LATENT VARIABLE MODELS OF SEQUENCE DATA FOR CLASSIFICATION AND DISCOVERY

by

Samuel J. Blasiak
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Huzefa Rangwala, Dissertation Director
_____	Dr. Daniel Barbara, Committee Member
_____	Dr. Carlotta Domeniconi, Committee Member
_____	Dr. Kathryn B. Laskey, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth S. Ball, Dean, The Volgenau School of Engineering

Date: _____ Fall Semester 2013
George Mason University
Fairfax, VA

Latent Variable Models of Sequence Data for Classification and Discovery

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Samuel J. Blasiak
Master of the Arts
Brandeis University, 2006
Bachelor of the Arts
Colorado College, 2004

Director: Dr. Huzefa Rangwala, Professor
Department of Computer Science

Fall Semester 2013
George Mason University
Fairfax, VA

Copyright © 2013 by Samuel J. Blasiak
All Rights Reserved

Dedication

I dedicate this dissertation to my parents, Jim and Mimi.

Acknowledgments

I am deeply grateful to my advisor, Dr. Huzefa Rangwala, for all his help during my PhD studies. Dr. Rangwala consistently provided me with guidance and encouragement even when it seemed like my work was not going well. I also especially want to Dr. Kathryn Laskey for her support through weekly meetings, for her attention to detail, and for the valuable advice that helped to improve the work going into this dissertation. Finally, I would like to thank my parents, siblings, family, and friends for their support and encouragement during my studies.

Table of Contents

List of Tables

List of Figures

Abstract

LATENT VARIABLE MODELS OF SEQUENCE DATA FOR CLASSIFICATION AND DISCOVERY

Samuel J. Blasiak, PhD

George Mason University, 2013

Dissertation Director: Dr. Huzefa Rangwala

The need to operate on sequence data is prevalent across a range of real world applications including protein/DNA classification, speech recognition, intrusion detection, and text classification. Sequence data can be distinguished from the more-typical vector representation in that the length of sequences within a dataset can vary and that the order of symbols within a sequence carries meaning. Although it has become increasingly easy to collect large amounts of sequence data, our ability to infer useful information from these sequences has not kept pace. For instance, in the domain of biological sequences, experimentally determining the order of amino acids in a protein is far easier than determining the protein's physical structure or its role within a living organism. This asymmetry holds over a number of sequence data domains, and, as a result, researchers increasingly rely on computational techniques to infer properties of sequences that are either difficult or costly to collect through direct measurement. The methods I describe in this dissertation attempt to mitigate this asymmetry by advancing state-of-the-art techniques for extracting useful information from sequence data.

This thesis explores a number of models over sequence data. These models were designed to produce alternate representations of sequences that distill relevant information, making

them both easier to process with traditional machine-learning tools and potentially improving on benchmarks over standard inference tasks such as classification and motif finding.

The first model I discuss in this thesis combines two types of statistical models, topic models and the Hidden Markov Model, in a novel way. Topic models, like Latent Dirichlet Allocation, make the simplifying assumption that words in a document are independently generated, while Hidden Markov Models assume a pairwise dependency over adjacent elements of a sequence. Our Hidden Markov Model Variant adds the pairwise dependency assumption back into the topic modeling structure. This structural change allows the HMM Variant to be used to extract fixed length representations of variable length sequences by accumulating statistics from the latent portions of the model. These fixed length representations can then be used as input to any number of standard machine learning algorithms that need fixed-length vector inputs. We show that these representations perform well for classifying protein sequences in conjunction with a support vector machine classifier.

The second model discussed in this thesis is an extension of the Profile HMM, a version of the Hidden Markov Model commonly used to represent biological sequences. Our Infinite Profile HMM modifies the basic Profile HMM to allow an infinite number of hidden states. To run inference given this infinite set of hidden states, we introduce a transformation of the model's hidden state space. This transformation allows us to compute an approximate marginal probability using only a finite amount of space by pruning low-probability configurations from the joint distribution. Our inference method not only allows inference for the infinite model but also significantly increases the speed of inference in the standard Profile HMM.

This thesis also covers methods to combine structure from multiple Profile HMMs. To accomplish this task, we first simplify the Profile HMM into a model that we call the Simplified Local Profile HMM (SL-pHMM). Two separate strategies can be used to combine multiple SL-pHMMs into a unified probabilistic model over sequences. The first strategy uses a separate "switching variable" for each element of a sequence. This switching variable selects which individual SL-pHMM generates an associated sequence element. The second strategy, which we call the Factorial SL-pHMM, constructs

probability distributions over individual sequence elements using a linear combination of the SL-pHMM hidden states. These strategies can then be further combined with a distribution over sequence labels, allowing the model to both generate sequence elements and the sequence label. We show that both of these strategies are effective for classifying synthetically-generated sets of sequences.

An extension of the Factorial SL-pHMM involves relaxing the hidden state space of the SL-pHMM to a continuous domain. If we place a regularizer that encourages sparsity on this new continuous space, then the new model shares many characteristics with a set of techniques frequently used in computer vision known as Sparse Dictionary Learning. This relaxation is the basis of our Relevant Subsequence Sparse Dictionary Learning (RS-DL) model. Applied to continuous sequences, RS-DL is effective at extracting human-recognizable motifs. In addition, subsequences extracted using RS-DL can improve on classification performance over standard nearest neighbor and dynamic time warping techniques.

The final contributions of this work involve incorporating Hidden Markov Model structure into a family of purely discriminative models. We call these models Subsequence Networks, and they operate by incorporating Profile HMM and Pair HMM structure into the lower level of a neural network. This structure is similar to convolutional neural networks, which have garnered state-of-the-art results in a number of tasks in computer vision. Subsequence Networks are competitive with state-of-the-art sequence Kernel methods for protein sequence classification but use a significantly different mode of operation.

Chapter 1: Introduction

In recent years, it has become increasingly easy to extract and store digital representations of sequence and time-series data. The pace of this collection is often far faster than our progress in making sense of this data. The methods I describe in this dissertation attempt to advance the state-of-the-art in understanding sequence data, focusing mainly on biological sequences.

Although large amounts of sequence data is available in public databases, extracting useful information from this data can be difficult. In the domain of protein sequences, for instance, determining characteristics such as a protein's physical structure or its role within a living organism remains difficult. As a result, biologists, as well as other domain experts, increasingly rely on computational techniques to infer important properties from raw sequence information.

Sequence data is not easily accepted by standard machine-learning algorithms which operate on sets of vectors. Unlike vector representations, sequences vary in length, and the order of symbols in a sequence carries meaning. A general approach for converting a sequence, $\mathbf{x}$, into vector representation is to define a fixed set of feature functions, $\varphi_1, \ldots, \varphi_M$, each of which extracts a different piece of domain-relevant information. We can then use these feature functions to map the original sequence to a vector suitable for input into a learning algorithm of choice by representing the original sequence as $\varphi(\mathbf{x}) = [\varphi_1(\mathbf{x}), \ldots, \varphi_M(\mathbf{x})]^\top$.

The obvious downside of this method is that it presupposes that we know which characteristics of our dataset are important and can define feature functions to capture these characteristics. For predicting structural or functional categories of protein sequences, these characteristics are not well-known. As such, researchers have attempted to define feature functions that can take into account a range of aspects of sequence data. A broad class of feature functions that exhibit this property stem from the mapping from sequences to counts

of each possible subsequence of length k. To create more inclusive feature representations of sequences, one could increase k. However, a vector of such subsequences would be of length $|\Sigma|^k$; as k increases, the size of the vector grows exponentially.

Circumventing this exponential feature vector growth leads to another core idea for representing sequences. A large class of learning algorithms allow training to take place given the dot products between training set feature vectors. Thus, instead of presenting $\varphi(\mathbf{x})$ to the learning algorithm directly, it is possible to precompute $\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle$ for all pairs of training set sequences $\mathbf{x}_i$, $\mathbf{x}_j$. This strategy is known as the kernel trick, where the kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is defined as $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle$. With the kernel trick, it is no longer necessary to define a set of feature functions. Rather, a kernel between pairs of sequences can be defined directly. Directly defining a kernel between sequences can allow additional prior knowledge to be incorporated into the learning algorithm. For instance, an effective set of kernel functions used for protein structure classification has incorporated edit distances between sequences parametrized by biologically-motivated scoring matrices between individual pairs of amino acids.

Although manually defined feature functions and kernels have led to successful classification algorithms over protein sequences, they possess a number of drawbacks. For the simpler kernels, the assumptions leading to the feature mapping are often not flexible enough to capture relevant characteristics of the data. For instance, feature vectors of k-mer counts often only allow a single fixed k - larger values of k require very long training set sequences to be effective, while smaller values of k are inhibited by noise. In contrast, more complex kernels often have a number of parameters which are typically learned through cross validation or set heuristically to fixed values.

Generative models that incorporate latent representations of training data offer a principled solution for extracting feature vectors from sequences [3]. By latent representation, I refer to any of the intermediate representations assumed in the generative process of a probabilistic model. For instance, a common probabilistic model over sequences, the Hidden Markov Model (Section 2.1), assumes that for each symbol in an observed sequence, a

hidden symbol is generated in the process of constructing the sequence. Given a dataset and a generative formulation, it is often possible to compute an optimum latent representation for the model. These optimal latent representation summarize important aspects of dataset elements and can often be used effectively to construct feature vectors for sequences.

1.1 Contributions

In this thesis I explore a number of models to extract useful features from sequences. A variety of common tasks can benefit from improved sequence representations. In much of this work, I focus on the classification task, but I also show how some of the models described in this thesis can be useful for motif finding. I explore these models from both the perspective of creating novel probabilistic models to generate sequence elements as well as from the discriminative perspective, in terms of classification systems that find boundaries between sequence categories by discovering structural features within sequence datasets. These contributions are summarized as follows:

- The **Hidden Markov Model Variant** [4] combines two types of statistical models, topic models and the Hidden Markov Model, in a novel way. Topic models, like Latent Dirichlet Allocation, make the simplifying assumption that words in a document are independently generated, while Hidden Markov Models assume a pairwise dependency over adjacent elements of a sequence. The Hidden Markov Model Variant adds the pairwise dependency assumption back into the topic modeling structure. This structural change allows the HMM Variant to extract fixed length representations of variable length sequences by accumulating statistics from the latent portions of the model. These fixed length representations can then be input to any number of standard machine learning algorithms that need fixed-length vector inputs. We show that these representations perform well for classifying protein sequences in conjunction with a support vector machine classifier.

- The **Infinite Profile HMM** [5] is a modification of the Profile HMM, a version of the Hidden Markov Model commonly used to model biological sequences, that allows an arbitrarily large number of hidden states. To run inference given this infinite set of hidden states we introduce a transformation of the Profile HMM hidden state space. This transformation allows us to compute an approximate marginal probability using a finite space by pruning low-probability configurations from the joint distribution. Our inference method not only allows feasible inference for the infinite model but also significantly increases the speed of inference in the standard Profile HMM.

- To **combine structure from multiple Profile HMMs**, I explore two strategies for combining hidden state spaces. These strategies operate on a simplified version of the basic Profile HMM, which we call the Simplified Local Profile HMM (SL-pHMM). The first strategy, the Switching SL-pHMM, uses a separate "switching variable" associated with each element of a sequence. This switching variable selects which individual SL-pHMM generates a given sequence element. The second strategy, known as the Factorial SL-pHMM, constructs a probability distribution over each sequence element using a linear combination of the SL-pHMM hidden states. These strategies can then be further combined with methods to represent sequence labels, allowing the model to both generate sequence elements and predict the label associated with the sequence. We show that both of these strategies are effective for classifying synthetically-generated sets of sequences.

- An extension of the Factorial SL-pHMM involves relaxing the discrete SL-pHMM hidden state space to a continuous domain. If we place a regularizer that encourages sparsity on this new continuous space, then the new model shares many characteristics with Sparse Dictionary Learning techniques frequently used in computer vision. This relaxation is therefore the basis of our **Relevant Subsequence Sparse Dictionary Learning** (RS-DL) model [6]. Applied to continuous sequences, RS-DL is effective at extracting human-recognizable motifs. In addition, subsequences extracted using RS-DL can improve on classification performance over standard nearest neighbor and

dynamic time warping techniques.

- **Subsequence Networks** [7] incorporate Hidden Markov Model structure into the lower level of a neural network. This structure is similar to that of convolutional neural networks, which have garnered state-of-the-art results in a number of tasks in computer vision. Subsequence Networks are competitive with state-of-the-art sequence Kernel methods on protein sequence classification problems but use a significantly different mode of operation.

Chapter 2: Background

In this chapter, I review a number of models and methods relevant to the work presented later in this thesis. Probabilistic models over sequences is a core topic of this thesis and, to better inform the discussion in later chapters, I present an overview of Hidden Markov Models (HMMs) and Profile Hidden Markov Models, a type of HMM commonly used for modeling biological sequences. Sequence classification is also a core area of this work, and I review a number of methods used for this task. I also discuss more-general methods for finding optimal parameters that I use repeatedly in later chapters. Finally, I discuss sequence datasets used for experiments in this work and methods for assessment.

2.1 Hidden Markov Models

When discussing distributions over sequences, the Hidden Markov Model (HMM) [2] is a common starting point. The HMM defines a probability distribution over sequences. The HMM assumes: (i) Each symbol in the sequence was generated from a mixture distribution; the mixture components are referred to as hidden states. (ii) The Markov property holds over hidden states i.e., the hidden state generating the current observation depends on the past only through the hidden state of the previous observation. Figure 2.1 illustrates the HMM's dependency structure.

The joint probability of a sequence, $x_{1:T}$ of length T, and a set of hidden states, $z_{1:T}$, under an HMM is given as follows:

$$p(x_{1:T}, z_{1:T}) = \prod_{t=1}^{T} p(z_t|z_{t-1})p(x_t|z_t) = \prod_{t=1}^{T} \theta^{trans}_{z_{t-1},z_t} \theta^{emit}_{x_t,z_t}, \tag{2.1}$$

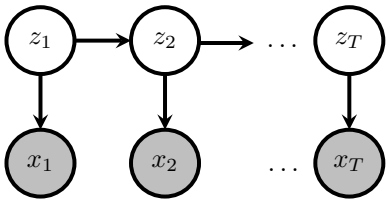


Figure 2.1: The dependency structure of the Hidden Markov Model.

K	the number of HMM hidden states
M	the number of possible amino acids
N	the number of protein sequences
T_n	the length of the n^{th} sequence
$\mathbf{x}_n$	is the n^{th} amino acid sequence, $x_{n,t}$ is the t^{th} symbol in the n^{th} sequence
y_n	indicates the category associated with the n^{th} sequence. $y_n \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of all categories
$\mathbf{z}_n$	is the n^{th} sequence of hidden states, $z_{n,t}$ is the value of the hidden state for the n^{th} sequence at position t. $\mathbf{z}_n \in \mathcal{Z}$, where $\mathcal{Z}$ the set of all possible hidden state sequences.
$\theta^{trans}_{k,k'}$	the probability of hidden state k occuring at position t when hidden state k' appears at position $t+1$
$w^{tran}_{k,k'}$	defined as $\log \theta^{trans}_{k,k'}$
$\theta^{emit}_{k,m}$	the probability of emitting symbol m at position t when hidden state at position t is k
$w^{emit}_{k,m}$	defined as $\log \theta^{emit}_{k,m}$
$\mathbf{w}$	defined as $\left[w^{trans}_{1,:} \ldots w^{trans}_{K,:} w^{emit}_{1,:} \ldots w^{emit}_{K,:}\right]^\top$, a vector comprising both the transition and emission weights - the subscripted ":" is matlab notation for the vector over the relevant index.
$n^{emit}_{k,m}$	the number of times hidden state k occurs in conjunction with observed symbol m
$n^{trans}_{k,k'}$	the number of times hidden state k occurs before hidden state k'

Table 2.1: Description of HMM parameters

where θ^{trans} is a set of transition probabilities and θ^{emit} is a set of emission probabilities[1]. Detailed descriptions of parameters are given in Table 2.1. Converting transitions from adjacent hidden states over the length of the sequence to counts of emissions and transitions gives

$$p(x_{1:T}, z_{1:T}) = \prod_{k,k'} \left(\theta^{trans}_{k,k'}\right)^{n^{trans}_{k,k'}} \prod_{k,m} \left(\theta^{emit}_{k,m}\right)^{n^{emit}_{k,m}} \tag{2.2}$$

In the logarithm, the joint probability of a sequence and associated hidden states under the HMM is a linear function:

[1] In this thesis we also often use the symbols A to indicate the emission probability matrix and B to indicate the transition probability matrix.

$$\begin{aligned} \log p(x_{1:T}, z_{1:T}) &= \sum_{k,k'} n_{k,k'}^{trans} \log \theta_{k,k'}^{trans} + \sum_{k,m} n_{k,m}^{emit} \log \theta_{k,m}^{trans} \\ &\stackrel{\text{def}}{=} \sum_{k,k'} n_{k,k'}^{trans} w_{k,k'}^{trans} + \sum_{k,m} n_{k,m}^{emit} w_{k,m}^{emit} \end{aligned} \tag{2.3}$$

where we define $w \stackrel{def}{=} \log \theta$ for both emissions and transitions. HMMs are probability distributions and thus require that $\sum_{\mathcal{X},\mathcal{Z}} p(\mathbf{x}, \mathbf{z}) = 1$, where $\mathcal{X}$ indicates the set of all possible sequences with alphabet size M, and $\mathcal{Z}$ indicates the set of all hidden states assignments for a sequence $x_{1:T}$. This constraint is satisfied as long as $\sum_{k'} \theta_{k,k'}^{trans} = 1$ and $\sum_m \theta_{k,m}^{emit} = 1$.

It is often useful to find the maximum probability assignment of values to hidden states, i.e., $\arg\max_{z_{1:T}} p(x_{1:T}, z_{1:T})$. The maximum can be computed efficiently by distributing the addition operator over max function to create the following recurrence:

$$\max_{z_{1:t}} \log p(x_{1:t}, z_{1:t}) = \max_{z_t} \left[\left(\max_{z_{1:t-1}} \log p(x_{1:t-1}, z_{1:t-1}) \right) + \log p(z_t | z_{t-1}) + \log p(x_t | z_t) \right] \tag{2.4}$$

The algorithm that uses this recurrence to compute the maximum over $z_{1:T}$ is known as the Viterbi algorithm [2]. A similar algorithm to compute marginal probabilities replaces the max in the Viterbi algorithm with a summation. this algorithm is known as the forward-backward algorithm and is a key step in HMM inference.

A convenient way to guide max and sum computations in the Viterbi and forward-backward algorithms is to use a graph, called a lattice[2], whose edges determine which terms take part in the summation. Figure 2.2 depicts the lattice for the HMM.

[2]We follow [2] in using the term "lattice" to refer to the directed acyclic graph of transitions between hidden states associated with each observed symbol. This graph is also commonly referred to as a trellis in the literature [8].

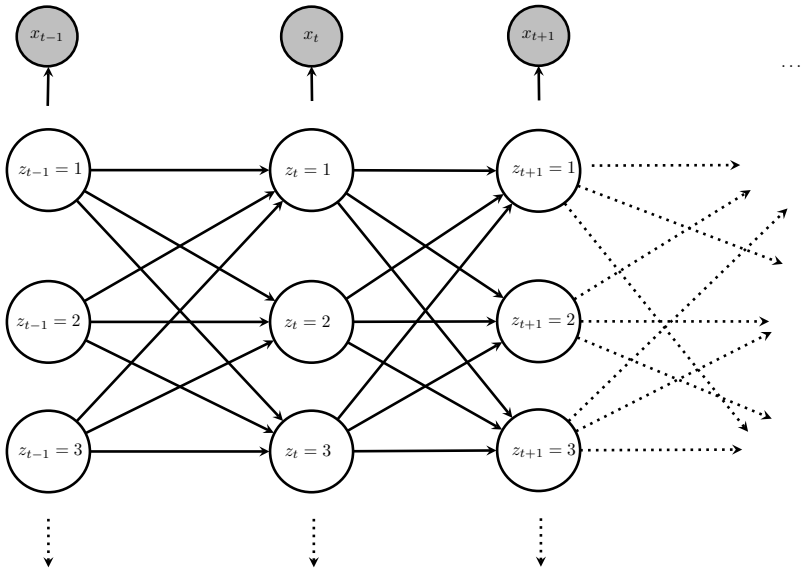


Figure 2.2: A portion of the HMM's lattice used for the Viterbi and forward-backward algorithms. The horizontal dimension represents elements in the sequence and the vertical dimension hidden state values. A forward computation is associated with each node in the graphs, and edges indicate which terms take place in the computation.

2.1.1 Profile HMMs

The Profile HMM (pHMM) [9] is an HMM with specific restrictions on transitions and emissions. The model is similar to the Bakis model [2] used in speech recognition in that it is a left-to-right, non-ergodic HMM. Like other left-to-right HMMs, the pHMM's utility lies in its ability to capture an archetypal sequence or sequence fragment through the emission distributions of a portion of the model's hidden states. Profile HMMs include three types of hidden states: *Match* states, which describe the archetypal sequence, *Insert* states, which allow the model to account for symbols not included in the archetypal sequence, and *Delete* states, which do not emit a symbol and allow the model to skip a *Match* or *Insert* state.

A sequence of symbols is generated from a pHMM by traversing states in the finite automata shown in Figure 2.3. The model begins in a designated start state, z_0, then transitions to the first *Match*, *Insert*, or *Delete* state. If the transition moves to a *Match* or *Insert* state, then a symbol is emitted. Typically, emissions from *Insert* states are evenly distributed across the symbol alphabet, while *Match* state emissions are attuned to symbol

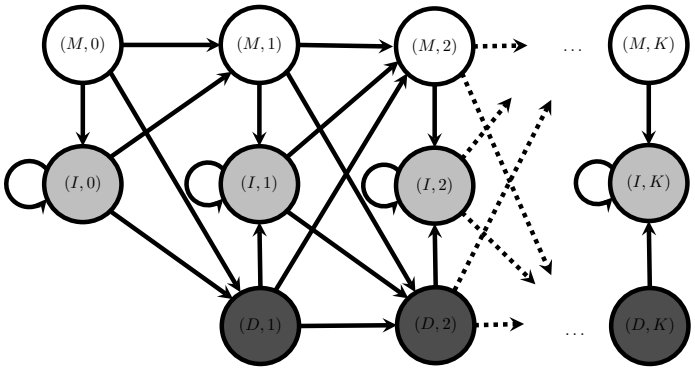


Figure 2.3: The pHMM's underlying Deterministic Finite-state Automaton (DFA) [2]: *Match* states are represented with a white background, *Insert* states by light gray, and *Delete* states by dark gray. A path through the DFA generates a sequence of observed symbols. In many pHMM constructions, transitions to the final state of the model (not pictured) can occur only from states $\{(M,K),(I,K),(D,K)\}$. In the model described in Chapter 4, any *Match* or *Insert* state can transition to the final state.

frequencies associated with a particular position in the archetypal sequence. If the model transitions to a *Delete* state, no symbol is emitted. From the k^{th} *Match, Insert,* or *Delete* state, denoted respectively as (M,k), (I,k), *or* (D,k), the pHMM can move to either *Insert* state (I,k), *Match* state $(M,k+1)$, or *Delete* state $(D,k+1)$. The standard pHMM can transition to a separate terminal state only from the K^{th} *Match, Insert,* or *Delete* state.

The joint probability of an observed sequence, $x_{1:T}$, and set of hidden states, $z_{1:|z|}$ is given by

$$p(x_{1:T}, z_{1:|z|}|\theta^{trans}, \theta^{emit}) = \prod_{(s,k),s'} \left(\theta^{trans}_{(s,k),s'}\right)^{n^{trans}_{(s,k),s'}} \prod_{(s\in\{M,I\},k),m} \left(\theta^{emit}_{(s,k),m}\right)^{n^{emit}_{(s,k),m}} \tag{2.5}$$

where $\theta^{trans}_{(s,k),s'}$ indicates the probability of transitioning from state (s,k) to the next $s' \in \{$*Match, Insert, or Delete*$\}$, and $\theta^{emit}_{(s,k),m}$ indicates the probability of emitting symbol m from state (s,k). Similarly, $n^{trans}_{(s,k),s'}$ indicates the number of transitions that occurred from hidden state (s,k) to s', and $n^{emit}_{(s,k),m}$ indicates the number of time symbol m was emitted from hidden

state (s, k).

2.2 Sequence Classification

Classifying sequences is a theme repeated many times across this thesis. In this section, I review a number of methods applied to the sequence classification problem.

The combination of Support Vector Machines (SVMs) and kernel methods are ubiquitous in sequence classification literature and have been highly successful in this domain. SVMs are linear classifiers; they assume that an input space, $\mathcal{X}$, can be partitioned by a hyperplane so that positive examples lie on one side of the plane and negative examples on the other. SVMs can capture nonlinear boundaries by mapping data into a transformed space, $\varphi : \mathcal{X} \rightarrow \mathcal{X}'$, where $\mathcal{X}$ is the original input space and $\mathcal{X}'$ is the transformed input space. Instead of computing this mapping directly, we can substitute the inner product between training examples in the transformed space, $\langle \varphi(x_i), \varphi(x_j) \rangle$, with a kernel function, $K(x_i, x_j)$, where $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ and $x_i, x_j \in \mathcal{X}$ [10].

String kernels extend the SVM to problem domains of variable-length sequences and also allow prior knowledge over a particular problem domain to be incorporated into the classifier. Examples of string kernels include the following: The *spectrum kernel* [11] computes, for a pair of sequences, x_i and x_j, the count of subsequences of length k that are present in both sequences. The *mismatch kernel* [12] can be best described as a fuzzy version of the spectrum kernel. For two sequences, x_i and x_j, the mismatch kernel computes the number of subsequences of length k across x_i and x_j that contain at most m mismatches. The *sparse spatial sample* kernel (SSSK) [13] extracts a set of substring probes of length k_i (in practice two or three are used), separated by gaps parametrized by a maximum length, from a sequence. Each set of probes is associated with a pattern of amino acids that could occur at multiple resolutions within a sequence. The *local alignment kernel* (LA-kernel) [14] computes the sum over all possible alignment scores between two sequences. Alignment scores generalize edit distance and score pairs of individual amino acids using a predefined

distance matrix, commonly the BLOSUM62 matrix [15]. Profile kernels [16, 17] are semi-supervised methods that augment training and test sequences with unlabeled sequences in the Protein Data Bank (PDB). Cluster kernels are another set of semi-supervised techniques. Weston et. al. [18] present a number of methods to augment a small set of labeled data with proteins from a large unlabeled corpus.

Non-SVM methods have also been successful for sequence classification. Ifrim et. al. [19] solves a regularized classification problem in a way that combines feature selection in the high dimensional space of fuzzy sequence matches with optimization with respect to feature weights. Ding and Dubchak [20] apply standard three-layer neural networks using a feature vector composed of a number of derived amino acid characteristics. Finally, models similar to the basic Profile HMM such as SAM **[21]** and HMMER **[22]** have been used for classification by comparing class-conditional probabilities from models trained on individual protein categories.

For continuous-valued sequences, different strategies are typically used for classification than when dealing with discrete-valued protein sequences. One important difference between these domains is that continuous sequences are often extracted from longer-length time series data, so researchers are more willing to crop sequences to a uniform length and simply treat them as vector data [23].

Once sequences are in vector format, standard classification tools, such as Nearest Neighbor algorithms, SVMs, Artificial Neural Networks, and Decision Trees, can be used [23–25]. These standard classifiers on vector representations, however, overlook a critical characteristic of sequence data: patterns in sequences are not tied to a fixed set of indices but can occur at any point in the sequence. This observation motivates Dynamic Time Warping (DTW) [26] distance between sequences, which aligns similar subsequences across sequence pairs. The DTW procedure shares commonalities with the LA-kernel and profile kernels from protein sequence classification literature. DTW distance is typically more effective than Euclidean distance [23] but suffers from high complexity ($O(MN)$, where M and N are lengths of the respective sequences). A major focus in sequence classification literature

has therefore been in creating fast approximate DTW algorithms [27,28].

2.3 Inference and Optimization Methods

In this section, I describe optimization methods that I use in a number of chapters in this thesis. The models in this thesis are formulated both as probabilistic graphical models and as optimization problems with well-defined objectives and constraints. In each case, different methods for discovering model parameters may be appropriate.

2.3.1 Variational Inference

The term variational inference can encompass any number of methods for constructing parametrized bounds on a marginal distribution [29,30]. Only the small subset, discussed below, of this large class of methods are employed in this work.

One meaning of the term variational inference [31,32] involves a class of methods used to maximize the likelihood of a probabilistic model with respect to a set of parameters. In this discussion, a probabilistic model defines a probability distribution over a set of examples, $\mathbf{x}$, parametrized by a set of variables, θ. These models make simplifying assumptions about the process used to generate this data that often involves postulating the existence of hidden information, which we represent by the vector $\mathbf{z}$. To compute the likelihood of the data under the model, all configurations of hidden states must be accounted for in the form of a sum or integral over these possibilities. Computing these sums is often intractable, so approximate methods become necessary for inference.

More formally, given a joint distribution $p(\mathbf{x}, \mathbf{z}|\theta)$, we would like to compute[3] $\max_\theta \sum_\mathbf{z} p(\mathbf{x}, \mathbf{z}|\theta)$, or, equivalently, $\max_\theta \log\left(\sum_\mathbf{z} p(\mathbf{x}, \mathbf{z}|\theta)\right)$. The basic strategy in variational inference involves constructing a lower bound on the log marginal probability as follows:

[3]For this explanation, we treat $\mathbf{z}$ as a discrete-valued vector. However, variational techniques are applicable for latent variables taking values from any Lebegue-measurable set. Thus, in the more-general case, the integral $\int_{d\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\theta)$ can be substituted for $\sum_\mathbf{z} p(\mathbf{x}, \mathbf{z}|\theta)$.

$$
\begin{aligned}
\log p\left(\mathbf{x}|\theta\right) &= \log\left(\sum_{\mathbf{z}} p(\mathbf{x},\mathbf{z}|\theta)\right) \\
&= \log\left(\sum_{\mathbf{z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p(\mathbf{x},\mathbf{z}|\theta)\right) \\
&\geq \sum_{\mathbf{z}} q(\mathbf{z}) \log\left(\frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z})}\right) \\
&= \sum_{\mathbf{z}} q(\mathbf{z}) \log\left(\frac{p(\mathbf{z}|\mathbf{x},\theta)}{q(\mathbf{z})}\right) + \log p(\mathbf{x}|\theta) \\
&= -\mathrm{KL}\left(q(\mathbf{z}) \,||\, p(\mathbf{z}|\mathbf{x},\theta)\right) + \log p(\mathbf{x}|\theta) \\
&\stackrel{\text{def}}{=} \mathcal{F}\left(\theta, q(\mathbf{z})\right)
\end{aligned}
\tag{2.6}
$$

where we have applied Jensen's inequality [33] in combination with the convexity of the log function in the third line of Equation 2.6 to obtain the lower bound $\mathcal{F}\left(\theta, q(\mathbf{z})\right)$. It is relevant to note that $\mathcal{F}\left(\theta, q(\mathbf{z})\right)$ is not a single lower bound, but a parametrized bound that holds for all values of the probability mass function $q(\mathbf{z})$.

The distribution, $q(\mathbf{z})$, is known as the "variational distribution" and can be chosen in a variety of ways. A common way of doing so is to take $q^*(\mathbf{z}) = \arg\max_{q(\mathbf{z})} \mathcal{F}\left(\theta, q(\mathbf{z})\right)$. By Gibbs' inequality [33], this maximum occurs at $q^*(\mathbf{z}) \propto p(\mathbf{x},\mathbf{z}|\theta) = p(\mathbf{z}|\mathbf{x},\theta)$. Once $q^*(\mathbf{z})$ is known, it then becomes practical to compute $\theta^* = \arg\max_{\theta} \mathcal{F}\left(\theta, q^*(\mathbf{z})\right)$ as long as it is also practical to compute $E_{q^*(\mathbf{z})}\left[\log p(\mathbf{z},\mathbf{x}|\theta)\right]$. This strategy is known as Expectation-Maximization (EM) [34, 35].

When computing $E_{q^*(\mathbf{z})}\left[\log p(\mathbf{z},\mathbf{x}|\theta)\right]$ is not practical, an alternative strategy is to assume independence between variational distributions over individual elements of the vector, $\mathbf{z}$. This independence assumption leads to a variational bound of

$$\mathcal{F}\left(\theta, q(z_1), \ldots, q(z_K)\right) = \sum_{z_1,\ldots,z_K} \prod_{k=1}^{K} q(z_k) \log\left(\frac{p(\mathbf{x}, \mathbf{z}|\theta)}{\prod_{k=1}^{K} q(z_k)}\right) \tag{2.7}$$

Isolating terms that depend on z_k we obtain

$$\mathcal{F}\left(\theta, q(z_1), \ldots, q(z_K)\right) = \sum_{z_k} q(z_k) \log\left(\frac{\exp\left(E_{\prod_{k'\neq k} q(z_{k'})}\left[\log p(\mathbf{x}, \mathbf{z}|\theta)\right]\right)}{q(z_k)}\right) \tag{2.8}$$

$$- \sum_{k'\neq k} \sum_{z_{k'}} q(z_{k'}) \log q(z_{k'})$$

where $z_{\neg k}$ indicates the set of all elements of $\mathbf{z}$ excluding z_k, $\{z_{k'} \,|\, k' \in [1 \ldots K]\,,\, k' \neq k\}$. Again, by Gibbs' inequality, we can find the with respect to each $q^*(z_k)$ at $q^*(z_k) \propto \exp\left(E_{\prod_{k'\neq k} q(z_{k'})}\left[\log p(\mathbf{x}, \mathbf{z}|\theta)\right]\right)$. This strategy of assuming independence between all elements of $\mathbf{z}$ in the variational distribution is known as mean-field variational inference, and the decomposition $q(\mathbf{z}) = \prod_{k=1}^{K} q(z_k)$ is known as the mean-field approximation [31, 36]. With the mean-field approximation, practical computation of the expectations

$$E_{\prod_{k'\neq k} q(z_{k'})}\left[\log p(\mathbf{x}, z_1, \ldots, z_K|\theta)\right] \tag{2.9}$$

and

$$E_{\prod_{k} q(z_k)}\left[\log p(\mathbf{x}, z_1, \ldots, z_K|\theta)\right] \tag{2.10}$$

is often possible because (for discrete-valued $\mathbf{z}$'s) their computation involves a series of independent summations rather than summing over the exponential number of configurations of $\mathbf{z}$. Inference then becomes a matter of iteratively maximizing $\mathcal{F}\left(\theta, q(z_1), \ldots, q(z_K)\right)$ first with respect to each $q(z_k)$, then with respect to θ, in a manner similar to EM.

A disadvantage of the mean field approximation is that the bound $\mathcal{F}(\theta, q(\mathbf{z}))$ is no longer tight, as is the case with EM. (Tightness of the variational bound in EM can be verified by substituting $q^*(\mathbf{z}) = p(\mathbf{z}|\mathbf{x}, \theta)$ into Equation 2.6 to obtain $\mathcal{F}(\theta, q^*(\mathbf{z})) = \log p(\mathbf{x}|\theta)$.) Typically, better solutions to the original maximum-likelihood problem can be found if tighter variational bounds can be constructed. Methods to construct tighter bounds are therefore often the focus when developing variational inference algorithms to solve a particular problem. One method to construct tighter bounds, known as structured variational inference [31,36], involves assuming independence between groups of elements, rather than individual elements, of $\mathbf{z}$. We employ this variational approximation in Chapters 4 and 5. Another method for tightening the lower bound involves marginalizing with respect to a subset of elements of $\mathbf{z}$ and is known as collapsed variational inference [37].

The term variational inference can also be used in a different sense to refer to methods for constructing parametrized bounds which allow a particular expectation to be evaluated. For instance, Jaakola et. al. [32,38] construct parametrized bounds on the logistic sigmoid, making inference feasible in the Sigmoid Belief Network and Bayesian logistic regression. We employ this approach in the models described in Chapter 5.

2.3.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) [39–41] is an optimization method for objective functions that decompose over individual examples in a dataset. SGD is often applied in the supervised empirical risk minimization setting in minimizing the error given by a loss function, $\ell(y_n, \hat{y})$, that compares a hypothesized output, $\hat{y}$, to the ground truth label, y_n, over a dataset consisting of N draws $\{x_n, y_n\}_{n=1}^N$ from a fixed distribution $P(x, y)$. Minimization occurs over a set of functions $f_\theta \in \mathcal{F}$, parametrized by θ:

$$L(y_{1:N}, x_{1:N}; \theta) = \sum_n \ell(y_n, f_\theta(x_n)) \tag{2.11}$$

It is often possible to use a batch gradient descent algorithm, which iteratively updates θ as follows:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma \frac{\partial}{\partial \theta} L(y_{1:N}, x_{1:N}; \theta) \tag{2.12}$$

where $\theta^{(t)}$ is the value of θ at step t of the algorithm and γ is a fixed learning rate. The SGD procedure contrasts with the batch procedure in that a single example is drawn from the dataset and the gradient with respect to the loss associated with this example is used to update the model parameters:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma\left(t\right) \frac{\partial}{\partial \theta} \ell\left(y_n, f_\theta\left(x_n\right)\right)$$

Bottou et. al. [39] show that if the learning rate, $\gamma\left(t\right)$, is chosen carefully, then, under certain assumptions about the expected loss function, in the limit as t grows to infinity, $\theta^{(t)}$ will converge to the optimum almost surely for convex loss functions. They also show that convergence to local minima with good performance is also possible in the non-convex case under reasonable conditions [39].

SGD excels because it generally requires fewer computations to achieve a specific level of expected risk, $E_{P(x,y)}\left[\ell\left(y, f_\theta(x)\right)\right]$, despite the fact that it often requires more iterations to achieve a specified level of empirical risk, $\sum_n \ell\left(y_n, f_\theta(x_n)\right)$, than batch methods [40]. This characteristic makes SGD especially effective in cases where datasets are large enough that computation time rather than the amount of available training data is a limiting factor in finding model parameters that produce low generalization error.

2.4 Datasets and Evaluation

For many experiments in this thesis, I use datasets derived from the Structural Classification of Proteins (SCOP) [42] database. The SCOP database categorizes proteins from the Protein Databank (PDB)[4] into a multilevel hierarchy that captures commonalities between protein structure at different levels of detail (see Figure 2.4). The ASTRAL compendium[5], provides versions of SCOP datasets filtered to remove sequences whose structures are significantly similar, allowing for less biased classification.

To eliminate high levels of similarity between sequences across the training and test sets that could lead to trivially good classification results, the standard method for partitioning a set of filtered SCOP sequences into a training and test set imposes certain constraints. For the fold level classification problem, training sets are partitioned so that no examples that share superfamily labels are included in both the training and test sets. Similarly, for the superfamily level classification problem (referred to as the remote homology detection problem [1, 43]), no examples that share a family-level category are included in both the training and test sets.

I evaluate the classification algorithms in this thesis primarily by comparing areas under the receiver operating characteristic (ROC) curve. The ROC curve plots the percentage of true positives against the percentage of false positives and the area under the curve (AUC) measures how frequently a positive example is ranked above a negative example. In some cases, I report AUC_{50} or $\text{AUC}_{10\%}$ which indicate the area under the ROC curve excluding all but the top 50 negative examples or top 10 percent of the negative examples respectively.

[4] http://www.pdb.org/pdb/home/home.do
[5] http://astral.berkeley.edu/

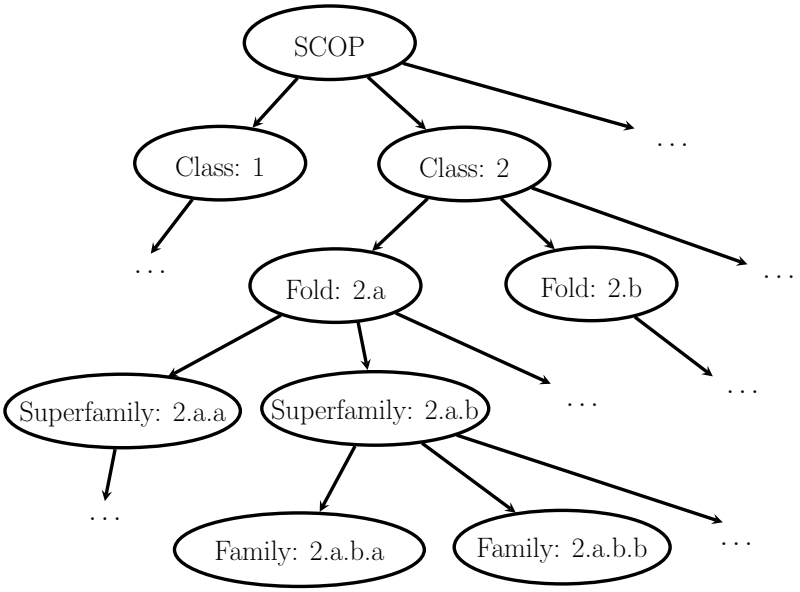


Figure 2.4: An example of the SCOP hierarchy of protein structural categories.

Chapter 3: A Hidden Markov Model Variant for Sequence Classification

In this Chapter, I discuss the Hidden Markov Model Variant, a model that allows a fixed-length vector representation to be constructed from variable-length sequences. This model provides a principled approach for constructing vector representations from sequence data.

A variety of strategies, depending on the problem type, are typically used to map sequences to representations that can be handled by machine learning methods that accept vector inputs. A simple technique involves selecting a fixed number of elements from the sequence and then using those elements as a fixed-length vector in the classification engine. In another technique, a small subsequence length, ℓ, is selected, and a size M^ℓ vector is constructed containing the counts of all length ℓ subsequences from the original sequence. A third method for classifying sequence data requires only that a positive definite mapping be defined between sequences rather than any direct mapping of sequences to vectors. This strategy, known as the kernel trick, is often used in conjunction with support vector machines and allows for a wide variety of sequence similarity measurements to be employed.

Hidden Markov Models (HMM) [44, 45] have a rich history for modeling sequence data (in speech recognition and bioinformatics applications) for the purposes of classification, segmentation, and clustering. HMMs' success is based on the convenience of their simplifying assumptions. The space of probable sequences is constrained by assuming only pairwise dependencies over hidden states. Pairwise dependencies also allow for a class of efficient inference algorithms whose critical steps build on the Forward-Backward algorithm [44].

This work describes an HMM variant over a set of sequences, with one transition matrix per sequence, as a novel alternative for handling sequence data. After training, the per-sequence transition matrices of the HMM variant are used as fixed-length vector representations for each associated sequence. There are a number of ways for understanding

how the HMM variant represents sequence data, and we connect these ways of understanding to both traditional explanations using simple Hidden Markov Models and more recent interpretations arising from topic models [46]. We then describe three methods to infer the parameters of our HMM variant, explore connections between these methods, and provide rationale for the classification behavior of the parameters derived through each.

We perform a comprehensive set of experiments, evaluating the performance of our method in conjunction with support vector machines, to classify synthetically generated data and sequences of amino acids into structural classes (fold recognition and remote homology detection problem [47]).

The combination of these methods, their interpretations, and their connections to prior work constitutes a new twist on classic ways of understanding sequence data that we believe is valuable to anyone approaching a sequence classification task and constitutes a significant contribution.

3.1 Problem Statement

Given a set of sequences, we would like to find a set of fixed-length vectors, A, that, when used as input to a function $f(A)$, maximize the probability of reconstructing the original set of sequences. Under our scheme, $f(A)$ is a Hidden Markov Model variant with one transition matrix, A_n, assigned to each sequence, a single emissions matrix, B, and a single start probability vector, a, for the entire set of sequences. By maximizing the likelihood of the set of sequences under the HMM variant model, we will also find the set of transition matrices that best represent our set of sequences. We further postulate that this maximum likelihood representation will achieve good classification results if each sequence is later associated with a meaningful label.

3.1.1 Model Description

We define a Hidden Markov Model variant that represents a set of sequences. Each sequence is associated with a separate transition matrix, while the emission matrix and initial state

Parameter	Description
N	the number of sequences
T_n	the length of sequence n
K	the number of hidden symbols
M	the number of observed symbols
a_i	start state probabilities, where i is the value of the first hidden state
A_{nij}	transition probabilities, where n indicates the sequence, i the originating hidden state, and j the destination hidden state
B_{im}	emission probabilities, where i indicates the hidden state, and m the observed symbol associated with the hidden state
z_{nt}	the hidden state at position t in sequence n
x_{nt}	the observed state at position t in sequence n

Table 3.1: HMM Variant model parameters

transition vector are shared across all sequences. We use the value of each transition matrix as a fixed-length representation of the sequence. We define the parameters and notation for the model in Table 3.1.

The probability of the model is shown below:

$$p(x, z|a, A, B) = \prod_{n=1}^{N} \left(a_{z_{n1}} \left(\prod_{t=2}^{T_n} \mathbf{A_{nz_{nt-1}z_{nt}}} \right) \left(\prod_{t=1}^{T_n} B_{z_{nt}x_{nt}} \right) \right) \quad (3.1)$$

This differs from the standard hidden Markov model only in the addition of a transition matrix for each sequence. The probability of a set of sequences under the standard HMM is shown below (differences highlighted in bold):

$$p(x, z|a, A, B) = \prod_{n=1}^{N} \left(a_{z_{n1}} \left(\prod_{t=2}^{T_n} \mathbf{A_{z_{nt-1}z_{nt}}} \right) \left(\prod_{t=1}^{T_n} B_{z_{nt}x_{nt}} \right) \right) \quad (3.2)$$

To regularize the model, we further augment the basic HMM by placing Dirichlet priors on a, each row of A, and each row of B. The prior parameters are the uniform Dirichlet parameters γ, α, and β for a, A, and B respectively. The probability of the model with priors is shown below, where the prior probabilities are the first three terms in the product below and take the form $Dir(x; a, K) = \frac{\Gamma(Ka)}{\Gamma(a)^K} \prod_i x_i^{a-1}$:

$$p(x, z, a, A, B|\alpha, \beta, \gamma) = \left(\frac{\Gamma(K\gamma)}{\Gamma(\gamma)^K} \prod_i a_i^{\gamma-1}\right) \left(\prod_{ni} \frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \prod_j A_{nij}^{\alpha-1}\right) \left(\prod_i \frac{\Gamma(M\beta)}{\Gamma(\beta)^M} \prod_m B_{im}^{\beta-1}\right)$$
$$\prod_{n=1}^{N} \left(a_{z_{n1}} \left(\prod_{t=2}^{T_n} A_{nz_{nt-1}z_{nt}}\right) \left(\prod_{t=1}^{T_n} B_{z_{nt}x_{nt}}\right)\right) \quad (3.3)$$

One potential difficulty that could be expected in classifying simple HMMs by transition matrix is that the probability of a sequence under an HMM does not change under a permutation of the hidden states. This problem is avoided when we force each sequence to share an emissions matrix, which locks the meaning of each transition matrix row to a particular emission distribution. If the emission matrix were not shared, then two HMMs with permuted hidden states could have transition matrices that with large Euclidean distances. For instance, the following HMMs have different transition matrices, but the probability of an observed sequence is the same under each.

$$HMM_1 : A_1 = \begin{bmatrix} .9 & .1 \\ .9 & .1 \end{bmatrix}, \; B_1 = \begin{bmatrix} .9 & .1 \\ .1 & .9 \end{bmatrix}$$

$$HMM_2 : A_2 = \begin{bmatrix} .1 & .9 \\ .1 & .9 \end{bmatrix}, \; B_2 = \begin{bmatrix} .1 & .9 \\ .9 & .1 \end{bmatrix}$$

However, a Euclidean distance between their two transition matrices, A_1 and A_2 is large.

3.1.2 A simple example

To gain an intuitive understanding of how our scheme operates, consider the following scenario. Assume that instead of learning the parameters of our emissions matrix, B, we fix B so that row m describes a multinomial distribution with probability of 1 of emitting the m^{th} observed symbol and zero probability of emitting any other symbol. For instance, if we have three possible emission symbols, $[a, b, c]$, then $M = 3$, $K = 3$, and B is set to I_3:

$$B = \begin{array}{c|ccc} & x_a & x_b & x_c \\ \hline z_1 & 1 & 0 & 0 \\ z_2 & 0 & 1 & 0 \\ z_3 & 0 & 0 & 1 \end{array}$$

Because there is a deterministic correspondence between observed and hidden states, the hidden states are effectively observed, and A_{nij} is simply $P(x_{nt} = j | x_{nt-1} = i)$, which can be estimated by taking the normalized count of the number of pairs of symbols ij in the sequence: $\{\#t : x_{nt-1} = i, x_{nt} = j,\ t > 1\}$ divided by the total number of x_{nt} with the value i,$\{\#t : x_{nt} = i,\ t < T_n\}$.

Our HMM variant is similar to this simplified scheme, but the number of hidden states, K, is set beforehand, and an inference algorithm is used to jointly optimize the transition and emission matrices to capture the best representation of the set of input sequences.

3.1.3 Another interpretation

Earlier we noted that we can interpret each transition matrix A_n as the argument to a function that allows us to reconstruct the sequence x_n with minimum error.

