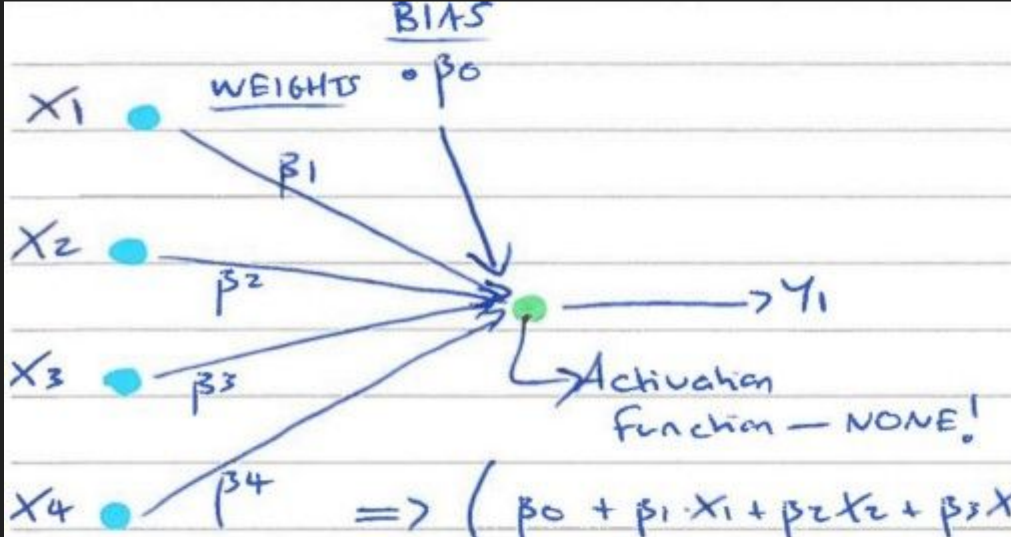
Neural Networks

(Or Multilayer Perceptrons)

Sources: Elements of Stat. Learning, Andrew Ng, 3blue1brown, and Google's Machine Learning Crash Course for developers

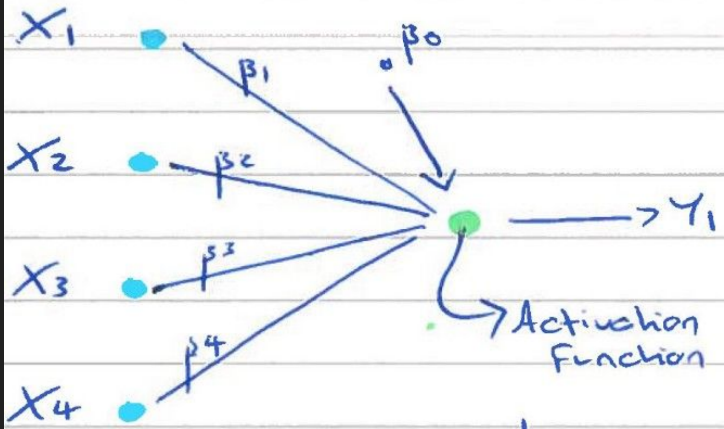
Linear Regression as a network:



$$\Rightarrow \left(\beta_0 + \beta_1 \cdot X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 \right) = y_1$$

Minimise Loss : $\frac{1}{2n} \sum_{x} \| y(x) - a(x) \|^2$

Logistic Regression as a network:



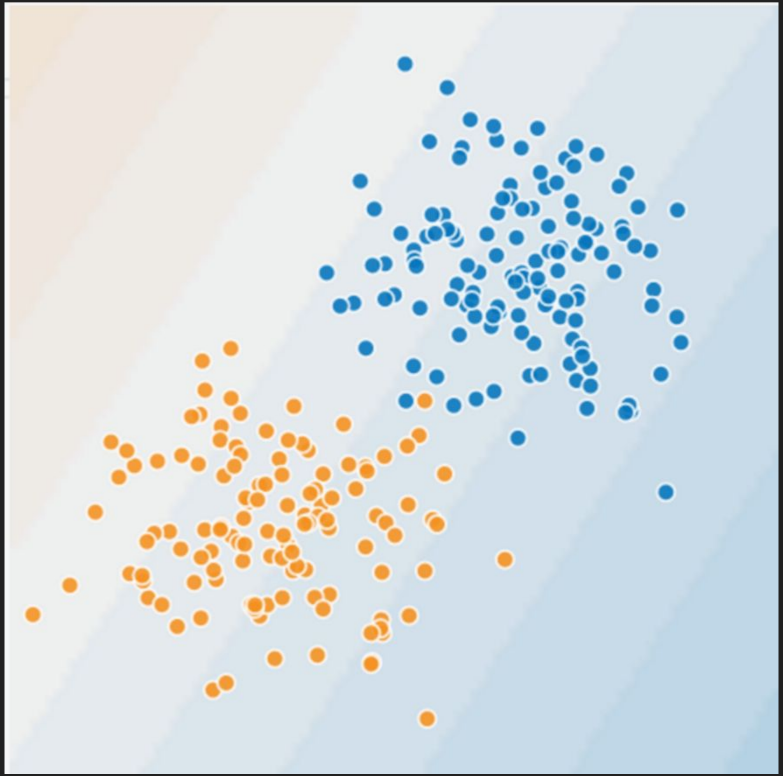
SIGMOID

$$— f(x) = \frac{1}{1+e^{-x}}$$

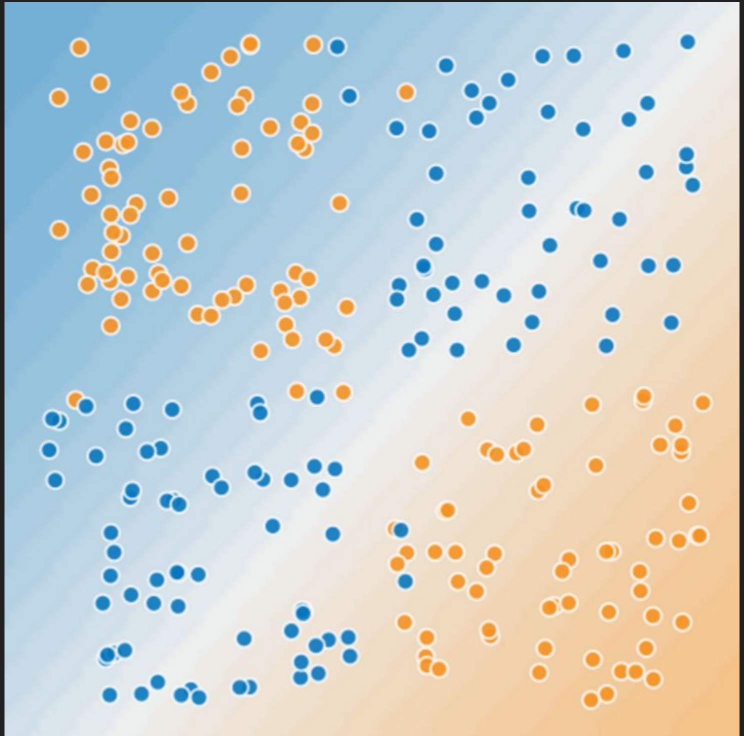
$$\Rightarrow \frac{1}{1+e^{-(\beta_0+\beta_1x_1+\beta_2x_2+\beta_3x_3+\beta_4x_4)}} = y'$$

Minimise Loss: $-\frac{1}{n}\sum_{x} y(x)\cdot\ln\left(a(x)\right) + \left(1-y(x)\right)\cdot\ln\left(1-a(x)\right)$

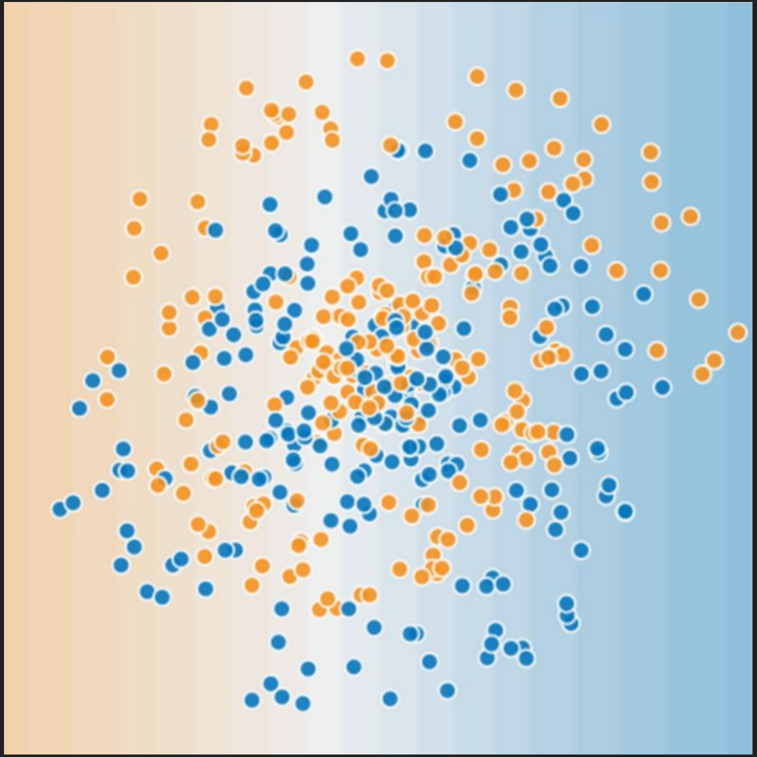
Linear models can fit a line to data like this:

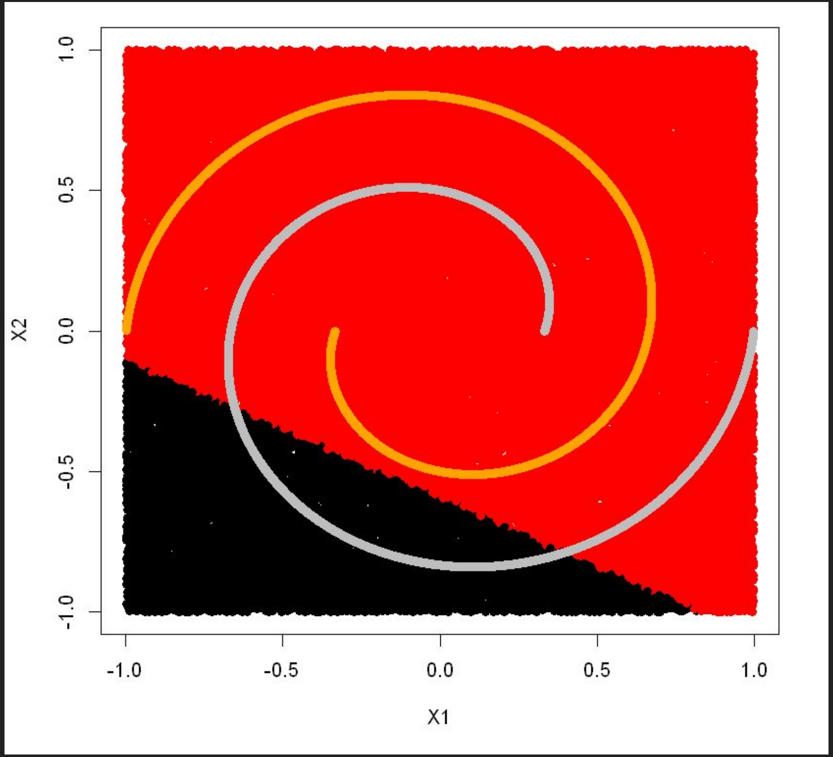


Linear models can even use creative interaction terms that fit a line to data like this:

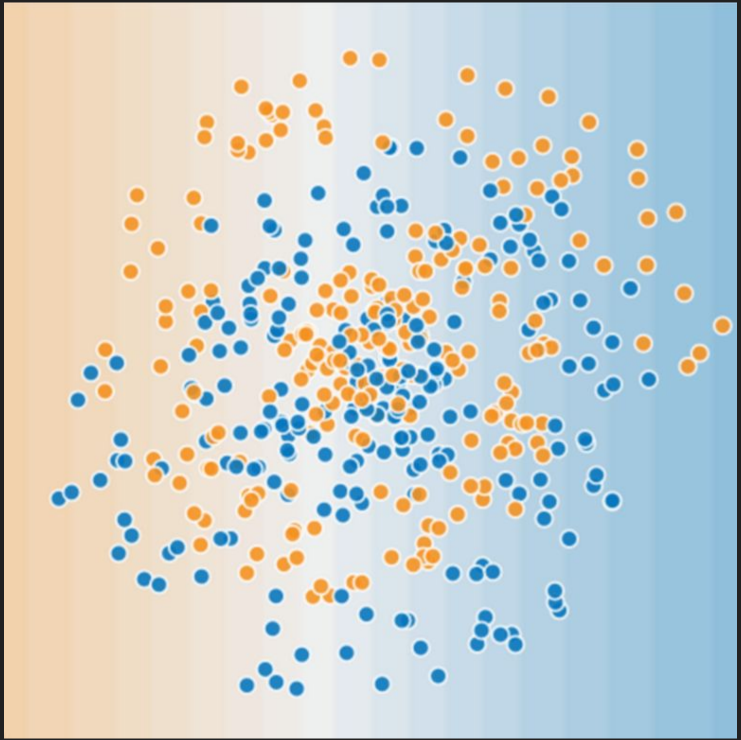


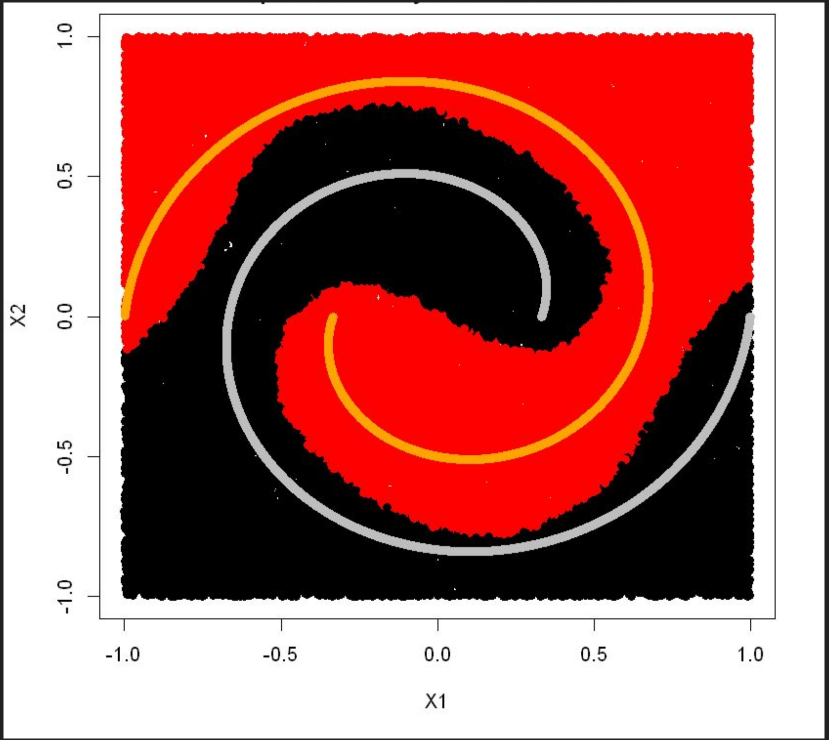
Linear models cannot, successfully fit a line to data like this:





Neural networks, however, can be trained to find hidden nonlinear relationships in complex data:





Advantages and Disadvantages of Neural Network Models:

Also known as a Multilayer Perceptron Models:

Advantages:

Capability to learn non-linear models.

