Density Estimation with Noisy Data

James Ritchie

Hipparcos

- Noisy observations of ~118,200 stars
- Astrometric measurements
  - Where is it?
  - How fast is it?
- Photometric measurements
  - How bright is it?
  - What colour is it?

Hipparcos

D-dimensional data with noise:

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

How can we do density estimation on $\mathbf{x}_i$ when we only have $\mathbf{v}_i$?

$$p(\mathbf{v}_i) = \int \mathcal{N}(\mathbf{v}_i \mid x_i\,, S_i)\, p(\mathbf{x}_i)\, d\mathbf{x}$$

Extreme Deconvolution [1]

Let's model $p(\mathbf{x}_i)$ using a Gaussian Mixture Model

$$p(\mathbf{x}_i) = \sum_{j=1}^{K} \alpha_j \mathcal{N}(\mathbf{x}_i \mid \mu_j, \Sigma_j)$$

---

[1] Bovy, Jo, David W. Hogg, and Sam T. Roweis. "Extreme deconvolution" The Annals of Applied Statistics 5.2B (2011): 1657-1677.

Extreme Deconvolution [1]

As the noise is Gaussian, everything works out nicely!

$$p(\mathbf{v}_i) = \sum_{j=1}^{K} \alpha_j \mathcal{N}(\mathbf{v}_i \mid \mu_j, T_{ij}), \quad T_{ij} = S_i + \Sigma_j$$

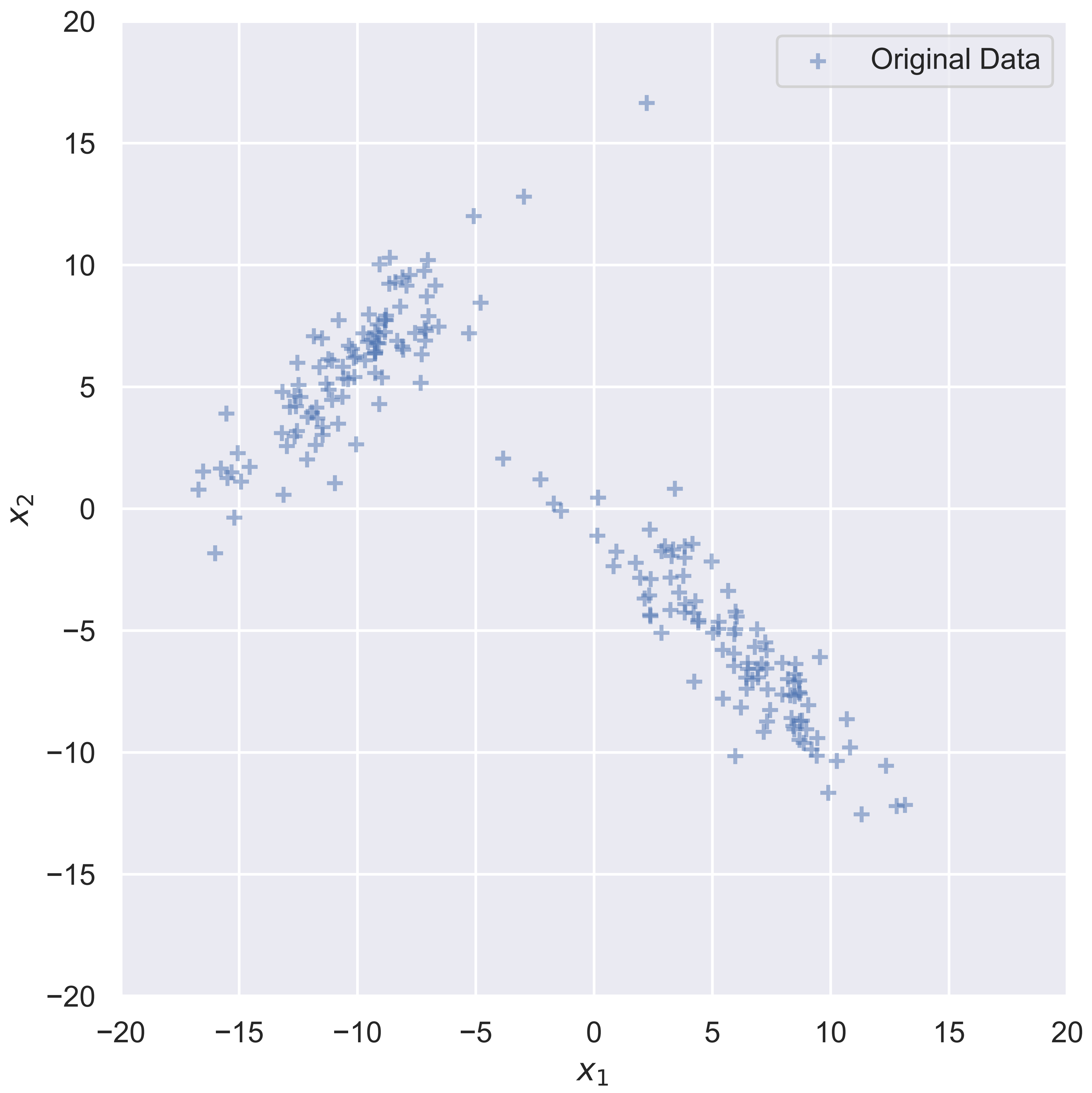
[1] Bovy, Jo, David W. Hogg, and Sam T. Roweis. "Extreme deconvolution" The Annals of Applied Statistics 5.2B (2011): 1657-1677.

