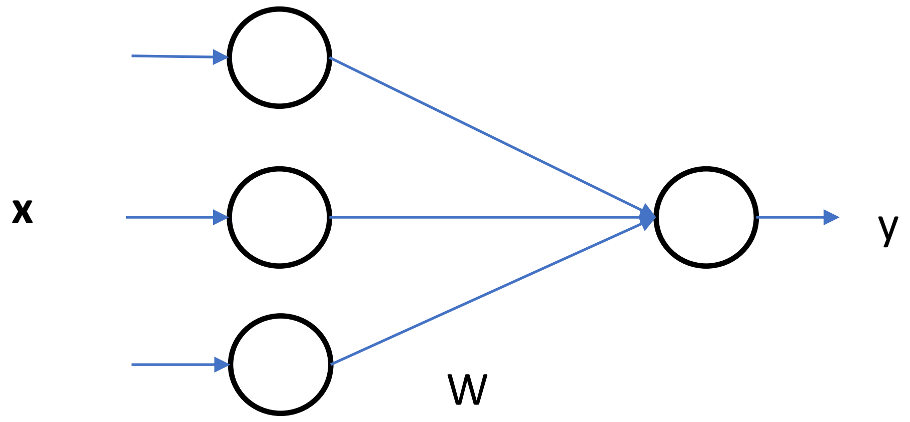
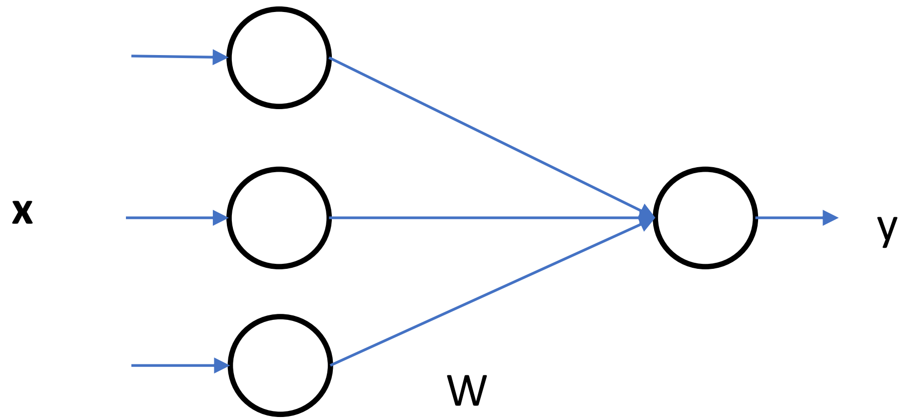
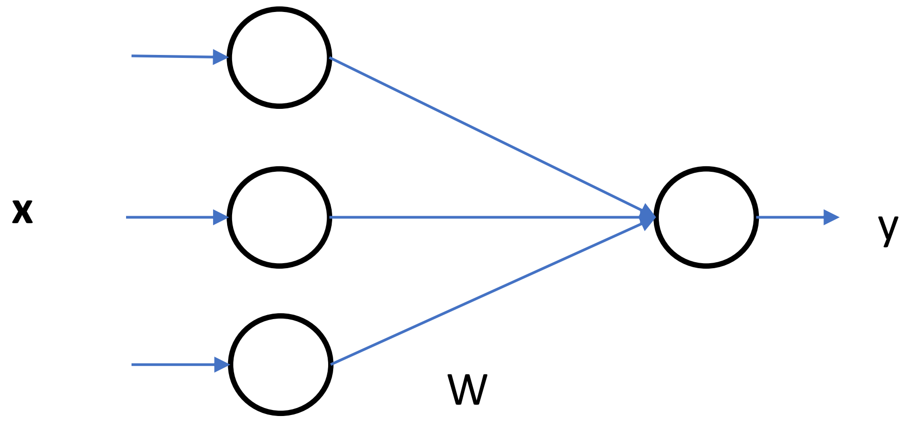
Lesson 25-27 : Implementing Neural Network in Python


