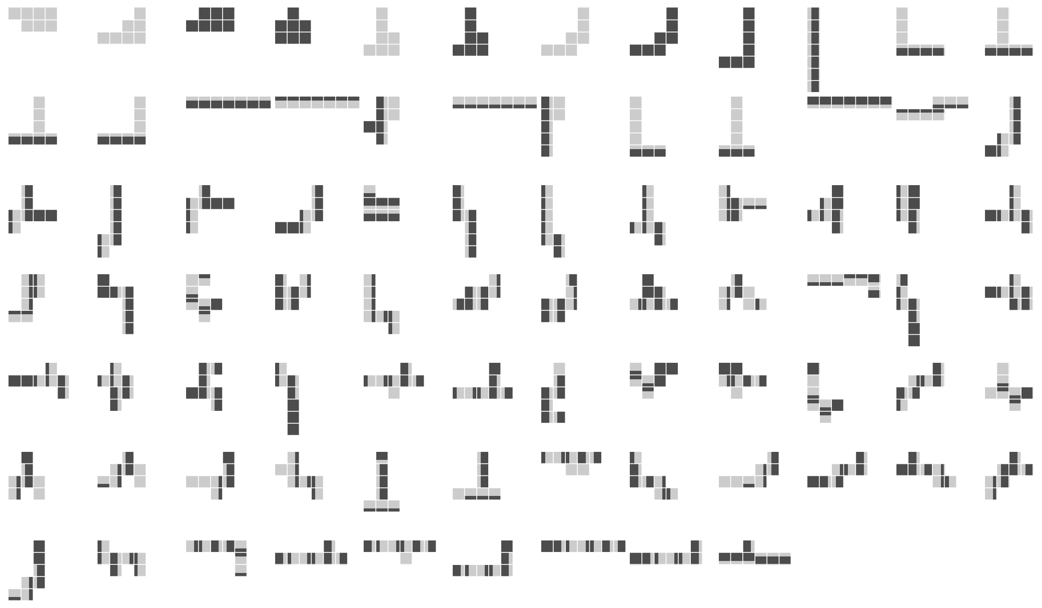

Figure 3.14: The 81 sparsened 7-patch compositions, MDL ranked in decreasing order from left to right.

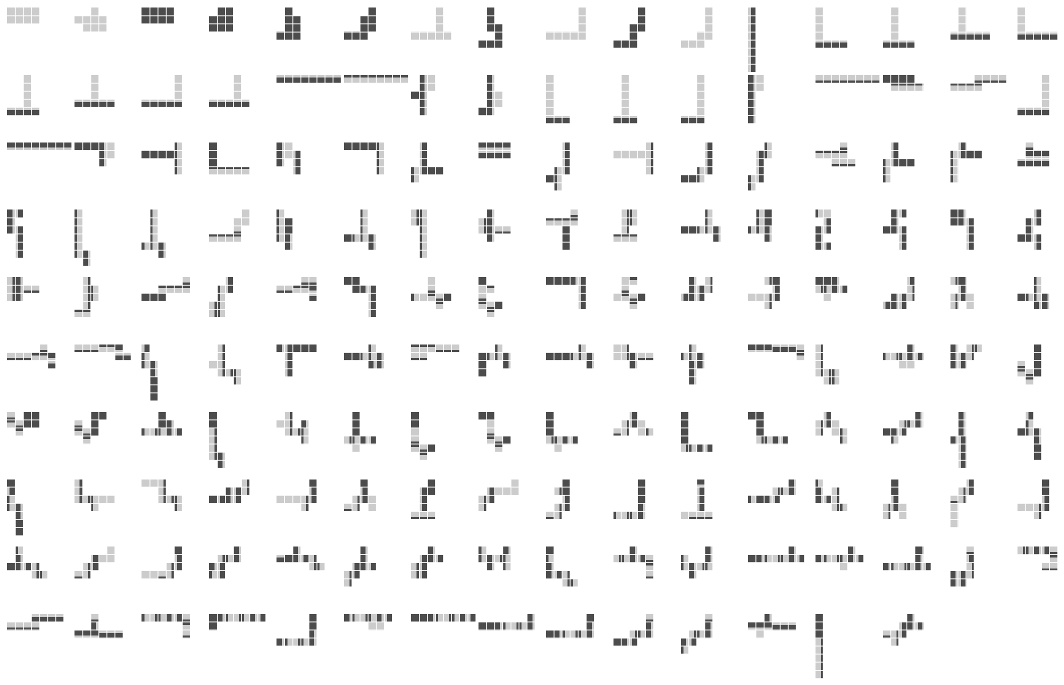

Figure 3.15: The 158 sparsened 8-patch compositions, MDL ranked in decreasing order from left to right.

# Chapter 4

# Learning invariance

## 4.1 Introduction

As demonstrated in Chapter 3, directly applying the learning principle from Chapter 2 appears to create hierarchies of increasing selectivity, but not of increasing invariance. This lack of invariance severely limits both the utility of the resulting representation and the robustness of the learning algorithm. In this chapter we explore the possibility that a similar learning principle can be used to build invariant representations.

One way to learn invariant representations involves using temporal information from image sequences [8, 17]. The idea is based on the observation that an object persists longer than any particular view of the object, so a useful invariant representation should be stable over time. For example, in a typical natural video sequence, the only things that change over several consecutive frames are things like camera position, illumination, and object deformations or articulations. These are exactly the types of invariances that we want to recover and a representation that is stable over time will likely be invariant to these changes.

In this chapter we describe a probabilistic model that tries to capture this idea of temporal stability. The main point is that this model can be built incrementally and hierarchically in the spirit of the general learning heuristic in Chapter 2. In particular, *temporal* suspicious coincidences indicate deficiencies in the model. These deficiencies can be corrected by adding a new state to the model. This new state will behave like an *invariant* feature detector. Recall that in Chapter 3 we used the same ideas with *spatial* suspicious coincidences to build *selective* feature detectors.

We first describe a general class of models (Sections 4.2–4.3), then a version tailored to image patches (Section 4.4) and then some very simple experiments fitting the model to natural (binary) image patches (Section 4.5). Several other methods, like slow feature analysis [17], have demonstrated that the principle of temporal stability can create representations with a variety of different invariances (Section 4.6). In light of this, although our experiments are quite preliminary, they hint at the possibility of iteratively learning hierarchies of both selectivity and invariance within a common probabilistic framework (Section 4.7).

## 4.2 Hierarchical independent switching models

We want to use the principle of temporal stability to create invariant representations. At the same time, we would like to follow as closely as possible the spirit of the incremental learning heuristic in Chapter 2. Most importantly, these newly created invariant representations should improve our model in a statistical sense. This suggests using new variables to better model temporal dependencies. The idea of temporal stability further suggests incorporating dependencies with some form of switching regime model.

For example, suppose the current model contains components that correspond to lines and edges in an image, but at specific locations, scales and orientations. Suppose also that the current model assumes that consecutive frames in a video sequence are independent. In natural video, however, an edge in one frame strongly indicates the presence of a similar edge in the next frame, perhaps at a slightly different location, scale or orientation. Incorporating these dependencies would improve the model. One way to incorporate them is with switching regimes.

For example, a new binary variable $Z$ that is strongly (positively) correlated with itself could be introduced into the model. When $Z = 1$, the original model is perturbed so that a certain collection of edges (presumably, all with similar locations, scales and orientations) is much more likely. When $Z = 0$, there is no perturbation. This new variable introduces temporal dependencies that better model the data. Furthermore, $Z$ will likely behave like an invariant edge detector if the model is used for (still) image interpretation. This is because $Z = 1$ will have higher posterior probability for images that contain any edge in the collection of edges that $Z$ influences.

We have formalized part of this idea in what we call a *hierarchical independent switching model (HISM)*. Later we will discuss hidden HISMs (HHISMs), which are a special case of hierarchical hidden Markov models [7]. A HISM is almost a sequence of probabilistic context free grammars (PCFGs), but not quite. The notation that we use is quite similar. Let $\mathcal{A} = \{1, 2, \ldots, N, \texttt{start}\}$ be an ordered alphabet, which is composed of terminals $\mathcal{T} = \{1, \ldots, M\}$, $(M \leq N)$, nonterminals $\mathcal{V} = \{M+1, \ldots, N\}$ and a special $\texttt{start}$ symbol. Let $p_{\texttt{start}}$ be a probability distribution on $\mathcal{A}_{\texttt{start}} = \mathcal{A} \setminus \{\texttt{start}\}$ and define $q_{\texttt{start}} = 0$. To each $\alpha \in \mathcal{V}$ we associate a switching parameter $q_\alpha \in (0, 1]$, a subalphabet $\mathcal{A}_\alpha \subseteq \{1, \ldots, \alpha - 1\}$ and a probability distribution $p_\alpha$ over $\mathcal{A}_\alpha$.

We sample from a HISM as follows. Begin with $S_1, S_2, \ldots$ i.i.d. with distribution $p_{\texttt{start}}$ over $\mathcal{A}_{\texttt{start}}$. For each nonterminal $S_i$, generate $S_{i1}, \ldots, S_{iN_i}$ i.i.d. with distribution $p_{S_i}$ over $\mathcal{A}_{S_i}$, where $N_i$ is geometric with stopping parameter $q_{S_i}$, that is,

$$\text{Prob}\{N_i = n | S_i = \alpha\} = (1 - q_\alpha)^{n-1} q_\alpha \qquad n \geq 1.$$

Now, for each nonterminal $S_{ij}$, generate $S_{ij1}, \ldots, S_{ijN_{ij}}$ i.i.d. with distribution $p_{S_{ij}}$ over $\mathcal{A}_{S_{ij}}$, where $N_{ij}$ is geometric with stopping parameter $q_{S_{ij}}$. Iterate this process until only terminals remain.

This process creates an i.i.d. sequence of trees, where each $S_i$ is a root, where each $S_{i_1 \cdots i_m}$ $(m > 1)$ has $S_{i_1 \cdots i_{m-1}}$ as its parent and where $S_{i_1 \cdots i_n}$ is a leaf if and only if it is a terminal. This is represented pictorially in Figure 4.1. Each tree is finite with probability one. Each is finite in depth because of the hierarchical arrangement of the subalphabets $\mathcal{A}_\alpha \subseteq \{1, \ldots, \alpha - 1\}$.

Indeed, the depth is at most $N - M + 1$. Because of this and the fact that the geometric distribution is finite with probability one (each $q_\alpha > 0$), each tree is also finite in breadth.

The distribution on each tree does not exactly correspond with the distribution on parse trees for any PCFG, although the difference is mostly notational. The problem is that the production rules for a PCFG are typically constrained to come from a finite set, whereas here the production rules for each $\alpha \in \mathcal{V}$ are all finite sequences from $\mathcal{A}_\alpha$. The distribution on terminals from each tree, however, does correspond with the distribution on terminals for a PCFG. In particular, the production rules for the `start` symbol are $\{\mathtt{start} \mapsto \alpha : \alpha \in \mathcal{A}_{\mathtt{start}}\}$ with distribution $\text{Prob}\{\mathtt{start} \mapsto \alpha\} = p_{\mathtt{start}}(\alpha)$, and the production rules for each $\alpha \in \mathcal{V}$ are $\{\alpha \mapsto \beta : \beta \in \mathcal{A}_\alpha\} \cup \{\alpha \mapsto \beta\alpha : \beta \in \mathcal{A}_\alpha\}$ with distribution $\text{Prob}\{\alpha \mapsto \beta\} = p_\alpha(\beta)q_\alpha$ and $\text{Prob}\{\alpha \mapsto \beta\alpha\} = p_\alpha(\beta)(1 - q_\alpha)$.

We do not further explore this connection to PCFGs here, although the PCFG representation does emphasize an important fact about our choice of geometric distributions for the $N$'s. Another way to think about generating $N$ i.i.d. samples from $p$ when $N$ is geometric with stopping parameter $q$ is the following. First, generate a single sample from $p$, then flip a coin to see if you stop (with probability $q$ of stopping). Repeat this (independently) until you stop. When you finally stop, the number of i.i.d. samples from $p$ will be a random variable $N$ with a geometric ($q$) distribution. That the $N$'s can be modeled as a sequence of independent binary decisions endows HISMs with a Markov structure that we will use for computation and estimation. In fact, on the appropriate state space a HISM is just a (strangely parameterized) Markov chain.

### 4.2.1 Markov formulation

The leaves (terminals) can be ordered from left to right just as they are depicted in Figure 4.1. In particular, leaves from the tree with root $S_i$ come before leaves from the tree with root $S_j$ for $i < j$, and similarly, leaves from a subtree with root $S_{i_1 \cdots i_m i}$ come before leaves from a subtree with root $S_{i_1 \cdots i_m j}$ for $i < j$. We use $T_1, T_2, \ldots$ to denote the sequence of terminals thus ordered.

For each $T_k$ we can trace the path through the appropriate tree from its root to $T_k$. In particular, if $T_k$ corresponds to $S_{i_1 \cdots i_m}$, then the path is

$$Z_k = (Z_{k1}, \ldots, Z_{km}) = (S_{i_1}, S_{i_1 i_2}, \ldots, S_{i_1 i_2 \cdots i_m}).$$

One such path is highlighted in Figure 4.1. Notationally, it will be convenient later to prepend the `start` symbol to each $Z_k$, so that

$$Z_k = (Z_{k0}, Z_{k1}, \ldots, Z_{km}) = (\mathtt{start}, S_{i_1}, S_{i_1 i_2}, \ldots, S_{i_1 i_2 \cdots i_m}).$$

A HISM thus generates a sequence of such paths $Z_1, Z_2, \ldots$, where each $Z_k$ is an element of

$$\mathcal{Z} = \{\text{finite sequences } (\alpha_0, \alpha_1, \ldots, \alpha_m) : \alpha_0 = \mathtt{start}, \alpha_m \in \mathcal{T}, \alpha_i \in \mathcal{A}_{\alpha_{i-1}}, i \geq 1\}.$$

It is not hard to see that this sequence is a (homogeneous, first order) Markov chain on the state space $\mathcal{Z}$.

In general, each sequence $Z_1, Z_2, \ldots$ will be consistent with multiple sequences of trees.

The end result of this is that the natural parameters of the HISM (the $q_\alpha$'s and the $p_\alpha$'s) do not conveniently parameterize the transition probability matrix of the corresponding Markov chain. One way to uniquely specify the sequence of trees is to augment each path $Z_k$ with the switch point $W_k$ from $Z_k$ to $Z_{k+1}$. $W_k$ is the root of the largest subtree containing $T_k$ and not $T_{k+1}$. For example, if $T_k = S_{i_1 \cdots i_m}$ and $W_k = S_{i_1 \cdots i_j}$, then $T_{k+1} = S_{i_1 \cdots i_{j-1}(i_j+1)1 \cdots 1}$. An example of the switch point for the highlighted path is shown in Figure 4.1. The switch point can never be the prepended `start` symbol.

Define $X_k = (Z_k, W_k)$. Then the sequence $X_1, X_2, \ldots$ is a Markov chain on

$$\mathcal{X} = \{(z, w) \in \mathcal{Z} \times \mathcal{A}_{\texttt{start}} : w = z_j \text{ for some j}\}.$$

Henceforth, we will assume that $X_1, X_2, \ldots$ is stationary (that is, the initial distribution is the stationary distribution) and occasionally we will use notation (like $X_0$) that makes sense by thinking about $X_1, X_2, \ldots$ embedded in a two sided, stationary sequence $\ldots, X_{-1}, X_0, X_1, \ldots$.

The transition probability matrix is

$$\begin{aligned} p(x|\tilde{x}) &= \operatorname{Prob}(X_{k+1} = x | X_k = \tilde{x}) \\ &= \operatorname{Prob}(W_{k+1} = w | Z_{k+1} = z) \operatorname{Prob}(Z_{k+1} = z | Z_k = \tilde{z}, W_k = \tilde{w}) \\ &= p(w|z) p(z|\tilde{z}, \tilde{w}), \end{aligned}$$

for $x = (z, w)$, $\tilde{x} = (\tilde{z}, \tilde{w})$,

$$p(w|z) = (1 - q_{z_{\ell-1}}) \prod_{j=\ell}^{m-1} q_{z_j} \quad \text{and} \quad p(z|\tilde{z}, \tilde{w}) = \mathbb{1}\{z_{0:\tilde{\ell}-1} = \tilde{z}_{0:\tilde{\ell}-1}\} \prod_{j=\tilde{\ell}}^{m} p_{z_{j-1}}(z_j),$$

where we define $\ell = \ell(z, w)$ and $m = m(z)$ so that $w = z_\ell$ and $z_m \in \mathcal{T}$ with similar conventions for $\tilde{z}, \tilde{w}, \tilde{\ell}, \tilde{m}$. We define the empty product to be 1 and if $m < n \leq \tilde{m}$, we naturally take $\mathbb{1}\{z_{0:n} = \tilde{z}_{0:n}\} = 0$. These formulas are easy to derive using the description of a HISM and the alternative representation of the geometric distribution mentioned above.

### 4.2.2 Maximum likelihood estimation

If we observe a sequence $X_{1:n} = x_{1:n}$ from a HISM, then the above Markov formulation makes maximum likelihood estimation straightforward. Let $\hat{P}(\tilde{x}, x)$ denote the empirical distribution of consecutive pairs in $x_1, \ldots, x_n$, and let $\hat{P}(x) = \sum_{\tilde{x}} \hat{P}(\tilde{x}, x)$. The maximum likelihood estimates for $q_\alpha$ and $p_\alpha$ are

$$\hat{q}_\alpha = \frac{\hat{P}\{x : z_j = \alpha, \text{for some } j \geq \ell(z, w)\}}{\hat{P}\{x : z_j = \alpha, \text{for some } j \geq \ell(z, w) - 1\}}, \tag{1a}$$

$$\hat{p}_\alpha(\beta) = \frac{\hat{P}\{\tilde{x}, x : (z_{j-1}, z_j) = (\alpha, \beta), \text{for some } j \geq \tilde{\ell}(\tilde{z}, \tilde{w})\}}{\lambda_\alpha}, \tag{1b}$$

where $\lambda_\alpha$ is chosen so that $\hat{p}_\alpha$ sums to 1. Note that $\hat{p}_\alpha$ does not depend on $w$ and it only depends on $\tilde{x}$ through $\tilde{\ell}$ so it can be expressed in the same way but in terms of the slightly simpler empirical distribution $\hat{P}(\tilde{\ell}, z) = \sum_{\tilde{x}:\tilde{z}_{\tilde{\ell}}=\tilde{w}} \sum_w \hat{P}(\tilde{x}, x)$.

The maximum likelihood estimates are almost local, in the sense that $\hat{q}_\alpha$ and $\hat{p}_\alpha$ only depend on relative frequencies related to $\alpha$ and not related to all of the other symbols in the alphabet. They are not completely local, however, because they also depend on knowing whether the branch point at some time is above or below $\alpha$ (or whether it is at $\alpha$'s parent).

**Derivation of the MLE equations.** We first do some preliminary computations.

$$\begin{aligned}\log p(x|\tilde{x}) &= \log p(w|z) + \log p(z|\tilde{z}, \tilde{w}) \\ &= \log(1 - q_{z_{\ell-1}}) + \sum_{j=\ell}^{m-1} \log q_{z_j} + \sum_{j=\tilde{\ell}}^{m} \log p_{z_{j-1}}(z_j).\end{aligned}$$

We can safely ignore the $\mathbb{1}\{z_{0:\tilde{\ell}-1} = \tilde{z}_{0:\tilde{\ell}-1}\}$ term in $p(z|\tilde{z}, \tilde{w})$ because for maximum likelihood estimation we will never be considering pairs $x, \tilde{x}$ for which this term is zero. Differentiating with respect to $q_\alpha$ ($\alpha \in \mathcal{V}$) gives

$$\frac{\partial}{\partial q_\alpha} \log p(x|\tilde{x}) = \begin{cases} -\frac{1}{1-q_\alpha} & \text{if } \alpha = z_{\ell-1}, \\ \frac{1}{q_\alpha} & \text{if } \alpha = z_j \text{ for some } j \geq \ell, \\ 0 & \text{otherwise.} \end{cases}$$

Differentiating with respect to $p_\alpha(\beta)$ ($\alpha \in \mathcal{V} \cup \{\texttt{start}\}, \beta \in \mathcal{A}_\alpha$)

$$\frac{\partial}{\partial p_\alpha(\beta)} \log p(x|\tilde{x}) = \begin{cases} \frac{1}{p_\alpha(\beta)} & \text{if } (\alpha, \beta) = (z_{j-1}, z_j) \text{ for some } j \geq \tilde{\ell}, \\ 0 & \text{otherwise.} \end{cases}$$

The (conveniently normalized) log-likelihood is

$$\begin{aligned} L(\theta|x_{1:n}) &= \frac{1}{n-1} \log \operatorname{Prob}(X_{1:n} = x_{1:n}|\theta) \\ &= \frac{1}{n-1} \log \operatorname{Prob}(X_1 = x_1|\mu) + \frac{1}{n-1} \sum_{k=2}^{n} \log \operatorname{Prob}(X_k = x_k | X_{k-1} = x_{k-1}; \theta) \\ &= \frac{1}{n-1} \log \operatorname{Prob}(X_1 = x_1|\mu) + \sum_{\tilde{x}, x \in \mathcal{X}} \hat{P}(\tilde{x}, x) \log p(x|\tilde{x}; \theta), \end{aligned} \tag{2}$$

where $\theta$ denotes the $q_\alpha$'s and the $p_\alpha(\beta)$'s. The final equality comes from collecting terms of $(x_{k-1}, x_k) = (\tilde{x}, x)$.

Differentiating (2) w.r.t. $q_\alpha$, ignoring any dependence of the initial distribution $\mu$ on $\theta$, and making use of our preliminary computations gives

$$\frac{\partial}{\partial q_\alpha} L(\theta|x_{1:n}) = \sum_{\tilde{x}\in\mathcal{X}} \sum_{\substack{x\in\mathcal{X}:\\ \exists j\geq \ell, z_j=\alpha}} \hat{P}(\tilde{x},x)\frac{1}{q_\alpha} - \sum_{\tilde{x}\in\mathcal{X}} \sum_{\substack{x\in\mathcal{X}:\\ z_{\ell-1}=\alpha}} \hat{P}(\tilde{x},x)\frac{1}{1-q_\alpha}$$

$$= \hat{P}\{x : \exists j \geq \ell, z_j = \alpha\}\frac{1}{q_\alpha} - \hat{P}\{x : z_{\ell-1} = \alpha\}\frac{1}{1-q_\alpha},$$

which has the unique critical point $\hat{q}_\alpha$ given in the text. The second derivative is always negative, so $\hat{q}_\alpha$ is indeed the MLE for $q_\alpha$.

Using the same assumptions to differentiate (2) w.r.t. $p_\alpha(\beta)$ and using a Lagrangian multiplier $\lambda_\alpha$ to keep $p_\alpha$ normalized gives

$$\frac{\partial}{\partial p_\alpha(\beta)} \left[L(\theta|x_{1:n}) - \lambda_\alpha \textstyle\sum_{\beta'\in\mathcal{A}_\alpha} p_\alpha(\beta')\right] = \sum_{\substack{\tilde{x},x\in\mathcal{X}:\exists j\geq\tilde{\ell}\\ (z_{j-1},z_j)=(\alpha,\beta)}} \hat{P}(\tilde{x},x)\frac{1}{p_\alpha(\beta)} - \lambda_\alpha,$$

which also has the unique critical point $\hat{p}_\alpha$ given in the text. Since $L(\cdot|x_{1:n})$ is concave in $p_\alpha$, $\hat{p}_\alpha$ is the MLE for $p_\alpha$. □

## 4.3 Hidden HISMs

A HISM is just a strangely parameterized Markov chain, not on the alphabet $\mathcal{A}$ or on the terminals $\mathcal{T}$, but on the state space $\mathcal{X}$. For our purposes we will not get to observe the sequence of states $X_1, X_2, \ldots$, but rather a sequence $Y_1, Y_2, \ldots$, such that each $Y_k$ is a (possibly stochastic) function of $X_k$ and such that each $Y_k$ is conditionally independent from all the other $Y_j$'s given $X_k$. Such a sequence is called a hidden HISM (HHISM). Note that a HHISM is just a hidden Markov model (HMM).

For example, the $k$th terminal $T_k$ is a deterministic function of the $k$th state $X_k$, so the sequence of terminals is a HHISM and thus a HMM. (We have already noted that it is also the concatenation of a sequence of i.i.d. realizations from a PCFG.)

We will restrict ourselves to the situation where each $Y_k$ depends not on all of $X_k$ (and not on $k$), but only on the terminal $T_k$. Let $\mathcal{Y}$ be the common range of the $Y_k$'s. For each terminal $\alpha \in \mathcal{T} = \{1,\ldots,M\}$, we denote $p_\alpha(y) = \text{Prob}(Y_k = y|T_k = \alpha)$. (More precisely, $p_\alpha$ is the conditional density of $Y_k$ given $T_k = \alpha$ w.r.t. some common measure $\nu$ that does not depend on $\alpha$.) To sample from such a HHISM, we sample from the HISM as usual and then sample (independently) from $p_{T_k}$ to get $Y_k$.

### 4.3.1 Parameter estimation

Given a sequence of observations $Y_{1:n} = y_{1:n}$ from a HHISM with known structure, but unknown parameters, we can use the expectation-maximization (EM) algorithm [4] to estimate the parameters. The EM update equations for the $q_\alpha$'s ($\alpha \in \mathcal{V}$) and the $p_\alpha$'s ($\alpha \in \mathcal{V} \cup \{\texttt{start}\}$) are exactly like (1) except that we replace the fully observed empirical distribution $\hat{P}$ with the posterior empirical distribution

$$\begin{aligned}\hat{P}(\tilde{x}, x | Y_{1:n} = y_{1:n}; q_\alpha^{\text{old}}, p_\alpha^{\text{old}}) \\ = \frac{1}{n-1} \sum_{k=2}^{n} \text{Prob}(X_{k-1} = \tilde{x}, X_k = x | Y_{1:n} = y_{1:n}; q_\alpha^{\text{old}}, p_\alpha^{\text{old}}).\end{aligned} \tag{3}$$

The posterior distribution is calculated using the last iteration's parameters $q_\alpha^{\text{old}}$ and $p_\alpha^{\text{old}}$. Note that we do not actually need the full joint posterior $\hat{P}(\tilde{x}, x | \cdots)$ to use (3), but only $\hat{P}(x | \cdots)$ and $\hat{P}(\tilde{\ell}, z | \cdots)$.

If the $p_\alpha$'s ($\alpha \in \mathcal{T}$) also need to be estimated, then usually this can also be done using EM. The weights will be based on the individual posterior terms (usually just $\text{Prob}(X_k | Y_{1:n})$) inside the summation on the right side of (3). In our experiments below we initially fit (with EM) the $p_\alpha$'s corresponding to the terminals, but then we leave them fixed while estimating the nonterminals.

### 4.3.2 Computing the posterior

For parameter estimation with EM and also for interpretation we need to compute the empirical (marginal) posterior distribution as in (3). Since a HISM is a HMM, one way to do this is with dynamic programming using the standard forward-backward equations for a HMM (see [6] for a review). In many applications where $k$ is conceptualized as time, including the one here, it is more natural to think about computing the posterior distribution of $X_k$ given only the present and the past $Y_{1:k}$. This can be computed recursively using only the forward part of the forward-backward equations as follows.

Suppose we know $\kappa_{k-1}(x) = \text{Prob}(X_{k-1} = x | Y_{1:k-1} = y_{1:k-1})$ for each $x \in \mathcal{X}$. Since the Markov properties of a HHISM give the general factorization

$$\begin{aligned}\text{Prob}(X_{k-1}, X_k | Y_{1:k}) &\propto \text{Prob}(X_{k-1}, X_k, Y_k | Y_{1:k-1}) \\ &= \text{Prob}(Y_k | X_k) \, \text{Prob}(X_k | X_{k-1}) \, \text{Prob}(X_{k-1} | Y_{1:k-1}),\end{aligned}$$

where the proportionality constant does not depend on $X_{k-1}$ or $X_k$, we can compute

$$\kappa_k(\tilde{x}, x) = \text{Prob}(X_{k-1} = \tilde{x}, X_k = x | Y_{1:k} = y_{1:k}) = \frac{p_{z_m}(y_k) p(x|\tilde{x}) \kappa_{k-1}(\tilde{x})}{\sum_{\tilde{x}', x'} p_{z'_{m'}}(y_k) p(x'|\tilde{x}') \kappa_{k-1}(\tilde{x}')} \tag{4}$$

and

$$\kappa_k(x) = \text{Prob}(X_k = x | Y_{1:k} = y_{1:k}) = \sum_{\tilde{x}} \kappa_k(\tilde{x}, x). \tag{5}$$

The further factorizations of $p(x|\tilde{x})$ and the locality properties of the MLE equations probably allow for highly efficient computation of these quantities. See, for example, [12] where efficient inference algorithms are derived for hierarchical HMMs by expressing them as a special type of dynamic Bayesian network. We do not explore this possibility here.

These recursive relationships are intuitively appealing not only because they do not use future information, but also (and perhaps more importantly) because they do not require storing the entire observation history $y_{1:k-1}$. The only information needed from the past is stored in $\kappa_{k-1}$, the posterior distribution on the states from the last time step. Because of this, in our experiments below, we will only work with these history-only posteriors.

Note that an intermediate computation gives the posterior distribution $\kappa_k(\tilde{x}, x)$ on state pairs given the specific observation sequence $y_{1:k}$. Averaging this over time ($k$) gives an estimate of the joint distribution on state pairs.

$$P_{X_{t-1},X_t}(\tilde{x}, x) \approx \frac{1}{n}\sum_{k=1}^{n} \kappa_k(\tilde{x}, x). \tag{6}$$

A loose justification of this approximation can be found at the end of this section.

Equation 6 assumes that the observations actually come from the same HHISM that is used to compute the posterior. Typically, however, the observations $y_{1:n}$ will come from some other distribution $\mathbb{P}_{Y_{-\infty:\infty}}$, which we assume to be stationary and ergodic. In this case, averaging $\kappa_k(\tilde{x}, x)$ over time gives an estimate of the "world's distribution on state pairs" (to use the terminology from Chapters 2 and 3), that is

$$\begin{aligned}\mathbb{P}_{X_t,X_{t+1}}(\tilde{x}, x) &= \lim_{s\to\infty} \mathbb{E}_{Y_{-\infty:\infty}}\left[P_{X_{t-1},X_t|Y_{t-s:t+s}}(\tilde{x}, x|Y_{t-s:t+s})\right] \\ &\approx \lim_{s\to\infty} \mathbb{E}_{Y_{-\infty:\infty}}\left[P_{X_{t-1},X_t|Y_{t-s:t}}(\tilde{x}, x|Y_{t-s:t})\right] \approx \frac{1}{n}\sum_{k=1}^{n}\kappa_k(\tilde{x}, x),\end{aligned} \tag{7}$$

where the first equality is a definition (see the remark below), where $t$ does not matter because everything is stationary and where the final approximation is just like (6) and is loosely justified below. The first approximation can be bad and would certainly be better if we included more future information. It is important to note, however, that this first approximation does not introduce artifacts, it merely reduces the power of the empirical posterior distribution for detecting problems with the model.

Comparing (6) and (7) and following the same reasoning from Chapter 2, if (6) is not satisfied, then this is evidence that the current model differs from the true data distribution and we may be able to improve the model in light of this evidence.

**Technical remark on (7).** The subscript $Y_{t-s:t+s}$ refers to the observation sequence of the HHISM under the model and specifies the regular conditional distribution $P_{X_{t-1},X_t|Y_{t-s:t+s}}(\tilde{x}, x|\cdot)$ which is just a function for fixed $(\tilde{x}, x)$. If $U_{-\infty:\infty}$ is a random process on $\mathcal{Y}_{-\infty:\infty}$, then we can evaluate this function on any segment of $U_{-\infty:\infty}$, say $P_{X_{t-1},X_t|Y_{t-s:t+s}}(\tilde{x}, x|U_{t-s:t+s})$, and then take the expectation

$$E_{U_{-\infty:\infty}}\left[P_{X_{t-1},X_t|Y_{t-s:t+s}}(\tilde{x}, x|U_{t-s:t+s})\right].$$

Since a HHISM is a finite state, aperiodic HMM, $P_{X_{t-1},X_t|Y_{t-s:t+s}}(\tilde{x}, x|\cdot)$ does not depend much on the distant future or the distant past and we can expect that the limit of these expectations exists as $s \to \infty$ under certain regularity conditions like $P_{U_{t-s:t+s}} \ll P_{Y_{t-s:t+s}}$ for all $s$ and $t$ which ensure that everything is well defined. For the case where $U_{-\infty:\infty}$ has distribution $\mathbb{P}_{Y_{-\infty:\infty}}$, this is what we mean by the notation in (7).

**Heuristic justification of (6) and (7).** Continuing the above technical remark, if $U_{-\infty:\infty}$

is stationary and ergodic with $P_{U_{-t:0}} \ll P_{Y_{-t:0}}$ for all $t$, we claim (but do not rigorously prove) that

$$f(U_{-\infty:\infty}) = \lim_{t\to\infty} f_t(U_{-\infty:\infty}) = \lim_{t\to\infty} P_{X_{-1},X_0|Y_{-t:0}}(\tilde{x},x|U_{-t:0})$$

exists a.s. and in expectation, where the two equalities are just definitions. The idea is the same: $P_{X_{t-1},X_t|Y_{-t:0}}(\tilde{x},x|\cdot)$ does not depend much on the distant past.

Assuming that this is valid, we can compute

$$\begin{aligned}\lim_{T\to\infty}\sum_{t=0}^{T-1} P_{X_{t-1},X_t|Y_{0:t}}(\tilde{x},x|U_{0:t}) &\overset{(a)}{=} \lim_{T\to\infty}\sum_{t=0}^{T-1} P_{X_{-1},X_0|Y_{-t:0}}(\tilde{x},x|U_{0:t})\\ &\overset{(b)}{=} \lim_{T\to\infty}\sum_{t=0}^{T-1} f_t(\varphi^t\circ U_{-\infty:\infty}) \overset{(c)}{=} E_{U_{-\infty:\infty}}[f(U_{-\infty:\infty})],\end{aligned}$$

where all of the equalities hold a.s. and in expectation. $(a)$ comes from the stationarity of the $(X_t,Y_t)$'s; $(b)$ is just a definition, where $\varphi$ is the shift; and $(c)$ is a well known generalization of the ergodic theorem for bounded r.v.'s (see Durrett [5], Exercise 2.2, p343).

When $U_{-\infty:\infty}$ has distribution $P_{Y_{-\infty:\infty}}$, then $f(\cdot) = P_{X_{-1},X_0|Y_{-\infty:0}}(\tilde{x},x|\cdot)$, so $Ef = P_{X_{-1},X_0}(\tilde{x},x)$. Approximating the limit with large $T$ (or $n$ in the text) gives (6). When $U_{-\infty:\infty}$ has distribution $\mathbb{P}_{Y_{-\infty:\infty}}$, the large $T$ approximation is exactly (7). $\Box$

### 4.3.3 Temporal suspicious coincidences

Let

$$\hat{\mathbb{P}}_{X_{t-1},X_t}(\tilde{x},x) = \frac{1}{n}\sum_{k=1}^{n}\kappa_k(\tilde{x},x) \tag{8}$$

denote the average (over time) of the empirical joint posterior distribution on state pairs given the past. In the last section we noted that

$$\hat{\mathbb{P}}_{X_{t-1},X_t} \not\approx P_{X_{t-1},X_t}$$

suggests that there is a deficiency in the current model. There are many ways that this approximation could fail. In Chapter 2 we noted that a particular type of failure, namely, a suspicious coincidence, might be a useful class of failures to look for. It turns out that $P_{X_{t-1},X_t}$ contains certain independence assumptions that lead to a large class of possible *temporal* suspicious coincidences.

The root nodes $S_1, S_2, \ldots,$ are i.i.d. $p_{\mathtt{start}}$ in a HHISM. In particular,

$$P_{S_{K-1},S_K}(\tilde{\alpha},\alpha) = P_{S_{K-1}}(\tilde{\alpha})P_{S_K}(\alpha) = p_{\mathtt{start}}(\tilde{\alpha})p_{\mathtt{start}}(\alpha)$$

for $\tilde{\alpha},\alpha \in \mathcal{A}_{\mathtt{start}}$. Note, however, that the "time scale" $(K-1,K)$ for the root nodes $S_1, S_2, \ldots$ is different from the "time scale" $(t-1,t)$ of the state sequence $X_1, X_2, \ldots$. This is because each $S_k$ is typically the root for a (possibly long) sequence of (consecutive) terminals $T_i,\ldots,T_j$,which is easy to see in Figure 4.1. Since we demarcate time with the terminals, the temporal sequence of roots $Z_{11}, Z_{21}, Z_{31}, \ldots$ relative to the terminals is typically not i.i.d. (It is a HMM.) So, for example, $S_k = Z_{i1} = \cdots = Z_{j1}$. Nevertheless, because the states space $\mathcal{X}$ specifies the switch point, we can easily derive $P_{S_{K-1},S_K}$ from $P_{X_{t-1},X_t}$.

Measuring time according to the terminals, the root node $Z_{k1}$ is chosen i.i.d. from $p_{\texttt{start}}$ exactly when the previous switch point was the previous root node, that is, $Z_{(k-1)1} = W_{k-1}$. One way to see this is to note that $Z_{(k-1)1} = W_{k-1}$ means that $T_{k-1}$ and $T_k$ do not belong to the same tree. So $Z_{(k-1)1}$ corresponds to the root, say $S_{K-1}$, of some tree and $Z_{k1}$ corresponds to the root $S_K$ of the next tree. This gives the formulas

$$\begin{aligned}P_{S_{K-1},S_K}(\tilde{\alpha},\alpha) &= \mathrm{Prob}(Z_{(k-1)1} = \tilde{\alpha}, Z_{k1} = \alpha | Z_{(k-1)1} = W_{k-1}) \\ &= \frac{\sum_{\tilde{x}:\tilde{z}_1=\tilde{w}=\tilde{\alpha}} \sum_{x:w=\alpha} P_{X_{t-1},X_t}(\tilde{x},x)}{\sum_{\tilde{x}':\tilde{z}'_1=\tilde{w}'} \sum_{x'} P_{X_{t-1},X_t}(\tilde{x}',x')},\end{aligned}$$

$$P_{S_{K-1}}(\tilde{\alpha}) = \sum_{\alpha} P_{S_{K-1},S_K}(\tilde{\alpha},\alpha) \quad \text{and} \quad P_{S_K}(\alpha) = \sum_{\tilde{\alpha}} P_{S_{K-1},S_K}(\tilde{\alpha},\alpha).$$

Assuming stationarity, these do not depend on the specific value of $K$, so $P_{S_{K-1}} = P_{S_K} = P_S = p_{\texttt{start}}$.

We can similarly define $\mathbb{P}_{S_{K-1},S_K}$ and $\hat{\mathbb{P}}_{S_{K-1},S_K}$ by replacing $P_{X_{t-1},X_t}$ with $\mathbb{P}_{X_{t-1},X_t}$, defined in (7), and with $\hat{\mathbb{P}}_{X_{t-1},X_t}$, defined in (8), respectively. One type of temporal suspicious coincidence, then, is when

$$\hat{\mathbb{P}}_{S_{K-1},S_K}(A \times B) \gg \hat{\mathbb{P}}_{S_{K-1}}(A)\hat{\mathbb{P}}_{S_K}(B).$$

This assumes that the empirical marginals match the model's marginals $\hat{\mathbb{P}}_S \approx P_S = p_{\texttt{start}}$. Usually, EM will ensure the validity of this approximation.

Once a suspicious coincidence like this has been detected, it is not clear exactly how to improve the model and remain within the HHISM class. A slightly simpler type of suspicious coincidence is one of the form

$$\hat{\mathbb{P}}_{S_{K-1},S_K}(A \times A) \gg \hat{\mathbb{P}}_{S_{K-1}}(A)\hat{\mathbb{P}}_{S_K}(A). \tag{9}$$

In this case, there is a natural way to modify the current model and incorporate the temporal dependency evidenced by the suspicious coincidence. In particular, a new nonterminal $\alpha = N + 1$ with children $\mathcal{A}_\alpha = A$ can be appended to the model and to the set of possible roots $\mathcal{A}_{\texttt{start}}$. If the other model parameters remain relatively stable, then this new node will perturb the model and create temporal dependencies among the nodes in $A$. Note, of course, that the nodes in $A$ do not now have dependencies when they are the root nodes. Rather, the new model allows the elements of $A$ to appear in non-root positions in the state space and thus to have dependencies by virtue of a common parent $\alpha$.

### 4.3.4 HHISMs and invariant feature detectors

The principle of temporal persistence suggests that a HHISM fit to natural image sequences might capture certain types of invariances. The sequence of observations $Y_1, Y_2, \ldots,$ represents the image sequence, perhaps consecutive frames of natural video. The terminals $\mathcal{T}$ represent the states in a model for single images, perhaps something analogous to the models in Chapters 2 and 3. Without any nonterminals, a HISM is just an i.i.d. sequence of terminals from $p_{\texttt{start}}$, so a HHISM is just an i.i.d. mixture model. In this case, each frame of the image sequence would be modeled independently.

If the HHISM has nonterminals, however, then these introduce dependencies into the sequence of observations. Since natural video contains dependencies across frames, a HHISM fit to natural video will presumably have many nonterminals to capture these dependencies. Typically, in a HISM, when a nonterminal $\alpha$ appears in $Z_k$, then it will appear in $Z_{k+1}$ and vice-versa. The presence of $\alpha$ is thus temporally stable and the principle of temporal stability suggests that (the presence of) $\alpha$ might capture some invariance.

One way to fit a HHISM is to determine the structure *a priori* and then use a learning method like EM to fit the parameters. Another way, more in the spirit of Chapter 2, is to incrementally add nonterminals to capture newly identified dependencies (and use EM to fit the parameters). We focus solely on this latter method. In particular, we will use the ideas from the previous section to detect and incorporate temporal suspicious coincidences into a HHISM fit to sequences of natural image patches.

## 4.4 An image patch model

In Section 4.5 we experiment with fitting HHISMs to sequences of 4×4 binary image patches. For the conditional distribution on the data given the terminals, we use the same single image patch model described in Chapter 3. The only difference is that now we are using 4×4 patches with $M$ possible terminal states. $M-1$ of these states can be thought of as an ideal patch with some small probability $\alpha$ of i.i.d. pixel noise (flips). The last state corresponds to "no ideal patch" and each pixel is i.i.d. Bernoulli($\beta$), where $\beta \approx 0.5$.

These $M$ terminal states are shown in Figures 4.2 and 4.3 for two different data sets that we experiment with below. They were learned from the data just like the ideal patches in Chapter 3 except all local maximum were kept, not just those whose empirical probability exceeded $2^{-16}$. For each data set, the prior probability distribution on the $M$ states and the two noise parameters $\alpha$ and $\beta$ were fit with EM. These are also shown in Figures 4.2 and 4.3. The respective data models (that is, $\alpha$ and $\beta$) were held constant throughout the experiments.

Our initial experiments with this terminal-only model were plagued by a persistent problem. The most striking suspicious coincidences were always each terminal transitioning to itself, as would be expected of course. So the model immediately became the $M$ terminals along with $M$ corresponding nonterminals, where each nonterminal had a single (unique) terminal in its set of children (that is, $\mathcal{A}_\alpha$ was a single terminal for each $\alpha \in \mathcal{V}$). This is a very sensible model, but we had a lot of difficultly fitting it with EM. Finding a good initialization was particularly difficult.

An easy remedy was to remove each terminal from the set of possible roots $\mathcal{A}_{\texttt{start}}$ once it is connected to its corresponding nonterminal (or equivalently, to set $p_{\texttt{start}}(\alpha) = 0$ for $\alpha \in \mathcal{T}$). There are several other equivalent ways to think about this.

The first is that it is just a modeling assumption. Instead of the initial HISM model being only $M$ terminals (and therefore i.i.d.), the initial model has $M$ terminals $\mathcal{T} = \{1, \ldots, M\}$ and $M$ nonterminals $\mathcal{V} = \{M+1, \ldots, 2M\}$ with $\mathcal{A}_\alpha = \{\alpha - M\}$ and $\mathcal{A}_{\texttt{start}} = \mathcal{V}$. Effectively, the initial model builds in the assumption that terminals should persist somewhat over time.

Another way to think about it is to change the formulation of a HISM so that each terminal $\alpha$ has its own $q_\alpha$. Once selected, it is repeated for $N$ times where $N$ is random with distribution geometric($q_\alpha$). So, in the case of a HHISM, each terminal does not generate a single observation from its data distribution $p_\alpha$, but rather $N$ i.i.d. samples from $p_\alpha$. Compared to the previous formulation, this effectively merges each terminal with its corresponding nonterminal into a single node. The only downside of this formulation is that a HHISM with only terminals is no longer an i.i.d. mixture model (unless all of the $q_\alpha$'s are 1), and there is something appealing about beginning with i.i.d. and then incorporating dependencies.

Either way, fitting this model with EM worked well (and there are half as many states, which is nice computationally). To summarize:

1. The empirical distribution of $4 \times 4$ binary image patches was used to generate $M-1$ sparsely distributed ideal patches.

2. These ideal patches along with a "no patch" state define an i.i.d. hidden mixture model on observed image patches with two parameters $\alpha$ and $\beta$. These two parameters and the $M$ prior probabilities ($M-1$ parameters) were fit to the empirical distribution of image patches using EM.

3. The $M$ states in the mixture model were labeled as terminals in a HHISM and each terminal was connected to a unique nonterminal. Only the $M$ nonterminals were included in $\mathcal{A}_{\texttt{start}}$. The $M$ $q_\alpha$'s and the $M-1$ parameters in $p_{\texttt{start}}$ were fit on image patch sequences using EM based on the average empirical history-only posterior distribution. The data model parameters $\alpha$ and $\beta$ were held constant.

4. This initial HHISM was the starting point for the experiments below.

Just to reiterate, creating a unique nonterminal for each terminal is consistent with the idea of model building using suspicious coincidences. Self-transitions among terminals are the most suspicious coincidences in the data. (In fact, they are exactly the suspicious coincidences selected by the greedy search algorithm used at later stages and described below.) What is perhaps not consistent is then removing the terminal from $\mathcal{A}_{\texttt{start}}$ as this is not a minor perturbation of the old model.

## 4.5 Experiments

### 4.5.1 Data sets

We experiment with two fundamentally different types of data, simulated video and actual video. Each is based on the concatenation of about 10 000 different subsequences of 1 000 frames each, for a total of about 10 million image patches.

In the simulated video each subsequence of 1000 image patches was created by slowly moving a $4 \times 4$-pixel window (of fixed orientation and scale) over an image. About 200 different images were used and 50 different subsequences were taken from each image. The images came from a large collection courtesy of Hans van Hateren and described in [14]. The gray-scale images were first converted to log-images, then reduced in size to $512 \times 768$ pixels (by averaging disjoint $2 \times 2$-pixel neighborhoods) and then converted to binary by thresholding each image at its median intensity value. This is the same data set used in Chapter 3, but with different preprocessing, so examples of the content of the images can be seen in Figure 3.1. The $M - 1 = 67$ ideal patches derived from this data set and used as the terminals were only based on spatial information and are shown in Figure 4.2.

The window movement process was created with an *ad hoc* smoothed 2D random walk in the image plane that we do not describe in detail here. When the random walk reached the edge of the image, it jumped to the other side. This happened less than 0.3% of the time. Ignoring these large jumps, the distribution on the absolute change in location (2D Euclidean distance in pixels) for consecutive time steps had the following empirical properties: mean = 1.4, median = 1.4, stddev = 0.8, 12% no change, 96% change of 3 pixels or less. An example of 100 consecutive frames produced by this process is shown in Figure 4.4. The specifics of the process undoubtedly affect the results, but we have not experimented with different methods of producing simulated video.

In the other type of data set each subsequence of 1000 image patches was created by taking the sequence of image patches in a fixed $4 \times 4$-pixel window of actual video. 12 different videos were used and 750 different subsequences were taken from each video. The videos were also courtesy of Hans van Hateren and are described in [13]. The videos are all taken from the window of a moving vehicle. Although, the camera position is not the same in each video, all movement directions are certainly not equally represented in the videos. There are also periods with a lot of glare that saturates the images.

Every two (disjoint) frames were time averaged (as recommended) so that the resulting video rate was 25 frames/second with 4800 frames/video and $128 \times 128$ pixels/frame. Each frame was then independently converted to binary by thresholding with its median intensity. An example of 100 consecutive frames of a $4 \times 4$-pixel patch is shown in Figure 4.5.

The $M - 1 = 89$ ideal patches derived from this data set and used as the terminals were only based on spatial information and are shown in Figure 4.3. Note that many of them are just noise and could be reasonably pruned based on their incredibly low prior probability, but this would just complicate things. (The $2^{-16}$ pruning seems slightly too severe.)

### 4.5.2 Finding suspicious coincidences

Once the initial HHISM was trained, the average empirical joint posterior distribution on states $\hat{\mathbb{P}}_{X_{t-1},X_t}$ was used to compute the average empirical joint posterior distribution on root nodes (at switch times) $\hat{\mathbb{P}}_{S_{K-1},S_K}$ as described in Section 4.3.3. In the initial HHISM this is a $M \times M$ matrix.

We want to find suspicious coincidences of the form

$$\hat{\mathbb{P}}_{S_{K-1},S_K}(A \times A) \gg \hat{\mathbb{P}}_{S_{K-1}}(A)\hat{\mathbb{P}}_{S_K}(A),$$

so each of the candidate suspicious coincidences is identified with a subset $A$ of the possible root nodes. Recall that in the initial model, each root node is one of the $M$ nonterminals, each of which is uniquely associated with one of the $M$ terminals. So we can identify each candidate suspicious coincidence with a subset of the $M$ terminals. There are thus on the order of $2^M$ different possible suspicious coincidences of the form that we are considering. (Recall that $M$ is either 68 or 90 in our two data sets, so this is a large number.)

One way to begin to visualize the results is with the following matrix, whose positive entries are shown on the left in Figures 4.6 (simulated video) and 4.7 (actual video):

$$C[i,j] = \log_2 \frac{\hat{\mathbb{P}}_{S_{K-1},S_K}(\alpha_i, \alpha_j)}{\hat{\mathbb{P}}_{S_{K-1}}(\alpha_i)\hat{\mathbb{P}}_{S_K}(\alpha_j)}.$$

The reason this is useful is that if $A \times A$ is a suspicious coincidence for which it is appropriate to model with a (single) new nonterminal, we might expect that all pairs $\alpha_i \times \alpha_j$, for $\alpha_i, \alpha_j \in A$, are also suspicious coincidences (though not necessarily of the form that we are considering). So pairs for which $C[i,j] > 0$ are candidates to be included in the same suspicious coincidence $A \times A$. The larger the value of $C$, the stronger the departure from independence.

Perhaps this is better visualized with the symmetric matrix

$$G[i,j] = \mathbb{1}\{C[i,j] > 0\}\mathbb{1}\{C[j,i] > 0\},$$

which is shown on the right in Figures 4.6 and 4.7. Ignoring the diagonal terms, this can be thought of as a connectivity matrix between the $M$ states in the model, where two states are connected exactly when each is more likely to switch to the other than would be predicted by independence.

Any clique of the graph defined by $G$ has the property that each of its members is more likely to switch to each of its other members than would be predicted by independence. Except for any problems introduced by the caveat that we ignored self-transitions, each clique of the graph defined by $G$ is a suspicious coincidence. All of the maximal cliques in this graph are represented in Figures 4.8–4.11 (simulated video) and Figures 4.12–4.15 (actual video). So any subset from these figures is (up to the caveat) a suspicious coincidence.

The cliques are far from random, but show a lot of interesting patterns that are consistent with the intuition that temporal suspicious coincidences in a HHISM should discover certain natural invariances. Naturally, collections of patches that might best be called translation-invariant are strongly prevalent, but so are collections that appear phase-invariant (which is just translation-invariance for a texture), contrast-invariant (which is just phase-invariant for a texture with features about the same size as the image patch), rotation-invariant and scale-invariant. These latter two are somewhat unintuitive in the simulated video but can perhaps be explained by a fixed window moving linearly over a curved edge or over a feature that is changing in scale (like a tapering tree-branch or something with perspective distortion), respectively. The results for the simulated video are overall more symmetric and less noisy than the results for the actual video.

Just as a sanity check, we generated a random graph with the same number of connections (as the simulated video). It only had cliques of size two, three and four and as a whole, these cliques did not tend to show any easily interpretable invariances (a few did, of course). We also experimented with various clustering algorithms based on using the matrix $C \vee 0$ as a similarity measure, for example, hierarchical clustering. Each of these tended to cluster perceptually similar patches closer together, as one might expect.

### 4.5.3 Selecting suspicious coincidences

There were 2 423 cliques and 123 maximal cliques in the simulated video data set (and similar magnitudes for the actual video). The sparse coding principles mentioned in Chapters 2 and 3 suggest that we do not want to introduce each one as a new node into the model. And, even if we did, this would drastically increase the computational demands.

One way to introduce sparsity is to partition $\mathcal{A}_{\texttt{start}}$ into disjoint subsets $A_0, \ldots, A_n$, where each $A_i \times A_i$ ($i \geq 1$) is a suspicious coincidence (and $A_0$ is a possibly empty subset of unused elements), and only add these suspicious coincidences to the model. Of course, we want the subsets $A_i$ to be as suspicious as possible. It also seems reasonable to prefer smaller subsets as larger ones could be built out of these smaller ones later in the hierarchy, but not vice-versa.

We will describe a (doubly) greedy heuristic that creates such a partition. The algorithm describes how to choose the next subset $A_i$ given the previously chosen subsets $A_1, \ldots, A_{i-1}$. (Recall that $A_0$ is the special set of unused elements, which is only defined at the end of the procedure.) $A_i$ only depends on the previously chosen subsets through the subset $D_i = \mathcal{A}_{\texttt{start}} \setminus \bigcup_{j=1}^{i-1} A_j$, which is just the collection of elements that have not yet been chosen. Let $D_1 = \mathcal{A}_{\texttt{start}} = \mathcal{V}$. To create $A_i \subseteq D_i$ we first create subsets $B_i^\alpha \subseteq D_i$ for each nonterminal $\alpha \in D_i$, where each $B_i^\alpha$ is created independently using the following greedy algorithm:

- initialize $B_i^\alpha = \{\alpha\}$, $\alpha \in D_i$
- while any $\beta \in D_i \setminus B_i^\alpha$ has

$$\frac{\hat{\mathbb{P}}_{S_{K-1},S_K}(B_i^\alpha \cup \beta \times B_i^\alpha \cup \beta)}{\hat{\mathbb{P}}_{S_{K-1}}(B_i^\alpha \cup \beta)\hat{\mathbb{P}}_{S_K}(B_i^\alpha \cup \beta)} > \frac{\hat{\mathbb{P}}_{S_{K-1},S_K}(B_i^\alpha \times B_i^\alpha)}{\hat{\mathbb{P}}_{S_{K-1}}(B_i^\alpha)\hat{\mathbb{P}}_{S_K}(B_i^\alpha)} \tag{10}$$

  set $B_i^\alpha = B_i^\alpha \cup \beta^*$, where $\beta^*$ maximizes the left side of (10) over $D_i \setminus B_i^\alpha$

- end

Once the $B_i^\alpha$'s are created, create $A_i$ with the following greedy step:

$$A_i = \operatorname*{arg\,max}_{B_i^\alpha : \alpha \in D_i} \frac{\hat{\mathbb{P}}_{S_{K-1},S_K}(B_i^\alpha \times B_i^\alpha)}{\hat{\mathbb{P}}_{S_{K-1}}(B_i^\alpha)\hat{\mathbb{P}}_{S_K}(B_i^\alpha)}.$$

Now remove the nodes in $A_i$ from consideration by taking $D_{i+1} = D_i \setminus A_i$. Repeat until all the nonterminals are gone (in which case $A_0 = \emptyset$) or until some $A_k \times A_k$ is not a suspicious coincidence (in which case $A_0 = D_k$).

Figures 4.16 (simulated video) and 4.18 (actual video) show those elements of the resulting partition that contain two or more nonterminals. Most of the suspicious coincidences contain only two nonterminals and nearly all can be characterized by translation (or phase) invariance. Figure 4.17 shows the partition on a different simulated video data set (the same images but without preprocessing – accidentally). We include it because it demonstrates that the greedy heuristic can create subsets with more than two nonterminals and because it gives an example of using the process on a data set that the terminals were not directly derived from.

### 4.5.4 Building higher levels

Once the suspicious coincidence $A \times A$ has been selected for inclusion into the model, the model is updated with a new nonterminal $\alpha$ that has children $\mathcal{A}_\alpha = A$ and $\alpha$ is added to $\mathcal{A}_{\texttt{start}}$. The new model can be fit in the same manner using EM.

We have not explored how to initialize the new model. Recall that we avoided this problem initially by removing terminals from $\mathcal{A}_{\texttt{start}}$ once they were in $\mathcal{A}_\beta$ for some nonterminal $\beta$. We could continue this, but then parts lower in the hierarchy would not be reusable, at least not using our method of finding suspicious coincidences by looking only in the root nodes. Furthermore, one of the important aspects of a parts-based model is that the parts are allowed to happen independently, without having to be composed into a larger whole.

We used the suspicious coincidences (shown in Figures 4.16 and 4.18) from the greedy partition algorithm to update the model. For the simulated video, this meant the addition of 25 nonterminals, each of which had two unique elements of the original nonterminal layer in its respective children set ($\mathcal{A}_\alpha$). For the actual video, this meant the addition of 15 nonterminals, 13 of which had two children and 2 of which had three children.

The partition algorithm sometimes returns a single node. We ignored these suspicious coincidences, since they can presumably be captured by lowering that node's $q_\alpha$, which we lowered by 10%. For each new node $\alpha$, we set $p_{\texttt{start}}(\alpha)$ equal to 90% of the total probability (in $p_{\texttt{start}}$) of its children and reduced each of its children by 90%. We set each $p_\alpha$ to the uniform distribution over its children and we set $q_\alpha$ equal to 90% of the mean of the $q_\beta$'s of its children. We did not change any of the other parameters in the model. Then we fit the model using EM in the same manner as before.

After training, we looked for suspicious coincidences in the same way. Figures 4.19 (simulated video) and 4.20 (actual video) show the next level of suspicious coincidences that were found with the greedy selection algorithm. Some of the suspicious coincidences involve only nodes in the original nonterminal layer. Others involve the new nonterminals. The new nonterminals are visualized with the collection of 4×4-binary patches that correspond to all of their children which are then connected with lines. These could be added to the model as before to create a new layer of nonterminals and so on, but we have not tried it yet.

In many ways, iterating the model at this stage is not the interesting thing to do. Ideally, after building a level of invariance using temporal suspicious coincidences, now we would build a new level of selectivity *using the invariant features* in a manner analogous to that of Chapter 3. We do not explore this here, although we anticipate some of the issues that are likely to arise in Chapter 5.

