CS6301: Optimization in Machine Learning

Lecture 15: Stochastic Gradient Descent and Family

Rishabh Iyer

Department of Computer Science
University of Texas, Dallas
https://sites.google.com/view/cs-6301-optml/home

March 11th, 2020

Announcements

- Assignment 2 due date extended to Monday, 3/16/2020.
- Next week Spring Break
- Class cancelled on 3/23 and likely 3/25 (I'll confirm closer to the date) due to a NSF review panel.
- Assignment 3 will be posted around 3/23 (this will be to implement Coordinate Descent and various flavors of SGD).
- Mid Term review of the Project deadline: 3/25. Please submit a PDF document with progress so far in the project.

Outline

- Recap of Gradient Descent and Machine Learning Loss Functions
- Stochastic Gradient Descent: Intuition, Convergence Results
- Hybrid Approaches: SGD and GD Combined
- Variants of SGD: AdaGrad, ADAM, NAG, RMSProp, ...
- Some interesting insights into SGD, Recent Research Directions

UT DALLAS

Recap: Convex Optimization Problem

- Formally, a convex optimization problem is an optimization problem of the form

$$\begin{aligned} &\textit{minimize } f(x) \\ &\text{subject to } x \in X \end{aligned}$$

  where f is a convex function, X is a convex set, and x is the optimization variable.
- if $X = dom(f)$, this becomes unconstrained optimization.
- Large chunk of ML Optimization Problems!

Recap: Gradient Descent

- Goal: Given a convex function f, find a $x \in \mathbb{R}^n$ such that $|f(x) - f(x^*)| \leq \epsilon$

Recap: Gradient Descent

- Goal: Given a convex function f, find a $x \in \mathbb{R}^n$ such that $|f(x) - f(x^*)| \leq \epsilon$
- Iterative algorithm: Initialize x_0 either randomly or say, 0. Then set

$$x_{t+1} = x_t - \gamma \nabla f(x_t)$$

γ is the step size parameter which needs to be set.

Recap: Gradient Descent

- Goal: Given a convex function f, find a $x \in \mathbb{R}^n$ such that $|f(x) - f(x^*)| \leq \epsilon$
- Iterative algorithm: Initialize x_0 either randomly or say, 0. Then set

$$x_{t+1} = x_t - \gamma \nabla f(x_t)$$

  γ is the step size parameter which needs to be set.
- Critical Question: How much time does it take to reach an ϵ-approximate solution?

Recap: Gradient Descent

- Goal: Given a convex function f, find a $x \in \mathbb{R}^n$ such that $|f(x) - f(x^*)| \leq \epsilon$
- Iterative algorithm: Initialize x_0 either randomly or say, 0. Then set

$$x_{t+1} = x_t - \gamma \nabla f(x_t)$$

  γ is the step size parameter which needs to be set.
- Critical Question: How much time does it take to reach an ϵ-approximate solution?
- Define x^* as the Global minimizer of f

