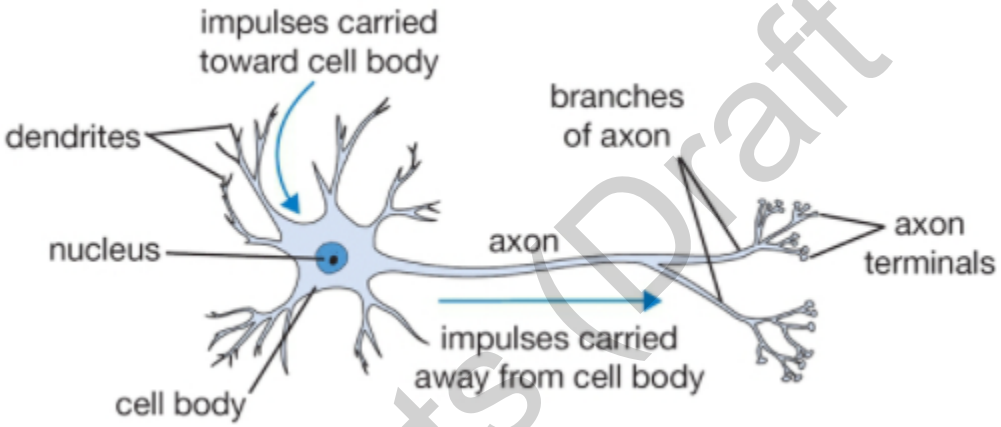
57.1 History of Neural networks and Deep Learning

The simplest Neural network model is the perceptron, discovered by Rosenblatt in 1957. It is loosely inspired by biological neurons.

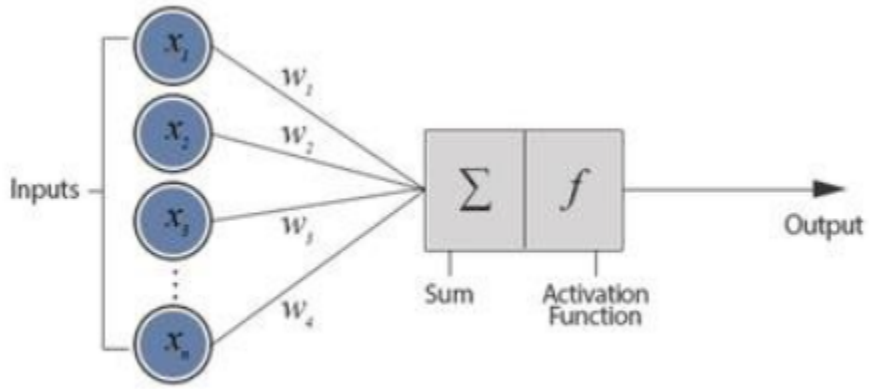
Biological Neuron:

The neuron is the basic building block of the human brain. The Mathematical representation of a simple neuron is called the perceptron. This is the basic structure of how a neuron looks like.



The Nucleus does mathematical computations when it gets some electrical pulses through dendrites and sends output to other neurons through the axon.

Mathematical representation:



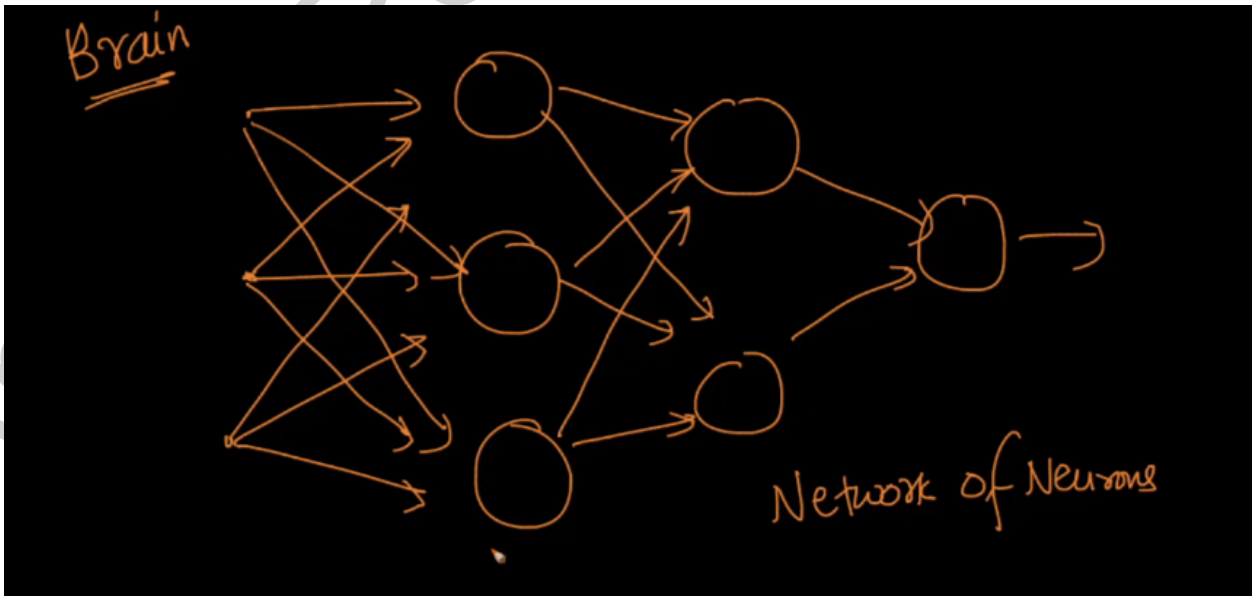
Let us take some inputs x1, x2,... And xn. And weights w1, w2, w3… and wn.

The weights are analogous to the thickness of dendrites in the biological neurons. The more the weight, more the importance is given to that input.

The output of a neuron is the function of the weighted sum of the inputs going into the neuron.

f(w1*x1+w2*x2+...+wn*xn)=output

In the actual brain, there are a huge number of neurons interconnected to each other.

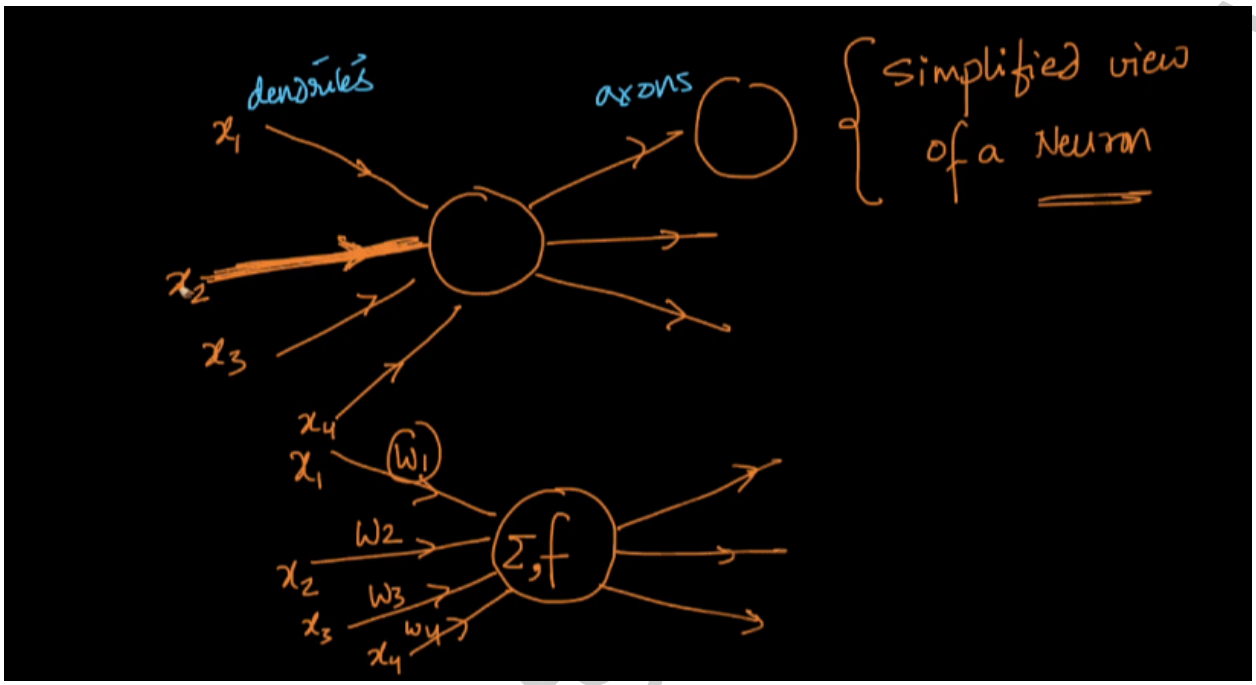


The first successful attempt to mimic this structure of neural network was in 1986 by Jeff Hinton and others. They come up with back propagation algorithms to train and learn from the network of neurons. But with the lack of data and computational power, all the hype on artificial Neural Networks died down by late 90's. This period is termed as "AI winter".

From 2012, people again started believing in ANN when imagenet, trained on the simple deep neural network, was able to beat all the other algorithms by huge margin. Even lots of popular companies like Google, Facebook, and Amazon started noticing the importance of deep learning. They started developing and investing in NN as these companies had lots of data and computational resources. Some of the greatest applications like voice assistants and Self driving cars are all powered by advances in deep learning.

Over the next few chapters, we learn about neural networks, some basic algorithms like Multi Layer Perceptrons and then CNN and RNN.

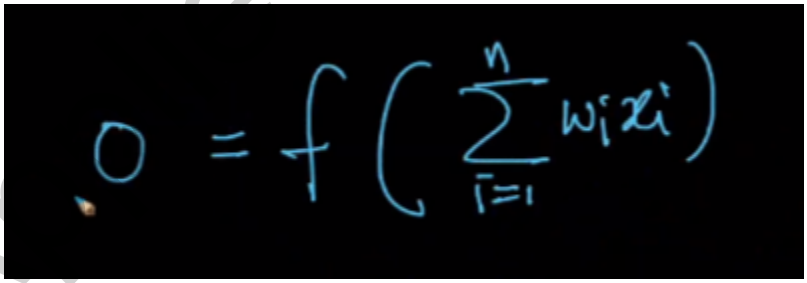
57.2 How Biological Neurons work?



Thicker dendrites in biological neurons are represented with larger weights in its mathematical representation. So, more importance is given to the input associated with that weight.

When the cell or nucleus gets enough input, it gets activated or fired, and then sends the output. So, the " f " is termed an activation function.

And the output of neuron will be



Where xi's are inputs to the neuron, wi's are the weights and f is the activation function.

57.3 Growth of biological neural networks

The growth of biological neurons is observed by visualizing the small part of the cerebral cortex.

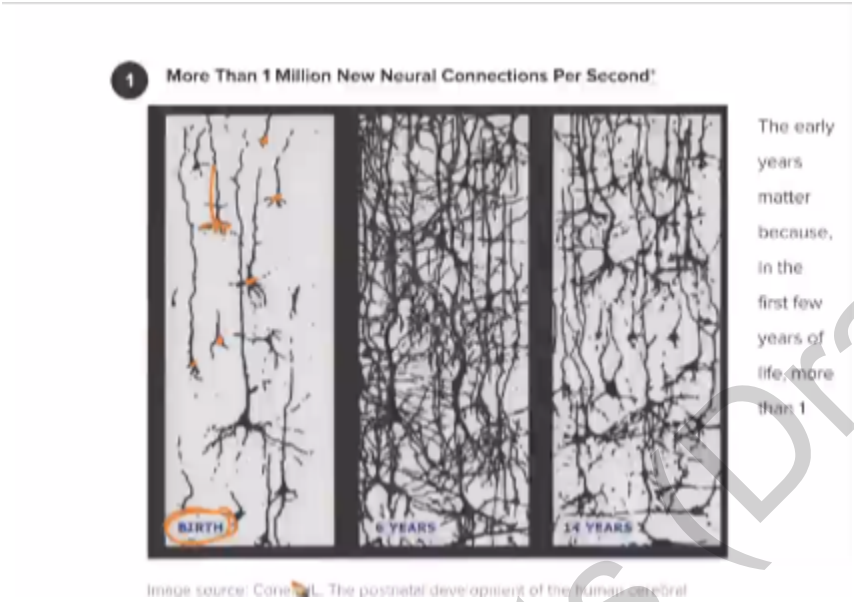
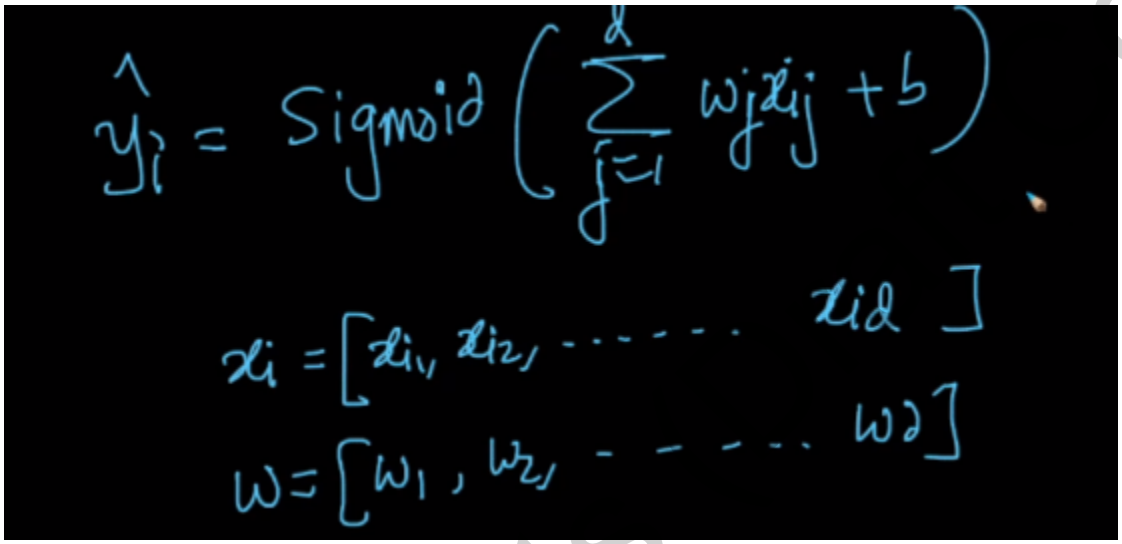


Image source: Conel, L. The postnatal development of the human cerebral

We can observe that the network is so sparse at the time of birth. And at the age of 6 years the network becomes so dense. Because a lot of brain development happens at that age based on the data they consume. Biological learning is nothing but the formation of connections between the edges among neurons. And the density of connections starts to decrease gradually with age.

