BRAINTANK
DEEP LEARNING

Week 3:
The Handwritten Hoedown

August 9, 2021

1. Neural Networks (continued)
2. Classification Loss
3. Mini-Batching
4. Saving and loading model state

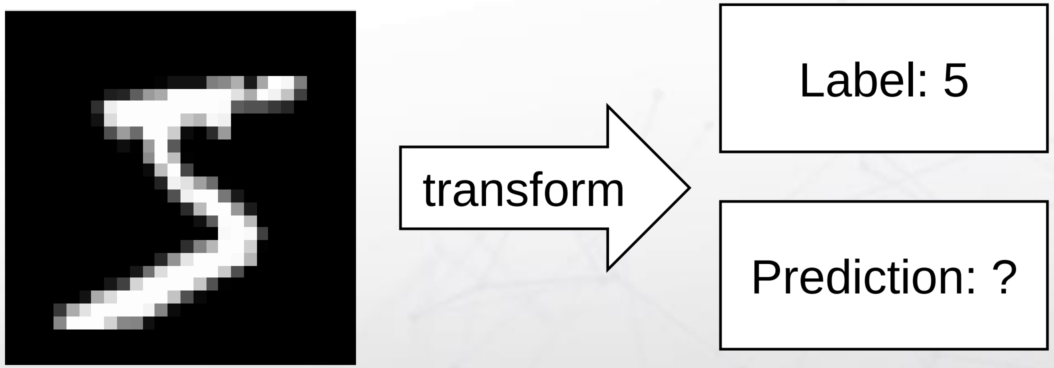
Problem:

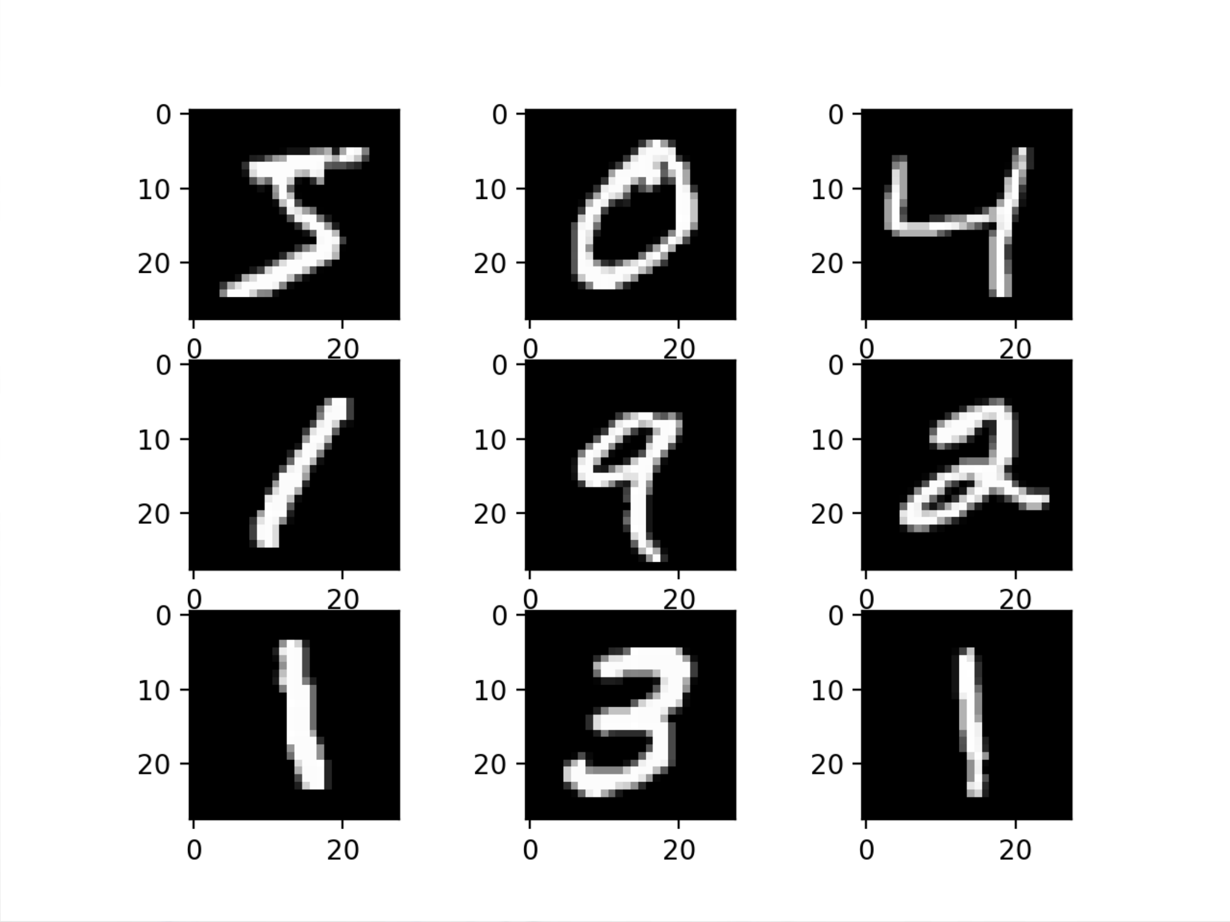
The University of Guelph runs a yearly square-dancing competition, where contestants send in videos of their choreography, and Agriculture department professors rate their moves on a scale of 0 to 9. Due to poor planning, the university used paper and pen to record their scores, and are having trouble digitizing their results. They have asked BrainTank to take these written scores and label them correctly. We are hoping to be 95% correct in our classification.

The Dataset

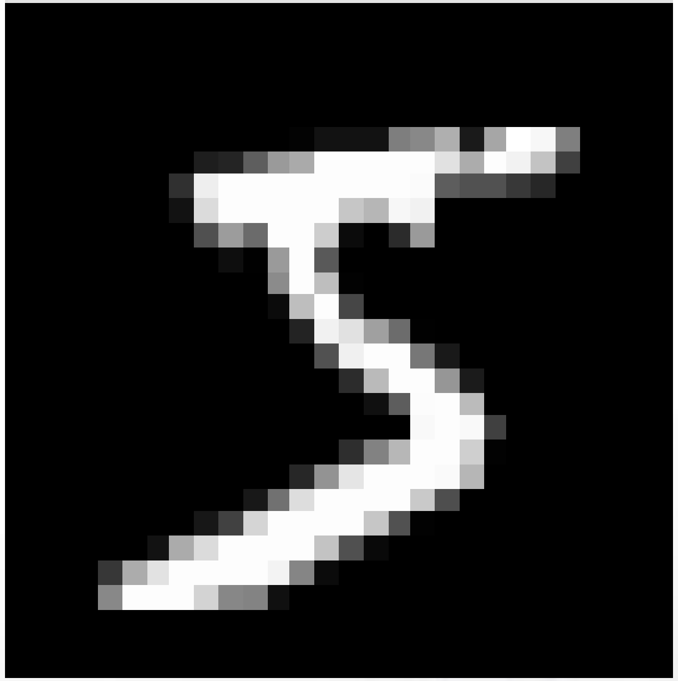
The Dataset

- 70,000 images.
- Each image is 28 by 28 pixels
- Each image is in greyscale (black and white)
- We will make a training set of 60,000 images
- We will make a test set of 10,000 images
- All images have a label associated with them. Labels are integers of values 0 to 9.





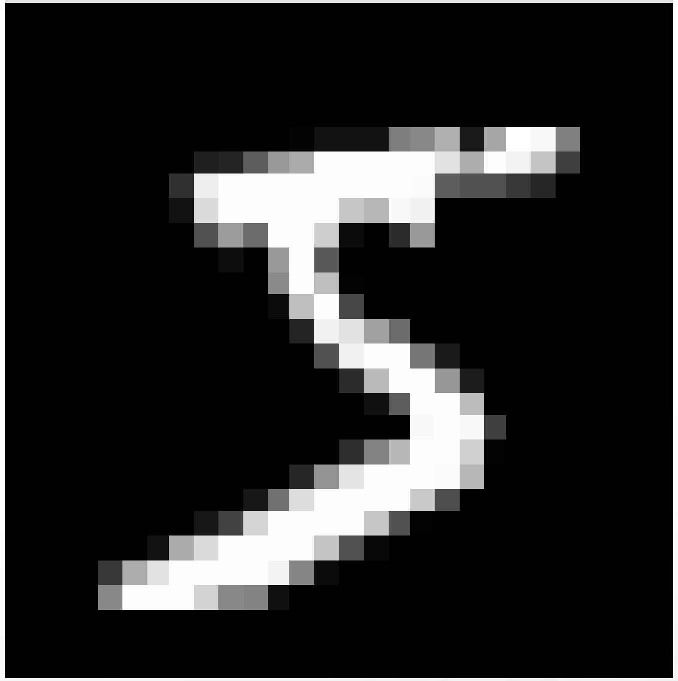
How are images represented



Representation

A tensor
(3 dimension array)
with a shape:
[1, 28, 28]

How are images represented

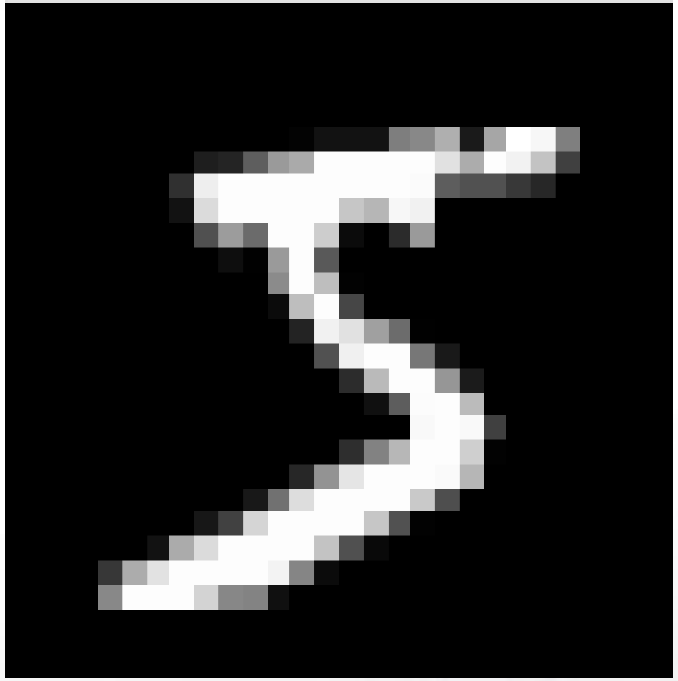


Representation

A tensor
(3 dimension array)
with a shape:
[1, 28, 28]

1. 1 colour dimension (black and white)
2. 28 pixels high
3. 28 pixels wide

How are images represented



Representation →

A tensor
(3 dimension array)
with a shape:
[1, 28, 28]

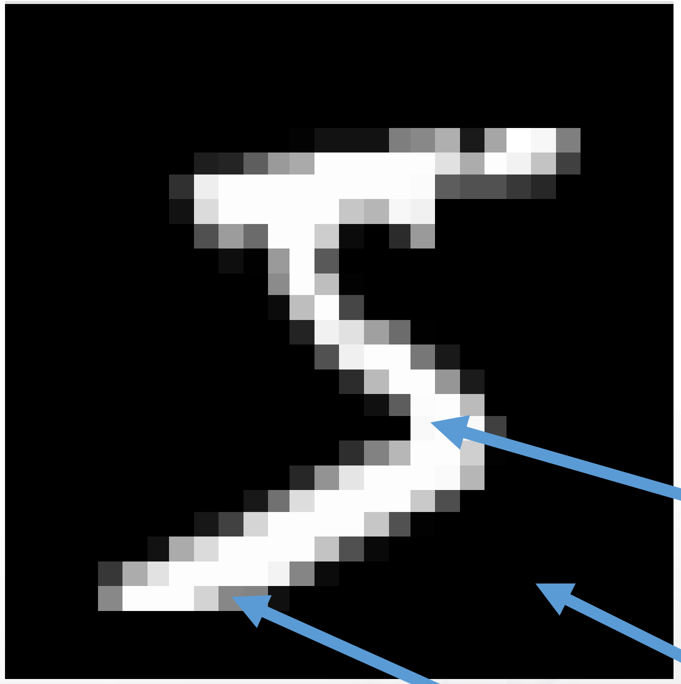
White pixels have a value of 256

Black pixels have a value of 0

Grey pixels have an intermediate value

1. 1 colour dimension (black and white)
2. 28 pixels high
3. 28 pixels wide

How are images represented



Representation

A tensor
(3 dimension array)
with a shape:
[1, 28, 28]

White pixels have a value of 256

Black pixels have a value of 0

Grey pixels have an intermediate value

1. 1 colour dimension (black and white)
2. 28 pixels high
3. 28 pixels wide

Normalization

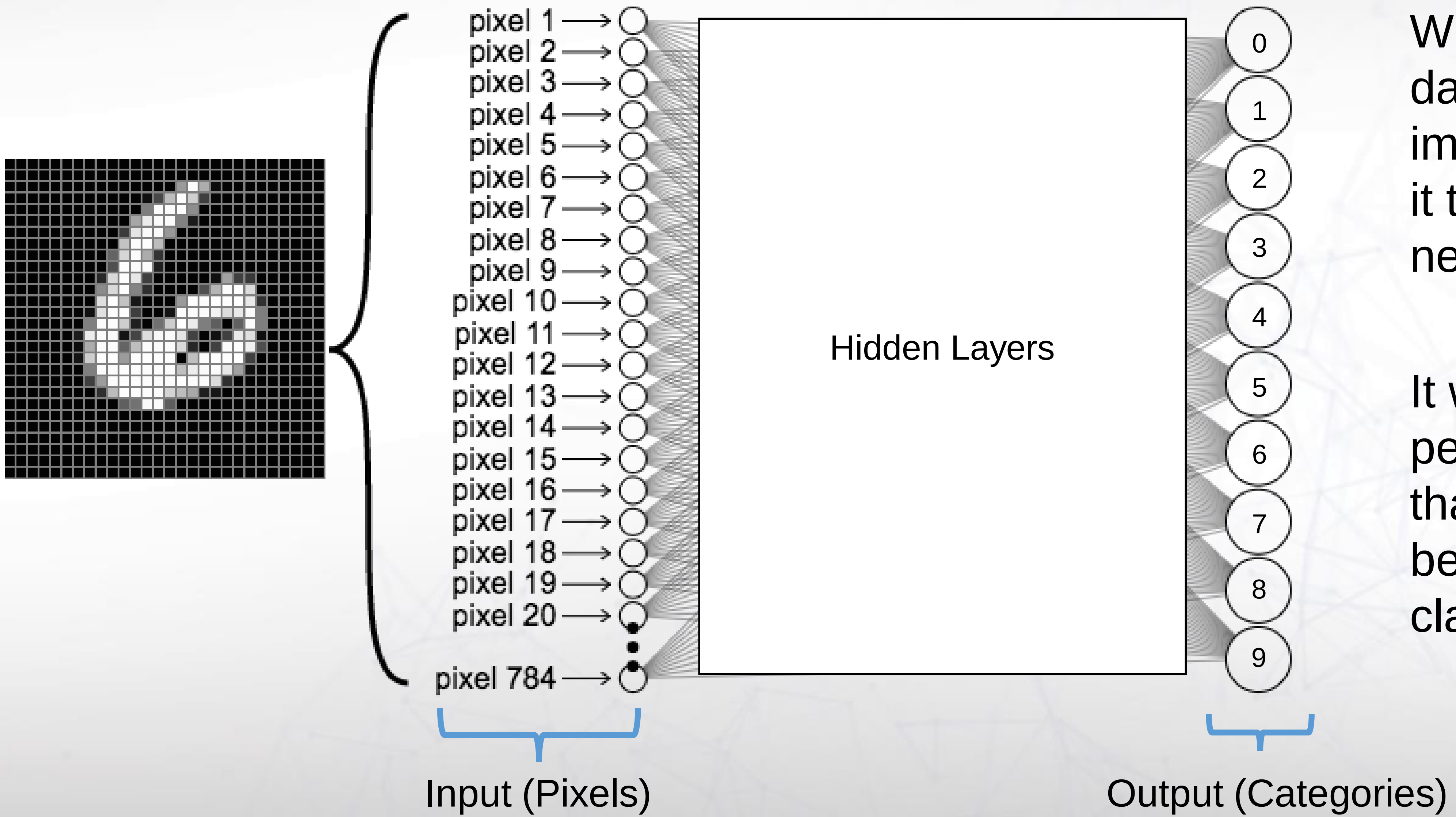
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	0.64	1.93	1.6	1.5	0.34	0.03	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	2.4	2.81	2.81	2.81	2.81	2.64	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	1.74	0.24	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	0.43	1.03	0.49	1.03	1.65	2.47	2.81	2.44	2.81	2.81	2.81	2.76	2.49	2.81	2.81	1.36	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.2	0.42	-0.2	0.43	0.43	0.43	0.33	-0.2	2.58	2.81	0.92	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	0.63	2.8	2.24	-0.2	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.1	2.54	2.82	0.63	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	1.22	2.81	2.61	0.14	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	0.33	2.75	2.81	0.36	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	1.27	2.81	1.96	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.3	2.19	2.73	0.31	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	1.18	2.81	1.89	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	0.53	2.77	2.63	0.3	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.2	2.39	2.81	1.69	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	2.16	2.81	2.36	0.02	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	0.06	2.81	2.81	0.56	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0	2.43	2.81	1.04	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	1.27	2.81	2.81	0.24	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	0.35	2.66	2.81	2.81	0.24	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	1.12	2.81	2.81	2.36	0.08	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	1.12	2.81	2.21	-0.2	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4

Look closely, you can make out a number 7.

All values are normalized to be between the values of -0.4 and 2.81.

This is an example of normalizing with mean and standard deviation values.

