Implementing Neural Net with back propagation

dataset: Iris.csv

This is the "Iris" dataset. Originally published at UCI Machine Learning Repository: Iris Data Set, this small dataset from 1936 is often used for testing out machine learning algorithms and visualizations (for example, Scatter Plot). Each row of the table represents an iris flower, including its species and dimensions of its botanical parts, sepal and petal, in centimeters.

In [0]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
```

In [0]:
```
iris = pd.read_csv("iris.csv")
```

In [0]:
```
x = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
x = np.array(x)
x[:5]
```

Out[0]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

In [0]:
```
iris.corr()
```

Out[0]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

In [0]:
```
one_hot_encoder = OneHotEncoder(sparse=False)
y = iris.species
y = one_hot_encoder.fit_transform(np.array(y).reshape(-1, 1))
y[:5]
```

Out[0]:
```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

In [0]:
```
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.15)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=
0.1)
```

In [0]:
```
def NeuralNetwork(X_train, Y_train, X_val=None, Y_val=None, epochs=10, nodes=
[], lr=0.15):
    hidden_layers = len(nodes) - 1
    weights = InitializeWeights(nodes)

    for epoch in range(1, epochs+1):
        weights = Train(X_train, Y_train, lr, weights)

        if(epoch % 20 == 0):
            print("Epoch {}".format(epoch))
            print("Training Accuracy:{}".format(Accuracy(X_train, Y_train, wei
ghts)))
            if X_val.any():
                print("Validation Accuracy:{}".format(Accuracy(X_val, Y_val, w
eights)))

    return weights
```

In [0]:
```
def InitializeWeights(nodes):
    """Initialize weights with random values in [-1, 1] (including bias)"""
    layers, weights = len(nodes), []

    for i in range(1, layers):
        w = [[np.random.uniform(-1, 1) for k in range(nodes[i-1] + 1)]
              for j in range(nodes[i])]
        weights.append(np.matrix(w))

    return weights
```

In [0]:
```
def ForwardPropagation(x, weights, layers):
    activations, layer_input = [x], x
    for j in range(layers):
        activation = Sigmoid(np.dot(layer_input, weights[j].T))
        activations.append(activation)
        layer_input = np.append(1, activation) # Augment with bias

    return activations
```

In [0]:
```
def BackPropagation(y, activations, weights, layers):
    outputFinal = activations[-1]
    error = np.matrix(y - outputFinal)

    for j in range(layers, 0, -1):
        currActivation = activations[j]

        if(j > 1):
            prevActivation = np.append(1, activations[j-1])
        else:

            prevActivation = activations[0]

        delta = np.multiply(error, SigmoidDerivative(currActivation))
        weights[j-1] += lr * np.multiply(delta.T, prevActivation)

        w = np.delete(weights[j-1], [0], axis=1)
        error = np.dot(delta, w)

    return weights
```

In [0]:
```
def Train(X, Y, lr, weights):
    layers = len(weights)
    for i in range(len(X)):
        x, y = X[i], Y[i]
        x = np.matrix(np.append(1, x)) # Augment feature vector

        activations = ForwardPropagation(x, weights, layers)
        weights = BackPropagation(y, activations, weights, layers)

    return weights
```

In [0]:
```
def Sigmoid(x):
    return 1 / (1 + np.exp(-x))

def SigmoidDerivative(x):
    return np.multiply(x, 1-x)
```

In [0]:
```
def Predict(item, weights):
    layers = len(weights)
    item = np.append(1, item)

    #Forward Propagation
    activations = ForwardPropagation(item, weights, layers)

    outputFinal = activations[-1].A1
    index = FindMaxActivation(outputFinal)
    y = [0 for i in range(len(outputFinal))]
    y[index] = 1
    return y


def FindMaxActivation(output):
    m, index = output[0], 0
    for i in range(1, len(output)):
        if(output[i] > m):
            m, index = output[i], i

    return index
```

In [0]:
```
def Accuracy(X, Y, weights):
    correct = 0

    for i in range(len(X)):
        x, y = X[i], list(Y[i])
        guess = Predict(x, weights)

        if(y == guess):
            correct += 1

    return correct / len(X)
```

In [0]:
```
f = len(x[0])
o = len(y[0])

layers = [f, 5, 10, o]
lr, epochs = 0.15, 100

weights = NeuralNetwork(X_train, Y_train, X_val, Y_val, epochs=epochs, nodes=l
ayers, lr=lr);
```

```
Epoch 20
Training Accuracy:0.6842105263157895
Validation Accuracy:0.7692307692307693
Epoch 40
Training Accuracy:0.8333333333333334
Validation Accuracy:0.9230769230769231
Epoch 60
Training Accuracy:0.9210526315789473
Validation Accuracy:1.0
Epoch 80
Training Accuracy:0.9385964912280702
Validation Accuracy:1.0
Epoch 100
Training Accuracy:0.9035087719298246
Validation Accuracy:1.0
```

In [0]:
```
print("Testing Accuracy: {}".format(Accuracy(X_test, Y_test, weights)))
```

```
Testing Accuracy: 0.9130434782608695
```

In [0]:
```

```