CS167: Machine Learning

Multilayer Perceptron (MLP)
PyTorch Basics

Thursday, April 4th, 2024

Drake UNIVERSITY

Announcements

- **Project#1**
  - due tonight 04/**04 by 11:59pm**
- **Quiz#2**
  - due tonight 04/**04 by 11:59pm**

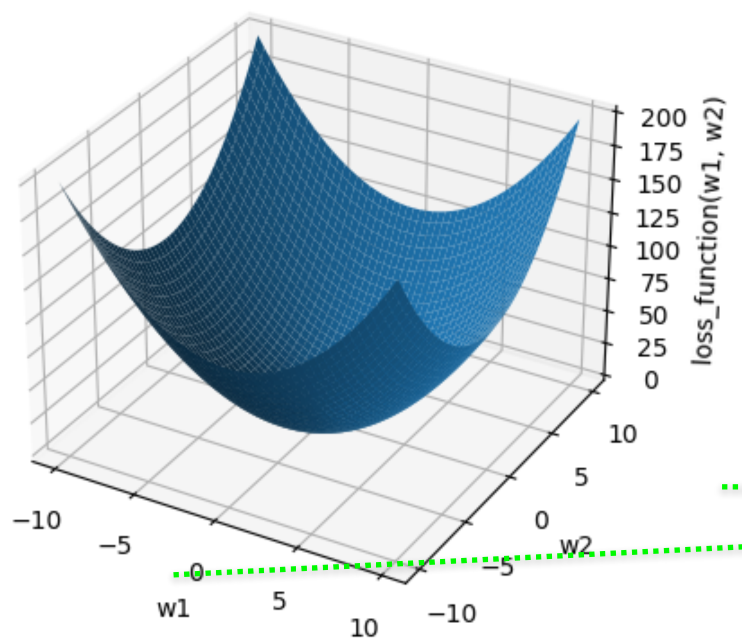
Recap: Optimization

- **minimization**: trying to find the subset of values for attributes that gives you the minimum value in the objective function
- The term objective function is generalized term which leaves room for the function to be something that we want to either **minimize** or **maximize**. The other terms used for the minimizing setting are as follows:
  - loss function
  - error function
  - cost function

Recap: Optimization Intuition

- **minimization**: trying to find the subset of values for attributes that gives you the minimum value in the objective function
- **How to reach to the minimum?**
  - we can start at an arbitrary point on the surface and gradually explore the surface until we reach the minimum value

A smooth 3D surface (each point correspond to a loss value)

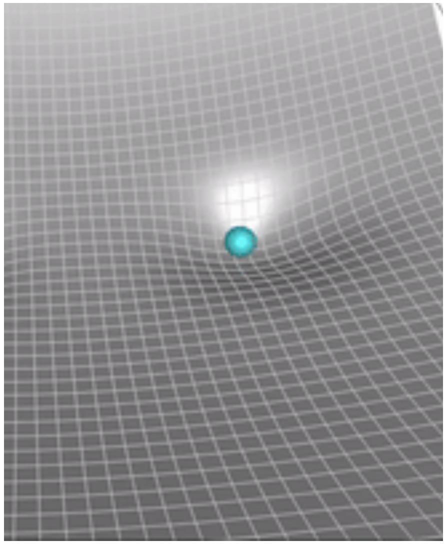


minimizing the loss function with respect to two values w_1 and w_2

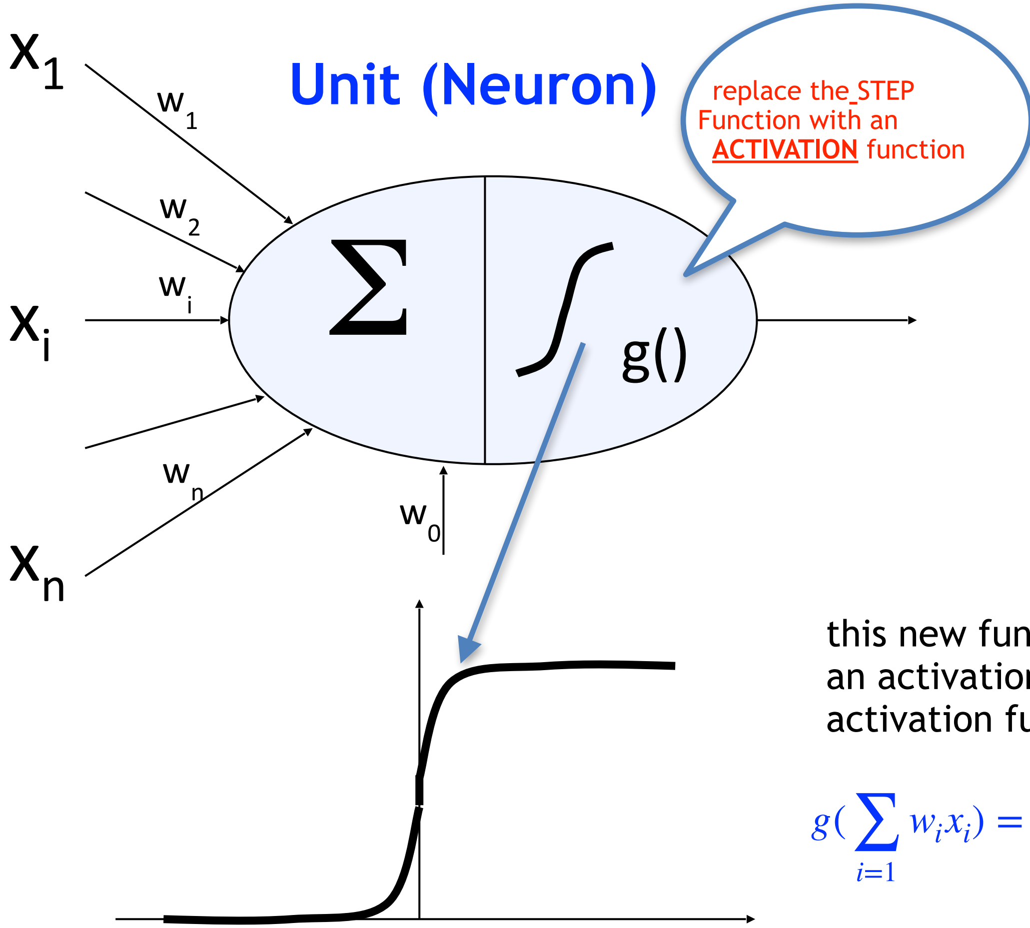
Recap: Optimization Intuition

- **minimization**: trying to find the subset of values for attributes that gives you the minimum value in the objective function

- **How to reach to the minimum?**
  - we can start at an arbitrary point on the surface and gradually explore the surface until we reach the minimum value



Recap: Make a Neuron with a Differentiable Function



$$y = f(\mathbf{x},\mathbf{w}) = g(\Sigma_{i=1,\ldots,n} w_i x_i)$$

this new function is referred to as an activation function. eg, sigmoid activation function

$$g(\sum_{i=1} w_i x_i) = \frac{1}{1 + exp(-\sum_{i=1} w_i x_i)}$$

It is smooth and differentiable

Recap: Learning Weight Parameters with the Modified Neuron Model

- Instead of our simple **Perceptron Update Rule**, we can now use a better learning algorithm to learn the weight parameters : $\begin{bmatrix} w_0 \\ w_1 \\ \ldots \\ w_n \end{bmatrix}$

- But what is this new weight parameter learning algorithm?

Gradient Descent
Stochastic Gradient Descent (SGD)

Recap: Gradient Descent

- Initialize the weight vector at a random position $\mathbf{w}^{\mathbf{old}}$ (random set of values)
- Keep doing the following two steps sequentially until the loss function gets to a low value (eg, below a threshold)
  - Step 1: calculate the gradient vector $\nabla E(\mathbf{w})$
  - Step 2: adjust (or update) the values of the weights based on the gradient vector computed in the previous step:

$$\mathbf{w}^{\mathbf{new}} = \mathbf{w}^{\mathbf{old}} - \eta \nabla \mathbf{E}(\mathbf{w})$$

a fixed learning rate

$$\begin{bmatrix} w_0^{new} \\ w_1^{new} \\ \cdots \\ w_n^{new} \end{bmatrix} = \begin{bmatrix} w_0^{old} \\ w_1^{old} \\ \cdots \\ w_n^{old} \end{bmatrix} - \eta \begin{bmatrix} \frac{\delta E(\mathbf{w})}{\delta w_0} \\ \frac{\delta E(\mathbf{w})}{\delta w_1} \\ \cdots \\ \frac{\delta E(\mathbf{w})}{\delta w_n} \end{bmatrix}$$

partial derivative w.r.t scalar term w_0

partial derivative w.r.t scalar term w_1

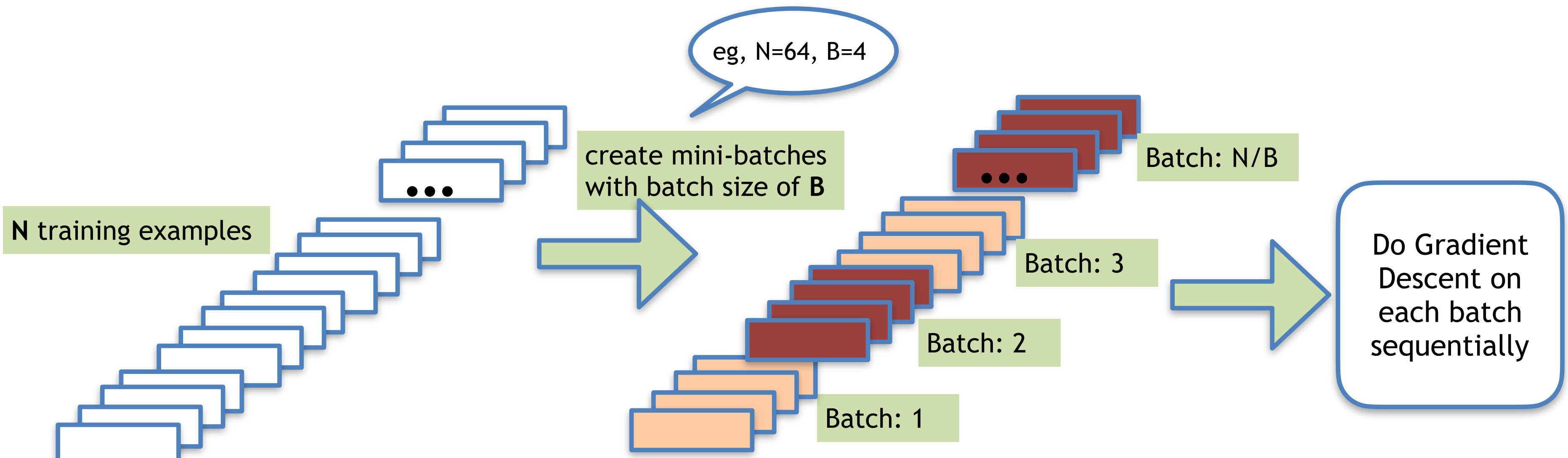
partial derivative w.r.t scalar term w_n

Gradient of $E(\mathbf{w})$ with respect to $\mathbf{w}$ is defined by the vector of partial derivatives

Recap: Stochastic Gradient Descent (SGD)

- Keep doing the Gradient Descent, but instead of using all the training samples, *use small subset of training samples* picked randomly when computing the gradient vector
  - divide the entire training data into mini batches
  - calculate the gradient vector based on that batch $\nabla E(\mathbf{w})$
  - adjust (or update) the values of the weights based on the gradient vector to that batch

