SPAMHMM: SPARSE MIXTURE OF HIDDEN MARKOV MODELS FOR GRAPH CONNECTED ENTITIES

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a framework to model the distribution of sequential data coming from a set of entities connected in a graph with a known topology. The method is based on a mixture of shared hidden Markov models (HMMs), which are trained in order to exploit the knowledge of the graph structure and in such a way that the obtained mixtures tend to be sparse. Experiments in different application domains demonstrate the effectiveness and versatility of the method.

1 INTRODUCTION AND STATE OF THE ART

Hidden Markov models (HMMs) are a ubiquitous tool for modelling sequential data. They started by being applied to speech recognition systems and from there they have spread to almost any application one can think of, encompassing computational molecular biology, data compression, and computer vision. In the emerging field of cognitive radars (Greco et al., 2018), for the task of opportunistic usage of the spectrum, HMMs have been recently used to model the occupancy of the channels by primary users (Stinco et al., 2016).

When the expressiveness of an HMM is not enough, mixtures of HMM have been adopted. Roughly speaking, mixtures of HMMs can be interpreted as the result of the combination of a set of independent standard HMMs which are observed through a memoryless transformation (Couvreur, 1996; Dias et al., 2010; Subakan et al., 2014; Helske & Helske, 2016).

In many real-life settings one does not have a single data stream but 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 relations 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., 2014). 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., 2017; Theagarajan et al., 2018). Thus, understanding a player's behavior implies understanding the behavior of his teammates and opponents over time.

The extraction of knowledge from these streams to support the decision-making process is still challenging and the adaptation of HMM to this scenario is immature at best. Ferles & Stafylopatis (2013) proposed a hybrid approach combining the Self-Organizing Map (SOM) and the HMM with applications in clustering, dimensionality reduction and visualization of large-scale sequence spaces. Note that the model at each node is limited to a simple HMM. Wireless local area networks have also been modeled with Markov-based approaches. For instance, Allahdadi et al. (2014) use HMMs for outlier detection in 802.11 wireless access points. However, the typical approaches include a common HMM model for all nodes (with strong limited flexibility) and a HMM model per node, independent of the others (not exploring the dependencies between nodes). Bolton et al. (2018) built a sparse coupled hidden Markov model (SCHMM) framework to parameterize the temporal evolution of data acquired with functional magnetic resonance imaging (fMRI). The coupling is captured in the transition matrix, which is assumed to be a function of the activity levels of all the streams; the model per node is still restricted to a simple HMM.

In general, in networked data streams, the stream observed in each sensor is often modeled by HMMs but the intercorrelations between sensors are seldom explored. The proper modeling of the intercorrelations has the potential to improve the learning process, acting as a regularizer in the learning process. In here we propose to tackle this void, by proposing as observation model at each node a sparse mixture of HMMs, where the dependencies between nodes are used to promote the sharing of HMM components between similar nodes.

2 PROPOSED MODEL

The proposed model finds intersections with distributed sparse representation and multitask learning.

2.1 OVERVIEW

Sparse representation/coding 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:

$$f(x) = \sum_i s_i \phi_i(x), \tag{1}$$

where $\phi_i(x)$ are the atoms and only a few of the scalars s_i are non-zero, providing a sparse representation of $f(x)$.

Distributed sparse representation (Baron et al., 2009) is an extension of the standard version that considers networks with K nodes. At each node, the signal sensed at the same node has its sparsity property because of its intracorrelation, while, for networks with multiple nodes, signals received at different nodes also exhibit strong intercorrelation. The intra- and inter-correlations lead to a joint sparse model. An interesting scenario in distributed sparse representation is when all signals/models share the common support but with different non-zero coefficients.

Multitask learning techniques rely on the idea that individual models for related tasks should share some structure (parameters or hyperparameters). An interesting approach is based on adapting knowledge instead of data, handled by parameter transfer approaches, where parameters for different tasks/models are shared or constrained to be similar (Fernandes & Cardoso, 2017).

Inspired by the formulation of equation 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 each node as a sparse HMM mixture over a 'large' 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 formalize now these ideas.

2.2 MODEL FORMULATION

2.2.1 DEFINITION

Assume we have a set of nodes $\mathbb{Y} = \{1, ..., K\}$ 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 $\mathbf{X} = \left(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)}\right)$, $\mathbf{x}^{(t)} \in \mathbb{R}^D$, whose conditional distribution we shall model using a mixture of HMMs:

$$p(\mathbf{X}|\mathrm{y}) = \sum_{\mathrm{z}} p(\mathrm{z}|\mathrm{y}) p(\mathbf{X}|\mathrm{z}), \tag{2}$$

where $\mathrm{z} \in \{1, ..., M\}$ is a latent random variable, being M the size of the mixture. Here, $p(\mathbf{X}|\mathrm{z})$ is the marginal distribution of observations of a standard first-order HMM:

$$p(\mathbf{X}|\mathrm{z}) = \sum_{\mathbf{h}} p(\mathrm{h}^{(0)}|\mathrm{z}) \prod_t p(\mathrm{h}^{(t)}|\mathrm{h}^{(t-1)}, \mathrm{z}) p(\mathbf{x}^{(t)}|\mathrm{h}^{(t)}, \mathrm{z}), \tag{3}$$

where $\mathbf{h} = \left(\mathrm{h}^{(0)}, ..., \mathrm{h}^{(T)}\right)$, $\mathrm{h}^{(t)} \in \{1, ..., S\}$, is the sequence of hidden states of the HMM, being S the number of hidden states. Note that the factorization in equation 2 imposes conditional independence between the sequence $\mathbf{X}$ and the node y, given the latent variable z. This is a key assumption of this model, since this way the distributions for the observations in the nodes in $\mathbb{Y}$ share the same pool of HMMs, promoting parameter sharing among the K mixtures.

