Deep Learning for Computer Vision

Gradient Descent and Variants

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

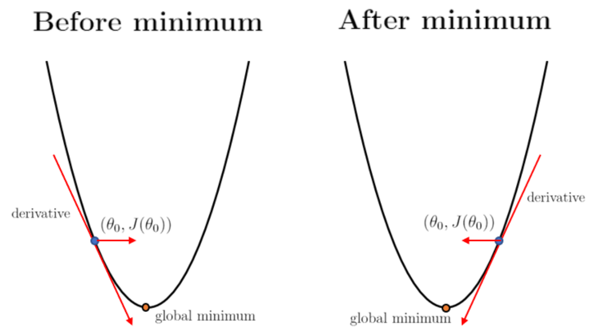
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Review: Gradient Descent (GD)

- Optimization algorithm used to find minima of a given differentiable function
- At each step, parameters (θ) are pushed in negative direction of gradient of a cost function ($J(\theta\ x, y)$, in figure alongside) w.r.t parameters

$$\theta_{new} = \theta_{old} - \alpha \Delta \theta_{old}$$

where α is learning rate



Since the derivative is negative, if we subtract the derivative from θ_0, it will increase and go closer the minimum.

Since the derivative is positive, if we subtract the derivative from θ_0, it will decrease and go closer the minimum.

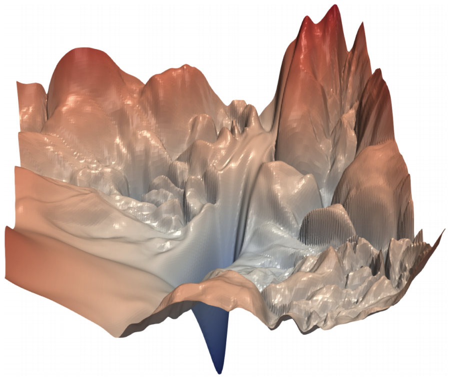
Credit: Albert Lai

Gradient Descent Algorithm

```
Require: Learning rate $\alpha$, initial parameters $\theta_t$, training dataset $\mathcal{D}_{tr}$
1: while stopping criterion not met do
2:     Initialize parameter updates $\Delta\theta_t = 0$
3:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ do
4:         Compute gradient using backpropagation $\nabla_{\theta_t}\mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
5:         Aggregate gradient $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t}\mathcal{L}$
6:     end for
7:     Apply update $\theta_{t+1} = \theta_t - \alpha\frac{1}{|\mathcal{D}_{tr}|}\Delta\theta_t$
8: end while
```

Error Surface of Neural Networks

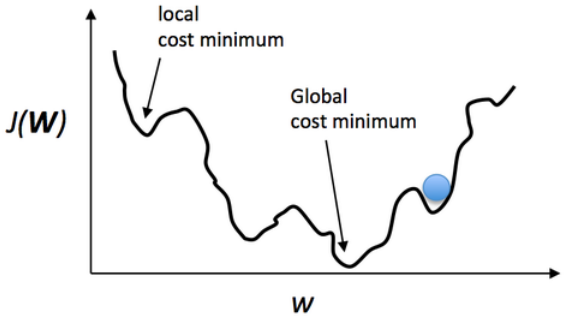
Visualization of error surface of a neural network (ResNet-56)



Credit: Li et al, Visualizing the Loss Landscape of Neural Nets, NeurIPS 2018

Local Minima

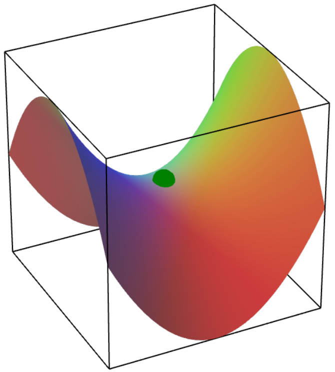
- Unlike convex objective functions that have a global minimum, non-convex functions as in deep neural networks have multiple local minima[1]



[1] Choromanska et al, The Loss Surface of Multilayer Nets, AISTATS 2015

Saddle Points

- Local maximum along one cross-section of cost function and local minimum along another
- Though it isn't a local minimum, gradient is zero (or almost close to zero) $\implies$ gives impression of convergence
- Though local minima are prevalent in lower-dimensional spaces, saddle points become more common in higher-dimensional spaces



Dauphin et al, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, NeurIPS 2014

Gradient Descent Traversal

Let us see a GD traversal example!

Notice anything interesting?