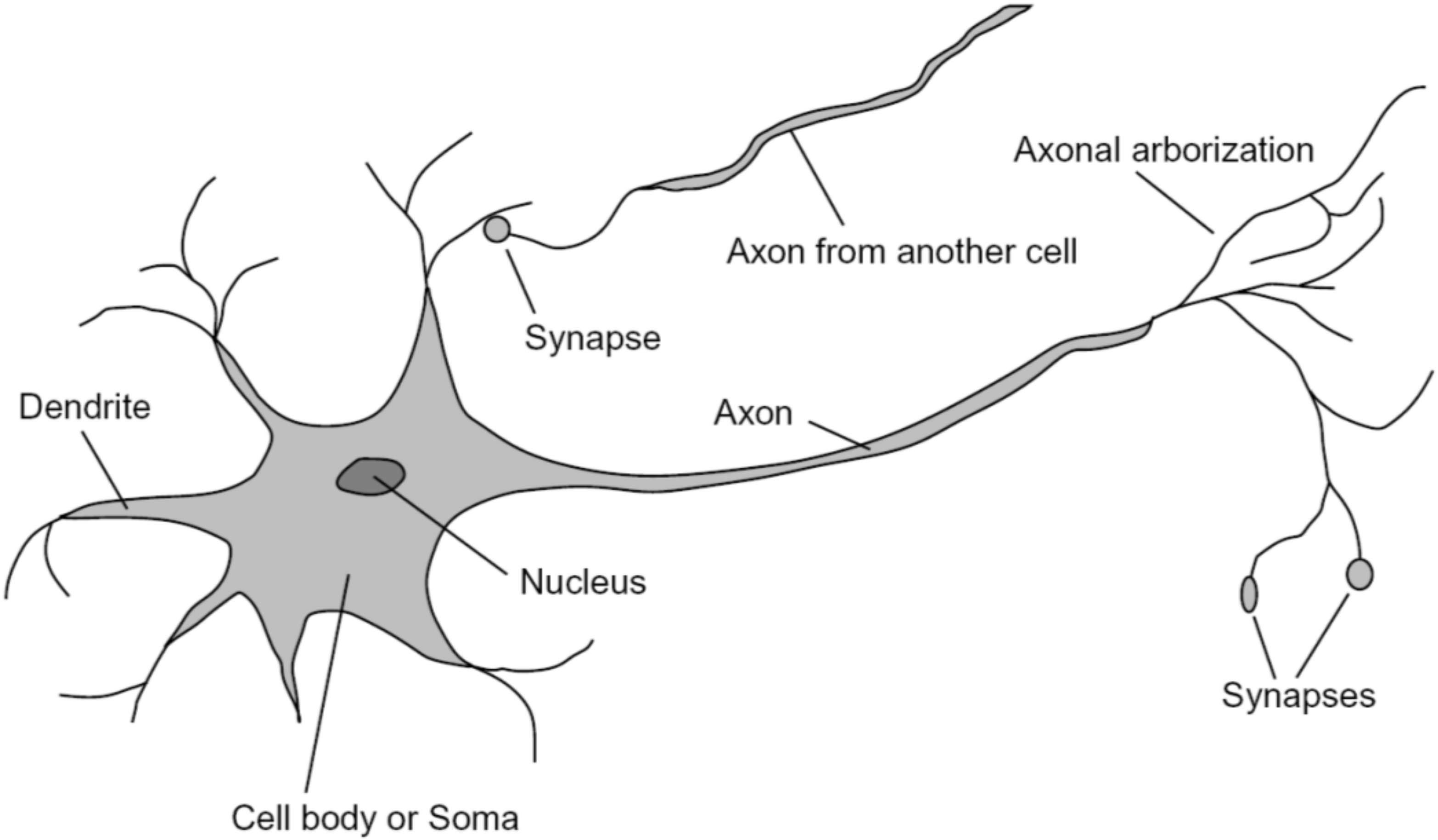
$$\mathbf{w}^{\mathbf{new}} = \mathbf{w}^{\mathbf{old}} - \eta \nabla \mathbf{E}(\mathbf{w})$$



Today's Agenda

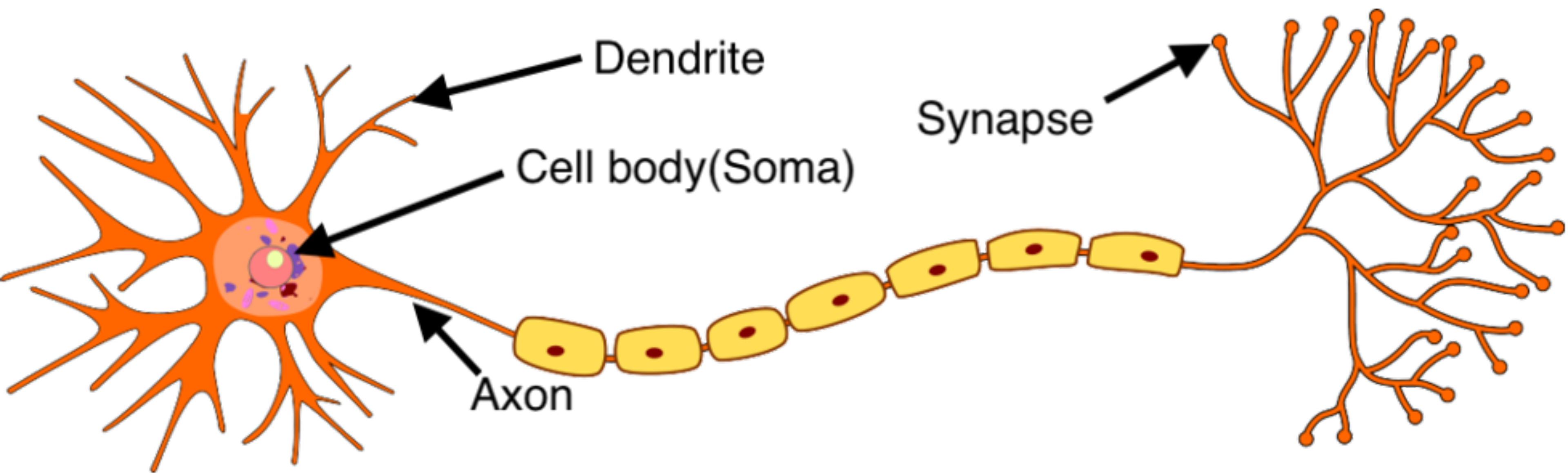
- Biological Inspiration to Connect Neurons

Inspiration: Neuron Cells

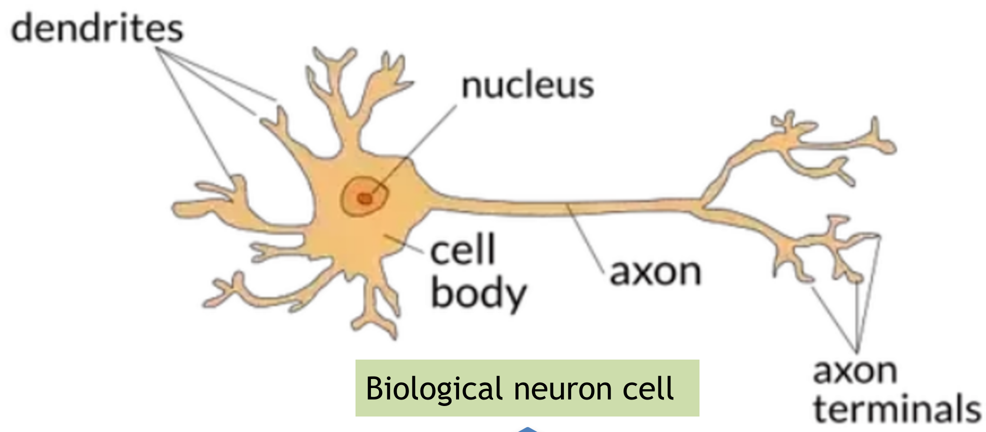


Inspiration: Neuron Cells

- Brains consist of a network of neurons:
  - Dense network: 10^{11} neurons
  - each neuron on average connected to 10^{4} other neurons
  - neuron switching times <0.001 seconds - relatively slow
  - fast recognition --> highly parallel brain



