My Thesis Title

Diogo Pernes

Doutoramento em Ciência de Computadores

Departamento de Ciência de Computadores

2021

Orientador

Jaime S. Cardoso, Professor Catedrático, Faculdade de Engenharia

Todas as correções determinadas

pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/________

UNIVERSIDADE DO PORTO

DOCTORAL THESIS

Learning from multi-entity data

Author:
Diogo PERNES

Supervisor:
Jaime S. CARDOSO

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy

at the

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

March 1, 2021

" I am and always will be the optimist, the hoper of far-flung hopes and the dreamer of improbable dreams "

Matt Smith as *The Doctor*, written by Matthew Graham

Acknowledgements

Acknowledge ALL the people!

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto

Departamento de Ciência de Computadores

Doctor of Philosophy

Learning from multi-entity data

by Diogo PERNES

This thesis is about something, I guess.

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto

Departamento de Ciência de Computadores

Doutoramento em Ciência de Computadores

Titulo da Tese em Português

por Diogo PERNES

Este tese é sobre alguma coisa

Contents

List of Figures

Notation and Conventions

In this section, we describe the notation adopted in this thesis. We mostly follow the notation proposed by Goodfellow et al. [1], which is also the recommended one by the International Conference on Learning Representations (ICLR).

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathsf{\mathbf{A}}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0,\dots,0,1,0,\dots,0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{1, \dots, n\}$	The set of all integers between 1 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$A_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathbf{\mathsf{A}}$
$\mathbf{\mathsf{A}}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Linear Algebra Operations

$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^+$	Moore-Penrose pseudoinverse of $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$
$\det(\boldsymbol{A})$	Determinant of $\boldsymbol{A}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathsf{\mathbf{X}}} y$	Tensor containing derivatives of y with respect to $\mathsf{\mathbf{X}}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla_{\boldsymbol{x}}^2 f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$\mathrm{a} \perp \mathrm{b}$	The random variables a and b are independent
$\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$	They are conditionally independent given c
$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \| Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \rightarrow \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise

Sometimes we use a function f whose argument is a scalar but apply it to a vector, matrix, or tensor: $f(\boldsymbol{x})$, $f(\boldsymbol{X})$, or $f(\mathsf{X})$. This denotes the application of f to the array element-wise. For example, if $\mathsf{C} = \sigma(\mathsf{X})$, then $C_{i,j,k} = \sigma(X_{i,j,k})$ for all valid values of i, j and k.

Datasets and Distributions

p_{data}	The data generating distribution
$\hat{p}_{\text{data}}$	The empirical distribution defined by the training set
$\mathbb{X}$	A set of training examples
$\boldsymbol{x}^{(i)}$	The i-th example (input) from a dataset
$y^{(i)}$ or $\boldsymbol{y}^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\boldsymbol{X}$	The $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$

Chapter 1

Background

In this chapter, we provide a brief overview of the main methods we are going to use.

Chapter 2

Networked data streams

The content presented in this Chapter was partially published in or adapted from [2, 3].

2.1 Motivation

A broad range of real-life settings can be well modeled by an arbitrary number of network connected entities that share and interact in the same medium and generate data streams in real-time. The streams produced by each of these entities form a set of time series with both intra- and inter-correlations between them. In neuroimaging studies, the brain can be regarded as a network: a connected system where nodes, or units, represent different specialized regions and links, or connections, represent communication pathways. From a functional perspective, communication is coded by temporal dependence between the activities of different brain areas (De Vico Fallani et al. [4]). Also team sports intrinsically involve fast, complex and interdependent events among a set of entities (the players), which interact as a team (Tora et al. [5], Theagarajan et al. [6]). The emergence of vehicular networks is also generating an ever-increasing amount of network data (Cheng et al. [7]), where interactions between neighboring vehicles may be exploited to build more accurate and reliable learning algorithms. Thus, in all these scenarios the behavior of each individual entity is better understood if its context information (i.e. the behavior of the neighboring instances) is leveraged. However, the extraction of knowledge from these streams to support the decision-making process is still challenging. Moreover, conventional algorithms that assume ability to store and centralize all the data in memory at the same time are impractical for many applications (Gama and Gaber [8]).

Modeling the generative process of distributed stream data is an unsupervised learning problem and, hence, a model can be learned directly from the large amounts of data that might be continuously produced or gathered at each network connected entity, without the requirement of any special human supervision or annotation. Moreover, generative models are powerful tools for a wide variety of problems that arise naturally in networked data streams, like anomaly and novelty detection, sequence forecasting, clustering, and network simulation. Hence, generative models for stream data have been developed and applied in several previous works (e.g. Laxman et al. [9], Hayat and Hashemi [10], Hofmann and Sick [11]). However, to the best of our knowledge, this problem is seldom explored in the distributed setting.

Given the distributed nature of network data and the high information rate that those streams could have, we argue that such generative model and/or the associated learning algorithm should ideally satisfy the following properties:

1. Learning and inferring distributedly, i.e. each entity should be able to update its own model and perform inference on it without observing the streams or the models associated with the remaining entities.

2. Learning online, i.e. in real-time and without the requirement of storing the whole dataset in memory.

3. Leveraging contextual information by incorporating prior knowledge about similarities and dissimilarities among sets of entities.

4. Universality, i.e. the model should be applicable to a wide variety of distributed stream data, of diverse nature.

In this chapter, we present two solutions for this problem, each with its own advantages and drawbacks. Both are inspired on the concept of sparse representations, which expresses a signal/model f, defined over some independent variable x, as a linear combination of a few atoms from a prespecified and overcomplete dictionary of size M:

$$f(x) = \sum_{m=1}^{M} s_m \phi_m(x), \tag{2.1}$$

where $\phi_m(x)$ are the atoms and only a few of the scalars s_m are non-zero, providing a sparse representation of $f(x)$. Distributed sparse representation (Baron et al. [12]) is an extension of the standard version that considers networks with K nodes. At each node, the signal sensed at the same node has its sparsity property because of its intracorrelation, while, for networks with multiple nodes, signals received at different nodes also exhibit strong intercorrelation. The intra and inter-correlations lead to a joint sparse model. An interesting scenario in distributed sparse representation, which we exploit here, is when all signals/models share the common support but with different non-zero coefficients.

Our contributions in this chapter are summarized as follows: i) we extend an existing model based on a combination of self-organizing maps (SOM) and HMMs and evaluate it in the context of anomaly detection in Wi-Fi networks; ii) we present a novel algorithm that learns an entity-dependent model based on a mixture of shared HMMs; iii) we discuss how the two models intersect each other and, importantly, how the latter can be viewed as a particular case of a general family of generative models for distributed data.

2.2 Hidden Markov Models on a Self-Organizing Map for Anomaly Detection in 802.11 Wireless Networks

2.2.1 Introduction

In large-scale 802.11 wireless networks, acquiring a baseline knowledge of the entire infrastructure is not straightforward. Owing to the time-varying and physically distributed nature of these networks, learning the usage characteristics of access points (APs), users, and locations becomes more and more challenging. The wireless-channel conditions evolve over time, as does the usage behavior of the wireless users in different parts of the network. Thus, the wireless users are likely to suffer from many types of connectivity and performance problems, e.g., interference, intermittent connections, or authentication failures. To constantly ensure that wireless users have reliable connections, network managers require tools and techniques to monitor the network, identify the problems, and resolve them efficiently.

Naive approaches, e.g., using a single HMM common to all APs, lose the flexibility to adapt to the specificities of each AP, while using one HMM per AP, trained independently from the others, fails to leverage the relations between observations of neighboring APs. HMM initialized with universal background model (HMM-UBM) [13] is an improvement in the right direction; however, the relations between APs are only used in the initial phase, where one trains the UBM to then initialize the individual HMM models for each AP. Thereafter, the individual HMMs evolve independently, only benefiting from the

AP's own data. Although this is a very sensible approach in the biometrics and speech-modeling fields, where HMM-UBM has been used to robustly train user-specific models, in our setting, it fails to properly explore the dependencies between data from APs with similar behavior.

In the current work, we focus on the actual proximity of APs as a determining factor in connectivity and performance problems. Anomalous cases, e.g., across-AP-vicinity interference, AP overloads, or AP shutdown/halt could eventually affect the usage behavior of the other APs in the neighborhood. To consider such behavioral changes in the local area and their respective influences, we employ the synergistic approach of self-organizing hidden Markov model map (SOHMMM) to exploit the semantic connectivity between adjacent HMMs.

The self-organizing map is an artificial neural network that defines a nonlinear transformation from the input space to the set of nodes in the output space ([14]). Each node or neuron in the SOM is associated with a model of the input space. Through an unsupervised-learning process, the models are tuned and organized in a lattice topology according to the input patterns. In SOHMMM, each neuron is literally associated with an HMM.

The training processes of the SOM and HMM sub-units are, in most cases, disjoint and conducted independently. There are two main approaches regarding these hybrid techniques. The first approach considers the SOM as a front-end processor (e.g., vector quantization, preprocessing, feature extraction); HMMs are then used in the higher processing stages [14–16]. The second approach places the SOM on top of the HMM [17–19].

In SOHMMM, the SOM unsupervised-learning approach is well combined with the HMM dynamic-programming technique. The structure of both corresponding components is unified in an integrated super-model. The presented online gradient-descent unsupervised-learning algorithm is inspired by the SOHMMM algorithm previously proposed in [17] and originated in [20]. We extend the model to fit the requirements of our anomaly-detection problem and improve the presented algorithm in [17] for multivariate Gaussian emissions[*].

[*]In [17], only the discrete-observation setting is addressed.

2.2.2 Related work

A number of studies in the literature have integrated the SOM and HMM in different manners. In Niina and Dozono [21], the spherical self-organizing map (S-SOM) is proposed, which uses HMM models as neurons (S-HMM-SOM) to classify the time-series data. Despite our work, the HMM models in Niina and Dozono [21] are discrete and the author applied the Baum-Welch algorithm for updating the model parameters. In Yamaguchi [22], the authors extended the self-organizing mixture models for multivariate time-series, assuming that the time-series are generated by HMMs. This model, which is called a self-organizing hidden Markov model (SOHMM), uses constrained expectation maximization (EM) for HMM parameter estimation. The self-organization in this work is used for meteorological-state visualization.

In another direction of work in Caridakis et al. [23], a SOMM-based architecture is presented for hand-gesture recognition. The approach involves a combination of SOMs and Markov models for gesture-trajectory classification. In this work, the neurons on the SOM map correspond to the states of the Markov models. In Jaziri et al. [24], the combination of the SOM and HMM (SOS-HMM: self-organizing structure of HMM) automatically extracts the structure of an HMM without any prior knowledge of the application domain. In this model, the macro-HMM is represented as a graph of macro-states, where each state represents a micro-HMM. In summary, each neuron in the SOM-HMM collaborative architecture is either an HMM by itself or a hidden state. In our work, each particular neuron on the SOM lattice is associated with an HMM.

In Lebbah et al. [19], a probabilistic self-organizing map called PrSOMS is presented for the clustering and visualization of dependent and non-identically distributed data. In this model, the SOM learning paradigm to produce topology-preserving maps is combined with the probabilistic-learning scheme of the HMM. The SOM is considered to be a grid, forming a discrete topology in which each cell represents a state; the parameters of the model are estimated by maximizing the likelihood of the sequential data set. In another direction of work in Morimoto [16], the effects of meteorological factors on the occurrence of strokes are investigated. The authors used the SOM to obtain weather patterns that would serve as states of the HMMs. They showed that HMMs with states given by the SOM are useful for describing a background process of stroke incidence. This approach considers the SOM as a front-end processor.

In Ferles and Stafylopatis [17, 25], Ferles et al. [26], the fusion and synergy of SOMs and HMMs are employed in biological-molecule studies to meet the increasing requirements imposed by the properties of deoxyribonucleic acid (DNA), ribonucleic acid (RNA), and protein chain molecules. The authors proposed a stochastic unsupervised-learning algorithm based on the integration of the SOM and HMM principles, called self-organizing hidden Markov model map (SOHMMM). The SOHMMM characteristics and capabilities are demonstrated through two series of experiments, based on artificial sequence data and splice-junction gene sequences. However, in these papers, only the discrete-observation setting is addressed. Here, we extend the algorithm for multivariate Gaussian in order to fit the requirements of our anomaly-detection project.

Incremental learning of HMM parameters is the core function of the SOHMMM algorithm, which is based on a stochastic gradient-descent technique. The incremental learning of new data sequences allows HMM parameters to adapt as new data become available, without having to retrain from the start on all the accumulated training data. Various techniques in the literature address this topic. These techniques are classified according to the objective function, optimization technique, and target application, involving the block-wise and symbol-wise learning of parameters. The authors in Khreich et al. [27] presented a comprehensive survey of techniques that are suitable for the incremental learning of HMM parameters, among which, the stochastic gradient-descent technique of the SOHMMM is referred to as one of the numerical-optimization methods.

Additionally, few efforts exist in the literature that exploit the SOM and HMM for anomaly-detection purposes (Cho [28], Wang et al. [29]). In Cho [28], the authors presented an intrusion-detection system in which the SOM determines the optimal measures of audit data and reduces them to an appropriate size for efficient modeling by the HMM. Two types of HMM are utilized: a single model for all the users and individual models for each user. In another relevant work by Wang et al. [29], the HMM and the SOM are investigated separately as intrusion-detection techniques. The testing results show that the HMM method using the events' transition property outperformed the SOM using the events' frequency property. Regarding the same subject of intrusion detection, the SOM and HMM have a collaborating connection in Cho [28] and competitive roles in Wang et al. [29].

In this work, we intend to benefit from the collaboration of these two techniques (SOM and HMM), as proposed by Ferles and Stafylopatis [25], to extend previous anomaly-detection frameworks applying only HMM (Allahdadi et al. [13, 30], Allahdadi and Morla [31]).

2.2.3 The Self-Organizing Hidden Markov Model Map for Discrete Observations

We start by reviewing SOHMMM as initially formulated by Ferles and Stafylopatis [25]. This model defines a mapping between an observed sequence $\mathbf{x} \triangleq \left(\mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(t)}\right)$ and a two-dimensional lattice of HMMs, which constitute its nodes. Each observation is assumed to be discrete, so $\mathbf{x}^{(\tau)} \in \{1, \ldots, o\}$, being o the size of the dictionary of observations. The topology of a 2-D lattice of m HMMs is defined by a function $\boldsymbol{r} : \{1, \ldots, m\} \mapsto \mathbb{R}^2$, mapping the indices of the m nodes to the respective coordinates in the plane. Given the lattice topology, a neighborhood function $v : \{1, \ldots, m\}^2 \mapsto [0, \infty)$ can be defined mapping two nodes in the lattice to a scalar representing how close the two nodes are. In SOHMMM, v is a Gaussian kernel:

$$v(z, z') \triangleq \exp\left(-\lambda ||\boldsymbol{r}(z) - \boldsymbol{r}(z')||^2\right), \tag{2.2}$$

where $\lambda > 0$ is a hyperparameter controlling the rate of the decay. Following the idea of SOMs, the SOHMMM algorithm aims to minimize an *energy function*, defined below:

$$E(\boldsymbol{x}; \Theta) \triangleq -\sum_{\mathrm{z}} p(\boldsymbol{x} \mid \mathrm{z}, \Theta) v(\mathrm{z}, z^*), \tag{2.3}$$

where Θ summarizes all parameters of the HMMs, z indexes the m HMMs in the lattice, and $p(\mathbf{x} \mid \mathrm{z})$ is the marginal distribution of observations of a standard HMM:

$$p(\mathbf{x} \mid \mathrm{z}) = \sum_{\mathbf{h}} p(\mathrm{h}^{(0)} \mid \mathrm{z}) \prod_{\tau=1}^{t} p(\mathrm{h}^{(\tau)} \mid \mathrm{h}^{(\tau-1)}, \mathrm{z}) p(\mathrm{x}^{(\tau)} \mid \mathrm{h}^{(\tau)}, \mathrm{z}), \tag{2.4}$$

where $\mathbf{h} = \left(\mathrm{h}^{(0)}, \ldots, \mathrm{h}^{(t)}\right)$ is the sequence of hidden states of the HMM and $\mathrm{h}^{(\tau)} \in \{1, \ldots, s\}$, being s the number of hidden states. Finally, z^* corresponds to the index of the *winner node* for the observation $\boldsymbol{x}$:

$$z^* \triangleq \underset{z' \in \{1, \ldots, m\}}{\arg\max} \sum_{\mathrm{z}} p(\boldsymbol{x} \mid \mathrm{z}, \Theta) v(\mathrm{z}, z'). \tag{2.5}$$

The set Θ consists of the following parameters:

- the s-dimensional initial state probabilities, $\boldsymbol{\pi}^{(z)}$, where $\pi_h^{(z)} \triangleq p(\mathrm{h}^{(0)} = h \mid \mathrm{z} = z)$, for $h \in \{1, \ldots, s\}$ and $z \in \{1, \ldots, m\}$;
- the $s \times s$ state transition matrices, $\boldsymbol{A}^{(z)}$, where $A_{h,h'}^{(z)} \triangleq p(\mathrm{h}^{(t)} = h' \mid \mathrm{h}^{(t-1)} = h, \mathrm{z} = z)$, for $h, h' \in \{1, \ldots, s\}$ and $z \in \{1, \ldots, m\}$;
- the $s \times o$ emission probability matrices, $\boldsymbol{B}^{(z)}$, where $B_{h,x}^{(z)} \triangleq p(\mathrm{x}^{(\tau)} = x \mid \mathrm{h}^{(\tau)} = h, \mathrm{z} = z)$, for $h \in \{1, \ldots, s\}$, $x \in \{1, \ldots, o\}$, and $z \in \{1, \ldots, m\}$.

