Parameters Optimization Explained

A brief yet descriptive guide to Gradient Descent, ADAM, ADAGRAD, RMSProp, NADAM

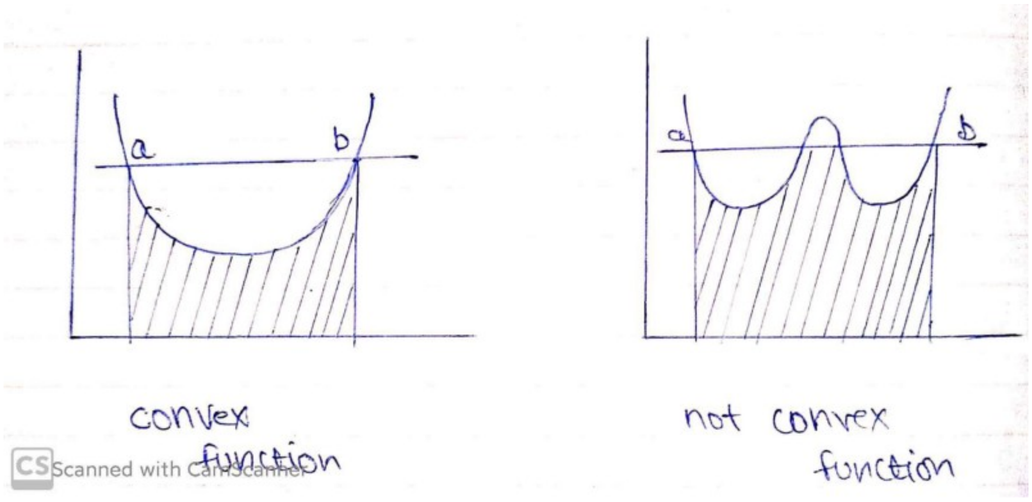
BY DAKSH TREHAN

Optimization is used to minimizing/maximizing an objective function f(x) parameterized by x. In machine/deep learning terminology, it's the task of minimizing the cost/loss function J(w) parameterized by the model's parameters w ∈ R^d.
Optimization algorithms (in the case of minimization) have one of the following goals:

1. Find the global minimum of the objective function. This is feasible if the objective function is convex, i.e. any local minimum is a global minimum.
2. Find the lowest possible value of the objective function within its neighborhood. That's usually the case if the objective function is not convex as the case in most deep learning problems.

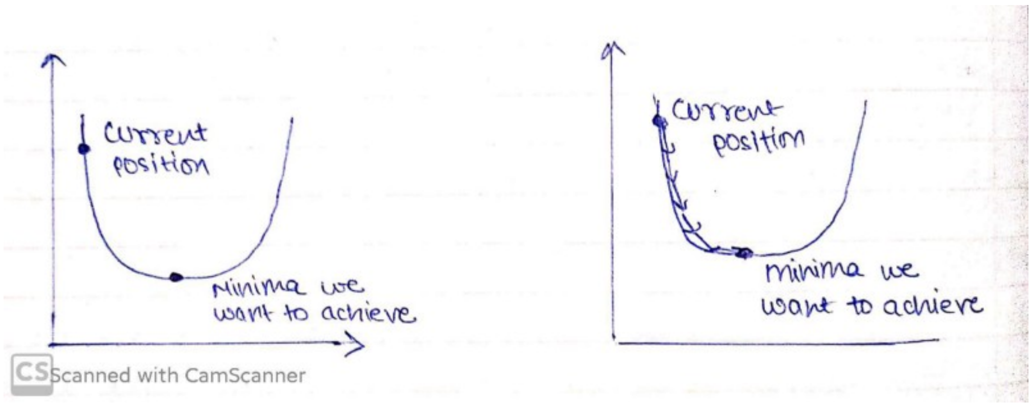
Gradient Descent

Gradient Descent is an optimizing algorithm used in Machine/ Deep Learning algorithms. The goal of Gradient Descent is to minimize the objective convex function f(x) using iteration.