Extreme Deconvolution [1]

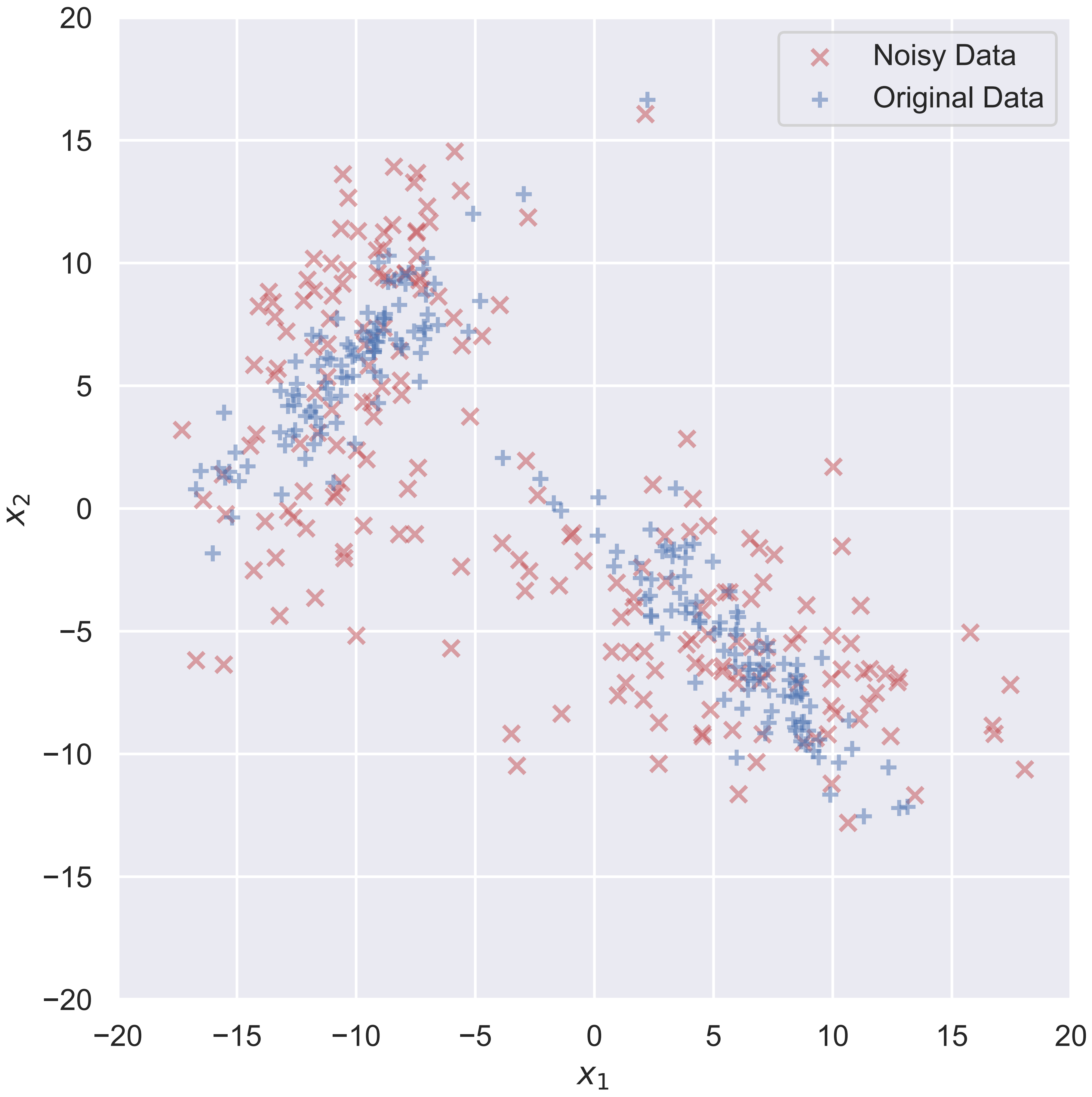
We can fit the GMM with Expectation-Maximisation

