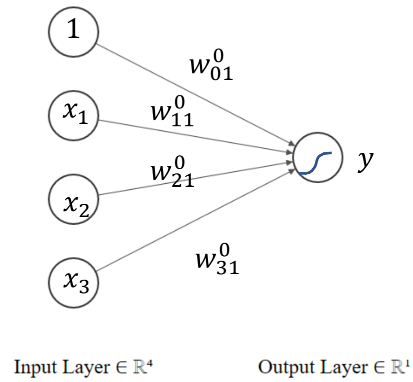
Python Tutorial: Building Multi-Layer Nets

Example 1: Create NNet(3,1): No Hidden Layer, Sigmoid Activation; 2-Class Classification



Import Numpy and matplotlib:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Create two classes with 3 features:

```
class1 = np.random.randn(20,3) + np.array([10,5,3])
class2 = np.random.randn(20,3) + np.array([5,10,8])

X1 = np.hstack(((np.ones((20,1)),class1)))              #20x4 for class 1
X2 = np.hstack(((np.ones((20,1)),class2)))              #20x4 for class 2
X = np.vstack ((X1,X2))                 #40x3 combined samples
Y = np.vstack ((np.ones((20,1)), np.zeros((20,1))))     #y=1 for class1, y=0 for class 2
```

Visualize data as a 3D scatter plot:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:20,1],X[:20,2],X[:20,3],marker='o')
ax.scatter(X[20:,1],X[20:,2],X[20:,3],marker='^')
ax.view_init(30, 40)                    #change view axis
```

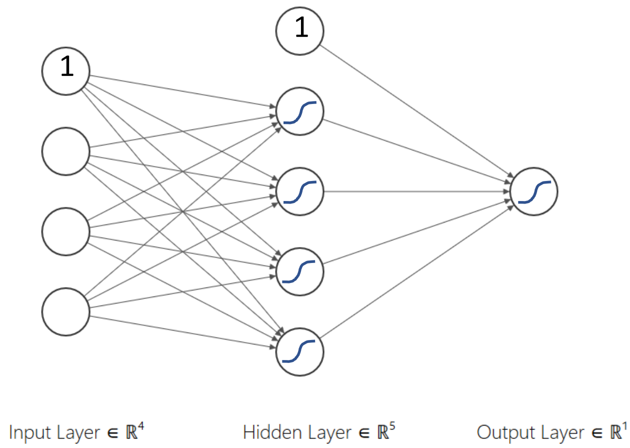
Create a weights:

```
width1=1                        #number neurons in output layer
W0 = np.random.rand(X.shape[1],width1)  #weights from layer 0
                                        #[b w1 w2 w3]
```

Set max epoc (iterations), learning rate, samples, and process data:

```
itera = 10000
alpha = 0.01
m = X.shape[0]          #number of samples
E = np.zeros(itera,)            #initialize error vector (not needed)

for epoc in range(itera):
    Z1 = np.dot(X,W0)
    a1 = 1/(1+np.exp(-Z1))      #sigmoid activation function of output neuron
    Yhat = a1           #nnet output
    d = Yhat - Y            #delta
    E[epoc] = np.sum (0.5*(d**2))   #MSE (not needed)

    g1 = a1*(1-a1)                  #derivative of sigmoid
    dEdW0 = np.dot(X.T, d * g1)         #dE/dW
    W0 -= alpha/m*dEdW0             #update weights

plt.plot(range(itera),E)
```

Example 2: Create NNet(3,4,1) : 1 Hidden Layer, Sigmoid Activation, 2-Class Classification



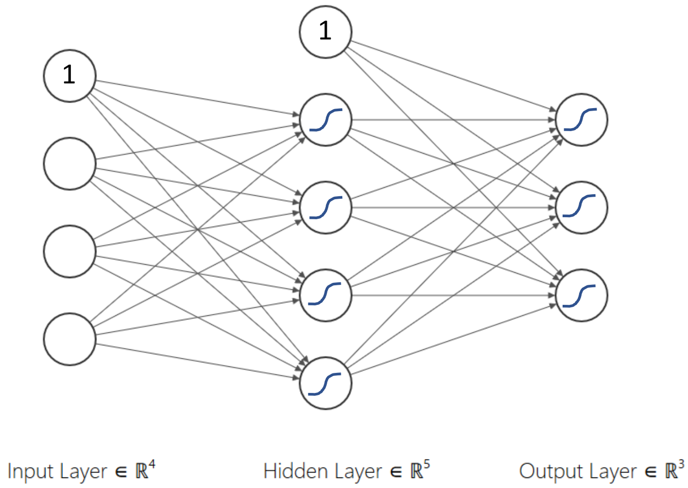
Create weights:

```
width1 = 4                    #number of neurons in hidden layer. Bias not counted
width2 = 1                    #number of neurons in output layer
W0 = np.random.rand(X.shape[1],width1)  #weights from layer 0.
W1 = np.random.rand(width1+1,width2)    #weights from layer 1. Add a neuron for bias.
```

Set max epoc (iterations), learning rate, samples, process data, and plot:

```
itera = 10000
alpha = 0.1
m = X.shape[0]          #number of samples
E = np.zeros(itera,)            #initialize error vector (not needed)

for epoc in range(itera):
    Z1 = np.dot(X,W0)       #1st layer
    a1 = 1/(1+np.exp(-Z1))      #sigmoid activation function of hidden layer
    a1 = np.hstack((np.ones((m,1)),a1)) #add a column for bias calc
    Z2 = np.dot(a1,W1)
    a2 = 1/(1+np.exp(-Z2))      #sigmoid activation function of output layer
    Yhat = a2               #nnet output
    d = Yhat - Y            #delta
    E[epoc] = np.sum (0.5*(d**2))   #MSE (not needed)

    g1 = a1*(1-a1)          #sigm derivative of layer 1
    g1[:,0] = 1         #gradient of bias
    g2 = a2*(1-a2)          #sigm derivative of layer 2

    dEdW1 = np.dot(a1.T, d  * g2)
    dEdW0 = np.dot( X.T, g1[:,1:] * np.dot(d * g2 , W1[1:,:].T))

    W0 -= alpha/m*dEdW0             #update weights
    W1 -= alpha/m*dEdW1

print(np.round(Yhat))
plt.plot(range(epoc+1),E)
```

Example 3: Create NNet(3,4,3) : 1 Hidden Layer, Sigmoid Activation, 3-Class Classification via One-Hot Encoding



Create 3-class data:

```
class1 = np.random.randn(20,3) + np.array([2,2,2])
class2 = np.random.randn(20,3) + np.array([4,4,4])
class3 = np.random.randn(20,3) + np.array([6,6,6])

X1 = np.hstack(((np.ones((20,1)),class1)))              #20x4 for class 1
X2 = np.hstack(((np.ones((20,1)),class2)))              #20x4 for class 2
X3 = np.hstack(((np.ones((20,1)),class3)))              #20x4 for class 3

X = np.vstack ((X1,X2,X3))                  #60x4 combined samples
Y = np.vstack((np.zeros((20,1))+([1,0,0]),
                 np.zeros((20,1))+([0,1,0]),
                 np.zeros((20,1))+([0,0,1])))
```

Scatter plot:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:20,1],X[:20,2],X[:20,3],marker='o')
ax.scatter(X[20:40,1],X[20:40,2],X[20:40,3],marker='^')
ax.scatter(X[40:,1],X[40:,2],X[40:,3],marker='*')
```

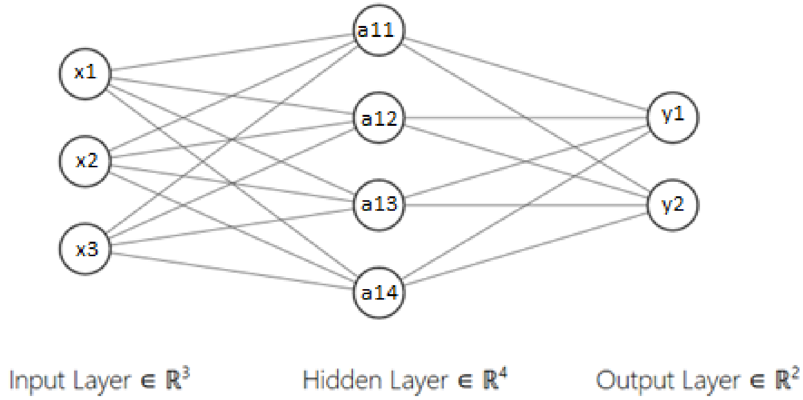
Define weights:

```
width1 = 4
width2 = 3
W0 = np.random.rand(X.shape[1],width1)
W1 = np.random.rand(width1+1,width2)
```