2.2.2 INFERENCE

Given an observed sequence $\boldsymbol{X}$ and its corresponding node $y \in \mathbb{Y}$, the inference problem here consists in finding the likelihood $p(\mathbf{X} = \boldsymbol{X}|\mathrm{y} = y)$ (from now on, abbreviated as $p(\boldsymbol{X}|y)$) as defined by equations 2 and 3. The marginals $p(\boldsymbol{X}|\mathrm{z})$ of each HMM in the mixture may be computed efficiently, in $O(S^2T)$ time, using the Forward algorithm (Rabiner & Juang, 1986). Then, $p(\boldsymbol{X}|y)$ is obtained by applying equation 2, so inference in the overall model is done in at most $O(MS^2T)$ time. As we shall see, however, the mixtures we get after learning will often be sparse (see section 2.2.3), leading to an even smaller time complexity.

2.2.3 LEARNING

Given an i.i.d. dataset consisting of N tuples $(\boldsymbol{X}_i, y_i)$ of sequences of observations $\boldsymbol{X}_i = \left(\boldsymbol{x}_i^{(1)}, ..., \boldsymbol{x}_i^{(T_i)}\right)$ and their respective nodes $y_i \in \mathbb{Y}$, the model defined by equations 2 and 3, may be easily trained using the Expectation-Maximization (EM) algorithm (Dempster et al., 1977), (locally) maximizing the usual log-likelihood objective:

$$J(\theta) = \sum_i \log p(\boldsymbol{X}_i|y_i, \theta), \tag{4}$$

where θ represents all model parameters, namely:

1. the mixture coefficients, $\boldsymbol{\alpha}_k := (p(\mathrm{z} = 1|\mathrm{y} = k), ..., p(\mathrm{z} = M|\mathrm{y} = k))$, for $k = 1, ..., K$;
2. the initial state probabilities, $\boldsymbol{\pi}_m := \left(p(\mathrm{h}^{(0)} = 1|\mathrm{z} = m), ..., p(\mathrm{h}^{(0)} = S|\mathrm{z} = m)\right)$, for $m = 1, ..., M$;
3. the state transition matrices, $\boldsymbol{A}^m$, where $A^m_{s,u} := p(\mathrm{h}^{(t)} = u|\mathrm{h}^{(t-1)} = s, \mathrm{z} = m)$, for $s, u = 1, ..., S$ and $m = 1, ..., M$;
4. the emission probability means, $\boldsymbol{\mu}_{m,s} \in \mathbb{R}^D$, for $m = 1, ..., M$ and $s = 1, ..., S$;
5. the emission probability diagonal covariance matrices, $\boldsymbol{\sigma}^2_{m,s}\boldsymbol{I}$, where $\boldsymbol{\sigma}^2_{m,s} \in \mathbb{R}^{+D}$, for $m = 1, ..., M$ and $s = 1, ..., S$.

Here, we are assuming that the observations are continuous and the emission probabilities $p(\mathrm{x}^{(t)}|\mathrm{h}^{(t)}, \mathrm{z})$ 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 equation 4 using EM is described in Algorithm 1, in section A.1. The update formulas follow from the standard EM procedure and can be obtained by viewing this model as Bayesian network or by following the derivation detailed in section A.2. However, the objective 4 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:

$$J_r(\theta) = \frac{1}{N}\sum_i \log p(\boldsymbol{X}_i|y_i, \theta) + \frac{\lambda}{2}\sum_{j,k\neq j} G_{j,k}\mathbb{E}_{\mathrm{z}\sim p(\mathrm{z}|\mathrm{y}=j,\theta)}[p(\mathrm{z}|\mathrm{y} = k, \theta)], \tag{5}$$

where $\lambda \geqslant 0$ controls the relative weight of the two terms in the objective. The expectations $\mathbb{E}_{\mathrm{z}\sim p(\mathrm{z}|\mathrm{y}=j,\theta)}[p(\mathrm{z}|\mathrm{y} = k, \theta)$ have interesting properties which are enlightened and proven in Proposition 1.

Proposition 1. *Let $\mathbb{P}$ be the set of all M-nomial probability distributions. We have:*

1. $\min_{p,q\in\mathbb{P}} \mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = 0$*;*

2. $\arg\min_{p,q\in\mathbb{P}} \mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = \{p, q \in \mathbb{P} \mid \forall\, m \in \{1, ..., M\} : p(\mathrm{z} = m)q(\mathrm{z} = m) = 0\}$;

3. $\max_{p,q\in\mathbb{P}} \mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = 1$;

4. $\arg\max_{p,q\in\mathbb{P}} \mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = \{p, q \in \mathbb{P} \mid \exists\, m \in \{1, ..., M\} :\ p(\mathrm{z} = m) = q(\mathrm{z} = m) = 1\}$.

Proof. By the definition of expectation,

$$\mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = \sum_m p(\mathrm{z} = m)q(\mathrm{z} = m). \tag{6}$$

Statements 1 and 2 follow immediately from the fact that every term in the right-hand side of equation 6 is non-negative.
For the remaining, we note that equation 6 may be rewritten 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:

$$\mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = \boldsymbol{\alpha}_p^\intercal \boldsymbol{\alpha}_q \leqslant ||\boldsymbol{\alpha}_p||_2 ||\boldsymbol{\alpha}_q||_2 \leqslant ||\boldsymbol{\alpha}_p||_1 ||\boldsymbol{\alpha}_q||_1 = 1, \tag{7}$$

where $||\cdot||_1$ and $||\cdot||_2$ are the L^1 and L^2 norms, respectively. Clearly, the equality $\mathbb{E}_{\mathrm{z}\sim p}[q(\mathrm{z})] = 1$ 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 equation 7 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 $j, k \in \mathbb{Y}$, if $G_{j,k} > 0$, the regularization term for these nodes is maximum (and equal to $G_{j,k}$) 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 $G_{j,k} < 0$, 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 1 that we are favoring sparse mixtures. We see sparsity as an important feature since: 1 – it allows the coefficients to better capture the graph structure, which is usually sparse, and 2 – it leads to mixtures with fewer components, yielding faster inference and (possibly) less overfitting.