x
y
W



$$perceptron = \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{W}$$

$$y = sigmoid(perceptron)$$

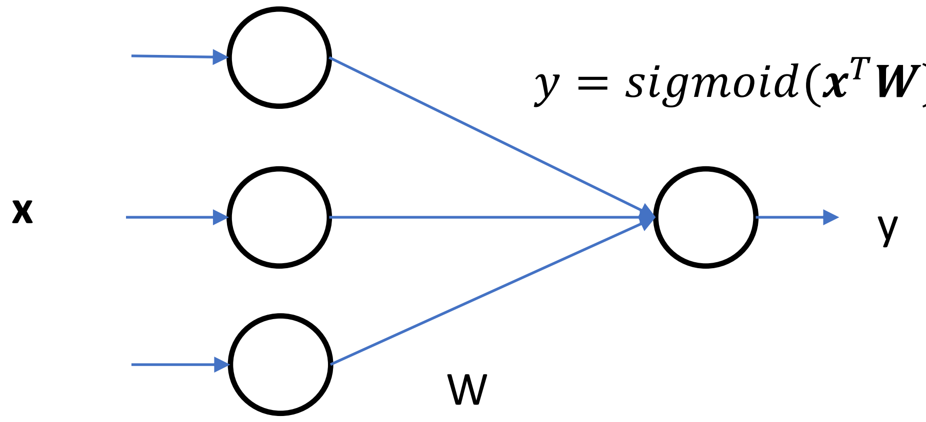


$$y = sigmoid(\boldsymbol{x}^T \boldsymbol{W})$$

$$perceptron = \boldsymbol{x}^T \boldsymbol{W}$$

$$y = sigmoid(perceptron)$$

Feed forward pass

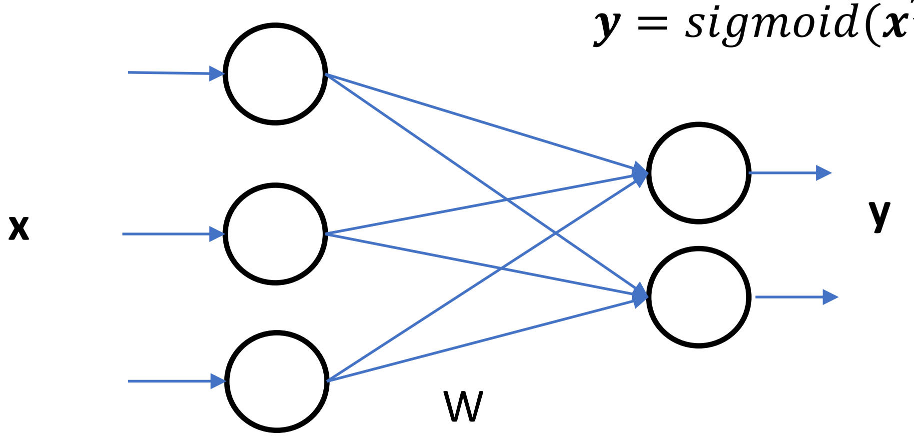


$$y = sigmoid(\boldsymbol{x}^T \boldsymbol{W})$$

```python
import numpy as np

D = np.array([[1,2,3],      # dataset
              [1,0,2],
              [0,1,4],
              [2,1,4]])

# Initialize weight matrix
np.random.seed(1)
w = np.random.random((3,1))
print("Weight Matrix : ")
print(w)

#Forward Pass
for iteration in range(1):
    iLayer = D
    oPer = np.dot(iLayer,w)                        # Perceptron
    oLayer = 1/(1+np.exp(-oPer))       # Sigmoid

print("Input :")
print(D)
print("Predicted Output: ")
print(oLayer)
```