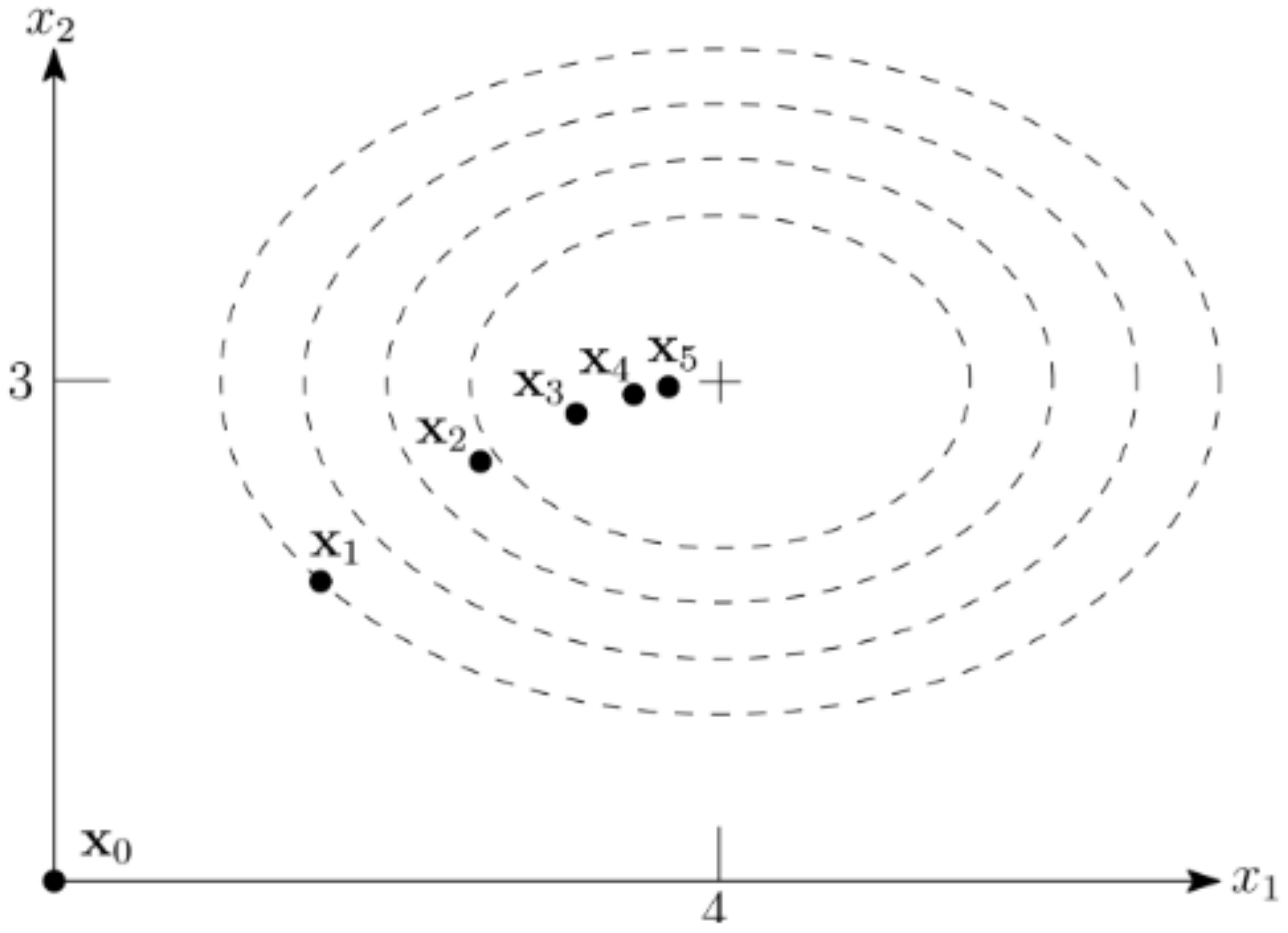
Recap: Gradient Descent

- Goal: Given a convex function f, find a $x \in \mathbb{R}^n$ such that $|f(x) - f(x^*)| \leq \epsilon$
- Iterative algorithm: Initialize x_0 either randomly or say, 0. Then set

$$x_{t+1} = x_t - \gamma \nabla f(x_t)$$

  γ is the step size parameter which needs to be set.
- Critical Question: How much time does it take to reach an ϵ-approximate solution?
- Define x^* as the Global minimizer of f
- Let f be Lipschitz continuous with parameter B. If f is smooth, let ∇f be Lipschitz continuous with parameter L.

Gradient Descent Illustration



Source: Martin Jaggi (CS 439)

Recap: Gradient Descent Results

- Lipschitz continuous functions (C): R^2B^2/ϵ^2 iterations

Recap: Gradient Descent Results

- Lipschitz continuous functions (C): R^2B^2/ϵ^2 iterations
- Lipschitz continuous functions + Strongly Convex (CS): $2B^2/\epsilon - 1$ iterations

Recap: Gradient Descent Results

- Lipschitz continuous functions (C): R^2B^2/ϵ^2 iterations
- Lipschitz continuous functions + Strongly Convex (CS): $2B^2/\epsilon - 1$ iterations
- Smooth Functions GD (SGD): $\frac{R^2L}{\epsilon}$ iterations.

