Single Layer FeedForward Network
PyTorch Module- nn.Linear

PyTorch - nn.Linear

- PyTorch - nn.Linear
- One of the fundamental components of PyTorch is nn.Linear, a module that applies a linear transformation to the incoming data.
- nn.Linear is a linear layer used in neural networks that applies a linear transformation to input data using weights and biases.
- nn.Linear(n,m) is a module that creates single layer feed forward network with n inputs and m output.
- Mathematically, this module is designed to calculate the linear equation Ax+b=y where x is input, y is output, A is weight.
- This is the reason for the name 'Linear'
- A feed-forward neural network is the simplest type of artificial neural network where the connections between the perceptrons do not form a cycle.
- To use nn.Linear module, import torch

Characteristics of a feed-forward neural network:

- Perceptrons are arranged in layers. The first layer takes in the input and the last layer gives the output. The middle layers are termed as hidden layers as they remain hidden from the external world.
- Each perceptron in a layer is connected to each and every perceptron of the next layer. This is the reason for information flowing constantly from a layer to the next layer and hence the name feed-forward neural network.
- There is no connection between the perceptrons of the same layer.
- There is no backward connection (called a feedback connection) from the current layer to the previous layer.

PyTorch - nn.Linear

- The nn.Linear module takes two parameters: in_features and out_features
- Representing the number of input and output features.
- When an nn.Linear object is created, it randomly initializes a weight matrix and a bias vector.
- The size of the weight matrix is out_features x in_features, and the size of the bias vector is out_features
- create an instance of nn.Linear with three input features and one output feature. This results in a 3x1 weight matrix and a 1x1 bias vector

```
import torch
from torch import nn
 ## Creating an object for the linear class
linear_layer = nn.Linear(in_features=3, out_features=1)
```

How Does nn.Linear Work?

- nn.Linear works by performing a matrix multiplication of the input data with the weight matrix and adding the bias term.
- This operation is applied to each layer in a feed-forward neural network.
- Here, we pass a tensor of size 3 (matching the number of input features) to the linear_layer.
- The output is a tensor of size 1 (matching the number of output features), which is the result of the linear transformation.

```
import torch
from torch import nn
torch.manual_seed(42)
 ## Creating an object for the linear class
linear = nn.Linear(in_features=3, out_features=1)
print('network structure : ',linear)
print('Weight of network :\n',linear.weight)
print('Bias of network :\n',linear.bias)
## Passing input to the linear layer
output = linear(torch.tensor([1,2,3], dtype=torch.float32))
print(output)
```

How Does nn.Linear Work?

```
import torch
from torch import nn
torch.manual_seed(42)
 ## Creating an object for the linear class
linear = nn.Linear(in_features=3, out_features=1)
print('network structure : ',linear)
print('Weight of network :\n',linear.weight)
print('Bias of network :\n',linear.bias)
## Passing input to the linear layer
output = linear(torch.tensor([1,2,3],
dtype=torch.float32))
print(output)
```

```
network structure :
Linear(in_features=3,
out_features=1, bias=True)
Weight of network :
 Parameter containing:
tensor([[ 0.4414,  0.4792, -
0.1353]], requires_grad=True)
Bias of network :
 Parameter containing:
tensor([0.5304],
requires_grad=True)
tensor([1.5244],
grad_fn=<ViewBackward0>)
```

Initializing Weights and Biases

- The weights and biases in nn.Linear are parameters that the model learns during training.
- Initially, they are set to random values.
- We can view the weights and biases using the weight and bias attributes.
- Print the weight matrix and bias vector of the nn.Linear layer.

```
## To see the weights and biases
print(linear_layer.weight)
print(linear_layer.bias)
```

- PyTorch initializes these parameters randomly
- We can also set them manually or use different initialization methods.
- Ex: Use torch.nn.init module to apply specific initialization methods to the weights and biases.