1. E-step: Compute expected statistics for each datapoint
2. M-step:
   - Sum these statistics together.
   - Normalise the sums to get the GMM parameters.

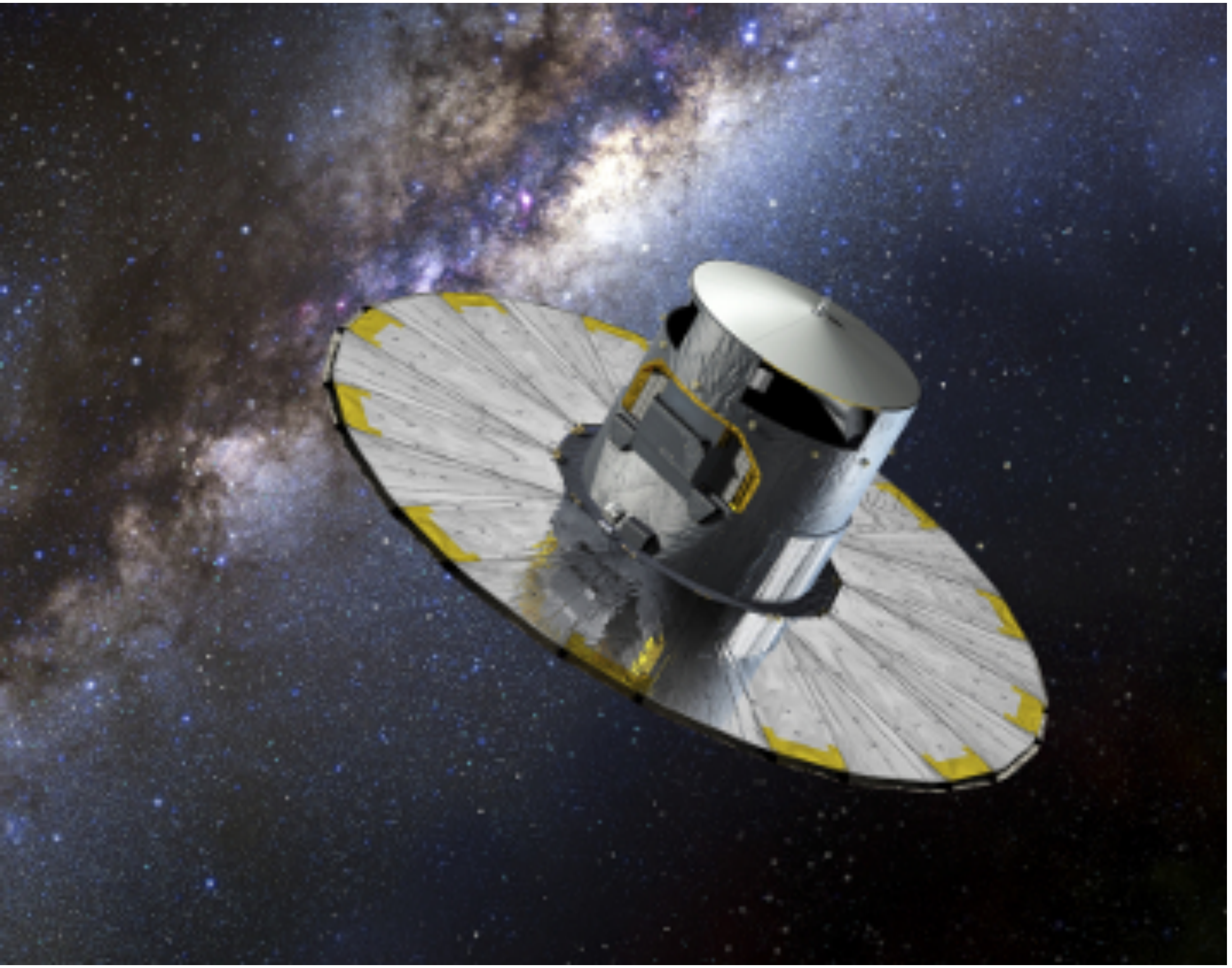
[1] Bovy, Jo, David W. Hogg, and Sam T. Roweis. "Extreme deconvolution" The Annals of Applied Statistics 5.2B (2011): 1657-1677.



Original Data
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20
x_1
x_2


Noisy Data
Original Data
x_1
x_2
−20
−15
−10
−5
0
5
10
15
20


Noisy Data
Original Data
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20
x_1
x_2



Gaia

- Let's do Hipparcos again, but bigger!
- Approx 1 billion observations! (Eventually)
- Currently 550GB when gzip-ed

Scalable Extreme Deconvolution [2]

- Can't fit the entire dataset in memory easily
- Are minibatch methods better?
- Will using a GPU make it faster?

[2] Ritchie, James A., and Iain Murray. "Scalable Extreme Deconvolution." arXiv preprint arXiv:1911.11663 (2019).

Minibatch EM [3]

Core idea: Replace the sum over the entire dataset with moving average estimates.

$$\phi_j^t = (1 - \lambda)\phi_j^{t-1} + \lambda\hat{\phi}_j$$

Normalise the sum estimates to get the parameters.

Write it with PyTorch so we can run it on the GPU.

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

Minibatch EM Problem

We really want to compute covariances like this:

$$\Sigma = E[(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T]$$

But we have to do it like this:

$$\Sigma = E[\mathbf{x}\mathbf{x}^T] - \mu\mu^T$$

What happens if Σ is small relative to $\mu\mu^T$?

Catastrophic Cancellation

Small difference between two large numbers

929661.734710**6681** - 929661.734710**5937**

Blows up suprisingly quickly with 32 bit single precision floats on GPU.

Stochastic Gradient Descent

Gradient-based minibatch optimisers are pretty good, can we use those?

Need to get rid of the constraints:

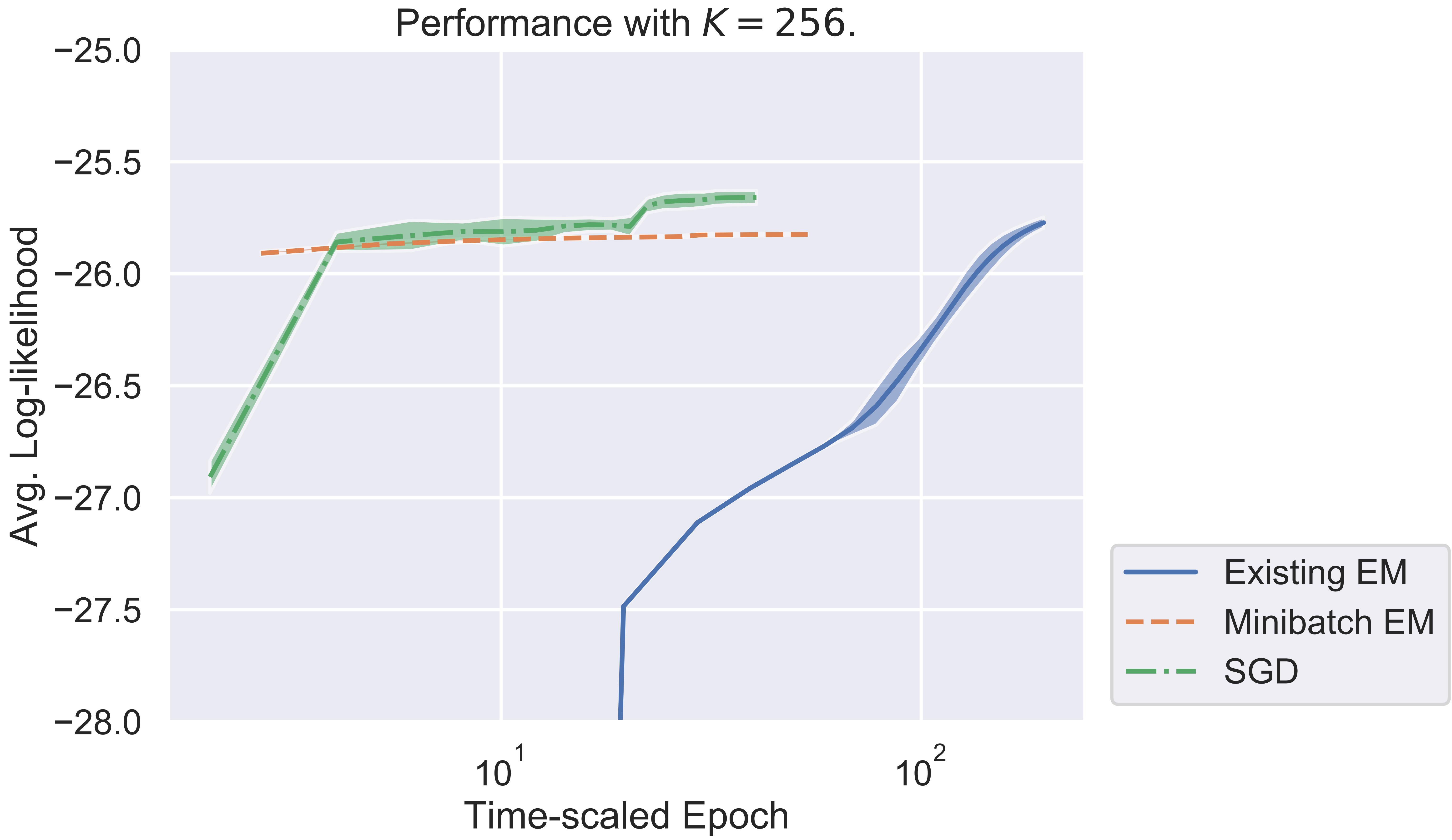
1. Mixture weights α_j must add up to 1.
2. Covariances Σ_j must be positive-definite.