Convex function v/s Not Convex function

Let’s see how Gradient Descent actually works by implementing it on a cost function.


current position
Minima we want to achieve
current position
minima we want to achieve
CS Scanned with CamScanner

Intuition behind Gradient Descent

For ease, let’s take a simple linear model.

Error = Y(Predicted)-Y(Actual)

A machine learning model always wants low error with maximum accuracy, in order to decrease error we will intuit our algorithm that you’re doing something wrong that is needed to be rectified, that would be done through Gradient Descent.

We need to minimize our error, in order to get a pointer to minima, we need to walk some steps that are known as alpha(learning rate).

Steps to implement Gradient Descent

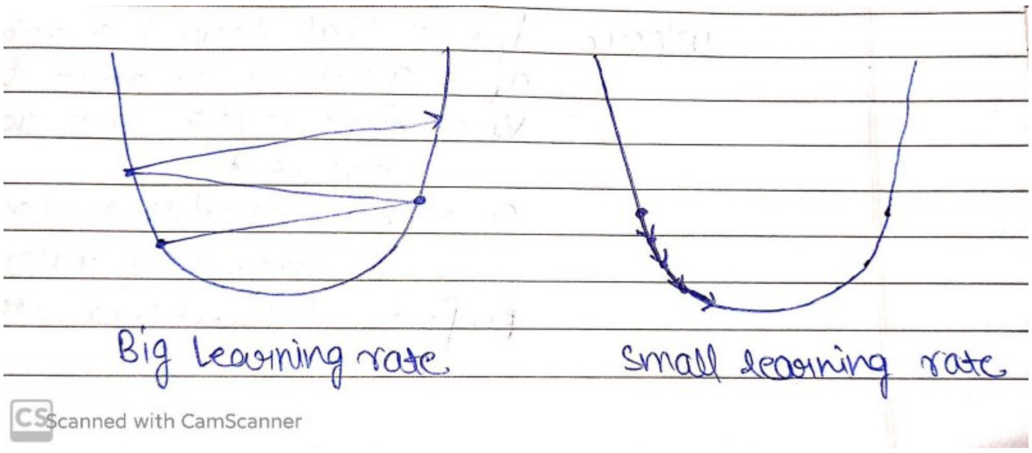
1. Randomly initialize values.
2. Update values.

$$weight^{(new)} = weight^{(old)} - constant\frac{\partial J(\Theta)}{\partial weight}$$

3. Repeat until slope =0

A derivative is a term that comes from calculus and is calculated as the slope of the graph at a particular point. The slope is described by drawing a tangent line to the graph at the point. So, if we are able to compute this tangent line, we might be able to compute the desired direction to reach the minima.

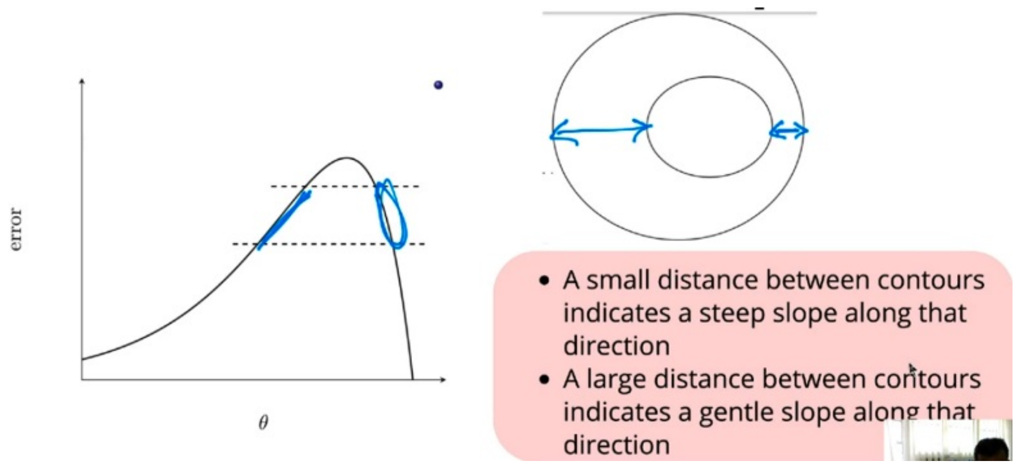
Learning rate must be chosen wisely as:
1. if it is too small, then the model will take some time to learn.
2. if it is too large, model will converge as our pointer will shoot and we'll not be able to get to minima.



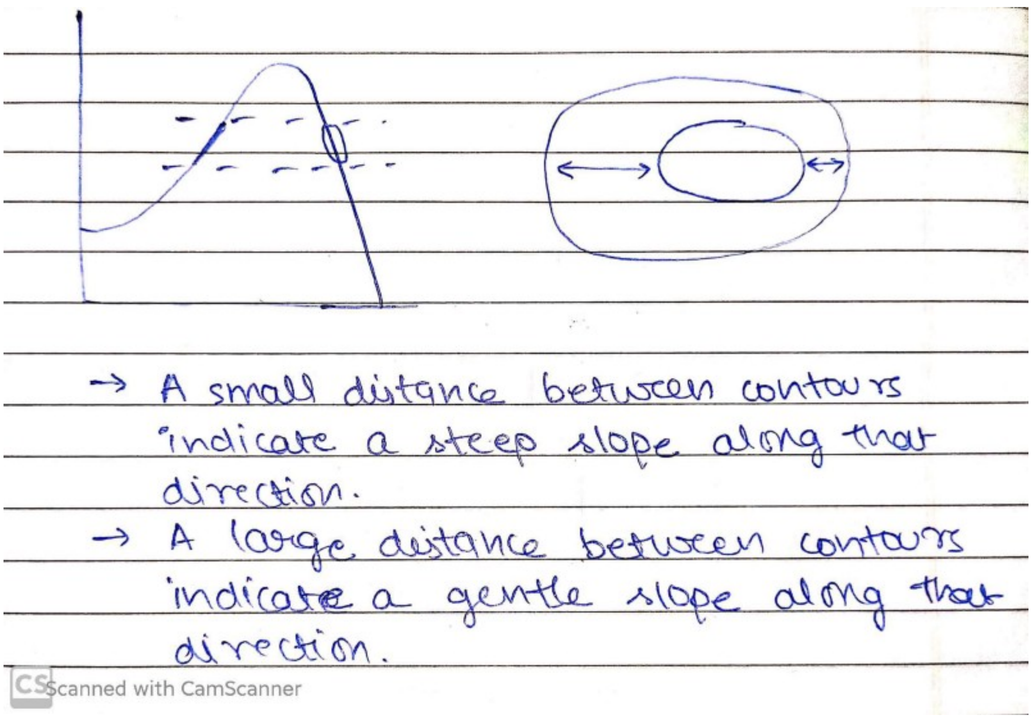
Comparison of various Learning Rate

Vanilla gradient descent, however, can't guarantee good convergence, due to following reasons:

- Picking an appropriate learning rate can be troublesome. A learning rate that is too low will lead to slow training and a higher learning rate will lead to overshooting of slope.
- Another key hurdle faced by Vanilla Gradient Descent is it avoid getting trapped in local minima; these local minimas are surrounded by hills of same error, which makes it really hard for vanilla Gradient Descent to escape it.



- A small distance between contours indicates a steep slope along that direction
- A large distance between contours indicates a gentle slope along that direction



→ A small distance between contours indicate a steep slope along that direction.

→ A large distance between contours indicate a gentle slope along that direction.

Contours representing gentle and steep slope

In simple words, every step we take towards minima tends to decrease our slope, now if we visualize, in steep region of curve derivative is going to be large therefore steps taken by our model too would be large but as we will enter gentle region of slope our derivative will decrease and so will the time to reach minima.

