Optimization Algorithms in Machine Learning

Rishabh Iyer
University of Texas at Dallas

Recap: The General ML Problem

- Given training data $\{(x_1, y_1), \cdots, (x_N, y_N)\}$
- Assume Parameters are w (weights)
- General ML Optimization Problem:

$$\min_{w} \sum_{i=1}^{N} L(x_i, y_i, w) + \lambda R(w) \tag{1}$$

- $R(w)$ is a regularizer (either L1 or L2 regularization)

Gradient Descent

- General ML Optimization Problem:

$$\min_{w} \sum_{i=1}^{N} L(x_i, y_i, w) + \lambda R(w) \tag{2}$$

- Gradient Descent computes the full gradient!
- Update equation: $w_{k+1} = w_k - \alpha_k \sum_{i=1}^{N} \nabla_w L(x_i, y_i, w) - \lambda \nabla_w R(w)$
- What is the problem with gradient descent?

Stochastic Gradient Descent

- Computing full gradient can be time consuming if N is very large!
- Idea of SGD: Compute an approximation of the full gradient and then move in that direction
- Idea: At iteration k, pick a random index i_k and then perform the following update:

$$w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k)$$

- Can be extended to minibatch setting as well.

SGD and Momentum

- Recall SGD:

$$w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k)$$

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

$$w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k + \gamma_t(w_k - w_{k-1})) + \beta_k(w_k - w_{k-1})$$

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

Issue With SGD/Momentum

- 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.

Adaptive Gradient Descent Framework of Algorithms

- Adaptive Methods try to automatically adapt the learning rate.
- Define H_k as a positive semi-definite Matrix (recall the Hessian method?)
- The simplest Adaptive Algorithm called AdaGrad (Duchi et al 2011) is:

$$w_{k+1} = w_k - \alpha_k H_k^{-1} \nabla f_{i_k}(w_k)$$

  where H_k is a positive semi-definite Matrix!
- The simplest definition of H_k is a diagonal matrix (recall we need SGD algorithms to be super-fast, so no matrix inversion possible!)
- Define:

$$H_k = \text{diag}(\{\sum_{i=1}^{k} \eta_i g_i \circ g_i\}^{1/2})$$

Unified Framework of Adaptive Algorithms

- We can unify all adaptive and non-adaptive variants into a single update equation:

$$w_{k+1} = w_k - \alpha_k H_k^{-1} \nabla f_{i_k}(w_k + \gamma_k(w_k - w_{k-1})) + \beta_k H_k^{-1} H_{k-1}(w_k - w_{k-1})$$

- Recall $H_k = \mathrm{diag}(\{\sum_{i=1}^{k} \eta_i g_i \circ g_i\}^{1/2})$. Define $G_k = H_k \circ H_k$ and $D_k = \mathrm{diag}(g_k \circ g_k)$.

	SGD	HB	NAG	AdaGrad	RMSProp	Adam
G_k	I	I	I	$G_{k-1} + D_k$	$\beta_2 G_{k-1} + (1-\beta_2) D_k$	$\frac{\beta_2}{1-\beta_2^k} G_{k-1} + \frac{(1-\beta_2)}{1-\beta_2^k} D_k$
α_k	α	α	α	α	α	$\alpha \frac{1-\beta_1}{1-\beta_1^k}$
β_k	0	β	β	0	0	$\frac{\beta_1(1-\beta_1^{k-1})}{1-\beta_1^k}$
γ	0	0	β	0	0	0

More on ADMM

- Adam is basically HB Momentum + Adaptive.
- Define $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$
- Define $v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k \circ g_k$
- Intuition of m_k and v_k are estimates of first moment (mean) and second moment (uncentered variance) of the gradients.
- Since m_k and v_k are initialized to 0, they are biased towards zero when the decay rates are small. To counter this, they are further normalized by $1 - \beta^k$.
- Define $\hat{m}_k = m_k / (1 - \beta_1^k)$ and $\hat{v}_k = v_k / (1 - \beta_2^k)$.
- The ADAM update is $w_{k+1} = w_k - \alpha_k \hat{m}_k \circ \hat{v}_k^{-1/2}$
- Parameters used in practice: $\beta_1 = 0.9, \beta_2 = 0.999$.

Extensions

Numerous extensions of the above techniques

- AdaMax is an extension of ADAM to use the l_{infty} norm (i.e. max) instead of square.
- NADAM applies Nesterovs momentum instead of HB Momentum to Adaptive Methods.
- ADADelta is an extension of RMSProp to use the RMS operator on the weight differences as well.
- Recent Algorithm (AMSGrad) by Reddi et al (ICLR 2018) which fixes a theoretical error in ADAM (causing it to not converge even for convex functions) simply by ensuring v_t's remain positive!
- See more details to compare the different optimization algorithms (and also what they are) here: `https://ruder.io/optimizing-gradient-descent/`.

Theoretical Results

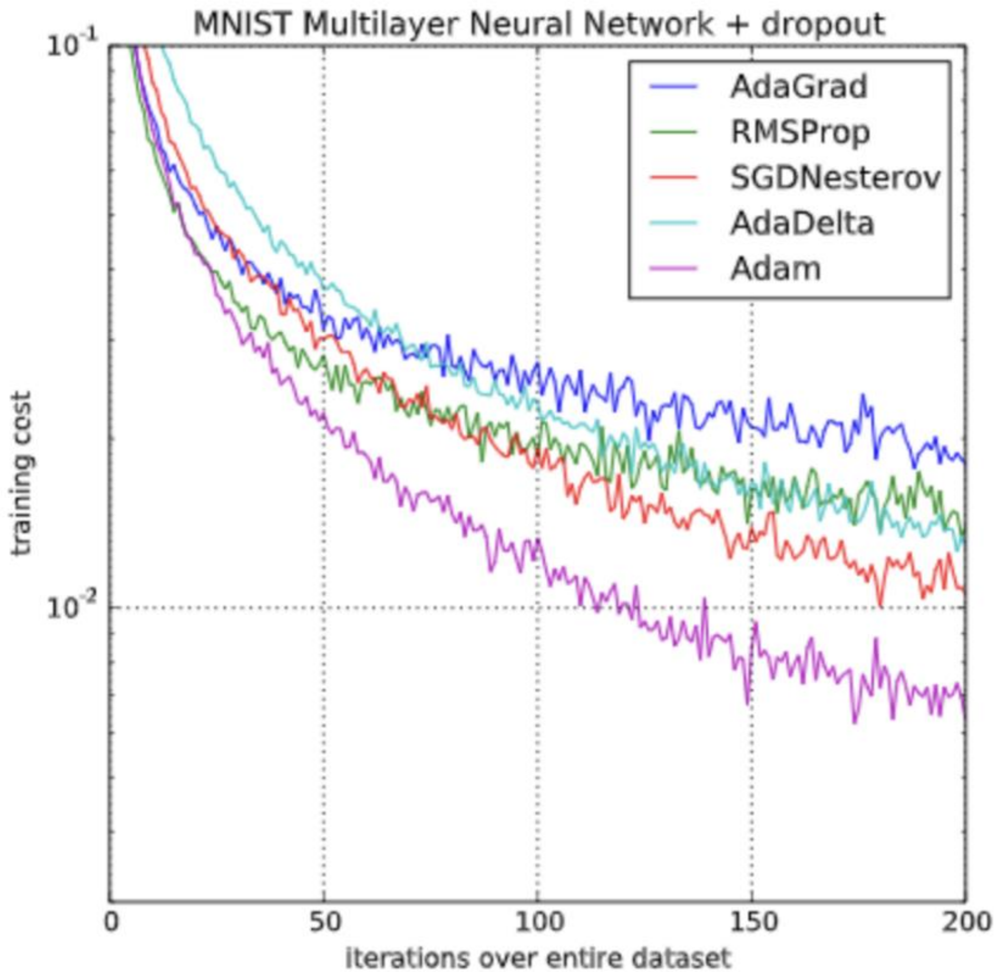
Numerous extensions of the above techniques

- The first theoretical result was shown for AdaGrad. The convergence result there is a *Regret* bound which is common for online algorithms.
- As mentioned above, the paper introducing ADAM actually had a bug in its analysis. The same also holds for RMSProp, AdaDelta and NADAM etc. They do not have **theoretical Regret bounds** backing them.
- Paper introducing AMSGrad showed regret bounds with a modified version of ADAM (and correspondingly RMSProp, NADAM, ...)
- All this only holds for convex functions. No results known for Non-Convex Functions.

Comparison of Different Methods

- AdaGrad (Duchi et al 2011) is one of the most influential papers of the last decade!
- The starting point of numerous new techniques for adaptive methods.
- There is really no one technique that is provably better than the other. Each technique has its own pros and cons!
- In the next few slides, I'll try to put together a few takeaways from some recent papers which have studied this specifically for non-convex optimization.

ADAM Paper



MNIST Multilayer Neural Network + dropout
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam
training cost
iterations over entire dataset
10^{-1}
10^{-2}
0
50
100
150
200

Adaptive vs Non Adaptive Techniques: Comparisons

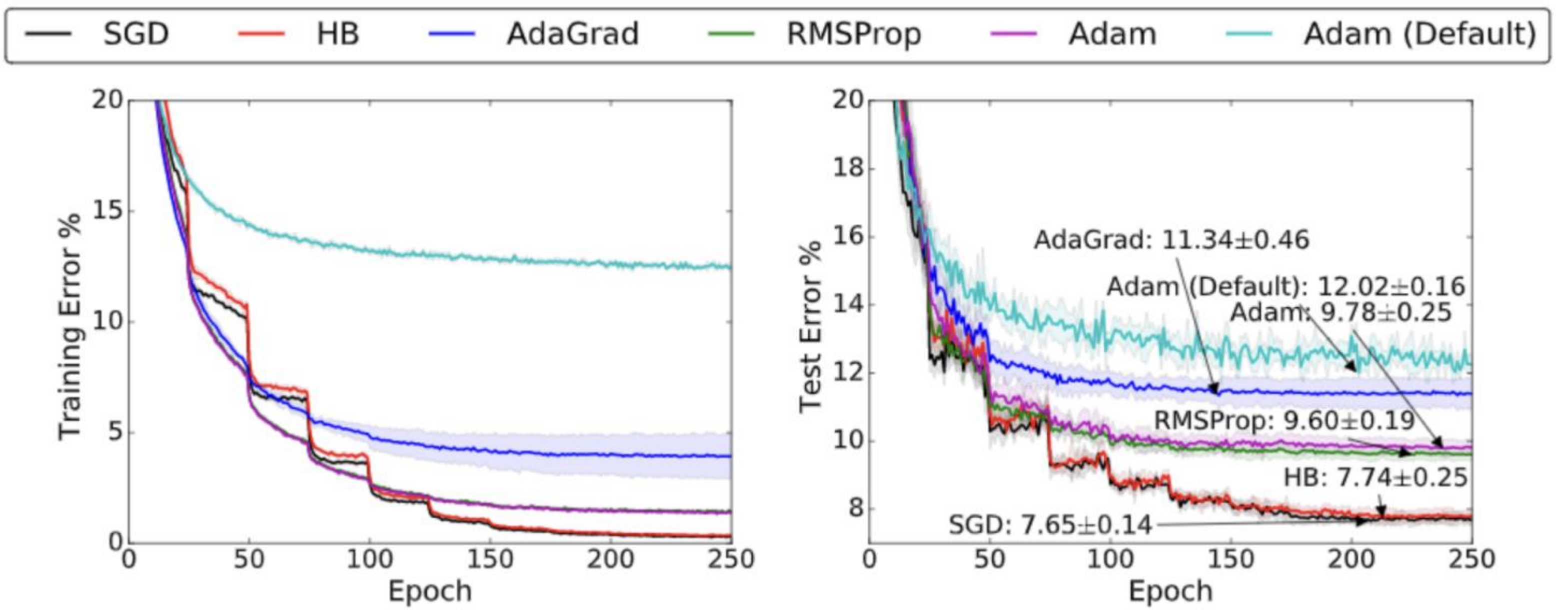
- Benefits of AdaGrad: AdaGrad can significantly improve upon SGD in sparse feature sets! It automatically sets the learning rate, and secondly, automatically updates the learning rates with a decay schedule! Also, it has a per coordinate learning rate!
- In dense settings and particularly in deep models, Adagrad works very poorly because of rapid decay in learning rates
- ADAM, RMSProp, AdaDelta, ... all try to fix this issue!
- In many cases, these adaptive algorithms improve upon SGD in terms of training loss and better/faster convergence!
- Also, momentum generally improves upon the non-momentum variants!
- But....

SGD and Generalization

- But, in Machine Learning, Generalization is more important compared to just Training Loss!
- Recent works (for example, Wilson et al, The Marginal Value of Adaptive Gradient Methodsin Machine Learning) showed a very surprising result!
- Though adaptive gradient methods tend to minimize training loss better, they do so by obtaining more complex and less generalizable solutions!
- They gave a few synthetic examples (particularly in over0parameterized scenarios) where SGD and its variants obtain the less complex solutions but Adaptive variants obtain solutions which do not generalize well!

SGD and Generalization

See Wilson et al, The Marginal Value of Adaptive Gradient Methodsin Machine Learning, NeurIPS 2017



(a) CIFAR-10 (Train) (b) CIFAR-10 (Test)

Please evaluate the course!

eval.utdallas.edu