Neural Processing Letters (2021) 53:4331–4348
https://doi.org/10.1007/s11063-021-10599-3

Gradient-Based Training of Gaussian Mixture Models for High-Dimensional Streaming Data

Alexander Gepperth[1] · **Benedikt Pfülb**[1]

Accepted: 15 July 2021 / Published online: 17 August 2021

Abstract
We present an approach for efficiently training Gaussian Mixture Model (GMM) by Stochastic Gradient Descent (SGD) with non-stationary, high-dimensional streaming data. Our training scheme does not require data-driven parameter initialization (e.g., k-means) and can thus be trained based on a random initial state. Furthermore, the approach allows mini-batch sizes as low as 1, which are typical for streaming-data settings. Major problems in such settings are undesirable local optima during early training phases and numerical instabilities due to high data dimensionalities. We introduce an adaptive annealing procedure to address the first problem, whereas numerical instabilities are eliminated by an exponential-free approximation to the standard GMM log-likelihood. Experiments on a variety of visual and non-visual benchmarks show that our SGD approach can be trained completely without, for instance, k-means based centroid initialization. It also compares favorably to an online variant of Expectation-Maximization (EM)—stochastic EM (sEM), which it outperforms by a large margin for very high-dimensional data.

Keywords Gaussian Mixture Model · Stochastic Gradient Descent · High-Dimensional Streaming Data

1 Introduction

This contribution focuses on Gaussian Mixture Model (GMM), which represent a probabilistic unsupervised model for clustering and density estimation, allowing sampling and outlier detection. GMMs have been used in a wide range of scenarios, see [18]. Commonly, free parameters of a GMM are estimated using the Expectation-Maximization (EM) algorithm [7], which does not require learning rates and automatically enforces all GMM constraints. A popular online variant is stochastic EM [2], which can be trained mini-batch wise and is thus more suited for large datasets or streaming data.

✉ Benedikt Pfülb
benedikt.pfuelb@cs.hs-fulda.de

Alexander Gepperth
alexander.gepperth@cs.hs-fulda.de

[1] Fulda University of Applied Sciences, Leipziger Str. 123, 36037 Fulda, Germany

1.1 Motivation

Intrinsically, EM is a batch-type algorithm. Therefore, memory requirements can become excessive for large datasets. In addition, streaming-data scenarios require data samples to be processed one by one, which is impossible for a batch-type algorithm. Moreover, data statistics may be subject to changes over time (concept drift/shift), to which the GMM should adapt. In such scenarios, an online, mini-batch type of optimization such as SGD is attractive since it can process samples one by one, has modest, fixed memory requirements, and can adapt to changing data statistics.

1.2 Related Work

Online EM is a technique for performing EM mini-batch wise, allowing to process large datasets. One branch of previous research [4,16,20] has been devoted to the development of stochastic Expectation-Maximization (sEM) algorithms that reduce to the original EM method in the limit of large batch sizes. The variant presented in [2] is widely used due to its simplicity and efficiency for large datasets. Such approaches come at the price of additional hyper-parameters (e.g., step size, mini-batch size, step size reduction), thus removing a key advantage of EM over SGD. Another approach is to modify the EM algorithm itself by, e.g., including heuristics for adding, splitting and merging centroids [3,10,15,24,26,29,30]. This allows GMM-like models to be trained by presenting one sample after another. Models of this type work well in several application scenarios, but their learning dynamics are impossible to analyze mathematically. They also introduce a high number of parameters. Apart from these works, some authors avoid the issue of extensive datasets by determining smaller "core sets" of representative samples and performing vanilla EM [11].

SGD for Training GMMs has, as far as we know, been recently treated only in [13,14]. In this body of work, GMM constraint enforcement is ensured by using manifold optimization techniques and re-parameterization/regularization, thereby introducing additional hyper-parameters. The issue of local optima is side-stepped by a k-means type centroid initialization, and the used datasets are low-dimensional (36 dimensions).

Annealing and Approximation Approaches for GMMs were proposed in [8,22,23,28]. However, the regularizers proposed in [22,28] significantly differ from our scheme. GMM log-likelihood approximations, similar to the one used here, are discussed in, e.g., [8,23], but only in combination with EM training. A similar "hard assignment" approximation is performed in [27].

GMM Training in High-Dimensional Spaces is discussed in several publications: A conceptually very interesting procedure is proposed in [12]. It exploits the properties of high-dimensional spaces in order to achieve learning with a number of samples that is polynomial in the number of Gaussian components. This is difficult to apply in streaming settings, since higher-order moments need to be estimated beforehand, and also because the number of samples is usually unknown. Training GMM-like lower-dimensional factor analysis models by SGD on high-dimensional image data is successfully demonstrated in [25]. This approach avoids numerical issues, but, again, sidesteps the local optima issue by using k-means initialization. The numerical issues associated with log-likelihood computation in high-dimensional spaces are generally mitigated by using the "logsumexp" trick [21], which is, however, insufficient for ensuring numerical stability for particularly high-dimensional data, such as images.

1.3 Goals and Contributions

The goals of this article are to establish GMM training by SGD as a simple and scalable alternative to sEM in streaming scenarios with potentially high-dimensional data. The main novel contributions are:

- a proposal for numerically stable GMM training by SGD that outperforms sEM for high data dimensionalities,
- an automatic annealing procedure that ensures SGD convergence without prior knowledge of the data (**no** k-means initialization) which is beneficial for streaming data,
- a computationally efficient method for enforcing all GMM constraints in SGD,
- a convergence proof for the annealing procedure.

Additionally, we provide a TensorFlow implementation.[1]

2 Gaussian Mixture Models

GMMs are probabilistic models that try to explain the observed data $X = \{\boldsymbol{x}_n\}$ by expressing their density as a weighted mixture of K Gaussian component densities $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{P}_k) \equiv \mathcal{N}_k(\boldsymbol{x})$:

$$p(\boldsymbol{x}_n) = \sum_{k=1}^{K} \pi_k \mathcal{N}_k(\boldsymbol{x}_n). \tag{1}$$

Here, we parameterize Gaussian densities by precision matrices $\boldsymbol{P}_k = \boldsymbol{\Sigma}_k^{-1}$ instead of covariances $\boldsymbol{\Sigma}_k$. The component weights π_k represent another set of GMM parameters, which modulate the overall influence of each Gaussian density. For a derivation of Eq. 1, we must introduce the probabilistic foundations of GMMs. GMMs assume that each observed data sample $\{\boldsymbol{x}_n\}$ is drawn from one of the Gaussian component densities $\mathcal{N}_k$. The selection of this component density is assumed to depend on an unobserved (and unobservable) *latent variable* $z_n \in \{1, \ldots, K\}$ which follows an unknown distribution. This is formalized for a GMM with K components by formulating the *complete-data likelihood* for a single data sample as:

$$p(\boldsymbol{x}_n, z_n) = \pi_{z_n} \mathcal{N}_{z_n}(\boldsymbol{x}_n), \tag{2}$$

