Scalable Extreme Deconvolution

James A. Ritchie
School of Informatics
University of Edinburgh
james.ritchie@ed.ac.uk

Iain Murray
School of Informatics
University of Edinburgh
i.murray@ed.ac.uk

Abstract

The Extreme Deconvolution method fits a probability density to a dataset where each observation has Gaussian noise added with a known sample-specific covariance, originally intended for use with astronomical datasets. The existing fitting method is batch EM, which would not normally be applied to large datasets such as the Gaia catalog containing noisy observations of a billion stars. We propose two minibatch variants of extreme deconvolution, based on an online variation of the EM algorithm, and direct gradient-based optimisation of the log-likelihood, both of which can run on GPUs. We demonstrate that these methods provide faster fitting, whilst being able to scale to much larger models for use with larger datasets.

1 Introduction

Extreme deconvolution is a method that fits Gaussian Mixture Models (GMMs) to noisy data where we know the covariance of the Gaussian noise added to each sample [3]. The method was originally developed to perform probabilistic density estimation on the dataset of stellar velocities produced by the Hipparcos satellite [13]. The Hipparcos catalogue consists of astrometric solutions (positions and velocities on the sky) and photometry (light intensity) for 118,218 stars, with associated noise covariances provided for each entry.

The successor to the Hipparcos mission, Gaia, aims to produce an even larger catalogue, with entries for an estimated 1 billion astronomical objects [17]. Previous work using an extreme deconvolution model on the Gaia catalogue worked with a subset of the data and restricted the number of mixture components, but the intention is to fit models with the full dataset [1]. The existing extreme deconvolution algorithm makes a full pass over the dataset before it can update parameters, and the reference implementation requires all the data to fit in memory. To fit such large datasets in reasonable time, we would normally use stochastic or online methods, with updates based on minibatches of data to make the methods practical on GPUs [2].

In this work, we develop two minibatch methods for fitting the extreme deconvolution model based on 1) an online variation of the Expectation-Maximisation (EM) algorithm, and 2) a gradient optimizer. Our implementations can run on GPUs, and provide comparable density estimates to the existing method, whilst being much faster to train.

2 Background

The aim of extreme deconvolution is to perform density estimation on a noisy d-dimensional dataset $\{\mathbf{x}_i\}_{i=0}^{N}$, where $\mathbf{x}_i$ was generated by adding zero-mean Gaussian noise ϵ_i with known per-datapoint covariance S_i to a projection R_i of a true value $\mathbf{v}_i$,

$$\mathbf{x}_i = R_i\mathbf{v}_i + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(\mathbf{0}, S_i). \tag{1}$$

Second Workshop on Machine Learning and the Physical Sciences (NeurIPS 2019), Vancouver, Canada.

We assume that $\mathbf{v}_i$ can be modelled by a mixture of Gaussians with K components,

$$p(\mathbf{v}_i \mid \theta) = \sum_j^K \alpha_j \mathcal{N}(\mathbf{v} \mid \mathbf{m}_j, V_j), \quad \theta = \{\alpha_j, \mathbf{m}_j, V_j\}_{j=1}^K, \tag{2}$$

parameterised by mixture weight α_j, mean $\mathbf{m}_j$ and covariance V_j. As the noise model is Gaussian and the model of the underlying density is a mixture of Gaussians, the probability of $\mathbf{x}_i$ is also a Gaussian mixture. The total log-likehood of the model is

$$\mathcal{L}(\theta) = \sum_i^N \log \sum_j^K \alpha_j \mathcal{N}(\mathbf{x}_i \mid R_i \mathbf{m}_j, T_{ij}), \quad T_{ij} = R_i V_j R_i^T + S_i. \tag{3}$$

Missing data can be handled either by making R_i rank-deficient, or by setting elements of the covariance matrix S_i to very large values.

3 Methods

3.1 Minibatch Expectation-Maximisation

The original method of fitting the extreme deconvolution model used a modification of the Expectation-Maximisation (EM) algorithm for mixture models [5]. Here we describe a minibatch version of this algorithm based on Cappé and Moulines [4]'s online EM algorithm for latent data models. At each iteration t, we compute the sufficient statistics of the latent data $\mathbf{v}_i$ for each component j in the minibatch of size M, using our current estimate of the parameters,

$$r_{ij} = \frac{\alpha_j \mathcal{N}(\mathbf{x}_i \mid R_i \mathbf{m}_j, T_{ij})}{\sum_k \alpha_k \mathcal{N}(\mathbf{x}_i \mid R_i \mathbf{m}_k, T_{ik})}, \ \mathbf{b}_{ij} = m_j + V_j R_i^T T_{ij}^{-1}(\mathbf{x}_i - R_i \mathbf{m}_j), \ B_{ij} = V_j - V_j R_i^T T_{ij}^{-1} R_i V_j. \tag{4}$$

