Scaling up sampled matrix factorization for single-cell transcriptomic data

Sumit Mukherjee
Electrical Engineering
University of Washington
Seattle, WA

Joshua Fan
Computer Science & Engineering
University of Washington
Seattle, WA

Abstract

With rapid advances in sequencing technologies, large single cell transcriptomic (SCS) datasets are becoming abundant and larger than ever before. This has raised the need to scale up the unsupervised learning methods for SCS datasets, most of which aren't designed for such large datasets. Recent work has used a variant of non-negative matrix factorization (called Sampled Matrix Factorization) to estimate the underlying true transcriptomic states from sampled observed data utilizing the underlying low rank of these datasets. While there has been a lot of work on developing scalable and online approaches to non-negative matrix factorization, most of these approaches cannot be directly extended to Sampled Matrix Factorization. Here we focus on identifying potential optimization tools to scale up Sampled Matrix Factorization for large datasets.

1 Introduction

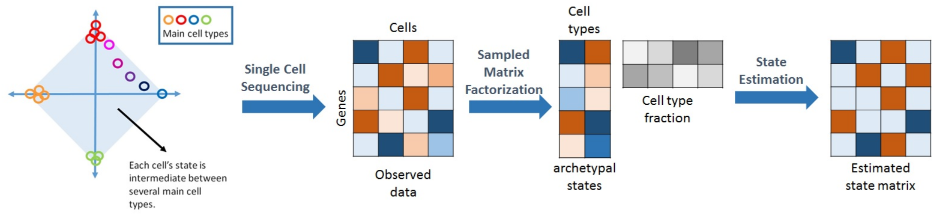


Figure 1: Illustration of the state estimation process using Sampled Matrix Factorization

A recently surge in development of low-cost large scale single cell sequencing technologies [6,8,10] have given rise to many large scale single cell transcriptomic datasets. This has led to the development of many bioinformatic tools to uncover biological insights about changes in gene expressions during differentiation [7], identification of novel differential regulator genes [14] and new cell types [15] etc. The main advantages gained from SCS can are: 1) they have large sample sizes which can improve the accuracy of computational methods designed to study them, and 2) high granularity information about each single cell helps avoid aggregation of traits leading to better information about individual cub-groups in these datasets [3].

A common assumption for most general purpose unsupervised learning learning algorithms is the Gaussian-ness of the data. Several commonly made algorithms such as PCA, tSNE, K-means clustering etc. either implicitly or explicitly assume that the data belongs to either Gaussian or t-distribution (or similar smooth continuous distributions). This assumption is however not true for SCS datasets, where the observed data is typically count valued and closer to Poisson or Negative Binomial distribution instead. This is because models of the sequencing process [1, 12] suggest that a cell's true transcript counts are sampled using one of many discrete sampling distributions to obtain the cell's observed transcript count. This affects the accuracy of the various general purpose learning approaches for these datasets.

A potential approach to overcome this issue would be to estimate the true transcriptomic state of the cells from the observed SCS data. Unfortunately, this problem is computationally infeasible even with a known sampling distribution for the case when no additional information is available. However, in case of SCS data, since cells have a few small number of cell types, the true transcriptomic states can be assumed to be low rank, with the rank bounded by the number of true cell types in the data. In our recent work we have developed Sampled Matrix Factorization [11] as a method to utilize this low rank structure to estimate the true transcriptomic state as seen in Figure 1. However, the current approach relies on general purpose optimization toolboxes to perform the constituent optimization steps. Unfortunately, this prevents the current method from being scaled up to very large datasets. For this class we have focused on exploring online optimization approaches to scale up the problem.

Several optimization routines have been discussed in class, namely Stochastic Gradient Descent (SGD) [4], Stochastic Variance Reduction (SVRG) [13], AdaGrad [5] etc. to tackle the large scale online learning problem. These problems have demonstrated benefits in the case of convex cost functions and also in the case of several non-convex cost functions. However, their relative performances vary based on the problem and here we have compared their efficiencies for our specific problem formulation. Morevoer, we have benchmarked these approaches against some 'batch' learning methods such as proximal gradient descent, mini-batch gradient descent and NoLips [2], which is a specialized algorithm for non-Lipschitz cost functions. In the coming sections we first formulate the Sampled Matrix Factorization problem and explain the different results obtained from our experiments.