## 4.6 Related work

The idea of using temporal stability to learn invariant representations has been explored by several authors. We mention a few important examples. A more thorough collection of early references can be found in [17]. Földiák (1991) [8] described a neural network model that included a temporal Hebbian-like synapse. This tended to create units with activity that slowly varied over time. Wallis and Rolls (1997) [15] embedded a temporal coherence criterion within a biologically inspired hierarchical neural network model. They also experimented with natural image sequences (but not natural video).

Becker (1993) [1] presented an algorithm based on maximizing the mutual information between the vector of hidden states at two successive time steps. This tends to create units that are temporally stable over short sequences and that have high information content over long sequences. Although the details and applications are quite different, the idea of using an information theoretic criterion is very close in spirit to the underlying motivation for our work here.

In more recent work, Wiskott and Sejnowski (2002) [17] introduced the slow feature analysis (SFA) algorithm which explicitly maximizes a temporal stability criterion to create a collection of whitened (and potentially nonlinear) units whose activities are slowly varying over time. In particular, SFA searches over a predetermined, finite-dimensional vector space of stimulus-response functions to find those functions whose temporal derivatives have smallest variance (subject to the whitened constraint). They also experiment with hierarchical collections of SFA units.

Berkes and Wiskott (2002,2003) [2, 3] experiment with SFA over quadratic functions using simulated natural images sequences, generated in a similar manner to the simulated sequences used in this paper. They find that the quadratic SFA criterion leads to a variety of different types of low-level invariances. Still restricted to quadratic nonlinearities, Hashimoto (2003) [9] modifies the SFA criterion to emphasize rarely changing signals instead of slowly changing signals by creating a cost function that seeks to make the distribution of temporal derivatives sparse. He experiments with natural video and finds receptive fields with invariance properties qualitatively similar to those of complex cells in V1. Because of the tight connection between sparsity and information theory, there are some strong underlying similarities to our work.

One interesting aspect of most of these models is that a temporal stability criterion leads to both invariance and selectivity. This is in contrast to (and arguably more elegant than) our approach, which creates selectivity through spatial dependencies and invariance through temporal dependencies. A clear example that temporal stability can create selectivity is [10, 11], where Gabor-like linear receptive fields are produced by a temporal stability criterion applied to natural video. Since the units are linear, they cannot have classical invariance properties and can only show selectivity.

There is a variety of other work that is similar to ours in that it tries to model temporal sequences of images, but that does not directly consider the idea of invariance. Typically, the focus of this work is to explicitly model the temporal dynamics, which we did not do. Again, we only mention a few examples. Hierarchical HMMs [7] are more general than HHISMs. They have been applied to unsupervised learning in a variety of contexts, including natural video, for example [18], although the details and applications are quite different from the work here. The computer graphics literature also contains several related lines of work. See, for example, [16], where a generative model for texture motion is trained on natural video and then used for texture motion synthesis. It would be interesting to see if the texture motion work could be modified to simultaneously learn temporal dynamics and static invariance.

## 4.7 Discussion

We developed a class of models called hidden hierarchical independent switching models (HHISMs) that happen to be peculiarly parameterized HMMs. The parameterization is designed to create temporally stable regions in the state space. Temporal stability is a common method for creating invariant representations; and in our experiments with natural image sequences, these temporally stable regions do indeed correspond to certain natural invariants.

Another important aspect of HHISMs is that they lend themselves to the general model building framework of Chapter 2. In particular, temporal suspicious coincidences can be detected and incorporated into the model in order to better model temporal dependencies in the data. Invariance is essentially a by-product of creating a better statistical model. This parallels the ideas in Chapter 3 where spatial suspicious coincidences were used to detect spatial dependencies and selectivity was the by-product of a better model.

Naturally, this brings up the following question: In what sort of models would the incorporation of spatio-temporal dependencies have both selectivity and invariance as a (simultaneous) by-product? Presumably spatio-temporal suspicious coincidences would be a useful method for detecting these dependencies.

Whether it be simultaneously, in alternating stages or in some other manner, combining invariance and selectivity into a hierarchical framework brings up certain potential problems. In particular, invariance at one stage hides information that might be important for selectivity at a later stage. We discuss this in more detail in the next chapter.

## Bibliography

[1] Suzanna Becker. Learning to categorize objects using temporal coherence. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 361–368. Morgan Kaufmann Publishers Inc., 1993.

[2] Pietro Berkes and Laurenz Wiskott. Applying slow feature analysis to image sequences yields a rich repertoire of complex cell properties. In José R. Dorronsoro, editor, *Proc. Intl. Conf. on Artificial Neural Networks - ICANN'02*, Lecture Notes in Computer Science, pages 81–86. Springer, 2002.

[3] Pietro Berkes and Laurenz Wiskott. Slow feature analysis yields a rich repertoire of complex-cell properties. Cognitive Sciences EPrint Archive (CogPrints) 2804, http://cogprints.ecs.soton.ac.uk/archive/00002804/ (12/08/2003), February 2003.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society B*, 46:1–38, 1977.

[5] Richard Durrett. *Probability: Theory and Examples.* Duxbury, Belmont, 1996.

[6] Yariv Ephraim and Neri Merhav. Hidden Markov processes. *IEEE Transactions on Information Theory*, 48(6):1518–1569, June 2002.

[7] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32:41–62, 1998.

[8] Peter Földiák. Learning invariance from transformation sequences. *Neural Computation*, 3(2):194–200, 1991.

[9] Wakako Hashimoto. Quadratic forms in natural images. *Network: Computation in Neural Systems*, 14:765–788, 2003.

[10] J. Hurri and A. Hyvärinen. Simple-cell-like receptive fields maximize temporal coherence in natural video. *Neural Computation*, 15(3):663–691, 2003.

[11] A. Hyvärinen, J. Hurri, and J. Väyrynen. Bubbles: a unifying framework for low-level statistical properties of natural image sequences. *Journal of the Optical Society of America A*, 20(7):1237–1252, 2003.

[12] K. P. Murphy and M. A. Paskin. Linear-time inference in hierarchical HMMs. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 833–840, Cambridge, MA, 2002. MIT Press.

[13] J. H. van Hateren and D. L. Ruderman. Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex. *Proceedings of the Royal Society of London B*, 265:2315–2320, 1998. `http://hlab.phys.rug.nl/archive.html`.

[14] J. H. van Hateren and A. van der Schaaf. Independent component filters of natural images compared with simple cells in primary visual cortex. *Proceedings of the Royal Society of London B*, 265:359–366, 1998. `http://hlab.phys.rug.nl/archive.html`.

[15] Guy Wallis and Edmund T. Rolls. Invariant object recognition in the visual system. *Progress in Neurobiology*, 51(2):167–194, February 1997.

[16] Y.Z. Wang and S.C. Zhu. Analysis and synthesis of textured motion: Particles and waves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1348–1363, October 2004.

[17] Laurenz Wiskott and Terrence J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14:715–770, 2002.

[18] Lexing Xie, Shih-Fu Chang, Ajay Divakaran, and Huifang Sun. Feature selection for unsupervised discovery of statistical temporal structures in video. In *IEEE International Conference on Image Processing (ICIP)*, Barcelona, Spain, September 2003.

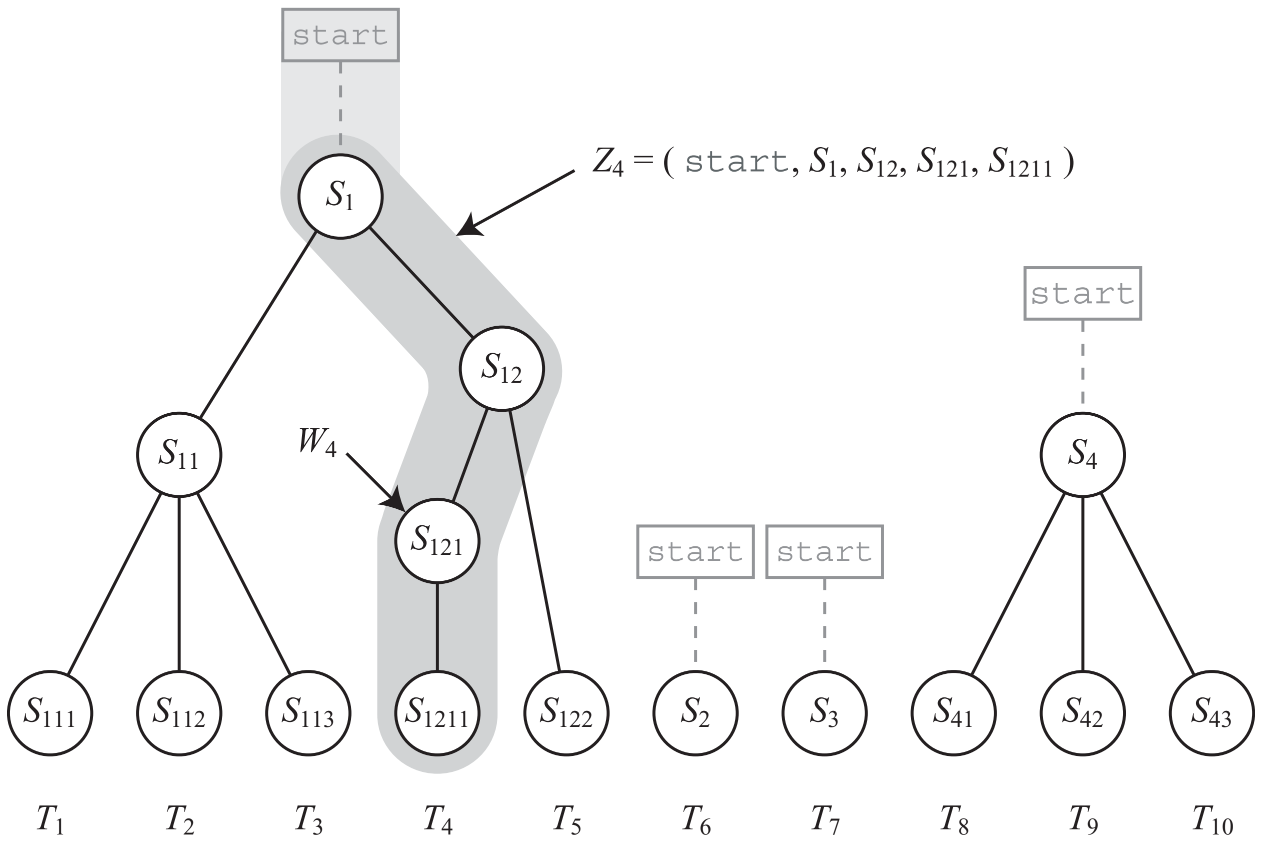

Figure 4.1: This figure shows an example of a sample from a HISM. The roots $S_1, S_2, \ldots$ are i.i.d. samples from $p_{\texttt{start}}$. For each root $S_k$, its immediate children $S_{k1}, \ldots, S_{kN}$ are i.i.d. samples from $p_{S_k}$ and the number of samples is random with distribution geometric$(q_{S_k})$. This process repeats for each nonterminal and always eventually ends with all terminals. The terminals can be arranged into a sequence $T_1, T_2, \ldots$. Each terminal $T_k$ is associated with a path $Z_k$ through its respective tree from the root (or more conveniently from `start`) to itself. The path corresponding to $Z_4$ has been highlighted. The sequence of $Z_k$'s is a Markov chain. If we augment $Z_k$ with information about how the tree branches from $Z_k$ to $Z_{k+1}$, say with $W_k$, which is the node just below the branch point, then the sequence of $X_k = (Z_k, W_k)$ is also a Markov chain and is conveniently parameterized.

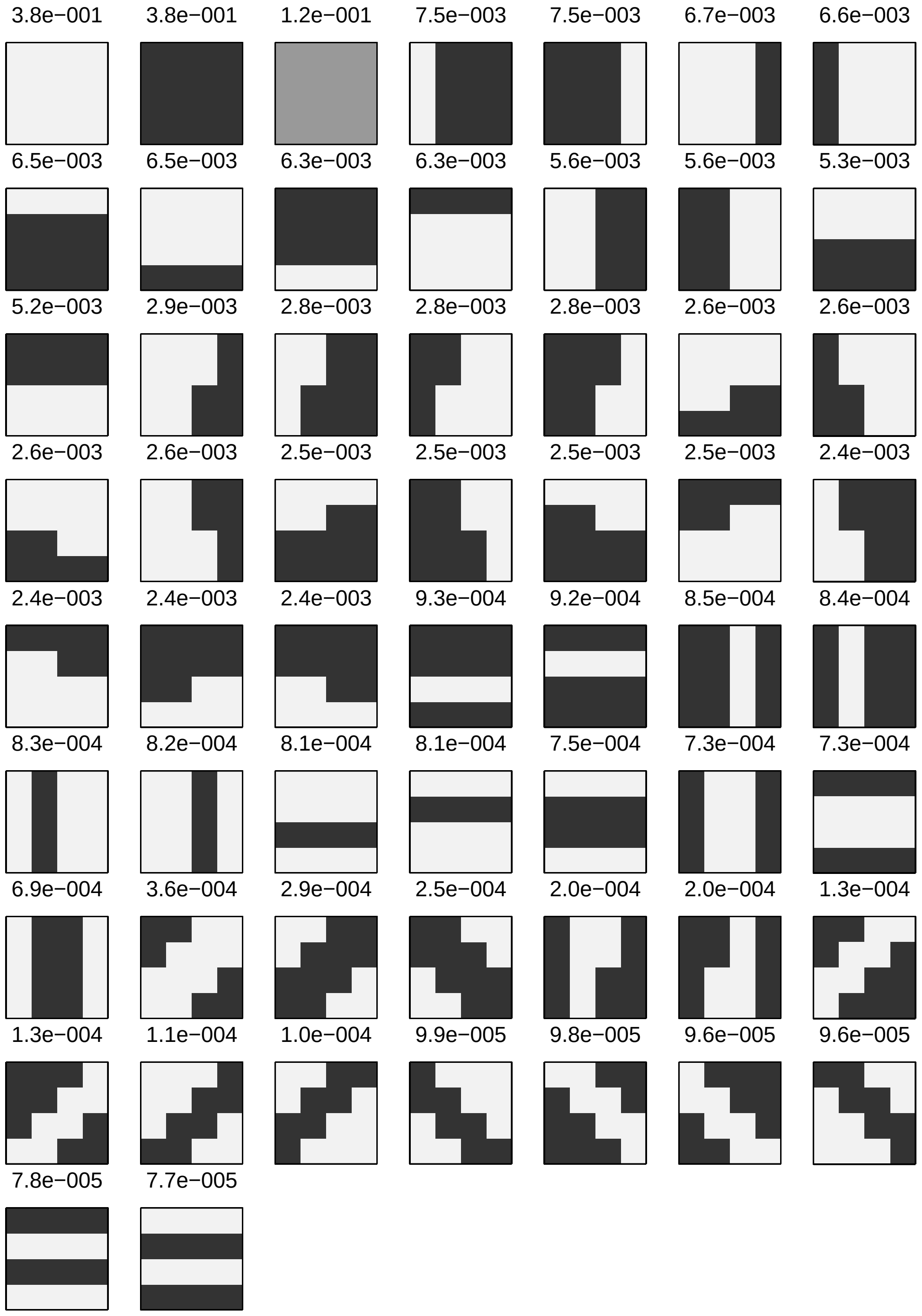

Figure 4.2: (Simulated video) The terminals for the HHISM are represented by the 67 “unusual” $4 \times 4$ patches found in binary images along with the “no patch” state (solid gray). The mixing proportions for each terminal (as estimated by EM) are shown above each patch.

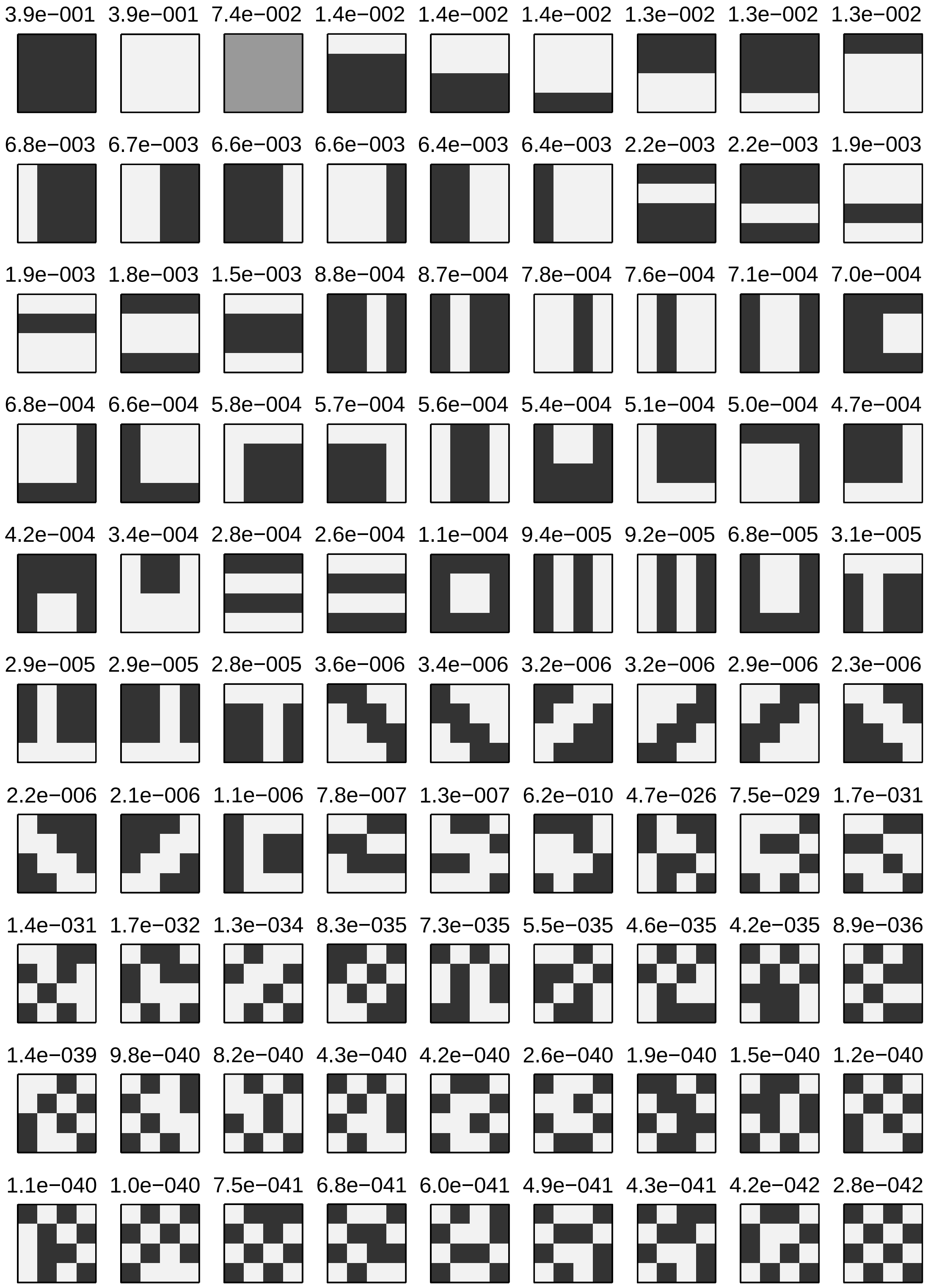
3.9e−001 3.9e−001 7.4e−002 1.4e−002 1.4e−002 1.4e−002 1.3e−002 1.3e−002 1.3e−002
6.8e−003 6.7e−003 6.6e−003 6.6e−003 6.4e−003 6.4e−003 2.2e−003 2.2e−003 1.9e−003
1.9e−003 1.8e−003 1.5e−003 8.8e−004 8.7e−004 7.8e−004 7.6e−004 7.1e−004 7.0e−004
6.8e−004 6.6e−004 5.8e−004 5.7e−004 5.6e−004 5.4e−004 5.1e−004 5.0e−004 4.7e−004
4.2e−004 3.4e−004 2.8e−004 2.6e−004 1.1e−004 9.4e−005 9.2e−005 6.8e−005 3.1e−005
2.9e−005 2.9e−005 2.8e−005 3.6e−006 3.4e−006 3.2e−006 3.2e−006 2.9e−006 2.3e−006
2.2e−006 2.1e−006 1.1e−006 7.8e−007 1.3e−007 6.2e−010 4.7e−026 7.5e−029 1.7e−031
1.4e−031 1.7e−032 1.3e−034 8.3e−035 7.3e−035 5.5e−035 4.6e−035 4.2e−035 8.9e−036
1.4e−039 9.8e−040 8.2e−040 4.3e−040 4.2e−040 2.6e−040 1.9e−040 1.5e−040 1.2e−040
1.1e−040 1.0e−040 7.5e−041 6.8e−041 6.0e−041 4.9e−041 4.3e−041 4.2e−042 2.8e−042

Figure 4.3: (Actual video) The terminals for the HHISM are represented by the 90 "unusual" 4×4 patches found in binary natural video along with the "no patch" state (solid gray). The mixing proportions for each terminal (as estimated by EM) are shown above each patch.

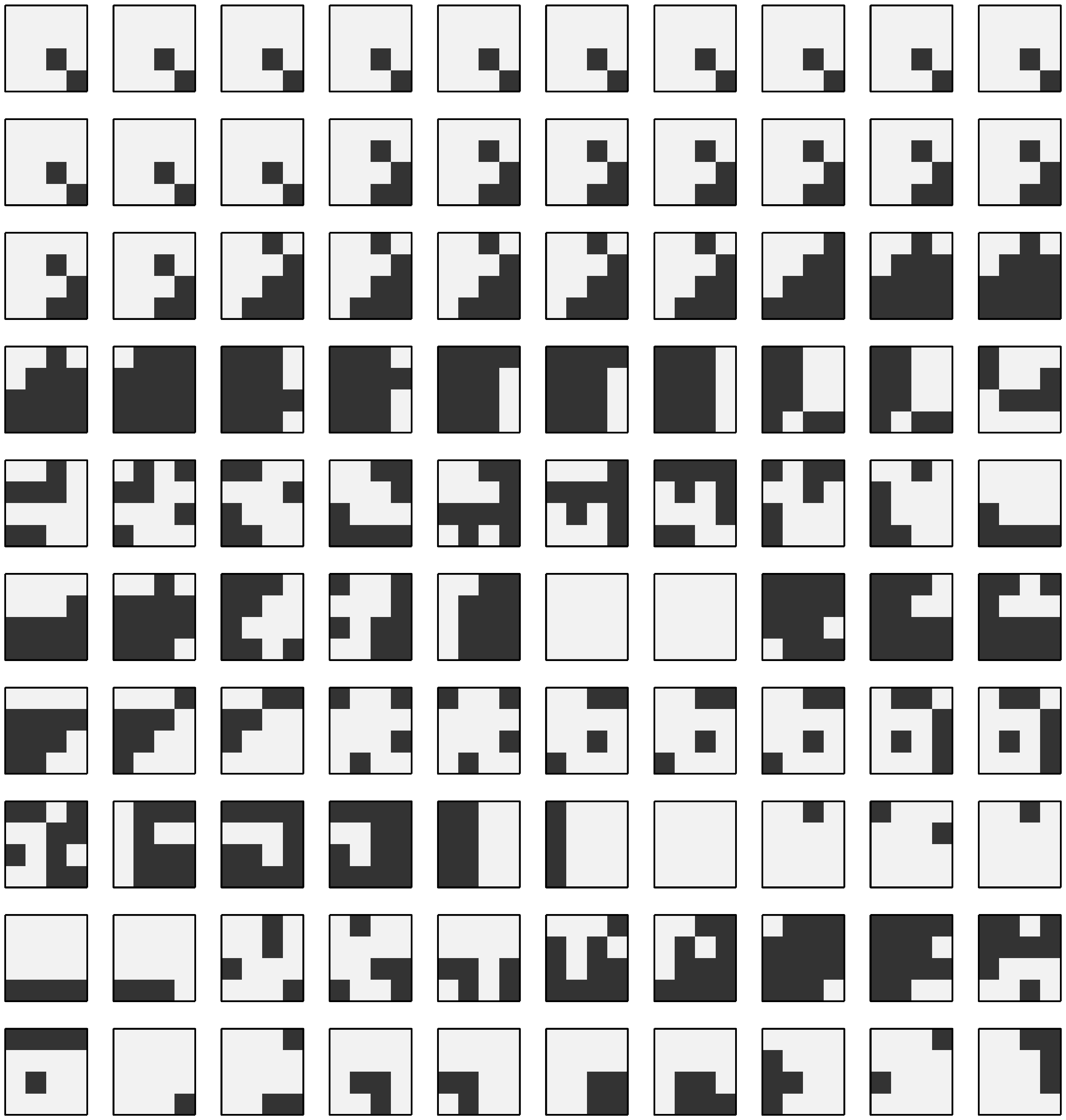

Figure 4.4: From left to right and then top to bottom, this figure shows the first 100 frames of an example simulated video sequence.

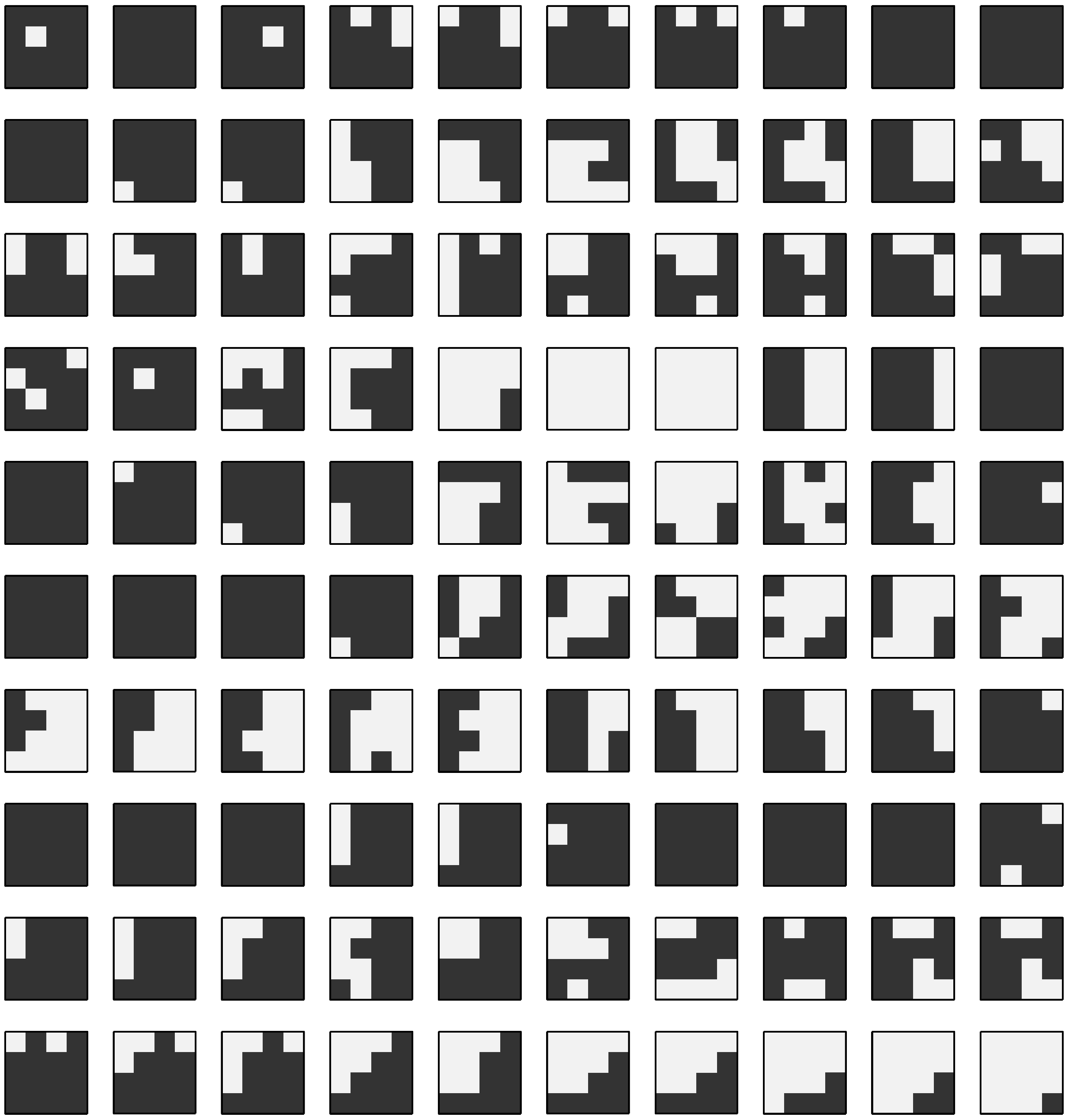

Figure 4.5: From left to right and then top to bottom, this figure shows the first 100 frames of an example actual video sequence.

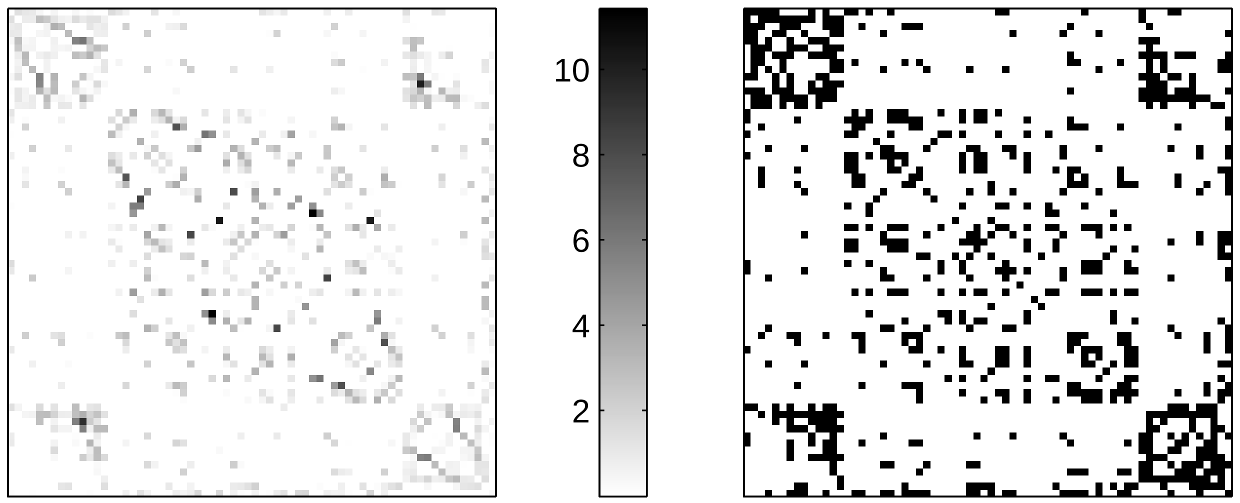

Figure 4.6: (Simulated video.) The matrix on the left is the $\log_2$ likelihood ratio for observed versus expected switching frequencies across all terminal pairs (as described in the text). Negative values are set to zero. The matrix on the right essentially indicates which values are positive from the matrix on the left.

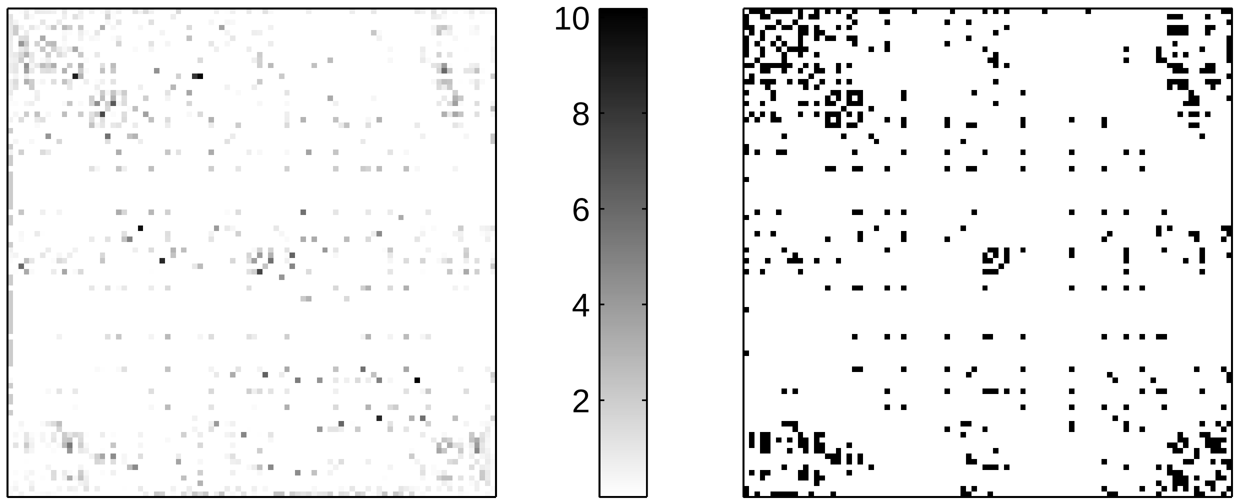

Figure 4.7: (Actual video.) The matrix on the left is the $\log_2$ likelihood ratio for observed versus expected switching frequencies across all terminal pairs (as described in the text). Negative values are set to zero. The matrix on the right essentially indicates which values are positive from the matrix on the left.

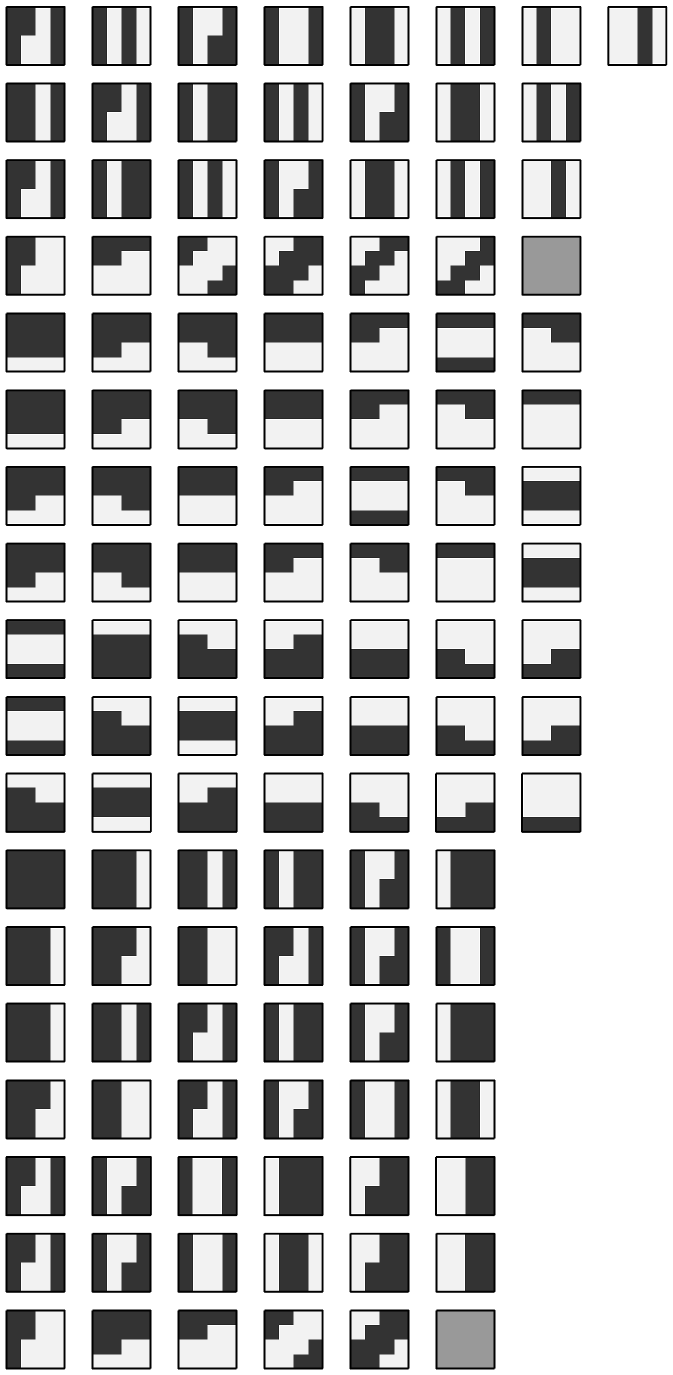
Figure 4.8: (Simulated video.) The maximal cliques in the graph defined by the connectivity matrix on the right in Figure 4.6. This figure continues until Figure 4.11. Each clique is represented by a horizontal sequence of patches. Larger breaks or a new line separate the different cliques. The only meaningful order is that the larger cliques come first.

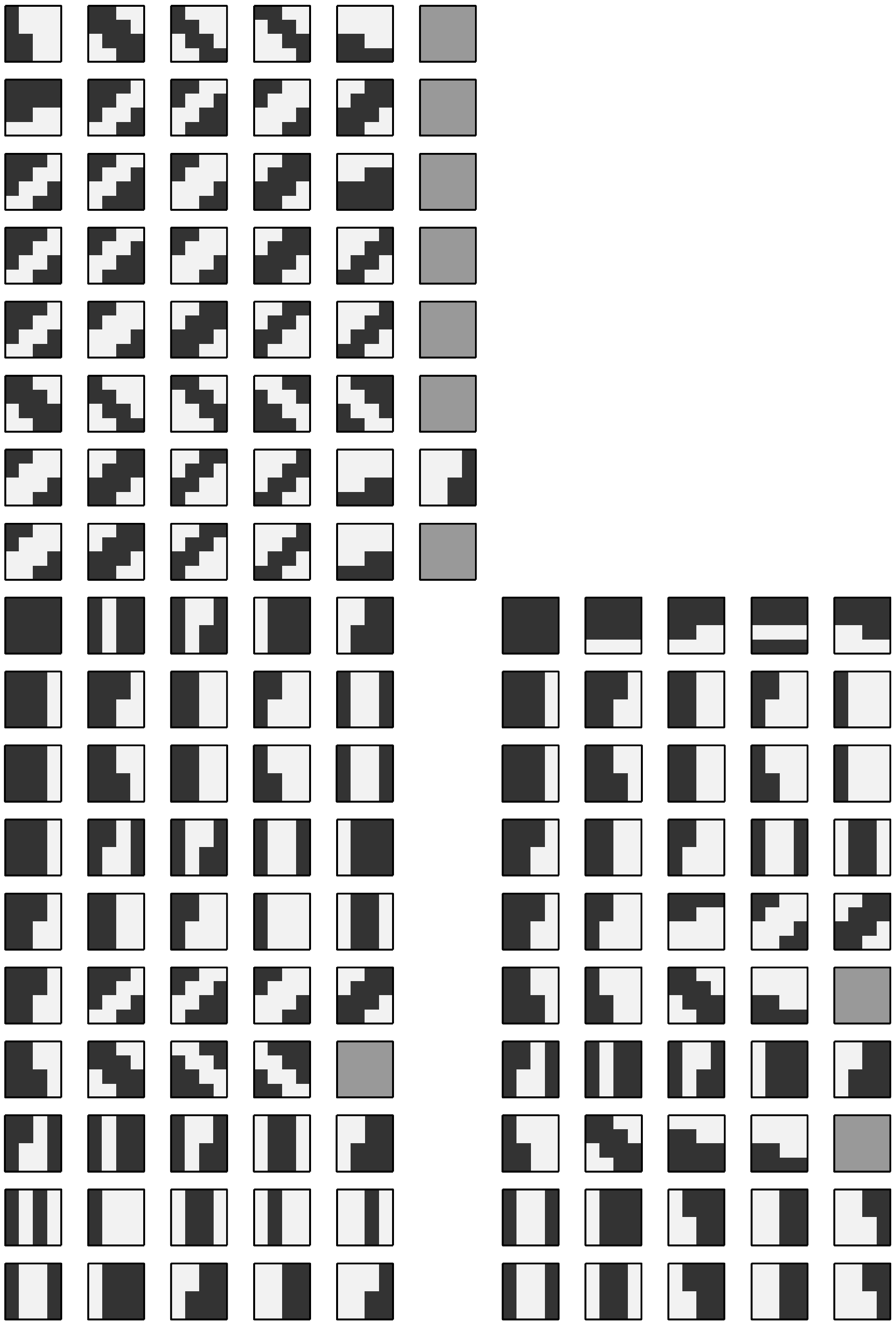

Figure 4.9: (Simulated video.) A continuation of Figure 4.8.

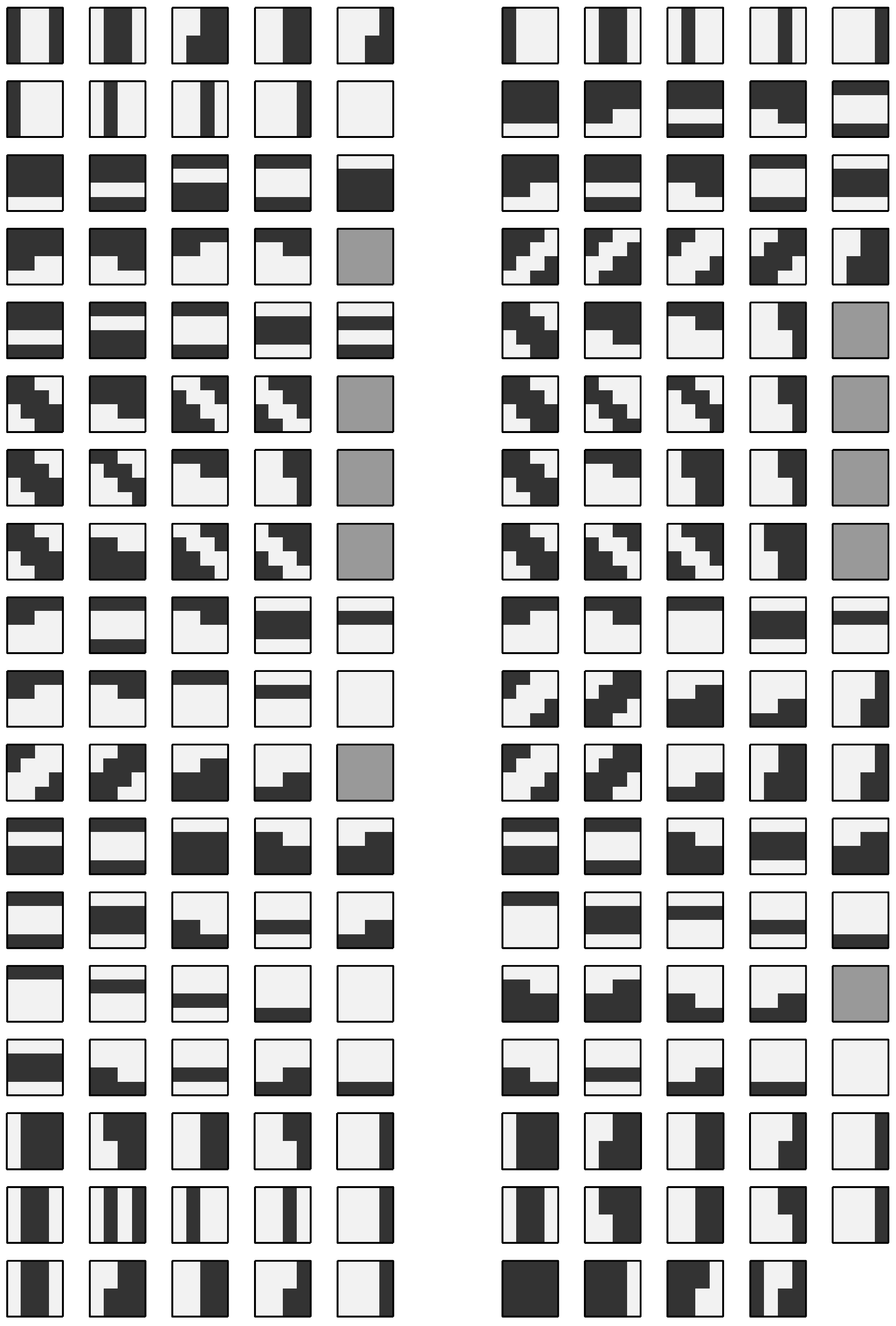

Figure 4.10: (Simulated video.) A continuation of Figure 4.8.

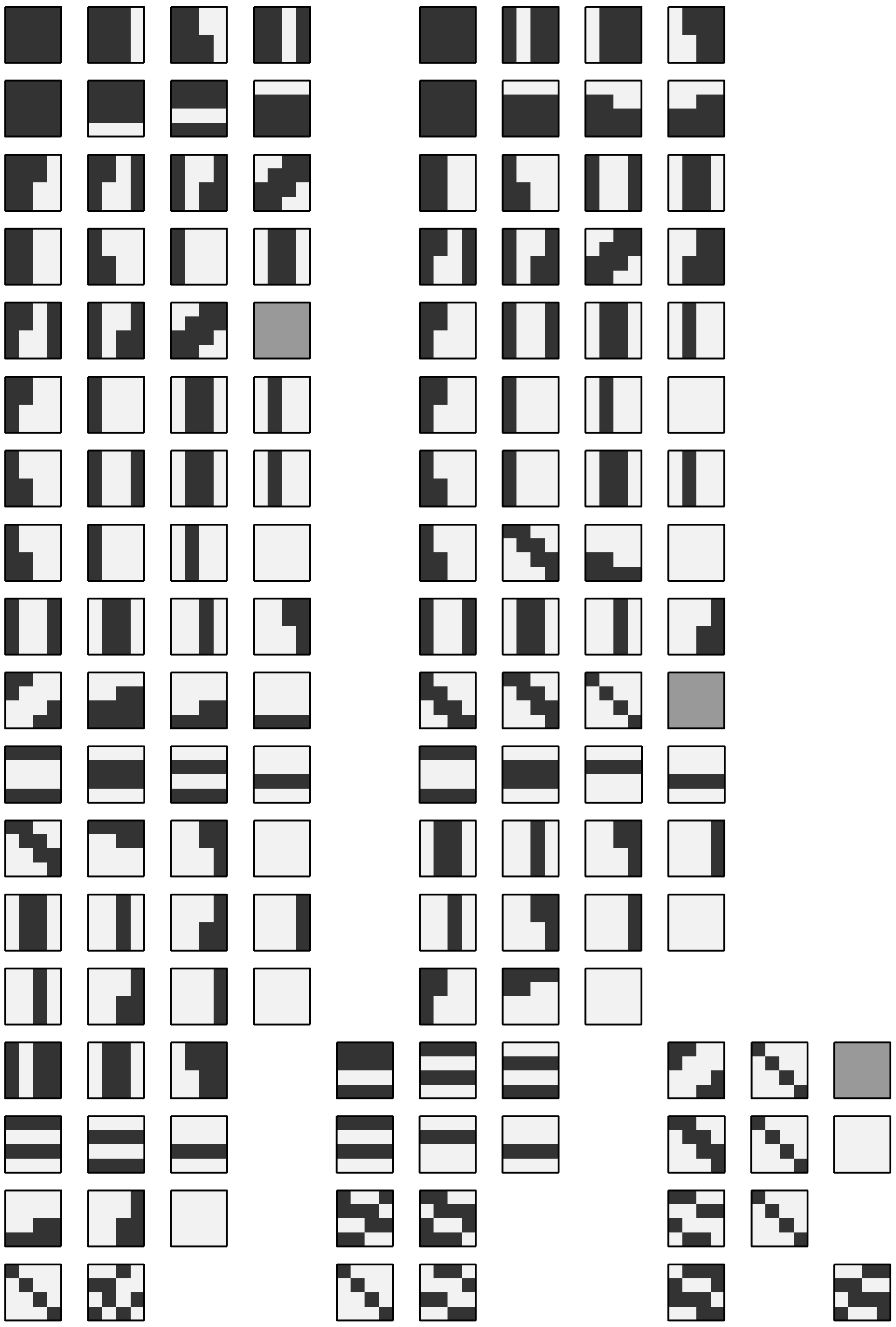

Figure 4.11: (Simulated video.) A continuation of Figure 4.8.

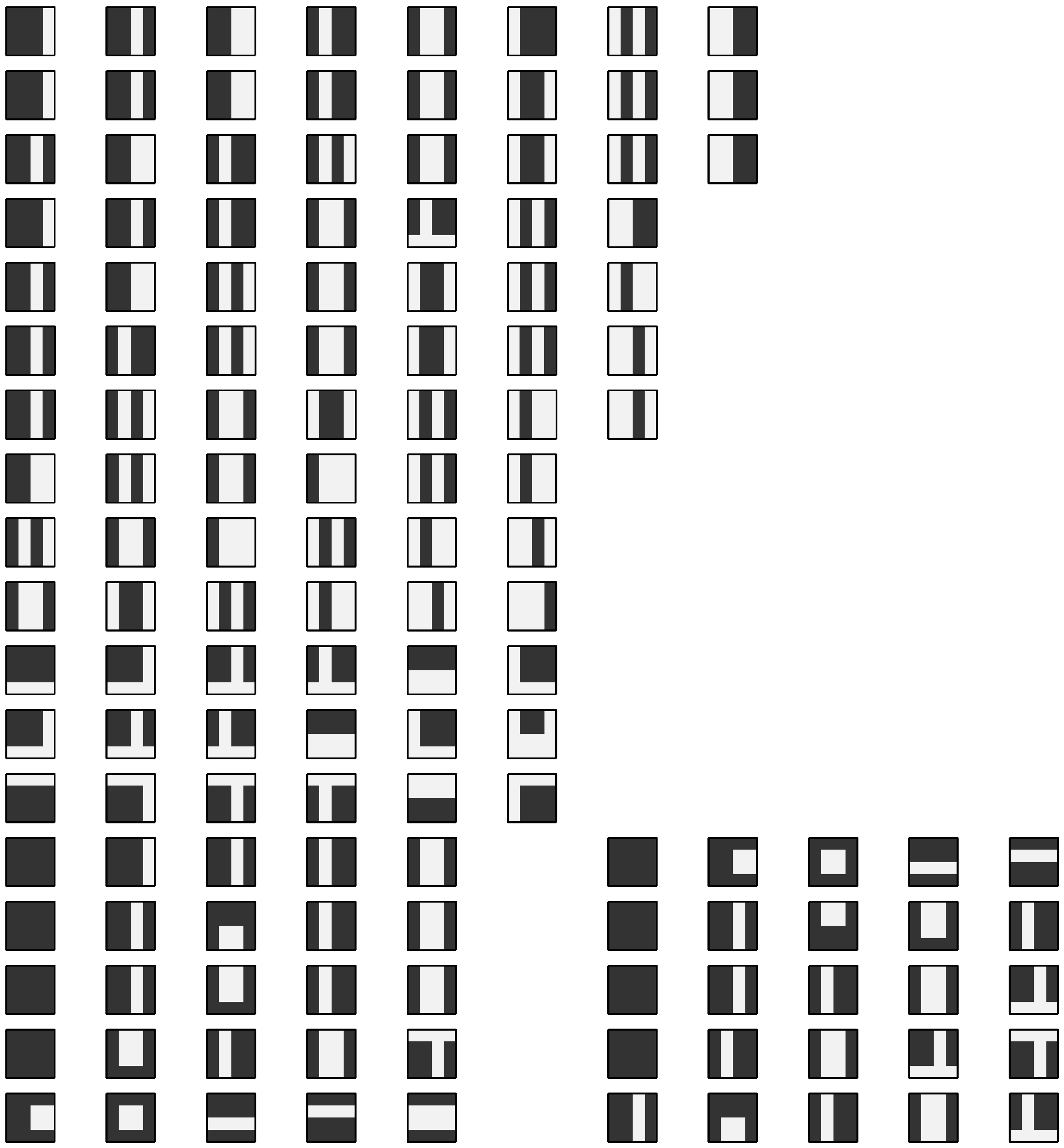

Figure 4.12: (Actual video.) The maximal cliques in the graph defined by the connectivity matrix on the right in Figure 4.7. This figure continues until Figure 4.15. Each clique is represented by a horizontal sequence of patches. Larger breaks or a new line separate the different cliques. The only meaningful order is that the larger cliques come first.

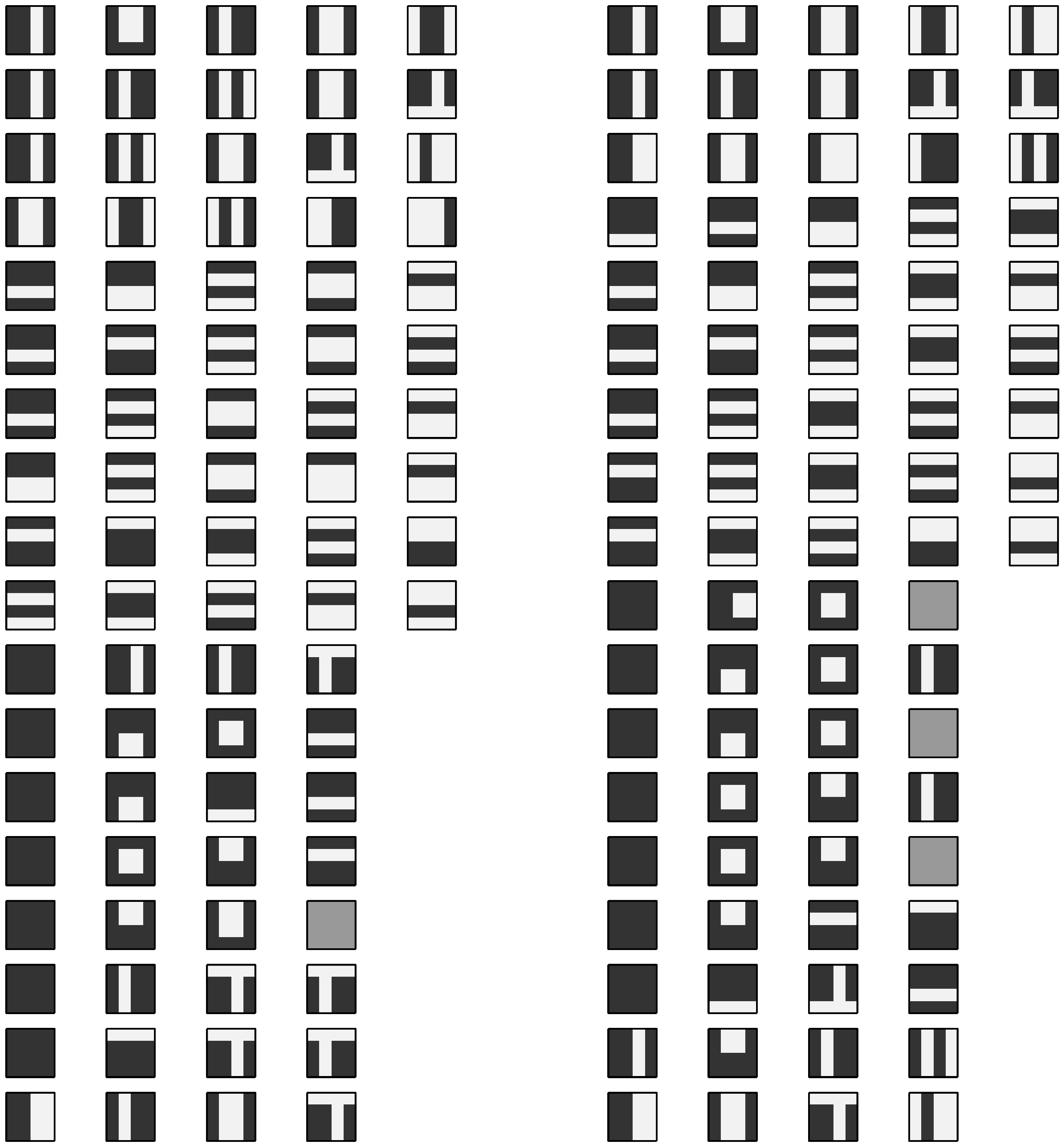

Figure 4.13: (Actual video.) A continuation of Figure 4.12.

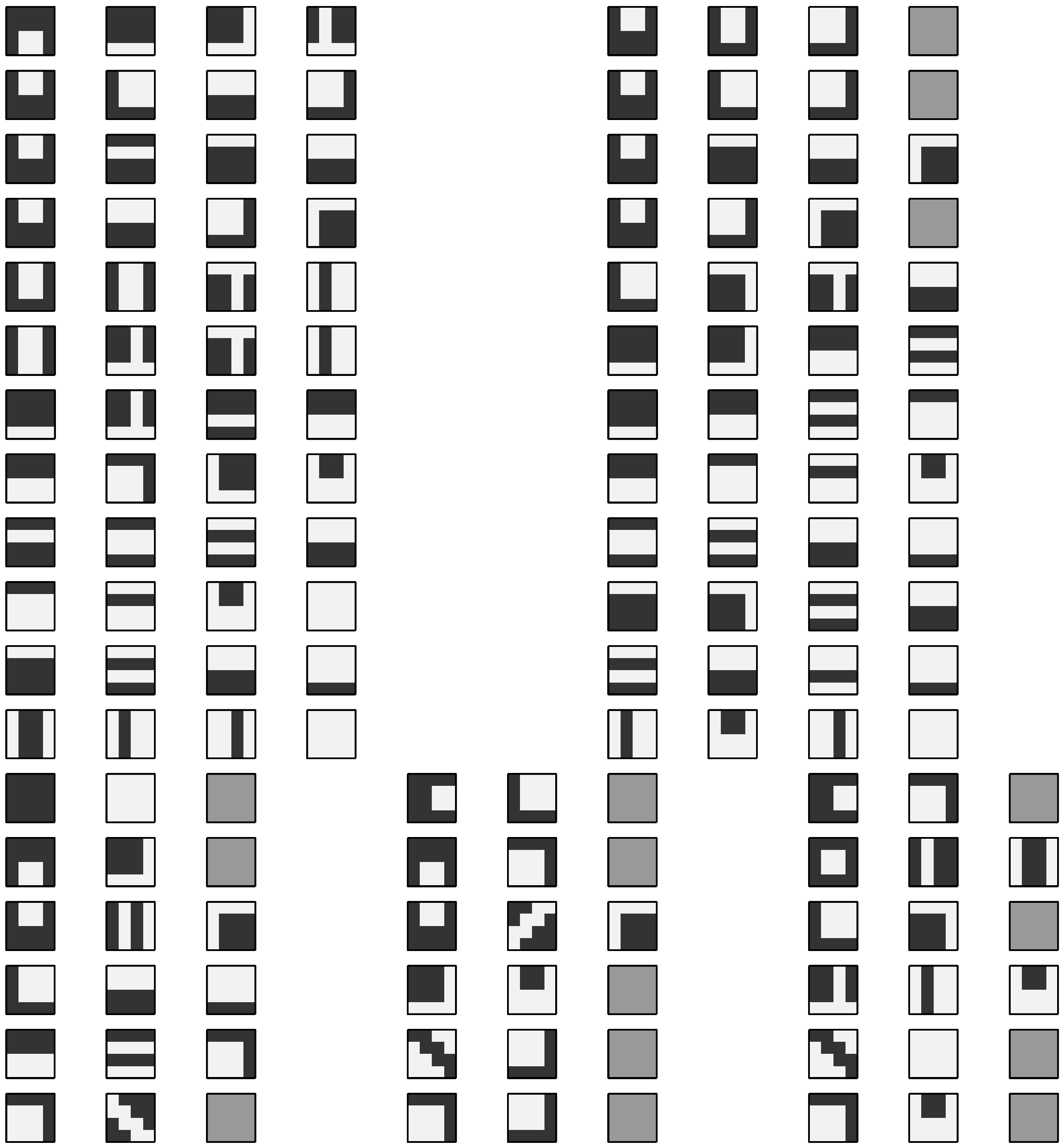

Figure 4.14: (Actual video.) A continuation of Figure 4.12.

