CONTINUOUSLY PARAMETERIZED MIXTURE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture models are universal approximators of smooth densities but are difficult to utilize in complicated datasets due to restrictions on typically available modes and challenges with initialiations. We show that by continuously parameterizing a mixture of factor analyzers using a learned ordinary differential equation, we can improve the fit of mixture models over direct methods. Once trained, the mixture components can be extracted and the neural ODE can be discarded, leaving us with an effective, but low-resource model. We additionally explore the use of a training curriculum from an easy-to-model latent space extracted from a normalizing flow to the more complex input space and show that the smooth curriculum helps to stabilize and improve results with and without the continuous parameterization. Finally, we introduce a hierarchical version of the model to enable more flexible, robust classification and clustering, and show substantial improvements against traditional parameterizations of GMMs.

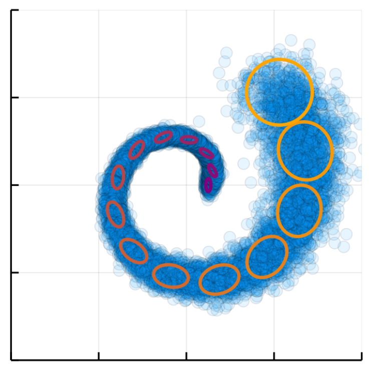
1 INTRODUCTION

Mixture models have long served as reliable tools for both modeling (density estimation) and summarizing (cluster analysis) datasets. Through their probabilistic nature, mixture models provide an estimate of the underlying data distribution; through a parsimonious collection of components, mixture models succinctly summarize the major patterns found in a dataset. Gaussian mixture models (GMMs), which provide a universal approximation for smooth densities (20), have been effectively deployed for modeling and clustering in a wide range of applications (4; 17; 31).

There are two major challenges in applying GMMs to complicated, high dimensional data. First, the partitions (modes) that we wish to characterize in data are rarely spherical or simple enough in nature to be faithfully captured with Gaussian components. As an alternative, one may consider more complicated components (27), however, this worsens identifiability issues and yields partitions that do not adequately summarize populations of interest in the data (4). Second, mixture models converge to local minima and thus their optimization is *extremely* sensitive to initialization. Some approaches propose to initialize via local fitting methods such as k-means (46). However, the distance metrics, for example Euclidean distance, implied in those local fitting methods is rarely appropriate in high dimensions, imposing initial partitioning that is arduous to escape.

We propose to offset the first difficult by taking a mixture of mixtures in a hierarchical approach by considering a *mixture of distinct dynamical systems*. In effect, we propose to learn to evolve the components of one partition (see Fig. 1a) in a fashion that results in an infinite mixture model in the limit. This alleviates the burden of learning a large number of separate modes through a shared generator of components in an approach that is akin to hypernetworks (23). We can then combine multiple partitions to cover the entire support (see Fig. 1b). Specifically, each partition is itself defined by multiple components that are spanned through smooth dynamics to represent complex, multi-modal distributions. We sample discrete components from this continuous parameterization during inference to construct a cheap, lightweight model that dramatically reduces the computational cost, maintains improved performance over typical mixture methods, and simplifies model interpretation.

We alleviate the second difficulty by creating a learned curriculum (2) over the data. We construct the curriculum by transforming the data into a standard Gaussian through a normalizing flow (7; 12; 22; 33) and slowly annealing back to the original space. This removes the difficulty with initialization since the distribution early in the training process is *known*. As we advance through the curriculum, the model only requires minor perturbations from its previous state. We find that this process results in considerably improved performance regardless of the mixture parameterization.


(a) Evolution of a single continuous mixture model.

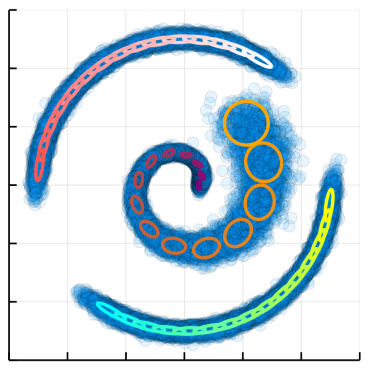

(b) Evolution of several continuous mixtures. Color schemes represent different mixtures.

Figure 1: We illustrate how a continuously parameterized mixture model can evolve components to model the data distribution (samples in blue). Each ellipse corresponds to equal likelihood contours for a different component. Colors across components implicitly indicates the arrow of pseudotime. (a) A single *trajectory* through the space provides a smooth probabilistic model. (b) A hierarchy of three different partitions allows for different manifolds and enables flexible clustering or classification.

Main Contributions We propose a parameterization of Gaussian mixture models through the output of a neural ordinary differential equation (NODE). This formulation induces a smoothly-varying trajectory through the data, in essence, learning a probabilistic sheath across the data manifold. Once trained, the components of the GMM can be extracted and the NODE discarded, rendering storage and inference notably cheaper than most other modern methods. We include a hierarchical component to the model by considering multiple trajectory starting points to allow us to utilize simple Bayesian methods for clustering or classification. Finally, we demonstrate a curriculum-based training method to significantly improve model performance. Code will be made available upon publication.

2 Background

2.1 Gaussian Mixture Models

Traditional (finite) mixture models model the presence of sub-populations within a dataset through a convex combination of distinct components of a chosen distribution. Gaussian mixture models (GMMs), the most common class of mixture models, restrain the set of chosen distributions that constitutes the mixture to the Gaussian family:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k) \,, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^M$ is the input data, π is the component weight, $\mu \in \mathbb{R}^M$ is the mean, $\Sigma \in \mathbb{R}^{M \times M}$ is the covariance matrix, and K is the number of components in the mixture. Despite the seemingly stringent simplification GMMs make, they encompass a broad set of distributions. In effect, they are universal approximators to any smooth probability distribution given enough mixture components (20). This offers us an insight when deploying GMMs for data modeling: the more complex the data distribution is, the more components the GMM would likely require to yield a reasonable approximation. Motivated by this insight, analogous to defining an integral as an infinite sum, we take the limit $K \to \infty$ in Eq. 1 to arrive at the following continuous representation of GMMs

$$p(\mathbf{x}) = (2\pi)^{-M/2} \int_0^1 |\Sigma(s)|^{-0.5} \exp\left(- (\mathbf{x} - \mu(s))^T \, \Sigma^{-1}(s) \, (\mathbf{x} - \mu(s)) \, /2 \right) \pi(s) \, ds \tag{2}$$

where the set of parameters, $\{\pi_k, \mu_k, \Sigma_k\}_k$, for the finite GMMs become the set of functions, $\{\pi(s), \mu(s), \Sigma(s)\}$, that is parametrized by the variable s whose meaning depends on the specific context of the problem. Eq. 1 can then be reversely seen as a discretized formulation of Eq. 2.