Credit: Mitesh Khapra, IIT Madras

Gradient Descent Traversal

Let us look at another example with a different initial position
HINT: Pay attention to how the relative position of parameters changes

Credit: Mitesh Khapra, IIT Madras

Gradient Descent Traversal

Top view of previous error surface represented as contours

Notice how parameter updates are smaller at points where gradient of error surface is small and larger at points where gradient is large.

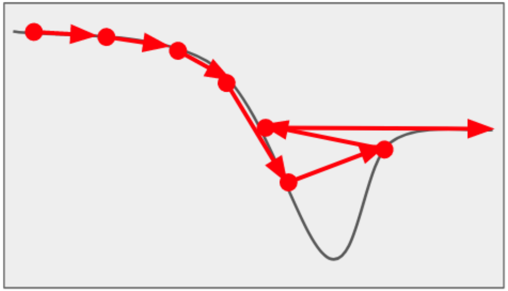
Credit: Mitesh Khapra, IIT Madras

Plateaus and Flat Regions

- They constitute portions of error surface where gradient is highly non-spherical
- Gradient descent spends a long time traversing in these regions as the updates are small
- Can we expedite this process?

Plateaus and Flat Regions

- They constitute portions of error surface where gradient is highly non-spherical
- Gradient descent spends a long time traversing in these regions as the updates are small
- Can we expedite this process?
- How about increasing the learning rate?
- Though traversal becomes faster in plateaus, there is a risk of divergence



Momentum-based GD

- **Intuition:** With increasing confidence, increase step size; and with decreasing confidence, decrease step size
- Weight update given by:

$$v_t = \overset{\text{Momentum Term}}{\boxed{\gamma v_{t-1}}} + \alpha \nabla_{\theta_t} \mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$$

$$\theta_{t+1} = \theta_t - v_t$$

Momentum-based GD

- **Intuition:** With increasing confidence, increase step size; and with decreasing confidence, decrease step size
- Weight update given by:

$$v_t = \overset{\text{Momentum Term}}{\boxed{\gamma v_{t-1}}} + \alpha \nabla_{\theta_t} \mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$$

$$\theta_{t+1} = \theta_t - v_t$$

From these equations, can you see how momentum possibly avoids divergence at 5th and 6th steps of previous figure?

Momentum-based GD

- **Intuition:** With increasing confidence, increase step size; and with decreasing confidence, decrease step size
- Weight update given by:

$$v_t = \underbrace{\boxed{\gamma v_{t-1}}}_{} + \alpha \nabla_{\theta_t} \mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$$

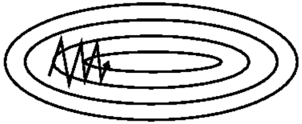
Momentum Term

$$\theta_{t+1} = \theta_t - v_t$$

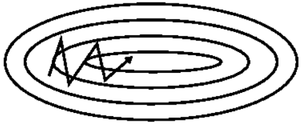
From these equations, can you see how momentum possibly avoids divergence at 5^{th} and 6^{th} steps of previous figure?

After 5^{th} step, momentum and gradient terms are in opposite directions

Momentum-based GD



Without momentum



With momentum

- Damps step sizes along directions of high curvature, yielding a larger effective learning rate along the directions of low curvature.
- Larger the γ, more the previous gradients affect the current step.
- Generally, γ is set to 0.5 until initial learning stabilizes and then increased to 0.9 or higher (practitioners often default it to 0.9/0.95 these days)

Momentum-based GD: Algorithm

Require: Learning rate α, momentum parameter γ, initial parameters θ_t, training dataset $\mathcal{D}_{tr}$

```
 1: Initialize $v_{t-1} = 0$
 2: while stopping criterion not met do
 3:     Initialize weight updates $\Delta\theta_t = 0$
 4:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ do
 5:         Compute gradient using backpropagation $\nabla_{\theta_t}\mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
 6:         Aggregate weight updates: $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t}\mathcal{L}$
 7:     end for
 8:     Update velocity: $v_t = \gamma v_{t-1} + \alpha\Delta\theta_t$
 9:     Apply update: $\theta_{t+1} = \theta_t - v_t$
10: end while
```

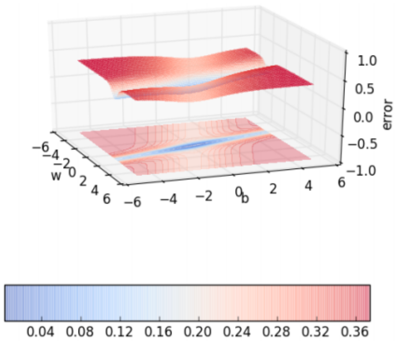
Momentum-based GD: Illustration

Convergence of Momentum vs Gradient Descent

Momentum performance was better than GD. But is the speed always good?