Can find relationships other models may miss

Disadvantages:

Random weight initializations can lead to different validation accuracy.

Requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.

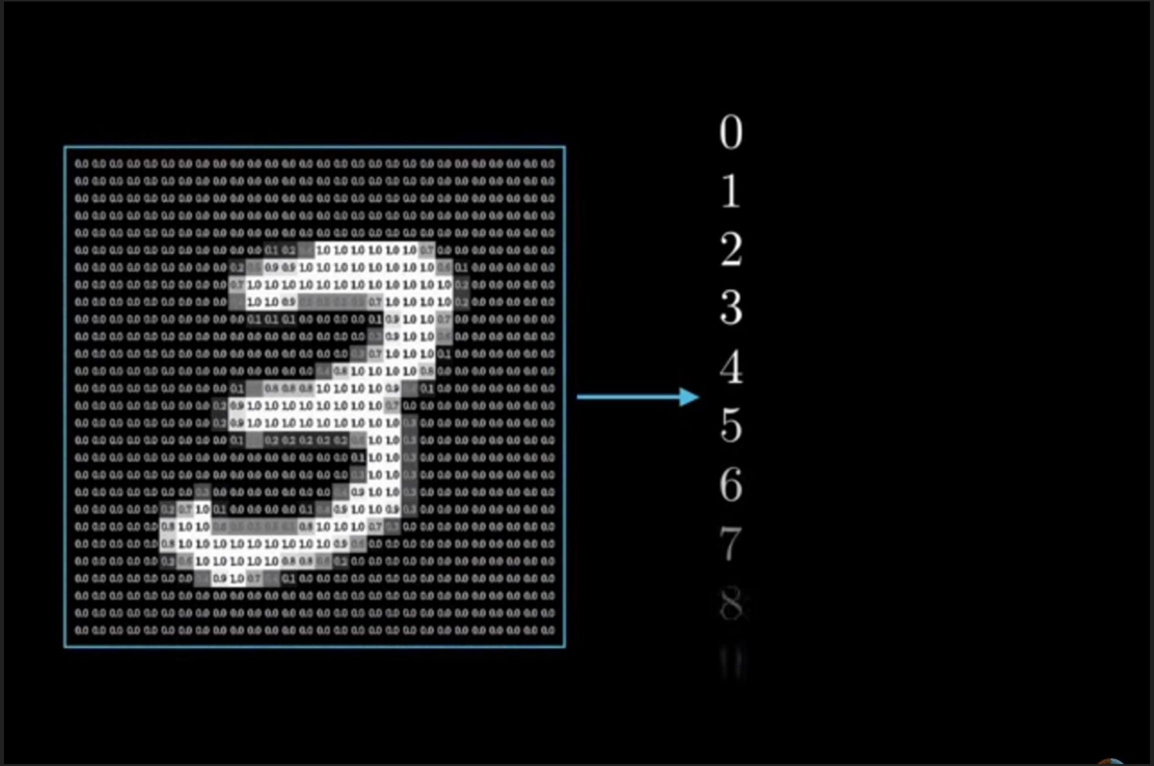
Also sensitive to feature scaling.

What is a Neural Network?

Let’s start by understanding the structure of a Neural Network through an example.

Example: Classification of handwritten digits

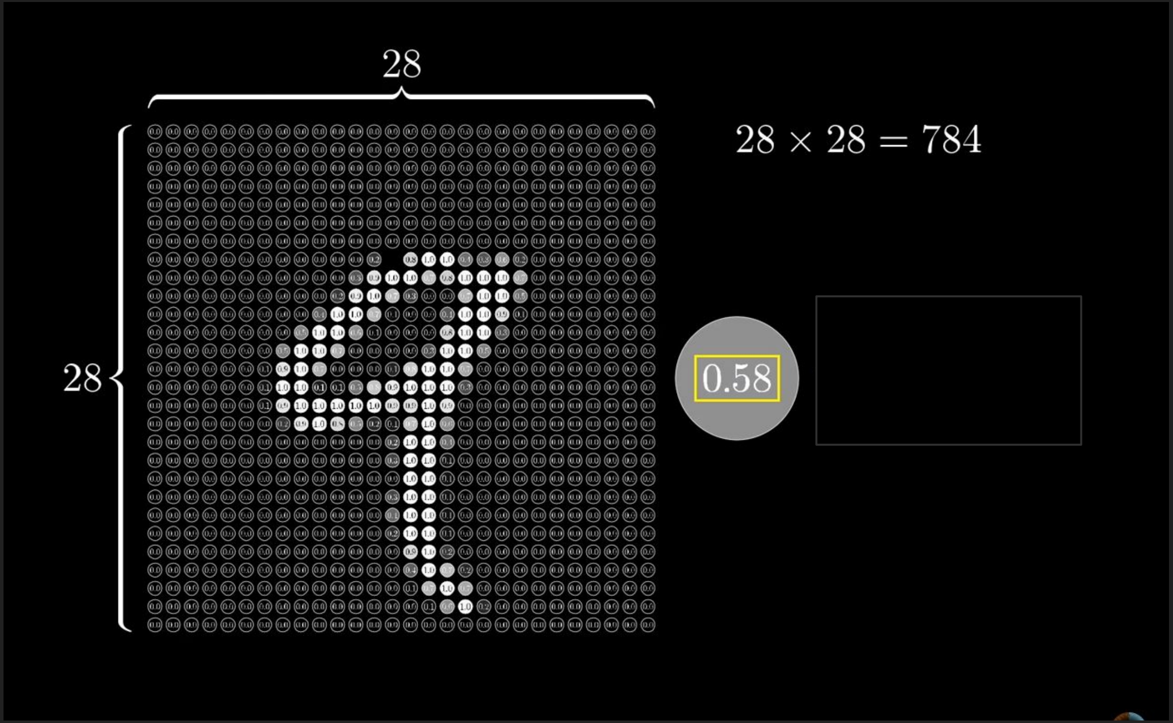
We want to take the 28 by 28 pixel data and classify it into a digit



