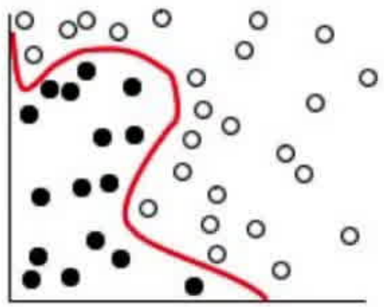
Stochastic Gradient Descent, in theory and practice

Rachel Ward

University of Texas at Austin
Department of Mathematics
The Oden Institute for Computational Engineering and Sciences
2017-2018: Facebook AI Research (FAIR)

SIAM CSE; Spokane, Washington
February, 2019

Optimization in Modern Machine Learning



Neural networks are functions of a particular parameterized form modeled after the human brain which interpolate or approximate a given set of *training examples* $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ (supervised setting).

Why are neural networks so powerful and versatile?

- Approximation power + simple form of neural networks
- **Modern computer hardware** and **simple optimization methods** like Stochastic Gradient Descent learn parameters of neural networks quickly

Where do we stand regarding theory vs practice?

Optimization in Modern Machine Learning

Given training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, find a vector of *weights* $\mathbf{w}$ which minimizes a loss function

$$F(\mathbf{w}) = \frac{1}{n} \sum_{j=1}^{n} (f(\mathbf{w}; \mathbf{x}_j) - y_j)^2,$$

where $f(\mathbf{w}; \mathbf{x})$ is a neural network function of depth d with ReLU activation function σ:

$$f(\mathbf{w}; \mathbf{x}) = \mathbf{w}_d^T \sigma \left(\mathbf{w}_{d-1}^T \sigma(\ldots \sigma(\mathbf{w}_1^T \mathbf{x})) \right), \qquad \sigma(\mathbf{x})_i = \max\{0, x_i\}.$$

Dimensions are large – for example, data dimension is 10 million, parameter dimension is 20 million.

For such high-dimensional problems, we must use first-order iterative optimization methods. Chain rule gives a simple closed-form expression for $\nabla F(\mathbf{w})$ – *back propagation*.

Stochastic Gradient Descent (SGD)

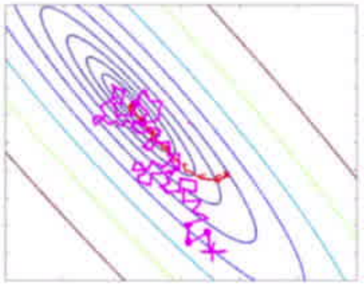
Minimize $F(\mathbf{w}) = \frac{1}{n}\sum_{j=1}^{n} f_j(\mathbf{w})$

- When n is large, even computing a single gradient $\nabla F(\mathbf{w})$ is too costly. Gradient descent is too costly.
- Cheaper to compute the gradient in a single (or batch of) component directions $\nabla f_i(\mathbf{w})$ at each iteration as a surrogate for $\nabla F(\mathbf{w})$.

SGD (Robbins, Monro 1951):

- **Initialize** step-size $\alpha > 0$ and $\mathbf{w}^{(0)} \in \mathbb{R}^d$;
- **Until** convergence,

$$t + 1 \leftarrow t$$
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla f_{i_t}(\mathbf{w}^{(t)})$$

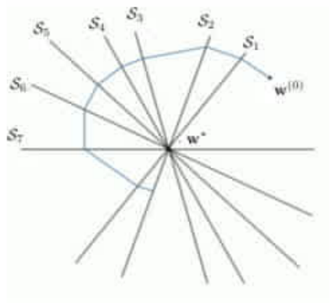


Choose index i_t at random so that $\mathbb{E}_{i_t} \nabla f_{i_t}(\mathbf{w}) = \nabla F(\mathbf{w})$

Stochastic Gradient Descent and Linear Systems

Special Case: Solving an overdetermined consistent system of linear equations. $F(\mathbf{w}) = \frac{1}{2}\|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2$, and $\exists \mathbf{w}^*$ s.t. $\mathbf{A}\mathbf{w}^* = \mathbf{b}$.

$$\mathbf{A} = \begin{pmatrix} - & \mathbf{a}_1 & - \\ - & \mathbf{a}_2 & - \\ - & \mathbf{a}_3 & - \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ - & \mathbf{a}_n & - \end{pmatrix}$$