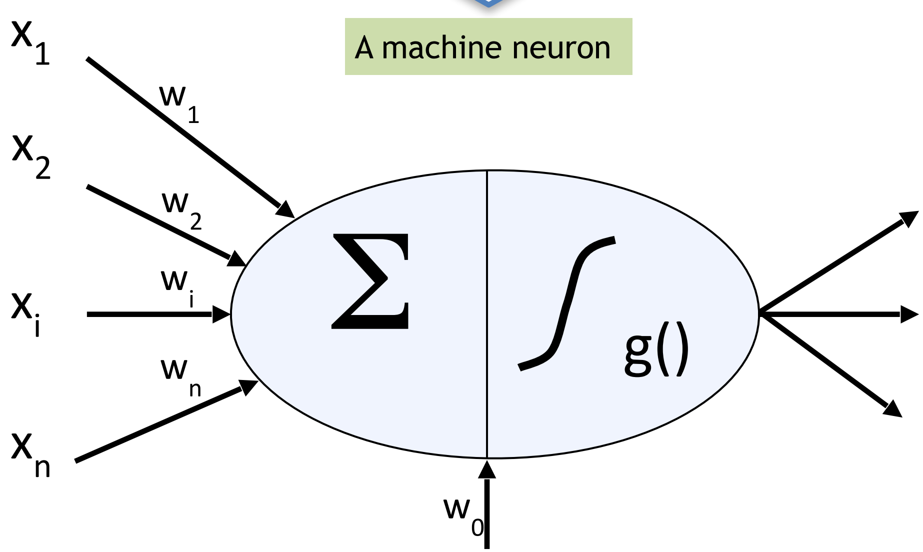
Inspiration: Neuron Cells



Biological neuron cell

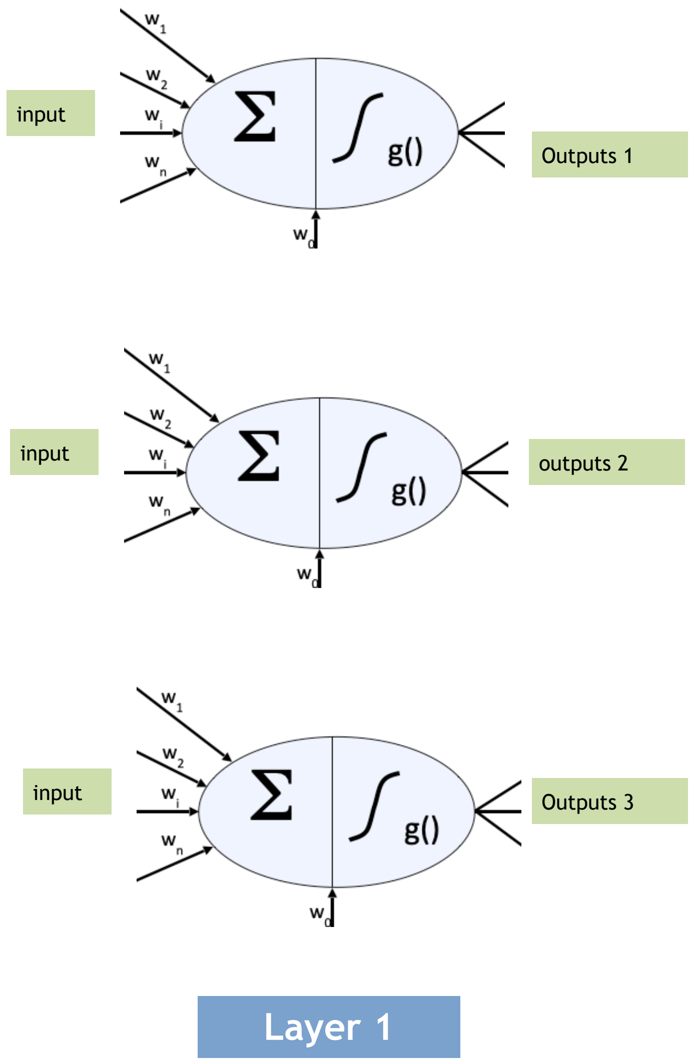
A machine neuron

Inputs

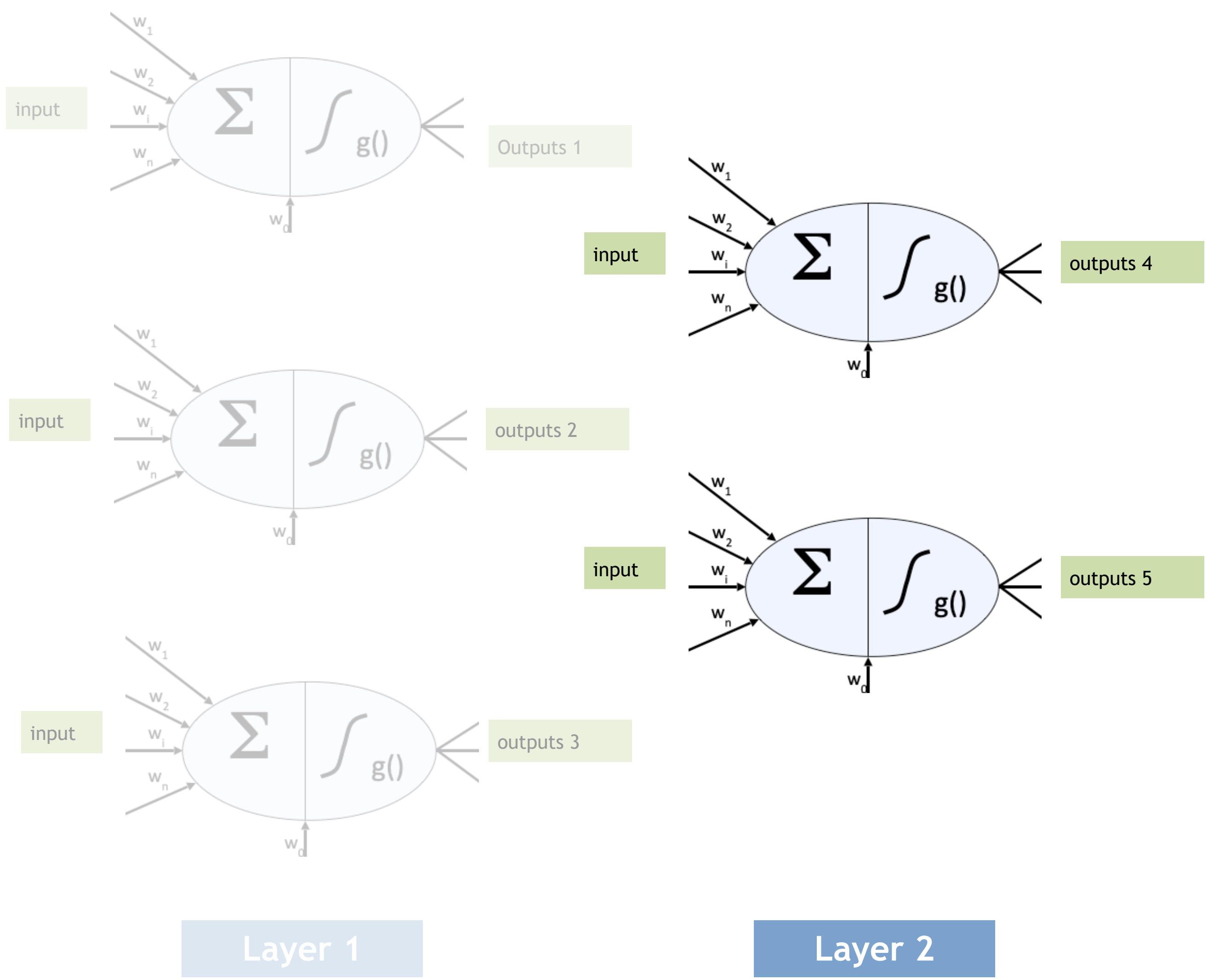


Outputs

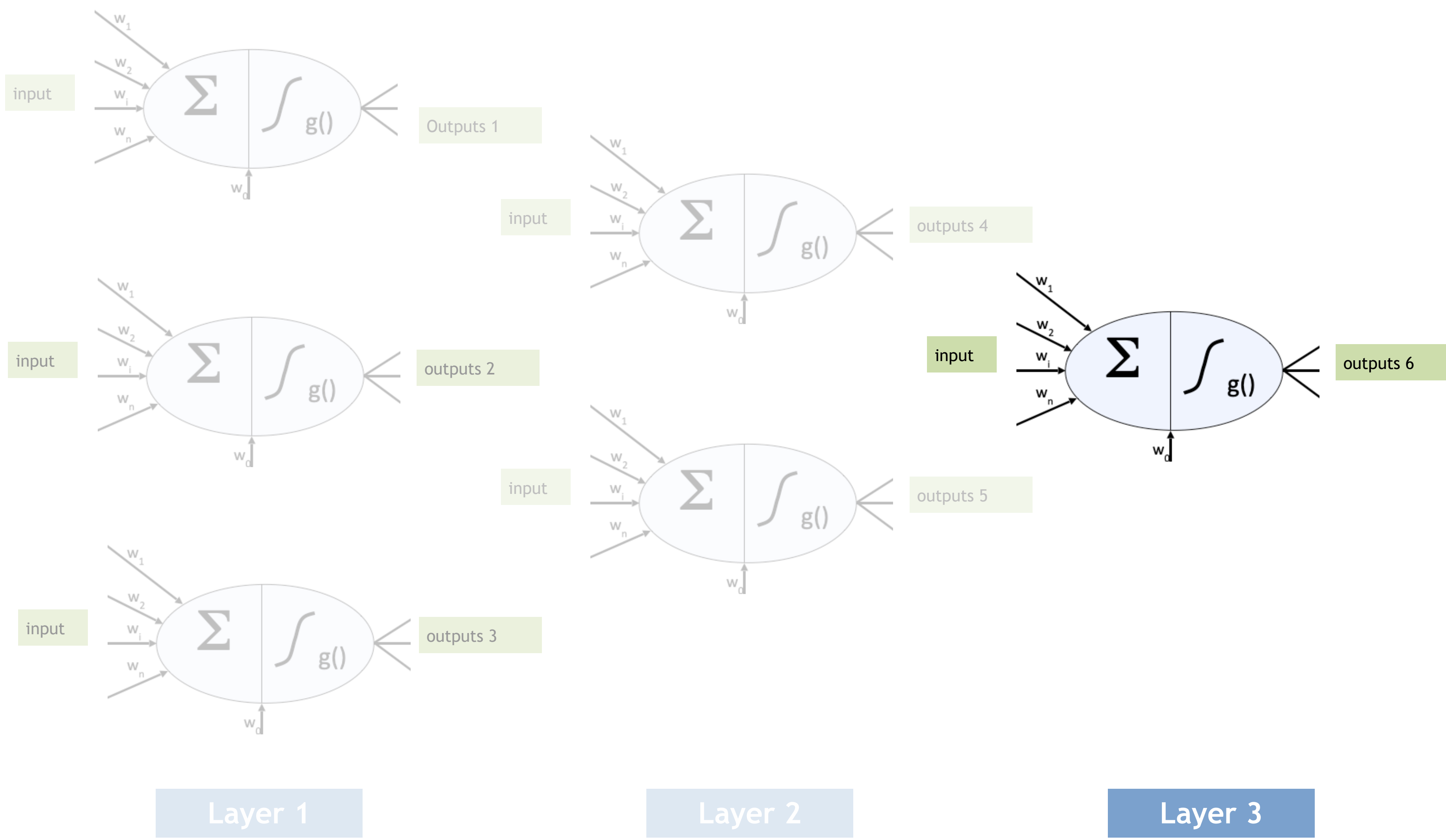
Add three neurons in the first Layer



Add a two more neuron in the second layer

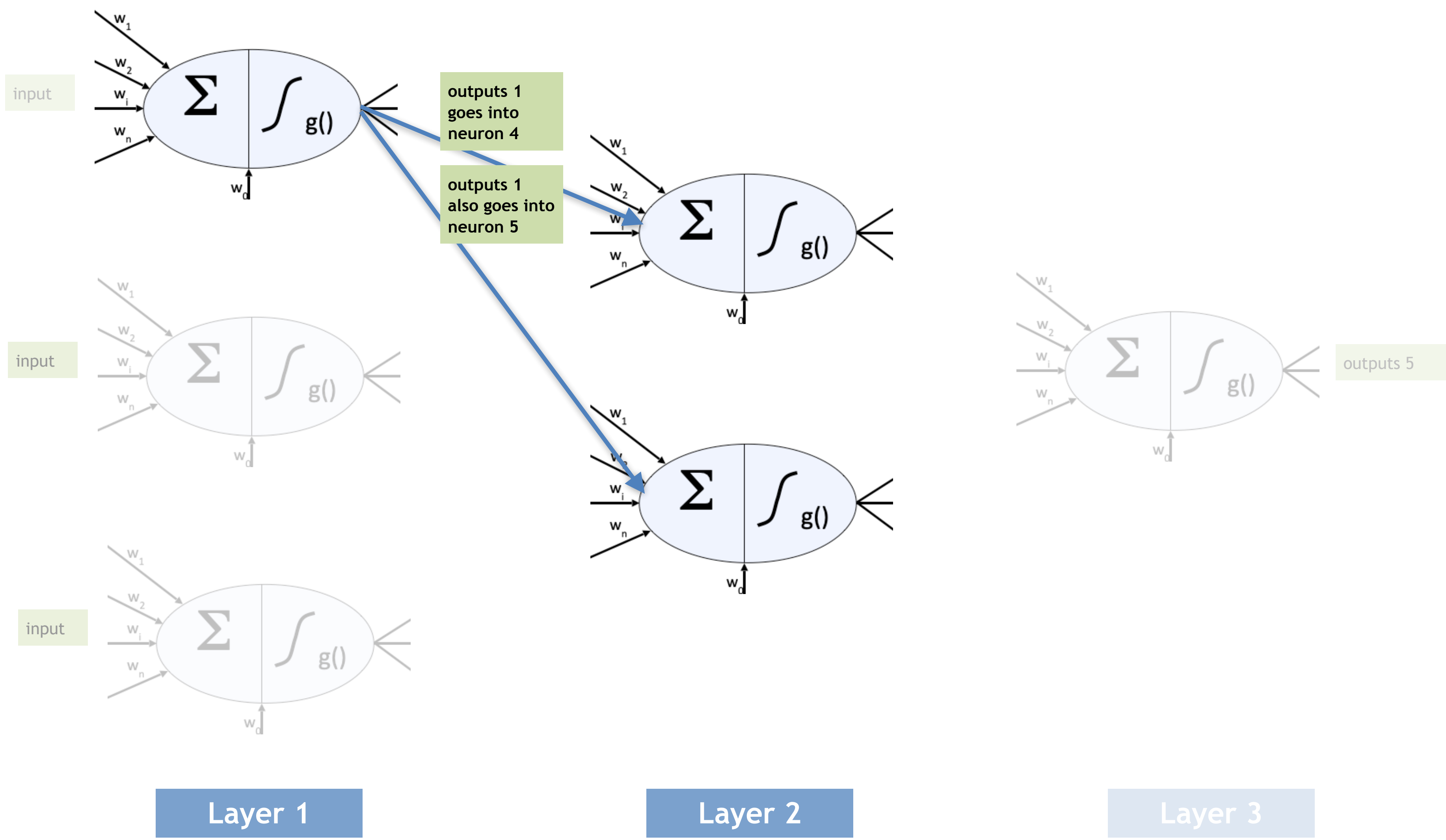


Add a two more neuron in the third layer



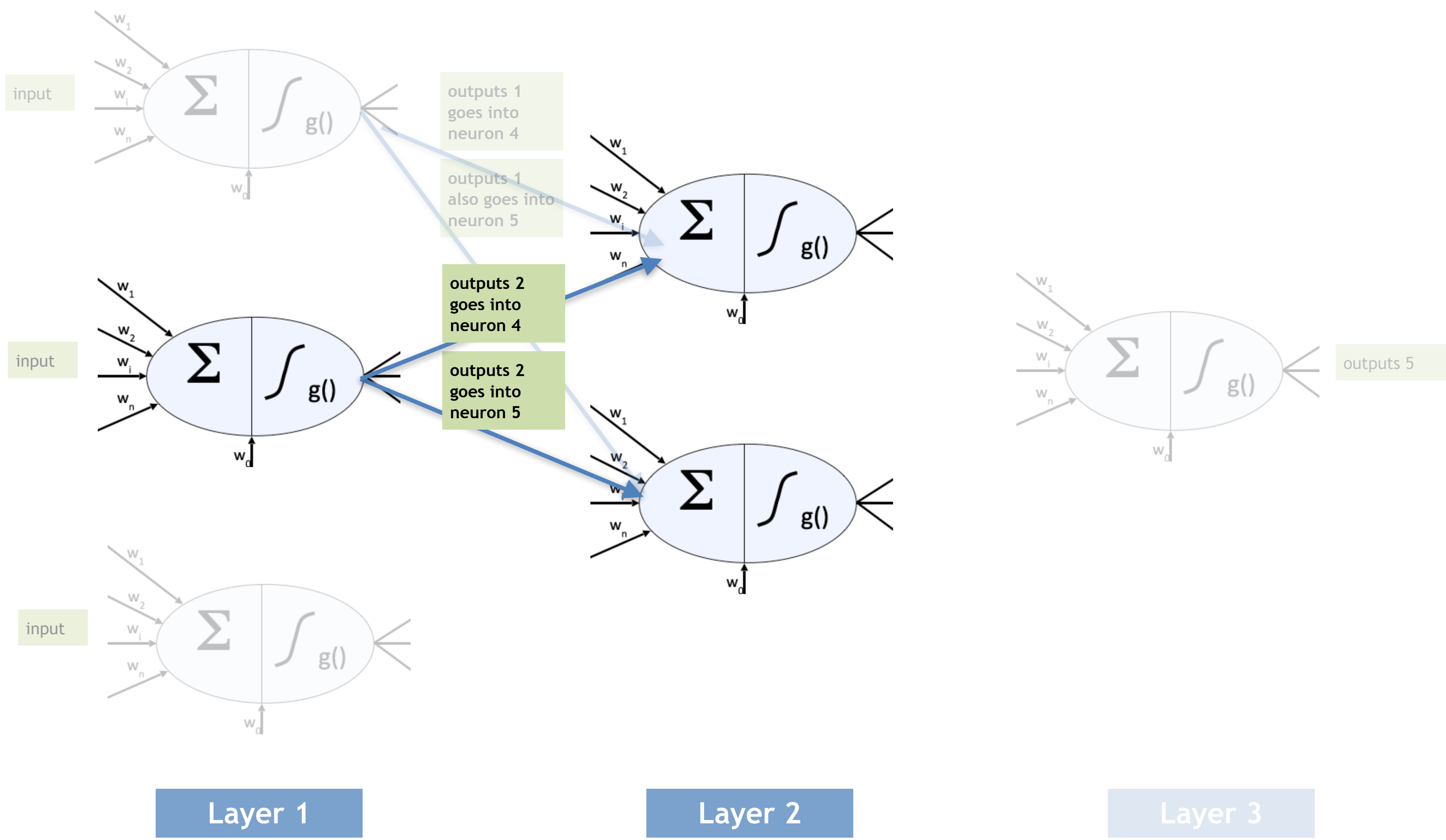