Recap: Gradient Descent Results

- Lipschitz continuous functions (C): R^2B^2/ϵ^2 iterations
- Lipschitz continuous functions + Strongly Convex (CS): $2B^2/\epsilon - 1$ iterations
- Smooth Functions GD (SGD): $\frac{R^2L}{\epsilon}$ iterations.
- Smooth Functions Nesterov's GD: $\sqrt{\frac{2LR^2}{\epsilon}}$ iterations

UT DALLAS

Recap: Gradient Descent Results

- Lipschitz continuous functions (C): R^2B^2/ϵ^2 iterations
- Lipschitz continuous functions + Strongly Convex (CS): $2B^2/\epsilon - 1$ iterations
- Smooth Functions GD (SGD): $\frac{R^2L}{\epsilon}$ iterations.
- Smooth Functions Nesterov's GD: $\sqrt{\frac{2LR^2}{\epsilon}}$ iterations
- Smooth + Strongly Convex (SS): With $\gamma = 1/L$, achieve an ϵ-approximate solution in $\frac{L}{\mu}\log(\frac{R^2L}{2\epsilon})$ iterations.

UT DALLAS

Loss Functions in Machine Learning

- L1/L2 Reg Logistic Regression:
  $L(\theta) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \theta^T x_i)) + \lambda \|\theta\|$

Loss Functions in Machine Learning

- L1/L2 Reg Logistic Regression:
  $L(\theta) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \theta^T x_i)) + \lambda \|\theta\|$
- L1/L2 Reg SVMs: $L(\theta) = \sum_{i=1}^{n} \max\{0, 1 - y_i \theta^T x_i\} + \lambda \|\theta\|$

Loss Functions in Machine Learning

- L1/L2 Reg Logistic Regression:
  $L(\theta) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \theta^T x_i)) + \lambda \|\theta\|$
- L1/L2 Reg SVMs: $L(\theta) = \sum_{i=1}^{n} \max\{0, 1 - y_i \theta^T x_i\} + \lambda \|\theta\|$
- L1/L2 Reg Multi-class Logistic Regression: $L(\theta_1, \cdots, \theta_k) = \sum_{i=1}^{n} -\theta_{y_i}^T x_i + \log(\sum_{c=1}^{k} \exp(\theta_c^T x_i))\} + \sum_{i=1}^{c} \lambda \sum_{j=1}^{m} \|\theta_j\|$

Loss Functions in Machine Learning

- L1/L2 Reg Logistic Regression:
  $L(\theta) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \theta^T x_i)) + \lambda \|\theta\|$
- L1/L2 Reg SVMs: $L(\theta) = \sum_{i=1}^{n} \max\{0, 1 - y_i \theta^T x_i\} + \lambda \|\theta\|$
- L1/L2 Reg Multi-class Logistic Regression: $L(\theta_1, \cdots, \theta_k) = \sum_{i=1}^{n} -\theta_{y_i}^T x_i + \log(\sum_{c=1}^{k} \exp(\theta_c^T x_i))\} + \sum_{i=1}^{c} \lambda \sum_{j=1}^{m} \|\theta_j\|$
- L1/L2 Reg Least Squares (Lasso): $L(\theta) = \sum_{i=1}^{n} (\theta^T x_i - y_i)^2 + \lambda \|\theta\|$

Loss Functions in Machine Learning

- L1/L2 Reg Logistic Regression:
  $L(\theta) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \theta^T x_i)) + \lambda \|\theta\|$
- L1/L2 Reg SVMs: $L(\theta) = \sum_{i=1}^{n} \max\{0, 1 - y_i \theta^T x_i\} + \lambda \|\theta\|$
- L1/L2 Reg Multi-class Logistic Regression: $L(\theta_1, \cdots, \theta_k) = \sum_{i=1}^{n} -\theta_{y_i}^T x_i + \log(\sum_{c=1}^{k} \exp(\theta_c^T x_i))\} + \sum_{i=1}^{c} \lambda \sum_{j=1}^{m} \|\theta_j\|$
- L1/L2 Reg Least Squares (Lasso): $L(\theta) = \sum_{i=1}^{n} (\theta^T x_i - y_i)^2 + \lambda \|\theta\|$
- Matrix Completion: $L(X) = \sum_{i=1}^{n} ||y_i - A_i(X)||_2^2 + ||X||_*$

