Name:KALYANAM VENKATA SREE SAI

ID:2000030439

SEC:01

CourseCode:20cs3026RA

In [10]:
```
import numpy as np
import pandas as pd
```

In [14]:
```
iris = pd.read_csv("Iris.csv")
iris = iris.sample(frac=1).reset_index(drop=True) # Shuffle
```

In [15]:
```
X = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
X = np.array(X)
X[:5]
```

Out[15]:
```
array([[6.1, 2.8, 4. , 1.3],
       [5.5, 2.5, 4. , 1.3],
       [6.7, 3.3, 5.7, 2.1],
       [6.5, 3. , 5.5, 1.8],
       [6.7, 3.1, 4.4, 1.4]])
```

In [16]:
```
from sklearn.preprocessing import OneHotEncoder
one_hot_encoder = OneHotEncoder(sparse=False)

Y = iris.Species
Y = one_hot_encoder.fit_transform(np.array(Y).reshape(-1, 1))
Y[:5]
```

Out[16]:
```
array([[0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.]])
```

In [17]:
```
from sklearn.model_selection import train_test_split
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 [18]:
```
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, weights)))
            if X_val.any():
                print("Validation Accuracy:{}".format(Accuracy(X_val, Y_val, weights)))

    return weights
```

In [19]:
```python
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 [20]:
```python
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 [21]:
```python
def BackPropagation(y, activations, weights, layers):
    outputFinal = activations[-1]
    error = np.matrix(y - outputFinal) # Error at output

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

        if(j > 1):
            # Augment previous activation
            prevActivation = np.append(1, activations[j-1])
        else:
            # First hidden layer, prevActivation is input (without bias)
            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) # Remove bias from weights
        error = np.dot(delta, w) # Calculate error for current layer

    return weights
```

In [22]:
```python
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 [23]:
```python
def Sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

In [24]:
```python
def Predict(item, weights):
    layers = len(weights)
    item = np.append(1, item) # Augment feature vector
```

```python
    ##_Forward Propagation_##
    activations = ForwardPropagation(item, weights, layers)

    outputFinal = activations[-1].A1
    index = FindMaxActivation(outputFinal)

    # Initialize prediction vector to zeros
    y = [0 for i in range(len(outputFinal))]
    y[index] = 1  # Set guessed class to 1

    return y # Return prediction vector


def FindMaxActivation(output):
    """Find max activation in 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 [25]:

```python
def Accuracy(X, Y, weights):
    """Run set through network, find overall accuracy"""
    correct = 0

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

        if(y == guess):
            # Guessed correctly
            correct += 1

    return correct / len(X)
```

In [27]:

```python
f = len(X[0]) # Number of features
o = len(Y[0]) # Number of outputs / classes

layers = [f, 5, 10, o] # Number of nodes in layers
lr, epochs = 0.15, 100

weights = NeuralNetwork(X_train, Y_train, X_val, Y_val, epochs=epochs, nodes=layers, lr=
```

```
Epoch 20
Training Accuracy:0.7192982456140351
Validation Accuracy:0.38461538461538464
Epoch 40
Training Accuracy:0.9473684210526315
Validation Accuracy:1.0
Epoch 60
Training Accuracy:0.9736842105263158
Validation Accuracy:1.0
Epoch 80
Training Accuracy:0.9298245614035088
Validation Accuracy:0.9230769230769231
Epoch 100
Training Accuracy:0.9824561403508771
Validation Accuracy:1.0
```

In [28]:

```python
print("Testing Accuracy: {}".format(Accuracy(X_test, Y_test, weights)))
```

```
Testing Accuracy: 0.9565217391304348
```

```
In [ ]:
```