Process data:

```
itera = 10000
alpha = 0.1
m = X.shape[0]          #number of samples
E = np.zeros(itera,)            #initialize error vector (not needed)

for epoc in range(itera):
    Z1 = np.dot(X,W0)           #1st layer
    a1 = 1/(1+np.exp(-Z1))      #sigmoid activation function of hidden layer
    a1 = np.hstack((np.ones((m,1)),a1))    #bias
    Z2 = np.dot(a1,W1)
    a2 = 1/(1+np.exp(-Z2))      #sigmoid activation function of output layer
    Yhat = a2               #nnet output
    d = Yhat - Y                #delta
    E[epoc] = np.sum (0.5*(d**2))   #MSE (not needed)

    g1 = a1*(1-a1)          #sigm derivative of layer 1
    g1[:,0] = 1         #gradient of bias
    g2 = a2*(1-a2)          #sigm derivative of layer 2

    dEdW1 = np.dot(a1.T, d  * g2)
    dEdW0 = np.dot( X.T, g1[:,1:] * np.dot(d * g2 , W1[1:,:].T))

    W0 -= alpha/m*dEdW0             #update weights
    W1 -= alpha/m*dEdW1

print(np.round(Yhat))               #display results
plt.plot(range(epoc+1),E)
```

Example 4: Create NNet(3,4,3) : 1 Hidden Layer, Sigmoid Activation in hidden, Softmax Output, 2-Class Classification (converges better with grad-descent) via One-Hot Encoding



Redefine data without the ones vectors embedded into X:

```
class1 = np.random.randn(20,3) + np.array([2,2,2])      #20x3
class2 = np.random.randn(20,3) + np.array([4,4,4])

X = np.vstack ((class1,class2))         #40x2 combined samples
Y = np.vstack((np.zeros((20,1))+([1,0]),
                 np.zeros((20,1))+([0,1])))
```

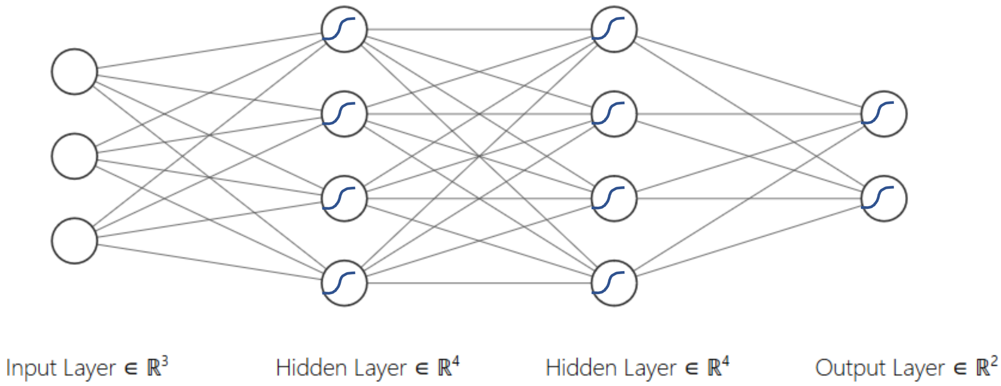
Define weights and biases:

```
width0 = 3                  # input layer neurons
width1 = 4                  # hidden layer neurons
width2 = 2                  # output layer neurons

W0 = np.random.randn(width0, width1)
b0 = np.zeros((1, width1))
W1 = np.random.randn(width1, width2)
b1 = np.zeros((1, width2))
```

