Pattern Recognition Letters
journal homepage: www.elsevier.com

Estimation of Gaussian Mixture Models via Tensor Moments with Application to Online Learning

Donya Rahmani[a,**], Mahesan Niranjan[b], Damien Fay[c], Akiko Takeda[d], Jacek Brodzki[a]

[a]School of Mathematics, University of Southampton, Highfield, Southampton, UK
[b]School of Electronics and Computer Science, University of Southampton, Southampton, UK
[c]Department of Analytics, Logicblox/Infor, Atlanta, GA
[d]Department of Creative Informatics, The University of Tokyo, Tokyo, Japan

ABSTRACT

In this paper, we present an alternating gradient descent algorithm for estimating parameters of a spherical Gaussian mixture model by the method of moments (AGD-MoM). We formulate the problem as a constrained optimisation problem which simultaneously matches the third order moments from the data, represented as a tensor, and the second order moment, which is the empirical covariance matrix. We derive the necessary gradients (and second derivatives), and use them to implement alternating gradient search to estimate the parameters of the model. We show that the proposed method is applicable in both a batch as well as in a streaming (online) setting. Using synthetic and benchmark datasets, we demonstrate empirically that the proposed algorithm outperforms the more classical algorithms like Expectation Maximisation and variational Bayes.

1. Introduction

The Gaussian mixture model (GMM) is a widely used and extremely practical probabilistic modelling tool used in statistics and machine learning. It is a powerful model for data clustering problems where the data is assumed to be generated from a mixture of several distinct underlying probability distributions. Central to a successful application of a GMM is the parameter estimation technique. This topic dates back more than a hundred years [1], and the most commonly used method is the Expectation-Maximisation (EM) algorithm which is based on maximum likelihood estimation. However, it has long been known that the EM algorithm may be slow to converge (as first noted in [2] and expanded in [3, 4]). Generally speaking, the EM algorithm works best when the component densities are well separated, but convergence may be exorbitantly slow and result in estimates with high variance when they are poorly separated [3]. In addition, the computational complexity for the EM algorithm depends significantly on the number of samples. When clustering N samples of D-dimensional data into K clusters this complexity grows as $O(DN + KN^2)$, [5]. As an alternative, in this paper, we investigate a variant of the Method of Moments (MoM) based on Alternating Gradient Descent (AGD). The earliest known application of the MoM was provided by Chebyshev (1887) in his study of the central limit theorem in statistics. This was then followed by Karl Pearson in his classic work on MoM in 1894, [6]. Pearson first used the MoM to estimate the five parameters of a two-component univariate Gaussian mixture. After Pearson, the MoM became one of the most popular ways of estimating the parameters of a finite mixture distribution (see [7, 1, 8]). One reason is that MoM estimators impose fewer restrictions on the model compared to MLE.

The basic idea behind a MoM estimator is that, given a sufficient number of moments, one may match the theoretical and sample moments to estimate the parameters. More specifically, the moment based approach solves systems of multivariate polynomial equations for computing all solutions of a given model without any prerequisite assumption. However, this can be computationally challenging. One approach proposed in [7] suggests that to speed up the computation one can transform the moment equations into a set of related linear equations (in essence marginal distributions, which can be solved efficiently), and one non-linear equation (which is cubic and can also be solved efficiently). In addition, a successful implementation of the MoM should yield estimators that are statistically effi-

**Corresponding author: Tel.: +0-44-2380525156;
e-mail: d.rahmani@soton.ac.uk (Donya Rahmani)

cient in the sense that their expectations maximise the likelihood function, [7]. Thus, the challenge in using the MoM approach is to preserve high statistical efficiency while reducing the computational burden.

In recent years, tensor based methods have received much attention together with the development of computationally efficient algorithms (see [9] for an excellent overview). These methods can be directly used for computing MoM estimators. For example, $n-$order moments can be regarded as an $n-$mode tensor which can then be simplified using one of several tensor decomposition methods. This is the core of the MoM approach used by [8] and also by Anadkumar et al. in [10]. They propose a moment matching estimator for mixtures of spherical Gaussians using a simple spectral decomposition of lower order moments obtained from the data. Their method uses the eigenvalue decomposition of symmetric matrices (see [8]) to decrease the computational complexity of the MoM. Note that the eigenvalues are found via the power iteration method and deflation method which may introduce errors in the parameter estimates due to approximation inherent in deflation methods.

In this paper, we propose an alternative approach to this problem. In particular, we reformulate the moment matching problem as a constrained optimisation problem which can be solved via Alternating Gradient Descent (AGD-MoM). This allows us to obtain the required estimators by a direct computation rather than through an iterative procedure. As will be seen, this leads to estimates with increased accuracy over the competing techniques. Furthermore, we demonstrate that in the case of online streaming data the algorithm tracks the evolution of the cluster centres with greater accuracy and less computational expense. In addition, as our method requires storage of the moments of the streaming data, the memory requirements are efficient and independent of the number of samples.

The remainder of the paper is organised as follows. In Section 2 we briefly review the moment-based estimation problem. Section 3 presents our proposed algorithm. Section 4 demonstrates how AGD-MoM can be adapted in an online setting. Section 5 compares AGD-MoM with several competing methods using both synthetic and real data. Finally, Section 6 draws conclusions and suggests avenues for future work.[1]

2. Moment-based estimation

A spherical GMM assumes that the D dimensional data observations are sampled from K clusters. The probability of sampling cluster $k \in \{1, \dots, K\}$ is given by a mixing weight vector, $\underline{\mathbf{w}} = (w_1, \dots, w_K)$, $w_k \in [0, 1]$, and $\sum_k w_k = 1$. Each cluster has a centre denoted by $\underline{\boldsymbol{\mu}}_k \in \mathbb{R}^D$ with an associated isometric covariance matrix $\sigma_k^2 I$. Denote a sample from this data generating process as $\underline{\mathbf{x}} = \underline{\boldsymbol{\mu}}_h + \underline{\mathbf{z}}$ where $\underline{\mathbf{x}} \in \mathbb{R}^D$, $\underline{\mathbf{z}}$ is assumed to be a random vector whose conditional distribution given $h = k$ is $N(0, \sigma_h^2 I)$, and $P(h = k) = w_k$ for $k \in [K]$. The goal of GMM estimation is to optimally recover the parameters $\{\underline{\boldsymbol{\mu}}_{1:K}, \sigma_{1:K}^2, w_{1:K}\}$ given a set of observations $\underline{\mathbf{x}}$.

[1] An implementation of our algorithm in tensorflow may be found here: https://github.com/drahmani/AGD-MoM.

With respect to GMM, the MoM solves multivariate polynomial equations, pairs the sample moments with their corresponding theoretical moments, to find the parameters of the model. In this paper, we examine a computationally efficient optimisation procedure to recover the parameters of GMM using the MoM. Our method assumes the non-degeneracy condition [8] is met. We recall that the non-degeneracy condition states that the component mean vectors, $\underline{\boldsymbol{\mu}}_1, \underline{\boldsymbol{\mu}}_2, \dots, \underline{\boldsymbol{\mu}}_K$, are linearly independent, and the mixing components $w_1, w_2, \dots, w_K > 0$, are strictly positive and sum to one. Note also that the non-degeneracy condition places a weaker restriction on the data generating process than the *spreading condition* from standard (e.g., EM) density estimation approaches (see [11, 12] for specifics). Given the non-degeneracy condition, the sample moments can be expressed as a sum of rank one tensors and therefore the relationship between the GMM parameters and their corresponding moments can be expressed by the following theorem.

Theorem 1. *(Hsu and Kakade, 2012) Assume $D \geq K$. The average variance $\bar{\sigma}^2 = \sum_{k=1}^K w_k \sigma_k^2$ is the smallest eigenvalue of the covariance matrix $C_{\underline{\mathbf{x}}} = E[\underline{\mathbf{x}} \otimes \underline{\mathbf{x}}] - E[\underline{\mathbf{x}}] \otimes E[\underline{\mathbf{x}}]$. Let, $\underline{\boldsymbol{\nu}}$ be any unit norm eigenvector corresponding to the eigenvalue $\bar{\sigma}^2$. Furthermore, if*

$$\underline{\boldsymbol{\mu}} = E[\underline{\mathbf{x}}(\underline{\boldsymbol{\nu}}^T(\underline{\mathbf{x}} - E[\underline{\mathbf{x}}]))^2],$$
$$M = E[\underline{\mathbf{x}} \otimes \underline{\mathbf{x}}] - \bar{\sigma}^2 I,$$
$$\mathcal{T} = E[\underline{\mathbf{x}} \otimes \underline{\mathbf{x}} \otimes \underline{\mathbf{x}}] - (\sum_{i=1}^D \underline{\boldsymbol{\mu}} \otimes e_i \otimes e_i + e_i \otimes \underline{\boldsymbol{\mu}} \otimes e_i + e_i \otimes e_i \otimes \underline{\boldsymbol{\mu}}),$$

where $\underline{\boldsymbol{\mu}} \in \mathbb{R}^D$, $M \in \mathbb{R}^{D \times D}$ and $\mathcal{T} \in \mathbb{R}^{D \times D \times D}$, then

$$M = \sum_{k=1}^K w_k \underline{\boldsymbol{\mu}}_k \otimes \underline{\boldsymbol{\mu}}_k, \qquad \mathcal{T} = \sum_{k=1}^K w_k \underline{\boldsymbol{\mu}}_k \otimes \underline{\boldsymbol{\mu}}_k \otimes \underline{\boldsymbol{\mu}}_k.$$

The empirical moments converge to the exact moments at a rate of $O(N^{\frac{-1}{2}})$ [8]. As the lower order moments, M and $\mathcal{T}$, are orthogonally decomposable tensors consequently the moment matching problem can be cast as eigenvalue decomposition of symmetric matrices. Such orthogonal decomposition problems can be efficiently solved by iterative approaches like the power iteration method, fixed-point iteration and gradient descent, see [13, 10, 8]. An orthogonal decomposition of a symmetric tensor $\mathcal{T} \in \mathbb{R}^{D \times D \times D}$ is a collection of orthonormal (unit) vectors $\underline{\mathbf{v}}_1, \underline{\mathbf{v}}_2, \dots, \underline{\mathbf{v}}_K$ together with corresponding positive scalars $\lambda_k > 0$ such that $\mathcal{T} = \sum_{k=1}^K \lambda_k \underline{\mathbf{v}}_k^{\otimes 3}$. Anadkumar et al. [10] use the tensor power method of Lathauwer et. al. proposed in [13] to obtain robust estimates of eigenvector/eigenvalue pairs, $\{(\underline{\mathbf{v}}_k, \lambda_k)\}$, for orthogonal tensor decomposition. Note that the orthogonality is required only for the whitened mean vectors (not original $\underline{\boldsymbol{\mu}}_i$'s), and this is guaranteed under the aforementioned non-degeneracy conditions. It is also worth pointing out that orthogonal decompositions do not necessarily exist for every symmetric tensor. A thorough review of these models goes beyond the scope of this paper.

We propose here an alternative approach to extract the orthogonal decomposition of M and $\mathcal{T}$ by reformulating the moment matching problem as a constrained optimisation problem. We provide a detailed description in the following section.

3. Learning mixture of spherical Gaussian via alternating gradient descent: AGD-MoM

In this section, we first reformulate the moment matching problem as a constrained optimisation problem and then present an AGD algorithm for GMM parameter estimation. We start by reformulating Theorem 1 in terms of minimising the objective function:

$$\min_{\substack{w_1,\dots,w_k\\ \underline{\mu}_1,\dots,\underline{\mu}_k}} \|\mathcal{T} - \sum_{k=1}^{K} w_k \underline{\mu}_k \otimes \underline{\mu}_k \otimes \underline{\mu}_k\|_F^2, \tag{1}$$

$$\text{subject to} \qquad M = \sum_{k=1}^{K} w_k \underline{\mu}_k \otimes \underline{\mu}_k,$$

where $\|.\|_F$ denotes the Frobenius tensor norm, which is the square root of the sum of the squares of all its elements. Our constrained optimisation problem involves minimising the discrepancy between the empirical tensor and exact tensor subject to a constraint. Examples of such problems for non-negative tensor factorisation can be found in [14, 15, 16].

Here, we require a more robust approach to find a set of efficient and optimal solutions to minimise the objective function (1). We use the *quadratic penalty method* [17] which is one way to solve this type of constrained optimisation problem. Note that equation (1) can be presented in a quadratic penalty format by adding the square of the violation of the equality constrained terms to the cost function (see [17]) as follows:

$$\min_{\substack{w_1,\dots,w_k\\ \underline{\mu}_1,\dots,\underline{\mu}_k}} F(w_1,\dots,w_k,\underline{\mu}_1,\dots,\underline{\mu}_k) = f + \lambda g, \tag{2}$$

where

$$f = \|\mathcal{T} - \sum_{k=1}^{K} w_k \underline{\mu}_k \otimes \underline{\mu}_k \otimes \underline{\mu}_k\|_F^2 \quad \text{and} \quad g = \|M - \sum_{k=1}^{K} w_k \underline{\mu}_k \otimes \underline{\mu}_k\|_F^2,$$

and λ is the penalty parameter which can be changed adaptively, according to the difficulty in minimising the equality constraint at each iteration. To examine the convergence properties of equation (2), we rely on Theorem 17.1 from [17]. According to this theorem, every limit point x^* of the sequence of $\{x_t\}$, where x_t is the exact global minimiser of $f(x_t) + \lambda_t \sum_i g_i^2(x_t)$, is a global solution of the problem as $\lambda_t \to \infty$.

As we now see, this unconstrained problem can be solved by an AGD based algorithm, alternating between updates of the two sets of parameters: mixing weight $\underline{\mathbf{w}}$ and component means $\underline{\mu}_1,\dots,\underline{\mu}_K$. Note that AGD is one such algorithms to solve unconstrained optimisation problems, [14, 18]. Using AGD to recover the model parameters, $\underline{\mu}_1,\dots,\underline{\mu}_K$ results in the following solutions:

Fix w_k and update $\underline{\mu}_k : \underline{\mu}_k^{t+1} = \underline{\mu}_k^t - \alpha_t\Big(\nabla f(\underline{\mu}_k^t|w_k^t) + \lambda_t \nabla g(\underline{\mu}_k^t|w_k^t)\Big)$,

Fix $\underline{\mu}_k$ and update $w_k : w_k^{t+1} = w_k^t - \beta_t\Big(\nabla f(w_k^t|\underline{\mu}_k^t) + \lambda_t \nabla g(w_k^t|\underline{\mu}_k^t)\Big)$.

Note that $\nabla f(\underline{\mu}_k^t|w_k^t)$, $\nabla g(\underline{\mu}_k^t|w_k^t)$, $\nabla f(w_k^t|\underline{\mu}_k^t)$ and $\nabla g(w_k^t|\underline{\mu}_k^t)$ are the necessary gradients of the cost function, $f + \lambda g$, with respect to the parameters $\underline{\mu}$ and $\underline{\mathbf{w}}$ and learning rates $\{\alpha_t, \beta_t\}$. The first step in finding these derivatives is to rewrite the error norm as a sum of quadratic terms:

$$f = \sum_{l=1}^{D}\sum_{j=1}^{D}\sum_{i=1}^{D}(t_{ijl} - \sum_{k=1}^{K} w_k \mu_{ki}\mu_{kj}\mu_{kl})^2, \tag{3}$$

$$g = \sum_{j=1}^{D}\sum_{i=1}^{D}(m_{ij} - \sum_{k=1}^{K} w_k \mu_{ki}\mu_{kj})^2. \tag{4}$$

We remark that $\mathcal{T}$ and M are symmetric, meaning the values of the elements[2], t_{ijl} and m_{ij}, are the same under any permutation of the indices, allowing efficient computation.[3] Next, differentiating equations (3) and (4) with respect to the (κ, d) component of $\underline{\mu}$ and the κ^{th} component of $\underline{\mathbf{w}}$, where $\kappa = 1,\dots,K; d = 1,\dots,D$, we have:[4]

$$\frac{\partial f}{\partial \mu_{\kappa d}} = 6w_\kappa \sum_{i=1}^{D} \mu_{\kappa i}^2\Big(\sum_{k=1}^{K} w_k \mu_{kd}\mu_{ki}^2 - t_{idi}\Big) + 12w_\kappa \sum_{\substack{i,j=1\\ j\neq i\neq d}}^{D} \mu_{\kappa i}\mu_{\kappa j}\Big(\sum_{k=1}^{K} w_k \mu_{kd}\mu_{ki}\mu_{kj} - t_{idj}\Big), \tag{5}$$

$$\frac{\partial g}{\partial \mu_{\kappa d}} = 4w_\kappa \sum_{i=1}^{D} \mu_{\kappa i}\Big(\sum_{k=1}^{K} w_k \mu_{kd}\mu_{ki} - m_{id}\Big), \tag{6}$$

$$\frac{\partial f}{\partial w_\kappa} = 2\sum_{i=1}^{D} \mu_{\kappa i}^3\Big(\sum_{k=1}^{K} w_k \mu_{ki}^3 - t_{iii}\Big) + 6\sum_{\substack{i,j=1\\ j\neq i}}^{D} \mu_{\kappa i}^2\mu_{\kappa j}\Big(\sum_{k=1}^{K} w_k \mu_{ki}^2\mu_{kj} - t_{iji}\Big), + 12\sum_{\substack{i,j,l=1\\ l\neq j\neq i}}^{D} \mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa l}\Big(\sum_{k=1}^{K}\sum_{\substack{i,j,l=1\\ i\neq j\neq l}}^{D} w_k \mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa l} - t_{idj}\Big), \tag{7}$$

$$\frac{\partial g}{\partial w_\kappa} = 2\sum_{i=1}^{D} \mu_{\kappa i}^2\Big(\sum_{k=1}^{K} w_k \mu_{ki}^2 - m_{ii}\Big) + 4\sum_{\substack{i,j=1\\ j>i}}^{D} \mu_{\kappa i}\mu_{\kappa j}\Big(\sum_{k=1}^{K} w_k \mu_{\kappa i}\mu_{\kappa j} - m_{ij}\Big). \tag{8}$$

Although the component-by-component differentiation involves cumbersome calculations, it is useful to note the correspondence between several terms in Equations (5) to (8) which need only be computed once. This can be more readily seen when expressed in vector form, following the notation of [19] which results in:

$$\frac{\partial f}{\partial \underline{\mu}_\kappa} = -6w_\kappa \mathcal{T}\underline{\mu}_\kappa^2 + 6w_\kappa \sum_{k=1}^{K} w_k\Big(\underline{\mu}_\kappa^T \underline{\mu}_k\Big)^2 \underline{\mu}_k, \tag{9}$$

$$\frac{\partial g}{\partial \underline{\mu}_\kappa} = 4w_\kappa \sum_{k=1}^{K} w_k\Big(\underline{\mu}_\kappa^T \underline{\mu}_k\Big)\underline{\mu}_k - 4w_\kappa M\underline{\mu}_\kappa, \tag{10}$$

$$\frac{\partial f}{\partial w_\kappa} = -2\mathcal{T}\underline{\mu}_\kappa^3 + 2\sum_{k=1}^{K} w_k\Big(\underline{\mu}_\kappa^T \underline{\mu}_k\Big)^3, \tag{11}$$

$$\frac{\partial g}{\partial w_\kappa} = -2M\underline{\mu}_\kappa^2 + 2\sum_{k=1}^{K} w_k\Big(\underline{\mu}_\kappa^T \underline{\mu}_k\Big)^2, \tag{12}$$

where $T\underline{\mu}_k^3 = \sum_{\mathbf{i}\in I} T_{\mathbf{i}}(\underline{\mu}_k)_{\mathbf{i}}^3$ for $\mathbf{i} \in \mathcal{I} = \{(i_1,i_2,i_3)|i_1,i_2,i_3 \in \{1,\dots,D\}\}$; and $i_1,i_2,i_3 \in \{1,2,3\}$. In fact, [19] defines $\mathcal{I}$ as an index representation which is a set of unique entries of the tensor (due to symmetry). For each $\mathbf{i} \in \mathcal{I}$, a corresponding monomial is defined as $\mathbf{c}$, which belongs to the following set:

[2] The i^{th} entry of a vector $\underline{\mu}_k$ is denoted by μ_{ki}, element (i, j) of a matrix M is denoted by m_{ij} , and element (i, j, l) of a third-order tensor $\mathcal{T}$ is denoted by t_{ijl}.