2.1.1 Mixture of Factor Analyzers

General GMMs require $\mathcal{O}(M^2)$ parameters for each component. To reduce this parameter cost, we utilize a Mixture of Factor Analyzers (19; 46) which parameterizes the covariance matrix of each component as the sum of a diagonal matrix, D, and the inner product of a low-rank matrix, $A \in \mathbb{R}^{M\times R}$, with itself where $R \ll M$, such that $\Sigma = D + AA^T$. This brings the number of parameters down to $\mathcal{O}(MR)$ and allows for $\mathcal{O}(MR^2)$ cost to estimate the determinant and inverse.

2.2 Neural Ordinary Differential Equations

Built upon the observation that a residual network (25) can be seen as an Euler discretization for solving an ODE, Chen et al. (8) introduced *neural* ODEs (NODEs), a class of continuous-depth neural networks that parameterize the class of ODEs that aim to solve the initial value problem:

$$\frac{dh(t)}{dt} = f\left(h(t), t; \theta\right), \ \ h(t_0) = h_{t_0}\,, \tag{3}$$

with known initial value h_{t_0}. Instead of being restricted to the Euler discretization, Neural ODEs are able to incorporate any black-box ODE solver in a memory efficient manner to compute

$$h(t_{i+1}) = h(t_i) + \int_{t_i}^{t_{i+1}} f\left(h(t), t; \theta\right) dt\,. \tag{4}$$

In practice, for tabular data, we set the architecture of f to be a multilayer perceptron with an explicit dependence on t and use either $\tanh$ or $\sin$ as the activation function. We find that when using $\sin$, the ODE seems to be less prone to becoming stiff (3) and hence is easier to numerically integrate. When using image data, we utilize a convolutional neural network with the same activation functions.

3 Methods

In this section we discuss how we utilize neural ordinary differential equations (NODE) to parameterize mixture models over continuous indices and how to improve the training process by instituting a curriculum to guide the model from an easy-to-learn space to the true data space. The NODE allows for a hyper-network (23) type approach, which shares weights of a network to output parameters over multiple components, and whose limit is an infinite GMM. We then extend a continuously parameterized mixture model by allowing for multiple trajectories in a hierarchical structure that allows us to perform clustering or classification.

3.1 Continuously Parameterized Mixture Models

Inspired by the universal approximator properties of Eq. 2, where the parameters of each component depend on the latent variable, s, we chose to parameterize the MFA via a neural network. We begin by constructing a joint state across the MFA parameters. When the data is tabular, this amounts to flattening each parameter and concatenating them together:

$$h(t) = \left[\mu(t), \bar{A}(t), \log(d)(t), \log(\pi)(t)\right] \tag{5}$$

where the bar over A indicates the matrix has been flattened, d is the diagonal portion of D, and $\log$ is applied element-wise. We choose to use a shared state across parameters rather than a separate ODE for each parameter so that the model can better coordinate and share relevant information. We learn the log of d and π instead of the parameters directly to ensure that the covariance is positive definite and that the weights are non-negative. As a result, $h \in \mathbb{R}^J$ where $J = (R+2)M + 1$. In most cases, we set the value of π to a constant and exclude it from the state to encourage contiguous modes.

There are two possible ways to proceed. If our goal was to best match Eq. 2, we would construct the NODE based on the joint state and solve the system of ODEs that includes the continuous mixture

$$\begin{aligned}
\frac{dh(t)}{dt} &= f(h(t), t; \theta) \\
\frac{dp(x)}{dt} &= (2\pi)^{-M/2}\left|\Sigma(t)\right|^{-0.5}\exp\Big(-\left(\mathbf{x}-\mu(t)\right)^T\Sigma^{-1}(t)\left(\mathbf{x}-\mu(t)\right)/2\Big)\,\pi(t)
\end{aligned}$$

where we evaluate each component in accordance with Eq. 2 on the instantaneous value of t within the ODE solver and set $p(x) = 0$ at $t = 0$ and take the integral over the range of t without producing intermediate values. Unfortunately, while this formulation is a better match to the continuous mixture model, integrating $p(x)$ is numerically unstable and often causes f to become stiff (3) or for the integrator to underflow, see Appendix for additional details. Additionally, this form would require that we keep the NODE once training is complete and resolve it for every new example encountered.

We instead choose to solve for the joint state directly and return the parameters at a (possibly variable) number of pseudotimes. That is, for a (uniformly drawn) set of pseudotimes, $\{t_j\}_{j=1}^{G}$, we model the density as $p(\mathbf{x}) = \sum_{j=1}^{G} \pi(t_j)\,\mathcal{N}\left(\mathbf{x};\, \mu(t_j), A(t_j)A(t_j)^T + D(t_j)\right)$, where $\pi(t_j), \mu(t_j), A(t_j), D(t_j)$ are the respective continuously indexed parameters and $\sum_j \pi(t_j) = 1$. This approach is essentially doing numerical integration to approximate Eq. 2, which is feasible given the $1d$ integral. This leaves us with a *mixture model* whose parameters are derived from a *continuous process*. We will refer to this model as a continuously parameterized mixture model (CPMM). Figure 1 illustrates a dataset overlayed with the components extracted from a CPMM.

