GENERATIVE MODELING OF SEQUENTIAL DATA

BY

Y. CEM SUBAKAN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Doctoral Committee:

Associate Professor Paris Smaragdis, Chair
Professor David Forsyth,
Professor Mark Hasegawa-Johnson,
Dr. Yunus Saatci, Uber AI LABS

ABSTRACT

In this thesis, we investigate various approaches for generative modeling, with a special emphasis on sequential data. Namely, we develop methodologies to deal with issues regarding representation (modeling choices), learning paradigm (e.g. maximum likelihood, method of moments, adversarial training), and optimization.

For the representation aspect, we make the following contributions:

- We argue that using a multi-modal latent representation (unlike popular methods such as variational autoencoders or generative adversarial networks) significantly enhances the generative model learning performance, as evidenced by the experiments we conduct on handwritten digit dataset (MNIST) and celebrity faces dataset (CELEB-A).
- We prove that the standard factorial Hidden Markov model defined in the literature is not statistically identifiable. We propose two alternative identifiable models, and show their validity on unsupervised source separation examples.
- We experimentally show that using a convolutional neural network architecture provides performance boost over time agnostic methods such as non-negative matrix factorization, and auto-encoders.
- We experimentally show that using a recurrent neural network with a diagonal recurrent matrix increases the convergence speed and final accuracy of the model in most cases in a symbolic music modeling task.

For the learning paradigm aspect, we make the following contributions:

- We propose a method of moment based parameter learning framework for Hidden Markov Models (HMMs) with special transition structures such as mixture of HMMs, switching HMMs and HMMs with mixture emissions.
- We propose a new generative model learning method which does approximate maximum likelihood parameter estimation for implicit generative models.
- We argue that using an implicit generative model for audio source separation increases the performance over models which specify a cost function, such as NMF or autoencoders trained via maximum likelihood. We show performance improvement in speech mixtures created from the TIMIT dataset.

For the optimization aspect, we make the following contributions:

- We show that using the method of moment framework we propose in this thesis boosts the model performance when used as an initialization scheme for the expectation maximization algorithm.
- We propose new optimization algorithms for identifiable alternatives to Factorial HMM.
- We propose a two-step optimization algorithm for learning implicit generative models which efficiently learns multi-modal latent representations.

To all kindred spirits

ACKNOWLEDGMENTS

Before anyone else the credit of course goes to my advisor Paris Smaragdis. As my dear labmate Johannes Traa once wrote in his thesis acknowledgements page, Paris really is a real kick-ass advisor (in the best way). Thank you Paris for tolerating all my crazy actions, giving me the freedom to do whatever I want to do, and saving me from the downward slope I was in during my 3rd year.

Also, Bulent hoca and Taylan hoca for all their support during my masters and undergrad. They were quite instrumental in motivating me to pursue a career in research.

I also want to thank Sanmi Koyejo for the collaboration which made Chapter 4 of this thesis possible.

Johannes Traa, the name I already mentioned, has been central in this thesis. He was my co-author for the first half of my Ph.D. (method of moments stuff and the first factorial hmm paper). But much more importantly he has been a great friend from whom I believe I learned a lot.

Minje Kim, who is now a serious college professor also helped me a lot in gaining self confidence. We never got to collaborate, but maybe someday.

I would like to thank Yves Petinot for being a kindred spirit.

My lab-mate Shrikant Venkataramani was also instrumental in my recovery in my fourth year. We wrote the convolutive neural network paper together in the dining hall of Uber's headquaters, and got a best paper award for it. Quite awesome stuff.

I also want to thank my friend Sertan Alkan for being a real friend throughout the years.

I also want to thank my labmates in office 3332 for tolerating all my assholish and territorial behavior.

Also thank you father and mother for all the support in all aspects, and allowing me to be and do whatever I want.

(I still don't have a significant other so I pass on that one.)

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 GENERATIVE MODELING

Generative modeling is a statistical modeling approach where one learns a distribution over a given data by assuming a process from which the data is generated. Applications can range from modeling the price distribution for a given stock, to recognizing handwritten digits, human faces, from de-noising speech signals to generating random bedroom pictures.

In general, given a dataset whose underlying distribution is described by $p_{\text{data}}(x)$, the goal in generative model learning is to fit a model with probability density $p_{\text{model}}(x|\theta)$, such that $d(p_{\text{data}}(x)\|p_{\text{model}}(x|\theta))$ is minimized, where $d(.)$ is some divergence measure, and θ denotes the model parameters.

To see generative modeling in action, let us consider the dataset [1] for weight and height of members of the !Kung tribe in Africa [2]. Let us first start by modeling the dataset for the adult members who are over the age of 20. We assume the following generative process:

$$x_n \sim \mathcal{N}(\mu, \Sigma), \ \forall n \in \{1, \ldots, N\},$$

where $x_n \in \mathbb{R}^2$ denotes the data item with index n, $\mathcal{N}(\mu, \Sigma)$ denotes a multivariate Gaussian distribution with mean $\mu \in \mathbb{R}^2$, and covariance matrix $\Sigma \in \mathbb{R}^{2\times 2}$ (Therefore the model parameters are $\theta = \{\mu, \Sigma\}$). We see in the left panel of Figure 1.1, that this is a reasonable assumption since the data points roughly form an ellipse. We learn the distribution parameters by maximizing the following objective:

$$\max_{\theta} \sum_{n=1}^{N} \log p_{\text{model}}(x_n|\theta) = \max_{\mu,\Sigma} \sum_{n=1}^{N} \log \mathcal{N}(x_n; \mu, \Sigma), \tag{1.1}$$

which is the maximum likelihood objective for this model. In the left panel of Figure 1.1, we see that the samples drawn from the learned distribution closely follow the original data items. Now, let us consider a slightly more complicated situation where we model children and adults together. We see from the left panel of Figure 1.2 that this dataset now exhibits a bi-modal behavior, and therefore we can not simply use a multivariate Gaussian. We can instead devise the following generative process which emulates the bi-modal behavior in the

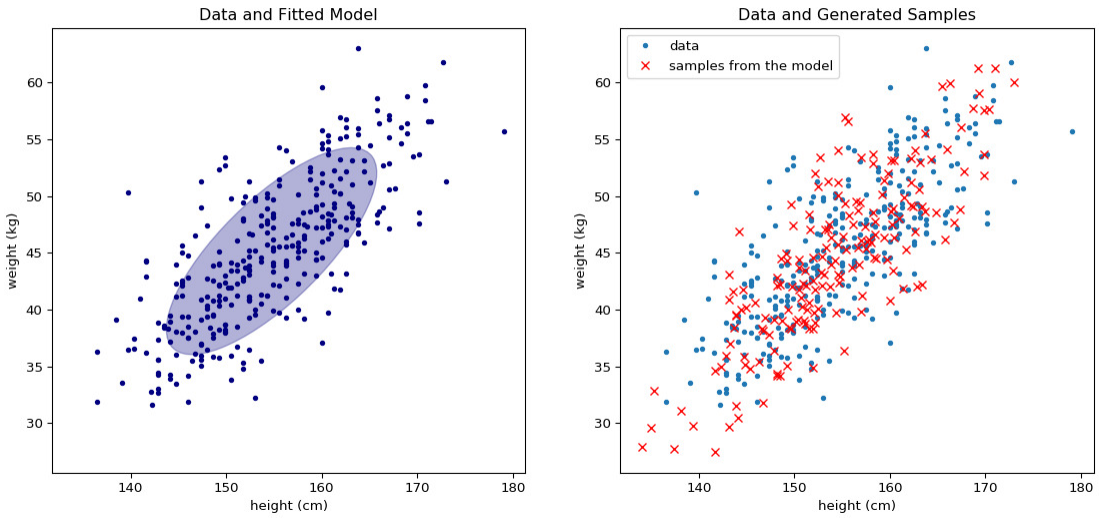


Figure 1.1: Learning a joint distribution over weight and height for the members over the age of 20 of the !Kung tribe in Africa. **(left)** Real Data and the covariance ellipsoid for the learned distribution. **(right)** Real data and samples drawn from the learned distribution.

dataset:

$$\begin{aligned} h_n &\sim \mathcal{BE}(\pi), \\ x_n|h_n &\sim \mathcal{N}(\mu_{h_n}, \Sigma_{h_n}). \end{aligned}$$

Note that the above process describes our assumption on how an observed data item is generated: We first sample the indicator variable $h_n \in \{0, 1\}$, which selects the output distribution. Then once the indicator variable is given, we sample from the selected output distribution. The parameters $\theta = \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, are again learned via maximum likelihood, usually with the EM algorithm [3, 4]. We see in Figure 1.2 that using this generative model enables us to generate samples which closely follow the original data distribution $p_{\text{data}}(x)$.

When the dataset comprises of scalars, or of low dimensional (2-3 dimensional) vectors and follow a unimodal distribution as in Figure 1.1, we therefore see that one can use a simple density model such as the multivariate Gaussian. And when the data exhibits a multi-modal behavior as in Figure 1.2 we can use linear latent variable models such as GMMs (the model described above). However, the modern goals in generative model learning revolve around learning distributions over more complicated data such as natural images, which are typically high dimensional and multi-modal in nature, for which linear models (even though multi-modal) fall short. And, if we are modeling data with temporal structure, such as audio or text, we need to model the temporal structure in the data.

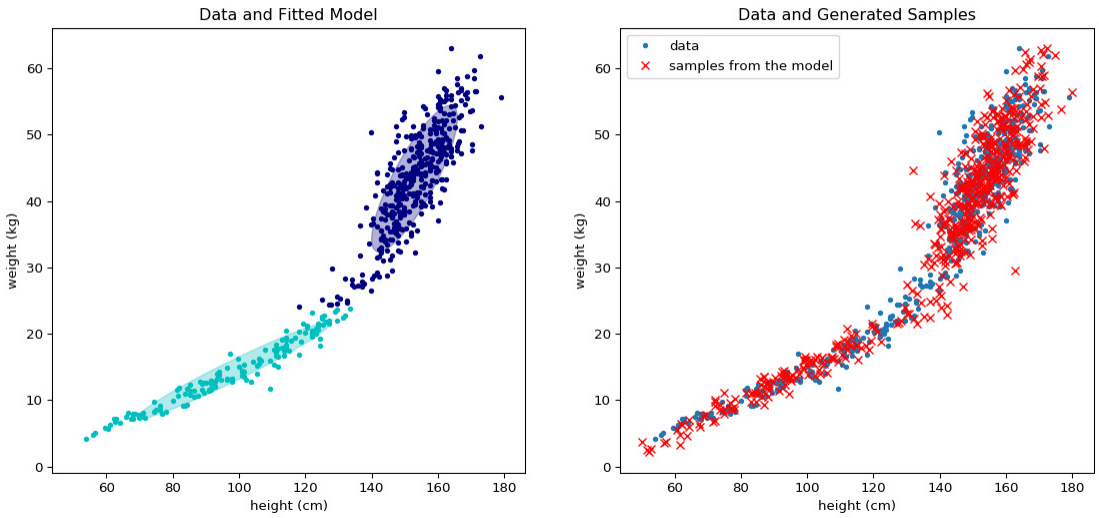


Figure 1.2: Learning a joint distribution over weight and height for the adult and child members of !Kung tribe in Africa. **(left)** Real Data and the covariance ellipsoid for the learned distribution. **(right)** Real data and samples from the learned distribution.

To give a concrete example, we consider learning a distribution over handwritten digit images from the MNIST dataset [5], where each digit image is in grayscale and of size 28×28 (therefore 784 dimensional vectors). In Figure 1.3, we show samples from the dataset, along with artificial samples generated using models with various complexity. In this example we see that there is a clear correlation between the model complexity and the quality of the generated samples: We see that using a multivariate Gaussian with a diagonal covariance matrix is only able to learn the region on which the image have non-zero values, but is unable to capture the digit shapes. Using a full covariance matrix enables the model to get the rough shapes. Using a GMM helps, but the generated samples still contain artifacts. We see that using the non-linear model suggested in Chapter 4, we get much clearer samples. The proposed model uses both latent variable modeling and non-linear neural networks which have been extremely popular in recent years.

Overall, in my view training a successful generative model amounts to dealing with the following issues:

- **Representation:** This item is concerned with how we model the generative process. For instance, in our example for modeling the height and weight of the !Kung tribe, and in a more complicated problem involving learning a distribution over hand written digits, we have seen that the model specification played a crucial role. Namely, in the case where we wanted to model the distribution jointly for adults and children, we

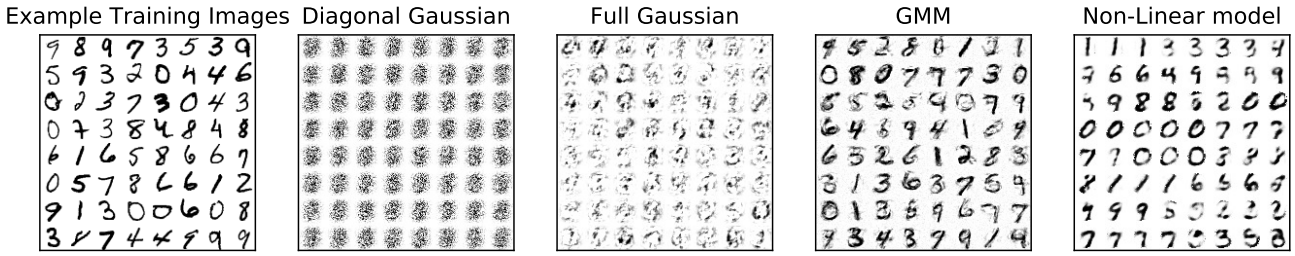


Figure 1.3: Learning a distribution over handwritten digits (MNIST dataset - images are of size 28×28) with models of increasing complexity. Leftmost figure shows samples from the training set. Rest of the figures show random samples drawn from the learned models. The model complexity increase as we go from left to right, and the used models indicated on top of each image.

have seen that using a latent variable enabled accurate modeling. In the literature there exists successful generative models which do not use latent variables such as autoregressive models [6], or models [7, 8] that make use of recurrent neural networks [9, 10].

Using latent variables adds more flexibility and interpretability, as shown in Figure 1.2, although learning becomes more complicated. A typical generative model that makes use of latent variables can roughly be generalized with the following generative process [11]:

$$\begin{aligned} h &\sim p(h), \\ x|h &\sim p_{\text{forward}}(x|h,\theta), \end{aligned}$$

where $h \in \mathbb{R}^K$ is the latent variable, and can be a discrete or continuous variable. The observations $x \in \mathbb{R}^L$ are generated conditioned on the latent variable h, and are drawn from the distribution $p_{\text{forward}}(x|h,\theta)$ which characterizes the distribution of the outputs given the latent representation h. The output distribution is typically parametrized by a deterministic mapping such that $p_{\text{forward}}(x|h,\theta) = p_{\text{out}}(x; f_\theta(h))$, where $p_{\text{out}}(.)$ characterizes the output noise and $f_\theta(h) : \mathbb{R}^K \to \mathbb{R}^L$ is a deterministic mapping (typically non-linear). If for instance $p_{\text{out}}(.)$ is a Poisson distribution, $f_\theta(h)$ predicts the Poisson intensities. Note that the model distribution is defined via the

following integral:

$$p_{\text{model}}(x|\theta) = \int p_{\text{forward}}(x|h,\theta)p(h)dh,$$
$$= \int p_{\text{out}}(x; f_\theta(h))p(h)dh, \tag{1.2}$$

which is intractable in general. There exists various approaches to estimate $p_{\text{model}}(x|\theta)$ in the literature, which brings us to the next issue of choosing the learning paradigm. The contributions in this thesis for the representation problem are as follows:

- In Chapter 3, we show that the standard factorial HMM model [12, 13], is not statistically identifiable. We therefore propose alternative identifiable models.
- In Chapter 4, we argue that using a multimodal latent representation is fundamental in obtaining accurate generative models.
- In Chapter 5, we show that using convolutional neural network mappings greatly enhances the source separation performance in source separation.
- In Chapter 5, we also show that using a diagonal recurrent matrix in recurrent neural networks enhances the algorithm performance on symbolic music prediction.

- **Choice of divergence / Learning Paradigm:** Arguably, the most common approach for training a generative model is the maximum likelihood principle. As we show in Chapter 4, this minimizes the divergence $\text{KL}(p_{\text{data}}(x)\|p_{\text{model}}(x|\theta))$. In the case where one uses fully observable models such as autoregressive or recursive models we mentioned above, one can directly obtain $p_{\text{model}}(x|\theta)$. However, when latent variables are used $p_{\text{model}}(x|\theta)$ is not directly available. A very common approach is to introduce a lower bound to model likelihood and iteratively maximize this lower bound which is known as the variational lower bound or evidence lower bound (ELBO) [14, 4, 15].

  A recent, but hugely popular training method known as "Generative Adversarial Networks" [16] however, completely drops the maximum likelihood paradigm and instead trains model parameters by estimating the ratios between model density and the data density. We specify the inner workings of GANs in Chapter 4.

  Another way of training generative models is by moment matching: For linear models such as HMMs or GMMs, it is possible to convert the parameter estimation problem into solving a set of equations, which can be reduced into an eigenvalue-eigenvector

estimation problem [17, 18, 19]. The advantages include guarantees for converging to a global optimum, and avoiding the need for multiple restarts.

The contributions in this thesis in terms of learning are as follows:

- In Chapter 2, we propose a method of moments (moment matching) based framework for learning in Hidden Markov Model variants with special transition structures.
- In Chapter 4, we propose a maximum likelihood based method for learning implicit generative models.
- In Chapter 5, we argue that using GANs in audio modeling enables performance increase in speech source separation.

- **Optimization:** Optimization for generative models usually involve optimizing non-convex loss functions. The gradient based methods, or the EM algorithm [3], usually tend to get stuck in local optima or get slow near saddle points [20]. Our contributions on this front are as follows:
  - In Chapter 2, we show that the proposed method of moments based framework can be used as an initialization scheme for the EM algorithm, to boost accuracy and increase the speed of the whole optimization procedure.
  - In Chapter 3, we propose a new learning algorithm for Factorial HMMs [21].
  - In Chapter 4, we propose a two step optimization algorithm which maximizes an approximate maximum likelihood objective.

In this thesis, we are particularly interested in learning distributions over sequential data. We therefore give an overview of existing latent variable and neural network approaches for modeling sequential data in the next section.

1.2 SEQUENCE MODELING

Many machine learning problems involve sequence data. Examples include sequence classification, sequence clustering, sequence prediction (e.g. language modeling, market prediction) sequence to sequence mapping (e.g. machine translation), non-sequence to sequence mapping (e.g. image captioning), sequence segmentation, de-noising, source separation, and many more.

In highest level, sequence models can be classified according to whether the model uses latent variables or not. Latent variables are a tool to model uncertainty and are extremely helpful in modeling complex stochastic dependencies. However, one does not always need to use latent variables, and in recent years we have seen successful application of neural networks (which are fully observable models) in many domains (e.g. speech [22], vision [23], natural language processing [24, 25]). Both approaches are valid and common place in machine learning.

In the rest of this introductory section, we give brief introductions into latent variable sequence models and observable sequence models. We describe latent variable sequence models via examples, and when we come to the relevant models we specify/highlight our related paper.

1.2.1 Latent Variable Sequence Modeling

Latent Variable sequence models assume that observations are conditioned on an underlying hidden (latent) random process. Latent variables enable modeling of complex dependencies and are quite useful for incorporating prior knowledge about the problem at hand. Few examples for latent variable sequence models are given below:

- **Hidden Markov Model/Linear Dynamical System:** In Hidden Markov Models (HMMs) [26] and Linear Dynamical Systems (LDS), is it assumed that the underlying process is a Markov process. The observations are conditioned directly on this Markov process. The generative model for a given sequence $x_{1:T}$ is therefore constructed as follows:

$$\begin{aligned} h_t|h_{t-1} &\sim p(h_t|h_{t-1}) \\ x_t|h_t &\sim p(x_t|h_t) \end{aligned}$$

  In an HMM, the underlying process is discrete. Therefore the conditional state distribution $p(h_t|h_{t-1})$ is a discrete distribution. In contrast, in Linear Dynamical Systems the underlying process is real valued, and $p(h_t|h_{t-1})$ is a Gaussian distribution. The emission distribution $p(x_t|h_t)$ depends on the application at hand, and can be chosen accordingly. For both models, the corresponding dependency graph is given in Figure 1.4. HMMs are used in a plethora of domains such as speech recognition, hand writing recognition, human action recognition, and many more [4].

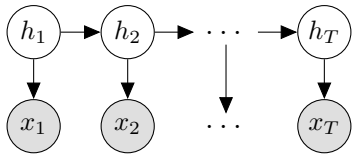


Figure 1.4: HMM Dependency graph.

- **Left-to-right HMM** One other example is an HMM where state transitions can only happen in one direction. That is, the hidden state can only increase or stay the same with increasing time index. This implies that the state transition matrix is lower triangular (or upper triangular depending on whether we use column stochastic or row stochastic transition matrix). One important variant of Left-to-right HMMs is Bakis HMM where the state index can only increase by one step. This model is useful in speech phoneme decoding and genetic base sequence modeling. **In [27], we derive a method of moments based learning algorithm for left-to right and Bakis HMMs.**

- **Explicit Duration Model:** Although this model is not contained in this thesis, let us briefly describe it to better illustrate the idea behind using latent variables in sequence modeling. In HMMs, the state durations are geometrically distributed. In applications where one has the prior knowledge on the state durations, we might want to explicitly model this quantity. Explicit Duration Model [28], precisely does this. For example, in [29] explicit duration model is used for hand gesture recognition, where the implicit geometric distribution on the state durations in HMMs, might lead to inaccurate modeling. The generative process is as follows:

$$\begin{aligned} d_t|d_{t-1}, h_{t-1} &\sim [d_{t-1} = 0]F_d(\lambda_{h_{t-1}}) + [d_{t-1} > 0]\delta(d_t - d_{t-1} + 1), \\ h_t|h_{t-1}, d_{t-1} &\sim [d_{t-1} = 0]\text{Discrete}(\pi_{h_{t-1}}) + [d_{t-1} > 0]\delta(h_t - h_{t-1} + 1), \end{aligned}$$

  where, d_t denotes the remaining duration for the current state at time t, $F_d(\lambda_{h_{t-1}})$ is the distribution that samples the state durations, and $\pi_{h_{t-1}}$ is the next state distribution when the duration counter hits zero.

- **Mixture of HMMs:** If we would like to cluster sequences while modeling them with HMMs, we can use mixture of HMMs [30]. The high level generative process can be

written as follows:

$$\begin{aligned} h_n &\sim \text{Discrete}(\pi), \\ \mathbf{x}_n &\sim \text{HMM}_{h_n}(\mathbf{x}_n), \end{aligned}$$

where the idea is to model a given sequence $\mathbf{x}_n$ using one of K different HMM clusters. The cluster of a given sequence is indicated by the latent variable h_n. The dependency graph of mixture of HMMs model is given in Figure 1.5. **In [31], we derive a method of moments based algorithm for mixture of HMMs.**

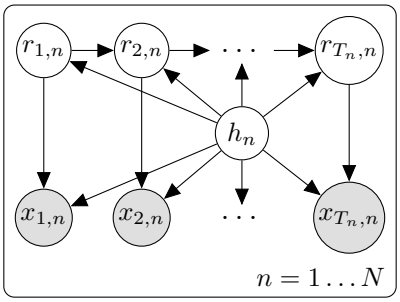


Figure 1.5: Dependency graph of a mixture of HMMs model.

- **Switching HMMs (SHMM):** We can easily construct a sequence segmentation model by modifying MHMM model above to have the model to switch between HMMs with a given sequence.

$$\begin{aligned} h_t|h_{t-1} &\sim \text{Discrete}(\pi_{h_{t-1}}) \\ r_t|r_{t-1}, h_t, h_{t-1} &\sim \text{Discrete}(A_{r_{t-1}})[h_t = h_{t-1}] + \mathcal{U}(1, K)[h_t \neq h_{t-1}] \\ x_t|r_t, h_t &\sim p(x_t|r_t, h_t) \end{aligned}$$

The dependency graph of SHMM is given in Figure 1.6. **In [32], we extend our method of moments for mixture of HMMs, to handle switching HMMs, and HMMs with mixture emissions (which is related to MHMM and SHMM).**

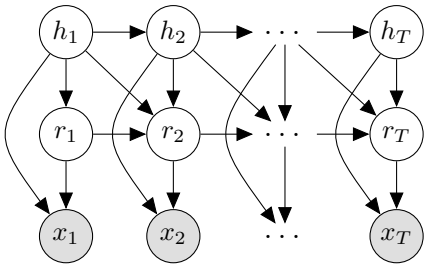


Figure 1.6: SHMM dependency graph.

- **HMM with mixture observations:** We note that an HMM with mixture emissions is a special case of Switching HMM. We can get an HMM with mixture emissions (HMM-M), out of an SHMM, if we set the transition matrices to be of the form $\rho_k 1^T$, where ρ_k is the mixing weights of the mixture that corresponds to the k'th HMM state. The corresponding generative process is as follows:

$$\begin{aligned} h_t|h_{t-1} &\sim \text{Discrete}(\pi_{h_{t-1}}) \\ r_t|h_t &\sim \text{Discrete}(\rho_{r_t}) \\ x_t|r_t, h_t &\sim p(x_t|r_t, h_t) \end{aligned}$$

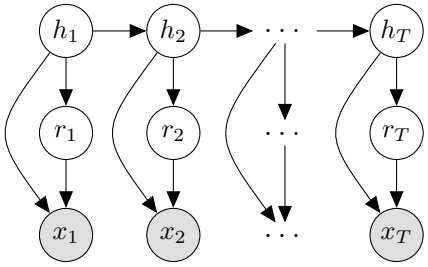


Figure 1.7: The dependency graph for an HMM with mixture emissions.

- **Factorial HMMs:** In the Factorial HMM model originally proposed in [21] the observed sequence is conditioned on K independent latent chains. Namely, the observation sequence is assumed to be the sum of the emissions from these K different latent chains. The goal is to model the mixing of the signal in a cocktail party [33] type setting: There are K independent speakers, and their voices add up to constitute what the cocktail party attendee hear. In the model we discuss here, there is only one

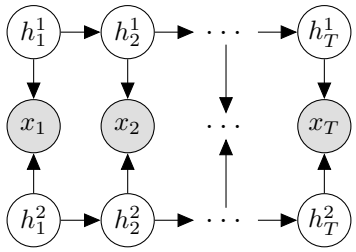


Figure 1.8: Dependency Graph of FHMM for the case $K = 2$.

observation sequence, and the inferential goal is to find the "sources" that make up the original signal. The generative process is written as follows:

$$\begin{aligned}
\mathbf{h}^1 &\sim \text{Markov}(A^1, \pi^1) \\
&\vdots \\
\mathbf{h}^K &\sim \text{Markov}(A^K, \pi^K) \\
\mathbf{x}|\mathbf{h}^{1:K} &= \sum_{k=1}^{K} O^k \mathbf{h}^k + \epsilon,
\end{aligned}$$

where ϵ is some Gaussian noise. The corresponding dependency graph for the case where $K = 2$ is given in Figure 1.8. As can be guessed from the generative process above, learning the model parameters for an FHMM is an ill-posed problem. **In our work [34], we prove that even if the true assignments $h^{1:K}$ are given, it is impossible to identify the emission parameters $O^{1:K}$. We then propose two different alternative identifiable factorial models in [34] and [35], and suggest algorithms which recover the parameters under certain assumptions.**

- **Random Fields:** Random fields are graphical models where the overall probability distribution is defined via local potential functions. The generative version is known as the Markov Random Field, which was first introduced for image processing [36]. There is also a discriminative version known as "conditional random fields" [37]. The likelihood function for an HMM analog, implemented as a random field is given in Equation 1.3.

$$p(x_{1:T}|\theta) = \sum_{h_{1:T}} \prod_{t=1}^{T} \exp\left(\sum_{x,y} \theta_{x,y} \psi_h(x,y) + \sum_{i,j} \theta_{i,j} \psi_x(i,j) \right) \tag{1.3}$$

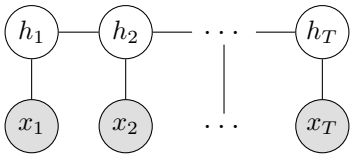


Figure 1.9: Undirected Graph of a MRF/CRF that is equivalent to an HMM.

If the potential functions are chosen as $\psi_h(x, y) = [h_t = x][h_{t-1} = y]$, and $\psi_x(i, j) = [h_t = i][x_t = j]$, then this model is equivalent to an Hidden Markov model. But note that there are other plausible choices for the potential functions, which makes undirected graphs flexible. Also note that, if some of the hidden variables are given, it is possible to train the model discriminatively (as mentioned before, this case is usually called conditional random field (CRF)). The corresponding undirected graph is given in Figure 1.9. Although we do not study random fields in this thesis, we have them in this for the sake of completeness.

Overall, latent variable models assume an underlying stochastic structure, and in learning time the goal is to infer the structure by learning the model parameters. As we will discuss in Chapters 2, 4, learning in latent variable models in general is problematic as optimizing the parameters by maximizing the model likelihood requires us to solve a highly non-convex optimization problem. In Chapter 2, we propose a method of moments based framework (which is applicable on MHMM, SHMM, left-to-right HMM, and HMM with mixture emissions) which circumvents the non-convex maximum likelihood optimization.

1.2.2 Observable Sequence Modeling

Observable sequence models are function approximations of the underlying process. As we have seen in Section 1.2.1, latent variable models model the underlying stochasticity. The observable sequence models do not do this and directly map past observations/features to current observation's probability distribution. Namely, the probability of an observation $x_t \in \mathbb{R}^L$ is typically modeled as, $p(x_t) = f_\theta(x_{1:t-1})$, where $f_\theta : \mathbb{R}^{L\times(T-1)}$ is some function parametrized by θ that maps the past observations to the current observation. A popular choice these days for the function f is neural networks. We give some examples for observable sequence models below, and we summarize them in Figure 1.14.

- **Markov Models:** The basic idea for a Markov model is that given the past observations, there is a probability for the current observation. Let us consider the case where

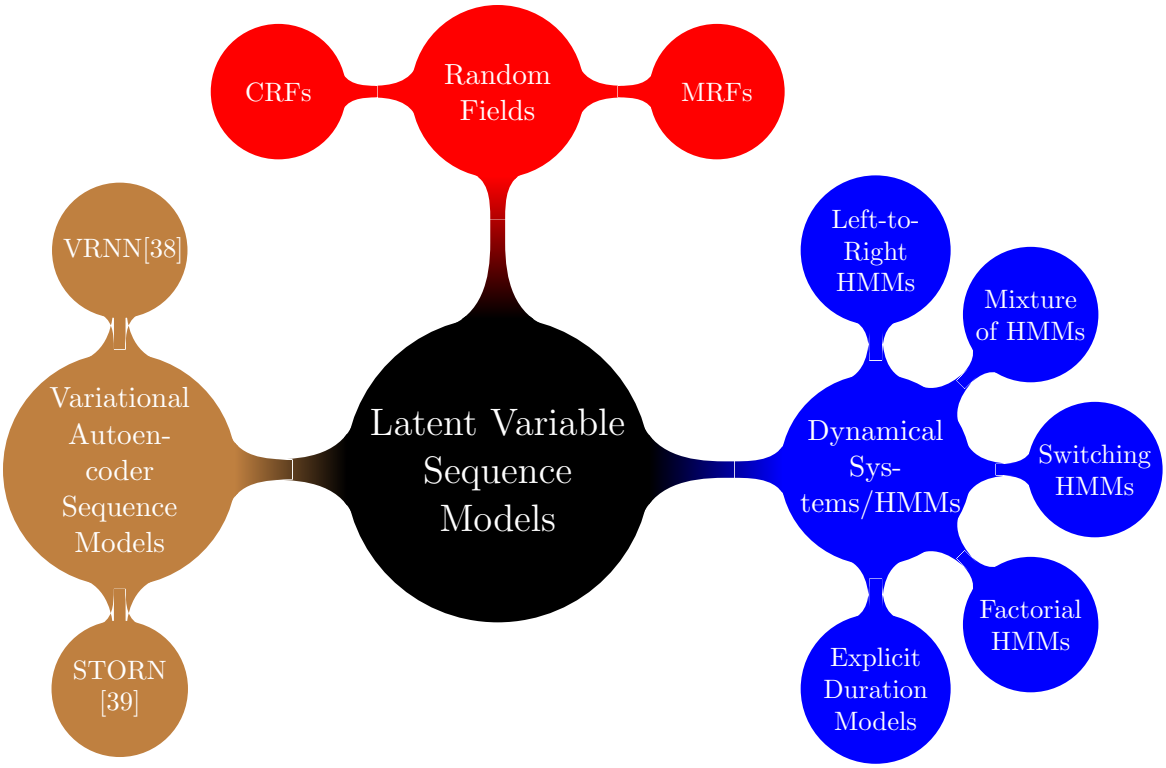


Figure 1.10: Summary of examples for latent variable sequence models given above.

we only look one step into the past, which is known as the first order Markov model. The probability of the current observation is modeled as $p(x_t) = Ap(x_{t-1})$, where A is some transition matrix, and $\theta = \{A\}$. Notice that the transition matrix has to have columns which sum up to one. An alternative implementation for this model is to have a "softmax" non-linearity after the matrix multiplication to ensure that the predicted vector for $p(x_t)$ sum up to one, such that $p(x_t) = \text{Softmax}(Ax_{t-1})$. The dependency graph for a first order Markov model is given in Figure 1.11.

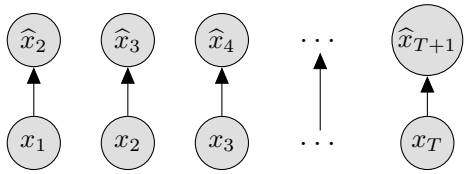


Figure 1.11: Dependency graph of a (first order) Markov model.