The r_{ij} term is the posterior probability of datapoint $\mathbf{x}_i$ coming from component j. The $\mathbf{b}_{ij}$ and B_{ij} terms result from the fact that $\mathbf{x}_i$ and $\mathbf{v}_i$ are jointly Gaussian, so the distribution of $\mathbf{v}_i$ conditioned on $\mathbf{x}_i$ is also Gaussian with mean $\mathbf{b}_{ij}$ and covariance B_{ij}. The expected sufficient statistics are then summed together over the minibatch,

$$q_{jt} = \sum_i r_{ijt}, \quad \mathbf{s}_{jt} = \sum_i r_{ijt} \mathbf{b}_{ijt}, \quad S_{jt} = \sum_i r_{ijt} [\mathbf{b}_{ijt} \mathbf{b}_{ijt}^T + B_{ijt}]. \tag{5}$$

Stochastic estimates $\hat{\phi}_{jt}$ of the sums of sufficient statistics over the whole dataset are then updated with a sufficiently small step size λ,

$$\hat{\phi}_{jt} = (1-\lambda)\hat{\phi}_{j(t-1)} + \lambda \phi_{jt}, \quad \phi_{jt} = \{q_{jt}, \mathbf{s}_{jt}, S_{jt}\}, \quad \hat{\phi}_{jt} = \{\hat{q}_{jt}, \hat{\mathbf{s}}_{jt}, \hat{S}_{jt}\}. \tag{6}$$

Finally, we normalise the updated sums of expected sufficient statistics to get updated estimates of the parameters,

$$\alpha_j = \frac{\hat{q}_{jt}}{M}, \quad \mathbf{m}_j = \frac{\hat{\mathbf{s}}_{jt}}{\hat{q}_{jt}}, \quad V_j = \frac{\hat{S}_{jt}}{\hat{q}_{jt}} - \mathbf{m}_j \mathbf{m}_j^T. \tag{7}$$

This procedure is repeated with new randomly-ordered minibatches until convergence. If we set $\lambda = 1$ and replace each minibatch with the entire dataset, then the update corresponds to the original batch fitting method. Numerically however, the update for V_j, as written in (7), is inadvisable compared to the batch update given in [3]. There is likely to be catastrophic cancellation if the variances of the components are small relative to the means, especially if single precision floats are used, as is standard with GPU computation. In Appendix A we show how the minibatch version of this update can be rewritten in a more numerically stable form.

3.2 Stochastic Gradient Descent

An alternative to EM-based methods is to optimise the log-likelihood directly. The optimization is constrained, because the mixture weights a_j are positive and sum to 1, and the covariances V_j are positive definite. Directly fitting the log-likelihood with unconstrained gradient-based optimisers

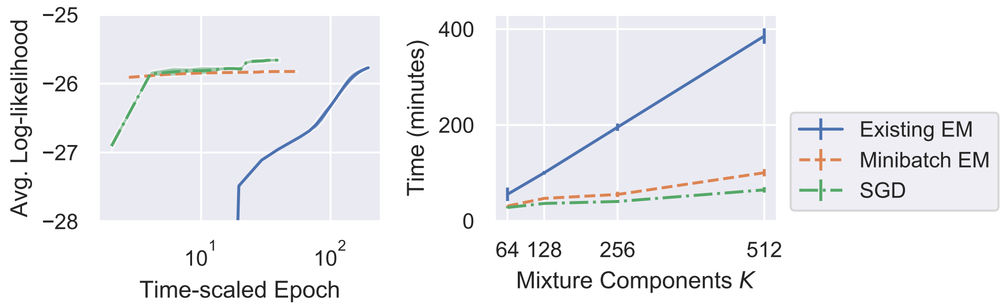

Figure 1: **Left**: Average training log-likelihood as a function of training on the Gaia subset for models with $K = 256$. Epochs rescaled by average training time. Error bars not visible. **Right**: Training time as a function of mixture components K. Error bars indicate $\pm$ 2 standard deviations.

requires a transformation of the parameters to remove the constraints [19]. The mixture weights α_j can be parameterised by taking the softmax of an unconstrained vector $\mathbf{z}$, and the covariances V_j represented by its lower triangular Cholesky decomposition L_j, where the diagonal elements qq of L_j are constrained positive by taking the exponential of unconstrained elements $\tilde{L}_q$,

$$\alpha_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad V_j = L_j L_j^T, \quad (L_j)_{qq} = \exp(\tilde{L}_q). \tag{8}$$

Having removed the constraints, we can optimise the likelihood using any standard minibatch gradient-based optimiser.

