CS167: Machine Learning

Modular Implementation of Multilayer Perceptron (MLP) with PyTorch

Wednesday, November 13th, 2024

Drake University

Recap: 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*

Recap: 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]
```

Recap: Important Design Questions for MLP

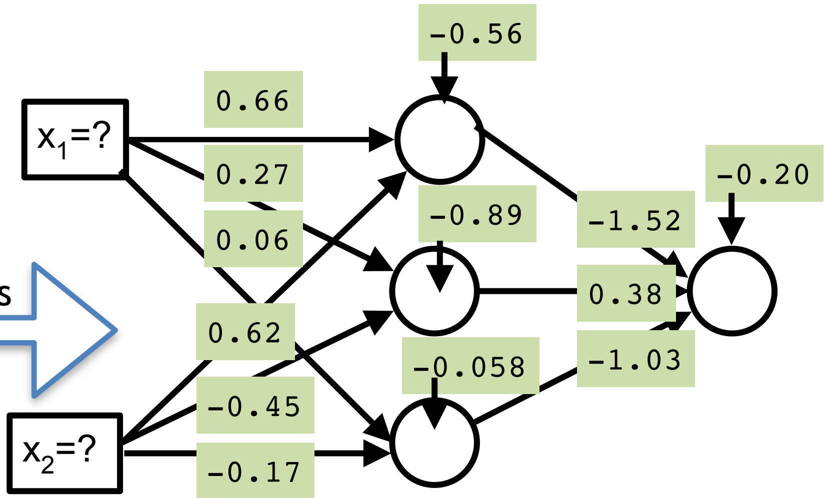
```python
torch.manual_seed(1) # for reproducibility
# Q1: how many hidden layers should be there? (depth)
# answer: there is only 1 hidden layer
num_of_hidden_layer = 1

# 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]
```

implies



Recap: 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)
```

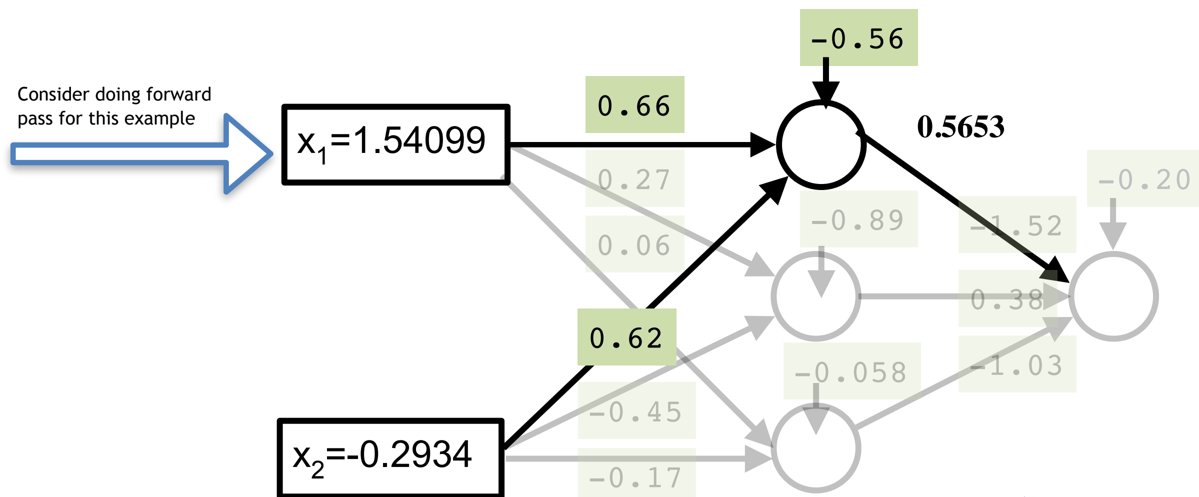
```
[21] # Q4: what should the activation be at each of the intermediate layers?
     # answer: let use sigmoid() activation function in the hidden layer
     sigmoid_activation_hidden = nn.Sigmoid()
```

```
[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()
```