- **Convolutive Models:** Convolutive models express the probability of current observation as a combination of past observations/features. These models are known

as "Autoregressive/Moving Average Models" in statistics and "Convolutional Neural Networks" in machine learning. The functional form looks as follows: $p(x_t) = \sigma(\sum_{t'=1}^{n} A_{t-t'} x_{t'})$, where A_t is either a vector or a matrix which transforms the corresponding observation, and $\sigma(.)$ is some output-non linearity function, and n is the number of time steps the model looks back, which is also known as number of filter taps in digital signal processing. Note that, conventional Autoregressive Moving Average (ARMA) models are special case of these models. The dependency graph for the case $n = 2$ is given in Figure 1.12. **We implemented analog of the convolutive Non-Negative Matrix factorization model [40], in our recent paper [41].**

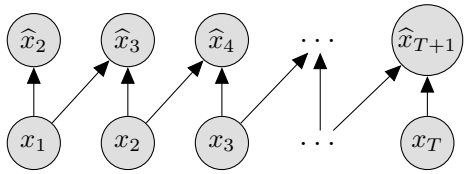


Figure 1.12: The dependency graph of convolutive sequence models. We use the shorthand $\widehat{x}_t$ to denote the estimate for $p(x_t)$. Note that $n = 2$ for this example.

- **Recurrent Models:** Convolutive networks are usually very powerful, and perform great. However their drawback is they can only incorporate information from a fixed number of past observations when predicting for the current observation. Recurrent models aim to incorporate all past information by recursively defining the model output. Namely, in a usual setup the probability for current output is $p(x_t) = \sigma(h_t)$, where $h_t = \sigma(W h_{t-1} + U x_t)$. Although in theory this seems heavenly, in practice it is quite difficult to train recurrent networks in this simple formulation [42]. The famous LSTM networks [9, 43], or GRU networks [10] alleviate this. The dependency graph for a general recurrent model is given in Figure 1.13 (Note that the same graph with arbitrarily long dependencies can be obtained with a convolutive model, but the advantage that comes with recurrent models is the model automatically models arbitrarily long dependencies without the need to have a bigger model, (unlike the convolutive models, where the model size increases with number of dependencies that is modeled)). **In our work [44], we propose using a diagonal recurrent matrix (diagonal W instead of full), and empirically show performance improvement. The dependency graph is given in Figure 1.13. We also use recurrent models in audio source separation in our paper [41].**

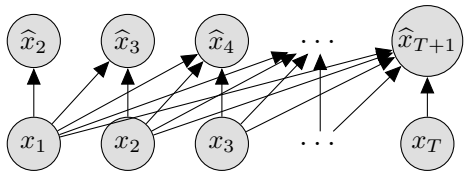


Figure 1.13: The dependency graph for recurrent sequence models.

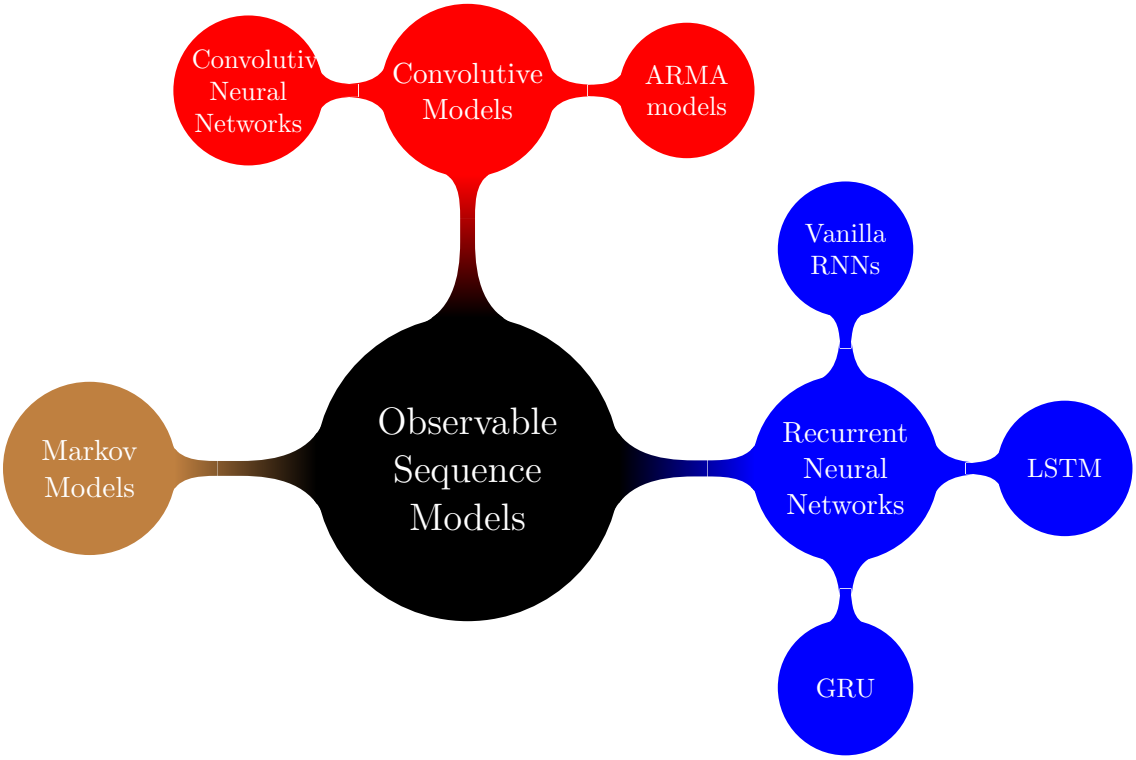


Figure 1.14: Summary picture of the examples given for observable sequence models.

Finally, we would like to note here that in the last few years, there has been works which incorporates stochasticity via latent variables in neural networks, which is known as "Variational Autoencoders" (VAE). Namely, the marginal distribution for the observations x is modeled as:

$$p(x) = \int p(x|f_\theta(h))p(h)\ dh,$$

where h is a latent variable, and $f_\theta(h)$, is a complicated mapping which renders the above integral intractable. In [15], the authors suggested using a neural network to model an approximate posterior distribution $q(h|x)$. Recently, there has been a line of papers which adapted this idea into recurrent neural networks [45, 39, 38]. We have therefore included VAE Sequence Models in our summary graph for latent variable models in Figure 1.10.

CHAPTER 2: METHOD OF MOMENTS LEARNING FOR HMMS WITH SPECIAL TRANSITION STRUCTURES

As we talked about in the introduction, maximum likelihood learning of linear latent variable models such as HMMs require local optimization algorithms such as expectation maximization. The line of work such as [17, 18, 46, 19], introduced method of moments based algorithms to learn basic latent variables such as mixture models and HMMs. The applicability of these algorithm is however limited to basic models (HMMs and mixture models), and in this chapter we introduce a framework which extends the applicability of method of moments based methods to HMMs with special transition structure. Let us first start with introducing method of moments learning.

2.1 METHOD OF MOMENTS

Method of moments is a statistical parameter estimation method which dates back to late 19'th century [47]. The idea is to estimate the models parameters θ by solving a system of equations formed with observable moments $\mathbb{E}[g_m(x)]$, $m \in \{1, \dots M\}$:

$$\begin{aligned}\mathbb{E}[g_1(x)] =& f_1(\theta)\\ &\vdots\\ \mathbb{E}[g_M(x)] =& f_M(\theta),\end{aligned}$$

where, $g_{1:M}$ are the set of functions through which the observable moments are calculated, and $f_{1:M}$ denote the set of functions of model parameters yielded by the moment expressions. An easy example for method of moments is obtained by writing the moments of Gamma distribution:

Method of Moments for gamma distribution:
Let $x \sim \mathcal{G}(a, b)$. Let us write the first two moments for x:

$$\begin{aligned}\mathbb{E}[x] =& ab,\\ \mathbb{E}[x^2] =& ab^2 + a^2b^2,\end{aligned}$$

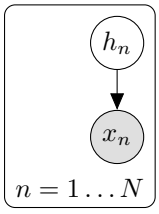
we can then solve for a and b in closed form:

$$\widehat{b} = (\mathbb{E}[x^2] - \mathbb{E}[x]^2)/\mathbb{E}[x]$$
$$\widehat{a} = \mathbb{E}[x]^2/(\mathbb{E}[x^2] - \mathbb{E}[x]^2)$$

Note that closed form solution maximum likelihood estimates for a and b do not exist, and we need to do iterative optimization for it. So, it seems like method of moments provide a convenient alternative to maximum likelihood. Now, note that maximum likelihood is the statistically the most efficient estimator (ML attains the Cramer-Rao lower bound). Although in general, for latent variables models finding the maximum likelihood estimate is NP-Hard [17]. We can however reduce the moment matching problem for several Latent variables to an eigenvalue problem. Let us introduce this with a simple Gaussian mixture model example.

2.1.1 Introduction to Method of Moments for LVMs:

Following [19], we demonstrate the moment matching procedure for a spherical GMM. Let us have the following generative model:



$$h_n \sim Cat(\pi)$$
$$x_n|h_n \sim \mathcal{N}(\mu_h, \sigma^2 I)$$

The moment equations unfold as follows:

$$\mathbb{E}[x \otimes x] = \sum_{k=1}^{K} \pi_k \, \mu_k \otimes \mu_k + \sigma^2 I,$$
$$\mathbb{E}[x \otimes x \otimes x] = \sum_{k=1}^{K} \pi_k \, \mu_k \otimes \mu_k \otimes \mu_k + \sigma^2 \left(\sum_{l=1}^{L} \mathbb{E}[x] \otimes e_l \otimes e_l + e_l \otimes \mathbb{E}[x] \otimes e_l + e_l \otimes e_l \otimes \mathbb{E}[x] \right),$$

where $\otimes$ denote tensor outer product (for vectors a, b, c, $(a \otimes b \otimes c)_{ijk} = a_i b_j c_k$). We can then subtract the terms shown with red from the left hand side of the equation to obtain

the following system of equations:

$$M_2 := \mathbb{E}[x \otimes x] - \text{garbage} = \sum_{k=1}^{K} \pi_k \ \mu_k \otimes \mu_k$$

$$M_3 := \mathbb{E}[x \otimes x \otimes x] - \text{garbage} = \sum_{k=1}^{K} \pi_k \ \mu_k \otimes \mu_k \otimes \mu_k$$

So wee see that we can relate the observable moments to an expression which is constituted of sum of rank-1 tensors. This form is in general known as the CP decomposition form [48]. In [17], it is shown that if the vectors $\mu_{1:K}$ are orthogonal, it is possible to recover them using power iterations (with global convergence guarantee). However, in general the parameter vectors $\mu_{1:K}$ are not orthogonal to each other. So the trick is to obtain a whitening matrix W such that $W^\top M_2 W = I$. This can be done with singular value decomposition (SVD), and we can use it to whiten M_3 also.

As can be seen from this example, the moment matching methods give globally optimal, and computationally cheap alternative to the EM algorithm. However, the moment methods are not as general as EM and therefore only applicable on a small number of models. Examples include mixture models/basic topic models [18, 19, 17], Latent Dirichlet Allocation [46], Hidden Markov Models [49, 18, 17], ICA [50], and PCA. There have been attempts of generalization such as an inference framework for tree graphical models [51] and junction trees [52], and a parameter estimation framework for graphs which satisfies certain conditions [53]. To the best of our knowledge, it was not clear how to do method of moments estimation with the existing methods in the literature.

One contribution in this thesis is to develop a method of moments framework for structured HMMs. Namely, we show that mixture of HMMs [31], switching HMMs [32], left-to right HMMs [27], and HMMs with mixture emissions [32] can be learnt by a method of moments based framework. Now let us write down the moments for the mixture HMMs to show why a straight-forward application of moment matching does not work.

2.2 METHOD OF MOMENTS FOR STRUCTURED HMMS

Let us write down the second order moment of a mixture of HMMs model (introduced in Section 1.2) to see how and why the naive approach fails:

$$\begin{aligned}\mathbb{E}[x_2 \otimes x_1] =& \sum_{h,r_1} \rho_h \pi_h \left(\mathbb{E}[x_2|r_1,h] \otimes \mathbb{E}[x_1|r_1,h]\right) \\ =& \sum_{h,r_1} \rho_h \pi_h \left(\sum_{r_2} A(r_1,r_2,h)\mu_{r_2,h}\right) \otimes \mu_{r_1,h} \\ =& O_{flat} A_{bdiag} \mathrm{diag}(\rho \otimes \pi) O_{flat}^\top\end{aligned}$$

The problem is that the moment estimator is agnostic to the block structure of the model: If we simply compute the moments, and input them into some eigenvalue solver, we would be ignoring the group/block structure of the model. We therefore need to somehow retain/enforce the group structure. But before doing that let us first formalize what we mean by the term "group structure".

Theorem 2.1. *An MHMM with local parameters $\theta_{1:K} = (O_{1:K}, A_{1:K}, \nu_{1:K}, \pi)$ is an HMM with global parameters $\bar{\theta} = (\bar{O}, \bar{A}, \bar{\nu})$, where*

$$\bar{O} = \begin{bmatrix} O_1 & \dots & O_K \end{bmatrix}, \quad \bar{A} = \begin{bmatrix} A_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & A_2 & \dots & \mathbf{0} \\ & & \ddots & \\ \mathbf{0} & \mathbf{0} & \dots & A_K \end{bmatrix}, \quad \bar{\nu} = \begin{bmatrix} \pi_1\nu_1 \\ \pi_2\nu_2 \\ \vdots \\ \pi_K\nu_K \end{bmatrix},$$

where $\bar{O}$ is the emission matrix, $\bar{A}$ is the transition matrix and $\bar{\nu}$ is the initial state distribution.

Proof of Theorem 2.1. Consider the MHMM likelihood for a sequence $\mathbf{x}_n$:

$$\begin{aligned}p(\mathbf{x}_n|\theta_{1:K}) =& \sum_{k=1}^K \pi_k \left\{ 1_M^\top \left(\prod_{t=1}^{T_n} A_k \,\mathrm{diag}(O_k(x_t)) \right) \nu_k \right\} \\ =& 1_{MK}^\top \left(\prod_{t=1}^{T_n} \begin{bmatrix} A_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & A_2 & \dots & \mathbf{0} \\ & & \ddots & \\ \mathbf{0} & \mathbf{0} & \dots & A_K \end{bmatrix} \mathrm{diag}(\begin{bmatrix} O_1 & \dots & O_K \end{bmatrix}(x_t)) \right) \begin{bmatrix} \pi_1\nu_1 \\ \pi_2\nu_2 \\ \vdots \\ \pi_K\nu_K \end{bmatrix} \\ =& 1_{MK}^\top \left(\prod_{t=1}^{T_n} \bar{A} \,\mathrm{diag}(\bar{O}(x_t)) \right) \bar{\nu}, \end{aligned} \tag{2.1}$$

where $\begin{bmatrix} O_1 & \dots & O_K \end{bmatrix}(x_t) := \bar{O}(x_t)$. We conclude that the MHMM and an HMM with parameters $\bar{\theta}$ describe equivalent probabilistic models. □

This theorem tells us that mixture of HMMs is in fact an HMM with a block diagonal transition matrix. The implication is that, if can somehow enforce the block diagonal structure on the learnt transition matrix, we can apply method of moments. Having this goal in mind, let us introduce the two stage estimation framework.

2.2.1 Two stage estimation framework for Structured HMMs:

This framework is mainly based on the observation that an HMM is a mixture model. The precise statement is as follows:

Theorem 2.2. *An HMM with state marginals $p(h_t)$ is equivalent to a mixture model with mixing weights $\pi := \frac{1}{T}\sum_{t=1}^{T} p(h_t)$, and the same emission parameters [54].*

This theorem implies one can separate the learning of the emission and transition matrices. Namely, we can simply fit a mixture model to get the emission matrix. Now, note that the second order moment for an HMM is given as follows:

$$\mathbb{E}[x_2 \otimes x_1] = OA\text{diag}(\pi)O, \tag{2.2}$$

where O is the emission matrix and A is the transition matrix. The usefulness of getting the emission matrix and the mixing proportions π is that, we can plug these quantities in Equation (2.2), and get an estimate for A with a convex program.

While doing so we can also impose structural constraints with affine constraints while not sacrificing from convexity if we can somehow undo permutation ambiguity that is borne out of the eigenvalue algorithm used in the estimation of the emission matrix. The overall two stage estimation framework is summarized in Figure 2.1.

- Get rough/permuted estimates for the parameters $\widehat{O}, \widehat{A}, \widehat{\pi}$, using method of moments.
- De-permute $\widehat{A}$. (Solve the graph problem dictated by model)
- Solve: (Refinement step)

$$\begin{aligned} \min_{A} \; & \|\widehat{M}_2 - \widehat{O} A \mathrm{diag}(\widehat{\pi}) \widehat{O}\|_F \\ s.t. \; & 1^\top A = 1^\top, \\ & A \geq 0, \\ & f(\mathcal{M}, A) = 0, \end{aligned}$$

  where $\widehat{M}_2$ denotes an empirical estimate for $\mathbb{E}[x_2 \otimes x_1]$.
- f, and $\mathcal{M}$ depend on the model.

Figure 2.1: The two stage estimation framework for method of moments learning of HMMs with special transition structures.

So, for a given structured HMM, if we can figure out a way to de-permute the initial estimate for A, we can derive a method of moments based parameter estimation algorithm. Once the parameter de-permutation is done, the estimate for A can be refined by imposing an affine constraint of the form $f(\mathcal{M}, A) = 0$, where $\mathcal{M}$ is a binary mask used to encode the structure of the transition matrix A. Here are the models that fit to our framework, along with the description of their masks $\mathcal{M}$:

- **MHMM**: $f(\mathcal{M}, A) = A \odot (1 - \mathcal{M}) = \mathbf{0}$. $\mathcal{M}$ is block diagonal.
- **SHMM**: $f(\mathcal{M}, A) = A \odot (1 - \mathcal{M}) - \widehat{B} \otimes \frac{1}{M} \mathbf{1}_M \mathbf{1}_M^\top = \mathbf{0}$. $\mathcal{M}$ is block diagonal.
- **Left-to-Right HMM**: $f(\mathcal{M}, A) = A \odot (1 - \mathcal{M}) = 0$, estimate $\mathcal{M}$ with a greedy graph traversal algorithm. $\mathcal{M}$ is lower triangular.
- **Bakis HMM**: $f(\mathcal{M}, A) = A \odot (1 - \mathcal{M}) = 0$, $\mathcal{M}$ corresponds to an Hamiltonian circuit (TSP approximation). $\mathcal{M}$ is binary lower first uni-triangular.
- **HMM with mixture emissions**: $f(\mathcal{M}, A^{i,j}) = A^{i,j} \mathbf{1}^\top$.

Now let us dwell on how to do the de-permutation for individual models, along with references to our corresponding publications.

2.2.2 Mixture of HMMs

In [31], we showed that we can use the fact that a block diagonal transition matrix with K blocks has K stationary distributions. We can therefore use a spectral clustering [55] type algorithm to de-permute the global transition matrix. Now, let us formalize this statement.

Lemma 2.1. *Assuming that each of the local transition matrices $A_{1:K}$ has only one eigenvalue which is 1, the global transition matrix $\bar{A}$ has K eigenvalues which are 1.*

Proof for Lemma 2.1.

$$\bar{A} = \begin{bmatrix} V_1\Lambda_1V_1^{-1} & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & V_K\Lambda_KV_K^{-1} \end{bmatrix} = \underbrace{\begin{bmatrix} V_1 & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & V_K \end{bmatrix} \begin{bmatrix} \Lambda_1 & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \Lambda_K \end{bmatrix} \begin{bmatrix} V_1 & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & V_K \end{bmatrix}^{-1}}_{\bar{V}\bar{\Lambda}\bar{V}^{-1}},$$

where $V_k\Lambda_kV_k^{-1}$ is the eigenvalue decomposition of A_k with V_k as eigenvectors, and Λ_k as a diagonal matrix with eigenvalues on the diagonal. The eigenvalues of $A_{1:K}$ appear unaltered in the eigenvalue decomposition of $\bar{A}$, and consequently $\bar{A}$ has K eigenvalues which are 1. □

Now that we have established that for a non-pathological local transition matrices for Kcomponent mixture of HMMs, we have K non-zero eigenvalues. This implies that if we exponentiate this global transition matrix to the infinity, we get a transition matrix K distinct columns:

Corrolary 2.1.

$$\lim_{e\to\infty} \bar{A}^e = \begin{bmatrix} \bar{v}_1 1_M^\top & \dots & \bar{v}_k 1_M^\top & \dots & \bar{v}_K 1_M^\top \end{bmatrix}, \tag{2.3}$$

where $\bar{v}_k = [0^\top \ \dots \ v_k^\top \ \dots \ 0^\top]^\top$ and v_k is the stationary distribution of A_k, $\forall k \in \{1, \dots, K\}$.

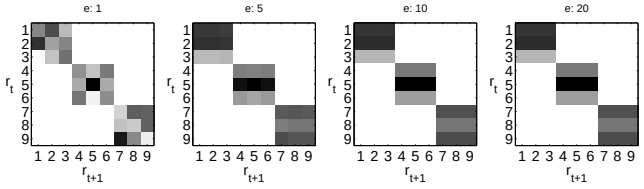
Proof for Corrolary 2.1.

$$\lim_{e\to\infty} (V_k\Lambda_kV_k^{-1})^e = \lim_{e\to\infty} V_k\Lambda_k^eV_k^{-1} = V_k \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & 0 \end{bmatrix} V_k^{-1} = v_k 1_M^\top.$$

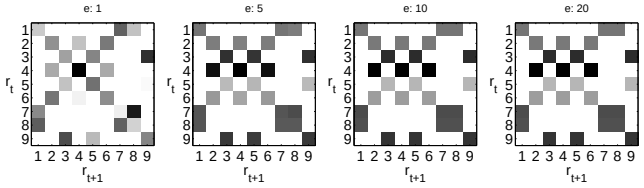
The third step follows because there is only one eigenvalue with magnitude 1. Since multiplying $\bar{A}$ by itself amounts to multiplying the corresponding diagonal blocks, we have the structure in equation (2.3). □

We therefore conclude that given a permuted global transition matrix $\mathcal{P}(\bar{A})$ (where $\mathcal{P}$ is the permutation mapping that is caused by the learning algorithm), we can cluster the columns of $\mathcal{P}(\bar{A})^{\infty}$, and group the columns of $\mathcal{P}(\bar{A})$ accordingly. This is obviously doable in the case where there is no noise on the global transition matrix. However in real life the estimate for the global transition matrix is noisy in addition to being permuted. This is illustrated in Figure 2.2.

- What happens when we exponentiate $\bar{A}$:



- What happens when we exponentiate $\mathcal{P}(\bar{A})$:



- What happens in practice:

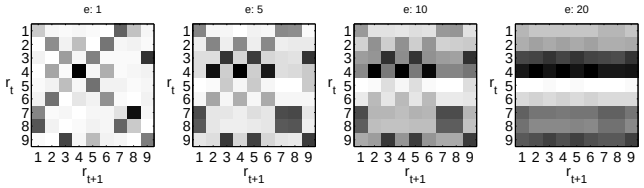


Figure 2.2: Illustration of the effect of exponentiating the global MHMM transition matrix.

Because the estimated transition matrix is not exactly block diagonal, in practice there is only one eigenvalue which is 1, and therefore the exponentiated matrix converges to a stationary distribution. However, if we can somehow find the number of HMMs, we can make a low-rank reconstruction, and cluster the columns of the reconstruction. Namely:

- If the eigenvalue structure is not too much corrupted (the precise condition is specified below), we see a plateau in the number of significant eigenvalues across exponentiations of $(\bar{A})^{e}$. The largest plateau corresponds to the number of HMM components.

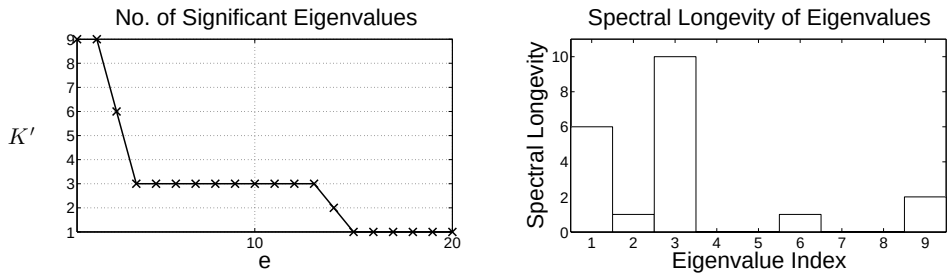


Figure 2.3: Estimating number of clusters via spectral longevity.

- Then we can calculate the rank-$\widehat{K}$ reconstruction A^r:

$$A^r = V_{1:\widehat{K}} \Lambda_{1:\widehat{K}} V^{-1}$$

- We can then cluster the columns of the low rank reconstruction.

The precise condition on the corruption of the eigenvalue structure is given below.

Definition 2.1. *We denote $\lambda_k^{\mathcal{G}} := \alpha_k \lambda_{1,k}$ for $k \in \{1, \ldots, K\}$ as the global, noisy eigenvalues with $|\lambda_k^{\mathcal{G}}| \geq |\lambda_{k+1}^{\mathcal{G}}|$, $\forall k \in \{1, \ldots, K-1\}$, where $\lambda_{1,k}$ is the original eigenvalue of the k^{th} cluster with magnitude 1 and α_k is the noise that acts on that eigenvalue (note that $\alpha_1 = 1$). We denote $\lambda_{j,k}^{\mathcal{L}} := \beta_{j,k} \lambda_{j,k}$ for $j \in \{2, \ldots, M\}$ and $k \in \{1, \ldots, K\}$ as the local, noisy eigenvalues with $|\lambda_{j,k}^{\mathcal{L}}| \geq |\lambda_{j+1,k}^{\mathcal{L}}|$, $\forall k \in \{1, \ldots, K\}$ and $\forall j \in \{1, \ldots, M-1\}$, where $\lambda_{j,k}$ is the original eigenvalue with the j^{th} largest magnitude in the k^{th} cluster, and $\beta_{j,k}$ is the noise that acts on that eigenvalue.*

Definition 2.2. *The low-rank eigendecomposition of the estimated transition matrix $\bar{A}_\epsilon^{\mathcal{P}}$ is defined as $A_\epsilon^r := V \Lambda^r V^{-1}$, where V is a matrix with eigenvectors in the columns and Λ^r is a diagonal matrix with eigenvalues $\lambda_{1:K}^{\mathcal{G}}$ in the first K entries.*

Conjecture 2.1. *If $|\lambda_K^{\mathcal{G}}| > \max_{k \in \{1,\ldots,K\}} |\lambda_{2,k}^{\mathcal{L}}|$, then A^r can be formed using the eigen-decomposition of $\bar{A}_\epsilon^{\mathcal{P}}$. Then, with high probability, $\|A_\epsilon^r - A^r\|_F \leq \mathcal{O}(1/\sqrt{TN})$, where TN is the total number of observed vectors.*

Justification for Conjecture 2.1.

$$\begin{aligned}
\|A_\epsilon^r - A^r\|_F = \|A_\epsilon^r - A + A - A^r\|_F \leq & \|A_\epsilon^r - A\|_F + \|A - A^r\|_F \\
= & \|A - A^r\|_F + \|A - A_\epsilon + A_\epsilon^{\bar{r}}\|_F \\
\leq & \|A - A^r\|_F + \|A_\epsilon^{\bar{r}}\|_F + \|A - A_\epsilon\|_F \\
\leq & 2KM + \mathcal{O}(1/\sqrt{TN}) = \mathcal{O}(1/\sqrt{TN}), \quad w.h.p.,
\end{aligned}$$

where A is used for $\bar{A}^{\mathcal{P}}$ to reduce the notation clutter (and similarly A^r for $(\bar{A}^{\mathcal{P}})^r$ and so on), we used the triangle inequality for the first and second inequalities and $A_\epsilon^{\bar{r}} = V\Lambda^{\bar{r}}V^{-1}$, where $\Lambda^{\bar{r}}$ is a diagonal matrix of eigenvalues with the first K diagonal entries equal to zero (complement of Λ^r). For the last inequality, we used the fact that $A \in \mathbb{R}^{MK\times MK}$ has entries in the interval $[0,1]$ and we used the sample complexity result from [18]. The bound specified in [18] is for a mixture model, but since the two models are similar and the estimation procedure is almost identical, we are reusing it. □

Conjecture 1 asserts that, if we have enough data we should obtain an estimate A_ϵ^r close to A^r in the squared error sense. Furthermore, if the following mixing rate condition is satisfied, we will be able to identify the number of clusters K from the data.

Definition 2.3. *Let $\widetilde{\lambda}_k$ denote the k^{th} largest eigenvalue (in decreasing order) of the estimated transition matrix $\bar{A}_\epsilon^{\mathcal{P}}$. We define the quantity,*

$$\mathfrak{L}_{\tilde{\lambda}_{K'}} := \sum_{e=1}^{\infty}\left(\left[\frac{\sum_{l=1}^{K'}|\tilde{\lambda}_l|^e}{\sum_{l'=1}^{MK}|\tilde{\lambda}_{l'}|^e} > 1-\gamma\right] - \left[\frac{\sum_{l=1}^{K'-1}|\tilde{\lambda}_l|^e}{\sum_{l'=1}^{MK}|\tilde{\lambda}_{l'}|^e} > 1-\gamma\right]\right), \tag{2.4}$$

as the spectral longevity of $\tilde{\lambda}_{K'}$. The square brackets [.] denote an indicator function which outputs 1 if the argument is true and 0 otherwise, and γ is a small number such as machine epsilon.

Lemma 2.2. *If $|\lambda_K^{\mathcal{G}}| > \max_{k\in\{1,\ldots,K\}}|\lambda_{2,k}^{\mathcal{L}}|$ and $\arg\max_{K'}\frac{|\widetilde{\lambda}_{K'}|^2}{|\widetilde{\lambda}_{K'+1}||\widetilde{\lambda}_{K'-1}|} = K$, for $K' \in \{2,3,\ldots,MK-1\}$, then $\arg\max_{K'}\mathfrak{L}_{\tilde{\lambda}_{K'}} = K$.*

Proof for Lemma 2.2. The first condition ensures that the top K eigenvalues are global eigenvalues. The second condition is about the convergence rates of the two ratios in equation (2.4). The first indicator function has the following summation inside:

$$\frac{\sum_{l=1}^{K'}|\tilde{\lambda}_l|^e}{\sum_{l'=1}^{MK}|\tilde{\lambda}_{l'}|^e} = \frac{\sum_{l=1}^{K'-1}|\tilde{\lambda}_l|^e + |\tilde{\lambda}_{K'}|^e}{\sum_{l'=1}^{K'-1}|\tilde{\lambda}_{l'}|^e + |\tilde{\lambda}_{K'}|^e + |\tilde{\lambda}_{K'+1}|^e + \sum_{l'=K'+2}^{MK}|\tilde{\lambda}_{l'}|^e}\ .$$

The rate at which this term goes to 1 is determined by the spectral gap $|\lambda_{K'}|/|\lambda_{K'+1}|$. The smaller this ratio is, the faster the term (it is non-decreasing w.r.t. e) converges to 1. For the second indicator function inside $\mathfrak{L}_{\tilde{\lambda}_{K'}}$, we can do the same analysis and see that the convergence rate is again determined by the gap $|\lambda_{K'-1}|/|\lambda_{K'}|$. The ratio of the two spectral gaps determines the spectral longevity. Hence, for the K' with largest ratio $\frac{|\widetilde{\lambda}_{K'}|^2}{|\widetilde{\lambda}_{K'+1}||\widetilde{\lambda}_{K'-1}|}$, we have $\arg\max_{K'}\mathfrak{L}_{\tilde{\lambda}_{K'}} = K$. □

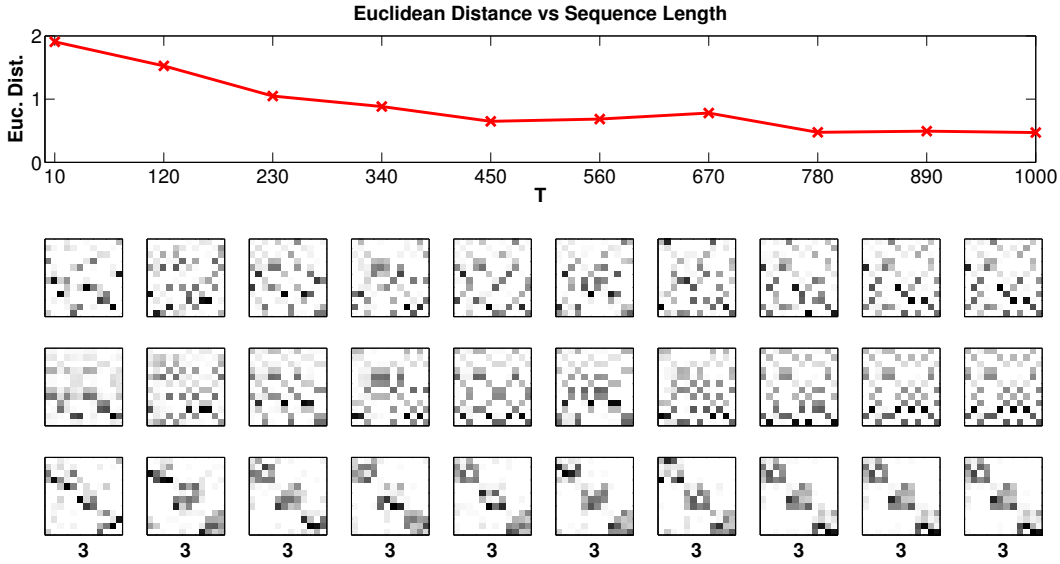


Figure 2.4: Top row: Euclidean distance vs T. Second row: Noisy input matrix. Third row: Noisy reconstruction A_ϵ^r. Bottom row: Depermuted matrix, numbers at the bottom indicate the estimated number of clusters.

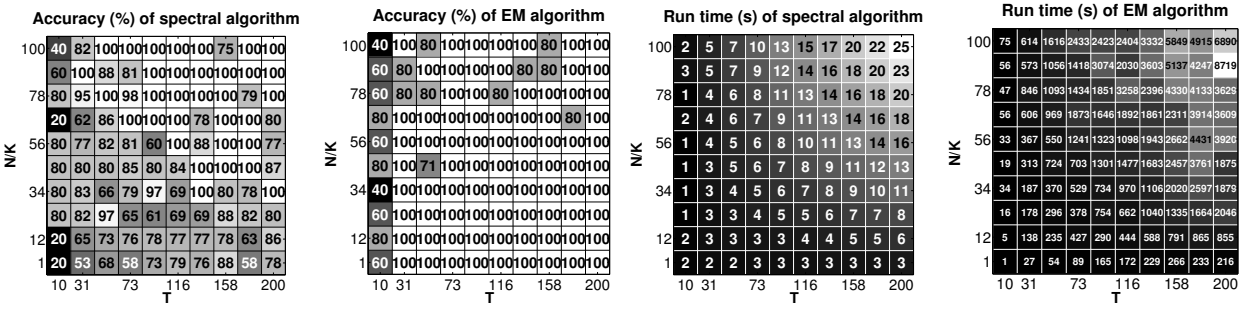


Figure 2.5: Clustering accuracy and run time results for synthetic data experiments.