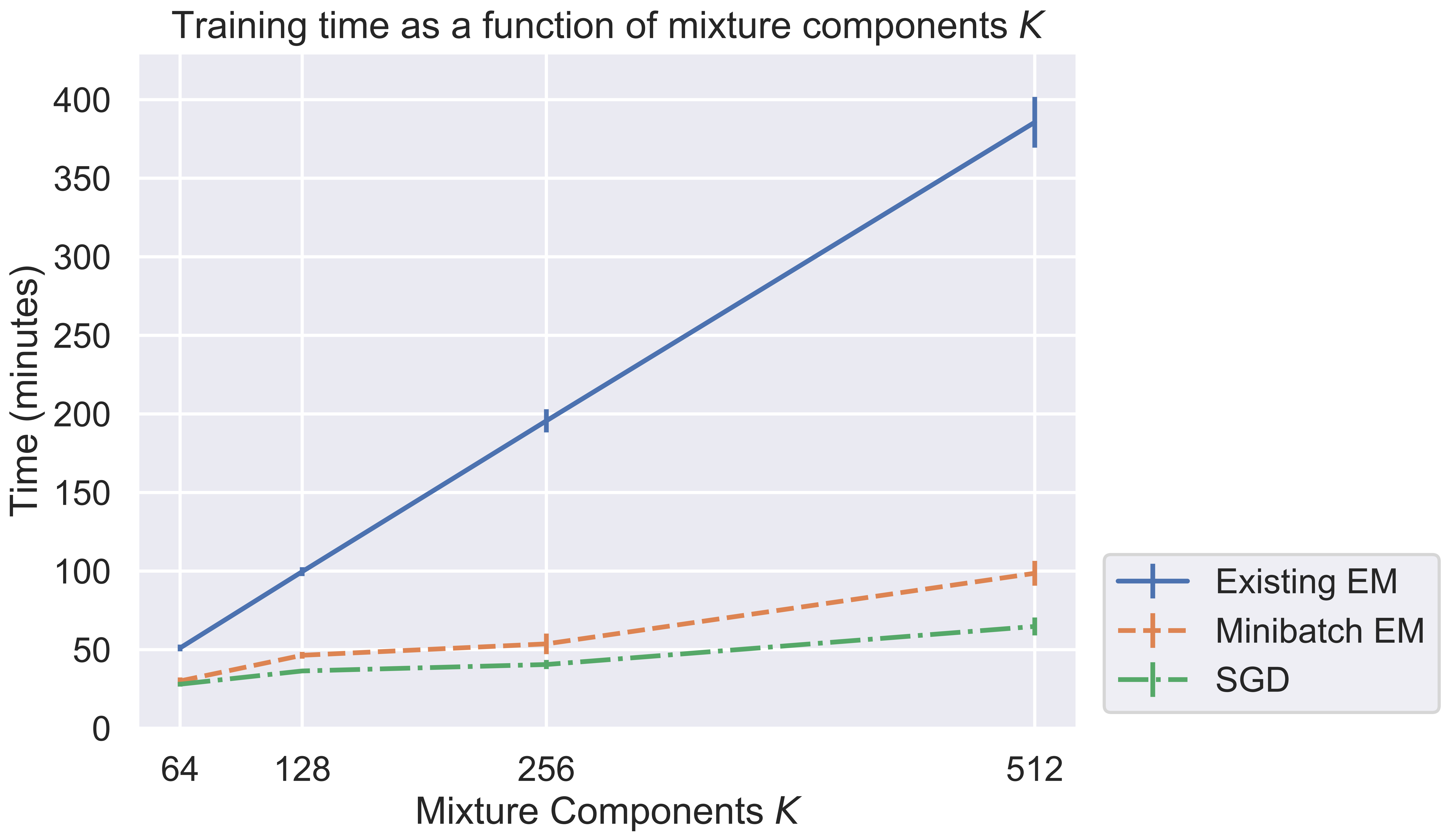
Stochastic Gradient Descent

Can do this with reparameterisation:

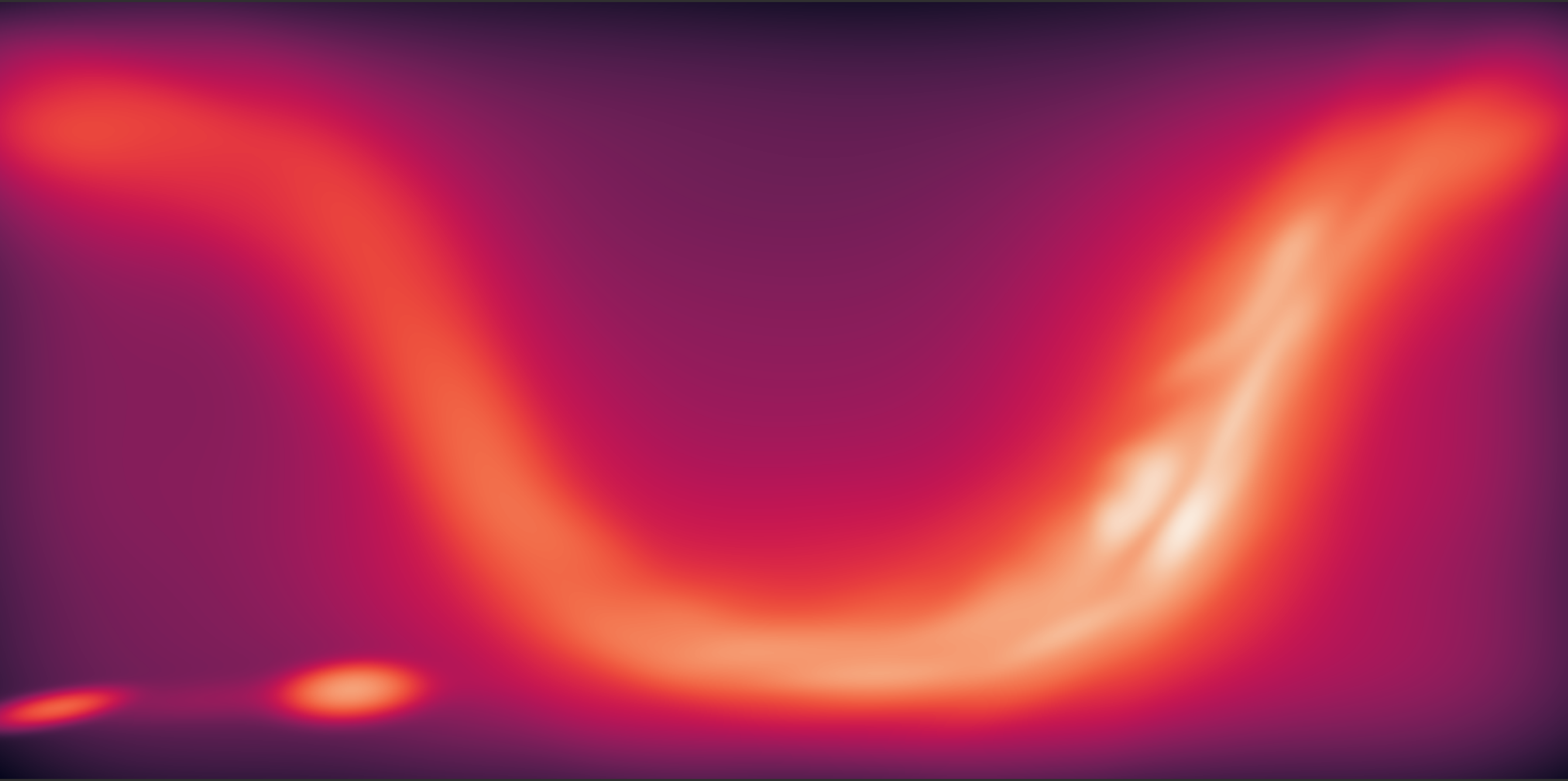
1. Take the softmax of an unconstrained vector $\mathbf{z}$ to get α.
2. Represent covariance via lower-triangular Cholesky, $\Sigma_j = L_j L_j^T$.
3. Keep the diagonal of L_j positive, $(L_j)_{qq} = \exp(\hat{L}_{jq})$.

PyTorch takes care of computing all the gradients.


Performance with $K = 256$.
Avg. Log-likelihood
−25.0
−25.5
−26.0
−26.5
−27.0
−27.5
−28.0
10^1
10^2
Time-scaled Epoch
Existing EM
Minibatch EM
SGD



Training time as a function of mixture components K
Time (minutes)
0
50
100
150
200
250
300
350
400
64
128
256
512
Mixture Components K
Existing EM
Minibatch EM
SGD



Conclusion

Don't use EM for mixture models

The Future

Gaussian Mixture Models are still pretty terrible.

— Cholesky decomposition for every combination of datapoint i and mixture component j
— Many mixture components needed to cover high dimensional space.
— Mixture components can't share information.

The Future

Could using a neural network to model $p(x)$ be better?

$$\mathrm{argmax}_\theta \log p(\mathbf{V}) = \sum_{i=1}^{N} \log \int \mathcal{N}(\mathbf{v}_i \mid \mathbf{x}_i, S_i)\, p(x_i \mid \theta)\, d\mathbf{x}$$

Questions?

For more details see:

"Scalable Extreme Deconvolution." Ritchie, James A., and Iain Murray. arXiv preprint arXiv:1911.11663 (2019)