Deep learning with TensorFlow 2.0 Keras basics

Hi, this is Sandeep, working as Software Engineer involved in Python, NLP, Machine Learning, Deep Leaning with TensorFlow 2.0 etc.

In this article I am exploring into I am exploring into Artificial Neural Networks topics.

These below topics we will cover in this document.

1. Cost Functions
2. Gradient Descent
3. Cross Entropy
4. Back Propagation
5. TensorFlow 2.0 vs Keras
6. Sequential Model

Cost Functions:

In Neural networks we create a network with Input Layers > Hidden Layers > Output Layers and we will add weightage and bias values to input variable in each layer.

z= W*X + b

a = Activation Function(f(z))

- **"a" is our final predicted value from overall networks.**

After network creates its prediction **how do we evaluate it?**

We need to take estimated outputs of network and then compare them to the real values of the Label.

In simple what cost function does?

Cost function are used to define how far we are in Neural Networks. We can call these are **Loss functions** or **Error Functions**.

Cost function must be an average so it can output a single value

From now I am representing the variable are

- ⇨ y represents True Value
- ⇨ a represents Predict Value
- ⇨ W*X + b = Z
  Pass into activation function **α(z) = a**

Quadratic cost function:

It's a common cost function. We simply calculate the difference between the real values **y(x)** and our predicted values **a(x)**

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

L = Last layer

a= output

We can think cost function $C(W, B, S^r, E^r)$

W = weight , Sr = input , B = Bias , Er = Desired Output

In real case, this means we have some cost functions C dependent lost of weights.

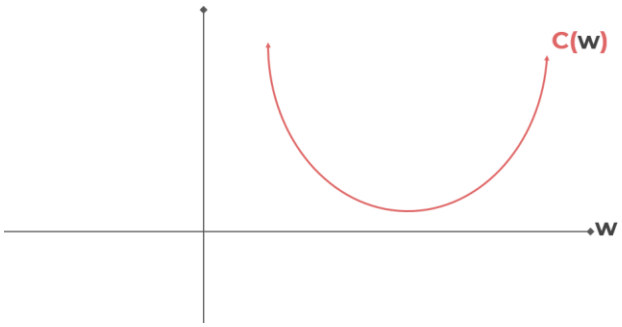
- C(w1,w2,w3,w4,.....wn)

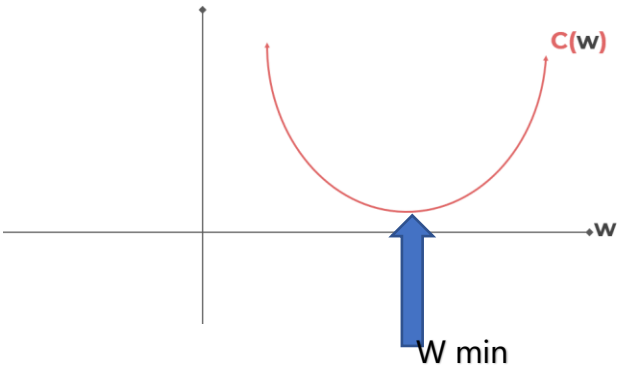
Now how do we figure out which weights lead us to the lowest cost?

Lets imagine we only had one weight in our cost function w.

We want to minimize our loss/cost functions.

⇨ Which means we need to figure out what values of w results in the minimum of C(w)





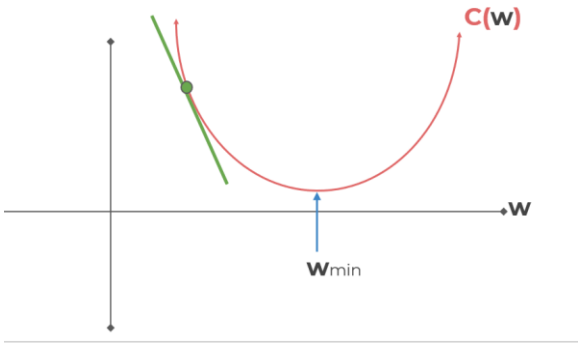
To solve this and find Wmin value we need to go for Gradient Descent.

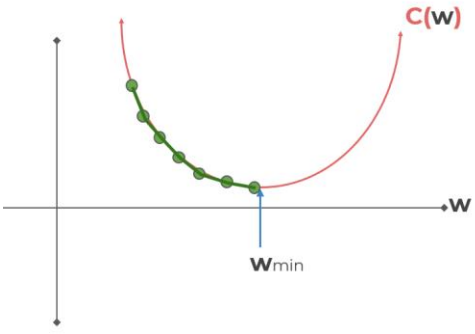
Gradient Descent:

From cost functions we need to find which weight gives minimum loss. For this we need to use

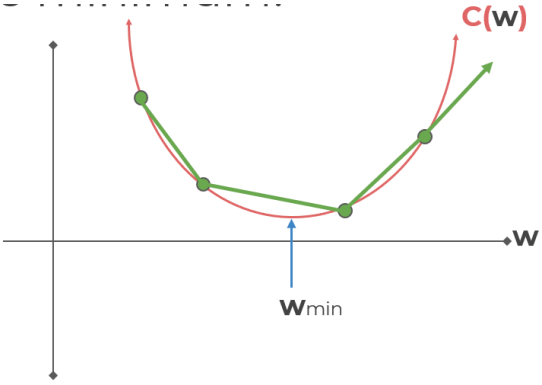
"**Gradient Descent**" Techniques.

We can calculate slope at a point.

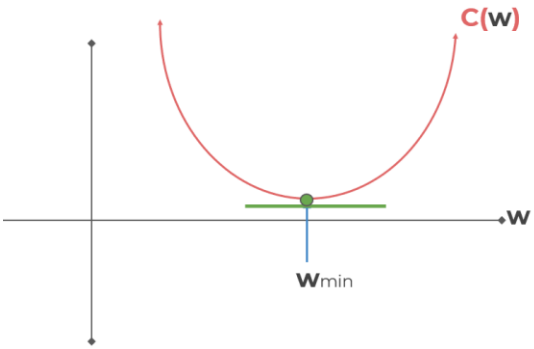




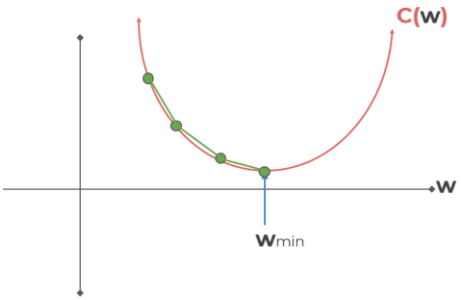
If we take smaller steps sizes take a long time to find the minimum W value.



If we take a long step sizes are faster, but we risk of overshooting minimum value.



Untill converge to zero indicating a minimum.



Find the correct step size know as **Learning Rate.**

Each step size is equal.

For this reason, we could start with larger steps then go smaller as we realize the slope gets closer to zero.

This is known as "Adaptive Gradient Descent"

Adam: A method for "Stochastic Optimization"

Adam is a much more efficient way of searching for these minimums. We often use this optimizer in Deep learning models.

Cross Entropy:

For classification problems, we often use the **cross entropy** loss function instead of Quadric Function.

Cross entropy works with Probability distribution.

For binary classification.

$$-(y\log(p)+(1-y)\log(1-p))$$

For Multi class classification M > 2

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

Back Propagation:

Fundamentally, we want to know how the cost function results changes with respect to the weights in the network, so we can update the weights to minimize the cost function

http://neuralnetworksanddeeplearning.com/chap2.html

TensorFlow vs Keras

TensorFlow is an open-source deep learning library developed by Google, and it has a large ecosystem of related components, including libraries like Tensor board, Deployment and Production APIs, and support for various programming languages.

The Keras API is easy to use and builds models by simply adding layers on top of each other through simple calls.

Sequential Models from Keras.

A Sequential model is appropriate for **a plain stack of layers** where each layer has **exactly one input tensor and one output tensor**.

```
from keras.models import Sequential
```

Early Stopping:

Keras can automatically stop training based on a loss condition on the validation data passed during the **model.fit ()** call.

Dropout Layers:

Dropout can be added to layers to "turn off" neurons during training to prevent overfitting.

Each Dropout layer will "drop" a user-defined percentage of neuron units in the previous layer every batch