Lemma 3 tells us the following. If the estimated transition matrix $\bar{A}_\epsilon^{\mathcal{P}}$ is not *too noisy*, we can determine the number of clusters by choosing the value of K' such that it maximizes $\mathfrak{L}_{\tilde{\lambda}_{K'}}$. This corresponds to exponentiating the sorted eigenvalues in a finite range, and recording the number of non-negligible eigenvalues. This is depicted in Figure 2.3.

Experiments

Effect of noise on depermutation algorithm:

We have tested the algorithm's performance with respect to amount of data. We used the parameters $K = 3$, $M = 4$, $L = 20$, and we have 2 sequences with length T for each cluster.

We used a Gaussian observation model with unit observation variance and the columns of the emission matrices $O_{1:K}$ were drawn from zero mean spherical Gaussian with variance 2. Results for 10 uniformly spaced sequence lengths from 10 to 1000 are shown in Figure 2.4. On the top row, we plot the total error (from centroid to point) obtained after fitting k-means with true number of HMM clusters. We can see that the correct number of clusters $K = 3$ as well as the block-diagonal structure of the transition matrix is correctly recovered even in the case where $T = 20$.

Amount of data vs accuracy and speed:

We have compared clustering accuracies of EM and our approach on data sampled from a Gaussian emission MHMM. Means of each state of each cluster is drawn from a zero mean unit variance Gaussian, and observation covariance is spherical with variance 2. We set $L = 20$, $K = 5$, $M = 3$. We used uniform mixing proportions and uniform initial state distribution. We evaluated the clustering accuracies for 10 uniformly spaced sequence lengths (every sequence has the same length) between 20 and 200, and 10 uniformly spaced number of sequences between 1 and 100 for each cluster. The results are shown in Figure 2.5. Although EM seems to provide higher accuracy on regions where we have less data, spectral algorithm is much faster. Note that, in spectral algorithm we include the time spent in moment computation. We used four restarts for EM, and take the result with highest likelihood, and used an automatic stopping criterion.

Real data experiment:

We ran an experiment on the handwritten character trajectory dataset from the UCI machine learning repository [56]. We formed pairs of characters and compared the clustering results for three algorithms: the proposed spectral learning approach, EM initialized at random, and EM initialized with MoM algorithm as explored in [57]. We take the maximum accuracy of EM over 5 random initializations in the third row. We set the algorithm parameters to $K = 2$ and $M = 4$. There are 140 sequences of average length 100 per class. In the original data, $L = 3$, but to apply MoM learning, we require that $MK < L$. To achieve this, we transformed the data vectors with a cubic polynomial feature transformation such that $L = 10$ (this is the same transformation that corresponds to a polynomial kernel). The results from these trials are shown in Table 2.1. We can see that although spectral learning doesn't always surpass randomly initialized EM on its own, it does serve as a very good initialization scheme.

Table 2.1: Clustering accuracies for handwritten digit dataset.

Algorithm	1v2	1v3	1v4	2v3	2v4	2v5
Spectral	100	70	54	83	99	99
EM init. w/ Spectral	100	99	100	96	100	100
EM init. at Random	96	99	98	83	100	100

2.2.3 Switching HMMs and HMMs with mixture emissions:

As introduced in Section 1.2, this model is intimately related to the MHMM model above. Namely, SHMM can also be expressed as a larger HMM with a transition which has a special structure. The difference from MHMM is there is non-zero probability mass on the off-diagonal of the global transition matrix. The formal definition is given below:

Theorem 2.3. *An SHMM with local parameters $\theta_{1:K} = (O_{1:K}, A_{1:K}, \nu_{1:K}, B)$ is an HMM with global parameters $\bar{\theta} = (\bar{O}, \bar{A}, \bar{\nu})$, where:*

$$\bar{O} = \begin{bmatrix} O_1 & \dots & O_K \end{bmatrix}, \bar{A} = \begin{bmatrix} B_{1,1}A_1 & B_{1,2}\frac{\mathbf{1}\mathbf{1}^\top}{M} & \dots & B_{1,M}\frac{\mathbf{1}\mathbf{1}^\top}{M} \\ B_{2,1}\frac{\mathbf{1}\mathbf{1}^\top}{M} & B_{2,2}A_2 & \dots & B_{2,M}\frac{\mathbf{1}\mathbf{1}^\top}{M} \\ & & \ddots & \\ B_{M,1}\frac{\mathbf{1}\mathbf{1}^\top}{M} & B_{M,2}\frac{\mathbf{1}\mathbf{1}^\top}{M} & \dots & B_{M,M}A_K \end{bmatrix},$$

$$\bar{\nu} = \begin{bmatrix} \pi_1\nu_1 & \pi_2\nu_2 & \dots & \pi_K\nu_K \end{bmatrix}^\top.$$

Proof Sketch for Theorem 2.3. The full joint distribution for SHMM is defined as follows:

$$\begin{aligned} & p(x_{1:T}, r_{1:T}, h_{1:T}) \\ =& p(r_1|h_1)p(h_1)\prod_{t=1}^{T} p(x_t|r_t, h_t)p(r_t|r_{t-1}, h_{t-1})p(h_t|h_{t-1}), \\ =& p(r_1, h_1)\prod_{t=2}^{T} p(x_t|r_t, h_t)p(r_t, h_t|r_{t-1}, h_{t-1}). \end{aligned}$$

At this point, we see that this expression is same as the HMM joint distribution if we define a new variable $rh_t : (r_t \otimes h_t)$, which is defined on the product space of r_t and h_t. Therefore, SHMM is equivalent to an HMM with MK states, where the transition matrix is given by $\bar{A}$. □

Similarly, an HMM with mixture emissions (HMM-M) can be seen as a larger HMM with a particular global transition structure.

Theorem 2.4. *An HMM-M with local parameters $\theta = (O_{1:K}, B, \nu_{1:K})$ is the k'th emission matrix, B is the global regime transition matrix, and ν_k is the mixture proportions for the k'th state, is an HMM with global parameters $\bar{\theta} = (\bar{O}, \bar{A})$, where:*

$$\bar{O} = \begin{bmatrix} O_1 & \dots & O_K \end{bmatrix}, \bar{A} = \begin{bmatrix} B_{1,1}\,\nu_1 \mathbf{1}_M^\top & B_{1,2}\,\nu_1 \mathbf{1}_M^\top & \dots & B_{1,K}\,\nu_1 \mathbf{1}_M^\top \\ B_{2,1}\,\nu_2 \mathbf{1}_M^\top & B_{2,2}\,\nu_2 \mathbf{1}_M^\top & \dots & B_{2,K}\,\nu_2 \mathbf{1}_M^\top \\ & & \ddots & \\ B_{K,1}\,\nu_K \mathbf{1}_M^\top & B_{K,2}\,\nu_K \mathbf{1}_M^\top & \dots & B_{K,K}\,\nu_K \mathbf{1}_M^\top \end{bmatrix}.$$

Proof Sketch for Theorem 2.4. The full joint distribution for HMM-M is defined as follows:

$$\begin{aligned} p(x_{1:T}, r_{1:T}, h_{1:T}) =& p(r_1|h_1)p(h_1)\prod_{t=1}^{T} p(x_t|r_t, h_t)p(r_t|h_t)p(h_t|h_{t-1}), \\ =& p(r_1, h_1)\prod_{t=2}^{T} p(x_t|r_t, h_t)p(r_t, h_t|h_{t-1}), \\ =& p(r_1, h_1)\prod_{t=2}^{T} p(x_t|r_t, h_t)p(r_t, h_t|r_{t-1}, h_{t-1}). \end{aligned}$$

We again see that the full joint distribution of HMM-M is equal to an HMM full-joint distribution where the state variable is defined on the product space of r_t and h_t, and the transition matrix is given by $\bar{A}$. The last equality is due to conditional independence structure of HMM-M model. □

Because MHMM, SHMM and HMM-M are closely related (e.g. as shown above, SHMM is an MHMM with a non-zero transition probability between the HMM groups), we can use the same depermutation procedure that we have described for MHMMs above. If the global mixing is not too strong (i.e. if the global HMM have the tendency to make transitions within a group) then it is possible to depermute the global transition matrix. We demonstrate this with a synthetic data experiment in Figure 2.6 for SHMM and in Figure 2.7 for HMM-M, where we change the probability of transitioning between groups. Namely, we synthesize sequences from an SHMM model by using "regime transition" matrices respecting the formula,

$$B = \begin{bmatrix} \alpha & 1-\alpha \\ 1-\alpha & \alpha \end{bmatrix},$$

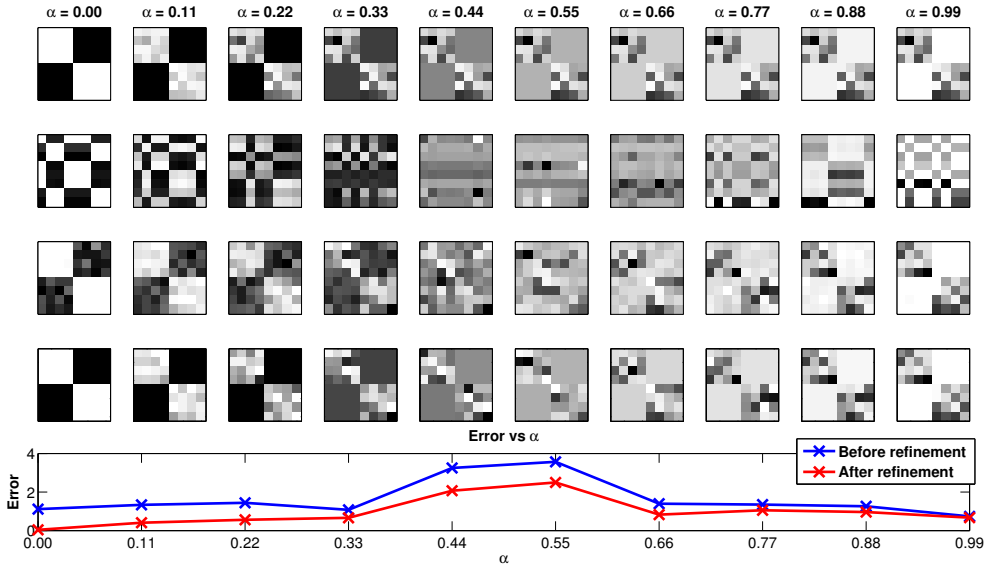


Figure 2.6: Switching HMM experiment: We synthetically generate sequences where we parametrically vary the probability with which we stay in an HMM component. The rightmost figure corresponds to the case where $\alpha = 0.99$, which means the model stays in the same HMM group almost certainly. (Top row) The original transition matrix. (Second row) Estimated permuted transition matrix. (Third row) Depermuted estimated transition matrix before refinement step. (Fourth row) Estimated transition after refinement step. (Last row) The l1 error between the estimated transition matrix and the ground truth before and after the refinement step.

that is, larger α is, the more lenient the model is on making transitions within a group (and hence the property "group persistence" in [32]. We vary α in the range $[0, 1]$, and see that we can recover the original transition matrix well when α is close to 0 or 1.

2.2.4 Left to Right HMMs

As we have discussed in Section 1.2.1, a left-to right HMM is an HMM where state transitions can only increase the state index. The challenge in applying a method-of moments algorithm is again the permutation ambiguity introduced by the learning algorithm. In [27], we propose using a greedy algorithm in the general left-to-right case. We also look at the case where the state transition can only increase the state index by one. This case is known as the Bakis HMM, and the depermutation amounts to solving the traveling salesman problem on a weighted graph. Our synthetic data experiments suggest that initializing an EM algorithm with our proposed framework provides a boost in running time. This is demonstrated in Figures 2.8, 2.9. As we can see from the Figures, as the dataset size increases, the benefits of

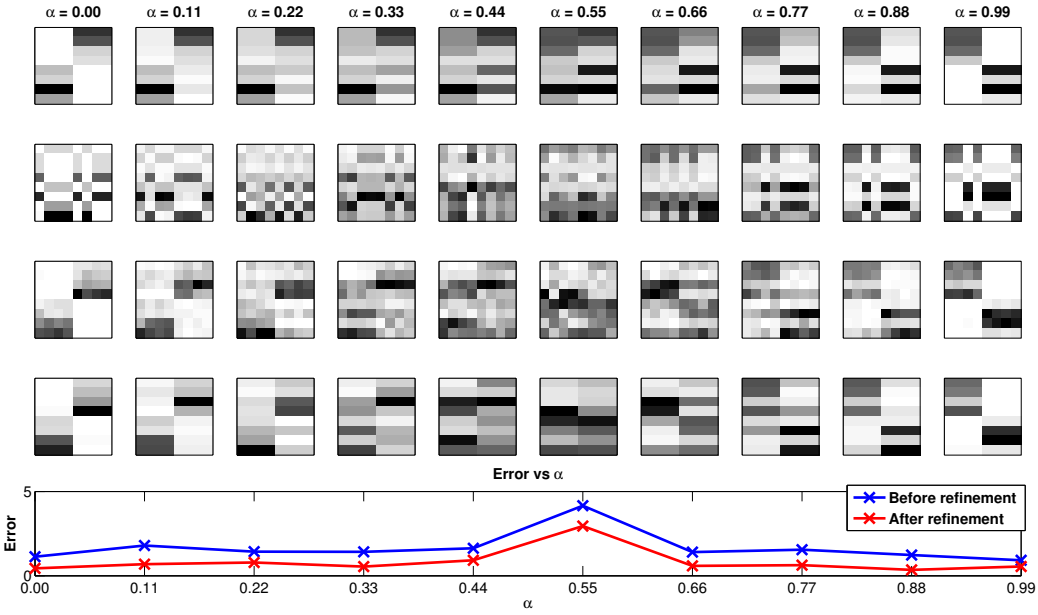


Figure 2.7: HMM-M experiment: The same caption in Figure 2.6 applies here.

using a method of moments initializer gets more and more apparent in the form of a speed boost. Therefore using our algorithm potentially helps an EM algorithm to be more scalable in larger datasets.

We now discuss how to recover the left-to-right ordering of the states which is needed to enforce the lower triangular structure in $\widehat{A}$.

Let $\mathcal{P}$: $[M] \to [M]$ denote the permutation (where $[M] = \{1, \dots M\}$) corresponding to the permutation matrix P from the first step of the general algorithm defined in Section 2.2. Once again note that $\mathcal{M} = P\mathcal{S}P^\top$ denotes the permuted binary mask that corresponds to transition matrix structure (e.g., $\mathcal{M} = P\mathcal{S}_{LR}P^\top$ in the general LR-HMM case). We give an algorithm to estimate $\mathcal{M}$ for both the general LR-HMM and the more constrained Bakis HMM.

Depermutation for a general LR-HMM

In the case of general LR-HMM, the transition matrix is lower triangular in its original form. This means that for each $i \in [M]$, the i'th row in A has the fewest non-zero entries among all rows $i' \geq i$ after excluding the diagonal entries of A and columns $j < i$. Thus, a simple greedy algorithm recovers the permutation $\mathcal{P}$ from a sufficiently accurate estimate $\widehat{A}$ of the transition matrix A (Algorithm 2.1).

Algorithm 2.1 The Depermutation Algorithm for General LR-HMM

Input: Transition matrix $\widehat{A}$, threshold $\gamma \geq 0$ (default: $\gamma = 0$).
Output: Binary mask $\widehat{\mathcal{M}}$.
Initialize $\widetilde{\mathcal{M}}_{i,j} := \mathbf{1}(\widehat{A}_{i,j} > \gamma)$ for all $(i,j) \in [M]^2$; $v := [M]$.
for $i = 1, 2, \ldots, M$ **do**
    $\mathcal{P}(i) := \text{argmin}_{i' \in v} \sum_{j \in v \setminus \{i'\}} \widetilde{\mathcal{M}}_{i',j}$.
    $v := v \setminus \{\mathcal{P}(i)\}$.
end for
return $\widehat{\mathcal{M}} = P\mathcal{S}_{LR}P^\top$, where P is the permutation matrix corresponding to $\mathcal{P}$.

Algorithm 2.1 takes a threshold parameter γ as input, which in practice can be tuned using cross-validation (e.g., try several values for γ and choose the result yielding the model with highest held-out likelihood). In the next section, we describe an algorithm specifically tailored for Bakis HMMs that avoids this extra parameter.

Learning the refinement mask for Bakis HMM

The state transition structure of a Bakis HMM defines an Hamiltonian path (a tour along the vertices, with each vertex visited exactly once) in the state space. Finding an Hamiltonian path is known to be NP-Hard [58]. We therefore propose a greedy algorithm in Algorithm 2.2.

This algorithm finds Hamiltonian paths starting from every state, and then picks the one yielding the highest likelihood. If the number of sequences is small, then it is possible to use a regularizer $R(.)$ for the choice of k'. For example in a musical chord segmentation experiment, one can choose the transition matrix which yields the Viterbi decoding with most uniform-length segments, since a-priori we know that chords are played for similar durations. Next, we show that if the estimated transition matrix is close to the true transition matrix, the algorithm returns the correct answer, and consequently as the number of observed sequences tends to infinity, the Algorithm is guaranteed to return the true parameters up to a permutation of the state indices.

Definition 2.4. *Let $\epsilon := \|\widehat{A} - PAP^\top\|_1$, where $\|.\|_1$ computes the sum of absolute values of the entries of the argument.*

Lemma 2.3. *If $\epsilon \leq \min_j \max_{i \neq j} A_{i,j}$, then the output of Algorithm 2.2 satisfies $(1_M 1_M^\top - \mathcal{M}) \odot \widehat{A} = \mathbf{0}$.*

Proof of Lemma 2.3. The condition requires that the deviation ϵ should be smaller than the

Algorithm 2.2 The greedy algorithm for Bakis HMM

Input: Noisy and Permuted Transition Matrix $\widehat{A}$.
Output: Binary Mask $\mathcal{M}$.

for $k = 1 : M$ **do**
  $i = 1$; $j = k$; vsts $= \{j\}$;
  while $i < M$ **do**
    $j' = \arg\max_{l \in \{1,\ldots,M\} \setminus \text{vsts}} A_{l,j}$;
    vsts $= \{\text{vsts}, j'\}$; ▷ Add j' to the list of visited vertices.
    $\text{Masks}(j', j, k) = 1$;
    $j = j'$; ▷ Next state to visit is j'.
    $i = i + 1$;
  end while
  $\text{Masks}(k, j', k) = 1$; ▷ Optional step to complete the cycle.
  $A'(:, :, k) = \text{Normalize}(A \odot (\text{Masks}(:, :, k) + I))$;
  ▷ Normalize A according to the estimated mask
end for
$k' = \arg\max_{l \in \{1,\ldots,M\}} \log p(x_{1:T}|A'(:, :, l)) - \lambda R(A'(:, :, l))$; ▷ Pick the mask with largest regularized log-likelihood.
return $\mathcal{M} = \text{Masks}(:, :, k') + I$;

smallest of second largest column entries in A. If this is satisfied, then then the algorithm will find the true Hamiltonian path since the true path will remain unaltered in $\widehat{A}$. □

Theorem 2.5. *As the number of observed sequences $N \to \infty$, Algorithm 2.2 is guaranteed to find the true mask $\mathcal{M}$.*

Proof Sketch for Theorem 2.5. As $N \to \infty$, the estimates of the tensor power method converges to the true emission matrix O and the mixing weights π. Furthermore, due to law of large numbers the empirical moment converges to the true moment: $\widehat{M_2} \to M_2$. When this is the case, one can show that a pseudo-inverse estimator $\widehat{O}^\dagger \widehat{M_2} (\widehat{O}^\top)^\dagger \text{diag}(\widehat{\pi})^{-1}$ converges to A. Since $\text{argmin}_{A'} \|M_2 - OA'\text{diag}(\pi)O^\top\|_F$ is in the feasible region, the solution of the optimization problem in Section 2.2 is equal to this pseudo inverse estimator, and therefore $\epsilon \to 0$. This results in the condition in Lemma 3 being satisfied, and therefore Algorithm 2.2 is guaranteed to return the true mask $\mathcal{M}$. □

Experiments

Synthetic Data Experiment:

We experimentally studied the time-accuracy tradeoff for the proposed algorithm (MoM),

expectation maximization (EM), and EM initialized by MoM for various number of EM iterations. For sequence lengths 400, 4000, and 40000 we generated 10 sequences for two classes from Bakis HMM, with 4 hidden states and Gaussian emission model. The means of the Gaussians were drawn from a zero mean unit variance Gaussian. The observation model was a Gaussian with variance 8. We learned Bakis HMMs from these sequences. We then did Viterbi decoding on 10 test sequences generated from the same Bakis HMMs used in training. For EM learning we used a code which has the E-step of EM implemented in MEX. For EM, we did 5 random initializations, and accepted the new set of parameters when we observed an increase in the log-likelihood. We repeated the experiment for 5 times. Error bars show the standard deviation of accuracy over the random repeats. We observed that the variance of the repeats vanished for longer sequences. The time-accuracy tradeoff curves averaged over 5 repeats are given in Figure 2.8. We see that MoM is faster for longer

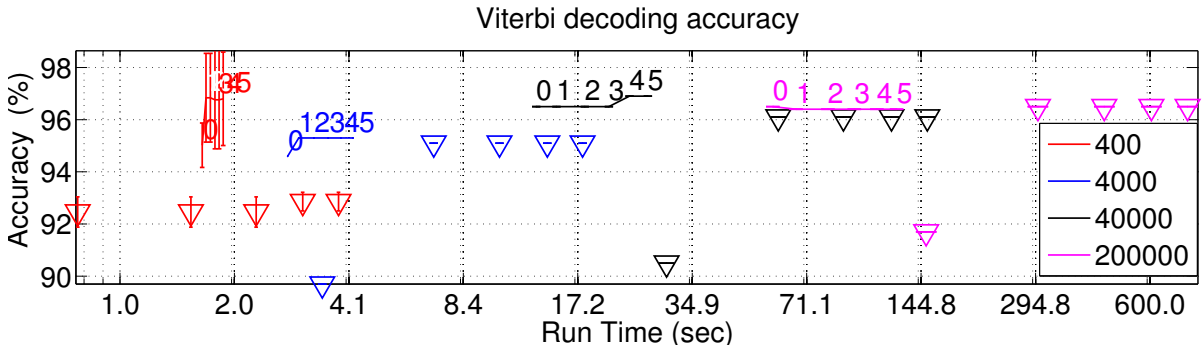


Figure 2.8: Time-Accuracy tradeoff curve for Synthetic Experiment. Different colors correspond to different sequence lengths. Triangles show the performance of randomly initialized EM. Different points with the same color correspond to an random EM initialization. (The further to the right, the larger the number of initializations) The solid curves show the performance of EM initialized with MoM. The numbers correspond to the number of EM iterations after initialization (0 means MoM only). Time axis is logarithmic.

sequences and thus more scalable than EM.

Real Data Experiment:

In this section, we work on detecting the speech onsets on a long sequence consisted of 48 concatenated utterances of digit 7 by the same person. We trained an ergodic Bakis HMM on the sequence, and used the Viterbi state sequence decoding to detect the utterance onsets. We defined an onset as a transition from the last state to the first state of the HMM, which are respectively determined by setting the first state as the very first and last elements of the Viterbi sequence. We used 29-dimensional MFCC features. To measure the performance of the onset detection we used the precision, recall and F-measure criterions defined in [59]. Similar to the previous section we compared the proposed algorithm (MoM),

randomly initialized EM (we used 5 random restarts and report only the restarts until the best F-measure), and EM initialized by MoM. Note that, the forward-backward part of the EM code is implemented with MEX (a C interface for MATLAB which accelerates the run time substantially), and the proposed method is implemented in MATLAB, with the optimization part implemented with CVX [60]. The F-measure - time tradeoff curves for 4 different sequence lengths are given in Figure 2.9. We are able to use sequences longer than 48 utterances by replicating the sequence. The numbers given in the legend correspond to the number of replicates.

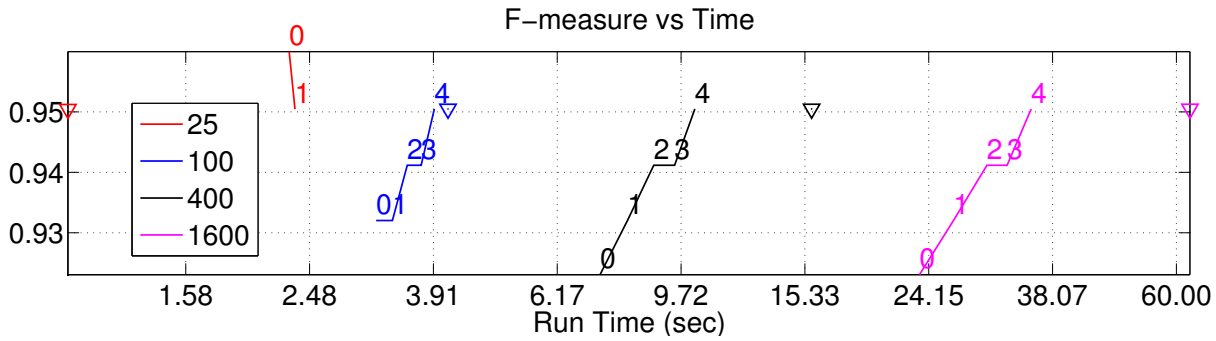


Figure 2.9: F-measure/time tradeoff curves. Different colors correspond to different number of replicates. Triangles correspond to randomly initialized EM. The solid curves correspond to EM initialized with MoM. The numbers show the number of EM iterations after initialization (0 means MoM only). Time axis is logarithmic.

As can be seen from the figure, the proposed Algorithm provides a more scalable alternative to EM. Even though the forward backward part of the EM code is implemented with MEX, the proposed algorithm is faster in longer sequence lengths. We also see that it's a fast way for initializing EM.

2.3 HIGH-LEVEL IMPRESSIONS ON METHOD OF MOMENTS (MOM) FOR LVMS

I can summarize my overall impression on method of moments learning algorithms for latent variables as follows:

- **Good**:
  - **Global:** This is probably the most attractive feature of MoM algorithms for me: The devised moment matching problem has a unique global optimum, and we have efficient algorithms to solve it.

- **Initialization:** Because we can get to the global optimum, we do not need to worry about initialization. Furthermore, as we demonstrated for MHMM and Left-to-right HMMs, we can initialize EM and improve the accuracy and speed of EM.
- **Scalable:** The algorithms are computationally cheap, therefore scalable. Namely, we only collect moments, and then factorize a small tensor.
- **Interesting/Theoretical:** Altough I haven't talked about it in this document, one reason people like these methods is because it possible to analyze them, and provide PAC style (probably approximately correct) style guarantees.
- **Subroutine:** As discussed earlier, altough the applicability of the MoM algorithms is limited compared to the established frameworks such as EM or MCMC methods, it is possible to use MoM algorithms as a subroutine. One application is using them in the M-step of EM (E.g. in a clustering algorihm, given a partition, we can fit individual models for each cluster using an MoM routine) I explored this idea in my master's thesis [61], and this potentially can be a good direction to explore.

- Bad:
  - **Limited Applicability:** This is the reason why it has been possible for us to publish papers. There is no established framework to capture a wide array of models, unlike EM or MCMC.
  - **Model Mismatch:** Since there is a hard assumption on the moments, the overall procedure is not very robust to model mismatch. Note that maximum likelihood explicity tries to minimize $\mathrm{KL}(p\|q)$, where p is the data distribution and q is the distribution/model we are trying to fit.
  - **Statistical Efficiency:** From Cramer-Rao lower bound, in terms of variance of the estimate we know that the most efficient parameter estimation method is maximum likelihood. Therefore, using MoM is statistically not the most efficient method.

- Ugly:
  - Since the eigenvalues algoritms utilized in estimation do not explicity constrain the estimates, it is possible to get estimates which are out of support of the distribution (e.g. it possible to obtain complex numbers for probabilities).

CHAPTER 3: IDENTIFIABLE LEARNING FOR FACTORIAL HIDDEN MARKOV MODELS

As we have introduced in Section 1.2.1, FHMM models the mixing process of sources in a cocktail party set-up. Although an accurate model, learning FHMM parameters from data is difficult. In fact, in this section we show that given the mixture, even if we know all the latent Markov chains, it is impossible to recover the correct emission matrices. Since the original model is unidentifiable, we propose two identifiable alternatives.

We note that for a factorial model, the emission model of a factorial model can be written in the following form:

$$\begin{aligned} x|r^{1:K} =& \sum_{k=1}^{K} O^k r^k + \epsilon \\ =& [O^1\ O^2\ \dots\ O^K] \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^K \end{bmatrix} + \epsilon \\ =& OR + \epsilon, \end{aligned} \tag{3.1}$$

where O, and R are the concatenated versions of emission matrices and the activations, respectively. Note that we use the one hot encoding to express the output with a matrix multiplication. The input/output shapes are as follows: $x \in \mathbb{R}^{L\times T}$, $O \in \mathbb{R}^{L\times KM}$, and $R \in \mathbb{R}^{KM\times T}$. Now, if want to get an estimate for the emission matrices O, and the activations R, we need to solve a problem very similar to the well known dictionary problem: [62, 63]

$$\begin{aligned} &\min_{O,R} \|x - OR\|_F^2 \\ &s.t. \text{ columns of } R \text{ are block } K \text{ sparse.} \end{aligned}$$

The dictionary learning is a very well studied in computer science and electrical engineering, and it is for example globally solvable for the following cases:

- **PCA:** Both O and R are orthogonal.
- **ICA:** Solvable if R has independent coordinates. [64, 50]
- **Mixture Model:** R is one sparse. Solvable is O has full column rank. [19]

- **Sparse Dictionary Learning:** Solvable if O is square and R is sparse Bernouilli-Gaussian. [65]

Despite the fact that we can view the learning problem for an FHMM as a dictionary learning problem, the problem is non-convex and consequently it is very difficult to reach a global optimum. But what is even worse for FHMM is, even if we are given the true activation matrix R, there are infinitely many plausible solutions for the emission matrices O. We formally prove this in the next subsection.

3.1 IDENTIFIABILITY IN STANDARD FACTORIAL MODEL

As stated earlier, the learning goal is to estimate the dictionary matrices

$$O^k = [\mu_1^k, \mu_2^k, \ldots, \mu_M^k],\ \forall k \in [K]$$

, up-to permutation of the columns $\mu_{1:M}^k$ of each dictionary, and up-to permutation of the dictionaries. We assume that the individual emission matrices have full column rank $\mathbf{rank}(O^k) = M$. Unfortunately the emission matrix of a Gaussian factorial model in its original form in [13, 12, 66] is unidentifiable: Even if an oracle gives the true assignment matrix R, there are infinitely many plausible dictionary matrices O. We will show that the assignment matrix R is rank deficient, which will lead us to the conclusion of unidentifiability.

Lemma 3.1. *Let $R^c \in \mathbb{R}^{MK \times M^K}$ denote a matrix whose columns consist of all possible combinations R_t can take (e.g. for $M = 2$, $K = 2$ case $R^c = \begin{bmatrix} e_1 & e_1 & e_2 & e_2 \\ e_1 & e_2 & e_1 & e_2 \end{bmatrix}$). We conclude that $\boldsymbol{rank}(R^c) = MK - (K - 1)$.*

Proof of Lemma 3.1. We will show this by computing the dimensionality of the left null space of R^c. Let,

$$r_m^k(m_1, m_2, \ldots, m_k, \ldots, m_K) := \begin{cases} 1, & \text{if } m_k = m \\ 0, & \text{otherwise} \end{cases},$$

where $k \in [K]$, and $m \in [M]$. This function returns the $(k-1)M+m$'th row of the column of R^c that corresponds to the combination represented by the tuple $(m_1, m_2, \ldots, m_k, \ldots, m_K)$, where $m_k \in [M]$. For a vector $\alpha \in \mathbb{R}^{MK} \in \mathbf{null}((R^c)^\top)$, by definition $\alpha^\top R^c = 0_{M^K}^\top$. Let us

consider the structure of such α:

$$\sum_{k=1}^{K}\sum_{m=1}^{M}\alpha_m^k r_m^k(m_1, m_2, \ldots, m_k, \ldots, m_K) = \sum_{k=1}^{K}\alpha_{m_k}^k = 0 \tag{3.2}$$

So, we see that the sum of the elements α that correspond to different k's should sum up to zero. Furthermore for a tuple that only differs in k'th element:

$$\sum_{k=1}^{K}\sum_{m=1}^{M}\alpha_m^k r_m^k(m_1, m_2, \ldots, \widetilde{m}_k, \ldots, m_K) = \sum_{k'\neq k}\alpha_{m_{k'}}^{k'} + \alpha_{\widetilde{m}_k}^k = 0, \tag{3.3}$$

where $m_k \neq \widetilde{m}_k$, and $\forall k \in [K]$. By comparing Equations (3.2) and (3.3), we see that $\alpha_{\widetilde{m}_k}^k = \alpha_{m_k}^k$. And consequently $\alpha_m^k = \alpha_{m'}^k$, $\forall (m, m') \in [M]$, and $\forall k \in [K]$. Together with the constraint $\sum_{k=1}^{K}\alpha_{m_k}^k = 0$, we conclude that $\mathbf{dim}(\mathbf{null}((R^c)^\top)) = K - 1$. Therefore, from the rank-nullity theorem, $\mathbf{rank}(R^c) = KM - \mathbf{dim}(\mathbf{null}((R^c)^\top)) = KM - (K-1)$. □

Corollary 3.1. *The rank of the assignment matrix $R \in \mathbb{R}^{MK\times T}$ is upper bounded: $\boldsymbol{rank}(R) \leq KM - (K-1)$.*

Proof of Correlary 3.1. The columns of the assignment are such that $R_t = R^c e_l,\ l \in [M^K]$. If R happens to contain all columns of R^c, it achieves the rank of R^c. In the case where R does not contain all columns of R^c, its rank is smaller than $KM - (K-1)$. Therefore, $\mathbf{rank}(R) \leq KM - (K-1)$. □

Theorem 3.1. *Given an assignment matrix $R \in \mathbb{R}^{KM\times T}$, the emission matrix of a Gaussian factorial model is not identifiable, meaning there exists $O_1 \neq O_2 \in \mathbb{R}^{L\times KM}$ such that $\prod_{t=1}^{T}\mathcal{N}(x_t|O_1R_t, \Sigma) = \prod_{t=1}^{T}\mathcal{N}(x_t|O_2R_t, \Sigma)$.*

Proof of Theorem 3.1. We observe that $\prod_{t=1}^{T}\mathcal{N}(x_t|O_1R_t, \Sigma) = \prod_{t=1}^{T}\mathcal{N}(x_t|O_2R_t, \Sigma)$, if $(O_1 - O_2)R_t = 0,\ \forall t \in [T]$, which is equivalent to $(O_1 - O_2)R = \mathbf{0}$. Due to Corollary 3.1, $\mathbf{dim}(\mathbf{null}(R^\top)) \geq K - 1$. Therefore we conclude that $(O_1 - O_2)R = 0$ for $O_1 \neq O_2$. □

We also intuitively see the model is unidentifiable since there are KM vectors to estimate in O but we only have $KM-(K-1)$ linearly independent equations, as Corollary 3.1 suggests. Making this observation, we reduce the number of model parameters to $KM - (K-1)$ by setting a shared component $\mu_M^k = s,\ \forall k \in [K]$, where $s \in \mathbb{R}^L$. □

Definition 3.1. ***(The Shared Component Factorial Model - SC-FM)*** *The emission matrix of a SC-FM is of the form* $\widetilde{O} = [\widetilde{O}^1, \ldots, \widetilde{O}^k, \ldots, \widetilde{O}^K, s]$, *where* $\widetilde{O}^k \in \mathbb{R}^{L\times(M-1)}$, *and* $s \in \mathbb{R}^L$ *is the shared component. The latent state indicators are either an indicator vector or an all zeros vector:* $\widetilde{r}_t^k \in (0_{M-1} \cup e_{1:M-1})$. *The columns of the assignment matrix* $\widetilde{R}$ *are of the form* $\widetilde{R}_t = [(\widetilde{r}_t^1)^\top, \ldots, (\widetilde{r}_t^k)^\top, \ldots, (\widetilde{r}_t^K)^\top, K - \sum_{k=1}^K \sum_{m=1}^{M-1} \mathbf{1}(\widetilde{r}_t^k = e_m)]^\top$.