By setting $\lambda = 0$, we clearly get the initial objective 4, where inter-node correlations are modeled only via parameter sharing. As $\lambda \to \infty$, two interesting scenarios may be anticipated. If $G_{j,k} > 0,\ \forall j, k$, 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 $G_{j,k} < 0,\ \forall j, k$, and $M \geqslant K$, each node would tend to learn its own HMM model independently from all the others.

The objective function 5 can still be maximized via EM (see details in section A.3). 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 reparameterization from Yang et al. (2018):

$$\alpha_{k,m} = \frac{\sigma\left(\beta_{k,m}\right)^2}{\sum_l \sigma\left(\beta_{k,l}\right)^2}, \tag{8}$$

where $\sigma(\cdot)$ is the rectifier linear (ReLU) function. This reparameterization clearly resembles the softmax function, but, contrarily to that one, admits sparse outputs. The squared terms in equation 8 aim only to make the optimization more stable. The optimization steps for the objective 5 using this reparameterization are described in Algorithm 2, in section A.1.

3 Experimental Evaluation

The model was developed on top of the library hmmlearn[1] for Python, which implements inference and unsupervised learning for the standard HMM using a wide variety of emission distributions.

[1] `https://github.com/hmmlearn/hmmlearn`

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[2].

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 and 3), trained to maximize the usual log-likelihood objective (equation 4); a mixture of HMMs (denoted as SpaMHMM) as the previous one, trained to maximize our regularized objective (equation 5). Models 1-HMM, K-HMM and MHMM will be our baselines. We shall compare the performance of these models with that of SpaMHMM and, for the case of MHMM, we shall also verify if SpaMHMM actually produces sparser mixtures in general, as argued in section 2.2.3. 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 1 and SpaMHMM is trained using Algorithm 2. However, we opted to use Adam (Kingma & Ba, 2014) instead of "vanilla" gradient ascent in the inner loop of Algorithm 2, since its per-parameter learning rate proved to be beneficial for faster convergence.

3.1 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 (RF) 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 & Morla (2017), performing extensive network simulations using OMNeT++[3] and INET[4]. The former is a discrete event simulator for modeling communication networks that is used in many problem domains, including wireless networks. The latter is a framework that provides detailed models for several communication protocols (TCP, IP, IEEE 802.11, etc.). Here, we used OMNeT++ and INET to generate traffic in a typical Wi-Fi network setup (IEEE 802.11 WLANg 2.4 GHz in infrastructure mode).

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

[2]URL available after decision

[3]`https://www.omnetpp.org`

[4]`https://inet.omnetpp.org/`

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 $G_{j,k}$, associated with the edge connecting nodes j and k in graph $\mathcal{G}$, was set to the inverse distance between APs j and k and normalized so that $\max_{j,k} G_{j,k} = 1$. As in Allahdadi & Morla (2017), 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 $M = 10$ 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 3. Every model was trained for 100 EM iterations or until the loss plateaus. For SpaMHMM, we did 100 iterations of the inner loop on each M-step, using a learning rate $\rho = 10^{-3}$.

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). Figure 1 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 1. 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 pool 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, mixture components 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) and by a small difference. The inexistence 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.

3.2 HUMAN MOTION FORECASTING

The human body is constituted by several interdependent parts, which interact as a whole producing sensible global motion patterns. These patterns may correspond to multiple activities like walking, eating, etc. Here, we use our model to make short-time prediction of sequences of human joint positions, represented as motion capture (mocap) data. The 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., 2015; Martinez et al., 2017) and in long-term motion generation (Jain et al., 2016; Pavllo et al., 2018).

Our experiments were conducted on the Human3.6M dataset from Ionescu et al. (2011; 2014), 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,

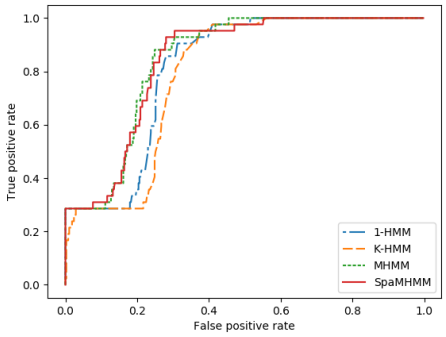

Figure 1: ROC curves for each model on the Wi-Fi dataset.

	AUC	Avg. log-likelihood Normal data	Anom. data
1-HMM	0.808	-6.28	-112.75
K-HMM	0.786	-21.01	**-136.24**
MHMM	**0.842**	-3.07	-11.99
SpaMHMM	0.839	**-3.06**	-14.57

Table 1: AUC and average log-likelihood per sample for each model in the Wi-Fi dataset. Best results are in bold.

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: $G_{j,k} = 1$, if there is an actual skeleton connection between joints j and k (e.g. the elbow joint is connected to the wrist joint by the forearm); $G_{j,k} = 1$, if joints j and k are symmetric (e.g. left and right elbows); $G_{j,k} = 0$, otherwise.