Credit: Mitesh Khapra, IIT Madras

Is Momentum Detrimental?



Any observations?

Credit: Mitesh Khapra, IIT Madras

Issues with Momentum

- Momentum-based GD oscillates around minima before eventually reaching it
- Even then, it converges faster than vanilla GD!
    - After 100 iterations, momentum-based GD has error of 0.00001, whereas vanilla GD is still at error of 0.36
- Nonetheless, it is wasting time in oscillations; how can we reduce oscillations?

Source: Mitesh Khapra, IIT Madras

Nesterov Accelerated Momentum

- Based on Nesterov's Accelerated Gradient Descent published in 1983[2]; re-introduced by Sutskever in ICML'13[3]
- Key idea: **Look before you leap**
- Assess how gradient changes after taking a step of **momentum**, γv_t, and use this to get better estimate of parameters

[2]Nesterov, A method of solving a convex programming problem with convergence rate O($1/k^2$), Soviet Mathematics Doklady, 1983, 27:2, pp 372–376

[3]Sutskever et al, On the importance of initialization and momentum in deep learning, ICML 2013, Vol 28, pp 1139—1147

Nesterov Accelerated Momentum

- Based on Nesterov's Accelerated Gradient Descent published in 1983[2]; re-introduced by Sutskever in ICML'13[3]
- Key idea: **Look before you leap**
- Assess how gradient changes after taking a step of **momentum**, γv_t, and use this to get better estimate of parameters
- Weight update given by:

$$v_t = \gamma v_{t-1} + \alpha \nabla_{\tilde{\theta}_t} \mathcal{L}(\theta_t - \gamma v_{t-1}; x^{(i)}, y^{(i)})$$

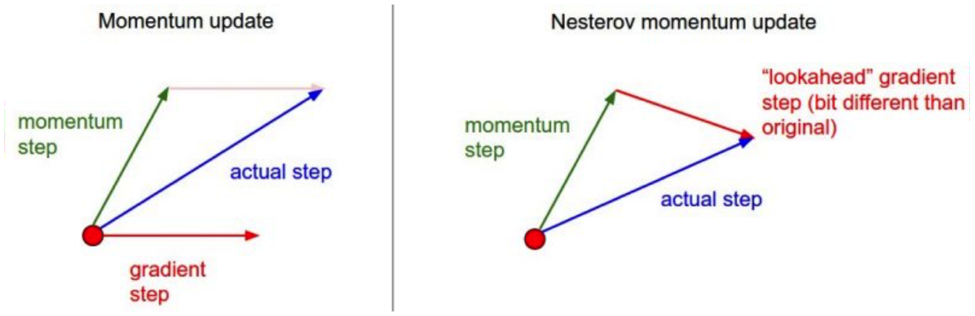
$$\theta_{t+1} = \theta_t - v_t$$

- Empirically found to give good performance

[2]Nesterov, A method of solving a convex programming problem with convergence rate O($1/k^2$), Soviet Mathematics Doklady, 1983, 27:2, pp 372–376

[3]Sutskever et al, On the importance of initialization and momentum in deep learning, ICML 2013, Vol 28, pp 1139—1147

Nesterov Accelerated Momentum: Visualization



Credit: Fei-Fei Li, CS231N course, Stanford Univ

Nesterov Accelerated Momentum: Algorithm

```
Require: Learning rate $\alpha$, momentum parameter $\gamma$, initial parameters $\theta_t$, training dataset $\mathcal{D}_{tr}$
 1: Initialize $v_{t-1} = 0$
 2: while stopping criterion not met do
 3:     Initialize gradients $\Delta\theta_t = 0$
 4:     Get look-ahead parameters $\tilde{\theta}_t = \theta_t - \gamma v_{t-1}$
 5:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ do
 6:         Compute gradient using look-ahead parameters $\nabla_{\tilde{\theta}_t}\mathcal{L}(\tilde{\theta}_t; x^{(i)}, y^{(i)})$
 7:         Aggregate gradient $\Delta\theta_t = \Delta\theta_t + \nabla_{\tilde{\theta}_t}\mathcal{L}$
 8:     end for
 9:     Update velocity $v_t = \gamma v_{t-1} + \alpha\Delta\theta_t$
10:     Apply update $\theta_{t+1} = \theta_t - v_t$
11: end while
```

Nesterov Accelerated Momentum: Illustration

Credit: Mitesh Khapra, IIT Madras

GD: Pros and Cons

What do you think?

