Optimization Algorithms in Machine Learning

Rishabh Iyer
University of Texas at Dallas

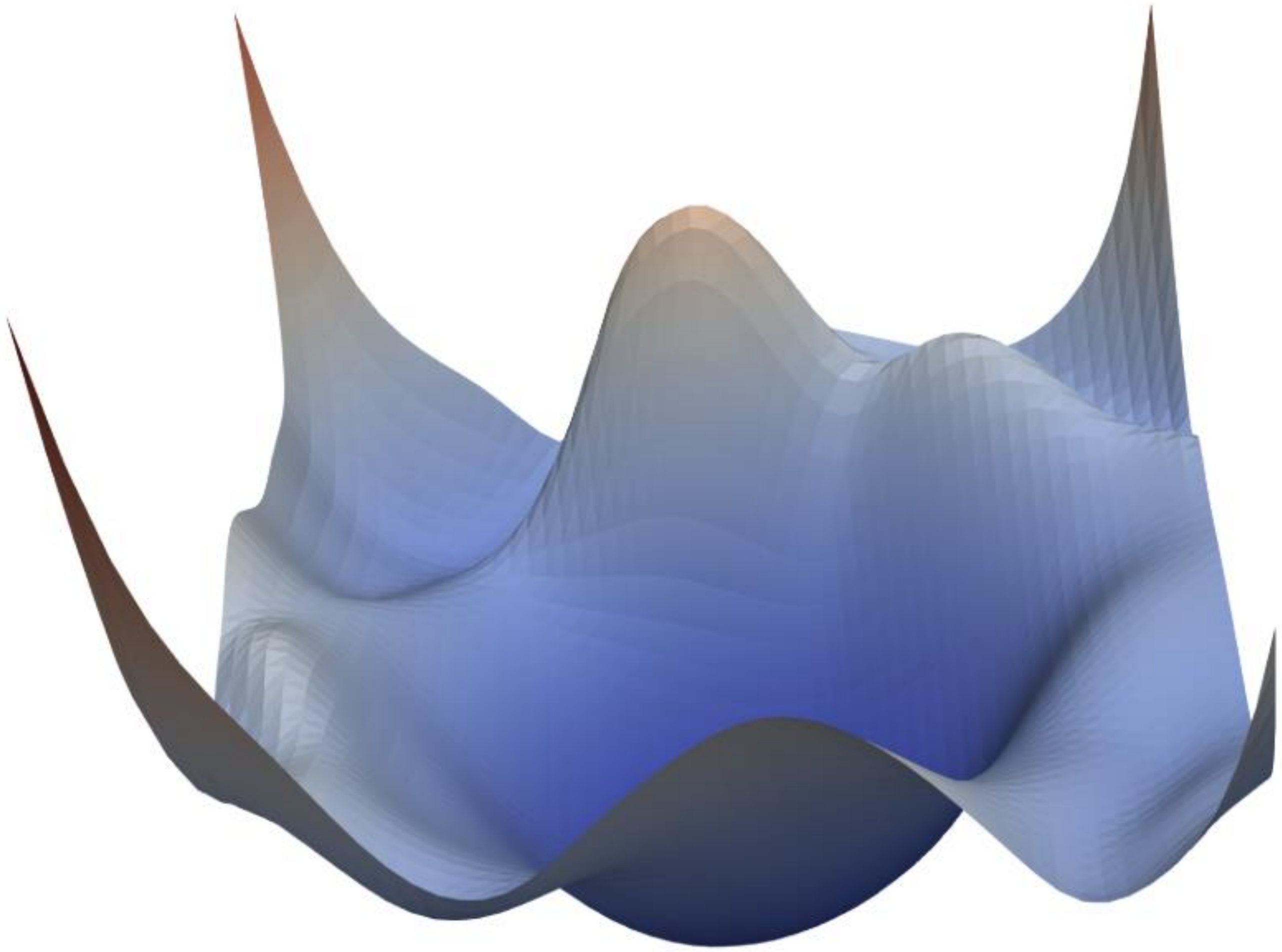
Recap: The General ML Problem

- 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)

Most ML Problems are Non-Convex



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): $\gamma_k = \beta_k$.