For a standard Gaussian mixture model, gradient based optimization has a scaling advantage over EM. There is no need to form the $D \times D$ covariance matrix V_j, since the Gaussian density can be evaluated directly from the Cholesky factor L_j in $O(D^2)$, whereas an EM update is $O(D^3)$. Unfortunately SGD updates are also $O(D^3)$ for the extreme deconvolution model, as we need to form the covariance T_{ij} for each datapoint.

4 Experiments

We implemented both minibatch approaches in PyTorch, and compared against the reference implementation from Bovy et al. [3]. To evaluate each method, we used a random sample of rows from the Gaia DR2 source table [18]. We selected the 5 primary astrometric features, along with the BP-RP colour and mean magnitude in the G-band. In total there were 2 million rows. Where data were missing, we set the field to zero and the noise variance to a large value. We set the projection R_i to the identity matrix for every sample. This preliminary study uses only a small fraction of the full dataset size, but this allows us to fit the training data into memory, a requirement for use with the original implementation of extreme deconvolution. We used a range of mixture component sizes K. In practice we would want to select a value of K by cross-validation.

The existing EM method ran on CPU, whilst the minibatch EM and SGD methods ran on GPU. While the absolute times depend strongly on hardware and fine implementation details, they give a sense of the sort of times possible on current workstations, and the relative times across model sizes illustrate how the methods scale. We used a validation set comprising 10% of the rows when developing our experiments. Final model performance was evaluated on a different held-out test set also comprising 10% of the rows at the last stage, with no parameter selection or development done based on this set. Details required for reproducibility are provided in Appendix B.

Table 1 reports the validation and test log-likelihoods for each method. The values are similar, but not exactly comparable, as the effect of regularisation differs for each method. Figure 1 plots the training log-likelihood against time-rescaled epoch for $K = 256$, and training time as function of mixture components K. Figure 2 shows a 2-D projection from an example model with $K = 256$ fitted with the minibatch-EM method.

Table 1: Average validation log-likelihoods for the Gaia data subset for different numbers of mixture components K, with average test log-likelihood for the best value of K by validation. Average over 10 runs with standard deviation.

Method	K	Validation	Test
Existing EM	64	-26.10 ± 0.03	-
Bovy et al. [3]	128	-25.96 ± 0.04	-
	256	-25.76 ± 0.02	-
	512	-25.67 ± 0.01	-25.66 ± 0.01
Minibatch EM	64	-26.05 ± 0.01	-
	128	-25.91 ± 0.01	-
	256	-25.83 ± 0.00	-
	512	-25.80 ± 0.00	-25.79 ± 0.00
SGD	64	-25.89 ± 0.02	-
	128	-25.77 ± 0.02	-
	256	-25.67 ± 0.02	-
	512	-25.59 ± 0.02	-25.57 ± 0.02

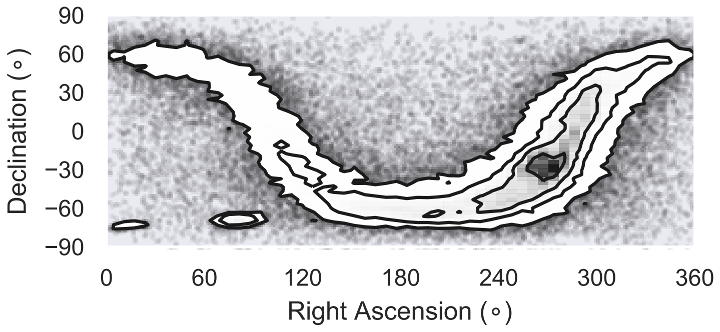

Figure 2: Density plot showing a 2-D projection of 100000 samples drawn from a model with $K = 256$ and fitted with the minibatch EM method. The plot shows the estimated density of star positions on the sky, and has correctly recovered the structure of the Milky Way and the Magellanic Clouds.

5 Discussion

Our results have shown that both of our proposed methods perform comparably to the existing method of fitting extreme deconvolution models, whilst converging faster. The results also show that using GPU-based computation speeds up fitting considerably, allowing sublinear scaling of training time with mixture component size K.

Further improvements to our approaches are possible. The original paper presents a method of getting out of local maxima by splitting and merging clusters, with the split-merge criteria evaluated on the whole dataset. It should be possible to replace the criteria with stochastic estimates, which would permit them to be used with both the SGD and minibatch EM methods. Our approaches also add more free parameters to be selected, including learning rate and batch size. This adds scope for hyperparameter optimisation to improve the log-likelihood.