We have specifically chosen f so that the ODE is not autonomous (3). From a practical perspective, this means that our models are capable of learning loops and cycles. However, to further increase the flexibility of the model, we additionally augment the joint state

$$h(t) = \left[\mu(t), \bar{A}(t), \log(d)(t), \alpha(t)\right] \tag{6}$$

where $\alpha \in \mathbb{R}^L$ and we have excluded $\log(\pi)$ from the state. α is not directly used in evaluating any portion of the mixture but instead provides an unconstrained pathway that the network can use to pass information along. Without the augmented state, the network must encode information relevant to future times into the parameters of the current time, overloading those parameters and notably decreasing performance. We generally choose L to be 2-4$\times$ larger than J, so $h \in \mathbb{R}^{5J}$.

In this form, the NODE does not depend on the evaluation data point, $\mathbf{x}$, directly and can be solved independently. In fact, the NODE can be evaluated with only the initial value and the parameter times. We can therefore consider that our model is a type of hypernetwork (23). We train the model by evaluating/solving the hypernetwork against the *learnable* initial state and then evaluating the likelihood for each example in the batch. The initialization and hypernetwork dynamics are then updated to maximize the likelihood of the batch. During evaluation, we solve the hypernetwork *once* and cache the parameters (for a uniformly drawn set of pseudotimes $\{t_j\}_{j=1}^{G}$) against future executions. This places the total computational cost for inference at $\mathcal{O}(KMR)$.

When the input data is an image, we construct the joint (augmented) state by keeping the parameters in the image shape for the NODE and concatenating over channels. We then flatten the data and parameters to calculate the likelihood. This means that the unaugmented portion of the channel dimension is increased by a factor of $R + 2$ relative to the image channel count (when π is held fixed).

3.2 HIERARCHICAL MIXTURE OF FACTOR ANALYZERS

To facilitate clustering and classification, we extend the mixture of factor analyzers (MFAs) in a hierarchical manner. More specifically, we model the data with a *mixture of MFAs*

$$p(\mathbf{x}) = \sum_{c=1}^{C} \eta_c\, p(\mathbf{x}\,|c) = \sum_{c=1}^{C} \eta_c \sum_{j=1}^{G} \pi_c(t_j)\,\mathcal{N}\left(\mathbf{x};\, \mu_c(t_j), A_c(t_j)A_c(t_j)^T + D_c(t_j)\right)\,, \tag{7}$$

where C denotes the number of MFAs and, therefore the number of clusters/classes, $\sum_c \eta_c = 1$, and $\{t_j\}_{j=1}^{G}$ is a set of pseudotimes. We construct the mixture of the C MFAs by providing C different initial values to the NODE. Alternatively, we could learn a separate NODE and initial state for each MFA. This should allow for greater flexibility in each trajectory. However, we generally find that the improvement is not appreciable while the increase in compute is significant. Sharing the ODE means that the trajectories will learn from one another and share certain dynamic characteristics.

The hierarchical mixture allows the model to learn disparate data partitions without requiring that a single trajectory transition through low probability regions. Additionally, we can utilize Bayes rule to estimate $p(y = c|\,\mathbf{x})$ from Eq. 7 to perform clustering or classification.

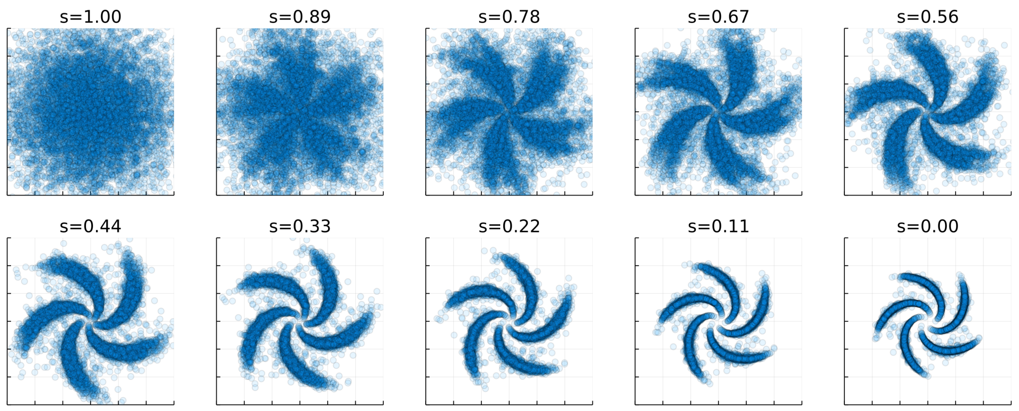


Figure 2: Evenly distributed transitions through different spaces. The data begins in a latent space as a standard Gaussian (top left) and ends in the true space (bottom right).

3.3 Curriculum Through Spaces

Unfortunately, training large mixture models in high dimensions often gets stuck in local minima and is *extremely* sensitive to initialization. Direct mixtures (mixtures with disconnected parameters) are often initialized via a combination of kmeans and local fitting prior to gradient descent (46). However, for CPMMs, the parameters are the product of a learnable process and cannot be directly initialized. As a heuristic, one may initialize the ODE initial value based on kmeans or labeled examples, however this only provides a limited signal and one must resolve how to initialize other parameters such as A or $\log d$.

In order to circumvent this difficulty, we borrow ideas from curriculum learning (2). We begin by defining a continuously indexible map, $\forall v \in [0, 1],\ \mathcal{M}_v : \mathbb{R}^M \mapsto \mathbb{R}^M$ such that $\mathcal{M}_0(\mathbf{x}) = \mathbf{x}$, and $\mathcal{M}_v$ progressively maps to a smoother, simpler space as v increases. We propose to leverage $\mathcal{M}$ to construct a curriculum for learning the CPMM. Intuitively, we may begin training on the simplest space induced by $\mathcal{M}_1$ and slowly progress to training on our target input space $\mathcal{M}_0$. That is, for a sequence $1 = v_1 > \ldots > v_T = 0$, we progressively train on respective datasets $\mathcal{D}_t = \{\mathcal{M}_t(\mathbf{x}_i)\}_{i=1}^{N}$. An accessible choice is to define $\mathcal{M}_v$ as a continuous normalizing flow (22), $\mathcal{M}_v = \mathbf{u}_v$, where

$$\mathbf{u}_v(\mathbf{x}) = \int_0^v g(\mathbf{u}_s(\mathbf{x}), s; \phi) ds, \quad \log p_{\mathbf{u}_v}(\mathbf{u}_v(\mathbf{x})) = \log \mathcal{N}(\mathbf{u}_1(\mathbf{x}); 0, I) + \int_v^1 \mathrm{tr}\left(\frac{\partial g}{\partial s}\right) ds, \tag{8}$$

$\mathbf{u}_0(\mathbf{x}) \equiv \mathbf{x}$ and $\mathbf{u}_v(\mathbf{x})$ is defined by a learnable function, g, which is trained via MLE so that $\mathbf{u}_1(\mathbf{x}) \sim \mathcal{N}(0, I)$.