Loss Functions in Machine Learning

- L1/L2 Reg Logistic Regression:
  $L(\theta) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \theta^T x_i)) + \lambda \|\theta\|$
- L1/L2 Reg SVMs: $L(\theta) = \sum_{i=1}^{n} \max\{0, 1 - y_i \theta^T x_i\} + \lambda \|\theta\|$
- L1/L2 Reg Multi-class Logistic Regression: $L(\theta_1, \cdots, \theta_k) = \sum_{i=1}^{n} -\theta_{y_i}^T x_i + \log(\sum_{c=1}^{k} \exp(\theta_c^T x_i))\} + \sum_{i=1}^{c} \lambda \sum_{j=1}^{m} \|\theta_j\|$
- L1/L2 Reg Least Squares (Lasso): $L(\theta) = \sum_{i=1}^{n} (\theta^T x_i - y_i)^2 + \lambda \|\theta\|$
- Matrix Completion: $L(X) = \sum_{i=1}^{n} ||y_i - A_i(X)||_2^2 + ||X||_*$
- Soft-Max Contextual Bandits: $L(\theta) = \sum_{i=1}^{n} \frac{r_i}{p_i} \frac{\exp(\theta^T x_i^{a_i})}{\sum_{j=1}^{k} \exp(\theta^T x_i^j)} + \lambda \|\theta\|$

Framework of Loss Functions in ML

- Machine Learning Loss functions are often a special class of continuous functions.

UT DALLAS

Framework of Loss Functions in ML

- Machine Learning Loss functions are often a special class of continuous functions.
- They involve a sum of a large number of components:

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$$

where f_i is the cost function of the ith observation, n is the total number of training examples

Framework of Loss Functions in ML

- Machine Learning Loss functions are often a special class of continuous functions.
- They involve a sum of a large number of components:

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$$

where f_i is the cost function of the ith observation, n is the total number of training examples

- For example, $f_i(\theta) = L(x_i, y_i, \theta)$ where (x_i, y_i) comprise of the ith training point in supervised learning, L is a Loss function

Framework of Loss Functions in ML

- Machine Learning Loss functions are often a special class of continuous functions.
- They involve a sum of a large number of components:

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$$

where f_i is the cost function of the ith observation, n is the total number of training examples

- For example, $f_i(\theta) = L(x_i, y_i, \theta)$ where (x_i, y_i) comprise of the ith training point in supervised learning, L is a Loss function
- Evaluating the full gradient ∇f of such an objective is $O(n)$

UT DALLAS

Framework of Loss Functions in ML

- Machine Learning Loss functions are often a special class of continuous functions.
- They involve a sum of a large number of components:

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$$

where f_i is the cost function of the ith observation, n is the total number of training examples

- For example, $f_i(\theta) = L(x_i, y_i, \theta)$ where (x_i, y_i) comprise of the ith training point in supervised learning, L is a Loss function
- Evaluating the full gradient ∇f of such an objective is $O(n)$
- In many applications, n is of the order of a million to a hundreds of millions of data points!

Stochastic Gradient Descent Algorithm

choose $\mathbf{x}_0 \in \mathbb{R}^d$.

sample $i \in [n]$ uniformly at random

$$\mathbf{x}_{t+1} := \mathbf{x}_t - \gamma_t \nabla f_i(\mathbf{x}_t).$$

for **times** $t = 0, 1, \ldots,$ and **stepsizes** $\gamma_t \geq 0$.

Only update with the gradient of f_i instead of the full gradient!

Iteration is n times cheaper than in full gradient descent.

The vector $\mathbf{g}_t := \nabla f_i(\mathbf{x}_t)$ is called a stochastic gradient.

$\mathbf{g}_t$ is a vector of d random variables, but we will also simply call this a random variable.

Unbiasedness in SGD

Can't use convexity

$$f(\mathbf{x}_t) - f(\mathbf{x}^\star) \leq \mathbf{g}_t^\top (\mathbf{x}_t - \mathbf{x}^\star)$$

on top of the vanilla analysis, as this may hold or not hold, depending on how the stochastic gradient $\mathbf{g}_t$ turns out.

We will show (and exploit): the inequality holds in expectation.