Using the basic HMM, we can describe a method to perform this reconstruction: first,

an initial hidden state, z_{n1}, is sampled from a. Next, for every z_{nt} with $1 < t \leq T_n$, z_{nt} is sampled from a multinomial with parameters $A_{nz_{nt-1}}$. Finally, each observed sequence element, x_{nt} is sampled from a multinomial with parameters $B_{z_{nt}}$.

Given a single sequence, we can create a standard HMM, with a probability of 1 of regenerating the source sequence by setting the number of hidden states equal to the length of the sequence, $K = T$. Next, at each hidden state, k, we set the probability of a transition to the hidden state $k+1$ to one and transitions to any other hidden states to zero. Finally, we set the matrix B so that at hidden state $k = t$ we emit the symbol x_k.

Taking this idea further, we can see intuitively how the size of A relates to some measure of information in the sequence. To best illustrate this, if we take a sequence that consists of two repeated sections, we would need only $K = \frac{T}{2}$ states to reconstruct it without error (with n repeats we would need $K = \frac{T}{n}$ states) because we can set state K to deterministically transition to state 1.

If the set of sequences has varying amounts of information per sequence, then, for small values of K, our scheme will be able to reconstruct low-information sequences with small error but will have a high error rate when reconstructing high-information sequences. If we choose a large K, then our scheme will be able to reconstruct high-information sequences well, but for low-information sequences some values of A will be meaningless.

3.2 Background

HMMs have a rich history in sequence classification and clustering [44, 45]. Smyth introduces a mixture of HMMs in [48] and presents an initialization technique that is similar to our model in that an individual HMM is learned for each sequence, but differs from our model in that the emission matrices are not shared between HMMs. In [48], these initial N models are used to compute the set of all pairwise distances between sequences, defined as the symmetrized log likelihood of each element of the pair under the other's respective model. Clusters are then computed from this distance matrix, which are used to initialize a

set of $K < N$ HMMs where each sequence is associated with one of K labels. Smyth notes that while the log probability of a sequence under an HMM is an intuitive distance measure between sequences, it is not intuitive how the parameters of the model are meaningful in terms of defining a distance between sequences. In this research, we demonstrate experimentally that the transition matrix of our model is useful for sequence classification when combined with standard distance metrics and tools.

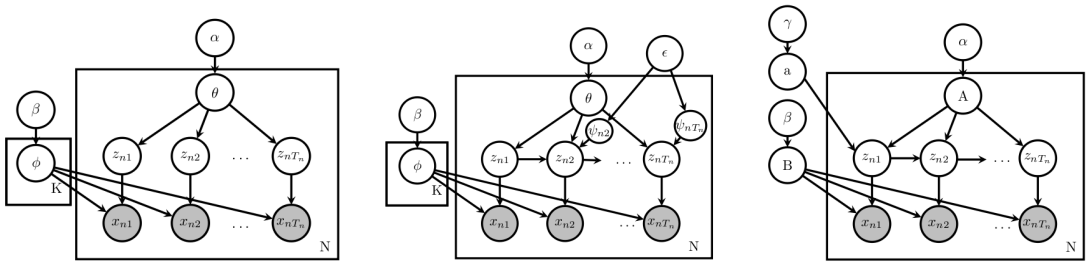
3.2.1 Topic Models

Simpler precursors of LDA [46] and pLSI [49], which represent an entire corpus of documents with a single topic distribution vector, are very similar to the basic Hidden Markov Model, which assigns a single transition matrix to the entire set of sequences that are being modeled. To extend the HMM to a pLSI analogue, all that is needed is to split the single transition matrix into a per-sequence transition matrix. To extend this model to an LDA analogue, we must go a step further and attach Dirichlet priors to the transition matrices.

Inference of the LDA model (Figure 3.1a) on a corpus of documents learns a matrix of document-topic probabilities. A row of this matrix, sometimes described as a mixed-membership vector, can be viewed as a measurement of how a given document is composed from the set of topics. In our HMM variant (Figure 3.1c), a single transition matrix, A_n, can be thought of as the analogue to a document-topic matrix row and can be viewed as a measurement of how a sequence is composed of pairs of adjacent symbols.

More recent topic models contain significant similarities to our HMM variant. Both the Hidden Topic Markov Model (HTMM) (Figure 3.1b) [50] and Conditional Topic Random Fields (CTRF) [51] are similar to our HMM variant in that they add pairwise dependencies between hidden topics to the LDA model. The key difference between our HMM variant and the HTMM lie in the HTMM's explicit modeling of text. Like LDA, the HTMM assigns one topic composition vector to each document. Dependencies between topics of adjacent words are modeled using a separate binomial parameter and associated set of indicator hidden variables per topic, rather than using a transition matrix like the HMM variant.

This binomial parameter has the effect of restricting the possible transitions between topics according to a per-sentence composition. For the CTRF, hidden states (topics) are modeled using a conditional random field .



(a) (b) (c)

Figure 3.1: Plate diagrams of the (a) LDA model, expanded to show each word separately, the (b) Hidden Topic Markov Model, and the (c) HMM variant.

3.3 Learning the model parameters

We experimented with three methods for learning parameters of our model: an MAP method based on the Baum-Welch algorithm, a Markov Chain Monte Carlo method based on the forward-backtrack sampler, and a mean field variational method.

3.3.1 Baum-Welch

A well-known method for learning HMM model parameters is the Baum-Welch algorithm. The Baum-Welch algorithm is an expectation maximization algorithm for the standard HMM model, and the basic algorithm is easily modified to learn the multiple transition matrices of our variant. The parameter updates shown below converges to a maximum a priori (MAP) estimate of $p(z, a, A, B|x, \gamma, \alpha, \beta)$ [44]:

$$a_i \propto \sum_n f_{ni}(1) b_{ni}(1) + \gamma - 1 \tag{3.4}$$

$$A_{nij}^{(new)} \propto \sum_{t=2}^{T_n} f_{ni}(t-1) A_{nij} B_{jx_t} b_{nj}(t) + \alpha - 1 \tag{3.5}$$

$$B_{im} \propto \sum_n \sum_{t:x_t=m} f_{ni}(t) b_{nj}(t) + \beta - 1 \tag{3.6}$$

where f and b are the forward and backward recurrences defined below:

$$f_{ni}(t) = \begin{cases} \sum_j f_{nj}(t-1) A_{nji} B_{ix_t}, & t > 1 \\ a_i B_{ix_1}, & t = 1 \end{cases} \tag{3.7}$$

$$b_{ni}(t) = \begin{cases} \sum_j A_{nij} B_{jx_{t+1}} b_{nj}(t+1), & t < T_n \\ \frac{1}{K}, & t = T_n \end{cases} \tag{3.8}$$

The complexity of the Baum-Welch-like algorithm for our variant is identical to the complexity of Baum-Welch for the standard HMM. The update for A_{ij} in the original HMM involves summing over $\sum_n T_n$ terms, while the update for a single A_{nij} is a sum over T_n terms, making the total number of terms over all the A_n's in our variant, $\sum_n T_n$, which is the same as number the original algorithm.

3.3.2 Gibbs Sampling

Two Gibbs sampling schemes are commonly used to infer Hidden Markov Model parameters [52]. Unlike the Baum-Welch algorithm which returns a MAP estimate of the parameters, these sampling schemes allow the expectation of the parameters to be computed over the posterior distribution $p(z, a, A, B|x, \gamma, \alpha, \beta)$.

In the Direct Gibbs sampler (DG), hidden states and parameters are initially chosen at random, then new hidden states are sampled using the current set of parameters:

$$p(z_{ti}^{(new)}|z_{t-1}, z_{t+1}) \propto A_{z_{t-1}i} B_{ix_t} A_{iz_{t+1}} \tag{3.9}$$

In the Forward Backward sampler (FB), the initial settings and parameter updates are the same as the DG scheme, but the hidden states are sampled in order from T_n down to 1 using values from the forward recursion. Specifically, each hidden state z_{nt} is sampled given $z_{nt+1} = j$ from a multinomial with parameters

$$p(z_{nT_n}^{(new)}|x_{n1:T_n}) \propto f_{ni}(T_n) \tag{3.10}$$

$$\begin{aligned} p(z_{nt}^{(new)}|x_{n1:T_n}, z_{nt+1}^{(new)}) &= p(z_{nt}^{(new)}|x_{n1:t}, z_{nt+1}^{(new)}) \\ &\propto f_{ni}(t) A_{nij}, \qquad t < T_n \end{aligned} \tag{3.11}$$

In both algorithms, after the hidden states are sampled, parameters are sampled from Dirichlet conditional distributions, shown for A below, where $\mathbb{I}(\omega) = 1$ if ω is true and 0 otherwise:

SCOP Version	Filtering	Taxonomic type	# sequences	# categories	SVM classifier
1.67	25%	class	4995	7	multiclass
1.67	25%	fold	1127	25	multiclass
1.67	40%	fold	1653	27	multiclass
1.67	40%	superfamily	1044	37	multiclass
1.53	1e-25	fold	4352	23	one-versus-rest
1.53	1e-25	superfamily	4352	54	one-versus-rest

Table 3.2: Datasets used to evaluate the HMM variant's ability to classify protein sequences.

$$p(A_{nij}|z_n, \alpha) = Dir(\sum_{t=2}^{T_n} \mathbb{I}(z_{nt-1} = i)\mathbb{I}(z_{nt} = j) + \alpha) \tag{3.12}$$

The FB sampler has been shown to mix more quickly than the DG sampler, especially in cases where adjacent hidden states are highly correlated [52]. We therefore use the FB sampler in our implementation.

3.3.3 Variational Algorithm

Another approach for inference of the HMM variant parameters is through variational techniques. We employ a mean field variational algorithm that follows a similar pattern as EM. When the variational update steps are run until convergence, Kullback-Leibler divergence between the variational distribution, $q(z, a, A, B)$, and the model's conditional probability distribution, $p(z, a, A, B|x, \gamma, \alpha, \beta)$, is minimized. The transition matrices returned by the variational algorithm are the expectations of those matrices under the variational distribution. Thus, like the Gibbs sampling algorithm, the parameters returned by the variational algorithm approximate the expectations of the parameters under the conditional distribution.

Our mean field variational approximation is shown below:

$$q(z, a, A, B) = q(a) \prod_{n=1}^{N} \prod_{i=1}^{K} q(A_{ni}) \prod_{i=1}^{K} q(B_i) \prod_{nt} q(z_{nt}) \tag{3.13}$$

$$= \left(\frac{\Gamma(\sum_i \tilde{\gamma}_i)}{\prod_i \Gamma(\tilde{\gamma}_i)} \prod_i a_i^{\tilde{\gamma}_i - 1} \right) \left(\prod_{ni} \frac{\Gamma(\sum_j \tilde{\alpha}_{nij})}{\prod_j \Gamma(\tilde{\alpha}_{nij})} \prod_j A_{nij}^{\tilde{\alpha}_{nij} - 1} \right)$$

$$\left(\prod_i \frac{\Gamma(\sum_m \tilde{\beta}_{im})}{\prod_m \Gamma(\tilde{\beta}_{im})} \prod_m B_{im}^{\tilde{\beta}_{im} - 1} \right) \prod_{nti} h_{nti}^{z_{nti}}$$

with variational parameters h_{nti}, which acts as an approximate mean for each z_{nti}, and $\tilde{\alpha}_{nij}$, $\tilde{\beta}_{im}$, and $\tilde{\gamma}_i$, which can be thought of as Dirichlet parameters approximating α, β, and γ.

When we maximize the variational free energy with respect to the variational parameters, we obtain the following update equations, where $\Psi(x) = \frac{d \log \Gamma(x)}{dx}$:

$$\tilde{\alpha}_{nij} = \sum_t h_{nt-1i} h_{ntj} + \alpha \tag{3.14}$$

$$\tilde{\beta}_{im} = \sum_{nt:x_t=m} h_{nti} + \beta \tag{3.15}$$

$$\tilde{\gamma}_i = \sum_n h_{n1i} + \gamma \tag{3.16}$$

$$h_{nti} \propto \begin{cases} \exp\Big(\Psi(\tilde{\gamma}_i) - \Psi(\sum_{i'} \tilde{\gamma}_{i'}) \\ \qquad + \sum_j h_{n2j} \Big(\Psi(\tilde{\alpha}_{nij}) - \Psi(\sum_{j'} \tilde{\alpha}_{nij})\Big) \\ \qquad + \Big(\Psi(\tilde{\beta}_{ix_{n1}}) - \Psi(\sum_m \tilde{\beta}_{im})\Big)\Big), \\ t = 1 \\ \exp\Big(\sum_{i'} h_{nt-1i'} \Big(\Psi(\tilde{\alpha}_{ni'i}) - \Psi(\sum_j \tilde{\alpha}_{ni'j})\Big) \\ \qquad + \sum_j h_{nt+1j} \Big(\Psi(\tilde{\alpha}_{nij}) - \Psi(\sum_{j'} \tilde{\alpha}_{nij'})\Big) \\ \qquad + \Big(\Psi(\tilde{\beta}_{ix_{nt}}) - \Psi(\sum_m \tilde{\beta}_{im})\Big)\Big), \\ 1 < t < T_n \\ \exp\Big(\sum_{i'} h_{nt-1i'} \Big(\Psi(\tilde{\alpha}_{ni'i}) - \Psi(\sum_j \tilde{\alpha}_{ni'j})\Big) \\ \qquad + \Big(\Psi(\tilde{\beta}_{ix_{nT_n-1}}) - \Psi(\sum_m \tilde{\beta}_{im})\Big)\Big), \\ t = T_n \end{cases} \tag{3.17}$$

Notice that the update for h_{nti} depends only on the adjacent h's, h_{nt-1i} and h_{nt+1i} as well as the expectations of the transition probabilities from the adjacent h's and the expectation of the emission probabilities from the current h_{nti}. This mean field algorithm can therefore be understood as an equivalent of the Direct Gibbs sampling method except that at subsequent time steps interactions occur between variational approximations rather than through the sampled values of z. A complete derivation of the variational algorithm is included in Appendix A.

Other authors have performed variational inference over hidden Markov models. Most notably, Ghahramani et. al. derive variational updates to iteratively optimize parameters of factorial HMMs [53] and switching state space models [54]. MacKay derives a variational ensemble for HMM posterior parameters [55]. Our variational algorithm is most similar to the mean field variational algorithm described by Ghahramani et. al. [53].

3.4 Experimental Setup

To evaluate our fixed-length representation scheme, for each classification experiment, we created three sets of fixed-length representations per trial over ten trials by running each of the three inference algorithms: (i) Baum-Welch, (ii) Gibbs Sampling, and (iii) the mean field variational algorithm, on the entire set of input data. We varied the number of hidden states from 5 to 20 in increments of 5 ($K = \{5, 10, 15, 20\}$). This procedure created a total of 120 ($3 \times 10 \times 4$) fixed-length representations for each dataset.

The fixed-length vector data was then used as input to a support vector machine (SVM) classifier [1]. We used the SVM to either perform either multiway classification on the dataset under the Crammer-Singer [57] construction or the one-versus-rest approach, where a binary classifier was trained for each of the classes. No attempt was made to optimize the SVM parameters in any of the experiments.

We compare classification results from our model with results from the Spectrum(2) kernel for all experiments. The Spectrum(ℓ) kernel is a simple string kernel whose vector representation is the set of counts of substrings of of observed symbols length ℓ in a given string [43]. For the one-versus rest experiments, we compare our results to more sophisticated biologically sensitive kernels for protein classification, described in Rangwala et. al [1].

3.4.1 Protein Datasets

To evaluate our representation, we ran sets of protein classification experiments on the three top levels of the SCOP taxonomy, class, fold, and superfamily. Table 3.2 provides a detailed description of the different SCOP-derived protein sequence classification datasets, that were obtained from previous studies [47, 58]. Specifically, the datasets were derived from either the SCOP 1.67 or the SCOP 1.53 versions and filtered at 25% and 40% pairwise sequence identities or with E-values [59] of less than 10^{-25}.

We partitioned the data into a single test and training set for each category. At the class

[1] We used SVM-light and SVM-struct for classification (http://www.cs.cornell.edu/People/tj/svm_light/) [56].

level, the original dataset was split randomly in to training and test sets. To eliminate high levels of similarity between sequences that could lead to trivially good classification results, we imposed certain constraints on the training/test set partitioning for classification in the fold and superfamily experiments. For the fold level classification problem, the training sets were partitioned so that no examples that shared the fold and superfamily labels were included in both the training and test sets. Similarly, for the superfamily level classification problem (referred to as the remote homology detection problem [1, 43]), no examples that shared the superfamily and family levels were included in both the training and test sets.

3.4.2 Synthetic Datasets

As a basic test of concept, we constructed synthetic datasets by drawing samples from two HMMs with transition matrices

$$A_1 = \begin{bmatrix} .6 & .4 \\ .4 & .6 \end{bmatrix}, \; A_2 = \begin{bmatrix} .4 & .6 \\ .6 & .4 \end{bmatrix}$$

with a discrete emissions matrix where each state's output is an eight state discretized version of a univariate normal distribution with means of 3 and 5 respectively. These HMMs were constructed to be discretized versions of the HMMs used to generate synthetic data in a previous study [48].

We ran experiments on a dataset consisting of 500 samples from each HMM with the length of each sequence Poisson distributed with a mean of 100. We then randomly partitioned the dataset into 20% test and 80% training samples and used the protocol described above to classify training samples by their generating HMM. The experiment was run 10 times on separately generated datasets.

For the synthetic experiments, we included two additional kernels. The first, a Kullback-Leibler Divergence Kernel, [60], was used to attempt to answer the question whether a "natural" metric over the space of transition matrices produces better results than the standard

dot product. Because each transition matrix row resides in a K-simplex, we considered the possibility that a symmetrized DKL measurement between rows of different matrices would allow for better comparisons.

We also tested a representation, that we call the gradient kernel, where the feature vector for each sequence was given by the gradient of the log probability of a sequence with respect to the transition matrix associated with that sequence. This representation is similar to the Fisher kernel [61] but differs from the standard Fisher kernel in that the scheme does not take advantage of information about the classes associated with sequences in the training set. A more complete version of the Fisher kernel for the HMM variant would take into account sequence labels in the training set and would require taking the gradient of a sequence under the portion of the HMM variant model associated with each positive and negative training example, a computationally intensive task.

3.4.3 Evaluation Metrics

We evaluated each classification experiment by computing the area under the ROC curve (AUC), a plot of the true positive rate against the false positive rate, constructed by adjusting the SVM's intercept parameter. We were worried about variance over different Baum-Welch runs due to convergence of the algorithm to different local optima. To mitigate this concern, we ran both the Baum-Welch algorithm and the other inference algorithms, for consistency, 10 separate times on each dataset. The results presented for each inference method are averages over individual results of the 10 trials across the different classes.

3.5 Results and Discussion

Table 3.3 shows a comparison of average AUC scores across the inference algorithms on the class, fold, and superfamily levels of the SCOP hierarchy using the multiclass SVM. Although the AUC scores are close for each algorithm in most cases, the Gibbs sampling algorithm outperforms the other algorithms the majority of the time.

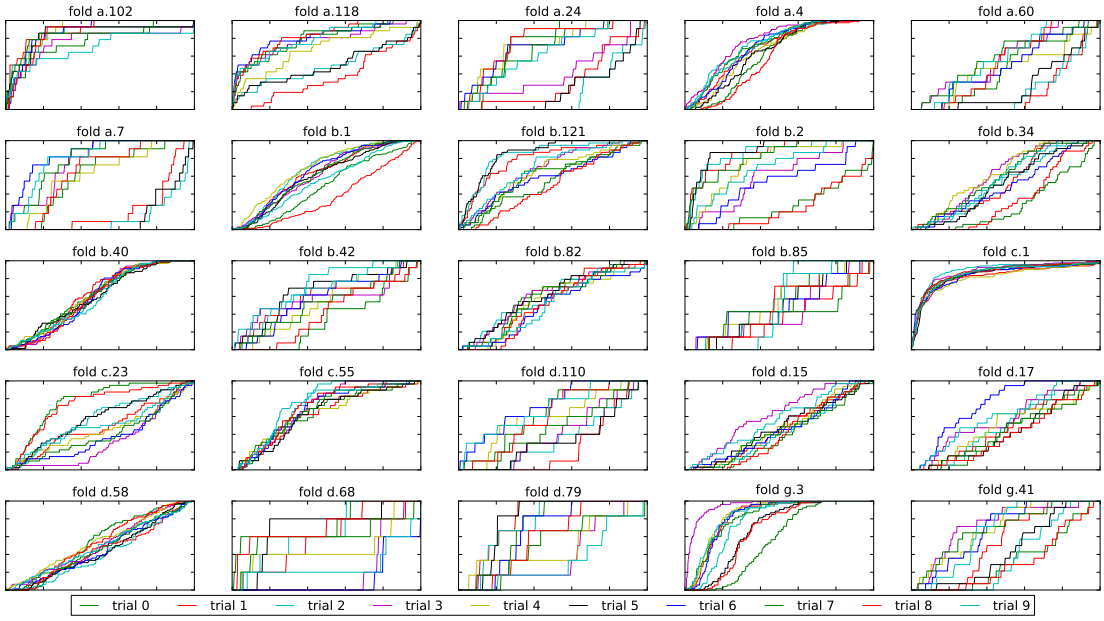


Figure 3.2: A comparison of AUC plots for the Baum Welch algorithm for $K = 10$ on the SCOP 1.67, 25% fold recognition dataset (25 classes) over a set of 10 parameters learned through randomly initialized Baum-Welch runs. From the plots, we can see that the variance of the classification of individual results can be high, especially for the classes with a small number of examples. However, there was a comparatively smaller amount of variation ($\sim 3\%$) in the average AUC score over all classes.

Table 3.4 shows a comparison of results over the inference algorithms for the one-versus-rest superfamily classification experiment on the SCOP 1.53 dataset. Similarly to the multiclass experiments using the linear kernel, the Gibbs sampling algorithm outperforms the other inference methods in the one-versus-rest experiments. The variational algorithm shows the largest improvement in AUC when switching from linear to Gaussian kernels, ranging from 6% to 30%.

The results over multiple trials also revealed an interesting characteristic of the data. Although the standard deviation of the AUC score between categories within a single trial is high, with an average of 0.17 over 10 trials, the standard deviation of the average AUC score over all categories was low (0.03 averaged over 10 trials), indicating that a trade-off is occurring. If the AUC on a single taxonomic category is high in one trial, then it will generally be offset by a low AUC score on another taxonomic category and vice versa.

Figure 3.2 graphically illustrates this trade-off, showing an overlay of AUC curves for fold classification over 10 Baum-Welch transition matrix representations.

3.5.1 Synthetic Results

Table 3.5 compares average AUC scores from the synthetic data classification experiments between each inference algorithm and the Spectrum(2) kernel. The table also shows the effects of a variety of additional kernels over the basic description vectors. Notably, the Baum-Welch algorithm outperforms the others inference algorithms under both the linear kernel and all of the external kernels. No HMM variant formulation performs better on the synthetic data than the Spectrum(2) vector under a linear kernel. However, the spectrum kernel results degrade across the non-linear kernels, while the HMM variant results are consistent.

We compared several kernel functions over the HMM variant sequence representation vectors. Although the DKL kernel appears to perform better than the linear kernel, it does not outperform the Gaussian. Similarly, the gradient kernel does not consistently perform better than the Gaussian. The Gibbs Sampling algorithm performs comparatively better under the Fisher kernel, a trend not reflected in the variational algorithm's performance.

3.5.2 Analysis of Inference Algorithms

Different inference algorithms for our model produced significantly different AUC scores in both the protein classification experiments (Table 3.3) and in the experiments on synthetic data (Table 3.5).

A potential explanation of these differences lies in the targets of each algorithm's objective function. While the Baum-Welch algorithm returns MAP parameters of the model, both the Gibbs sampling method and the variational algorithm produce expectations of the parameters. These different convergence points would seem to make a much larger difference in the results if the posterior distribution of the transition matrix is highly multimodal. Under a multimodal distribution, we expect the Baum-Welch algorithm to converge to a local

Class Level, 7 categories
SCOP 1.67, 25%

Alg/K	5	10	15	**20**
Baum Welch	**0.61**	0.64	0.65	0.63
Gibbs Sampling	**0.61**	**0.65**	**0.66**	**0.68**
Variational	0.60	0.63	0.60	0.60

Fold Level, 25 categories
SCOP 1.67, 25%

Alg/K	5	10	15	**20**
Baum Welch	**0.56**	**0.59**	**0.59**	0.58
Gibbs Sampling	**0.56**	0.58	**0.59**	**0.61**
Variational	0.54	0.57	**0.59**	0.58

Fold Level, 27 categories
SCOP 1.67, 40%

Alg/K	5	10	15	**20**
Baum Welch	0.58	0.60	0.55	0.57
Gibbs Sampling	**0.60**	**0.63**	**0.65**	**0.67**
Variational	0.58	0.58	0.59	0.59

Superfamily, 37 categories
SCOP 1.67, 40%

Alg/K	5	10	**15**	20
Baum Welch	**0.59**	**0.63**	0.63	0.61
Gibbs Sampling	**0.59**	0.62	**0.64**	**0.63**
Variational	**0.59**	0.57	0.58	0.57

Table 3.3: AUC results from all of the multi-class SVM experiments are displayed. The best performing algorithm, the best performing setting of K, and the best combination of K and algorithm is marked in bold. The Gibbs-Sampling-derived representation most frequently returned the most accurate level of classification on the majority of the datasets.

maximum at one of these modes. In contrast, the expectation computed under both the Gibbs sampling and the variational algorithm may lie at a point in the parameter space described by the weighted center of mass of these modes. Different topologies of the posterior distribution will clearly have an effect on how well the expected value of the parameters compares to the MAP parameters. If many maxima occur in a single region, then the expected parameter may perform well. However, if a small number of maxima are highly separated, then the MAP solution, which will likely reside at one of the maxima, will probably contain

Linear Kernel

Evaluation Metric	AUC			
Algorithm/K	5	10	**15**	**20**
Baum Welch	0.58	0.55	0.52	0.57
Gibbs Sampling	**0.64**	**0.67**	**0.69**	**0.69**
Variational	0.63	0.59	0.54	0.58

Gaussian Kernel

Evaluation Metric	AUC			
Algorithm/K	5	10	**15**	20
Baum Welch	0.61	0.60	0.58	0.59
Gibbs Sampling	0.63	**0.63**	0.63	0.63
Variational	**0.67**	0.60	**0.70**	**0.68**

Table 3.4: AUC results on the SCOP 1.53 Fold dataset over a selected set of 23 superfamilies using Gaussian and linear kernels in one-versus-rest SVM classification.

AUC

Alg/Kernel	Linear	Gaussian	DKL	Gradient
Baum Welch	0.894	**0.926**	**0.919**	**0.860**
Gibbs	0.606	0.537	0.563	0.828
Variational	0.562	0.562	0.578	0.546
Spectrum	**0.997**	0.472	0.548	N/A

Table 3.5: AUC results from all synthetic data experiments averaged over 10 trials. For each HMM variant, the number of hidden states is 2. Counts of substrings of length 2 were used to construct the spectrum kernel. The best performing entry is marked in bold.

more information about the sequence than the expected parameter value.

To test this hypothesis, we constructed histograms of transition matrices using 1000 samples from the Gibbs sampling algorithm. Figure 3.3 shows a histogram of the 2×2 transition matrix associated with the first sequence from the synthetic data and is clearly bimodal with the modes at opposite ends of the distribution for each matrix entry. Similarly, Figure 3.4 shows a histogram of the 5×5 transition matrix associated with the first sequence from the SCOP 1.67, 25% Astral filtering dataset. This histogram exhibits much less multimodality than the histogram from the synthetic experiments. Taken together, the histograms, which

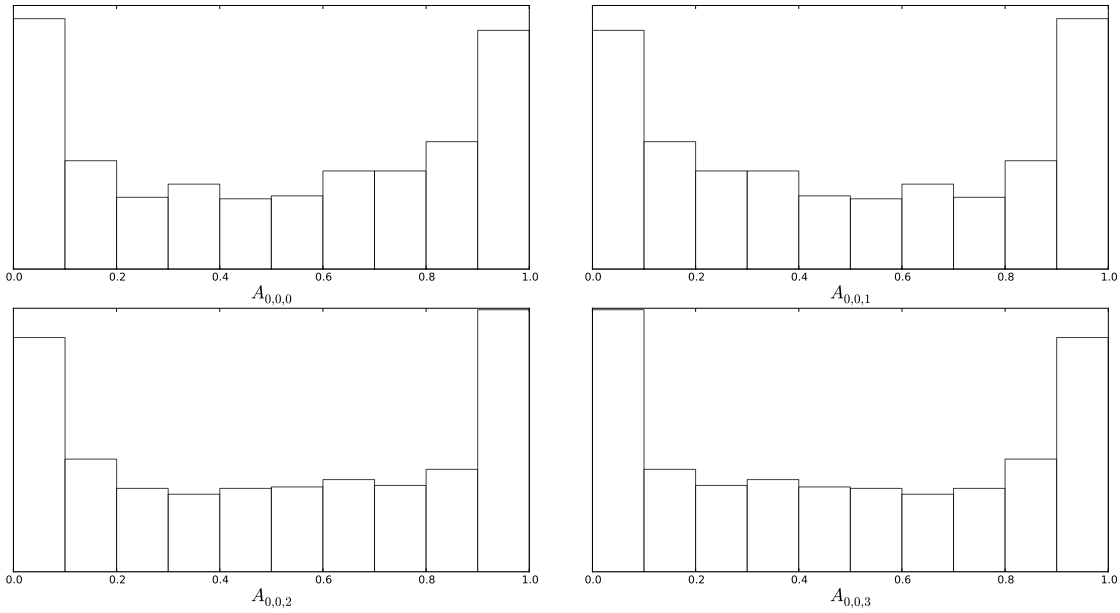


Figure 3.3: Histogram of the transition matrix associated with the first sequence of the synthetic dataset. The matrix entries have two modes at either end of the probability distribution.

approximate the posterior distribution of the transition matrices, illustrate how the multimodality seen in the posteriors of the synthetic data but not in posteriors of the the protein data likely causes the Gibbs sampling and variational transition matrices to perform poorly for classification.

To further analyze the results on the synthetic data, we observe the emission matrices learned by each inference method. There are clearly observable differences between the emission matrices returned from Baum-Welch and from both the Gibbs sampling and variational algorithm. Although all inferred emissions matrices exhibit some amount of bimodality, the degree of bimodality in the Baum-Welch emissions matrix is much less than the other algorithms (see Figure 3.5), allowing hidden states, and thus transitions between hidden states, to be more easily distinguished.

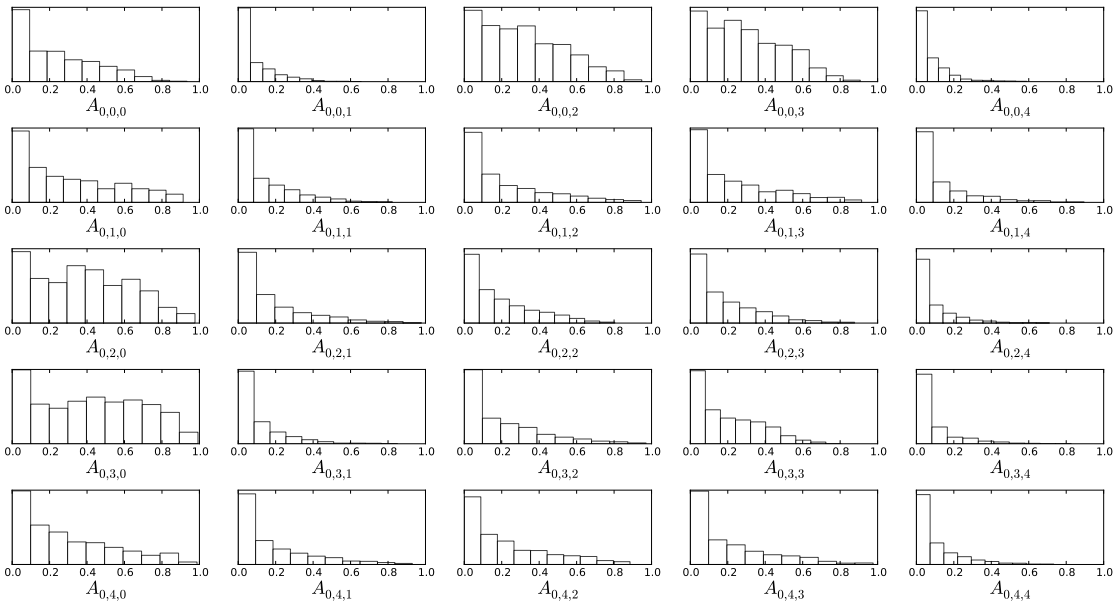


Figure 3.4: Histogram of the transition matrix associated with the first sequence of the SCOP 1.67 dataset with 25% Astral filtering. Modes of the distribution are more evenly distributed compared to the synthetic data transition matrices.

The gap in performance between the Gibbs sampling algorithm and the variational algorithm in the protein classification experiments is not as surprising as the different between the Baum-Welch algorithm and the rest. Both the Gibbs sampling algorithm and the variational algorithm compute expectations of the parameters under an approximate posterior distribution, but each uses a different method to construct this approximation. The variational algorithm will be less likely to converge to a good approximation of the marginal distribution because the mean field variational approximation necessarily does away with the direct coupling between adjacent hidden states characteristic of the HMM.

3.5.3 Comparison with Existing Protein Classification Methods

Tables 3.5, 3.6 show a comparison between the HMM variant and common classification methods for the synthetic and multiclass experiments respectively. Table 3.7 shows a separate experiment conducted on the one-versus rest classification problem over SCOP superfamilies. This problem is known as the remote homology detection problem and many previous efforts at protein sequence classification ran experiments on the same dataset [12], allowing us to compare our method to results from the literature. Results from the HMM Variant on the remote homology detection problem were averaged over five runs of the Gibbs sampler using 20 hidden states. Feature vectors from the Gibbs sampler were normalized to a magnitude of one.

AUC scores indicate that our scheme produces a representation that is roughly equivalent in power to the Spectrum kernel for multiclass protein classification but is outperformed by the Spectrum kernel for synthetic data classification. For the remote homology detection problem, the HMM Variant outperforms the Spectrum Kernel and the Fisher Kernel but is outperformed by the Mismatch kernel and the semi-supervised SW-PSSM kernel (Table 3.7). In defense of the HMM variant, the size of the vector representation produced by the the Mismatch and SW-PSSM kernels is significantly larger than the typical representations produced by our HMM variant. The Mismatch(5,1) kernel, used for SCOP 1.53 superfamily classification, is similar to the Spectrum(5) kernel but also counts substrings of length 5 that differ by one amino acid residue from those found in an observed sequence. The size of the vector representation associated with this kernel can be up to 20^5 and is not sparse. This value is large compared to the largest vector representation in our experiments, which is 400 for the HMM variant with 20 hidden states. Similarly, in the synthetic data classification task, the HMM variant uses a vector representation of length 4, while the Spectrum kernel vector representation is of length 64. Even though the vector representation associated with these kernels does not need to be explicitly computed, the size of the representation itself is an indication of the relative amounts of information used by these methods compared to the HMM variant.

Dataset/Kernel	HMM Variant	Spectrum
Class	0.68	0.66
Fold (25 Categories)	0.61	0.62
Fold (27 Categories)	0.67	0.67
Superfamily	0.69	0.64

Table 3.6: A comparison of results between the Spectrum kernel and the HMM variant under experiments using the multiclass SVM formulation. The HMM variant scores are the best performing from Tables 3.3 and 3.4.

Algorithm	AUC
Spectrum(2) [43]	0.712
Fisher [61]	0.773
HMM Variant	0.815
Mismatch(5,1) [62]	0.870
SW-PSSM [1]	0.982

Table 3.7: A selection of AUC scores using a variety of SVM kernels on the same dataset (see [1] for details on additional kernel methods). The HMM variant scores averages over five trials from representations derived from Gibbs sampler inference with 20 hidden states.

The HMM variant does not perform as well as profile HMM kernels like the SW-PSSM kernel, which are members of a family of kernels commonly used for protein taxonomy classification. Profile kernels, unlike the HMM variant, employ domain specific knowledge, such as carefully tuned position-specific scoring matrices, to aid classification. In contrast, only parameter that needs to be adjusted in the HMM variant is the value of K. In addition, profile kernels use "profiles" rather than protein sequences as input. These profiles are constructed from a labeled set protein sequences augmented by large databases of unlabeled sequences and are, in effect, semi-supervised methods.

3.5.4 Number of Hidden States

Table 3.3 not only shows how AUC scores are effected by the inference algorithm used but also how the performance of the algorithms changes as the number of hidden states change.

For many of the trials, a maximum AUC score occur at 10 or 15 hidden states (although notably, the Gibbs sampling results appear to increase as K increases), indicating that the best performing value of K in each experiment is probably near the range that we tested.

Although we do not present experimental results with larger numbers of hidden states, our experiments show that larger values of K tend to produce worse AUC results (The Baum-Welch Algorithm with K=75 results in a AUC score of .54 on the class-level dataset using a Gaussian kernel, which was superior to the linear kernel's performance.). This trend can be accounted for using arguments similar to those that explain overfitting. As the transition matrix size increases it seems likely that the number of "junk" hidden states, hidden states that contribute to generating observed symbols for only a small number of sequences in the dataset, would also increase. These meaningless hidden states would cause distances between transition matrices to be corrupted with noise.

3.5.5 Higher Order Models

In addition to the experiments described above, we also ran a limited number of experiments with higher order HMM variants, where transition matrices are defined between the group of hidden states $z_{nt-\ell}, \ldots, z_{nt-1}$ and z_{nt}, where ℓ is the order of the HMM. Classification performance under this set of models was also inferior to the first order results presented in Table 3.3. Decreasing performance with higher order results also occurs in the Spectrum and Mismatch kernels. The best performing Spectrum or Mismatch models on the multi-class protein taxonomy classification problems use substrings of length 5, and performance decreases using counts of larger substrings. This trend of reduced performance under higher order correlations can be explained by observing that the number of parameters in the higher order models increase by a factor of K when ℓ is incremented by 1 but our dataset size remains constant. Thus, as the order of the model increases, the uncertainty in the HMM variant transition matrices will also likely increase.

3.6 Conclusion

Our HMM variant is an extension of the standard HMM that assigns individual transition matrices to each sequence in a dataset. At least two intuitive interpretations describe the mechanisms that allow the HMM variant to capture meaningful information about a set of sequences. In addition, we describe three inference algorithms, two of which, a Baum-Welch-like algorithm and a Gibbs sampling algorithm are similar to standard methods used to infer HMM parameters. A third, the variational inference algorithm, is related to algorithms used for inference on topic models and more complex HMM extensions. We demonstrate, by comparing results on protein sequence classification using our method in conjunction with SVMs, that each of these algorithms infers transition matrices that capture useful characteristics of individual sequences. We further show, through an analysis of the transition matrices, what types of information are best captured by each the inference method. Although classification performance using our model does not outperform either highly optimized kernels or simple string kernels, our HMM variant facilitates new ways of understanding issues in sequence classification.

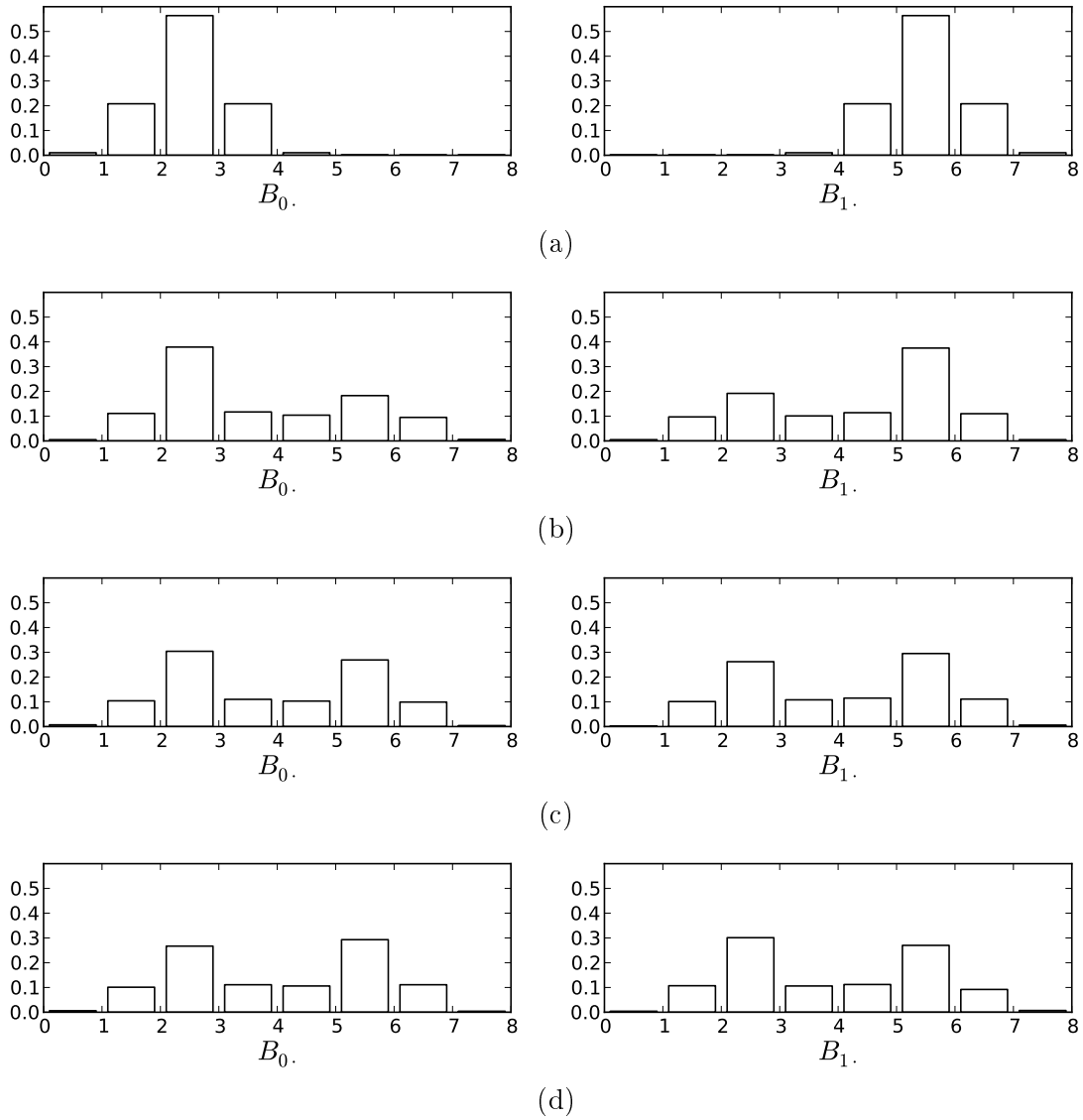


Figure 3.5: The original emission matrix used to generate the synthetic dataset (a) compared to typical emission matrices inferred the synthetic dataset using the Baum-Welch (b), Gibbs Sampling (c), and variational (d) algorithms. The charts in the first column show the first row of the matrix and the second column shows the second row of the matrix.

Chapter 4: The Infinite Profile Hidden Markov Model

4.1 Introduction

Profile Hidden Markov Models (pHMM) [9] have been useful in bioinformatics applications for analyzing amino acid, DNA, and RNA sequences. A pHMM is a left-to-right HMM that captures position-specific commonalities of amino acids within a group of related proteins. Profile HMMs are often constructed from multiple sequence alignments (MSAs) [63], which are arrangements of amino acid sequences used to identify regions of structural similarity or evolutionary relationships. Profile HMMs can be used in applications such as classifying sequences, querying sequence databases, and constructing multiple sequence alignments for additional sets of sequences. In this chapter, I improve on the basic Profile HMM by showing how the model can be extended to allow for an unbounded number of hidden states. This infinite extension also gives a new perspective on how the model operates, and this perspective can be used to develop fast approximate inference methods.

A core concept in constructing the Infinite-pHMM involves a transformation from the pHMM to an equivalent standard-HMM. This equivalent standard-HMM, which we call the Geometric Transition HMM, allows us to use beam methods [8, 64, 65], both traditional and novel, to run an approximate version of the forward-backward [2] algorithm. We show experimentally that these beam methods can lead to significant performance gains compared to the standard forward-backward algorithm in pHMMs.

The Infinite-pHMM and resulting beam methods constitutes an advance in understanding Profile HMMs. In addition, our novel beam method has the potential to increase the computational efficiency of bioinformatics software that currently relies heavily on running the forward-backward algorithm over Profile HMMs.

4.2 Background

The Hidden Markov Model (HMM) explains a sequence of observed symbols by postulating that a hidden state is responsible for generating each observation (Section 2.1). In an HMM, the sequence of hidden states exhibits the Markov property - that is, the value of each hidden state depends on past states only through the immediately preceding hidden state. A number of extensions have been proposed to the basic HMM. These include the Bayesian HMM [66], where prior probabilities are added to emission and transition probabilities, and non-parametric HMMs [67,68], which make no *a priori* assumptions about the number of hidden states.

There are two key requirements for a non-parametric HMM suitable for modeling amino acid sequences: (i) transition probabilities from each hidden state should be concentrated on a small subset of the entire set of possible hidden states; and (ii) transitions out of all hidden states should share a common, countably-infinite set of possible destination states.

A common model for prior probabilities on an unbounded number of latent components is the Dirichlet Process (DP) [69] distribution. Condition (i) can be satisfied by using a DP prior for transition probabilities of an HMM. However, if separate, uncoupled DP distributions are used for transitions out of each hidden state, then with probability 1, the set of possible destination states for transitions out of any two hidden states will be disjoint [67], violating condition (ii).

To ensure that multiple observed symbols share hidden states, Beal et. al. [67] use the Hierarchical Dirichlet Process (HDP) [70]. The HDP models a prior over emission distributions for an unbounded number of hidden states as a set of Dirichlet Processes with a shared base measure given by a root Dirichlet Process. Using this construction allows HDP-HMM to satisfy condition (ii), thereby allowing a sparse set of hidden states to model the observed sequence.

Another infinite HMM is the Stick Breaking HMM (SB-HMM) [68]. The SB-HMM assigns a separate stick-breaking prior [71] to each transition probability. These stick-breaking

Symbol	Description
Σ	the set of observed symbols; $\|\Sigma\|$ indicates the number of observed symbols.
T	the length of the observed sequence
t	indexes a position in a sequence
K	the number of base hidden states, where a "base hidden state" can be thought of as an index to the *Match* columns in a multiple sequence alignment
$k \in \{1 \dots K\}$	indexes a base hidden state in a Profile HMM
(s, k)	indexes the current base hidden state, k, as well as the current choice of $s = \{Match, Insert, Delete\}$, which are indicated in subscripts by the capital letters $\{M, I, D\}$
$x_t \in \{1 \dots \|\Sigma\|\}$	an observed symbol in sequence n at position t
$z_{1:\|z\|}$ or $\vec{z}$	A sequence of hidden states consisting of pairs ($s \in \{Match, Insert, Delete\}$, $k \in 1..K$). Each sequence of pairs forms a path through the DFA shown in Figure 2.3. Unlike the standard HMM, due to non-emitting *Delete* states, the number of hidden states used to generate an observed sequence in the pHMM could be larger than the length of the observed sequence, so we use $\|z\|$ to indicate the length of the sequence of hidden states.
$z_0 = (\text{Match}, 0)$	The first hidden state is defined as an observed start state, allowing us to encode the distribution of start transitions inside the main transition matrix.
$n_{(s,k),s'}$	The number of transitions from hidden state (s, k) to one of the following three states: $s' = M \implies (s', k') = (M, k+1)$; $s' = I \implies (s', k') = (I, k)$; $s' = D \implies (s', k') = (D, k+1)$ for a given sequence of hidden states $z_{1:\|z\|}$. Note that the choice of s' uniquely determines the value of k for at the destination state of the transition.
$n_{(s,k),m}$	The number of emissions of symbol m from hidden state (s, k) for a given sequence of hidden states $z_{1:\|z\|}$.
A	The transition probability matrix. $A_{(s,k),s'}$ indicates the probability of transitioning from state (s, k) to one of the following three states: $s' = M \implies (s', k') = (M, k+1)$; $s' = I \implies (s', k') = (I, k)$; $s' = D \implies (s', k') = (D, k+1)$. For instance, $A_{(M,1),D}$ represents the transition probability from hidden state $(M, 1)$ to hidden state $(D, 2)$.
B	The emission probability matrix. $B_{(s,k),m}$ indicates the probability of emitting symbol m from hidden state (s, k), where $s \in \{M, I\}$ can only be a *Match* or *Insert* state. For instance, $B_{(M,1),\text{"a"}}$ represents the probability of emitting an "a" from state $(M, 1)$.
$\alpha_{s,s'}$	Dirichlet prior parameters on transition probabilities. $A_{(s,k),s'}$ shares the prior parameter $\alpha_{s,s'}$ for all k.
$\beta_{s,m}$	Dirichlet prior parameters on emission probabilities. $B_{(s,k),m}$ shares the parameter $\beta_{s,m}$ for all k.