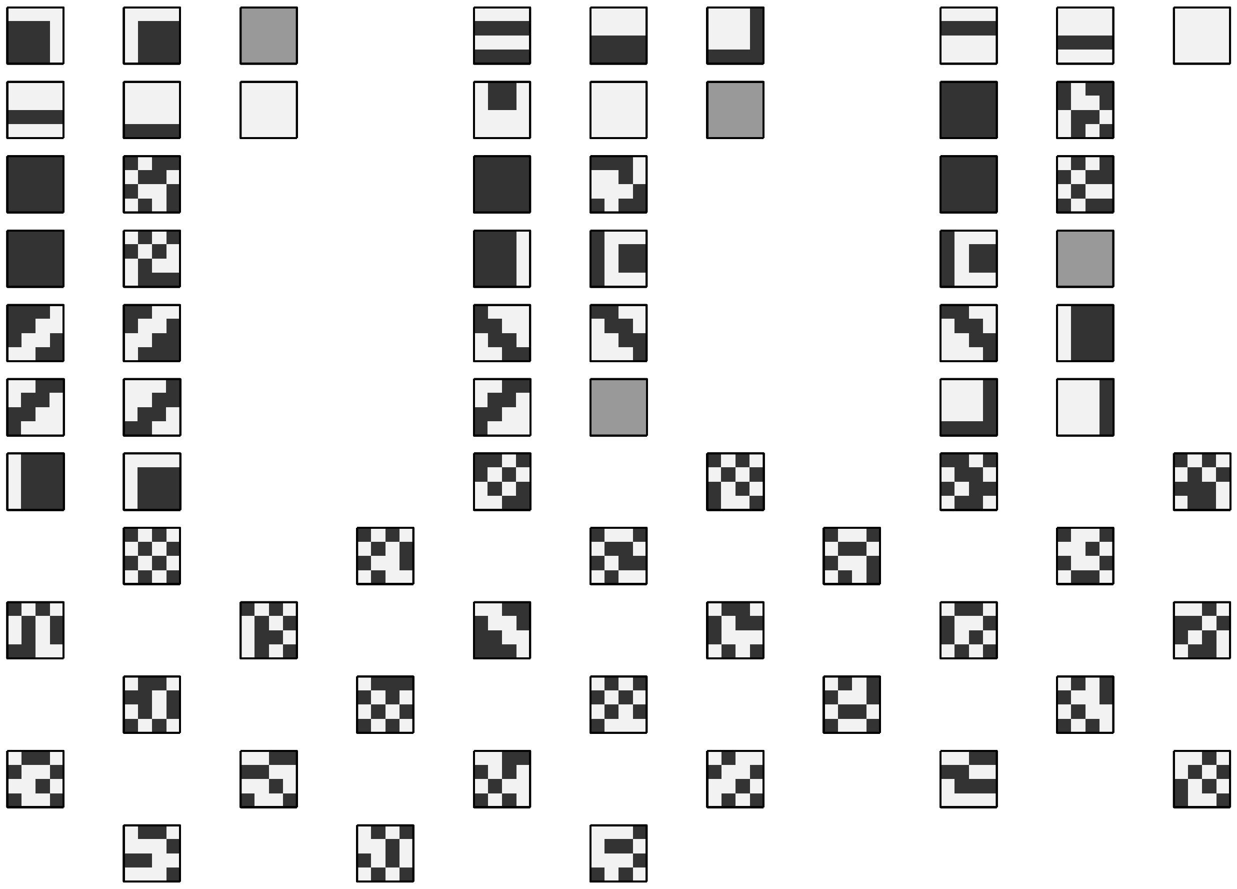

Figure 4.15: (Actual video.) A continuation of Figure 4.12.

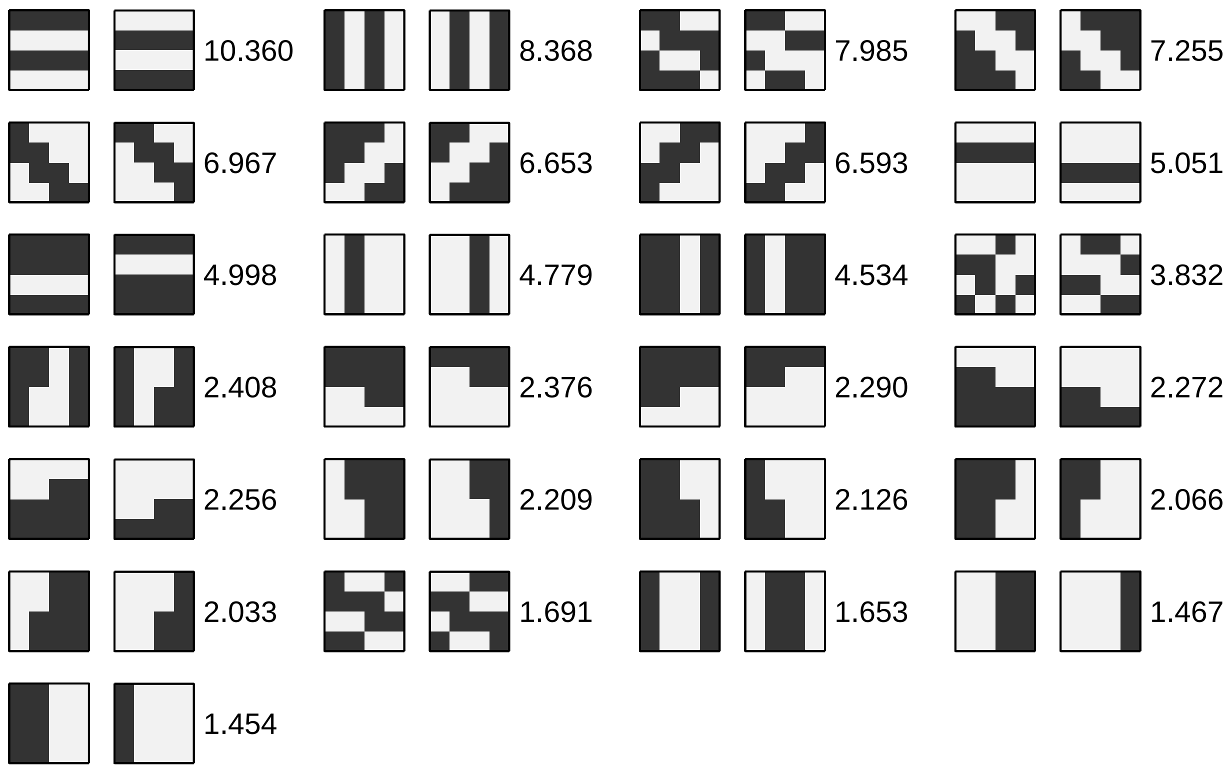

Figure 4.16: (Simulated video.) Each group represents the children of nonterminals added to the HISM. The groups were selected from all subsets of suspicious coincidences by using a greedy procedure described in the text. The numbers to the right of each group are the $\log_2$ likelihood ratios.

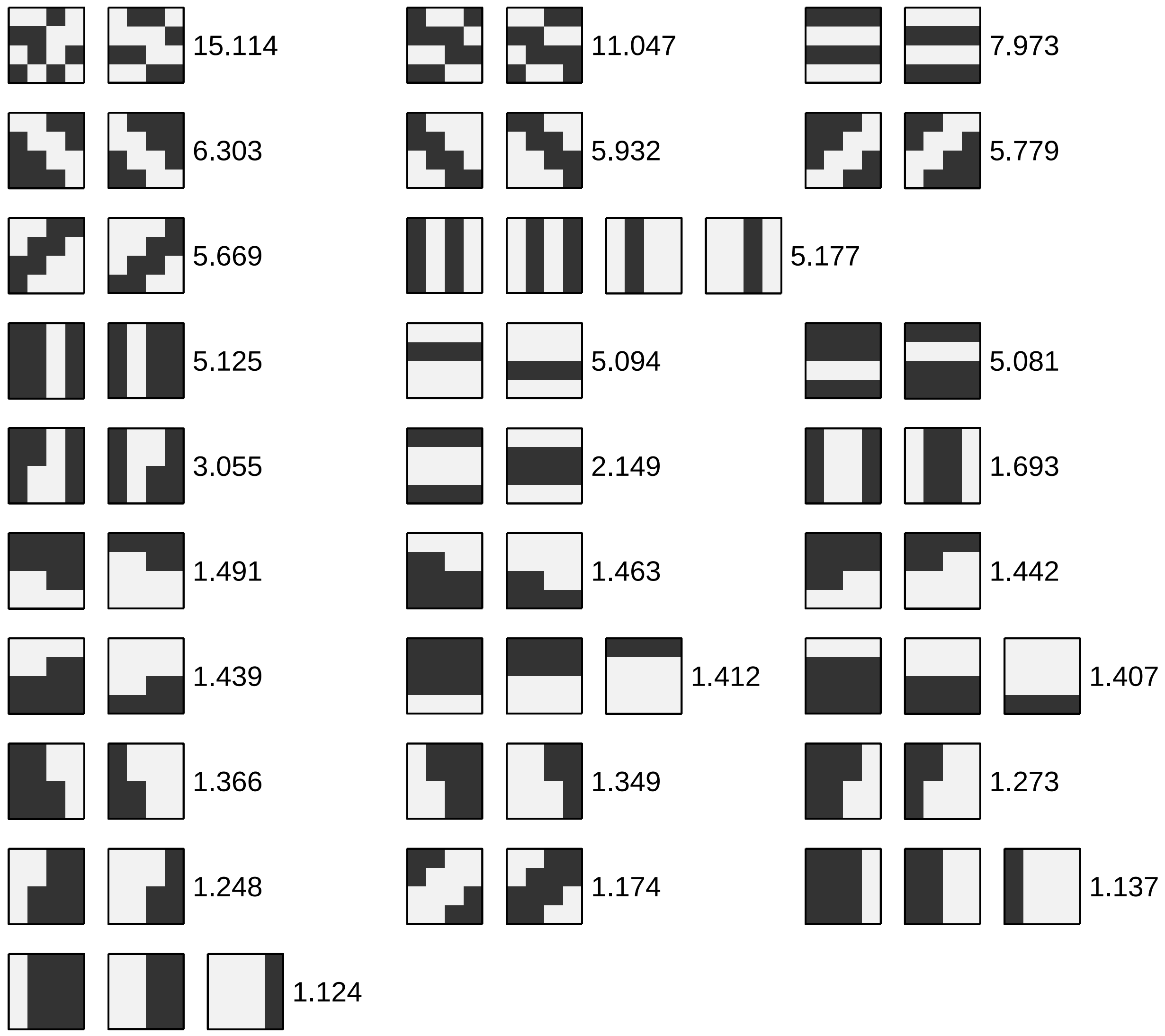

Figure 4.17: (Simulated video.) This is just like Figure 4.16 except that it came from a different data set. It illustrates that the greedy selection procedure can result in groupings of more than two children.

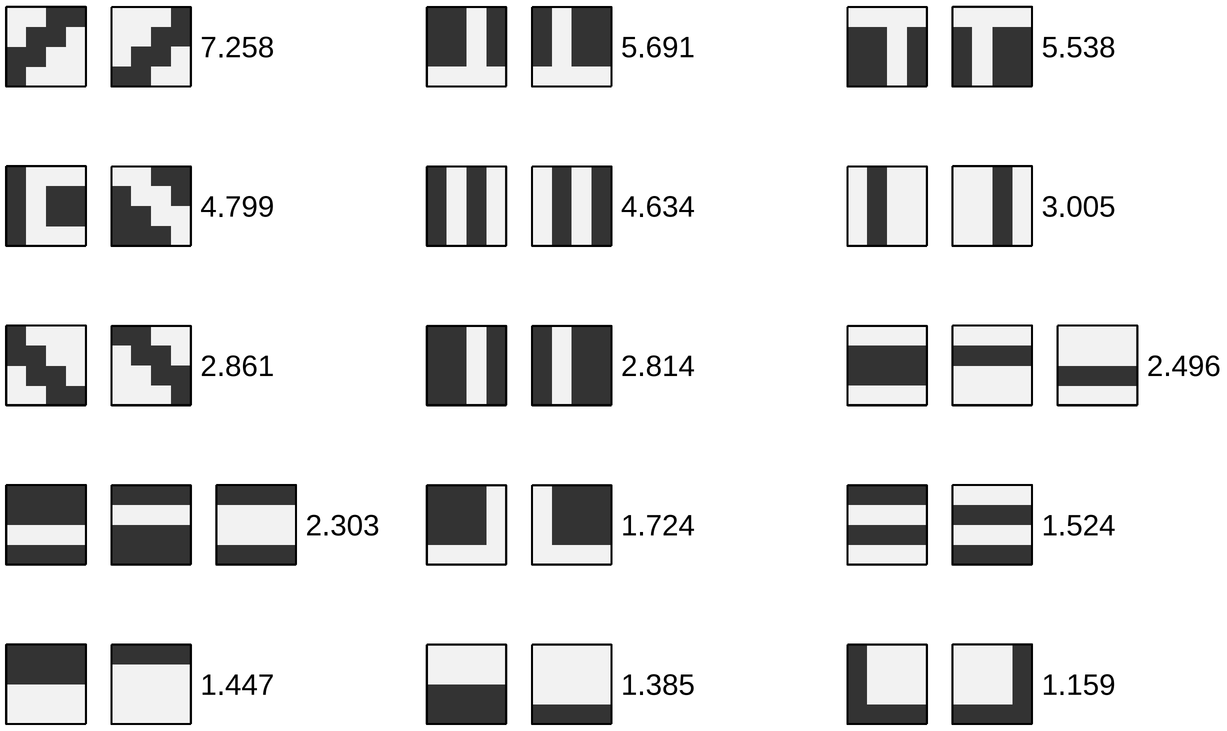

Figure 4.18: (Actual video.) Each group represents the children of nonterminals added to the HISM. The groups were selected from all subsets of suspicious coincidences by using a greedy procedure described in the text. The numbers to the right of each group are the $\log_2$ likelihood ratios.

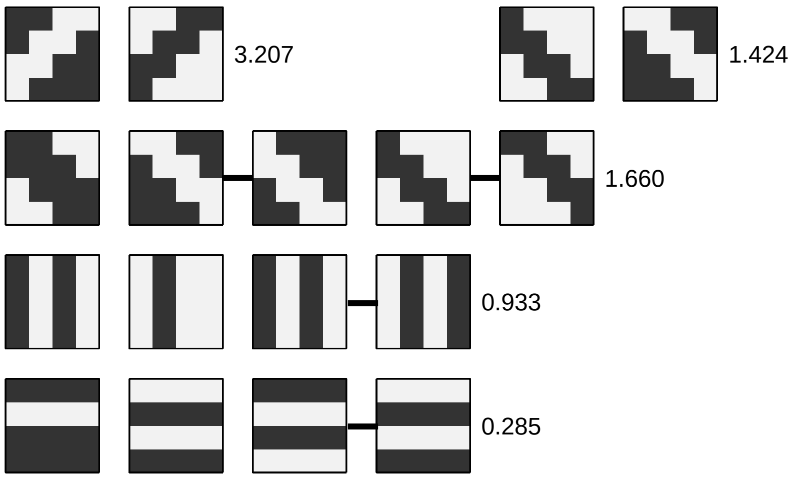

Figure 4.19: (Simulated video.) Adding the nonterminals indicated in Figure 4.16, fitting the model and then using the greedy partition procedure again gives the above suspicious coincidences. Elements of the new nonterminal set are visualized by showing their children connected with lines. The numbers to the right of each group are the $\log_2$ likelihood ratios.

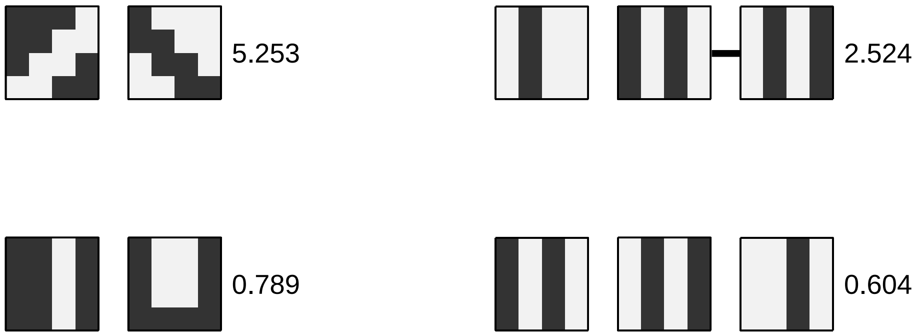

Figure 4.20: (Actual video.) Adding the nonterminals indicated in Figure 4.18, fitting the model and then using the greedy partition procedure again gives the above suspicious coincidences. Elements of the new nonterminal set are visualized by showing their children connected with lines. The numbers to the right of each group are the $\log_2$ likelihood ratios.

# Chapter 5

# Future work

## 5.1 Learning selectivity and invariance

The experiments in Chapters 3 and 4 illustrate that selectivity can result from incorporating spatial dependencies into a probabilistic graphical model and that invariance can result from incorporating temporal dependencies. In future work we hope to address the original goal set forth in Chapter 1 of combining both into a hierarchal framework, specifically, a hierarchy of reusable parts, or a composition system.

There are several major hurdles. The first and foremost is probably computation, although many recent examples in the literature have demonstrated the feasibility of computing with large graphical models using a variety of different techniques. (For some specific examples, see [5, 7, 6].) We do not address computation here. A more theoretical concern is how to learn selectivity on top of invariance. Of course the solution to this problem undoubtedly affects computation.

## 5.2 The Markov dilemma

Invariance, by definition, hides information. Translation invariance hides location. Rotation invariance hides orientation. Scale invariance hides size. In the context of statistical computing, invariance can be viewed as a Markov assumption. The decisions made on top of a translation invariant representation are conditionally independent of actual location information. Location information is accumulated (and thereby lost) by the invariant layer before this information is passed on to further layers. Markov assumptions form the backbone of all probabilistic graphical models and it is not clear how to proceed with computation and estimation in their absence.

Furthermore, as we mentioned in Chapter 1, the right type of invariance seems crucial for fast learning and generalization because it can substantially lower the dimensionality of the state space. Of course, it has to be the right type of invariance. If the information that needs to be learned is hidden by an earlier level of invariance, then learning – fast or slow – is impossible. And therein lies the problem: how do we know what it is the right type of invariance? Stuart Geman and Elie Bienenstock refer to this as the *Markov dilemma*: Markov assumptions (i.e., invariance) seem necessary for computation and learning, yet

Markov assumptions often hide information that is needed for later decisions. In particular, invariance seems likely to hide information that is needed for later selectivity. Note that this affects both computation and learning. Note also that it is peculiar to hierarchies of invariance and selectivity. Figure 5.1 is their canonical illustration of the problem [1]. Trying to build a long (translation invariant) bar detector out of two adjacent, smaller translation invariant bar detectors does not necessarily work. The large bar detector cannot tell when the smaller bars are appropriately aligned.

The Markov dilemma is closely related to the well known *binding problem* in neural systems [8, 9]. (See the special issue of *Neuron*, Vol. 24, Sept. 1999, for a collection of discussions about the binding problem.) Presumably such a problem will present itself when we try to combine the two approaches from Chapters 3 and 4. Some of the avenues that we will likely need to explore are mentioned below. They are certainly not mutually exclusive.

- **No dilemma.** If we think about selectivity as arising from spatial dependencies and invariance as arising from temporal dependencies, it is not clear why there should necessarily be a conflict. A better theoretical understanding of this connection might reveal a good solution. For example, the ideas of sparsity and entropy reduction that have played such an important role in methods like sparse coding and ICA, which are essentially learning selectivity, can also be applied to the temporal stability methods for learning invariance. Indeed, a binary random sequence is sparse (i.e., has low entropy) not just when 1's are rare, but also when 1's clump together (temporal stability). Perhaps an information theoretic framework that unified the two would also give insight into the Markov dilemma.

- **Over-representation.** A common solution when a useful Markov assumption is too strong is to increase the state space. Consider, for example, the situation in Figure 5.1 where the information in the two smaller invariant detectors is not sufficient to distinguish one long bar from two short bars. There are differences between the two cases other than location, however. The case with two bars also has features like end-of-bar or parallel-bars. Similarly the case with one bar also has a variety of short bars in between (spatially) the two short bar detectors. If all of these types of features were also represented by the system, even if they were represented invariantly, it might be possible to build an invariant long bar detector not with just two short bars, but perhaps with many short bars and also with the absence of end-of-bars and parallel-bars. By building a long bar detector out of many more features (increasing the state space) we might be able to overcome the information that is lost by the invariance. (See [4] for a discussion and examples of this strategy for object detection and/or recognition. See also [6] for an example where location information is explicitly included in the state space.)

  The drawback, of course, is that increasing the state space typically makes both computation and learning more difficult. It is not clear how large the state space will need to be in order to create a recognition system that is robust to clutter. Another drawback is that suspicious coincidence detection is overwhelmed in a densely distributed representation (see Chapter 2 and the references therein), so our model building strategy would likely have to be modified.

- **Auxiliary information.** It might be possible to partially violate the Markov property and still preserve much of the benefits. For example, certain summary auxiliary information like location and scale could be passed around the invariance. The difference between this idea and increasing the state space is that presumably this auxiliary information would be treated fundamentally differently, perhaps augmenting a more classical approach based on the Markov property.

## 5.3 Neural systems

Another approach is to investigate how the brain might surmount the Markov dilemma, or to see if it even needs to. An obvious first step is to gather more detailed information about the response properties of visual neurons. Many groups have and continue to do just this. Nevertheless, the technical and statistical hurdles are tremendous. It is still not clear if the caricature from Chapter 1 is an accurate picture. In Chapter 6 we suggest some statistical techniques for investigating the response properties of visual neurons in a more agnostic manner. Agnostic methods are becoming more and more popular for making precise statements about the amount and type of selectivity and invariance in the visual system (see Chapter 6 for references).

Working on the assumption that the visual hierarchy exists, there have been several proposed solutions to the Markov dilemma (or variants of it, like perceptual binding) that loosely fit into our "auxiliary information" category above, for example, the use of sophisticated attention mechanisms [3]. Several of the proposed solutions in the literature are based on the fine temporal structure of neural firing patterns. (See [2] for a recent review of the best known variants of this idea.)

In the specific context here, Stuart Geman has proposed a solution that uses partially synchronous firing to pass information about functional connectivity through the invariance [1]. This information is sufficient for distinguishing the two cases in Figure 5.1. In Chapter 7 we describe some jitter-based statistical techniques for investigating the temporal resolution of neural firing patterns in a more agnostic manner. Jitter methods offer a great deal of flexibility for investigating things like synchronous firing, but with few modeling assumptions.

## Bibliography

[1] Stuart Geman. Invariance and selectivity in the ventral visual pathway. 2004 (in preparation).

[2] Charles M. Gray. The temporal correlation hypothesis of visual feature integration: Still alive and well. *Neuron*, 24:31–47, September 1999.

[3] John H. Reynolds and Robert Desimone. The role of neural mechanisms of attention in solving the binding problem. *Neuron*, 24:19–29, September 1999.

[4] Maximilian Riesenhuber and Tomaso Poggio. Are cortical models really bound by the "binding problem". *Neuron*, 24:87–93, September 1999.

[5] Stefan Roth and Michael J. Black. Field of experts: A framework for learning image priors with applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005 (submitted).

[6] A.J. Storkey and C.K.I. Williams. Image modelling with position-encoding dynamic trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7):859–871, 2003.

[7] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7):787–800, 2003.

[8] C. von der Malsburg. The correlation theory of brain function. Internal Report 81–2, MPI Biophysical Chemisty, 1981. Reprinted in *Models of Neural Networks II.* E. Domany, et.al., eds. Springer, Berlin, 1994.

[9] C. von der Malsburg. Binding in models of perception and brain function. *Current Opinion in Neurobiology*, 5:520–526, 1995.

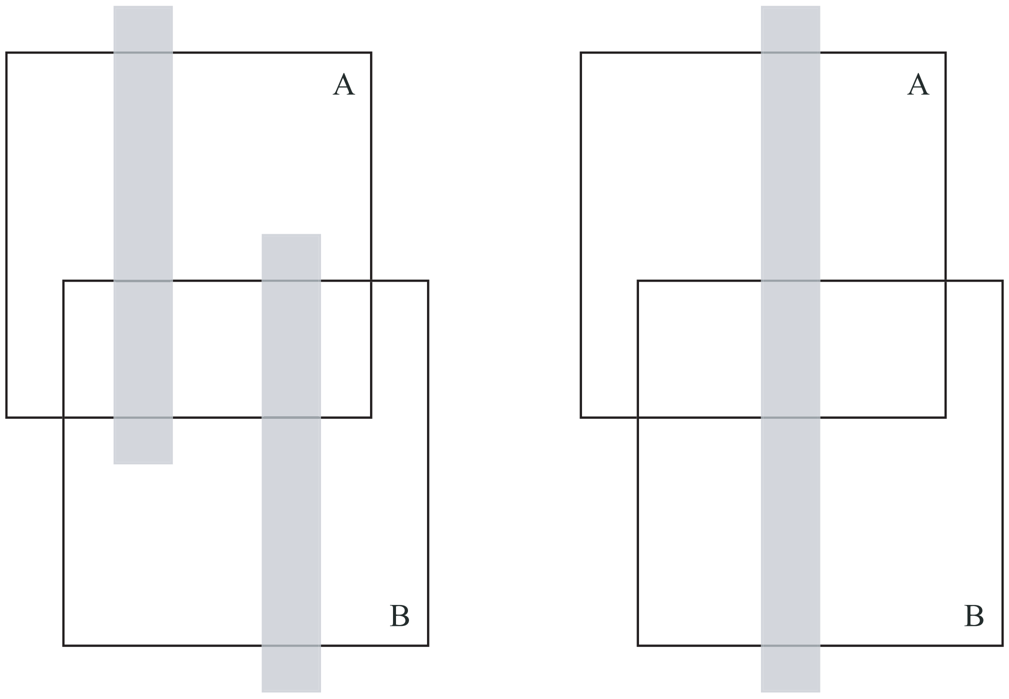

Figure 5.1: This figure from [1] illustrates the Markov dilemma. An invariant bar detector responds equally to any vertical bar in box A. Another responds equally to any vertical bar in box B. The situations on the left and the right cannot be distinguished based on the responses of these two filters alone. If we wanted to use these two invariant bar detectors to build a longer invariant bar detector we would need extra information. The invariance of the small detectors hides information that we need for later selectivity.

# Part II

# Neuroscience

# Chapter 6

# Fitting receptive fields

This chapter first appeared with minor differences as [7]: M. Harrison, S. Geman and E. Bienenstock. *Using statistics of natural images to facilitate automatic receptive field analysis.* APPTS Report #04-2, February 2004.

## 6.1 Introduction

Part of the motivation underlying this thesis is the caricature of the visual system described in Chapter 1. It is not known whether or not this caricature is an accurate description of physiology. Even if we take the highly simplified view that a single neuron is a static feature detector, discovering the feature (or class of features) that it presumably detects is extremely difficult. A common alternative is to describe how the neuron behaves in response to a relatively small collection of stimuli, like oriented edges, but this does not necessarily address how it behaves in response to more complicated (and more realistic) stimuli. In this chapter we describe some methods that might be useful for investigating neural response properties in more detail.

A neuron can be conceptualized as performing a function on its inputs. This stimulus-response function characterizes "what the neuron does" and is the main object of interest for cellular-level neuroscience. Automated methods for discovering certain properties of stimulus-response functions have many advantages over traditional methods [5]. They can perform more exhaustive searches, use less-biased inputs and have tighter feedback loops for online, adaptive analysis. However, automated methods require many stimulus presentations, often many more than are feasible for a given physiology experiment. Finding ways to reduce the number of stimulus presentations is crucial for increasing the power and scope of these important techniques. We believe that one way to do this is to take advantage of the statistics of natural images.

Neurons in V1 appear to be tuned to the specific statistical properties of natural images. For example, a variety of techniques like sparse coding [14, 8], independent component analysis (ICA) [1, 9] and slow feature analysis (SFA) [18, 2], when applied to natural images, yield response properties strikingly similar to both simple and complex cells in V1. That neurons are tuned to natural inputs makes sense from both an evolutionary perspective and from a developmental perspective. We can use this knowledge to more efficiently probe the response properties of visual neurons.

A simple thought experiment partially illustrates our idea. Natural images can be significantly compressed using algorithms like JPEG without suffering any noticeable loss in quality. This suggests one (or both) of two things. (1) Perhaps neurons in the visual cortex ignore certain types of variation in images. So if we want to somehow search for the stimulus that makes a neuron respond the strongest, then it seems reasonable to ignore these same types of variation. This constrains our search and makes it more efficient. Or, (2) Perhaps (undoubtedly) natural images are lower-dimensional than, say, the number of pixels might suggest. So searching only in “the space of natural images” will be more efficient. This does not necessarily find the optimal stimulus, but it does find the optimal natural stimulus.

Here we focus on two particular types of analysis. The first is designed to identify the optimal stimuli for a neuron, that is, to find the maximizers of a neuron's stimulus-response function. The second attempts to approximate the entire stimulus-response function by using an appropriate parametric model, like linear or quadratic. We briefly describe each method and present some preliminary simulation results.

## 6.2 Stimulus Optimization

The goal is to find the input that maximizes the response of a given neuron using an adaptive, online search algorithm. Two basic methods are described in Földiák (2001; area V1, anesthetized monkey) [5] and Földiák et al. (2003; area STSa, awake monkey) [6]. In both cases they demonstrate the feasibility of these techniques in physiology experiments. We propose to increase the efficiency and thereby the utility of these methods. Here we will focus on the method in Földiák (2001).

### 6.2.1 Gradient ascent

The basic idea is to do gradient ascent (“hill-climbing”) on the stimulus-response function. A stimulus $x(t)$ is presented on trial $t$ and the neural response $y(t)$ is recorded. Now we update the stimulus by moving it in the direction of the response gradient. This gives a new stimulus $x(t+1)$, based on the previous, that should yield a higher response from the neuron. Repeatedly updating in this way allows us to climb the stimulus-response function until we find the optimal stimulus.

Of course, there are many caveats. The main problem is how to find the response gradient. We conceptualize the response $y$ to the stimulus $x$ as a noisy version of some ideal or mean response $f(x)$. The function $f$ is the stimulus-response function of the neuron. We want to discover the gradient vector $\nabla f(x)$ of $f$ at $x$. This can be done using a block of stimulus presentations, each of which is the original $x$ plus noise. The gradient is (to a first approximation) proportional to the covariance of the noise and the response [5]. More details can be found in the Appendix.

The basic experimental design is thus a series of blocks of trials. Within each block the baseline stimulus is held constant and the animal is presented noisy versions of it. At the end of a block, the neural response gradient is estimated. Then the baseline stimulus is updated in the direction of the gradient before moving to the next block of trials. Because a block of trials are needed for each update of the stimulus, this method requires a large number of total trials for a given experiment. The computations can be carried out more or less instantaneously with a modern desktop computer and should add no further constraints on the time course of an experiment.

### 6.2.2 Dimensionality reduction

We can reduce the number of trials by reducing the dimensionality of the stimulus space. An arbitrary dimensionality reduction will strongly bias our search for the optimal stimulus. By using the properties of natural images to reduce the dimensionality in an intelligent way, however, we should be able to bias the search toward the optimum and not away from it.

One of the simplest types of dimensionality reduction is principal component analysis (PCA). Using a collection of natural image patches ($16 \times 16$ pixels), we estimated the first 10 principal components. The gradient ascent method can be applied in this 10-dimensional subspace of principal components or in the original 256-dimensional pixel space. We compared the two methods in a simulation study.

In the first experiment we simulated the response of a V1 complex cell as a quadratic function of the pixel input, followed by a monotonic nonlinearity, followed by Poisson noise with this rate. That is, the mean of the Poisson observation was $f(x^T Ax + x^T B + C)$, where $f(z) = 10 \exp(z/2)/(1 + \exp(z/2))$. The parameters $A$, $B$ and $C$ of the quadratic were fit using slow feature analysis (SFA) trained on natural images. SFA produces cells with many of the properties of V1 complex cells including phase invariance, active inhibition and direction selectivity [2]. The form and parameters of $f$ were chosen by hand.

Figure 6.1 shows three different full-dimensional searches for the optimal stimulus. This is the search that was used in Földiák (2001). Each graph shows the mean response of the neuron to the baseline image over 50 blocks. The response is normalized between 0 and 1 where 0 is the minimum possible mean response and 1 is the maximum possible mean response. (These values were obtained numerically using $A$, $B$ and $C$.) The first graph uses 10 trials per block to estimate the response gradient. The second uses 25 and the third uses 50. In each case the search was started from the same random image patch. Beside each graph is the final estimate of the optimal stimulus.

For comparison, Figure 6.2 shows three different reduced-dimensional searches. The starting point and block structures are identical. Notice that the reduced-dimensional search performs better, especially when using only a few trials per block. Notice also that the three different estimates of the optimal response vary somewhat even though the response has been nearly maximized. This is because the complex cell shows some invariance.

The main quantities for comparison in Figures 6.1 and 6.2 are how quickly and how close the responses approach the maximal response. The images of the maximizing stimuli are somewhat misleading. Projecting the noisy stimuli for the full-dimensional search onto 10-dimensional PCA space will produce smooth, edge-like stimuli. In the case of 50 trials per block, where the full-dimensional search closely approached the maximum, the 10-dimensional projection of the final estimate looks similar to the estimates from the reduced-dimensional search.

To further illustrate how this method behaves with invariance, we created an artificial neuron that responds to a T-junction and is invariant to rotation. The cell returns the maximum of 8 linear filters, each of which looks like a T-junction but at a different rotation. (As before, this is followed by a non-linearity and Poisson noise.) Applying the method using 3 different random starting points shows some of the different maximizing stimuli. Figure 6.3 shows the method using all 256 pixel dimensions. Figure 6.4 shows the same thing using the first 25 principal components. In each case we used 100 trials per block. This cell responds in a complicated way to fine features in the input. Because of this, we needed more principal components and more trials per block to get good performance. Unlike the previous example, projecting the stimuli found by the full dimensional search onto the first 25 principal components does not typically show anything resembling a T.

### 6.2.3 Research directions

These methods open up several exciting areas of research. One line of research focuses on further efficiency improvements. PCA is perhaps the simplest dimensionality reduction technique. Other bases like wavelets or curvelets [3] may work better in practice. The hierarchical nature of these bases opens up the possibility of coarse-to-fine searches which have the potential of dramatically improving efficiency. There may also be room for improvement in the gradient ascent method itself. For example, adjusting the step size and the number of trials per block in an adaptive way seem like useful ways to hone in on the optimal stimuli.

Another line of research focuses on modifications of the technique. These methods can be easily altered to search for stimuli other than the optimal one. For example, in a cell that shows baseline firing, we can search for the least-optimal stimulus, that is, the stimulus that inhibits the cell the most. We can also add time as an input dimension and look for optimal spatio-temporal stimuli. One experiment that particularly interests us involves invariance. We would first use gradient ascent to find the optimal stimulus. A simple modification of the method would then allow us to vary the baseline stimulus in the direction of least response variation. This would map out what might be called an *invariance ridge* for the cell. Other invariance properties could be explored in a similar manner.

A further avenue for investigation is where the methods are applied. V1 is an obvious choice, but the methods should be applicable to higher levels of visual cortex, like inferior temporal cortex (IT). Földiák et al. (2003) have demonstrated that online, adaptive stimulus presentation is possible even in higher levels of visual cortex. It should also be possible to apply these methods in auditory cortex, using auditory stimuli. These methods are even applicable for computational vision. Already algorithms like SFA are producing “neurons” whose response characteristics are difficult to determine and visualize. These methods can be immediately applied to the cells produced by complex computational vision algorithms in order to gain understanding about how these algorithms perform.

## 6.3 Stimulus-Response Function Approximation

The goal of our second approach is to find not just the maxima, but the entire stimulus-response function. This is impossible for arbitrary functions, but perhaps we can find a good fit of the true stimulus-response function within some restricted parametric class.

A simple example is the linear-nonlinear-Poisson (LNP) model class, where a neuron’s response function is characterized as a linear function of its inputs, followed by a nonlinear function. The output of the nonlinear function becomes the instantaneous firing rate of the neuron, where the neuron's firing is modeled as an inhomogeneous Poisson process. The goal is to estimate the linear function and the nonlinear function given the input stimulus and the output spiking process. A variety of techniques have been developed to address this problem. See for example, Simoncelli et al. (2004) [17].

Certain cells in V1 (simple cells) appear to be well approximated by the LNP model (although see [15]). Other cells in V1 (complex cells) and most cells in higher visual areas do not fit the LNP model. One way to extend the LNP model is called multidimensional LNP. In these models, the initial stage involves many linear filters. The Poisson firing rate is now a nonlinear combination of all of these filters. The techniques used for the simple LNP model can also be extended to handle the multidimensional LNP model, however, the analysis becomes more delicate and the number of stimulus presentations required increases dramatically [17].

Another way to extend the LNP model is to first transform the stimuli using a fixed, known collection of (nonlinear) functions and then apply the LNP model. This is a common technique for approaching nonlinear problems with linear methods, and is somewhat related to the Poisson regression used here and described in the Appendix. Again, the analysis becomes much more delicate. See Nykamp (2003) [12] for more details and for some interesting examples using more powerful models to quantify receptive field structure.

We believe that most, if not all of the current techniques used for neural response fitting can benefit from dimensionality reduction techniques like the ones demonstrated in the previous section. The extensions are obvious and we do not go into them here. Instead, we will describe another way to use the statistics of natural images that opens up exciting new possibilities for neural response fitting.

### 6.3.1 Filter response distributions

One of the difficulties of the LNP model is that both the linearity and the nonlinearity are unknown. Current techniques either try to estimate them simultaneously [13] or try to estimate the linearity (or linearities) first and then use this to infer the nonlinearity [17]. Not surprisingly, estimating the linearity in the presence of an unknown nonlinearity is difficult. Estimating more complicated models becomes even more difficult or perhaps impossible.

On the other hand, if somehow we had a good estimate of the nonlinearity, then the whole situation would be changed. Not only would it be straightforward to estimate linear models, but more complicated models would also be accessible. For example, the quadratic-nonlinear-Poisson (QNP) model used earlier could be estimated. In this hypothetical situation, only the quadratic part is unknown and can be easily fit using standard regression techniques. This is a very difficult model to estimate if the nonlinearity is not known. At first glance, it might seem impossible to estimate the nonlinearity first, but we think a surprising property of natural images will actually make this straightforward.

When a linear filter is applied to a random collection of natural image patches, the resulting distribution of filter responses often looks *sparse*, that is, it looks qualitatively similar to a double exponential distribution – symmetric, with a sharp peak and heavy tails [4]. This seems to be true for all zero-mean, local, linear filters. The reasons underlying this characteristic shape are not completely understood, but the phenomenon is remarkably robust.

A V1 simple cell is often approximated by an LNP model where the linear part is zero-mean and local. Thus, when presented with a collection of natural images, the output of the linear part (before the nonlinearity and the Poisson spike generation) will have a distribution that looks like a double-exponential, irrespective of the particular filter. We can use this to estimate the nonlinearity before estimating the linearity.

In fact, the method that we will propose does not rely on linearity in any way. The initial linear filter can be replaced by any (nonlinear) function of the input whose response to natural images shows this same characteristic double exponential distribution. In our experience, this includes several other models of visual neurons. It includes all linear models, as we mentioned, including overcomplete basis models like sparse coding and adaptive wavelets which seem linear but are actually nonlinear because of competition among units. It also includes units discovered by more modern techniques like slow feature analysis (SFA). Figure 6.5 shows the response distributions from two different types of functions applied to natural images. The second plot is the quadratic SFA cell used previously.

### 6.3.2 Estimating the (second) nonlinearity (first)

We want to generalize the LNP model to an NNP model – nonlinear-nonlinear-Poisson. We will call the first nonlinearity the response function and the second nonlinearity the rectifier. This reflects the intuitive notion that the response function characterizes how the neuron ideally responds to input and that the rectifier maps this ideal response into physiologically appropriate units, perhaps via a sigmoidal function. Of course, both functions are important for understanding the entire behavior of the neuron. Nonlinear rectification can drastically alter the properties of the response function.

The NNP model seems rather ill-defined. How do we distinguish between the two different types of nonlinearities? What is even the point of two nonlinearities? One will suffice. We can constrain things somewhat by requiring that the response nonlinearity has a specific distribution when presented with natural stimuli – namely, that the distribution is a double exponential. Without loss of generality, we can further assume that the distribution is mean 0 and variance 1, because centering and scaling constants can be incorporated into the rectifier. As we discussed earlier, several ideal models of response functions show distributions that are double-exponential like.

The NNP model is now constrained enough to estimate the rectifier using standard statistical techniques. We present a neuron with a random collection of natural images and use the expectation-maximization (EM) algorithm to approximate a maximum-likelihood estimate (MLE) of the entire nonlinear rectifier. Details can be found in the Appendix. Figure 6.6 shows the results of this estimating procedure applied to a simulated neuron. Each of the three plots shows a different number of stimulus presentations used for the estimation. The middle plot shows 1000 stimulus presentations and seems like a reasonable trade-off between goodness of fit and experimental duration.

The simulated neuron was the same quadratic SFA function used previously. Figure 6.7 shows the method applied to an LNP neuron with the same linear filter that was used for Figure 6.5. Note that the method is agnostic to the form of the response function, as long as its distribution looks double exponential on natural images. As shown in Figure 6.5 the distributions of these two cells are actually only approximately double exponential, but the method still works. One possible explanation for this robustness is that a sigmoidal-like rectifier, which seems physiologically reasonable, helps to mitigate the effects of outliers in the tails of the distributions. Another possible explanation is implicit smoothing in our approximation of the MLE. The true MLE of the rectifier is probably much less regular than the estimates we found.

### 6.3.3 Fitting the response function

Once the rectifier has been estimated, fitting a model to the remaining response function is conceptually simple. Theoretically, any model can be estimated (at least in the range over which the rectifier is not constant). Practically, the dimensionality of the model needs to be small enough to obtain a meaningful estimate. The same dimensionality reduction techniques that we advocated earlier can be used here.

For example, consider the simulated quadratic SFA cell used throughout. In the previous section we used natural stimuli to estimate the nonlinear rectifier. Now we present the cell with (artificial, noisy) stimuli and use standard Poisson regression techniques to estimate the parameters of the quadratic function. The Poisson regression techniques rely explicitly on our estimate of the nonlinear rectifier and cannot be used when the rectifier is not specified. Details can be found in the Appendix.

Figure 6.8 shows two examples of fitting the QNP model. The left example corresponds to the quadratic SFA unit used throughout. The right is another quadratic SFA unit. The parameters were estimated in 10-dimensional PCA space and the figures show the true parameters projected into this space. Figure 6.8 clearly shows that the qualitative properties of the parameters in the QNP model can be estimated, at least in these simulations. Quantitatively, the fit is not bad. The (normalized) inner products of the true and fitted parameter vectors are 0.9821 and 0.9409 for the left and right examples, respectively. More training examples (10000) improves the estimates until the inner product is essentially 1.

Figure 6.9 compares the true QNP model to the fitted QNP model on natural image data (not the training data). The fitted QNP model includes both the estimated rectifier and the estimated quadratic parameters. On natural images, the true and fitted cells behave quite similarly. Again, more training examples makes this fit almost perfect. Note that Figure 6.9 compares the mean response of the cells. Since this is only observed in practice through a Poisson process, which is quite noisy, these true and fitted units would be nearly indistinguishable with limited data.

### 6.3.4 Research directions

These preliminary simulations are promising and suggest several methods of possible improvement. The nonparametric rectifier estimation can probably be improved dramatically by switching to a parametric model. Not only will fewer training examples be required, but physiological experiments and biophysical theories may be able to provide insights into the form of the model. As far as fitting the response function, we have only used the simplest of methods. Regression is well understood and there are undoubtedly better methods for experimental design, estimation and validation.

The double exponential is only a crude approximation of the response distributions of linear and quadratic filters. Furthermore, other functions, like classical energy models of complex cells, have one-sided distributions that are better approximated by a (one-sided) exponential. Better models of the response distribution would improve our estimation of the rectifier and can easily be incorporated into the methods used here. It may even be possible to simultaneously estimate the rectifier and the response distribution if each is restricted to a small parametric class.

Modeling the entire stimulus / response function of a neuron provides a wealth of information about how the neuron behaves. This information can be used to investigate things like functional connectivity, which are crucial for understanding the computational strategies used by the brain. If this modeling program is successful, it will certainly create many more questions and directions for further research.

## 6.4 Mathematical Appendix

### 6.4.1 Response gradient approximation

Let $x$ be an $n$-dimensional vector and let $g : \mathbb{R}^n \to \mathbb{R}$ be any real-valued function of $x$. We want to approximate the gradient $\nabla g$ at a fixed point $\tilde{x}$. We do not know $g$ or $\nabla g$, but for any point $x$ we can observe independent realizations of a random variable $Y(x)$ with mean $g(x)$. For example, $x$ is an image patch, $g(x)$ is the (unobservable) average response of a neuron to that image patch and $Y(x)$ is the observed spike count in some window after a single presentation of the stimulus $x$.

Let $\Delta X$ be an $n$-dimensional random vector (noise) with mean 0 and nonsingular covariance matrix $\Sigma$. Let $\Delta Y = Y(\tilde{x} + \Delta X) - E[Y(\tilde{x} + \Delta X)]$ be the response to $\tilde{x} + \Delta X$, shifted to have 0 mean. ($E$ is expected value and in this case is taken over both $\Delta X$ and $Y$.) We claim that to a first order approximation

$$\nabla g(\tilde{x}) \approx \Sigma^{-1} E[\Delta X \Delta Y]. \tag{1}$$

The covariance $E[\Delta X \Delta Y]$ is an $n$-dimensional vector because $\Delta X$ is a vector and $\Delta Y$ is a scalar.

In practice, we create a sequence of independent realizations of $\Delta X$, say $\Delta X_1, \ldots, \Delta X_S$. We add this noise to the baseline stimulus $\tilde{x}$ and collect the sequence of responses $Y(\tilde{x} + \Delta X_1), \ldots, Y(\tilde{x} + \Delta X_S)$. We subtract the empirical mean

$$\Delta Y_s = Y(\tilde{x} + \Delta X_s) - \langle Y(\tilde{x} + \Delta X_t) \rangle_t$$

and use (1) to approximate

$$\nabla g(\tilde{x}) \approx \Sigma^{-1} \langle \Delta X_s \Delta Y_s \rangle_s .$$

(The empirical mean is $\langle c_s \rangle_s = S^{-1} \sum_{s=1}^{S} c_s$.) In the context of gradient ascent, we would then update the baseline stimulus $\tilde{x}$ by

$$\tilde{x} \mapsto \tilde{x} + \epsilon\Sigma^{-1} \langle \Delta X_s \Delta Y_s \rangle_s$$

for some small positive constant $\epsilon$. (If we take $\epsilon < 0$, then this is gradient descent.) In the simulations in the text we take $\epsilon = .1$ and $\Sigma = .25I$, where $I$ is the identity matrix.

In many situations is makes more sense to perform an online search subject to some constraint. In this case we follow each gradient ascent by a projection back into the constrain space [16]. For example, in the simulations in the text we did gradient ascent subject to a constant norm (intensity) constraint on the stimulus. So we updated the baseline stimulus as before but then projected back to the appropriate norm:

$$\begin{aligned} \tilde{x} &\mapsto \tilde{x} + \epsilon\Sigma^{-1} \langle \Delta X_s \Delta Y_s \rangle_s \\ \tilde{x} &\mapsto C\frac{\tilde{x}}{\|\tilde{x}\|}. \end{aligned}$$

The norm constraint was $C = 2.0030$, which was the average norm from the (centered) PCA training data.

**Proof of (1).** This calculation is outlined in [5]. The formula is based on the first order, linear approximation of $g$ as

$$g(x + \Delta x) \approx g(x) + \Delta x^T \nabla g(x), \tag{2}$$

where $\cdot^T$ denotes transpose. Throughout $x$ is fixed and known.

Using this we first approximate

$$\begin{aligned} E[Y(x + \Delta X)] &= E[E[Y(x + \Delta X)|\Delta X]] = E[g(x + \Delta X)] \\ &\approx E[g(x) + \Delta X^T \nabla g(x)] = g(x) + E[\Delta X^T]\nabla g(x) = g(x), \end{aligned} \tag{3}$$

since $\Delta X$ has mean 0. Now we have

$$\begin{aligned} E[\Delta X \Delta Y] &= E[E[\Delta X \Delta Y|\Delta X]] = E[\Delta X E[\Delta Y|\Delta X]] \\ &\approx E[\Delta X E[Y(x + \Delta X) - g(x)|\Delta X]] = E[\Delta X(g(x + \Delta X) - g(x))] \\ &\approx E[\Delta X \Delta X^T \nabla g(x)] = \Sigma \nabla g(x), \end{aligned}$$

where the first approximation comes from (3) and the second from (2). Multiplying by $\Sigma^{-1}$ gives (1). □

### 6.4.2 Rectifier estimation

Let $X$ be a random $n$-dimensional vector with unknown distribution and $r : \mathbb{R} \to \mathbb{R}$ be an unknown function. Suppose however that $Z = r(X)$ has a known distribution, say with density $p_Z$. In the text we take this distribution to be double exponential distribution with mean 0 and variance 1, that is

$$p_Z(z) = \frac{1}{\sqrt{2}} e^{-\sqrt{2}|z|}, \quad -\infty < z < \infty,$$

but here the specific form of $p_Z$ is not important. We are given a sequence of i.i.d. r.v.'s $X_1, \ldots, X_S$ with the same distribution as $X$. We also get to observe Poisson counts from an NNP model, namely $Y_1, \ldots, Y_S$ where $Y_s$ is a Poisson random variable with mean $f(r(X_s)$. We want to estimate the (rectifier) function $f : \mathbb{R} \to [0, \infty)$, which is unknown. As we mentioned, we do not know $r$, but we know the distribution of $Z = r(X)$.

Let $Z_s = r(X_s)$. We will completely ignore our knowledge of the $X_s$ (this is obscured by $r$ anyway) and use only the fact that $Z_s$ has known density $p_Z$. The model becomes: $Y_1, \ldots, Y_S$ are independent Poisson random variables and $Y_s$ has mean $f(Z_s)$ for unknown $f$ and unobserved $Z_1, \ldots, Z_S$. Since we know the distribution of the $Z_s$ we can use maximum likelihood estimation to estimate $f$. The log likelihood is

$$\begin{aligned}\log p(Y_1, \ldots, Y_S | f) &= \sum_{s=1}^{S} \log p(Y_s | f) = \sum_{s=1}^{S} \log \int_{\mathbb{R}} p(Y_s | Z_s = z; f) p_Z(z) dz \\ &= \sum_{s=1}^{S} \log \int_{\mathbb{R}} \frac{e^{-f(z)} f(z)^{Y_s}}{Y_s!} p_Z(z) dz. \qquad (4)\end{aligned}$$

It is not clear that this can be maximized analytically.

Instead, we frame this as a missing data problem – the $Z_s$ are missing – and use the expectation maximization (EM) algorithm (see McLachlan and Krishnan, 1997, for details and references [10]). The EM update equation is

$$f_{k+1}(z) = \frac{\sum_{s=1}^{S} Y_s p_Z(z | Y_s; f_k)}{\sum_{s=1}^{S} p_Z(z | Y_s; f_k)}. \qquad (5)$$

This calculation is detailed below. The functions $p_Z(z|Y_s; f_k)$ can be determined using Bayes' Rule

$$p_Z(z | Y_s; f_k) = \frac{p(Y_s | Z_s = z; f_k) p_Z(z)}{\int_{\mathbb{R}} p(Y_s | Z_s = \tilde{z}; f_k) p_Z(\tilde{z}) d\tilde{z}} = \frac{e^{-f_k(z)} f_k(z)^{Y_s} p_Z(z)}{\int_{\mathbb{R}} e^{-f_k(\tilde{z})} f_k(\tilde{z})^{Y_s} p_Z(\tilde{z}) d\tilde{z}}. \qquad (6)$$

All of these calculations can be carried out on a grid (in $z$) over the effective range of $p_Z$, which is known. The computations can be sped up significantly by taking advantage of the multiplicities of the $Y_s$ which are Poisson counts. For example, the integrals in the denominator of (6) need only be evaluated for each distinct value of the $Y_s$. If the distinct values of the $Y_s$ are $Y_{(1)}, \ldots, Y_{(M)}$, with multiplicities $N_1, \ldots, N_M$, then (5) becomes

$$f_{k+1}(z) = \frac{\sum_{m=1}^{M} N_m Y_{(m)} p_Z(z | Y_{(m)}; f_k)}{\sum_{m=1}^{M} N_m p_Z(z | Y_{(m)}; f_k)}.$$

For the simulations in the text, we take $p_Z$ to be a double exponential and we initialize the EM algorithm with

$$f_1(z) = \begin{cases} \alpha & \text{if } z \le -10 \\ (\beta - \alpha)(z+10)/20 + \alpha & \text{if } -10 < z < 10 \\ \beta & \text{if } z \ge 10 \end{cases},$$

which is linear over the effective range of $p_Z$ and then held constant outside of that range. $\alpha$ and $\beta$ are determined from the data. We take $\alpha$ to be the 5th percentile of the $Y_s$ and $\beta$ to be the 95th percentile. We estimate $f$ on a .1 grid from $-10$ to 10.

The EM algorithm is run until the successive estimates of $f$ are not changing much at any point on the grid, specifically, until $\max_z |f_{k+1}(z) - f_k(z)| < .01$ or 100 iterations, which ever comes first (usually the former). The entire process typically takes under 1/2 second on a desktop PC. The stopping criterion for the EM algorithm appears to be crucial in this context. The MLE estimate of $f$ is probably not very smooth. Stopping the EM algorithm earlier than computationally necessary effectively introduces some smoothing into the estimate. This is what we did in the text. Allowing the EM algorithm to iterate longer generates worse estimates of $f$. It is likely that an explicit regularization term or a parameterized model of $f$ would help to create better and more robust estimates than those used here.

In our experience, a single large value for one of the $Y_s$ can raise the far right side of the estimated $f$ much higher than the true rectifier. This is easy to spot visually because of a discontinuity and can be remedied by removing the outlier from the data. Outlier removal could be automated, but may not be necessary in practice because of physiological upper limits on the number of spikes that can occur in a given time window. These are the sorts of issues that cannot be adequately addressed by simulation experiments.

**Proof of (5).** The complete data log likelihood is

$$\log p(Y_1, Z_1, \ldots, Y_S, Z_S|f) = \sum_{s=1}^{S} \log p(Y_s, Z_s|f) = \sum_{s=1}^{S} \log \left[ \frac{e^{-f(Z_s)} f(Z_s)^{Y_s}}{Y_s!} p_Z(Z_s) \right]$$

$$= \sum_{s=1}^{S} \big[ -f(Z_s) + Y_s \log f(Z_s) - \log Y_s! + \log p_Z(Z_s) \big]. \tag{7}$$

The EM algorithm creates a sequence of estimates $f_1, f_2, \ldots$ that increase the likelihood in (4). Given $f_k$ we find $f_{k+1}$ by maximizing over $f$ the expected value of (7) given the observations $Y_1, \ldots, Y_S$ and using $f_k$. This conditional expectation is

$$E\big[\log p(Y_1, Z_1, \ldots, Y_S, Z_S|f) \big| Y_1, \ldots, Y_S; f_k\big]$$
$$= \sum_{s=1}^{S} E\big[ -f(Z_s) + Y_s \log f(Z_s) - \log Y_s! + \log p_Z(Z_s) \big| Y_s; f_k \big]$$
$$= \sum_{s=1}^{S} \int_{\mathbb{R}} \big[ -f(z) + Y_s \log f(z) - \log Y_s! + \log p_Z(z) \big] p_Z(z|Y_s; f_k) dz.$$

To maximize this over $f$ we can ignore the parts that do not depend on $f$ and choose

$$f_{k+1} = \arg\max_f \sum_{s=1}^{S} \int_{\mathbb{R}} \big[-f(z) + Y_s \log f(z)\big] p_Z(z|Y_s; f_k) dz. \tag{8}$$

The argument of (8) is concave in $f$ because of the concavity of the logarithm, so any critical point of the (functional) derivative will be a global maximizer. Perturbing $f$ by $f+\epsilon\eta$ for an arbitrary function $\eta$, taking the derivative w.r.t. $\epsilon$, evaluating at $\epsilon = 0$ and setting the result equal to zero gives the following equation for critical points:

$$\sum_{s=1}^{S} \int_{\mathbb{R}} p_Z(z|Y_s; f_k) \left[-\eta(z) + Y_s \frac{\eta(z)}{f_{k+1}(z)}\right] dz = 0 \quad \text{for all functions } \eta,$$

or equivalently,

$$\int_{\mathbb{R}} \eta(z) \sum_{s=1}^{S} p_Z(z|Y_s; f_k) \left[\frac{Y_s}{f_{k+1}(z)} - 1\right] dz = 0 \quad \text{for all functions } \eta. \tag{9}$$

Since $\eta$ is arbitrary, we must have

$$\sum_{s=1}^{S} p_Z(z|Y_s; f_k) \left[\frac{Y_s}{f_{k+1}(z)} - 1\right] = 0 \quad \text{for all } z,$$

which has the unique solution given in (5) (except for a few pathological cases like when all the $p_Z(z|Y_s; f_k)$ are 0 for some $z$ and then $f_{k+1}(z)$ can be anything). $\square$

### 6.4.3 Poisson regression

Poisson regression is well studied [11]. The simplest form is that the observations $Y_s$ are independent Poisson random variables with mean $f(\beta^T h(X_s))$, where $\beta^T = (\beta_0, \beta_1, \dots, \beta_n)$ is the vector of parameters to be estimated, $X_s$ is the $s$th stimulus and

$$h(X_s) = (h_0(X_s), h_1(X_s), \dots, h_n(X_s))^T$$

is the vector of predictor variables for the $s$th stimulus. The functions $f$ and $h$ are known.

Typically $h_0 \equiv 1$ and allows a constant term to enter the model. If $X_s$ is a vector (like pixels in a receptive field), then $h_j$ can be $X_{sj}$, the $j$th element of $X_s$. This corresponds to the LNP model. Often, there will be many more predictor variables than stimulus dimensions. For example, in the QNP model $Y_s$ has mean $f(X_s^T A X_s + X_s^T B + C)$ for matrix $A$, vector $B$ and constant $C$. We can write this as $f(\beta^T h(X_s))$ by including not only the elements of $X_s$ in $h$ but also all of the interaction terms like $h_\ell(X_s) = X_{sj} X_{sk}$. The entries in $\beta$ will thus correspond to certain elements of $A$, $B$ or $C$.