We construct the curriculum for the CPMM by first training the CPMM on $\mathbf{u}_1$. Since the data is (ideally) a standard Gaussian in this space, it is trivial to learn a good CPMM and the initialization of the CPMM parameters can be created as perturbations from the standard Gaussian. We then marginally perturb the space by taking a small step in v towards the input, e.g., we train the CPMM on $\mathbf{u}_{1-\epsilon}$ where, for sufficiently small ϵ, $p_{\mathbf{u}_{1-\epsilon}}(\mathbf{u}_{1-\epsilon}) \approx p_{\mathbf{u}_1}(\mathbf{u}_1)$. We repeat this process of perturbing the space and then updating the CPMM until we have arrived back at the input space. Figure 2 shows this smooth transition on a two-dimensional dataset as the map transitions between the latent space, through several intermediate spaces, before arriving back at the input space.

One interpretation of this curriculum is that we learn the initialization for one space based on training over a marginally simpler version of the data. This interpretation applies to both the parameters of the NODE hypernetwork and the initial values of each trajectory. Without this slow update procedure, even with a good initial value, the CPMM can be hard to train since the initial trajectories can point in the wrong directions and the model can struggle to shift the probability mass appropriately.

An important distinction between our use of a curriculum and the typical form of curriculum learning is that we do not want to remember earlier "tasks." Doing so would mean that the CPMM would still capture the intermediate spaces between the Gaussian noise and the input space. Since these spaces are only used as a guide and have no intrinsic meaning, maintaining them has no direct value.

4 Related Work

Mixture Models Mixture models can be applied to solve an array of ML problems, e.g., clustering and density estimation (24), and are widely deployed in modern ML methods. Viroli et al. (49) model the variables at each layer of a deep network with a Gaussian mixture model (GMM), leading to a set of nested mixtures of linear models. Izmailov et al. (28) use a GMM as the base distribution in conjunction with normalizing flows for semi-supervised learning. Richardson et al. (46) demonstrate that GMMs better capture the statistical modes of the data distribution than generative adversarial models (GANs), while Eghbal-zadeh et al. (14) incorporate a GMM into the discriminator of a GAN to encourage the generator to exploit different modes in the data.

Tractable Likelihood Models Normalizing flows and autoregressive models are two common types of extremely effective tractable likelihood estimators. Normalizing flows (NF) learn invertible transformations to a latent space where the data has a known, prechosen, distribution, typically the standard normal. There exist considerable variations between models based on the families of invertible functions allowed (7; 12; 22; 33). Autoregressive (AR) models (39; 40; 48) exploit the probabilistic chain rule and learn a distribution for each dimension conditioned on the previous features. Despite their impressive performance as likelihood estimators and as generative methods, these models are surprisingly brittle. In particular, they show higher likelihoods for completely different datasets than the ones they were trained on which prevents them from being utilized for outlier detection (26; 38).

Deep Clustering Several approaches to apply deep networks for clustering have been proposed in the past few years (5; 51), centering around the concept that the input space in which traditional clustering algorithms operate is of importance. There have also been works on incorporating traditional clustering methods, such as spectral clustering or hierarchical clustering, directly into deep networks (6; 34; 47). Mukherjee et al. (36) extend GANs for clustering by using a combination of discrete and continuous latent variables.

Mixture Models + Deep Nets for Clustering Since mixture models are the traditional models of choice for clustering tasks, arming them further with the recent development in deep learning is only natural. Zhang et al. (52) directly deployed a mixture of autoencoders with the assumption that different clusters are effectively different local data manifolds, and thus could be parametrized and learned by autoencoders. In the same vein, Pires et al. (44) model the data using a mixture of normalizing flows, where the mixture weights are computed through optimizing the variational lower bound. Jiang et al. (30) use a GMM as the prior distribution for the latent code in a variational auto-encoder (VAE), thus allowing for clustering of the input data in the latent space.

Non-parametric Bayesian Mixture Models Non-parametric Bayesian models (37) assume the number of parameters of the underlying model grows with the number of data samples. In the case of the underlying model being a non-parametric Bayesian mixture model, this indicates that for every new data sample, the model can either group it with the already-discovered mixture components or initiate a new mixture component for that data sample, increasing the number of mixture components deployed and hence the number of parameters utilized (21). This is accomplished by using a Dirichlet process for the prior distribution of the parameters of non-parametric Bayesian mixture models (18). Despite effectively deploying an infinite number of mixture components during the training stage, the number of components eventually learned is determined by the finite data samples available, resulting in a learned finite mixture distribution at test time.

Sum-Product Networks Sum-product networks (SPN) (42; 43; 45) are a class of probabilistic circuits (9) that produce a tractable likelihood estimate akin to normalizing flows and autoregressive methods. SPNs are constructed as a directed acyclic graph composed of alternating "sum" and "product" operations over the evaluation of (typically) one-dimensional densities at each leaf node. Learning is often performed through the EM algorithm by updating the parameters of each leaf distribution and the weights at each sum node. This structure allows SPNs to cheaply evaluate a variety of statistical quantities (e.g., marginals) without approximation. When leaf densities are Normal distributions, the SPN can be expressed as a GMM with many components (29).

5 Experiments

We construct our models in PyTorch (41) and train using PyTorch Lightning (15). We used Adam (32) as the optimizer in all cases. Unless otherwise stated, we extract 25 components per trajectory for each hierarchical CPMM. Curriculum models were constructed using simple CNFs from FFJORD (22). Since the execution of a CNF is computationally expensive, we extracted the smoothly varying datasets (see Sec. 3.3), $\mathcal{D}_t$, immediately after training and saved them to disk. In general, we choose to use 51 total spaces (including the latent space, $\mathbf{u}_1$, and the input space, $\mathbf{x}$) and allocate one epoch per space. We then fine-tune on the input space. All models trained directly in the input space utilize standard data augmentations (see Appendix for further details). Models trained with the curriculum are initialized randomly; other models are initialized via kmeans or a combination of kmeans and fitting a local Factor Analyzer as in (46). When training a hierarchical model, we often find it helpful to include entropy regularizations across trajectories.

5.1 Synthetic Data

We demonstrate the effectiveness and weaknesses of CPMM on two toy datasets. The first dataset consists of two concentric, broken circles. The radii of both circles are chosen to differ by 10% and have similar thicknesses. The second dataset consists of a noisy five-armed spiral that begins near the origin before curving outward. This dataset is intentionally constructed with "outliers" (points between the arms without a clear cluster identity) to assess how the CPMM will handle examples "far" from the manifold. Both CPMMs are trained using a curriculum.

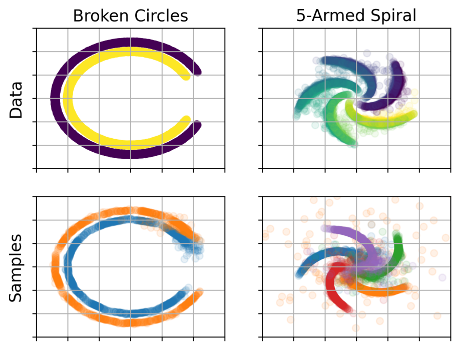


Figure 3: Two synthetic datasets designed to assess the limitations of the CPMM.

Figure 3 contains true data points in the top row and samples drawn from the CPMM on the bottom row. Since the model is trained unsupervised, we use different colors between true labels and cluster labels (e.g., between the top and bottom row). We see that the CPMM does a reasonable job of capturing the general manifolds of both datasets. However, the model seems to struggle with the end points of the trajectory and the samples are less precise than during the other portions of the trajectory. We observe this trend on all datasets: the CPMM often struggles with endpoints, typically the early pseudotimes. In the spiral data, we see that the model has attempted to capture the outliers, though they have larger spread than the true outliers and it attributes them all to the same cluster.

5.2 Images

To process image datasets, we first rescale the input pixels to be between zero and one and then apply an element-wise logit transform, a standard preprocessing technique (e.g., (22)). We perform these transformations so that the "input" data has support over $\mathbb{R}$. These transformations are applied before we train any model and the corresponding $\log\det \texttt{Jacobian}$ is accounted for when estimating bits per dimension as in a normalizing flow.

We test training CPMMs on MNIST (10) and Fashion-MNIST (50) and compare to two different GMM baselines along with several standard likelihood models and clustering models. CPMM NODEs are constructed using CNNs with a depth of 3, a hidden channel width of 64, and utilize $\sin$ activations. We additionally augment the state space by a factor of four and explicitly condition the ODEs on pseudotime by concatenating it as an extra channel. To avoid degeneracies, we soft-clip $\log d$ so that the minimum value cannot be less than -6.

We consider two baseline GMMs for comparison. In the first model, we choose the total number of components equal to the number of clusters (e.g., 10). This simple procedure allows for easy cluster assignments; each component corresponds to exactly one cluster. In the second case, we train a GMM with 250 components since this corresponds exactly to the number of components extracted across trajectories when using a CPMM. However, attributing components to clusters is less obvious in this case. We choose to use spectral clustering (16) over components based on the Fréchet distance over

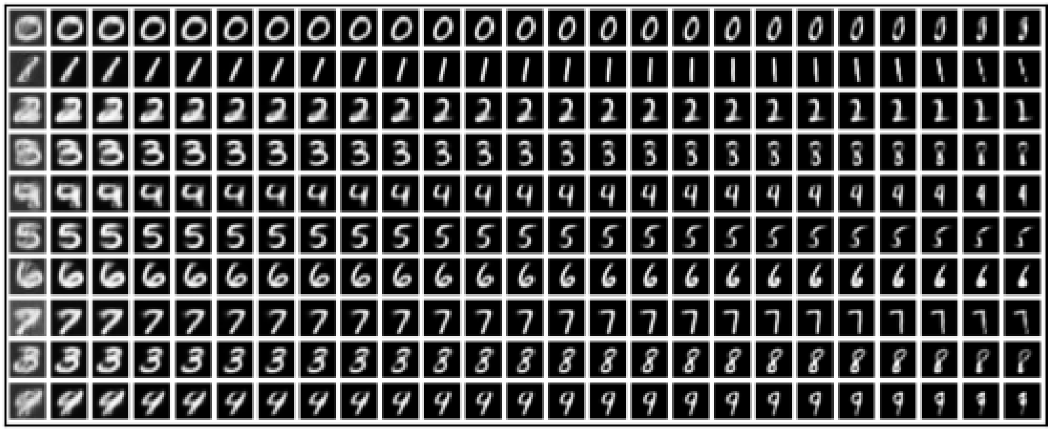


Figure 4: Means from a hierarchical CPMM trained on MNIST with a space-based curriculum. Each trajectory is evaluated at evenly spaced pseudotimes between zero and one and displayed from left to right. Despite the 3/8 and 4/9 confusion, the model does an excellent job of learning smoothly-varying transitions between digits. The initial component (far left cases) are never the most likely component and could be discarded.

Gaussians (13). Specifically, we attribute examples to components and components to clusters based on a spectral clustering model that constructs an affinity matrix through the Fréchet distance. Both models are initialized and trained as discussed in Sec. 5.

Table 1: Bits per dimension (BPD) and clustering accuracy (Acc.) for several image datasets.

	MNIST		Fashion-MNIST	
	BPD	Acc.	BPD	Acc.
Matched GMM	6.61	61.75	6.58	55.00
GMM w/ S.Clust.	5.21	48.91	5.71	48.52
EiNet (Matched)	2.19	-	4.18	-
EiNet (Large)	1.95	-	3.98	-
GMM w/ Curr.	2.61	64.47	4.35	58.41
CPMM	2.21	80.07	3.91	65.01
Teacher	1.04	-	2.85	-
FFJORD	1.01	-	2.75	-
VADE	-	94.5	-	57.8
SPC	-	99.21	-	67.94
DLS	-	97.6	-	69.3

Table 1 summarizes the results for the different mixture models in juxtaposition to standard baselines (1; 11; 35; 42). The matched GMM models (one component per cluster) achieves surprisingly high clustering accuracies but subpar BPD. Unsurprisingly, introducing more components (GMM w/ S.Clust.) improves the BPD. However, spectral clustering across components is only marginally successful and the clustering accuracy drops relative to the baseline GMM. We additionally train two EiNets (42), a type of SPN. The first EiNet (Matched) is constrained to have the same total number of parameters (not components) as the the CPMM. The second EiNet (Large) utilizes an order of magnitude more parameters. Finally, we train a standard GMM with 250 components that additionally utilizes the curriculum. The CPMM trained with the curriculum enjoys significantly improved BPD and clustering accuracy relative to the GMM baselines, better performance compared to the matched EiNet and standard GMM with the curriculum, and similar performance against the large EiNet. (See Appendix for additional ablation experiments.) Our results indicate that our proposed methodology closes the considerable gap between mixture models and modern neural baselines (listed in the bottom of Table 1). An especially notable case is the Fashion MNIST clustering accuracy, where the CPMM results in comparable-to-better than the standard baselines. Thus, CPMM allows one to retain the interpretability and inference speed of mixture models with improvements in modeling performance over tradition mixture approaches.

Interpretability Figure 4 shows the trajectories learned by a CPMM where we have manually ordered the clusters. The figure shows smooth transitions along the trajectory for all digits, in general transitioning from fatter, curvier digits to slimmer, straight digits. The early pseudotimes (far left) are never the most likely component and could be excluded for a slight increase in the BPD. We have not done so here for the sake of transparency and to avoid arbitrary post-hoc operations. This particular model does an excellent job of isolating different digits into different trajectories with the exception of some 3/8 and 4/9 confusion that results in a clustering accuracy of 80%. This confusion is understandable given the similarities in the digit pairs but more importantly, this analysis highlights the *interpretability* of our model. That is, it is simple to resolve the model's confusion from the trajectories since CPMM directly operates over input features. In contrast to more opaque clustering models, a quick visual inspection reveals CPMM's partioning of the space. Similar results and discussion can be found for Fashion-MNIST in the Appendix.

OOD Detection Finally, we test the CPMM's ability to distinguish between different datasets based on likelihood. This is a known shortcoming of neural likelihood models (26; 38) where the networks unfortunately predict that simpler datasets are more likely than the data the model was trained on; e.g., MNIST instances yield a higher likelihood than Fashion-MNIST instances for models trained on Fashion-MNIST. This surprising result limits the applicability of modern likelihood models for detecting out-of-distribution (OOD) and anomalous instances, despite their intuitive capabilities. Table 2 illustrates the performance of our model against a CNF for detecting instances that are out-of-distribution when trained on a distinct inlier distribution with a simple thresholding of likelihoods. Results are given as areas under the receiver operating characteristic and precision/recall curves as a percent. Higher is better. Although both models perform nearly perfectly when MNIST is in-distribution and Fashion-MNIST is out-of-distribution, the CPMM performs quite well on the reverse problem where the CNF fails miserably. Notwithstanding a gap in the likelihood that is obtainable with CPMM as compared to CNF (Table 1), this experiment illustrates an advantage to our simplified approach of directly modeling the input data through a mixture of components.

Table 2: OOD Detection via Likelihood Thresholding

		CPMM		FFJORD	
In	Out	AUROC	AUPR	AUROC	AUPR.
MNIST	Fashion	99.95	99.95	99.95	99.95
Fashion	MNIST	93.38	93.28	8.98	31.65

6 CONCLUSIONS

Mixture models are interpretable, well-behaved, computationally-cheap likelihood models that are, unfortunately, increasingly difficult to employ as the data complexity and dimensionality increases. These difficulties are largely due to the relatively simple components and sensitivities to model initializations. Neural networks have proven themselves to be incredibly powerful models but their performance is inexplicable and they exhibit bizarre failure modes. We have proposed a hierarchical method to parameterize a continuum of mixture models from neural ODEs to create a rich, multi-modal density over disparate partitions, essentially constructing a mixture of mixtures where the outer mixture encompasses different data regions and the inner mixture can be arbitrarily complex.

We have additionally innovated a curriculum that alleviates many of the difficulties associated with initialization. The curriculum utilizes an annealing process from a prescribed latent space chosen for its simplicity towards the true, nuanced data space. The transitions are learned via maximum likelihood estimation through a continuous normalizing flow. Since the initial distribution is known, it is trivial to initialize a model that is well-matched. Finally, we sample a finite number of components from the dynamic NODE to achieve an effective, light-weight model that significantly outperforms mixtures with a similar number of components, indicating that the combination of the curriculum and dynamic system allows for a more efficient use of the available components than traditional methods.

While this method provides notable improvements in terms of likelihood predictions and clustering accuracies in comparison to typical mixture methods, it unsurprisingly, fall short of neural network methods that utilize learned latent spaces on the primary metrics typically used in model assessment. However, we do not exhibit the same brittleness (e.g., larger likelihoods on OOD data), require significantly less computation (once trained), and are completely interpretable. We additionally have the ability to arbitrarily condition our models if a subset of features is known a priori. Similarly, we can easily marginalize a subset of features to produce a completely consistent model without requiring any additional training. Neither of these benefits are available to pure neural network models.

REFERENCES

[1] A. F. Agarap and A. P. Azcarraga. Improving k-means clustering performance with disentangled internal representations. *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[3] P. Blanchard, R. Devaney, and G. Hall. *Differential Equations*. Thomson Brooks/Cole, 2006.

[4] D. Böhning, W. Seidel, M. Alfò, B. Garel, V. Patilea, and G. Walther. Advances in mixture models. *Comput. Stat. Data Anal.*, 51:5205–5210, 2007.

[5] M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

[6] C. Chen, G. Li, R. Xu, T. Chen, M. Wang, and L. Lin. Clusternet: Deep hierarchical cluster network with rigorously rotation-invariant representation for point cloud analysis. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4989–4997, 2019.

[7] R. T. Q. Chen, J. Behrmann, D. K. Duvenaud, and J.-H. Jacobsen. Residual flows for invertible generative modeling. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[8] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

[9] A. Darwiche. A differential approach to inference in bayesian networks. 50(3), 2003.

[10] L. Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[11] F. Ding, F. Luo, and Y. Yang. Double cycle-consistent generative adversarial network for unsupervised conditional generation, 2019.

[12] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[13] D. Dowson and B. Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

[14] H. Eghbal-zadeh, W. Zellinger, and G. Widmer. Mixture density generative adversarial networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5813–5822, 2019.

[15] e. Falcon, WA. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[16] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41(1):176–190, 2008.

[17] M. M. Gao, C. Tai-hua, and X. xiang Gao. Application of gaussian mixture model genetic algorithm in data stream clustering analysis. *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, 3:786–790, 2010.

[18] A. E. Gelfand, A. Kottas, and S. N. MacEachern. Bayesian nonparametric spatial modeling with dirichlet process mixing. *Journal of the American Statistical Association*, 100(471):1021–1035, 2005.

[19] Z. Ghahramani and G. E. Hinton. The EM algorithm for mixtures of factor analyzers. 1996.

[20] I. J. Goodfellow, Y. Bengio, and A. C. Courville. Deep learning. *Nature*, 521:436–444, 2015.

[21] D. Görür and C. E. Rasmussen. Dirichlet process gaussian mixture models: Choice of the base distribution. *Journal of Computer Science and Technology*, 25:653–664, 2010.

[22] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, and D. Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019.

[23] D. Ha, A. Dai, and Q. V. Le. Hypernetworks, 2016.

[24] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning. 2001.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[26] D. Hendrycks, M. Mazeika, and T. Dietterich. Deep anomaly detection with outlier exposure. In *International Conference on Learning Representations*, 2019.

[27] H. Holzmann, A. Munk, and T. Gneiting. Identifiability of finite mixtures of elliptical distributions. *Scandinavian Journal of Statistics*, 33, 2006.

[28] P. Izmailov, P. Kirichenko, M. Finzi, and A. Wilson. Semi-supervised learning with normalizing flows. *ArXiv*, abs/1912.13025, 2020.

[29] P. Jaini, P. Poupart, and Y. Yu. Deep homogeneous mixture models: Representation, separation, and approximation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[30] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *IJCAI*, 2017.

[31] L. Jiao, T. Denoeux, Z. Liu, and Q. Pan. Egmm: an evidential version of the gaussian mixture model for clustering. *ArXiv*, abs/2010.01333, 2020.

[32] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[33] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10236–10245, 2018.

[34] M. T. Law, R. Urtasun, and R. S. Zemel. Deep spectral clustering learning. In *ICML*, 2017.

[35] L. Mahon and T. Lukasiewicz. Selective pseudo-label clustering. In S. Edelkamp, R. Möller, and E. Rueckert, editors, *KI 2021: Advances in Artificial Intelligence*, pages 158–178, Cham, 2021. Springer International Publishing.

[36] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan. Clustergan : Latent space clustering in generative adversarial networks. *ArXiv*, abs/1809.03627, 2019.

[37] P. Müller and F. A. Quintana. Nonparametric Bayesian Data Analysis. *Statistical Science*, 19(1):95 – 110, 2004.

[38] E. Nalisnick, A. Matsukawa, Y. W. Teh, and B. Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using typicality. *arXiv preprint arXiv:1906.02994*, 2019.

[39] J. Oliva, A. Dubey, M. Zaheer, B. Poczos, R. Salakhutdinov, E. Xing, and J. Schneider. Transformation autoregressive networks. In *International Conference on Machine Learning*, pages 3898–3907, 2018.

[40] G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[42] R. Peharz, S. Lang, A. Vergari, K. Stelzner, A. Molina, M. Trapp, G. Van Den Broeck, K. Kersting, and Z. Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7563–7574. PMLR, 13–18 Jul 2020.

[43] R. Peharz, A. Vergari, K. Stelzner, A. Molina, X. Shao, M. Trapp, K. Kersting, and Z. Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In R. P. Adams and V. Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 334–344. PMLR, 22–25 Jul 2020.

[44] G. G. P. F. Pires and M. A. T. Figueiredo. Variational mixture of normalizing flows. *ArXiv*, abs/2009.00585, 2020.

[45] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690, 2011.

[46] E. Richardson and Y. Weiss. On gans and gmms. In *NeurIPS*, 2018.

[47] U. Shaham, K. Stanton, H. Li, B. Nadler, R. Basri, and Y. Kluger. Spectralnet: Spectral clustering using deep neural networks. *ArXiv*, abs/1801.01587, 2018.

[48] A. van den Oord, N. Kalchbrenner, L. Espeholt, k. kavukcuoglu, O. Vinyals, and A. Graves. Conditional image generation with pixelcnn decoders. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[49] C. Viroli and G. McLachlan. Deep gaussian mixture models. *Statistics and Computing*, 29:43–51, 2019.

[50] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

[51] J. Xie, R. B. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, 2016.

[52] D. Zhang, Y. Sun, B. Eriksson, and L. Balzano. Deep unsupervised clustering using mixture of autoencoders. *ArXiv*, abs/1712.07788, 2017.

A NUMERICAL INTEGRATION

The typical strategy to train a likelihood model is to optimize based on the predicted *log-likelihood* and not on the likelihood directly. In fact, when training a Gaussian mixture model, we typically calculate the log-likelihood of each component separately prior to utilizing the numerically-stable $\log \sum \exp$ operator instead of performing the $\exp$, followed by the $\sum$, followed by the $\log$. This joint version stably handles difficulties resulting from over- or under-flow when the log-likelihood of any single component is very (large) small.

When attempting to evaluate the log-likelihood for a continuum of components we should utilize a similar $\log \int \exp$:

$$\log p(\mathbf{x}) = \frac{-M}{2} \log(2\pi) + \log \int_0^1 \exp\Big(-\log|\Sigma(s)|/2 - (\mathbf{x} - \mu(s))^T \Sigma^{-1}(s) (\mathbf{x} - \mu(s))/2 + \log \pi(s) \Big)\, ds.$$

Unfortunately, none of the available black-box integrators match this form. This requires that we utilize a naive implementation of the $\log \int \exp$ operator. In practice, we find this sufficient to train the model for several epochs without issue but the ODE becomes increasingly stiff before ultimately underflowing and "`NaN`ing" out. A better strategy to estimate a continuous mixture model will require a new black-box integrator which we consider an excellent avenue for future work.

B AUGMENTATIONS

For all image datasets we added uniform random noise between zero and one *prior to rescaling*, e.g., a pixel with an 8bit value of 34 would then be distributed uniformly between 34 and 35. For color images, we additionally perform standard random translations and horizontal flips. For tabular data, we add Gaussian noise to each feature such that the standard deviation of the noise is 1-10% of the spread of the data itself, e.g., we normalize each feature to be zero mean and unit standard deviation, add Gaussian noise scaled by 0.01 to 0.1 and unnormalize that data.

When training with the curriculum (see Sec. 3.3), we utilize similar augmentations prior to transforming into the various spaces such that the augmentation is shared across spaces. When training the CPMM student, we then add a small mount of Gaussian noise (standard deviation of 0.01) regardless of the space, including when fine-tuning on the input space.

C MNIST ABLATIONS

We experiment with different types of initializations and modifications of the CPMM on MNIST. In particular, we test random initializations and kmeans-based initializations. We also explore restricting the diagonal portion of the covariance within each MFA to be constant (though still learnable and still with the low rank updates, A) and attempt to utilize a separate ODE for each cluster.

Table 3: MNIST Ablation

Init.	Diagonal	BPD	Acc.
Random	Constant	7.67	48.83
kmeans	Constant	7.56	64.93
kmeans	Variable	4.49	46.42
kmeans*	Constant	7.55	71.22

Table 3 contains some representative runs attempting to train the CPMM without guidance from the teacher. We find that training the models with a variable diagonal often results in the model degenerating and failing to train. In fact, training with a variable diagonal and a random initialization tends to fail after a few epochs and is therefore excluded from the table. Utilizing a kmeans initialization produces moderately good BPD (better than standard GMM training methods) but slightly worsened accuracies. Restricting the model to use a constant diagonal greatly stabilizes training and can produce reasonably good clustering accuracies. However, this strong assumption on the covariance results in very poor BPD. This is not surprising, as the model must raise the variance of pixels that would normally be very small (background pixels) to sufficiently capture the spread elsewhere. Finally, the bottom row in Table 3 contains results using a kmeans initialization with a constant diagonal but utilizes a separate ODE for each trajectory. This added flexibility results in virtually no improvement to the BPD but does provide a noticeable increase in accuracy. We find that training separate ODEs per cluster results in a significant increase to run time.

The results in Table 3 were chosen as representative values across many different runs. In general, it is possible to trade accuracy for BPD in any given row by adjust other hyperparameters or seeds. Utilizing the curriculum learning process helps to stabilize training and allows us to simultaneously improve the generative and discriminative portions of the model.

D FASHION MNIST RESULTS

Figure 5 contains the trajectories from a curriculum-based CPMM trained on Fashion-MNIST. Similar to MNIST, the trajectories show smooth variations between means. Many of the trajectories contain visible evolutions from start to finish, i.e., `bags` begin without handles but end with handles or shoes evolve from boots to stilletos. However, some of the fine detail is missing from the trajectories, in particular for the `shirts`, `pullovers`, and `coats` classes.

As in MNIST, we can clearly see why the model achieves 65.74% clustering accuracy. The `shirts`, `pullovers`, and `coats` classes all produce very similar trajectories. Similarly, the `sandals`, `sneakers`, and `ankle boots` classes see considerable overlap. Abstractly, this information can be extracted from any clustering model based on the confusion matrix. But, thanks to the interpretability of the mixture models, the decision process and resulting confusion is clear.

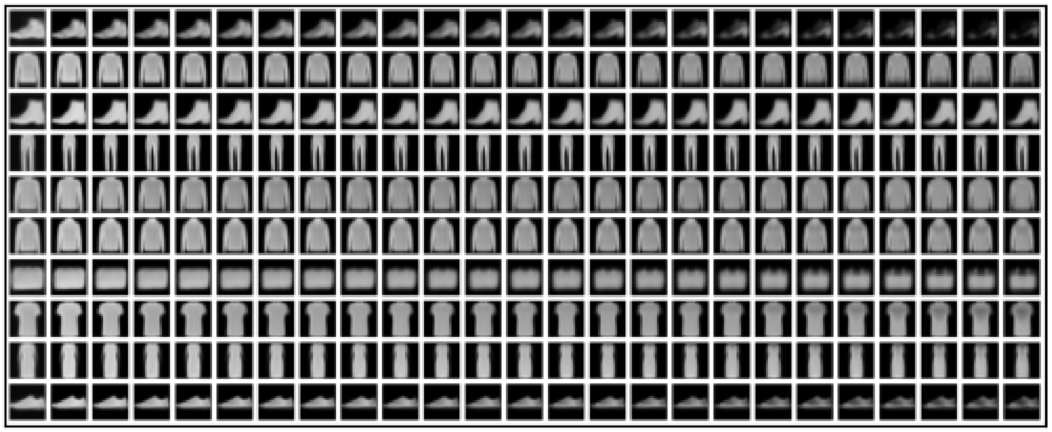


Figure 5: Means from a hierarchical CPMM trained on Fashion-MNIST with a space-based curriculum. Each trajectory is evaluated at evenly spaced pseudotimes between zero and one and displayed from left to right. The initial component (far left cases) are never the most likely component and could be discarded.