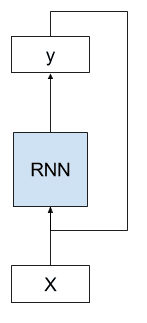
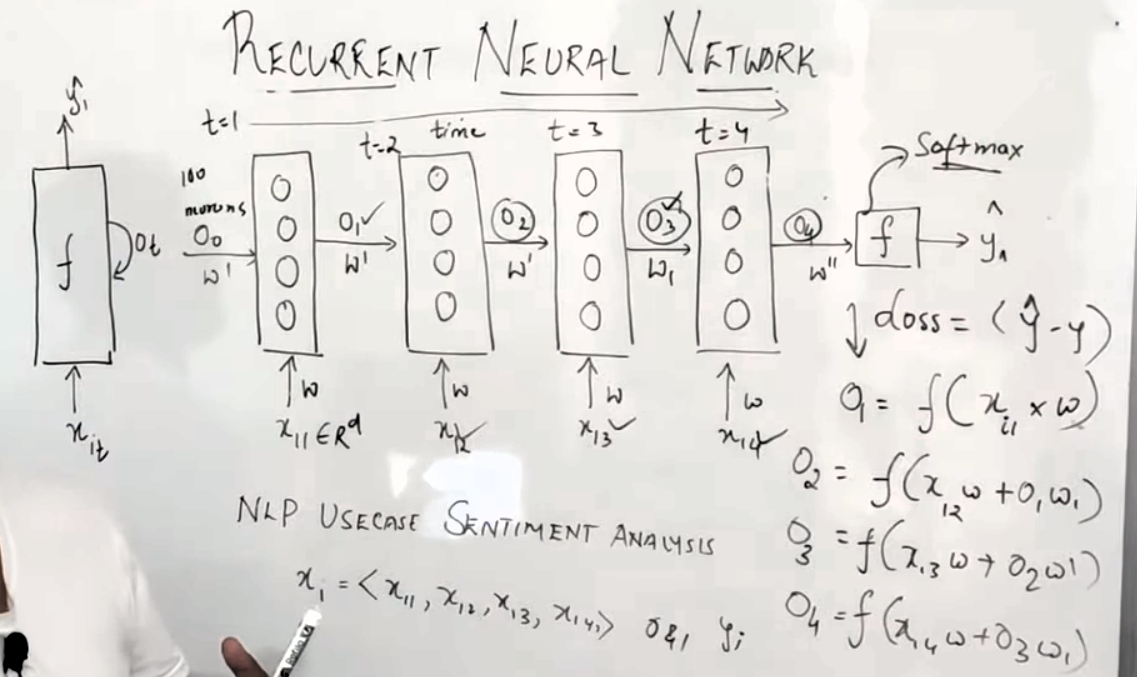
**LONG SHORT TERM MEMORY (LSTM)**

**1. Recurrent Neural Networks:**

Recurrent Neural Network (RNN) is a kind of neural network having loops that is the output of previous time step act as an input for the next time step or cycles.The general view of RNN is shown below;

****

In the above diagram, one can clearly observe that any initially input X is fed to the model and it gives any output Y. The output Y is again fed to the model with a new input X. Let me elaborate it more, look at the diagram;

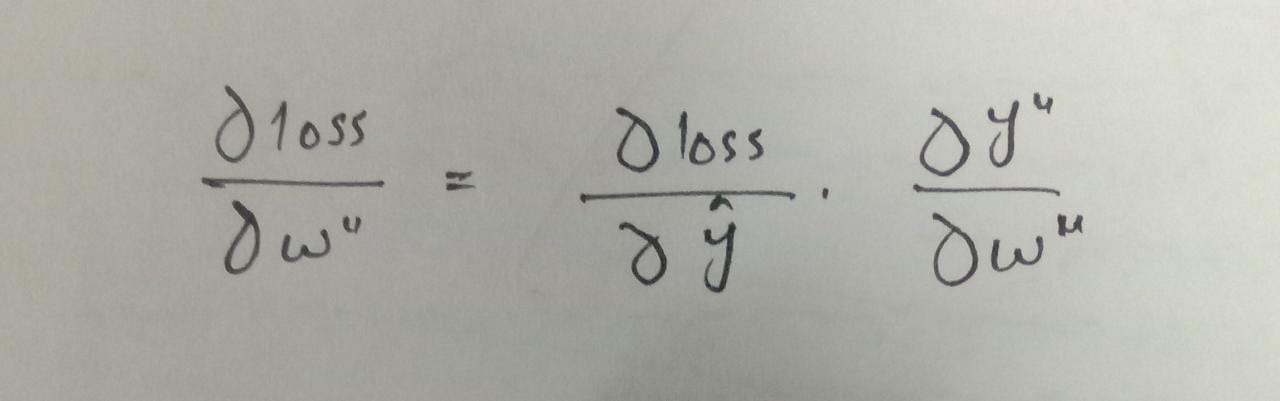


Now suppose we have unfolded the model presented in the above mentioned figure. Let we have a sentiment analysis problem. We provided a sentence x1 consist of words x11,x12,x13,x14, the sentence is sequence of words means that the meaning of next word depends upon the meaning of previous word. We provide first input x1 at time t1 to the layer having number of neurons. It is multiplied with some wight w and with the previous output which is initially none. At this time step some output o1 is generated which acts as an input for the next layer. Now the second word x12 is supplied as an input at time t2 to the second layer. This input is multiplied with same weight as was in the previous layer i.e w plus the product output o1 generated by previous layer and the same weight w1. This layer generates a new output o2 and process continues same as in last two layers. After fedding the last word x14 and output o4 is generated. Now activation function comes into picture, in our case it is softmax. The activate fucntion gives the final out y (predicted) which is compared with the actual value and a loss is calculated as shown in the above figure.

