Optimization in Machine Learning

Lecture 17: Minibatch, Hybrid and Accelerated Stochastic Gradient Descent

Ganesh Ramakrishnan

Department of Computer Science
Dept of CSE, IIT Bombay
https://www.cse.iitb.ac.in/~ganesh

March, 2025

Outline of next few topics

- Algorithms for Optimization: First Order and thereafter [Done]
- Accelerated Gradient Descent [Done]
- Stochastic Gradient Descent [Started]
- Accelerated Stochastic Variants: Hybrid, Adam, Adagrad, RMSProp, *etc.*
- Generalized/Proximal Gradient Descent
- Constrained Optimization and Projected Gradient Descent

[Recap] Summary after including subgradient descent...

- 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 AGD (SAGD): $\sqrt{\frac{2R^2L}{\epsilon}}$ iterations. Speeds up rate of convergence from $\frac{LR^2}{2T}$ to $\frac{2LR^2}{T^2}$ to match the lower bound of $\frac{3L}{32}\frac{R^2}{(T+1)^2}$
- Smooth + Strongly Convex (SS): $\frac{L}{\mu}\log(\frac{R^2L}{2\epsilon})$ iterations ($O\left(\log(\frac{1}{\epsilon})\right)$ even with heavy ball acceleration but with better factors).
- Subgradient method has convergence rate of $O(\frac{1}{\sqrt{T}})$, and to get $f(x_T^{best}) - f(x_*) \leq \epsilon$, needs $O(\frac{1}{\epsilon^2})$ iterations.

① Subgradient requires similar # of steps BUT with more assumptions on r_t !!

② Also subgradient (descent) analysis is in terms of x_T^{best}

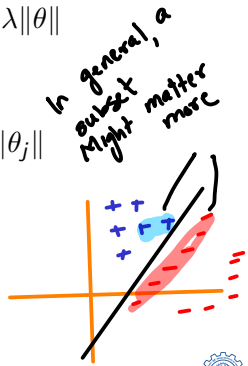
Our running Colab Notebook:
CS769_sms_RunningNotebook_from_GradientDescent-WithUnsupervisedAndSubsetSelection.ipynb
https://colab.research.google.com/drive/1f1y8_ZkL_WS7Kfd7q-KB9Yt2VME1wW35#scrollTo=r50pPWrwLQ=1

[Recap] Loss Functions in Machine Learning

While subgradient descent is somewhat specific to machine learning and so is lot of analysis based on L-smoothness and continuity, can we be even more specific to 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: $L(\theta) = \sum_{i=1}^{n} \frac{r_i}{p_i} \frac{\exp(\theta^T x_i^{a_i})}{\sum_{j=1}^{k} \exp(\theta^T x_i^j)} + \lambda \|\theta\|$

In all these optimization objectives, we sum over all examples



[Recap] 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: $f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$ where f_i is the cost function of the ith observation, n is the total number of training examples
- *Eg:*, $f_i(w) = L(x_i, y_i, w)$ where (x_i, y_i) is i^{th} training point for supervised learning, L is a Loss function
- Evaluating the full gradient ∇f of such an objective involves $O(n)$ computation
- In many applications, n is of the order of a million to a hundreds of millions of data points!

Moreover it is redundant

Notation: In what follows, we use the following new (and old for consistency with previous discussions)

- i as index into the example,
- w (so far x) to represent the function parameter
- t as index into the iteration

[Recap] Stochastic Gradient Descent Algorithm

RECAP:
A simple and strong start: Since previous approaches leverage data redundancy, why not pick a random subset of points?

For times $t = 0, 1, \ldots T$, and stepsizes $\gamma_t > 0$

- Choose $w_0 \in \Re^d$.
- sample $i \in [n]$ uniformly at random
- Compute the vector $g_t^i := \nabla f_i(w_t)$ - a stochastic gradient (a vector of d random variables). We will often drop the index i and call it g_t.
- $w_{t+1} := w_t - \gamma_t \nabla g_t^i = w_t - \gamma_t \nabla f_i(w_t)$ - Only update with the gradient of $f_i = f(x_i)$ instead of the full gradient!

Iteration is n times cheaper than full gradient descent!

[Recap] Unbiasedness in SGD

Can we use convexity for vanilla analysis? What inequality holds in (1) consistently?

$$f(w_t) - f(w_*) \quad \text{----} \quad g_t^T(w_t - w_*) \tag{1}$$

[Recap] Unbiasedness in SGD

because of choosing points "Uniformly at random" (with replacement is default in practice? without replacement is default in theory?)