Since $h$ is known and $X_s$ is observed, we can write $Z_s = h(X_s)$, where $Z_s$ is a vector, and forget about $h$ and $X_s$. $Y_s$ has mean $f(\beta^T Z_s)$. The log likelihood equation is

$$\log p(Y_1, \ldots, Y_S|\beta) = \sum_{s=1}^{S} \log p(Y_s|\beta) = \sum_{s=1}^{S} \log \frac{e^{-f(\beta^T Z_s)} f(\beta^T Z_s)^{Y_s}}{Y_s!}$$
$$= \sum_{s=1}^{S} \left[ -f(\beta^T Z_s) + Y_s \log f(\beta^T Z_s) - \log Y_s! \right].$$

Since $f$, the $Y_s$ and the $Z_s$ are known, this can be maximized with standard nonlinear optimization tools. The gradient vector is easy to calculate and this can be used to speed the convergence. The gradient vector requires an estimate of the derivative of the rectifier $f$. For this we just approximated $f'(z) \approx (f(z + \Delta) - f(z - \Delta))/(2\Delta)$ at grid points $z$, where $\Delta$ is the grid resolution. We took both $f$ and $f'$ to be linear between grid points.

For the simulations in the text, we initialized the optimization at $\beta^T = (1, 0, \ldots, 0)$, that is, only a constant term. Using Matlab's unconstrained nonlinear optimization typically took under 2 minutes for the 65-parameter QNP problem.

For stimuli, we first chose standard Gaussian (white) noise in the 10-dimensional PCA parameter space. To get a better sampling of the input space, we self-normalized each noise vector and then scaled by a random amount (Gaussian, mean 0, standard deviation 3). Fitting with white noise or with natural stimuli seemed to work, but not quite as well as with this method. There are probably much better experimental design methods in the literature, but we did not investigate this possibility.

# Bibliography

[1] Anthony J. Bell and Terrence J. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338, 1997.

[2] Pietro Berkes and Laurenz Wiskott. Slow feature analysis yields a rich repertoire of complex-cell properties. Cognitive Sciences EPrint Archive (CogPrints) 2804, http://cogprints.ecs.soton.ac.uk/archive/00002804/ (12/08/2003), February 2003.

[3] David L. Donoho and Ana Georgina Flesia. Can recent innovations in harmonic analysis 'explain' key findings in natural image statistics? *Network: Computation in Neural Systems*, 12(3):371–393, 2001.

[4] D. J. Field. What is the goal of sensory coding? *Neural Computation*, 6:559–601, 1994.

[5] Peter Földiák. Stimulus optimisation in primary visual cortex. *Neurocomputing*, 38–40:1217–1222, 2001.

[6] Peter Földiák, Dengke Xiao, Christian Keyers, Robin Edwards, and David Ian Perrett. Rapid serial visual presentation for the determination of neural selectivity in area STSa. *Progress in Brain Research*, 144:107–116, 2003.

[7] Matthew Harrison, Stuart Geman, and Elie Bienenstock. Using statistics of natural images to facilitate automatic receptive field analysis. APPTS #04-2, Brown University, Division of Applied Mathematics, Providence, RI, February 2004.

[8] Patrik O. Hoyer and Aapo Hyvärinen. A multi-layer sparse coding network learns contour coding from natural images. *Vision Research*, 42:1593–1605, 2002.

[9] Aapo Hyvärinen and Patrik O. Hoyer. Topographic independent component analysis as a model of V1 organization and receptive fields. *Neurocomputing*, 38–40:1307–1315, 2001.

[10] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. John Wiley & Sons, New York, 1997.

[11] John Neter, Michael H. Kutner, Christopher J. Nachtsheim, and William Wasserman. *Applied Linear Regression Models*. Irwin, Chicago, 3rd edition, 1996.

[12] Duane Q. Nykamp. Measuring linear and quadratic contributions to neuronal response. *Network: Computation in Neural Systems*, 14:673–703, 2003.

[13] Duane Q. Nykamp and Dario L. Ringach. Full identification of a linear-nonlinear system via cross-correlation analysis. *Journal of Vision*, 2:1–11, 2002.

[14] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[15] Nicole C Rust, Odelia Schwartz, J Anthony Movshon, and Eero Simoncelli. Spike-triggered characterization of excitatory and suppressive stimulus dimensions in monkey V1. *Neurocomputing*, 2004 (to appear).

[16] L. E. Scales. *Introduction to Non-Linear Optimization*. Springer-Verlag, New York, 1985.

[17] Eero P. Simoncelli, Liam Paninski, Jonathan Pillow, and Odelia Schwartz. Characterization of neural responses with stochastic stimuli. In M. Gazzaniga, editor, *The New Cognitive Sciences*. MIT Press, 3rd edition, 2004 (to appear).

[18] Laurenz Wiskott and Terrence J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14:715–770, 2002.

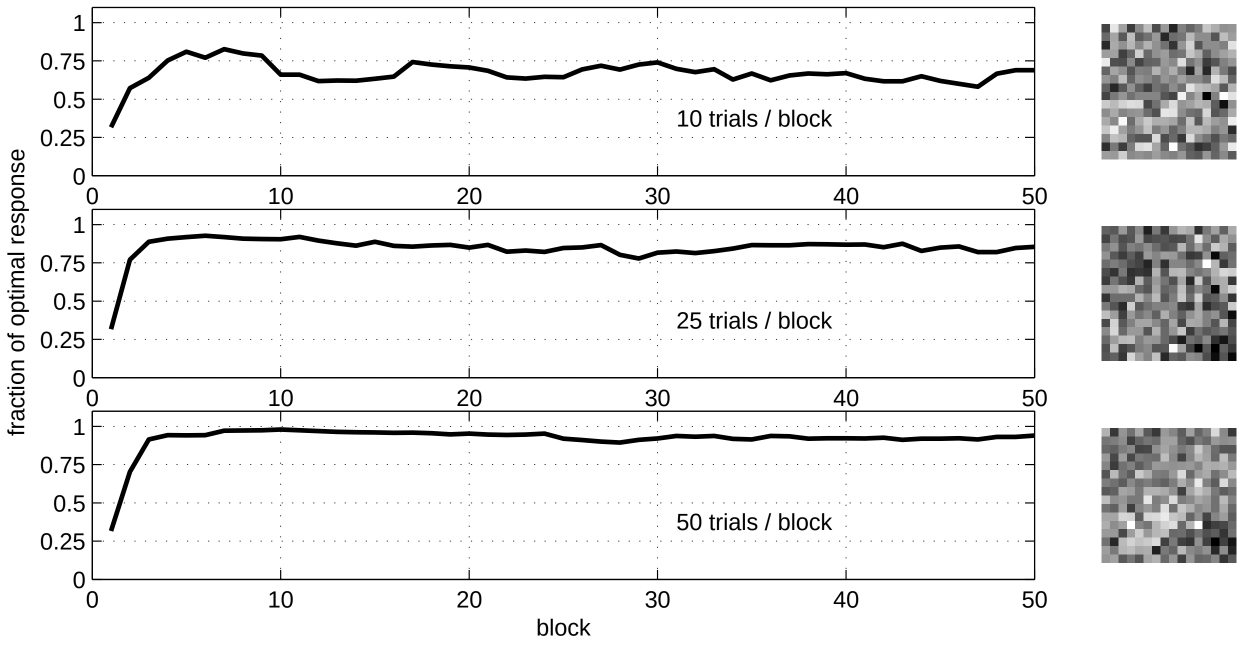

Figure 6.1: Full-dimensional search.

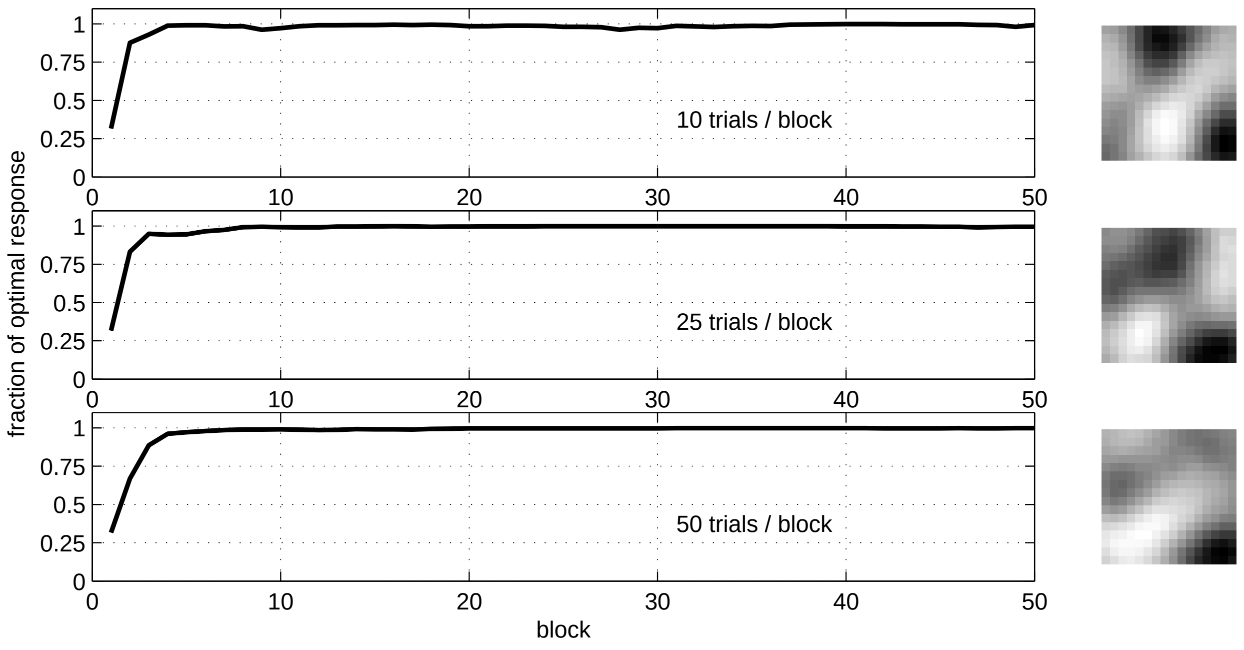

Figure 6.2: Reduced-dimensional search.

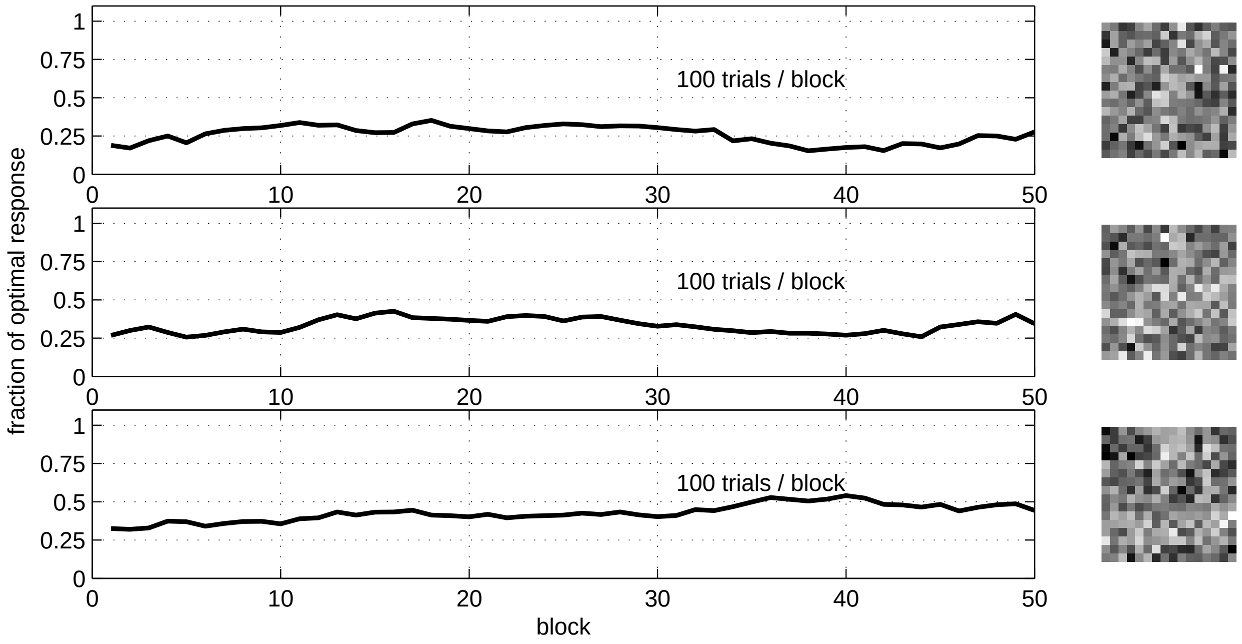

Figure 6.3: Full-dimensional search.

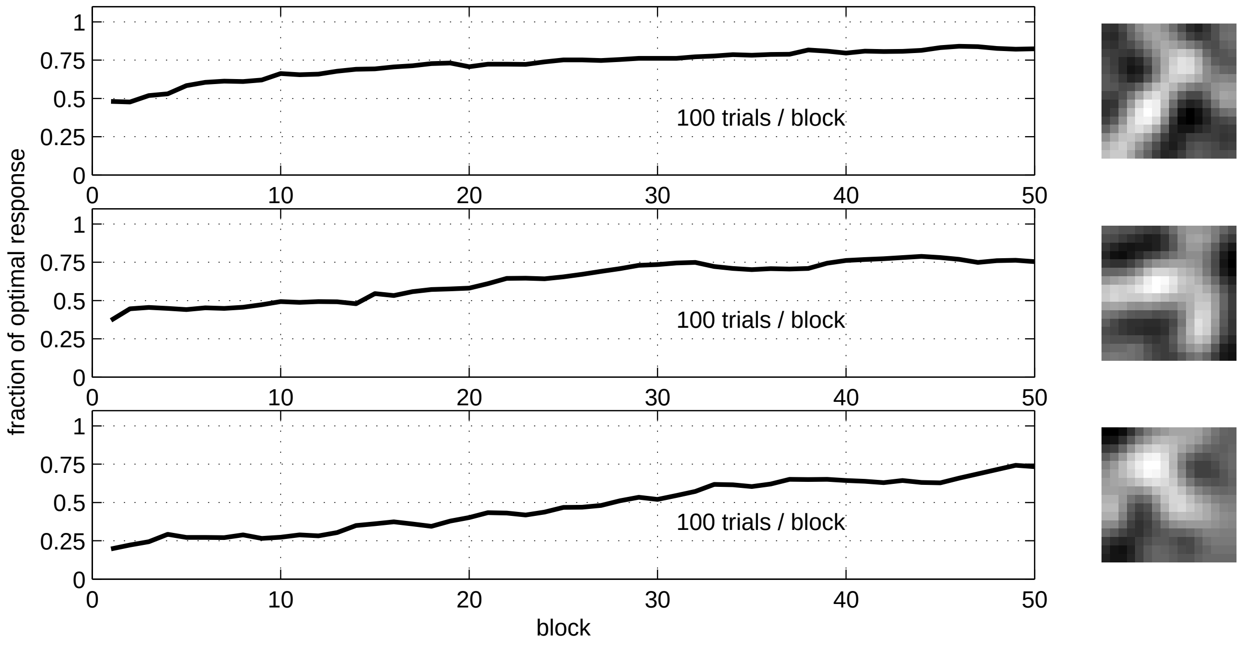

Figure 6.4: Reduced-dimensional search.

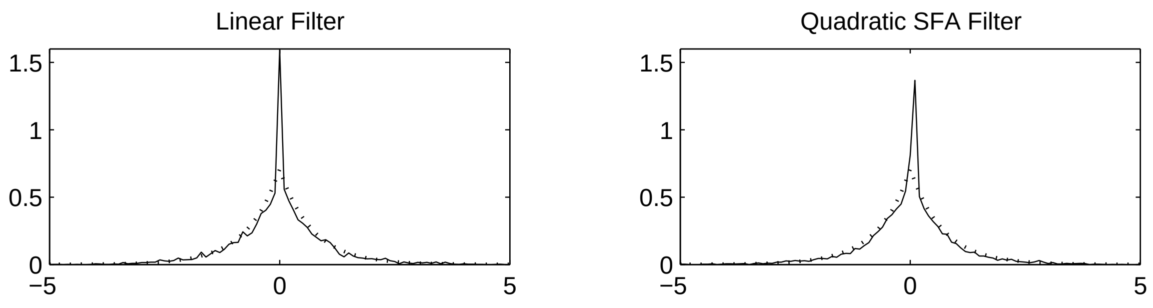

Figure 6.5: Solid line: filter distributions over natural images normalized to mean 0 and variance 1. Dotted line: double exponential distribution.

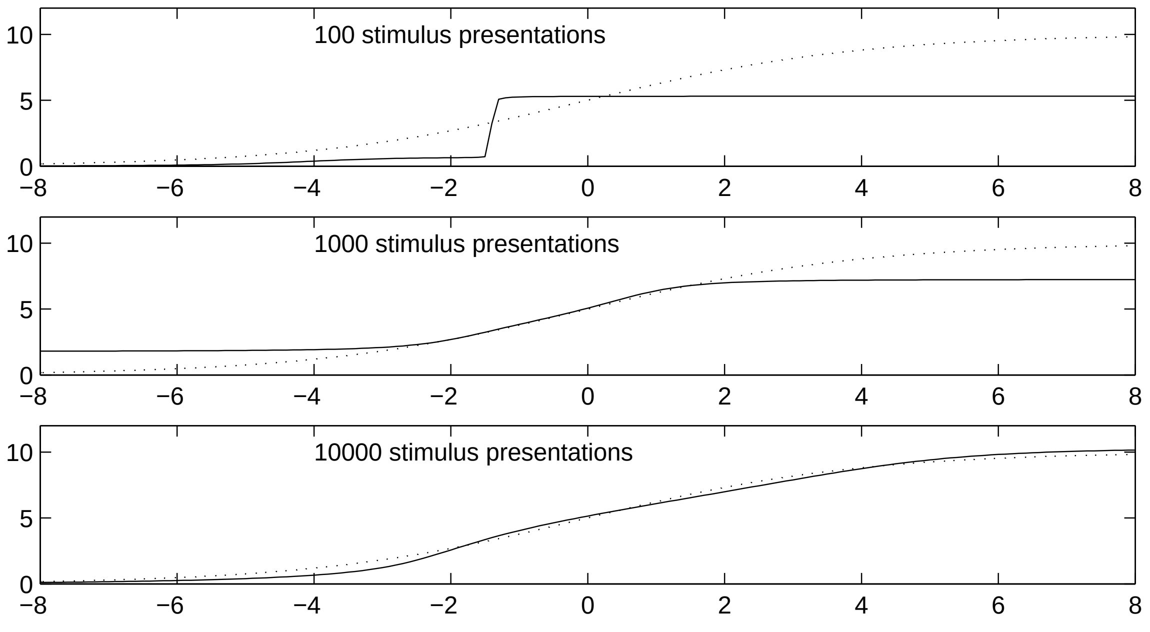

Figure 6.6: Quadratic SFA unit. Solid line: fitted rectifier. Dotted line: true rectifier.

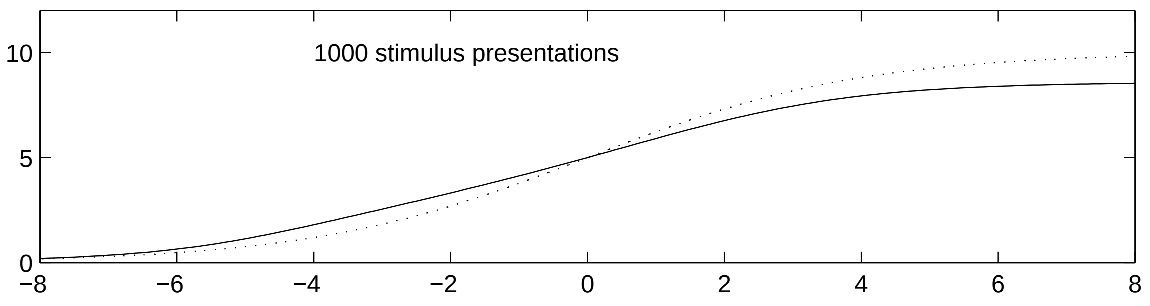

Figure 6.7: Linear Unit. Solid line: fitted rectifier. Dotted line: true rectifier.

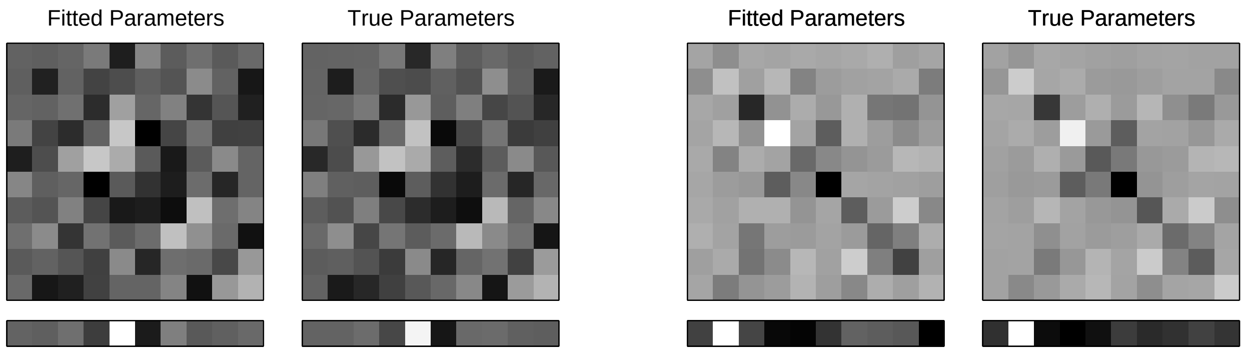

Figure 6.8: Fitted versus true parameters in the quadratic model. The upper, square images show the A matrices in the quadratic form. The lower, rectangular images show the B vectors. The left two plots are from the SFA unit used in the text. The right two are from another quadratic SFA unit. The true parameters have been projected into the 10-dimensional PCA space for easy comparison with the fitted parameters.

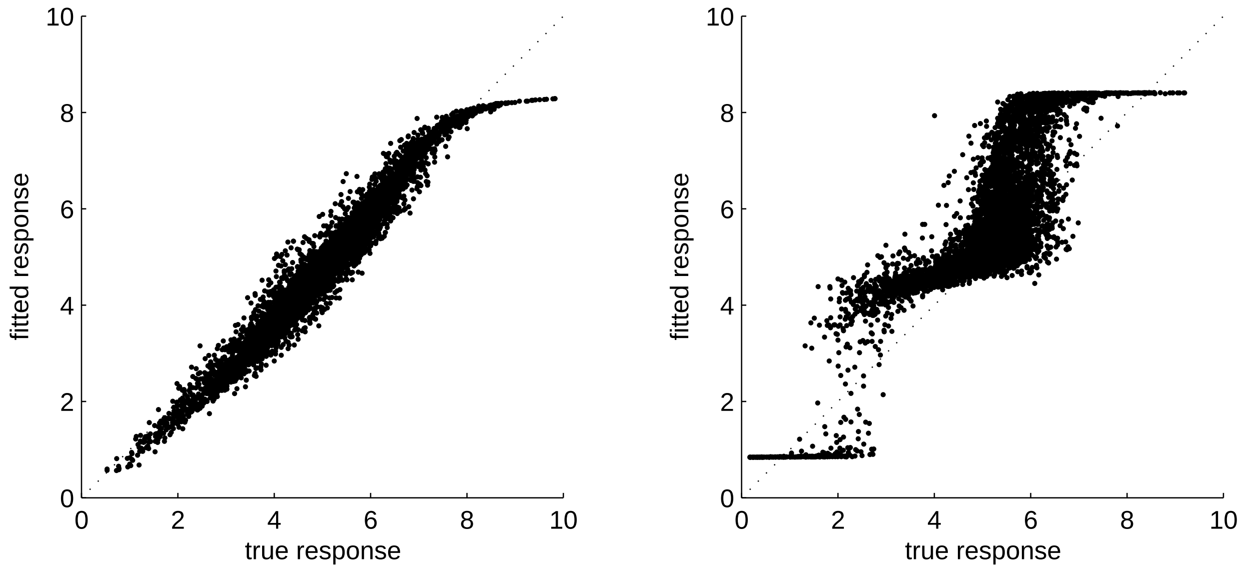

Figure 6.9: Comparison of true and fitted models on natural images. The left and right models are the same as in Figure 6.8. If the models were identical, all the points would lie on the diagonal dotted line because these plots show the mean (ideal) response, not the noisy Poisson observations. The estimated rectifier in the model on the right was wavy, unlike the true rectifier.

# Chapter 7

# Jitter methods

## 7.1 Introduction

We are given a neural spike train with spike times $t_1, \ldots, t_n$ and we want to discover something about the temporal resolution over which these spike times were generated. *Jitter methods* [4, 10, 1] are one way to begin approaching this problem. They are all based on the following intuitive approach:

- Use the observed spike train to randomly generate many "$\Delta$-similar" spike trains by locally perturbing individual spike times. "$\Delta$-similar" is not a precise term, but loosely means that they do not differ much on time scales greater than $\Delta$.

- See if the observed spike train looks unusual within the population of $\Delta$-similar spike trains. If so, then perhaps it has temporal structure at a finer resolution than $\Delta$.

Jitter methods are not intended to give exact estimates of $\Delta$. Except in certain specific (and non-biological) examples, the "temporal resolution of structure in a spike train" is probably not a well-defined quantity. Jitter methods are better interpreted as qualitative and exploratory. $\Delta$ is a knob that can be turned. We know that for large $\Delta$, say 1 second, the classical firing rates are not preserved, so spike trains have structure at resolutions finer than 1 second. We also know that for small $\Delta$, say 1 microsecond, the timing of an action potential is not well-defined, so (almost by definition) individual spike trains do not have structure on these time scales. So where is the transition from structure to no structure? 100–200 milliseconds? 10–20 ms? 1–2 ms? These sorts of questions have recently become hotly debated issues as competing theories of neural processing demand widely differing degrees of spike timing precision.

Jitter methods effectively operate by locally perturbing individual spikes. This necessarily preserves classical firing rates over time scales longer than the amount of perturbation and is one of the main advantages of using jitter methods. Most other methods try to preserve classical firing rates (or some other quantity) by using the assumption that these rates do not vary across similar experimental trials. Unfortunately, this assumption is almost certainly too strong and can introduce significant artifacts [2, 3].

### 7.1.1 Organization of the chapter

This chapter is loosely divided into three parts. The first part is Section 7.2 which contains a technical description of the main computational techniques at the core of exact jitter methods. It is basically a mathematical appendix. It was placed first because it is referred to throughout the text, but can probably be skipped on an initial reading. Except for Sections 7.2.1.5–7.2.1.6 and 7.2.2.8–7.2.2.9 on tail probabilities, Section 7.2 is essentially nothing more than a description of dynamic programming tailored for this context.

The second part of the chapter describes several different jitter methods. The organization proceeds from simple to complex, with each new method correcting a specific deficiency in the previous ones. Sections 7.3, 7.4 and 7.5 describe the three main methods. Section 7.6 describes some extensions that address things like refractory period and bursting, which are examples of fine temporal structure that we might want to ignore. One of the recurring themes is that Monte Carlo methods, which drive the intuition, can be replaced by fast and exact methods based on dynamic programming.

The third part of the chapter is a collection of distinct jitter-related topics. Section 7.7 gives a method for sampling spike trains that contain certain fine-temporal structures. Sampling is necessary when exact methods are not applicable. Sampling is also useful for verifying and modifying the parameters in jitter methods. Section 7.8 considers the specific case of using synchrony between simultaneously recorded spike trains in the context of jitter methods. Section 7.9 describes some experiments with the jitter methods. Sections 7.10 and 7.11 conclude the chapter.

### 7.1.2 Notation

Specific notation is described throughout the text, but there are a few general trends. $|A|$ is the number of elements in the set $A$. $\mathbb{1}\{A\}$ is the indicator function for the set $A$, i.e. $\mathbb{1}\{A\} = 1$ if $A$ is true and 0 otherwise. $A \oplus B = \{c : c = a + b, a \in A, b \in B\}$ denotes setwise addition. Here we use the notation $f(n) = O(g(n))$ to mean that $|f(n)| \leq Cg(n) + B$ for all $n$ ($n$ can be vector valued) and for some constants $C$ and $B$, independent of $n$.

We use Matlab indexing notation in a variety of different situations. For example, $\omega_{j:k} = (\omega_j, \omega_{j+1}, \ldots, \omega_k)$ in the context of sequences, $\Omega_{j:k} = \Omega_j \times \Omega_{j+1} \times \cdots \times \Omega_k$ in the context of sequence spaces and $P_{j:k}(\omega_{j:k}) = \prod_{i=j}^{k} P_i(\omega_i)$ in the context of product distributions on a sequence space. All the different usages should be clear from the context. We also tend to use subscripts for indexing spikes or windows or iterations in a recursive algorithm (these usually turn out to be the same thing) and we use brackets $[\cdot]$ for indexing into a vector or a matrix. So we have notation like $\beta_k[i, j]$, which is the $(i, j)$th element of a matrix $\beta_k$, where $k$ refers to something like the $k$th spike, or like $\mu_\ell[1 : n]$, which is the first $n$ elements of a vector $\mu_\ell$. Occasionally we use superscripts, usually to make suppressed notation explicit or to index Monte Carlo samples.

There is one situation where our notation is somewhat ambiguous. We often use the same notation to describe mathematics and computation. For example, suppose $X$ is a function on a finite set $\Omega$. We will use function notation for mathematics: $X(\omega)$. But for describing algorithms it is more convenient to think of $\Omega$ and $X$ as vectors: $X[i] = X(\Omega[i])$, where $\Omega[i]$ is the $i$th element of $\Omega$.

## 7.2 Computational Components

The exact jitter methods are all based on convolutions of probability distributions. This section describes a few different varieties of convolution that we will repeatedly use. We only deal with discrete state spaces.

### 7.2.1 Independent convolution

Let $X_i$ ($i = 1:n$) be a collection of independent random variables with distributions $P_{X_i}$. Define the new random variable $Z_{j:k} = \sum_{i=j}^{k} X_i$, where we take $Z_{j:k} = 0$ if $k < j$. We can use convolutions to compute the distribution of $Z_{j:k}$, namely

$$P_{Z_{j:k}} = P_{X_j} * \cdots * P_{X_k}.$$

If $k = j$, we have $P_{Z_{j:k}} = P_{X_j}$, and if $k < j$, we have $P_{Z_{j:k}} = \delta_0$, the point mass at zero.

Just as summations can be carried out in any order, convolutions can performed in any order, such as

$$\begin{aligned} P_{Z_{j:k}} &= P_{Z_{j:i_1}} * P_{Z_{i_1+1:i_2}} * \cdots * P_{Z_{i_{m-1}+1:i_m}} * P_{Z_{i_m+1:k}} \\ &= \left(P_{X_j} * \cdots * P_{X_{i_1}}\right) * \left(P_{X_{i_1+1}} * \cdots * P_{X_{i_2}}\right) * \cdots \\ &\quad * \left(P_{X_{i_{m-1}+1}} * \cdots * P_{X_{i_m}}\right) * \left(P_{X_{i_m+1}} * \cdots * P_{X_k}\right) \end{aligned}$$

for any sequence $j \leq i_1 \leq \cdots \leq i_m \leq k$.

Because of this, we need only describe how to compute a single convolution, say $P_Z = P_X * P_Y$. The computation can be applied recursively to compute multiple convolutions. The well known formula for a single convolution is

$$P_Z(z) = (P_X * P_Y)(z) = \sum_y P_X(z - y) P_Y(y).$$

If $A_X$ and $A_Y$ are the supports for $P_X$ and $P_Y$, respectively, then we can restrict the summation to $y \in A_Y$ and we need only consider $z \in A_X \oplus A_Y$.

#### 7.2.1.1 A convolution algorithm

We want to compute $P_Z = P_X * P_Y$. To represent $P_X$ we need a pair of identical length vectors, $A_X[1:L_X]$ and $P_X[1:L_X]$ for the support and distribution, respectively. This dual notation of thinking about $P_X$ as both a distribution and a vector is convenient. They are related by $P_X[i] = P_X(A_X[i])$, where $A_X[i]$ is the $i$th element of the set $A_X$, for $i = 1, \ldots, L_X = |A_X|$. Similarly, we can represent $P_Y$ with $A_Y[1:L_Y]$ and $P_Y[1:L_Y]$, and $P_Z$ with $A_Z[1:L_Z]$ and $P_Z[1:L_Z]$.

An easy convolution algorithm, which we denote by

$$(A_Z, P_Z) = \texttt{conv}(A_X, P_X, A_Y, P_Y),$$

is the following: We first compute the $L_X \times L_Y$ matrices $A[i,j] = A_X[i] + A_Y[j]$ and $P[i,j] =$

$P_X[i]P_Y[j]$. We determine $A_Z[1\!:\!L_Z]$ by all of the unique elements in $A$. For each $\ell$ we add up those $P[i,j]$ for which $A[i,j] = A_Z[\ell]$ to get $P_Z[\ell]$. Finally, we prune any $\ell$ for which $P_Z[\ell] = 0$. Pruning allows the algorithm to handle the situation where $A_X$ and $A_Y$ are supersets of the supports for $P_X$ and $P_Y$, while still returning the support $A_Z$ of $P_Z$.

Many of these steps can be combined in an actual implementation. Computing $A$ and $P$ requires $O(L_X L_Y)$ operations and takes $O(L_X L_Y)$ elements of storage. Computing $A_Z$ and finding the $(i,j)$ pairs for which $A[i,j] = A_Z[\ell]$ essentially requires sorting $A$ which takes something like $O(L_X L_Y \log L_X L_Y)$ operations depending on the sorting algorithm. Several situations can dramatically reduce the computation and storage requirements. For example, if $A_Z$ (or a small superset of it) is known ahead of time, particularly if it is a set of small integers, then it is not necessary to store and sort the elements of $A$.

#### 7.2.1.2 Convolving random variables

Sometimes it is more convenient to represent a random variable $X$, not with its support and distribution $(A_X, P_X)$, but as a triple $(\Omega, U, X)$, where $\Omega$ is some (finite) state space, $U$ is a probability distribution over $\Omega$ and $X$ is a real-valued function on $\Omega$. The distribution $P_X$ of $X$ is defined by

$$P_X(x) = U\{\omega : X(\omega) = x\} = \sum_{\omega\in\Omega : X(\omega)=x} U(\omega).$$

Let $X_i$ $(i = 1 : n)$ be a collection of independent random variables $(\Omega_i, U_i, X_i)$. By independent, we mean that the $U_i$ are the marginals of a product distribution $\prod_i U_i$ over the product space $\times_i \Omega_i$. To compute the distribution of $Z_{j:k} = \sum_{i=j}^{k} X_k$ we can first compute the distributions $P_{X_i}$ and then convolve them to get $P_{Z_{j:k}}$. Unfortunately, this does not extend to the other types of convolution that we consider, so we introduce the notion of convolving a random variable (instead of convolving its distribution).

If $Y$ is a random variable with distribution $P_Y$ and $X$ is a random variable $(\Omega, U, X)$, we define the convolution $P_Y * X$ by

$$(P_Y * X)(z) = \sum_{\omega\in\Omega} P_Y(z - X(\omega))U(\omega).$$

Let $Z = X + Y$. Note that

$$\begin{aligned} P_Z(z) = (P_Y * P_X)(z) &= \sum_x P_Y(z-x)P_X(x) = \sum_x \sum_{\omega : X(\omega)=x} P_Y(z - X(\omega))U(\omega) \\ &= \sum_\omega P_Y(z - X(\omega))U(\omega) = (P_Y * X)(z), \end{aligned}$$

so there is nothing different about this type of convolution. It just emphasizes that we are not explicitly using the distribution $P_X$ of $X$. Note that $\delta_0 * X = P_X$, so we can define $X * Y = (\delta_0 * X) * Y$. We also define $X * P_Y = P_Y * X$.

The original `conv` algorithm described in the previous section also works for convolving a distribution with a random variable or for convolving two random variables. For the random variable $(\Omega, U, X)$ we use the vectors $(X, U)$ instead of $(A_X, P_X)$. The vectors $(X, U)$ are related to the random variable $(\Omega, U, X)$ by $X[i] = X(\Omega[i])$ and $U[i] = U(\Omega[i])$, where $\Omega[i]$ is the $i$th element of $\Omega$ for $i = 1, \dots, |\Omega|$. To convolve two random variables, we simply use this representation for both of them. There is no need to go through the formula $X * Y = (\delta_0 * X) * Y$. Sometimes, however, it may be more efficient to compute $X * P_Y$ by first computing $P_X = \delta_0 * X$ and then $P_X * P_Y$. This is the case, for example, when $|\Omega|$ is much larger than $|A_X|$.

#### 7.2.1.3 Recursive convolution

Let $(\Omega_i, U_i, X_i)$ be independent random variables for $i = 1\!:\!n$ and let $Z_{j:k} = \sum_{i=j}^{k} X_i$. As we have seen, we can compute $P_{Z_{1:n}}$ recursively by

$$P_{Z_{1:k}} = P_{Z_{1:k-1}} * X_k.$$

In order to draw some parallels with later sections, we will establish the validity of this recursion another way.

Let $U_{1:k}$ be the product distribution over the sequence space $\Omega_{1:k}$, that is

$$U_{1:k}(\omega_{1:k}) = \prod_{i=1}^{k} U_i(\omega_i) \quad \text{for } \omega_{1:k} \in \Omega_{1:k}.$$

The distribution of $Z_{1:k}$ is

$$\begin{aligned} P_{Z_{1:k}}(z) &= \sum_{\substack{\omega_{1:k} \in \Omega_{1:k}: \\ \sum_{i=1}^{k} X_i(\omega_i) = z}} U_{1:k}(\omega_{1:k}) = \sum_{\omega_k \in \Omega_k} \sum_{\substack{\omega_{1:k-1} \in \Omega_{1:k-1}: \\ \sum_{i=1}^{k-1} X_i(\omega_i) = z - X_k(\omega_k)}} U_{1:k-1}(\omega_{1:k-1}) U_k(\omega_k) \\ &= \sum_{\omega_k \in \Omega_k} P_{Z_{1:k-1}}(z - X_k(\omega_k)) U_k(\omega_k) = (P_{Z_{1:k-1}} * X_k)(z) \end{aligned}$$

as claimed. Although the algebra does not extend cleanly, the recursion is also valid for $k = 1$ since we have defined $P_{Z_{1:0}} = \delta_0$.

Suppose we want to compute $P_{Z_{1:n}}$ using recursive convolution. The number of operations is

$$O\big(\textstyle\sum_{k=1}^{n} |A_{Z_{1:k-1}}||\Omega_k| \log |A_{Z_{1:k-1}}||\Omega_k|\big),$$

where $A_{Z_{1:k}}$ is the support of $P_{Z_{1:k}}$. Since $A_{Z_{1:k}} = \oplus_{i=1}^{k} A_{X_i}$, where $A_{X_i}$ is the support of $P_{X_i}$, we always have $|A_{Z_{1:k}}| \leq \prod_{i=1}^{k} |A_{X_i}|$ with equality being achieved when each possible sequence in $(A_{X_i})_{i=1:k}$ has a different sum. This worst case scenario causes the number of computations to grow exponentially fast using recursive convolution.

The best case scenario is when $|A_{Z_{1:k}}| = O(k)$ grows at most linearly with $k$. For example, if each $A_{X_i} \subseteq \{-ad, -(a-1)d, \dots, -d, 0, d, \dots, ad\}$ for some integer $a$ and a constant $d$ (both independent of $i$), then $A_{Z_{1:k}} \subseteq \{-kad, -(ka-1)d, \dots, kad\}$ and $|A_{Z_{1:k}}| \leq 2ak + 1$. The convolution algorithms described in this chapter are usually only practical in this special situation where the alphabet grows linearly with the number of convolutions, in which case the number of operations is

$$O\big(\textstyle\sum_{k=1}^{n} ak|\Omega_k|\log ak|\Omega_k|\big) = O(n^2 aL\log naL)$$

where $L$ bounds the $|\Omega_k|$ and $a$ is related to the size of the $A_{X_i}$'s as above.

#### 7.2.1.4 Example: uniformly jittering spikes

Fix a time window $\Omega$ with $L$ bins. Suppose we observe $n$ spikes in $\Omega$ from spike train 1 and suppose we also observe spikes from spike train 2. For each time bin $\omega \in \Omega$ associate a synchrony statistic $X(\omega) = \mathbb{1}\{\text{spike from train 2 in } \omega\}$, which is 1 if there is a spike in train 2 in time bin $\omega$ and 0 otherwise. We want the distribution of the total (sum) amount of synchrony over all possible ways to have $n$ spikes from train 1 in $\Omega$. We want each of the possibilities to be equally likely and we will allow multiple spikes in the same time bin.

The Monte Carlo approach is to independently and randomly choose one of the $L$ bins for each of the $n$ spikes. For that bin $\omega_k$, we compute the synchrony $X(\omega_k)$. Then we add up the individual synchronies to get the total synchrony $Z = X(\omega_1) + \cdots + X(\omega_n)$. We repeat this over and over, independently, and approximate the distribution $P_Z$ of total synchrony with the empirical distribution of the $Z$'s.

The exact approach is (recursive, independent) convolution. We simply compute

$$P_Z = \underbrace{X * \cdots * X}_{n \text{ times}}$$

for the random variable $(\Omega, U, X)$, where $U$ is the uniform distribution over $\Omega$. Since $X \in \{0,1\}$, the alphabet grows linearly with the number of convolutions and the computational complexity will be $O(n^2 L\log nL)$.

#### 7.2.1.5 Tail probabilities

This section and the next focus on algorithms that are used for variable partition jitter in Section 7.5. Variable partition jitter is concerned with computing maximum tail probabilities over a large class of random variables.

The tail probability function of a random variable $Z$ is the unique function $G_Z$ defined by

$$G_Z(z) = \text{Prob}\{Z \geq z\} = \sum_{\zeta \geq z} P_Z(\zeta).$$

A tail probability function for a random variable $Z$ over a finite alphabet is piecewise constant, nonincreasing, continuous from the left, eventually 1 for $z$ small enough and eventually 0 for $z$ large enough. There are finitely many jump points corresponding exactly with $A_Z$, the support of $P_Z$. Furthermore, for any function $G$ with these properties, there is a unique random variable $Z$ over a finite alphabet with $G_Z = G$.

Since $G_Z$ is piecewise constant, we need only represent its value at the jump points $A_Z$. Assuming that $A_Z$ is represented as a vector of increasing values, then we can represent $G_Z$ and $P_Z$ as vectors of the same size with $G_Z[i] = G_Z(A_Z[i])$ and $P_Z[i] = P_Z(A_Z[i])$ for

$i = 1\colon L_Z = |A_Z|$. The vectors $G_Z$ and $P_Z$ are related by

$$G_Z[i] = \sum_{j=i}^{L_Z} P_Z[j] = G_Z[i+1] + P_Z[i] \quad \text{and} \quad P_Z[i] = G_Z[i] - G_Z[i+1], \tag{1}$$

where we take $G_Z[L_Z+1] = 0$. Note that the recursive formula for $G_Z$ allows us to compute it from $P_Z$ using $O(L_Z)$ operations and vice-versa.

If $Z$ and $Z'$ are two random variables such that $G_Z(z) \geq G_{Z'}(z)$ for all $z$, then we say that $G_Z$ dominates $G_{Z'}$ and we write $G_Z \succeq G_{Z'}$. Another characterization is given in the next lemma. A proof can be found at the end of this section.

**Lemma 7.2.1.** *$G_Z \succeq G_{Z'}$ if and only if $G_{Z+X} \succeq G_{Z'+X}$ for all (independent) random variables $X$.*

Or equivalently, $G_Z \succeq G_{Z'}$ if and only if $P_Z * P$ has a tail probability function that dominates that of $P_{Z'} * P$ for all probability distributions $P$. It is easy to see that $\succeq$ is transitive.

Suppose we are given an indexed set of tail probability functions $G^\varphi$ for $\varphi \in \mathcal{L}$ and we want to choose a new collection of tail probability functions $\mathcal{M}$ with the following properties:

$$\text{for each } \varphi \in \mathcal{L} \text{ there exists a } G \in \mathcal{M} \text{ with } G \succeq G^\varphi; \tag{2a}$$
$$\text{for each } G \in \mathcal{M} \text{ there exists a } \varphi \in \mathcal{L} \text{ with } G = G^\varphi; \tag{2b}$$
$$\text{if } G_1, G_2 \in \mathcal{M} \text{ and } G_1 \neq G_2, \text{ then } G_1 \not\succeq G_2. \tag{2c}$$

Because of (2b), we can think about $\mathcal{M}$ as a subset of $\mathcal{L}$. The reason we did not frame the problem this way is because $\mathcal{M}$ is unique when viewed as a collection of functions, whereas it may not be unique when viewed as a subset of $\mathcal{L}$ because of multiplicities within $\mathcal{L}$. In fact, $\mathcal{M}$ is the smallest collection of tail probability functions satisfying both (2a) and (2b). Proofs for these claims can be found below.

If $\mathcal{L}$ is finite, then $\mathcal{M}$ can be generated using a simple selection / pruning strategy: Loop through $\varphi \in \mathcal{L}$. Add $G^\varphi$ to $\mathcal{M}$ if $G^\varphi$ it is not dominated by any $G$ already in $\mathcal{M}$, otherwise skip this $\varphi$. Whenever an element $G^\varphi$ is added to $\mathcal{M}$, remove any elements $G$ already in $\mathcal{M}$ with $G^\varphi \succeq G$. At the end of this process, the resulting set $\mathcal{M}$ evidently satisfies (2).

A related problem is to choose a new tail probability function $\tilde{G}$ such that $\tilde{G} \succeq G^\varphi$ for all $\varphi \in \mathcal{L}$ and such that any $\tilde{G}'$ with this same property has $\tilde{G}' \succeq \tilde{G}$. When $\mathcal{L}$ is finite, the unique solution is the (finite alphabet) tail probability function $\tilde{G}$ defined by

$$\tilde{G}(z) = \max_{\varphi \in \mathcal{L}} G^\varphi(z). \tag{3}$$

$\tilde{G}$ is the "smallest" tail probability function that still dominates each $G^\varphi$. A proof is given at the end of this section.

We will now describe more detailed algorithms for generating $\mathcal{M}$ and $\tilde{G}$. They both begin by representing the tail probability functions over a common alphabet. Order the elements of $\mathcal{L}$ as $\varphi_1, \ldots, \varphi_M$, where $M = |\mathcal{L}|$. To simplify the notation a little, we will use $G_m = G^{\varphi_m}$. Similarly, we will use $A_m$ and $P_m$ to denote the support and distribution of the random variable corresponding to $G_m$. Let $A = \bigcup_{m=1}^{M} A_m$. If $A$ is not known, then it can be computed by concatenating the $A_m$ and finding the unique elements, which takes $O(Mb \log Mb)$ operations and $O(Mb)$ units of memory, where $b$ bounds the $|A_m|$.

For each $m$ and each $z \in A$, compute

$$G_m(z) = \sum_{\zeta \in A_m : \zeta \geq z} P_m(\zeta) = \max_{\zeta \in A_m : \zeta \geq z} G_m(\zeta) = G_m\bigl(\min\{\zeta \in A_m : \zeta \geq z\}\bigr).$$

Assuming that each $P_m$ is represented by a pair of vectors $(A_m, P_m)$ and that $A$ is also represented by a vector, then we can represent each $G_m$ as a vector with the same size as $A$, where $G_m[i] = G_m(A[i])$. We will assume that the elements of each $A_m$ and $A$ are increasing. In this case, computing the collection of vectors $G_m[1 : |A|]$ $(m = 1 : M)$ takes $O(M|A|)$ operations and the same amount of storage.

Computing $\tilde{G}[i] = \max_m G_m[i]$ for each $i$ also takes $O(M|A|)$ operations. Once we know $\tilde{G}$, we can easily get the distribution $\tilde{P}$ that it corresponds to by using (1). If necessary, any $i$ with $\tilde{P}[i] = 0$ can be pruned from $A$ to get $\tilde{A}$, the support of $\tilde{P}$. These final computations are negligible. The total number of operations for computing $\tilde{G}$ is $O(M|A| + Mb \log Mb)$.

For computing $\mathcal{M}$ we begin by defining $\mathcal{M}^0 = \emptyset$. For $m = 1 : M$ we will recursively compute $\mathcal{M}^m$ using $\mathcal{M}^{m-1}$ and $G_m$. If there exists a $G \in \mathcal{M}^{m-1}$ with $G \succeq G_m$, set $\mathcal{M}^m = \mathcal{M}^{m-1}$. Otherwise, remove each $G \in \mathcal{M}^{m-1}$ with $G_m \succeq G$ (if any) to get $\tilde{\mathcal{M}}^m$ and set $\mathcal{M}^m = \tilde{\mathcal{M}}^m \cup G_m$. Once we finish with $m = M$, we set $\mathcal{M} = \mathcal{M}^M$.

The number of operations required for this selection / pruning algorithm depends heavily on the specifics of the problem. Comparing two tail probability functions takes at most $|A|$ operations depending on how they differ. The worse case scenario is when $\mathcal{M} = \mathcal{M}'$, in which case we do $m - 1$ comparisons for each $G_m$ giving a total of $O(M^2|A|)$ operations. The best case scenario is when $|\mathcal{M}^m| = 1$, for each $m$, in which case we do 1 comparison for each $G_m$ giving a total of $O(M|A|)$ operations. When we use this algorithm for variable partition jitter, we need to be near this optimal performance for the jitter algorithm to be practical. The total number of operations for computing $\mathcal{M}$ is thus $O(MM|A| + Mb \log Mb)$, where the extra $M$ in the first term is a worst case scenario.

**Proof of Lemma 7.2.1.** Suppose $G_Z \succeq G_{Z'}$. Then for any random variable $X$

$$\begin{aligned} G_{Z+X}(z) &= \sum_{\zeta \geq z} (P_Z * P_X)(\zeta) = \sum_{\zeta \geq z} \sum_x P_Z(\zeta - x) P_X(x) = \sum_x P_X(x) G_Z(z - x) \\ &\geq \sum_x P_X(x) G_{Z'}(z - x) = G_{Z'+X}(z). \end{aligned}$$

Now suppose $G_Z \not\succeq G_{Z'}$ and let $X = 0$. Then $G_{Z+X} = G_Z \not\succeq G_{Z'} = G_{Z'+X}$. □

**Proofs for (2).** We want to show that $\mathcal{M}$ in (2) is unique. Suppose $\mathcal{M}$ and $\mathcal{M}'$ both satisfy (2) and suppose that there exists a $G' \in \mathcal{M}'$ that is not an element of $\mathcal{M}$. Choose $\varphi' \in \mathcal{L}$ with $G^{\varphi'} = G'$, choose $G \in \mathcal{M}$ with $G \succeq G^{\varphi'}$, choose $\varphi \in \mathcal{L}$ with $G^{\varphi} = G$, and finally choose $G'' \in \mathcal{M}'$ with $G'' \succeq G^{\varphi}$.

We have

$$G'' \succeq G^{\varphi} = G \succeq G^{\varphi'} = G'.$$

Since $G \neq G'$, we must have $G'' \neq G'$ and $G'' \succeq G'$, which contradicts (2c) for $\mathcal{M}'$ and shows that $\mathcal{M}$ is unique. A similar line of reasoning can be used to show that $\mathcal{M}$ is the smallest collection of random variables satisfying both (2a) and (2b). □

**Proofs for (3).** We want to show that $\tilde{G}$ in (3) is the "smallest" tail probability function that bounds each of the $G^\varphi$. Clearly $\tilde{G} \succeq G^\varphi$ for all $\varphi \in \mathcal{L}$. Also, since $\mathcal{L}$ is finite, the maximum is achieved and $\tilde{G}$ has the properties of a tail probability function for a random variable over a finite alphabet.

Suppose that $\tilde{G}' \neq \tilde{G}$ is another tail probability function with $\tilde{G}' \succeq G^\varphi$ for each $\varphi \in \mathcal{L}$. Then $\tilde{G}'(z) \geq G^\varphi(z)$ for all $\varphi \in \mathcal{L}$ and

$$\tilde{G}'(z) \geq \max_{\varphi \in \mathcal{L}} G^\varphi(z) = \tilde{G}(z).$$

So $\tilde{G}' \succeq \tilde{G}$. This also shows that $\tilde{G}$ is the unique solution. □

#### 7.2.1.6 Example: optimal partitions

This example describes the main algorithms that are used for variable partition jitter. We will state the problem here, but it is not motivated until Section 7.5. Suppose that we are given a finite sequence of integers $\omega^*_{1:n}$ with $\omega^*_i \leq \omega^*_{i+1}$ for each $i = 1:n-1$. Suppose also that for each pair of integers $(k, \ell)$ with $k < \ell$ and each $i = 1:n$, we are given a random variable $(\Omega_i^{k,\ell}, U_i^{k,\ell}, X_i^{k,\ell})$. All of the random variables are independent (on the appropriate product space).

Let $\pi = \cdots < \pi_{m-1} < \pi_m < \cdots$ be a partition of the integers, that is, a doubly infinite sequence of increasing integers. For any partition $\pi$ define the random variable

$$Z^\pi = \sum_{i=1}^n \sum_m \mathbb{1}\{\pi_{m-1} < \omega^*_i \leq \pi_m\} X_i^{\pi_{m-1},\pi_m}.$$

The indicator function is only 1 for a single $m$, so $Z^\pi$ is a sum of exactly $n$ (independent) random variables. Another way to express $Z^\pi$ is to define the integer valued functions

$$k(\omega, \pi) = \max_m \{\pi_m : \pi_m < \omega\} \quad \text{and} \quad \ell(\omega, \pi) = \min_m \{\pi_m : \pi_m \geq \omega\}$$

so that

$$Z^\pi = \sum_{i=1}^n X_i^{k(\omega^*_i,\pi),\ell(\omega^*_i,\pi)}.$$

The idea is that each partition creates random variables based on the segments in the partition. We do not add up all of these random variables, but only the ones corresponding to segments that contain an element of $\omega^*_{1:n}$. If a segment contains multiple elements, then it contributes multiple random variables to the sum.

Let $G^\pi = G_{Z^\pi}$ be the tail probability function for the random variable $Z^\pi$. We are interested in the following function:

$$G^*(z) = \max_{\pi \in \mathcal{L}(r,R)} G^\pi(z).$$

The maximum is over all partitions in the set

$$\mathcal{L}(r,R) = \{\pi : \min_m \pi_m - \pi_{m-1} \geq r \text{ and } \max_m \pi_m - \pi_{m-1} \leq R\}$$

for some $1 \leq r \leq R < \infty$. This restricts the minimum and maximum segment lengths of the partitions that we need to consider and constrains the maximum to be taken over an effectively finite set as shown in the next lemma.

**Lemma 7.2.2.** The number of distinct random variables in the set $\{Z^\pi\}_{\pi \in \mathcal{L}(r,R)}$ is bounded above by

$$\big((R+r)(R-r+1)/2\big)^n$$

and if $R \neq r$, then the bound is tight.

A proof is given at the end of this section. The idea is that each $Z^\pi$ does not depend on all of $\pi$, but only on those segments that contain an element of $\omega^*_{1:n}$, that is, only on the sequences $k(\omega^*_i, \pi)$ and $\ell(\omega^*_i, \pi)$ for $i = 1\!:\!n$. The maximum number of possible outcomes for these sequences is easy to enumerate.

For a given partition $\pi$ the distribution $P^\pi = P_{Z^\pi}$ of $Z^\pi$ is

$$P^\pi = X_1^{k(\omega^*_1,\pi),\ell(\omega^*_1,\pi)} * \cdots * X_n^{k(\omega^*_n,\pi),\ell(\omega^*_n,\pi)},$$

which we can compute recursively as described in Sections 7.2.1.1–7.2.1.3. Then we can compute $G^\pi$ using (1). Repeating this for each $\pi \in \mathcal{L}(r,R)$ and taking the maximum gives $G^*$. Lemma 7.2.2 shows that $\mathcal{L}(r,R)$ is effectively finite, so this exhaustive approach will work in principle. Unfortunately, Lemma 7.2.2 also shows that the number of partitions we need to consider can grow exponentially fast, so this exhaustive approach will not work in practice.

Here we will describe two recursive algorithms for computing $G^*$. The first computes $G^*$ exactly. In the worst case it also takes exponentially many convolutions, however, in many cases it will scale polynomially. The second algorithm bounds $G^*$. It always takes polynomially many convolutions. The algorithms are nearly identical and can be mixed together in order to get a bound as close as possible to the exact answer for a given amount of computation. We will begin with the exact algorithm.

**Exact algorithm.** The exact algorithm is based on two key observations. The first is that many partitions agree on some segments that contain elements of $\omega^*_{1:n}$. The convolutions do not need to be carried out separately for these partitions because many of the computations are identical. The second observation is that many partitions can be discarded without completing all of the convolutions for that partition. Define

$$Z^\pi_{i:j} = \sum_{m=i}^{j} X_m^{k(\omega^*_m,\pi),\ell(\omega^*_m,\pi)}.$$

If $\pi$ and $\pi'$ have identical segments containing $\omega^*_{j+1:n}$, then $Z^\pi_{j+1:n} = Z^{\pi'}_{j+1:n}$. Suppose we have computed the distributions of $Z^\pi_{1:j}$ and $Z^{\pi'}_{1:j}$ and we find that $G_{Z^\pi_{1:j}} \succeq G_{Z^{\pi'}_{1:j}}$. Lemma 7.2.1

gives

$$G^{\pi} = G_{Z^{\pi}_{1:n}} = G_{Z^{\pi}_{1:j}+Z^{\pi}_{j+1:n}} \succeq G_{Z^{\pi'}_{1:j}+Z^{\pi}_{j+1:n}} = G_{Z^{\pi'}_{1:j}+Z^{\pi'}_{j+1:n}} = G^{\pi'}.$$

So we know immediately that $\pi'$ does not contribute to the maximum in the definition of $G$ and we can discard it from consideration.

For each integer $\ell$ define

$$\mathcal{L}_{\ell}(r,R) = \{\pi \in \mathcal{L}(r,R) : \pi_m = \ell \text{ for some } m\}$$

to be those allowable partitions that have a segment ending at $\ell$ and define

$$K_{\ell} = K_{\ell}(\omega^*_{1:n}) = \{\# \text{ of } \omega_i \leq \ell\} = \sum_{i=1}^{n} \mathbb{1}\{\omega^*_i \leq \ell\}$$

to be the number of elements in $\omega^*_{1:n}$ less than or equal to $\ell$. To simplify notation, let $G^{\pi}_{i:j} = G_{Z^{\pi}_{i:j}}$ and $G^{\pi}_{i} = G^{\pi}_{1:i}$ and similarly for $P^{\pi}_{i:j}$ and $P^{\pi}_{i}$. Let $\mathcal{M}_{\ell}$ be the unique collection of tail probability functions with the following properties:

$$\text{for each } \pi \in \mathcal{L}_{\ell}(r,R) \text{ there exists a } G \in \mathcal{M}_{\ell} \text{ with } G \succeq G^{\pi}_{K_{\ell}}; \tag{4a}$$

$$\text{for each } G \in \mathcal{M}_{\ell} \text{ there exists a } \pi \in \mathcal{L}_{\ell}(r,R) \text{ with } G = G^{\pi}_{K_{\ell}}; \tag{4b}$$

$$\text{if } G_1, G_2 \in \mathcal{M} \text{ and } G_1 \neq G_2, \text{ then } G_1 \not\succeq G_2. \tag{4c}$$

The uniqueness of $\mathcal{M}_{\ell}$ is discussed following (2).

