Vanishing/Exploding Gradients in Deep Neural Networks

Words By Nisha Arya Ahmed November 10, 2022

Building a Neural Network model can be very complicated and tuning the Neural Network model can make it even more confusing. One of the most common problems when working with Deep Neural Networks is the Vanishing and/or Exploding Gradient Descent. In order to prevent this from happening, one solution is initializing weights.

Initializing weights in Neural Networks helps to prevent layer activation outputs from Vanishing or Exploding during forward feedback. If either Vanishing or Exploding occurs, the loss gradient will either be too small or too large, meaning the network requires more time to converge.

Background on Deep Neural Networks:

A **Neural Network** is a network of biological neurons. In the use of Artificial Intelligence, Deep Neural Network contains artificial neurons or nodes.

Deep Neural Networks are made up of nodes that contain three different layers: an input layer, one or more hidden layers, and an output layer. Each node is connected to another node and is where computation happens.

1. **Input Layer** — receives the input data
2. **Hidden Layer(s)** — perform mathematical computations on the input data
3. **Output Layer** — returns the output data.

The nodes in the Neural Networks are made up of parameters which are called weights and used to calculate a weighted sum of the inputs.

Weight controls the strength of the connection between two neurons. The weight is a big factor in deciding how much influence the input has on the output.

Bias is to guarantee that there will always be activation in the neurons, even if the input is 0. Bias will always have a value of 1 and is an additional input into the next layer.

Feedforward and Backpropagation

A **Cost Function** is a mathematical formula used to calculate the error, it is the difference between our predicted value and the actual value. In the ideal world, we would want a Cost Function of 0, telling us that our outputs are the same as the data set outputs.

Neural Network models use a Cost Function optimization algorithm called Stochastic Gradient Descent. Its aim is to minimize the Cost Function by incrementally changing the weights of the network. Aiming to produce a set of weights that is capable of making useful predictions. In order to begin the optimization process, the algorithm requires a starting point in the space of possible weight values.

Feedforward Network is the process in which the result in the next neuron now becomes the input for the next neuron as the information always moves in one direction (forward).

In Neural Network, there is also a process called Backpropagation, also abbreviated as “backprop.” **Backpropagation** is the messenger who tells the Neural Network whether it made a mistake when it made a prediction.

Backpropagation goes through these steps:

1. The Neural Network makes a guess about data
2. The Neural Network is measured with a loss function

3. The error is backpropagated to be adjusted and corrected

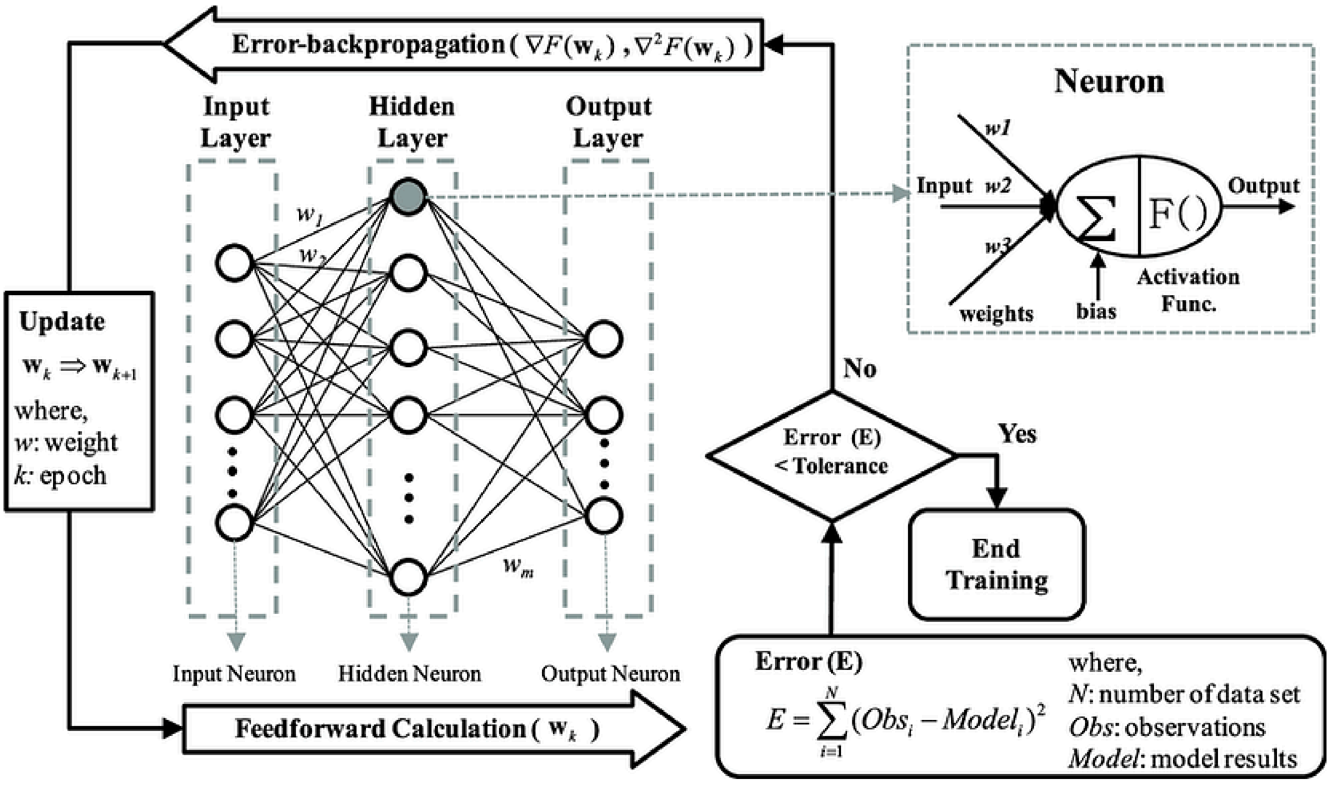
The Process:

As the input features are propagated, going through the various hidden layers consisting of different or same activation functions, we produce a sample of predictive probabilities.

The backpropagation algorithm moves towards the input layer and away from the output layer calculating error gradients.

> ***Most projects fail before they get to production. Check out our free ebook to learn how to implement an MLOps lifecycle to better monitor, train, and deploy your machine learning models to increase output and iteration.***

The gradient of the Cost Function in the Neural Network processes in relation to each parameter; weights and biases. The algorithm then takes a Gradient Descent steps towards the minimum cost and updates the value of each parameter in the Neural Network using these updated gradients.



Source: researchgate

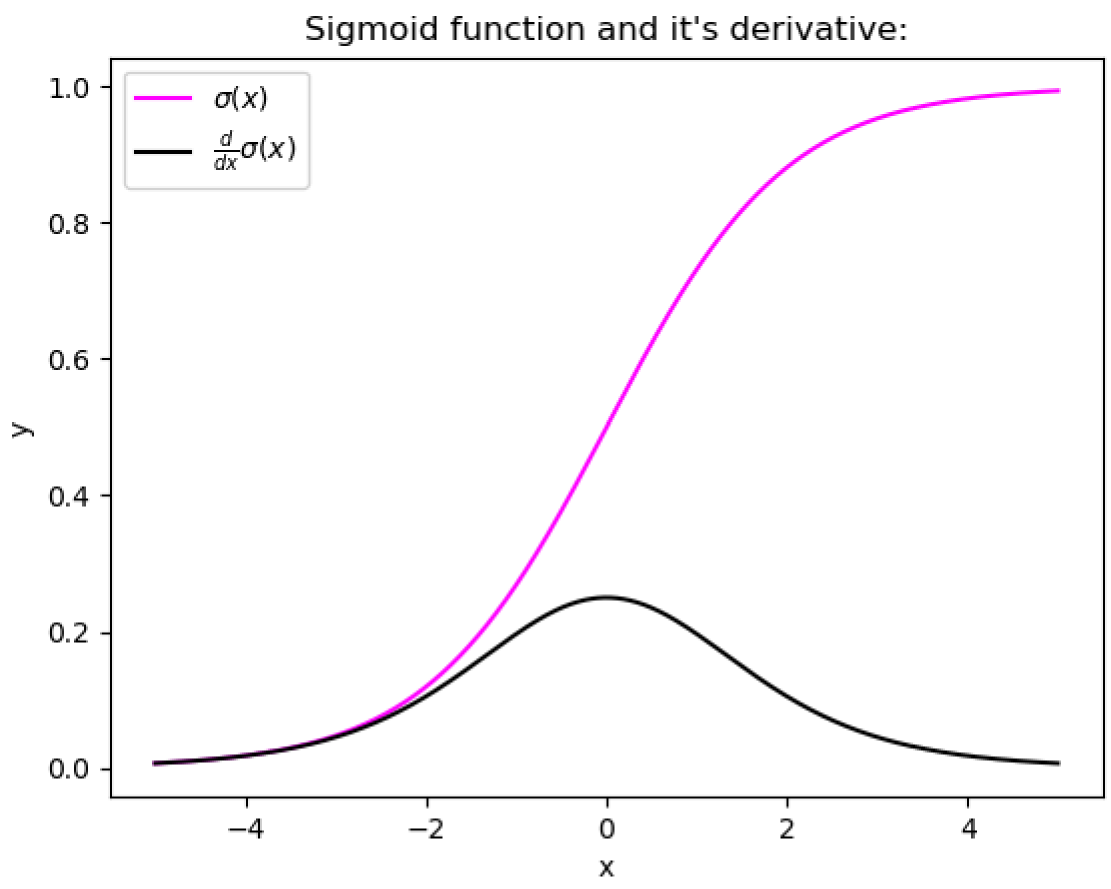
Why do the Gradients Vanish or Explode?

