Discussion of Optimization Techniques

Vahid Tarokh

ECE 685D, Fall 2025

Introduction

- We will quickly review optimization algorithms and SGD.
- Discuss more why they are preferred more than batch methods
- Discuss mathematical analysis for (and applications to) SGD
- Discuss extensions/variants (RMSprop, ADAgrad, ADAM)

- Important Note: Source of some of my slides (with great appreciation and acknowledgements)
  - Professor David Carlson Slides
  - Professor Ruslan Salakhutdinov's slides (available online).

Optimization Goal

- Have some model or network parameterized by $\boldsymbol{w}$
- Goal: given data, find the best $\boldsymbol{w}$
- What is the best $\boldsymbol{w}$?
  - Gives the best prediction (or other metric) on the *true* task
  - *True* task refers to future, unseen data (i.e. real-world performance)
  - Quick reminder: often estimate performance with a test set
- Many examples shown so far:
  - Image recognition
  - Object detection
  - Text classification
  - Etc.

Optimization Goal

- In optimization, this amounts to solving for $\boldsymbol{w}^*$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \mathbb{E}_{p_{true}(\mathbf{x},y)}[\ell(h_{\boldsymbol{w}}(\mathbf{x}), y)]$$

- $\ell(\cdot,\cdot)$ it the "loss," can be defined in many ways. Some examples:
  - squared loss: $\ell(\hat{y}, y) = ||\hat{y} - y||_2^2$
  - (binary) cross-entropy loss: $\ell(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$
  - Negative Log-Likelihood: $\ell(h_{\boldsymbol{w}}(\mathbf{x}), y) = -\log p_{\boldsymbol{w}}(\boldsymbol{x}, y)$
- $h_{\boldsymbol{w}}(\mathbf{x})$ defines a transformation from the data to the output space – only function that changes when the parameters do (e.g. in logistic regression $h_{\boldsymbol{w}}(\mathbf{x}) = \sigma(\boldsymbol{w}^T\mathbf{x})$)
- $p_{true}(\mathbf{x}, y)$ is the probability distribution over data (unknown!)

Empirical Risk Minimization

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \mathbb{E}_{p_{true}(\mathbf{x},y)}[\ell(h_{\boldsymbol{w}}(\mathbf{x}), y)]$$

- Issue: The probably distribution $p_{true}(\mathbf{x}, y)$ is unknown!
- But we have N data examples $\mathcal{D} = \{\mathbf{x}_n, y_n\} \sim p_{true}(\mathbf{x}, y)$
- Will approximate the above with finite data examples (i.e. "Empirical Risk Minimization" (ERM))

$$\min_{\boldsymbol{w}} \frac{1}{N} \sum_{n=1}^{N} \ell(h_{\boldsymbol{w}}(\mathbf{x}_n), y_n)$$

  - Why does this matter?

Why does using finite data matter?

- Using empirical samples is biased, leads to overfitting

Definition (Overfitting)

Overfitting occurs when the learned parameters $\boldsymbol{w}$ capture random error or noise. This implies the parameters $\boldsymbol{w}$ are "learning" the noise rather than properties of the data. Mathematically:

$$\mathbb{E}_{p(\mathbf{x},y)}[\ell(h_{\boldsymbol{w}}(\mathbf{x}), y)] > \frac{1}{N}\sum_{n=1}^{N} \ell(h_{\boldsymbol{w}}(\mathbf{x}_n), y_n)$$

or,

"generalization error" > "training error"

Consequences for Iterative Optimizers

- No benefit to exact optimization, only need "**moderate**" accuracy as fast as possible
- *modus operandi* in big data: use stochastic iterative methods
  - Less information per iteration, but many many more iterations in the same amount of time

- We assume $\boldsymbol{w} \in \mathbb{R}^D$, but can easily consider constrained set (most important thing is dimensionality D)
- The above optimization goal is often rewritten for simplicity:

$$\min_{\boldsymbol{w}} \frac{1}{N} \sum_{n=1}^{N} f_n(\boldsymbol{w}), \quad f_n(\boldsymbol{w}) = \ell(h_{\boldsymbol{w}}(\mathbf{x}_n), y_n)$$

Binary Logistic Regression

- A canonical model for classification is logistic regression:

$$P(y = 1|\mathbf{x}) = \sigma(\boldsymbol{w}^T\mathbf{x}), \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Why is this a good example for optimization?
  - This gives our logistic loss function
  - Can simply derive constants used in convergence analysis (allowing theorems to have *precise* values)

Introduction and Intuition of Stochastic Gradients

(Stochastic) Gradient Descent:

Gradient Descent (GD)

With step size α_k, use updates:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \left[\frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\boldsymbol{w}_k) \right]$$

Stochastic Gradient Descent (SGD)

With step size α_k and random index i_k, use updates:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \left[\nabla f_{i_k}(\boldsymbol{w}_k) \right]$$

Note that $\mathbb{E}_{p(i_k)} \left[\nabla f_{i_k}(\boldsymbol{w}_k) \right] = \left[\frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\boldsymbol{w}_k) \right]$ (unbiased). (Underlying i.i.d assumption in our dataset, $\mathcal{D}$)

How expensive are these iterative algorithms?

- Gradient descent takes:
  - $\mathcal{O}(N)$ to estimate gradients
  - $\mathcal{O}(1)$ to update the parameters
- Stochastic gradient descent takes:
  - $\mathcal{O}(1)$ to estimate gradients
  - $\mathcal{O}(1)$ to update the parameters
- What about Newton's method (classical optimization approach)?
  - $\mathcal{O}(N)$ to estimate gradients and $\mathcal{O}(ND^2)$ to estimate Hessian
  - $\mathcal{O}(D^3)$ to update the parameters

- Consider implication for GoogLeNet image classifier on the ImageNet dataset, with $N = 10^6$ and $D \simeq 5 \times 10^6$

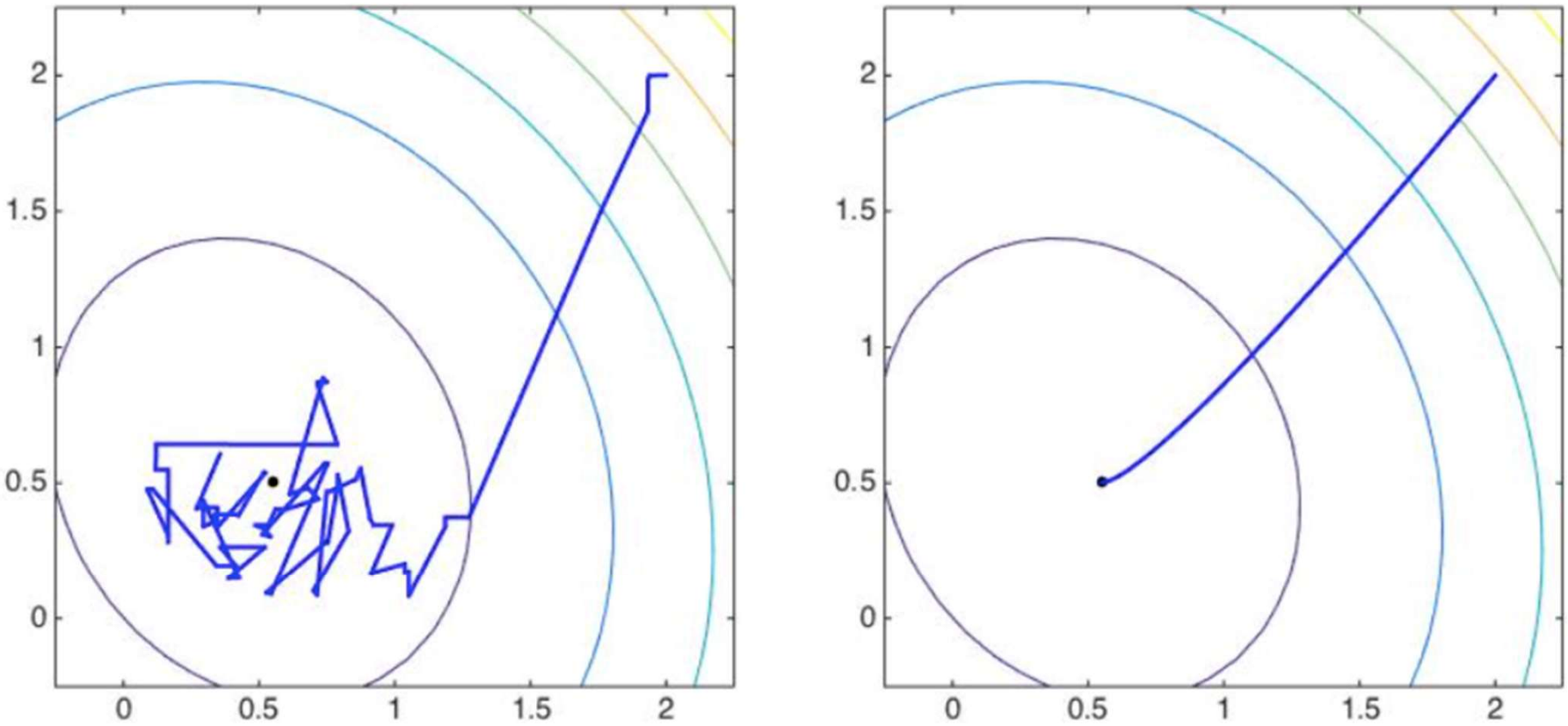


Figure: (Left) Stochastic gradient descent with a fixed step size. (Right) Batch gradient descent.