2 Sampled Matrix Factorization

The task of estimating the true transcriptomic state works under the assumption that the true state of the cells lie in the convex hull spanned by the states of the main cell types [11]. For this problem, we assume that we have been provided with a matrix of initial means $M \in \mathbf{R}^{genes \times k}$ and a data matrix $X \in \mathbf{R}^{genes \times cells}$. The Sampled Matrix Factorization model assumes that the observed transcriptomic state of each cell is a discrete sampled version of the true state (this can be any sampling distribution but here we have focused on the Poisson distribution, since it is a common assumption for SCS datasets) i.e.

$$X \sim SamplingDistribution(X_{true}), \quad \textit{where } X_{true} = M \times w$$

X_{true} is the matrix of true transcriptomic states of all cells (this is hidden from us) and $w \in \mathbf{R}^{k \times cells}$ is the cell type fraction which statisfies the property $\mathbf{1}^T w_i = 1$ and $w_i \succeq 0$. These conditions ensure that each cell's original state will lie in the convex hull of the cell states of all the different cell types. The goal is to find the optimal M and w matrices which maximizes the log-likelihood of the observed data matrix X. Here we note that this problem is unfortunately non-convex for most sampling distributions but the sub-problems of estimating either M or w with the other matrix fixed are convex problems for any exponential family distribution. We thus adopt an algorithm similar to EM or coordinate descent to estimate these model parameters. In the first step we estimate the mixture parameter while keeping the means fixed as follows:

$$\begin{aligned} w = \arg\min_{w} & \, log(Prob(X|M, w, \Theta)) \\ & \textit{subject to} \\ & w_i \succeq 0, \quad \forall i \in [0, 1, ..N] \end{aligned}$$

Here the cost function is used to describe the log-likelihood of the observed data given the matrices M, w and any additional statistical parameters Θ. The parameter Θ is distribution dependent and is the empty set in case of the Poisson distribution. Having performed this step, the following step is to estimate M by solving the following optimization problem:

$$\begin{aligned} M = \arg\min_{w} log(Prob(X|M, w, \Theta)) \\ \textit{subject to} \\ [M]_{ij} \geq 0, \quad \forall i, j \end{aligned}$$

The condition $[M]_{ij} \geq 0$ is needed to guarantee that the true transcriptomic state cannot be negative. These two steps are then performed iteratively till convergence.

Once converged, the columns of w are normalized in order to sum to 1 to ensure the condition $\mathbf{1}^T w_i = 1$ is satisfied. This is not enforced during the optimization steps to ensure that cells with similar transcriptomic profiles but different cell sizes (thereby larger number of total transcripts) can converge to their respective optimal mixing weights first. The post normalization step then ensures cells with different cell sizes but with similar transcriptomic expression patterns end up having similar estimated states. In the case of Poisson distribution, this process can be summarized as below in Algorithm 1.

Algorithm 1 Sampled Matrix Factorization (Poisson)

```
function ESTIMATE-STATE($X, M, k, maxiters, \epsilon$)
    $W \leftarrow$ initial (random-positive) matrix of size $k \times cells$
    for $iter \leftarrow 1...maxiters$ do
        Update $W$ to minimize $\mathbf{1}^T(MW - X \circ \log(MW))\mathbf{1}$ , subject to $W$ nonnegative
        Update $M$ to minimize $\mathbf{1}^T(MW - X \circ \log(MW))\mathbf{1}$ , subject to $M$ nonnegative
        if $M$ and $W$ changed less than $\epsilon$ this iteration then
            return $M, W$
        end if
    end for
    for $iter \leftarrow 1...cells$ do
        $W[:, i] = \frac{W[:,i]}{sum(W[:,i])}$
    end for
    return $M, W$
end function
```

For this class project we omit the last normalization step since it is application specific and is computationally cheap to calculate.

