10/1: Neural Networks

Discord: https://discord.gg/68VpV6

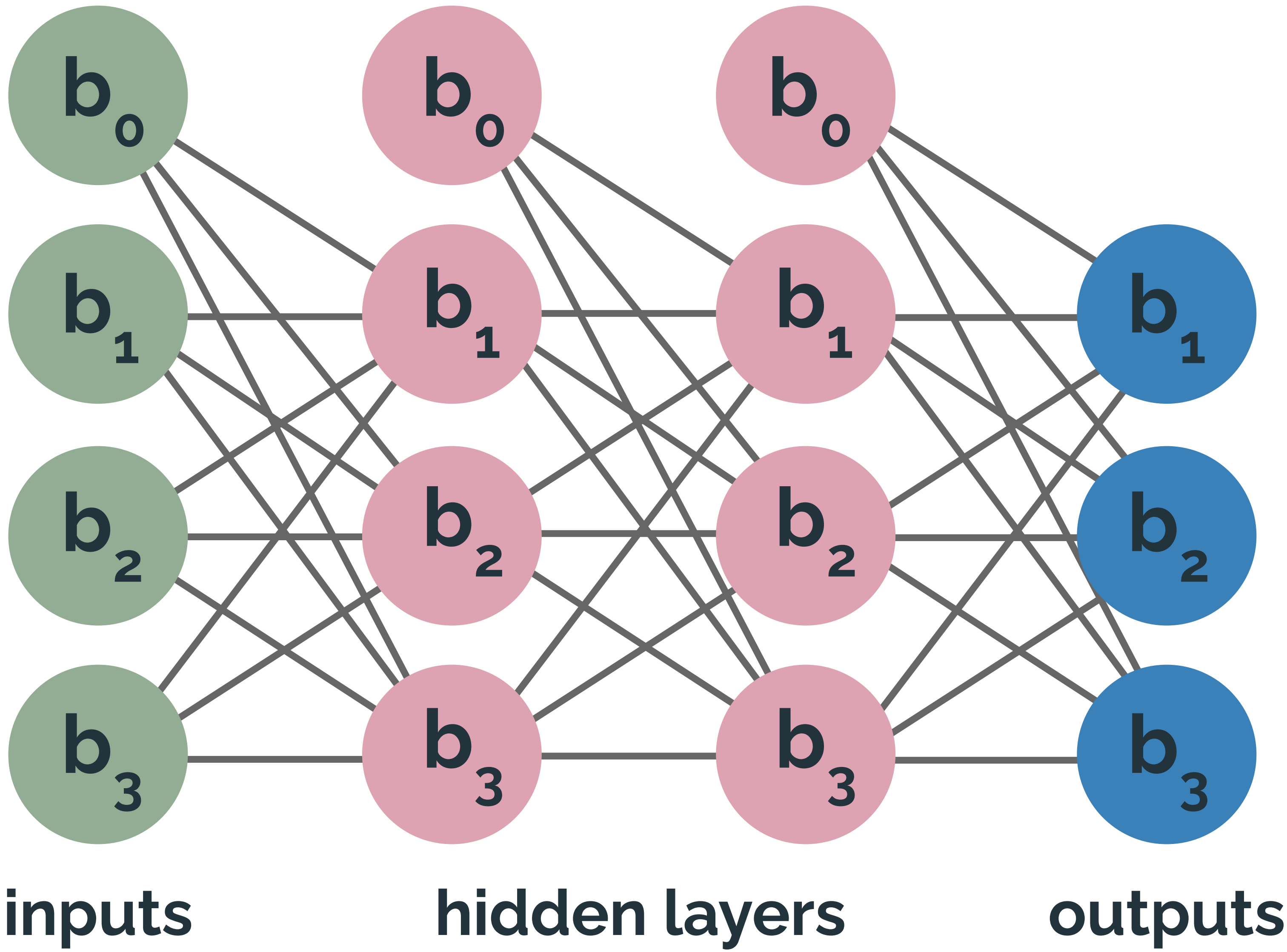
Neural Networks

How do we make predictions using complex data?