Following the proof of Lemma 7.2.2, the number of distinct random variables in $\{Z^{\pi}_{1:K_{\ell}}\}_{\pi \in \mathcal{L}_{\ell}(r,R)}$ is bounded above by

$$\big((R+r)(R-r+1)/2\big)^{K_{\ell}}$$

and the bound is tight (except when $r = R$). So $|\mathcal{M}_{\ell}|$ is bounded by the same quantity and is finite. Although the bound is still tight, for variable partition jitter we will often be in the situation where $\mathcal{M}_{\ell}$ only has a few elements or even just a single element.

We can recursively generate $\mathcal{M}_{\ell}$. The recursion is easy to initialize because $\mathcal{M}_k = \{g_0\}$ for all $k < \omega^*_1$, where $g_0(z) = \mathbb{1}\{z \leq 0\}$ is the tail probability function for the zero random variable. The next result gives a stopping criterion for the recursion and motivates why we care about $\mathcal{M}_{\ell}$ in the first place.

$$G^*(z) = \max_{\omega^*_n \leq \ell \leq \omega^*_n + R - 1} \; \max_{\pi \in \mathcal{L}_{\ell}(r,R)} G^{\pi}(z) = \max_{\omega^*_n \leq \ell \leq \omega^*_n + R - 1} \; \max_{G \in \mathcal{M}_{\ell}} G(z). \tag{5}$$

The first equality is trivial because each $\pi \in \mathcal{L}(r,R)$ has a segment ending somewhere in $[\omega^*_n : \omega^*_n + R - 1]$. The second equality comes almost directly from (4a) and (4b) for the cases $\leq$ and $\geq$, respectively.

Suppose we have already computed $\mathcal{M}_k$ for all $k < \ell$. The next lemma takes care of those $\ell$ in long gaps between the elements of $\omega^*_{1:n}$.

**Lemma 7.2.3.** Define
$$\gamma = \left\lceil \frac{r-1}{R-r} \right\rceil r + R - 1,$$
where we take $0/0 = 0$. If $\omega_i^* + \gamma \le k < \ell < \omega_{i+1}^*$. Then $\mathcal{M}_\ell = \mathcal{M}_k$.

A proof can be found at the end of this section. If Lemma 7.2.3 does not apply, then we compute $\mathcal{M}_\ell$ using the following procedure.

For each $k = \ell - R{:}\ell - r$ and each $G \in \mathcal{M}_k$, compute the distribution
$$P^{G,k,\ell} = P^G * \underbrace{X_{K_k+1}^{k,\ell} * \cdots * X_{K_\ell}^{k,\ell}}_{P^{k,\ell}},$$
where $P^G$ is the distribution corresponding to $G$. If $K_k + 1 > K_\ell$, that is, there are no elements of $\omega_{1:n}^*$ in $(k, \ell]$, then we simply take $P^{G,k,\ell} = P^G$. Note that if $\mathcal{M}_k$ has multiple elements and $K_\ell - K_k > 2$, then it is more efficient to first compute $P^{k,\ell}$ and then compute $P^{G,k,\ell} = P^G * P^{k,\ell}$.

Let $G^{G,k,\ell}$ be the tail probability function corresponding to $P^{G,k,\ell}$ and let $\mathcal{M}'_\ell = \{(k, G) : \ell - R \le k \le \ell - r, G \in \mathcal{M}_k\}$. For each $\varphi = (k, G) \in \mathcal{M}'_\ell$, let $G^\varphi = G^{G,k,\ell}$. Note that $|\mathcal{M}'_\ell|$ is finite. We can use the selection / pruning algorithm described in the previous section to choose a unique set of tail probability functions $\mathcal{M}_\ell$ from $\{G^\varphi\}_{\varphi \in \mathcal{M}'_\ell}$ that satisfy (2) with $\mathcal{L}$ replaced by $\mathcal{M}'_\ell$ and $\mathcal{M}$ replaced by $\mathcal{M}_\ell$. This $\mathcal{M}_\ell$ is the same $\mathcal{M}_\ell$ implicitly defined by (4).

**Proof that this $\mathcal{M}_\ell$ satisfies (4).** Choose $\pi \in \mathcal{L}_\ell(r, R)$ and let $(k, \ell]$ be a segment in $\pi$ for some $\ell - R \le k \le \ell - r$. So $\pi \in \mathcal{L}_k(r, R)$ and there is a $G' \in \mathcal{M}_k$ with $G' \succeq G_{K_k}^\pi$. We have
$$P^{G',k,\ell} = P^{G'} * X_{K_k+1}^{k,\ell} * \cdots * X_{K_\ell}^{k,\ell} \quad \text{and } P_{K_\ell}^\pi = P_{K_k}^\pi * X_{K_k+1}^{k,\ell} * \cdots * X_{K_\ell}^{k,\ell},$$
so Lemma 7.2.1 shows that $G^{G',k,\ell} \succeq G_{K_\ell}^\pi$. Finally, because of the selection / pruning algorithm, there is a $G$ in the set $\mathcal{M}_\ell$ that we just created with $G \succeq G^{G',k,\ell} \succeq G_{K_\ell}^\pi$ and we see that this $\mathcal{M}_\ell$ does indeed satisfy (4a).

Now choose $G \in \mathcal{M}_\ell$. The selection / pruning algorithm guarantees that there is a $\varphi = (k, G') \in \mathcal{M}'_\ell$ with $G = G^\varphi = G^{G',k,\ell}$. Since $G' \in \mathcal{M}_k$, there is a $\pi' \in \mathcal{L}_k(r, R)$ with $G' = G_{K_k}^{\pi'}$ and also a $\pi \in \mathcal{M}_k$ that is identical to $\pi'$ up to $k$ and with the segment $(k, \ell]$. So $G' = G_{K_k}^\pi$ as well. For this $\pi$, we have
$$P_{K_\ell}^\pi = P_{K_k}^\pi * X_{K_k+1}^{k,\ell} * \cdots * X_{K_\ell}^{k,\ell} = P_{K_k}^{\pi'} * P^{k,\ell} = P^{G',k,\ell},$$
so $G = G^{G',k,\ell} = G_{K_\ell}^\pi$ and $\mathcal{M}_\ell$ satisfies (4b). The selection / pruning algorithm also gives (4c) and the proof is complete. □

In summary, we have described a recursive procedure for generating the $\mathcal{M}_\ell$. We can begin with $\ell = \omega_1^* - R$. Once we get to $\ell = \omega_n^* + R - 1$, we can use (5) to compute the function $G^*$. The number of operations needed for this computation depends heavily on the specifics of the problem. Of particular importance are the size of $\mathcal{M}_\ell$ and the size of the combined supports $A_{1:i}^\pi$ of all the $Z_{1:i}^\pi$. We will analyze the situation in terms of the bounds $|\mathcal{M}_\ell| \le M$ for all $\ell$ and $|\cup_\pi A_{1:i}^\pi| \le b_i$ for all $i$. For the algorithm to be practical we will need to be in the situation where $M$ is small and $b_i = O(ai)$ grows at most linearly with the number of convolutions $i$.

**Computational complexity.** We will first analyze the number of operations needed for the convolution part of the algorithm. Let $C(n)$ be the maximum number of operations needed to compute the distribution of any $Z^\pi$. When $b_i = O(ai)$, we have $C(n) = O(n^2 aL \log naL)$, where $L$ bounds the size of the $|\Omega_i^{k,\ell}|$; see Section 7.2.1.3. Following the proof of Lemma 7.2.2, there are $O(R(R-r+1))$ possible segments that can contain $\omega_i^*$. Let $(k,\ell]$ be one of these segments. The random variable $X_i^{k,\ell}$ will be convolved with $P^G$ for each $G \in \mathcal{M}_k$. This takes at most $M$ convolutions and this is the only time $X_i^{k,\ell}$ is considered by the algorithm. This happens for each $(k,\ell]$, giving $O(MR(R-r+1))$ single convolutions for $\omega_i^*$. This corresponds to the same number of full recursive convolutions over $i = 1{:}n$, resulting in the total

$$\text{\# of operations for convolutions} = C(n)O(MR(R-r+1)).$$

The other part of the procedure is the selection / pruning algorithm. This algorithm is applied to each $\mathcal{M}'_\ell$ to generate $\mathcal{M}_\ell$. If each $\mathcal{M}_k$ for $k < \ell$ has at most $M$ elements, then $\mathcal{M}'_\ell$ has at most $M' = M(R-r+1)$ elements. From the last section, the number of operations used to compute $\mathcal{M}_\ell$ from $\mathcal{M}'_\ell$ is $O(M'M'b_{K_\ell} + M'b_{K_\ell}\log M'b_{K_\ell})$. The extra leading $M'$ term is a worst case situation. Also, the $b_{K_\ell}$'s in the second term are conservative. Let $S(n)$ be the number of times that this algorithm is used. Since the algorithm is used at most once for each step in the recursion, we always have $S(n) \le \omega_n^* - \omega_1^* + 2R$. On the other hand, Lemma 7.2.3 implies that we need only consider $O(\gamma)$ recursions per $\omega_i^*$, so $S(n) = O(n\gamma) = O(nR^2/(R-r))$. This bound is important because it emphasizes that the computation complexity need not grow with the number of recursions. Using the idea behind this bound, we can express the total number of operations for this part of the procedure as

$$\begin{aligned}
\text{\# of operations for selection / pruning} &= \sum_{\text{recursions } \ell} O(\text{operations at } \ell) \\
&= \sum_{i=1}^{n} \sum_{\ell=\omega_i^*}^{\omega_{i+1}^*-1} O(\text{operations at } \ell) = \sum_{i=1}^{n} \sum_{\ell=\omega_i^*}^{\omega_{i+1}^*-1} O(\text{operations using supports } A_{1:i}^\pi) \\
&= \sum_{i=1}^{n} O(\gamma)O(\text{operations using } A_{1:i}^\pi) = \sum_{i=1}^{n} O(\gamma)O(M'M'b_i + M'b_i \log M'b_i) \\
&= O(R^2/(R-r)) \times \\
&\quad \sum_{i=1}^{n} O\big(M(R-r+1)b_i(\log M(R-r+1)b_i + M(R-r+1))\big).
\end{aligned}$$

Note that the final $M(R-r+1)$ term is a worst case scenario. In the best case it disappears completely.

Combining the two totals, assuming $R - r \approx R$ (in particular $R \neq r$) and assuming that the supports grow linearly, that is, $b_i = O(ai)$, gives

$$\begin{aligned}&\#\text{ of operations for computing } G^* \\ &\quad = O(n^2 aL \log naL) O(MR^2) + O(R) O\big(MRn^2 a(\log MRna + MR)\big) \\ &\quad = O\big(MR^2 n^2 a(L \log naL + \log MRna + MR)\big).\end{aligned}$$

The extra $MR$ term is a worst case situation. In the best case it disappears completely.

The variable partition jitter method described in Section 7.5 will be framed in terms of a parameter $\Delta$ with $R-r \approx R \approx L \approx \Delta$. Assuming the best case of $M = 1$ and assuming that the supports grow linearly, the number of operations is $O(n^2 a\Delta^3 \log na\Delta)$, which is often practical.

**Bounding algorithm.** For many examples, $M$ is small, say 1 or 2. In general, however, $|M_\ell|$ can grow exponentially with $K_\ell$ and $M$ is much too large. In these cases, this exact algorithm for computing $G^*$ is impractical. We can nevertheless bound $G^*$. Any $\pi \in \mathcal{L}(r, R)$ gives the trivial lower bound $G^* \succeq G^\pi$, which is easy to compute. For certain situations, there may be heuristics that give a good choice of $\pi$ for this type of bound. The more complicated case is the upper bound, however, it is easy to modify the exact procedure for computing $G^*$ and compute an upper bound instead.

The idea is to replace each $\mathcal{M}_\ell$ with a new collection of tail probability functions $\tilde{\mathcal{M}}_\ell$, with the property that for each $G \in \mathcal{M}_\ell$ there is a $\tilde{G} \in \tilde{\mathcal{M}}_\ell$ with $\tilde{G} \succeq G$. The computational demands of this algorithm are easier to control because for any $\tilde{M} \geq 1$, we can enforce the constraint $|\tilde{\mathcal{M}}_\ell| \leq \tilde{M}$.

We begin using the exact algorithm. If for some $\ell$, $|\mathcal{M}_\ell| > \tilde{M}$, then we replace two $G_1, G_2 \in \mathcal{M}_\ell$ with a single $\tilde{G}(z) = \max_{i=1:2} G_i(z)$ as in (3), perhaps repeating this more than once until the new set $\tilde{\mathcal{M}}_\ell$ has no more than $\tilde{M}$ elements. Now we continue the recursive algorithm with $\tilde{\mathcal{M}}_\ell$ instead of $\mathcal{M}_\ell$. Every time the collection of tail probability functions gets too large, we make it smaller with this maximum operation. It is easy to see that the resulting collections $\tilde{\mathcal{M}}_\ell$ have the desired properties and that the final maximum in (5) gives a tail probability function $\tilde{G}^* \succeq G^*$.

The number of operations for this recursive bounding algorithm is the same as the exact algorithm with $M$ replaced by $\tilde{M}$, which we can choose. There may be interesting heuristics for choosing which $G_i$'s to combine with the maximum operations, but we do not explore that here. In the special case $\tilde{M} = 1$, the entire collection of $G's$ is replaced by their pointwise maximum $\tilde{G}$ at each step in the recursion. For this special case, the extra $MR$ term in the number of operations disappears. This special case also gives $O(n^2 a\Delta^3 \log na\Delta)$ operations for variable partition jitter.

**Proof of Lemma 7.2.2.** The random variable $Z^\pi$ only depends on the sequences $k(\omega_i^*, \pi)$ and $\ell(\omega_i^*, \pi)$ for $i = 1\!:\!n$. Since the integers $[k(\omega_i^*, \pi) + 1 : \ell(\omega_i^*, \pi)]$ are one segment of the partition $\pi \in \mathcal{L}(r, R)$, the segment length $L = \ell(\omega_i^*, \pi) - k(\omega_i^*, \pi) + 1 \in [r : R]$. For each of these possible lengths $L$, $\omega_i^*$ can be at any of the $L$ different positions in the segment. This gives a total of

$$\sum_{L=r}^{R} L = \frac{(R + r)(R - r + 1)}{2}$$

different possible $k(\omega_i^*, \pi), \ell(\omega_i^*, \pi)$ combinations.

Repeating this for each $i$ gives $\big((R+r)(R-r+1)/2\big)^n$ different sequences $k(\omega_i^*, \pi)$ and $\ell(\omega_i^*, \pi)$ for $i = 1:n$. For certain $\omega_{1:n}^*$, some of these sequences may not correspond to a partition (for example, if $k(\omega_i^*, \pi) < k(\omega_{i+1}^*, \pi) < \ell(\omega_i^*, \pi)$), however, if the $\omega_{1:n}^*$ are sufficiently separated (see the proof of Lemma 7.2.3; note the exception for $R = r$), then each of these sequences will correspond to some partition $\pi \in \mathcal{L}(r, R)$. It is possible to choose the $X_i^{k,\ell}$'s so that different sequences give different sums. This shows that the bound is tight. □

**Proof of Lemma 7.2.3.** Note that we can assume $r < R$. We will first show that

$$\text{if } t - s \geq \gamma - R + 1 \text{ then } \mathcal{L}_s(r, R) \cap \mathcal{L}_t(r, R) \neq \emptyset. \tag{6}$$

In other words, it is possible to partition the integers $[s+1:t]$ into segments with lengths between $r$ and $R$, inclusive. The first segment is $[s+1:s+L_1]$ for $L_1 \in [r:R]$. The $j$th segment is $[s+L_1+\cdots+L_{j-1}+1:s+L_1+\cdots+L_j]$, where each $L_i \in [r:R]$. Since this is the only constraint, each of the integers $[s+jr:s+jR]$ is an allowable end point for the $j$th segment. Notice that if $s+(j+1)r \leq s+jR+1$, then the possible end points for the $(j+1)$th segment form a contiguous sequence with the possible endpoints for the $j$th segment and all integers $t \geq s+jr$ are possible endpoints for some segment. The constraint on $j$ is

$$s+(j+1)r \leq s+jR+1 \implies j(R-r) \geq r-1 \implies j \geq \frac{r-1}{R-r}.$$

Since $j$ must be an integer, the constraint on $t$ becomes

$$t \geq s + jr \geq s + \left\lceil \frac{r-1}{R-r} \right\rceil r = s + \gamma - R + 1.$$

This proves (6). Note that (6) remains valid for $R = r = 1$ if we take $0/0 = 0$.

To prove the lemma, it is sufficient to show that

$$\{G^\pi_{K_\ell}\}_{\pi \in \mathcal{L}_\ell(r,R)} = \{G^\pi_{K_k}\}_{\pi \in \mathcal{L}_k(r,R)}.$$

Since $K_\ell = K_k = i$, we need only show that

$$\{G^\pi_i\}_{\pi \in \mathcal{L}_\ell(r,R)} = \{G^\pi_i\}_{\pi \in \mathcal{L}_k(r,R)}.$$

Fix $\pi \in \mathcal{L}_\ell(r, R)$. Let $\pi_m$ be the endpoint of the segment containing $\omega_i^*$. We know that $k - \pi_m \geq \omega_i^* + \gamma - (\omega_i^* + R - 1) = \gamma - R + 1$. (6) implies that we can construct a new partition $\pi' \in \mathcal{L}_k(r, R)$ that is identical to $\pi$ up to and including the segment that ends at $\pi_m$. So $G_i^\pi = G_i^{\pi'}$ and half of the desired equality is established. Since $\ell > k$, an identical argument beginning with $\pi \in \mathcal{L}_k(r, R)$ completes the proof. □

### 7.2.2 Markov dependent convolution

Consider the situation in Section 7.2.1.3. We have a product distribution $U = U_{1:n}$ over the sequence space $\Omega_{1:n}$ and we are interested in the distribution of the random variable $Z_{1:n}(\omega_{1:n}) = \sum_{i=1}^n X_i(\omega_i)$ for some functions $X_i$ on $\Omega_i$ $(i = 1:n)$. We want to relax the requirement that $U$ is a product distribution. For the jitter methods, it is convenient to represent the dependencies in $U$ as a departure from a product distribution $U_{1:n}$. In particular, we will use the joint distribution $U = U^W_{1:n}$ defined by

$$U^W_{1:n}(\omega_{1:n}) = \kappa^{-1} W(\omega_{1:n}) U_{1:n}(\omega_{1:n})$$

where $W$ is a nonnegative function on $\Omega_{1:n}$ and $\kappa = \sum_{\omega_{1:n}\in\Omega_{1:n}} W(\omega_{1:n})U_{1:n}(\omega_{1:n})$ is a normalizing constant. The distribution of $Z_{1:n}$ is

$$P_{Z_{1:n}}(z) = \sum_{\substack{\omega_{1:n}\in\Omega_{1:n}: \\ \sum_{i=1}^n X_i(\omega_i)=z}} U^W_{1:n}(\omega_{1:n}).$$

For computational reasons, we will mostly focus on the Markov situation where $W$ factors into $W(\omega_{1:n}) = W_2(\omega_1,\omega_2)W_3(\omega_2,\omega_3)\cdots W_n(\omega_{n-1},\omega_n)$ for nonnegative functions $W_i$ on $\Omega_{i-1:i} = \Omega_{i-1}\times\Omega_i$. In this case, we can compute the distribution of $Z_{1:n}$ using a recursive convolution-like procedure (or dynamic programming) that we call *Markov dependent convolution*. This is somewhat of a misnomer. $Z_{1:n}$ is a sum of the $X_i$'s. For independent convolution, the $X_i$'s are independent (by virtue of the product distribution on the sample space $\Omega_{1:n}$). For Markov dependent convolution, the $X_i$'s are not necessarily a Markov chain, rather, the joint distribution on the sample space $\Omega_{1:n}$ is Markov. It would be more precise to say something like *convolution of deterministic functions of a Markov chain*.

We begin by describing some new notation and the recursive formula at the heart of Markov dependent convolution. The notation and terminology is meant to draw parallels between Markov dependent convolution and (independent) convolution of random variables described in Sections 7.2.1.2 and 7.2.1.3.

Let $\Omega$ be a finite sample space. A *Markov convolution function* on $\Omega$ is a function $H : \mathbb{R}\times\Omega \to [0,1]$ such that $\sum_{z,\omega} H(z,\omega) = 1$. The support of $H$ is $B = \{z : H(z,\omega) > 0 \text{ for some } \omega\in\Omega\}$, which we assume is finite.

Given a Markov convolution function $H_Y$ on the sample space $\Lambda$ with support $B_Y$, a function $X$ on a finite sample space $\Omega$, a reference probability $U$ on $\Omega$ and a weighting function $W : \Lambda\times\Omega \to [0,\infty)$, define the new Markov convolution function $H_Y *^W X$ on $\Omega$ by

$$\big(H_Y *^W X\big)(z,\omega) = \kappa^{-1}\sum_{\lambda\in\Lambda} H_Y(z - X(\omega),\lambda)W(\lambda,\omega)U(\omega),$$

where $\kappa$ is a normalizing constant, that is

$$\kappa = \sum_{z\in\mathbb{R},\omega\in\Omega,\lambda\in\Lambda} H_Y(z - X(\omega),\lambda)W(\lambda,\omega)U(\omega).$$

We need only consider those $z \in B_Y \otimes B_X$, where $B_X$ is the range of the function $X$. In the pathological case $\kappa = 0$, we say that $H_Y *^W X$ is undefined. We use the term *Markov dependent convolution* for the operation $*^W$.

Suppose we are given a collection $(\Omega_i, U_i, X_i)$, for $i = 1 : n$, of (independent) random variables. We can compute the distribution of $Z_{1:n} = \sum_{i=1}^n X_i$ by recursively convolving

$$P_{Z_{1:n}} = X_1 * \cdots * X_n = (\cdots(((\delta_0 * X_1) * X_2) * X_3) \cdots) * X_n.$$

Now suppose that we want to introduce some dependencies using a Markov weighting function $W = W_2 \cdots W_n$. The distribution over the sequence space $\Omega_{1:n}$ is now $U_{1:n}^W$ and we are interested in the new distribution of the function $Z_{1:n} = \sum_{i=1}^n X_i$. We can use recursive Markov dependent convolution:

$$\begin{aligned} P_{Z_{1:n}}(z) &= \sum_{\omega_n \in \Omega_n} H_{Z_{1:n}}(z, \omega_n) \quad \text{where} \\ H_{Z_{1:n}} &= \Big(\cdots\Big(\Big(\Big(h_0 *^1 X_1\Big) *^{W_2} X_2\Big) *^{W_3} X_3\Big)\cdots\Big) *^{W_n} X_n. \end{aligned} \tag{7}$$

This formula is established below. The reference probabilities are the $U_i$'s and the $X_i$'s are the functions on the $\Omega_i$'s. While we still think about the triple $(\Omega_i, U_i, X_i)$ as a random variable, this not technically correct. $U_i$ is the reference probability and is *not* necessarily the marginal of $U_{1:n}^W$ on $\Omega_i$. In particular $P_{X_i}$ cannot be inferred from $U_i$ but depends on all of $U_{1:n}^W$. The Markov convolution function $h_0$ on $\Omega_0 = \{\omega_0\}$ corresponds to the point mass at zero and is defined by $h_0(z, \omega_0) = \mathbb{1}\{z = 0, \omega = \omega_0\}$ and the weighting function on $\Omega_0 \times \Omega_1$ used for $*^1$ is identically 1. The initialization $h_0 *^1 X_1$ thus converts $X_1$ into the Markov convolution function defined by

$$(h_0 *^1 X_1)(z, \omega_1) = \mathbb{1}\{X_1(\omega_1) = z\} U_1(\omega_1).$$

For the independent situation we used the convention that $X * Y = (\delta_0 * X) * Y$ and we will use the same convention here, namely, that if $X$ and $Y$ are "random variables", then $X *^W Y = (h_0 *^1 X) *^W Y$. Also, for the independent situation we suppressed the grouping for describing the order of the convolution. This was not a problem since convolution can be performed in any order. Here, for convenience, we will also suppress the grouping with the caveat that the convolutions must be performed from left to right. (7) becomes the much cleaner

$$P_{Z_{1:n}}(z) = \sum_{\omega_n \in \Omega_n} H_{Z_{1:n}}(z, \omega_n) \quad \text{where} \quad H_{Z_{1:n}} = X_1 *^{W_2} X_2 *^{W_3} \cdots *^{W_n} X_n. \tag{8}$$

We need to show that (8) actually gives the distribution of $Z_{1:n}$ like we are claiming. It is evidently true for $k = 1$ (and even $k = 0$ if we take $H_{Z_{1:0}} = h_0$). For $k = 2{:}n$ define

$$H_{Z_{1:k}} = X_1 *^{W_2} \cdots *^{W_k} X_k.$$

We will show that

$$H_{Z_{1:k}}(z, \omega_k) = \kappa_k^{-1} \sum_{\substack{\omega_{1:k-1} \in \Omega_{1:k-1}: \\ \sum_{i=1}^k X_i(\omega_i) = z}} U_1(\omega_1) \prod_{i=2}^k W_i(\omega_{i-1}, \omega_i) U_i(\omega_i), \tag{9}$$

where $\kappa_k$ is the normalizing constant

$$\kappa_k = \sum_{\omega_{1:k} \in \Omega_{1:k}} U_1(\omega_1) \prod_{i=2}^{k} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i).$$

Note that $\kappa_k$ is chosen so that $\sum_{z,\omega_k} H_{Z_{1:k}}(z, \omega_k) = 1$.

When $k = 2$,

$$\begin{aligned} H_{Z_{1:2}}(z, \omega_2) &= \tilde{\kappa}_2^{-1} \sum_{\omega_1 \in \Omega_1} (h_0 *^1 X_1)(z - X_2(\omega_2), \omega_1) W_2(\omega_1, \omega_2) U_2(\omega_2) \\ &= \tilde{\kappa}_2^{-1} \sum_{\omega_1 \in \Omega_1} \mathbb{1}\{X_1(\omega_1) = z - X_2(\omega_2)\} U_1(\omega_1) W_2(\omega_1, \omega_2) U_2(\omega_2) \\ &= \tilde{\kappa}_2^{-1} \sum_{\substack{\omega_1 \in \Omega_1: \\ \sum_{i=1}^{2} X_i(\omega_i) = z}} U_1(\omega_1) \prod_{i=2}^{2} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i), \end{aligned}$$

where $\tilde{\kappa}_2$ is chosen so that $\sum_{z,\omega_2} H_{Z_{1:2}}(z, \omega_2) = 1$. So $H_{Z_{1:2}}$ indeed satisfies (9). Now suppose that $H_{Z_{1:k-1}}$ satisfies (9) for some $2 < k \leq n$. Then

$$\begin{aligned} H_{Z_{1:k}}(z, \omega_k) &= \tilde{\kappa}_k^{-1} \sum_{\omega_{k-1} \in \Omega_{k-1}} H_{Z_{1:k-1}}(z - X_k(\omega_k), \omega_{k-1}) W_k(\omega_{k-1}, \omega_k) U_k(\omega_k) \\ &= \tilde{\kappa}_k^{-1} \sum_{\omega_{k-1} \in \Omega_{k-1}} \kappa_{k-1}^{-1} \sum_{\substack{\omega_{1:k-2} \in \Omega_{1:k-2}: \\ \sum_{i=1}^{k-1} X_i(\omega_i) = z - X_k(\omega_k)}} U_1(\omega_1) \prod_{i=2}^{k-1} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i) \;\; W_k(\omega_{k-1}, \omega_k) U_k(\omega_k) \\ &= \tilde{\kappa}_k^{-1} \kappa_{k-1}^{-1} \sum_{\substack{\omega_{1:k-1} \in \Omega_{1:k-1}: \\ \sum_{i=1}^{k} X_i(\omega_i) = z}} U_1(\omega_1) \prod_{i=2}^{k} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i), \end{aligned}$$

where the constant $\tilde{\kappa}_k$ is chosen so that $H_{Z_{1:k}}$ sums to 1. Since the constant $\kappa_{k-1}$ is just absorbed into $\tilde{\kappa}_k$, we see by induction that $H_{Z_{1:k}}$ satisfies (9) for all $k = 2{:}n$.

Finally, for the case $k = n$, we can use the alternative representation in (9) to get

$$\begin{aligned} \sum_{\omega_n \in \Omega_n} H_{Z_{1:n}}(z, \omega_n) &= \kappa_n^{-1} \sum_{\substack{\omega_{1:n} \in \Omega_{1:n}: \\ \sum_{i=1}^{n} X_i(\omega_i) = z}} U_1(\omega_1) \prod_{i=2}^{n} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i) \\ &= \sum_{\substack{\omega_{1:n} \in \Omega_{1:n}: \\ Z_{1:n}(\omega_{1:n}) = z}} U_{1:n}^{W}(\omega_{1:n}) = P_{Z_{1:n}}(z). \end{aligned}$$

Since $\kappa_n$ is chosen so that $H_{Z_{1:n}}$ sums to 1, it is easy to see that $\kappa_n$ is the same normalizing constant in the definition of $U_{1:n}^{W}$. This completes the proof of (8) and (7).

#### 7.2.2.1 A dynamic programming algorithm

Given a Markov convolution function $H_Y$ over $\Lambda$ with support $B_Y$, a "random variable" $(\Omega, U, X)$ and a Markov weighting function $W$ on $\Lambda \times \Omega$, we want to compute $H_Z = H_Y *^W X$. To represent $H_Y$ we need an $|B_Y| \times |\Lambda|$ matrix for the values and a $|B_Y|$ length vector for the support $B_Y$. We will use the notation $H_Y[i,j]$ and $B_Y[i]$ for this matrix and vector, respectively, where $H_Y[i,j] = H_Y(B_Y[i], \Lambda[j])$, $B_Y[i]$ is the $i$th element of $B_Y$ and $\Lambda[j]$ is the $j$th element of $\Lambda$. We can represent $H_Z$ in a similar fashion with a vector $B_Z$ for the support and a $|B_Z| \times |\Omega|$ matrix $H_Z$ for the values. To represent $X$ and $U$ we use two vectors of length $|\Omega|$, namely $X[i] = X(\Omega[i])$ and $U[i] = U(\Omega[i])$. Finally, to represent $W$ we use a $|\Lambda| \times |\Omega|$ matrix $W[i,j] = W(\Lambda[i], \Omega[j])$.

An algorithm that is closely related to the `conv` algorithm in Section 7.2.1.1 and which we denote

$$(B_Z, H_Z) = \texttt{convm}(B_Y, H_Y, W, X, U),$$

(the `m` stands for Markov) is the following: We first compute the $|B_Y| \times |\Omega|$ matrices $B[i,j] = B_Y[i] + X[j]$ and

$$C[i,j] = \sum_{\ell=1}^{|\Lambda|} H_Y[i,\ell] W[\ell,j] U[j].$$

We determine $B_Z$ by all of the unique values in $B$. For each $m$ and each $j$ we add up those $C[i,j]$ for which $B[i,j] = B_Z[m]$ to get $H_Z[m,j]$. We may need to prune those $m$ (from both $B_Z$ and $H_Z$) for which $H_Z[m,j] = 0$ for all $j$. Finally, we need to normalize $H_Z$ so that it has sum 1. (In principle, if we are recursively applying `convm`, then the normalization can be postponed until the final convolution, saving a little computation. In practice, normalization is important for preventing underflow and overflow problems.)

The only significant difference between this algorithm and `conv` is the added computational complexity for computing $C$, which now takes $O(|B_Y||\Omega||\Lambda|)$ operations. The storage requirements are similar in most practical situations, although $W$ takes $O(|\Lambda||\Omega|)$ memory units and $H_Z$ takes $O(|B_Z||\Omega|)$, either of which could be significantly larger in general than the $O(|B_Y||\Omega|)$ storage requirements for independent convolution of random variables (where $B_Y$ is similar to $A_Y$). The sorting demands are the same, so a single iteration takes $O\big(|B_Y||\Omega|(|\Lambda| + \log|B_Y||\Omega|)\big)$ operations.

For recursive Markov dependent convolution, say $H_{Z_{1:n}} = X_1 *^{W_2} \cdots *^{W_n} X_n$, the total number of operations needed to compute $H_{Z_{1:n}}$ is

$$O\Big(\textstyle\sum_{k=1}^n \big|B_{Z_{1:k-1}}\big|\big|\Omega_k\big|\big(\big|\Omega_{k-1}\big| + \log\big|B_{Z_{1:k-1}}\big|\big|\Omega_k\big|\big)\Big).$$

$B_{Z_{1:k}} \subseteq \oplus_{i=1}^k B_{X_i}$. Following the discussion at the end of Section 7.2.1.3, $\big|B_{Z_{1:k}}\big|$ can grow exponentially in the worst case or linearly in the best case. In practice, we will essentially always need to be in the linear situation.

If $\big|B_{Z_{1:k}}\big| = O(ak)$ for a constant $a$ related to size of the range of the $X_i$'s, then the number of operations is

$$O\Big(\textstyle\sum_{k=1}^n ak\big|\Omega_k\big|\big(\big|\Omega_{k-1}\big| + \log ak\big|\Omega_k\big|\big)\Big) = O\big(n^2 aL(L + \log naL)\big),$$

where $L$ bounds the $|\Omega_i|$'s. Note that computing $P_{Z_{1:n}}$ from $H_{Z_{1:n}}$ adds a negligible amount of computation, in this case, $O(naL)$ operations.

#### 7.2.2.2 Hard constraints

Consider the situation where the weighting function $W : \Omega_{1:n} \mapsto \{0, 1\}$ is binary. Let $C = W^{-1}[1]$ so that $W = \mathbb{1}\{\omega_{1:n} \in C\}$. The elements in $C$ are the allowable sequences in $\Omega_{1:n}$. One way to understand $U^W_{1:n}$ is to think about sampling. We independently choose $\omega_{1:n}$ from the product distribution $U_{1:n}$. If $\omega_{1:n}$ is allowable, then we keep it, otherwise we ignore this $\omega_{1:n}$ and try again until we get an allowable sequence. Another way to think about $U^W_{1:n}$ is conditioning, namely, conditioning on the set $C$. It is easy to see that $U^W_{1:n}(\omega_{1:n}) = U_{1:n}(\omega_{1:n}|C)$.

#### 7.2.2.3 Example: uniformly jittering spikes with a refractory period

This example is a modification of the example in Section 7.2.1.4. Everything is the same, except we do not want to allow multiple spikes from train 1 to land in the same bin. A single sample in the Monte Carlo approach is to randomly choose $n$ of the possible $L$ bins *without replacement.* Every bin is equally likely. These $n$ bins represent the $n$ spike times from train 1.

The exact approach is Markov dependent convolution. We compute

$$P_Z = \underbrace{X *^W \cdots *^W X}_{n \text{ times}}$$

for the "random variable" $(\Omega, U, X)$ with the binary constraint function $W(\omega, \omega') = \mathbb{1}\{\omega < \omega'\}$, which is 1 if the $k$th spike bin comes after the $(k-1)$th spike bin and 0 otherwise. The reference probability distributions $U$ are still uniform because we do not weight any of the $L$ positions more than any other one.

We can modify this example slightly by making $W(\omega, \omega') = \mathbb{1}\{\omega + \tau < \omega'\}$, which enforces a refractory period (minimum spacing between spikes) of $\tau$ bins. Now the Monte Carlo approach is to sample spikes without replacement as before, but to throw out any samples that violate the refractory period. While the exact approach does not change, this Monte Carlo method can be very inefficient if many samples have to be discarded. It turns out that the Markov structure of the constraints can also be exploited for efficient sampling procedures, as we will see in Section 7.2.2.6

#### 7.2.2.4 Soft constraints

In the general situation the weighting function $W : \Omega_{1:n} \mapsto [0, \infty)$ need not be binary and the interpretation of $U^W_{1:n}$ is a little more complicated. Consider the case where each $U_i$ is uniform over $\Omega_i$. Then the independent joint distribution is $U_{1:n}(\omega_{1:n}) = c$ for some constant $c$. Each of the possible configurations is initially weighted the same. The weighted joint is $U^W_{1:n}(\omega_{1:n}) = \kappa^{-1}cW(\omega_{1:n})$. Since any scaling of $W$ would just be absorbed into the normalization constant $\kappa$, it is clear that the exact magnitude of $W$ is meaningless. In this case, $W$ gives the *relative* probabilities of different events as evidenced by the formula

$$\frac{U_{1:n}^W(\omega_{1:n})}{U_{1:n}^W(\tilde{\omega}_{1:n})} = \frac{\kappa^{-1}cW(\omega_{1:n})}{\kappa^{-1}cW(\tilde{\omega}_{1:n})} = \frac{W(\omega_{1:n})}{W(\tilde{\omega}_{1:n})}.$$

For the general situation, when the $U_i$'s are not necessarily uniform, we have

$$\frac{U_{1:n}^W(\omega_{1:n})}{U_{1:n}^W(\tilde{\omega}_{1:n})} = \frac{\kappa^{-1}W(\omega_{1:n})U_{1:n}(\omega_{1:n})}{\kappa^{-1}W(\tilde{\omega}_{1:n})U_{1:n}(\tilde{\omega}_{1:n})} = \frac{W(\omega_{1:n})}{W(\tilde{\omega}_{1:n})}\frac{U_{1:n}(\omega_{1:n})}{U_{1:n}(\tilde{\omega}_{1:n})}.$$

In this case $W$ describes the change in the relative probabilities from the independent setting. Again, the exact magnitude of $W(\omega_{1:n})$ is meaningless. Only the relative magnitudes of $W$ at different points in $\Omega_{1:n}$ are important.

For computing the distribution of $Z_{1:n}$, the Monte Carlo sampling intuition is the following: we sample $\omega_{1:n}$ from the independent joint $U_{1:n}$ and compute $Z_{1:n} = \sum_{i=1}^n X_i(\omega_i)$, but we also associate the weight $W(\omega_{1:n})$ with this particular $Z_{1:n}$. Instead of approximating $P_{Z_{1:n}}$ with the empirical distribution of the $Z_{1:n}$'s, we use the *weighted* empirical distribution, that is

$$P_{Z_{1:n}}(z) \approx \frac{\sum_{r=1}^R W^r \mathbb{1}\{Z_{1:n}^r = z\}}{\sum_{r=1}^R W^r},$$

where $Z_{1:n}^r$ and $W^r$ are the $r$th Monte Carlo samples of $Z_{1:n}$ and its weighting. $R$ is the number of Monte Carlo samples.

Any joint probability distribution on $\Omega_{1:n}$ can be described with an appropriate weighting function $W$. This flexibility causes computational problems, making exact methods impossible and making Monte Carlo methods inefficient. As we have seen, however, when the weighting function $W$ is Markov, we can efficiently compute the distribution of $Z_{1:n}$ using Markov dependent convolution.

#### 7.2.2.5 Example: uniformly jittering spikes with a relative refractory period and rebound

We continue modifying the example from Sections 7.2.1.4 and 7.2.2.3. We use the weighting function

$$W(\omega,\omega') = \begin{cases} 0 & \text{if } \omega' \le \omega + \tau_0 \\ 1/2 & \text{if } \omega + \tau_0 < \omega' \le \omega + \tau_1 \\ 2 & \text{if } \omega + \tau_2 < \omega' \le \omega + \tau_3 \\ 1 & \text{otherwise} \end{cases},$$

where $0 \le \tau_0 < \tau_1 < \tau_2 < \tau_3$. The weighting function is designed to model a hard refractory period of $\tau_0$ bins (following the preceding spike), followed by a relative refractory period between $\tau_0$ and $\tau_1$, followed by a normal period between $\tau_1$ and $\tau_2$, followed by a rebound period between $\tau_2$ and $\tau_3$, and finally a return to normal after $\tau_3$.

We chose to represent the normal period weighting as 1, but as we mentioned earlier, the exact value of the weighting is meaningless. The relative weightings carry meaning. In this context the weights were chosen so that spikes in the absolute refractory period (or before the previous spike) are impossible, spikes in the relative refractory period are half as likely (per unit time) as during the normal period, and spikes in the rebound period are twice as likely as during the normal period.

We can use Markov dependent convolution as in Example 7.2.2.3 to compute the distribution of synchrony exactly and efficiently. The generic Monte Carlo approach of sampling and weighting can be very slow to converge. It turns out that the Markov structure of the constraints can also be exploited for efficient sampling procedures.

#### 7.2.2.6 Markov sampling

We are given $(\Omega_i, U_i)$, $i = 1:n$, as before, (the $X_i$'s do not matter) but this time we would like to create a random sample $\omega_{1:n} \in \Omega_{1:n}$ according to the joint distribution

$$U^W_{1:n}(\omega_{1:n}) = \kappa^{-1} W(\omega_{1:n}) U_{1:n}(\omega_{1:n}) = \kappa^{-1} W(\omega_{1:n}) \prod_{i=1}^{n} U_i(\omega_i),$$

where $W : \Omega_{1:n} \mapsto [0, \infty)$ is a weighting function and $\kappa$ is a normalizing constant.

Assuming that $W$ has a Markov structure, namely, $W(\omega_{1:n}) = \prod_{i=2}^{n} W_i(\omega_{i-1}, \omega_i)$, then the joint distribution factors into

$$U^W_{1:n}(\omega_{1:n}) = \kappa^{-1} U_1(\omega_1) \prod_{i=2}^{n} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i). \tag{10}$$

We used this Markov factorization for efficient computation in the last several sections and we will use it here for efficient sampling.

The key observation is that $U^W_{1:n}$ is a Markov chain and can be rewritten as

$$U^W_{1:n}(\omega_{1:n}) = \mu_1(\omega_1) \prod_{i=2}^{n} \mu_{i|i-1}(\omega_{i-1}, \omega_i), \tag{11}$$

where $\mu_1(\omega_1) = U^W_{1:n}(\omega_1)$ and $\mu_{i|i-1}(\omega_{i-1}, \omega_i) = U^W_{1:n}(\omega_i|\omega_{i-1})$.

If we are given the initial distribution $\mu_1$ and the conditional distributions $\mu_{i|i-1}$ for $1 < i \leq n$, then sampling is easy. We first sample $\omega_1$ from $\mu_1$. Given this $\omega_1$, we next sample $\omega_2$ from the conditional distribution $\mu_{2|1}(\omega_1, \cdot)$ which is a probability distribution on $\Omega_2$ for each $\omega_1$. Given $\omega_2$ we sample $\omega_3$ from $\mu_{3|2}(\omega_2, \cdot)$, and so forth until we have an entire sample $\omega_{1:n}$. We can independently repeat this process to generate many Monte Carlo samples from $U^W_{1:n}$.

The Markov property does two things. First, it makes the sampling procedure Markov in the sense that we only need the value of $\omega_{i-1}$ to generate $\omega_i$. This is important because the memory requirements necessary to specify complete conditional distributions like $U^W_{1:n}(\omega_i|\omega_{1:i-1})$ would be overwhelming. Second, it makes it possible to efficiently compute the conditional distributions $\mu_{i|i-1}$ from the weighted representation in (10). We will now describe this process.

For $k = n$ define $\beta_n(\omega_{n-1}, \omega_n) = c_n^{-1} W_n(\omega_{n-1}, \omega_n) U_n(\omega_n)$ and for $1 < k < n$ define

$$\beta_k(\omega_{k-1}, \omega_k) = c_k^{-1} \sum_{\substack{\omega_{k+1:n} \in \\ \Omega_{k+1:n}}} \prod_{i=k}^{n} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i).$$

Each $\beta_k$ ($k > 1$) is defined over $\Omega_{k-1:k}$. The $c_k$'s are arbitrary positive constants that can be used to control overflow or underflow in an actual implementation. A reasonable method is to choose $c_k$ so that $\beta_k$ sums to 1, that is

$$c_k = \sum_{\substack{\omega_{k-1:n} \in \\ \Omega_{k-1:n}}} \prod_{i=k}^{n} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i).$$

The $\beta_k$'s can be computed recursively in reverse as evidenced by

$$\begin{aligned}
\beta_k(\omega_{k-1}, \omega_k) &= c_k^{-1} \sum_{\omega_{k+1} \in \Omega_{k+1}} \sum_{\substack{\omega_{k+2:n} \in \\ \Omega_{k+2:n}}} \prod_{i=k+1}^{n} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i)\; W_k(\omega_{k-1}, \omega_k) U_k(\omega_k) \\
&= c_k^{-1} c_{k+1} \sum_{\omega_{k+1} \in \Omega_{k+1}} \beta_{k+1}(\omega_k, \omega_{k+1}) W_k(\omega_{k-1}, \omega_k) U_k(\omega_k) \qquad (12)
\end{aligned}$$

for $1 < k < n$. If, for example, $c_k$ is chosen so that $\beta_k$ sums to 1, then we can replace $c_k^{-1} c_{k+1}$ above with a single (appropriate) normalization constant.

The $\beta_k$'s are useful because

$$\begin{aligned}
U_{1:n}^W(\omega_{1:k}) &= \sum_{\substack{\omega_{k+1:n} \in \\ \Omega_{k+1:n}}} U_{1:n}^W(\omega_{1:n}) \\
&= \kappa^{-1} U_1(\omega_1) \prod_{i=2}^{k-1} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i) \sum_{\substack{\omega_{k+1:n} \in \\ \Omega_{k+1:n}}} \prod_{i=k}^{n} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i) \\
&= \kappa^{-1} c_k U_1(\omega_1) \prod_{i=2}^{k-1} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i)\; \beta_k(\omega_{k-1}, \omega_k),
\end{aligned}$$

which lets us express

$$\begin{aligned}
U_{1:n}^W(\omega_k | \omega_{1:k-1}) &= \frac{U_{1:n}^W(\omega_{1:k})}{\sum_{\tilde{\omega}_k \in \Omega_k} U_{1:n}^W(\omega_{1:k-1}, \tilde{\omega}_k)} \\
&= \frac{\kappa^{-1} c_k U_1(\omega_1) \prod_{i=2}^{k-1} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i)\; \beta_k(\omega_{k-1}, \omega_k)}{\kappa^{-1} c_k U_1(\omega_1) \prod_{i=2}^{k-1} W_i(\omega_{i-1}, \omega_i) U_i(\omega_i) \sum_{\tilde{\omega}_k \in \Omega_k} \beta_k(\omega_{k-1}, \tilde{\omega}_k)} \\
&= \frac{\beta_k(\omega_{k-1}, \omega_k)}{\sum_{\tilde{\omega}_k \in \Omega_k} \beta_k(\omega_{k-1}, \tilde{\omega}_k)} \qquad (13)
\end{aligned}$$

for $1 < k \leq n$.

Since the expression for $U^W_{1:n}(\omega_k|\omega_{1:k-1})$ derived in (13) only depends on $\omega_{k-1}$ and $\omega_k$, we must have

$$\mu_{k|k-1}(\omega_{k-1}, \omega_k) = U^W_{1:n}(\omega_k|\omega_{k-1}) = U^W_{1:n}(\omega_k|\omega_{1:k-1}) = \frac{\beta_k(\omega_{k-1}, \omega_k)}{\sum_{\tilde{\omega}_k \in \Omega_k} \beta_k(\omega_{k-1}, \tilde{\omega}_k)}. \qquad (14)$$

This establishes the representation (11) and the claim that $U^W_{1:n}$ is a Markov chain. We have not dealt with the case $k = 1$, but using the same type of arguments it is easy to show that

$$\mu_1(\omega_1) = U^W_{1:n}(\omega_1) = \frac{\sum_{\omega_2 \in \Omega_2} \beta_2(\omega_1, \omega_2) U_1(\omega_1)}{\sum_{\tilde{\omega}_1 \in \Omega_1} \sum_{\omega_2 \in \Omega_2} \beta_2(\tilde{\omega}_1, \omega_2) U_1(\tilde{\omega}_1)},$$

where the sum over $\omega_2$ is analogous to the formula derived in (12) and the sum over $\tilde{\omega}_1$ is analogous to (13).

As we mentioned (12), (14) and (11) lead to an efficient sampling procedure for $U^W_{1:n}$. The sampling procedure has some overhead cost to convert the weighted representation into the Markov chain representation. Once this is finished, many samples can be easily generated.

We begin with the overhead computation. Let $L_k = |\Omega_k|$. The algorithm takes as input the collection of probability distributions (vectors) $U_k[1 : L_k]$, for $k = 1 : n$, and the collection of Markov weighting functions (matrices) $W_k[1 : L_{k-1}, 1 : L_k]$, for $k = 2 : n$. It outputs the initial distribution $\mu_1[1 : L_1]$ and the collection of conditional distributions $\mu_{k|k-1}[1 : L_{k-1}, 1 : L_k]$, for $k = 2 : n$, which we represent in the usual way, namely $\mu_{k|k-1}[i, j] = \mu_{k|k-1}(\Omega_{k-1}[i], \Omega_k[j])$. The support sets for all of these distributions are the $\Omega_k$'s and are not needed for the computations.

For $1 < k \leq n$ we recursively compute the $L_{k-1} \times L_k$ matrices

$$\beta_k[i, j] = W_k[i, j] U_k[j] \sum_{\ell=1}^{L_{k+1}} \beta_{k+1}[j, \ell]$$

beginning with $\beta_n[i, j] = W_n[i, j] U_n[j]$ and working backwards. After each $k$ we need to renormalize $\beta_k$ to prevent underflow or overflow. For $k = 1$ we compute the $L_1$-vector

$$\beta_1[j] = U_1[j] \sum_{\ell=1}^{L_2} \beta_2[j, \ell].$$

Then for each $1 < k \leq n$, we compute

$$\mu_{k|k-1}[i, j] = \frac{\beta_k[i, j]}{\sum_{\ell=1}^{L_k} \beta_k[i, \ell]}$$

and for $k = 1$ we compute

$$\mu_1[j] = \frac{\beta_1[j]}{\sum_{\ell=1}^{L_1} \beta_1[\ell]}.$$

Notice that the sum over $\ell$ in the formula for $\beta_k[i, j]$ can be computed independently of $i$ in an intermediate step.

Let $L$ bound the $L_k$'s. Then this algorithm requires $O(nL^2)$ elements of storage for the $W_k$'s, the $\beta_k$'s and the $\mu_k$'s, and it requires $O(nL^2)$ operations to compute the $\beta_k$'s and the $\mu_{k|k-1}$'s. The other computational requirements are insignificant compared to these.

Once we have computed the $\mu_{k|k-1}$'s, we can repeatedly draw independent samples from $U^W_{1:n}$ as we mentioned earlier. Sampling from a discrete distribution is straightforward and we do not go into the details here. Perhaps it is worth mentioning that efficient sampling takes $O(n \log L)$ operations for a single sample from $\Omega_{1:n}$. This requires more preprocessing of the $\mu_{k|k-1}$'s. Sampling directly from the $\mu_{k|k-1}$'s takes $O(nL)$ operations per sample.

#### 7.2.2.7 Intermediate convolutions

The previous section describes an algorithm for computing the Markov chain representation for $U^W_{1:n}$ from the Markov weighting function $W = W_2 \cdots W_n$ and the reference probabilities $U_i$ $(i = 1\!:\!n)$. In particular

$$U^W_{1:n}(\omega_{1:n}) = \mu_1(\omega_1) \prod_{i=2}^{n} \mu_{i|i-1}(\omega_{i-1}, \omega_i)$$

where $\mu_1(\omega_1) = U^W_{1:n}(\omega_1)$ and $\mu_{i|i-1}(\omega_{i-1}, \omega_i) = U^W_{1:n}(\omega_i|\omega_{i-1})$. This representation can be used for sampling, but it also has other uses. In particular, it can be used to compute the distribution of $Z_{1:k}$ for $k < n$.

Let $V_1 = \mu_1$ over $\Omega_1$, let $V_i(\omega_i) = 1/|\Omega_i|$ be the uniform distribution over $\Omega_i$ $(i = 2 : n)$ and let $\mu : \Omega_{1:n} \to [0, 1]$ be the Markov weighting function defined by $\mu(\omega_{1:n}) = \prod_{i=2}^{n} \mu_{i|i-1}(\omega_{i-1}, \omega_i)$. Notice that

$$V^\mu_{1:n}(\omega_{1:n}) = \kappa^{-1} \left[\prod_{i=2}^{n} |\Omega_i|\right]^{-1} \mu_1(\omega_1) \prod_{i=2}^{n} \mu_{i|i-1}(\omega_{i-1}, \omega_i) = \kappa^{-1} \left[\prod_{i=2}^{n} |\Omega_i|\right]^{-1} U^W_{1:n}(\omega_{1:n}).$$

Since both $V^\mu_{1:n}$ and $U^W_{1:n}$ sum to 1, we must have $V^\mu_{1:n} = U^W_{1:n}$.

Let $X_i$ $(i = 1 : n)$ be functions on $\Omega_i$ and define $Z_{1:k} = \sum_{i=1}^{k} X_i$. We can use Markov dependent convolution and either $U_{1:n}$ and $W$ or $V_{1:n}$ and $\mu$ to compute the distribution of $Z_{1:n}$.

$$P_{Z_{1:n}}(z) = \sum_{\omega_n} H_{Z_{1:n}}(z, \omega_n) \quad \text{where}$$
$$H_{Z_{1:n}} = \underbrace{X_1 *^{W_1} \cdots *^{W_n} X_n}_{\text{using } U_{1:n}} = \underbrace{X_1 *^{\mu_{2|1}} \cdots *^{\mu_{n|n-1}} X_n}_{\text{using } V_{1:n}}.$$

One advantage of using the Markov chain representation is that the intermediate convolutions are also meaningful.

Define

$$H^\mu_{Z_{1:k}} = \underbrace{X_1 *^{\mu_{2|1}} \cdots *^{\mu_{k|k-1}} X_k}_{\text{using } V_{1:k}}.$$

Note that in general, for $k < n$,

$$H^{\mu}_{Z_{1:k}} \neq \underbrace{X_1 *^{W_1} \cdots *^{W_k} X_k}_{\text{using } U_{1:k}}.$$

We have

$$P_{Z_{1:k}}(z) = \sum_{\omega_k \in \Omega_k} H^{\mu}_{Z_{1:k}}(z, \omega_k). \tag{15}$$

This relationship need not hold for intermediate Markov dependent convolutions using other representations. (15) follows directly from (9) applied to the $\mu$, $V_{1:n}$ representation.

#### 7.2.2.8 Tail probabilities (the Markov case)

Following Section 7.2.1.5 we focus on the upper tail probabilities $G(z) = \text{Prob}\{Z \geq z\}$. In particular, we are interested in the situation where we can alter an intermediate step in the recursive Markov dependent convolution algorithm and ensure that the altered final distribution has heavier tails than the original. The Markov situation is not as clean as the independent one. This section and the next are relevant for the variable partition jitter methods.

Let $H$ and $H'$ be Markov convolution functions on $\Omega$. We say that $H$ dominates $H'$ and we write $H \succeq H'$ whenever

$$\sum_z H(z, \omega) = \sum_z H'(z, \omega) \quad \text{for all } \omega \in \Omega \text{, and} \tag{16a}$$

$$\sum_{\zeta \geq z} H(\zeta, \omega) \geq \sum_{\zeta \geq z} H'(\zeta, \omega) \quad \text{for all } z \in \mathbb{R} \text{ and all } \omega \in \Omega. \tag{16b}$$

Note that (16a) is actually implied by (16b) because of the normalization constraint $\sum_{\omega,z} H(z, \omega) = \sum_{\omega,z} H'(z, \omega) = 1$. If $H$ and $H'$ are not defined on the same sample space $\Omega$, we say that $H \not\succeq H'$ (and also $H \neq H'$). It easy to see that $\succeq$ is transitive in this usage.

If $Z$ and $Z'$ are random variables defined by their distributions

$$P_Z(z) = \sum_{\omega} H(z, \omega) \quad \text{and} \quad P_{Z'}(z) = \sum_{\omega} H'(z, \omega),$$

then $H \succeq H'$ implies $G_Z \succeq G_{Z'}$. This comes directly from (16b). The next Lemma shows that $H \succeq H'$ persists after Markov dependent convolution. A proof is given at the end of this section.

**Lemma 7.2.4.** Suppose that $H, H'$ are Markov convolution functions on a finite sample space $\Omega_0$. Define the random variables $Z$ and $Z'$ by the distributions

$$P_Z(z) = \sum_{\omega_n \in \Omega_n} H_Z(z, \omega_n) \quad \text{and} \quad P_{Z'}(z) = \sum_{\omega_n \in \Omega_n} H_{Z'}(z, \omega_n),$$

respectively, where

$$H_Z = H *^{W_1} X_1 *^{W_2} \cdots *^{W_n} X_n \quad \text{and} \quad H_{Z'} = H' *^{W_1} X_1 *^{W_2} \cdots *^{W_n} X_n$$

for functions $X_i$ and reference probabilities $U_i$ on the finite sets $\Omega_i$ and for weighting functions $W_i : \Omega_{i-1:i} \mapsto [0,\infty)$ ($i = 1:n$). We refer to $n \geq 0$, $\Omega_i$, $U_i$, $X_i$, and $W_i$ ($i = 1:n$) as the parameters and if $H_Z$ or $H_{Z'}$ is not well defined, then we say that $Z$ or $Z'$ is undefined, respectively. The following equivalence holds: $H \succeq H'$ if and only if (i) $Z$ and $Z'$ are either both defined or both undefined for all choices of the parameters, and (ii) $H_Z \succeq H_{Z'}$ for all choices of the parameters for which $Z$ and $Z'$ are both defined. The statement is still true if we replace (ii) with (iii): $G_Z \succeq G_{Z'}$ for all choices of the parameters for which $Z$ and $Z'$ are both defined. Furthermore, if (16a) holds for $H$ and $H'$, then (i) holds.

Suppose we are given an indexed set of Markov convolution functions $H^\varphi$ (not necessarily all defined on the same sample space) for $\varphi \in \mathcal{L}$ and we want to choose a new collection of Markov convolution functions $\mathcal{M}$ with the following properties:

$$\text{for each } \varphi \in \mathcal{L} \text{ there exists an } H \in \mathcal{M} \text{ with } H \succeq H^\varphi; \tag{17a}$$

$$\text{for each } H \in \mathcal{M} \text{ there exists a } \varphi \in \mathcal{L} \text{ with } H = H^\varphi; \tag{17b}$$

$$\text{if } H_1, H_2 \in \mathcal{M} \text{ and } H_1 \neq H_2, \text{ then } H_1 \not\succeq H_2. \tag{17c}$$

This is essentially identical to (2). When $\mathcal{L}$ is finite we can use the same selection / pruning algorithm described in Section 7.2.1.5 to generate the unique set $\mathcal{M}$ satisfying (17).