Fot this, we use that by definition, $\mathbf{g}_t$ is an **unbiased estimate** of $\nabla f(\mathbf{x}_t)$:

$$\mathbb{E}\big[\mathbf{g}_t \big| \mathbf{x}_t = \mathbf{x}\big] = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{x}) = \nabla f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d.$$

Convergence Result for SGD

Theorem

Let $f : \mathbb{R}^d \to \mathbb{R}$ be convex and differentiable, $\mathbf{x}^\star$ a global minimum; furthermore, suppose that $\|\mathbf{x}_0 - \mathbf{x}^\star\| \leq R$, and that $\mathbb{E}\left[\|\mathbf{g}_t\|^2\right] \leq B^2$ for all t. Choosing the constant stepsize

$$\gamma := \frac{R}{B\sqrt{T}}$$

stochastic gradient descent yields

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[f(\mathbf{x}_t)\right] - f(\mathbf{x}^\star) \leq \frac{RB}{\sqrt{T}}.$$

Convergence Rates for SGD with Strong Convexity

Theorem

Let $f : \mathbb{R}^d \to \mathbb{R}$ be differentiable and strongly convex with parameter $\mu > 0$; let $\mathbf{x}^\star$ be the unique global minimum of f. With decreasing step size

$$\gamma_t := \frac{2}{\mu(t+1)}$$

stochastic gradient descent yields

$$\mathbb{E}\Big[f\Big(\frac{2}{T(T+1)}\sum_{t=1}^{T} t \cdot \mathbf{x}_t\Big) - f(\mathbf{x}^\star)\Big] \leq \frac{2B^2}{\mu(T+1)},$$

where $B^2 := \max_{t=1}^{T} \mathbb{E}\big[\,\|\mathbf{g}_t\|^2\,\big]$.

Almost same result as for subgradient descent, but in expectation.

Comparison of GD vs SGD: Convergence Rates

Classic GD: For vanilla analysis, we assumed that $\|\nabla f(\mathbf{x})\|^2 \leq B_{\text{GD}}^2$ for all $\mathbf{x} \in \mathbb{R}^d$, where B_{GD} was a constant. So for sum-objective:

$$\left\|\frac{1}{n}\sum_i \nabla f_i(\mathbf{x})\right\|^2 \leq B_{\text{GD}}^2 \qquad \forall \mathbf{x}$$

SGD: Assuming same for the expected squared norms of our stochastic gradients, now called B_{SGD}^2.

$$\frac{1}{n}\sum_i \left\|\nabla f_i(\mathbf{x})\right\|^2 \leq B_{\text{SGD}}^2 \qquad \forall \mathbf{x}$$

So by Jensen's inequality for $\|.\|^2$

- $B_{\text{GD}}^2 \approx \left\|\frac{1}{n}\sum_i \nabla f_i(\mathbf{x})\right\|^2 \leq \frac{1}{n}\sum_i \left\|\nabla f_i(\mathbf{x})\right\|^2 \approx B_{\text{SGD}}^2$
- B_{GD}^2 can be smaller than B_{SGD}^2, but often comparable. Very similar if larger mini-batches are used.

Comparison of GD vs SGD: Convergence Rates

Method	Assumptions	Full	Stochastic
Subgradient	convex	$O(1/\sqrt{k})$	$O(1/\sqrt{k})$
Subgradient	strongly cvx	$O(1/k)$	$O(1/k)$

So using stochastic subgradient, solve n times faster.

Method	Assumptions	Full	Stochastic
Gradient	convex	$O(1/k)$	$O(1/\sqrt{k})$
Gradient	strongly cvx	$O((1-\mu/L)^k)$	$O(1/k)$

– For smooth problems, stochastic gradient needs more iterations
– Widely used in ML, rapid initial convergence
– Several speedup techniques studied, but worst case remains same

UTDALLAS

Mini Batch SGD

- Instead of taking single instances (x_i, y_i), use a batch of them and take the average:

$$\tilde{g_t} := \frac{1}{m} \sum_{j=1}^{m} g_t^j$$

- Extreme cases:
  - $m = 1$ is SGD
  - $m = n$ is full gradient descent
- Property of Batch SGD: The variance of the gradient estimate is $1/m$ of that of vanilla SGD!
- Variance is important in the stability and performance of SGD!

Stochastic SGD

For problems which are not necessarily differentiable, we modify SGD to use a subgradient of f_i in each iteration. The update of **stochastic subgradient descent** is given by

sample $i \in [n]$ uniformly at random
let $\mathbf{g}_t \in \partial f_i(\mathbf{x}_t)$
$\mathbf{x}_{t+1} := \mathbf{x}_t - \gamma_t \mathbf{g}_t$.

In other words, we are using an unbiased estimate of a subgradient at each step, $\mathbb{E}\left[\mathbf{g}_t \middle| \mathbf{x}_t\right] \in \partial f(\mathbf{x}_t)$.