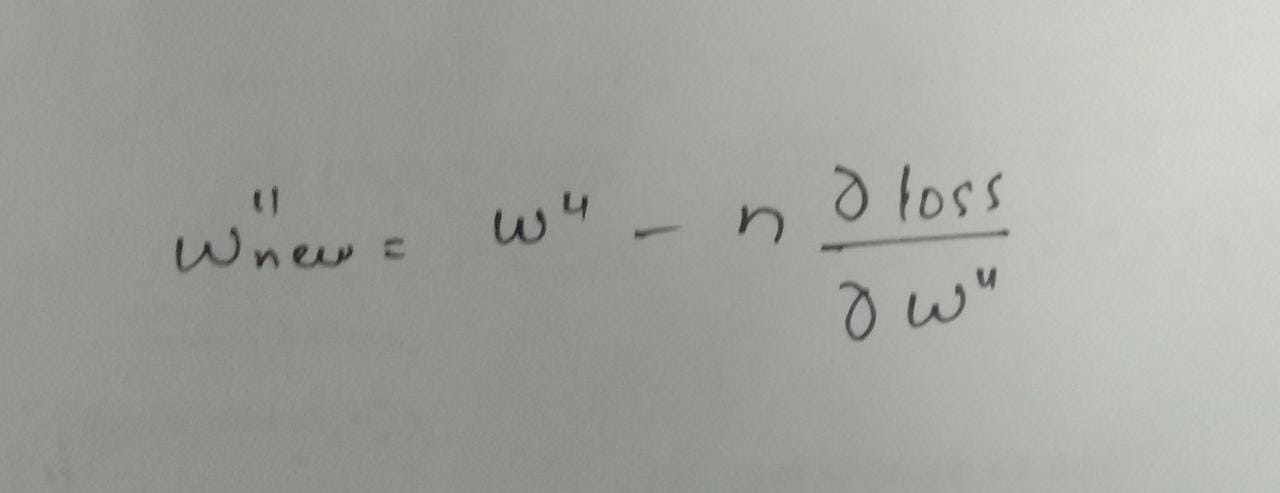
This was the Forward Propagation of RNNs.

At the end of forward propagation we calculated loss, to reduce the loss as much as possible back propagation takes place in RNNs. The loss is reduced by adjusting the wights. These weights are adjusted using the gradient descent (optimization function) which calculates the global minima.

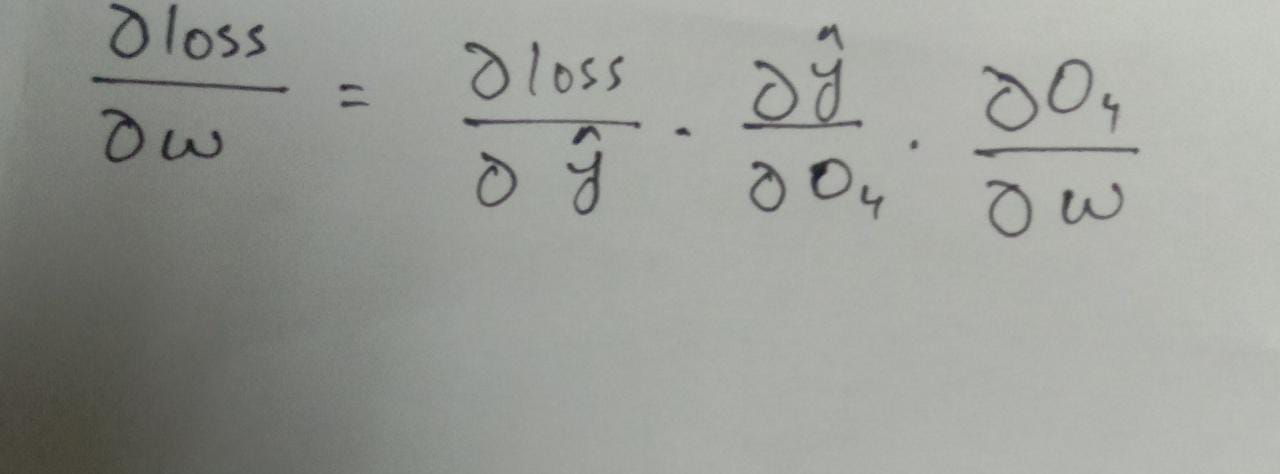
To optimize all the weights, we take the derivative of loss w.r.t to the weights using Chain Rule. To update weight (w’’), we take the derivative of loss w.r.t w’’.