Thus, $\Theta \triangleq \{(\boldsymbol{\pi}^{(z)}, \boldsymbol{A}^{(z)}, \boldsymbol{B}^{(z)})\}_{z=1}^{m}$, where each triplet $(\boldsymbol{\pi}^{(z)}, \boldsymbol{A}^{(z)}, \boldsymbol{B}^{(z)})$ completely defines one HMM in the lattice. The SOHMMM online learning algorithm corresponds to stochastic gradient descent over the energy function (2.3). Constrained optimization is avoided by reparametrizing the model using softmax functions, so that standard, unconstrained stochastic gradient descent can be performed over the new parameters $\boldsymbol{u}^{(z)}$, $\boldsymbol{W}^{(z)}$, and $\boldsymbol{R}^{(z)}$:

$$\pi_h^{(z)} = \frac{\exp(u_h^{(z)})}{\sum_{h'=1}^{s} \exp(u_{h'}^{(z)})}, \quad A_{h,h'}^{(z)} = \frac{\exp(W_{h,h'}^{(z)})}{\sum_{h''=1}^{s} \exp(W_{h,h''}^{(z)})}, \quad B_{h,x}^{(z)} = \frac{\exp(R_{h,x}^{(z)})}{\sum_{x'=1}^{o} \exp(R_{h,x'}^{(z)})}. \tag{2.6}$$

The full algorithm, comprising the update equations for each parameter, is presented in Algorithm 1 and its derivation can be found in Ferles and Stafylopatis [25].

2.2.4 Extending SOHMMM to Gaussian Observations

For the purpose of modeling AP usage data, we should consider a slightly different setting where each observation $\mathbf{x}^{(\tau)}$ takes values on the d-dimensional plane $\mathbb{R}^d$, rather than being drawn from a discrete and finite set. Hence, now, sequences $\mathbf{X} \triangleq \left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}\right)$ take values on $\mathbb{R}^{d \times t}$. A sensible option, which may be useful in a broad range of applications, is considering the case of multivariate Gaussian emissions, i.e. $\mathbf{x}^{(\tau)} \mid (\mathrm{h}^{(\tau)}, \mathrm{z}) \sim \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}_h^{(z)}, \boldsymbol{\Sigma}_h^{(z)}\right)$, where $\boldsymbol{\mu}_h^{(z)} \in \mathbb{R}^d$ is the mean and $\boldsymbol{\Sigma}_h^{(z)} \in \mathbb{R}^{d \times d}$ is the covariance of the Gaussian component corresponding to state h in the lattice node z. These means and covariances replace the emission probability matrices $\boldsymbol{B}^{(z)}$ defined for the case of discrete observations.

We shall now derive the necessary readjustments in Algorithm 1. Let us denote by $\boldsymbol{x}$ the observation at a given time τ (i.e., $\mathbf{x}^{(\tau)} = \boldsymbol{x}$) and consider the corresponding state $\mathrm{h}^{(\tau)}$ and the node z as fixed at h and z, respectively. Thus, we use the notation $p(\boldsymbol{x} \mid h, z)$ as a

Algorithm 1 SOHMMM Learning Algorithm for Discrete Observations [25]

```
1: Inputs: the lattice $r$ with $m$ nodes, the hyperparameter $\lambda$ of the neighborhood function $v$, the set of initial parameters $\Theta^{(0)}$, the learning rate $\eta$, and the number of training iterations trainIter.
2: for $i = 1, \ldots,$ trainIter do
3:     Observe $\mathbf{x} = \boldsymbol{x}$, with length $t$.
4:     for $z = 1, \ldots, m$ do    (Forward-Backward algorithm)
5:         $\alpha_1^{(z)}(h) := \pi_h^{(z)} B_{h,x^{(1)}}^{(z)}, \quad h = 1, \ldots, s;$
6:         $\alpha_{\tau+1}^{(z)}(h) := B_{h,x^{(\tau+1)}}^{(z)} \sum_{h'=1}^{s} \alpha_{\tau+1}^{(z)}(h') A_{h',h}^{(z)}, \quad \tau = 1, \ldots, t-1, \quad h = 1, \ldots, s;$
7:         $\beta_t^{(z)}(h) := 1, \quad h = 1, \ldots, s;$
8:         $\beta_\tau^{(z)}(h) := \sum_{h'=1}^{s} A_{h,h'}^{(z)} B_{h,x^{(\tau+1)}}^{(z)} \beta_{\tau+1}^{(z)}(h'), \quad \tau = t-1, \ldots, 1, \quad h = 1, \ldots, s.$
9:     end for
10:    $z^* := \arg\max_{z' \in \{1,\ldots,m\}} \sum_{z=1}^{m} p(\boldsymbol{x} \mid z, \Theta^{(i-1)}) v(z, z').$
11:    for $z = 1, \ldots, m$ do
12:        $u_h^{(z)} := u_h^{(z)} + \eta \pi_h^{(z)} \left[ B_{h,x^{(1)}}^{(z)} \beta_1^{(z)}(h) - p(\boldsymbol{x} \mid z, \Theta^{(i-1)}) \right], \quad h = 1, \ldots, s;$
13:        $W_{h,h'}^{(z)} := W_{h,h'}^{(z)} + \eta v(z, z^*) A_{h',h}^{(z)} \sum_{\tau=1}^{t-1} \left[ \alpha_\tau^{(z)}(h) B_{h,x^{(\tau+1)}}^{(z)} \beta_{\tau+1}^{(z)}(h') - \alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h) \right], \quad h, h' = 1, \ldots, s;$
14:        $R_{h,x'}^{(z)} := R_{h,x'}^{(z)} + \eta v(z, z^*) \sum_{\tau=1}^{t} \left[ \mathbf{1}_{x^{(\tau)} = x'} \alpha_\tau^{(z)}(h) \beta_\tau(h) - B_{h,x'}^{(z)} \alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h) \right], \quad h = 1, \ldots, s, \quad x' = 1, \ldots, o;$
15:        $\pi_h^{(z)} := \exp(u_h^{(z)}) \Big/ \sum_{h'=1}^{s} \exp(u_{h'}^{(z)}), \quad h = 1, \ldots, s;$
16:        $A_{h,h'}^{(z)} := \exp(W_{h,h'}^{(z)}) \Big/ \sum_{h''=1}^{s} \exp(W_{h,h''}^{(z)}), \quad h, h' = 1, \ldots, s;$
17:        $B_{h,x'}^{(z)} := \exp(R_{h,x'}^{(z)}) \Big/ \sum_{x''=1}^{o} \exp(R_{h,x''}^{(z)}), \quad h = 1, \ldots, s, \quad x' = 1, \ldots, o.$
18:    end for
19:    $\Theta^{(i)} := \{(\boldsymbol{u}^{(z)}, \boldsymbol{W}^{(z)}, \boldsymbol{R}^{(z)})\}_{z=1}^{m}.$
20: end for
```

short for $p(\mathrm{x}^{(\tau)} = \boldsymbol{x} \mid \mathrm{h}^{(\tau)} = h, \mathrm{z} = z)$. We have:

$$\begin{aligned}
\frac{\partial p(\boldsymbol{X} \mid z)}{\partial p(\boldsymbol{x} \mid h, z)} &= \frac{\partial}{\partial p(\boldsymbol{x} \mid h, z)} \left[\sum_{\mathbf{h}} p(\boldsymbol{X}, \mathbf{h} \mid z) \right] \\
&= \frac{\partial}{\partial p(\boldsymbol{x} \mid h, z)} \left[\sum_{\mathbf{h}} p(\mathrm{h}^{(0)} \mid \mathrm{z}) \prod_{\tau'=1}^{t} p(\mathrm{h}^{(\tau')} \mid \mathrm{h}^{(\tau'-1)}, \mathrm{z}) p(\mathbf{x}^{(\tau')} \mid \mathrm{h}^{(\tau')}, \mathrm{z}) \right] \\
&= \frac{1}{p(\boldsymbol{x} \mid h, z)} \sum_{\mathbf{h} \colon \mathrm{h}^{(\tau)} = h} p(\boldsymbol{X}, \mathbf{h} \mid z) \\
&= \frac{p(\boldsymbol{X}, h \mid z)}{p(\boldsymbol{x} \mid h, z)} \\
&= \frac{\alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h)}{p(\boldsymbol{x} \mid h, z)}.
\end{aligned} \tag{2.7}$$

Note that, for discrete observations, $p(x \mid h, z) = B_{h,x}^{(z)}$. However, when we consider the multivariate Gaussian case,

$$\begin{aligned}
p(\boldsymbol{x} \mid h, z) &= \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}_h^{(z)}, \boldsymbol{\Sigma}_h^{(z)}\right) \\
&= \frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma}_h^{(z)})}} \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_h^{(z)})^\top {\boldsymbol{\Sigma}_h^{(z)}}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_h^{(z)}) \right),
\end{aligned} \tag{2.8}$$

where the covariance matrix $\boldsymbol{\Sigma}_h^{(z)}$ must be positive definite. This constraint can be easily guaranteed by reparametrizing it as $\boldsymbol{\Sigma}_h^{(z)} = \boldsymbol{S}_h^{(z)} {\boldsymbol{S}_h^{(z)}}^\top$, where $\boldsymbol{S}_h^{(z)} \in \mathbb{R}^{d \times d}$ is constraint-free. The required gradients are:

$$\nabla_{\boldsymbol{\mu}_h^{(z)}} p(\boldsymbol{x} \mid h, z) = p(\boldsymbol{x} \mid h, z) (\boldsymbol{S}\boldsymbol{S}^\top)^{-1} (\boldsymbol{x} - \boldsymbol{\mu}), \tag{2.9}$$

$$\nabla_{\boldsymbol{S}_h^{(z)}} p(\boldsymbol{x} \mid h, z) = p(\boldsymbol{x} \mid h, z) \boldsymbol{S}^{-1} \left[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top (\boldsymbol{S}\boldsymbol{S}^\top)^{-1} - I_d \right], \tag{2.10}$$

where we have abbreviated $\boldsymbol{\mu}_h^{(z)}$ as $\boldsymbol{\mu}$ and $\boldsymbol{S}_h^{(z)}$ as $\boldsymbol{S}$ to alleviate the notation. Now, by the chain rule,

$$\begin{aligned}
\nabla_{\boldsymbol{\mu}_h^{(z)}} p(\boldsymbol{X} \mid z) &= \sum_{\tau=1}^{t} \frac{\partial p(\boldsymbol{X} \mid z)}{\partial p(\boldsymbol{x}^{(\tau)} \mid h, z)} \nabla_{\boldsymbol{\mu}_h^{(z)}} p(\boldsymbol{x}^{(\tau)} \mid h, z) \\
&= \sum_{\tau=1}^{t} \frac{\alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h)}{p(\boldsymbol{x}^{(\tau)} \mid h, z)} \nabla_{\boldsymbol{\mu}_h^{(z)}} p(\boldsymbol{x}^{(\tau)} \mid h, z) \\
&= \sum_{\tau=1}^{t} \alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h) (\boldsymbol{S}\boldsymbol{S}^\top)^{-1} (\boldsymbol{x}^{(\tau)} - \boldsymbol{\mu}),
\end{aligned} \tag{2.11}$$

$$\nabla_{\boldsymbol{S}_h^{(z)}} p(\boldsymbol{X} \mid z) = \sum_{\tau=1}^{t} \alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h) \boldsymbol{S}^{-1} \left[(\boldsymbol{x}^{(\tau)} - \boldsymbol{\mu})(\boldsymbol{x}^{(\tau)} - \boldsymbol{\mu})^\top (\boldsymbol{S}\boldsymbol{S}^\top)^{-1} - I_d \right]. \tag{2.12}$$

Finally, by applying the chain rule once again, we get the desired gradients of the energy function E with respect to $\boldsymbol{\mu}_h^{(z)}$ and $\boldsymbol{S}_h^{(z)}$:

$$\begin{aligned}
\nabla_{\boldsymbol{\mu}_h^{(z)}} E(\boldsymbol{X}) &= \frac{\partial E}{\partial p(\boldsymbol{X} \mid z)} \nabla_{\boldsymbol{\mu}_h^{(z)}} p(\boldsymbol{X} \mid z) \\
&= -v(z, z^*) \nabla_{\boldsymbol{\mu}_h^{(z)}} p(\boldsymbol{X} \mid z) \\
&= -v(z, z^*) \sum_{\tau=1}^{t} \alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h) (\boldsymbol{S}\boldsymbol{S}^\top)^{-1} (\boldsymbol{x}^{(\tau)} - \boldsymbol{\mu}), \qquad (2.13) \\
\nabla_{\boldsymbol{S}_h^{(z)}} E(\boldsymbol{X}) &= -v(z, z^*) \sum_{\tau=1}^{t} \alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h) \boldsymbol{S}^{-1} \left[(\boldsymbol{x}^{(\tau)} - \boldsymbol{\mu})(\boldsymbol{x}^{(\tau)} - \boldsymbol{\mu})^\top (\boldsymbol{S}\boldsymbol{S}^\top)^{-1} - I_d \right]. \qquad (2.14)
\end{aligned}$$

Given equations (2.13) and (2.14), it is straightforward to adapt Algorithm 1 to our setting. For completeness, the full algorithm is provided in Algorithm 2.

Algorithm 2 SOHMMM Learning Algorithm for Gaussian Observations

```
1:  Inputs: Same as in Algorithm 1.
2:  for $i = 1, \ldots,$ trainIter do
3:      Observe $\mathbf{X} = \boldsymbol{X}$, with length $t$.
4:      Proceed as in lines 4–8 of Algorithm 1, replacing all occurrences of $B_{h,x^{(\tau)}}^{(z)}$ with $\mathcal{N}\left(\boldsymbol{x}^{(\tau)}; \boldsymbol{\mu}_h^{(z)}, \boldsymbol{S}_h^{(z)} \boldsymbol{S}_h^{(z)\top}\right)$.
5:      $z^* := \arg\max_{z' \in \{1,\ldots,m\}} \sum_{z=1}^{m} p(\boldsymbol{X} \mid z, \Theta^{(i-1)}) v(z, z')$.
6:      for $z = 1, \ldots, m$ do
7:          Proceed as in lines 12 and 13 of Algorithm 1, replacing all occurrences of $B_{h,x^{(\tau)}}^{(z)}$ with $\mathcal{N}\left(\boldsymbol{x}^{(\tau)}; \boldsymbol{\mu}_h^{(z)}, \boldsymbol{S}_h^{(z)} \boldsymbol{S}_h^{(z)\top}\right)$ and all occurences of $\boldsymbol{x}$ with $\boldsymbol{X}$;
8:          $\boldsymbol{\mu}_h^{(z)} := \boldsymbol{\mu}_h^{(z)} + \eta v(z, z^*) \sum_{\tau=1}^{t} \alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h) (\boldsymbol{S}_h^{(z)} \boldsymbol{S}_h^{(z)\top})^{-1} (\boldsymbol{x}^{(\tau)} - \boldsymbol{\mu}_h^{(z)}), \quad h = 1, \ldots, s$;
9:          $\boldsymbol{S}_h^{(z)} := \boldsymbol{S}_h^{(z)} + \eta v(z, z^*) \sum_{\tau=1}^{t} \alpha_\tau^{(z)}(h) \beta_\tau^{(z)}(h) \boldsymbol{S}_h^{(z)-1} \cdot \left[ (\boldsymbol{x}^{(\tau)} - \boldsymbol{\mu}_h^{(z)})(\boldsymbol{x}^{(\tau)} - \boldsymbol{\mu}_h^{(z)})^\top (\boldsymbol{S}_h^{(z)} \boldsymbol{S}_h^{(z)\top})^{-1} - I_d \right], \quad h = 1, \ldots, s$;
10:         Proceed as in lines 15 and 16 of Algorithm 1.
11:     end for
12:     $\Theta^{(i)} := \{(\boldsymbol{u}^{(z)}, \boldsymbol{W}^{(z)}, (\boldsymbol{\mu}_h^{(z)}, \boldsymbol{S}_h^{(z)})_{h=1}^{s})\}_{z=1}^{m}$.
13: end for
```

2.2.5 Experiments

In this section, we consider two types of experiments to analyze the capabilities of the SOHMMM algorithm for nonlinear projection and unsupervised clustering. We first validated the accuracy and convergence of the SOHMMM using *synthetic data*, and then explored its significance and efficiency in anomaly detection using *wireless simulation data*.