Process data:

```
itera = 500
alpha = 0.2
m = X.shape[0]          #number of samples
E = np.zeros(itera,)            #initialize error vector (not needed)
L = np.zeros(itera,)            #log likelihood, aka loss (also not needed)

for epoc in range(itera):
    Z1 = np.dot(X,W0)+b0        #1st layer (output)
    a1 = 1/(1+np.exp(-Z1))      #sigmoid activation function of hidden layer
    Z2 = np.dot(a1,W1)+b1
    expo = np.exp(Z2-np.max(Z2))    #softmax numerator. subtract to stabilize.
    a2 = expo/ np.sum(expo, axis=1, keepdims=True)      #softmax activation.

    Yhat = a2               #nnet output
    d = Yhat - Y            #delta
    E[epoc] = np.sum (0.5*(d**2))   #MSE (not needed)
    L[epoc] = -np.sum(np.log(np.sum(a2*Y,axis=1)))/m  #loss

    g1 = a1*(1-a1)          #sigm derivative of layer 1
    g2 = 1              #deriv of softmax = Yhat-Y
                        #alternatively g2 = deriv_softmax(a2, 20)

    dLdW1 = np.dot(a1.T, d  * g2)
    dLdb1 = np.sum(d * g2, axis=0, keepdims=True)   #same as: np.dot(np.ones((m,1)).T, d  * g2))

    dLdW0 = np.dot( X.T, g1 * np.dot(d * g2 , W1.T))
    dLdb0 = np.sum(np.dot (d*g2, W1.T) * g1, axis=0, keepdims=True)
#same as: np.dot( np.ones((m,1)).T, g1 * np.dot(d * g2 , W1.T))

    W0 += -alpha/m * dLdW0
    b0 += -alpha/m * dLdb0
    W1 += -alpha/m * dLdW1
    b1 += -alpha/m * dLdb1

    print("iter:", epoc, " success,class1:", np.sum(Yhat[0:20,0]>0.5,axis=0),
          ", class 2: ",np.sum(Yhat[20:,1]>0.5,axis=0))                      #mini-report

print(np.argmax(Yhat,axis=1))               #display results
plt.plot(range(epoc+1),E)
```

Example 5: Create NNet(3,4,4,2) : 2 Hidden Layers w/ Sigmoid Activation, 2-Class Classification via One-Hot Encoding



Define weights: Same data X,Y as before (3 features, 2 classes, one-hot encoding).

```
width0 = 3                    # input layer neurons
width1 = 4                    # hidden layer 1 neurons
width2 = 4                    # hidden layer 2 neurons
width3 = 2                    # output layer neurons

W0 = np.random.randn(width0, width1)
b0 = np.zeros((1, width1))
W1 = np.random.randn(width1, width2)
b1 = np.zeros((1, width2))
W2 = np.random.randn(width2, width3)
b2 = np.zeros((1, width3))
```

Process data:

```
itera = 10000
alpha = 0.05
m = X.shape[0]          #number of samples
E = np.zeros(itera,)            #initialize error vector (not needed)

for epoc in range(itera):
    Z1 = np.dot(X,W0)+b0        #1st layer (output)
    a1 = 1/(1+np.exp(-Z1))      #sigmoid activation of hidden layer 1
    Z2 = np.dot(a1,W1)+b1
    a2 = 1/(1+np.exp(-Z2))      #sigmoid activation of hidden layer 2
    Z3 = np.dot(a2,W2)+b2
    a3 = 1/(1+np.exp(-Z3))      #sigmoid activation of output, layer 3

    Yhat = a3               #nnet output
    d = Yhat - Y            #delta
    E[epoc] = np.sum (0.5*(d**2))   #MSE (not needed)

    g1 = a1*(1-a1)          #sigm derivative of layer 1
    g2 = a2*(1-a2)
    g3 = a3*(1-a3)

    dEda3 = d * g3
    dEda2 = np.dot(dEda3 , W2.T) * g2
    dEda1 = np.dot(dEda2 , W1.T) * g1

    dEdW2 = np.dot(a2.T, dEda3)
    dEdb2 = np.sum(dEda3, axis=0, keepdims=True)

    dEdW1 = np.dot( a1.T,dEda2)
    dEdb1 = np.sum(dEda2, axis=0)

    dEdW0 = np.dot( X.T, dEda1)
    dEdb0 = np.sum(dEda1, axis=0)

    W0 += -alpha/m * dEdW0
```

```
    b0 += -alpha/m * dEdb0
    W1 += -alpha/m * dEdW1
    b1 += -alpha/m * dEdb1
    W2 += -alpha/m * dEdW2
    b2 += -alpha/m * dEdb2


print(np.round(Yhat))                  #display results
plt.plot(range(epoc+1),E)
```