Introduction to artificial neural network (deep NN)

MIT Introduction to Deep Learning | 6.S191 https://youtu.be/5tvmMX8r_OM

Lee, Hung-yi Machine Learning (English version)
https://youtu.be/Y87Ct23H3Kw

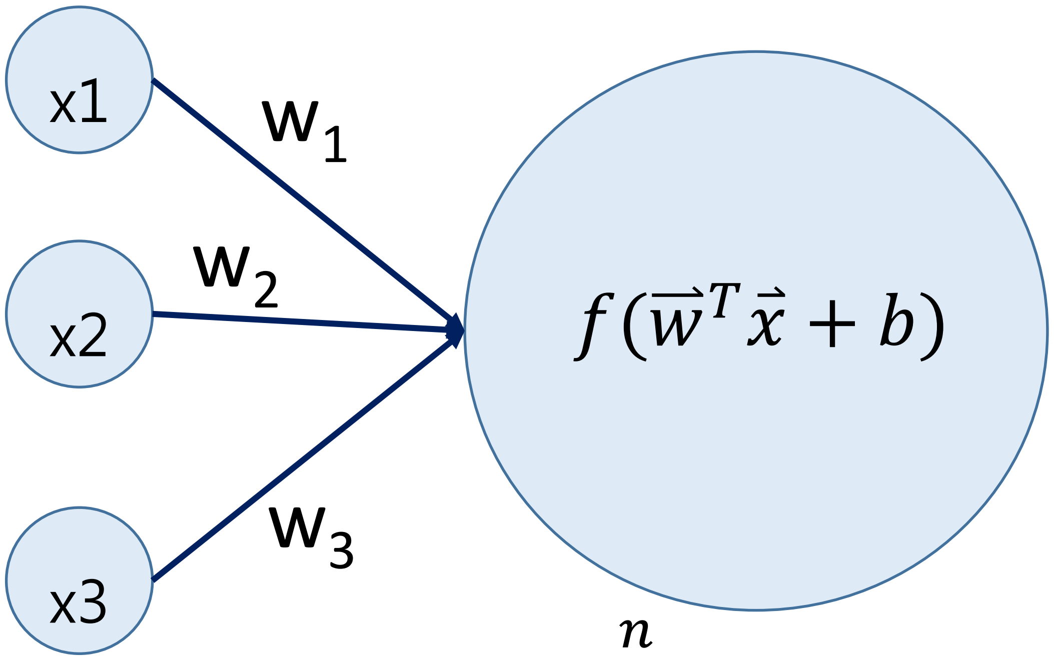
Lee, Hung-yi ML Lecture https://youtu.be/CXgbekl66jc

Neuron (perceptron)

- Neuron performs weighted linear combination with bias and activation function

1.1. Perceptron.ipynb

$$\begin{array}{ccc} x1 & x2 & x3 \\ [1 & 2 & 3] \end{array}$$

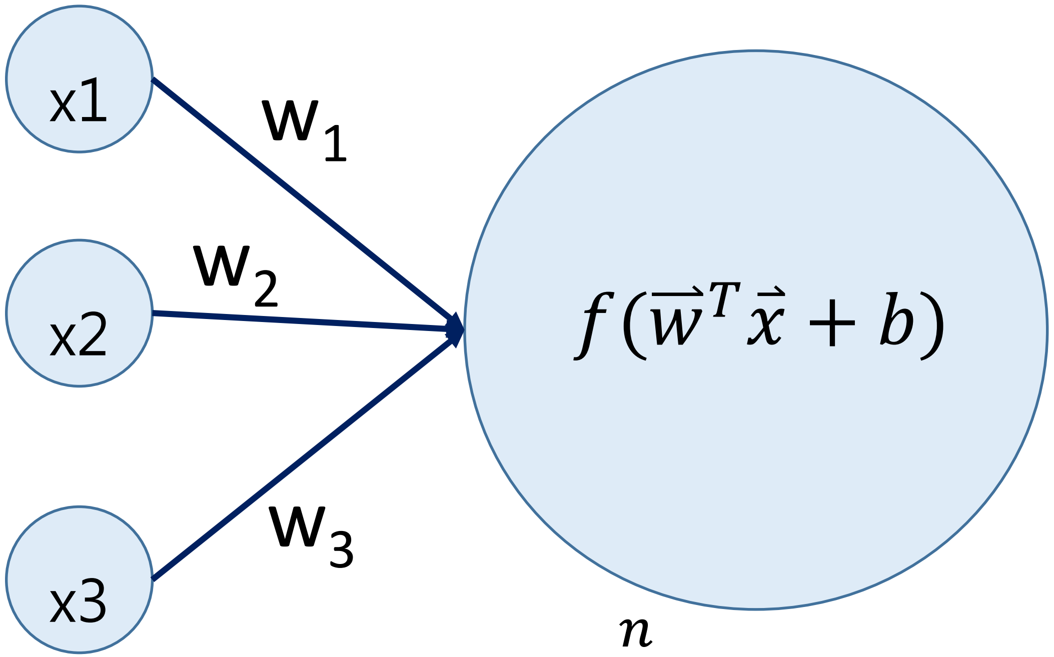


$$f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Neuron (perceptron)

1.1. Perceptron(2).ipynb

$$\begin{array}{ccc} x1 & x2 & x3 \end{array}$$
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$