Can we use convexity for vanilla analysis? What inequality holds in (1) consistently?

Total loss

$$f(w_t) - f(w_*) \quad ? \quad g_t^T (w_t - w_*) \qquad (1)$$

stochastic grad

May not hold since f is total loss but g_t is for 1 pt only

$$\sum_i (f_i(w_t) - f_i(w_*)) \quad ? \quad \sum (g_t)_i^T (w_t - w_*)$$

$$\sum_i (f_i(w_t) - f_i(w_*)) \leq \sum_i (\bar{g}_t^T (w_t - w_*)) \rightarrow \text{Holds for complete loss}$$

If $\bar{g}_t$ were complete gradient

based on $f(w_*) \geq f(w_t) + \bar{g}_t^T (w_* - w_t)$

$$\bar{g}_t = \nabla \left(\sum_i f_i(w_t) \right)$$

$$f_i(w_t) - f_i(w^*) \leq (g_t)_i^T (w_t - w^*)$$

This holds

Loss on i^{th} pt

stochastic gradient

[Recap] Unbiasedness in SGD

Can we use convexity for vanilla analysis? What inequality holds in (1) consistently?

$$f(w_t) - f(w_*) \quad \text{----} \quad g_t^T(w_t - w_*) \tag{1}$$

Assumption is that this g_t is evaluated on a randomly sampled point only

- Ans: $f(w_t) - f(w_*) \leq g_t^T(w_t - w_*)$ may hold or may not hold, depending on how the stochastic gradient g_t turns out.
- We will show (and exploit) that the inequality holds in expectation. For this, we use that by definition, g_t is an unbiased estimate of $\nabla f(w_t)$:

$$\mathbb{E}\left[g_t | w_t = w\right] = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w) = \nabla f(w), \;\; w \in \Re^d$$

Convergence Result for SGD

Theorem

Let $f : \Re^d \to \Re$ be convex and differentiable, w_* a global minimum; furthermore, suppose that $||w_0 - w_*|| \leq R$, and that $\mathbb{E}\left[||g_t||^2\right] \leq B^2$ for all t. Choosing the constant stepsize

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

stochastic gradient descent yields

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[f(w_t)\right] - f(w_*) \leq \frac{RB}{\sqrt{T}}$$

Convergence Result for SGD

Theorem

Let $f : \mathbb{R}^d \to \mathbb{R}$ be convex and differentiable, w_* a global minimum; furthermore, suppose that $||w_0 - w_*|| \leq R$, and that $\mathbb{E}\left[||g_t||^2\right] \leq B^2$ for all t. Choosing the constant stepsize

Lipschitz continuity in expectation

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

Recall this choice motivated from subgradient descent !

stochastic gradient descent yields

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[f(w_t)\right] - f(w_*) \leq \frac{RB}{\sqrt{T}}$$

Result similar to Lipschitz continuity for both gradient and subgradient descent

Convergence Rates for SGD with Strong Convexity

Theorem

Let $f : \Re^d \to \Re$ be differentiable and strongly convex with parameter $\mu > 0$; let w_* be the unique global minimum of f. With decreasing step size specified as

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

stochastic gradient descent yields

$$\mathbf{E}\left[f\left(\frac{2}{T(T+1)}\sum_{t=1}^{T} t.w_t\right) - f(w_*)\right] \leq \frac{2B^2}{\mu(T+1)}$$

where $B^2 := max_{t=1}^{T}\ \mathbb{E}\left[||g_t||^2\right]$

Result is almost the same as for subgradient descent, but in expectation.

Convergence Rates for SGD with Strong Convexity

Theorem

Let $f : \Re^d \to \Re$ be differentiable and strongly convex with parameter $\mu > 0$; let w_* be the unique global minimum of f. With decreasing step size specified as

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

wtd avg

stochastic gradient descent yields

$$\mathbf{E}\left[f\left(\frac{2}{T(T+1)}\sum_{t=1}^{T} t.w_t\right) - f(w_*)\right] \leq \frac{2B^2}{\mu(T+1)}$$

$\sum_{t=1}^{T} t$

Similar to strong convexity + L-continuity

where $B^2 := max_{t=1}^{T}\ \mathbb{E}\left[||g_t||^2\right]$

Result is almost the same as for subgradient descent, but in expectation.

