CS167: Machine Learning

PyTorch Basics
A Simple Implementation of Multilayer Perceptron (MLP) with PyTorch

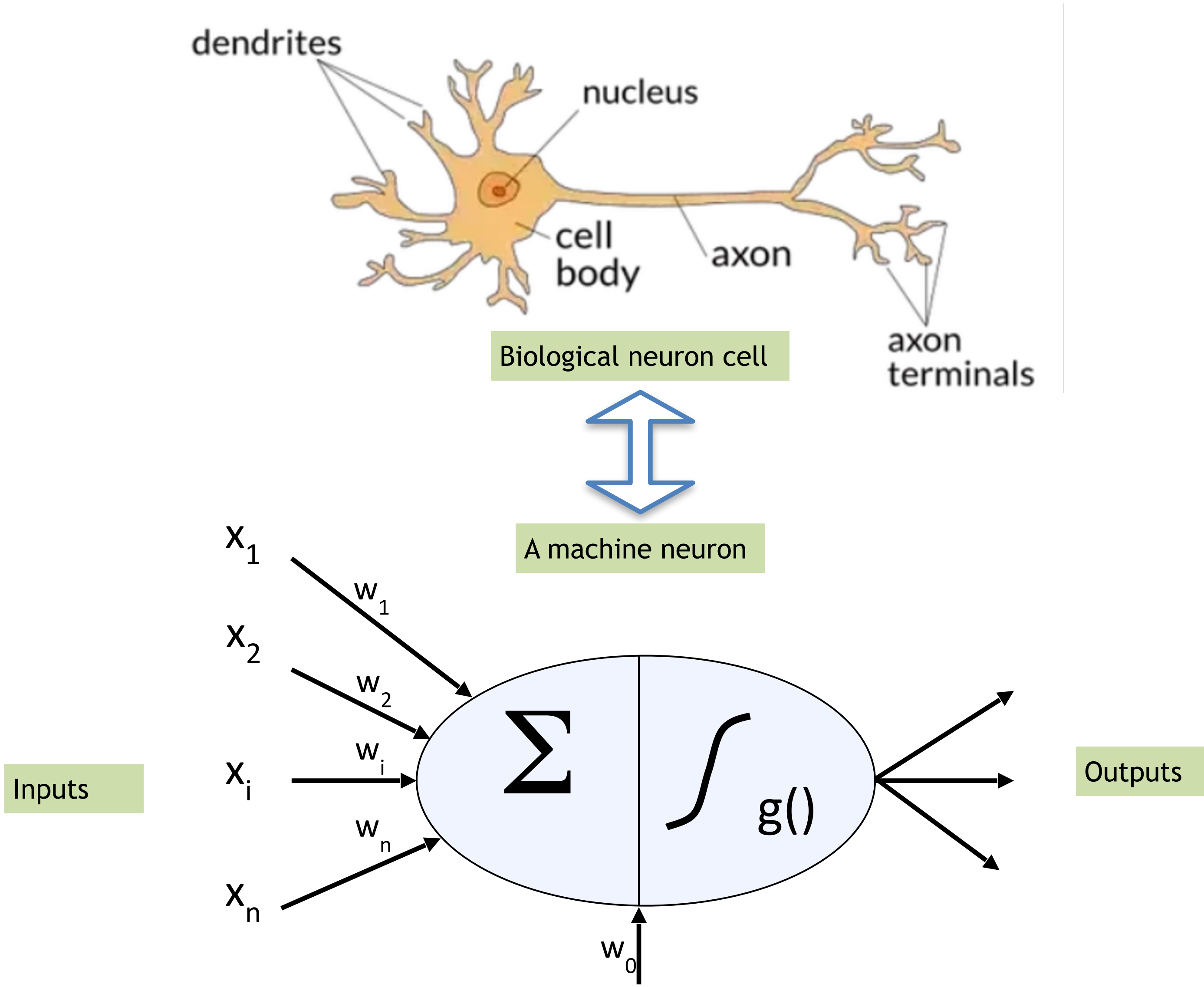
Tuesday, April 8th, 2024

Drake UNIVERSITY

Recap

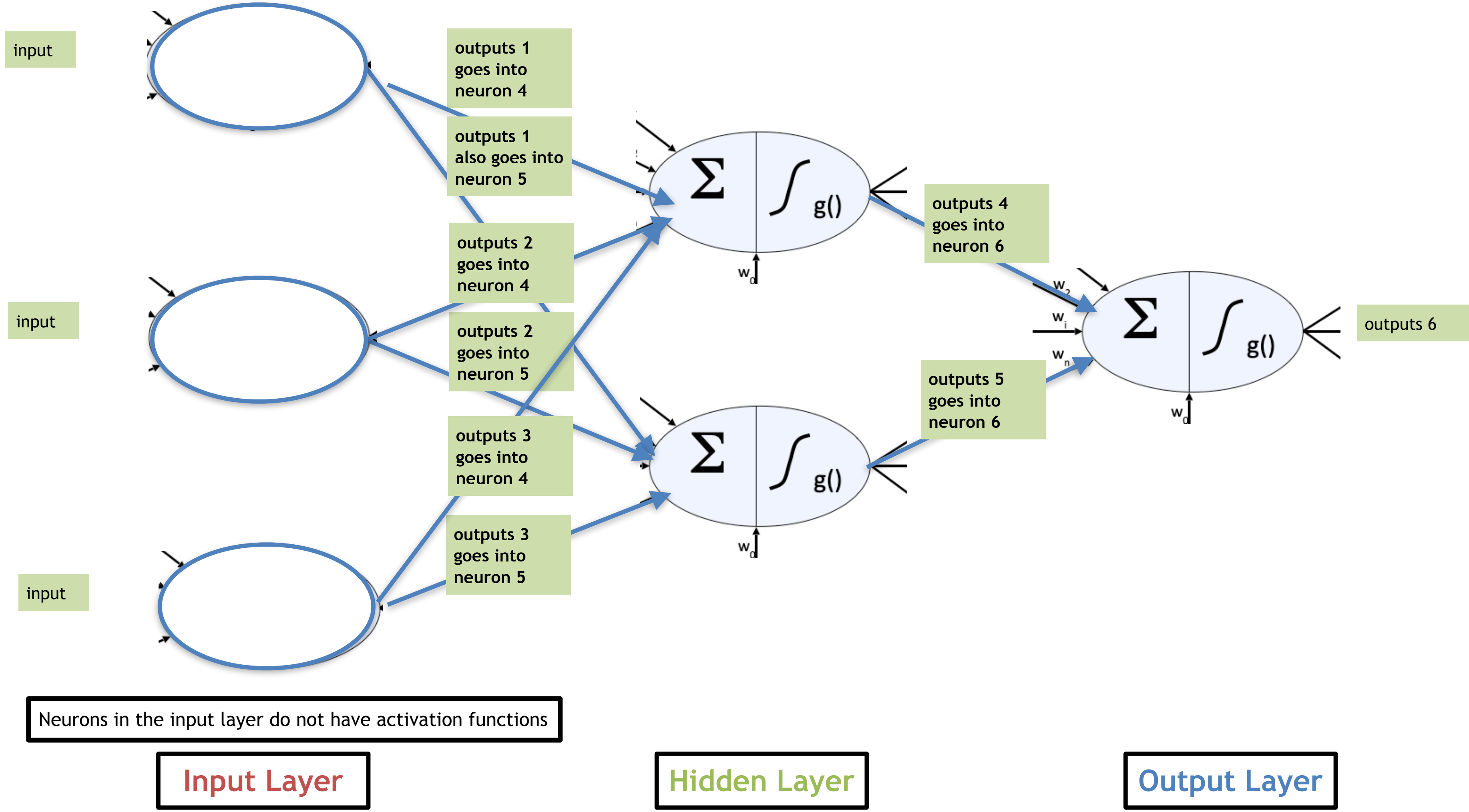
- Biological Inspiration to Connect Neurons
- Multilayer Perceptrons (MLP)
- MLP Structure
- Learning MLP Weight Parameters

