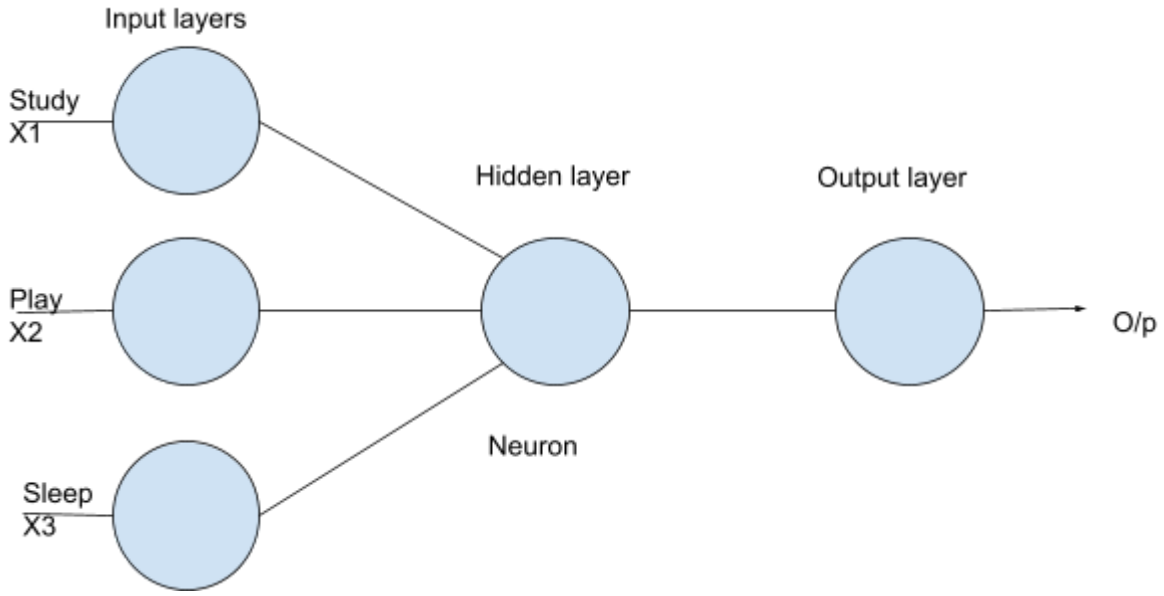
What is the perceptron?

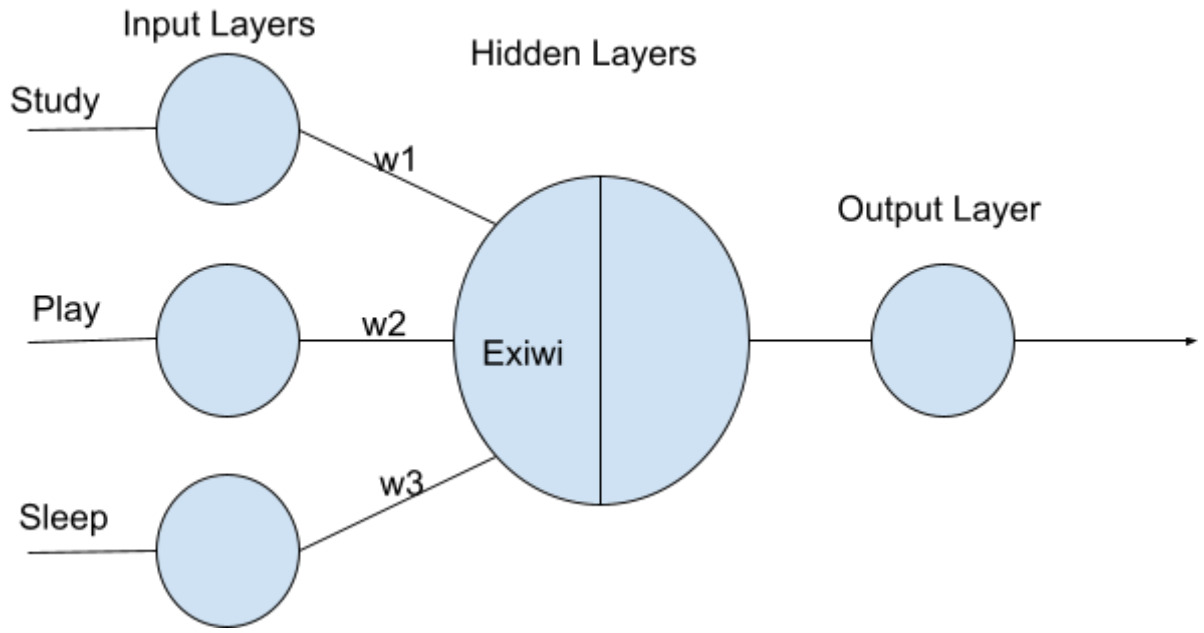
A perceptron is an artificial neural network and it's the simplest form of neural network. Primarily, it is very helpful for binary classification tasks. It takes several binary inputs, combines them by using weighted sums and applies a threshold to produce a single binay output. Mathematically, it can be represented as a **single-layered neural network** with no hidden layers, also there is one more neural network which is **multi-layered neural network**.

A perceptron is adjusts the weights of inputs based on the error in classification, **aiming to minimize** the errors over time. While it's limited to linearly seperatable data, it's formed the basis for complex neural network architecture and development of modern machine learning techniques.

Now, let's take one example , we have dataset of students if they pass or fail, there are 3 inputs which are **study, play, sleep** and 1 output **pass/fail.**



Data will be passed through input layers and then hidden layer will pre-process the data ,eventually we will get output. Whenever we create neural network from scratch, we need to train them based on the outputs.



First, we assign the weights (W1,W2,W3), and in hidden layers , we do two things first do the summation of XiWi and then pass it in activation function.

The Equation is:

Summation of XiWi = X1W1 + X2W2 +X3W3 + … + XnWn ,
also can write wTx(W transpose x)

Let's understand the importance of Weights, If we put hot object on my left hand, what happen?

We take our hand away our right hand right? as neuros pass through our right hand , those are getting activated and send signal to bain that some hot object touch your right hand and we move our hand. **So This Weights will help neurons to act whether it should be activated or what level it should be activated.**

During the training of neural network, we take care of weights that which values should take so that neurons get activated till the level. So, basically weights say how much neurons should activated or deactivated.

Suppose we take weights value as 0, then definitely our Summation of XiWi will be Zero, So to overcome these we add one more paramete which is **Bias.**

Now, **let's understand what is activation function,** one of the activation function is **Sigmoid activation function which is used for binary classification.** The activation function is just a mathematical function applies to a neuron in neuron network that primary purpose to introduce non-linearity into the network, enabling it to learn and perform complex tasks.

Sigmoid Function = $\frac{1}{1+e^{-y}}$ **,** here y = ($\sum XiWi$ + bias),

The **sigmoid function** gives output **between 0 and 1,** we put condition if **grater than or equal to 0.5 then consider it as 1, and if less than 0.5 then consider it as 0.** Eventually, this activation function say whether neuron will activate or not.