[3] For instance, in the tensor case $t_{ijl} = t_{ilj} = t_{jil} = t_{jli} = t_{lij} = t_{lji}$ for all $i, j, l = 1,\dots,D$, [9].

[4] Note that the differentials have been validated with the symbolic toolbox in *MATLAB*.

$$C = \Big\{(c_1, c_2, \ldots, c_D) | c_1, \ldots, c_D \in \{0, \ldots, 3\}\Big\},$$

where we assume that $c_1 + \ldots + c_D = 3$. It is shown in [19] that the multiplicity of the entry corresponding to a monomial representation $\mathbf{c} \in C$ is:

$$\sigma_{\mathbf{c}} = \binom{3}{c_1, c_2, \ldots, c_D} = \frac{3!}{c_1! c_2! \ldots, c_D!}.$$

Therefore, $\|\mathcal{T}\|_F^2 = \sum_{\mathbf{i} \in \mathcal{R}} t_{\mathbf{i}} = \sum_{\mathbf{i} \in I} \sigma_{\mathbf{i}} t_{\mathbf{i}} = \sum_{\mathbf{c} \in C} \sigma_{\mathbf{c}} t_{\mathbf{c}}$ and $(\underline{\mu}_k^3)_{\mathbf{i}} = \underline{\mu}_{i_1 k} \underline{\mu}_{i_2 k} \underline{\mu}_{i_3 k} = (\underline{\mu}_k^3)_{\mathbf{c}} = \underline{\mu}_{1k}^{c_1} \underline{\mu}_{2k}^{c_2} \ldots \underline{\mu}_{Dk}^{c_D}$. As can be seen, this representation gives a more compact formulation compared to component-wise differentiation. Similar expressions can also be derived for the second derivative of the cost function. In particular, second derivatives are useful in performing Newton-type updates with quadratic convergences. However, this involves inverting a matrix making each step computationally expensive. Though we have not used the second derivatives in our empirical work here, in the Supplementary Material we give the required expressions for completeness. Algorithm 1 presents the pseudo-code for AGD-MoM . We provide a complete implementation of our code in *MATLAB*, and a Tensorflow implementation which uses automatic gradient computation. Experiments confirmed that the differences between the two implementations is within negligible numerical limits.

Before turning to the application of AGD-MoM and presenting our results, we discuss some points that need attention when specifying an optimisation model. First, in terms of convergence, our algorithm will eventually converge to a local optimum of the objective function. As with any gradient descent approach, the speed of convergence will depend on the choice of the learning rate parameter. A learning rate which is too small leads to a slow convergence, whereas a large learning rate causes the loss function to oscillate around the minimum or even to diverge. Various techniques are known to speed up the convergence. These include the use of a Momentum term [20], Nesterov accelerated gradient [21], Adaptive moment estimation (Adam) [22], etc. In our implementation we found faster convergences using a Momentum term and adaptive learning rate.[5]

Second, the computational complexity of AGD-MoM is $O\Big(KDlog(1/\epsilon)\Big)$ for accuracy ϵ whereas the total running time of the power iteration method is $O\Big(K^{5+\delta}\Big(log(K) + loglog(1/\epsilon)\Big)\Big)$ for at most $K^{1+\delta}$ random starts to find each eigenvectors, $O\Big(log(K) + loglog(1/\epsilon)\Big)$ iteration per start and each iteration needs $O(K^3)$ operations. In terms of computation complexity it seems that the AGD is more efficient than the power iteration method. However, as will be seen in Section 5, the power iteration method converges faster than AGD.

[5]Momentum helps to accelerate GD in the relevant direction and dampens oscillations [20]. Basically a fraction γ of the update vector of the previous step is added to the current update vector. On the other hand, Adam adapts the learning rate for each parameter individually, performing larger updates for infrequent and smaller updates for frequent parameters [23]. Essentially, the learning rate α and β are tuned according to the scale of the smoothness of the gradient function. More details can be found in [22].

Algorithm 1 AGD-MoM for learning GMM parameters:

Input:
Third-order moment as 3-mode tensor $\mathcal{T} \in \mathbb{R}^{D \times D \times D}$,
Second order moment as matrix $M \in \mathbb{R}^{D \times D}$.
Initialise:
Choose an initial vectors of parameters $\underline{\mu}_1, \ldots, \underline{\mu}_K$, $\underline{w}$ and learning rates α, β, and penalty parameter λ.
for $n = 1, 2, ...,$epochs **do**
  for $k = 1, 2, ..., K$ **do**
    Compute $\nabla f(\underline{\mu}_k^n | {w_k}^n)$, $\nabla g(\underline{\mu}_k^n | {w_k}^n)$, $\nabla f({w_k}^n | \underline{\mu}_k^n)$ and $\nabla g({w_k}^n | \underline{\mu}_k^n)$ from equations (5) to (8);
    Update $\hat{\underline{\mu}}_k^{n+1} \leftarrow \hat{\underline{\mu}}_k^n - \alpha_n(\nabla f(\underline{\mu}_k^n | w_k^n) + \lambda \nabla g(\underline{\mu}_k^n | w_k^n))$;
    Update $\hat{w}_k^{n+1} \leftarrow \hat{w}_k^n - \beta_n(\nabla f(w_k^n | \underline{\mu}_k^n) + \lambda \nabla g(w_k^n | \underline{\mu}_k^n))$;
  end for
end for
Output:
Return $\hat{\underline{\mu}}_1, \ldots, \hat{\underline{\mu}}_K$ and $\hat{\mathbf{w}}$.

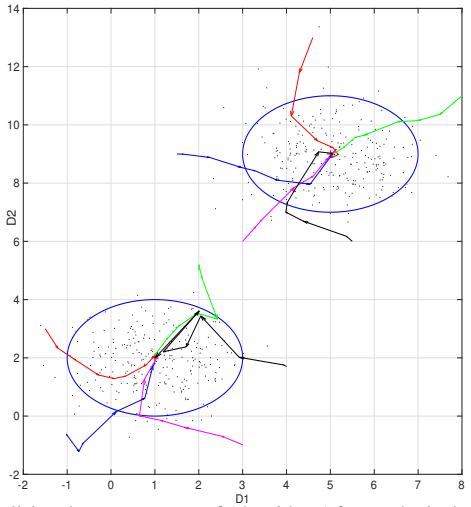


Fig. 1: Visualising the convergence of Algorithm 1 for synthetic data: spherical Gaussian mixture ($K = 2$, $D = 2$). Dots in different colours show different starting points for the algorithm. Only 5 paths are shown for clarity.

To illustrate AGD-MoM in operation, Figure 1 shows 5 different pairs of initial points and their sample paths as the algorithm progresses. Specifically, a simple two-dimensional two-components GMM ($K = 2$ and $D = 2$) is illustrated. Given two pairs of randomly selected points, gradient descent results in estimates close to the true (known) mean for both clusters and eventually converges to a local optimum. As far as convergence is concerned, AGD-MoM eventually converges to the correct solution in this case, regardless of the initial starting points; full analysis of convergence is left to Section 5.

4. Application to streaming data

In this Section we show how our algorithm may be extended to cater for streaming data in which the cluster centres are evolving over time. In streaming scenarios, it is typical that a significant amount of data needs to be quickly processed in real time, while historical data is typically jettisoned and no longer available. There exist few algorithms for performing online clustering via GMM; see, for example [24, 25, 26, 27] and

references therein. The majority of these algorithms are based on a split and merge criterion to add or remove clusters, followed by a rearrangement with an EM approach (see [24]). The IGMM algorithm proposed in [24] has been further extended to allow for evolving data either by dropping old data entirely [28] or by use of an exponential filter (i.e. a forgetting factor) as in [29]. In [26] it is argued that merge and split approaches are either too slow for online learning [25] or do not guarantee the accuracy of the resulting model. So later, a modification of the tensor decomposition algorithm for such online setting has been explored in [30], for community detection and topic modelling but not for GMM. Their study shows that although a moment-based formulation may seem computationally expensive and complicated at first sight, implementing implicit *tensor* operations leads to significant speed-ups and guarantees a learning process as opposed to several heuristic approaches [30]. The above would suggest that while split and merge may be applicable to AGD-MoM, the key advantage of our approach is that tensor updates based on newly arrived data may be computationally efficient and lead to improved parameter estimates.

Assume that the incoming data points arrive in sequential batches (or chunks [31]), $S_1, S_2, \ldots, S_m$, and that the next data batch to arrive is S_{m+1}. Upon the arrival S_{m+1}, the empirical moments must be updated. This involves updating C_x, and then $\mathcal{T}, M$; essentially these operations are a weighted combination of the old and new variables (see [32]); details of the *update* function are left to the Supplementary Material. The weighting uses a forgetting factor $\gamma \in [0, 1]$ which allows the updates to act as an exponential filter, thus allowing the evolving cluster centres to be tracked (as is used in IGMM [29]). Given updated moments AGD-MoM may be applied with the previous parameters used as initial estimates, denoted as AGD-MoM $(\mathcal{T}^m, M^m \mid \{\hat{\underline{\mu}}^{m-1}_{1:K}, \hat{\underline{w}}^{m-1}\})$. Therefore, the number of iterations taken by gradient descents to converge can be drastically reduced. Algorithm 2 gives a pseudo-code for implementing the Incremental-AGD-MoM (I-AGD-SGD) algorithm.

Algorithm 2 Incremental AGD-MoM for online learning:

Input:
  An initial data batch, S_1, and exponential weighting. γ
Initialise:
  Choose initial parameters $\{\underline{\mu}^0_{1:K}, \underline{w}^0\}$, α, β, λ.
for $m = 2, \ldots$ **do**
  while awaiting batch S_m **do**
    $\{\hat{\underline{\mu}}^{m-1}_{1:K}, \hat{\underline{w}}^{m-1}\} \leftarrow$ AGD-MoM $(\mathcal{T}^{m-1}, M^{m-1} \mid \{\hat{\underline{\mu}}^{m-2}_{1:K}, \hat{\underline{w}}^{m-2}\})$.
  end while
  Output:
    Return $\{\hat{\underline{\mu}}^{m-1}_{1:K}, \hat{\underline{w}}^{m-1}\}$
  $\{\mathcal{T}^m, M^m\} \leftarrow update(\mathcal{T}^{m-1}, M^{m-1} \mid C^m, \gamma)$
end for

5. Empirical results

In this section, we present the empirical performance of our algorithm and compare it to the state-of-the-art algorithms such as the EM algorithm, the Variational Bayes (VB) method, and Tensor decomposition using the power iteration method (TPM).[6] We perform experiments for both synthetic and real data. In the first set of experiments, we begin with a simple 2-D case examining the convergence and results in detail. The analysis is then extended to higher dimensions. The second experiment is conducted on two real datasets from the UCI repository[7]: the Iris and Wine datasets. Last but not least, the incremental AGD-MoM is tested on a large set of synthetic streaming data.

Our algorithm settings are as follows for all experiments. The learning rates of AGD, α and β, are set to lower values between 10^{-4} to 10^{-5} in order to ensure that the algorithm does not miss a local optimum while keeping the computational cost as low as possible. The maximum number of epochs is set to 1000. We found $\lambda = 10$ as a suitable choice for the penalty parameter. Each experiment is run 100 time with different random starting points allowing us to estimate parameter statistics.[8] The same starting points are used for all algorithms.[9] *The estimation accuracy is measured using the Mean Absolute Percentage Error (MAPE). The MAPE is considered because of its scale independence and also its ease of interpretation.*[10]

5.1. Synthetic data experiments

In the following experiment a synthetic dataset is generated to examine in detail the convergence rate and results of Algorithm 1. The purpose of this experiment is to determine which of the three aforementioned optimisation techniques, standard gradient descent, Momentum and Adam (Section 3), speeds up the converges rate. To better visualise our results, a set of two-dimensional data are generated from a two-component GMM (N=500). Figure 2a shows the average convergence error rate obtained by standard gradient descent, Momentum and Adam. As can be seen, Momentum converges slightly faster than Adam, both having essentially converged after approximately 100 epochs, while GD does not converge until 400 epochs and then requires an asymptotic number of iterations to reach final convergence.[11]

Our empirical results further show that GD with a fixed learning rate achieves the optimum minimum, but requires significantly more epochs than Adam and Momentum to do so. In order to provide a closer picture of how these algorithms perform regarding 100 different starting points, box-plots of $\mathbf{w}$, $\underline{\mu}_1$ and $\underline{\mu}_2$ are shown in Figures 2b to 2c. Figure 2b shows the Momentum estimates are slightly more concentrated than Adam, while, the GD box-plot implies a far wider range for $\mathbf{w} = 0.5$. On the

[6] The implementation may be found in [4].

[7] http://archive.ics.uci.edu/ml

[8] Specifically, the sensitivity of the solutions to the initial starting point.

[9] We implemented all algorithms for our synthetic data experiments using *MATLAB*.

[10] *MAPE is calculated as the average of the unsigned differences between an estimated value and the known true value divided by the known true value. In other words, we use the formula* $\frac{1}{n}\sum_{i=1}^{n} \|\underline{e}_i\|_1$ *where* $\underline{e}_i = (\frac{\mu_{i1}-\hat{\mu}_{i1}}{\mu_{i1}}, \ldots, \frac{\mu_{in}-\hat{\mu}_{id}}{\mu_{id}})$ *and* $\|\ .\ \|_1$ *is* ℓ_1-*norm. is used as the main measure of estimation accuracy.*

[11] Most machine learning algorithms exhibit the behaviour in Figure 2, quickly obtaining a '*reasonable*' solution and then requiring many more iterations to refine that solution. When these algorithms are being run on the cloud, time equates directly to monetary cost, and so many applications seek algorithms that provide the fastest reasonable solution.

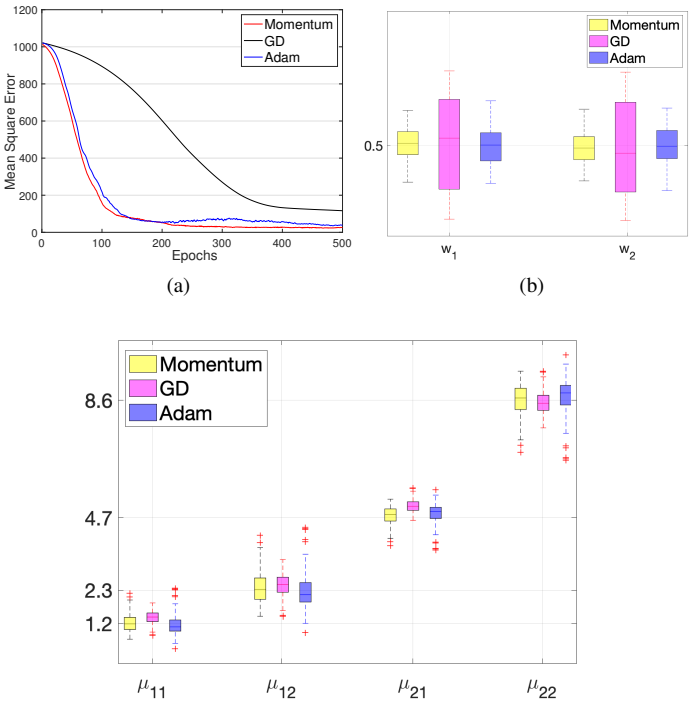


Fig. 2: Convergence of algorithm (1) in a synthetic setting over 100 runs: (a) Summary of average convergence error rate GD, momentum and Adam. Boxplots for (b) w_1 and w_2, (c) $\underline{\boldsymbol{\mu}}_1$ and $\underline{\boldsymbol{\mu}}_2$. The known true values for each component are indicated in y-axis.

other hand, for both mean components, $\underline{\boldsymbol{\mu}}_1$ and $\underline{\boldsymbol{\mu}}_2$, GD results in estimates with a lower variance but a distinct bias. As can be seen in Figure 2c, both Adam and Momentum span much the same range of values and their medians are roughly around the true known values (y-axis). These experiments (together with other cases presented in the Supplementary Material) indicate that overall Momentum and Adam provide the fastest convergence with a lower bias. We can conclude that the learning rate does not need to be tuned according to the scale of the smoothness of the gradient function but likely achieves better results by dampening the oscillations via Momentum.