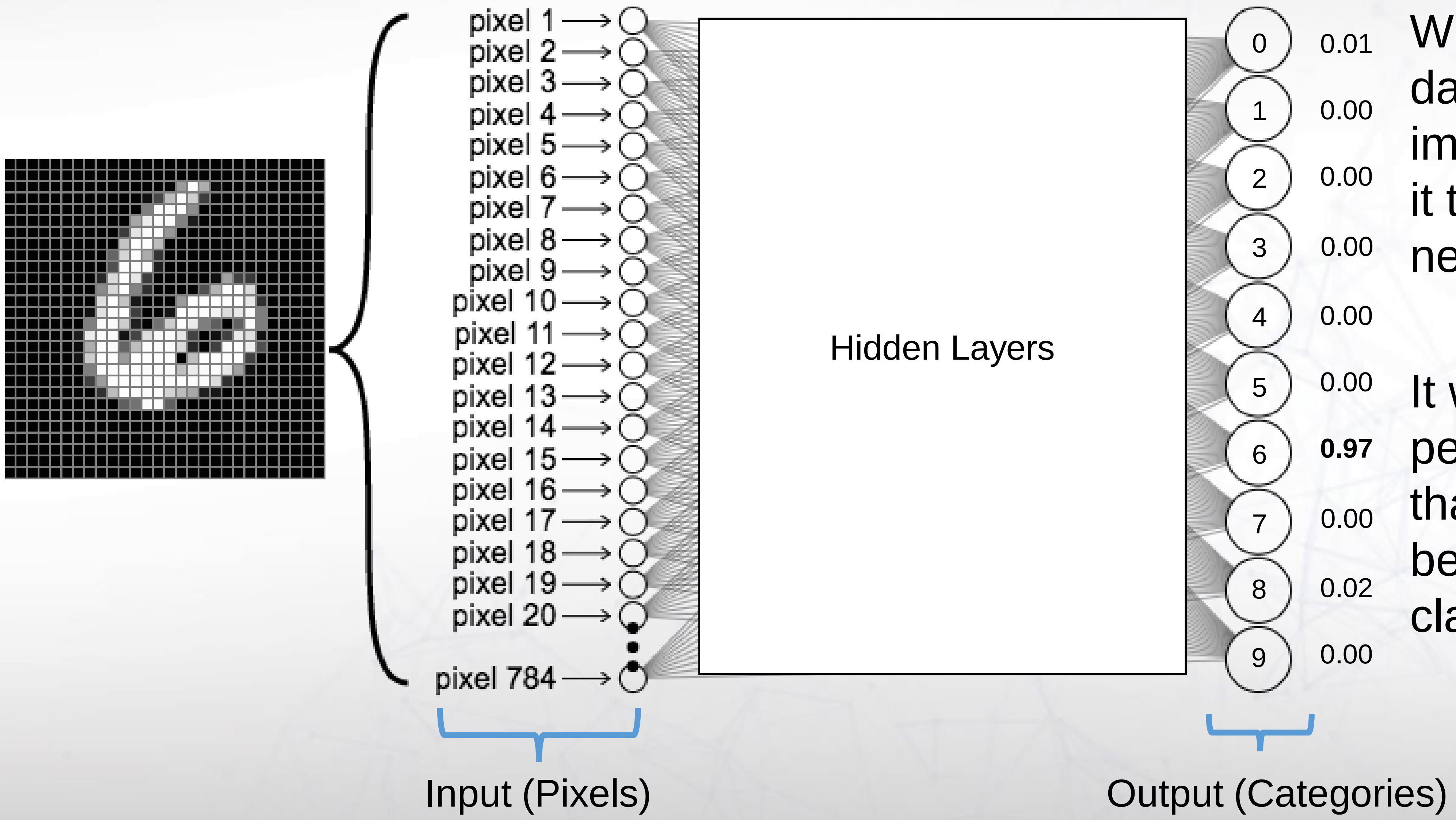
How we going to tackle this problem



We will the pixel data of each image, and will run it through a neural network.

It will output percent likelihood that that image belongs to each class.

How we going to tackle this problem

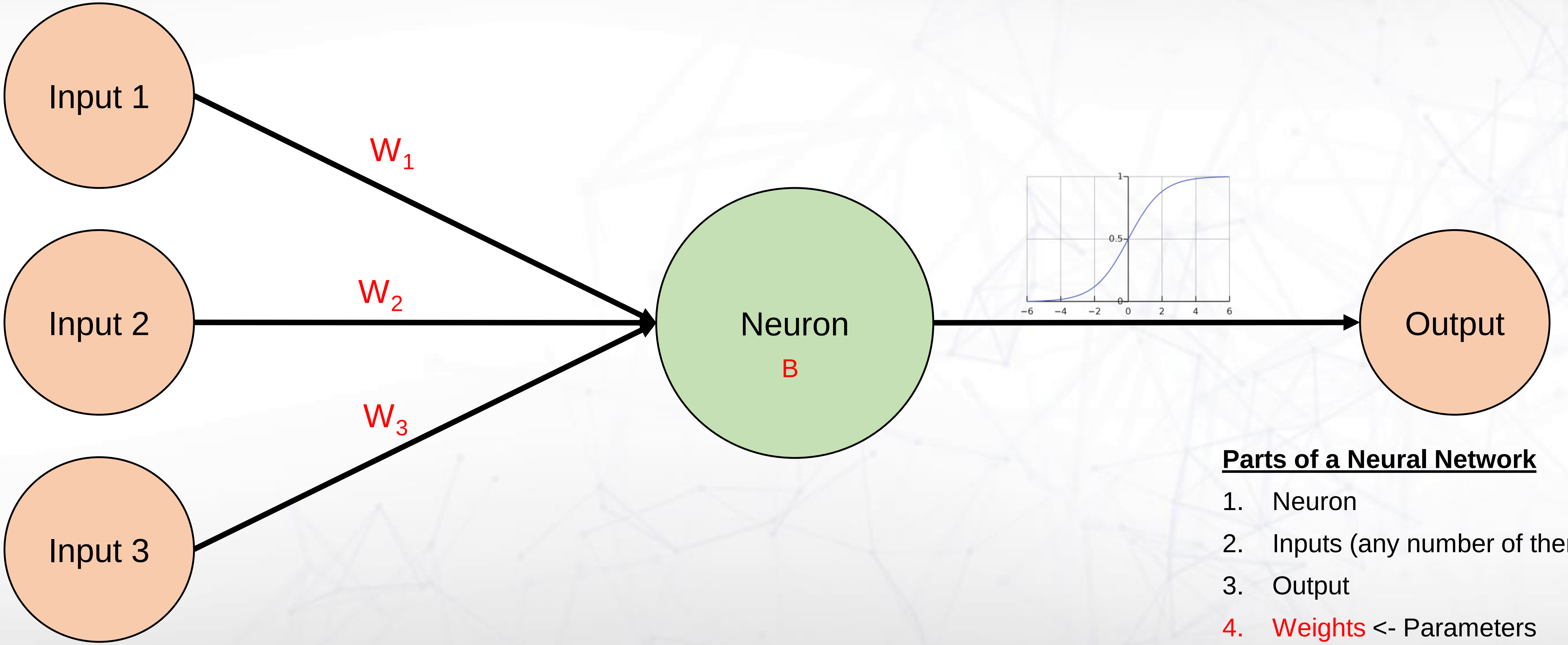


We will the pixel data of each image, and will run it through a neural network.

It will output percent likelihood that that image belongs to each class.

Getting Deeper with Neural Networks

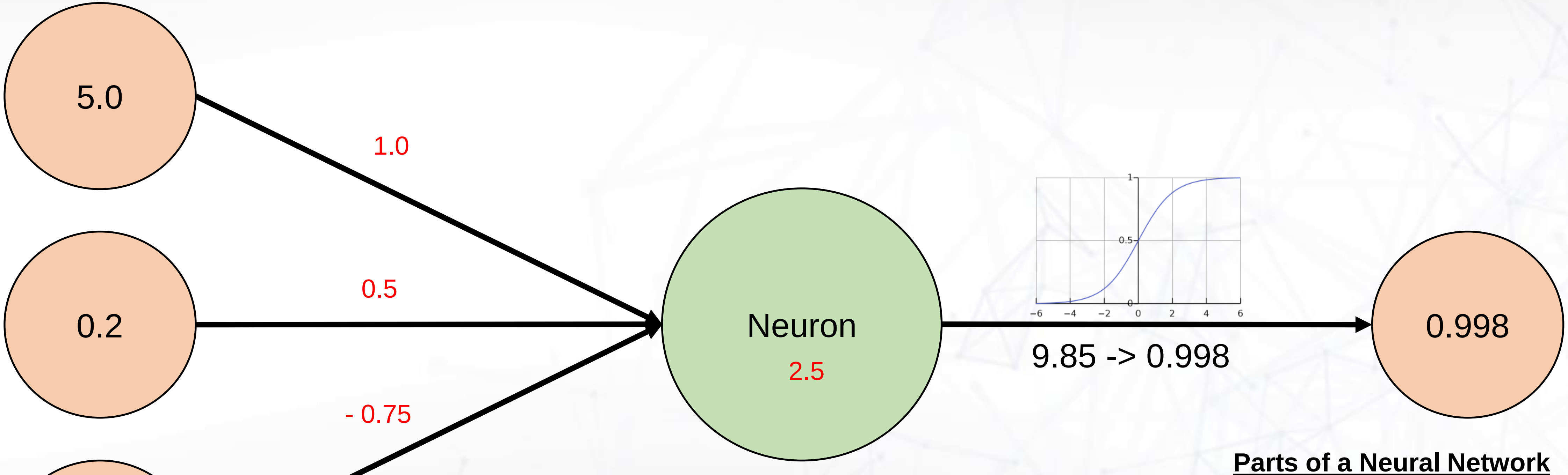
The Neuron



<u>Parts of a Neural Network</u>

1. Neuron
2. Inputs (any number of them)
3. Output
4. Weights <- Parameters
5. Bias <- Parameter
6. NonLinearity

The Neuron



$$\text{Output} = Sigmoid\left(\sum(Wi \ * Input\ i) + b\right)$$

$= (5.0 * 1.0) + (0.2 * 0.5) + (- 3 * - 0.75) + 2.5$

$= 5.0 + 0.1 + 2.25 + 2.5$

$= 9.85$

Remember: Parameters are just numbers

Parts of a Neural Network

1. Neuron
2. Inputs (any number of them)
3. Output
4. Weights <- Parameters
5. Bias <- Parameter
6. NonLinearity

If one neuron is strong, thousands are stronger

How sunny is it today?

Value -1 -> raining

Value 1 -> sunny

Do I feel Like Walking?

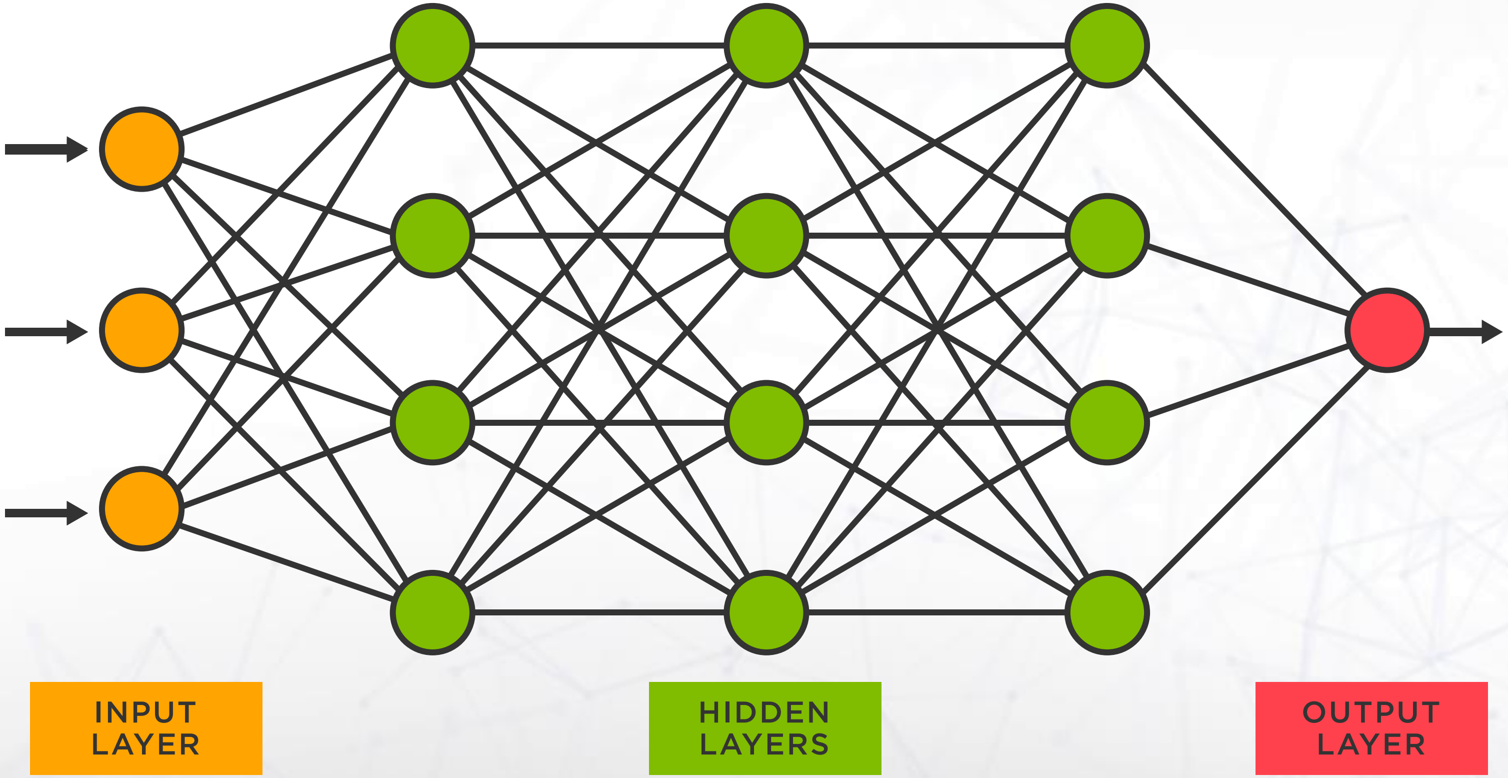
Value -1 -> no

Value 1 -> yes

Is the dog at Grandma's right now?

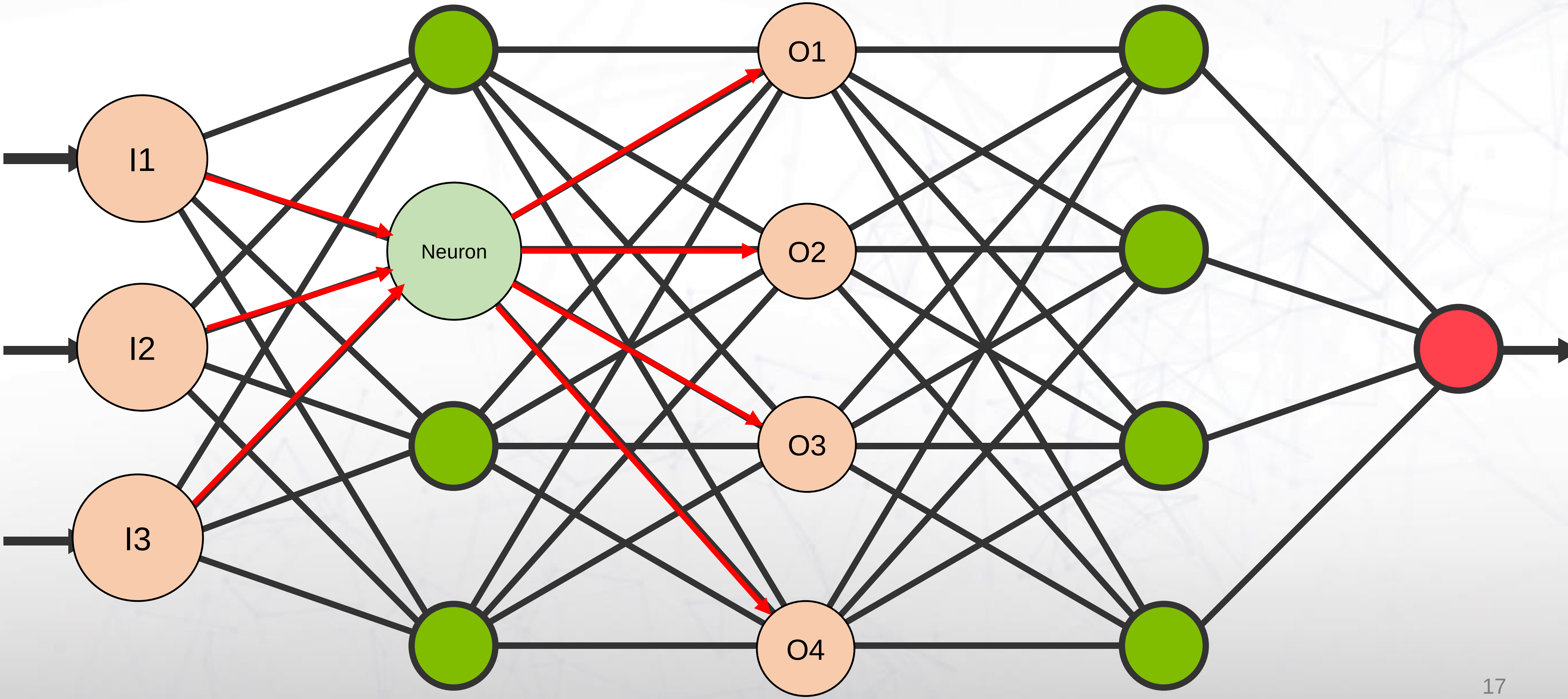
Value 0 -> no

Value 1 -> yes



Likeliness I will walk my dog

If one neuron is strong, thousands are stronger



Using PyTorch to define models

```
self.layer1 = torch.nn.Linear(24, 1000)
self.layer2 = torch.nn.Linear(1000, 500)
self.layer3 = torch.nn.Linear(500, 200)
self.layer4 = torch.nn.Linear(500, 200)
self.layer5 = torch.nn.Linear(200, 1)
```

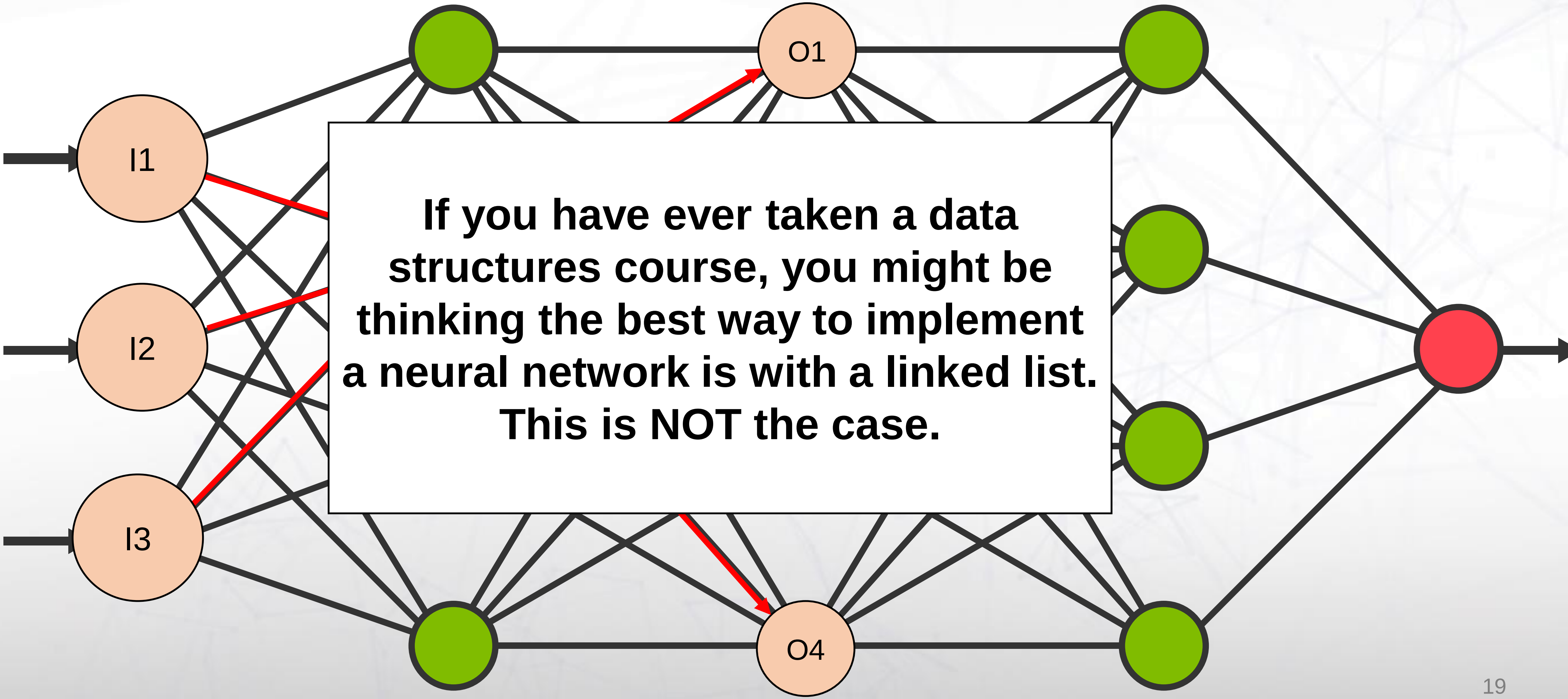
This neural net has 5 layers.