Since the latent variables are, by construction, unobservable, we must marginalize them out of Eq. 2 in order to obtain an expression suitable for optimization. This gives us the *incomplete-data likelihood* for a single data sample $\boldsymbol{x}_n$:

$$p(\boldsymbol{x}_n) = \sum_{k=1}^{K} p(\boldsymbol{x}_n, k), \tag{3}$$

which depends on observable quantities only. Please compare this to Eq. 1. The incomplete-data likelihood for all samples is thus given by:

$$p(X) = \prod_n p(\boldsymbol{x}_n) = \prod_n \sum_k p(\boldsymbol{x}_n, k) = \prod_n \sum_k \pi_k \mathcal{N}_k(\boldsymbol{x}_n), \tag{4}$$

[1] https://gitlab.cs.hs-fulda.de/ML-Projects/sgd-gmm.

where we have inserted Eq. 2 in the last step. Passing to the log-domain (as it is common for probabilistic models), we obtain the *total incomplete-data log-likelihood* for all observed data samples:

$$\mathcal{L} = \log p(X) = \sum_n \log \sum_k \pi_k \mathcal{N}_k(\boldsymbol{x}_n). \tag{5}$$

The function $\mathcal{L}$ contains only observable quantities and is a suitable loss function for optimization. For convenience and numerical stability, the sum is usually replaced by an expectation, and we follow this convention:

$$\mathcal{L} = \mathbb{E}_n \left[\log \sum_k \pi_k \mathcal{N}_k(\boldsymbol{x}_n) \right]. \tag{6}$$

Please note that $\mathcal{L}$ represents the likelihood of the observed data under the GMM with current parameters, and must therefore be *maximized* to obtain a better explanation of the data.

2.1 GMM Constraint Enforcement for SGD

GMMs require the mixture weights to be normalized: $\sum_k \pi_k = 1$ and the precision matrices to be positive definite: $\boldsymbol{x}^\top \boldsymbol{P}_k \boldsymbol{x} \geq 0 \ \forall \boldsymbol{x}$. These constraints must be explicitly enforced after each SGD step:

Weights π_k are adapted according to [13], which replaces them by other free parameters ξ_k from which the π_k are computed so that normalization is ensured:

$$\pi_k = \frac{\exp(\xi_k)}{\sum_j \exp(\xi_j)}. \tag{7}$$

Diagonal Precision Matrices are re-parameterized as $\boldsymbol{P}_k = \boldsymbol{D}_k^2$, with diagonal matrices $\boldsymbol{D}_k$ (Cholesky decomposition). They are, therefore, guaranteed to be positive definite. Hence, $\det \boldsymbol{\Sigma}_k = \det \boldsymbol{P}_k^{-1} = \left(\det(\boldsymbol{D}_k^2)\right)^{-1} = \left(\mathrm{Tr}(\boldsymbol{D}_k)\right)^{-2}$ can be computed efficiently. Since we are dealing with high-dimensional data, precision matrices are always taken to be diagonal, since full matrices would be prohibitive w.r.t. memory consumption and the number of free parameters.

Full Precision Matrices are treated here for completeness' sake, since they are infeasible for high-dimensional data. We represent them as a spectral decomposition into eigenvectors $\boldsymbol{v}_i$ and eigenvalues λ_i^2: $\boldsymbol{P}_k = \sum_i \lambda_i^2 \boldsymbol{v}_i \boldsymbol{v}_i^\top$, which ensures positive-definiteness. This can be seen from $\det \boldsymbol{\Sigma}_k = \det \boldsymbol{P}_k^{-1} = \prod_i \lambda_i^{-2}$. In order to maintain a correct representation of eigenvectors, these have to be orthonormalized after each SGD step.

2.2 Max-Component Approximation for GMM

The log-likelihood Eq. 5 is difficult to optimize by SGD due to numerical problems (mainly underflows and resulting divisions by zero) for high data dimensionalities. This is why we intend to find a lower bound that we can optimize instead. A simple scheme is given by

$$\begin{aligned} \mathcal{L} &= \mathbb{E}_n \left[\log \sum_k \pi_k \mathcal{N}_k(\boldsymbol{x}_n) \right] \geq \mathbb{E}_n \left[\log \max_k \left(\pi_k \mathcal{N}_k(\boldsymbol{x}_n) \right) \right] \\ &= \hat{\mathcal{L}} = \mathbb{E}_n \left[\log \left(\pi_{k^*} \mathcal{N}_{k^*}(\boldsymbol{x}_n) \right) \right] \end{aligned} \tag{8}$$

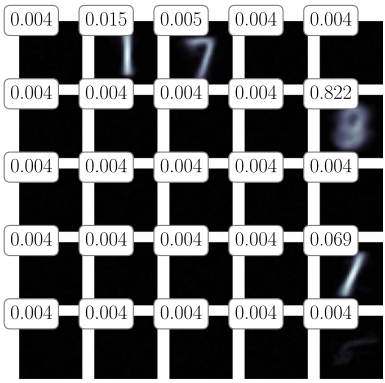


Fig. 1 A sparse-component-solution with superimposed component weights π_k, obtained when performing naive SGD on MNIST (a dataset consisting of handwritten digits, see Sect. 3.1)

where $k^* = \arg\max_k \pi_k \mathcal{N}_k(\boldsymbol{x}_n)$. This is what we call the *max-component approximation* of Eq. 8. In contrast to the lower bound that is constructed for EM-type algorithms, our bound is usually not tight. Nevertheless, we will demonstrate later that it is a very good approximation when data are high-dimensional. The advantage of $\hat{\mathcal{L}}$ is the elimination of exponentials causing numerical instabilities. The "logsumexp" trick is normally employed with GMMs to rectify this by factoring out the largest component probability $\mathcal{N}_{k^*}$. This mitigates but does not avoid numerical problems when distances are high, a common occurrence for high data dimensions. To give an example: we normalize the component probability $\mathcal{N}_k = e^{-101}$ (using 32-bit floats) by the highest probability $\mathcal{N}_{k^*} = e^3$, and we obtain $\frac{\mathcal{N}_k}{\mathcal{N}_{k^*}} = e^{-104}$, which produces an underflow respectively NaN values.

2.3 Undesirable Local Optima in SGD Training

A crucial issue when optimizing $\hat{\mathcal{L}}$ (and indeed $\mathcal{L}$ as well) by SGD without k-means initialization concerns undesirable local optima. Most notable are the **single/sparse-component solutions**, see Fig. 1. They are characterized by one or several components $\{k_i\}$ having large weights, with centroid and precision matrices given by the mean and covariance of a significant subset $\boldsymbol{X}_{k_i} \subset \boldsymbol{X}$ of the data $\boldsymbol{X}$: $\pi_{k_i} \gg 0$, $\boldsymbol{\mu}_{k_i} = \mathbb{E}[\boldsymbol{X}_{k_i}]$, $\boldsymbol{\Sigma}_{k_i} = \mathrm{Cov}(\boldsymbol{X}_{k_i})$, whereas the remaining components k are characterized by $\pi_k \approx 0$, $\boldsymbol{\mu}_k = \boldsymbol{\mu}(t=0)$, $\boldsymbol{P}_k = \boldsymbol{P}(t=0)$. Thus, these unconverged components are almost never the best-matching units (BMUs) k^*. The max-operation in $\hat{\mathcal{L}}$ causes gradients like $\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{\mu}_k}$ to contain δ_{kk^*}:

$$\begin{aligned}
\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{\mu}_k} &= \mathbb{E}_n \left[\boldsymbol{P}_k \left(\boldsymbol{x}_n - \boldsymbol{\mu}_k \right) \delta_{kk^*} \right] \\
\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{P}_k} &= \mathbb{E}_n \left[\left((\boldsymbol{P}_k)^{-1} - (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \right) \delta_{kk^*} \right] \\
\frac{\partial \hat{\mathcal{L}}}{\partial \pi_k} &= \pi_k^{-1} \mathbb{E}_n \left[\delta_{kk^*} \right].
\end{aligned} \tag{9}$$

This implies that the gradients are non-zero only for the BMU k^*. Thus, the gradients of unconverged components vanish, meaning that component parameters remain in their unconverged state.