Convergence in $\mathcal{O}(1/\varepsilon^2)$, by using the subgradient property at the beginning of the proof, where convexity was applied.

SGD for Constrained Problems

- For constrained problems, the results carry over (similar to GD)
- At every step, the projection needs to be applied as usual.
- Resulting algorithm is Projected SGD!
- Requires the projection step to be exactly done.

Why Machine Learners prefer SGD/Batch SGD and its variants?

Cons of SGD:

- Slower convergence compared to GD!
- Though gradient is unbiased, it is noisy (typically high variance)
- Not guaranteed to reduce objective at every iteration!
- Setting Learning rate correctly is more of an art than a science!

UT DALLAS

Why Machine Learners prefer SGD/Batch SGD and its variants?

Cons of SGD:

- Slower convergence compared to GD!
- Though gradient is unbiased, it is noisy (typically high variance)
- Not guaranteed to reduce objective at every iteration!
- Setting Learning rate correctly is more of an art than a science!

Pros of SGD:

- Per iteration cost is n times cheaper compared to GD!
- Very fast initial convergence of SGD!
- Mini-batch gradients often accurate if the batch size is decent and the dataset is massive!

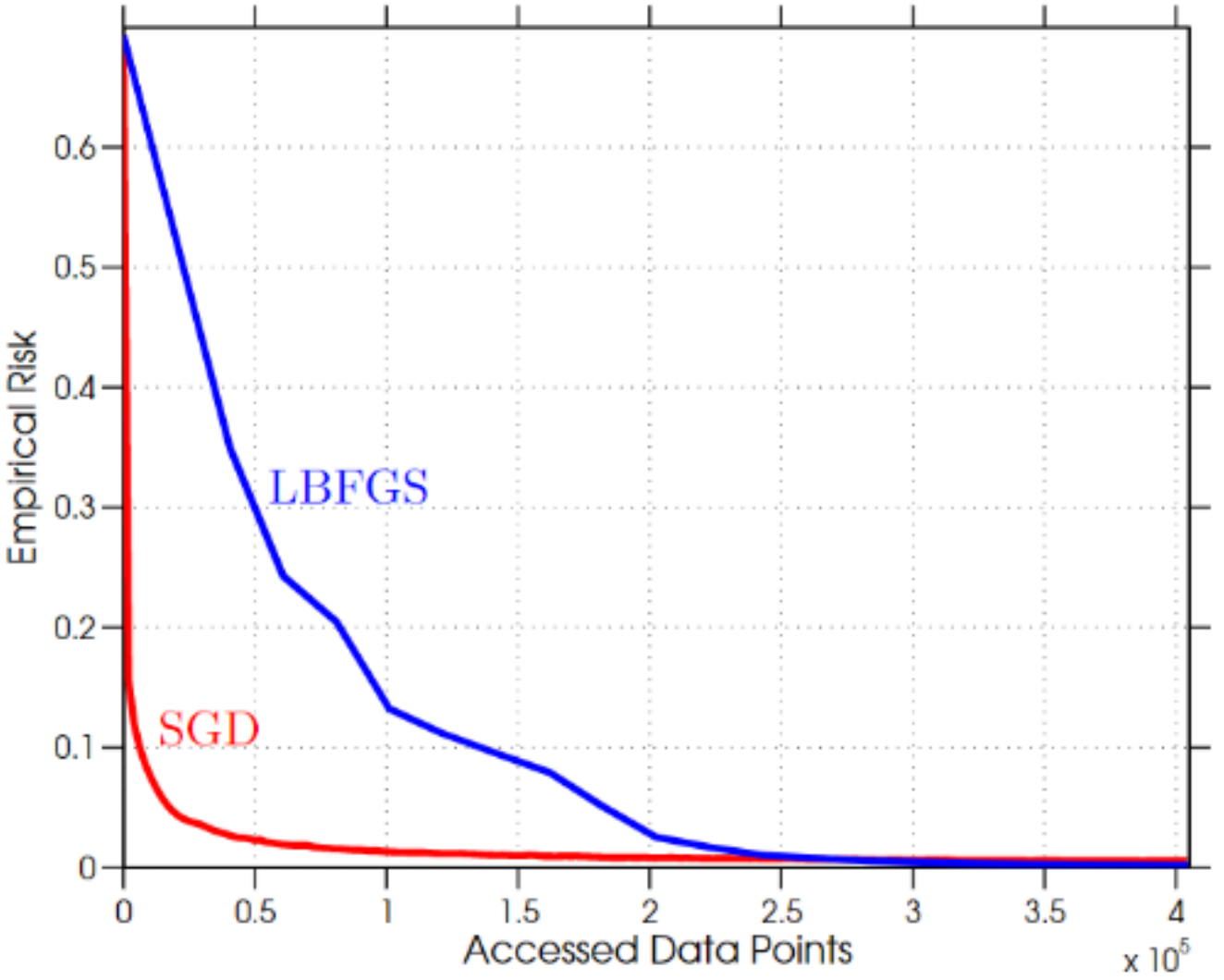
UT DALLAS

Why Machine Learners prefer SGD/Batch SGD and its variants?

- **Key Insight:** In Machine Learning (and Deep Learning), generalization is **more important** compared to getting the global minima (overfitting)
- Several times, we see techniques which perform better on the training set perform worse on generalization.
- SGD naturally generalizes very well! We do not care often times to get the exact global minima – we just want to get close by quickly.
- Large gap today between the **theory** of SGD and empirical performance!
- Not always trivial to tune the learning rates – complex rules for learning rates.

UT DALLAS

SGD vs LBFGS



DALLAS