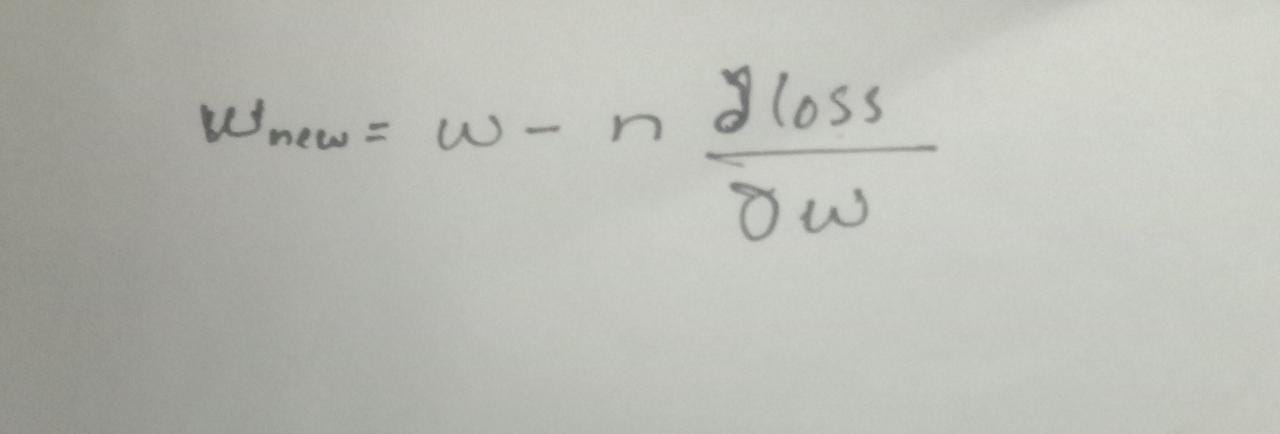
To update the weight, we subtract the product of learning rate (n) and the above taken derivative from the previous weight.



In the same way we update weight (w). Just see the below figure;

****

Here first we calculate the derivative of loss w.r.t weight (w) using Chain Rule. The loss depends upon the predicted value, the predicted value relies on output the forth layer i.e o4 while the output o4 itself depends on the weight (w). Now we the value of weight (w);



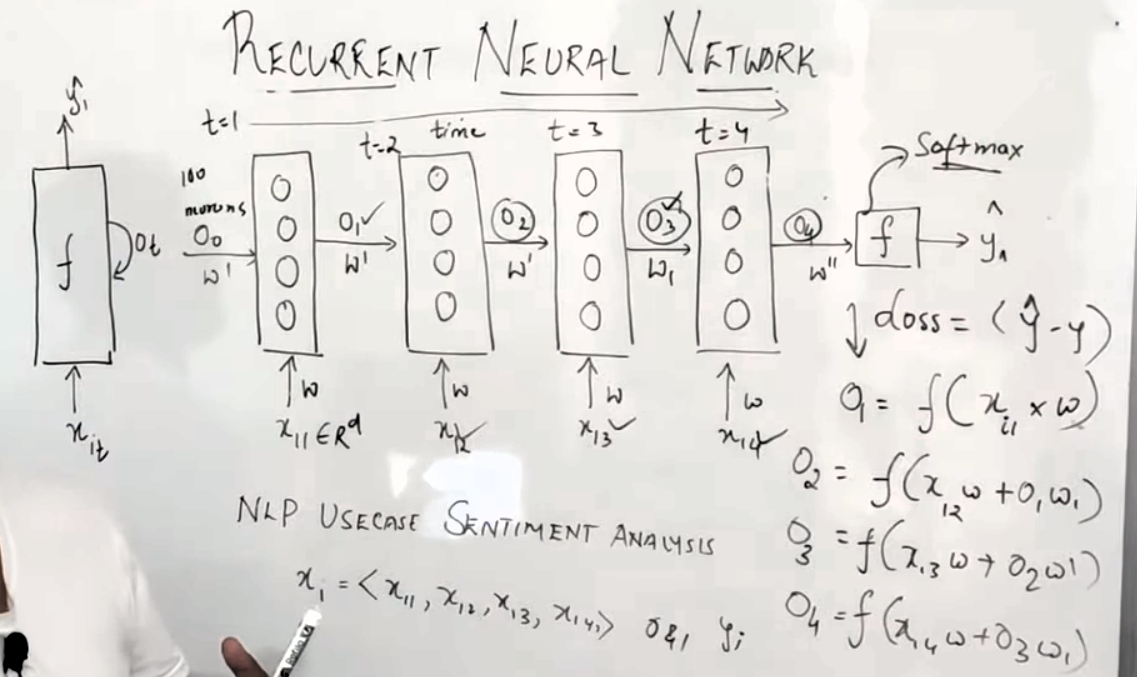
Here n is the learning rate which is defined during implementation.

