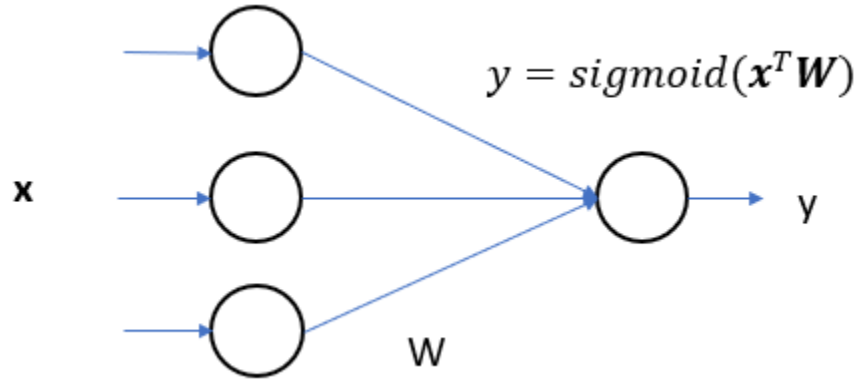
Lesson 28-29

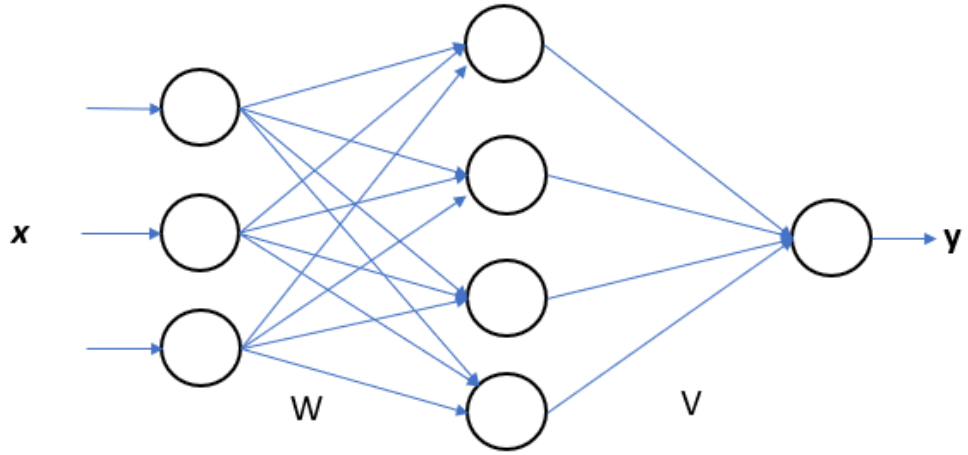
Tensorflow and Keras




```
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)
```

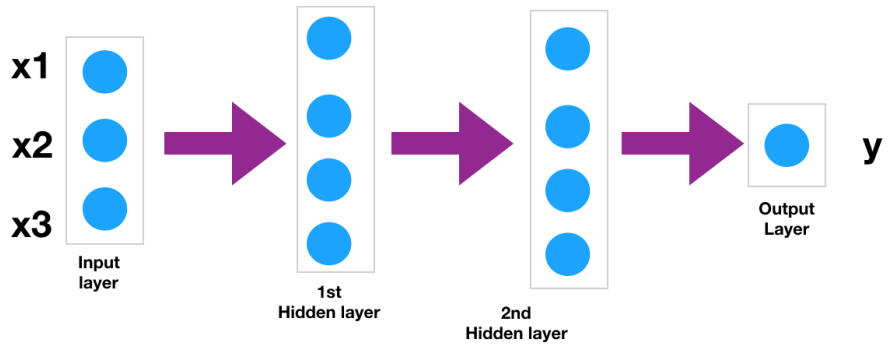
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

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


model = Sequential()
model.add(Dense(1, input_shape=(3,),activation='sigmoid'))
model.summary()

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(training, output, epochs=10, batch_size=250, verbose=1, validation_split=0.2)
```

```python
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)
```



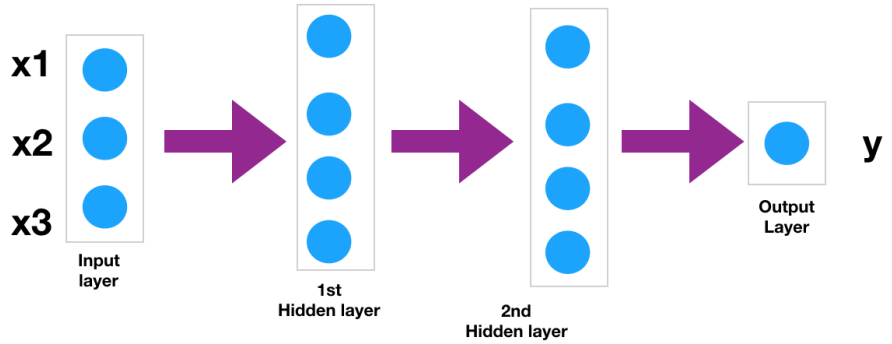

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

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


model = Sequential()
model.add(Dense(4, input_shape=(3,),activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(training, output, epochs=10, batch_size=250, verbose=1, validation_split=0.2)
```