3 Optimization methods tested

We implemented six optimization approaches for updating the matrices.

- *Gradient Descent* is the most straightforward approach: we simply find the direction of the likelihood's gradient and adjust the row or column in that direction.
- *Stochastic Gradient Descent* approximates the gradient using a single data point (a single entry in our observed data matrix in our case).
- *Stochastic Gradient Descent with mini-batching* approximates the gradient using a small batch of data points (100 in our case).
- *SVRG* (Stochastic Variance Reduced Gradient) attempts to reduce the inherent variance in stochastic gradient descent. We store a snapshot of the weight vector after every m iterations, which approximates the current weight vector. Each time we store the snapshot of the weight vector, we also compute the average gradient over the snapshot weights.

$$\tilde{\mu} = \Delta P(\tilde{w}) = \frac{1}{n} \sum_{i=1}^{n} \Delta \psi_i(\tilde{w})$$

Then, when we update on a new data point, we adjust the point's gradient by our estimate (based on the snapshot weights) of how this point's gradient differs from the average gradient (on the snapshot weights).

$$w^{(t)} = w^{(t-1)} - \eta_t(\Delta\psi_i(w^{(t-1)}) - \Delta\psi_{i_t}(\tilde{w}) + \tilde{\mu})$$

- *AdaGrad* is an adaptive gradient method which uses feature-specific learning rates which incorporate the past geometry of the data.
- *NoLip* is a specialized algorithm for non-Lipschitz functions.This method utilizes a different defintion of smoothness using Bregman distance and demonstrates that for certain classes of non-Lipschitz functions (including our example) it is possible to construct gradient update rules with sub-linear convergence rates.

4 Results

4.1 Convergence rate and run time comparison of different methods

We benchmark the various algorithms on synthetic data sets of sizes 100, 500 and 1000. In each case, the optimization procedure involves first using 5 passes through the data to optimize for W and then using 5 passes through the data to optimize for M. We created plots tracking the cost function against the number of iterations and the runtime. A few representative plots are included (Figures 2-5).

4.2 Discussion

In terms of the number of iterations, the stochastic methods (including SGD, SVRG, and AdaGrad) generally converged faster. SGD reached good results after just several iterations, which was helped using an adaptive method of selecting step sizes. Overall, the stochastic methods also seemed to convege to better cost-function values, which was interesting.

Surprisingly, regular SGD seemed to do slightly better than SVRG (which explicitly attempts variance reduction) and AdaGrad (which implements feature-specific learning rates). Part of this could simply be due to our choice of learning rates, and because our implementation of SGD already adjusts learning rates per feature, based on their success in decreasing the cost. It also seems like the data is well-behaved enough that variance reduction becomes less necessary.

However, in terms of the runtime, the picture is much more mixed. Even though the stochastic methods converge in a small number of iterations, each pass through the data set takes much longer with the stochastic methods. This may be because the batch methods take advantage of large matrix operations when computing gradients, which are more optimized on a hardware level than many small operations. On the other hand, the stochastic methods involve for loops, which can be very slow. The NoLip method seems to converge in the shortest amount of time, and ordinary Gradient Descent also does well. However, it seems like all the algorithms perform similarly against runtime.

An advantage of the stochastic methods is that they do not require storing as much information in memory. While gradient descent requires that the entire W matrix and an entire row of the data matrix be stored in memory to compute one gradient, stochastic gradient descent only requires one column of W and one entry of the data matrix. This is a major improvement, which is important when the datasets become large enough that it is impossible to store the matrices in memory.

4.3 Effect of number of passes through dataset per iteration

At first, all of the algorithms made one pass of the dataset when updating M, then switched to updating W for one pass, and so on. We experimented with changing the number of passes through the data set of updating M before switching to updating W, and so on. Figure 6 contains a plot with a 1000 x 1000 matrix (and SGD). We plotted it so that the x-axis is proportional to the number of passes through the dataset, to avoid giving an advantage to methods that do more work within one update to M/W.