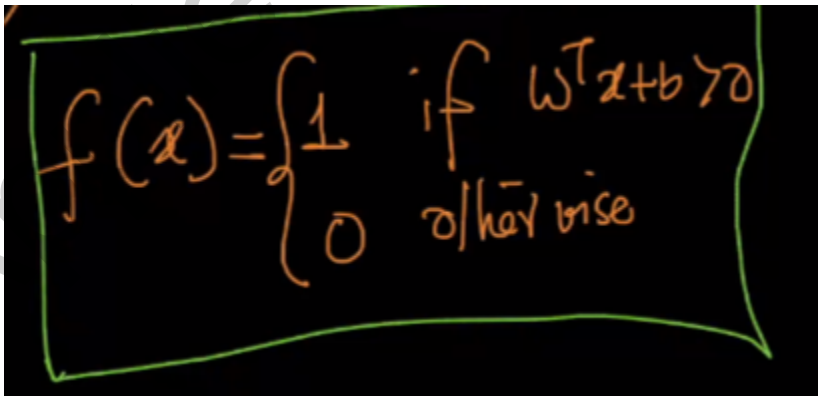
57.4 Diagrammatic Representation: Logistic Regression and perceptron

In Logistic regression, given any point xi, the predicted value of yi will be sigmoid(w.T*xi+b).



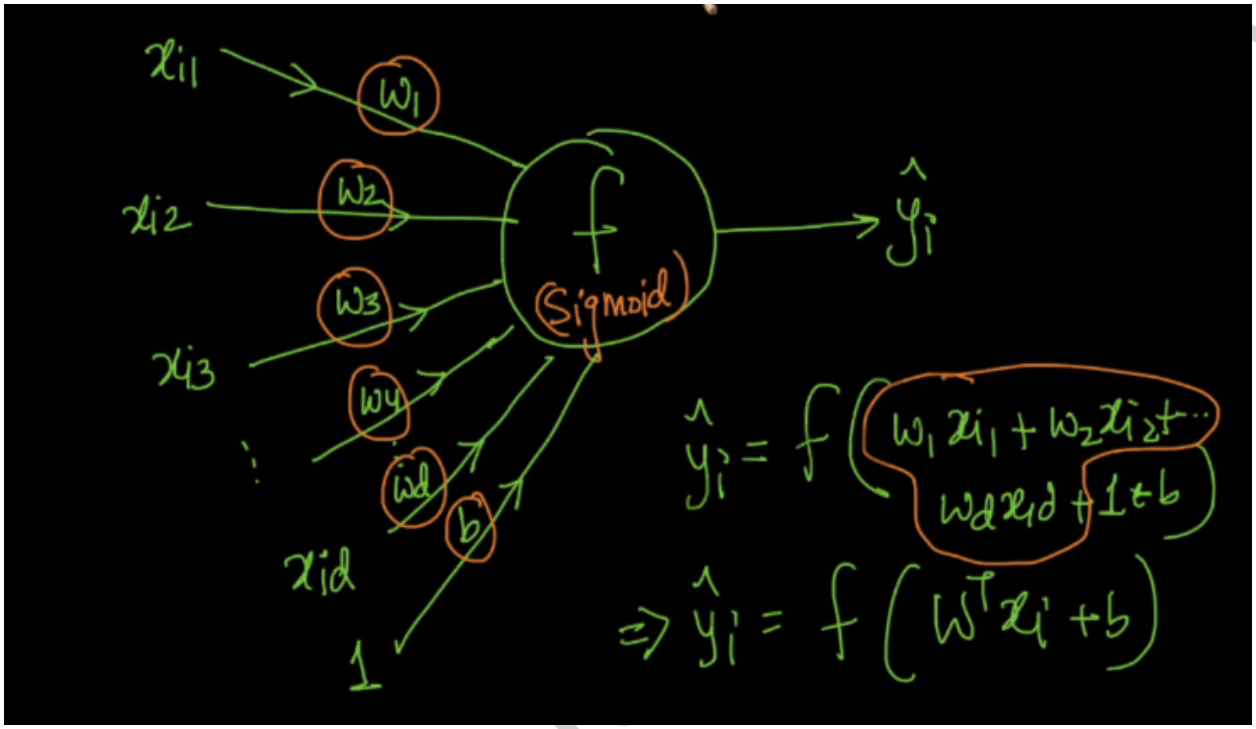
Perceptron:

The activation function in perceptrons is the thresholding function.
It is the only difference between logistic regression and perceptron.



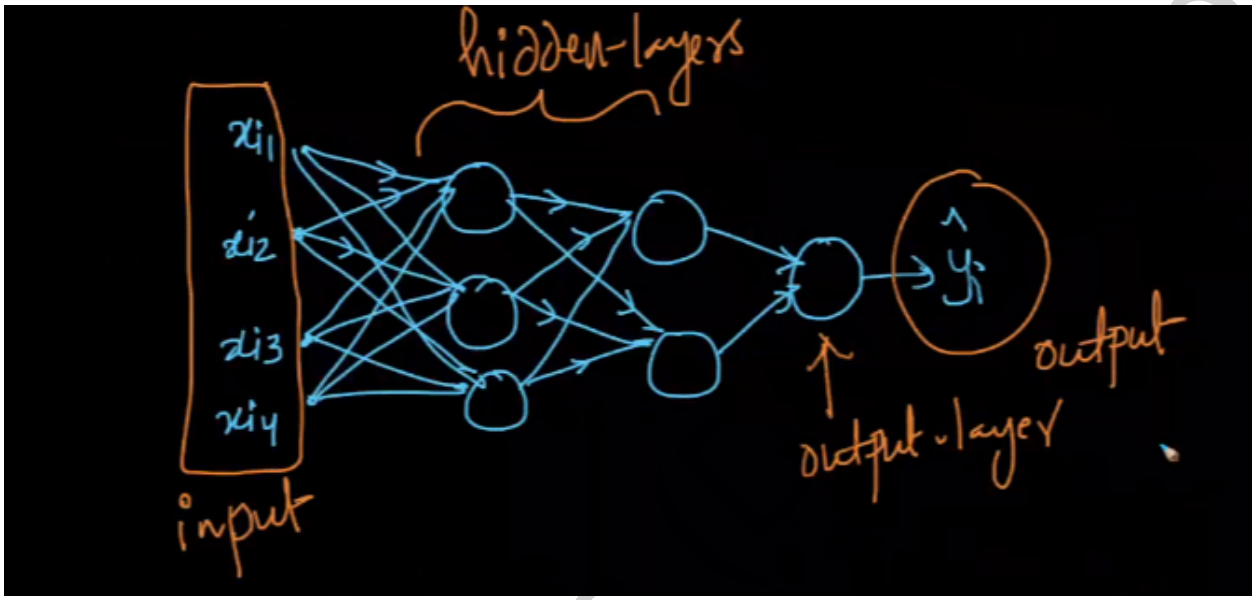
The Neuron interpretation of logistic Regression:

We can easily represent logistic regression as a simple neuron.
The activation function in Logistic regression will be the sigmoid function.



57.5 Multi-Layered Perceptron (MLP)

In a simple Neuron or single perceptron, we only have an input layer and output layer. But in Multi-Layered Perceptron, we have a network of neurons.

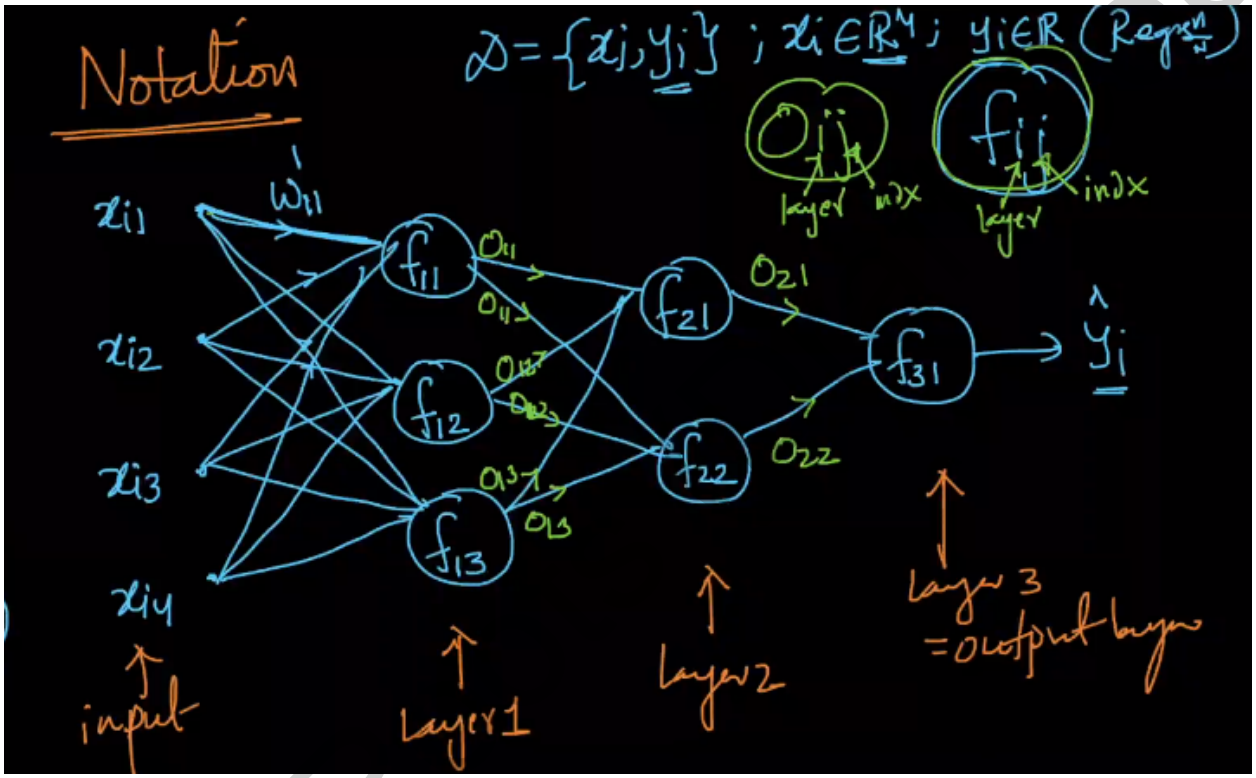


Why should we care about MLPs?

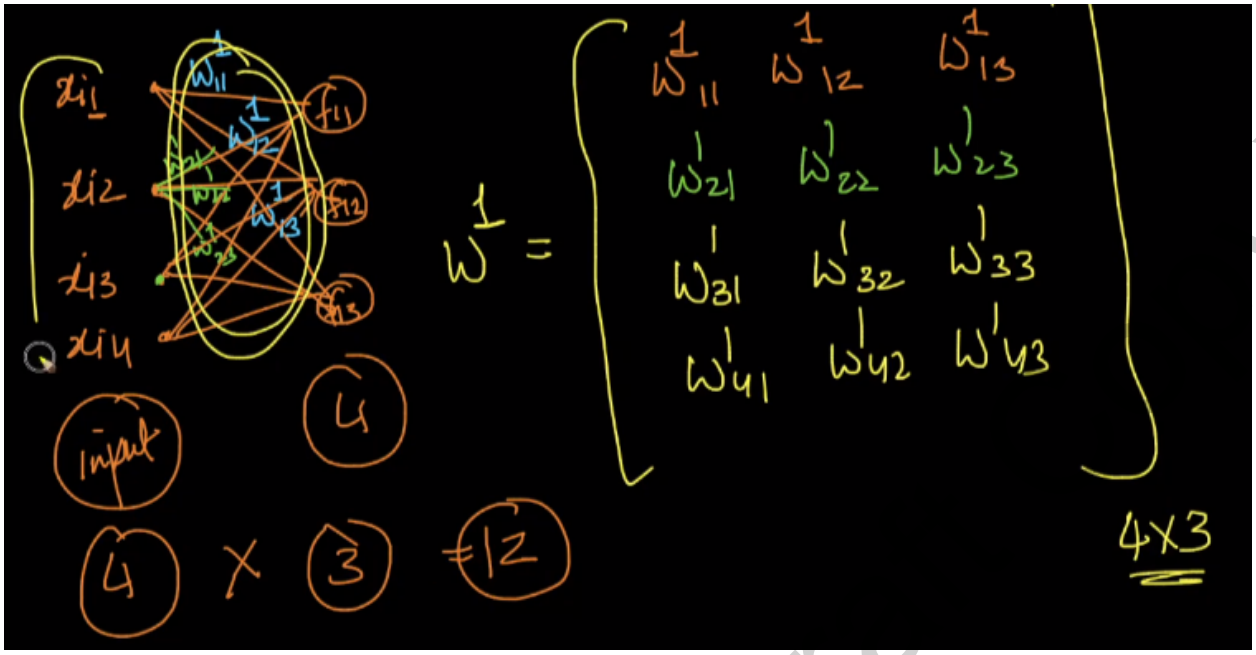
a. As a single neuron itself is powerful over all the algorithms, Interconnected neural structure will be extremely powerful.
b. Using multiple neural layers, we can solve complex mathematical computations.