How can we Connect the Layers?

- Dense connection: connect neurons in between **Layer 1** and **Layer 2**



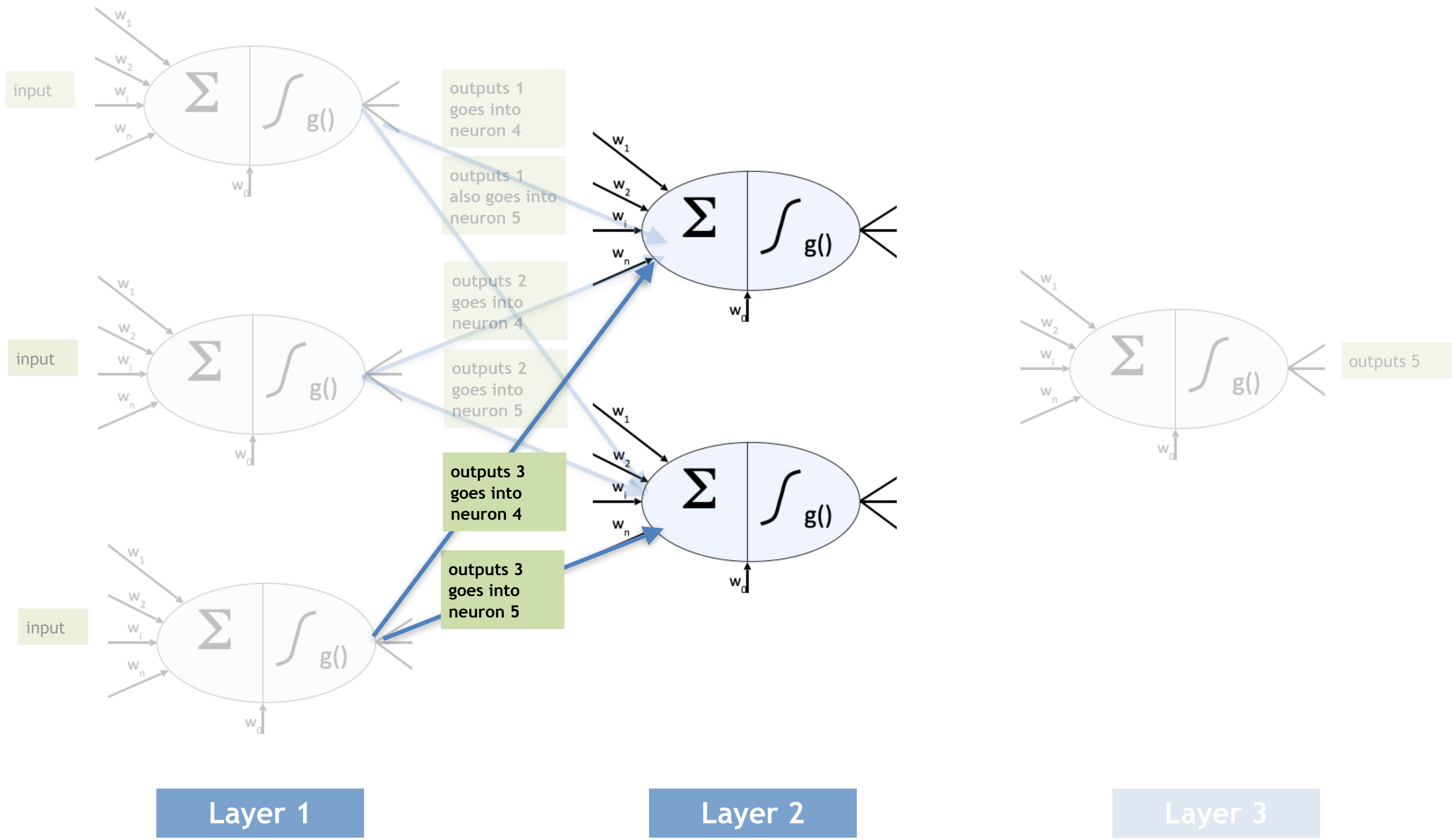
How can we Connect the Layers?

- Dense connection: connect neurons in between **Layer 1** and **Layer 2**



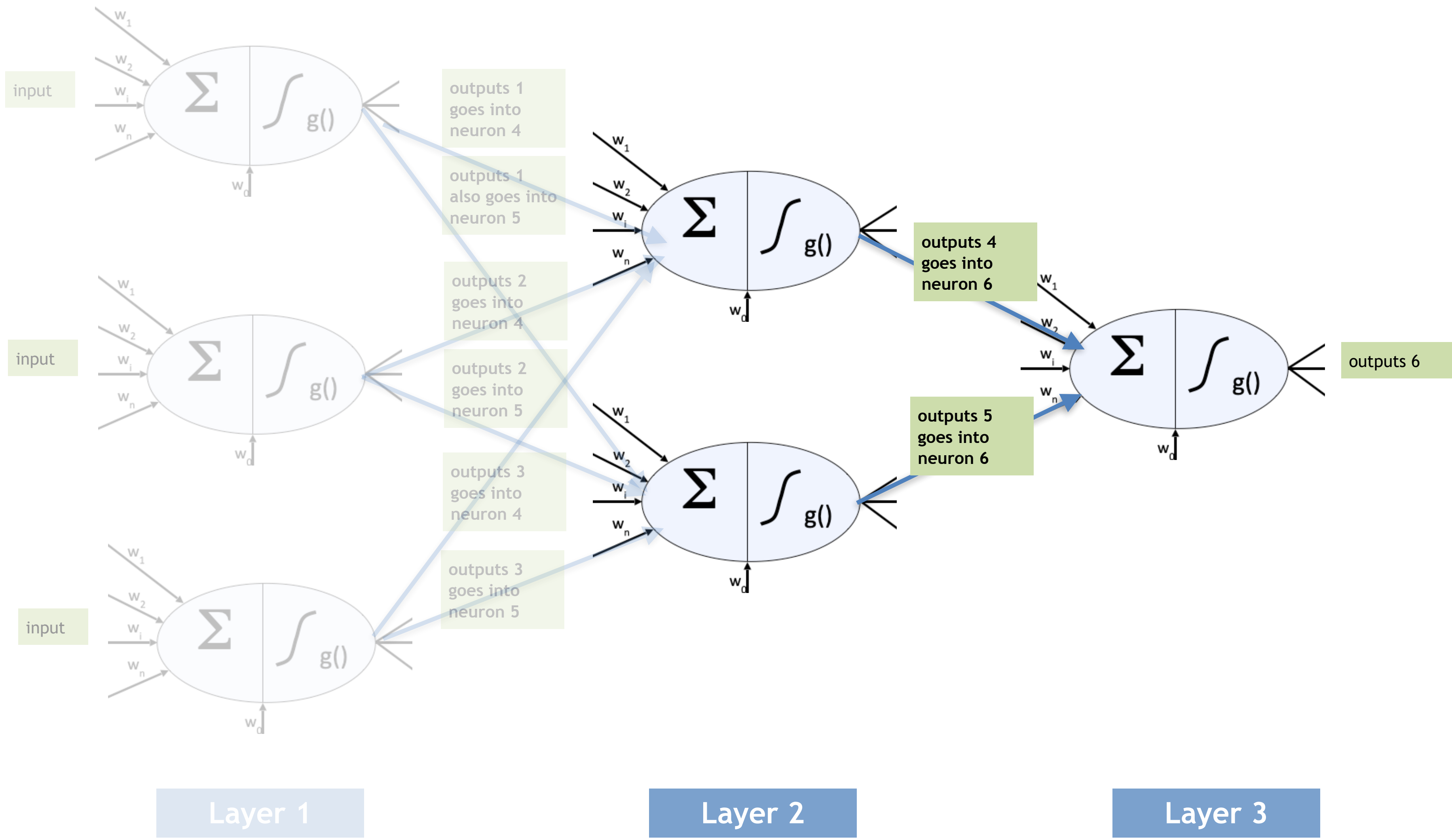
How can we Connect the Layers?

- Dense connection: connect neurons in between **Layer 1** and **Layer 2**



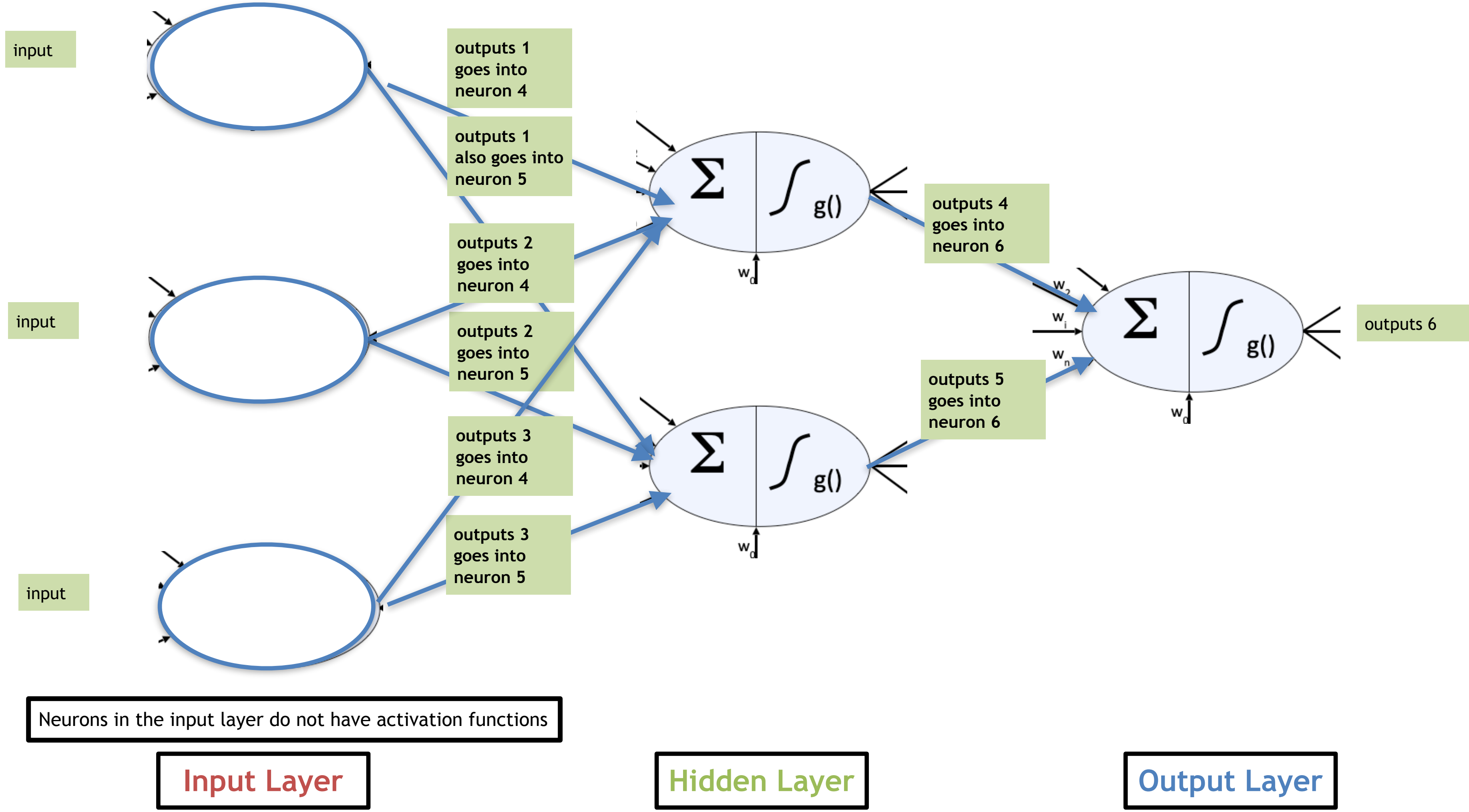
How can we Connect the Layers?