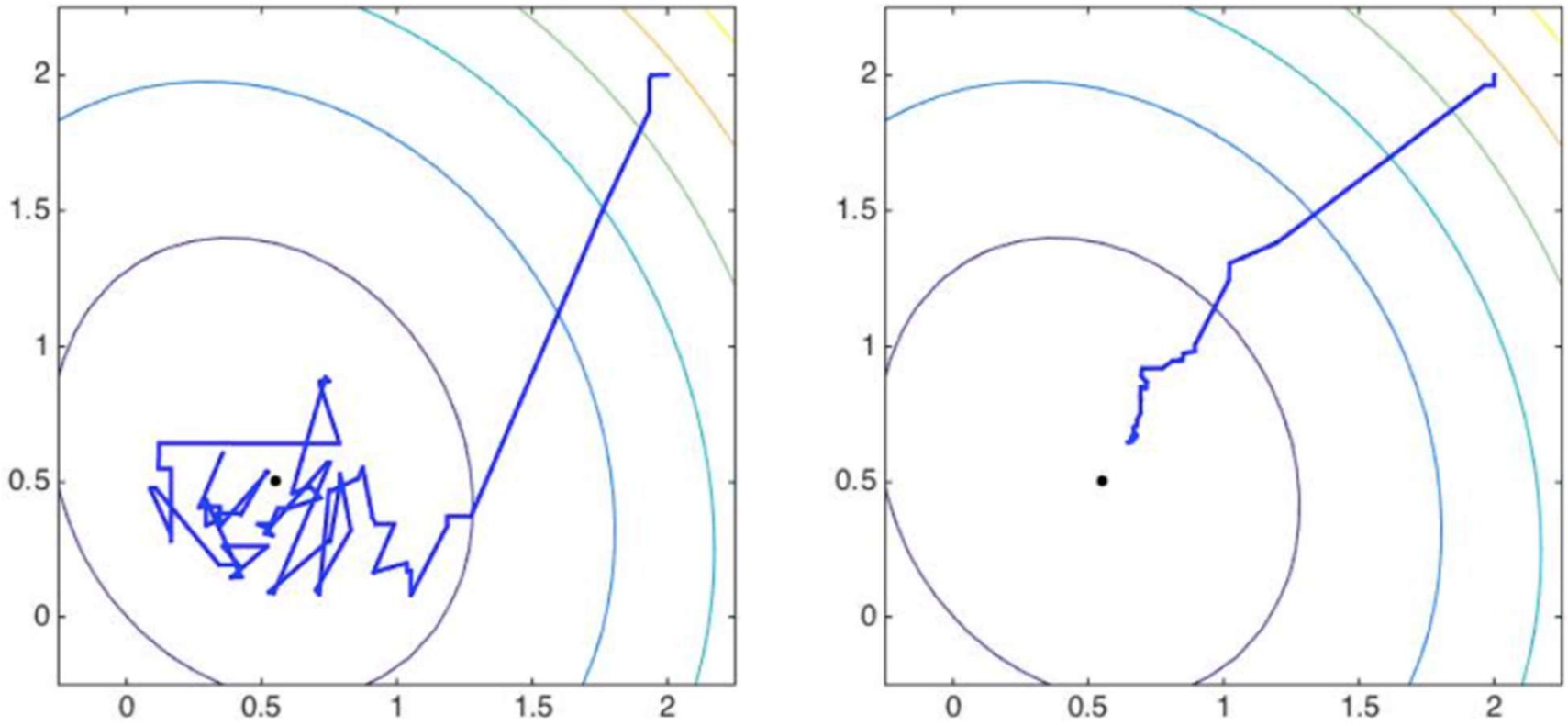


Figure: (Left) Stochastic gradient descent with a fixed step size. (Right) Stochastic gradient descent with diminishing step size.

When will Stochastic Gradient Descent work?

Necessary Definitions for Convergence Analysis

Definition (Lipschitz-continuous functions)

Given an objective function $F : \mathbb{R}^D \to \mathbb{R}$, the function is Lipschitz continuous if

$$|F(y) - F(x)| \leq L_0 ||\boldsymbol{y} - \boldsymbol{x}||_2$$

Necessary Definitions for Convergence Analysis

Definition (Lipschitz-continuous objective gradients)

Given an objective function $F : \mathbb{R}^D \to \mathbb{R}$ is continuously differentiable, the gradient of F is Lipschitz continuous with Lipschitz constant $L > 0$ if

$$||\nabla F(\boldsymbol{y}) - \nabla F(\boldsymbol{x})||_2 \leq L||\boldsymbol{y} - \boldsymbol{x}||_2.$$

For continuous second gradients, note $L \geq \max_{\boldsymbol{x}} ||\nabla^2 F(\boldsymbol{x})||_{S_\infty}$

- The matrix $||\cdot||_{S_\infty}$ norm is defined as the largest singular value of the matrix (also known as the spectral norm, the Schatten-∞ norm, and the matrix 2-norm):

$$||\mathbf{A}||_{S_\infty} = \max_{||\boldsymbol{x}||_2 \leq 1} ||\mathbf{A}\boldsymbol{x}||_2$$

Lipschitz Gradient for Logistic Regression

- Remember the cost function in logistic regression:

$$\min_{\boldsymbol{w}} F(\boldsymbol{w}) = \min_{\boldsymbol{w}} \frac{1}{N} \sum_{n=1}^{N} -y_n \log(\sigma(\boldsymbol{w}^T \mathbf{x})) - (1 - y_n) \log(1 - \sigma(\boldsymbol{w}^T \mathbf{x}))$$

- Hessian is given by

$$\nabla^2 F(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \left(\sigma(\boldsymbol{w}^T \mathbf{x}_n)(1 - \sigma(\boldsymbol{w}^T \mathbf{x_n})) \right) \mathbf{x}_n \mathbf{x}_n^T$$

Lipschitz Gradient for Logistic Regression

- First term is bound by:

$$0 \leq \Big(\sigma(\boldsymbol{w}^T\mathbf{x})(1-\sigma(\boldsymbol{w}^T\mathbf{x}))\Big) \leq \frac{1}{4}$$

- Linear algebra allows us to state:

$$\mathbf{0} \preceq \nabla^2 F(\boldsymbol{w}) \preceq \frac{1}{4N}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^T$$

($\mathbf{A} \preceq \mathbf{B}$ means that $\mathbf{B} - \mathbf{A}$ is a positive semidefinite matrix, e.g. $\boldsymbol{x}^T\mathbf{A}\boldsymbol{x} \leq \boldsymbol{x}^T\mathbf{B}\boldsymbol{x}$ for any $\boldsymbol{x}$ if matrices are symmetric)

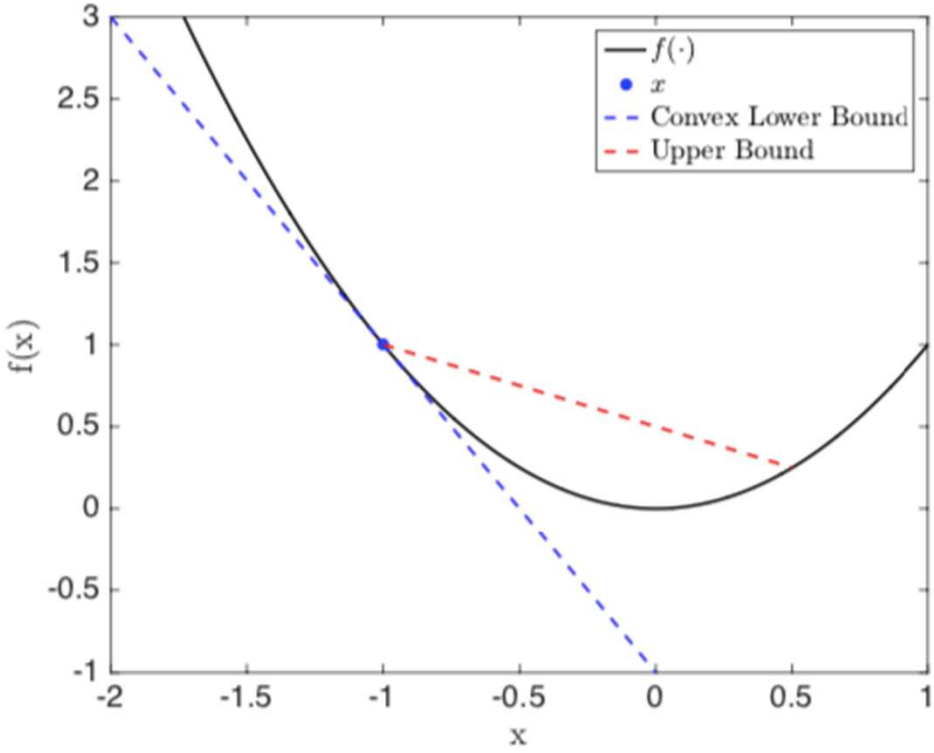
- A simple bound is $L \boxed{\geq} \frac{1}{4}\mathbb{E}_{\mathcal{D}}[||\mathbf{x}||_2^2]$

Convexity

- A convex function has two definitions:

$$\forall \mathbf{x},\ \mathbf{y} \in \mathcal{X}, t \in [0,1]$$
$$f(t\mathbf{y} + (1-t)\mathbf{x}) \leq tf(\mathbf{y}) + (1-t)f(\mathbf{x})$$
$$f(\mathbf{y}) \geq f(\mathbf{x}) + (\nabla f(\mathbf{x}))^T (\mathbf{y} - \mathbf{x})$$

- First definition is more general, gradient doesn't always exist



Strong Convexity

Definition (Strong Convexity:)

An objective function $F : \mathbb{R}^d \to \mathbb{R}$ is strongly convex in that there is a constant $c > 0$ such that

$$F(\boldsymbol{y}) \geq F(\boldsymbol{x}) + \nabla F(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2}c||\boldsymbol{y} - \boldsymbol{x}||_2^2$$

Then F will have a unique minimizer $\boldsymbol{x}_*$ with function value F_*. If only $c = 0$ holds, then the function is convex.