To know more about Gradient Descent methods and it's strategies follow: -

Gradient Descent Explained
*A comprehensive guide to Gradient Descent*towardsdatascience.com

ADAGRAD(Adaptive Gradient Algorithm)

The drawback of Gradient Descent was decay of learning rate due to which our pointer wasn't able to converge. Now to overcome this drawback we intuit to change learning rate for each and every layer, each and every iteration and each and every neuron as well.

Adagrad eliminates the need to manually change the hyperparameters.

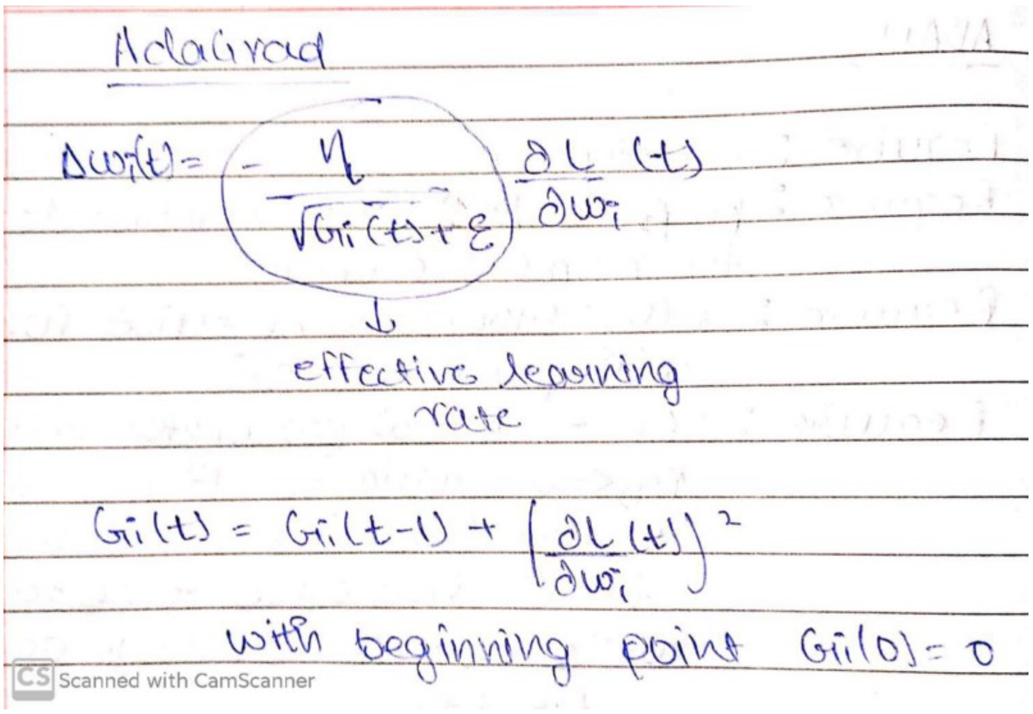
The reason to adaptively change learning rate is because there are two types of data :-

1. Sparse data
2. Dense data

In case of sparse data, we would experience sparse ON(1) features and more frequent OFF(0) features, now, most of the time gradient update will be NULL as derivative is zero in most cases and when it will be one, the steps would be too small to reach minima.

For frequent features we require low learning rate, but for high features we require high learning rate.

So, in order to boost our model for sparse nature data, we need to chose adaptive learning rate.



Adagrad

$$\Delta w_i(t) = \underbrace{-\frac{\eta}{\sqrt{G_i(t)+\varepsilon}}}_{\text{effective learning rate}} \frac{\partial L}{\partial w_i}(t)$$

$$G_i(t) = G_i(t-1) + \left(\frac{\partial L}{\partial w_i}(t)\right)^2$$

with beginning point $G_i(0) = 0$

ADAGRAD Algorithm

G(t) is the current step and G(t-1) is the previous step, Epsilon is included as if G(t) becomes zero our learning rate will crash so in order to maintain consistency we use epsilon which denotes a random small positive number.