- Dense connection: connect neurons in between **Layer 2** and **Layer 3**



1-Hidden Layer Neural Network

- We created our first multilayer perceptron (MLP)
- Any layers in between input layer and output layer are called hidden layers
- Hence this MLP can also be called 1-hidden layer neural network



Group Activity

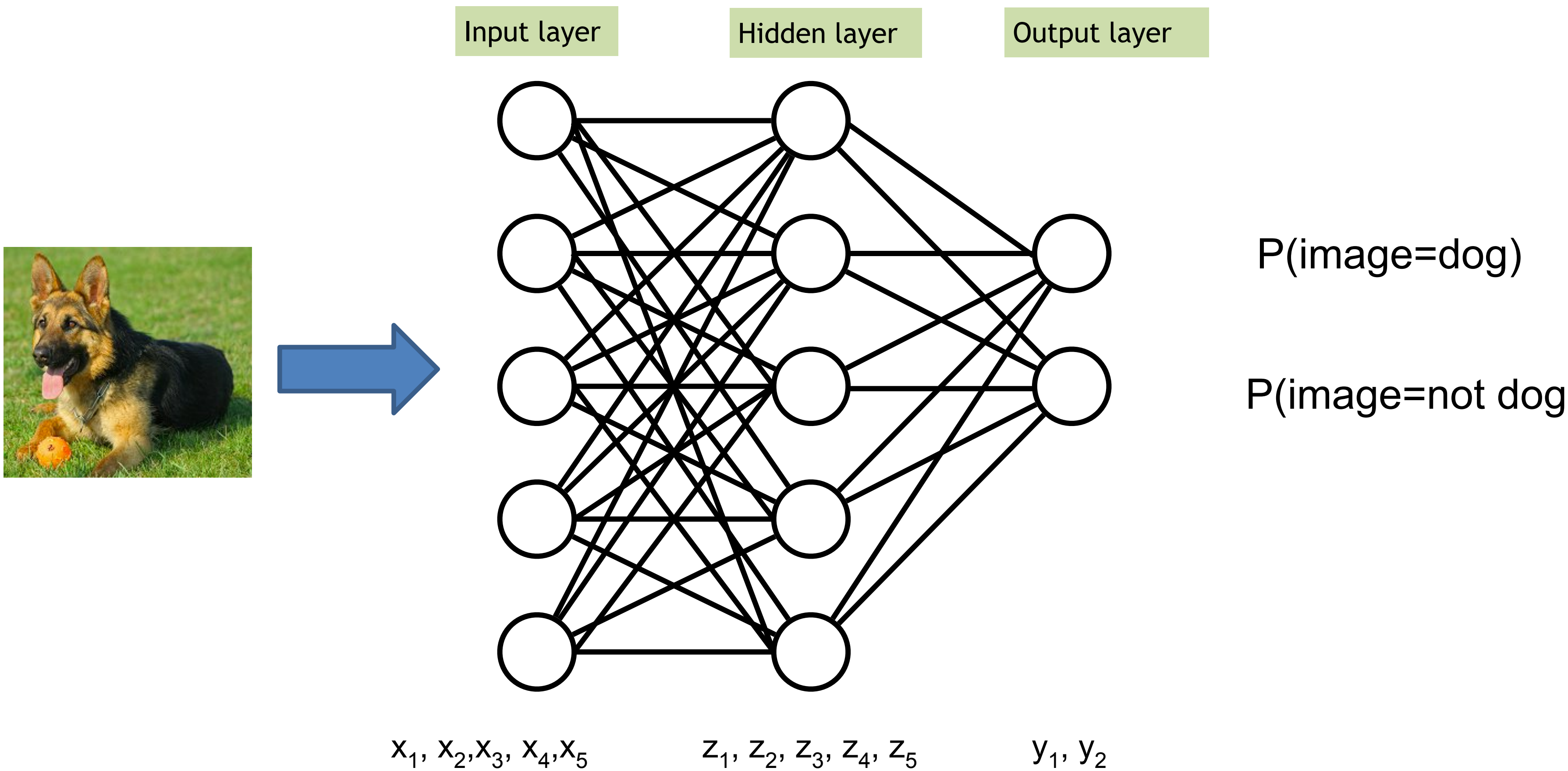
- Challenge: Devise an algorithmic means for determining whether a photo contains a Dog.

Today's Agenda

- Biological Inspiration to Connect Neurons
- Multilayer Perceptrons (MLP)

Multilayer Perceptron

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers

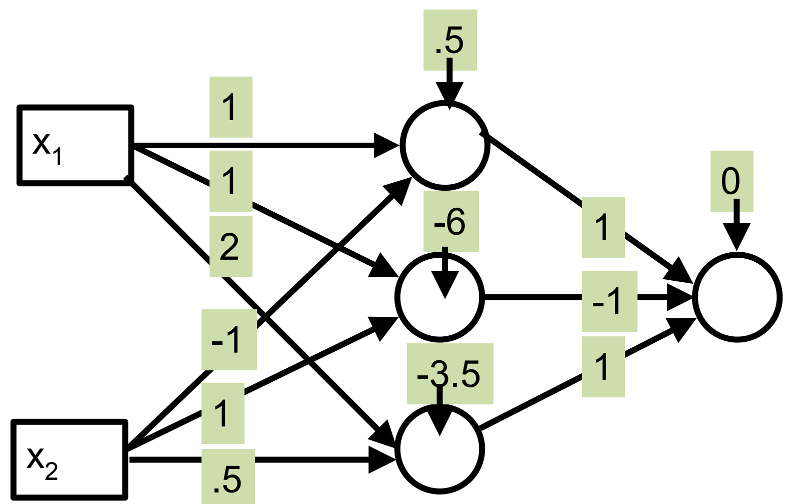


Forward Pass in Multilayer Perceptron

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output

Sample#	x_1	x_2
1	3	5
2	2	7
3	1	1
4	2	3

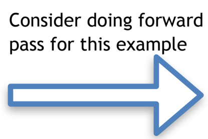


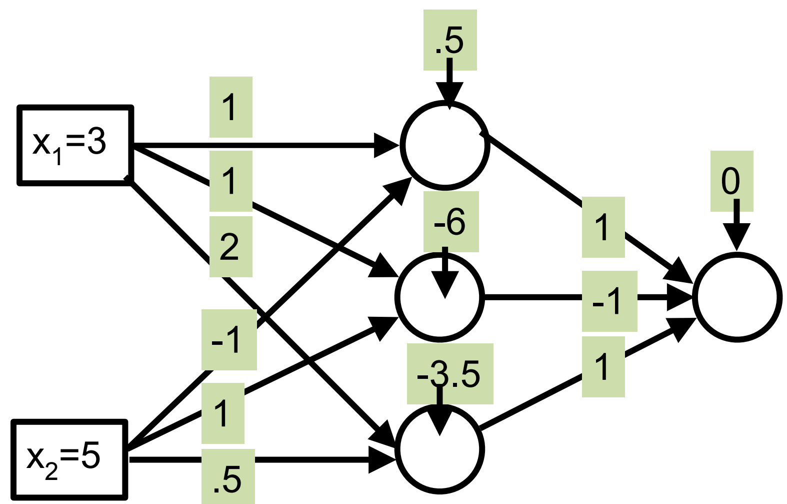
Forward Pass in Multilayer Perceptron

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output

Sample#	x_1	x_2
1	3	5
2	2	7
3	1	1
4	2	3





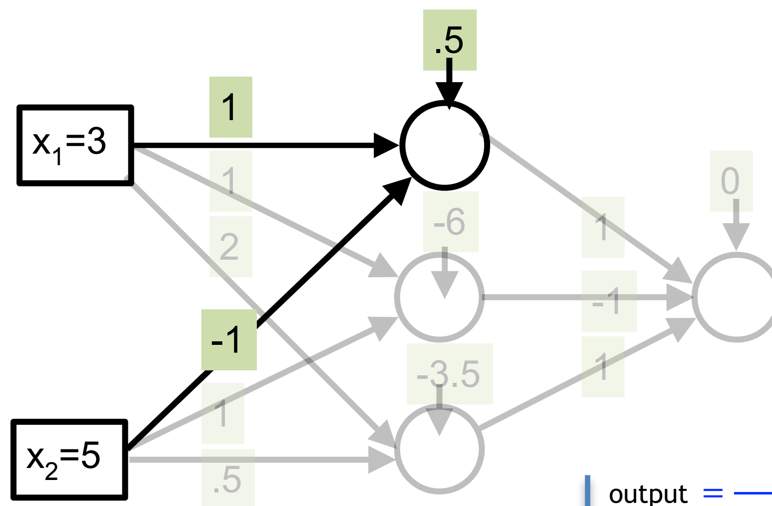
Forward Pass in Multilayer Perceptron

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output

Sample#	x_1	x_2
1	3	5
2	2	7
3	1	1
4	2	3

Consider doing forward pass for this example



$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = [w_0 \quad w_1 \quad w_2]\begin{bmatrix}1\\3\\5\end{bmatrix} = [0.5 \quad 1 \quad -1]\begin{bmatrix}1\\3\\5\end{bmatrix} = (1*0.5 + 1*3 + (-1)*5)$$
$$= -1.5$$

$$\text{output} = \frac{1}{1+exp^{-\mathbf{w}^{\mathbf{T}}\mathbf{x}}}$$
$$= \frac{1}{1+exp^{-(-1.5)}}$$
$$= 0.1824$$

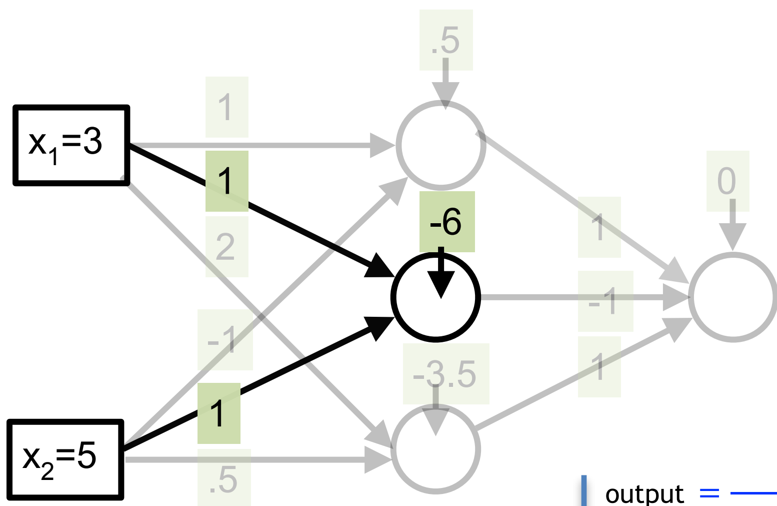
Forward Pass in Multilayer Perceptron

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output

Sample#	x_1	x_2
1	3	5
2	2	7
3	1	1
4	2	3

Consider doing forward pass for this example



$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = [w_0 \quad w_1 \quad w_2]\begin{bmatrix}1\\3\\5\end{bmatrix} = [-6 \quad 1 \quad 1]\begin{bmatrix}1\\3\\5\end{bmatrix} = (-6)*1 + 1*3 + 1*5$$
$$= 2$$

$$\text{output} = \frac{1}{1+exp^{-\mathbf{w}^{\mathbf{T}}\mathbf{x}}} = \frac{1}{1+exp^{-2}} = 0.8807$$

