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 30, 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
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
$\{1, \ldots, n\}$	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
$\mathbb{A} \setminus \mathbb{B}$	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 $\mathbf{A}$
$\mathbf{A}_{:,:,i}$	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 $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tensor containing derivatives of y with respect to $\mathbf{X}$
$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla_{\boldsymbol{x}}^2 f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$	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
$D_{\mathrm{KL}}(P \Vert Q)$	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

$f : \mathbb{A} \to \mathbb{B}$	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 $f(\mathbf{X})$. This denotes the application of f to the array element-wise. For example, if $\mathbf{C} = \sigma(\mathbf{X})$, 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

Some parts of this chapter were originally published in or adapted from:

> [2] 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 (presented in Section 2.2)
>
> [3] 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 (presented in Section 2.3)

In [2], Diogo Pernes worked primarily on the mathematical derivation of the algorithm and provided some advice about the experimental protocol. Allahdadi was responsible for motivating the application, researching related work, conducting the experimental evaluation, and generating the wireless simulation dataset. Cardoso and Morla supervised the work.

2.1 Introduction

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. 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:

$$f(x) = \sum_{z=1}^{M} s_z \phi_z(x), \tag{2.1}$$

where $\phi_z(x)$ are the atoms and only a few of the scalars s_z 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 intra-correlation, 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 (Section 2.2); ii) we present a novel algorithm that learns an entity-dependent model based on a mixture of shared HMMs (Section 2.3); 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 (Section 2.5).

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 (Somervuo [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 (Somervuo [14], Kurimo and Somervuo [15], Morimoto [16]). The second

approach places the SOM on top of the HMM (Ferles and Stafylopatis [17], Ferles et al. [18], Lebbah et al. [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 by Ferles and Stafylopatis [17] and motivated by Baldi and Chauvin [20]. We extend the model to fit the requirements of our anomaly-detection problem and extend the presented algorithm by Ferles and Stafylopatis [17] to multivariate Gaussian emissions*.

2.2.2 Related work

A number of studies in the literature have integrated the SOM and HMM in different manners. Niina and Dozono [21] proposed the spherical self-organizing map (S-SOM), which uses HMM models as neurons (S-HMM-SOM) to classify the time-series data. This work only considers discrete observations and the model parameters are updated using the Baum-Welch algorithm (Baum [22]). Yamaguchi [23] 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, Caridakis et al. [24] present a SOMM-based architecture 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. Jaziri et al. [25] also exploit the combination of the SOM and HMM (SOS-HMM: self-organizing structure of HMM) and 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.

Lebbah et al. [19] presents a probabilistic self-organizing map called PrSOMS for clustering and visualization of dependent and non-identically distributed data. In this model,

*Ferles and Stafylopatis [17] only address the discrete-observation setting.

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. Morimoto [16] investigate the effects of meteorological factors on the occurrence of strokes. 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.

Ferles and Stafylopatis [17, 26], Ferles et al. [27] apply the fusion and synergy of SOMs and HMMs 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. Khreich et al. [28] 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 [29], Wang et al. [30]). Cho [29] 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 applied: a single model for all the users and individual models for each user.

Wang et al. [30] investigated the HMM and the SOM 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 [29] and competitive roles in [30].

In this work, we intend to benefit from the collaboration of these two techniques (SOM and HMM), as proposed by Ferles and Stafylopatis [26], to extend previous anomaly-detection frameworks applying only HMM (Allahdadi et al. [13, 31], Allahdadi and Morla [32]).

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 [26]. 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 atoms. Each observation is assumed to be discrete, so $\mathbf{x}^{(t)} \in \{1, \ldots, o\}$, 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 atoms 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:

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

where $\mathbf{h} = \left(\mathrm{h}^{(0)}, \ldots, \mathrm{h}^{(T)}\right)$ is the sequence of hidden states of the HMM and $\mathrm{h}^{(t)} \in \{1, \ldots, s\}$, 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 $B_{h,x}^{(z)} \triangleq p(\mathrm{x}^{(t)} = x \mid \mathrm{h}^{(t)} = h, \mathrm{z} = z)$, 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 2.1 and its derivation can be found in Ferles and Stafylopatis [26].

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 $\mathbf{x}^{(t)}$ 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. $\mathbf{x}^{(t)} \mid (\mathrm{h}^{(t)}, \mathrm{z}) \sim \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}_h^{(z)}, \boldsymbol{\Sigma}_h^{(z)}\right)$, 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

Algorithm 2.1 SOHMMM Learning Algorithm for Discrete Observations [26]

Inputs: the lattice r with m nodes, the hyperparameter λ of the neighborhood function v, the set of initial parameters $\Theta^{(0)}$, the learning rate η, and the number of training iterations `trainIter`.

for $i = 1, \ldots,$ `trainIter` **do**

  Observe $\mathbf{x} = \boldsymbol{x}$, with length t.

  for $z = 1, \ldots, m$ **do** (Forward-Backward algorithm)

    $\alpha_1^{(z)}(h) := \pi_h^{(z)} B_{h,x^{(1)}}^{(z)}, \quad h = 1, \ldots, s;$

    $\alpha_{t+1}^{(z)}(h) := B_{h,x^{(t+1)}}^{(z)} \sum_{h'=1}^{s} \alpha_{t+1}^{(z)}(h') A_{h',h}^{(z)}, \quad t = 1, \ldots, T-1, \quad h = 1, \ldots, s;$

    $\beta_t^{(z)}(h) := 1, \quad h = 1, \ldots, s;$

    $\beta_t^{(z)}(h) := \sum_{h'=1}^{s} A_{h,h'}^{(z)} B_{h,x^{(t+1)}}^{(z)} \beta_{t+1}^{(z)}(h'), \quad t = T-1, \ldots, 1, \quad h = 1, \ldots, s.$

  end for

  $z^* := \arg\max_{z' \in \{1,\ldots,m\}} \sum_{z=1}^{m} p(\boldsymbol{x} \mid z, \Theta^{(i-1)}) v(z, z').$

  for $z = 1, \ldots, m$ **do**

    $u_h^{(z)} := u_h^{(z)} + \eta \pi_h^{(z)} \left[B_{h,x^{(1)}}^{(z)} \beta_1^{(z)}(h) - p(\boldsymbol{x} \mid z, \Theta^{(i-1)}) \right], \quad h = 1, \ldots, s;$

    $W_{h,h'}^{(z)} := W_{h,h'}^{(z)} + \eta v(z, z^*) A_{h',h}^{(z)} \sum_{t=1}^{T-1} \left[\alpha_t^{(z)}(h) B_{h,x^{(t+1)}}^{(z)} \beta_{t+1}^{(z)}(h') - \alpha_t^{(z)}(h) \beta_t^{(z)}(h) \right], \quad h, h' = 1, \ldots, s;$

    $R_{h,x'}^{(z)} := R_{h,x'}^{(z)} + \eta v(z, z^*) \sum_{t=1}^{T} \left[\mathbf{1}_{x^{(t)}=x'} \alpha_t^{(z)}(h) \beta_t(h) - B_{h,x'}^{(z)} \alpha_t^{(z)}(h) \beta_t^{(z)}(h) \right], \quad h = 1, \ldots, s, \quad x' = 1, \ldots, o;$

    $\pi_h^{(z)} := \exp(u_h^{(z)}) \Big/ \sum_{h'=1}^{s} \exp(u_{h'}^{(z)}), \quad h = 1, \ldots, s;$

    $A_{h,h'}^{(z)} := \exp(W_{h,h'}^{(z)}) \Big/ \sum_{h''=1}^{s} \exp(W_{h,h''}^{(z)}), \quad h, h' = 1, \ldots, s;$

    $B_{h,x'}^{(z)} := \exp(R_{h,x'}^{(z)}) \Big/ \sum_{x''=1}^{o} \exp(R_{h,x''}^{(z)}), \quad h = 1, \ldots, s, \quad x' = 1, \ldots, o.$

  end for

  $\Theta^{(i)} := \{(\boldsymbol{u}^{(z)}, \boldsymbol{W}^{(z)}, \boldsymbol{R}^{(z)})\}_{z=1}^{m}.$

end for

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 2.1. Let us denote by $\boldsymbol{x}$ the observation at a given time t (i.e., $\mathbf{x}^{(t)} = \boldsymbol{x}$) and consider the corresponding state $\mathrm{h}^{(t)}$ and the node z as fixed at h and z, respectively. Thus, we use the notation $p(\boldsymbol{x} \mid h, z)$ as a short for $p(\mathbf{x}^{(t)} = \boldsymbol{x} \mid \mathrm{h}^{(t)} = h, \mathrm{z} = z)$. We have:

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

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,

$$\begin{aligned}
\nabla_{\boldsymbol{\mu}_h^{(z)}} p(\boldsymbol{X} \mid z) &= \sum_{t=1}^{T} \frac{\partial p(\boldsymbol{X} \mid z)}{\partial p(\boldsymbol{x}^{(t)} \mid h, z)} \nabla_{\boldsymbol{\mu}_h^{(z)}} p(\boldsymbol{x}^{(t)} \mid h, z) \\
&= \sum_{t=1}^{T} \frac{\alpha_t^{(z)}(h)\beta_t^{(z)}(h)}{p(\boldsymbol{x}^{(t)} \mid h, z)} \nabla_{\boldsymbol{\mu}_h^{(z)}} p(\boldsymbol{x}^{(t)} \mid h, z) \\
&= \sum_{t=1}^{T} \alpha_t^{(z)}(h)\beta_t^{(z)}(h)(\boldsymbol{S}\boldsymbol{S}^\top)^{-1}(\boldsymbol{x}^{(t)} - \boldsymbol{\mu}), \qquad (2.11) \\
\nabla_{\boldsymbol{S}_h^{(z)}} p(\boldsymbol{X} \mid z) &= \sum_{t=1}^{T} \alpha_t^{(z)}(h)\beta_t^{(z)}(h)\boldsymbol{S}^{-1}\left[(\boldsymbol{x}^{(t)} - \boldsymbol{\mu})(\boldsymbol{x}^{(t)} - \boldsymbol{\mu})^\top (\boldsymbol{S}\boldsymbol{S}^\top)^{-1} - I_d\right]. \qquad (2.12)
\end{aligned}$$

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 $\boldsymbol{S}_h^{(z)}$:

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

Given equations (2.13) and (2.14), it is straightforward to adapt Algorithm 2.1 to our setting. For completeness, the full algorithm is provided in Algorithm 2.2.

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

Algorithm 2.2 SOHMMM Learning Algorithm for Gaussian Observations

1: **Inputs:** Same as in Algorithm 2.1.

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 2.1, replacing all occurrences of $B^{(z)}_{h,x^{(t)}}$ with $\mathcal{N}\left(\boldsymbol{x}^{(t)}; \boldsymbol{\mu}_h^{(z)}, \boldsymbol{S}_h^{(z)} \boldsymbol{S}_h^{(z)\top}\right)$.

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 2.1, replacing all occurrences of $B^{(z)}_{h,x^{(t)}}$ with $\mathcal{N}\left(\boldsymbol{x}^{(t)}; \boldsymbol{\mu}_h^{(z)}, \boldsymbol{S}_h^{(z)} \boldsymbol{S}_h^{(z)\top}\right)$ and all occurences of $\boldsymbol{x}$ with $\boldsymbol{X}$;

8: $\boldsymbol{\mu}_h^{(z)} := \boldsymbol{\mu}_h^{(z)} + \eta v(z, z^*) \sum_{t=1}^{T} \alpha_t^{(z)}(h) \beta_t^{(z)}(h) (\boldsymbol{S}_h^{(z)} \boldsymbol{S}_h^{(z)\top})^{-1} (\boldsymbol{x}^{(t)} - \boldsymbol{\mu}_h^{(z)}), \quad h = 1, \ldots, s;$

9: $$\boldsymbol{S}_h^{(z)} := \boldsymbol{S}_h^{(z)} + \eta v(z, z^*) \sum_{t=1}^{T} \alpha_t^{(z)}(h) \beta_t^{(z)}(h) \boldsymbol{S}_h^{(z)^{-1}} \cdot \left[(\boldsymbol{x}^{(t)} - \boldsymbol{\mu}_h^{(z)}) (\boldsymbol{x}^{(t)} - \boldsymbol{\mu}_h^{(z)})^\top (\boldsymbol{S}_h^{(z)} \boldsymbol{S}_h^{(z)\top})^{-1} - I_d \right], \quad h = 1, \ldots, s;$$

10: Proceed as in lines 15 and 16 of Algorithm 2.1.

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**

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 $\sqrt{2}$.

In order to perform this experiment, we need a quantitative measurement of 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

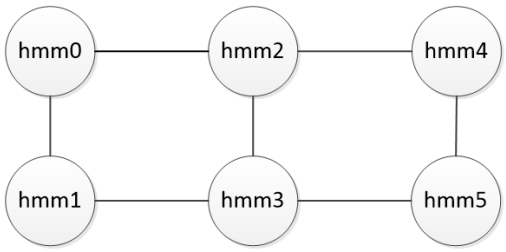

FIGURE 2.1: 2×3 rectangular lattice.

(Rabiner [33]), so the dissimilarity should not be computed in parameter space. An alternative to compare two HMMs z and z', proposed by Juang and Rabiner [34], 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.

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

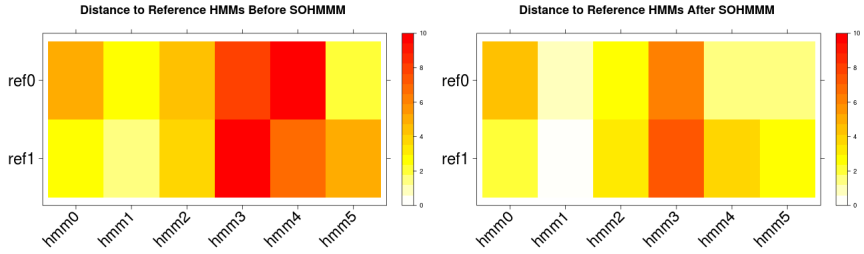

FIGURE 2.2: Monte Carlo approximation of the Kullback-Leibler divergence between the random and reference HMMs before and after applying the SOHMMM algorithm.

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.

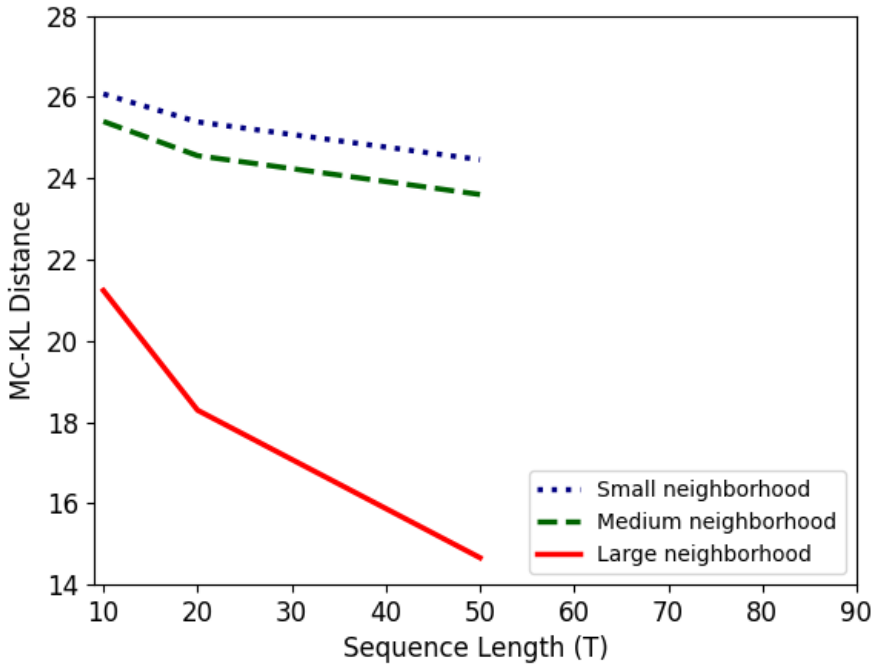

FIGURE 2.3: Monte Carlo approximation of the Kullback-Leibler divergence between the random and reference HMMs with different neighborhood sizes and sequence lengths.

2.2.5.2 Wireless Simulation Data

We implemented a set of wireless simulations using the OMNeT++ [35] simulator along with the INET framework [36]. 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 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 by Allahdadi et al. [13], Allahdadi and Morla [32]). 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:

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

where $\text{Adj}(z')$ 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.

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.