Recap: Inspiration from Neuron Cells



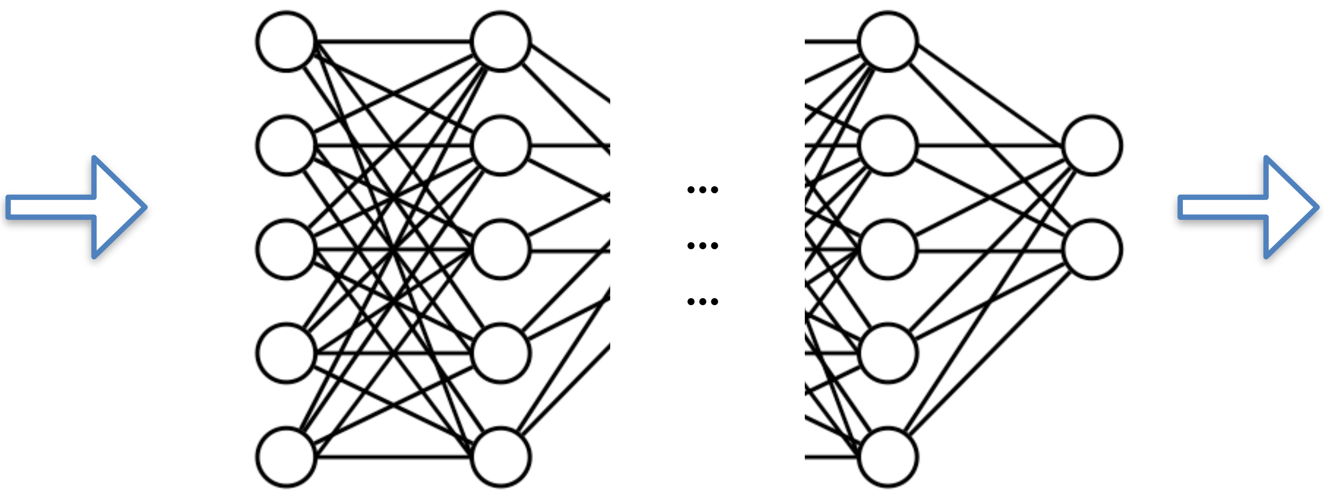
Recap: 1-Hidden Layer Neural Network

- 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



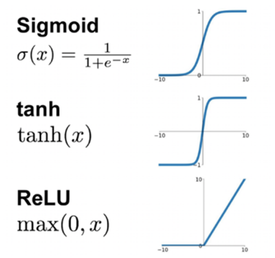
Recap: MLP (Network) Structure

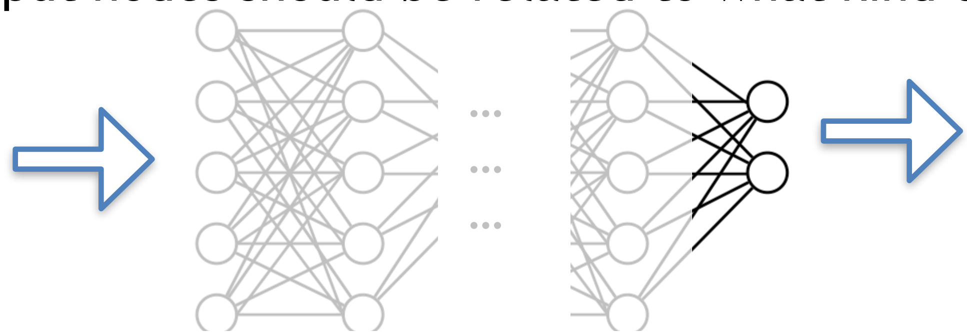
- Each of these questions need to be answered before you set up your neural network:
  - how many hidden layers should I have? (depth)
  - how many neurons should be in each layer? (width)
  - what should your activation be at each of the layers?