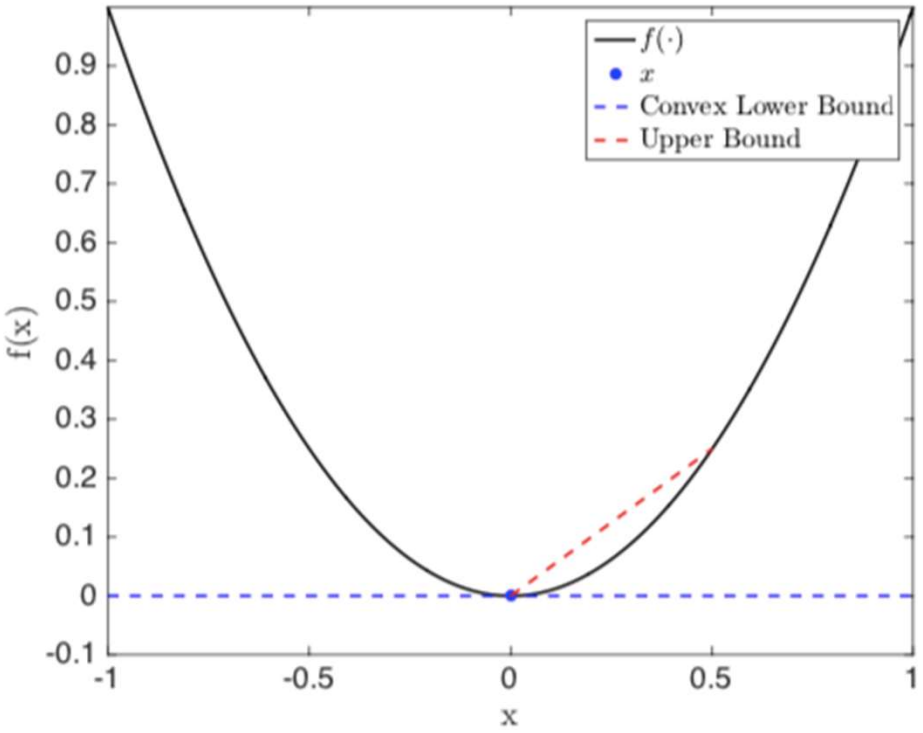
- This definition covers a significant subset of machine learning problems (e.g. penalized logistic regression, etc.)
- Does *not* cover models such as LDA, PFA, RBM, SBN, MLP, CNN, RNN, etc.
  - Gives clear theoretical results, and is very useful for intuitions in these problems

Implications

- Convex function implies a local minima is a global minima
- The gradient at the optima (if gradient exists)

$$\nabla f(\mathbf{x}) = \mathbf{0}$$

- Most state-of-the-art machine learning models are not convex
- However, very useful for intuition – nonconvex functions are often convex around a local minima



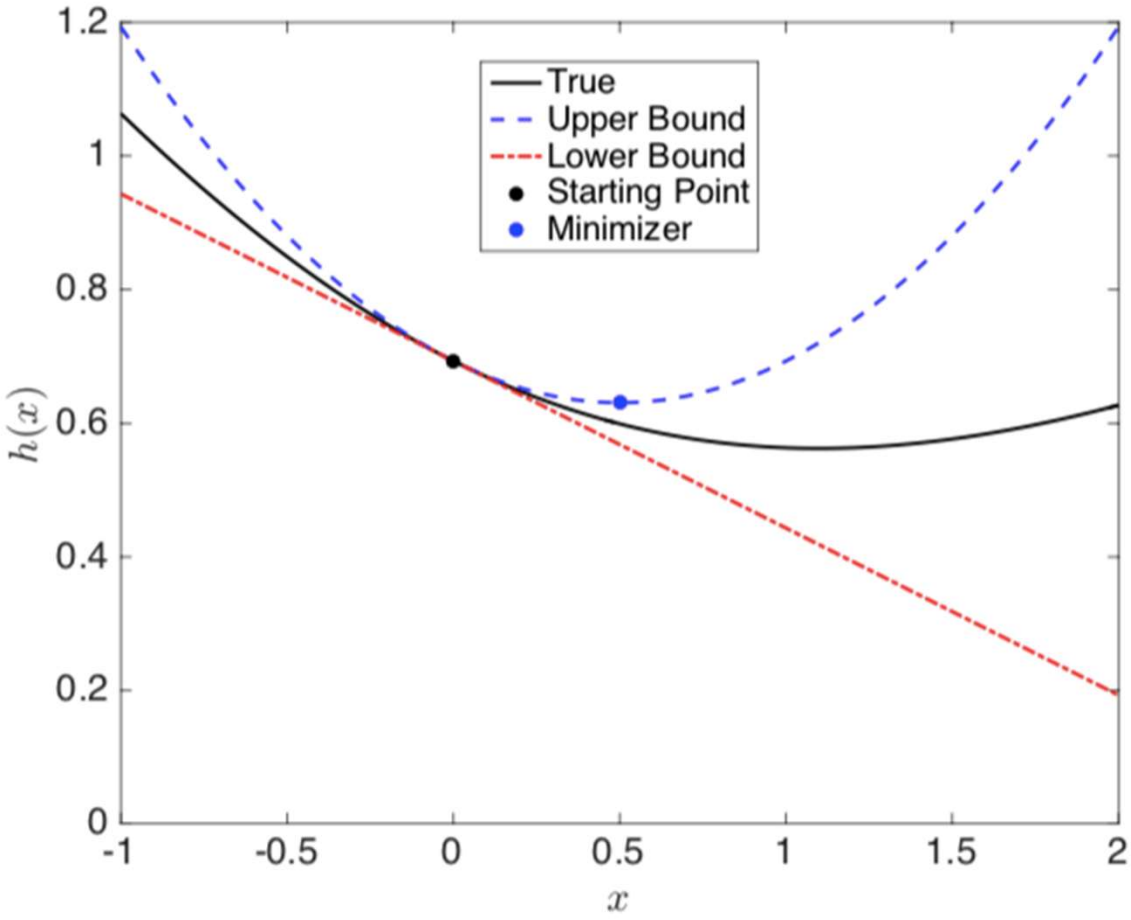


Figure: Examples of the upper bound on logistic loss. The Lipschitz gradient provides a conservative upper bound on the function value.

Consequences for Gradient Methods

Lemma (Decreasing sequence for gradient descent)

Consider a function F with Lipschitz gradient with constant L. For the deterministic (i.e. batch) gradient descent method with iterates

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha \nabla F(\boldsymbol{w}_k),$$

then the Lipschitz gradient upper bound with this sequence yields

$$\begin{aligned} F(\boldsymbol{w}_{k+1}) &\leq F(\boldsymbol{w}_k) - [\nabla F(\boldsymbol{x})]^T(\alpha \nabla F(\boldsymbol{w}_k)) + \frac{1}{2} L ||\alpha \nabla F(\boldsymbol{w}_k)||_2^2, \\ &\leq F(\boldsymbol{w}_k) - \alpha ||\nabla F(\boldsymbol{w}_k)||_2^2 + \frac{\alpha^2 L}{2} ||\nabla F(\boldsymbol{w}_k)||_2^2, \\ &\leq F(\boldsymbol{w}_k) + (\frac{L\alpha^2}{2} - \alpha) ||\nabla F(\boldsymbol{w}_k)||_2^2. \end{aligned}$$

"Optimal" Step Size

Lemma (Optimal step size for gradient descent)

Consider the upper bound for gradient descent

$$F(\boldsymbol{w}_{k+1}) \leq F(\boldsymbol{w}_k) + (\frac{L\alpha^2}{2} - \alpha)||\nabla F(\boldsymbol{w}_k)||_2^2.$$

The RHS (optimal guaranteed improvement) is minimized at $\alpha = \frac{1}{L}$,

$$F(\boldsymbol{w}_{k+1}) \leq F(\boldsymbol{w}_k) - \frac{1}{2L}||\nabla F(\boldsymbol{w}_k)||_2^2.$$

What are the implications of the upper bound?

- If the gradient is nonzero, the next iteration will have a lower function value (sequence is non-increasing)
- If the function has a minimum F^*, will converge to a fixed point (i.e. $\nabla F(\boldsymbol{w}) = \boldsymbol{0}$)

Implications

Algorithm (Minibatch Gradient Estimator)

Inputs: Data $\{\boldsymbol{x}_n, y_n\}_{n=1,\ldots,N}$, *minibatch size* B, *parameters* $\boldsymbol{w}_k$
Sample B *minibatch indicies* $\{i_1, \ldots, i_B\}$
Return gradient estimate: $\tilde{\mathbf{g}}_k \leftarrow \frac{1}{B}\sum_{m=1}^{B} \nabla f_{i_m}(\boldsymbol{w}_k)$

Lemma (Sequence for stochastic gradient)

Consider a function F *with Lipschitz gradient with constant* L. *For the minibatch gradient descent method with iterates*

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \tilde{g}_k,$$

then the sequence satisfies in expectation

$$\mathbb{E}[F(\boldsymbol{w}_{k+1})] - F(\boldsymbol{w}_k) \leq -\alpha_k [\nabla F(\boldsymbol{w}_k)]^T \mathbb{E}[\tilde{g}_k] + \frac{\alpha_k^2 L}{2}\mathbb{E}[||\tilde{g}_k||_2^2]$$

Consequences for an unbiased estimator

- First, we consider an unbiased gradient estimator (i.e $\mathbb{E}[\tilde{g}_k] = g_k$)
- Assume the variance is bounded (i.e. $\text{var}(||\tilde{g}_k||_2) \leq M$)
- Then the previous lemma reveals that

$$\mathbb{E}[F(\boldsymbol{w}_{k+1})] - F(\boldsymbol{w}_k) \leq \left(\frac{\alpha_k^2 L}{2} - \alpha_k\right) ||g_k||_2^2 + \frac{\alpha_k^2 LM}{2}$$

- *Only* difference to deterministic case is the term $\frac{\alpha_k^2 LM}{2}$
- Question: When is a gradient step expected to improve our cost function?