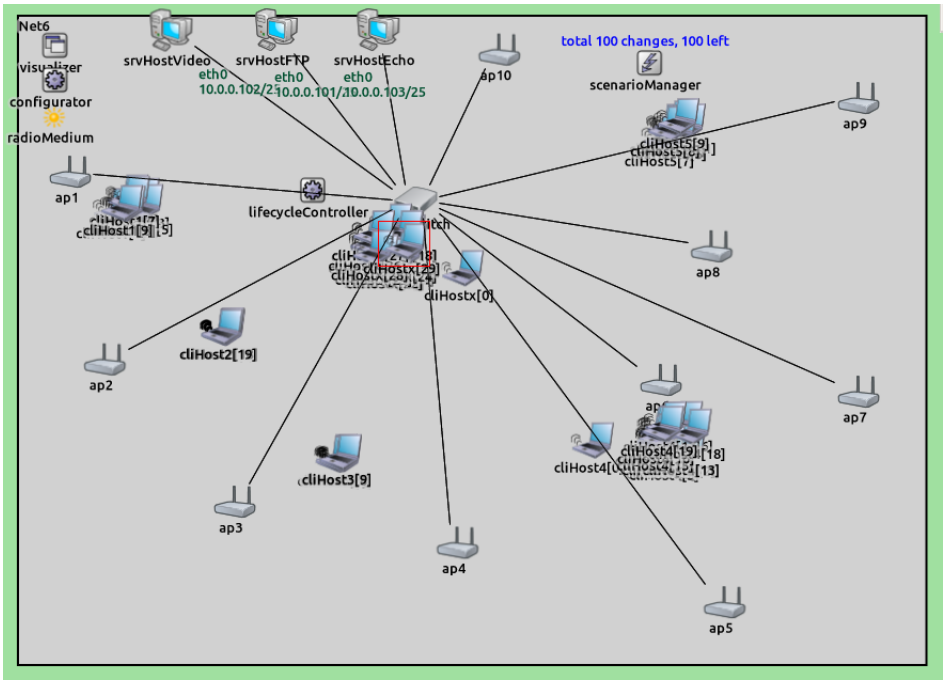

FIGURE 2.4: Wireless network simulated in OMNeT++/INET.

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. [2] for more details about the experiments and further experimental results.

2.2.6 Conclusion

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

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**

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 $\mathcal{Y} = \{1, \ldots, k\}$ 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 $\mathbf{X} = \left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}\right)$, $\mathbf{x}^{(t)} \in \mathbb{R}^d$, 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:

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

where $\mathbf{h} = \left(\mathrm{h}^{(0)}, \ldots, \mathrm{h}^{(T)}\right)$ is the sequence of hidden states of the HMM and $\mathrm{h}^{(t)} \in \{1, \ldots, s\}$, 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 $\mathcal{Y}$ 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.

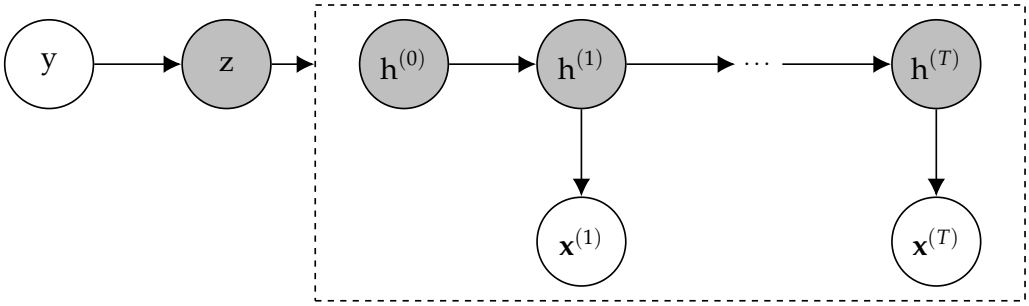

FIGURE 2.5: Representation of the model as a Bayesian network. The node z is a parent of all nodes inside the dashed box (the connections were omitted for clarity). Gray nodes are used for latent variables.

2.3.2.2 Inference

Given an observed sequence $\boldsymbol{X}$ and its corresponding node $y \in \mathcal{Y}$, 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 $O(Ts^2)$ time, using the Forward algorithm (Rabiner and Juang [37]). Then, $p(\boldsymbol{X} \mid y)$ is obtained by applying equation (2.18), so inference in the overall model is done in at most $O(Ts^2m)$ 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 $y_i \in \mathcal{Y}$, the model defined by equations (2.18) and (2.19) may be easily trained using the Expectation-Maximization (EM) algorithm (Dempster et al. [38]), (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 2.3. 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.1.1. 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:

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

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 $\mathbb{E}_{z\sim p(z|y)}[p(z \mid y')]$.

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

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

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

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. 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 $y, y' \in \mathcal{Y}$, 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 $G_{y,y'} > 0$, $\forall y, y' \in \mathcal{Y}$, 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 $G_{y,y'} < 0$, $\forall y, y' \in \mathcal{Y}$, 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.1.2). 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. [39]:

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

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 2.4.

Algorithm 2.3 EM algorithm for the mixture without regularization (MHMM).

1: **Inputs:** The training set, consisting of n tuples $(\boldsymbol{X}_i, y_i)$, a set of initial parameters $\Theta^{(0)}$, and the number of training iterations `trainIter`.

2: **for** $j = 1, \ldots,$ `trainIter` **do**

3: **E-step:**

4: $n_y := \sum_{i=1}^{n} \mathbf{1}_{y_i=y}$, for $y = 1, \ldots, k$;

5: Obtain the mixture posteriors $\eta_i^{(z)} := p(z \mid \boldsymbol{X}_i, y_i)$, for $i = 1, \ldots, n$ and $z = 1, \ldots, m$, by computing $\tilde{\eta}_i^{(z)} := p(\boldsymbol{X}_i \mid z)p(z \mid y_i)$ and normalizing it;

6: Obtain the state posteriors $\gamma_{i,h}^{(z)}(t) := p(\mathrm{h}^{(t)} = h \mid z, \boldsymbol{X}_i)$ and $\xi_{i,h,h'}^{(z)}(t) := p(\mathrm{h}^{(t-1)} = h, \mathrm{h}^{(t)} = h' \mid z, \boldsymbol{X}_i)$, for $i = 1, \ldots, n$, $t = 1, \ldots, T_i$, $z = 1, \ldots, m$, and $h, h' = 1, \ldots, s$, as done in the Baum-Welch algorithm (Baum [22]).

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: $A_{h,h'}^{(z)} := \frac{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{t=1}^{T_i} \xi_{i,h,h'}^{(z)}(t)}{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{t=0}^{T_i-1} \gamma_{i,h}^{(z)}(t)}$, for $z = 1, \ldots, m$, and $h, h' = 1, \ldots, s$, obtaining $\boldsymbol{A}^{(z)}$;

11: $\boldsymbol{\mu}_h^{(z)} := \frac{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{t=1}^{T_i} \gamma_{i,h}^{(z)}(t) \boldsymbol{x}_i^{(t)}}{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{t=1}^{T_i} \gamma_{i,h}^{(z)}(t)}$, for $z = 1, \ldots, m$ and $h = 1, \ldots, s$;

12: ${\boldsymbol{\sigma}_h^{(z)}}^2 := \frac{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{t=1}^{T_i} \gamma_{i,h}^{(z)}(t) (\boldsymbol{x}_i^{(t)} - \boldsymbol{\mu}_h^{(z)})^2}{\sum_{i=1}^{n} \eta_i^{(z)} \sum_{t=1}^{T_i} \gamma_{i,h}^{(z)}(t)}$, 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 2.4 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 λ, 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 ρ to perform gradient ascent over the mixture coefficients.

2: **for** $j = 1, \ldots,$`trainIter` **do**

3: **E-step:** same as in Algorithm 2.3.

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 2.3;

13: $\Theta^{(j)} := \bigcup_{h,y,z} \left\{ \boldsymbol{\beta}^{(y)}, \boldsymbol{\pi}^{(z)}, \boldsymbol{A}^{(z)}, \boldsymbol{\mu}_h^{(z)}, \boldsymbol{\sigma}_h^{(z)} \right\}$.

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 (2.18) and (2.19)), 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 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 2.3 and SpaMHMM is trained using Algorithm 2.4. However, we opted to use Adam (Kingma and Ba [41]) instead of *vanilla* gradient ascent in the inner loop of Algorithm 2.4, 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++ [35] and INET [36] 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 $\max_{y,y'} G_{y,y'} = 1$. Following 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 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 $\rho = 10^{-3}$. 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.

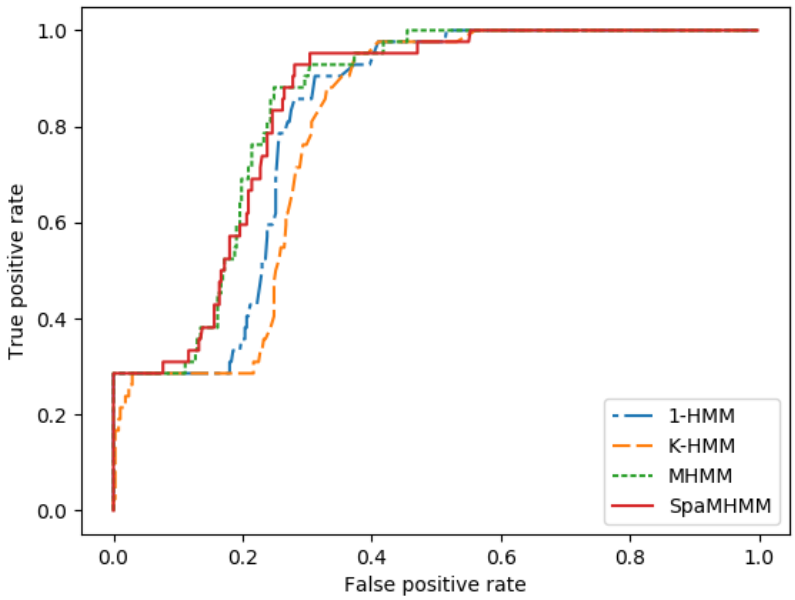

FIGURE 2.6: ROC curves for each model on the Wi-Fi dataset, for one of the 10 runs.

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

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

sequences of human joint positions, represented as motion capture (mocap) data. The 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.2) 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 Spa MHMM), 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]	0.21	0.34	0.56	0.62	0.20	0.35	0.58	0.70	0.25	0.47	0.93	0.90	0.26	0.60	0.85	0.93

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.

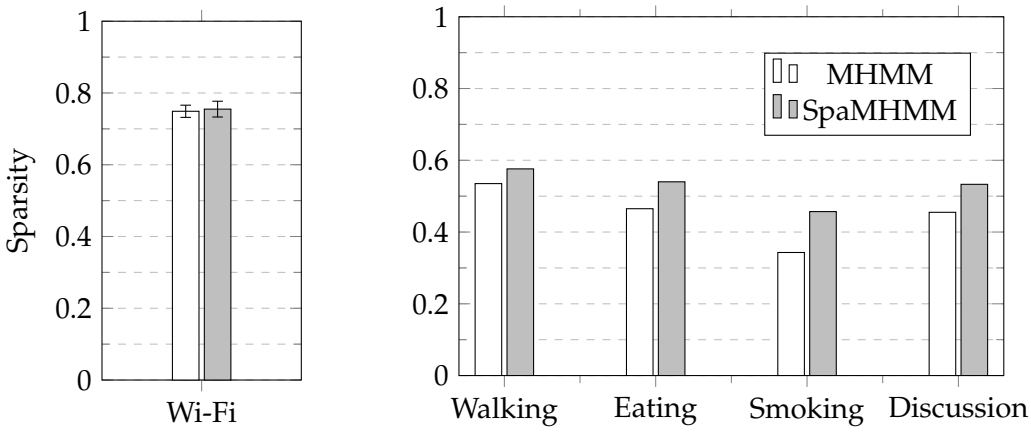

FIGURE 2.7: Relative sparsity (number of coefficients equal to zero / total number of coefficients) of the obtained MHMM and SpaMHMM models on the Wi-Fi dataset (left) and on the Human3.6M dataset for different actions (right). For the Wi-Fi dataset, the average value over the 10 training runs is shown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

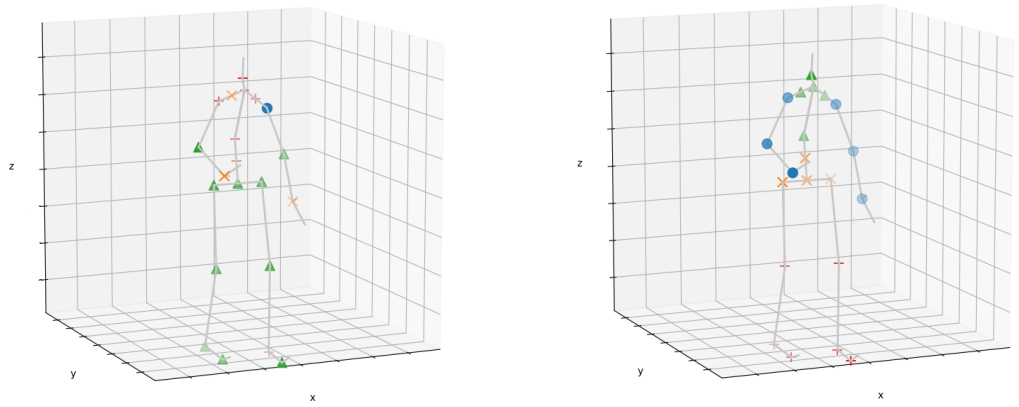

FIGURE 2.8: Assignments of joints to clusters in MHMM (left) and SpaMHMM (right). The different colors (blue, green, orange, red) and the respective symbols ('○', '△', 'x', '+') on each joint represent the cluster that the joint was assigned to. on each joint represent the cluster that the joint was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

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

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 $\mathcal{Y}$ produces sequences $\mathbf{X} = \left(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)}\right)$. The prior knowledge about the pairwise affinity degrees between entities in $\mathcal{Y}$ 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:

$$p(\mathbf{X} \mid \mathrm{y}) = \int p(\mathbf{X} \mid \mathbf{z}) p(\mathbf{z} \mid \mathrm{y}) \, \mathrm{d}\mathbf{z}. \tag{2.27}$$

This model comprises a latent space 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 is discrete and finite and the latent generative model is an HMM parametrized by $\mathbf{z}$. The discrete nature of the latent space and the underlying independence assumptions of the HMM allow for efficient exact inference in the resulting model using the Forward algorithm [37].

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 parametrize a Gaussian mixture model with c components:

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

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

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:

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

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:

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

where the $\boldsymbol{z}_i$ 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 [3], 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):

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

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, Blei et al. [51]) or approximating the posterior with a parametric model, like a deep neural network (e.g. variational auto-encoders, Kingma and Welling [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 (Baldi and Chauvin [20]), incremental versions of EM (Digalakis [53], Mongillo and Deneve [54]), and recursive maximum likelihood estimation (Rydén [55]). Khreich et al. [28] provided an overview of these methods. Both gradient-based optimization and incremental EM may be applied 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 $y \in \mathcal{Y}$ 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 $p(\mathrm{z} \mid y)$ to accommodate the new data. By doing this, the entity y is performing a search over the latent space 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.

Chapter 3

Multi-source domain adaptation

Some parts of this chapter were originally published in or adapted from:

> [56] F. T. de Andrade, "Adversarial domain adaptation for sensor networks," Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2020 (presented in Section 3.3)
>
> [57] D. Pernes and J. S. Cardoso, "Tackling unsupervised multi-source domain adaptation with optimism and consistency," *CoRR*, vol. abs/2009.13939, 2020 (presented in Section **??**)

The Master's thesis [56] was supervised by Jaime S. Cardoso and co-supervised by Diogo Pernes.

3.1 Introduction

In Chapter 2, we have addressed the situation where the set of entities was the same at training and testing time. The goal there was to exploit inter-correlations between entities to learn better generative models for each individual entity. Now, we focus on the problem of learning a discriminative model for one particular entity (the *target*) for which no annotated data is available. Assuming some invariance properties, we can hope to accomplish this task by learning a discriminative model using the combination of annotated data from the remaining entities (the *sources*) and unlabeled data from the target entity. Since entities do not need to (and generally do not) correspond to physical objects and may refer to different contexts where the data was collected, they are more commonly

called *domains* and the problem itself is known as *domain adaptation* (DA). In the following, we motivate the practical importance of this problem and summarize our contributions.

