CS5489
Lecture 10.1: Neural Networks and Deep Learning Part III: Optimization

Kede Ma

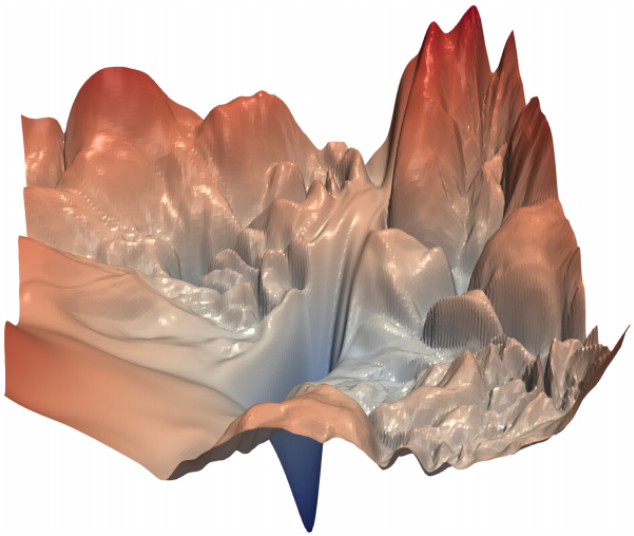
City University of Hong Kong (Dongguan)

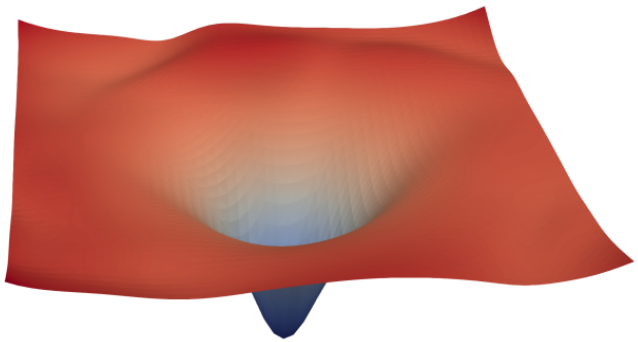
Slide template by courtesy of Benjamin M. Marlin

Outline

Training Deep Neural Networks

- A highly non-convex optimization problem in high dimensional space
  - With lots of plateaus, saddle points, and local optima





- Optimization techniques we have learned
  - Gradient descent
  - Stochastic gradient descent (and its mini-batch version)

Gradient Descent (GD)

Algorithm (Batch) Gradient Descent at Iteration t

Require: Learning Rate $\alpha^{(t)}$

Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$

while stopping criteria not met **do**

  Compute gradient estimate over M examples:

  $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}} \frac{1}{M} \sum_{i=1}^{M} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$

  Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha^{(t)} \hat{\mathbf{g}}$

  $t \leftarrow t + 1$

end while

- Pros: Gradient estimates are stable
- Cons: Need to compute gradients over the entire training set for one update

Stochastic Gradient Descent (SGD)

Algorithm Stochastic Gradient Descent at Iteration t

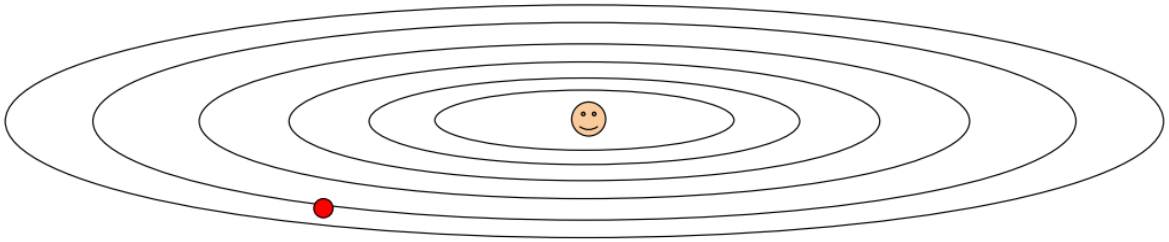
Require: Learning Rate $\alpha^{(t)}$
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Compute gradient estimate:
  $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
  Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha^{(t)} \hat{\mathbf{g}}$
  $t \leftarrow t + 1$
end while