In this preliminary study the SGD method provided the best log-likelihood values, was faster to train, and scaled better with component size K. In addition, we found SGD to be more numerically stable than minibatch-EM during training. Both minibatch methods will allow us to fit larger models going forwards.

Acknowledgments

We are grateful to David W. Hogg for providing us with the problem and advising us on the Gaia data. Our experiments made use of AstroPy [15, 16], corner.py [6], Matplotlib [7], Numpy [10], Pandas [9], PyTorch [11] and Scikit-learn [12]. This work was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

References

[1] L. Anderson, D. W. Hogg, B. Leistedt, A. M. Price-Whelan, and J. Bovy. Improving Gaia Parallax Precision with a Data-driven Model of Stars. *The Astronomical Journal*, 156(4):145, Sept. 2018.

[2] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Rev*, 60(2):223–311, 2018.

[3] J. Bovy, D. W. Hogg, and S. T. Roweis. Extreme deconvolution: Inferring complete distribution functions from noisy, heterogeneous and incomplete observations. *The Annals of Applied Statistics*, 5(2B):1657–1677, June 2011.

[4] O. Cappé and E. Moulines. On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):593–613, 2009.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1): 1–38, 1977.

[6] D. Foreman-Mackey. corner.py: Scatterplot matrices in python. *The Journal of Open Source Software*, 24, 2016.

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9 (3):90–95, 2007.

[8] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *3rd International Conference for Learning Representations*, 2014.

[9] W. McKinney. Data structures for statistical computing in python. In S. van der Walt and J. Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[10] T. Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006.

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] M. A. C. Perryman, L. Lindegren, J. Kovalevsky, E. Høg, U. Bastian, P. L. Bernacca, M. Crézé, F. Donati, M. Grenon, M. Grewing, F. Van Leeuwen, H. Van Der Marel, F. Mignard, C. A. Murray, R. S. Le Poole, H. Schrijver, C. Turon, F. Arenou, M. Froeschlé, and C. S. Petersen. The Hipparcos Catalogue. *Astronomy and Astrophysics*, 323(1):49–52, 1997.

[14] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web - WWW '10*, page 1177, Raleigh, North Carolina, USA, 2010. ACM Press.

[15] The Astropy Collaboration. Astropy: A community Python package for astronomy. *Astronomy & Astrophysics*, 558:A33, Oct. 2013.

[16] The Astropy Collaboration. The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. *The Astronomical Journal*, 156:123, Sept. 2018.

[17] The Gaia Collaboration. The Gaia mission. *Astronomy and Astrophysics*, 595:A1, Nov. 2016.

[18] The Gaia Collaboration. Gaia Data Release 2. Summary of the contents and survey properties. *Astronomy & Astrophysics*, 616:A1, Aug. 2018.

[19] P. M. Williams. Using neural networks to model conditional multivariate densities. *Neural Computation*, 8(4):843–854, 1996.

A Stable Covariance Update

In Section 3.1 describing our minibatch-EM method, we noted that the M-step update for the variance of each component V_j as presented in Equation 7 would be prone to catastrophic cancellation as a result of taking a small difference between large values using single-precision floats. Here we present an alternative update for V_j that is less prone to numerical instability, and show that it is equivalent to Equation 7. For clarity we drop the component indicator j from the parameters, and add indicators t and $t-1$ to distinguish between current and previous estimates of parameters.

First, we define an adjustment operation,

$$\begin{aligned}\text{adjust}(V, s, \mathbf{c}, \mathbf{d}) &= sV + \frac{1}{2}(\mathbf{c}-\mathbf{d})(\mathbf{c}+\mathbf{d})^T + \frac{1}{2}(\mathbf{c}+\mathbf{d})(\mathbf{c}-\mathbf{d})^T & (9)\\ &= s(V + \mathbf{c}\mathbf{c}^T) - \mathbf{d}\mathbf{d}^T, & (10)\end{aligned}$$

which can be thought of as recentering a scaled variance around a new mean. Equation 9 is how we actually compute the adjustment, to minimise taking small differences between large values, whilst Equation 10 shows the identity we are interested in.