57.6 Notation

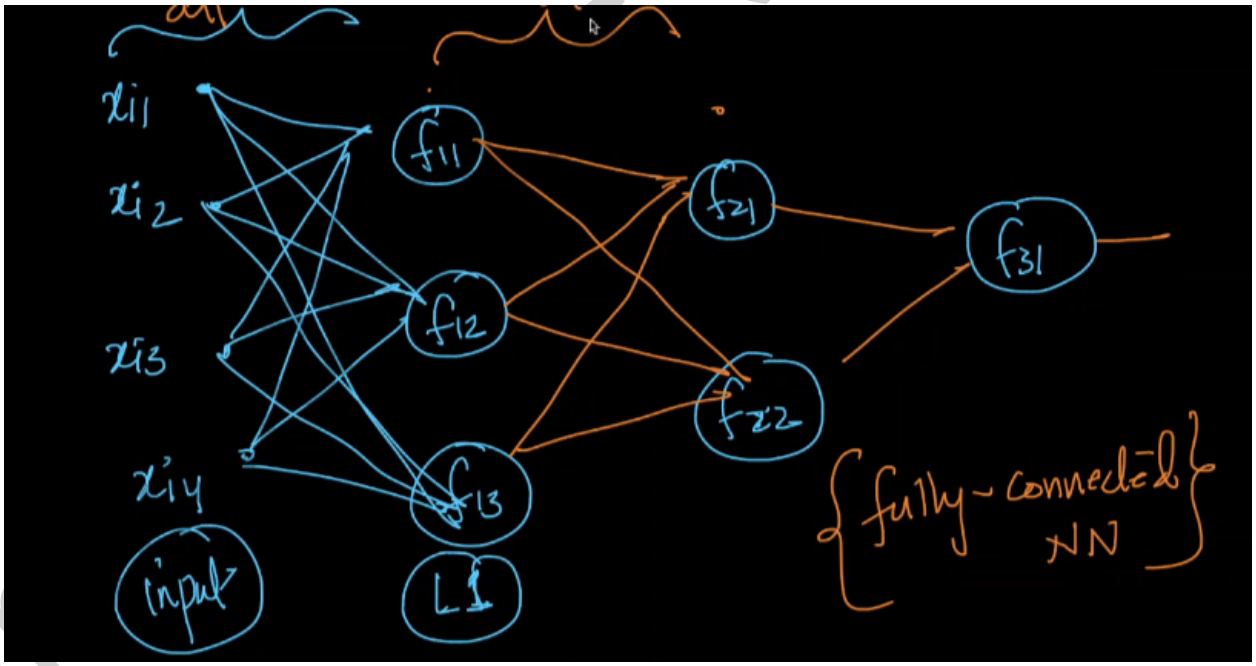
Suppose D={xi,yi} ,xi & yi belongs to 4D. And yi_hat is the output predicted by the model.



In the notation w_{ij}^{k}, i refers to the layer number, j refers to the index and k refers to the nxt layer.
For eg: f12 is the function in 1st layer in 2nd position.
O12 refers to the output from the 2nd neuron in the 1st layer..
W_{12}^{2} refers to the weight vector of the 2nd neuron in the first layer and going to Second layer.
The number of weights between layer 1 and layer 2 will be the multiplication of neurons in both layers.



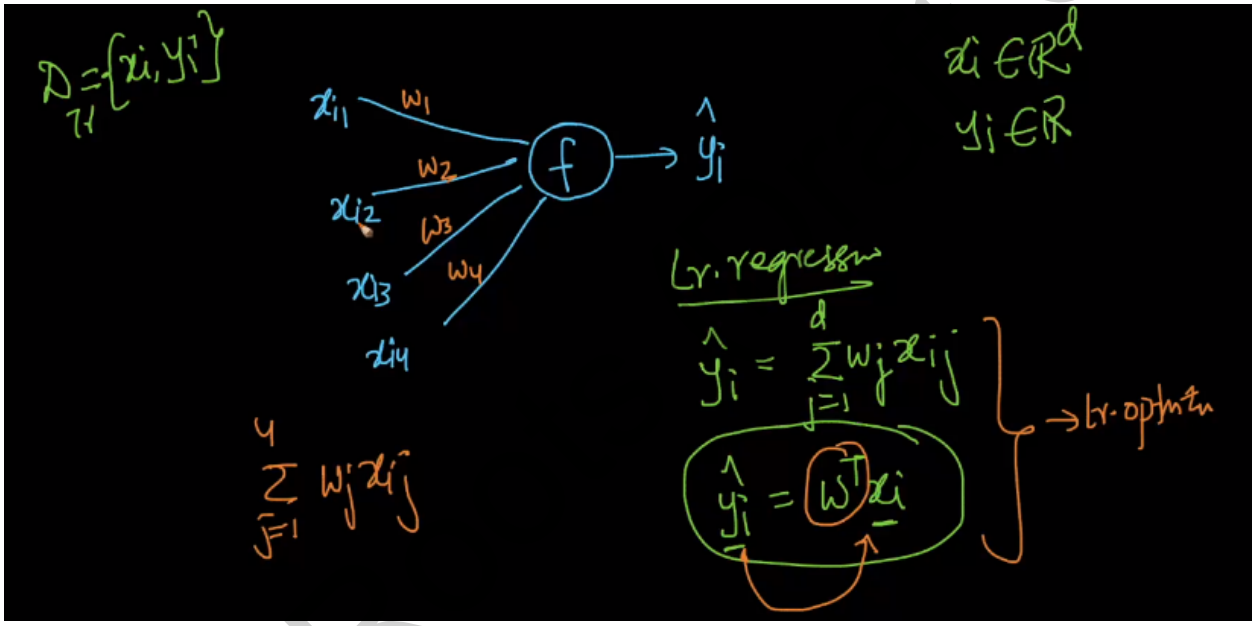
When we have connections between all hidden layers, we call it a fully connected layer.



57.7 Training a single-neuron model

The process of finding the best weights using the training data is called Training.
In single neuron models, Perceptron and Logistic regression are the examples for classification and Linear Regression is an example for regression.

Recap of Linear Regression:

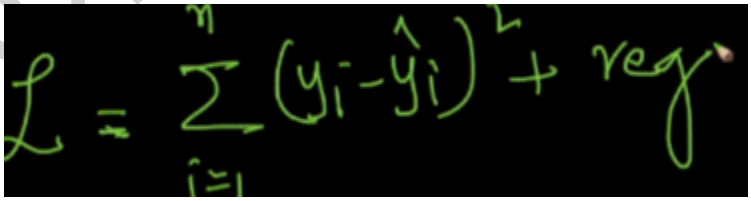


In the case of Linear Regression, the activation function is the Identity function.
f is said to be an Identity function if f(Z)=Z.

Training procedure:

1. **Define loss function**.

L=sum(yi-yi_hat)**2



2. **Optimization equation:**

Optimizing is finding the weights that minimize the loss function.

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - f(w^T x_i))^2 + reg$$

Where f is Identity function for linear regression, sigmoid for logistic regression and threshold function for perceptron.

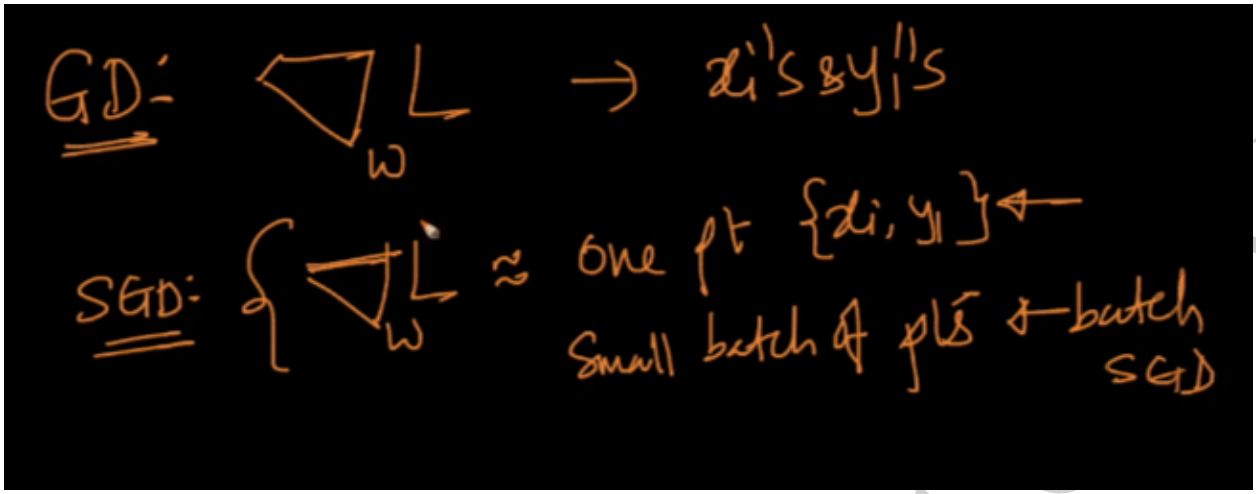
3. **Solve the optimization problem:**
   a. Initialize the weights.
   b. Compute the gradient of L wrt wi

$$\nabla_w L = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_n}\right]^T$$

   c. Calculate new weights.

$$w_{new} = w_{old} - \eta \left[\nabla_w L\right]_{w_{old}}$$

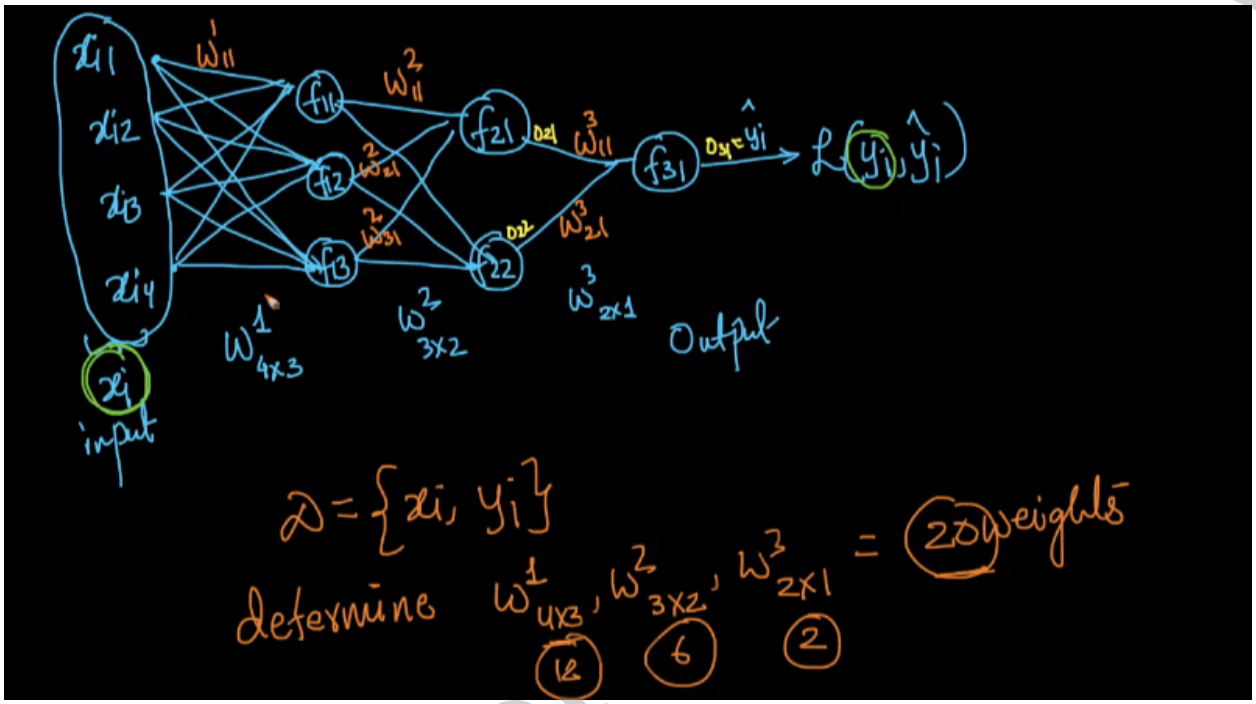
In GD, we calculate the gradient using all the points from training data, while in sgd, we only take a set of points.



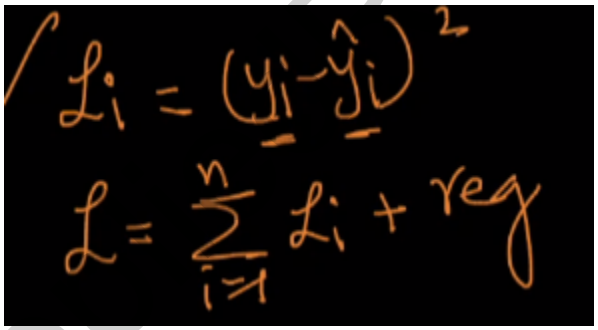
For calculating the derivatives, we use the chain rule of differentiation.

57.8 Training an MLP: Chain Rule

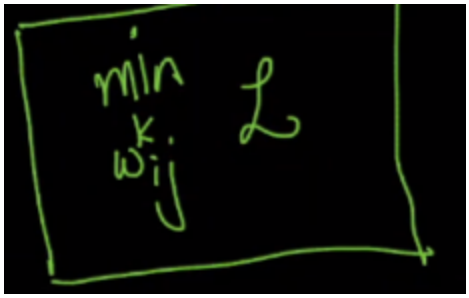
In this section, we see the training for given Multi Layer Perceptron model.



1. **Define loss function**
   The loss function we are using for linear regression is squared loss.