We want to take the 28 by 28 pixel data and classify it into a digit

Each pixel data

Between 0 and 1



We want to take the 28 by 28 pixel data and classify it into a digit

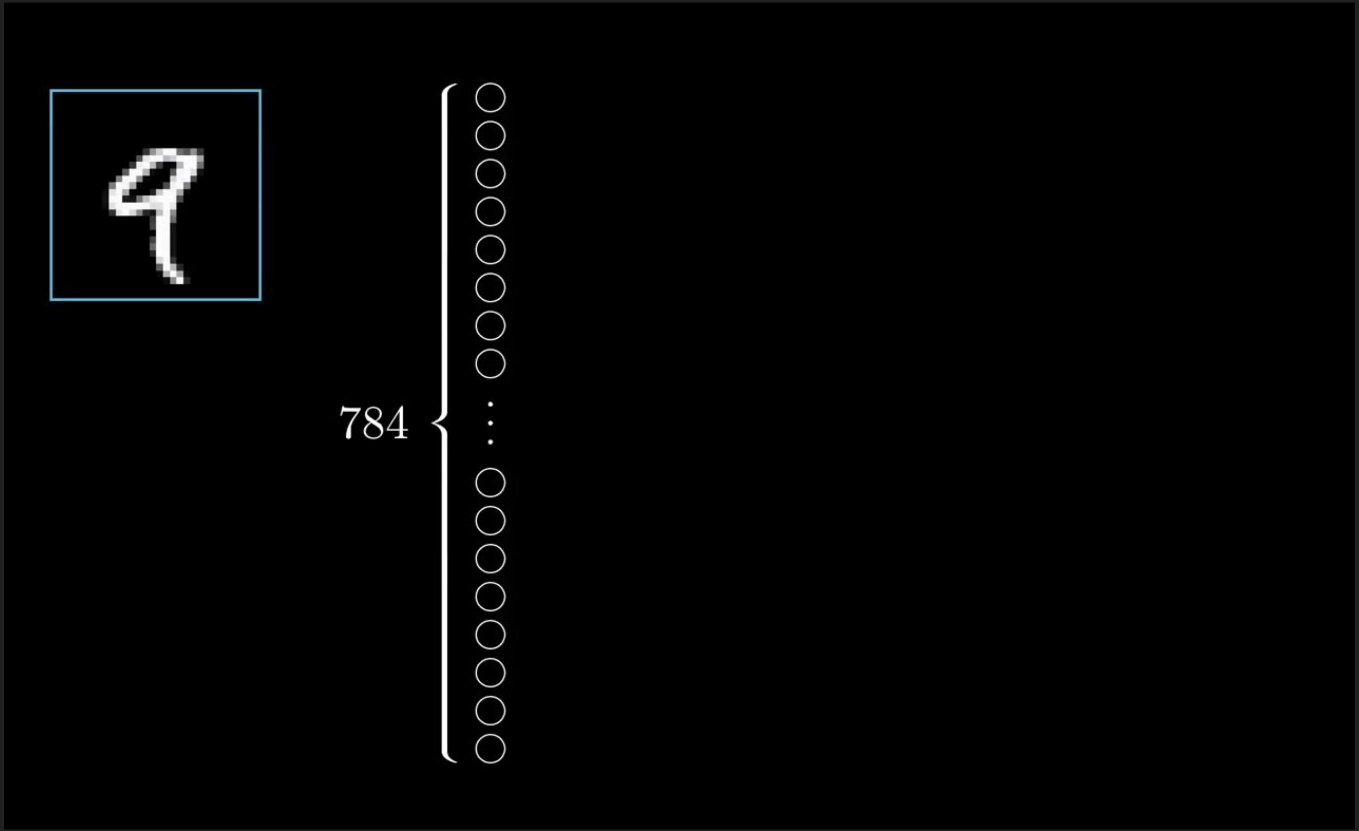
Each cell of input

Data becomes

A “neuron” in

Layer one of the

network



Building a Neural Network

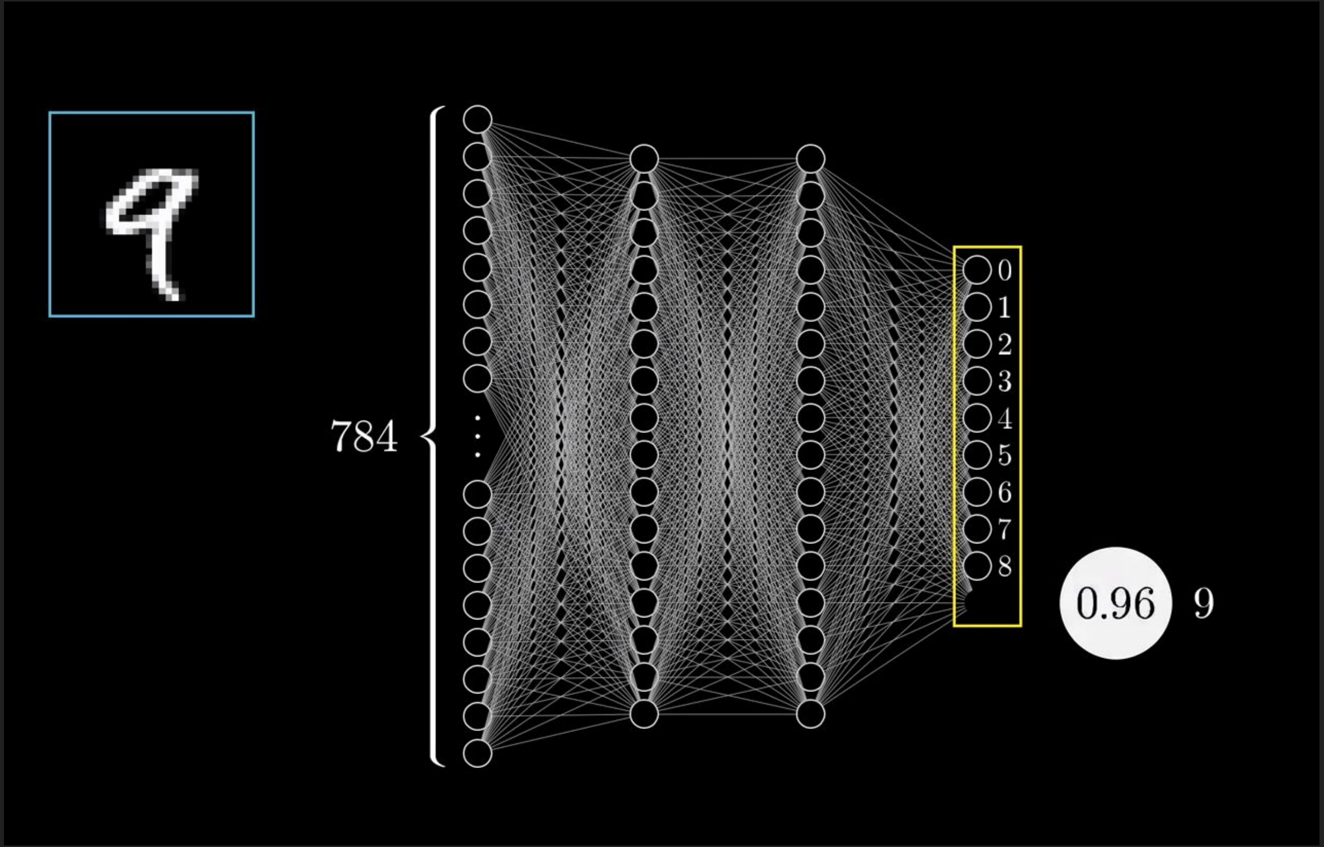
Goal: Build a network

That classifies

Data into correct

0-9 digits in our

output



Building a Neural Network

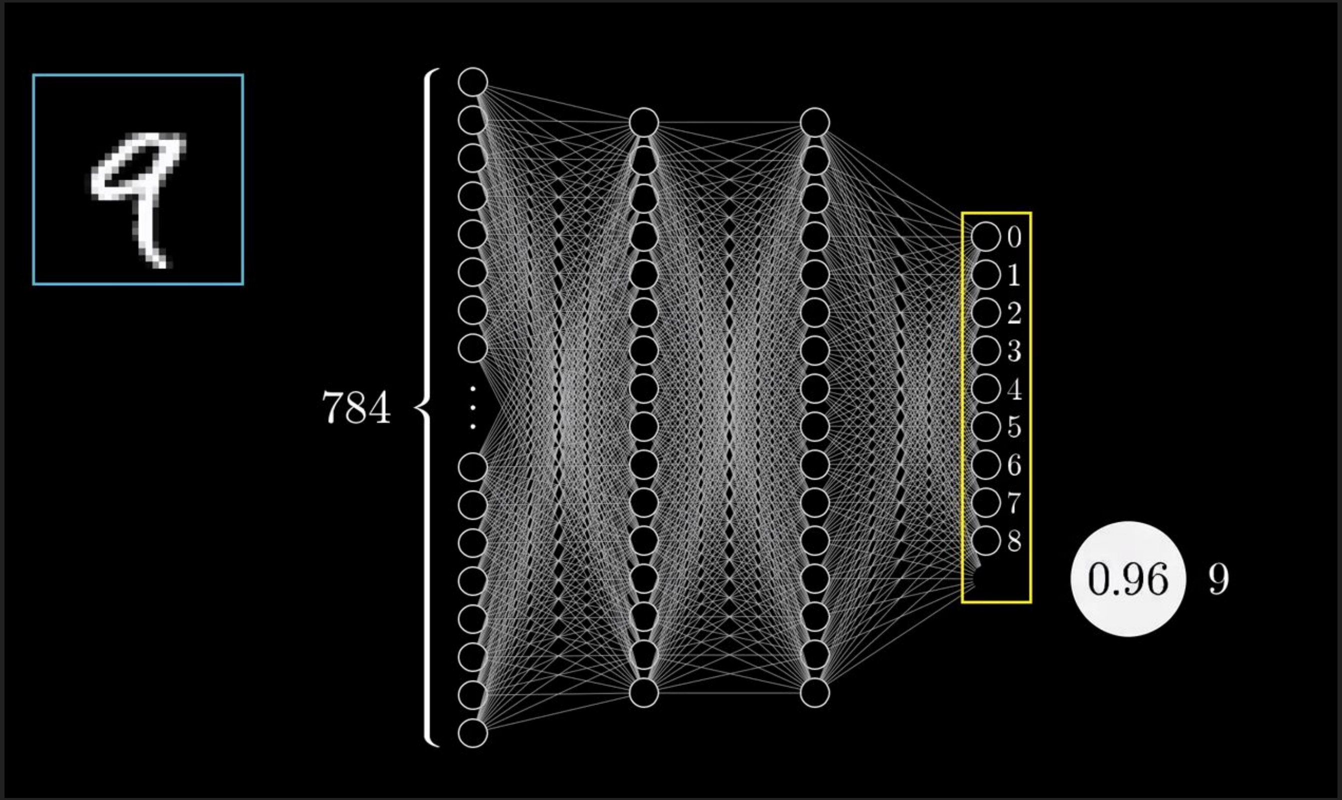
Inputs: X data

Outputs:

Neurons that are

“Activated” to signify

A prediction value



Building a Neural Network

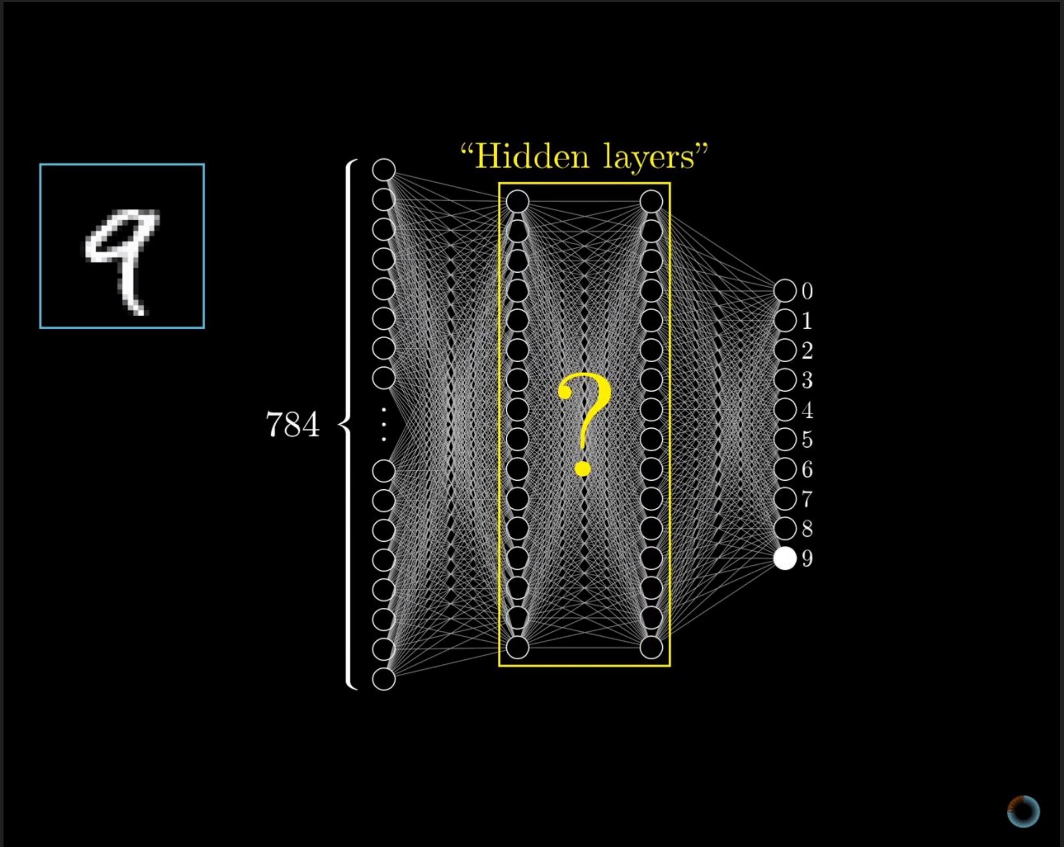
We will also add

Two hidden layers.

For now, let's wait

To discuss what these

Layers are.



Building a Neural Network

For now, note that

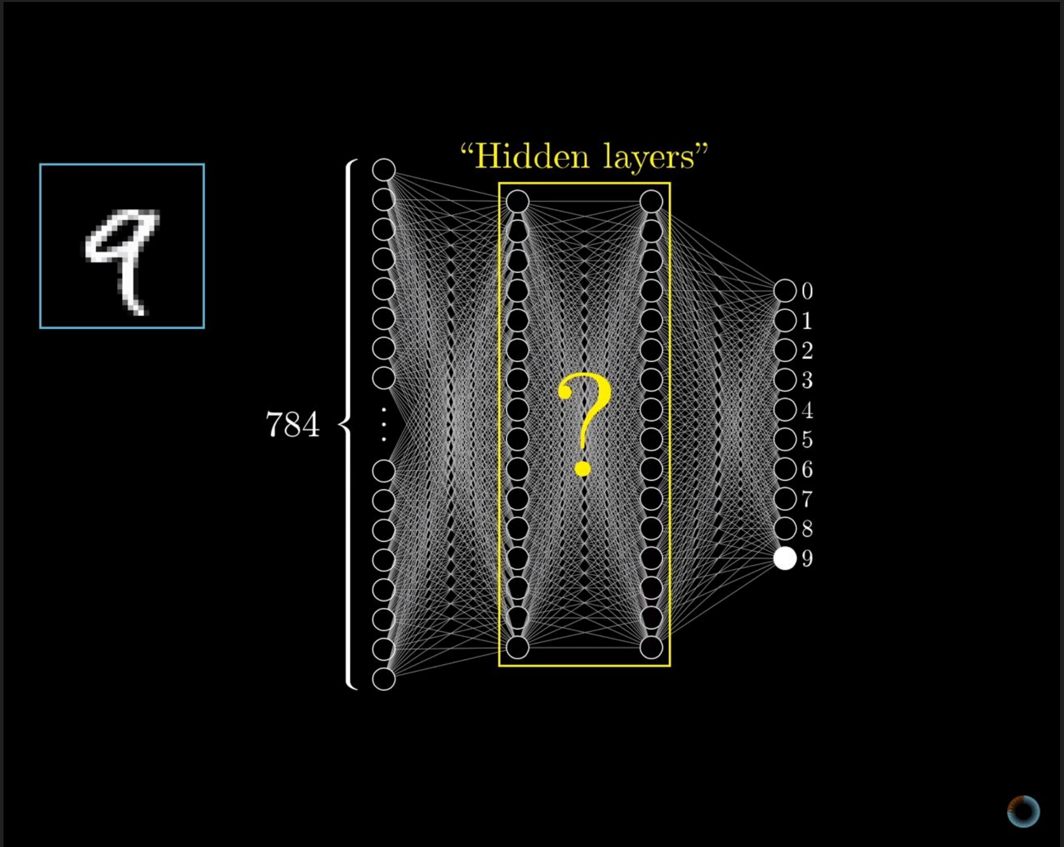
We are choosing

Two hidden layers.

Both have 16 neurons.

Why?

Arbitrarily chosen.



An aside: What are deep learning models?

Plain Vanilla Network Models vs. Deep Learning Models:

What’s the difference between a neural network model and a deep learning network model?

The number of hidden layers.

When we use more than one hidden layer we are doing deep learning.