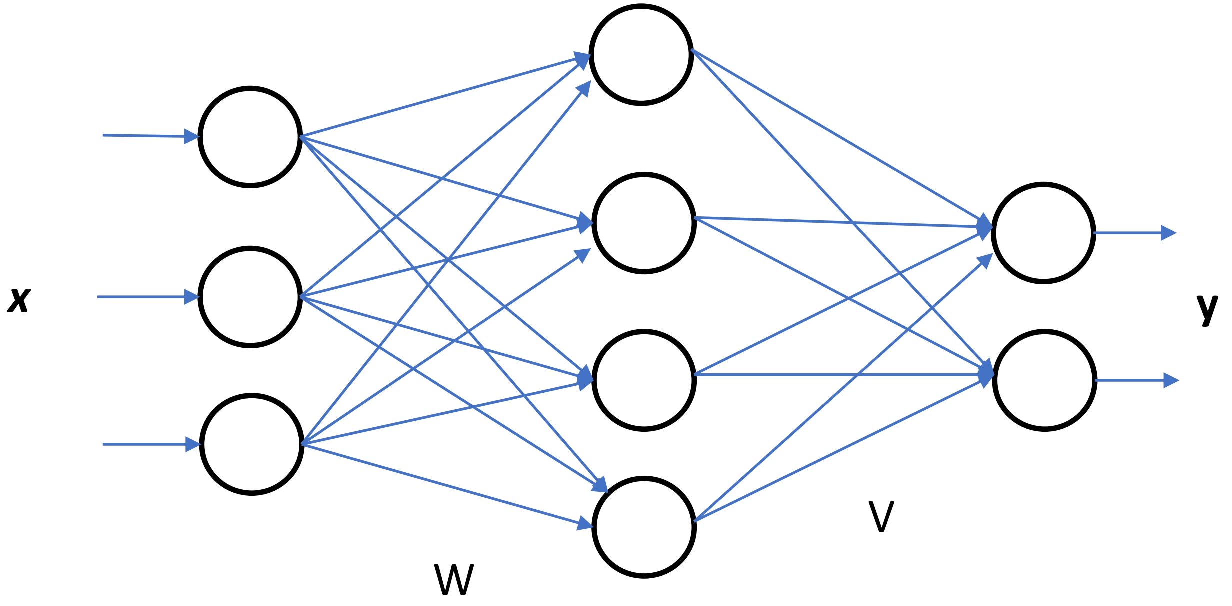
Feed forward pass

$$y = sigmoid(\boldsymbol{x}^T \boldsymbol{W})$$




```
import numpy as np

D = np.array([[1,2,3],     # dataset
              [1,0,2],
              [0,1,4],
              [2,1,4]])

# Initialize weight matrix
np.random.seed(1)
w = np.random.random((3,2))
print("Weight Matrix : ")
print(w)

#Forward Pass
for iteration in range(1):
    iLayer = D
    oPer = np.dot(iLayer,w)                # Perceptron
    oLayer = 1/(1+np.exp(-oPer))      # Sigmoid

print("Input :")
print(D)
print("Predicted Output: ")
print(oLayer)
```

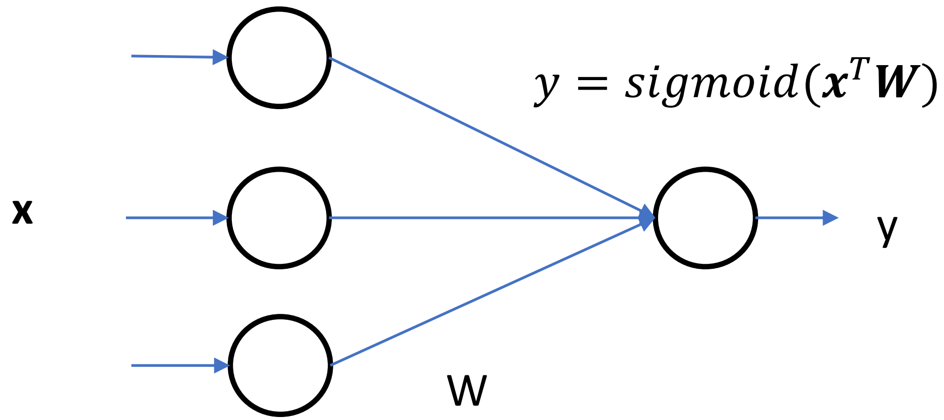
Feed forward pass



$$y = sigmoid(sigmoid(\boldsymbol{x}^T\boldsymbol{W})^{\boldsymbol{T}}V)$$

```
import numpy as np

D = np.array([[1,2,3],
              [1,0,2],
              [0,1,4],
              [2,1,4]])

# Initialize Weight Matrices
np.random.seed(1)
W = np.random.random((3,4))
print("W weight matrix:")
print(W)

print("V weight matrix:")
V = np.random.random((4,2))
print(V)

for iteration in range(1):
    iLayer = D

    hP = np.dot(iLayer,W)          # Hidden Layer
    hLayer = 1/(1+np.exp(-hP))

    oP = np.dot(hLayer,V)          # Output Layer
    oLayer = 1/(1+np.exp(-oP))

print("Input :")
print(training)
print("Predicted Output: ")
print(oLayer)
```