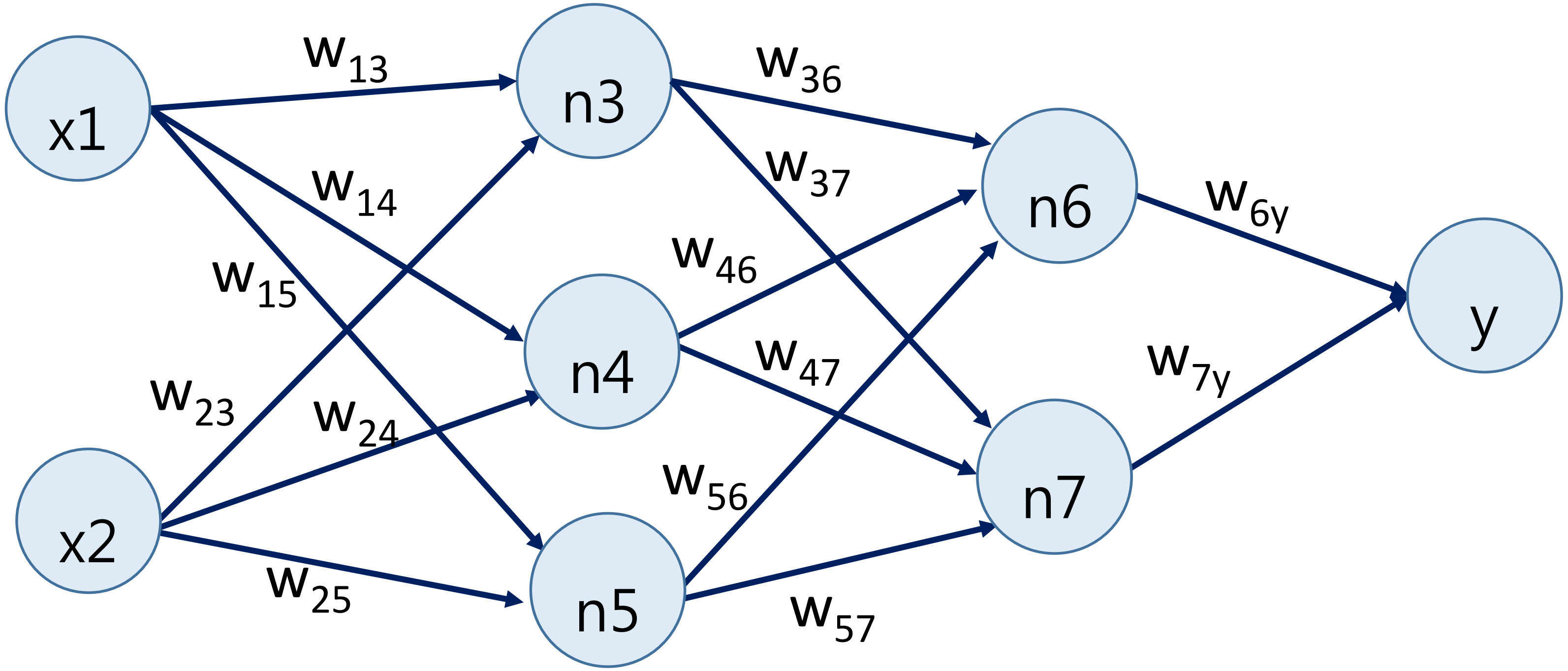
$$f(\sum_{i=1}^{n} w_i x_i + b)$$

$$\begin{bmatrix} x_1^1 & x_2^1 & x_3^1 \\ x_1^2 & x_2^2 & x_3^2 \\ x_1^3 & x_2^3 & x_3^3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

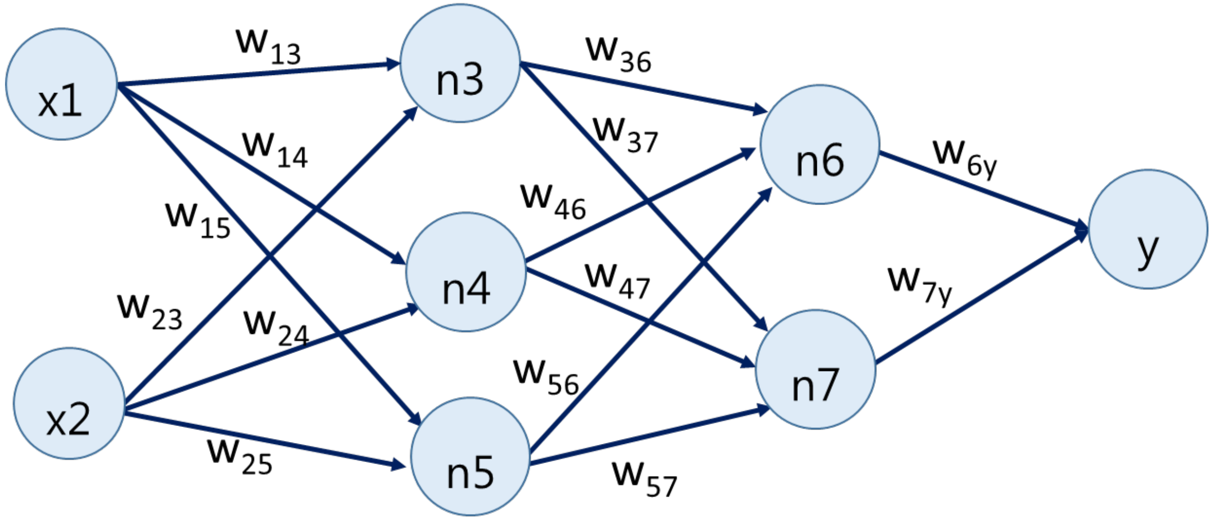
Multiple-layer perception (MLP)

1.2 MLP forward propagation.ipynb

```
MyNet = nn.Sequential(
    nn.Linear(2, 3),
    nn.Linear(3, 2),
    nn.Linear(2, 1)
)
print(MyNet)
```