2.2.5.1 Synthetic Data

In this experiment, we generate observations from two reference HMMs, with one-third of the observations coming from one of the models, and the remaining two-thirds from the other reference model. Then, we train a SOHMMM with six nodes, randomly initialized, with the data from the reference models. It is expected that the SOHMMM nodes would converge to the reference models, with the majority of nodes grouping around the dominant reference model. The SOHMMM nodes (HMMs initialized randomly) are organized in a 2×3 rectangular lattice, as displayed in Figure 2.1. This figure shows the positions and connections of the HMMs. The Euclidean distance between adjacent HMMs is set to 1 and the other distances are computed accordingly. For example, the distance between *hmm0* and *hmm4* is 2, and between *hmm0* and *hmm3* is $\sqrt{2}$.

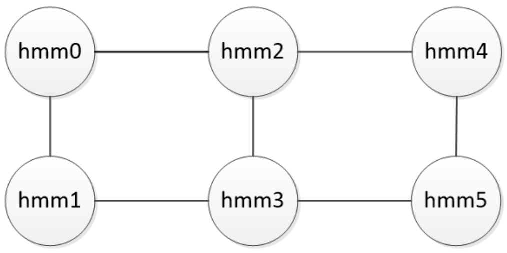

FIGURE 2.1: 2×3 rectangular lattice.

In order to perform this experiment, we need a quantitative measurement of (dis)similarity between two HMMs, so that we can evaluate how close each HMM in the lattice is from the each of the two reference HMMs. It is known that two HMMs representing the same marginal distribution of observations may be parametrized by different parameters (Rabiner [32]), so the dissimilarity should not be computed in parameter space. An alternative to compare two HMMs z and z', proposed by Juang and Rabiner [33], is the following divergence function:

$$D(z,z') = \frac{1}{t}\left[\log p(\boldsymbol{X} \mid z) - \log p(\boldsymbol{X} \mid z')\right], \tag{2.15}$$

where $\boldsymbol{X}$ has length t and is sampled from $p(\mathbf{X} \mid z)$. Equation (2.15) is a Monte Carlo approximation of the Kullback-Leibler divergence between two HMMs and therefore measures how well model z' matches observations generated by model z, relative to how well model z matches observations generated by itself. Since this relationship is asymmetric, we use the following symmetrized version instead:

$$D_s(z,z') = \frac{D(z,z') + D(z',z)}{2}. \tag{2.16}$$

Figure 2.2 demonstrates the initial and final distances between the random HMMs and the reference HMMs, upon applying the SOHMMM algorithm. The training set contains 60 observation sequences from the two reference HMMs (the *ref0* model generates 40 observation sequences and *ref1* generates 20). We trained the random HMMs with these observation sequences. Our main goal in analyzing the aforementioned distance is to examine how the observation sequences influence the random HMMs and whether they maintain their proximity while moving in different directions. Measuring this distance will indicate whether the random HMMs converge to a specific reference HMM.

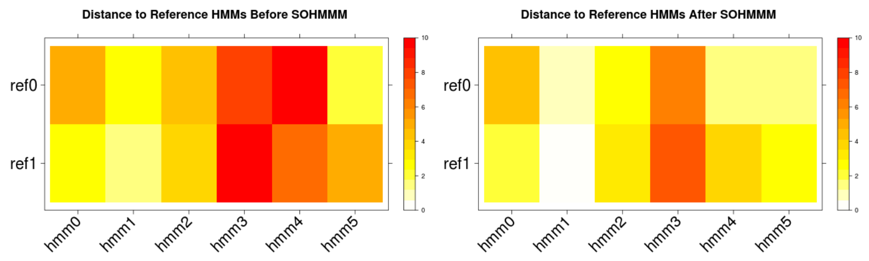

FIGURE 2.2: Monte Carlo approximation of the Kullback-Leibler divergence between the random and reference HMMs before and after applying the SOHMMM algorithm.

As the heatmap plots of Figure 2.2 show, *hmm4* is initially assigned to the *ref1* cluster, as there is a shorter distance between these two models. After applying the algorithm, *hmm4* becomes closer to *ref0*. As another example, *hmm3* is closer to *ref0*, before and after applying the algorithm; however, the overall distance to the reference models decreases and *hmm3* moves closer to both of them while maintaining its relative distance to the references.

The minimum distance between a given random HMM and the reference HMMs defines the cluster that the HMM belongs to. In this experiment, the random HMMs belong to the $(ref1, ref1, ref1, ref0, ref1, ref0)$ and $(ref1, ref1, ref0, ref0, ref0, ref0)$ clusters, before and after applying the algorithm, respectively. The latter clusters show that $\{hmm2, hmm3, hmm4, hmm5\}$ belong to the $ref0$ cluster, the dominant reference models; $\{hmm0, hmm1\}$ are assigned to the $ref1$ cluster, the reference model generating less data. Having analyzed the Euclidean distances of the HMM nodes, Figure 2.1 shows that nearby HMMs are grouped in the same cluster.

We repeated the experiment for various sequence lengths and different types of neighborhood, in terms of the vicinity. We selected a large neighborhood and updated the winner neuron, in addition to all the other neurons in the SOM lattice, based on their relative distances in the lattice. The large-neighborhood experiment employs the SOHMMM algorithm in its original form, so all neurons are updated. In the medium-neighborhood experiment, only the winner HMM and the adjacent neighbors are updated. In the small-neighborhood experiment, only the winner HMM is updated (this setting is known as Z-SOHMMM).

Figure 2.3 displays the Monte Carlo approximation of the Kullback-Leibler divergence between the random and reference HMMs, with large, medium, and small neighborhood sizes, and sequence lengths of 10, 20, and 50. In each experiment, the sum of the distance between the random HMMs and the assigned reference HMM is computed. As Figure 2.3 shows, the sum of the distances to the assigned reference HMMs decreases as the sequence length increases. In addition, the lower distance values are obtained in the large-neighborhood experiment rather than the medium- and the small-neighborhood experiments. This shows that the SOHMMM algorithm provides a better estimation of the reference models by including all the HMMs in the vicinity. The heatmap plots of Figure 2.2 belong to the large-neighborhood experiment with a sequence length of 50, which produced the best overall distance estimate according to Figure 2.3.

2.2.5.2 Wireless Simulation Data

We implemented a set of wireless simulations using the OMNeT++ [34] simulator along with the INET framework [35]. OMNeT++ is a C++-based discrete-event simulator (DES) for modeling communication networks, multiprocessors, and other distributed or parallel systems. As in any discrete-event simulator, events in OMNeT++ take place at discrete

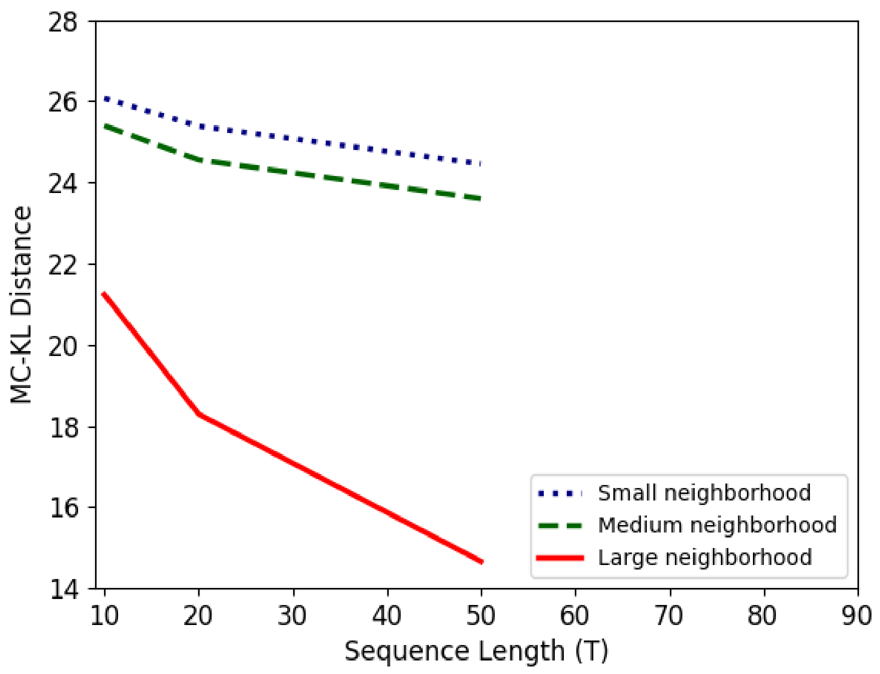

FIGURE 2.3: Monte Carlo approximation of the Kullback-Leibler divergence between the random and reference HMMs with different neighborhood sizes and sequence lengths.

instances in time, and they take zero time to occur. It is assumed that nothing important occurs between two consecutive events. Thus, the simulation time is relevant to the order of the events in the events queue; it could take more or less time than the real CPU time, based on the number of nodes, amount of traffic transferred, and other network details. In our experiment, with the current number of nodes (10 access points (APs) and 100 wireless stations (STAs)) and the defined traffic plan, 10 minutes of simulation time took around two hours of CPU time.

The normal scenario contains 10 APs and 100 STAs. Each STA is initially associated with one of the available APs, depending on its location. During the simulation, the STAs are handed over to other APs, based on their mobility models, when moving around the simulation ground. Furthermore, according to the defined traffic plans, each node sends and receives packets to the existing servers. Figure 2.4 shows images of a normal scenario, the location of the APs, STAs, and servers, and the location of the wireless stations. More details on the mobility models of the wireless stations, traffic generation, available servers, and path-loss models can be found in [13].

Each sequence contains 40 consecutive time-slots of 15s simulation time each. Our simulation consists of one normal scenario and four anomalous scenarios: AP shutdown/halt, AP overload, noise, and flash crowd. We simulated 15 instances of 3000s of simulation time for the normal scenario, and five instances of 3000s of simulation time for each of the anomalous scenarios. In the following paragraphs, we explain how we employed these data for training and testing the SOHMMM algorithm. Data is divided into training and test sets according to a 80%/20% random split.

In the SOHMMM adapted to the problem of anomaly detection in AP usage data, the self-organization learning process is elaborated as follows. The models associated with the SOM neurons are three-state HMMs (according to the best practice found in Allahdadi et al. [13], Allahdadi and Morla [31]). As the new sequence arrives, an HMM with the highest log-likelihood value is selected as the winner model. In the AP usage data, as the newly arrived sequence belongs to a pre-determined AP, in most cases, the winner model is related to the same AP that originated the observation sequence. However, this is not always the case, and the competition defines the winner HMM eventually. Thereafter, the HMM model of the winner AP is updated by the new data sequence, and the HMMs in the neighborhood of the winner AP are also updated.

At this point, it should be noted that the APs in the vicinity of the winner AP are updated, to some extent, relative to their proximity (or similarity) to the winner AP. In our case, the neighborhood area has an irregular shape and contains the first-level adjacent APs in the vicinity of the winner AP. As the locations of the APs are already determined in the wireless ground (Figure 2.4), the Euclidean distances between all APs are calculated and kept in a complete graph. Then, a filter is applied to update only APs with a certain distance from the winner AP (in this case, half of the maximum distance between AP pairs in the distance graph).

During the learning process, the nearby HMMs, up to a certain distance, activate each other to gain some information from the new observation sequence. As the distant HMMs can only gain an insignificant amount of information from the new observation sequence, we utilized the aforementioned filter to avoid updating very distant HMMs, and quicken the process. The neighborhood function that we used in this experiment is a slightly modified version of equation (2.2) (with $\lambda = 10$), based on the relative distances of the winner AP and all its adjacent neighbors:

$$v(z, z') \triangleq \exp\left(-10 \frac{||\boldsymbol{r}(z) - \boldsymbol{r}(z')||^2}{\sum_{z'' \in \mathrm{Adj}(z')} ||\boldsymbol{r}(z'') - \boldsymbol{r}(z')||^2}\right), \tag{2.17}$$

where $\mathrm{Adj}(z')$ denotes the set of nodes adjacent to node z'.

We compared the SOHMMM algorithm to two other approaches: i) hidden Markov model initialized with universal background model (HMM-UBM), addressed in detail in [13], and ii) the SOHMMM algorithm with a zero neighborhood, which utilizes the incremental-learning part of the SOHMMM algorithm, without updating the HMMs in the vicinity (Z-SOHMMM).

Equivalent amounts of normal and anomalous data were used to initialize the HMM-UBM model: Five weeks of normal data and one week for each anomalous case (5X), 10 weeks' data overall. Each week contains five working days. In the anomalous cases, the time and period of the anomalies are different for each day. We used one more week of each scenario to train the model and then used the one remaining week of each case to test the model. The results are represented as receiver operating characteristic (ROC) curves on the test set.

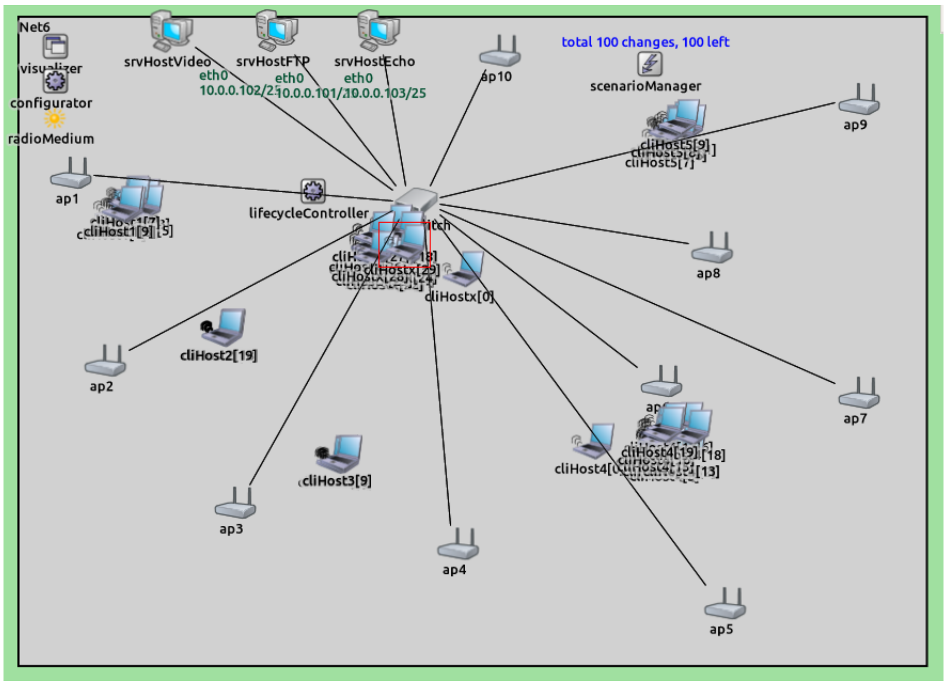

FIGURE 2.4: Wireless network simulated in OMNeT++/INET.

The following anomalous scenarios were simulated:

- AP shutdown/halt, where the AP simply stops working (e.g. due to power failure);
- AP overload, which happens when excessive channel utilization occurs as a consequence of excessive traffic by a number of wireless users;
- noise in the wireless communication channel;

- flash crowd, where multiple users associate (flash crowd arrival) or dis-associate from the same AP almost simultaneously;
- miscelaneous anomalies, where multiple anomalies occur on the same day.

Table 2.1 presents the results for anomaly detection in the aforementioned scenarios. In this table, the AUC values and F1 scores of each anomalous scenario are presented for the HMM-UBM, Z-SOHMMM, and SOHMMM models. The anomalous APs are also indicated. The higher AUC values demonstrate that SOHMMM outperforms the remaining models since it is better at discriminating normal and anomalous samples. The F1 score, on the other hand, is a measure of the accuracy, considering both the precision and recall to compute the score. The obtained values for the F1 score once again confirm the superiority of the SOHMMM algorithm.

Please refer to Allahdadi et al. [3] for more details about the experiments and further experimental results.