3.2.1 FORECASTING

We reproduced as much as possible the experimental setup followed in Fragkiadaki et al. (2015). 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 learning 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 build the distribution $p(\mathbf{X}|\boldsymbol{X}_{\text{pref}}, y)$ (see details in section A.4) and we sample sequences from that posterior. Our evaluation method and metric again followed Fragkiadaki et al. (2015). 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 ms). 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. Among our models (1-HMM, K-HMM, MHMM and SpaMHMM), SpaMHMM outperformed the remaining in all actions except "eating". For this action in particular, MHMM was slightly better than SpaMHMM, probably due to the lack of symmetry between the right and left sides of the body, which was one of the prior assumptions that we have used to build the graph $\mathcal{G}$. "Smoking" and "discussion" activities may also be highly non-symmetric, but results in our and others' models show that these activities are generally harder to predict than "walking" and "eating'. Thus, here, the skeleton structure information encoded in $\mathcal{G}$ behaves as a useful prior for SpaMHMM, guiding it towards better solutions than MHMM. The worse results for 1-HMM and K-HMM likely result from the same limitations that we have pointed out in section 3.1: each component in K-HMM is inherently trained with less data than the remaining models, while

	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 (Fragkiadaki et al., 2015)	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 (Fragkiadaki et al., 2015)	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 (Jain et al., 2016)	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. (Martinez et al., 2017)	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 (Pavllo et al., 2018)	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: 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. (2018). Best results among our models are in bold, best overall results are underlined.

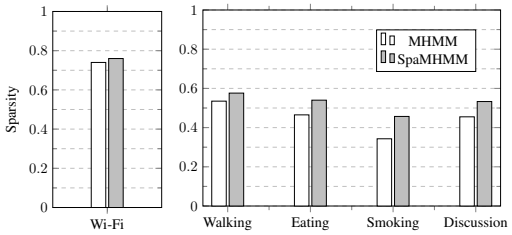

Figure 2: Relative sparsity (number of coefficients equal to zero / total number of coefficients) of the obtained MHMM and SpaMHMM models on the Wi-Fi dataset (left) and on the Human3.6M dataset for different actions (right). Both models for the Wi-Fi dataset have 150 coefficients. All models for the Human3.6M dataset have 396 coefficients.

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. 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.

3.2.2 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:

$$C_k = \underset{m \in \{1,2,3,4\}}{\arg\max}\ p(\mathrm{z} = m | \mathrm{y} = k) = \underset{m \in \{1,2,3,4\}}{\arg\max}\ \alpha_{k,m}, \tag{9}$$

where C_k is, therefore, the cluster assigned to joint k. The results are in figure 3. 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

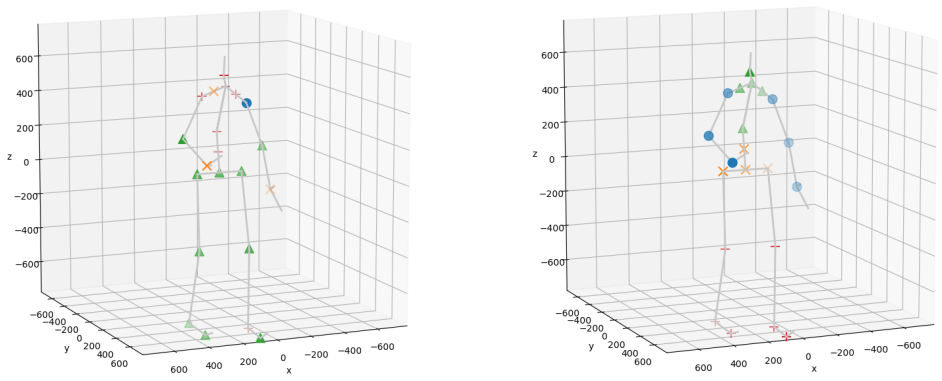

Figure 3: Assignments of joints to clusters in MHMM (left) and SpaMHMM (right). The different symbols (‘○’, ‘△’, ‘x’, ‘+’) and the respective colors (blue, green, orange and red) on each joint represent the cluster that the joint was assigned to.

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/‘○’ 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/‘△’ 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/‘△’ cluster; arms in the blue/‘○’ 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.

4 Conclusion and Future Work

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 the proposed regularizer promotes sparsity in the mixtures, which is achieved through a fully differentiable loss function (i.e. with no explicit L^0 penalty term). We evaluate the method’s performance in two completely different tasks (anomaly detection in Wi-Fi networks and human motion forecasting), showing its effectiveness and versatility.

For future work, we plan to extend/evaluate the usage of SpaMHMM for sequence clustering. This is an obvious extension that we did not explore thoroughly in this work, since its main focus was modeling the generative distribution of data. In this context, extending the idea behind SpaMHMM to mixtures of more powerful generative distributions is also in our plans. As is known, HMMs have limited expressiveness due to the strong independence assumptions they rely on. Thus, we plan to extend these ideas to develop an architecture based on more flexible generative models for sequence modeling, like those attained using deep recurrent architectures.

Acknowledgments

Anonymous acknowledgments.

REFERENCES

Anisa Allahdadi and Ricardo Morla. 802.11 wireless access point usage simulation and anomaly detection. *CoRR*, abs/1707.02933, 2017. URL http://arxiv.org/abs/1707.02933.

Anisa Allahdadi, Ricardo Morla, and Jaime S. Cardoso. Outlier detection in 802.11 wireless access points using hidden markov models. In *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP*, pp. 1–8. IEEE, 2014.

Dror Baron, Marco F. Duarte, Michael B. Wakin, Shriram Sarvotham, and Richard G. Baraniuk. Distributed compressive sensing. *CoRR*, abs/0901.3403, 2009. URL http://arxiv.org/abs/0901.3403.

Leonard Baum. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a markov process. *Inequalities*, 3:1–8, 1972.

T. A. W. Bolton, A. Tarun, V. Sterpenich, S. Schwartz, and D. Van De Ville. Interactions between large-scale functional brain networks are captured by sparse coupled hmms. *IEEE Transactions on Medical Imaging*, 37(1):230–240, Jan 2018. ISSN 0278-0062.

Christophe Couvreur. Hidden markov models and their mixtures. In *DEA Report, Dep. of Mathematics, Université catholique de Louvain*, 1996.