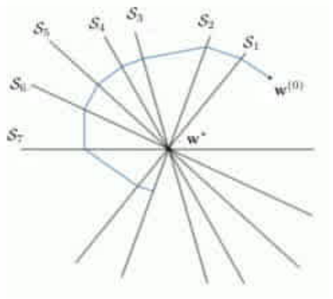


In this case, convex sets $\mathcal{S}_i = \{\mathbf{w} : \nabla f_i(\mathbf{w}) = 0\}$ are *affine*, and SGD as an iterative *projection onto convex sets* (POCS) algorithm with random order of selection

Stochastic Gradient Descent and Linear Systems

Special Case: Solving an overdetermined consistent system of linear equations. $F(\mathbf{w}) = \frac{1}{2}\|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2$, and $\exists \mathbf{w}^*$ s.t. $\mathbf{A}\mathbf{w}^* = \mathbf{b}$.

$$\mathbf{A} = \begin{pmatrix} - & \mathbf{a}_1 & - \\ - & \mathbf{a}_2 & - \\ - & \mathbf{a}_3 & - \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ - & \mathbf{a}_n & - \end{pmatrix}$$



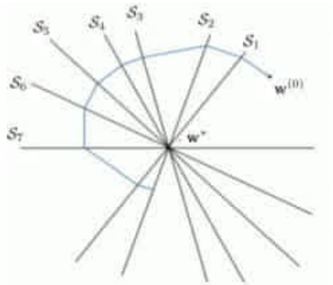
In this case, convex sets $\mathcal{S}_i = \{\mathbf{w} : \nabla f_i(\mathbf{w}) = 0\}$ are *affine*, and SGD as an iterative *projection onto convex sets* (POCS) algorithm with random order of selection

Linear convergence of POCS with cyclic selection first studied by von Neumann (1933).

For linear systems: convergence to least squares solution in noisy setting. **Convergence to least-norm solution in underdetermined setting.**

Stochastic Gradient Descent and Linear Systems

$$\mathbf{A} = \begin{pmatrix} - & \mathbf{a}_1 & - \\ - & \mathbf{a}_2 & - \\ - & \mathbf{a}_3 & - \\ & \vdots & \\ & \vdots & \\ & \vdots & \\ - & \mathbf{a}_n & - \end{pmatrix}$$



For linear systems, condition number governing linear convergence rate of SGD is $\kappa = \frac{L}{\mu} = n \max_i \|\mathbf{a}_i\|^2 \|(\mathbf{A}^T\mathbf{A})^\dagger\|_2$.

[Strohmer Vershynin, 2007] If row $\mathbf{a}_i$ is chosen at random proportionally to $\|\mathbf{a}_i\|_2^2$ instead of uniformly and step-size is preconditioned, the linear convergence rate can be improved to $\kappa' = \|\mathbf{A}\|_F^2 \|(\mathbf{A}^T\mathbf{A})^\dagger\|_2$.

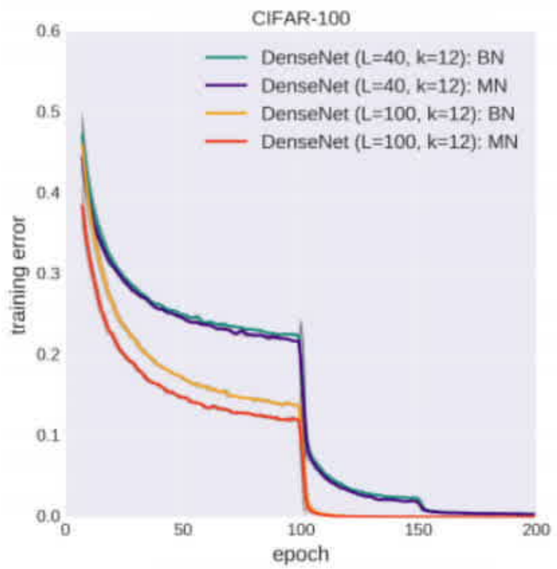
[Needell, Srebro, W 2013] More generally, sampling component functions in SGD at random proportionally to their Lipschitz constants (importance sampling), linear convergence rate of SGD improves from $\kappa = \frac{\max_i L_i}{\mu}$ to $\kappa' = \frac{\frac{1}{n}\sum_i L_i}{\mu}$.