TABLE 2.1: Summary of the information regarding the wireless anomalous scenarios. The best results for each scenario are in bold. (HU– HMM-UBM, ZS– Z-SOHMMM, S– SOHMMM)

		AUC Value			F1 Score		
		HU	ZS	S	HU	ZS	S
AP Shutdown/Halt	AP2	0.91	0.95	**0.99**	0.84	0.85	**0.86**
	AP4	0.71	0.93	**1.00**	0.84	0.85	**0.98**
AP Overload	AP2	0.99	0.99	**1.00**	0.87	0.87	0.87
Noise	AP2	0.78	0.62	**0.82**	0.89	0.89	**0.92**
Flash Crowd - Arrival	AP2	0.80	0.91	**0.94**	0.85	0.85	**0.86**
Flash Crowd - Departure	AP2	0.96	**0.97**	**0.97**	0.82	**0.83**	**0.83**
Miscellaneous Anomalies	AP2	0.74	0.73	**0.88**	0.70	0.70	**0.82**
	AP3	0.69	0.69	**0.71**	0.70	0.70	**0.77**

2.2.6 Conclusions

In the current work, we applied a hybrid integration of the self-organizing map (SOM) and the hidden Markov model (HMM), called the SOHMMM, for anomaly detection in 802.11 wireless networks. We further extended the online gradient-descent unsupervised-learning algorithm of the SOHMMM for multivariate Gaussian emissions. We employed this algorithm specifically for anomaly detection in 802.11 wireless AP usage data.

The experimental analysis investigated two main data sets: *synthetic data* and *wireless simulation data*. In the synthetic data analysis, we generated six random HMMs and trained them with the observation sequences of two reference HMMs with predefined parameters. We then estimated the distance between HMMs as the Monte Carlo approximation of the Kullback-Leibler divergence, and computed the final HMM clusters, based on the minimum distance between the random HMMs and reference HMMs. We repeated the experiment for various observation-sequence lengths and different neighborhood sizes, and showed that, for the large neighborhood, the SOHMMM algorithm provided a better estimation of the reference models, including all the HMMs in the vicinity. Moreover, we presented the decay of the learning rate and the convergence of the loss function.

In the analysis regarding the wireless-simulation data, we showed how the SOHMMM algorithm improved the anomaly-detection accuracy and sensitivity, compared to the HMM-UBM and Z-SOHMMM techniques in the AP shutdown/halt, AP overload, noise, and flash-crowd anomalous scenarios. We further investigated the combination of several anomalies in one observation sequence as miscellaneous anomalies and showed that the SOHMMM was capable of detecting contrasting anomalous cases while the HMM-UBM was not.

2.3 SpaMHMM: Sparse Mixture of Hidden Markov Models for Graph Connected Entities

2.3.1 Overview

Inspired by the formulation of equation (2.1), we propose to model the generative distribution of the data coming from each of the k nodes of a network as a sparse mixture obtained from a dictionary of generative distributions. Specifically, we shall model the distribution for each node as a sparse mixture over a 'large' shared dictionary of HMMs, where each HMM corresponds to an individual atom from the dictionary. The field knowledge about the similarities between nodes is summarized in an affinity matrix. The objective function of the learning process promotes reusing HMM atoms between similar nodes. We now formalize these ideas.

2.3.2 Model formulation

2.3.2.1 Definition

Assume we have a set of nodes $\mathbb{Y} = \{1, \ldots, k\}$ connected by an undirected weighted graph $\mathcal{G}$, expressed by a symmetric matrix $\boldsymbol{G} \in \mathbb{R}^{k \times k}$. These nodes thus form a network, in which the weights are assumed to represent degrees of affinity between each pair of nodes (i.e. the greater the edge weight, the more the respective nodes *like* to agree). The nodes y in the graph produce d-dimensional sequences $\mathbf{X} = \left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}\right)$, $\mathbf{x}^{(\tau)} \in \mathbb{R}^d$, whose conditional distribution we shall model using a mixture of HMMs:

$$p(\mathbf{X} \mid \mathrm{y}) = \sum_{\mathrm{z}} p(\mathrm{z} \mid \mathrm{y}) p(\mathbf{X} \mid \mathrm{z}), \tag{2.18}$$

where $\mathrm{z} \in \{1, \ldots, m\}$ is a latent random variable, being m the size of the mixture. This is a particular realization of equation (2.1) where f is the probability density function $p(\mathbf{X} \mid \mathrm{y})$ and the coefficients s_z correspond to the probabilities $p(\mathrm{z} = z \mid \mathrm{y})$. Here, $p(\mathbf{X} \mid \mathrm{z})$ is the marginal distribution of observations of a standard first-order homogeneous HMM:

$$p(\mathbf{X} \mid \mathrm{z}) = \sum_{\mathbf{h}} p(\mathrm{h}^{(0)} \mid \mathrm{z}) \prod_{t} p(\mathrm{h}^{(t)} \mid \mathrm{h}^{(t-1)}, \mathrm{z}) p(\mathbf{x}^{(t)} \mid \mathrm{h}^{(t)}, \mathrm{z}), \tag{2.19}$$

where $\mathbf{h} = \left(\mathrm{h}^{(0)}, \ldots, \mathrm{h}^{(t)}\right), \mathrm{h}^{(\tau)} \in \{1, \ldots, s\}$, is the sequence of hidden states of the HMM, being s the number of hidden states. Note that the factorization in equation (2.18) imposes conditional independence between the sequence $\mathbf{X}$ and the node y, given the latent variable z. This is a key assumption of this model, since this way the distributions for the observations in the nodes in $\mathbb{Y}$ share the same dictionary of HMMs, promoting parameter sharing among the k mixtures. The Bayesian network representing this model is presented in Figure 2.5.

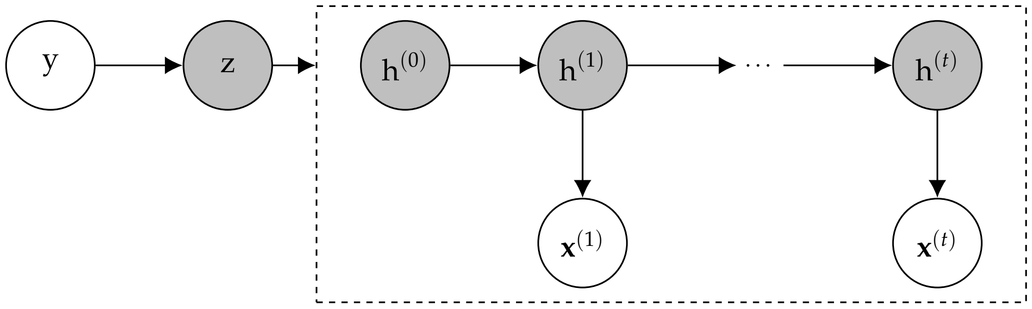

FIGURE 2.5: Representation of the model as a Bayesian network. The node z is a parent of all nodes inside the dashed box (the connections were omitted for clarity). Gray nodes represent latent variables.

2.3.2.2 Inference

Given an observed sequence $\boldsymbol{X}$ and its corresponding node $y \in \mathbb{Y}$, the inference problem here consists in finding the likelihood $p(\mathbf{X} = \boldsymbol{X} \mid \mathrm{y} = y)$ (from now on, abbreviated as $p(\boldsymbol{X} \mid y)$) as defined by equations (2.18) and (2.19). The marginals $p(\boldsymbol{X} \mid \mathrm{z})$ of each HMM in the mixture may be computed efficiently, in $O(s^2 t)$ time, using the Forward algorithm (Rabiner and Juang [36]). Then, $p(\boldsymbol{X} \mid y)$ is obtained by applying equation (2.18), so inference in the overall model is done in at most $O(ms^2 t)$ time. As we shall see, however, the mixtures we get after learning will often be sparse (see Section 2.3.2.3), leading to an even smaller time complexity.

2.3.2.3 Learning

Given an i.i.d. dataset consisting of n tuples $(\boldsymbol{X}_i, y_i)$ of sequences of observations $\boldsymbol{X}_i = \left(\boldsymbol{x}_i^{(1)}, \ldots, \boldsymbol{x}_i^{(t_i)}\right)$ and their respective nodes $y_i \in \mathbb{Y}$, the model defined by equations (**??**) and (**??**) may be easily trained using the Expectation-Maximization (EM) algorithm (Dempster et al. [37]), (locally) maximizing the usual log-likelihood objective:

$$J(\theta) \triangleq \sum_{i=1}^{n} \log p(\boldsymbol{X}_i \mid y_i; \Theta), \tag{2.20}$$

where Θ represents all model parameters, namely:

1. the m-dimensional mixture coefficients, $\boldsymbol{\alpha}^{(y)}$, where $\alpha_z^{(y)} \triangleq p(\mathrm{z} = z \mid \mathrm{y} = y)$, for $z \in \{1, \ldots, m\}$ and $y \in \{1, \ldots, k\}$;

2. the s-dimensional initial state probabilities, $\boldsymbol{\pi}^{(z)}$, where $\pi_h^{(z)} \triangleq p(\mathrm{h}^{(0)} = h \mid \mathrm{z} = z)$, for $h \in \{1, \ldots, s\}$ and $z \in \{1, \ldots, m\}$;

3. the $s \times s$ state transition matrices, $\boldsymbol{A}^{(z)}$, where $A_{h,h'}^{(z)} \triangleq p(\mathrm{h}^{(t)} = h' \mid \mathrm{h}^{(t-1)} = h, \mathrm{z} = z)$, for $h, h' \in \{1, \ldots, s\}$ and $z \in \{1, \ldots, m\}$;

4. the emission probability means, $\boldsymbol{\mu}_h^{(z)} \in \mathbb{R}^d$, for $z \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, s\}$;

5. the emission probability diagonal covariance matrices, $\Sigma_h^{(z)} = \mathrm{diag}(\boldsymbol{\sigma}_h^{(z)})^2$, where $\boldsymbol{\sigma}_h^{(z)} \in \mathbb{R}^d$, for $z \in \{1, \ldots, m\}$ and $h \in \{1, \ldots, s\}$.

Here, we are assuming that the emission probabilities $p(\mathrm{x}^{(t)} \mid \mathrm{h}^{(t)}, \mathrm{z})$ are Gaussian with diagonal covariances. This introduces almost no loss of generality since the extension of this work to discrete observations or other types of continuous emission distributions is straightforward.

The procedure to maximize objective (2.20) using EM is described in Algorithm 3. The update formulas follow from the standard EM procedure and can be obtained by viewing this model as a Bayesian network or by following the derivation detailed in Section A.1. However, the objective (2.20) does not take advantage of the known structure of $\mathcal{G}$. In order to exploit this information, we introduce a regularization term, maximizing the following objective instead:

$$\begin{aligned} J_r(\Theta) &\triangleq \frac{1}{n}\sum_{i=1}^{n}\log p(\boldsymbol{X}_i \mid y_i;\Theta) + \frac{\lambda}{2}\sum_{\substack{y,y'=1,\\ y'\neq y}}^{k} G_{y,y'}\mathbb{E}_{\mathrm{z}\sim p(\mathrm{z}\mid y;\Theta)}[p(\mathrm{z} \mid y';\Theta)] \\ &= \frac{1}{n}\sum_{i=1}^{n}\log p(\boldsymbol{X}_i \mid y_i;\Theta) + \frac{\lambda}{2}\sum_{\substack{y,y'=1,\\ y'\neq y}}^{k} G_{y,y'}\boldsymbol{\alpha}^{(y)\top}\boldsymbol{\alpha}^{(y')}, \end{aligned} \tag{2.21}$$

where $\lambda \geq 0$ controls the relative weight of the two terms in the objective. Note that this regularization term favors nodes connected by edges with large positive weights to have similar mixture coefficients and thus share mixture components. On the other hand, nodes connected by edges with large negative weights will tend to have orthogonal mixture coefficients, being described by disjoint sets of components. These observations agree with our prior assumption that the edge weights express degrees of similarity between each pair of nodes. Proposition 2.1 formalizes these statements and enlightens interesting properties about the expectations $\mathbb{E}_{\mathrm{z}\sim p(\mathrm{z}\mid y)}[p(\mathrm{z} \mid y')]$.

Proposition 2.1. *For any integer $m > 1$, let $\mathbb{P}_m$ be the set of all probability distributions over the set $\{1,\ldots,m\}$. We have:*

1. $\min_{p,q\in\mathbb{P}_m}\mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = 0$;
2. $\arg\min_{p,q\in\mathbb{P}_m}\mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = \{p,q \in \mathbb{P}_M \mid \forall z \in \{1,\ldots,m\} : p(\mathrm{z}=z)q(\mathrm{z}=z) = 0\}$;
3. $\max_{p,q\in\mathbb{P}_m}\mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = 1$;
4. $\arg\max_{p,q\in\mathbb{P}_m}\mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = \{p,q \in \mathbb{P}_M \mid \exists z \in \{1,\ldots,m\} : p(\mathrm{z}=z) = q(\mathrm{z}=z) = 1\}$.

Proof. By the definition of expectation, for any $p,q \in \mathbb{P}_m$,

$$\mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = \sum_{\mathrm{z}} p(\mathrm{z})q(\mathrm{z}). \tag{2.22}$$

Statements 1 and 2 follow immediately from the fact that every term in the right-hand side of (2.22) is non-negative and $m > 1$. For the remaining, we rewrite (2.22) as the dot product of two m-dimensional vectors $\boldsymbol{\alpha}_p$ and $\boldsymbol{\alpha}_q$, representing the two distributions p and q, respectively, and we use the following linear algebra inequalities to build an upper bound for this expectation:

$$\mathbb{E}_{z\sim p}[q(z)] = \boldsymbol{\alpha}_p^\top \boldsymbol{\alpha}_q \leq ||\boldsymbol{\alpha}_p||_2 ||\boldsymbol{\alpha}_q||_2 \leq ||\boldsymbol{\alpha}_p||_1 ||\boldsymbol{\alpha}_q||_1 = 1, \tag{2.23}$$

where $||\cdot||_1$ and $||\cdot||_2$ are the L^1 and L^2 norms, respectively. Clearly, the equality $\mathbb{E}_{z\sim p}[q(z)] = 1$ holds if p and q are chosen from the set defined in statement 4, where the distributions p and q are the same and they are non-zero for a single assignment of z. This proves statement 3. Now, to prove statement 4, it suffices to show that there are no other maximizers. The first inequality in (2.23) is transformed into an equality if and only if $\boldsymbol{\alpha}_p = \boldsymbol{\alpha}_q$, which means $p \equiv q$. The second inequality becomes an equality when the L^1 and L^2 norms of the vectors coincide, which happens if and only if the vectors have only one non-zero component, concluding the proof. □

Specifically, given two distinct nodes $y, y' \in \mathbb{Y}$, if $G_{y,y'} > 0$, the regularization term for these nodes is maximum (and equal to $G_{y,y'}$) when the mixtures for these two nodes are the same and have one single active component (i.e. one mixture component whose coefficient is non-zero). On the contrary, if $G_{y,y'} < 0$, the term is maximized (and equal to zero) when the mixtures for the two nodes do not share any active components. In both cases, though, we conclude from Proposition 2.1 that we are favoring sparse mixtures. We see sparsity as an important feature since it allows the size m of the dictionary of models to be large and therefore expressive without compromising our rational that the observations in a given node are well modeled by a mixture of only a few HMMs. This way, some components will specialize on describing the behavior of some nodes, while others will specialize on different nodes. Moreover, sparse mixtures yield faster inference, more interpretable models and (possibly) less overfitting. By setting $\lambda = 0$, we clearly get the initial objective (2.20), where inter-node correlations are modeled only via parameter sharing. As $\lambda \to \infty$, two interesting scenarios may be anticipated. If $G_{y,y'} > 0$, $\forall y, y' \in \mathbb{Y}$, all nodes will tend do share the same single mixture component, i.e. we would be learning one single HMM to describe the whole network. If $G_{y,y'} < 0$, $\forall y, y' \in \mathbb{Y}$, and $m \geq K$, each node would tend to learn its own HMM model independently from all the others. Again, in both scenarios, the obtained mixtures are sparse.

The objective function (2.21) can still be maximized via EM (see details in Section A.2). However, the introduction of the regularization term in the objective makes it impossible to find a closed form solution for the update formula of the mixture coefficients. Thus, in the M-step, we need to resort to gradient ascent to update these parameters. In order to ensure that the gradient ascent iterative steps lead to admissible solutions, we adopt the following reparametrization from Yang et al. [38]:

$$\alpha_z^{(y)} = \frac{\max(0, \beta_z^{(y)})^2}{\sum_{z'=1}^{m} \max(0, \beta_{z'}^{(y)})^2}, \tag{2.24}$$

and so we can treat $\boldsymbol{\beta}^{(y)}$ as an unconstrained parameter vector. This reparametrization clearly resembles the softmax function, but, contrarily to that one, admits sparse outputs. The squared terms in equation (2.24) aim only to make the optimization more stable. The optimization steps for the objective (2.21) using this reparametrization are described in Algorithm 4.