Lemma 3.2. *Let* $\widetilde{R}^c \in \mathbb{R}^{(KM-(K-1))\times M^K}$ *denote a matrix whose columns consist of all possible combinations* $\widetilde{R}_t$ *can take. We conclude that* $\boldsymbol{rank}(\widetilde{R}^c) = KM - (K-1)$, *and consequently* $\boldsymbol{rank}(\widetilde{R}) \leq KM - (K-1)$. *For example, for the* $M = 3$, $K = 2$ *case,*

$$\widetilde{R}^c = \begin{bmatrix} e_1 & e_1 & e_2 & e_2 & e_1 & e_2 & 0_2 & 0_2 & 0_2 \\ e_1 & e_2 & e_1 & e_2 & 0_2 & 0_2 & e_1 & e_2 & 0_2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 \end{bmatrix}.$$

Proof of Lemma 3.2. We will prove this by showing that the left null space of $\widetilde{R}$ only contains an all-zeroes vector. Let,

$$\widetilde{r}_m^k(m_1, m_2, \ldots, m_k, \ldots, m_K) := \begin{cases} 1, & \text{if } m_k = m \\ 0, & \text{otherwise} \end{cases},$$

and $q(m_1, m_2, \ldots, m_k, \ldots, m_K) := K - \sum_{k=1}^K \mathbf{1}(m_k \neq 0)$ for $k \in [K]$, $m \in [M-1]$ and $m_k \in 0 \cup [M-1]$. The first function represents the first $(M-1)K$ rows, and the second function represents the last row of $\widetilde{R}^c$, for the column that corresponds to the tuple $(m_1, m_2, \ldots, m_k, \ldots, m_K)$. For a vector $\alpha \in \mathbb{R}^{KM-(K-1)}$ in the left null space of R^c, $\alpha^\top R^c = 0_{M^K}$. Let us evaluate $\sum_{k,m} \alpha_m^k \widetilde{r}_m^k(m_1, m_2, \ldots, m_k, \ldots, m_K) + \alpha_q q(m_1, m_2, \ldots, m_k, \ldots, m_K)$ for the tuple $(0, \ldots, 0)$:

$$\sum_{k=1}^K \sum_{m=1}^{M-1} \alpha_m^k \widetilde{r}_m^k(0, \ldots, 0) + \alpha_q q(0, \ldots, 0) = K\alpha_q = 0 \tag{3.4}$$

So we conclude that $\alpha_q = 0$. Next, we do the evaluation for the tuple $(0, \ldots, m_k, \ldots, 0)$, where only one element m_k is not equal to zero:

$$\sum_{k=1}^K \sum_{m=1}^{M-1} \alpha_m^k \widetilde{r}_m^k(0, \ldots, m_k, \ldots, 0) + \alpha_q q(0, \ldots, m_k, \ldots, 0) = \alpha_{m_k}^k + (K-1)\alpha_q. \tag{3.5}$$

By comparing Equations (3.4) and (3.5), we see that $\alpha_m^k = 0$, $\forall k \in [K]$ and $\forall m \in [M-1]$.

So, we conclude that $\mathbf{dim}(\mathbf{null}((\widetilde{R}^c)^\top)) = 0$, and therefore from the rank-nullity theorem, $\mathbf{rank}(\widetilde{R}^c) = KM - (K-1)$. And, if $\widetilde{R}$ contains all columns of $\widetilde{R}^c$ it has the same rank, which is the upper limit. □

Theorem 3.2. *Given an assignment matrix $\widetilde{R}$ which contains all columns of R^c, the emission matrix of an SC-FM is identifiable.*

Proof of Theorem 3.2. After going through the same reasoning in Lemma 3.1, we again end up with the condition of having the term $(\widetilde{O}_1 - \widetilde{O}_2)\widetilde{R}$ not equal to zero for two different emission matrices $\widetilde{O}_1 \neq \widetilde{O}_2$ for identifiability. As we have seen in Lemma 3.2, $\mathbf{dim}(\mathbf{null}(\widetilde{R}^\top)) = 0$ in the case where $\widetilde{R}$ contains all possible assignment vectors. Therefore we conclude that $(\widetilde{O}_1 - \widetilde{O}_2)\widetilde{R} \neq \mathbf{0}$ for $\widetilde{O}_1 \neq \widetilde{O}_2$, and consequently the emission matrix of an SC-FM is identifiable, given an assignment matrix $\widetilde{R}$. □

This theorem shows that the mapping $\widetilde{O} \to \widetilde{X} = \widetilde{O}\widetilde{R}$ is one-to-one. Even though this is the case, it is still not trivial to extract the columns of the emission matrix $\widetilde{O}$ from the observed data $\widetilde{X}$, simply because we do not have $\widetilde{R}$. However, we know the structure of $\widetilde{R}^c$, which contains all possibilities for the columns of $\widetilde{R}^c$. In the next section we will describe an algorithm which uses this fact.

3.2.1 Parameter Learning for Shared Component Factorial Model

We propose the following algorithm for learning the parameters of the shared component factorial model: We first calculate an estimate for the combinations observed in the data X, which we denote with $\widetilde{X}^c$, with a clustering algorithm. Naturally, columns of $\widetilde{X}^c$ contains an arbitrary and an unknown permutation, which leads us to the system $\widetilde{X}^c\Pi = \widetilde{O}\widetilde{R}^c\Pi$, where $\Pi \in \mathbb{R}^{KM \times KM}$ is a permutation matrix. This system has a different solution for different Π matrices, and therefore we cannot solve this system for the true emission matrix unless we know Π. However, by assuming that the shared component s is less correlated to the non-shared components than the correlation between the non-shared components, we will show that it is possible to extract the components by computing pairwise correlations between the columns of $\widetilde{X}^c$.

To reduce the notation clutter we drop tilde's, although we still refer to the SC-FM parameters, and we use the regular factorial model notation where the indicator variable $r_t^k \in [M]$, for $k \in [K]$, where the notation $[N]$, for some integer N denotes the set $\{1, \ldots, N\}$. Conforming with that notation we set the last columns of all the emission matrices to be the shared component, such that $\mu_M^k = s$, $\forall k \in [K]$. E.g., for $M = 2$, $K = 2$ case $O = [\mu_1^1, s, \mu_1^2, s]$.

Learning the emission matrix from X^c

In this section we describe an algorithm which extracts the columns of the emission matrix by looking at the pairwise correlations of the columns of X^c matrix. The first step is to find which column of X^c corresponds to the shared component.

Definition 3.2. *Let x_l denote l'th column of X^c, so $x_l := X^c(:,l) = \sum_{k=1}^{K}\sum_{m=1}^{M-1}\mu_m^k r_{m,l}^k + \sum_{k=1}^{K} s\, r_{M,l}^k$, where $r_{m,l}^k, l \in [M^K]$ denotes the m'th entry of an indicator vector of length M where only the m'th entry is one and the rest is zero, for the k'th emission matrix and l'th possible combination.*

Definition 3.3. *Let $v(x_{l'}) : \mathbb{R}^L \to \mathbb{R}^{M^K}$ denote a vector valued function with the argument $x_{l'}$, such that $v(x_{l'}) = \omega\left([\langle x_1, x_{l'}\rangle, \langle x_2, x_{l'}\rangle, \ldots, \langle x_l, x_{l'}\rangle, \ldots, \langle x_{M^K}, x_{l'}\rangle]\right)$, where $\omega : [M^K] \to [M^K]$ is an ascending sorting mapping such that $v_1(x_{l'}) \leq v_2(x_{l'}) \leq \cdots \leq v_{M^K}(x_{l'})$, where $v_l(x_{l'})$ is the l'th smallest element in $v(x_{l'})$ vector.*

Lemma 3.3. *If $\left\langle \mu_{m''}^{k''}, s\right\rangle \leq \left\langle \mu_m^k, \mu_{m'}^{k'}\right\rangle$, $\forall (k,k',k'') \in [K]$, and $\forall (m,m',m'') \in [M-1]$, i.e. for any component μ_m^k, the least correlated component is s, and $\left\langle \mu_m^k, s\right\rangle \leq \langle s, s\rangle$, $\forall k \in [K]$, $m \in [M-1]$, i.e., the shared component s has a non-trivial magnitude (e.g. all zeros vector doesn't satisfy this condition), then*

$$Ks = argmin_{x_{l'}, l' \in [M^K]} \sum_{l=1}^{(M-1)^K} v_l(x_{l'}),\ for\ M > 2, K \geq 1. \tag{3.6}$$

Proof of Lemma 3.3. We want to show that given that the specified incoherence conditions are satisfied, the sum of the smallest $(M-1)^K$ terms in $\{\langle x_l, x_{l'}\rangle : l \in [M^K]\}$ get minimized when we set $x_{l'} = Ks$. In the proof given in supplemental material, we consider all possibilities for $x_{l'}$ and conclude that the minimizing possibility is Ks. □

Lemma 3.3 suggests that by computing pairwise correlations, it is possible to identify the column in X^c which corresponds to Ks component: The summation of first $(M-1)^K$ terms in $v(x_{l'})$ is minimized when we set $x_{l'} = Ks$. Therefore, we compute $v(x_{l'})$ for all columns of X^c, and assign the minimizing column to the term Ks. In $M = 2$ case argmin of this summation contains multiple minimizers (including Ks), and we suggest a fix for that specific case with an additional assumption in the supplemental material. Now that we know how to estimate the Ks term, next we look at the structure of $v(Ks)$ to extract the non-shared components.

Definition 3.4. *Let $\mathcal{B}_{K'} := \{l \in [M^K] : \sum_{k=1}^{K} r_{M,l}^k = K'\}$, i.e. the indices l for which s*

appears $K - K'$ *times, which corresponds to the terms of the form* $\sum_{k=1}^{K}\sum_{m=1}^{M-1}\mu_m^k r_{m,l}^k + (K-K')s$, $l \in \mathcal{B}_{K'}$.

Lemma 3.4. *Let* $B_l^{K'} := \left\langle \sum_{k=1}^{K}\sum_{m=1}^{M-1}\mu_m^k r_{m,l}^k + (K-K')s, Ks \right\rangle$, $l \in \mathcal{B}_{K'}$. *If* $\langle s, \mu_m^k\rangle \leq \langle s, s\rangle$, $\forall k \in [K]$, *and* $\forall m \in [M-1]$, *then for* $M^K - (M-1)K \leq l' \leq M^K - 1$, $v_{l'}(Ks) = B_l^1$ *for some* $l \in \mathcal{B}_1$.

Proof of Lemma 3.4. Let us expand the expression $B_l^{K'}$:

$$B_l^{K'} = K\sum_{k=1}^{K}\sum_{m=1}^{M-1}\left\langle \mu_m^k, s\right\rangle r_{m,l}^k + (K-K')K\left\langle s, s\right\rangle, l \in \mathcal{B}_{K'}.$$

Since only K' terms are active on the first term, and due to the condition $\langle s, s\rangle \geq \left\langle s, \mu_m^k\right\rangle$, $\forall k \in [K]$, $\forall m \in [M-1]$, we see that the above expression reaches the maximum value when $K' = 0$. By the same token, we conclude that $B_l^1 > B_{l'}^{K'}$, $\forall K' > 1$, $l \in \mathcal{B}_1$, $l' \in \mathcal{B}_{K'}$, since the number of $\langle s, s\rangle$ terms decrease as K' increases. Therefore, the largest elements of $v(Ks)$ after $v_{M^K}(Ks)$ correspond to B_l^1, $l \in \mathcal{B}_1$, as suggested by the lemma. □

We had an estimate for s in the previous step, and now that we know which observed x_l vectors correspond to the vectors comprised partly of $(K-1)s$ (i.e. terms corresponding to $\mathcal{B}_1$) from Lemma 3.4, we can estimate the non-shared components simply by subtracting $(K-1)s$ from each term in $\mathcal{B}_1$. The only remaining problem is to group them into proper emission matrices $O^{1:K}$.

Finding the grouping of the components

We know from Lemma 3.3 that the $(M-1)^K$ smallest elements of $v(Ks)$ (which also correspond to $\mathcal{B}_K$) are associated with all possible combinations of non-shared components that do not contain any term involving s. To find the groupings for the dictionary elements we solve a linear system of the form $Y = WH$ for H, where the columns of the W matrix are the non-shared components estimated by subtracting $(K-1)s$ from components corresponding to $\mathcal{B}_1$, and columns of Y correspond to all possible combinations of the non-shared components which correspond to $\mathcal{B}_K$. Solving this system figures out which combinations of the non-shared components corresponding to $\mathcal{B}_1$ add up to the combinations corresponding to $\mathcal{B}_K$, which are encoded in H. In practice we have observed that solving the following optimization problem which enforces sparsity on the columns of H works well: $\widehat{H} = \text{argmin}_H \|\widehat{Y} - \widehat{W}H\|_F + \sum_t \|H(:,t)\|_1$.

Summary of emission matrix learning

For a shared component factorial model (HMM or Mixture model), given the matrix of all possible observations $X^c \in \mathbb{R}^{L \times K^M}$, and provided that the columns of the emission matrix satisfy $\left\langle \mu_{m''}^{k''}, s \right\rangle \leq \left\langle \mu_m^k, \mu_{m'}^{k'} \right\rangle$, and $\left\langle \mu_{m''}^{k''}, s \right\rangle \leq \langle s, s \rangle$, $\forall (k, k', k'') \in [K]$, $k \neq k'$ and $\forall (m, m', m'') \in [M-1]$, Algorithm 3.1 finds the columns of the emission matrix O upto permutation among the columns of each emission matrix O^k and permutation of the emission matrices.

Algorithm 3.1 Emission matrix learning for F-GMM/F-HMM.

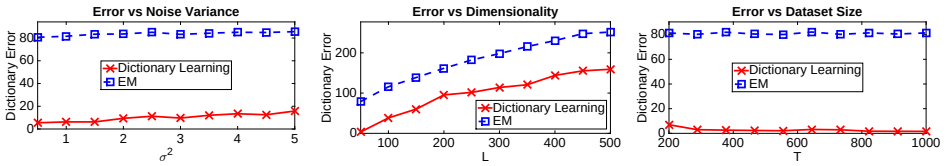
Input: The clustered data matrix $X^c \in \mathbb{R}^{L \times K^M}$
Output: Estimated emission matrix $\widehat{O} \in \mathbb{R}^{L \times KM}$

- Compute the correlation matrix $C_{i,j} = \langle X^c(:,i), X^c(:,j) \rangle$, $\forall i, j \in \mathbb{R}^{M^K}$.
- Let C^s denote the C matrix with sorted rows in increasing order. Set $i^* = \text{argmin}_i \sum_{j=1}^{(M-1)^K} C_{i,j}^s$, $v = C^s(:, i^*)$, and $\widehat{s} = X^c(:, i^*)/K$.
- Find the indices of $(M-1)K$ largest elements in v, write the indices in $\mathcal{B}_1$. Set $\widehat{W} = X^c(:, \mathcal{B}_1) - (K-1)s1_{K-1}^\top$.
- Find the indices of $(M-1)^K$ smallest elements in v, write the indices in $\mathcal{B}_K$. Set $\widehat{Y} = X^c(:, \mathcal{B}_K)$.
- Set $\widehat{H} = \text{argmin}_H \|\widehat{Y} - \widehat{W}H\|_F + \sum_t \|H(:,t)\|_1$, and group the columns of $\widehat{W}$ according to $\widehat{H}$ in $\widehat{O}$.
- Output the corresponding estimate $\widehat{O}$.

Estimating the auxiliary parameters

Hidden state parameters:
Once we have an estimate $\widehat{O}$ for the emission matrix, the assignment matrix can be estimated by solving the optimization problem, $\widehat{R} = \text{argmin}_R \|\widehat{O}R - X\|_F + \sum_{t=1}^T \|R(:,t)\|_1$. We estimate the assignment probabilities $\pi^{1:K}$ for F-GMM, or the transition matrices $A^{1:K}$ for F-HMM simply by counting the occurrences in $\widehat{R}$:
$\widehat{\pi}_i^k = \frac{1}{T} \sum_{t=1}^T \mathbf{1}(\widehat{r}_t^k = e_i)$, $\widehat{A}_{i,j}^k = \frac{1}{T-1} \sum_{t=1}^{T-1} \mathbf{1}(\widehat{r}_{t+1}^k = e_i)\mathbf{1}(\widehat{r}_t^k = e_j)$, $i, j \in [M], k \in [K]$. In practice, $\widehat{R}$ is noisy and the entries are not binary. We threshold the $\widehat{R}$ matrix to make it binary before the counting step.

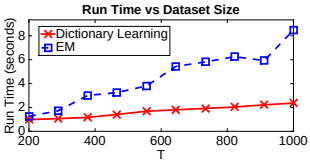
Covariance matrix:
Once we have estimates for the emission and the assignment matrix, we subtract the reconstruction from the data to make it zero mean. After that the covariance matrix is estimated



(a) Error vs σ^2, $L = 50$, $T = 200$.

(b) Error vs L, $\sigma^2 = 0.5$, $T = 200$.

(c) Error vs T, $L = 50$, $\sigma^2 = 0.5$.



(d) Run time vs T, $L = 50$, $\sigma^2 = 0.5$.

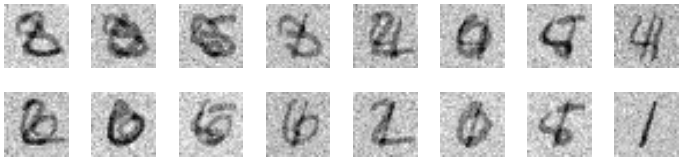
Figure 3.1: Various performance measures for the proposed algorithm and EM on synthetic data averaged over 50 trials.

with the usual covariance estimator: $\widehat{\Sigma} = \frac{1}{T-1}\sum_{t=1}^{T}\left(X_t - (\widehat{O}\widehat{R})_t\right)\left(X_t - (\widehat{O}\widehat{R})_t\right)^\top$, where $(\widehat{O}\widehat{R})_t$ denotes the reconstruction at time t.

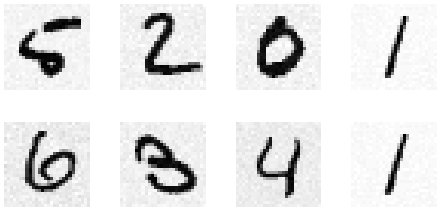
Experiments

Synthetic Data We conducted experiments with synthetic data generated from shared component factorial model. We set $M = 3$ and $K = 2$. The columns of the emission matrix are sampled from a Gaussian with variance 10. The observation noise variance σ^2, data dimensionality L, and number of observations T were all varied to compare the behavior of the proposed approach and EM. For the clustering step in the proposed approach, we applied the algorithm in [67]. For EM, we used 10 restarts with dictionaries started at the perturbed versions of the mean of the observed data. We report the result of the initialization that resulted in the highest likelihood. As error, we report the euclidean distance between the estimated dictionary matrix O and the true dictionary, by resolving the permutation ambiguity. Figure 3.1 shows various comparisons between the two algorithms in terms of accuracy in recovering the true dictionaries and run time. The parameter setup for the fixed variables is shown under each figure. We see that the algorithm works much better than EM in general. We also see from Figure 3.1d that the proposed approach is faster, and potentially more scalable than EM.

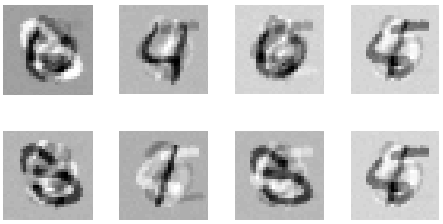
Digit Data In this experiment, we work with digit images from the MNIST dataset [5]. We compare the proposed dictionary learning approach with an EM algorithm [21], on synthetically combined images according to the shared component factorial model, where



(a) All possible combinations for the observations



(b) Dictionary Learning



(c) EM

Figure 3.2: Unmixing of synthetically mixed noisy digit images with SC-FM. Figures (b) and (c) show the the learned emission matrices for the proposed algorithm and EM. A row in Figures (a) and (b) corresponds to the components corresponding to the same group.

we set $M = 4$, and $K = 2$. We generate 2000 such images. The images are of size 28×28. We normalize the pixel values so that they take on values between 0 and 1. We add spherical Gaussian noise with standard deviation $\sigma = 0.22$ to every generated image. We initialize the columns of the emission matrix in EM with the randomly perturbed versions of the mean of the generated data. We do 10 such random initializations and pick the initialization with the highest likelihood. In Figure 3.2, we show the all noisy versions of 16 possible combinations. We also show the reshaped versions of the learned columns of the dictionaries for the proposed algorithm and EM.

We see that the estimates obtained with dictionary learning approach are close to the true digits, whereas EM finds a local solution which deviates from the true digits significantly.

Conclusions & Discussion for SC-FM

We have shown that the standard factorial model in the literature is not learnable. We then proposed an exact algorithm for the case where there is a one column sharing assumption between K emission matrices. Although the proposed algorithm solves the parameter estimation problem in a global fashion, it requires all combinations to be observed, which scales exponentially as $\mathcal{O}(M^K)$. Furthermore, the shared component assumption does not hold in many cases. In the next section, we propose another identifiable factorial model for which we only require linearly many combinations to be observed.

3.3 REVEALING FACTORIAL MODEL

In this section we show that introducing random binary switches for each observation gives us an identifiable model. The reason behind why the model is identifiable is because thanks to the random binary switches, some of the component do contribute to the observations in isolation, which enables to obtain a direct estimate for dictionary elements (columns of the O matrix). The corresponding generative process can be written as follows:

$$\begin{aligned}
s_t^k &\sim \mathcal{BE}(\pi),\ k \in \{1, \dots, K\}, \\
r_t^1 | r_{t-1}^1, s_t^k &\sim s_t^1 \text{Discrete}(A^1 r_{t-1}^1), \\
&\vdots \\
r_t^K | r_{t-1}^K, s_t^k &\sim s_t^K \text{Discrete}(A^K r_{t-1}^K), \\
x_t | r_t^1, \dots, r_t^K, s_t^1, \dots, s_t^K &\sim \mathcal{N}([O^1 s_t^1, \dots, O^K s_t^K] \begin{bmatrix} r_t^1 \\ \dots \\ r_t^K \end{bmatrix}, \sigma^2 I),
\end{aligned}$$

where s_t^k denotes the introduced switch variable. (Other than the switch variable the model remains exactly the same as the original factorial model) The corresponding emission model is defined as follows:

$$x_t = [O^1 s_t^1, O^2 s_t^2, \dots, O^K s_t^K] R_t + \epsilon,\ \forall t \in [T], \tag{3.7}$$

where the only modification to the standard factorial model is the addition of the switch variable $s_k \in \{0, 1\}$ which denotes a binary *switch* variable assigned to the k'th factor: If s_k is equal to one, then the k'th factor is active. Otherwise the k'th factor does not contribute to the observation at time step t. Alternatively, we can represent the activation variables $R_t = [(r_t^1)^\top, (r_t^2)^\top, \dots, (r_t^K)^\top]^\top$, by adding the zero vector to the domain of indicators for each factor such that $r_t^k \in e_{1:M^{(k)}} \cup \mathbf{0}_{M^{(k)}}$, $\forall k \in [K]$.

3.3.1 Identifiability of the modified model

Lemma 3.5. *Let $R^c \in \mathbb{R}^{KM \times C_{all}}$ denote a matrix which consists of all possible combinations R^t can take for an RFM. We conclude that **rank**$(R^c) = KM$. For example for $M = 2$,*

$K = 2$ *case,* $R^c = \begin{bmatrix} e_1 & e_1 & e_2 & e_2 & e_1 & e_2 & \mathbf{0}_M & \mathbf{0}_M \\ e_1 & e_2 & e_1 & e_2 & \mathbf{0}_M & \mathbf{0}_M & e_1 & e_2 \end{bmatrix}$.

Proof of Theorem 3.5. Following the definition given for RFM above, we can see that columns of the new combinations matrix R^c contains the canonical basis set $e_{1:MK}$. We therefore conclude that the matrix R^c has full row rank, and thus $\mathbf{rank}(R^c) = KM$. □

Theorem 3.3. *Given an assignment matrix $R \in \mathbb{R}^{KM \times T}$ whose columns contain the canonical basis set $e_{1:MK}$, the emission matrix of an RFM is identifiable, meaning for all non-equal emission matrices $O_1 \neq O_2 \in \mathbb{R}^{L \times KM}$, the corresponding likelihoods are not the same: $\prod_{t=1}^{T} \mathcal{N}(x_t | O_1 R_t, \Sigma) \neq \prod_{t=1}^{T} \mathcal{N}(x_t | O_2 R_t, \Sigma)$.*

Proof of Theorem 3.3. We know from Lemma 3.5 that $\mathbf{rank}(R^c) = MK$ for an RFM. Therefore from rank-nullity theorem we see that $\mathbf{dim}(\mathbf{null}((R^c)^\top)) = 0$. Thus $(O_1 - O_2)R \neq 0$ for $O_1 \neq O_2$. □

Now that we have established that the revealing factorial model is identifiable, we describe an efficient algorithm for estimating the emission matrix of an RFM in the next section. □

3.3.2 Learning

In this section we describe an efficient (a polynomial time) algorithm to estimate the emission matrix of an RFM for the case where [1]$K = 2$. The algorithm is described in Algorithm 3.2. The explanations for each step of this algorithm are as follows:

1.The clustering step: In the first step of Algorithm 3.2, the data items (the columns) of the input data matrix X are clustered so as to produce an approximated version of X. The obtained cluster centers are denoted with $X^c \in \mathbb{R}^{L \times C}$, which we refer to as the *combinations matrix*. The subsequent steps of Algorithm 3.2 are not based on an exact knowledge of the number of cluster representatives C, and hence do not require for X^c to form a "perfect" clustering of the input data. This is demonstrated in Figure 3.5 in the experiments section.
2.Estimating the activations for the X^c matrix: The combinations matrix X^c contains the representatives for the distinct observations in the observation matrix X. By finding a non-trivial solution that minimizes the error $\|X^c - X^c H\|$ with respect to $H \in \mathbb{R}^{C \times C}$, we aim to discover tuples of columns (pairs in the $K = 2$ case) that sum up to an other

[1]We set $K = 2$ in the following sections in order to to simplify the narrative. The proposed algorithm is valid for the cases where $K > 2$ with straightforward modifications.

Algorithm 3.2 Parameter estimation for RFM

Input: The observation matrix $X \in \mathbb{R}^{L \times T}$.
Output: Estimated emission matrix $\widehat{O}$.

1. Apply clustering on X to get cluster centers $X^c \in \mathbb{R}^{L \times C}$.
2. Solve the following convex optimization problem:

$$\begin{aligned} &\min_H \|X^c - X^c H\|_F^2 + \beta \|H\|_1, \\ &s.t.\ H_{i,i} = 0, \text{ for } 1 \leq i \leq C, \\ &\quad H \geq 0, \end{aligned} \tag{3.8}$$

where $H \in \mathbb{R}^{C \times C}$. We denote the solution with $\widehat{H}$.
3. Construct a bipartite graph G from $\widehat{H}$. Let $\mathcal{S}_k$ denote the k'th part of G (one of the 2 disjoint node sets).
Set $\widehat{O} = X^c(:, \{\mathcal{S}_1 \cup \mathcal{S}_2\})$.

column of X^c. The non-negativity constraint, together with the constraint that sets the diagonal of H to zero pushes the solution away from the trivial identity matrix solution. Furthermore, the term $\|H\|_1$ pushes the solution towards sparser solutions. For an example combinations matrix $X^c = \begin{bmatrix} x+y & x & y \end{bmatrix}$, where x, y are arbitrary real vectors, the solution to the optimization problem in Algorithm 3.2 would be $\widehat{H} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$, for large enough β. In a real world application the combinations matrix X^c would typically be contaminated with noise, and therefore extracting pairs from it is not trivial, as we get many non-zero elements $\widehat{H}$. We deal with this issue in the next step.

3.Reading the $\widehat{H}$ matrix: The goal in this step is to construct a bi-partite graph $G = (V, E)$, where the vertex set V is decomposed into two disjoint sets such that $V = \mathcal{S}_1 \cup \mathcal{S}_2$, and $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$, where $V, E, \mathcal{S}_1, \mathcal{S}_2 \subset [C]$, by reading the graph edges from columns of $\widehat{H}$. The model definition of the factorial model implies that the output combinations in X^c where all factors are active (the combinations for which $r_t^1 \neq \mathbf{0}$ and $r_t^2 \neq \mathbf{0}$) would correspond to the edges of a bi-partite graph. This is demonstrated in Figure 3.3, for an example combinations matrix $X^c = [x_0, x_1, y_0, y_1, x_0 + y_0, x_0 + y_1, x_1 + y_1]$: We see that the combinations $x_0 + y_0, x_0 + y_1, x_1 + y_1$ correspond to the cases where both sets contribute to the output (We will refer to such combinations as *all-active combinations*). Our task in this case is to group the emission matrix columns x_0, x_1, y_0, y_1 (which correspond to the nodes of the graph G) in two disjoint sets $\mathcal{S}_1$ and $\mathcal{S}_2$, given the all-active combinations (edges) that

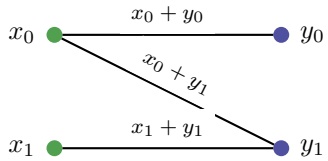


Figure 3.3: The bi-partite graph that corresponds to an example combinations matrix $X^c = [x_0, x_1, y_0, y_1, x_0 + y_0, x_0 + y_1, x_1 + y_1]$: The bi-partition yields the emission matrices $O_1 = [x_0\ , x_1]$, and $O_2 = [y_0\ , y_1]$. Note that the combination $x_1 + y_0$ is absent from this output instance.

we infer from the columns of the $\widehat{H}$ matrix.

Typically, the output of the optimization problem in step 2 is noisy, so we can not directly apply a standard graph bi-partition algorithm on the columns of the $\widehat{H}$ matrix. Instead, we apply the following greedy procedure: We first list all possible pairs (edges) which are formed by the non-trivial entries of $\widehat{H}$ (entries in the same column, larger than a small value). We then eliminate all combinations which are formed by similar pairs, by using cosine similarity as the measure. We also make sure that the edge set and the vertex set do not overlap. After that, we form bi-partite graphs on the connected components obtained from the list of candidate edges. We finally merge the disconnected bi-partite graphs, if there is a significant similarity between nodes in two different sets which do not belong to the same bi-partite graph.

The important thing to note is that, for unique bi-partition the graph G has to be connected. E.g. in graph given in Figure 3.3, suppose that the edge $x_1 + y_0$ is absent. Note that, $\mathcal{S}_1 = \{x_0, x_1\}, \mathcal{S}_2 = \{y_0, y_1\}$ and $\mathcal{S}_1 = \{x_0, y_1\}, \mathcal{S}_2 = \{y_0, x_1\}$ are both valid bi-partitions. The overall algorithm therefore requires for proper grouping that the all-active combinations that are present in the combinations matrix X^c to form a connected bi-partite graph. Note that the minimum number of such combinations grows linearly in M (and not quadratically). Note that we also require observing all nodes of the graph to be observed also (We require observing all columns of the emission matrix in isolation - This condition can in future be relaxed by potentially using a non-negativity constraint for the dictionary elements).