Weights and bias




```
for param in MyNet.parameters():
    if param.requires_grad:
        print(param.data)
```

```
tensor([[ 0.4727, -0.5188],
        [-0.5681, -0.6032],
        [-0.0252, -0.3011]])
tensor([-0.6986, -0.6602, -0.4860])
tensor([[-0.5549,  0.2550,  0.4584],
        [ 0.2930,  0.0849, -0.3146]])
tensor([0.1677, 0.0736])
tensor([[ 0.4106, -0.3618]])
tensor([-0.2270])
```

$$\begin{bmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \\ w_{15} & w_{25} \end{bmatrix}$$

$$\begin{bmatrix} b_3 & b_4 & b_5 \end{bmatrix}$$

$$\begin{bmatrix} w_{36} & w_{46} & w_{56} \\ w_{37} & w_{47} & w_{57} \end{bmatrix}$$

$$\begin{bmatrix} b_6 & b_7 \end{bmatrix}$$

$$\begin{bmatrix} w_{6y} & w_{7y} \end{bmatrix}$$

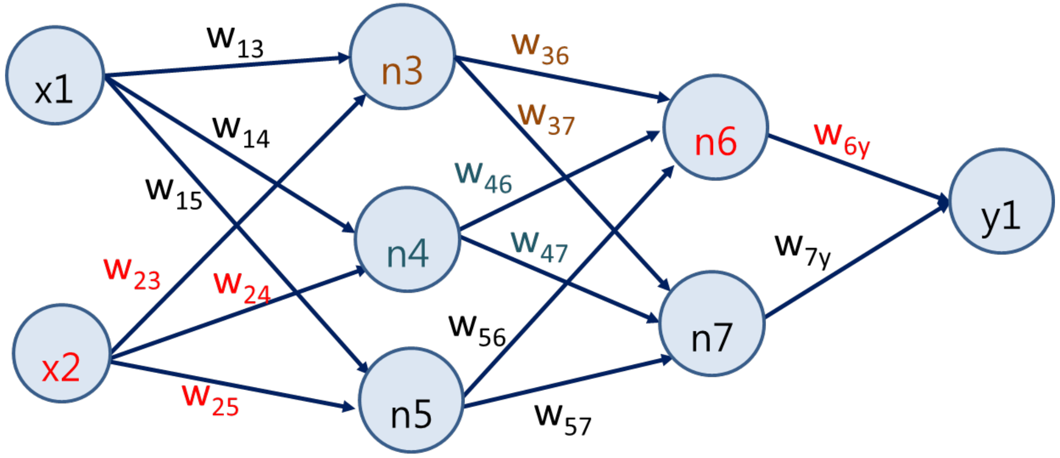
$$\begin{bmatrix} b_y \end{bmatrix}$$

Prepare input

```
lstX = [[1, 2], [2, 3], [10, 5]]
tensorX = torch.FloatTensor(lstX)
print(tensorX, "\n", tensorX.shape)
```

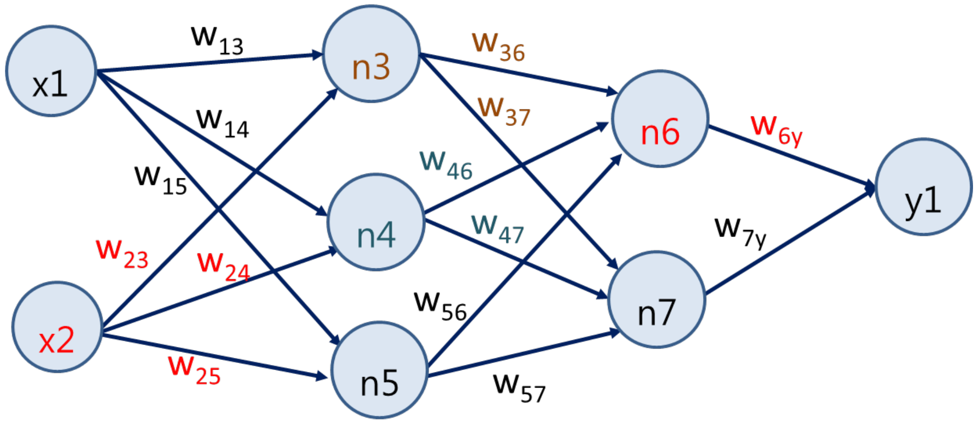
```
tensor([[ 1.,  2.],
        [ 2.,  3.],
        [10.,  5.]])
 torch.Size([3, 2])
```

$$\vec{X} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 10 & 5 \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 \\ x_1^2 & x_2^2 \\ x_1^3 & x_2^3 \end{bmatrix}$$



Forward propagation (layer 1)

```
#Calculate  n3,  n4,  n5  using  Pytorch  matrix  operation
Layer1_1  =  tensorX.mm(torch.transpose(W0,  1,  0))  +  b0
print(Layer1_1)
```

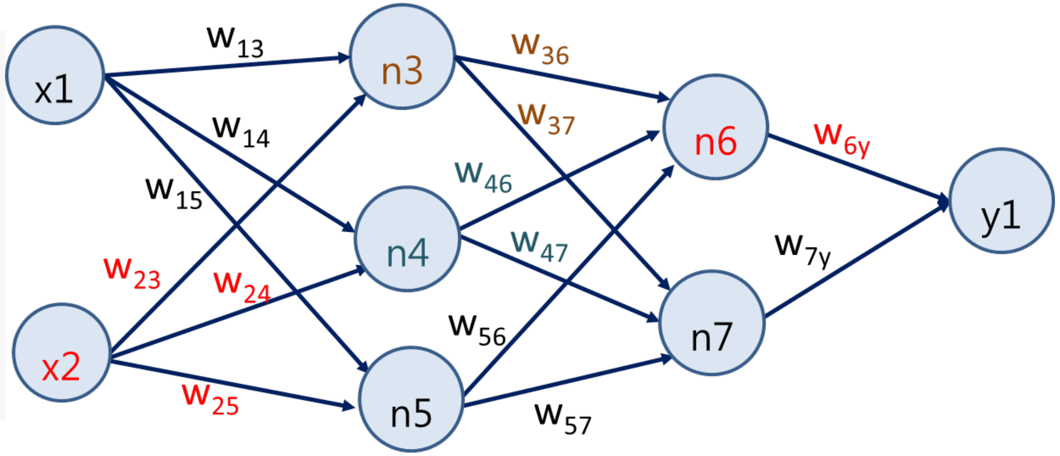


$$\begin{bmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \\ w_{15} & w_{25} \end{bmatrix}^T = \begin{bmatrix} w_{13} & w_{14} & w_{15} \\ w_{23} & w_{24} & w_{25} \end{bmatrix}$$

$$\begin{bmatrix} x_1^1 & x_2^1 \\ x_1^2 & x_2^2 \\ x_1^3 & x_2^3 \end{bmatrix} \begin{bmatrix} w_{13} & w_{14} & w_{15} \\ w_{23} & w_{24} & w_{25} \end{bmatrix} + \begin{bmatrix} b_3 & b_4 & b_5 \end{bmatrix} = \begin{bmatrix} k_3^1 & k_4^1 & k_5^1 \\ k_3^2 & k_4^2 & k_5^2 \\ k_3^3 & k_4^3 & k_5^3 \end{bmatrix} + \begin{bmatrix} b_3 & b_4 & b_5 \\ b_3 & b_4 & b_5 \\ b_3 & b_4 & b_5 \end{bmatrix} = \begin{bmatrix} n_3^1 & n_4^1 & n_5^1 \\ n_3^2 & n_4^2 & n_5^2 \\ n_3^3 & n_4^3 & n_5^3 \end{bmatrix}$$