Supervised training of deep neural networks has achieved outstanding results on multiple learning tasks greatly due to the availability of large and rich annotated datasets. Unfortunately, annotating such large-scale datasets is often prohibitively time-consuming and expensive. Furthermore, in many practical cases, it is not possible to collect annotated data with the same characteristics as the test data, and, as a result, training and test data are drawn from distinct underlying distributions. As a consequence, the model performance tends to decrease significantly on the test data. The goal of DA algorithms is to minimize this gap by finding transferable knowledge from the source to the target domain. Sometimes, it is assumed that a small portion of labeled target data are available at training time – a setting that is known as *semi-supervised* DA (e.g. Daumé III et al. [58], Donahue et al. [59], Kumar et al. [60], Saito et al. [61], Yao et al. [62]). In this chapter, we focus mostly on the more challenging scenario, where no labeled target data are available for training – known as *unsupervised* DA (e.g. Baktashmotlagh et al. [63], Ganin and Lempitsky [64], Kang et al. [65], Long et al. [66], Zhao et al. [67]). The DA problem, in its semi-supervised and unsupervised variants, has received increased attention in recent years, both from theoretical (e.g. Ben-David et al. [68, 69], Blitzer et al. [70], Cortes and Mohri [71], Gopalan et al. [72], Hoffman et al. [73], Zhao et al. [74]) and algorithmic perspectives (e.g. Ajakan et al. [75], Becker et al. [76], Fernando et al. [77], Jhuo et al. [78], Long et al. [79], Louizos et al. [80], Sun et al. [81], Tzeng et al. [82]). In many situations, the annotated training data may consist of a combination of distinct datasets, some of which may be closer or further away from the target data. Finding nontrivial ways of combining multiple datasets to approximate the target distribution and extracting relevant knowledge from such combination is the purpose of multi-source DA algorithms (e.g. Zhao et al. [67], Hoffman et al. [73], Kim et al. [83], Guo et al. [84], Mansour et al. [85], Schoenauer-Sebag et al. [86], Zhang et al. [87]) and is also our main focus in this chapter.

The remainder of the chapter is organized as follows: i) we formalize the problem and provide some useful background by reviewing some important theoretical results and state of the art algorithms (Section 3.2); ii) we discuss some exploratory solutions for this problem in the context of sensor networks (Section 3.3); and finally iii) we present our own novel algorithm for multi-source domain adaptation (Section **??**).

3.2 Background

We start by analyzing the problem of DA from a theoretical perspective and then we overview the main approaches that constitute the state of the art. Although some authors have considered the problem of DA for regression problems (e.g. [88], [67]), the literature on classification is far more vast. Moreover, many of the results and methods explained in this section can be extended to regression problems with minor modifications. For these reasons, we shall focus our discussion mostly on classification.

3.2.1 Theoretical foundation

3.2.1.1 Single source setting

Ben-David et al. [68] developed a rigorous yet comprehensive theoretical model for domain adaptation that we summarize here. This formulation enlightens the intrinsic difficulties associated with this task and provides a deep foundation for many of the algorithms we discuss in this chapter and, particularly, to our own approach, presented in Section **??**.

Before we present and discuss the most important results, let us introduce a few preliminary definitions. A *domain* $\mathcal{D}$ is defined by a joint distribution $p_{\mathcal{D}}(\mathbf{x}, \mathbf{y})$ over input features $\mathbf{x} \in \mathcal{X}$ and target variables $\mathbf{y} \in \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ denote the input and target spaces, respectively. For the domain adaptation task to be well defined, at least two domains must be considered: a *source* domain $\mathcal{S}$, with joint distribution denoted by $p_{\mathcal{S}}(\mathbf{x}, \mathbf{y})$, from which abundant annotated data is usually available, and a *target* domain $\mathcal{T}$, with joint distribution $p_{\mathcal{T}}(\mathbf{x}, \mathbf{y})$, from which scarce or even zero annotated data is available at training time. Following most classical results from statistical learning theory, Ben-David et al. [68] focused on binary classification, thus $\mathcal{Y} = \{0, 1\}$. Under this setting, it is possible to define a *labeling function* $f_{\mathcal{D}} : \mathcal{X} \mapsto [0, 1]$ for each domain, given by $f_{\mathcal{D}}(\boldsymbol{x}) = p_{\mathcal{D}}(\mathrm{y} = 1 \mid \boldsymbol{x})$. A *hypothesis* is any function $h : \mathcal{X} \mapsto \{0, 1\}$ and a set $\mathcal{H}$ of these functions is called a *hypothesis class*. The expected absolute difference between h and $f_{\mathcal{D}}$ is called the *risk* (or *error*) of hypothesis h (with respect to the labeling function $f_{\mathcal{D}}$):

$$\epsilon(h, f_{\mathcal{D}}) \triangleq \mathbb{E}_{\mathbf{x} \sim p_{\mathcal{D}}} |h(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x})|. \tag{3.1}$$

We use $\epsilon_{\mathcal{S}}(h)$ and $\epsilon_{\mathcal{T}}(h)$ as shorthands for $\epsilon(h, f_{\mathcal{S}})$ and $\epsilon(h, f_{\mathcal{T}})$ and refer to them as the source and target risks (or errors), respectively. The empirical estimates of these are denoted as $\widehat{\epsilon}_{\mathcal{S}}(h)$ and $\widehat{\epsilon}_{\mathcal{T}}(h)$, respectively.

Given two domains $\mathcal{D}$ and $\mathcal{D}'$ and a hypothesis class $\mathcal{H}$, the $\mathcal{H}$-divergence provides a distance measure between the marginal distributions of features in $\mathcal{D}$ and $\mathcal{D}'$ (according to $\mathcal{H}$):

$$d_{\mathcal{H}}(\mathcal{D}, \mathcal{D}') \triangleq \sup_{h \in \mathcal{H}} 2|\Pr_{\mathcal{D}}(\mathbf{1}_h) - \Pr_{\mathcal{D}'}(\mathbf{1}_h)|,$$

where $\mathbf{1}_h \triangleq \{\boldsymbol{x} \in \mathcal{X} : h(\boldsymbol{x}) = 1\}$ and $\Pr_{\mathcal{D}}(\mathbf{1}_h)$ is the probability assigned by the distribution $p_{\mathcal{D}}(\mathbf{x})$ to the subset $\mathbf{1}_h \subseteq \mathcal{X}$. As is often the case, when the true underlying marginal distributions are unknown or intractable but finite sets of (unlabeled) samples from both domains are available, an empirical $\mathcal{H}$-divergence can be constructed by replacing the true probabilities $\Pr_{\mathcal{D}}(\mathbf{1}_h)$ and $\Pr_{\mathcal{D}'}(\mathbf{1}_h)$ by the respective empirical estimates. Remarkably, under weak conditions on the hypothesis class, computing this empirical $\mathcal{H}$-divergence is equivalent to finding the hypothesis in $\mathcal{H}$ that maximally discriminates between samples of the two domains. This result is enunciated formally in Lemma 3.1 and, as we shall see later, is exploited by adversarial-based approaches for domain adaptation.

Lemma 3.1. *(Lemma 2 from Ben-David et al. [68]) Let $\mathcal{H}$ be a hypothesis class such that if $h \in \mathcal{H}$ then $1 - h \in \mathcal{H}$. Given two sets $\widehat{D}$ and $\widehat{D}'$ of n samples each drawn from two domains D and D', the empirical $\mathcal{H}$-divergence between D and D' is given by:*

$$\widehat{d}_{\mathcal{H}}(\mathcal{D}, \mathcal{D}') = 2\left(1 - \min_{h \in \mathcal{H}}\left[\frac{1}{n}\sum_{\boldsymbol{x}:h(\boldsymbol{x})=1} \mathbf{1}_{\boldsymbol{x} \in \widehat{\mathcal{D}}} + \frac{1}{n}\sum_{\boldsymbol{x}:h(\boldsymbol{x})=0} \mathbf{1}_{\boldsymbol{x} \in \widehat{\mathcal{D}}'}\right]\right). \tag{3.2}$$

Note that if the two sets $\widehat{\mathcal{D}}$ and $\widehat{\mathcal{D}}'$ can be discriminated perfectly by a hypothesis $h \in \mathcal{H}$ (i.e. if there is an $h \in \mathcal{H}$ such that $h(\boldsymbol{x}) = 0$ if $x \in \widehat{\mathcal{D}}$ and $h(\boldsymbol{x}) = 1$ if $x \in \widehat{\mathcal{D}}'$), then $\widehat{d}_{\mathcal{H}}(\mathcal{D}, \mathcal{D}')$ is maximum and equal to 2.

For a hypothesis class $\mathcal{H}$, we may define the *symmetric difference hypothesis class* $\mathcal{H}\Delta\mathcal{H}$ as:

$$\mathcal{H}\Delta\mathcal{H} \triangleq \{l : l(\boldsymbol{x}) = h(\boldsymbol{x}) \oplus h'(\boldsymbol{x}),\ h, h' \in \mathcal{H}\}, \tag{3.3}$$

where $\oplus$ denotes the "exclusive or" (xor) operation. Combining this definition with equation (3.2.1.1), the definition of $\mathcal{H}\Delta\mathcal{H}$-divergence, which happens to play a major role in

Theorem 3.2, follows immediately. This theorem is the main result in this section as it provides an upper bound for the target risk given the source risk and the $\mathcal{H}\Delta\mathcal{H}$-divergence between the source and target domains.

Theorem 3.2. *(Theorem 2 from Ben-David et al. [68]) Let $\mathcal{H}$ be a hypothesis class with VC-dimension d. Consider n unlabeled samples drawn from each of the two domains $\mathcal{S}$ (source) and $\mathcal{T}$ (target). Then, for every $h \in \mathcal{H}$ and any $\delta \in (0,1)$, with probability at least $1-\delta$ over the choice of samples,*

$$\epsilon_{\mathcal{T}}(h) \leq \epsilon_{\mathcal{S}}(h) + \frac{1}{2}\widehat{d}_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S},\mathcal{T}) + \lambda + 2\sqrt{\frac{2d\log(2n) + \log(\frac{2}{\delta})}{n}}, \tag{3.4}$$

where $\lambda \triangleq \min_{h\in\mathcal{H}} \epsilon_{\mathcal{S}}(h) + \epsilon_{\mathcal{T}}(h)$.

This bound immediately confirms the intuition that a low target error can be achieved by training a classifier to minimize the error in the source domain, provided that the marginal distributions of features are similar (i.e. $\widehat{d}_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S},\mathcal{T})$ is small) and a low error on the combination of the two domains can be achieved (i.e. λ is also small). A deeper and more complete interpretation of this bound shall be provided later on, when we take into account the fact that, by applying deep neural networks, we can not only construct rich hypothesis classes but also manipulate and learn feature representations. The latter observation suggests that this kind of the classifiers may also have an impact on the $\widehat{d}_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S},\mathcal{T})$ and λ terms in equation (3.4), which will indeed be the case.

3.2.1.2 Multi-source setting

So far we have only considered the setting where a single source domain was available. However, in many practical cases, the annotated training dataset consists of a collection of subdatasets, each one belonging to its own domain. Therefore, it makes sense to consider k distinct source domains $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ and, in particular, to see how Theorem 3.2 can be generalized to this setting.

Theorem 3.3. *(Theorem 2 from Zhao et al. [67]) Let $\mathcal{H}$ be a hypothesis class with VC-dimension d. Consider n unlabeled samples drawn from the target domain $\mathcal{T}$ and n/k annotated samples drawn from each of the k source domains $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$. Then, for every $h \in \mathcal{H}$, any $\boldsymbol{\alpha} \in [0,1]^k$: $\sum_{j=1}^k \alpha_i = 1$, and any $\delta \in (0,1)$, with probability at least $1-\delta$ over the choice of samples,*

$$\epsilon_{\mathcal{T}}(h) \leq \sum_{j=1}^{k} \alpha_j \left(\widehat{\epsilon}_{\mathcal{S}_j}(h) + \frac{1}{2}\widehat{d}_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S}_j,\mathcal{T})\right) + \lambda_{\boldsymbol{\alpha}} + O\left(\sqrt{\frac{1}{n}\left(\log\frac{1}{\delta} + d\log\frac{n}{d}\right)}\right), \tag{3.5}$$

where $\lambda_{\boldsymbol{\alpha}} \triangleq \min_{h \in \mathcal{H}} \epsilon_{\mathcal{T}}(h) + \sum_{j=1}^{k} \alpha_j \epsilon_{\mathcal{S}_j}(h)$.

Unsurprisingly, the bound in Theorem 3.3 is essentially a convex combination of the bounds provided by Theorem 3.2 for each individual source domain. Thus, the same interpretation applies here. Nonetheless, the source weights $\boldsymbol{\alpha}$ provide an extra degree of freedom that should be taken into account. Depending on how much each source domain differs from the target, it may be beneficial to weight each source domain differently. Adjusting these weights is therefore an extra non-trivial task, exclusive to the multi-source setting, that may have a significant impact in the performance of the domain adaptation algorithm.

3.2.2 State of the art

We now do a brief overview of the most relevant DA algorithms, both in the single source and multi-source settings.

As we have just seen from a theoretical point of view, the success of the DA task depends on how similar the target domain is to the source(s), which is equivalent to saying that some properties of the underlying distributions must be invariant across domains. Different DA algorithms can therefore be categorized according to the invariance properties they assume.

3.2.2.1 Target shift

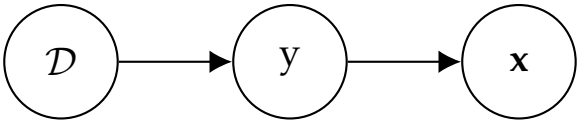

FIGURE 3.1: Graphical representation of the target shift setting as a Bayesian network.

In the *target shift* setting, only the marginal distribution of labels is allowed to vary across domains. A practical example where this assumption may be realistic is when the label y represents having or not a certain disease and the input features $\mathbf{x}$ are symptoms. It is plausible to assume that the prevalence of the disease may vary over time or across different populations, but the probability of some symptom being present or absent given that one has or not the disease should remain constant. Thus, as represented in Figure 3.1, the joint distribution of any domain $\mathcal{D}$ is assumed to factorize as $p_{\mathcal{D}}(\mathbf{x}, \mathrm{y}) = p(\mathbf{x} \mid \mathrm{y}) p_{\mathcal{D}}(\mathrm{y})$, where $p(\mathbf{x} \mid \mathrm{y})$ is domain-invariant. By further assuming that

$\text{Supp}(p_{\mathcal{T}}(\mathrm{y})) \subseteq \text{Supp}(p_{\mathcal{S}}(\mathrm{y}))$, we have:

$$\begin{aligned} p_{\mathcal{T}}(\mathrm{y} \mid \mathbf{x}) &\propto p(\mathbf{x} \mid \mathrm{y}) p_{\mathcal{T}}(\mathrm{y}) \\ &= p(\mathbf{x} \mid \mathrm{y}) p_{\mathcal{S}}(\mathrm{y}) \frac{p_{\mathcal{T}}(\mathrm{y})}{p_{\mathcal{S}}(\mathrm{y})} \\ &\propto p_{\mathcal{S}}(\mathrm{y} \mid \mathbf{x}) \frac{p_{\mathcal{T}}(\mathrm{y})}{p_{\mathcal{S}}(\mathrm{y})}. \end{aligned} \tag{3.6}$$

Thus, if the class ratios $p_{\mathcal{T}}(\mathrm{y})/p_{\mathcal{S}}(\mathrm{y})$ are known, the problem of DA under target shift is solved by learning a probabilistic classifier on the source domain, reweighting it with the class ratios, and then normalizing the class scores. Therefore, the literature for DA under target shift focuses on estimating class ratios when these are unknown and cannot be estimated directly from the training data, due to the absence of labels for the target samples.

An elegant and simple solution to this problem was proposed by Lipton et al. [89]. Specifically, for any classifier $h : \mathcal{X} \mapsto \mathcal{Y}$ trained with labeled data from the source domain, the target shift assumption implies that $p_{\mathcal{T}}(h(\mathbf{x}) \mid \mathrm{y}) = p_{\mathcal{S}}(h(\mathbf{x}) \mid \mathrm{y})$ and hence:

$$\begin{aligned} p_{\mathcal{T}}(h(\mathbf{x})) &= \sum_{\mathrm{y}} p_{\mathcal{T}}(h(\mathbf{x}) \mid \mathrm{y}) p_{\mathcal{T}}(\mathrm{y}) \\ &= \sum_{\mathrm{y}} p_{\mathcal{S}}(h(\mathbf{x}) \mid \mathrm{y}) p_{\mathcal{T}}(\mathrm{y}) \qquad (3.7) \\ &= \sum_{\mathrm{y}} p_{\mathcal{S}}(h(\mathbf{x}), \mathrm{y}) \frac{p_{\mathcal{T}}(\mathrm{y})}{p_{\mathcal{S}}(\mathrm{y})}. \qquad (3.8) \end{aligned}$$

Note that $p_{\mathcal{S}}(h(\mathbf{x}) \mid \mathrm{y})$, $p_{\mathcal{S}}(h(\mathbf{x}), \mathrm{y})$, and $p_{\mathcal{S}}(\mathrm{y})$ can all be estimated from labeled source samples and $p_{\mathcal{T}}(h(\mathbf{x}))$ can be estimated from unlabeled target samples. Thus, one can either use equation (3.7) to estimate $p_{\mathcal{T}}(\mathrm{y})$ or equation (3.8) to estimate class ratios directly.