Fabrizio De Vico Fallani, Jonas Richiardi, Mario Chavez, and Sophie Achard. Graph analysis of functional brain networks: practical issues in translational neuroscience. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 369(1653), 2014. ISSN 0962-8436. doi: 10.1098/rstb.2013.0521. URL http://rstb.royalsocietypublishing.org/content/369/1653/20130521.

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

José G. Dias, Jeroen K. Vermunt, and Sofia Ramos. Mixture hidden markov models in finance research. In Andreas Fink, Berthold Lausen, Wilfried Seidel, and Alfred Ultsch (eds.), *Advances in Data Analysis, Data Handling and Business Intelligence*, pp. 451–459, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

Christos Ferles and Andreas Stafylopatis. Self-organizing hidden markov model map (sohmmm). *Neural Networks*, 48:133 – 147, 2013. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2013.07.011. URL http://www.sciencedirect.com/science/article/pii/S0893608013001974.

Kelwin Fernandes and Jaime S. Cardoso. Hypothesis transfer learning based on structural model similarity. *Neural Computing and Applications*, Nov 2017. ISSN 1433-3058. doi: 10.1007/s00521-017-3281-4. URL https://doi.org/10.1007/s00521-017-3281-4.

Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4346–4354, 2015.

M. S. Greco, F. Gini, P. Stinco, and K. Bell. Cognitive radars: On the road to reality: Progress thus far and possibilities for the future. *IEEE Signal Processing Magazine*, 35(4):112–125, July 2018. ISSN 1053-5888. doi: 10.1109/MSP.2018.2822847.

Satu Helske and Jouni Helske. Mixture hidden markov models for sequence data : The seqhmm package in r. 2016.

Catalin Ionescu, Fuxin Li, and Cristian Sminchisescu. Latent structured models for human pose estimation. In *International Conference on Computer Vision*, 2011.

Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian 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.

Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5308–5317, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Julieta Martinez, Michael J Black, and Javier Romero. On human motion prediction using recurrent neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4674–4683. IEEE, 2017.

Dario Pavllo, David Grangier, and Michael Auli. Quaternet: A quaternion-based recurrent model for human motion. *arXiv preprint arXiv:1805.06485*, 2018.

Lawrence R Rabiner and Biing-Hwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

P. Stinco, M. S. Greco, and F. Gini. Spectrum sensing and sharing for cognitive radars. *IET Radar, Sonar Navigation*, 10(3):595–602, 2016. ISSN 1751-8784. doi: 10.1049/iet-rsn.2015.0372.

Cem Subakan, Johannes Traa, and Paris Smaragdis. Spectral learning of mixture of hidden markov models. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2249–2257. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5518-spectral-learning-of-mixture-of-hidden-markov-models.pdf.

Rajkumar Theagarajan, Federico Pala, Xiu Zhang, and Bir 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.

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)*, pp. 147–154, July 2017. doi: 10.1109/CVPRW.2017.24.

Zhilin Yang, Jake Zhao, Bhuwan Dhingra, Kaiming He, William W. Cohen, Ruslan Salakhutdinov, and Yann LeCun. Glomo: Unsupervisedly learned relational graphs as transferable representations, 2018. URL https://arxiv.org/abs/1806.05662.

A APPENDIX

A.1 ALGORITHMS

Data: 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 $\mathcal{I}$.

for $j = 1, ..., \mathcal{I}$ **do**

Sufficient statistics:

1. $n_k \coloneqq \sum_i \mathbf{1}_{y_i=k}$, where $\mathbf{1}_{\{\cdot\}}$ is the indicator function, for $k = 1, ..., K$.
2. Obtain the mixture posteriors $\eta_{i,m} \coloneqq p(\mathrm{z} = m|\boldsymbol{X}_i, y_i, \theta^{(j-1)})$, for $i = 1, ..., N$ and $m = 1, ..., M$, by computing $\tilde{\eta}_{i,m} \coloneqq p(\boldsymbol{X}_i|\mathrm{z} = m, \theta^{(j-1)})p(\mathrm{z} = m|y_i, \theta^{(j-1)})$ and normalizing it.
3. Obtain the state posteriors $\gamma_{i,m,s}(t) \coloneqq p(\mathrm{h}^{(t)} = s|\mathrm{z} = m, \boldsymbol{X}_i, \theta^{(j-1)})$ and $\xi_{i,m,s,u}(t) \coloneqq p(\mathrm{h}^{(t-1)} = s, \mathrm{h}^{(t)} = u|\mathrm{z} = m, \boldsymbol{X}_i, \theta^{(j-1)})$, for $i = 1, ..., N$, $m = 1, ..., M$ and $s, u = 1, ..., S$, as done in the Baum-Welch algorithm (Baum, 1972).

M-step:

1. $\alpha_{k,m} = \frac{\sum_i \eta_{i,m} \mathbf{1}_{y_i=k}}{n_k}$, for $k = 1, ..., K$ and $m = 1, ..., M$, obtaining $\boldsymbol{\alpha}_k$.
2. $\pi_{m,s} = \frac{\sum_i \eta_{i,m} \gamma_{i,m,s}(0)}{\sum_i \eta_{i,m}}$, for $m = 1, ..., M$ and $s = 1, ..., S$, obtaining $\boldsymbol{\pi}_m$.
3. $A^m_{s,u} = \frac{\sum_i \eta_{i,m} \sum_{t=1}^{T_i} \xi_{i,m,s,u}(t)}{\sum_i \eta_{i,m} \sum_{t=0}^{T_i-1} \gamma_{i,m,s}(t)}$, for $m = 1, ..., M$ and $s, u = 1, ..., S$, obtaining $\boldsymbol{A}^m$.
4. $\boldsymbol{\mu}_{m,s} = \frac{\sum_i \eta_{i,m} \sum_{t=1}^{T_i} \gamma_{i,m,s}(t) \boldsymbol{x}_i^{(t)}}{\sum_i \eta_{i,m} \sum_{t=1}^{T_i} \gamma_{i,m,s}(t)}$, for $m = 1, ..., M$ and $s = 1, ..., S$.
5. $\boldsymbol{\sigma}^2_{m,s} = \frac{\sum_i \eta_{i,m} \sum_{t=1}^{T_i} \gamma_{i,m,s}(t) \left(\boldsymbol{x}_i^{(t)} - \boldsymbol{\mu}_s^m\right)^2}{\sum_i \eta_{i,m} \sum_{t=1}^{T_i} \gamma_{i,m,s}(t)}$, for $m = 1, ..., M$ and $s = 1, ..., S$.
6. $\theta^{(j)} = \bigcup_{k,m,s} \left\{\boldsymbol{\alpha}_k, \boldsymbol{\pi}_m, \boldsymbol{A}^m, \boldsymbol{\mu}_{m,s}, \boldsymbol{\sigma}^2_{m,s}\right\}$.