A related problem is to choose the smallest set $\tilde{\mathcal{M}}$ satisfying (17a). Even if all of the $H^\varphi$ ($\varphi \in \mathcal{L}$) are defined on the same sample space $\Omega$, because of (16a) we cannot necessarily choose a single $\tilde{H}$ with $\tilde{H} \succeq H^\varphi$ for all $\varphi \in \mathcal{L}$. This differs markedly from the independent situation (cf., (3)).

Define an equivalence relation between Markov convolution functions as follows: $H \sim H'$ if $H$ and $H'$ are defined on the same sample space $\Omega$ and (16a) holds. This equivalence relation partitions $\mathcal{L}$ into disjoint equivalence classes $\mathcal{L}^j$, where $j$ indexes the equivalence classes. No single $\tilde{H}$ can dominate $H^\varphi$'s from multiple equivalence classes. The best we can hope to do is to dominate each equivalence class $\mathcal{L}^j$ with a single $\tilde{H}^j$, in which case $\tilde{\mathcal{M}}$ would have a single element for each equivalence class in $\mathcal{L}$.

When $\{H^\varphi\}_{\varphi \in \mathcal{L}^j}$ is finite, this strategy is achievable for $\mathcal{L}^j$. In particular, we can choose a single dominating $\tilde{H}^j$ such that

$$\sum_{\zeta \geq z} \tilde{H}^j(\zeta, \omega) = \max_{\varphi \in \mathcal{L}^j} \sum_{\zeta \geq z} H^\varphi(\zeta, \omega) \quad \text{for all } z \in \mathbb{R} \text{ and all } \omega \in \Omega^j, \tag{18}$$

where $\Omega^j$ is the common sample space. The proof is more or less identical to the proof of (3). That proof also demonstrates that this $\tilde{H}^j$ is the "smallest" dominating Markov convolution function in the sense that $\hat{H}^j \succeq \tilde{H}^j$ for any other $\hat{H}^j$ such that $\hat{H}^j \succeq H^\varphi$ for all $\varphi \in \mathcal{L}^j$.

We will now describe more detailed algorithms for generating $\mathcal{M}$ and $\tilde{\mathcal{M}}$. This closely follows the algorithms in Section 7.2.1.5. Order the elements of $\mathcal{L}$ as $\varphi_1, \ldots, \varphi_M$. Let $B_m$ and $\Omega_m$ be the support and sample space, respectively, for $H_m = H^{\varphi_m}$. It is convenient to work with the tail representation

$$C_m(z,\omega) = \sum_{\zeta \geq z} H_m(\zeta,\omega),$$

which can be represented as a $|B_m|{\times}|\Omega_m|$ matrix and computed with $O(|B_m||\Omega_m|)$ operations from a similar matrix representation of $H_m$. We will assume the matrices $C_m$ are defined so that $C_m[1,j] = \sum_z H_m(z,\Omega_m[j])$.

Using the $\Omega_m$ and the vectors $C_m[1,1\,{:}\,|\Omega_m|]$, we can partition $\mathcal{L}$ into equivalence classes $\mathcal{L}^j$, $j = 1:J$, as described above. One way to do this is to sort the collection of vectors $C_m[1,\cdot]$ which takes $O(ML\log M)$ operations, where $L$ bounds the $|\Omega_m|$.

To compute $\mathcal{M}$ or $\tilde{\mathcal{M}}$ we can use the selection / pruning algorithm or the maximization algorithm, respectively, on each equivalence class. These algorithms are described for the independent case in Section 7.2.1.5 and modifying them for the Markov case is straightforward.

The first step in each equivalence class is to represent the $C_m$ on a common alphabet, say $B^j$ for the $j$th equivalence class. This takes $O(M^j|B^j||\Omega^j| + M^j b^j \log M^j b^j)$ operations, where $M_j = |\mathcal{L}^j|$ is the number of elements in the $j$th equivalence class, $\Omega^j$ is the common sample space and $b^j$ bounds the size of the supports $B_m$ for the $j$th equivalence class.

Computing the dominating function for the $j$th equivalence class takes $O(M^j|B^j||\Omega^j|)$ operations. The entire computation for $\tilde{M}$ thus takes

$$\begin{aligned} &O\Big(ML\log M + \textstyle\sum_{j=1}^{J} M^j|B^j||\Omega^j| + M^j b^j \log M^j b^j\Big) \\ &\quad = O\big(ML\log M + MLB + Mb\log Mb\big) \end{aligned}$$

operations, where $L$ bounds the $|\Omega^j|$, $b$ bounds the $b^j$ and $B$ bounds the $|B^j|$. The selection / pruning algorithm for the $j$th equivalence class takes $O\big((M^j)^2|B^j||\Omega^j|\big)$ operations in the worst case and $O(M^j|B^j||\Omega^j|)$ operations in the best case. If each equivalence class has the best case, then the total computation for $\mathcal{M}$ uses the same number of operations as the computation for $\tilde{\mathcal{M}}$. The worst case is

$$O\big(ML\log M + M^2LB + Mb\log Mb\big)$$

operations.

**Proof of Lemma 7.2.4.** Let $H_0 = H$ and $H_0' = H'$. For $k = 1\,{:}\,n$ define

$$H_k = H *^{W_1} X_1 *^{W_2} \cdots *^{W_k} X_k \quad \text{and} \quad H_k' = H' *^{W_1} X_1 *^{W_2} \cdots *^{W_k} X_k.$$

Suppose (16a) holds for $H$ and $H'$. We first will prove by induction that either (16a) holds for $H_k$ and $H_k'$ for all $k = 0\,{:}\,n$ or $H_k$ and $H_k'$ are undefined for all $k$ large enough. This is true by hypothesis for $k = 0$.

Suppose it is true for some $0 \leq k < n$. We can assume that (16a) holds for $H_k$ and $H_k'$, otherwise $H_k$ and $H_k'$ must be undefined and so $H_{k+1}$ and $H_{k+1}'$ are also both undefined. Let $\kappa_k$ and $\kappa_k'$ be the normalizing constants in the definition of $H_k$ and $H_k'$, respectively. We have

$$\begin{aligned}\kappa_{k+1} &= \sum_{z,\omega_{k+1}} \sum_{\omega_k} H_k(z - X_{k+1}(\omega_{k+1}), \omega_k) W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \\ &= \sum_{\omega_{k+1},\omega_k} W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \sum_z H_k(z, \omega_k) \\ &= \sum_{\omega_{k+1},\omega_k} W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \sum_z H'_k(z, \omega_k) = \kappa'_{k+1}, \end{aligned} \tag{19}$$

where the next to last equality comes from (16a). If $\kappa_{k+1} = 0$, then $H_{k+1}$ is undefined and so is $H'_{k+1}$. Otherwise, they are both well defined and

$$\begin{aligned}\sum_z H_{k+1}(z, \omega_{k+1}) &= \sum_z \kappa_{k+1}^{-1} \sum_{\omega_k} H_k(z - X_{k+1}(\omega_{k+1}), \omega_k) W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \\ &= \kappa_{k+1}^{-1} \sum_{\omega_k} W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \sum_z H_k(z - X_{k+1}(\omega_{k+1}), \omega_k) \\ &= \kappa_{k+1}^{-1} \sum_{\omega_k} W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \sum_z H_k(z, \omega_k) \\ &\overset{(a)}{=} (\kappa'_{k+1})^{-1} \sum_{\omega_k} W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \sum_z H'_k(z, \omega_k) \overset{(b)}{=} \sum_z H'_{k+1}(z, \omega_{k+1}), \end{aligned} \tag{20}$$

where $(a)$ comes from (16a) applied to $H_k$ and $H'_k$ and from (19), and where $(b)$ comes from reversing the algebra that led up to $(a)$. The completes the induction. Note that we have now established (i) whenever (16a) holds for $H$ and $H'$.

Now further suppose that $H \succeq H'$ and suppose $Z$ and $Z'$ are well defined. We will use induction to show that $H_k \succeq H'_k$ for all $k = 0 : n$. By definition $H_0 \succeq H'_0$. Suppose that $H_k \succeq H'_k$. We have already established (19) and (20) so

$$\begin{aligned}\sum_{\zeta \ge z} H_{k+1}(\zeta, \omega_{k+1}) &= \sum_{\zeta \ge z} \kappa_{k+1}^{-1} \sum_{\omega_k} H_k(\zeta - X_{k+1}(\omega_{k+1}), \omega_k) W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \\ &= \kappa_{k+1}^{-1} \sum_{\omega_k} W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \sum_{\zeta \ge z} H_k(z - X_{k+1}(\omega_{k+1}), \omega_k) \\ &\ge (\kappa'_{k+1})^{-1} \sum_{\omega_k} W_{k+1}(\omega_k, \omega_{k+1}) U_{k+1}(\omega_{k+1}) \sum_{\zeta \ge z} H'_k(z - X_{k+1}(\omega_{k+1}), \omega_k) \\ &= \sum_{\zeta \ge z} H'_{k+1}(\zeta, \omega_{k+1}), \end{aligned}$$

where the inequality comes from (16b) for $H_k$ and $H'_k$. This completes the induction.

Note that we have shown that $H \succeq H'$ (and $Z$ well defined) implies that $H_Z \succeq H_{Z'}$, which implies

$$G_Z(z) = \sum_{\zeta \ge z} P_Z(\zeta) = \sum_{\omega_n} \sum_{\zeta \ge z} H(\zeta, \omega_n) \ge \sum_{\omega_n} \sum_{\zeta \ge z} H'(\zeta, \omega_n) = \sum_{\zeta \ge z} P_{Z'}(\zeta) = G_{Z'}(z)$$

for all $z$. So $G_Z \succeq G_{Z'}$. This completes half of the proof.

Now suppose $H \not\succeq H'$. We will construct three examples (for three different cases) where either (i) fails or (iii) fails for certain choices of the parameters. Note that (ii) implies (iii), so that if (iii) fails, then (ii) does also. First, consider the case where

$$\sum_z H(z,\omega_0') > 0 = \sum_z H'(z,\omega_0') \tag{21}$$

for some $\omega_0'$. Taking $n=1$, $\Omega_1 = \{\omega_1'\}$, $W_1(\omega_0,\omega_1) = \mathbb{1}\{\omega_0 = \omega_0'\}\mathbb{1}\{\omega_1 = \omega_1'\}$, $U_1(\omega_1) = \mathbb{1}\{\omega_1 = \omega_1'\}$ and $X_1 \equiv 0$ gives

$$\begin{aligned}\kappa_1' &= \sum_z \sum_{\omega_1\in\Omega_1} H_1'(z,\omega_1) = \sum_z \sum_{\omega_1\in\Omega_1} \sum_{\omega_0\in\Omega_0} H_0'(z-0,\omega_0)\mathbb{1}\{\omega_0=\omega_0'\}\mathbb{1}\{\omega_1=\omega_1'\} \\ &= \sum_z H'(z,\omega_0') = 0.\end{aligned}$$

The same algebra gives $\kappa_1 > 0$. So $H_Z = H_1$ is well defined, but $H_{Z'} = H_1'$ is not and (i) does not hold. The same thing happens when $H$ and $H'$ are reversed in (21).

Now, consider the more general case where (16a) does not hold. We have already dealt with the case when (21) holds (or its equivalent with $H$ and $H'$ reversed), so we can assume that it does not hold and we can choose $\omega_0', \omega_0'' \in \Omega_0$ such that

$$\frac{\sum_z H(z,\omega_0')}{\sum_z H(z,\omega_0'')} > \frac{\sum_z H'(z,\omega_0')}{\sum_z H'(z,\omega_0'')} \tag{22}$$

and such that the numerators and denominators on both sides are positive. Taking $n=1$, $\Omega_1 = \{\omega_1',\omega_1''\}$, $W_1(\omega_0,\omega_1) = \mathbb{1}\{(\omega_0,\omega_1)\in(\omega_0',\omega_1')\cup(\omega_0'',\omega_1'')\}$, $U_1(\omega_1) = \frac{1}{2}\mathbb{1}\{\omega_1 \in \omega_1' \cup \omega_1''\}$, $X_1(\omega_1) = z'\mathbb{1}\{\omega_1 = \omega_1'\} + z''\mathbb{1}\{\omega_1 = \omega_1''\}$ and $z = z' + z''$ for $z' < \min\{B\cup B'\}$ and $z'' > \max\{B \cup B'\}$, where $B$ and $B'$ are the supports of $H$ and $H'$, respectively, gives

$$\begin{aligned}G_Z(z) &= \sum_{\zeta\ge z} P_Z(\zeta) = \sum_{\zeta \ge z'+z''} \sum_{\omega_1\in\Omega_1} H_1(\zeta,\omega_1) \\ &= \frac{\sum_{\zeta\ge z'+z''}\sum_{\omega_1\in\Omega_1}\sum_{\omega_0\in\Omega_0} H_0(\zeta - X_1(\omega_1),\omega_0)\frac{1}{2}\mathbb{1}\{(\omega_0,\omega_1)\in(\omega_0',\omega_1')\cup(\omega_0'',\omega_1'')\}}{\sum_\zeta\sum_{\omega_1\in\Omega_1}\sum_{\omega_0\in\Omega_0} H_0(\zeta - X_1(\omega_1),\omega_0)\frac{1}{2}\mathbb{1}\{(\omega_0,\omega_1)\in(\omega_0',\omega_1')\cup(\omega_0'',\omega_1'')\}} \\ &= \frac{\sum_{\zeta\ge z'+z''} H(\zeta - z',\omega_0') + \sum_{\zeta\ge z'+z''} H(\zeta - z'',\omega_0'')}{\sum_\zeta H(\zeta,\omega_0') + \sum_\zeta H(\zeta,\omega_0'')} \\ &= \frac{\sum_{\zeta\ge z''} H(\zeta,\omega_0') + \sum_{\zeta\ge z'} H(\zeta,\omega_0'')}{\sum_\zeta H(\zeta,\omega_0') + \sum_\zeta H(\zeta,\omega_0'')} = \frac{0 + \sum_\zeta H(\zeta,\omega_0'')}{\sum_\zeta H(\zeta,\omega_0') + \sum_\zeta H(\zeta,\omega_0'')} \\ &= \left(1 + \frac{\sum_\zeta H(\zeta,\omega_0')}{\sum_\zeta H(\zeta,\omega_0'')}\right)^{-1}.\end{aligned}$$

The same formula holds for $\mathrm{Prob}\{Z' \geq z\}$ by replacing $H$ with $H'$ and we have

$$G_Z(z) = \left(1 + \frac{\sum_\zeta H(\zeta, \omega_0')}{\sum_\zeta H(\zeta, \omega_0'')}\right)^{-1} < \left(1 + \frac{\sum_\zeta H'(\zeta, \omega_0')}{\sum_\zeta H'(\zeta, \omega_0'')}\right)^{-1} = G_{Z'}(z),$$

so $G_Z \not\succeq G_{Z'}$. The inequality comes from (22).

Finally, consider the case where (16a) holds, but (16b) does not hold. Choose $z$ and $\omega_0'$ so that

$$\sum_{\zeta \geq z} H(\zeta, \omega_0') < \sum_{\zeta \geq z} H'(\zeta, \omega_0').$$

Taking $n = 1$, $\Omega_1 = \{\omega_1'\}$, $W_1(\omega_0, \omega_1) = \mathbb{1}\{\omega_0 = \omega_0'\}\mathbb{1}\{\omega_1 = \omega_1'\}$, $U_1(\omega_1) = \mathbb{1}\{\omega_1 = \omega_1'\}$ and $X_1 \equiv 0$ gives

$$\begin{aligned} G_Z(z) &= \sum_{\zeta \geq z} \sum_{\omega_1 \in \Omega_1} H_1(\zeta, \omega_1) \\ &= \frac{\sum_{\zeta \geq z} \sum_{\omega_1 \in \Omega_1} \sum_{\omega_0 \in \Omega_0} H_0(\zeta - 0, \omega_0)\mathbb{1}\{\omega_0 = \omega_0'\}\mathbb{1}\{\omega_1 = \omega_1'\}}{\sum_{\zeta} \sum_{\omega_1 \in \Omega_1} \sum_{\omega_0 \in \Omega_0} H_0(\zeta - 0, \omega_0)\mathbb{1}\{\omega_0 = \omega_0'\}\mathbb{1}\{\omega_1 = \omega_1'\}} = \frac{\sum_{\zeta \geq z} H(\zeta, \omega_0')}{\sum_\zeta H(\zeta, \omega_0')}. \end{aligned}$$

As before the same formula holds for $Z'$ and $H'$, giving

$$G_Z(z) = \frac{\sum_{\zeta \geq z} H(\zeta, \omega_0')}{\sum_\zeta H(\zeta, \omega_0')} < \frac{\sum_{\zeta \geq z} H'(\zeta, \omega_0')}{\sum_\zeta H'(\zeta, \omega_0')} = G_{Z'}(z)$$

because the denominators are the same and positive. So $G_Z \not\succeq G_{Z'}$ and the proof is complete. □

#### 7.2.2.9 Example: optimal partitions (the Markov case)

This example mirrors the example in Section 7.2.1.6 for the Markov case. As in the independent case, we are given a finite sequence of integers $\omega^*_{1:n}$ with $\omega^*_i \leq \omega^*_{i+1}$ for each $i = 1{:}n-1$. For each pair of integers $(k, \ell)$ with $k < \ell$ and each $i = 1{:}n$, we are given a state space $\Omega_i^{k,\ell}$, a reference probability distribution $U_i^{k,\ell}$ and a function $X_i^{k,\ell}$ on $\Omega_i^{k,\ell}$. Also, for each $i = 2{:}n$ and each pair of state spaces $\Omega_{i-1}^{k,\ell}$ and $\Omega_i^{k',\ell'}$, we are given a nonnegative weighting function $W_i^{k,\ell,k',\ell'}$ on $\Omega_{i-1}^{k,\ell} \times \Omega_i^{k',\ell'}$.

Let $\pi = \cdots < \pi_{m-1} < \pi_m < \cdots$ be a partition of the integers. For any partition $\pi$ define the random variable $Z^\pi$ by its distribution

$$P^\pi(z) = \sum_\omega H^\pi(z, \omega) \quad \text{for} \quad H^\pi = X_1^\pi *^{W_2^\pi} X_2^\pi *^{W_3^\pi} \cdots *^{W_n^\pi} X_n^\pi,$$

where

$$X_i^\pi = X_i^{k(\omega_i^*, \pi), \ell(\omega_i^*, \pi)} \quad \text{and} \quad W_i^\pi = W_i^{k(\omega_{i-1}^*, \pi), \ell(\omega_{i-1}^*, \pi), k(\omega_i^*, \pi), \ell(\omega_i^*, \pi)}$$

for the integer valued functions

$$k(\omega,\pi) = \max_m \{\pi_m : \pi_m < \omega\} \text{ and } \ell(\omega,\pi) = \min_m \{\pi_m : \pi_m \geq \omega\}.$$

This is identical to the independent situation described in Section 7.2.1.6 except that here we have added the weighting functions to create dependencies in the $X_i$'s. Note that $Z^\pi$ may be undefined if $H^\pi$ is not well defined.

Let $G^\pi = G_{Z^\pi}$ be the tail probability function for the random variable $Z^\pi$. We are interested in the following function:

$$G^*(z) = \max_{\pi \in \mathcal{L}(r,R)} G^\pi(z).$$

The maximum is over all partitions in the set

$$\mathcal{L}(r,R) = \left\{\pi : \min_m \pi_m - \pi_{m-1} \geq r \text{ and } \max_m \pi_m - \pi_{m-1} \leq R \text{ and } Z^\pi \text{ is defined}\right\}$$

for some $1 \leq r \leq R < \infty$.

Lemma 7.2.2 still holds, so the brute force approach of computing each $G^\pi$ using (in this case) Markov dependent convolution is possible, but not practical. The intuition for the more efficient (and sometimes practical) algorithms described for the independent situation continues to hold for the Markov case. We replace the notion of $G \succeq G'$ with the notion of $H \succeq H'$. One important difference is that we no longer have good control of the number of operations for the bounding algorithm. So for some examples, even bounding $G^*$ may be impractical in the Markov case.

Define

$$H_i^\pi = X_1^\pi *^{W_2^\pi} \cdots *^{W_i^\pi} X_i^\pi.$$

Following the discussion after (7), we will define $H_0^\pi = h_0$ so that $Z_0^\pi$ is identically zero. For each integer $\ell$ define

$$K_\ell = K_\ell(\omega_{1:n}^*) = \{\# \text{ of } \omega_i \leq \ell\} = \sum_{i=1}^n \mathbb{1}\{\omega_i^* \leq \ell\}$$

and

$$\mathcal{L}_\ell(r,R) = \{\pi \in \mathcal{L}(r,R) : \pi_m = \ell \text{ for some } m \text{ and } H_{K_\ell}^\pi \text{ is well defined}\}.$$

Let $\mathcal{M}_\ell$ be the unique collection of Markov convolution functions with the following properties:

$$\text{for each } \pi \in \mathcal{L}_\ell(r,R) \text{ there exists an } H \in \mathcal{M}_\ell \text{ with } H \succeq H_{K_\ell}^\pi; \tag{23a}$$

$$\text{for each } H \in \mathcal{M}_\ell \text{ there exists a } \pi \in \mathcal{L}_\ell(r,R) \text{ with } H = H_{K_\ell}^\pi; \tag{23b}$$

$$\text{if } H_1, H_2 \in \mathcal{M} \text{ and } H_1 \neq H_2, \text{ then } H_1 \not\succeq H_2. \tag{23c}$$

As in the independent case, we can recursively generate $\mathcal{M}_\ell$. The recursion is easy to initialize because $\mathcal{M}_k = \{h_0\}$ for all $k < \omega_1^*$. The stopping criterion is the same:

$$G^*(z) = \max_{\omega_n^* \le \ell \le \omega_n^* + R - 1} \max_{\pi \in \mathcal{L}_\ell(r,R)} G^\pi(z) = \max_{\omega_n^* \le \ell \le \omega_n^* + R - 1} \max_{G \in \mathcal{M}_\ell} G(z).$$

The first equality is trivial because each $\pi \in \mathcal{L}(r,R)$ has a segment ending somewhere in $[\omega_n^* : \omega_n^* + R - 1]$. Also, for $\ell \ge \omega_n^*$, $K_\ell = n$, so $Z^\pi$ is defined exactly when $H^\pi_{K_\ell}$ is well defined. The second equality comes almost directly from (23a) and (23b) for the cases $\le$ and $\ge$, respectively. This makes use of the equivalence from Lemma 7.2.4.

Suppose we have already computed $\mathcal{M}_k$ for all $k < \ell$. Lemma 7.2.3 remains valid in this situation and takes care of those $\ell$ in long gaps between the elements of $\omega^*_{1:n}$. If Lemma 7.2.3 does not apply, then we compute $\mathcal{M}_\ell$ using the following procedure.

For each $k = \ell - R : \ell - r$ and each $H \in \mathcal{M}_k$, compute the Markov convolution function

$$H^{H,k,\ell} = H *^{W^{k_H,\ell_H,k,\ell}_{K_k+1}} X^{k,\ell}_{K_k+1} *^{W^{k,\ell,k,\ell}_{K_k+2}} \cdots *^{W^{k,\ell,k,\ell}_{K_\ell}} X^{k,\ell}_{K_\ell},$$

where $H$ is defined over $\Omega^{k_H,\ell_H}_{K_k}$. If $K_k = 0$, then $H = h_0$ and we can take $\Omega^{k_H,\ell_H}_{K_k} = \Omega_0$ and $W^{k_H,\ell_H,k,\ell}_{K_k+1} \equiv 1$. This is just for initialization. If $K_k + 1 > K_\ell$, that is, there are no elements of $\omega^*_{1:n}$ in $(k,\ell]$, then we simply take $H^{H,k,\ell} = H$. Note that if there are any elements of $\omega^*_{1:n}$ in $(k,\ell]$, then the sample space for $H^{H,k,\ell}$ will be $\Omega^{k,\ell}_{K_\ell}$.

Let $\mathcal{M}'_\ell = \{(k,H) : \ell - R \le k \le \ell - r, H \in \mathcal{M}_k \text{ and } H^{H,k,\ell} \text{ is well defined}\}$. For each $\varphi = (k,H) \in \mathcal{M}'_\ell$, let $H^\varphi = H^{H,k,\ell}$. Note that $|\mathcal{M}'_\ell|$ is finite. We can use the selection / pruning algorithm described in the previous section to choose a unique set of Markov convolution functions $\mathcal{M}_\ell$ from $\{H^\varphi\}_{\varphi \in \mathcal{M}'_\ell}$ that satisfy (17) with $\mathcal{L}$ replaced by $\mathcal{M}'_\ell$ and $\mathcal{M}$ replaced by $\mathcal{M}_\ell$. This $\mathcal{M}_\ell$ is the same $\mathcal{M}_\ell$ implicitly defined by (23).

**Proof that this $\mathcal{M}_\ell$ satisfies (23).** Choose $\pi \in \mathcal{L}_\ell(r,R)$ and let $(k,\ell]$ be a segment in $\pi$ for some $\ell - R \le k \le \ell - r$. So $\pi \in \mathcal{L}_k(r,R)$ and there is a $H' \in \mathcal{M}_k$ with $H' \succeq H^\pi_{K_k}$. We have

$$H^{H',k,\ell} = H' *^{W^{k_{H'},\ell_{H'},k,\ell}_{K_k+1}} X^{k,\ell}_{K_k+1} *^{W^{k,\ell,k,\ell}_{K_k+2}} \cdots *^{W^{k,\ell,k,\ell}_{K_\ell}} X^{k,\ell}_{K_\ell} \quad \text{and}$$
$$H^\pi_{K_\ell} = H^\pi_{K_k} *^{W^{k_{H'},\ell_{H'},k,\ell}_{K_k+1}} X^{k,\ell}_{K_k+1} *^{W^{k,\ell,k,\ell}_{K_k+2}} \cdots *^{W^{k,\ell,k,\ell}_{K_\ell}} X^{k,\ell}_{K_\ell},$$

so Lemma 7.2.4 shows that $H^{H',k,\ell} \succeq H^\pi_{K_\ell}$ (the latter is well defined because $\pi \in \mathcal{L}_\ell(r,R)$). Finally, because of the selection / pruning algorithm, there is an $H$ in the set $\mathcal{M}_\ell$ that we just created with $H \succeq H^{H',k,\ell} \succeq H^\pi_{K_\ell}$ and we see that this $\mathcal{M}_\ell$ does indeed satisfy (23a).

Now choose $H \in \mathcal{M}_\ell$. The selection / pruning algorithm guarantees that there is a $\varphi = (k,H') \in \mathcal{M}'_\ell$ with $H = H^\varphi = H^{H',k,\ell}$. Since $H' \in \mathcal{M}_k$, there is a $\pi' \in \mathcal{L}_k(r,R)$ with $H' = H^{\pi'}_{K_k}$ and also a $\pi \in \mathcal{M}_k$ that is identical to $\pi'$ up to $k$ and with the segment $(k,\ell]$. So $H' = H^\pi_{K_k}$ as well. For this $\pi$, we have

$$\begin{aligned} H^\pi_{K_\ell} &= H^\pi_{K_k} *^{W^{k_{H'},\ell_{H'},k,\ell}_{K_k+1}} X^{k,\ell}_{K_k+1} *^{W^{k,\ell,k,\ell}_{K_k+2}} \cdots *^{W^{k,\ell,k,\ell}_{K_\ell}} X^{k,\ell}_{K_\ell} \\ &= H^{\pi'}_{K_k} *^{W^{k_{H'},\ell_{H'},k,\ell}_{K_k+1}} X^{k,\ell}_{K_k+1} *^{W^{k,\ell,k,\ell}_{K_k+2}} \cdots *^{W^{k,\ell,k,\ell}_{K_\ell}} X^{k,\ell}_{K_\ell} = H^{H',k,\ell}, \end{aligned}$$

so $H = H^{H',k,\ell} = H^{\pi}_{K_\ell}$ and $\mathcal{M}_\ell$ satisfies (23b). The selection / pruning algorithm also gives (23c) and the proof is complete. $\square$

The recursive bounding algorithm is very similar. We just replace the selection / pruning algorithm with the smallest dominating set ($\tilde{\mathcal{M}}$) algorithm described in the previous section. Unlike the independent setting, we cannot always choose a dominating set with a single element. This means that the bounding algorithm can also require exponentially many convolutions in the worst case.

**Computation complexity.** As in the independent setting, the number of operations required for the recursive computation of $G^*$ depends heavily on the number of elements in the $\mathcal{M}_\ell$'s. Following the proof of Lemma 7.2.2, there are $(R+r)(R-r+1)/2$ different possible state spaces $\Omega_i^{k,\ell}$ for the $i$th Markov dependent convolution. These state spaces will generally differ, so there are typically at least $(R+r)(R-r+1)/2$ different equivalence classes of Markov convolution functions in $\{H_i^\pi\}_{i\in\mathcal{L}(r,R)}$ (and maybe many more). Neither the selection / pruning algorithm nor the smallest dominating set algorithm can reduce the number of equivalence classes, so it makes sense to analyze the number of operations using the bound $|\mathcal{M}_\ell| \le M(R+r)(R-r+1)/2$ for all $\ell$. The best case is $M = 1$. To analyze the situation we will also use the bounds $|\bigcup_\pi B_i^\pi| \le b_i$ for all $i$ and $|\Omega_i^\pi| \le L$ for all $i$ and $\pi$, where is $B_i^\pi$ the support of $H_i^\pi$ and $\Omega_i^\pi$ is the state space.

Following the discussion for the independent setting and making the appropriate adjustments, the number of operations for the recursive computation of $G^*$ can be computed as follows.

$$\#\text{ of operations for convolutions} = C(n)O(MR(R+r)(R-r+1)^2).$$

Note that when the supports grow linearly, that is $b_i = O(ai)$, then $C(n) = O(n^2aL(L + \log naL))$; see Section 7.2.2.1.

$$\begin{aligned}&\#\text{ of operations for selection / pruning}\\ &\quad = O(\gamma)\sum_{i=1}^{n} O(M'M'Lb_i + M'L\log M' + M'b_i\log M'b_i),\end{aligned}$$

where $M' = M(R+r)(R-r+1)^2/2$ and $\gamma = O(R^2/(R-r))$. The extra $M'$ in the leading term is a worst case situation.

Combining these two, assuming that $R - r \approx R$ and assuming that $b_i = O(ai)$, gives

$$\begin{aligned}&\#\text{ of operations for computing } G^*\\ &\quad = O(n^2aL(L+\log naL))O(MR^4)\\ &\qquad + O(R)O(MR^3MR^3Ln^2a + MR^3L\log MR + MR^3n^2a\log MRna)\\ &\quad = O(MR^4L\log MR + MR^4n^2a(L(L+\log naL) + \log MRna + MR^3L)).\end{aligned}$$

The final $MR^3L$ term is a worst case situation. In the best case and in the (smallest) bounding algorithm it disappears completely.

Variable partition jitter depends on a parameter $\Delta$ with $R - r \approx R \approx L \approx \Delta$. In the best case and with $M = 1$, this gives $O\big(n^2 a\Delta^6(\Delta + \log na)\big)$ operations for variable partition jitter. Note that we essentially always have to be in this ideal situation to use the algorithm (and that depends heavily on the specifics of the problem), and even then $\Delta$ cannot be very large.

### 7.2.3 Convolution on arbitrary graphs

Independent convolution and Markov dependent convolution are each examples of a more general convolution framework. Consider a collection of discrete random variables $V_{1:N}$. Let $G$ be an undirected graph with $N$ vertices labeled $\{1, \ldots, N\}$. We associate $V_k$ with the $k$th vertex and we use $B_k$ to denote the range of $V_k$. The distribution $P_{V_{1:N}}$ of $V_{1:N}$ respects the graph $G$ if $P_{V_{1:N}}$ can be factored into terms that depend only on the cliques (completely connected subgraphs) of $G$, that is

$$P_{V_{1:N}}(v_{1:N}) = \kappa^{-1} \prod_{C \in \mathcal{C}(G)} f_C(v_C), \tag{24}$$

where $\mathcal{C}(G)$ is the collection of the cliques of $G$ and the $f_C$'s are nonnegative functions indexed by cliques. If $C = \{i_1, \ldots, i_m\} \subseteq \{1, \ldots, N\}$, we use the notation $v_C = (v_{i_1}, \ldots, v_{i_m})$ and similarly for $V_C$. The function $f_C$ is defined on $B_C = B_{i_1} \times \cdots \times B_{i_m}$. The constant $\kappa$ is a normalization constant. Computation in this general framework is often amenable to generalized dynamic programming approaches. See [7, 11, 8] for references and reviews.

Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ be a subset of $\{1, \ldots, N\}$. We are interested in the distribution $P_Z$ of the random variable $Z = \sum_{a \in \mathcal{A}} V_a = \sum_{i=1}^n V_{a_i}$. This can be computed using a general convolution framework that is a mixture of dynamic programming on the graph $G$ and convolution in the usual sense.

Choose an ordering of the vertices $V_{M_1}, \ldots, V_{M_N}$. This choice can drastically affect computation; see for example [8]. Without loss of generality we can assume that the ordering is $V_1, \ldots, V_N$. This simplifies the notation. We will compute $P_Z$ recursively by traversing this ordering. Some important quantities (that depend on the ordering and the graph) are

$$\begin{aligned}
\mathcal{C}_k &= \{C \in \mathcal{C}(G) : C \cap \{1, \ldots, k\} \neq \emptyset\} \\
\mathcal{N}_k &= \bigcup_{C \in \mathcal{C}_k} C = \{j : j \leq k \text{ or } j \text{ has an edge to some } i \leq k\} \\
\mathcal{D}_k &= \mathcal{N}_k \setminus \{1, \ldots, k\} = \{j > k : j \text{ has an edge to some } i \leq k\} \\
\mathcal{A}_k &= \mathcal{A} \cap \mathcal{N}_k.
\end{aligned}$$

$\mathcal{C}_k$ is the set of all cliques in $G$ that contain at least one of the first $k$ vertices. $\mathcal{N}_k$ is the set of the first $k$ vertices and all of their neighbors in $G$. $\mathcal{D}_k$ is the “boundary” in $G$ of the first $k$ vertices. Define

$$F_k(z, v_{\mathcal{D}_k}) = \kappa_k^{-1} \sum_{\substack{v_{1:k} \in B_{1:k}: \\ \sum_{a \in \mathcal{A}_k} v_a = z}} \prod_{C \in \mathcal{C}_k} f_C(v_C), \tag{25}$$

where $\kappa_k$ is a positive constant chosen so that $F_k$ sums to 1, that is

$$\kappa_k = \sum_{v_{\mathcal{N}_k} \in B_{\mathcal{N}_k}} \prod_{C \in \mathcal{C}_k} f_C(v_C).$$

Note that we need only consider $z \in \oplus_{a \in \mathcal{A}_k} B_a$ and $v_{\mathcal{D}_k} \in B_{\mathcal{D}_k}$. When $\mathcal{D}_k$ is empty, then we think about $F_k(z, v_{\mathcal{D}_k}) = F_k(z)$ as a function of $z$ only. Here and below we define $\sum_{a \in \emptyset} v_a \equiv 0$.

For $k > 1$, we can compute $F_k$ recursively from $F_{k-1}$ as follows:

$$\begin{aligned} F_k(z, v_{\mathcal{D}_k}) &= \kappa_k^{-1} \sum_{v_k \in B_k} \sum_{\substack{v_{1:k-1} \in B_{1:k-1}: \\ \sum_{a \in \mathcal{A}_{k-1}} v_a = \\ z - \sum_{a' \in \mathcal{A}_k \setminus \mathcal{A}_{k-1}} v_{a'}}} \prod_{C \in \mathcal{C}_{k-1}} f_C(v_C) \underbrace{\prod_{C' \in \mathcal{C}_k \setminus \mathcal{C}_{k-1}} f_{C'}(v_{C'})}_{\text{does not depend on } v_{1:k-1}} \\ &= \tilde{\kappa}_k^{-1} \sum_{v_k \in B_k} F_{k-1}\big(z - \textstyle\sum_{a \in \mathcal{A}_k \setminus \mathcal{A}_{k-1}} v_a, v_{\mathcal{D}_{k-1}}\big) \displaystyle\prod_{C \in \mathcal{C}_k \setminus \mathcal{C}_{k-1}} f_C(v_C), \end{aligned} \tag{26}$$

where $\tilde{\kappa}_k^{-1} = \kappa_k^{-1} \kappa_{k-1}$ is chosen so that $F_k$ sums to 1. We define $\prod_{C \in \emptyset} f_C \equiv 1$. The support of $F_k$ is the set

$$B_{F_k} = \{z : F_k(z, v_{\mathcal{D}_k}) > 0 \text{ for some } v_{\mathcal{D}_k} \in B_{\mathcal{D}_k}\}.$$

Since $B_{F_k} \subseteq B_{F_{k-1}} \oplus \{\oplus_{a \in \mathcal{A}_k \setminus \mathcal{A}_{k-1}} B_a\}$, we need only consider $z$ in the latter set, which can also be computed recursively. (We take $\oplus_{a \in \emptyset} B_a = \{0\}$.) $B_{F_k}$ can then be computed from $F_k$ using the definition.

Notice that $\mathcal{D}_N = \emptyset$, $\mathcal{C}_N = \mathcal{C}(G)$, and $\mathcal{A}_N = \mathcal{A}$, so comparing (24) and (25) gives

$$P_Z(z) = \sum_{\substack{v_{1:N} \in B_{1:N}: \\ \sum_{a \in \mathcal{A}} v_a = z}} P_{V_{1:N}}(v_{1:N}) = F_N(z).$$

Depending on the problem, (26) can be used to efficiently compute $F_N$ and thus $P_Z$. The support $B_{F_N}$ of $F_N$ is also the support of $Z$.

#### 7.2.3.1 Example: independent convolution

Suppose we are given $n$ independent random variables $X_k$ with distributions $P_{X_k}$ and supports $A_{X_k}$ and we want to compute the distribution $P_Z$ of $Z = \sum_{k=1}^n X_k$. We can use the recursive convolution method from Section 7.2.1, namely

$$P_Z = P_{X_1} * \cdots * P_{X_n}.$$

We will translate that method into the more general setting.

Let $N = n$, let $V_k = X_k$ $(k = 1 : n)$ and let $\mathcal{A} = \{1, \ldots, N\}$. We want to compute the distribution of $Z = \sum_{k=1}^n X_k = \sum_{a \in \mathcal{A}} V_a$. Since the $V_k$'s are independent, their joint distribution is

$$P_{V_{1:N}}(v_{1:N}) = \prod_{k=1}^N P_{V_k}(v_k) = \prod_{k=1}^N P_{X_k}(v_k),$$

which obeys the graph $G$ with no edges, as shown in Figure 7.1. Since $\mathcal{D}_k = \emptyset$ for all $k$, (26) becomes

$$F_k(z) = \tilde{\kappa}_k^{-1} \sum_{v_k \in A_{X_k}} F_{k-1}(z - v_k) P_{X_k}(v_k).$$

Note that $\tilde{\kappa}_k^{-1} = 1$ and this is the standard convolution recursion.

### 7.2.3.2 Example: independent convolution of random variables

Suppose we are given $n$ independent random variables $X_k$ specified by the triples $(\Omega_k, U_k, X_k)$ and we want to compute the distribution $P_Z$ of $Z = \sum_{k=1}^n X_k$. We can use the recursive convolution of random variables method from Section 7.2.1, namely

$$P_Z = X_1 * \cdots * X_n.$$

We will translate that method into the more general setting.

Let $N = 2n$ and let $\mathcal{A} = \{2k-1 : k = 1, \ldots, n\}$. Let $V_{2k}$ $(k = 1:n)$ be independent random variables where $V_{2k}$ takes values in $\Omega_k$ with distribution $P_{V_{2k}} = U_k$, and let $V_{2k-1} = X_k(V_{2k})$. The joint distribution of the $V_k$'s is

$$P_{V_{1:N}}(v_{1:N}) = \prod_{k=1}^n P_{V_{2k}}(v_{2k}) P_{V_{2k-1}|V_{2k}}(v_{2k-1}|v_{2k}) = \prod_{k=1}^n U_k(v_{2k}) \mathbb{1}\{v_{2k-1} = X_k(v_{2k})\},$$

which obeys the graph $G$ with edges only between vertices $2k$ and $2k-1$, as shown in Figure 7.2. Note that $Z$ has the same distribution as $\sum_{a \in \mathcal{A}} V_a$.

$\mathcal{D}_{2k-1} = \{2k\}$ and $\mathcal{D}_{2k} = \emptyset$ for all $k$, so (26) becomes

$$\begin{aligned} F_{2k-1}(z, v_{2k}) &= \tilde{\kappa}_{2k-1}^{-1} \sum_{v_{2k-1}} F_{2(k-1)}(z - v_{2k-1}) \mathbb{1}\{v_{2k-1} = X_k(v_{2k})\} \\ &\overset{(a)}{=} \tilde{\kappa}_{2k-1}^{-1} F_{2(k-1)}(z - X_k(v_{2k})) \end{aligned}$$

and

$$F_{2k}(z) = \tilde{\kappa}_{2k}^{-1} \sum_{v_{2k} \in \Omega_k} F_{2k-1}(z, v_{2k}) U_k(v_{2k}) \overset{(b)}{=} \hat{\kappa}_{2k}^{-1} \sum_{v_{2k} \in \Omega_k} F_{2(k-1)}(z - X_k(v_{2k})) U_k(v_{2k}),$$

where $(b)$ comes from substituting the formula from $(a)$. Note that we have expressed the recursion in terms of $F_{2k}$ $(k = 1 : n)$. Since $\hat{\kappa}_{2k} = \tilde{\kappa}_{2k}\tilde{\kappa}_{2k-1}$ is chosen so that $F_{2k}$ sums to one, $\hat{\kappa}_{2k} = 1$ and $(b)$ is the recursive formula derived in Section 7.2.1.2 for independent convolution of random variables.

### 7.2.3.3 Example: Markov dependent convolution

Consider the usual Markov dependent convolution setup

$$P_Z = X_1 *^{W_2} X_2 *^{W_3} \cdots *^{W_n} X_n$$

for the "random variables" $(\Omega_k, U_k, X_k)$, for $k = 1:n$, and the weighting functions $W_k$ for $k = 2:n$.

Let $N = 2n$ and let $\mathcal{A} = \{2k-1 : k = 1, \ldots, n\}$. Let $V_{2k}$ $(k = 1:n)$ be random variables with joint distribution

$$P_{V_2,V_4,\ldots,V_N}(v_2, v_4, \ldots, v_N) = \kappa^{-1} U_1(v_2) \prod_{k=2}^{n} W_k(v_{2(k-1)}, v_{2k}) U_k(v_{2k}),$$

where $V_{2k}$ takes values in $\Omega_k$. Let $V_{2k-1} = X_k(V_{2k})$. The joint distribution of the $V_k$'s is

$$P_{V_{1:N}}(v_{1:N}) = \kappa^{-1} U_1(v_2) \prod_{k=2}^{n} W_k(v_{2(k-1)}, v_{2k}) U_k(v_{2k}) \prod_{k=1}^{n} \mathbb{1}\{v_{2k-1} = X_k(v_{2k})\},$$

which obeys the graph $G$ with edges between vertices $2k$ and $2k-1$ and with edges between vertices $2k$ and $2(k-1)$, as shown in Figure 7.3. Note that $Z$ has the same distribution as $\sum_{a \in \mathcal{A}} V_a$.

$\mathcal{D}_{2k-1} = \{2k\}$ for all $k$, $\mathcal{D}_{2k} = \{2(k+1)\}$ for all $k < n$ and $\mathcal{D}_{2n} = \emptyset$, so (26) becomes

$$\begin{aligned} F_{2k-1}(z, v_{2k}) &= \tilde{\kappa}_{2k-1}^{-1} \sum_{v_{2k-1}} F_{2(k-1)}(z - v_{2k-1}, v_{2k}) \mathbb{1}\{v_{2k-1} = X_k(v_{2k})\} \\ &= \tilde{\kappa}_{2k-1}^{-1} F_{2(k-1)}(z - X_k(v_{2k}), v_{2k}) \end{aligned}$$

and

$$\begin{aligned} F_{2k}(z, v_{2(k+1)}) &= \tilde{\kappa}_{2k}^{-1} \sum_{v_{2k} \in \Omega_k} F_{2k-1}(z, v_{2k}) U_k(v_{2k}) W_{k+1}(v_{2k}, v_{2(k+1)}) \\ &= \hat{\kappa}_{2k}^{-1} \sum_{v_{2k} \in \Omega_k} F_{2(k-1)}(z - X_k(v_{2k}), v_{2k}) U_k(v_{2k}) W_{k+1}(v_{2k}, v_{2(k+1)}) \end{aligned}$$

for $k < n$, and

$$F_N(z) = F_{2n}(z) = \hat{\kappa}_{2n}^{-1} \sum_{v_{2n} \in \Omega_n} F_{2(n-1)}(z - X_n(v_{2n}), v_{2n}) U_n(v_{2n}).$$

As before, this gives a recursion over $F_{2k}$ for $k = 1:n$. While this is not the identical recursion that we used in Section 7.2.2 for Markov dependent convolution, it is quite similar. Both methods give the same answer for $P_Z$, of course.

## 7.3 Spike Centered Jitter

The intuition behind jitter methods is a method itself, which we call *spike centered jitter*. It has many drawbacks which are corrected in later versions, but remains an important technique because of its speed and simplicity.

### 7.3.1 Monte Carlo jitter

Consider the following Monte Carlo method for generating a random spike train that is similar to a fixed observed spike train $t^*_{1:n^*} = (t^*_1, \dots, t^*_{n^*})$ with $n^*$ spikes: Independently perturb each spike time $t^*_k$ by a random amount uniformly distributed in the interval $(-\Delta/2, \Delta/2]$. Call this new time $\tilde{T}^1_k$. The collection of perturbed times $\tilde{T}^1_{1:n^*}$ can be ordered to create a new spike train $T^1_{1:n^*}$. This new train has the property that for each spike $t^*_k$ in the original train, there is a spike $T^1_\ell$ in the new train with $|t^*_k - T^1_\ell| \le \Delta/2$. (Usually this happens for $k = \ell$, but right now we are allowing the perturbed spikes to swap ordinal positions.) We can independently repeat this process $R$ times to get a collection of random, artificial spike trains $T^r_{1:n^*}$, $r = 1\!:\!R$. Figure 7.4 details this process.

All of these spike trains are similar to the original and to each other. They each have $n^*$ spikes (right now we allow more than one spike at the same time) and for any two trains $r, s$ and any spike time $T^r_k$ in train $r$, there is a spike time $T^s_\ell$ in train $s$ with $|T^r_k - T^s_\ell| \le \Delta$. Note that this similarity is parameterized by $\Delta$, thus giving us a notion of "$\Delta$-similar". Figure 7.5 clearly shows that each of these spike trains preserves the original structure at coarse resolutions much larger than $\Delta$ and that the collection as a whole is unstructured at fine resolutions much smaller than $\Delta$. Only the original spike train can have non-accidental structure at finer resolutions than $\Delta$.

We want to evaluate how unusual the original spike train looks compared to the collection of simulated spike trains. The intuition is that any differences must be a result of fine temporal structure that is present in the original but not in the artificial collection. One way of doing this is to specify (beforehand) some statistic $Z$ that assigns a number to a spike train and that measures something about fine temporal resolution. Let $z^* = Z(t^*_{1:n^*})$ be the original value of the statistic and let $Z^r = Z(T^r_{1:n^*})$, $r = 1\!:\!R$, be the value of the statistic on the $r$th artificial spike train. We can then measure the unusualness of the original spike train by how much $z^*$ is an outlier in the empirical distribution of the $Z^r$'s. For example, we could use the tail probability

$$\hat{\alpha}^* = \frac{\{\#\text{ of } Z^r \ge z^*\} + 1}{R+1} = \frac{\sum_{r=1}^R \mathbb{1}\{Z^r \ge z^*\} + 1}{R+1}. \tag{27}$$

If the sequence $z^*, Z^1, \dots, Z^R$ was independent and identically distributed (i.i.d.) under some null hypothesis $H_0$, or more generally, exchangeable under $H_0$, then $\hat{\alpha}^*$ could be interpreted as the $p$-value in a hypothesis test for $H_0$ (see Section 7.7). Unfortunately, $z^*, Z^1, \dots, Z^R$ may be neither i.i.d. nor exchangeable. The reason is that the original spike train (which created $z^*$) is special among the rest of the collection. Since it generated them all, it is in the center of the collection. This is discussed further in Section 7.3.4.

### 7.3.2 Choice of statistic

In principle the statistic is any function $Z$ of the entire spike train. For our exact methods detailed below, the function must be of an appropriate form. In particular, we will assume that

$$Z(t_{1:n}) = \sum_{k=1}^{n} X(t_k), \tag{28}$$

that is, $Z$ evaluated on a spike train with many spikes is just the sum of a fixed function $X$ evaluated on each spike individually. Section 7.7 discusses situations where (28) does not hold. (Technically, the exact jitter methods will still work if $X$ also depends on $k$.)

Within this additive constraint, we are free to choose $X$. We happen to believe that a particularly powerful $X$ for detecting fine temporal structure will involve some sort of synchrony measure between the jittered spike and a fixed, comparison spike train. For example, let $s_{1:m}$, be the spike times of another spike train, perhaps recorded at the same time as the original spike train. A common measure of synchrony is

$$X(t_k) = \mathbb{1}\left\{ \min_{1 \le \ell \le m} |t_k - s_\ell| \le \epsilon \right\}, \tag{29}$$

for some small $\epsilon$. The particular application will dictate the specifics of the test statistic.

In practice, for our exact methods to be efficient, we must also constrain $Z$ to take values in some discrete alphabet which remains relatively small for typical sized spike trains. The easiest way to accomplish this is to have $X$ take values in some small subset of a lattice, like the integers. For example, in the synchrony example above, $X$ takes values in $\{0, 1\}$ and $Z$ takes values in $\{0, \dots, n\}$ for spike trains with at most $n$ spikes.

### 7.3.3 Exact jitter

One drawback of the Monte Carlo spike centered jitter is the inherent randomness of Monte Carlo methods. The statistics $Z^{1:R}$, their empirical distribution and any property of it, like the function $\hat{\alpha}^*$ in (27), all inherit the randomness of the artificial collection of spike trains. As we increase the size $R$ of the collection of artificial spike trains, the effects of this randomness get smaller, but the tradeoff is increased computation. In many cases, however, it is possible to efficiently compute the limiting value (as $R \to \infty$) of a Monte Carlo jitter method, thus removing all of the Monte Carlo randomness and saving a lot of computation. We call these *exact jitter methods*.

Consider the Monte Carlo spike centered jitter detailed in Section 7.3.1 and evaluated using some functional of the empirical distribution of the $Z^r$'s, like $\hat{\alpha}^*$. As $R \to \infty$ the empirical distribution converges to some limiting distribution $P$. If the $Z^r$'s come from an additive function as in (28) and take values in some finite alphabet, then we can compute $P$ and find the limiting value of any functional like $\hat{\alpha}^*$.

Let $\tilde{T}_k$, $k = 1, \dots, n^*$, be independent and uniformly distributed random variables (r.v.'s) in the interval $t_k^* \pm \Delta/2$, and let $Z = \sum_{k=1}^{n^*} X(\tilde{T}_k)$. The $Z^r$'s are i.i.d. with the same distribution as $Z$, so their limiting empirical distribution $P$ is also the distribution of $Z$ (by the law of large numbers, loosely speaking). We can thus compute $P$ by convolving the distributions of $X(\tilde{T}_k)$ for $k = 1, \dots, n^*$. In the next sections we carefully detail this algorithm and lay out the framework for the remaining jitter methods.

### 7.3.3.1 Notation and assumptions

Although not strictly necessary for exact spike centered jitter, later exact jitter methods will essentially require restricting all times to a fixed, equispaced grid with resolution $\delta$. In practice, $\delta$ is usually 1 ms or maybe a few $\mu$s depending on the statistic and the time scales of interest. We will always work in units of the grid and we assume that $\Delta$ is a positive integer in these units. Spike times will also be restricted to the grid, that is, integer valued. The grid can be conceptualized as demarcating the centers of disjoint, equisized bins into which spikes are placed. The exact timing within the bin is ignored. $\Delta = 1$ corresponds to the trivial case of no jitter, so we will always assume that $\Delta > 1$.

To emphasize that we are working in grid units, we will use $\omega_{1:n}$ to denote spike times, instead of $t_{1:n}$, which we reserve for possibly continuous valued times. For any sequence $\omega_{1:n}$ of integers (grid points), we associate the additive function

$$Z(\omega_{1:n}) = \sum_{k=1}^{n} X(\omega_k),$$

where $X$ is a real-valued function taking values in a finite alphabet $A_X$. Since $X$ is only evaluated at the grid points, this essentially means that we are assuming $X$ is constant within each bin. We define $Z(\emptyset) = 0$.

Let $A_n = \oplus_{i=1}^{n} A_X$ be the set of all possible values of $Z$ evaluated on $n$ spikes. For efficient computation it will be important that $A_n$ does not grow too quickly with $n$. This essentially means that $A_X$ must look something like $\{-ad, \ldots, -2d, -d, 0, d, \ldots, ad\}$ for some small $a$. In this case $|A_n| \approx n|A_X|$ which is much smaller than the worst case of $|A_n| \approx n^{|A_X|}$. This constraint is not important for describing the method, however.

The observed spike train is denoted $\omega^*_{1:n^*}$ for integers (grid points) $\omega^*_k \leq \omega^*_{k+1}$, and $n^*$ is reserved for the observed spike count. We allow the possibility that multiple spikes fall in the same bin, that is, they have the same quantized spike time. Each of the jitter methods is based on perturbing the $\omega^*_k$ in some random manner. We will use $T_{1:n^*}$ to denote the random vector of perturbed spike times, where $T_k$ denotes the random variable corresponding to $k$th spike time. The distribution of $T_{1:n^*}$ will depend on $\omega^*_{1:n^*}$ and the specifics of the method under consideration. Once the distribution of $T_{1:n^*}$ has been specified, the random variable of interest will be

$$Z = Z(T_{1:n^*}) = \sum_{k=1}^{n^*} X(T_k).$$

In particular, we will be interested in the distribution of $Z$,

$$P(z) = \text{Prob}\{Z = z\} \quad \text{for } z \in A_{n^*}.$$

Occasionally we will focus on

$$\alpha^* = \sum_{z \geq z^*} P(z) = \text{Prob}\{Z \geq z^*\},$$

where $z^* = Z(t^*_{1:n^*})$. We will always assume that $n^* > 0$. When $n^* = 0$ we are in the trivial situation where $Z = z^* = 0$, so $P$ is a point mass at 0 and $\alpha^* = 1$.

Monte Carlo jitter methods are based on generating many i.i.d. samples of $T_{1:n^*}$ and $Z$. $P$ is approximated by the empirical distribution of these samples. Exact jitter methods compute $P$ directly without sampling.

#### 7.3.3.2 An exact algorithm

Let $\Omega_k = \Omega_k[1:L_k] \subset \mathbb{Z}$ be a sequence of $L_k$ grid points containing $\omega_k^*$. For spike centered jitter, each $L_k = \Delta$ and

$$\Omega_k = [1:\Delta] - \lfloor \Delta/2 \rfloor - 1 + t_k^*,$$

which is a window of $\Delta$ consecutive grid points centered around $t_k^*$. The exact algorithm will not make use of this fact, but the statistical interpretation will depend heavily on it.

Define the function $X_k$ over $\Omega_k$ by $X_k(\omega) = X(\omega)$. We can also think about $X_k$ as a vector $X_k[1:L_k]$, where $X_k[i] = X_k(\Omega_k[i]) = X(\Omega_k[i])$.

For spike centered jitter, each random artificial spike train $T_{1:n^*}$ is created by independently and uniformly selecting a spike time (grid point) $\omega_k$ within $\Omega_k$ and ordering the resulting spike times. The statistic $Z = Z(T_{1:n}^*)$ does not depend on the ordering, so we can ignore this and express

$$Z = \sum_{k=1}^{n^*} X(\omega_k).$$

To exactly compute $P$, the distribution of $Z$, we can use independent convolution, detailed in Section 7.2.1, applied to the random variables $(\Omega_k, U_k, X_k)$, for $k = 1 : n^*$, where

$$U_k(\omega) = \frac{1}{L_k}\mathbb{1}\{\omega \in \Omega_k\}$$

is the uniform distribution over $\Omega_k$. Using the nonstandard convolution notation described in Section 7.2.1 we have

$$P = X_1 * \cdots * X_{n^*}.$$

### 7.3.4 Statistical problems

Spike centered jitter returns a probability distribution $P$ and an observation $z^*$ in the support of $P$. $P$ is an attempt at quantifying the potential variation in $z^*$ if $t_{1:n^*}^*$ was unstructured at time scales smaller than $\Delta$. As we mentioned in Section 7.3, we can quantify the unusualness of $t_{1:n^*}^*$ by some measure of how much $z^*$ is an outlier in $P$, like the right tail probability $\alpha^*$. But what does $\alpha^*$ mean in this context?

If $z^*$ had distribution $P$ under some suitable null hypothesis $H_0$, then $\alpha^*$ would be the $p$-value for a hypothesis test of $H_0$. Let $\mathbb{P}^*$ be the random process that generated $t_{1:n^*}^*$ and let $T_{1:n}^*$ be a random element with distribution $\mathbb{P}^*$. Define $\mathcal{B}(t_{1:n}, \Delta)$ to be the set of all spike trains created by jittering the spike times in $t_{1:n}$ in windows of length $\Delta$ centered around each spike. Thinking about how $P$ was created, $H_0$ must be something like

$H_0$: For each $t_{1:n}$, conditioned on the event $T_{1:n}^* \in \mathcal{B}(t_{1:n}, \Delta)$, the exact spike times in $T_{1:n}^*$ are distributed independently and uniformly within $\mathcal{B}(t_{1:n}, \Delta)$.

Ignoring the fact that this $H_0$ leads to the uninteresting hypothesis that "all spike trains with the same number of spikes are equally likely," which is not at all what we wanted to test, there is a more fundamental problem with the way we would be testing $H_0$ with $P$ and $\alpha^*$.

By construction, spike centered jitter returns

$$P(z) = \text{Prob}\{Z(T^*_{1:n}) = z | T^*_{1:n} \in \mathcal{B}(t^*_{1:n^*}, \Delta); H_0\}$$

for any process in $H_0$. Unfortunately, $z^*$ is not a sample from $P$ because we know that $z^*$ came from the specific event when $T^*_{1:n} = t^*_{1:n^*}$, instead of an arbitrary event within $\mathcal{B}(t^*_{1:n^*}, \Delta)$. So we cannot use $P$ and $z^*$ to construct a hypothesis test and $\alpha^*$ should not be interpreted as a $p$-value.