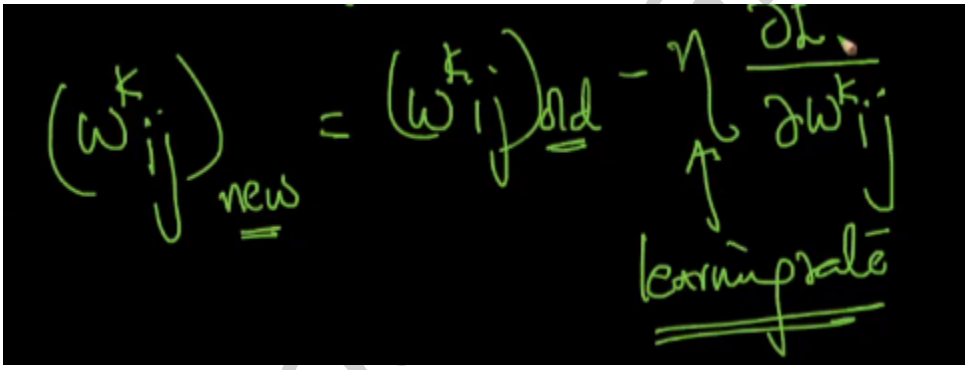


2. **Optimization equation**
   Here, the L is the addition of both squared loss and regularization. So, the optimization equation minimizes the L for all possible values of w^1, w^2, w^3



3. **SGD or GD**

   a. Initialize the weights randomly or using any of the initialization techniques. [we will discuss them in the coming sections.]
   b. Update the weights

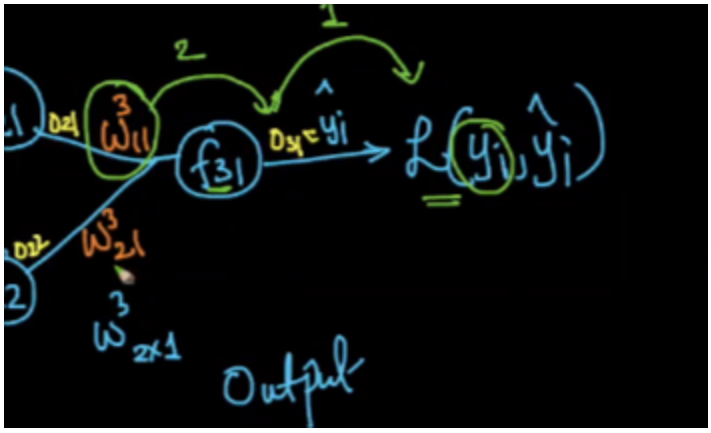


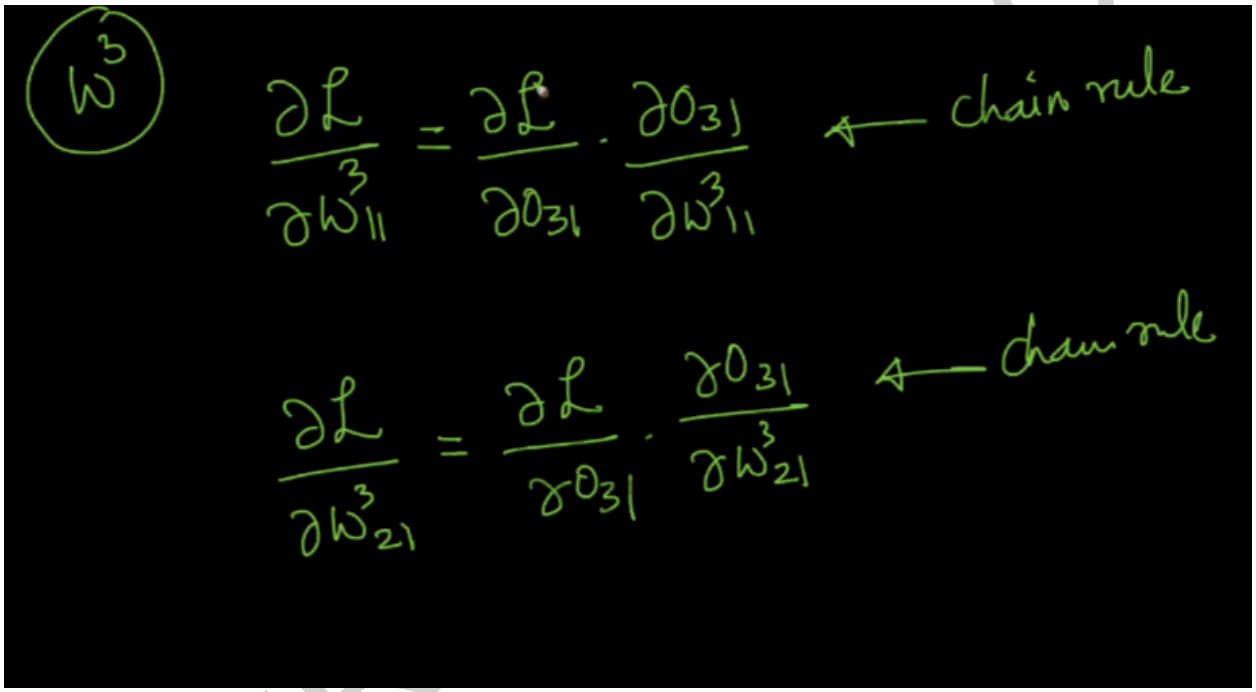
   Where eta is the learning rate.
   c. Perform the updates till convergence.

Calculating Derivatives:

dL/dw^3 :

We can calculate this using a simple chain rule.

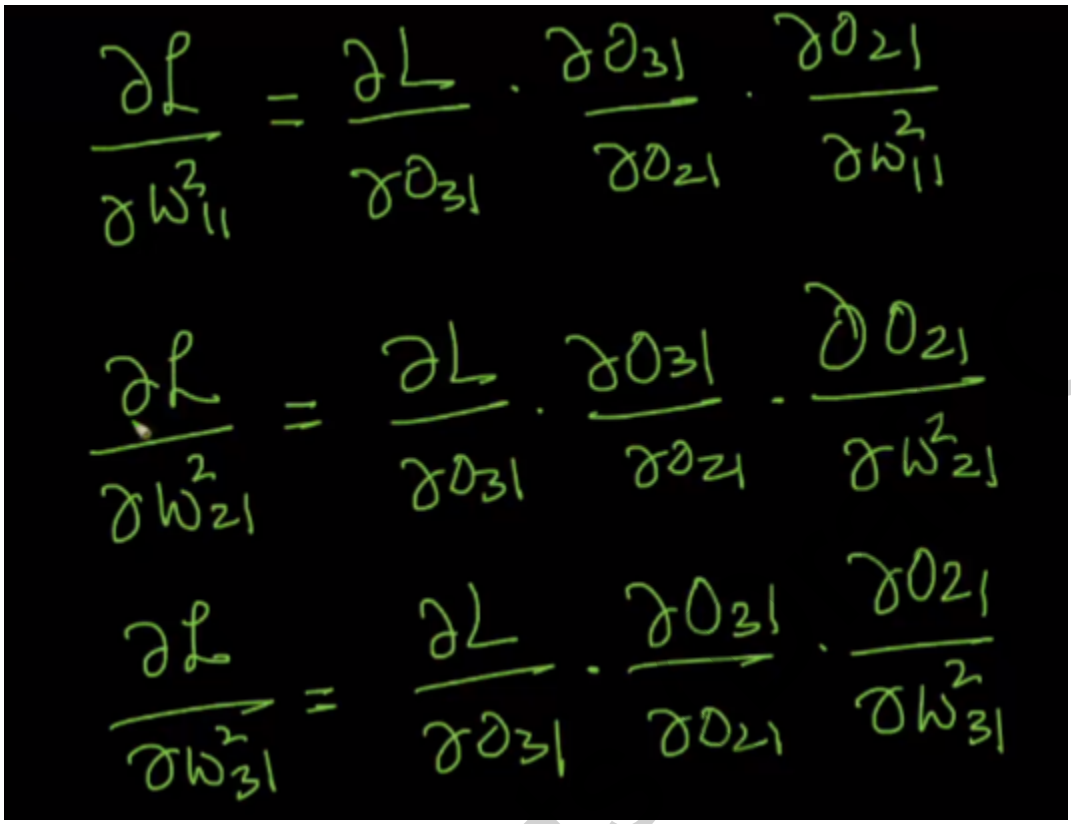




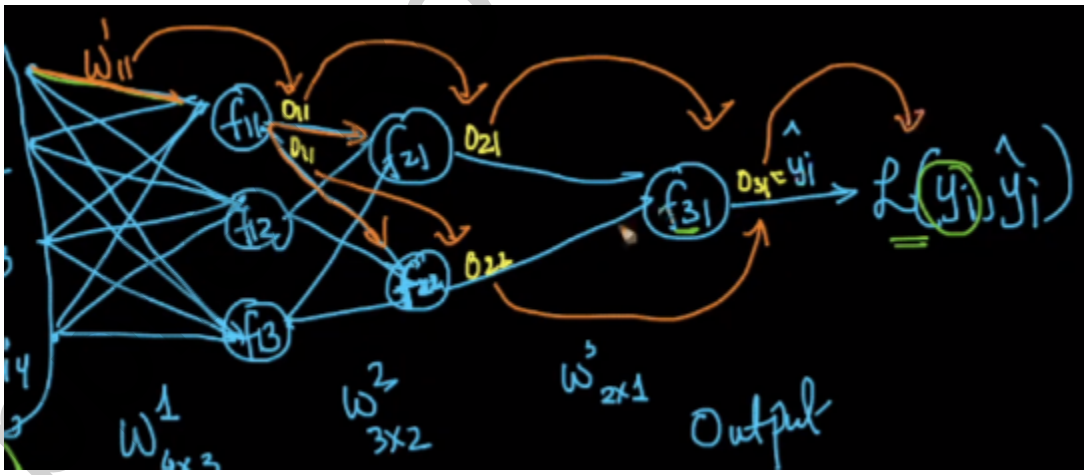
dL/dw^2:

For calculating $dL/dW_{11}^{2:}$

L is getting impacted by the O_{31} and O_{31} by O_{21} and O_{21} by W_{11}^{2}.
Similarly,

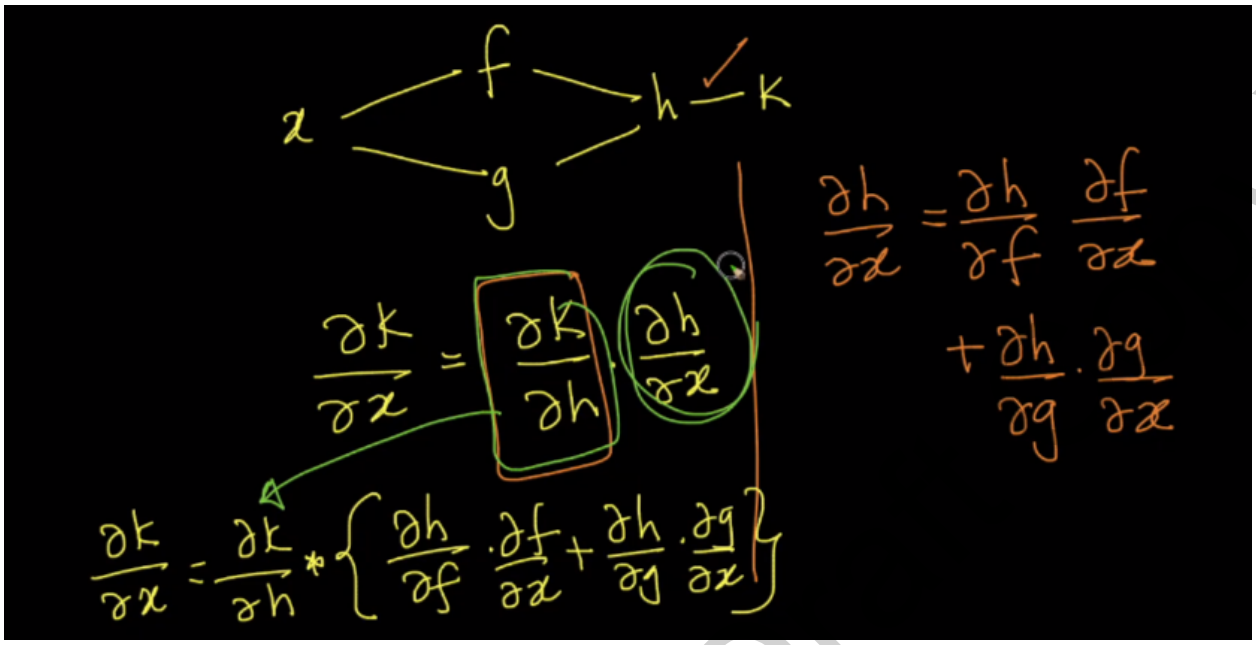


dL/dw[1]:

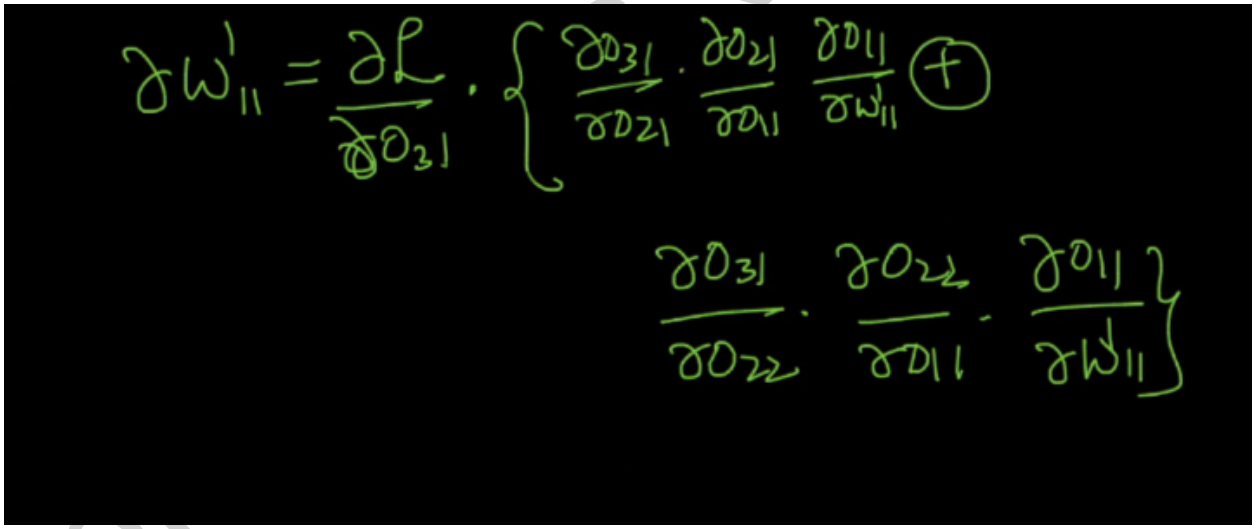


Here O_{11} impacts both O_{21} and O_{22}. So, we have two paths. This is special case of a chain rule.