Overall, the effect of changing the number of passes per iteration was minimal. It seems like performing 5 or 10 passes (instead of a single pass) through the data set for each update to M/W yields

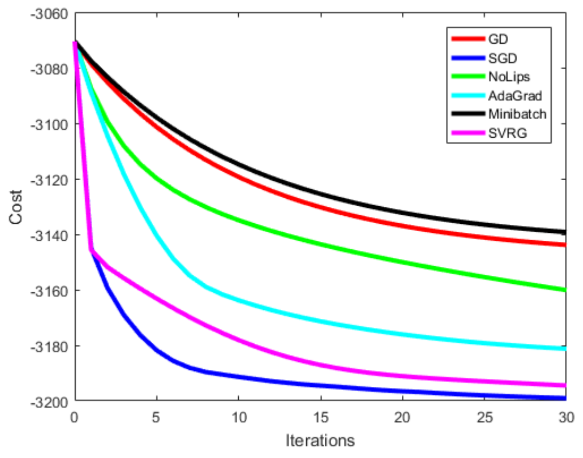


Figure 2: Cost vs # iterations (100 x 1000)

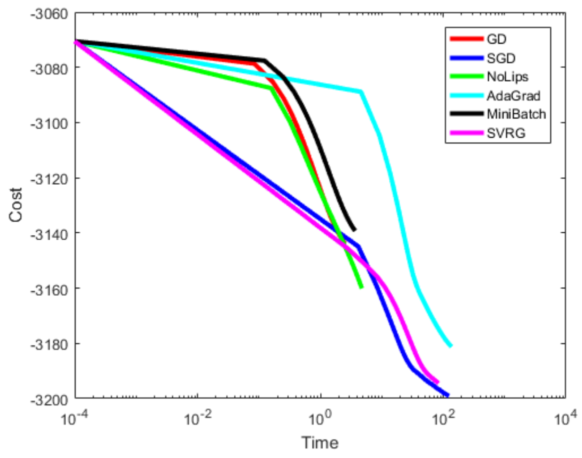


Figure 3: Cost vs runtime (100 x 1000)

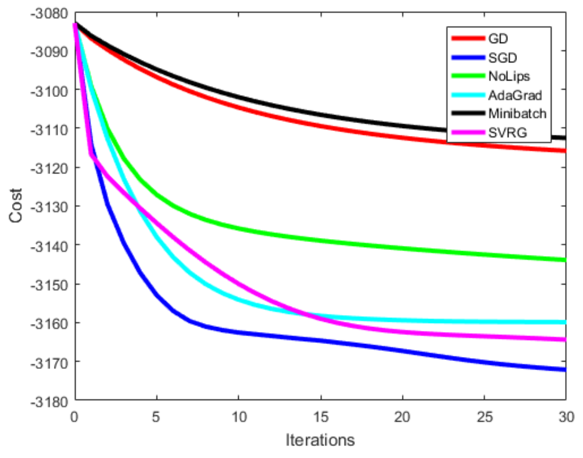


Figure 4: Cost vs # iterations (1000 x 1000)

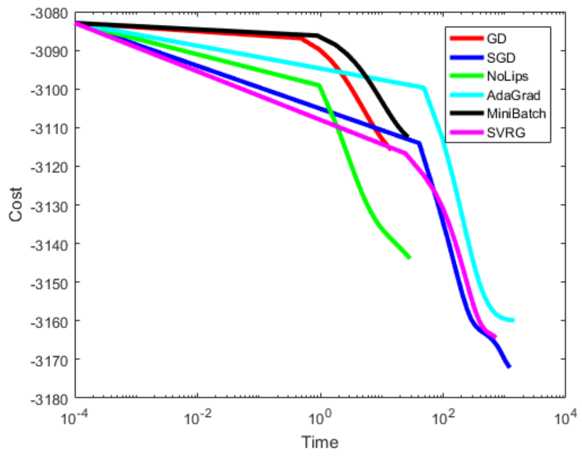


Figure 5: Cost vs runtime (1000 x 1000)