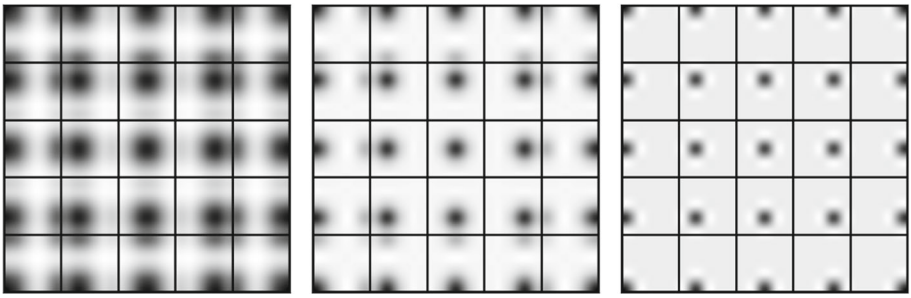


Fig. 2 Visualization of Gaussian smoothing filters $\boldsymbol{g}_k$ used in annealing, of width σ, for three different values of σ. The $\boldsymbol{g}_k$ are placed on a 2D grid, darker pixels indicate larger values. Over time, $\sigma(t)$ is reduced (middle and right pictures) and the Gaussians approach a delta peak, thus recovering the original, non-annealed loss function. Note that the grid is considered periodic in order to avoid boundary effects, so the $\boldsymbol{g}_k$ are themselves periodic

2.4 Annealing Procedure for Avoiding Local Optima

Our approach for avoiding sparse-component solutions is to punish their characteristic response patterns by replacing $\hat{\mathcal{L}}$ by the *smoothed max-component log-likelihood* $\hat{\mathcal{L}}^\sigma$:

$$\begin{aligned}\hat{\mathcal{L}}^\sigma &= \mathbb{E}_n \max_k \left[\sum_j \boldsymbol{g}_{kj}(\sigma) \log\left(\pi_j \mathcal{N}_j(\boldsymbol{x}_n)\right)\right] \\ &= \mathbb{E}_n \sum_j \boldsymbol{g}_{k^* j}(\sigma) \log\left(\pi_j \mathcal{N}_j(\boldsymbol{x}_n)\right).\end{aligned} \tag{10}$$

Regarding its interpretation, we are assuming that the K GMM components are arranged in a quadratic 2D grid of size $\sqrt{K} \times \sqrt{K}$. Equally, each $\boldsymbol{g}_k$ is interpreted as 2D grid of size $\sqrt{K} \times \sqrt{K}$, (see Fig. 2), with values given by a periodically continued 2D Gaussian centered on component k. With this interpretation, Eq. 10 represents a 2D convolution with periodic boundary conditions (in the sense used in image processing) of the $\log(\pi_k \mathcal{N}_k(\boldsymbol{x}))$ by a smoothing filter whose width is controlled by σ.

Thus, Eq. 10 is maximized if the log-probabilities follow a uni-modal Gaussian profile of spatial variance $\sim \sigma^2$, which heavily punishes single/sparse-component solutions that have a locally delta-like response. A 1D grid for annealing, together with 1D smoothing filters, was verified to fulfill this purpose as well. We chose 2D because it allows for an easier visualization while incurring an identical computational cost.

Annealing starts with a large value of $\sigma(t) = \sigma_0$ and reduces it over time to an asymptotic small value of $\sigma = \sigma_\infty$, thus, smoothly transitioning from $\hat{\mathcal{L}}^\sigma$ in Eq. 10 into $\hat{\mathcal{L}}$ in Eq. 8.

Annealing Control regulates the decrease of σ. This quantity defines an effective upper bound on $\hat{\mathcal{L}}^\sigma$ (see Sect. 2.6 for a proof). An implication is that the loss will be stationary once this bound is reached, which we consider a suitable indicator for reducing σ. We implement an annealing control that sets $\sigma \leftarrow 0.9\sigma$ whenever the loss is considered sufficiently stationary. Stationarity is detected by maintaining an exponentially smoothed average log-likelihood $\ell(t) = (1-\alpha)\ell(t-1) + \alpha \hat{\mathcal{L}}^\sigma(t)$ on time scale α. Every $\frac{1}{\alpha}$ iterations, we compute the fractional increase of $\hat{\mathcal{L}}^\sigma$ as

$$\Delta = \frac{\ell(t) - \ell(t - \alpha^{-1})}{\ell(t - \alpha^{-1}) - \hat{\mathcal{L}}^\sigma(t = 0)} \tag{11}$$

and consider the loss stationary iff $\Delta < \delta$ (the latter being a free parameter). The choice of the time constant for smoothing $\hat{\mathcal{L}}^{\sigma}$ is outlined in the following section.

2.5 Training Procedure for SGD

Training GMMs by SGD is performed by maximizing the smoothed max-component log-likelihood $\hat{\mathcal{L}}^{\sigma}$ from Eq. 10. At the same time, we enforce the constraints on the component weights and covariances as described in Sect. 2.1 and transition from $\hat{\mathcal{L}}^{\sigma}$ into $\hat{\mathcal{L}}$ by annealing (see Sect. 2.4). SGD requires a learning rate ϵ to be set, which in turn determines the parameter α (see Sect. 2.4) as $\alpha = \epsilon$ since stationarity detection should operate on a timescale similar to that of SGD. The diagonal matrices $\boldsymbol{D}_k$ are initialized to $D_{\max} I$ and are clipped after each iteration, so that diagonal entries remain in the range $[0, D_{\max}^2]$. This is necessary to avoid excessive growth of precisions for data entries with vanishing variance, e.g., pixels that are always black. Weights are uniformly initialized to $\pi^i = \frac{1}{K}$, centroids in the range $[-\mu^i, +\mu^i]$ (see algorithm 1 for a summary). Please note that our SGD approach requires no centroid initialization by k-means, as it is recommended when training GMMs with (s)EM. We discuss and summarize good practices for choosing hyper-parameters in Sect. 5.

Algorithm 1: Steps of SGD-GMM training.

