CODE SUMMARY

For this assignment I am building a backprop network to solve a multi-class classification problem. The goal is to explore how miss-classification costs can be taken into account in training a backprop network.

File names and their objectives:

- hw2_UCI_withCostMatrix: Implements a backprop network incorporating cost matrix
- hw2_UCI_withoutCostMatrix: Computes accuracy without incorporating cost matrix
- hw2_UCI_withKeras_withoutCostMatrix: Achieve highest possible accuracy for UCI dataset. Here I am using the entire dataset.

UCI human activity recognition dataset has been used for testing.

Source: UCI_human_activity_dataset

Due to high computational requirements to train a network using the entire dataset, I have used only 100 random points in the dataset for training and 100 random points for testing. The output labels are walking, walking upstairs, walking downstairs, sitting, standing, laying. Total number of iterations used are 500. Training beyond that point makes our network finely tuned to the training data set and reduces accuracy for test datasets.

I am using tanh as my Activation function. The pre-defined learning rate is set to 0.005 and momentum factor is set to 0.01. I have selected 60 attributes out of 561 which means input layer has 60 nodes. Hidden layer has 20 nodes and output layer has 6 nodes.

COST MATRIX:

The technique used here is **rebalancing of weights** to account for costs of misclassification during training. I have defined the cost matrix such that if walking is correctly classified the value returned is 0, whereas if wrongly classified as sitting/laying/standing I return some excessive cost value. Same is true for misclassifying sitting as walking/walking upstairs/walking downstairs. This matrix can be modified as per the requirements.

```
38
39  costmatrix = []
40  costmatrix.append([0,1,2,6,7,5])
41  costmatrix.append([1,0,2,3,4,3])
42  costmatrix.append([1,3,0,3,4,5])
43  costmatrix.append([2,3,2,0,2,1])
44  costmatrix.append([2,3,4,1,0,1])
45  costmatrix.append([3,4,2,2,1,0])
46
```

EXPLANATION:

At line 110, I am adding the cost of misclassifying using the defined cost matrix. This takes place in hw2_UCI_withCostMatrix.py. I am increasing the error value at the output layer which in turn affects the weight values as well. Here the correct value is the index of the actual output. For example, correct = 2 for 001000. During back propagation the max value will be the prediction. Actual is calculated by finding the index containing the max value. If these are found to be same the error value passed back is not modified else it is modified by a factor of the corresponding cost matrix values. We will find in our results how this modification affects the results.

```
    # backpropogate to update weights
    def backPropagate(self, targets, N, M,confustionmatrx):
        output_deltas = [0.0] * self.no
        correct = (targets.index(1))
        actual = (self.ao).index(max(self.ao))
        for k in range(self.no):
            error = (targets[k]-self.ao[k])
            if(k==actual):
                confustionmatrx[correct][actual] += 1
                error = error + error*costmatrix[correct][actual]
                # Here I am incorporating the cost by referring this cost matrix
                # These values are asymmetric
                # We notice a rebalancing of weights due to account for the cost
                # This implementation can be quite useful is some scenarios

            output_deltas[k] = dfunc_tan(self.ao[k]) * error
```

We also make use of a confusion matrix which keeps track of the number of misclassifications. At each step we are incrementing confusionmatrix[i][j] by 1 if i is misclassified as j. If we look at the output in the Cost_with_CostMatrx.txt file we note a decrease in the misclassification cost as our models tries to adapt and is nudged towards the right direction by increasing the error value as mentioned above.

```
for i in range(iterations):
    # Keeps track of the number of missclassifications that occur
    confustionmatrx = []
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    error = 0.0
    for p in patterns:
        inputs = p[0]
        targets = p[1]
        self.update(inputs)
        error = error + self.backPropagate(targets, N, M,confustionmatrx)
        current = self.calc_cost(confustionmatrx,costmatrix)
    if(last!=current and last>current):
        last = current
        print('*********************************************************************************')
        print('Confusion Matrix: ',confustionmatrx)
        print('Missclassification cost while training:',self.calc_cost(confustionmatrx,costmatrix))
        print('Error at iteration',i,':',error)
        print('*********************************************************************************')
```

TESTING OUR MODELS:

This function is used for testing our model. After training our network we calculate the total misclassification cost. We note that the misclassification cost decreases if we account for the cost matrix in line 110 as mentioned above.

```
# Test my model
def test(self, patterns):
    count = 0
    cost = 0
    confustionmatrx = []
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    confustionmatrx.append([0,0,0,0,0,0])
    for p in patterns:
        target = (p[1].index(1))
        result = self.update(p[0])
        index = result.index(max(result))
        count += (target == index)
        if(target!=index):
            confustionmatrx[target][index] += 1

    accuracy = float(count/len(patterns))
    print('accuracy: %-.9f' % (accuracy*100))
    return self.calc_cost(confustionmatrx,costmatrix)
```

CALCULATING COST:

This function finds the total misclassification cost using the confusion and cost matrix.

```
def calc_cost(self,confustionmatrx,costmatrix):
    cost = 0
    for i in range(5):
        for j in range(5):
            cost += confustionmatrx[i][j]*costmatrix[i][j]
    return cost
```

RESULT ANALYSIS:

hw2_UCI_withoutCostMatrix.py:

- If we look at the result in Cost_normal.txt for training without using a cost matrix, we can see that when we train our model without incorporating the cost matrix at line 110 and test it on the test dataset, the **cost of misclassification is 33**. We note that our training is successful because of the decrease in error values and an accuracy of 85%.

hw2_UCI_withCostMatrix.py:

- Now when rebalancing our weights by accounting for the cost matrix at line 110, we will find that training is successful and when we test our model on the test data, **misclassification cost is found to be 24.** Hence, we note that accounting for the costs has rebalanced the weights and nudged the model to account for misclassification cost. These results are consistent and found to be true when we test our model on a separate set of points. The accuracy of this model is still 85%.

METHOD DESCRIPTION:

- **update**: Helper function to update the input values
- **backpropagate**: Helper function to update the weight values using backpropagation
- **train**: This is our training function where we defined the total number of iterations, learning rate and momentum factor.
- **test**: This function is used for testing our model and finding its accuracy
- **dependencies**: Numpy and Pandas

Sample Output for finding the misclassification cost when incorporating the cost matrix. The full output is available in Cost_withCostmtrix.txt

```
********************************************************************************
Confusion Matrix:  [[9, 1, 0, 0, 2, 0], [10, 2, 0, 0, 0, 0], [6, 3, 0, 0, 0, 1], [14, 6, 0, 4, 4, 0], [4, 19, 0, 1, 0, 0], [2, 2, 0, 5, 4, 1]]
Missclassification cost while training: 160
Error at iteration 0 : 20.49936281647073
********************************************************************************
********************************************************************************
Confusion Matrix:  [[7, 5, 0, 0, 0, 0], [0, 12, 0, 0, 0, 0], [0, 6, 4, 0, 0, 0], [0, 3, 0, 25, 0, 0], [0, 2, 0, 0, 22, 0], [0, 1, 0, 0, 0, 13]]
Missclassification cost while training: 38
Error at iteration 367 : 1.6037053814944529
********************************************************************************
********************************************************************************
Confusion Matrix:  [[12, 0, 0, 0, 0, 0], [0, 11, 0, 0, 1, 0], [0, 2, 8, 0, 0, 0], [0, 1, 0, 27, 0, 0], [0, 0, 0, 2, 22, 0], [0, 0, 0, 0, 0, 14]]
Missclassification cost while training: 15
Error at iteration 368 : 2.0234735346631
********************************************************************************
********************************************************************************
Confusion Matrix:  [[12, 0, 0, 0, 0, 0], [0, 11, 1, 0, 0, 0], [0, 1, 9, 0, 0, 0], [0, 1, 0, 27, 0, 0], [0, 2, 0, 0, 22, 0], [0, 0, 0, 0, 0, 14]]
Missclassification cost while training: 14
Error at iteration 369 : 1.5837207697262297
********************************************************************************
********************************************************************************
Confusion Matrix:  [[12, 0, 0, 0, 0, 0], [0, 12, 0, 0, 0, 0], [0, 0, 10, 0, 0, 0], [0, 0, 0, 27, 1, 0], [0, 1, 0, 1, 22, 0], [0, 0, 0, 0, 0, 14]]
Missclassification cost while training: 6
Error at iteration 370 : 0.8957958246011979
********************************************************************************
********************************************************************************
Confusion Matrix:  [[12, 0, 0, 0, 0, 0], [0, 12, 0, 0, 0, 0], [0, 0, 10, 0, 0, 0], [0, 0, 0, 28, 0, 0], [0, 0, 0, 0, 24, 0], [0, 0, 0, 0, 0, 14]]
Missclassification cost while training: 0
Error at iteration 371 : 0.6624361113279589
********************************************************************************


---------- Training finished at:  2017-10-19 17:09:31 ----------
-----
accuracy: 85.000000000
Missclassification cost on the Test Data:  24
```