Practical Aspects of SGD (Particularly for Deep Learning)

- Not always trivial to tune the learning rates – complex rules for learning rates.
- *When the learning rate is too large, gradient descent can inadvertently increase rather than decrease the training error. [. . .] When the learning rate is too small, training is not only slower, but may become permanently stuck with a high training error.*
- If there is only time to optimize one hyper-parameter and one uses stochastic gradient descent, then learning rate is the hyper-parameter that is worth tuning!
- If using a fixed learning rate, tune it over a validation set.
- Learning Rate Schedules: Linear rate Decay.
- With a Learning rate decay, SGD is less sensitive to the initial LR.

Improving the convergence rate of SGD

- SGD has a much lower per iteration complexity
- But it can have much slower convergence compared to GD
- Can we have the same low per iteration cost but with the convergence of GD?

UT DALLAS

Hybrid SGD: Stochastic Average Gradient

- Hybrid of stochastic gradient with full gradient.

Stochastic Average Gradient (SAG) (Le Roux, Schmidt, Bach 2012)

- store the gradients of ∇f_i for $i = 1, .., n$
- Select uniformly at random $i(k) \in \{1, \ldots, n\}$
- Perform the update

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} y_i^k \quad y_i^k = \begin{cases} \nabla f_i(x_k) & \text{if } i = i(k) \\ y_i^{k-1} & \text{otherwise.} \end{cases}$$

- Randomized / stochastic version of incremental gradient method of Blatt et al (2008)
- Storage overhead; acceptable in some ML settings:
  - $f_i(x) = \ell(l_i, x^T \Phi(a_i))$, $\nabla f_i(x) = \nabla \ell(l_i, x^T \Phi(a_i)) \Phi(a_i)$
  - Store only n scalars (since depends only on $x^T a_i$)

Hybrid SGD: Stochastic Average Gradient

Method	Assumptions	Rate
Gradient	convex	$O(1/k)$
Gradient	strongly cvx	$O((1-\mu/L)^k)$
Stochastic	strongly cvx	$O(1/k)$
SAG	strongly convex	$O((1-\min\{\frac{\mu}{n}, \frac{1}{8n}\})^k)$

This speedup also observed in practice

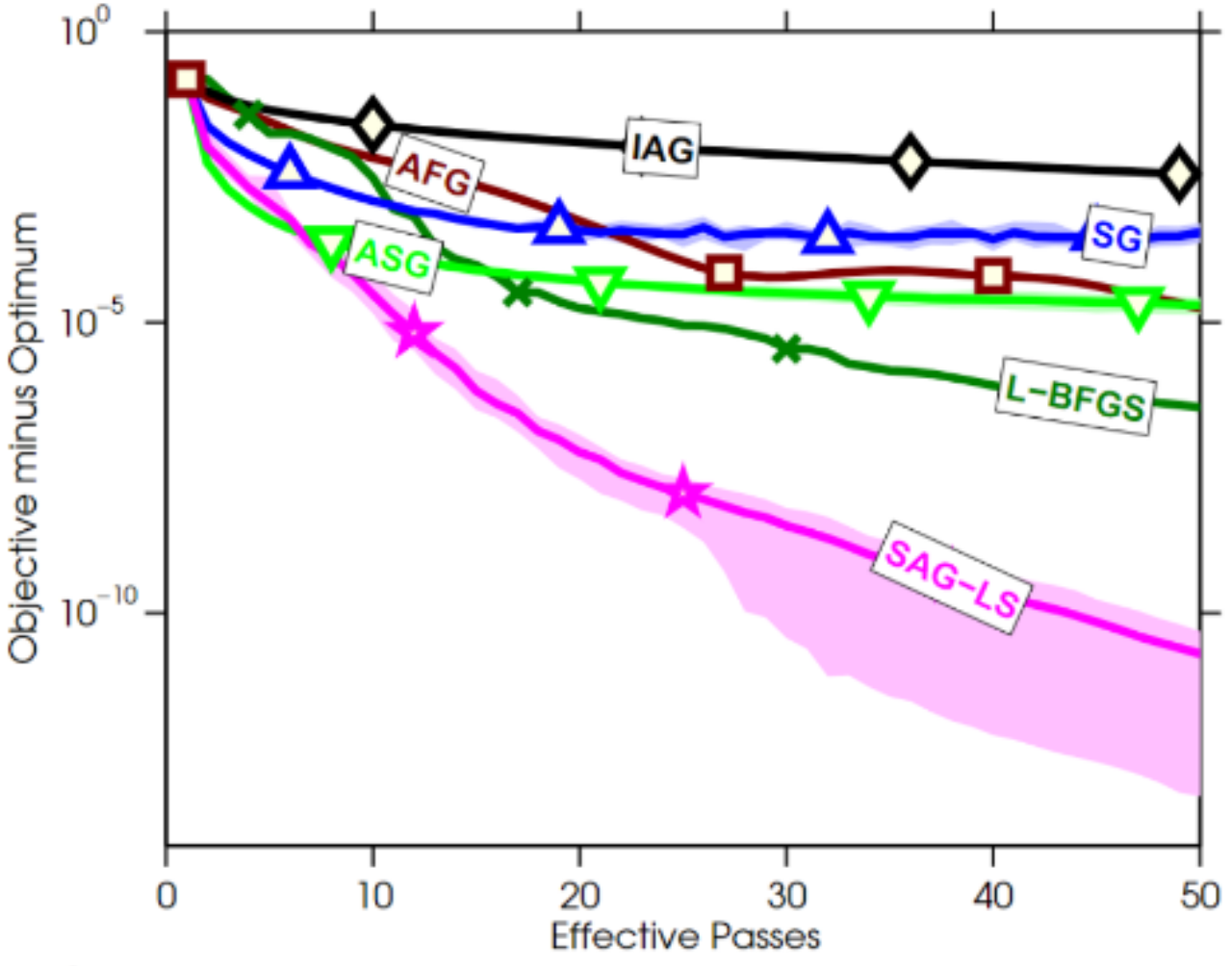
Complicated convergence analysis

Similar rates for many other methods

- stochastic dual coordinate (SDCA); [Shalev-Shwartz, Zhang, 2013]
- stochastic variance reduced gradient (SVRG); [Johnson, Zhang, 2013]
- proximal SVRG [Xiao, Zhang, 2014]
- hybrid of SAG and SVRG, SAGA (also proximal); [Defazio et al, 2014]
- accelerated versions [Lin, Mairal, Harchoui; 2015]
- asynchronous hybrid SVRG [Reddi et al. 2015]
- incremental Newton method, S2SGD and MS2GD, ...

LAS

Performance of SAG



SGD and its variants for Deep Learning

- Next, we will study some variants of SGD used commonly in deep learning.
- Since deep learning involves non-convex optimization, the theoretical convergence results no longer apply!
- However, it is still a very important and evolving research area to obtain the right algorithm for large scale non-convex (deep) learning.
- Three kinds of algorithms:
  - Vanilla Stochastic Gradient Descent
  - Momentum SGD variants: Havy Ball, Nesterov's
  - Adaptive Methods: Adam, AdaGrad, RMSProp

UT DALLAS