In our next experiment, we focus on higher order GMMs to examine how the algorithm behaves in the presence of multiple local minima. Figure 3 shows the MAPE achieved by AGD-MoM, TPM, VB and the EM algorithm for GMM with $K = 5$, and 10 components. Figures 3a and 3b, for $K = 5$, show that AGD performs significantly better than the alternatives in all cases. Note also, that Figures 3c and 3d show that TPM performs well when K and D are large.

5.2. *Benchmark dataset experiments*

Iris datasets. ($N = 150, K = 3, D = 4$) One of the most well-known datasets in classification is the Iris dataset. The dataset contains 3 types of Iris plants, with 50 instances of each. Four features were measured from each sample, sepal length, sepal width, petal length, and petal width. The MAPE's achieved by each of the competing algorithms are shown in Figures 4a and 4b. Note that for each parameter the lowest MAPE is always provided by one of the moment based methods (i.e. TPM or AGD). AGD does not provide a good estimate for w_1 (Figure 4b) but outperforms the other techniques otherwise.

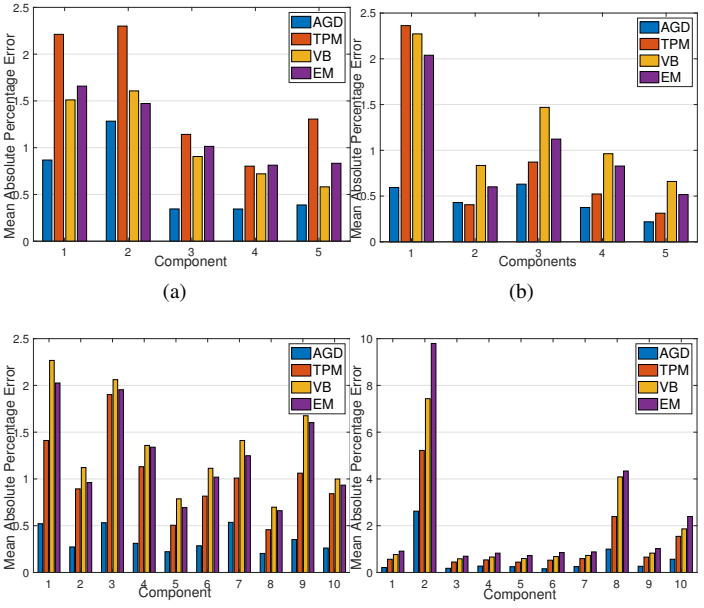


Fig. 3: The MAPE achieved by the AGD-MoM, TPM, VB and EM algorithms, in a GMM with (a) $K = 5, D = 5, N = 1000$ (b) $K = 5, D = 20, N = 1000$ (c) $K = 10, D = 20, N = 5000$ (d) $K = 10, D = 50, N = 5000$.

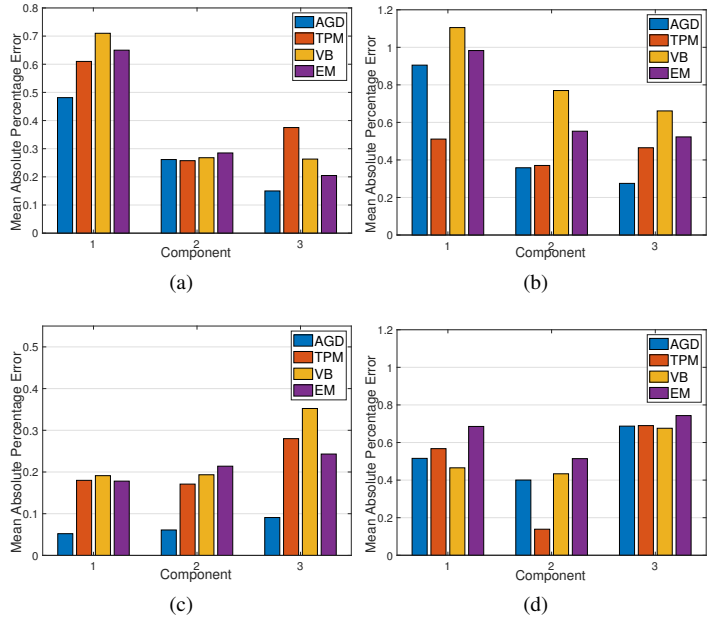


Fig. 4: The MAPE achieved by the AGD, TPM, VB and EM algorithms (a) $\underline{\boldsymbol{\mu}}$ (Iris data), (b) $\mathbf{w}$ (Iris data), (c) $\underline{\boldsymbol{\mu}}$ (Wine data) and (d) $\mathbf{w}$ (Wine data). The results are averaged over 100 runs.

Wine data set. ($N = 178, K = 3, D = 13$) The Wine data set consists of wine samples from 3 different culitvars (known classes) grown in the same region in Italy, using 13 features (derived from chemical tests of constituent elements). Figures 4c and 4d show the MAPE's achieved by the competing algorithms. As can be seen, AGD results in a better mean estimate in all cases, however, TPM gives a better result for one of the weight estimates. *In addition to MAPE, we have also reported the trace of the error covariance matrix [33], the overall uncertainty reported by the estimators, for the estimates obtained from AGD-MoM, TPM, EM and VB in Table 1. It is formulated as the empirical sum of square errors over all dimensions,*

Table 1: The sum of the trace of the error covariance matrix

	Iris		Wine	
Methods	μ	w	μ	w
AGD	**0.36**	0.55	**0.13**	0.73
TPM	1.06	**0.53**	0.59	**0.71**
VB	1.34	2.07	1.82	1.04
EM	1.24	1.30	1.46	1.53

$\mathbf{tr}\Big(E[(\underline{\mu}_k - \underline{\hat{\mu}}_k^n)(\underline{\mu}_k - \underline{\hat{\mu}}_k^n)^T]\Big)$ *where the expectation is over* $n = 100$ *runs, where* $\underline{\hat{\mu}}_k^n$ *is the estimated mean obtained at iteration* n^{th}. As can be seen, the AGD-MoM results in significantly better estimates for the cluster centres. For the cluster weights the AGD-MoM estimates are comparable to those from TPM and significantly better than VB and EM. Last but not least we focus on empirical testing of the convergence rate for AGD and TPM. Figures 5a, 5b and 5c show the convergence error rate of loss function obtained by moment-based methods for simulated, Iris and Wine datasets, respectively. As can be seen TPM converges to its solution faster than AGD does. Note however, that the two solutions differ in their *accuracies* as discussed above.

5.3. Streaming experiments

In our final experiment, we examine Incremental AGD-MoM for online learning. The scenario we specifically examine is that in which the cluster centres are evolving over time. This is a common occurrence in real-world data and occurs when *causal* but *unmeasured* variables also evolve (ex: the *weather* influence on wine quality year on year or trending topics on the internet etc.). The baseline algorithm for comparison is IGMM from [29] which employs a forgetting factor, γ. Note that γ for IGMM and I-AGD-SGD play the same role and are set equal to 0.7 in all experiments. Appropriate selection of forgetting factors is discussed in many text books (ex:[34]). Here the high forgetting rate is a consequence of the large batch size.

The basic goal of this experiment is to iteratively move cluster centres to gradually reduce the distances between cluster centres and their associated data points (Figure 6a). In order to achieve that, we generated two sets of 2-Dimensional mean vectors from two parametrised curves in $\mathbb{R}^2$ given by $\Big(cos(\frac{t\pi}{b_1} + a_1)t, cos(\frac{t\pi}{b_2} + a_2)t\Big)$ and $\Big(cos(\frac{t\pi}{d_1} + c_1)(t - 100), cos(\frac{t\pi}{d_2} + c_2)(t - 100)\Big)$, where $t = (1, \ldots, 100)$, $a = (a_1, a_2), b = (b_1, b_2), c = (c_1, c_2)$ and $d = (d_1, d_2)$ are chosen from $[50, 125]$ and $[15, 25]$. This provides a set of cluster centres which *twist like a rope segment* in time providing a nice challenge to the algorithm. We have generated 500,000 points form these 2-D mean vectors; 100 batches with size 5000. Figure 6a shows a heat map of the simulated data points drifting slowly when the new batch becomes available.

Figure 6b displays cluster centres obtained using the I-AGD-MoM algorithm given 500 epochs. The result is that the algorithm tracks the centres quite tightly. Next we drop the number of epochs available to 100 mimicking the event when less time is available between batches. As Figure 6c demonstrates, initially the algorithm has trouble tracking the evolution but after a burn-in period tracks onto the evolution. It then looses this track again for two subsequent periods examined below. Figures 7a and 7b show the MAPE as a function of time for I-AGD-MoM and IGMM. Note that there is an initial burn-in period before both techniques start to track the evolving centres. We can also observe that for $\underline{\mu}_2$ (7b) there are two periods (times 10, 60) where the MAPE, for both I-AGD-MoM and IGMM, jump due to a change in direction of the centre evolution (as seen by the gradient plotted alongside the MAPE in Figure 7b). However, as can be seen I-AGD-MoM deviates less from the true track than IGMM.

6. Conclusion

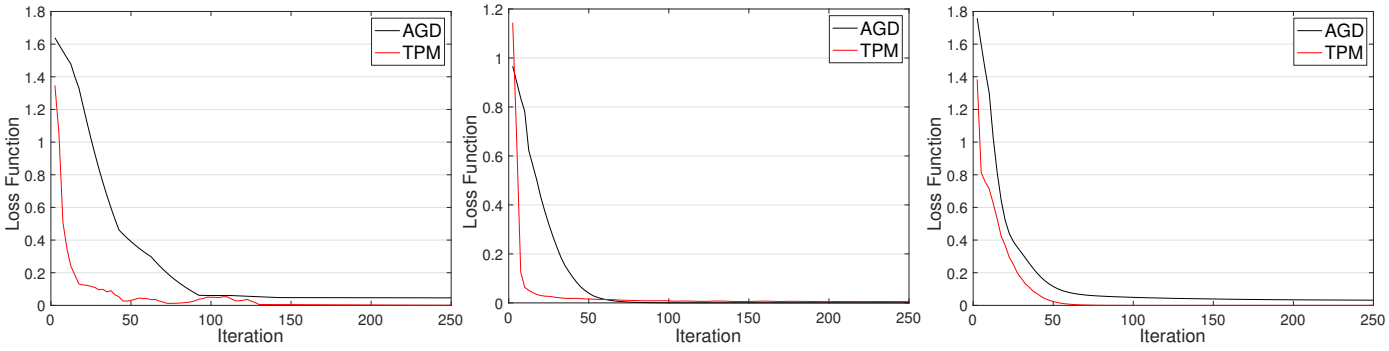
In this paper, we have presented an alternative approach to estimate parameters of a spherical Gaussian mixture model. The approach we present is based on matching second and third order moments which we solve by an alternating gradient descent approach. The method is shown to converge reliably on synthetic data where the assumptions about spherical covariances are met. We also demonstrate the performance of the algorithm on benchmark datasets from classification problems in which the ground truth of cluster centres is known from the class labels. Our experiments show that AGD-MoM leads to better estimates for most parameters but may take longer than TPM to converge suggesting that the best choice of algorithm depends on the problem at hand. Furthermore, we show that the approach is suitable for streaming data in an incremental mode, using previously estimated parameters to deal with newly arriving data. In addition, the computational complexity our algorithm is essentially independent of the size of the data set unlike the classical EM algorithm. Our current work is focused on extending this framework to relax the assumptions of isotropic covariances and the applicability of the algorithm to large-scale clustering problems. In addition, use of (one of the many) adaptive forgetting factor algorithms may prove useful.

Acknowledgements

This work was supported by an EPSRC grant EP/N014189/1.

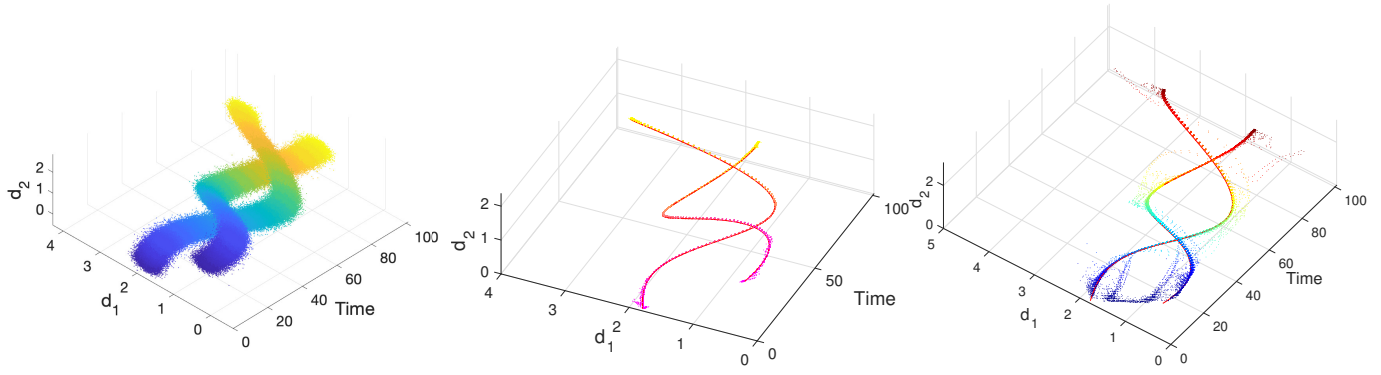
References

[1] C. P. Robert, Monte carlo methods, Wiley Online Library, 2004.
[2] C. F. J. Wu, On the Convergence Properties of the EM Algorithm, The Annals of Statistics 11 (1) (1983) 95–103.
[3] R. A. Redner, H. F. Walker, Mixture densities, maximum likelihood and the EM algorithm, SIAM review 26 (2) (1984) 195–239.
[4] A. Ray, J. Neeman, S. Sanghavi, S. Shakkottai, The Search Problem in Mixture Models, JMLR. 18 (2016) 206:1–206:61.
[5] Z. Chen, S. Haykin, J. J. Eggermont, S. Becker, Correlative Learning: A Basis for Brain and Adaptive Systems, Wiley-Interscience, 2007.
[6] L. Le Cam, The central limit theorem around 1935, Statistical science (1986) 78–91.
[7] B. G. Lindsay, P. Basak, Multivariate normal mixtures: a fast consistent method of moments, JASA 88 (422) (1993) 468–476.
[8] D. Hsu, S. M. Kakade, Learning mixtures of spherical gaussians: moment methods and spectral decompositions, in: Proceedings of the 4th conference on Innovations in Theoretical Computer Science, ACM, 11–20, 2013.
[9] T. G. Kolda, B. W. Bader, Tensor decompositions and applications, SIAM review 51 (3) (2009) 455–500.
[10] M. Anandkumar, A., Ge, R., Hsu, D. J., Kakade, S. M., & Telgarsky, Tensor Decompositions for Learning Latent Variable Models, JMLR 15 (2014) 2773–2832, ISSN 1532-4435.
[11] K. Chaudhuri, S. Rao, Learning Mixtures of Product Distributions Using Correlations and Independence., in: COLT: 21st Annual Conference on Learning Theory, vol. 4, 9–20, 2008.



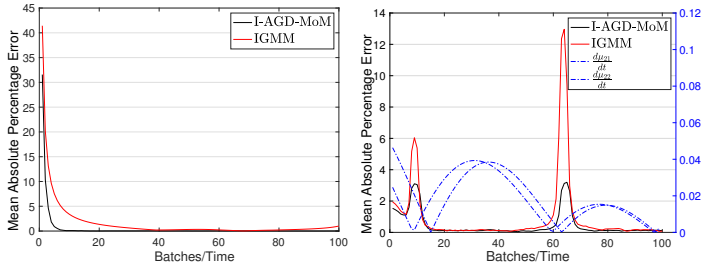
(a) (b) (c)

Fig. 5: Comparisons of convergence rate between TPM and AGD for (a) Simulated datasets (b) Iris datasets (c) Wine datasets.



(a) (b) (c)

Fig. 6: Convergence of I-AGD-MoM in a streaming setting over 100 runs: (a) synthetic data ($K = 2, D = 2$) (the third dimension is time), with the cluster centres slowly drifting (see text); (b) trajectories of estimated cluster centres (nepochs=500), red line shows the true mean and dots around it show the AGD-MoM estimated mean; and (c) trajectories of estimated means (nepochs=100), red line shows the true mean and (100) dots around it show the I-AGD-MoM estimated.



(a) (b)

Fig. 7: The MAPE comparison between AGD and IGMM in a streaming setting over 100 runs (a) $\underline{\mu}_1$, (b) $\underline{\mu}_2$. Noted that dashed blue line shows the gradient alongside the MAPE.

[12] S. Dasgupta, L. Schulman, A probabilistic analysis of EM for mixtures of separated, spherical Gaussians, JMLR 8 (Feb) (2007) 203–226.

[13] L. De Lathauwer, B. De Moor, J. Vandewalle, On the best rank-1 and rank-($r_1, r_2, \ldots, r_n$) approximation of higher-order tensors, SIAM journal on Matrix Analysis and Applications 21 (4) (2000) 1324–1342.

[14] A. Shashua, T. Hazan, Non-negative tensor factorization with applications to statistics and computer vision, in: Proceedings of the 22nd international conference on Machine learning, ACM, 792–799, 2005.

[15] J. Liu, P. Wonka, J. Ye, Sparse non-negative tensor factorization using columnwise coordinate descent, Pattern Recognition 45 (2012) 649–656.

[16] Y. Xu, W. Yin, A block coordinate descent method for regularized multi-convex optimization with applications to nonnegative tensor factorization and completion, SIAM Journal on imaging sciences 6 (2013) 1758–1789.

[17] J. Nocedal, S. Wright, Numerical optimization, Springer Science & Business Media, 2006.

[18] V. Kuleshov, A. Chaganty, P. Liang, Tensor factorization via matrix factorization, in: Artificial Intelligence and Statistics, 507–516, 2015.

[19] T. G. Kolda, Numerical optimization for symmetric tensor decomposition, Mathematical Programming 151 (1) (2015) 225–248.

[20] N. Qian, On the momentum term in gradient descent learning algorithms, Neural networks 12 (1) (1999) 145–151.

[21] Y. Nesterov, A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$, in: Doklady AN USSR, vol. 269, 543–547, 1983.

[22] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980.

[23] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, JMLR 12 (Jul) (2011) 2121–2159.

[24] M. Song, H. Wang, Highly efficient incremental estimation of Gaussian mixture models for online data stream clustering, in: Intelligent Computing: Theory and Applications III, vol. 5803, 174–184, 2005.

[25] P. M. Hall, Y. Hicks, T. Robinson, A method to add Gaussian mixture models, University of Bath, Department of Computer Science, 2005.