3.3.3 Experiments

In this section we demonstrate the validity and robustness of the proposed algorithm by an unsupervised audio source separation experiment. We generate 100 sequences of clarinet-double bass mixture using a midi-synthesizer which samples at 16kHz. In a random order, we sample randomly chosen notes for each instrument for a prespecified range by making sure that the observed combinations form a connected bi-partite graph. The duration of each

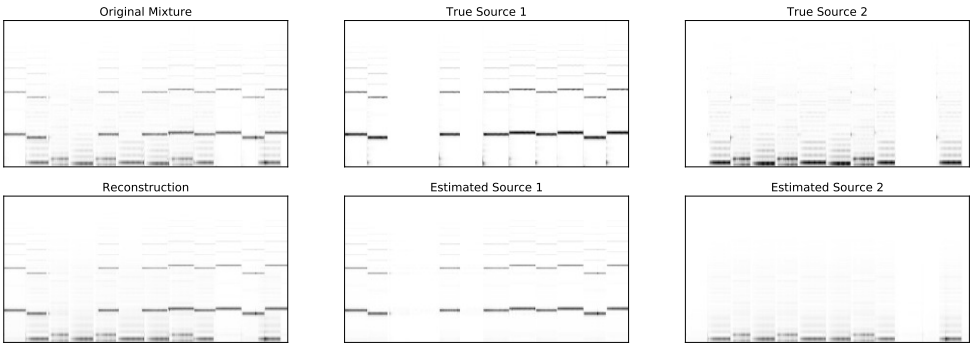


Figure 3.4: Example spectrograms from the source separation from the used data set. (Left-top) Given Mixture (Left-bottom) Reconstruction as sum of two estimated sources (Middle-top) Ground Truth Source 1 (Middle-bottom) Estimated Source 1 (Right-Top) Ground Truth Source 2 (Right-bottom) Estimated Source 2. In this example BSS eval scores are: SDR= 17.4 SIR=22.6 SAR=19.0 (dB).

note is sampled from a normal distribution centered around 1 second and with a standard deviation of 0.1 seconds.

For a given audio mixture, we compute a short-time Fourier transform spectrogram with 1024 point fft and hop-size 256. Before running our algorithm, we also warp the magnitude spectrogram into Bark scale [68] for each time window to increase the distinctiveness of observed spectra. After the separation on magnitude spectrograms on Bark scale, we map the separated source spectrograms back to regular frequency domain. We element-wise multiply the separated spectra with the phase of the original mixture before reconstructing the separation results. We used k-means initialized by k-means++ [69] as the clustering algorithm. We have set the sparsity parameter $\beta = 1$ in our experiments.

To estimate the separated magnitude source spectra $\widehat{X_1}$, $\widehat{X_2}$, we compute the pseudo inverse of the estimated emission matrix $\widehat{O}$ and multiply it with the magnitude spectrum of the mixture to get an estimate for the activation matrix $\widehat{R} = \widehat{O}^{\dagger} X$. We then set $\widehat{X_k} = \widehat{O}\widehat{R}(:, \mathcal{S}_k)$, for $k \in 1, 2$.

When we set the number of clusters parameter $C = 12$, we obtain average BSS-Eval [70] SDR, SIR, SAR scores on our dataset respectively as **15.9**, **21.7**, and **17.4** dB. We demonstrate an example source separation with $C = 12$ in Figure 3.4. As can be seen from the figure, the separation results are very close to the ideal solution. The results on the whole dataset for $C = 12$ can be found in the link `http://cal.cs.illinois.edu/~cem/sourcesep/sourcesep_images.html` (We suggest using Chrome as browser, and suggest downloading the file if it does not play on the browser.)

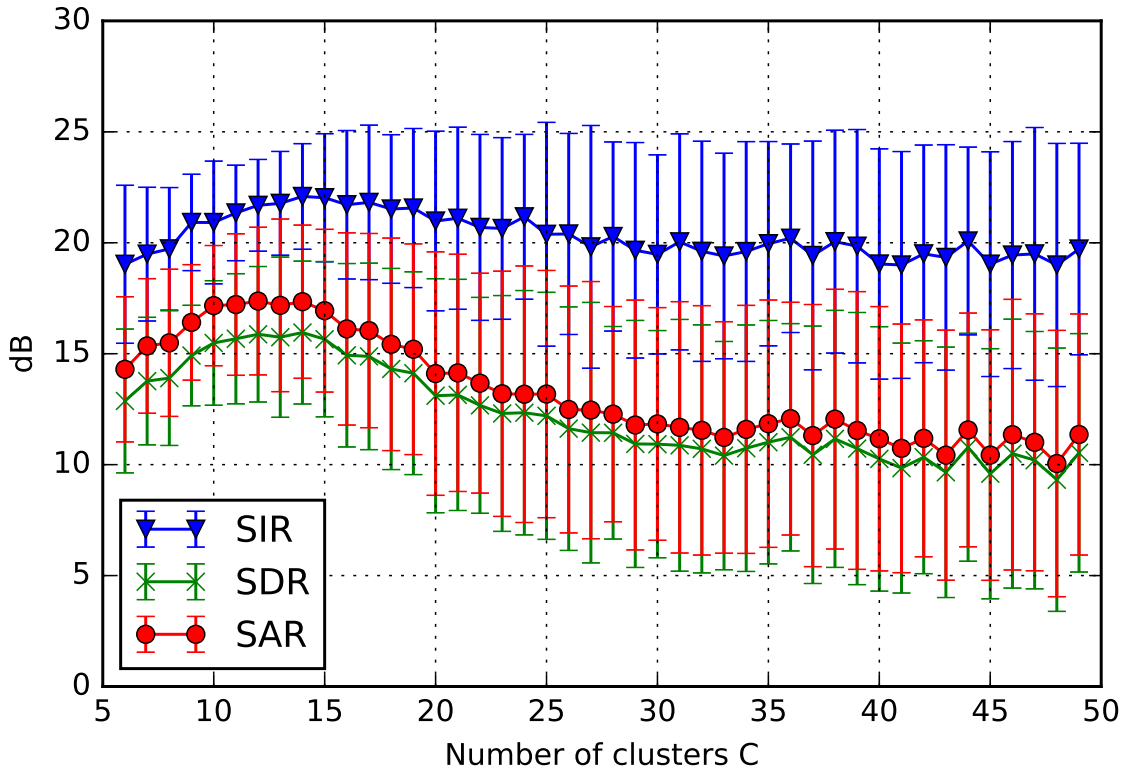


Figure 3.5: Average SDR, SIR and SAR values, along with corresponding error-bars obtained on our source separation dataset.

Further on, we test the robustness of the whole algorithm to the choice of number of clusters C. In Figure 3.5, we report the average BSS-eval scores with error-bars that shows one standard deviation, for varying number of clusters C ranging from 6 to 49. We see that SDR peaks around $C = 12$. Although we see that the scores tend to decrease after the peak at $C = 12$, they do not diminish drastically, and the algorithm is still able to get a reasonable SDR score with increasing C values. Note that SIR remains relatively steady around 20 dB.

3.3.4 Conclusions and Potential Future Improvements

We have proposed an identifiable alternative to standard Factorial model, which is more realistic than the shared component factorial model we have proposed above. We have also proposed an efficient algorithm to learn the parameters of this model. The algorithm is polynomial time and only requires linearly many combinations to be observed, instead of the exponential number of combinations required for the algorithm for the SC-FM model.

At its current state the algorithm requires all the nodes in the parameter bi-partite graph to be observed. A potential future work might focus on relaxing the constraint that requires all nodes to be observed to a subset of the nodes to be observed, by potentially using a non-negativity constraint on the dictionary elements.

CHAPTER 4: LEARNING THE BASE DISTRIBUTION IN IMPLICIT GENERATIVE MODELS

As we talked about on the introduction section, generative model learning is the task where the goal is to learn a model to generate artificial samples which follow the underlying probability density function of a given dataset. When the dataset comprises of scalars, or of low dimensional (2-3 dimensions) vectors and follow a unimodal distribution, one can use a simple density model such as the multivariate Gaussian, and fit the model to the data using maximum likelihood. Unfortunately, such simple densities do not have sufficient expressive power to learn the distribution of more complicated data such as natural images, or audio because of the aforementioned high dimensional and multi-modal nature of the data.

There exists several generative model learning methods in the machine learning literature. One way of approaching the problem is to use a linear latent variable model (LVM) such as a mixture model [4], a latent factor model such as probabilistic PCA [71], Hidden Markov model (HMM) [26], or linear dynamical systems [4, 11]. These models can successfully capture the multi-modality, or low rank nature of the datasets, however they rely on linear and tractable forward mappings, and therefore lack the expressive power of modern neural network models.

More recently, the mainstream approaches for learning a generative model for complicated datasets have been centered around models that combine latent variable modeling with non-linear neural network mappings. One prominent example of such approaches is Variational Autoencoders (VAEs) [15]. VAEs consider a latent variable model where the latent representation is mapped to the observation space via a complicated neural network. The variational expectation maximization algorithm in [15] maximize a variational lower bound on the maximum likelihood objective. The prior distribution is typically chosen as a simple distribution such that the KL-divergence term in the lower bound is tractable. In this chapter we argue that using a simple prior distribution is detrimental to the overall quality of the learned generative model.

Another very popular method that also uses a restricted latent representation is Generative Adversarial Networks (GANs) [16]. The main conceptual differences of GANs from typical latent variable models (including VAEs) is that GANs are an implicit generative model learning methodology [72], where the model distribution is defined without specifying an output density. More importantly, unlike LVMs GANs do not maximize the standard maximum likelihood objective. Instead, GANs approximate the underlying dataset density via an additional discriminator network. Although an appealing idea, GANs are incredibly hard to train (as evidenced by the sheer number of GAN training papers in the last few

years), and suffer from the predictable mode collapse problem (We delve more into this in the main text).

In this chapter, we propose an implicit generative model learning method which maximizes the maximum likelihood training objective. Unlike GANs, the method does not rely on auxiliary networks such as discriminator or critic networks. For training, we propose a simple two stage training method, which maximizes a maximum likelihood training objective, and therefore does not suffer from the mode collapse problem that GANs are notorious for.

4.1 GENERATIVE MODEL LEARNING

The purpose of this section is to set the notation and the required concepts before we formally introduce our algorithm. As we discussed in the introduction, the goal in generative model learning is to approximate the underlying data density $p_{\text{data}}(x)$ with the density that our model implies, which we denote by $p_{\text{model}}(x|\theta)$, where θ denotes the model parameters. Maximum likelihood training minimizes the Kullback-Leibler (KL) divergence between the data density and model density:

$$\begin{aligned}
&\min_{\theta} \ \mathrm{KL}(p_{\text{data}}(x) \| p_{\text{model}}(x|\theta)) \\
&= \min_{\theta} \int p_{\text{data}}(x) \log \frac{p_{\text{data}}(x)}{p_{\text{model}}(x|\theta)} dx \\
&\propto \min_{\theta} - \int p_{\text{data}}(x) \log p_{\text{model}}(x|\theta) dx \\
&\approx \max_{\theta} \sum_{n} \log p_{\text{model}}(x_n|\theta)
\end{aligned} \tag{4.1}$$

where the last step is a Monte Carlo approximation to the integral, and we recognize Equation (4.1) as the maximum likelihood objective. Note that $x \in \mathbb{R}^L$ denotes the variable we use to denote the observation space, and we use the subscripted version x_n to denote the data item with index n.

It is usually not easy to compute (not tractable) the likelihood function $p_{\text{model}}(x|\theta)$ unless we work with very simple models. In LVMs, Jensen's inequality is used to compute a lower

bound to the maximum likelihood objective:

$$\begin{aligned}
\log p_{\text{model}}(x|\theta) &= \log \int p_{\text{forward}}(x|h,\theta)p(h)dh, \\
&= \log \int \frac{p_{\text{forward}}(x|h,\theta)p(h)}{q(h)} q(h)dh, \\
&\geq \mathbb{E}_{q(h)}[\log p_{\text{forward}}(x|h,\theta)] - \text{KL}(q(h)\|p(h)), \qquad (4.2)
\end{aligned}$$

where Equation (4.2) is known as the variational lower bound [4], or ELBO [14], where $h \in \mathbb{R}^K$ denotes the latent variable, and $q(h)$ denotes the variational distribution over the latent variable. In linear LVMs with tree structured latent variables (e.g. mixture models, HMMs), we can use the posterior $p(h|\theta)$ as the variational distribution, because the posterior makes this bound tight [73, 4].

In the general situation where the forward mapping is defined via a non-linear mapping, such that $p_{\text{forward}}(x|h,\theta) = p_{\text{out}}(x; f_\theta(h))$, where $f_\theta(h) : \mathbb{R}^K \to \mathbb{R}^L$ is the nonlinear deterministic mapping, and $p_{\text{out}}(.)$ is the employed noise model, computing the posterior distribution is not analytically tractable in general. VAEs therefore use a neural network mapping for the variational distribution $q_\phi(h) = \mathcal{N}(x; \mu_\phi(x), \sigma^2_\phi(x)I)$, where $\mathcal{N}(.)$ denotes the Normal distribution and the neural network mappings $\mu_\phi(x), \sigma^2_\phi(x) : \mathbb{R}^L \to \mathbb{R}^K$ parametrize the variational distribution.

Although the likelihood computation in VAEs is intractable and require the variational EM algorithm described above, we argue in this chapter that the main failure mode of VAEs is caused by the simplistic prior choices for $p(h)$, as we demonstrate this in the experiments section.

Another popular way to learn generative models is via GANs. GANs are implicit generative models, therefore they do not employ an output distribution p_{out}. Namely, the data generation mechanism is defined as follows:

$$h \sim p^0(h),\ x = f_\theta(h), \qquad (4.3)$$

where we call $p^0(h)$ *the base distribution*, typically chosen as a simplistic distribution such as an isotropic Gaussian distribution, and $f_\theta(h)$ is a deterministic forward mapping similar to what we have denoted for VAEs above. GANs therefore do not employ an output distribution $p_{\text{out}}(.)$, but rather define $p_{\text{model}}(x|\theta)$ via a deterministic transformation of the base distribution $p^0(h)$.

In this chapter, we also argue that one of the reasons why GANs might underperform is because of the simplistic base distribution choice. In addition to this, GANs also complicate

the model parameter optimization by introducing a discriminator network. GANs in their original formulation [16], approximate the ratio between the data density and the model density [72]:

$$\begin{aligned}\mathcal{L}(\theta, \xi) &= \sum_n \log D_\xi(x_n) + \sum_{n'} \log 1 - D_\xi(x'_{n'}) \\ &\rightarrow \sum_n \log \frac{p_{\text{data}}(x_n)}{p_{\text{model}}(x_n|\theta) + p_{\text{data}}(x_n)} \\ &\quad + \sum_{n'} \log \frac{p_{\text{model}}(x'_{n'}|\theta)}{p_{\text{model}}(x'_{n'}|\theta) + p_{\text{data}}(x'_{n'})} \end{aligned} \tag{4.4}$$

where x_n denotes the training instances, and x'_n denotes samples generated from the model. The convergence to the second line (which can be recognized as the Monte Carlo estimate for the Jensen-Shannon divergence) can be easily seen by maximizing the objective $\mathcal{L}(\theta, \xi)$ with respect to the discriminator parameters ξ [16]. The big conceptual problem with GANs is that the optimization step for the generator parameters cause mode collapse. This can be easily seen by examining the corresponding loss function. The original paper suggests the maximization of the objective in Equation 4.5.

$$\begin{aligned}&\max_\theta \sum_{n'} D(x'_{n'}),\ x'_{n'} \sim p_{\text{model}}(x|\theta) \\ \approx &\max_\theta \int p_{\text{model}}(x|\theta) \log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x|\theta)} dx,\end{aligned} \tag{4.5}$$

where we assumed that the discriminator is trained until convergence. We can see that the objective in the last equation has a mode seeking/zero avoiding behavior, similar to $\text{KL}(p_{\text{model}}(x|\theta) \| p_{\text{data}}(x))$ [74]. In practice, therefore the discriminator is not trained until convergence, and there are various heuristics that tries to deal with mode collapse [75].

There exists several other variants of GANs which use other divergences [76], or which are based on approximate optimal transport metrics [77, 78]. Or, some approaches use a GAN ensemble to approximate the whole density [79].

In this chapter, we propose a much simpler approach, which optimizes a maximum likelihood objective using an implicit density model. The optimization does not involve an additional discriminator, and the approach does not suffer from mode collapse since it maximizes a maximum likelihood objective.

We would also like to point out that there is a recent work on generative model learning, which does maximum likelihood for implicit models [80] for certain types of invertible mappings such as convolutions. However, they do not consider general mappings as we do.

In addition to this we advocate using multi-modal distributions in the latent space in this chapter.

4.2 LEARNING IN IMPLICIT GENERATIVE MODELS

We know from probability theory that in an implicit generative model as defined in Equation (4.3), the output probability density is related to the base distribution via the cumulative density function:

$$p_{\text{model}}(x|\theta,\phi) = \frac{\partial}{\partial x}\int_{\{x:f_\theta(h)\leq x\}} p^0_\phi(h)dh, \tag{4.6}$$

where note that the base distribution is parametrized by ϕ. The integral in Equation (4.6) is not tractable in general, however if we have an invertible mapping $f_\theta(h)$, we can obtain an analytical expression for the density function of the model using the following formula [81]:

$$p_{\text{model}}(x|\theta,\phi) = p^0_\phi(f^{-1}_\theta(x))V_\theta(x), \tag{4.7}$$

where $V_\theta(x) := \left|\det \frac{\partial f^{-1}_\theta(x)}{\partial x}\right| = \left|\det \frac{\partial f_\theta(h)}{\partial h}\right|^{-1}$, which measures the volume change due to the transformation. It is possible to construct exactly invertible mappings using typical neural network mappings such as matrix multiplications and convolutions. Constraining the forward mapping to be exactly invertible is restrictive however, mainly because invertibility only holds for transformations which do not change the dimensionality. In section 4.2.2 we describe an algorithm which maximizes the model likelihood for a general mappings for which we also have an approximate inverse.

4.2.1 Maximum Likelihood for Implicit Generative Models

If we work with invertible forward mappings, the optimization problem for maximum likelihood in an implicit generative model is the following:

$$\begin{aligned}&\max_{\theta,\phi}\sum_n \log p_{\text{model}}(x_n|\theta,\phi),\\ =&\max_{\theta,\phi}\sum_n \log p^0_\phi(f^{-1}_\theta(x_n)) + \log V_\theta(x_n),\end{aligned} \tag{4.8}$$

where the first term can be interpreted as maximizing the likelihood of the mappings $f^{-1}_\theta(x)$ in the base distribution space, and the volume term $V_\theta(x)$ ensures that the distribution

properly normalized. If we think about this objective from a sampling perspective, in order to the generate plausible samples, the maximum likelihood objective tries to match the samples from the base distribution with the observations mapped to the base distribution space $f_\theta^{-1}(x)$.

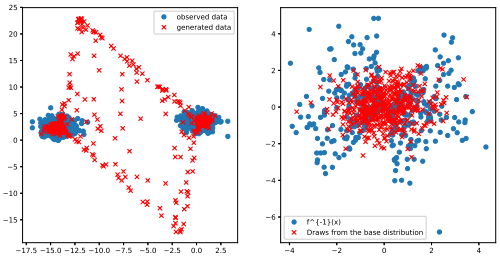
Note that in GANs, only the forward mapping parameters θ is optimized, and the base distribution is fixed to be simple unimodal distribution. Optimizing both the forward mapping parameters θ and a multi-modal base distribution constitutes the main idea in this chapter. We argue that mapping a multimodal dataset onto a unimodal base distribution is harder to achieve than fitting a multimodal distribution on $f_\theta^{-1}(x)$. We demonstrate this in Figure 4.1. Using an invertible linear mapping $f_\theta(h) = Wh$, where $h \in \mathbb{R}^2$, and $W \in \mathbb{R}^{2\times 2}$, we show that on a two dimensional mixture of Gaussians example that, if we do maximum likelihood on the objective in Equation (4.9), we fail to map the observations to the samples drawn from a fixed isotropic base distribution. However, as shown in Figure (b) if we set the base distribution as a flexible distribution such as mixture of Gaussians, and learn its parameters ϕ, we are able to learn a much more accurate distribution. We also show that if we train the same mapping using the standard GAN formulation, we get the mode collapse behavior, where only one of the Gaussians is captured in the learned distribution.

We acknowledge that in the cases where the forward mapping has the same dimensionality in the domain and range spaces (such as the example in Figure 4.1), learning an implicit generative model by maximizing Equation (4.9) is pointless, because we could have very well just fitted a mixture model on the data. For this reason, in the next section we propose the two stage learning algorithm which allows the use of forward mappings which change the dimensionality.

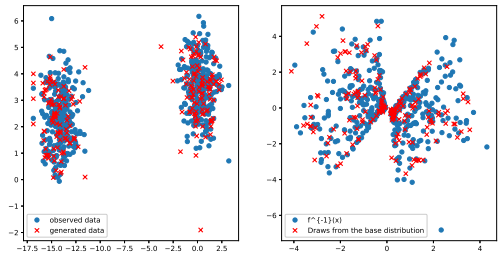
4.2.2 The Two Stage Algorithm

In practice, we typically would like to have base distribution defined on a space which has lower dimensionality than the observation space. If this is the case, then it is impossible to have an exactly invertible mapping $f_\theta(h)$. It is however possible to have an approximately invertible forward mapping. This idea gives the hint for a very simple two stage maximum likelihood algorithm: We first fit an auto-encoder such that the error $\sum_n \|f_\theta(f_\psi^{\text{enc}}(x_n)) - x_n\|$ is minimized. Once the we are done with optimizing the autoencoder, we simply fit a base distribution on the embeddings $f_\psi^{\text{enc}}(x)$. The formal algorithm is specified in Algorithm 4.1.

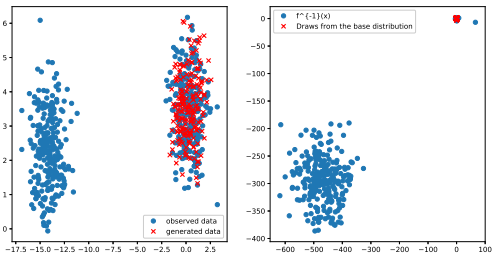
To see that this is a maximum likelihood algorithm, let us reconsider the likelihood function



(a) Using a simple and fixed base distribution



(b) Learning the base distribution



(c) What GAN learns

Figure 4.1: We demonstrate the differences between our proposed method and two methods that use a simple base distributions on a two dimensional mixture of two Gaussians. (Maximizing (4.9) and GAN with a fixed isotropic Gaussian as base distribution). In figure (a) we maximize the implicit model likelihood defined in (4.9) with respect to forward mapping parameters θ. In figure (b) we fit a mixture of Gaussians for the same forward mapping as figure (a). In figure (c) we see what GAN learns for the same dataset. In Each figure, left plot shows the generated data overlaid on the observed dataset, and left plot show the samples from the base distribution overlaid on the observations mapped to the observation space ($f_\theta^{-1}(x)$).

Algorithm 4.1 The two stage implicit generative model learning algorithm

Consider an autoencoder such that $f_\theta(f_\psi^{\text{enc}}(x)) \approx x$.

-Train the auto-encoder parameters θ, ψ such that:

$$\min_{\theta,\psi} \sum_n \|f_\theta(f_\psi^{\text{enc}}(x_n)) - x_n\|$$

-Fit the base distribution on the latent space such that:

$$\max_\phi \sum_n \log p_\phi^0(f_\psi^{enc}(x_n))$$

of the implicit generative model with the autoencoder:

$$= \max_\phi \sum_n \log p_\phi^0(f_\psi^{enc}(x_n)) + \log V(x_n), \tag{4.9}$$

where we easily see that the base distribution parameters ϕ are independent of the volume term $V(x)$. Assuming that that the autoencoder learns a mapping close to the identity, we conclude that maximizing with respect to the base distribution parameters maximizes the model likelihood.

Note that since the optimization for the forward mapping parameters θ, and the base distribution is decoupled, it is easy to fit a multi-modal distribution for the base distribution on the embeddings $f^{\text{enc}}(x)$. One natural choice is to use a mixture distribution. We demonstrate this on handwritten zero and one digits from the MNIST dataset [5] in Figure 4.2. We choose the dimensionality of the latent space $K = 2$ to be able to visualize the base distribution space. We a three component Gaussian mixture model for this example.

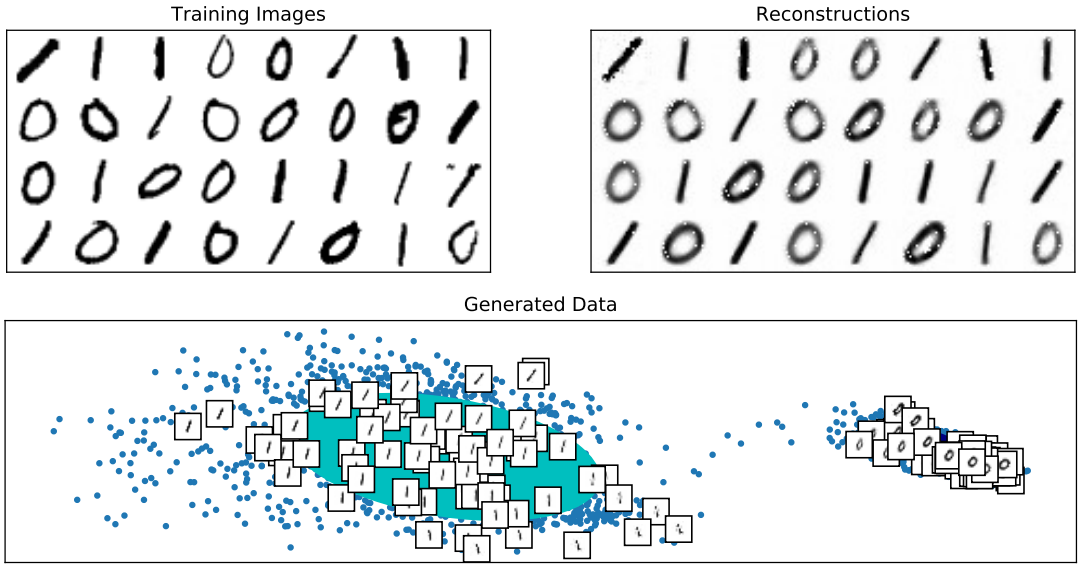


Figure 4.2: Demonstration of the two stage algorithm on a toy dataset with handwritten 0's and 1's. The purpose of this figure is to give a sense on how the proposed algorithm work. (Top row) Samples from the training set, and corresponding reconstructions. (Bottom row) Two dimensional embeddings of the training samples are shown with blue dots. We overlay sampled images. The solid color ellipses show the covariance components of the learned Gaussian mixture model for the base distribution.

4.2.3 Learning Generative Models for Sequential Data

The framework we propose also offers the flexibility to learn distributions over sequences by simply learning a sequential distribution such as HMM on the latent representations. The

likelihood of a sequence is expressed in Equation 4.10.

$$p_{\text{model}}(x_{1:T}|\psi,\phi) = \prod_{t=1}^{T} p_\phi^0(f_\psi^{\text{enc}}(x_t)|f_\psi^{\text{enc}}(x_{1:T-1}))V(x_t), \tag{4.10}$$

where a sequence is denoted as $x_{1:T} := \{x_1, x_2, \dots x_T\}$. and thus $f_\psi^{\text{enc}}(x_{1:T-1}) = \{f_\psi^{\text{enc}}(x_1), f_\psi^{\text{enc}}(x_2), \dots, f_\psi^{\text{enc}}(x_{T-1})\}$. According to this density model, the observations $x_{1:T}$ are mapped to latent space independent from each other. This suggests that we can closely follow the two stage algorithm defined in Algorithm 4.1: Same as before we first fit the autoencoder, and obtain the latent representations. In the second stage, instead of fitting an exchangeable model such as a mixture model, we fit a base distribution which models the temporal structure of the latent space. Potential options for such a distribution include Hidden Markov Models (HMMs), and RNNs, or convolutional models. In our audio experiments, we used HMMs with Gaussian emissions.

4.3 EXPERIMENTS

4.3.1 Images

We learn generative models on the MNIST [5] (hand written digits) and CELEB-A [82] (celebrity faces). We compare our algorithm (which we abbreviate with IML - Implicit Maximum Likelihood), with VAE, standard GAN and Wasserstein GAN. As the main quality metric, we compare likelihoods computed on a test set using kernel density estimator (KDE).

For the MNIST dataset, we use an invertible perceptron in our approach to demonstrate that we can also use our approach to compute model likelihoods on the test set using the implicit generative model density function in Equation (4.7). (Note that in general our framework allows non-invertible mappings: We use a general convolutional autoencoder for the CELEB-A dataset) The invertible perceptron we use for the MNIST dataset is defined as follows:

$$\begin{aligned} h_1 =& \tanh_{\text{invt}}\left(\mathbf{Linear}[K, 600](h)\right), \\ x =& \sigma_{\text{invt}}\left(\mathbf{Linear}[600, 784](h_1)\right), \end{aligned}$$

where $h \in \mathbb{R}^K$ denotes the latent representation, and $\mathbf{Linear}[L_1, L_2](h) = Wh + b$, $W \in \mathbb{R}^{L_2 \times L_1}$, $b \in \mathbb{R}^{L_2}$ represents a linear layer (we follow the pytorch API convention to denote the input and output dimensionalities). The invertible non-linearity functions are denoted

with $\tanh^{\text{invt}}(.)$, and $\sigma^{\text{invt}}(.)$, which respectively stand for invertible tangent-hyperbolic and invertible sigmoid functions. We basically use the original non linearity in the invertible regime, and a linear function in the saturation regimes. Namely, for hyperbolic tangent we have the following function:

$$\tanh_{\text{invt}}(t) = \begin{cases} ct - b & t \leq -1 + \epsilon \\ \tanh(t) & |t| \leq 1 - \epsilon \\ ct + b & t \geq 1 - \epsilon \end{cases}, \tag{4.11}$$

We use $c = 0.01$, and choose the bias term b, and the threshold ϵ so that the function is continuous and smooth (has a continuous first derivative). Similarly, the invertible sigmoid function is defined as follows:

$$\sigma_{\text{invt}}(t) = \begin{cases} ct - b & t \leq \epsilon \\ \sigma(t) & 0 \leq t \leq 1 - \epsilon \\ ct + b & t \geq 1 - \epsilon \end{cases}, \tag{4.12}$$

Note that it is straightforward to derive the inverse functions once the parameters of the non-linearities are set. Therefore the inverse network is defined in Equation 4.13,

$$\begin{aligned} h_1 =& \sigma_{\text{invt}}^{-1}(\mathbf{Linear}^{-1}[784, 600](x)), \\ h =& \tanh_{\text{invt}}^{-1}(\mathbf{Linear}^{-1}[600, K](h_1)), \end{aligned} \tag{4.13}$$

where $\mathbf{Linear}^{-1}[L_2, L_1](x) := (W^\top W)^{-1} W^\top (x - b)$. Note that the parameters W, b are shared for a given forward and inverse Linear layers. To obtain the volume term due to the rectangular transformation, we note that the volume change due to the rectangular linear transformation in a linear layer is given by $\sqrt{\det(W^\top W)}$ [83]. Therefore to the correction term involves dividing the original pdf with this volume change (we note that the implicit model likelihood holds, because the mapping is approximately invertible due to the first step of the algorithm).

To do objective comparisons between models we compute Kernel density estimates (KDE) on the test set: For each batch, we sample 1000 points from the trained models, and represent the learned density as the sum of Kernel functions centered at these samples. We then compute the average score for all the test set. We sample 10 such random batches and report the mean and the standard variance in Table 4.1. We use Gaussian Kernels, with

variance 0.1. The KDE scores we compute for the models are defined in Equation 4.14.

$$\text{KDE score} = \frac{1}{N_{\text{test}} N_{\text{samples}}} \sum_{n=1}^{N_{\text{test}}} \log \sum_{m=1}^{N_{\text{samples}}} \mathcal{N}(x_n^{\text{test}}; x_m^{\text{sample}}, 0.1I). \tag{4.14}$$

Notice that for small kernel bandwidth, the above objective is tantamount to computing the nearest neighbor distance for all test instances. To get high scores from this estimator, the observed samples need to capture the diversity of the test instances. Also note that this estimator is computing an estimate for $\text{KL}(p_{\text{testset}} \| p_{\text{model}})$, so this metric penalizes mode collapse.

In the left panel of Figure 4.3, we compare the KDE scores for our two-stage algorithm, GAN, Wasserstein GAN and VAE on the MNIST dataset. We use the standard training-test split defined in the pytorch data utilities (60000 training instances and 6000 test instances). We try 7 different latent dimensionality K for all algorithms ranging from 20 to 140 with increments of 20. In our algorithm, we use a GMM with 30 full-covariance components for all K values. We see that performance drops with increasing K, however we manage to stay better than VAEs and GANs. The performance drop is expected to happen with increasing K, because the density estimation problem in the latent space gets more difficult with increasing latent dimensionality. We would like to note that it possible to use a more complicated base distribution and compensate.

In the right panel of Figure 4.3, we compare the model likelihood computed with the implicit likelihood equation in (4.7) with the base distribution likelihood (the complete likelihood minus the Jacobian term). The purpose of this is to examine if there is a correlation between these quantities. As we pointed out before, our algorithm does not require an exactly invertible mapping, and as can be seen from the figure the base distribution likelihood is somewhat correlated with the overall model likelihood, and therefore can potentially be used as a proxy for the complete likelihood for mappings for which we don't know how to compute the Jacobian term.

In Figure 4.4, we show the random nearest neighbor samples for randomly selected test instances for all four algorithms in the top panel. We see that IML method is able to capture the diversity of the test instances well. On top of that we see much more definition in the generated images thanks to the multi-modal base distribution that we are using. As we earlier illustrated in Figure 4.1, using a simplistic base distribution causes a mismatch between the mappings to the latent space and the draws from the base distribution. Due to the simplistic distributions used in VAEs, and GANs we see that these approaches tend to generate more samples which do not resemble handwritten digits. We also observe that

quality of the samples (and nearest neighbor samples) are correlated with the KDE metric.

In Figure 4.5, we do the same nearest neighbor sample measurement on the CELEB-A dataset. We have set the latent dimensionality as 100 for all algorithms. We cropped the images using a face detector, and resized them to size 64×64 in RGB space. We used 146209 such images for training, and 10000 images for test. We see that the proposed IML algorithm has more accurate nearest neighbor samples. We see that although the VAE is able to generate less distorted samples than GAN and WGAN, it's generated images contain much more distortion than IML, potentially because of the simplistic latent representation. The generated samples from IML contain much less distortion than GANs.