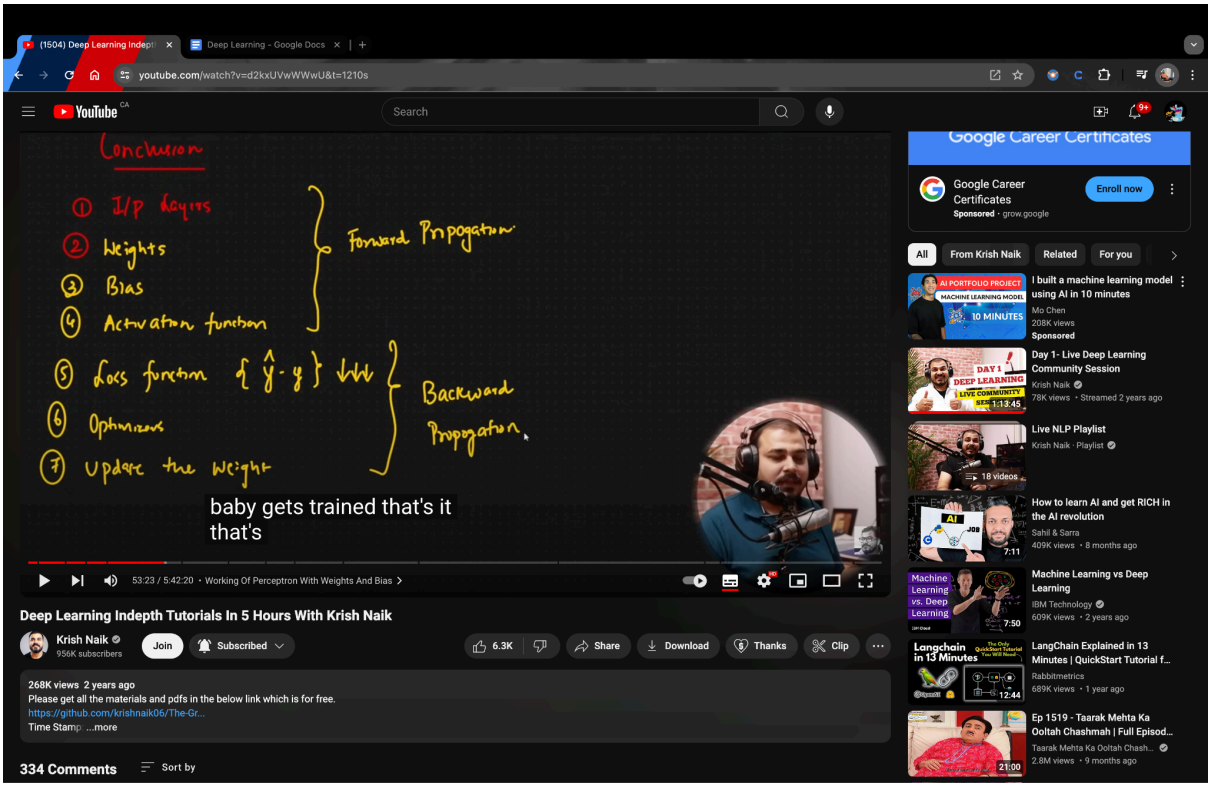
Now, this all happen in every neuron, first do the **summation + Bias** and pass it in **activation function,** after that **hidden layers output (O1)** , it will multiply with **another weights** and pass it in **output layer** where it pass in **activation function** and give **output 0 or 1.** There is another linear activation function for linear regression.

What is Forward Propagation?

When we pass our inputs, then make summation with weights and pass it in activation after then till the output layer , so this whole process is known as **Forward Propagation.**

The important thing, suppose we take inputs 7,3,7 and get output 0 which is **predicted value(y^)**, however, our **actual value(y)** is 1.So, **the difference (y - y^)** is 1 which is wrong , so what we need to ensure that **difference is near to 0.** This **(y - y^)** is known as **loss function.** The aims of loss function is to minimize the difference to 0.

What we do if the loss function is high ?, then we do back propagation.



What is Backward propagation ?

The main aim of **back propagation** is to **update the weights,** and it's the responsibility of the **optimizers** , So when we do back propagation **optimers ensure to update the weights**.

Two things , first **weight update formula, then chain rule of derivative**

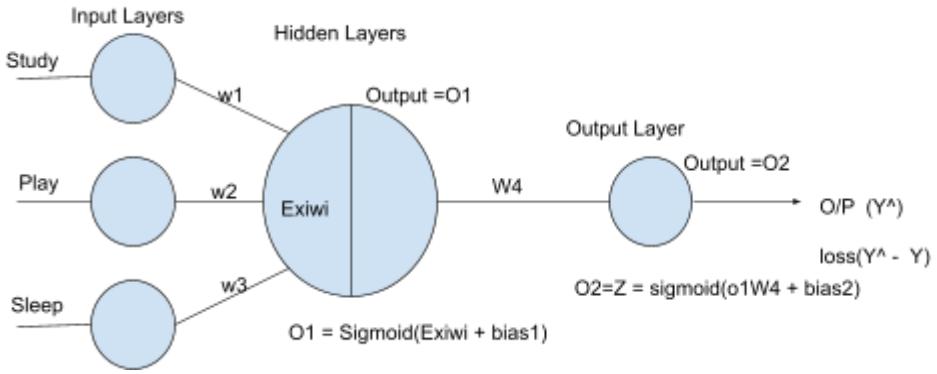
How we update the weights?

There is a formula which is,

$W_{new} = W_{old} - n(dL/dW_{old})$, here **n** is learning rate, **dL** is derivative of loss function , **dW_{old}** is derivative of old weights.

As we know we take learning rate value small to get global minima , if we take learning rate value high, we will never reach global minima and it jump from here to there and in some cases we never reach to global minima. So , it woul be an ideal to use learning rate 0.001 or 0.01.

Let's understand Chain rule of derivatives



How do we find **dL/dWold,** let's update w4 value, so

$$\mathbf{W4new = W4old - n(}\frac{dL}{dW4old}\mathbf{)}\ ,$$

We know how we get our loss function value, with help of **O2** we get that value So we can write **dl / dO2**, now according to chain rule next step is to find **O2** where O2 is dependent on **W4.**

$$\frac{dL}{dW4old} = \frac{dL}{dO2} * \frac{dO2}{dW4old}$$

We also update the bias in back propagation. It has the same formula as used for weights.

$$\mathbf{B2new = B2old - n(}\frac{dL}{dB2old}\mathbf{)}\ ,$$

So far we see how tp update **W4 where** we use chain rule **now , how** we update the W1,W2,W3. For example, let's find W1.

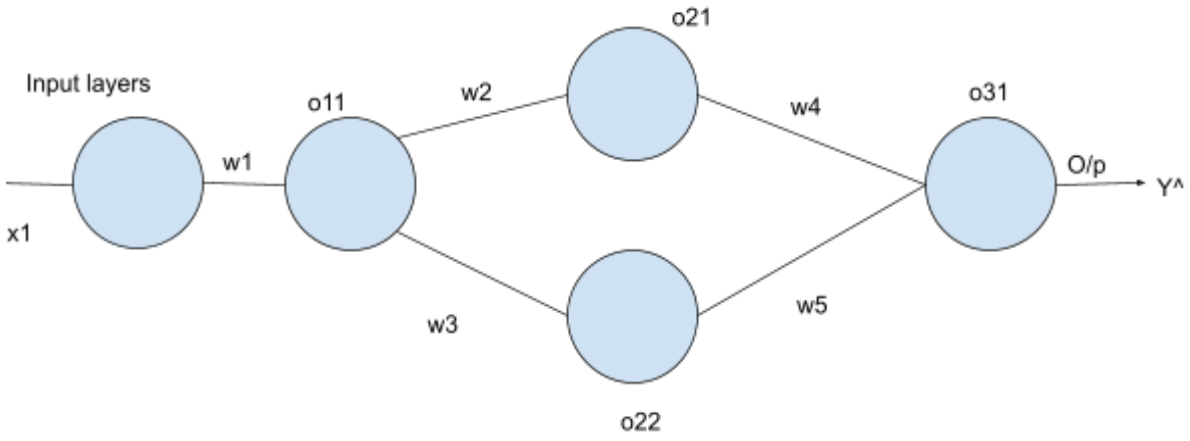
$$\mathbf{W1new = W1old - n(}\frac{dL}{dW1old}\mathbf{)}\ ,\ \text{how we find}\ \frac{dL}{dW1old}\ ?$$

We know that **loss function** is dependent on **O2**, while **O2** is dependent on **O1**, also **O1** is dependent on **W1**.

$$\frac{dL}{dW1old} = \frac{dL}{dO2} * \frac{dO2}{dO1} * \frac{dO1}{dW1old},$$ we can also expand this

formula by adding $\frac{dO2}{dW4}\frac{dW4}{dO1}$, **eventually** we get $\frac{dL}{dW1old}$

For W2 and W3 we do the same thing.