[26] A. Declercq, J. H. Piater, Online Learning of Gaussian Mixture Models-a Two-Level Approach., in: VISAPP (1), 605–611, 2008.

[27] R. C. Pinto, P. M. Engel, A fast incremental Gaussian mixture model, PloS one 10 (10) (2015) e0139931.

[28] L. S. Oliveira, G. E. A. P. A. Batista, IGMM-CD: A Gaussian Mixture Classification Algorithm for Data Streams with Concept Drifts, in: 2015 Brazilian Conference on Intelligent Systems (BRACIS), 55–61, 2015.

[29] C. A. Juan M. Acevedo-Valle, Karla Trejo, Multivariate Regression with Incremental Learning of Gaussian Mixture Models, in: Recent Advances in Artificial Intelligence Research and Development, vol. 300, IOS Press, 196 – 205, 2017.

[30] F. Huang, U. Niranjan, M. U. Hakeem, A. Anandkumar, Online tensor methods for learning latent variable models, JMLR 16.

[31] J. Gama, Knowledge discovery from data streams, CRC Press, 2010.

[32] E. Schubert, M. Gertz, Numerically Stable Parallel Computation of (Co)Variance, in: SSDBM, 10:1–10:12, 2018.

[33] K. Dormann, B. Noack, U. Hanebeck, Optimally Distributed Kalman Filtering with Data-Driven Communication, Sensors 18 (2018) 1034.

[34] S. Haykin, Adaptive filter theory, Prentice Hall, Upper Saddle River, NJ, 4th edn., 2002.

Research Highlights (Required)

To create your highlights, please type the highlights against each `\item` command.

It should be short collection of bullet points that convey the core findings of the article. It should include 3 to 5 bullet points (maximum 85 characters, including spaces, per bullet point.)

- Moment matching to estimate GMM is considered as a constrained optimisation problem.
- The AGD algorithm is used to solve the moment matching optimisation problem.
- Our algorithm empirically outperforms EM and VB in converging to correct solutions.
- An online version of the algorithm is presented which outperforms IGMM.

Pattern Recognition Letters
journal homepage: www.elsevier.com

Estimation of Gaussian Mixture Models via Tensor Moments with Application to Online Learning

Donya Rahmani[a,**], Mahesan Niranjan[b], Damien Fay[c], Akiko Takeda[d], Jacek Brodzki[a]

[a]School of Mathematics, University of Southampton, Highfield, Southampton, UK
[b]School of Electronics and Computer Science, University of Southampton, Southampton, UK
[c]Department of Analytics, Logicblox/Infor, Atlanta, GA
[d]Department of Creative Informatics, The University of Tokyo, Tokyo, Japan

ABSTRACT

In this paper, we present an alternating gradient descent algorithm for estimating parameters of a spherical Gaussian mixture model by the method of moments (AGD-MoM). We formulate the problem as a constrained optimisation problem which simultaneously matches the third order moments from the data, represented as a tensor, and the second order moment, which is the empirical covariance matrix. We derive the necessary gradients (and second derivatives), and use them to implement alternating gradient search to estimate the parameters of the model. We show that the proposed method is applicable in both a batch as well as in a streaming (online) setting. Using synthetic and benchmark datasets, we demonstrate empirically that the proposed algorithm outperforms the more classical algorithms like Expectation Maximisation and variational Bayes.

1. Introduction

The Gaussian mixture model (GMM) is a widely used and extremely practical probabilistic modelling tool used in statistics and machine learning. It is a powerful model for data clustering problems where the data is assumed to be generated from a mixture of several distinct underlying probability distributions. Central to a successful application of a GMM is the parameter estimation technique. This topic dates back more than a hundred years [1], and the most commonly used method is the Expectation-Maximisation (EM) algorithm which is based on maximum likelihood estimation. However, it has long been known that the EM algorithm may be slow to converge (as first noted in [2] and expanded in [3, 4]). Generally speaking, the EM algorithm works best when the component densities are well separated, but convergence may be exorbitantly slow and result in estimates with high variance when they are poorly separated [3]. In addition, the computational complexity for the EM algorithm depends significantly on the number of samples. When clustering N samples of D-dimensional data into K clusters this complexity grows as $O(DN + KN^2)$, [5]. As an alternative, in this paper, we investigate a variant of the Method of Moments (MoM) based on Alternating Gradient Descent (AGD). The earliest known application of the MoM was provided by Chebyshev (1887) in his study of the central limit theorem in statistics. This was then followed by Karl Pearson in his classic work on MoM in 1894, [6]. Pearson first used the MoM to estimate the five parameters of a two-component univariate Gaussian mixture. After Pearson, the MoM became one of the most popular ways of estimating the parameters of a finite mixture distribution (see [7, 1, 8]). One reason is that MoM estimators impose fewer restrictions on the model compared to MLE.

The basic idea behind a MoM estimator is that, given a sufficient number of moments, one may match the theoretical and sample moments to estimate the parameters. More specifically, the moment based approach solves systems of multivariate polynomial equations for computing all solutions of a given model without any prerequisite assumption. However, this can be computationally challenging. One approach proposed in [7] suggests that to speed up the computation one can transform the moment equations into a set of related linear equations (in essence marginal distributions, which can be solved efficiently), and one non-linear equation (which is cubic and can also be solved efficiently). In addition, a successful implementation of the MoM should yield estimators that are statistically effi-

**Corresponding author: Tel.: +0-44-2380525156;
e-mail: d.rahmani@soton.ac.uk (Donya Rahmani)

cient in the sense that their expectations maximise the likelihood function, [7]. Thus, the challenge in using the MoM approach is to preserve high statistical efficiency while reducing the computational burden.

In recent years, tensor based methods have received much attention together with the development of computationally efficient algorithms (see [9] for an excellent overview). These methods can be directly used for computing MoM estimators. For example, $n-$order moments can be regarded as an $n-$mode tensor which can then be simplified using one of several tensor decomposition methods. This is the core of the MoM approach used by [8] and also by Anadkumar et al. in [10]. They propose a moment matching estimator for mixtures of spherical Gaussians using a simple spectral decomposition of lower order moments obtained from the data. Their method uses the eigenvalue decomposition of symmetric matrices (see [8]) to decrease the computational complexity of the MoM. Note that the eigenvalues are found via the power iteration method and deflation method which may introduce errors in the parameter estimates due to approximation inherent in deflation methods.

In this paper, we propose an alternative approach to this problem. In particular, we reformulate the moment matching problem as a constrained optimisation problem which can be solved via Alternating Gradient Descent (AGD-MoM). This allows us to obtain the required estimators by a direct computation rather than through an iterative procedure. As will be seen, this leads to estimates with increased accuracy over the competing techniques. Furthermore, we demonstrate that in the case of online streaming data the algorithm tracks the evolution of the cluster centres with greater accuracy and less computational expense. In addition, as our method requires storage of the moments of the streaming data, the memory requirements are efficient and independent of the number of samples.

The remainder of the paper is organised as follows. In Section 2 we briefly review the moment-based estimation problem. Section 3 presents our proposed algorithm. Section 4 demonstrates how AGD-MoM can be adapted in an online setting. Section 5 compares AGD-MoM with several competing methods using both synthetic and real data. Finally, Section 6 draws conclusions and suggests avenues for future work.[1]

2. Moment-based estimation

A spherical GMM assumes that the D dimensional data observations are sampled from K clusters. The probability of sampling cluster $k \in \{1, \dots, K\}$ is given by a mixing weight vector, $\underline{\mathbf{w}} = (w_1, \dots, w_K)$, $w_k \in [0, 1]$, and $\sum_k w_k = 1$. Each cluster has a centre denoted by $\underline{\boldsymbol{\mu}}_k \in \mathbb{R}^D$ with an associated isometric covariance matrix $\sigma_k^2 I$. Denote a sample from this data generating process as $\underline{\mathbf{x}} = \underline{\boldsymbol{\mu}}_h + \underline{\mathbf{z}}$ where $\underline{\mathbf{x}} \in \mathbb{R}^D$, $\underline{\mathbf{z}}$ is assumed to be a random vector whose conditional distribution given $h = k$ is $N(0, \sigma_h^2 I)$, and $P(h = k) = w_k$ for $k \in [K]$. The goal of GMM estimation is to optimally recover the parameters $\{\underline{\boldsymbol{\mu}}_{1:K}, \sigma_{1:K}^2, w_{1:K}\}$ given a set of observations $\underline{\mathbf{x}}$.

[1] An implementation of our algorithm in tensorflow may be found here: https://github.com/drahmani/AGD-MoM.

With respect to GMM, the MoM solves multivariate polynomial equations, pairs the sample moments with their corresponding theoretical moments, to find the parameters of the model. In this paper, we examine a computationally efficient optimisation procedure to recover the parameters of GMM using the MoM. Our method assumes the non-degeneracy condition [8] is met. We recall that the non-degeneracy condition states that the component mean vectors, $\underline{\boldsymbol{\mu}}_1, \underline{\boldsymbol{\mu}}_2, \dots, \underline{\boldsymbol{\mu}}_K$, are linearly independent, and the mixing components $w_1, w_2, \dots, w_K > 0$, are strictly positive and sum to one. Note also that the non-degeneracy condition places a weaker restriction on the data generating process than the *spreading condition* from standard (e.g., EM) density estimation approaches (see [11, 12] for specifics). Given the non-degeneracy condition, the sample moments can be expressed as a sum of rank one tensors and therefore the relationship between the GMM parameters and their corresponding moments can be expressed by the following theorem.

Theorem 1. *(Hsu and Kakade, 2012) Assume $D \geq K$. The average variance $\bar{\sigma}^2 = \sum_{k=1}^K w_k \sigma_k^2$ is the smallest eigenvalue of the covariance matrix $C_{\underline{\mathbf{x}}} = E[\underline{\mathbf{x}} \otimes \underline{\mathbf{x}}] - E[\underline{\mathbf{x}}] \otimes E[\underline{\mathbf{x}}]$. Let, $\underline{\boldsymbol{\nu}}$ be any unit norm eigenvector corresponding to the eigenvalue $\bar{\sigma}^2$. Furthermore, if*

$$\underline{\boldsymbol{\mu}} = E[\underline{\mathbf{x}}(\underline{\boldsymbol{\nu}}^T(\underline{\mathbf{x}} - E[\underline{\mathbf{x}}]))^2],$$
$$M = E[\underline{\mathbf{x}} \otimes \underline{\mathbf{x}}] - \bar{\sigma}^2 I,$$
$$\mathcal{T} = E[\underline{\mathbf{x}} \otimes \underline{\mathbf{x}} \otimes \underline{\mathbf{x}}] - (\sum_{i=1}^D \underline{\boldsymbol{\mu}} \otimes e_i \otimes e_i + e_i \otimes \underline{\boldsymbol{\mu}} \otimes e_i + e_i \otimes e_i \otimes \underline{\boldsymbol{\mu}}),$$

where $\underline{\boldsymbol{\mu}} \in \mathbb{R}^D$, $M \in \mathbb{R}^{D \times D}$ and $\mathcal{T} \in \mathbb{R}^{D \times D \times D}$, then

$$M = \sum_{k=1}^K w_k \underline{\boldsymbol{\mu}}_k \otimes \underline{\boldsymbol{\mu}}_k, \qquad \mathcal{T} = \sum_{k=1}^K w_k \underline{\boldsymbol{\mu}}_k \otimes \underline{\boldsymbol{\mu}}_k \otimes \underline{\boldsymbol{\mu}}_k.$$

The empirical moments converge to the exact moments at a rate of $O(N^{\frac{-1}{2}})$ [8]. As the lower order moments, M and $\mathcal{T}$, are orthogonally decomposable tensors consequently the moment matching problem can be cast as eigenvalue decomposition of symmetric matrices. Such orthogonal decomposition problems can be efficiently solved by iterative approaches like the power iteration method, fixed-point iteration and gradient descent, see [13, 10, 8]. An orthogonal decomposition of a symmetric tensor $\mathcal{T} \in \mathbb{R}^{D \times D \times D}$ is a collection of orthonormal (unit) vectors $\underline{\mathbf{v}}_1, \underline{\mathbf{v}}_2, \dots, \underline{\mathbf{v}}_K$ together with corresponding positive scalars $\lambda_k > 0$ such that $\mathcal{T} = \sum_{k=1}^K \lambda_k \underline{\mathbf{v}}_k^{\otimes 3}$. Anadkumar et al. [10] use the tensor power method of Lathauwer et. al. proposed in [13] to obtain robust estimates of eigenvector/eigenvalue pairs, $\{(\underline{\mathbf{v}}_k, \lambda_k)\}$, for orthogonal tensor decomposition. Note that the orthogonality is required only for the whitened mean vectors (not original $\underline{\boldsymbol{\mu}}_i$'s), and this is guaranteed under the aforementioned non-degeneracy conditions. It is also worth pointing out that orthogonal decompositions do not necessarily exist for every symmetric tensor. A thorough review of these models goes beyond the scope of this paper.

We propose here an alternative approach to extract the orthogonal decomposition of M and $\mathcal{T}$ by reformulating the moment matching problem as a constrained optimisation problem. We provide a detailed description in the following section.

3. Learning mixture of spherical Gaussian via alternating gradient descent: AGD-MoM

In this section, we first reformulate the moment matching problem as a constrained optimisation problem and then present an AGD algorithm for GMM parameter estimation. We start by reformulating Theorem 1 in terms of minimising the objective function:

$$\min_{\substack{w_1,\dots,w_k\\ \underline{\mu}_1,\dots,\underline{\mu}_k}} \|\mathcal{T} - \sum_{k=1}^{K} w_k \underline{\mu}_k \otimes \underline{\mu}_k \otimes \underline{\mu}_k\|_F^2, \tag{1}$$

$$\text{subject to} \qquad M = \sum_{k=1}^{K} w_k \underline{\mu}_k \otimes \underline{\mu}_k,$$

where $\|.\|_F$ denotes the Frobenius tensor norm, which is the square root of the sum of the squares of all its elements. Our constrained optimisation problem involves minimising the discrepancy between the empirical tensor and exact tensor subject to a constraint. Examples of such problems for non-negative tensor factorisation can be found in [14, 15, 16].

Here, we require a more robust approach to find a set of efficient and optimal solutions to minimise the objective function (1). We use the *quadratic penalty method* [17] which is one way to solve this type of constrained optimisation problem. Note that equation (1) can be presented in a quadratic penalty format by adding the square of the violation of the equality constrained terms to the cost function (see [17]) as follows:

$$\min_{\substack{w_1,\dots,w_k\\ \underline{\mu}_1,\dots,\underline{\mu}_k}} F(w_1,\dots,w_k,\underline{\mu}_1,\dots,\underline{\mu}_k) = f + \lambda g, \tag{2}$$

where

$$f = \|\mathcal{T} - \sum_{k=1}^{K} w_k \underline{\mu}_k \otimes \underline{\mu}_k \otimes \underline{\mu}_k\|_F^2 \quad \text{and} \quad g = \|M - \sum_{k=1}^{K} w_k \underline{\mu}_k \otimes \underline{\mu}_k\|_F^2,$$

and λ is the penalty parameter which can be changed adaptively, according to the difficulty in minimising the equality constraint at each iteration. To examine the convergence properties of equation (2), we rely on Theorem 17.1 from [17]. According to this theorem, every limit point x^* of the sequence of $\{x_t\}$, where x_t is the exact global minimiser of $f(x_t) + \lambda_t \sum_i g_i^2(x_t)$, is a global solution of the problem as $\lambda_t \to \infty$.

As we now see, this unconstrained problem can be solved by an AGD based algorithm, alternating between updates of the two sets of parameters: mixing weight $\underline{\mathbf{w}}$ and component means $\underline{\mu}_1,\dots,\underline{\mu}_K$. Note that AGD is one such algorithms to solve unconstrained optimisation problems, [14, 18]. Using AGD to recover the model parameters, $\underline{\mu}_1,\dots,\underline{\mu}_K$ results in the following solutions:

Fix w_k and update $\underline{\mu}_k : \underline{\mu}_k^{t+1} = \underline{\mu}_k^t - \alpha_t\Big(\nabla f(\underline{\mu}_k^t|w_k^t) + \lambda_t \nabla g(\underline{\mu}_k^t|w_k^t)\Big)$,

Fix $\underline{\mu}_k$ and update $w_k : w_k^{t+1} = w_k^t - \beta_t\Big(\nabla f(w_k^t|\underline{\mu}_k^t) + \lambda_t \nabla g(w_k^t|\underline{\mu}_k^t)\Big)$.