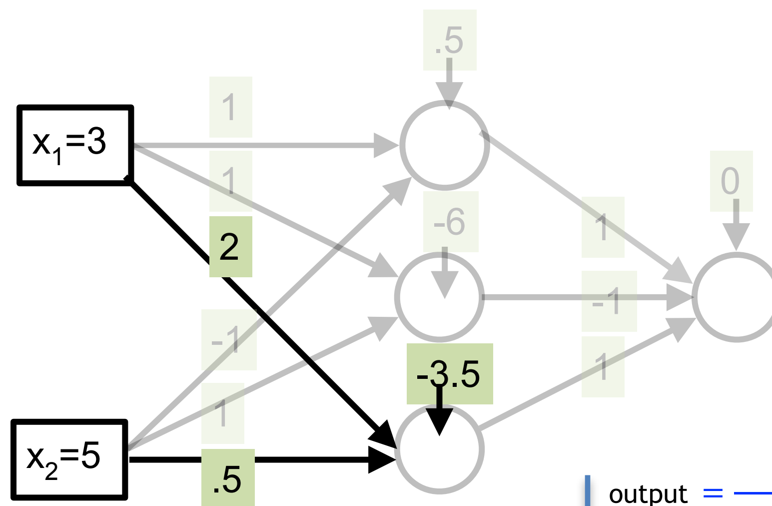
Forward Pass in Multilayer Perceptron

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output

Sample#	x_1	x_2
1	3	5
2	2	7
3	1	1
4	2	3

Consider doing forward pass for this example



$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = [w_0 \quad w_1 \quad w_2]\begin{bmatrix}1\\3\\5\end{bmatrix} = [-3.5 \quad 2 \quad 0.5]\begin{bmatrix}1\\3\\5\end{bmatrix} = (-3.5)*1 + 2*3 + 0.5*5$$
$$= 5$$

$$\text{output} = \frac{1}{1+exp^{-\mathbf{w}^{\mathbf{T}}\mathbf{x}}}$$
$$= \frac{1}{1+exp^{-5}}$$
$$= 0.9933$$

MLP Forward Pass Group Exercise

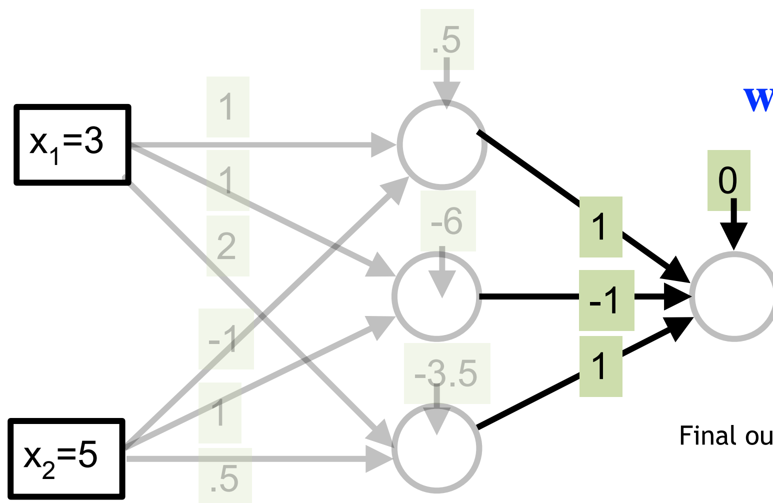
- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*
  - that number through an activation function, which produces a number as an output

$$\mathbf{w}^{\mathbf{T}} = [? \quad ? \quad .. \quad .. \quad ..] = ?$$

$$\mathbf{x} = \begin{bmatrix} 1 \\ ? \\ ? \\ .. \\ .. \end{bmatrix} = ?$$

Sample#	x1	x2
1	3	5
2	2	7
3	1	1
4	2	3

Consider doing forward pass for this example



$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = ?$$

Final output $= \dfrac{1}{1+exp^{-\mathbf{w}^{\mathbf{T}}\mathbf{x}}}$

$= \dfrac{1}{1+exp^{-?}}$

$= ?$

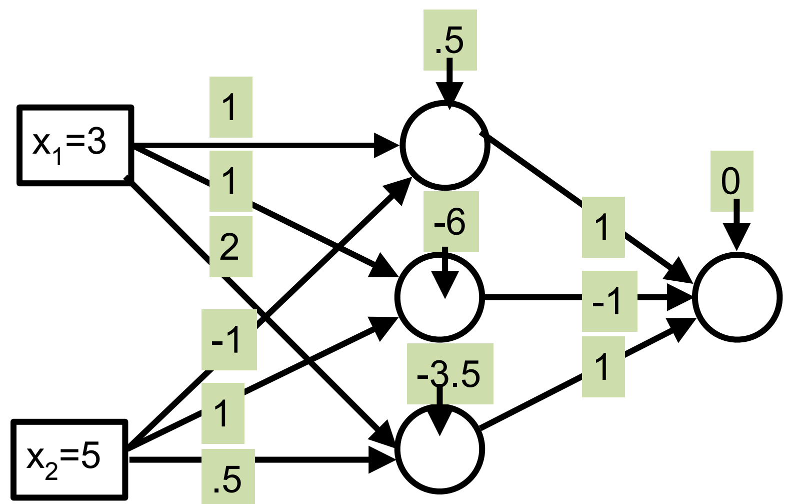
Multilayer Perceptron

- A **multilayer perceptron** is the simplest type of neural network. It consists of perceptrons (aka nodes, neurons) arranged in layers

- Each neuron contains two operations:
  - a dot product between *a weight vector (edges in the graph)* and *an input vector*, which produces a number
  - Then, that number through an activation function, which produces a number as an output

Sample#	x_1	x_2
1	3	5
2	2	7
3	1	1
4	2	3

You can do the same forward pass for the next examples sequentially

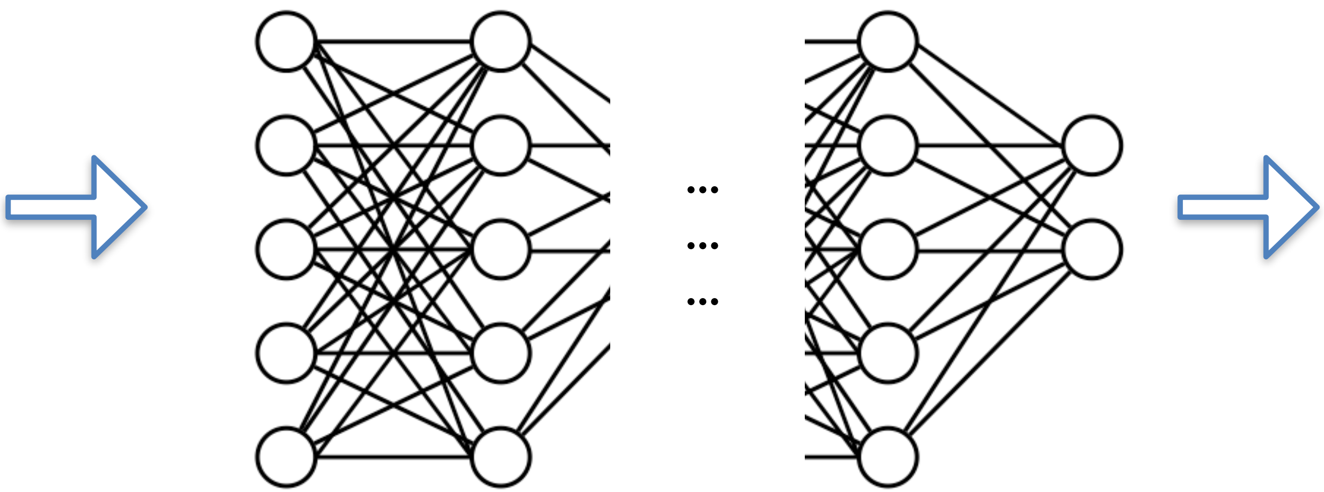


Today's Agenda

- Biological Inspiration to Connect Neurons
- Multilayer Perceptrons (MLP)
- MLP Structure

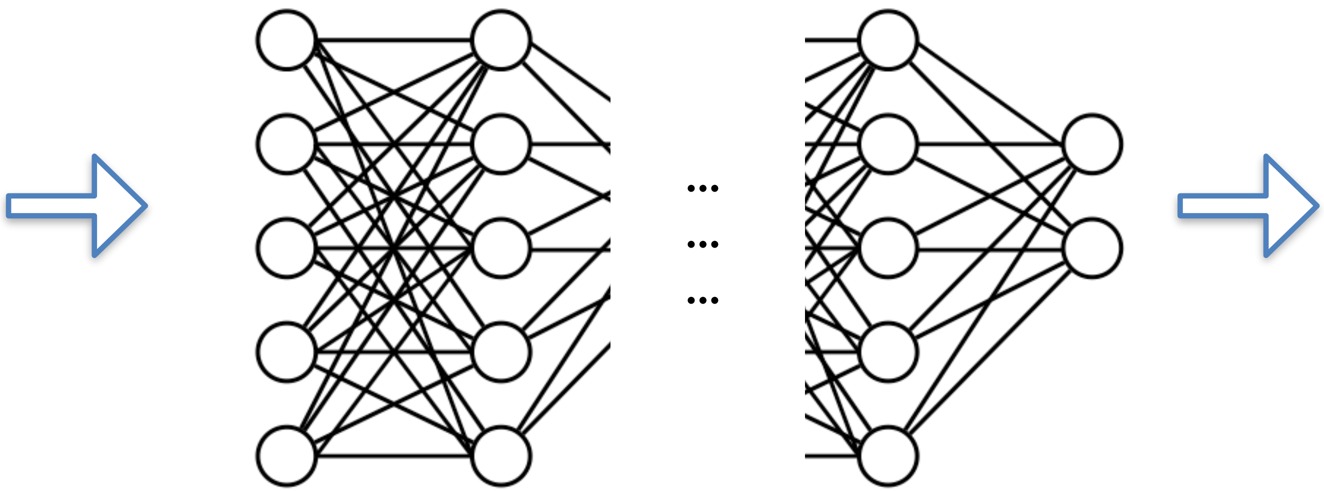
MLP (Network) Structure

- Each of these questions need to be answered before you set up your neural network:
  - how many hidden layers should I have? (depth)
  - how many neurons should be in each layer? (width)
  - what should your activation be at each of the layers?



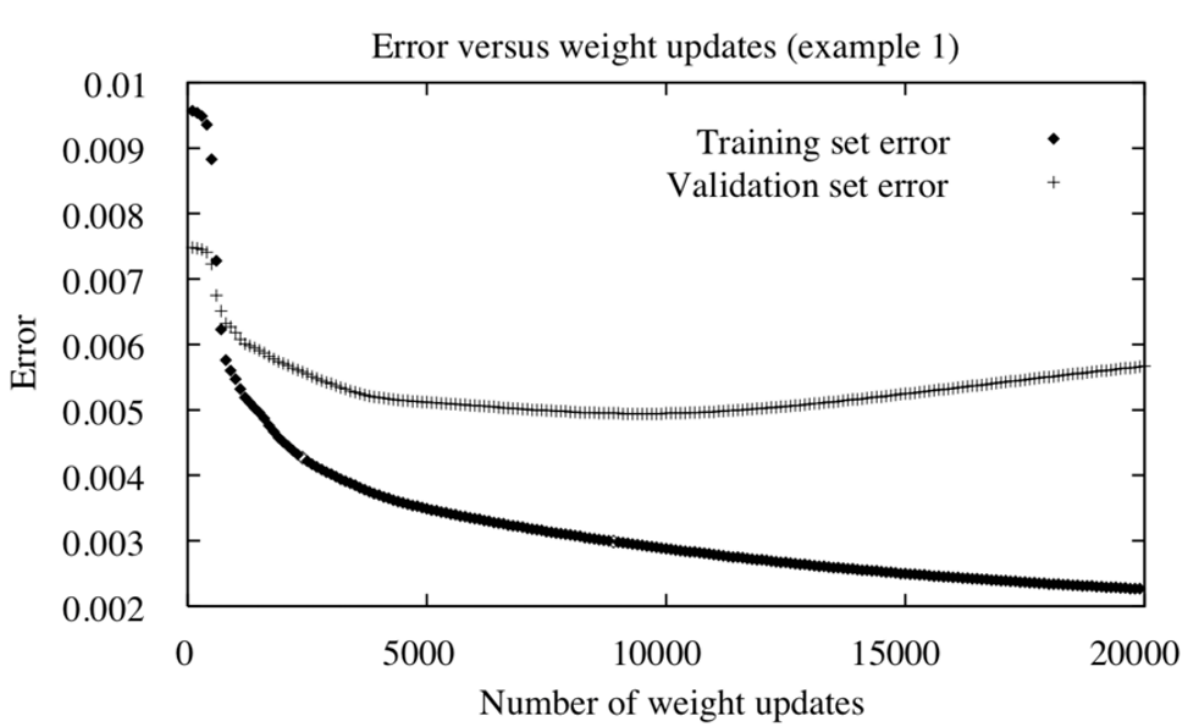
MLP (Network) Structure

- How to choose the size and structure of networks?
  - If network is too large, risk of over-fitting (data caching)
  - If network is too small, representation may not be rich enough



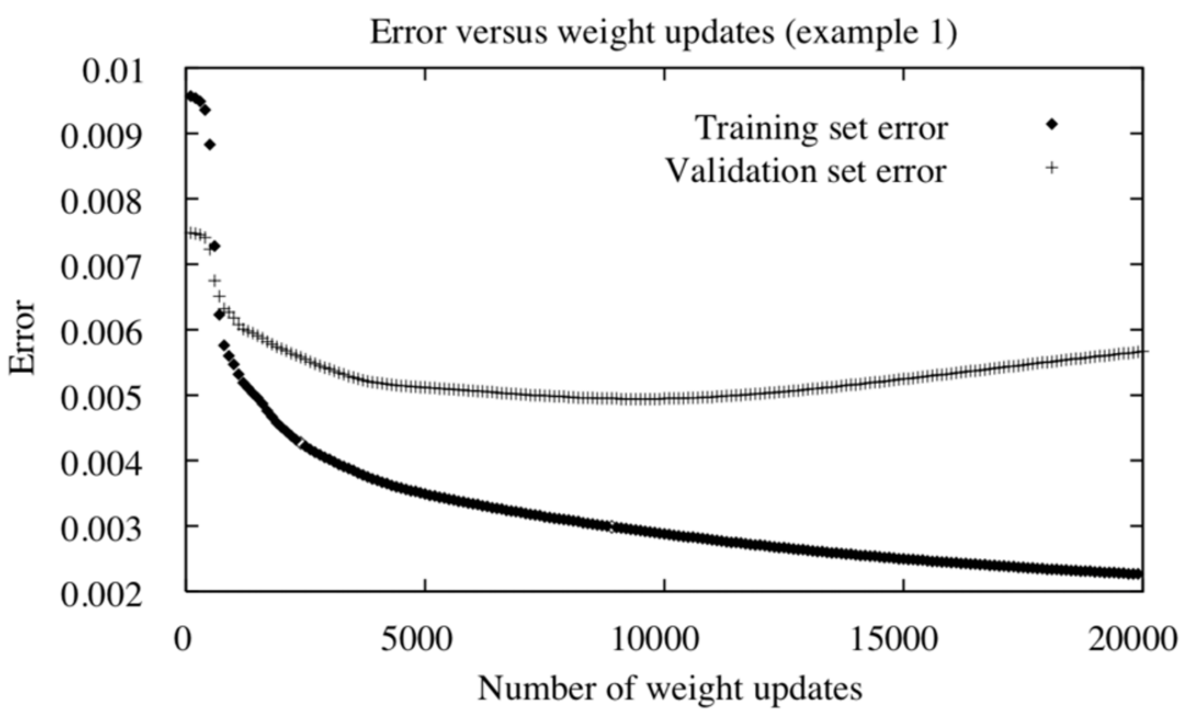
Overfitting

- MLPs, like many machine learning models, are susceptible to **overfitting.**
  - How can we recognize overfitting?
  - Given the graph below, at what point do you think our model started overfitting?



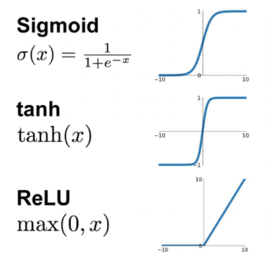
Overfitting

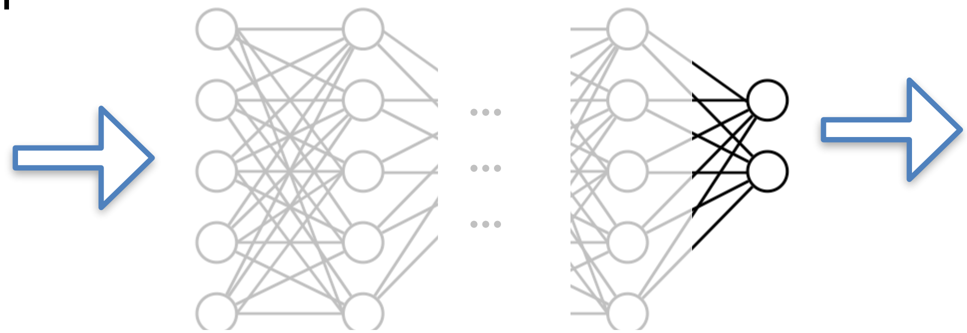
- **Overfitting** happens when the *training set error* continues to improve, but the *validation (testing) set error* starts to worsen (increase).
  - So... how do we know when to stop training our model to avoid overfitting?



Final Output Nodes

- In general, the complexity of your network should match the complexity of your problem. The final output nodes should be related to what kind of problem you are solving





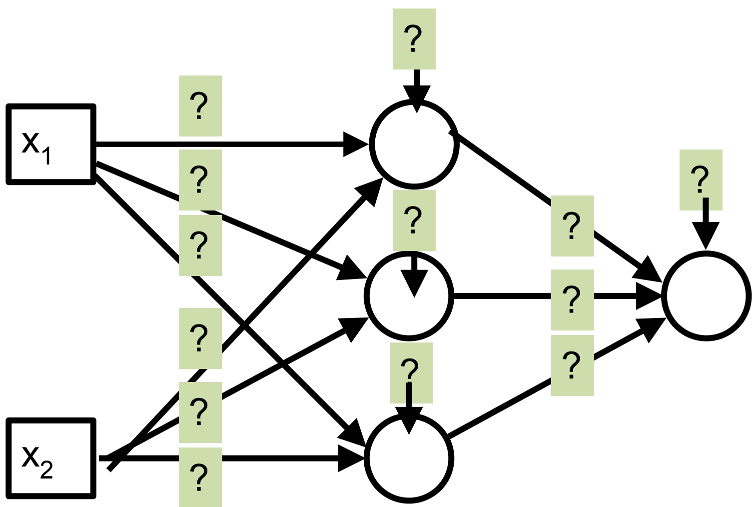
Activation Function	Function	Lower bound	Upper bound	Type of Machine Learning
Linear	$f(z) = az$	$-\infty$	∞	regression where results can be negative
Rectified Linear Unit (ReLU)	$relu(z) = max(0, z)$	0	∞	regression where results can't be negative
Sigmoid	$sigmoid(z) = \frac{1}{1+e^{-z}}$	0	1	binary classification
Softmax	$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$	0	1	multiclass classification

Today’s Agenda

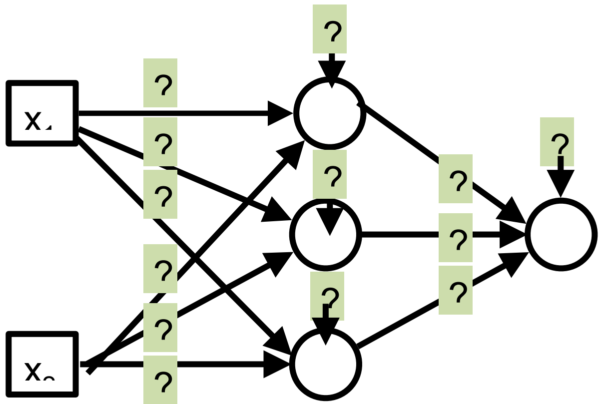
- Biological Inspiration to Connect Neurons
- Multilayer Perceptrons (MLP)
- MLP Structure
- Learning MLP Weight Parameters

Training to Learn MLP (Network) Structure Parameters

- The trainable parameters are the *weights (w's)* which are learned from the training data



Training to Learn MLP (Network) Structure Parameters



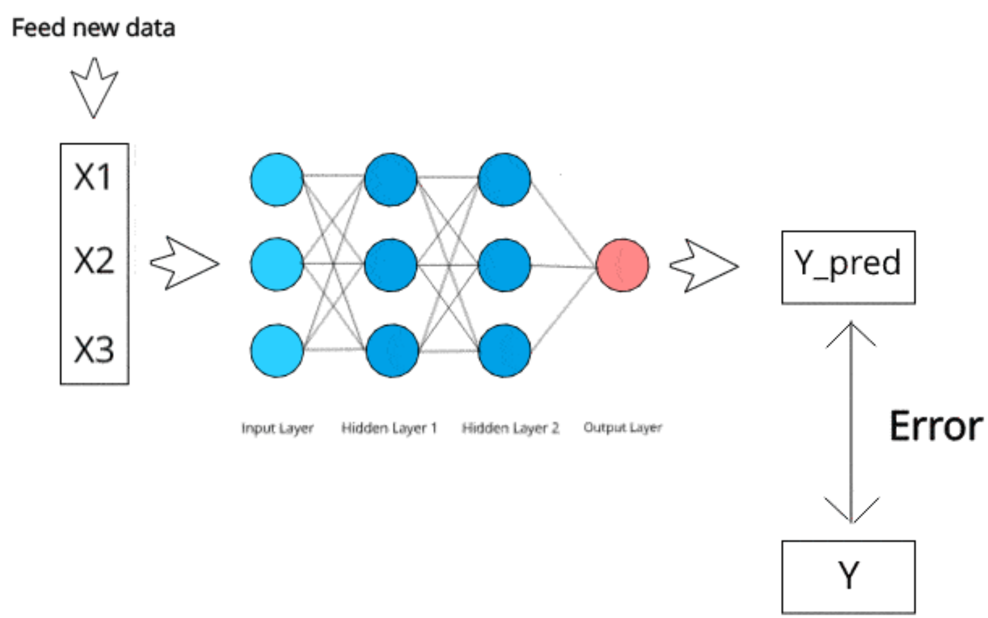
- The goal is to **minimize the error** predicted by the network (from last lecture) from the training data
  - Gradient Descent
  - Stochastic Gradient descent

- Gradient Descent
  - calculate the gradient vector based on that batch $\nabla E(\mathbf{w})$
  - adjust (or update) the values of the weights based on the gradient vector to that batch

$$\mathbf{w}^{\mathbf{new}} = \mathbf{w}^{\mathbf{old}} - \eta \nabla \mathbf{E}(\mathbf{w})$$