Recap: 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

Recap: 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.

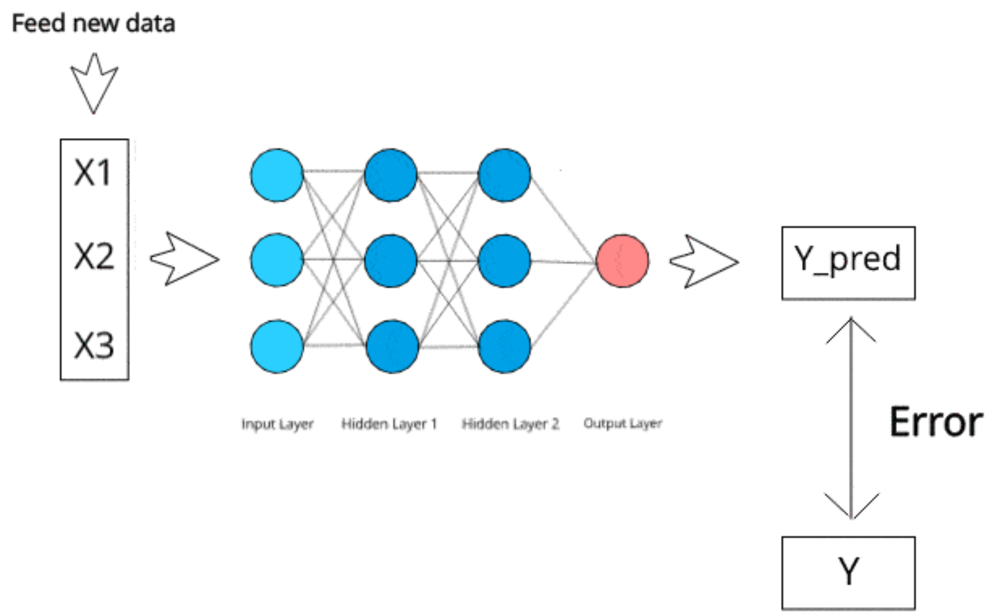


Image reference

Today's Agenda

- PyTorch Basics
- Simple Multilayer Perceptrons (MLP) Implementation using PyTorch

PyTorch

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

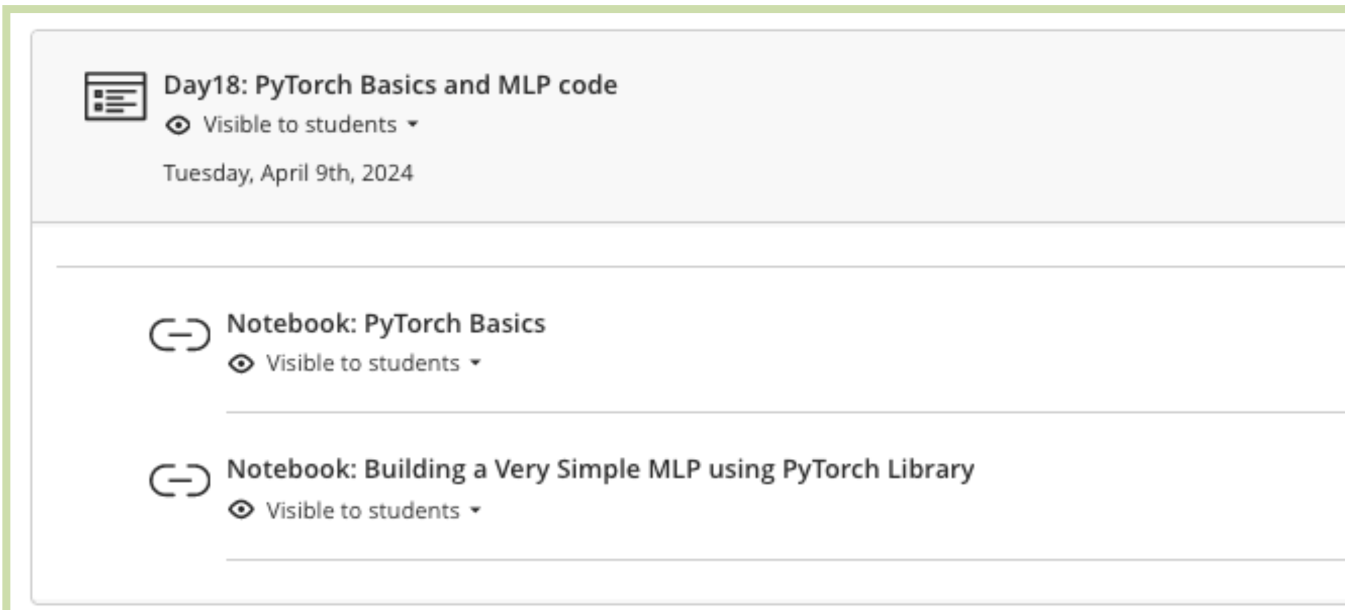
- This is a very popular framework for building and deploying deep learning application including MLP, and other future models we will learn about in this course