Sequential Vs Functional Keras API

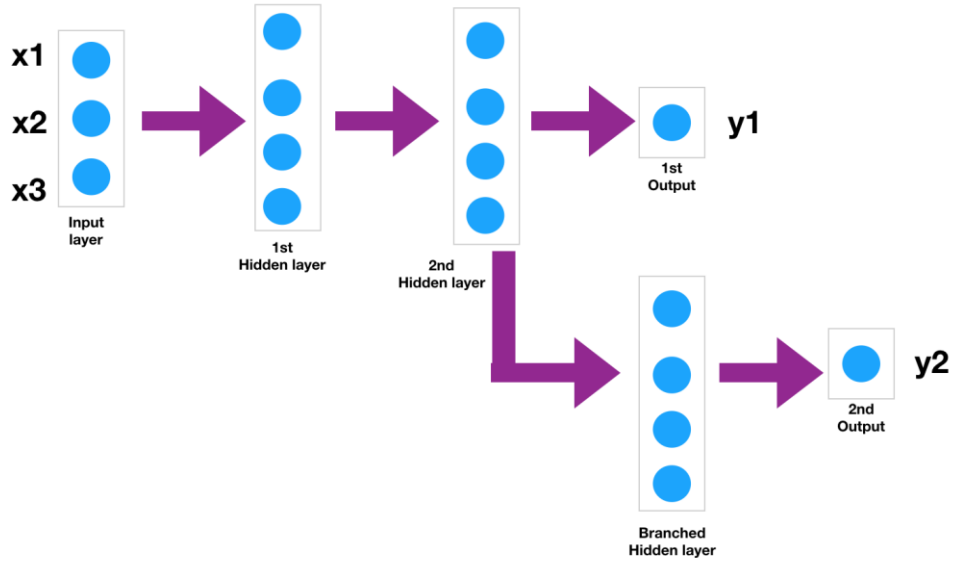


Sequential

Sequential Vs Functional Keras API

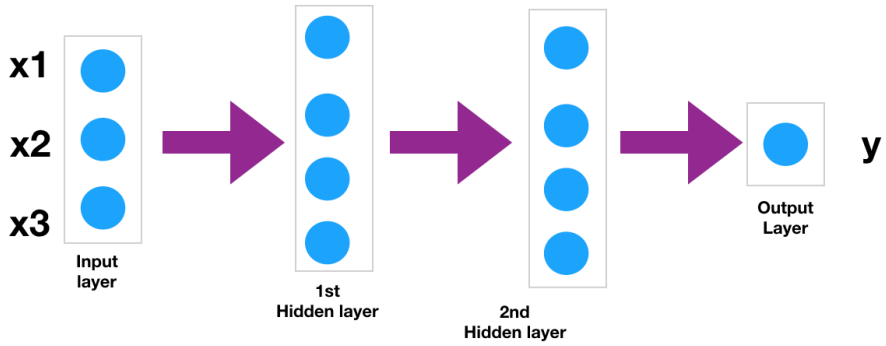


Sequential



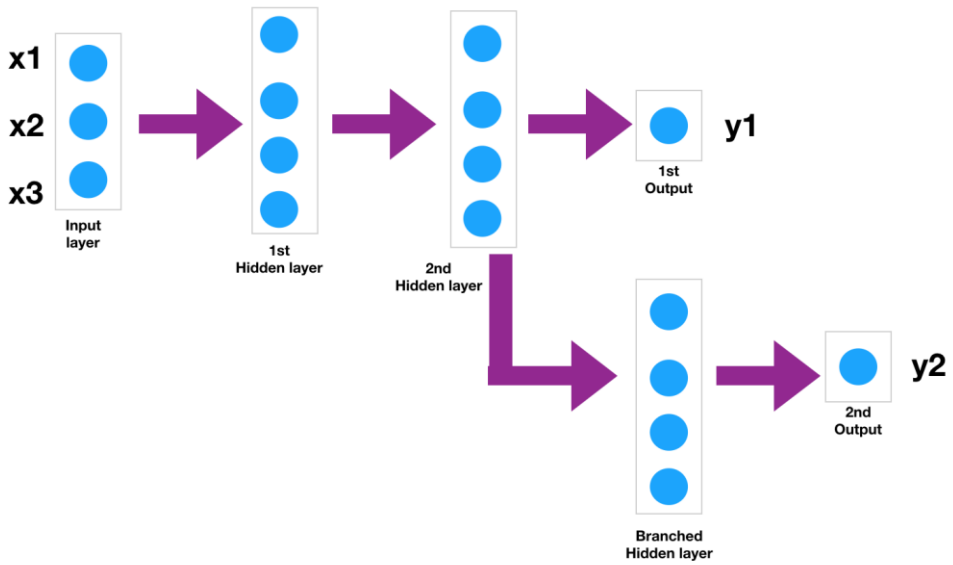
Functional

Functional Keras API




```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,Dense

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

## Creating the layers
input_layer = Input(shape=(3,))
layer_1 = Dense(4, activation="relu")(input_layer)
layer_2 = Dense(4, activation="relu")(layer_1)
o_layer = Dense(4, activation="relu")(layer_2)

##Defining the model by specifying the input and output layers
model = Model(inputs=input_layer, outputs=o_layer)
model.summary()

## defining the optimiser and loss function
model.compile(optimizer='adam', loss='mse')

## training the model
model.fit(D, label,epochs=2, batch_size=128,validation_data=(D,label))
```

Functional Keras API




```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,Dense

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

## Creating the layers
input_layer = Input(shape=(3,))
layer_1 = Dense(4, activation="relu")(input_layer)
layer_2 = Dense(4, activation="relu")(layer_1)
layer_3 = Dense(4, activation="relu")(layer_2)
o1_layer= Dense(1, activation="linear")(layer_2)
o2_layer= Dense(1, activation="linear")(layer_3)

##Defining the model by specifying the input and output layers
model = Model(inputs=input_layer, outputs=[o1_layer,o2_layer])
model.summary()

## defining the optimiser and loss function
model.compile(optimizer='adam', loss='mse')

## training the model
model.fit(D, label,epochs=2, batch_size=128,validation_data=(D,label))
```