Consequences for an unbiased estimator

- Consider a step size $\alpha_k = \frac{1}{L}$

$$\mathbb{E}[F(\boldsymbol{w}_{k+1})] - F(\boldsymbol{w}_k) \leq \frac{1}{2L}(M - ||g_k||_2^2)$$

- Whether the function is improved depends on the quantity $M - ||g_k||_2^2$
  - Often, M is fairly constant, but $||g_k||_2^2 \to 0$ at the optimum
- Smaller step sizes allow for further optimization
  - Smaller step sizes take (much) longer if not necessary!

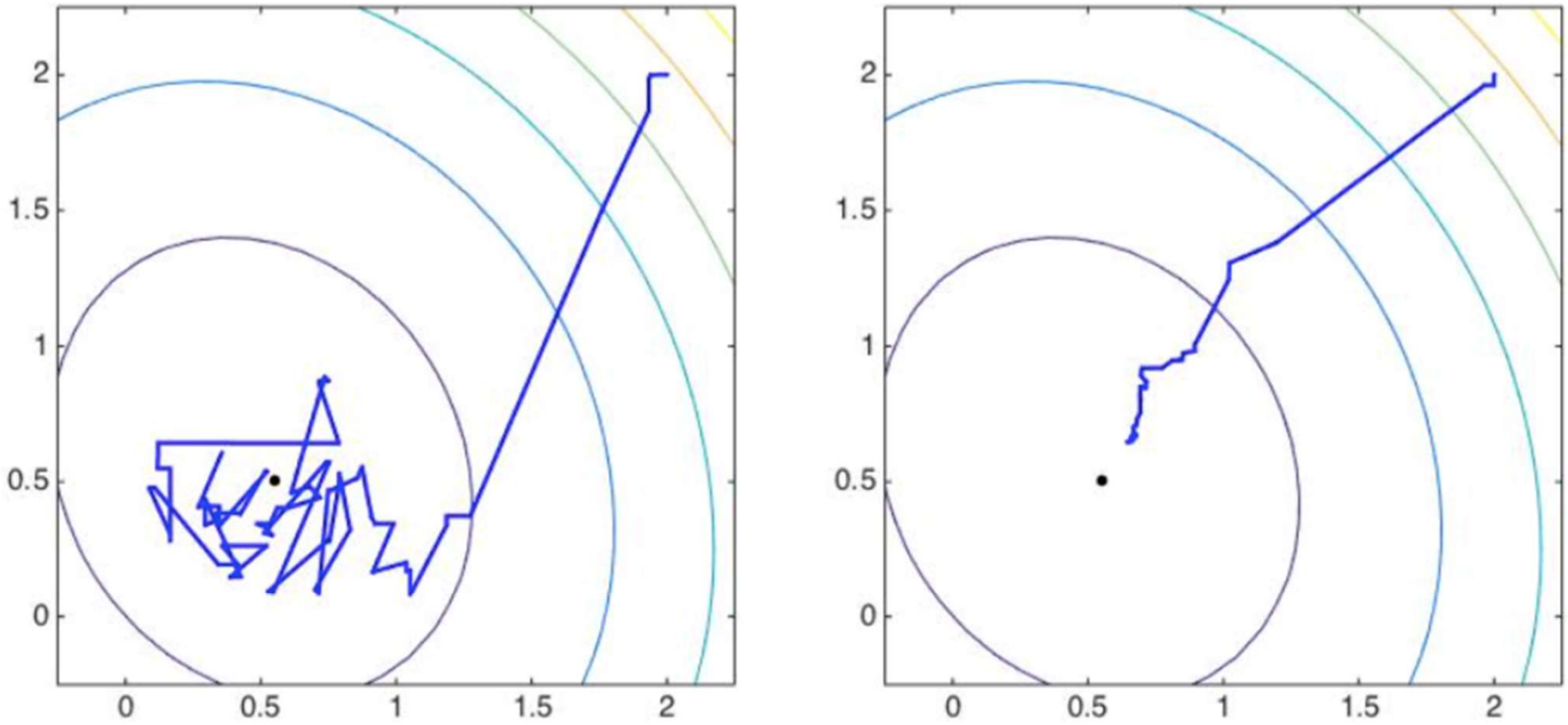


Figure: (Left) Stochastic gradient descent with a fixed step size. (Right) Stochastic gradient descent with diminishing step size.

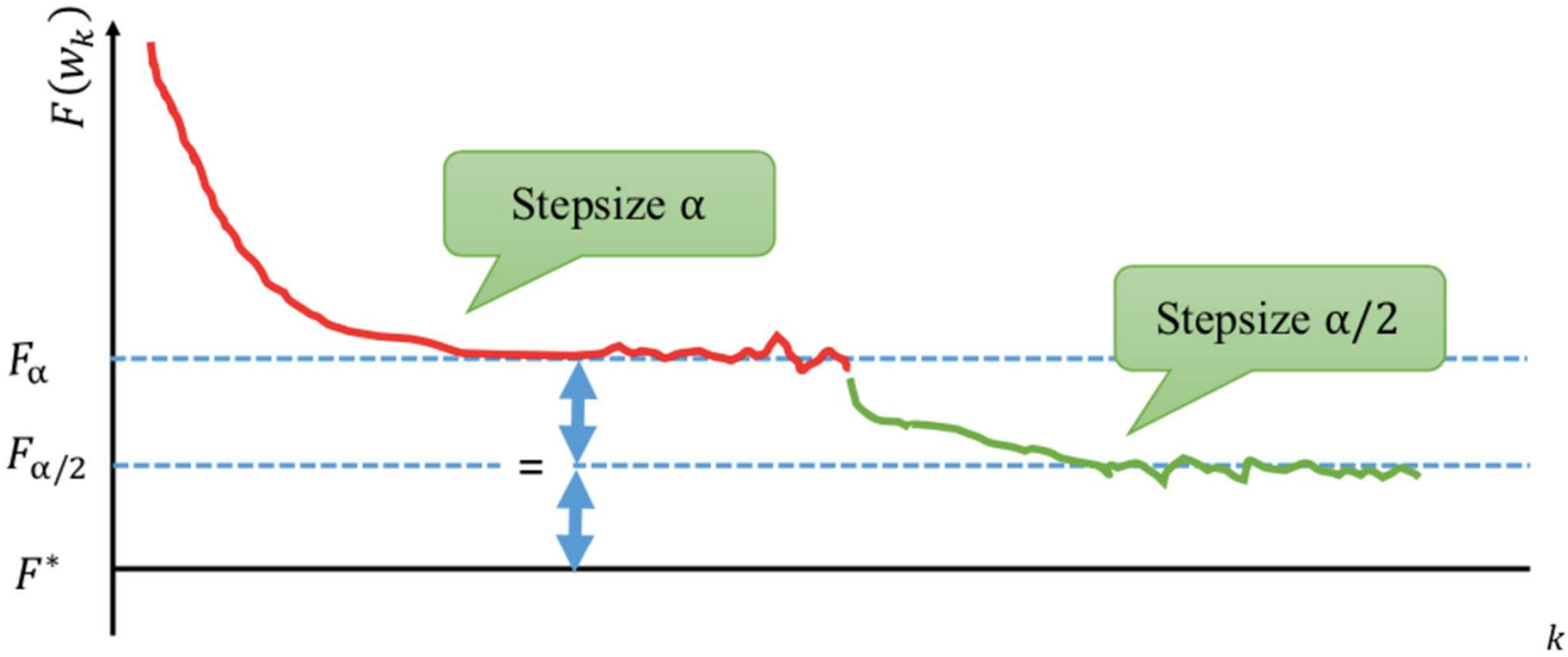


Figure: Reducing the step size allows the objective to reach closer to the optimal value.

Summary/Recap

- Rigorous analysis depends on constants that can be derived for many standard models
  - Hard to derive for deep neural networks, but intuition is important for understanding how these methods work
- The variance in the gradient estimator limits how well the function can be optimized
- Reducing the step size *or* increasing the minibatch size allows the optimization algorithm to reach closer to the optimum

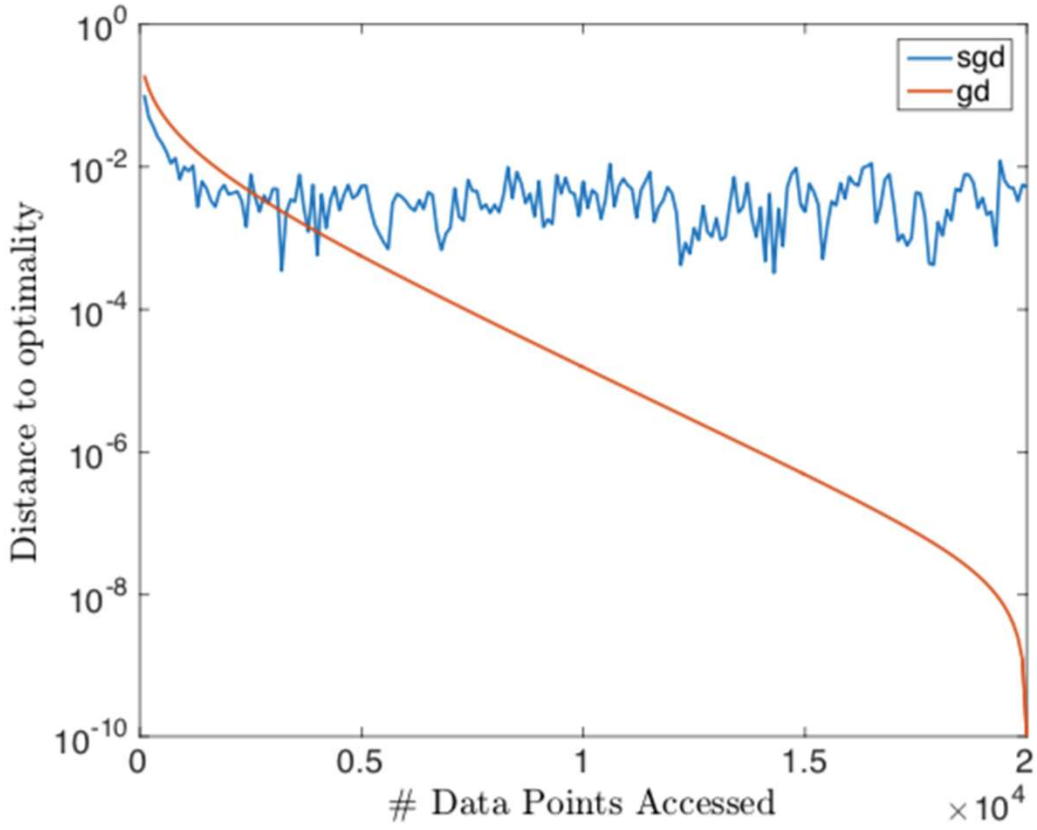


Figure: In the beginning, SGD has an exponential decay, but on small datasets GD will catch up and pass SGD.