Table 4.1: Parameter definitions for the Profile Hidden Markov Model

priors are parametrized by an infinite length vector of concentration parameters, which are each given Gamma hyperpriors to make the number of prior parameters on the model finite. This careful combination of stick-breaking distributions allows each transition probability to be attached to a single set of emission parameters, also fulfilling condition (ii).

4.2.1 Profile HMMs

The Profile HMM (pHMM) (Section 2.1.1) is an HMM with specific restrictions on transitions and emissions. These restrictions allow the pHMM to capture an archetypal sequence or sequence fragment through the emission distributions of a portion of the model's hidden

states.

As with the Bayesian HMM, Dirichlet priors can be added to the transition and emission probabilities of the pHMM to produce a model with the following joint probability:

$$p(x_{1:T}, z_{1:|z|}, A, B|\alpha, \beta) = \prod_{(s,k),s'} \left(A_{(s,k),s'}\right)^{n_{(s,k),s'}} \prod_{(s\in\{M,I\},k),m} \left(B_{(s,k),m}\right)^{n_{(s,k),m}} \prod_{(s,k)} \mathrm{Dir}(A_{(s,k),\cdot}; \alpha_{s,s'}) \prod_{(s,k)} \mathrm{Dir}(B_{(s,k),\cdot}; \beta_{s,m}) \quad (4.1)$$

4.3 The Infinite Profile HMM

To construct an infinite pHMM, we consider the case where K is unbounded. In addition, we modify the pHMM so that the model can transition to the end state from any *Match* or *Insert* state (Figure 2.3). This modification allows the model to emit a sequence of observed symbols without needing to traverse infinitely many *Delete* states. We believe that we are the first to propose an infinite version of the pHMM.

Unlike the HDP-HMM and SB-HMM, we run into difficulties constructing a non-informative prior. An uninformative prior on pHMM transitions places a uniform distribution on all paths through the three-dimensional lattice. In the limit of infinite states, this leads to a Dirichlet prior that places all of its probability mass on *Delete* transitions.[1] Rather than using an uninformative prior in our model, we choose priors that make reasonable assumptions about how amino-acid sequences in a dataset should align for practical purposes. For example, it is unreasonable to expect that a model will use more than $K = \sum_n |x_n|$ states. With these assumptions in mind, we note that the major difference between the finite and infinite models is that in the finite model we encode our assumptions about the length of a sequence alignment in the value of K. In contrast, for the infinite model, our assumptions about the total length of the alignment are encoded in the Dirichlet priors on transition probabilities.

[1] To see this, note that each path that generates an observed sequence passes through a fixed number of *Match* and *Insert* states but can use an arbitrarily large number of *Delete* states.

This characteristic of the model can be seen visually in the upper right-hand heat map in Figure 4.4. In the heat map, areas of high intensity under uniform transition and emission probabilities at the start of inference do not reach the model's truncation threshold, K. The model should therefore be more flexible in the sense that a small number of hidden states should produce emitted symbols and the number of these states should reflect characteristics of the dataset rather than be determined by a preset model parameter.

Symbol	Description
$A^{(2S)}_{(s,k),k'}$	The probability of a sequence of pHMM transitions beginning at state (s,k) ending in a transition to either a Match or Insert state from the hidden state at k'.
$B^{(2S)}_{(s,k),m}$	Given than the transition to state (s,k) is not a delete state, the probability of making a single pHMM transition to (s,k) then emitting symbol m.
$n^{(2S)}_{(s,k),k'}$	Given a set of hidden states, $\vec{z}^{(2S)}$, the total number of transitions in the dataset between hidden states (s,k) and transitioning to either a Match or Insert state from the hidden state at k'.

Table 4.2: Additional parameter definitions for the 2S-HMM

To show the the Infinite pHMM is not degenerate, we must show that the model traverses a finite number of hidden states to generate a finite-length observed sequence. We establish non-degenerate behavior by showing that transitions in the Infinite pHMM are drawn from a stick breaking distribution. To do so, we consider an aggregation of the original pHMM transition probabilities, which we term the Two-Step HMM (2S-HMM). In this reformulation, we view each transition as a two-step process. In the first part of the process, we begin at hidden state k in the pHMM and draw a sequence of delete transitions from $A_{(k,s),:}, A_{(k+1,D),:}, A_{(k+2,D),:}, \ldots, A_{(k',D),:}$. When the sequence of deletes terminates, we draw either a *Match* or *Insert* transition from $A_{(k',D),:}$. This draw constitutes the second part of the process.

We can now rewrite these two-part transition probabilities in terms of the original pHMM transition probabilities. In contrast to the GT-HMM transformation, we merge the *Match*/*Insert* portion of the transition into a modified emission probability:

$$A^{(2S)}_{(s,k),k'} = \begin{cases} A_{(s,k),D}\left(\prod_{k''=k+1}^{k'-1} A_{(D,k''),D}\right)\left(1 - A_{(D,k'),D}\right) & k \leq k'-1 \quad (a) \\ 1 - A_{(s,k),D} & k = k'-1 \quad (b) \\ 0 & k > k' \quad (c) \end{cases} \tag{4.2}$$

$$B^{(2S)}_{(s,k),m} = \begin{cases} \frac{A_{(s,k-1),M}}{1-A_{(s,k-1),D}} B_{(M,k),m} & s = M \quad (a) \\ \frac{A_{(s,k),I}}{1-A_{(s,k),D}} B_{(I,k),m} & s = I \quad (b) \end{cases} \tag{4.3}$$

To see that this transformation produces a valid HMM that is equivalent to the original pHMM, observe that each sequence of Delete transitions always terminates with a transition to a non-delete $(1 - A_{(D,k'),D})$ (Equation 4.2). Each transition to a non-delete is also associated with a choice between a *Match* or *Insert* state, which cancels the factor, $1 - A_{(D,k'),D}$, from the non-delete transition (Equation 4.3). The *Match* or *Insert* transition is included in the modified emission, $B^{(2S)}_{(s,k),m}$. This sequence of steps has the same probability as each sequence of transitions that lead to an emitted symbol in the original pHMM.

Sethuraman's stick-breaking construction of the [71] Dirichlet Process is defined by the following generative procedure:

$$\begin{aligned} V_i &\sim \text{Beta}\,(1, \gamma) \\ \pi_j &= V_j \prod_{i<j} (1 - V_i) \end{aligned} \tag{4.4}$$

The distribution π is then said to be generated by a stick-breaking process, and we denote a draw from this infinite multinomial by $\pi \sim SB(\gamma)$.

To establish equivalence between pHMM transitions and the stick-breaking process, we use the aggregation property of the Dirichlet distribution [72] to decompose the Dirichlet prior on transition probabilities as follows:

$$\begin{aligned}&\mathrm{Dir}(A_{(s,k),\cdot}; \alpha_{s,s'}) \\ &\quad = \mathrm{Beta}\left(A_{(s,k),D}; \alpha_{s,D}, \alpha_{s,M} + \alpha_{s,I}\right) \mathrm{Beta}\left(\frac{A_{(s,k),M}}{1 - A_{(s,k),D}}; \alpha_{s,M}, \alpha_{s,I}\right) \quad (4.5)\end{aligned}$$

The Beta distribution that generates $A_{(s,k),D}$ (Equation 4.5) is a prior on the choice of transitioning to a *Delete* or non-*Delete* state. If we set $\alpha_{s,D} = 1$, the process of selecting a non-*Delete* state for a sequence of length one becomes equivalent to the stick-breaking construction for the Dirichlet process (Equation 4.4):

$$\begin{aligned} A_{(s,k),D} &\sim \mathrm{Beta}(1, \alpha_{s,M} + \alpha_{s,I}) \\ &\Downarrow \\ A^{(2S)}_{(s,k),k'} &\sim SB(\alpha_{s,M} + \alpha_{s,I}) \end{aligned} \quad (4.6)$$

All valid settings of Dirichlet prior parameters (i.e. $\alpha_{s,s'} > 0$) also lead to valid distributions over transition probabilities to an unbounded number of hidden states as determined by the criterion given by Ishwaran and James [73]: $\sum_{i=1}^{\infty} \pi_i = 1$ a.s. iff $\sum_{i=1}^{\infty} \log\left(1 + \frac{\alpha_{s,D}}{\alpha_{s,M}+\alpha_{s,I}}\right) = +\infty$.

From the 2S-HMM construction, we can also derive an expression for the a-priori probability for n_D, the number of *Delete* states that the model traverses to generate a single observed symbol:

$$
\begin{aligned}
p(n_D|\alpha) &= \int_{dA} p(A|\alpha) A^{(2S)}_{(s,k),n_D-k} \\
&= \prod_{s,k} \frac{\Gamma(\alpha.)}{\Gamma(\alpha_D)\Gamma(\alpha_{\neg D})} \int_{dA} \left(\prod_{s'} A^{\alpha_{s'}-1}_{(s,k),s'} \right) A^{(2S)}_{(s,k),n_D-k} \\
&= \frac{\Gamma(\alpha.)}{\Gamma(\alpha_D)\Gamma(\alpha_{\neg D})} \frac{\Gamma(\alpha_D)\Gamma(\alpha_{\neg D}+1)}{\Gamma(\alpha.+1)} \\
&\quad \prod_{k=1}^{n_D} \frac{\Gamma(\alpha.)}{\Gamma(\alpha_D)\Gamma(\alpha_{\neg D})} \frac{\Gamma(\alpha_D+1)\Gamma(\alpha_{\neg D})}{\Gamma(\alpha.+1)} \\
&= \frac{\alpha_{\neg D}}{\alpha.} \left(\frac{\alpha_D}{\alpha.} \right)^{n_D}
\end{aligned}
\tag{4.7}
$$

where , where we have overloaded the definition of n_D and fixed $\alpha_D \doteq \alpha_{s,D}$ and $\alpha_{\neg D} \doteq \alpha_{s,\neg D}$ to be the same for all s. It is clear from the expression in Equation 4.7 that the probability decreases to zero as the number of delete states approaches infinity. We can also compute the expected number of Delete states traversed over a single transition as follows:

$$
\begin{aligned}
E[n_D|\alpha] &= \sum_{n_D=1}^{\infty} n_D \frac{\alpha_{\neg D}}{\alpha.} \left(\frac{\alpha_D}{\alpha.} \right)^{n_D} \\
&= \frac{\alpha_D}{\alpha_{\neg D}}
\end{aligned}
\tag{4.8}
$$

In the Infinite pHMM we can therefore expect, a-priori, to traverse a finite number of Delete states for a sequence of length one. We can perform a similar derivation for a single sequence of T emitted symbols, where the model traverses $n_{D,t}$ delete states for each position in the sequence:

Symbol	Description
$z_t^{(GT)}$	The hidden state associated with the emitted symbol at position t of the sequence, taking on values from the pair $(k \in 1..K,\ s \in \{\text{Match}, \text{Insert}\})$, $z_t^{(GT)} \in (s,k)$ (*Delete* states are not included in this sequence).
$A^{(GT)}_{(s,k),(s',k')}$	The probability of a sequence of pHMM transitions beginning at state (s,k) ending at state (s',k'), $s, s' \in \{M, I\}$.
$n^{(GT)}_{(s,k),(s',k')}$	Given a set of hidden states, $z^{(GT)}_{1:T}$, the total number of transitions in the dataset between hidden states (s,k) and (s',k').

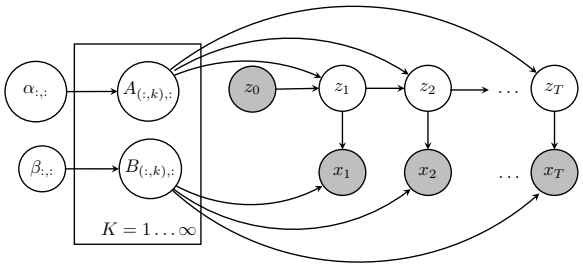
Table 4.3: Parameter definitions for the GT-HMM

$$p(n_{D,:}|\alpha, T) \quad = \quad \prod_{t=1}^{T} \frac{\alpha_{\neg D}}{\alpha_.} \left(\frac{\alpha_D}{\alpha_.}\right)^{n_{D,t}} \tag{4.9}$$

In a single sequence, the $n_{D,t}$'s are independent, and we can expect to traverse $E\left[\sum_t n_{D,t}|\alpha\right] = T\frac{\alpha_D}{\alpha_{\neg D}} = O(T)$ hidden states, a-priori, for a sequence of length T. Because the Infinite pHMM must traverse hidden states in order by base hidden state (e.g. to get to state $(s, k+\ell), s'$ the model must have traversed $(s_p, k), s'_p$ for some $s_p, s'_p \in \{M, I, D\}$,$\ell > 1$), modeling a set of sequences causes the desired behavior of forcing the Infinite pHMM to reuse hidden states as it generates separate sequences from the set.

4.3.1 The Geometric Transition HMM

To construct a set of efficient inference methods, we present a transformation from the pHMM to an equivalent HMM, which we refer to as the Geometric Transition HMM (GT-HMM) due to the geometrically decreasing transition probabilities from each hidden state. This transformation merges sequences of *Delete* transitions with a single terminating *Match* or *Insert* transition so that every transition becomes associated with an emission. This



(a)

$$A_{(s,k),:} \sim \text{Dir}\left(\alpha_{s,:}\right)$$
$$B_{(s,k),:} \sim \text{Dir}\left(\beta_{s,:}\right)$$
$$z_t \sim \text{Mult}\left(A^{(GT)}_{z_{t-1},(:,:)}\right)$$
$$x_t \sim \text{Mult}\left(B_{z_t,:}\right)$$

(b)

Figure 4.1: (a) The plate diagram of the infinite pHMM, and (b) the generative process for the infinite pHMM. Note that emission and transition probabilities are from the GT-HMM. We use ":" symbols in subscripts as in Matlab notation, indicating a vector or matrix with all possible values of the replaced parameter.

transformation is similar to existing techniques in speech recognition applications for transforming an HMM with non-emitting states to one that emits after every transition [8]. To the best of our knowledge, we are the first to propose this transformation for pHMMs.

We illustrate the effect of the transformation from pHMM to GT-HMM in Figure 4.2, which shows all transitions to hidden state (M, k) emitting x_t from the set of previous hidden states that could emit x_{t-1} in a portion the lattice used for the forward-backward algorithm. The GT-HMM does not contain any non-emitting *Delete* states. Instead, it merges *Delete* states in the pHMM with emitting states, allowing transitions from a much larger number of states in column $t-1$.

The infinite pHMM can be represented by a plate diagram using the GT-HMM as shown in Figure 4.1a. The generative procedure for the infinite pHMM is given in Figure 4.1b.

Equation 4.10 shows GT-HMM transition probabilities to *Match* states in terms of the original pHMM transitions, where (a) indicates a sequence of delete transitions followed by a *Match*, (b) indicates a single *Match* transition, and (c) disallows transition to a state with a value of k less than or equal to that of the current state. Similarly, Equation 4.11 shows GT-HMM transition probabilities to *Insert* states in terms of the original pHMM transitions, where (d) indicates a sequence of *Deletes* followed by an *Insert*, (e) indicates a single *Insert* transition, and (f) disallows transition to a state with a smaller value of k.

$$A^{(GT)}_{(s,k),(M,k')} = \begin{cases} A_{(s,k),D}\left(\prod_{k''=k+1}^{k'-2} A_{(D,k''),D}\right) A_{(D,k'-1),M} & k < k'-1 \quad (a) \\ A_{(s,k),M} & k = k'-1 \quad (b) \\ 0 & k \geq k' \quad (c) \end{cases} \tag{4.10}$$

$$A^{(GT)}_{(s,k),(I,k')} = \begin{cases} A_{(s,k),D}\left(\prod_{k''=k+1}^{k'-1} A_{(D,k''),D}\right) A_{(D,k'),I} & k < k' \quad (d) \\ A_{(s,k),I} & k = k' \quad (e) \\ 0 & k > k' \quad (f) \end{cases} \tag{4.11}$$

The probability of a set of sequences for the GT-HMM is given by the same expression as the standard HMM, but with hidden states parametrized by (s,k) pairs with $s \in \{M, I\}$ and $k \in [1 \ldots K]$:

$$p(x_{1:T}, z_{1:|z|}|A,B) = \prod_{(s,k),(s',k')} \left(A^{(GT)}_{(s,k),(s',k')}\right)^{n^{(GT)}_{(s,k),(s',k')}} \prod_{(s,k),m} \left(B_{(s,k),m}\right)^{n_{(s,k),m}} \tag{4.12}$$

Because each transition under the GT-HMM results in an emission, we can now consider all possible transitions between hidden states (s,k) and (s',k'), which allows forward and backward recursions to be run using the standard two-dimensional lattice (shown in Figure 4.2 for transitions to a single *Match* state) rather than the three-dimensional lattice commonly used in pHMM inference.

In addition, Equations 4.10 and 4.11 indicate that increasing the destination state index k' while holding the source index k fixed causes additional $A_{(D,k),D}$ terms (*Delete*-to-*Delete* transitions) to be included in $A^{(GT)}_{(s,k),(s',k')}$, leading to both an exponential decrease in the GT-HMM transition probability and to a limit of 0 as k increases indefinitely.

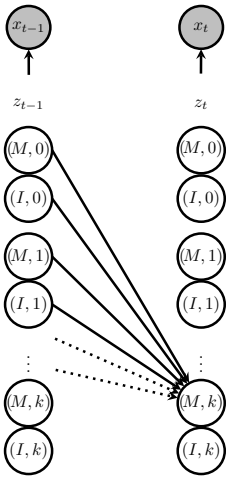


Figure 4.2: A section of the two-dimensional lattice used for GT-HMM inference showing transitions used to compute the forward recurrence for state (M,k) for position t in the sequence. Unlike the pHMM, the GT-HMM no longer includes *Delete* states. In addition, transitions have been added from all states in column $t-1$ to states in column t with larger values of k.

4.4 Inference

We use a structured variational algorithm for inference, factoring the variational distribution as follows:

$$q(\vec{z}, A, B) = q(\vec{z}) \prod_{(s,k)} q(A_{(s,k),:}) \prod_{(s,k)} q(B_{(s,k),:})$$

The update equations for maximizing the variational bound[2] on the marginal likelihood and minimizing the KL divergence between the true posterior and variational posterior are given in Algorithm 4.4 (Figure 4.1).

Similar structured variational methods for performing HMM inference are described in [66,68,74], so we omit the details of our derivation. Care must be taken, however, to correctly

[2]The full expression for the bound is provided in Appendix B.1.

Algorithm 4.1 Variational Inference algorithm for the infinite pHMM. Parameter definitions are given in Tables 4.1 and 4.3. The Ψ symbol indicates the digamma function: $\Psi(x) = \frac{\partial \log \Gamma(x)}{\partial x}$.

Repeat until the variational bound converges:
1) Compute Expectations

$$
\begin{aligned}
E\left[n_{(s,k),s'}\right] &\text{ from the forward-backward algorithm} \\
E\left[n_{(s,k),m}\right] &\text{ from the forward-backward algorithm} \\
E\left[\log A_{(s,k),s'}\right] &= \Psi\left(\tilde{\alpha}_{(s,k),s'}\right) - \Psi\left(\sum_{s''} \tilde{\alpha}_{(s,k),s''}\right) \\
E\left[\log B_{(s,k),m}\right] &= \Psi\left(\tilde{\beta}_{(s,k),m}\right) - \Psi\left(\sum_{m'} \tilde{\beta}_{(s,k),m'}\right)
\end{aligned}
$$

2) Maximize with respect to variational parameters

$$
\begin{aligned}
\tilde{\alpha}_{(s,k),s'} &\leftarrow \alpha_{s,s'} + E\left[n_{(s,k),s'}\right] \\
\tilde{\beta}_{(s,k),m} &\leftarrow \beta_{s,m} + E\left[n_{(s,k),m}\right] \\
\tilde{A}_{(s,k),s'} &\leftarrow \exp\left(E\left[\log A_{(s,k),s'}\right]\right) \\
\tilde{B}_{(s,k),m} &\leftarrow \exp\left(E\left[\log B_{(s,k),m}\right]\right)
\end{aligned}
$$

convert counts of transitions (the sufficient statistic associated with transition probabilities) in the GT-HMM ($E\left[n^{(GT)}_{(s,k),(s',k')}\right]$) back to pHMM counts ($E\left[n_{(s,k),s'}\right]$), used to update the variational parameter, $\tilde{A}$:

$$
E\left[n_{(s,k),s'}\right] = \sum_{(s,k),s' \in (s,k),(s',k')} E\left[n^{(GT)}_{(s,k),(s',k')}\right] \tag{4.13}
$$

If a pHMM transition is an element of the aggregate GT-HMM transition, then we add this expectation to the expectation under the pHMM.

These expected emission and transition counts, which are normally computed using the forward-backward algorithm, are computed for the GT-HMM using a beam search version

of the forward backward algorithm described in the next section.

We use a truncated model for our variational approximation, setting the value of K to a large, but finite, value. Truncation, of course, transforms the infinite model into a finite model. To ensure that this approximation is accurate, it is critical to set the truncation levels large enough to capture the bulk of the posterior probability mass over the set of hidden states.

4.4.1 Beam Methods

Beam methods [8, 64] increase the speed of computation in message passing algorithms by eliminating hidden variable assignments that do not contribute significantly to the expectation or marginal probability. Specifically, we would like to eliminate, without major degradation in accuracy, as many terms as possible from the sum over z_{t-1} in each forward recurrence, $p(z_t, x_{1:t}) = \sum_{z_{t-1}} p(x_t|z_t)p(z_t|z_{t-1})p(z_{t-1}, x_{1:t-1})$, where z_{t-1} indicates the column in the lattice associated with the $t-1^{th}$ observation. Figure 4.3 shows a beam containing the values of z_{t-1} that are retained in the sum.

Beam methods are not effective in the standard pHMM because the forward and backward recurrences in the three-dimensional lattice are computed over only three states, *Match*, *Insert*, and *Delete* (Figure 2.3). In contrast, the forward and backward recurrences in the GT-HMM involve sums over the $O(K)$ moves (K is the truncation level of the pHMM), between adjacent columns in the two-dimensional lattice (Figure 4.2). This allows us to eliminate a larger number of terms from the sum in each recurrence.

Although the *GT*-HMM transformation has the potential to speed up inference, it may have the opposite effect in certain cases. These cases can be quantified by a straightforward analysis of the number of messages passed for each observed symbol. The forward-backward algorithm on the GT-HMM passes $O\left(T \cdot K^2\right)$ messages per sequence, while on the pHMM it passes $O\left(T \cdot K\left(3^2\right)\right)$ messages. This means that for the beam method to run faster than the standard pHMM forward-backward algorithm, the size of the beam, $K^{(beam)}$, (if we

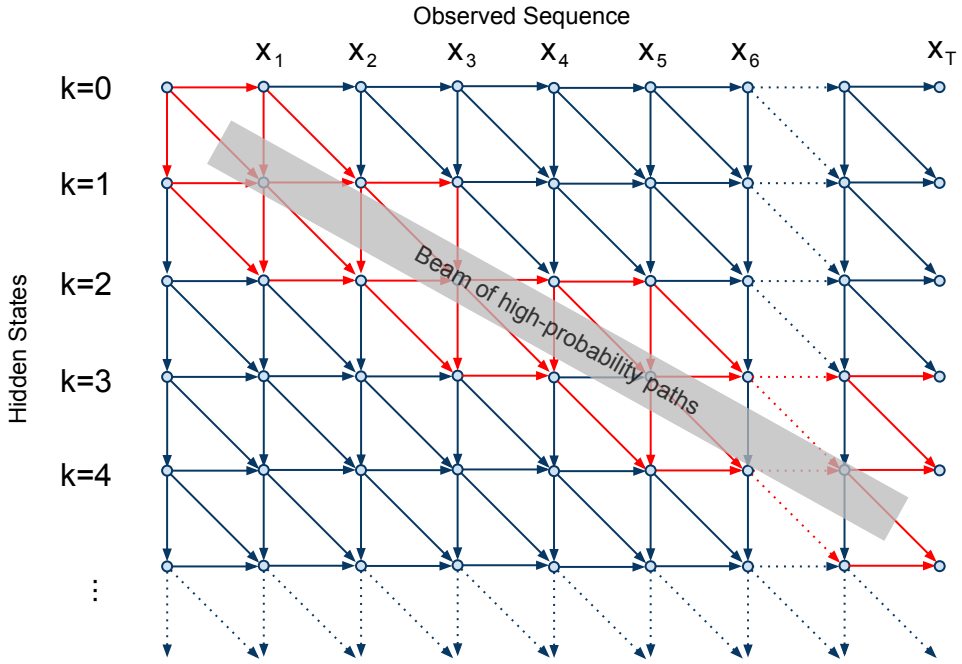


Figure 4.3: The lattice used for pHMM inference. Observed sequences are generated by paths of hidden states through the graph. Transitions marked in red indicate a potential beam of highly probable paths within the total set of possible paths. *Match*, *Insert*, and *Delete* states are merged for clarity.

assume a constant beam size) must satisfy $K^{(beam)} < 3\sqrt{K}$. Therefore, if the number of pHMM hidden states needed to accurately model the set of observed sequences is known to be small beforehand, the standard forward-backward algorithm may be more desirable than our beam methods. However, if the number of hidden states needed to accurately model a dataset is large, then beam methods will be useful in pHMM inference because the size of $K^{(beam)}$ depends primarily on the prior parameters in the distributions over transition and emission probabilities, α and β, rather than the truncation level, K.

In the *GT*-HMM, we apply the KL divergence beam method from [64]. However, the KL-divergence approach by itself is not enough. We are still required to compute $O(K)$ forward messages for each lattice column t even though some these messages are very close to zero. These tiny values result from the combined effect of a small number of forward messages in the beam of column $t-1$ and exponentially decreasing transition probabilities. For a faster beam method, we can specify, for each column of the lattice, a threshold on the hidden state index k, and stop computing forward messages when the threshold is exceeded.

We derive this thresholding criterion from an adaptation of the auxiliary variable beam method described by Van Gael et. al. [65]. In the auxiliary variable method, a uniformly distributed auxiliary variable, u_t, is added to the model. The distribution of u_t is conditioned on the values of hidden states t and $t-1$, yielding the joint distribution:

$$p(u_{1:T}, z_{1:T}, x_{1:T}|A, B) = \prod_t p(u_t|z_t, z_{t-1})p(z_t|z_{t-1})p(x_t|z_t) \tag{4.14}$$

$p\left(u_t|z_t, z_{t-1}\right)$ is chosen so that marginalizing with respect to $u_{1:T}$ returns the original joint distribution: $p\left(u_t|z_t, z_{t-1}\right) = \frac{\mathbb{I}(u_t < p(z_t|z_{t-1}))}{p(z_t|z_{t-1})}$, where $\mathbb{I}\left(\omega\right) = 1$ if ω is true and 0 otherwise and z_t indicates the hidden state at position t in the sequence.

We use the auxiliary variables in conjunction with a variational argument to justify truncating the computation of forward probabilities using a fixed thresholding criterion. With the addition of auxiliary variables, forward messages in the GT-HMM can be expressed as follows:

$$p(z_t, x_{1:t}) = \int_{u_t} \sum_{z_{t-1}} p(x_t|z_t)p(u_t|z_t, z_{t-1})p(z_t|z_{t-1})p(z_{t-1}, x_{1:t-1}) \tag{4.15}$$

where $x_{1:t}$ indicates the sub-sequence of observed symbols from positions 1 to t.

This formulation allows us to construct the following variational bound:

$$\begin{aligned}
\log p(z_t, x_{1:t}) &\geq \int_{u_t} \sum_{z_{t-1}} q_{z_t}(z_{t-1}, u_t) \log \frac{p(x_{1:t}, z_t, z_{t-1}, u_t)}{q_{z_t}(z_{t-1}, u_t)} \\
&= \log p(x_t|z_t) + \int_{u_t} \sum_{z_{t-1}} q_{z_t}(z_{t-1}, u_t) \log \frac{p(u_t|z_t, z_{t-1})p(z_t|z_{t-1})p(z_{t-1}, x_{1:t-1})}{q_{z_t}(z_{t-1}, u_t)}
\end{aligned} \tag{4.16}$$

We assume a factorization of $q_{z_t}(z_{t-1}, u_t) = q_{z_t}(z_{t-1})q_{z_t}(u_t)$, which allows us to maximize with respect to the individual factored portions of the distribution. This maximization results in following expressions (derivation in Appendix B.2):

$$q_{z_t}(z_{t-1}) \propto \mathbb{I}\left(p(z_t|z_{t-1}) \geq \tilde{\theta}_t\right) p(z_{t-1}, x_{t-1}) \tag{4.17}$$

$$q_{z_t}(u_t) = \frac{\mathbb{I}\left(u_t < \tilde{\theta}_t\right)}{\tilde{\theta}_t} \tag{4.18}$$

where $\tilde{\theta}_t$ is a variational parameter specifying a truncation threshold for each position in the observed sequence. A set of approximate posteriors, $q_{z_t}(z_{t-1})$ and $q_{z_t}(u_t)$, are computed for the forward recurrence, $p(z_t, x_{1:t})$, for each value of z_t, hence the subscript on the variational distributions.

The distribution $q_{z_t}(z_{t-1})$ is over z_{t-1} rather than z_t. Therefore, it does not make sense to use these quantities directly for computing forward recurrences. Instead, we pretend that

we are using $q_{z_t}(z_{t-1})$ to sample values of z_{t-1} in the backward direction given that we know the value of z_t. If $q_{z_t}(z_{t-1})$ is empty, i.e., there are no transitions to z_t where $p(z_t|z_{t-1}) \geq \tilde{\theta}_t$, then we would have no way to sample a hidden state given that the value of the current hidden state is z_t. We therefore use this criterion for truncating computation of forward recurrence values: if transitions from all states in the beam for lattice-column $t-1$ to the current state $z_t = (s, k)$ fall below $\tilde{\theta}_t$, then we do not compute values of $p(z_t = (s, k'), x_{1:t})$ for $k' > k$.

4.5 Experiments

We evaluated the effectiveness of the beam method using the GT-HMM model, computing both the speed of the inference and test-set perplexities. We compared our approach to the forward-backward algorithm in the standard pHMM. We conducted experiments on a synthetic dataset generated from three pHMMs designed to emphasize different possible latent configurations. The structures of these pHMMs are described in Table 4.4. We also ran experiments on an ASTRAL[3]-filtered subset of the Structural Classification of Proteins (SCOP) [75] database (see Section 2.4). Tests were run using 5-fold cross validation on a subset of the SCOP 1.75 dataset filtered at 95% identity. This dataset contained classes having between 80 and 200 sequences.

We evaluated the ability of our beam method to increase the speed of inference while maintaining an accuracy (in terms of perplexity) comparable to the standard (no beam) forward-backward algorithm (We refer to the standard pHMM as the "no beam" model in results Tables 4.5, 4.6, and 4.7). For all experiments, we conducted inference using the variational algorithm described in Figure 4.1. The algorithm was stopped when either the criterion, $0 \leq \frac{B_t}{B_{t-1}} - 1 < 10^{-6}$, was reached, where B_t indicates the value of the variational bound at iteration t, or until 300 iterations were performed. We set the prior parameters,

[3] http://astral.berkeley.edu/

Name	Hidden States (K)	Emitted Symbols (M)	Description
$\text{PHMM}_{\mathbf{M}}$	20	2	90% probability of a match transition; emission of symbol zero from match states increases as the value of k increases; uniform emissions from insert states
$\text{PHMM}_{\mathbf{I}}$	20	3	99.9% probability transitioning to an insert state and a 90% probability of remaining in an insert state at the 10th hidden state
$\text{PHMM}_{\mathbf{D}}$	40	3	47.5% probability of a delete transition and a 47.5% probability of a match transition from the central 20 hidden states; 90% probability of a match transition from all other hidden states

Table 4.4: Profile HMMs used to generate synthetic datasets.

α and β, uniformly to 0.5 and truncated the infinite model by setting K to be twice the maximum length sequence in the dataset. Learning transitions from *Insert* states tends to drive inference toward bad local minima with multiple chains of insertions. Hence, we fixed the distribution of *Insert* transitions to uniform.

To compute perplexity under the Bayesian model, we approximated A and B using the expectations under the variational distributions $q(A;\tilde{\alpha})$ and $q(B;\tilde{\beta})$:

$$\text{perplexity} \quad = \quad \exp\left(-\frac{\sum_n \log p(x_n|E_{q(A)}[A], E_{q(B)}[B])}{\sum_n |x_n|}\right)$$

For all experiments, perplexity was computed using expectation from the standard (no beam) forward-backward algorithm.

In all experiments, we tested three settings of beam parameters, "narrow," "medium," and "wide" with threshold settings of $\epsilon = \left[10^{-2}, 10^{-3}, 10^{-4}\right]$ and $\tilde{\theta} = \left[10^{-16}, 10^{-17}, 10^{-18}\right]$. The parameter ϵ indicates the threshold used for the KL divergence beam and $\tilde{\theta}$ indicates the threshold used for the auxiliary variable beam. Smaller beam thresholds are associated with larger beams.

Synthetic Data: Test Set Perplexities				
Model/ SCOP Category	Beam			No Beam
	Narrow	Medium	Wide	
$\text{PHMM}_{\mathbf{M}}$	1.72	1.71	1.71	2.15
$\text{PHMM}_{\mathbf{I}}$	2.27	2.21	2.22	2.65
$\text{PHMM}_{\mathbf{D}}$	2.34	2.33	2.33	2.74

(a)

Synthetic Data: Inference Times (seconds)				
Model/ SCOP Category	Beam			No Beam
	Narrow	Medium	Wide	
$\text{PHMM}_{\mathbf{M}}$	11.36	22.98	35.31	101.11
$\text{PHMM}_{\mathbf{I}}$	3.95	38.91	47.86	280
$\text{PHMM}_{\mathbf{D}}$	9.2	36.09	54.96	245.57

(b)

Table 4.5: The charts above show a comparison of test-set perplexities **(a)** and run times **(b)** between inference using our beam methods and the standard (no beam) forward-backward method on sets of synthetic datasets. "Narrow," "medium," and "wide" indicate threshold settings of $\epsilon = \left[10^{-2}, 10^{-3}, 10^{-4}\right]$ and $\tilde{\theta} = \left[10^{-16}, 10^{-17}, 10^{-18}\right]$, where ϵ indicates the KL divergence beam threshold and $\tilde{\theta}$ indicates the auxiliary variable threshold.

4.5.1 Synthetic Data

To test the performance of our beam method, we generated 100 training and 25 test sequences from the pHMM models described in Table 4.4. We then compared pHMMs inferred on the training set using our beam methods against models inferred using the standard forward-backward algorithm by both computing perplexities on test sets and recording the time needed for inference. Table 4.5 shows the results of these experiments. The beam methods achieved a lower perplexity than the standard forward-backward algorithm, with an average decrease of 17% over all beam thresholds. The beam methods were able to converge to a better optimum by eliminating paths of hidden states that were not useful for inference. The results confirmed our expectations that beam methods improve runtime, with an average improvement of 84% over all of the thresholds tested (Table 4.5a). As the threshold values increased, more hidden states were included in the beam, and runtime increased by an average of 13% from the narrow to the medium beam and 9.5% from the medium to the wide beam (Table 4.5b).

To show that our beam methods capture similar characteristics of the data as standard forward-backward inference, we present a series of heat maps. Cells in each heat map (Figure 4.4) associate expectations of the number of paths though hidden states to intensity in a two-dimensional grid - i.e., a cell (k, t) in the heat map is the sum expectations that a

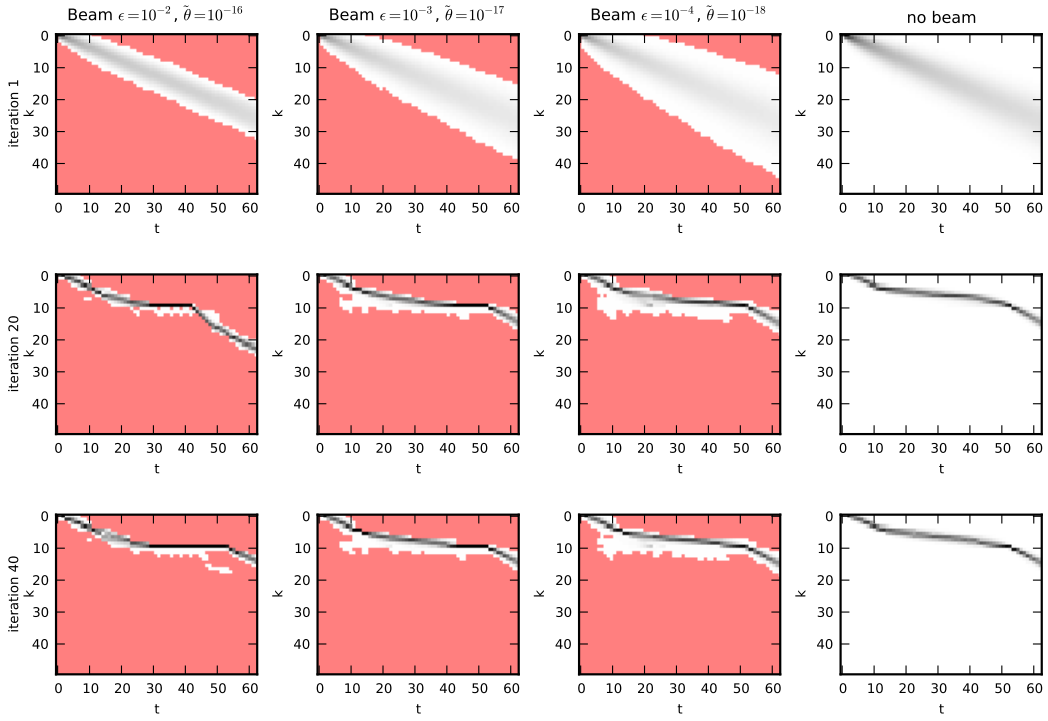


Figure 4.4: A set of heat maps generated at 20-step intervals during inference on the pHMM$_{\mathbf{I}}$ dataset for a variety of beam settings. Each cell in the heat map indicates the expectation that either a *Match* or *Insert* state with parameter k (row index) generated the observed symbol at position t (column value) in the heat map. Darker colors indicate higher values, and red indicates that a hidden state was excluded from the beam.

path of hidden states passing though either (M, k) or (I, k) generates symbol x_t. The heat maps depict expectations for the longest sequence in the training set generated by pHMM$_{\mathbf{I}}$ collected at intervals of 20 steps during inference. We created heat maps for each beam threshold setting and for the no-beam experiment. For all levels of the beam threshold, the configuration of hidden state expectations nearly matches the configuration in the no-beam setting. The areas of red in the beam-inference heat maps indicate hidden states that were excluded from inference. Darker colors in the heat maps indicate larger values. Comparing areas of red in the graphs shows how lower beam thresholds consider larger numbers of hidden states.

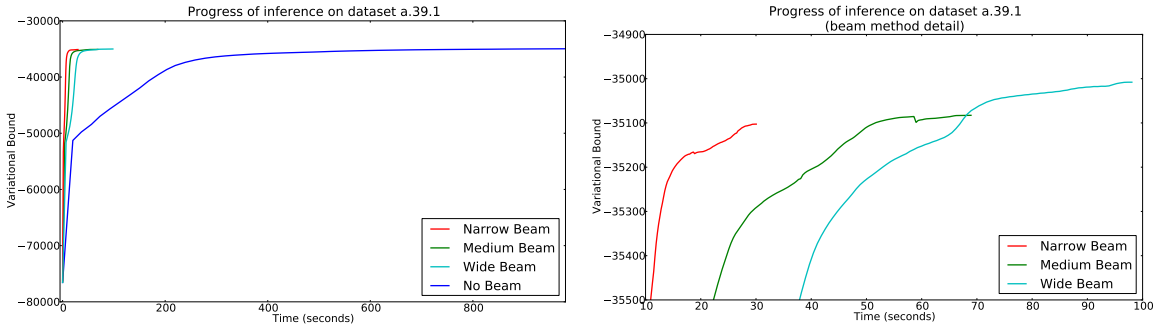
4.5.2 SCOP Datasets: Uniform Initialization

To compare the infinite pHMM's performance on the SCOP dataset, we ran experiments using 5-fold cross validation on each SCOP category. Models were initialized using uniform expectations of transition and emission probabilities, i.e. we initialize variational parameters $\tilde{A}_{(s,k),s'} = \frac{1}{3}$ (standard pHMM) and $\tilde{B}_{(s,k),m} = \frac{1}{|\Sigma|}$. Results on the SCOP datasets (Table 4.6) show a similar pattern as in the synthetic experiments: in comparison to the no-beam method, the beam method improved both perplexity (3.51% for the narrow beam, 4.56% for the medium beam, and 4.75% for wide beam averaged over all categories - see Table 4.6a) and substantially decreased inference runtime (98.60% for the narrow beam, 96.90% for the medium beam, and 94.46% for the wide beam, averaged over all categories - see Table 4.6b).

To better visualize the effect of the beam on inference, we plotted the variational bound on the marginal likelihood against runtime for the first fold of the 5-fold training partition (Figure 4.5a). As expected, narrower beam thresholds converge faster, and the beam methods converge faster than the no-beam forward-backward algorithm for all thresholds tested. Figure 4.5b shows a zoomed portion of the graph in Figure 4.5a toward the end of convergence of the beam methods. This portion shows that inference using the beam method, unlike the standard forward-backward algorithm, does not necessarily increase the variational bound at every step. If the optimum that the algorithm approaches is well-behaved, then it is possible for the beam method to perform better than the non-beam method by ignoring irrelevant states. However, the beam could also ignore useful states, causing inference to progress to a non-optimal point in the model's parameter space.

4.5.3 SCOP Datasets: MSA initialization

We ran additional inference experiments where we attempted to improve perplexities from pHMMs constructed using multiple sequence alignments (MSAs) [63]. The MSAs were created on each training partition of our SCOP 1.75 dataset using the MUSCLE program



(a) (b)

Figure 4.5: **(a)** Shows a comparison of rates of improvement of the variational bound between the beam algorithms using the three separate thresholds and the standard (no-beam) forward-backward algorithm. As the beam threshold decreases, inference speed increases. All beam settings converge faster than the no-beam method. **(b)** Shows detail for the beam methods. Unlike the non-beam method, each iteration of beam inference is not guaranteed to increase the variational bound. Both graphs show inference on superfamily a.39.1 from the SCOP 1.75, 95% dataset. "Narrow," "medium," and "wide" indicate threshold settings of $\epsilon = \left[10^{-2}, 10^{-3}, 10^{-4}\right]$ and $\tilde{\theta} = \left[10^{-16}, 10^{-17}, 10^{-18}\right]$, where ϵ indicates the KL divergence beam threshold and $\tilde{\theta}$ indicates the auxiliary variable threshold.

[76] with default settings. We constructed pHMMs from each MSA using an 80% *Match*-state cutoff value. The cutoff values were used to determine the number of non-blanks in an alignment column that would cause us to associate that column with a *Match* state in the derived pHMM. The pHMM size K was set to the number of Match columns, and emission probabilities for each *Match* state were estimated using the frequencies of amino acids in each *Match* column of the alignment. We computed initial perplexities of the pHMM-MSAs using the marginal likelihood, $p(x_n|A, B)$, computed directly from the (no-beam) forward-backward algorithm.

Once these initial pHMMs were constructed, we used expected counts as a starting point for variational inference using both the standard forward-backward algorithm as well as our beam method with the same set of beam thresholds as in the previous set of SCOP 1.75 experiments. Unlike the previous experiments, we did not fix insert transition probabilities.

Narrower beams (large thresholds) produced higher perplexities than wider beams. In some cases, beams that were too narrow led to inference steps that decreased the variational

Uniform Initialization: Test Set Perplexities				
Model	Beam			No Beam
	Narrow	Medium	Wide	
Average	16.68	16.50	16.48	17.31
StDev	1.81	1.80	1.83	1.94

(a)

Uniform Initialization: Inference Times (seconds)				
Model	Beam			No Beam
	Narrow	Medium	Wide	
Average	85	205	373	9138
StDev	54	151	260	7959

(b)

Table 4.6: The charts above show comparisons of average test-set perplexities **(a)** and convergence times **(b)** between the beam method and the standard (no beam) forward-backward inference averaged over all folds and categories of our SCOP 1.75 dataset. Inference was initialized with uniform expected transition and emission distributions. "Narrow," "medium," and "wide" indicate threshold settings of $\epsilon = \left[10^{-2}, 10^{-3}, 10^{-4}\right]$ and $\tilde{\theta} = \left[10^{-16}, 10^{-17}, 10^{-18}\right]$, where ϵ indicates the KL divergence beam threshold and $\tilde{\theta}$ indicates the auxiliary variable threshold.

bound. The beam methods improved perplexity from the MSA-induced pHMM for 5, 7, and 10 out of 26 categories for the narrow, medium, and wide beams respectively (see Table 4.7a for a full list of perplexities). For the categories where the beam method produced an improvement in perplexity, we saw average improvements in runtime of 92.68%, 95.88%, and 97.79% for the narrow, medium, and wide beams over no-beam inference, respectively (see Table 4.7b for a full list of runtimes).

Using approximate expectations from the beam during inference does not guarantee an inference algorithm that increases the variational bound at each step (see Figure 4.5). To mitigate this concern, the beam can be increased to allow inference to perform more similarly to the no-beam method, as indicated by the trend of decreasing average perplexities in Table 4.7a as the beam thresholds decrease. However, increasing the size of the beam reduces the speed of the algorithm.

MSA Initialization: Test Set Perplexities					
Model/ SCOP Category	Beam			No Beam	MSA PHMM
	Narrow	Medium	Wide		
a.1.1	13.89	**13.34**	**12.85**	**12.65**	13.67
a.39.1	14.75	13.31	13.44	**11.18**	11.66
a.4.1	17.63	17.13	16.71	**14.56**	15.33
b.1.18	17.72	17.33	**16.99**	**16.70**	17.10
b.1.2	18.19	18.19	18.08	**16.41**	16.94
b.121.4	15.56	14.07	14.46	**12.87**	13.80
b.29.1	**16.17**	**16.47**	**16.11**	**15.46**	16.59
b.36.1	13.65	13.15	13.25	**9.56**	9.85
b.40.4	**17.57**	**17.57**	**17.57**	**17.69**	18.05
b.47.1	**9.98**	**9.85**	**9.75**	**9.86**	10.22
b.55.1	17.78	17.60	17.85	**16.36**	16.85
b.6.1	17.81	17.51	16.85	**14.83**	15.31
c.3.1	17.36	16.60	**16.43**	**15.89**	16.43
c.47.1	17.99	17.99	17.99	**15.45**	15.98
c.55.1	17.10	17.36	17.13	**16.65**	16.88
c.66.1	18.09	18.08	18.09	**16.49**	17.29
c.67.1	17.87	17.85	17.86	**15.02**	15.51
c.69.1	17.91	17.70	17.68	**16.39**	17.04
c.94.1	**13.47**	**13.01**	**12.97**	**13.61**	15.38
d.108.1	18.29	18.26	18.28	**17.03**	17.28
d.144.1	18.34	17.41	17.20	**10.62**	11.17
d.15.1	16.69	16.42	16.69	**14.05**	14.46
d.3.1	**15.61**	**15.42**	**15.62**	**14.13**	15.96
d.58.7	11.89	11.51	**11.50**	**11.21**	11.51
g.37.1	12.53	**11.65**	**11.67**	**11.28**	11.70
g.39.1	14.16	14.38	14.36	**12.88**	13.38
Average	16.08	15.74	15.67	**14.18**	14.60

Table 4.7: The charts above show comparisons of average test-set perplexities between our beam methods, the standard (no beam) forward-backward, and the MSA-derived pHMM for 26 superfamilies of the SCOP 1.75 dataset. Bolded numbers indicate experiments where a test perplexity from the inferred pHMM was lower than that of the MSA-derived pHMM on the same category. Inference was initialized using the MSA-induced pHMM. "Narrow," "medium," and "wide" indicate threshold settings of $\epsilon = \left[10^{-2}, 10^{-3}, 10^{-4}\right]$ and $\tilde{\theta} = \left[10^{-16}, 10^{-17}, 10^{-18}\right]$, where ϵ indicates the KL divergence beam threshold and $\tilde{\theta}$ indicates the auxiliary variable threshold.