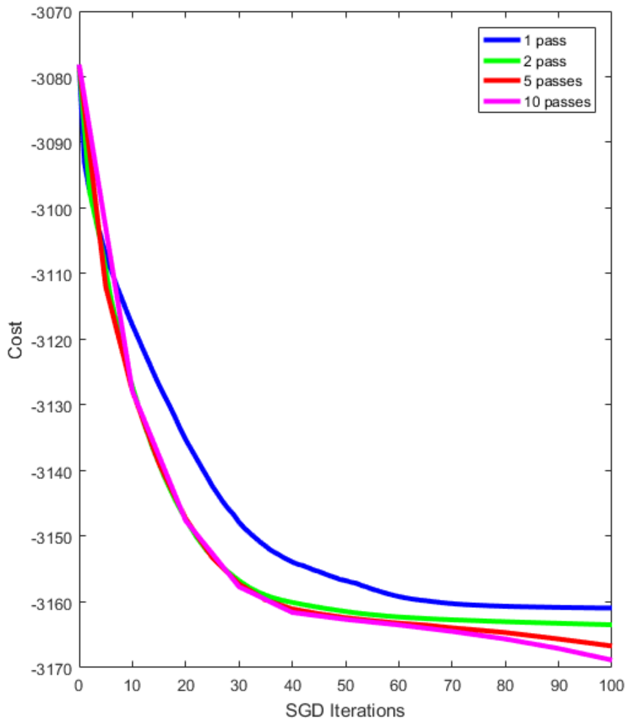


Figure 6: Experimenting with number of updates

a slightly faster convergence rate, but going beyond 5 passes does not seem to produce much further improvement. This could be because if we want to update W in a useful way, we need a good estimate of M (which is fixed), but we don't want to waste too much time finding the optimal value of W if the value of M isn't optimal yet.

4.4 Effect of adaptive step size

(Udell et al, 2016) suggests a more nuanced rule for selecting step sizes that allows for different step sizes per row/column. Every time we update a row or column, we check if the objective function (cost) increased or decreased. If the cost decreased, we are moving in the right direction, so we can increase the step size (the paper recommends by 5%). However, if the cost increased, we jumped too far, so we should decrease the step size for this row/column (the paper recommends by 30%).

We implemented this modification for SGD and SVRG. As shown in Figures 7-8, using this adaptive method yields faster convergence to a better cost-function value (for both SGD and SVRG) than sticking with a fixed step size throughout. This makes sense because the approach automatically adjusts and customizes step sizes for each row/column, instead of imposing the same step size on everything. This optimization was thus used in the above comparison of methods.

5 Applications: Dimensionality reduction

To test how using scalable optimization routines affects the downstream unsupervised learning methods, we look at the specific example of dimensionality reduction. The dimensionality reduction method as described in [11] assumes makes explicity use of the estimated mean matrix M and a convex weight matrix w. This is done by first calculating the distances between the means as follows:

$$[D]_{ij} = d(M_i, M_j)$$

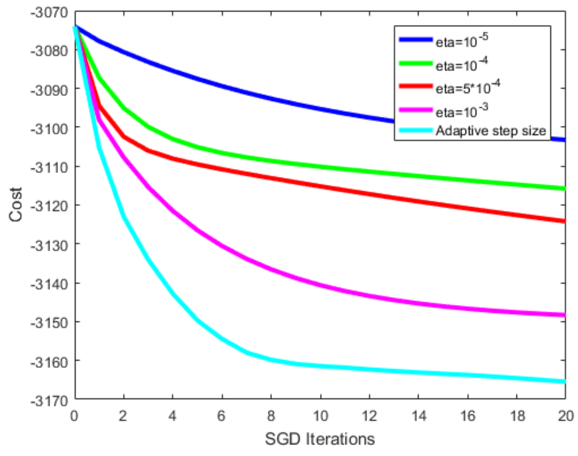


Figure 7: SGD step size comparison (1000 x 1000)