Note that $\nabla f(\underline{\mu}_k^t|w_k^t)$, $\nabla g(\underline{\mu}_k^t|w_k^t)$, $\nabla f(w_k^t|\underline{\mu}_k^t)$ and $\nabla g(w_k^t|\underline{\mu}_k^t)$ are the necessary gradients of the cost function, $f + \lambda g$, with respect to the parameters $\underline{\mu}$ and $\underline{\mathbf{w}}$ and learning rates $\{\alpha_t, \beta_t\}$. The first step in finding these derivatives is to rewrite the error norm as a sum of quadratic terms:

$$f = \sum_{l=1}^{D}\sum_{j=1}^{D}\sum_{i=1}^{D}(t_{ijl} - \sum_{k=1}^{K} w_k \mu_{ki}\mu_{kj}\mu_{kl})^2, \tag{3}$$

$$g = \sum_{j=1}^{D}\sum_{i=1}^{D}(m_{ij} - \sum_{k=1}^{K} w_k \mu_{ki}\mu_{kj})^2. \tag{4}$$

We remark that $\mathcal{T}$ and M are symmetric, meaning the values of the elements[2], t_{ijl} and m_{ij}, are the same under any permutation of the indices, allowing efficient computation.[3] Next, differentiating equations (3) and (4) with respect to the (κ, d) component of $\underline{\mu}$ and the κ^{th} component of $\underline{\mathbf{w}}$, where $\kappa = 1,\dots,K; d = 1,\dots,D$, we have:[4]

$$\frac{\partial f}{\partial \mu_{\kappa d}} = 6w_\kappa \sum_{i=1}^{D} \mu_{\kappa i}^2\Big(\sum_{k=1}^{K} w_k \mu_{kd}\mu_{ki}^2 - t_{idi}\Big) + 12w_\kappa \sum_{\substack{i,j=1\\ j\neq i\neq d}}^{D} \mu_{\kappa i}\mu_{\kappa j}\Big(\sum_{k=1}^{K} w_k \mu_{kd}\mu_{ki}\mu_{kj} - t_{idj}\Big), \tag{5}$$

$$\frac{\partial g}{\partial \mu_{\kappa d}} = 4w_\kappa \sum_{i=1}^{D} \mu_{\kappa i}\Big(\sum_{k=1}^{K} w_k \mu_{kd}\mu_{ki} - m_{id}\Big), \tag{6}$$

$$\frac{\partial f}{\partial w_\kappa} = 2\sum_{i=1}^{D} \mu_{\kappa i}^3\Big(\sum_{k=1}^{K} w_k \mu_{ki}^3 - t_{iii}\Big) + 6\sum_{\substack{i,j=1\\ j\neq i}}^{D} \mu_{\kappa i}^2\mu_{\kappa j}\Big(\sum_{k=1}^{K} w_k \mu_{ki}^2\mu_{kj} - t_{iji}\Big), + 12\sum_{\substack{i,j,l=1\\ l\neq j\neq i}}^{D} \mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa l}\Big(\sum_{k=1}^{K}\sum_{\substack{i,j,l=1\\ i\neq j\neq l}}^{D} w_k \mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa l} - t_{idj}\Big), \tag{7}$$

$$\frac{\partial g}{\partial w_\kappa} = 2\sum_{i=1}^{D} \mu_{\kappa i}^2\Big(\sum_{k=1}^{K} w_k \mu_{ki}^2 - m_{ii}\Big) + 4\sum_{\substack{i,j=1\\ j>i}}^{D} \mu_{\kappa i}\mu_{\kappa j}\Big(\sum_{k=1}^{K} w_k \mu_{\kappa i}\mu_{\kappa j} - m_{ij}\Big). \tag{8}$$

Although the component-by-component differentiation involves cumbersome calculations, it is useful to note the correspondence between several terms in Equations (5) to (8) which need only be computed once. This can be more readily seen when expressed in vector form, following the notation of [19] which results in:

$$\frac{\partial f}{\partial \underline{\mu}_\kappa} = -6w_\kappa \mathcal{T}\underline{\mu}_\kappa^2 + 6w_\kappa \sum_{k=1}^{K} w_k\Big(\underline{\mu}_\kappa^T \underline{\mu}_k\Big)^2 \underline{\mu}_k, \tag{9}$$

$$\frac{\partial g}{\partial \underline{\mu}_\kappa} = 4w_\kappa \sum_{k=1}^{K} w_k\Big(\underline{\mu}_\kappa^T \underline{\mu}_k\Big)\underline{\mu}_k - 4w_\kappa M\underline{\mu}_\kappa, \tag{10}$$

$$\frac{\partial f}{\partial w_\kappa} = -2\mathcal{T}\underline{\mu}_\kappa^3 + 2\sum_{k=1}^{K} w_k\Big(\underline{\mu}_\kappa^T \underline{\mu}_k\Big)^3, \tag{11}$$

$$\frac{\partial g}{\partial w_\kappa} = -2M\underline{\mu}_\kappa^2 + 2\sum_{k=1}^{K} w_k\Big(\underline{\mu}_\kappa^T \underline{\mu}_k\Big)^2, \tag{12}$$

where $T\underline{\mu}_k^3 = \sum_{\mathbf{i}\in I} T_{\mathbf{i}}(\underline{\mu}_k)_{\mathbf{i}}^3$ for $\mathbf{i} \in \mathcal{I} = \{(i_1,i_2,i_3)|i_1,i_2,i_3 \in \{1,\dots,D\}\}$; and $i_1,i_2,i_3 \in \{1,2,3\}$. In fact, [19] defines $\mathcal{I}$ as an index representation which is a set of unique entries of the tensor (due to symmetry). For each $\mathbf{i} \in \mathcal{I}$, a corresponding monomial is defined as $\mathbf{c}$, which belongs to the following set:

[2] The i^{th} entry of a vector $\underline{\mu}_k$ is denoted by μ_{ki}, element (i, j) of a matrix M is denoted by m_{ij} , and element (i, j, l) of a third-order tensor $\mathcal{T}$ is denoted by t_{ijl}.

[3] For instance, in the tensor case $t_{ijl} = t_{ilj} = t_{jil} = t_{jli} = t_{lij} = t_{lji}$ for all $i, j, l = 1,\dots,D$, [9].

[4] Note that the differentials have been validated with the symbolic toolbox in *MATLAB*.

$$C = \Big\{(c_1, c_2, \ldots, c_D) | c_1, \ldots, c_D \in \{0, \ldots, 3\}\Big\},$$

where we assume that $c_1 + \ldots + c_D = 3$. It is shown in [19] that the multiplicity of the entry corresponding to a monomial representation $\mathbf{c} \in C$ is:

$$\sigma_{\mathbf{c}} = \binom{3}{c_1, c_2, \ldots, c_D} = \frac{3!}{c_1! c_2! \ldots, c_D!}.$$

Therefore, $\|\mathcal{T}\|_F^2 = \sum_{\mathbf{i} \in \mathcal{R}} t_{\mathbf{i}} = \sum_{\mathbf{i} \in I} \sigma_{\mathbf{i}} t_{\mathbf{i}} = \sum_{\mathbf{c} \in C} \sigma_{\mathbf{c}} t_{\mathbf{c}}$ and $(\underline{\mu}_k^3)_{\mathbf{i}} = \underline{\mu}_{i_1 k} \underline{\mu}_{i_2 k} \underline{\mu}_{i_3 k} = (\underline{\mu}_k^3)_{\mathbf{c}} = \underline{\mu}_{1k}^{c_1} \underline{\mu}_{2k}^{c_2} \ldots \underline{\mu}_{Dk}^{c_D}$. As can be seen, this representation gives a more compact formulation compared to component-wise differentiation. Similar expressions can also be derived for the second derivative of the cost function. In particular, second derivatives are useful in performing Newton-type updates with quadratic convergences. However, this involves inverting a matrix making each step computationally expensive. Though we have not used the second derivatives in our empirical work here, in the Supplementary Material we give the required expressions for completeness. Algorithm 1 presents the pseudo-code for AGD-MoM . We provide a complete implementation of our code in *MATLAB*, and a Tensorflow implementation which uses automatic gradient computation. Experiments confirmed that the differences between the two implementations is within negligible numerical limits.

Before turning to the application of AGD-MoM and presenting our results, we discuss some points that need attention when specifying an optimisation model. First, in terms of convergence, our algorithm will eventually converge to a local optimum of the objective function. As with any gradient descent approach, the speed of convergence will depend on the choice of the learning rate parameter. A learning rate which is too small leads to a slow convergence, whereas a large learning rate causes the loss function to oscillate around the minimum or even to diverge. Various techniques are known to speed up the convergence. These include the use of a Momentum term [20], Nesterov accelerated gradient [21], Adaptive moment estimation (Adam) [22], etc. In our implementation we found faster convergences using a Momentum term and adaptive learning rate.[5]

Second, the computational complexity of AGD-MoM is $O\Big(KDlog(1/\epsilon)\Big)$ for accuracy ϵ whereas the total running time of the power iteration method is $O\Big(K^{5+\delta}\Big(log(K) + loglog(1/\epsilon)\Big)\Big)$ for at most $K^{1+\delta}$ random starts to find each eigenvectors, $O\Big(log(K) + loglog(1/\epsilon)\Big)$ iteration per start and each iteration needs $O(K^3)$ operations. In terms of computation complexity it seems that the AGD is more efficient than the power iteration method. However, as will be seen in Section 5, the power iteration method converges faster than AGD.

[5]Momentum helps to accelerate GD in the relevant direction and dampens oscillations [20]. Basically a fraction γ of the update vector of the previous step is added to the current update vector. On the other hand, Adam adapts the learning rate for each parameter individually, performing larger updates for infrequent and smaller updates for frequent parameters [23]. Essentially, the learning rate α and β are tuned according to the scale of the smoothness of the gradient function. More details can be found in [22].

Algorithm 1 AGD-MoM for learning GMM parameters:

Input:
Third-order moment as 3-mode tensor $\mathcal{T} \in \mathbb{R}^{D \times D \times D}$,
Second order moment as matrix $M \in \mathbb{R}^{D \times D}$.
Initialise:
Choose an initial vectors of parameters $\underline{\mu}_1, \ldots, \underline{\mu}_K$, $\underline{w}$ and learning rates α, β, and penalty parameter λ.
for $n = 1, 2, ...,$epochs **do**
  for $k = 1, 2, ..., K$ **do**
    Compute $\nabla f(\underline{\mu}_k^n | {w_k}^n)$, $\nabla g(\underline{\mu}_k^n | {w_k}^n)$, $\nabla f({w_k}^n | \underline{\mu}_k^n)$ and $\nabla g({w_k}^n | \underline{\mu}_k^n)$ from equations (5) to (8);
    Update $\hat{\underline{\mu}}_k^{n+1} \leftarrow \hat{\underline{\mu}}_k^n - \alpha_n(\nabla f(\underline{\mu}_k^n | w_k^n) + \lambda \nabla g(\underline{\mu}_k^n | w_k^n))$;
    Update $\hat{w}_k^{n+1} \leftarrow \hat{w}_k^n - \beta_n(\nabla f(w_k^n | \underline{\mu}_k^n) + \lambda \nabla g(w_k^n | \underline{\mu}_k^n))$;
  end for
end for
Output:
Return $\hat{\underline{\mu}}_1, \ldots, \hat{\underline{\mu}}_K$ and $\hat{\mathbf{w}}$.

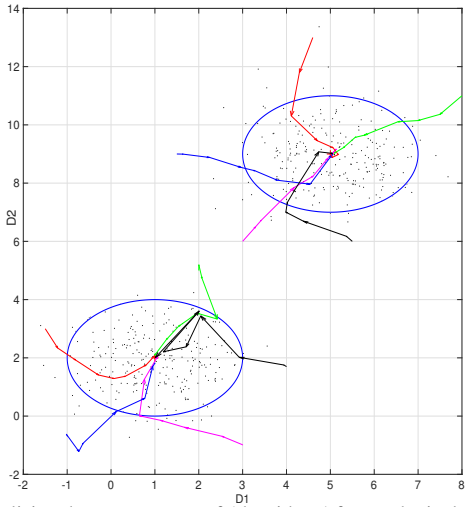


Fig. 1: Visualising the convergence of Algorithm 1 for synthetic data: spherical Gaussian mixture ($K = 2$, $D = 2$). Dots in different colours show different starting points for the algorithm. Only 5 paths are shown for clarity.

To illustrate AGD-MoM in operation, Figure 1 shows 5 different pairs of initial points and their sample paths as the algorithm progresses. Specifically, a simple two-dimensional two-components GMM ($K = 2$ and $D = 2$) is illustrated. Given two pairs of randomly selected points, gradient descent results in estimates close to the true (known) mean for both clusters and eventually converges to a local optimum. As far as convergence is concerned, AGD-MoM eventually converges to the correct solution in this case, regardless of the initial starting points; full analysis of convergence is left to Section 5.

4. Application to streaming data

In this Section we show how our algorithm may be extended to cater for streaming data in which the cluster centres are evolving over time. In streaming scenarios, it is typical that a significant amount of data needs to be quickly processed in real time, while historical data is typically jettisoned and no longer available. There exist few algorithms for performing online clustering via GMM; see, for example [24, 25, 26, 27] and

references therein. The majority of these algorithms are based on a split and merge criterion to add or remove clusters, followed by a rearrangement with an EM approach (see [24]). The IGMM algorithm proposed in [24] has been further extended to allow for evolving data either by dropping old data entirely [28] or by use of an exponential filter (i.e. a forgetting factor) as in [29]. In [26] it is argued that merge and split approaches are either too slow for online learning [25] or do not guarantee the accuracy of the resulting model. So later, a modification of the tensor decomposition algorithm for such online setting has been explored in [30], for community detection and topic modelling but not for GMM. Their study shows that although a moment-based formulation may seem computationally expensive and complicated at first sight, implementing implicit *tensor* operations leads to significant speed-ups and guarantees a learning process as opposed to several heuristic approaches [30]. The above would suggest that while split and merge may be applicable to AGD-MoM, the key advantage of our approach is that tensor updates based on newly arrived data may be computationally efficient and lead to improved parameter estimates.

Assume that the incoming data points arrive in sequential batches (or chunks [31]), $S_1, S_2, \ldots, S_m$, and that the next data batch to arrive is S_{m+1}. Upon the arrival S_{m+1}, the empirical moments must be updated. This involves updating C_x, and then $\mathcal{T}, M$; essentially these operations are a weighted combination of the old and new variables (see [32]); details of the *update* function are left to the Supplementary Material. The weighting uses a forgetting factor $\gamma \in [0, 1]$ which allows the updates to act as an exponential filter, thus allowing the evolving cluster centres to be tracked (as is used in IGMM [29]). Given updated moments AGD-MoM may be applied with the previous parameters used as initial estimates, denoted as AGD-MoM $(\mathcal{T}^m, M^m \mid \{\hat{\underline{\boldsymbol{\mu}}}^{m-1}_{1:K}, \hat{\underline{\boldsymbol{w}}}^{m-1}\})$. Therefore, the number of iterations taken by gradient descents to converge can be drastically reduced. Algorithm 2 gives a pseudo-code for implementing the Incremental-AGD-MoM (I-AGD-SGD) algorithm.

Algorithm 2 Incremental AGD-MoM for online learning:

1: **Input:**
   An initial data batch, S_1, and exponential weighting. γ
2: **Initialise:**
   Choose initial parameters $\{\underline{\boldsymbol{\mu}}^0_{1:K}, \underline{\boldsymbol{w}}^0\}$, α, β, λ.
3: **for** $m = 2, \ldots$ **do**
4: **while** awaiting batch S_m **do**
5: $\{\hat{\underline{\boldsymbol{\mu}}}^{m-1}_{1:K}, \hat{\underline{\boldsymbol{w}}}^{m-1}\} \leftarrow$ AGD-MoM $(\mathcal{T}^{m-1}, M^{m-1} \mid \{\hat{\underline{\boldsymbol{\mu}}}^{m-2}_{1:K}, \hat{\underline{\boldsymbol{w}}}^{m-2}\})$.
6: **end while**
7: **Output:**
   Return $\{\hat{\underline{\boldsymbol{\mu}}}^{m-1}_{1:K}, \hat{\underline{\boldsymbol{w}}}^{m-1}\}$
8: $\{\mathcal{T}^m, M^m\} \leftarrow update(\mathcal{T}^{m-1}, M^{m-1} \mid C^m, \gamma)$
9: **end for**

5. Empirical results

In this section, we present the empirical performance of our algorithm and compare it to the state-of-the-art algorithms such as the EM algorithm, the Variational Bayes (VB) method, and Tensor decomposition using the power iteration method (TPM).[6] We perform experiments for both synthetic and real data. In the first set of experiments, we begin with a simple 2-D case examining the convergence and results in detail. The analysis is then extended to higher dimensions. The second experiment is conducted on two real datasets from the UCI repository[7]: the Iris and Wine datasets. Last but not least, the incremental AGD-MoM is tested on a large set of synthetic streaming data.

Our algorithm settings are as follows for all experiments. The learning rates of AGD, α and β, are set to lower values between 10^{-4} to 10^{-5} in order to ensure that the algorithm does not miss a local optimum while keeping the computational cost as low as possible. The maximum number of epochs is set to 1000. We found $\lambda = 10$ as a suitable choice for the penalty parameter. Each experiment is run 100 time with different random starting points allowing us to estimate parameter statistics.[8] The same starting points are used for all algorithms.[9] The estimation accuracy is measured using the Mean Absolute Percentage Error (MAPE). The MAPE is considered because of its scale independence and also its ease of interpretation.[10]

5.1. Synthetic data experiments

In the following experiment a synthetic dataset is generated to examine in detail the convergence rate and results of Algorithm 1. The purpose of this experiment is to determine which of the three aforementioned optimisation techniques, standard gradient descent, Momentum and Adam (Section 3), speeds up the converges rate. To better visualise our results, a set of two-dimensional data are generated from a two-component GMM (N=500). Figure 2a shows the average convergence error rate obtained by standard gradient descent, Momentum and Adam. As can be seen, Momentum converges slightly faster than Adam, both having essentially converged after approximately 100 epochs, while GD does not converge until 400 epochs and then requires an asymptotic number of iterations to reach final convergence.[11]

Our empirical results further show that GD with a fixed learning rate achieves the optimum minimum, but requires significantly more epochs than Adam and Momentum to do so. In order to provide a closer picture of how these algorithms perform regarding 100 different starting points, box-plots of $\mathbf{w}$, $\underline{\boldsymbol{\mu}}_1$ and $\underline{\boldsymbol{\mu}}_2$ are shown in Figures 2b to 2c. Figure 2b shows the Momentum estimates are slightly more concentrated than Adam, while, the GD box-plot implies a far wider range for $\mathbf{w} = 0.5$. On the

[6] The implementation may be found in [4].

[7] http://archive.ics.uci.edu/ml

[8] Specifically, the sensitivity of the solutions to the initial starting point.

[9] We implemented all algorithms for our synthetic data experiments using *MATLAB*.

[10] MAPE is calculated as the average of the unsigned differences between an estimated value and the known true value divided by the known true value. In other words, we use the formula $\frac{1}{n}\sum_{i=1}^{n} \|\varrho_i\|_1$ where $\varrho_i = (\frac{\mu_{i1}-\hat{\mu}_{i1}}{\mu_{i1}}, \ldots, \frac{\mu_{in}-\hat{\mu}_{id}}{\mu_{id}})$ and $\| \cdot \|_1$ is ℓ_1-norm. is used as the main measure of estimation accuracy.

[11] Most machine learning algorithms exhibit the behaviour in Figure 2, quickly obtaining a '*reasonable*' solution and then requiring many more iterations to refine that solution. When these algorithms are being run on the cloud, time equates directly to monetary cost, and so many applications seek algorithms that provide the fastest reasonable solution.