Comparison of GD vs SGD: Convergence Rates

- **Classic GD:** For vanilla analysis, we assumed that $\|\nabla f(w)\|^2 < B_{GD}^2$ for all $w \in \Re^d$, where B_{GD} was a constant. So for sum-of-errors objective:

$$\|\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(w)\|^2 \leq B_{GD}^2 \quad \forall w$$

- **SGD:** With similar upperbound B_{SGD}^2 on *expected* squared norms of stochastic gradients

$$\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(w)\|^2 \leq B_{SGD}^2 \quad \forall w$$

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

- $B_{GD}^2 \approx \|\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(w)\|^2 \leq \frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(w)\|^2 \approx B_{SGD}^2$
- B_{GD}^2 can be smaller than B_{SGD}^2, but comparable, especially if larger mini-batches are used.

Comparison of GD vs SGD: Convergence Rates

- **Classic GD:** For vanilla analysis, we assumed that $\|\nabla f(w)\|^2 < B_{GD}^2$ for all $w \in \Re^d$, where B_{GD} was a constant. So for sum-of-errors objective:

$$\|\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(w)\|^2 \leq B_{GD}^2 \quad \forall w$$

- **SGD:** With similar upperbound B_{SGD}^2 on *expected* squared norms of stochastic gradients

$$\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(w)\|^2 \leq B_{SGD}^2 \quad \forall w$$

Gap somewhat bridged when we use minibatches (will see today)

Hence, by Jensen's inequality for $\|.\|^2$ → le $\|\lambda w_1 + (1-\lambda) w_2\|^2 \leq \lambda \|w_1\|^2 + (1-\lambda)\|w_2\|^2$

- $B_{GD}^2 \approx \|\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(w)\|^2 \leq \frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(w)\|^2 \approx B_{SGD}^2$ → A more liberal L-continuity constant
- B_{GD}^2 can be smaller than B_{SGD}^2, but comparable, especially if larger mini-batches are used.

Comparison of GD vs SGD: Convergence Rates

Method	Assumptions	Full	Stochastic
Subgradient	Convex	$O(\frac{1}{\sqrt{T}})$	$O(\frac{1}{\sqrt{T}})$
Subgradient	Strongly convex	$O(\frac{1}{T})$	$O(\frac{1}{T})$

So using stochastic subgradient, we can solve n times faster!!

Method	Assumptions	Full	Stochastic
Gradient	Convex	$O(\frac{1}{T})$	$O(\frac{1}{\sqrt{T}})$
Gradient	Strongly convex	$O\left(\left(1-\frac{\mu}{L}\right)^T\right)$	$O(\frac{1}{T})$

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

Comparison of GD vs SGD: Convergence Rates

We already noted: $E[\tilde{g}_t | w_t] = g_t$

Based on sampling uniformly at random — Full update

Method	Assumptions	Full	Stochastic
Subgradient	Convex	$O(\frac{1}{\sqrt{T}})$	$O(\frac{1}{\sqrt{T}})$
Subgradient	Strongly convex	$O(\frac{1}{T})$	$O(\frac{1}{T})$

Comparable

So using stochastic subgradient, we can solve n times faster!!

Advantage of SGD: Each iteration is n times faster than the case of full iteration as in GD

Method	Assumptions	Full	Stochastic
Gradient	Convex	$O(\frac{1}{T})$	$O(\frac{1}{\sqrt{T}})$
Gradient	Strongly convex	$O\left(\left(1-\frac{\mu}{L}\right)^T\right)$	$O(\frac{1}{T})$

Less significant gap

Something amiss!

Linear convergence

Var $[\|\tilde{g}_t\| | w_t]$ expected to be high.

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

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!

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!

$$Var\left[\|\tilde{g}_t\| / w_t\right] = \frac{1}{m} Var\left[\|\tilde{g}_t^s\| / w_t\right]$$

- 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.
For times $t = 0, 1, \ldots T$, and stepsizes $\gamma_t > 0$

- Choose $w_0 \in \Re^d$.
- sample $i \in [n]$ uniformly at random
- Sample the vector $g_t \in \partial f_i(w_t)$ - a stochastic sub-gradient.
- $w_{t+1} := w_t - \gamma_t \nabla g_t = w_t - \gamma_t \nabla f_i(w_t)$

Iteration is still n times cheaper than full sub-gradient descent!
We are using an unbiased estimate of a subgradient at each step, $\mathbb{E}[g_t | w_t] \in f(w_t)$.
Convergence in $O\left(\frac{1}{\epsilon^2}\right)$, by using the sub-gradient property at the beginning of the proof, where convexity was applied.

Stochastic SGD

For problems which are not necessarily differentiable, we modify SGD to use a subgradient of f_i in each iteration.

For times $t = 0, 1, \ldots T$, and stepsizes $\gamma_t > 0$

- Choose $w_0 \in \Re^d$.
- sample $i \in [n]$ uniformly at random — Minibatch brings stability by samping a batch of size b at each iteration
- Sample the vector $g_t \in \partial f_i(w_t)$ - a stochastic sub-gradient.
- $w_{t+1} := w_t - \gamma_t \nabla g_t = w_t - \gamma_t \nabla f_i(w_t)$ — n/m

Iteration is still n times cheaper than full sub-gradient descent!

We are using an unbiased estimate of a subgradient at each step, $\mathbb{E}[g_t | w_t] \in f(w_t)$.

Convergence in $O\left(\frac{1}{\epsilon^2}\right)$, by using the sub-gradient property at the beginning of the proof, where convexity was applied.

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! (Armijo conditions offer some science of setting learning rate)

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

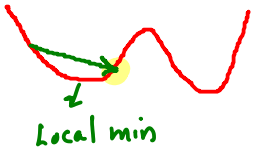
Cons of SGD:

- 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! (Armijo conditions offer some science of setting learning rate)

"descent" is a misnomer here

Homework: Despite these cons, why do ML researchers prefer SGD and its variants?

Pros:



* Might help escape local minima
* Might help generalize better especially when the point(s) randomly picked are adversarial

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! (Armijo conditions offer some science of setting learning rate)

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!

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! (Armijo conditions offer some science of setting learning rate)

Pros of SGD:

- Per iteration cost is n times cheaper compared to GD! (n/b for batch SGD)
- Very fast initial convergence of SGD! Even for non-Deep ML
- Mini-batch gradients often accurate if the batch size is decent and the dataset is massive! More practical for proper "representation"

Identity Crisis: Memorization and Generalization Under Extreme Overparameterization , ICLR 2020 paper
https://openreview.net/forum?id=B1l6y0VFPr

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! Often, We are less concerned about getting (an) exact global minimum than about quickly getting close to (a) global minimum.
- Large gap today between the **theory** of SGD and empirical performance!
- Not always trivial to tune the learning rates – complex rules for learning rates.

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! Often, We are less concerned about getting (an) exact global minimum than about quickly getting close to (a) global minimum.
- Large gap today between the **theory** of SGD and empirical performance!
- Not always trivial to tune the learning rates – complex rules for learning rates.

1) Analysis is largely through expected function value converging to the true value
2) Extra optional reading: Regret bounds in online learning

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.

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* (& local min)
- 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. (& Cyclic rates)
- With a Learning rate decay, SGD is less sensitive to the initial LR.

Improving the convergence rate of SGD

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

Improving the convergence rate of SGD

Bringing in some additive constant factor

Points that bother us

1) How can we derive a principle for reducing variance? Is mini-batch the only way of reducing variance? Is that the only way out?
2) Can we improve the convergence of SGD? Through...
    a) Learning rate schedules?
    b) Acceleration/Momentum
    c) Subset Selection (recall motivation from last class)

Especially in the case of strong convexity, can the gap be reduced?

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

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 $i(t) \in 1, ..., n$
- Perform the update

$$w_{t+1} := w_t - \frac{\gamma_t}{n} \sum_{i=1}^{n} g_t^i \quad \text{where } g_t^i = \begin{cases} \nabla f_i(w_t), & \text{if } i = i(t) \\ g_{t-1}^i, & \text{otherwise} \end{cases}$$

- Randomized/stochastic version of incremental gradient method of Blatt *et. al.* (2008)
- Storage overhead; acceptable in some ML settings:
  - $f_i(w) = \mathcal{L}(y_i, w^T \phi(x_i))$, $\nabla f_i(w) = \nabla \mathcal{L}(y_i, w^T \phi(x_i)) \phi(x_i)$
  - Store only n scalars (since depends only on $w^T \phi(x_i)$)

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 $i(t) \in 1, ..., n$
- Perform the update

$$w_{t+1} := w_t - \frac{\gamma_t}{n} \sum_{i=1}^{n} g_t^i \quad \text{where } g_t^i = \begin{cases} \nabla f_i(w_t), & \text{if } i = i(t) \\ g_{t-1}^i, & \text{otherwise} \end{cases}$$

- Randomized/stochastic version of incremental gradient method of Blatt *et. al.* (2008)
- Storage overhead; acceptable in some ML settings:
  - $f_i(w) = \mathcal{L}(y_i, w^T \phi(x_i))$, $\nabla f_i(w) = \nabla \mathcal{L}(y_i, w^T \phi(x_i)) \phi(x_i)$
  - Store only n scalars (since depends only on $w^T \phi(x_i)$)

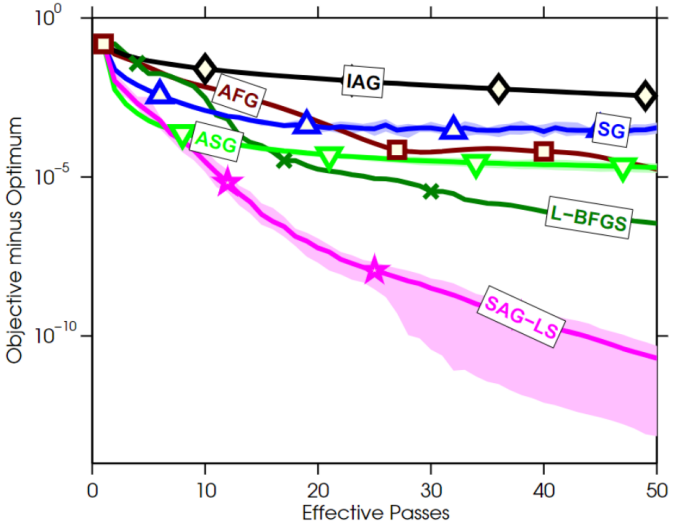
Hybrid SGD: Stochastic Average Gradient

Method	Assumptions	Rate
Gradient	Convex	$O\left(\frac{1}{T}\right)$
Gradient	Strongly Convex	$O\left(\left(1-\frac{\mu}{L}\right)^T\right)$
Stochastic	Strongly Convex	$O\left(\frac{1}{T}\right)$
SAG	Strongly Convex	$O\left(\left(1-min\left[\frac{\mu}{n},\frac{1}{8n}\right]\right)^T\right)$

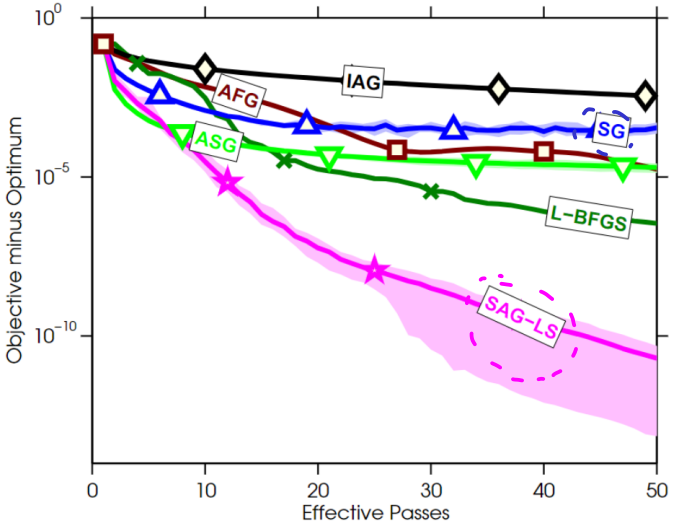
- This speedup is also observed in practice. Convergence analysis is complicated!
- Similar rates for many other methods –
  - stochastic dual coordinate (SDCA) [Shalev-Shwartz, Zhang, 2013]
  - stochastic variance reduced gradient (SVRG) [Johnson, Zhang, 2013]
  - proximal SVRG [Xiao, Zhang, 2014], accelerated versions [Lin, Mairal, Harchoui; 2015]
  - hybrid of SAG and SVRG, SAGA (also proximal) [Defazio et al, 2014]
  - asynchronous hybrid SVRG [Reddi et al. 2015]
  - incremental Newton method, S2SGD and MS2GD
  - ...

Potentially relevant to Projects

Performance of SAG



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

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

Especially learning rates

Note a hidden advantage of convex analysis

1) In convex analysis we often see gaps

2) We then try to bridge such gaps using improvements

(such as smooth and convex lower bound vs. GD ==> lead to accelerared GD

such as (strongly) convex SGD vs. (strongly) convex GD ==> lead to Stochastic Average Descent)

3) Such improved algorithms benefit both the convex and non-convex optimization problems!

4) Also adaptive learning rates motivated by Armijo/Goldsteing/Wolfe conditions

motivate adative methods such as Adam, Adagrad

SGD and Momentum

- Recall SGD:

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t)$$

SGD and Momentum

$i_t = i(t)$ example randomly sampled

- Recall SGD:

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t)$$

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}\left(w_t + \gamma_t(w_t - w_{t-1})\right) + \beta_t(w_t - w_{t-1})$$

Polyak's Heavy Ball (HB) $\Rightarrow \gamma_t = 0$

Nesterov's momentum (NAG) $\Rightarrow \gamma_t = \beta_t$

SGD and Momentum

- Recall SGD:

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t)$$

- Stochastic Momentum: Improves upon SGD via Momentum (similar to the GD case):

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t + \gamma_t(w_t - w_{t-1})) + \beta_t(w_t - w_{t-1})$$

SGD and Momentum

- Recall SGD:

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t)$$

- Stochastic Momentum: Improves upon SGD via Momentum (similar to the GD case):

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t + \gamma_t(w_t - w_{t-1})) + \beta_t(w_t - w_{t-1})$$

- Heavy Ball (HB) Momentum: $\gamma_t = 0$

SGD and Momentum

- Recall SGD:

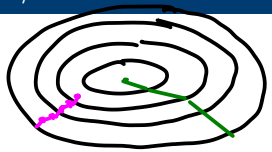
$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t)$$

- Stochastic Momentum: Improves upon SGD via Momentum (similar to the GD case):

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t + \gamma_t(w_t - w_{t-1})) + \beta_t(w_t - w_{t-1})$$

- Heavy Ball (HB) Momentum: $\gamma_t = 0$
- Nesterov's Accelerated Gradient (NAG): $\gamma_t = \beta_t$.

Issues with SGD/Momentum



H_t should be positive semi-definite

$H_t \approx \nabla^2 f(w_t)$ captured through interaction between g_t's

Adapting to behaviour of g_t's

- These techniques are not adaptive: Require extensive tuning for learning rate schedules.
- Without the right LR schedule, convergence can be slow!
- They are also less robust to initialization
- Fix: Adapt learning rate based on gradient information until now.

$g_t \cdot\cdot g_t \cdot\cdot \cdots$

$w^{t+1} = w^t - r\nabla f(w^t) \equiv$ Minimizer of Taylor series Approximation limited to until the gradient term

somewhat related Newton's update: $w^{t+1} = w^t - (\nabla^2 f(w_t))^{-1} \nabla f(w^t) \equiv$ Minimizer of Taylor series Approximation limited to until the second order or Hessian term And accounts for the function's curvature

A SLIGHT DETOUR: LET US UNDERSTAND THE BASIC PREMISE OF SECOND ORDER METHODS (SUCH AS THE NEWTON AND QUASI NEWTON METHODS) THAT INVOKE THE HESSIAN AND ITS APPROXIMATIONS

https://moodle.iitb.ac.in/pluginfile.php/143843/mod_resource/content/1/Basics%20of%20Convex%20Optimization%20My%20Notes.pdf

THIS WILL LEAD TO BETTER APPRECIATION OF MATRICES SUCH AS H TO FOLLOW

BASIC PREMISE OF Second Order Descent and Approximations (including Newton and Quasi-Newton Methods)

Sections 4.5.2 - 4.5.6 of BasicsOfConvexOptimization.pdf

Newton's Algorithm as a Steepest Descent Method

- This choice of $\Delta \mathbf{x}^{k+1}$ corresponds to the direction of steepest descent under the matrix norm[1] induced by the Hessian $\nabla^2 f(\mathbf{x}^k)$: $\Delta \mathbf{x}^{(k)} = \operatorname{argmin} \left\{ \nabla^T f(\mathbf{x}^{(k)}) \mathbf{v} \mid ||\mathbf{v}||_{\nabla^2 f(\mathbf{x}^k)} = 1 \right\}$.
- Equivalently, based on approximating a function around the current iterate $\mathbf{x}^{(k)}$ using a second degree Taylor expansion.

$$Q(\mathbf{x}) \approx \widetilde{f}(\mathbf{x}) = f(\mathbf{x}^{(k)}) + \nabla^T f(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(k)})^T \nabla^2 f(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)})$$

- Convex $f \Rightarrow$

[1] $\left(\mathbf{v}^T \nabla^2 f(\mathbf{x}^k) \mathbf{v}\right)^{\frac{1}{2}}$

Newton's Algorithm as a Steepest Descent Method

- This choice of $\Delta \mathbf{x}^{k+1}$ corresponds to the direction of steepest descent under the matrix norm[1] induced by the Hessian $\nabla^2 f(\mathbf{x}^k)$: $\Delta \mathbf{x}^{(k)} = \operatorname{argmin} \left\{ \nabla^T f(\mathbf{x}^{(k)}) \mathbf{v} \mid ||\mathbf{v}||_{\nabla^2 f(\mathbf{x}^k)} = 1 \right\}$.
- Equivalently, based on approximating a function around the current iterate $\mathbf{x}^{(k)}$ using a second degree Taylor expansion.

$$Q(\mathbf{x}) \approx \widetilde{f}(\mathbf{x}) = f(\mathbf{x}^{(k)}) + \nabla^T f(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(k)})^T \nabla^2 f(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)})$$

- Convex $f \Rightarrow$ convex quadratic approximation. Newton's method is based on solving the approximation exactly
- Setting gradient of quadratic approximation (with respect to $\mathbf{x}$) to $\mathbf{0}$ gives

$$\nabla^T f(\mathbf{x}^{(k)}) + \nabla^2 f(\mathbf{x}^{(k)})(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = 0$$

Assuming $\nabla^2 f(\mathbf{x}^k)$ is invertible, next iterate is $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left(\nabla^2 f(\mathbf{x}^{(k)}) \right)^{-1} \nabla f(\mathbf{x}^{(k)})$

[1] $\left(\mathbf{v}^T \nabla^2 f(\mathbf{x}^k) \mathbf{v} \right)^{\frac{1}{2}}$

Newton's Algorithm as a Steepest Descent Method

Find a starting point $\mathbf{x}^{(0)} \in \mathcal{D}$.
Select an appropriate tolerance $\epsilon > 0$.
repeat

1. Set $\Delta\mathbf{x}^{(k)} = -\left(\nabla^2 f(\mathbf{x}^{(k)})\right)^{-1} \nabla f(\mathbf{x})$.
2. Let $\lambda^2 = \nabla^T f(\mathbf{x}^{(k)}) \left(\nabla^2 f(\mathbf{x}^{(k)})\right)^{-1} \nabla f(\mathbf{x}^{(k)}) \Leftrightarrow$ Directional derivative in the Newton Direction
3. If $\frac{\lambda^2}{2} \leq \epsilon$, **quit**.
4. Set step size $t^{(k)} = 1$. Obtain $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + t^{(k)}\Delta\mathbf{x}^{(k)}$.
5. Set $k = k + 1$.

until

Figure 1: The Newton's method which typically uses a step size of 1. $\Delta\mathbf{x}^{(k)}$ can be shown to be always a Descent Direction (Theorem 83 of notes). For $\mathbf{x} \in \Re^n$, each Newton's step takes $O(n^3)$ time (without using any fast matrix multiplication methods).

Variants of Newtons's Method

- **Special Cases:** When Objective function is a composition of two functions (such as Loss l over some Prediction function m): Gauss Newton Approximation (Section 4.5.4 of BasicsOfConvexOptimization.pdf) and Levenberg-Marquardt (Section 4.5.5)
- **Quasi-Newton Algorithms:** When Hessian inverse $\left(\nabla^2 f(\mathbf{x}^{k+1})\right)^{-1}$ is approximated by a matrix B^{k+1} such that
  - gradient of quadratic approximation $Q(\mathbf{x}^k)$ agrees at $\mathbf{x}^k$ and $\mathbf{x}^{k+1}$
  - B^{k+1} is as close as possible to B^k in some norm (such as the Frobenius norm)

  See BFGS (Section 4.5.6), LBFGS *etc.*

END OF DETOUR TOWARD UNDERSTANDING THE BASIC PREMISE OF SECOND ORDER METHODS (SUCH AS THE NEWTON AND QUASI NEWTON METHODS) THAT INVOKE THE HESSIAN AND ITS APPROXIMATIONS