Recap: 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



W0 W1 W2 ; X0 X1 X2

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

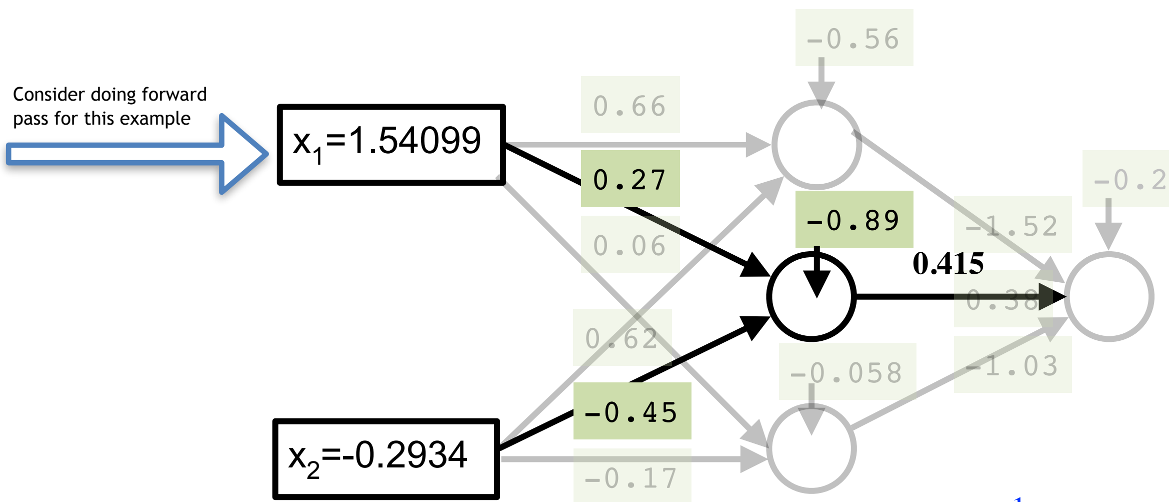
$$\text{output} = \frac{1}{1+exp^{-\mathbf{w^T x}}} = \frac{1}{1+exp^{-0.263}} = 0.5653$$

X_0 will always be 1.0

Recap: 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



$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = [w_0 \quad w_1 \quad w_2] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} \begin{matrix} x_0 \\ x_1 \\ x_2 \end{matrix} = [-0.89 \quad 0.27 \quad -0.45] \begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = (-0.89) + 1.54 * 0.27 + (-0.293) * (-0.45) \quad = -0.34$$