1. Activation layer with 24 inputs
2. Hidden layer of dimension 1000
3. Hidden layer of dimension 500
4. Hidden layer of dimension 200
5. Singular output layer

Notice how PyTorch focuses on layers of neural networks instead of neurons instead.

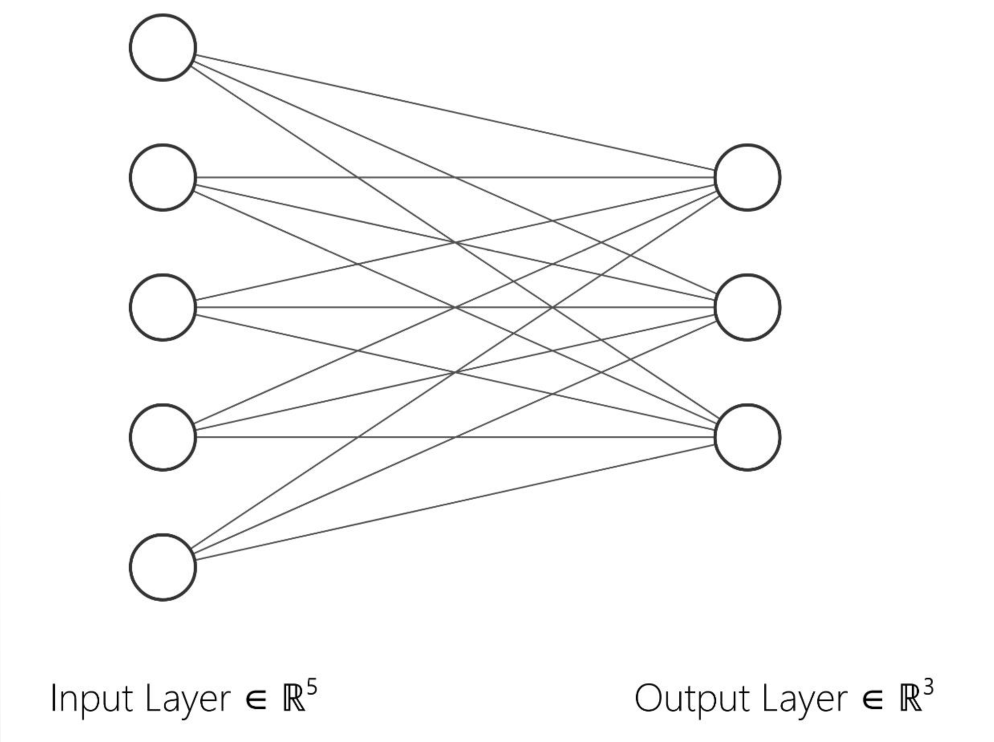
We will be investigating why this is by creating our own version of torch.nn.Linear(input_dim, output_dim)

If one neuron is strong, thousands are stronger



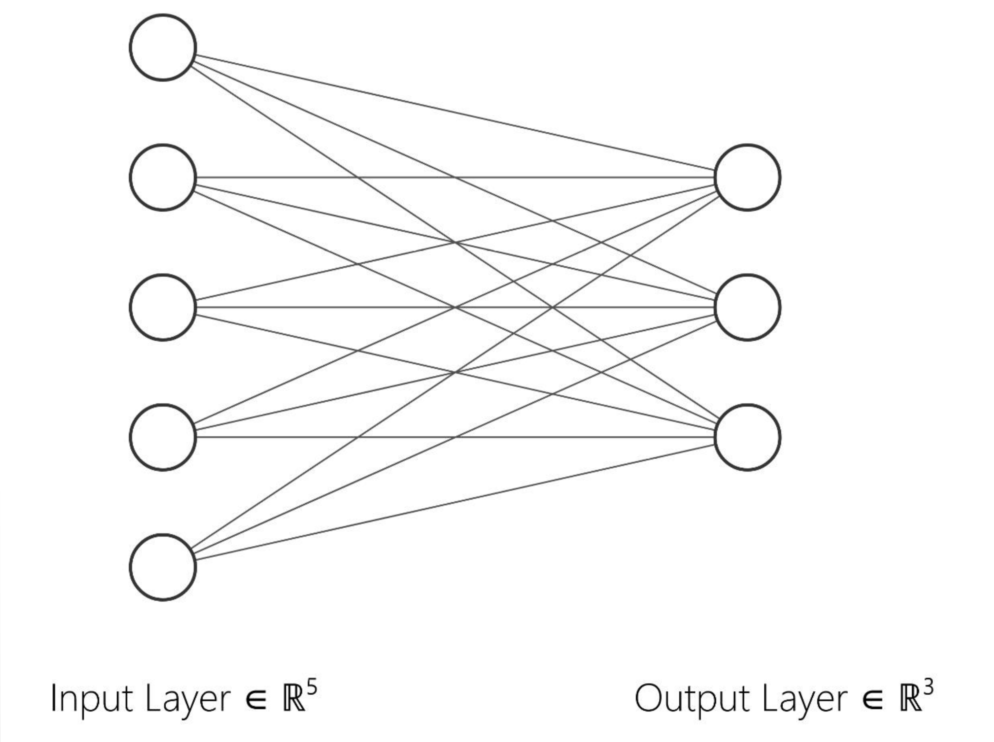
If you have ever taken a data structures course, you might be thinking the best way to implement a neural network is with a linked list. This is NOT the case.

Let’s use this network as an example



This network has 5 inputs and 3 outputs

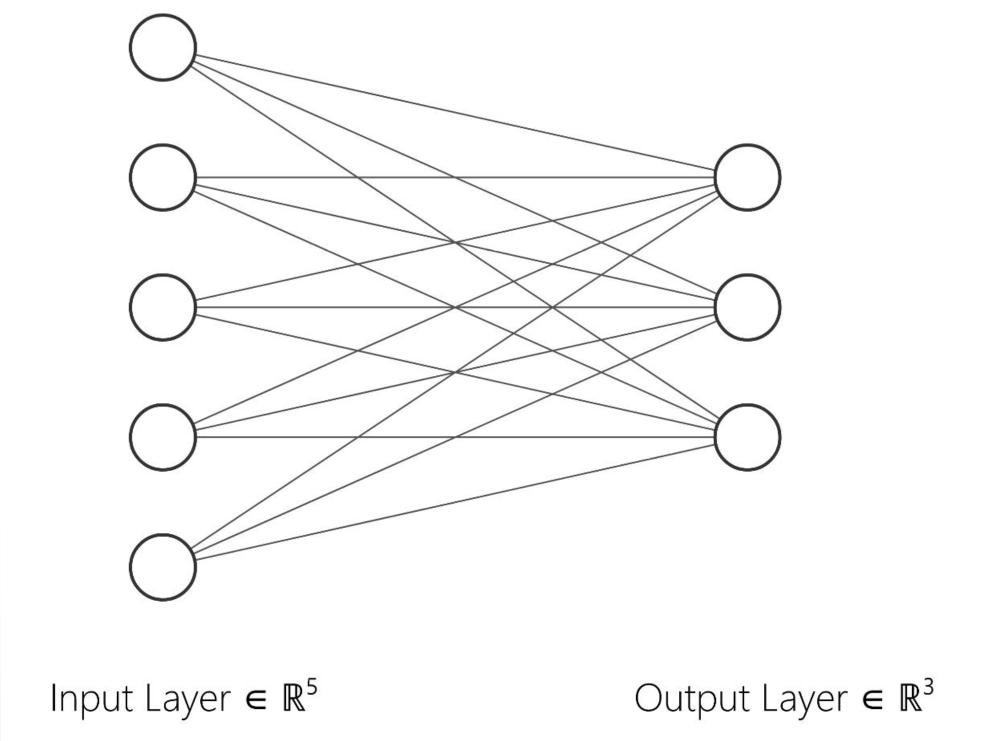
Let’s use this network as an example



This network has 5 inputs and 3 outputs

```
self.layer = torch.nn.Linear(5, 3)
```

Let’s use this network as an example

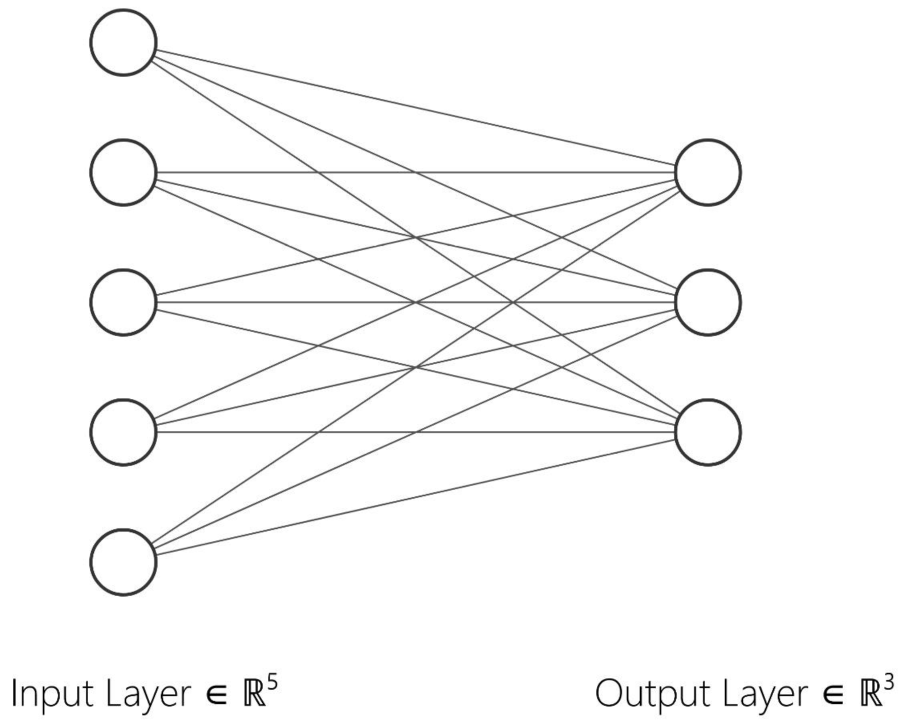


This network has 5 inputs and 3 outputs

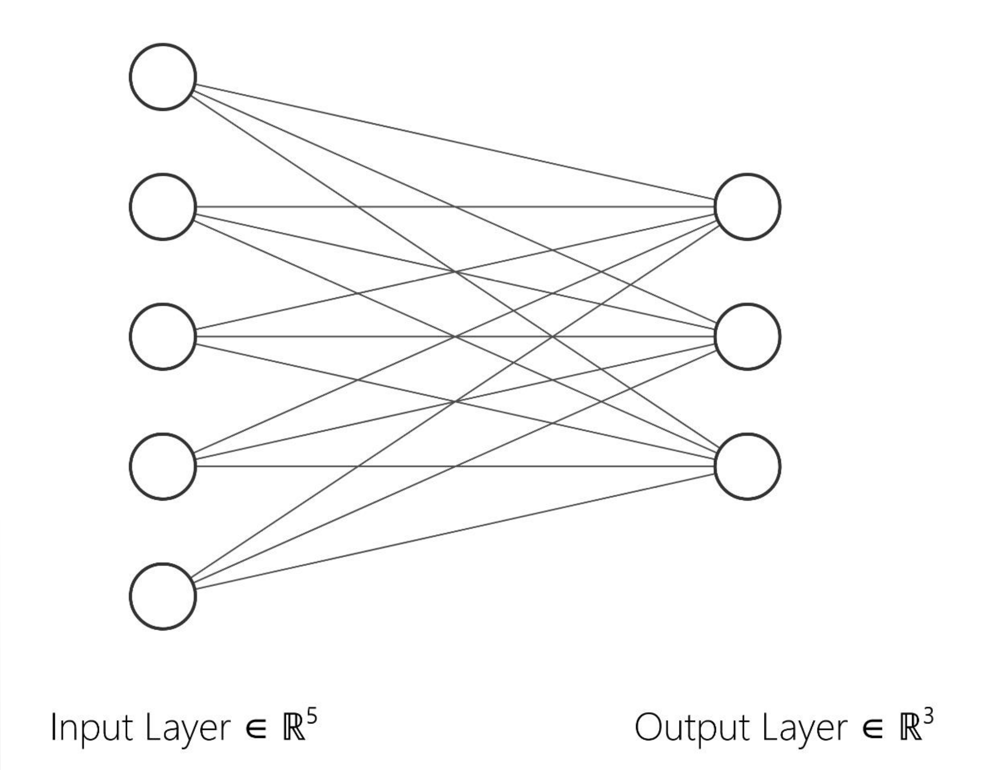
```
self.layer = torch.nn.Linear(5, 3)
```

Vector (Array) of 5 numbers → Neural Network Implementation → Vector (Array) of 3 numbers