- Pros: Computation time per update does not depend on M, allowing convergence on extremely large datasets
- Cons: Gradient estimates can be noisy. Use mini batches

Outline

More Problems with SGD

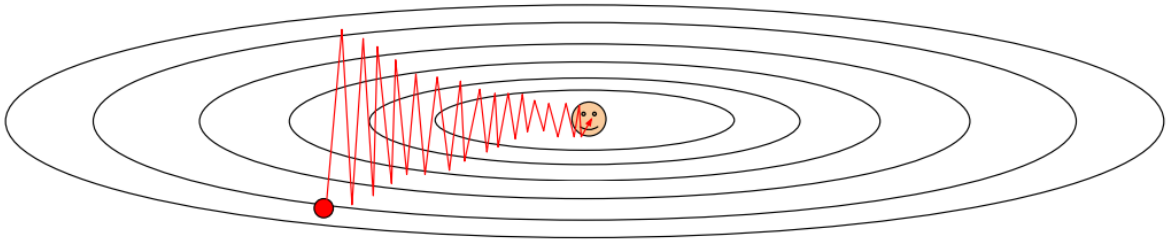
- What if loss changes quickly in one direction and slowly in another (*i.e.*, error surface has high curvature)?



- What does gradient descent do?

More Problems with SGD

- What if loss changes quickly in one direction and slowly in another (*i.e.*, error surface has high curvature)?



- What does gradient descent do?
  - Very slow progress along shallow direction, jittering along steep direction

Momentum

- How do we try to solve this problem?
- Introduce a new variable $\mathbf{v}$, the velocity
- We think of $\mathbf{v}$ as the direction and speed by which the parameters move as the learning dynamics progress
- The velocity is an **exponentially decaying moving average** of the negative gradients

$$\mathbf{v}^{(t+1)} = \rho \mathbf{v}^{(t)} - \alpha^{(t)} \nabla_{\boldsymbol{\theta}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$$

  - $\rho \in [0, 1)$

- Update rule: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{v}^{(t+1)}$

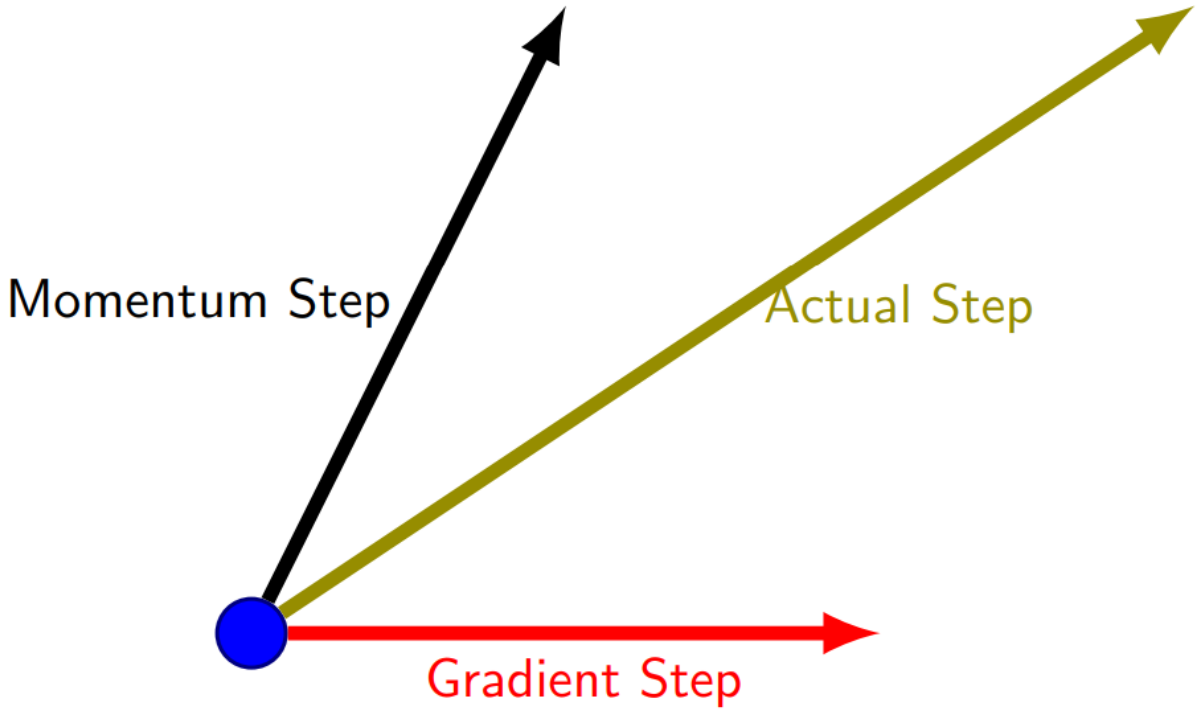
Momentum

- Let's look at the velocity term:

$$\mathbf{v}^{(t+1)} = \rho \mathbf{v}^{(t)} - \alpha^{(t)} \nabla_{\boldsymbol{\theta}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$$

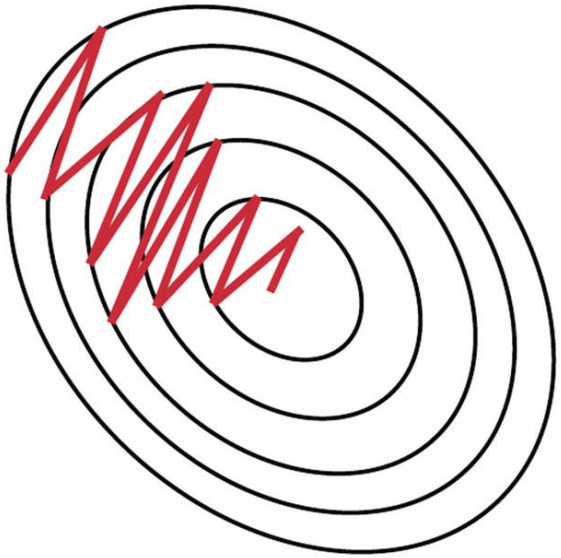
- The velocity **accumulates** the previous gradients
- What is the role of ρ?
  - If ρ is large than $\alpha^{(t)}$, the current update is more affected by the previous gradients
  - Usually values for ρ are set high (e.g, 0.9)

Momentum

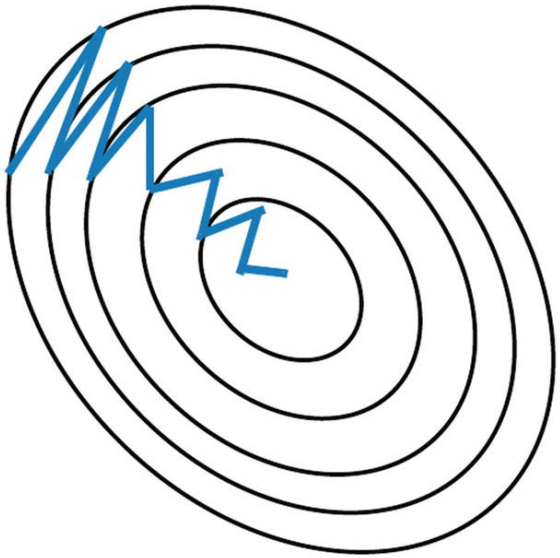


Momentum

- Illustration of how momentum traverses such an error surface better compared to SGD



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

SGD with Momentum

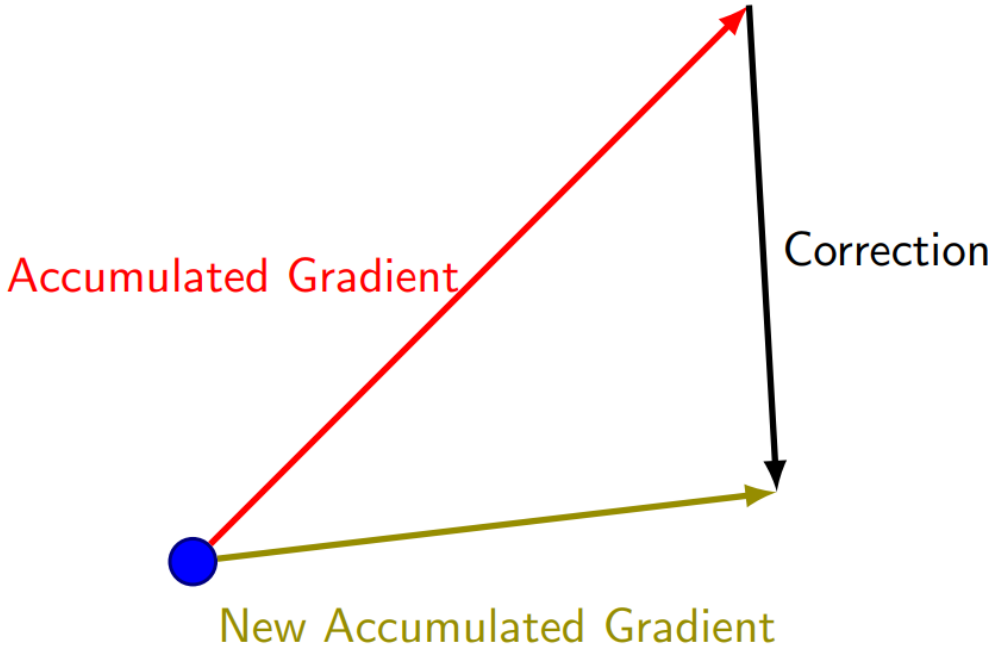
Algorithm SGD with Momentum at Iteration t

Require: Learning Rate $\alpha^{(t)}$
Require: Momentum Parameter ρ
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$
Require: Initial Velocity $\mathbf{v}^{(0)}$

1: **while** stopping criteria not met **do**
2: Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
3: Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
4: Compute the velocity update: $\mathbf{v}^{(t+1)} = \rho \mathbf{v}^{(t)} - \alpha^{(t)} \hat{\mathbf{g}}$
5: Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{v}^{(t+1)}$
6: $t \leftarrow t + 1$
7: **end while**

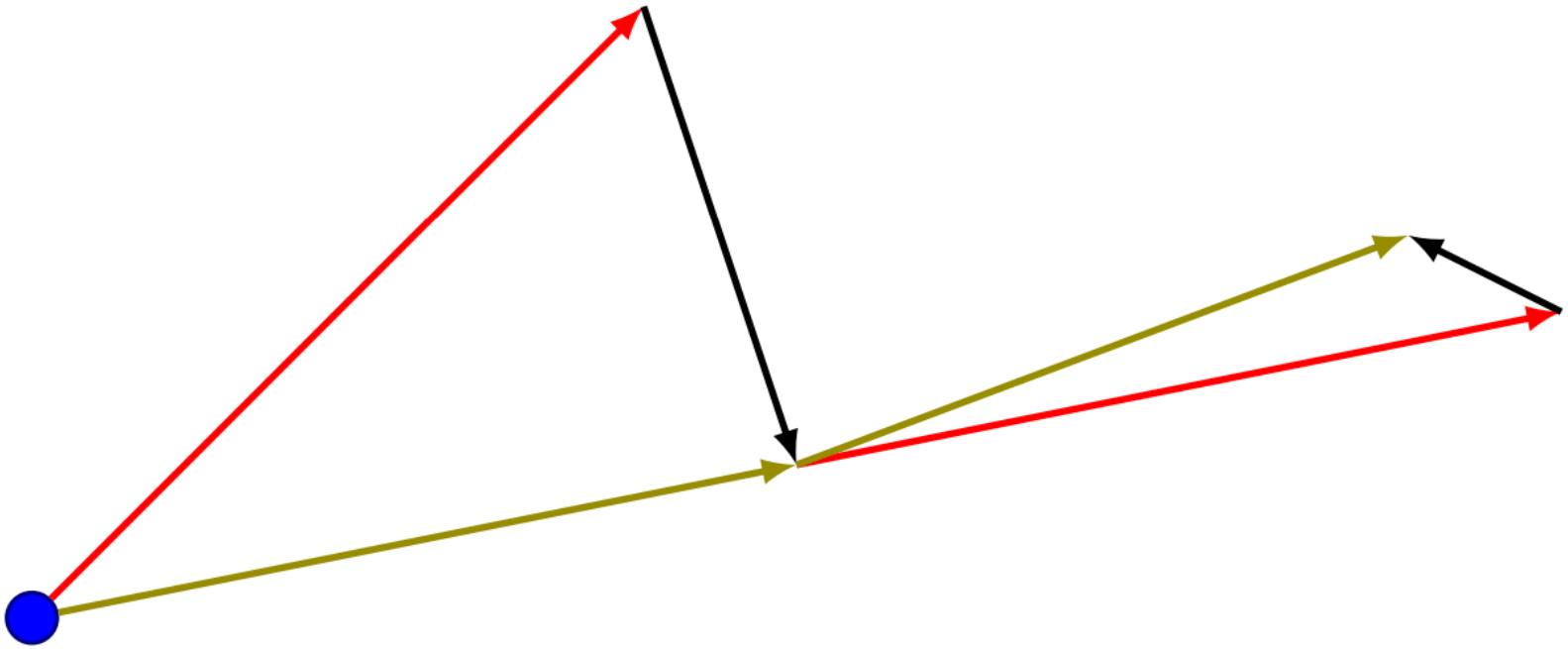
Nesterov Momentum

- Another approach:
  - First take a step in the direction of the accumulated gradient
  - Then calculate the gradient and make a correction



Nesterov Momentum

- Next Step



Nesterov Momentum

- Let's write it out
- Recall the velocity term in the Momentum method:

$$\mathbf{v}^{(t+1)} = \rho\mathbf{v}^{(t)} - \alpha^{(t)}\nabla_{\boldsymbol{\theta}}\ell(f(\mathbf{x}^{(i)};\boldsymbol{\theta}^{(t)}), y^{(i)})$$

- Nesterov Momentum:

$$\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(t)} + \rho\mathbf{v}^{(t)}$$

$$\mathbf{v}^{(t+1)} = \rho\mathbf{v}^{(t)} - \alpha^{(t)}\nabla_{\boldsymbol{\theta}}\ell(f(\mathbf{x}^{(i)};\tilde{\boldsymbol{\theta}}), y^{(i)})$$

- Update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{v}^{(t+1)}$

SGD with Nesterov Momentum

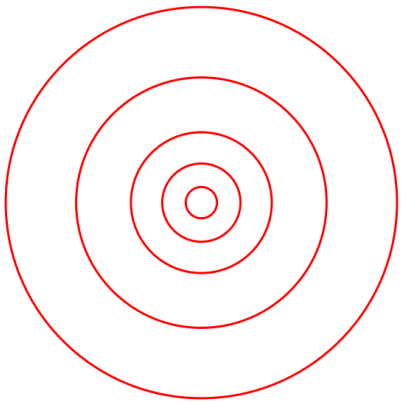
Algorithm SGD with Nesterov Momentum at Iteration t

Require: Learning Rate $\alpha^{(t)}$
Require: Momentum Parameter ρ
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$
Require: Initial Velocity $\mathbf{v}^{(0)}$

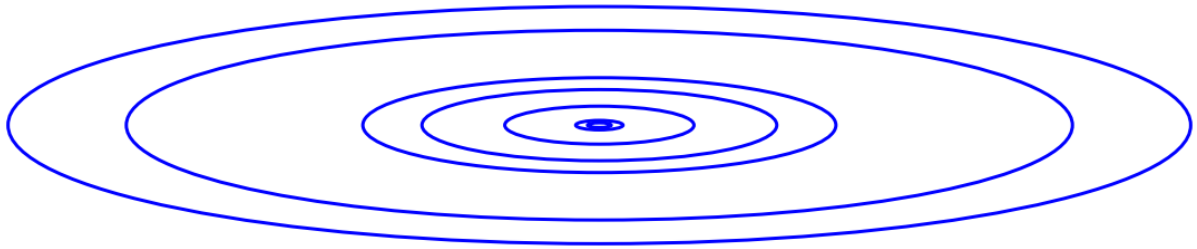
1: **while** stopping criteria not met **do**
2: Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
3: Update parameters temporarily: $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(t)} + \rho \mathbf{v}^{(t)}$
4: Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}} \ell(f(\mathbf{x}^{(i)}; \tilde{\boldsymbol{\theta}}), y^{(i)})$
5: Compute the velocity update: $\mathbf{v}^{(t+1)} = \rho \mathbf{v}^{(t)} - \alpha^{(t)} \hat{\mathbf{g}}$
6: Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbf{v}^{(t+1)}$
7: $t \leftarrow t + 1$
8: **end while**

Adaptive Learning Rate Methods

- Till now we assign the same learning rate to all parameters θ_j's
- If θ_j's vary in importance and speed, why is this a good idea?
- It's probably not!

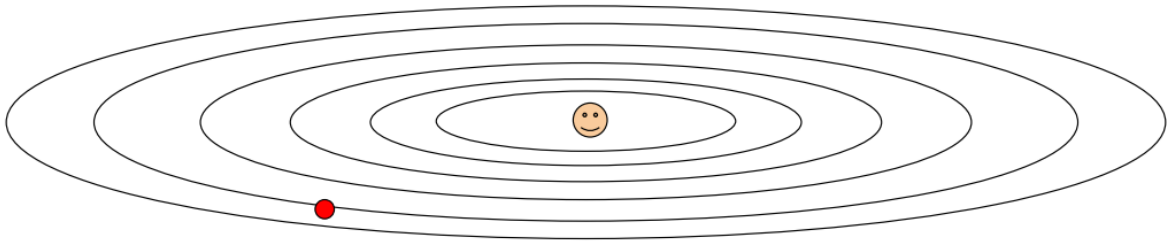


Nice (all features are equally important)



Harder!

AdaGrad



- Idea: Scale the gradient of a model parameter by square root of sum of squares of all its historical values
  - Progress along "steep" directions (with large partial derivatives) is damped
  - Progress along "flat" directions (with small partial derivatives) is accelerated
- What happens to the gradient magnitude over long time?
  - Decays to zero

AdaGrad

Algorithm AdaGrad

Require: Global Learning Rate α
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$

Initialize $\mathbf{r}^{(0)} = 0$
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
  Accumulate: $\mathbf{r}^{(t+1)} = \mathbf{r}^{(t)} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  Compute update: $\Delta\boldsymbol{\theta} = -\frac{\alpha}{\delta + \sqrt{\mathbf{r}^{(t+1)}}} \odot \hat{\mathbf{g}}$
  Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}$
  $t \leftarrow t + 1$
end while

RMSProp: "Leaky AdaGrad"

- AdaGrad is good when the objective is convex
- AdaGrad might shrink the learning rate too aggressively
- We can adapt it to perform better in non-convex settings by accumulating an exponentially decaying average of the gradient
- This is an idea that we use again and again in deep learning
- Has about 8,156 citations on Google Scholar (as of Nov. 2024), but was proposed in a slide in Geoffrey Hinton's Coursera course
  - Had about 7,340 citations on Google Scholar (as of Nov. 2023)

RMSProp

Algorithm RMSProp

Require: Global Learning Rate α
Require: Decay Parameter ρ
Require: Initial Parameter $\boldsymbol{\theta}^{(0)}$

Initialize $\mathbf{r}^{(0)} = 0$
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
  Accumulate: $\mathbf{r}^{(t+1)} = \rho \mathbf{r}^{(t)} + (1-\rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  Compute update: $\Delta\boldsymbol{\theta} = -\frac{\alpha}{\delta + \sqrt{\mathbf{r}^{(t+1)}}} \odot \hat{\mathbf{g}}$
  Apply update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}$
  $t \leftarrow t + 1$
end while

Adam: ADAptive Moments

- Adam is currently THE default optimization algorithm for training deep neural networks
- Introduced by Kingma and Ba in 2015 and quickly accumulated more than $200,240$ Google Scholar Citations as of Nov. 2024
- Adam is like RMSProp with Momentum but with bias correction terms for the first and second moments

Adam

Algorithm Adam

Require: Global Learning Rate: α (set to 0.0001), Decay Rates: ρ_1 (set to 0.9), ρ_2 (set to 0.9), and Initial Parameter: $\boldsymbol{\theta}^{(0)}$

Initialize moment variables $\mathbf{s}^{(0)} = 0$ and $\mathbf{r}^{(0)} = 0$
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)})$
  Update: $\mathbf{s}^{(t+1)} = \rho_1 \mathbf{s}^{(t)} + (1 - \rho_1)\hat{\mathbf{g}}$
  Update: $\mathbf{r}^{(t+1)} = \rho_2 \mathbf{r}^{(t)} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  Correct biases: $\hat{\mathbf{s}} = \frac{\mathbf{s}^{(t+1)}}{1-\rho_1^{t+1}}$ and $\hat{\mathbf{r}} = \frac{\mathbf{r}^{(t+1)}}{1-\rho_2^{t+1}}$
  Compute and apply update: $\Delta\boldsymbol{\theta} = -\alpha \frac{\hat{\mathbf{s}}}{\delta + \sqrt{\hat{\mathbf{r}}}}$, $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}$
  $t \leftarrow t + 1$
end while

AdamW: ADAptive Moments with Weight Decay

- AdamW is an extension of Adam, and has gained popularity, especially in training Transformer-based deep models
- Weight decay (a.k.a., ℓ_2-regularization and ridge regularization) is a regularization technique to prevent overfitting
  - It is just $\|\boldsymbol{\theta}\|_2^2$
- AdamW applies weight decay as a separate regularization term during the update step, decoupling it from the adaptive learning rate mechanism
- AdamW addresses the issue of large weights being incorrectly penalized by weight decay
- AdamW provides improved regularization and better generalization compared to Adam

AdamW

Algorithm Adam Versus AdamW

Require: Global Learning Rate: α (set to 0.0001), Decay Rates: ρ_1 (set to 0.9), ρ_2 (set to 0.9), Weight Decay: λ, and Initial Parameter: $\boldsymbol{\theta}^{(0)}$

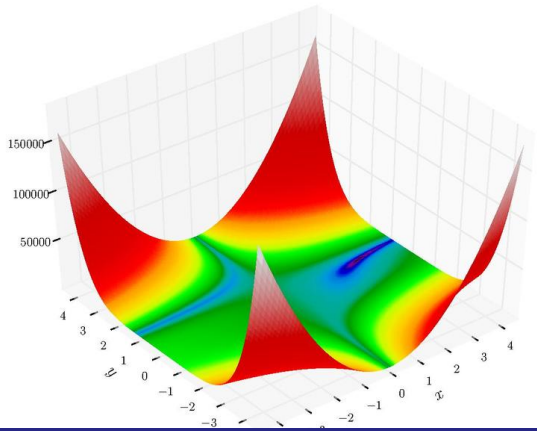
Initialize moment variables $\mathbf{s}^{(0)} = 0$ and $\mathbf{r}^{(0)} = 0$
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, y^{(i)})$ from the training set
  Compute gradient estimate: $\hat{\mathbf{g}} = \nabla_{\boldsymbol{\theta}} \ell(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(t)}), y^{(i)}) + \lambda \boldsymbol{\theta}^{(t)}$
  Update: $\mathbf{s}^{(t+1)} = \rho_1 \mathbf{s}^{(t)} + (1 - \rho_1)\hat{\mathbf{g}}$
  Update: $\mathbf{r}^{(t+1)} = \rho_2 \mathbf{r}^{(t)} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  Correct biases: $\hat{\mathbf{s}} = \frac{\mathbf{s}^{(t+1)}}{1-\rho_1^{t+1}}$ and $\hat{\mathbf{r}} = \frac{\mathbf{r}^{(t+1)}}{1-\rho_2^{t+1}}$
  Compute and apply update: $\Delta\boldsymbol{\theta} = -\alpha \frac{\hat{\mathbf{s}}}{\delta + \sqrt{\hat{\mathbf{r}}}} - \lambda \boldsymbol{\theta}^{(t)}$, $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}$
  $t \leftarrow t + 1$
end while

Optimization Example

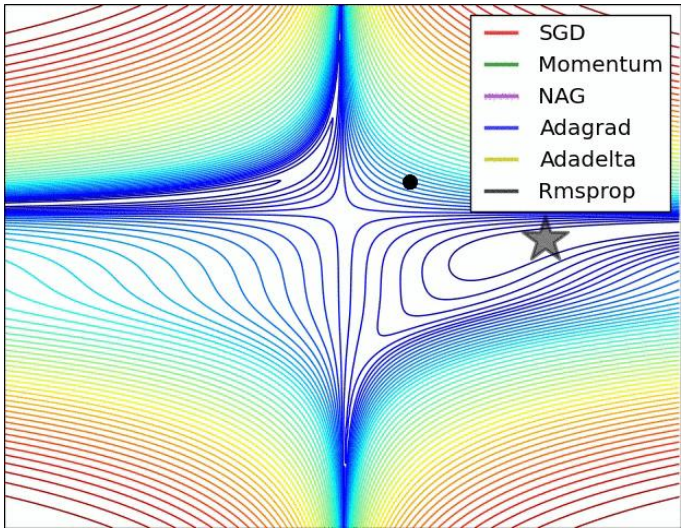
- Beale function $f : \mathbb{R}^2 \mapsto \mathbb{R}$

$$f(\mathbf{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$$

  - **Description**: The Beale function is nonconvex and multimodal, with sharp peaks at the corners of the input domain
  - **Input domain**: The function is usually evaluated on the square $x_j \in [-4.5, 4.5]$, for all $j = 1, 2$
  - **Global optimum**: $f(\mathbf{x}^\star) = 0$ at $\mathbf{x}^\star = (3, 0.5)$; the Beale function exhibits flat plateaus near $\mathbf{x}^\star$, posing challenges for gradient-based optimization



Optimization Example

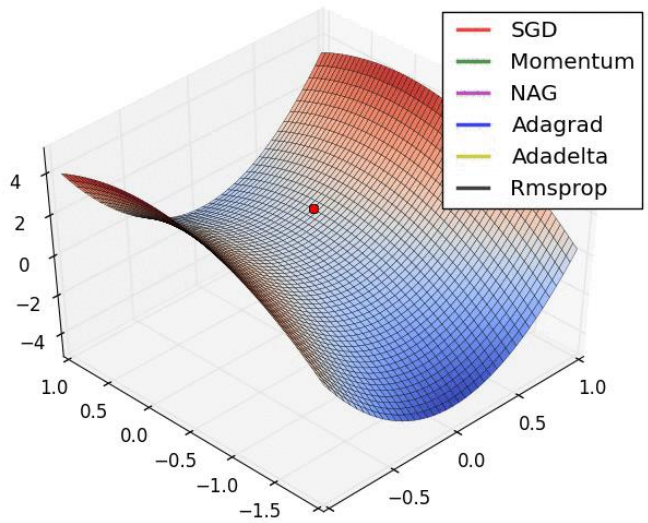


https://ruder.io/optimizing-gradient-descent/

- AdaGrad, RMSProp, and AdaDelta (a simplified version of Adam) almost immediately head off in the right direction and converge similarly fast
- Momentum and NAG (Nesterov momentum) are led off-track
  - NAG, however, is quickly able to correct its course by looking ahead and heads to the minimum

Optimization Example

- Behaviours of the algorithms at a saddle point
  - I.e., a point where one dimension has a positive curvature, while the other dimension has a negative curvature



- SGD, Momentum, and NAG find it difficulty to break symmetry
  - The two latter eventually manage to escape the saddle point
- AdaGrad, RMSProp, and AdaDelta quickly head down the negative slope

Other Emerging Optimization Techniques

- AdaBelief: Unlike Adam, AdaBelief divides by the exponentially decaying moving average of variance of gradients
  - In Adam: $\mathbf{r}^{(t+1)} = \rho_2 \mathbf{r}^{(t)} + (1-\rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
  - In AdaBelief: $\mathbf{r}^{(t+1)} = \rho_2 \mathbf{r}^{(t)} + (1-\rho_2)(\hat{\mathbf{g}} - \mathbf{s}^{(t+1)})^2$
- AdaHessian: Second-order optimization technique that incorporates the curvature of the loss function via adaptive estimates of the Hessian diagonals
  - Admits efficient estimation through Hessian vector product