Neural Network Architecture

- A neural network contains layers of **neurons**, or **nodes**
- Each **node** can be represented as a circle
- Lines are drawn connecting these circles, called **weights**
- In general the value of a node is the sum of **its previous nodes** times **its previous weights**

Neural Network Architecture



Naive Example #1: Not Gate

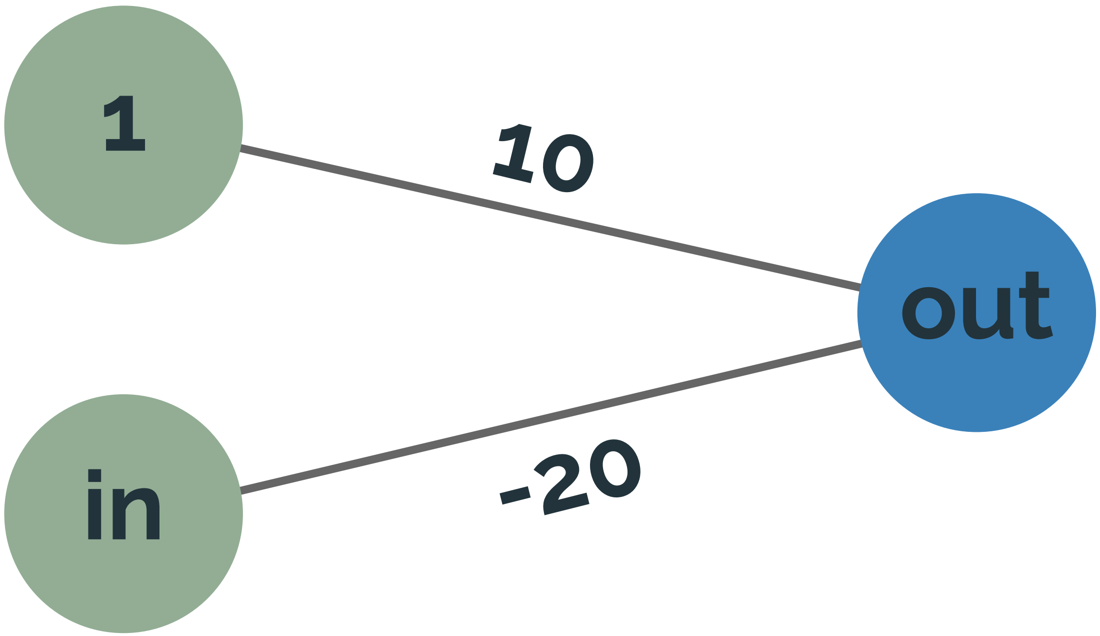
Below is the truth table for a **not gate**, which returns the opposite of the input:

Input	Output
0	1
1	0

While this is not particularly difficult to compute, we can still represent it as a neural network!

*Recall: Very positive values are 1 and very negative values are 0 as a result of the sigmoid function

Naive Example #1: Not Gate



`in = 0  ->  1(10) + 0(-20) = 10  ->  out = 1`

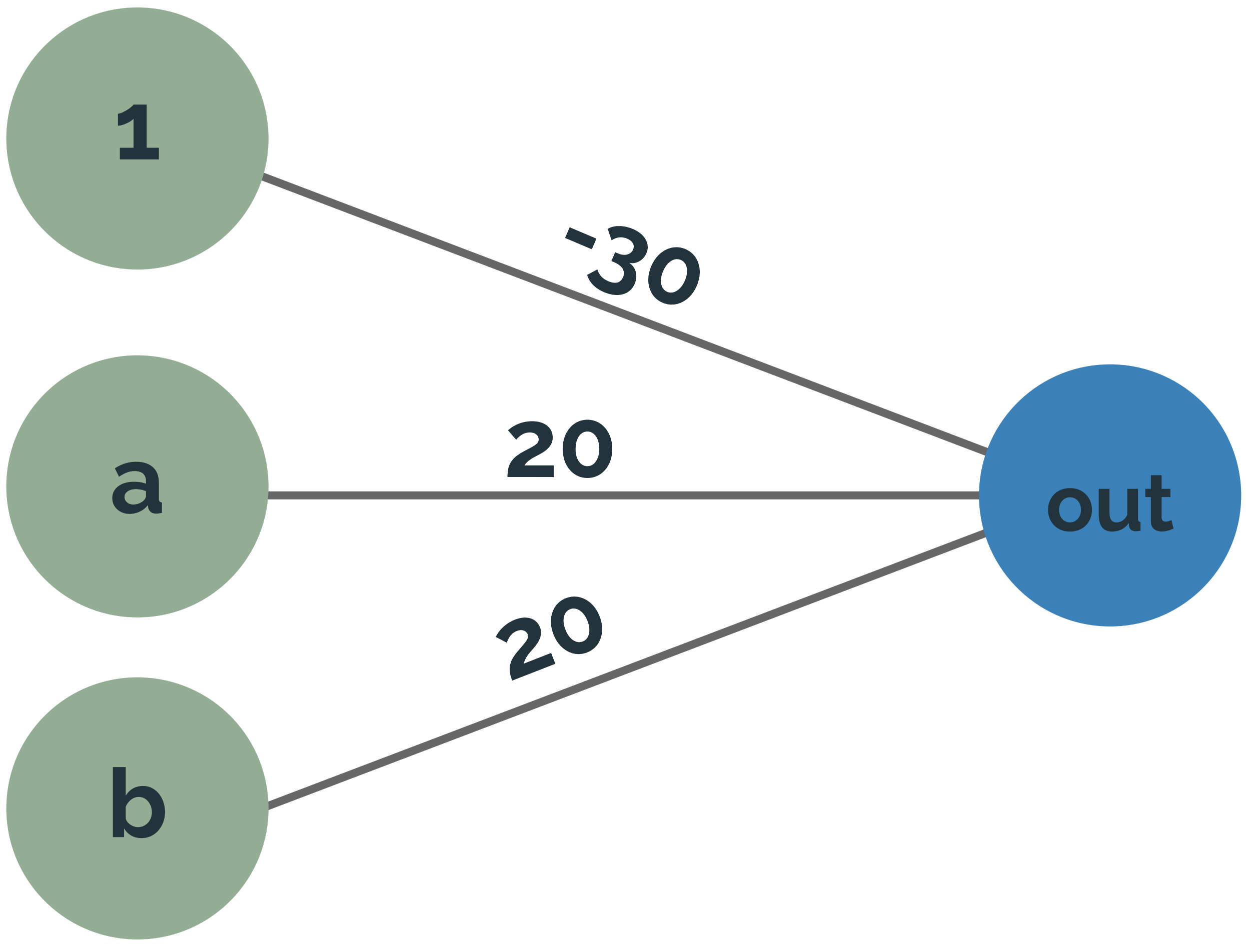
`in = 1  ->  1(10) + 1(-20) = -10  ->  out = 0`

Naive Example #2: And Gate

Below is the truth table for an **and gate**, which outputs 1 only if a and b are both 1

a	**b**	**out**
0	0	0
1	0	0
0	1	0
1	1	1

Naive Example #2: And Gate



*Note that **out** can only be positive if **a** and **b** are 1

Multiple Layer Example: Xnor Gate

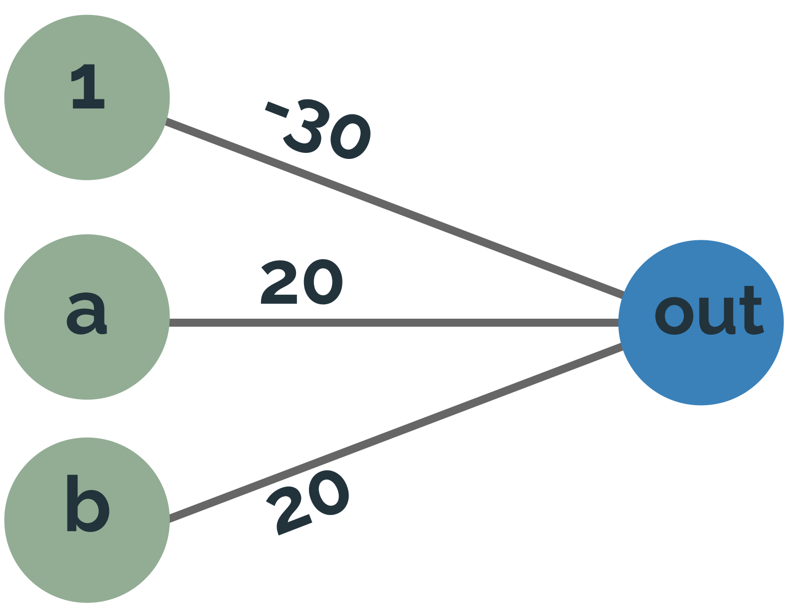
Below is the truth table for an **xnor gate**, which basically checks if a and b are equal

a	b	out
0	0	1
1	0	0
0	1	0
1	1	1

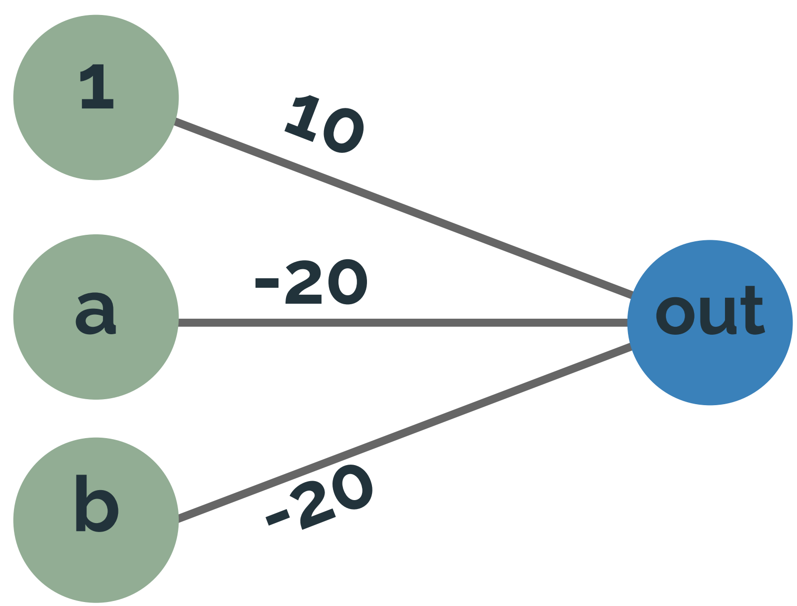
First Layer

xnor is logically equivalent to: (**a and b**) or (**not a and not b**)

<u>**a and b**</u>

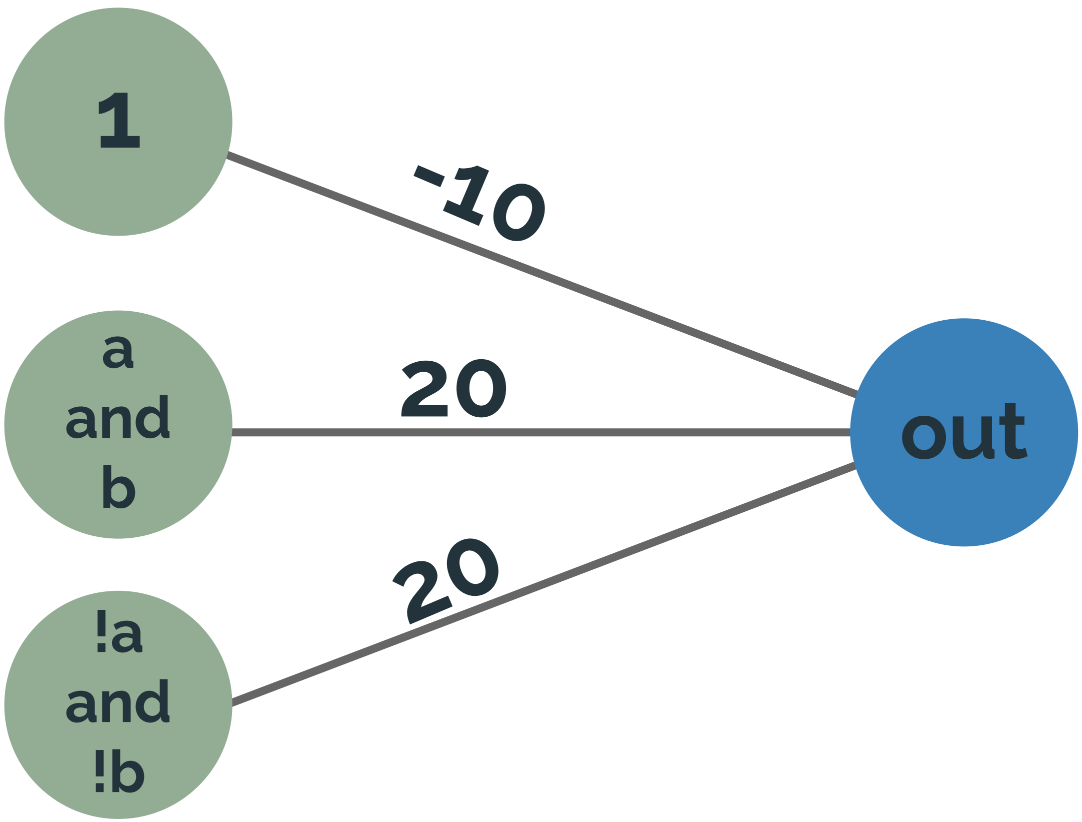


<u>**not a and not b**</u>



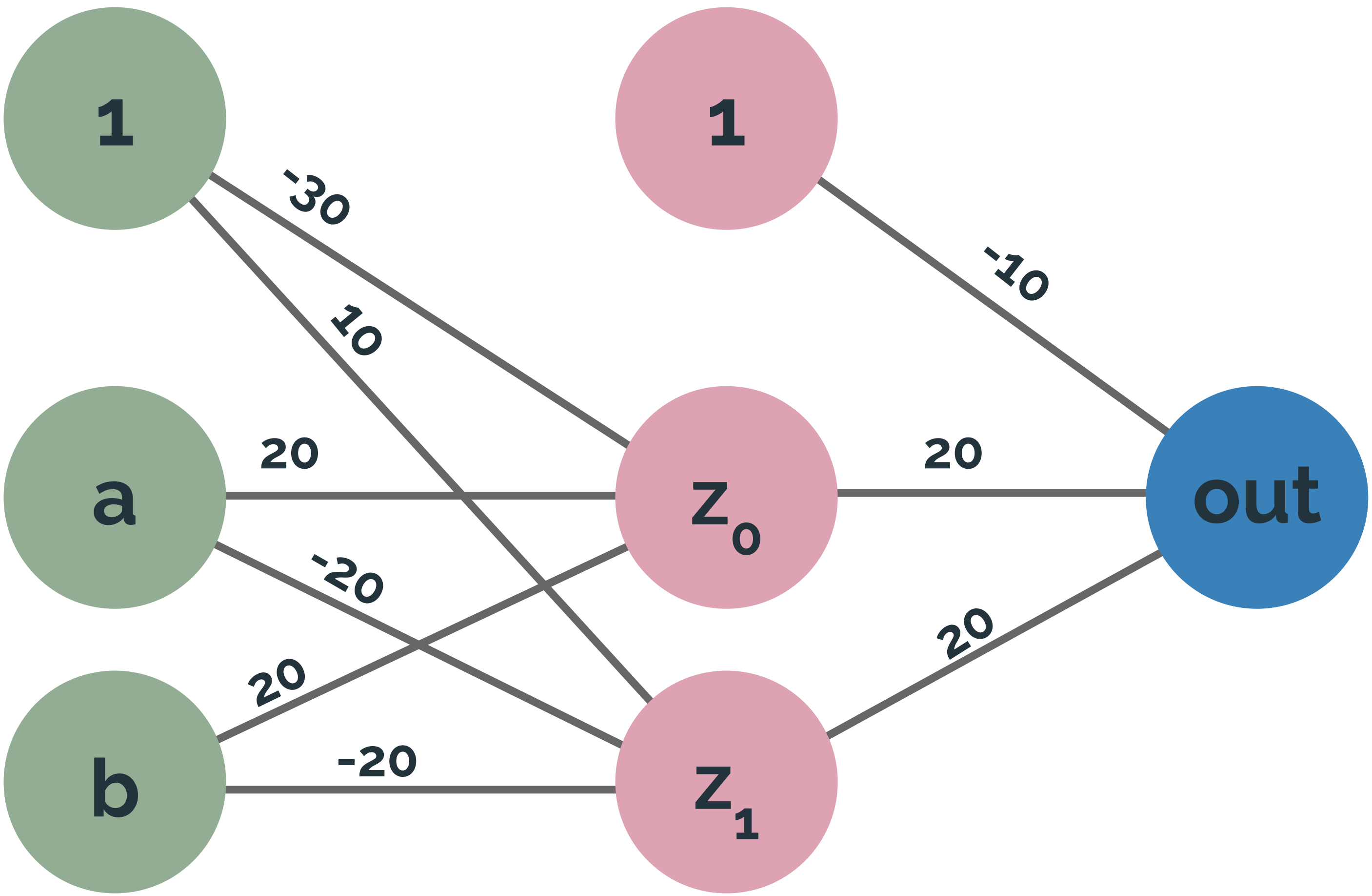
Second Layer

xnor is logically equivalent to: (a and b) **or** (not a and not b)



Putting it All Together

xnor is logically equivalent to: **(a and b) or (not a and not b)**



Neural Networks with AI

How do we find the correct weights to fit our data?

Steps for Neural Networks

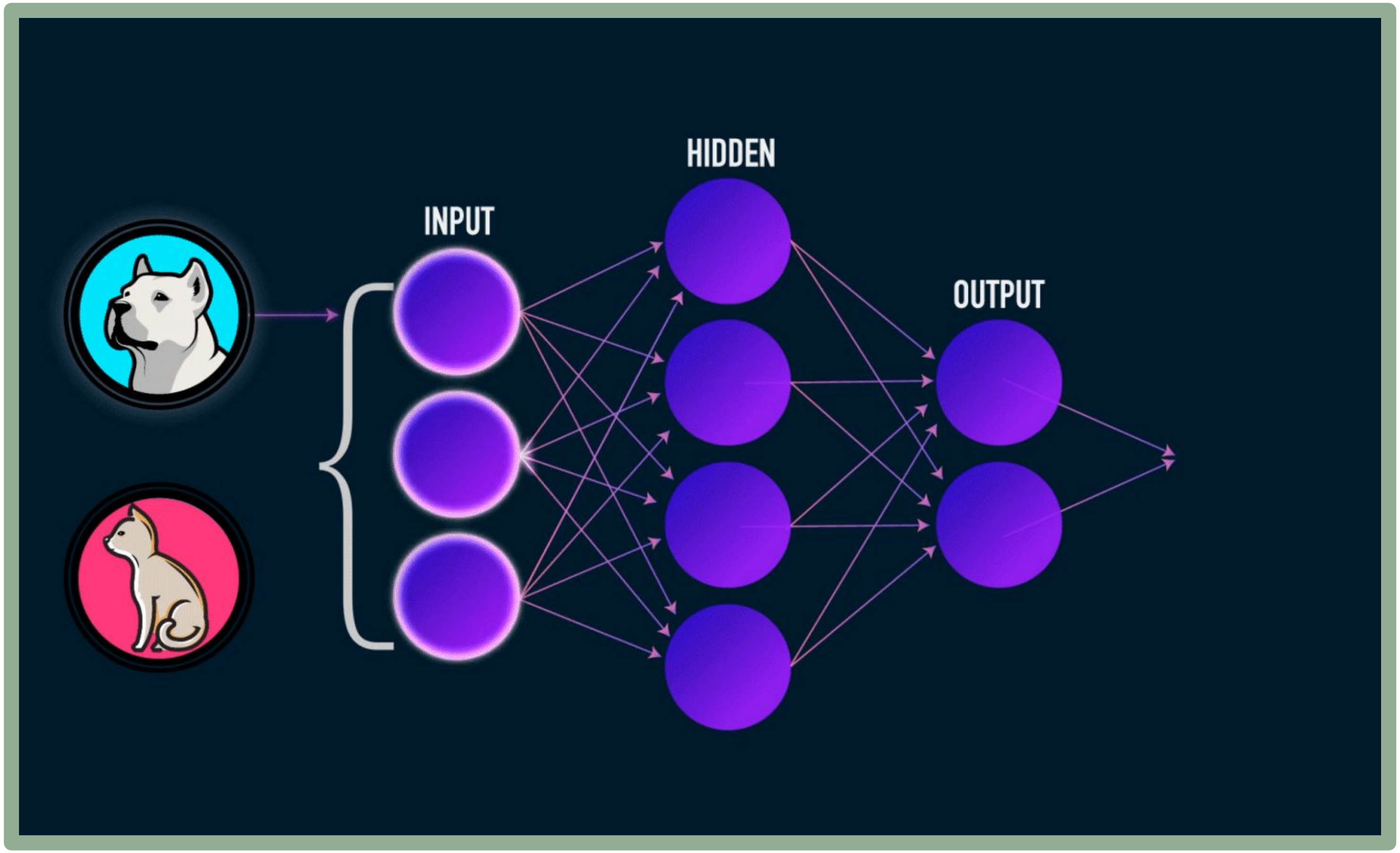
Every neural network has two main functions:

1. Forward Propagation
   - Predicting output values from our given neural network weights

2. Backward Propagation
   - Taking derivatives and adjusting these weights for multiple iterations

Forward Propagation

We've been doing this already with our examples!



Forward Propagation

More formally, for all layers in our neural network:

$$layer_i = sigmoid(layer_{i-1} \cdot w_{i-1})$$

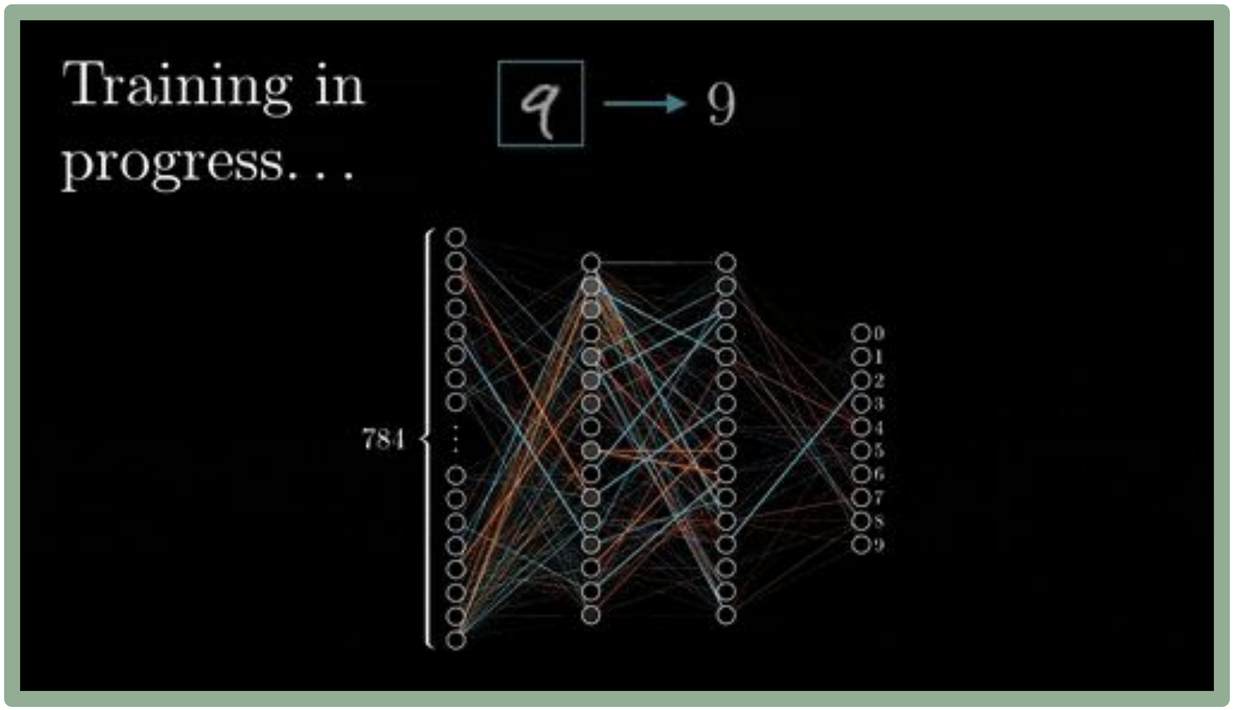
Where: $layer_i$ is the i^{th} layer in our NN,

w_i is the i^{th} set of weights in our NN

Back Propagation

Back propagation is just like the "adjusting thetas" step in gradient descent, but a little more complicated

However, we need to compute partial derivatives for each layer instead of doing it all at once



Back Propagation Formulas

Recall:

$$C(\hat{y}) = \frac{1}{2}(\hat{y} - y)^2$$

$$\hat{y} = sigmoid(X \cdot \Theta)$$

Hence,

$$cost = C(sigmoid(X_n, sigmoid(X_{n-1} \cdot \Theta_{n-1}))...)$$

Back Propagation Calculation

$$cost = C(sigmoid(X_n, sigmoid(X_{n-1} \cdot \Theta_{n-1})))$$

To adjust our network, we need to find the derivative for each theta:

$$\frac{d}{dCost} = \frac{dC}{d\hat{y}} \cdot \frac{d\hat{y}}{d(X_n \cdot \Theta_n)} \cdot \frac{d(X_n \cdot \Theta_n)}{d(sigmoid(X_{n-1} \cdot \Theta_{n-1}))} \cdot \ldots \cdot \frac{d(X_1 \cdot \Theta_1)}{d\Theta_1}$$

This is an application of the chain rule! In the notebook there's a more detailed explanation on how everything works

Tasks to Complete

1) Work on the notebook (**nn_code.zip**) in the google drive folder https://drive.google.com/file/d/1MOJ97mTWGGeFkqwhdOsV-YXs9BaQtL8f/view?usp=sharing

Try to work on it collaboratively! You might meet some people you could do a project with in the future

As always, let us know if you need any help!

SIG
NLL