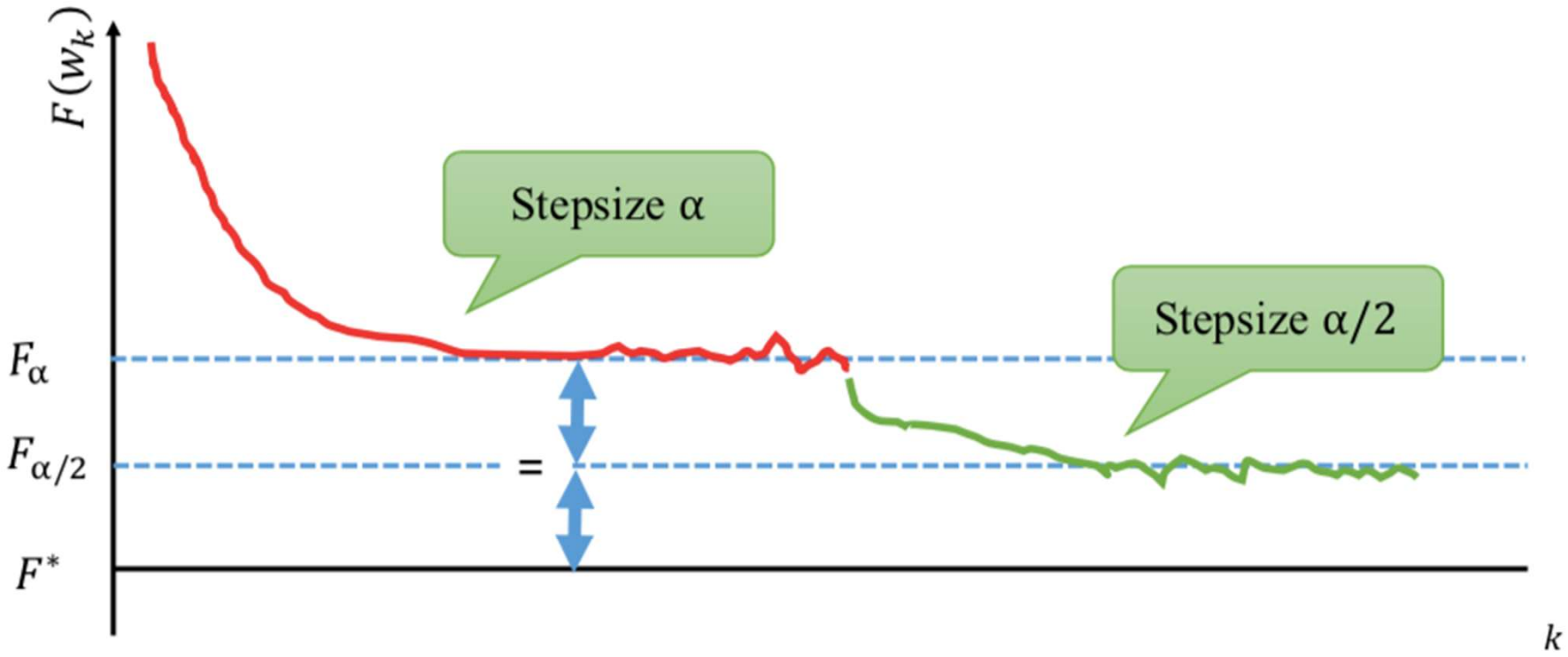


Figure: Reducing the step size allows the objective to reach closer to the optimal value.

SGD with diminishing step sizes

Algorithm (Minibatch Gradient Descent)

Inputs: $\beta, \gamma, \kappa, \boldsymbol{w}_0$
Initialize: $k \leftarrow 0$
for $k=0,\ldots$ ***do***
  Estimate gradient: $\tilde{\mathbf{g}}_k$
  Calculate step size: $\alpha_k = \beta(\gamma + k)^{-\kappa}$
  Update parameters: $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha_k \tilde{\mathbf{g}}_k$
end for

Theorem (Convergence of minibatch gradient descent)

For $\kappa = 1$, $\beta \geq \frac{1}{c}$ *and* $\gamma \geq \beta L$, *the expected optimality gap satisfies*

$$\mathbb{E}[F(\boldsymbol{w}_k) - F_*] \leq \frac{\nu}{\gamma + k}, \quad \nu = \max\left\{(\gamma+1)(F(\boldsymbol{w}_1) - F_*), \frac{\beta^2 LM}{2(\beta c - 1)}\right\}$$

- F is assumed to be a strongly convex function with constant c and Lipsitz gradient with constant L

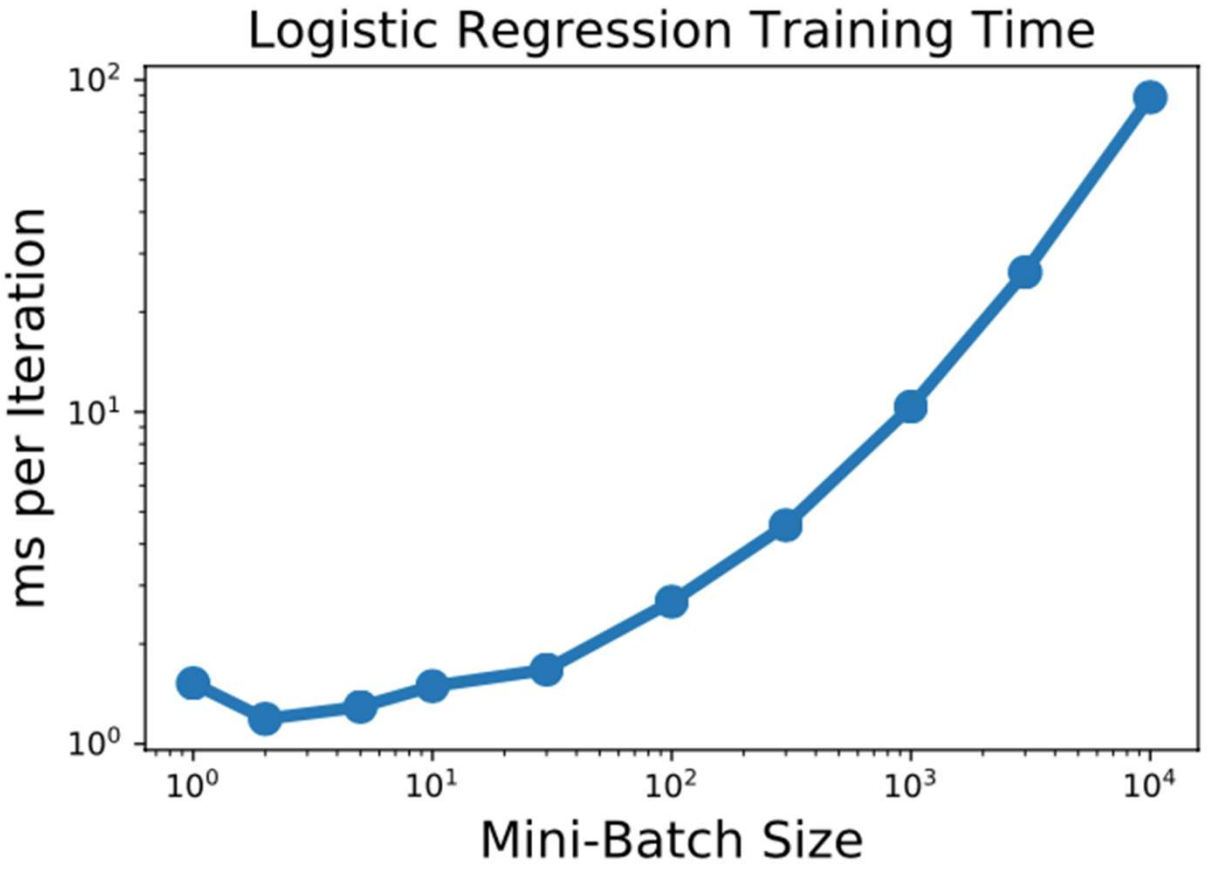
Thoughts and conclusions up to this point in time

- Practically, SGD can dominate batch methods to reasonable accuracy
- The limitations of the stochastic gradient method are:
  - Constants (and hence step sizes) are usually unknown
  - Variance in the gradient estimation limits convergence (much of the recent literature and our forthcoming discussion will discuss ways of minimizing variance)
  - Does not utilize the curvature in space, can be very poorly conditioned.
- So far, we assumed the gradient estimates were unbiased
  - Often gradients depend on a sequential procedure (RBMs, PFA, variational methods, recurrent neural nets) – gradients are not necessarily unbiased

Improving Accuracy

- One of the fundamental quantities that control the convergence of stochastic gradient is the variance M
- Complex approach: develop algorithmic methods to reduce variance (included in slides, will not get to today)
- Simple approach: increase minibatch size. Typically $M \propto 1/(\text{minibatch size})$
- Problem: Time to estimate the gradient is $\propto$ (minibatch size)
- No theoretical benefit to using minibatch over a single example – in fact, theoretical convergence rate is typically worse.