Other approaches involve learning class-dependent weights to match the mean conditional features of source data with the mean marginal features of target data in a reproducing kernel Hilbert space (e.g. Iyer et al. [90], Zhang et al. [91]), or require density estimation to model $p(\mathbf{x} \mid \mathrm{y})$ (e.g. Chan and Ng [92], Storkey [93]).

3.2.2.2 Conditional shift

In the *conditional shift* setting, the marginal distribution of labels is constant and the conditional of features given labels may change across domains. This scenario is represented in Figure 3.2, from which it becomes clear that the joint distribution takes the form $p_{\mathcal{D}}(\mathbf{x}, \mathrm{y}) = p(\mathrm{y}) p_{\mathcal{D}}(\mathbf{x} \mid \mathrm{y})$, where $p(\mathrm{y})$ is domain-invariant. Besides being less realistic

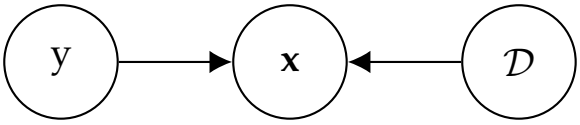

FIGURE 3.2: Graphical representation of the conditional shift setting as a Bayesian network.

than other assumptions, DA under conditional shift is in general an ill-posed problem. Nonetheless, Zhang et al. [91] show that identifiability of $p_{\mathcal{T}}(\mathbf{x} \mid \mathrm{y})$ holds when it is assumed that, for any given y, $p_{\mathcal{T}}(\mathbf{x} \mid y)$ only differs from $p_{\mathcal{S}}(\mathbf{x} \mid y)$ in location and scale and derive a kernel-based approach to estimate these parameters.

3.2.2.3 Concept shift

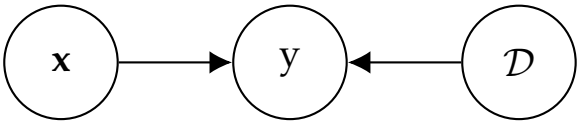

FIGURE 3.3: Graphical representation of the concept shift setting as a Bayesian network.

Concept shift refers to the situation where the marginal feature distributions $p(\mathbf{x})$ are constant but the conditional distribution of the target variable $p_{\mathcal{D}}(\mathrm{y} \mid \mathbf{x})$ is domain-dependent, thus $p_{\mathcal{D}}(\mathbf{x}, \mathrm{y}) = p(\mathbf{x})p_{\mathcal{D}}(\mathrm{y} \mid \mathbf{x})$, as implied by Figure 3.3. When the change happens over time, this setting is also known as *concept drift* (Webb et al. [94]). The literature on concept drift is vast and focuses mostly on the detection of its occurrence so that the model can be updated using new data. Gama et al. [95] overview the most popular techniques to address this problem.

3.2.2.4 Covariate shift

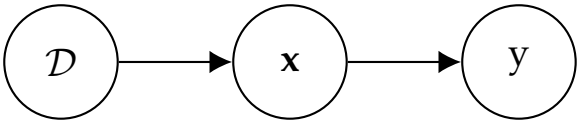

FIGURE 3.4: Graphical representation of the covariate shift setting as a Bayesian network.

Covariate shift is by far the most common assumption and therefore the most widely addressed setting in the DA literature. As implied by the graphical representation in Figure 3.4, here the conditional distribution of labels given features is constant and the marginal distribution of features is domain-dependent. Thus, $p_{\mathcal{D}}(\mathbf{x}, \mathrm{y}) = p(\mathrm{y} \mid \mathbf{x})p_{\mathcal{D}}(\mathbf{x})$, where $p(\mathrm{y} \mid \mathbf{x})$ is constant across domains. This assumption might hold for instance in

image classification problems where the domain shift is caused by different sensors or lighting conditions.

Since $p_{\mathcal{T}}(\mathrm{y} \mid \mathbf{x}) = p_{\mathcal{S}}(\mathrm{y} \mid \mathbf{x})$, infinite labeled data from the source domain and a consistent estimator of this conditional distribution would solve this DA task. However, the former is obviously unrealistic and therefore the problem should be analyzed taking into account that the available data is finite. Let $\ell(\cdot, \cdot)$ be any loss function for the supervised learning problem. The goal is then to find an unbiased estimator of the target loss $\mathbb{E}_{\mathbf{x},\mathrm{y}\sim p_{\mathcal{T}}}\ell(h(\mathbf{x}), \mathrm{y})$ when no labeled samples from the target domain are available. Following Sugiyama et al. [96], if the marginal densities $p_{\mathcal{S}}(\mathbf{x})$ and $p_{\mathcal{T}}(\mathbf{x})$ are known and assuming $\mathrm{Supp}(p_{\mathcal{T}}(\mathbf{x})) \subseteq \mathrm{Supp}(p_{\mathcal{S}}(\mathbf{x}))$, this estimator can be obtained using *importance weights* $w(\mathbf{x}) \triangleq p_{\mathcal{T}}(\mathbf{x})/p_{\mathcal{S}}(\mathbf{x})$:

$$\begin{aligned}
\mathbb{E}_{\mathbf{x},\mathrm{y}\sim p_{\mathcal{T}}}\ell(h(\mathbf{x}), \mathrm{y}) &= \sum_{\mathrm{y}} \int l(h(\mathbf{x}), \mathrm{y}) p_{\mathcal{T}}(\mathbf{x}, \mathrm{y})\, \mathrm{d}\mathbf{x} \\
&= \sum_{\mathrm{y}} \int \ell(h(\mathbf{x}), \mathrm{y}) p(\mathrm{y} \mid \mathbf{x}) p_{\mathcal{T}}(\mathbf{x})\, \mathrm{d}\mathbf{x} \\
&= \sum_{\mathrm{y}} \int \frac{p_{\mathcal{T}}(\mathbf{x})}{p_{\mathcal{S}}(\mathbf{x})} \ell(h(\mathbf{x}), \mathrm{y}) p(\mathrm{y} \mid \mathbf{x}) p_{\mathcal{S}}(\mathbf{x})\, \mathrm{d}\mathbf{x} \\
&= \sum_{\mathrm{y}} \int w(\mathbf{x}) \ell(h(\mathbf{x}), \mathrm{y}) p_{\mathcal{S}}(\mathbf{x}, \mathrm{y})\, \mathrm{d}\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x},\mathrm{y}\sim p_{\mathcal{S}}} w(\mathbf{x}) \ell(h(\mathbf{x}), \mathrm{y}). \qquad (3.9)
\end{aligned}$$

Hence, $(1/n)\sum_{i=1}^{n} w(\boldsymbol{x}_i)\ell(h(\boldsymbol{x}_i), y_i)$ is an unbiased estimator of the target loss, constructed using only source data, which can therefore be used as the loss for the supervised learning problem. When the marginal distributions are unknown and the feature space is low-dimensional, kernel density estimation techniques can be employed to estimate them (e.g. Sugiyama et al. [96], Shimodaira [97], Cortes et al. [98]). An alternative that removes the constraint on the support of the marginal target distribution is learning a transformation $T : \mathcal{X} \mapsto \mathcal{X}$ such that $p_{\mathcal{S}}(T(\mathbf{x})) = p_{\mathcal{T}}(\mathbf{x})$, which can be accomplished using optimal transport theory (Courty et al. [99]). Later approaches aim to relax the covariate shift assumption by trying to align the joint source and target distributions (Courty et al. [100]) and extend the same principles to the multi-source setting (Turrisi et al. [101]).

For high-dimensional data (e.g. images), where the marginal distributions are typically unknown and hard to estimate from finite data, deep learning models play an important role by providing a successful tool learn semantically rich low-dimensional feature representations. It is then possible to learn a function $g : \mathcal{X} \mapsto \mathcal{Z}$ mapping input data

to a new feature space $\mathcal{Z}$ such that $p_{\mathcal{T}}(g(\mathbf{x}))/p_{\mathcal{S}}(g(\mathbf{x})) \approx 1$, i.e. where the marginal feature distributions of the two domains coincide. This approach has the additional benefit of moving the overlapping support assumption to a lower-dimensional space, where it is less likely to be violated than in the original space (e.g. pixel space). In this new feature space the covariate shift has vanished and therefore any classifier $h : \mathcal{Z} \mapsto \mathcal{Y}$ that achieves low error on the source domain will also perform well in the target. An alternative way of motivating these approaches follows from analyzing again the target risk bound provided in equation (3.4). When we presented this bound, we assumed for simplicity that the feature space was fixed and therefore the upper bound would be minimized by minimizing the source risk. However, if the feature space can itself be optimized, the term $\widehat{d}_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S},\mathcal{T})$ can be minimized by finding a feature space where the source and target marginal distributions coincide. This idea has been exploited extensively in recent years, either by matching the distributions using maximum mean discrepancy (e.g. Long et al. [79], Guo et al. [84]) or, in most cases, using an adversarial neural network (e.g. Ganin and Lempitsky [64], Zhao et al. [67], Schoenauer-Sebag et al. [86], Pei et al. [102]).

Adversarial-based DA was originally introduced by Ganin and Lempitsky [64] and results from the observation that computing $\widehat{d}_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S},\mathcal{T})$ is equivalent to finding a classifier that maximally discriminates between samples of the source and target domains (Lemma 3.1). Intuitively, if no classifier exists that can distinguish between source and target features, then the distributions of these two must coincide. Thus, if we have access to sets $\widehat{\mathcal{S}}$ and $\widehat{\mathcal{T}}$ of labeled samples from the source domain and unlabeled samples from the target, respectively, we can train a feature extractor network $g : \mathcal{X} \mapsto \mathcal{Z}$, a classifier $h : \mathcal{Z} \mapsto \mathcal{Y}$, and a domain discriminator $d : \mathcal{Z} \mapsto \{0,1\}$ to solve the following minimax problem*:

$$\min_{g,h} \max_{d} \quad \widehat{\epsilon}_{\mathcal{S}}(h \circ g) + 1 - \left(\frac{1}{n} \sum_{\boldsymbol{x}: d(g(\boldsymbol{x}))=1} \mathbf{1}_{\boldsymbol{x} \in \widehat{\mathcal{S}}} + \frac{1}{n} \sum_{\boldsymbol{x}: d(g(\boldsymbol{x}))=0} \mathbf{1}_{\boldsymbol{x} \in \widehat{\mathcal{T}}} \right), \tag{3.10}$$

where $\circ$ denotes function composition. Several variations to this idea have been proposed so far, aiming to extend it to the multi-source setting (Zhao et al. [67]), or beyond the covariate shift assumption (Pei et al. [102]), or both (Schoenauer-Sebag et al. [86]).

*This objective is merely formal since the 0-1 loss is non-smooth and intractable. In practice, the usual classification losses are used instead.

3.2.2.5 Invariance of causal mechanisms

The settings we have discussed so far consider all features $\mathbf{x}$ as atomic and hence do not take into account how different features interact to produce the target variable y. Other approaches drop this limitation by decomposing the feature vector into individual features $x_1, x_2, \ldots, x_m$, taking into account the structure of the causal Bayesian network governing the data generating process, and using causal inference tools to identify the target distribution. These methods assume that the flow of cause and effect cannot be reversed by domain shift and therefore changes in distribution are due to different interventions in the same causal graph $\mathcal{G}$ (e.g. presence of additional exogenous variables inducing non-causal associations between features and the target variable). Bareinboim and Pearl [103] assume $\mathcal{G}$ is known and all interventions are perfect (i.e. all interventions consist of edge removal operations) and derive conditions for identifiability of $p_{\mathcal{T}}(y \mid \cdots)$ given $\mathcal{G}$ and $p_{\mathcal{S}}(\mathbf{x}, y)$. Rojas-Carulla et al. [104] and Magliacane et al. [105] relax the covariate shift setting by assuming that there exists a strict subset $\bar{\mathbf{x}} \subset \mathbf{x}$ such that $p_{\mathcal{D}}(y \mid \bar{\mathbf{x}})$ is domain-invariant and propose algorithms to infer $\bar{\mathbf{x}}$ given data from multiple source domains.

Despite being (arguably) more trustworthy than purely data-driven approaches, causality-based methods still struggle to be applied in practice. First of all, they depend to some extent on $\mathcal{G}$ being given. In many applications, domain knowledge is insufficient to build such a graph. Moreover, causal discovery algorithms, which aim to learn the structure of the graph from data, are computationally expensive and, given observational data, can only recover $\mathcal{G}$ up to its Markov equivalence class, at least when no parametric assumptions are made (Peters et al. [106]). Furthermore, these methods are unsuitable to be applied to image data as no meaningful causal reasoning can be built in pixel space and even deep neural networks are still incapable of finding suitable representations for this goal (Schölkopf et al. [107]).

3.3 Adversarial domain adaptation for object counting in videos

3.3.1 Motivation

As different sensors are added to and excluded from a network, we should take into account the fact that the sensors used at training time are different from the ones where the model will make predictions. Since domain shifts between different sensors on a

sensor network are usually substantial, it is imperative that robust domain adaptation methods are developed that take into account the constraints of the sensor network.

There is a lack of domain adaptation methods focusing on how to handle the temporal component of data. Chen et al. [108] proposed an algorithm called Temporal Attentive Alignment for implementing DA in video datasets that explicitly attends to temporal dynamics and Liu and Li [109] proposed a spatio-temporal DA model named TrCbrBoost for classifying land use. It should be noted, though, that both of these works only deal with a single-source-single-target setting, which is simpler than the multi-source case present when dealing with sensor networks.

For dealing with a multi-source setting, adversarial approaches have proven successful. Zhao et al. [67] introduced an algorithm called multi-source domain adversarial networks (MDAN) that makes use of k domain discriminators that aim to distinguish between the target and the k source domains. MDAN showed superior performance when compared to other state of the art methods on the task of counting vehicles in images obtained from city cameras videos. Given the positive results obtained, and given the fact that MDAN does not consider the temporal component of the video frames, we consider it is worthy to investigate the adaptation of this model so that it can receive a temporal sequence as an input.

The setting where sensors correspond to video cameras is especially interesting since video data is very high-dimensional. Moreover, the fact that different cameras are located in different places, and therefore have distinct points of view, increases the domain shift and therefore makes this task even more challenging.

Here, we shall explore how to adapt the MDAN model so that both of its adversarial networks are LSTM-based. We decided to go with this type of network since it has already shown promising results in our task: Zhang et al. [110] introduced an LSTM network architecture for counting vehicles in images obtained from city cameras and reported an improvement of the mean absolute error when compared to other state of the art methods.

3.3.2 MDAN: Multi-source domain adversarial networks

We now review the MDAN model introduced by Zhao et al. [67]. This model is motivated by the target risk bound provided in Theorem 3.3 and is an extension to the multi-source setting of the single source model by Ganin and Lempitsky [64]. Specifically, the following

formal objective is considered:

$$\min_{g,h} \max_{j\in\{1,\cdots,k\}} \quad \widehat{\epsilon}_{\mathcal{S}_j}(h\circ g) + \frac{1}{2}\widehat{d}_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S}_j^g, \mathcal{T}^g), \tag{3.11}$$

where $g : \mathcal{X} \mapsto \mathcal{Z}$ and $h : \mathcal{Z} \mapsto \mathcal{Y}$ are, as before, the feature extractor and the classifier networks and $\mathcal{S}_j^g$ and $\mathcal{T}^g$ are, respectively, the j-th source domain and the target domain representations in the new feature space $\mathcal{Z}$ (i.e. $p_{\mathcal{D}^g}(\mathbf{x}) \triangleq p_{\mathcal{D}}(g(\mathbf{x}))$ for any domain $\mathcal{D}$). Here, the empirical $\mathcal{H}\Delta\mathcal{H}$-divergence between $\mathcal{S}_j^g$ and $\mathcal{T}^g$ is also implemented with an adversarial domain discriminator $d_j : \mathcal{Z} \mapsto \{0,1\}$ aiming to discriminate between samples of the two domains. Thus, the model comprises k domain discriminator networks, i.e. one for each source domain. This objective is therefore identical to equation (3.10) with the only difference that, since here there are multiple source domains, the model is optimized for the hardest source domain at each training iteration. In this formulation, $\boldsymbol{\alpha}$ in equation (3.5) is a one-hot vector whose active component corresponds to the hardest source domain. Because the bound holds for any convex combination of source domains, the authors also explore a soft-max version of this problem, where smaller positive weights are assigned to the easier source domains:

$$\min_{g,h} \quad \frac{1}{\gamma}\log\sum_{j=1}^{k}\exp\left(\gamma(\widehat{\epsilon}_{\mathcal{S}_j}(h\circ g) + \frac{1}{2}\widehat{d}_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{S}_j^g, \mathcal{T}^g))\right). \tag{3.12}$$

Here, $\gamma > 0$ is a hyperparameter controlling the softness of the max operation ($\gamma \to \infty$ corresponds to the hard-max). In our experiments, we will also evaluate the scenario where the weights $\boldsymbol{\alpha}$ are equal for all domains, i.e. where the multi-domain loss consists of the simple average of the losses across source domains.