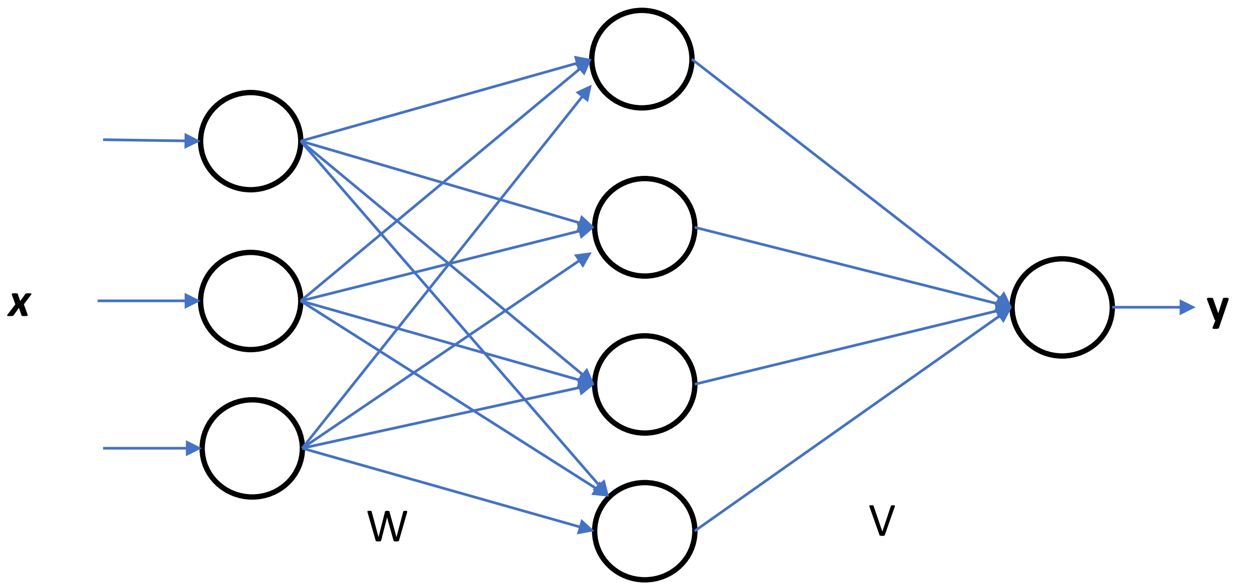
Backpropagation



$$\frac{\delta E}{\delta W_{11}} = \frac{\delta z_1}{\delta W_{11}} \times \frac{\delta y_1}{\delta z_1} \times \frac{\delta E}{\delta y_1}$$

$$\frac{\delta E}{\delta W_{11}} = x \ \times z(1-z) \times 2(\acute{y} - y)$$

```
import numpy as np

D = np.array([[4,500,6],
              [4,550,5.5],
              [2,200,3.5],
              [2,250,4]])
label = np.array([[1,1,0,0]]).T

np.random.seed(1)
w = np.random.random((3,1))

for iteration in range(10):
    iLayer = D
    p = np.dot(iLayer,w)       # Perceptron
    oLayer = 1/(1+np.exp(-p))  # Sigmoid(x)

    MSE = 2*np.square(np.subtract(oLayer,label)).mean() # Mean Square Error
    print(MSE)

    der = oLayer * (1-oLayer) # dirivatives of sigmoid
    grad = np.dot(iLayer.T, der *MSE)

    w += 0.01*grad
    print(w)

print(oLayer)
```



$$\frac{\delta E}{\delta V_{11}} = \frac{\delta z_1}{\delta V_{11}} \times \frac{\delta y_1}{\delta z_1} \times \frac{\delta E}{\delta y_1} = \frac{\delta E}{\delta V_{11}}$$
$$= x \quad \times z(1-z) \times 2(\acute{y} - y)$$

$$\frac{\delta E}{\delta W_{11}} = \frac{\delta a_1}{\delta W_{11}} \times \frac{\delta h_1}{\delta a_1} \times \frac{\delta z_1}{\delta h_1} \times \frac{\delta y_1}{\delta z_1} \times \frac{\delta E}{\delta y_1}$$
$$= x \times a(1-a) \times V_{11} \times z(1-z) \times 2(\acute{y} - y)$$

```
import numpy as np

D = np.array([[4,500,6],
              [4,550,5.5],
              [2,200,3.5],
              [2,250,4]])
label = np.array([[1,1,0,0]]).T

np.random.seed(1)
w = np.random.random((3,4))
v = np.random.random((4,1))

for iteration in range(10):
    iLayer = D
    hP = np.dot(iLayer,w)        # Perceptron
    hLayer = 1/(1+np.exp(-hP))   # Sigmoid(x)

    oP = np.dot(hLayer,v)        # Perceptron
    oLayer = 1/(1+np.exp(-oP))   # Sigmoid(x)

    MSE = 2*np.square(np.subtract(oLayer,label)).mean() # Mean Square Error
    print(MSE)

    oDer = oP * (1-oP) # dirivatives of sigmoid
    vGrad = np.dot(oLayer.T, oDer *MSE)
    v += 0.00000001*vGrad
    print(v)

    hDer = hP * (1-hP) # dirivatives of sigmoid
    wGrad = np.dot(iLayer.T, hDer *v*oDer*MSE)
    w += 0.00000001*wGrad
    print(w)

print(oLayer)
```