Data: initializer values: μ^i, K, $\epsilon_0/\epsilon_\infty$, σ_0/σ_∞, δ and data $\boldsymbol{X}$
Result: trained GMM model

$\boldsymbol{\mu} \leftarrow \mathcal{U}(-\mu^i, +\mu^i)$, $\pi \leftarrow 1/K$, $\boldsymbol{P} \leftarrow I D_{\max}$, $\sigma \leftarrow \sigma_0$, $\epsilon \leftarrow \epsilon_0$
forall the $t < T$ **do** // training loop
  $\boldsymbol{g}(t) \leftarrow$ create_annealing_mask(σ,t) // see Sect. 2.4
  $\boldsymbol{\mu}(t) \leftarrow \epsilon \frac{\partial \hat{\mathcal{L}}^{\sigma}}{\partial \boldsymbol{\mu}} + \boldsymbol{\mu}(t\text{-}1)$, // SGD updates
  $\boldsymbol{P}(t) \leftarrow \epsilon \frac{\partial \hat{\mathcal{L}}^{\sigma}}{\partial \boldsymbol{P}} + \boldsymbol{P}(t\text{-}1)$,
  $\pi(t) \leftarrow \epsilon \frac{\partial \hat{\mathcal{L}}^{\sigma}}{\partial \pi} + \pi(t\text{-}1)$
  $\boldsymbol{P}(t) \leftarrow$ precisions_clipping($\boldsymbol{P}$, $D_{\max}$) //see Sect. 2.5
  $\pi(t) \leftarrow$ normalization($\pi(t)$) //see Eq. 7
  $\ell(t) \leftarrow (1-\alpha)\ell(t-1) + \alpha \hat{\mathcal{L}}^{\sigma}(\boldsymbol{x}(t))$ // sliding likelihood
  if *annealing update iteration* **then** // see Sect. 2.4
    if $\Delta < \delta$ **then** // Δ see Eq. 11
      $\sigma(t) \leftarrow 0.9\sigma(t-1), \epsilon(t) \leftarrow 0.9\epsilon(t-1)$

2.6 Proof that Annealing is Convergent

We assume that, for a fixed value of σ, SGD optimization has reached a stationary point where the derivative w.r.t. all GMM parameters is 0 on average. In this situation, we claim that decreasing σ will always increase the loss except when $\sigma \to 0$. If true, this would show that σ defines an effective upper bound for the loss. For this to be consistent, we have to show that the loss gradient w.r.t. σ vanishes as $\sigma \to 0$: as the annealed loss approaches the original one, decreases of σ have less and less effects.

Proposition The gradient $\frac{\partial \hat{\mathcal{L}}^{\sigma}}{\partial \sigma}$ is strictly positive for $\sigma > 0$

Proof For each sample, the 2D profile of $\log(\pi_k \mathcal{N}_k) \equiv f_k$ is assumed to be centered on the BMU k^* and depending on the distance from it as a function of $||k - k^*||$. We thus have $f_k = f_k(r)$ with $r \equiv ||k - k^*||$. Passing to the continuous domain, the indices in the Gaussian "smoothing filter" g_{k^*k} become continuous variables, and we have $g_{k^*k} \rightarrow g(||k - k^*||, \sigma) \equiv g(r, \sigma)$. Similarly, $f_k(r) \rightarrow f(r)$. Using 2D polar coordinates, the smoothed max-component likelihood $\hat{\mathcal{L}}^\sigma$ becomes a polar integral around the position of the BMU: $\hat{\mathcal{L}}^\sigma \sim \int_{\mathbb{R}^2} g(r, \sigma) f(r) dr d\phi$. It is trivial to show that for the special case of a constant log-probability profile, i.e., $f(r) = L$, $\mathcal{L}^\sigma$, does not depend on σ because Gaussians are normalized, and that the derivative w.r.t. σ vanishes:

$$\begin{aligned} \frac{d\hat{\mathcal{L}}^\sigma}{d\sigma} &\sim \int_0^\infty dr \Big(\frac{r^2}{\sigma^2} - 1\Big) \exp\Big(-\frac{r^2}{2\sigma^2}\Big) L \\ &= L \int_0^\sigma dr \Big(\frac{r^2}{\sigma^2} - 1\Big) \exp\Big(-\frac{r^2}{2\sigma^2}\Big) - L \int_\sigma^\infty \Big(\frac{r^2}{\sigma^2} - 1\Big) \exp\Big(-\frac{r^2}{2\sigma^2}\Big) \\ &\equiv L\mathcal{N} - L\mathcal{P} \end{aligned} \tag{12}$$

where we have split the integral into parts where the derivative w.r.t. σ is negative ($\mathcal{N}$) and positive ($\mathcal{P}$). We know that $\mathcal{N} = \mathcal{P}$ since the derivative must be zero for a constant function $f(r) = L$ due to the fact that Gaussians are normalized to the same value regardless of σ.

For a function $f(r)$ that satisfies $f(r) > L \forall r \in [0, \sigma[$ and $f(r) < L \forall r \in]\sigma, \infty[$, the inner and outer parts of the integral behave as follows:

$$\begin{aligned} \tilde{\mathcal{N}} &= \int_0^\sigma g(r) \Big(\frac{r^2}{\sigma^2} - 1\Big) f(r) < \int_0^\sigma g(r) \Big(\frac{r^2}{\sigma^2} - 1\Big) L = L\mathcal{N} \\ \tilde{\mathcal{P}} &= \int_\sigma^\infty g(r) \Big(\frac{r^2}{\sigma^2} - 1\Big) f(r) < \int_\sigma^\infty g(r) \Big(\frac{r^2}{\sigma^2} - 1\Big) L = L\mathcal{P} \end{aligned} \tag{13}$$

since $f(r)$ is minorized/majorized by L by assumption, and the contributions in both integrals have the same sign for the whole domain of integration. Thus, it is shown that

$$\frac{d\hat{\mathcal{L}}^\sigma}{d\sigma} = \tilde{\mathcal{N}} - \tilde{\mathcal{P}} < L\mathcal{N} - L\mathcal{P} = 0 \tag{14}$$

for $\sigma > 0$ and, furthermore, that this derivative is zero for $\sigma = 0$ because $\hat{\mathcal{L}}^\sigma$ no longer depends on σ for this case.

Taking everything into consideration, in a situation where the log-likelihood $\hat{\mathcal{L}}^\sigma$ has reached a stationary point for a given value of σ, we have shown that:

- the value of $\hat{\mathcal{L}}^\sigma$ depends on σ,
- without changing the log-probabilities, we can increase $\hat{\mathcal{L}}^\sigma$ by reducing σ, assuming that the log-probabilities are mildly decreasing around the BMU,
- increasing $\hat{\mathcal{L}}^\sigma$ works as long as $\sigma > 0$. At $\sigma = 0$ the derivative becomes 0.

Thus, σ indeed defines an upper bound to $\hat{\mathcal{L}}^\sigma$ which can be increased by decreasing σ. The assumption of log-probabilities that decrease, on average, around the BMU is reasonable, since such a profile maximizes $\hat{\mathcal{L}}^\sigma$. All functions $f(r)$ that, e.g., decrease monotonically around the BMU, fulfill this criterion, whereas the form of the decrease is irrelevant.