In the denominator, we calculate the sum of the square of the previous gradients. Each term is a positive term so it continues expanding to make the learning rate η insignificant to the point that algorithm is no longer able to learn.

> The tag line for ADAGRAD is ***Fewer Updates, Lesser decay.***

If you're familiar with working of Gradient Descent, you must've noticed with every step taken towards minima our learning rate shortens which leads to its decay and more time consumption, that's fixed using ADAGRAD.

Advantages:-

Parameters corresponding to sparse features gets better updates.

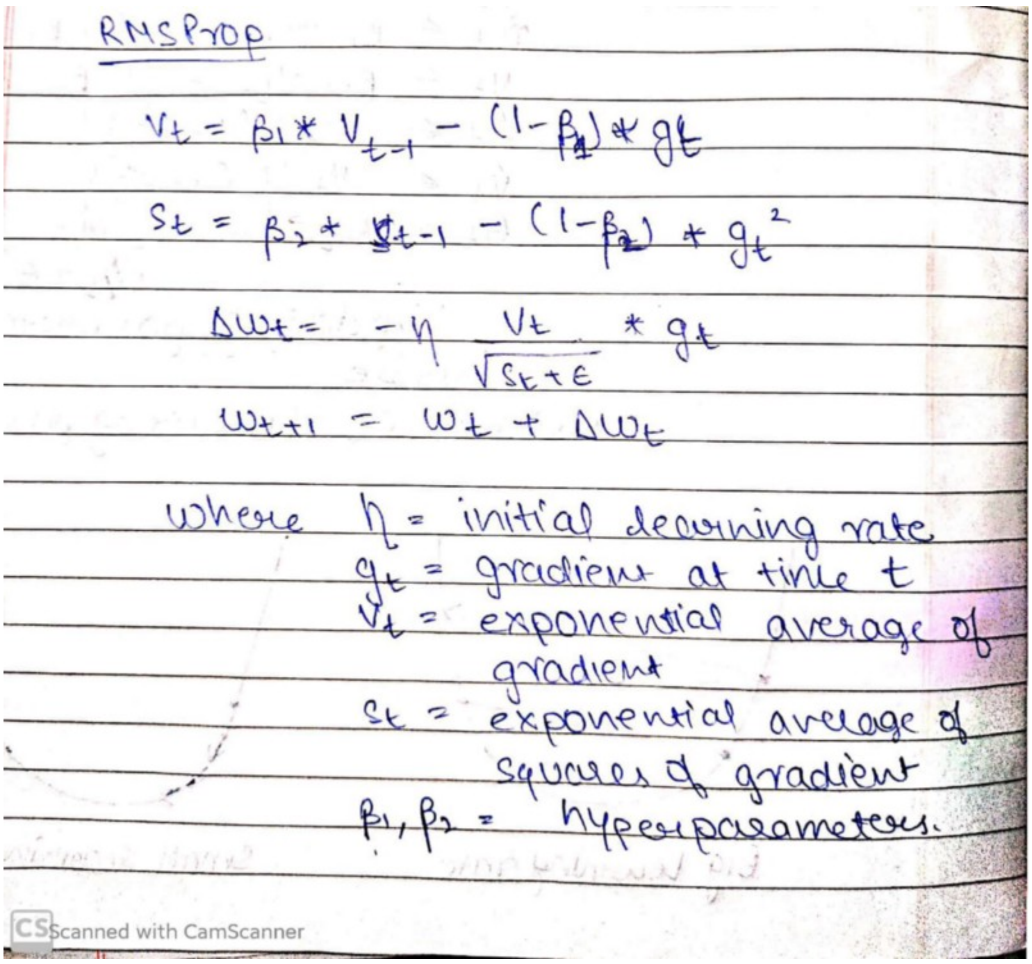
Disadvantages:-

Learning rate decays very aggressively as denominator grows.