end

Algorithm 1: EM algorithm for the mixture without regularization (MHMM).

Data: 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 $\mathcal{I}$, the number of gradient ascent iterations $\mathcal{J}$ to perform on each M-step, the learning rate ρ for the gradient ascent.

for $j = 1, ..., \mathcal{I}$ **do**

Sufficient statistics: same as in Algorithm 1.

M-step:

for $l = 1, ..., \mathcal{J}$ **do**

1. $\psi_{k,m} \coloneqq \frac{1}{N} \sum_i (\eta_{i,m} - \alpha_{k,m}) \mathbf{1}_{y_i=k}$, for $k = 1, ..., K$ and $m = 1, ..., M$.
2. $\omega_{k,m} \coloneqq \alpha_{k,m} \sum_{j \neq k} G_{j,k} \left(\alpha_{j,m} - \boldsymbol{\alpha}_j^\intercal \boldsymbol{\alpha}_k\right)$, for $k = 1, ..., K$ and $m = 1, ..., M$.
3. $\delta_{k,m} \coloneqq \mathbf{1}_{\beta_{k,m}>0} \frac{2\sigma'(\beta_{k,m})}{\sigma(\beta_{k,m})} (\psi_{k,m} + \lambda \omega_{k,m})$, where $\sigma'(\cdot)$ is the derivative of $\sigma(\cdot)$, for $k = 1, ..., K$ and $m = 1, ..., M$.
4. $\beta_{k,m} \leftarrow \beta_{k,m} + \rho \delta_{k,m}$, for $k = 1, ..., K$ and $m = 1, ..., M$.
5. Use equation 8 to obtain $\alpha_{k,m}$, for $k = 1, ..., K$ and $m = 1, ..., M$.

end

Do steps 2) – 6) in the M-step of Algorithm 1.

end

Algorithm 2: EM algorithm for the mixture with regularization (SpaMHMM).

A.2 PROOF OF ALGORITHM 1

Algorithm 1 follows straightforwardly from applying EM to the model defined by equations 2 and 3 with the objective 4.

Let us define the following notation: $\mathbf{X} \coloneqq \{\boldsymbol{X}_i\}_{i=1}^N$, $\boldsymbol{y} \coloneqq \{y_i\}_{i=1}^N$, $\mathbf{z} \coloneqq \{\mathrm{z}_i\}_{i=1}^N$ and $\mathbf{H} \coloneqq \{\mathbf{h}_i\}_{i=1}^N$. After building the usual variational lower bound for the log-likelihood and performing the E-step, we get the following well-known objective:

$$\tilde{J}(\theta, \theta^{\text{-}}) = \sum_{\mathbf{z},\mathbf{H}} p(\mathbf{X}, \mathbf{z}, \mathbf{H}|\boldsymbol{y}, \theta^{\text{-}}) \log p(\mathbf{X}, \mathbf{z}, \mathbf{H}|\boldsymbol{y}, \theta), \tag{10}$$

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

$$\begin{aligned} L(\theta, \theta^{\text{-}}, \boldsymbol{\lambda}) = \tilde{J}(\theta, \theta^{\text{-}}) &+ \sum_k \lambda_k^{\text{mix}} \left(1 - ||\boldsymbol{\alpha}_k||_1\right) \\ &+ \sum_{m,s} \lambda_{m,s}^{\text{state}} \left(1 - \sum_u A_{s,u}^m\right) \\ &+ \sum_m \lambda_m^{\text{ini}} \left(1 - ||\boldsymbol{\pi}_m||_1\right), \end{aligned} \tag{11}$$

where $\boldsymbol{\lambda}$ summarizes all Lagrange multipliers used here (not to be confused with the regularization hyperparameter λ used in equation 5). Differentiating equation 11 with respect to each model parameter and Lagrange multiplier and solving for the critical points yields:

$$\alpha_{k,m} = \frac{\sum_i p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\text{-}}) \mathbf{1}_{y_i=k}}{\sum_i \mathbf{1}_{y_i=k}}, \tag{12}$$

$$\pi_{m,s} = \frac{\sum_i p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\text{-}}) p(\mathrm{h}_i^{(0)} = s|\mathrm{z}_i = m, \boldsymbol{X}_i, y_i, \theta^{\text{-}})}{\sum_i p(\mathrm{z}_i = m|\mathbf{X}_i, y_i, \theta^{\text{-}})}, \tag{13}$$

$$A_{s,u}^m = \frac{\sum_i p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\text{-}}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t-1)} = s, \mathrm{h}_i^{(t)} = u|\mathrm{z}_i = m, \boldsymbol{X}_i, y_i, \theta^{\text{-}})}{\sum_i p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\text{-}}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t-1)} = s|\mathrm{z}_i = m, \boldsymbol{X}_i, y_i, \theta^{\text{-}})}, \tag{14}$$