In such cases, we calculate the chain rule as follows:



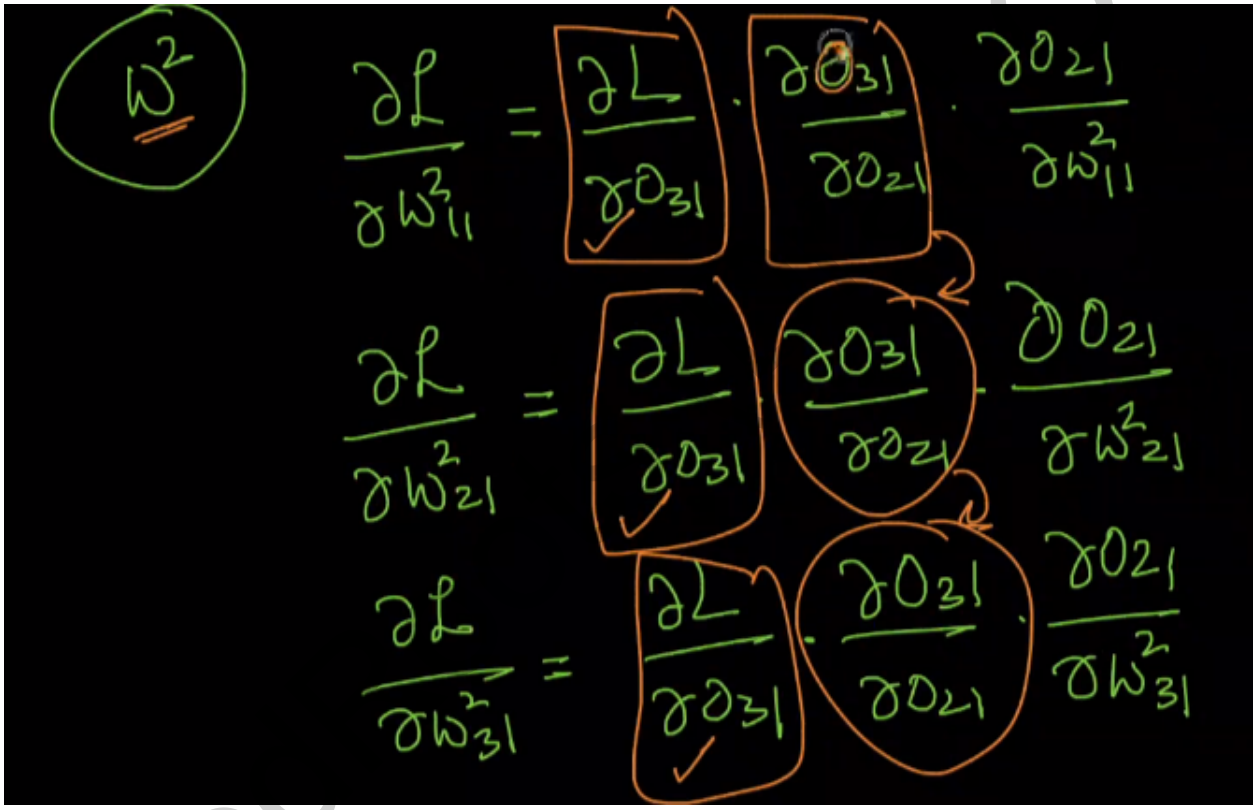
We have to apply the above example for our problem.



57.9 Training an MLP:Memoization

The core idea of memoization is, If there is any operation that is used many times, then it is a good idea to compute it once and store it and re-use it.

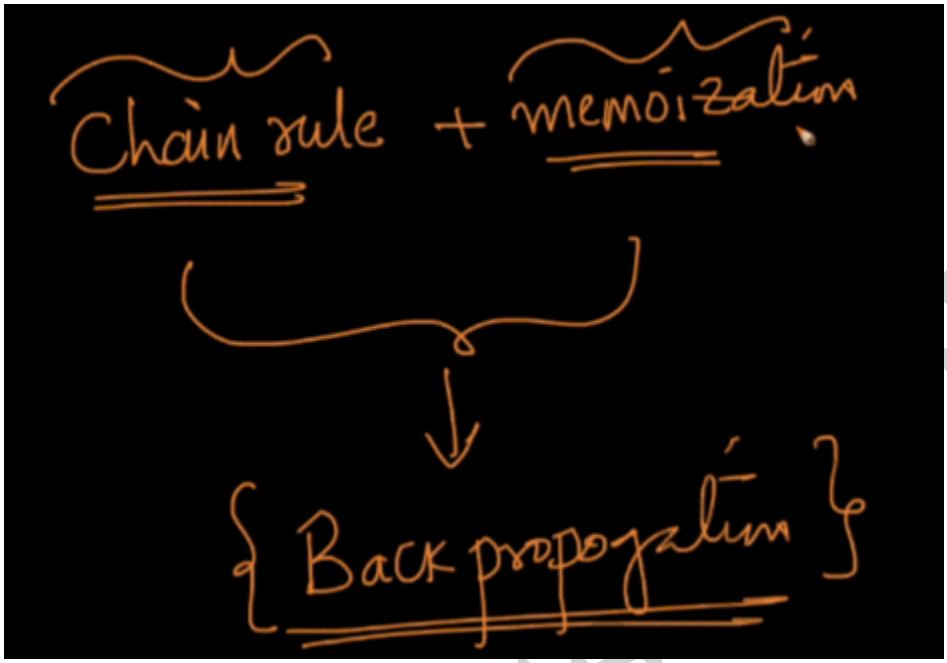
For example, while calculating dL/dW^2, We come across the following derivatives many times. So, instead of computing repeatedly, we can calculate those derivatives and store them.



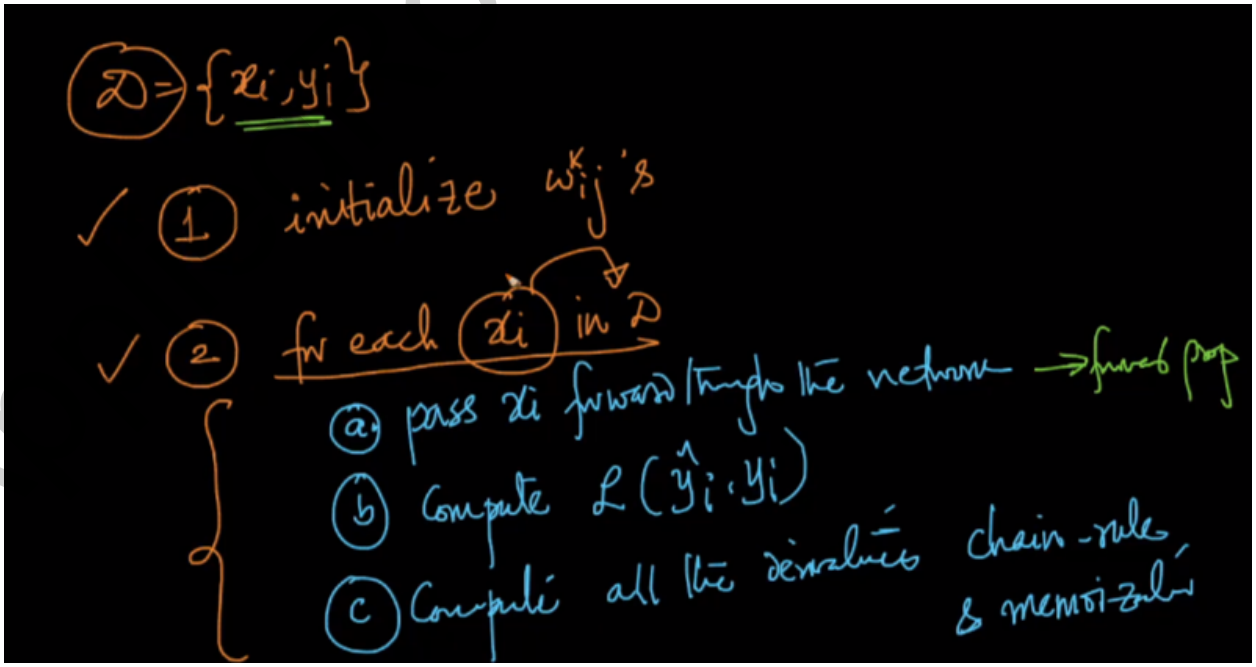
It might take slightly more memory, but we can reduce the training time by a huge margin.

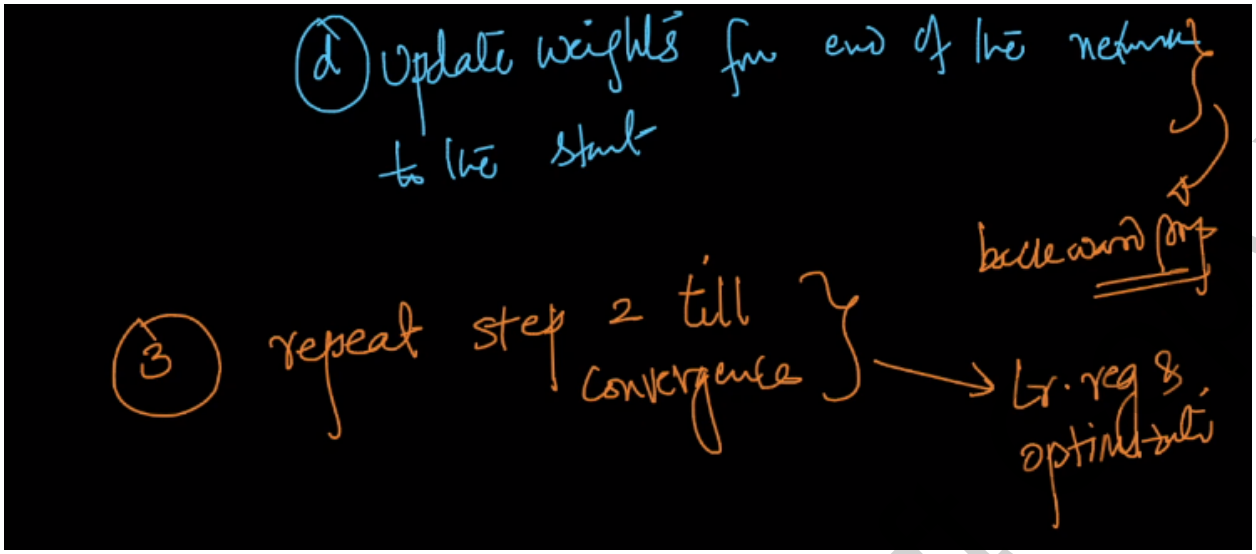
57.10 Backpropagation

Back propagation algorithm is a multi-epoch training methodology which we use to leverage chain rule and memoization to update weights.



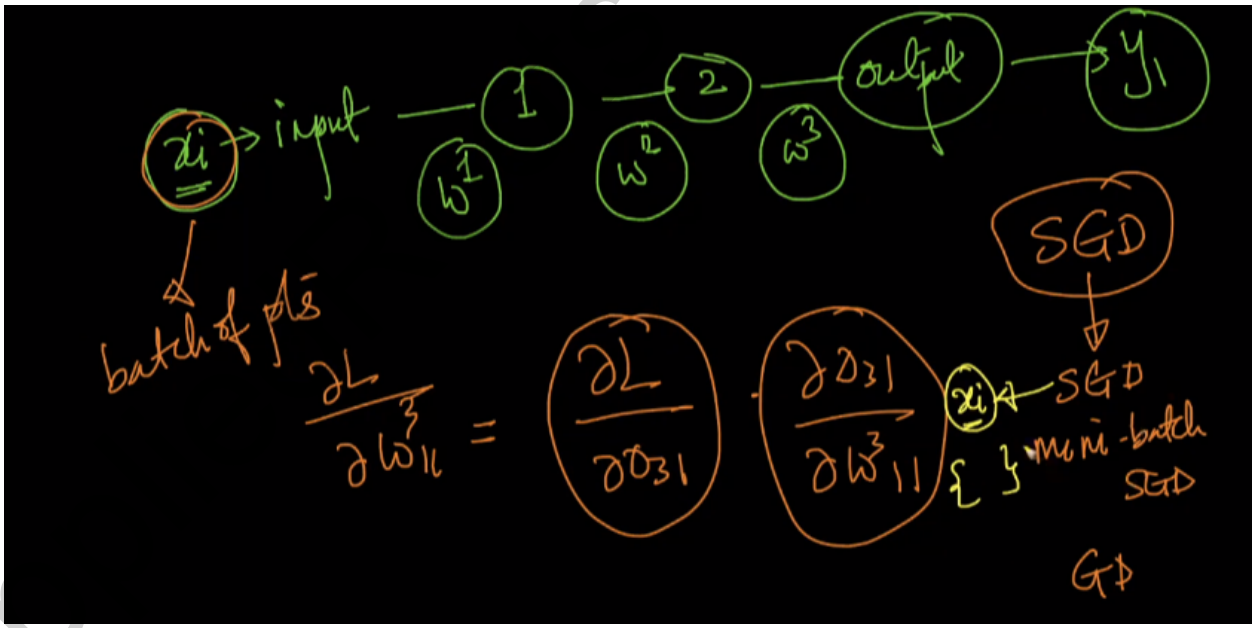
The Backpropagation algorithm is discussed below:



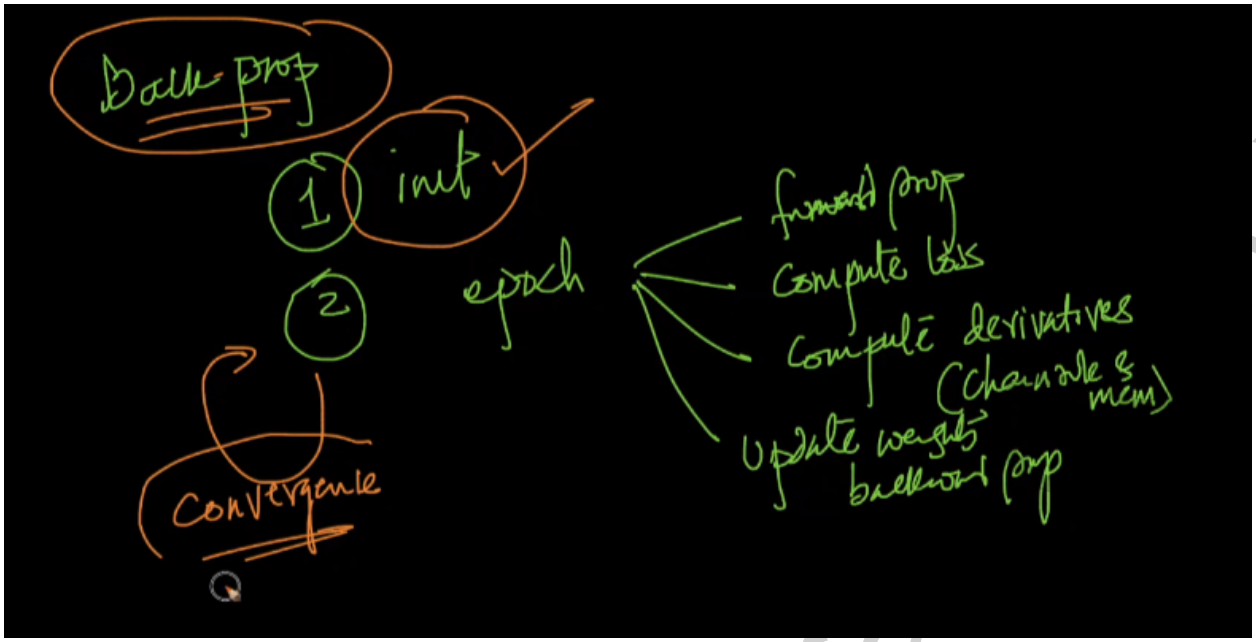


First we calculate loss using forward propagation and then we update the third layer weights W_{11}^{3} and W_{21}^{3}, then second layer weights and so on..