Forward propagation (layer 2)

```
#Calculate n6, n7 using PyTorch matrix operation
W1 = MyNet[1].weight
b1 = MyNet[1].bias
Layer2_1 = Layer1.mm(torch.transpose(W1, 1, 0)) +b1
print(Layer2_1)
```



$$\begin{bmatrix} w_{36} & w_{46} & w_{56} \\ w_{37} & w_{47} & w_{57} \end{bmatrix}^T = \begin{bmatrix} w_{36} & w_{37} \\ w_{46} & w_{47} \\ w_{56} & w_{57} \end{bmatrix}$$

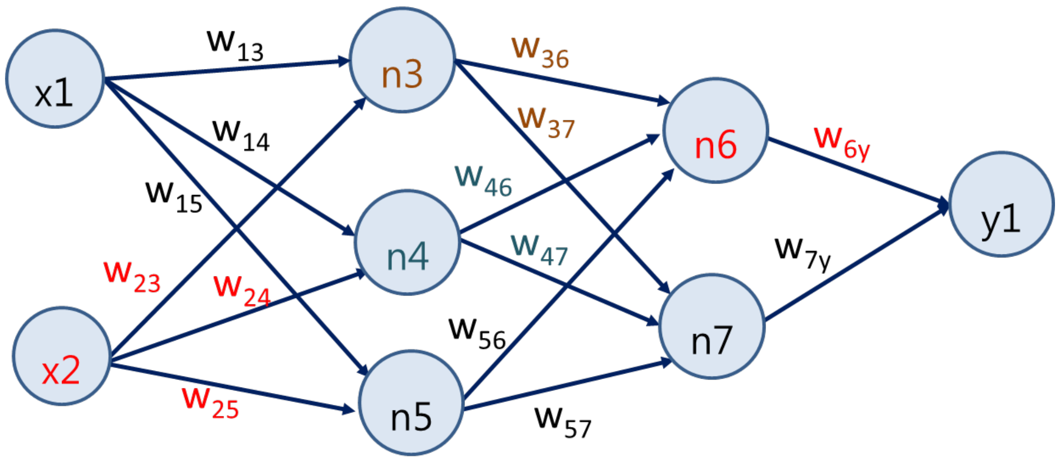
$$\begin{bmatrix} n_3^1 & n_4^1 & n_5^1 \\ n_3^2 & n_4^2 & n_5^2 \\ n_3^3 & n_4^3 & n_5^3 \end{bmatrix} \begin{bmatrix} w_{36} & w_{37} \\ w_{46} & w_{47} \\ w_{56} & w_{57} \end{bmatrix} + \begin{bmatrix} b_6 & b_7 \end{bmatrix} = \begin{bmatrix} k_6^1 & k_7^1 \\ k_6^2 & k_7^2 \\ k_6^3 & k_7^3 \end{bmatrix} + \begin{bmatrix} b_6 & b_7 \\ b_6 & b_7 \\ b_6 & b_7 \end{bmatrix} = \begin{bmatrix} n_6^1 & n_7^1 \\ n_6^2 & n_7^2 \\ n_6^3 & n_7^3 \end{bmatrix}$$

Forward propagation (output)

```
#Calculate y by matrix operation
W2 = MyNet[2].weight
b2 = MyNet[2].bias
tensorY_1 = Layer2.mm(torch.transpose(W2, 1, 0)) +b2
print(tensorY_1)
```

$$[w_{6y} \quad w_{7y}]^T = \begin{bmatrix} w_{6y} \\ w_{7y} \end{bmatrix}$$

$$\begin{bmatrix} n_6^1 & n_7^1 \\ n_6^2 & n_7^2 \\ n_6^3 & n_7^3 \end{bmatrix} \begin{bmatrix} w_{6y} \\ w_{7y} \end{bmatrix} + [b_y] = \begin{bmatrix} k_y^1 \\ k_y^2 \\ k_y^3 \end{bmatrix} + \begin{bmatrix} b_y \\ b_y \\ b_y \end{bmatrix} = \begin{bmatrix} y^1 \\ y^2 \\ y^3 \end{bmatrix}$$



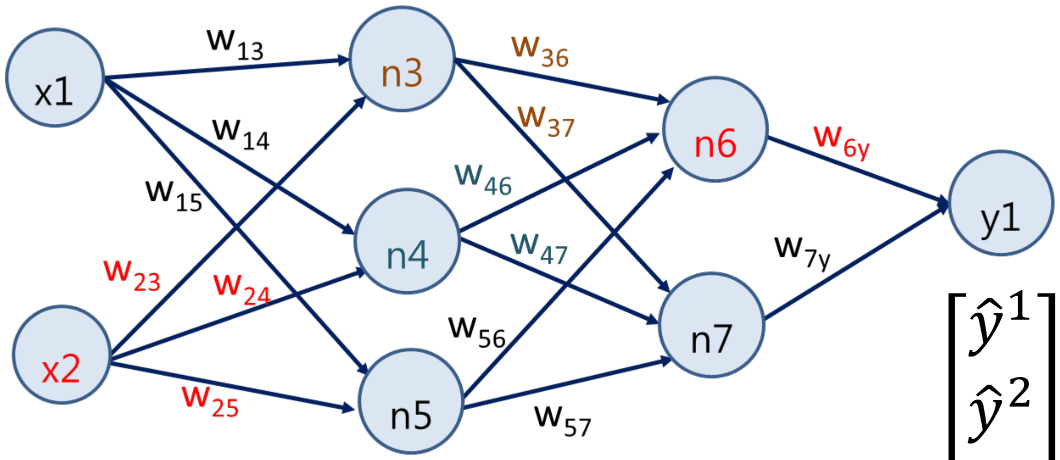
Calculate prediction error

1.3 MLP backward propagation.ipynb

```
lstX  =  [[1,  2],  [2,  3],  [10,  5]]
lstY  =  [[7],  [12],  [40]]    #  y=3x1+2x2
tensorX  =  torch.FloatTensor(lstX)
tensorY_hat  =  torch.FloatTensor(lstY)
```