## 7.4 Fixed Partition Jitter

The fixed partition jitter method is designed to fix the interpretation problems of spike centered jitter outlined in the previous section. Fixed partition jitter returns an $\alpha^*$ that can be interpreted in the context of a hypothesis test. We will continue with the assumptions from Section 7.3.3.1. Fixed partition jitter depends on an *a priori* fixed partition of time $\pi = \cdots < \pi_m < \pi_{m+1} < \cdots$. The interpretation of the partition will be discussed in Section 7.4.1. For that interpretation to make sense, the partition must be chosen independently of the observation $t^*_{1:n^*}$. Typically, each segment of the partition will have length on the order of $\Delta$, that is, $\pi_m - \pi_{m-1} \approx \Delta$, but this is not important for describing the method.

An important quantity associated with a partition $\pi$ and a spike train $t_{1:n}$ is the number of spikes in each segment, $N(\pi, t_{1:n}) = \ldots, N_m(\pi, t_{1:n}), N_{m+1}(\pi, t_{1:n}), \ldots$, where

$$N_m(\pi, t_{1:n}) = \{\# \text{ of } t_k \in (\pi_{m-1}, \pi_m]\} = \sum_{k=1}^{n} \mathbb{1}\{\pi_{m-1} < t_k \leq \pi_m\}. \tag{30}$$

The Monte Carlo intuition behind fixed partition jitter is that we randomly generate spike trains that preserve $N(\pi, t^*_{1:n^*})$. For each spike $t^*_k$ we independently and uniformly perturb it within the segment of the partition that it occurred in. The only difference between fixed partition jitter and spike centered jitter is that the jitter windows are chosen beforehand, independently of the observed spike train, and two windows cannot partially overlap. Figure 7.6 shows some example simulated spike trains using a partition with segments of length $\Delta$. As in Figure 7.5, structures on time scales larger than $\Delta$ are preserved and structures on time scales smaller than $\Delta$ are destroyed.

We evaluate using the same methods as before. For the Monte Carlo method we compute $Z^r = Z^r(T^r_{1:n^*})$ for the $r$th artificial spike train and approximate $P$ by the empirical distribution of the $Z^r$'s.

The exact jitter method is also identical to the one we used for spike centered jitter. We quantize the observed spike train (now denoted $\omega^*_{1:n^*}$) and the partition (still denoted $\pi$, but now in new units). For the $k$th spike we use the jitter window

$$\Omega_k = [\pi_{m_k-1} + 1 \colon \pi_{m_k}], \quad \text{where } \pi_{m_k-1} < \omega^*_k \leq \pi_{m_k}, \tag{31}$$

which is the segment of the partition that contains the spike. Once the jitter window is fixed, the random variables are $X_k(\omega) = X(\omega)$ for our statistic $X$ and for grid points $\omega \in \Omega_k$. The jitter distributions $U_k$ are uniform over each jitter window. We can compute

$$P = X_1 * \cdots * X_{n^*}.$$

The computational costs and the benefits over the Monte Carlo method are basically the same as in spike centered jitter.

In both Monte Carlo and exact fixed partition jitter we only consider those segments of the partition that actually contain spikes. Partitions that differ in regions without spikes are indistinguishable using these jitter methods.

### 7.4.1 Statistical interpretation

The distribution $P$ returned by fixed partition jitter (with segments of length $\Delta$) is presumably quite similar to that returned by spike centered jitter (with a jitter window of length $\Delta$). In both cases, spikes are jittered uniformly in small windows of length $\Delta$ around each observed spike. The only difference is the exact offset of the windows from the observed spikes. Unlike spike centered jitter, however, fixed partition jitter leads to an $\alpha^*$ that can be used in an actual hypothesis test.

Let $\mathbb{P}^*$ be the random process that generated $t^*_{1:n^*}$ and let $T^*_{1:n}$ be a random element with distribution $\mathbb{P}^*$. Fix a partition $\pi$ and let $N(\pi, T^*_{1:n})$ be the spike counts in each segment of the partition, as in (30). The null hypothesis $H_0(\pi)$ is that $\mathbb{P}^*$ has the following property:

$H_0(\pi)$: Conditioned on $N(\pi, T^*_{1:n})$, the exact spike times in $T^*_{1:n}$ are distributed independently and uniformly within each segment of $\pi$.

To illustrate what we mean by this, we will describe how to sample from $T^*_{1:n}$ conditioned on $N(\pi, T^*_{1:n})$ under $H_0(\pi)$. For each segment $(\pi_{m-1}, \pi_m]$, choose $N_m(\pi, T^*_{1:n})$ spike times independently and uniformly within the segment. Then order all of the spike times (nondecreasing) to get $T^*_{1:n}$.

$H_0(\pi)$ is a compound null hypothesis because the distribution of $N(\pi, T^*_{1:n})$ is not specified, however, once a realization of $N(\pi, T^*_{1:n})$ is fixed, then every process in $H_0(\pi)$ has the same conditional distribution. An example of a process that is included in $H_0(\pi)$ is an inhomogeneous Poisson process with an intensity rate that is constant within each segment of the partition $\pi$. Note that $H_0(\pi)$ depends strongly on the specific partition $\pi$.

Define $Z^* = Z(T^*_{1:n})$. By construction, fixed partition jitter returns

$$P(z) = \text{Prob}\{Z^* = z | N(\pi, T^*_{1:n}) = N(\pi, t^*_{1:n^*}); H_0(\pi)\}$$

for any process in $H_0(\pi)$. Since $z^*$ is a sample from this distribution (under the null), we can use this to construct a hypothesis test in the usual manner. The difference between this and spike centered jittered is that the event $N(\pi, t^*_{1:n^*})$ hides information about the specific spike times in $t^*_{1:n^*}$, whereas the event $\mathcal{B}(t^*_{1:n^*}, \Delta)$ does not.

Fix a level of significance $\alpha$ for a (one-sided) hypothesis test of $H_0(\pi)$. For each possible realization of $N(\pi, T^*_{1:n})$, compute the lower end point $C_\alpha = C_\alpha(N(\pi, T^*_{1:n}))$ of the critical region such that

$$\text{Prob}\{Z^* \geq C_\alpha | N(\pi, T^*_{1:n}); H_0(\pi)\} \leq \alpha. \tag{32}$$

Taking the expected value over $N(\pi, T^*_{1:n})$ gives

$$\text{Prob}\{Z^* \geq C_\alpha | H_0(\pi)\} = E\big[\text{Prob}\{Z^* \geq C_\alpha | N(\pi, T^*_{1:n}); H_0(\pi)\}\big] \leq E[\alpha] = \alpha.$$

So this method of creating a critical region leads to a hypothesis test for $H_0(\pi)$ with significance $\leq \alpha$. (Ideally, we would replace the $\leq$ with $=$ in (32) to get a test with significance of exactly $\alpha$, but such a selection of $C_\alpha$ might not be possible because of the discrete nature of $Z^*$.)

Of course we do not have to compute $C_\alpha$ for every possible realization of $N(\pi, T^*_{1:n})$, only the observed realization $N(\pi, t^*_{1:n^*})$. In particular, $c^*_\alpha = C_\alpha(N(\pi, t^*_{1:n^*}))$ can be chosen as

$$c^*_\alpha = \min\big\{c \in \mathcal{A}_{n^*} : \textstyle\sum_{z \geq c} P(z) \leq \alpha\big\},$$

where $P$ is the distribution returned by fixed partition jitter. We reject if $z^* \geq c^*_\alpha$ and fail to reject, otherwise. Note that $z^* \geq c^*_\alpha$ exactly when $\alpha^* \leq \alpha$. In this way we can also interpret $\alpha^*$ as a $p$-value for a hypothesis test of $H_0(\pi)$. We reject if $\alpha^* \leq \alpha$ and fail to reject, otherwise.

One point that we have neglected is when the statistic $Z$ is itself stochastic, for example, when $X$ is some sort of synchrony measure with respect to a comparison spike train $s_{1:m}$ as in (29). In this case, for the preceding analysis to be correct the null hypothesis $H_0(\pi)$ must include the further assumption that

$H_0(\pi)$: *(Addendum)* Conditioned on $N(\pi, T^*_{1:n})$, the choice of the statistic $Z$ is independent of the exact spike times in $T^*_{1:n}$.

We return to this point further in Section 7.8.

If each of the segments in the partition has length $\approx \Delta$, then the null hypothesis implies that temporal precision does not exist at times scales much shorter than $\Delta$. All of the structure is found in the variation of the rates among different segments of the partition. An extreme example is useful for illustrating what this means.

Consider two neighboring segments of the partition, each with length $\Delta$. Suppose the first segment almost always has 0 spikes in it (low rate in the Poisson case) and the second almost always has a spike in each bin (high rate in the Poisson case). The rate jumps instantaneously between the two segments and a spike occurs with high probability near the boundary. This is a type of fine temporal precision, however, the presence of a spike near the boundary is implied by the rate over the entire segment of length $\Delta$. In some sense it is an artifact of structure at a coarse time scale. Indeed, the only way for the process to create a spike with fine temporal precision is to fill an entire segment with spikes.

As this example illustrates, fixed partition jitter does not consider instantaneous rate changes to be fine temporal structure as long as the rate stays constant for a while after it changes. In certain cases this is desirable, like when a rapid stimulus onset creates a quick but sustained change in firing rate and we do not want to think about this as fine structure in the spiking process. In other cases, we might want to identify this sort of phenomena as fine temporal structure. The jitter methods found in [1] can begin to address this latter situation.

### 7.4.2 Intuitive drawbacks

The main benefit of fixed partition jitter over spike centered jitter is that $\alpha^*$ is a $p$-value. The null hypothesis that we are testing, however, seems a little strange. Among other things, why should we know the particular partition $\pi$ ahead of time. If we reject the null, are we rejecting the spirit of the null, namely, that firing rates are nearly piecewise constant over segments of length $\Delta$, or are we rejecting the particular partition that we chose? Maybe there is another partition, perhaps offset slightly from the one that we chose, that better matches the true process and that we would not have rejected. The next method, variable partition jitter, takes this possibility into account.

## 7.5 Variable Partition Jitter

Unlike the previous two methods, variable partition jitter is designed specifically for testing a particular null hypothesis. It tries to remedy the arbitrary choice of the partition in fixed partition jitter by exhaustively exploring a large class of potential partitions. In certain cases this exhaustive search can be computationally feasible because of an efficient pruning strategy. In the remaining cases, bounds on the $p$-value can still be quickly computed. The focus is more about $\alpha^*$ than $P$.

### 7.5.1 The null hypothesis

Let $\mathbb{P}^*$ be the random process that generated $t^*_{1:n^*}$ and let $T^*_{1:n}$ be a random element with distribution $\mathbb{P}^*$. Let $N(\pi, T^*_{1:n})$ be the sequence of spike counts in each segment of the partition as in (30). For any partition $\pi$ define the minimum and maximum segment lengths

$$L_{\min}(\pi) = \min_m \pi_m - \pi_{m-1} \quad \text{and} \quad L_{\max}(\pi) = \max_m \pi_m - \pi_{m-1}.$$

The null hypothesis $H_0(\Delta)$ is that $\mathbb{P}^*$ has the following property:

$H_0(\Delta)$: There exists a partition $\pi$ with $L_{\min}(\pi) \geq \Delta$ such that conditioned on $N(\pi, T^*_{1:n})$, the exact spike times in $T^*_{1:n}$ are distributed independently and uniformly within each segment of $\pi$.

This is a compound null hypothesis. It includes all allowable partitions and for each fixed partition $\pi$ it includes all possible distributions of $N(\pi, T^*_{1:n})$. Comparing this situation to the fixed partition case in Section 7.4.1, we see that the only difference is that $H_0(\Delta)$ does not specify the exact partition. In fact, another way to state $H_0(\Delta)$ is to say that $\mathbb{P}^*$ satisfies $H_0(\pi)$ for some $\pi$ with $L_{\min}(\pi) \geq \Delta$.

$H_0(\Delta)$ more or less quantifies what we mean in this chapter by no fine temporal structure at time scales shorter than $\Delta$. The intuition and the caveats are the same as in fixed partition jitter without the strange requirement that we know the partition. The allowable partitions can have segments much longer than $\Delta$. This is not a problem. A rejection of $H_0(\Delta)$ implies that we need a partition that includes some segments with lengths smaller than $\Delta$ in order to explain the data.

The following simple fact about $H_0(\pi)$ is crucial for our implementation of variable partition jitter:

**Lemma 7.5.1.** If $\mathbb{P}^*$ satisfies $H_0(\pi)$, then $\mathbb{P}^*$ also satisfies $H_0(\tilde{\pi})$ for any refinement $\tilde{\pi}$ of $\pi$.

A proof can be found at the end of this section. The basic idea is that if spikes are distributed uniformly over a segment, then they are also distributed uniformly over any smaller subsegment. Or, in the case of a Poisson process, if the rate is constant over a large time interval, then it is constant over any smaller subinterval.

Any segment of length $2\Delta$ or greater can be split into multiple disjoint segments with lengths between $\Delta$ and $2\Delta$. So any partition $\pi$ with $\Delta \leq L_{\min}(\pi)$ can be refined into a partition $\tilde{\pi}$ with $\Delta \leq L_{\min}(\tilde{\pi}) \leq L_{\max}(\tilde{\pi}) < 2\Delta$. This observation and Lemma 7.5.1 let us restate $H_0(\Delta)$ as

$H_0(\Delta)$: There exists a partition $\pi$ with $\Delta \leq L_{\min}(\pi) \leq L_{\max}(\pi) < 2\Delta$ such that conditioned on $N(\pi, T^*_{1:n})$, the exact spike times in $T^*_{1:n}$ are distributed independently and uniformly within each segment of $\pi$.

This new version of $H_0(\Delta)$ is equivalent to the old version and is computationally more tractable because we do not need to consider partitions with arbitrarily long segments. It will be convenient to refer to this class of partitions as

$$\mathcal{L}(\Delta) = \{\text{partitions } \pi : \Delta \leq L_{\min}(\pi) \leq L_{\max}(\pi) < 2\Delta\}.$$

Fixed partition jitter computes $\alpha^*(\pi)$, a $p$-value for $H_0(\pi)$. The $p$-value for the compound $H_0(\Delta)$ is

$$\alpha^*(\Delta) = \max_{\pi \in \mathcal{L}(\Delta)} \alpha^*(\pi) = \max_{\pi \in \mathcal{L}(\Delta)} \sum_{z \geq z^*} P^{\pi}(z). \tag{33}$$

If we want to perform a hypothesis test for $H_0(\Delta)$ with level of significance $\alpha$ and we observe $\alpha^*(\Delta) \leq \alpha$, then this means we can reject every single process in $H_0(\Delta)$ with significance $\alpha$ or better. Conversely, if we observe $\alpha^*(\Delta) > \alpha$, then there is a process in $H_0(\Delta)$ that we would fail to reject at level $\alpha$. Variable partition jitter tries to compute $\alpha^*(\Delta)$. Sometimes it is only practical to bound $\alpha^*(\Delta)$.

As mentioned in Section 7.4.1 and discussed further in Section 7.8, whenever the statistic $Z$ itself is stochastic we need the following addition to $H_0(\Delta)$:

$H_0(\Delta)$: *(Addendum)* Furthermore, conditioned on $N(\pi, T^*_{1:n})$, the choice of the statistic $Z$ is independent of the exact spike times in $T^*_{1:n}$.

**Proof of Lemma 7.5.1.** Another way to think about $H_0(\pi)$ is to say that the conditional distribution of $T^*_{1:n}$ given $N(\pi, T^*_{1:n})$ is constant over its support. The result then follows from the next lemma by taking $W \equiv 1$. □

**Lemma 7.5.2.** For each doubly infinite sequence of integers, $s = \ldots, s_{m-1}, s_m, \ldots$, define

$$\mathcal{T}(\pi, s) = \{t_{1:n} : N(\pi, t_{1:n}) = s\}$$

to be those finite sequences $t_{1:n}$ that have $s_m$ events in the $m$th segment of the partition $\pi$ for all $m$. Let $W$ be a nonnegative function on finite sequences of real numbers and let $T^*_{1:n}$ be a random variable on the same space with distribution $\mathbb{P}^*$. We say that $\mathbb{P}^*$ satisfies $H_0^W(\pi)$ if for every $s$, the conditional density $p(t_{1:n}|\mathcal{T}(\pi, s))$ of $T^*_{1:n}$ given $N(\pi, T^*_{1:n}) = s$ is

$$p(t_{1:n}|\mathcal{T}(\pi, s)) = cW(t_{1:n})\mathbb{1}\{N(\pi, t_{1:n}) = s\}$$

(relative to some fixed measure) for a normalization constant $c$ depending perhaps on $s$ and $\pi$, but not depending on $t_{1:n}$. If $\mathbb{P}^*$ satisfies $H_0^W(\pi)$, then $\mathbb{P}^*$ satisfies $H_0^W(\tilde{\pi})$ for any refinement $\tilde{\pi}$ of $\pi$.

**Proof.** Let $\tilde{\pi}$ be a refinement of $\pi$. Given any outcome of event counts $\tilde{s}$ for $\tilde{\pi}$, let $s$ be the (deterministic) outcome of event counts for $\pi$, that is,

$$\mathcal{T}(\tilde{\pi}, \tilde{s}) \subseteq \mathcal{T}(\pi, s).$$

We can simply compute

$$\begin{aligned} p(t_{1:n}|\mathcal{T}(\tilde{\pi}, \tilde{s})) &= \frac{p(t_{1:n}|\mathcal{T}(\pi, s))\mathbb{1}\{t_{1:n} \in \mathcal{T}(\tilde{\pi}, \tilde{s})\}}{\mathbb{P}^*\big(\mathcal{T}(\tilde{\pi}, \tilde{s})|\mathcal{T}(\pi, s)\big)} \\ &= \tilde{c}W(t_{1:n})\mathbb{1}\{N(\pi, t_{1:n}) = s\}\mathbb{1}\{N(\tilde{\pi}, t_{1:n}) = \tilde{s}\} = \tilde{c}W(t_{1:n})\mathbb{1}\{N(\tilde{\pi}, t_{1:n}) = \tilde{s}\}, \end{aligned}$$

which is the desired result. □

### 7.5.2 Algorithms

More than the other methods, variable partition jitter relies on the discretization of time. In particular, we need to quantize the partitions in order to have a reasonable chance of exploring them all. So as before, the spike times $\omega^*_{1:n^*}$, the partitions $\pi$ and the jitter parameter $\Delta$ are all assumed to be integer valued, represented in some small appropriate units, like milliseconds. Note that $\mathcal{L}(\Delta)$ becomes

$$\mathcal{L}(\Delta) = \{\text{grid partitions } \pi : \Delta \leq L_{\min}(\pi) \leq L_{\max}(\pi) \leq 2\Delta - 1\}.$$

In both of the previous jitter methods we convolve the random variables $(\Omega_i, U_i, X_i)$ for $i = 1 : n^*$ to create the distribution $P$ of $Z$, namely, $P = X_1 * \cdots * X_{n^*}$, using the notation from Section 7.2.1. In fixed partition jitter, these depend on the partition and we make that explicit here with a superscript $\pi$. In particular, $\Omega^\pi_i$ is the jitter window for the $i$th observed spike using the partition $\pi$. This is the segment of the partition that the $i$th spike falls in as detailed in (31). Given the jitter window, the jitter distribution $U^\pi_i$ is uniform over the window and the statistic $X^\pi_i(\omega) = X(\omega)$ for grid points $\omega \in \Omega^\pi_i$. We can compute the distribution $P^\pi$ of $Z^\pi = \sum_{i=1}^{n^*} X_i^\pi$ by recursive convolution, namely,

$$P^\pi = X_1^\pi * \cdots * X_{n^*}^\pi.$$

Define the function

$$G^*(z) = \max_{\pi \in \mathcal{L}(\Delta)} \sum_{\zeta \geq z} P^\pi(\zeta).$$

Note that $\alpha^*(\Delta) = G^*(z^*)$. An exact algorithm for computing $G^*$ (and thus $\alpha^*$) is described in Section 7.2.1.6. Transforming this problem into the notation there: $n = n^*$, $\Omega_i^{k,\ell} = [k+1\colon \ell]$ is the segment between $k$ and $\ell$, $U_i^{k,\ell}(\omega) = 1/(\ell - k)$ is the discrete uniform distribution on $\Omega_i^{k,\ell}$ and $X_i^{k,\ell}(\omega) = X(\omega)$ is the restriction of our statistic $X$ to $\Omega_i^{k,\ell}$. Note that $\mathcal{L}(\Delta)$ here is the same thing as $\mathcal{L}(\Delta, 2\Delta - 1)$ there.

For certain examples this exact algorithm may require too many computations to be practical. Section 7.2.1.6 also describes a method to compute upper and lower bounds on $G^*$ (and thus $\alpha^*$) that will typically be practical whenever fixed partition jitter is practical. These bounds will give bounds on the p-value $\alpha^*(\Delta)$ and can thus be used for a hypothesis test.

## 7.6 Incorporating Physiological Constraints

Each of the three jitter methods described above *independently* jitters spikes within some window. Neighboring spikes in a new jittered spike train can occupy identical time bins or violate the absolute refractory period constraints. Furthermore, the jittered spike train will typically not show any bursting or relative refractory period or rebound, even if these structures are present in the original spike train.

On the one hand, this is exactly what we wanted to happen. Refractory periods, bursting, rebound – each of these is an example of fine temporal structure and jittered spike trains should not have fine temporal structure. On the other hand, these particular examples of fine temporal structure can often be explained by the internal dynamics of the neuron. They are typically viewed as uninteresting (i.e., already explained) sources of fine temporal structure. We want to ask, Is there anything else?

The jitter methods described above can be modified to incorporate certain types of temporal structure into the jitter distribution. This is accomplished by introducing a weighting function $W_k$ between the $(k-1)$st and the $k$th spike time that creates dependencies in the jittered spike times. We will describe several types of weighting functions in the next few sections.

The ability of jitter methods to easily and efficiently incorporate many physiological dependencies is one of their most appealing features. Except perhaps for variable partition jitter, which becomes much less efficient in the dependent situation, the modified jitter methods described in this section are almost always preferable to the original independent jitter methods. At the very least, it seems reasonable to prevent multiple jittered spikes from occupying the same time bin.

### 7.6.1 Absolute refractory periods

We want to generate jittered spike trains that have no fine temporal structure except for an absolute refractory period: a minimum spacing between consecutive spike times. The Monte Carlo intuition is to uniformly and independently jitter the observed spike times, but then to throw out any spike trains that have a violation of the refractory period.

We can make this notion precise by introducing a binary weighting function $W$ that assigns a 1 to spike trains that do not violate the refractory period and that assigns a 0 to spike trains that do violate the refractory period. Fix the observed spike train $\omega^*_{1:n^*}$. If $P_{\text{jitter}}$ is the probability distribution on spike trains $\omega_{1:n^*}$ created by independently and uniformly jittering the spike times in $\omega^*_{1:n^*}$, then

$$P^W_{\text{jitter}}(\omega_{1:n^*}) = \kappa^{-1} P_{\text{jitter}}(\omega_{1:n^*}) W(\omega_{1:n^*})$$

corresponds to the distribution that results from sampling from $P_{\text{jitter}}$, but then throwing out any spike trains that violate the refractory period. The constant $\kappa^{-1}$ is a normalization constant that makes sure that $P^W_{\text{jitter}}$ is a probability distribution (i.e., sums to 1).

An important observation is that the weighting function $W$ factors into

$$W(\omega_{1:n}) = \prod_{i=2}^{n} W_i(\omega_{i-1}, \omega_i) = \prod_{i=2}^{n} \mathbb{1}\{\omega_i - \omega_{i-1} > \tau\}, \tag{34}$$

where $\tau$ is the duration of the absolute refractory period (in the appropriate grid units). Any weighting function that factors like this is called Markov. Since absolute refractory period can be modeled using a Markov weighting function, we can use the Markov dependent convolution methods described in Section 7.2.2 to create exact jitter methods. Markov dependent convolution requires more computation than independent convolution, but is still highly efficient.

Note that $\tau = 0$ corresponds to simple requirement that jittered spike times cannot occupy the same time bin. Note also that if the observed spike train $\omega^*_{1:n^*}$ contains any violations of the modeled absolute refractory period, then $W(\omega^*_{1:n^*}) = 0$ and $P^W_{\text{jitter}}(\omega^*_{1:n^*}) = 0$. This makes it difficult to interpret the results from the jitter methods. In general, whenever physiological constraints are introduced into the jitter methods, it is important that the observed spike train looks relatively typical given these constraints.

#### 7.6.1.1 Spike centered jitter

Return to the setup in Section 7.3.3.1 and 7.3.3.2, except that now we want to modify the distribution of the jittered spike train $T_{1:n^*}$ by throwing out any violations of the absolute refractory period. Before we computed the distribution $P_Z$ of $Z$ exactly using independent convolution. Now we can use Markov dependent convolution to exactly compute

$$P_Z = X_1 *^{W_2} \cdots *^{W_{n^*}} X_{n^*},$$

where $W_i : \Omega_{i-1} \times \Omega_i \to \{0, 1\}$ is the weighting function defined by $W_i(\omega_{i-1}, \omega_i) = \mathbb{1}\{\omega_i - \omega_{i-1} > \tau\}$ for $i = 2 : n^*$ and $\tau \geq 0$ is the duration of the refractory period.

As before, $\alpha^* = \sum_{z \geq z^*} P_Z(z)$ is a nice summary statistic for evaluating the unusualness of the observed spike train within the population of jittered spike trains. It still cannot be interpreted a p-value for a hypothesis test.

#### 7.6.1.2 Fixed partition jitter

Modifying fixed partition jitter to respect the refractory period is the same as modifying spike centered jitter and we can compute

$$P_Z = X_1 *^{W_2} \cdots *^{W_{n^*}} X_{n^*}.$$

Recall that the only difference between fixed partition jitter and spike centered jitter is how the jitter windows are chosen.

The original fixed partition jitter was associated with a hypothesis test and so is this modification of fixed partition jitter. Using the notation from Section 7.4.1, the null hypothesis becomes

$H_0^W(\pi)$: Conditioned on $N(\pi, T^*_{1:n})$, the exact spike times in $T^*_{1:n}$ are distributed independently and uniformly within each segment of $\pi$ up to the absolute refractory period constraint encoded by $W$.

That is, for each segment $(\pi_{m-1}, \pi_m]$, choose $N_m(\pi, T^*_{1:n})$ spike times independently and uniformly within the segment. If the refractory period constraint is violated, throw out this sample and keep trying again until the sample $T^*_{1:n}$ obeys the refractory period. (Note that the resulting jittered spike times will necessarily be ordered.) The statement of Lemma 7.5.2 contains an equivalent, but more technical definition of $H_0^W(\pi)$.

#### 7.6.1.3 Variable partition jitter

We want to test the null hypothesis

$H_0^W(\Delta)$: There exists a partition $\pi$ with $L_{\min}(\pi) \geq \Delta$ such that conditioned on $N(\pi, T^*_{1:n})$, the exact spike times in $T^*_{1:n}$ are distributed independently and uniformly within each segment of $\pi$ up to the refractory period constraint encoded by $W$.

Lemma 7.5.1 extends to the dependent setting. We use the notation from Sections 7.5.1 and 7.6.1.2.

**Lemma 7.6.1.** If $\mathbb{P}^*$ satisfies $H_0^W(\pi)$, then $\mathbb{P}^*$ satisfies $H_0^W(\tilde{\pi})$ for any refinement $\tilde{\pi}$ of $\pi$.

This is stated and proved more precisely and in more generality in Lemma 7.5.2. As in the independent setting, we can use these refinement lemmas to restate $H_0^W(\Delta)$ as

$H_0^W(\Delta)$: There exists a partition $\pi$ with $\Delta \leq L_{\min}(\pi) \leq L_{\max}(\pi) < 2\Delta$ such that conditioned on $N(\pi, T^*_{1:n})$, the exact spike times in $T^*_{1:n}$ are distributed independently and uniformly within each segment of $\pi$ up to the refractory period constraint encoded by $W$.

The algorithms described in Sections 7.2.2.8–7.2.2.9 show how to compute (or bound) the function

$$G^*(z) = \max_{\pi \in \mathcal{L}(\Delta)} \sum_{\zeta \geq z} P^{\pi}(\zeta),$$

where $P^{\pi}$ is the distribution returned by fixed partition jitter using the partition $\pi$ and the weighting function $W$. In particular, we discretize time and use $r = \Delta$, $R = 2\Delta - 1$, $n = n^*$, $\Omega_i^{k,\ell} = [k+1 : \ell]$, $U_i^{k,\ell}(\omega) = 1/(\ell - k)$ on $\Omega_i^{k,\ell}$, $X_i^{k,\ell}(\omega) = X(\omega)$ on $\Omega_i^{k,\ell}$ and $W_i^{k,\ell,k',\ell'}(\omega_{i-1}, \omega_i) = \mathbb{1}\{\omega_i - \omega_{i-1} > \tau\}$ on $\Omega_{i-1}^{k,\ell} \times \Omega_i^{k',\ell'}$.

Once we have computed $G^*$, we can immediately get the p-value for a hypothesis test of $H_0^W(\Delta)$, namely, $\alpha^*(\Delta) = G^*(z^*)$. In certain examples, exact variable partition jitter with dependencies may not be computationally feasible. Unlike the independent situation, computing bounds on $G^*$ may not be practical either. Even in the best case, the number of operations grows something like $n^2\Delta^7$, so $\Delta$ must be pretty small relative to the discretization for these methods to ever be practical.

## 7.6.2 Relative refractory periods and rebound

We modeled absolute refractory period as a binary valued weighting function $W$ that factored into pairwise terms between neighboring spikes (34). The computational methods make use of the fact that $W$ factors, but they do not make use of the binary assumption. By allowing $W$ to take any (nonnegative) value, we can introduce softer structure into the jitter distribution. This is convenient for structures like relative refractory period and rebound (bursting).

Consider the following Markov weighting function on $n$ spikes $\omega_{1:n}$: $W = W_2 \cdots W_n$, where $W_i(\omega_{i-1}, \omega_i) = f(\omega_i - \omega_{i-1})$ and

$$f(t) = \mathbb{1}\{t > \tau\}\big(\sigma(t) + \xi(t)\big)$$

for an increasing function $\sigma$ that is 0 at 0 and 1 eventually and a unimodal, nonnegative function $\xi$ that is 0 at 0 and 0 eventually. $\mathbb{1}\{t > \tau\}$ models an absolute refractory period, $\sigma$ models the relative refractory period and $\xi$ models the rebound.

Analogous to absolute refractory period, we can modify the independent jitter distribution by

$$P_{\text{jitter}}^{W}(\omega_{1:n^*}) = \kappa^{-1} P_{\text{jitter}}(\omega_{1:n^*}) W(\omega_{1:n^*}).$$

As before, spike trains $\omega_{1:n^*}$ for which $W(\omega_{1:n^*}) = 0$ are not allowed. For all other spike trains, $W$ indicates the relative probabilities of the different spike time configurations. Higher $W$ is more likely than lower $W$. The exact magnitude of $W$ is meaningless, because changes are just absorbed into the normalization constant $\kappa$. See Section 7.2.2.4 for a more formal discussion.

The three different jitter methods will work for any Markov weighting function, including the one described here. Sections 7.6.1.1–7.6.1.3 detail each method. We simply change the specifics of the pairwise weighting functions $W_i(\omega_{i-1}, \omega_i)$. Lemma 7.5.2 shows that the refinement lemma underlying independent variable partition jitter extends to the dependent setting.

Incorporating soft constraints in this manner introduces assumptions about the relative probabilities of specific spiking events. The validity of these assumptions can be difficult to verify, much more so than in the case of an absolute refractory period. In the next section we present a more agnostic way to incorporate certain types of fine temporal structure, like refractory periods and bursting, into the jitter distribution. These model-free methods are more in the spirit of the original jitter intuition and we prefer them in most cases over the explicit modeling methods described here.

### 7.6.3 Model free constraints

Fix the observed spike train $\omega^*_{1:n^*}$ and define the Markov weighting function $W = W_2 \cdots W_{n^*}$ by

$$W_i(\omega_{i-1}, \omega_i) = \begin{cases} \mathbb{1}\{\omega_i - \omega_{i-1} = \omega^*_i - \omega^*_{i-1}\} & \text{if } \omega^*_i - \omega^*_{i-1} \leq q, \\ \mathbb{1}\{\omega_i - \omega_{i-1} > q\} & \text{if } \omega^*_i - \omega^*_{i-1} > q, \end{cases}$$

for $i = 2 : n^*$. This weighting function is based on the observed train and a parameter $q \geq 0$. Any interspike intervals of $q$ or less in the observed spike train are preserved exactly in each jittered spike train. Also, no interspike intervals of $q$ or less are created in any jittered spike trains where they did not exist already in the observed spike train.

One way to think about how this weighting function behaves is to partition the observed spike train into maximal $q$-patterns. A $q$-pattern is any sequence of spike times $\omega_{i:j}$ with either exactly one spike, i.e., $j = i$, or with $\omega_{k+1} - \omega_k \leq q$ for all $k = i : j - 1$. Note that the first and last spike times in a $q$-pattern can be separated by much more than $q$. In the context of a spike train, a maximal $q$-pattern is a one that is not contained in any other $q$-pattern, that is, a $q$-pattern that consists of all the spikes between its first and last spike and whose first spike and last spike are separated by more than $q$ from the spike that precedes and follows it, respectively.

When used with one of the jitter methods, the above weighting function uniformly jitters each maximal $q$-pattern in $\omega^*_{1:n^*}$ so that all of the spikes remain in their respective jitter windows and so that no other $q$-patterns are created. Figure 7.7 contains some examples.

For spike centered jitter, since the jitter window for each spike is centered around itself, each $q$-pattern is jittered uniformly in a window centered around itself, up to the constraint that it cannot come within $q$ of the $q$-pattern immediately preceding or following it (otherwise a new $q$-pattern would be created). Fixed partition jitter is more problematic, however. A $q$-pattern that has one spike at the beginning of its segment in the partition and that has another spike at the end of its respective segment cannot move. Since the probability of this increases as the $q$-pattern gets longer, long $q$-patterns often will not be jittered at all. We illustrate how to remedy this in the next section.

#### 7.6.3.1 Jittering patterns

Fix a partition $\pi$ and a pattern parameter $q$. Given the observed spike times $\omega^*_{1:n^*}$, partition them into maximal $q$-patterns, say $\omega^*_{i_1:i_2-1}, \ldots, \omega^*_{i_{m^*}:i_{m^*+1}-1}$, where $1 = i_1 < \cdots < i_{m^*+1} = n^* + 1$ and $m^*$ is the number of maximal $q$-patterns.

Consider the following method of creating a jittered spike train with the identical sequence of maximal $q$-patterns: Jitter the initial spike times of each $q$-pattern, $\omega^*_{i_1}, \ldots, \omega^*_{i_{m^*}}$, uniformly in their respective segments of $\pi$ and jitter the remaining spikes in each maximal $q$-pattern by the same amount so that the exact pattern is preserved. Throw out this jittered spike train, if necessary, and repeat until the jittered spike train has the same sequence of maximal $q$-patterns. (The only thing that can go wrong is if two patterns get within $q$ of each other, or swap positions.)

This sampling scheme is like the scheme underlying fixed partition jitter, except that now entire $q$-patterns are jittered. We can still use the same computational routines to compute the exact distribution of $Z$ under this new sampling scheme. The key is that $Z$ can be written as an additive function of *deterministic* functions of *pattern positions*. Let $T_{1:n^*}$ denote the jittered spike times. Note that $T_{i_1:i_2-1}, \dots, T_{i_{m^*}:i_{m^*+1}-1}$ are still the maximal $q$-patterns. We can express

$$Z = Z(T_{1:n^*}) = \sum_{i=1}^{n^*} X(T_i) = \sum_{k=1}^{m^*} \sum_{i=i_k}^{i_{k+1}-1} X(T_i) = \sum_{k=1}^{m^*} \underbrace{\sum_{i=i_k}^{i_{k+1}-1} X(T_{i_k} + \omega_i^* - \omega_{i_k}^*)}_{Y_k(T_{i_k})}$$

$$= \sum_{k=1}^{m^*} Y_k(T_{i_k}).$$

The jittered pattern positions are the times of the first spike in each pattern $T_{i_1}, \dots, T_{i_{m^*}}$. $T_{i_k}$ takes values in $\Omega_{i_k}$, which is the segment of $\pi$ that contains $\omega_{i_k}^*$ as in (31). The $T_{i_k}$ are jittered uniformly up to the constraints that they do not swap positions and that no new $q$-patterns are created. We can encode this into a (binary) Markov weighting function $W = W_2 \cdots W_{m^*}$, where

$$W_k(\omega_{k-1}, \omega_k) = \mathbb{1}\{\omega_k - \omega_{k-1} > \underbrace{\omega_{i_k-1}^* - \omega_{i_k}^*}_{\substack{\text{length of } (k-1)\text{st} \\ \text{maximal } q\text{-pattern}}} + q\}.$$

To compute the distribution of $Z$, we can use Markov dependent convolution

$$Z = Y_1 *^{W_2} \cdots *^{W_{m^*}} Y_{m^*},$$

where $Y_k$ is the function defined above on $\Omega_{i_k}$ and where we use the discrete uniform distribution $U_{i_k}$ over $\Omega_{i_k}$ as the $k$th reference probability.

As in fixed partition jitter, we can use $\alpha^* = \sum_{z \geq z^*} P_Z(z)$ to test a null hypothesis:

$H_0^q(\pi)$: Conditioned on the sequence of maximal $q$-patterns in $T_{1:n}^*$ and conditioned on which segment of the partition $\pi$ each pattern starts in, the exact starting positions of each maximal $q$-pattern are distributed independently and uniformly within their respective segment of $\pi$.

By independently and uniformly, we mean that the joint density is constant over the support. Note that they are not truly independent because the sequence of maximal $q$-patterns must be preserved, but the dependencies exist only in the support. The sequence of maximal $q$-patterns includes a description of each of the $m^*$ $q$-patterns, that is, spike times relative to the starting position (initial spike time) of the pattern, and also the order that the patterns appear in the spike train.

Since we can use Markov dependent convolution to address this null hypothesis, we can also use the variable partition jitter methods to address the compound null hypothesis that $\mathbb{P}^*$ satisfies $H_0^q(\pi)$ for some $\pi$ with $L_{\min}(\pi) \geq \Delta$. Note that the refinement lemmas easily extend to this situation, so we can restrict ourselves to $\pi \in \mathcal{L}(\Delta)$. This is variable partition jitter with dependencies, so the specifics of the problem will dictate whether or not it is practical to compute or even to bound $\alpha^*(\Delta)$.

One interesting property of the pattern jitter methods is that the statistic $Z$ only needs to be additive over the maximal $q$-patterns, not necessarily over individual spikes. So we could modify the functions $Y_k$ to be any function of the pattern start time and the specific pattern. The $Y_k$'s need not be reducible to additive functions over the spike times within the pattern.

### 7.6.4 Edge effects

In practice, the observed spike train $\omega^*_{1:n^*}$ is actually embedded into a larger spike train. This brings up the issue of edge effects, that is, what happens if either $\omega^*_1$ or $\omega^*_{n^*}$ are jittered into positions that violate physiological constraints based on unobserved spikes.

An easy way to remedy this is to hold the endpoints $\omega^*_1$ and $\omega^*_{n^*}$ fixed, that is $U_1(\omega_1) = \mathbb{1}\{\omega_1 = \omega^*_1\}$ and $U_{n^*}(\omega_{n^*}) = \mathbb{1}\{\omega_{n^*} = \omega^*_{n^*}\}$. If the weighting function that we are using prevents spike swapping (and except for the independent jitter methods, it usually will), then no spikes will be jittered into positions that violate any physiological constraints (that we can model with Markov dependencies) with respect to unobserved spikes.

If we are using the jitter the methods to test a null hypothesis, then we include in the null hypothesis that we are also conditioning on the exact first and last spike times. For example, $H_0^W(\pi)$ from Section 7.6.1.2 becomes

$H_0^W(\pi)$: Conditioned on $N(\pi, T^*_{1:n})$, $T^*_1$ and $T^*_n$, the exact spike times in $T^*_{2:n-1}$ are distributed independently and uniformly within each segment of $\pi$ up to the absolute refractory period constraint encoded by $W$.

### 7.6.5 Higher order dependencies

Certain types of physiological phenomenon cannot be well modeled by Markov dependencies. For example, we might want to model a rebound that depends on the number of spikes in the last 50 ms. The jitter methods can be modified somewhat to handle these situations. A recursive convolution algorithm for general dependencies is described in Section 7.2.3.

Higher order dependencies dramatically increase the computational demands of convolution, so the utility of these methods is limited and we do not discuss them further here. Section 7.8.1 does contains a slightly more detailed example of using the general convolution recursion in the context of jitter. That example can be easily modified to fit most situations.

## 7.7 Spike Train Sampling

In certain situations we may want to actually sample from the underlying distribution on jittered spike trains. For example, if we are interested in a statistic $Z$ that cannot be written as

$$Z(T_{1:n}) = \sum_{i=1}^{n} X_k(T_k)$$

for some known functions $X_k$, where $T_{1:n}$ is a jittered spike train, then the exact jitter methods cannot be used to compute the distribution of $Z$ and we need to use a Monte Carlo approach. Sampling is also desirable when we just want to visualize a particular jitter distribution. For example, the jitter methods in Section 7.6, especially Sections 7.6.2–7.6.3, create nonstationary dependencies in the jitter distribution that can be difficult to form intuitions about without being able to sample.

Sampling in the independent jitter methods (spike centered jitter or fixed partition jitter) is easy and efficient, following exactly from the sampling intuition. We just independently and uniformly select spike times from the appropriate windows. Sampling in the Markov dependent jitter setting is also efficient, but we cannot use the sampling intuition. For example, in Section 7.6.1.1 the sampling intuition was to sample independently and uniformly in windows centered around the observed spike times and then to throw out any samples that violated the absolute refractory period. This method of sampling, while correct, is too inefficient to be practical. We will now describe a more practical sampling method.

Given an observed spike train $\omega^*_{1:n^*}$, each of the jitter methods first creates an independent jitter distribution $P_{\text{jitter}}$ over the appropriate jitter windows $\Omega_{1:n^*}$. In all of the cases that we have considered $P_{\text{jitter}}$ is constant because it is a product of uniform distributions $U_i$ over the $\Omega_i$, that is, $P_{\text{jitter}} = \prod_{i=1}^{n^*} U_i$. The convolution algorithms for the exact jitter methods do not make use of the fact that the $U_i$ are uniform and the sampling methods described here will not either. $P_{\text{jitter}}$ is easy to sample from.

We are interested in sampling from the dependent situation where we modify $P_{\text{jitter}}$ by

$$P^W_{\text{jitter}}(\omega_{1:n^*}) = \kappa^{-1} P_{\text{jitter}}(\omega_{1:n^*}) W(\omega_{1:n^*}).$$

The nonnegative function $W$ models dependencies in the jitter spikes. When $W$ is Markov, that is,

$$W(\omega_{1:n^*}) = \prod_{i=2}^{n^*} W_i(\omega_{i-1}, \omega_i),$$

we can still sample efficiently from $P^W_{\text{jitter}}$. The algorithms for sampling are described in Section 7.2.2.6. The main step algorithm transforms $P^W_{\text{jitter}}$ into a Markov chain representation: an initial distribution for the first jittered spike and then a sequence of transition probabilities for the remaining spikes. Once this new representation has been computed, we can quickly sample repeatedly from it.

Figure 7.7 illustrates several different jitter distributions described in the text using samples created in this manner. Besides visualization, sampling is good alternative when the exact methods do not work. This can happen when the problem is too large or when the statistic $Z$ is not additive or when $Z$ takes values in a strange alphabet that causes the convolution algorithms to scale badly.

Let $Z$ be any function of a finite sequence of spike times and let $P^W_{\text{jitter}}$ be a given jitter distribution that we can sample from. Let $T^1_{1:n^*}, \ldots, T^R_{1:n^*}$ be $R$ independent samples from $P^W_{\text{jitter}}$ and let $Z^r = Z(T^r_{1:n^*})$. One way to evaluate the unusualness of the observed spike train $\omega_{1:n^*}$ among the collection of jittered spike trains is to see how unusual $z^* = Z(\omega_{1:n^*})$ is in the empirical distribution of the $Z^r$. For example, we could use

$$\hat{\alpha}^* = \frac{1 + \{\#\ Z^r \geq z^*\}}{1+R} = \frac{1 + \sum_{r=1}^R \mathbb{1}\{Z^r \geq z^*\}}{1+R}.$$

One nice thing about $\hat{\alpha}^*$ is that it can be interpreted as a p-value in any situation that $\alpha^*$ could be.

Each of the null hypotheses for fixed partition jitter that we have considered in this chapter is of the general form: $\omega^*_{1:n^*}$ is a sample from $\mathbb{P}^*$ and the conditional distribution of $\mathbb{P}^*$ given $N = N(\omega^*_{1:n^*})$ is $P^W_{\text{jitter}}$, where $N$ is some property of the observed spike train, like spike counts in each segment of a fixed partition $\pi$, but maybe other things as well, and where $P^W_{\text{jitter}}$ is some specified distribution. Let $T^1_{1:n^*}, \ldots, T^R_{1:n^*}$ be independent samples from $P^W_{\text{jitter}}$ (and also independent from $\omega^*_{1:n^*}$ given $N(\omega^*_{1:n^*})$). Then under the null hypothesis, $\omega_{1:n^*}, T^1_{1:n^*}, \ldots, T^R_{1:n^*}$ are exchangeable. This follows from Lemma 7.7.1 below.

Since $\omega_{1:n^*}, T^1_{1:n^*}, \ldots, T^R_{1:n^*}$ are exchangeable, $z^*, Z^1, \ldots, Z^R$ are exchangeable and Lemma 7.7.2 below shows that

$$\text{Prob}\{\hat{\alpha}^* \leq \alpha\} \leq \alpha$$

for $\alpha \in [0,1]$, so $\hat{\alpha}^*$ can be interpreted as (an upper bound of) a p-value for the null hypothesis.

**Lemma 7.7.1.** Let $Y$ have distribution $P$ (over a standard Borel space) and let $N(Y)$ be a function of $Y$. For each possible outcome $\eta \in \mathcal{N}$ of $N(Y)$ define the (regular) conditional distribution $P_\eta = \text{Prob}(Y \in \cdot | N(Y) = \eta)$. For each $\eta$, choose $Y^\eta_1, \ldots, Y^\eta_n$ i.i.d. from $P_\eta$ and independently from $Y$. Then the random variables $Y, Y^{N(Y)}_1, \ldots, Y^{N(Y)}_n$ are exchangeable.

**Proof.** For convenience, define $Y_0 = Y$ and $Y_k = Y^{N(Y)}_k$ for $k = 1:n$, so that we want to prove that $Y_{0:n}$ has the same distribution as $\pi \circ Y_{0:n}$, where $\pi$ is an arbitrary permutation of the elements, say $\pi \circ (Y_0, Y_1, \ldots, Y_n) = (Y_{k_0}, \ldots, Y_{k_n})$.

Let $P_N$ be the distribution of $N(Y)$. We have

$$\text{Prob}(Y_{0:n} \in A) = E[\text{Prob}(Y_{0:n} \in A)|N(Y)] = \int_{\mathcal{N}} \text{Prob}(Y_{0:n} \in A|N(Y) = \eta) P_N(d\eta)$$

$$= \int_{\mathcal{N}} \int_A P(dy_0|N(Y) = \eta) \prod_{i=1}^n P_\eta(dy_i) P_N(d\eta) = \int_{\mathcal{N}} \int_A \prod_{i=0}^n P_\eta(dy_i) P_N(d\eta),$$

so

$$\text{Prob}(\pi \circ Y_{0:n} \in A) = \text{Prob}(Y_{0:n} \in \pi^{-1}[A]) = \int_{\mathcal{N}} \int_{\pi^{-1}[A]} \prod_{i=0}^{n} P_\eta(dy_i) P_N(d\eta)$$

$$= \int_{\mathcal{N}} \int_{A} \prod_{i=0}^{n} P_\eta(dy_{k_i}) P_N(d\eta) = \int_{\mathcal{N}} \int_{A} \prod_{i=0}^{n} P_\eta(dy_i) P_N(d\eta) = \text{Prob}(Y_{0:n} \in A).$$

Finally, to connect this result to how it is used in the text, take $P = \mathbb{P}^*$, let $\omega^*_{1:n^*}$ be a realization of $Y$ so that $\eta = N(\omega^*_{1:n^*})$ and $P_\eta = P^W_{\text{jitter}}$, and let $Y_i = T^i_{1:n^*}$ for $i = 1, \ldots, n = R$. □

**Lemma 7.7.2.** If $Z_1, \ldots, Z_n$ are exchangeable random variables, then

$$\text{Prob}\Big\{\textstyle\sum_{i=1}^{n} \mathbb{1}\{Z_1 \leq Z_i\} \leq k\Big\} \leq \frac{k}{n}$$

for each $k = 0\!:\!n$.

**Proof.** Order the $Z_i$'s, say $Z_{i_1} \geq \cdots \geq Z_{i_n}$, deciding ties randomly without preference and independently of $Z_{1:n}$. Let $K$ be the position of $Z_1$ in the ordered sequence, that is, $i_K = 1$. The symmetry that comes from exchangeability implies that

$$\text{Prob}\{K = j\} = \frac{1}{n}$$

for each $j = 1\!:\!n$. Note that because of possible ties $\sum_{i=1}^{n} \mathbb{1}\{Z_1 \leq Z_i\} \geq K$, so for each $k = 0\!:\!n$,

$$\text{Prob}\Big\{\textstyle\sum_{i=1}^{n} \mathbb{1}\{Z_1 \leq Z_i\} \leq k\Big\} \leq \text{Prob}\{K \leq k\} = \sum_{j=1}^{k} \text{Prob}\{K = j\} = \frac{k}{n}.$$

To connect this result to how it is used in the text, take $Z_1 = z^*$ and $Z_{2:n} = Z^{1:R}$, noting that $n = R + 1$ and that $\mathbb{1}\{z^* \leq z^*\} = 1$, so that

$$\text{Prob}\{\hat{\alpha}^* \leq \alpha\} = \text{Prob}\Big\{1 + \textstyle\sum_{r=1}^{R} \mathbb{1}\{z^* \leq Z^r\} \leq \lfloor \alpha(R+1) \rfloor\Big\} \leq \frac{\lfloor \alpha(R+1) \rfloor}{R+1} \leq \alpha,$$

where the first inequality comes from this Lemma. □

## 7.8 Synchrony

We are often interested in the situation where $Z$ is some measure of synchronous firing between the jittered spike train and another spike train. For example, if $s^*_{1:m^*}$ is another observed spike train, then we could choose the statistic $Z = Z(t_{1:n}) = \sum_{i=1}^{n} X(t_i)$, where

$$X(t) = \mathbb{1}\left\{\min_{1 \leq j \leq m^*} |t - s^*_j| \leq \epsilon\right\}$$

for some (presumably small) $\epsilon \geq 0$. On the one hand, $Z$ is fixed because it is the same for all jittered spike trains $t_{1:n}$. But on the other hand, $Z$ is random because it depends on another observed spike train $s^*_{1:m^*}$.

Recall from Sections 7.4.1 and 7.5.1 that when $Z$ is random in this way, if we want to interpret $\alpha^*$ as a p-value for some null hypothesis $H_0$, then $H_0$ must include the additional hypothesis that $Z$ is conditionally independent of the observed spike train. Exactly what we are conditioning on is specified in the particular $H_0$. Usually it will involve the spike counts in segments of a fixed partition $\pi$. A rejection of the null hypothesis when $Z$ is random might happen because this conditional independence is not satisfied. Because of this, jitter methods can be used to partially assess the time resolution of certain dependencies.

We will describe two different examples to illustrate this. Both involve using jitter methods in conjunction with a synchrony statistic $Z$ like the one above. Typically $\epsilon$ will need to be significantly smaller than $\Delta$ for $Z$ to be a powerful statistic. The observed spike train is $t^*_{1:n^*}$. This is the spike train that will be jittered. $Z$ is based on another spike train $s^*_{1:m^*}$ that is not jittered.

The first example is when $s^*_{1:m^*}$ is recorded from the same neuron as $t^*_{1:n^*}$, but on a different trial, maybe minutes or hours later. In this situation it seems reasonable to assume that $s^*_{1:m^*}$ and $t^*_{1:n^*}$ are independent. (One spike train might convey information, in the colloquial sense, about the other to an experimenter, but only because it provides information about the unknown, underlying distribution. From the point of view of this underlying distribution, which is what matters here, independence is probably a good approximation.) A rejection of the null suggests that the spike trains have a time resolution finer than $\Delta$. The neuron's firing pattern is more repeatable than what would be expected if spike timing was more or less unstructured at time scales smaller than $\Delta$. In this example, the main reason for using a synchrony statistic is that it might be a particularly powerful statistic for rejecting the null.

The second example is when $t^*_{1:n^*}$ and $s^*_{1:m^*}$ are simultaneously recorded from two different neurons. The two spike trains are not likely to be independent. External factors, like the particular stimulus and behavior are certainly the same, and it would not be surprising if many internal factors, like attentional states and metabolic rates were also similar. Any of these might create dependencies between the two spike trains, especially coarse dependencies in the firing rates. What is unclear is the degree to which these dependencies extend to the precise timing of spikes. A rejection of the null suggests that these dependencies affect the resolution of spike timing at time scales smaller than $\Delta$, even after conditioning on the dependencies in the coarser firing rates. Strictly speaking, the null might also be rejected if the two spike trains were completely independent but $t^*_{1:n^*}$ was precisely timed, however, it seems unlikely that a statistic based on an independent $s^*_{1:n^*}$ would have much power for rejecting a hypothesis about the distribution of $t^*_{1:n^*}$. In this example, the main reason for using a synchrony statistic is that we explicitly want to test the hypothesis that the spike train and the statistic are conditionally independent.

These examples illustrate two conceptually different uses of a random statistic in conjunction with jitter methods. The first asserts the conditional independence between the spike train and the statistic, using the statistic simply because it might be powerful. The second does not assume the conditional independence, but tests it as part of the null hypothesis.

### 7.8.1 Simultaneous jitter

Jittering one spike train in conjunction with a synchrony statistic is not ideal for two reasons. The first is simply the asymmetry that arises when one spike train is held fixed while the other is jittered. Why not jitter both simultaneously? The second is that the null hypothesis is difficult to interpret. It may not be clear exactly what it means for $t^*_{1:n^*}$ and $Z$ (based on $s^*_{1:m^*}$) to be conditionally independent given, for example, $N(\pi, t^*_{1:n^*})$. Perhaps a more understandable situation would be that $t^*_{1:n^*}$ and $s^*_{1:m^*}$ were conditionally independent given $N(\pi, t^*_{1:n^*})$ and $N(\pi, s^*_{1:m^*})$. In this case, the null hypothesis would specify two jitter distributions, one for $t^*_{1:n^*}$ and one for $s^*_{1:m^*}$, and we would need to simultaneously jitter the two spike trains to compute p-values.

Monte Carlo methods are the most straightforward way to address simultaneous jitter. Each spike train is jittered independently using the sampling methods described in Section 7.7. This is repeated multiple times and the empirical distribution of some statistic $Z = Z(t_{1:n}, s_{1:m})$ is used to evaluate the temporal precision of the pair $t^*_{1:n^*}$ and $s^*_{1:m^*}$. The discussion in 7.7 extends almost verbatim to this situation. In particular, if the jitter distributions come from a fixed partition, then the tail probability $\hat{\alpha}^*$ can often be interpreted as a p-value in a hypothesis test, albeit a hypothesis test that might depend strongly on the particular partition. $Z$ need not be additive and it can be chosen so that it is symmetric in $t^*_{1:n^*}$ and $s^*_{1:m^*}$.

In certain cases it may be computationally feasible to do exact simultaneous jitter, that is, to compute the limiting value of the empirical distribution of the $Z$'s (that is, the distribution of $Z$) in the above Monte Carlo experiment. This is based on the general convolution framework described in Section 7.2.3.

Let $P^W_{\text{jitter}}$ and $Q^{W'}_{\text{jitter}}$ be the jitter distributions corresponding to $t^*_{1:n^*}$ and $s^*_{1:m^*}$, respectively, and let $T_i$ and $S_j$ be the random variables corresponding to the jittered versions of $t^*_i$ and $s^*_j$, that is, the $i$th and $j$th coordinate random variables of $P^W_{\text{jitter}}$ and $Q^{W'}_{\text{jitter}}$, respectively. If the constraints $W$ and $W'$ are Markov, then the joint distribution $P^W_{\text{jitter}} \times Q^{W'}_{\text{jitter}}$ on $(T_{1:n^*}, S_{1:m^*})$, respects the dependency graph shown in Figure 7.8, which has edges between the vertices corresponding to $T_i$ and $T_{i+1}$ and also between $S_j$ and $S_{j+1}$.

The joint density of $(T_{1:n^*}, S_{1:m^*}, Z)$ for a general statistic $Z = Z(T_{1:n^*}, S_{1:m^*})$ is

$$P^W_{\text{jitter}}(t_{1:n^*})Q^{W'}_{\text{jitter}}(s_{1:m^*})\mathbb{1}\{z = Z(t_{1:n^*}, s_{1:m^*})\},$$

which need not lend itself to efficient computation. In many cases, however, $Z$ can be decomposed into

$$Z(T_{1:n^*}, S_{1:m^*}) = \sum_{k=1}^{K} Y_k(T_{A_k}, S_{B_k}),$$

where $A_k \subseteq \{1, \ldots, n^*\}$ and $B_k \subseteq \{1, \ldots, m^*\}$. In this case the joint density of $(T_{1:n^*}, S_{1:m^*}, Y_{1:M})$ factors into

$$P^W_{\text{jitter}}(t_{1:n^*})Q^{W'}_{\text{jitter}}(s_{1:m^*})\prod_{k=1}^{K}\mathbb{1}\{y_k = Y_k(t_{A_k}, s_{B_k})\}.$$

This respects the dependency graph for $P^{W}_{\text{jitter}} \times Q^{W'}_{\text{jitter}}$ with added vertices corresponding to the $Y_k$'s and added edges between all vertices corresponding to elements of $(Y_k, T_{A_k}, S_{B_k})$.