3.3.2.1 The gradient reversal layer

Adversarial DA algorithms, of which MDAN is a particular case, all aim to solve some variant of the following minimax problem:

$$\min_{\boldsymbol{\theta}_g,\boldsymbol{\theta}_h} \max_{\boldsymbol{\theta}_d} \quad \left\{L(\boldsymbol{\theta}_g,\boldsymbol{\theta}_h,\boldsymbol{\theta}_d) = L_{\text{task}}(\boldsymbol{\theta}_g,\boldsymbol{\theta}_h) - \lambda_d L_{\text{disc}}(\boldsymbol{\theta}_g,\boldsymbol{\theta}_d)\right\}, \tag{3.13}$$

where L_{task} is the supervised loss for the desired task, L_{disc} is the classification loss for the domain discrimination task, $\boldsymbol{\theta}_g$, $\boldsymbol{\theta}_h$, and $\boldsymbol{\theta}_d$ are the parameters of the feature extractor, task classifier, and domain discriminator networks, respectively, and $\lambda_d > 0$ is a hyperparameter. This is a treatable surrogate of objective (3.10).

Solving this problem then consists in finding a saddle point of this loss function. Using automatic differentiation libraries (e.g. PyTorch or TensorFlow), the naive solution would involve declaring two optimizers, one for parameters $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_h$ and another for $\boldsymbol{\theta}_d$, and then performing gradient descent with the former and gradient ascent with the latter:

$$\boldsymbol{\theta}_g \leftarrow \boldsymbol{\theta}_g - \rho \left(\nabla_{\boldsymbol{\theta}_g} L_{\text{task}} - \lambda_d \nabla_{\boldsymbol{\theta}_g} L_{\text{disc}} \right), \tag{3.14}$$

$$\boldsymbol{\theta}_h \leftarrow \boldsymbol{\theta}_h - \rho \nabla_{\boldsymbol{\theta}_h} L_{\text{task}}, \tag{3.15}$$

$$\boldsymbol{\theta}_d \leftarrow \boldsymbol{\theta}_d + \rho \left(-\lambda_d \nabla_{\boldsymbol{\theta}_d} L_{\text{disc}} \right), \tag{3.16}$$

where $\rho > 0$ is the learning rate. This implies an extra computational burden because gradients need to be backpropagated through the discriminator network twice, one for computing $\nabla_{\boldsymbol{\theta}_d} L_{\text{disc}}$ and another for computing $\nabla_{\boldsymbol{\theta}_g} L_{\text{disc}}$.

The gradient reversal layer (Ganin and Lempitsky [64]) is an ingenious solution to this problem. This layer is a pseudo-function r that behaves as the identify in the forward pass but inverts the sign of the gradient in the backward, i.e.:

$$r(\boldsymbol{x}) \triangleq \boldsymbol{x}, \quad \frac{\partial r}{\partial \boldsymbol{x}} \triangleq -I. \tag{3.17}$$

By placing it in between the feature extractor g and the domain discriminator d, the gradient $\nabla_{\boldsymbol{\theta}_g} L_{\text{disc}}$ will come with its sign inverted. Thus, performing gradient descent over all parameters of

$$L_{\text{task}}(\boldsymbol{\theta}_g, \boldsymbol{\theta}_h) + \lambda_d L_{\text{disc}}(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d), \tag{3.18}$$

yields exactly update equations (3.14), (3.15), and (3.16).

3.3.3 FCN-rLSTM: Spatio-temporal deep neural network for object counting

Zhang et al. [110] proposed FCN-rLSTM, a deep neural network architecture for counting vehicles in low-quality videos captured by city cameras that will constitute the backbone of our models.

Although each video frame should contain all the information required to identify the number of vehicles in it, issues with the quality of the collected data can make vehicle counting a difficult problem, namely low resolution, vehicle occlusion, and different vehicle scales, particularly noticeable when the camera is too close to the road. Thus, assuming that the frame rate is sufficiently large when compared to the vehicles speed, leveraging

information from the previous frames should help improving the accuracy of the prediction for the current frame. For this reason, FCN-rLSTM combines a fully convolutional network with a recurrent module, which preserves memory from the previous frames in LSTM cells. This is a density-based estimation method, being able to deal well with low frame rates, low resolutions, and vehicle occlusions, but having difficulty in accounting for different vehicle scales.

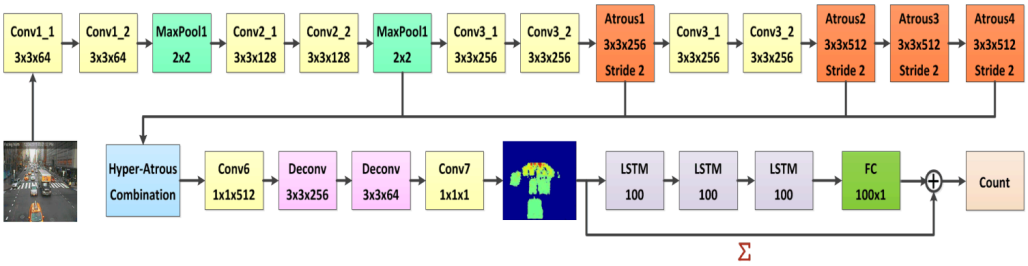

FIGURE 3.5: Architecture of the FCN-rLSTM model (reprinted from Zhang et al. [110]).

Figure 3.5 shows the full architecture of this model, from which it can be observed that the convolutional part outputs a density map $\widehat{\boldsymbol{D}}$. The density maps are normalized so that the sum of the pixels corresponding to each vehicle sum up to 1 and, therefore, the sum of all pixels in the density map sup to the total number of vehicles in the frame. Thus, the final predicted count $\widehat{y}^{(t)}$ is the result of this sum plus a residual provided by a recurrent neural network, which aims to correct the predicted count by leveraging information from the previous frames.

Training this model implies that each input frame $\boldsymbol{X}^{(t)}$ is annotated with the corresponding ground-truth density map $\boldsymbol{D}^{(t)}$ and vehicle count $y^{(t)}$. The loss function is defined as:

$$L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} ||\widehat{\boldsymbol{D}}_i^{(t)} - \boldsymbol{D}_i^{(t)}||_F^2 + \frac{\lambda_c}{n}\sum_{i=1}^{n} ||\widehat{y}_i^{(t)} - y_i^{(t)}||^2, \tag{3.19}$$

where $||\cdot||_F$ is the Frobenius norm, $\lambda_c > 0$ is a hyperparameter controlling the relative weight of the vehicle counting loss, and the dependency of $\widehat{\boldsymbol{D}}_i^{(t)}$ and $\widehat{y}_i^{(t)}$ on the network parameters $\boldsymbol{\theta}$ is omitted to ease the notation.

3.3.4 Combining MDAN and FCN-rLSTM

We now explore several possibilities to combine MDAN and FCN-rLSTM into a single model capable of accurately counting vehicles in images from a target camera, provided that at training time we only have access to annotated data from other cameras and unlabeled data from the target.

3.3.4.1 Non-temporal model

The non-temporal model uses a sub-network in the FCN-rLSTM model as its backbone. It consists in a simplification of this, by not using LSTMs or making any other consideration on the temporal or sequential nature of the data. Figure 3.6 shows the architecture of the non-temporal model.

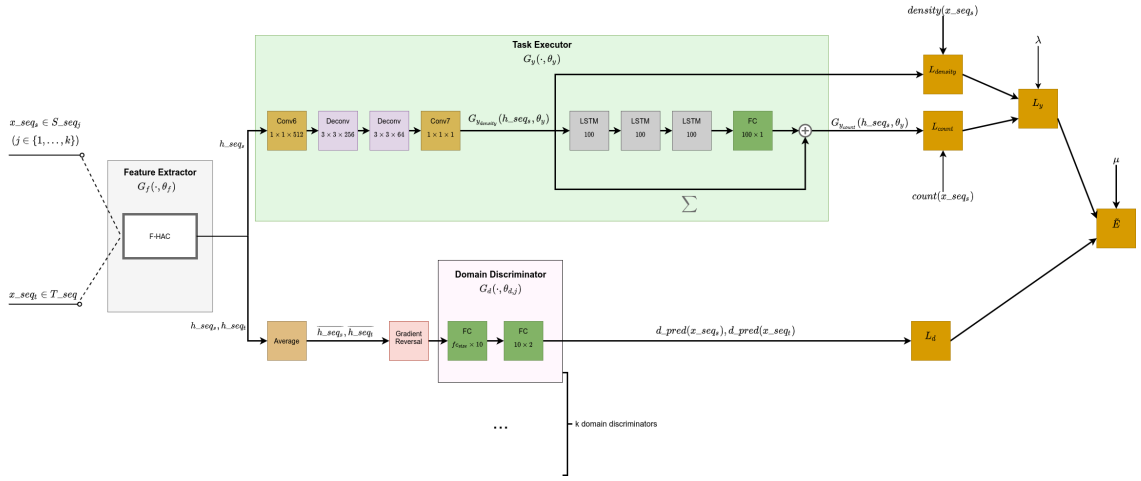

FIGURE 3.6: Non-temporal model. Reprinted from de Andrade [56].

The component F-HAC in that figure corresponds to all layers of the FCN-rLSTM up to the hyper-atrous combination. It is used as a feature extractor, whereas the rest of the convolutional layers are used for the object counting task. The k domain discriminators consist of two fully connected layers which are preceded by a gradient reversal layer.

This model will be our baseline as we are primarily interested in merging the benefits of DA techniques and sequential modeling.

3.3.4.2 SingleLSTM model

The SingleLSTM model uses the whole FCN-rLSTM model for the desired regression task, but the domain discriminators are still non-sequential, as shown in Figure 3.7. Thus, in this model, the domain discrimination task only takes into account the domain-specific information provided by each frame individually and does not account for the domain-specific temporal dynamics that may exist.

3.3.4.3 DoubleLSTM model

The DoubleLSTM model overcomes the limitations of the temporal regression model by incorporating three LSTM layers in each domain discriminator. These aim to learn the

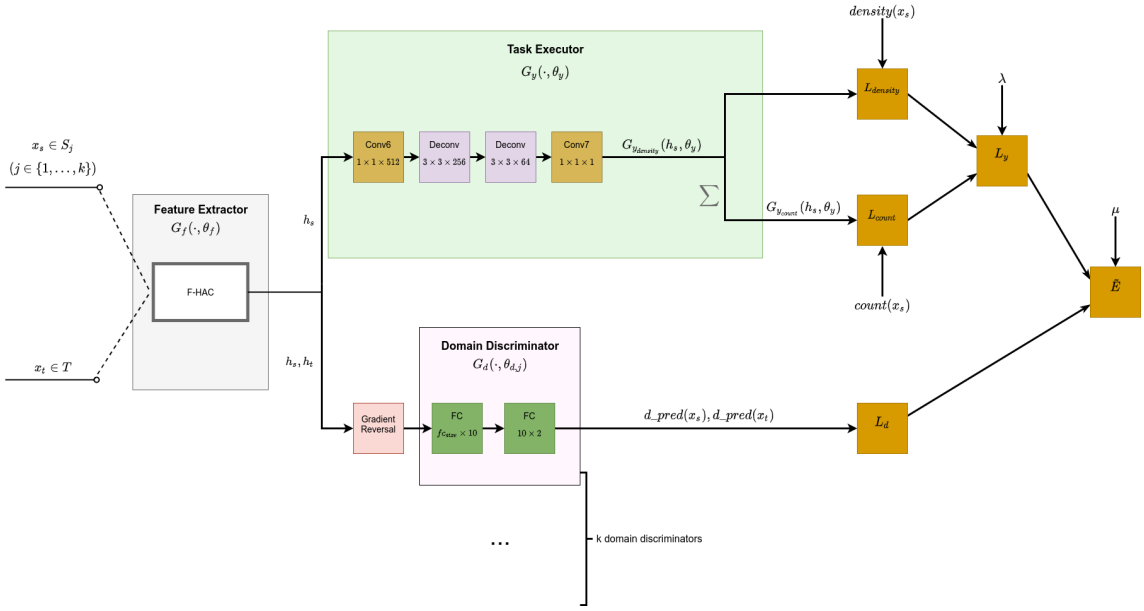

FIGURE 3.7: Temporal regression model. Reprinted from de Andrade [56].

domain-specific temporal dynamics that might be helpful for the discrimination task. The schematic is in Figure 3.8.

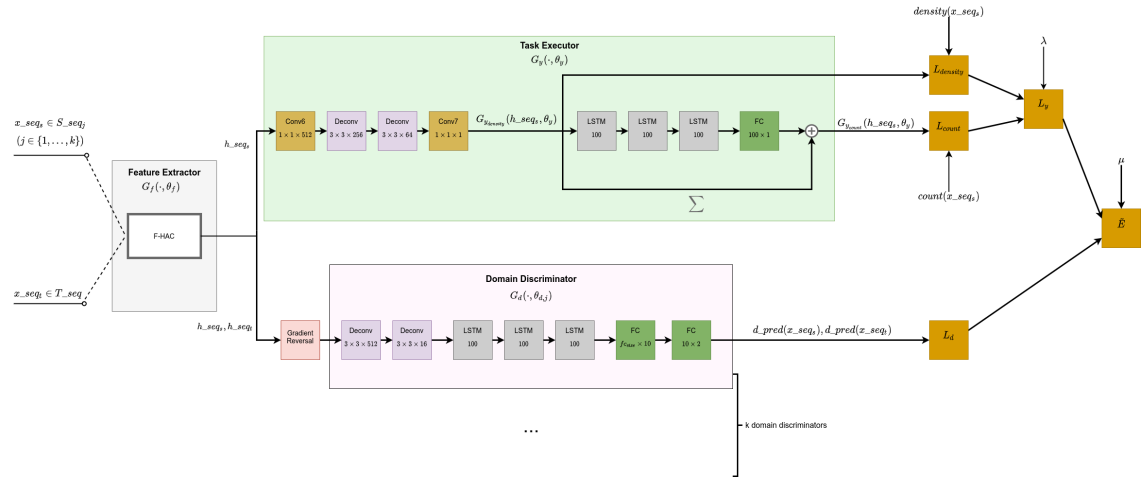

FIGURE 3.8: Double-Temporal model. Reprinted from de Andrade [56].

3.3.4.4 CommonLSTM model

An alternative to the DoubleLSTM model is to extract temporal features that are common to the desired object counting task and to the domain discrimination task. This is accomplished by the CommonLSTM model, which pushes the domain discriminators after the LSTMs and just before the final fully connected layer, as shown in Figure 3.9.

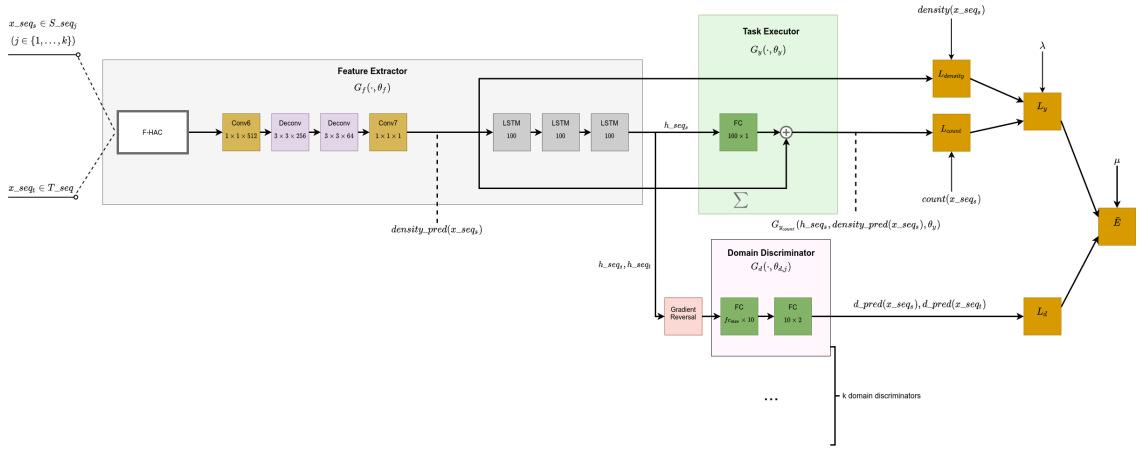

FIGURE 3.9: Common-Temporal model. Reprinted from de Andrade [56].