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

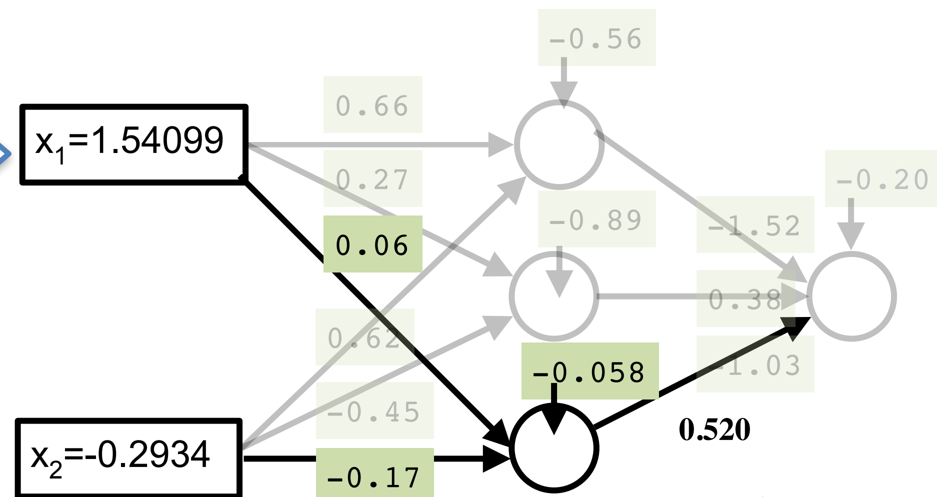
x_0 will always be 1.0

Recap: 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



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

$$\mathbf{w}^{\mathbf{T}}\mathbf{x} = [w_0 \quad w_1 \quad w_2]\begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix}\begin{matrix} x_0 \\ x_1 \\ x_2 \end{matrix} = [-0.058 \quad 0.06 \quad -0.17]\begin{bmatrix} 1 \\ 1.54 \\ -0.293 \end{bmatrix} = (-0.058) + 1.54 * 0.06 + (-0.293) * (-0.17) = 0.084$$

x_0 will always be 1.0

Recap: 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())
```

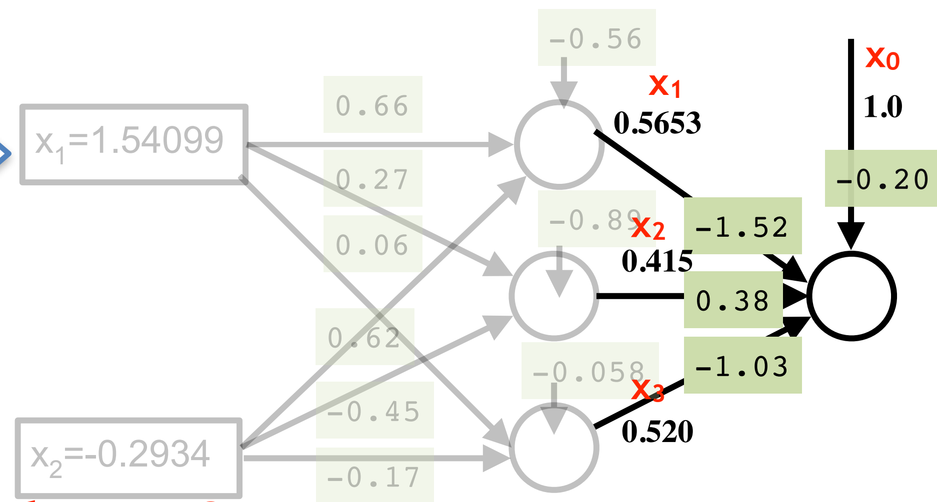
```
hidden layer input vector and weight vector dot products:
 [ 0.27377588 -0.3483593   0.08554165]
output of hidden layer:
 [0.5680196  0.41378036 0.5213724 ]
```

Recap: 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



x_0 will always be 1.0

$$\mathbf{w}^T\mathbf{x} = [w_0 \quad w_1 \quad w_2 \quad w_3] \begin{bmatrix} 1 \\ 0.5653 \\ 0.415 \\ 0.520 \end{bmatrix} \begin{matrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{matrix} = [-0.20 \; -1.52 \quad 0.38 \quad -1.03] \begin{bmatrix} 1 \\ 0.5653 \\ 0.415 \\ 0.520 \end{bmatrix} = (-0.20) * 1 + (-1.52) * 0.5653 + 0.38 * 0.415 + (-1.03) * (0.520)$$

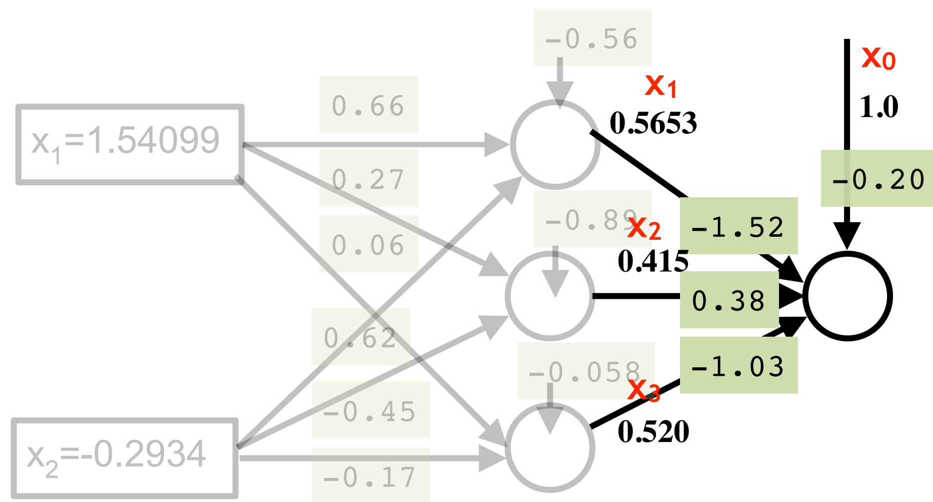
$$= -1.437156$$

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

Recap: Forward Pass in our Multilayer Perceptron (MLP)