Another Variant of Momentum

- Another variant of Momentum used in Deep Learning is From the Gradient Accumulation Perspective
- Define Velocity v_t instead of the gradient

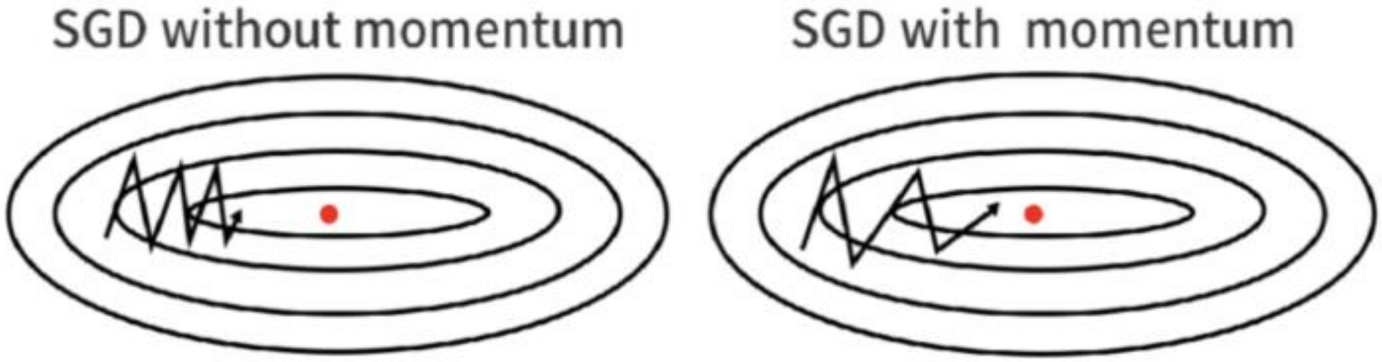
$$v_t = \gamma_t v_{t-1} + (1 - \gamma_t)\nabla L(\theta_t)$$

- In the above, γ_t is the momentum parameter, set to 0.9 for example
- Then we update the model parameters as:

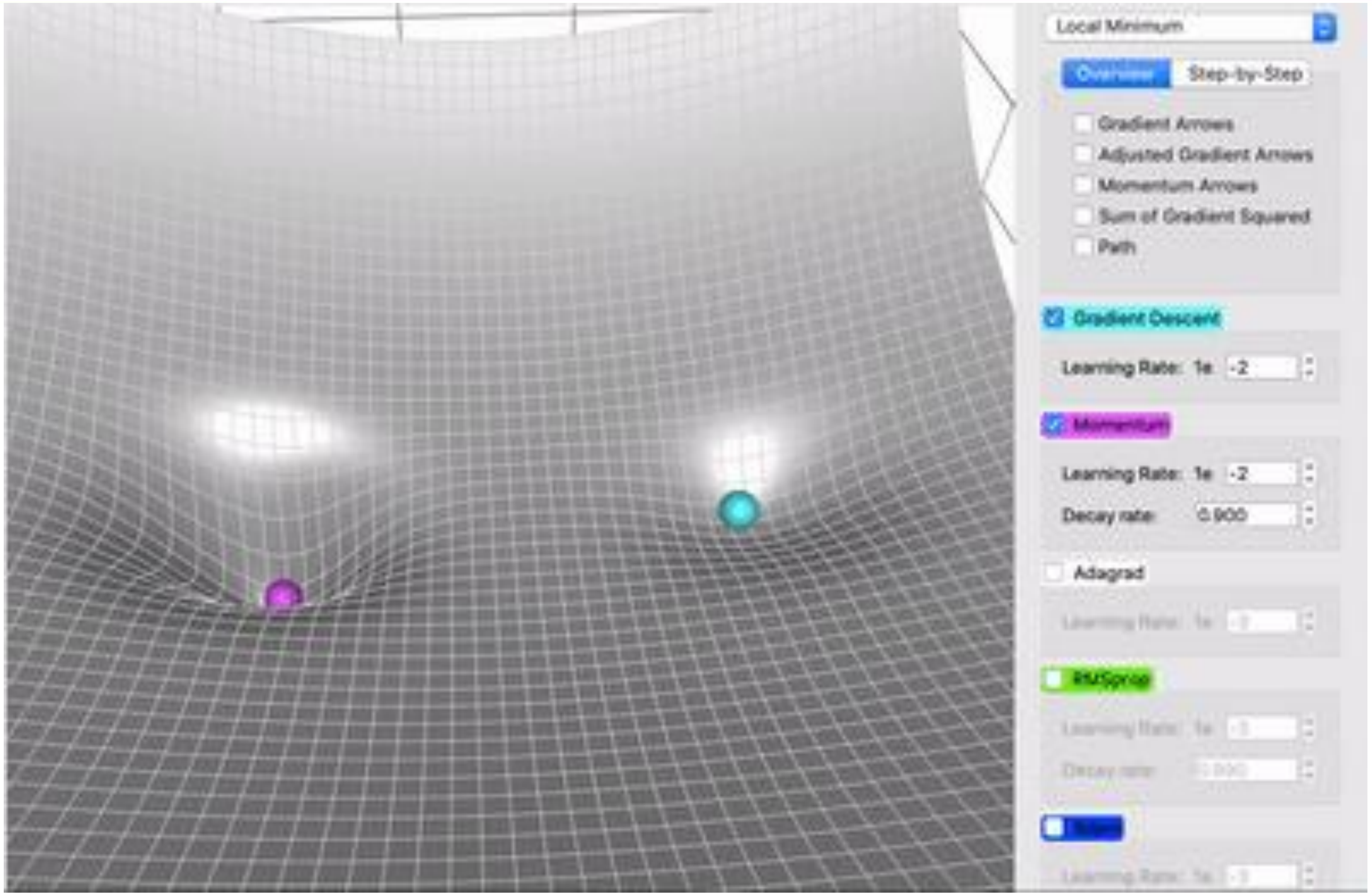
$$\theta_{t+1} = \theta_t - \eta_t \mathrm{v}_\mathrm{t}$$

Advantages of Momentum

- Avoid Local Minima: Momentum accelerates through small dips preventing the optimizer from getting stuck
- Reducing Oscillations: Momentum dampens oscillations especially in high-curvature areas
- Faster Convergence: By “remembering” past gradients, momentum accelerates convergence of the optimizers



SGD vs Momentum



✓ Gradient Descent
Learning Rate: 1e -2

✓ Momentum
Learning Rate: 1e -2
Decay rate: 0.900

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:

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

AdaGrad Algorithm

1️⃣ Accumulate the Squares of Gradients

Adagrad keeps a **running sum of squares** of past gradients for each parameter:

$$G_t = G_{t-1} + g_t^2$$

- G_t is a vector (one value per parameter).
- g_t^2 means element-wise squaring.

2️⃣ Adapt the Learning Rate Per Parameter

The update rule becomes:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

where:

- η = initial learning rate
- ϵ = small constant for numerical stability (like 10^{-8})
- $\odot$ = element-wise multiplication

AdaGrad Algorithm

🧩 Why Adagrad Helps:

- Great for **sparse data** (e.g., NLP, recommendation systems).
- Infrequent features get **larger steps** → faster learning.
- Frequent features slow down → prevent exploding updates.

🚨 Downside of Adagrad:

- G_t **keeps accumulating forever**.
- Eventually, learning rate becomes **too small** → training stalls.

✅ This is why later methods like **RMSProp** and **Adam** modify Adagrad by:

- Using a **moving average** instead of pure accumulation.

TLDR of AdaGrad

Basic Idea of AdaGrad:

- No need to worry about the learning rate
- SGD will diverge with a very large learning rate
- AdaGrad manages to adapt the learning rate so the initial learning rate is not a concern anymore!