This model is fundamentally different from all previous ones since here the feature space where domain-invariance is promoted happens much later in the network. Specifically, the density maps $\widehat{D}$ are a few layers earlier, so they should not be very much affected by the domain-invariance constraint. This might have a beneficial effect on the performance since density maps are defined by the positions in the image where vehicles appear. Since these positions depend on the shape of the street that each camera is capturing, domain-invariant density maps will almost surely be inaccurate.

3.3.4.5 Overview

All the proposed models have specific strengths and weaknesses that would make them suitable for certain scenarios and unlikely to show a good performance in others. Here, we anticipate a few of those scenarios.

If the video frame rate is low compared to the objects speed, the number of objects in a given frame is not a good predictor for the number of objects in a subsequent frame. In this setting, the non-temporal model is ideal, as it does not make any temporal consideration. In this case, using a temporal model would probably just upset the training process.

If the frame rate is sufficiently high and the temporal dynamics in the target domain are close to the dynamics in the sources, the SingleLSTM model is likely the best choice. By using an LSTM for the task executor but none for the domain discriminators, it will be able to make small adjustments in the predicted object count and avoid the unnecessary computational cost introduced by having a sequential model in the domain discriminator.

When there is a strong correlation in the object count in consecutive video frames and the dynamics in the target domain are dissimilar to the sources, it may be beneficial to

employ sequential models in both the task executor and domain discriminators. Thus, DoubleLSTM and CommonLSTM models are likely to outperform the remaining. Their approach differs, as DoubleLSTM model uses an LSTM network for the task executor and another one for each domain discriminator, whereas CommonLSTM includes an LSTM network in the feature extractor. As the common-temporal model will not enforce similarity between the predicted density maps as much as the double-temporal version, it is probably the best choice when the density maps of the target domain differ significantly from the sources.

3.3.5 Experiments

3.3.5.1 Experimental protocol

We will run experiments with networks FCN-HA and FCN-rLSTM, proposed by Zhang et al. [110] in paper "FCN-rLSTM: Deep Spatio-Temporal Neural Networks for Vehicle Counting in City Cameras", described in detail in section **??**. The FCN-HA does not consider the temporal nature of data, whereas the FCN-rLSTM does, and neither of them . Additionally, we will run experiments with the models presented in Section 3.3.4.

In every experiment, assume we have $k+1$ domains $D_1, D_2, ..., D_{k+1}$. In a domain adaptation scenario, we do $k+1$ runs, so that in run number t, domain D_t will be chosen as the target domain and all the others will be source domains.

For every experiment, we have computed results in both unsupervised and semi-supervised settings, described below:

- **Unsupervised:** In this case, we do not have a validation dataset, and, for each run t, we calculate the testing results with the model we obtained at the end of the training epochs, for all the samples extracted from domain D_t.
- **Semi-supervised:** For semi-supervised results, we have a validation dataset, that corresponds to 30% of the samples extracted from target D_t. We use those samples to select the best epoch obtained throughout training. The model obtained in that epoch will be used to test the remaining 70% of the samples in D_t.

For each dataset, we ran a total of 5 experiments for the DA models, where we varied the value of hyperparameter λ_d in the range $[10^{-5}, 10^{-1}]$. Recall that λ_d controls the weight to give to the domain discrimination loss, relative to the task execution loss.

In all experiments, we train the model for 50 epochs, using Adam optimizer (Kingma and Ba [41]) with a learning rate of 10^{-4}. The adopted evaluation metric is the Mean Absolute Error (MAE) between the predicted object count and the ground truth for each frame.

3.3.5.2 WebCamT dataset

WebCamT (Zhang et al. [111]) is a vehicle counting dataset that contains city camera videos, taken from a stationary camera in different points of the city of New York. All videos are in color, with dimensions 240×352, at a 1 frame per second rate. Every frame in the dataset has a ground truth file with annotations that indicate the center of each vehicle and also its bounding box.

There are a total of 14 cameras, that cover multiple scenes, camera perspectives, congestion states and weather conditions. We chose a total of 4 cameras from those 14 to correspond to our domains. Each of those 4 cameras had a number of videos, from which we selected 1000 frames. The frames were selected consecutively, so that if frames f_1 and f_2 from the same video V were selected, then every frame between f_1 and f_2 in video V was also selected.

Figures 3.10-3.13 show examples of frames in each one of the four domains. Table 3.1 shows the mean and standard deviation for the number of vehicles in a frame, for each domain. As it is possible to observe, the mean number of vehicles in each frame differs significantly across domains, being more than three times larger in domain *691* than in domain *846*. Hence, there is significant target shift in this dataset and, consequently, the covariate shift assumption does not hold.

Domain	Mean	Std
511	12.039	6.32
551	18.362	4.21
691	25.470	10.738
846	6.873	2.477

TABLE 3.1: Mean number of vehicles and respective standard deviation for each domain in the WebCamT dataset.

After extracting each frame from the videos, we computed its density map, by placing a 4×4 Gaussian kernel with sum 1 on the center of each car. We then had to resize the frames and density to maps to size 120×176 so that we could deal with speed and

FIGURE 3.10: Domain *511* from WebCamT dataset (Zhang et al. [111]).

FIGURE 3.11: Domain *551* from WebCamT dataset (Zhang et al. [111]).

FIGURE 3.12: Domain *691* from WebCamT dataset (Zhang et al. [111]).

FIGURE 3.13: Domain *846* from WebCamT dataset (Zhang et al. [111]).

memory constraints. Figure 3.14 shows an example of a density map in a resized frame, where the Gaussian kernel around each car is shown in red.

FIGURE 3.14: Example density map for WebCamT dataset

3.3.6 Choice of optimization problem

As in each run of the experiment there are three different source domains, this is a multi-source scenario. As such, there are three possible optimization problem formulations for our DA models to solve, namely hard-max, soft-max, and average, as described in section 3.3.2. Table 3.2 shows results for each of our models on these formulations, evaluated in an unsupervised setting. These suggest that averaging performs consistently better than the other two approaches in this dataset. This is somewhat intuitive: given the wide range of values for the mean number of vehicles in each frame across domains, it would be likely for one source domain to show a significantly higher loss than the others. If we use any of the other two optimization problems, our models would dedicate an outweighed importance to the hardest source domain, disregarding the other domains. For this reason, we shall adopt the averaging formulation in all subsequent experiments.

	Optimization Problem		
Model	Hard-max	Soft-max	Average
Non-temporal	18.146	11.160	9.846
SingleLSTM	10.543	9.326	10.532
DoubleLSTM	9.589	7.731	6.148
CommonLSTM	9.847	12.069	8.146

TABLE 3.2: Comparison of different optimization problems. The table indicates the Avg. MAE Count across domains. The experiments were run with $\lambda_d = 10^{-3}$.

3.3.7 Unsupervised Setting

Table 3.3 shows the results in the unsupervised setting. For the domain adaptation models (non-temporal, temporal regression, double-temporal and common-temporal), columns *511*, *551*, *691* and *846* indicate the best MAE Count obtained for the respective domain in the 5 experiments with the different values of λ_d indicated before. Column Avg indicates the best average of the MAE Count across domains *511*, *551*, *691* and *846* in the 5 experiments. That is, for each experiment, we compute the average MAE Count across the 4 domains. We then select the best of the computed average MAE Count.

For the average of the MAE Count across domains, the DoubleLSTM model showed the best results with a MAE of 6.148, followed by the FCN-HA with a MAE of 7.370. In this dataset, the temporal models did not seem to perform particularly better than the non-temporal ones. Even though the double-temporal showed the best results, the second best was the FCN-HA, a model that does not make any temporal consideration. We also

	Domain				
Model	*511*	*551*	*691*	*846*	Avg
FCN-HA	**5.006**	**2.577**	17.224	4.674	7.370
FCN-rLSTM	6.26	8.789	13.632	6.305	8.746
Non-temporal	9.629	3.497	18.271	5.581	9.846
SingleLSTM	6.579	5.100	15.264	9.553	10.532
DoubleLSTM	5.013	4.576	10.973	**3.029**	**6.148**
CommonLSTM	6.106	5.552	**10.508**	3.564	8.146

TABLE 3.3: MAE Count by domain. For each domain, the best MAE obtained from experiments run with different values of λ_d is shown. Column Avg indicates the best average of the MAE Count for domains *511*, *551*, *691* and *846* in experiments run with different values of λ_d.

could not verify a marked difference in the average MAE count of the DA and non-DA methods. These observations ascertain that it is not enough to apply domain adaptation or consider the temporal nature of data to obtain good results. Thus, the careful design of models and its integration with the two mentioned techniques is essential.

When it comes to the accuracy of models across different domains, we find that domain *691* has the worst results, since it has the most dissimilar vehicle count distribution (highest mean and also highest standard deviation), which cannot be matched by any mixture of the source vehicle count distributions. It is observable that the performance of models across different domains is not consistent. For example, the FCN-HA model shows the best results for domains *511* and *551*, but the second worst result for domain *691*.

We find that the model with the most balanced performance across domains was the DoubleLSTM as its MAE Count in a given model was either the best or the second best. This coupled with the fact that the DoubleLSTM was also the model with the best average MAE count allows us to conclude that this method was the best-performing in the WebCamT dataset.

Figure 3.15 shows the average MAE count across domains for every DA model experiment, run with a different value of λ_d.

This graph confirms that the model DoubleLSTM was the best-performing in the WebCamT dataset. The SingleLSTM, on the other hand, showed the worst results, even when comparing it against Simple Model, a non-temporal method. It is also noticeable that model DoubleLSTM has the best accuracy for the intermediate value of $\lambda_d = 10^{-3}$. On the other hand, CommonLSTM shows a tendency to improve the MAE Count as the λ_d decreases.

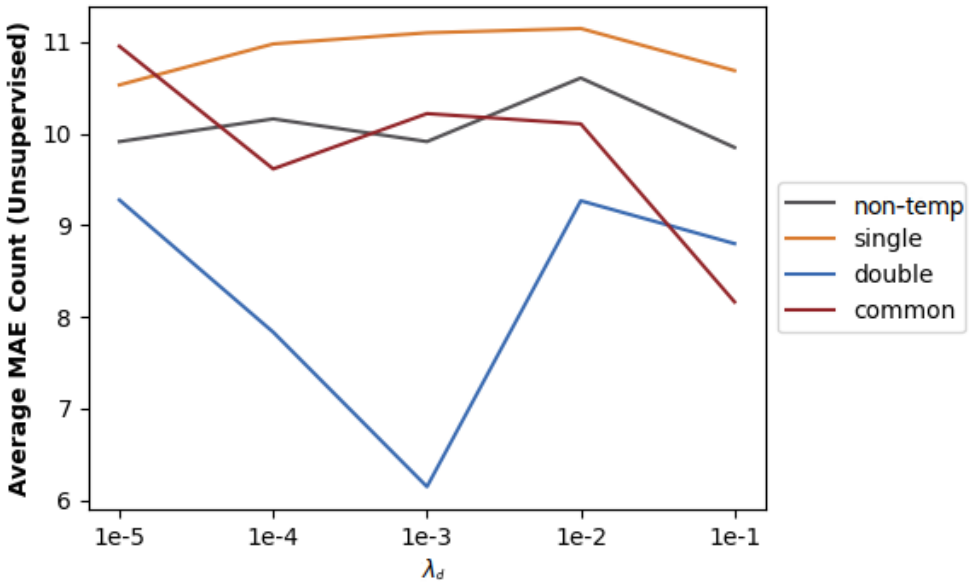

FIGURE 3.15: Avg. MAE Count across domains for every λ_d (unsupervised setting).

3.3.8 Semi-supervised Setting

Table 3.4 shows the results in the semi-supervised setting. Like in the table 3.3, for the domain adaptation models (Simple, SingleLSTM, DoubleLSTM and CommonLSTM), columns *511*, *551*, *691* and *846* indicate the best MAE Count obtained in the 5 experiments with the different values of λ_d indicated before. Column Avg indicates the best average of the MAE Count for domains *511*, *551*, *691* and *846* in the 5 experiments.

	Domain				
Model	*511*	*551*	*691*	*846*	Avg
FCN-HA	**2.050**	**2.589**	6.560	2.394	**3.398**
FCN-rLSTM	5.400	3.486	9.153	3.537	5.394
Simple	3.112	3.565	10.964	2.049	5.035
SingleLSTM	4.239	4.393	7.032	**1.520**	4.327
DoubleLSTM	4.223	4.552	**6.456**	1.583	4.271
CommonLSTM	4.227	4.543	8.668	1.634	4.984

TABLE 3.4: MAE Count by domain. For each domain, the best MAE obtained from experiments run with different values of λ_d is shown. Column Avg indicates the best average of the MAE Count for domains *511*, *551*, *691* and *846* in the experiments run with different values of λ_d.

There is a noticeable difference between the semi-supervised results and the unsupervised ones, with the former being significantly better that the latter. As the experiments we ran with the WebCamT had more data and more domains, that led to a reduction in the instability of the models, which caused the difference in accuracy between the two settings to not be as wide.

Like in the unsupervised results, we were not able to notice a significant difference between the performance of the temporal methods and the non-temporal ones. As we have also verified before, the DA models do not show a marked improvement over the non-DA models for the WebCamT dataset. In fact, in this case, it was a non-temporal, non-DA model, the FCN-HA, that showed the best average MAE count.

Figure 3.16 shows the average MAE count across domains for every DA model experiment, run with a different value of λ_d.

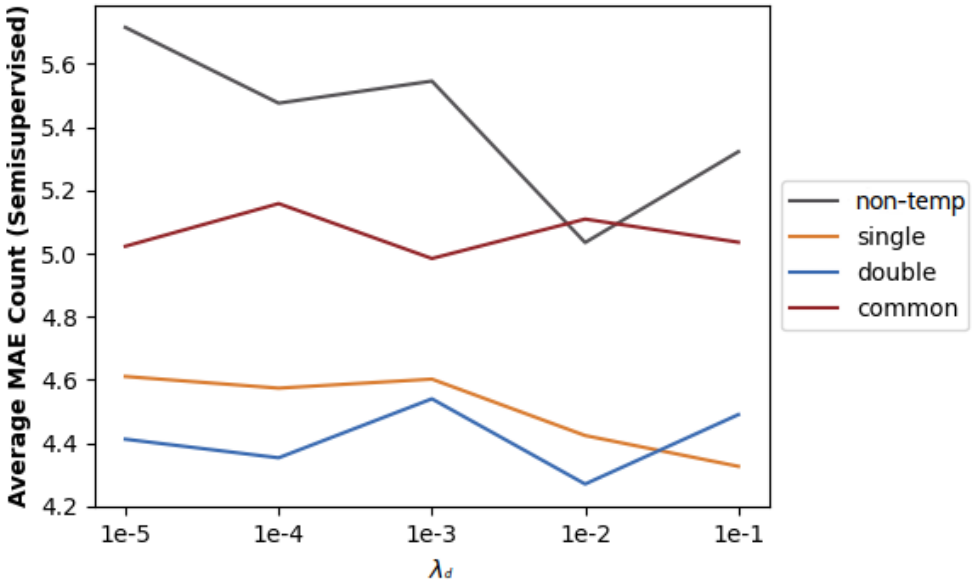

FIGURE 3.16: Avg. MAE Count across domains for every λ_d (semi-supervised setting).

The DoubleLSTM had, in general, the best average MAE count in the semi-supervised graph, similar to what it was verified in the unsupervised case. The Simple Model was the worst, unlike in the unsupervised setting, where it was the SingleLSTM that showed the worst results.

The average MAE count across different values of λ_d is considerably more stable in the semi-supervised results than in the unsupervised ones. We were not able to notice an improvement in the performance of the CommonLSTM as λ_d decreased, just like we were not able observe a notorious lower average MAE count in the DoubleLSTM model, for $\lambda_d = 10^{-3}$. Because the unsupervised results are more unstable, they are also more sensible to variations of the value of λ_d that are not discernible in the semi-supervised graph.