In the M-step at iteration t of our minibatch EM approach, we compute estimates of $\hat{q}_t$, α_t and $\mathbf{m}_t$ as before using Equations 6 and 7. We also compute minibatch-specific parameters using exact sums over the minibatch:

$$q_b = \sum_i^M r_i, \quad \mathbf{m}_b = \frac{\sum_i^M r_i \mathbf{x}_i}{q_b}, \quad V_b = \frac{\sum_i^M r_i[(\mathbf{x}_i - \mathbf{b}_i)(\mathbf{x}_i - \mathbf{b}_i)^T + B_i]}{q_b} \tag{11}$$

We then compute our new estimate of the variance V_t as a function of the previous estimates $\{\hat{q}_{t-1}, \mathbf{m}_{t-1}, V_{t-1}\}$, the minibatch values $\{q_b, \mathbf{m}_b, V_b\}$, and the current estimates $\{\hat{q}_t, \mathbf{m}_t\}$:

$$\begin{aligned}V_t &= (1-\lambda)\,\text{adjust}(V_{t-1}, \frac{\hat{q}_{t-1}}{\hat{q}_t}, \mathbf{m}_{t-1}, \mathbf{m}_t) + \lambda\,\text{adjust}(V_b, \frac{q_b}{\hat{q}_t}, \mathbf{m}_b, \mathbf{m}_t) & (12)\\ &= (1-\lambda)\left[\frac{\hat{q}_{t-1}}{\hat{q}_t}\left(V_{t-1} + \mathbf{m}_{t-1}\mathbf{m}_{t-1}^T\right) - \mathbf{m}_t\mathbf{m}_t^T\right] + \lambda\left[\frac{q_b}{\hat{q}_t}\left(V_b + \mathbf{m}_b\mathbf{m}_b^T\right) - \mathbf{m}_t\mathbf{m}_t^T\right] & (13)\\ &= (1-\lambda)\left[\frac{\hat{S}_{t-1}}{\hat{q}_t} - \mathbf{m}_t\mathbf{m}_t^T\right] + \lambda\left[\frac{S_t}{\hat{q}_t} - \mathbf{m}_t\mathbf{m}_t^T\right] & (14)\\ &= \frac{(1-\lambda)\hat{S}_{t-1} + \lambda S_t}{\hat{q}_t} - \mathbf{m}_t\mathbf{m}_t^T & (15)\\ &= \frac{\hat{S}_t}{\hat{q}_t} - \mathbf{m}_t\mathbf{m}_t^T & (16)\end{aligned}$$

Again, Equation 12 is how we actually compute the update to minimise numerical errors, whilst Equation 16 shows that the update is equivalent to the covariance update defined in Equation 7. Whilst we found this update worked better in practice than a direct implementation, numerical instability is still possible if the standard deviations of the components are small enough relative to the means, and further work is needed to determine if a more stable update can be performed.

B Experiment Details

Here we provide specific details of our experiments for reproducibility. Code used to run the experiments is available at `https://github.com/bayesiains/scalable_xd`.

B.1 Dataset

From the Gaia DR2 source table we selected the columns `RA`, `DEC`, `PARALLAX`, `PMRA`, `PMDA`, `BP_RP` and `PHOT_G_MEAN_MAG` to assemble the observed dataset $\{\mathbf{x}_i\}_{i=0}^N$ [18]. Random subsampling was done by selecting rows with the value of the `RANDOM_INDEX` column less than 2,000,000. Noise covariance matrices S_i were assembled using the corresponding error and correlation columns for each variable. Where a column of a row was marked as missing, the corresponding element of $\mathbf{x}_i$

was set to zero, the corresponding diagonal element of S_i was set to 10^{12}, and the corresponding off-diagonal elements set to zero. For columns which do not have associated noise, the corresponding diagonal elements of S_i were set to a value of 10^{-2}, and corresponding off-diagonal elements to zero.

B.2 Initialisation

For each method, initialisation of the means and weights was done using the estimated counts and centroids after 10 epochs of minibatch k-means clustering [14]. Covariances V_j were set to the identity matrix.

B.3 Training

All methods were trained for a total of twenty epochs. Both minibatch methods used a batch size of 500. For the minibatch-EM method, the step size λ was started at 10^{-2} and reduced by a factor of two after ten epochs. For the SGD method, we used the Adam optimiser with a learning of 10^{-2} for the first ten epochs, reducing by a factor of ten for the last ten epochs, and all other parameters set to the suggested defaults [8].

For numerical stability, a very small amount of regularisation was applied to the covariances V_j. Using the original implementation, we set the regularisation constant $w = 10^{-3}$. For the minibatch-EM method, we added diagonal matrix wI directly to each covariance matrix after updating them. For the SGD method, we added a penalty term $\sum_j \frac{w}{\mathrm{Trace}[V_j]}$ to the loss function. As noted in Section 4, the effect of w is not comparable across methods. If we were using larger values of w to prevent overfitting rather than just avoiding numerical instability, w would be tuned specifically for each method.