4.6 Conclusion

In this chapter, I have shown how the Profile HMM can be extended to allow an infinite number of hidden states. Our understanding of the infinite model is closely linked to two views of the pHMM where transitions are aggregated to form standard HMMs. These aggregate views lead to a novel beam search inference algorithm. A sub-portion of our beam method, the variational auxiliary variable criterion, is a thresholding scheme derived from

MSA Initialization: Inference Times (seconds)				
Model/ SCOP Category	Beam			No Beam
	Narrow	Medium	Wide	
a.1.1	29	47	106	839
a.39.1	24	37	67	760
a.4.1	4	10	23	401
b.1.18	5	11	72	475
b.1.2	8	23	71	867
b.121.4	30	83	116	2023
b.29.1	28	122	311	2867
b.36.1	5	15	33	499
b.40.4	4	15	39	809
b.47.1	32	42	70	1365
b.55.1	15	22	37	582
b.6.1	33	31	69	709
c.3.1	18	60	165	2860
c.47.1	21	38	80	1545
c.55.1	11	22	86	993
c.66.1	21	62	171	3398
c.67.1	23	102	108	5133
c.69.1	34	151	323	5732
c.94.1	50	77	198	2497
d.108.1	6	19	32	767
d.144.1	8	17	28	2454
d.15.1	14	19	16	239
d.3.1	35	69	103	1346
d.58.7	17	18	43	366
g.37.1	2	3	11	60
g.39.1	16	15	26	168
Average	19	43	92	1529

Table 4.8: The charts above show comparisons of average convergence times in seconds between our beam methods, the standard (no beam) forward-backward for 26 superfamilies of the SCOP 1.75 dataset. Inference was initialized using the MSA-induced pHMM. "Narrow," "medium," and "wide" indicate threshold settings of $\epsilon = \left[10^{-2}, 10^{-3}, 10^{-4}\right]$ and $\tilde{\theta} = \left[10^{-16}, 10^{-17}, 10^{-18}\right]$, where ϵ indicates the KL divergence beam threshold and $\tilde{\theta}$ indicates the auxiliary variable threshold.

the beam sampling method in [65] and constitutes a novel way of adapting auxiliary-variable sampling methods to the variational realm.

In experiments on both synthetically generated datasets and a subset of the SCOP 1.75 dataset, we show that our beam method significantly increases inference speed over standard pHMM inference methods and also maintains the model's ability to represent amino acid sequences. The ways of understanding pHMMs that we presented in this chapter have allowed us to construct more efficient inference methods, which have the potential to increase the speed of commonly-used bioinformatics applications.

Chapter 5: Local Profile HMMs for Classification

5.1 Introduction

In this chapter, I discuss a number of probabilistic models which attempt to discover sets of relevant subsequences within individual dataset sequences. Constructing models to discover sets of subsequences involves combining multiple instances of individual models each of which detect a single subsequence. These combined models not only represent a flexible method for understanding sequence data but also provide context useful for understanding concepts and structures discussed in the next chapters.

The models covered in this chapter are incorporate portions derived from the Profile HMM (pHMM) (see Chapter 2). These pHMM derivatives are called the Local Profile HMM (L-pHMM) and the Simplified Local pHMM (SL-pHMM). Both of these models extend the standard Profile HMM so that the model captures a single relevant subsequence within each sequence. We show how to combine the latent spaces of SL-pHMMs using two strategies. These involve (i) including an additional switching variable in the Switching SL-pHMM and (ii) constructing multinomial distributions over observed sequence elements through linear combinations of SL-pHMM hidden states in the Factorial SL-pHMM.

To use these combined models, we further extend them into joint models over sequences and associated labels. We do so by tying latent information from the models to a Sigmoid Belief Network-like structure, which is then used to define a probability distribution over label assignments. We validate basic operation of these joint models on sets of synthetically generated sequences.

5.2 Background

Understanding the probabilistic models in this section require knowledge of a variety of basic components. In this section, I review and define structures needed to build the more complicated models described later in this section.

5.2.1 Sigmoid Belief Networks

Sigmoid Belief Networks (SBNs) [77, 78] can be viewed as a probabilistic interpretation of the standard multilayer perceptron [79]. For the purposes of this work, we are interested in standard three-layer networks. These networks consist of a number of components: an input layer, $\mathbf{x} \in \mathbb{R}^d$, a hidden layer, $\mathbf{z} \in \{0,1\}^K$, and an output layer, $\mathbf{y} \in \{0,1\}^D$, a matrix of linear weights for each layer, $W^{(i)}$, where i indicates the layer, and a vector of biases, $\mathbf{b}^{(i)}$, for each layer (see Figure 5.1).

Each element, z_i, of the hidden layer, $\mathbf{z}$, can take on values of zero or one. These values are drawn from Bernoulli distributions whose parameters are computed from squashed, linear combinations of values, $\mathbf{x}$, from the first layers of the network. Specifically, $p(z_i|\mathbf{x}, W, b) = f(W^{(1)}_{i,:}\mathbf{x}+b^{(1)}_i)^{z_i}\left(1 - f(W^{(1)}_{i,:}\mathbf{x} + b^{(1)}_i)\right)^{1-z_i}$, where f indicates the logistic sigmoid function, $f(z) = (1+\exp(-z))^{-1}$. In this work, we are only interested in the case where our the output variables, $\mathbf{y}$, are binary, i.e. $y_i \in \{0,1\}\ \forall i$. The output vector, $\mathbf{y}$, is treated as a set of random variables drawn from a Bernoulli distribution whose parameters are a function of the middle layer, $\mathbf{z}$: $p(y_i|\mathbf{z}, W, b) = f(W^{(2)}_{i,:}\mathbf{z} + b^{(2)}_i)^{y_i}\left(1 - f(W^{(2)}_{i,:}\mathbf{z} + b^{(2)}_i)\right)^{1-y_i}$.

The joint probability of the hidden layer and the output layer, given the input layer can be used as a measure of quality of a prediction from the network over a dataset of N elements, $\mathbf{x}_{1:N}$, $\mathbf{y}_{1:N}$:

$$p(\mathbf{y}_{1:N}, \mathbf{z}_{1:N}|\mathbf{x}_{1:N}) = \prod_{n=1}^{N} \left(\prod_{j=1}^{D} p(y_{n,j}|\mathbf{z}_n, W, b) \right) \left(\prod_{k=1}^{K} p(z_{n,k}|\mathbf{x}_n, W, b) \right)$$

The marginal probability is given by

$$p(\mathbf{y}_{1:N}|\mathbf{x}_{1:N}) = \prod_{n=1}^{N} \sum_{\mathbf{z}_n} \left(\prod_{j=1}^{D} p(y_{n,j}|\mathbf{z}_n, W, b) \right) \left(\prod_{k=1}^{K} p(z_{n,k}|\mathbf{x}_n, W, b) \right)$$

To train the model, we would like to maximize the marginal likelihood with respect to the parameters W and b. However, there are 2^K terms in the sum over $\mathbf{z}_n$, so even for relatively small values of K, this becomes intractable. Thus, strategies to learn SBN parameters must resort to approximate methods. A common approximate method used for the SBN is variational inference [38, 78] (see Section 2.3.1 for an overview of variational methods).

We can construct a mean field variational bound on the SBN's marginal likelihood as follows:

$$
\begin{aligned}
\log p(\mathbf{y}_{1:N}|\mathbf{x}_{1:N}) &= \sum_{n=1}^{N} \log \sum_{\mathbf{z}_n} \prod_{j=1}^{D} p(y_{n,j}|\mathbf{z}_n, W, b) \prod_{k=1}^{K} p(z_{n,k}|\mathbf{x}_n, W, b) \\
&\geq \sum_{n=1}^{N} \sum_{\mathbf{z}_n} \left(\prod_{k=1}^{K} q(z_{n,k}) \right) \log \frac{\left(\prod_{j=1}^{D} p(y_{n,j}|\mathbf{z}_n, W, b)\right) \left(\prod_{k=1}^{K} p(z_{n,k}|\mathbf{x}_n, W, b)\right)}{\prod_{k=1}^{K} q(z_{n,k})} \\
&= \sum_{n=1}^{N} \sum_{j=1}^{D} E_{\prod_{k=1}^{K} q(z_{n,k})} \left[\log p(y_{n,j}|\mathbf{z}_n, W, b)\right] \\
&\quad + \sum_{n=1}^{N} \sum_{k=1}^{K} E_{q(z_{n,k})} \left[\log p(z_{n,k}|\mathbf{x}_n, W, b)\right] \\
&\quad + \sum_{n=1}^{N} \sum_{k=1}^{K} H\left(q(z_{n,k})\right)
\end{aligned}
$$

where $H(q(z)) = -\sum_z q(z) \log q(z)$ indicates the entropy. This variational bound allows us to decompose the expectation over the expression for each $E_{q(z_{n,k})} [\log p(z_{n,k}|\mathbf{x}_n, W, b)]$. However, the expectation does not decompose over the expressions for $E_{\prod_{k=1}^{K} q(z_{n,k})} [\log p(y_{n,j}|\mathbf{z}_n, W, b)]$. To enable such a decomposition it is possible to resort to an additional variational bound on the log of the logistic sigmoid. This bound must be applied on each instance of $\log p(y_{n,j}|\mathbf{z}_n, W, b)$. A number of these variational bounds are possible . In this work, we employ the quadratic bound described by Jaakkola et. al. [38].

5.3 Local Profile Hidden Markov Models

Hidden Markov Models (HMMs) (Section **2.1**) are a popular generative model over sequences. HMMs describe sequences by postulating generating each observation in the sequence from a mixture model and conditioning each mixture component on the mixture assignment from the previous observation. The Profile HMM (pHMM) (Section **2.1.1**) is a

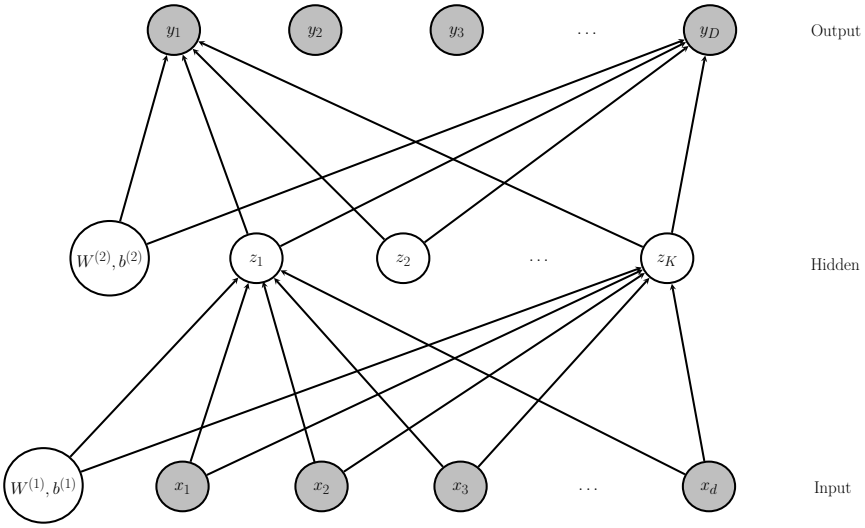


Figure 5.1: A diagram illustrating the dependency structure of the basic Sigmoid Belief Network.

type of HMM commonly used to model biological sequences. The probability distribution of a sequence under a pHMM is an implicit measurement of the distance of the sequence from an archetypal sequence encoded in the model. It is possible to modify the pHMM to target smaller subsequences from within a sequence with a small modification to the original hidden state structure (see Figure 5.2a). This modification includes a requirement symbols from the sequence generated by the model and occurring before or after the *Match* states that describe the archetypal sequence be generated from a background emission distribution indexed by a "Start" or "End" *Insert* state. We call the subsequences captured by the Match states in the model "relevant subsequences," and we call this model the Local-pHMM (L-pHMM).

A potential problem with the L-pHMM is that if the emission distributions from *Match* states have high variance (i.e. if a discrete distribution does not assign high probability to a small number of symbols from the alphabet), then high probability paths through the L-pHMM may include long chains of *Insert* states separating *Match* states. Thus, in certain cases, the L-pHMM may have difficulty capturing desired subsequence information. To increase both the ease of inference and the identifiability of subsequences during inference,

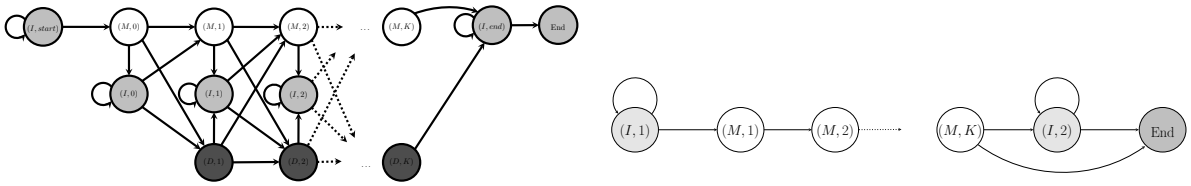
we can simplify the L-pHMM to a model which we call the Simplified Local pHMM (SL-pHMM).

The Simplified Local Profile HMM (SL-pHMM) postulates that an observed sequence contains a single contiguous relevant subsequence surrounded by symbols drawn from a background distribution. The SL-pHMM represents the relevant subsequence using two type of hidden states. A *Match* hidden state, denoted by the pair,(M, k), is associated with a position k in the relevant subsequence and emits an observed symbol from the multinomial distribution, $B_{k,:}$, where we use ":" as Matlab notation indicating the array consisting of all values over the final index. The relevant subsequence is surrounded symbols generated from a background distribution indexed by sequences of *Insert* hidden states. The *Insert* states, (I, Start) and (I, End), cause the model to emit a symbol from a background multinomial distribution B_0. We give a diagram of possible hidden states in Figure 5.2b.

The set of hidden states, $\mathbf{z}_n$, in the SL-pHMM can be summarized by a single variable t_n, which indicates the starting position of the relevant subsequence (hidden state $(M, 1)$) in sequence n. Transitions from *Insert* to *Match* states or the *Insert* to the End state occur with probability ω. Transitions from Insert to Insert state occur with probability $(1 - \omega)$. Transitions from *Match* states occur with probability 1 for states (M, k) , $k < K$ (the model transitions deterministically to the next symbol in the relevant subsequence). For *Match* state (M, K), transitions occur with probability ω to hidden state (I, End) and $(1-\omega)$ to the End hidden state to remove any locational preference within the sequence for the relevant subsequence. To simplify inference, unless otherwise indicated, we leave parameters ω and B_0 fixed and place Dirichlet priors on each $B_{k,:}$. We use A to represent the full transition probability matrix between SL-pHMM hidden states. We give the A matrix in terms of ω below in Equation 5.1:

Parameter	Definition
	General Notation
$\mathbf{x}_n$	the n^{th} observed sequence
y_n	the binary response variable associated with the n^{th} sequence, $y_n \in \{-1, 1\}$
$q(\dots)$	indicates a variational distribution over the parameters inside the parentheses
	SL-pHMM Variables
$t_{n,c}$	start of the relevant subsequence of the c^{th} SL-pHMM
$\mathbf{z}_{n,c}$	the set of hidden variables associated with the c^{th} pHMM used to generate the n^{th} observed sequence. $\mathbf{z}_{n,c}$ can be fully reconstructed given only $t_{n,c}$, the position of the first *Match* hidden state
A	fixed transition matrix for all SL-pHMMs
	Switching Model Variables
$\mathbf{s}_n$	the n^{th} sequence of switching variables
$A^{(s)}$	transition matrix on the switching variables $p(s'\|s) = A^{(s)}_{s,s'}$
B	an emissions matrix. $B_{c,k,m}$ is the probability of observed symbol m given that the symbol is generated from the c^{th} SL-pHMM's k^{th} relevant subsequence position
B_0	a fixed background distribution
α	Dirichlet prior parameter on $A^{(s)}$, $A^{(s)}_{s,:} \sim \text{Dirichlet}(\alpha)\ \forall s$
β	Dirichlet prior parameter on B, $B_{c,k,:} \sim \text{Dirichlet}(\beta)\ \forall c, k$
	Factorial Model Variables
$\mathbf{w}$	weights used to compute emission probabilities given the SL-pHMM hidden states
λ_w	regularization parameter on $\mathbf{w}$. i.e. $\mathbf{w} \sim \mathcal{N}\left(0, \lambda_w^{-1}\right)$
	Sigmoid Belief Network Variables
$e_{n,c}$	variables in the Sigmoid Belief Network's hidden layer, $e_{n,c} \in \{0, 1\}$
$\mathbf{v}$	Sigmoid Belief Network weights. $\mathbf{v}^{(1)}$ are the weights for the lower level of the network, with $\mathbf{v}^{(1)}_c$ and $v^{(1)}_{c,0}$ (bias term) associated with the c^{th} SL-pHMM. $\mathbf{v}^{(2)}$ and $v^{(2)}_0$ are associated with the top layer of the network.
λ_v	regularization parameter on $\mathbf{v}$. i.e. $\mathbf{v} \sim \mathcal{N}\left(0, \lambda_v^{-1}\right)$

Table 5.1: Description of parameters for combined pHMM models.



(a) (b)

Figure 5.2: Finite State Automata describing transitions between hidden states in the L-pHMM **(a)** and the SL-pHMM **(b)**. The *Match* hidden state for the k^{th} position of the relevant subsequence is indicated by the pair (M, k). Insert hidden states are indicated by (I, k), within the relevant subsequence and (I, Start) and (I, End) outside of it. *Delete* hidden states, (D, k), allow the model to skip the k^{th} *Match* hidden state.

$$A = \begin{array}{c|ccccccc} & (I,\text{Start}) & (M,1) & \dots & (M,K) & (I,\text{End}) & \text{End} \\ \hline (I,\text{Start}) & \omega & (1-\omega) & & & & \\ (M,1) & & & 1 & & & \\ \vdots & & & & \ddots & & \\ (M,K) & & & & & \omega & (1-\omega) \\ (I,\text{End}) & & & & & \omega & (1-\omega) \\ \text{End} & & & & & & \end{array} \tag{5.1}$$

A full description of model parameters is given in Table 5.1.

5.3.1 Combined Models using SL-pHMMs

It can often be advantageous to postulate that a set of components rather than a single component can contribute toward generating elements of a dataset. To apply this principle on datasets of discrete-valued sequences, we propose two methods for combining hidden state sequences from multiple SL-pHMMs. Both of these methods allow the model to find multiple relevant subsequences within a single sequence.

We call the first of these models the Switching SL-pHMM. As shown in Figure 5.3, C constituent SL-pHMMs contribute to generating an observed sequence. For the n^{th} sequence

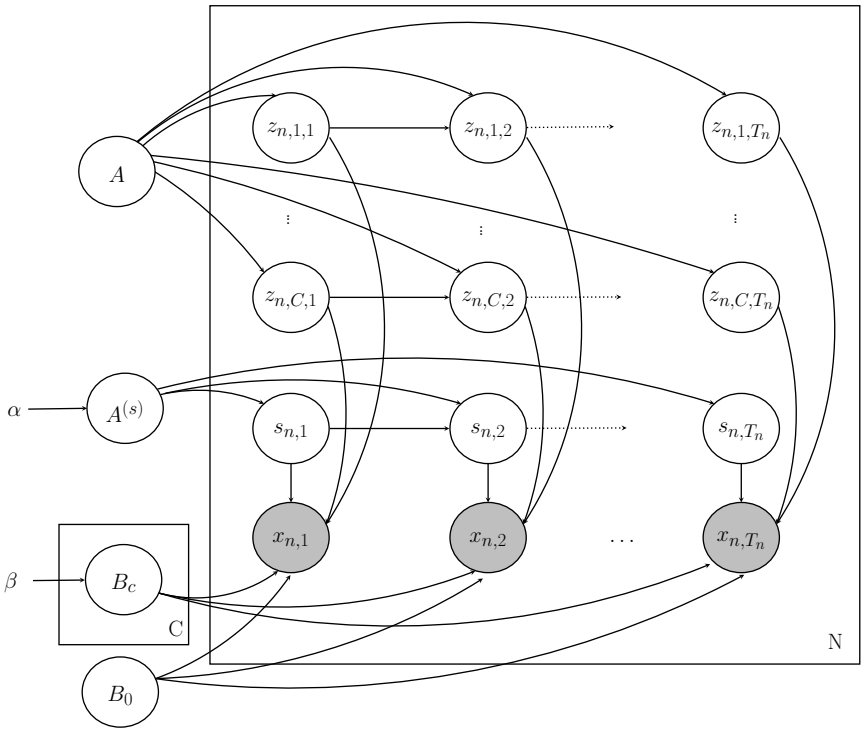


Figure 5.3: This figure shows a plate diagram depicting the dependency structure of the Switching SL-pHMM. C separate SL-pHMMs contribute toward generating an observed sequence. For each position, t, in observed sequence $\mathbf{x}_n$, a switching variable $s_{n,t}$ selects the SL-pHMM used to generate the observed symbol $x_{n,t}$.

in the training set, $\mathbf{x}_n$, of length T_n, the sequence of hidden states associated with the c^{th} constituent SL-pHMM is given by $\mathbf{z}_{n,c} = z_{n,c,1}, \ldots, z_{n,c,T_n}$. For these SL-pHMM models, we encode the entire sequence of hidden states associated with the c^{th} SL-pHMM and the n^{th} sequence by specifying the index of the first *Match* hidden state as $t_{n,c}$. Each SL-pHMM's contribution toward generating the observed sequence, $\mathbf{x}_n$, is mediated through a sequence of switching variables, $\mathbf{s}_n = s_{n,1}, \ldots, s_{n,T_n}$. We assume the Markov property on the switching variables and specify the transition probabilities between switching variables using the matrix $A^{(s)}$: $p(s_{n,t}|s_{n,t-1}) = A^{(s)}_{s_{n,t-1},s_{n,t}}$, with $p(\mathbf{s}_n|A^{(s)}) = \prod_{t=0}^{T} p(s_{n,t}|s_{n,t-1})$ and $s_{n,0}$ fixed to a dedicated start state. A full description of Switching SL-pHMM parameters is given in Table 5.1. Given both the switching variables and the SL-pHMM hidden states, the probability of an observed symbol at position t in the sequence is as follows:

$$p(x_{n,t}|s_{n,t}, z_{n,1:C,t}) = \begin{cases} B_{0,x_{n,t}} & z_{n,s_{n,t},t} \in \{(I, \text{Start}), (I, \text{End})\} \\ B_{s_{n,t},k,x_{n,t}} & z_{n,s_{n,t},t} \in \{(M,k)\},\ k \in [1 \ldots K] \end{cases} \tag{5.2}$$

Equivalently, in term of the t variables we have

$$p(\mathbf{x}_n|\mathbf{s}_n, t_{n,1:C}, B, B_0) = \prod_{t=1}^{T_n} B_{0,x_{n,t}}^{\mathbb{I}\left(t<t_{n,s_{n,t}} \vee t \geq t_{n,s_{n,t}}+K\right)} B_{s_{n,t},t_{n,s_{n,t}}-t+1,x_{n,t}}^{\mathbb{I}\left(t_{n,s_{n,t}} \leq t < t_{n,s_{n,t}}+K\right)} \tag{5.3}$$

$$= \prod_{t=1}^{T_n} \left(\prod_{c} B_{0,x_{n,t}}^{\mathbb{I}(c=s_{n,t})\mathbb{I}(t<t_{n,c} \vee t \geq t_{n,c}+K)} \right) \left(\prod_{c,k} B_{c,k,x_{n,t}}^{\mathbb{I}(c=s_{n,t})\mathbb{I}(k=t_{c,n}-t+1)} \right) \tag{5.4}$$

where B_0 is a background emissions distribution and $B_{c,k,m}$ is the probability of observed symbol m under the k^{th} Match hidden state of the c^{th} constituent SL-pHMM. The joint distribution of a set of sequences, hidden variables of the Switching SL-pHMM is given by

the following expression:

$$
\begin{aligned}
& p(\mathbf{x}_{1:N}, t_{1:N,1:C}, \mathbf{s}_{1:N} | B, A^{(s)}, A, B_0) \qquad (5.5) \\
= & \prod_n p(\mathbf{x}_n | \mathbf{s}_n, \mathbf{z}_{n,1:C}, B_{1:C}, B_0) p(\mathbf{s}_n | A^{(s)}) \prod_c p(\mathbf{z}_{n,c} | A)
\end{aligned}
$$

The switching structure allows a valid joint distribution over observed sequences from multiple sets of relevant subsequence positions given by the set of SL-pHMMs. A plate diagram showing the model's dependency structure is given in Figure 5.3.

We call the second model the Factorial SL-pHMM. Like the Switching SL-pHMM, the Factorial SL-pHMM combines hidden states from C SL-pHMMs. However, rather than using a separate switching state to index emissions probabilities, the Factorial SL-pHMM computes multinomial emission parameters for each position in each dataset sequence by passing a linear combination of SL-pHMM hidden states through a softmax function:

$$
p(x_{n,t} = m | z_{n,1:C,1:T_n}, \mathbf{w}) = \frac{\exp\left(\sum_c w^{\top}_{c,z_{n,c,t},m}\right)}{\sum_{m'} \exp\left(\sum_c w^{\top}_{c,z_{n,c,t},m'}\right)} \qquad (5.6)
$$

where $w_{c,k,m}$ indicates a weight associated with emitting character m in the alphabet using the k^{th} position of the c^{th} SL-pHMM. The Factorial SL-pHMM uses these weights, rather than directly defining an emission distribution, B, as in the switching model. The Factorial SL-pHMM has a potential advantage over the switching model, in that it has the ability to use the same set of SL-pHMM states to represent a sequence without the addition of extra hidden variables in the switching states.

$$\underbrace{0,0,0,0,0,0,1,0,0,0}_{x_{n,t_{n,c}}},\underbrace{0,0,0,0,0,0,0,0,0,1}_{x_{n,t_{n,c}+1}},\ldots,\underbrace{0,0,1,0,0,0,0,0,0,0}_{x_{n,t_{n,c}+K-1}}$$

Figure 5.4: An illustration of the feature vector, $\phi(x)$. The vector is composed of K indicator vectors, each describing an observed symbol the relevant subsequence.

5.4 Sequence Classification with Combinations of SL-pHMMs

The generative Switching and Factorial SL-pHMMs both employ a latent space on which we can condition the label assignment of over an entire sequence. Our basic approach toward sequence classification with these models involves defining joint models over both sequences and associated labels. The probability distributions over labels are conditioned on both the SL-pHMM hidden states and over elements of the sequence. We call these joint models over sequences and labels the Joint Switching SL-pHMM and Joint Factorial SL-pHMM.

Both joint models use a similar strategy to predict sequence labels: First, given sequence $\mathbf{x}_n$, a set of relevant subsequences beginning at the first *Match* state in each constituent SL-pHMM are selected. We indicate these start positions by the variable $t_{n,c}$. As mentioned in Section 5.3, for the SL-pHMM the entire sequence of hidden states, $\mathbf{z}_{n,c}$, can be reconstructed given only $t_{n,c}$. Second, a Sigmoid Belief Network (SBN) is given the C relevant subsequences as input (i.e., each subsequence $x_{t_{n,c}:t_{n,c}+K-1}$), and the output label of the network is set to the sequence label y_n. A diagram of the feature vector extracted from the c^{th} SL-pHMM for use as input to the SBN is given in Figure 5.4.

The probability over sequences, labels, and hidden states in the Switching model is given below in Equation 5.7:

$$p(y_{1:N}, \mathbf{x}_{1:N}, \mathbf{z}_{1:N,1:C}, \mathbf{s}_{1:N}, \mathbf{v}|B, A^{(s)}, A, B_0, \lambda_v) =$$

$$p(\mathbf{v}|\lambda_v) \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{z}_{n,1:C}, \mathbf{s}_n, B, B_0) p(\mathbf{s}_n|A^{(s)}) p(\mathbf{z}_{n,1:C}|A) p(y_n|e_{n,1:C}, \mathbf{v}) \prod_c p(e_{n,c}|\mathbf{x}_n, \mathbf{z}_{n,c}, \mathbf{v}) \tag{5.7}$$

where the Sigmoid Belief Network weights are encoded in the vector, $\mathbf{v}$, and the hidden nodes in the SBN are given by $e_{n,1:C}$ for sequence n. For the factorial model, the probability can be expressed similarly as the generative portion of the model multiplied by the SBN over labels:

$$p(y_{1:N}, \mathbf{x}_{1:N}, \mathbf{z}_{1:N,1:C}, \mathbf{w}, \mathbf{v}|A, \lambda_w, \lambda_v) =$$

$$p(\mathbf{w}|\lambda_w) p(\mathbf{v}|\lambda_v) \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{z}_{n,1:C}, \mathbf{w}) p(\mathbf{z}_{n,1:C}|A) p(y_n|e_{n,1:C}, \mathbf{v}) \prod_c p(e_{n,c}|\mathbf{x}_n, \mathbf{z}_{n,c}, \mathbf{v}) \tag{5.8}$$

5.4.1 Inference

Because of the interdependence between relevant subsequence positions, $t_{n,c}$, across constituent SL-pHMMs, switching variables, $\mathbf{s}_n$, and SBN hidden states, $e_{n,c}$, we use a variational Bayesian approach to compute approximate maximum-a-posteriori solutions with respect to the model parameters (see Table 5.1). Variational inference (see Chapter 2) constructs a lower bound on the marginal likelihood of the model by postulating a set of variational distributions where an independence assumption is made over latent variables in the model. These independence assumptions allow the expectation of the latent variables in the model to be computed relatively easily, leading to tractable inference. Algorithm 5.1 and 5.2 gives the steps used for the variational inference during the training phase of the Joint

Algorithm 5.1 Training Phase - Variational Inference for the Joint Switching SL-pHMM

Initialize inference with $\mathbf{v}$, $\mathbf{w}$, $A^{(s)}$, and B sampled from their prior distributions. Initialize $q(t_{n,c})$ and $q(e_{n,c})$ to uniform distributions. Below, $\tau()$ indicates the following function: $\tau(\xi) \stackrel{\text{def}}{=} \frac{1}{4\xi}\tanh\left(\frac{\xi}{2}\right)$ [38].

- Repeat until the variational bound converges:

1. Maximize with respect to $q(\mathbf{s}_n)$ for all n

$$q(\mathbf{s}_n) \propto \prod_{t=1}^{T_n} A^{(s)}_{s_{n,t-1},s_{n,t}} B_{0,x_{n,t}}^{E_{q(t_{s_{n,t},n})}\left[\mathbb{I}\left(t<t_{s_{n,t},n}\vee t\geq t_{c,n}+K\right)\right]} \prod_k B_{s_{n,t},k,x_{n,t}}^{E_{q(t_{s_{n,t},n})}\left[\mathbb{I}\left(k=t_{s_{n,t},n}-t+1\right)\right]}$$

2. Maximize with respect to $q(t_{n,c})$ for all n, c

$$q(t_{n,c}) \propto \exp\left(-E_{q(e_{n,c})}\left[e_{n,c}\right]\left(v^{(1)}_{c,0}+\sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right)\left(1+\exp\left(-v^{(1)}_{c,0}-\sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right)^{-1} \times$$
$$\prod_{t=1}^{T_n} B_{0,x_{n,t}}^{E_{q(\mathbf{s}_n)}[\mathbb{I}(c=s_{n,t})]\mathbb{I}(t<t_{c,n}\vee t\geq t_{c,n}+K)} B_{c,k,x_{n,t}}^{E_{q(\mathbf{s}_n)}[\mathbb{I}(c=s_{n,t})]\mathbb{I}(k=t_{c,n}-t+1)}$$

3. Maximize with respect to $q(e_{n,c})$ for all n, c

$$q(e_{n,c}) \propto \exp\left(e_{n,c}\left(\frac{1}{2}y_n v^{(2)}_c - \tau\left(\zeta_n\right)\left(2v^{(2)}_c\left(v^{(2)}_0+\sum_{c'\neq c} E_{q(e_{n,c'})}\left[e_{n,c'}\right]v^{(2)}_{c'}\right)+\left(v^{(2)}_c\right)^2\right)\right.\right.$$
$$\left.\left.-E_{q(t_{n,c})}\left[v^{(1)}_{c,0}+\sum_{k=1}^{K} v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right]\right)\right)$$

Switching SL-pHMM. A variational inference procedure is also needed to predict labels of test set sequences because predictions are a function of the variational bound (Algorithm 5.1). Variational distributions over latent variables in each model are indicated by $q(\omega)$, where ω is a latent variable. Both the training and prediction phases of the Joint Factorial SL-pHMM are follow a similar outline to that of the Joint Switching SL-pHMM. Derivations of the variational update steps for both models are given in Appendix C.

5.4.2 Experimental Analysis: Classifying Synthetic Sequences

In sets of experiments on a set of synthetically generated discrete sequences, we verify that the Joint Switching and Factorial SL-pHMM models can simultaneously extract subsequence

Algorithm 5.2 Training Phase - Variational Inference for the Joint Switching SL-pHMM (continued)

1. Maximize with respect to $A_c^{(s)}$ for all c

$$A_{c,c'}^{(s)} \propto E_{q(\mathbf{s}_n)}\left[\mathbb{I}(s_{n,t-1}=c, s_{n,t}=c')\right]+\alpha-1$$

5. Maximize with respect to $B_{c,k}$ for all c,k

$$B_{c,k,m} \propto \sum_n \sum_t E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\right] E_{q(t_{c,n})}\left[\mathbb{I}\left(t_{c,n}-t+1=k\right)\right]\mathbb{I}\left(x_{n,t}=m\right)+\beta-1$$

6. Maximize with respect to $\mathbf{v}^{(1)}$ using L-BFGS [80] with the following gradients

$$\begin{aligned}\frac{\partial \mathcal{F}(v_c^{(1)})}{\partial v_{c,k,m}^{(1)}} &= -\sum_n E_{q(e_{n,c})}\left[e_{n,c}\right] E_{q(t_{n,c})}\left[\mathbb{I}\left(x_{n,t_{n,c}+k}=m\right)\right] \\ &\quad +\sum_n\left(1-E_{q(t_{n,c})}\left[\sigma\left(v_{c,0}^{(1)}+\sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right]\right) \\ &\quad -\lambda v_{c,k,m}^{(1)}\end{aligned}$$

$$\frac{\partial \mathcal{F}(v_{c,0}^{(1)})}{\partial v_{c,0}^{(1)}} = \sum_n\left(1-E_{q(t_{n,c})}\left[\sigma\left(v_{c,0}^{(1)}+\sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right]\right)-\sum_n E_{q(e_{n,c})}\left[e_{n,c}\right]-\lambda v_{c,0}^{(1)}$$

7. Maximize with respect to the variational parameter,ζ_n, for all n

$$\zeta_n = y_n\left(v_0^{(2)}+\sum_c e_{n,c} v_c^{(2)}\right)$$

8. Maximize with respect to $\mathbf{v}^{(2)}$

$$\mathbf{v}^{(2)} = \left(\lambda_v I+\sum_n \tau\left(\zeta_n\right) E_{\prod_c q(e_{n,c})}\left[\begin{bmatrix} e_{n,1:C} \\ 1\end{bmatrix}\begin{bmatrix} e_{n,1:C} \\ 1\end{bmatrix}^\top\right]\right)^{-1}\left(\sum_n \frac{1}{2} y_n \begin{bmatrix} \bar{e}_{n,1:C} \\ 1\end{bmatrix}\right)$$

Algorithm 5.3 Prediction Phase - Variational Inference for the Joint Switching SL-pHMM

Initialize $q(t_{n,c})$ and $q(e_{n,c})$ to uniform distributions. Values of $\mathbf{v}$, $\mathbf{w}$, $A^{(s)}$, and B set using values from the training phase.

- Repeat until the variational bound converges:

1. Maximize with respect to $q(\mathbf{s}_n)$ for all n

$$q(\mathbf{s}_n) \propto \prod_{t=1}^{T_n} A^{(s)}_{s_{n,t-1},s_{n,t}} B_{0,x_{n,t}}^{E_{q(t_{s_{n,t},n})}\left[\mathbb{I}\left(t<t_{s_{n,t},n} \vee t \geq t_{c,n}+K\right)\right]} \prod_k B_{s_{n,t},k,x_{n,t}}^{E_{q(t_{s_{n,t},n})}\left[\mathbb{I}\left(k=t_{s_{n,t},n}-t+1\right)\right]}$$

2. Maximize with respect to $q(t_{n,c})$ for all n, c

$$q(t_{n,c}) \propto \exp\left(-E_{q(e_{n,c})}\left[e_{n,c}\right]\left(v^{(1)}_{c,0} + \sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right)\left(1+\exp\left(-v^{(1)}_{c,0} - \sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right)^{-1} \times$$
$$\prod_{t=1}^{T_n} B_{0,x_{n,t}}^{E_{q(\mathbf{s}_n)}[\mathbb{I}(c=s_{n,t})]\mathbb{I}(t<t_{c,n} \vee t \geq t_{c,n}+K)} B_{c,k,x_{n,t}}^{E_{q(\mathbf{s}_n)}[\mathbb{I}(c=s_{n,t})]\mathbb{I}(k=t_{c,n}-t+1)}$$

3. Maximize with respect to $q(e_{n,c})$ for all n, c

$$q(e_{n,c}) \propto \exp\left(-e_{n,c} \sum_{t_{n,c}=1}^{T_n-K} q(t_{n,c})\left(v^{(1)}_{c,0} + \sum_{k=1}^{K} v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right)$$

- Predict $y_n = \text{sign}\left(\left(\mathbf{v}^{(2)}\right)^\top \begin{bmatrix} \bar{e}_{n,1:C} \\ 1 \end{bmatrix}\right)$

features and use these features to construct a decision boundary between positive and negative sets of sequences. Our synthetic dataset was designed to ensure that our assumption that relevant subsequences are highly correlated with class labels holds. The datasets consisted of sequences generated to contain up to three non-overlapping motifs. These motifs consisted of 5 repetitions of "a," "r," or "n," with a 10% chance at each motif position for a motif character to be replaced by a character generated uniformly from the full sequence alphabet of 20 possible characters. Non-motif sequence elements were chosen uniformly at random from the full 20-characters alphabet. Sequence lengths were generated uniformly at random from a range of 25 to 75. A sequence was selected to be in the positive category if all three possible relevant subsequences occurred within the sequence. A sequence was selected to be in the negative category if it contained either one or two relevant subsequences.

We ran classification experiments on this dataset, using both the Joint Switching SL-pHMM and the Joint Factorial SL-pHMM, varying both parameters of the models and the number of sequences generated. Figures 5.5a and 5.5b show results from experiments using the Joint Switching SL-pHMM and Joint Factorial SL-pHMM for different values of regularization parameters as the size of the dataset increases. In the switching model, we selected the β parameter, the Dirichlet prior on the emission distribution B, from the set $\{1.01, 1.1, 1.5, 2.0\}$. In the factorial model, we selected the λ_w parameter, the precision parameter on the Normally-distributed weights $\mathbf{w}$, from the set $\{.1, .5, 1.0, 10.0\}$. We trained each model on our synthetically generated datasets using the following numbers of training and testing examples: $\{5, 10, 25, 50, 100, 200\}$. All reported results are averaged over 10 trials with the same settings. To evaluate the robustness of our model with respect to preset parameters differing from the ground truth, we ran experiments using the number of relevant subsequences, $C \in \{2, 3, 6\}$, where the values 2 and 6 were smaller and larger than the ground truth number of relevant subsequences respectively.

As shown in Figures 5.5a and 5.5b, smaller values of β seemed to lead to better performance in the switching model, while larger values of λ_w seemed to lead to better performance in the factorial model. The graphs also indicate that the optimal values of these parameters

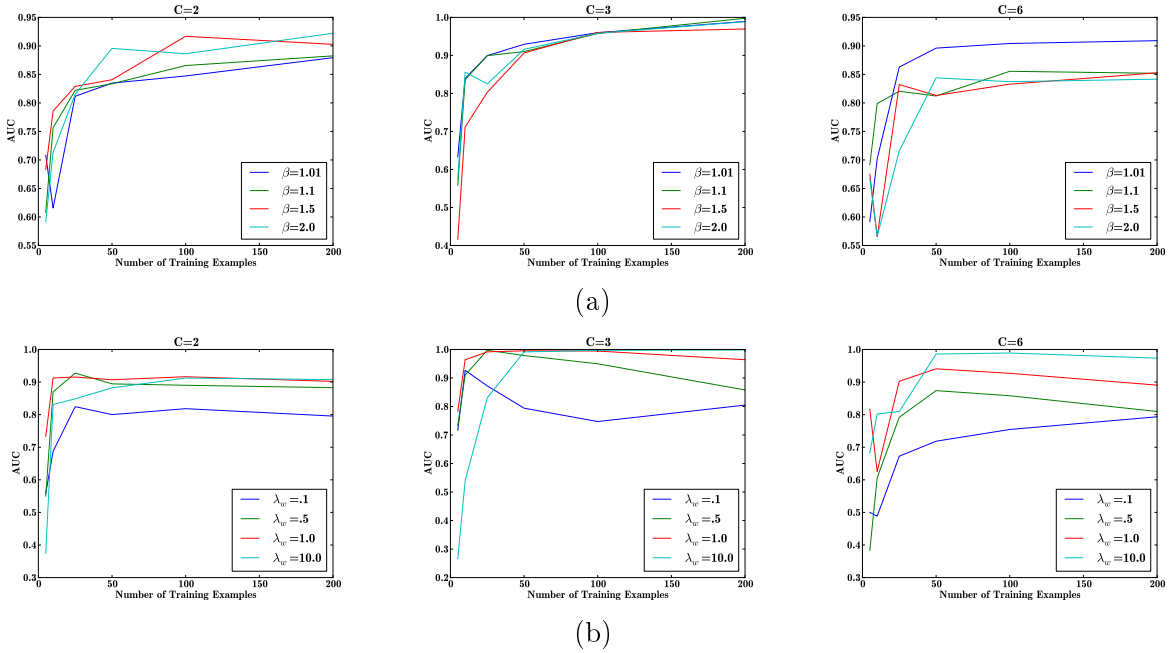


Figure 5.5: Average area under the ROC curve (AUC) on synthetically generated test set data. Results for the Joint Switching SL-pHMM are given in **(a)** while results for the Joint Factorial SL-pHMM are given in **(b)**. Each graph shows the variation in AUC as the number of training set examples increases. Different graphs show results from models with different numbers of constituent SL-pHMMs (indicated by C). Individual lines in each graph indicate different values of the regularization parameter. For the switching model, we varied the Dirichlet prior parameter on the emissions distributions, β, while for the factorial model, we varied, λ_w, the precision parameter on the weights, $\mathbf{w}$.

is correlated with the size of the dataset. Our experiments also indicate that the factorial model outperforms the switching model in the sense that the factorial model produced higher AUCs both with smaller training sets and also when the number of ground truth relevant subsequences was smaller than the preset C parameter in the model.

5.5 Conclusions

In this chapter, we have introduced a series of probabilistic models to combine latent structure from multiple HMMs. First, we introduced the Local pHMM and the Simplified Local pHMM, which modify the Profile HMM to attempt to capture relevant subsequences. We

then extended the SL-pHMM to the Switching and Factorial SL-pHMMs, which attempt to discover multiple relevant subsequences within a sequence. Finally, we showed how to use this latent subsequence information in conjunction with a structure similar to the Sigmoid Belief Network to predict sequence labels. Experiments on synthetic data validate that these models can both discover subsequence features within sequence data and are able to use these subsequence features to predict sequence categories.

Chapter 6: Relevant Subsequence Detection with Sparse Dictionary Learning

Sparse Dictionary Learning has recently become popular for discovering latent components that can be used to reconstruct elements in a dataset. It has seen particular success in computer vision where it has been incorporated into solutions for problems in image reconstruction, in-painting, and classification [81–86].

Sparse Dictionary Learning's success in computer vision makes it a promising candidate for discovering patterns in sequence data. Sequence data, however, is not natively accepted by the Sparse Dictionary Learning model: Sequences can be of variable length, and patterns within sequences are not associated with a fixed set of indices. These patterns can occur at any point within a sequence and can be repeated multiple times. A strategy for making sequence data more manageable is to extract all subsequences of length K from the original dataset and use these as input to a Sparse Dictionary Learning algorithm. This strategy poses a problem, however, because self-similar patterns within sequences are over-represented.

In this chapter, I propose an alternative to this standard subsequence dataset approach, which we call Relevant Subsequence Dictionary Learning (RS-DL). Our method involves constructing separate dictionaries for each sequence in a dataset from shared "relevant subsequence patterns." This structured dictionary can be used to pick out shared information from sets of sequences and can be learned using standard optimization methods. An important contribution of this chapter is in showing how to efficiently run the LARS algorithm given our relevant subsequence dictionary structure.

To show the utility of the RS-DL model, we run experiments on several types of sequence data. Running our algorithm on synthetic sets of sequences with discrete-valued elements, continuous electrocardiogram data, and text datasets, we show that our RS-DL model is

effective for discovering repeated patterns meaningful to humans (also called motifs). We also show that RS-DL is effective for classification. To do so, we use RS-DL to extract features from time-series data and show that these features can reduce classification error compared to standard methods.

6.1 Background: Sparse Dictionary Learning

Sparse Dictionary Learning is a method of decomposing a dataset into the product of a dictionary matrix and a sparse vector of coefficients. Here we represent the N dataset vectors as $x_{1:N}$, with the n^{th} vector given by $x_n \in \mathbb{R}^d$, the dictionary matrix as $W \in \mathbb{R}^{d \times C}$, and the set of N sparse vectors of coefficients as $\alpha_{1:N}$, $\alpha_n \in \mathbb{R}^C$. The number of dictionary columns, C, is chosen beforehand. The Sparse Dictionary Learning objective is typically defined as follows:

$$f(x_n; W) = \min_{\alpha_n} \underbrace{\frac{1}{2}||x_n - W\alpha_n||_2^2}_{\text{loss}} + \underbrace{\lambda\psi(\alpha_n)}_{\text{sparsity-inducing term}} \tag{6.1}$$

where ψ is a regularization function, typically an L_1 norm.

There has been a significant amount of research to develop efficient algorithms for solving the Sparse Dictionary Learning problem [83]. These algorithms typically consist of repeating two optimization steps. In the first step, a linear regression problem with the sparsity-inducing regularization term is solved to compute $\alpha_n = \min_{\alpha_n} ||x_n - W\alpha_n||_2^2 + \lambda\psi(\alpha_n)$ given the current value of the dictionary, W, for each example in the dataset. Common algorithms to perform this task include pursuit algorithms [87], Least Angle Regression (LARS) [88], coordinate-wise descent methods [89], and proximal methods [90].

In the second step, the value of the dictionary, W, is updated given the current minimum values of α_n. As with methods for optimizing with respect to the α's, it is possible to use any of a number of different methods to minimize with respect to the dictionary. These

methods include K-SVD [87] (which also updates the α terms), stochastic gradient methods [86], and solutions of the dual problem (for a constrained dictionary) [85], among others.

Sparse Dictionary Learning is similar to other decomposition techniques like Principal Component Analysis (PCA). PCA decomposes elements of a dataset into linear combinations of vectors from an orthogonal basis. Sparse Dictionary Learning differs from PCA in two important respects. First, dictionary columns are non-orthogonal, and second, the sparsity inducing regularization term forces only a small number of columns to be used for reconstruction. These characteristics can be advantageous compared to PCA because the sparsity inducing term allows the dictionary to include more columns that the dimensionality of the vector being reconstructed [83]. This "overcomplete" representation allows a large number of patterns to be found in the data but only a small number of these patterns are used to reconstruct each data element.

6.1.1 Least Angle Regression (LARS) for RS-DL

Least Angle Regression (LARS) [88] is a method for solving L_1 and L_0 regularized linear least squares problems efficiently. We describe LARS here briefly because the factorial formulation requires that a number of matrix multiplication steps in the original LARS algorithm be modified for efficient computation.

LARS solves the problem

$$\min_{\alpha} \quad \frac{1}{2}||x - W\alpha||^2 \tag{6.2}$$

by constructing an incremental approximation of x, which we define as $\mu_{\mathcal{A}}$, by taking steps along a vector that is equally correlated with an active set, $\mathcal{A}$, of columns of W. This active set is incrementally constructed by choosing the next column of W that is most correlated with the residual, $y - \mu_{\mathcal{A}}$. Because the approximation $\mu_{\mathcal{A}}$ is constructed incrementally, LARS is well suited to computing solutions where α is sparse, by stopping

iteration early, and can be extended to the L_1-regularized case ($\min_\alpha \frac{1}{2}||x - W\alpha||^2 + \lambda|\alpha|$), described later in this section.

To compute this next maximally correlated column, a vector of correlations is defined as follows:

$$\vec{c} \quad = \quad W^\top (x - \mu_\mathcal{A}) \tag{6.3}$$

The active set is defined as

$$\mathcal{A} \quad = \quad \{c_i : c_i = \max |\mathbf{c}|\} \tag{6.4}$$

With the maximum correlation, $c_\mathcal{A}$, defined as

$$c_A \quad \doteq \quad \max |\mathbf{c}| \tag{6.5}$$

and the active columns, $W_\mathcal{A}$, defined as

$$W_\mathcal{A} \quad \stackrel{\text{def}}{=} \quad [\text{sgn}\,(c_i)\, W_{:,i} | c_i \in \mathcal{A}] \tag{6.6}$$

At each iteration of LARS, $\mu_\mathcal{A}$ is incremented along an equi-angular unit vector, $u_\mathcal{A}$, satisfying

$$W_\mathcal{A}^\top u_\mathcal{A} \quad \propto \quad \vec{1} \tag{6.7}$$

$$u_\mathcal{A}^\top u_\mathcal{A} \quad = \quad \vec{1} \tag{6.8}$$

$u_{\mathcal{A}}$ can be computed as follows:

$$G_{\mathcal{A}}^{-1} = \left(W_{\mathcal{A}}^{\top} W_{\mathcal{A}}\right)^{-1} \tag{6.9}$$

$$a_{\mathcal{A}} = \left(\vec{1}^{\top} G_{\mathcal{A}}^{-1} \vec{1}\right)^{-\frac{1}{2}} \tag{6.10}$$

$$\omega_{\mathcal{A}} = a_{\mathcal{A}} G_{\mathcal{A}}^{-1} \vec{1} \tag{6.11}$$

$$u_{\mathcal{A}} = W_{\mathcal{A}} \omega_{\mathcal{A}} \tag{6.12}$$

At each step of the LARS algorithm attempts to find a real number γ such that for $\mu_{\mathcal{A}^+} = \mu_{\mathcal{A}} + \gamma u_{\mathcal{A}}$, no element of $\mathbf{c}^+ = W^{\top}(x - \mu_{\mathcal{A}^+})$ is greater than $c_{\mathcal{A}^+}$ (the maximum correlation with the residual should decrease at every iteration), where $\mathcal{A}^+$ indicates the active set after one step of the algorithm - i.e. $\mathcal{A}^+ = \left\{\mathcal{A} + j^{(new)}\right\}$ where $j^{(new)}$ indexes a new column of W.

To find the value of γ that gives the next correlation, observe that

$$c_j^+ = W_{:,j}(x - \mu_{\mathcal{A}^+}) \tag{6.13}$$

$$= W_{:,j}(y - \mu_{\mathcal{A}} - \gamma u_{\mathcal{A}}) \tag{6.14}$$

$$= c_j - \gamma a_j \tag{6.15}$$

$$= \operatorname{sgn}(c_j)(c_{\mathcal{A}} - \gamma a_{\mathcal{A}}) \quad \text{if } j \in \mathcal{A} \tag{6.16}$$

where c_j is from the correlation vector, $\mathbf{c}$, defined in Equation 6.3 and $\mathbf{a} = W^{\top} u_{\mathcal{A}}$. We solve the following equation to find the smallest γ that makes the correlation of the previous active set equal to the current correlation:

$$
\begin{aligned}
|s_{j(\mathcal{A})}\left(c_{\mathcal{A}} - \gamma a_{\mathcal{A}}\right)| &= |c_j - \gamma a_j| && (6.17)\\
c_{\mathcal{A}} - \gamma a_{\mathcal{A}} &= c_j - \gamma a_j && (6.18)\\
&\text{or} && (6.19)\\
c_{\mathcal{A}} - \gamma a_{\mathcal{A}} &= \gamma a_j - c_j && (6.20)
\end{aligned}
$$

$a_{\mathcal{A}}$ and $c_{\mathcal{A}}$ must be positive ($a_{\mathcal{A}}$ because G^{-1} is positive definite, and $c_{\mathcal{A}}$ from its definition), so γ must be positive if we want to reduce the maximum correlation at each step of the algorithm, giving

$$
\gamma = \min_j{}^{+}\left(\frac{c_{\mathcal{A}} - c_j}{a_{\mathcal{A}} - a_j}, \frac{c_{\mathcal{A}} + c_j}{a_{\mathcal{A}} + a_j}\right) \tag{6.21}
$$

where $\min_j{}^{+}$ indicates taking the minimum over all non-negative elements.

After taking the next step on the LARS path, we update α using the definitions $\mu = W\alpha$ and $u_{\mathcal{A}} = W_{\mathcal{A}} G_{\mathcal{A}}^{-1} W_{\mathcal{A}} \vec{1}$:

$$
\begin{aligned}
\mu_{\mathcal{A}^+} &= \mu_{\mathcal{A}} + \gamma u_{\mathcal{A}} && (6.22)\\
&= W_{\mathcal{A}}\alpha + \gamma W_{\mathcal{A}} G_{\mathcal{A}}^{-1} a_{\mathcal{A}} \vec{1} && (6.23)\\
&= W_{\mathcal{A}}(\alpha + \gamma d) && (6.24)
\end{aligned}
$$

The vector α is therefore updated as $\alpha \leftarrow \alpha + \gamma d$, where $d_j = \begin{cases} \operatorname{sgn}(c_j)\left(G_{\mathcal{A}}^{-1} a_{\mathcal{A}} \vec{1}\right)_{j(\mathcal{A})} & j \in \mathcal{A} \\ 0 & j \notin \mathcal{A} \end{cases}$

.

To extend LARS to the L_1-regularized case, we can compute optimality conditions in

the L_1-regularized problem by observing the set of subgradients with respect to α at the optimum:

$$\begin{cases} W_{:,i}^{\top}\left(x - W\alpha^{\star}\right) = \lambda \text{sgn}\left(\alpha_i^{\star}\right) & \alpha_i^{\star} \neq 0 \\ W_{:,i}^{\top}\left(x - W\alpha^{\star}\right) \leq \lambda & \alpha_i^{\star} = 0 \end{cases}$$

These conditions imply that the sign of the maximum correlations must equal the sign of each associated value of α at each step of the LARS algorithm. Therefore, to maintain this condition, we must verify that $\text{sgn}\left(\alpha_i\right) = \text{sgn}\left(\alpha_i + \gamma d_i\right) \quad \forall i$. If the condition does not hold, then we then choose a new γ so that the violating example is removed from the active set. That is, the new value of γ must enforce the condition $\alpha_i + \gamma d_i = 0$, leading to the update $\gamma = \min_i -\frac{\alpha_i}{d_i}$.

6.2 Relevant Subsequence Dictionary Learning

We propose an approach, which we call Relevant Subsequence Dictionary Learning (RS-DL)[1], to extend Sparse Dictionary Learning to the domain of sequences. Sequences differ from more-standard vector representations in that they can vary in length across a single dataset, and patterns within sequences can occur at any position rather than being associated with a fixed set of indices. To account for these characteristics of sequence data, RS-DL constructs dictionaries from C different subsequence dictionary components. We refer to these constituent components as "relevant subsequence patterns" and indicate these patterns by the two-dimensional array, $\mathbf{v}$, of size $C \times K$, where $v_{c,k}$ is a value associated with the k^{th} position in the c^{th} relevant subsequence pattern.

Unlike standard Sparse Dictionary Learning, RS-DL constructs a separate dictionary, W_n, for each sequence, x_n, in a dataset by positioning relevant subsequence parameters,

[1] We have made code available at http://cs.gmu.edu/∼sblasiak/RS-DL.tar.gz

$v_{c,:}$, so that they cover all possible subsequence starting positions. Positions in dictionary columns that are not given by relevant subsequence parameters are set to zero.

Figure 6.1 shows how the array of constituent relevant subsequence patterns, $\mathbf{v}$, is used to construct W_n, the dictionary associated with sequence x_n. Table 6.1 gives descriptions of all parameters in the RS-DL formulation. After building the dictionaries, W_n, we are left with an objective very similar to that of standard Sparse Dictionary Learning:

$$f(x_{1:N}; \mathbf{v}) = \sum_{n=1}^{N} \min_{\alpha_n} \frac{1}{2}||x_n - W_n\alpha_n||_2^2 + \lambda|\alpha_n|_1 \tag{6.25}$$

To optimize with respect to this objective, we employ a stochastic gradient descent procedure that alternatively solves first for each α_n, then takes a gradient step with respect to the array, $\mathbf{v}$. This procedure is similar to existing Sparse Dictionary Learning optimization algorithms. For the optimization step with respect to α_n, we apply a variation of the Least Angle Regression (LARS) [88] algorithm. The LARS algorithm requires computing a number of matrix products involving W_n. However, computing these matrix products directly would be inefficient, as each W_n matrix is of size $O(T_n) \times O(CT_n)$. To improve performance, we can take advantage of the sparse construction of each W_n, allowing these products to be computed more quickly, as we describe in the next section.

After computing each new value of α_n, the RS-DL algorithm takes a single stochastic gradient step in $\mathbf{v}$: $\mathbf{v}^{i+1} \leftarrow \mathbf{v}^i - \left(\frac{\gamma}{i+1}\right) \frac{\partial \frac{1}{2}||x_n - W_n^i \alpha_n||_2^2}{\partial \mathbf{v}}$, where γ is a learning rate term. We found empirically that, for RS-DL, this single stochastic gradient step is often faster than solving for $\mathbf{v}$ after accumulating information from a batch of α_n's as in Mairal et. al. [83].

6.2.1 Efficiently running the LARS algorithm with RS-DL

RS-DL involves constructing dictionaries, W_n of size $O(T_n) \times O(CT_n)$, many of whose entries are set to zero. If not carefully handled, this large, sparse matrix can cause the RS-DL

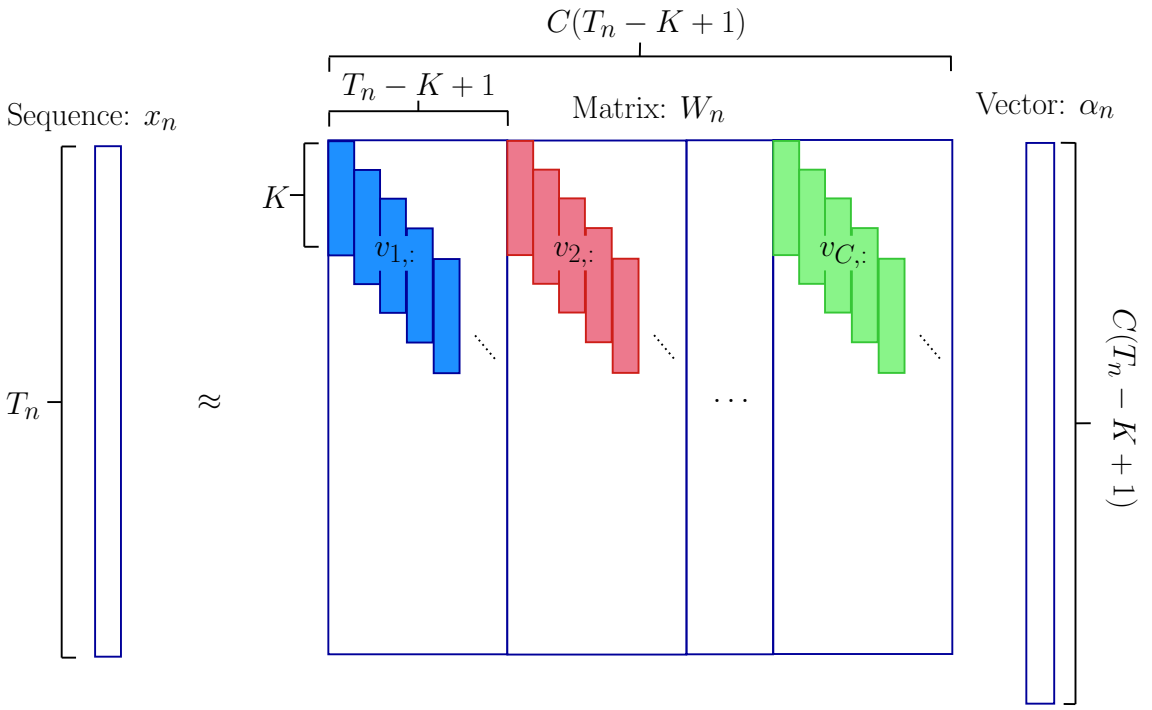


Figure 6.1: The figure above illustrates the Relevant Subsequence Dictionary Learning setup. The matrix W_n is constructed from the weights $v_{c,k}$ in C blocks so that the relevant subsequence patterns given by each $v_{c,:}$ are arranged to create dictionary elements (columns of W_n) that cover every K length subsequence of the sequence x_n (illustrated in blue). White areas of the W_n matrix are set to zero. The vector α_n is L_1-regularized to select a small number of dictionary columns associated with positioned relevant subsequences patterns. The α_n-weighted sum of these positioned relevant subsequences patterns approximates x_n.

training algorithm to operate inefficiently. The LARS algorithm constitutes a major substep in RS-DL training and requires a number of computations involving W_n. Efficiency of these computations can be considerably improved by taking W_n's sparse construction from elements of the array $\mathbf{v}$ into account. Three LARS computations involving the dictionaries, W_n are (i) the matrix-matrix product $(W_n)_{\mathcal{A}}^{\top}(W_n)_{\mathcal{A}}$, (ii) the matrix-vector product $(W_n)_{\mathcal{A}}\,\omega_{\mathcal{A}}$, and (iii) the matrix-vector product $W_n^{\top}u$, where $\mathcal{A}$ indicates an active set of columns (the number of non-zero components of α), $(W_n)_{\mathcal{A}}$ indicates a matrix constructed from this active set, $\omega_{\mathcal{A}}$ is a vector of length $|\mathcal{A}|$, and u is a vector of length T_n. Below, $t(i)$ indicates the index of the start of the subsequence associated with the i^{th} column of W_n (see Figure

Parameter	Definition
M	The size of the alphabet for discrete sequences. We omit the M parameter when dealing with continuous-valued sequences.
$\mathbf{x}_{1:N}$	A set of N observed sequences. Individual sequences, x_n, can be of variable length. Discrete sequences are expanded to M concatenated sequences of T_n indicator variables.
T_n	The length of the n^{th} sequence.
$\alpha_{1:N}$	A set of N vectors. Each α_n vector is of length $C\left(T_n - K + 1\right)$.
W_n	A weight matrix of size $T_n \times C\left(T_n - K + 1\right)$ created from elements of the array $\mathbf{v}$.
$\mathbf{v}$	An array of values used to construct dictionary elements. $v_{c,k}$ is associated with the k^{th} position in the c^{th} relevant subsequence pattern.
λ	The L_1 regularization parameter associated with each α_n.

Table 6.1: Relevant Subsequence Dictionary Learning parameters

6.1), $c(i)$ indicates the relevant subsequence position associated with the i^{th} column of W_n, and $\text{sgn}(i)$ indicates the sign of the correlation between the i^{th} matrix column, $(W_n)_i^\top$, and the current residual: $\text{sgn}\left((W_n)_i^\top \left(x_n - (W_n)_{\mathcal{A}}^\top \alpha_n\right)\right)$.

The matrix-matrix product, $X = (W_n)_{\mathcal{A}}^\top (W_n)_{\mathcal{A}}$, can be computed as follows:

$$X_{ij} = \begin{cases} \sum_{k=0}^{\max(K-t(j)+t(i),0)} \text{sgn}\left(i\right) v_{c(i),k} \text{sgn}\left(j\right) v_{c(j),t(j)-t(i)+k} & t(j) \geq t(i) \\ \sum_{k=0}^{\max(K-t(i)+t(j),0)} \text{sgn}\left(i\right) v_{c(i),t(i)-t(j)+k} \text{sgn}\left(j\right) v_{c(j),k} & t(j) < t(i) \end{cases} \tag{6.26}$$

This matrix-matrix product has an overall complexity of $O(|\mathcal{A}|^2 K)$. However, the full product does not need to be computed at each LARS iteration. Rather, as additional columns are added to the active set, we update a stored Cholesky decomposition of $(W_n)_{\mathcal{A}}^\top (W_n)_{\mathcal{A}}$, at a cost of $O(|\mathcal{A}|K)$ for each update (updates involve computing a single column of the product in Equation 6.26), plus $O(|\mathcal{A}|^2)$ for a back-substitution operation.

We compute the matrix-vector product, $\mathbf{x} = (W_n)_{\mathcal{A}} \omega_{\mathcal{A}}$, incrementally as the weighted sum of components of $\mathbf{v}$:

$$\mathbf{x}^{(1:i)} = \sum_{k=1}^{K} \mathbf{x}_{t(i)+k}^{(1:i-1)} + \operatorname{sgn}(i) (\omega_{\mathcal{A}})_i \, v_{c(i),k} \tag{6.27}$$

where $\mathbf{x}^{(1:i)}$ indicates the sum up to the i^{th} term, and $i \in [1 \ldots |\mathcal{A}|]$. This matrix-vector product has an overall complexity of $O(|\mathcal{A}|K)$.

Finally, we compute the matrix-vector product, $\mathbf{x} = W_n^{\top} u$, as follows:

$$\mathbf{x}_i = \sum_{k=1}^{K} v_{c(i),k} u_{t(i)+k} \tag{6.28}$$

with an overall complexity of $O(CT_nK)$.

For each LARS iteration, we must also compute CT_n correlations between each column of the matrix, W_n, and the current residual at a cost of K each. These are computed in the same way as the matrix-vector product in Equation 6.28. In most of out experiments, we restrict $|\mathcal{A}|$ to values less than or equal to C. Thus, each LARS iteration has a complexity of $O(CT_nK)$ when $|\mathcal{A}|$ is small, which can be a significant reduction from $O(CT_n^2)$. However, with no restrictions on the size of the active set, $|\mathcal{A}|$ can potentially grow to CT_n. In this case, complexity is eventually dominated by back-substitution operations involving the incrementally-updated Cholesky decomposition of $(W_n)_{\mathcal{A}}^{\top} (W_n)_{\mathcal{A}}$ at a cost of up to $O(C^2T_n^2)$ per iteration.

6.2.2 Modification and Related Work

The procedure for constructing the RS-DL dictionary (Figure 6.1) is applicable only to sequences with continuous-valued elements. To allow RS-DL to find decompositions of sequences of discrete symbols, we first transform each original sequence into M separate

binary sequences, where elements of the m^{th} binary sequence indicate if the symbols in the original sequence are equal to the m^{th} symbol in the alphabet. These M binary sequences are then concatenated to obtain the input sequence to RS-DL. Dictionary construction must also be modified for discrete sequences. In this case, $\mathbf{v}$ becomes a three-dimensional constituent array, where $v_{c,k,m}$ is associated with the m^{th} symbol of the k^{th} position in the c^{th} relevant subsequence. Separate dictionaries are constructed for each of the M possible symbols using $v_{c,:,m}$ for the c^{th} relevant subsequence pattern associated with the m^{th} constructed binary sequence. These M dictionaries are then stacked vertically to create a composite dictionary.

It is also possible to use RS-DL to find decompositions of multi-variate sequences. To do so, we rearrange each multivariate sequence as a concatenation of univariate sequences. We then create a stack of M dictionaries as we did to create the dictionary for discrete sequences.

Another modification of the basic RS-DL algorithm includes appending a column to the dictionary whose entries are set to a constant value. This addition has the effect of including a bias term whose magnitude varies depending on the associated α_n term. This bias term is useful for modeling time series datasets where the amplitudes of major trends that occur in individual sequences are offset by varying amounts. We employ this bias term in all experiments conducted on time series sequences. A similar strategy can also be employed to capture linear trends.

Finally, we can modify the LARS algorithm so that, rather than finding an L_1-regularized solution for α, it finds solutions with one or fewer non-zero α terms associated with each of the C relevant subsequence patterns. Although the L_1 regularization is no longer enforced in this case, sparsity is maintained in a similar manner to the L_0-regularized[2] version of LARS[88].

Other authors have proposed similar dictionary learning structures to RS-DL. Mailhe

[2]The "L_0 norm" [87] is a pseudo-norm that counts the number of non-zero components in a vector, i.e., $||\mathbf{x}||_0 = \sum_i \mathbb{I}(x_i \neq 0)$.

et. al. proposed Shift Invariant Dictionary Learning [91], which introduces a similarly-constructed dictionary to the one in RS-DL. In Shift Invariant Dictionary Learning single sequences are decomposed into a linear combination of dictionary elements. Training is conducted using Matching Pursuit combined with K-SVD to solve for the dictionary and associated weights.

6.3 Relationship to Hidden Markov Models

The Factorial SL-pHMM, described in Chapter 5, shares characteristics of RS-DL. The Factorial HMM [53] extends the basic HMM by postulating that the distribution over sequence elements depends on hidden states from multiple, parallel HMMs. If SL-pHMM factors are used, then the resulting Factorial SL-pHMM (Section 5.3.1), with Gaussian emission distributions, operates very similarly to RS-DL.

Figure 6.2 shows an example configuration of *Match* hidden states in a Factorial SL-pHMM. This hidden state configuration leads to the same additive composition of parameters used to represent symbols of an observed sequence in RS-DL. The primary differences between RS-DL and the Factorial SL-pHMM lie in how the parameters of each model are constrained. In the Factorial SL-pHMM, the model's hidden states can be encoded in a vector, $\alpha_n^{(FHMM)}$ of length $C(T_n - K + 1)$, where C indicates the number of factors in the model, T_n is the length of the sequence, and K is the number of *Match* states in the SL-pHMM. Because the hidden state sequence in the SL-pHMM only allows a single start position for each chain of *Match* states, encoding the positions of initial *Match* hidden states requires that $\alpha_n^{(FHMM)}$ be constrained as $\alpha_{n,i}^{(FHMM)} \in \{0, 1\}$ and $\sum_{t=0}^{T_n-K} \alpha_{n,c(T_n-K+1)+t}^{(FHMM)} = 1 \quad \forall c \in [1 \dots C]$. In contrast, the α_n-vectors in RS-DL are not explicitly constrained but are instead subject to L_1 regularization. Substituting an L_1 regularizer in RS-DL for the binary constraint in the Factorial SL-pHMM is advantageous, because it converts the combinatorial optimization problem associated with the MAP solution over hidden state configurations of the Factorial SL-pHMM into one that is more-easily solvable.

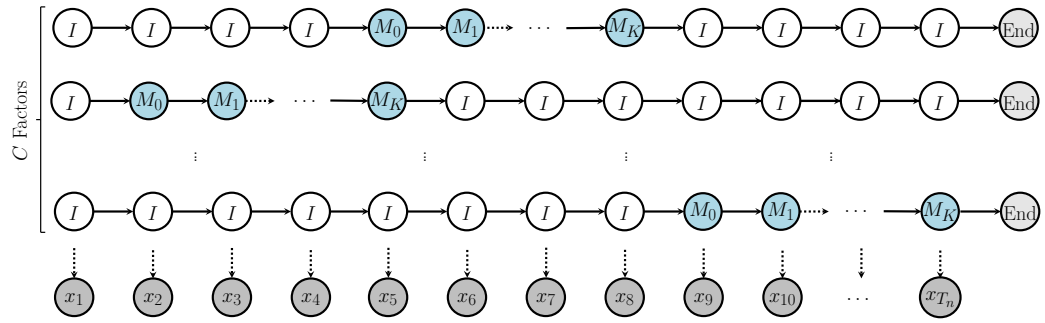


Figure 6.2: The diagram above illustrates example hidden state assignments for the Factorial SL-pHMM. Sequences of SL-pHMM *Match* states are indicated by blue nodes with the text "M_k," indicating the k^{th} *Match* state. *Insert* states are indicated by white-colored nodes with the text "I." SL-pHMM transition probabilities are defined so that only a single sequence of *Match* states per individual SL-pHMM can occur. For a Factorial SL-pHMM with Gaussian emission distributions, hidden states are associated with different weights which are summed over the C constituent SL-pHMMs (vertically in the diagram) to obtain the mean parameter used to generate the appropriate observed sequence element (in gray).

6.4 Experiments

We evaluate Relevant Subsequence Dictionary Learning using two types of measurements. First, we expect RS-DL to find meaningful subsequences within a dataset. This task is also referred to as "motif finding" [92, 93]. We quantitatively assess motif finding on a synthetic dataset consisting of discrete sequences where the ground truth motif positions are known. We also qualitatively assess motif finding results on sets of both time-series and text sequences to verify that RS-DL can pick out portions of a sequence meaningful to humans. In the next sections we make a distinction between the terms "relevant subsequence pattern" and "motif". We use "relevant subsequence pattern" to indicate the pattern encoded in RS-DL parameters, and "motif" to denote subsequences selected from a dataset because of their association with a particular relevant subsequence pattern.

We also test RS-DL in sequence classification. We hypothesize that if RS-DL can discover informative subsequences with no access to label information, then these subsequence features will be effective for classification. In these experiments, RS-DL features are input to a one-nearest-neighbor classifier to isolate the effect of different feature representations.

6.4.1 Datasets

We employ four types of datasets to evaluate our algorithm. To evaluate the ability of RS-DL to discover known motifs, we generated a synthetic dataset of discrete-valued sequences containing three predefined subsequences. We also assessed motif finding ability using a set of continuous-valued ECG sequences[3] and the Associated Press (AP) dataset[4], consisting of English language text. We assessed classification ability using only continuous-valued sequences. These included both a synthetic dataset, which we call the "Bumps" dataset, and datasets from the University of California Riverside (UCR) Time Series Classification Database [23].

6.4.2 Finding Motifs in Synthetic Sequences

To verify basic motif finding abilities of RS-DL, we constructed a synthetic dataset, allowing us to control the location and frequency of motifs. The synthetic dataset consisted of 20 sequences, generated to contain up to three non-overlapping motifs. These motifs consisted of 5 repetitions of "a," "r," or "n," with a 10% chance at each motif position for a motif character to be replaced by a character generated uniformly from the full sequence alphabet of 20 possible characters. Non-motif sequence elements were chosen uniformly at random from the full 20-characters alphabet. Sequence lengths were generated uniformly at random from a range of 25 to 75.

To explore the behavior of the RS-DL model, we ran a number of experiments, varying the values of K, the length of the relevant subsequence pattern, from 3 to 7, and the values of λ, the L_1-regularization parameter, from 0.4 to 0.8 in steps of 0.05. We configured the algorithm to use at most one of each relevant subsequence pattern to reconstruct each sequence.

Figure 6.3a shows graphs of the average precision and recall associated with motifs recovered by the RS-DL algorithm over 20 trials for each configuration of K and λ. We

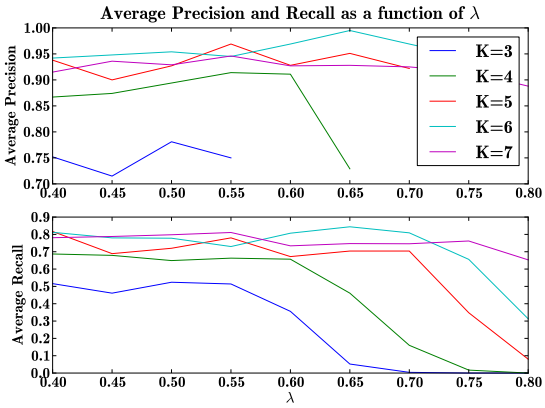
[3] http://www.cs.ucr.edu/~eamonn/discords/ECG_data.zip
[4] http://www.cs.princeton.edu/~blei/lda-c/ap.tgz

counted a ground truth motif as "discovered" if its start position was within $\lfloor K/2 \rfloor$ of the motif returned by the RS-DL algorithm. To verify the upper limit of algorithm performance and to confirm the trend that ground truth motifs were associated with larger values of α than false positive motifs, we counted motifs as "not found" if their associated α values were below 0.25.

Figure 6.3b, shows the output of the run of the RS-DL algorithm with the lowest mean-squared error (MSE) out of 20 random initializations. Columns in the figure display both values of α and motifs selected from the dataset sequence. Different dataset sequences are associated with different rows in the figure. In this run, low values of α are consistently associated with incorrectly discovered motifs (in red), and, out of four possible relevant subsequence patterns given by the model, only three are used, which is consistent with the ground truth.

Figure 6.3a shows that both precision and recall tend to increase as the value of K increases. In addition, the figure shows that if we set λ to a value that is too high, both precision and recall are degraded. This behavior, when varying λ, occurs because at high λ levels, the model becomes too sparse, reducing the number of motifs returned. In this case, we do not see a corresponding increase in precision because sparsity is only enforced in the number of relevant subsequences patterns used to reconstruct a sequence. Also from Figure 6.3a, the best precision scores were near 1.0, occurring with $K = 6$ and $\lambda = 0.65$ and filtering motifs with α coefficients less that 0.25. This result contrasts with top precision values of 0.5 (not shown in the figure) when the α filtering level is set to zero. The reason for this trend is illustrated in Figure 6.3b, where motifs associated with small α coefficients also tend to be less correlated with core relevant subsequence patterns.

To avoid low recall solutions it is possible to rerun the model for a number of trials with initial relevant subsequence patterns, $\mathbf{v}$, drawn from a standard Normal distribution. Because the RS-DL problem is non-convex, the optimization algorithm will converge to different areas in the parameter space depending on initial parameter settings. We found that, with our synthetic dataset, low-MSE runs consistently produced recall values of 1.0



(a)

α	Motif 1	α	Motif 2	α	Motif 3	α	Motif 4
0.41	aaaaa	0.00	—	0.44	rrrrv	0.44	nnnnn
0.32	asaaa	0.00	—	0.42	rrryr	0.18	yrnnl
0.33	laaaa	0.00	—	0.00	—	0.36	nnnen
0.41	aaaaa	0.00	—	0.53	rrrrr	0.00	—
0.41	aaaaa	0.00	—	0.53	rrrrr	0.29	nrnkn
0.17	safra	0.00	—	0.10	anrqv	0.44	nnnnn
0.26	eaapa	0.00	—	0.21	rgqrt	0.08	qrtnd
0.41	aaaaa	0.00	—	0.53	rrrrr	0.44	nnnnn
0.41	aaaaa	0.00	—	0.53	rrrrr	0.44	nnnnn
0.41	aaaaa	0.00	—	0.00	—	0.07	lenma
0.41	aaaaa	0.00	—	0.20	rsgpr	0.09	aaten
0.08	vsnca	0.00	—	0.53	rrrrr	0.44	nnnnn
0.41	aaaaa	0.00	—	0.43	rnrrr	0.44	nnnnn
0.41	aaaaa	0.00	—	0.53	rrrrr	0.44	nnnnn
0.17	vafsa	0.00	—	0.32	rrvry	0.08	famln
0.33	aaæa	0.00	—	0.00	—	0.18	nwtwn
0.41	aaaaa	0.00	—	0.53	rrrrr	0.37	nrnnn
0.41	aaaaa	0.00	—	0.43	rrrrm	0.44	nnnnn
0.08	lytia	0.00	—	0.43	rtrrr	0.44	nnnnn
0.08	nigpa	0.00	—	0.08	lkrei	0.36	fnnnn

(b)

Figure 6.3: **(a)** The graphs in the left-hand figure depict precision and recall over 20 runs of the RS-DL model on a synthetic dataset. As the L_1-regularization term, λ, increases, fewer motifs are returned, leading to a drop in both recall and precision. As the length of the relevant subsequence patterns increase, precision and recall tend to increase. **(b)** Recovered α coefficients (left side of Figure b) and associated subsequences (right) from a low-error run (the run with the smallest MSE out of 20 random initializations) of the RS-DL algorithm on the synthetic dataset. The low-error run gave a precision of 0.7 (with an α cutoff of 0) and a recall of 1.0. The number of relevant subsequences patterns, C, in the model was set to four, while the number of ground truth motifs was three. Consistent with the ground truth, the model only used three relevant subsequences patterns to reconstruct the data. Incorrectly discovered motifs are depicted in red.

(see Figure 6.3b). Selecting a low-MSE run also allows us to better take advantage of RS-DL's sparsity. For instance, if we set C, the number of relevant subsequence patterns, to 4, larger than the 3 ground truth motifs in our dataset, then low-MSE solutions return only 3 discovered motifs (higher error solutions find a fourth motif with noisy parameters).

6.4.3 Motifs in Time-Series Data

To show that RS-DL can pick out patterns meaningful to humans in continuous-valued sequences, we trained it on a single sequence of electrocardiogram (ECG) data, containing 3750 datapoints. The ECG sequence consists of a recording of electrical signals from the human heart measured at the surface of the skin. A plot of the signal (Figure 6.4) contains repeated patterns easily identifiable to humans. The ECG sequence also contains an anomalous motif, which, like the main set of patterns in the sequence, is easily identified by humans. We ran

the RS-DL algorithm on the sequence with the L_1 regularization term, λ, set to .1 and the length of the relevant subsequence pattern, K, set to 150, and C, the number of relevant subsequence patterns, set to 15. In Figure 6.4, we plotted the relevant subsequence patterns learned by RS-DL associated with the largest 50 regression coefficients, α. Each pattern in the plot (top three graphs) consists of 150 values of the relevant subsequence pattern given by the constituent vector, $\mathbf{v}$, multiplied by its corresponding α coefficient. Summing over all of these plotted subsequences gives the approximate sequence reconstructed by RS-DL (bottom plot in green, offset by -1). The original sequence is also shown in the figure (bottom plot in blue). The MSE between reconstructed sequence and the original sequence was 0.98. As expected, the figure demonstrates how the relevant subsequence patterns in the upper graphs are strongly correlated with the human-perceptible patterns from the original sequence in the bottom graph. Another interesting property of the RS-DL decomposition shown in Figure 6.4 relates to the sparsity of the model. Only a three (6, 10, and 11) out of fifteen possible relevant subsequence patterns account for the main patterns in the sequence while additional patterns are responsible for increasingly fine-grained approximations. This type of behavior is similar to commonly-used orthonormal bases, such as the DCT basis, which consist of low frequency components that capture major trends, while high-frequency basis elements capture finer-grained variations. Another characteristic of the RS-DL solution is that the anomalous portion of the sequence is associated with a different relevant subsequence pattern (Relevant Subsequence Pattern 10) than the common ECG pattern. This characteristic shows how RS-DL can be used not only to find positions of recurrent patterns but also to distinguish between pattern types.

6.4.4 Motifs in Text Data

As an additional test of RS-DL's motif-finding ability, we trained the model on the Associated Press (AP) corpus. We preprocessed the corpus by removing words that occurred more than 500 times or in fewer than three documents. We then removed documents containing fewer than 10 words. The processed corpus size was 2213 documents. Finally, to make

processing the text dataset tractable, rather than representing each word as a large binary vector (which would typically have a length of at least 10,000), we used the "word embedding" representation from Collobert et. al. [94]. These word embeddings are vectors in $\mathbb{R}^{50}$ and were constructed with the intent that Euclidean distances between pairs of vectors should be small if the meanings of their associated words are similar.

Figure 6.5 shows the top 15 examples, as ordered by the absolute value of the associated α coefficient, of the top four relevant subsequence patterns (out of $C = 10$ total possible relevant subsequence patterns) learned from a run of the RS-DL algorithm. Unlike text processing methods that treat words independently, RS-DL preserves the order of words within each document (minus words removed in the document preprocessing step). As the columns of five-word groups in the figure show, RS-DL, in minimizing reconstruction error over sequences of word embeddings, is capable of finding and grouping together meaningful sequences of words within the text. In the figure, all columns of discovered motifs share internally consistent semantic themes. Moreover, these themes tend to center around phrases containing important nouns. For instance, Motif 1 includes organization-related phrases like "product safety commission defended", "public health system plagued", and "environmental protection agency banned". Motifs 2 and 4 contain phrases including a person and occupation description such as "defense attorney thomas e. wilson", "district attorney william h. ryan", and "secretary james a. baker" in Motif 2 and "attorney michael rosen," "education secretary william bennett," and "assistant district attorney ted stein." Motif 3 is centered on organizations and concepts like "natural resources," "public services," and "tough economic conditions."

6.4.5 Classification Experiments

To assess whether features derived by RS-DL are useful for classification, we compared the performance of these features on both a synthetic dataset of our own design and five UCR Time Series datasets that satisfied the underlying assumptions of our model. Because RS-DL selects subsequences, we do not expect features from the algorithm to be effective

for classification when discriminative information between sequence categories lies in global trends over an entire sequence or if the order of different patterns within a sequence is highly correlated with its category. Similarly, because RS-DL is a sparse regression algorithm, we expect relevant subsequence patterns to be matched to high-magnitude areas of dataset sequences. Therefore, if dataset sequences contain large-magnitude areas (e.g. spikes in an ECG sequence), but discriminative information found elsewhere in the sequence, we do not expect RS-DL features to be effective for classification.

We these assumptions in mind, we generated a set of continuous sequences, which we call the "Bumps" dataset[5] (see Figure 6.6c). Each sequence in this dataset contains two large magnitude bumps placed at random and without overlap. In the negative category one out of the two bumps in each sequence contains a divot. We also selected five datasets from the UCR Time Series database that conform to the underlying assumptions about RS-DL: CBF, Coffee, DiatomSizeReduction, ECGFiveDays, and TwoLeadECG. These datasets consist of sequences that contain large magnitude patterns occurring in all or nearly all sequences, satisfying the assumptions needed for RS-DL to extract useful features.

We ran RS-DL with randomly initialized $\mathbf{v}$ arrays for ten trials on all sequences in both the training and test sets, excluding label information, for each dataset. For all experiments, we set $C = 10$, K to 30% of the sequence length, and $\lambda = 3.0$. We also enabled the restriction on the LARS algorithm (see Section 6.2.2) where only a single relevant subsequence pattern of each type was used. For each sequence, we created feature vectors by concatenating the subsequences associated with each relevant subsequence pattern. Table 6.2 shows a comparison of classification errors using the one-nearest-neighbor algorithm on (i) features given by treating sequences as vectors in Euclidean space, (ii) Dynamic Time Warping (DTW)[6] [26] distances between sequences, and (iii) Euclidean distance between RS-DL feature vectors. As assessed by McNemar's test [95], RS-DL features reduce classification error over raw sequence vectors with p-values of less than 0.014 for all datasets. For all datasets except for the CBF dataset, RS-DL features improved on the classification error

[5]This dataset is included with the RS-DL code at http://cs.gmu.edu/∼sblasiak/RS-DL.tar.gz

[6]DTW scores were computed using the R DTW package at http://dtw.r-forge.r-project.org/

Dataset	# Categories	# Train	# Test	Sequence	DTW	RS-DL
Bumps (synthetic)	2	50	50	0.460	0.140	**0.056**
CBF	3	30	900	0.148	**0.030**	0.108
Coffee	2	28	28	0.250	0.214	**0.171**
DiatomSizeReduction	4	16	306	0.065	0.042	**0.028**
ECGFiveDays	2	23	861	0.203	0.249	**0.095**
TwoLeadECG	2	23	1139	0.253	0.073	**0.035**

Table 6.2: Classification results using RS-DL features on the UCR Time Series datasets. The "Sequence", "DTW", and "RS-DL" columns give error rates from the one-nearest-neighbor algorithm using the Euclidean distance between sequences, Dynamic Time Warping scores, and Euclidean distance between RS-DL features respectively. RS-DL features improved the classification error for four out of five UCR datasets.

over Dynamic Time Warping. Here, all results were significant with p-values of less than 0.01, except for the Coffee dataset, where RS-DL's improvement over DTW was significant with a p-value of 0.17.

Figure 6.6 shows examples of positive and negative category sequences from three of the classification datasets. In each case, RS-DL features lead to improved time-series category prediction by isolating large-magnitude trends in subsequences shared across the set of sequences (as shown in the upper portions of each plot in the figure). Constructing features from these isolated subsequences aligns these major subsequence trends, allowing minor variations that occur between the positive and negative sequence categories to be more-easily distinguished. When variations in the general trend are highly correlated with a category label, then the feature isolation provided by RS-DL can lead to more accurate classification.

6.5 Conclusions

In this chapter, I have presented Relevant Subsequence Dictionary Learning, a novel method for adapting Sparse Dictionary Learning to discover interesting subsequence patterns across sets of sequences. RS-DL is related to standard statistical models over sequences through a version of the Factorial HMM with specially formulated restrictions on transition probabilities. In a series of experiments, we have shown that RS-DL can discover useful information

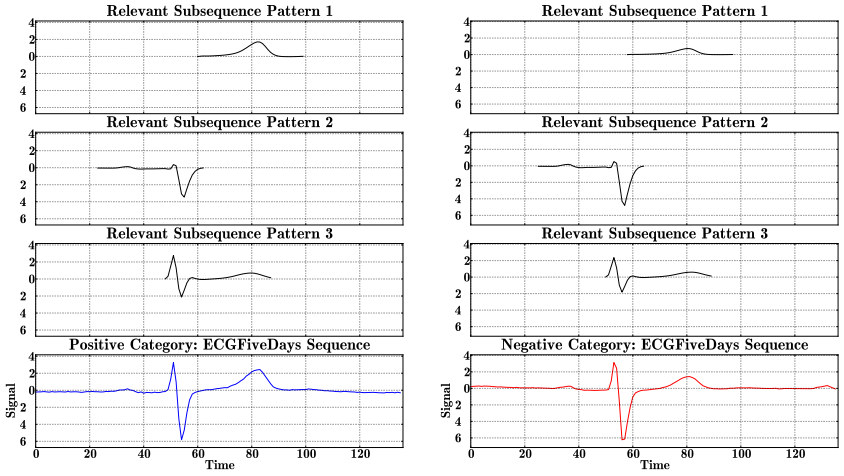
across a variety of sequence domains. In addition, as demonstrated on time-series data, sequence features extracted using RS-DL can improve sequence classification performance.

Figure 6.4: A plot of the relevant subsequences patterns (upper plots) associated with the largest 50 coefficients of the vector α that were learned by RS-DL to approximate the ECG sequence (bottom plot, blue line). Only 3 out of the 15 possible relevant subsequence patterns appear in this set of 50. Relevant subsequence patterns learned by RS-DL are strongly associated with human-identifiable patterns in the sequence. The figure also shows that the approximation learned by RS-DL (bottom plot, green line) is very similar to the original sequence with an MSE of 0.98. The RS-DL approximation is offset by -1 on the y-axis to aid in presentation.

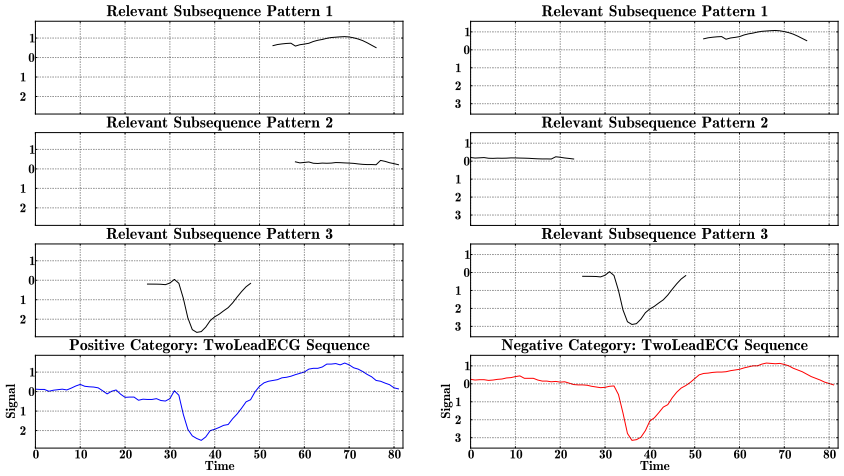
Motif 1, max $\|\alpha\| = 5.357$					Motif 2, max $\|\alpha\| = 1.659$				
product	safety	commission	defended	record	defense	attorney	thomas	e.	wilson
community	service	act	provides	community	district	attorney	william	h.	ryan
standards	computer	industry	announced	technology	coalition	secretary	james	a.	baker
intelligence	support	ship	passed	black	assistant	secretary	john	h.	kelly
flight	training	manuals	brought	date	treasury	secretary	james	a.	baker
public	service	employees	received	raise	treasury	secretary	james	a.	baker
public	health	system	plagued	months	district	judge	william	t.	hart
foreign	debt	economy	warned	loss	navy	secretary	james	h.	webb
center	health	promotion	issued	warning	washington	secretary	james	a.	baker
korea	security	force	placed	armed	letter	secretary	james	a.	baker
u.n.	security	council	passed	use	washington	secretary	james	a.	baker
emergency	fund	funds	provided	embassies	date	secretary	james	a.	baker
environmental	protection	agency	banned	chemical	announced	secretary	james	a.	baker
forces	law	order	opened	fire	campaign	chairman	james	a.	baker
justice	information	organization	conducted	survey	assistant	attorney	stephen	a.	mansfield

Motif 3, max $\|\alpha\| = 1.635$					Motif 4, max $\|\alpha\| = 1.628$				
workers	discrimination	foreign	workers	impact	education	university	attorney	michael	rosen
authority	conduct	foreign	policy	direct	debt	education	secretary	william	bennett
detainees	paying	legal	fees	work	hearing	assistant	attorney	john	carroll
assets	include	private	loans	trade	assistant	district	attorney	ted	stein
workers	give	local	unions	grievances	assistant	immigration	commissioner	james	kennedy
program	financed	foreign	aid	assets	white	assistant	attorney	richard	roberts
bills	raise	environmental	protection	agency	questioning	assistant	attorney	john	carroll
planning	aid	foreign	organizations	perform	controversy	press	secretary	david	beckwith
penalties	sought	corporate	executives	release	talks	treasury	secretary	james	brady
problem	finding	foreign	aid	money	deputy	assistant	attorney	john	martin
policies	managing	public	affairs	front	phone	treasury	secretary	james	baker
law	banned	agricultural	products	coal	hearing	district	judge	edward	davis
proposal	noted	natural	resources	issue	list	assistant	attorney	william	reynolds
money	available	public	services	whose	ruling	district	judge	edward	davis
reasons	tough	economic	conditions	particular	acting	prime	minister	ben	jones

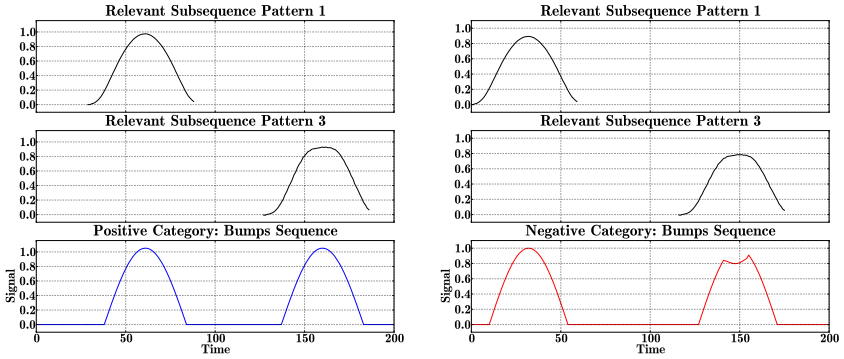
Figure 6.5: The figure above shows motifs discovered by the RS-DL model in the Associated Press corpus. It lists the top 15 motifs by α coefficient of the top four (out of ten possible) relevant subsequence patterns. Motifs found by RS-DL have, in general, captured sets of semantically coherent phrases. Motifs 1 and 2 contain phrases including organization and noun/concept phrases while Motifs 2 and 4 contain phrases including a person and occupation descriptions.



(a)



(b)



(c)

Figure 6.6: Figures **a**, **b**, and **c** above show the top two (by α value) relevant subsequence patterns that approximate positive (bottom blue) and negative category (bottom red) sequences in the ECGFiveDays, TwoLeadECG, and our synthetically-generated Bumps datasets respectively. For each of these datasets, RS-DL features improve classification performance by picking out similarly shaped subsequences from different dataset categories. Classification performance improves because class distinctions occur in minor variations in the major trends captured by RS-DL. After processing by RS-DL, these minor variations can more easily be distinguished by standard classification algorithms.

Chapter 7: A Family of Feed-forward Models for Protein Sequence Classification

Popular and successful approaches for protein sequence classification employ Support Vector Machines (SVM) [11, 12, 14, 16, 96, 97]. Performance of SVM-based classifiers is highly dependent on the kernel function, which can be difficult to specify and to interpret. Kernel functions often have free parameters that must be set either through cross validation or heuristics. Further, ad hoc techniques are often employed to normalize pre-computed kernels so that the algorithm can learn larger margins between classes.

We present a sequence classification framework that differs from the SVM/kernel-based approach. We construct a type of neural network called a Subsequence Network (SN) that incorporates structural models over subsequences. These structural models, called Sequence Scoring Models (SSMs), are similar to Hidden Markov Models and act as a mechanism to extract relevant features from sequences. Our feed-forward structure allows standard optimization techniques to be used for learning linear discrimination weights in conjunction with sequence-level features. We compare our algorithm against state of the art kernel methods on a set of canonical datasets for structural and functional protein sequence classification.

7.1 Background

The methods we present in this chapter combine ideas from Hidden Markov Models (HMMs) (Section 2.1) and Artificial Neural Networks. Hidden Markov models are probabilistic models over sequences. They postulate a generative process where a sequence of hidden states is generated from a Markov process. Each hidden state is then used as a component of mixture distribution to generate observed sequence symbols. The type of HMM most commonly used to model protein sequence is the Profile HMM (Section 2.1.1). The models in this chapter

rely on structures similar to the Profile HMM to map sequences to a latent space, which is then used for classification in a neural network-like architecture.

7.1.1 Neural Networks

A feed-forward artificial neural network (ANN) is a nonlinear classifier or regression function where the input to output transformation is a composition of differentiable functions: $f^{(H)}\left(\ldots\left(f^{(1)}\left(x\right)\right)\ldots\right)$. We assume a dataset $\{(x_n, y_n)\}_{n=1}^{N}$, $x \in \mathcal{X}$, $y \in \mathcal{Y}$ where x_n is an input vector associated with an output y_n. We denote the vector valued output of each layer as $\mathbf{f}^{(h)}$, which we call "layer h" of the neural network. The top layer of the network (layer H) is compared against the true output, y, using a loss function $\ell\left(\mathbf{f}^{(H)}, y\right)$. In a standard ANN, $\mathbf{f}_i^{(h)}$, the i^{th} element of the vector $\mathbf{f}^{(h)}$, is computed by passing a linear combination of the values from the previous layer through a squashing function (usually the hyperbolic tangent function). Each $\mathbf{f}_i^{(h)}$ is computed independently of $\mathbf{f}_j^{(h)}$, $i \neq j$ given values from layer $h-1$:

$$\mathbf{f}^{(h)} = \left[f_1^{(h)}\left(\mathbf{f}^{(h-1)}\right), \ldots, f_{n_h}^{(h)}\left(\mathbf{f}^{(h-1)}\right)\right]^{\top}, \tag{7.1}$$

where n_h is the number of elements in layer h and $f_i^{(h)}$ denotes the composition of squashing and linear functions used to compute the i^{th} element of $\mathbf{f}^{(h)}$.

Convolutional Neural Networks (CNNs) [98] are inspired by neural connections in the human visual cortex. In CNNs, lower levels of the network respond to local portions of the input. For instance, the Time Delay Neural Network (TDNN) [99] is a type of CNN used in speech recognition. In the TDNN, the first hidden layer of the network is computed from a set of overlapping windows of an input sequence i.e., $\mathbf{f}^{(1)}$ is computed from an input

sequence $x_{1:T}$:

$$\mathbf{f}^{(1)} = \Big[f_1^{(1)}\left(x_{1:n_{cnv}}\right), f_1^{(1)}\left(x_{2:n_{cnv}+1}\right), \ldots, f_1^{(1)}\left(x_{T-n_{cnv}:T}\right), \ldots$$
$$, f_{n_h}^{(1)}\left(x_{1:n_{cnv}}\right), f_{n_h}^{(1)}\left(x_{2:n_{cnv}+1}\right), \ldots, f_{n_h}^{(1)}\left(x_{T-n_{cnv}:T}\right) \Big]^\top \quad (7.2)$$

where n_{h-1} indicates the number of elements in the vector $\mathbf{f}^{(h-1)}$ and n_{cnv} is the size of the "convolutional window" (the number of elements from the input, x, which contribute to produce the value of layer 1). We use Matlab slice notation, $x_{i_1:i_2}$, to indicate a sub-vector of the input sequence starting at index i_1 and ending at index i_2.

7.2 Sequence Classification with Subsequence Networks

Our family of feed-forward classification models are convolutional neural networks that assume a protein's structural or functional category can be predicted by the presence of a set of subsequences. We call these models Subsequence Networks (SN). In a Subsequence Network, the convolutional layer learns the degree to which a subsequence is present in a protein sequence. The degree of presence of a subsequence acts as a feature, which can be input to a linear classification layer, allowing combinations of these subsequence features to be detected.

Convolutional units in the Subsequence Network are structured like HMMs (Section 2.1), except that we relax the constraint that the output of each unit defines a probability distribution over sequences. That is, we perform unconstrained optimization with respect to the logarithm of the transition and emission probabilities, i.e., $w_{k,k'}^{trans} \stackrel{def}{=} \log \theta_{k,k'}^{trans}$ and $w_{k,m}^{emit} \stackrel{def}{=} \log \theta_{k,m}^{emit}$, rather than enforcing the constraint that each θ vector sums to one. We refer to these unnormalized models as "Sequence Scoring Models" (SSM). Figure 7.1 shows a diagram of our Subsequence Network using an SSM convolutional layer.

7.2.1 Pair-SSMs

The Pair-SSM is an unnormalized version of the Pair HMM [100]. Pair HMMs probabilistically extend the concept of edit distance by assigning probabilities to a symmetric set of insertions, deletions, and substitutions that allows one sequence from a pair to be created from the other.

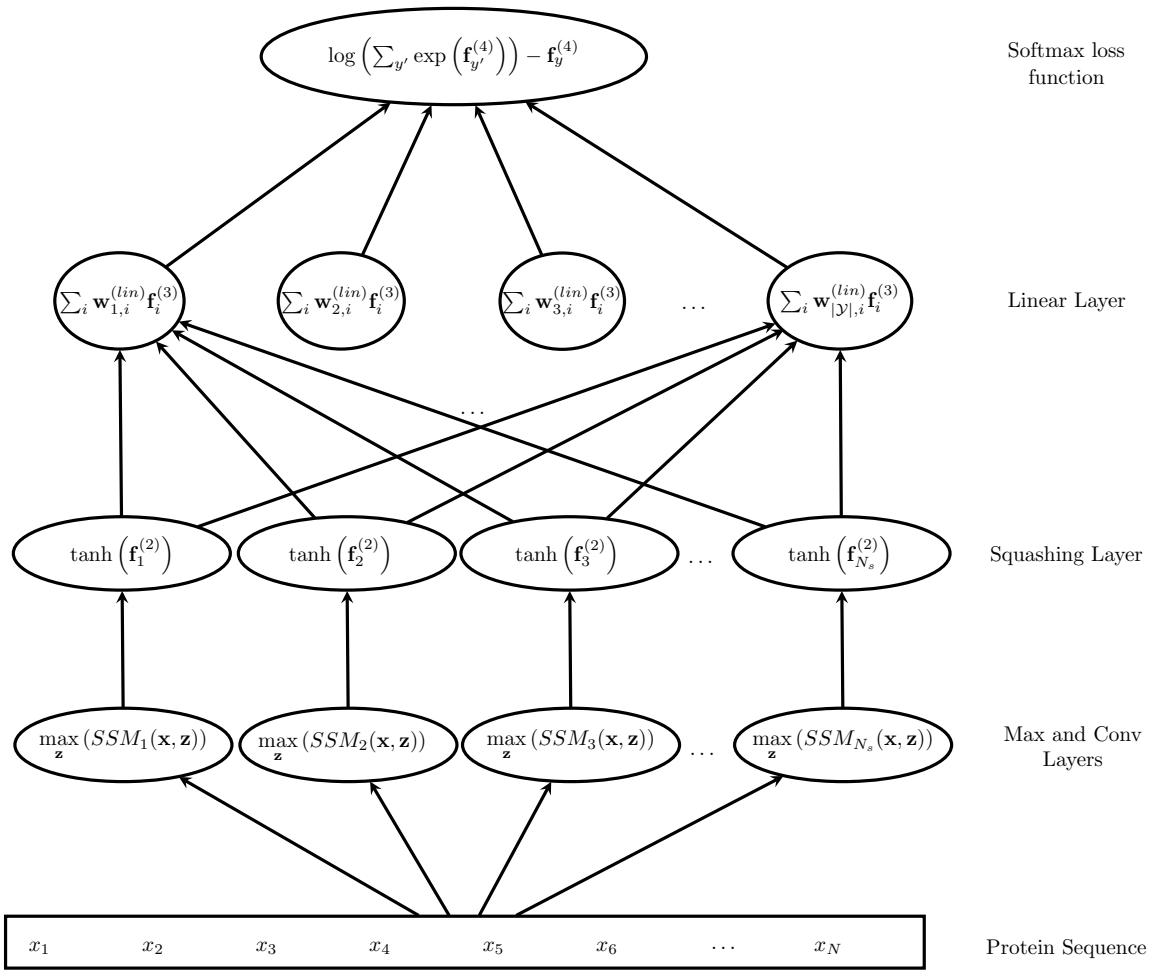


Figure 7.1: A diagram illustrating a Subsequence Network being applied to an input sequence. In the bottom row of the network, the maximum of the scores from each SSM are taken over the input sequence. The Conv layer is defined by a score from an SSM. In the second row, a squashing function is applied to the maximum SSM scores. The third row computes the distance of these squashed scores from hyperplanes used to define boundaries between sequence categories. Finally, the loss function compares the category given by the hyperplane to the true sequence category.

The log probability of a pair of sequences, $\mathbf{x}_i$ and $\mathbf{x}_j$, in the Pair HMM can be given by a linear model in the log of the distribution parameters:

$$\log p(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}) = \sum_{k,k'} n_{k,k'}^{trans} w_{k,k'}^{trans} + \sum_{m,m'} n_{m,m'}^{emit} w_{m,m'}^{emit} \tag{7.3}$$

where $n_{m,m'}^{emit}$ indicates the number of times we substituted an amino acid, m, from sequence $\mathbf{x}_i$ with m' from sequence $\mathbf{x}_j$, and $w_{m,m'}^{emit}$ is the associated cost of this substitution. The expression in Equation 7.3 differs from the probability of the standard HMM (Equation 2.3) in that we replace counts of emissions from a hidden state by counts of substitutions of amino acid m from sequence i with amino acid m' from sequence j i.e., $n_{k,m}^{emit}$ becomes $n_{m,m'}^{emit}$ and $w_{k,m}^{emit}$ becomes $w_{m,m'}^{emit}$. The Pair-SSM includes three types of hidden states: In an *Insert* hidden state, the model emits a symbol from sequence $\mathbf{x}_i$. In a *Delete* hidden state the model emits a symbol from sequence $\mathbf{x}_j$. In a *Match* hidden state, the model emits a symbol from both sequences. To allow the model to capture relevant subsequences, we add additional hidden states I_{start} and I_{end}. These hidden states emit symbols of either $\mathbf{x}_i$ or $\mathbf{x}_j$ from a background distribution, allowing the main portion of the Pair-SSM to emit symbols from the relevant subsequence. We show a diagram of hidden state transitions in Figure 7.2a.

If used in the convolutional layer of a Subsequence Network, the Pair-SSM extracts features using a similar technique as the LA-kernel. In this type of network, subsequences that best match an input sequence are selected from within each training set sequence by the Pair-SSMs. This view is closely related to the empirical kernel map [14, 101]. In the empirical kernel map, a feature vector associated with an unknown sequence is given by a vector of kernel evaluations over the training set i.e., we map the sequence x to the vector $[K(x_1, x), \ldots, K(x_N, x)]^\top$, where $K(\cdot, \cdot)$ is the kernel function and $\{x_n\}$, $n \in [1 \ldots N]$ is the set of training sequences. In the first layer of empirical kernel map, a feature vector is computed from an input sequence by evaluating a fixed kernel function on the input sequence

paired with each training set sequence. This set of values is then combined linearly using weights, w_n, to produce an overall score for the query sequence: $s(x) = \sum_n w_n K(x_n, x)$. We classify the query sequence, x, as a member of the positive class if $s(x) > 0$ and as a member of the negative class otherwise. In SVM/kernel classification, kernel evaluations for all pairs of sequences are computed independently. Then, given $K(x_i, x_j)$, $\forall i, j$, the SVM learning algorithm solves a (convex) quadratic program to compute the linear weights, w_n.

Although optimization over the Subsequence Network with a Pair-SSM convolutional layer is tractable, it is not yet practical without distributing computation over multiple processors. For each SGD epoch we must compute the Viterbi paths over N^2 pairs of sequences, x_i and x_j, at a cost of $O(|x_i| \times |x_j|)$ (where N is the number of training sequences and $|x|$ is the length of a sequence).

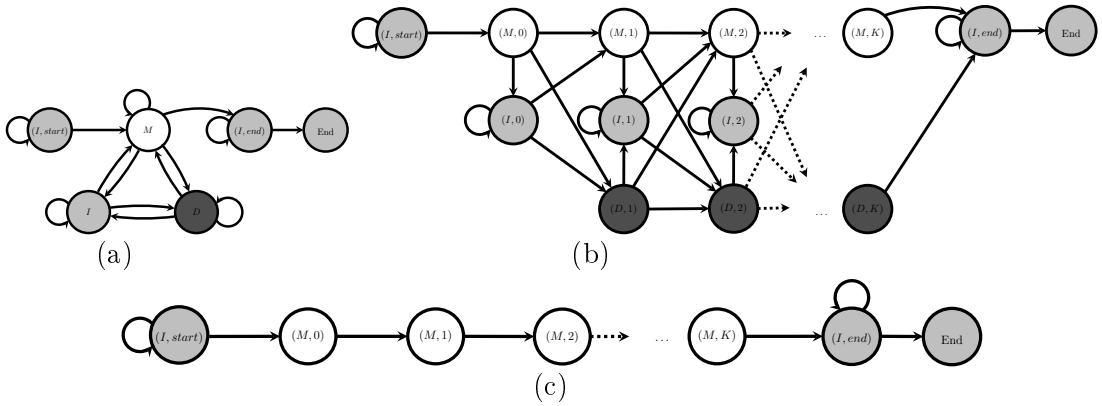


Figure 7.2: A diagram of the deterministic finite-state automaton associated with **(a)** the Pair-SSM **(b)** the Local SSM (L-SSM) and **(c)** the Simplified Local SSM (SL-SSM). Match states are indicated with a white background, Insert states with a light-gray background, and Delete states with a dark-gray background.

7.2.2 Local SSM (L-SSM)

The Local SSM (Figure 7.2b) is an unnormalized version of the Profile HMM (pHMM) adapted to model a single subsequence within an observed sequence. Profile HMMs [9]

are a variation of the standard HMM commonly used for modeling biological sequences. They are left-to-right, non-ergodic HMMs [2] that represent sequences in relation to an archetypal sequence encoded in the emission distributions of the pHMM's hidden states. Profile HMMs use three types of hidden states: (1) *Match (M)* states encode individual symbols of the archetypal sequence, (2) *Insert (I)* states allow additional symbols to be inserted between matched symbols, and (3) *Delete (D)* states allow matched symbols to be skipped. Hidden states of the archetypal sequence are expressed as pairs of symbols (s, k), where $s \in \{M, I, D\}$ indicates a Match (M), Insert (I), or Delete (D), paired with a base state, $k \in [1 \dots K]$, which can be thought of as indexing a symbol in the archetypal sequence. The form of the archetypal sequence is encoded by the emission distributions from each of the K match states. In the local version of the pHMM, which models a single subsequence within the observed sequence, I_{start} and I_{end} are special insert states that allow portions of the sequence before the archetypal sequence to be skipped. We fix transition and emission probabilities from I_{start} and I_{end}, allowing these to be ignored during optimization. In addition, we explicitly include an *End* state to mark the end of the observed sequence. We must include the *End* state because without transition from I_{end} to *End*, the model favors archetypal subsequences positioned near the beginning of the observed sequence. As in the standard pHMM, in our L-SSM, emissions occur only from Match and Insert states.

7.2.3 Simplified Local SSM (SL-SSM)

Like the L-SSM, the SL-SSM models relevant subsequences within a set of sequences. The SL-SSM (Figure 7.2c) simplifies the L-SSM by removing Insert and Delete states from the model. This change in the model results in contiguous subsequences of match states. This structural change speeds inference i.e., the highest scoring set of hidden states can be efficiently computed by sliding the window of K match states over a sequence and returning the position with the highest probability. We use the same parametrization of hidden states as in the L-SSM: $\{I_{start}$,$(M, 1)$,$\dots$,(M, K),I_{end},END$\}$, where the pair (M, k), indicates a hidden state that emits the k^{th} symbol of the archetypal sequence. Transitions from the

Match state (M, k), $k < K$ to $(M, k+1)$ and transitions from the state (M, K) to the state I_{end} occur with probability one.

The L-SSM or SL-SSM convolutional layer in the Subsequence Network can be interpreted as a simplification of the SN with a Pair-SSM convolutional layer. This simplification is motivated by two assumptions about the domain of protein sequences: (i) the set of protein sequences lies on a lower-dimensional manifold within the sequence space and (ii) the basis given by the training set spans the manifold of our domain and is redundant. With these assumptions it becomes reasonable to simplify the Pair-SSM convolutional layer by creating a model with a lower-dimensionality basis independent of the training examples. For our model, we choose this basis to be a fixed set of L-SSMs or SL-SSMs.

When we replace Pair-SSMs with (S)L-SSMs, additional computational efficiencies become possible because the (S)L-SSM allows us to store only the locally relevant pattern rather than an entire sequence. Computing the score of a sequence under an (S)L-SSMs where hidden states are restricted to small, fixed lengths therefore requires less computation time than evaluating the Pair-SSM between pairs of sequences. A disadvantage of these simplifications is that new parameters are added to the model. In particular, the number of (S)L-SSMs and the number of hidden states for each (S)L-SSM must be specified in advance. In the results section, we show that these simplifications not only maintain much of the accuracy of the Pair-SSM layer, but they are also robust to variations in parameter choices.

7.2.4 Subsequence Network Objective Function

Parameter	Definition
N_s	number of SSMs in the convolutional layer
$\mathbf{f}^{(h)}$	the vector of values that comprise hidden layer h
$\mathbf{w}^{all}$	a combined vector of all SSM and linear weights, $\left[\left(\mathbf{w}^{(lin)}\right)^\top, \left(\mathbf{w}^{(1)}\right)^\top, \ldots, \left(\mathbf{w}^{(N_s)}\right)^\top\right]^\top$
$\mathbf{w}^{(lin)}$	linear weights that define linear boundaries between dataset categories, $\mathbf{w}^{(lin)} = \left[w_{1,1}^{(lin)}, \ldots, w_{1,N_s}^{(lin)}, \ldots, w_{\lvert\mathcal{Y}\rvert,1}^{(lin)}, \ldots, w_{\lvert\mathcal{Y}\rvert,N_s}^{(lin)}\right]$
$\mathbf{w}^{(i)}$	a vector of transition and emission weights for the i^{th} SSM, $\mathbf{w}^{(i)} = \left[\left(w_{1,:}^{trans}\right)^{(i)}, \ldots, \left(w_{K,:}^{trans}\right)^{(i)}, \left(w_{1,:}^{emit}\right)^{(i)}, \ldots, \left(w_{K,:}^{emit}\right)^{(i)}\right]^\top$ where K is the number of hidden states in the SSM
$\mathcal{Z}^{(i)}$	the set of hidden states for the i^{th} SSM

Table 7.1: Subsequence Network parameters

The objective function for our model includes a loss for each sequence in the dataset and a regularization term that penalizes large parameter values:

$$F(\mathbf{x}) \quad = \quad \sum_n \ell\left(\mathbf{x}_n, y_n; \mathbf{w}^{all}\right) + \frac{\lambda}{2}||\mathbf{w}^{all}||^2 \tag{7.4}$$

where $\mathbf{w}^{all}$ is an agglomeration of all of the SSM weight vectors in the model (the composition of $\mathbf{w}^{all}$ varies depending on which type of SSM is used for the convolutional layer) and linear combination weights associated with each SSM; $\ell(x, y; \mathbf{w}^{all})$ is a loss function. The λ term determines the trade off between the loss and magnitude of the weights.

A loss function compares the output of a CNN with the label of a single protein sequence. In our model, we use the a softmax loss, shown in the first row of Table 7.2. The output of the network is given by

Layer	Definition	Jacobian with respect to $\mathbf{f}^{(h)}$
$\ell\left(\mathbf{f}^{(4)}, y\right)$	$\log \sum_i \exp\left(\mathbf{f}_i^{(4)}\right) - \mathbf{f}_y^{(4)}$	$\begin{bmatrix} \frac{\exp\left(\mathbf{f}_1^{(4)}\right)}{\sum_i \exp\left(\mathbf{f}_i^{(4)}\right)}, \ldots, \frac{\exp\left(\mathbf{f}_{\lvert\mathcal{Y}\rvert}^{(4)}\right)}{\sum_i \exp\left(\mathbf{f}_i^{(4)}\right)} \\ \vdots \\ \frac{\exp\left(\mathbf{f}_1^{(4)}\right)}{\sum_i \exp\left(\mathbf{f}_i^{(4)}\right)}, \ldots, \frac{\exp\left(\mathbf{f}_{\lvert\mathcal{Y}\rvert}^{(4)}\right)}{\sum_i \exp\left(\mathbf{f}_i^{(4)}\right)} \end{bmatrix} - I$
$\mathbf{f}^{(4)} \equiv \text{Lin}\left(\mathbf{f}^{(3)}\right)$	$\left[\sum_{i=1}^{N_s} w_{1,i}^{(lin)} \mathbf{f}_i^{(2)}, \ldots, \sum_{i=1}^{N_s} w_{\lvert\mathcal{Y}\rvert,i}^{(lin)} \mathbf{f}_i^{(2)}\right]^\top$	$\begin{bmatrix} w_{1,1}^{(lin)}, \ldots, w_{\lvert\mathcal{Y}\rvert,1}^{(lin)} \\ \vdots \\ w_{1,N_s}^{(lin)}, \ldots, w_{\lvert\mathcal{Y}\rvert,N_s}^{(lin)} \end{bmatrix}$
$\mathbf{f}^{(3)} \equiv \text{Tanh}(\mathbf{f}^{(2)})$	$\left[\tanh\left(\mathbf{f}_1^{(2)}\right), \ldots, \tanh\left(\mathbf{f}_{N_s}^{(2)}\right)\right]^\top$	$\left[d \tanh\left(\mathbf{f}_1^{(2)}\right), \ldots, d \tanh\left(\mathbf{f}_{N_s}^{(2)}\right)\right]^\top I$
$\mathbf{f}^{(2)} \equiv \text{Max}\left(\mathbf{f}^{(1)}\right)$	$\begin{bmatrix} \max\left(\left[f_{1,1}^{(1)}, \ldots, f_{1,\lvert\mathbf{Z}^{(1)}\rvert}^{(1)}\right]\right) \\ \vdots \\ \max\left(\left[f_{N_s,1}^{(1)}, \ldots, f_{N_s,\lvert\mathbf{Z}^{(N_s)}\rvert}^{(1)}\right]\right) \end{bmatrix}$	$\begin{bmatrix} \mathbb{I}\left(f_{1,1}^{(1)} = \max\left(f_{1,:}^{(1)}\right)\right), \ldots, \mathbb{I}\left(f_{N_s,1}^{(1)} = \max\left(f_{N_s,:}^{(1)}\right)\right) \\ \vdots \\ \mathbb{I}\left(f_{1,\lvert\mathcal{Z}^{(1)}\rvert}^{(1)} = \max\left(f_{1,:}^{(1)}\right)\right), \ldots, \mathbb{I}\left(f_{N_s,\lvert\mathcal{Z}^{(1)}\rvert}^{(1)} = \max\left(f_{N_s,:}^{(1)}\right)\right) \end{bmatrix}$
$\mathbf{f}^{(1)} \equiv \text{Conv}(x)$	$\begin{bmatrix} \text{SSM}_1\left(x, \mathbf{z}_1^{(1)}\right), \ldots, \text{SSM}_1\left(x, \mathbf{z}_{\lvert\mathcal{Z}^{(1)}\rvert}^{(1)}\right) \\ \vdots \\ \text{SSM}_{N_s}\left(x, \mathbf{z}_1^{(N_s)}\right), \ldots, \text{SSM}_{N_s}\left(x, \mathbf{z}_{\lvert\mathcal{Z}^{(N_s)}\rvert}^{(N_s)}\right) \end{bmatrix}$	

Table 7.2: The table above describes the composition of each layer in the Subsequence Network and gives an expression for the Jacobian with respect to the layer's input. The values of each layer are given by the vector $\mathbf{f}^{(h)}$ for hidden layer h. The Jacobian of the first layer (Conv) with respect to the input is not used during inference.

$$\mathbf{f}^{(4)} \stackrel{def}{=} f^{(4)}\left(f^{(3)}\left(f^{(2)}\left(f^{(1)}\left(\mathbf{x}_n\right)\right)\right)\right) \stackrel{def}{=} \text{Lin}\left(\text{Tanh}\left(\text{Max}\left(\text{Conv}\left(\mathbf{x}_n\right)\right)\right)\right) \tag{7.5}$$

where *Conv* is a convolutional layer containing multiple convolutional units, *Max* computes the maximum over the responses of each convolutional unit, *Tanh* is the hyperbolic tangent function and maps values in the range $(-\infty, \infty)$ to $(-1, 1)$, *Lin* is a linear layer. As in Section 7.1.1, we denote the output from hidden layer h as $\mathbf{f}^{(h)}$. Table 7.2 gives the full form of each layer of the network, and Table 7.1 gives a description of network parameters.

If the number of hidden states for each SSM, $|\mathcal{Z}^{(i)}|$, across an individual network is the same ($|\mathcal{Z}^{(i)}| = |\mathcal{Z}^{(j)}| \quad \forall i, j$) layer $\mathbf{f}^{(1)}$ becomes a matrix of size $N_s \times |\mathcal{Z}|$, where N_s is the number of SSMs in the convolutional layer. In the L-SSM and Pair-SSMs, $|\mathcal{Z}|$ is exponential in the size of the input sequence. We make the feed-forward and backpropagation steps for the network tractable by computing the composition $\text{Max}(\text{Conv}(x))$ directly using the

Viterbi algorithm [2]. The locations of the non-zero indicator functions in the Jacobian of the Max layer are then given by the Viterbi path [2] through the SSM.

Our Subsequence Network incorporates the hyperbolic tangent squashing function, $\tanh(x) = \frac{e^{-2x}+1}{e^{-2x}-1}$. We denote the derivative of this function with respect to the hyperbolic tangent input as $d\tanh(x) = 1 - \tanh^2(x)$. In the Max layer, the function $\max(\vec{v})$ returns the largest scalar element of the vector $\vec{v}$.

7.2.5 Training Subsequence Networks

Training is performed using stochastic gradient descent (SGD). In SGD, the gradient of the objective is evaluated for each training example. The gradient is then scaled by a learning rate and subtracted from the current set of parameters to obtain a new set of parameters. This procedure contrasts with batch gradient learning where the gradient is computed for the entire set of training examples. We compute gradients using the backpropagation procedure [102]. Our model includes a locally non-smooth Max function, causing the the gradient of the objective to be undefined at the non-smooth points. To deal with this potential issue, we skip the gradient update in these non-smooth areas [39].

SGD updates take the form

$$\mathbf{w}_t^{all} \leftarrow \mathbf{w}_{t-1}^{all} - \eta_t \frac{\partial F(\mathbf{x}_n)}{\partial \mathbf{w}^{all}} \tag{7.6}$$

where $F(\mathbf{x}_n) = \ell\left(\mathbf{x}_n, y_n; \mathbf{w}^{all}\right) + \frac{\lambda}{2}\left|\left|\mathbf{w}^{all}\right|\right|^2$ is the objective for a single sequence, t indicates the iteration number in the SGD algorithm, $\mathbf{w}_t^{all}$ indicates the value of the weights at the t^{th} iteration, and η_t is the learning rate at iteration t and has the form $\eta_t = \eta_0\left(1 + \lambda\eta_0 t\right)^{-1}$, where λ is the regularization parameter.

The gradient with respect to the linear weights is given by

$$\frac{\partial F(x_n)}{\partial w_{yi}^{(lin)}} = -\frac{\partial \ell\left(\mathbf{x}_n, y_n; \mathbf{w}^{all}\right)}{\partial \mathbf{f}_y^{(4)}} \mathbf{f}_i^{(3)} \tag{7.7}$$

This leads to an update where w_{yi} (the linear weight associated with the i^{th} SSM and category y) is increased if the current training example matches the weight's category and decreased otherwise. The change in the weight's value is proportional to, $\mathbf{f}_i^{(3)}$, the squashed response of the i^{th} SSM. The expression for $\frac{\partial \ell\left(\mathbf{x}_n, y_n; \mathbf{w}^{all}\right)}{\partial \mathbf{f}_y^{(4)}}$ is given in Table 7.2.

The gradient with respect to the SSM weights is given by

$$\frac{\partial F(x_n)}{\partial \mathbf{w}^{(i)}} = \frac{\partial \ell\left(x_n, y_n\right)}{\partial \mathbf{f}_i^{(3)}} \frac{\partial \mathbf{f}_i^{(3)}}{\partial \mathbf{w}^{(i)}} \tag{7.8}$$

where

$$\frac{\partial \ell}{\partial \mathbf{f}_i^{(3)}} = \left(\sum_{y \in \mathcal{Y}} \frac{\exp\left(\sum_{i=1}^{N_s} w_{yi}^{(lin)} \mathbf{f}_i^{(3)}\right) w_{yi}^{(lin)}}{\sum_{y' \in \mathcal{Y}} \exp\left(\sum_{i=1}^{N_s} w_{y'i}^{(lin)} \mathbf{f}_i^{(3)}\right)} - w_{y_n i}^{(lin)} \right) \tag{7.9}$$

The gradient of $\mathbf{f}_i^{(3)}$ with respect to the SSM weights, $\mathbf{w}^{(i)}$, is given by

$$\frac{\partial \mathbf{f}_i^{(3)}}{\partial \mathbf{w}^{(i)}} = d\tanh\left(\mathrm{SSM}_i\left(x_n\right)\right) \mathbf{n}_{\mathbf{z}_{max}}^{(i)} \tag{7.10}$$

where $\mathbf{n}_{z_{max}}^{(i)}$ is a vector of counts of emissions and transitions associated with the set of hidden states that maximizes the value of $\mathrm{SSM}_i\left(\mathbf{x}_n, \mathbf{z}\right)$ and $d\tanh$ is the derivative of the tanh function with respect to its input. As in the HMM, $\mathbf{z}_{max}$ for the SSM can be computed efficiently with the Viterbi algorithm [2]. Gradient steps therefore change $\mathbf{w}^{(i)}$ in proportion

Dataset	# Train	# Test	# Categories
SF	2948	1366	54
FD	2196	2155	23
EC	379	110	7
GO	115	57	23

Table 7.3: Datasets Sizes - # Train indicates the average number of sequences in the training set over all categories, # Test indicates the average number of test set sequences, and # Categories indicates the number of one-versus-rest classification problems defined by the dataset.

to the counts of emissions and transitions in the highest-scoring set of hidden states.

7.3 Experiments

We perform classification experiments on four protein datasets. Of these datasets, two are derived from the Structural Classification of Proteins (SCOP) [75] version 1.53 (see Section 2.4). The first structural dataset [96], denoted by SF, defines 54 fixed superfamily partitions. The second dataset [16], FD, consists of 23 predefined partitions at the fold level. Both of the SCOP datasets were constructed so that no overlap between lower levels in the hierarchy occurs between training and test sets.

The other protein datasets divide sequences into functional, rather than structural, categories. The enzyme classification dataset [103], which we refer to as EC, contains sequences from six enzyme categories and a set of non-enzymes for a total of 7 one-versus-rest datasets. The fourth dataset [103] categorizes proteins by Gene Ontology. We refer to this dataset as GO. Information about the protein datasets is given in Table 7.3.

7.3.1 Comparative Classifiers

We compared the three SVM string kernels to our Subsequence Network. The BLAST kernel was computed by performing a BLAST [59] database search on each sequence. If another sequence from the training set was returned by the BLAST search, then we set the corresponding Kernel value to the returned E-value. The mismatch kernel was described in

Section 2.2 and has two parameters. We denote a mismatch evaluation by Mismatch(k, m), where k is the subsequence length and m is the number of allowable mismatches. The LA-Kernel was also described in Section 2.2. For all experiments, the LA-Kernel's temperature parameter, β, was set to 0.2.

7.3.2 Models and Parameters

We compared three variations of our Subsequence Network. In the first variation, "Pair-SSM," the convolutional layer consisted of Pair-SSMs associated with each training sequence in the model. Similarly, the "L-SSM" and "SL-SSM" variations use L-SSMs and SL-SSMs in the convolutional layer respectively.

For the Pair-SSM network, we initialized pairwise weights using a scaled version of the BLOSUM62 matrix [15] and ran inference for 5 epochs on the FD dataset and 10 epochs on the EC and GO datasets. We set the precision parameter associated with the Gaussian regularizer to $\lambda = .005$. We set multiplicative factor in the learning rate (Section 7.2.5) $\eta_0 = .1$ for the linear weights. For the Pair-SSM parameters, we set $\eta_0 = \frac{.1}{10 \times (\#\text{Train})}$, where # Train is the number of training set sequences. To allow training of the Pair-SSM model to take place in a reasonable amount of time, we distributed gradient computations of SSMs in the convolutional layer within each backpropagation step over 50 machines[1].

Choices of parameters were the same for the L-SSM and SL-SSM networks: We used 96 SSMs in the convolutional layer and set K (the number of states for each SSM) to 11. We set the precision parameter to $\lambda = .005$ for both SSM weights and linear weights, and we set $\eta_0 = .1$. For each experiment, we ran inference for 30 epochs.

The weight vector for these models were set by generating subsequences, x, of length K uniformly and at random. For position k in the subsequence, weight w_{kx_k} was set to $\frac{1}{K}$ and weights w_{km}, $m \neq x_k$ was set to $-\frac{1}{K}$.

[1] Training the (S)L-SSM networks was significantly faster than the Pair-SSM network. To give a rough comparison, SL-SSM network training with the parameters described was faster than computing the Mismatch(5,2) kernel.

To compensate for unbalanced numbers of positive and negative examples, we oversampled the positive training set so that the same number of positive and negative examples were presented to the SGD trainer during each epoch. In the (S)L-SSM networks, we found that initializing the linear weights so that half of the SSMs were associated with the positive class and the other half associated with the negative class improved performance of our algorithm.

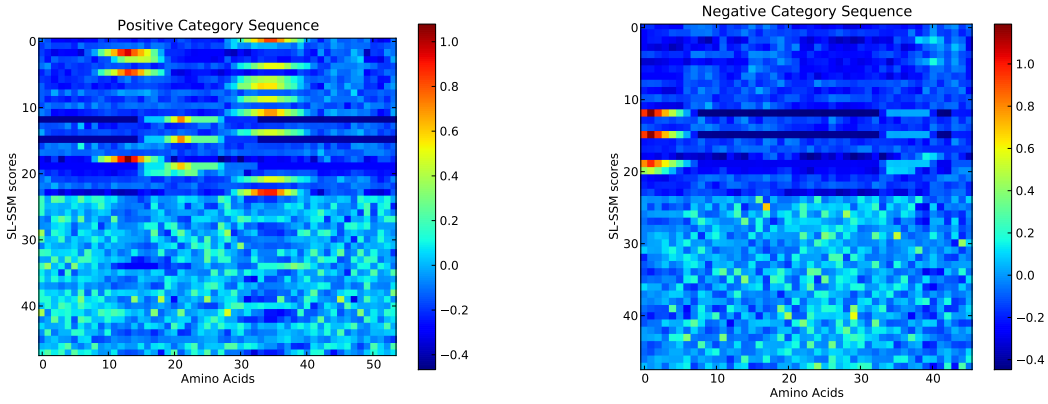
7.3.3 Evaluation Metrics

We measured the performance of our algorithm by computing the average area under the ROC curve (AUC) for each of the one-versus-rest classification problems defined by our datasets and either AUC_{50} or $AUC_{10\%}$ scores depending on the dataset. To compare the performance of different models, we performed the Wilcoxson signed rank tests at a 5% significance level using each one-versus-rest category. We report AUC results based on the algorithm's scoring of sequences on the test sets.

7.3.4 Synthetic Experiments

We constructed a synthetic dataset to verify that our network can detect the relevant subsequence features that we propose will lead to good protein sequence classification performance. Specifically, we generated 1000 sequences with lengths generated from a Poisson distribution with a mean of 50 symbols. Each sequence contained between one and three fixed relevant subsequences, with the positive class containing all three subsequences and the negative class containing either one or two sequences of any type. The relevant subsequences were arranged in random order within the sequence. After placement of the relevant sequences, noise was added - we replaced each every relevant subsequence amino acid with a random amino acid with 10% probability. Amino acids outside relevant subsequences were generated from a uniform multinomial distribution.

Figure 7.3 shows responses of the lowest layer in the SL-SSM network on two example sequences from the synthetic dataset. Strong responses in portions of the sequences that



(a) (b)

Figure 7.3: The figures above show responses from each of the 48 unnormalized SL-SSMs over each length 7 subsequence for a sequence generated from the positive class (a) and the negative class (b). The positive example contains all three relevant subsequences while the negative example contains only one relevant sequence. The first 24 SL-SSMs (top half both figures) were constrained to be associated with the positive category, while the last 24 were constrained to be associated with the negative category. The heat maps show that sets of positive SL-SSMs have adapted to each of the three relevant subsequences in the synthetic dataset - both the three relevant subsequences in the positive example and the one relevant subsequence in the negative example were detected by subsets of the first 24 SL-SSMs. In contrast, SL-SSMs associated with the negative category learn a background distributions of symbols.

contain relevant subsequences indicate that our model is able to effectively learn features that discriminate well for a dataset where categories are determined based on the presence of subsequences. The SL-SSM achieves an AUC of .9998 and AUC_{50} of .997 on the test portion of the synthetic dataset, showing that after detecting relevant subsequences, our model has the ability to effectively classify sequences generated according to a relevant subsequences assumption on the dataset.

7.3.5 Parameter Adjustment

We adjusted the parameters of (S)L-SSM models by comparing both the number of SSMs in each network and the number of hidden states for the SSMs (we used the same

Hidden States	AUC	AUC_{50}
7	.801	.146
9	.813	.145
11	.815	.153
13	.815	.153
15	.807	.163

(a)

# SL-SSMs	AUC	AUC_{50}
64	.812	.144
80	.812	.156
96	.814	.153
112	.816	.145
128	.816	.147

(b)

Table 7.4: Average ROC results for different settings of the SL-SSM network on the FD dataset. ROCs were averaged over ten independent trials initialized with random pattern weights. When varying the number of SL-SSM hidden states in (a), 96 SL-SSMs were used in the network. In (b), 11 hidden states were used for each SL-SSM when varying the number of SL-SSMs.

number of hidden states for all SSMs) in experiments on the FD dataset. Table 7.4 shows results from these comparisons. Within a relatively wide range of parameter settings, the performance of the model stays roughly the same, showing that, although our method may require some adjustment using cross validation, micromanagement of parameter settings is not critical to maintaining acceptable performance. For the results shown in Table 7.5, we selected parameter settings for the other experiments ($K = 11$ with 96 convolutional units) that performed best on FD.

7.3.6 Protein Classification Experiments

We compared four subsequence models used in the convolutional layer of our Subsequence Network. The lower rows of Table 7.5a and b show results for the L-SSM and the SL-SSM. These indicate that the simpler model, the SL-SSM outperforms the L-SSM. Superior performance of the SL-SSM results from unstable inference in the L-SSM when attempting to learn insert transition weights. If these weights grow above zero for the positive class, then the model tends to explain every protein sequence using long sequences of insert states. For this reason, we fix the insert transition weights in the L-SSM to small negative values ($-\frac{1}{2K}$ where K is the number of Match states in the L-SSM), but this fix affects the flexibility of the model. On the FD dataset, the Pair-SSM outperforms our other models in both AUC

and AUC_{50}. However, on the functional datasets, both L-SSM and SL-SSM outperform the Pair-SSM. These results indicate that the simpler (S)L-SSM assumptions may be better models of protein structure in certain cases.

Table 7.5 also compares our subsequence networks to SVM/kernel methods from the literature. Compared to the LA-kernel on the FD dataset, the Pair-SSM model is statistically equivalent in both AUC and AUC_{50} measurements. Compared to the Mismatch kernel on the FD dataset, the Pair-SSM model performs better in AUC but is equivalent for AUC_{50}. We note, however, that the β parameter used for the LA-kernel is the best of many settings on both the SF and FD training/test set split. Due to this extensive adjustment on the FD dataset, it is likely that the LA-kernel overfits. In contrast, Pair-SSM network performance is only weakly dependent on parameter settings (the regularization parameter) and we did not perform extensive adjustment of these values, so overfitting likely to be less problematic for our Pair-SSM on the FD test set. The Pair-SSM is equivalent to both the LA-kernel and Mismatch kernel on the EC dataset in both AUC and $AUC_{10\%}$. On the GO dataset, the the Pair-SSM is outperformed by the LA-kernel in both AUC and $AUC_{10\%}$ but is equivalent to the Mismatch kernel.

In AUC, the LA-kernel outperforms the L-SSM and SL-SSM models on the SF and GO datasets but is statistically equivalent for the FD and EC datasets. In AUC_{50}, the LA-kernel outperforms the L-SSM and SL-SSM on both the SF and FD datasets at a 5% significance level. In $AUC_{10\%}$, both of our methods outperform the LA-kernel on the EC dataset. On the GO dataset, the LA-kernel's performance is statistically equivalent to the SL-SSM but outperforms the L-SSM. Compared to the Mismatch(5,2) kernel, both the L-SSM and SL-SSM models have equivalent performance in SF and FD in both AUC and AUC_{50}. Our algorithms outperform the Mismatch kernel in $AUC_{10\%}$ on both of the functional datasets.

For the EC dataset, none of the algorithms perform particularly well. It is possible that both the size of the dataset and weak correlations between sequence and function cause subsequence-based approaches to fail. A class of methods based on a different set of assumptions may be necessary to achieve strong functional classification performance.

Dataset	SF		FD	
Method	AUC	AUC_{50}	AUC	AUC_{50}
BLAST	.756	.341	.603	.100
Mismatch (5,1)	.872	.403	.802	.146
Mismatch (5,2)	.888	.479	.782	.188
LA-Kernel($\beta = .2$)	.926	.663	.805	.216
Pair-SSM	-	-	0.826	0.168
L-SSM	0.901 ± 0.003	0.456 ± 0.013	0.804 ± 0.006	0.141 ± 0.010
SL-SSM	0.903 ± 0.003	0.503 ± 0.012	0.815 ± 0.009	0.153 ± 0.008

(a)

Dataset	EC		GO	
Method	AUC	$AUC_{10\%}$	AUC	$AUC_{10\%}$
Mismatch (4,1)	.580	.007	.740	.238
Mismatch (5,2)	.581	.008	.747	.245
LA-Kernel($\beta = .2$)	.574	.004	.801	.339
Pair-SSM	.544	.021	.737	.264
L-SSM	0.587 ± 0.018	0.081 ± 0.011	0.736 ± 0.004	0.280 ± 0.010
SL-SSM	0.583 ± 0.017	0.092 ± 0.024	0.731 ± 0.011	0.306 ± 0.018

(b)

Table 7.5: AUC results for the FD and SF datasets (a) and the EC and GO datasets (b). Because our model is non-convex, we report means and standard deviations of AUCs from multiple starting points in the SSM weight space. Ten trials were averaged for both the L-SSM and SL-SSM models for both structural and functional datasets. Due to the length of Pair-SSM network's runtime, we report results from only a single trial.

7.4 Conclusions

The empirical kernel map applied in conjunction with SVM classifiers is strongly related to feed-forward models like convolutional neural networks. Based on this relationship, we show how to construct a family of models, which we call Subsequence Networks, where kernel parameters can be learned in conjunction with linear classification boundaries. Our Subsequence Networks operate differently from state-of-the-art protein sequence classification models yet can achieve comparable performance. We hope that Subsequence Networks can shift the focus in biological sequence classification from increasingly fine-tuned kernel methods toward developing structures with self-tuning abilities.

Our networks also contribute to existing neural network literature by extending the convolutional layer to a maximization over latent parameter spaces in standard sequence models. The effectiveness of this framework for protein sequence classification shows that it has potential in other classification domains.

Chapter 8: Conclusions and Future Work

In this thesis, I have shown that incorporating structure into models over sequences can aid in tasks like classification and motif finding.

The Hidden Markov Model Variant incorporates structure in the form of the common pairwise dependency assumption between sequence elements. The structure of the HMM Variant strategically replicates the single transition probability distribution that would be used to by the standard HMM model a set of sequences. This structure allows it to extract fixed length representations of variable length sequences which can then be used for classification or other tasks.

The Profile HMM also incorporates a characteristic structure that is well suited toward modeling protein sequences. Our infinite extension of this model relies on a procedure that transforms the Profile HMM into a standard HMM. We can then exploit this transformed structure to define a beam method that both makes approximate inference possible in the infinite model and speeds inference in the finite model.

The local Profile HMM models described in Chapter 5 combine structure from multiple Profile HMMs. We show how the latent space from multiple SL-pHMMs can be merged to produce a single unified probabilistic model over sequences. The latent structure from these models can then be attached to discriminative machinery, producing a model over both sequences and labels. This combined model can be applied in sequence classification tasks.

A relaxation of the Factorial SL-pHMM latent space gives rise to the Relevant Subsequence Sparse Dictionary Learning (RS-DL) model. By allowing this latent space to take on continuous values subject to a sparse regularizer, the RS-DL model generates sequence elements through linear combinations of dictionary vectors. The relevant subsequences discovered by the model are often well correlated with human-recognizable motifs and can also

be used effectively to classify sequences.

Our family of feed forward networks take advantage of Hidden Markov Model structure to extract features from sequence data. In this case, HMM structure is incorporated into the lower level of a neural network. These networks are highly effective for classifying protein sequences, achieving results competitive with state-of-the-art Kernel methods on canonical datasets.

Postulating a latent generating structure can be a powerful tool for finding pertinent information in sequence datasets. The work described in this thesis demonstrates how structure from HMMs can be incorporated into a diverse set of models over sequences and further shows that this structure can aid in solving problems of classification and discovery.

8.1 Future Directions

An unanswered question in much of the work in this thesis is "Why do these models work?". In general, although we have shown experimentally how our models behave when trained and tested on certain datasets, we would like to know more exactly what conditions must be satisfied to achieve a certain level of operation. For instance, if we assume that in a set of sequences with discrete elements, each sequence contains a certain number of relevant subsequences, what level of noise prevents exact recovery of this set? Bayesian formulations of probabilistic models and well-defined optimization problems allow us to specify structure and solve for parameters but leave these types of questions unanswered.

Another interesting area of future research involves developing methods to evaluate assumptions about a given dataset. We may hypothesize, for instance, that protein sequences can be separated into structural categories by extracting features that measure to what degree a small subset of subsequences is present within a sequence. In this work, our method of validating this type of assumption involved building a model that made the assumption and evaluating the performance of the model on our original dataset. However, it seems possible that more effective methods for evaluating these structural assumptions exist. Faster methods would speed the search for structural assumptions that work well, allowing better

models to be developed more quickly.

Finally, there is much room for developing better models of sequences across many domains. In the domain of biological sequences, an interesting direction pursued by a number of authors involves methods that incorporate information about long-range interactions within protein sequences [104, 105]. However, this type of approach seems useful primarily on a small set of protein structural categories. It would be interesting to extend these approaches to infer correlations between subsequences, rather than assuming that these correlations are given beforehand, and to use this information for either classification or for identifying properties of individual amino acids.

Appendix A: Variational Algorithm for the HMM Variant

Parameter	Description
N	the number of sequences
T_n	the length of sequence n
K	the number of hidden symbols
M	the number of observed symbols
a_i	start state probabilities, where i is the value of the first hidden state
A_{nij}	transition probabilities, where n indicates the sequence, i the originating hidden state, and j the destination hidden state
B_{im}	emission probabilities, where i indicates the hidden state, and m the observed symbol associated with the hidden state
z_{nt}	the hidden state at position t in sequence n
x_{nt}	the observed state at position t in sequence n
γ	Dirichlet prior parameter for a
α	Dirichlet prior parameter for A
β	Dirichlet prior parameter for B
h_{nti}	Variational parameter that approximates the mean of z_{nti}
$\tilde{\gamma}$	Variational parameter that approximates the Dirichlet prior for a
$\tilde{\alpha}_{ni}$	Variational parameter that approximates the Dirichlet prior for A_{ni}
$\tilde{\beta}_i$	Variational parameter that approximates the Dirichlet prior for B_i

Figure A.1: Parameters used in the mean field variational algorithm

A.1 HMM Variant Probability

$$
\begin{aligned}
& p(x, z, a, A, B|\gamma, \alpha, \beta) \\
&= \left(\frac{\Gamma(\sum_i \gamma_i)}{\prod_i \Gamma(\gamma_i)} \prod_i a_i^{\gamma_i - 1}\right) \left(\prod_{ni} \frac{\Gamma(\sum_j \alpha_{nij})}{\prod_j \Gamma(\alpha_{nij})} \prod_j A_{nij}^{\alpha_{nij}-1}\right) \qquad \text{(A.1)} \\
&\quad \left(\prod_i \frac{\Gamma(\sum_m \beta_{im})}{\prod_m \Gamma(\beta)_{im}} \prod_m B_{im}^{\beta_{im}-1}\right) \prod_n a_{z_1} \prod_{t=2}^{T_n} A_{n z_{t-1} z_t} \prod_t B_{z_t x_t} \\
&= \left(\frac{\Gamma(\sum_i \gamma_i)}{\prod_i \Gamma(\gamma_i)} \prod_i a_i^{\gamma - 1}\right) \left(\prod_{ni} \frac{\Gamma(\sum_j \alpha_{nij})}{\prod_j \Gamma(\alpha_{nij})} \prod_j A_{nij}^{\alpha-1}\right) \\
&\quad \left(\prod_i \frac{\Gamma(\sum_m \beta_{im})}{\prod_m \Gamma(\beta)_{im}} \prod_m B_{im}^{\beta-1}\right) \prod_i a_i^{n_i} \prod_{nij} A_{nij}^{n_{nij}} \prod_{im} B_{im}^{n_{im}} \\
&= \left(\frac{\Gamma(\sum_i \gamma_i)}{\prod_i \Gamma(\gamma_i)} \prod_i a_i^{\gamma_i - 1 + n_i}\right) \left(\prod_{ni} \frac{\Gamma(\sum_j \alpha_{nij})}{\prod_j \Gamma(\alpha_{nij})} \prod_j A_{nij}^{\alpha_{nij}-1+n_{nij}}\right) \\
&\quad \left(\prod_i \frac{\Gamma(\sum_m \beta_{im})}{\prod_m \Gamma(\beta)_{im}} \prod_m B_{im}^{\beta_{im}-1+n_{im}}\right)
\end{aligned}
$$

A.1.1 Mean Field Variational Approximation

$$
\begin{aligned}
q(z, a, A, B) =& q(a) \prod_{n=1}^{N} \prod_{i=1}^{K} q(A_{ni}) \prod_{i=1}^{K} q(B_i) \prod_{nt} q(z_{nt}) \qquad \text{(A.2)} \\
&= \left(\frac{\Gamma(\sum_i \tilde{\gamma}_i)}{\prod_i \Gamma(\tilde{\gamma}_i)} \prod_i a_i^{\tilde{\gamma}_i - 1}\right) \left(\prod_{ni} \frac{\Gamma(\sum_j \tilde{\alpha}_{nij})}{\prod_j \Gamma(\tilde{\alpha}_{nij})} \prod_j A_{nij}^{\tilde{\alpha}_{nij}-1}\right) \\
&\quad \left(\prod_i \frac{\Gamma(\sum_m \tilde{\beta}_{im})}{\prod_m \Gamma(\tilde{\beta}_{im})} \prod_m B_{im}^{\tilde{\beta}_{im}-1}\right) \prod_{nti} h_{nti}^{z_{nti}} \qquad \text{(A.3)}
\end{aligned}
$$

Using the standard variational formulation [31], we construct $\mathcal{F}(q)$ by applying Jensen's

inequality to create a lower bound on the marginal likelihood:

$$\mathcal{F}(q) = \int da \int dA \int dB \sum_{\vec{z}} q(z, a, A, B) \log \frac{p(x, z, a, A, B|\alpha, \beta)}{q(z, a, A, B)} \tag{A.4}$$

$$= E_q \left[\log p(x, z, a, A, B|\alpha, \beta)\right] - E_q \left[\log q(z, a, A, B)\right]$$

$$\begin{aligned}
\mathcal{F}(q) = & \log \Gamma(\sum_i \gamma_i) - \sum_i \log \Gamma(\gamma_i) + \sum_i (\gamma_i - 1) E_q \left[\log a_i\right] \\
& \sum_{ni} \left(\log \Gamma(\sum_j \alpha_{nij}) - \sum_j \log \Gamma(\alpha_{nij}) \right) + \sum_{nij} (\alpha_{nij} - 1) E_q \left[\log A_{nij}\right] \\
& \sum_i \left(\log \Gamma(\sum_m \beta_{im}) - \sum_m \log \Gamma(\beta_{im}) \right) + \sum_{im} (\beta_{im} - 1) E_q \left[\log B_{im}\right] \\
& \sum_i E_q \left[n_i \log a_i\right] + \sum_{nij} E_q \left[n_{nij} \log A_{nij}\right] + \sum_{im} E_q \left[n_{im} \log B_{im}\right] \\
& - \log \Gamma(\sum_i \tilde{\gamma}_i) + \sum_i \log \Gamma(\tilde{\gamma}_i) - \sum_i (\tilde{\gamma}_i - 1) E_q \left[\log a_i\right] \\
& - \sum_{ni} \log \Gamma(\sum_j \tilde{\alpha}_{nij}) + \sum_{nij} \log \Gamma(\tilde{\alpha}_{nij}) - \sum_{nij} (\tilde{\alpha}_{nij} - 1) E_q \left[\log A_{nij}\right] \\
& - \sum_i \log \Gamma(\sum_m \tilde{\beta}_{im}) + \sum_{im} \log \Gamma(\tilde{\beta}_{im}) - \sum_{im} (\tilde{\beta}_{im} - 1) E_q \left[\log B_{im}\right] \\
& - \sum_{nti} E_q \left[z_{nti}\right] \log h_{nti}
\end{aligned} \tag{A.5}$$

A.1.2 Expectations

The expectations of $\log a$, $\log A$, and $\log B$ are given by the formula for expectations of the log of the parameters under the Dirichlet distribution [46].

$$E_q\left[\log a_i\right] = \Psi(\tilde{\gamma}_i) - \Psi(\sum_{i'} \tilde{\gamma}_{i'}) \tag{A.6}$$

$$E_q\left[\log A_{nij}\right] = \Psi(\tilde{\alpha}_{nij}) - \Psi(\sum_{j'} \tilde{\alpha}_{nij'}) \tag{A.7}$$

$$E_q\left[\log B_{im}\right] = \Psi(\tilde{\beta}_{im}) - \Psi(\sum_{m'} \tilde{\beta}_{im'}) \tag{A.8}$$

$$E_q\left[n_i\right] = \sum_n h_{n1i} \tag{A.9}$$

$$\begin{aligned}
E_q\left[n_{nij}\right] &= \sum_Z n_{nij} \prod_{nti'} h_{nti'}^{z_{nti'}} \qquad (A.10)\\
&= \sum_Z \left(\sum_{t=2}^{T_n} I(z_{nt-1i}) I(z_{ntj}) \right) \prod_{nti'} h_{nti'}^{z_{nti'}} \\
&= \sum_Z \left(\sum_{t=2}^{T_n} I(z_{nt-1i}) I(z_{ntj}) \prod_{nti'} h_{nti'}^{z_{nti'}} \right) \\
&= \sum_{t=2}^{T_n} \left(\sum_Z I(z_{nt-1i}) I(z_{ntj}) \prod_{nti'} h_{nti'}^{z_{nti'}} \right) \\
&= \sum_{t=2}^{T_n} \left(h_{nt-1i} h_{ntj} \sum_{Z^{\neg z_{nt-1} z_{nt}}} \prod_{nti'} h_{nti'}^{z_{nti'}} \right) \\
&= \sum_{t=2}^{T_n} h_{nt-1i} h_{ntj}
\end{aligned}$$

$$E_q\left[n_{im}\right] = \sum_{nt:x_{nt}=m} h_{nti} \tag{A.11}$$

$$E_q\left[z_{nti}\right] = h_{nti} \tag{A.12}$$

A.1.3 Maximize $F(q)$ with respect to $\tilde{\gamma_i}$

Here we use a Dirichlet prior on a with uniform parameters γ.

$$\mathcal{L}(\tilde{\gamma_i}) = (\gamma - 1) \sum_i \left(\Psi(\tilde{\gamma_i}) - \Psi(\sum_{i'} \tilde{\gamma}_{i'}) \right) \tag{A.13}$$

$$+ \sum_i \sum_n h_{n1i} \left(\Psi(\tilde{\gamma_i}) - \Psi(\sum_{i'} \tilde{\gamma}_{i'}) \right) \tag{A.14}$$

$$- \log\Gamma(\sum_i \tilde{\gamma_i}) + \sum_i \log\Gamma(\tilde{\gamma_i})$$

$$- \sum_i (\tilde{\gamma_i} - 1) \left(\Psi(\tilde{\gamma_i}) - \Psi(\sum_{i'} \tilde{\gamma}_{i'}) \right) \tag{A.15}$$

$$= \sum_i \left(\Psi(\tilde{\gamma_i}) - \Psi(\sum_{i'} \tilde{\gamma}_{i'}) \right) \left(\sum_n h_{n1i} + \gamma - \tilde{\gamma_i} \right)$$

$$- \log\Gamma(\sum_i \tilde{\gamma_i}) + \sum_i \log\Gamma(\tilde{\gamma_i})$$

$$\frac{\partial \mathcal{L}(\tilde{\gamma_i})}{\partial \tilde{\gamma_i}} = \sum_i \Psi'(\sum_{i'} \tilde{\gamma}_{i'}) \left(\sum_n h_{n1i} + \gamma - \tilde{\gamma_i} \right) \tag{A.16}$$

$$- \Psi'(\tilde{\gamma_i}) \left(\sum_n h_{n1i} + \gamma - \tilde{\gamma_i} \right) \tag{A.17}$$

Setting the partial derivative to 0, we find the update that maximizes $\tilde{\gamma_i}$ below:

$$\tilde{\gamma_i} = \sum_n h_{n1i} + \gamma \tag{A.18}$$

A.1.3.1 Maximize $F(q)$ with respect to $\tilde{\alpha}_{nij}$

Here we use a Dirichlet prior on A with uniform parameters α.

$$\mathcal{L}(\tilde{\alpha}_{nij}) =(\alpha - 1) \sum_{nij} \left(\Psi(\tilde{\alpha}_{nij}) - \Psi(\sum_{j'} \tilde{\alpha}_{nij'}) \right) \tag{A.19}$$

$$+ \sum_{nij} \sum_t h_{nt-1i} h_{ntj} \left(\Psi(\tilde{\alpha}_{nij}) - \Psi(\sum_{j'} \tilde{\alpha}_{nij'}) \right) \tag{A.20}$$

$$- \sum_{ni} \log \Gamma(\sum_j \tilde{\alpha}_{nij}) + \sum_{nij} \log \Gamma(\tilde{\alpha}_{nij})$$

$$- \sum_{nij} (\tilde{\alpha}_{nij} - 1) \left(\Psi(\tilde{\alpha}_{nij}) - \Psi(\sum_{j'} \tilde{\alpha}_{nij'}) \right) \tag{A.21}$$

$$= \sum_{nij} \left(\sum_t h_{nt-1i} h_{ntj} + \alpha - \tilde{\alpha}_{nij} \right) \left(\Psi(\tilde{\alpha}_{nij}) - \Psi(\sum_{j'} \tilde{\alpha}_{nij'}) \right)$$

$$- \sum_{ni} \log \Gamma(\sum_j \tilde{\alpha}_{nij}) + \sum_{nij} \log \Gamma(\tilde{\alpha}_{nij}) \tag{A.22}$$

$$\frac{\partial \mathcal{L}(\tilde{\alpha}_{nij})}{\partial \tilde{\alpha}_{nij}} =\Psi'(\tilde{\alpha}_{nij}) \left(\sum_t h_{nt-1i} h_{ntj} + \alpha - \tilde{\alpha}_{nij} \right) \tag{A.23}$$

$$- \sum_j \Psi'(\sum_{j'} \tilde{\alpha}_{nij'}) \left(\sum_t h_{nt-1i} h_{ntj} + \alpha - \tilde{\alpha}_{nij} \right) \tag{A.24}$$

Setting the partial derivative to 0, we find the update that maximizes $\tilde{\alpha}_{nij}$ below:

$$\tilde{\alpha}_{nij} = \sum_t h_{nt-1i} h_{ntj} + \alpha \tag{A.25}$$

A.1.3.2 Maximize $F(q)$ with respect to $\tilde{\beta}_{im}$

Here we use a Dirichlet prior on B with uniform parameters β

$$\begin{aligned}
\mathcal{L}(\tilde{\beta}_{im}) =& (\beta - 1) \sum_{im} \left(\Psi(\tilde{\beta}_{im}) - \Psi(\sum_{m'} \tilde{\beta}_{im'}) \right) + \\
& \sum_{im} \left(\sum_{nt:x_t=m} h_{nti} \right) \left(\Psi(\tilde{\beta}_{im}) - \Psi(\sum_{m'} \tilde{\beta}_{im'}) \right) - \sum_i \log \Gamma(\sum_m \tilde{\beta}_{im}) + \\
& \sum_{im} \log \Gamma(\tilde{\beta}_{im}) - \sum_{im} (\tilde{\beta}_{im} - 1) \left(\Psi(\tilde{\beta}_{im}) - \Psi(\sum_{m'} \tilde{\beta}_{im'}) \right) \\
=& \sum_{im} \left(\sum_{nt:x_t=m} h_{tni} + \beta - \tilde{\beta}_{im} \right) \left(\Psi(\tilde{\beta}_{im}) - \Psi(\sum_{m'} \tilde{\beta}_{im'}) \right) - \\
& \sum_i \log \Gamma(\sum_m \tilde{\beta}_{im}) + \sum_{im} \log \Gamma(\tilde{\beta}_{im})
\end{aligned} \tag{A.26}$$

Setting the partial derivative to 0, we find the update that maximizes $\tilde{\beta}_{im}$ below:

$$\tilde{\beta}_{im} = \sum_{nt:x_t=m} h_{nti} + \beta \tag{A.27}$$

A.1.3.3 Maximize $F(q)$ with respect to h_{nti}

$$\begin{aligned}\mathcal{L}(h_{nti}) = &\sum_i \left(\sum_n h_{n1i}\right)\left(\Psi(\tilde{\gamma}_i) - \Psi(\sum_{i'}\tilde{\gamma}_{i'})\right) + \\ &\sum_{nij}\left(\sum_{t=2}^{T_n} h_{nt-1i}h_{ntj}\right)\left(\Psi(\tilde{\alpha}_{nij}) - \Psi(\sum_{j'}\tilde{\alpha}_{nij'})\right) + \\ &\sum_{im}\left(\sum_{t=1}^{T_n}\mathbb{I}(x_t = m)h_{nti}\right)\left(\Psi(\tilde{\beta}_{im}) - \Psi(\sum_{m'}\tilde{\beta}_{im'})\right) \\ &-\sum_{nti} h_{nti}\log h_{nti} - \lambda\left(\sum_i h_{nti} - 1\right)\end{aligned} \tag{A.28}$$

$$\frac{\partial\mathcal{L}(h_{nti})}{\partial h_{n't'i'}} = \begin{cases} (\Psi(\tilde{\gamma}_{i'}) - \Psi(\sum_{i''}\tilde{\gamma}_{i''})) + & \\ \quad \sum_j h_{n'2j}\left(\Psi(\tilde{\alpha}_{n'i'j}) - \Psi(\sum_{j'}\tilde{\alpha}_{n'i'j'})\right) + & \\ \quad \left(\Psi(\tilde{\beta}_{ix_{n'1}}) - \Psi(\sum_{m'}\tilde{\beta}_{im'})\right) - & \\ \quad 1 - \log h_{n't'i'} - \lambda & t' = 1 \\ \sum_i h_{n't'-1i}\left(\Psi(\tilde{\alpha}_{n'ii'}) - \Psi(\sum_{j'}\tilde{\alpha}_{n'ij'})\right) + & \\ \quad \sum_j h_{n't'+1j}\left(\Psi(\tilde{\alpha}_{n'i'j}) - \Psi(\sum_{j'}\tilde{\alpha}_{n'i'j'})\right) + & \\ \quad \left(\Psi(\tilde{\beta}_{ix_{n't'}}) - \Psi(\sum_{m'}\tilde{\beta}_{im'})\right) - & \\ \quad 1 - \log h_{n't'i'} - \lambda & 1 < t' < T' \\ \sum_i h_{n'T_{n'}-1i}\left(\Psi(\tilde{\alpha}_{n'ii'}) - \Psi(\sum_{j'}\tilde{\alpha}_{n'ij'})\right) + & \\ \quad \left(\Psi(\tilde{\beta}_{ix_{n'T_{n'}-1}}) - \Psi(\sum_{m'}\tilde{\beta}_{im'})\right) - & \\ \quad 1 - \log h_{n't'i'} - \lambda & t' = T' \end{cases} \tag{A.29}$$

Setting $\frac{\partial\mathcal{L}(h_{nti})}{\partial h_{n't'i'}}$ to zero, we find the expression for h_{nti} below:

$$h_{n't'i'} \propto \begin{cases} \exp\left(\Psi(\tilde{\gamma}_{i'}) - \Psi(\sum_{i''} \tilde{\gamma}_{i''})\right) & \\ \quad + \sum_j h_{n'2j} \left(\Psi(\tilde{\alpha}_{n'i'j}) - \Psi(\sum_{j'} \tilde{\alpha}_{n'i'j'})\right) & \\ \quad + \left(\Psi(\tilde{\beta}_{ix_{n'1}}) - \Psi(\sum_{m'} \tilde{\beta}_{im'})\right) & t' = 1 \\ \exp \sum_i h_{n't'-1i} \left(\Psi(\tilde{\alpha}_{n'ii'}) - \Psi(\sum_{j'} \tilde{\alpha}_{n'ij'})\right) & \\ \quad + \sum_j h_{n't'+1j} \left(\Psi(\tilde{\alpha}_{n'i'j}) - \Psi(\sum_{j'} \tilde{\alpha}_{n'i'j'})\right) & \\ \quad + \left(\Psi(\tilde{\beta}_{ix_{n't'}}) - \Psi(\sum_{m'} \tilde{\beta}_{im'})\right) & 1 < t' < T'_{n'} \\ \exp \sum_i h_{n't'-1i} \left(\Psi(\tilde{\alpha}_{n'ii'}) - \Psi(\sum_{j'} \tilde{\alpha}_{n'ij'})\right) & \\ \quad + \left(\Psi(\tilde{\beta}_{ix_{n'T_{n'}-1}}) - \Psi(\sum_{m'} \tilde{\beta}_{im'})\right) & t' = T'_{n'} \end{cases} \tag{A.30}$$

Appendix B: Infinite pHMM Derivations

B.1 Variational Bound

We bound the marginal likelihood of the Infinite pHMM using the following variational distribution:

$$\begin{aligned}
\log p(x_{1:T}|\alpha,\beta) &= \log \int_A \int_B \sum_{\vec{z}} p(A|\alpha)p(B|\beta)p(x_{1:T},\vec{z}|A,B) \qquad \text{(B.1)}\\
&\geq \int_A \int_B \sum_{\vec{z}} q(\vec{z},A,B) \log \frac{p(A|\alpha)p(B|\beta)p(x_{1:T},\vec{z}|A,B)}{q(\vec{z},A,B)}\\
&= \int_A \int_B \sum_{\vec{z}} q(\vec{z}) \left(\prod_{(s,k)} q(A_{(s,k)})\right) \left(\prod_{(s,k)} q(B_{(s,k)})\right)\\
&\qquad \log \frac{p(A|\alpha)p(B|\beta)p(x_{1:T},\vec{z}|A,B)}{q(\vec{z}) \left(\prod_{(s,k)} q(A_{(s,k)})\right) \left(\prod_{(s,k)} q(B_{(s,k)})\right)}
\end{aligned}$$

After maximizing with respect to $q(A)$ and $q(B)$, $\left(\prod_{(s,k)} q(A_{(s,k)})\right) \left(\prod_{(s,k)} q(B_{(s,k)})\right) = p(A|\alpha)p(B|\beta)p(x_{1:T},\vec{z}|A,B)$. We then compute the variational bound for the truncated model as follows:

$$\begin{aligned}
\log p(x_{1:T}|\alpha,\beta) &\geq \sum_{\vec{z}} q(\vec{z}) \log q(\vec{z}) \qquad \text{(B.2)}\\
&= \sum_{(s,k),s'} E_{q(\vec{z})}\left[n_{(s,k),s'}\right] E_{q(A_{(s,k),:})}\left[\log A_{(s,k),s'}\right]\\
&\qquad + \sum_{(s,k),m} E_{q(\vec{z})}\left[n_{(s,k),m}\right] E_{q(B_{(s,k),:})}\left[\log B_{(s,k),m}\right]
\end{aligned}$$

B.2 Auxiliary Variable Beam Method

B.2.1 Maximum with respect to $q(u_t)$

The variational free energy can be written as

$$\mathcal{F}(q(u_t)) = E_{q(u_t)}\left[\log \frac{\exp E_{q(z_{t-1})}\left[\log p(u_t|z_t, z_{t-1})p(z_t|z_{t-1})p(z_{t-1}, x_{1:t-1})\right]}{q(u_t)}\right] \tag{B.3}$$

$$+ H\left(q(z_{t-1})\right)$$

By Gibb's inequality, the maximum with respect to $q(u_t)$ is therefore

$$q(u_t) \propto \exp\left(E_{q(z_{t-1})}\left[\log p(u_t|z_t, z_{t-1})p(z_t|z_{t-1})p(z_{t-1}, x_{1:t-1})\right]\right) \tag{B.4}$$

with expectations computed as follows:

$$\begin{aligned} E\left[\log p(u_t|z_t, z_{t-1})\right] &= \sum_{z_{t-1}} q(z_{t-1}) \log p(u_t|z_t, z_{t-1}) \\ &= \sum_{z_{t-1}} q(z_{t-1}) \log \mathbb{I}\left(u_t < p(z_t|z_{t-1})\right) - \sum_{z_{t-1}} q(z_{t-1}) \log p(z_t|z_{t-1}) \\ &= \begin{cases} -\infty \\ \quad \exists z_{t-1} : (u_t \geq p(z_t|z_{t-1})) \wedge (q(z_{t-1}) > 0) \\ -\sum_{z_{t-1}} \mathbb{I}\left(q(z_{t-1}) > 0\right) q(z_{t-1}) \log p(z_t|z_{t-1}) \quad \text{o.w.} \end{cases} \end{aligned} \tag{B.5}$$

$$E\left[\log p(z_t|z_{t-1})\right] = \sum_{z_{t-1}:q(z_{t-1})>0} q(z_{t-1}) \log p(z_t|z_{t-1}) \tag{B.6}$$

The reduced expression for $q(u_t)$ becomes

$$q(u_t) \quad \propto \quad \begin{cases} 0 & \\ & \exists z_{t-1} : (u_t \geq p(z_t|z_{t-1})) \wedge (q(z_{t-1}) > 0) \\ E_{q(z_{t-1})}\left[\log p(z_t|z_{t-1})p(z_{t-1}, x_{1:t-1})\right] & \text{o.w.} \end{cases} \tag{B.7}$$

This final expression gives us the form of $q(u_t)$, but the threshold above which $q(u_t) = 0$ depends on the values where $q(z_{t-1}) = 0$. We can therefore choose an initial $q(u_t)$ either by truncating $q(z_{t-1})$, or by providing $q(u_t)$ with an explicit threshold. We choose to use the latter scheme: $q(u_t) = \frac{\mathbb{I}(u_t < \tilde{\theta}_t)}{\tilde{\theta}_t}$ with an initial variational threshold parameter $\tilde{\theta}_t$.

B.2.2 Maximum with respect to $q(z_{t-1})$

The variational free energy can be written as

$$\mathcal{F}(q(z_{t-1})) = E_{q(z_{t-1})}\left[\log \frac{\exp\left(E_{q(u_t)}\left[\log p(u_t|z_t, z_{t-1})\right]\right) p(z_t|z_{t-1})p(z_{t-1}, x_{1:t-1})}{q(z_{t-1})}\right] \tag{B.8}$$

$$+ H\left(q(u_t)\right)$$

By Gibb's inequality, the maximum with respect to $q(z_{t-1})$ is

$$q(z_{t-1}) \propto \exp\left(E\left[\log p(u_t|z_t, z_{t-1})\right]\right) p(z_t|z_{t-1})p(z_{t-1}, x_{1:t-1}) \tag{B.9}$$

$E\left[\log p(u_t|z_t, z_{t-1})\right]$ is computed as follows:

$$
\begin{aligned}
E\left[\log p(u_t|z_t, z_{t-1})\right] &= \int_0^1 q(u_t) \log p(u_t|z_t, z_{t-1}) du_t \\
&= \int_0^{\tilde{\theta}_t} q(u_t) \log \mathbb{I}\left(u_t < p(z_t|z_{t-1})\right) du_t - \log p(z_t|z_{t-1}) \\
&= \begin{cases} -\infty & \tilde{\theta}_t \geq p(z_t|z_{t-1}) \\ -\log p(z_t|z_{t-1}) & \text{o.w.} \end{cases}
\end{aligned} \tag{B.10}
$$

with

$$
\begin{aligned}
& E_{q(u_t)}\left[\log \mathbb{I}\left(u_t < p(z_t|z_{t-1})\right)\right] \qquad \text{(B.11)} \\
&= \int_0^{\tilde{\theta}_t} q(u_t) \log \mathbb{I}\left(u_t < p(z_t|z_{t-1})\right) du_t \\
&= \begin{cases} \int_0^{\tilde{\theta}_t} q(u_t) \log(1)\, du_t & \tilde{\theta}_t < p(z_t|z_{t-1}) \\ \int_0^{p(z_t|z_{t-1})} q(u_t) \log(1)\, du_t + \int_{p(z_t|z_{t-1})}^{\tilde{\theta}_t} q(u_t) \log(0)\, du_t & \tilde{\theta}_t \geq p(z_t|z_{t-1}) \end{cases} \\
&= \begin{cases} 0 & \tilde{\theta}_t < p(z_t|z_{t-1}) \\ -\infty & \tilde{\theta}_t \geq p(z_t|z_{t-1}) \end{cases}
\end{aligned}
$$

The expression for $q(z_{t-1})$ therefore becomes

$$
q(z_{t-1}) \propto \begin{cases} 0 & \tilde{\theta}_t \geq p(z_t|z_{t-1}) \\ p(z_{t-1}, x_{1:t-1}) & \text{o.w.} \end{cases} \tag{B.12}
$$

The form of $q(z_{t-1})$ shows us that a specific value of $\tilde{\theta}_t$ will act as a cutoff, forcing values of $q(z_{t-1})$ with associated $p(z_t|z_{t-1})$ to zero. This setting of $q(z_{t-1})$, in turn, adjusts $q(u_t)$ so that $\tilde{\theta}_t$ moves to the smallest value of $p(z_t|z_{t-1})$ greater than the initial $\tilde{\theta}_t$. After this second maximization step, no further changes occur in either variational distribution.

Appendix C: Variational Inference for the Joint SL-pHMM Models

C.1 Switching Model

Using the Switching Model, the joint probability of the set of sequences, $\mathbf{x}_{1:N}$, set of labels, $y_{1:N}$, switching variables, $\mathbf{s}_{1:N}$, and SL-pHMM hidden states, $t_{1:N,1:C}$ is given by the expression in Equation C.1. We also include priors over the emissions distributions, B, the switching transition distributions, $A^{(s)}$, and the Sigmoid Belief Network parameters, $\mathbf{v}$, but rather than attempt to approximate posterior distributions with respect to these variables, we compute maximum-a-posteriori solutions.

$$
\begin{aligned}
&p(y_{1:N}, \mathbf{x}_{1:N}, \mathbf{s}_{1:N}, t_{1:N,1:C}, B, A^{(s)}, \mathbf{v} | A, B_0, \alpha, \beta, \lambda_v) \\
&\quad = p(\mathbf{v}|\lambda_v) p(B|\beta) p(A^{(s)}|\alpha) \prod_n p(\mathbf{x}_n|\mathbf{s}_n, t_{1:C,n}, B, B_0) p(\mathbf{s}_n|A^{(s)}) \left(\prod_c p(t_{n,c}|A) \right) \\
&\qquad\qquad p(y_n|\mathbf{x}_n, e_{1:C,n}, \mathbf{v}^{(2)}) \prod_c p(e_{c,n}|\mathbf{x}_n, t_{c,n}, \mathbf{v}^{(1)}) \qquad \text{(C.1)}
\end{aligned}
$$

In the Switching model, the probability of the observed sequence given hidden variables is given as follows:

$$
\begin{aligned}
p(\mathbf{x}_n|\mathbf{s}_n, t_{n,1:C}, B, B_0) &= \prod_{t=1}^{T_n} B_{0,x_{n,t}}^{\mathbb{I}\left(t<t_{s_{n,t},n} \vee t \geq t_{s_{n,t},n}+K\right)} B_{s_{n,t},t_{s_{n,t},n}-t+1,x_{n,t}}^{\mathbb{I}\left(t_{s_{n,t},n} \leq t < t_{s_{n,t},n}+K\right)} \qquad \text{(C.2)} \\
&= \prod_{t=1}^{T_n} \left(\prod_s B_{0,x_{n,t}}^{\mathbb{I}\left(s=s_{n,t}\right)\mathbb{I}\left(t<t_{s,n} \vee t \geq t_{s,n}+K\right)} \right) \left(\prod_{s,k} B_{s,k,x_{n,t}}^{\mathbb{I}\left(s=s_{n,t}\right)\mathbb{I}\left(k=t_{s,n}-t+1\right)} \right) \qquad \text{(C.3)}
\end{aligned}
$$

The variational bound, excluding the prior distributions on $A^{(s)}$, B, and $\mathbf{v}$ is given by

Parameter	Definition
	General Notation
$\mathbf{x}_n$	the n^{th} observed sequence
y_n	the binary response variable associated with the n^{th} sequence, $y_n \in \{-1, 1\}$
$q(\dots)$	indicates a variational distribution over the parameters inside the parentheses
	SL-pHMM Variables
$t_{n,c}$	start of the relevant subsequence of the c^{th} SL-pHMM
$\mathbf{z}_{n,c}$	the set of hidden variables associated with the c^{th} pHMM used to generate the n^{th} observed sequence. $\mathbf{z}_{n,c}$ can be fully reconstructed given only $t_{n,c}$, the position of the first *Match* hidden state
A	fixed transition matrix for all SL-pHMMs
	Switching Model Variables
$\mathbf{s}_n$	the n^{th} sequence of switching variables
$A^{(s)}$	transition matrix on the switching variables $p(s'\|s) = A^{(s)}_{s,s'}$
B	an emissions matrix. $B_{c,k,m}$ is the probability of observed symbol m given that the symbol is generated from the c^{th} SL-pHMM's k^{th} relevant subsequence position
B_0	a fixed background distribution
α	Dirichlet prior parameter on $A^{(s)}$, $A^{(s)}_{s,:} \sim \text{Dirichlet}(\alpha)\ \forall s$
β	Dirichlet prior parameter on B, $B_{c,k,:} \sim \text{Dirichlet}(\beta)\ \forall c, k$
	Factorial Model Variables
$\mathbf{w}$	weights used to compute emission probabilities given the SL-pHMM hidden states
λ_w	regularization parameter on $\mathbf{w}$. i.e. $\mathbf{w} \sim \mathcal{N}\left(0, \lambda_w^{-1}\right)$
	Sigmoid Belief Network Variables
$e_{n,c}$	variables in the Sigmoid Belief Network's hidden layer, $e_{n,c} \in \{0, 1\}$
$\mathbf{v}$	Sigmoid Belief Network weights. $\mathbf{v}^{(1)}$ are the weights for the lower level of the network, with $\mathbf{v}^{(1)}_c$ and $v^{(1)}_{c,0}$ (bias term) associated with the c^{th} SL-pHMM. $\mathbf{v}^{(2)}$ and $v^{(2)}_0$ are associated with the top layer of the network.
λ_v	regularization parameter on $\mathbf{v}$. i.e. $\mathbf{v} \sim \mathcal{N}\left(0, \lambda_v^{-1}\right)$

Table C.1: Parameter definitions

$$
\begin{aligned}
& \log p(y_{1:N}, \mathbf{x}_{1:N}, \mathbf{v}|A, B_0, A^{(s)}, B, \lambda) \qquad \text{(C.4)} \\
= \; & \log \sum_{\mathbf{s},\mathbf{t}} p(y_{1:N}, \mathbf{x}_{1:N}, \mathbf{s}_{1:N}, t_{1:N,1:C}, \mathbf{v}|B, A^{(s)}, A, B_0, \lambda) \\
\geq \; & E_q \left[\log p(y_{1:N}, \mathbf{x}_{1:N}, \mathbf{s}_{1:N}, t_{1:N,1:C}, \mathbf{v}|B, A^{(s)}, A, B_0, \lambda)\right] + H(q) \\
= \; & \sum_n E_{q(\mathbf{s}_n)\prod_c q(t_{n,c})} \left[\log p(\mathbf{x}_n|\mathbf{s}_n, t_{n,1:C}, B, B_0)\right] + \\
& \sum_{n,c} E_{q(e_{n,c})q(t_{n,c})} \left[\log p(e_{n,c}|\mathbf{x}_n, t_{n,c}, \mathbf{v}_c^{(1)})\right] + \\
& \sum_n E_{\prod_c q(e_{n,c})} \left[\log p(y_n|e_{n,1:C}, \mathbf{v}^{(1)})\right] + \\
& \sum_n E_{q(\mathbf{s}_n)} \left[\log p(\mathbf{s}_n|A^{(s)})\right] + \\
& \sum_{n,c} E_{q(t_{n,c})} \left[\log p(t_{n,c}|A)\right] \\
& + H(q)
\end{aligned}
$$

where $H(q)$ indicates the entropy over all of the variational distributions.

C.1.1 Variational EM algorithm (Training Phase)

- Repeat until the variational bound converges:

  1. Maximize with respect to $q(\mathbf{s}_n)$ for all n - Equation C.6
  2. Maximize with respect to $q(t_{n,c})$ for all n, c - Equation C.8
  3. Maximize with respect to $q(e_{n,c})$ for all n, c - Equation C.37
  4. Maximize with respect to $A_c^{(s)}$ for all c - Equation C.9
  5. Maximize with respect to $B_{c,k}$ for all c, k - Equation C.10
  6. Maximize with respect to $\mathbf{v}^{(1)}$ using L-BFGS [80] - gradients given in Equation C.45 and C.46

7. Maximize with respect to the variational parameter,ζ_n, for all n

   (a) set $\zeta_n = y_n \left(v_0^{(2)} + \sum_c e_{n,c} v_c^{(2)} \right)$

8. Maximize with respect to $\mathbf{v}^{(2)}$ - Equation C.49

C.1.2 Variational EM algorithm (Prediction Phase)

- Repeat until the variational bound converges:

  1. Maximize with respect to $q(\mathbf{s}_n)$ for all n - Equation C.6
  2. Maximize with respect to $q(t_{n,c})$ for all n, c - Equation C.8
  3. Maximize with respect to $q(e_{n,c})$ for all n, c - Equation C.43

- Predict $y_n = \text{sign}\left(\left(\mathbf{v}^{(2)}\right)^\top \begin{bmatrix} \bar{e}_{n,1:C} \\ 1 \end{bmatrix} \right)$ (this choice of y_n corresponds to maximizing the variational bound with respect to y_n)

C.1.3 Compute the maximum with respect to $q(\mathbf{s}_n)$

$$\mathcal{F}\left(q(\mathbf{s}_n)\right) \tag{C.5}$$

$$= E_{q(\mathbf{s}_n)\prod_c q(t_{c,n})}\left[\log \frac{p(\mathbf{x}_n|\mathbf{s}_n, t_{n,1:C}, B, B_0)p(\mathbf{s}_n|A^{(s)})}{q(\mathbf{s}_n)}\right]$$

$$= E_{q(\mathbf{s}_n)}\Bigg[E_{\prod_c q(t_{c,n})}\Bigg[\log \prod_{t=1}^{T_n} A^{(s)}_{s_{n,t-1},s_{n,t}} \times$$

$$B_{0,x_{n,t}}^{\mathbb{I}\left(t<t_{s_{n,t},n}\vee t\geq t_{s_{n,t},n}+K\right)} \times$$

$$B_{s_{n,t},t_{s_{n,t},n}-t+1,x_{n,t}}^{\mathbb{I}\left(t_{s_{n,t},n}\leq t<t_{s_{n,t},n}+K\right)}\Bigg] - \log q(\mathbf{s}_n)\Bigg]$$

$$= E_{q(\mathbf{s}_n)}\Bigg[\log\Bigg(\prod_{t=1}^{T_n} A^{(s)}_{s_{n,t-1},s_{n,t}} \times$$

$$B_{0,x_{n,t}}^{E_{q(t_{s_{n,t},n})}\left[\mathbb{I}\left(t<t_{s_{n,t},n}\vee t\geq t_{c,n}+K\right)\right]} \times$$

$$\prod_k B_{s_{n,t},k,x_{n,t}}^{E_{q(t_{s_{n,t},n})}\left[\mathbb{I}\left(k=t_{s_{n,t},n}-t+1\right)\right]} - \log q(\mathbf{s}_n)\Bigg]$$

$$q(\mathbf{s}_n) \propto \prod_{t=1}^{T_n} A^{(s)}_{s_{n,t-1},s_{n,t}} \times \tag{C.6}$$

$$B_{0,x_{n,t}}^{E_{q(t_{s_{n,t},n})}\left[\mathbb{I}\left(t<t_{s_{n,t},n}\vee t\geq t_{c,n}+K\right)\right]} \times$$

$$\prod_k B_{s_{n,t},k,x_{n,t}}^{E_{q(t_{s_{n,t},n})}\left[\mathbb{I}\left(k=t_{s_{n,t},n}-t+1\right)\right]}$$

C.1.4 Compute the maximum with respect to $q(t_{n,c})$

$$\mathcal{F}\left(q(t_{c,n})\right) \tag{C.7}$$

$$= E_{q(\mathbf{s}_n)\prod_c q(t_{c,n})}\left[\log \frac{p(\mathbf{x}_n|\mathbf{s}_n, t_{n,1:C}, B, B_0)p(t_{c,n}|A)p(e_{n,c}|\mathbf{x}_n, t_{n,c}, \mathbf{v})}{q(t_{c,n})}\right]$$

$$= E_{q(t_{c,n})}\Bigg[E_{q(\mathbf{s}_n)\prod_{c'\neq c} q(t_{c,n})}\Bigg[\log\Bigg(\omega^{T_n-K-2}\left(1-\omega\right)^2 \times$$

$$\exp\left(E_{q(e_{n,c})}\left[-e_{n,c}\left(v^{(1)}_{c,0} + \sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right) - \log\left(1+\exp\left(-v^{(1)}_{c,0} - \sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right)\right]\right) \times$$

$$\prod_{t=1}^{T_n} B_{0,x_{n,t}}^{\mathbb{I}\left(t<t_{s_{n,t},n}\vee t\geq t_{s_{n,t},n}+K\right)} B_{s_{n,t},t_{s_{n,t},n}-t+1,x_{n,t}}^{\mathbb{I}\left(t_{s_{n,t},n}\leq t<t_{s_{n,t},n}+K\right)}\Bigg)\Bigg]$$

$$-\log q(t_{n,c})\Bigg]$$

$$= \text{const} +$$

$$E_{q(t_{c,n})}\Bigg[\log\Bigg(\left(-\bar{e}_{n,c}\left(v^{(1)}_{c,0} + \sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right) - \log\left(1+\exp\left(-v^{(1)}_{c,0} - \sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right)\right) \times$$

$$\prod_{t=1}^{T_n} B_{0,x_{n,t}}^{E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\mathbb{I}\left(t<t_{c,n}\vee t\geq t_{c,n}+K\right)+\sum_{c'\neq c}\mathbb{I}\left(c'=s_{n,t}\right)E_{q(t_{c',n})}\left[\mathbb{I}\left(t<t_{c',n}\vee t\geq t_{c',n}+K\right)\right]\right]} \times$$

$$B_{c,k,x_{n,t}}^{E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\mathbb{I}\left(k=t_{c,n}-t+1\right)+\sum_{c'\neq c}\mathbb{I}\left(c'=s_{n,t}\right)E_{q(t_{c',n})}\left[\mathbb{I}\left(k=t_{c',n}-t+1\right)\right]\right]}\Bigg) - \log q(t_{c,n})\Bigg]$$

$$= \text{const} +$$

$$E_{q(t_{c,n})}\Bigg[\log\Bigg(\exp\left(-\bar{e}_{n,c}^{\top}\left(v^{(1)}_{c,0} + \sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right) - \log\left(1+\exp\left(-v^{(1)}_{c,0} - \sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right)\Bigg) \times$$

$$\prod_{t=1}^{T_n} B_{0,x_{n,t}}^{E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\right]\mathbb{I}\left(t<t_{c,n}\vee t\geq t_{c,n}+K\right)} \times$$

$$B_{c,k,x_{n,t}}^{E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\right]\mathbb{I}\left(k=t_{c,n}-t+1\right)}\Bigg) - \log q(t_{c,n})\Bigg]$$

$$q(t_{n,c}) \propto \exp\left(-\bar{e}_{n,c}\left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\left(1+\exp\left(-v_{c,0}^{(1)} - \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)^{-1} \times \quad \text{(C.8)}$$

$$\prod_{t=1}^{T_n} B_{0,x_{n,t}}^{E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\right]\mathbb{I}\left(t<t_{c,n}\vee t\geq t_{c,n}+K\right)} \times$$

$$B_{c,k,x_{n,t}}^{E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\right]\mathbb{I}\left(k=t_{c,n}-t+1\right)}$$

C.1.4.1 Compute the maximum with respect to the switching transition distributions, $A_c^{(s)}$

The portion of the variational bound that depends on $A_c^{(s)}$ is given as follows:

$$\mathcal{F}\left(A_c^{(s)}\right) = \sum_n E_{q(\mathbf{s}_n)}\left[\log p(\mathbf{x}_n|\mathbf{s}_n, t_{n,1:C}, B, B_0) p(A^{(s)}|\alpha)\right]$$

$$s.t. \quad \sum_{c'} A_{c,c'}^{(a)} = 1$$

This gives the following Lagrangian:

$$\mathcal{L}\left(A_c^{(s)}\right) = \sum_{n,c'}\left(E_{q(\mathbf{s}_n)}\left[\mathbb{I}(s_{n,t-1}=c, s_{n,t}=c')\right] + \alpha - 1\right)\log\left(A_{c,c'}^{(s)}\right) - \lambda\left(\sum_{c'} A_{c,c'}^{(a)} - 1\right)$$

$$\frac{\partial \mathcal{L}\left(A_c^{(s)}\right)}{\partial A_{c,c'}^{(s)}} = \frac{\sum_n E_{q(\mathbf{s}_n)}\left[\mathbb{I}(s_{n,t-1}=c, s_{n,t}=c')\right] + \alpha - 1}{A_{c,c'}^{(s)}} - \lambda$$

First order optimality conditions give

$$A_{c,c'}^{(s)} \propto E_{q(\mathbf{s}_n)}\left[\mathbb{I}(s_{n,t-1}=c, s_{n,t}=c')\right] + \alpha - 1 \quad \text{(C.9)}$$

.

The expectation, $E_{q(\mathbf{s}_n)}\left[\mathbb{I}(s_{n,t-1}=c, s_{n,t}=c')\right]$, is computed using the Forward-Backward algorithm **[44]** in conjunction with the variational distribution, $q(\mathbf{s}_n)$.

C.1.4.2 Compute the maximum with respect to the emissions, $B_{c,k}$

The portion of the variational bound that depends on $B_{c,k}$ is given as follows:

$$\begin{aligned}\mathcal{F}\left(B_{c,k}\right) &= E_{q(\mathbf{s}_n)\prod_c q(t_{c,n})}\left[\log p(\mathbf{x}_n|\mathbf{s}_n, t_{n,1:C}, B, B_0)\right] + \log p(B_{c,k}|\beta) \\ s.t. &\quad \sum_m B_{c,k,m} = 1\end{aligned}$$

$$\begin{aligned}\mathcal{L}(B_{c,k}) &= \sum_n \sum_t \left(E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\right] E_{q(t_{c,n})}\left[\mathbb{I}\left(t_{c,n}-t+1=k\right)\right] \mathbb{I}\left(x_{n,t}=m\right) + \beta - 1\right) \log B_{c,k,m} \\ &\quad -\lambda\left(\left(\sum_m B_{c,k,m}\right) - 1\right)\end{aligned}$$

$$\frac{\partial \mathcal{L}(B_{c,k})}{\partial B_{c,k,m}} = \frac{\sum_n \sum_t E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\right] E_{q(t_{c,n})}\left[\mathbb{I}\left(t_{c,n}-t+1=k\right)\right] \mathbb{I}\left(x_{n,t}=m\right) + \beta - 1}{B_{c,k,m}} - \lambda$$

First order optimality conditions give

$$B_{c,k,m} \propto \sum_n \sum_t E_{q(\mathbf{s}_n)}\left[\mathbb{I}\left(c=s_{n,t}\right)\right] E_{q(t_{c,n})}\left[\mathbb{I}\left(t_{c,n}-t+1=k\right)\right] \mathbb{I}\left(x_{n,t}=m\right) + \beta - 1 \qquad \text{(C.10)}$$

The expectation, $E_{q(\mathbf{s}_n)}\left[\mathbb{I}(s_{n,t}=c)\right]$, is computed using the Forward-Backward algorithm **[44]** in conjunction with the variational distribution, $q(\mathbf{s}_n)$.

C.2 Factorial Model

Using the Factorial Model, the joint probability of the set of sequences, $\mathbf{x}_{1:N}$, set of labels, $y_{1:N}$, and SL-pHMM hidden states, $t_{1:N,1:C}$ is given by the expression in Equation C.11. We also include priors over the factorial weights, $\mathbf{w}$, and the Sigmoid Belief Network parameters, $\mathbf{v}$, but, as for the Switching Model, rather than attempt to approximate posterior distributions with respect to these variables, we compute maximum-a-posteriori solutions.

$$
\begin{aligned}
& p(y_{1:N}, \mathbf{x}_{1:N}, t_{1:C,1:N}, \mathbf{w}, \mathbf{v}|A, \lambda_w, \lambda_v) \\
& = p(\mathbf{w}|\lambda_w)p(\mathbf{v}|\lambda_v) \prod_n p(\mathbf{x}_n|t_{n,1:C}, \mathbf{w}) \left(\prod_c p(t_{n,c}|A) \right) p(y_n|x_n, e_{1:C,n}, \mathbf{v}^{(2)}) \prod_c p(e_{c,n}|x_n, t_{c,n}, \mathbf{v}^{(1)})
\end{aligned} \tag{C.11}
$$

For the Factorial Model, the following expression gives the probability of the observed sequence given the SL-pHMM variables, $t_{n,1:C}$, and the model weights, $\mathbf{w}$:

$$
\begin{aligned}
& p(\mathbf{x}_n|t_{n,1:C}, \mathbf{w}) \\
= & \prod_{t=1}^{T_n} \frac{\exp\left(\sum_c w_{c,0,x_{n,t}} \mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right) + \sum_{c,k} w_{c,k,x_{n,t}} \mathbb{I}\left(k = t_{n,c} - t + 1\right) \right)}{\sum_m \exp\left(\sum_c w_{c,0,m} \mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right) + \sum_{c,k} w_{c,k,m} \mathbb{I}\left(k = t_{n,c} - t + 1\right) \right)}
\end{aligned} \tag{C.12}
$$

The variational bound for the Factorial Model, excluding prior distributions over $\mathbf{w}$ and $\mathbf{v}$, is given as follows:

$$
\begin{aligned}
& \log p(y_{1:N}, \mathbf{x}_{1:N} | \mathbf{w}, \mathbf{v}, A, \lambda) && \text{(C.13)} \\
= \; & \log \sum_{\mathbf{e},\mathbf{t}} p(y_{1:N}, \mathbf{x}_{1:N}, e_{1:N,1:C}, t_{1:N,1:C} | \mathbf{w}, \mathbf{v}, A) \\
\geq \; & E_q \left[\log p(y_{1:N}, \mathbf{x}_{1:N}, e_{1:N,1:C}, t_{1:N,1:C} | \mathbf{w}, \mathbf{v}, A) \right] + H(q) \\
= \; & \sum_n E_{\prod_c q(t_{n,c})} \left[\log p(\mathbf{x}_n | t_{n,1:C}, \mathbf{w}) \right] + \\
& \sum_{n,c} E_{q(e_{n,c}) q(t_{n,c})} \left[\log p(e_{n,c} | \mathbf{x}_n, t_{n,c}, \mathbf{v}_c^{(1)}) \right] + \\
& \sum_n E_{\prod_c q(e_{n,c})} \left[\log p(y_n | e_{n,1:C}, \mathbf{v}^{(2)}) \right] + \\
& \sum_{n,c} E_{q(t_{n,c})} \left[\log p(t_{n,c} | A) \right] \\
& + H(q)
\end{aligned}
$$

C.2.1 Variational EM algorithm (Training)

- Repeat until the variational bound converges:

  1. Maximize with respect to the variational parameters $\alpha_{n,t}$ and $\xi_{n,t,m}$ for all n, c, m
     (a) Section C.2.3
  2. Maximize with respect to $q(t_{n,c})$ for all n, c
     (a) Equation C.20
  3. Maximize with respect to $q(e_{n,c})$ for all n, c
     (a) Equation C.41
  4. Maximize with respect to $\mathbf{w}$ using L-BFGS
     (a) gradients are given in Equations C.25, C.26, and (depending on parameter tying) C.27
  5. Maximize with respect to $\mathbf{v}^{(1)}$ using L-BFGS (depending on parameter tying)

(a) gradients given in Equations C.45 and C.46

6. Maximize with respect to the variational parameter,ζ_n, for all n

   (a) set $\zeta_n = y_n \left(v_0^{(2)} + \sum_c e_{n,c} v_c^{(2)} \right)$

7. Maximize with respect to $\mathbf{v}^{(2)}$

   (a) Equation C.49

C.2.2 Variational EM algorithm (Training)

- Repeat until the variational bound converges:

  1. Maximize with respect to the variational parameters $\alpha_{n,t}$ and $\xi_{n,t,m}$ for all n, c, m

     (a) Section C.2.3

  2. Maximize with respect to $q(t_{n,c})$ for all n, c

     (a) Equation C.20

  3. Maximize with respect to $q(e_{n,c})$ for all n, c

     (a) Equation C.43

- Predict $y_n = \text{sign}\left(\left(\mathbf{v}^{(2)}\right)^\top \begin{bmatrix} \bar{e}_{n,1:C} \\ 1 \end{bmatrix} \right)$ (this choice of y_n corresponds to maximizing the variational bound with respect to y_n)

C.2.3 Compute the maximum with respect to $q(t_{n,c})$

To compute the maximum with respect to $q(t_{n,c})$ for the Factorial model, we first write the expression for the variational bound including all of the terms that depend on $t_{n,c}$.