$$\begin{bmatrix} y^1 \\ y^2 \\ y^3 \end{bmatrix} = \begin{bmatrix} 7 \\ 12 \\ 40 \end{bmatrix}$$

$$\vec{X} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 10 & 5 \end{bmatrix}$$



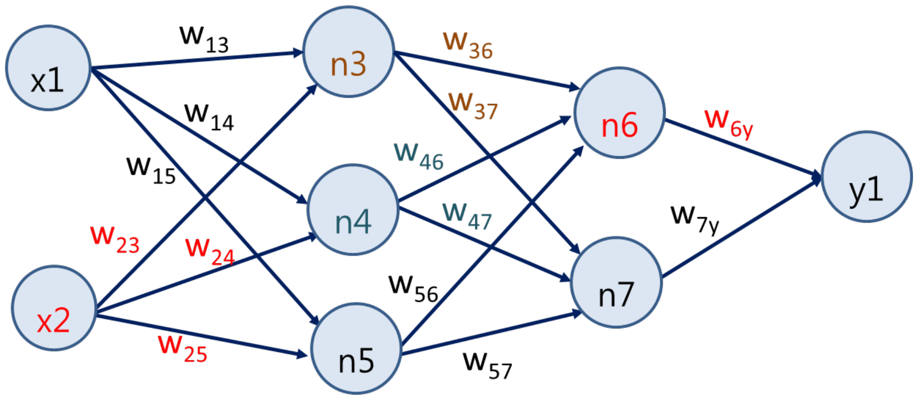
$$\begin{bmatrix} \hat{y}^1 \\ \hat{y}^2 \\ \hat{y}^3 \end{bmatrix}$$

```
loss  =  loss_func(tensorY ,  tensorY_hat)
print(loss)
```

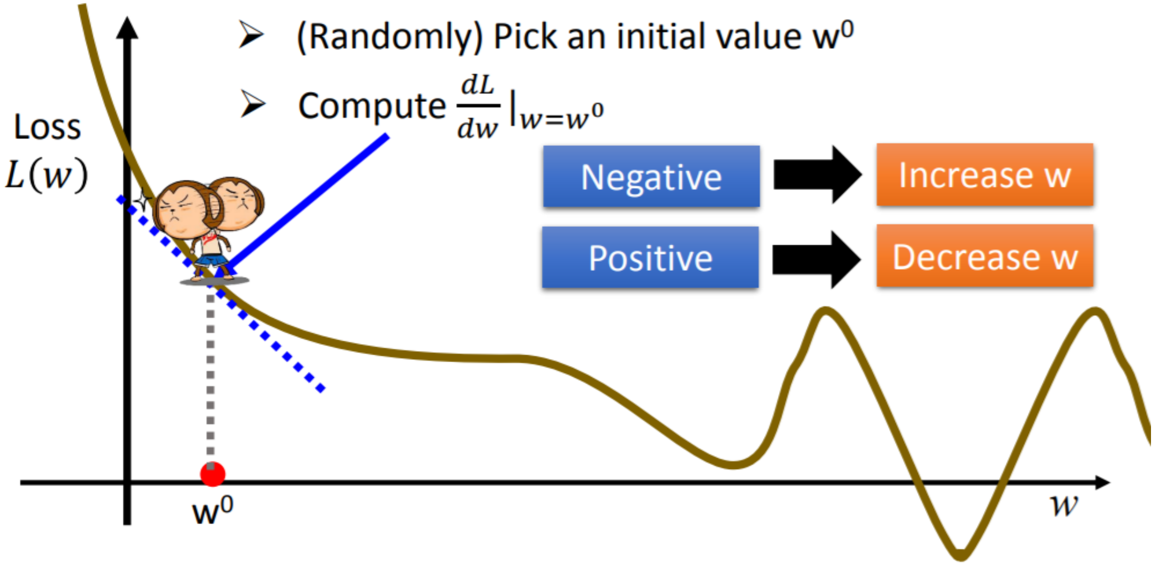
$$L = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{y}^i)^2$$

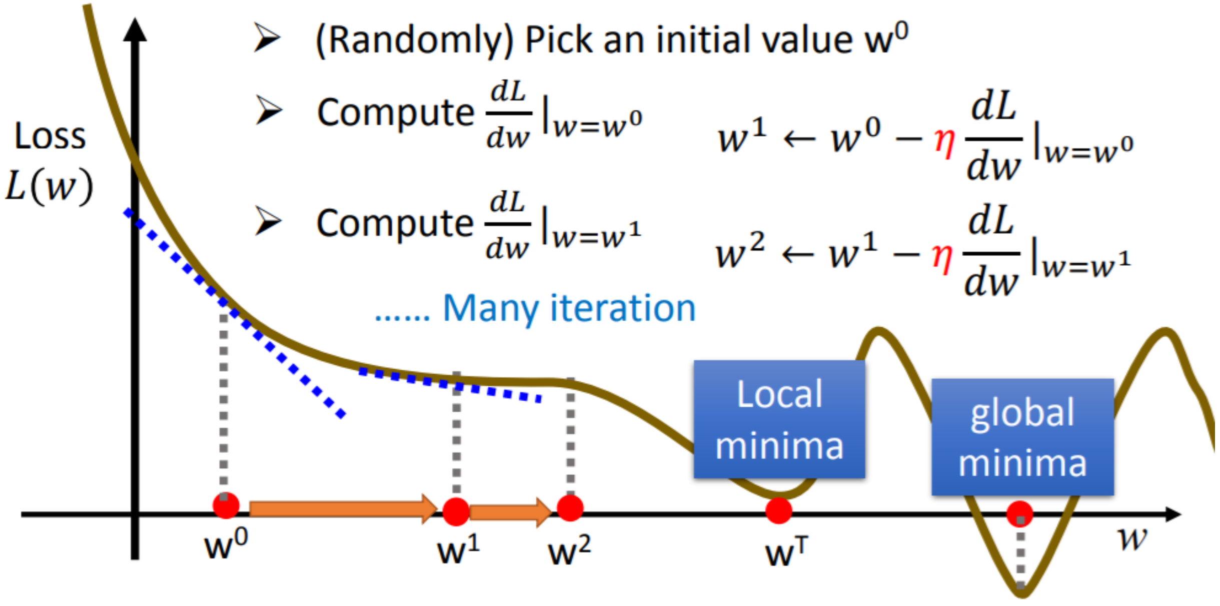
```
tensorY=  MyNet(tensorX)
print(tensorY)
```