In mini batch sgd, instead of sending a single point, we propagate a batch of points.



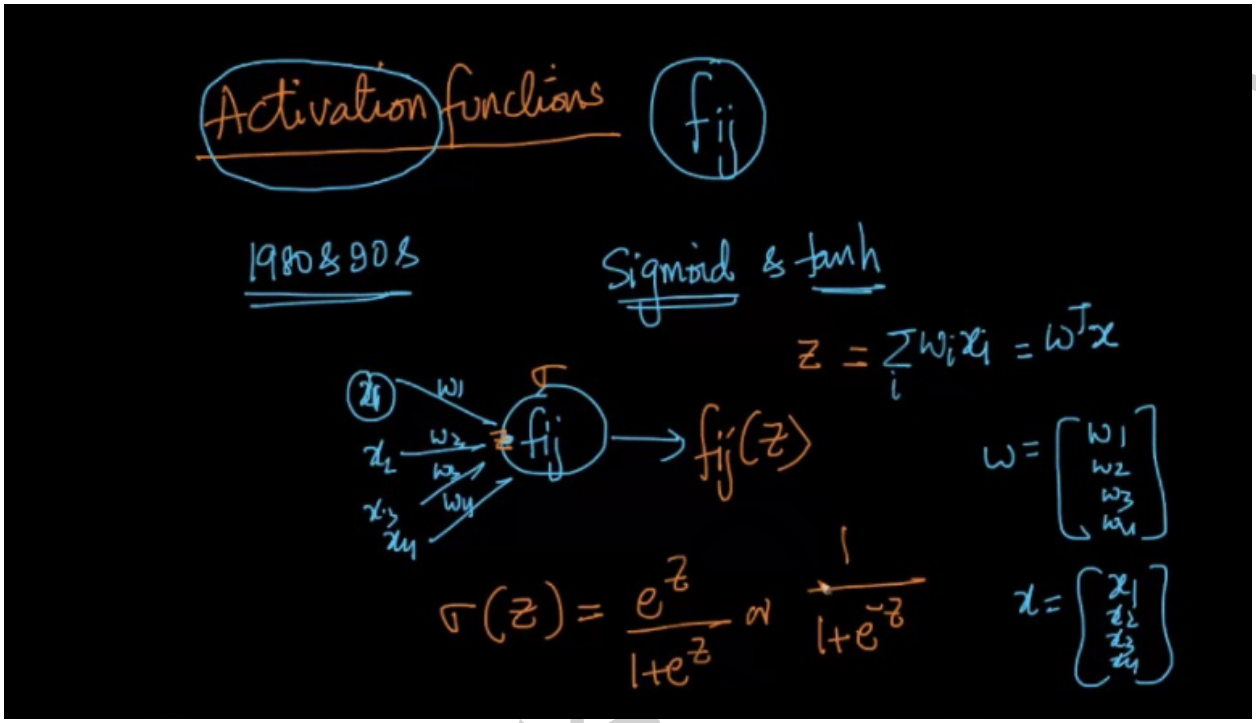
Mini Batch SGD is the more popular method of gradient descent.
Mostly the batch sizes vary in powers of 2. Like 4,16,32,64,128….



Note:
Back propagation algorithm is applicable only when the activation functions are differentiable.

57.11 Activation functions

In the 1980 and 1990's, the most widely used activation functions are sigmoid and tanh.



Timestamp : 02:26

X is a vector representing the inputs and W is the vector representing the weights corresponding to inputs.

fij is the activation function. In this case, it's a sigmoid function.

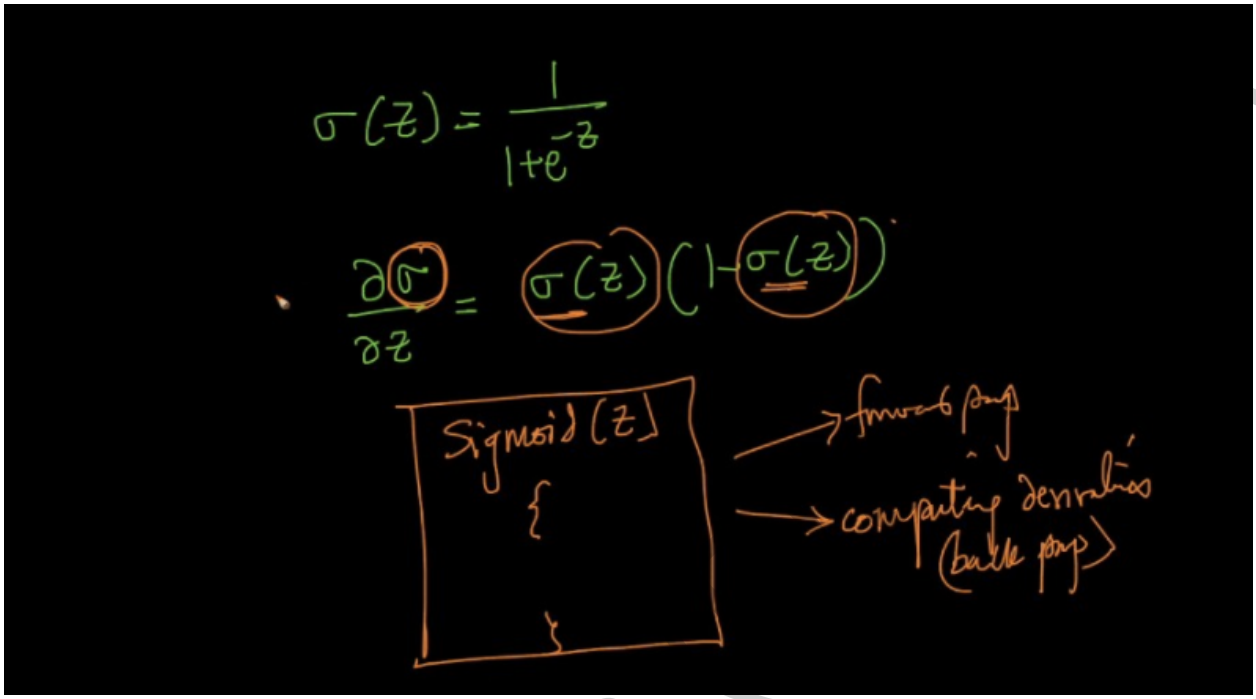
First, a dot product is computed between X and W. The resultant is passed as an argument to the sigmoid function.

In the above illustration, the equation for the sigmoid function is written.

Criteria for activation functions :

1.) It must be a differentiable function.
2.) It must be easy to differentiate. This is because we are going to implement it in our machine. So, we also care about the time complexity.

Let’s check the above condition for the sigmoid activation function.

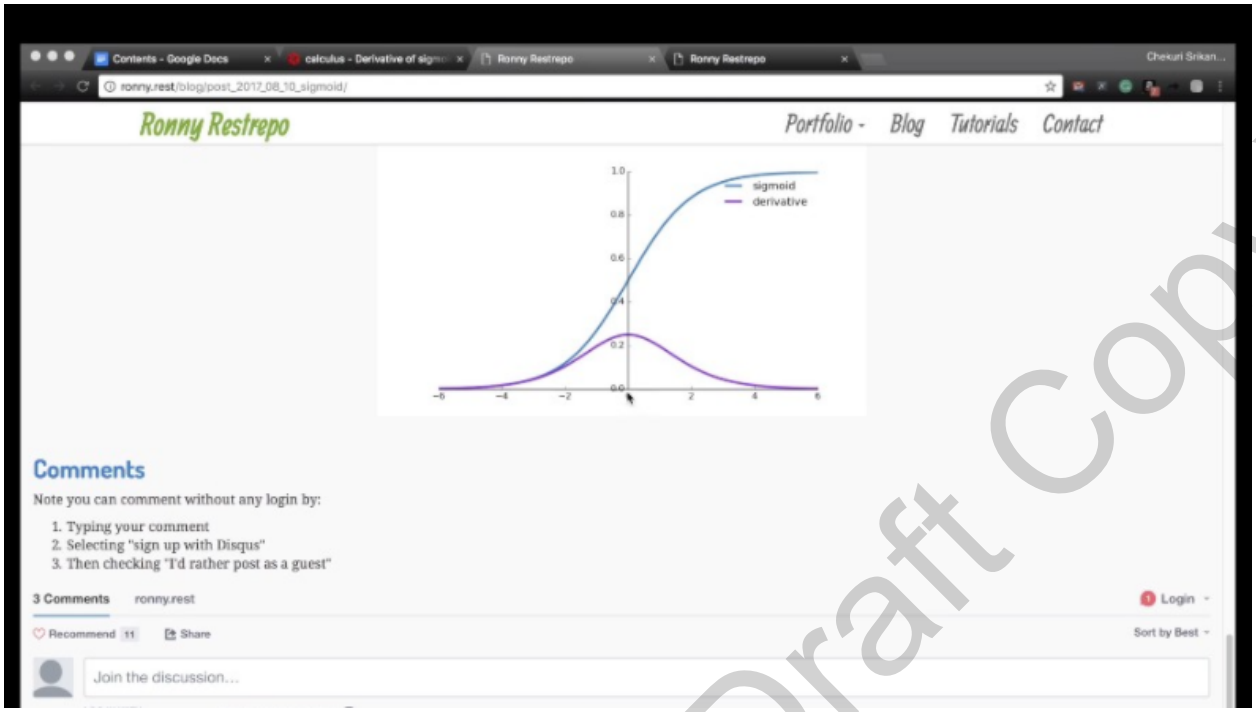


Timestamp: 06:06

The first condition says the activation function must be differentiable i.e first order differentiable. In the above figure, we can clearly say it’s differentiable and the derivative term is also written.

The second condition says it must be easier to compute. The derivative of the sigmoid function is expressed in terms of it. So, once we compute the sigmoid function in the forward propagation, we can simply save it for the backpropagation.

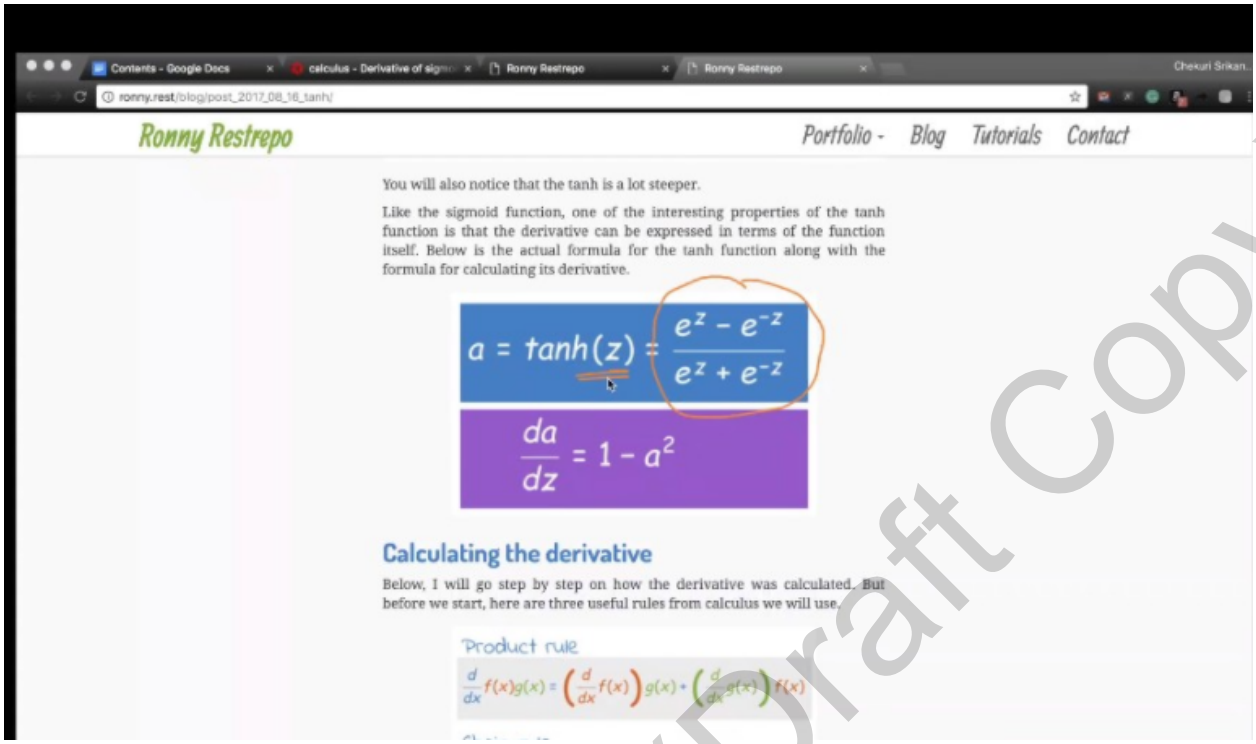
To know the derivation for the above explanation, please refer to this



Timestamp : 07:48

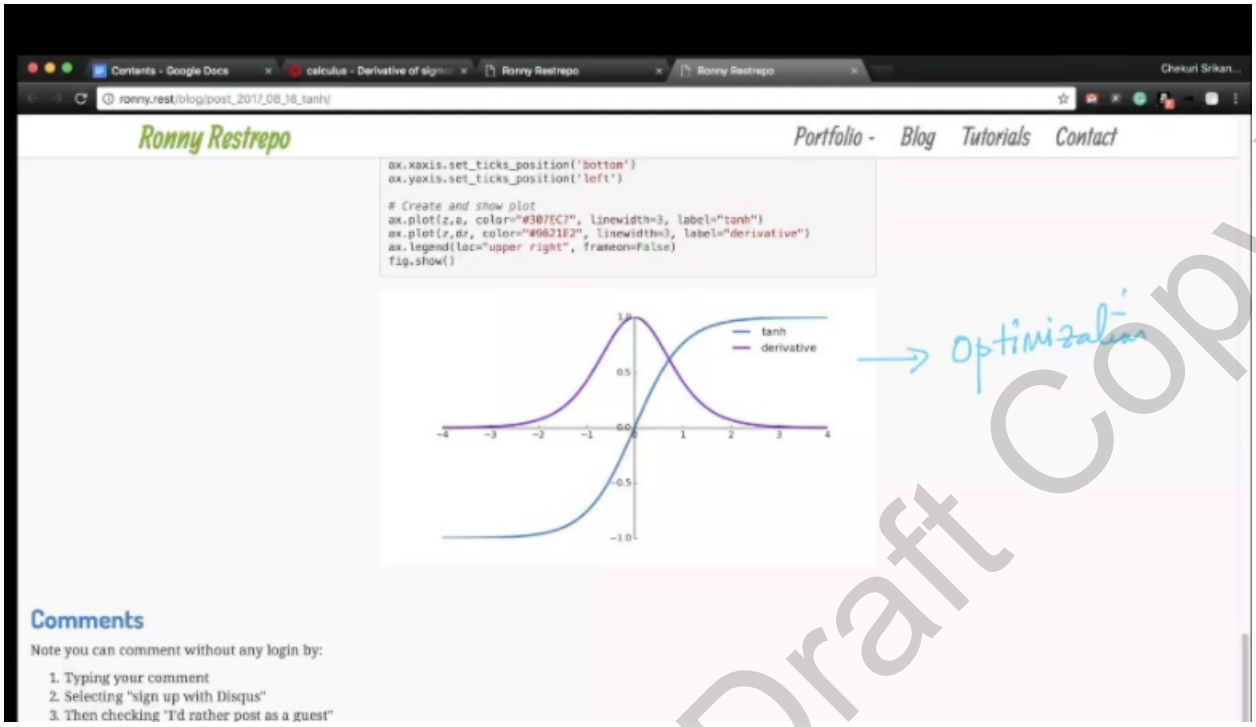
The blue curve is the sigmoid curve and the purple curve is the derivative of the sigmoid curve. We can see the sigmoid function saturates after some point. The range of derivative of the sigmoid curve is approximately from 0 to 0.3 and in other areas it flattens or close to zero. Simply, the derivative lies between 0 to 1.

Tanh activation :



Timestamp : 11:15

Here also, the derivative of the tanh function is represented by itself. So, it also satisfies the two constraints for a function to be an activation function.



Timestamp : 12:07

The range of the derivative of the tanh function is from 0 to 1. The range of the tanh function is from -1 to 1. The derivative of the tanh function saturates or flattens to zero outside the range of [-3,3].

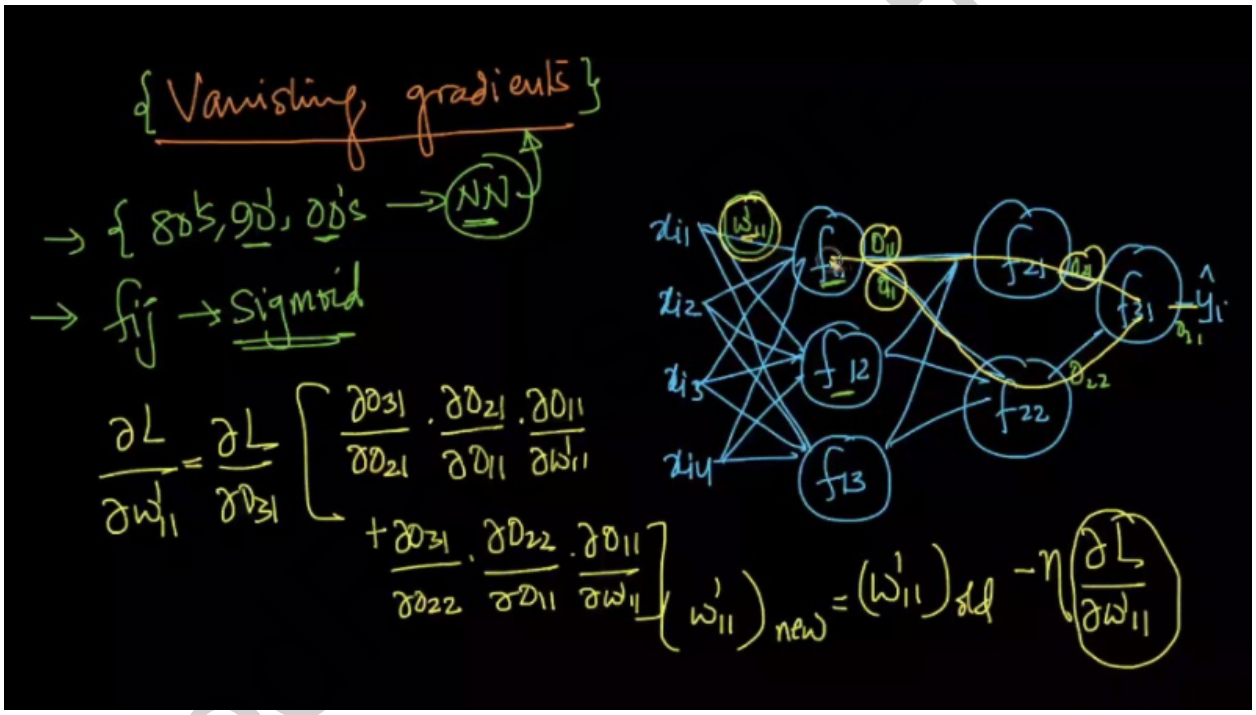
In the modern era, the most widely used activation function is ReLU and other variants of it.

57.12 Vanishing Gradient problem.

In the classical era of deep learning, one of the problems in the neural networks is vanishing gradients.

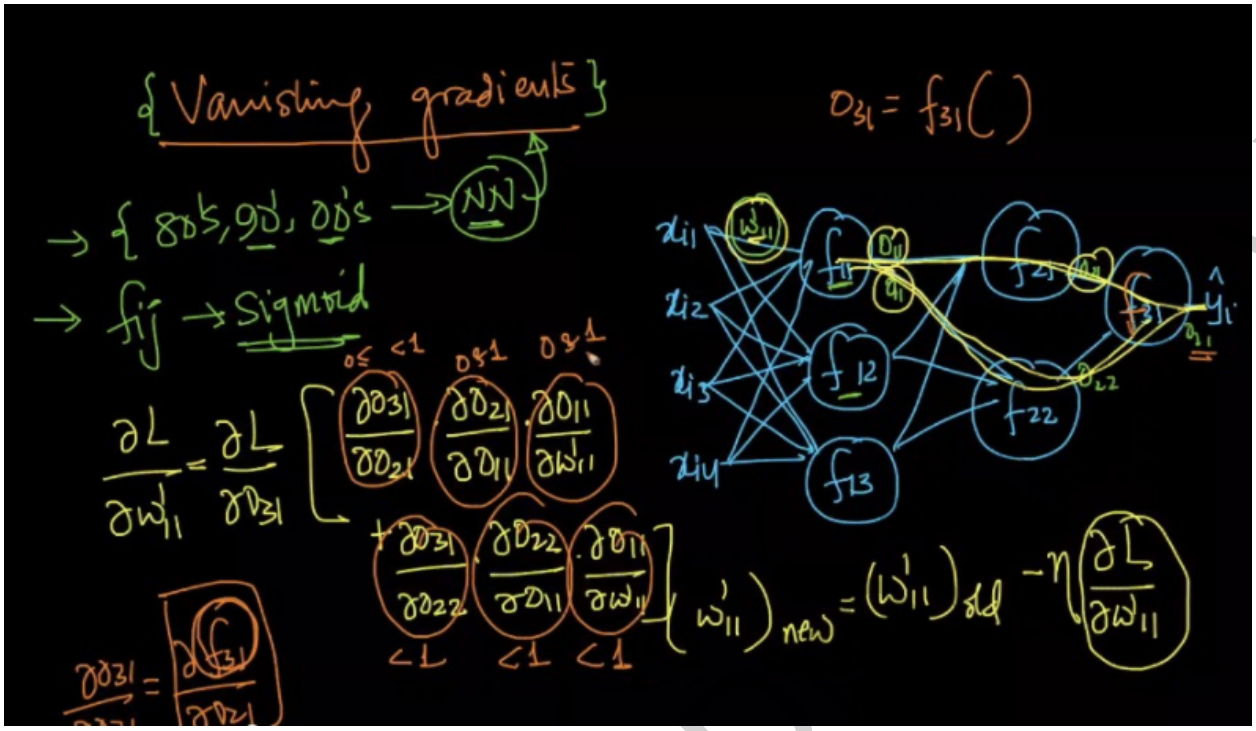
Let's understand this problem through a simple example.

Once the backpropagation process starts, we update the weights simultaneously. We need the gradient to update the parameters in the network.



Timestamp : 02:41

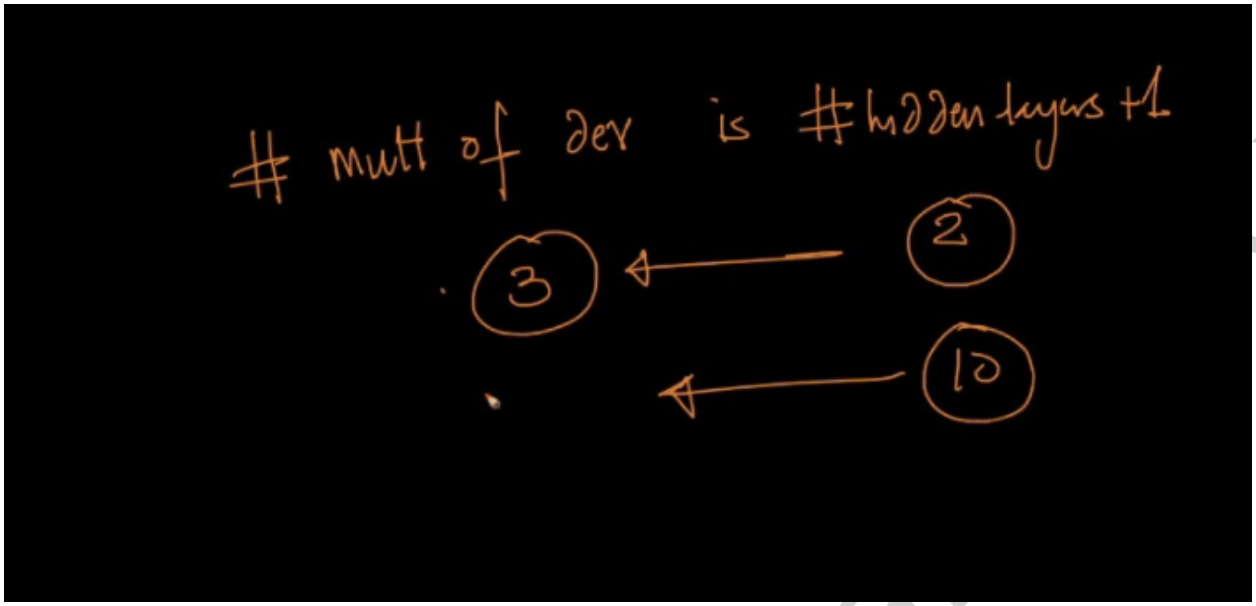
In the above illustration, we can see the update equation for the parameter w11. In the derivative terms, for now we will only concentrate on the part where we compute the derivative of the activation function which is sigmoid.



Timestamp : 04:13

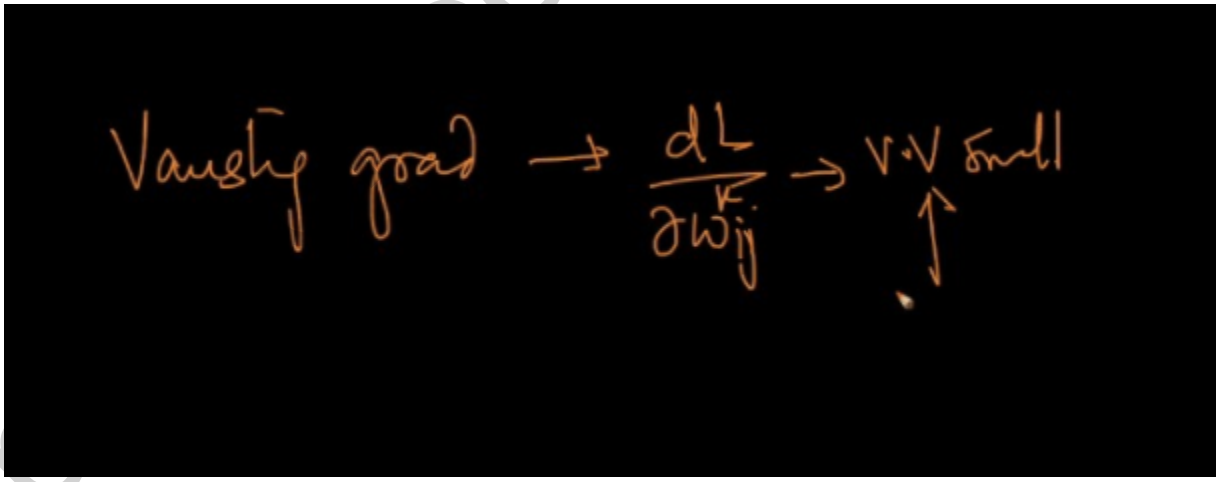
We can see from the above illustration that we are multiplying many terms which are lying in the range [0,1]. So, if we repeatedly multiply these small numbers, we will get a much smaller number which is quite negligible. This means, the final gradient term will be very small. So, the update won't happen much since the gradients are small.

Number of multiplications of the derivative term is nothing but the number of hidden layers with one added.

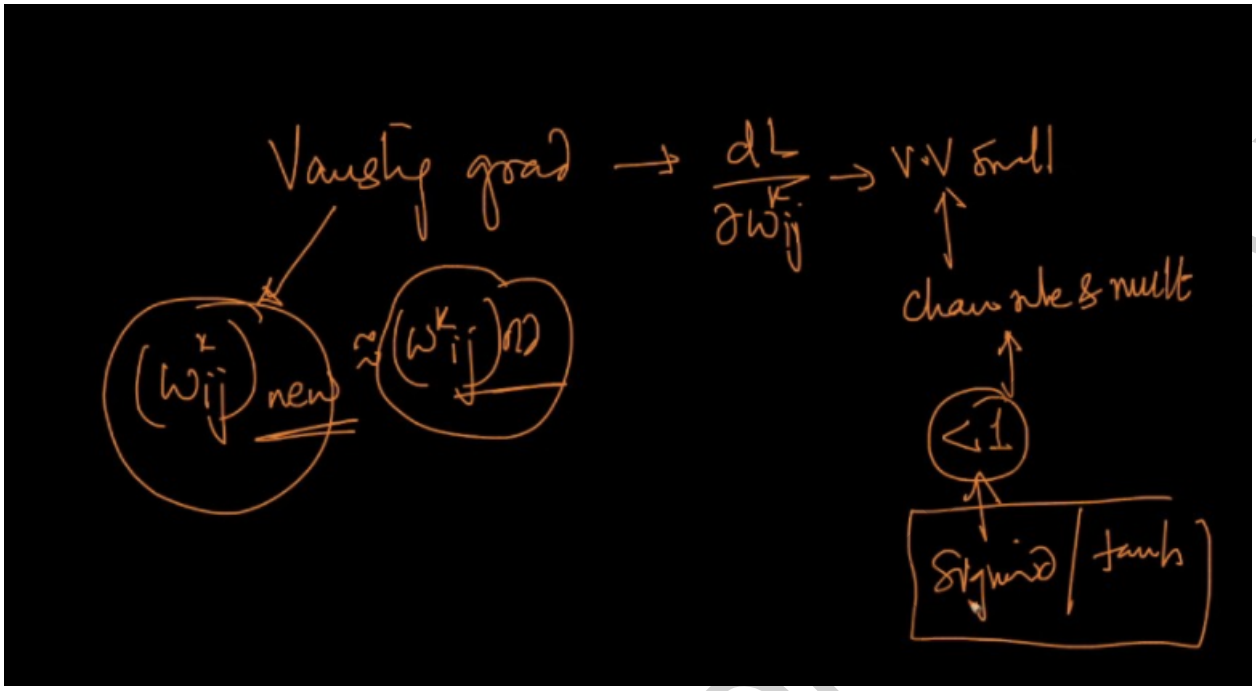


Timestamp : 09:57

So, as the number of hidden layers increases, the terms also increase which means the final resultant is going to be smaller. This makes the gradient term to be very small. So, the magnitude of the update is less or the learning doesn't happen significantly since we are unable to update the parameters much. This problem is called vanishing gradient.



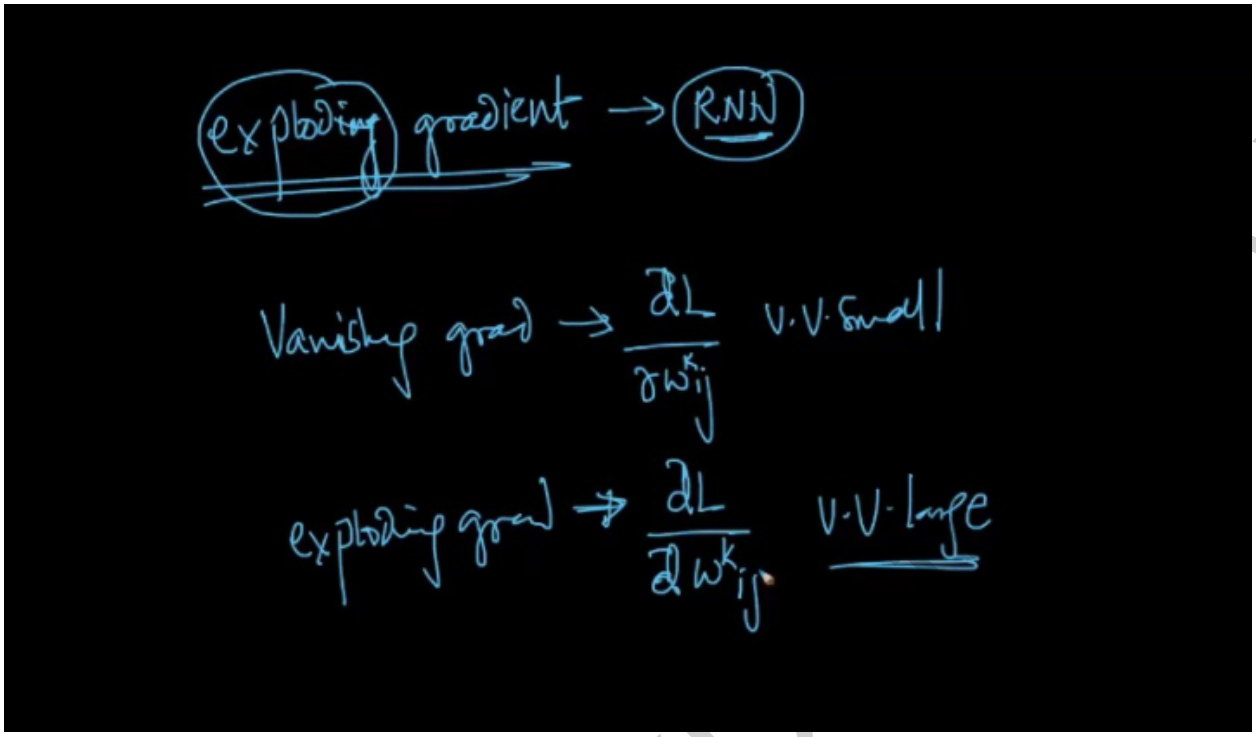
Timestamp : 11:54



Timestamp : 12:39

If the gradient is very small, then the old value of the parameter is approximately equal to the new value.

This vanishing gradient problem occurs in sigmoid and tanh.

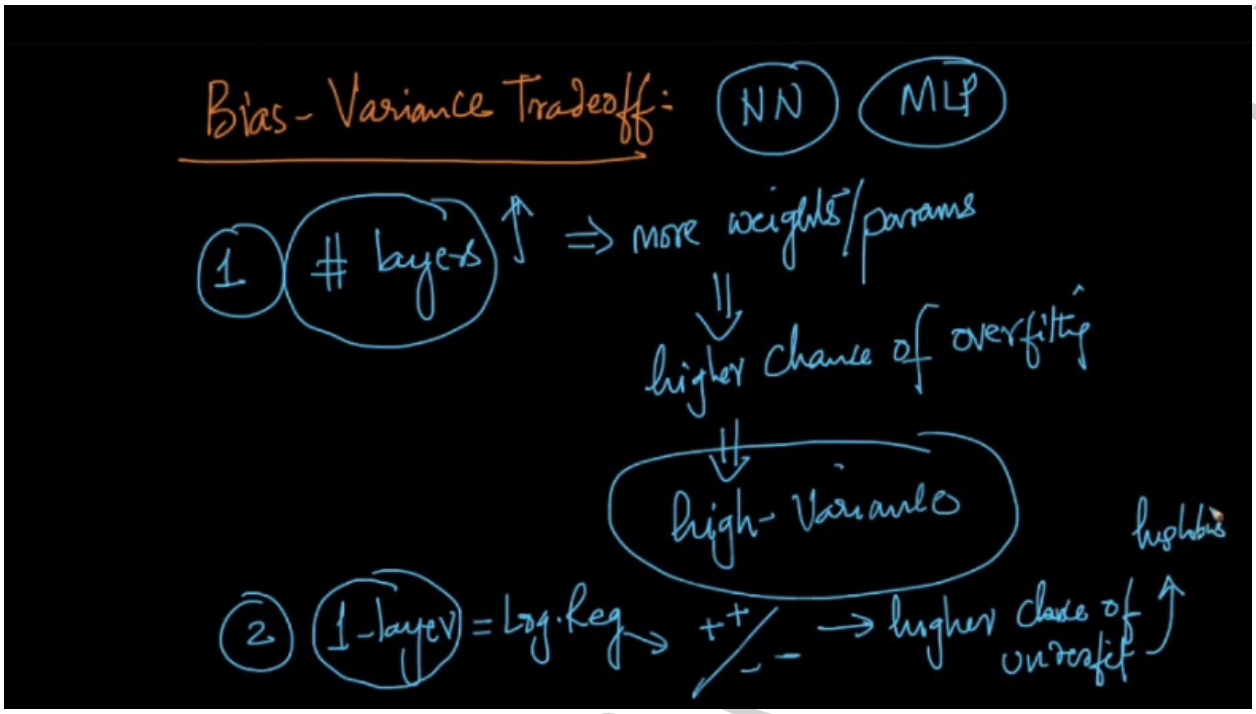


Timestamp : 15:37

There's another problem called the exploding gradient which is exactly opposite to the vanishing gradient problem. Here, the gradients are very large as opposed to small in the previous case.

Due to this, the convergence is affected. The problem of exploding gradients occurs when we are multiplying the terms which are greater than 1. This problem occurs mostly in RNN.

57.13 Bias-Variance tradeoff.

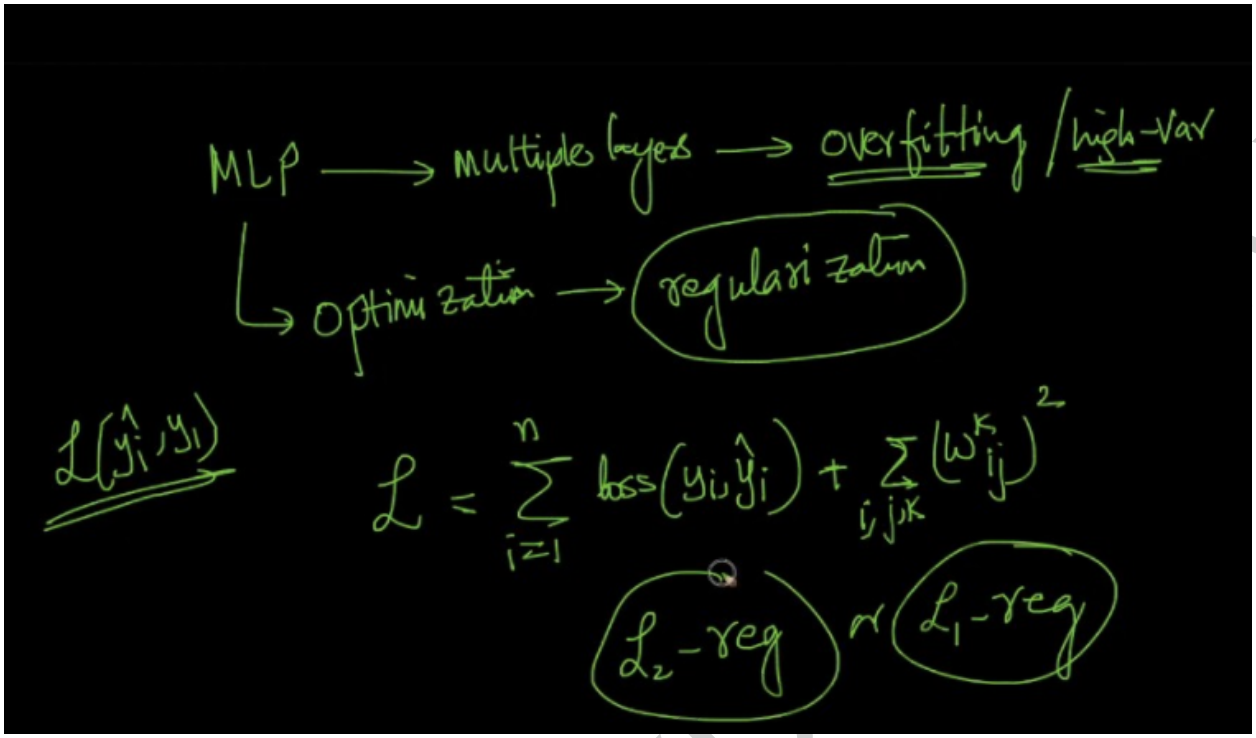


Timestamp : 01:44

1.) As the number of layers increases, we have more parameters. So, our model will become complex. This leads to a higher chance of overfitting. For example, if the function is very simple and we use a very complex model for it essentially we are overfitting. Overfitting means also the model has high variance.
2.) If we have less number of layers, we are underfitting. This is because we can't learn a complex function with few parameters. This means the model has high bias.

Typically in MLP's we have the problem of overfitting/high variance.

To solve this problem, we simply add regularization to the cost function or the loss function. For example, we can add the L2 regularization term to the cost function or loss function.



Timestamp : 05:36

So, we multiply the regularization term by a constant lambda. This lambda parameter controls the bias-variance tradeoff. Here, if we have a larger value for lambda, then the overfitting is reduced. We have discussed the specifics of these regularization techniques in one of our course videos previously.

L1 regularization creates sparsity.

Note: As the video lecture 57.14 is associated with the visualization of the classification by varying various parameters, we aren't providing any notes for it. We suggest you go through the website and try different visualizations by varying the parameters.
For any queries, you can post them in the comments section below the video lecture, (or) you can mail us at mentors.diploma@appliedroots.com

The link to the playground can be found **here**