To clarify our presentation, we have also defined $z_{n,t,m}$ as follows for use in the new bound:

$$z_{n,t,m} \stackrel{\text{def}}{=} \sum_c w_{c,0,x_{n,t}} \mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right) + \sum_{c,k} w_{c,k,x_{n,t}} \mathbb{I}\left(k = t_{n,c} - t + 1\right) \tag{C.14}$$

$$\begin{aligned}
\mathcal{F}\left(q(t_{c,n})\right) &= E_{q(e_{n,c}) \prod_c q(t_{n,c})} \left[\log \frac{p(\mathbf{x}_n | t_{n,1:C}, \mathbf{w}) p(t_{n,c}|A) p(e_{n,c}|\mathbf{x}_n, t_{n,c}, \mathbf{v}^{(1)})}{q(t_{n,c})} \right] \\
&= E_{q(t_{c,n})} \Bigg[E_{\prod_{c' \neq c} q(t_{c,n})} \Bigg[\log \Bigg(\omega^{T_n - K - 2} \left(1 - \omega\right)^2 \times \\
&\quad \exp\left(E_{q(e_{n,c})} \left[- e_{n,c} \left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)} \right) \right. \right. \\
&\quad \left. \left. - \log\left(1 + \exp\left(-v_{c,0}^{(1)} - \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)} \right) \right) \right] \right) \times \\
&\quad \left. \prod_{t=1}^{T_n} \frac{\exp\left(z_{n,t,m}\right)}{\sum_m \exp\left(z_{n,t,m}\right)} \right) \Bigg] \\
&\quad \left. - \log q(t_{n,c}) \right]
\end{aligned} \tag{C.15}$$

$$
\begin{aligned}
= \quad & E_{q(t_{c,n})}\Bigg[-\bar{e}_{n,c}^{\top}\left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right) \\
& \quad -\log\left(1+\exp\left(-v_{c,0}^{(1)} - \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\Bigg) + \\
& \quad \sum_{t=1}^{T_n} w_{c,0,x_{n,t}} \mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c}+K\right) + \sum_k w_{c,k,x_{n,t}} \mathbb{I}\left(k = t_{n,c} - t + 1\right) + \\
& \quad \sum_{t=1}^{T_n} \sum_{c' \neq c} w_{c',0,x_{n,t}} E_{q(t_{n,c'})}\left[\mathbb{I}\left(t < t_{n,c'} \vee t \geq t_{n,c'}+K\right)\right] + \\
& \quad \sum_{c' \neq c,k} w_{c',k,x_{n,t}} E_{q(t_{n,c'})}\left[\mathbb{I}\left(k = t_{n,c'} - t + 1\right)\right] \\
& \quad -E_{\prod_{c' \neq c} q(t_{n,c'})}\left[\log \sum_m \exp\left(z_{n,t,m}\right)\right] \\
& \Bigg) - \log q(t_{c,n})\Bigg]
\end{aligned}
$$

To allow tractable computation of expectations over the softmax function in the Factorial Model, we apply the following bound [106]:

$$
\begin{aligned}
-\log \sum_k e^{x_k} \quad & \geq \quad -\alpha - \sum_k \log\left(1 + e^{-(\alpha - x_k)}\right) \qquad \text{(C.16)} \\
& \geq \quad -\alpha + \sum_k \left(-\log\left(1+e^{-\xi_k}\right) + \frac{\alpha - x_k - \xi_k}{2} - \tau\left(\xi_k\right)\left(\left(x_k - \alpha\right)^2 - \xi_k^2\right)\right)
\end{aligned}
$$

The bound requires that we include the variational parameters, $\alpha_{n,t}$ (for each sequence, n, and each element, t), and $\xi_{n,t,m}$ (for each sequence, n, element, t, and alphabet symbol, m). Here, we define $\tau(\xi) \stackrel{\text{def}}{=} \frac{1}{4\xi}\tanh\left(\frac{\xi}{2}\right)$.

To maximize with respect to the variational parameters of the bound, we first maximize with respect to each $\alpha_{n,c}$. Noting that Equation C.16 is concave in α, we set $\alpha = -\log K +$

$\log \sum e^{x_k}$, which is near the maximum, then run several iterations of Newton's method. Note that this first bound (line 1 of Equation C.16) is not tight in general. With respect to each $\xi_{n,t,m}$, the bound has a positive maximum at $\xi_k = \alpha - x_k$. Because the second bound (line 2 of Equation C.16) is tight, this two step procedure is valid, i.e., the maximum of the first bound with respect to α is the best possible lower bound with respect to both α and all ξ_k's.

$$
\begin{aligned}
& \mathcal{F}\left(q(t_{c,n})\right) && \text{(C.17)} \\
\geq\ & \tilde{\mathcal{F}}\left(q(t_{c,n})\right) \\
=\ & E_{q(t_{c,n})}\Bigg[- \bar{e}_{n,c}^{\top} \left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)} \right) - \log\left(1 + \exp\left(-v_{c,0}^{(1)} - \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\Bigg) + \\
& \sum_{t=1}^{T_n} w_{c,0,x_{n,t}} \mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right) + \sum_k w_{c,k,x_{n,t}} \mathbb{I}\left(k = t_{n,c} - t + 1\right) \\
& + \sum_{t=1}^{T_n} E_{\prod_{c' \neq c} q(t_{n,c'})}\Bigg[- \alpha_{n,t} + \sum_m \Bigg(\log \sigma\left(\xi_{n,t,m}\right) \\
& + \frac{\alpha_{n,t} - z_{n,t,m} - \xi_{n,t,m}}{2} - \tau\left(\xi_{n,t,m}\right)\left(\left(\alpha_{n,t} - z_{n,t,m}\right)^2 - \xi_{n,t,m}^2\right) \Bigg)\Bigg] && \text{(C.18)} \\
& \Bigg) - \log q(t_{c,n})\Bigg]
\end{aligned}
$$

$$\tilde{\mathcal{F}}\left(q(t_{c,n})\right) \tag{C.19}$$

$$\begin{aligned}
= \quad & E_{q(t_{c,n})}\Bigg[-\bar{e}_{n,c}^{\top}\left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right) - \log\left(1+\exp\left(-v_{c,0}^{(1)} - \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\Bigg) + \\
& \sum_{t=1}^{T_n}\left(w_{c,0,x_{n,t}}\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c}+K\right) + \sum_k w_{c,k,x_{n,t}}\mathbb{I}\left(k = t_{n,c}-t+1\right)\right) \\
& + \sum_{t=1}^{T_n}\sum_m\left(\left(\alpha_{n,t}2\tau\left(\xi_{n,t,m}\right) - \frac{1}{2}\right)\left(w_{c,0,m}\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c}+K\right) + \sum_k w_{c,k,m}\mathbb{I}\left(k = t_{n,c}-t+1\right)\right)\right) \\
& - \sum_{t=1}^{T_n}\sum_m \tau\left(\xi_{n,t,m}\right) E_{\prod_{c'\neq c} q(t_{n,c'})}\left[z_{n,t,m}^2\right] \\
& \Bigg) - \log q(t_{c,n})\Bigg]
\end{aligned}$$

Applying Gibbs' inequality to $\tilde{\mathcal{F}}\left(q(t_{n,c})\right)$ above gives a maximum at

$$q(t_{n,c}) \tag{C.20}$$

$$\begin{aligned}
\propto \quad & \exp\Bigg(-\bar{e}_{n,c}^{\top}\left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right) - \log\left(1+\exp\left(-v_{c,0}^{(1)} - \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\Bigg) + \\
& \sum_{t=1}^{T_n}\left(w_{c,0,x_{n,t}}\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c}+K\right) + \sum_k w_{c,k,x_{n,t}}\mathbb{I}\left(k = t_{n,c}-t+1\right)\right) \\
& + \sum_{t=1}^{T_n}\sum_m\left(\left(\alpha_{n,t}2\tau\left(\xi_{n,t,m}\right) - \frac{1}{2}\right)\left(w_{c,0,m}\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c}+K\right) + \sum_k w_{c,k,m}\mathbb{I}\left(k = t_{n,c}-t+1\right)\right)\right) \\
& - \sum_{t=1}^{T_n}\sum_m \tau\left(\xi_{n,t,m}\right) E_{\prod_{c'\neq c} q(t_{n,c'})}\left[z_{n,t,m}^2\right] \Bigg)
\end{aligned}$$

We provide expressions to compute the expectations over the $z_{n,t,m}$'s below. To save on computation, we compute the expectation excluding the c^{th} SL-pHMM, $E_{\prod_{c'\neq c} q(t_{n,c'})}\left[z_{n,t,m}\right]$, by adding and subtracting terms from the full variational expectation, $E_{\prod_{c'} q(t_{n,c'})}\left[z_{n,t,m}\right]$:

$$E_{\prod_{c'} q(t_{n,c'})}[z_{n,t,m}] \tag{C.21}$$

$$= \sum_{c'} w_{c',0,m} E_{q(t_{n,c'})}[\mathbb{I}(t < t_{n,c'} \vee t \geq t_{n,c'} + K)] + \sum_{c',k} w_{c',k,m} E_{q(t_{n,c'})}[\mathbb{I}(k = t_{n,c'} - t + 1)]$$

$$E_{\prod_{c' \neq c} q(t_{n,c'})}[z_{n,t,m}] \tag{C.22}$$

$$= E_{\prod_{c'} q(t_{n,c'})}[z_{n,t,m}]$$

$$- w_{c,0,m} E_{q(t_{n,c})}[\mathbb{I}(t < t_{n,c} \vee t \geq t_{n,c} + K)] - \sum_{k} w_{c,k,m} E_{q(t_{n,c})}[\mathbb{I}(k = t_{n,c} - t + 1)]$$

$$+ w_{c,0,m} \mathbb{I}(t < t_{n,c} \vee t \geq t_{n,c} + K) + \sum_{k} w_{c,k,m} \mathbb{I}(k = t_{n,c} - t + 1)$$

$$= \text{const} + w_{c,0,m} \mathbb{I}(t < t_{n,c} \vee t \geq t_{n,c} + K) + \sum_{k} w_{c,k,m} \mathbb{I}(k = t_{n,c} - t + 1)$$

Above, we have excluded terms that do not include $t_{n,c}$ because these do not contribute to the maximum with respect to $q(t_{n,c})$.

Below, we compute $E_{\prod_{c' \neq c} q(t_{n,c'})}\left[z_{n,t,m}^2\right]$ using $E_{\prod_{c'} q(t_{n,c'})}[z_{n,t,m}]$. Note that we have added terms $E_{q(t_{n,c})}[\mathbb{I}(\omega(t_{n,c}))]$ and substracted terms with $\left(E_{q(t_{n,c})}[\mathbb{I}(\omega(t_{n,c}))]\right)^2$ because $E_{q(t_{n,c})}\left[\mathbb{I}(\omega(t_{n,c}))^2\right] = E_{q(t_{n,c})}[\mathbb{I}(\omega(t_{n,c}))]$. Simply squaring $E_{\prod_{c'} q(t_{n,c'})}[z_{n,t,m}]$ would compute the expectation incorrectly.

$$E_{\prod_{c' \neq c} q(t_{n,c'})}\left[z_{n,t,m}^2\right] \tag{C.23}$$

$$= \left(\sum_{c'} w_{c',0,m} E_{q(t_{n,c'})}\left[\mathbb{I}\left(t < t_{n,c'} \vee t \geq t_{n,c'} + K\right)\right] + \sum_{c',k} w_{c',k,m} E_{q(t_{n,c'})}\left[\mathbb{I}\left(k = t_{n,c'} - t + 1\right)\right] \right.$$

$$- \left(w_{c,0,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right)\right]\right) - \sum_{k} \left(w_{c,k,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c} - t + 1\right)\right]\right)$$

$$\left. + w_{c,0,m}\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right) + \sum_{k} w_{c,k,m}\mathbb{I}\left(k = t_{n,c} - t + 1\right) \right)^2$$

$$- \sum_{c' \neq c} \left(w_{c',0,m} E_{q(t_{n,c'})}\left[\mathbb{I}\left(t < t_{n,c'} \vee t \geq t_{n,c'} + K\right)\right] \right)^2 - \sum_{c' \neq c,k} \left(w_{c',k,m} E_{q(t_{n,c'})}\left[\mathbb{I}\left(k = t_{n,c'} - t + 1\right)\right] \right)^2$$

$$+ \sum_{c' \neq c} w_{c',0,m}^2 E_{q(t_{n,c'})}\left[\mathbb{I}\left(t < t_{n,c'} \vee t \geq t_{n,c'} + K\right)\right] - \sum_{c' \neq c,k} w_{c',k,m}^2 E_{q(t_{n,c'})}\left[\mathbb{I}\left(k = t_{n,c'} - t + 1\right)\right]$$

$$= \text{const} +$$

$$2 \left(\sum_{c' \neq c} w_{c',0,m} E_{q(t_{n,c'})}\left[\mathbb{I}\left(t < t_{n,c'} \vee t \geq t_{n,c'} + K\right)\right] + \sum_{c' \neq c,k} w_{c',k,m} E_{q(t_{n,c'})}\left[\mathbb{I}\left(k = t_{n,c'} - t + 1\right)\right] \right) \times$$

$$\left(w_{c,0,m}\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right) + \sum_{k} w_{c,k,m}\mathbb{I}\left(k = t_{n,c} - t + 1\right) \right)$$

$$+ \left(w_{c,0,m}\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right) + \sum_{k} w_{c,k,m}\mathbb{I}\left(k = t_{n,c} - t + 1\right) \right)^2$$

C.2.4 Compute the maximum with respect to $\mathbf{w}_c$

We compute the the maximum with respect to $\mathbf{w}_c$ using L-BFGS. This method requires that we compute the gradients of the variational bound with respect to $\mathbf{w}$. We derive expressions for these gradients below. As before, we have defined $\tau(\xi) \stackrel{\text{def}}{=} \frac{1}{4\xi}\tanh\left(\frac{\xi}{2}\right)$ and $z_{n,t,m} \stackrel{\text{def}}{=} \sum_c w_{c,0,x_{n,t}}\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right) + \sum_{c,k} w_{c,k,x_{n,t}}\mathbb{I}\left(k = t_{n,c} - t + 1\right)$.

Below, we have also included terms from the lower level of the Sigmoid Belief Network (SBN) (see Section C.3) in the variational bound. We do so because in one of our these

lower-level SBN weights were fixed to the same values as the weights in the Factorial Model, i.e., $\mathbf{w}_c \stackrel{\text{def}}{=} \mathbf{v}_c^{(1)}$. The bias terms, $v_{c,0}^{(1)}$ and $w_{c,0,m}$, however, are always treated separately.

$$
\begin{aligned}
& \tilde{\mathcal{F}}(\mathbf{w}_c) \qquad\qquad (C.24)\\
= \; & -\frac{\lambda_w}{2}\mathbf{w}_c^\top \mathbf{w}_c \\
& - \sum_n \bar{e}_{n,c} E_{q(t_{n,c})}\left[v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right] - \sum_n E_{q(t_{n,c})}\left[\log\left(1+\exp\left(-v_{c,0}^{(1)} - \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\right] + \\
& \sum_{t=1}^{T_n} E_{\prod_c q(t_{n,c})}\left[z_{n,t,x_{n,t}}\right] + \\
& \sum_{t=1}^{T_n}\sum_m \left(2\alpha_{n,t}\tau\left(\xi_{n,t,m}\right) - \frac{1}{2}\right) E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}\right] - \sum_m \tau\left(\xi_{n,t,m}\right) E_{\prod_c q(t_{n,c'})}\left[z_{n,t,m}^2\right]
\end{aligned}
$$

In the gradient computation below, we treat $w_{c,k,m} \stackrel{\text{def}}{=} v_{c,k,m}^{(1)}$, but the $v_{c,k,m}^{(1)}$ term can be excluded in a model where the Factorial Model weights and SBN weights are independent.

$$
\begin{aligned}
& \frac{\partial \tilde{\mathcal{F}}(w_c)}{\partial w_{c,k,m}} \qquad\qquad (C.25)\\
= \; & -\sum_n \bar{e}_{n,c} E_{q(t_{n,c})}\left[\mathbb{I}\left(m = x_{n,t_{n,c}+k}\right)\right] \\
& + \sum_n E_{q(t_{n,c})}\left[\left(1+\exp\left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)^{-1} \mathbb{I}\left(m = x_{n,t_{n,c}+k}\right)\right] + \\
& \sum_{t=1}^{T_n} \frac{\partial E_{\prod_c q(t_{n,c})}\left[z_{x_{n,t}}\right]}{\partial w_{c,k,m}} \mathbb{I}\left(x_{n,t} = m\right) + \\
& \sum_{t=1}^{T_n} \left(2\alpha_{n,t}\tau\left(\xi_{n,t,m}\right) - \frac{1}{2}\right) \frac{\partial E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}\right]}{\partial w_{c,k,m}} - \tau\left(\xi_{n,t,m}\right) \frac{\partial E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}^2\right]}{\partial w_{c,k,m}} \\
& -\lambda_w w_{c,k,m}
\end{aligned}
$$

$$\frac{\partial \tilde{\mathcal{F}}(w_c)}{\partial w_{c,0,m}} = \sum_{t=1}^{T_n} \frac{\partial E_{\prod_c q(t_{n,c})}\left[z_{x_{n,t}}\right]}{\partial w_{c,0,m}} + \tag{C.26}$$

$$\sum_{t=1}^{T_n}\sum_{m}\left(2\alpha_{n,t}\tau\left(\xi_{n,t,m}\right)-\frac{1}{2}\right)\frac{\partial E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}\right]}{\partial w_{c,0,m}} - \sum_{m}\tau\left(\xi_{n,t,m}\right)\frac{\partial E_{\prod_c q(t_{n,c})}\left[z^2_{n,t,m}\right]}{\partial w_{c,0,m}}$$

$$-\lambda w_{c,0,m}$$

$$\frac{\partial \tilde{\mathcal{F}}(v^{(1)}_{c,0})}{\partial v^{(1)}_{c,0}} \tag{C.27}$$

$$= -\sum_n \bar{e}_{n,c} + \sum_n E_{q(t_{n,c})}\left[\left(1+\exp\left(v^{(1)}_{c,0}+\sum_k v^{(1)}_{c,k,x_{n,t_{n,c}+k}}\right)\right)^{-1}\right] - \lambda_v v^{(1)}_{c,0} \tag{C.28}$$

Computing the above gradients requires computing the gradients of expectations over $z_{n,t,m}$ and $z^2_{n,t,m}$, which we give below.

$$E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}\right] \tag{C.29}$$

$$= \sum_c w_{c,0,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c}+K\right)\right] + \sum_{c,k} w_{c,k,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c}-t+1\right)\right]$$

$$\frac{\partial E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}\right]}{\partial w_{c,k,m}} = E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c}-t+1\right)\right] \tag{C.30}$$

$$\frac{\partial E_{\prod_c q(t_{n,c})}\left[z_{n,tm}\right]}{\partial w_{c,0,m}} = E_{q(t_{n,c})}\left[\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c}+K\right)\right] \tag{C.31}$$

As mentioned in the previous section, when computing $E_{\prod_c q(t_{n,c})}\left[z^2_{n,t,m}\right]$, we subtract the squared of this expectation, $E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c}-t+1\right)\right]^2$ and add in terms $E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c}-t+1\right)\right]$. The complexity of this procedure is $O(CK)$, lower than the

alternative of summing over the expectation of each individual term, which has a complexity of $O(C^2K^2)$.

$$E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}^2\right] \tag{C.32}$$

$$\begin{aligned}
= & \left(\sum_c w_{c,0,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right)\right] + \sum_{c,k} w_{c,k,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c} - t + 1\right)\right]\right)^2 \\
& - \sum_c \left(w_{c,0,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right)\right]\right)^2 - \sum_{c,k} \left(w_{c,k,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c} - t + 1\right)\right]\right)^2 \\
& + \sum_c w_{c,0,m}^2 E_{q(t_{n,c})}\left[\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right)\right] + \sum_{c,k} w_{c,k,m}^2 E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c} - t + 1\right)\right]
\end{aligned}$$

$$\frac{\partial E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}^2\right]}{\partial w_{c,k,m}} \tag{C.33}$$

$$\begin{aligned}
= & 2E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}\right] E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c} - t + 1\right)\right] \\
& -2w_{c,k,m}\left(E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c} - t + 1\right)\right]\right)^2 \\
& +2w_{c,k,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c} - t + 1\right)\right] \\
= & 2E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c} - t + 1\right)\right]\left(E_{\prod_c q(t_{n,c})}\left[z_m\right] + w_{c,k,m}\left(1 - E_{q(t_{n,c})}\left[\mathbb{I}\left(k = t_{n,c} - t + 1\right)\right]\right)\right)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}^2\right]}{\partial w_{c,0,m}} = & 2E_{\prod_c q(t_{n,c})}\left[z_{n,t,m}\right] E_{q(t_{n,c})}\left[\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right)\right] \\
& -2w_{c,0,m}\left(E_{q(t_{n,c})}\left[\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right)\right]\right)^2 \\
& +2w_{c,0,m} E_{q(t_{n,c})}\left[\mathbb{I}\left(t < t_{n,c} \vee t \geq t_{n,c} + K\right)\right]
\end{aligned}$$

C.3 Sigmoid Belief Network

Both the Joint Factorial and Joint Switching SL-pHMMs use a variation of the Sigmoid Belief Network to define probabilties over sequence labels given relevant subsequences. Below we give expressions for the probability of individual layers of the network.

We define the sigmoid function as follows:

$$\sigma(z) \quad = \quad (1+\exp(-z))^{-1} \tag{C.34}$$

The probability of a hidden variable, $e_{n,c} \in \{0,1\}$, associated with the c^{th} relevant subsequence in the n^{th} sequence in the first layer of the network is

$$\begin{aligned} & p(e_{n,c}|\mathbf{x}_n, t_{c,n}, \mathbf{v}^{(1)}) \qquad (C.35) \\ = \quad & \sigma\left(\sum_{k,m} v^{(1)}_{c,k,m} \mathbb{I}\left(x_{n,t_{n,c}+k} = m\right)\right)^{1-e_{n,c}} \left(1 - \sigma\left(\sum_{k,m} v^{(1)}_{c,k,m} \mathbb{I}\left(x_{n,t_{n,c}+k} = m\right)\right)\right)^{e_{n,c}} \end{aligned}$$

The probability of the n^{th} sequence label given the SBN's hidden variables, $e_{n,1:C}$ is given as follows:

$$p(y_n|e_{n,1:C}, \mathbf{v}^{(2)}) = \sigma\left(y_n\left(v^{(2)}_0 + \sum_c v^{(2)}_c e_{n,c}\right)\right) \tag{C.36}$$

C.3.1 Compute the maximum with respect to $q(e_{n,c})$

Below, we use the fact that $e_{n,c} \in \{0,1\}$ implies $e^2_{n,c} = e_{n,c}$.

$$\begin{aligned} & \mathcal{F}(q(e_{n,c})) \qquad (C.37) \\ = \quad & E_{q(t_{c,n})\prod_{c'} q(e_{n,c'})}\left[\log \frac{p(y_n|e_{n,1:C}, \mathbf{v}^{(2)}) p(e_{n,c}|\mathbf{x}_n, t_{n,c}, \mathbf{v}^{(1)})}{q(e_{n,c})}\right] \\ = \quad & E_{\prod_{c'} q(e_{n,c'})}\left[\log \sigma(y_n e^\top_{n,1:C} \mathbf{v}^{(2)})\right] \\ & + E_{q(t_{n,c})q(e_{n,c})}\left[\log\left(p(e_{n,c}|\mathbf{x}_n, t_{n,c}, \mathbf{v}^{(1)})\right)\right] - \log q(e_{n,c}) \end{aligned}$$

Similar to the Factorial portion of the Joint SL-pHMM, we need an additional bound to allow the expectations over $\prod_c q(e_{n,c})$ to decompose. This bound **[38]** is similar to the

bound used in the Factorial Model over softmax.

$$\log \sigma(z) \quad \geq \quad \log \sigma(\zeta) + \frac{z-\zeta}{2} - \tau(\zeta)\left(z^2 - \zeta^2\right) \tag{C.38}$$

where we have defined $\tau(\zeta) \stackrel{\text{def}}{=} \frac{1}{4\zeta}\tanh\left(\frac{\zeta}{2}\right)$.

$$\begin{aligned}
& \mathcal{F}(q(e_{n,c})) && \text{(C.39)}\\
= \quad & \tilde{\mathcal{F}}(q(e_{n,c})) && \text{(C.40)}\\
\geq \quad & E_{\prod_{c'} q(e_{n,c'})}\Bigg[\log \sigma(\zeta_n) + \frac{1}{2} y_n \left(v_0^{(2)} + \sum_{c''} e_{n,c''} v_{c''}^{(2)}\right) - \frac{1}{2}\zeta_n \\
& -\tau(\zeta_n)\left(\left(y_n\left(v_0^{(2)} + \sum_{c''} e_{n,c''} v_{c''}^{(2)}\right)\right)^2 - \zeta_n^2\right) \\
& + E_{q(t_{n,c})}\left[\log\left(p(e_{n,c}|\mathbf{x}_n, t_{n,c}, w)\right)\right] - \log q(e_{n,c})\Bigg] \\
= \quad & E_{q(e_{n,c})}\Bigg[\frac{1}{2} y_n e_{n,c} v_c^{(2)} - \tau(\zeta_n)\left(\left(v_0^{(2)} + \sum_{c' \neq c} \bar{e}_{n,c'} v_{c'}^{(2)}\right) + e_{n,c} v_c^{(2)}\right)^2 \\
& - e_{n,c} E_{q(t_{n,c})}\left[v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right] - \log q(e_{n,c})\Bigg] \\
= \quad & E_{q(e_{n,c})}\Bigg[\frac{1}{2} y_n e_{n,c} v_c^{(2)} - e_{n,c}\tau(\zeta_n)\left(2 v_c^{(2)}\left(v_0^{(2)} + \sum_{c' \neq c} \bar{e}_{n,c'} v_{c'}^{(2)}\right) + \left(v_c^{(2)}\right)^2\right) \\
& - e_{n,c} E_{q(t_{n,c})}\left[v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right] - \log q(e_{n,c})\Bigg]
\end{aligned}$$

Applying Gibbs' inequality to $\tilde{\mathcal{F}}(q(e_{n,c}))$ above gives a maximum at

$$
\begin{aligned}
& q(e_{n,c}) && \text{(C.41)} \\
\propto\ & \exp\left(e_{n,c}\left(\frac{1}{2}y_n v_c^{(2)} - \tau\left(\zeta_n\right)\left(2v_c^{(2)}\left(v_0^{(2)} + \sum_{c'\neq c}\bar{e}_{n,c'}v_{c'}^{(2)}\right) + \left(v_c^{(2)}\right)^2\right)\right.\right. \\
& \left.\left. - \sum_{t_{n,c}=1}^{T_n-K} q(t_{n,c})\left(v_{c,0}^{(1)} + \sum_{k=1}^{K} v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\right)
\end{aligned}
$$

C.3.2 Compute the maximum with respect to $q(e_{n,c})$ on the test set

When performing inference on examples from the test set, the label information, y_n, is not available. Marginalizing over the possible labels (i.e., summing over $y_n \in \{-1, 1\}$ for each n) eliminates the top level of the SBN, which gives a different objective with respect to $q(e_{n,c})$:

$$
\mathcal{F}\left(q(e_{n,c})\right) \quad = \quad E_{q(t_{c,n})\prod_{c'} q(e_{n,c'})}\left[\log \frac{p(e_{n,c}|\mathbf{x}_n, t_{n,c}, \mathbf{v}^{(1)})}{q(e_{n,c})}\right] \tag{C.42}
$$

Applying Gibbs' inequality gives a maximum at

$$
q(e_{n,c}) \quad \propto \quad \exp\left(-e_{n,c}\sum_{t_{n,c}=1}^{T_n-K} q(t_{n,c})\left(v_{c,0}^{(1)} + \sum_{k=1}^{K} v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right) \tag{C.43}
$$

C.3.3 Maximize with respect to $\mathbf{v}^{(1)}$

We maximize with respect to $\mathbf{v}^{(1)}$ using L-BFGS. L-BFGS requires only the gradients of the variational bound, which we derive separately with respect to both the weights associated with each element of the subsequence, $\mathbf{v}_c^{\top}$, and the bias terms, $v_{c,0}^{\top}$.

$$
\begin{aligned}
& \mathcal{F}(\mathbf{v}_c) \qquad\qquad (C.44)\\
= & \sum_n E_{q(e_{n,c})q(t_{n,c})}\left[\log p(e_{n,c}|\mathbf{x}_n, t_{n,c}, \mathbf{v}_c^{(1)})\right]\\
= & -\sum_n \bar{e}_{n,c} E_{q(t_{n,c})}\left[v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right] - \sum_n E_{q(t_{n,c})}\left[\log\left(1+\exp\left(-v_{c,0}^{(1)} - \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\right]\\
& -\frac{\lambda}{2}\left(\mathbf{v}_c^{(1)}\right)^\top\left(\mathbf{v}_c^{(1)}\right)\\
= & \sum_n \sum_{t_{n,c}=1}^{T_n-K} q(t_{n,c})\left(-\bar{e}_{n,c}\left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\\
& + \sum_n \sum_{t_{n,c}=1}^{T_n-K} q(t_{n,c})\left(-\log\left(1+\exp\left(-v_{c,0}^{(1)} - \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\right)\\
& -\frac{\lambda}{2}\left(\mathbf{v}_c^{(1)}\right)^\top\left(\mathbf{v}_c^{(1)}\right)
\end{aligned}
$$

Below, in Equation C.45, we compute the gradient of the SBN variational bound with respect to $\mathbf{v}_c^{(1)}$, the lower level weights excluding the bias term.

$$
\begin{aligned}
\frac{\partial \mathcal{F}(v_c^{(1)})}{\partial v_{c,k,m}^{(1)}} = & -\sum_n \sum_{t_{n,c}=1}^{T_n-K} q(t_{n,c})\bar{e}_{n,c}\mathbb{I}\left(x_{n,t_{n,c}+k} = m\right) \qquad (C.45)\\
& + \sum_n \sum_{t_{n,c}=1}^{T_n-K} q(t_{n,c})\left(1-\sigma\left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)}\right)\right)\\
& -\lambda v_{c,k,m}^{(1)}
\end{aligned}
$$

Below, in Equation C.46, we compute the gradient of the SBN variational bound with respect to $v_{c,0}^{(1)}$, the bias term associated with the lower level weights.

$$\frac{\partial \mathcal{F}(v_{c,0}^{(1)})}{\partial v_{c,0}^{(1)}} \tag{C.46}$$

$$= \sum_n \left(\sum_{t_{n,c}=1}^{T_n - K} q(t_{n,c}) \left(1 - \sigma \left(v_{c,0}^{(1)} + \sum_k v_{c,k,x_{n,t_{n,c}+k}}^{(1)} \right) \right) \right) - \sum_n \bar{e}_{n,c} - \lambda v_{c,0}^{(1)}$$

C.3.4 Maximize with respect to $\mathbf{v}^{(2)}$

Given the quadratic variational bound on the logistic function, the optimization problem with respect to $\mathbf{v}^{(2)}$ becomes a linear least-squares problem. We have defined $\tau(\zeta) \stackrel{\text{def}}{=} \frac{1}{4\zeta} \tanh\left(\frac{\zeta}{2}\right)$ as above.

$$\begin{aligned}
\tilde{\mathcal{F}}(\mathbf{v}^{(2)}) &= \sum_n \frac{1}{2} y_n \left(v_0^{(2)} + \bar{e}_{n,1:C}^\top \mathbf{v}_{1:C}^{(2)} \right) \qquad \text{(C.47)} \\
&\quad - \sum_n \tau(\zeta_n) E_{\prod_c q(e_{n,c})} \left[\left(y_n \left(v_0^{(2)} + e_{n,1:C}^\top \mathbf{v}_{1:C}^{(2)} \right) \right)^2 \right] \\
&\quad - \frac{\lambda}{2} \left(\mathbf{v}^{(2)} \right)^\top \left(\mathbf{v}^{(2)} \right) \\
&= \sum_n \frac{1}{2} y_n \left(\mathbf{v}^{(2)} \right)^\top \begin{bmatrix} \bar{e}_{n,1:C} \\ 1 \end{bmatrix} \\
&\quad - \sum_n \tau(\zeta_n) \left(\mathbf{v}^{(2)} \right)^\top Q_n \left(\mathbf{v}^{(2)} \right) - \frac{\lambda}{2} \left(\mathbf{v}^{(2)} \right)^\top \left(\mathbf{v}^{(2)} \right) \qquad \text{(C.48)}
\end{aligned}$$

where we have defined $Q_n \stackrel{\text{def}}{=} E_{\prod_c q(e_{n,c})} \left[\begin{bmatrix} e_{n,1:C} \\ 1 \end{bmatrix} \begin{bmatrix} e_{n,1:C} \\ 1 \end{bmatrix}^\top \right]$ and $\mathbf{v}^{(2)} = \begin{bmatrix} \mathbf{v}_{1:C}^{(2)} \\ v_0^{(2)} \end{bmatrix}$.

We also note that the y_n^2 term in Equation C.47 always evaluates to 1 because $y_n \in \{-1, 1\}$. Given the quadratic bound, a closed form solution for $\mathbf{v}^{(2)}$ can be derived using the first-order optimality conditions:

$$\frac{\partial \mathcal{F}(\mathbf{v}^{(2)})}{\partial \mathbf{v}^{(2)}} = \sum_n \frac{1}{2} y_n \begin{bmatrix} \bar{e}_{n,1:C} \\ 1 \end{bmatrix} - \sum_n \tau(\zeta_n) Q_n \mathbf{v}^{(2)} - \lambda_v \mathbf{v}^{(2)}$$

$$\mathbf{v}^{(2)} = \left(\lambda_v I + \sum_n \tau(\zeta_n) Q_n \right)^{-1} \left(\sum_n \frac{1}{2} y_n \begin{bmatrix} \bar{e}_{n,1:C} \\ 1 \end{bmatrix} \right) \tag{C.49}$$

Bibliography

Bibliography

[1] H. Rangwala and G. Karypis, "Profile-based direct kernels for remote homology detection and fold recognition," *Bioinformatics*, vol. 21, no. 23, p. 4239, 2005.

[2] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ASSP Magazine, IEEE*, vol. 3, no. 1, pp. 4–16, 1986.

[3] T. Jaakkola, M. Diekhans, and D. Haussler, "Using the fisher kernel method to detect remote protein homologies," in *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, 1999, pp. 149–158.

[4] S. J. Blasiak and H. Rangwala, "A hidden markov model variant for sequence classification," in *Proceedings of the twenty-second International Joint Conference on Artificial Intelligence*, 2011.

[5] S. J. Blasiak, H. Rangwala, and K. B. Laskey, "Beam methods for the profile hidden markov model," in *SDM*, 2012, pp. 331–342.

[6] ——, "Relevant subsequence detection with sparse dictionary learning," in *European Conference on Machine Learning*, 2013.

[7] S. Blasiak, H. Rangwala, and K. B. Laskey, "A family of feed-forward models for protein sequence classification," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 419–434.

[8] F. Jelinek, *Statistical methods for speech recognition*. the MIT Press, 1997.

[9] S. R. Eddy, "Profile hidden markov models." *Bioinformatics*, vol. 14, no. 9, p. 755, 1998.

[10] A. Smola and B. Schölkopf, *Learning with kernels*. Citeseer, 1998.

[11] C. Leslie, E. Eskin, and W. Noble, "The spectrum kernel: a string kernel for svm protein classification." in *Pacific symposium on biocomputing*, vol. 575. Hawaii, USA., 2002, pp. 564–575.

[12] C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble, "Mismatch string kernels for discriminative protein classification," *Bioinformatics*, vol. 20, no. 4, p. 467, 2004.

[13] P. Kuksa, P. Huang, and V. Pavlovic, "Fast protein homology and fold detection with sparse spatial sample kernels," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. IEEE, 2008, pp. 1–4.

[14] H. Saigo, J. P. Vert, N. Ueda, and T. Akutsu, "Protein homology detection using string alignment kernels," *Bioinformatics*, vol. 20, no. 11, pp. 1682–1689, 2004.

[15] R. Durbin, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge Univ Pr, 1998.

[16] H. Rangwala and G. Karypis, "Profile-based direct kernels for remote homology detection and fold recognition," *Bioinformatics*, vol. 21, no. 23, p. 4239, 2005.

[17] R. Kuang, E. Ie, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie, "Profile-based string kernels for remote homology detection and motif extraction," in *Computational Systems Bioinformatics Conference, 2004. CSB 2004. Proceedings. 2004 IEEE*, 2004, pp. 152–160.

[18] J. Weston, C. Leslie, E. Ie, D. Zhou, A. Elisseeff, and W. Noble, "Semi-supervised protein classification using cluster kernels," *Bioinformatics*, vol. 21, no. 15, pp. 3241–3247, 2005.

[19] G. Ifrim and C. Wiuf, "Bounded coordinate-descent for biological sequence classification in high dimensional predictor space," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 708–716.

[20] C. Ding and I. Dubchak, "Multi-class protein fold recognition using support vector machines and neural networks," *Bioinformatics*, vol. 17, no. 4, pp. 349–358, 2001.

[21] A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler, "Hidden markov models in computational biology: Applications to protein modeling," *Journal of molecular biology*, vol. 235, no. 5, pp. 1501–1531, 1994.

[22] S. Eddy *et al.*, "A new generation of homology search tools based on probabilistic inference," in *Genome Inform*, vol. 23, no. 1, 2009, pp. 205–211.

[23] E. Keogh, X. Xi, L. Wei, and C. A. Ratanamahatana, "The ucr time series classification/clustering homepage," 2011.

[24] C. M. Bishop *et al.*, *Pattern recognition and machine learning*. springer New York, 2006, vol. 1.

[25] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[26] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 26, no. 1, pp. 43–49, 1978.

[27] E. Keogh, "Exact indexing of dynamic time warping," in *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 406–417.

[28] D. Lemire, "Faster retrieval with a two-pass dynamic-time-warping lower bound," *Pattern recognition*, vol. 42, no. 9, pp. 2169–2180, 2009.

[29] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1-2, pp. 1–305, 2008.

[30] M. Opper, *Advanced mean field methods: Theory and practice*. MIT press, 2001.

[31] M. Beal and U. of London, *Variational algorithms for approximate Bayesian inference*. Citeseer, 2003.

[32] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, *An introduction to variational methods for graphical models*. Springer, 1998.

[33] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[34] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38, 1977.

[35] R. M. Neal and G. E. Hinton, "A view of the em algorithm that justifies incremental, sparse, and other variants," in *Learning in graphical models*. Springer, 1998, pp. 355–368.

[36] J. Winn and C. M. Bishop, "Variational message passing," *Journal of Machine Learning Research*, vol. 6, no. 1, p. 661, 2006.

[37] Y. W. Teh, D. Newman, and M. Welling, "A collapsed variational bayesian inference algorithm for latent dirichlet allocation," *Advances in neural information processing systems*, vol. 19, p. 1353, 2007.

[38] T. S. Jaakkola and M. I. Jordan, "Bayesian parameter estimation via variational methods," *Statistics and Computing*, vol. 10, no. 1, pp. 25–37, 2000.

[39] L. Bottou, "Online algorithms and stochastic approximations," *Online Learning and Neural Networks*, 1998. [Online]. Available: http://leon.bottou.org/papers/bottou-98x

[40] L. Bottou and O. Bousquet, "The tradeoffs of large-scale learning," *Optimization for Machine Learning*, p. 351, 2011.

[41] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, Y. Lechevallier and G. Saporta, Eds. Paris, France: Springer, August 2010, pp. 177–187. [Online]. Available: http://leon.bottou.org/papers/bottou-2010

[42] A. Murzin, S. Brenner, T. Hubbard, and C. Chothia, "SCOP: a structural classification of proteins database for the investigation of sequences and structures," *Journal of molecular biology*, vol. 247, no. 4, pp. 536–540, 1995.

[43] C. Leslie, E. Eskin, and W. S. Noble, "The spectrum kernel: A string kernel for svm protein classification," *Proceedings of the Pacific Symposium on Biocomputing*, pp. 564–575, 2002.

[44] L. Rabiner and B. Juang, "An introduction to hidden Markov models," *IEEE ASSp Magazine*, vol. 3, no. 1 Part 1, pp. 4–16, 1986.

[45] S. Eddy, "Profile hidden markov models," *Bioinformatics*, vol. 14, no. 9, pp. 755–763, 1998.

[46] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[47] H. Rangwala and G. Karypis, "Building multiclass classifiers for remote homology detection and fold recognition." *BMC Bioinformatics*, vol. 7, p. 455, 2006. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-7-455

[48] P. Smyth, "Clustering sequences with hidden Markov models," *Advances in neural information processing systems*, pp. 648–654, 1997.

[49] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1999, pp. 50–57.

[50] A. Gruber, M. Rosen-Zvi, and Y. Weiss, "Hidden topic Markov models," *Artificial Intelligence and Statistics (AISTATS)*, 2007.

[51] J. Zhu and E. Xing, "Conditional Topic Random Fields," in *International Conference on Machine Learning (To appear)*. Citeseer, 2010.

[52] S. Scott, "Bayesian methods for hidden Markov models: Recursive computing in the 21st century," *Journal of the American Statistical Association*, vol. 97, no. 457, pp. 337–351, 2002.

[53] Z. Ghahramani and M. Jordan, "Factorial hidden Markov models," *Machine learning*, vol. 29, no. 2, pp. 245–273, 1997.

[54] Z. Ghahramani and G. Hinton, "Variational learning for switching state-space models," *Neural Computation*, vol. 12, no. 4, pp. 831–864, 2000.

[55] D. MacKay, "Ensemble learning for hidden Markov models," 1997.

[56] T. Joachims, "SVMLight: Support Vector Machine," *SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund*, 1999.

[57] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *The Journal of Machine Learning Research*, vol. 2, pp. 265–292, 2002.

[58] R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie, "Profile-based string kernels for remote homology detection and motif extraction," *Computational Systems Bioinformatics*, pp. 152–160, 2004.

[59] S. Altschul, T. Madden, A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped blast and psi-blast: a new generation of protein database search programs," *Nucleic acids research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[60] P. Moreno, P. Ho, and N. Vasconcelos, "A Kullback-Leibler divergence based kernel for SVM classification in multimedia applications," *Advances in Neural Information Processing Systems*, vol. 16, pp. 1385–1392, 2004.

[61] T. Jaakkola, M. Diekhans, and D. Haussler, "A discriminative framework for detecting remote protein homologies," *Journal of Computational Biology*, vol. 7, no. 1-2, pp. 95–114, 2000.

[62] C. Leslie, E. Eskin, W. S. Noble, and J. Weston, "Mismatch string kernels for svm protein classification," *Advances in Neural Information Processing Systems*, vol. 20, no. 4, pp. 467–476, 2003.

[63] S. R. Eddy, "Multiple alignment using hidden markov models," in *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, vol. 3, 1995, pp. 114–120.

[64] C. Pal, C. Sutton, and A. McCallum, "Sparse forward-backward using minimum divergence beams for fast training of conditional random fields," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 5, 2006, pp. V–V.

[65] J. Van Gael, Y. Saatci, Y. W. Teh, and Z. Ghahramani, "Beam sampling for the infinite hidden markov model," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1088–1095.

[66] D. J. MacKay, *Ensemble learning for hidden Markov models.* Technical report, Cavendish Laboratory, University of Cambridge, 1997.

[67] M. J. Beal, Z. Ghahramani, and C. E. Rasmussen, "The infinite hidden markov model," *Advances in Neural Information Processing Systems*, vol. 1, pp. 577–584, 2002.

[68] J. Paisley and L. Carin, "Hidden markov models with stick-breaking priors," *Signal Processing, IEEE Transactions on*, vol. 57, no. 10, pp. 3905–3917, 2009.

[69] T. S. Ferguson, "A bayesian analysis of some nonparametric problems," *The annals of statistics*, vol. 1, no. 2, pp. 209–230, 1973.

[70] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Hierarchical dirichlet processes," *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1566–1581, 2006.

[71] J. Sethuraman, "A constructive definition of dirichlet priors," FLORIDA STATE UNIV TALLAHASSEE DEPT OF STATISTICS, Tech. Rep., 1991.

[72] B. A. Frigyik, A. Kapila, and M. R. Gupta, "Introduction to the dirichlet distribution and related processes," Tech. Rep. 206, 2010.

[73] H. Ishwaran and L. F. James, "Gibbs sampling methods for stick-breaking priors," *Journal of the American Statistical Association*, vol. 96, no. 453, pp. 161–173, 2001.

[74] M. J. Beal and M. MA, "Variational algorithms for approximate bayesian inference," *Unpublished doctoral dissertation, University College London*, 2003.

[75] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia, "SCOP: a structural classification of proteins database for the investigation of sequences and structures," *Journal of molecular biology*, vol. 247, no. 4, pp. 536–540, 1995.

[76] R. C. Edgar, "MUSCLE: multiple sequence alignment with high accuracy and high throughput," *Nucleic acids research*, vol. 32, no. 5, p. 1792, 2004.

[77] R. M. Neal, "Connectionist learning of belief networks," *Artificial intelligence*, vol. 56, no. 1, pp. 71–113, 1992.

[78] L. K. Saul, T. Jaakkola, and M. I. Jordan, "Mean field theory for sigmoid belief networks," *arXiv preprint cs/9603102*, 1996.

[79] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[80] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[81] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Discriminative learned dictionaries for local image analysis," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[82] J. Mairal, M. Leordeanu, F. Bach, M. Hebert, and J. Ponce, "Discriminative sparse image models for class-specific edge detection and image interpretation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2008.

[83] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *The Journal of Machine Learning Research*, vol. 11, pp. 19–60, 2010.

[84] J. Yang, K. Yu, Y. Gong, and T. Huang, "Linear spatial pyramid matching using sparse coding for image classification," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1794–1801.

[85] H. Lee, A. Battle, R. Raina, and A. Y. Ng, "Efficient sparse coding algorithms," *Advances in neural information processing systems*, vol. 19, p. 801, 2007.

[86] Y. Boureau, F. Bach, Y. LeCun, and J. Ponce, "Learning mid-level features for recognition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2559–2566.

[87] M. Aharon, M. Elad, and A. Bruckstein, "K-svd: Design of dictionaries for sparse representation," *Signal Processing, IEEE Transactions on*, vol. 54, no. 11, pp. 4311–4322, 2006.

[88] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *The Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[89] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, "Pathwise coordinate optimization," *The Annals of Applied Statistics*, vol. 1, no. 2, pp. 302–332, 2007.

[90] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[91] B. Mailhé, S. Lesage, R. Gribonval, F. Bimbot, P. Vandergheynst *et al.*, "Shift-invariant dictionary learning for sparse representations: extending k-svd," in *16th EUropean SIgnal Processing COnference (EUSIPCO'08)*, 2008.

[92] T. L. Bailey and C. Elkan, "Fitting a mixture model by expectation maximization to discover motifs in biopolymers," *vectors*, vol. 1, p. 2.

[93] J. Buhler and M. Tompa, "Finding motifs using random projections," *Journal of computational biology*, vol. 9, no. 2, pp. 225–242, 2002.

[94] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Arxiv preprint arXiv:1103.0398*, 2011.

[95] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural computation*, vol. 10, no. 7, pp. 1895–1923, 1998.

[96] L. Liao and W. Noble, "Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships," *Journal of computational biology*, vol. 10, no. 6, pp. 857–868, 2003.

[97] S. Blasiak and H. Rangwala, "A hidden markov model variant for sequence classification," in *International Joint Conference on Artificial Intelligence*, 2011.

[98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[99] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 3, pp. 328–339, 1989.

[100] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis*. Cambridge university press Cambridge, UK:, 2002.

[101] B. Schölkopf, S. Mika, C. Burges, P. Knirsch, K. Müller, G. Rätsch, and A. Smola, "Input space versus feature space in kernel-based methods," *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 1000–1017, 1999.

[102] D. Rumelhart, G. Hintont, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[103] J. Qiu, M. Hue, A. Ben-Hur, J. Vert, and W. Noble, "A structural alignment kernel for protein structures," *Bioinformatics*, vol. 23, no. 9, pp. 1090–1098, 2007.

[104] N. M. Daniels, R. Hosur, B. Berger, and L. J. Cowen, "Smurflite: combining simplified markov random fields with simulated evolution improves remote homology detection for beta-structural proteins into the twilight zone," *Bioinformatics*, vol. 28, no. 9, pp. 1216–1222, 2012.

[105] Y. Liu, J. Carbonell, P. Weigele, and V. Gopalakrishnan, "Protein fold recognition using segmentation conditional random fields (scrfs)," *Journal of Computational Biology*, vol. 13, no. 2, pp. 394–406, 2006.

[106] G. Bouchard, "Efficient bounds for the softmax function, applications to inference in hybrid models," 2007.

Curriculum Vitae

Sam Blasiak earned a B.A. in English Literature from Colorado College in 2004 and an M.A. in Computer Science from Brandeis University in 2006. Before and after these studies, Sam was a paratrooper in the U.S. Army's 82nd Airborne Division and was deployed with the North Dakota National Guard in Afghanistan. Following his deployment, in the Fall of 2009, Sam joined George Mason University's Computer Science PhD program.