- 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/

PyTorch

- Go to Blackboard and work on the notebook titled "PyTorch Basics."



https://pytorch.org/

PyTorch

- Upload your notebook to Blackboard (under 'Assignment' section) once completed!

Notebook 0 : Onboarding
Due date: 2/1/24, 11:59 PM
Visible to students ▾

In-class activity#8 - PyTorch basics
Due date: 4/9/24, 11:59 PM
Visible to students ▾

upload your notebook

In-class activity#7 (Stochastic Gradient Descent - SGD)
Due date: 4/2/24, 11:59 PM
Visible to students ▾

Complete the group activity from class today and upload your notebook. Here is the reference notebook: https://github.com/alimoorreza/CS167-sp24-notes/blob/main/Day16_Stochastic_Gradient_Descent_SGD.ipynb

https://pytorch.org/

Today’s Agenda

- PyTorch Basics
- Simple Multilayer Perceptrons (MLP) Implementation using PyTorch

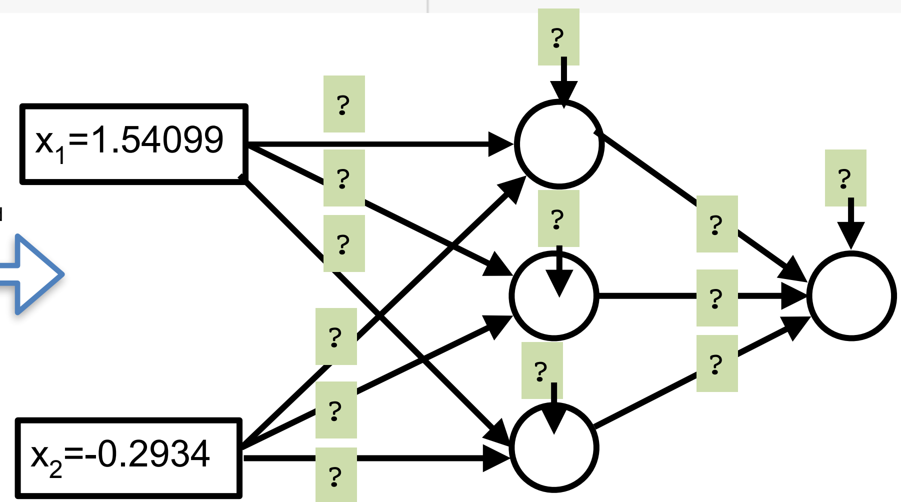
Generate Random Samples for the MLP Below

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

```
# let's generate 4 random samples of (x1, x2) for the above network
torch.manual_seed(0)
random_X = torch.randn(4,2) # you could imagine that these are pairs of (x1, x2) as shown in the above table
print('random_X = \n', random_X.numpy())
input_feature_size = random_X.shape[1] # number of columns corresponds to feature dimension
print('\n\ninput feature dimension: ', input_feature_size)
```

Sample#	x_1	x_2
1	1.5409961	-0.2934289
2	-2.1787894	0.56843126
3	-1.0845224	-1.3985955
4	0.40334684	0.83802634

Consider doing forward pass for this example



Important Design Questions for MLP

- Each of these questions need to be answered before you set up your **multilayer perceptron**
  - Q1: how many hidden layers should be there? (depth)
  - Q2: how many neurons should be in each layer? (width)
  - Q3: how many dense connections should be there in between each adjacent layers
  - Q4: what should the activation be at each of the intermediate layers?
    - `sigmoid(), tanh(), rectified-linear-unit(), etc`
  - Q5: what should be activation of the final layer
    - `depends the task` *`classification`* `(sigmoid(), softmax()) vs.` *`regression`*