GD: Pros and Cons

What do you think?

- For every parameter update, GD parses the entire dataset, hence called **Batch GD**
- Advantages of Batch GD
    - Conditions of convergence well-understood
    - Many acceleration techniques (e.g. conjugate gradient) operate in batch GD setting

GD: Pros and Cons

What do you think?

- For every parameter update, GD parses the entire dataset, hence called **Batch GD**
- Advantages of Batch GD
  - Conditions of convergence well-understood
  - Many acceleration techniques (e.g. conjugate gradient) operate in batch GD setting
- Disadvantages of Batch GD
  - Computationally slow
  - E.g. ImageNet (http://www.image-net.org), a commonly used dataset in vision, has ~ 14.2 million samples; an iteration over it is going to be very slow

Stochastic Gradient Descent

Stochastic GD (SGD): Randomly shuffle the training set, and update parameters after gradients are computed for each training example

Require: Learning rate α, initial parameters θ_t, training dataset $\mathcal{D}_{tr}$

```
1: while stopping criterion not met do
2:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ do
3:         Compute gradient using backpropagation $\nabla_{\theta_t}\mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
4:         Gradient $\Delta\theta_t = \nabla_{\theta_t}\mathcal{L}$
5:         Apply update $\theta_{t+1} = \theta_t - \alpha\Delta\theta_t$
6:     end for
7: end while
```

Mini-batch Stochastic Gradient Descent

Mini-Batch Stochastic GD: Update parameters after gradients are computed for a randomly drawn mini-batch of training examples *(default option today, often simply called as SGD)*

Require: Learning rate α, initial parameters θ_t, mini-batch size m, training dataset $\mathcal{D}_{tr}$

```
1: while stopping criterion not met do
2:     Initialize gradients $\Delta\theta_t = 0$
3:     Sample m examples from $\mathcal{D}_{tr}$ (call it $\mathcal{D}_{mini}$)
4:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{mini}$ do
5:         Compute gradient using backpropagation $\nabla_{\theta_t}\mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
6:         Aggregate gradient $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t}\mathcal{L}$
7:     end for
8:     Apply update $\theta_{t+1} = \theta_t - \alpha\Delta\theta_t$
9: end while
```

Illustration of SGD

Do the oscillations remind you of something?

Illustration of SGD

Do the oscillations remind you of something? Momentum?

Credit: Mitesh Khapra, IIT Madras

Illustration of Mini-batch SGD

Notice how the traversal is oscillating less for mini-batch GD $(m = 2)$ when compared to SGD

Credit: Mitesh Khapra, IIT Madras

SGD: Pros and Cons

SGD: Pros and Cons

Advantages of SGD

- Usually much faster than batch learning; because there is lot of redundancy in batch learning
- Often results in better solutions; SGD's noise can help in escaping local minima (provided neighborhood provides enough gradient information) and saddle points.[a]
- Can be used for tracking changes

[a]http://mitliagkas.github.io/ift6085-2019/ift-6085-bonus-lecture-saddle-points-notes.pdf

SGD: Pros and Cons

Advantages of SGD

- Usually much faster than batch learning; because there is lot of redundancy in batch learning
- Often results in better solutions; SGD's noise can help in escaping local minima (provided neighborhood provides enough gradient information) and saddle points.[a]
- Can be used for tracking changes

[a] http://mitliagkas.github.io/ift6085-2019/ift-6085-bonus-lecture-saddle-points-notes.pdf

Disadvantages of SGD

- Noise in SGD weight updates – can lead to no convergence!
- Can be controlled using learning rate, but identifying proper learning rate is a problem of its own

Choosing Learning Rate

- SGD requires a good learning rate to perform
- If learning rate is too small, it takes a long time to converge; and if it is too large, the gradients explode. How to choose?

Choosing Learning Rate

- SGD requires a good learning rate to perform
- If learning rate is too small, it takes a long time to converge; and if it is too large, the gradients explode. How to choose?
- Possible methods:
    - **Naive linear search** (learning rate remains constant throughout the process)

Choosing Learning Rate

- SGD requires a good learning rate to perform
- If learning rate is too small, it takes a long time to converge; and if it is too large, the gradients explode. How to choose?
- Possible methods:
  - **Naive linear search** (learning rate remains constant throughout the process)
  - Annealing-based methods:
    - **Step decay:** Reduce learning rate after every n iteration/epochs or if certain degenerative conditions are met (e.g. if current error more than previous error)

Choosing Learning Rate

- SGD requires a good learning rate to perform
- If learning rate is too small, it takes a long time to converge; and if it is too large, the gradients explode. How to choose?
- Possible methods:
    - **Naive linear search** (learning rate remains constant throughout the process)
    - Annealing-based methods:
        - **Step decay:** Reduce learning rate after every n iteration/epochs or if certain degenerative conditions are met (e.g. if current error more than previous error)
        - **Exponential decay:** $\alpha = \alpha_0^{-kt}$ where α and k are hyperparameters and t is iteration number

Choosing Learning Rate

- SGD requires a good learning rate to perform
- If learning rate is too small, it takes a long time to converge; and if it is too large, the gradients explode. How to choose?
- Possible methods:
  - **Naive linear search** (learning rate remains constant throughout the process)
  - Annealing-based methods:
    - **Step decay:** Reduce learning rate after every n iteration/epochs or if certain degenerative conditions are met (e.g. if current error more than previous error)
    - **Exponential decay:** $\alpha = \alpha_0^{-kt}$ where α and k are hyperparameters and t is iteration number
    - **1/t Decay:** $\alpha = \frac{\alpha_0}{1+kt}$ where α and k are hyperparameters and t is iteration number

Choosing Learning Rate

- SGD requires a good learning rate to perform
- If learning rate is too small, it takes a long time to converge; and if it is too large, the gradients explode. How to choose?
- Possible methods:
  - **Naive linear search** (learning rate remains constant throughout the process)
  - Annealing-based methods:
    - **Step decay:** Reduce learning rate after every n iteration/epochs or if certain degenerative conditions are met (e.g. if current error more than previous error)
    - **Exponential decay:** $\alpha = \alpha_0^{-kt}$ where α and k are hyperparameters and t is iteration number
    - **1/t Decay:** $\alpha = \frac{\alpha_0}{1+kt}$ where α and k are hyperparameters and t is iteration number
- The above are heuristic however, any automatic way to pick learning rate?

Credit: Mitesh Khapra, IIT Madras

Adaptive Gradients (Adagrad)

Intuition

- Sparse but important features may often have small gradients when compared to others; learning is slow in their direction
- We can impose different learning rates to each feature such that sparse features have a higher learning rate

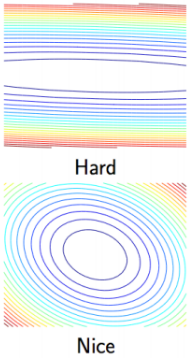
Weight update given by:

$$r_t = r_{t-1} + (\Delta\theta_t)^2$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\delta + \sqrt{r_t}}\Delta\theta_t$$

Adagrad

Why adapt to geometry?



y_t	$\phi_{t,1}$	$\phi_{t,2}$	$\phi_{t,3}$
1	1	0	0
-1	.5	0	1
1	-.5	1	0
-1	0	0	0
1	.5	0	0
-1	1	0	0
1	-1	1	0
-1	-.5	0	1

1. Frequent, irrelevant
2. Infrequent, predictive
3. Infrequent, predictive

Figure Credit: http://seed.ucsd.edu/mediawiki/images/6/6a/Adagrad.pdf

Adagrad Algorithm

Require: Learning rate α, initial parameters θ_t, small constant δ (usually 10^{-7} for numeric stability), training dataset $\mathcal{D}_{tr}$

```
 1: Initialize gradient accumulation $r_{t-1} = 0$
 2: while stopping criterion not met do
 3:     Initialize gradients $\Delta\theta_t$ = 0
 4:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ do
 5:         Compute gradient using backpropagation $\nabla_{\theta_t}\mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
 6:         Aggregate gradient $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t}\mathcal{L}$
 7:     end for
 8:     Update gradient accumulation $r_t = r_{t-1} + (\Delta\theta_t)^2$
 9:     Apply update $\theta_{t+1} = \theta_t - \frac{\alpha}{\delta+\sqrt{r_t}}\Delta\theta_t$
10: end while
```

RMSProp

Intuition

- Adagrad increases denominator term very quickly for frequent features as their gradients
- Ultimately, the effective learning rate becomes too low to take a step
- Why not apply a weighted decay to the denominator to curb its aggressive increase?

Weight update given by:

$$r_t = (\rho) r_{t-1} + (1-\rho)(\Delta\theta_t)^2$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\delta + \sqrt{r_t}} \Delta\theta_t$$

RMSProp Algorithm

```
Require: Learning rate $\alpha$, decay rate $\rho$, initial parameters $\theta_t$, small constant $\delta$(usually $10^{-6}$ for
    numeric stability), training dataset $\mathcal{D}_{tr}$
 1: Initialize gradient accumulation $r_{t-1} = 0$
 2: while stopping criterion not met do
 3:     Initialize gradients $\Delta\theta_t = 0$
 4:     for each $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ do
 5:         Compute gradient using backpropagation $\nabla_{\theta_t}\mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
 6:         Aggregate gradient $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t}\mathcal{L}$
 7:     end for
 8:     Update gradient accumulation $r_t = (\rho)r_{t-1} + (1-\rho)(\Delta\theta_t)^2$
 9:     Apply update $\theta_{t+1} = \theta_t - \frac{\alpha}{\delta+\sqrt{r_t}}\Delta\theta_t$
10: end while
```

Adaptive Moments (ADAM)

Intuition

Combine Momentum and RMSProp alorithms

Weight update given by:

$$s_t = (\rho_1)r_{t-1} + (1-\rho_1)(\Delta\theta_t)$$

$$r_t = (\rho_2)r_{t-1} + (1-\rho_2)(\Delta\theta_t)^2$$

$$\text{Bias Correction: } \tilde{s}_t = \frac{s_t}{1-\rho_1^t}, \tilde{r}_t = \frac{r_t}{1-\rho_2^t}$$

$$\theta_{t+1} = \theta_t - \alpha\frac{\tilde{s}_t}{\delta + \sqrt{\tilde{r}_t}}$$

Since we are using a running average over both moments, for the initial few steps, both moments are biased towards initial moments s_0 and r_0.

ADAM algorithm

Require: Learning rate α, decay rate for moment estimates ρ_1 **and** ρ_2, initial parameters θ_t, small constant δ(usually 10^{-8} for numeric stability), training dataset $\mathcal{D}_{tr}$

```
 1: Initialize first and second moment estimates $r_{t-1} = 0, s_{t-1} = 0$
 2: **while** stopping criterion not met **do**
 3:     Initialize gradients $\Delta\theta_t = 0$
 4:     **for each** $(x^{(i)}, y^{(i)})$ in $\mathcal{D}_{tr}$ **do**
 5:         Compute gradient using backpropagation $\nabla_{\theta_t}\mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
 6:         Aggregate gradient $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t}\mathcal{L}$
 7:     **end for**
 8:     Update first moment estimate $s_t = (\rho_1)r_{t-1} + (1-\rho_1)(\Delta\theta_t)$
 9:     Update second moment estimate $r_t = (\rho_2)r_{t-1} + (1-\rho_2)(\Delta\theta_t)^2$
10:     Correct for biases $\tilde{s_t} = \frac{s_t}{1-\rho_1^t}, \tilde{r_t} = \frac{r_t}{1-\rho_2^t}$
11:     Apply update $\theta_{t+1} = \theta_t - \alpha\frac{\tilde{s_t}}{\delta+\sqrt{\tilde{r_t}}}$
12: **end while**
```

Training NNs with SGD: Challenges

Issues that we might encounter when traversing error surfaces using GD

- Plateaus and flat regions
- Local minima and saddle points
- Vanishing and exploding gradients/cliffs
- Other challenges[a]
    - Ill-conditioning[b]
    - Inexact gradients
    - Poor correspondence between local and global structure
    - Choosing learning rate and other hyperparameters

[a] http://www.deeplearningbook.org/contents/optimization.html
[b] ftp://ftp.sas.com/pub/neural/illcond/illcond.html

Training NNs with SGD: Challenges

Given the issues:

- Cost surface is often non-quadratic, non-convex, high-dimensional
- Potentially, many minima and flat regions
- No strong guarantees that
  - Network will converge to a good solution
  - Convergence is swift
  - Convergence occurs at all

But it works!

Homework

Readings

- DL Book Chapter 8
- https://cs231n.github.io/neural-networks-3/sgd

Questions

- How to know if you are in a local minima (or any other critical point on the loss surface)?
- Why is training deep neural networks using GD and Mean-Squared Error (MSE) as cost function, a non-convex optimization problem?
- Let us assume a linear deep neural network (only linear activation functions, $f(x) = x$). Would using GD and MSE still be a non-convex problem?

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.