2.7 Training Procedure for sEM

We use sEM proposed by [2] as a reference point to which we compare our SGD approach. We choose the step size of the form $\rho_t = \rho_0 (t + 1)^{-0.5+\alpha}$, with $\alpha \in [0, 0.5]$, $\rho_0 < 1$ and

enforce $\rho(t) \geq \rho_\infty$. Values for these parameters are determined via a grid search in the ranges $\rho_0 \in \{0.01, 0.05, 0.1\}$, $\alpha \in \{0.01, 0.25, 0.5\}$ and $\rho_\infty \in \{0.01, 0.001, 0.0001\}$. Each sEM iteration uses a batch size B. Initial accumulation of sufficient statics is conducted for 10% of an epoch. Parameter initialization and clipping of precisions is performed just as for SGD, see Sect. 2.5.

2.8 Comparing SGD and sEM

Since sEM optimizes the log-likelihood $\mathcal{L}$, whereas SGD optimizes the annealed approximation $\hat{\mathcal{L}}^\sigma$, the comparison of these measures should be considered carefully. We claim that the comparison is fair assuming that (i) SGD annealing has converged and (ii) GMM responsibilities are sharply peaked so that a single component has a responsibility of ≈ 1. It follows from (i) that $\hat{\mathcal{L}}^\sigma \approx \hat{\mathcal{L}}$ and (ii) implies that $\hat{\mathcal{L}} \approx \mathcal{L}$. Condition (ii) is usually satisfied to high precision especially for high-dimensional inputs: if it is not, the comparison is biased in favor of sEM, since $\mathcal{L} > \hat{\mathcal{L}}$ by definition.

3 Experiments

Unless stated otherwise, the experiments in this section will be conducted with the following parameter values for sEM and SGD (where applicable): mini-batch size $B = 1$, $K = 8 \times 8$, $\mu^i = 0.1$, $\sigma_0 = 2$, $\sigma_\infty = 0.01$, $\epsilon = 0.001$, $D_{\max} = 20$. Each experiment is repeated 10 times with identical parameters but different random seeds for parameter initialization. See Sect. 5 for a justification of these choices. Due to high input dimensionalities, all precision matrices are assumed to be diagonal. The training/test data comes from the datasets shown below (see Sect. 3.1).

3.1 Datasets

We use a variety of different image-based datasets, as well as a non-image dataset for evaluation purposes. All datasets are normalized to the [0, 1] range.

MNIST [17] contains gray-scale images, which depict handwritten digits from 0 to 9 in a resolution of 28×28 pixels—the common benchmark for computer vision systems and classification problems.

SVHN [31] contains color images of house numbers (0–9, resolution 32×32).

FashionMNIST [32] contains grayscale images of 10 clothing categories and is considered as a more challenging classification task compared to MNIST.

Fruits 360 [19] consists of color pictures ($100 \times 100 \times 3$ pixels) showing different types of fruits. The ten best-represented classes are selected.

Devanagari [1] includes grayscale images of handwritten Devanagari letters with a resolution of 32×32 pixels—the first 10 classes are selected.

NotMNIST [33] is a grayscale image dataset (resolution 28×28 pixels) of letters from A to J extracted from different publicly available fonts.

ISOLET [5] is a non-image dataset containing 7 797 samples of spoken letters recorded from 150 subjects. Each sample is encoded and is represented by 617 float values.

3.2 Robustness of SGD to Initial Conditions

Here, we train GMMs for three epochs on classes 1 to 9 for each dataset. We use different random and non-random initializations of the centroids and compare the final log-likelihood values. Random centroid initializations are parameterized by $\mu^i \in \{0.1, 0.3, 0.5\}$, whereas non-random initializations are defined by centroids from a previous training run on class 0 (one epoch). The latter is done to have a non-random centroid initialization that is as dissimilar as possible from the training data. The initialization of the precisions cannot be varied, because empirical data shows that training converges to undesirable solutions if the precisions are not initialized to large values. While this will have to be investigated further, we nevertheless observe convergence to near-identical levels, regardless of centroid initialization, for all datasets (see Table 1 for more details).

3.3 Added Value of Annealing

To demonstrate the beneficial effects of annealing, we perform experiments on all datasets with annealing turned off. This is achieved by setting $\sigma_0 = \sigma_\infty$. This invariably produces sparse-component solutions with strongly inferior log-likelihoods after training, please refer to Table 1.

3.4 Clustering Performance Evaluation

To compare the clustering performance of sEM and GMM the Davies-Bouldin score [6] and the Dunn index [9] are determined. We evaluate the grid-search results to find the best parameter setup for each metric for comparison. Only sEM is initialized by k-means to show that our approach does not depend on parameter initialization. Table 2 indicates that SGD can egalize sEM performance.

3.5 Streaming Experiments with Constant Statistics

We train GMMs for three epochs (enough for convergence in all cases) using SGD and sEM on all datasets as described in Sects. 2.5 and 2.7. The resulting centroids of our SGD-based approach are shown in Fig. 3, whereas the final loss values for SGD and sEM are compared in Table 3. The centroids for both approaches are visually similar, except for the topological organization due to annealing for SGD, and the fact that in most sEM experiments, some components do not converge, while the others do. Table 3 indicates that SGD achieves performances superior to sEM in the majority of cases, in particular for the highest-dimensional datasets (SVHN: 3072 and Fruits 360: 30,000 dimensions).

3.6 Visualization of High-Dimensional sEM Outcomes

Figure 4 was obtained after training GMMs by sEM on both the Fruits 360 and the SVHN dataset. It should be compared to Fig. 3, where an identical procedure was used to visualize centroids of SGD-trained GMMs. It is notable that the effect of unconverged components does not occur at all for our SGD approach, which is due to the annealing mechanism that "drags" unconverged components along.

Table 1 Effect of different random and non-random centroid initializations on SGD training