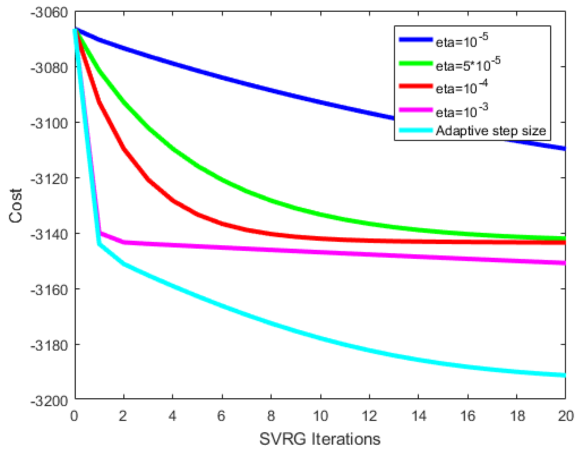
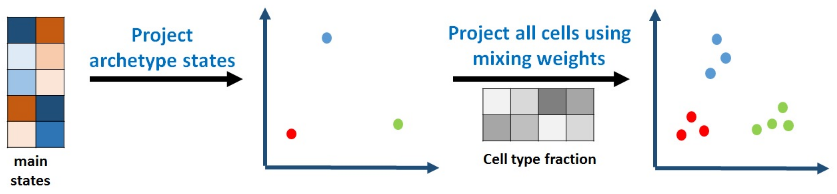
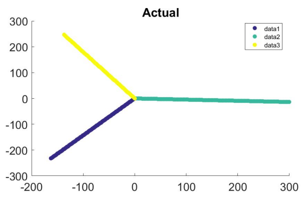


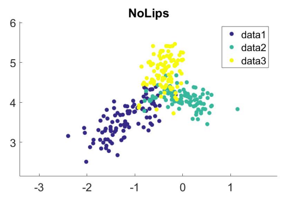
Figure 8: SVRG step size comparison (100 x 1000)



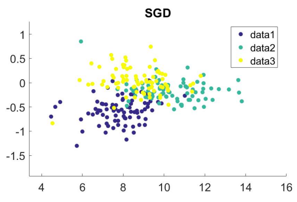
(a)



(b)



(c)



(d)

Figure 9: a) Illustration of the dimensionality reduction using Sampled Matrix Factorization. b) True low dimensional manifold (pre-sampling). c) Low dimensional visualization inferred using NoLips algorithm. d) Low dimensional visualization inferred using SGD.

Multi-dimensional Scaling (MDS) [9] is then used to convert the high dimensional means to the desired lower dimension while preserving the distances between the points. Since, the number of cell type is a lot smaller than the number of cells, this leads to a more faithful representation of the underlying manifold. The lower dimensional representation of all cells X^{LD} is then obtained as follows:

$$X^{LD} = M^{LD} \times w$$

The outline of this method is seen in Figure 9 a. This method of dimensionality reduction forces the relative positions of the points to be the same as that of the high dimension and has been used in [11] to obtain efficient visualization of heterogeneous biological samples and lineage estimation. Here we focus on the comparison of performing this method on an online and batch setting.

To perform the previously described test, we use the tree structured synthetic dataset that was used in [11]. This dataset comprises of high dimensional data (1000 genes and 300 cells), which has an underlying tree structured manifold as seen in Figure 9 c. This 'true transcriptomic state' data is then sampled using a Poisson distribution to obtain the observed data. We then perform sampled matrix factorization with k = 3 to estimate M and w. These are then used to perform dimensionality reduction to visualize the cells in 2 dimensions. We perform this using the NoLips algorithm in the batch setting and SGD in the online setting (with $\eta = 10^{-5}$). We see that while the dimensionality reduction using NoLips leads to a visualization which is clearly similar to the underlying true manifold structure (albeit more noisy), that obtained using SGD is more noisy but the manifold structure is still somewhat discernable. Hence, there still needs to be more work done on selecting appropriate learning rates for the online setting to improve the run time and the final cost function value, which can lead to similar (or better) performances in downstream learning tasks.