Thinking through structure of Neural Network

Input data >>

Activates second layer

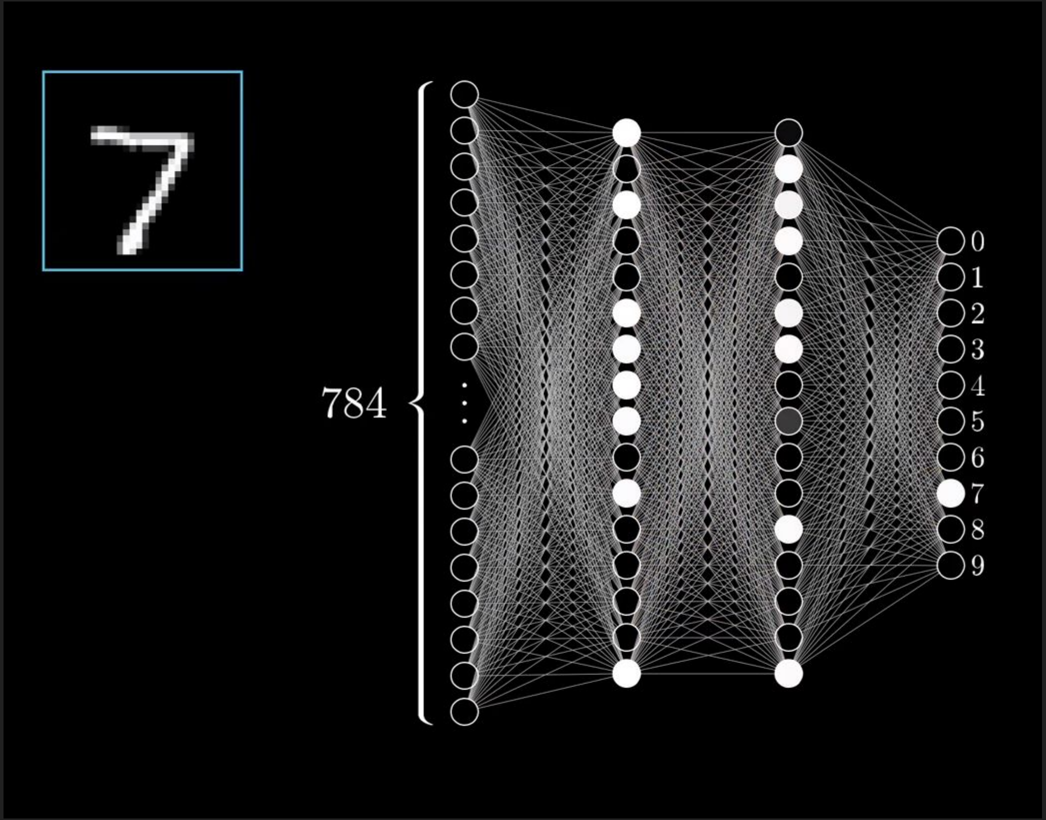
Nodes (neurons) >>

Activation of second layer

Nodes >> activates third layer

>> activation of third layer

>>activates final output node



Thinking through structure of Neural Network

One group of neurons/nodes

Firing causes another group

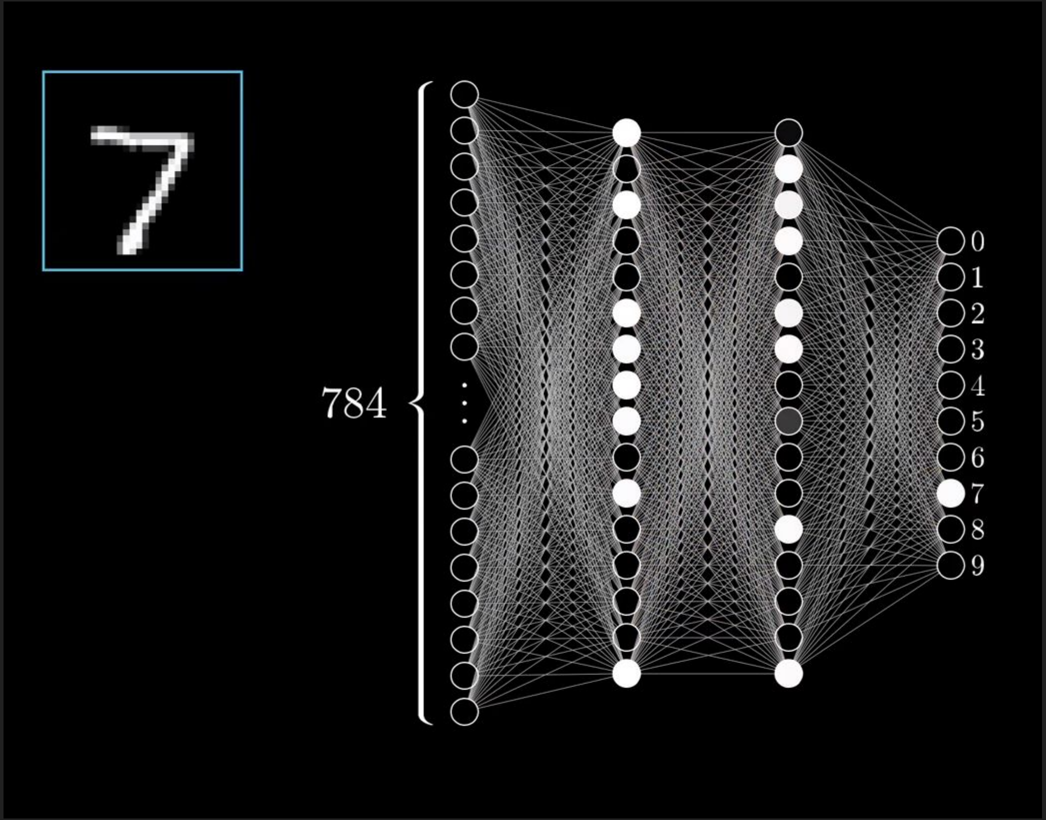
To fire.

New input data leads to new

Prediction.

Like an artificial brain trained

To a particular task.



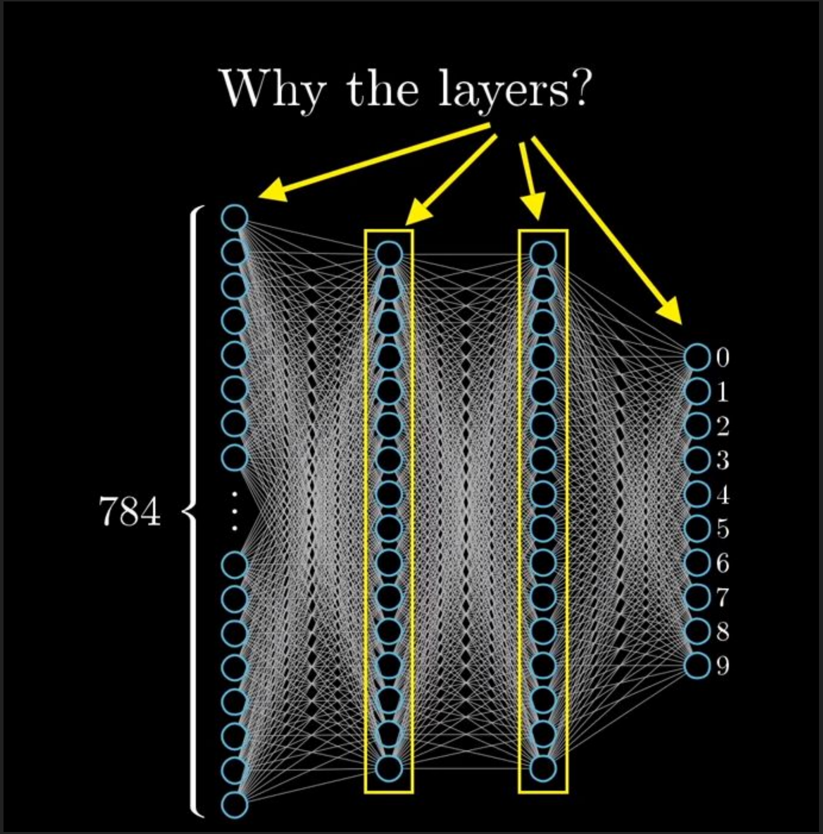
Thinking through structure of Neural Network

Goal:

Break down input data

To find hidden relat'ships

Among features.



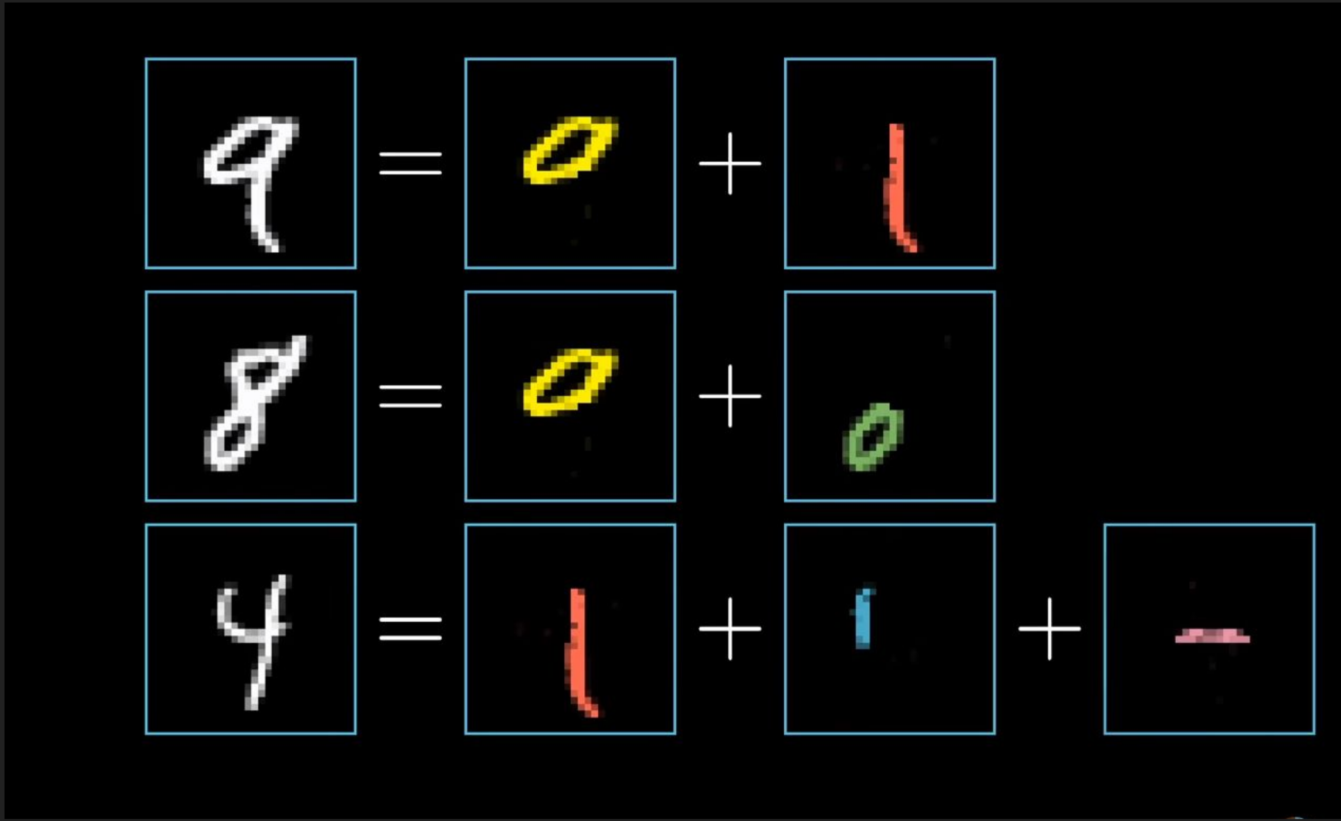
Thinking through structure of Neural Network

Goal:

Break down input data

To find hidden relat’ships

Among features.



Could be

That final hidden

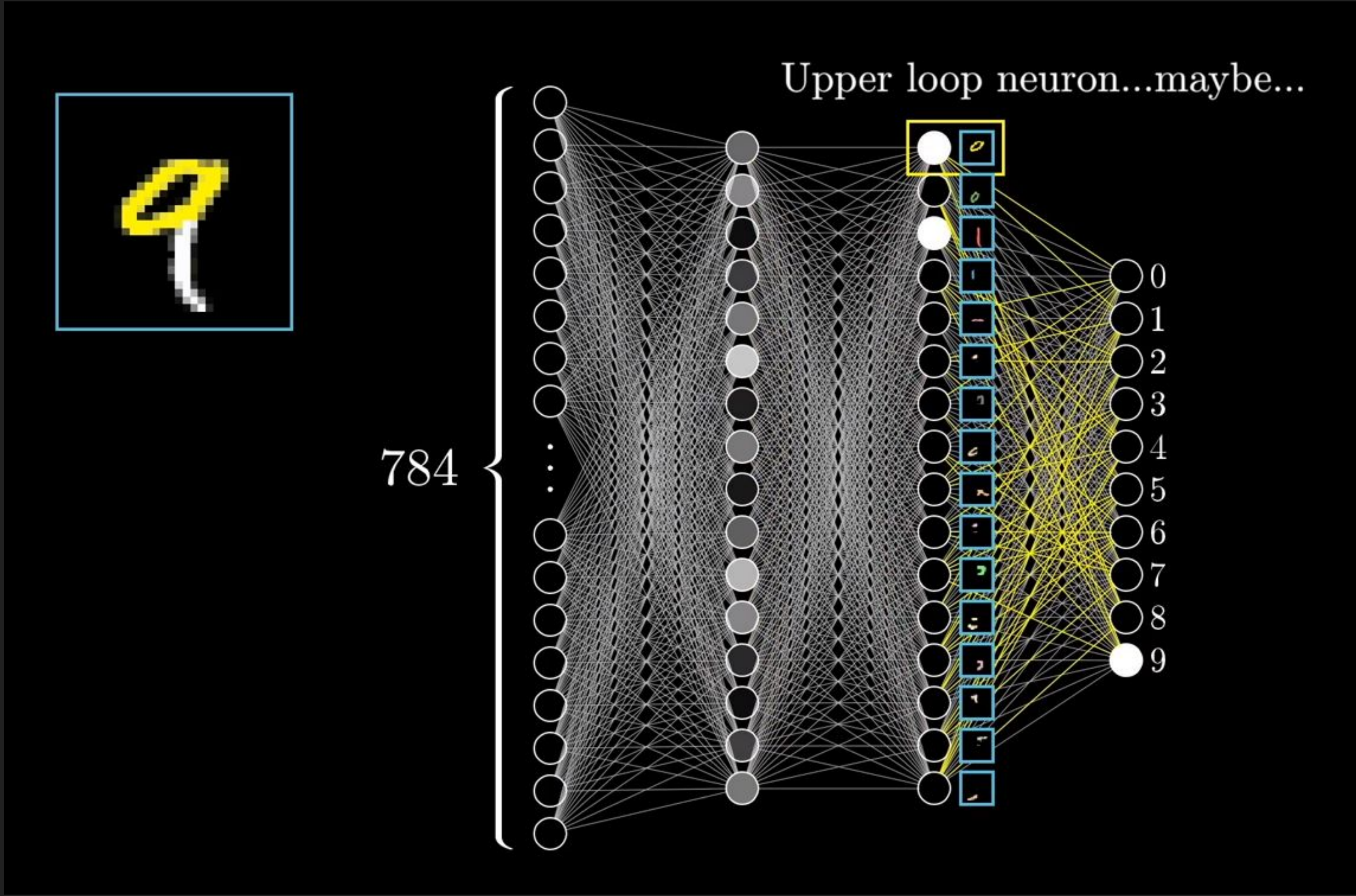
Layer

Activations

Break shape of

Number into

Useful pieces:



One sub-

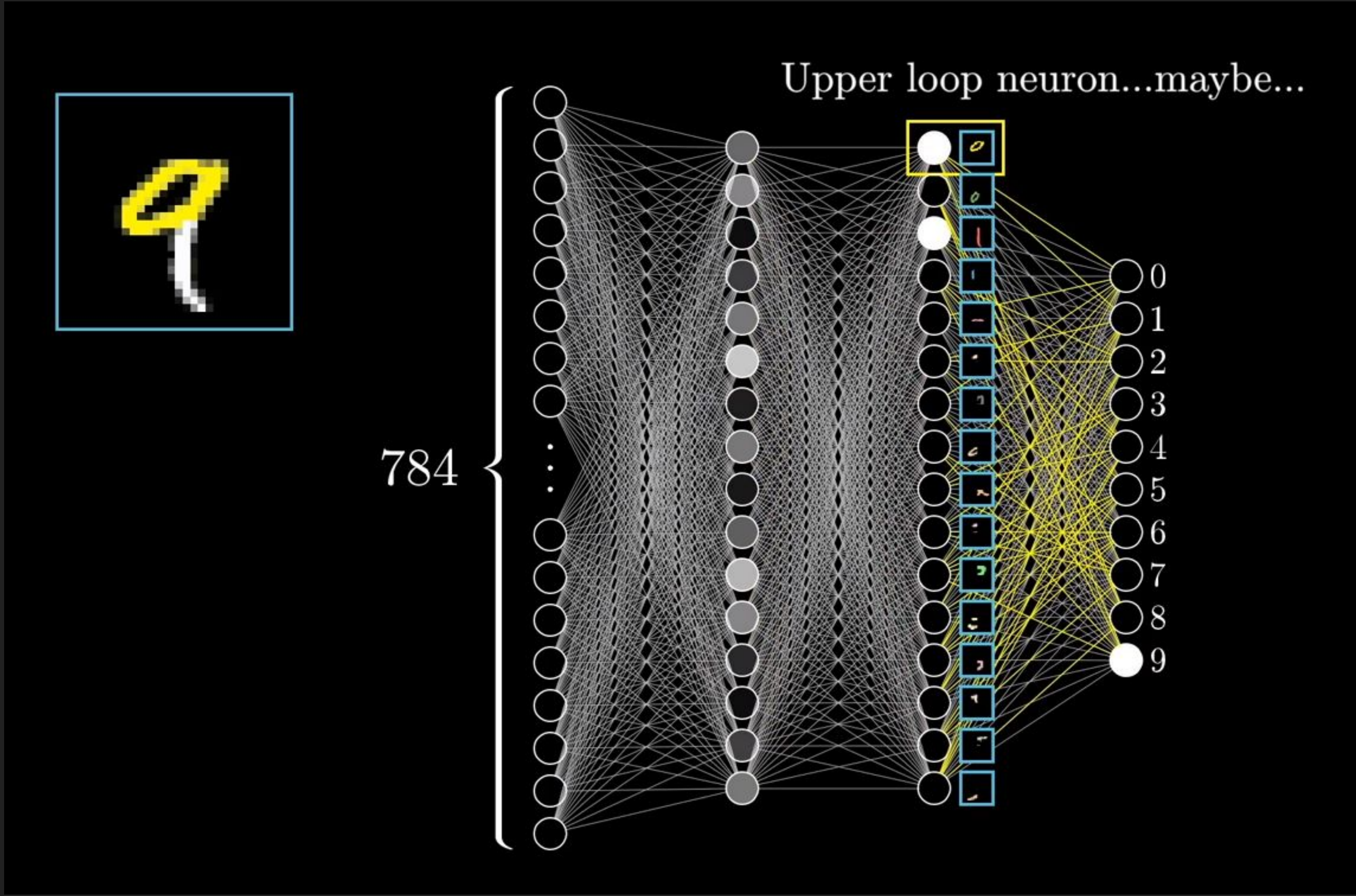
Part of shape

Of digit could

Be a loop

Another a

line.



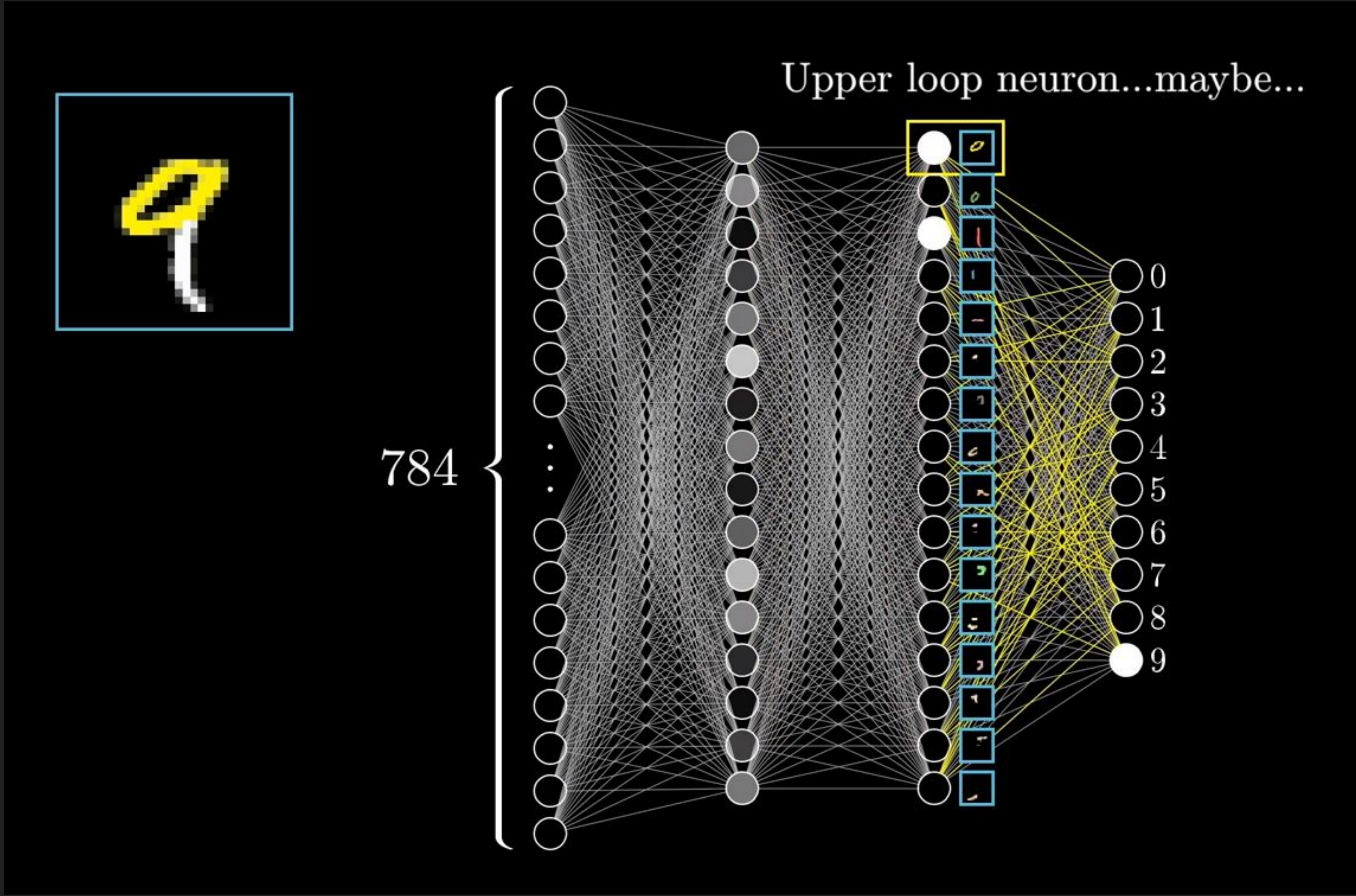
Sub-parts

Themselves

Could be

Broken into

Further pieces...

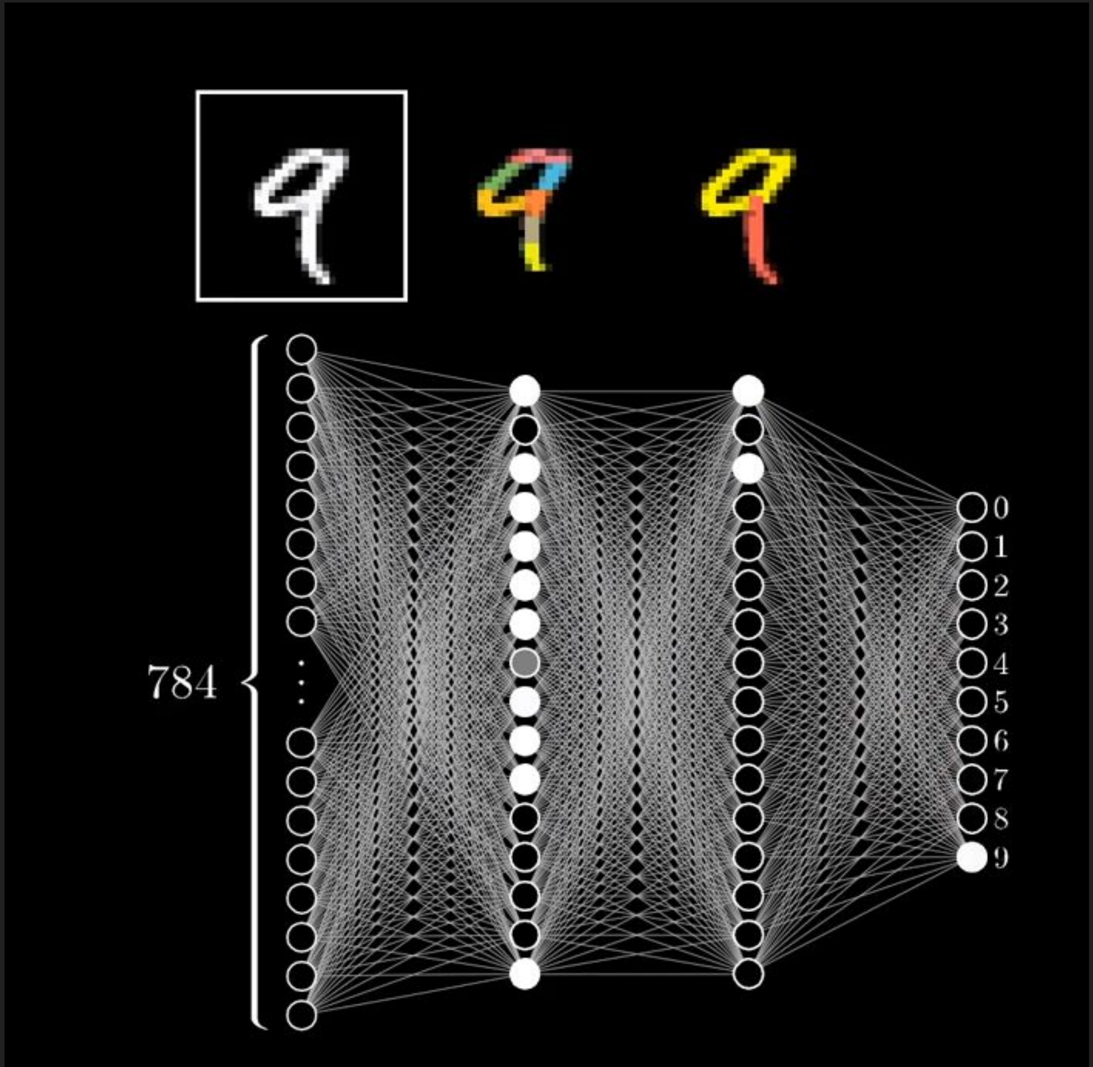


But how do we train

A network to successfully

Reveal these hidden

patterns?



Let’s walk through the main parts of a network:

We will discuss how inputs are weighted, summed up, and adjusted for bias…

And how we convert these results to a single value to activate each node.

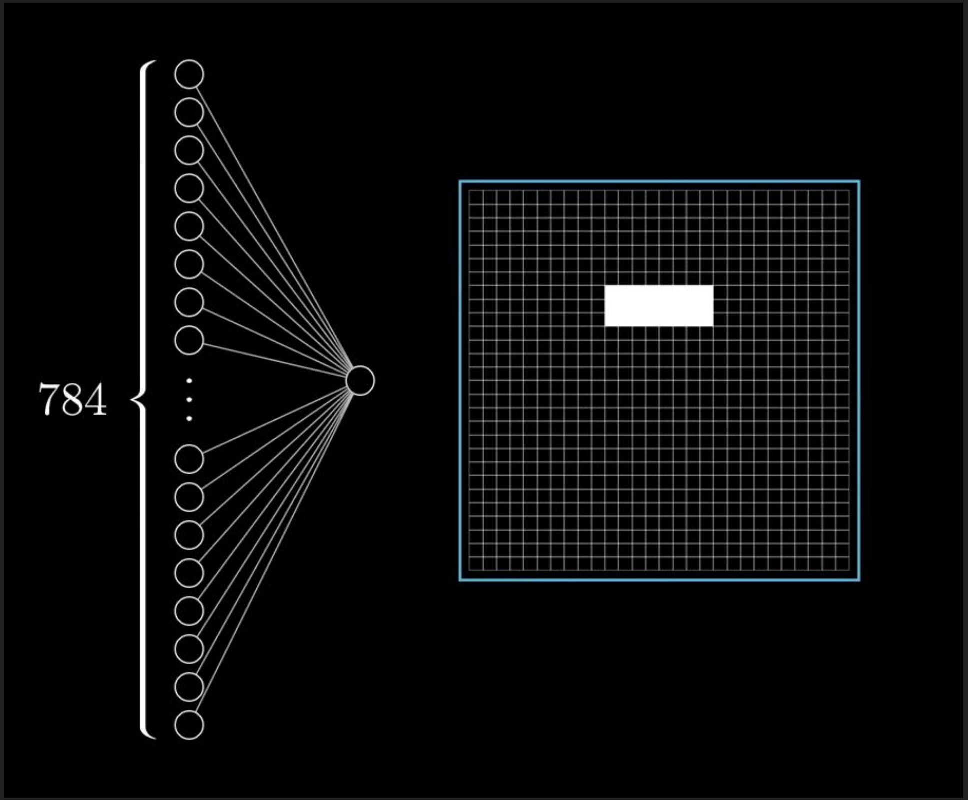
How do we calculate the “activation” value of a node in second “hidden” layer?

Let’s Focus on an input layer and

a single Node from first hidden layer.

--The node will signify the sub-

Pattern in the data on the right.--



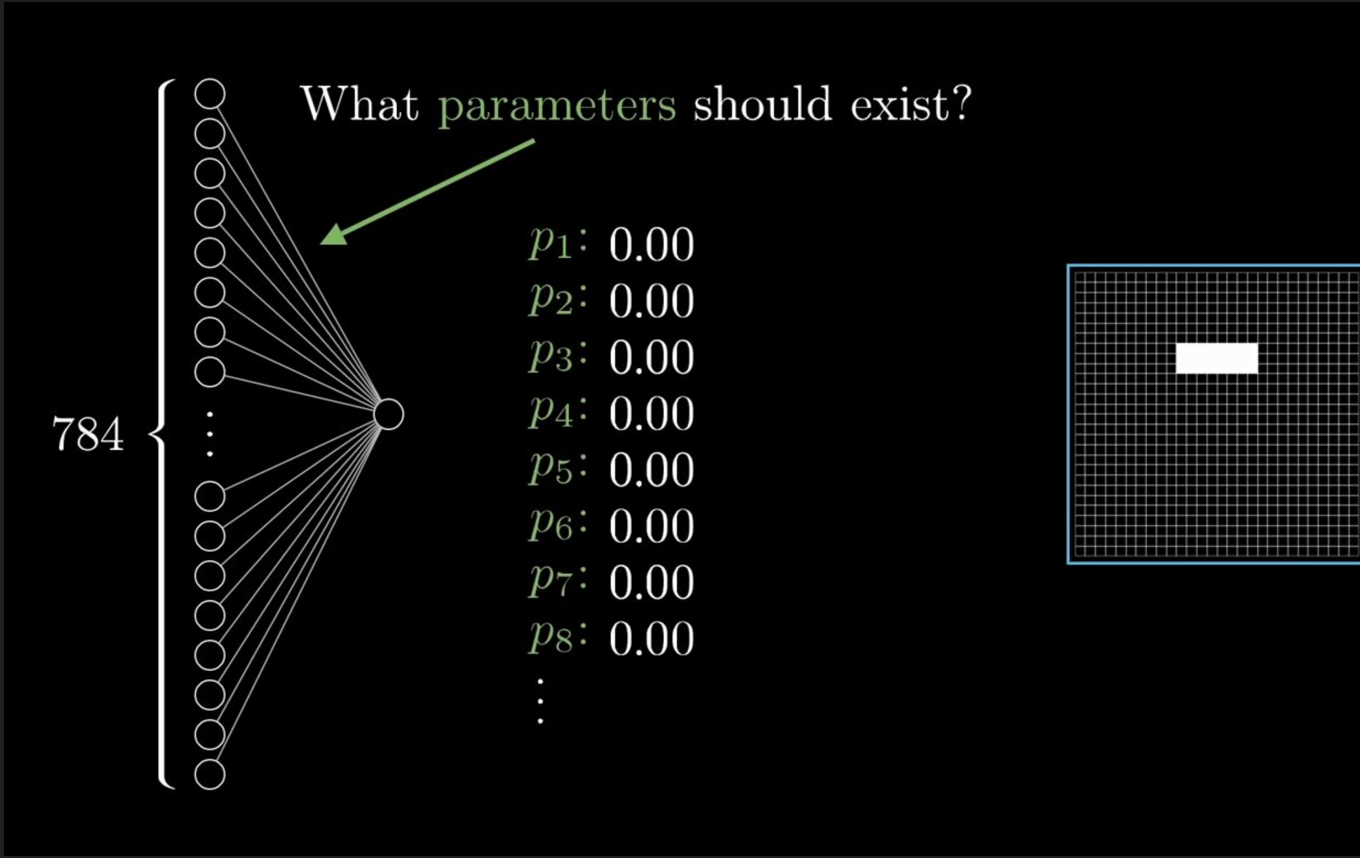
What parameters should we select to generate this pattern?

Would need to

Convert Input data such

That it activates this

Pattern in the node



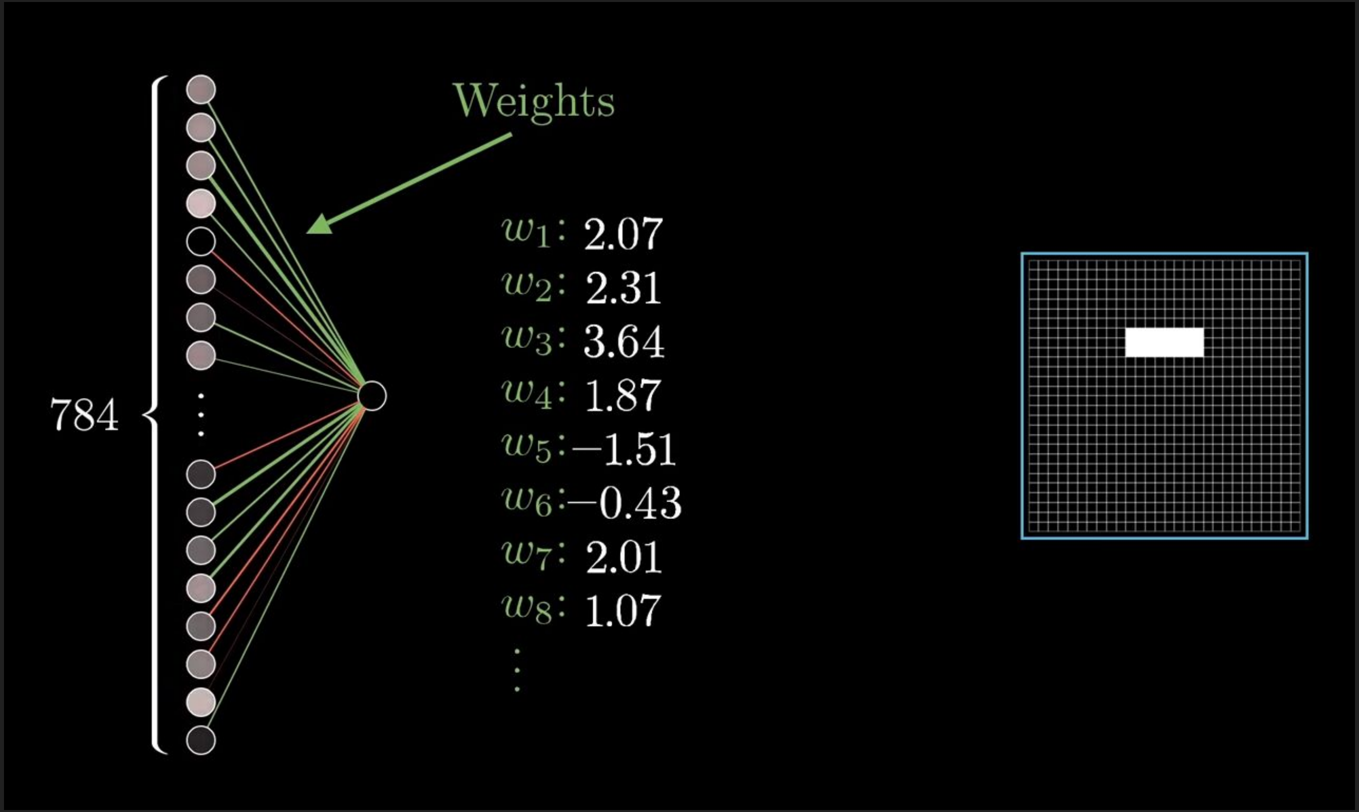
Need to assign correct weights to each input:

We can do

So by weighting

Each input value

accordingly



Next, we multiply these weights by input values, then we add results together:

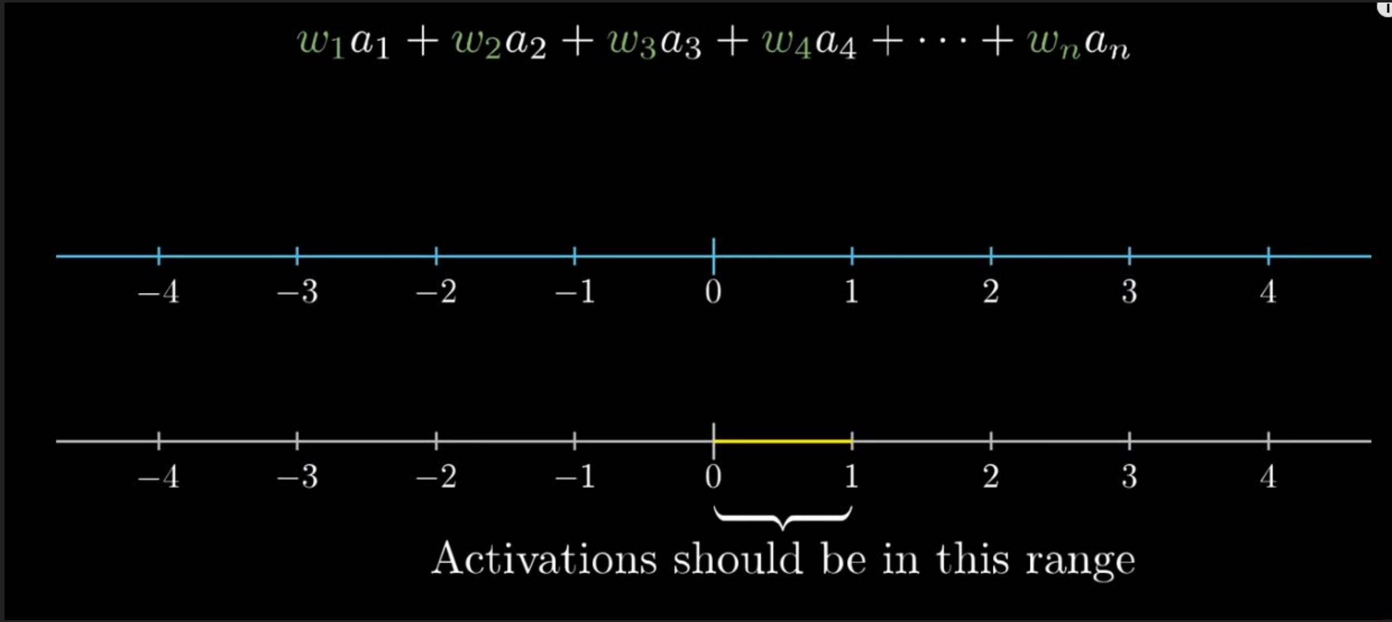
Lastly,

We convert summed

Results to “activation”

Value between 0

And 1.


$w_1a_1 + w_2a_2 + w_3a_3 + w_4a_4 + \cdots + w_na_n$
−4
−3
−2
−1
0
1
2
3
4
Activations should be in this range

Use sigmoid function to convert to single 0 to 1 number:

1 indicates more

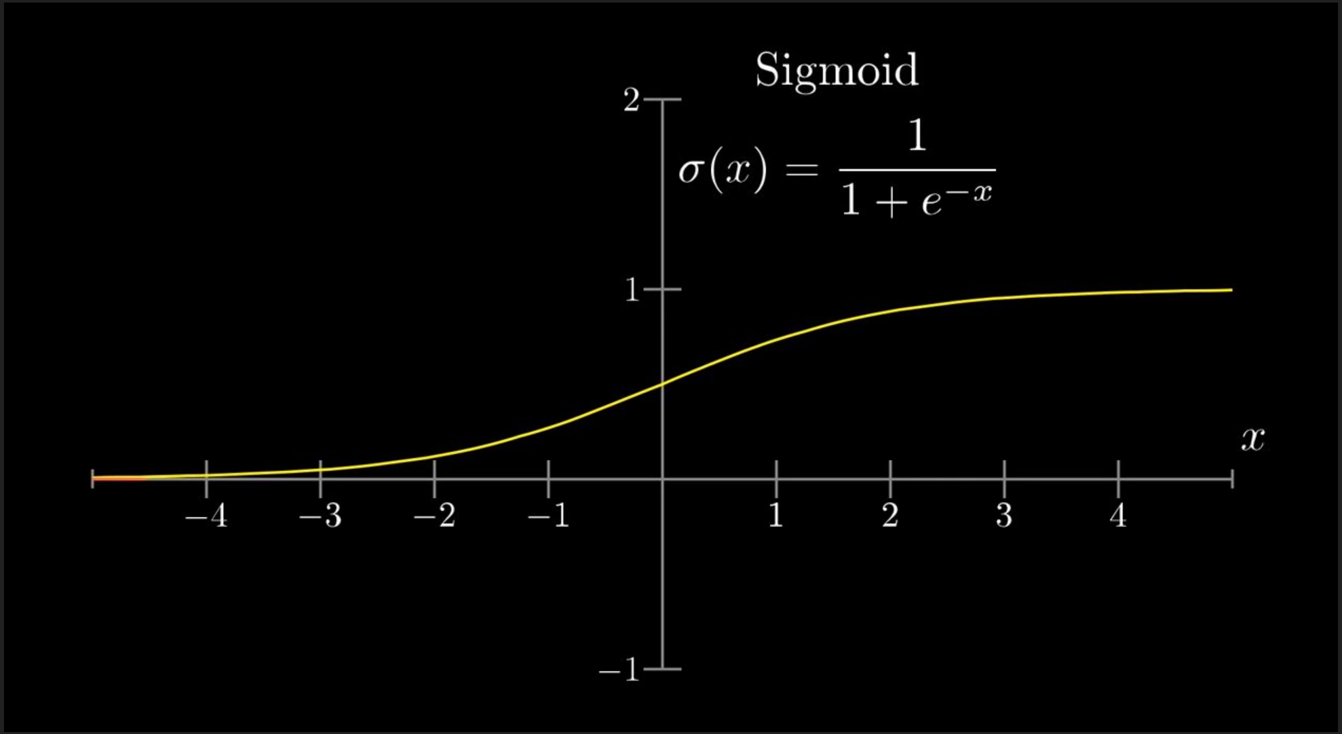
Activation

0 indicates less

(another popular

Transformation is a

Relu)



One last missing piece for calculation: Bias

We also include

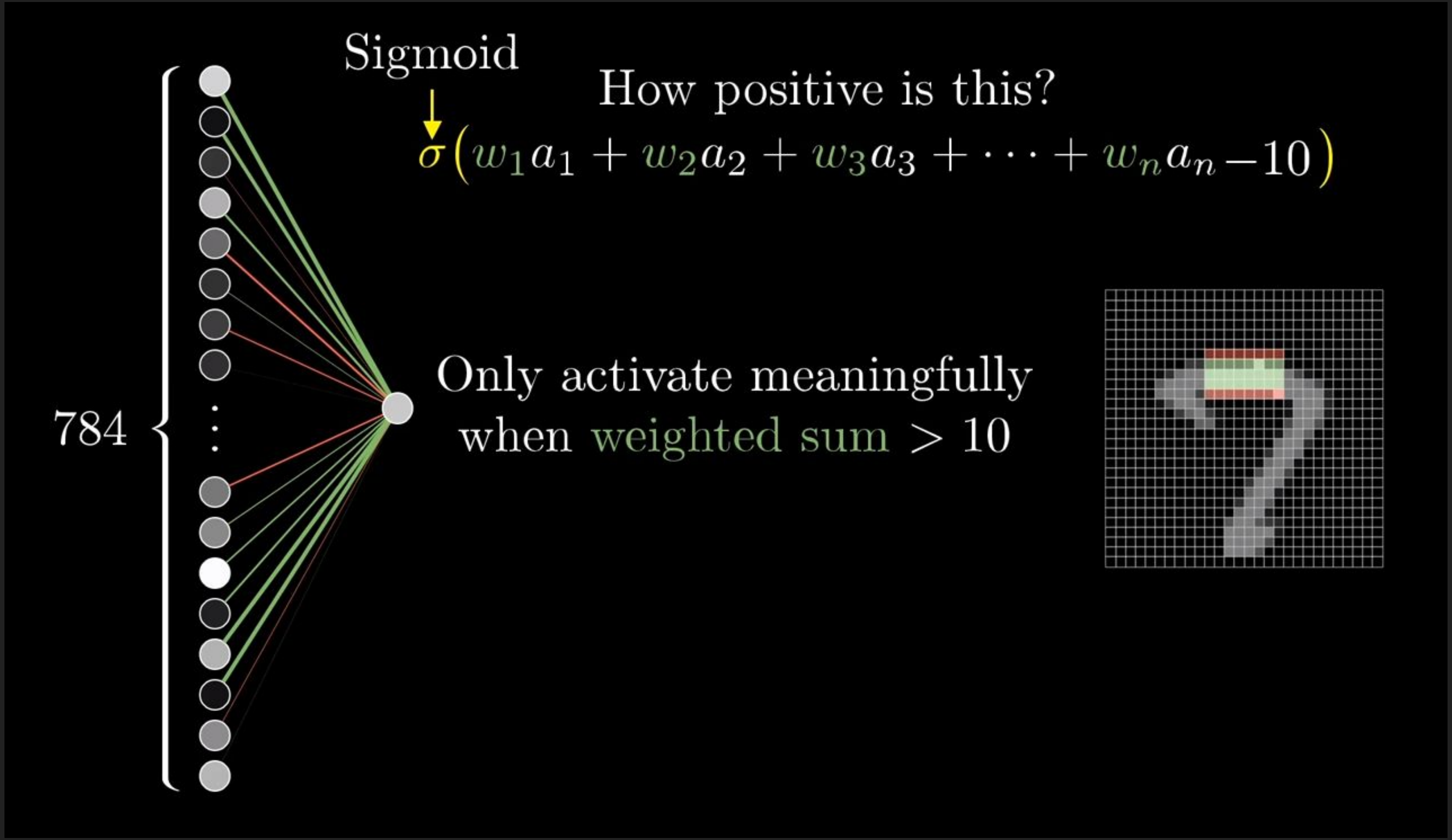
A term that

Adjusts activation

Called a bias

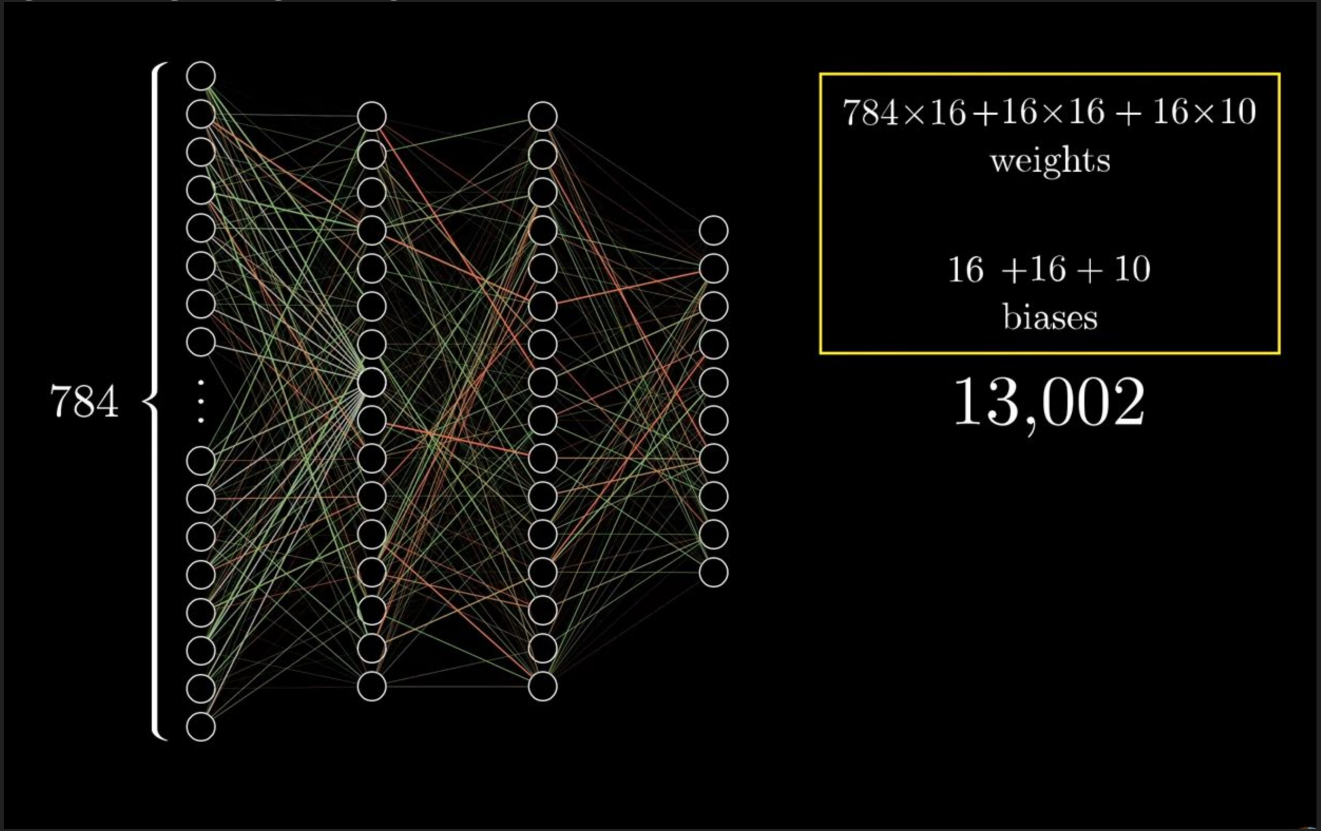
Before sigmoid

transformation


Sigmoid
How positive is this?
$\sigma(w_1a_1 + w_2a_2 + w_3a_3 + \cdots + w_na_n - 10)$
784
Only activate meaningfully when weighted sum > 10

Then we do the same thing for each node in the second layer of the network:

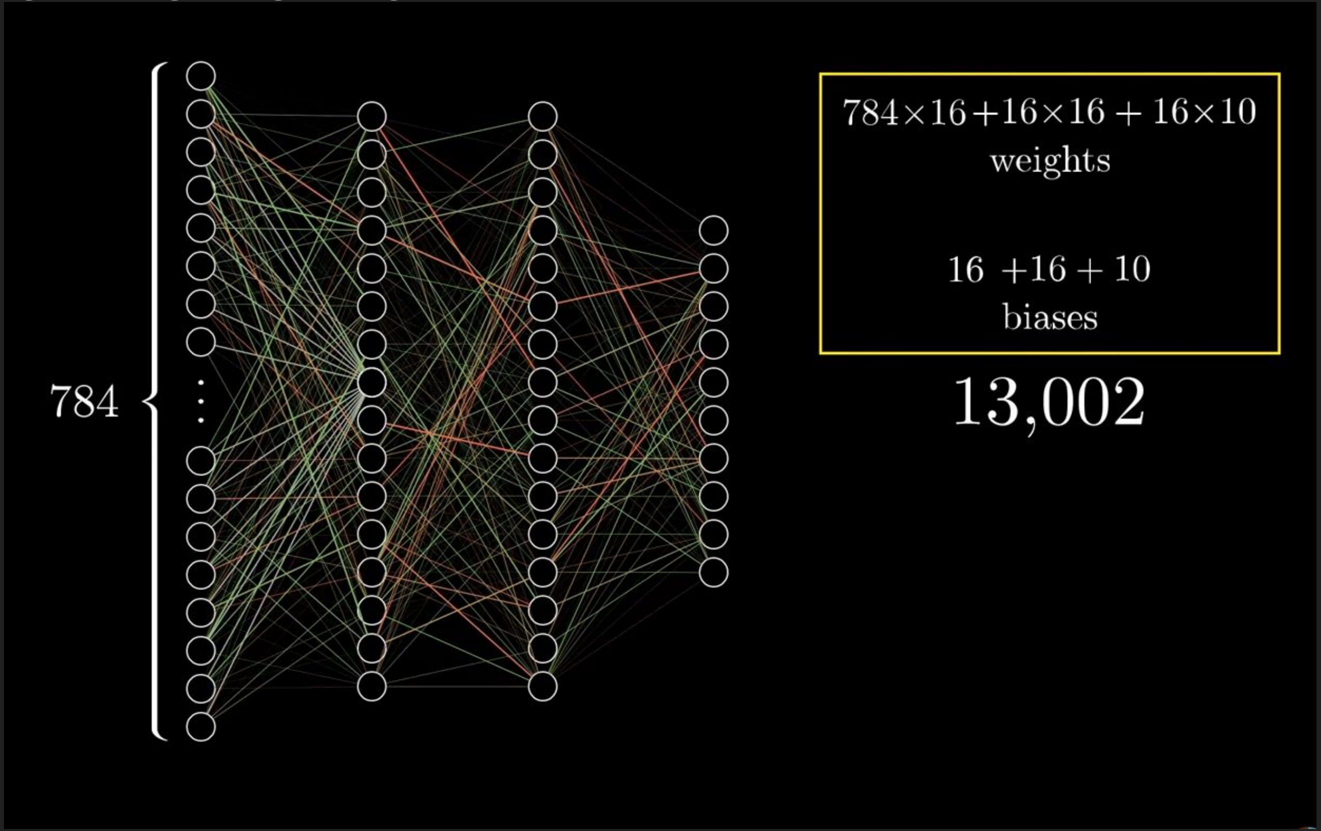
Total # of weights and biases in network = 13,002



Then we do the same thing for each node in the second layer of the network:

To train a model we need to find correct weights and biases to classify training data well

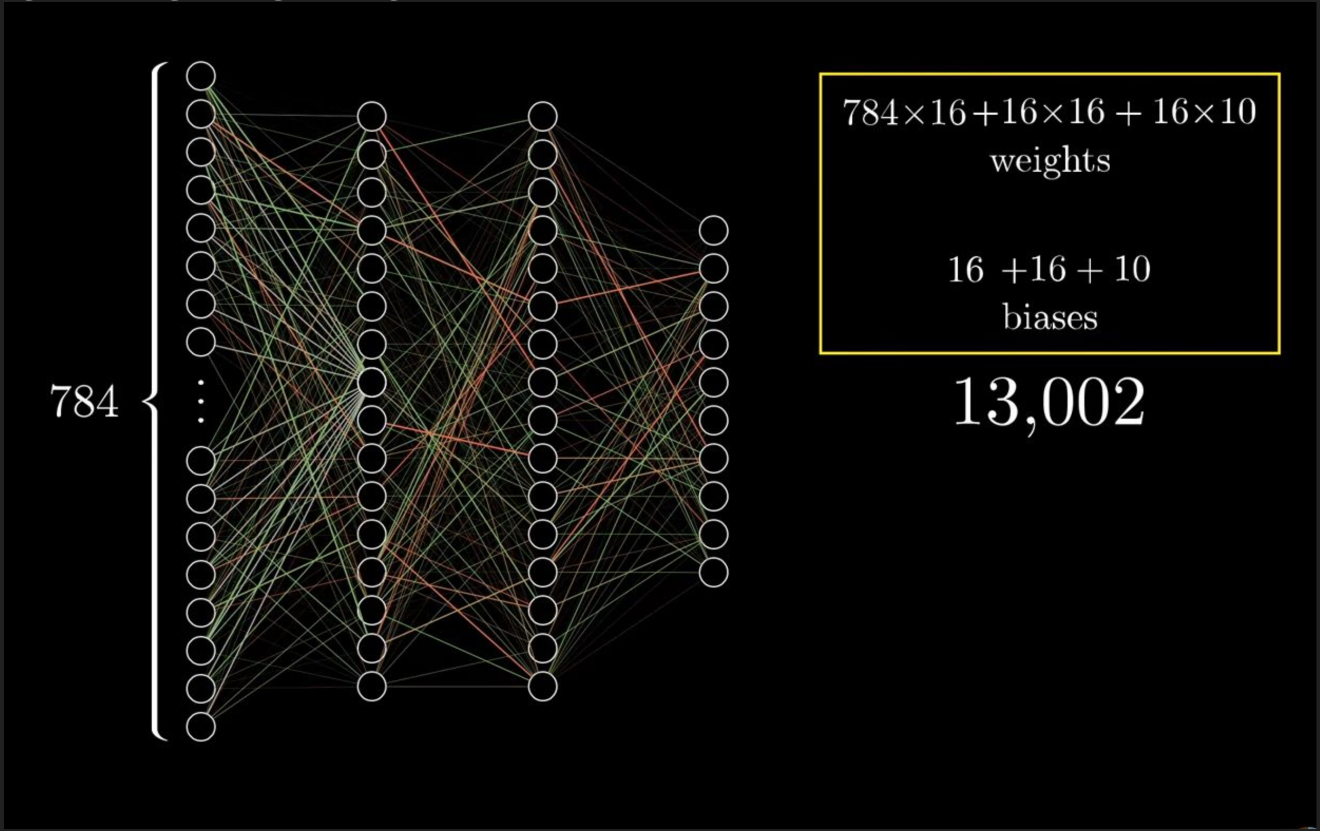
(and test data)



Then we do the same thing for each node in the second layer of the network:

The final trained network amounts to a function….

An absurdly complex one!



Part 2: How do we train a network model? (how do we find the best weights and biases?)

1) Assign first weights and biases randomly
2) Feed data forward to generate first predicted values
    a) **Initialize random weights** between each layer for every node (between -0.5 and 0.5).
    b) **Feed the data forward**:
        i) Multiply input values times weights for the hidden layers.
        ii) Add those products together and use the sum in the activation function.

Part 2: How do we train a network model? (how do we find the best weights and biases?)

1) Assign first weights and biases randomly
2) Feed data forward to generate first predicted values

C) Take the results of the hidden layer and multiply them by weights for the output layer.

D) Add up the product and bias for the given output node as the result for that node.

-This is gives you your predicted values for your output layer

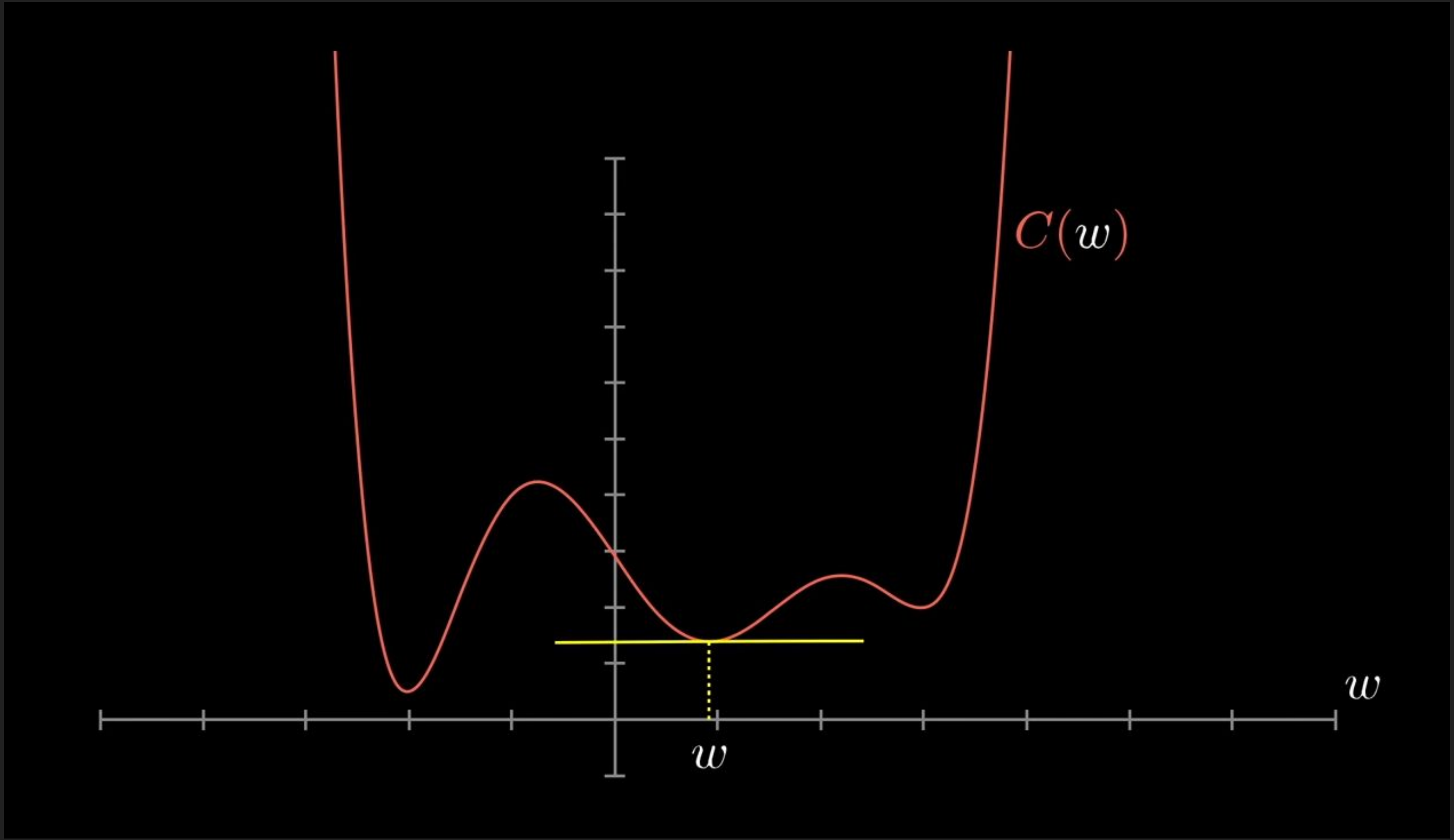
Predictions from first forward propagation via randomized weights/biases bad

We need to update the weights/biases intelligently to train the model.

How do we do it?

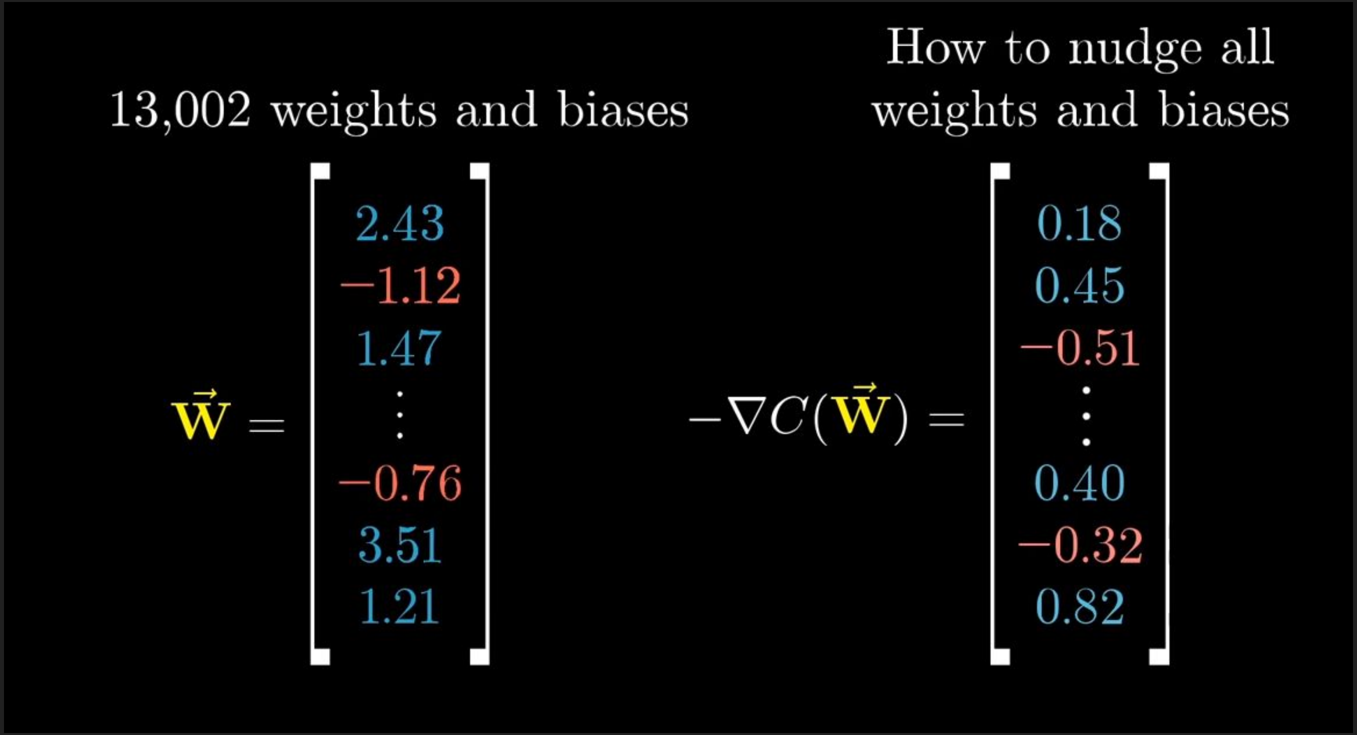
1. Define and calculate a cost function to minimize or maximize prediction error:
   a. We want to minimize error of predictions given our complex model
   b. So we will find the weights and biases that minimize prediction error
   c. Cost function could be RSS or aggregated error from cross entropy, etc.
   d. To tune our model, we use gradient descent to minimize error calculated via the cost function

Goal: Use gradient descent to find weights/biases that minimize cost function:



Iteratively adjust weights and biases by subtracting gradients:

1) Find error associated with each weight
2) Correct weights to minimize total prediction error



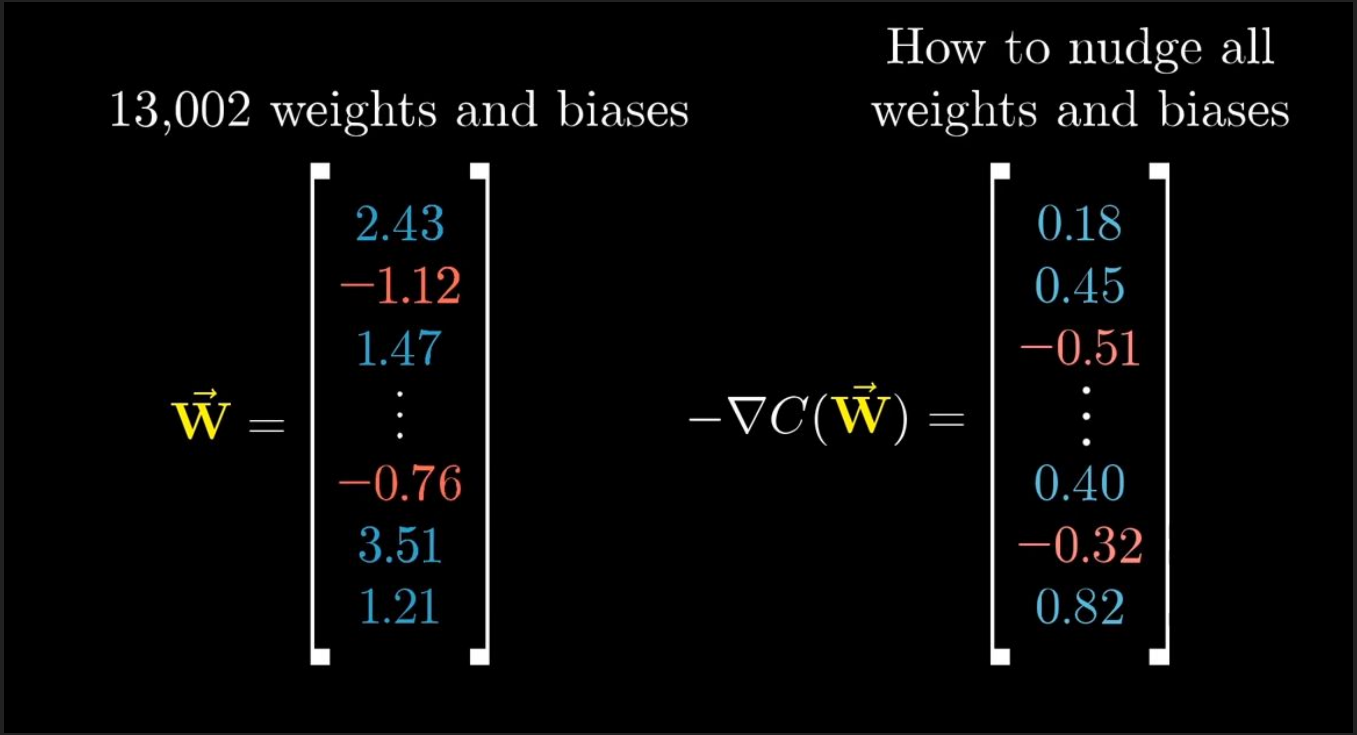
13,002 weights and biases

How to nudge all weights and biases

$$\vec{\mathbf{W}} = \begin{bmatrix} 2.43 \\ -1.12 \\ 1.47 \\ \vdots \\ -0.76 \\ 3.51 \\ 1.21 \end{bmatrix} \qquad -\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.18 \\ 0.45 \\ -0.51 \\ \vdots \\ 0.40 \\ -0.32 \\ 0.82 \end{bmatrix}$$

Iteratively adjust weights and biases by subtracting gradients:

1) Repeat small adjustments to minimize total model prediction error



13,002 weights and biases
How to nudge all weights and biases

$$\vec{\mathbf{W}} = \begin{bmatrix} 2.43 \\ -1.12 \\ 1.47 \\ \vdots \\ -0.76 \\ 3.51 \\ 1.21 \end{bmatrix} \qquad -\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.18 \\ 0.45 \\ -0.51 \\ \vdots \\ 0.40 \\ -0.32 \\ 0.82 \end{bmatrix}$$

Iteratively adjust weights and biases by subtracting gradients:

So we start with random weights and forward propagate to generate a network.

Then we take our predictions from the model, calculate total prediction error, and adjust weights and biases a bit. This is called back propagation.

Repeat forward and backward propagation until you minimize prediction error.

Iteratively adjust weights and biases by subtracting gradients:

Let’s use Python code to walk through how gradients can be used to adjust predictions in NNs...

...To get a feel for how weights and biases are adjusted

Now let’s practice fitting Neural Network models in Python...