Important Design Questions for MLP

```python
torch.manual_seed(1) # for reproducibility
# Q1: how many hidden layers should be there? (depth)
# answer: there is only 1 hidden layer
num_of_hidden_layer = 1

# Q2: how many neurons should be in each layer? (width)
# answer: there are 2 neurons in the input  layer
#         there are 3 neurons in the hidden layer
#         there are 1 neurons in the output layer
num_of_neurons_input_layer  = 2
#num_of_neurons_input_layer  = input_feature_size # also can be assigned from 'input_feature_size' (which we computed in the previous cell
num_of_neurons_hidden_layer = 3
num_of_neurons_output_layer = 1

# Q3 how many dense connections should be there in between each adjacent layers
# answer: there should be 2x3 dense connnections (between input  layer and hidden layer: dense_connections_W1)
#         there should be 3x1 dense connnections (between hidden layer and output layer: dense_connections_W2)
dense_connections_W1 = torch.randn(num_of_neurons_input_layer,  num_of_neurons_hidden_layer)
dense_connections_W2 = torch.randn(num_of_neurons_hidden_layer, num_of_neurons_output_layer)
print('Random initialized weights between input  layer and hidden layer: dense_connections_W1=\n', dense_connections_W1.numpy())
print('Random initialized weights between input  layer and hidden layer: dense_connections_W2=\n', dense_connections_W2.numpy())
# add the bias terms for all the layers except input layer
bias_terms_hidden    = torch.randn(num_of_neurons_hidden_layer)
bias_terms_output    = torch.randn(num_of_neurons_output_layer)
print('bias_terms_hidden:\n', bias_terms_hidden.numpy())
print('bias_terms_output:\n', bias_terms_output.numpy())
```

```
Random initialized weights between input  layer and hidden layer: dense_connections_W1=
 [[ 0.66135216  0.2669241   0.06167726]
 [ 0.6213173  -0.45190597 -0.16613023]]
Random initialized weights between input  layer and hidden layer: dense_connections_W2=
 [[-1.5227685 ]
 [ 0.38168392]
 [-1.0276086 ]]
bias_terms_hidden:
 [-0.5630528  -0.89229053 -0.05825018]
bias_terms_output:
 [-0.19550958]
```

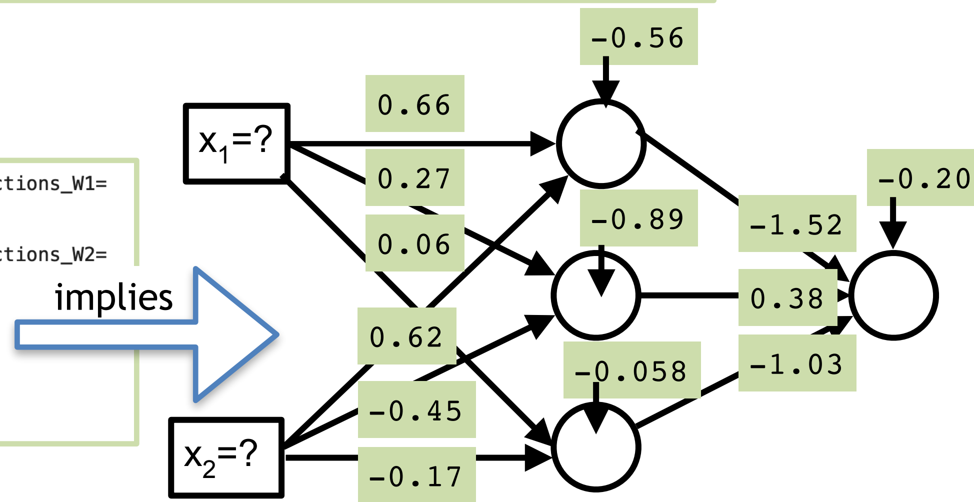
Important Design Questions for MLP

```python
torch.manual_seed(1) # for reproducibility
# Q1: how many hidden layers should be there? (depth)
# answer: there is only 1 hidden layer
num_of_hidden_layer = 1

# Q2: how many neurons should be in each layer? (width)
# answer: there are 2 neurons in the input  layer
#         there are 3 neurons in the hidden layer
#         there are 1 neurons in the output layer
num_of_neurons_input_layer  = 2
#num_of_neurons_input_layer  = input_feature_size # also can be assigned from 'input_feature_size' (which we computed in the previous cell
num_of_neurons_hidden_layer = 3
num_of_neurons_output_layer = 1

# Q3 how many dense connections should be there in between each adjacent layers
# answer: there should be 2x3 dense connnections (between input  layer and hidden layer: dense_connections_W1)
#         there should be 3x1 dense connnections (between hidden layer and output layer: dense_connections_W2)
dense_connections_W1 = torch.randn(num_of_neurons_input_layer,  num_of_neurons_hidden_layer)
dense_connections_W2 = torch.randn(num_of_neurons_hidden_layer, num_of_neurons_output_layer)
print('Random initialized weights between input  layer and hidden layer: dense_connections_W1=\n', dense_connections_W1.numpy())
print('Random initialized weights between input  layer and hidden layer: dense_connections_W2=\n', dense_connections_W2.numpy())
# add the bias terms for all the layers except input layer
bias_terms_hidden    = torch.randn(num_of_neurons_hidden_layer)
bias_terms_output    = torch.randn(num_of_neurons_output_layer)
print('bias_terms_hidden:\n', bias_terms_hidden.numpy())
print('bias_terms_output:\n', bias_terms_output.numpy())
```

```
Random initialized weights between input  layer and hidden layer: dense_connections_W1=
 [[ 0.66135216  0.2669241   0.06167726]
 [ 0.6213173  -0.45190597 -0.16613023]]
Random initialized weights between input  layer and hidden layer: dense_connections_W2=
 [[-1.5227685 ]
 [ 0.38168392]
 [-1.0276086 ]]
bias_terms_hidden:
 [-0.5630528  -0.89229053 -0.05825018]
bias_terms_output:
 [-0.19550958]
```