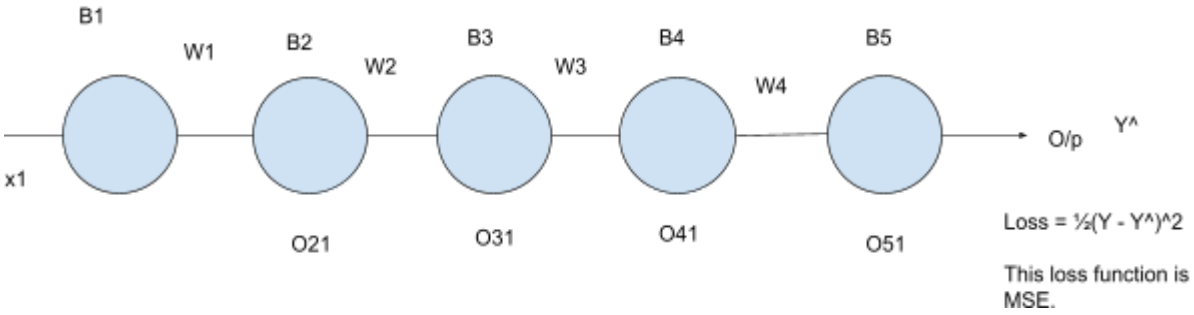
Now, if we want to update the **W1,**how we do that ?

As usual, $\mathbf{W1new = W1old - n(\frac{dL}{dW1old})}$

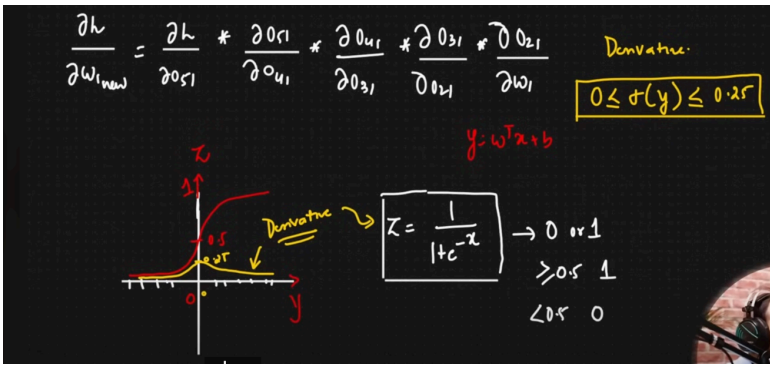
$$\frac{dL}{dW1old} = \left[\frac{dL}{dO31} * \frac{dO31}{dO21} * \frac{dO21}{dO11} * \frac{dO11}{dW1old}\right] +$$

$$\left[\frac{dL}{dO31} * \frac{dO31}{dO22} * \frac{dO22}{dO11} * \frac{dO11}{dW1old}\right]$$

Vanishing Gradient Problem



During back propagation, we update the weights and also use chain rule of derivatives. After then, we apply activation function, let's consider here we use sigmoid function.



In sigmoid function, our output between 0 and 1. We always get the **value of derivative of Sigmoid Function** between **0 and 0.25.**

We use sigmoid function in every output of neuron.

$$\frac{dL}{dW1old} = \frac{dL}{dO51} * \frac{dO51}{dO41} * \frac{dO41}{dO31} * \frac{dO31}{dO21} * \frac{dO21}{dW1}$$

Let's consider each derivatives output as per the below.

$$\frac{dL}{dW1old} = 0.25 * 0.15 * 0.10 * 0.5 * 0.05$$

As we go ahead, derivatives value will be reduced. When we multiply with small number, we get small value here. Then we use this value to update the weight. For example, we use above value to update weight.

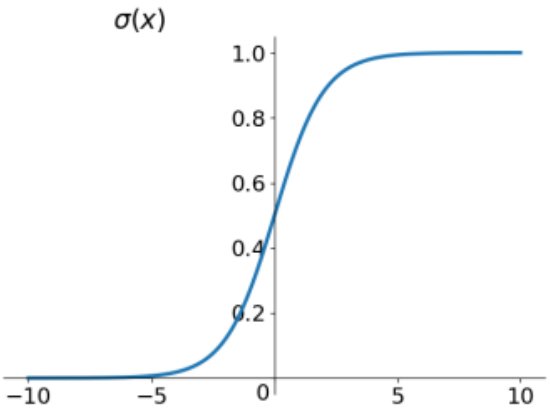
$$W1new = W1old - n(\frac{dL}{dW1old})$$

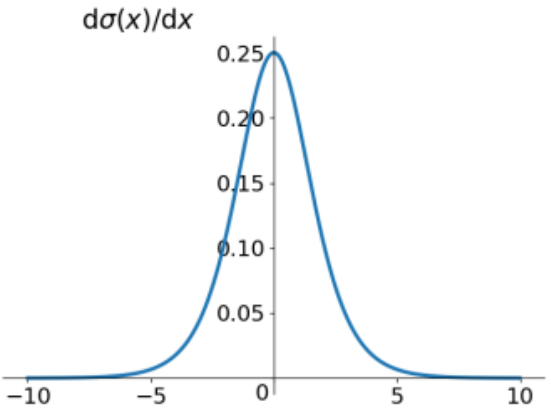
Here, larning rate value is small so when multiply it with $\frac{dL}{dW1old}$, again it's small value. So **new weight value is nearly equal to old weight value.** There is hardly change in new value , So basically we arn't getting updated weight's value and this is called **Vanishing gradient Problem.**

To Avoid this problem, we used different activation function such as Sigmoid, Rely, Tan h , leaky Relu, Pre relu.