From linear systems to neural networks

Back to the neural network loss function:

$$F(\mathbf{w}) = \frac{1}{n}\sum_{j=1}^{n}(f(\mathbf{w},\mathbf{x}_j)-y_j)^2, \quad f(\mathbf{w},\mathbf{x}) = \mathbf{w}_d^T\sigma\left(\mathbf{w}_{d-1}^T\sigma(\ldots\sigma(\mathbf{w}_1^T\mathbf{x}))\right)$$

A plot of SGD convergence in the *overparameterized* setting:



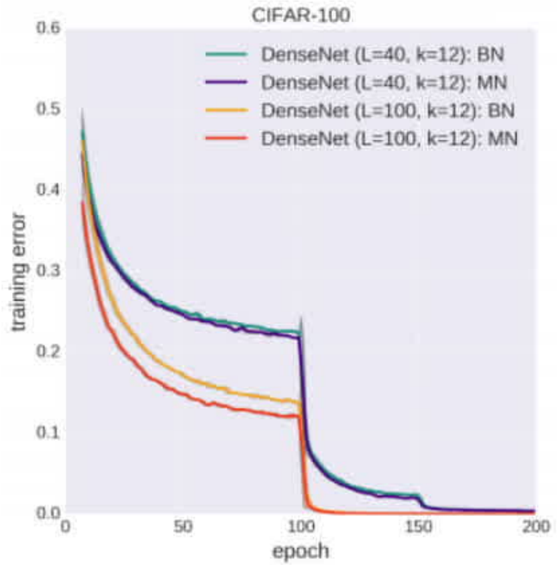
Note *the training error goes to zero.* In other words, the component functions share a minimizer $\mathbf{w}^*$ and SGD converges to such a minimizer.

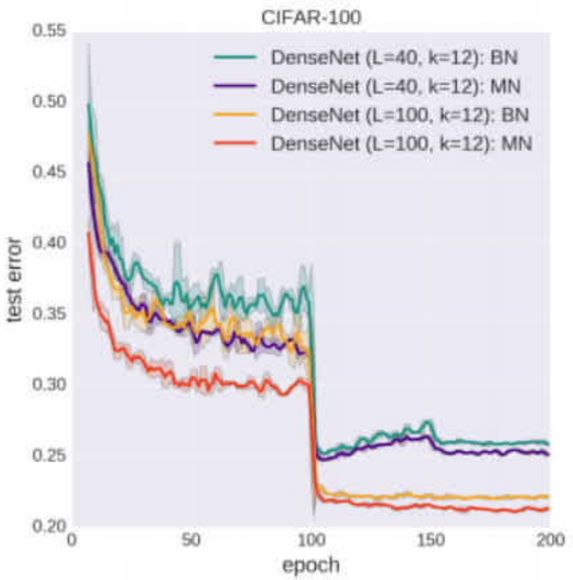
This contradicts conventional wisdom that optimizing such functions is NP-hard.

Plot from [Kalayeh, Shah, 2018]

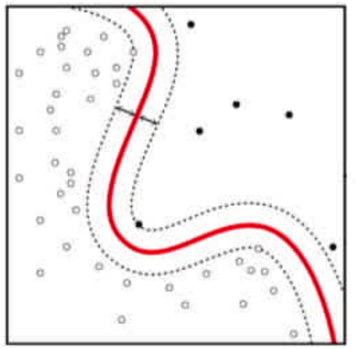
From linear systems to neural networks

A plot of SGD *training* vs *testing* error in the *overparameterized* setting:





Not only does SGD converge to a global minimizer of the neural network loss function in over-parameterized setting, it converges to a *good* global minimizer – *implicit regularization* of gradient descent.



From linear systems to neural networks

Modern neural networks are *over-parameterized* if possible – the number of parameters is larger than the size of the training data. SGD as an optimization method for such neural networks is surprisingly similar to SGD as an optimization method for *underdetermined* linear systems.

Recent theoretical work in this direction:

- With polynomial overparameterization ratio and random initialization $\mathbf{w}^{(0)}$, SGD will provably converge to a global minimizer of certain neural networks, with linear convergence rate[1]. The optimization problem is *locally convex* in a neighborhood of $\mathbf{w}^{(0)}$ and some $\mathbf{w}^*$.
- For "shallow" neural networks[2], gradient descent provably converges to the minimum-norm solution $\mathbf{w}^*_{LS}$.

---

[1] [Soudry, Carmon, 2016], [Li, Yuan, 2017], [Du, Zhai, Poczos, Singh 2019]

[2] [Neyshabur, Tomioka, Salakhutdinov, Srebro 2017], [Gunasekar, Lee, Soudry, Srebro 2018], [Molitor, Needell, W 2019]

Part II: Making SGD robust to hyper-parameter tuning

"We want to design methods for machine learning that are not as ideal as Newton's method but have [these] properties: first of all, they tend to turn towards the right directions and they have the right length, [i.e.] the step size of one is going to be working most of the time ...".

Prof. Jorge Nocedal
IPAM Summer School, 2012

We are still quite far from this in theory and practice.