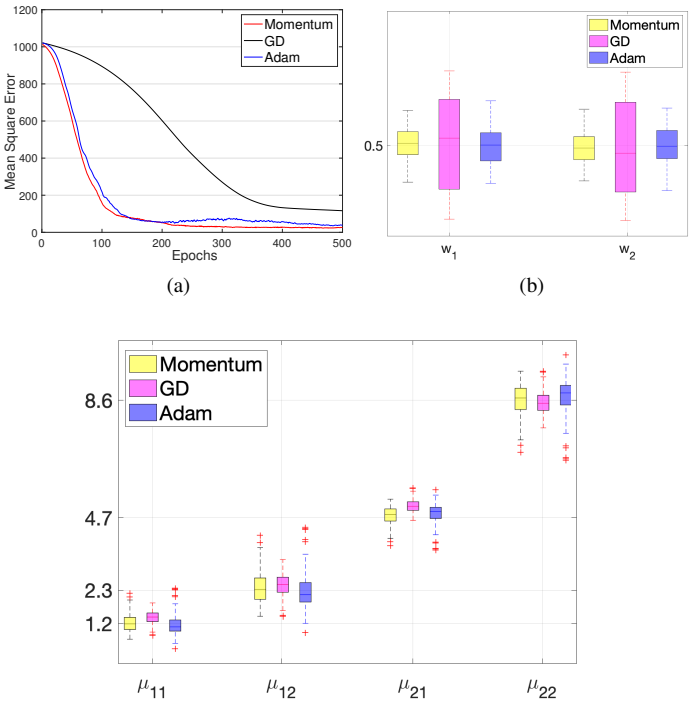


Fig. 2: Convergence of algorithm (1) in a synthetic setting over 100 runs: (a) Summary of average convergence error rate GD, momentum and Adam. Boxplots for (b) w_1 and w_2, (c) $\underline{\boldsymbol{\mu}}_1$ and $\underline{\boldsymbol{\mu}}_2$. The known true values for each component are indicated in y-axis.

other hand, for both mean components, $\underline{\boldsymbol{\mu}}_1$ and $\underline{\boldsymbol{\mu}}_2$, GD results in estimates with a lower variance but a distinct bias. As can be seen in Figure 2c, both Adam and Momentum span much the same range of values and their medians are roughly around the true known values (y-axis). These experiments (together with other cases presented in the Supplementary Material) indicate that overall Momentum and Adam provide the fastest convergence with a lower bias. We can conclude that the learning rate does not need to be tuned according to the scale of the smoothness of the gradient function but likely achieves better results by dampening the oscillations via Momentum.

In our next experiment, we focus on higher order GMMs to examine how the algorithm behaves in the presence of multiple local minima. Figure 3 shows the MAPE achieved by AGD-MoM, TPM, VB and the EM algorithm for GMM with $K = 5$, and 10 components. Figures 3a and 3b, for $K = 5$, show that AGD performs significantly better than the alternatives in all cases. Note also, that Figures 3c and 3d show that TPM performs well when K and D are large.

5.2. *Benchmark dataset experiments*

Iris datasets. ($N = 150, K = 3, D = 4$) One of the most well-known datasets in classification is the Iris dataset. The dataset contains 3 types of Iris plants, with 50 instances of each. Four features were measured from each sample, sepal length, sepal width, petal length, and petal width. The MAPE's achieved by each of the competing algorithms are shown in Figures 4a and 4b. Note that for each parameter the lowest MAPE is always provided by one of the moment based methods (i.e. TPM or AGD). AGD does not provide a good estimate for w_1 (Figure 4b) but outperforms the other techniques otherwise.

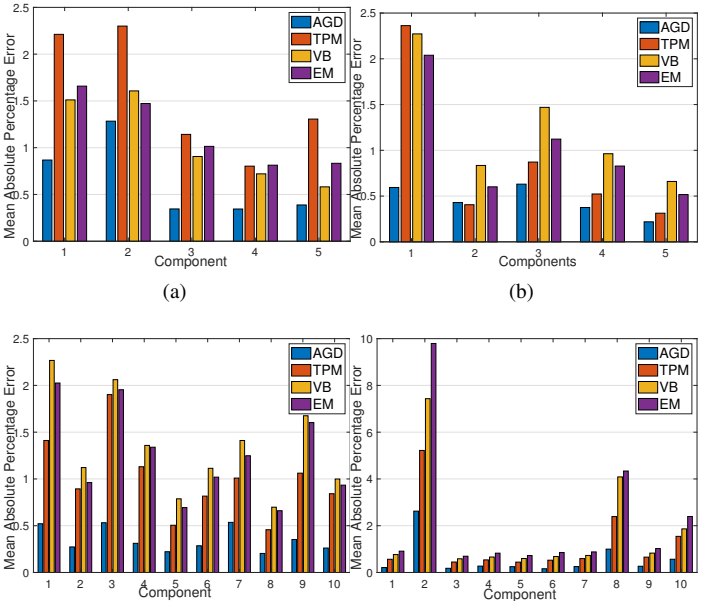


Fig. 3: The MAPE achieved by the AGD-MoM, TPM, VB and EM algorithms, in a GMM with (a) $K = 5, D = 5, N = 1000$ (b) $K = 5, D = 20, N = 1000$ (c) $K = 10, D = 20, N = 5000$ (d) $K = 10, D = 50, N = 5000$.

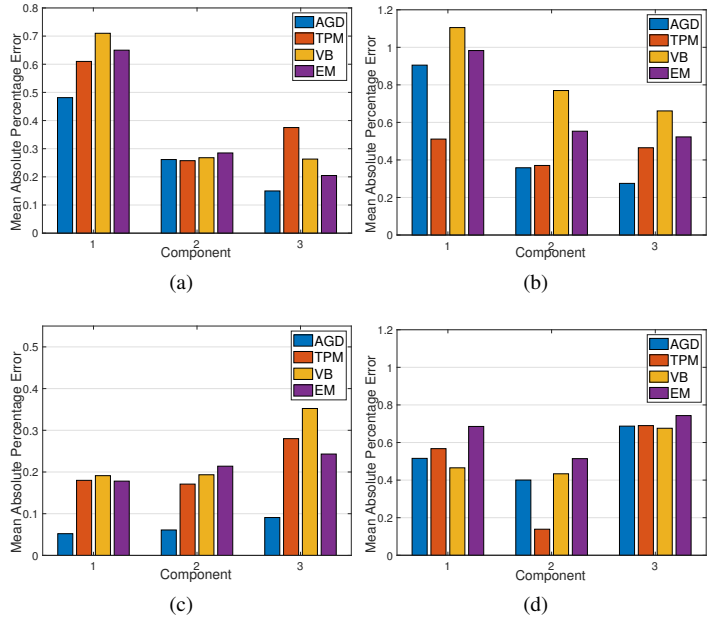


Fig. 4: The MAPE achieved by the AGD, TPM, VB and EM algorithms (a) $\underline{\boldsymbol{\mu}}$ (Iris data), (b) $\mathbf{w}$ (Iris data), (c) $\underline{\boldsymbol{\mu}}$ (Wine data) and (d) $\mathbf{w}$ (Wine data). The results are averaged over 100 runs.

Wine data set. ($N = 178, K = 3, D = 13$) The Wine data set consists of wine samples from 3 different culitvars (known classes) grown in the same region in Italy, using 13 features (derived from chemical tests of constituent elements). Figures 4c and 4d show the MAPE's achieved by the competing algorithms. As can be seen, AGD results in a better mean estimate in all cases, however, TPM gives a better result for one of the weight estimates. In addition to MAPE, we have also reported the trace of the error covariance matrix [33], the overall uncertainty reported by the estimators, for the estimates obtained from AGD-MoM, TPM, EM and VB in Table 1. It is formulated as the empirical sum of square errors over all dimensions,

Table 1: The sum of the trace of the error covariance matrix

	Iris		Wine	
Methods	μ	w	μ	w
AGD	**0.36**	0.55	**0.13**	0.73
TPM	1.06	**0.53**	0.59	**0.71**
VB	1.34	2.07	1.82	1.04
EM	1.24	1.30	1.46	1.53

$\mathbf{tr}\Big(\mathrm{E}[(\underline{\mu}_k - \hat{\underline{\mu}}_k^n)(\underline{\mu}_k - \hat{\underline{\mu}}_k^n)^T]\Big)$ where the expectation is over $n = 100$ runs, where $\hat{\underline{\mu}}_k^n$ is the estimated mean obtained at iteration n^{th}. As can be seen, the AGD-MoM results in significantly better estimates for the cluster centres. For the cluster weights the AGD-MoM estimates are comparable to those from TPM and significantly better than VB and EM. Last but not least we focus on empirical testing of the convergence rate for AGD and TPM. Figures 5a, 5b and 5c show the convergence error rate of loss function obtained by moment-based methods for simulated, Iris and Wine datasets, respectively. As can be seen TPM converges to its solution faster than AGD does. Note however, that the two solutions differ in their accuracies as discussed above.

5.3. Streaming experiments

In our final experiment, we examine Incremental AGD-MoM for online learning. The scenario we specifically examine is that in which the cluster centres are evolving over time. This is a common occurrence in real-world data and occurs when *causal* but *unmeasured* variables also evolve (ex: the *weather* influence on wine quality year on year or trending topics on the internet etc.). The baseline algorithm for comparison is IGMM from [29] which employs a forgetting factor, γ. Note that γ for IGMM and I-AGD-SGD play the same role and are set equal to 0.7 in all experiments. Appropriate selection of forgetting factors is discussed in many text books (ex:[34]). Here the high forgetting rate is a consequence of the large batch size.

The basic goal of this experiment is to iteratively move cluster centres to gradually reduce the distances between cluster centres and their associated data points (Figure 6a). In order to achieve that, we generated two sets of 2-Dimensional mean vectors from two parametrised curves in $\mathbb{R}^2$ given by $\Big(cos(\frac{t\pi}{b_1} + a_1)t, cos(\frac{t\pi}{b_2} + a_2)t\Big)$ and $\Big(cos(\frac{t\pi}{d_1} + c_1)(t - 100), cos(\frac{t\pi}{d_2} + c_2)(t - 100)\Big)$, where $t = (1, \ldots, 100)$, $a = (a_1, a_2), b = (b_1, b_2), c = (c_1, c_2)$ and $d = (d_1, d_2)$ are chosen from $[50, 125]$ and $[15, 25]$. This provides a set of cluster centres which *twist like a rope segment* in time providing a nice challenge to the algorithm. We have generated 500,000 points form these 2-D mean vectors; 100 batches with size 5000. Figure 6a shows a heat map of the simulated data points drifting slowly when the new batch becomes available.

Figure 6b displays cluster centres obtained using the I-AGD-MoM algorithm given 500 epochs. The result is that the algorithm tracks the centres quite tightly. Next we drop the number of epochs available to 100 mimicking the event when less time is available between batches. As Figure 6c demonstrates, initially the algorithm has trouble tracking the evolution but after a burn-in period tracks onto the evolution. It then looses this track again for two subsequent periods examined below. Figures 7a and 7b show the MAPE as a function of time for I-AGD-MoM and IGMM. Note that there is an initial burn-in period before both techniques start to track the evolving centres. We can also observe that for $\underline{\mu}_2$ (7b) there are two periods (times 10, 60) where the MAPE, for both I-AGD-MoM and IGMM, jump due to a change in direction of the centre evolution (as seen by the gradient plotted alongside the MAPE in Figure 7b). However, as can be seen I-AGD-MoM deviates less from the true track than IGMM.

6. Conclusion

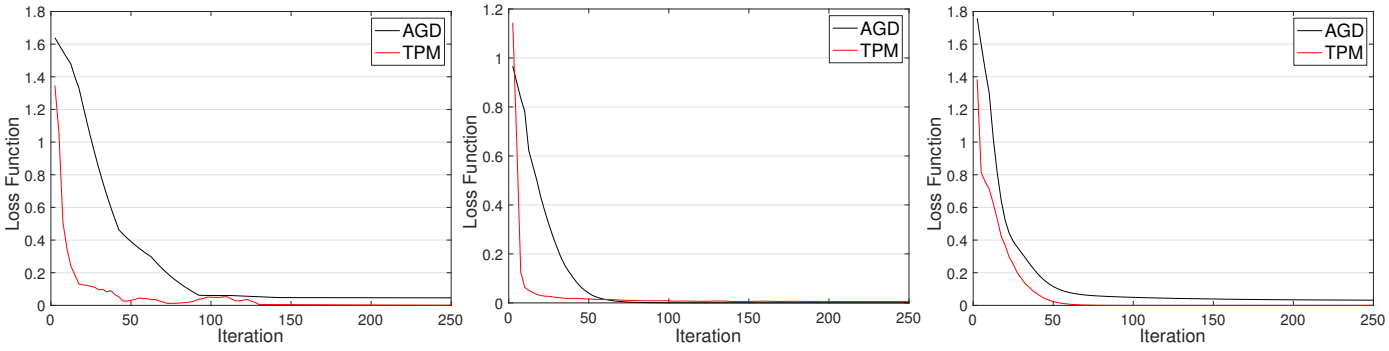
In this paper, we have presented an alternative approach to estimate parameters of a spherical Gaussian mixture model. The approach we present is based on matching second and third order moments which we solve by an alternating gradient descent approach. The method is shown to converge reliably on synthetic data where the assumptions about spherical covariances are met. We also demonstrate the performance of the algorithm on benchmark datasets from classification problems in which the ground truth of cluster centres is known from the class labels. Our experiments show that AGD-MoM leads to better estimates for most parameters but may take longer than TPM to converge suggesting that the best choice of algorithm depends on the problem at hand. Furthermore, we show that the approach is suitable for streaming data in an incremental mode, using previously estimated parameters to deal with newly arriving data. In addition, the computational complexity our algorithm is essentially independent of the size of the data set unlike the classical EM algorithm. Our current work is focused on extending this framework to relax the assumptions of isotropic covariances and the applicability of the algorithm to large-scale clustering problems. In addition, use of (one of the many) adaptive forgetting factor algorithms may prove useful.

Acknowledgements

This work was supported by an EPSRC grant EP/N014189/1.

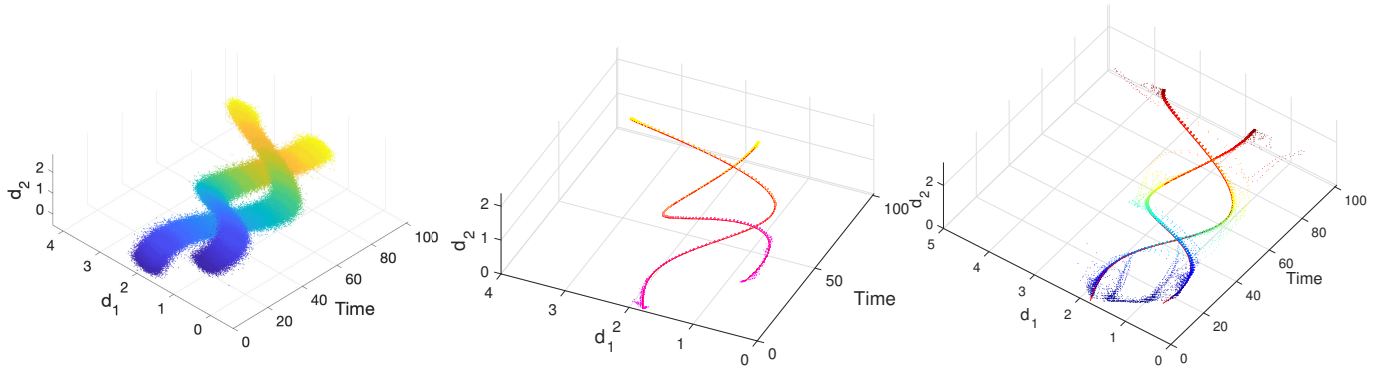
References

[1] C. P. Robert, Monte carlo methods, Wiley Online Library, 2004.
[2] C. F. J. Wu, On the Convergence Properties of the EM Algorithm, The Annals of Statistics 11 (1) (1983) 95–103.
[3] R. A. Redner, H. F. Walker, Mixture densities, maximum likelihood and the EM algorithm, SIAM review 26 (2) (1984) 195–239.
[4] A. Ray, J. Neeman, S. Sanghavi, S. Shakkottai, The Search Problem in Mixture Models, JMLR. 18 (2016) 206:1–206:61.
[5] Z. Chen, S. Haykin, J. J. Eggermont, S. Becker, Correlative Learning: A Basis for Brain and Adaptive Systems, Wiley-Interscience, 2007.
[6] L. Le Cam, The central limit theorem around 1935, Statistical science (1986) 78–91.
[7] B. G. Lindsay, P. Basak, Multivariate normal mixtures: a fast consistent method of moments, JASA 88 (422) (1993) 468–476.
[8] D. Hsu, S. M. Kakade, Learning mixtures of spherical gaussians: moment methods and spectral decompositions, in: Proceedings of the 4th conference on Innovations in Theoretical Computer Science, ACM, 11–20, 2013.
[9] T. G. Kolda, B. W. Bader, Tensor decompositions and applications, SIAM review 51 (3) (2009) 455–500.
[10] M. Anandkumar, A., Ge, R., Hsu, D. J., Kakade, S. M., & Telgarsky, Tensor Decompositions for Learning Latent Variable Models, JMLR 15 (2014) 2773–2832, ISSN 1532-4435.
[11] K. Chaudhuri, S. Rao, Learning Mixtures of Product Distributions Using Correlations and Independence., in: COLT: 21st Annual Conference on Learning Theory, vol. 4, 9–20, 2008.



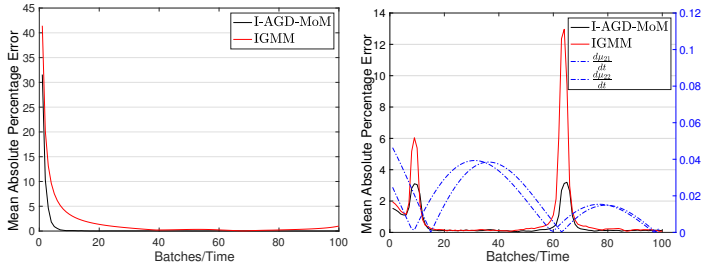
(a) (b) (c)

Fig. 5: Comparisons of convergence rate between TPM and AGD for (a) Simulated datasets (b) Iris datasets (c) Wine datasets.



(a) (b) (c)

Fig. 6: Convergence of I-AGD-MoM in a streaming setting over 100 runs: (a) synthetic data ($K = 2, D = 2$) (the third dimension is time), with the cluster centres slowly drifting (see text); (b) trajectories of estimated cluster centres (nepochs=500), red line shows the true mean and dots around it show the AGD-MoM estimated mean; and (c) trajectories of estimated means (nepochs=100), red line shows the true mean and (100) dots around it show the I-AGD-MoM estimated.