Important Design Questions for MLP

```
torch.manual_seed(1) # for reproducibility
# Q1: how many hidden layers should be there? (depth)
# answer: there is only 1 hidden layer
num_of_hidden_layer = 1

# Q2: how many neurons should be in each layer? (width)
# answer: there are 2 neurons in the input  layer
#         there are 3 neurons in the hidden layer
#         there are 1 neurons in the output layer
num_of_neurons_input_layer  = 2
#num_of_neurons_input_layer  = input_feature_size # also can be assigned from 'input_feature_size' (which we computed in the previous cell
num_of_neurons_hidden_layer = 3
num_of_neurons_output_layer = 1

# Q3 how many dense connections should be there in between each adjacent layers
# answer: there should be 2x3 dense connnections (between input  layer and hidden layer: dense_connections_W1)
#         there should be 3x1 dense connnections (between hidden layer and output layer: dense_connections_W2)
dense_connections_W1 = torch.randn(num_of_neurons_input_layer,  num_of_neurons_hidden_layer)
dense_connections_W2 = torch.randn(num_of_neurons_hidden_layer, num_of_neurons_output_layer)
```

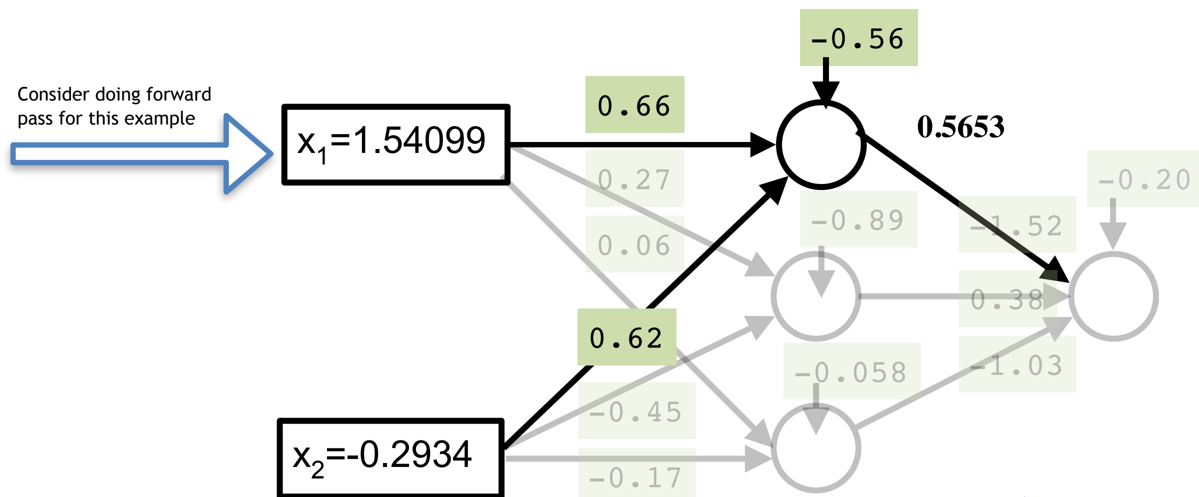
```
[21] # Q4: what should the activation be at each of the intermediate layers?
     # answer: let use sigmoid() activation function in the hidden layer
     sigmoid_activation_hidden = nn.Sigmoid()
```

```
[22] # Q5: what should be activation of the final layer (let's assume we are using a binary classification task for which sigmoid ctivation is
     sigmoid_activation_output = nn.Sigmoid()
```

Forward Pass in our Multilayer Perceptron (MLP)

- 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
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