$$\boldsymbol{\mu}_{m,s} = \frac{\sum_i p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\text{-}}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t)} = s|\mathrm{z}_i = m, \boldsymbol{X}_i, y_i, \theta^{\text{-}}) \boldsymbol{x}_i^{(t)}}{\sum_i p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\text{-}}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t)} = s|\mathrm{z}_i = m, \boldsymbol{X}_i, y_i, \theta^{\text{-}})}, \tag{15}$$

$$\boldsymbol{\sigma}_{m,s}^2 = \frac{\sum_i p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\text{-}}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t)} = s|\mathrm{z}_i = m, \boldsymbol{X}_i, y_i, \theta^{\text{-}}) \left(\boldsymbol{x}_i^{(t)} - \boldsymbol{\mu}_s^m\right)^2}{\sum_i p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\text{-}}) \sum_{t=1}^{T_i} p(\mathrm{h}_i^{(t)} = s|\mathrm{z}_i = m, \boldsymbol{X}_i, y_i, \theta^{\text{-}})}, \tag{16}$$

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

Defining n_k, $\rho_{i,m}$, $\gamma_{i,m,s}$ and $\xi_{i,m,s,u}$ as in Algorithm 1 the result follows.

A.3 PROOF OF ALGORITHM 2

Using the same notation as in section A.2, we may rewrite equation 5 as:

$$J_r(\theta) = \frac{1}{N} \log \sum_{\mathbf{z},\mathbf{H}} p(\mathbf{X}, \mathbf{z}, \mathbf{H}|\boldsymbol{y}, \theta) + \frac{\lambda}{2} \sum_{j,k \neq j} G_{j,k} \mathbb{E}_{\mathrm{z} \sim p(\mathrm{z}|\mathrm{y}=j,\theta)}[p(\mathrm{z}|\mathrm{y} = k, \theta)]. \tag{17}$$

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

$$\begin{aligned} J_r(\theta) &\geqslant \frac{1}{N} \mathbb{E}_{\mathbf{z},\mathbf{H} \sim q} \left[\log \frac{p(\mathbf{X}, \mathbf{z}, \mathbf{H}|\boldsymbol{y}, \theta)}{q(\mathbf{z}, \mathbf{H})}\right] + \frac{\lambda}{2} \sum_{j,k \neq j} G_{j,k} \mathbb{E}_{\mathrm{z} \sim p(\mathrm{z}|\mathrm{y}=j,\theta)}[p(\mathrm{z}|\mathrm{y} = k, \theta)] \\ &\coloneqq V_r(\theta, q). \end{aligned} \tag{18}$$

Clearly,

$$J_r(\theta) - V_r(\theta, q) = \frac{1}{N}\left(\log p(\mathbf{X}|\boldsymbol{y}, \theta) - \mathbb{E}_{\mathbf{z},\mathbf{H}\sim q}\left[\log \frac{p(\mathbf{X}, \mathbf{z}, \mathbf{H}|\boldsymbol{y}, \theta)}{q(\mathbf{z}, \mathbf{H})}\right]\right)$$
$$= \frac{1}{N} D_{\mathrm{KL}}\left(q(\mathbf{z}, \mathbf{H})||p(\mathbf{z}, \mathbf{H}|\mathbf{X}, \boldsymbol{y}, \theta)\right), \tag{19}$$

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

$$\arg\max_\theta V_r(\theta, q^*) = \arg\max_\theta \frac{1}{N}\sum_{\mathbf{z},\mathbf{H}} p(\mathbf{X}, \mathbf{z}, \mathbf{H}|\boldsymbol{y}, \theta^{\text{-}}) \log p(\mathbf{X}, \mathbf{z}, \mathbf{H}|\boldsymbol{y}, \theta)$$
$$+ \frac{\lambda}{2} p(\mathbf{X}|\boldsymbol{y}, \theta^{\text{-}}) \sum_{j,k\neq j} G_{j,k}\mathbb{E}_{\mathrm{z}\sim p(\mathrm{z}|\mathrm{y}=j,\theta)}[p(\mathrm{z}|\mathrm{y}=k, \theta)]$$
$$= \arg\max_\theta \frac{1}{N}\tilde{J}(\theta, \theta^{\text{-}}) + \lambda R(\theta, \theta^{\text{-}}) \tag{20}$$
$$:= \arg\max_\theta \tilde{J}_r(\theta, \theta^{\text{-}}), \tag{21}$$

where $\tilde{J}(\theta, \theta^{\text{-}})$ is as defined in equation 10 and $R(\theta, \theta^{\text{-}})$ is our regularization weighted by the data likelihood, which is simply a function of the parameters $\boldsymbol{\alpha}$:

$$R(\theta, \theta^{\text{-}}) = \frac{1}{2} p(\mathbf{X}|\boldsymbol{y}, \theta^{\text{-}}) \sum_{j,k\neq j} G_{j,k}\mathbb{E}_{\mathrm{z}\sim p(\mathrm{z}|\mathrm{y}=j,\theta)}[p(\mathrm{z}|\mathrm{y}=k, \theta)]$$
$$= \frac{1}{2} p(\mathbf{X}|\boldsymbol{y}, \theta^{\text{-}}) \sum_{j,k\neq j} G_{j,k}\boldsymbol{\alpha}_j^\intercal \boldsymbol{\alpha}_k$$
$$= R(\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_k, \theta^{\text{-}}) \tag{22}$$

Now, we may build the Lagrangian as done in section A.2. Since R only depends on the $\boldsymbol{\alpha}$'s, the update equations for the remaining parameters are unchanged. However, for $\boldsymbol{\alpha}$, it is not possible to obtain a closed form update equation. Thus, we use the reparameterization defined in equation 8 and update the new unconstrained parameters $\boldsymbol{\beta}$ via gradient ascent.
We have:

$$\frac{\partial \tilde{J}}{\partial \alpha_{k,m}} = \frac{p(\mathbf{X}|\boldsymbol{y}, \theta^{\text{-}})}{\alpha_{k,m}} \sum_i p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\text{-}}) \mathbf{1}_{y_i=k}, \tag{23}$$
$$\frac{\partial R}{\partial \alpha_{k,m}} = p(\mathbf{X}|\boldsymbol{y}, \theta^{\text{-}}) \sum_{j\neq k} G_{j,k}\alpha_{j,m}. \tag{24}$$