(a) (b)

Fig. 7: The MAPE comparison between AGD and IGMM in a streaming setting over 100 runs (a) $\underline{\mu}_1$, (b) $\underline{\mu}_2$. Noted that dashed blue line shows the gradient alongside the MAPE.

[12] S. Dasgupta, L. Schulman, A probabilistic analysis of EM for mixtures of separated, spherical Gaussians, JMLR 8 (Feb) (2007) 203–226.

[13] L. De Lathauwer, B. De Moor, J. Vandewalle, On the best rank-1 and rank-($r_1, r_2, \ldots, r_n$) approximation of higher-order tensors, SIAM journal on Matrix Analysis and Applications 21 (4) (2000) 1324–1342.

[14] A. Shashua, T. Hazan, Non-negative tensor factorization with applications to statistics and computer vision, in: Proceedings of the 22nd international conference on Machine learning, ACM, 792–799, 2005.

[15] J. Liu, P. Wonka, J. Ye, Sparse non-negative tensor factorization using columnwise coordinate descent, Pattern Recognition 45 (2012) 649–656.

[16] Y. Xu, W. Yin, A block coordinate descent method for regularized multi-convex optimization with applications to nonnegative tensor factorization and completion, SIAM Journal on imaging sciences 6 (2013) 1758–1789.

[17] J. Nocedal, S. Wright, Numerical optimization, Springer Science & Business Media, 2006.

[18] V. Kuleshov, A. Chaganty, P. Liang, Tensor factorization via matrix factorization, in: Artificial Intelligence and Statistics, 507–516, 2015.

[19] T. G. Kolda, Numerical optimization for symmetric tensor decomposition, Mathematical Programming 151 (1) (2015) 225–248.

[20] N. Qian, On the momentum term in gradient descent learning algorithms, Neural networks 12 (1) (1999) 145–151.

[21] Y. Nesterov, A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$, in: Doklady AN USSR, vol. 269, 543–547, 1983.

[22] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980.

[23] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, JMLR 12 (Jul) (2011) 2121–2159.

[24] M. Song, H. Wang, Highly efficient incremental estimation of Gaussian mixture models for online data stream clustering, in: Intelligent Computing: Theory and Applications III, vol. 5803, 174–184, 2005.

[25] P. M. Hall, Y. Hicks, T. Robinson, A method to add Gaussian mixture models, University of Bath, Department of Computer Science, 2005.

[26] A. Declercq, J. H. Piater, Online Learning of Gaussian Mixture Models-a Two-Level Approach., in: VISAPP (1), 605–611, 2008.

[27] R. C. Pinto, P. M. Engel, A fast incremental Gaussian mixture model, PloS one 10 (10) (2015) e0139931.

[28] L. S. Oliveira, G. E. A. P. A. Batista, IGMM-CD: A Gaussian Mixture Classification Algorithm for Data Streams with Concept Drifts, in: 2015 Brazilian Conference on Intelligent Systems (BRACIS), 55–61, 2015.

[29] C. A. Juan M. Acevedo-Valle, Karla Trejo, Multivariate Regression with Incremental Learning of Gaussian Mixture Models, in: Recent Advances in Artificial Intelligence Research and Development, vol. 300, IOS Press, 196 – 205, 2017.

[30] F. Huang, U. Niranjan, M. U. Hakeem, A. Anandkumar, Online tensor methods for learning latent variable models, JMLR 16.

[31] J. Gama, Knowledge discovery from data streams, CRC Press, 2010.

[32] E. Schubert, M. Gertz, Numerically Stable Parallel Computation of (Co)Variance, in: SSDBM, 10:1–10:12, 2018.

[33] K. Dormann, B. Noack, U. Hanebeck, Optimally Distributed Kalman Filtering with Data-Driven Communication, Sensors 18 (2018) 1034.

[34] S. Haykin, Adaptive filter theory, Prentice Hall, Upper Saddle River, NJ, 4th edn., 2002.

Declaration of interest:

This work was supported by an EPSRC grant EP/N014189/1. It is also mentioned in the manuscript.

Your Sincerely,

Donya Rahamni
Email: d.rahmani@soton.ac.uk
Research Fellow in Statistical Machine Learning and Applied Topology
Building 54, Mathematical Science, Highfield campus, University of Southampton
Phone: 02380525156

Pattern Recognition Letters

Authorship Confirmation

Please save a copy of this file, complete and upload as the "Confirmation of Authorship" file.

As corresponding author I, Donya Rahmani, hereby confirm on behalf of all authors that:

1. This manuscript, or a large part of it, has not been published, was not, and is not being submitted to any other journal.
2. If presented at or submitted to or published at a conference(s), the conference(s) is (are) identified and substantial justification for re-publication is presented below. A copy of conference paper(s) is(are) uploaded with the manuscript.
3. If the manuscript appears as a preprint anywhere on the web, e.g. arXiv, etc., it is identified below. The preprint should include a statement that the paper is under consideration at Pattern Recognition Letters.
4. All text and graphics, except for those marked with sources, are original works of the authors, and all necessary permissions for publication were secured prior to submission of the manuscript.
5. All authors each made a significant contribution to the research reported and have read and approved the submitted manuscript.

Signature Donya Date 29/08/2018

List any pre-prints:

Relevant Conference publication(s) (submitted, accepted, or published):

Justification for re-publication:

7. Supplementary material

7.1. Second order differentiability and Hessian

The Newton's method is yet another method for minimising the sum-of-squares objective function (2). More specifically, in this method, it is assumed that the objective function is approximately quadratic in the parameters near the optimal solution [35]. In doing so, we have also calculated the Hessian, $(\boldsymbol{H}(f+\lambda g))$, the second derivative with respect to each element $\mu_{\kappa d}$ while $\kappa = 1,\ldots,K; d = 1,\ldots,D$ as follows (for simplicity we considered $\lambda = 1$):

$$\begin{aligned}
\frac{\partial^2 f}{\partial\mu_{\kappa d}\partial\mu_{\kappa d}} &= w_\kappa \sum_{i,j=1}^{D}\delta_{ij}\mu_{\kappa i}\mu_{\kappa j}\sum_{k=1}^{K} w_k\mu_{ki}\mu_{kj} + 12w_\kappa\sum_{i=1}^{D}\mu_{\kappa i}\\
&\quad(\sum_{k=1}^{K} w_k\mu_{kd}^2\mu_{ki} - T_{ddi}) + 4w_\kappa(\sum_{k=1}^{K} w_k\mu_{kd}^2 - M_{dd})\\
&\quad + 4w_\kappa^2\sum_{i=1}^{D}\mu_{\kappa i}^2 \qquad \delta_{ij} = \begin{cases} 6, & i=j\neq d,\\ 12, & i\neq j\neq d,\\ 18, & i=j=d,\\ 24, & i=d; j\neq d.\end{cases}\\
\frac{\partial^2 f}{\partial\mu_{\kappa d}\partial\mu_{\kappa' d}} &= w_\kappa w_{\kappa'}\sum_{i,j=1}^{D}\delta_{ij}\mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa' i}\mu_{\kappa j} + 4w_\kappa w'_\kappa\sum_{i=1,i\neq d}^{D}\mu_{\kappa i}\mu_{\kappa' i}\\
&\quad + 8w_\kappa w_{\kappa'}\mu_{\kappa d}\mu_{\kappa' d}\\
\frac{\partial^2 f}{\partial\mu_{\kappa d}\partial\mu_{\kappa d'}} &= 12w_\kappa^2\mu_{\kappa d'}\mu_{\kappa d}\sum_{j=1}^{D}\mu_{\kappa j}^2 + 4w_\kappa^2\mu_{\kappa d}\mu_{\kappa d'}\\
&\quad + 12w_\kappa\sum_{j=1}^{D}\mu_{\kappa j}(\sum_{k=1}^{K} w_k\mu_{kd'}\mu_{kd}\mu_{kj} - T_{jdd'})\\
&\quad + 4w_\kappa(\sum_{k=1}^{K} w_k\mu_{kd}\mu_{kd'} - M_{dd'})\\
\frac{\partial^2 f}{\partial\mu_{\kappa d}\partial\mu_{\kappa' d'}} &= 12w_\kappa w_{\kappa'}\mu_{\kappa d'}\mu_{\kappa' d}\sum_{i=1,i\neq d,d\neq d'}^{D}\mu_{\kappa i}\mu_{\kappa' i}\\
&\quad + 4w_\kappa\mu_{\kappa d'}w_{\kappa'}\mu_{\kappa' d}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial^2 f}{\partial w_\kappa\partial w_{\kappa'}} &= 2\sum_{i=1}^{D}\mu_{\kappa i}^3\mu_{\kappa' i}^3 + 6\sum_{i,j=1,j\neq i}^{D}\mu_{\kappa i}^2\mu_{\kappa j}\mu_{\kappa' i}^2\mu_{\kappa' j}\\
&\quad + 12\sum_{i,j,l=1,l\neq j\neq i}^{D}\mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa l}\mu_{\kappa' i}\mu_{\kappa' j}\mu_{\kappa' l}\\
&\quad + 4\sum_{i,j=1,j\neq i}^{D}\mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa' i}\mu_{\kappa' j} + 2\sum_{i=1}^{D}\mu_{\kappa i}^2\mu_{\kappa' i}^2
\end{aligned}$$

This can also be organised in a more compact form as:

$$\begin{aligned}
\frac{\partial f}{\partial w_\kappa w_{\kappa'}} &= 2(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})^3 + 2(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})^2\\
\frac{\partial f}{\partial\underline{\boldsymbol{\mu}}_\kappa\underline{\boldsymbol{\mu}}_{\kappa'}} &= 18w_\kappa w_{\kappa'}(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})^2 + 8w_\kappa w_{\kappa'}(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})
\end{aligned}$$

$$\begin{aligned}
\frac{\partial f}{\partial\underline{\boldsymbol{\mu}}_\kappa\underline{\boldsymbol{\mu}}_\kappa} &= -12w_\kappa T\underline{\boldsymbol{\mu}}_\kappa + 12w_\kappa\sum_{k=1}^{K} w_k(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_k)\underline{\boldsymbol{\mu}}_k - 4w_\kappa M\\
&\quad + 4w_k\sum_{k=1}^{K} w_k\underline{\boldsymbol{\mu}}_k^2
\end{aligned}$$

Note that in the above formula $(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})^2$ equals $\sum_{i=1}^{D}\mu_{\kappa i}^2\mu_{\kappa' i}^2 + \sum_{i,j=1,j\neq i}^{D}\mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa' i}\mu_{\kappa' j}$ which can be written based on index representation and its monomial as:

$$\sum_{i_1}\sum_{i_2}\sum_{i_3}\delta_{\mathbf{c}}(\mu_{\kappa 1}\mu_{\kappa' 1})^{c_1}\ldots(\mu_{\kappa D}\mu_{\kappa' D})^{c_D}$$

for $\mathbf{c} = (c_1,\ldots,c_D)\in\{0,1,2\}$.

Provided that $f+\lambda g$ is a twice-differentiable function then the Newton's method with a quadratic rate of convergence towards a local minimum than gradient descent can be used instead of gradient descent. Thus, our method can be easily modified for solving the problem (2) using the second order information which converges with much fewer iterations than gradient methods. To do so the following modification can be applied on steps 8 and 10 in Algorithm 1.
First we fix w_k and update $\underline{\boldsymbol{\mu}}_k$:

$$\underline{\boldsymbol{\mu}}_k^{n+1} = \underline{\boldsymbol{\mu}}_k^n + \alpha^n\Big(\nabla f(\underline{\boldsymbol{\mu}}_k^n|w_k) + \lambda\nabla g(\underline{\boldsymbol{\mu}}_k^n|w_k)\Big)\Big[\boldsymbol{H}\Big(f(\underline{\boldsymbol{\mu}}_k^n|w_k) + \lambda g(\underline{\boldsymbol{\mu}}_k^n|w_k)\Big)\Big]^{-1}$$

Fix $\underline{\boldsymbol{\mu}}_k$ and update w:

$$w_k^{n+1} = w_k^n + \beta^n\Big(\nabla f(w_k^n|\underline{\boldsymbol{\mu}}_k) + \lambda\nabla g(w_k^n|\underline{\boldsymbol{\mu}}_k)\Big)\Big[\boldsymbol{H}[f(w_k^n|\underline{\boldsymbol{\mu}}_k) + \lambda g(w_k^n|\underline{\boldsymbol{\mu}}_k)\Big)\Big]^{-1}$$

A refinement of the Newton approach is Levenberg-Marquardt method which accelerates the chance of local convergence and avoids divergence. Basically, adding up a scaled version of the diagonal elements of the Hessian matrix to itself, $[\mathbf{H} + \eta\mathrm{diag}(\mathbf{H})]$, results in a faster convergence than the Levenberg damping and Gauss-Newton method. The Levenberg-Marquardt algorithm adaptively varies the parameter updates between the gradient descent update and the Gauss-Newton update, where small values of the algorithmic parameter λ result in a Gauss-Newton update and large values of λ result in a gradient descent update. The parameter λ is initialised to be large so that first updates are small steps in the steepest descent direction. If any iteration happens to result in a worse approximation $F(\underline{\boldsymbol{\mu}}^{n+1},\underline{\mathbf{w}}^{n+1}) > F(\underline{\boldsymbol{\mu}}^n,\underline{\mathbf{w}}^n)$, then λ is increased. Otherwise, as the solution improves, λ is decreased.

7.2. Moment updates for online learning

In this Section we detail the *update* function used in Algorithm 2. The function combines stored variables from the previous data sets, $S_1, S_2,\ldots,S_m$, with the newly arrived data batch, $S_{m+1}, S_{m+2},\ldots$, in a weighted manor allowing for exponential smoothing of the updates (this is what allows the algorithm to track evolving centres). Moment and statistic updates using exponential smoothing are well known and may be found in [32]. The updates consist of 4 steps detailed below.

- 1. Updates for the mean: $\underline{\boldsymbol{\mu}}^{m+1} = \underline{\boldsymbol{\mu}}^{1:m} + \frac{n_{m+1}}{n_{1:m}+n_{m+1}}(\underline{\boldsymbol{\mu}}^{1:m} - \underline{\boldsymbol{\mu}}^{m+1})$.
- 2. Updates for the covariance matrix: Given an update of the mean we may now turn to updating the covariance matrix of

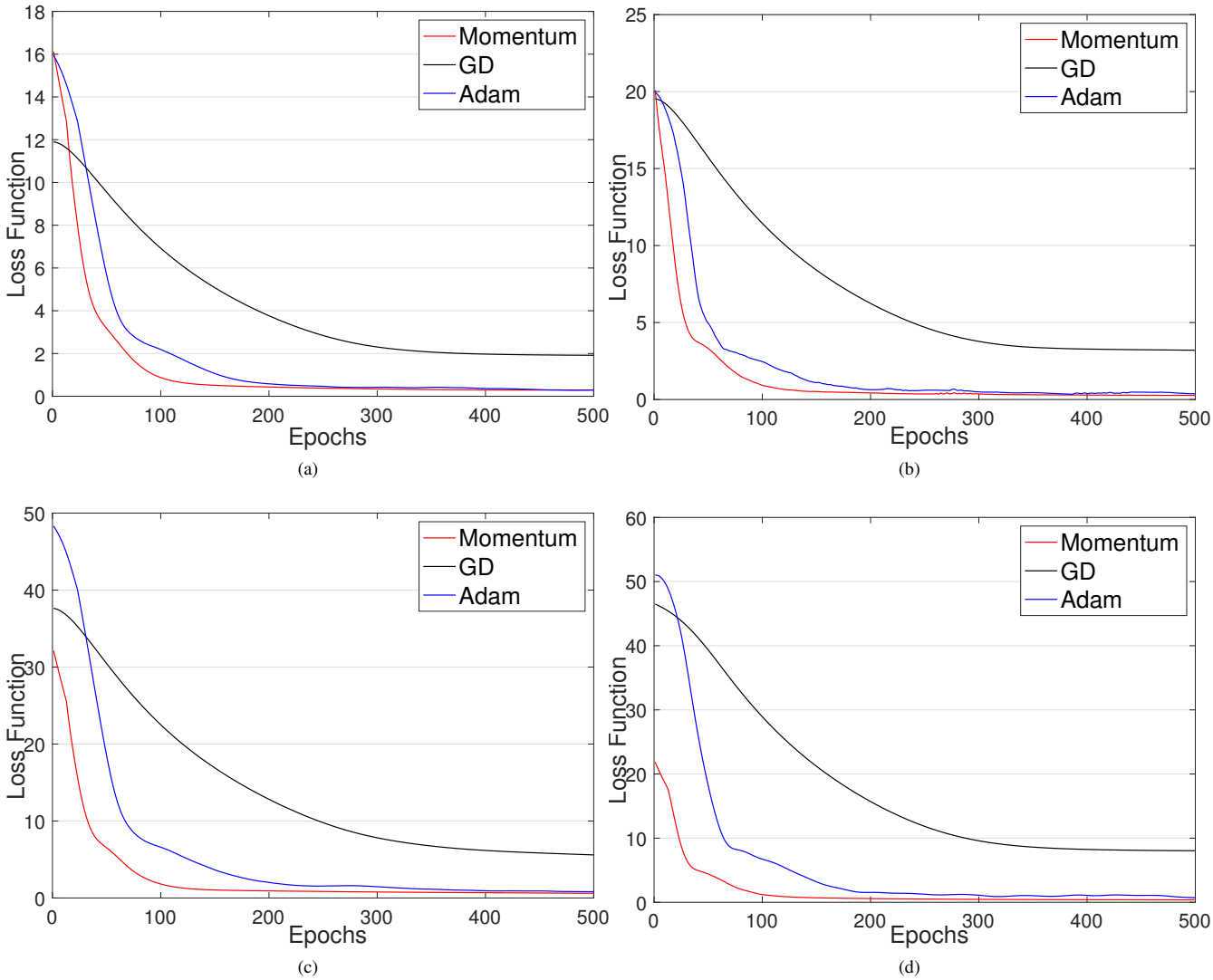


Fig. 8: Summary of average convergence error rate GD, momentum and Adam. (a) $K = 5$, $D = 5$, (b) $K = 5$, $D = 20$, (c) $K = 10$, $D = 20$ and (d) $K = 10$, $D = 50$.

the data, i.e. $C_{\underline{\mathbf{x}}}$ from Theorem 1 at time m denoted $C_{\underline{\mathbf{x}}}^{1:m}$. Note that $C_{\underline{\mathbf{x}}}^{m+1}$ is now the covariance matrix of a partitioned set $S_{1:m} \cup S_{m+1}$. In general, for two variables $\underline{\mathbf{x}}_2, \underline{\mathbf{x}}_2$ and two partitions of the data the cross-covariance of the union may be expressed as [32] (Eqn. 22) as:

$$C_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{m+1} = \frac{1}{n-1}\left(\mathcal{S}_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{1:m} + \mathcal{S}_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{m+1} + \frac{(\gamma_2\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_1}^{1:m} - \gamma_1\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_1}^{m+1})(\gamma_2\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_2}^{1:m} - \gamma_1\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_2}^{m+1})}{\gamma_1\gamma_2(\gamma_1+\gamma_2)}\right), \tag{13}$$

where $C_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{m+1}$ is the combined covariance of $\underline{\mathbf{x}}_1$ with $\underline{\mathbf{x}}_2$, $n_{1:m}$ is the number of samples in $S_1, \ldots, S_m$, $\mathcal{S}_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{1:m}$ is sum of the dot product of $\underline{\mathbf{x}}_1$ and $\underline{\mathbf{x}}_2$ in set $S_1, \ldots, S_m$, $\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_1}^{m+1}$ denotes the mean of $\underline{\mathbf{x}}_1$ in set S_{m+1}, and γ_m and γ_{m+1} are weights which can allow one to give a greater weight to one set over the other; this allows for exponential smoothing in an online setting (See [32] Section 4.2). Specifically we set $\gamma_1 = \gamma$ and $\gamma_2 = 1 - \gamma$ which implements exponential smoothing.

