**Gradient Descent**

It is a slippery slope, but promise it gets better at the bottom4

[Hamza Mahmood](https://hamzamahmood.medium.com/?source=post_page-----3a7db7520711--------------------------------)

[Jan 3, 2019·5 min read](https://towardsdatascience.com/gradient-descent-3a7db7520711?source=post_page-----3a7db7520711--------------------------------)

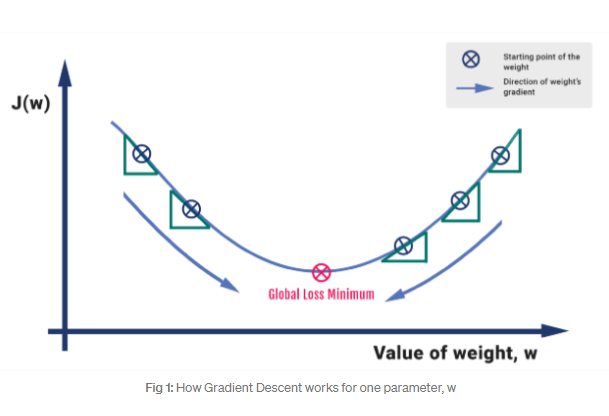
***tl;dr****Gradient Descent is an optimization technique that is used to improve deep learning and neural network-based models by minimizing the cost function.*

In our previous post, we talked about activation functions (link [here](https://towardsdatascience.com/activation-functions-in-neural-networks-83ff7f46a6bd)) and where it is used in machine learning models. However, we also heavily used the term ‘Gradient Descent’ which is a key element in deep learning models, which are going to talk about in this post.

Definition and Nomenclature

Gradient Descent is a process that occurs in the **backpropagation** phase where the goal is to continuously resample the gradient of the model’s parameter in the opposite direction based on the weight *w*, updating consistently until we reach the **global minimum**of function *J(w)*.

To put it simply, we use gradient descent to minimize the cost function, *J(w)*.



An analogy could be drawn in the form of a steep mountain whose base touches the sea. We assume a person’s goal is to reach down to sea level. Ideally, the person would have to take one step at a time to reach the goal. Each step has a gradient in the negative direction (Note: the value can be of different magnitude). The person continues hiking down till he reaches the bottom or to a threshold point, where there is no room to go further down.

## Mathematics

Let’s formalize the analogy into an algorithmic form. We compute the activation for the incoming parameters, we carry out feedforward by taking the weighted sum of the activation and its bias. We extract the error term of the output sample by subtracting it with the actual ‘target’ value.

The gradient descent process is exhibited in the form of the backpropagation step where we compute the error vectors δ backward, starting from the final layer. Depending upon the activation function, we identify how much change is required by much change is required by taking the partial derivative of the function with respect to **w.**The change value gets multiplied by the learning rate. As part of the output, we subtract this value from the previous output to get the updated value. We continue this till we reach convergence.

In the code below, I wanted to highlight how one can write a simple code to get a visualization of how gradient descent works. Running this piece of code; using the Tanh activation function, we will observe the current value of **10** go down to a value of **8.407e-06** on the 10000th iteration, which is our global minima.



There are a number of gradient descent algorithms out there. I’ll mention a few below:

* Batch Gradient Descent
* Stochastic Gradient Descent
* Mini-batch Gradient Descent

If you want to go into the technicalities of the more recent ones, I highly recommend going through Sebastian Ruder’s [article](http://ruder.io/optimizing-gradient-descent/) on the topic.

## Exploding & Vanishing Gradients

In deep networks or recurrent neural networks, there are two known issues explained in a paper by Pascanu et al (1994) — of exploding and vanishing gradients. This happens when are doing back propagation as we iterate through the code, there is a chance that the normal of the weight matrices going beyond 1. If this happens, the gradient explodes but if the normal is below 1, the gradient vanishes.

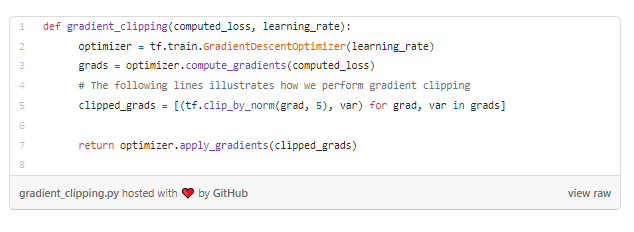
If we want to visualize exploding gradients, you will encounter at least one of the problems:

* The model will output ‘Nan’ values
* The model will display very large changes upon each step
* The error gradient values are consistently above 1.0 for each node in the training layer.

## Solution: Gradient Clipping

The solution to the exploding and vanishing gradient problem, we introduce gradient clipping, where we ‘clip’ the gradients if it goes over a certain threshold represented by a maximum absolute value. Hence, we keep the neural network stable as the weight values never reach the point that it returns a ‘Nan’. In a coded implementation removing the clipped gradients leads to ‘Nan’ values or infinite in the losses and fails to run further.

The code below showcases how to perform gradient clipping. Given that we have a vector of losses and a learning rate, we are able to compute a vector of gradients which are next clipped based on a maximum L2-norm value, which in this case I have written as 5.



So at the end of the day, when a question is posed to data scientist of what optimizer to use in order to minimize loss, there are a couple of factors to consider:

* The size of the training dataset
* How quickly do we need the training data to achieve convergence

Link of the paper referenced: <http://proceedings.mlr.press/v28/pascanu13.pdf>

## Conclusion

In this write-up, we covered a number of things: We covered what is Gradient Descent is and how it works in a neural network. We went through the mathematics involved and implemented a coded version of it. Lastly, we covered the issues involving gradient descent in the form of the vanishing and exploding gradient problem and discussed the solution for it using gradient clipping. In the next lecture, we will explore what activation functions are and how they are crucial in deep learning models, so stay tuned for that!