For all algorithms we used the Adam optimizer [84]. As mentioned before, in the MNIST experiment, for IML we used the invertible network we introduced in this section. For GANs and VAE we used a standard one hidden layer perceptron with exact same sizes. Namely, the decoders maps K dimensions into 600, and 600 dimensions then gets mapped into 784 dimensions (MNIST images are of size 28×28). We use the mirror image encoder for the VAE, that is we map 784 dimensions to 600, and that gets mapped into K dimensional vectors for the mean and variance of the posterior. For the CELEB-A dataset, we used a 5 layer convolutional encoders and decoders (We used the basic DC-GAN [85] generator architecture for all algorithms, with exact same parameter setting - only with the exception that for VAE the latent representations are obtained without passing through ReLU in order not to allow negative values as we use isotropic Gaussian as the prior). For W-GAN would like to point that we used to code published by the authors with the default parameter set-up. For GAN and VAE our code is based on code provided for pytorch examples.

Table 4.1: Best KDE scores on test set for MNIST and CELEB-A datasets using 4 different algorithms. We show the mean (larger is better), and standard deviation of the KDE likelihoods, over 10 batches of random samples consisting of 1000 samples each.

Algorithm	**MNIST**	**CELEB-A**
IML	144.5 ± 0.7	-8340 ± 33
VAE	117.7 ± 2.7	-10493 ± 66
GAN	-7.6 ± 2.3	-11835 ± 71
WGAN	61.2 ± 1.6	-12993 ± 105

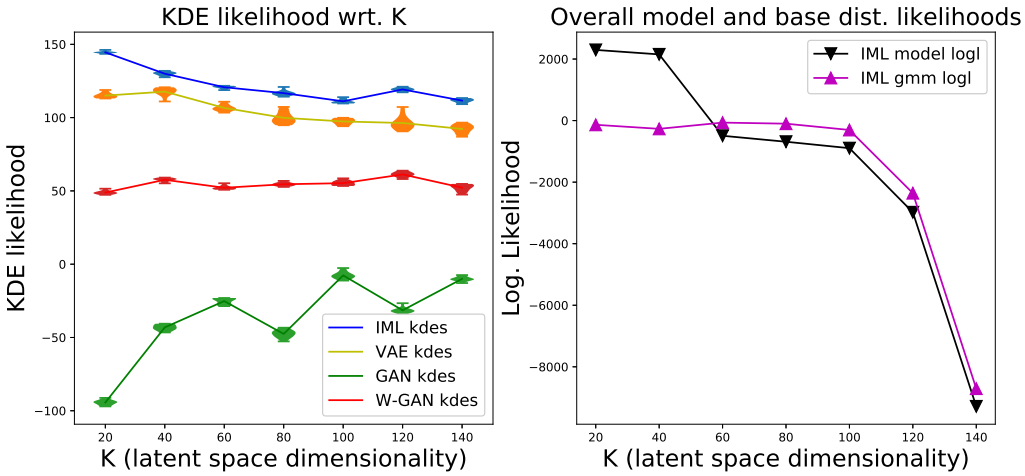


Figure 4.3: KDE likelihood with respect to the dimensionality of the latent space K on the MNIST dataset. **(left)** Violin plots which show the distributions for the KDE scores. **(right)** Computing the model likelihood using the inverse mapping.

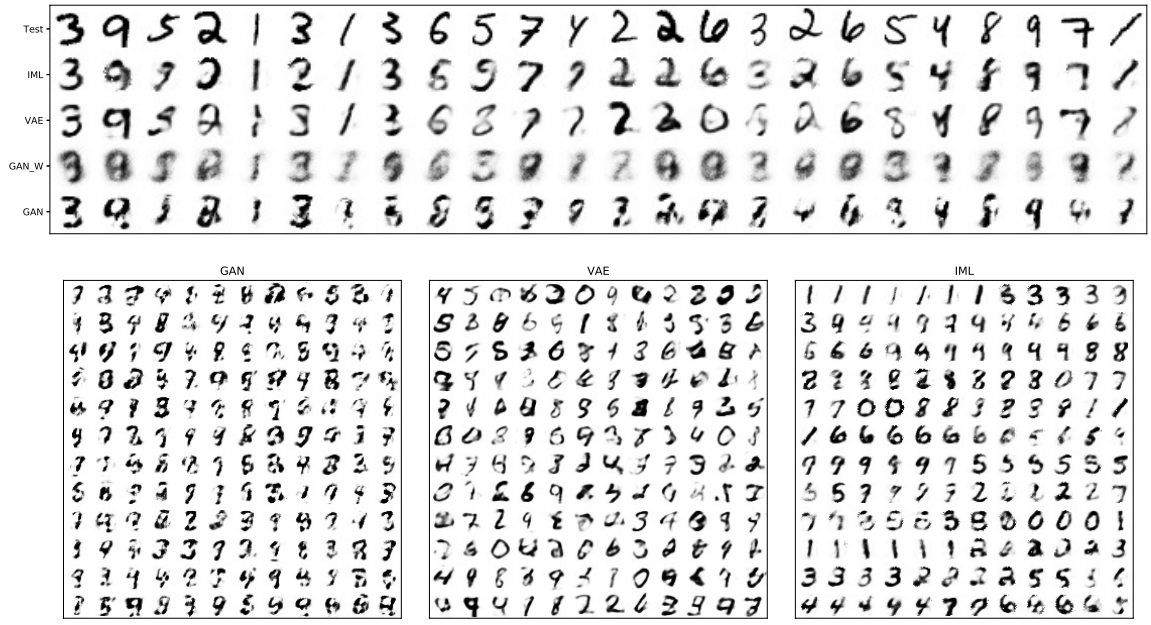


Figure 4.4: Samples from the MNIST dataset. (**top**) Generated nearest neighbor samples (nearest to test instances which are shown on the top row) for four different algorithms. (**bottom-left**) Random images generated with a GAN, (**bottom-middle**) Random images generated with a VAE, (**bottom-right**) Generated Samples with IML, samples from the same cluster are grouped together.

4.3.2 Audio

To show that our algorithm can be used to learn a generative model for sequential data, we experiment with generating speech and music in the waveform domain. In all datasets,

Figure 4.5: Samples from the CELEB-A dataset. (**top**) Generated nearest neighbor samples (nearest to test instances which are shown above) for four different algorithms. (**bottom-left**) Random images generated with GAN, (**bottom-middle**) Random images generated with VAE (**bottom-right**) Random Samples with IML, samples from the same cluster are grouped together.

we work with audio with 8kHz sampling rate. We dissect the audio into 100ms long chunks, where consecutive chunks overlap by 50ms, and each window is multiplied by a Hann window. The autoencoder learns 80 dimensional latent representations for each chunk which is 800 samples long. We use three layer convolutional networks both in the encoder and decoder, where we use filters of length 200 samples.

We fit an HMM to the extracted latent representations. We use 300 HMM states, where each state has a diagonal covariance Gaussian emission model. The random samples are obtained by sampling from the fitted HMMs, and passing the sampled latent representation through the decoder. To reconstruct the generated chunks as an audio waveform, we follow the overlap-add procedure [86]: We overlap the each generated by chunk by 50 percent and add.

As a speech experiment, we learn a generative model over digit utterances. We work with the free spoken digit dataset [87]. As the training sequence, we give the model a concatenated set of digit utterances. We consider the cases where the training data only contains one digit type, and the case where the training data contains all digits. In Figure 4.6, we show the spectrograms of generated digit utterances (this example contained all 10 digit types - we

used 1000 utterances for training) along with spectrograms of the training digit utterances. Note that the generated digit utterances are generated in sequences (We generate one long sequence which contains multiple digits). In figure 4.8 we show three cases for the one-digit only training task. We see that we are able to learn a generative model over one digit with a some variety.

As the music experiment, we train a model on a 2 minute long violin piece. We downloaded the audio file for the violin etude in `https://www.youtube.com/watch?v=0uSI6t54KWY`. We show the spectrogram of the first 10 seconds of the piece and our generated sequence in Figure 4.7. We see from the spectrogram that the model is able to learn some musical structure, although there is additional background artifacts. The generated samples for the spoken digit utterances and the generated music sequence can be downloaded and listened from the following link: `https://www.dropbox.com/sh/6mvzf9ca1wl3uej/AAAkBTdNBumU61_mnMu7epDla?dl=0` (we suggest copy and pasting the link, and watching for spaces, also we suggest opening the files with vlc player if your native player does not work)

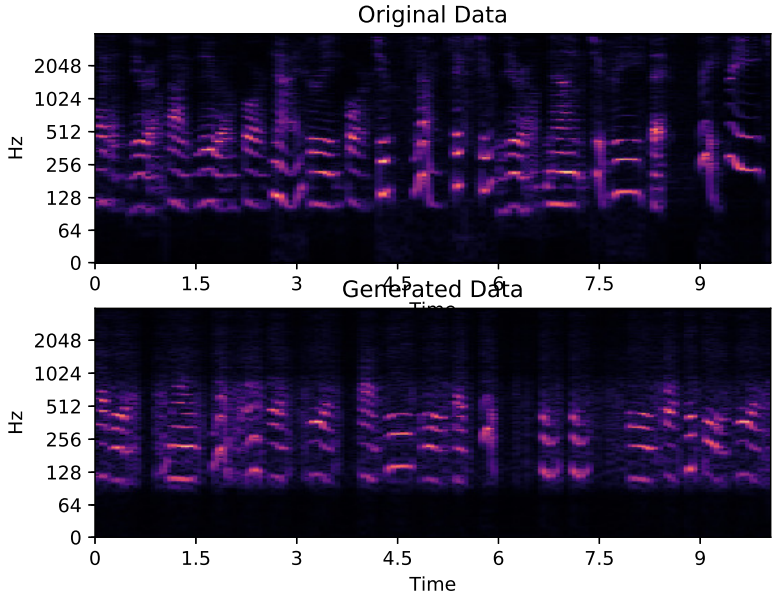


Figure 4.6: We illustrate the spectrograms for generated digits. Top figure contains the spectrogram for the true digit utterances, and below figure contains the spectrogram of the generated utterance.

More Samples from CELEB-A

We show more random samples in Figures 4.9, and 4.10 with IML, VAE, GAN, and Wasserstein-GAN.

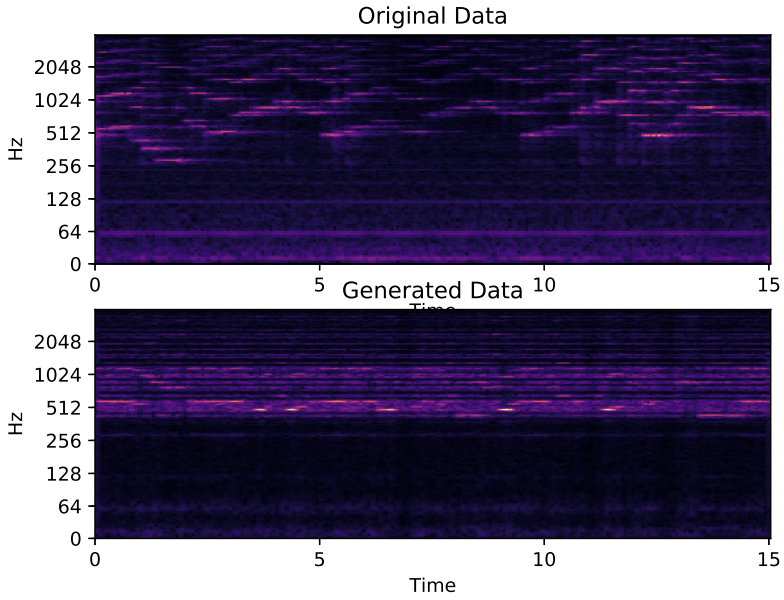


Figure 4.7: Excerpt from the spectrogram of the generated sequence learned from the violin etude.

4.3.3 Discussion

The algorithm we propose is very simple and effective. It is also principled in the sense that it performs maximum likelihood. We would like to emphasize that, compared the GANs the performance is much less sensitive to the network design choices and training parameters such as the learning rate. In author's experience, GANs are extremely sensitive to training parameters such as the learning rate. We have observed that decoupling the training of the base distribution from the neural network mapping makes the training much easier: In our approach it suffices to pick a small enough learning rate so that the encoder converges, and successfully embeds the data in a lower dimensional space.

In our experience, VAE's seem to be easier to train (much less susceptible to hyperparameter choices). However, as we have seen in the results and figures, the simplistic choice for the base distribution results in distorted outputs. In our experiments we have used relatively more standard models to model the latent distribution, but it is possible to use complex methods such as Dirichlet Process Mixture models to obtain complicated base distributions.

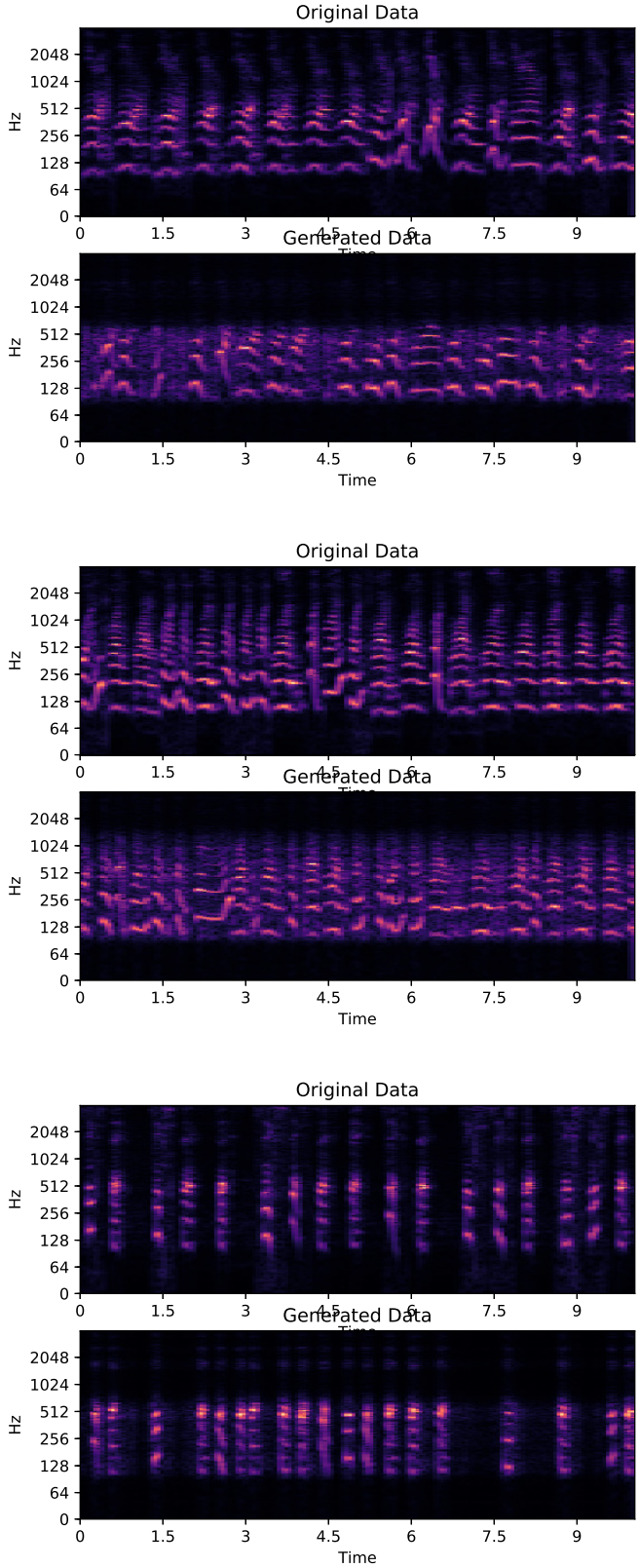


Figure 4.8: Spectrogram of the generated utterance sequence for digit 0 **(top)**, digit 1 **(middle)**, digit 6 **(bottom)**.

Figure 4.9: More Random Samples with IML **(left)**, and VAE **(right)** on CELEB-A dataset.

Figure 4.10: More Random Samples with GAN **(left)**, and Wasserstein-GAN **(right)** on CELEB-A dataset.

CHAPTER 5: GENERATIVE SEQUENCE MODELING IN AUDIO

Generative models are commonplace in audio modeling. Non-negative matrix factorization (NMF) [88], for instance is a very popular approach which models spectrogram columns as a linear combination of latent frequency templates. Or, recurrent neural networks have been used in symbolic audio modeling [89], to learn distributions over MIDI data.

In this chapter, we discuss three main contributions. First, we experimentally show that using Generative Adversarial Networks [16] in audio source separation provides increase in source separation performance over maximum likelihoods methods such as NMF or autoencoders, in a speech source separation task. Our corresponding publication is [90].

Second, we argue that using convolutional neural networks for audio source separation in spectrogram domain greatly enhances the source separation in speech-speech source separation over the base line methods such as Non-Negative Matrix Factorization [88], and autoencoders [91] which do not model the temporal structure in spectrograms. The proposed models are analogs of the convolutive NMF model [40], which use templates which are defined over both time and frequency. Our corresponding publication is [41].

Third, we experimentally show that on four symbolic music datasets, using recurrent neural networks with diagonal recurrent matrices improves the performance of the model. Our corresponding paper is [44].

5.1 USING GANS IN GENERATIVE SOURCE SEPARATION

Many popular audio modeling/source separation algorithms such as Non-Negative Matrix Factorization (NMF) [88], autoencoders [92], or tensor factorization models [93] are all generative models and they are trained with maximum likelihood (ML) which require the specification of output distributions, as showcased in the previous section. For instance, NMF models with different loss functions (e.g. KL-NMF, Euclidean NMF, IS-NMF) [94], actually have the same underlying mapping from latent space to observed space (same underlying network), but their performances typically differ on a given dataset. The output distribution/loss function therefore biases the model.

Generative Adversarial Networks (GANs) [16] offer a generative model learning framework, which does not require the specification of an output distribution. GANs are able to learn the processes which are implicitly defined via a transformation of a random variable. Namely, the generative process is defined such that a random latent variable is mapped to the data domain via getting transformed through a deterministic neural network. This removes the

bias that comes from assuming a parametric output distribution and leads to more accurate modeling of distributions. [95]

GANs have been very popular in computer vision since their first introduction [95]. However, to the best of our knowledge, usage of GANs in the audio modeling domain has been limited. In [96], authors train de-noising networks by using an adversarial framework. In this section, we propose using GANs to learn a generative model over magnitude spectrogram frames, which are used in a speech source separation task.

Source separation is the task where the goal is to decompose a given signal into additive components which approximates the original sources as accurately as possible. In generative source separation, we train generative models to recover the sources from an observed mixture. We experimentally show on speech mixtures that an adversarially trained two layer perceptron outperforms NMF and ML-trained autoencoders in terms of source-to-distortion ratio [70]. In our experiments, we have observed that the original GAN formulation in [16] is hard to train. We therefore showed the performance improvement over standard audio models with the more recent Wasserstein-GAN formulation [77].

5.1.1 Generative Supervised Source Separation

In generative source separation, an observed mixture signal $x \in \mathbb{R}^L$ is assumed to follow the generative process below:

$$\begin{aligned} h_1 \sim p_{\text{latent}}(h_1),\ s_1|h_1 &\sim p_{\text{forward}_1}(s_1|h_1) \\ h_2 \sim p_{\text{latent}}(h_2),\ s_2|h_2 &\sim p_{\text{forward}_2}(s_2|h_2) \\ x|s_1, s_2 &\sim p_{\text{mixture}}(x|s_1 + s_2) \end{aligned}$$

where the source $s_k \in \mathbb{R}^L$, $k \in \{1, 2\}$, follow the distribution,

$$p_{\text{source}_k}(s_k) = \int p_{\text{forward}}(s_k|h_k)p(h_k)dh_k,$$

where $h_k \in \mathbb{R}^K$ is a latent variable with lower dimensionality, such that $K < L$, and $p_{\text{forward}}(s_k|h_k)$ is the forward model for the sources. Given the sources, the mixture x is assumed to be distributed according to the conditional distribution $p_{\text{mixture}}(x|s_1+s_2)$, where x is conditioned on the sum of the sources. Note that we have not yet assumed parametric forms for the distributions above. Also note that, in our experiments we consider the case where there are only two sources, although methods discussed can be generalized to more sources. In the context of our audio application, mixture x here corresponds to a column of

a magnitude spectrogram.

The goal in source separation is to compute accurate estimates for the sources given a mixture signal x. In supervised generative source separation, the approach is to first train the forward models $p_{\text{forward}_1}(.)$, $p_{\text{forward}_2}(.)$ such that the source distributions $p_{\text{source}_1}(.)$, $p_{\text{source}_2}(.)$ are approximated as accurately as possible. Given the trained models for both sources, in testing we compute source estimates $\widehat{s}_1$, $\widehat{s}_2$ such that the conditional likelihood $p_{\text{mixture}}(x|\widehat{s}_1 + \widehat{s}_2)$ is maximized (or equivalently the reconstruction error for the mixture is minimized). In the next section, we describe the specifics on how to go through supervised source separation with maximum likelihood training.

5.1.2 Maximum Likelihood Training for Sources

A common way to go about approximating the source distributions is through assuming that the sources are generated by transforming a K dimensional latent variable $h \sim p_{\text{latent}}(h)$, through a non-linear mapping (such as a neural network) $f_\theta(h)$ with parameters θ, and adding noise to the transformed variable. This corresponds to the following generative model:

$$h \sim p_{\text{latent}}(h),\ s|h \sim p_{\text{out}}(s; f_\theta(h)), \tag{5.1}$$

where $p_{\text{out}}(.)$ is the output distribution which models the noise at the output of the mapping $f_\theta(h)$. E.g. When modeling spectrograms, $p_{\text{out}}(.)$ is usually taken as Poisson distribution, which corresponds to the unnormalized KL divergence. Under these modeling assumptions, the optimization problem for approximating source distribution is written as follows:

$$\max_\theta \sum_t \log \int \mathcal{PO}(s^t; f_\theta(h)) p(h) dh, \tag{5.2}$$

where the integral over the latent variable h is intractable in the general case. Using variational auto-encoder framework in [15], the objective in expression (5.2) can be maximized by computing a variational lower bound.

In practice however, especially in audio modeling, the integral over the latent variable h is not computed, and only the conditional forward model $p_{\text{forward}}(s|h)$ is learnt, by simultaneously optimizing over the forward model parameters and the latent variables. This is written as the following optimization problem:

$$\max_{\theta,h} \sum_t \log \mathcal{PO}(s^t; f_\theta(h)), \tag{5.3}$$

If $f_\theta(h) = Wh$, where $W, h \geq 0$, then this formulation corresponds to the widely used Non-Negative Matrix Factorization (NMF) model [88, 97, 94]. It is also possible to include the latent variable estimation part in the model with an auto-encoder. This results in the following optimization problem:

$$\max_{\theta, \theta^{\text{enc}}} \sum_t \log \mathcal{PO}(s^t; f_\theta(f^{\text{enc}}_{\theta^{\text{enc}}}(s^t))), \tag{5.4}$$

where $f^{\text{enc}}_{\theta^{\text{enc}}}(.) : \mathbb{R}^L \to \mathbb{R}^K$, is the encoder, and $f_\theta(.) : \mathbb{R}^K \to \mathbb{R}^L$ is decoder part. In [92], $f_\theta(h) = \log(\exp(W_1 h) + 1)$, and $f^{\text{enc}}_{\theta^{\text{enc}}}(s) = \log(\exp(W_2 s) + 1)$ is used.

The conceptual problem with the training objectives discussed in this section is that by picking a specific output distribution, we are sacrificing from the generality of the approximated source distributions. To remove this assumption, in this section we use generative adversarial networks, which is a neural network framework for learning generative models without explicitly specifying an output distribution when training the generator network.

5.1.3 Adversarial Training for Sources

We have seen in the previous section that maximum likelihood training involves a parametric assumption for the output distribution. As an alternative, in this section we propose using an implicit generative model in training, which does not require an explicit loss function. An implicit generative model for the sources can be specified as follows:

$$h \sim p_{\text{latent}}(h), \; s = f_\theta(h), \tag{5.5}$$

where the source s is deterministically related to the latent variable h, unlike the source models in the previous section. This process implies an intractable density function $p_{\text{model}}(.)$ for s in the general case where $f_\theta(h)$ is a complicated non-linear mapping such as a neural network. Learning under implicit generative models is a currently a very active field of research [72]. One way to attack this problem is to use discriminator function $D_\xi(.)$ which aims to distinguish between the samples generated from the model and the training instances. The goal in training then becomes to generate samples using the process in expression (5.5) so that, the discriminator $D_\xi(.)$ becomes unable to distinguish between the generated samples and the training data. This described setup is known as a generative adversarial network (GAN) [16], and the corresponding minimax game is specified as follows:

$$\min_\theta \max_\xi \mathbb{E}_s \log D_\xi(s) + \mathbb{E}_h \log(1 - D_\xi(f_\theta(h))), \tag{5.6}$$

This expression can be recognized as the sum of Bernoulli log-likelihoods, where $D_\xi(.)$ tries to maximize by outputting 1 for the training data s, and outputting 0 for the generated samples $f_\theta(h)$. The generator however tries to minimize the expression by *fooling* the discriminator. It can be shown that under some assumptions this scheme minimizes the Jensen-Shannon divergence between the actual source distribution $p_{\text{source}}(.)$ and the model distribution $p_{\text{model}}(.)$. However in practice, this scheme is unstable, and usually suffers from the mode collapse problem where the learnt distribution $p_{\text{model}}(.)$ only captures a subset of the actual sample space. [95]. This is unfortunately not acceptable for our source separation application.

An alternate formulation known as the Wasserstein-GAN, alleviates the mode collapse problem by minimizing the Wasserstein-1 distance between the learnt and data distributions, which results in smooth gradients [77]. Authors show that this can be achieved with following minimax game:

$$\min_{\theta} \max_{\xi \in \mathcal{W}} \mathbb{E}_s D_\xi(s) - \mathbb{E}_h D_\xi(f_\theta(h)), \tag{5.7}$$

where $\mathcal{W}$ denotes the set for parameters ξ for which $D_\xi(.)$ will be γ-Lipschitz continuous, for some γ. In the algorithm provided in the paper, this constraint is achieved by clipping the weights ξ. In our experiments, Wasserstein GANs showed significant improvement over the original GAN formulation. Note that $D_\xi(.)$ is referred to as critic in this formulation.

5.1.4 Testing

After training the forward models for each source, given a test mixture, the estimates $\widehat{s}_1$, $\widehat{s}_2$ for the sources is obtained by minimizing the reconstruction error via finding the optimal latent variables as inputs to the forward mappings:

$$\widehat{h}_1,\ \widehat{h}_2 \ = \ \arg\max_{h_1, h_2} \log p_{\text{mixture}}(x; f_{\widehat{\theta}_1}(h_1) + f_{\widehat{\theta}_2}(h_2)), \tag{5.8}$$

and then we get the estimates for the sources by setting $\widehat{s}_1,\ \widehat{s}_2 = f_{\widehat{\theta}_1}(\widehat{h}_1),\ f_{\widehat{\theta}_2}(\widehat{h}_2)$, where $\widehat{\theta}_1$, $\widehat{\theta}_2$ denote the trained network parameters.

An extra benefit we get by training our generative models with GANs is that, in addition to the generator networks $f_{\widehat{\theta}_k}(.)$, we also get discriminators/critics $D_{\widehat{\xi}_k}(.)$. We can therefore use them in the separation stage to score how much the obtained source looks like the instances in training set. We also noticed that using a smoothing term to enforce smooth first difference across time improves the quality of the estimated sources for both GANs and Maximum likelihood based auto-encoders. Therefore, the optimization for separating the

sources, given T mixture spectrogram columns $x^{1:T}$, becomes the following:

$$\begin{aligned}\max_{h_1^{1:T},h_2^{1:T}} \frac{1}{T}\sum_{t=1}^{T}\log p_{\text{mixture}}(x^t; f_{\widehat{\theta}_1}(h_1^t)+f_{\widehat{\theta}_2}(h_2^t)) \\ + \frac{\alpha}{T}\sum_{t=1}^{T}\Big(D_{\widehat{\xi}_1}(f_{\widehat{\theta}_1}(h_1^t))+D_{\widehat{\xi}_2}(f_{\widehat{\theta}_2}(h_2^t))\Big) \\ + \frac{\beta}{T-1}\sum_{t=1}^{T-1}\left(\sum_{k=1}^{2}\|f_{\widehat{\theta}_k}(h_k^{t+1})-f_{\widehat{\theta}_k}(h_k^t)\|_1\right),\end{aligned} \tag{5.9}$$

where α is a trade-off scalar between the reconstruction quality and discriminator/critic score. In our experiments we fixed $\alpha = 0.1$, but it can also potentially be optimized on a validation set. For the smoothing term, we used $\beta = 0.1$. Finally, note that for magnitude spectrograms it is very common to use Poisson distribution for $p_{\text{mixture}}(.)$.

5.1.5 Empirical Results

To show the validity of using GANs in source separation, we compare adversarially trained networks with auto-encoders trained with maximum likelihood, variational auto-encoders and NMF.

The experiment set-up is as follows: We form mixtures of male and female speaker utterances and corresponding training data from the TIMIT speech corpus [98]. To form the training/test data pairs, we randomly pick male and female speaker pairs from the *train* folder of the TIMIT corpus. Each speaker has 10 available utterances. For both speakers, of the 10 available utterances, we use 9 for training and 1 for testing. The resulting training set for each source is around 30 seconds long. The selected test utterances are around 3 seconds, and the mixture signal is obtained by mixing the test utterances at 0 dB. We form 25 such mixtures/training sets and test each algorithm on these randomly selected sets (The speaker pairs are the same across algorithms). As the preprocessing step, we compute Fourier spectrograms of the utterances. We use 1024 point FFT, and a hop size of 256 samples. The learning and source separation are performed on the columns (Fourier magnitude vectors for each time window) of the magnitude spectrograms. When reconstructing the separated sources in the time domain, we use the Wiener filtering equation:

$$\widehat{s}_k^{\text{time}} = \text{ISTFT}\left(\frac{\widehat{s}_k}{\widehat{s}_1+\widehat{s}_2}\odot x \odot x^{\text{phase}}\right), \text{ for } k\in\{1,2\}, \tag{5.10}$$

where x and x^{phase} are respectively the magnitude and phase spectrograms of the mixture.

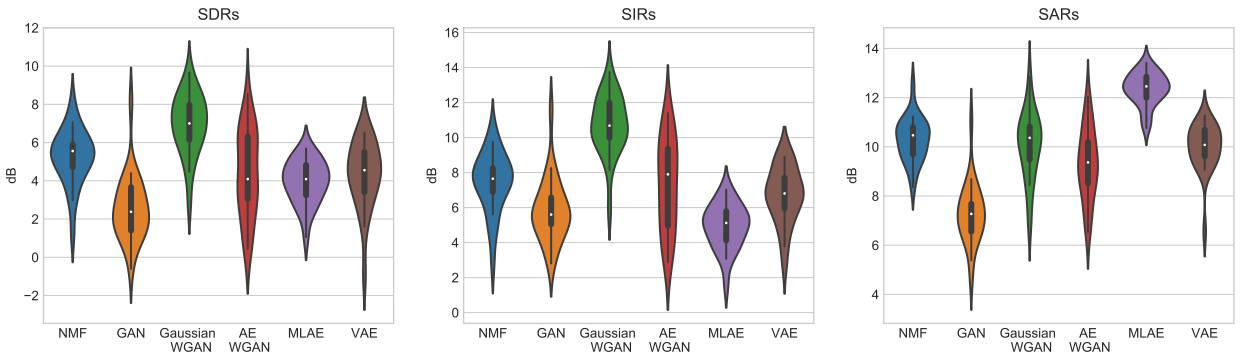


Figure 5.1: The distributions of the BSS eval scores for our speech source separation experiment. Acronyms for each algorithm are indicated below each violin plot. In order from left to right the algorithms are ordered as: NMF, Standard GAN, Wasserstein GAN with Gaussian inputs, Autoencoding Wasserstein GAN, Autoencoder trained with maximum likelihood, and Variational Autoencoder. Each violin shows the distribution of the corresponding score in dB over different speaker pairs. The subplots are organized as follows: **Left:** SDR scores, **Middle:** SIR scores, **Right:** SAR scores.

The magnitude spectra for the estimated sources are denoted by $\widehat{s}_1$ and $\widehat{s}_2$. The estimated time domain signal is denoted by $\widehat{s}_k^{\text{time}}$. The division and the multiplication $\odot$ are both element wise, and ISTFT(.) designates the inverse short time Fourier transform operation to get the time domain signal from a complex Fourier spectrogram. We obtain results with the following models:

- KL-NMF model.
- The auto-encoder model suggested in [92], trained with maximum likelihood using a Poisson likelihood (equivalently unnormalized KL divergence).
- Standard GAN with Gaussian random inputs.
- Wasserstein GAN with Gaussian random inputs.
- Autoencoding Wasserstein GAN, where instead of Gaussian random inputs, we feed the training samples to the generator network.
- Variational Autoencoder with Gaussian prior on the latent variable, as in [15], and Poisson likelihood at the output.

For all GANs, we used the following architecture for the generator:

$$f_\theta(h) = \text{SP}(W_2\text{SP}(W_1 h)), \tag{5.11}$$

where SP(.) is the soft-plus nonlinearity, such that $\text{SP}(x) = \log(\exp(x) + 1)$. Note that we have omitted the bias terms from the equation to reduce clutter. For all GANs with

Gaussian random inputs, we used 513 dimensional inputs h (This is the dimensionality of the data items since we use 1024 point fft), and 100 hidden units. Therefore W_1 was of size 100×513, and W_2 was of size 513×100. For the auto-encoding GAN, and the auto-encoder trained with maximum likelihood, the network architecture of generator/forward model are exactly the same, except that the inputs are the data items s, instead of random variable h. For the VAE, we used the encoder $f^{\text{enc}}_{\theta^{\text{enc}}}(s) = W_2\text{ReLU}(W_1 s)$, both for the mean and the variance terms of the latent variable, where W_1 was of size 100×513, and W_2 was of size 20×100. For the decoder of the VAE, we used $f_\theta(h) = \text{SP}(W_3 h)$, where W_3 was of size 513×20.

For the discriminator/critic networks of GANs, we use the following architecture:

$$D_\xi(s) = \sigma(V_2 \tanh(V_1 s)), \tag{5.12}$$

where V_1 is of size 90×513 (we use 90 hidden units for the discriminator), and V_2 is of size 1×90. In standard GANs, $\sigma(.)$ is the sigmoid function. In Wasserstein GAN, we do not use a non-linearity at the end of the network, and therefore $\sigma(.)$ is the identity function. This gives smoother gradients.