Formal Definition

- A linear layer computes the linear transformation as below-
- y=xA^T+b   Where
- x   is the incoming data. It must be a tensor of dtype float32 and shape (*, in_features). Here * is any number of dimensions. in_features is number of features in the input data.
- y   is the output data after the transformation with same dtype as x   and with shape (*, out_features). Note that all dimensions except last are of the same shape as input data.
- A   is the learnable weights of shape (out_features, in_features). out_features is the last dimension of the output data.
- b   is the additional bias learned during the training.
- weights A   and biases b   are initialized randomly

Creating a FeedForwardNetwork - Syntax

- CLASS torch.nn.Linear(in_features, out_features, bias=True, device=None, dtype=None)

Parameters

- **in_features** (*int*) – size of each input sample
- **out_features** (*int*) – size of each output sample
- **bias** (*bool*) – If set to `False`, the layer will not learn an additive bias. Default: `True`

Shape:

- Input: $(*, H_{in})$ where $*$ means any number of dimensions including none and $H_{in} = \text{in\_features}$.
- Output: $(*, H_{out})$ where all but the last dimension are the same shape as the input and $H_{out} = \text{out\_features}$.

Setting parameters using ones Initialization

```
import torch
from torch import nn
net = nn.Linear(2,1)
torch.manual_seed(42)
print('network structure : torch.nn.Linear(2,1) :\n',net)
print('Weight of network :\n',net.weight)
print('Bias of network :\n',net.bias)
# Initializing the weights with the
# ones initialization method
torch.nn.init.ones_(net.weight)
torch.nn.init.ones_(net.bias)
# Displaying the initialized weights
print("newly initialized weight", net.weight)
print("newly initialized bias", net.bias)
x = torch.tensor([[1.0,1.0]])
print("input = x :\n ",x)
print('net.forward(x) :\n',net.forward(x))
y = torch.mm(x, net.weight.t()) + net.bias
print('xw + b :\n',y)
```