Types Of Activation Functions

1. Sigmoid Function





1. In the process of neural network backpropagation, we all use the chain rule of differential to calculate the differential of each weight w. When the backpropagation passes through the sigmod function, the differential on this chain is very small. Moreover, it may pass through many sigmod functions, which will eventually cause the weight w to have little effect on the loss function, which is not conducive to the optimization of the weight. This The problem is called gradient saturation or gradient dispersion.
2. The function output is not centered on 0, which will reduce the efficiency of weight update.
3. The sigmod function performs exponential operations, which is slower for computers. As we know sigmoid function is also exponential function, whenever we perform this function it takes time, **Time complexity is higher since there is a Exponential function.**

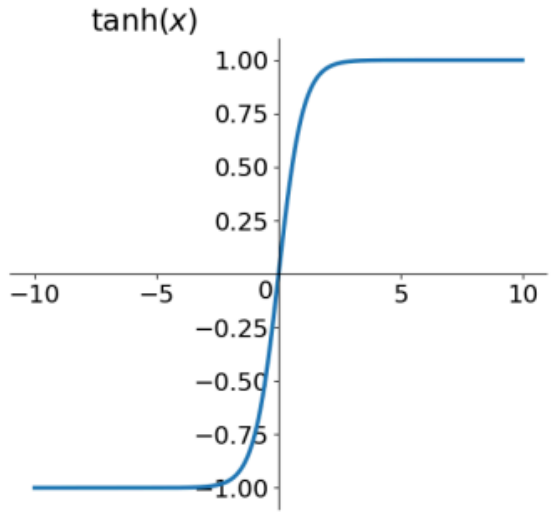
Advantage Of sigmoid function

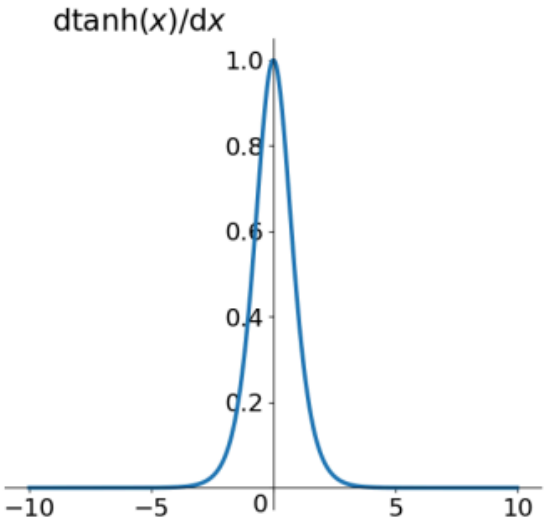
- Smooth gradient preventing jumps in output values. In above image, in leftside curve, we have smooth gradient because of it the convergence will be quickly done and help us to work in better way. However , in above second point, what is the **mean of centered not Zero.** It means that if our curve passing through Origin (**where curve passing through 0, mean =0)**, that is called zero centered curve. Weights updation become very easy.
- Output values bound between 0 and 1, normalizing the output of each neuron.
- Clear predictions, i.e very close to 1 or 0.

Disadvantages of sigmoid function

- Prone to gradient vanishing
- Function output is not Zero-centered
- Power operations are time consuming

2. Tanh Function





$$\text{Tanh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

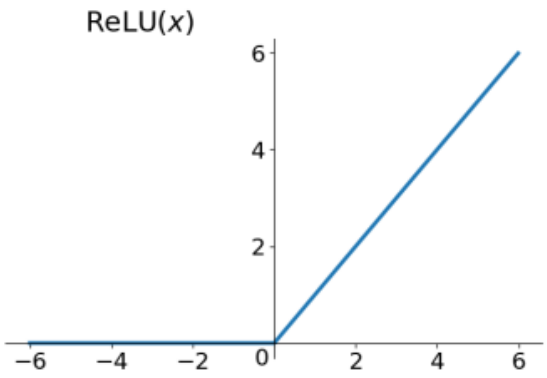
Tanh is also known as **hyperbolic tangent function.** Whenever we apply Tanh(x) function, we get value **between 1 and -1,** while in the case of derivatives value **between 0 and 1.**

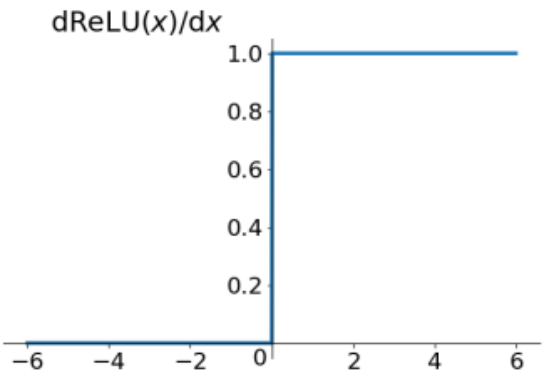
In the previous activation function, it ranges **between 0 and 0.25**, does it prevent vanishing gradient problem? **Answer** is that there is a still chance of get vanishing gradient problem. In deep neural network, at one point we may get this problem.

The difference between **Tanh** and **Sigmoid** function is that Tanh **zero-centered is solved here.**

Generally, in binary classification problem, the **Tanh is** used in **Input layers,** while the **sigmoid function is used** in **Output layer.**

3. ReLU Function





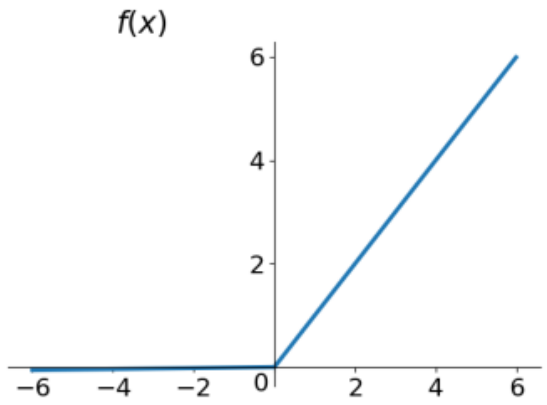
Relu(x) = max(0,x)

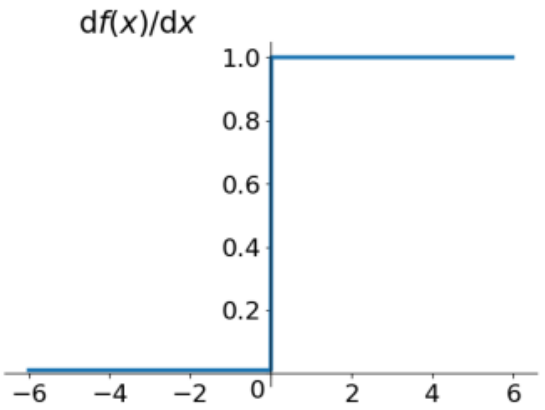
The value is either **Zero** or **whatever the value of X, if the X value is a negative , it will be Zero.** Whenever we try to find out the derivatives of value the value is **either 0 or 1.**

Let's say, during the derivatives , one of the weight's value is **zero.** During the back propagation it one of the neural becomes **zero or negative**, entire derivative of chain rule become **Zero. So, our new weight is nearly equal to zero.** This is the major disadvantage of **ReLU Function, other than** this is doing an exceptional performing as there is not any **Exponentional function. Which means one negative number entered, ReLu will die.**

Also it is not zero-centered it passes through zero but not getting negative problem, apart from this it solves **sigmoid and Tanh function problems, and faster too.**

4.Leaky ReLu Function

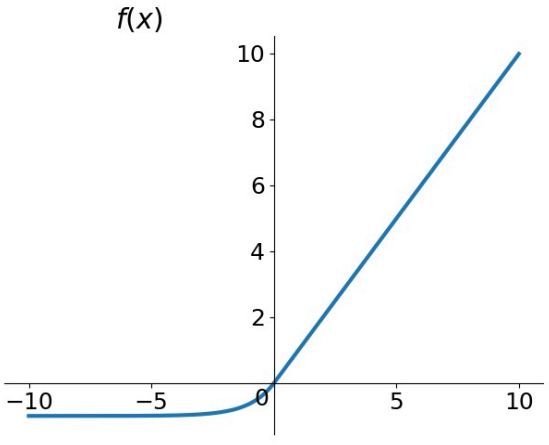


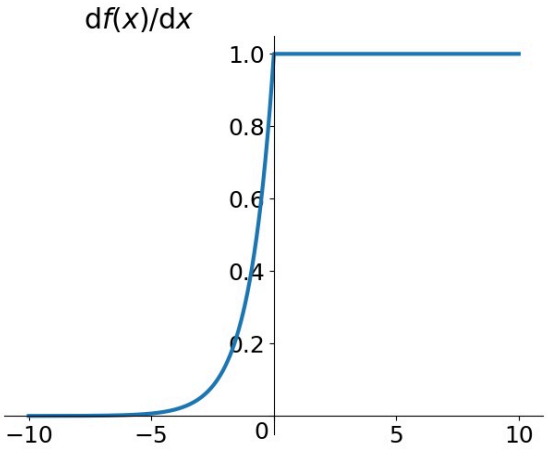


F(x) = max(0.01x + x)

There is only one thing will be changed in function is that now it will start from **0.01(x) and not from 0.** In this activation function, it will never be zero. It solves dead ReLU problem. **However,** There is No problem to use Dead ReLU, i**t has not been fully proof that Leaky ReLU is always better than ReLU function.** Another intuitive idea is a parameter-based method, Parametric ReLU : f(x)= max(alpha x,x), which alpha can be learned from back propagation.