Algorithm 3 EM algorithm for the mixture without regularization (MHMM).

```
1: **Inputs:** The training set, consisting of $n$ tuples $(\mathbf{X}_i, y_i)$, a set of initial parameters $\Theta^{(0)}$, and the number of training iterations trainIter.
2: **for** $j = 1, \ldots,$ trainIter **do**
3:     **Sufficient statistics:**
4:         $n_y := \sum_{i=1}^{n} \mathbf{1}_{y_i = y}$, for $y = 1, \ldots, k$;
5:         Obtain the mixture posteriors $\eta_i^{(z)} := p(z \mid \mathbf{X}_i, y_i)$, for $i = 1, \ldots, n$ and $z = 1, \ldots, m$, by computing $\tilde{\eta}_i^{(z)} := p(\mathbf{X}_i \mid z) p(z \mid y_i)$ and normalizing it;
6:         Obtain the state posteriors $\gamma_{i,h}^{(z)}(\tau) := p(\mathrm{h}^{(\tau)} = h \mid z, \mathbf{X}_i)$ and $\xi_{i,h,h'}^{(z)}(\tau) := p(\mathrm{h}^{(t-1)} = h, \mathrm{h}^{(t)} = h' \mid z, \mathbf{X}_i)$, for $i = 1, \ldots, n$, $\tau = 1, \ldots, t_i$, $z = 1, \ldots, m$, and $h, h' = 1, \ldots, s$, as done in the Baum-Welch algorithm (Baum [39]).
7:     **M-step:**
8:         $\alpha_z^{(y)} := \frac{\sum_{i=1}^{n} \eta_i^{(z)} \mathbf{1}_{y_i = y}}{n_y}$, for $y = 1, \ldots, k$ and $z = 1, \ldots, m$, obtaining $\boldsymbol{\alpha}_y$;
9:         $\pi_h^{(z)} := \frac{\sum_{i=1}^{n} \eta_i^{(z)} \gamma_{i,h}^{(z)}(0)}{\sum_{i=1}^{n} \eta_i^{(z)}}$, for $z = 1, \ldots, m$ and $h = 1, \ldots, s$, obtaining $\boldsymbol{\pi}^{(z)}$;
10:        $A_{h,h'}^{(z)} := \frac{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{\tau=1}^{t_i} \xi_{i,h,h'}^{(z)}(\tau)}{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{\tau=0}^{t_i - 1} \gamma_{i,h}^{(z)}(\tau)}$, for $z = 1, \ldots, m$, and $h, h' = 1, \ldots, s$, obtaining $\boldsymbol{A}^{(z)}$;
11:        $\boldsymbol{\mu}_h^{(z)} := \frac{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{\tau=1}^{t_i} \gamma_{i,h}^{(z)}(\tau) \boldsymbol{x}_i^{(\tau)}}{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{\tau=1}^{t_i} \gamma_{i,h}^{(z)}(\tau)}$, for $z = 1, \ldots, m$ and $h = 1, \ldots, s$;
12:        ${\boldsymbol{\sigma}_h^{(z)}}^2 := \frac{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{\tau=1}^{t_i} \gamma_{i,h}^{(z)}(\tau) \left( \boldsymbol{x}_i^{(t)} - \boldsymbol{\mu}_h^{(z)} \right)^2}{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{\tau=1}^{t_i} \gamma_{i,h}^{(z)}(\tau)}$, for $z = 1, \ldots, m$ and $h = 1, \ldots, s$;
13:        $\Theta^{(j)} := \bigcup_{y,z,h} \left\{ \boldsymbol{\alpha}^{(y)}, \boldsymbol{\pi}^{(z)}, \boldsymbol{A}^{(z)}, \boldsymbol{\mu}_h^{(z)}, \boldsymbol{\sigma}_h^{(z)} \right\}$.
14: **end for**
```

Algorithm 4 EM algorithm for the mixture with regularization (SpaMHMM).

```
1: Inputs: The training set, consisting of $n$ tuples $(\boldsymbol{X}_i, y_i)$, the matrix $\boldsymbol{G}$ describing the graph $\mathcal{G}$, the regularization hyperparameter $\lambda$, a set of initial parameters $\Theta^{(0)}$, the number of training iterations trainIter, the number of gradient ascent iterations mIter to perform on each M-step, and the learning rate $\rho$ to perform gradient ascent over the mixture coefficients.
2: for $j = 1, \ldots,$trainIter do
3:     Sufficient statistics: same as in Algorithm 3.
4:     M-step:
5:         for $l = 1, \ldots,$mIter do
6:             $\psi_z^{(y)} := \frac{1}{n} \sum_{i=1}^{n} \left( \eta_i^{(z)} - \alpha_z^{(y)} \right) \mathbf{1}_{y_i = y}$, for $y = 1, \ldots, k$ and $z = 1, \ldots, m$;
7:             $\omega_z^{(y)} := \alpha_z^{(y)} \sum_{y'=1}^{k} G_{y',y} \left( \alpha_z^{(y')} - \boldsymbol{\alpha}^{(y')\top} \boldsymbol{\alpha}^{(y)} \right) \mathbf{1}_{y' \neq y}$, for $y = 1, \ldots, k$ and $z = 1, \ldots, m$;
8:             $\delta_z^{(y)} := \frac{2}{\beta_z^{(y)}} \left( \psi_z^{(y)} + \lambda \omega_z^{(y)} \right) \mathbf{1}_{\beta_z^{(y)} > 0}$, for $y = 1, \ldots, k$ and $m = 1, \ldots, z$;
9:             $\beta_z^{(y)} := \beta_z^{(y)} + \rho \delta_z^{(y)}$, for $y = 1, \ldots, k$ and $z = 1, \ldots, m$, obtaining $\boldsymbol{\beta}^{(y)}$;
10:            $\alpha_z^{(y)} := \frac{\max(0, \beta_z^{(y)})^2}{\sum_{z'=1}^{m} \max(0, \beta_{z'}^{(y)})^2}$, for $y = 1, \ldots, k$ and $z = 1, \ldots, m$, , obtaining $\boldsymbol{\alpha}^{(y)}$.
11:        end for
12:        Proceed as in lines 9–12 of Algorithm 3;
13:        $\Theta^{(j)} := \bigcup_{y,z,h} \left\{ \boldsymbol{\beta}^{(y)}, \boldsymbol{\pi}^{(z)}, \boldsymbol{A}^{(z)}, \boldsymbol{\mu}_h^{(z)}, \boldsymbol{\sigma}_h^{(z)} \right\}$.
14: end for
```

2.4 Experimental Evaluation

The model was developed on top of the library hmmlearn Lebedev [40] for Python, which implements inference and unsupervised learning for the standard HMM using a wide variety of emission distributions. Both learning and inference use the hmmlearn API, with the appropriate adjustments for our models. For reproducibility purposes, we make our source code, pre-trained models and the datasets publicly available [*].

We evaluate four different models in our experiments: a model consisting of a single HMM (denoted as 1-HMM) trained on sequences from all graph nodes; a model consisting of k HMMs trained independently (denoted as k-HMM), one for each graph node; a mixture of HMMs (denoted as MHMM) as defined in this work (equations (**??**) and (**??**)), trained to maximize the usual log-likelihood objective (2.20); a mixture of HMMs (denoted as SpaMHMM) as the previous one, trained to maximize our regularized objective (2.21).

Models 1-HMM, k-HMM and MHMM will be our baselines. We shall compare the performance of these models with that of SpaMHMM and, for the case of MHMM, we shall also verify if SpaMHMM actually produces sparser mixtures in general, as argued in Section 2.3.2.3. In order to ensure a fair comparison, we train models with approximately the same number of possible state transitions. Hence, given an MHMM or SpaMHMM with m mixture components and s states per component, we train a 1-HMM with $\approx s\sqrt{m}$ states and a k-HMM with $\approx s\sqrt{m/k}$ states per HMM. We initialize the mixture coefficients in MHMM and SpaMHMM randomly, while the state transition matrices and the initial state probabilities are initialized uniformly. Means are initialized using k-means, with k equal to the number of hidden states in the HMM, and covariances are initialized with the diagonal of the training data covariance. Models 1-HMM and k-HMM are trained using the Baum-Welch algorithm, MHMM is trained using Algorithm 3 and SpaMHMM is trained using Algorithm 4. However, we opted to use Adam (Kingma and Ba [41]) instead of *vanilla* gradient ascent in the inner loop of Algorithm 4, since its per-parameter learning rate proved to be beneficial for faster convergence.

[*] https://github.com/dpernes/spamhmm

2.4.1 Anomaly detection in Wi-Fi networks

A typical Wi-Fi network infrastructure is constituted by k access points (APs) distributed in a given space. The network users may alternate between these APs seamlessly, usually connecting to the closest one. There is a wide variety of anomalies that may happen during the operation of such network and their automatic detection is, therefore, of great importance for future mitigation plans. Some anomalous behaviors are: overloaded APs, failed or crashed APs, persistent radio frequency interference between adjacent APs, authentication failures, etc. However, obtaining reliable ground truth annotation of these anomalies in entire wireless networks is costly and time consuming. Under these circumstances, using data obtained through realistic network simulations is a common practice.

In order to evaluate our model in the aforementioned scenario, we have followed the procedure of Allahdadi et al. [13], performing extensive network simulations in a typical Wi-Fi network setup (IEEE 802.11 WLANg 2.4 GHz in infrastructure mode) using OMNeT++ [34] and INET [35] simulators. Our network consists of 10 APs and 100 users accessing it. The pairwise distances between APs are known and fixed. Each sequence contains information about the traffic in a given AP during 10 consecutive hours and is divided in time slots of 15 minutes without overlap. Thus, every sequence has the same length, which is equal to 40 samples (time slots). Each sample contains the following 7 features: the number of unique users connected to the AP, the number of sessions within the AP, the total duration (in seconds) of association time of all current users, the number of octets transmitted and received in the AP and the number of packets transmitted and received in the AP. Anomalies typically occur for a limited amount of time within the whole sequence. However, in this experiment, we label a sequence as "anomalous" if there is at least one anomaly period in the sequence and we label it as "normal" otherwise. One of the simulations includes normal data only, while the remaining include both normal and anomalous sequences. In order to avoid contamination of normal data with anomalies that may occur simultaneously in other APs, we used the data of the normal simulation for training (150 sequences) and the remaining data for testing (378 normal and 42 anomalous sequences).

In a Wi-Fi network, as users move in the covered area, they disconnect from one AP and they immediately connect to another in the vicinity. As such, the traffic in adjacent APs may be expected to be similar. Following this idea, the weight $G_{y,y'}$, associated with the edge connecting nodes y and y' in graph $\mathcal{G}$, was set to the inverse of the distance between APs y and y' and normalized so that $\max_{y,y'} G_{y,y'} = 1$. As in Allahdadi et al. [13], sequences were preprocessed by subtracting the mean and dividing by the standard deviation and applying PCA, reducing the number of features to 3. For MHMM, we did 3-fold cross validation of the number of mixture components M and hidden states per component s. We ended up using $m = 15$ and $s = 10$. We then used the same values of m and s for SpaMHMM and we did 3-fold cross validation for the regularization hyperparameter λ in the range $[10^{-4}, 1]$. The value $\lambda = 10^{-1}$ was chosen. We also cross-validated the number of hidden states in 1-HMM and k-HMM around the values indicated in Section 2.4. Every model was trained for 100 EM iterations or until the loss plateaus. For SpaMHMM, we did 100 iterations of the inner loop on each M-step, using a learning rate $\rho = 10^{-3}$. We repeat training 10 times for each model, starting from different random initializations, in order to reduce the likelihood of erroneous results due to local minima trapping.

Models were evaluated by computing the average log-likelihood per sample on normal and anomalous test data, plotting the receiver operating characteristic (ROC) curves and computing the respective areas under the curves (AUCs). The small standard deviations in Table 2.2 attest the robustness of the adopted initialization scheme and learning algorithms. Figure 2.6 shows that the ROC curves for MHMM and SpaMHMM are very similar and that these models clearly outperform 1-HMM and k-HMM. This is confirmed by the AUC and log-likelihood results in Table 2.2. Although k-HMM achieved the best (lowest) average log-likelihood on anomalous data, this result is not relevant, since it also achieved the worst (lowest) average log-likelihood on normal data. This is in fact the model with the worst performance, as shown by its ROC and respective AUC.

The bad performance of k-HMM likely results mostly from the small amount of data that each of the K models is trained with: in k-HMM, each HMM is trained with the data from the graph node (AP) that it is assigned to. The low log-likelihood value of the normal test data in this model confirms that the model does not generalize well to the test data and is probably highly biased towards the training data distribution. On the other hand, in 1-HMM there is a single HMM that is trained with the whole training set. However, the same HMM needs to capture the distribution of the data coming from all APs. Since each AP has its own typical usage profile, these data distributions are different and one single HMM may not be sufficiently expressive to learn all of them correctly. MHMM and SpaMHMM combine the advantages and avoid the disadvantages of both previous models. Clearly, since the mixtures for each node share the same dictionary of HMMs, every model in the mixture is trained with sequences from all graph nodes, at least in the first few training iterations. Thus, at this stage, the models may capture behaviors that are shared by all APs. As mixtures become sparser during training, some components in the dictionary may specialize on the distribution of a few APs. This avoids the problem observed in 1-HMM, which is unaware of the AP where a sequence comes from. We would also expect SpaMHMM to be sparser and have better performance than MHMM, but only the former supposition was true (see Figure 2.7). The absence of performance gains in SpaMHMM might be explained from the fact that this dataset consists of simulated data, where users are static (i.e. they do not swap between APs unless the AP where they are connected stops working) and so the assumption that closer APs have similar distributions does not bring any advantage.

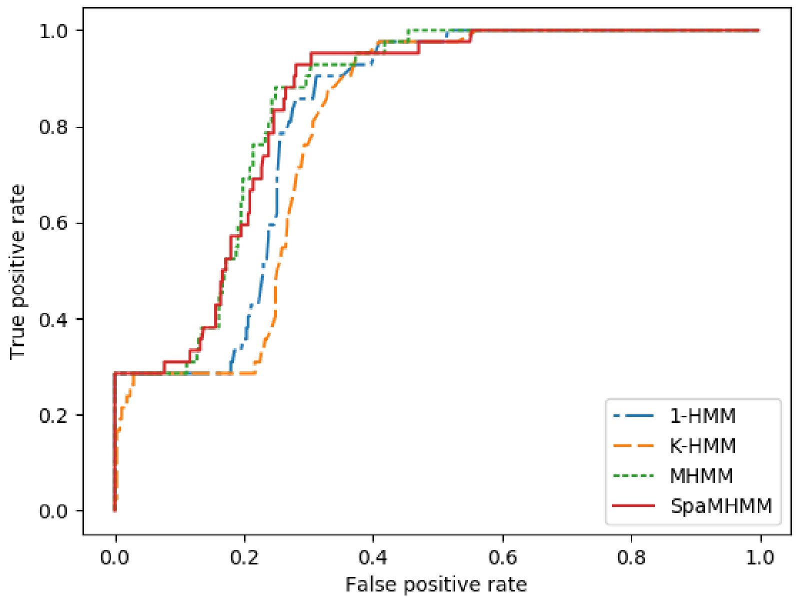

FIGURE 2.6: ROC curves for each model on the Wi-Fi dataset, for one of the 10 runs.

2.4.2 Human motion forecasting

The human body is constituted by several interdependent parts, which interact as a whole producing sensible global motion patterns. These patterns may correspond to multiple activities like walking, eating, etc. Here, we use our model to make short-time prediction of sequences of human joint positions, represented as motion capture (mocap) data. The

		Average log-likelihood	
	AUC	Normal data	Anomalous data
1-HMM	0.806 (±0.01)	−6.36 (±0.66)	−129.40 (±22.22)
k-HMM	0.776 (±0.01)	−22.09 (±1.12)	−**130.36** (±26.30)
MHMM	**0.830** (±0.01)	−3.31 (±0.21)	−10.99 (±1.10)
SpaMHMM	0.829 (±0.01)	−**3.26** (±0.12)	−11.29 (±1.39)

TABLE 2.2: AUC and average log-likelihood per sample for each model in the Wi-Fi dataset averaged over 10 training runs. Standard deviations are in brackets. Best results are in bold.

current state of the art methodologies use architectures based on deep recurrent neural networks (RNNs), achieving remarkable results both in short-time prediction Fragkiadaki et al. [42], Martinez et al. [43] and in long-term motion generation Jain et al. [44], Pavllo et al. [45].

Our experiments were conducted on the Human3.6M dataset from Ionescu et al. [46, 47], which consists of mocap data from 7 subjects performing 15 distinct actions. In this experiment, we have considered only 4 of those actions, namely "walking", "eating", "smoking" and "discussion". There, the human skeleton is represented with 32 joints whose position is recorded at 50 Hz. We build our 32x32-dimensional symmetric matrix $\boldsymbol{G}$ representing the graph $\mathcal{G}$ in the following sensible manner: $G_{y,y'} = 1$, if there is an actual skeleton connection between joints y and y' (e.g. the elbow joint is connected to the wrist joint by the forearm); $G_{y,y} = 1$, if joints y and y' are symmetric (e.g. left and right elbows); $G_{y,y'} = 0$, otherwise.