Find optimal parameters that minimize the loss



$$L = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{y}^i)^2$$





Reference: 李弘毅 ML Lecture 1 https://youtu.be/CXgbekl66jc

Find optimal parameters that minimize the loss

- How about two parameters? $w^*, b^* = arg \min_{w,b} L(w,b)$

  - (Randomly) Pick an initial value w^0, b^0
  - Compute $\frac{\partial L}{\partial w}|_{w=w^0,b=b^0}$, $\frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$

$$\begin{bmatrix} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{bmatrix} \text{gradient}$$

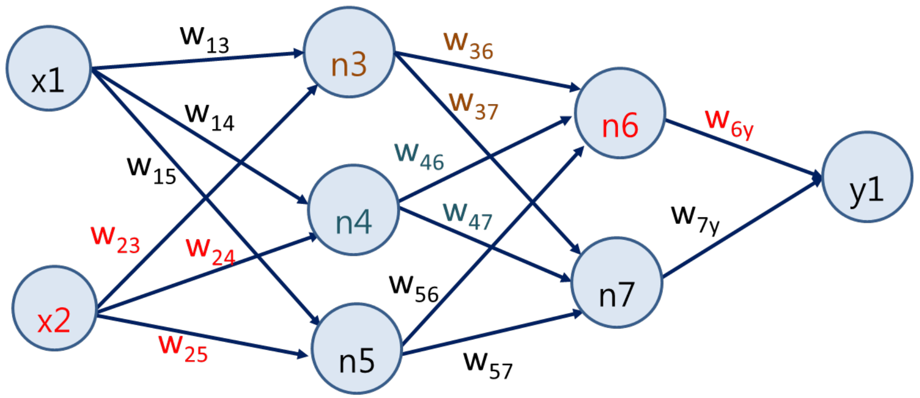
$$w^1 \leftarrow w^0 - \eta \frac{\partial L}{\partial w}|_{w=w^0,b=b^0} \qquad b^1 \leftarrow b^0 - \eta \frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$$

  - Compute $\frac{\partial L}{\partial w}|_{w=w^1,b=b^1}$, $\frac{\partial L}{\partial b}|_{w=w^1,b=b^1}$

$$w^2 \leftarrow w^1 - \eta \frac{\partial L}{\partial w}|_{w=w^1,b=b^1} \qquad b^2 \leftarrow b^1 - \eta \frac{\partial L}{\partial b}|_{w=w^1,b=b^1}$$

Reference: 李弘毅 ML Lecture 1 https://youtu.be/CXgbekl66jc

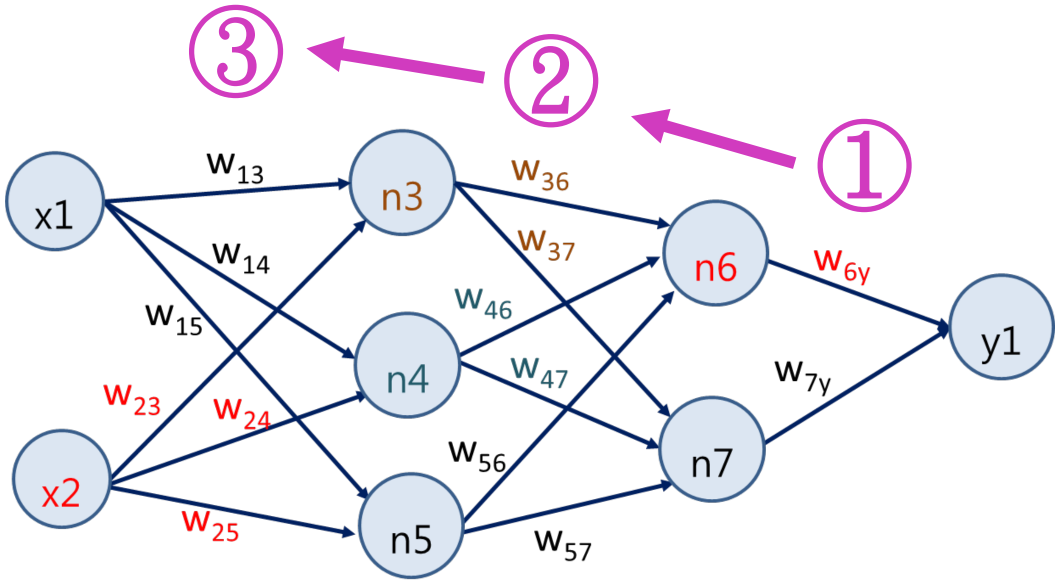
Gradient decent



$$L = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{y}^i)^2$$

$$\nabla L = [\frac{\partial L}{w_{13}}, \frac{\partial L}{w_{14}}, \frac{\partial L}{b_3}, \ldots, \frac{\partial L}{w_{36}}, \ldots]$$