Empirical Minibatch Timing



Nestrov Acceleration Methods

Nesterov's Accelerated Gradient Descent:

- Nesterov methods invented originally by Yurii Nesterov is used for acceleration of the first order methods.
- Remember that function F with **only Lipchitz gradient** assumption has sublinear rate $\mathcal{O}(\frac{1}{k})$ convergence.
- Nesterov method establishes a better rate like $\mathcal{O}(\frac{1}{k^2})$.
- In addition to the gradient information from ***the previous iteration***, gradients from other iteration(s) are contributed with appropriate weights to the calculation of the current estimate of the parameter.

Nesterov's Accelerated Gradient Descent:

- First define the following sequences:

$$\boldsymbol{\lambda_0 = 0}\,, \qquad \boldsymbol{\lambda_k = \frac{1 + \sqrt{1 + 4\lambda_{k-1}{}^2}}{2}}, \qquad \boldsymbol{\gamma_k = \frac{1 - \lambda_k}{\lambda_{k+1}}}$$

- For $k = 1\ldots$ do (for some initial point $w_1 = t_1$):

$$\boldsymbol{t_{k+1} = w_k - \frac{1}{\beta}\nabla F(w_k)},$$

$$\boldsymbol{w_{k+1} = (1 - \gamma_k)t_{k+1} + \gamma_k t_k},$$

- The update involves computing of gradient in time steps k and $k + 1$ (using momentum)

Theorem (Nesterov 1983). *Let F be a convex and β-smooth function, then the Nesterov's Accelerated Gradient Descent satisfies for all* $k > 1$*:*

$$F(w_k) - F(w_*) \leq \frac{2\beta||w_1 - w_*||_2^2}{k^2}$$

Improving Stochastic Methods with Momentum (Acceleration methods)

Including Momentum in SGD

Algorithm (Minibatch Gradient Descent with Momentum)

Inputs: α, β
Initialize: $k \leftarrow 0, \mathbf{m}_k$
for $k=0,\ldots$ ***do***
  Estimate gradient: $\tilde{\mathbf{g}}_k$
  Update with momentum: $\mathbf{m}_{k+1} \leftarrow \beta \mathbf{m}_k + \tilde{\mathbf{g}}_k$
  Update parameters: $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha \mathbf{m}_{k+1}$
end for

- Classically, momentum "dampens" the highly oscillatory terms
- Analysis reduces dependency of $\frac{L}{c}$ (a worst-case condition number of the Hessian) to $\sqrt{\frac{L}{c}}$ for a strongly convex model.

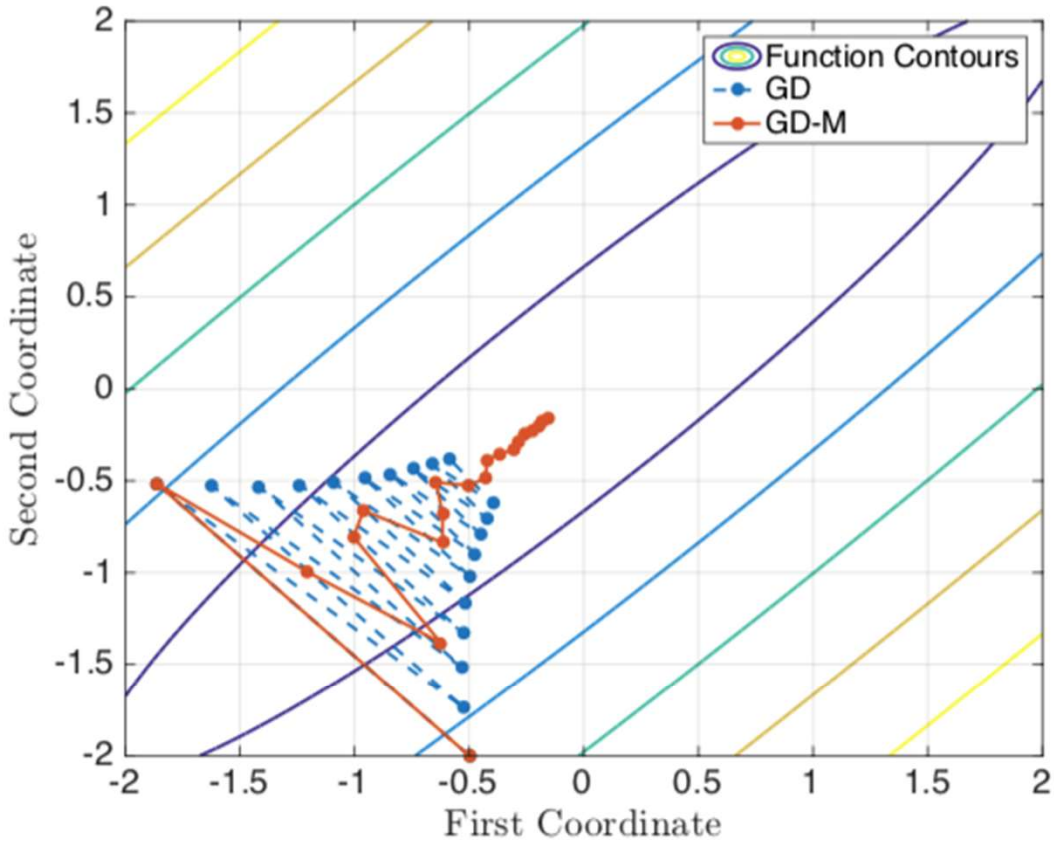


Figure: Effect of using momentum for a skewed 2D Gaussian. Gradient descent bounces back and forth, but using momentum averages out the first dimension and finds the correct path.

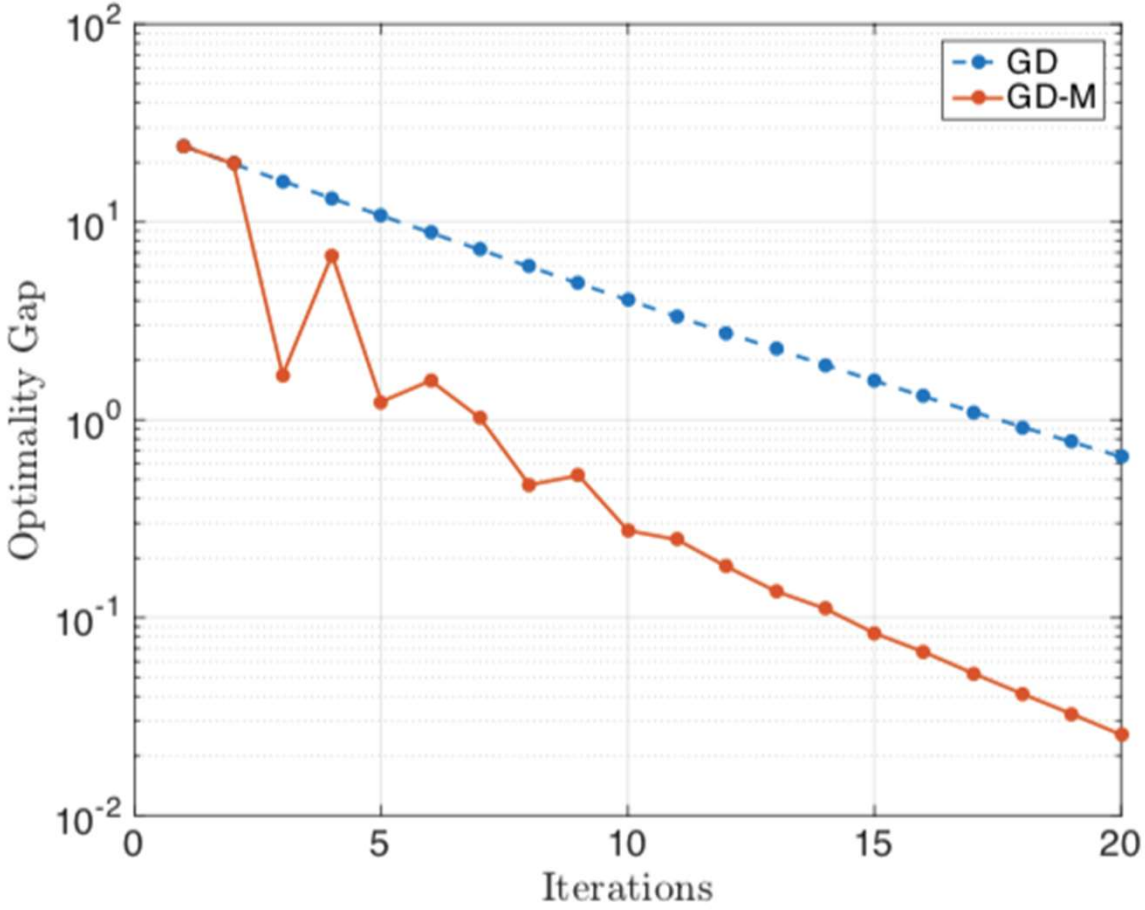


Figure: Effect of using momentum for a skewed 2D Gaussian in the previous figure. Using momentum greatly improves the convergence speed.