Computing the distribution of $Z = \sum_k Y_k$ can be carried out using the convolution algorithm described in Section 7.2.3. Depending on the specifics of the graph, this may or may not be computationally feasible. For example, if $Z$ is the synchrony statistic then we can write

$$Z(T_{1:n^*}, S_{1:m^*}) = \sum_{i=1}^{n^*} \mathbb{1}\Big\{\min_{j=1:m^*} |T_i - S_j| \le \epsilon\Big\} = \sum_{k=1}^{n^*} \underbrace{\mathbb{1}\Big\{\min_{j \in B_k} |T_k - S_j| \le \epsilon\Big\}}_{Y_k(T_{A_k}, S_{B_k})},$$

for $A_k = \{k\}$ and $B_k = \{j : d(\Omega_k, \Lambda_j) \le \epsilon\}$, where $\Omega_k$ and $\Lambda_j$ are the ranges of $T_k$ and $S_j$, respectively, and $d(\Omega_k, \Lambda_j) = \inf_{\omega \in \Omega_k, \lambda \in \Lambda_j} |\omega - \lambda|$, so that $B_k$ is the set (of indices) of spikes in $s^*_{1:m^*}$ that could possibly be jittered to within $\epsilon$ of a jittered version of $t^*_k$. An example dependency graph is shown in Figure 7.9. If the sets $B_k$ are small, which will often happen if $\Delta$ and $\epsilon$ are small relative to interspike intervals, then this graph might be amenable to efficient computation. Note that the graph (and thus the computation) depends heavily on the specific observations $t^*_{1:n^*}$ and $s^*_{1:m^*}$ and their relationship to the specific jitter parameters.

## 7.8.2 Jitter as a measure of synchrony

Much of the focus in this chapter has been on interpreting jitter methods in the context of a hypothesis test. Another promising application of jitter methods is simply as another measurement that can be applied to spike trains. This measurement can be used like any other, say for comparing different experimental conditions. Measurements based on jitter methods will typically be controlled (somewhat) for firing rate and maybe even other temporal dynamics like refractory period.

Consider one of the spike centered jitter methods, perhaps the exact model free jitter in Section 7.6.3, with an additive statistic $Z$ appropriate to the particular application. While $\alpha^*$ does not correspond to the p-value for any hypothesis test, it does return a number that tends to be low (near 0) if the observed spike train has "unusual" fine temporal structure and tends to be higher (above 0.5) if it does not. Across different experimental trials, $\alpha^*$ might vary significantly. Of course the interpretation of this will depend on $Z$ and the jitter parameters.

When $Z$ is a synchrony statistic, then $\alpha^*$ provides a measure of how much synchrony there is between $t^*_{1:n^*}$ and $s^*_{1:m^*}$. This measure is controlled for firing rates and other short firing patterns and might be a useful alternative to more traditional correlation measures. Transformations like $1-\alpha^*$ or $-\log \alpha^*$ might put $\alpha^*$ into more intuitive or more discriminative units. Other quantities based on $P_Z$ (which is what the exact jitter methods compute), like the standardized score $(z^* - \mu_Z)/\sigma_Z$, could also be used, where $\mu_Z$ and $\sigma_Z$ are the mean and standard deviation of $P_Z$, respectively. Running the algorithm twice, first jittering $t^*_{1:n^*}$ and then $s^*_{1:m^*}$, and then combining the results appropriately would be a simple way to symmetrize the synchrony measure. Or, if possible, exact simultaneous jitter could be used.

## 7.9 Experiments

In this section we describe two simple experiments with actual neural data. The point is to illustrate the techniques and not to draw any scientific conclusions from the data. The data is courtesy of Ben Philip, Wilson Truccolo and Carlos Vargas in John Donoghue's lab at Brown University. It consists of 3 simultaneously recorded neurons from an awake, behaving monkey (macaca mulata). The monkey is using a joystick to perform a delayed-response radial direction task, i.e., the monkey must move a cursor radially outward from a center location to one of several possible targets that has been previously visually specified. The exact nature of the task is irrelevant here, since we are only using the data for illustrative purposes.

### 7.9.1 Repeating patterns

In this section we give an example of how incorporating physiological constraints helps to prevent some of the artifacts that can be introduced by simpler versions of the jitter method. We will focus on a single neuron, labeled `11a`, recorded from area 5d of parietal cortex over a period of about 1 hour with an average firing rate of 19.7 Hz. The interspike interval (ISI) histogram is shown in Figure 7.10. The shortest ISI is 1.6 ms (absolute refractory period). There is evidence of bursting and other structure in the histogram for ISIs under about 30 ms. The distribution of ISIs longer than about 30 ms looks qualitatively like the exponential decay one would expect from a homogeneous Poisson process.

One way to look at fine temporal structure is to look at the frequency of repeating patterns of firing. See, for example, [4, 12, 10]. Consider pairs of interspike intervals $(\delta_1, \delta_2)$ defined by three consecutive spikes. (Note that this is much less general than the firing patterns considered in the above references, where the pattern need not come from consecutive spikes and may even come from multiple neurons.) For each pair $(\delta_1, \delta_2)$, let $N(\delta_1, \delta_2)$ be the frequency with which that pair occurred in the data. One possible summary statistic is

$$\max_{\delta_1, \delta_2} N(\delta_1, \delta_2),$$

the maximum frequency that any pattern repeats. If the observed spike train has fine temporal structure, then we might expect certain patterns to repeat "more frequently than chance".

Of course, "more frequently than chance" is hard to quantify. In jitter methods, "chance" is defined by jittering spikes to create a large collection of artificial spike trains with similar coarse temporal structure, but no fine temporal structure (because the fine temporal structure was destroyed by jittering). We can compute the distribution of the maximum frequency of repeating patterns over this collection and use this distribution to define exactly what we mean by "chance".

The first thing that we tried was the original fixed partition jitter. We used a time resolution of 1 ms and we partitioned time into 20 ms windows. We created 999 artificial spike trains by jittering each of the observed spikes independently and uniformly in its respective window (and then sorting in case of spike swapping). Note that the statistic of interest, namely, the maximum frequency of repeating pairs of ISIs, is not additive and exact jitter methods are not applicable. The original spike train had a pair (4 ms,2 ms) that repeated 41 times. None of the artificial spike trains had a pair that repeated as frequently. Under the null hypothesis that jittering spikes over a 20 ms window did not change the statistical properties of the spike train, such an extreme value in the observed spike train would happen 1 in a 1000 times. So we can reject this null hypothesis ($p \leq .001$) in favor of the alternative hypothesis that the observed spike train has statistical structure on time scales smaller than 20 ms.

Of course, this is not surprising. We know already that neurons have fine temporal structure. In particular, refractory periods and bursting are types of temporal structure on fine time scales. We can preserve these particular types of structure using the model-free pattern jitter described in Section 7.6.3.1. This method has a pattern parameter $q$. Any consecutive spikes separated by $q$ or less will maintain their exact separation after jittering. Also, any spikes separated by more than $q$ will still be separated by more than $q$ after jittering. Among other things, this fixes the ISI histogram for ISIs $\leq q$.

We can try increasing $q$ to account for some of these features. When $q = 0$, the method prevents spikes from landing in the same bin (or swapping positions). Perhaps the original result was an artifact introduced because the jittered trains could have spikes in the same time bin? Using the $q = 0$ ms pattern jitter (with 20 ms jitter windows) to generate 999 artificial spike trains, we still find that none of them have a pair of ISIs that repeats at least 41 times, so we continue to reject the null ($p \leq .001$). When $q = 1$, the method enforces the (approximately) 1 ms refractory period. Nevertheless, we get the same result. When $q = 2$, the method begins to preserve any structure that comes from very rapid bursts, and in fact, 42 of the artificial spike trains have a pattern that repeats at least 41 times. This gives a p-value of .043, which is marginally significant. When $q = 3$, the p-value actually goes down to .022, presumably because of the stochastic nature of the Monte Carlo test.

The pattern jitter not only fixes the ISI histogram for ISIs $\leq q$, it also fixes the joint histogram for consecutive pairs of ISIs, each of which is $\leq q$. So when $q \geq 4$ ms, the ISI pair (4 ms,2 ms), which just happened to be the pair that repeated most frequently in the observed spike train, will occur exactly 41 times in every jittered spike train and the observed spike train will no longer be an outlier. So for $q \geq 4$ ms, the p-value is 1 and we cannot reject the null. This particular statistic does not provide any evidence for fine temporal structure (on the order of 20 ms or less) beyond that which can be explained by things like refractory period and bursting that occur on the order of 5 ms following a spike. A summary of these results are shown in Figure 7.11.

Another statistic that we might try is the average number of repeats over all of pairs of ISIs that occur in the data, namely,

$$\frac{\sum_{\delta_1,\delta_2} N(\delta_1,\delta_2)}{\sum_{\delta_1,\delta_2} \mathbb{1}\{N(\delta_1,\delta_2) > 0\}}.$$

Again, the original fixed partition jitter and the $q = 0$ pattern jitter (20 ms windows, 999 Monte Carlo samples) return p-values of .001. This time, however, the $q = 1$ pattern jitter returns a p-value of .187 and increasing $q$ tends to further increase the p-value, this time more progressively. The results are shown in Figure 7.12. As before, we would conclude that there is no evidence for fine temporal structure beyond that which comes within a few milliseconds after a spike.

In both of these examples it is easy to see how things like refractory period and bursting could introduce unwanted artifacts when using the original, independent jitter. Because of this, the artifacts could be corrected for without using a more sophisticated jitter method, say, by ignoring all pairs that contain an ISI less than $q$ ms. In more realistic and complicated settings, however, it may not always be evident how to anticipate or correct for such effects. The model-free pattern jitter methods provide a fast and simple alternative.

### 7.9.2 Nonaccidental synchrony

In this section we will demonstrate using the jitter methods as a way to measure synchronous firing between two simultaneously recorded neurons, while accounting for such influences like firing rates and short term history effects. We will show two stereotypical examples. Both involve jittering neuron `11a` (also used in the last section) which was recorded from area 5d of parietal cortex.

The first example compares `11a` with a simultaneously recorded neuron `70a` from primary motor cortex (area M1). `11a` and `70a` show statistically significant synchrony (fast, zero-lag temporal correlations) using the jitter methods. The second example compares `11a` with another simultaneously recorded M1 neuron `95a`. `11a` and `95a` do not show significant synchrony. The (smoothed) cross-correlograms of `11a/70a` and `11a/95a` are shown on the left and right, respectively, of Figure 7.13. Note that `11a/70a` have a peak around 0 ms, whereas `11a/95a` do not. It is known, however, that peaks in cross-correlograms can be misleading. In particular, although the peak appears to indicate fast temporal correlations, it could be an artifact of slow rate variations, perhaps even amplified by fast temporal history effects like refractory period and bursting.

It is worth noting that `11a` was recorded from a different multi-electrode array than `70a` and `95a`. It is also important to note that we are not using the full hour of recording, but are only using the subset of spikes that happened during the memory period of each successful trial, that is, the period between when the visual instruction (which indicates the target) was presented and when the animal was cued to begin moving. The go-cue appears between 900 and 1800 ms (randomly, uniformly) following the instruction. There were 268 successful trials. The mean firing rate of the three neurons during the the memory period was 17.9 Hz for `11a`, 16.6 Hz for `70a` and 19.1 Hz for `95a`. This corresponds to 6690 total spikes under consideration for `11a`.

About 9.7% of the spikes for `11a` had a corresponding synchronous spike in `70a` and 9.4% for `95a`. We call two spikes synchronous if they occur within $\pm 2.5$ ms of each other. This corresponds to synchronous firing rates of about 1.73 Hz and 1.68 Hz for `11a/70a` and `11a/95a`, respectively, during the memory period. It is interesting to try to quantify how many of these synchronous spikes cannot be explained by chance. For example, assuming that `70a` is an i.i.d. Bernoulli process (the discretization of a homogeneous Poisson process) in 1 ms bins, independent of `11a`, gives the easy calculation that

$$\begin{aligned}&\text{Prob}\{\text{spike } i \text{ in } \texttt{11a} \text{ has a synchronous spike in } \texttt{70a}\}\\&\quad = 1 - \prod_{\text{bins}} \text{Prob}\{\text{no synchronous spike in that bin}\} = 1 - (1 - \text{spikes / ms})^5 \approx .080.\end{aligned}$$

The 5 comes from the definition of synchrony as $\pm 2.5$ ms, which leads to 5 possible 1 ms bins that can result in a synchronous spike. Multiplying by the rate of firing of `11a` gives an expected synchrony rate of 1.43 Hz which is about 20% less than the observed rate of 1.73 Hz. The same calculation for `95a` gives an expected synchrony rate of 1.64 Hz which is quite close (2% less) than the observed rate of 1.68 Hz. This agrees qualitatively with the cross-correlograms. Note that accounting for mean rate variations across trials (by doing the above calculation separately for each trial) only slightly changes the results: 1.47 Hz for `70a` and 1.65 Hz for `95a`.

The previous calculations do not take into account covariations in rate or history effects in the spiking process. We can use jitter methods to do this. In particular, we can use exact methods to (effectively) jitter the spike times in `11a` by some amount $\Delta$ and compute the distribution of synchrony rates over all possible jittered spike trains. The mean of this distribution is the expected synchrony rate that results from taking into account all rate covariations on time scales larger than $\Delta$. If the jitter distribution accounts for certain patterns, like refractory period, then this is the expected synchrony after accounting for those effects as well. We used the spike centered pattern jitter described in Section 7.6.3 with a pattern parameter of 10 ms. Among other things, this will account for any effects resulting from refractory periods and bursting. We also fixed the first and last spike position in each trial to avoid any edge effects, but this has a negligible influence (and it can only make things more conservative).

The top two plots in Figure 7.14 show the expected synchrony as the jitter window size $\Delta$ is varied. (The plots are parameterized by how much a spike could be jittered, so that 5 ms on the $x$-axis corresponds to jittering by $\pm 5$ ms. After discretization, this is a jitter window with 11 1-ms bins.) We call this the accidental synchrony rate because this much synchrony can be expected to happen accidentally from the rates alone. Note that accidental synchrony depends on the parameter $\Delta$. As $\Delta$ decreases, the rates are allowed to fluctuate rapidly on finer and finer time scales, so the accidental synchrony should increase because more and more synchrony can be accounted for by allowing the firing rates of the two neurons to vary together. When $\Delta = 0$, every pair of synchronous spikes is accidental because they can be accounted for by allowing the rates to vary arbitrarily quickly.

An interesting measure is the nonaccidental synchrony rate, that is, the difference between the observed synchrony rate and the accidental synchrony rate. This is shown in the bottom two plots in Figure 7.14 as a function of $\Delta$. Controlling for rate variations on time scales greater than about 20 ms and controlling for fast history effects, the nonaccidental synchrony rate between `11a` and `70a` is about 0.19 Hz, or about 1.05% of all spikes. For `11a` and `95a` it is about .05 Hz or about 0.26% of all `11a` spikes. Even though the p-values returned by spike centered jittered do not correspond to a meaningful hypothesis test, it is worth noting that the nonaccidental synchrony rates for `11a/70a` were strongly significant (from .04 to .0003 for 5 ms to 30 ms, respectively), whereas they were not for `11a/95a` (all above .19).

## 7.10 Related Work

A variety of non-jitter methods for assessing the time scale of neural firing have appeared in the literature. See [1] for a review and a critique in the context of jitter methods. Monte Carlo fixed partition jitter (without allowing multiple spikes to land in the same bin) was first described by A. Date, E. Bienenstock and S. Geman [4, 5, 6]. They used the p-values returned by the method as a means to quantify the time scale of the spiking process. Monte Carlo spike centered jittered was used in [10] both to generate p-values (which is somewhat difficult to interpret) and as a statistic for comparing neural synchrony across experimental conditions. Exact, dynamic programming versions of these methods were described in a precursor to this work [9].

Exact (i.e., not Monte Carlo) jitter methods first appeared in A. Amarasingham's Ph.D. thesis [1]. The methods there preserve certain spiking patterns, analogous to Section 7.6.3.1, but the patterns are not necessarily jittered uniformly. Instead, the patterns are jittered using a distribution that gives the most extreme statistic (i.e., the most difficult to reject in a hypothesis test) but that is still "close" to uniform. The notion of closeness is parameterized so that it can be loosely interpreted as how fast the spiking process is changing. This parameter can then be rejected in a hypothesis test. Notice that this concept of fine temporal structure differs somewhat from the concept here, so the methods can be used in conjunction. One drawback of the methods in [1], at least in their current implementation, is that finding the jitter distribution requires a computationally intensive search.

## 7.11 Conclusion

We have described a large class of statistical methods that can be used to investigate the temporal resolution of neural spike trains. These so-called jitter methods are based on the intuitive idea of creating a collection of artificial spike trains that are identical to an observed spike train except perhaps that they do not have statistical structure finer than a certain time scale. If the observation can be distinguished among the collection, then this suggests that it has finer temporal structure than the collection. Some of the important features of these methods include the following:

- Even though many of the methods are exact, the implicit artificial collection can be sampled from and thus easily visualized and communicated.
- The experimenter has great flexibility for including certain types of fine structure (like refractory period, bursting or other patterns) in this collection and thereby excluding them from consideration.
- The methods are not restricted to hypothesis testing and p-values, but can also provide physiologically meaningful measurements of fine temporal structure (like nonaccidental synchronous firing rates).
- The methods are fast, especially on small spike trains, and so they are appropriate for use in online physiological experiments.

These last two considerations will be especially important as the community turns from questions about whether there is fine temporal structure to questions about whether it plays a role in neural information processing.

## Bibliography

[1] Asohan Amarasingham. *Statistical methods for the assessment of temporal structure in the activity of the nervous system.* PhD thesis, Division of Applied Mathematics, Brown University, 2004.

[2] C. D. Brody. Slow variations in neuronal resting potentials can lead to artefactually fast cross-correlations in the spike trains. *Journal of Neurophysiology*, 80:3345–3351, 1998.

[3] C. D. Brody. Correlations without sunchrony. *Neural computation*, 11:1537–1551, 1999.

[4] Akira Date, Elie Bienenstock, and Stuart Geman. On the temporal resolution of neural activity. Technical report, Division of Applied Mathematics, Brown University, Providence, RI, May 1998.

[5] Akira Date, Elie Bienenstock, and Stuart Geman. A statistical technique for the detection of fine temporal structure in multi-neuronal spike trains. *Society for Neuroscience Abstracts*, 25(568.5 (part 2)):1411, 1999.

[6] Akira Date, Elie Bienenstock, and Stuart Geman. A statistical tool for testing hypothesis about the temporal resolution of neural activity. *Society for Neuroscience Abstracts*, 26(828.6 (part 2)):2202, 2000.

[7] B.J. Frey. *Graphical Models for Machine Learning and Digital Communication.* MIT Press, Cambridge, MA, 1998.

[8] Stuart Geman and Kevin Kochanek. Dynamic programming and the graphical representation of error-correcting codes. *IEEE Transactions on Information Theory*, 47(2):549–568, February 2001.

[9] Matthew Harrison and Stuart Geman. An exact jitter method using dynamic programming. APPTS #04-3, Brown University, Division of Applied Mathematics, Providence, RI, March 2004.

[10] N. Hatsopoulos, S. Geman, A. Amarasingham, and E. Bienenstock. At what time scale does the nervous system operate? *Neurocomputing*, 52–54:25–29, June 2003.

[11] S.L. Lauritzen. *Graphical Models.* Oxford University Press, Oxford, U.K., 1996.

[12] M.W. Oram, M.C. Wiener, R. Lestienne, and B.J. Richmond. The stochastic nature of precisely timed spike patterns in visual system neural responses. *Journal of Neurophysiology*, 81:3021–3033, 1999.

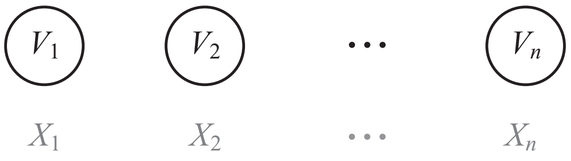

Figure 7.1: This figure shows the dependency graph for (independent) convolution. See Section 7.2.3.1.

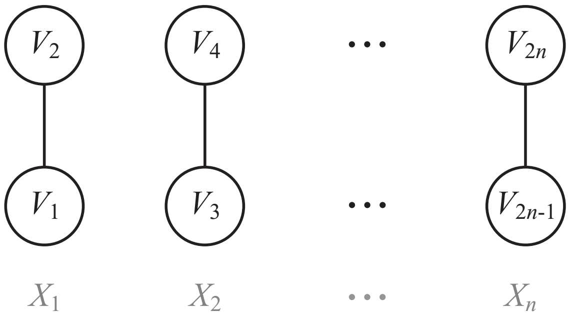

Figure 7.2: This figure shows the dependency graph for (independent) convolution of random variables. See Section 7.2.3.2.

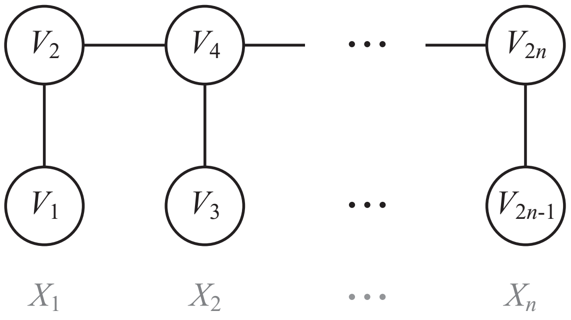

Figure 7.3: This figure shows the dependency graph for Markov dependent convolution. See Section 7.2.3.3.

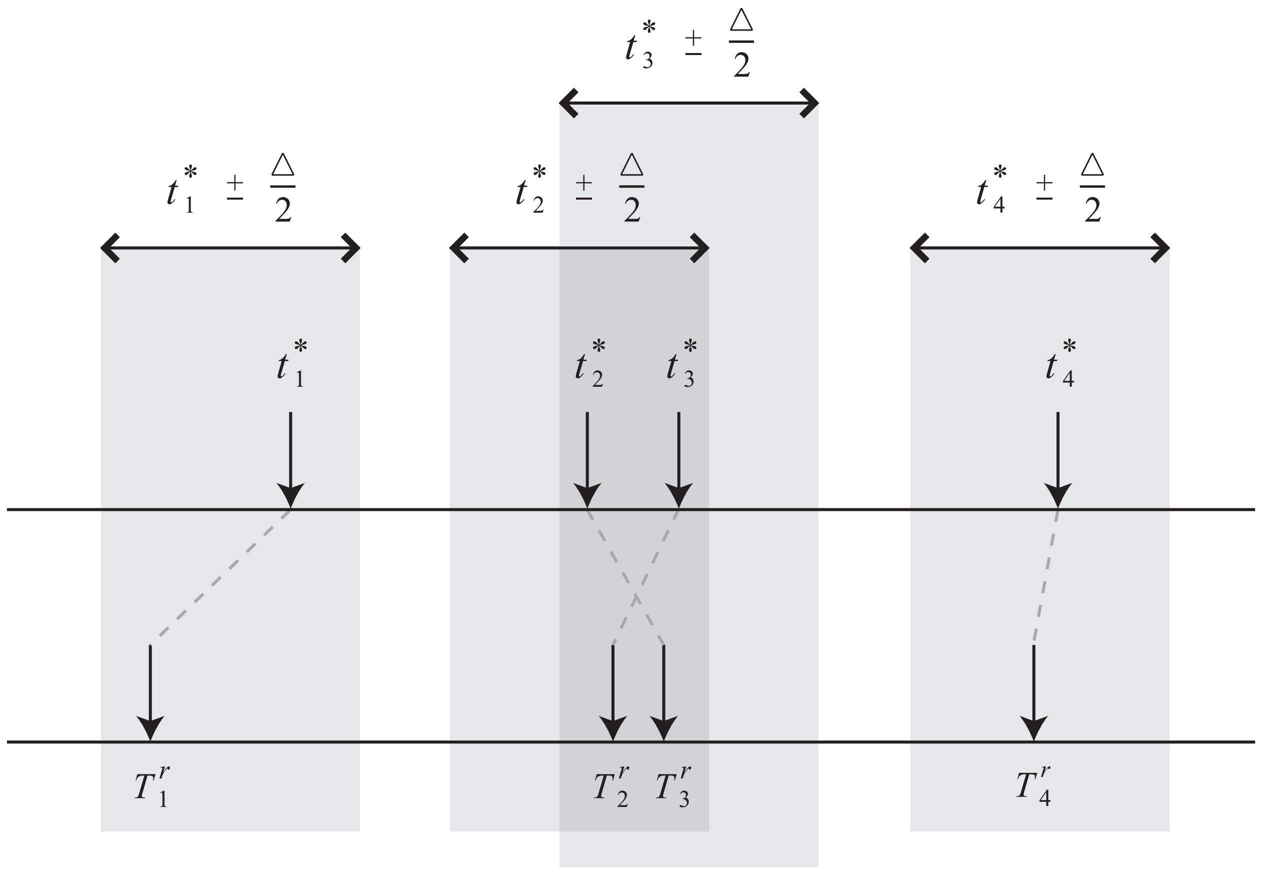
$t_3^* \pm \frac{\Delta}{2}$
$t_1^* \pm \frac{\Delta}{2}$
$t_2^* \pm \frac{\Delta}{2}$
$t_4^* \pm \frac{\Delta}{2}$
$t_1^*$
$t_2^*$
$t_3^*$
$t_4^*$
$T_1^r$
$T_2^r$
$T_3^r$
$T_4^r$

Figure 7.4: This figure details spike centered jitter. The observed spike train is indicated along the horizontal line in the middle. It has 4 spikes, $t_1^*, \ldots, t_4^*$. Each of these spikes is jittered uniformly by at most $\pm\Delta/2$. The gray bands centered on each of the observed spikes shows these $\Delta$-width jitter windows. The $r$th Monte Carlo sample is generated by sampling uniformly and independently from each jitter window. The dotted lines indicate where each of the observed spikes was jittered. Note that spikes are allowed to swap positions. This is remedied in later versions of the jitter method. The results of the jitter are ordered to create a Monte Carlo spike train $T_1^r, \ldots, T_4^r$. This can be repeated to create a large collection of artificial spike trains.

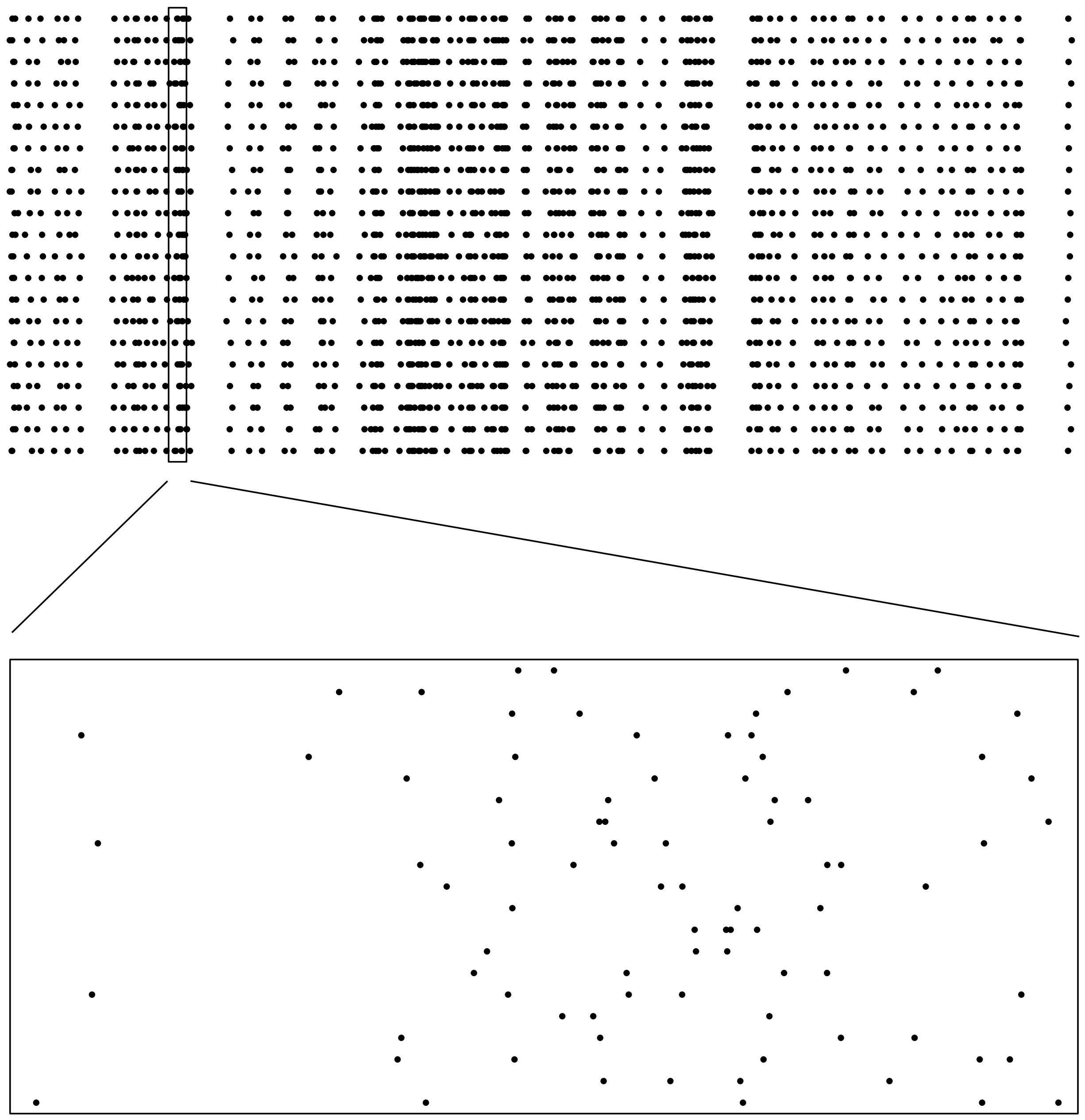

Figure 7.5: This figure shows an example of spike centered jitter on actual neural data. The top line shows a raster of spike times over a 3 second period. The remaining lines show examples of artificial spike trains generated by Monte Carlo spike centered jitter with a jitter window of $\pm 10$ ms. Note that the firing rate over coarse time scales (as indicated by the density of spikes) is preserved within the collection of artificial spike trains. The enlargement shows a 50 ms period. Note that any fine temporal structure in the original spike train is not preserved in the artificial samples.

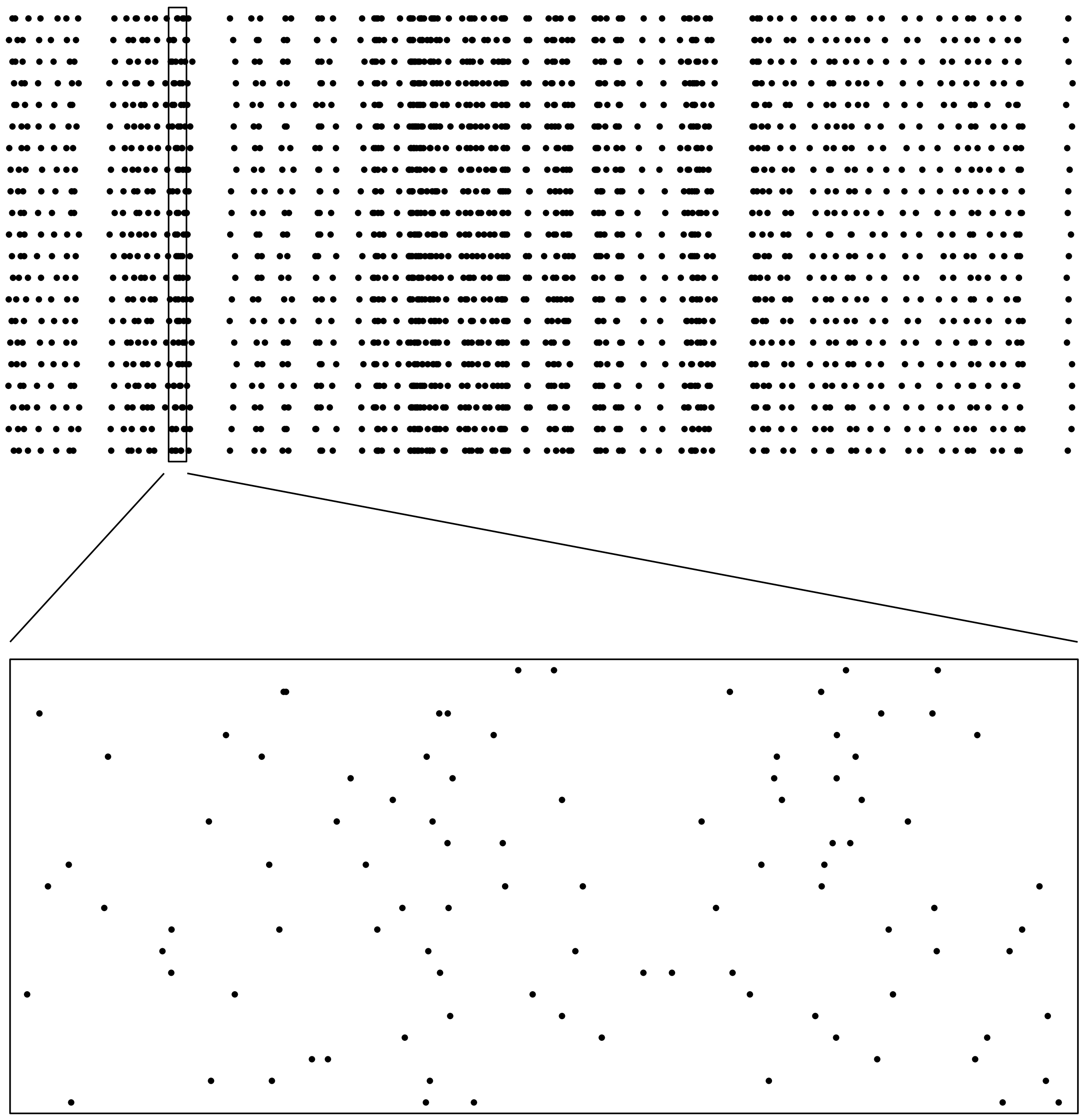

Figure 7.6: This is identical to Figure 7.5 except that the jitter distribution is now fixed partition jitter.

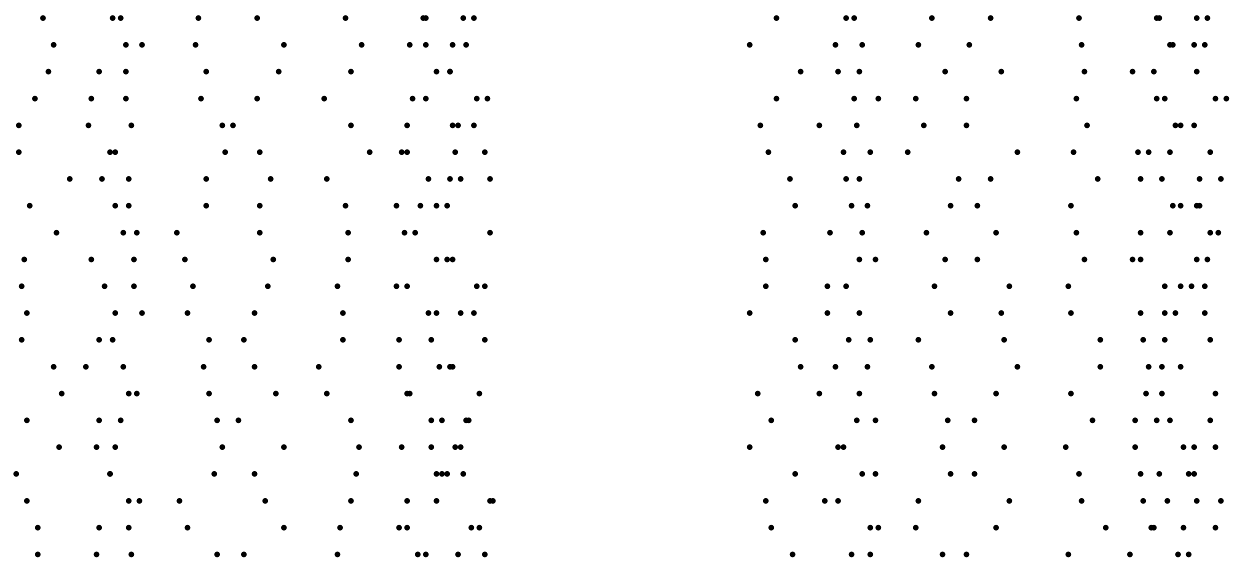
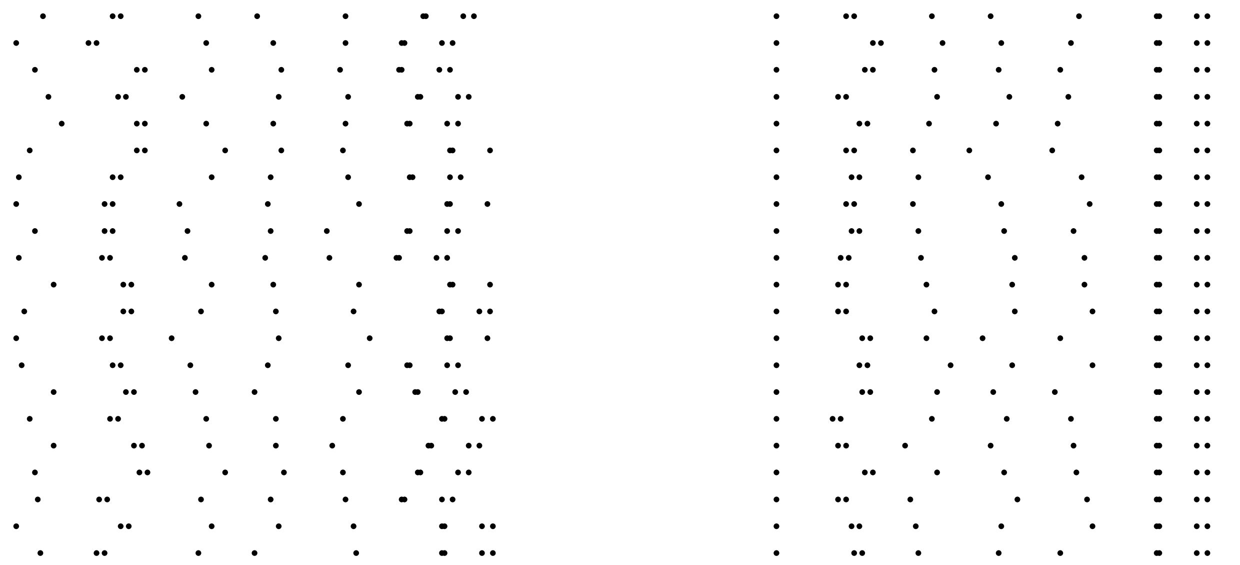
Figure 7.7: Examples of spike train sampling. The top raster in each of the four plots shows the same observed spike train over a 200 ms period. The remaining rasters show samples using a 20 ms jitter window (i.e., ±10 ms). The jitter distribution in the upper left plot is the original spike centered jitter. The upper right plot is spike centered jitter with the constraint that spikes cannot swap positions or occupy the same bin. The lower left plot shows model free spike centered jitter with a 20 ms pattern size. The lower right plot is the same thing except that the first and last spike positions are held constant.

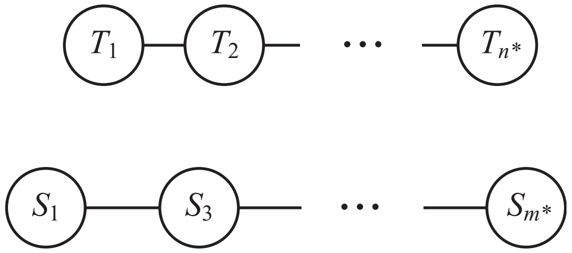

Figure 7.8: The dependency graph for the jitter distribution for simultaneous jitter (Section 7.8.1). Note that the statistic can add arbitrary complexity to this graph. See Figure 7.9.

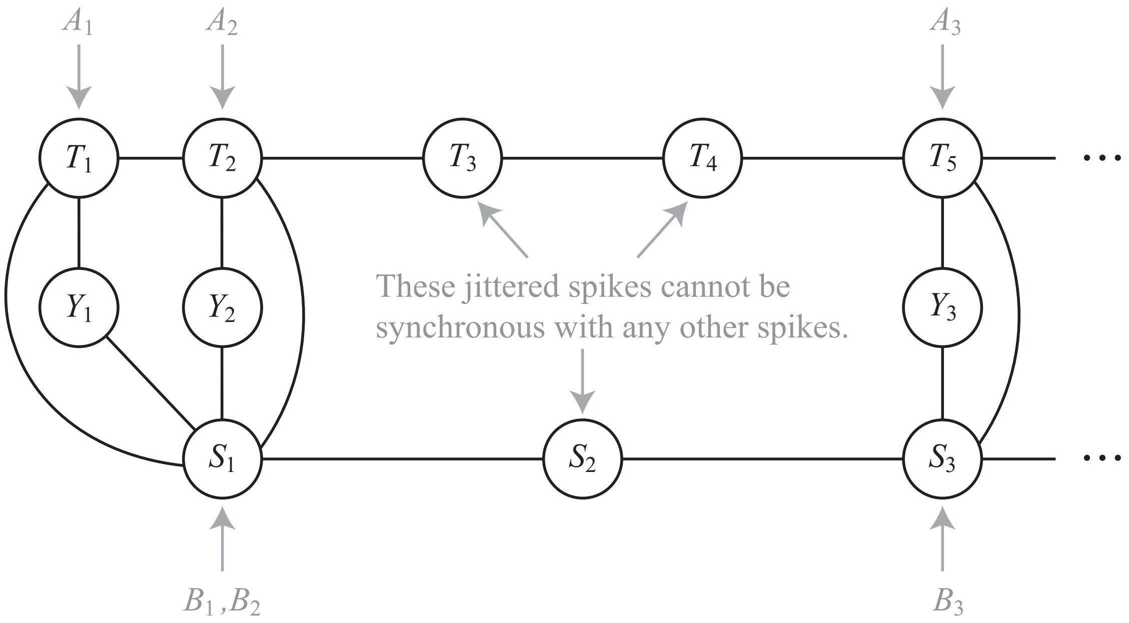

Figure 7.9: This shows an example of the observation/statistic-dependent dependency graph that can arise from a typical synchrony statistic with simultaneous jitter. See Section 7.8.1 for more details.

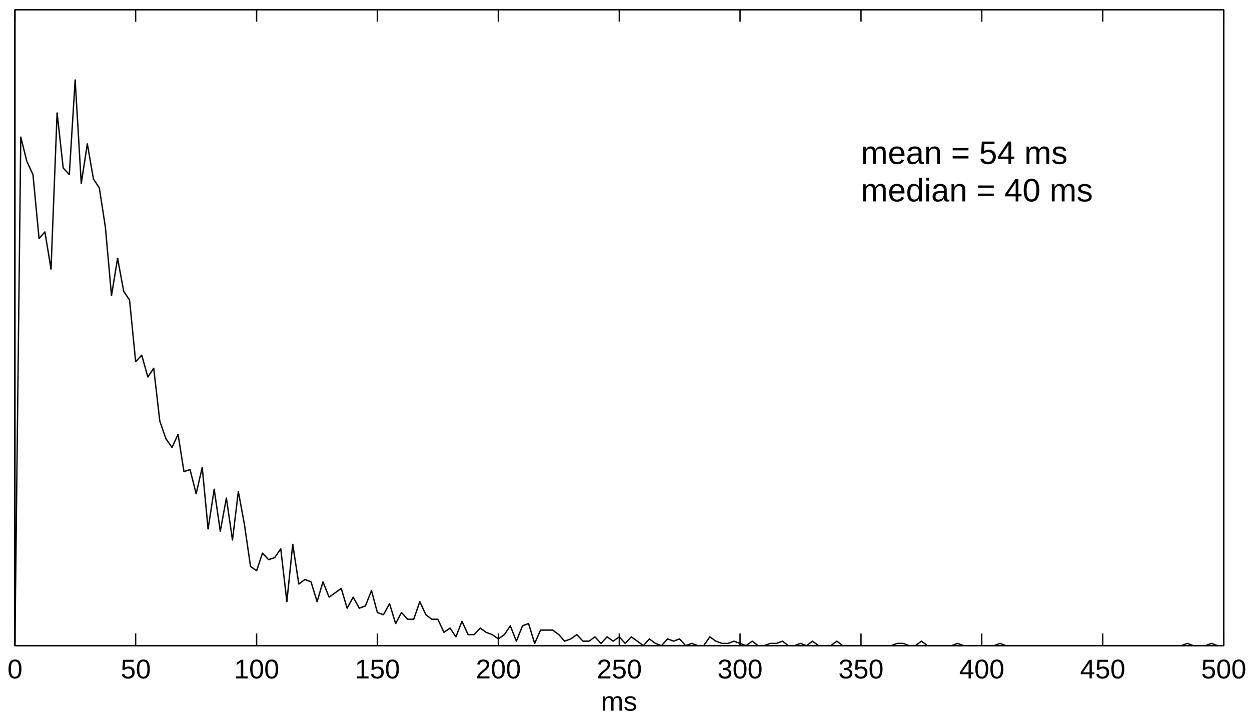

Figure 7.10: Interspike interval (ISI) histogram for neuron `11a`.

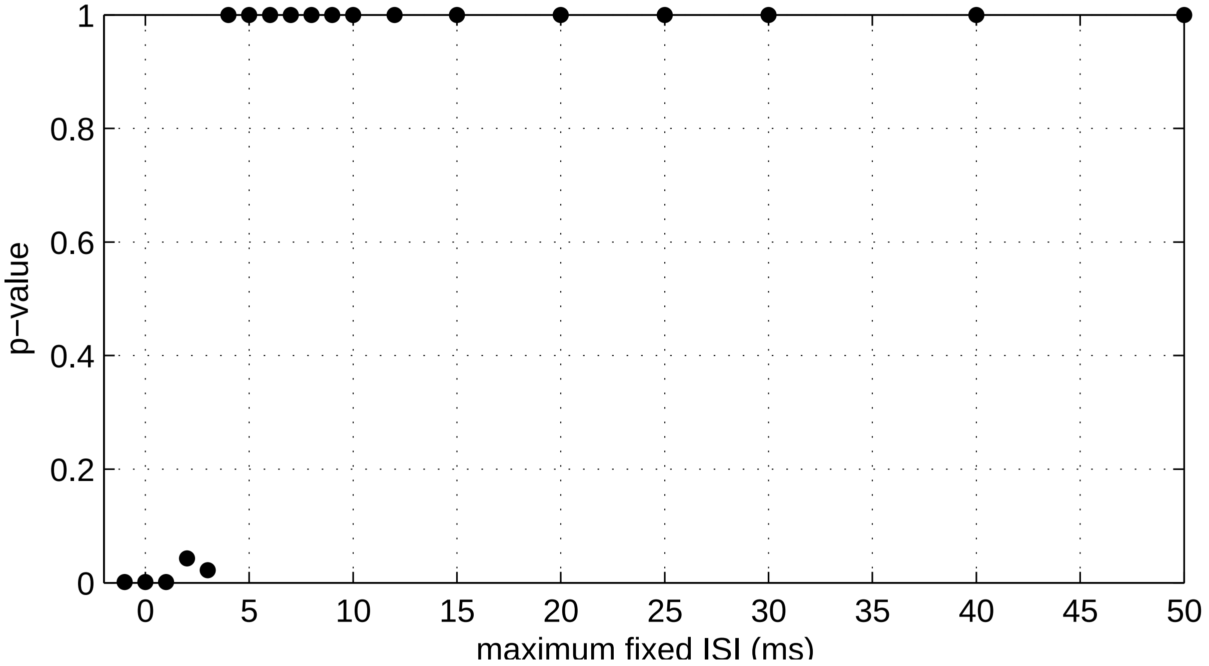

Figure 7.11: p-Values for the maximum frequency of repeating patterns. The first data point corresponds to no pattern constraints.

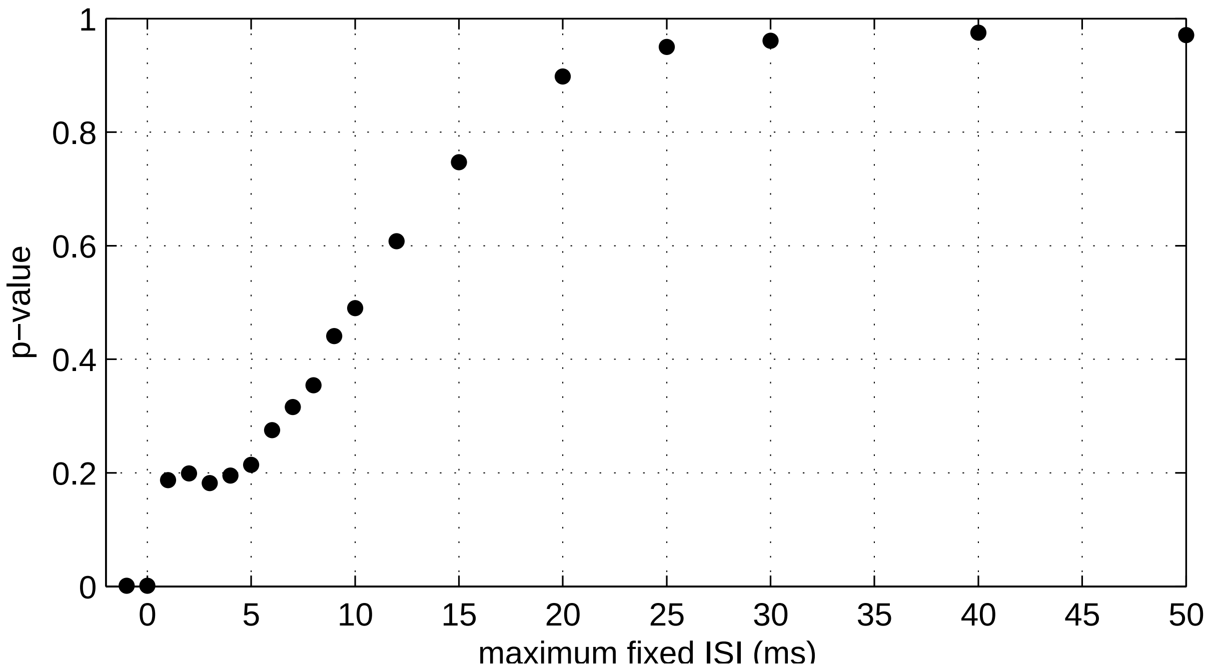

Figure 7.12: p-Values for the mean frequency of repeating patterns. The first data point corresponds to no pattern constraints.

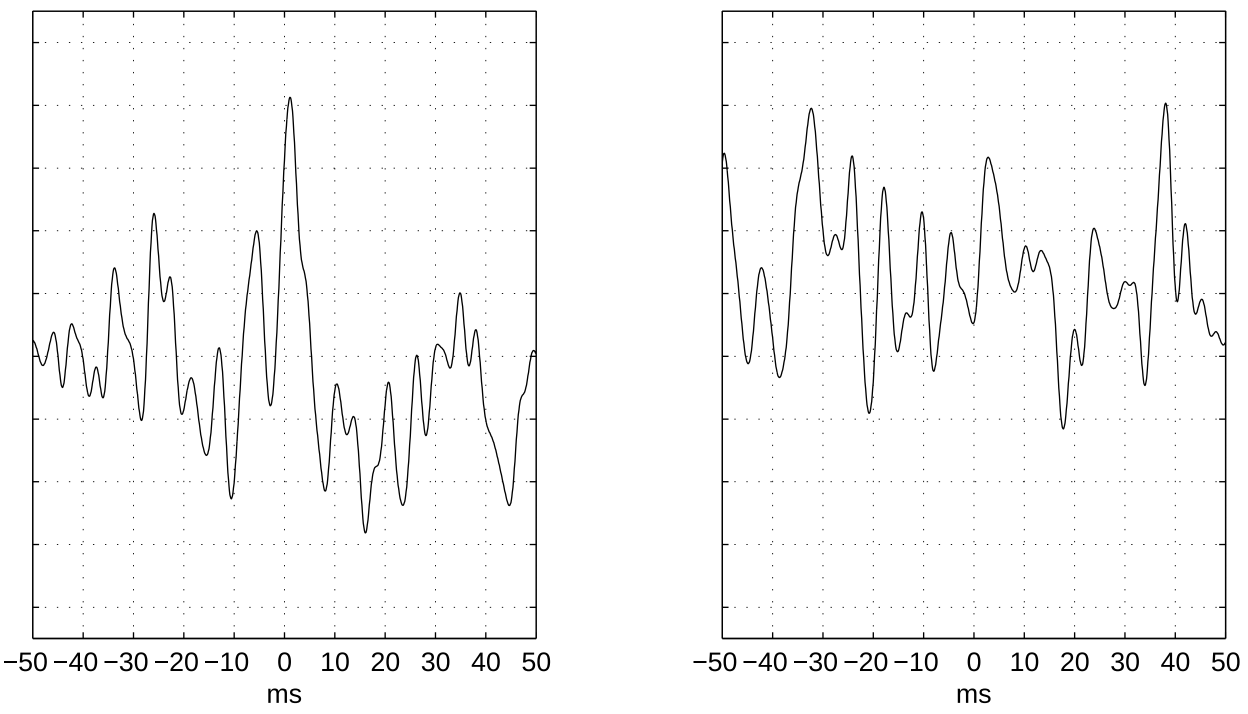

Figure 7.13: Smoothed cross correlograms (cross correlations). The left plot is for the neuron pair (`11a`,`70a`). The right is for (`11a`,`70a`).

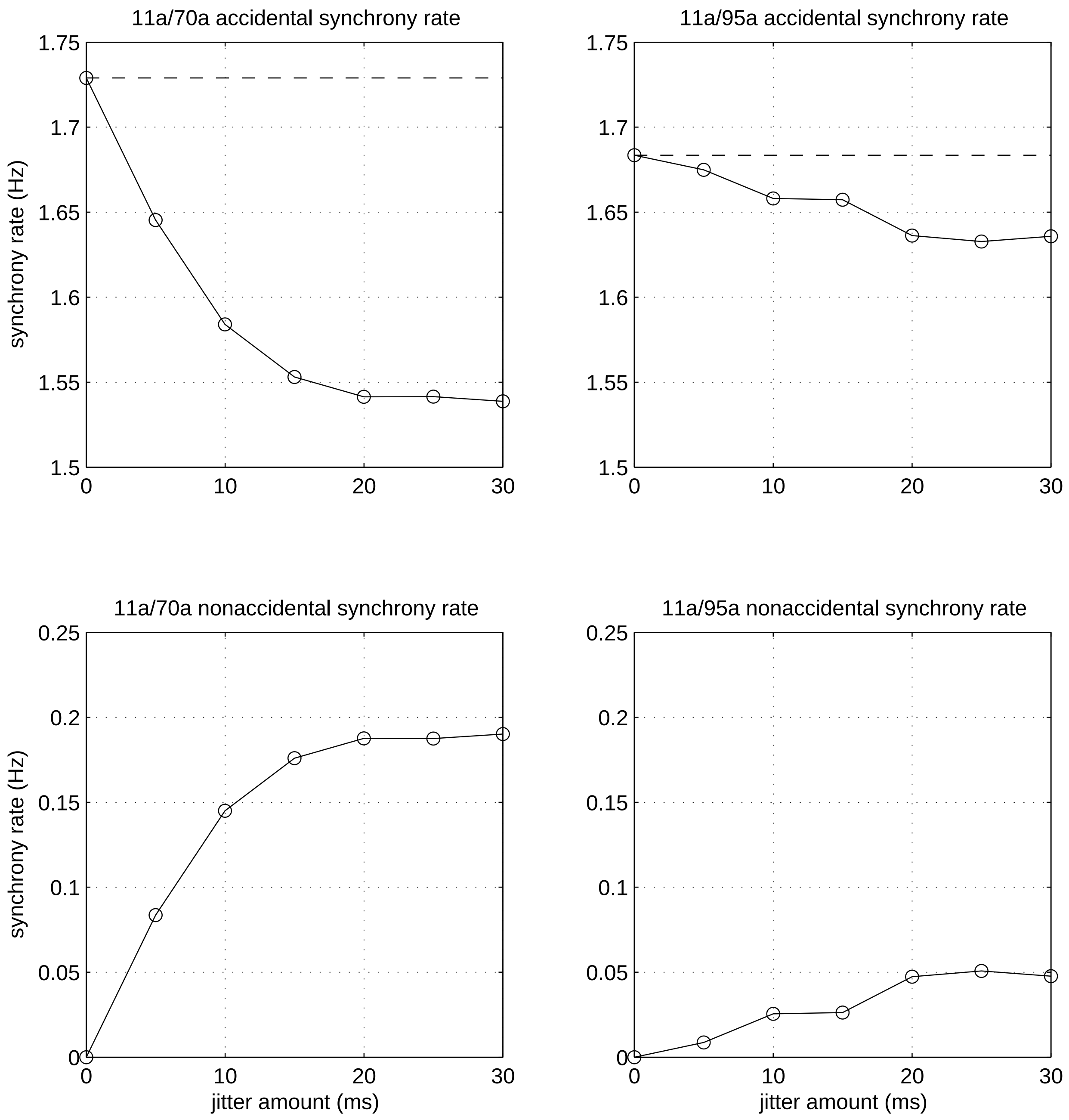

Figure 7.14: Accidental (top) and nonaccidental (bottom) synchrony rates. The left column compares neurons `11a` and `70a`. The right column compares `11a` and `95a`. The upper figures show the accidental synchrony rate, that is, the rate of synchronous firing that can be explained by the rates alone. The $x$-axis varies the jitter window size. Small values allow the rates to change quickly, so the accidental synchrony rate is higher – more synchronous spikes can be explained by allowing the rates to vary. The dashed line shows the amount of observed synchrony. The lower figures plot the difference between the observed synchrony and the accidental synchrony. This nonaccidental synchrony is a rate-controlled measure of excess synchrony.

# Chapter 8

# Conclusion

The general theme of this thesis was to suggest methods for discovering compositional structures in data. The unsupervised learning algorithms in the first part of the thesis hinted at some principles that might be useful for building compositional priors for image analysis. In particular, spatial dependencies were used to discover new compositions of old parts and temporal dependencies were used create invariant parts.

In future work, we hope to combine the two methods in order to create a truly compositional representation. This presumably involves the important and difficult task of designing computational principles for operating on a compositional hierarchy. We anticipate that creative solutions to the Markov dilemma, or the binding problem, will be crucial for both learning and computation within this framework.

Another line of future work will be to investigate information theoretic approaches that can unify selectivity and invariance. The principle of temporal stability offers one solution, but it is not clear how temporal stability formally relates to more statistical goals. Since, as we have shown, dependencies can be used to create both selectivity and invariance, it should also be possible to create an information theoretic criterion that can discover both.

The statistical methods in the second part of the thesis were designed for trying to understand how the brain might create compositional representations. Agnostic methods for investigating the receptive field properties of visual neurons are severely constrained by data limitations. We suggested that certain properties of natural images might facilitate their use. We also introduced a new class of jitter techniques which might be useful for investigating synchrony solutions to the binding problem.

In future work, we hope to experiment more with these methods on real data. The agnostic methods will undoubtedly need substantial work before they are practical. There may also be unanticipated physiological constraints other than mere sample size that severely limit their utility. The jitter methods, on the hand, are immediately applicable for spike train analysis. They should be useful for rate-controlled measures of synchrony regardless of how the brain actually uses synchrony, if it uses it at all.

Compositionality is an important, yet elusive principle for general pattern recognition. We believe that it might underlie the remarkably fast visual learning demonstrated by a variety of biological vision systems. Hopefully this thesis will contribute to increased interest in compositionality within the broader vision community.