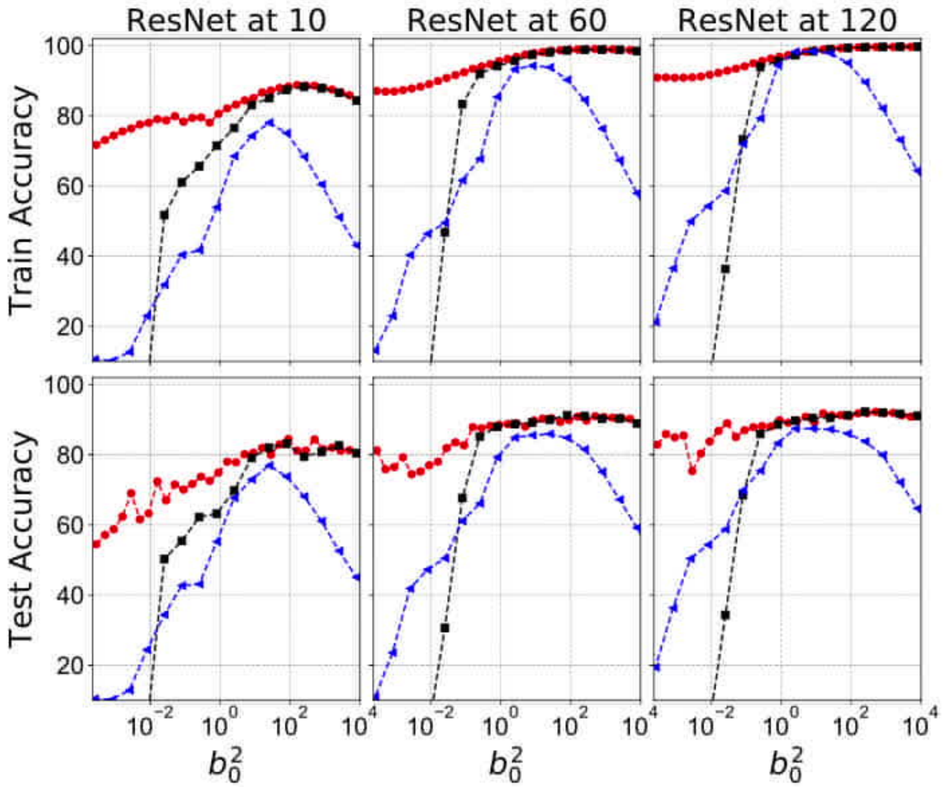
SGD with AdaGrad adaptive learning rate update

SGD update with AdaGrad adaptive learning rate update[3]: starting from $b_0 > 0$,

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \frac{\nabla f_{i_t}(\mathbf{w}^{(t)})}{\sqrt{\sum_{s=0}^{t}(b_0^2 + \|\nabla f_{i_s}(\mathbf{w}^{(s)})\|^2)}}$$

[3] [Duchi, Hazan, Singer 2011], [McMahan, Streeter, 2010]

Numerical illustration of robustness



Training and testing error on CIFAR10 dataset, after 10, 60, and 120 epochs over data, respectively.

Red- AdaGrad with $\alpha = 1$ starting from $\frac{1}{b_0}$. **Black** - SGD step-size $\frac{1}{b_0}$. Blue - SGD step-size $\frac{1}{b_0\sqrt{t}}$.

Conclusions

We motivated the use of *Stochastic Gradient Descent* as an optimization method for neural networks.

In the past several years, we are understanding why SGD performs so well in this context – *SGD for over-parameterized neural networks is similar to SGD for underdetermined linear systems*. Convergence to global minimizer, implicit regularization.

Still, SGD as implemented in practice is far from being *online*, involving costly and time-consuming hyper-parameter tuning.

Recent theoretical results on adaptive gradient methods such as AdaGrad justify what were previously heuristics for making SGD more robust to hyper-parameters such as step-size. However, other tricks such as *batch normalization* are even more powerful in practice, and these are not understood ...

Perspectives

Neural networks are powerful and versatile new tools in computational science, but they are just one tool.

Stochastic Gradient Descent is a useful algorithm beyond neural networks for *online* or *streaming* optimization. But for SGD to truly be a streaming algorithm, we must construct adaptive variations of SGD which are less sensitive to e.g. choice of step-size.

Several open questions remain. Generative adversarial networks (GANS) (unsupervised learning) are even more powerful in practice, but less well understood. SGD is even less robust in this setting.

Thank You.

Thanks also to AFOSR, NSF, Facebook AI Research for funding