In training and testing, for all neural network models we use the RMSprop algorithm [99] with a learning rate of 0.001. During the training of GANs, we do 5 iterations of discriminator/critic updates per generator update. For all neural network models, we do 4000 training iterations, and 20000 test iterations. For Wasserstein-GAN we clip the critic parameters at -0.01 and 0.01 for lower and upper limits respectively.

We report the BSS-eval [70] scores obtained after recovering the sources from the mixture signals. The BSS-eval scores are Source to Distortion Ratio (SDR), Source to Interference Ratio (SIR), and Source to Artifacts Ratio (SAR), where SDR being the summary measure on how good the separation is. For each speaker pair, we have averaged the BSS-eval scores of the recovered sources, and in Figure 5.1, with violin plots we show the distribution of the averages of the two BSS-eval scores over all speaker pairs.

Experiments indicate that, The Wasserstein GAN with a Gaussian noise input outperforms NMF, ML auto-encoder and Variational auto-encoder in terms of source to distortion ratio. Note that we are obtaining these results with very similar underlying networks. For all models except VAE, we have kept the exact generator architecture defined in Equation (5.11). We have also observed that the standard GAN formulation is not very reliable. Although occasionally we have seen good SDRs with it, we have observed through inspecting its outputs that it is not able to capture the variety in the source spectrogram distribution as good as the Wasserstein GAN, and therefore the source separation performance

of the standard GAN is not as good. We have also experimented with training an auto-encoder with adversarial training, and have seen that although it is less reliable than the Wasserstein GAN with Gaussian inputs, it is sometimes able to give great SDRs. In general, adversarial methods give great SIRs, by losing a bit from SAR, especially compared to the ML-autoencoder. Finally, note that the code for our experiments is available at `https://github.com/ycemsubakan/sourceseparation_misc`.

5.1.6 Conclusions

We have experimentally shown that Wasserstein GANs can obtain good performance in generative source separation. In addition to not requiring the specification of an output distribution, GANs fit into the source separation task nicely since the discriminator/critic functions help in source separation. We believe that there exists many research opportunities to use GANs in the audio domain. One natural next step from the results in this section is to extend the results shown in this section with an end-to-end generative adversarial audio model.

5.2 CONVOLUTIVE NEURAL NETWORK MODELS FOR AUDIO SOURCE SEPARATION

As we have discussed in the previous section, non-negative matrix factorization (NMF) for magnitude-spectrograms is a very popular method for modeling sources for supervised source separation applications [88, 97, 100]. However NMF based methods, e.g. the methods we have discussed in the previous section, ignore the temporal structure between columns of a given spectrogram. In this section we develop two neural network models which explicitly model the temporal structure of audio spectrograms.

NMF factorizes a matrix of non-negative elements $X \in \mathbb{R}_{\geq 0}^{L \times N}$ as a product of the basis matrix $W \in \mathbb{R}_{\geq 0}^{L \times K}$ and the activation matrix $H \in \mathbb{R}_{\geq 0}^{K \times N}$. In the case of audio signals, NMF is applied on audio spectrograms, where the columns of W act as representative basis vectors for the source. The rows of H indicate the activity of these basis vectors in time.

As shown in [91], it is possible to construct autoencoder analogs for NMF as follows:

$$\begin{aligned} &\text{Encoder: } H = f^{\text{enc}}(X) = g(W^{\ddagger}X) \\ &\text{Decoder: } X = f(H) = g(WH), \end{aligned} \tag{5.13}$$

where, X represents the input spectrogram, $W^{\ddagger}$ represents an approximate-inverse of W and

$g(.) : R \rightarrow R^{\geq 0}$ is an element-wise function that maps a real number to the space of positive real numbers, (such as softplus or relu), and $f^{\text{enc}}(.) : \mathbb{R}^L \rightarrow \mathbb{R}^K$, $f(.) : \mathbb{R}^K \rightarrow \mathbb{R}^L$ respectively denote the encoder and the decoder. As before, the columns of W act as representative basis vectors and the corresponding rows of H indicate their respective activations. Additionally, this interpretation enables a pathway to propose variants to this basic autoencoder stucture by exploiting the wealth of available neural net architectures that could potentially lead to superior separation performance. Namely, by exploiting the flexibility of neural networks, we extend the our proposed convolutive architecture to a recurrent architecture.

Spectrograms of speech and audio signals incorporate temporal dependencies that span multiple time frames. However, NMF and its neural network equivalent are unable to explicitly utilize these cross-frame patterns available in a spectrogram. In [40] a convolutive version to NMF (conv-NMF) has been proposed, which allows spectro-temporal patterns as representative basis elements. In this section, we develop neural network alternatives to such convolutive NMF for supervised source separation.

Several neural network architectures have been recently proposed for supervised source separation [101, 102, 103], where the networks are discriminatively trained. In other words, these networks operate directly on the mixtures and separate them into individual sources. Although discriminative training of a source separation network results in good seperation performance, the network is restricted to work on a particular type of mixture. The convolutional architecture that we discuss is generatively trained on the magnitude spectrograms of clean utterances. Therefore, these networks are not restricted to the types of mixtures used in training. By the virtue of flexibility of neural networks, we also propose a variant where recurrent neural network is used.

5.2.1 Non-negative Convolutional Auto-encoders

The convolutive NMF model in [40] approximates a non-negative matrix $X \in \mathbb{R}_{\geq 0}^{L \times N}$ as,

$$X(f,t) \approx \sum_{i=1}^{K} \sum_{k=0}^{T-1} W_i(k,f) H(i,t-k), \tag{5.14}$$

where, $W_i \in \mathbb{R}_{\geq 0}^{L \times T}$ acts as the i^{th} spectro-temporal basis matrix out of K such matrices and $H \in \mathbb{R}_{\geq 0}^{K \times N}$ contains the corresponding weights. The notation $X(i,j)$ represents the element of X indexed by the i^{th} row and the j^{th} column. We can interpret this operation as

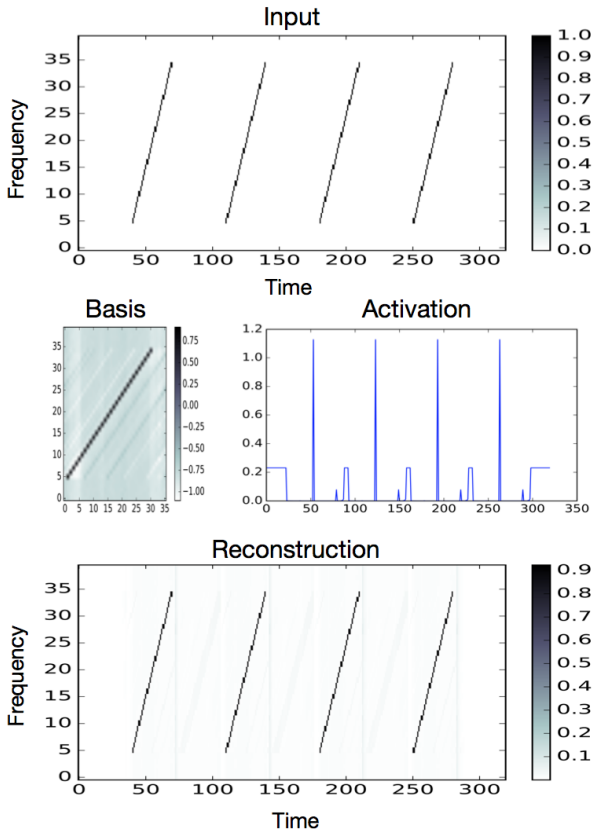


Figure 5.2: Basis decomposition of a toy-illustration obtained using a CNN-CNN auto-encoder.

a two-layer convolutional auto-encoder as follows,

$$\text{Encoder layer: } H(i,t) = f^{\text{enc}}(X)(i,t) = \sum_{j=0}^{M-1}\sum_{k=0}^{T-1} W_i^{\ddagger}(j,k)X(j,t-k)$$

$$\text{Decoder layer: } \hat{X}(f,t) = f(H)(f,t) = \sum_{i=1}^{K}\sum_{k=0}^{T-1} W_i(k,f)H(i,t-k) \tag{5.15}$$

subject to non-negativity of W_i and H. Here, we assume that the convolutional layer filters W, $W^{\ddagger}$ have a size of $L \times T$ where, T represents the depth of the convolution (filter length) and L denotes the height of the input matrix X (number of frequency bins). In this representation, W_i and H correspond to the i^{th} basis matrix and the activation matrix respectively. The filters of the first convolutional neural network (CNN) act as inverse filters in defining the auto-encoder. We refer to this auto-encoder as the CNN-CNN auto-encoder (CCAE). We can satisfy the non-negativity constraints by incorporating a non-linearity into the defi-

nitions of the encoder and the decoder. Thus,

$$\text{Encoder layer: } H(i,t) = f^{\text{enc}}(X)(i,t) = g\left(\sum_{j=0}^{M-1}\sum_{k=0}^{T-1} W_i^{\ddagger}(j,k)X(j,t-k)\right)$$
$$\text{Decoder layer: } \hat{X}(f,t) = f(H)(f,t) = g\left(\sum_{i=1}^{K}\sum_{k=0}^{T-1} W_i(k,f)H(i,t-k)\right) \tag{5.16}$$

where, the $g(.) : \mathbb{R} \rightarrow \mathbb{R}^{\geq 0}$ applies an element-wise non-linearity and ensures that the activation matrix and the reconstruction are non-negative. The block diagram of the whole CCAE is given in Figure 5.3. In our experiments, we used the soft-plus function which is given by the formula $g(x) = \log(1 + \exp(x))$ as the non-linearity. Using (5.16), we now note some key points about the CCAE. (i) The output of the encoder gives the latent representation (analog of activations in NMF) of the decomposition. (ii) The filters of the decoder act as the spectro-temporal bases of the decomposition. (iii) We do not explicitly apply non-negativity constraints on the bases (decoder filters). Thus, the basis matrices can assume negative values. To train the auto-encoder, we minimize the KL-divergence between the input spectrogram X and its reconstruction $\hat{X}$ given by,

$$D\left(X, \hat{X}\right) = \sum_{i,j} X(i,j)\log\frac{X(i,j)}{\hat{X}(i,j)} - X(i,j) + \hat{X}(i,j) \tag{5.17}$$

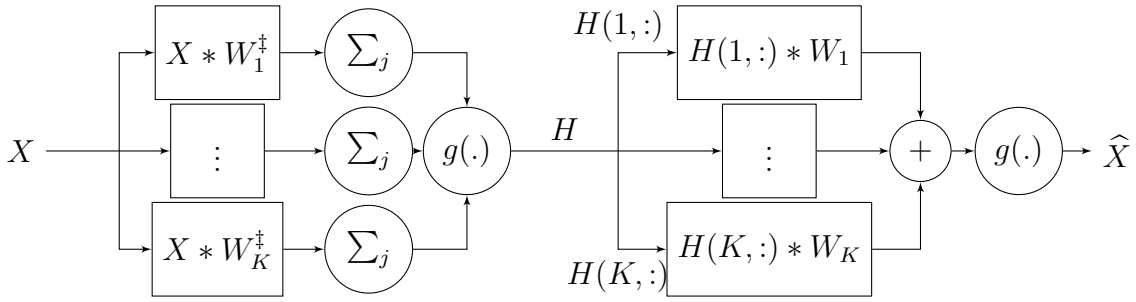


Figure 5.3: Block Diagram of CNN-CNN Autoencoder.

Although the filters can assume negative values, the use of a non-linearity does not allow cross-cancellations across the basis elements.

5.2.2 Practical Considerations

Having developed the CCAE equivalent to conv-NMF, we can now begin to understand the nature of the bases and activations learned by the network. To do so, we train the

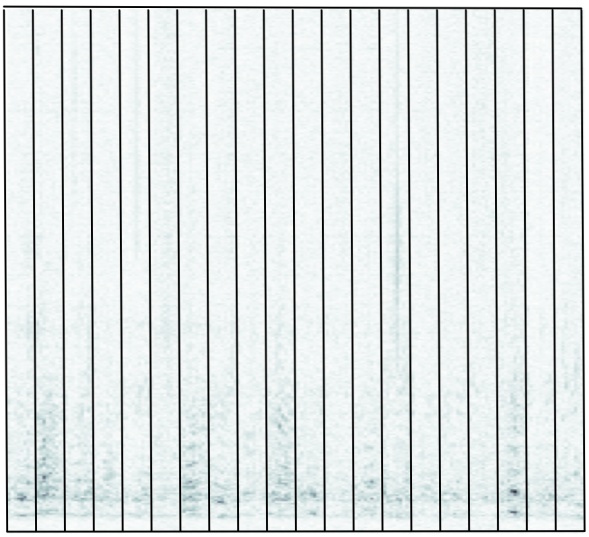


Figure 5.4: A subset of decoder filters obtained by training the CAE on magnitude-spectrograms of utterances of a male speaker. This decomposition is obtained for the configuration $r = 80$ and $T = 8$. We see that the filters resemble snippets of a speech spectrogram.

CCAE defined by (5.16) on a simple toy example as shown in figure 5.2. The input is a spectrogram-like image that consists of a repeating pattern of diagonal structures and has a size of 40×350 pixels. We use this spectrogram to train a CCAE with filters of size 40×36. We also incorporate sparsity constraints on the activation, i.e., the output of the first CNN. As shown in the figure, the basis of the decomposition learned by the decoder CNN resembles a snippet of the input spectrogram. As expected, the decoder filters take negative values unlike conv-NMF bases. We also see that the activation comprises a series of impulse trains. Thus, the encoder acts as a matched-filter and identifies the points in time when the corresponding pattern becomes active. As shown in (5.16), the time-frequency pattern is captured by the filters of the decoder. Given the nature of the activation, we see that the encoder attempts to learn the inverse filter to the decoder. Figure 5.4 shows the decoder filters obtained by training the CAE on speech utterances of a male speaker. Similar to the previous toy-example, the decoder filters learn patterns that resemble snippets of a speech spectrogram.

Note that the encoder attempts to approximate the inverse of the decoder. From our knowledge of linear filtering in signal processing, we know that the inverse of a finite length filter is given by a recursive filter[1] [104]. Following this analogy, in the next section we

[1] `https://ccrma.stanford.edu/~jos/fp/Inverse\_Filters.html`

explore using a recurrent neural network in the encoder.

5.2.3 Using a recurrent filter in the encoder

In this section, the goal is to construct a recurrent encoder analogous to the convolutional encoder we discussed in the previous section. We will refer to this auto-encoder as a Recurrent-Convolutional Auto-encoder (RCAE). The potential gain of using a recurrent encoder over a finite length convolutional encoder is due to the fact that a recurrent filter in theory can capture arbitrarily long temporal dependencies. From a signal processing point of view the motivation for going with a recurrent filter is that, the inverse of an finite length filter (the convolutive basis in the encoder) is given by a recurrent filter.

The way we go about building an encoder is by passing the input through K separate recurrent neural networks (RNNs) (Note that K was the number of filters/components in the convolutive model). The k'th RNN recursion in the encoder is given by the following equation:

$$Z(k_1,t,k) = \tanh\left(\sum_{k_2=1}^{K_{in}} W^{\ddagger^k}(k_1,k_2)Z(k_2,t-1,k) + \sum_{l} U^{\ddagger^k}(k_1,l)X(l,t)\right),\ k\in\{1,\dots,K\}, \tag{5.18}$$

where $Z(:,t,k) \in \mathbb{R}^{K_{in}}$ denotes the latent state vector of the k'th RNN at time t. We denote the hidden state dimensionality of each RNN with K_{in}. The recurrent and projection matrices of the k^{th} RNN are respectively denoted with $W^{\ddagger^k}$, and $U^{\ddagger^k}$. Note that although the given recursion corresponds to the vanilla-RNN architecture, there is no restriction on the RNN architecture choice. In our experiments, we have used the LSTM architecture [9, 43]. After going through the RNN recursions, the encoder output $H(i,t)$ is obtained by summing the RNN outputs over the first dimension:

$$H(i,t) = \sum_{k_1=1}^{K_{in}} Z(k_1,t,i) \tag{5.19}$$

The recurrent encoder's block diagram is given in Figure 5.5.

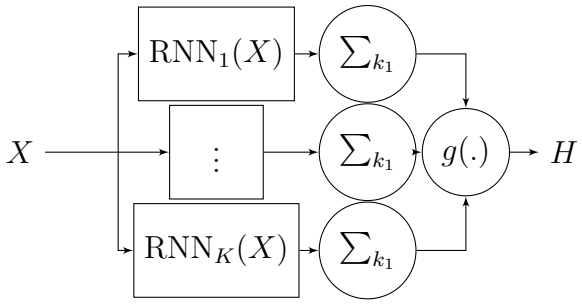


Figure 5.5: Block Diagram of RNN Encoder.

5.2.4 Overview on Solving the Supervised Source Separation Problem

The problem of supervised source separation is solved as a two-step procedure [105]. The first step of the procedure is to learn suitable models for a given source. We refer to this step as the training step. In the second step, we use these models to explain the contribution of the source in a test mixture. In sections 5.2.1 and 5.2.3, we have developed the auto-encoder architecture to learn suitable convolutive models for a given source. We now turn our attention to the problem of using the models for separating the source in an unknown mixture.

Given an input spectrogram X, the auto-encoder produces an approximation of the input spectrogram which is a linear combination of its weights. We will denote to this approximation as,

$$\hat{X} = Ae(X|\theta), \tag{5.20}$$

where, θ denotes the weights (parameters) of the auto-encoder. For the separation procedure, given the trained auto-encoders, (i.e., given θ_1 and θ_2), the goal is to identify suitable input spectrograms X_1 and X_2 such that,

$$X_m = Ae(X_1|\theta_1) + Ae(X_2|\theta_2) \tag{5.21}$$

In this equation, X_m represents the spectrogram of the mixture and X_1, X_2 denote the separated source spectrograms. Thus, similar to NMF, this approach assumes that the magnitude-spectrogram of the mixture is the sum of magnitude-spectrograms of the underlying sources. However, in this separation procedure, we directly estimate the source magnitude-spectrograms without estimating the latent representation. To do so, we optimize for the best inputs X_1, X_2, in Equation (5.21) instead of training for the weights of the network. As before, we minimize the KL divergence between mixture spectrogram X_m and its approximation $(X_1 + X_2)$.

Having obtained the contributions of the sources (separated spectrograms), the next step is to transform these spectrograms back into the time domain. This is given as,

$$x_i(t) = \text{STFT}^{-1}\left(\frac{X_i}{\sum_i X_i} \odot X_m \odot e^{i\Phi_m}\right) \text{ for } i \in \{1, 2\} \tag{5.22}$$

Here $x_i(t)$ denotes the separated speech signal in time and Φ_m represents the phase of the mixture and STFT^{-1} is the inverse short-time Fourier transform operation that transforms the complex spectrogram into its corresponding time domain representation. Also, $\odot$ represents the element-wise multiplication operation and the division is also element-wise.

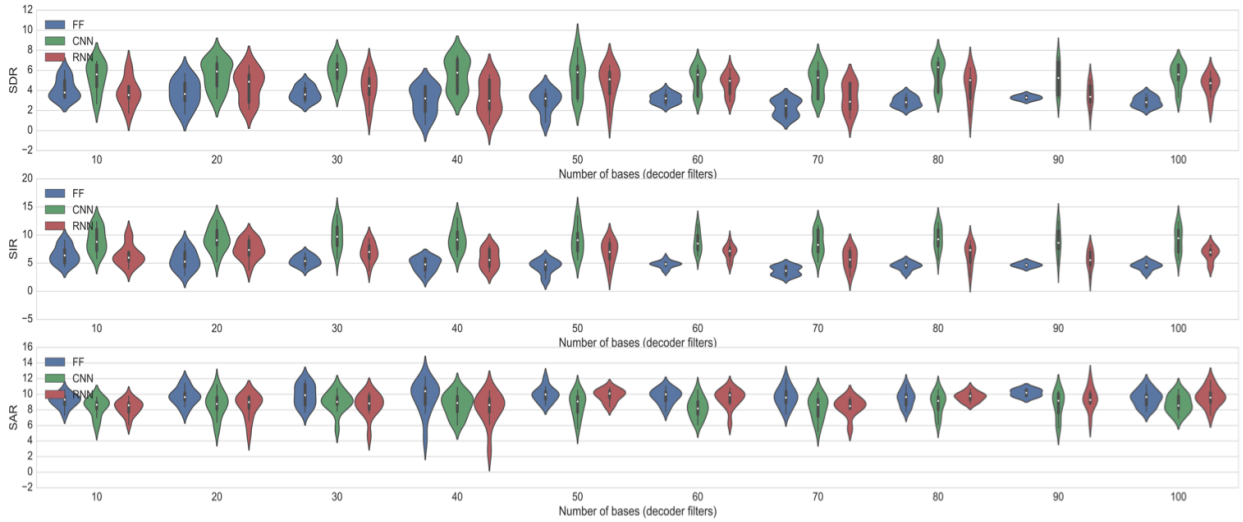


Figure 5.6: Separation performance of convolutive models obtained using FF (blue), CAE (green) and RCAE (red) for varying numbers of bases (K). We compare these models to feed-forward auto-encoder based models (left) in [91]. The legend indicates the encoder of the corresponding architectures.

5.2.5 Experiments

We now describe the experimental setup used to evaluate our auto-encoder based convolutive audio models. We construct a set of training and test examples using the TIMIT corpus [98] for the evaluation. To form the examples, we randomly select a pair of male-female speakers from the TIMIT corpus. Of the 10 utterances available for each speaker, one utterance is randomly selected for each speaker. These two selected utterances are mixed at 0 dB to generate the testing mixture. The remaining 9 utterances are used as training data to construct models for the sources. In other words, these examples are used to train

the auto-encoders. For the evaluation, we generate 20 such mixtures and compare the models for different parameter configurations. As a pre-processing step, we apply a 1024 point short-time Fourier transform representation with a hop of 25%. The magnitude spectrogram is then given as an input to the network.

The networks are trained by applying a batch gradient descent training procedure and the parameters updated using the RMSProp algorithm [99], with a learning rate and momentum of 0.001 and 0.7 respectively. The neural networks are initialized using the Xavier initialization scheme [106].

The CNN filters are selected to be 512-point tall and 8-point wide. Thus, the convolutions are performed only along the time axis. The number of CNN filters also decides the number of components in the decomposition. We evaluate these models over a varying number of CNN filters ranging from 10 to 100 uniformly in steps of 10. We compare the separation performance in terms of median BSS_eval metrics [70] i.e., signal-to-distortion (SDR), signal-to-interference (SIR) and signal-to-artifact ratio (SAR) parameters. The code for our experiments can be downloaded from `https://github.com/ycemsubakan/sourceseparation_nn`.

Figure 5.6 gives the separation performance for the CCAE models (center) and RCAE models (right) for varying values of number of filters K. We evaluate the proposed models by comparing the separation performance to their equivalent feed-forward (FF) counterparts (left) proposed in [91]. We plot the results in terms of a violin plot. The white dots at the center denote the median value and the thick line denotes the inter-quartile range for each K.

We see that the CCAE models significantly out-perform their corresponding FF versions. This can be seen from the fact that the inter-quartile range in SDR for CAE models is higher than the inter-quartile range of corresponding FF models for several values of K. This improvement is a consequence of reduced interference generated by these models in source separation (as seen in the SIR plots). Although the SAR for CCAE models degrades slightly as compared to FF models, the significant improvement in SIR compensates for this loss. The convolutive speech models obtained by training the RCAE also outperform the FF auto-encoder models significantly, as seen by the median values and inter-quartile ranges. This improvement in performance is not as pronounced as the CCAE models. However, the experimentation serves as a definite proof of concept that exploring non-uniform auto-encoder architectures could lead to interesting and potentially powerful algorithms for other datasets and applications.

The performance of the convolutive models achieves a peak value for $K = 80$. However, the median separation performance does not degrade significantly for other values of K.

Thus, the choice of K does not appear to be a very critical consideration for auto-encoder based convolutive models unlike FF models. At the same time, the variance in SDR seems to be dependent on K. We observe that the variance in SDR is considerably lower for higher values of K ($K \geq 50$) as seen by the separation results for both the convolutive models. For $K \geq 80$, we note that the variance continues to be relatively small even though the spread of the violin plot is large, as shown by the inter-quartile ranges. This implies that the violin plots spread out due to the effect of a few outlier values. In other words, auto-encoder based convolutive audio models produce superior results consistently for a high number of CNN filters.

5.2.6 Conclusions

We have developed and investigated the use of a auto-encoders to learn convolutive basis decompositions of speech and audio signals. The ability of the networks to include temporal dependencies allow the auto-encoders to learn cross-frame structures in the input spectrogram. We demonstrated that this results in a significant improvement in separation performance as compared to feed-forward auto-encoder models. This approach also allows for several extensions and generalizations to convolutive audio models, that can be easily implemented using the modeling flexibility that comes with neural networks. One such extension considered in this section is the use of auto-encoders formed by a cascade of recurrent and convolutional layers. Although these models have not outperformed the CAE models for separation of speech mixtures, we have shown that these models are significantly superior to feed-forward auto-encoder models.

5.3 IMPROVING RECURRENT NEURAL NETWORKS ON SYMBOLIC MUSIC MODELING

During the recent resurgence of neural networks in 2010s, Recurrent Neural Networks (RNNs) have been utilized in a variety of sequence learning applications with great success. Examples include language modeling [107], machine translation [25], handwriting recognition [108], speech recognition [109], and symbolic music modeling [89].

In this section, we empirically show that in symbolic music modeling, using a diagonal recurrent matrix in RNNs results in significant improvement in terms of convergence speed and test likelihood. Although diagonal RNNs were used in control literature in the past [110, 111], to the best of our knowledge this is the first work to use them in a machine learning task. We are aware of unitary RNNs [112], which use a unitary recurrent matrix

by decomposing it into unitary matrices. We do not constrain the recurrent matrix to be unitary but only diagonal.

One interpretation of the diagonal recurrent matrix idea is via multivariate Gaussian Mixture Models: In Gaussian mixture modeling (or Gaussian models in general) it is known that using a diagonal covariance matrix often results in better generalization performance, increased numerical stability and reduced computational complexity [113, 114]. We adapt this idea to RNNs by using diagonal recurrent matrices.

We investigate the consequences of using diagonal recurrent matrices for the vanilla RNNs, and for more popular Long Short Term Memory Networks (LSTMs) [9, 43] and Gated Recurrent Units (GRUs) [10]. We empirically observe that using diagonal recurrent matrices results in an improvement in convergence speed in training and the resulting test likelihood for all three models, on four standard symbolic music modeling datasets.

5.3.1 Recurrent Neural Networks

The vanilla RNN (VRNN) recursion is defined as follows:

$$h_t = \sigma_1(W h_{t-1} + U x_t + b), \tag{5.23}$$

where $h_t \in \mathbb{R}^K$ is the hidden state vector with K hidden units, and $x_t \in \mathbb{R}^L$ is the input vector at time t (which has length L). The $U \in \mathbb{R}^{K \times L}$ is the input matrix that transforms the input x_t from an L to K dimensional space and $W \in \mathbb{R}^{K \times K}$ is the recurrent matrix (factor) that transforms the previous state. Finally, $b \in \mathbb{R}^K$ is the bias vector. Note that, in practice this recursion is either followed by an output stage on top of h_t to get the outputs as $y_t = \sigma_2(V h_t) \in \mathbb{R}^L$, or another recursion to obtain a multi-layer recurrent neural network. The hidden layer non-linearity $\sigma_1(.)$ is usually chosen as hyperbolic tangent. The choice of the output non-linearity $\sigma_2(.)$ is dependent on the application, and is typically softmax or sigmoid function.

Despite its simplicity, RNN in its original form above is usually not preferred in practice due to the well known gradient vanishing problem [42]. People often use the more involved architectures such as LSTMs and GRUs, which alleviate the vanishing gradient issue using gates which filter the information flow to enable the modeling of long-term dependencies.

5.3.2 LSTM and GRU

The GRU Network is defined as follows:

$$\begin{aligned} f_t =& \sigma(W_f h_{t-1} + U_f x_t), \\ w_t =& \sigma(W_w h_{t-1} + U_w x_t), \\ c_t =& \tanh(W(h_{t-1} \odot w_t) + U x_t), \\ h_t =& h_{t-1} \odot f_t + (1 - f_t) \odot c_t, \end{aligned} \tag{5.24}$$

where $\odot$ denotes element-wise (Hadamard) product, $\sigma(.)$ is the sigmoid function, $f_t \in \mathbb{R}^K$ is the forget gate, and $w_t \in \mathbb{R}^K$ is the write gate: If f_t is a zeros vector, the current state h_t depends solely on the candidate vector c_t. On the other extreme where f_t is a ones vector, the state h_{t-1} is carried over unchanged to h_t. Similarly, w_t determines how much h_{t-1} contributes to the candidate state c_t. Notice that if w_t is a ones vector and f_t is a zeros vector, the GRU architecture reduces to the VRNN architecture in Equation (5.23). Finally, note that we have omitted the biases in the equations for f_t, w_t, and c_t to reduce the notation clutter. We will omit the bias terms also in the rest of this section.

The LSTM Network is very much related to the GRU network above. In addition to the gates in GRU, there is the output gate o_t to control the output of the RNN, and the forget gate is decoupled into gates f_t and w_t, which blend the previous state and the candidate state c_t:

$$\begin{aligned} f_t =& \sigma(W_f h_{t-1} + U_f x_t), \\ w_t =& \sigma(W_w h_{t-1} + U_w x_t), \\ o_t =& \sigma(W_o h_{t-1} + U_o x_t), \\ c_t =& \tanh(W h_{t-1} + U x_t), \\ h'_t =& h'_{t-1} \odot f_t + w_t \odot c_t, \\ h_t =& o_t \odot \tanh(h'_t), \end{aligned} \tag{5.25}$$

Also notice the application of the tangent hyperbolic on h'_t before yielding the output. This prevents the output from assuming values with too large magnitudes. In [115] it is experimentally shown that this output non-linearity is crucial for the LSTM performance.

5.3.3 Diagonal RNNs

We define the Diagonal RNN as an RNN with diagonal recurrent matrices. The simplest case is obtained via the modification of the VRNN. After the modification, the VRNN recursion becomes the following:

$$h_t = \sigma_1(W \odot h_{t-1} + Ux_t), \tag{5.26}$$

where this time the recurrent term W is a length K vector, instead of a $K \times K$ matrix. Note that element wise multiplying the previous state h_{t-1} with the W vector is equivalent to having a matrix-vector multiplication $W_{diag}h_{t-1}$ where W_{diag} is a diagonal matrix, with diagonal entries set to the W vector, and hence the name for Diagonal RNNs. For the more involved GRU and LSTM architectures, we also modify the recurrent matrices of the gates. This results in the following network architecture for GRU:

$$\begin{aligned} f_t &= \sigma(W_f \odot h_{t-1} + U_f x_t), \\ w_t &= \sigma(W_w \odot h_{t-1} + U_w x_t), \\ c_t &= \tanh(W \odot h_{t-1} \odot w_t + Ux_t), \\ h_t &= h_{t-1} \odot f_t + (1 - f_t) \odot c_t, \end{aligned} \tag{5.27}$$

where $W_f, W_w, W \in \mathbb{R}^K$. Similarly for LSTM, we obtain the following:

$$\begin{aligned} f_t &= \sigma(W_f \odot h_{t-1} + U_f x_t), \\ w_t &= \sigma(W_w \odot h_{t-1} + U_w x_t), \\ o_t &= \sigma(W_o \odot h_{t-1} + U_o x_t), \\ c_t &= \tanh(W \odot h_{t-1} + Ux_t), \\ h'_t &= h'_{t-1} \odot f_t + w_t \odot c_t, \\ h_t &= o_t \odot \tanh(h'_t), \end{aligned} \tag{5.28}$$

where again $W_f, W_w, W_o, W \in \mathbb{R}^K$. One more thing to note is that the total number of trainable parameters in this model scales as $\mathcal{O}(K)$ and not $\mathcal{O}(K^2)$ like the regular full architectures, which implies lower memory and computation requirements.

5.3.4 Intuition on Diagonal RNNs

In order to gain some insight on how diagonal RNNs differ from regular full RNNs functionally, let us unroll the VRNN recursion in Equation 5.23:

$$\begin{aligned} h_t &= \sigma(W\sigma(Wh_{t-2} + Ux_{t-1}) + Ux_t) \\ &= \sigma(W\sigma(W\sigma(Wh_{t-3} + Ux_{t-2}) + Ux_{t-1}) + Ux_t) \\ &= \sigma(W\sigma(W\sigma(\dots W\sigma(Wh_0 + Ux_1) + \dots) + Ux_{t-1}) + Ux_t) \end{aligned} \tag{5.29}$$

So, we see that the RNN recursion forms a mapping from $x_{1:t} = (x_1, \dots, x_{t-1}, x_t)$ to h_t. That is, the state h_t is a function of all past inputs and the current input. To get an intuition on how the recurrent matrix W interacts with the inputs $x_{1:t}$ functionally, we can temporarily ignore the $\sigma(.)$ non-linearities:

$$\begin{aligned} h_t &= W^t h_0 + W^{t-1}Ux_1 + W^{t-2}Ux_2 + \dots + Ux_t \\ &= W^t h_0 + \sum_{k=1}^{t} W^{t-k}Ux_k. \end{aligned} \tag{5.30}$$

Although this equation sacrifices from generality, it gives a notion on how the W matrix effects the overall transformation: After the input transformation via the U matrix, the inputs are further transformed via multiple application of W matrices: The exponentiated W matrices act as "weights" on the inputs. Now, the question is, why are the weights applied via W are the way they are? The input transformations via U are sensible since we want to project our inputs to a K dimensional space. But the transformations via recurrent weights W are rather arbitrary as there are multiple plausible forms for W.

We can now see that a straightforward alternative to the RNN recursion in equation (5.23) is considering linear transformations via diagonal, scalar and constant alternatives for the recurrent matrix W, similar to the different cases for Gaussian covariance matrices [114]. In this section, we explore the diagonal alternative to the full W matrices.

One last thing to note is that using a diagonal matrix does not completely eliminate the ability of the neural network to model inter-dimensional correlations since the projection matrix U gets applied on each input x_t, and furthermore, most networks typically has a dense output layer.