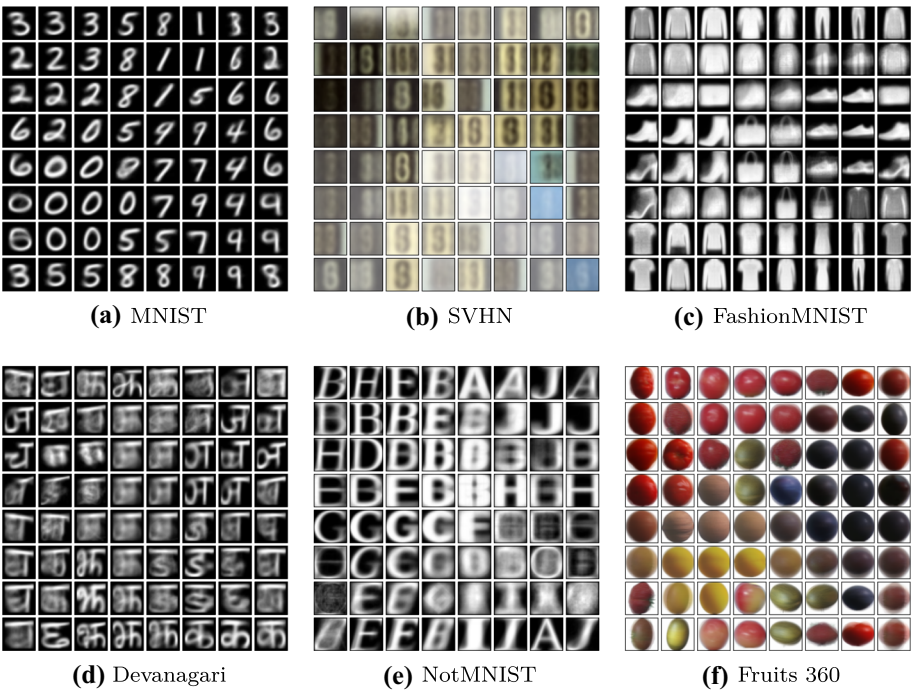
Initialization	Random						Non-random		No annealing
	$\mu^i = 0.1$		$\mu^i = 0.3$		$\mu^i = 0.5$		Init class 0		$\mu^i = 0.1$
Dataset	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean
MNIST	205.47	1.08	205.46	0.77	205.68	0.78	205.37	0.68	124.1
FashionMNIST	231.22	1.53	231.58	2.84	231.00	1.11	229.59	0.59	183.0
NotMNIST	−48.41	1.77	−48.59	1.56	−48.32	1.13	−49.37	2.32	− 203.8
Devanagari	−15.95	1.59	−15.76	1.34	−17.01	1.11	−22.07	4.59	− 263.4
Fruits 360	12,095.80	98.02	12,000.70	127.00	12,036.25	122.06	10,912.79	1727.61	331.2
SVHN	1328.06	0.94	1327.99	1.59	1328.40	1.17	1327.80	0.94	863.2
ISOLET	354.34	0.04	354.36	0.04	354.36	0.04	354.20	0.05	201.5

Given are the means and standard deviations of final log-likelihoods (10 repetitions per experiment). To show the added value of annealing, the right-most column indicates the final log-likelihoods when annealing is turned off. This value should be compared to the leftmost entry in each row, where annealing is turned on. Standard deviations in this case are very small, so they are omitted

Table 2 Clustering performance comparison of SGD and sEM training using Davies–Bouldin score (less is better) and Dunn index (more is better)

Metric algo.	Davies–Bouldin score				Dunn index			
	SGD		sEM		SGD		sEM	
Dataset	Mean	Std	Mean	Std	Mean	Std	Mean	Std
MNIST	2.50	0.04	**2.47**	0.04	0.18	0.02	0.16	0.02
FashionMNIST	**2.06**	0.05	2.20	0.04	0.20	0.03	0.19	0.02
NotMNIST	2.30	0.03	**2.12**	0.03	0.15	0.03	0.14	0.04
Devanagari	**2.60**	0.04	2.64	0.02	**0.33**	0.01	0.27	0.04
SVHN	**2.34**	0.04	2.41	0.03	0.15	0.02	0.15	0.02

Each time, mean metric value (of 10 experiment repetitions) at the end of training, and their standard deviations are presented. Results are in bold face whenever they surpass all other results by more than half a standard deviation



(a) MNIST **(b)** SVHN **(c)** FashionMNIST

(d) Devanagari **(e)** NotMNIST **(f)** Fruits 360

Fig. 3 Exemplary results for centroids learned by SGD for different datasets

4 Assumptions Made by EM and SGD

The EM algorithm assumes that the observed data samples $\{\boldsymbol{x}_n\}$ depend on unobserved latent variables $\boldsymbol{z}_n$ in a non-trivial manner, see Sect. 2. The derivation of the EM algorithm starts out with the total incomplete-data log-likelihood

Table 3 Comparison of SGD and sEM training on all datasets in a streaming-data scenario

Algo.	SGD			sEM	
Dataset	$\varnothing$ max p_{k*}	Mean	Std	Mean	Std
MNIST	0.992674	216.6	0.31	216.8	1.38
FashionMNIST	0.997609	**234.5**	2.28	222.9	6.03
NotMNIST	0.998713	**−34.7**	1.16	−40.0	8.90
Devanagari	0.999253	**−14.6**	1.09	−13.4	6.16
Fruits 360	0.999746	**11,754.3**	75.63	5483.0	1201.60
SVHN	0.998148	**1329.8**	0.80	1176.0	16.91
ISOLET	0.994069	354.2	0.01	354.5	0.37

Shown are log-likelihoods at the end of training, averaged over 10 repetitions, along with their standard deviations. Results are in bold face whenever they are higher by more than half a standard deviation. Additionally, the averaged maximum responsibilities (p_{k*}) for test data are given, as a justification of the max-component approximation

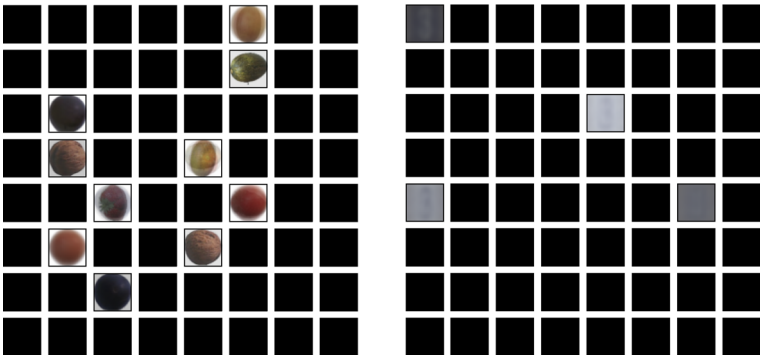


Fig. 4 Visualization of centroids after training runs (3 epochs) on high-dimensional datasets for sEM: Fruits 360 (left, 30,000 dimensions) and SVHN (right, 3000 dimensions). Component entries are displayed "as is", meaning that low brightness means low RGB values. Many GMM components remain unconverged, which is analogous to a sparse-component solution (compare Fig. 1) and explains the low log-likelihood values for these high-dimensional datasets

$$\begin{aligned}\mathcal{L} &= \log p(X) = \log \prod_n p(\boldsymbol{x}_n) = \sum_n \log p(\boldsymbol{x}_n) \\ &= \sum_n \log \sum_k p(\boldsymbol{x}_n, z_n = k) \\ &= \sum_n \log \sum_k p(z_n = k) \frac{p(\boldsymbol{x}_n, z_n = k)}{p(z_n = k)}.\end{aligned} \tag{15}$$

Due to the assumption that $\mathcal{L}$ is obtained by marginalizing out the latent variables, an explicit dependency on z_n can be re-introduced. For the last expression, Jensen' inequality can be used to construct a lower bound:

$$\begin{aligned}\mathcal{L} &\sim \sum_n \log \sum_k p(z_n = k) \frac{p(\boldsymbol{x}_n, z_n = k)}{p(z_n = k)} \\ &\geq \sum_n \sum_k p(z_n = k) \log \frac{p(\boldsymbol{x}_n, z_n = k)}{p(z_n = k)}.\end{aligned} \tag{16}$$