Vanish

Vanishing is when as backpropagation occurs, the gradients normally get smaller and smaller, gradually approaching zero. This leaves the weights of the initial or lower layers unchanged, causing the Gradient Descent to never converge to the optimum.

For example, Activation Functions such as the sigmoid function have a very prominent difference between the variance of their inputs and outputs. They shrink and transform a large input space into a smaller output space, which lies between [0,1].

Looking at the graph below of the Sigmoid Function, we can see that using larger inputs, regardless if they are negative or positive will classify at either 0 or 1. However, when the Backpropagation processes, it has no gradient to propagate backward in the Neural Network. The little gradient that does exist, will continuously keep diluting as the algorithm continues to process through the top layers, leaving nothing for the lower layers.



Source: Machine Learning Journey

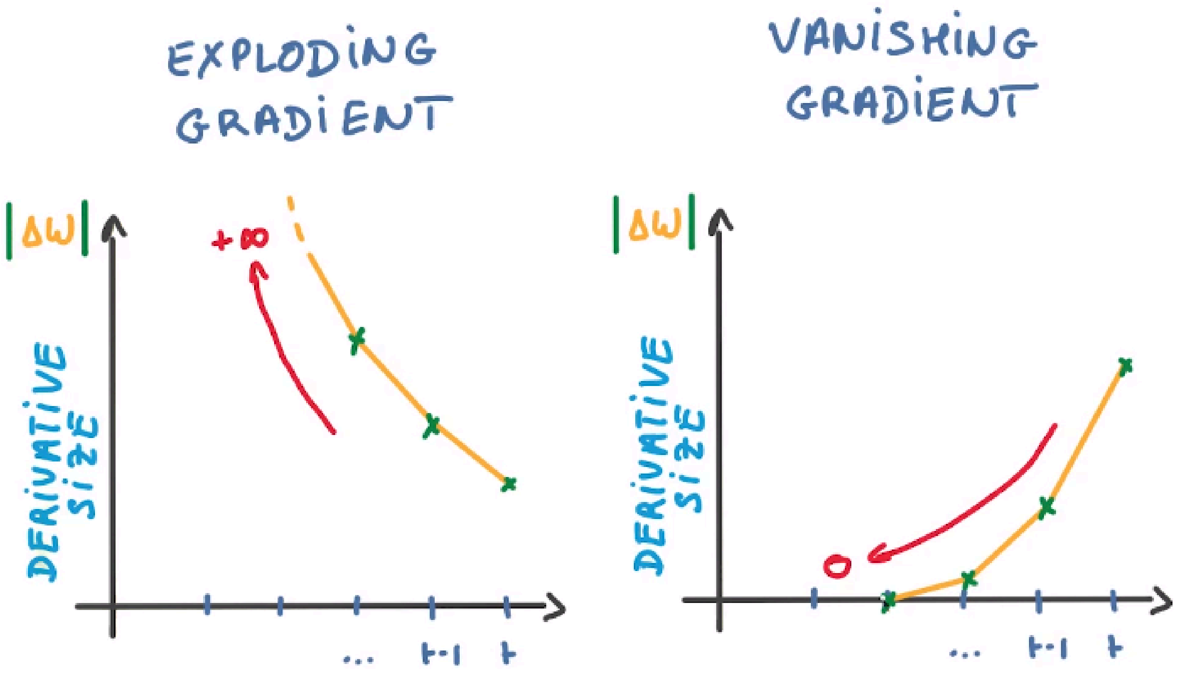
Explode

Exploding is the opposite of Vanishing and is when the gradient continues to get larger which causes a large weight update and results in the Gradient Descent to diverge.