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	$\mathrm{G}_{k-1} + \mathrm{D}_k$	$\beta_2 \mathrm{G}_{k-1} + (1-\beta_2)\mathrm{D}_k$	$\frac{\beta_2}{1-\beta_2^k}\mathrm{G}_{k-1} + \frac{(1-\beta_2)}{1-\beta_2^k}\mathrm{D}_k$
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 ADAM

🧠 Motivation

- SGD struggles with noisy gradients and needs careful learning rate tuning.
- Adagrad shrinks learning rates too much over time.
- Momentum improves stability but still uses fixed learning rate.

✅ **Adam combines the best of Momentum + Adagrad.**

More on Adam

1️⃣ First Moment Estimate (Momentum)

Update the **running average of gradients**:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

2️⃣ Second Moment Estimate

Update the **running average of squared gradients**:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

3️⃣ Bias Correction

Correct initial bias in m_t and v_t:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad , \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

4️⃣ Parameter Update

Update parameters using adapted step size:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Extensions

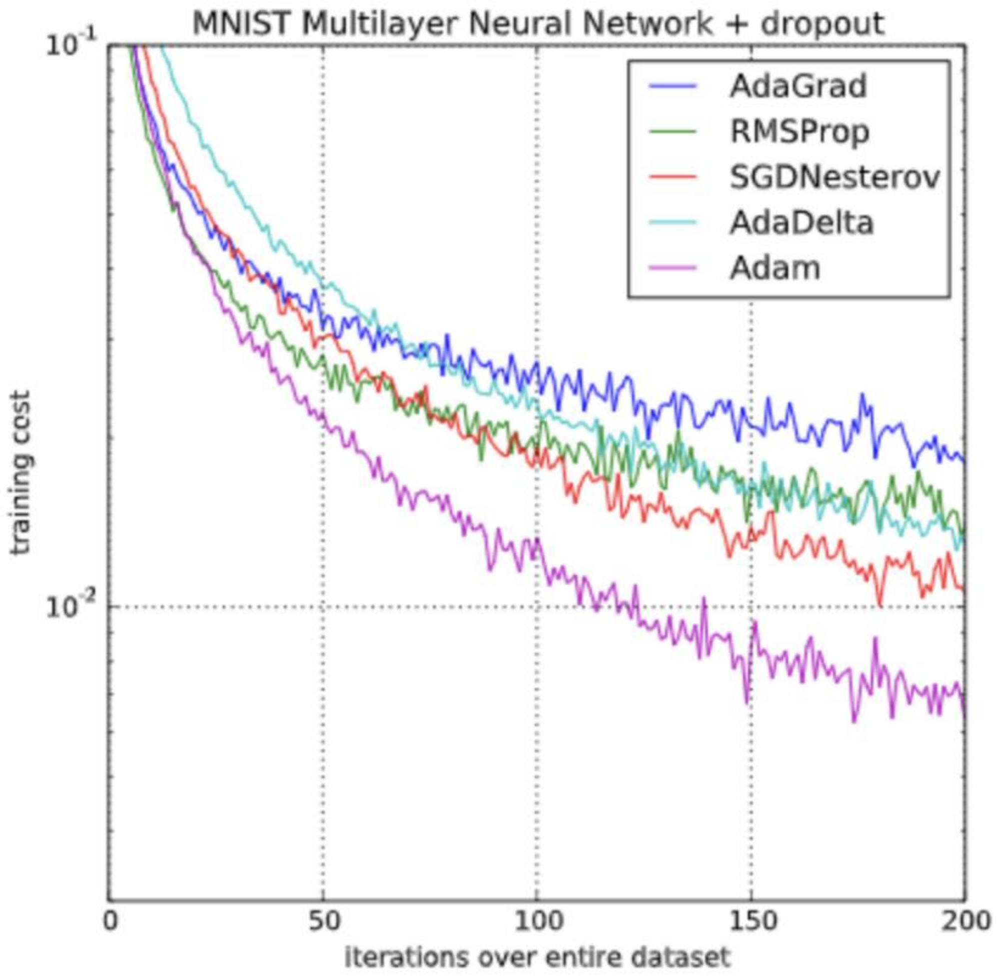
Numerous extensions of the above techniques

- 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/`.

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
10^{-1}
10^{-2}
0
50
100
150
200
iterations over entire dataset

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!

Please evaluate the course!

eval.utdallas.edu