2.4.2.1 Forecasting

We reproduced as much as possible the experimental setup followed in Fragkiadaki et al. [42]. Specifically, we down-sampled the data by a factor of 2 and transformed the raw 3-D angles into an exponential map representation. We removed joints with constant exponential map, yielding a dataset with 22 distinct joints, and pruned our matrix $\boldsymbol{G}$ accordingly. Training was performed using data from 6 subjects, leaving one subject (denoted in the dataset by "S5") for testing. We did 3-fold cross-validation on the training data of the action "walking" to find the optimal number of mixture components m and hidden states s for the baseline mixture MHMM. Unsurprisingly, since this model can hardly overfit in such a complex task, we ended up with $m = 18$ and $s = 12$, which were the largest values in the ranges we defined. Larger values are likely to improve the results, but the training time would become too large to be practical. For SpaMHMM, we used these same values of m and s and we did 3-fold cross validation on the training data of the action "walking" to fine-tune the value of λ in the range $[10^{-4}, 1]$. We ended up using $\lambda = 0.05$. The number of hidden states in 1-HMM was set to 51 and in k-HMM it was set to 11 hidden states per HMM. The same values were then used to train the models for the remaining actions. Every model was trained for 100 iterations of EM or until the loss plateaus. For SpaMHMM, we did 100 iterations of the inner loop on each M-step, using a learning rate $\rho = 10^{-2}$.

In order to generate predictions for a joint (node) y starting from a given prefix sequence $\boldsymbol{X}_{\text{pref}}$, we compute the posterior distribution $p(\mathbf{X}|\boldsymbol{X}_{\text{pref}}, y)$ (see details in Section A.3) and we sample sequences from that posterior. Our evaluation method and metric again followed Fragkiadaki et al. [42]. We fed our model with 8 prefix subsequences with 50 frames each (corresponding to 2 seconds) for each joint from the test subject and we predicted the following 10 frames (corresponding to 400 miliseconds). Each prediction was built by sampling 100 sequences from the posterior and averaging. We then computed the average mean angle error for the 8 sequences at different time horizons.

Results are in Table 2.3. Among our models (1-HMM, k-HMM, MHMM and SpaMHMM), SpaMHMM outperformed the remaining in all actions except "eating". For this action in particular, MHMM was slightly better than SpaMHMM, probably due to the lack of symmetry between the right and left sides of the body, which was one of the prior assumptions that we have used to build the graph $\mathcal{G}$. "Smoking" and "discussion" activities may also be highly non-symmetric, but results in our and others' models show that these activities are generally harder to predict than "walking" and "eating'. Thus, here, the skeleton structure information encoded in $\mathcal{G}$ behaves as a useful prior for SpaMHMM, guiding it towards better solutions than MHMM. The worse results for 1-HMM and K-HMM likely result from the same limitations that we have pointed out in Section 2.4.1: each component in k-HMM is inherently trained with less data than the remaining models, while 1-HMM does not make distinction between different graph nodes. Extending the discussion to the state of the art solutions for this problem, we note that SpaMHMM compares favorably with ERD, LSTM-3LR and SRNN, which are all RNN-based architectures. Moreover, ERD and LSTM-3LR were designed specifically for this task, which is not the case for SpaMHMM. This is also true for GRU supervised and QuaterNet, which clearly outperform all remaining models, including ours. This is unsurprising, since RNNs are capable of modeling more complex dynamics than HMMs, due to their intrinsic non-linearity and continuous state representation. This also allows their usage for long-term motion generation, in which HMMs do not behave well due their linear dynamics and lack of long-term memory. However, unlike GRU supervised and QuaterNet, SpaMHMM models the probability distribution of the data directly, allowing its application in domains like novelty detection. Regarding sparsity, the experiments confirm that the SpaMHMM mixture coefficients are actually sparser than those of MHMM, as shown in Figure 2.7.

	Walking				Eating				Smoking				Discussion			
miliseconds	80	160	320	400	80	160	320	400	80	160	320	400	80	160	320	400
1-HMM	0.91	1.04	1.22	1.31	1.00	1.08	1.15	1.21	1.45	1.55	1.70	1.75	1.19	1.42	1.55	1.56
k-HMM	1.29	1.33	1.34	1.38	1.16	1.22	1.28	1.34	1.70	1.77	1.90	1.95	1.47	1.61	1.68	1.63
MHMM	**0.78**	**0.93**	1.13	1.21	**0.77**	**0.87**	**0.98**	**1.06**	1.44	1.53	1.69	1.77	1.14	1.36	1.52	1.54
SpaMHMM	0.80	**0.93**	**1.11**	**1.18**	0.81	0.90	0.99	**1.06**	**1.29**	**1.39**	**1.61**	**1.67**	**1.09**	**1.30**	**1.44**	**1.49**
ERD [42]	0.93	1.18	1.59	1.78	1.27	1.45	1.66	1.80	1.66	1.95	2.35	2.42	2.27	2.47	2.68	2.76
LSTM-3LR [42]	0.77	1.00	1.29	1.47	0.89	1.09	1.35	1.46	1.34	1.65	2.04	2.16	1.88	2.12	2.25	2.23
SRNN [44]	0.81	0.94	1.16	1.30	0.97	1.14	1.35	1.46	1.45	1.68	1.94	2.08	1.22	1.49	1.83	1.93
GRU sup. [43]	0.28	0.49	0.72	0.81	0.23	0.39	0.62	0.76	0.33	0.61	1.05	1.15	0.31	0.68	1.01	1.09
QuaterNet [45]	<u>0.21</u>	<u>0.34</u>	<u>0.56</u>	<u>0.62</u>	<u>0.20</u>	<u>0.35</u>	<u>0.58</u>	<u>0.70</u>	<u>0.25</u>	<u>0.47</u>	<u>0.93</u>	<u>0.90</u>	<u>0.26</u>	<u>0.60</u>	<u>0.85</u>	<u>0.93</u>

TABLE 2.3: Mean angle error for short-term motion prediction on Human3.6M for different actions and time horizons. The results for ERD, LSTM-3LR, SRNN, GRU supervised and QuaterNet were extracted from Pavllo et al. [45]. Best results among our models are in bold, best overall results are underlined.

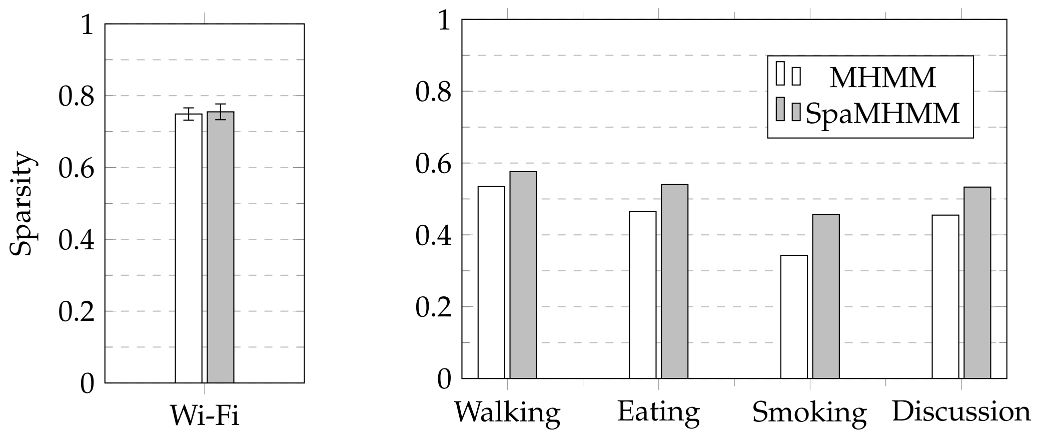

FIGURE 2.7: Relative sparsity (number of coefficients equal to zero / total number of coefficients) of the obtained MHMM and SpaMHMM models on the Wi-Fi dataset (left) and on the Human3.6M dataset for different actions (right). For the Wi-Fi dataset, the average value over the 10 training runs is shown together with the standard deviation. Both models for the Wi-Fi dataset have 150 coefficients. All models for the Human3.6M dataset have 396 coefficients.

2.4.2.2 Joint cluster analysis

We may roughly divide the human body in four distinct parts: upper body (head, neck and shoulders), arms, torso and legs. Joints that belong to the same part naturally tend to have coherent motion, so we would expect them to be described by more or less the same components in our mixture models (MHMM and SpaMHMM). Since SpaMHMM is trained to exploit the known skeleton structure, this effect should be even more apparent in SpaMHMM than in MHMM. In order to confirm this conjecture, we have trained MHMM and SpaMHMM for the action "walking" using four mixture components only, i.e. $m = 4$, and we have looked for the most likely component (cluster) for each joint:

$$C_y \triangleq \underset{z \in \{1,\dots,m\}}{\arg\max}\, p(z \mid y) = \underset{z \in \{1,\dots,m\}}{\arg\max}\, \alpha_z^{(y)}, \tag{2.25}$$

where C_y is, therefore, the cluster assigned to joint y. The results are in Figure 2.8. From there we can see that MHMM somehow succeeds on dividing the body in two main parts, by assigning the joints in the torso and in the upper body mostly to the red/'+' cluster, while those in the hips, legs and feet are almost all assigned to the green/'$\triangle$' cluster. Besides, we see that in the vast majority of the cases, symmetric joints are assigned to the same cluster. These observations confirm that we have chosen the graph $\mathcal{G}$ for this problem in an appropriate manner. However, some assignments are unnatural: e.g. one of the joints in the left foot is assigned to the red/'+' cluster and the blue/'$\circ$' cluster is assigned to one single joint, in the left forearm. We also observe that the distribution of joints per clusters is highly uneven, being the green/'$\triangle$' cluster the most represented by far. SpaMHMM, on the other hand, succeeds on dividing the body in four meaningful regions: upper body and upper spine in the green/'$\triangle$' cluster; arms in the blue/'$\circ$' cluster; lower spine and hips in the orange/'x' cluster; legs and feet in the red/'+' cluster. Note that the graph $\mathcal{G}$ used to regularize SpaMHMM does not include any information about the body part that a joint belongs to, but only about the joints that connect to it and that are symmetric to it. Nevertheless, the model is capable of using this information together with the training data in order to divide the skeleton in an intuitive and natural way. Moreover, the distribution of joints per cluster is much more even in this case, what may also help to explain why SpaMHMM outperforms MHMM: by splitting the joints more or less evenly by the different HMMs in the mixture, none of the HMM components is forced to learn too many motion patterns. In MHMM, we see that the green/'+' component, for instance, is the most responsible to model the motion of almost all joints in the legs and hips and also some joints in the arms and the red/'+' component is the prevalent on the prediction of the motion patterns of the neck and left foot, which are presumably very different.

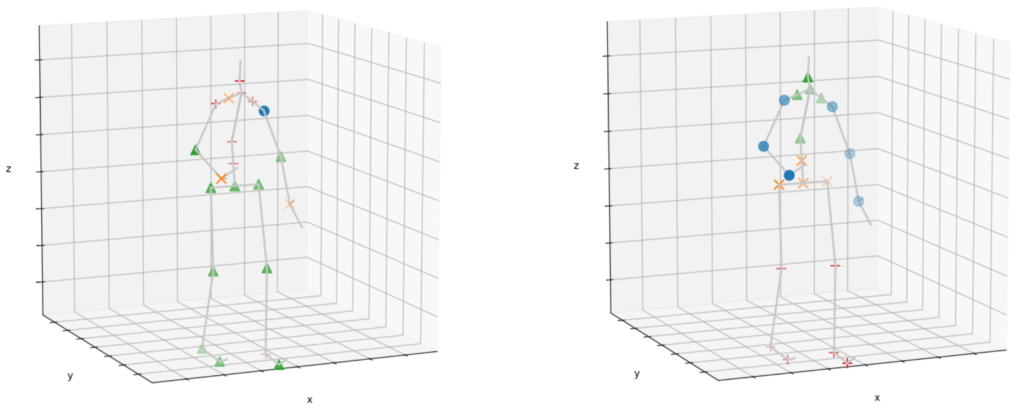

FIGURE 2.8: Assignments of joints to clusters in MHMM (left) and SpaMHMM (right). The different colors (blue, green, orange, red) and the respective symbols ('○', '△', 'x', '+') on each joint represent the cluster that the joint was assigned to. on each joint represent the cluster that the joint was assigned to.

2.4.3 Conclusion

In this work we propose a method to model the generative distribution of sequential data coming from nodes connected in a graph with a known fixed topology. The method is based on a mixture of HMMs where its coefficients are regularized during the learning process in such a way that affine nodes will tend to have similar coefficients, exploiting the known graph structure. We also prove that pairwise optimization of the coefficients leads to sparse mixtures. Experimental results suggest that sparsity holds in the general case. We evaluate the method performance in two completely different tasks (anomaly detection in Wi-Fi networks and human motion forecasting), showing its effectiveness and versatility.

2.5 Summary and directions for future work

After a detailed presentation of SOHMMM and SpaMHMM algorithms in sections 2.2 and 2.3, respectively, it is instructive to offer a brief comparative overview of the two models. Additionally, we shall discuss how these frameworks could be extended and generalized.

2.5.1 SOHMMM vs. SpaMHMM

The similarities between SOHMMM and SpaMHMM go beyond the obvious fact that both models use the hidden Markov model as their core component. It should be noted that equations (2.3) and (2.18), which govern the learning dynamics of these models, are both structurally similar to equation (2.1) and, therefore, similar to each other. Despite the fact that the former is an energy function and the latter is a proper density, the SOHMMM energy (2.3) could also be transformed into a density by normalization:

$$p(\mathbf{X};\Theta) = \frac{E(\mathbf{X};\Theta)}{Z(\Theta)}, \quad \text{where} \quad Z(\Theta) = \int E(\mathbf{X};\Theta)d\mathbf{X}. \tag{2.26}$$

This observation makes it clear that, in the learning stage, SOHMMM can also be regarded as a mixture of HMMs.

However, major differences arise when we examine how the two models associate entities y with atoms (i.e. HMMs) z. SOHMMM does not include any explicit dependency on y since there is a one-to-one correspondence between entities and atoms in the lattice (e.g. in the experiment with wireless simulation data, each AP is modeled by one HMM and there are not two APs sharing the same HMM). Moreover, for the mixture model to exist, the topology of the lattice must be known or at least a distance must be defined between each pair of nodes. On the other hand, in SpaMHMM, the mixture is defined by an explicit many-to-many relationship between entities and atoms, defined by the distribution $p(\mathrm{z} \mid \mathrm{y})$. As a result, the model for each entity is a mixture of HMMs and nothing impedes two distinct entities of sharing the same HMM, although possibly assigned to different mixture weights. Thus, SpaMHMM is a more expressive model than SOHMMM, which comes at the cost of having a few extra parameters to model $p(\mathrm{z} \mid \mathrm{y})$. This would also allow new entities to be added to the model in a simple manner and without affecting the remaining entities models, as will be discussed in section 2.5.5. Additionally, in SpaMHMM, the graph structure is used as a regularization term only, so, in case this information is absent, the model can still be learned and the correlations between multiple entities would still be exploited, although no prior information about them would be provided to the learning algorithm.

Another apparent difference between the two models is the fact that SOHMMM was conceived to be learned online and SpaMHMM was presented with an offline learning algorithm. These were mere design choices, though. For instance, it is straightforward to replace SOHMMM stochastic gradient descent (SGD) algorithm with a batch version and SGD for SpaMHMM could also be derived following the same principles that were used in its derivation for SOHMMM.

2.5.2 Generalizing SpaMHMM

Two obvious limitations of SpaMHMM are its inherent dependency on hidden Markov models, which have limited expressiveness, and the fact that its learning algorithm was conceived to an offline and non-distributed setting. In this section, we discuss how these two drawbacks could be overcome by an extended and generalized framework, of which SpaMHMM is a particular case. Specifically, we consider again the setting where each entity y in a set $\mathbb{Y}$ produces sequences $\mathbf{X} = \left(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(t)}\right)$. The prior knowledge about the pairwise affinity degrees between entities in $\mathbb{Y}$ at time t is summarized in an undirected weighted graph $\mathcal{G}$, where each entity corresponds to a graph node. The ultimate goal is to find the distribution of the observed sequences given the corresponding node, that is, to model the generative distribution $p(\mathbf{X} \mid \mathrm{y})$. For this purpose, we shall consider models of the form:

$$p(\mathbf{X} \mid \mathrm{y}) = \int_{\mathcal{Z}} p(\mathbf{X} \mid \mathbf{z}) p(\mathbf{z} \mid \mathrm{y}) d\mathbf{z}. \tag{2.27}$$

This model comprises a latent space $\mathcal{Z}$ that is shared by all entities and where all observations $\mathbf{X}$ are produced, according to a *conditional observation density* $p(\mathbf{X} \mid \mathrm{z})$. Through the *latent density* $p(\mathrm{z} \mid \mathrm{y})$, each entity chooses regions of the latent space that are more likely to explain its own sequences. While not specifying the structure of both $p(\mathbf{X} \mid \mathrm{z})$ and $p(\mathrm{z} \mid \mathrm{y})$, this is a meta-model that makes no assumptions on the nature of the observed data streams. However, depending on those choices, exact inference on the resulting model may or may not be a tractable problem.