Training to Learn MLP (Network) Structure Parameters

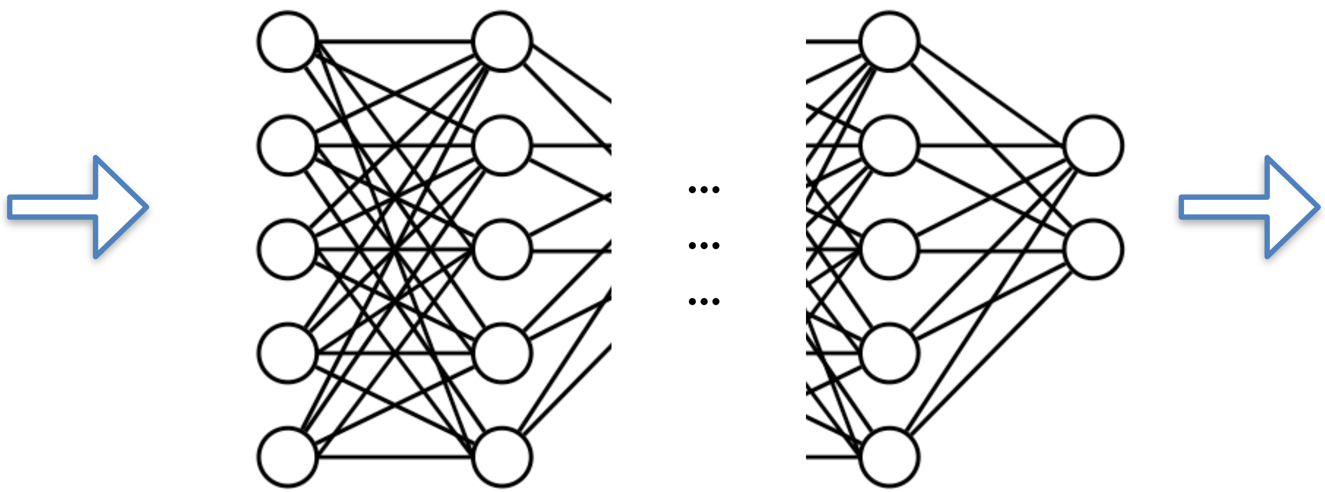
- The specific name for the weight learning algorithm is Backpropagation. It is glorified name but it is gradient descent under the hood.
- It tunes **the weights** over a neural network using **gradient descent** to iteratively reduce the error in the network.



[Image reference]()

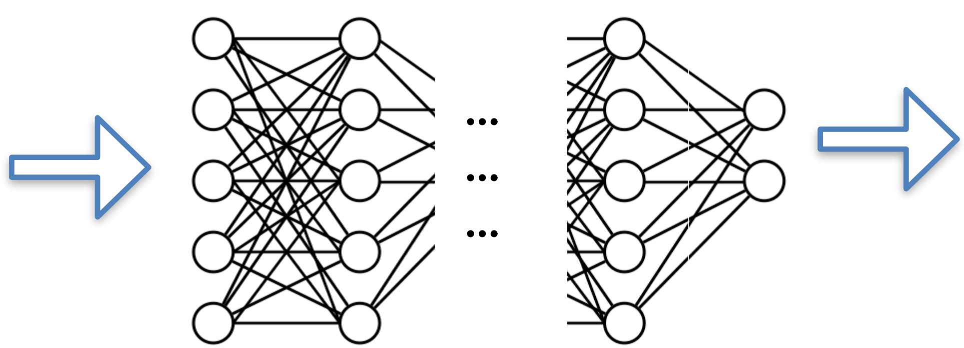
Neural Networks as Universal Function Approximators

- MLPs, neural networks in general, are *universal function approximators*
  - given any function, and a complicated enough network, they can accurately model that function



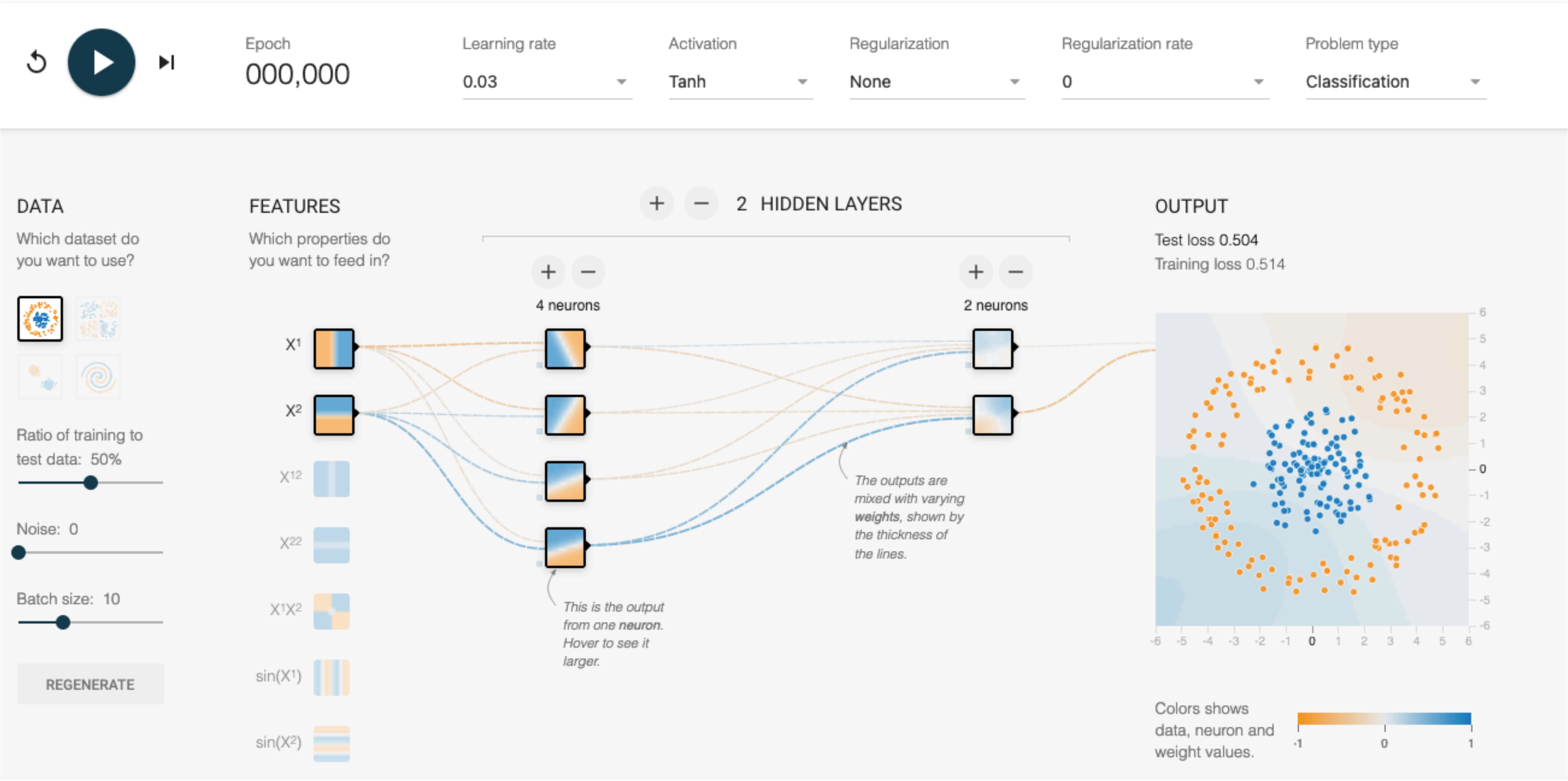
MLP Summary

- MLPs are effective in finding non-linear patterns in the training data
  - can be applied to **regression** or **classification**.
  - **backpropagation** tunes the weights over a neural network using **gradient descent** to iteratively reduce the error in the network
  - **overfitting** the training data is common and is important to avoid
  - the following parameters should be tuned when using MLPs:
    - number of epochs
    - structure of the network (depth, width)
    - activation function
    - eta (learning rate)



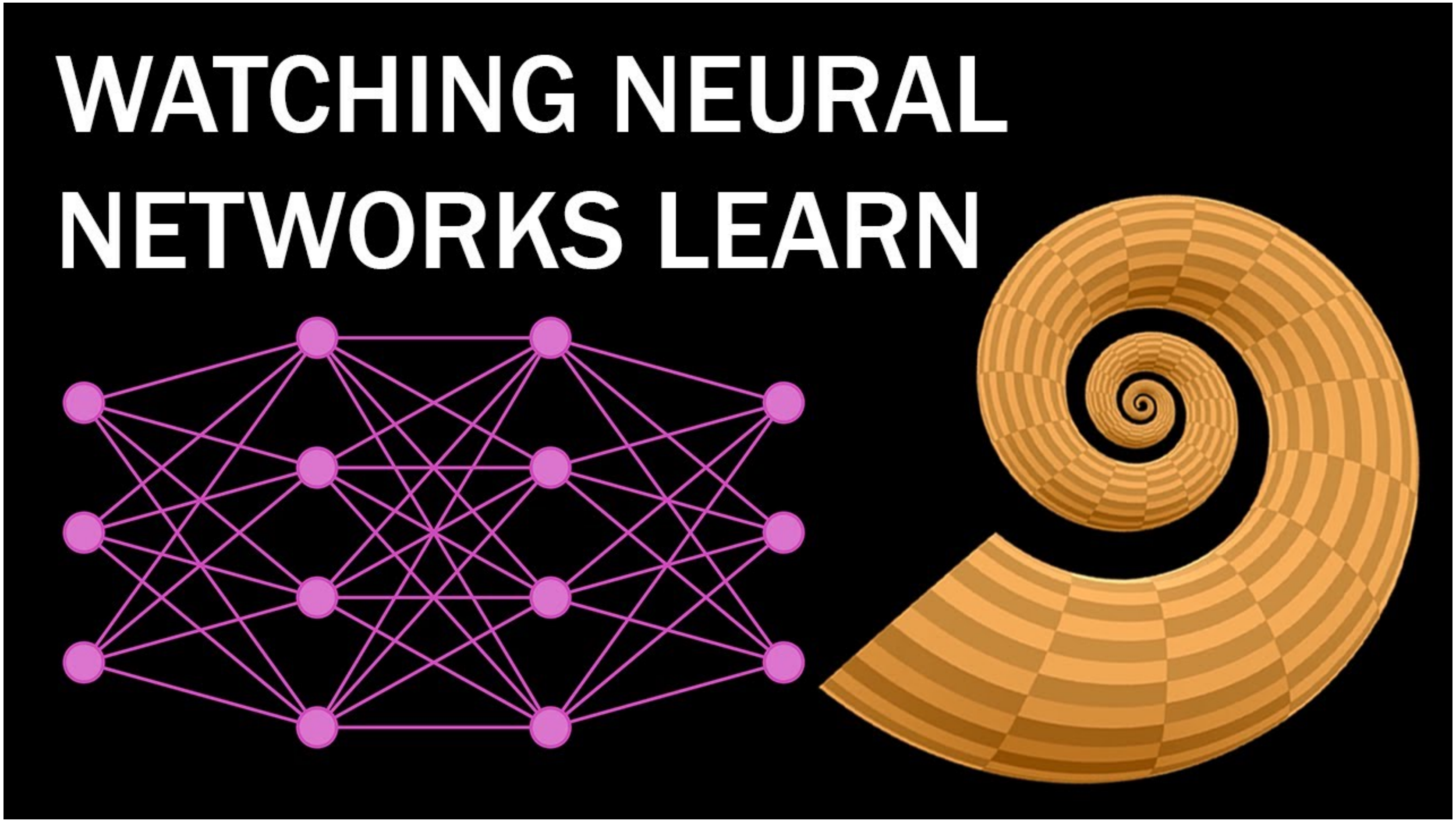
Tinker with the Following to See MLP in Action

- MLPs are effective in finding non-linear patterns in the training data



https://playground.tensorflow.org

Neural Networks as Universal Function Approximators



Reference: https://www.youtube.com/watch?v=TkwXa7Cvfr8&source_ve_path=MjM4NTE&feature=emb_title

Today's Agenda

- Biological Inspiration to Connect Neurons
- Multilayer Perceptrons (MLP)
- MLP Structure
- Learning MLP Weight Parameters
- PyTorch Basics

PyTorch

- PyTorch is machine learning framework based on Torch library. It has a Python interface.

- This is a very popular framework for building and deploying deep learning application including MLP, and other future models we will learn about in this course

- Colab and Kaggle both has PyTorch support hence we can readily run our PyTorch code without worrying about the installation. But optionally, if you have GPU in your workstation (laptop/desktop), you can install a fresh copy of PyTorch there.

https://pytorch.org/