Since the realizations of the latent variables are unknown, we can assume any form for their distribution. In particular, for the choice $p(z_n) \sim p(\boldsymbol{x}_n, z_n)$, the lower bound becomes tight. Simple algebra and the fact that the distribution $p(z_n)$ must be normalized gives us:

$$\begin{aligned} p(z_n = k) &= \frac{p(z_n = k, \boldsymbol{x}_n)}{p(\boldsymbol{x}_n)} \\ &= p(z_n = k|\boldsymbol{x}_n) \\ &= \frac{p(z_n = k, \boldsymbol{x}_n)}{\sum_l p(z_n = l, \boldsymbol{x}_n)} \\ &= \frac{\pi_k \mathcal{N}_k(\boldsymbol{x}_n)}{\sum_l \pi_l \mathcal{N}_l(\boldsymbol{x}_n)} \end{aligned} \tag{17}$$

where we have used Eq. 2 in the last step. $p(z_n = k|\boldsymbol{x}_n)$ is a quantity that can be computed from data with no reference to the latent variables. For GMM it is usually termed *responsibility* and we write it as $p(z_n = k|\boldsymbol{x}_n) \equiv \gamma_{nk}$.

However, the construction of a tight lower bound, which is actually different from $\mathcal{L}$, only works when $p(\boldsymbol{x}_n, z_n)$ depends non-trivially on the latent variable z_n. If this is not the case, we have $p(\boldsymbol{x}_n, z_n) = K^{-1} p(\boldsymbol{x}_n)$ and the derivation of Eq. 16 goes down very differently:

$$\begin{aligned} \mathcal{L} \sim \sum_n \log p(\boldsymbol{x}_n) &\geq \sum_n \sum_k p(z_n = k) \log \frac{p(\boldsymbol{x}_n, z_n = k)}{p(z_n = k)} \\ &= \sum_n \sum_k p(z_n = k) \log \frac{K^{-1} p(\boldsymbol{x}_n)}{p(z_n = k)} \\ &= \sum_n \log \left(K^{-1} p(\boldsymbol{x}_n) \right) - \sum_k p(z_n = k) \log p(z_n = k) \\ &\equiv \sum_n \Big(\log p(\boldsymbol{x}_n) - \big(\log K - \mathcal{H}[z_n] \big) \Big) \end{aligned} \tag{18}$$

where $\mathcal{H}$ represents the Shannon entropy of $p(\boldsymbol{z})$. The highest value this can have is $\log K$ for an uniform distribution of the z_n, finally leading to a lower bound for $\mathcal{L}$ of

$$\mathcal{L} \geq \sum_n \Big(\log p(\boldsymbol{x}_n) \Big) \tag{19}$$

which is trivial by Jensen's inequality, but not tight. In particular, no closed-form solutions to the associated extremal value problem can be computed.

This shows that optimizing GMM by EM assumes that each sample has been drawn from a single component in a set of K uni-modal Gaussian distributions. Which distribution is selected for sampling depends on a latent random variable. On the other hand, optimization by SGD uses the incomplete-data log-likelihood $\mathcal{L}$ as basis for optimization, without assuming the existence of hidden variables at all. This may be advantageous for problems where the assumption of Gaussianity is badly violated, although empirical studies indicate that optimization by EM works very well in a very wide range of scenarios.

5 Discussion and Conclusion

The Relevance of this Article is outlined by the fact that training GMMs by SGD was recently investigated in the community by [13,14]. We go beyond, since our approach does not rely

on off-line data-driven model initialization, and works for high-dimensional streaming data. The presented SGD scheme is simple and very robust to initial conditions due to the proposed annealing procedure, see Sects. 3.2 and 3.3. In addition, our SGD approach compares favorably to the reference model for online EM [2] in terms of achieved log-likelihoods, which was verified on multiple real-world datasets. Superior SGD performance is observed for the high-dimensional datasets.

The Analysis of the Results suggests that SGD performs better than sEM on average, see Sect. 3.5, although the differences are modest. It is neither expected, nor is it the goal to show here, that sEM is outperformed by SGD. Instead, we aim at achieving a similar performance. However, if sEM is used without an initialization, e.g., k-means, components may not converge (see Fig. 4) for high-dimensional data like Fruits 360 and SVHN datasets. In such cases, SGD does outperform sEM. Another important advantage of SGD over sEM is the fact that the only parameter that needs to be tuned is the learning rate ϵ. In contrast to that, sEM has a complex and non-intuitive dependency on ρ_0, ρ_∞ and α_0.

Small Batch Sizes and Streaming Data are possible with the SGD-based approach. Throughout the experiments, we used a batch size of 1, which allows streaming-data processing without the need to store any samples at all. Larger batch sizes are possible and strongly increase execution speed. In the conducted experiments, SGD (and sEM) usually converged within the first two epochs, which is a substantial advantage whenever huge sets of data have to be processed.

Low-Dimensional Data can be processed as well using our SGD-based approach, which makes numerical issues and undesirable local minima less relevant. Thus, we recommend optimizing the full incomplete-data log-likelihood in this case. Although there is no real need to use the max-component approximation and annealing for low-dimensional data, it is nevertheless possible. To this effect, we performed experiments with synthetic data drawn from various 2D Gaussian mixture distributions, using the same parameters as in the experiments of Sect. 3. We always observed rapid convergence if the initialization range of the centroids, μ^i, fits the ranges of the individual data components. This ensures that initial centroids cover the data space sufficiently so that each cluster in the data has at least one centroid close to it.

No Assumptions About Data Generation are made by SGD in contrast to the EM and sEM algorithms. The latter guarantees that the loss will not decrease due to an M-step. This, however, assumes a non-trivial dependency of the data on an unobservable latent variable (shown in Sect. 4). In contrast, SGD makes no hard-to-verify assumptions, which is a rather philosophical point, but may be an advantage in certain situations where data are strongly non-Gaussian.

Numerical Stability is assured by our SGD training approach. It does not optimize the log-likelihood but its max-component approximation. This approximation contains no exponentials at all, and is well justified by the results of Table 3 which shows that component probabilities are strongly peaked. In fact, it is the gradient computations where numerical problems occurred, e.g., NaN values. The "logsumexp" trick mitigates the problem, but does not eliminate it (see Sect. 2.2). It cannot be used when gradients are computed automatically, which is what most machine learning frameworks do.

Hyper-Parameter Selection Guidelines are as follows: the learning rate ϵ must be set by cross-validation (a good value is 0.001). We empirically found that initializing precisions to the cut-off value $D_{\max}$ and an uniform initialization of the π_i are beneficial, and that centroids are best initialized to small random values. A value of $D_{\max} = 20$ always worked in our experiments. Generally, the cut-off must be much larger than the inverse of the data variance. In many cases, it should be possible to estimate this roughly, even in streaming settings,

especially when samples are normalized. For density estimation, choosing higher values for K leads to higher final log-likelihoods. For clustering, K should be selected using standard techniques for GMMs. The parameter δ controls loss stationarity detection for the annealing procedure and was shown to perform well for $\delta = 0.05$. Larger values will lead to a faster decrease of $\sigma(t)$, which may impair convergence. Smaller values are always admissible but lead to longer convergence times. The annealing time constant α should be set to the GMM learning rate ϵ or lower. Smaller values of α lead to longer convergence times since $\sigma(t)$ will be updated less often. The initial value σ_0 needs to be large in order to enforce convergence for all components. A typical value is $\sqrt{K}$. The lower bound on σ_∞ should be as small as possible in order to achieve high log-likelihoods (e.g., 0.01, see Sect. 2.6 for a proof).