Clearly, the SpaMHMM model is a particular case of equation (2.27), where the latent space $\mathcal{Z}$ is discrete and finite and the latent generative model is an HMM parametrized by $\mathbf{z}$. The discrete nature of $\mathcal{Z}$ and the underlying independence assumptions of the HMM allow for efficient exact inference in the resulting model using the Forward algorithm [36].

An interesting, more expressive, and still unexplored possibility would be to implement the conditional observation model $p(\mathbf{X} \mid \mathrm{z})$ through a recurrent mixture density network (RMDN) [48–50]. An RMDN consists of a recurrent neural network (RNN) whose outputs at each time step parameterize a Gaussian mixture model with c components:

$$p(\boldsymbol{X} \mid \boldsymbol{z}) = \prod_{\tau=1}^{t} p(\boldsymbol{x}^{(\tau)} \mid \boldsymbol{x}^{(\tau)}, ..., \boldsymbol{x}^{(\tau-1)}, \boldsymbol{z}) = \prod_{\tau} \sum_{j=1}^{c} \alpha_j^{(t)} \mathcal{N}(\boldsymbol{x}^{(t)}; \boldsymbol{\mu}_j^{(t)}, \Sigma_j^{(t)}), \tag{2.28}$$

$$\{\alpha_j^{(t)}, \boldsymbol{\mu}_j^{(t)}, \Sigma_j^{(t)}\}_{j=1}^{c} = \mathrm{RNN}(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(t-1)}; \boldsymbol{z}). \tag{2.29}$$

The latent space may be discrete, in which case we shall end up with a discrete mixture of RMDNs, where exact inference is straightforward. Another possibility is to choose a continuous latent space and, in such scenario, a sensible option is using a Gaussian density for the latent density model:

$$p(z \mid y) = \mathcal{N}(z; \boldsymbol{\mu}_y, \Sigma_y). \tag{2.30}$$

Under this setting, computing equation (2.27) becomes infeasible and, hence, exact inference is not possible. However, the marginals $p(\boldsymbol{X} \mid y)$ may be approximated using a Monte Carlo estimation:

$$p(\boldsymbol{X} \mid y) \approx \frac{1}{n} \sum_{i=1}^{n} p(\boldsymbol{X} \mid z_i), \tag{2.31}$$

where the z_i are sampled from $p(\mathbf{z} \mid y)$ and n is the desired number of samples, which should be sufficiently large for the estimator to have a small variance.

Although very different in terms of their internal structure, all aforementioned models share the external structure defined by equation (2.27). In the forthcoming sections, we outline how training and inference could be done in this broad family of models under multiple settings.

2.5.3 Offline, centralized learning

It is reasonable to assume that an initial dataset composed by sequences from all entities may be gathered and used to learn an initial model for all entities. This model could then be refined online in each instance using a distributed algorithm. Hence, it makes sense to derive a algorithm to learn the meta-model defined by equation (2.27) in an offline and centralized setting. This is the idea we followed in [2], which solves this problem for the particular of the SpaMHMM model using EM. This algorithm is based on the alternated maximization over variational densities $q(\mathbf{z})$ and model parameters of an evidence lower bound (ELBO):

$$\log p(\mathbf{X} \mid \mathrm{y}) \geq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log p(\mathbf{X}, \mathbf{z} \mid \mathrm{y})\right] - \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log q(\mathbf{z})\right] \triangleq \mathrm{ELBO}. \tag{2.32}$$

For fixed model parameters, maximization of the ELBO with respect to $q(\mathbf{z})$ is attained when this distribution matches the true posterior $p(\mathbf{z} \mid \mathbf{X}, \mathrm{y})$. For the case of SpaMHMM, computing this posterior is a tractable problem. When this is not the case, some options are constraining the variational density to have a simpler, factorizable form (*e.g.* mean-field approximation [51]) or approximating the posterior with a parametric model, like a deep neural network (*e.g.* variational auto-encoders [52]).

In any of those cases, leveraging the prior contextual information represented by the graph $\mathcal{G}$ through the introduction of a regularizer term like in equation (2.21) poses no significant additional difficulties to the optimization algorithm.

2.5.4 Online, centralized learning

Assuming that a few new training examples $(\boldsymbol{X}_i, y_i)$ are periodically uploaded to a central unit and that entities are capable of downloading updated model parameters from that unit, being able to update both the conditional observation model $p(\mathbf{X} \mid \mathbf{z})$ and the latent density model $p(\mathbf{z} \mid \mathrm{y})$ is a sensible goal.

Incremental training of HMMs has been widely treated in previous works. Existing approaches fall into one of three categories: direct gradient-based optimization over the log-likelihood objective [20], incremental versions of EM [53, 54], and recursive maximum likelihood estimation [55]. An overview of these methods may be found in [27]. Both gradient-based optimization and incremental EM may be extended to SpaMHMM, making it suitable to an online learning setting. More complex models, particularly those based in neural networks, are typically trained using stochastic gradient descent (or any of its many variants), which is inherently applicable in an online setting. If an updated graph $\mathcal{G}$ is available, this information can also be incorporated in the learning objective as before.

2.5.5 Online, distributed learning

We now consider a scenario where each entity aims to update its own model autonomously, that is, without observing any information from the remaining entities. Therefore, we assume y is fixed to some $y \in \mathbb{Y}$ and, therefore, all new training examples are of the form $(\boldsymbol{X}_i, y)$.

This particular setting is favorably accommodated by the structure of equation (2.27). By comprising a shared (and desirably rich) conditional observation model $p(\mathbf{X} \mid \mathrm{z})$ and a (presumably simpler) entity-dependent latent density model $p(\mathrm{z} \mid \mathrm{y})$, an entity y may adapt its own model $p(\mathbf{X} \mid y)$ without changing the model $p(\mathbf{X} \mid y)$ for any other entity $y' \neq y$. The idea here is to freeze the parameters of the conditional observation model and update only the latent density model $p(z \mid y)$ to accommodate the new data. By doing this, the entity y is performing a search over the latent space $\mathcal{Z}$ to find regions where the new observations have higher likelihood. If the resulting model is still unsatisfactory, this likely means that the shared latent space was still not diverse enough to explain the new data. In this circumstance, the entity may decide to upload its observations $(\boldsymbol{X}_i, y)$ to a central unit, which in turn may update the whole model as outlined in Section 2.5.4.

Appendix A

SpaMHMM – supplementary material

A.1 Derivation of Algorithm 3 (MHMM training)

Algorithm 3 follows straightforwardly from applying EM to the model defined by equations (2.18) and (2.19) with the objective (2.20). Let us define the following notation: $\mathbf{X} := \{X_i\}_{i=1}^N$, $\boldsymbol{y} := \{y_i\}_{i=1}^N$, $\mathbf{z} := \{z_i\}_{i=1}^N$ and $\mathbf{H} := \{\mathbf{h}_i\}_{i=1}^N$. After building the usual variational lower bound for the log-likelihood and performing the E-step, we get the following well-known objective:

$$\tilde{J}(\Theta, \Theta^{(-)}) \triangleq \sum_{\mathbf{z},\mathbf{H}} p(\mathbf{X}, \mathbf{z}, \mathbf{H} \mid \boldsymbol{y}, \Theta^{(-)}) \log p(\mathbf{X}, \mathbf{z}, \mathbf{H} \mid \boldsymbol{y}, \Theta), \tag{A.1}$$

which we want to maximize with respect to Θ and where $\Theta^{(-)}$ are the model parameters that were kept fixed in the E-step. Some of the parameters in the model are constrained to represent valid probabilities, yielding the following Lagrangian:

$$\begin{aligned} L(\Theta, \Theta^{(-)}, \Lambda) = & \tilde{J}(\Theta, \Theta^{(-)}) + \sum_k \lambda_k^{\text{mix}} \left(1 - ||\boldsymbol{\alpha}_k||_1\right) \\ & + \sum_{m,s} \lambda_{m,s}^{\text{state}} \left(1 - \sum_u A_{s,u}^m\right) \\ & + \sum_m \lambda_m^{\text{ini}} \left(1 - ||\boldsymbol{\pi}_m||_1\right), \end{aligned} \tag{A.2}$$

where Λ summarizes all Lagrange multipliers used here. Differentiating equation (A.2) with respect to each model parameter and Lagrange multiplier and solving for the critical points yields:

$$\alpha_{k,m} = \frac{\sum_i p(\mathrm{z}_i = m | \mathbf{X}_i, y_i, \Theta^{(-)}) \mathbf{1}_{y_i=k}}{\sum_i \mathbf{1}_{y_i=k}}, \tag{A.3}$$

$$\pi_{m,s} = \frac{\sum_i p(\mathrm{z}_i = m \mid \mathbf{X}_i, y_i, \Theta^{(-)}) p(\mathrm{h}_i^{(0)} = s \mid \mathrm{z}_i = m, \mathbf{X}_i, y_i, \Theta^{(-)})}{\sum_i p(\mathrm{z}_i = m \mid \mathbf{X}_i, y_i, \Theta^{(-)})}, \tag{A.4}$$

$$A^m_{s,u} = \frac{\sum_i p(\mathrm{z}_i = m \mid \mathbf{X}_i, y_i, \Theta^{(-)}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t-1)} = s, \mathrm{h}_i^{(t)} = u \mid \mathrm{z}_i = m, \mathbf{X}_i, y_i, \Theta^{(-)})}{\sum_i p(\mathrm{z}_i = m \mid \mathbf{X}_i, y_i, \Theta^{(-)}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t-1)} = s \mid \mathrm{z}_i = m, \mathbf{X}_i, y_i, \Theta^{(-)})}, \tag{A.5}$$

$$\boldsymbol{\mu}_{m,s} = \frac{\sum_i p(\mathrm{z}_i = m \mid \mathbf{X}_i, y_i, \Theta^{(-)}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t)} = s \mid \mathrm{z}_i = m, \mathbf{X}_i, y_i, \Theta^{(-)}) \boldsymbol{x}_i^{(t)}}{\sum_i p(\mathrm{z}_i = m \mid \mathbf{X}_i, y_i, \Theta^{(-)}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t)} = s \mid \mathrm{z}_i = m, \mathbf{X}_i, y_i, \Theta^{(-)})}, \tag{A.6}$$

$$\boldsymbol{\sigma}^2_{m,s} = \frac{\sum_i p(\mathrm{z}_i = m \mid \mathbf{X}_i, y_i, \Theta^{(-)}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t)} = s \mid \mathrm{z}_i = m, \mathbf{X}_i, y_i, \Theta^{(-)}) \left(\boldsymbol{x}_i^{(t)} - \boldsymbol{\mu}_s^m\right)^2}{\sum_i p(\mathrm{z}_i = m \mid \mathbf{X}_i, y_i, \Theta^{(-)}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t)} = s \mid \mathrm{z}_i = m, \mathbf{X}_i, y_i, \Theta^{(-)})}, \tag{A.7}$$

$$\forall\, k, m, s, u.$$

Defining n_k, $\rho_{i,m}$, $\gamma_{i,m,s}$ and $\xi_{i,m,s,u}$ as in Algorithm 3 the result follows.

A.2 Derivation of Algorithm 4 (SpaMHMM training)

Using the same notation as in Section A.1, we may rewrite equation (**??**) as:

$$\begin{aligned} J_r(\Theta) =& \frac{1}{N} \log \sum_{\mathbf{z}, \mathbf{H}} p(\mathbf{X}, \mathbf{z}, \mathbf{H} \mid \boldsymbol{y}, \Theta) \\ &+ \frac{\lambda}{2} \sum_{j, k \neq j} G_{j,k} \mathbb{E}_{\mathrm{z} \sim p(\mathrm{z} | \mathrm{y} = j, \Theta)} [p(\mathrm{z} \mid \mathrm{y} = k, \Theta)]. \end{aligned} \tag{A.8}$$

Despite the regularization term, we may still lower bound this objective by introducing a variational distribution $q(\mathbf{z}, \mathbf{H})$ and using Jensen's inequality in the usual way:

$$\begin{aligned} J_r(\Theta) \geq & \frac{1}{N} \mathbb{E}_{\mathbf{z}, \mathbf{H} \sim q} \left[\log \frac{p(\mathbf{X}, \mathbf{z}, \mathbf{H} \mid \boldsymbol{y}, \Theta)}{q(\mathbf{z}, \mathbf{H})} \right] \\ &+ \frac{\lambda}{2} \sum_{j, k \neq j} G_{j,k} \mathbb{E}_{\mathrm{z} \sim p(\mathrm{z} | \mathrm{y} = j, \Theta)} [p(\mathrm{z} \mid \mathrm{y} = k, \Theta)] \\ :=& V_r(\Theta, q). \end{aligned} \tag{A.9}$$

Clearly,

$$\begin{aligned}
&J_r(\Theta) - V_r(\Theta, q) = \\
&= \frac{1}{N}\left(\log p(\mathbf{X}|\boldsymbol{y}, \Theta) - \mathbb{E}_{\mathbf{z},\mathbf{H}\sim q}\left[\log \frac{p(\mathbf{X}, \mathbf{z}, \mathbf{H} \mid \boldsymbol{y}, \Theta)}{q(\mathbf{z}, \mathbf{H})}\right]\right) \\
&= \frac{1}{N} D_{\mathrm{KL}}\left(q(\mathbf{z}, \mathbf{H}) || p(\mathbf{z}, \mathbf{H} \mid \mathbf{X}, \boldsymbol{y}, \Theta)\right), \qquad \text{(A.10)}
\end{aligned}$$

which, fixing the parameters Θ to some value $\Theta^{(-)}$ and minimizing with respect to q, yields the usual solution $q^*(\mathbf{z}, \mathbf{H}) = p(\mathbf{z}, \mathbf{H} \mid \mathbf{X}, \boldsymbol{y}, \Theta^{(-)})$. Thus, in the M-step, we want to find:

$$\begin{aligned}
&\arg\max_{\Theta} V_r(\Theta, q^*) = \\
&= \arg\max_{\Theta} \frac{1}{N} \sum_{\mathbf{z},\mathbf{H}} p(\mathbf{X}, \mathbf{z}, \mathbf{H} \mid \boldsymbol{y}, \Theta^{(-)}) \log p(\mathbf{X}, \mathbf{z}, \mathbf{H} \mid \boldsymbol{y}, \Theta) \\
&\quad + \frac{\lambda}{2} p(\mathbf{X}|\boldsymbol{y}, \Theta^{(-)}) \sum_{j,k \neq j} G_{j,k} \mathbb{E}_{\mathrm{z}\sim p(\mathrm{z}|\mathrm{y}=j,\Theta)}[p(\mathrm{z} \mid \mathrm{y} = k, \Theta)] \\
&= \arg\max_{\Theta} \frac{1}{N} \tilde{J}(\Theta, \Theta^{(-)}) + \lambda R(\Theta, \Theta^{(-)}) \\
&:= \arg\max_{\Theta} \tilde{J}_r(\Theta, \Theta^{(-)}), \qquad \text{(A.11)}
\end{aligned}$$

where $\tilde{J}(\Theta, \Theta^{(-)})$ is as defined in equation (A.1) and $R(\Theta, \Theta^{(-)})$ is our regularization (weighted by the data likelihood), which is simply a function of the parameters $\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K$:

$$\begin{aligned}
R(\Theta, \Theta^{(-)}) &= \frac{1}{2} p(\mathbf{X}|\boldsymbol{y}, \Theta^{(-)}) \sum_{j,k \neq j} G_{j,k} \mathbb{E}_{\mathrm{z}\sim p(\mathrm{z}|\mathrm{y}=j,\Theta)}[p(\mathrm{z} \mid \mathrm{y} = k, \Theta)] \\
&= \frac{1}{2} p(\mathbf{X}|\boldsymbol{y}, \Theta^{(-)}) \sum_{j,k \neq j} G_{j,k} \boldsymbol{\alpha}_j^\top \boldsymbol{\alpha}_k \\
&= R(\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K, \Theta^{(-)}). \qquad \text{(A.12)}
\end{aligned}$$

Now, we may build the Lagrangian as done in Section A.1. Since R only depends on the $\boldsymbol{\alpha}$'s, the update equations for the remaining parameters are unchanged. However, for the $\boldsymbol{\alpha}$'s, it is not possible to obtain a closed form update equation. Thus, we use the reparameterization defined in equation (2.24) and update the new unconstrained parameters $\boldsymbol{\beta}$ via gradient ascent.