5. ELU(Exponential Linear Units) Function





$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$

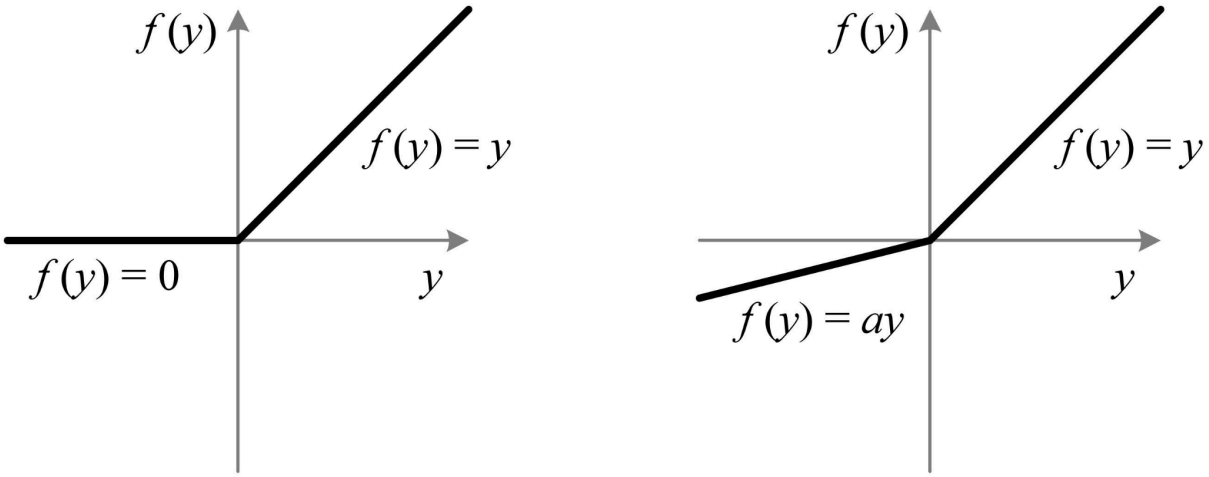
The derivatives also have value range between 0 and 1.

ELU is also proposed to solve the problems of ReLU. Obviously, ELU has all the advantages of ReLU, and:

- No Dead ReLU issues
- The mean of the output is close to 0, zero-centered

One small problem is that it is slightly more computationally intensive as if X is smaller than 0 we use different function which has exponential function. Similar to Leaky ReLU, although theoretically better than ReLU, there is currently no good evidence in practice that ELU is always better than ReLU.

6. Pre ReLU



$$f(y_i) = \begin{cases} y_i, \text{ if } y_i > 0 \\ \alpha_i y_i, \text{ if } y_i \leq 0 \end{cases}$$

PReLU is also an improved version of ReLU. In the negative region, PReLU has a small slope, which can also avoid the problem of ReLU death. Compared to ELU, PReLU is a linear operation in the negative region. Although the slope is small, it does not tend to 0, which is a certain advantage.

We look at the formula of PReLU. The parameter α is generally a number between 0 and 1, and it is generally relatively small, such as a few zeros. When α = 0.01, we call PReLU as Leaky Relu , it is regarded as a special case PReLU it.

Above, y_i is any input on the ith channel and a_i is the negative slope which is a learnable parameter.

- if a_i=0, f becomes ReLU
- if a_i>0, f becomes leaky ReLU
- if a_i is a learnable parameter, f becomes PReLU

Techniques Which activation function to use.

Let's say we use **binary classification problem** and have a neural network and in that network, we use **ReLU function in hidden layers**, also it's binary problem we use **sigmoid in output layer**. Now, if Convergence is not happening So we use other functions such as Pre ReLU, ELU, and so on, But it's binary problem so we **don't replace activation function of output layer.**

Now, in Multiclassification problem, We use ReLU function in hidden layers and if Convergence is not happening So we use other functions, and in output layers we use softmax function.

In the case of regression problem, as usual we use **relu for hidden layers** and for output layers, use **Linear function. Also,** there will be separate Loss function.

Types of Loss functions

In Deep learning, There are two things in ANN, 1. Regression and 2. Classification

In **regression**, There are **three Loss functions,**

1. **Mean squared Error (MSE)**
2. **Mean Absolute Error (MAE)**
3. **Hubor Loss**

Let's see the difference between Loss function and Cost function,

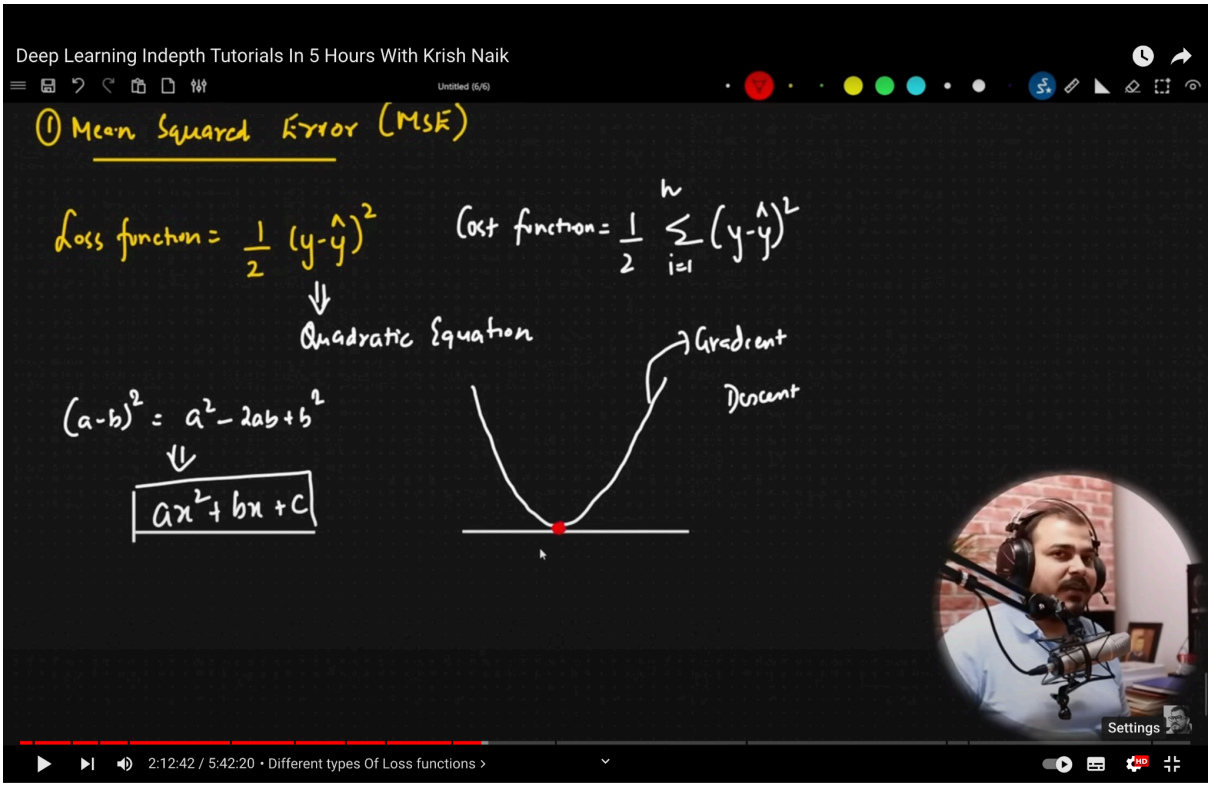
For an example, we have 100 records and as usual we do forward propagation, find loss function (Y^) and do backward propagation. This happened for **every single records which is not an efficient scenario.**

Now, we pass 10 out 100 records for every forward propagation, find loss for 10 records, formula will be changed from **loss function = ½(y- y^)^2 to Cost function = ½ summation from 1 to10 (y - y^)^2.**

In Loss function, we provide one records, **while in Cost function,** we provide **bunch of data points.**

1. Mean squared error (MSE)

Loss function = ½(y - y^)^2 , Cost function = ½ summation from 1 to n(y- y^)^2



Advantages of MSE

1. It is **Differentiable,** it means that we have seen how weight updation works.
2. It has only one global minima or local minima.
3. It converges faster

Disadvantages of MSE

1. It isn't robust to outliers,

   Let's say we have outliers then we see changes in best fit line, why we see changed in best fit line as we penalizing our error , we sequared our error thats why we see changes in our best fit line.

2. Mean Absolute Error (MAE)

Loss function = ½| y - y^ | , Cost function = ½ summation from 1 to n **| y - y^ |**

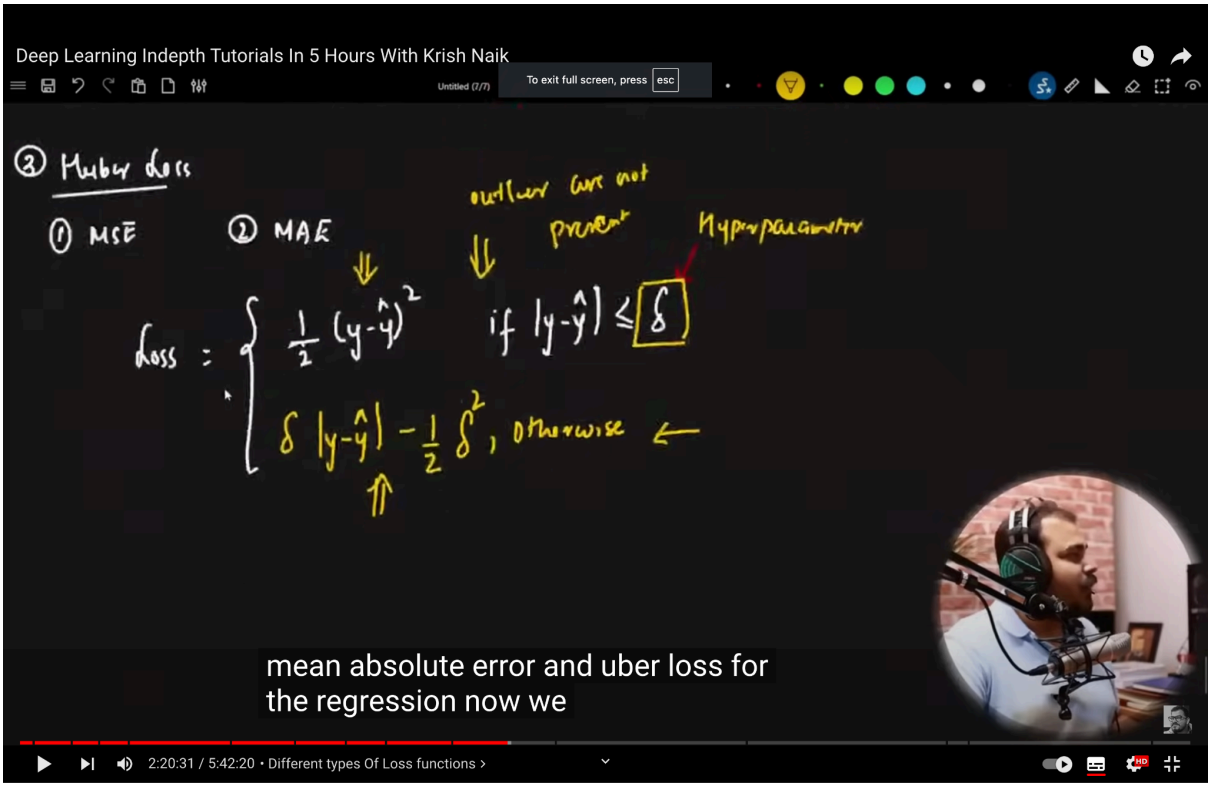
Advantages of MAE

1. Robust to Outliers.

   Even tough we have outliers, there will be minor shift in best fit line.Also, MAE doesn't have curve proprly it's divided into two parts. So it has to take subgradient, it means we have to divide into small parts and then find slops. Thats why it takes much times.

3. Huber Loss:

It's the combination of **MAE & MSE**



In Classification, there is a term **cross entropy, and it** has two types 1. **Binary Cross Entropy (**Use for binary classification **) 2. Categorical cross entropy (**Use for multi-classification**)**

1. **Binary Cross Entropy:**

   Loss = -y log(y^) - (1-y) log(1-y^) => Logistic regression

   Log loss => Loss = { -y log(1-y^) if y=0 , -log(y^) if y = 1 }, we use sigmoid function to find **y^**

   y^ = 1/1+ e^x

   After that we apply Y^ in log loss and then we do back propagation.

2. **Categorical cross entropy:**

   Let's sau we have dataset as per the below table

f1	**f2**	**f3**	**O/P**
2	3	4	Good
5	6	7	Bad
8	9	10	Neutral

Then With the help of one-Hot encoder, There are new three column will be added

Good	**Bad**	**Neutral**
1	0	0
0	1	0
0	0	1

We just put 1 as per the output and rest will be considered as 0. We apply the below formula for loss function.

$$\textbf{loss(Xi , Yi) = } -\sum_{i=1}^{C} Yij \ * \ ln(\hat{Yij}) \textbf{ ,} \text{ i = Row, j = Column}$$

Yi = [yi1, yi2, yi3, … , yic] , numbers of category

Yij = {1 If element is in class , 0 otherwise }

$Yij^{\wedge}$, **Using softmax activation function** that we use in output layer for **multi-classification problem**.

$$P(y = i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Suppose we have neural network in which there are **three output layers**. Now, in **hidden** layers we use **relu activation function** and with the help of it , we find weights value ,let's use above formula, assume weights value are 10,20,30,40,50

P = e^10/e^10+20+30+40+50

Now, for each output layers we get **probability such as 0.5,0.3,0.2** and sum of all output l**ayers will be 1**, and **Whichever output layer has the highest probability among them is taken for final output. For example,** p is 0.6 for good, 0.3 for bad and 0.1 for neutral so total sum is 1 and highest is good, eventually we take **Good as** final output.

Conclusion:

In binary classification, we use **relu** activation function for **hidden layers** and **Sigmoid** for **output layer** and we use **binary cross entropy** for **loss function , while** we use as usual **relu** activation function for **hidden layers** and **Softmax** for output layers in **multi-classification** and we use **category cross entropy** for **loss function.**

In regression, we use **relu** for **hidden layers**, while **Linear** activation function for **output layer** and also use **loss function** which are **MAE, MSE, Haber loss.**

Optimizers

1. **Gradient Descent**

   As we know what Gradient descent is and already discuss it. When one **forward Propagation and one backward propagation completes then it calles Epoch.**

   The major disadvantage of Gradient descent is It's quite expensive when we have huge amount of data. For an example, we have 1M data, So in every Epoch it passes 1M data, for this must need a heavy system along with good ram and Gpu.

   So **what can we do?**

   We can use another gradient descent which is **stochastic gradient descent**

2. **stochastic gradient descent**

   In this gradient descent, Let's say we have 1M data, we pass 1 record in each Epoch this is known as iteration, So There will be 1M epoch, it means 1m iterations. One Iteration means in that epoch we pass only 1 records.

   For this we don't need much ram, however , **It has one major disadvantage is that convergence will be very slow. So, Time complexity is also high.**

3. **Mini-Batch Stochastic gradient descent (SGD)**

   In this gradient descent, we set batch-size and that size pass in each epoch.With the help of this we can minize our resource intensive, improved convergence result and also Time complexity is also better then **SGD.**

   For Example, we set batch-size to 1000, it means that we pass 1000 records in one epoch, it is one iteration, so for 1M data There will be 1000 iterations.

Gradient descent will reach Global minima by taking small small steps, while **in SGD, it has zig-zac moments then it reaches global minima.**On the other hand, Mini-Batch SGD also have zig-zac moments but it's not high as compared to SGD.

That moments known as **Noice, and** SGD has very high noice, and Mini-batch SGD also have noice but minimal. As we have high noice as it takes more time to reach global minima.

It's necessary to remove that noice, How we remove it ? Using **Momentum.**

4. **SGD with Momentum**

What the momentum does?

Well it helps to make a smoother journey to reach global minima and reduce noice.

So far, we know how to Update the weights and bias using below formula

$W_{new} = W_{old} - n\,(dl/dW_{old})$, $B_{new} = B_{old} - n(dl/dW_{old})$

There will be minor change in formula, which is like

$W_t = W_{t-1} - n(dl/dW_{t-1})$, T means current time and t-1 means previous time.

Let'say we have some columns from t1 to tn and theirs values from a1 to an respectively.

So value of t1 (V_{t1}) = a1,

Also ensure to perform weighted average between t1 and t2, with this help we can do forecasting.

For forecasting, we also need previous data, if want to apply Exponential weighted average, we will assign a weight using below formula.

$V_{t2} = ß * V_{t1} + (1-ß) * a2$, ß is hyperparameter, and a2 is current value of v2

Beta tells us on which values should focus more either previous time value or current time value.

Beta value lies between 0 and 1. For example, beta value is 0.96, apply this value in above formula , so we give more importance to Vt1 value.So, where do we apply this formula ?

Now, **$W_t = W_{t-1} - nV_{dw}$,** here **$V_{dWt} = ß * V_{dWt-1} + (1-ß) * dL/dWt-1$** and this helps to bring exponential curve initially.

Interview question

5. **AdaGrad -> Adaptive Gradient Descent**

As we know during the weight updation, **Learning rate** is an important factor which has a **fixed value** that helps us to maintain speed of the convergence to reach global minima**,** but can we change the learning rate in such a way that initially it's very high and as we near to the global minima learning rate will decrease.

Now, the formula of weight updation for AdaGrad is similar but there is one thing is different which is **learning rate.**

Before, Wt = Wt-1 - n(dl/dWt-1) **,** **now** Wt = Wt-1 - n' (dl/dWt-1) , n' = n / $\sqrt{\alpha t + \varepsilon}$

n' = n / $\sqrt{\alpha t + \varepsilon}$, here epsilon is very small number to avoid dividing by zero condition.

$$\alpha t = \sum_{i=1}^{t} (dl/dWt)^2$$

n' = n / $\sqrt{\alpha t + \varepsilon}$, in this formula learning rate always divided by **HIgh value,** so eventually as we go ahead n' will decease to reach global minima.

However, αt might have high number , so because of this learning rate will be changed and that will become negligible at some times.So, in order to prevent this we **use below optimizer.**

6. **Adadelta and RMSprop**

To prevent problem of **AdaGrad,** we use below formula.

n' = n / $\sqrt{Sdw + \varepsilon}$, here we apply exponential weighted average

Sdw = 0 , Sdwt = **ß*** Sdwt-1 + (1- **ß**) $(dl/dWt - 1)^2$,

Let's say our **ß =** 0.95 , Sdwt = **0.95*** Sdwt-1 + **0.05** $(dl/dWt - 1)^2$,

Above $(dl/dWt - 1)^2$ is controlled by (1- **ß**) which is **0.05, so** does't matter that $(dl/dWt - 1)^2$ value will increase, eventually it's controlled by (1- **ß**).

Because of Sdwt, there will be a slow decrease in learning rate.

7. Adam Optimizer

We know for smoothing we use **SGD with momentum (V_{dw}) ,** and to change learning rate as we reach global minima we use **AdaGrad (αt) , and Adadelta(S_{dw}) + RMsprop So** we can't do smoothing here for that we use **Adam Optimizer.**

Adam Optimizer is combined of **Momentum + RMSprop. Also,**Same formula apply for bias also.

Vdw, Vdb, Sdw,Sdb initially all these values are 0.

How my **weight updation formula** look like, $W_t = W_{t-1} - n'V_{dw}$

How my **bias updation formula** look like, $B_t = W_{bt-1} - n'V_{db}$

In above formula, **n' = n / $\sqrt{Sdw + \varepsilon}$, VdWt = ß * VdWt-1 + (1-ß) * dL/dWt-1 ,** and **Vdbt = ß * VdWbt-1 + (1-ß) * dL/dWbt-1**

Therefore, Adam optimizer solve the problem of smoothing and ensure that learning rate is adaptive.

ANN practical implementation

For which algorithms Feature scaling is required?

Feature scaling is required for those where distance based problem, So if the value is bigger, calculation will take time.Also, wherever gradient descent or optimizers involve in that case scaling is also required.

It requires for Logistic regression, linear regression, KNN, K-means, ANN.

Three Important terms, firstly **Sequential,** it means a whole neural network ,included with forward and backward propagation. Secondly, **Dense,** it means we create neurals in other word **layers** such as input layers, Hidden layers, and output layers. Moreover**, Activation function,** we use it in hidden layer and output layer such as sigmoid, tan h , relu, prelu, Leakyrelu. Lastly, **Dropout,** let's say we crated interconnected neual network, in this nerual network, we get overfitting(accuracy is good on training data), but for test data accuracy is low(Underfitting). In order to reduce this, we introduce **dropout layer which is regularization parameter.** Let's say our drop our ratio is 0.3 for hidden layer that means 30% of neural are deactivated from entire neural network during **training**. We basically cut off the layers, also known as dead neuron.

Black box model VS White Box model

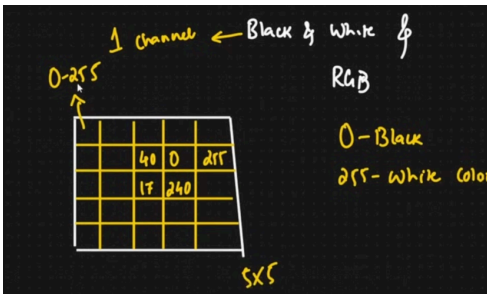
If we can checkout calculation of an algorithm than it's **white box model**, while the **black box** is unlike **whit box model**. For example, **random forest is black box model** as we know there are multiple decision tree so it's quite hard to check all calculations also **ANN is black box**, **whereas decision tree is white box model**, we can draw it even on peper and check all calculation.

Convolutional Neural Network(CNN)

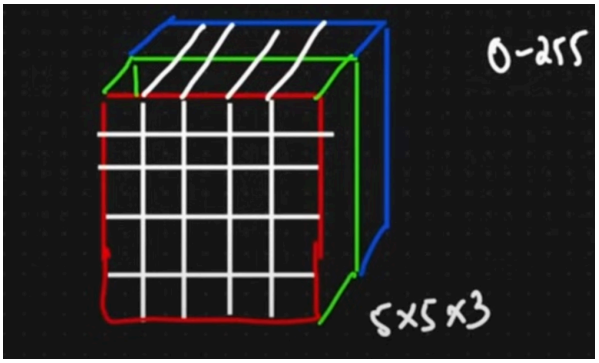
First, let's understand the basic of Image.So, there are basically two types of image 1. **Black & White , 2. RGB**

Black and white image has a single channel, it means eniter image either in black or white.

Let's say our image have 5x5 pixels, and each pixel ranges between 0 and 255, and 0 is nothing but black color, 255 is white color. Every pixle has different values based on the image composition.

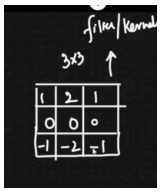


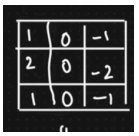
RGB image has three channel one is R channel , G channel, B channel , and all values range between 0 and 255 ans its 5x5 pixels and 3 means three channel



Let's say we have an image of 6x6 pixels and we pass it by **filter/Kernel** which has 3x3 pixels and imagine we got output of 4x4. Now, in convolutional we convert value between 0 and 1. Let's say we have some zeros and others are 255 in 6x6 pixels. So we divide each value with 255 so 255 values will be 1. This is the **first step** which we call **min max scaling.**

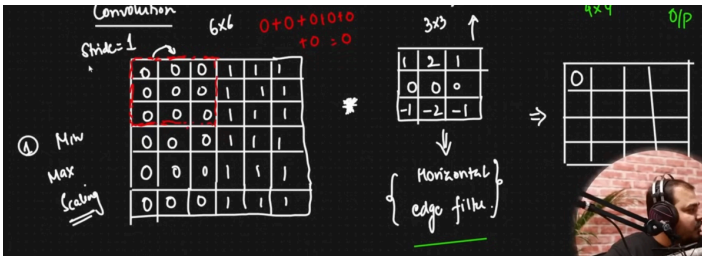
Now, as per the below image we provide values in filter and this filter is known as **Horizontal edge filter.**



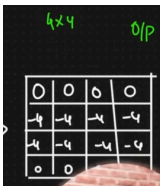


Horizontal edge filter **Vertical edge filter**

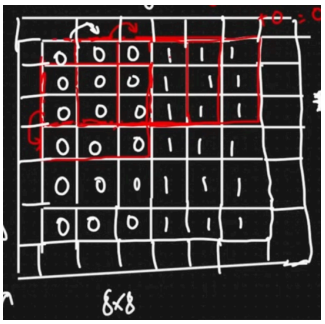
Let's understand how convolutional operation, it says that it take the filter (3x3) and place top of the image (6x6) from left. As we can see in below image then we multiply 11 of the filter with 11 input image that we do and then we add all the output of multiplication. And final value will be add in **4x4** output table.



Now, we move to one step to he right as seen in picture above which is known as stride =1. If stride is 2 then we jump two step to the right. Furthermore, when we have nothing left in our right side then we jump one step down of the input. **After completing** all calculations we get below table.



So, how do we get 4x4 output? That happened using formula which is **n-f+1.** Here **n** is our input pixel which is 6x6, **f** is our filter so n-f+1 = 6-3+1 = 4, therefore our output is 4x4.
So it means we loss some information from image. In order to preven this we used **padding.**
What is padding?padding means we apply layer on top of the input image so it becomes to 8x8 from 6x6 pixel.



Now, what values will be there when we appl one more layer, either we give 0 value or we give whatever the nearest value is.**So,** after padding, formula will be updated, it's **n+2p-f+1=**

Here, n is 6 and apply 1 padding so p=1, f is 3 so n+2p-f+1 = 6+2-3+1 = 6

What is the importance of padding? - To prevent the information loss of image

Let's say we have initialized filter randomly, now as we do back propagation in ANN, likewise we do in CNN , we update the filter **based on the input image.**

Where do we apply activation function in CNN? - After this all things/ Convolutional operations then we apply Activation function on top of our output which is **RELU** or others.

When we have output we apply activation function and back propagation helps to get derivatives to change the filter values. Now we know about **stride,** if its 1 then there is no problem, however if stride is 2 then there will be minor changes in formula, so it will be

(n+2p-f+1) / s = (6+2-3+1)/2

What is Max pooling?

Now we know what is convolutinal operations, take input then use filter and it may be 3x3 or anything also there are numbers of filter apply, after then we get out output along with activation function , this is Convolutional operations. So when we stack the same operations beside this one it is known as **Horizontal stacked.**

Let's say we have image of three cat, then we pass filter and i get the output and appling activation functions so this entire process is convolutional process. There might me more layers. After this, in learning techniques they have used one more layer which is called max pooling.

After that convolutional operations, we get **3x3 output** and using **2x2 max pooling,**Now why we use max pooling, when we use filters we are able to extract some informations.

Now there is a concept which is location invariant, it means that when we pass through more convolutional layers we should be able to extract more or better way and for that we use max pooling.

There are three types of pooling which are **Average pooling, min pooling and max pooling.**

Let's use max pooling here, so we place 2x2 max pooling on top of the 3x3 output, and we take the highest number, it means we get some clear portion of image and jump to the right as we do during convolutional operations.

In average poling and min pooling, we get average of all value and minimum value respectively. We use pooling **based on the problem statement.**

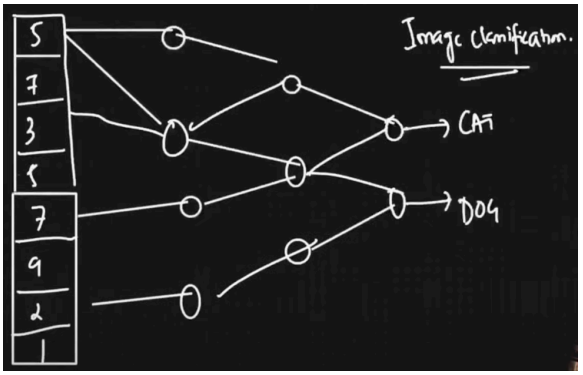
Flatting layer

It looks like ANN dense layer, so here whatever output we get from max pooling, we make it straight forward flattened.

Let's say we have 2x2 output with values of 5,7,3,5 and also we use different filters and accrodinng to those filter we get various outputs so we combine them and make flattened.

Like below table which is exactly looked ANN input layers and then we create neural network and suppose we have problem of image classification of dog and cat so there will be two output neural as shown in below image.

5
7
3
5
3
4



In short how CNN works: