Expectation Maximization for Gaussian Mixture Models on the GPU

Garrett Lewellen

April 1, 2017

1 Introduction

Gaussian Mixture Models [1, 435-439] offer a simple way to capture complex densities by employing a linear combination of K multivariate Gaussian distributions, each with their own mean, covariance, and mixture coefficient, π_k, s.t. $\sum_k \pi_k = 1$.

$$p(x) = \sum_{k=1}^{K} \pi_k p(x|\mu_k, \Sigma_k) \tag{1}$$

Of practical interest is the learning of the number of components and the values of the parameters. Evaluation criteria, such as AIC or BIC, can be used to identify the number of components, or non-parametric models like Dirichlet processes can be used to avoid the matter all together. We won't cover these techniques here, but will instead focus on finding the values of the parameters given sufficient training data using the Expectation-Maximization algorithm [2], and doing so efficiently on the GPU. Technical considerations will be discussed and the work will conclude with an empirical evaluation of sequential and parallel implementations for the CPU, and a massively parallel implementation for the GPU for varying numbers of components, points, and point dimensions.

2 Multivariate Gaussian Distribution

The multivariate Gaussian distribution With mean, $\mu \in \mathbb{R}^d, d \in \mathbb{N}_1$, and symmetric, positive definite covariance, $\Sigma \in \mathbb{R}^{d \times d}$, is given by:

$$p(x|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-(x-\mu)^T \Sigma^- (x-\mu)/2\right) \tag{2}$$

From a computational perspective, we will be interested in evaluating the density for N values. Thus, a naive implementation would be bounded by $\mathcal{O}\left(Nd^4\right)$ due to the matrix determinate in the normalization term. We can improve upon this by computing the Cholesky factorization, $\Sigma = LL^T$, where L is a lower triangular matrix [4, 157-158]. The factorization requries $\mathcal{O}\left(d^3\right)$ time and computing the determinate becomes $\mathcal{O}\left(d\right)$ by taking advantage of the fact that $\det\left(LL^T\right) = \det(L)^2 = \prod_i L_{i,i}^2$. Further, we can precompute the factorization and normalization factor for a given parameterization which leaves us with complexity of the Mahalanobis distance given by the quadratic form in the exponential. Naive computation requires one perform two vector matrix operations and find the inverse of the covariance matrix with worst case behavior $\mathcal{O}\left(d^3\right)$. Leveraging the Cholesky factorization, we'll end up solving a series of triangular systems by forward and backward substituion in $\mathcal{O}\left(d^2\right)$ and completing an inner product in $\mathcal{O}\left(d\right)$ as given by $Lz = x - \mu$, $L^T z = y$, and $(x-\mu)^T y$. Thus, our pre-initialization time is $\mathcal{O}\left(d^3\right)$ and density determination given by $\mathcal{O}\left(Nd^2\right)$. Further

optimizations are possible by considering special diagonal cases of the covariance matrix, such as the isotropic, $\Sigma = \sigma I$, and non-isotropic, $\Sigma_{k,k} = \sigma_k$, cases. For robustness, we'll stick with the full covariance.

$$\log p(x|\mu, \Sigma) = -\frac{1}{2}\left(d \log 2\pi + \log|\Sigma|\right) - \frac{1}{2}(x-\mu)^T \Sigma^{-}(x-\mu) \tag{3}$$

To avoid numerical issues such as overflow and underflow, we're going to consider $\log p(x|\mu, \Sigma)$ throughout the remainder of the work. For estimates of the covariance matrix, we will want more samples than the dimension of the data to avoid a singular covariance matrix [3]. Even with this criteria satisfied, it may still be possible to produce a singular matrix if some of the data are collinear and span a subspace of $\mathbb{R}^d$.

3 Expectation Maximization

From an unsupervised learning point of view, GMMs can be seen as a generalization of k-means allowing for partial assignment of points to multiple classes. A possible classifier is given by $k^* = \arg\max_k \log \pi_k + \log p(x|\mu_k, \Sigma_k)$. Alternatively, multiple components can be used to represent a single class and we argmax over the corresponding subset sums. The Expectation-Maximization (EM) algorithm will be used to find the parameters of of the model by incrementally computing probabilities given a fixed set of parameters, then updating those parameters by maximizing the log-likelihood of the data:

$$\mathcal{L}\left(\mathcal{D}|\mu, \Sigma\right) = \sum_{n=1}^{N} \log p(x_n) = \sum_{n=1}^{N} \log \left[\sum_{k=1}^{K} \pi_k p\left(x_n|\mu_k, \Sigma_k\right)\right] \tag{4}$$