We have:

$$\frac{\partial \tilde{J}}{\partial \alpha_{k,m}} = \frac{p(\mathbf{X}|\boldsymbol{y}, \Theta^{(-)})}{\alpha_{k,m}} \sum_i p(z_i = m \mid \boldsymbol{X}_i, y_i, \Theta^{(-)}) \mathbf{1}_{y_i=k}, \tag{A.13}$$

$$\frac{\partial R}{\partial \alpha_{k,m}} = p(\mathbf{X}|\boldsymbol{y}, \Theta^{(-)}) \sum_{j \neq k} G_{j,k} \alpha_{j,m}. \tag{A.14}$$

From equations (A.13) and (A.14), we see that the the resulting gradient $\nabla_{\boldsymbol{\alpha}_k} \tilde{J}_r = \frac{1}{N} \nabla_{\boldsymbol{\alpha}_k} \tilde{J} + \lambda \nabla_{\boldsymbol{\alpha}_k} R$ is equal to some vector scaled by the joint data likelihood $p(\mathbf{X}|\boldsymbol{y}, \Theta^{(-)})$, which we discard since it only affects the learning rate, besides being usually very small and somewhat costly to compute. This option is equivalent to using a learning rate that changes at each iteration of the outter loop of the algorithm.

Equation (2.24) yields the following derivatives:

$$\frac{\partial \alpha_{k,m}}{\partial \beta_{k,m}} = \mathbf{1}_{\beta_{k,m}>0} \frac{2\sigma'(\beta_{k,m})}{\sigma(\beta_{k,m})} \alpha_{k,m}(1 - \alpha_{k,m}), \tag{A.15}$$

$$\frac{\partial \alpha_{k,m}}{\partial \beta_{k,l}} = \mathbf{1}_{\beta_{k,m}>0} \frac{-2\sigma'(\beta_{k,m})}{\sigma(\beta_{k,m})} \alpha_{k,m} \alpha_{k,l}, \text{ for } l \neq m. \tag{A.16}$$

Finally, by the chain rule, we obtain:

$$\begin{aligned} \frac{\partial \tilde{J}}{\partial \beta_{k,m}} &= \sum_l \frac{\partial \tilde{J}}{\partial \alpha_{k,l}} \frac{\partial \alpha_{k,l}}{\partial \beta_{k,m}} \\ &= \mathbf{1}_{\beta_{k,m}>0} \frac{2\sigma'(\beta_{k,m})}{\sigma(\beta_{k,m})} \sum_i \left(p(z_i = m \mid \boldsymbol{X}_i, y_i, \Theta^{(-)}) - \alpha_{k,m} \right) \mathbf{1}_{y_i=k}, \end{aligned} \tag{A.17}$$

$$\begin{aligned} \frac{\partial R}{\partial \beta_{k,m}} &= \sum_l \frac{\partial R}{\partial \alpha_{k,l}} \frac{\partial \alpha_{k,l}}{\partial \beta_{k,m}} \\ &= \mathbf{1}_{\beta_{k,m}>0} \frac{2\sigma'(\beta_{k,m})}{\sigma(\beta_{k,m})} \alpha_{k,m} \sum_{j \neq k} G_{j,k} \left(\alpha_{j,m} - \boldsymbol{\alpha}_j^\top \boldsymbol{\alpha}_k \right). \end{aligned} \tag{A.18}$$

Defining $\delta_{k,m} := \frac{\partial \tilde{J}_r}{\partial \beta_{k,m}} = \frac{1}{N} \frac{\partial \tilde{J}}{\partial \beta_{k,m}} + \lambda \frac{\partial R}{\partial \beta_{k,m}}$ and applying the gradient ascent update formula to $\beta_{k,m}$ the result follows.

A.3 Getting the posterior distribution of observations in SpaMHMM

In this section, we show how to obtain the posterior distribution $p(\mathbf{X} \mid X_{\text{pref}}, y)$ of sequences $\mathbf{X} \triangleq \left(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(t)}\right)$ given an observed prefix sequence $X_{\text{pref}} \triangleq \left(x^{(-t_{\text{pref}}+1)}, ..., x^{(0)}\right)$, both coming from the graph node y. We start by writing this posterior as a marginalization with respect to the latent variable z:

$$\begin{aligned} p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, y) &= \sum_{\mathrm{z}} p(\mathbf{X}, \mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y) \\ &= \sum_{\mathrm{z}} p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, y, \mathrm{z}) p(\mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y) \\ &= \sum_{\mathrm{z}} p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) p(\mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y), \end{aligned} \tag{A.19}$$

where the last equality follows from the fact that the observations **X** are conitionally independent of the graph node y given the latent variable z. The posterior $p(\mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y)$ may be obtained as done in Algorithm 3:

$$p(\mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y) \propto p(\boldsymbol{X}_{\text{pref}} \mid z) p(z \mid y). \tag{A.20}$$

We now focus on the computation of $p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z})$. Let $\mathbf{h}_{\text{pref}} \triangleq \left(\mathrm{h}^{(-t_{\text{pref}}+1)}, ..., \mathrm{h}^{(-1)}\right)$ and $\mathbf{h} \triangleq \left(\mathrm{h}^{(0)}, ..., \mathrm{h}^{(t)}\right)$, then:

$$\begin{aligned} p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) &= \sum_{\mathbf{h}_{\text{pref}}, \mathbf{h}} p(\mathbf{X}, \mathbf{h}_{\text{pref}}, \mathbf{h} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \\ &= \sum_{\mathbf{h}_{\text{pref}}, \mathbf{h}} p(\mathbf{X} \mid \mathbf{h}_{\text{pref}}, \mathbf{h}, \boldsymbol{X}_{\text{pref}}, \mathrm{z}) p(\mathbf{h}_{\text{pref}}, \mathbf{h} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \\ &= \sum_{\mathbf{h}_{\text{pref}}, \mathbf{h}} p(\mathbf{X} \mid \mathbf{h}, \mathrm{z}) p(\mathbf{h}_{\text{pref}} \mid \mathbf{h}, \boldsymbol{X}_{\text{pref}}, \mathrm{z}) p(\mathrm{h} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \\ &= \sum_{\mathbf{h}} p(\mathbf{X} \mid \mathbf{h}, \mathrm{z}) p(\mathbf{h} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \\ &= \sum_{\mathbf{h}} p(\mathrm{h}^{(0)} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \prod_{\tau=1}^{t} p(\mathrm{h}^{(\tau)} \mid \mathrm{h}^{(\tau-1)}, \mathrm{z}) p(\mathbf{x}^{(\tau)} \mid \mathrm{h}^{(\tau)}, z), \end{aligned} \tag{A.21}$$

where we have used the independence assumptions that characterize the HMM. Here, the initial state posteriors $p(\mathrm{h}^{(0)} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z})$ are actually the final state posteriors for the sequence $\boldsymbol{X}_{\text{pref}}$ for each HMM in the mixture, so they can also be computed as indicated Algorithm 3.

Thus, we see that, in order to obtain the posterior $p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, y)$, we only need to update the mixture coefficients $p(\mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y)$ and the initial state probabilities $p(\mathrm{h}^{(0)} \mid \mathrm{z}, \boldsymbol{X}_{\text{pref}})$. All remaining parameters are unchanged.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org. [Cited on page xv.]

[2] D. Pernes and J. S. Cardoso, "SpaMHMM: Sparse mixture of hidden Markov models for graph connected entities," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–10. [Cited on pages 3 and 40.]

[3] A. Allahdadi, D. Pernes, J. S. Cardoso, and R. Morla, "Hidden Markov models on a self-organizing map for anomaly detection in 802.11 wireless networks," *Neural Computing and Applications*, pp. 1–18, 2021. [Cited on pages 3 and 20.]

[4] F. De Vico Fallani, J. Richiardi, M. Chavez, and S. Achard, "Graph analysis of functional brain networks: practical issues in translational neuroscience," *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, vol. 369, no. 1653, 2014. [Online]. Available: http://rstb.royalsocietypublishing.org/content/369/1653/20130521 [Cited on page 3.]

[5] M. R. Tora, J. Chen, and J. J. Little, "Classification of puck possession events in ice hockey," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, July 2017, pp. 147–154. [Cited on page 3.]

[6] R. Theagarajan, F. Pala, X. Zhang, and B. Bhanu, "Soccer: Who has the ball? generating visual analytics and player statistics," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018. [Cited on page 3.]

[7] N. Cheng, F. Lyu, J. Chen, W. Xu, H. Zhou, S. Zhang, and X. S. Shen, "Big data driven vehicular networks," *IEEE Network*, vol. 32, no. 6, pp. 160–167, November 2018. [Cited on page 3.]

[8] J. Gama and M. M. Gaber, *Learning from data streams: processing techniques in sensor networks*. Springer, 2007. [Cited on page 3.]

[9] S. Laxman, V. Tankasali, and R. W. White, "Stream prediction using a generative model based on frequent episodes in event sequences," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2008, pp. 453–461. [Cited on page 4.]

[10] M. Z. Hayat and M. R. Hashemi, "A dct based approach for detecting novelty and concept drift in data streams," in *2010 International Conference of Soft Computing and Pattern Recognition*, Dec 2010, pp. 373–378. [Cited on page 4.]

[11] A. Hofmann and B. Sick, "Online intrusion alert aggregation with generative data stream modeling," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 2, pp. 282–294, 2011. [Cited on page 4.]

[12] D. Baron, M. F. Duarte, M. B. Wakin, S. Sarvotham, and R. G. Baraniuk, "Distributed compressive sensing," *CoRR*, vol. abs/0901.3403, 2009. [Online]. Available: http://arxiv.org/abs/0901.3403 [Cited on page 4.]

[13] A. Allahdadi, R. Morla, and J. S. Cardoso, "802.11 wireless simulation and anomaly detection using HMM and UBM," *SIMULATION*, vol. 96, no. 12, pp. 939–956, 2020. [Online]. Available: https://doi.org/10.1177/0037549720958480 [Cited on pages 5, 9, 17, 18, 19, 30, and 31.]

[14] P. Somervuo, "Competing hidden Markov models on the self-organizing map," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 3. IEEE, 2000, pp. 169–174. [Cited on page 6.]

[15] M. Kurimo and P. Somervuo, "Using the self-organizing map to speed up the probability density estimation for speech recognition with mixture density HMMs," in *Spoken Language, 1996. ICSLP 96. Proceedings, Fourth International Conference on*, vol. 1. IEEE, 1996, pp. 358–361.

[16] H. Morimoto, "Hidden Markov models and self-organizing maps applied to stroke incidence," *Open Journal of Applied Sciences*, vol. 6, no. 3, pp. 158–168, 2016. [Cited on pages 6 and 7.]

[17] C. Ferles and A. Stafylopatis, “Self-organizing hidden Markov model map (SOHMMM),” *Neural Networks*, vol. 48, pp. 133 – 147, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608013001974 [Cited on pages 6 and 8.]

[18] C. Ferles, G. Siolas, and A. Stafylopatis, “Scaled self-organizing map–hidden Markov model architecture for biological sequence clustering,” *Applied Artificial Intelligence*, vol. 27, no. 6, pp. 461–495, 2013.

[19] M. Lebbah, R. Jaziri, Y. Bennani, and J.-H. Chenot, “Probabilistic self-organizing map for clustering and visualizing non-IID data,” *International Journal of Computational Intelligence and Applications*, vol. 14, no. 02, p. 1550007, 2015. [Cited on pages 6 and 7.]

[20] P. Baldi and Y. Chauvin, “Smooth on-line learning algorithms for hidden Markov models,” *Neural Computation*, vol. 6, no. 2, pp. 307–318, 1994. [Cited on pages 6 and 41.]

[21] G. Niina and H. Dozono, “The spherical hidden Markov self organizing map for learning time series data,” in *International Conference on Artificial Neural Networks*. Springer, 2012, pp. 563–570. [Cited on page 7.]

[22] N. Yamaguchi, “Self-organizing hidden Markov models,” in *International Conference on Neural Information Processing*. Springer, 2010, pp. 454–461. [Cited on page 7.]

[23] G. Caridakis, K. Karpouzis, A. Drosopoulos, and S. Kollias, “SOMM: Self organizing Markov map for gesture recognition,” *Pattern Recognition Letters*, vol. 31, no. 1, pp. 52–59, 2010. [Cited on page 7.]

[24] R. Jaziri, M. Lebbah, Y. Bennani, and J.-H. Chenot, “SOS-HMM: self-organizing structure of hidden Markov model,” in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 87–94. [Cited on page 7.]

[25] C. Ferles and A. Stafylopatis, “Sequence clustering with the self-organizing hidden Markov model map,” in *2008 8th IEEE International Conference on BioInformatics and BioEngineering*. IEEE, 2008, pp. 1–7. [Cited on pages 8, 9, 10, and 11.]

[26] C. Ferles, W.-S. Beaufort, and V. Ferle, “Self-organizing hidden Markov model map (SOHMMM): Biological sequence clustering and cluster visualization,” in *Hidden Markov Models*. Springer, 2017, pp. 83–101. [Cited on page 8.]

[27] W. Khreich, E. Granger, A. Miri, and R. Sabourin, “A survey of techniques for incremental learning of HMM parameters,” *Information Sciences*, vol. 197, pp. 105–130, 2012. [Cited on pages 8 and 41.]

[28] S.-B. Cho, “Incorporating soft computing techniques into a probabilistic intrusion detection system,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 2, pp. 154–160, 2002. [Cited on page 8.]

[29] W. Wang, X. Guan, X. Zhang, and L. Yang, “Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data,” *Computers & Security*, vol. 25, no. 7, pp. 539–550, 2006. [Cited on page 8.]

[30] A. Allahdadi, R. Morla, and J. S. Cardoso, “Outlier detection in 802.11 wireless access points using hidden Markov models,” in *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP*. IEEE, 2014, pp. 1–8. [Cited on page 9.]

[31] A. Allahdadi and R. Morla, “Anomaly detection and modeling in 802.11 wireless networks,” *Journal of Network and Systems Management*, vol. 27, no. 1, pp. 3–38, Jan 2019. [Cited on pages 9 and 18.]

[32] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989. [Cited on page 14.]

[33] B.-H. Juang and L. R. Rabiner, “A probabilistic distance measure for hidden Markov models,” *AT&T Technical Journal*, vol. 64, no. 2, pp. 391–408, 1985. [Cited on page 14.]

[34] OMNeT++, discrete event simulator. [Online]. Available: https://www.omnetpp.org/ [Cited on pages 16 and 30.]

[35] Inet framework. [Online]. Available: https://inet.omnetpp.org/ [Cited on pages 16 and 30.]

[36] L. R. Rabiner and B.-H. Juang, “An introduction to hidden markov models,” *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986. [Cited on pages 23 and 39.]

[37] 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. [Cited on page 23.]

[38] Z. Yang, J. Zhao, B. Dhingra, K. He, W. W. Cohen, R. Salakhutdinov, and Y. LeCun, "Glomo: Unsupervisedly learned relational graphs as transferable representations," 2018. [Online]. Available: https://arxiv.org/abs/1806.05662 [Cited on page 26.]

[39] L. Baum, "An inequality and associated maximization technique in statistical estimation of probabilistic functions of a markov process," *Inequalities*, vol. 3, pp. 1–8, 1972. [Cited on page 27.]

[40] S. Lebedev. hmmlearn, hidden Markov models in python, with scikit-learn like API. [Online]. Available: https://github.com/hmmlearn/hmmlearn [Cited on page 29.]

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. [Cited on page 29.]

[42] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, "Recurrent network models for human dynamics," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4346–4354. [Cited on pages 33, 34, and 35.]

[43] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 4674–4683. [Cited on pages 33 and 35.]

[44] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5308–5317. [Cited on pages 33 and 35.]

[45] D. Pavllo, D. Grangier, and M. Auli, "Quaternet: A quaternion-based recurrent model for human motion," *arXiv preprint arXiv:1805.06485*, 2018. [Cited on pages 33 and 35.]

[46] C. Ionescu, F. Li, and C. Sminchisescu, "Latent structured models for human pose estimation," in *International Conference on Computer Vision*, 2011. [Cited on page 33.]

[47] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014. [Cited on page 33.]

[48] C. M. Bishop, "Mixture density networks," Citeseer, Tech. Rep., 1994. [Cited on page 39.]

[49] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[50] L. Bazzani, H. Larochelle, and L. Torresani, "Recurrent mixture density network for spatiotemporal visual attention," *arXiv preprint arXiv:1603.08199*, 2016. [Cited on page 39.]

[51] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. Jan, pp. 993–1022, 2003. [Cited on page 41.]

[52] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013. [Cited on page 41.]

[53] V. V. Digalakis, "Online adaptation of hidden Markov models using incremental estimation algorithms," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 3, pp. 253–261, 1999. [Cited on page 41.]

[54] G. Mongillo and S. Deneve, "Online learning with hidden markov models," *Neural computation*, vol. 20, pp. 1706–16, 08 2008. [Cited on page 41.]

[55] T. Rydén, "On recursive estimation for hidden Markov models," *Stochastic Processes and their Applications*, vol. 66, no. 1, pp. 79–96, 1997. [Cited on page 41.]