NEURAL NETWORKS (3B1B)

Convolutional neural network → Good for image recognition

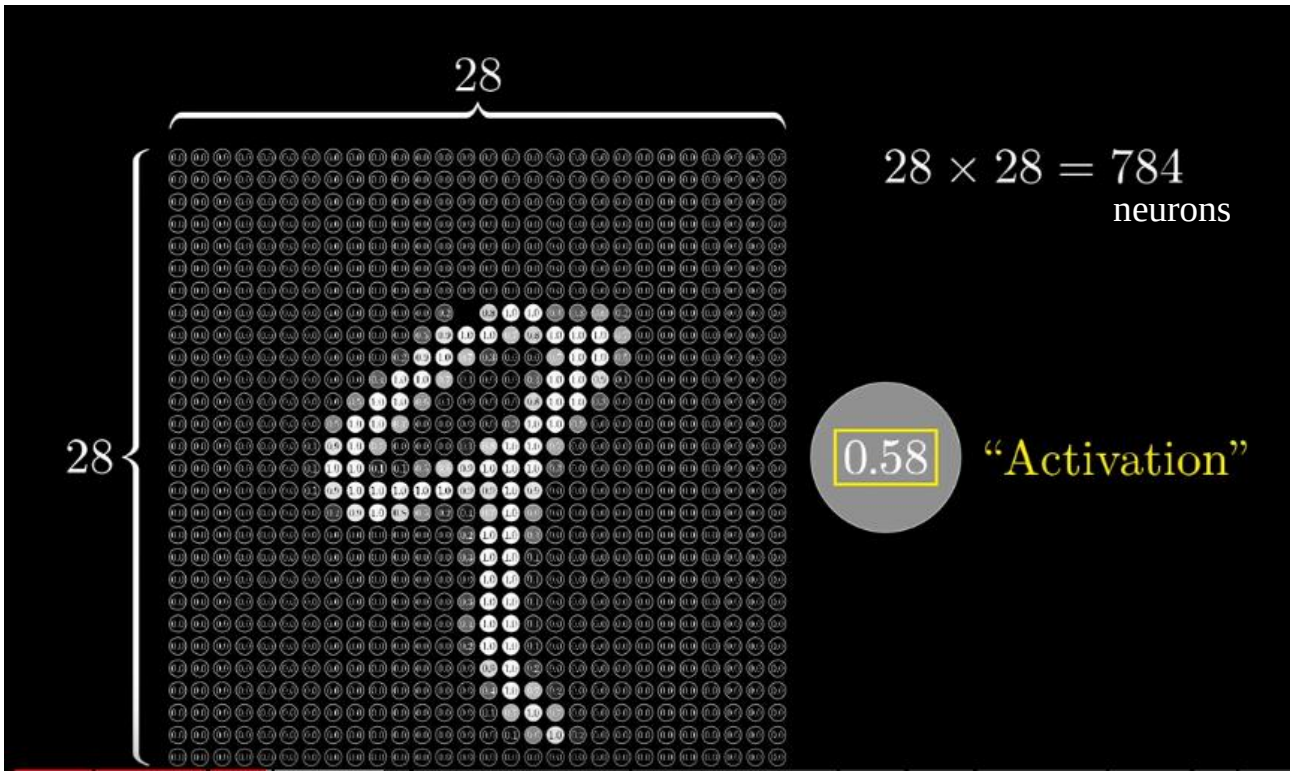
Long short- term memory network –> Good for speech recognition

Neural Network

What are the neurons?

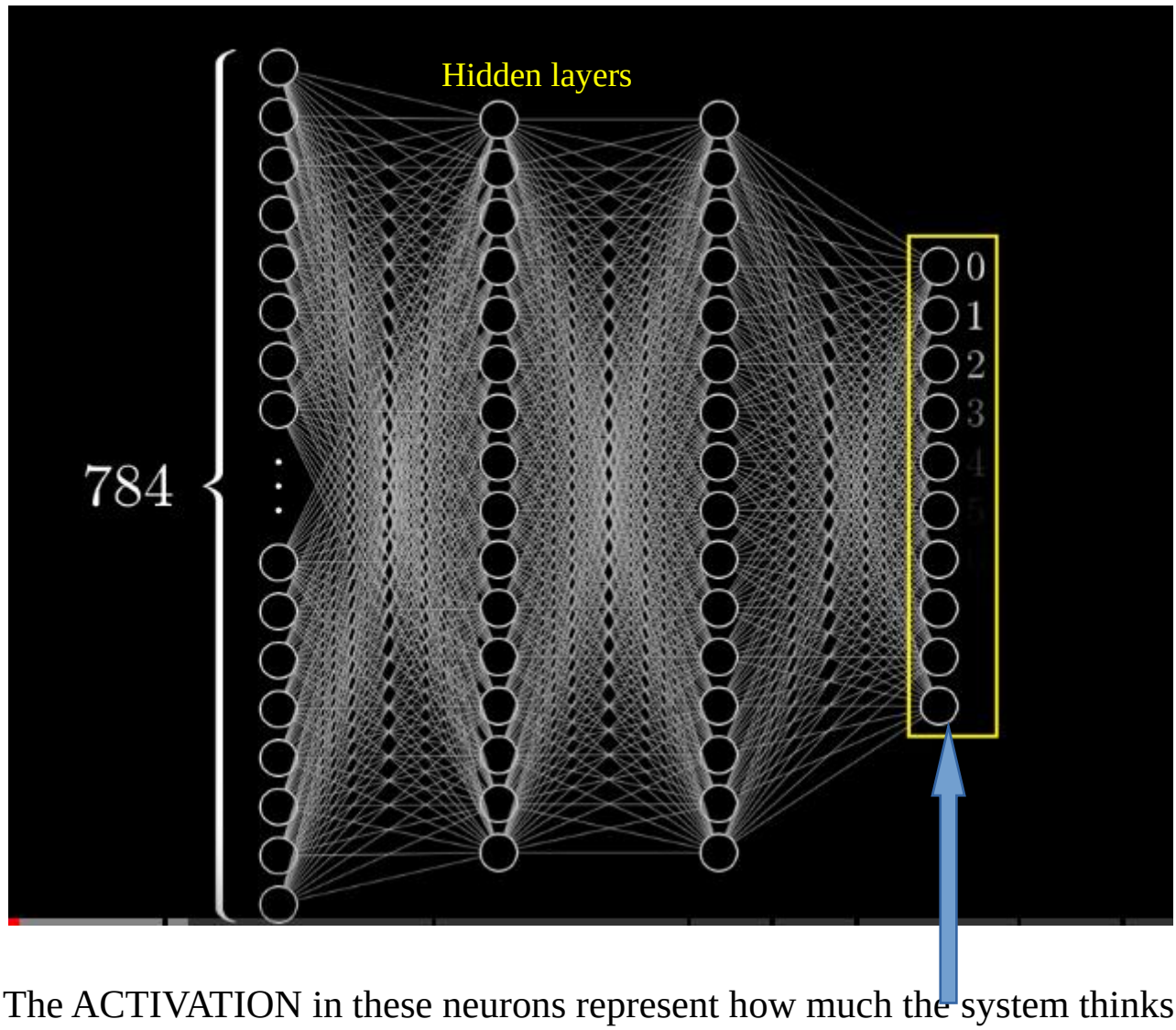
How they are connected?

Neuron → Thing that holds a number (specifically between 0 and 1)



The number inside neuron is called ACTIVATION.

These 784 neurons make first layer of network.



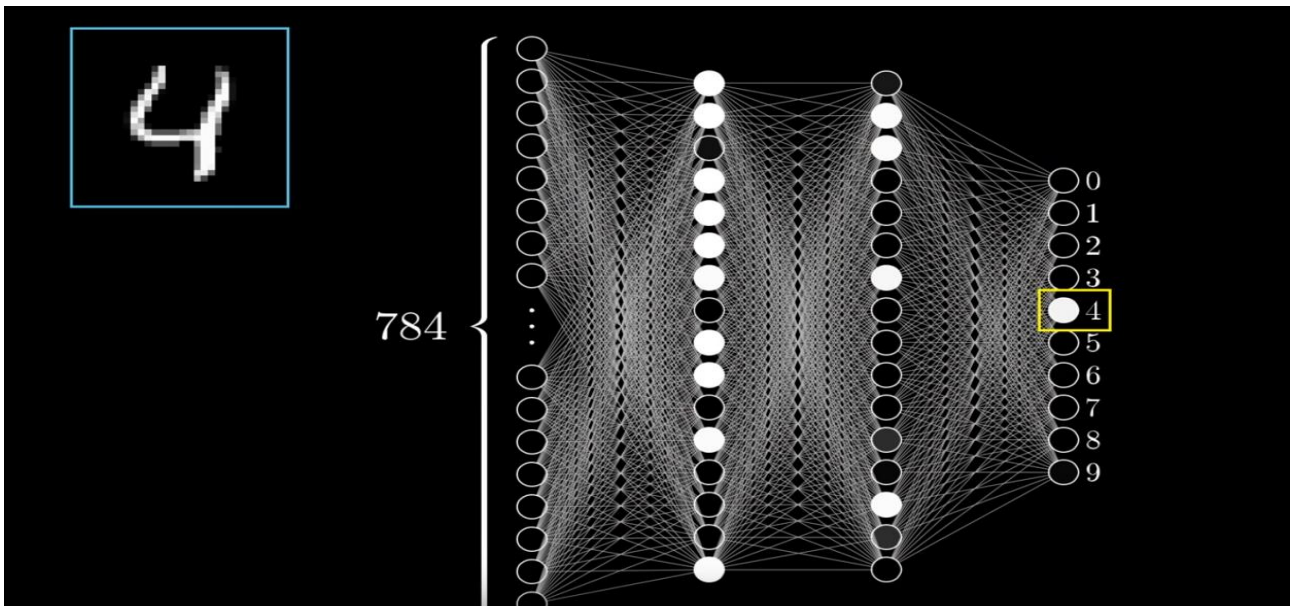
The ACTIVATION in these neurons represent how much the system thinks that a given image corresponds with a given digit.

Brief Intro:

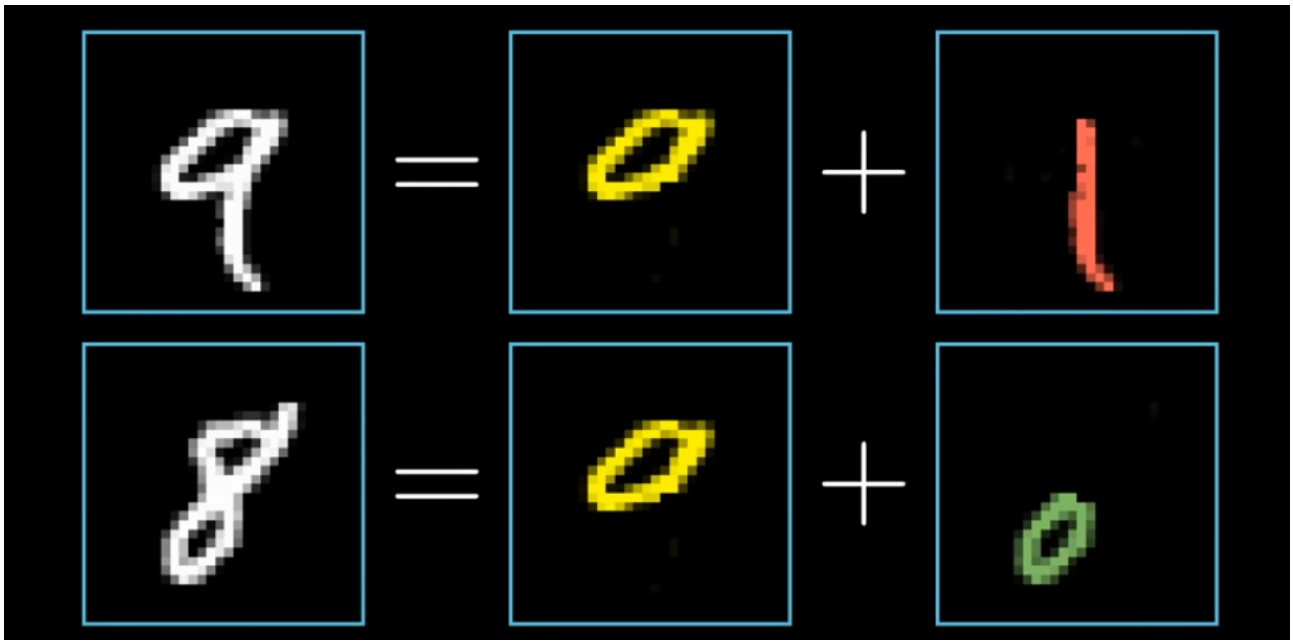
Activations in one layer brings activation in next layer.

All bright neurons forms pattern of activations and causes pattern of activation in next layer and so on and finally gives some pattern in output layer.

The brightest neuron of output layer gives the final result. (Here the digit).

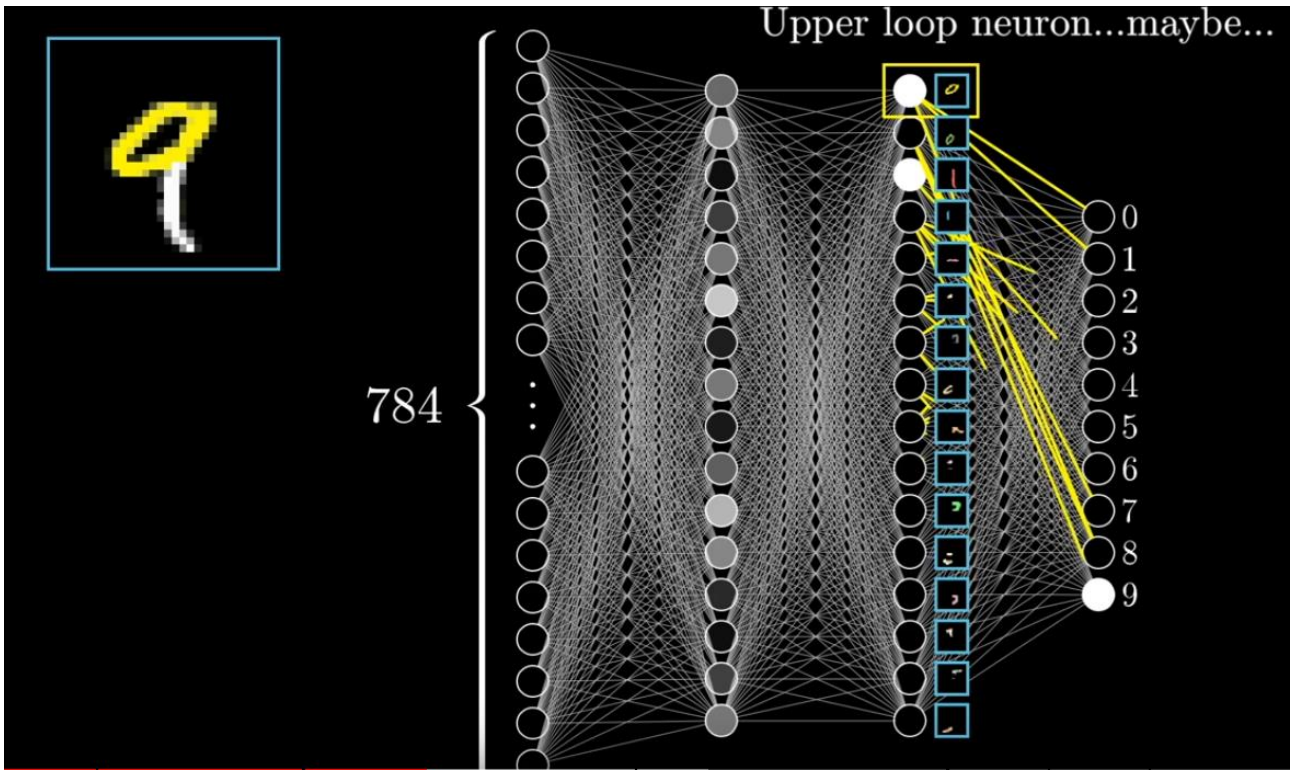


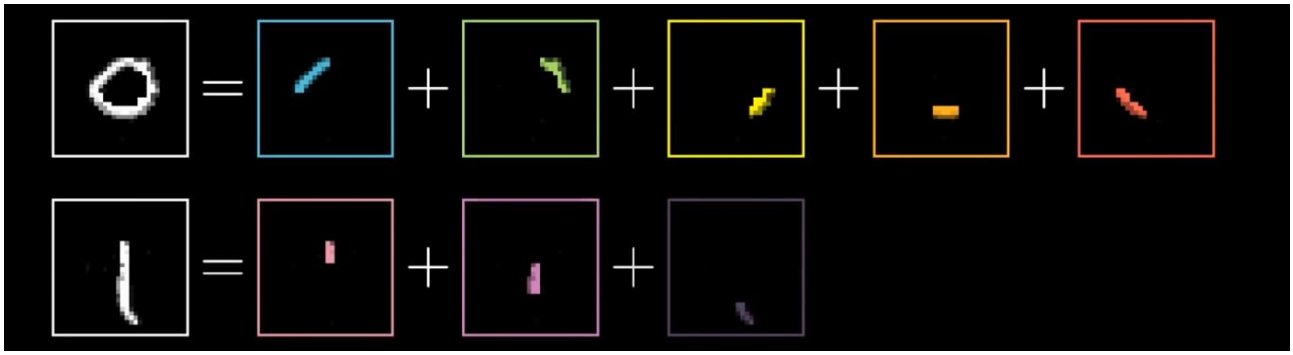
Why the layers:



The digits are broken down into smaller patterns like loops,lines ,etc.

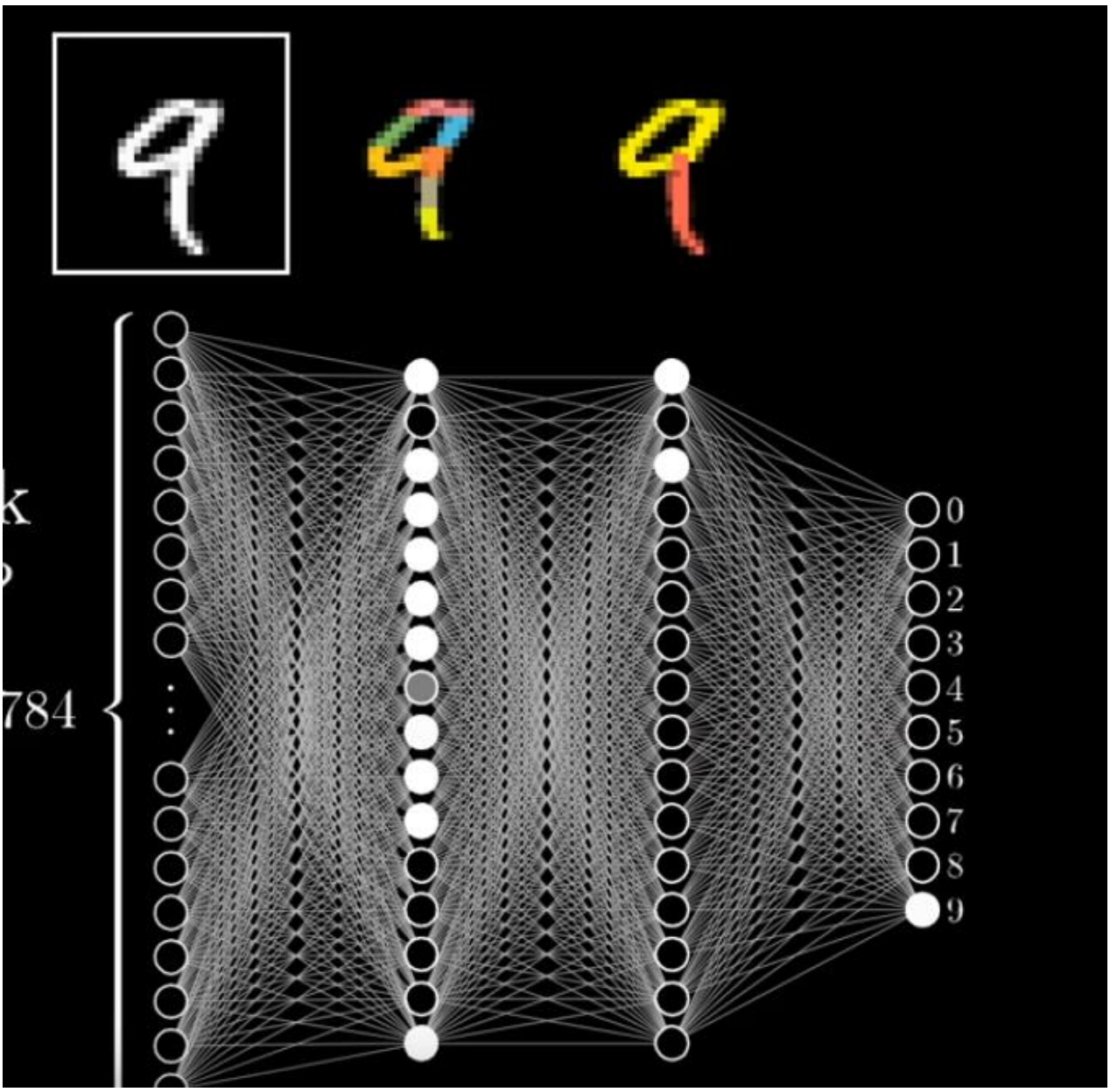
Each neuron in 3rd layer (or 2nd hidden layer) represents a sub pattern of digit and based on the combination of activation in these neurons , corresponding neuron of output layer is brighten up and digit is recognised.





The sub-patterns (loops,lines) are further broken down into more simpler patterns which forms the activation of neurons of 2nd layer or 1st hidden layer.

Neurons of this 2nd layer or “thin edge layer” combines accordingly and results in pattern of activation in 3rd layer and so on.



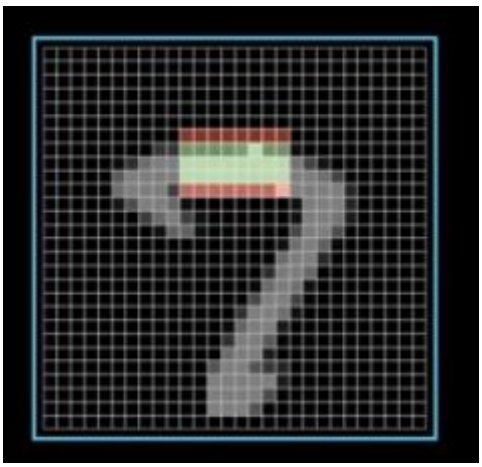
Edge detection:

we assign a weight to each one of the connections between our neuron and neurons of first layer. Then we take all activations of first layer and compute their weighted sum.

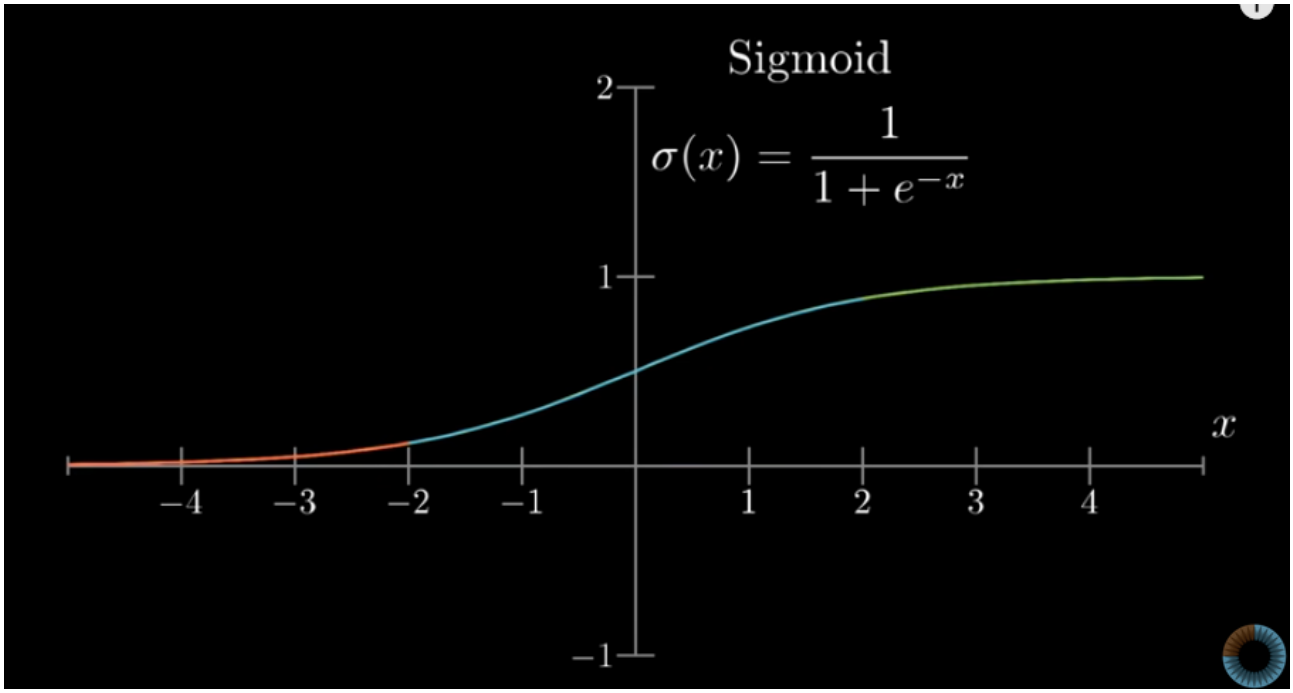
$w_1a_1 + w_2a_2 + w_3a_3 + \ldots\ldots + w_na_n$

we take weights of all pixels 0 except for the positive weights in the required region and calculate its weighted sum.

For detecting edge we take negative pixels associated with surrounding pixels. The sum becomes max when middle pixels are bright and surrounding pixels are dark.



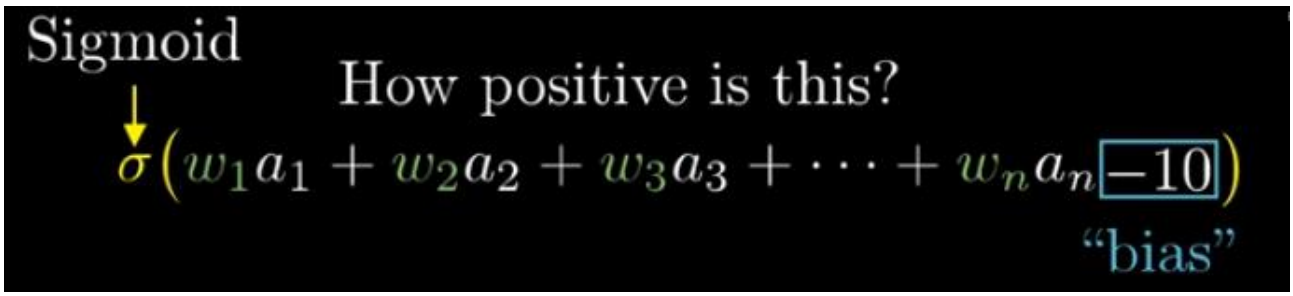
Since we want activations in range 0 to 1, we use **sigmoid** fn (logistic curve) which squishes its value between 0 and 1.



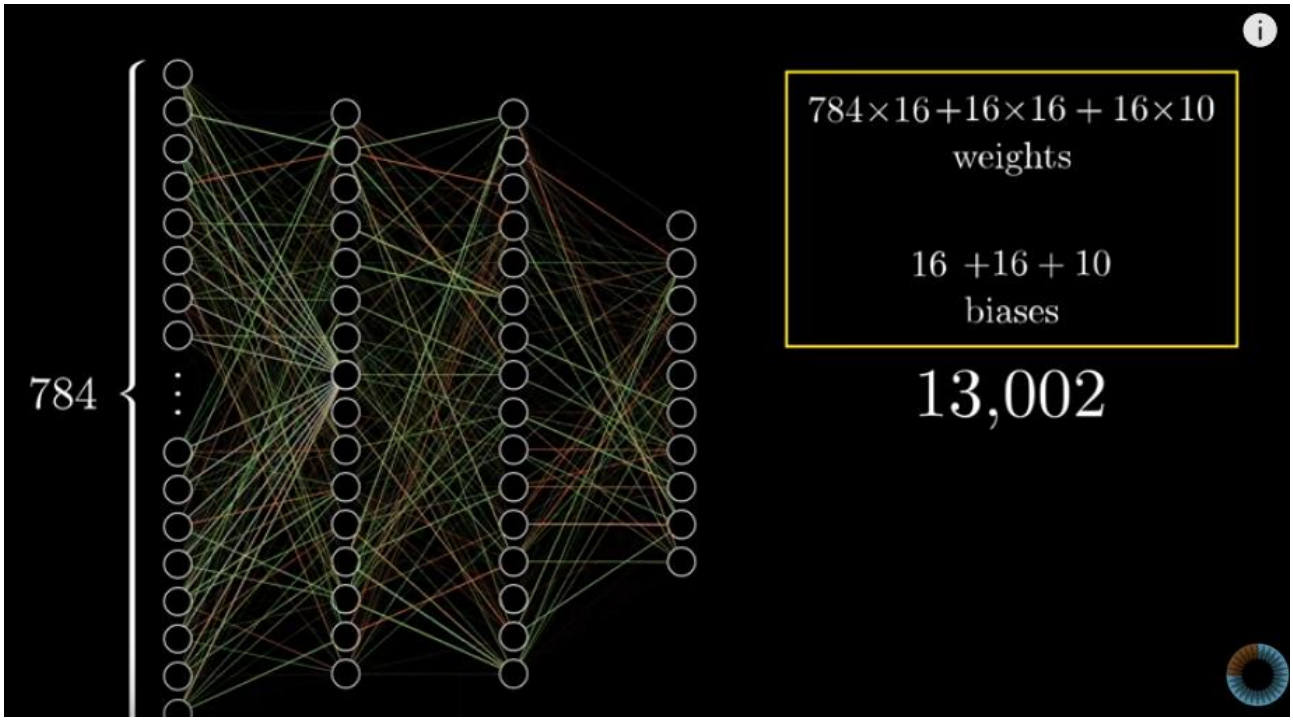
Activation of neuron is measure of how positive the relevant weighted sum is.

You might want neuron to be active when weighted sum > 10 (Bias for inactivity) so we add -10 to weighted sum before plugging it through sigmoid squishification fn.

The additional no. is called the **Bias.**



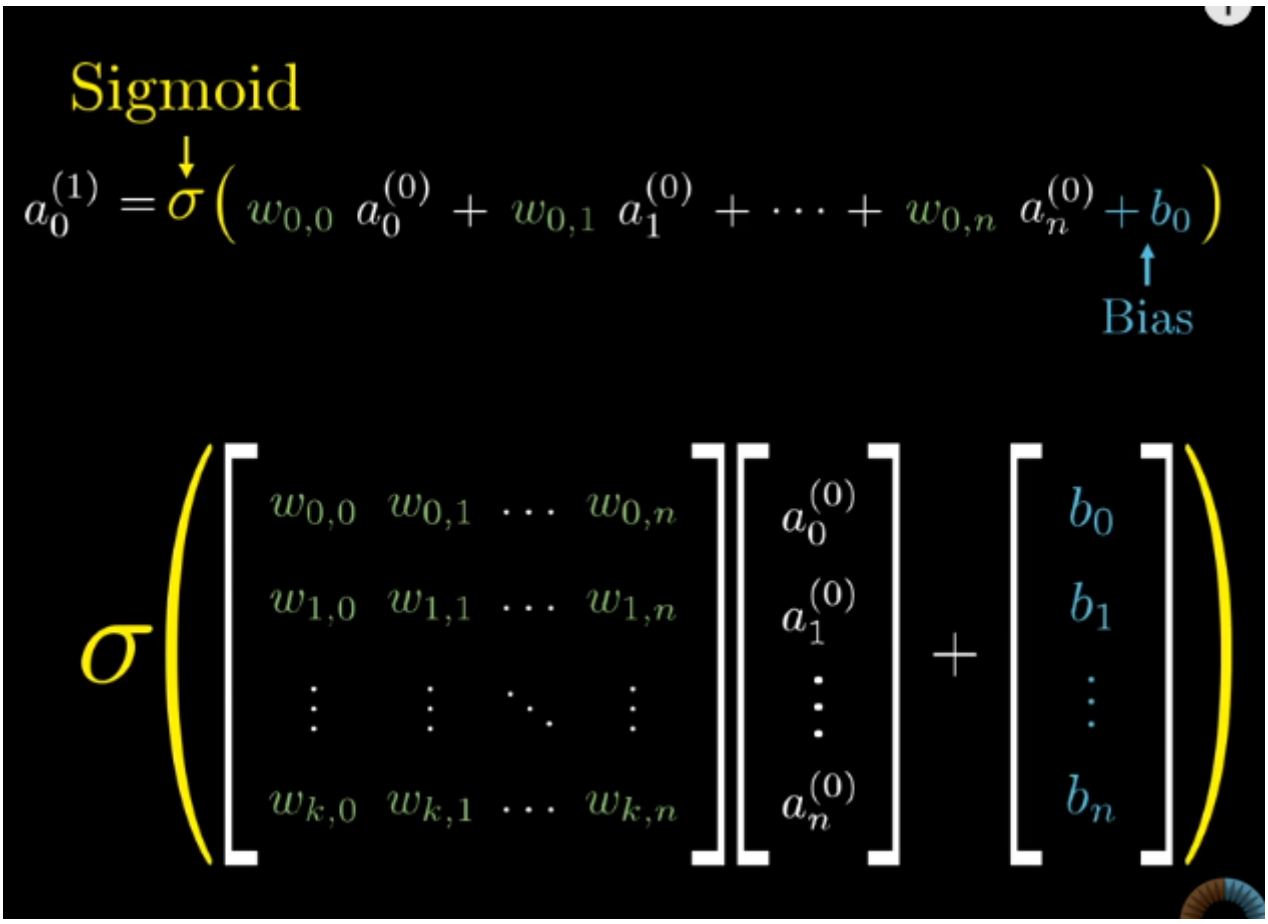
Weights tell us what pixel this neuron in 2nd layer is picking up on and bias tells us how high the weighted sum needs to be bfor neuron starts getting meaningfully active.



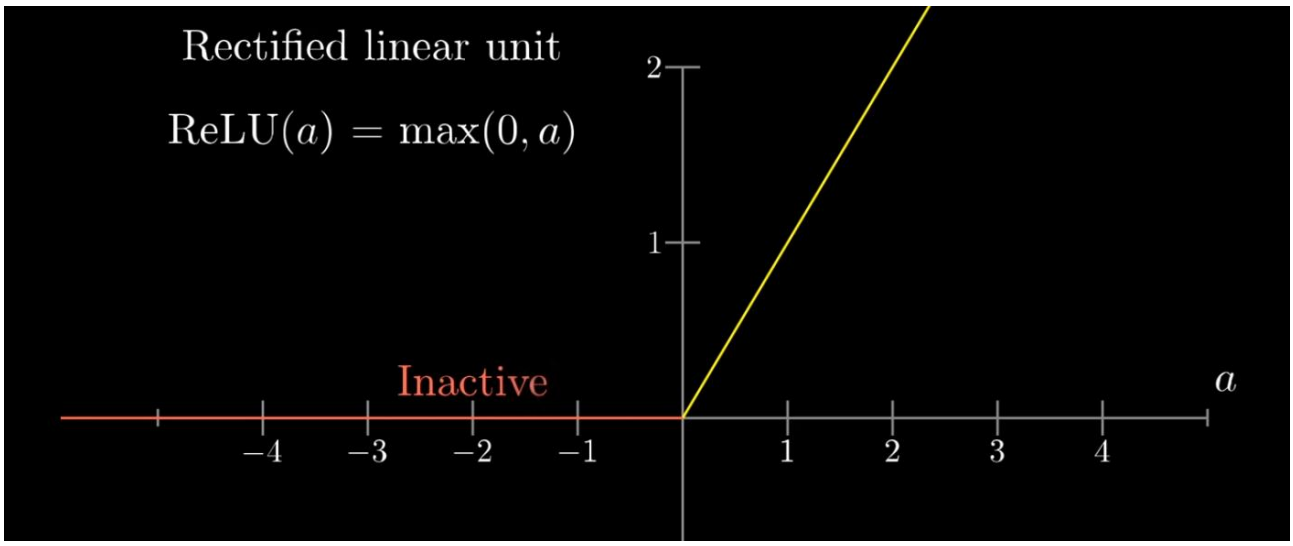
use of LA:

we organize all activations from one layer into a cloumn as a vector and organize all weights as matrix where each row corresponds to connections between one layer and particular neuron in next layer

we also add vector of bias to the earlier matrix product and finally calculate sigmoid of the expression.

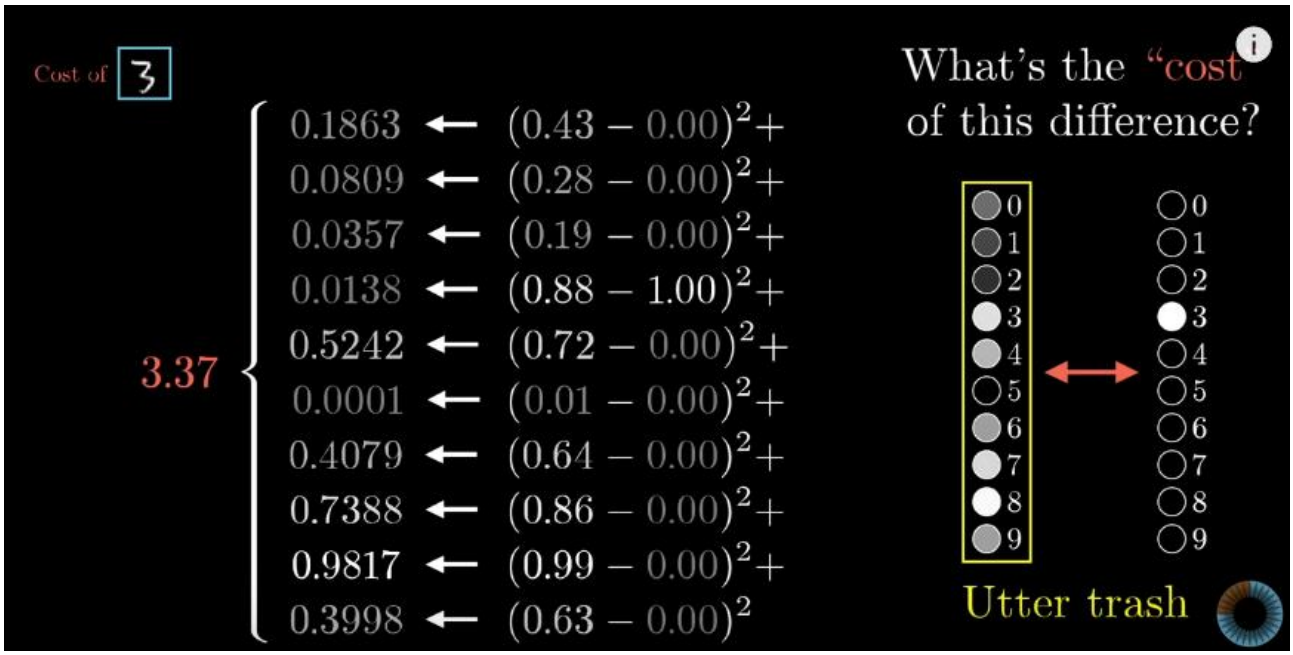


$$\mathbf{a}^{(1)} = \sigma\left(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b}\right)$$

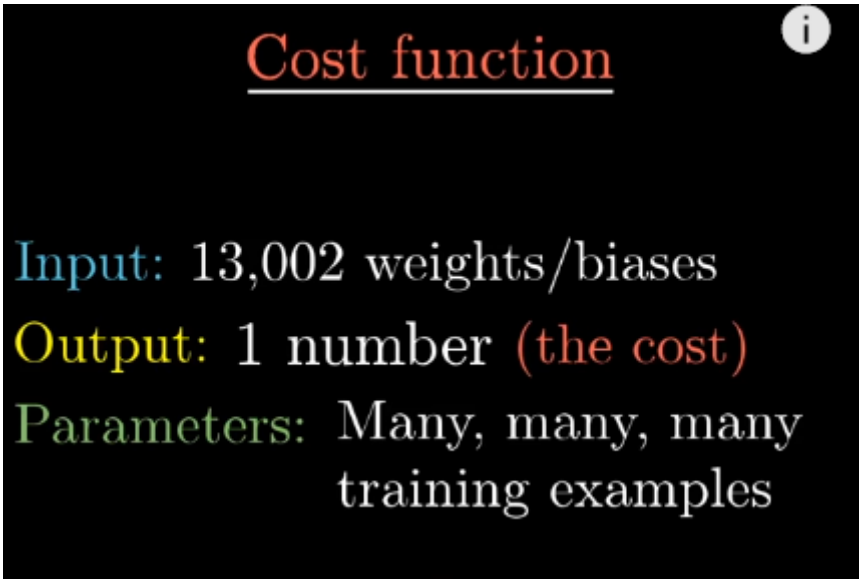


cost function:

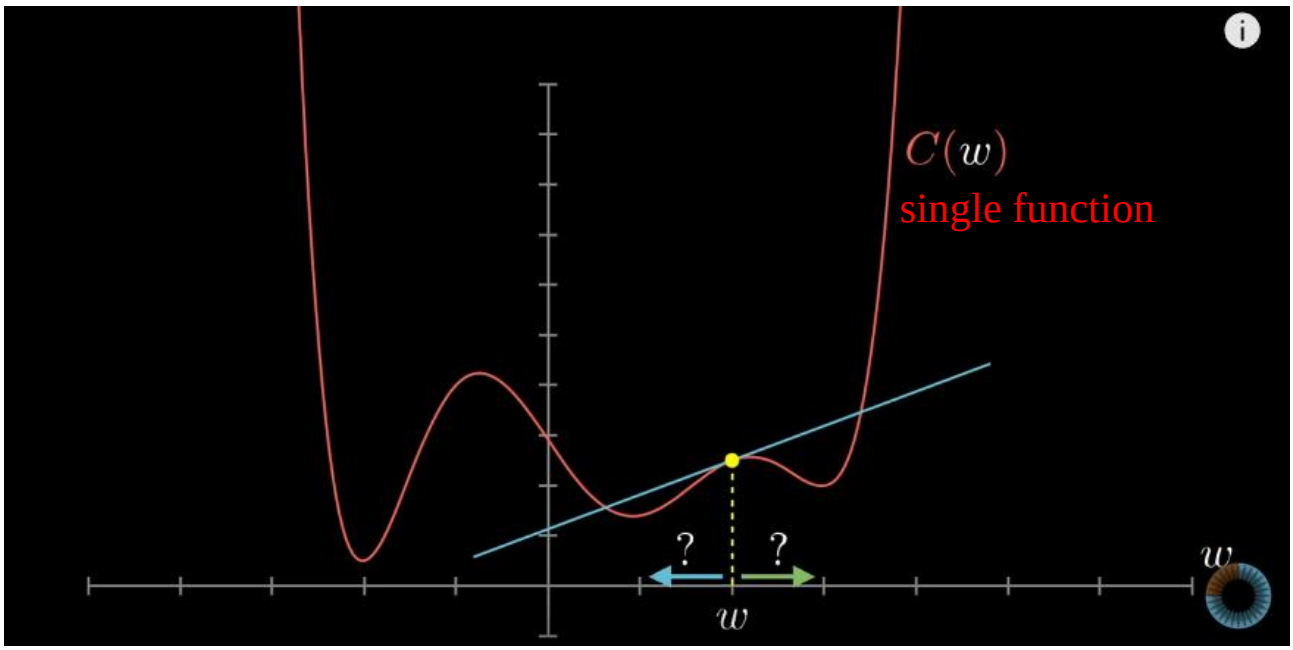
we add up square of the differences between each of the trash output activations and the value that we want them to have which is called **cost** of single training example



We take avg cost of all training examples which tells us how much incorrect the calculation can be.



since the process becomes awful for many training examples having large number of inputs, we tranform cost fn which takes only single input.

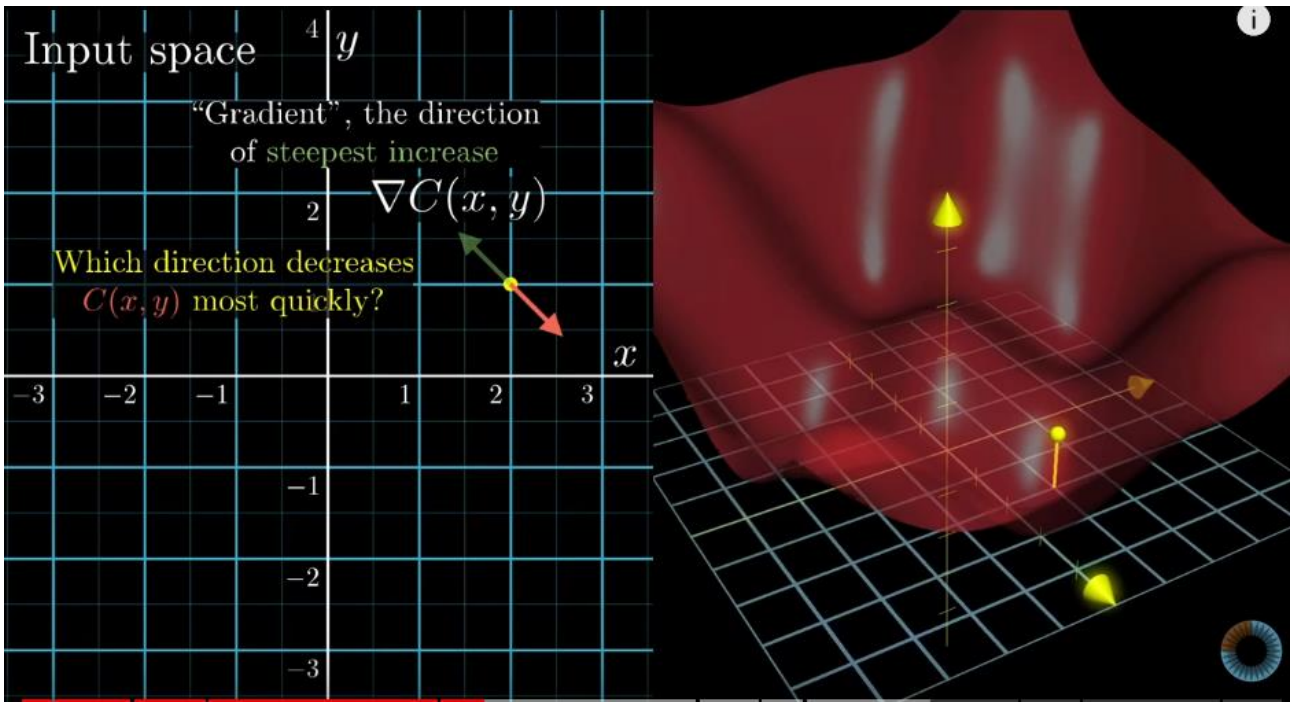


The minima of graph gives the minimum value of cost function which is required. Using differential to find minima sometimes gives infeasible results. Therfore we do it using slope of graph.

If the slope is positive move towards left and if slope is negative shift towards right.

GRADIENT DESCENT:

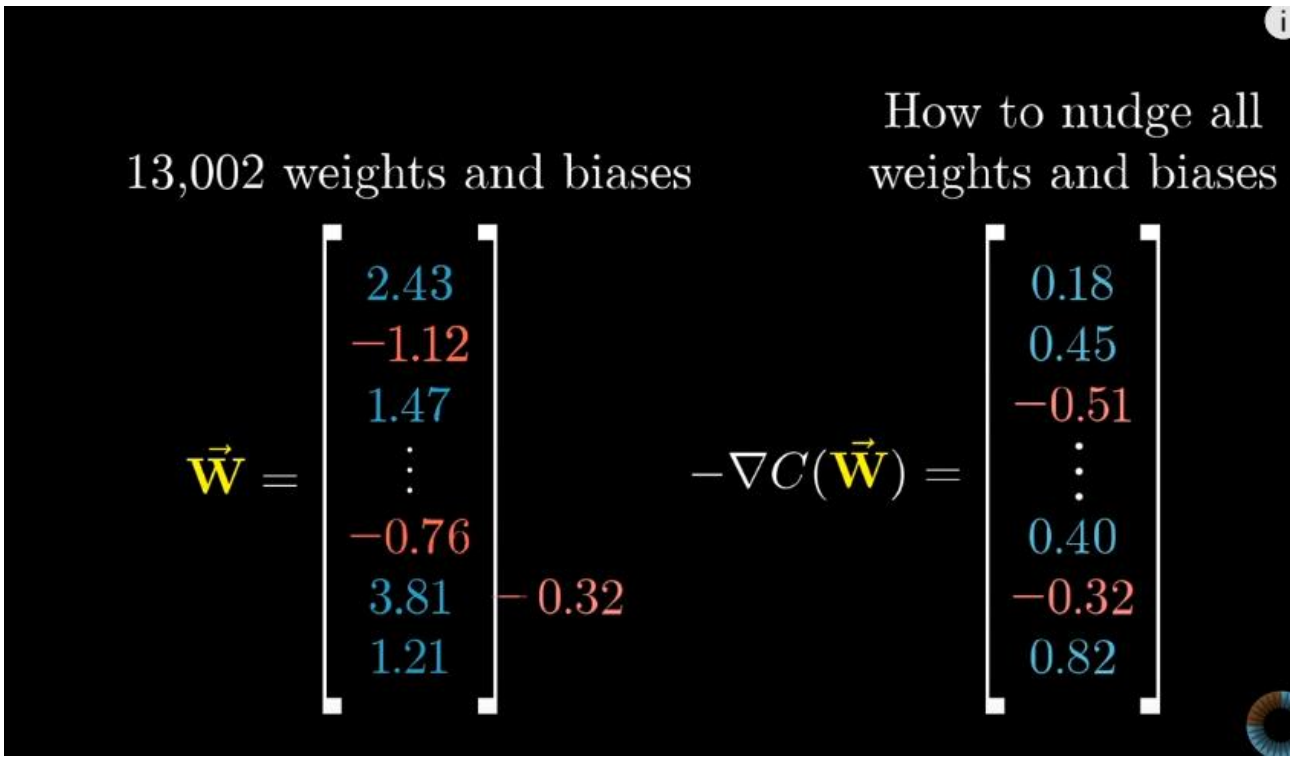
Gradient of fn gives direction of steepest ascent. Taking negative of that fn gives dirn of steepest descent. Length of gradient vector is measure of steepness of slope.



Algorithm for minimizing the fn:

Compute ∇C
Small step in $-\nabla C$ direction
Repeat.

The negative gradient of cost fn is a vector. It tells us which nudges to all those numbers is going to cause rapid decrease to cost Fn



This cost fn involves an avg over all training data so if we minimize it, it means better performance on all samples.

The algorithm for computing this gradient effectively which is effectively at heart of how a neural network learns is called **back propagation.**

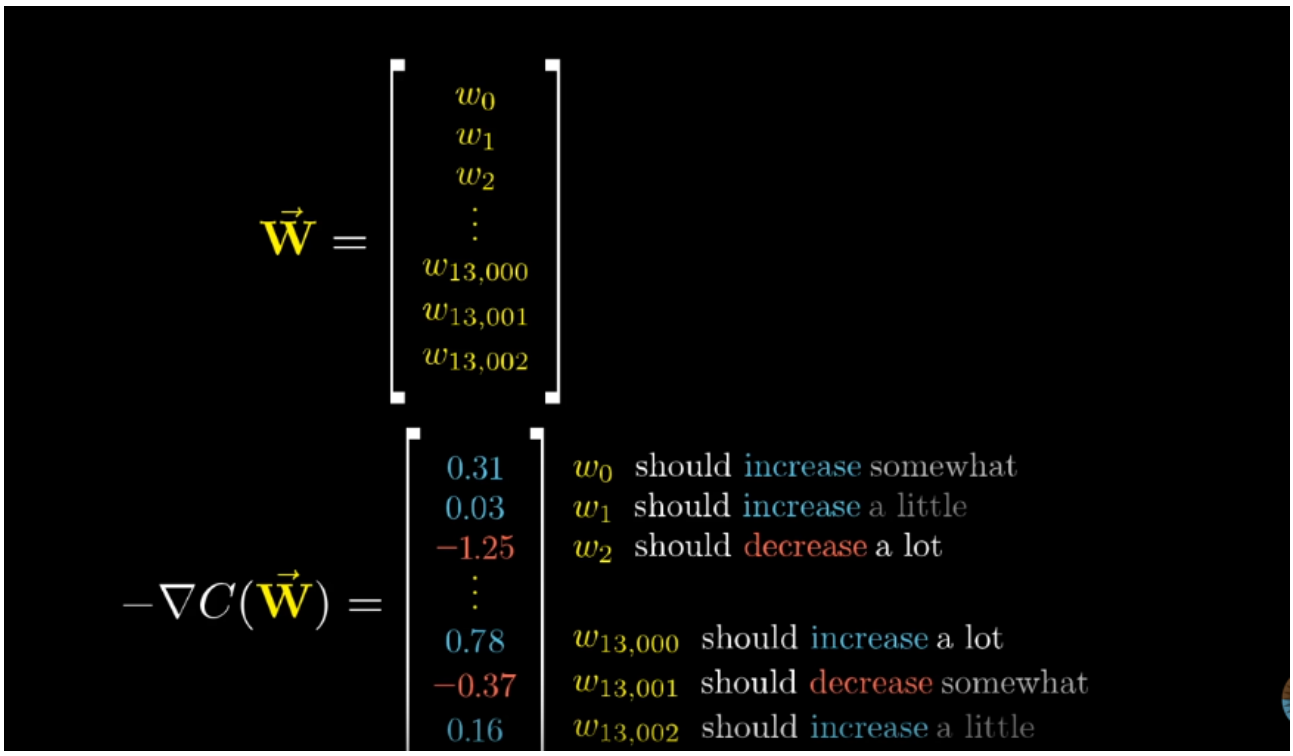
We minimize cost fn and cost fn should have a smooth output

This process of repeatedly nudging an input of fn by some multiple of the negative gradient is called gradient descent.

Each component of -ve gradient tells us 2 things:

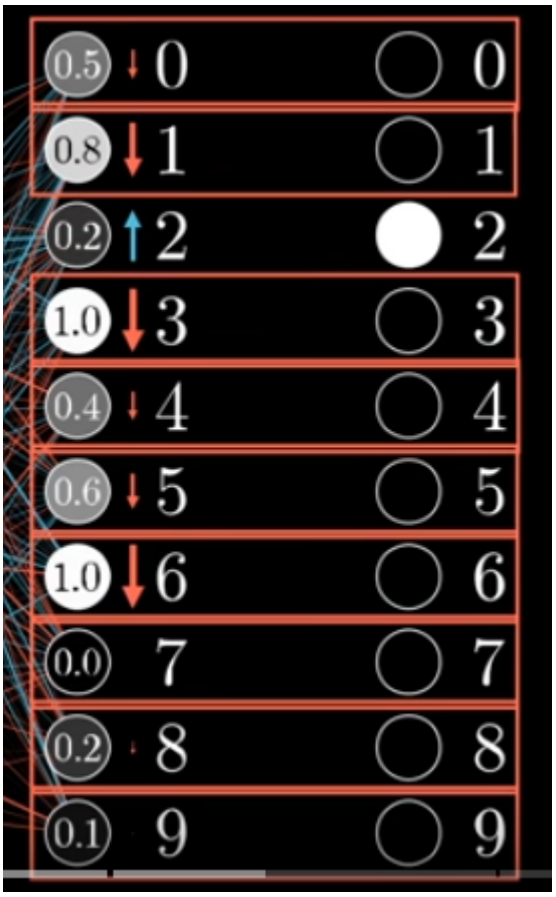
sign- whether corresponding component of input vector should be nudged up or down.

Relative magnitude of all components- which changes matters more



BACK PROPAGATION:

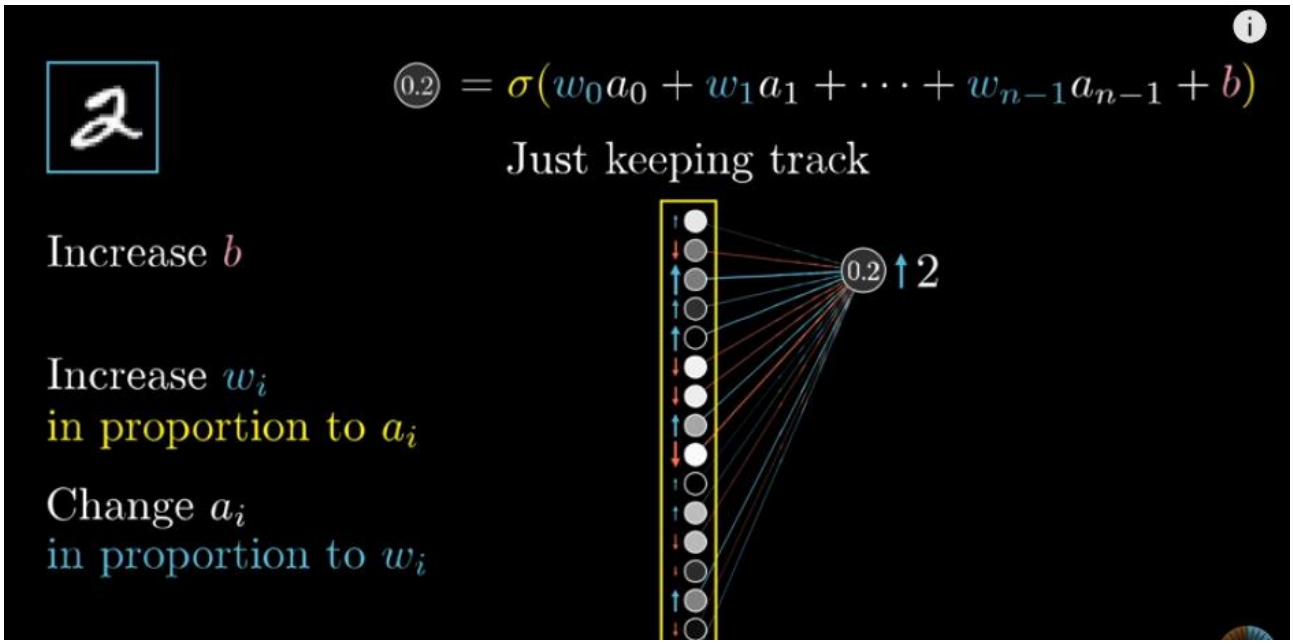
Example: recognize digit 2



Since we want to classify image as 2, we want the third value to nudge up and others nudged down.

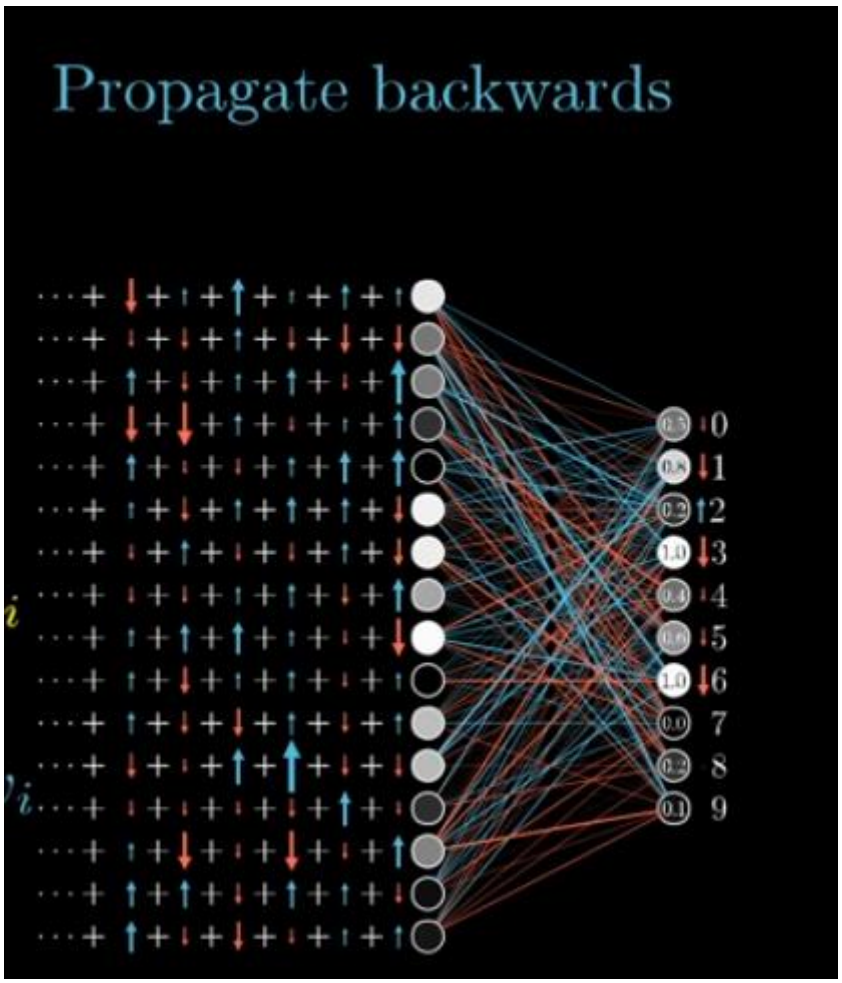
Size of nudges depends of how far the current value is from target value.

For increasing activation of 2, we can:



we simultaenously also want other neurons in last layer to become less active. So desire of digit 2 neuron is added together with desires of all output neurons.

This forms the basis of idea of



propagating backwards. By adding all these effects we get list of nudges that we want to happen in second-to-last-layer.

Then we can recursively apply those to relevant weights and biases that determine those values, repeating the same process and moving backwards through network.
