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 with a cost matrix
- hw2_UCI_withoutCostMatrix: Computes accuracy without using a 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.

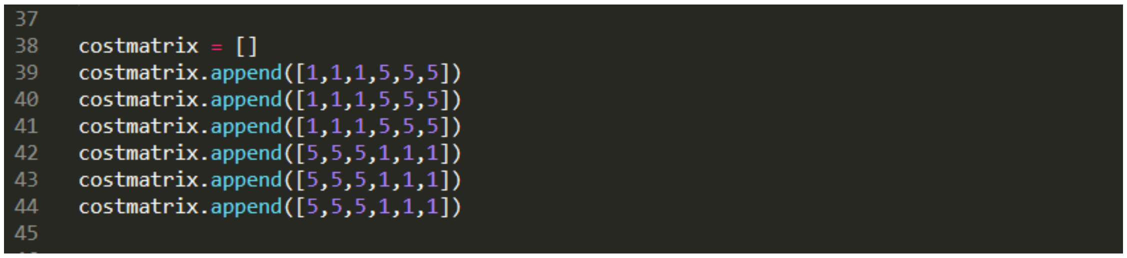
Source: UCI_human_activity_dataset

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 2000.

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.

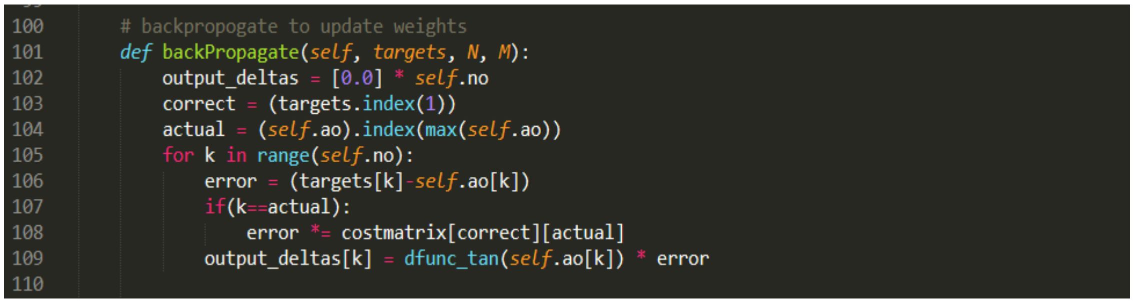
COST MATRIX:

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 classified as walking the value returned is 1, whereas if wrongly classified as sitting/laying/standing I return 5. Same is true for misclassifying sitting as walking/walking upstairs/walking downstairs.



```
costmatrix = []
costmatrix.append([1,1,1,5,5,5])
costmatrix.append([1,1,1,5,5,5])
costmatrix.append([1,1,1,5,5,5])
costmatrix.append([5,5,5,1,1,1])
costmatrix.append([5,5,5,1,1,1])
costmatrix.append([5,5,5,1,1,1])
```

At line 108, I am adding the cost of misclassifying using the defined cost matrix. This takes place in hw2_UCI_withCostMatrix.py. I am modifying the error values which in turn affects the weight values at line 120 and 121.



```
100     # backpropogate to update weights
101     def backPropagate(self, targets, N, M):
102         output_deltas = [0.0] * self.no
103         correct = (targets.index(1))
104         actual = (self.ao).index(max(self.ao))
105         for k in range(self.no):
106             error = (targets[k]-self.ao[k])
107             if(k==actual):
108                 error *= costmatrix[correct][actual]
109             output_deltas[k] = dfunc_tan(self.ao[k]) * error
110
```

RESULT ANALYSIS:

- If we look at the result in output.txt for training without using a cost matrix, we can see the error loss value decrease after every iteration and is finally set to 0.039. This means our training is successful and for testing our network we use new data. Accuracy was found to be 82%
- Now when using a cost matrix for accounting misclassification costs while training our network we notice a slight increase in accuracy to 83%. This implies that handling cost-sensitive classification problems in such a manner might produce better results for cost-sensitive datasets.

FUTURE WORK:

The effect of using a cost matrix is apparent but we need to further tune this matrix for even better results. Although accuracy rate is still low compared to the Keras implementation but that is a matter of increasing the number of layers in our network. I intend to work on this further by adding more layers and tuning the cost matrix.

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