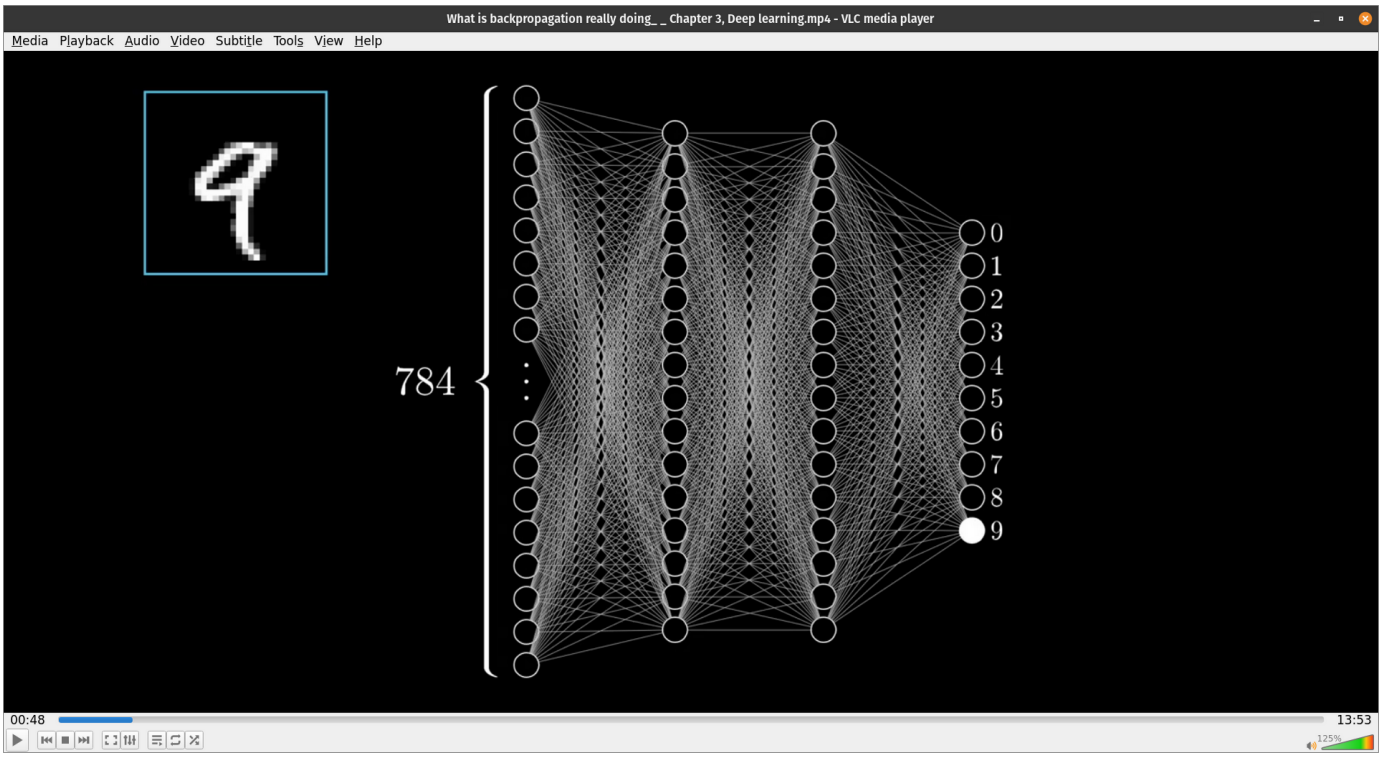
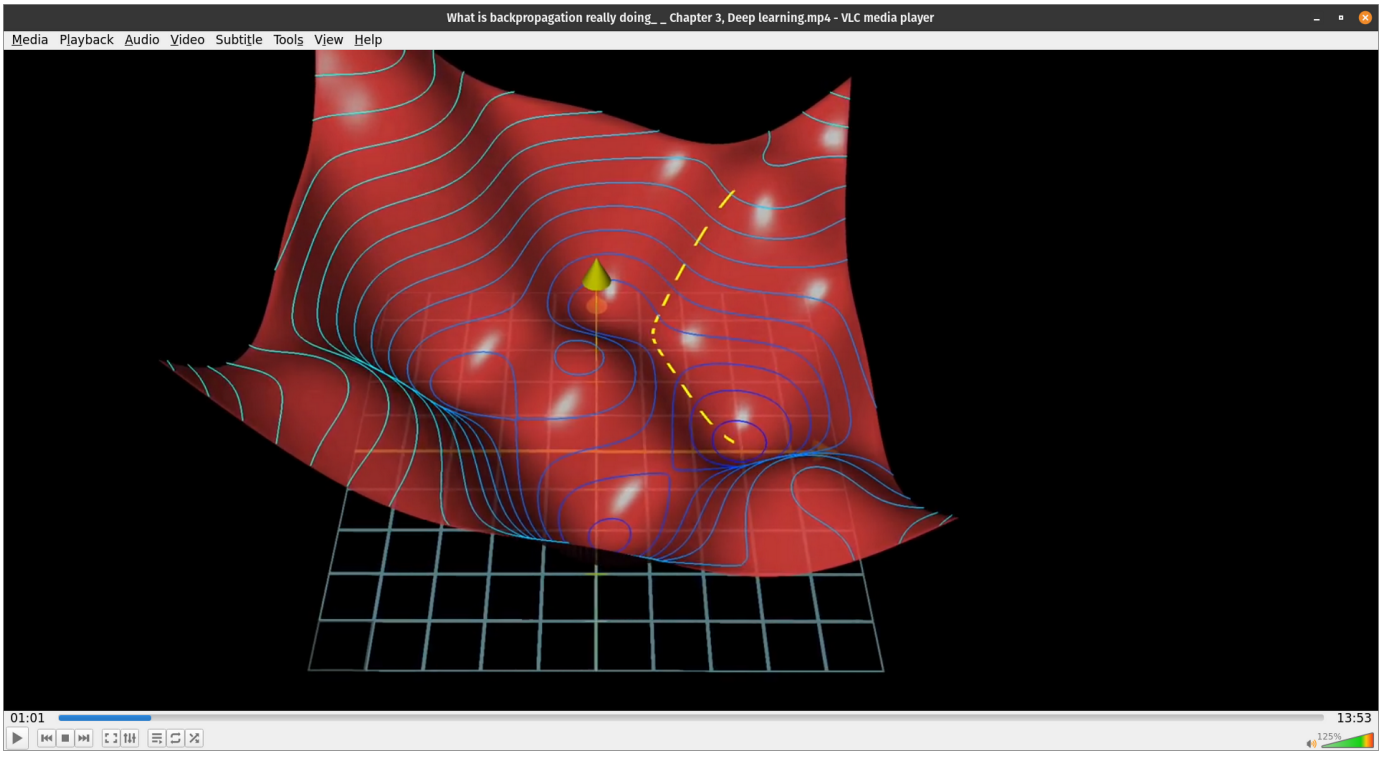
What is Backpropagation Doing, 3Blue1Brown

Handling the classic example of DIgit Recognition

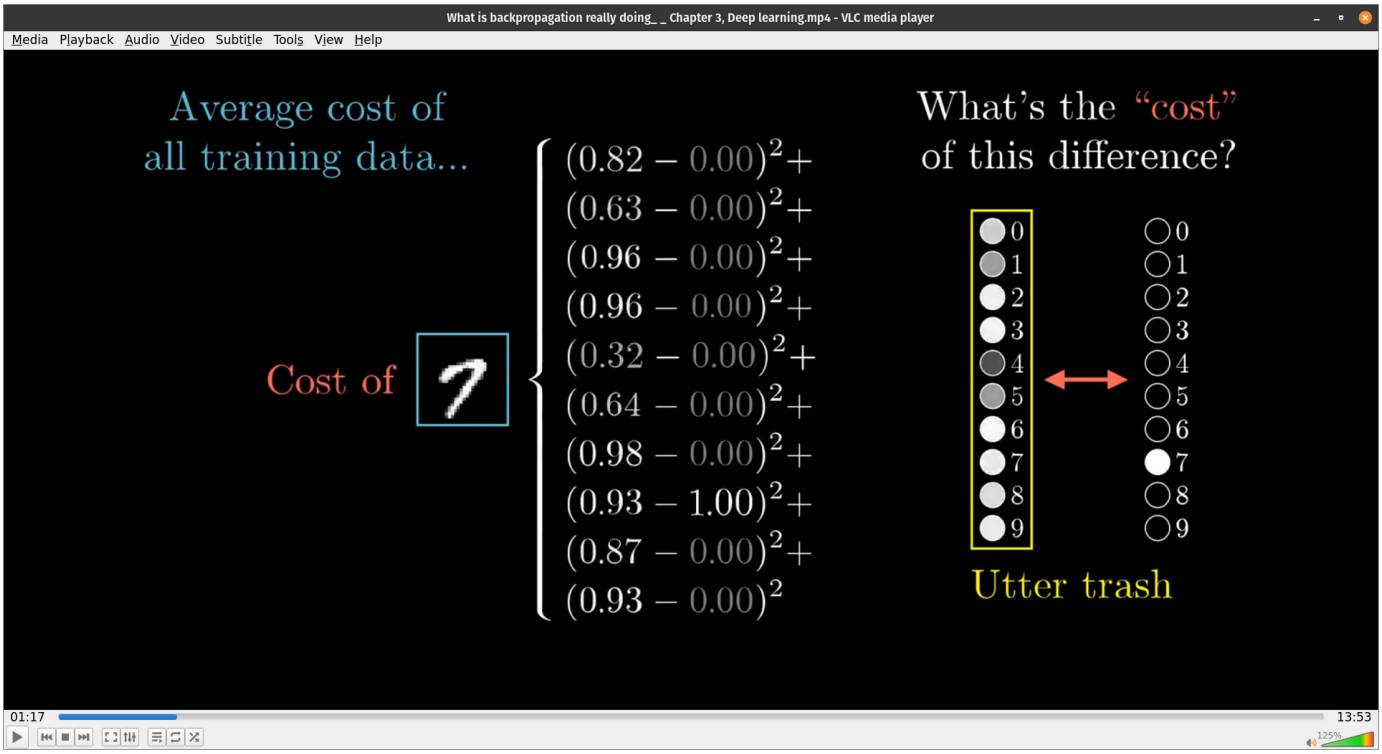


Gardient Descent visualisation of 2 input function



The yellow line depicts the path of the gradient descent to find the minima of the function.

How to get the cost of the current predictions:

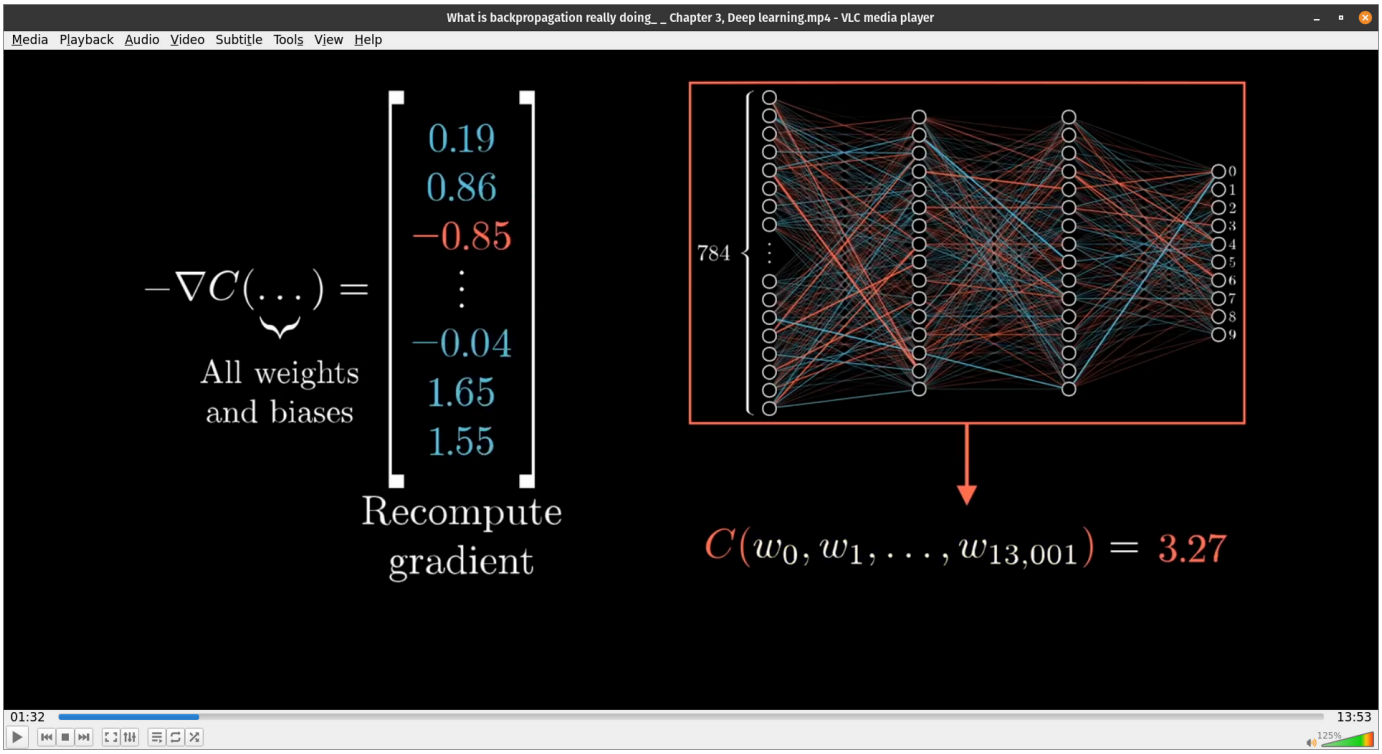


The cost of the model prediction is calculated by taking out the average of the squared difference of the expected prediction and current prediction.

The above image shows the current predictions of our model and the right column shows the expected result.

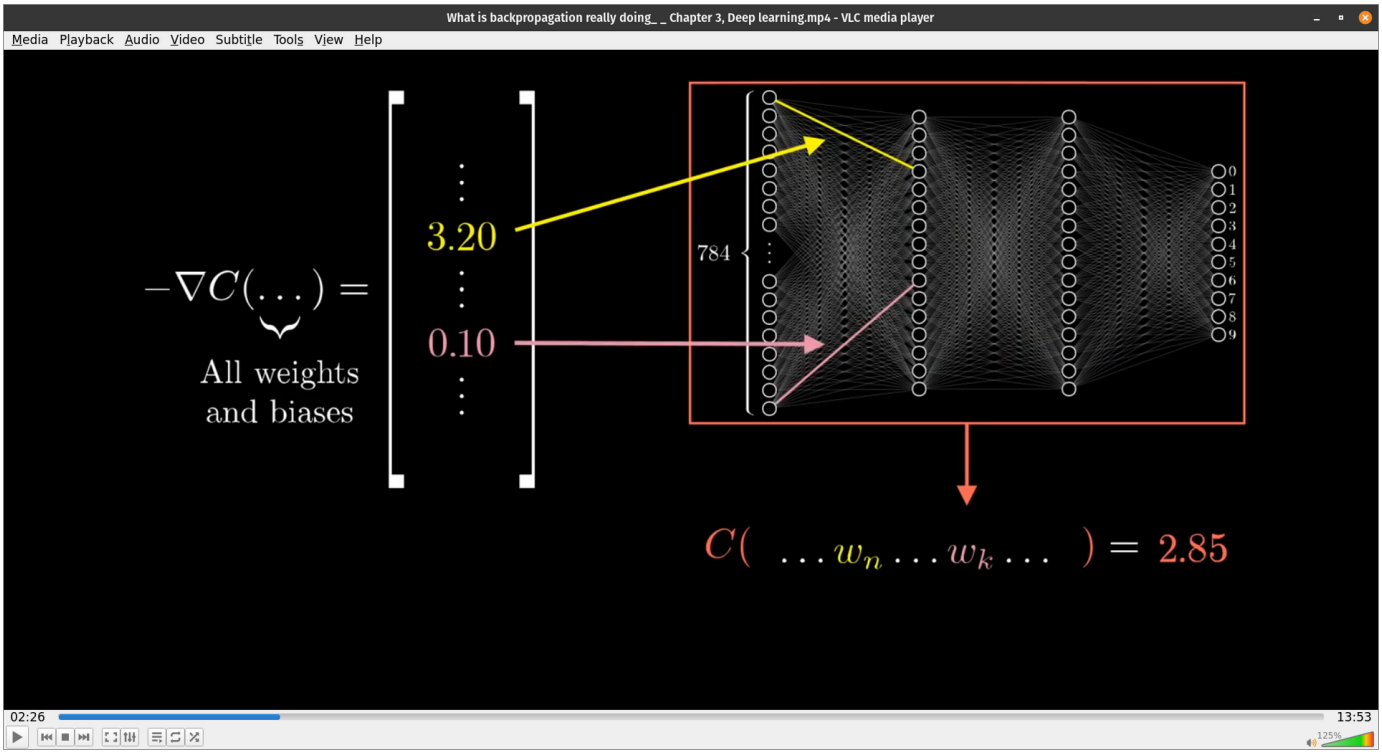
$$Cost = \frac{1}{m}\sum_{1}^{m}(predicted\ value - expected\ value)^2$$

We actually need to look at the `Negative Gardient` of the cost function to reach the minima.



Here, ∇ represents the Gradient of a multidimensional vector.
The magnitude of each weight in the Gradient Vector of the cost function, represents how sensitive is the cost to that weight.

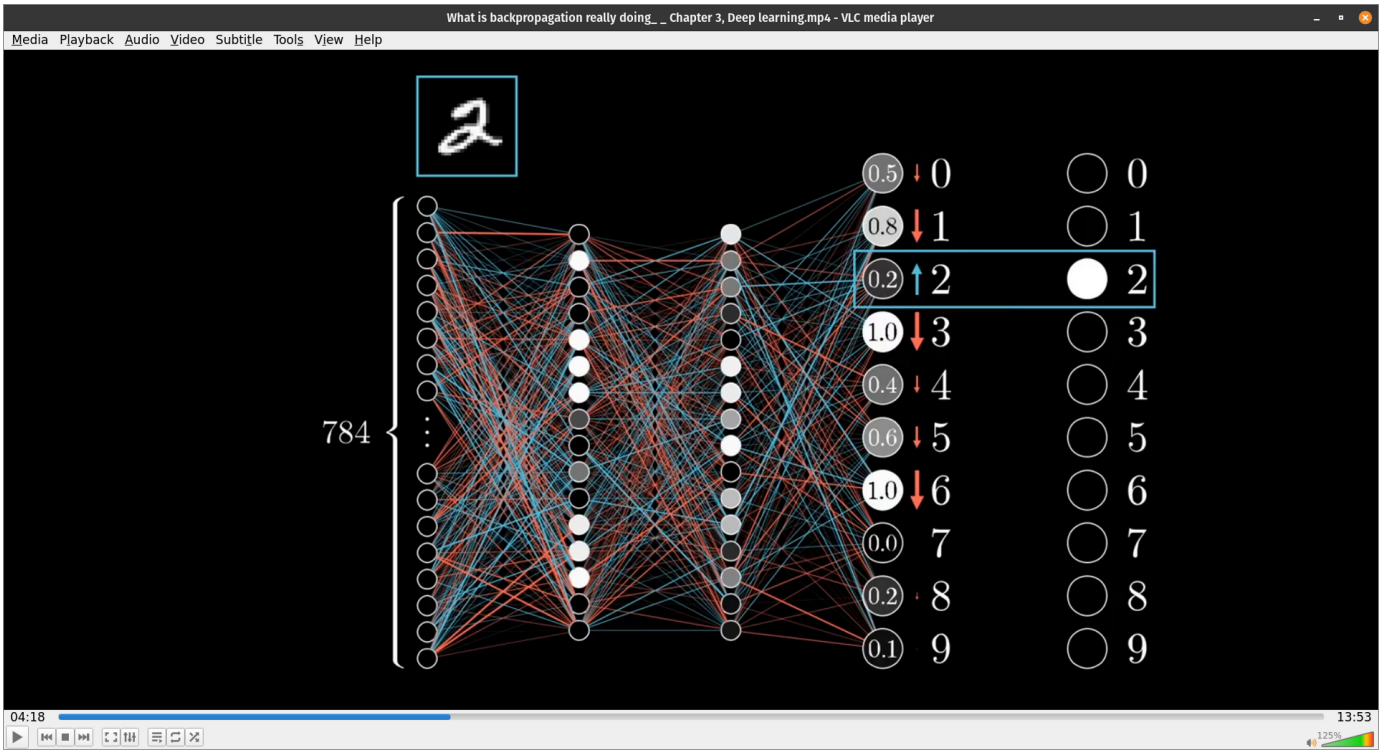
For eg:



Here, the cost function is 32 time more sensitive to the changes in the first weight, than in the other.

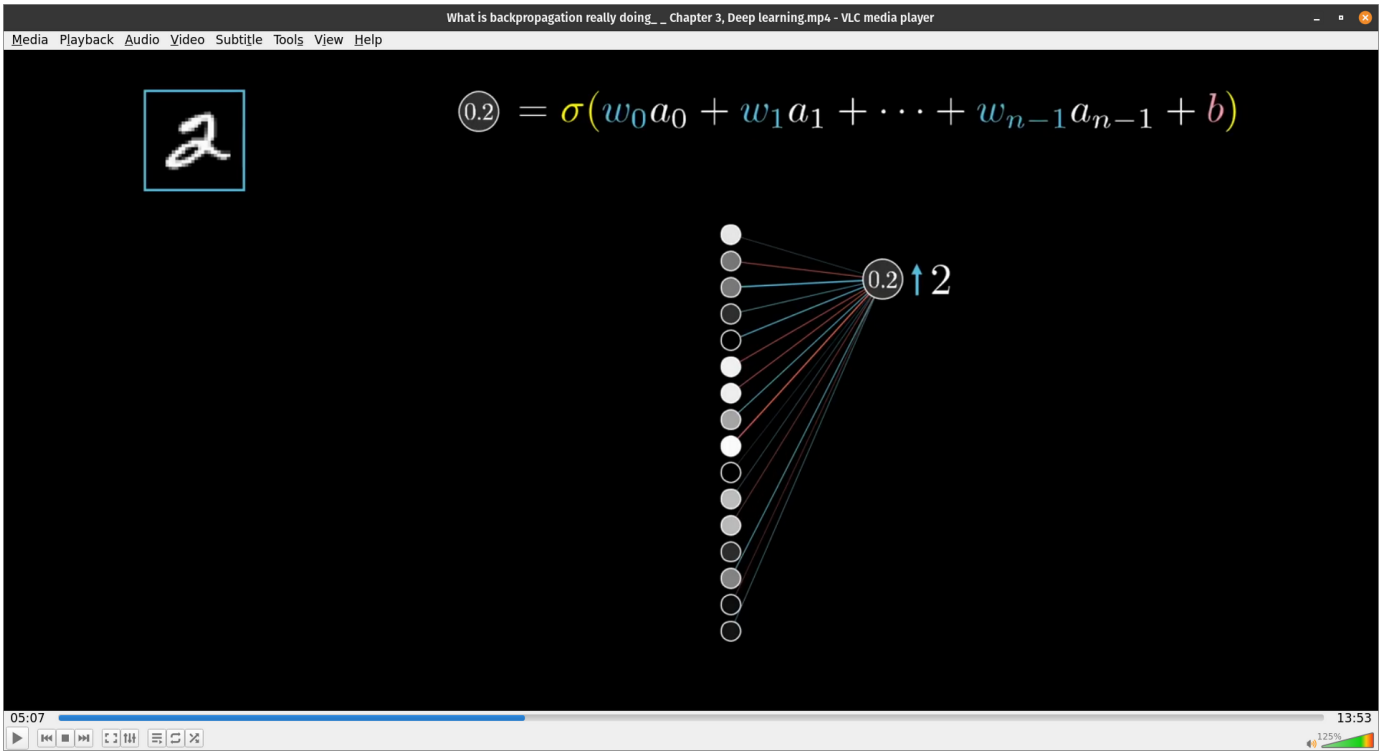
Intuition of Backpropagation

We want to increase the activations in the final layer to get close to predictions, but we can only change the weights and biases to get closer. Though, it is necessary to take note of the variation of the results with the change:



for eg. the increase in activation of neuron 2 is more important the decrement in activation of neuron 8 which is already closer to where it should be.

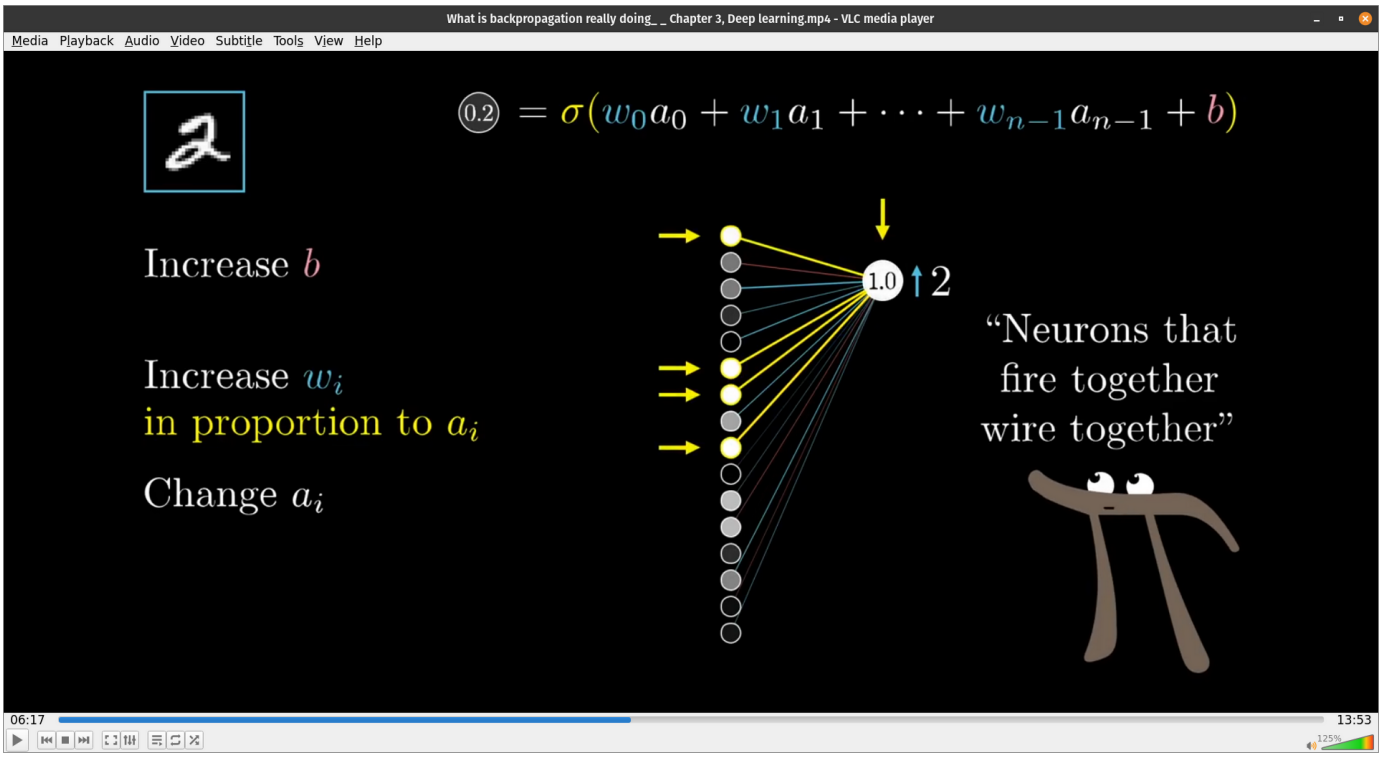
To do this, we see the notation of the activation of the target neuron:



We can alter the activation of the target neuron by 3 ways:

1. Increase weights w_i
2. Alter the activations from the previous layer a_i
3. Increase the Bias b

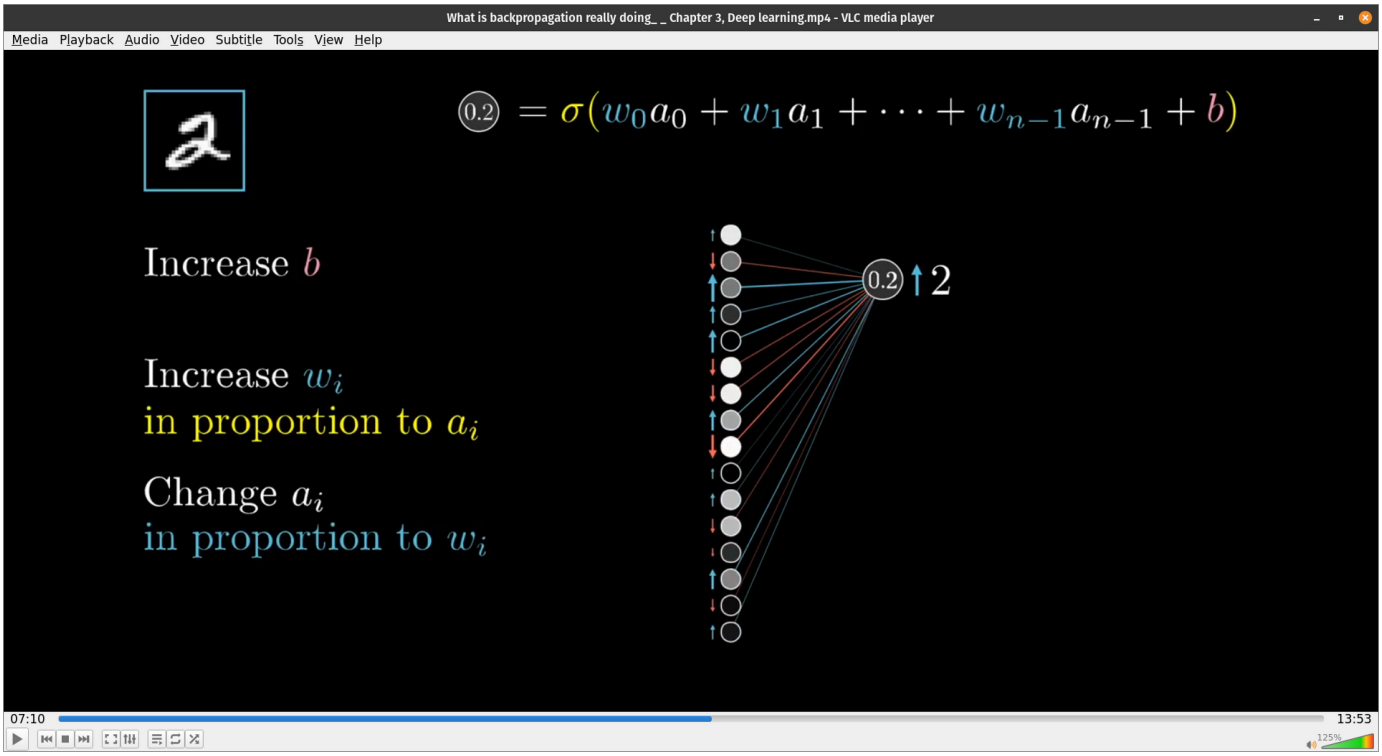
The weights of the brighter neurons (high activations) have higher say in the change in the activation of the current neuron since it is multiplied by larger activation value.



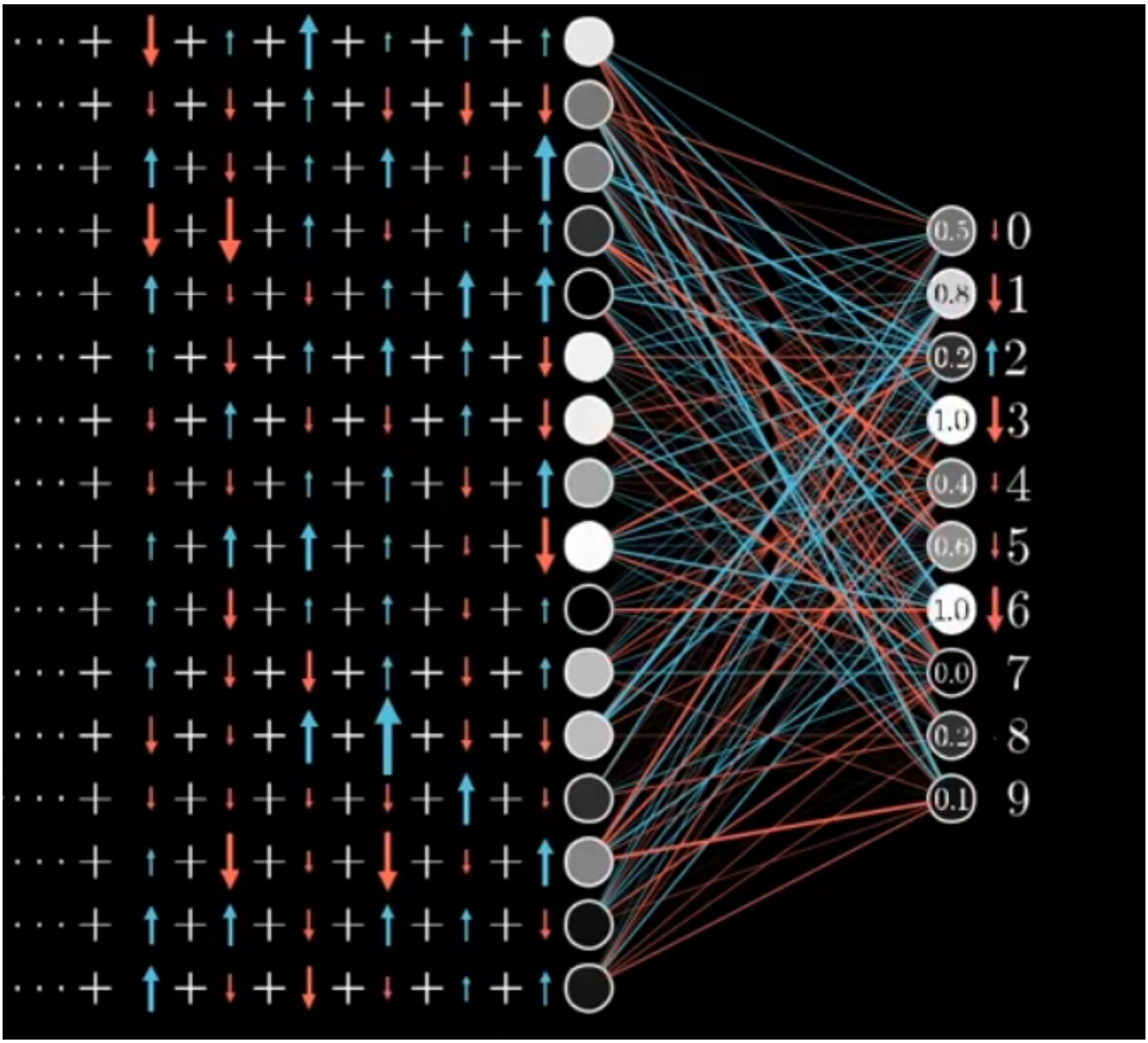
Coming to the second way, which is to alter the activations from previous layer. Though we can't do that directly, but just keeping track of that.

Here, we want:

1. The neurons with the positive weights to get brighter, and
2. The neurons with the negative weight to get dimmer.



Note that this is only what the target neuron wants, but the activation changes in previous layer also affect the other activations of the other neurons in the final layer.

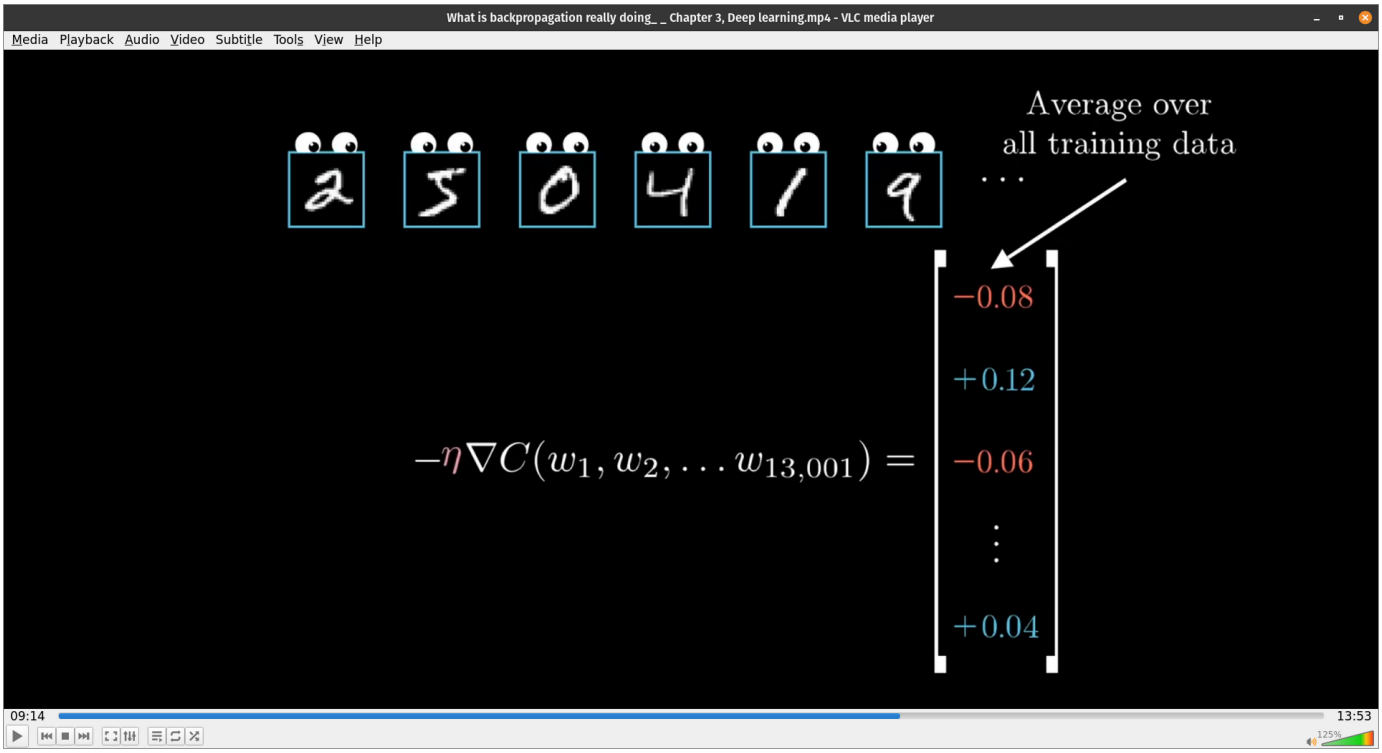


Hence, we just add the desirable changes of all the neurons together, and this is where the concept of `BACK PROPAGATION ARISES`.

This gives a list of desired changes in the second last layer, and we can recurssively follow up this to the starting layer.

We follow up the same process for all the training examples and average them out.

This is same as the -ve gradient of COST FUNCTION, or something proportional to it



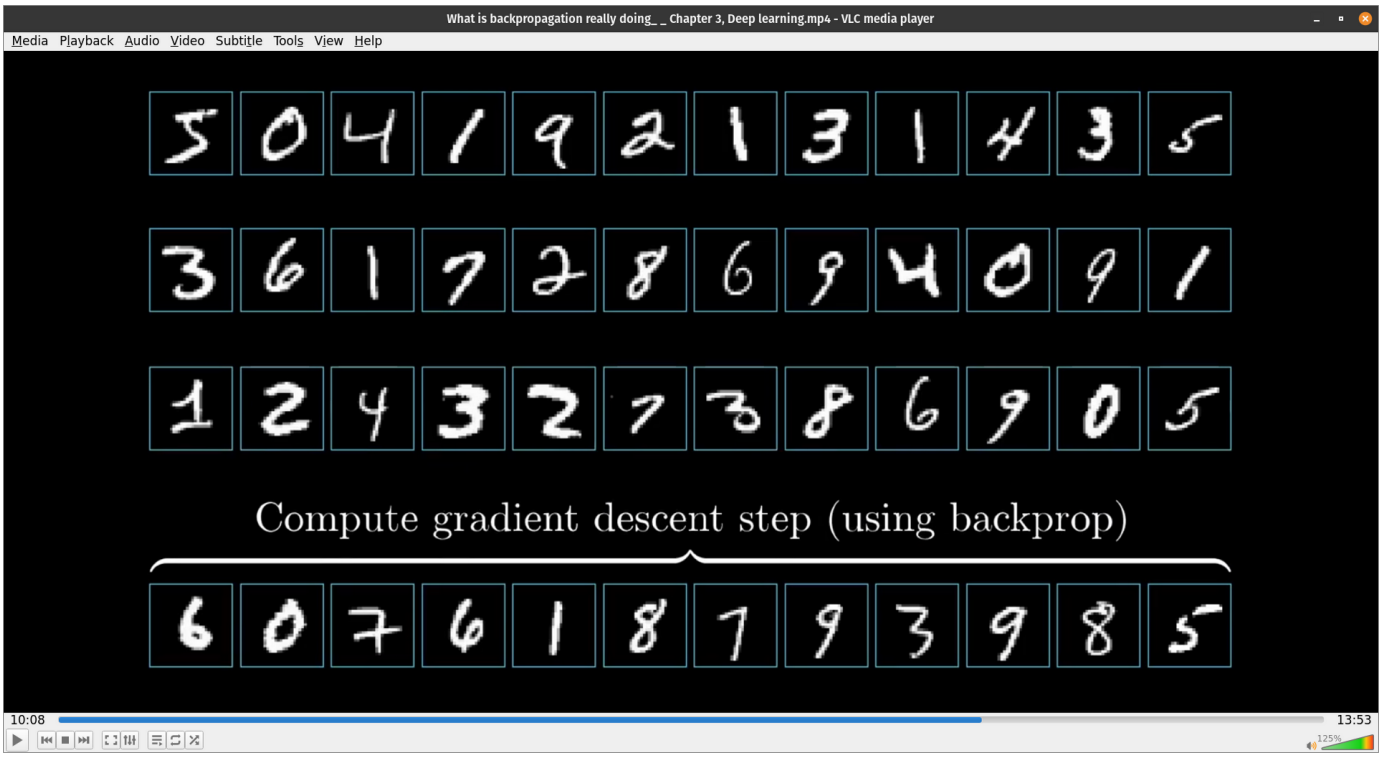
Stochastic Gradient Descent

The computation of the changes over all the training data takes too much time and resources for the Computers.

So, to get rid of this problem, we:

1. Randomly shuffle the complete dataset
2. create mini batches of training examples
3. Do computations on all of those mini batches seperately

Even though this would cost on our accuracy, (hence not the most efficient) but it is more preferred for lower computation and each mini batch



The actual path of a gradient descent would look more like a Drunk man going down aimlessly, rather than a careful calculating man choosing each step towards the bottom point.