Because we are dealing with exponents and logarithms, it's very easy to end up with underflow and overflow situtations, so we'll continue the trend of working in log-space and also make use of the "log-sum-exp trick" to avoid these complications:

$$\log p(x) = a + \log \left[\sum_{k=1}^{K} \exp\left(\log \pi_k + \log p(x|\mu_k, \Sigma_k) - a\right)\right] \tag{5}$$

Where the a term is the maximum exponential argument within a stated sum. Within the expectation stage of the algorithm we will compute the posterior distributions of the components conditioned on the training data (we ommit the mixing coefficient since it cancels out in the maximization steps of μ_k and Σ_k, and account for it explicitly in the update of π_k):

$$\gamma_{k,n} = \frac{p\left(x_n|\mu_k, \Sigma_k\right)}{p(x)} \qquad \Gamma_k = \sum_{n=1}^{N} \gamma_{k,n} \tag{6}$$

$$\log \gamma_{k,n} = \log p\left(x_n|\mu_k, \Sigma_k\right) - \log p(x) \qquad \log \Gamma_k = a + \log \left[\sum_{n=1}^{N} \exp\left(\log \gamma_{k,n} - a\right)\right] \tag{7}$$

The new parameters are resolved within the maximization step:

$$\pi_k^{(t+1)} = \frac{\pi_k^{(t)} \Gamma_k}{\sum_{i=1}^{K} \pi_i^{(t)} \Gamma_i} \qquad \log \pi_k^{(t+1)} = \log \pi_k^{(t)} + \log \Gamma_k - a - \log \left[\sum_{i=1}^{K} \exp\left(\log \pi_i^{(t)} + \log \Gamma_i - a\right)\right] \tag{8}$$

$$\mu_k^{(t+1)} = \frac{\sum_{n=1}^{N} x_n \gamma_{n,k}}{\Gamma_k} \qquad \mu_k^{(t+1)} = \frac{\sum_{n=1}^{N} x_n \exp \log \gamma_{n,k}}{\exp \log \Gamma_k} \tag{9}$$

$$\Sigma_k^{(t+1)} = \frac{\sum_{n=1}^{N}(x_n - \mu_k^{(t+1)})(x_n - \mu_k^{(t+1)})^T \gamma_{n,k}}{\Gamma_k} \tag{10}$$

$$\Sigma_k^{(t+1)} = \frac{\sum_{n=1}^{N}(x_n - \mu_k^{(t+1)})(x_n - \mu_k^{(t+1)})^T \exp\log \gamma_{n,k}}{\exp\log \Gamma_k} \tag{11}$$

The algorithm continues back and forth between expectation and maximization stages until the change in log likelihood is less than some epsilon, or a maximum number of user specified iterations has elapsed.

4 Implementations

Sequential Per iteration complexity given by $\mathcal{O}\left(2KNd^2 + KNd + 2K + N + Kd^3\right)$. We expect $d \leq K < N$ because too many dimensions leads to a lot of dead space and too many components results in overfitting of the data. Thus, the dominating term for sequential execution is given by $\mathcal{O}\left(2KNd^2\right)$.

Parallel

Massively Parallel

5 Evaluation

Three graphs represent sequential, parallel, and gpu:

1. Fix d, k, vary N
2. Fix d, N, vary k
3. Fix k, N, vary d

Two graphs

1. Fix d, k, N vary number of threads
2. Fix d, k, n vary number of streaming multiprocessors (need to think that one over)

Each graph should have about 10-15 samples per data point; each graph should include error bars after rejecting outliers. Times should be measured in seconds. Need to decide on commenting on gpu 32-bit single precision (for cheapo GPUs) and 64-bit double precision values on CPU.

6 Conclusion

References

[1] Bishop, C. M. *Pattern recognition and machine learning*. Springer, 2006.

[2] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)* (1977), 1–38.

[3] Fan, J., Liao, Y., and Liu, H. An overview of the estimation of large covariance and precision matrices. *The Econometrics Journal 19*, 1 (2016), C1–C32.

[4] Kincaid, D., and Cheney, W. *Numerical analysis: mathematics of scientific computing*, 3 ed. Brooks/Cole, 2002.