RMSProp

The main aim for RMSProp is to prevent overly aggressive behavior of G(t)in ADAGRAD. Now if we go deep in ADAGRAD we can comprehend that G(t) is exploding as we are collectively aggregating square of past gradient from first iteration to last iteration.

To escape blasting of G(t) we intend to average the decay in order to make process more smooth and less time consuming.



RMSProp

$$V_t = \beta_1 * V_{t-1} - (1-\beta_1) * g_t$$

$$S_t = \beta_2 * S_{t-1} - (1-\beta_2) * g_t^2$$

$$\Delta w_t = -\eta \frac{V_t}{\sqrt{S_t+\epsilon}} * g_t$$

$$w_{t+1} = w_t + \Delta w_t$$

where η = initial learning rate
g_t = gradient at time t
V_t = exponential average of gradient
S_t = exponential average of squares of gradient
β_1, β_2 = hyperparameters.

RMSProp algorithm

Even if our G(t) increases, it will be diminished to small quantity as we are computing its product by (1-Б2) which is itself very small thus cancelling its high effect.

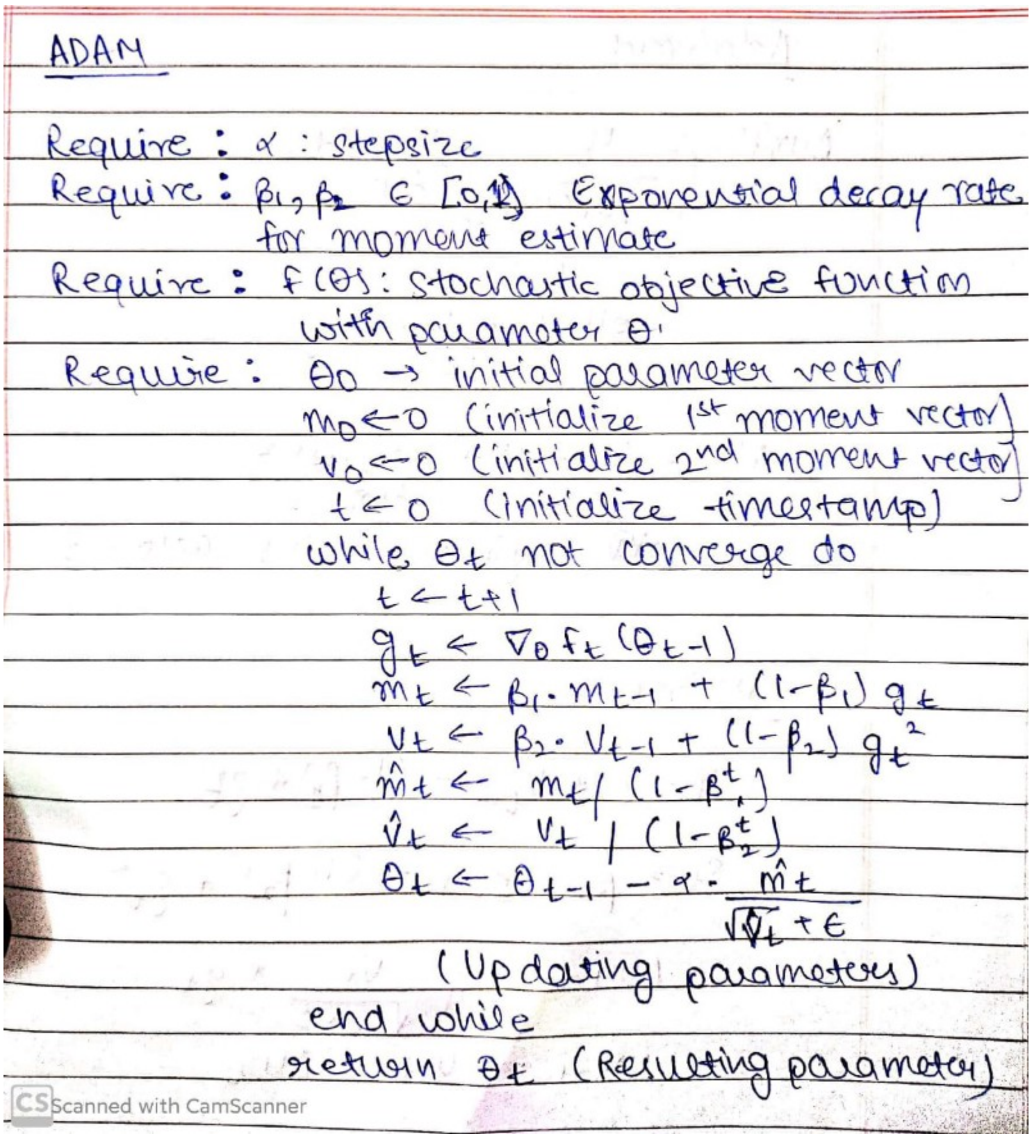
Here, the learning rate is decreasing but not as fast as in case of ADAGRAD thus making convergence more fluid with less time consumption.

In nutshell, Rmsprop is a very clever way to deal with the problem. It uses a moving average of squared gradients to normalize the gradient itself. That is able to avoid large gradient and can boost the step for small gradient thus making it a perfect match.

ADAM

The algorithms uses the power of adaptive learning rates methods to find individual learning rates for each parameter. It also constitutes positives of Adagrad, which works really well with sparse gradients, but is slowed down for non-convex optimization of neural networks.

Adam can be regarded as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to mapthe learning rate like RMSprop and it takes pros of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.



ADAM

```
Require : $\alpha$ : stepsize
Require : $\beta_1, \beta_2 \in [0,1)$ Exponential decay rate
          for moment estimate
Require : $f(\theta)$ : Stochastic objective function
          with parameter $\theta$.
Require : $\theta_0 \rightarrow$ initial parameter vector
          $m_0 \leftarrow 0$ (initialize 1st moment vector)
          $v_0 \leftarrow 0$ (initialize 2nd moment vector)
          $t \leftarrow 0$ (initialize timestamp)
          while $\theta_t$ not converge do
              $t \leftarrow t+1$
              $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
              $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) g_t$
              $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) g_t^2$
              $\hat{m}_t \leftarrow m_t / (1-\beta_1^t)$
              $\hat{v}_t \leftarrow v_t / (1-\beta_2^t)$
              $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
                        (Updating parameters)
          end while
          return $\theta_t$ (Resulting parameter)
```

ADAM Algorithm

Adam algorithm first updates the exponential moving averages of the gradient(mt) and the squared gradient(vt) which is the estimates of the first and second moment.

Hyper-parameters β1, β2 ∈ [0, 1) control the exponential decay rates of these moving averages as shown below

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

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

m_t and v_t are estimates of first and second moment respectively

Moving midpoints are introduced as 0 leading to estimation of moment that are one-sided around 0 particularly during the starting timesteps. This initialization is later altered resulted in correct bias approximations.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{V_t}{1-\beta_2^t}$$

$\hat{m}_t$ and $\hat{v}_t$ are bias corrected estimates of first and second moment respectively

Finally, we update the parameter as shown below

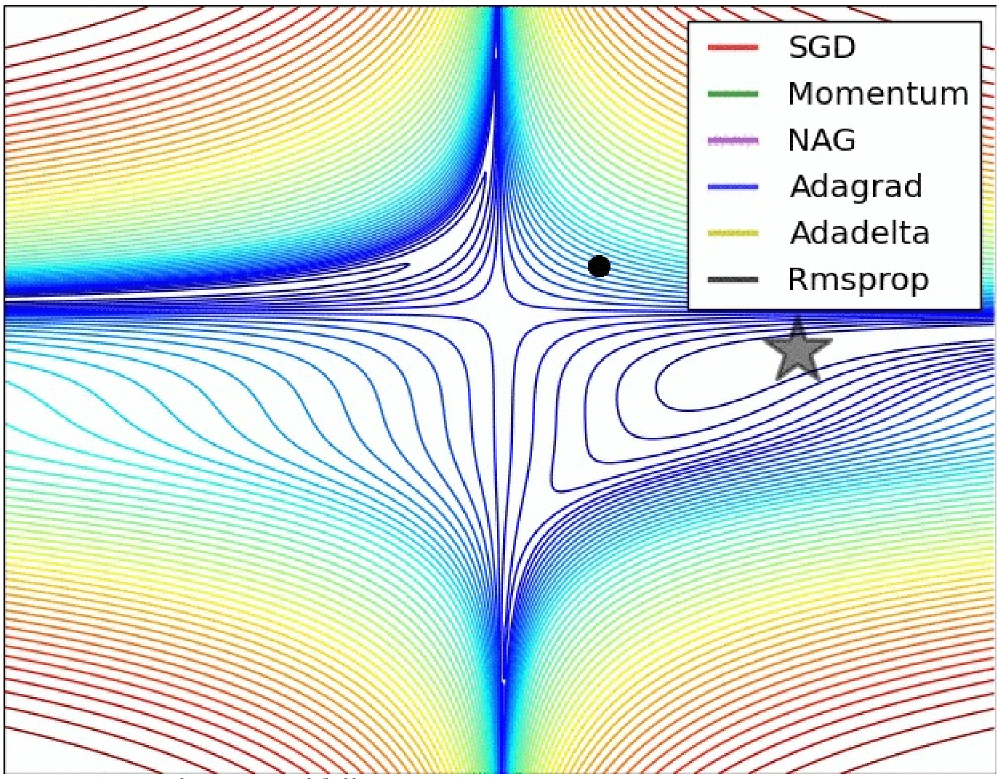
$$\theta_{t+1} = \theta_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

Adam implements the exponential moving average of the gradients to scale the learning rate instead of a simple average as in Adagrad. It keeps an exponentially decaying average of past gradients.

Adam is computationally efficient and has very little memory requirement.

NADAM(Nesterov-accelerated Adaptive Moment Estimation)

- NADAM combines NAG and Adam
- NADAM is employed for noisy gradients or for gradients with high curvatures
- The learning process is accelerated by summing up the exponential decay of the moving averages for the previous and current gradient



Comparing performance of different optimizers, Source

Conclusion

Hopefully, this article has increased your understanding of optimization techniques.

As always, thank you so much for reading, and please share this article if you found it useful! :)

References:

[1] An overview of gradient descent optimization algorithms By Sebastian Ruder

[2] An Empirical Comparison of Optimizers for Machine Learning Models By Rick Wierenga

[3] Overview of different Optimizers for neural networks By Renu Khandelwal

[4] Adam — latest trends in deep learning optimization By Vitaly Bushaev

[5] Learning Parameters, Part 5: AdaGrad, RMSProp, and Adam By Akshay L Chandra

The cover template is designed by me on canva.com, the source is mentioned on every visual, and the un-mentioned visuals are from my notebook or are designed by me on Paint 3D.

Feel free to connect:

Join me at www.dakshtrehan.com

LinkedIN ~ https://www.linkedin.com/in/dakshtrehan/

Instagram ~ https://www.instagram.com/_daksh_trehan_/

Github ~ https://github.com/dakshtrehan

Check my other articles:-

Detecting COVID-19 using Deep Learning.

Gradient Descent Explained

Logistic Regression Explained

Linear Regression Explained

> *Determining perfect fit for your ML Model.*
>
> *Serving Data Science to a Rookie.*
>
> *Relating Machine Learning techniques to Real Life.*

Follow for further Machine Learning/ Deep Learning blogs.

> *Cheers.*

By Daksh Trehan on May 24, 2020.

Canonical link

Exported from Medium on July 15, 2020.