Exploding gradients occur due to the weights in the Neural Network, not the activation function.

The gradient linked to each weight in the Neural Network is equal to a product of numbers. If this contains a product of values that is greater than one, there is a possibility that the gradients become too large.

The weights in the lower layers of the Neural Network are more likely to be affected by Exploding Gradient as their associated gradients are products of more values. This leads to the gradients of the lower layers being more unstable, causing the algorithm to diverge.



Source: Udacity

Weight Initialization

Weight Initialization is the process of setting the weights of a Neural Network to small random values that help define the starting point for the optimization of the model.

Weight Initialization Techniques:

Initializing weights to zero

If we initialize all our weights to zero, our Neural Network will act as a linear model because all the layers are learning the same thing.

Therefore, the important thing to note with initializing your weights for Neural Networks is to not initialize all the weight to zero.

Initializing weights randomly

Using random initialization defeats the problem caused by initializing weights to zero, as it prevents the neurons from learning the exact same features of their inputs. Our aim is for each neuron to learn the different functionalities of its input.

However, using this technique can also lead to vanishing or exploding gradients, due to incorrect activation functions not being used. It currently works effectively with the RELU activation function.

Initializing weights using Heuristic

This is considered the best technique to initialize weights for Neural Networks.

Heuristics serve as good starting points for weight initialization as they reduce the chances of vanishing or exploding gradients from occurring. This is due to the fact that the weights are neither too bigger than 1, nor less than 1. They also help in the avoidance of slow convergence.

The most common heuristics used are:

1. He-et-al Initialization.

When using the RELU activation function, this heuristic is used by multiplying the randomly generated values of W by:

$$\sqrt{\frac{2}{size^{[l-1]}}}$$

$$W^{[l]} = np.random.randn(size\_l, size\_l\text{-}1) * np.sqrt(2/size\_l\text{-}1)$$

Source

2. Xavier initialization

When using the Tanh activation function, this heuristic is used by multiplying the randomly generated values of W by:

$$\sqrt{\frac{1}{size^{[l-1]}}}$$

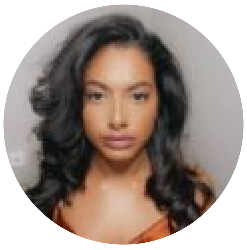
$$W^{[l]} = np.random.randn(size\_l, size\_l\text{-}1) * np.sqrt(1/size\_l\text{-}1)$$

Source

Conclusion

The rise in Machine Learning and the implementation and application of models in our day-to-day lives raises concern about how to efficiently these models work.

A lot of time and money is spent on models that don't accurately produce the outputs we expected. Therefore, it's important to understand Cost Function and how Stochastic Gradient Descent minimizes the Cost Function by changing the weights of the network. This will benefit the overall process of building your model, producing accurate outputs to make the right conclusions and decisions, at a reduced cost.



Nisha Arya Ahmed

Related Articles

Image Augmentation : A Fun and Easy Way to Improve Computer Vision Models

Daniel Onugha
March 4, 2024

Addressing the Challenges in Multilingual Prompt Engineering

Tioluwani Oyedele
February 26, 2024

The Deep of Deep Learnin g

Irem Komurcu
February 20, 2024

Photo by Almos Bechtold on Unsplash Deep learning is a machine learning

Image by istockphoto Computer vision has become a ground-breaking area in artificial intelligence and…

In an increasingly interconnected and diverse world where communication transcends language barriers, the ability to…

sub-branch that can automatically…

Subscribe to Comet

Sign up to receive the Comet newsletter and never miss out on the latest ML updates, news and events!

First Name: *

Last Name: *

Email Address: *

Get Updates

Products

Experiment Management

Artifacts

Model Registry

Model Production Monitoring

LLMOps

Learn

Documentation

Resources

Comet Blog

Deep Learning Weekly

Heartbeat

LLM Course

Company

About Us

News and Events

Careers

Contact Us

Pricing

Pricing

Create a Free Account

Contact Sales

Follow Us