5.3.5 Experiments

We trained VRNNs, LSTMs and GRUs with full and diagonal recurrent matrices on the symbolic midi music datasets. We downloaded the datasets from `http://www-etud.iro.umontreal.ca/~boulanni/icml2012` which are originally used in the paper [89]. The learning goal is to predict the next frame in a given sequence using the past frames. All datasets are divided into training, test, and validation sets. The performance is measured by the per-frame negative log-likelihood on the sequences in the test set.

The datasets are ordered in increasing size as, JSB Chorales, Piano-Midi, Nottingham and MuseData. We did not apply any transposition to center the datasets around a key center, as this is an optional preprocessing as indicated in [89]. We used the provided piano roll sequences provided in the aforementioned url, and converted them into binary masks where the entry is one if there is a note played in the corresponding pitch and time. We also eliminated the pitch bins for which there is no activity in a given dataset. Due to the large size of our experiments, we limited the maximum sequence length to be 200 (we split the sequences longer than 200 into sequences of length 200 at maximum) to take advantage of GPU parallelization, as we have noticed that this operation does not alter the results significantly.

We randomly sampled 60 hyper-parameter configurations for each model in each dataset, and for each optimizer. We report the test accuracies for the top 6 configurations, ranked according to their performance on the validation set. For each random hyper-parameter configuration, we trained the given model for 300 iterations. We did these experiments for two different optimizers. Overall, we have 6 different models (VRNN full, VRNN diagonal, LSTM full, LSTM diagonal, GRU full, GRU diagonal), and 4 different datasets, and 2 different optimizers, so this means that we obtained $6 \times 4 \times 2 \times 60 = 2880$ training runs, 300 iterations each. We trained our models on Nvidia Tesla K80 GPUs.

As optimizers, we used the Adam optimizer [84] with the default parameters as specified in the corresponding paper, and RMSprop [99]. We used a sigmoid output layer for all models. We used mild dropout in accordance with [116] with keep probability 0.9 on the input and output of all layers. We used Xavier initialization [106] for all cases. The sampled hyper-parameters and corresponding ranges are as follows:

- Number of hidden layers: Uniform Samples from {2,3}.
- Number of hidden units per hidden layer: Uniform Samples from {50,...,300} for LSTM, and uniform samples from {50,...,350} for GRU, and uniform samples from {50,...,400} for VRNN.

- Learning rate: Log-uniform samples from the range $[10^{-4}, 10^{-2}]$.
- Momentum (For RMS-Prop): Uniform samples from the range $[0, 1]$.

As noted in the aforementioned url, we used the per-frame negative log-likelihood measure to evaluate our models. The negative log-likelihood is essentially the cross-entropy between our predictions and the ground truth. Per frame negative log-likelihood is given by the expression in Equation 5.31.

$$\text{Per Frame Negative Log-Likelihood} = -\frac{1}{T}\sum_{t=1}^{T} y_t \log \hat{y}_t + (1 - y_t)\log(1 - \hat{y}_t), \qquad (5.31)$$

where y_t is the ground truth for the predicted frames and $\hat{y}_t$ is the output of our neural network, and T is the number of time steps (frames) in a given sequence.

In Figures 5.7, 5.8, 5.9, and 5.10 we show the training iterations vs negative test log-likelihoods for top 6 hyperparameter configurations on JSB Chorales, Piano-midi, Nottingham and MuseData datasets, respectively. That is, we show the negative log-likelihoods obtained on the test set with respect to the training iterations, for top 6 hyper-parameter configurations ranked on the validation set according to the performance attained at the last iteration. The top rows show the training iterations for the Adam optimizer and the bottom rows show the iterations for the RMSprop optimizer. The curves show the negative log-likelihood averaged over the top 6 configurations, where cyan curves are for full model and black curves are for diagonal models. We use violin plots, which show the distribution of the test negative log-likelihoods of the top 6 configurations. We also show the average number of parameters used in the models corresponding to top 6 configurations in the legends of the figures. The minimum negative log-likelihood values obtained with each model using Adam and RMSprop optimizers are summarized in Table 5.1.

We implemented all models in Tensorflow [117], and our code can be downloaded from our github page `https://github.com/ycemsubakan/diagonal_rnns`. All of the results presented in this section are reproducible with the provided code.

5.3.6 Conclusions

- We see that using diagonal recurrent matrices results in an improvement in test likelihoods in almost all cases we have explored in this section. The benefits are extremely pronounced with the Adam optimizer, but with RMSprop optimizer we also see improvements in training speed and the final test likelihoods. The fact that this modification results in an improvement for three different models and two different optimizers

Table 5.1: Minimum Negative Log-Likelihoods on Test Data (Lower is better) with Adam and RMSProp optimizers. **F** stands for Full models and **D** stands for Diagonal models.

Dataset/Optimizer	**RNN-F**	**RNN-D**	**LSTM-F**	**LSTM-D**	**GRU-F**	**GRU-D**
JSB Chorales/Adam	8.91	**8.12**	8.56	8.23	8.64	8.21
Piano-Midi/Adam	7.74	**7.53**	8.83	7.59	8.28	7.54
Nottingham/Adam	**3.57**	3.69	3.90	3.74	**3.57**	3.61
MuseData/Adam	7.82	7.26	8.96	**7.08**	7.52	7.20
JSB Chorales/RMSprop	8.72	8.22	8.51	**8.14**	8.53	8.22
Piano-Midi/RMSprop	7.65	7.51	7.84	7.49	7.62	**7.48**
Nottingham/RMSprop	**3.40**	3.67	3.54	3.65	3.45	3.62
MuseData/RMSprop	7.14	7.23	7.20	7.09	7.11	**6.96**

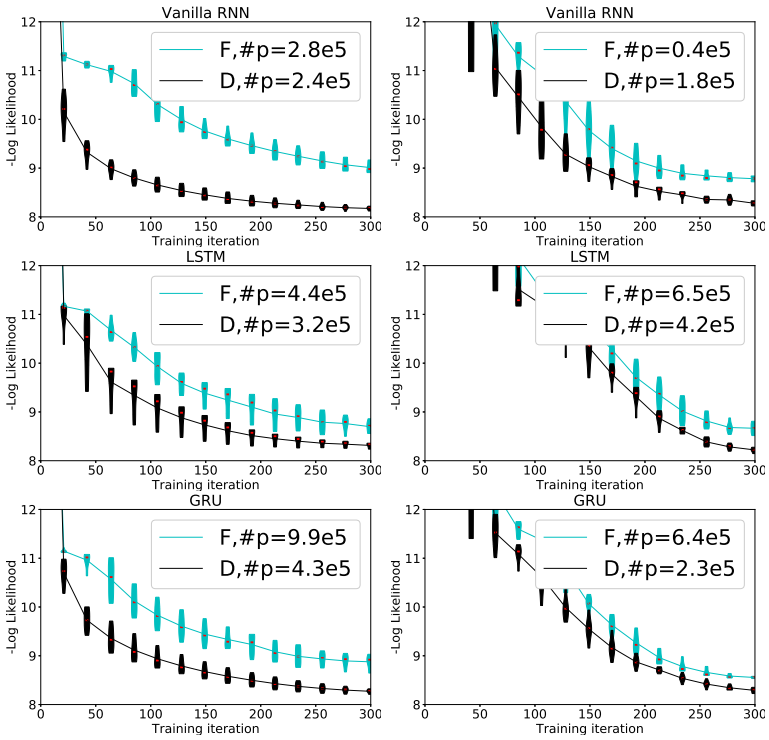


Figure 5.7: Training iterations vs test negative log-likelihoods on JSB Chorales dataset for full and diagonal models. The First column is for the Adam optimizer and the second column is for RMSProp. Black curves are for the diagonal models and cyan (gray in grayscale) curves are for full (regular) models. Top row is for VRNN, middle row is for LSTM and the bottom row is for GRU. Legends show the average number of parameters used by top 6 models (F is for Full, D is for Diagonal models). This caption also applies to Figures 5.8, 5.9, 5.10, with corresponding datasets.

strongly suggests that using diagonal recurrent matrices is suitable for modeling symbolic music datasets, and is potentially useful in other applications.

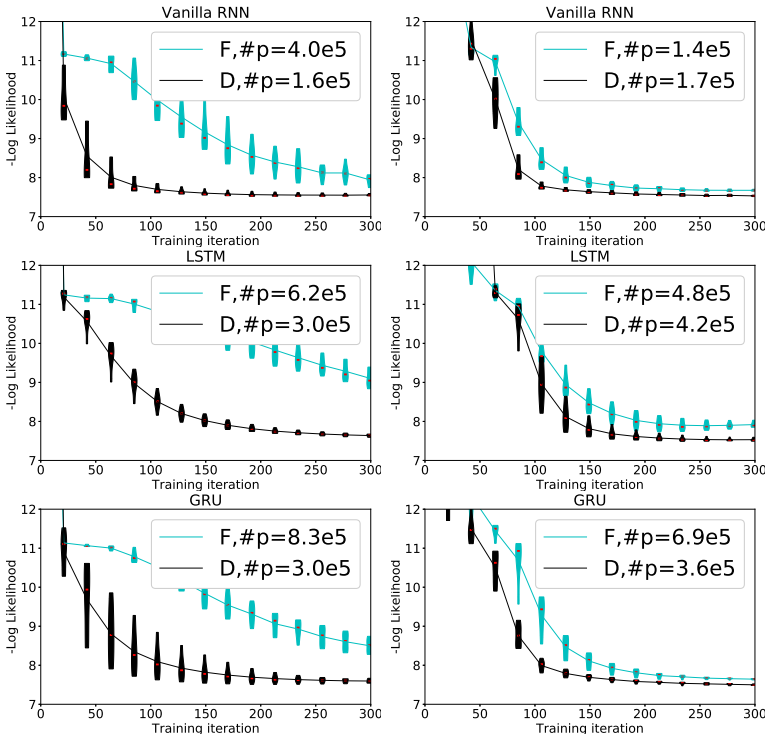


Figure 5.8: Training iterations vs test negative log-likelihoods on Piano-midi dataset.

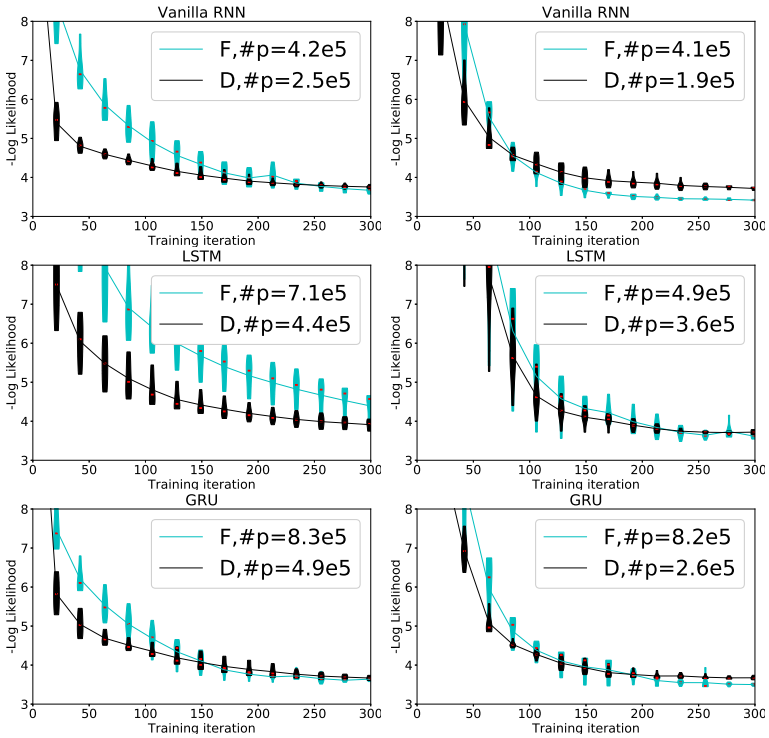


Figure 5.9: Training iterations vs test negative log-likelihoods on Nottingham dataset.

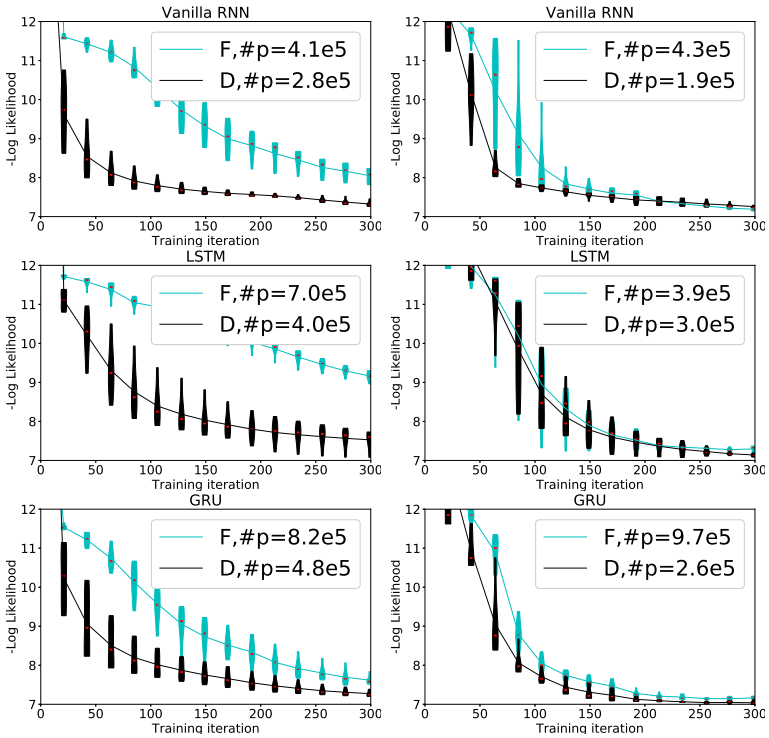


Figure 5.10: Training iterations vs test negative log-likelihoods on MuseData dataset.

- Except the Nottingham dataset, using the diagonal recurrent matrix results in an improvement in final test likelihood in all cases. Although the final negative likelihoods on the Nottingham dataset are larger for diagonal models, we still see some improvement in training speed in some cases, as we see that the black curves lie below the cyan curves for the most part.

- We see that the average number of parameters utilized by the top 6 diagonal models is in most cases smaller than that of the top 6 full models: In these cases, we observe that the diagonal models achieve comparable (if not better) performance by using fewer parameters.

Overall, we provide experimental data which strongly suggests that the diagonal RNNs can be a great alternative for regular full-recurrent-matrix RNNs.

CHAPTER 6: CONCLUSIONS

In this thesis we have studied various aspects of generative modeling for sequential data. As discussed in the introduction, in my view there are three main issues one deals with when doing generative modeling: Representation (modeling), learning paradigms (e.g. maximum likelihood, adversarial learning etc.), and optimization. We summarize our contributions regarding each aspect in Table 6.1:

Table 6.1: Summary of the contributions in the chapters of this thesis.

	Representation	**Learning Paradigm**	**Optimization**
Chapter 2	N.A.	MoM learning framework for HMMs	EM initialization with the MoM framwork
Chapter 3	Identifiable FHMM alternatives	N.A.	Proposed algorithms for FHMM
Chapter 4	Multi modal latent representation with IMLs	Maximum Likelihood Learning for Implicit Models	Two-Step optimization procedure
Chapter 5	Convolutive Architectures for Audio, Diagonal RNNs	GANs in Audio	N.A.

In more detail, our contributions specific to each chapter is as follows:

- In Chapter 2, we have studied an alternative parameter estimation method for HMMs with special transition structures: We have proposed a method of moments (MoM) based parameter estimation framework which is shown to be able to applicable to mixture of HMMs, switching HMMs, HMMs with mixture emissions, and left-to-right HMMs. We have experimentally shown that when the proposed method is used to initialized expectation maximization algorithm, it provides speed and accuracy boost.

- In Chapter 3, we have studied identifiability of Factorial Hidden Markov models. We have shown that the standard factorial model [13, 12] is not identifiable. We then proposed two alternative models which are identifiable. We have also proposed efficient algorithms for learning these alternative models.

- In Chapter 4, we have proposed an alternative generative model learning method based on maximum likelihood. The method learns multi-modal latent representations over the latent space, which we argue is vital for learning distributions over complicated real data such as natural images or audio. The experiments indicate that our method outperforms vastly popular methods such variational autoencoders [15] and GANs [16] in terms of KDE likelihood computed on the test set. We also see that random samples generated using our method are much less distorted compared to the aforementioned methods.

- In Chapter 5, we studied audio modeling with generative models. Our contributions are as follows: First, we proposed a convolutive neural network architecture for modeling spectrograms. We have shown that it enables significant performance increase in a speech source separation task. Second, we have argued that using an implicit generative model for the supervised source separation task increases the source separation performance. Third, we proposed an alternative recurrent neural network architecture for learning distributions over symbolic music representations (e.g. MIDI). The proposed model is able to achieve better test likelihoods, and converges faster than the standard models.

6.1 CONCLUDING THOUGHTS AND POTENTIAL FOLLOWUP WORK

Overall, after this thesis I think the representation issue is vital for successful learning as I tried to show in Chapter 4. I believe that using the better model with an approximate algorithm is better than using an approximate model with an exact algorithm (or at least should be the goal in mind): We saw in Chapter 4 that VAEs and GANs significantly underperform because of the simplistic assumptions on the latent representation. We also saw in Chapter 5 that explicitly modeling the temporal structure in spectrograms improves the source separation performance.

Unfortunately not all models are easy to learn (optimization for the model parameters is difficult) and alternative learning paradigms are created to alleviate the learning such as method of moments. Simpler learning algorithms such as method of moments is not useless as we have shown that they can be used to initialize more general methods such as maximum likelihood to make optimization easier. We have also seen that sometimes using a smaller model such as diagonal RNNs compared to full RNNs make the optimization easier.

The factorial HMM work in Chapter 3 shows that the standard model is not identifiable, and therefore it is a bad idea to try to learn the parameters of an FHMM. I do think that in the tasks where we care about the latent representations using identifiable/simpler models might help. I think that as a next step we should consider a more realistic unsupervised source separation task with real data, perhaps by using an EM algorithm for the identifiable models we have proposed in the chapter.

As we just alluded to, in Chapter 4, we have showed that using a multi-modal representation improves the quality of the learned distribution. I think it is a natural next step to consider different loss functions for autoencoder part of the algorithm to get the model to produce even higher quality samples.

REFERENCES

[1] McElreath, "Statistical rethinking," https://github.com/rmcelreath/rethinking, 2016, accessed: 2018-09-17.

[2] "!kung people," https://en.wikipedia.org/wiki/%C7%83Kung_people.

[3] 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*, vol. 39, no. 1, pp. 1–38, 1977.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[5] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[6] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016. [Online]. Available: http://arxiv.org/abs/1609.03499

[7] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *CoRR*, vol. abs/1601.06759, 2016. [Online]. Available: http://arxiv.org/abs/1601.06759

[8] L. Theis and M. Bethge, "Generative image modeling using spatial lstms," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969442.2969455 pp. 1927–1935.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[10] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[11] S. Roweis and Z. Ghahramani, "A unifying review of linear gaussian models," *Neural Comput.*, vol. 11, no. 2, pp. 305–345, Feb. 1999.

[12] Z. Ghahramani and M. I. Jordan, "Factorial hidden markov models," *Mach. Learn.*, vol. 29, no. 2-3, pp. 245–273, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1023/A:1007425814087

[13] Z. Ghahramani, "Factorial learning and the em algorithm," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, 1995, pp. 617–624.

[14] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 1303–1347, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2502581.2502622

[15] D. P. Kingma and M. Welling, "Auto-encoding variational bayes." *CoRR*, vol. abs/1312.6114, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1312.html#KingmaW13

[16] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[17] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, "Tensor decompositions for learning latent variable models," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 2773–2832, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2697055

[18] A. Anandkumar, D. J. Hsu, and S. M. Kakade, "A method of moments for mixture models and hidden markov models," *CoRR*, vol. abs/1203.0683, 2012. [Online]. Available: http://arxiv.org/abs/1203.0683

[19] D. J. Hsu and S. M. Kakade, "Learning mixtures of spherical gaussians: moment methods and spectral decompositions," in *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, 2013. [Online]. Available: http://doi.acm.org/10.1145/2422436.2422439 pp. 11–20.

[20] Y. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *CoRR*, vol. abs/1406.2572, 2014. [Online]. Available: http://arxiv.org/abs/1406.2572

[21] Z. Ghahramani and M. I. Jordan, "Factorial hidden markov models," *Mach. Learn.*, vol. 29, no. 2-3, pp. 245–273, Nov. 1997. [Online]. Available: https://doi.org/10.1023/A:1007425814087

[22] G. Hinton, L. Deng, D. Yu, G. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *Signal Processing Magazine*, 2012.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[24] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, "Natural language processing (almost) from scratch," *CoRR*, vol. abs/1103.0398, 2011. [Online]. Available: http://arxiv.org/abs/1103.0398

[25] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, pp. 3104–3112.

[26] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *PROCEEDINGS OF THE IEEE*, 1989, pp. 257–286.

[27] Y. C. Subakan, J. Traa, P. Smaragdis, and D. Hsu, "Method of moments learning for left-to-right hidden markov models," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2015.

[28] S. Chiappa, "Explicit-duration markov switching models," *Found. Trends Mach. Learn.*, vol. 7, no. 6, pp. 803–886, Dec. 2014. [Online]. Available: http://dx.doi.org/10.1561/2200000054

[29] C. Keskin, A. T. Cemgil, and L. Akarun, "Dtw based clustering to improve hand gesture recognition," in *Proceedings of the Second International Conference on Human Behavior Unterstanding*, ser. HBU'11. Berlin, Heidelberg: Springer-Verlag, 2011. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25446-8_8 pp. 72–81.

[30] P. Smyth, "Clustering sequences with hidden markov models," in *Advances in Neural Information Processing Systems*. MIT Press, 1997, pp. 648–654.

[31] Y. C. Subakan, J. Traa, and P. Smaragdis, "Spectral learning of mixture of hidden markov models," in *Neural Information Processing Systems (NIPS)*, 2014.

[32] Y. C. Subakan, J. Traa, and P. Smaragdis, "Spectral learning of hidden markov models with group persistence," 2015.

[33] J. H. McDermott, "The cocktail party problem," *Current Biology*, vol. 19, no. 22, pp. R1024–R1027, 2009.

[34] Y. C. Subakan, J. Traa, P. Smaragdis, and N. Stein, "A dictionary learning approach for factorial gaussian models," *https://arxiv.org/abs/1508.04486*, 2015.

[35] Y. C. Subakan and P. Smaragdis, "A dictionary learning approach for factorial models," in *Submitted to ICASSP 2017*, 2017.

[36] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, no. 6, pp. 721–741, Nov. 1984. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.1984.4767596

[37] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=645530.655813 pp. 282–289.

[38] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," *CoRR*, vol. abs/1506.02216, 2015. [Online]. Available: http://arxiv.org/abs/1506.02216

[39] O. Fabius and J. R. van Amersfoort, "Variational Recurrent Auto-Encoders," *ArXiv e-prints*, Dec. 2014.

[40] P. Smaragdis, "Non-negative matrix factor deconvolution; extraction of multiple sound sources from monophonic inputs." in *ICA*, ser. Lecture Notes in Computer Science, vol. 3195. Springer, 2004, pp. 494–499.

[41] S. Venkataramani, Y. C. Subakan, and P. Smaragdis, "Neural network alternatives to convolutive audio models for source separation," in *IEEE Workshop on Machine Learning for Signal Processing (MLSP)*, 2017.

[42] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning (ICML) 2013, Atlanta, GA, USA, 16-21 June 2013*, 2013. [Online]. Available: http://jmlr.org/proceedings/papers/v28/pascanu13.html pp. 1310–1318.

[43] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[44] Y. C. Subakan and P. Smaragdis, "Diagonal rnns in symbolic music modeling," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2017.

[45] J. Bayer and C. Osendorfer, "Learning Stochastic Recurrent Networks," *ArXiv e-prints*, Nov. 2014.

[46] A. Anandkumar, D. P. Foster, D. J. Hsu, S. M. Kakade, and Y. Liu, "Two svds suffice: Spectral decompositions for probabilistic topic modeling and latent dirichlet allocation," *CoRR*, vol. abs/1204.6703, 2012. [Online]. Available: http://arxiv.org/abs/1204.6703

[47] K. Pearson, "Contributions to the mathematical theory of evolution," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 185, pp. 71–110, 1894. [Online]. Available: http://rsta.royalsocietypublishing.org/content/185/71

[48] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009. [Online]. Available: https://doi.org/10.1137/07070111X

[49] D. J. Hsu, S. M. Kakade, and T. Zhang, "A spectral algorithm for learning hidden markov models," *J. Comput. Syst. Sci.*, vol. 78, no. 5, pp. 1460–1480, 2012. [Online]. Available: https://doi.org/10.1016/j.jcss.2011.12.025

[50] P. Comon and C. Jutten, *Handbook of Blind Source Separation: Independent Component Analysis and Applications*, 1st ed. Academic Press, 2010.

[51] A. P. Parikh, L. Song, and E. P. Xing, "A spectral algorithm for latent tree graphical models," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, 2011, pp. 1065–1072.

[52] A. P. Parikh, L. Song, M. Ishteva, G. Teodoru, and E. P. Xing, "A spectral algorithm for latent junction trees," *CoRR*, vol. abs/1210.4884, 2012. [Online]. Available: http://arxiv.org/abs/1210.4884

[53] A. T. Chaganty and P. Liang, "Estimating latent-variable graphical models using moments and likelihoods," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 2. Bejing, China: PMLR, 22–24 Jun 2014, pp. 1872–1880.

[54] A. Kontorovich, B. Nadler, and R. Weiss, "On learning parametric-output hmms," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, 2013. [Online]. Available: http://jmlr.org/proceedings/papers/v28/kontorovich13.html pp. 702–710.

[55] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000. [Online]. Available: https://doi.org/10.1109/34.868688

[56] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[57] A. Chaganty and P. Liang, "Spectral experts for estimating mixtures of linear regressions," in *International Conference on Machine Learning (ICML)*, 2013.

[58] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[59] P. Brossier and P. Leveau, "2013:audio onset detection," http://www.music-ir.org/mirex/wiki/2013:Audio_Onset_Detection, June 2013, accessed: 2018-03-20.

[60] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," http://cvxr.com/cvx, Mar. 2014, accessed: 2018-03-20.

[61] Y. C. Subakan, "Probabilistic time series classification," M.S. thesis, Bogazici University, 2013.

[62] M. Aharon, M. Elad, and A. Bruckstein, "*rmk*-svd: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Transactions on Signal Processing*, vol. 54, no. 11, pp. 4311–4322, Nov 2006.

[63] K. Kreutz-Delgado, J. F. Murray, B. D. Rao, K. Engan, T.-W. Lee, and T. J. Sejnowski, "Dictionary learning algorithms for sparse representation," *Neural Comput.*, vol. 15, no. 2, pp. 349–396, Feb. 2003.

[64] P. Comon, "Independent component analysis, a new concept?" *Signal Processing*, vol. 36, no. 3, pp. 287 – 314, 1994, higher Order Statistics. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0165168494900299

[65] D. A. Spielman, H. Wang, and J. Wright, "Exact recovery of sparsely-used dictionaries," *CoRR*, vol. abs/1206.5882, 2012. [Online]. Available: http://arxiv.org/abs/1206.5882

[66] Z. Ghahramani, "Hidden markov models." River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2002, ch. An Introduction to Hidden Markov Models and Bayesian Networks, pp. 9–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=505741.505743

[67] S. Dasgupta, "Learning mixtures of gaussians," in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, ser. FOCS '99. Washington, DC, USA: IEEE Computer Society, 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=795665.796496 pp. 634–.

[68] E. Zwicker, "Subdivision of the audible frequency range into critical bands (frequenzgruppen)," *The Journal of the Acoustical Society of America*, vol. 33, no. 2, p. 248, 1961.

[69] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[70] E. Vincent, R. Gribonval, and C. Fvotte, "Performance measurement in blind audio source separation." *IEEE Trans. Audio, Speech and Language Processing*, vol. 14, no. 4, pp. 1462–1469, 2006.

[71] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society, Series B*, vol. 61, pp. 611–622, 1999.

[72] S. Mohamed and B. Lakshminarayanan, "Learning in Implicit Generative Models," *ArXiv e-prints*, Oct. 2016.

[73] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[74] T. P. Minka, "Expectation propagation for approximate bayesian inference," in *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, ser. UAI '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=647235.720257 pp. 362–369.

[75] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2234–2242. [Online]. Available: http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf

[76] S. Nowozin, B. Cseke, and R. Tomioka, "f-gan: Training generative neural samplers using variational divergence minimization," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 271–279. [Online]. Available: http://papers.nips.cc/paper/6066-f-gan-training-generative-neural-samplers-using-variational-divergence-minimization.pdf

[77] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *CoRR*, vol. abs/1701.07875, 2017. [Online]. Available: http://arxiv.org/abs/1701.07875

[78] T. Salimans, H. Zhang, A. Radford, and D. Metaxas, "Improving GANs using optimal transport," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rkQkBnJAb

[79] Y. Saatci and A. G. Wilson, "Bayesian gan," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3622–3631. [Online]. Available: http://papers.nips.cc/paper/6953-bayesian-gan.pdf

[80] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," *CoRR*, vol. abs/1605.08803, 2016. [Online]. Available: http://arxiv.org/abs/1605.08803

[81] L. Devroye, *Non-Uniform Random Variate Generation(originally published with*, 1986.

[82] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[83] "Volume and jacobian determinant," https://en.wikipedia.org/wiki/Determinant, accessed: 2018-03-09.

[84] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[85] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2015. [Online]. Available: http://arxiv.org/abs/1511.06434

[86] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[87] "Free spoken digit dataset," https://github.com/Jakobovski/free-spoken-digit-dataset, accessed: 2018-03-09.

[88] P. Smaragdis and J. C. Brown, "Non-negative matrix factorization for polyphonic music transcription," in *In IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2003, pp. 177–180.

[89] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription." in *ICML*, 2012.

[90] Y. C. Sübakan and P. Smaragdis, "Generative adversarial source separation," *CoRR*, vol. abs/1710.10779, 2017. [Online]. Available: http://arxiv.org/abs/1710.10779

[91] P. Smaragdis and S. Venkataramani, "A neural network alternative to non-negative audio models," *CoRR*, vol. abs/1609.03296, 2016. [Online]. Available: http://arxiv.org/abs/1609.03296

[92] P. Smaragdis and S. Venkataramani, "A neural network alternative to non-negative audio models," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, 2017. [Online]. Available: https://doi.org/10.1109/ICASSP.2017.7952123 pp. 86–90.

[93] A. T. Cemgil, U. Simsekli, and Y. C. Sübakan, "Probabilistic latent tensor factorization framework for audio modeling," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, WASPAA 2011, New Paltz, NY, USA, October 16-19, 2011*, 2011. [Online]. Available: https://doi.org/10.1109/ASPAA.2011.6082315 pp. 137–140.

[94] C. Févotte and J. Idier, "Algorithms for nonnegative matrix factorization with the beta-divergence," *CoRR*, vol. abs/1010.1763, 2010. [Online]. Available: http://arxiv.org/abs/1010.1763

[95] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *CoRR*, vol. abs/1701.00160, 2017. [Online]. Available: http://arxiv.org/abs/1701.00160

[96] S. Pascual, A. Bonafonte, and J. Serrà, "SEGAN: speech enhancement generative adversarial network," *CoRR*, vol. abs/1703.09452, 2017. [Online]. Available: http://arxiv.org/abs/1703.09452

[97] P. Smaragdis, C. Fevotte, G. J. Mysore, N. Mohammadiha, and M. Hoffman, "Static and dynamic source separation using nonnegative factorizations: A unified view," *IEEE Signal Processing Magazine*, vol. 31, no. 3, pp. 66–75, 2014.

[98] J. S. Garofolo, L. F. Lamel, W. M. F. J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue, "Timit acoustic phonetic continuous speech corpus," Philadelphia, 1993.

[99] "Root mean square propagation (rmsprop)," http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm, accessed: 2017-April-12.

[100] T. Virtanen, J. F. Gemmeke, B. Raj, and P. Smaragdis, "Compositional models for audio processing: Uncovering the structure of sound mixtures," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 125–144, 2015.

[101] E. M. Grais and M. D. Plumbley, "Single channel audio source separation using convolutional denoising autoencoders," *arXiv preprint arXiv:1703.08019*, 2017.

[102] S. Venkataramani and P. Smaragdis, "End-to-end source separation with adaptive front-ends," *arXiv preprint arXiv:1705.02514*, 2017.

[103] P. Chandna, M. Miron, J. Janer, and E. Gómez, "Monoaural audio source separation using deep convolutional neural networks," in *International Conference on Latent Variable Analysis and Signal Separation*. Springer, 2017, pp. 258–266.

[104] J. O. Smith, *Introduction to Digital Filters with Audio Applications*. http://ccrma.stanford.edu/~jos/fp/.

[105] P. Smaragdis, B. Raj, and M. Shashanka, "Supervised and semi-supervised separation of sounds from single-channel mixtures," *Independent Component Analysis and Signal Separation*, pp. 414–421, 2007.

[106] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, 2010. [Online]. Available: http://www.jmlr.org/proceedings/papers/v9/glorot10a.html pp. 249–256.

[107] T. Mikolov, M. Karafit, L. Burget, J. Cernock, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH*. ISCA, 2010, pp. 1045–1048.

[108] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013.

[109] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *CoRR*, vol. abs/1303.5778, 2013. [Online]. Available: http://arxiv.org/abs/1303.5778

[110] C.-C. Ku and K. Lee, "Diagonal recurrent neural networks for dynamic systems control," *IEEE Transactions on Neural Networks*, 1995.

[111] "Diagonal recurrent neural network based adaptive control of nonlinear dynamical systems using lyapunov stability criterion," *ISA Transactions*, vol. 67, pp. 407 – 427, 2017.

[112] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," 2015.

[113] D. Reynolds, *Gaussian Mixture Models*. Boston, MA: Springer US, 2015, pp. 827–832. [Online]. Available: http://dx.doi.org/10.1007/978-1-4899-7488-4_196

[114] E. Alpaydin, *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[115] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *CoRR*, vol. abs/1503.04069, 2015. [Online]. Available: http://arxiv.org/abs/1503.04069

[116] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[117] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/