Use gradient decent to find optimal NN weights



$$w_i \leftarrow w_i - \eta \frac{\partial e}{\partial w_i}$$

$$L = g(y - \hat{y}) \qquad y = \sigma(n_6 * w_{6y} + n_7 * w_{7y} + b_y)$$

①

$$w_{6y} \leftarrow w_{6y} - \eta \frac{\partial L}{\partial w_{6y}} \qquad \frac{\partial L}{\partial w_{6y}} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{6y}}$$

$$w_{7y} \leftarrow w_{7y} - \eta \frac{\partial L}{\partial w_{7y}} \qquad \frac{\partial L}{\partial w_{7y}} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{7y}}$$

②

$$w_{57} \leftarrow w_{57} - \eta \frac{\partial L}{\partial w_{57}} \qquad \frac{\partial L}{\partial w_{57}} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial n_7} \frac{\partial n_7}{\partial w_{57}}$$

$$n_7 = f(n_3 * w_{37} + n_4 * w_{47} + n_5 * w_{57} + b_7)$$

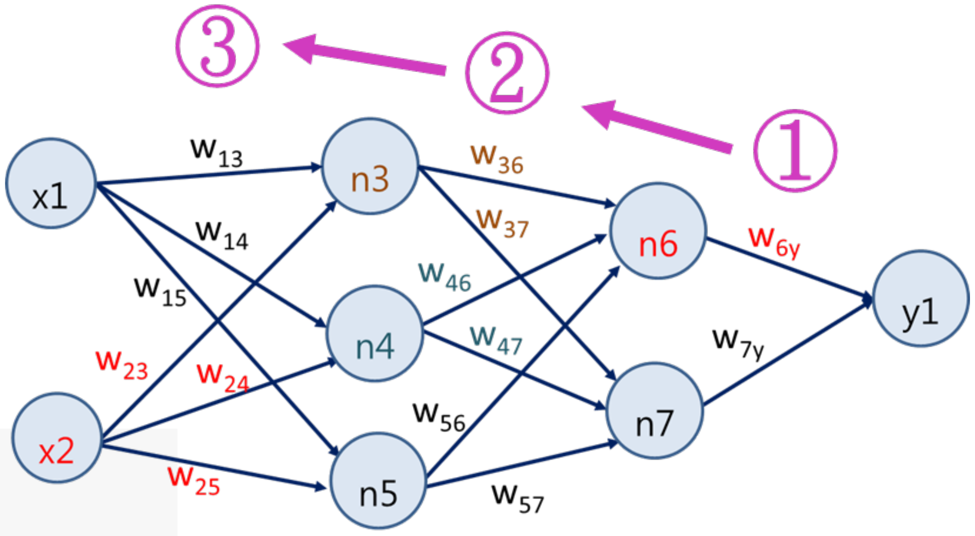
Back propagation

```
loss.backward()
```

```
for name, param in MyNet.named_parameters():
        if param.requires_grad:
                print(name, param.data, param.grad)
```

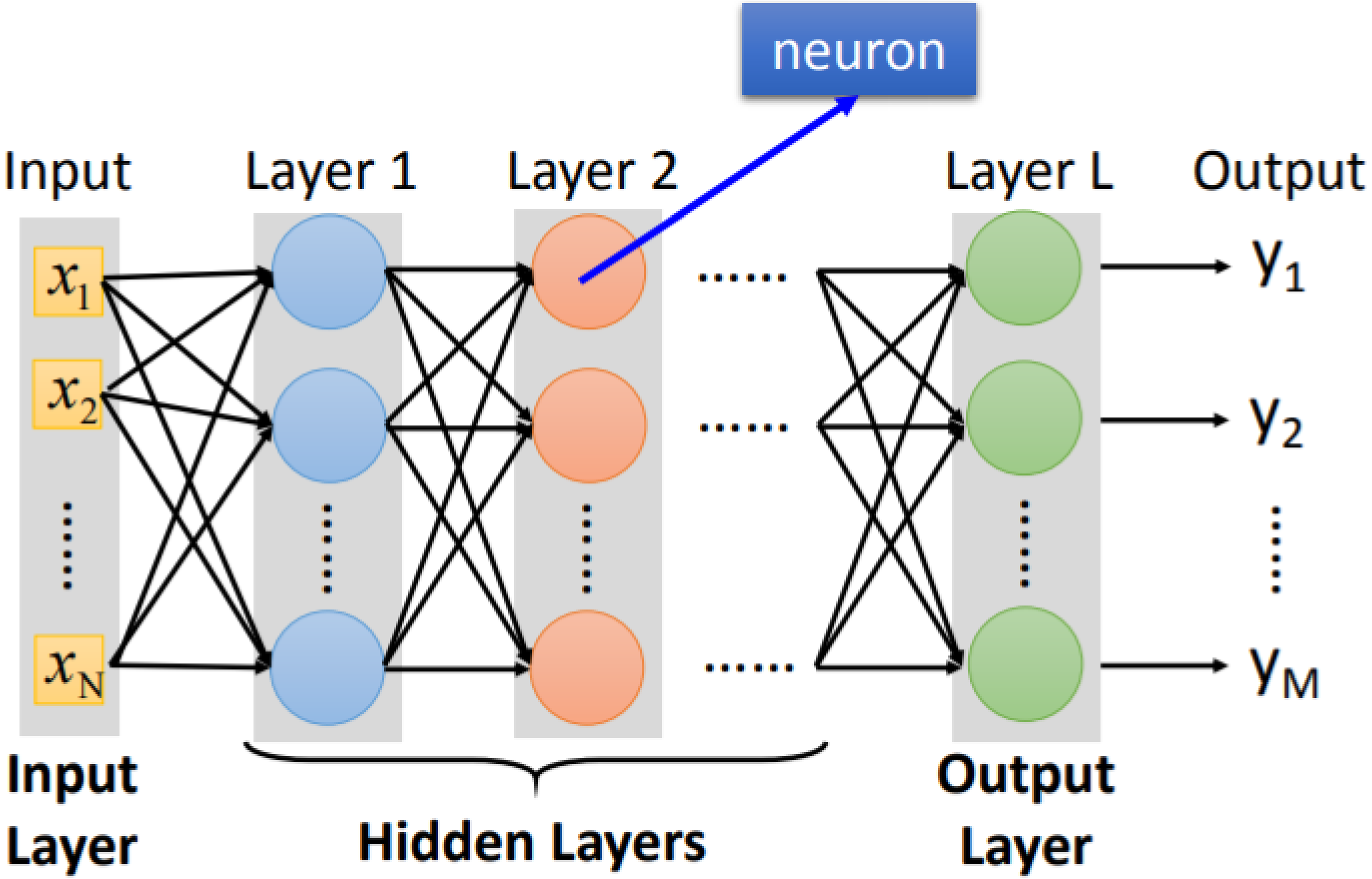
```
for name, param in MyNet.named_parameters():
        if param.requires_grad:
            param = param - learning_rate*param.grad

            nameLst = name.split(".")    #"0.weight" -> ['0', 'weight']
            layerNo = int(nameLst[0])
            s = nameLst[1]
            if(s=="weight"):
                MyNet[layerNo].weight = torch.nn.parameter.Parameter(param)
            elif(s=="bias"):
                MyNet[layerNo].bias = torch.nn.parameter.Parameter(param)
            else:
                print("wrong label")
```