```python
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]
```



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

$$= -1.437156$$

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

Today's Agenda

- Simple Multilayer Perceptrons (MLP) Implementation using PyTorch
  - Basic functions and utilities

so that we don't need to explicitly apply functions such as: ***torch.matmult()***

- Modular MLP Implementation using PyTorch
  - structural aspect
  - following the convention of research community

List of PyTorch Functions We Need

- nn.Linear()
  creates the dense connections between two adjacent layers (*left layer* and *right layer*)
  just provide **#neurons_left_layer** and **#neurons_right_layer**
- nn.ReLU()
- nn.Softmax()
- nn.flatten()
- nn.Sequential()

- Let's jump into the notebook for a detailed discussion
  - https://github.com/alimoorreza/CS167-fall24-notes/blob/main/Day20_MLP_with_PyTorch.ipynb

nn.Linear() function

Group Exercise#1

Create a new Linear layer with the following structure:

> The first layer has 2 input nodes and 16 output nodes.

```
[ ] # your code here
    # ...
```

Group Exercise#2

> Apply a tensor through your linear layer now.

> Change the value in torch.manual_seed(0) to something else, generate new inputs, and pass the tensor through your linear layer again.

> Observe the the output values.

```
[ ] # your code here.
    # ...
```

Activation Functions: nn.Sigmoid() nn.ReLU() etc

Group Exercise#3

> Experiment with different activation functions like sigmoid, tanh, and relu, and then pass a tensor through the linear layer you created for Group Exercises #1 and #2.

> Change the value in torch.manual_seed(2) to something else, generate new inputs, and pass the tensor through your linear layer again.

> Take a look at the output values and make sure they match what you were expecting!

Combining everything to make an MLP

Group Exercise#4

Let's create three Linear layers and connect them in sequence to build an MLP with the following structure:

> The first layer has 2 input nodes and 3 output nodes.

> The second layer takes 3 input nodes and outputs 6 nodes.

> The final layer connects 6 input nodes to 2 output nodes.

```
[ ] # your code here
    # ...
```

Group Exercise#5

> Apply a tensor through your MLP now.

```
[ ] # your code here
    # ...
```

Today's Agenda

- Simple Multilayer Perceptrons (MLP) Implementation using PyTorch
  - Basic functions and utilities

- Modular MLP Implementation using PyTorch
  - structural aspect
  - following the conventions of the research community

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. Create a network class with two methods:

- *init()*
- *forward()*

```
import torch
from torch import nn

# You can give any name to your new network, e.g., SimpleMLP.
# However, you have to mandatorily inherit from nn.Module to
# create your own network class. That way, you can access a lot of
# useful methods and attributes from the parent class nn.Module

class SimpleMLP(nn.Module):
  def __init__(self):
    super().__init__()
    # your network layer construction should take place here
    # ...
    # ...

  def forward(self, x):
    # your code for MLP forward pass should take place here
    # ...
    # ...
    return x
```

List of PyTorch Functions We Need

- nn.CrossEntropyLoss()
- torch.optim.SGD

  Training the network using loss function
  Optimizer

- Let's jump into the notebook for a detailed discussion
  - https://github.com/alimoorreza/CS167-fall24-notes/blob/main/Day20_Building_Modular_MLP_with_PyTorch.ipynb