- 3. Updates for the eigenpairs: Recalculate the first eigenvalue and eigenvector, for $C_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{m+1}$.
- 4. higher order moment updates: Give updates for M and $\mathcal{T}$ with new data and new eigenvalue and eigenvector.

$$M^{m+1} = (1-\gamma)M^m + \gamma(x \otimes x) + \bar{\sigma}^{2^{m+1}} I,$$
$$\mathcal{T}^{m+1} = (1-\gamma)\mathcal{T}^m + \gamma(x \otimes x \otimes x).$$

The updated M and $\mathcal{T}$ are used as inputs for AGD-MoM algorithm and update parameter estimates for $\hat{\underline{\boldsymbol{\mu}}}_1, \ldots, \hat{\underline{\boldsymbol{\mu}}}_K$ and $\hat{\underline{\mathbf{w}}}$ for the next batch.

7. Supplementary material

7.1. Second order differentiability and Hessian

The Newton's method is yet another method for minimising the sum-of-squares objective function (2). More specifically, in this method, it is assumed that the objective function is approximately quadratic in the parameters near the optimal solution [35]. In doing so, we have also calculated the Hessian, $(\boldsymbol{H}(f+\lambda g))$, the second derivative with respect to each element $\mu_{\kappa d}$ while $\kappa = 1,\dots,K; d = 1,\dots,D$ as follows (for simplicity we considered $\lambda = 1$):

$$\frac{\partial^2 f}{\partial \mu_{\kappa d}\partial \mu_{\kappa d}} = w_\kappa \sum_{i,j=1}^{D}\delta_{ij}\mu_{\kappa i}\mu_{\kappa j}\sum_{k=1}^{K} w_k\mu_{ki}\mu_{kj} + 12w_\kappa\sum_{i=1}^{D}\mu_{\kappa i}$$
$$(\sum_{k=1}^{K} w_k\mu_{kd}^2\mu_{ki} - T_{ddi}) + 4w_\kappa(\sum_{k=1}^{K} w_k\mu_{kd}^2 - M_{dd})$$
$$+ 4w_\kappa^2\sum_{i=1}^{D}\mu_{\kappa i}^2 \qquad \delta_{ij} = \begin{cases} 6, & i=j\neq d,\\ 12, & i\neq j\neq d,\\ 18, & i=j=d,\\ 24, & i=d; j\neq d.\end{cases}$$

$$\frac{\partial^2 f}{\partial \mu_{\kappa d}\partial \mu_{\kappa' d}} = w_\kappa w_{\kappa'}\sum_{i,j=1}^{D}\delta_{ij}\mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa' i}\mu_{\kappa j} + 4w_\kappa w'_\kappa\sum_{i=1,i\neq d}^{D}\mu_{\kappa i}\mu_{\kappa' i}$$
$$+ 8w_\kappa w_{\kappa'}\mu_{\kappa d}\mu_{\kappa' d}$$

$$\frac{\partial^2 f}{\partial \mu_{\kappa d}\partial \mu_{\kappa d'}} = 12w_\kappa^2\mu_{\kappa d'}\mu_{\kappa d}\sum_{j=1}^{D}\mu_{\kappa j}^2 + 4w_\kappa^2\mu_{\kappa d}\mu_{\kappa d'}$$
$$+ 12w_\kappa\sum_{j=1}^{D}\mu_{\kappa j}(\sum_{k=1}^{K} w_k\mu_{kd'}\mu_{kd}\mu_{kj} - T_{jdd'})$$
$$+ 4w_\kappa(\sum_{k=1}^{K} w_k\mu_{kd}\mu_{kd'} - M_{dd'})$$

$$\frac{\partial^2 f}{\partial \mu_{\kappa d}\partial \mu_{\kappa' d'}} = 12w_\kappa w_{\kappa'}\mu_{\kappa d'}\mu_{\kappa' d}\sum_{i=1,i\neq d,d\neq d'}^{D}\mu_{\kappa i}\mu_{\kappa' i}$$
$$+ 4w_\kappa\mu_{\kappa d'}w_{\kappa'}\mu_{\kappa' d}$$

$$\frac{\partial^2 f}{\partial w_\kappa \partial w_{\kappa'}} = 2\sum_{i=1}^{D}\mu_{\kappa i}^3\mu_{\kappa' i}^3 + 6\sum_{i,j=1,j\neq i}^{D}\mu_{\kappa i}^2\mu_{\kappa j}\mu_{\kappa' i}^2\mu_{\kappa' j}$$
$$+ 12\sum_{i,j,l=1,l\neq j\neq i}^{D}\mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa l}\mu_{\kappa' i}\mu_{\kappa' j}\mu_{\kappa' l}$$
$$+ 4\sum_{i,j=1,j\neq i}^{D}\mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa' i}\mu_{\kappa' j} + 2\sum_{i=1}^{D}\mu_{\kappa i}^2\mu_{\kappa' i}^2$$

This can also be organised in a more compact form as:

$$\frac{\partial f}{\partial w_\kappa w_{\kappa'}} = 2(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})^3 + 2(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})^2$$
$$\frac{\partial f}{\partial \underline{\boldsymbol{\mu}}_\kappa\underline{\boldsymbol{\mu}}_{\kappa'}} = 18w_\kappa w_{\kappa'}(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})^2 + 8w_\kappa w_{\kappa'}(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})$$

$$\frac{\partial f}{\partial \underline{\boldsymbol{\mu}}_\kappa\underline{\boldsymbol{\mu}}_\kappa} = -12w_\kappa T\underline{\boldsymbol{\mu}}_\kappa + 12w_\kappa\sum_{k=1}^{K} w_k(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_k)\underline{\boldsymbol{\mu}}_k - 4w_\kappa M$$
$$+ 4w_k\sum_{k=1}^{K} w_k\underline{\boldsymbol{\mu}}_k^2$$

Note that in the above formula $(\underline{\boldsymbol{\mu}}_\kappa^T\underline{\boldsymbol{\mu}}_{\kappa'})^2$ equals $\sum_{i=1}^{D}\mu_{\kappa i}^2\mu_{\kappa' i}^2 + \sum_{i,j=1,j\neq i}^{D}\mu_{\kappa i}\mu_{\kappa j}\mu_{\kappa' i}\mu_{\kappa' j}$ which can be written based on index representation and its monomial as:

$$\sum_{i_1}\sum_{i_2}\sum_{i_3}\delta_{\mathbf{c}}(\mu_{\kappa 1}\mu_{\kappa' 1})^{c_1}\dots(\mu_{\kappa D}\mu_{\kappa' D})^{c_D}$$

for $\mathbf{c} = (c_1,\dots,c_D)\in\{0,1,2\}$.

Provided that $f+\lambda g$ is a twice-differentiable function then the Newton's method with a quadratic rate of convergence towards a local minimum than gradient descent can be used instead of gradient descent. Thus, our method can be easily modified for solving the problem (2) using the second order information which converges with much fewer iterations than gradient methods. To do so the following modification can be applied on steps 8 and 10 in Algorithm 1.
First we fix w_k and update $\underline{\boldsymbol{\mu}}_k$:

$$\underline{\boldsymbol{\mu}}_k^{n+1} = \underline{\boldsymbol{\mu}}_k^n + \alpha^n\Big(\nabla f(\underline{\boldsymbol{\mu}}_k^n|w_k) + \lambda\nabla g(\underline{\boldsymbol{\mu}}_k^n|w_k)\Big)\Big[\boldsymbol{H}\Big(f(\underline{\boldsymbol{\mu}}_k^n|w_k) + \lambda g(\underline{\boldsymbol{\mu}}_k^n|w_k)\Big)\Big]^{-1}$$

Fix $\underline{\boldsymbol{\mu}}_k$ and update w:

$$w_k^{n+1} = w_k^n + \beta^n\Big(\nabla f(w_k^n|\underline{\boldsymbol{\mu}}_k) + \lambda\nabla g(w_k^n|\underline{\boldsymbol{\mu}}_k)\Big)\Big[\boldsymbol{H}[f(w_k^n|\underline{\boldsymbol{\mu}}_k) + \lambda g(w_k^n|\underline{\boldsymbol{\mu}}_k)\Big)\Big]^{-1}$$

A refinement of the Newton approach is Levenberg-Marquardt method which accelerates the chance of local convergence and avoids divergence. Basically, adding up a scaled version of the diagonal elements of the Hessian matrix to itself, $[\mathbf{H} + \eta\mathrm{diag}(\mathbf{H})]$, results in a faster convergence than the Levenberg damping and Gauss-Newton method. The Levenberg-Marquardt algorithm adaptively varies the parameter updates between the gradient descent update and the Gauss-Newton update, where small values of the algorithmic parameter λ result in a Gauss-Newton update and large values of λ result in a gradient descent update. The parameter λ is initialised to be large so that first updates are small steps in the steepest descent direction. If any iteration happens to result in a worse approximation $F(\underline{\boldsymbol{\mu}}^{n+1},\underline{\mathbf{w}}^{n+1}) > F(\underline{\boldsymbol{\mu}}^n,\underline{\mathbf{w}}^n)$, then λ is increased. Otherwise, as the solution improves, λ is decreased.

7.2. Moment updates for online learning

In this Section we detail the update function used in Algorithm 2. The function combines stored variables from the previous data sets, $S_1, S_2, \dots, S_m$, with the newly arrived data batch, $S_{m+1}, S_{m+2}, \dots$, in a weighted manor allowing for exponential smoothing of the updates (this is what allows the algorithm to track evolving centres). Moment and statistic updates using exponential smoothing are well known and may be found in [32]. The updates consist of 4 steps detailed below.

1. Updates for the mean: $\underline{\boldsymbol{\mu}}^{m+1} = \underline{\boldsymbol{\mu}}^{1:m} + \frac{n_{m+1}}{n_{1:m}+n_{m+1}}(\underline{\boldsymbol{\mu}}^{1:m} - \underline{\boldsymbol{\mu}}^{m+1})$.
2. Updates for the covariance matrix: Given an update of the mean we may now turn to updating the covariance matrix of

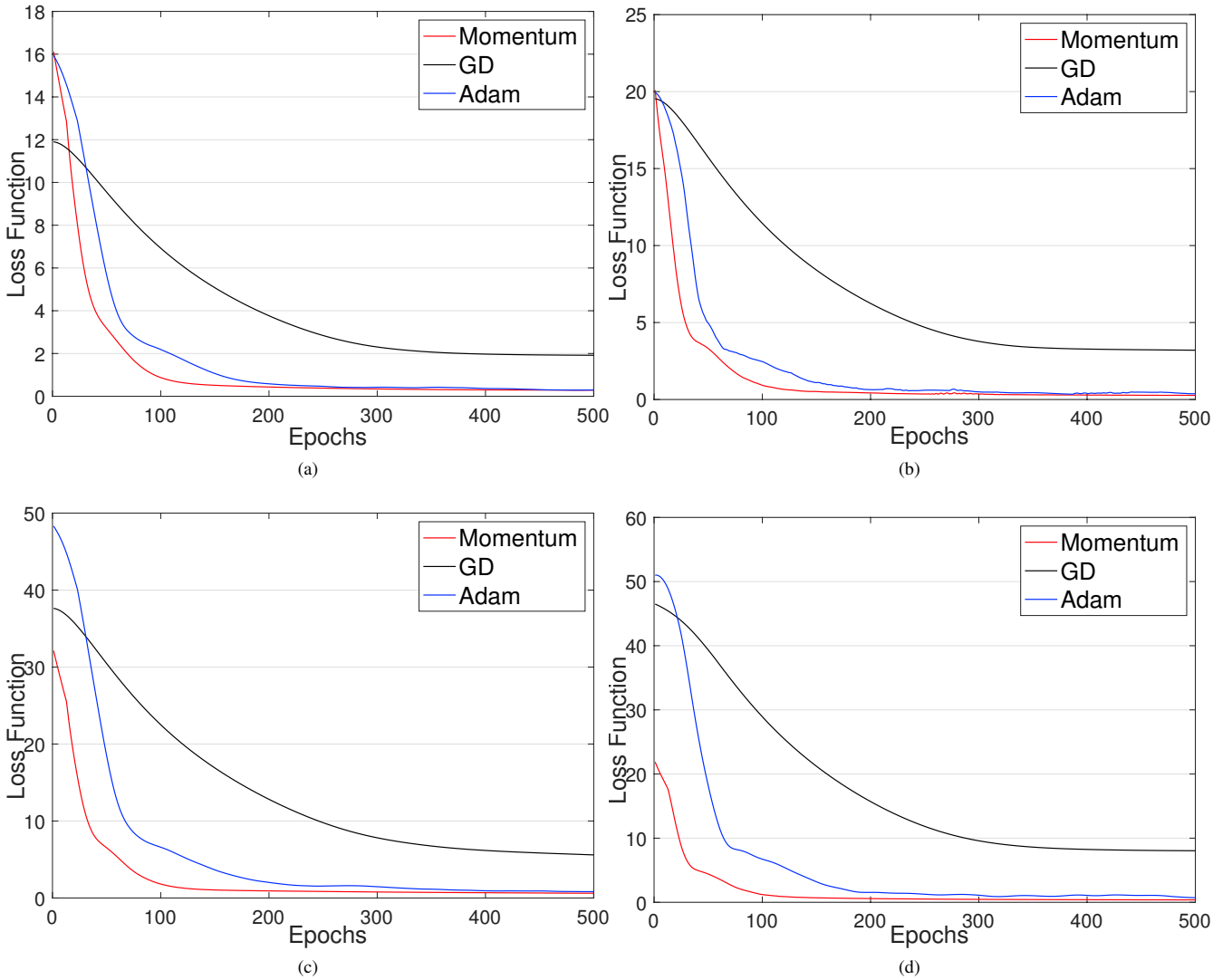


Fig. 8: *Summary of average convergence error rate GD, momentum and Adam. (a) $K = 5$, $D = 5$, (b) $K = 5$, $D = 20$, (c) $K = 10$, $D = 20$ and (d) $K = 10$, $D = 50$.*

the data, i.e. $C_{\underline{\mathbf{x}}}$ from Theorem 1 at time m denoted $C_{\underline{\mathbf{x}}}^{1:m}$. Note that $C_{\underline{\mathbf{x}}}^{m+1}$ is now the covariance matrix of a partitioned set $S_{1:m} \cup S_{m+1}$. In general, for two variables $\underline{\mathbf{x}}_2, \underline{\mathbf{x}}_2$ and two partitions of the data the cross-covariance of the union may be expressed as [32] (Eqn. 22) as:

$$C_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{m+1} = \frac{1}{n-1}\left(\mathcal{S}_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{1:m} + \mathcal{S}_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{m+1} + \frac{(\gamma_2\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_1}^{1:m} - \gamma_1\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_1}^{m+1})(\gamma_2\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_2}^{1:m} - \gamma_1\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_2}^{m+1})}{\gamma_1\gamma_2(\gamma_1+\gamma_2)}\right), \tag{13}$$

where $C_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{m+1}$ is the combined covariance of $\underline{\mathbf{x}}_1$ with $\underline{\mathbf{x}}_2$, $n_{1:m}$ is the number of samples in $S_1, \ldots, S_m$, $\mathcal{S}_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{1:m}$ is sum of the dot product of $\underline{\mathbf{x}}_1$ and $\underline{\mathbf{x}}_2$ in set $S_1, \ldots, S_m$, $\underline{\boldsymbol{\mu}}_{\underline{\mathbf{x}}_1}^{m+1}$ denotes the mean of $\underline{\mathbf{x}}_1$ in set S_{m+1}, and γ_m and γ_{m+1} are weights which can allow one to give a greater weight to one set over the other; this allows for exponential smoothing in an online setting (See [32] Section 4.2). Specifically we set $\gamma_1 = \gamma$ and $\gamma_2 = 1 - \gamma$ which implements exponential smoothing. 3. Updates for the eigenpairs: Recalculate the first eigenvalue and eigenvector, for $C_{\underline{\mathbf{x}}_1\underline{\mathbf{x}}_2}^{m+1}$. 4. higher order moment updates: Give updates for M and $\mathcal{T}$ with new data and new eigenvalue and eigenvector.

$$M^{m+1} = (1-\gamma)M^m + \gamma(x \otimes x) + \bar{\sigma}^{2^{m+1}} I,$$
$$\mathcal{T}^{m+1} = (1-\gamma)\mathcal{T}^m + \gamma(x \otimes x \otimes x).$$

The updated M and $\mathcal{T}$ are used as inputs for AGD-MoM algorithm and update parameter estimates for $\hat{\underline{\boldsymbol{\mu}}}_1, \ldots, \hat{\underline{\boldsymbol{\mu}}}_K$ and $\hat{\underline{\mathbf{w}}}$ for the next batch.