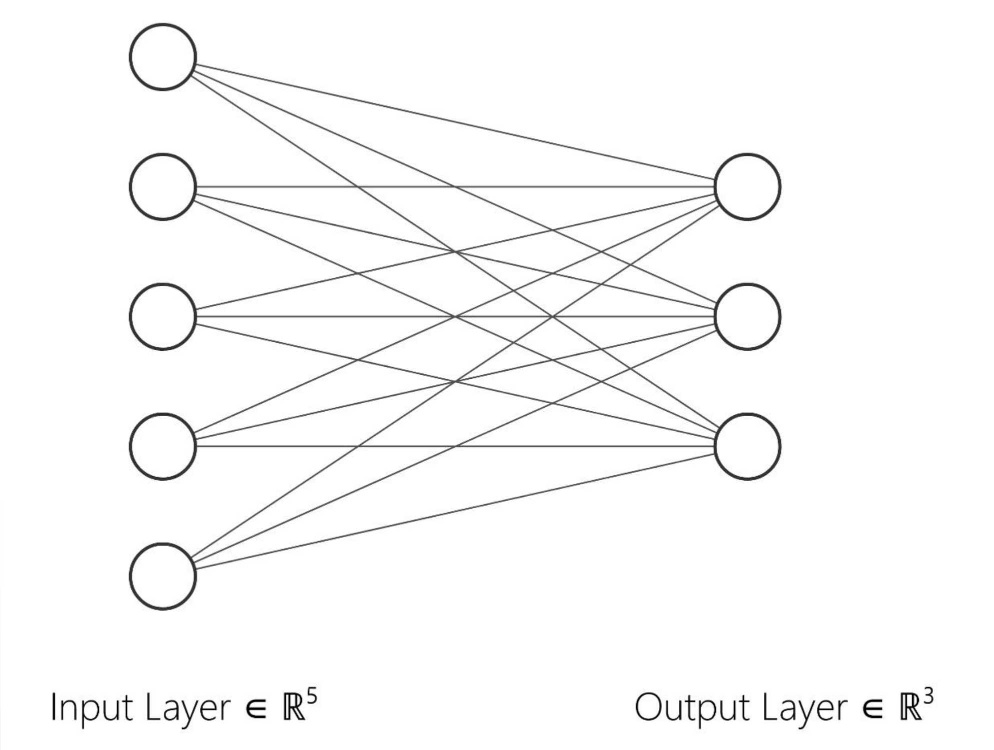
Let’s use this network as an example

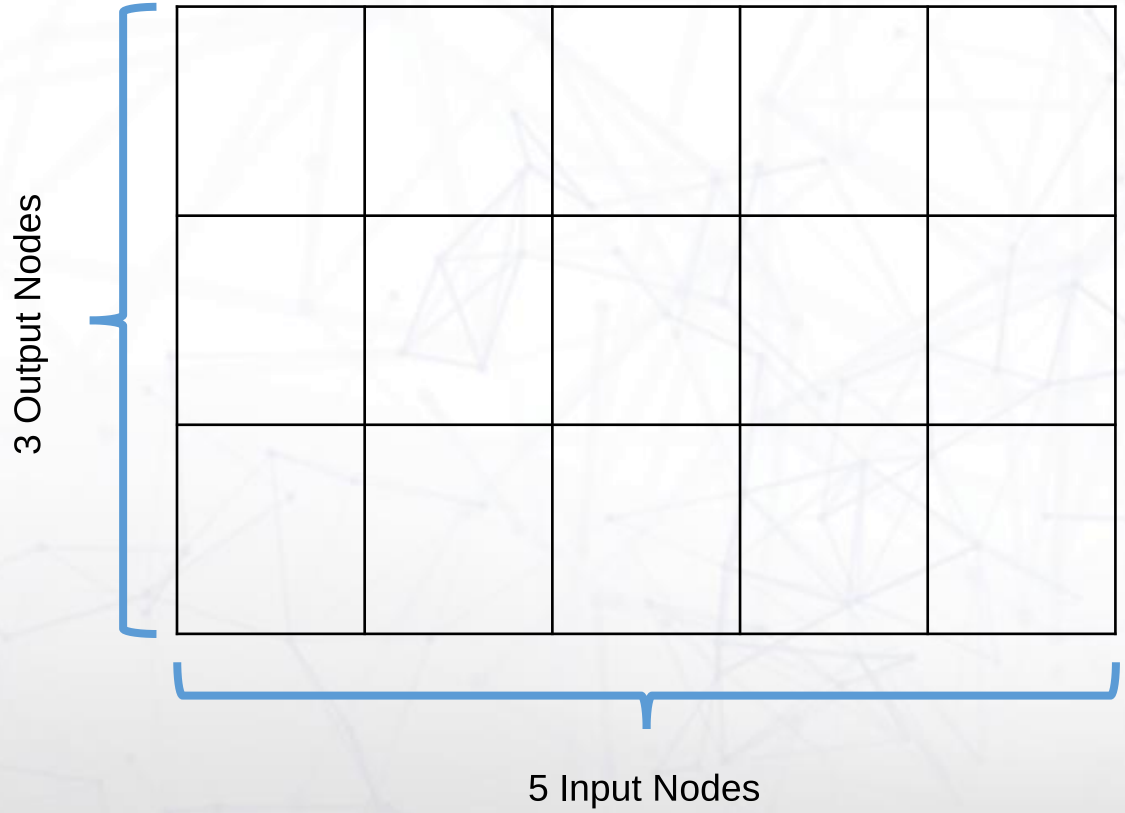


Let's use this network as an example



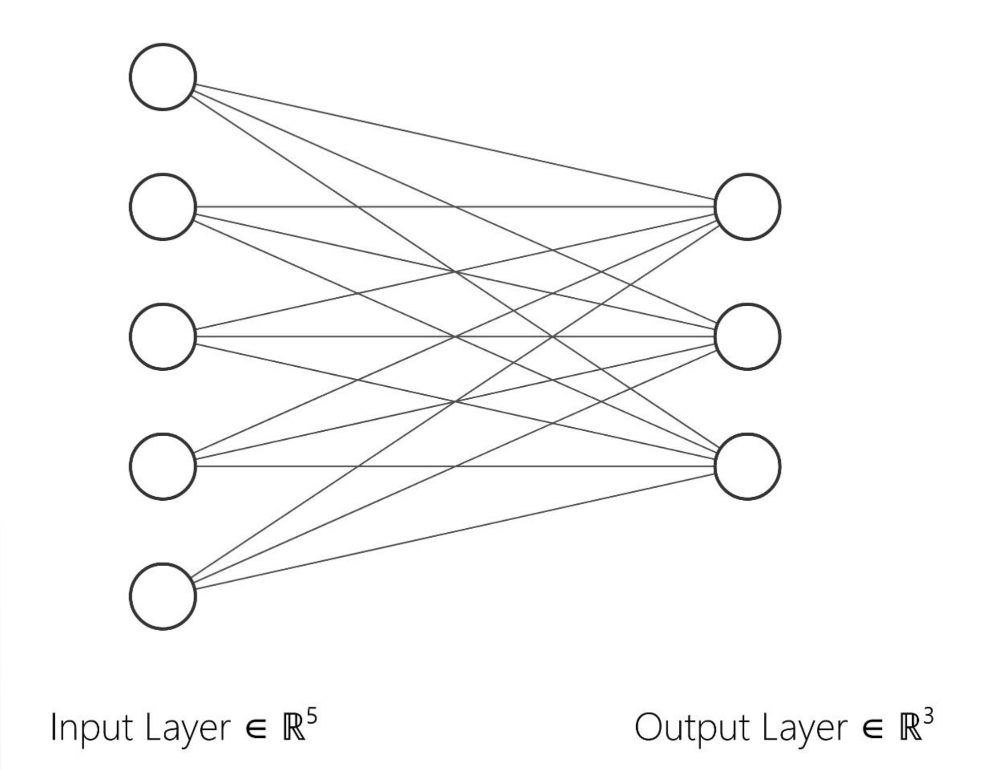
Let's use this network as an example

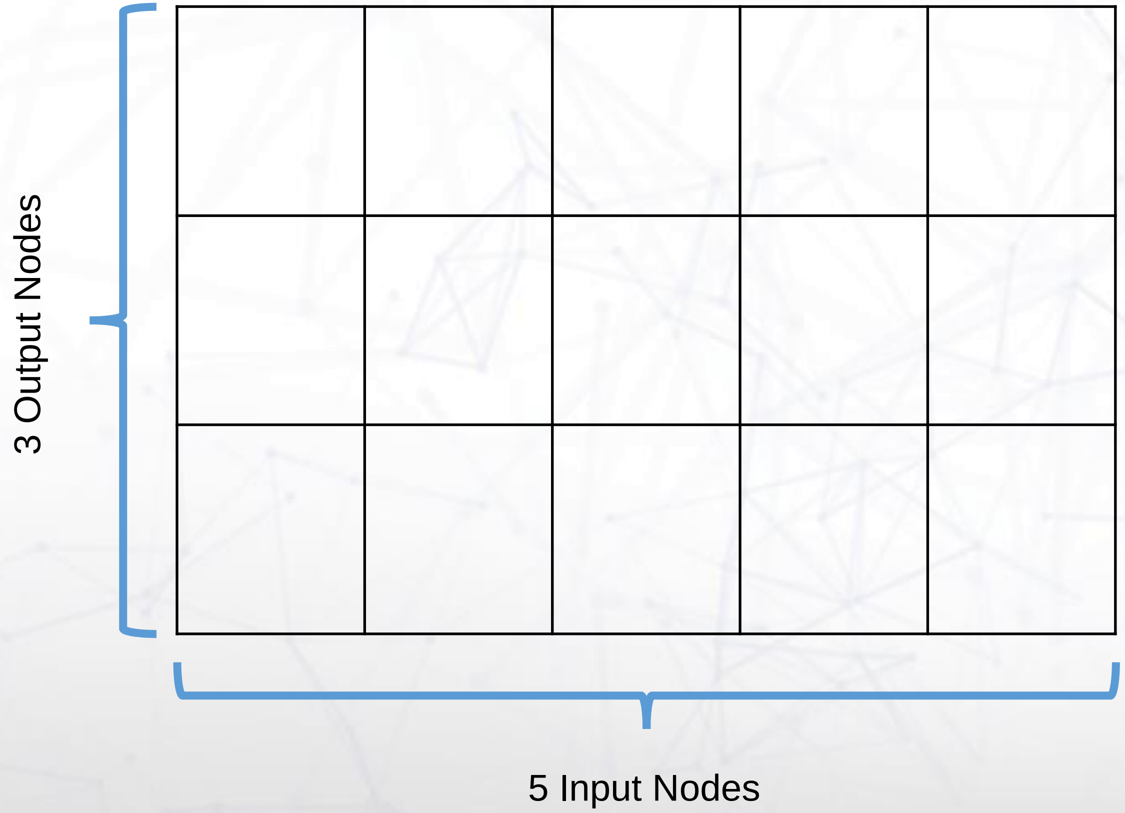




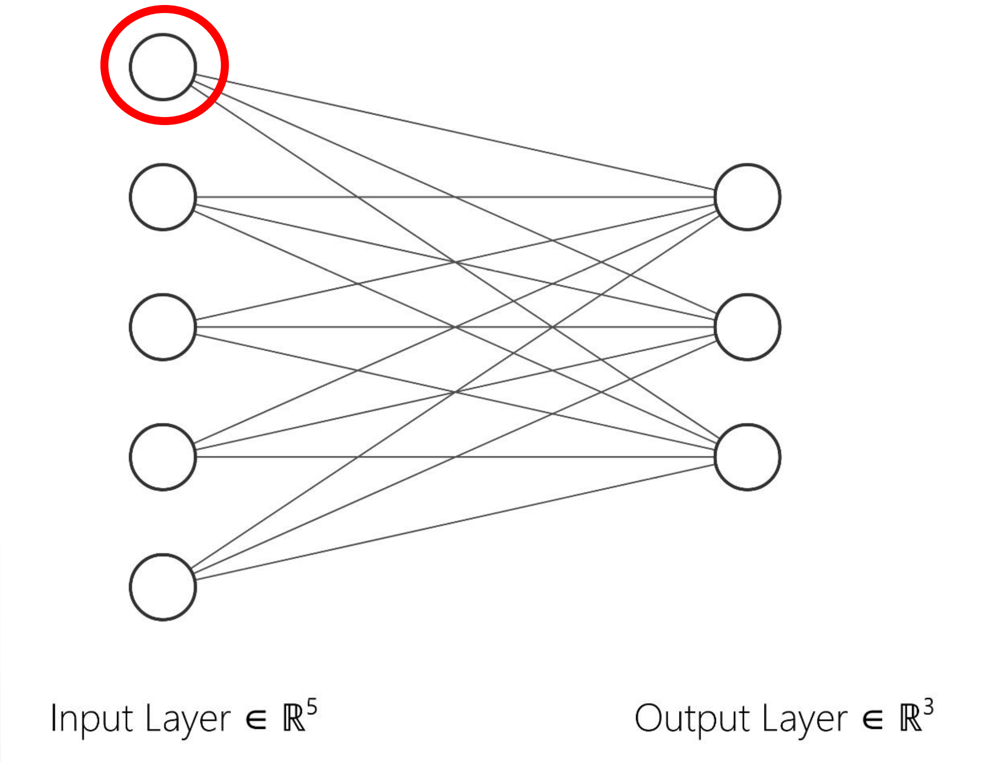
Let’s use this network as an example

$\mathbf{M} \in R^{3 \times 5}$

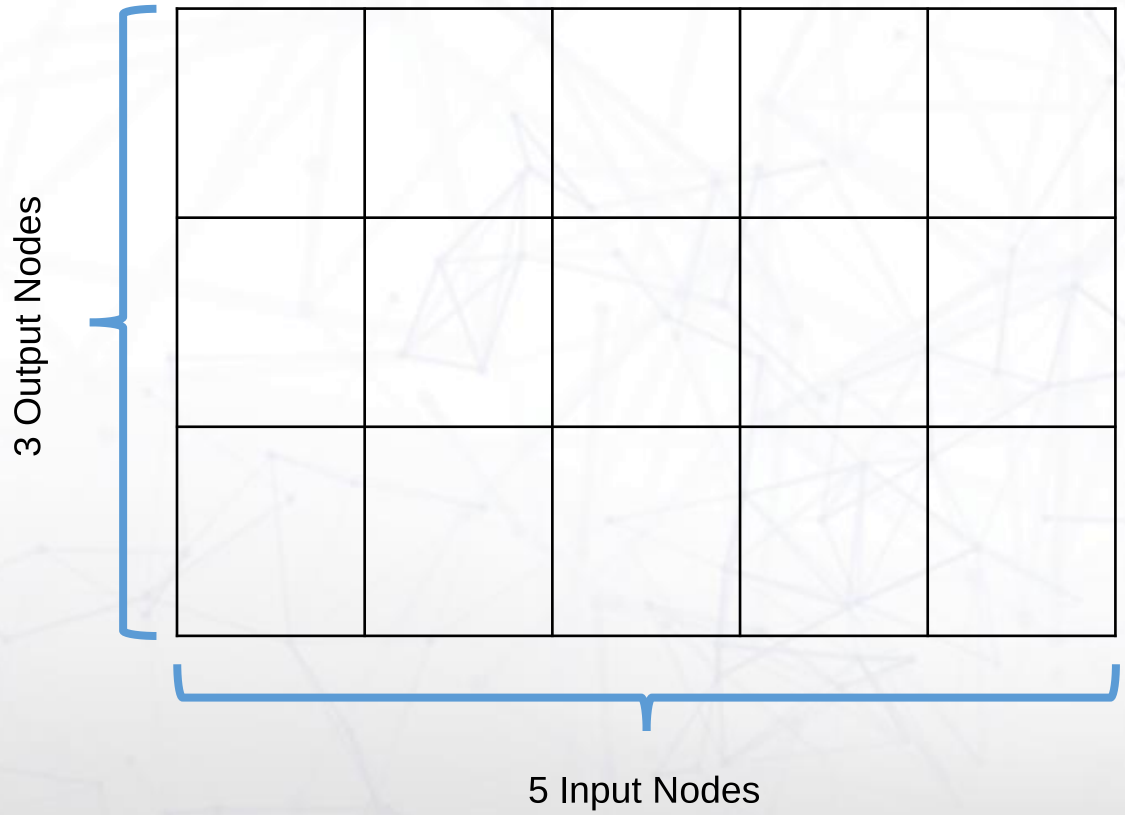




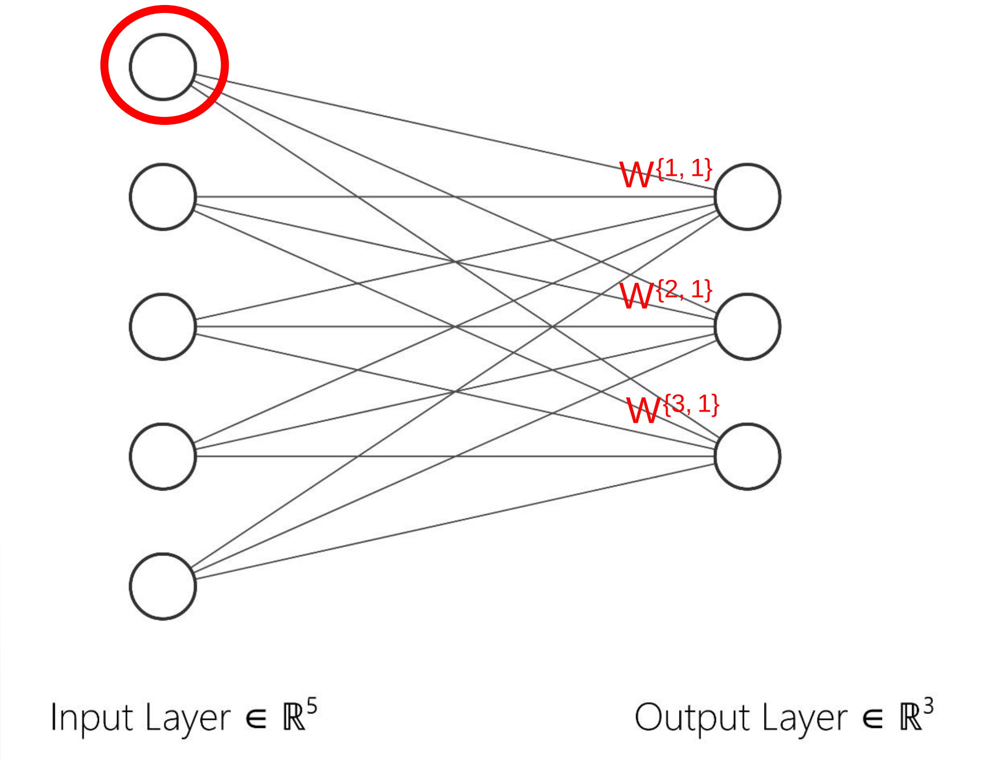
Let's use this network as an example



$\mathbf{M} \in R^{3 \times 5}$



Let’s use this network as an example



$\mathbf{M} \in R^{3 \times 5}$

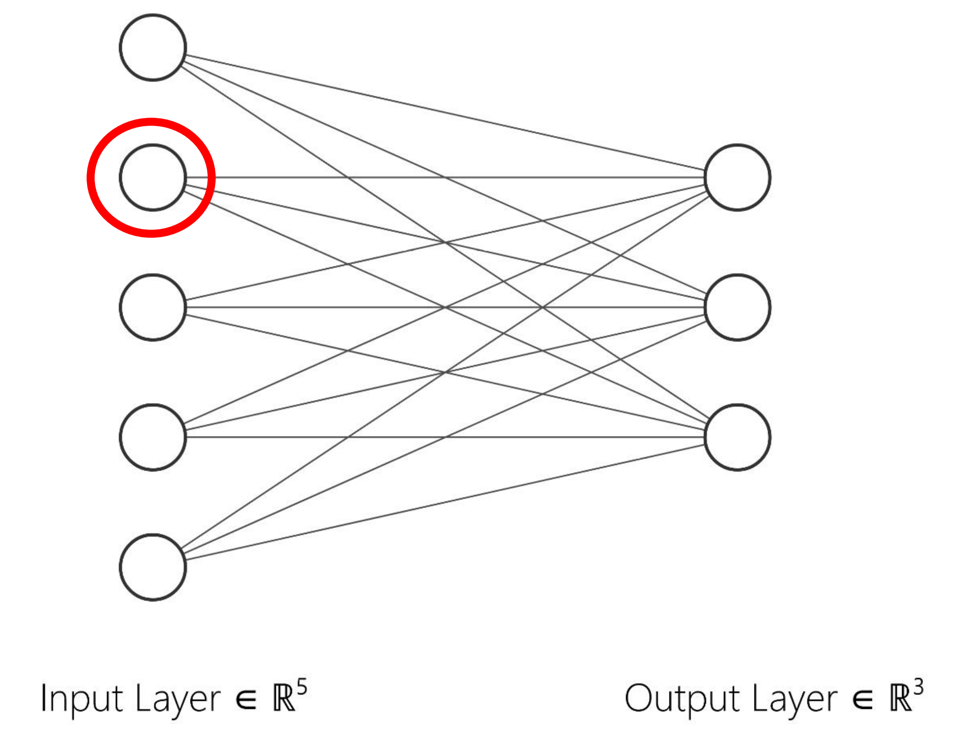
3 Output Nodes

$W^{\{1, 1\}}$				
$W^{\{2, 1\}}$				
$W^{\{3, 1\}}$				

5 Input Nodes

$W^{\{\text{Output Node Number, Input Node Number}\}}$

Let’s use this network as an example



$$\mathbf{M} \in R^{3 \times 5}$$

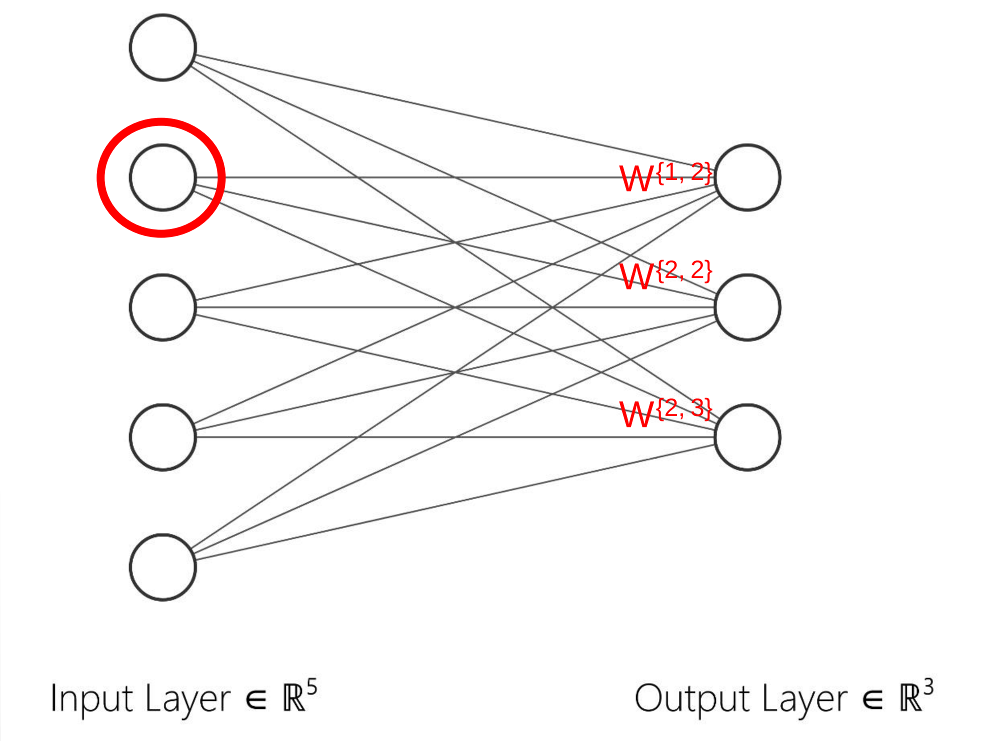
3 Output Nodes

$W^{\{1, 1\}}$				
$W^{\{2, 1\}}$				
$W^{\{3, 1\}}$				

5 Input Nodes

$W^{\{\text{Output Node Number, Input Node Number}\}}$

Let's use this network as an example



$$\mathbf{M} \in R^{3 \times 5}$$

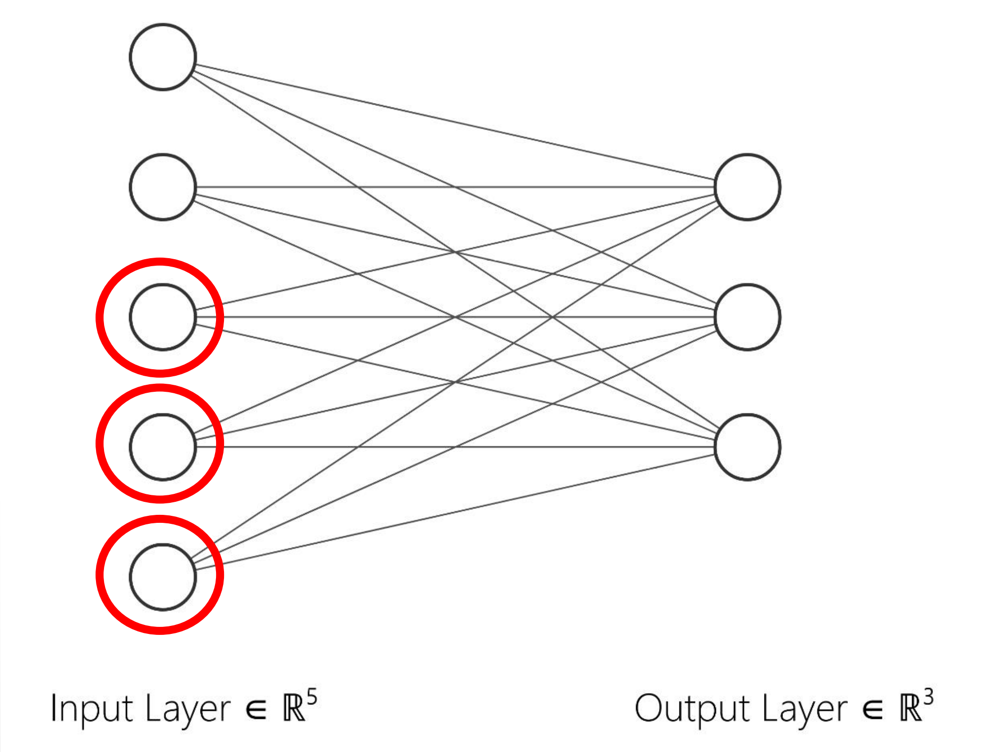
3 Output Nodes

$W^{\{1, 1\}}$	$W^{\{1, 2\}}$			
$W^{\{2, 1\}}$	$W^{\{2, 2\}}$			
$W^{\{3, 1\}}$	$W^{\{3, 2\}}$			

5 Input Nodes

$W^{\{\text{Output Node Number, Input Node Number}\}}$

Let’s use this network as an example



$$\mathbf{M} \in R^{3 \times 5}$$

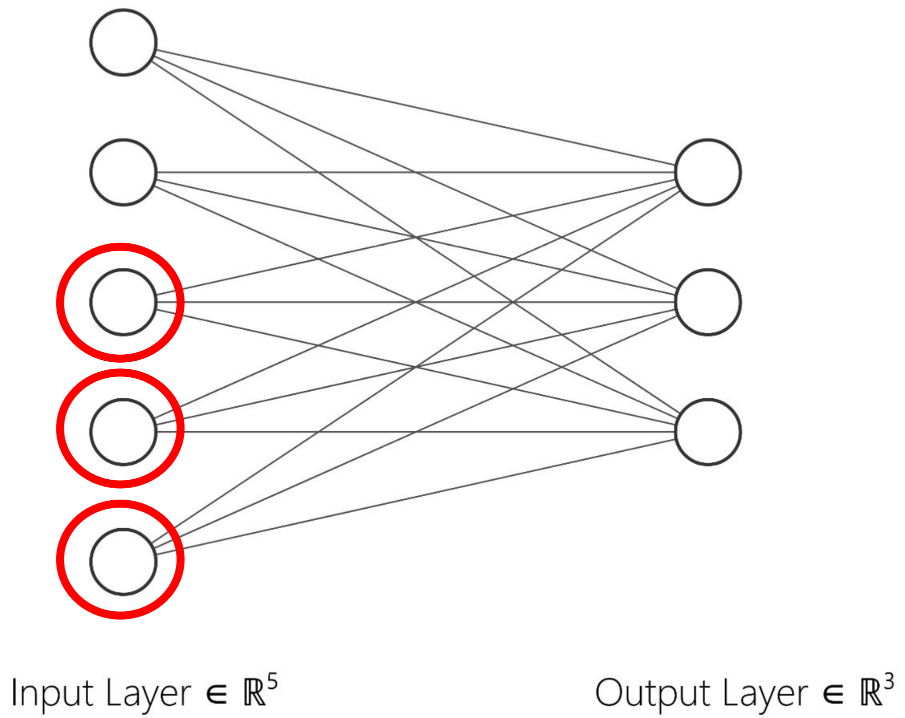
3 Output Nodes

$W^{\{1, 1\}}$	$W^{\{1, 2\}}$			
$W^{\{2, 1\}}$	$W^{\{2, 2\}}$			
$W^{\{3, 1\}}$	$W^{\{3, 2\}}$			

5 Input Nodes

$W^{\{\text{Output Node Number, Input Node Number}\}}$

Let’s use this network as an example



$$\mathbf{M} \in R^{3 \times 5}$$

$$\text{3 Output Nodes}\left\{\underbrace{\begin{array}{|c|c|c|c|c|}\hline W^{\{1,\,1\}} & W^{\{1,\,2\}} & W^{\{1,\,3\}} & W^{\{1,\,4\}} & W^{\{1,\,5\}} \\ \hline W^{\{2,\,1\}} & W^{\{2,\,2\}} & W^{\{2,\,3\}} & W^{\{2,\,4\}} & W^{\{2,\,5\}} \\ \hline W^{\{3,\,1\}} & W^{\{3,\,2\}} & W^{\{3,\,3\}} & W^{\{3,\,4\}} & W^{\{3,\,5\}} \\ \hline \end{array}}_{\text{5 Input Nodes}}\right.$$

$W^{\{\text{Output Node Number, Input Node Number}\}}$

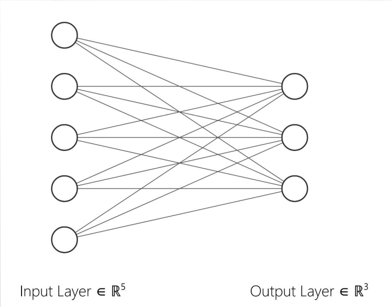
Let's use this network as an example

$\mathbf{v} \in R^{1 \times 5}$

$Input_1$
$Input_2$
$Input_3$
$Input_4$
$Input_5$

$\mathbf{M} \in R^{3 \times 5}$

$W^{\{1, 1\}}$	$W^{\{1, 2\}}$	$W^{\{1, 3\}}$	$W^{\{1, 4\}}$	$W^{\{1, 5\}}$
$W^{\{2, 1\}}$	$W^{\{2, 2\}}$	$W^{\{2, 3\}}$	$W^{\{2, 4\}}$	$W^{\{2, 5\}}$
$W^{\{3, 1\}}$	$W^{\{3, 2\}}$	$W^{\{3, 3\}}$	$W^{\{3, 4\}}$	$W^{\{3, 5\}}$



$W^{\{\text{Output Node Number, Input Node Number}\}}$

Let's use this network as an example

BRAINTANK DEEP LEARNING

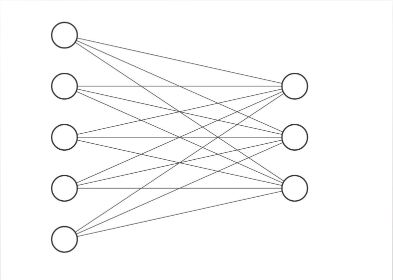
$\mathbf{v} \in R^{1 \times 5}$

$Input_1$
$Input_2$
$Input_3$
$Input_4$
$Input_5$

$\mathbf{M} \in R^{3 \times 5}$

$W^{\{1, 1\}}$	$W^{\{1, 2\}}$	$W^{\{1, 3\}}$	$W^{\{1, 4\}}$	$W^{\{1, 5\}}$
$W^{\{2, 1\}}$	$W^{\{2, 2\}}$	$W^{\{2, 3\}}$	$W^{\{2, 4\}}$	$W^{\{2, 5\}}$
$W^{\{3, 1\}}$	$W^{\{3, 2\}}$	$W^{\{3, 3\}}$	$W^{\{3, 4\}}$	$W^{\{3, 5\}}$

Transpose Operator (Flipping the axis)



$W^{\{\text{Output Node Number, Input Node Number}\}}$

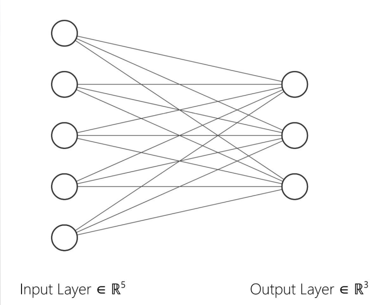
Let's use this network as an example

$\mathbf{V} \in R^{1 \times 5}$ $\mathbf{M} \in R^{5 \times 3}$

$Input_1$
$Input_2$
$Input_3$
$Input_4$
$Input_5$

X

$W^{\{1, 1\}}$	$W^{\{2, 1\}}$	$W^{\{3, 1\}}$
$W^{\{1, 2\}}$	$W^{\{2, 2\}}$	$W^{\{3, 2\}}$
$W^{\{1, 3\}}$	$W^{\{2, 3\}}$	$W^{\{3, 3\}}$
$W^{\{1, 4\}}$	$W^{\{2, 4\}}$	$W^{\{3, 4\}}$
$W^{\{1, 5\}}$	$W^{\{2, 5\}}$	$W^{\{3, 5\}}$



$W^{\{\text{Output Node Number, Input Node Number}\}}$

Let’s use this network as an example

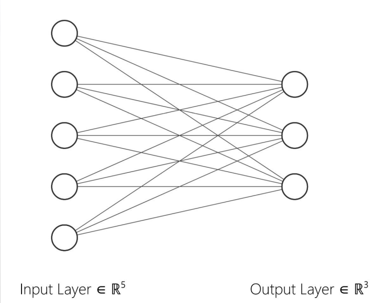
$\mathbf{v} \in R^{1 \times 5}$ $\quad$ $\mathbf{M} \in R^{5 \times 3}$ $\quad$ $\mathbf{v} \in R^{3}$

$Input_1$
$Input_2$
$Input_3$
$Input_4$
$Input_5$

X

$W^{\{1, 1\}}$	$W^{\{2, 1\}}$	$W^{\{3, 1\}}$
$W^{\{1, 2\}}$	$W^{\{2, 2\}}$	$W^{\{3, 2\}}$
$W^{\{1, 3\}}$	$W^{\{2, 3\}}$	$W^{\{3, 3\}}$
$W^{\{1, 4\}}$	$W^{\{2, 4\}}$	$W^{\{3, 4\}}$
$W^{\{1, 5\}}$	$W^{\{2, 5\}}$	$W^{\{3, 5\}}$

=

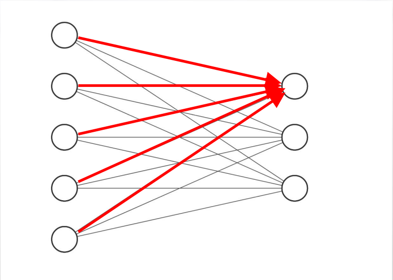


$W^{\{\text{Output Node Number, Input Node Number}\}}$

Let's use this network as an example

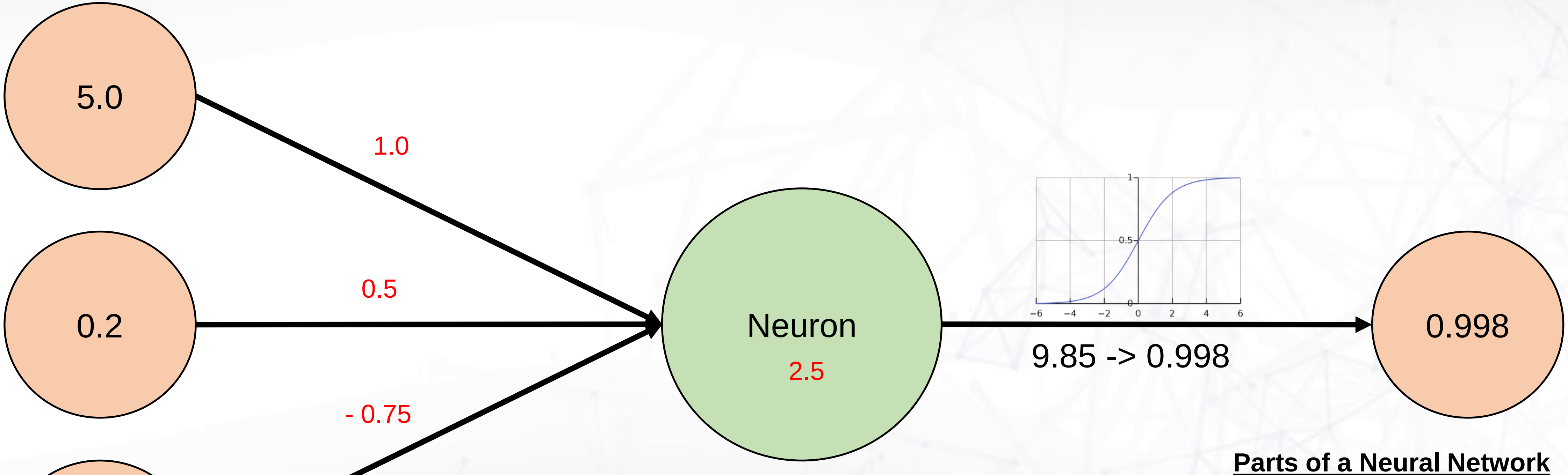
$$\mathbf{v} \in R^{1 \times 5} \qquad \mathbf{M} \in R^{5 \times 3} \qquad \mathbf{v} \in R^{3}$$

$$\begin{bmatrix} \text{Input}_1 \\ \text{Input}_2 \\ \text{Input}_3 \\ \text{Input}_4 \\ \text{Input}_5 \end{bmatrix} \times \begin{bmatrix} W^{\{1, 1\}} & W^{\{2, 1\}} & W^{\{3, 1\}} \\ W^{\{1, 2\}} & W^{\{2, 2\}} & W^{\{3, 2\}} \\ W^{\{1, 3\}} & W^{\{2, 3\}} & W^{\{3, 3\}} \\ W^{\{1, 4\}} & W^{\{2, 4\}} & W^{\{3, 4\}} \\ W^{\{1, 5\}} & W^{\{2, 5\}} & W^{\{3, 5\}} \end{bmatrix} = \begin{bmatrix} I_1W^{\{1, 1\}} + I_2W^{\{1, 2\}} + I_3W^{\{1, 3\}} + I_4W^{\{1, 4\}} + I_5W^{\{1, 5\}} \\ \\ \\ \end{bmatrix}$$



$W^{\{\text{Output Node Number, Input Node Number}\}}$

The Neuron



$$\text{Output} = Sigmoid(\sum(Wi \ * Input\ i) + b)$$

= (5.0 * 1.0) + (0.2 * 0.5) + (- 3 * - 0.75) + 2.5

= 5.0 + 0.1 + 2.25 + 2.5

= 9.85

Parts of a Neural Network

1. Neuron
2. Inputs (any number of them)
3. Output
4. Weights <- Parameters
5. Bias <- Parameter
6. NonLinearity

Remember: Parameters are just numbers

Let's use this network as an example

$\mathbf{v} \in R^{1 \times 5}$ $\times$ $\mathbf{M} \in R^{5 \times 3}$ $=$ $\mathbf{v} \in R^{3}$

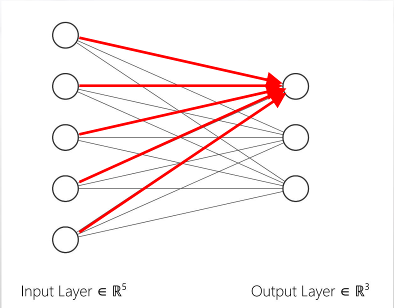
$Input_1$
$Input_2$
$Input_3$
$Input_4$
$Input_5$

x

$W^{\{1, 1\}}$	$W^{\{2, 1\}}$	$W^{\{3, 1\}}$
$W^{\{1, 2\}}$	$W^{\{2, 2\}}$	$W^{\{3, 2\}}$
$W^{\{1, 3\}}$	$W^{\{2, 3\}}$	$W^{\{3, 3\}}$
$W^{\{1, 4\}}$	$W^{\{2, 4\}}$	$W^{\{3, 4\}}$
$W^{\{1, 5\}}$	$W^{\{2, 5\}}$	$W^{\{3, 5\}}$

=

$I_1W^{\{1, 1\}} + I_2W^{\{1, 2\}} + I_3W^{\{1, 3\}} + I_4W^{\{1, 4\}} + I_5W^{\{1, 5\}}$



$W^{\{\text{Output Node Number, Input Node Number}\}}$

Let’s use this network as an example

BRAINTANK DEEP LEARNING

$\mathbf{v} \in R^{1 \times 5}$ $\mathbf{M} \in R^{5 \times 3}$ $\mathbf{v} \in R^{3}$

$Input_1$
$Input_2$
$Input_3$
$Input_4$
$Input_5$

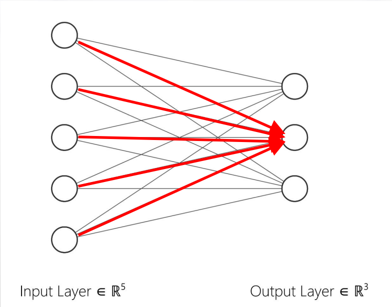
x

$W^{\{1, 1\}}$	$W^{\{2, 1\}}$	$W^{\{3, 1\}}$
$W^{\{1, 2\}}$	$W^{\{2, 2\}}$	$W^{\{3, 2\}}$
$W^{\{1, 3\}}$	$W^{\{2, 3\}}$	$W^{\{3, 3\}}$
$W^{\{1, 4\}}$	$W^{\{2, 4\}}$	$W^{\{3, 4\}}$
$W^{\{1, 5\}}$	$W^{\{2, 5\}}$	$W^{\{3, 5\}}$

=

$I_1W^{\{1, 1\}} + I_2W^{\{1, 2\}} + I_3W^{\{1, 3\}} + I_4W^{\{1, 4\}} + I_5W^{\{1, 5\}}$
$I_1W^{\{2, 1\}} + I_2W^{\{2, 2\}} + I_3W^{\{2, 3\}} + I_4W^{\{2, 4\}} + I_5W^{\{2, 5\}}$

$W^{\{Output\ Node\ Number,\ Input\ Node\ Number\}}$



Let’s use this network as an example

$\mathbf{v} \in R^{1 \times 5}$

$Input_1$
$Input_2$
$Input_3$
$Input_4$
$Input_5$

x

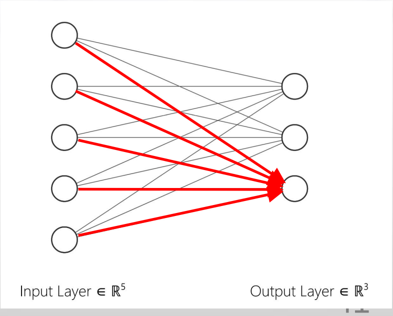
$\mathbf{M} \in R^{5 \times 3}$

$W^{\{1, 1\}}$	$W^{\{2, 1\}}$	$W^{\{3, 1\}}$
$W^{\{1, 2\}}$	$W^{\{2, 2\}}$	$W^{\{3, 2\}}$
$W^{\{1, 3\}}$	$W^{\{2, 3\}}$	$W^{\{3, 3\}}$
$W^{\{1, 4\}}$	$W^{\{2, 4\}}$	$W^{\{3, 4\}}$
$W^{\{1, 5\}}$	$W^{\{2, 5\}}$	$W^{\{3, 5\}}$

=

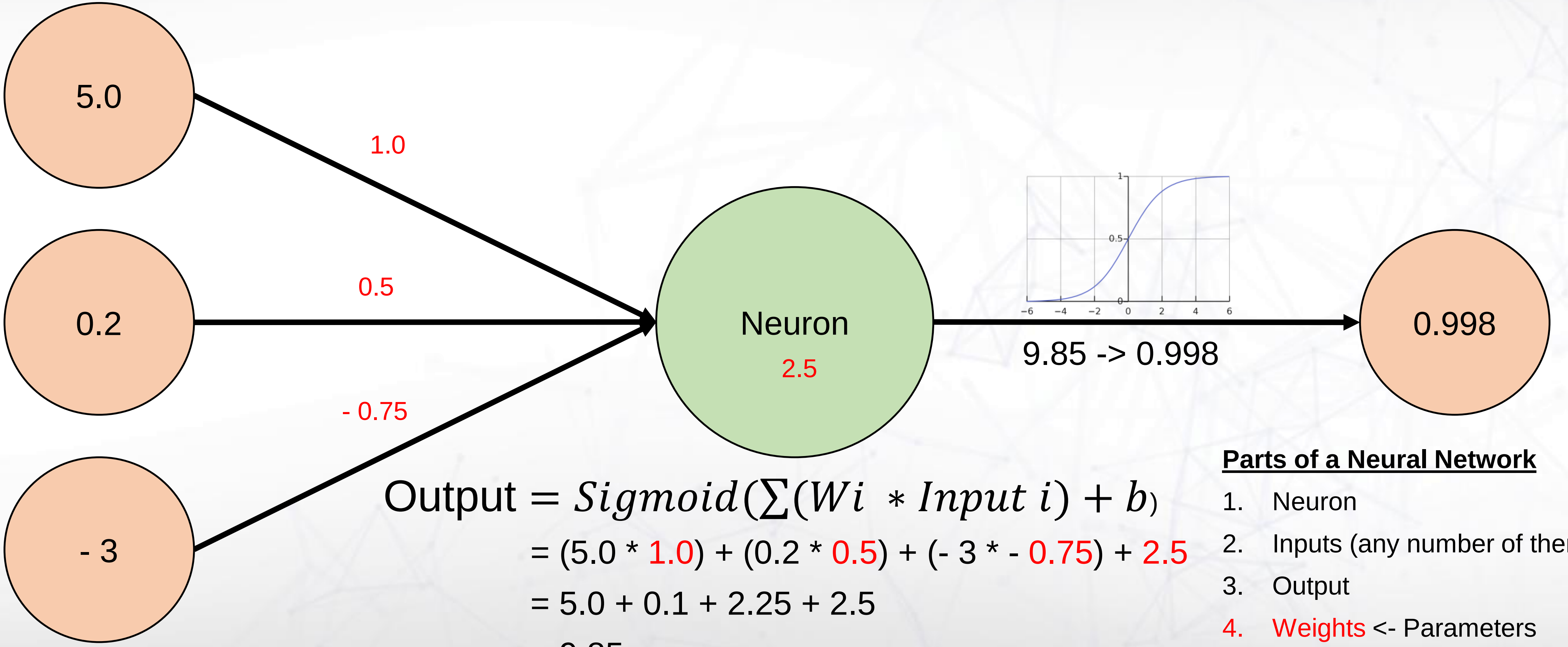
$\mathbf{v} \in R^{3}$

$I_1W^{\{1, 1\}} + I_2W^{\{1, 2\}} + I_3W^{\{1, 3\}} + I_4W^{\{1, 4\}} + I_5W^{\{1, 5\}}$
$I_1W^{\{2, 1\}} + I_2W^{\{2, 2\}} + I_3W^{\{2, 3\}} + I_4W^{\{2, 4\}} + I_5W^{\{2, 5\}}$
$I_1W^{\{3, 1\}} + I_2W^{\{3, 2\}} + I_3W^{\{3, 3\}} + I_4W^{\{3, 4\}} + I_5W^{\{3, 5\}}$



$W^{\{\text{Output Node Number, Input Node Number}\}}$

The Neuron



Output $= Sigmoid(\sum(Wi \ * Input\ i) + b)$

= (5.0 * 1.0) + (0.2 * 0.5) + (- 3 * - 0.75) + 2.5

= 5.0 + 0.1 + 2.25 + 2.5

= 9.85

Remember: Parameters are just numbers

Parts of a Neural Network

1. Neuron
2. Inputs (any number of them)
3. Output
4. Weights <- Parameters
5. Bias <- Parameter
6. NonLinearity

Let's use this network as an example

$\mathbf{v} \in R^{1 \times 5}$

Input_1
Input_2
Input_3
Input_4
Input_5

x

$\mathbf{M} \in R^{5 \times 3}$

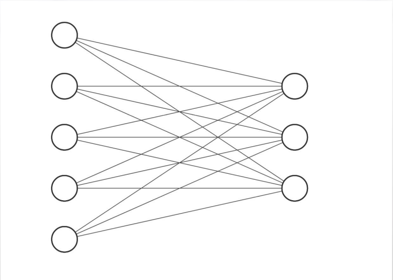
$W^{\{1, 1\}}$	$W^{\{2, 1\}}$	$W^{\{3, 1\}}$
$W^{\{1, 2\}}$	$W^{\{2, 2\}}$	$W^{\{3, 2\}}$
$W^{\{1, 3\}}$	$W^{\{2, 3\}}$	$W^{\{3, 3\}}$
$W^{\{1, 4\}}$	$W^{\{2, 4\}}$	$W^{\{3, 4\}}$
$W^{\{1, 5\}}$	$W^{\{2, 5\}}$	$W^{\{3, 5\}}$

+

=

$\mathbf{v} \in R^{3}$

$I_1W^{\{1, 1\}} + I_2W^{\{1, 2\}} + I_3W^{\{1, 3\}} + I_4W^{\{1, 4\}} + I_5W^{\{1, 5\}}$
$I_1W^{\{2, 1\}} + I_2W^{\{2, 2\}} + I_3W^{\{2, 3\}} + I_4W^{\{2, 4\}} + I_5W^{\{2, 5\}}$
$I_1W^{\{3, 1\}} + I_2W^{\{3, 2\}} + I_3W^{\{3, 3\}} + I_4W^{\{3, 4\}} + I_5W^{\{3, 5\}}$

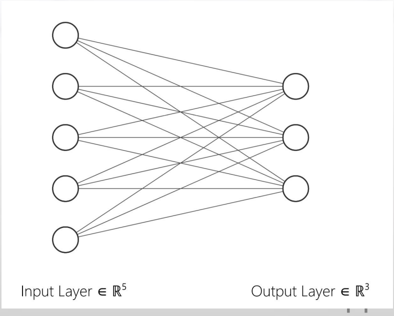


$W^{\{\text{Output Node Number, Input Node Number}\}}$

Let's use this network as an example

$\mathbf{v} \in R^{1 \times 5}$ $\quad$ $\mathbf{M} \in R^{5 \times 3}$ $\quad$ $\mathbf{v} \in R^{3}$

$$
\begin{bmatrix} \text{Input}_1 \\ \text{Input}_2 \\ \text{Input}_3 \\ \text{Input}_4 \\ \text{Input}_5 \end{bmatrix}
\times
\begin{bmatrix}
W^{\{1,1\}} & W^{\{2,1\}} & W^{\{3,1\}} \\
W^{\{1,2\}} & W^{\{2,2\}} & W^{\{3,2\}} \\
W^{\{1,3\}} & W^{\{2,3\}} & W^{\{3,3\}} \\
W^{\{1,4\}} & W^{\{2,4\}} & W^{\{3,4\}} \\
W^{\{1,5\}} & W^{\{2,5\}} & W^{\{3,5\}}
\end{bmatrix}
+
\begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}
=
\begin{bmatrix}
I_1W^{\{1,1\}} + I_2W^{\{1,2\}} + I_3W^{\{1,3\}} + I_4W^{\{1,4\}} + I_5W^{\{1,5\}} + B_1 \\
I_1W^{\{2,1\}} + I_2W^{\{2,2\}} + I_3W^{\{2,3\}} + I_4W^{\{2,4\}} + I_5W^{\{2,5\}} + B_2 \\
\boxed{I_1W^{\{3,1\}} + I_2W^{\{3,2\}} + I_3W^{\{3,3\}} + I_4W^{\{3,4\}} + I_5W^{\{3,5\}} + B_3}
\end{bmatrix}
$$



$W^{\{\text{Output Node Number, Input Node Number}\}}$

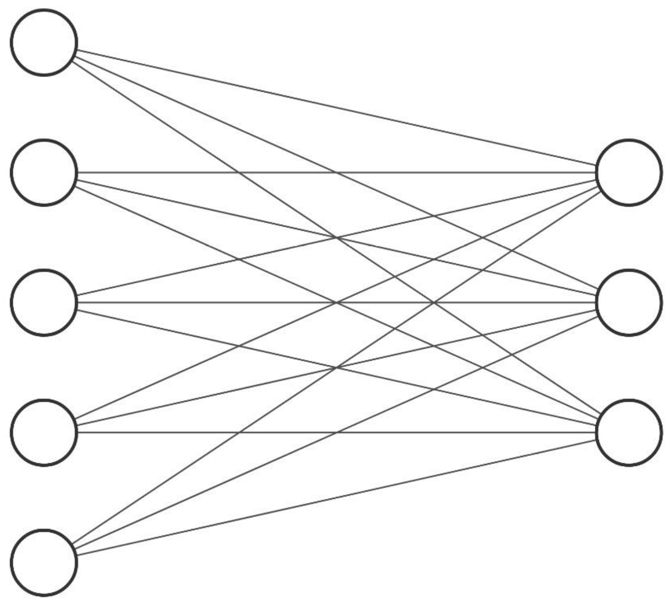
In summary…

$$y = xA^T + b$$

Why do we use matrix multiplication?

1. **It’s fast**
2. **It’s easily differentiable**
3. **It produces the same results as a series of linked lists**

Let’s use this network as an example

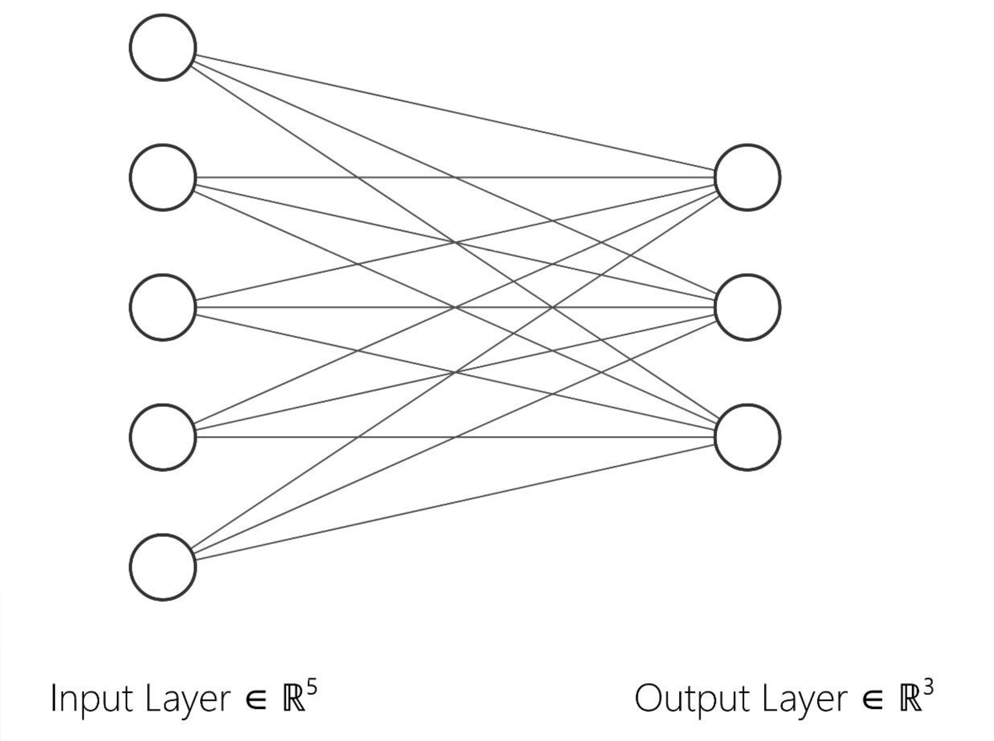


This network has 5 inputs and 3 outputs

```
self.layer = torch.nn.Linear(5, 3)
```

Vector (Array) of 5 numbers → Neural Network Implementation → Vector (Array) of 3 numbers

Let's use this network as an example



This network has 5 inputs and 3 outputs

```
self.layer = custom_net_layer(5, 3)
```

Vector (Array) of 5 numbers → Neural Network Implementation → Vector (Array) of 3 numbers

Initializing and basic operations

A tensor can be constructed from a Python `list` or sequence using the `torch.tensor()` constructor:

```
>>> torch.tensor([[1., -1.], [1., -1.]])
tensor([[ 1.0000, -1.0000],
        [ 1.0000, -1.0000]])
>>> torch.tensor(np.array([[1, 2, 3], [4, 5, 6]]))
tensor([[ 1,  2,  3],
        [ 4,  5,  6]])
```

• WARNING

`torch.tensor()` always copies `data`. If you have a Tensor `data` and just want to change its `requires_grad` flag, use `requires_grad_()` or `detach()` to avoid a copy. If you have a numpy array and want to avoid a copy, use `torch.as_tensor()`.

A tensor of specific data type can be constructed by passing a `torch.dtype` and/or a `torch.device` to a constructor or tensor creation op:

```
>>> torch.zeros([2, 4], dtype=torch.int32)
tensor([[ 0,  0,  0,  0],
        [ 0,  0,  0,  0]], dtype=torch.int32)
>>> cuda0 = torch.device('cuda:0')
>>> torch.ones([2, 4], dtype=torch.float64, device=cuda0)
tensor([[ 1.0000,  1.0000,  1.0000,  1.0000],
        [ 1.0000,  1.0000,  1.0000,  1.0000]], dtype=torch.float64, device='cuda:0')
```

h.html#torch.Tensor.detach

Classification Loss

A new kind of loss function

Week 1 & 2:

Question:

Given a cost weight, what **number** should the cow's cost be?

Given information about a diamond, what **number** should the diamond's value be

How We Approach the Loss:

Model's Prediction vs Ground Truth:

We need to make the model's prediction as close to the ground truth as possible.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

A new kind of loss function

Week 1 & 2:

Question:

Given a cost weight, what **number** should the cow's cost be?

Given information about a diamond, what **number** should the diamond's value be

How We Approach the Loss:

Model's Prediction vs Ground Truth:

We need to make the model's prediction as close to the ground truth as possible.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Week 3:

Question:

Given a list of pixels, which number do those pixels correlate to

How We Approach the Loss:

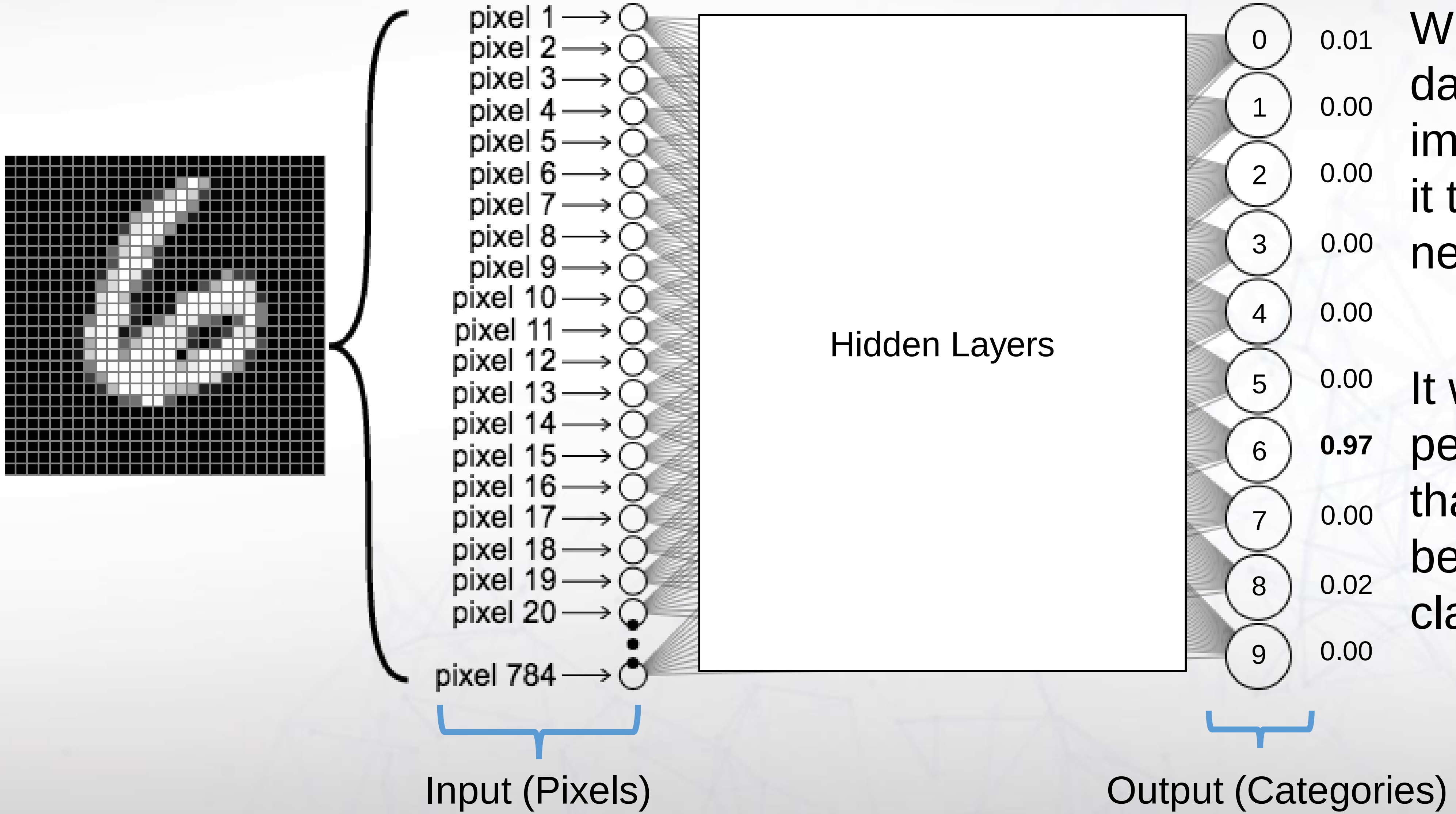
Model's Prediction vs Ground Truth:

Model will output 10 different numbers correlating to the likelihood it believes the picture to belong to a certain classification.

We need to compare that to the image's actual label

CrossEntropy Loss

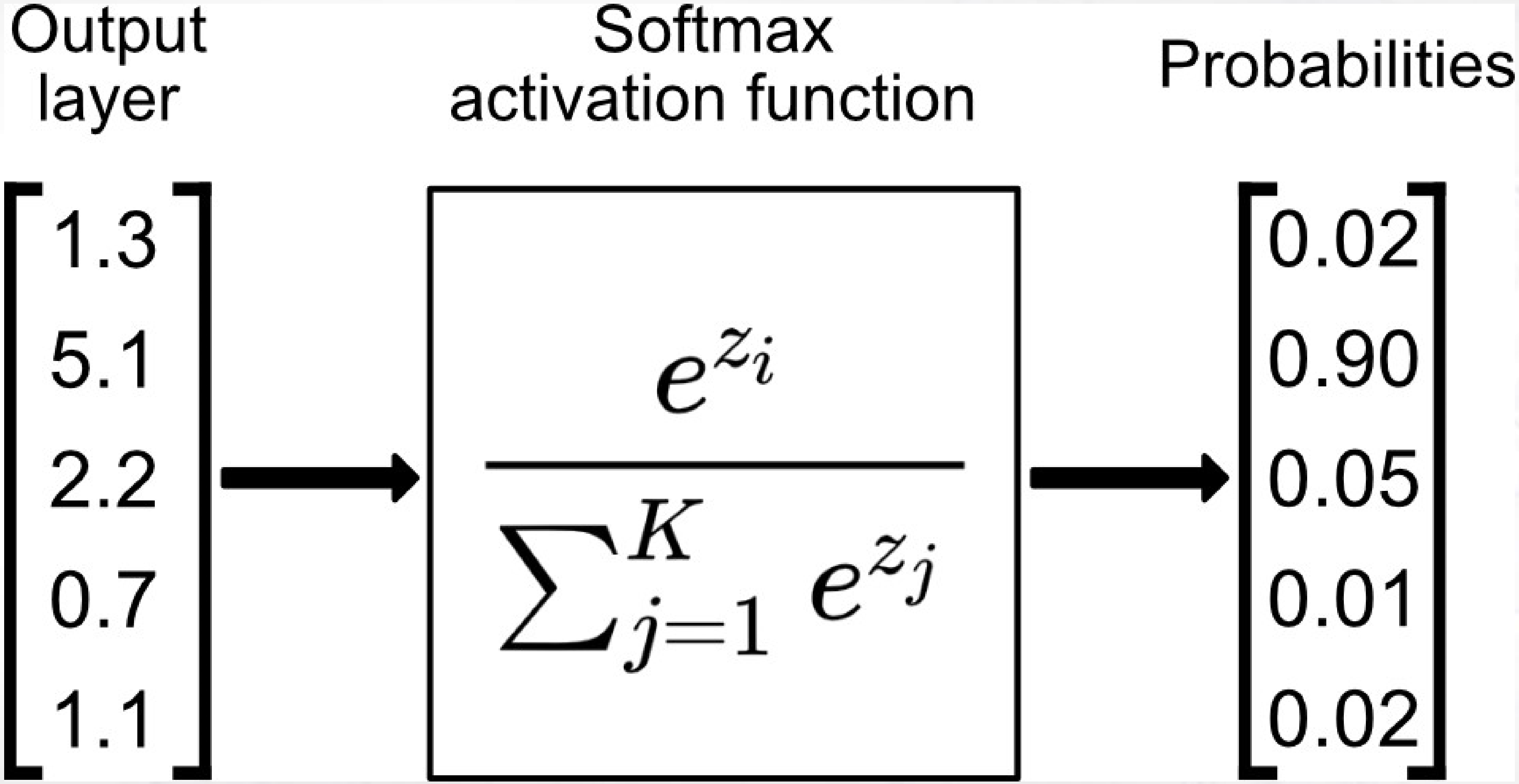
How we going to tackle this problem



We will the pixel data of each image, and will run it through a neural network.

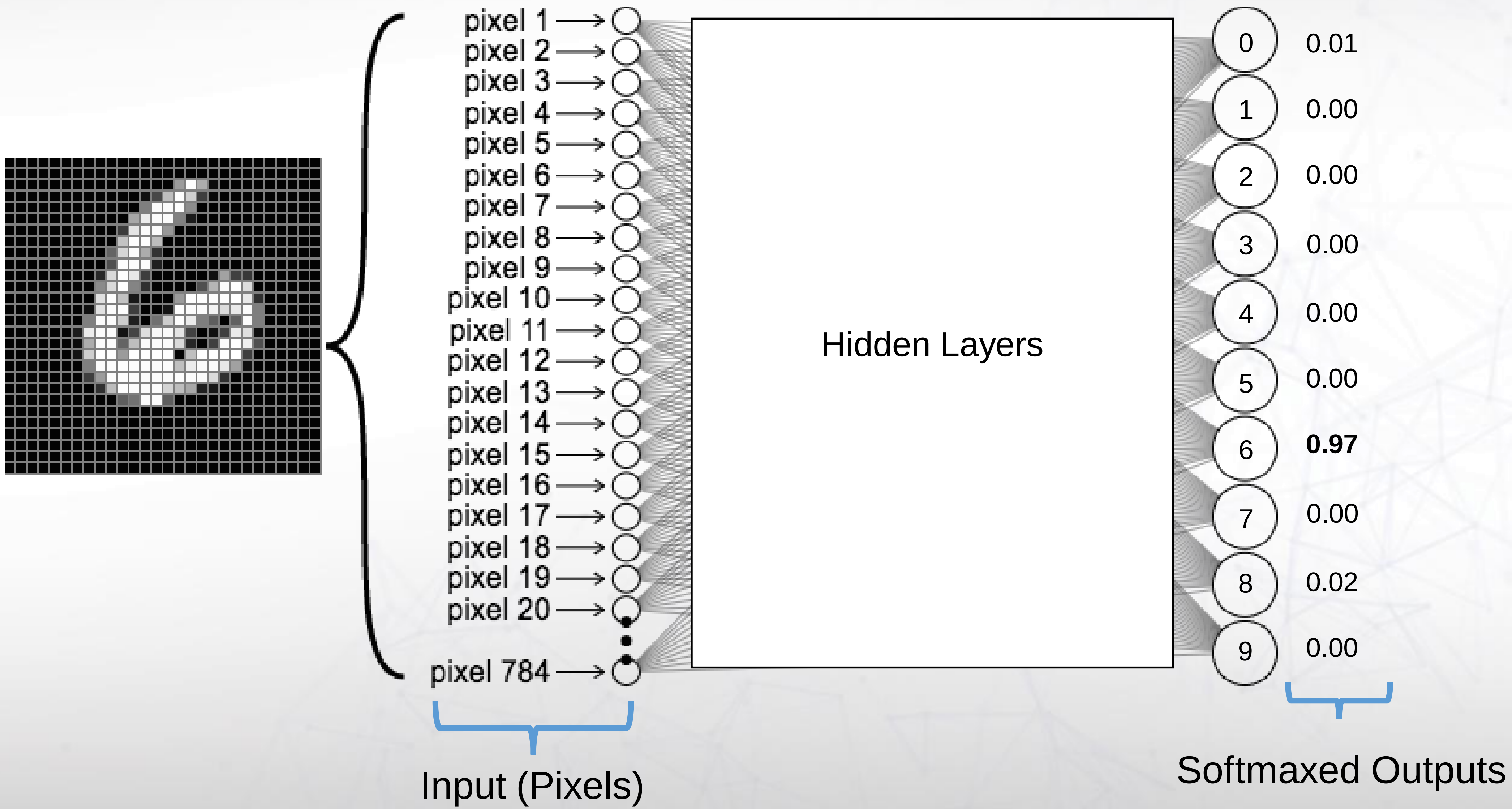
It will output percent likelihood that that image belongs to each class.

What is SoftMax

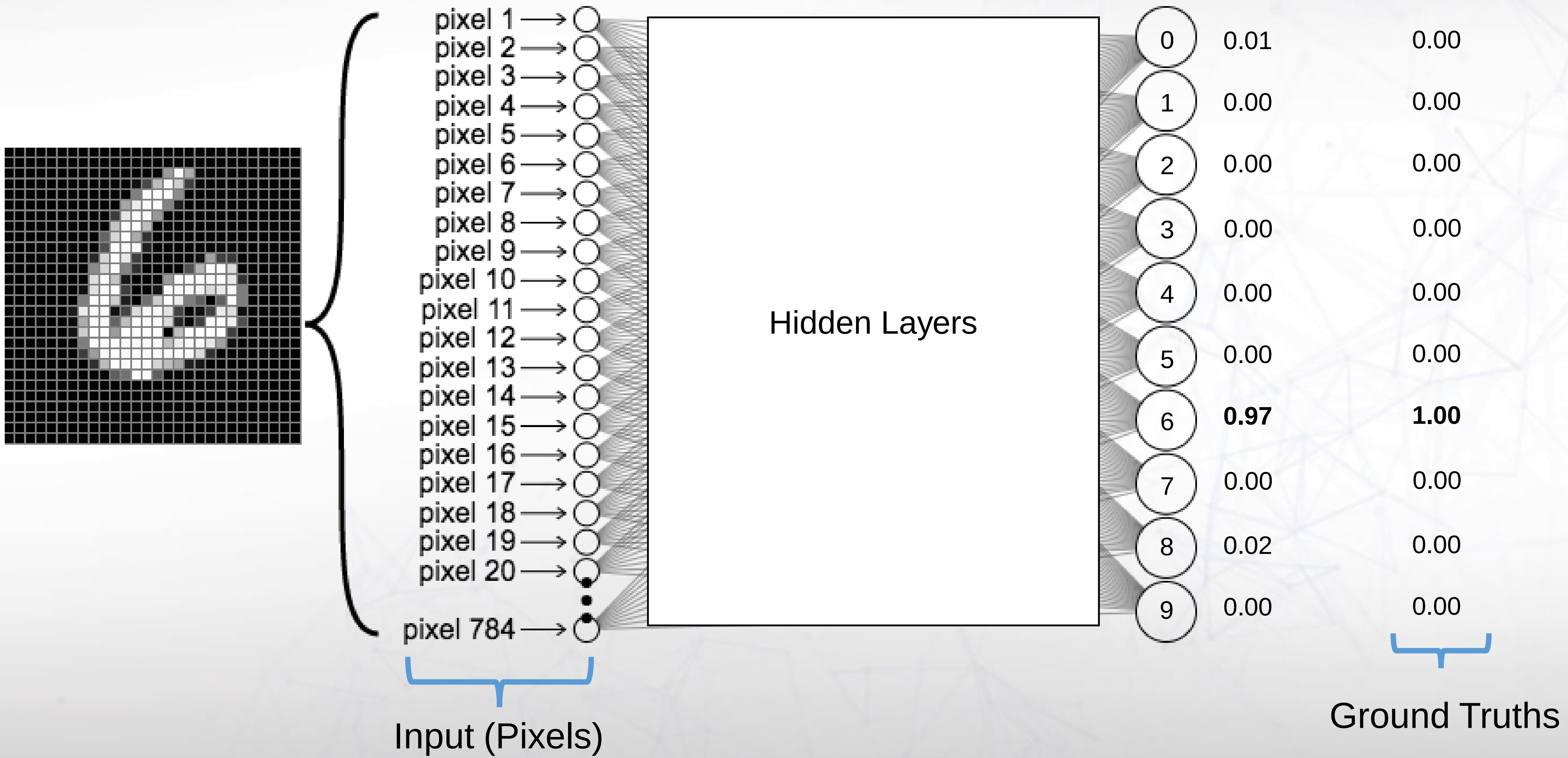


Models can output a large range of numbers, the SoftMax function will transform those values into “percentages”. They will all add up to 1. This is how we will evaluate what class the images belong to.

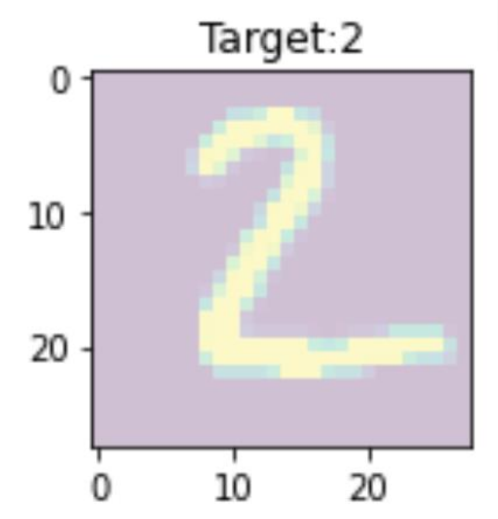
How we going to tackle this problem



How we going to tackle this problem



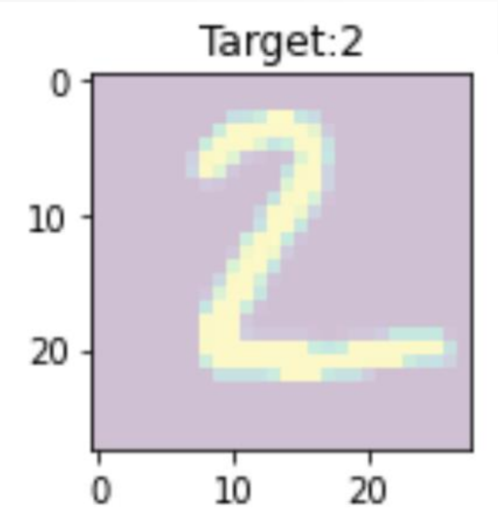
CrossEntropy Loss



This correlates to a "number 2". We can also see that the model has done a very poor job of classifying it

#	Model Prediction	Ground Truth
0.	0.10	0.00
1.	0.10	0.00
2.	0.10	1.00
3.	0.10	0.00
4.	0.10	0.00
5.	0.10	0.00
6.	0.10	0.00
7.	0.10	0.00
8.	0.10	0.00
9.	0.10	0.00

CrossEntropy Loss



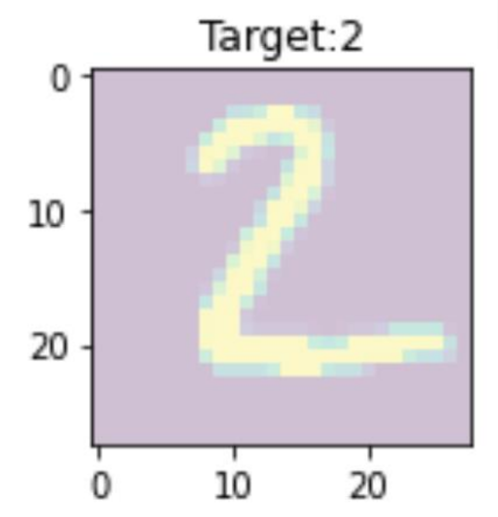
This correlates to a "number 2". We can also see that the model has done a very poor job of classifying it

#	Model Prediction	Ground Truth
0.	0.10	0.00
1.	0.10	0.00
2.	0.10	1.00
3.	0.10	0.00
4.	0.10	0.00
5.	0.10	0.00
6.	0.10	0.00
7.	0.10	0.00
8.	0.10	0.00
9.	0.10	0.00

CrossEntropy Loss Function

Loss: 2.3446

CrossEntropy Loss



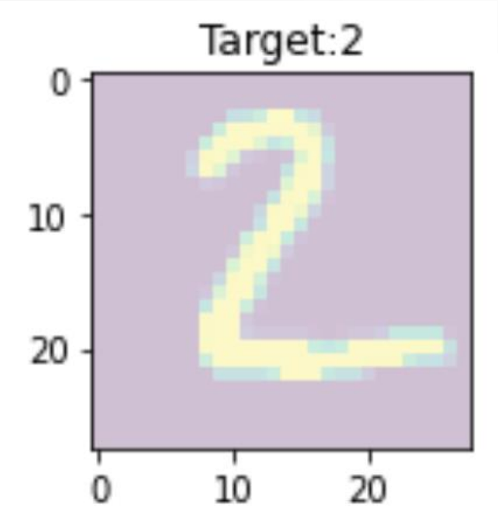
The model is now doing a better job at predicting “2”, but it could be better.

#	Model Prediction	Ground Truth
0.	0.12	0.00
1.	0.11	0.00
2.	0.46	1.00
3.	0.01	0.00
4.	0.03	0.00
5.	0.04	0.00
6.	0.10	0.00
7.	0.03	0.00
8.	0.05	0.00
9.	0.05	0.00

CrossEntropy Loss Function

Loss: 0.8654

CrossEntropy Loss



The model does an amazing job predicting "2"

#	Model Prediction	Ground Truth
0.	0.00	0.00
1.	0.00	0.00
2.	0.99	1.00
3.	0.00	0.00
4.	0.00	0.00
5.	0.01	0.00
6.	0.00	0.00
7.	0.00	0.00
8.	0.00	0.00
9.	0.00	0.00

CrossEntropy Loss Function

Loss: 0.0001

CrossEntropy Loss

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -\sum_{c=1}^{C} w_c \log \frac{\exp(x_{n,c})}{\exp(\sum_{i=1}^{C} x_{n,i})} y_{n,c}$$

The CrossEntropy Loss function is quite involved and tough to understand. CrossEntropy loss will be provided to you when coding today.

Even though it is a tough function it shares many of the same properties as other loss functions you are comfortable with.

1. A more incorrect algorithm will create a high level of loss on average
2. A more correct algorithm will create a low level of loss on average
3. The optimizer's job is to minimize loss in the model

You should also understand it's purpose; Namely, in classification problems.

Mini-Batching

The New Deep Learning Flow:

1. Initialize your model. Randomly assign your parameters a value.
2. Grab **a single entry** from the dataset. An input and the ground truth associated with that input.
3. Take that input and put it through the model and save the result.
4. Use the loss function to measure how different the result is from the ground truth.
5. Give the OPTIMIZER, it will then measure how it needs to change the parameters (numbers we change to get a desired result).
6. Repeat steps 2-5 many many times, until the model can perform the task you are asking it to do

The New Deep Learning Flow

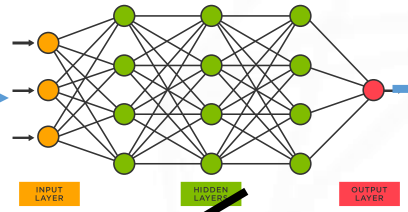
1. Initialize your model. Randomly assign your parameters a value.
2. Grab **a minibatch of data** from the dataset. A few inputs and their **associated groundtruths with those inputs**.
3. Take the inputs and put it through the model and save the results.
4. Use the loss function to measure how different the results are from the ground truths.
5. Give the OPTIMIZER, it will then measure how it needs to change the parameters (numbers we change to get a desired result).
6. Repeat steps 2-5 many many times, until the model can perform the task you are asking it to do

The New Deep Learning Flow

Dataset:

Price of Diamond	Input Features
0.18	$(24)_1$
0.52	$(24)_2$
1.00	$(24)_3$
0.00	$(24)_4$
0.35	$(24)_5$
…	

Model:

$(24)_1$ →  → 1.3

*modified

Result:

1.3

$2,039031.52

Ground Truth:

0.18

$4,371.04

Loss Function (measure of model success):

$L = (\text{Result} - \text{Ground Truth})^2$

$L = (1.3 - 0.18)^2$

$L = 1.254$

OPTIMIZER:

The OPTIMIZER will look at the loss, and use it to determine how to adjust the models parameters. We want the loss to be small and the model will try and reduce it for us.

The New Deep Learning Flow

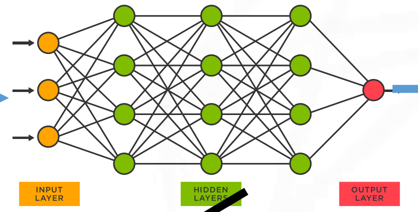
BRAINTANK DEEP LEARNING

Dataset:

Price of Diamond	Input Features
0.18	$(24)_1$
0.52	$(24)_2$
1.00	$(24)_3$
0.00	$(24)_4$
0.35	$(24)_5$
…	

Model:

$(24)_1$
$(24)_2$
$(24)_3$



1.3
0.4
5.7

*modified

Result:

1.3
0.4
5.7

Ground Truth:

0.18
0.52
1.00

Loss Function (measure of model success):

L = (Result – Ground Truth)^2
L = (1.3 – 0.18)^2
L = 1.254

L = (Result – Ground Truth)^2
L = (0.4 – 0.52)^2
L = 0.0144

L = (Result – Ground Truth)^2
L = (5.7 – 1.00)^2
L = 22.09

Average

OPTIMIZER:

The OPTIMIZER will look at the loss, and use it to determine how to adjust the models parameters. We want the loss to be small and the model will try and reduce it for us.

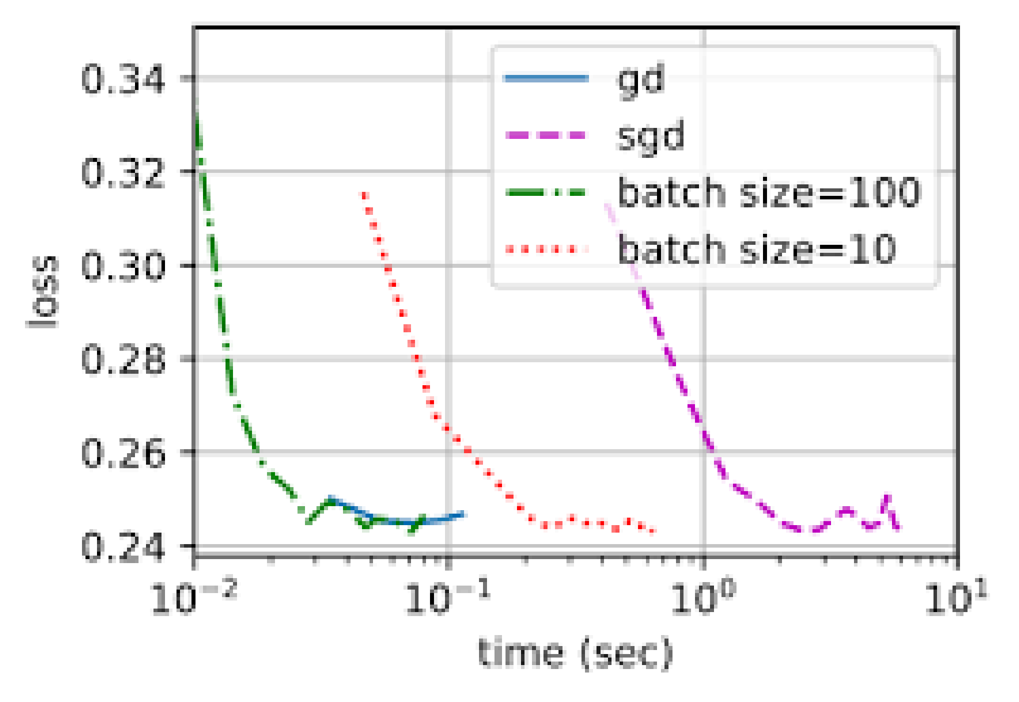
Why Use Mini-Batching?

Mini-Batching is done to make training **FASTER.**

The same data is being inputted, and each piece of data in the batch is independent from the other pieces of data. As a result, the model will still learn the same way.

However, instead of calculating loss and optimizing on every piece of data, it does it in bunches, speeding the process up.

The standard length of a batch is 32. But, it can very depending on a number of factors.



Saving and Loading Model

PyTorch Makes Saving and Loading Easy

Save:

```
torch.save(model.state_dict(), PATH)
```

Load:

```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

PyTorch makes it easy to train once, and to test multiple times. Often, models are saved after every couple of epochs. State_dict() refers to the model's state dictionary, which contains the parameters for the model.

The Deep Learning Flow

The New Deep Learning Flow

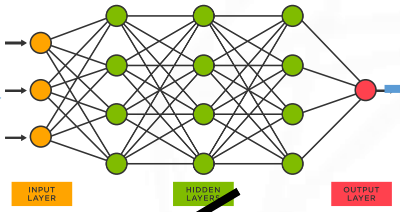
BRAINTANK DEEP LEARNING

Dataset:

Image Label	Input Features
“9”	$(1, 28, 28)_1$
“6”	$(1, 28, 28)_2$
“0”	$(1, 28, 28)_3$
“4”	$(1, 28, 28)_4$
“9”	$(1, 28, 28)_5$
…	

Mini-Batch Size: 3

Model:

(3, 748) →  → (3, 10)

*modified

Result:

$(10)_{1P}$

$(10)_{2P}$

$(10)_{3P}$

Ground Truth:

$(10)_{1\text{->”9”}}$

$(10)_{2\text{->”6”}}$

$(10)_{3\text{->”0”}}$

Loss Function (measure of model success):

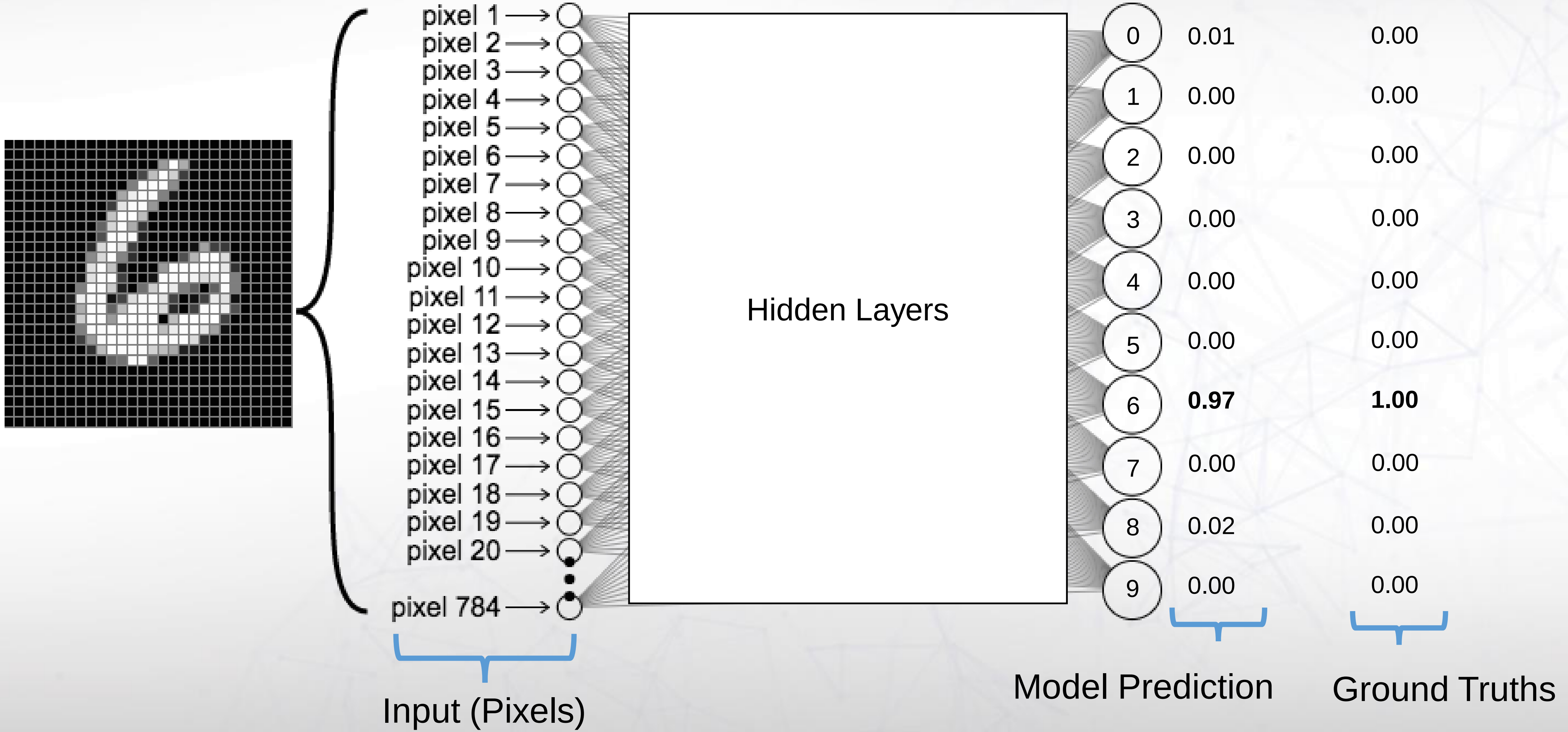
CrossEntropy Loss	CrossEntropy Loss	CrossEntropy Loss
L = 0.356	L = 0.564	L = 0.785

Average

OPTIMIZER:

The OPTIMIZER will look at the loss, and use it to determine how to adjust the models parameters. We want the loss to be small and the model will try and reduce it for us.

How we going to tackle this problem



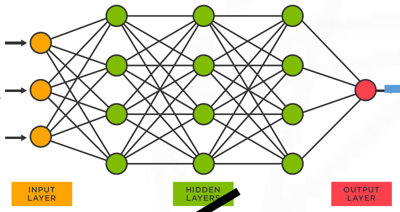
The New Deep Learning Flow

Dataset:

Image Label	Input Features
“9”	$(1, 28, 28)_1$
“6”	$(1, 28, 28)_2$
“0”	$(1, 28, 28)_3$
“4”	$(1, 28, 28)_4$
“9”	$(1, 28, 28)_5$
…	

Mini-Batch Size: 3

Model:

(3, 748) → 

→ (3, 10)

*modified

Result:

$(10)_{1P}$

$(10)_{2P}$

$(10)_{3P}$

Ground Truth:

$(10)_{1\text{->"9"}}$

$(10)_{2\text{->"6"}}$

$(10)_{3\text{->"0"}}$

Loss Function (measure of model success):

CrossEntropy Loss	CrossEntropy Loss	CrossEntropy Loss
L = 0.356	L = 0.564	L = 0.785

Average

OPTIMIZER:

The OPTIMIZER will look at the loss, and use it to determine how to adjust the models parameters. We want the loss to be small and the model will try and reduce it for us.

The New Deep Learning Flow

BRAINTANK DEEP LEARNING

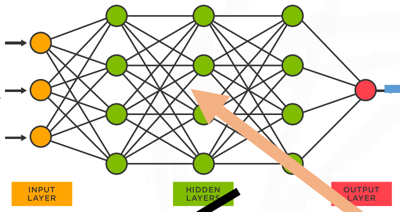
Dataset:

Image Label	Input Features
"9"	$(1, 28, 28)_1$
"6"	$(1, 28, 28)_2$
"0"	$(1, 28, 28)_3$
"4"	$(1, 28, 28)_4$
"9"	$(1, 28, 28)_5$
…	

Mini-Batch Size: 3

Model:

(3, 748)



(3, 10)

*modified

Result:

$(10)_{1P}$

$(10)_{2P}$

$(10)_{3P}$

Ground Truth:

$(10)_{1\text{->}"9"}$

$(10)_{2\text{->}"6"}$

$(10)_{3\text{->}"0"}$

Loss Function (measure of model success):

CrossEntropy Loss	CrossEntropy Loss	CrossEntropy Loss
L = 0.356	L = 0.564	L = 0.785

Average

OPTIMIZER:

The OPTIMIZER will look at the loss, and use it to determine how to adjust the models parameters. We want the loss to be small and the model will try and reduce it for us.

The New Deep Learning Flow

BRAINTANK DEEP LEARNING

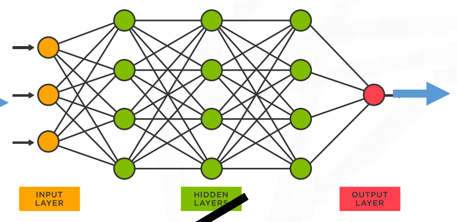
Dataset:

Image Label	Input Features
"9"	$(1, 28, 28)_1$
"6"	$(1, 28, 28)_2$
"0"	$(1, 28, 28)_3$
"4"	$(1, 28, 28)_4$
"9"	$(1, 28, 28)_5$
…	

Mini-Batch Size: 3

Model:

(3, 748)



(3, 10)

*modified

Result:

$(10)_{1P}$

$(10)_{2P}$

$(10)_{3P}$

Ground Truth:

$(10)_{1\text{->"9"}}$

$(10)_{2\text{->"6"}}$

$(10)_{3\text{->"0"}}$

Loss Function (measure of model success):

CrossEntropy Loss	CrossEntropy Loss	CrossEntropy Loss
L = 0.356	L = 0.564	L = 0.785

Average

OPTIMIZER:

The OPTIMIZER will look at the loss, and use it to determine how to adjust the models parameters. We want the loss to be small and the model will try and reduce it for us.

The New Deep Learning Flow

BRAINTANK
DEEP LEARNING

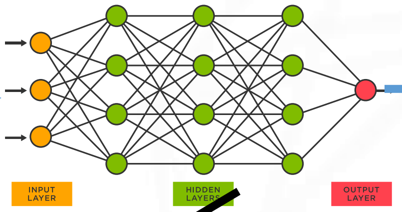
Dataset:

Image Label	Input Features
“9”	$(1, 28, 28)_1$
“6”	$(1, 28, 28)_2$
“0”	$(1, 28, 28)_3$
“4”	$(1, 28, 28)_4$
“9”	$(1, 28, 28)_5$
…	

Mini-Batch Size: 3

Model:

(3, 748) →



→ (3, 10)

*modified

Result:

$(10)_{4P}$

$(10)_{5P}$

$(10)_{6P}$

Ground Truth:

$(10)_{1\text{->”4”}}$

$(10)_{2\text{->”9”}}$

$(10)_{3\text{->”1”}}$

Loss Function (measure of model success):

CrossEntropy Loss	CrossEntropy Loss	CrossEntropy Loss
L = 0.0134	L = 0.0245	L = 0.0653

Average

OPTIMIZER:

The OPTIMIZER will look at the loss, and use it to determine how to adjust the models parameters. We want the loss to be small and the model will try and reduce it for us.

BRAINTANK
DEEP LEARNING