Sample#	x_1	x_2
1	1.5409961	-0.2934289



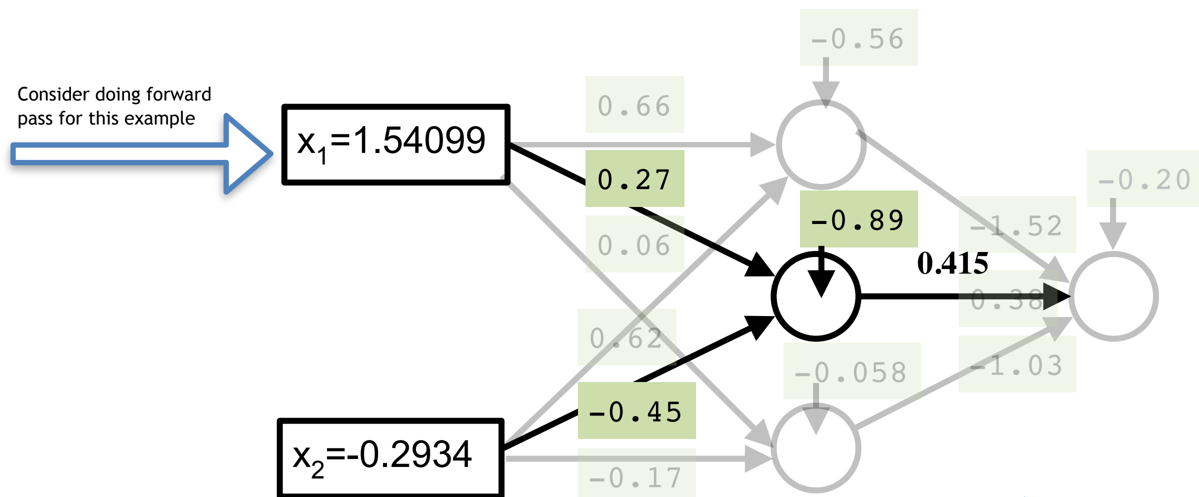
$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = [w_0 \quad w_1 \quad w_2]\begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = [-0.56 \quad 0.66 \quad 0.62]\begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = (-0.56) + 1.54 * 0.66 + (-0.293) * 0.66 \quad = 0.263$$

$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w}^{\mathbf{T}}\mathbf{x}}} = \frac{1}{1 + exp^{-0.263}} = 0.5653$$

Forward Pass in our Multilayer Perceptron (MLP)

- 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
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

Sample#	x_1	x_2
1	1.5409961	-0.2934289



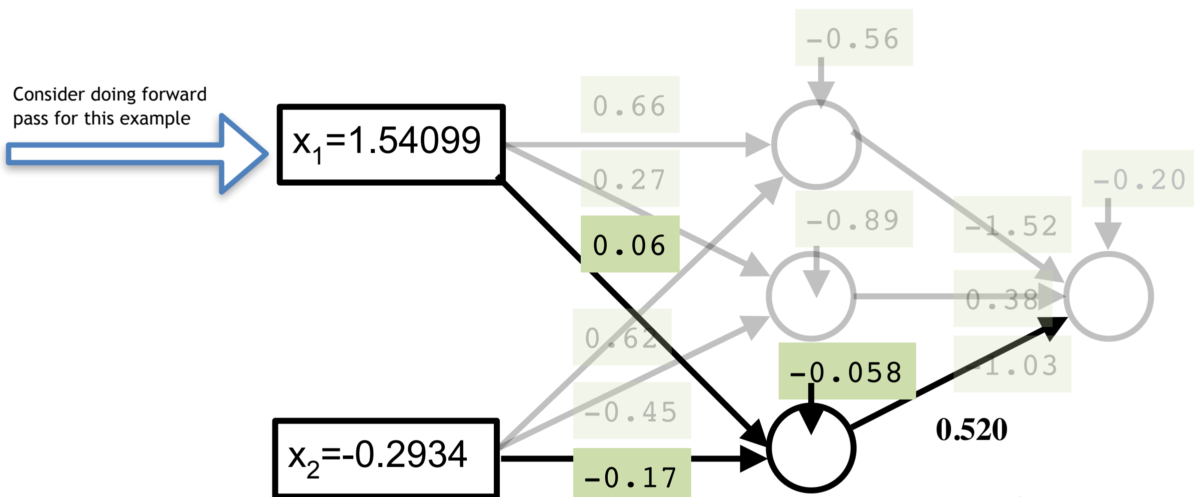
$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = [w_0 \quad w_1 \quad w_2]\begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = [-0.89 \quad 0.27 \quad -0.45]\begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = (-0.89) + 1.54 * 0.27 + (-0.293) * (-0.45) \quad = -0.34$$

$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w}^{\mathbf{T}}\mathbf{x}}} = \frac{1}{1 + exp^{-(-0.34)}} = 0.415$$

Forward Pass in our Multilayer Perceptron (MLP)

- 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
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

Sample#	x_1	x_2
1	1.5409961	-0.2934289



$$\mathbf{w^T x} = [w_0 \quad w_1 \quad w_2] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = [-0.058 \quad 0.06 \quad -0.17] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = (-0.058) + 1.54 * 0.06 + (-0.293) * (-0.17) = 0.084$$

$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w^T x}}} = \frac{1}{1 + exp^{-0.084}} = 0.520$$

Forward Pass in our Multilayer Perceptron (MLP)

- 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
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

```
matrix_mult_X_and_W1 = torch.matmul(random_X[0,:], dense_connections_W1) + bias_terms_hidden
print('hidden layer input vector and weight vector dot products: \n', matrix_mult_X_and_W1.numpy())
output_hidden_layer = sigmoid_activation_hidden(matrix_mult_X_and_W1)
print('output of hidden layer: \n', output_hidden_layer.numpy())
```