3.3.8.1 Discussion

It stands out that models DoubleLSTM and CommonLSTM showed a better performance than the state-of-art method for counting objects FCN-rLSTM in every dataset and in every setting (unsupervised and semi-supervised). In particular, when it comes to the average MAE count in the unsupervised results, the DoubleLSTM model showed an average improvement of 27% , whereas the CommonLSTM model showed an average improvement of 17%, when compared against the FCN-rLSTM. As such, these results underline the contribution this dissertation made to the problem of adversarial domain adaptation in sensor networks.

Models Simple and SingleLSTM, on the other hand, had a very unsatisfactory performance, by not showing better results than methods FCN-rLSTM and FCN-HA, even though they leveraged the power of domain adaptation and, in the case of SingleLSTM, the power of LSTMs to make temporal considerations about the nature of data. When it comes to the Non-temporal model, it appears the reason for its low accuracy is the fact that, without the power of LSTMs, applying domain adaptation to the FCN-HA just disturbs the training process. As for the SingleLSTM, having a too big of a difference in the complexity of its task executor and its domain discriminator is the most likely cause for its failure to show improvements relative to the state-of-art methods.

We have thus verified that integrating domain adaptation with a method does not automatically guarantee better results. Careful considerations on how to divide the previous method between feature extractor and task executor should be made, just like careful deliberations on the domain discriminator design.

It should also be noticed that the results showed an inconsistency in the performance of models across domains. That is, whereas a model performed better in one domain, another model performed better in another. Hence, when applying a model for counting objects in a real world scenario, one should ponder what model to use depending on the characteristics of the particular target domain.

Please refer to de Andrade [56] for an extended discussion and further experimental results.

Appendix A

SpaMHMM – supplementary material

A.1 Derivation of the EM learning algorithms for MHMM and SpaMHMM

A.1.1 EM for MHMM (Algorithm 2.3)

Algorithm 2.3 follows straightforwardly from applying EM to the model defined by equations (2.18) and (2.19) with the objective (2.20). As explained in section **??**, for table CPDs, the E-step consists in finding expected sufficient statistics $M(x,u)$ and $M(u)$, as defined in equations (**??**) and (**??**) for each $(x,u) \in \mathrm{Val}(\mathrm{x}, Pa_{\mathcal{G}}(\mathrm{x}))$, where $\mathcal{G}$ is the Bayesian network in Figure 2.5. Thus, in our setting, those equations yield:

$$M(z,y) = \sum_{i=1}^{n} p(y,z \mid \boldsymbol{x}_i, y_i) = \sum_{i=1}^{n} p(z \mid \boldsymbol{x}_i, y_i)\mathbf{1}_{y_i=y}, \tag{A.1}$$

$$M(y) = \sum_{i=1}^{n} p(y \mid \boldsymbol{x}_i, y_i) = \sum_{i=1}^{n} \mathbf{1}_{y_i=y}, \tag{A.2}$$

$$\begin{aligned} M(\mathrm{h}^{(0)} = h, z) &= \sum_{i=1}^{n} p(\mathrm{h}^{(0)} = h, z \mid \boldsymbol{x}_i, y_i) = \sum_{i=1}^{n} p(\mathrm{h}^{(0)} = h \mid \boldsymbol{x}_i, y_i, z)p(z \mid \boldsymbol{x}_i, y_i) \\ &= \sum_{i=1}^{n} p(\mathrm{h}^{(0)} = h \mid \boldsymbol{x}_i, y_i)p(z \mid \boldsymbol{x}_i, y_i), \end{aligned} \tag{A.3}$$

$$M(z) = \sum_{i=1}^{n} p(z \mid \boldsymbol{x}_i, y_i), \tag{A.4}$$

$$M(\mathrm{h}^{(t)} = h', \mathrm{h}^{(t-1)} = h, z) = \sum_{i=1}^{n} p(\mathrm{h}^{(t)} = h', \mathrm{h}^{(t-1)} = h, z \mid \boldsymbol{x}_i, y_i)$$

$$= \sum_{i=1}^{n} p(\mathrm{h}^{(t)} = h' \mid \mathrm{h}^{(t-1)} = h, \boldsymbol{x}_i, z) p(\mathrm{h}^{(t-1)} = h \mid \boldsymbol{x}_i, z) p(z \mid \boldsymbol{x}_i, y_i), \quad \text{(A.5)}$$

$$M(\mathrm{h}^{(t-1)} = h, z) = \sum_{i=1}^{n} p(\mathrm{h}^{(t-1)} = h, z \mid \boldsymbol{x}_i, y_i)$$

$$= \sum_{i=1}^{n} p(\mathrm{h}^{(t-1)} = h \mid \boldsymbol{x}_i, z) p(z \mid \boldsymbol{x}_i, y_i). \quad \text{(A.6)}$$

These equations allow us to compute the following updates in the M-step:

$$\alpha_z^{(y)} = \frac{M(y, z)}{M(y)} \quad \text{and} \quad \pi_h^{(z)} = \frac{M(\mathrm{h}^{(0)} = h, z)}{M(z)}. \quad \text{(A.7)}$$

For the state transition matrices update, the same idea applies after summing over the sequence length:

$$A_{h,h'}^{(z)} = \frac{\sum_{t=1}^{T} M(\mathrm{h}^{(t)} = h', \mathrm{h}^{(t-1)} = h, z)}{\sum_{t=1}^{T} M(\mathrm{h}^{(t-1)} = h, z)}. \quad \text{(A.8)}$$

Now, defining n_y, $\eta_i^{(z)}$, $\gamma_{i,h}^{(z)}(t)$, and $\xi_{i,h,h'}^{(z)}(t)$ as in Algorithm 2.3, the update formulas for $\alpha_z^{(y)}$, $\pi_h^{(z)}$, and $A_{h,h'}^{(z)}$ become as defined in that algorithm.

Deriving the update equations for the means and covariances of the Gaussian emission distributions is not so simple, since these are not table CPDs and therefore the procedure we have just used is no longer valid. Nonetheless, the desired equations can be obtained by explicitly maximizing the ELBO with respect to these parameters. As we have seen in section **??**,

$$\text{ELBO} = \sum_{i=1}^{n} \mathbb{E}_{(\mathbf{h}_i, \mathrm{z}_i) \sim q} \log p(\boldsymbol{X}_i, \mathbf{h}_i, \mathrm{z}_i \mid y_i; \Theta), \quad \text{(A.9)}$$

where $q(\mathbf{h}_i, \mathrm{z}_i) = p(\mathbf{h}_i, \mathrm{z}_i \mid \boldsymbol{x}_i, y_i; \Theta^{(-)})$, being $\Theta^{(-)}$ the set of current values for all parameters. We are solely interested in maximizing the ELBO with respect to the parameters $\boldsymbol{\mu}_h^{(z)}$ and $\boldsymbol{\sigma}_h^{(z)}$, so we can plug in the expression for the joint $p(\boldsymbol{x}_i, \mathbf{h}_i, \mathrm{z}_i \mid y_i; \Theta)$ and ignore all terms that do not depend on these parameters:

$$\text{ELBO} = \sum_{i=1}^{n} \sum_{\mathbf{h}_i, \mathrm{z}_i} q(\mathbf{h}_i, \mathrm{z}_i) \log \left[p(\mathrm{z}_i \mid y_i) p(\mathrm{h}_i^{(0)} \mid \mathrm{z}_i) \prod_t p(\mathrm{h}_i^{(t)} \mid \mathrm{h}_i^{(t-1)}, \mathrm{z}_i) p(\boldsymbol{x}_i^{(t)} \mid \mathrm{h}_i^{(t)}, \mathrm{z}_i) \right]$$

$$= \sum_{i=1}^{n} \sum_{\mathrm{h}_i^{(t)}, \mathrm{z}_i} \sum_{t=1}^{T_i} q(\mathrm{h}_i^{(t)}, \mathrm{z}_i) \log p(\boldsymbol{x}_i^{(t)} \mid \mathrm{h}_i^{(t)}, \mathrm{z}_i) + \text{const.} \quad \text{(A.10)}$$

Now, all that remains is computing the gradient of the ELBO with respect to the parameters and solving for the critical points:

$$\begin{aligned}\nabla_{\boldsymbol{\mu}_h^{(z)}}\text{ELBO} &= \sum_{i=1}^{n}\sum_{t=1}^{T_i}\frac{q(\mathrm{h}_i^{(t)}=h,z)}{p(\boldsymbol{x}_i^{(t)}\mid \mathrm{h}_i^{(t)}=h,z)}\nabla_{\boldsymbol{\mu}_h^{(z)}}p(\boldsymbol{x}_i^{(t)}\mid \mathrm{h}_i^{(t)}=h,z)\\ &= \sum_{i=1}^{n}\sum_{t=1}^{T_i} q(\mathrm{h}_i^{(t)}=h,z)\frac{\boldsymbol{x}_i^{(t)}-\boldsymbol{\mu}_h^{(z)}}{{\boldsymbol{\sigma}_h^{(z)}}^2}, \qquad \text{(A.11)}\\ \nabla_{\boldsymbol{\sigma}_h^{(z)}}\text{ELBO} &= \sum_{i=1}^{n}\sum_{t=1}^{T_i}\frac{q(\mathrm{h}_i^{(t)}=h,z)}{p(\boldsymbol{x}_i^{(t)}\mid \mathrm{h}_i^{(t)}=h,z)}\nabla_{\boldsymbol{\sigma}_h^{(z)}}p(\boldsymbol{x}_i^{(t)}\mid \mathrm{h}_i^{(t)}=h,z)\\ &= \sum_{i=1}^{n}\sum_{t=1}^{T_i} q(\mathrm{h}_i^{(t)}=h,z)\left(\frac{(\boldsymbol{x}_i^{(t)}-\boldsymbol{\mu}_h^{(z)})^2}{{\boldsymbol{\sigma}_h^{(z)}}^3}-\frac{1}{\boldsymbol{\sigma}_h^{(z)}}\right), \qquad \text{(A.12)}\end{aligned}$$

where vector division and exponentiation should be interpreted as elementwise operations. Finally, solving for the critical points yields:

$$\begin{aligned}\boldsymbol{\mu}_h^{(z)} &= \frac{\sum_{i=1}^{n}\sum_{t=1}^{T_i} q(\mathrm{h}_i^{(t)}=h,z)\boldsymbol{x}_i^{(t)}}{\sum_{i=1}^{n}\sum_{t=1}^{T_i} q(\mathrm{h}_i^{(t)}=h,z)}\\ &= \frac{\sum_{i=1}^{n} p(z\mid \boldsymbol{X}_i,y_i;\Theta^{(-)})\sum_{t=1}^{T_i} p(\mathrm{h}^{(t)}=h\mid \boldsymbol{X}_i,z;\Theta^{(-)})\boldsymbol{x}_i^{(t)}}{\sum_{i=1}^{n} p(z\mid \boldsymbol{X}_i,y_i;\Theta^{(-)})\sum_{t=1}^{T_i} p(\mathrm{h}^{(t)}=h\mid \boldsymbol{X}_i,z;\Theta^{(-)})}, \qquad \text{(A.13)}\\ {\boldsymbol{\sigma}_h^{(z)}}^2 &= \frac{\sum_{i=1}^{n}\sum_{t=1}^{T_i} q(\mathrm{h}_i^{(t)}=h,z)(\boldsymbol{x}_i^{(t)}-\boldsymbol{\mu}_h^{(z)})^2}{\sum_{i=1}^{n}\sum_{t=1}^{T_i} q(\mathrm{h}_i^{(t)}=h,z)}\\ &= \frac{\sum_{i=1}^{n} p(z\mid \boldsymbol{X}_i,y_i;\Theta^{(-)})\sum_{t=1}^{T_i} p(\mathrm{h}^{(t)}=h\mid \boldsymbol{X}_i,z;\Theta^{(-)})(\boldsymbol{x}_i^{(t)}-\boldsymbol{\mu}_h^{(z)})^2}{\sum_{i=1}^{n} p(z\mid \boldsymbol{X}_i,y_i;\Theta^{(-)})\sum_{t=1}^{T_i} p(\mathrm{h}^{(t)}=h\mid \boldsymbol{X}_i,z;\Theta^{(-)})}. \qquad \text{(A.14)}\end{aligned}$$

Again, the formulas in Algorithm 2.3 follow by plugging $\eta_i^{(z)}$ and $\gamma_{i,h}^{(z)}(t)$ into the expressions above. □

A.1.2 EM for SpaMHMM (Algorithm 2.4)

We start by showing that the E-step is unaffected by the regularization term in equation (2.21) and then we will derive the update equations for the M-step.

As is a common practice in EM and variational methods, let $q(\mathrm{z},\mathbf{h})$ be an arbitrary distribution over the hidden variables of our model whose support is contained in the

support of p. Then,

$$J_r(\Theta) = \frac{1}{n}\sum_{i=1}^{n} \log \mathbb{E}_{(\mathbf{z}_i,\mathbf{h}_i)\sim q}\left[\frac{p(\boldsymbol{X}_i \mid y_i;\Theta)}{q(\mathbf{z}_i,\mathbf{h}_i)}\right] + \lambda R(\mathcal{G};\Theta)$$
$$\geq \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{(\mathbf{z}_i,\mathbf{h}_i)\sim q} \log\left[\frac{p(\boldsymbol{X}_i \mid y_i;\Theta)}{q(\mathbf{z}_i,\mathbf{h}_i)}\right] + \lambda R(\mathcal{G};\Theta). \tag{A.15}$$

where the inequality is a direct result of Jensen's inequality and $R(\mathcal{G};\Theta)$ is our regularization term, as defined in equation (2.21). We are therefore interested in maximizing this lower bound with respect to both Θ and q. Since $R(\mathcal{G};\Theta)$ does not depend on q, maximization with respect to this distribution while fixing $\Theta = \Theta^{(-)}$ yields $q(\mathbf{z}_i,\mathbf{h}_i) = p(\mathbf{z}_i,\mathbf{h}_i \mid \boldsymbol{X}_i, y_i;\Theta^{(-)})$, as for the case of standard (i.e. unregularized) EM. Thus,

$$J_r(\Theta) \geq \frac{1}{n}\text{ELBO}(\Theta) + \lambda R(\mathcal{G};\Theta) + \text{const}, \tag{A.16}$$

where the ELBO takes the exact same form as in equation (A.9).

Now all that remains is the maximization of this lower bound with respect to Θ. After noting that some parameters are constrained to sum up to 1, we could build the following Lagrangian and solve for the critical points:

$$L(\Theta,\Lambda) = \frac{1}{n}\text{ELBO}(\Theta) + \lambda R(\mathcal{G};\Theta) + \sum_{y=1}^{k}\mu_y\left(1 - ||\boldsymbol{\alpha}^{(y)}||_1\right)$$
$$+ \sum_{z=1}^{m}\sum_{h=1}^{s}\nu_h^{(z)}\left(1 - \sum_{h'=1}^{s}A_{h,h'}^{(z)}\right) + \sum_{z=1}^{m}\varrho_z\left(1 - ||\boldsymbol{\pi}_z||_1\right), \tag{A.17}$$

where $\Lambda \triangleq \bigcup_{h,y,z}\{\mu_y, \nu_h^{(z)}, \varrho_z\}$ is the set of all Lagrange multipliers. Since $R(\mathcal{G};\Theta)$ only depends on the mixture coefficients $\boldsymbol{\alpha}^{(y)}$, the update equations for all parameters except those are exactly the same as in Algorithm 2.3.

Unfortunately, setting $\nabla L = 0$ yields a system of equations that is non-linear in $\boldsymbol{\alpha}^{(y)}$ and no analytical solution can be found. However, we can resort to the reparametrization defined in equation (2.24) and update the new parameters $\boldsymbol{\beta}^{(y)}$ by performing gradient ascent over the unconstrained lower bound. Thus, computing the required gradient concludes the derivation of the algorithm.

Now we only care about the dependency of the ELBO on the mixture coefficients, so, following the same idea as in equation (A.10), we can write:

$$\begin{aligned}\text{ELBO} &= \frac{1}{n}\sum_{i=1}^{n}\sum_{z_i} q(z_i)\log p(z_i \mid y_i) + \text{const} \\ &= \frac{1}{n}\sum_{i=1}^{n}\sum_{z=1}^{m} q(z)\log \alpha_z^{(y_i)} + \text{const},\end{aligned} \tag{A.18}$$

and therefore,

$$\frac{\partial \text{ELBO}}{\partial \alpha_z^{(y)}} = \frac{1}{n}\sum_{i=1}^{n}\frac{q(z)}{\alpha_z^{(y)}}\mathbf{1}_{y_i=y}, \quad \frac{\partial R}{\partial \alpha_z^{(y)}} = \frac{1}{2}\sum_{\substack{y'=1,\\ y'\neq y}}^{k} G_{y,y'}\alpha_z^{(y')}. \tag{A.19}$$

Finally, by the chain rule,

$$\begin{aligned}\frac{\partial \text{ELBO}}{\partial \beta_z^{(y)}} &= \sum_{z'=1}^{m}\frac{\partial \text{ELBO}}{\partial \alpha_{z'}^{(y)}}\frac{\partial \alpha_{z'}^{(y)}}{\partial \beta_z^{(y)}} \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(q(z) - \alpha_z^{(y)}\right)\mathbf{1}_{y_i=y\wedge\beta_z^{(y)}>0} \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(p(z \mid \boldsymbol{X}_i, y_i; \Theta^{(-)}) - \alpha_z^{(y)}\right)\mathbf{1}_{y_i=y\wedge\beta_z^{(y)}>0},\end{aligned} \tag{A.20}$$

$$\begin{aligned}\frac{\partial R}{\partial \beta_z^{(y)}} &= \sum_{z'=1}^{m}\frac{\partial R}{\partial \alpha_{z'}^{(y)}}\frac{\partial \alpha_{z'}^{(y)}}{\partial \beta_z^{(y)}} \\ &= \alpha_z^{(y)}\sum_{\substack{y'=1,\\ y'\neq y}}^{k} G_{y,y'}\left(\alpha_z^{(y')} - \boldsymbol{\alpha}^{(y')^\top}\boldsymbol{\alpha}^{(y)}\right)\mathbf{1}_{\beta_z^{(y)}>0},\end{aligned} \tag{A.21}$$

and considering $\psi_z^{(y)}$, $\omega_z^{(y)}$, and $\delta_z^{(y)}$ as defined in Algorithm 2.4 and the gradient ascent update rule the formulas follow. □

A.2 Posterior distribution of observations

In this section, we show how to obtain the posterior distribution $p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, y)$ of sequences $\mathbf{X} \triangleq \left(\boldsymbol{x}^{(1)}, ..., \mathbf{x}^{(t)}\right)$ given an observed prefix sequence $\boldsymbol{X}_{\text{pref}} \triangleq \left(\boldsymbol{x}^{(-t_{\text{pref}}+1)}, ..., \boldsymbol{x}^{(0)}\right)$,

both coming from the graph node y. We start by writing this posterior as a marginalization with respect to the latent variable z:

$$\begin{aligned} p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, y) &= \sum_{\mathrm{z}} p(\mathbf{X}, \mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y) \\ &= \sum_{\mathrm{z}} p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, y, \mathrm{z}) p(\mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y) \\ &= \sum_{\mathrm{z}} p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) p(\mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y), \end{aligned} \tag{A.22}$$

where the last equality follows from the fact that the observations $\mathbf{X}$ are conditionally 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 2.3:

$$p(\mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y) \propto p(\boldsymbol{X}_{\text{pref}} \mid \mathrm{z}) p(\mathrm{z} \mid y). \tag{A.23}$$

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:

$$\begin{aligned} p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) &= \sum_{\mathbf{h}_{\text{pref}}, \mathbf{h}} p(\mathbf{X}, \mathbf{h}_{\text{pref}}, \mathbf{h} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \\ &= \sum_{\mathbf{h}_{\text{pref}}, \mathbf{h}} p(\mathbf{X} \mid \mathbf{h}_{\text{pref}}, \mathbf{h}, \boldsymbol{X}_{\text{pref}}, \mathrm{z}) p(\mathbf{h}_{\text{pref}}, \mathbf{h} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \\ &= \sum_{\mathbf{h}_{\text{pref}}, \mathbf{h}} p(\mathbf{X} \mid \mathbf{h}, \mathrm{z}) p(\mathbf{h}_{\text{pref}} \mid \mathbf{h}, \boldsymbol{X}_{\text{pref}}, \mathrm{z}) p(\mathbf{h} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \\ &= \sum_{\mathbf{h}} p(\mathbf{X} \mid \mathbf{h}, \mathrm{z}) p(\mathbf{h} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \\ &= \sum_{\mathbf{h}} p(\mathrm{h}^{(0)} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z}) \prod_{\tau=1}^{t} p(\mathrm{h}^{(\tau)} \mid \mathrm{h}^{(\tau-1)}, \mathrm{z}) p(\mathbf{x}^{(\tau)} \mid \mathrm{h}^{(\tau)}, \mathrm{z}), \end{aligned} \tag{A.24}$$

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 2.3.

Thus, we conclude that inference in the posterior model $p(\mathbf{X} \mid \boldsymbol{X}_{\text{pref}}, y)$ corresponds to inference in a (Spa)MHMM where the original mixture coefficients and initial state probabilites are replaced with their respective posteriors, $p(\mathrm{z} \mid \boldsymbol{X}_{\text{pref}}, y)$ and $p(\mathrm{h}^{(0)} \mid \boldsymbol{X}_{\text{pref}}, \mathrm{z})$. 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 xvii.]

[2] 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.]

[3] 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.]

[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. [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 4.]

[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 4.]

[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 4.]

[8] J. Gama and M. M. Gaber, *Learning from data streams: processing techniques in sensor networks*. Springer, 2007. [Cited on page 4.]

[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. [Cited on page 5.]

[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. [Cited on pages 6, 9, 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. [Cited on page 6.]

[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 8.]

[17] C. Ferles and A. Stafylopatis, "Self-organizing hidden Markov model map (SOHMMM)," *Neural Networks*, vol. 48, pp. 133 – 147, 2013. [Cited on pages 7 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. [Cited on page 7.]

[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 page 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 7 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] 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 pages 7 and 27.]

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

[24] 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.]

[25] 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.]

[26] 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.]

[27] 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.]

[28] 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.]

[29] 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 pages 8 and 9.]

[30] 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 pages 8 and 9.]

[31] 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.]

[32] 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.]

[33] 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 15.]

[34] 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 15.]

[35] OMNeT++, discrete event simulator. https://www.omnetpp.org. [Cited on pages 17 and 30.]

[36] Inet framework. https://inet.omnetpp.org. [Cited on pages 17 and 30.]

[37] 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.]

[38] 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.]

[39] 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. [Cited on page 26.]

[40] S. Lebedev. hmmlearn, hidden Markov models in python, with scikit-learn like API. 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 pages 29 and 62.]

[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.]

[56] F. T. de Andrade, "Adversarial domain adaptation for sensor networks," Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2020. [Cited on pages xv, 43, 58, 59, 60, and 68.]

[57] D. Pernes and J. S. Cardoso, "Tackling unsupervised multi-source domain adaptation with optimism and consistency," *CoRR*, vol. abs/2009.13939, 2020. [Cited on page 43.]

[58] H. Daumé III, A. Kumar, and A. Saha, "Frustratingly easy semi-supervised domain adaptation," in *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*. Association for Computational Linguistics, 2010, pp. 53–59. [Cited on page 44.]

[59] J. Donahue, J. Hoffman, E. Rodner, K. Saenko, and T. Darrell, "Semi-supervised domain adaptation with instance constraints," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 668–675. [Cited on page 44.]

[60] A. Kumar, A. Saha, and H. Daume, "Co-regularization based semi-supervised domain adaptation," in *Advances in Neural Information Processing Systems*, 2010, pp. 478–486. [Cited on page 44.]

[61] K. Saito, D. Kim, S. Sclaroff, T. Darrell, and K. Saenko, "Semi-supervised domain adaptation via minimax entropy," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 8050–8058. [Cited on page 44.]

[62] T. Yao, Y. Pan, C.-W. Ngo, H. Li, and T. Mei, "Semi-supervised domain adaptation with subspace learning for visual recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2142–2150. [Cited on page 44.]

[63] M. Baktashmotlagh, M. T. Harandi, B. C. Lovell, and M. Salzmann, "Unsupervised domain adaptation by domain invariant projection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 769–776. [Cited on page 44.]

[64] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *International Conference on Machine Learning*, 2015, pp. 1180–1189. [Cited on pages 44, 52, 54, and 56.]

[65] G. Kang, L. Jiang, Y. Yang, and A. G. Hauptmann, "Contrastive adaptation network for unsupervised domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4893–4902. [Cited on page 44.]

[66] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Unsupervised domain adaptation with residual transfer networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 136–144. [Cited on page 44.]

[67] H. Zhao, S. Zhang, G. Wu, J. M. Moura, J. P. Costeira, and G. J. Gordon, "Adversarial multiple source domain adaptation," in *Advances in Neural Information Processing Systems*, 2018, pp. 8559–8570. [Cited on pages 44, 45, 47, 52, and 54.]

[68] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Machine learning*, vol. 79, no. 1-2, pp. 151–175, 2010. [Cited on pages 44, 45, 46, and 47.]

[69] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, "Analysis of representations for domain adaptation," in *Advances in Neural Information Processing Systems*, 2007, pp. 137–144. [Cited on page 44.]

[70] J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Wortman, "Learning bounds for domain adaptation," in *Advances in Neural Information Processing Systems*, 2008, pp. 129–136. [Cited on page 44.]

[71] C. Cortes and M. Mohri, "Domain adaptation and sample bias correction theory and algorithm for regression," *Theoretical Computer Science*, vol. 519, pp. 103–126, 2014. [Cited on page 44.]

[72] R. Gopalan, R. Li, and R. Chellappa, "Unsupervised adaptation across domain shifts by generating intermediate data representations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2288–2302, 2013. [Cited on page 44.]

[73] J. Hoffman, M. Mohri, and N. Zhang, "Algorithms and theory for multiple-source adaptation," in *Advances in Neural Information Processing Systems*, 2018, pp. 8246–8256. [Cited on page 44.]

[74] H. Zhao, R. T. Des Combes, K. Zhang, and G. Gordon, "On learning invariant representations for domain adaptation," in *International Conference on Machine Learning*, 2019, pp. 7523–7532. [Cited on page 44.]

[75] H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, and M. Marchand, "Domain-adversarial neural networks," *arXiv preprint arXiv:1412.4446*, 2014. [Cited on page 44.]

[76] C. J. Becker, C. M. Christoudias, and P. Fua, "Non-linear domain adaptation with boosting," in *Advances in Neural Information Processing Systems*, 2013, pp. 485–493. [Cited on page 44.]

[77] B. Fernando, A. Habrard, M. Sebban, and T. Tuytelaars, "Unsupervised visual domain adaptation using subspace alignment," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2960–2967. [Cited on page 44.]

[78] I.-H. Jhuo, D. Liu, D. Lee, and S.-F. Chang, "Robust visual domain adaptation with low-rank reconstruction," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 2168–2175. [Cited on page 44.]

[79] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," *arXiv preprint arXiv:1502.02791*, 2015. [Cited on pages 44 and 52.]

[80] C. Louizos, K. Swersky, Y. Li, M. Welling, and R. Zemel, "The variational fair autoencoder," in *International Conference on Learning Representations*, 2016. [Cited on page 44.]

[81] B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. [Cited on page 44.]

[82] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7167–7176. [Cited on page 44.]

[83] Y.-B. Kim, K. Stratos, and D. Kim, "Domain attention with an ensemble of experts," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 643–653. [Cited on page 44.]

[84] J. Guo, D. Shah, and R. Barzilay, "Multi-source domain adaptation with mixture of experts," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 4694–4703. [Cited on pages 44 and 52.]

[85] Y. Mansour, M. Mohri, and A. Rostamizadeh, "Domain adaptation: Learning bounds and algorithms," in *22nd Conference on Learning Theory, COLT 2009*, 2009. [Cited on page 44.]

[86] A. Schoenauer-Sebag, L. Heinrich, M. Schoenauer, M. Sebag, L. Wu, and S. Altschuler, "Multi-domain adversarial learning," in *International Conference on Learning Representations*, 2019. [Cited on pages 44 and 52.]

[87] K. Zhang, M. Gong, and B. Schölkopf, "Multi-source domain adaptation: A causal view," in *Twenty-ninth AAAI conference on artificial intelligence*, 2015. [Cited on page 44.]

[88] C. Cortes and M. Mohri, "Domain adaptation in regression," in *Algorithmic Learning Theory*, J. Kivinen, C. Szepesvári, E. Ukkonen, and T. Zeugmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 308–323. [Cited on page 45.]

[89] Z. Lipton, Y.-X. Wang, and A. Smola, "Detecting and correcting for label shift with black box predictors," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3122–3130. [Cited on page 49.]

[90] A. Iyer, S. Nath, and S. Sarawagi, "Maximum mean discrepancy for class ratio estimation: Convergence bounds and kernel selection," in *International Conference on Machine Learning*, 2014, pp. 530–538. [Cited on page 49.]

[91] K. Zhang, B. Schölkopf, K. Muandet, and Z. Wang, "Domain adaptation under target and conditional shift," in *International Conference on Machine Learning*, 2013, pp. 819–827. [Cited on pages 49 and 50.]

[92] Y. S. Chan and H. T. Ng, "Word sense disambiguation with distribution estimation," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, ser. IJCAI'05. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, p. 1010–1015. [Cited on page 49.]

[93] A. Storkey, "When training and test sets are different: characterizing learning transfer," *Dataset shift in machine learning*, pp. 3–28, 2009. [Cited on page 49.]

[94] G. I. Webb, L. K. Lee, B. Goethals, and F. Petitjean, "Analyzing concept drift and shift from sample data," *Data Mining and Knowledge Discovery*, vol. 32, no. 5, pp. 1179–1199, 2018. [Cited on page 50.]

[95] J. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, Mar. 2014. [Cited on page 50.]

[96] M. Sugiyama, M. Krauledat, and K.-R. Müller, "Covariate shift adaptation by importance weighted cross validation." *Journal of Machine Learning Research*, vol. 8, pp. 985–1005, 05 2007. [Cited on page 51.]

[97] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227–244, 2000. [Cited on page 51.]

[98] C. Cortes, Y. Mansour, and M. Mohri, "Learning bounds for importance weighting," in *Advances in Neural Information Processing Systems*, 2010, pp. 442–450. [Cited on page 51.]

[99] N. Courty, R. Flamary, D. Tuia, and A. Rakotomamonjy, "Optimal transport for domain adaptation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 9, pp. 1853–1865, 2017. [Cited on page 51.]

[100] N. Courty, R. Flamary, A. Habrard, and A. Rakotomamonjy, "Joint distribution optimal transportation for domain adaptation," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Cited on page 51.]

[101] R. Turrisi, R. Flamary, A. Rakotomamonjy, and M. Pontil, "Multi-source domain adaptation via weighted joint distributions optimal transport," *arXiv preprint arXiv:2006.12938*, 2020. [Cited on page 51.]

[102] Z. Pei, Z. Cao, M. Long, and J. Wang, "Multi-adversarial domain adaptation," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. [Cited on page 52.]

[103] E. Bareinboim and J. Pearl, "Causal inference and the data-fusion problem," *Proceedings of the National Academy of Sciences*, vol. 113, no. 27, pp. 7345–7352, 2016. [Cited on page 53.]

[104] M. Rojas-Carulla, B. Schölkopf, R. Turner, and J. Peters, "Invariant models for causal transfer learning," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 1309–1342, 2018. [Cited on page 53.]

[105] S. Magliacane, T. van Ommen, T. Claassen, S. Bongers, P. Versteeg, and J. M. Mooij, "Domain adaptation by using causal inference to predict invariant conditional distributions," in *NeurIPS*, 2018. [Cited on page 53.]

[106] J. Peters, J. M. Mooij, D. Janzing, and B. Schölkopf, "Causal discovery with continuous additive noise models," *Journal of Machine Learning Research*, vol. 15, pp. 2009–2053, 2014. [Cited on page 53.]

[107] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio, "Toward causal representation learning," *Proceedings of the IEEE*, 2021. [Cited on page 53.]

[108] M.-H. Chen, Z. Kira, and G. AlRegib, "Temporal attentive alignment for video domain adaptation," in *International Conference on Computer Vision (ICCV)*, 2019. [Cited on page 54.]

[109] Y. Liu and X. Li, "Domain adaptation for land use classification: A spatio-temporal knowledge reusing method," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 98, pp. 133 – 144, 2014. [Cited on page 54.]

[110] S. Zhang, G. Wu, J. P. Costeira, and J. M. F. Moura, "FCN-rLSTM: Deep spatio-temporal neural networks for vehicle counting in city cameras," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 3687–3696, 2017. [Cited on pages xv, 54, 56, 57, and 61.]

[111] S. Zhang, G. Wu, J. P. Costeira, and J. M. Moura, "Understanding traffic density from large-scale web camera data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5898–5907. [Cited on pages xv, xvi, 62, and 63.]