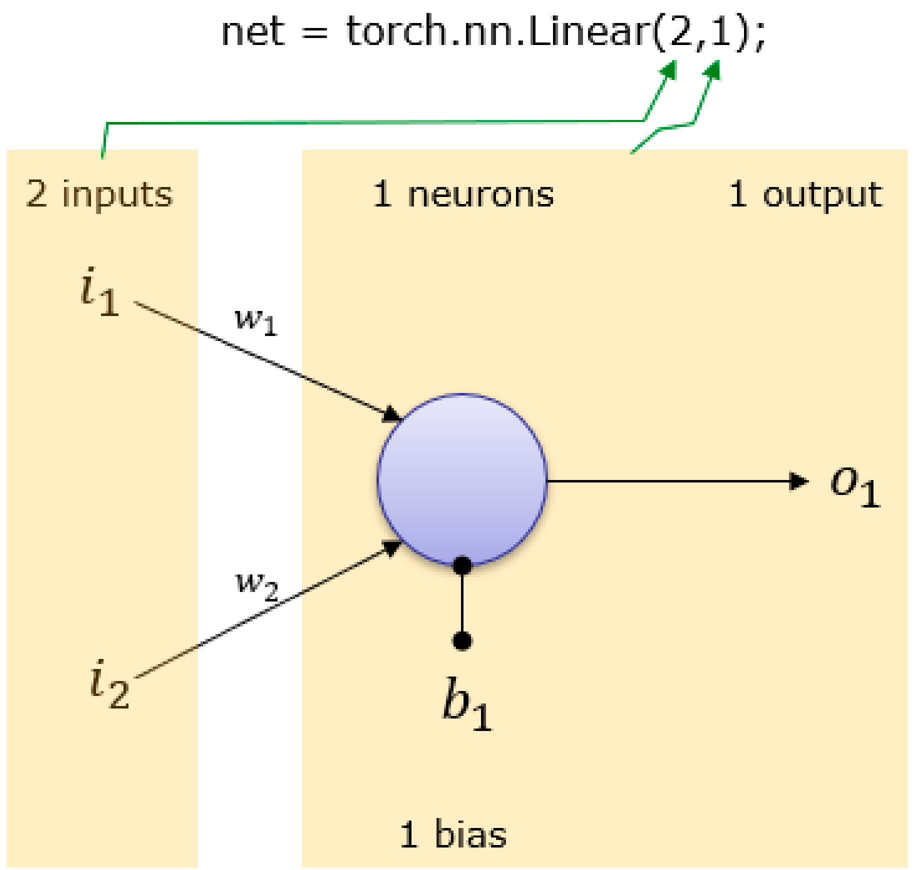
Setting parameters using ones Initialization

```
import torch
from torch import nn
net = nn.Linear(2,1)
torch.manual_seed(42)
print('network structure : torch.nn.Linear(2,1) :\n',net)
print('Weight of network :\n',net.weight)
print('Bias of network :\n',net.bias)
# Initializing the weights with the
# ones initialization method
torch.nn.init.ones_(net.weight)
torch.nn.init.ones_(net.bias)
# Displaying the initialized weights
print("newly initialized weight", net.weight)
print("newly initialized bias", net.bias)
x = torch.tensor([[1.0,1.0]])
print("input = x :\n ",x)
print('net.forward(x) :\n',net.forward(x))
y = torch.mm(x, net.weight.t()) + net.bias
print('xw + b :\n',y)
```

```
network structure : torch.nn.Linear(2,1) :
 Linear(in_features=2, out_features=1, bias=True)
Weight of network :
 Parameter containing:
tensor([[0.5406, 0.5869]], requires_grad=True)
Bias of network :
 Parameter containing:
tensor([-0.1657], requires_grad=True)
newly initialized weight Parameter containing:
tensor([[1., 1.]], requires_grad=True)
newly initialized bias Parameter containing:
tensor([1.], requires_grad=True)
input = x :
  tensor([[1., 1.]])
net.forward(x) :
 tensor([[3.]], grad_fn=<AddmmBackward0>)
w x + b :
 tensor([[3.]], grad_fn=<AddBackward0>)
```

Creating a FeedForwardNetwork – (2,1)

- 2 Inputs and 1 output
- net = torch.nn.Linear(2,1)
- This creates a network as shown
- Weight and Bias is set automatically



Creating a FeedForwardNetwork – (2,1)

```
import torch
net = torch.nn.Linear(2,1)
print('network structure : torch.nn.Linear(2,1) :\n',net)
print('Weight of network :\n',net.weight)
print('Bias of network :\n',net.bias)
```

```
network structure : torch.nn.Linear(2,1) :
 Linear(in_features=2, out_features=1, bias=True)
Weight of network :
 Parameter containing:
tensor([[0.4430, 0.6060]], requires_grad=True)
Bias of network :
 Parameter containing:
tensor([-0.4325], requires_grad=True)
```

FeedForwardNetwork – (2,1)- Evaluation

- Now creates an input vector x = torch.tensor([[1.0,1.0]])

- Evalute the network with the input vector using forward() function
- perform the linear formula Ax where A is weight matrix, x is input vector).

```
print('net.forward(x) :\n',net.forward(x))
```

- Verify the evaluation result by performing the linear formula A.x without using forward() function Here A is weight matrix, x is input vector
- This shows the same result as the one with forward() function.

```
 o = torch.mm(net.weight,x.t()) + net.bias;
print('w x + b :\n',o)
```