6 Conclusion

In this work we have explored different online optimization schemes to perform Sampled Matrix Factorization and benchmarked the results against 'batch' based approaches. We demonstrate on several synthetic datasets that it is possible to converge to better cost function values using online methods, with fewer passes through the datasets. Such methods do take slightly longer per iteration because of the fixed costs of for loops, but they have the advantage of not needing to keep large amounts of data in memory.

We then compared different number of passes through the dataset per iteration of online algorithms and demonstrate that increasing number of passes per iteration leads to minor improvements in the final cost function value. We also compare the effect of adaptive step size selection on different online methods. We demonstrate that this can be largely beneficial and can lead to dramatic improvements in both convergence rates and final cost function values.

Finally, we demonstrate the usability of these algorithms on a downstream learning problem, namely dimensionality reduction. We demonstrate that although batch methods currently outperform the online approaches, they are still able to estimate the underlying manifold structure of the data from noisy sampled observed data. Future work will be aimed at developing automating the selection of learning rates for various datasets and exploring custom online methods suited for this class of problems.

References

[1] S. Anders and W. Huber. Differential expression analysis for sequence count data. *Genome biology*, 11(10):1, 2010.

[2] H. H. Bauschke, J. Bolte, and M. Teboulle. A descent lemma beyond lipschitz gradient continuity: first-order methods revisited and applications. *Mathematics of Operations Research*, 2016.

[3] C. R. Blyth. On simpson's paradox and the sure-thing principle. *Journal of the American Statistical Association*, 67(338):364–366, 1972.

[4] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[5] C. Dyer. Notes on adagrad. *School of Computer Science, Carnegie Mellon University*, 5000.

[6] D. Grün and A. van Oudenaarden. Design and analysis of single-cell sequencing experiments. *Cell*, 163(4):799–810, 2015.

[7] N. K. Hanchate, K. Kondoh, Z. Lu, D. Kuang, X. Ye, X. Qiu, L. Pachter, C. Trapnell, and L. B. Buck. Single-cell transcriptomics reveals receptor transformations during olfactory neurogenesis. *Science*, 350(6265):1251–1255, 2015.

[8] A. M. Klein, L. Mazutis, I. Akartuna, N. Tallapragada, A. Veres, V. Li, L. Peshkin, D. A. Weitz, and M. W. Kirschner. Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. *Cell*, 161(5):1187–1201, 2015.

[9] J. B. Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129, 1964.

[10] E. Z. Macosko, A. Basu, R. Satija, J. Nemesh, K. Shekhar, M. Goldman, I. Tirosh, A. R. Bialas, N. Kamitaki, E. M. Martersteck, et al. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell*, 161(5):1202–1214, 2015.

[11] S. Mukherjee, Y. Zhang, S. Kannan, and G. Seelig. Prior knowledge and sampling model informed learning with single cell rna-seq data. *bioRxiv*, page 142398, 2017.

[12] E. Pierson and C. Yau. Zifa: Dimensionality reduction for zero-inflated single-cell gene expression analysis. *Genome biology*, 16(1):1, 2015.

[13] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola. Stochastic variance reduction for nonconvex optimization. *arXiv preprint arXiv:1603.06160*, 2016.

[14] C. Trapnell, D. Cacchiarelli, J. Grimsby, P. Pokharel, S. Li, M. Morse, N. J. Lennon, K. J. Livak, T. S. Mikkelsen, and J. L. Rinn. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature biotechnology*, 32(4):381–386, 2014.

[15] A. Zeisel, A. B. Muñoz-Manchado, S. Codeluppi, P. Lönnerberg, G. La Manno, A. Juréus, S. Marques, H. Munguba, L. He, C. Betsholtz, et al. Cell types in the mouse cortex and hippocampus revealed by single-cell rna-seq. *Science*, 347(6226):1138–1142, 2015.