6 Future Work

The presented work can be extended in several ways: First of all, annealing control could be simplified further by inferring good δ values from α. Likewise, *increases* of σ might be performed automatically when the loss rises sharply, indicating a task boundary. As we found that GMM convergence times grow linear with the number of components, we will investigate hierarchical GMM models that operate like a Convolutional Neural Network (CNN), in which individual GMM only see a local patch of the input and can therefore have low K.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflicts of interest The authors declare that they have no conflict of interest.

References

1. Acharya S, Pant AK, Gyawali PK (2016) Deep learning based large scale handwritten Devanagari character recognition. In: SKIMA 2015: 9th international conference on software, knowledge, information management and applications. https://doi.org/10.1109/SKIMA.2015.7400041
2. Cappé O, Moulines E (2009) On-line expectation-maximization algorithm for latent data models. J R Stat Soc Ser B Stat Methodol 71(3):593–613. https://doi.org/10.1111/j.1467-9868.2009.00698.x
3. Cederborg T, Li M, Baranes A, Oudeyer PY (2010) Incremental local online Gaussian mixture regression for imitation learning of multiple tasks. In: IEEE/RSJ 2010 international conference on intelligent robots and systems, IROS 2010: conference proceedings, pp 267–274. https://doi.org/10.1109/IROS.2010.5652040
4. Chen J, Zhu J, Teh YW, Zhang T (2018) Stochastic expectation maximization with variance reduction. In: Advances in neural information processing systems (NeurIPS), pp 7967–7977
5. Cole R, Fanty M (1990) Spoken letter recognition, pp 385–390. https://doi.org/10.3115/116580.116725
6. Davies DL, Bouldin DW (1979) A cluster separation measure. IEEE Trans Pattern Anal Mach Intell 2:224–227. https://doi.org/10.1109/TPAMI.1979.4766909

7. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Ser B (Methodol) 39(1):1–22. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x
8. Dognin PL, Goel V, Hershey JR, Olsen PA (2009) A fast, accurate approximation to log likelihood of Gaussian mixture models. Proc ICASSP IEEE Int Conf Acoust Speech Signal Process 3:3817–3820. https://doi.org/10.1109/ICASSP.2009.4960459
9. Dunn JC (1973) A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. J Cybern 3(3):32–57. https://doi.org/10.1080/01969727308546046
10. Engel PM, Heinen MR (2010) Incremental learning of multivariate Gaussian mixture models. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 6404, LNAI, pp 82–91. https://doi.org/10.1007/978-3-642-16138-4_9
11. Feldman D, Faulkner M, Krause A (2011) Scalable training of mixture models via coresets. In: 25th Annual conference on advances in neural information processing systems, NIPS 2011, pp 1–9
12. Ge R, Huang Q, Kakade SM (2015) Learning mixtures of gaussians in high dimensions. In: Proceedings of the annual ACM symposium on theory of computing, 14–17 June, pp 761–770. https://doi.org/10.1145/2746539.2746616
13. Hosseini R, Sra S (2015) Matrix manifold optimization for Gaussian mixtures. In: Advances in neural information processing systems, 2015 January, pp 910–918
14. Hosseini R, Sra S (2019) An alternative to em for gaussian mixture models: batch and stochastic riemannian optimization. In: Mathematical programming, pp 1–37
15. Kristan M, Skocaj D, Leonardis A (2008) Incremental learning with gaussian mixture models. In: Computer vision winter workshop, pp 25–32
16. Lange K (1995) A gradient algorithm locally equivalent to the EM algorithm. J R Stat Soc Ser B (Methodol) 57(2):425–437. https://doi.org/10.1111/j.2517-6161.1995.tb02037.x
17. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2323. https://doi.org/10.1109/5.726791
18. Melnykov V, Maitra R (2010) Finite mixture models and model-based clustering. Stat Surv 4(October):80–116. https://doi.org/10.1214/09-SS053
19. Mureşan H, Oltean M (2018) Fruit recognition from images using deep learning. Acta Universitatis Sapientiae Informatica 10(1):26–42. https://doi.org/10.2478/ausi-2018-0002
20. Newton J, Titterington DM, Smith AFM, Makov UE (1986) Statistical analysis of finite mixture distributions. Biometrics 42(3):679. https://doi.org/10.2307/2531224
21. Nielsen F, Sun K (2016) Guaranteed bounds on information-theoretic measures of univariate mixtures using piecewise log-sum-exp inequalities. Entropy. https://doi.org/10.3390/e18120442
22. Ormoneit D, Tresp V (1998) Averaging, maximum penalized likelihood and Bayesian estimation for improving Gaussian mixture probability density estimates. IEEE Trans Neural Netw 9(4):639–650. https://doi.org/10.1109/72.701177
23. Pinheiro JC, Bates DM (1995) Approximations to the log-likelihood function in the nonlinear mixed-effects model. J Comput Graph Stat 4(1):12–35. https://doi.org/10.1080/10618600.1995.10474663
24. Pinto RC, Engel PM (2015) A fast incremental Gaussian mixture model. PLoS ONE. https://doi.org/10.1371/journal.pone.0139931
25. Richardson E, Weiss Y (2018) On GANs and GMMs.In: Advances in neural information processing systems (NeurIPS), pp 5847–5858
26. Song M, Wang H (2005) Highly efficient incremental estimation of Gaussian mixture models for online data stream clustering. Intell Comput Theory Appl III 5803:174. https://doi.org/10.1117/12.601724
27. Van Den Oord A, Schrauwen B (2014) Factoring variations in natural images with deep gaussian mixture models. In: Neural information processing systems
28. Verbeek JJ, Vlassis N, Kröse BJ (2005) Self-organizing mixture models. In: Neurocomputing (SPEC. ISS.), vol 63, pp 99–123. https://doi.org/10.1016/j.neucom.2004.04.008
29. Vijayakumar S, DSouza A, Schaal S (2005) Incremental online learning in high dimensions. Neural Comput 17(12):2602–2634. https://doi.org/10.1162/089976605774320557
30. Vlassis N, Likas A (2002) A greedy EM algorithm for Gaussian mixture learning. Neural Process Lett 15(1):77–87. https://doi.org/10.1023/A:1013844811137
31. Wang T, Wu DJ, Coates A, Ng AY (2012) End-to-end text recognition with convolutional neural networks. In: Proceedings of the 21st international conference on pattern recognition (ICPR2012), IEEE, pp 3304–3308
32. Xiao H, Rasul K, Vollgraf R (2017) Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, pp 1–6
33. Yaroslav B (2011) Machine learning, etc notMNIST dataset. http://yaroslavvb.blogspot.com.br/2011/09/notmnist-dataset.html