Two Viewpoints on Momentum

- Define momentum updates as exponential smoothing

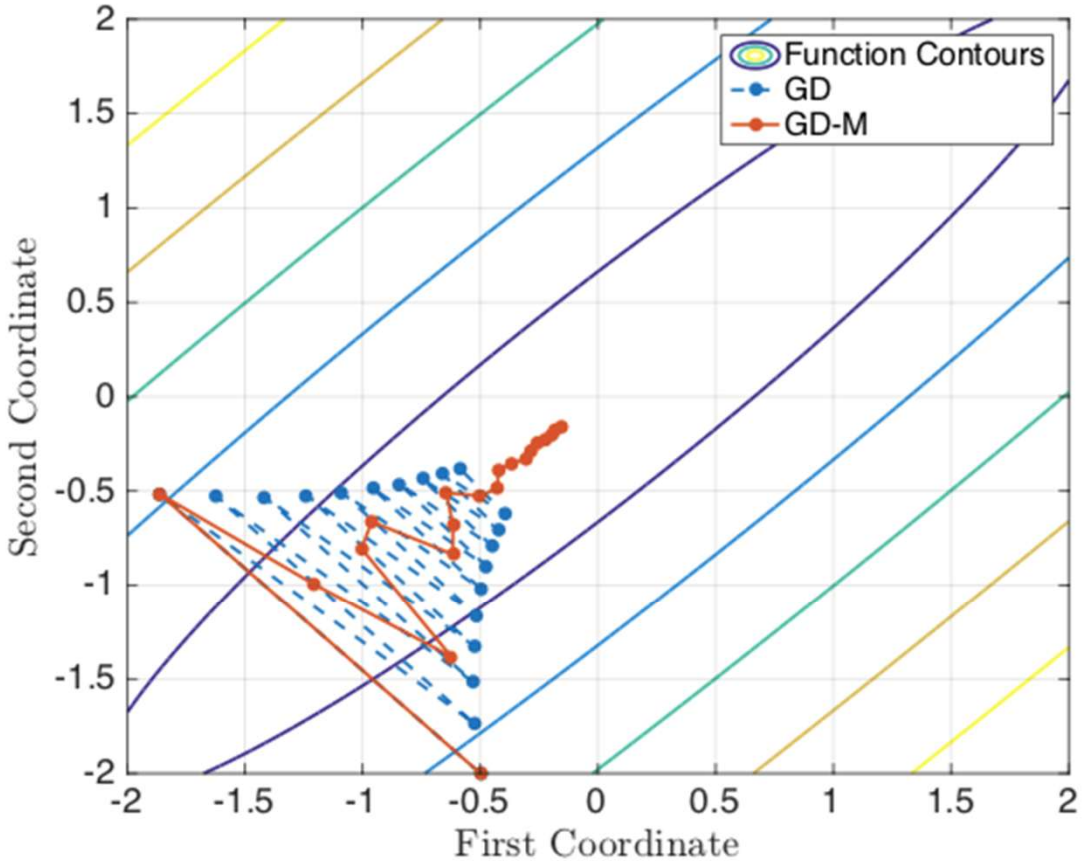
$$\bar{\mathbf{g}} = (1-\beta)\sum_{i=0}^{k}\beta^i\tilde{\mathbf{g}}_{k-i}.$$

- Note: $\sum_{i=0}^{\infty}\beta^i = (1-\beta)^{-1}$
- Then SGD with momentum is implemented with the update

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k\bar{\mathbf{g}}.$$

- This can be viewed as:
  - A filter that dampens out transient patterns
  - A bias-variance tradeoff (decrease variance for an small bias) on noisy gradient estimates

Dampening of Transient Patterns



Momentum as De-noising Gradient Estimates

- Examining the term

$$\bar{\mathbf{g}} = (1-\beta)\sum_{i=0}\beta^i\tilde{\mathbf{g}}_{k-i}.$$

- If $\mathbb{E}[\tilde{\mathbf{g}}_k] = \mathbf{g}_k$ and $\text{var}(||\mathbf{g}_k||_2) \leq M$, then as $k \to \infty$

$$\text{var}(||\bar{\mathbf{g}}_k||_2) = (1-\beta)^2\frac{1}{1-\beta^2}M = \frac{1-\beta}{1+\beta}M.$$

- Decreases variance by a multiplicative factor of $\frac{1-\beta}{1+\beta}$
- Downside: introduces bias (i.e. $\mathbb{E}[\bar{\mathbf{g}}_k] \neq \mathbf{g}_k$)

Higher Order Methods

Higher Order Methods

- SGD only uses gradient (first-order) information
- Sometimes the *curvature* is drastically different depending on the direction
  - Recall the momentum example!
- Can change the direction based on curvature
- step size tuning can pose problems, can use curvature to estimate an appropriate step size

Second Order Methods

Definition (Second order approximation)

Consider a second order expansion:

$$\begin{aligned} q_k(\boldsymbol{w}) =& F(\boldsymbol{w}_k) + \nabla F(\boldsymbol{w}_k)^T(\boldsymbol{w} - \boldsymbol{w}_k) \\ &+ \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_k)^T \hat{B}^{-1}(\boldsymbol{w} - \boldsymbol{w}_k). \end{aligned}$$

Move $\boldsymbol{w}$ in the direction that minimizes q_k:

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha_k \hat{B} \nabla F(\boldsymbol{w}_k).$$

Note: $\alpha_k = 1$ will minimize $q_k(\boldsymbol{w})$.

Newton's Methods

Algorithm (Newton's method)

Let $\hat{B}^{-2} = \nabla^2 F(\boldsymbol{w}_k)$ *(i.e. the Hessian). Then*

$$\begin{aligned} \boldsymbol{w}_{k+1} &= \arg\min_{\boldsymbol{w}} q_k(\boldsymbol{w}) \\ &= \boldsymbol{w}_k - [\nabla^2 F(\boldsymbol{w}_k)]^{-1} \nabla F(\boldsymbol{w}_k) \end{aligned}$$

If certain conditions are satisfied, this approach converges with quadratic convergence $\mathcal{O}(\rho^{k^2})$.

- Convergence is great, but many issues:
  - Gradient typically costs $\mathcal{O}(ND)$
  - Forming the Hessian typically costs $\mathcal{O}(ND^2)$
  - Applying the Hessian to the gradient typically costs $\mathcal{O}(D^3)$
  - Often assumptions are not satisfied, may diverge!
- Can often get a "good enough" solution from SGD by the time Newton runs a single iteration for big data and big models

Practical Approaches

- A large neural network may have $>>$ 1 million parameters
  - Neither feasible to calculate Hessian, nor apply it
- Still want to be able to use curvature information
- Common approach is to use a diagonal approximation

$$\begin{aligned} q_k(\boldsymbol{w}) =& F(\boldsymbol{w}_k) + \nabla F(\boldsymbol{w}_k)^T(\boldsymbol{w} - \boldsymbol{w}_k) \\ &+ \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_k)^T \text{diag}(\mathbf{b})^{-1}(\boldsymbol{w} - \boldsymbol{w}_k) \end{aligned}$$

with the minimizer with $\alpha_k = 1$ as

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha_k \mathbf{b} \odot \nabla F(\boldsymbol{w}_k)$$

- Several approaches to developing **b**

Consequences of diagonal approximations

- The follow algorithms *do not* improve the convergence rate
- They can improve optimizations constants (provably on regret)
  - The constants may be dramatically improved
- Biggest reason to use the following methods is their *robustness* to settings
  - SGD methods are very sensitive to the step size sequences
  - Many of the so-called “adaptive metric” methods can use the same settings across many models and datasets

Adagrad Algorithm

Algorithm (ADAgrad)

Inputs: $\epsilon, \gamma, \boldsymbol{w}_0$
Initialize: $k \leftarrow 0, \quad \boldsymbol{v}_0 \leftarrow \mathbf{0}$
for *k=0,...* ***do***
  Estimate gradient: $\tilde{\mathbf{g}}_k$
  Update sum-of-squares: $\boldsymbol{v}_{k+1} \leftarrow \boldsymbol{v}_k + \tilde{\mathbf{g}}_k \odot \tilde{\mathbf{g}}_k$
  Calculate element-wise step sizes: $\alpha_k = \gamma \mathbf{1} \oslash \left(\epsilon + \sqrt{\boldsymbol{v}_{k+1}}\right)$
  Update parameters: $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha_k \odot \tilde{\mathbf{g}}_k$
end for

- Key idea: sparsely occurring features may be *very* informative – only decrease step size when information relating to that parameter is seen
- *Provably* improves regret bounds compared to online (stochastic) gradient descent

RMSprop Algorithm

Algorithm (RMSprop)

Inputs: $\epsilon, \gamma, \beta, \boldsymbol{w}_0, \alpha_k$
Initialize: $k \leftarrow 0, \boldsymbol{v}_0 \leftarrow \mathbf{0}$
for *k=0,. . .* ***do***
  Estimate gradient: $\tilde{\mathbf{g}}_k$
  Update sum-of-squares: $\boldsymbol{v}_{k+1} \leftarrow (1-\beta)\boldsymbol{v}_k + \beta(\tilde{\mathbf{g}}_k \odot \tilde{\mathbf{g}}_k)$
  Calculate element-wise preconditioner: $\mathbf{b}_k = \gamma \mathbf{1} \oslash \left(\epsilon + \sqrt{\boldsymbol{v}_{k+1}}\right)$
  Update parameters: $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha_k \mathbf{b}_k \odot \tilde{\mathbf{g}}_k$
end for

- Will converge if $\alpha \to 0$ with at rate $t^{\frac{1}{2}}$
- Typically a constant step size is used $\alpha_k = \bar{\alpha} \simeq 10^{-3}$
- In practice, has been largely replaced the ADAM optimizer

- Comments on Adagrad

  - ADAgrad is a great algorithm when features are sparse
  - Step sizes can diminish much too quickly (no forgetting–once an element-wise step size goes small, it stays small)
  - Intuitive to add a “forgetting” term, especially in deep learning

- Comments on RMSprop

  - Ad-hoc: originally no convergence guarantee
  - No inclusion of momentum
  - Adaptive Moments (ADAM) addressed these issues
  - RMSprop had significant practical successes, but is less used now

ADAM Algorithm

Algorithm (ADAM)

Inputs: $\alpha_{1,\ldots,T}, \beta_1, \beta_2, \boldsymbol{w}_0$

Initialize: $k \leftarrow 0, \mathbf{m}_0^{(1)} \leftarrow \mathbf{0}, \mathbf{m}_0^{(2)} \leftarrow \mathbf{0}$

for *k=0,...* ***do***

Estimate gradient: $\tilde{\mathbf{g}}_k$

Update first moment: $\mathbf{m}_{k+1}^{(1)} \leftarrow \beta_1 \mathbf{m}_k^{(1)} + (1-\beta_1)\tilde{\mathbf{g}}_k$

"Debias" first moment $\tilde{\mathbf{m}}_{k+1}^{(1)} \leftarrow \mathbf{m}_{k+1}^{(1)} (1-{\beta_1}^{(k+1)})^{-1}$

Update second moment: $\mathbf{m}_{k+1}^{(2)} \leftarrow \beta_2 \mathbf{m}_k^{(2)} + (1-\beta_2)\tilde{\mathbf{g}}_k \odot \tilde{\mathbf{g}}_k$

"Debias" second moment $\tilde{\mathbf{m}}_{k+1}^{(2)} \leftarrow \mathbf{m}_{k+1}^{(2)} (1-{\beta_2}^{(k+1)})^{-1}$

Update parameters: $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha_t (\tilde{\mathbf{m}}_{k+1}^{(1)}) \oslash \left(\epsilon + \sqrt{\tilde{\mathbf{m}}_{k+1}^{(2)}}\right)$

end for

Analysis of ADAM Algorithm

- RMSprop inspired ADAM – if $\beta_1 = 0$ then they are very similar algorithms
- Provable regret bound under a decreasing step size
- Can typically use "standard" parameters for many problems, e.g.

$$\alpha_k = \bar{\alpha} = 10^{-3}, \ \beta_1 = .9, \ \beta_2 = .999$$

- Reminder: β_1 going higher reduces variance of gradients, but can add bias

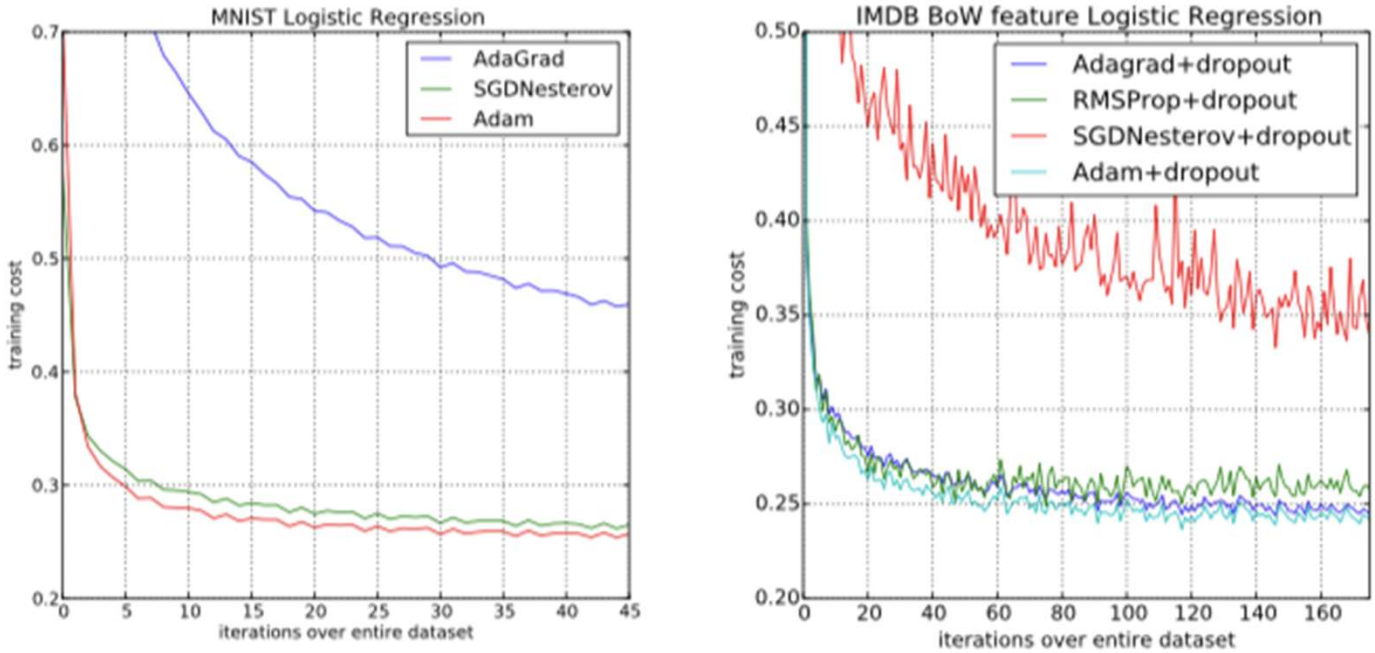


Figure: Comparison of several learning rules on large-scale logistic regression.

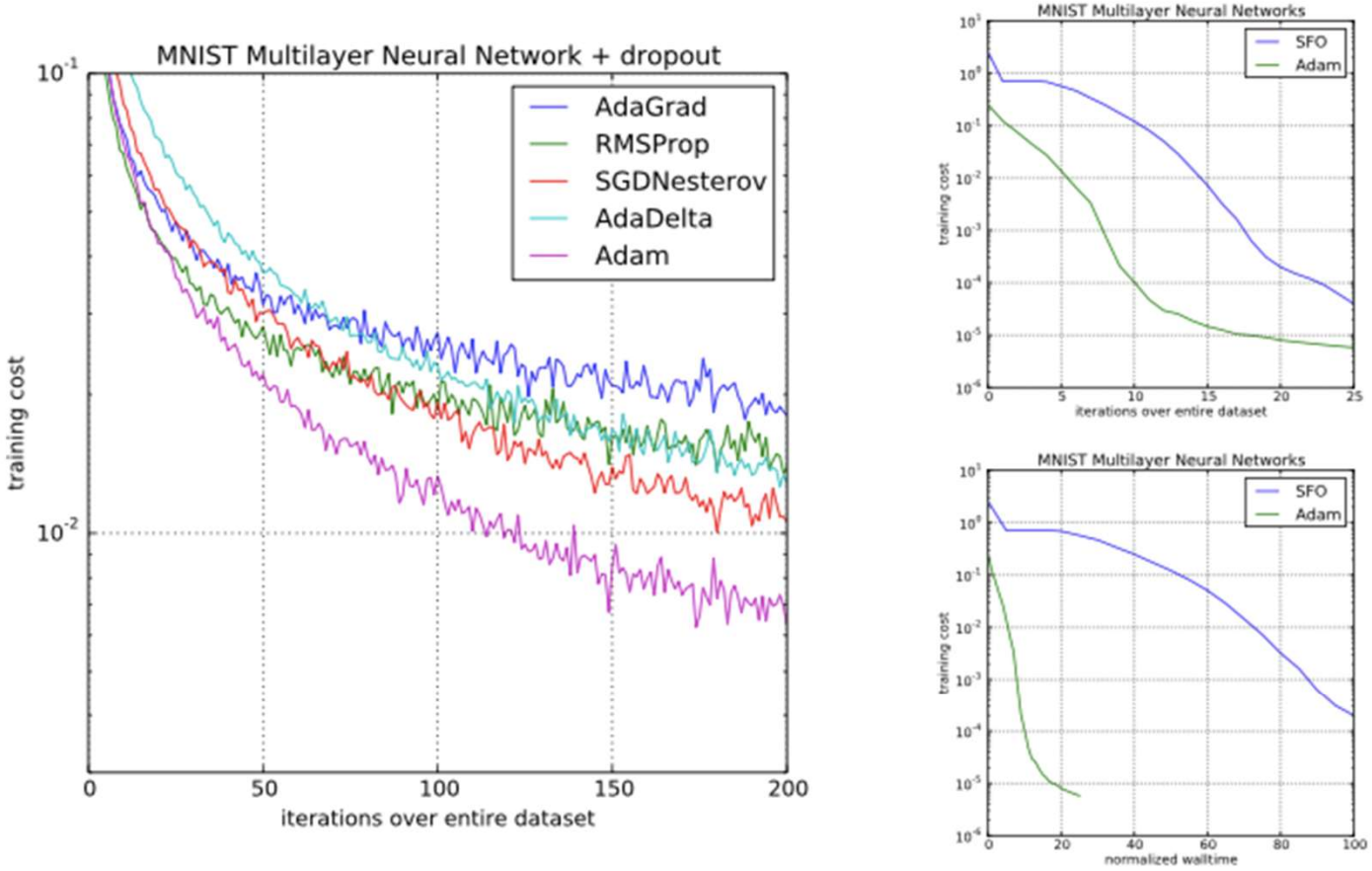


Figure: Comparison of several learning rules on deep neural networks.

Conclusions

- Stochastic gradient descent will converge to the optimum if given enough time
  - Will converge to a fixed point in nonconvex problems as well
- Adding in adaptive higher order information can reduce parameter tuning and account for curvature information
- Stochastic gradient methods are effective at quickly obtaining a reasonable solution
- Machines are learned with stochastic gradient methods
- Many theoretical issues but many practical successes
- Still a very active area of research!
- Many tuning parameters within these algorithms
  - Hopefully their meaning and how to set them is clearer now