From equations 23 and 24, we see that the the resulting gradient $\nabla_{\boldsymbol{\alpha}_k}\tilde{J}_r = \nabla_{\boldsymbol{\alpha}_k}\tilde{J} + \lambda\nabla_{\boldsymbol{\alpha}_k}R$ is equal to some vector scaled by the joint data likelihood $p(\mathbf{X}|\boldsymbol{y}, \theta^{\text{-}})$, which we discard since it only affects the learning rate, besides being usually very small and somewhat costly to compute. This option is equivalent to using a learning rate that changes at each iteration of the outter loop of the algorithm.
Equation 8 yields the following derivatives:

$$\frac{\partial \alpha_{k,m}}{\partial \beta_{k,m}} = \mathbf{1}_{\beta_{k,m}>0} \frac{2\sigma'(\beta_{k,m})}{\sigma(\beta_{k,m})} \alpha_{k,m}(1 - \alpha_{k,m}), \tag{25}$$
$$\frac{\partial \alpha_{k,m}}{\partial \beta_{k,l}} = \mathbf{1}_{\beta_{k,m}>0} \frac{-2\sigma'(\beta_{k,m})}{\sigma(\beta_{k,m})} \alpha_{k,m}\alpha_{k,l}, \text{ for } l \neq m. \tag{26}$$

Finally, by the chain rule, we obtain:

$$\begin{aligned}\frac{\partial \tilde{J}}{\partial \beta_{k,m}} &= \sum_l \frac{\partial \tilde{J}}{\partial \alpha_{k,l}} \frac{\partial \alpha_{k,l}}{\partial \beta_{k,m}} \\ &= \mathbf{1}_{\beta_{k,m}>0} \frac{2\sigma'(\beta_{k,m})}{\sigma(\beta_{k,m})} \sum_i \left(p(\mathrm{z}_i = m|\boldsymbol{X}_i, y_i, \theta^{\cdot}) - \alpha_{k,m}\right) \mathbf{1}_{y_i=k}, \qquad (27) \\ \frac{\partial R}{\partial \beta_{k,m}} &= \sum_l \frac{\partial R}{\partial \alpha_{k,l}} \frac{\partial \alpha_{k,l}}{\partial \beta_{k,m}} \\ &= \mathbf{1}_{\beta_{k,m}>0} \frac{2\sigma'(\beta_{k,m})}{\sigma(\beta_{k,m})} \alpha_{k,m} \sum_{j \neq k} G_{j,k} \left(\alpha_{j,m} - \boldsymbol{\alpha}_j^{\intercal} \boldsymbol{\alpha}_k\right). \qquad (28)\end{aligned}$$

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

A.4 Getting the posterior distribution of observations in SpaMHMM

In this section, we show how to obtain posterior distribution $p(\mathbf{X}|\boldsymbol{X}_{\text{pref}}, y)$ of sequences $\mathbf{X} = \left(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)}\right)$ given an observed prefix sequence $\boldsymbol{X}_{\text{pref}} = \left(\boldsymbol{x}^{(-T_{\text{pref}}+1)}, ..., \boldsymbol{x}^{(0)}\right)$, both coming from the graph node y. We consider the case where $p(\mathbf{X}|\mathrm{y})$ is a SpaMHMM (or MHMM) model and so we have:

$$p(\mathbf{X}|\boldsymbol{X}_{\text{pref}}, y) = \sum_{\mathrm{z}} p(\mathbf{X}|\mathrm{z}, \boldsymbol{X}_{\text{pref}}, y) p(\mathrm{z}|\boldsymbol{X}_{\text{pref}}, y) = \sum_{\mathrm{z}} p(\mathbf{X}|\mathrm{z}, \boldsymbol{X}_{\text{pref}}) p(\mathrm{z}|\boldsymbol{X}_{\text{pref}}, y), \qquad (29)$$

where the second equality follows from the fact that the observations $\mathbf{X}$ are independent from the graph node y given the latent variable z. The posterior $p(\mathrm{z}|\boldsymbol{X}_{\text{pref}}, y)$ may be obtained as done in Algorithm 1, so we now focus on $p(\mathbf{X}|\mathrm{z}, \boldsymbol{X}_{\text{pref}})$:

$$\begin{aligned}p(\mathbf{X}|\mathrm{z}, \boldsymbol{X}_{\text{pref}}) &= \sum_{\mathbf{h}} p(\mathrm{h}^{(0)}|\mathrm{z}, \boldsymbol{X}_{\text{pref}}) \prod_t p(\mathrm{h}^{(t)}|\mathrm{h}^{(t-1)}, \mathrm{z}, \boldsymbol{X}_{\text{pref}}) p(\mathbf{x}^{(t)}|\mathrm{h}^{(t)}, \mathrm{z}, \boldsymbol{X}_{\text{pref}}) \\ &= \sum_{\mathbf{h}} p(\mathrm{h}^{(0)}|\mathrm{z}, \boldsymbol{X}_{\text{pref}}) \prod_t p(\mathrm{h}^{(t)}|\mathrm{h}^{(t-1)}, \mathrm{z}) p(\mathbf{x}^{(t)}|\mathrm{h}^{(t)}, \mathrm{z}), \qquad (30)\end{aligned}$$

where we have used the independence assumptions of an HMM. Here, the initial state posteriors $p(\mathrm{h}^{(0)}|\mathrm{z}, \boldsymbol{X}_{\text{pref}})$ 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 1.

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