**2. Vanishing Gradient Problem:**

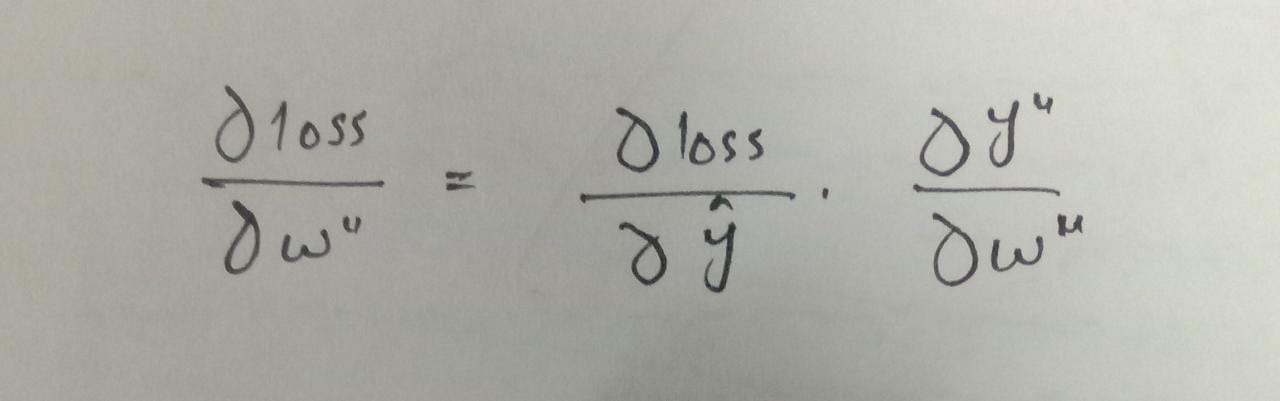
Before going further, just remember **Sigmoid function** is responsible for vanishing gradient problem. And the derivative of Sigmoid funciton is always between **0** and **0.25.**

The problem occur during back propagation.

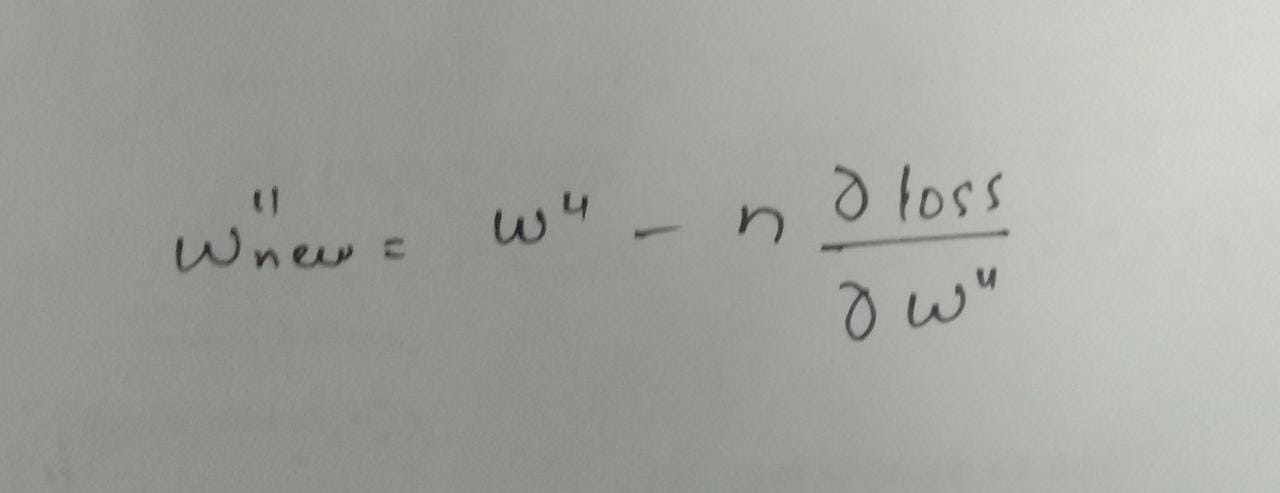
We know that the sigmoid function limits the values between 0 and 1.



In this case, suppose we use a sigmoid function as an activation function in place of softmax function and during back propagation we have to adjust weights and for that we use chain rule. Now if we are to adjust the weight (w’’), it would be calculated like;

****

Since we are using sigmoid function as an activation function, so its value would be something between 0 to 0.25. Let suppose here it is 0.10. After calculating derivative of a loss w.r.t a particular weight, we multiply it with a learning rate and subtract it from the previous value of the same weight.

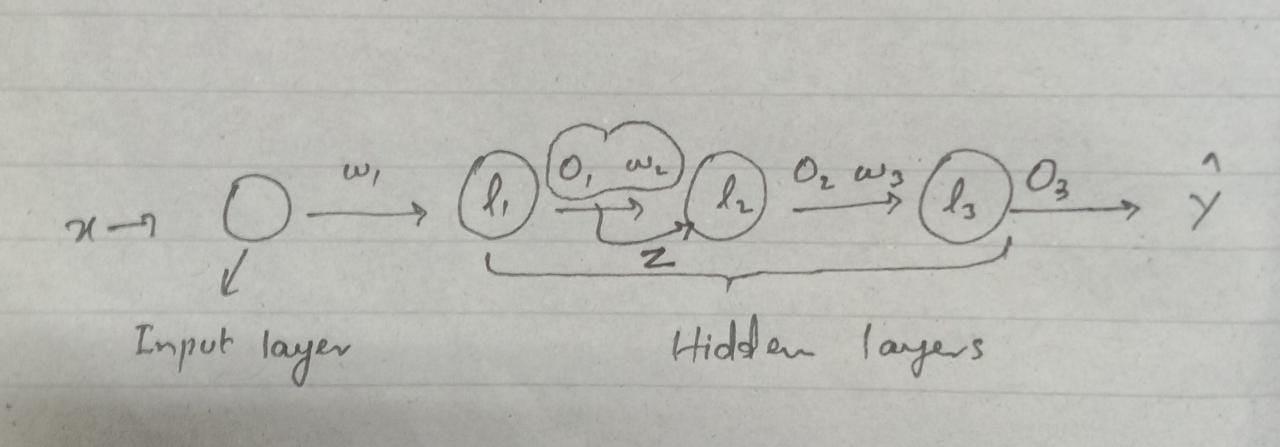


Here, we take n=1 which in practical mostly less than 1. If we had the value w’’=2.5 initially, then now after calculation will become 2.4. Even though here we can observe the difference but if we go for more layers and if in a network the number of layers are more, then this value becomes very very low such that (w’’new) approx. equal to (w’’old).