FeedForwardNetwork – (2,1)- Evaluation

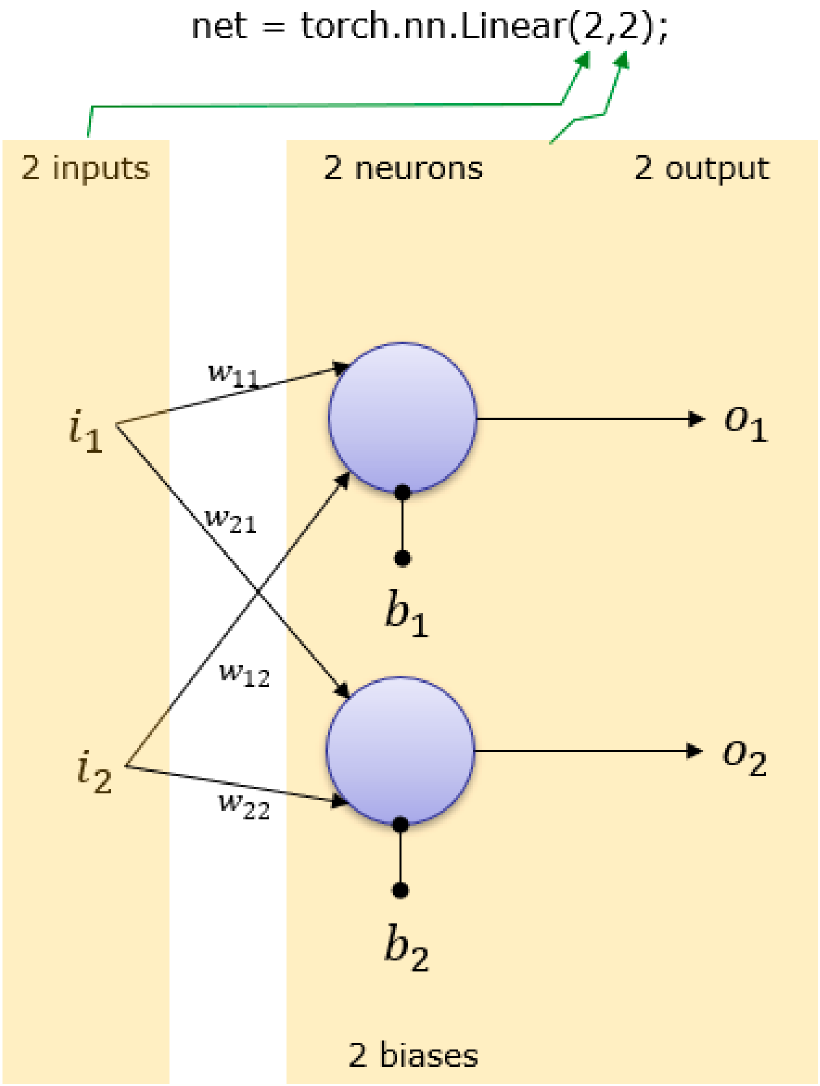
```
import torch
net = torch.nn.Linear(2,1)
print('network structure : torch.nn.Linear(2,1) :\n',net)
print('Weight of network :\n',net.weight)
print('Bias of network :\n',net.bias)
x = torch.tensor([[1.0,1.0]])
print("input = x :\n ",x)
print('net.forward(x) :\n',net.forward(x))
y = torch.mm(x, net.weight.t()) + net.bias
print(‘xw + b :\n’,y)
```

```
network structure : torch.nn.Linear(2,1) :
 Linear(in_features=2, out_features=1,
bias=True)
Weight of network :
 Parameter containing:
tensor([[0.5839, 0.0172]], requires_grad=True)
Bias of network :
 Parameter containing:
tensor([-0.5402], requires_grad=True)
input = x :
  tensor([[1., 1.]])
net.forward(x) :
 tensor([[0.0609]],
grad_fn=<AddmmBackward0>)
w x + b :
 tensor([[0.0609]], grad_fn=<AddBackward0>)
```

Creating a FeedForwardNetwork – (2,2)

2 Inputs and 2 outputs net = torch.nn.Linear(2,2)

- This creates a network as shown
- Weight and Bias is set automatically



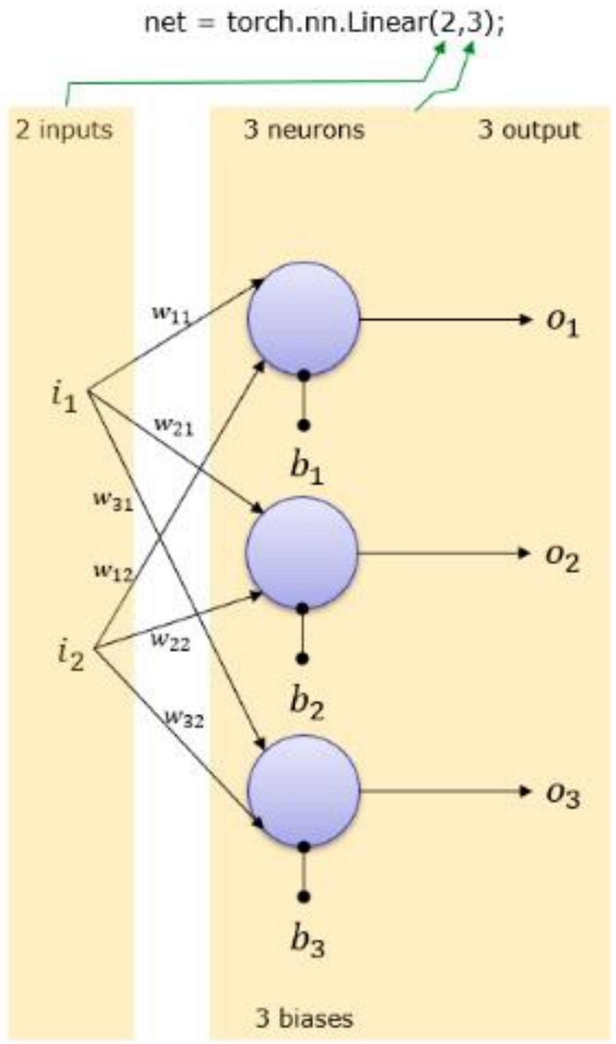
Creating a FeedForwardNetwork – (2,2)

```
import torch
net = torch.nn.Linear(2,1)
print('network structure : torch.nn.Linear(2,2) :\n',net)
print('Weight of network :\n',net.weight)
print('Bias of network :\n',net.bias)
```

```
network structure : torch.nn.Linear(2,2) :
        Linear(in_features=2, out_features=2, bias=True)
 Weight of network :
        Parameter containing:
            tensor([[ 0.4992, -0.1154],
                [ 0.2762, -0.0332]], requires_grad=True)
Bias of network :
        Parameter containing:
            tensor([-0.5019,  0.2884], requires_grad=True)
```

Creating a FeedForwardNetwork – (2,3)

- 2 Inputs and 3 output
- net = torch.nn.Linear(2,3)
- This creates a network as shown
- Weight and Bias is set automatically.