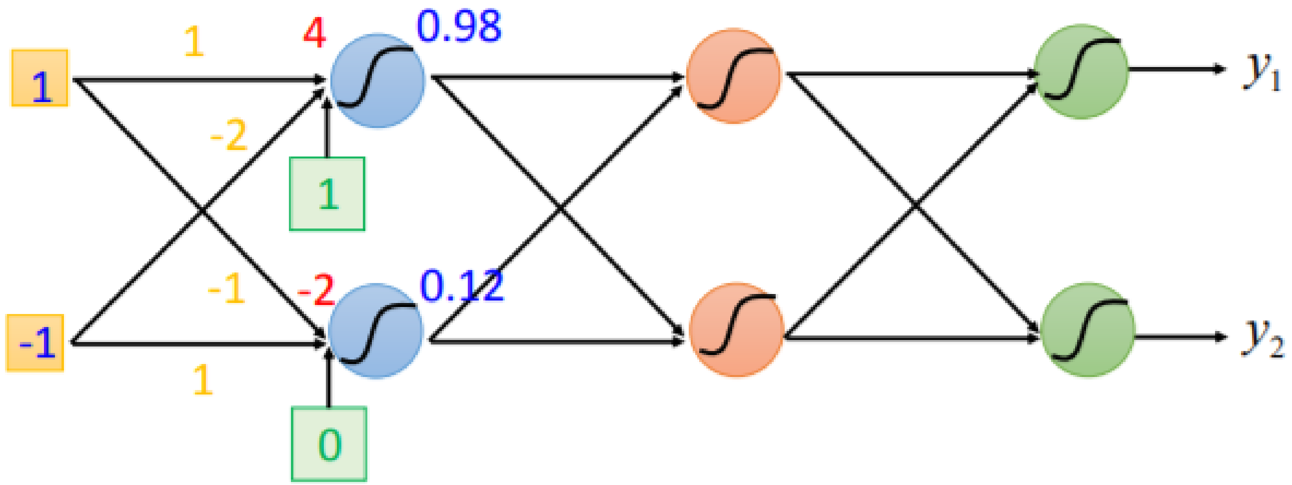
$$w_i \leftarrow w_i - \eta\frac{\partial e}{\partial w_i}$$

MLP is a fully connected feedforward network



Reference: 李弘毅 ML Lecture 6 https://youtu.be/Dr-WRlEFefw

Fully connected feed forward network is implemented as matrix operation

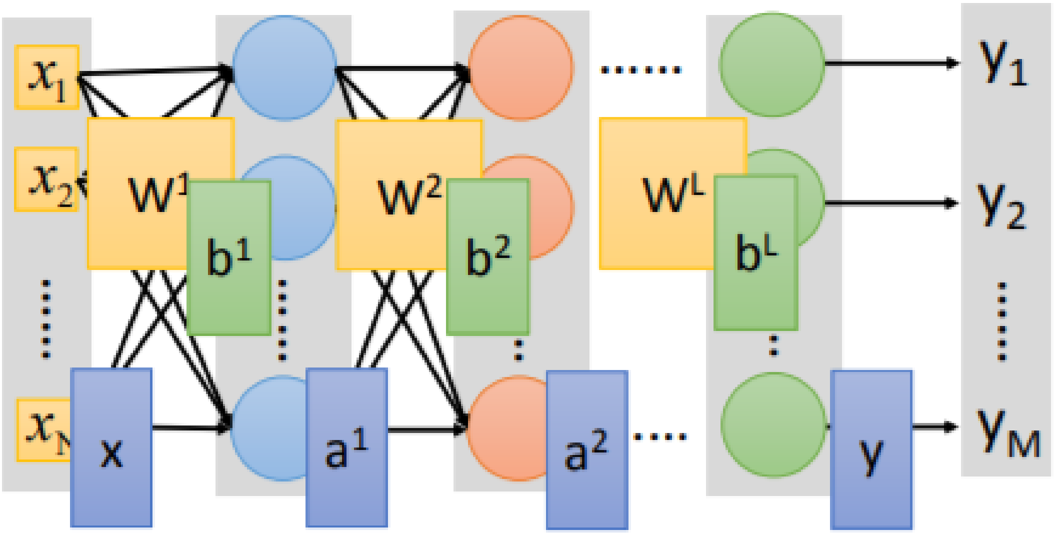


$$y = \sigma(w \cdot x + b)$$

$$\sigma\Big(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}} \Big) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Reference: 李弘毅 ML Lecture 6 https://youtu.be/Dr-WRlEFefw

Use parallel computing to speed up matrix operation



$$y = f(x)$$

Using parallel computing techniques to speed up matrix operation

$$= \sigma(W^L \cdots \sigma(W^2\, \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

Reference: 李弘毅 ML Lecture 6 https://youtu.be/Dr-WRlEFefw