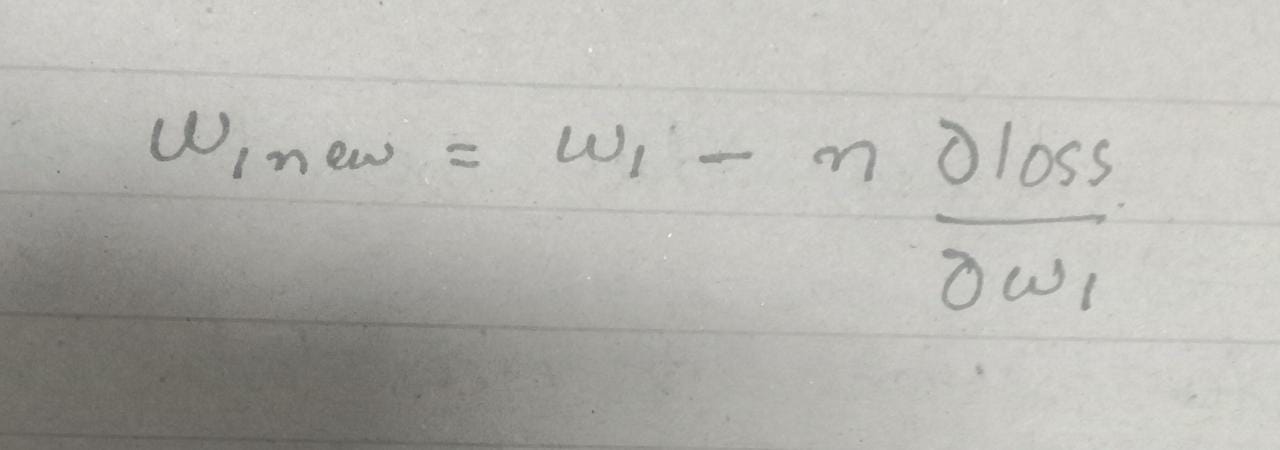
This problem can be removed by using another activation function know as ReLU (Rectified Linear Unit).

**2. Exploding Gradient Problem:**

Remember exploding gradient problem occurs due to **weights** not because of any activation function. We try to understand it from a new example.

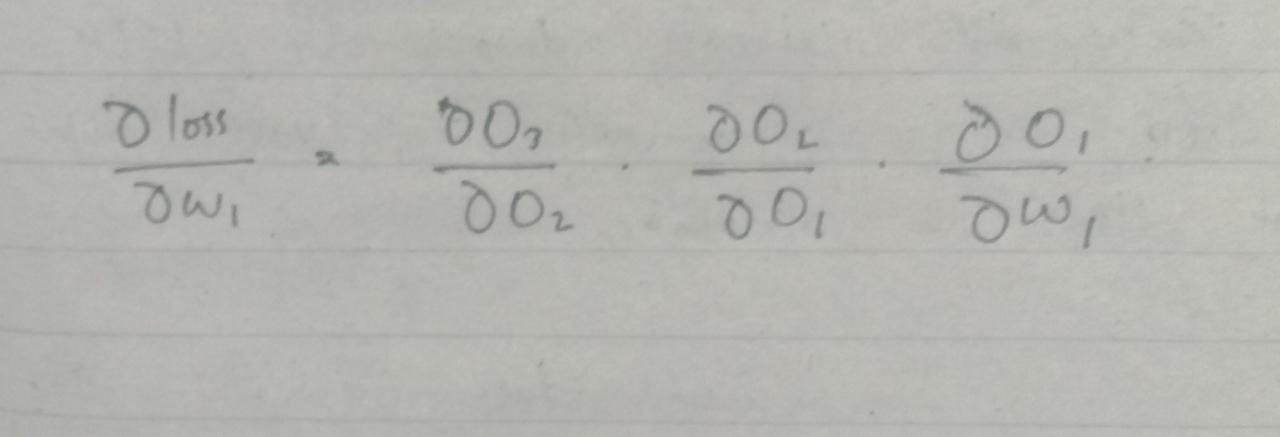


This neural network having three hidden layers l1,l2 and l3 which is also called output layer. In the second layer, the product of output of first layer (o1) and weight (w2) act as a input which is shown by Z. Now during back propagation let suppose we have to update the weight (w1). We will use an optimizer gradient descend which uses the chain rule.

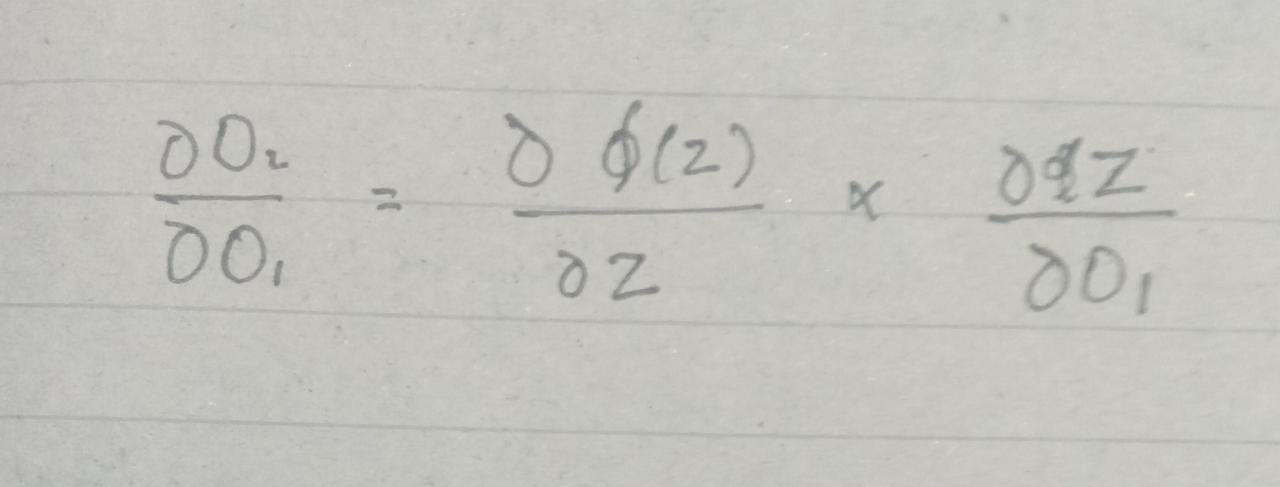


This is how the gradient descend updates the weight (w1). Now just look at the

derivative part in the equation.

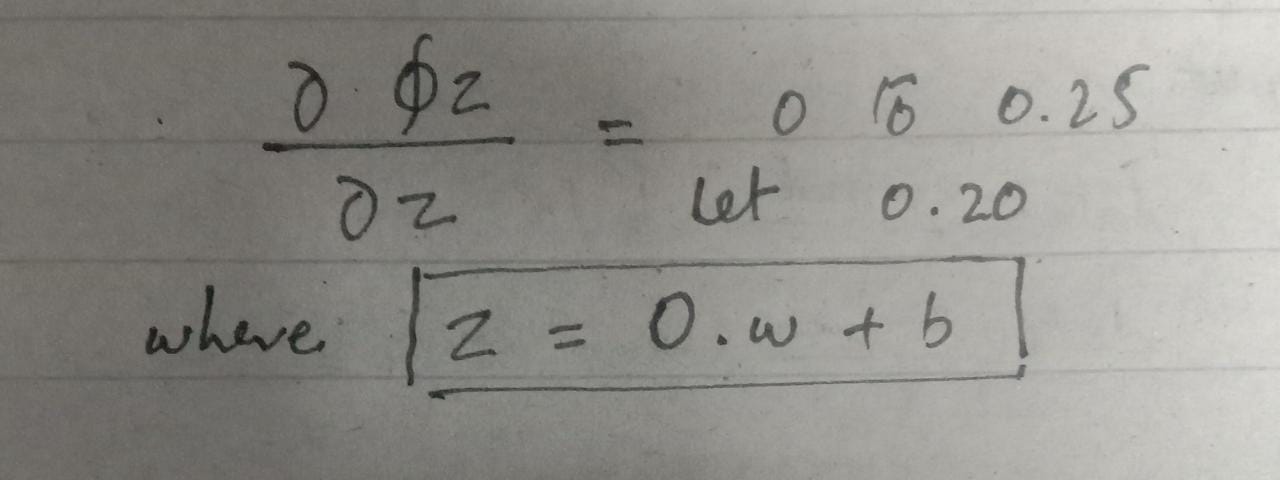


Here if we just take the second derivative from R.H.S i.e derivative of o2 w.r.t o1.

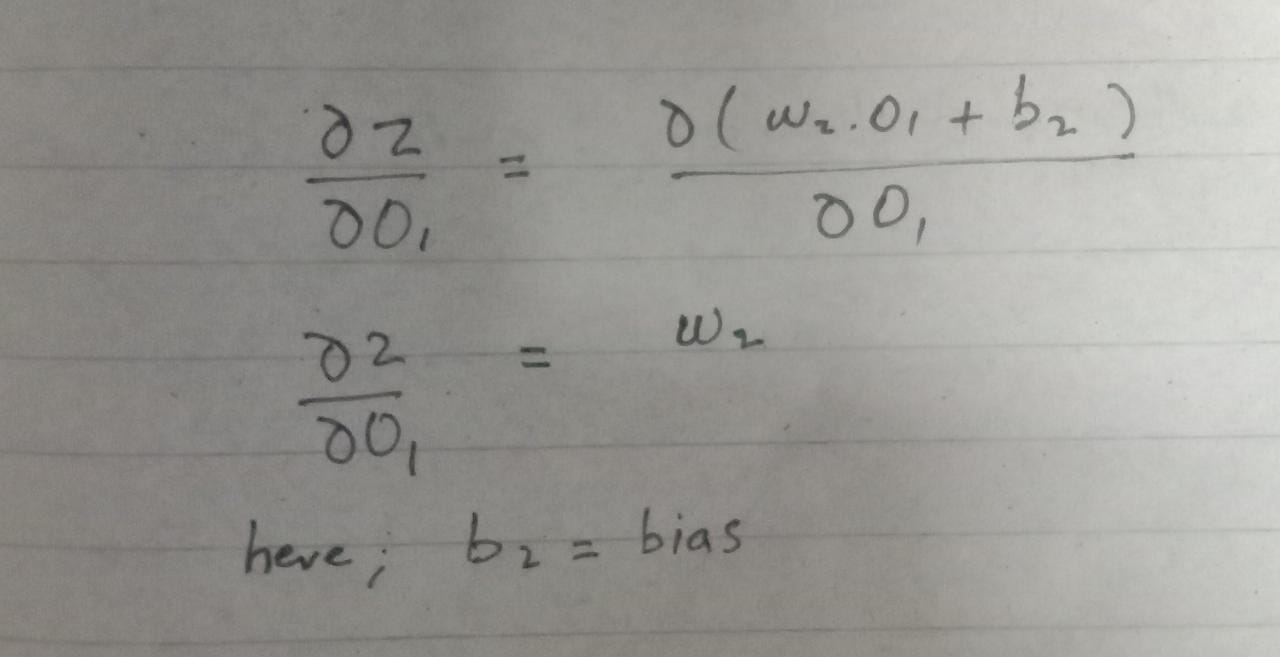


Now here actually Chain Rule is applied. We know that the z is the input to layer(L2) and what it actually does that it applies activation function on that input, let suppose sigmoid function and generates output (o2). So, the output (o2) depends upon the sigmoid function &(z) which in turn depends upon the input (z) and the input (z) itself depends upon the output of layer (L1) i.e o1.

We know that the derivative of sigma function is between 0 to 0.25, so;

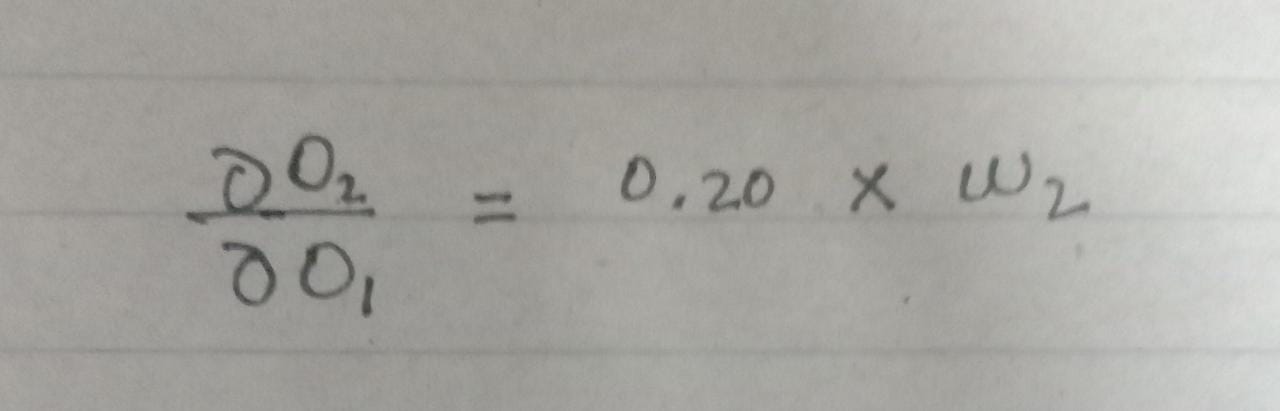


and,

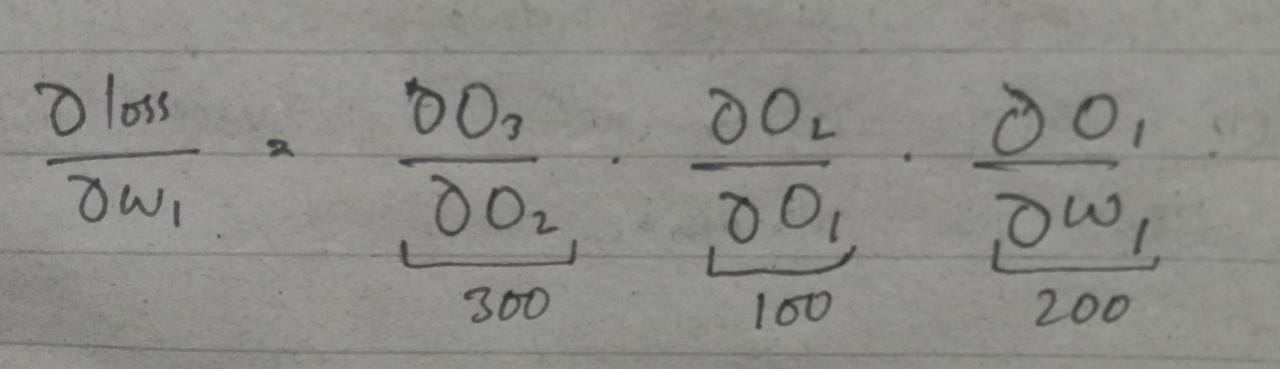


In the above example simple derivative rule has been applied.

So, the final equation becomes something like;



If we have initially assigned 500 to w2, so the answer will be 100. And the same goes for the other parts i.e



Now if this derivative is multiplied with any learning rate, let suppose 1 and then subtract from the initial weight i.e 500, so the new weight would be negative value. In this case these two weights are far different from each other. So if we plot these kind of weights, they will jump here and there on plane and the gradient descend will never be able to find the global minima or we could say that they will never converge.

**3. Activation Function:**

Any neural network consist of the following three layers:

1.Input layer

2.Hidden layer

3.Output

The activation function resides in hidden and output layers. It transforms the value in a particular range generated by hypothesis function i.e the product of input and weight to the sum of bias. In this way it decides weather the neuron to be fired/activated or not e.g.

Suppose we have a data of insurance company and we have to decide weather a person will buy an insurance or not?.It means that the output will be either 0 or 1.If 0 means he is not buying while 1 represents that the person is buying insurance. We supply data to the input layer, it transfers it to the hidden layer. From hidden computation starts and it will generate a value using hypothesis function for a particular neuron. This value might be out of the range of 0 and 1, so here activation function comes into picture. The value produced by hypothesis function is supplied as a input to activation function so that it limits the values between the range 0 and 1. Now the value generated through activation function of hidden layer is supplied to the neuron of output layer. The same process as stated before repeats here.

Some of the most widely used activation functions are:

1.Sigmoid function, range (0,1)

Equation :  
A = 1/(1 + e-x)

2.Tangent Hyperbolic function, tanh, range (-1,1)

Equation :

f(x) = tanh(x) = 2/(1 + e-2x) - 1

3.RELU, Rectified Linear Unit, range (0, number]

Equation :

A(x) = max(0,x). It gives an output x if x is positive and 0 otherwise.

4.Softmax function: useful in classification problem

Moreover, one of the main purpose of activation function is to introduce non-linearity to the network, otherwise it will be just like a regression model which is obvious not best for classification problems.

**4. LSTM (Long Short Term Memory) Networks**

We have discussed the issues occurs in standard RNNs i.e vanishing and exploding gradient. In other words we also say that the standard RNNs face long term memory dependency problem. It means that in many cases we need long memory for our prediction such as;

I live in Pakistan, which is situated in South of Asia …, I speak fluent *Urdu.*

Now in this example if the network has to predict Urdu. Here in this problem the network will easily understand that it has to predict any language but what language? This might be tough job for him because for this it has to look in the context of Pakistan and between Pakistan and Urdu, there may be a huge gap. So practically the standard RNNs are unable to memorize or remove this gap. And this is because of the **Vanishing Gradient** problem. We know that as the numbers of layers increases, the value of weight gradually moves towards consistency and a time comes when the previous value of weight approximately becomes equal to the updated value of weight. Hence, the network losses the long term memory.

To remove the problems of vanishing and exploding gradient, we use a special kind of RNN i.e LSTM networks, which is also called the LSTM architecture of RNN.

**The problem.**

With conventional “Back-Propagation Through Time" (BPTT, e.g., Williams

and Zipser 1992, Werbos 1988) or “Real-Time Recurrent Learning" (RTRL, e.g., Robinson and Fallside 1987), error signals “owing backwards in time" tend to either

(1) blow up or (2) vanish:

the temporal evolution of the backpropagated error exponentially depends on the size of the weights (Hochreiter 1991).

**Case**

(1) may lead to oscillating weights, while in case

(2) learning to bridge long time lags takes a prohibitive amount of time, or does not work at all

**The remedy**

This paper presents “Long Short-Term Memory" (LSTM),

1. a novel recurrent network architecture in conjunction with an
2. appropriate gradient-based learning algorithm.

LSTM is designed to overcome these error back-flow problems. It can learn to bridge time intervals in excess of 1000 steps even in case of noisy, incompressible input sequences, without loss of short time lag capabilities. This is achieved by an efficient, gradient-based algorithm for an architecture.

enforcing constant (thus neither exploding nor vanishing) error ow through internal states of special units

**(provided the gradient computation is truncated at certain architecture-specific points, this does not affect long-term error flow though).**

|  |  |  |
| --- | --- | --- |
| 1 | **Gradient-descent variants.** The approaches of Elman (1988), Fahlman (1991), Williams  (1989), Schmidhuber (1992a), Pearlmutter (1989), and many of the related algorithms in Pearl-mutter's comprehensive overview (1995) suffer from the same problems as BPTT and RTRL (see Sections 1 and 3). | 1. Elman, J. L. (1988). Finding structure in time. Technical Report CRL Technical Report 8801, Center for Research in Language, University of California, San Diego.  2. Fahlman, S. E. (1991). The recurrent cascade-correlation learning algorithm. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, Advances in Neural Information Processing Systems 3 , pages 190{196. San Mateo, CA: Morgan Kaufmann..  3.Schmidhuber, J. (1992a). A xed size storage O ( n 3 ) time complexity learning algorithm for fully recurrent continually running networks. Neural Computation , 4(2):243{248. |
| 2 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |