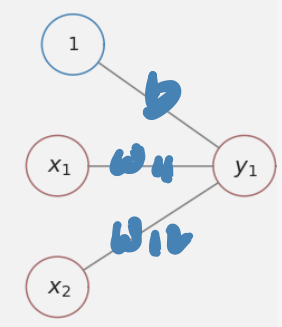
Neural Networks Part II

Feed-Forward Neural Networks

Previously on CSCI 4622

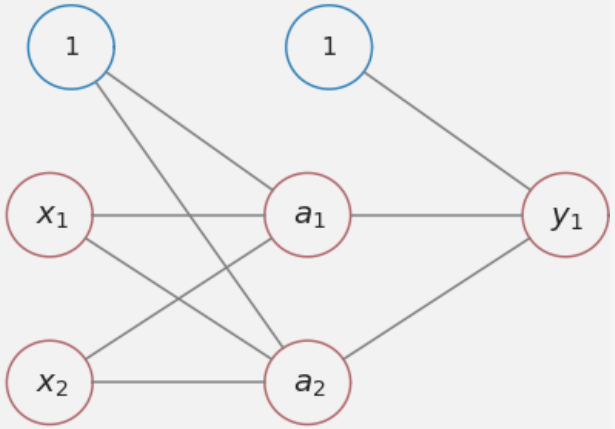
The Perceptron as a NN: Can learn linearly separable things like OR, AND, NAND, etc



$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^T\mathbf{x} + b \leq 0 \\ 1 & \text{if } \mathbf{w}^T\mathbf{x} + b > 0 \end{cases}$$

x_1	0	1	0	1
x_2	0	0	1	1
x_1 OR x_2	0	1	1	1

The Multilevel Perceptron: Can learn non-linearly separable things like XOR



Layer 1 to 2:

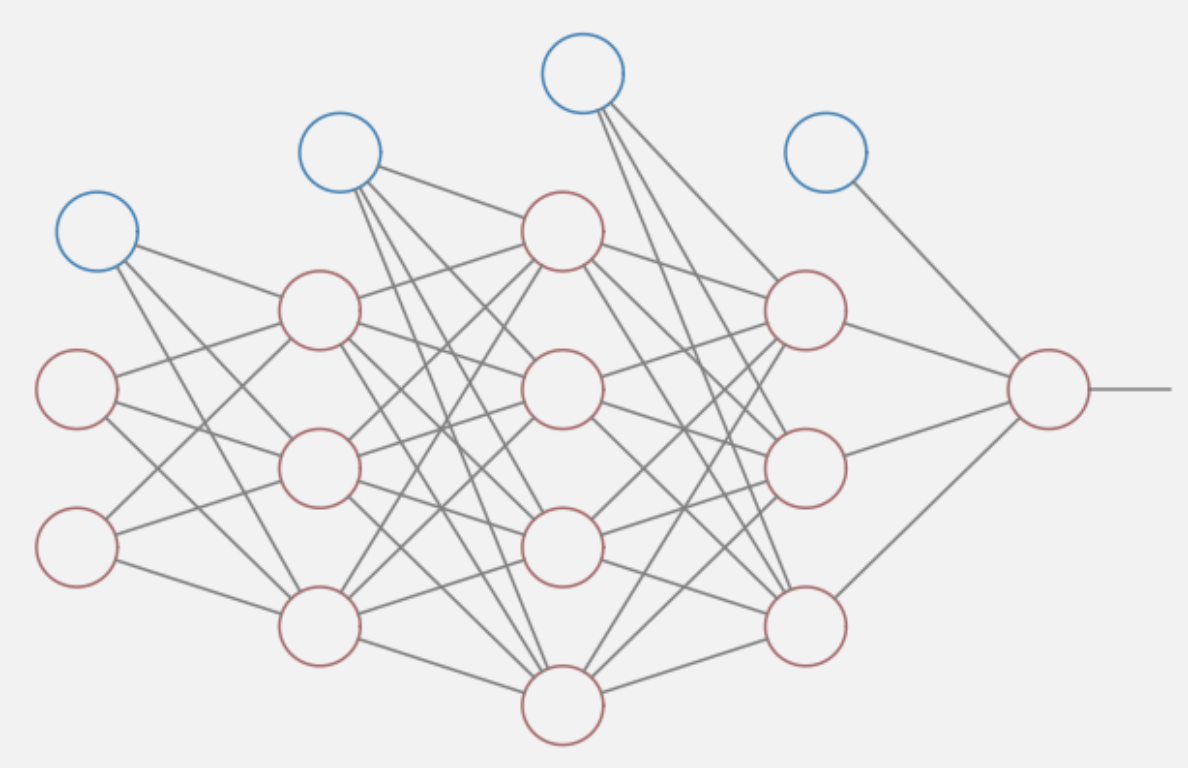
$$\begin{aligned} \mathbf{z}^2 &= W^1\mathbf{x}^1 + \mathbf{b}^1 \\ \mathbf{a}^2 &= \mathrm{I}(\mathbf{z}^2 > 0) = \mathrm{I}(W^1\mathbf{x}^1 + \mathbf{b}^1 > 0) \end{aligned}$$

Layer 2 to 3:

$$\begin{aligned} \mathbf{z}^3 &= W^2\mathbf{a}^2 + \mathbf{b}^2 \\ \hat{y} &= \mathrm{I}(\mathbf{z}^3 > 0) = \mathrm{I}(W^2\mathbf{a}^2 + \mathbf{b}^2 > 0) \end{aligned}$$

The Multilayer Perceptron

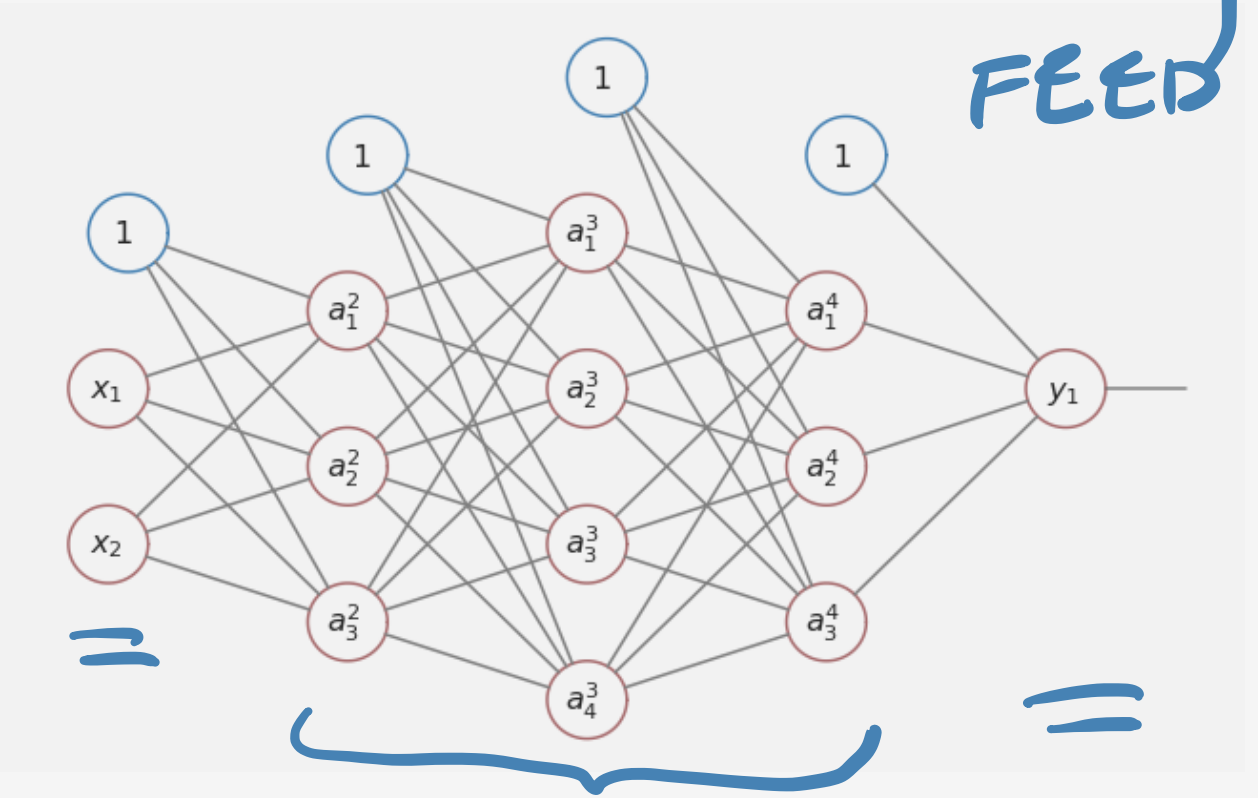
String together lots of perceptron units



The Multilayer Perceptron

String together lots of perceptron units

FULLY-CONNECT
FEED-FORWARD
MLP

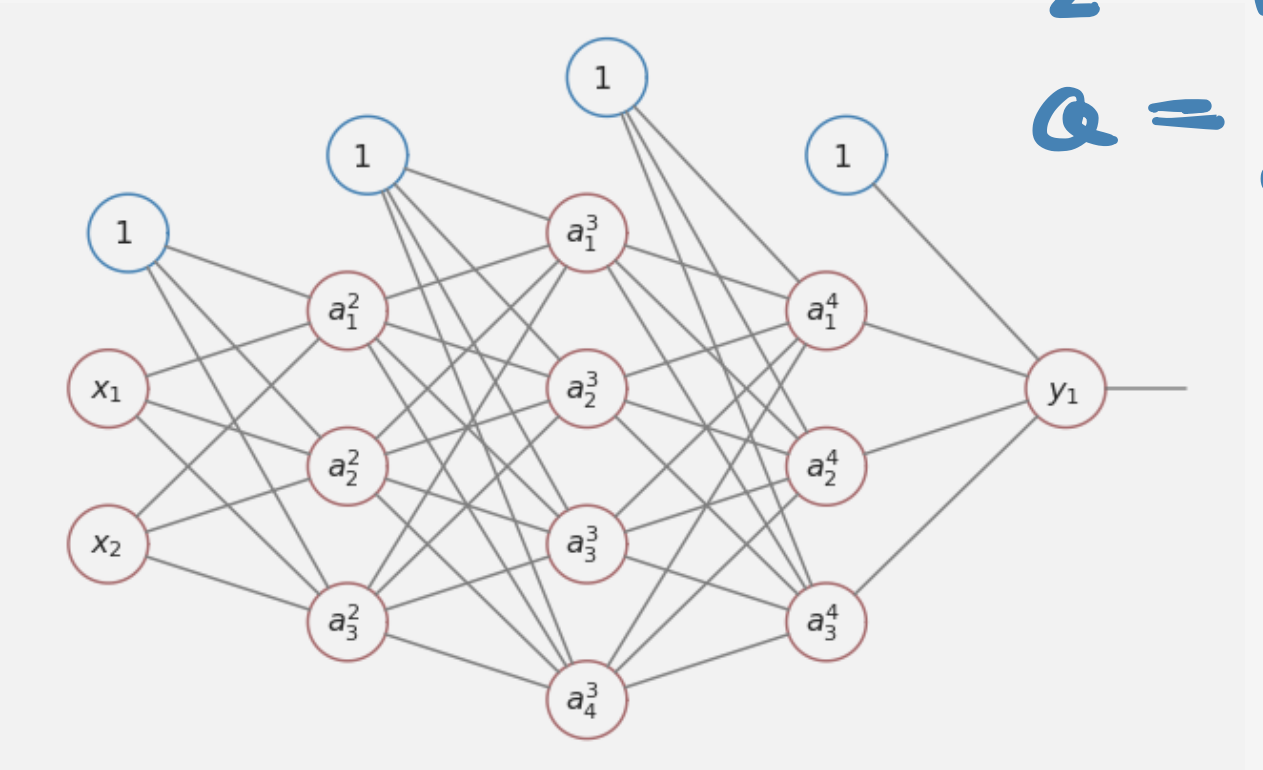


Input Layer Hidden Layers Output Layer

The Multilayer Perceptron

String together lots of perceptron units

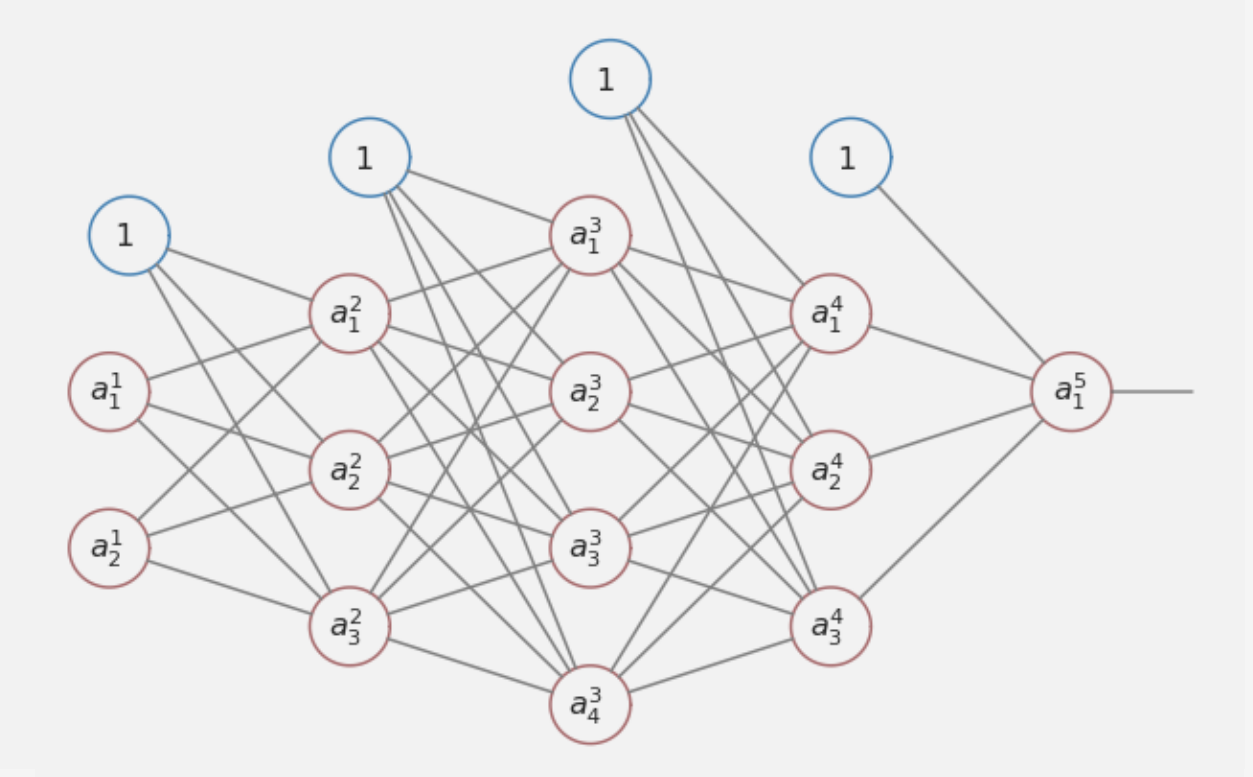
$$z = Wa + b$$
$$a = g(z)$$



Inputs, activities, activations, and outputs all live on nodes (neurons)

The Multilayer Perceptron

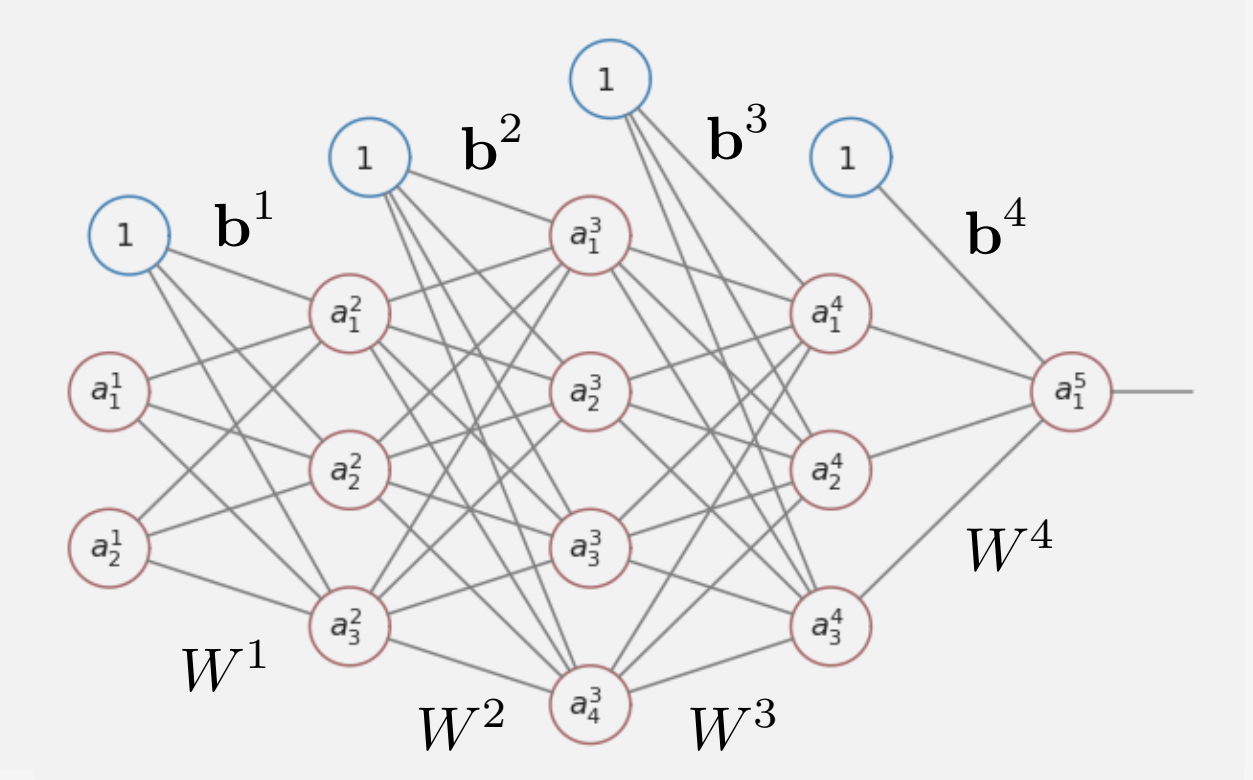
String together lots of perceptron units



For simplicity: Think of inputs and outputs as activations too

The Multilayer Perceptron

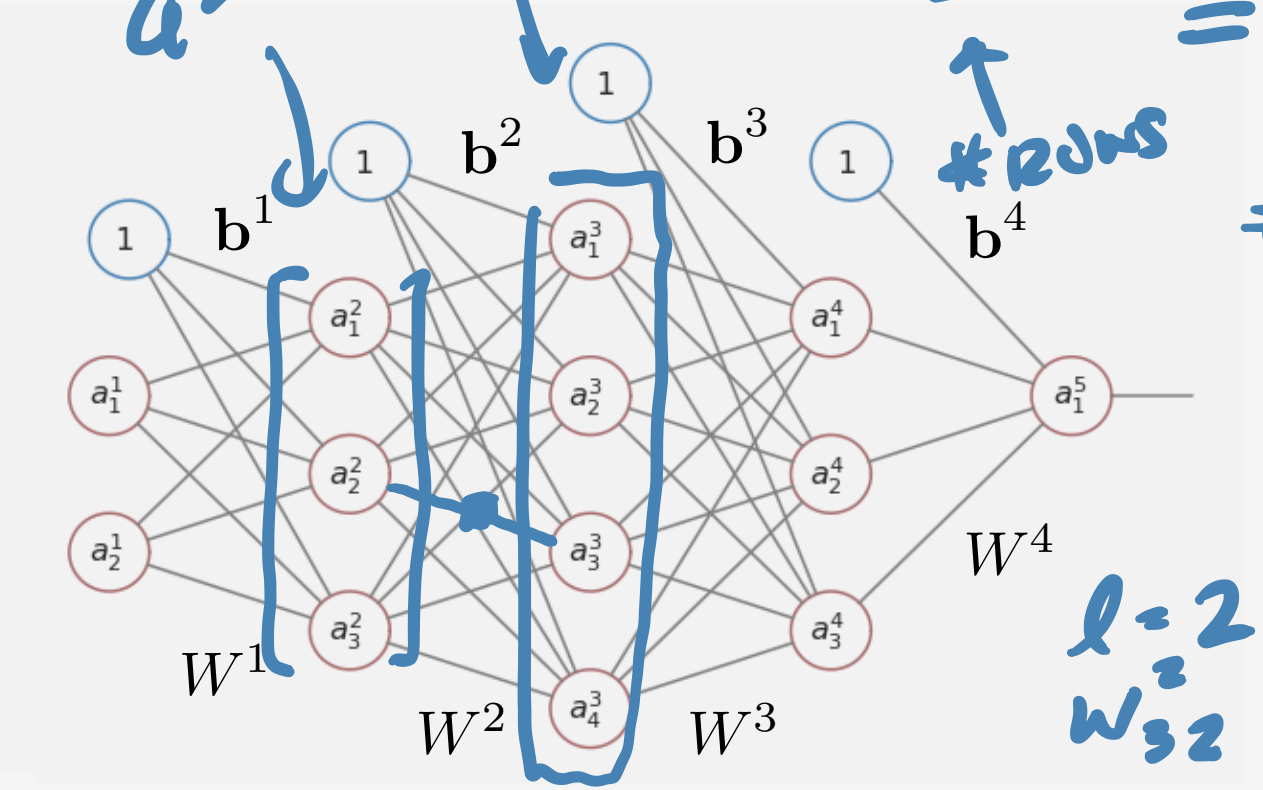
String together lots of perceptron units



Each transition between layers has associated set of weights an biases

The Multilayer Perceptron

String together lots of perceptron units



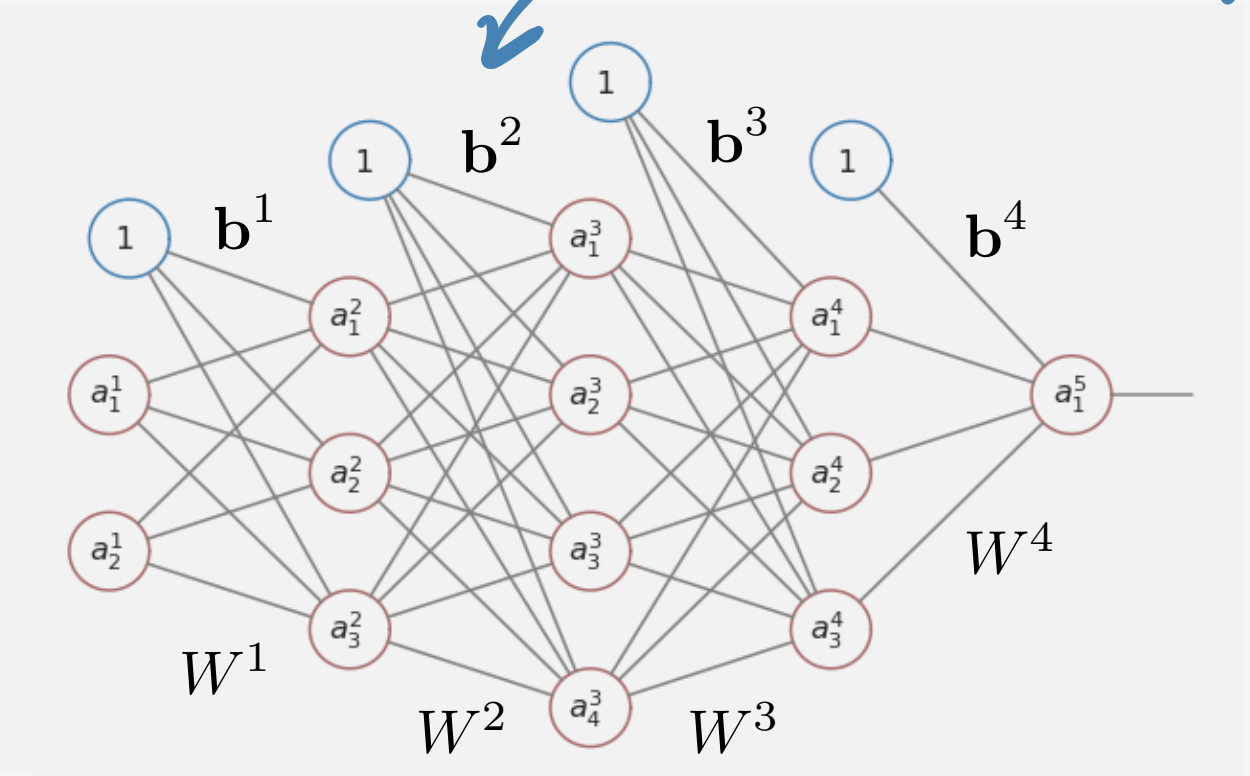
Store the weights connecting layers ℓ and $\ell+1$ in the matrix W^ℓ

w_{ij}^ℓ is the weight from node j in layer ℓ to node i in layer $\ell+1$

The Multilayer Perceptron

String together lots of perceptron units

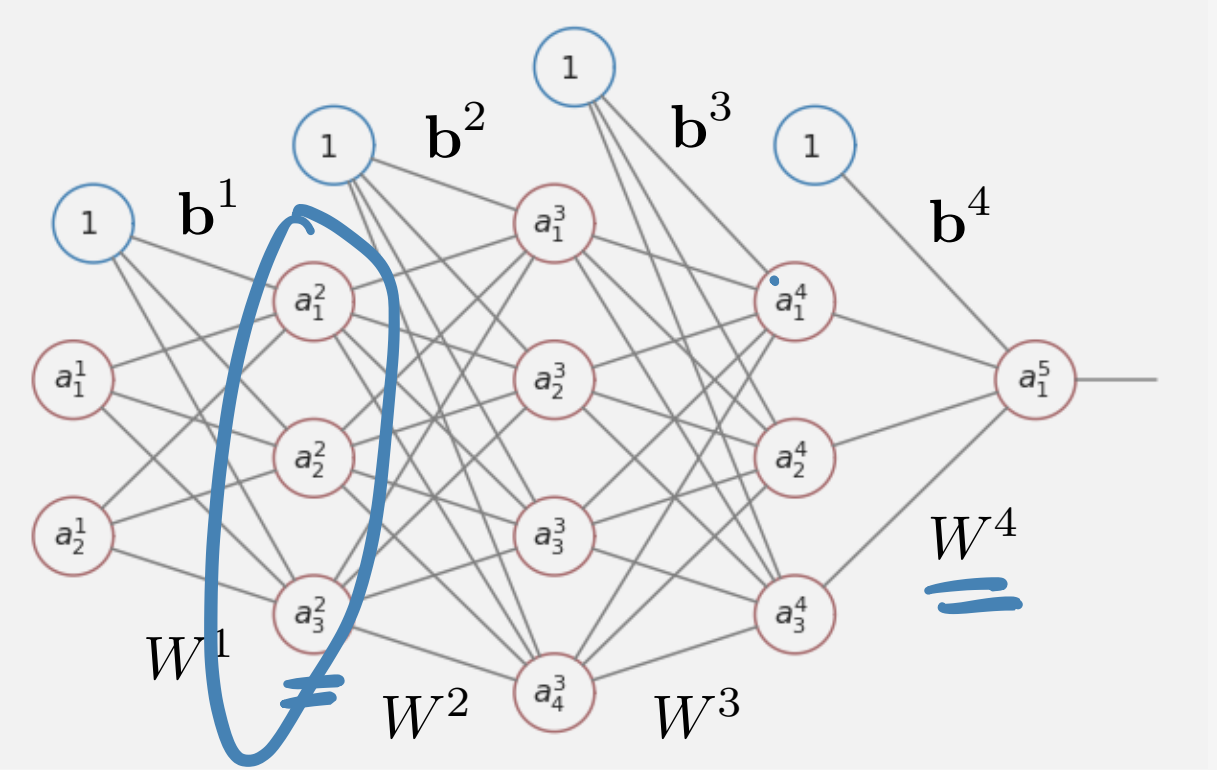
biases in layer 3



Store the biases for layer $\ell + 1$ in the vector

$\mathbf{b}^\ell$ is the bias feeding into node i in layer $\ell + 1$

The Multilayer Perceptron



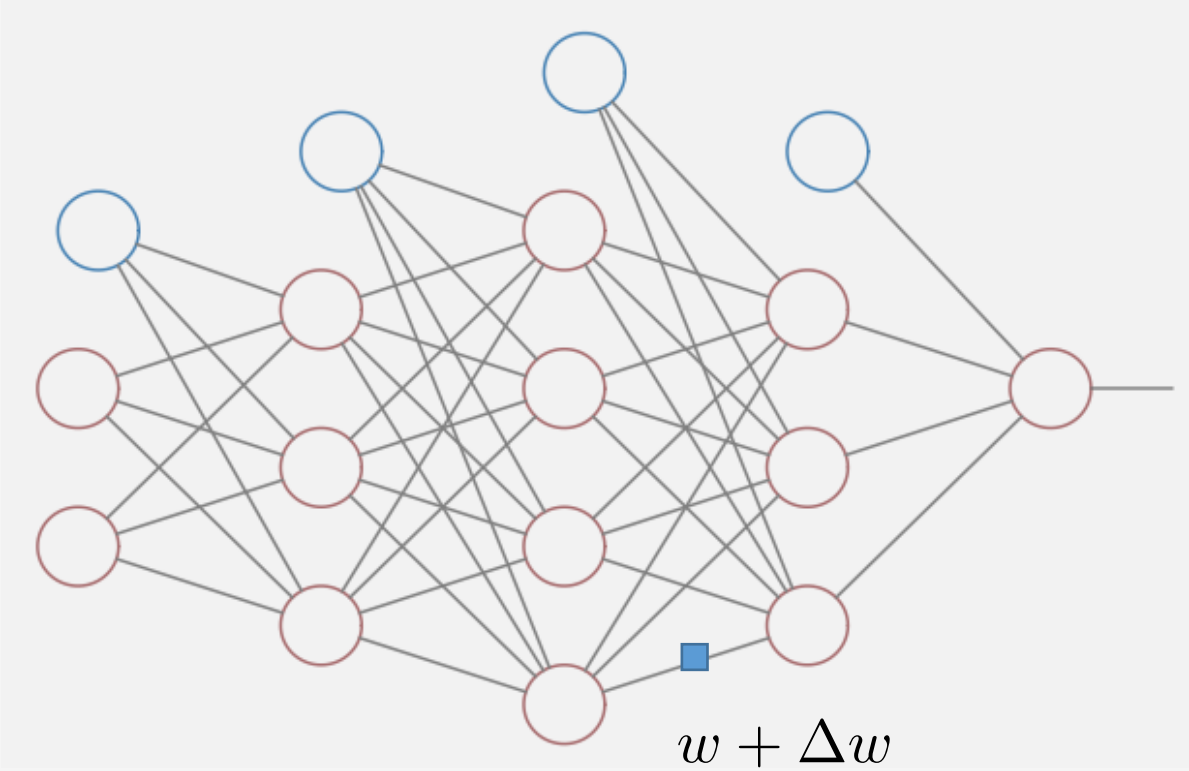
Question: What are the dimensions of each of the following?

W^2: 4×3, $\mathbf{b}^3$: 3, $\mathbf{z}^2$: 3, W^4: 1×3

Sigmoid Neurons

Next week we'll see how we can learn the weights and the biases in a NN using SGD

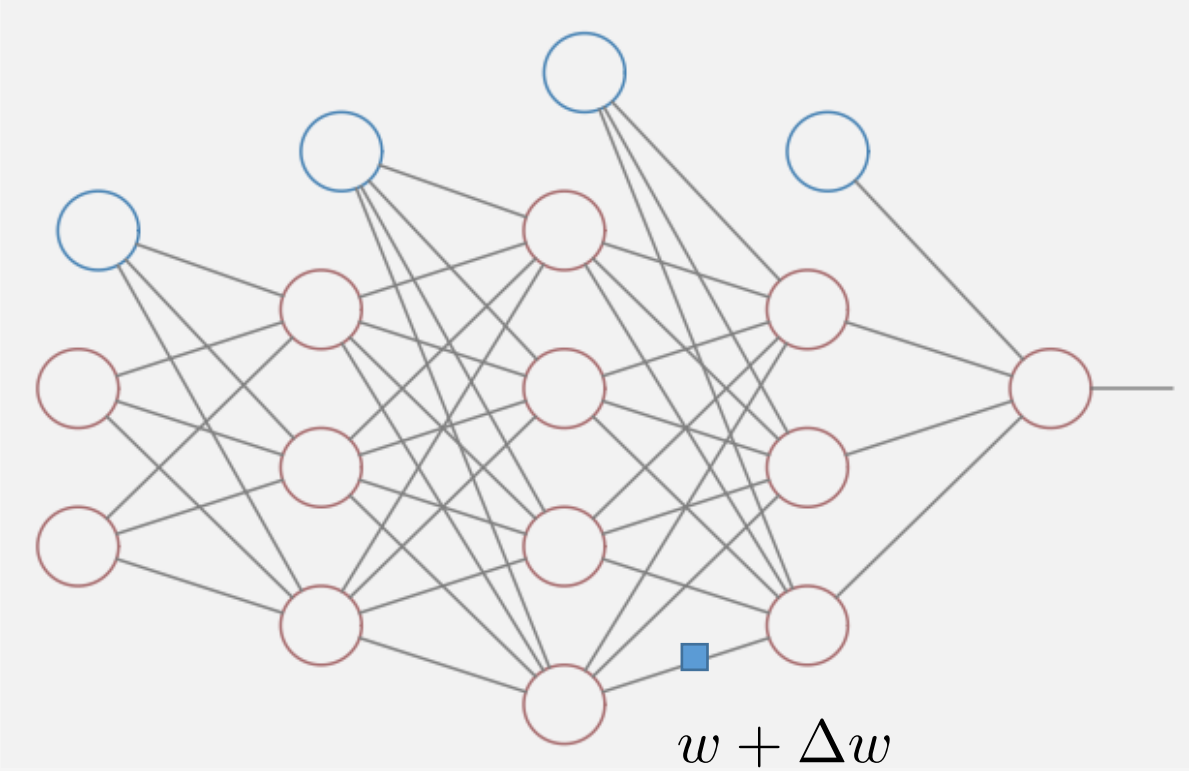
This poses a problem with standard perceptron neurons with the Indicator activation function



Sigmoid Neurons

Next week we'll see how we can learn the weights and the biases in a NN using SGD

This poses a problem with standard perceptron neurons with the Indicator activation function



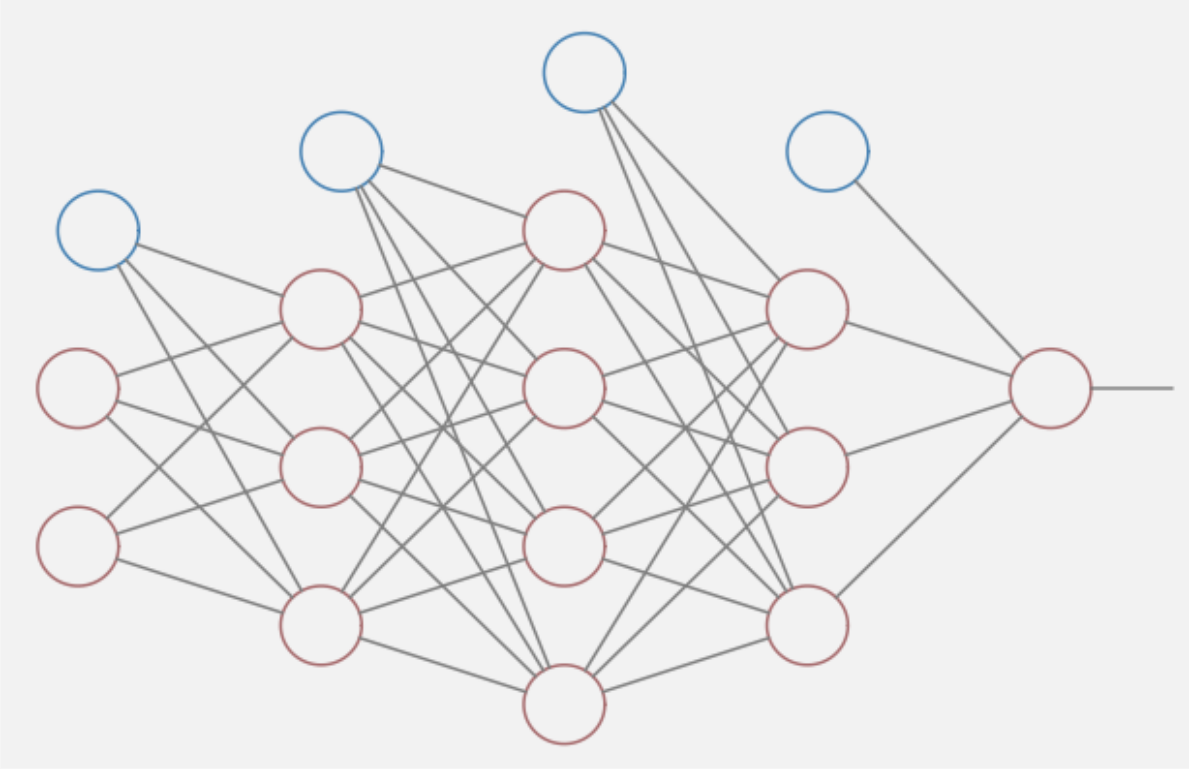
Q: What activation could we use instead?

$$\text{output} + \Delta\text{output}$$

Sigmoid Neurons

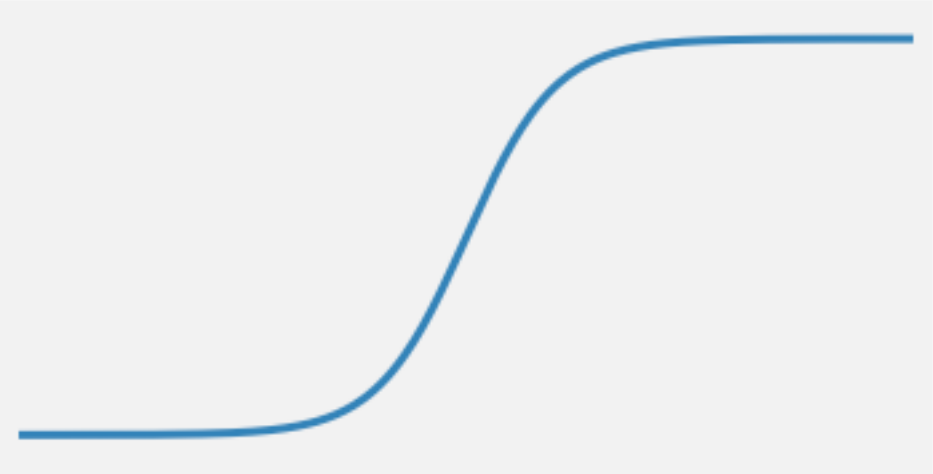
Next week we'll see how we can learn the weights and the biases in a NN using SGD

This poses a problem with standard perceptron neurons with the Indicator activation function



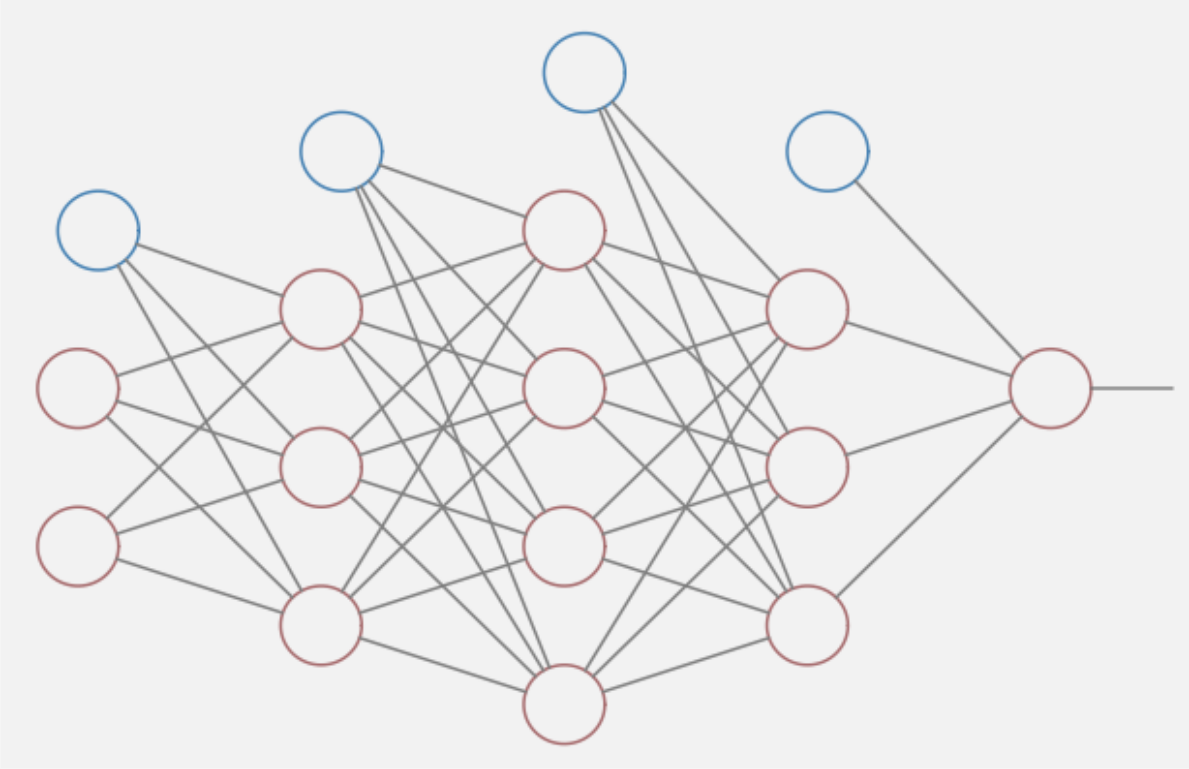
Q: What activation could we use instead?

A: How about a sigmoid activation?



Sigmoid Neurons

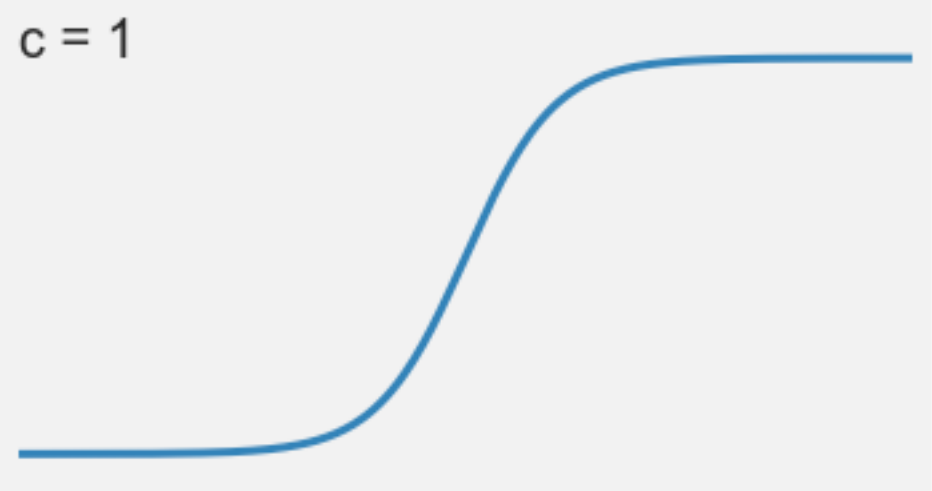
Is a sigmoid neuron all that different from a perceptron?



For a fixed input. $\mathbf{x}$,

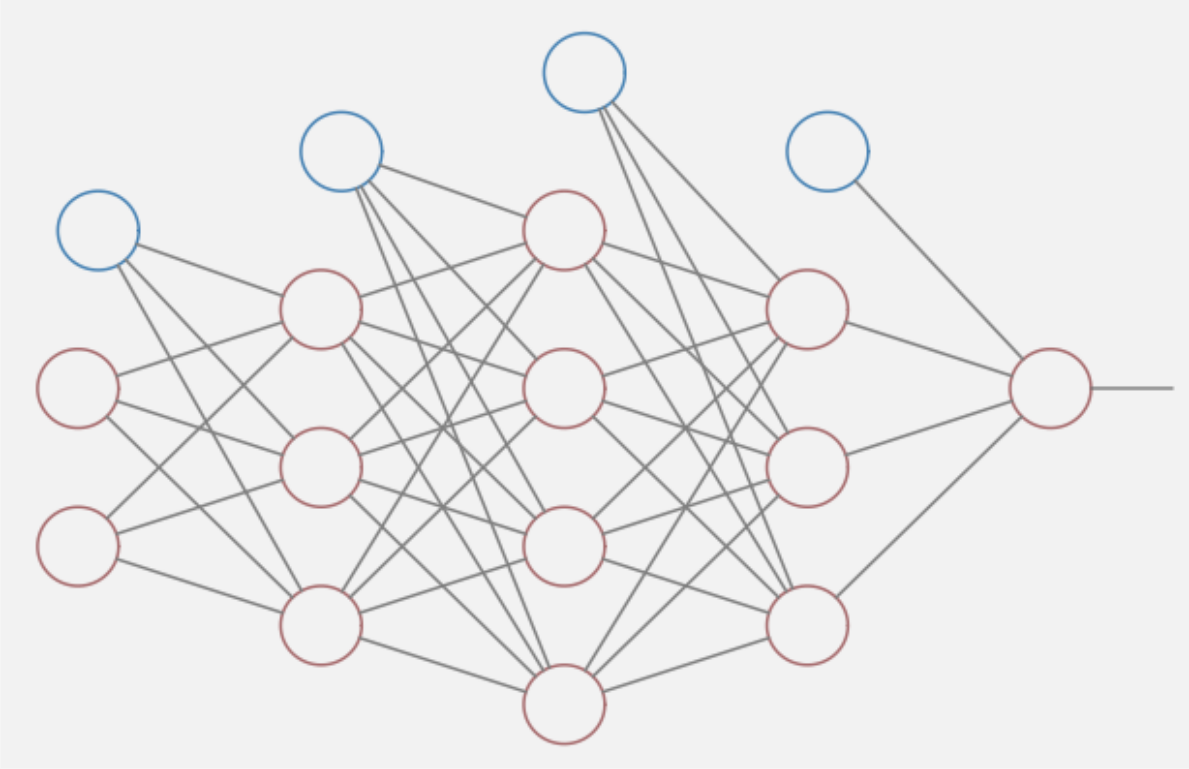
multiply weights and bias by a factor c

$$\text{sigm}[c(\mathbf{w}^T x + b)]$$



Sigmoid Neurons

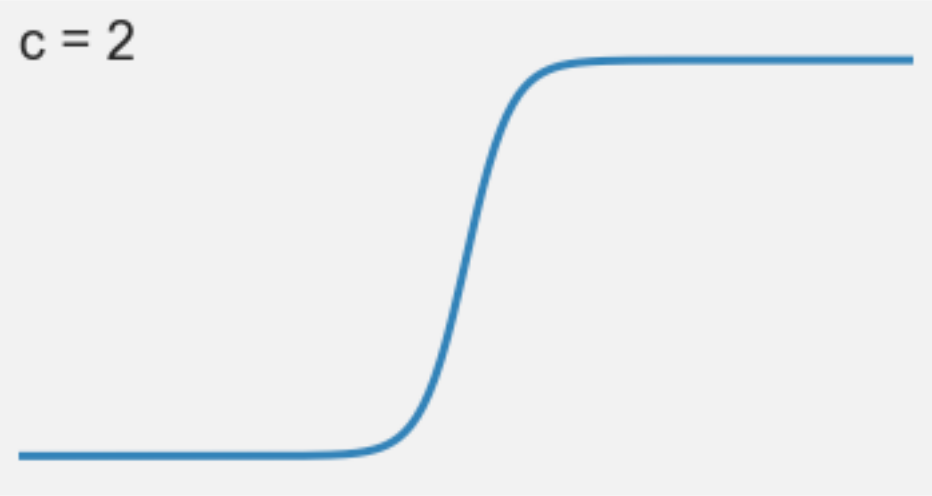
Is a sigmoid neuron all that different from a perceptron?



For a fixed input. $\mathbf{x}$,

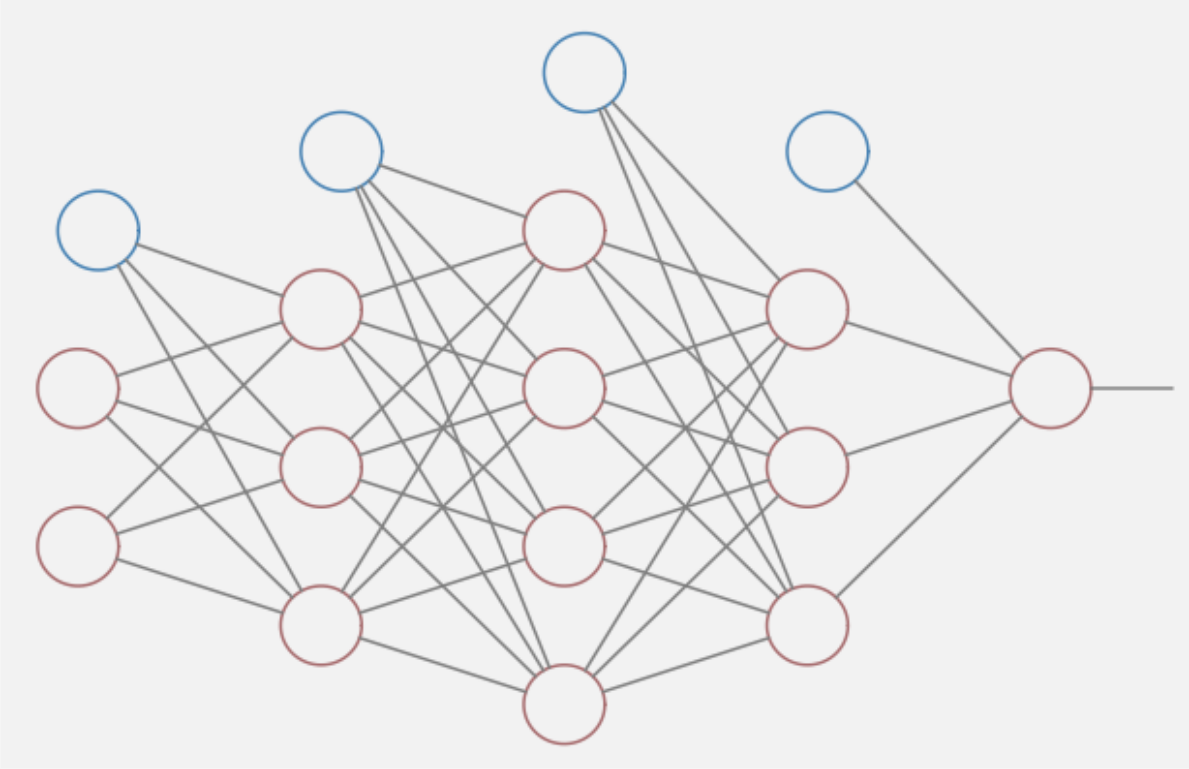
multiply weights and bias by a factor c

$$\text{sigm}[c(\mathbf{w}^T x + b)]$$



Sigmoid Neurons

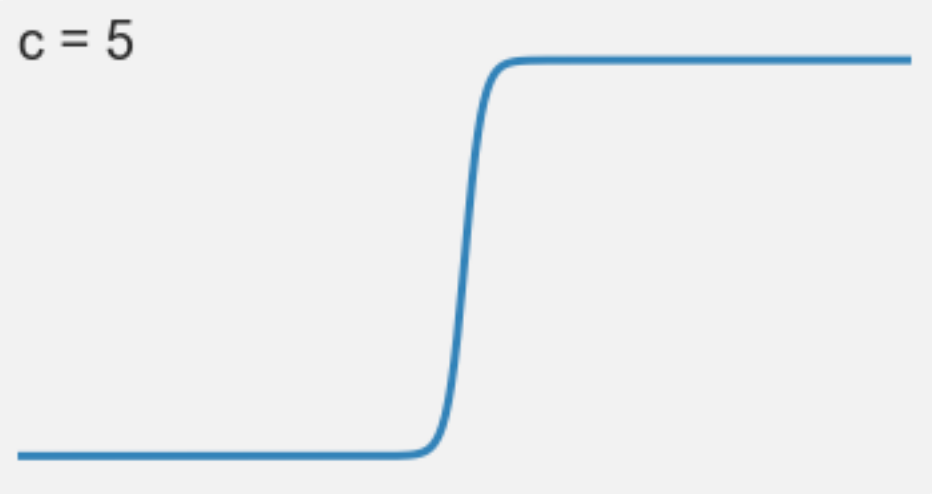
Is a sigmoid neuron all that different from a perceptron?



For a fixed input. $\mathbf{x}$,

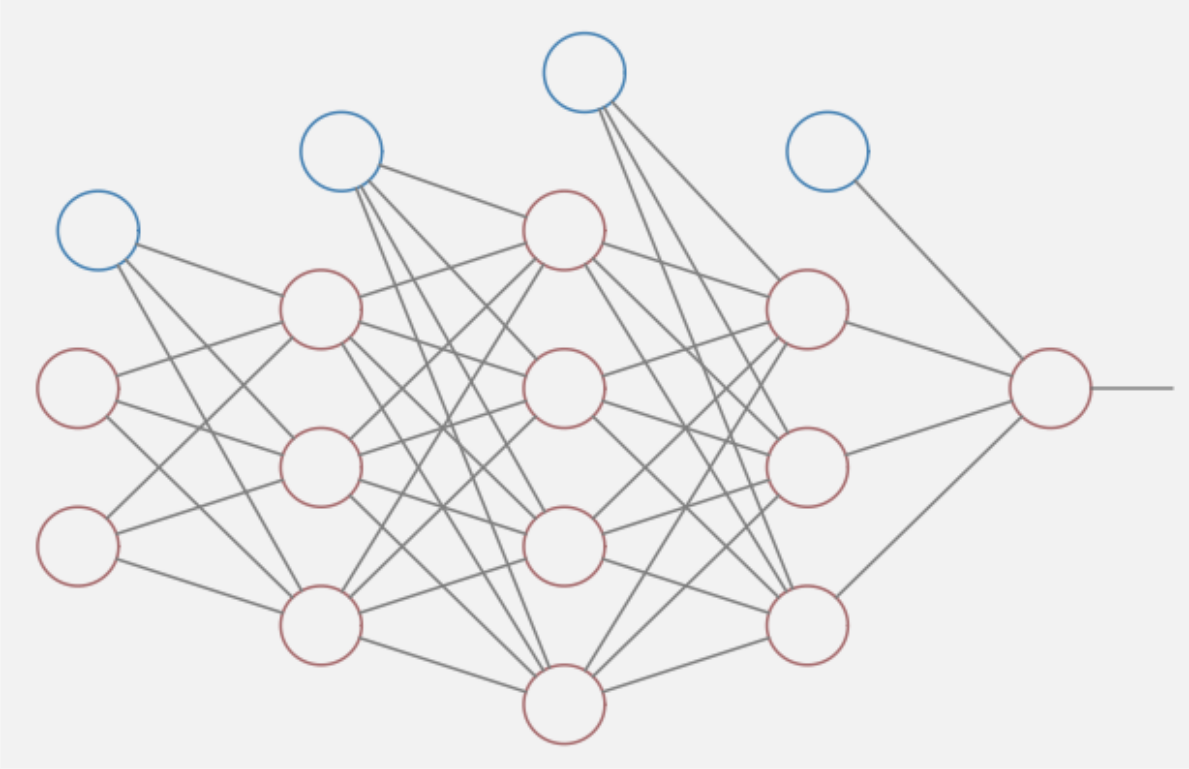
multiply weights and bias by a factor c

$$\text{sigm}[c(\mathbf{w}^T x + b)]$$



Sigmoid Neurons

Is a sigmoid neuron all that different from a perceptron?



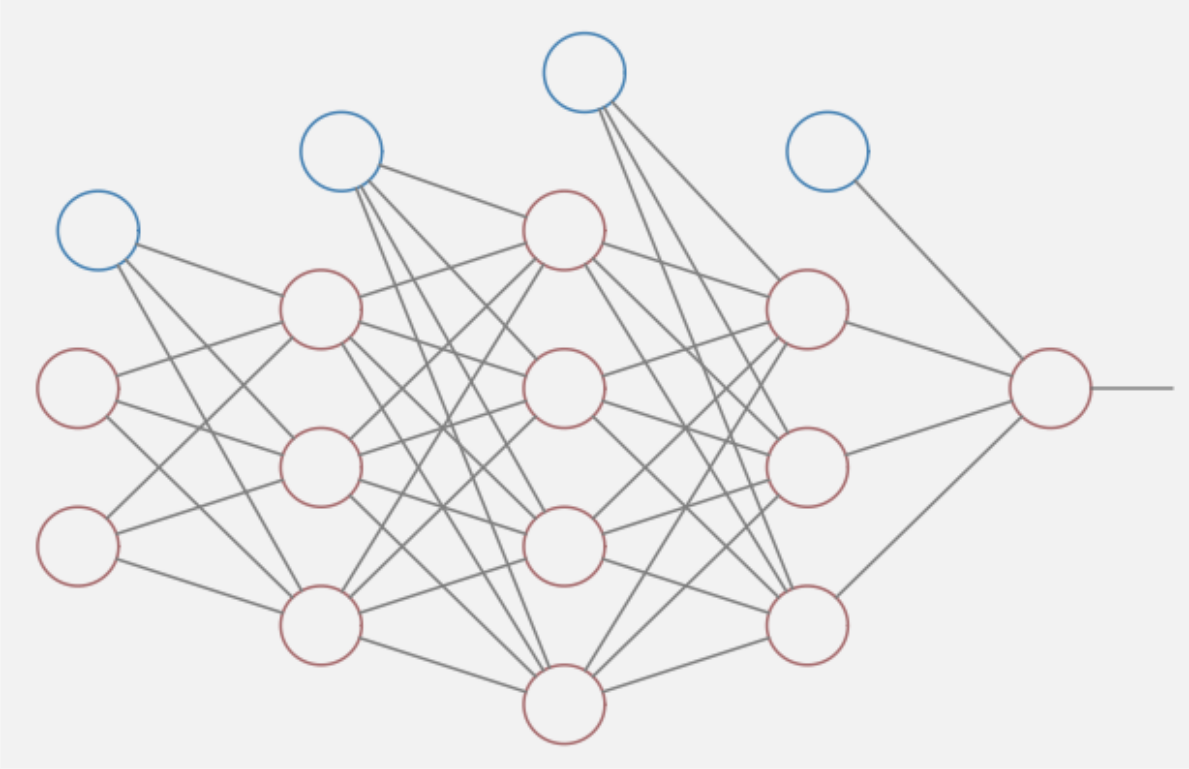
For a fixed input. $\mathbf{x}$,

multiply weights and bias by a factor c

$$\text{sigm}[c(\mathbf{w}^T x + b)]$$

Sigmoid Neurons

Is a sigmoid neuron all that different from a perceptron?



For a fixed input. $\mathbf{x}$,

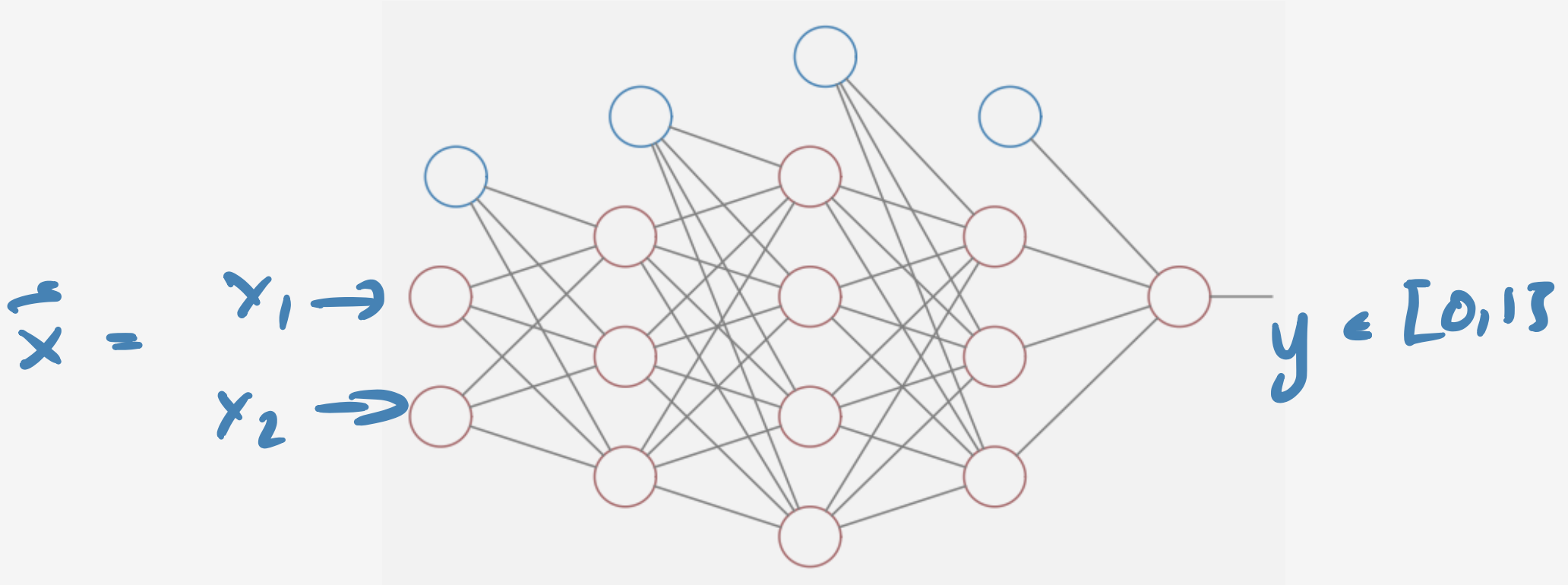
multiply weights and bias by a factor c

$$\text{sigm}[c(\mathbf{w}^T x + b)]$$

Predictions: Forward Propagation

Suppose we've learned the weights in a NN with sigmoid neurons.

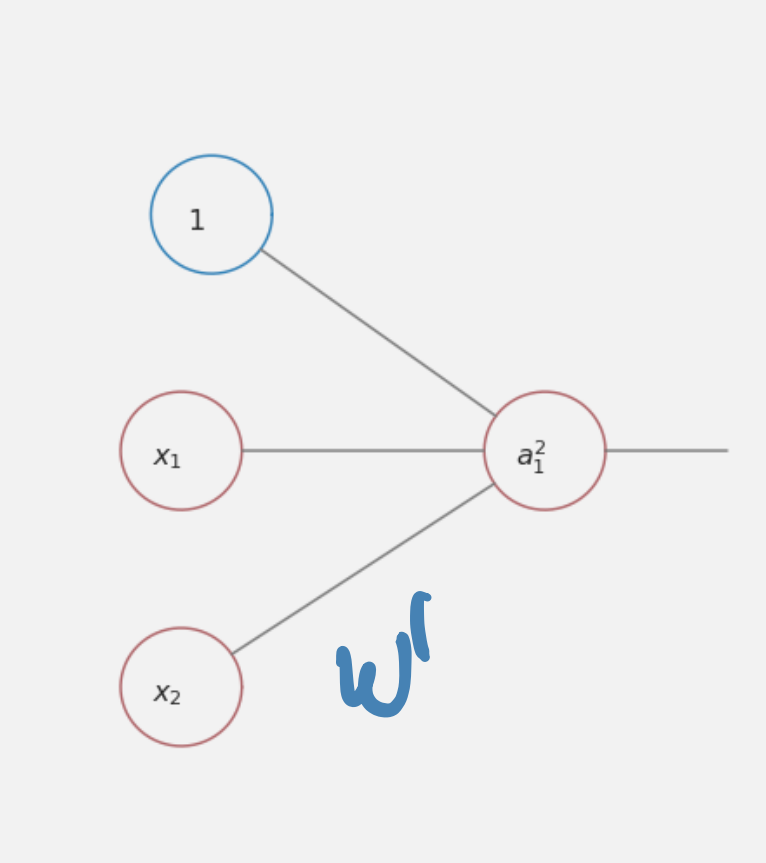
Given an input $\mathbf{x}$, how do we make a prediction?



Predictions: Forward Propagation

Suppose we've learned the weights in a NN with sigmoid neurons.

Given an input $\mathbf{x}$, how do we make a prediction? First, think about a simple network:



$$W^1 = [w^1_{11} \; w^1_{12}] \qquad b^1 = [b^1_1]$$

$$z^2 = W^1 x + b$$

$$a^2 = g(W^1 x + b)$$

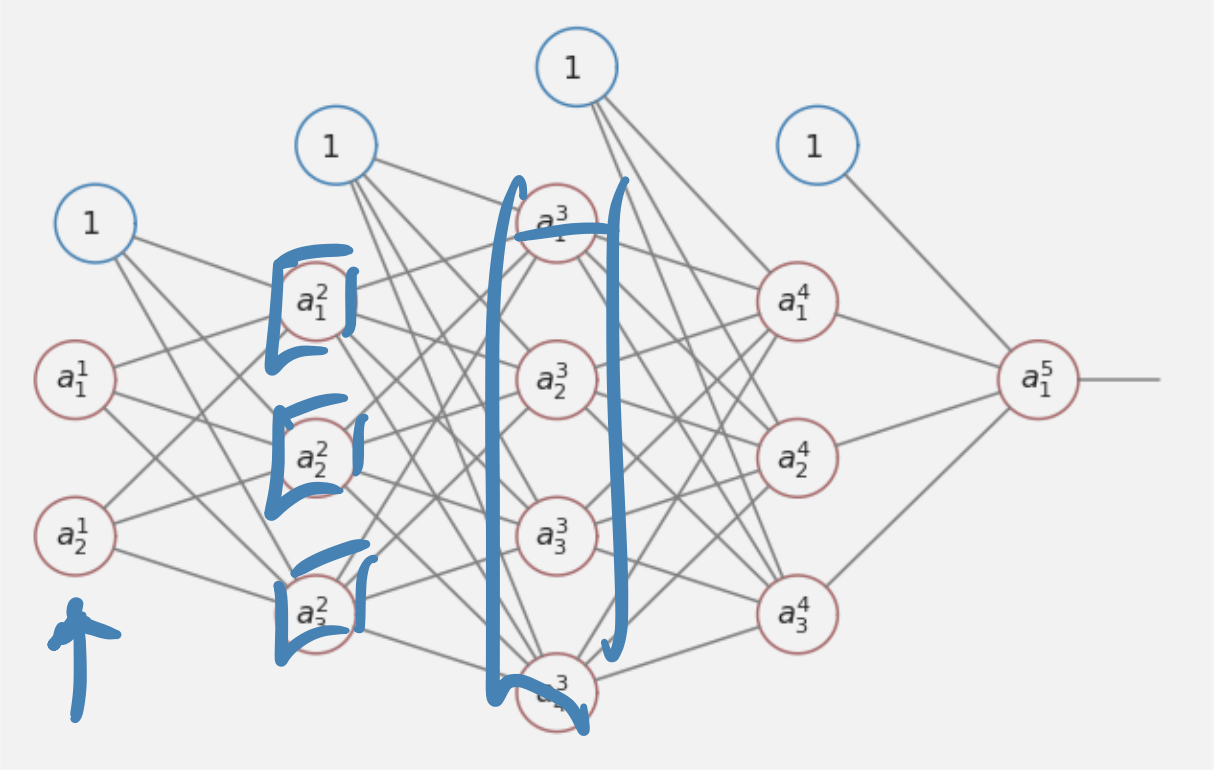
$$\downarrow$$

$$y$$

Predictions: Forward Propagation

Suppose we've learned the weights in a NN with sigmoid neurons.

Given an input $\mathbf{x}$, how do we make a prediction?



$z^2 = W^1 a^1 + b^1$

$a^2 = g(z^2)$

$z^3 = W^2 a^2 + b^2$

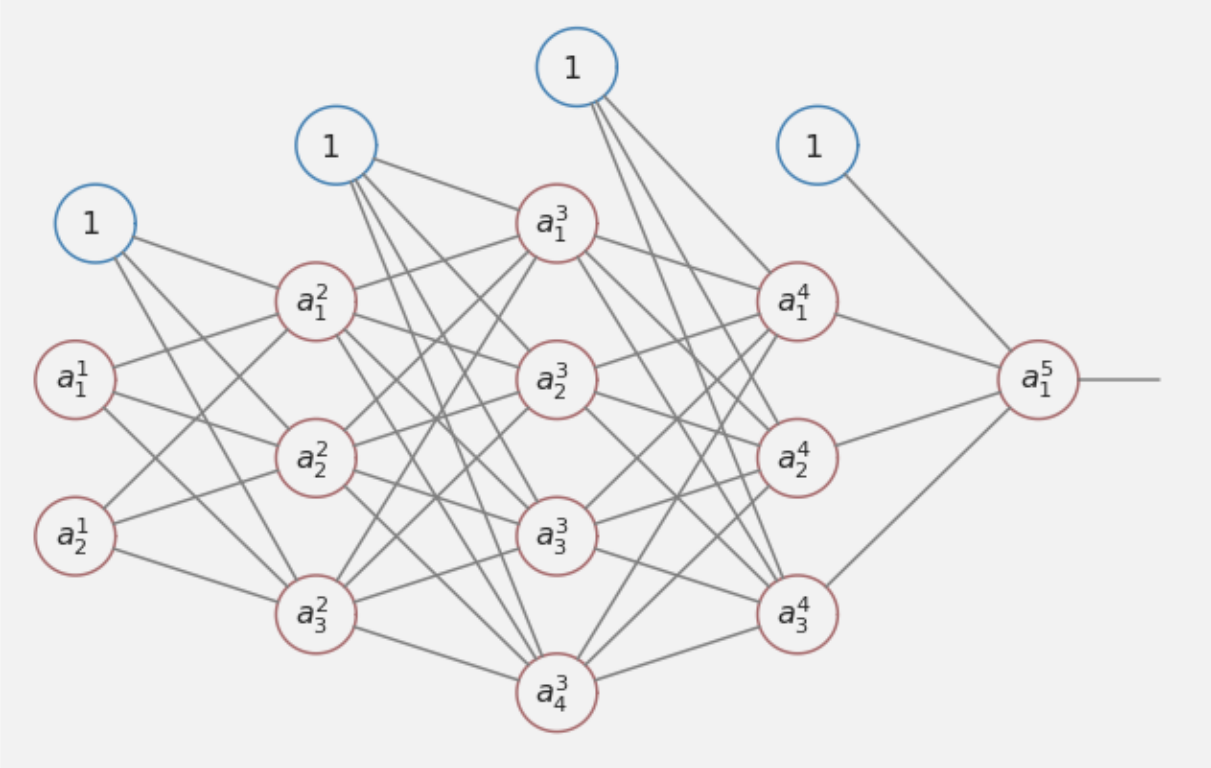
$a^3 = g(z^3)$

Re: $\mathbf{a}^{\ell+1} = g(\mathbf{z}^{\ell+1})$ where $\mathbf{z}^{\ell+1} = W^\ell \mathbf{a}^\ell + \mathbf{b}^\ell$

Predictions: Forward Propagation

Suppose we've learned the weights in a NN with sigmoid neurons.

Given an input $\mathbf{x}$, how do we make a prediction?



Suppose the network has L layers

Intialize $\mathbf{a}^1 = \mathbf{x}$

for $\ell = 1, \ldots, L-1$

$$\mathbf{z}^{\ell+1} = W^\ell \mathbf{a}^\ell + \mathbf{b}^\ell$$

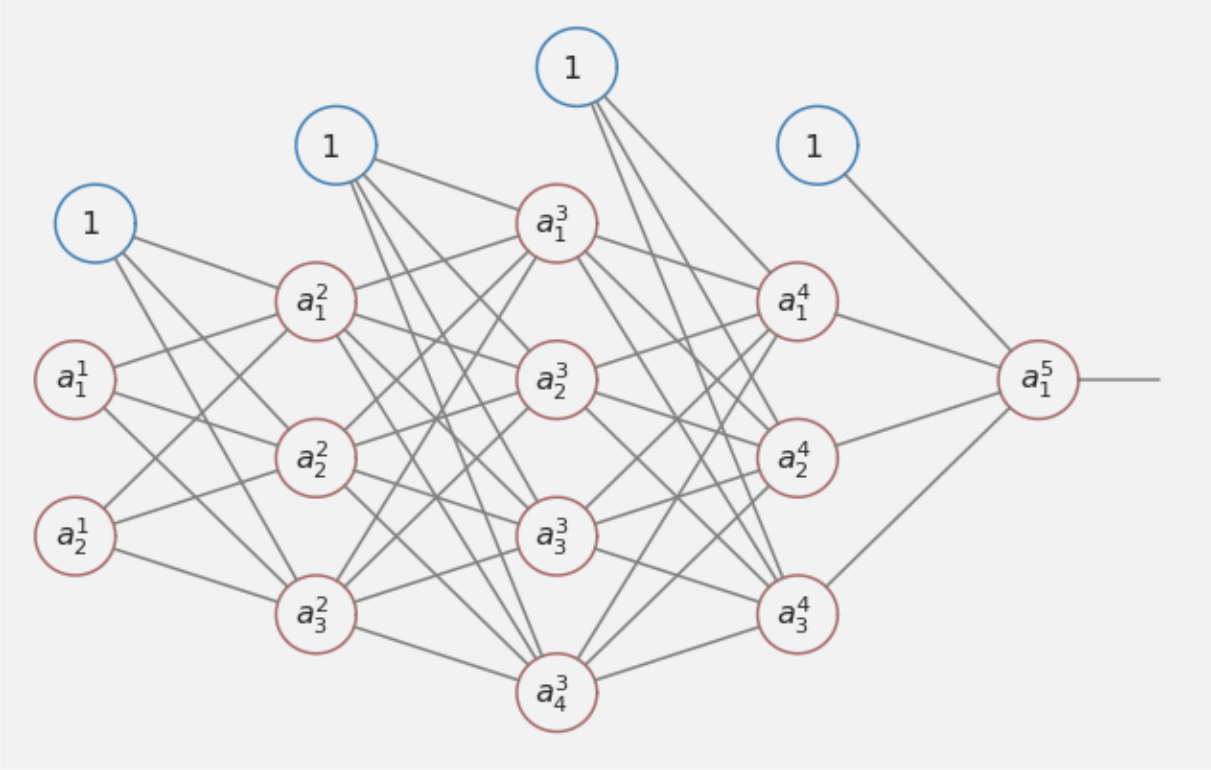
$$\mathbf{a}^{\ell+1} = g(\mathbf{z}^{\ell+1})$$

Re: $\mathbf{a}^{\ell+1} = g(\mathbf{z}^{\ell+1})$ where $\mathbf{z}^{\ell+1} = W^\ell \mathbf{a}^\ell + \mathbf{b}^\ell$

Predictions: Forward Propagation

Suppose we've learned the weights in a NN with sigmoid neurons.

Given an input $\mathbf{x}$, how do we make a prediction?



Re: $\mathbf{a}^{\ell+1} = g(\mathbf{z}^{\ell+1})$ where $\mathbf{z}^{\ell+1} = W^\ell \mathbf{a}^\ell + \mathbf{b}^\ell$

Suppose the network has L layers

Intialize $\mathbf{a}^1 = \mathbf{x}$

for $\ell = 1, \ldots, L-1$

$$\mathbf{z}^{\ell+1} = W^\ell \mathbf{a}^\ell + \mathbf{b}^\ell$$

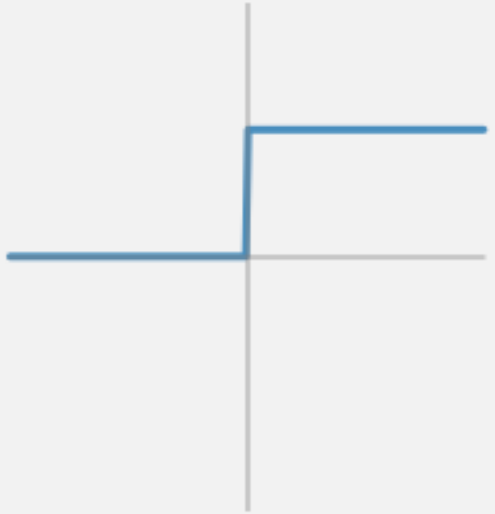
$$\mathbf{a}^{\ell+1} = g(\mathbf{z}^{\ell+1})$$

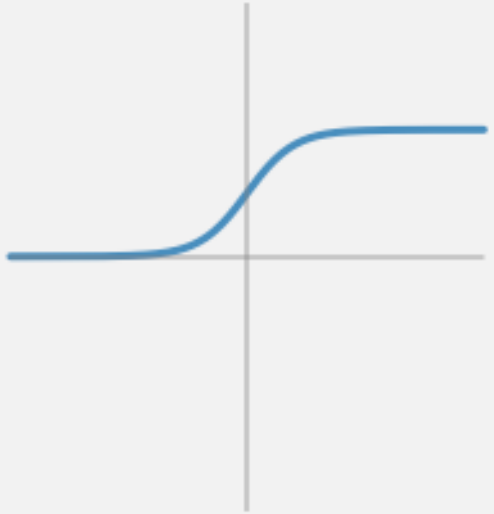
If classification and sigmoid neuron output

then predict based on $\mathbf{a}^L$

Other Popular Activation Functions

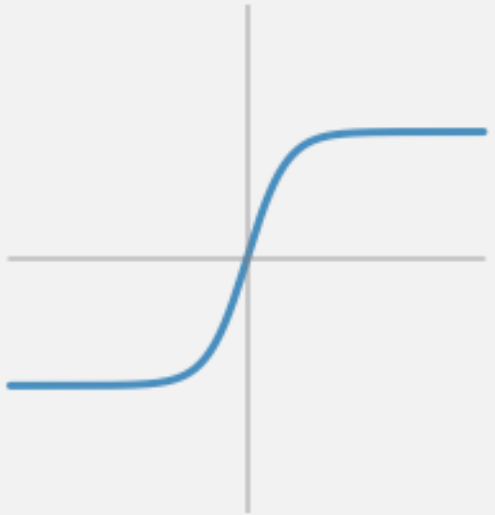
perceptron





sigmoid

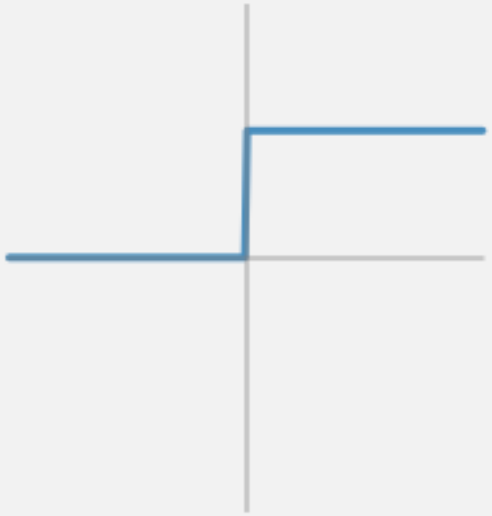
tanh

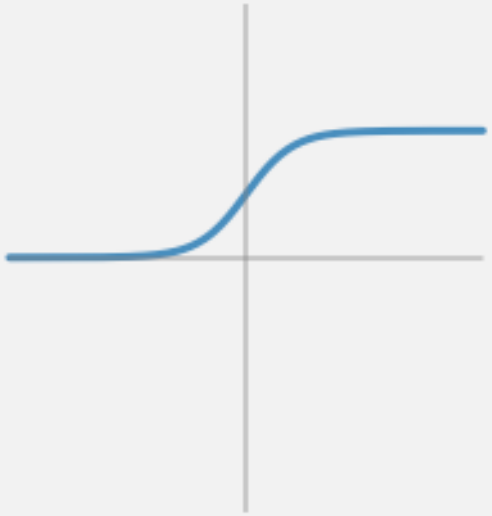


ReLu

Other Popular Activation Functions

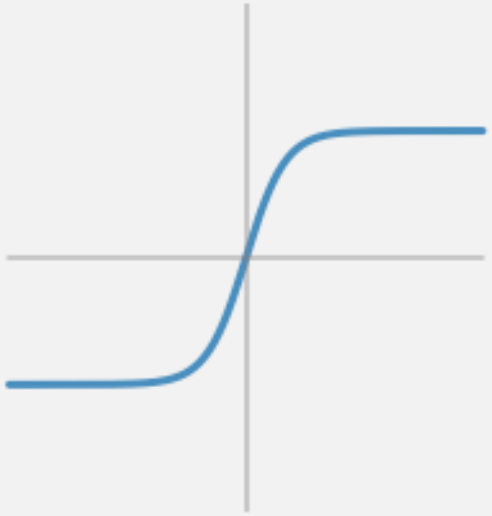
perceptron

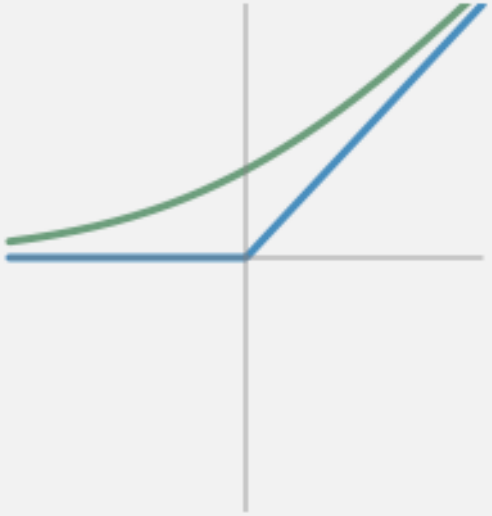




sigmoid

tanh

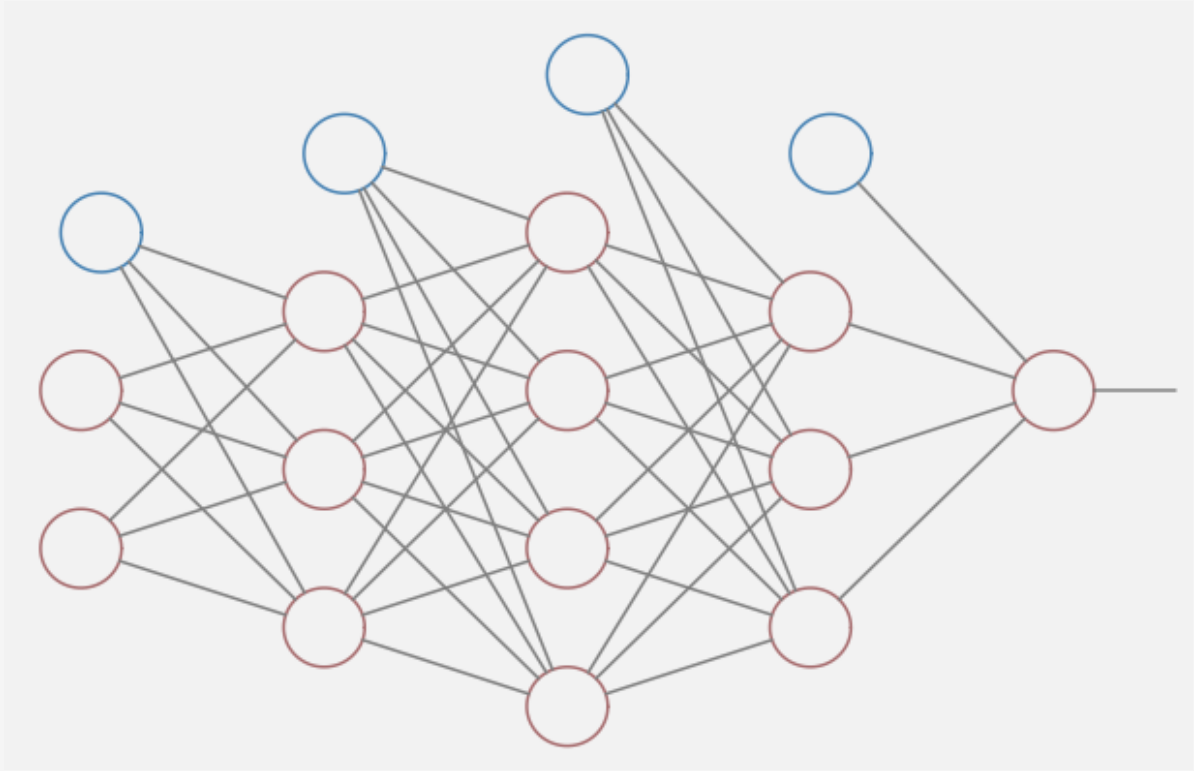




ReLu

Different Probs – Different Architectures

What if we want to do regression instead of classification?



Different Probs - Different Architectures

What if we want to do classification with more than two classes?

Pedestrian

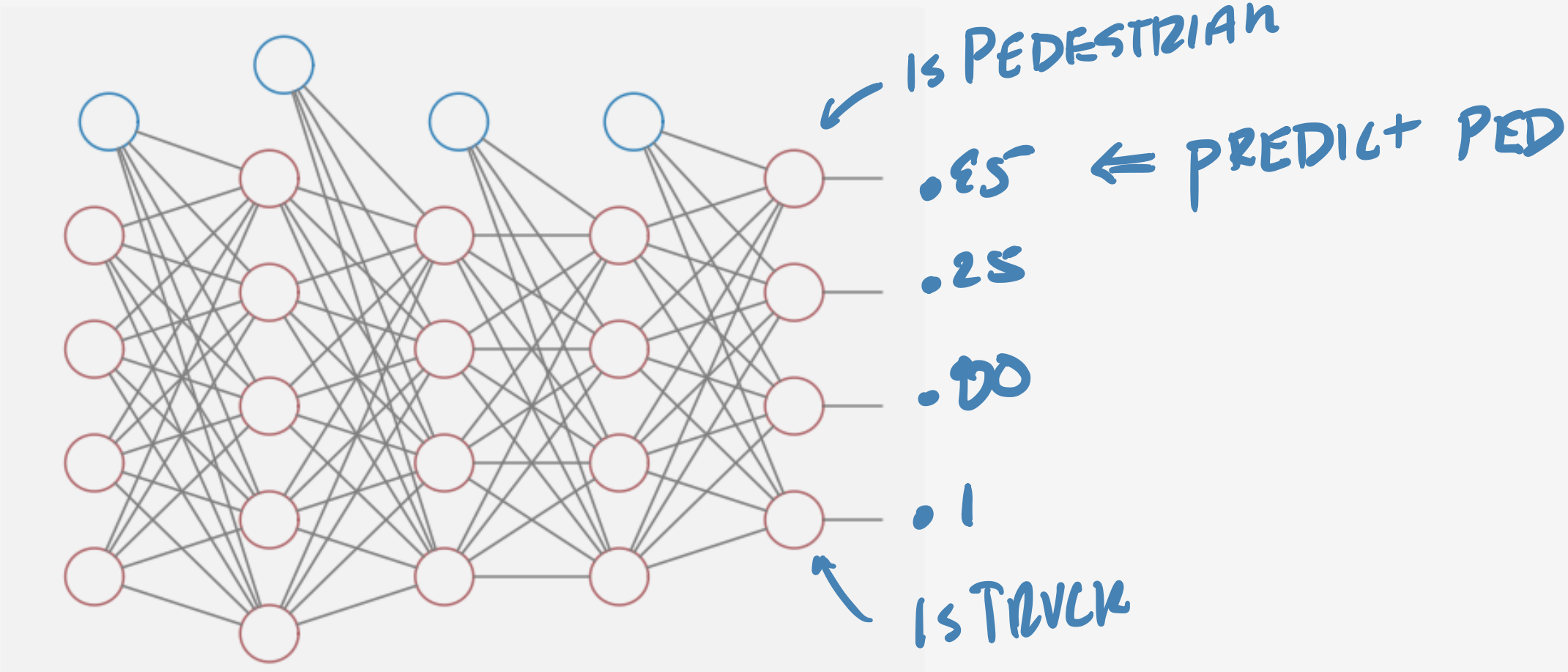
Car

Motorcycle

Truck

Different Probs – Different Architectures

What if we want to do classification with more than two classes?

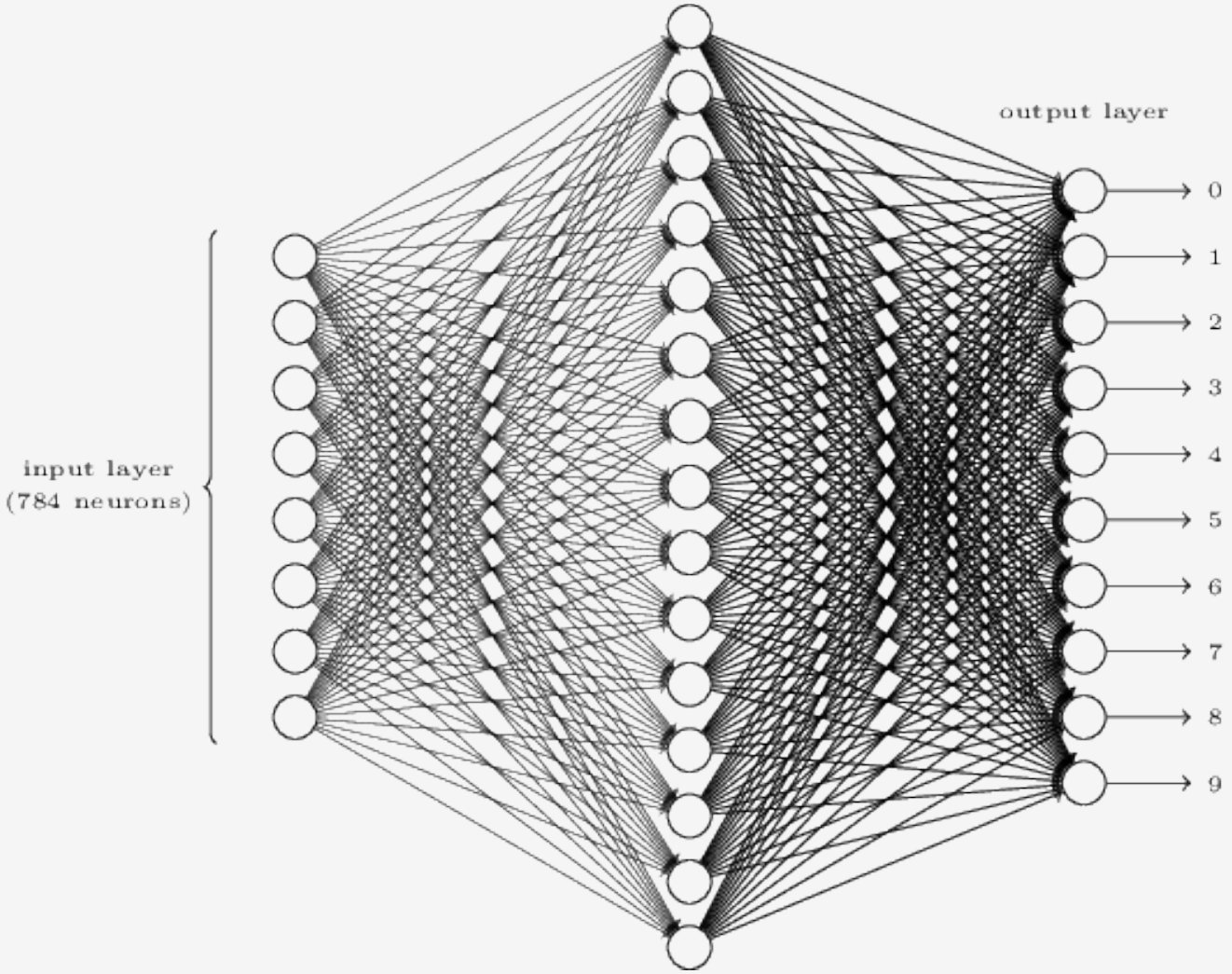


Different Probs - Different Architectures

What if we want to do classification with more than two classes?

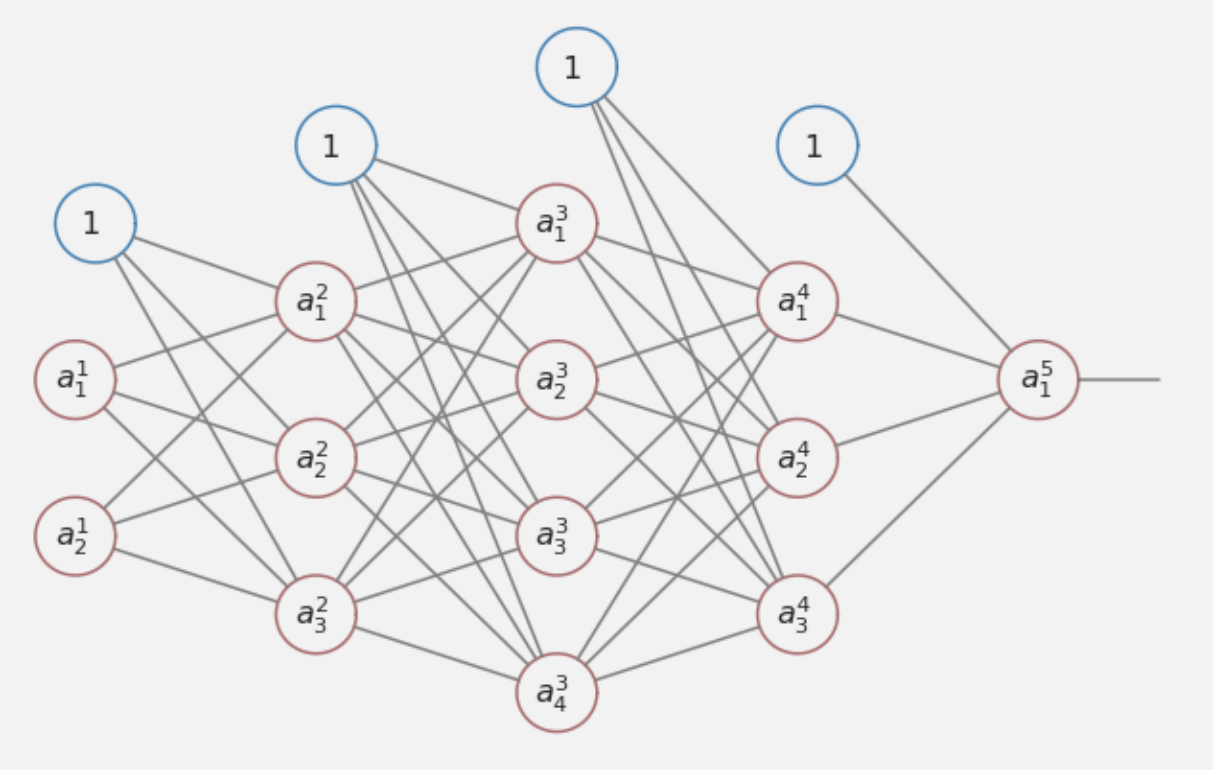
Different Probs - Different Architectures

What if we want to do classification with more than two classes?



Loss Functions

How do we know if we're doing well?



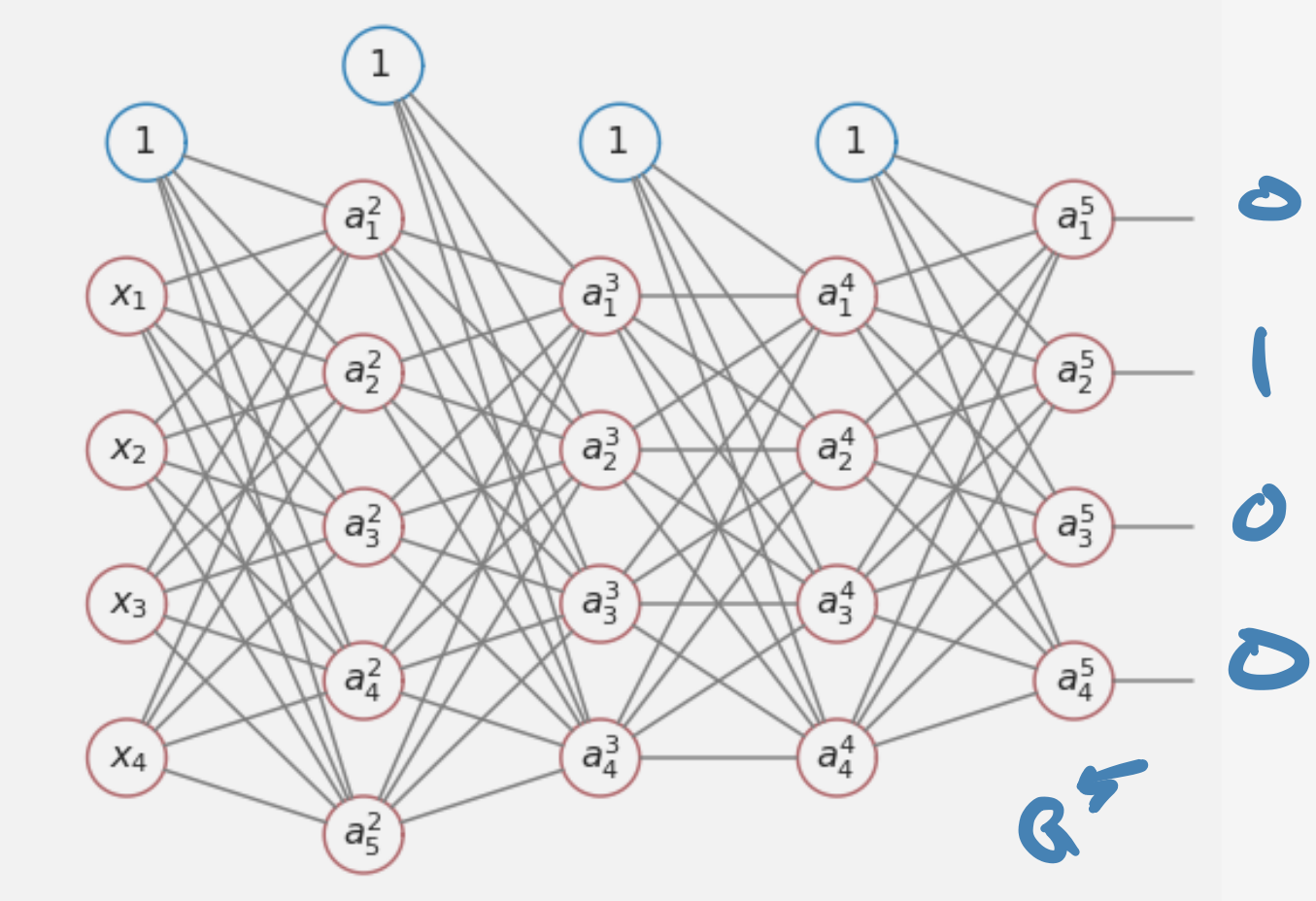
Single Output - Squared Loss

Regression or Classification

$$L(W, b, x, y) = \sum_{i=1}^{n} (y_i - a_i^5)^2$$

Loss Functions

How do we know if we're doing well?



Multiple Outputs - Squared Loss

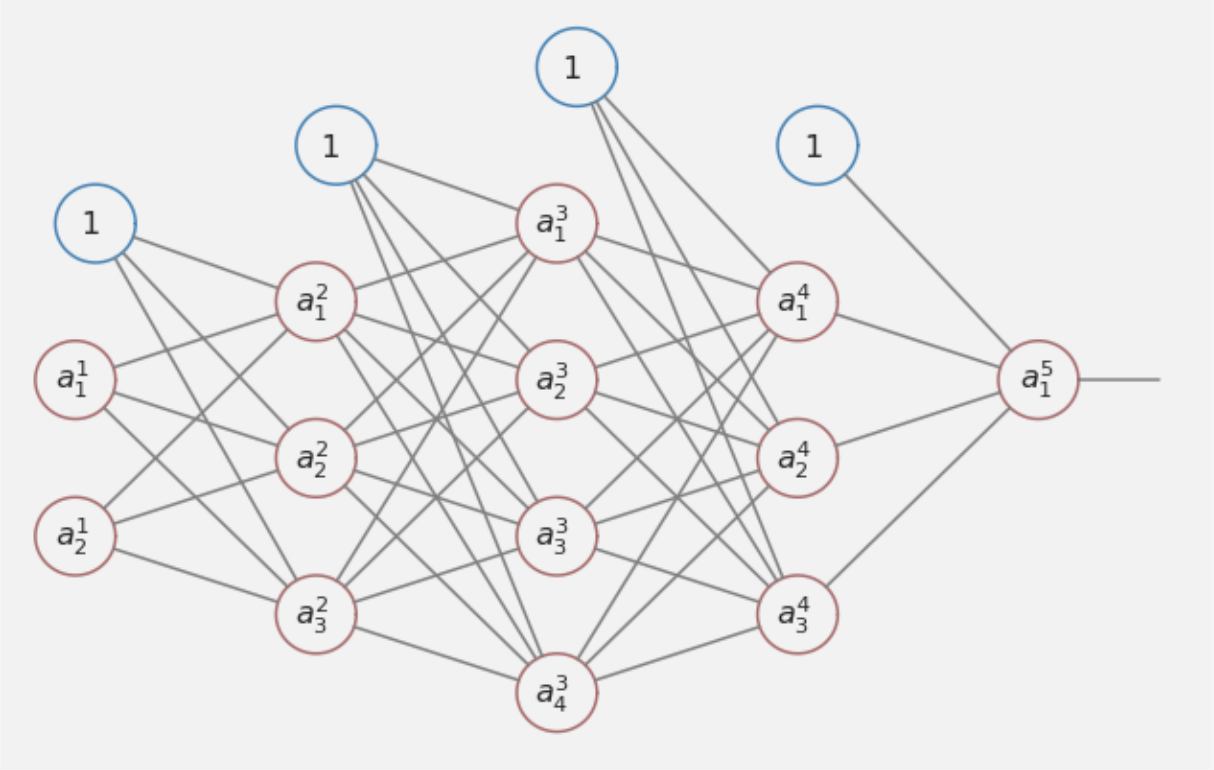
Classification

$$\sum_{i=1}^{n} \| \vec{y}(x) - a^L \|^2$$

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} .01 \\ .85 \\ .01 \\ .01 \end{bmatrix}$$

Loss Functions

How do we know if we're doing well?



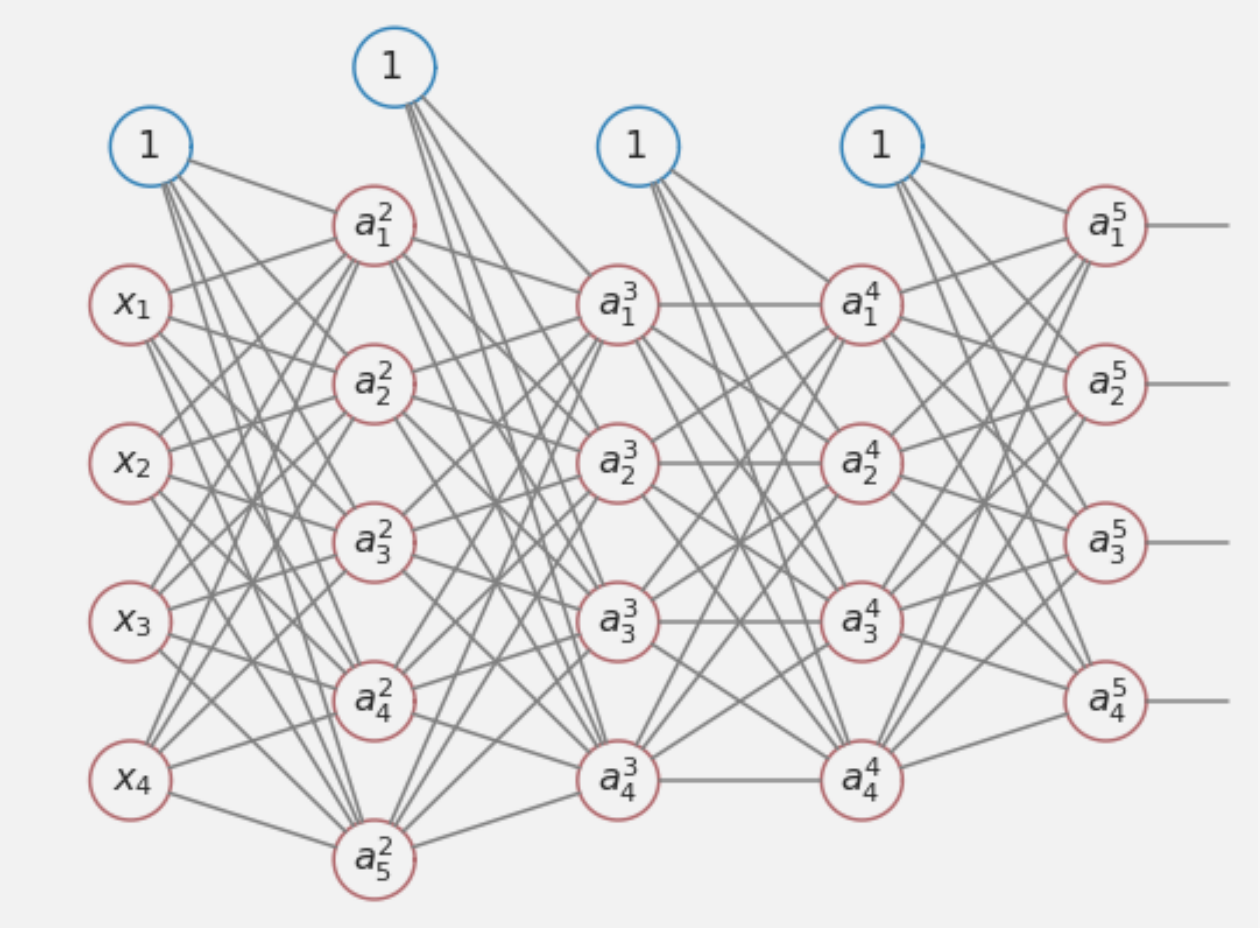
Single Output - Cross-Entropy

~~Regression or~~ Classification

$$-\sum_{i=1}^{\tilde{n}} \{ y_i \cdot \log(a^L) + (1-y_i) \log(1-a^L) \}$$

Loss Functions

How do we know if we're doing well?



Multiple Outputs – SoftMax

Classification

Training a Feed-Forward NN

So how do we train such a monstrosity?

Training a Feed-Forward NN

So how do we train such a monstrosity?

Pretty much the usual way

Define a loss function: $\text{Loss}(\mathbf{x}, \mathbf{y}; \mathbf{W}, \mathbf{b}) = L(\mathbf{W}, \mathbf{b})$

Training a Feed-Forward NN

So how do we train such a monstrosity?

Pretty much the usual way

Define a loss function: $\mathrm{Loss}(\mathbf{x}, \mathbf{y}; \mathbf{W}, \mathbf{b}) = L(\mathbf{W}, \mathbf{b})$

Then we just do Stochastic Gradient Descent: $w_{ij}^{\ell} \leftarrow w_{ij}^{\ell} - \eta \frac{\partial L}{\partial w_{ij}^{\ell}}, \quad b_i^{\ell} \leftarrow b_i^{\ell} - \eta \frac{\partial L}{\partial b_i^{\ell}}$

No problem right …

Training a Feed-Forward NN

So how do we train such a monstrosity?

Pretty much the usual way

Define a loss function: $\text{Loss}(\mathbf{x}, \mathbf{y}; \mathbf{W}, \mathbf{b}) = L(\mathbf{W}, \mathbf{b})$

Then we just do Stochastic Gradient Descent: $w_{ij}^{\ell} \leftarrow w_{ij}^{\ell} - \eta \frac{\partial L}{\partial w_{ij}^{\ell}}, \quad b_i^{\ell} \leftarrow b_i^{\ell} - \eta \frac{\partial L}{\partial b_i^{\ell}}$

No problem right …

Turns out those partial derivatives are not so easy to compute

This is where **Back Propagation** comes in. It will require:

Calculus **Linear Algebra** **An Iron Stomach**