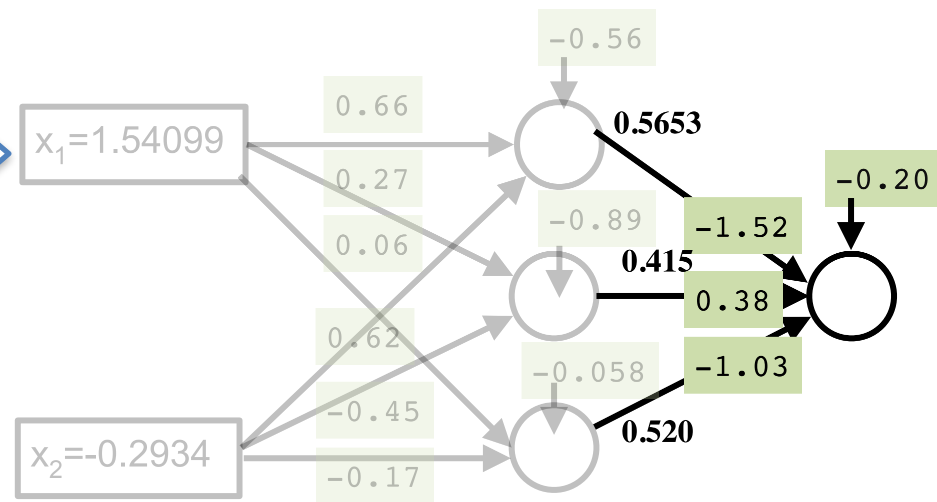
```
hidden layer input vector and weight vector dot products:
 [ 0.27377588 -0.3483593   0.08554165]
output of hidden layer:
 [0.5680196  0.41378036 0.5213724 ]
```

Forward Pass in our Multilayer Perceptron (MLP)

- 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
- We can collective do all these dot products in a single layer using a single matrix-matrix multiplication torch.matmul() as follows.
- Also add the bias-term after computing the matrix multiplication

Sample#	x_1	x_2
1	1.5409961	-0.2934289

Consider doing forward pass for this example



$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = [w_0 \quad w_1 \quad w_2 \quad w_3]\begin{bmatrix}-0.20\\0.5653\\0.415\\0.520\end{bmatrix} = [1 \; -1.52 \quad 0.38 \quad -1.03]\begin{bmatrix}-0.20\\0.5653\\0.415\\0.520\end{bmatrix} = 1*(-0.20) + (-1.52)*0.5653 + 0.38*0.415 + (-1.03)*(0.520)$$

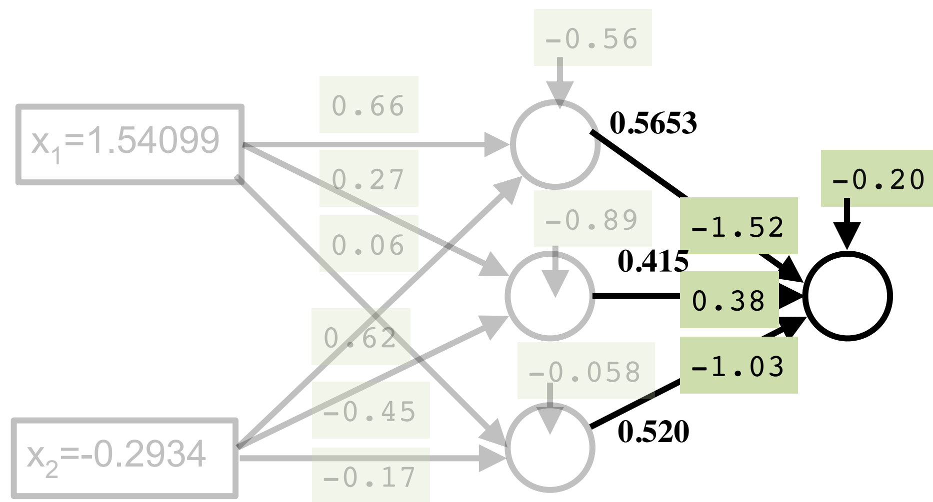
$$= -1.437156$$

$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w}^{\mathbf{T}}\mathbf{x}}} = \frac{1}{1 + exp^{-(-1.437156)}} = 0.191$$

Forward Pass in our Multilayer Perceptron (MLP)

```
matrix_mult_hidden_and_W2 = torch.matmul(output_hidden_layer, dense_connections_W2) + bias_terms_output
print('output of output layer: \n', matrix_mult_hidden_and_W2)
final_output = sigmoid_activation_output(matrix_mult_hidden_and_W2)
print('output of hidden layer: \n', final_output.numpy())
```

```
output of output layer:
 tensor([-1.4383])
output of hidden layer:
 [0.1918079]
```



$$\mathbf{w^T x} = [w_0 \quad w_1 \quad w_2 \quad w_3] \begin{bmatrix} -0.20 \\ 0.5653 \\ 0.415 \\ 0.520 \end{bmatrix} = [1 - 1.52 \quad 0.38 \quad -1.03] \begin{bmatrix} -0.20 \\ 0.5653 \\ 0.415 \\ 0.520 \end{bmatrix} = 1 * (-0.20) + (-1.52) * 0.5653 + 0.38 * 0.415 + (-1.03) * (0.520)$$

$$\text{output} = \frac{1}{1 + exp^{-\mathbf{w^T x}}} = \frac{1}{1 + exp^{-(-1.437156)}} = 0.191$$

Next lecture: Modular Code Multilayer Perceptron using MLP

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