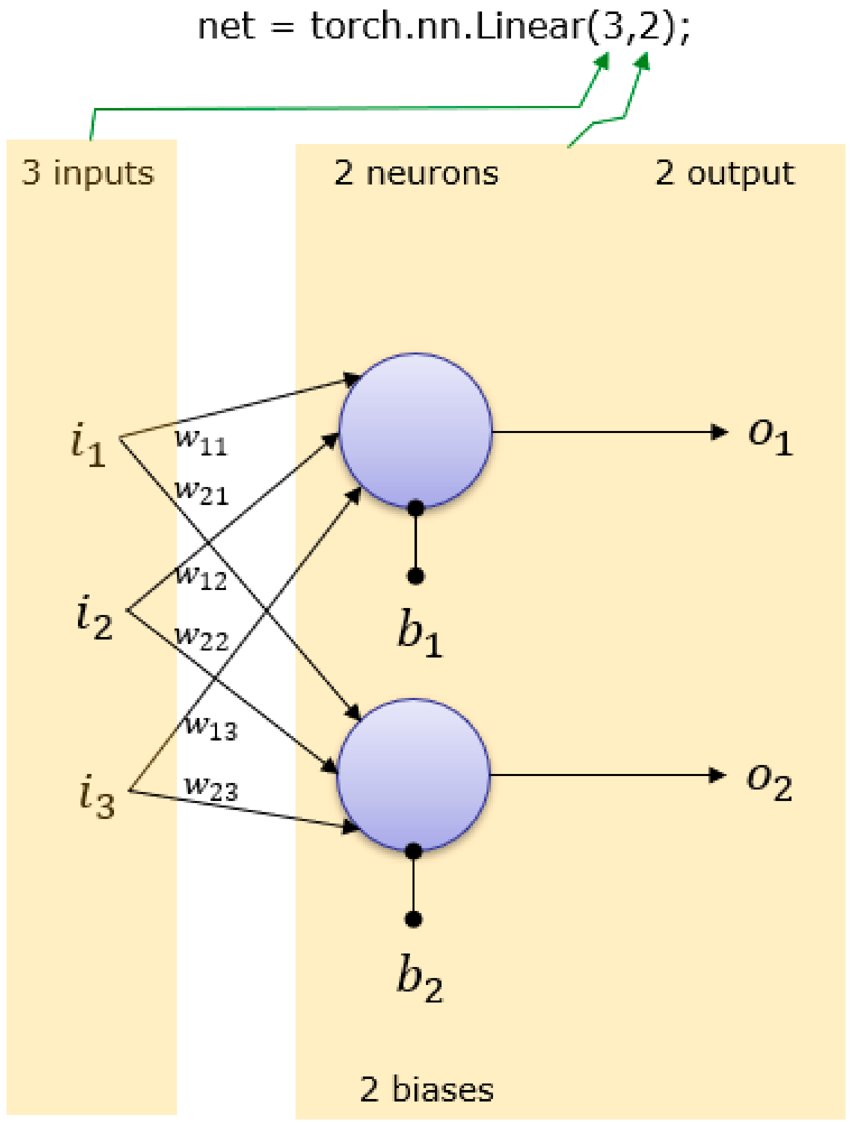
Creating a FeedForwardNetwork – (2,3)

```
import torch
net = torch.nn.Linear(2,3)
print('network structure : torch.nn.Linear(2,3) :\n',net)
print('Weight of network :\n',net.weight)
print('Bias of network :\n',net.bias)
```

```
network structure : torch.nn.Linear(2,3) :
        Linear(in_features=2, out_features=3, bias=True)
Weight of network :
        Parameter containing:
            tensor([[ 0.2799,  0.6430],
                    [ 0.4635, -0.2675],
                    [-0.1784, -0.4651]], requires_grad=True)
Bias of network :
        Parameter containing:
            tensor([-0.3769, -0.2818, -0.4946], requires_grad=True)
```

Creating a FeedForwardNetwork – (3, 2)

- 3 Inputs and 2 output
- net = torch.nn.Linear(3,2);
- This creates a network as shown
- Weight and Bias is set automatically.



Creating a FeedForwardNetwork – (3, 2)

```
import torch
net = torch.nn.Linear(3, 2)
print('network structure : torch.nn.Linear(3, 2) :\n',net)
print('Weight of network :\n',net.weight)
print('Bias of network :\n',net.bias)
```

```
 network structure : torch.nn.Linear(3,2) :
                Linear(in_features=3, out_features=2, bias=True)
    Weight of network :
                Parameter containing:
                        tensor([[ 0.3149, -0.0778,  0.0579],
                                [ 0.0947,  0.0997,  0.2743]], requires_grad=True)
                Bias of network :
                        Parameter containing:
                                tensor([-0.4785, -0.1434], requires_grad=True)
```