To run:

```
$ g++ mlp.cpp -o mlp
$ ./mlp
```

Input Format:

N	Number of dimensions of Input Data
M	Number of classes
L	Number of training samples
TestL	Number of testing samples
H	Number of hidden layers
A	Activation function to be used - 0 for sigmoid, 1 for ReLU
Filename	Name of file with training and testing data

File with training and testing data (Filename):

```
x11 x12 x13 … x1N class1
x21 x22 x23 … x2N class2
  .   .   .  …  .    .
  .   .   .  …  .    .
xL1 xL2 xL3 … xLN classL
y11 y12 y13 … y1N class1
y21 y22 y23 … y2N class2
  .   .   .  …  .    .
  .   .   .  …  .    .
yTestL1 yTestL2 yTestL3 … yTestLN classTestL
```

Output Format:

```
Total Loss for Epoch 1 = L1
Total Loss for Epoch 2 = L2
          .
          .
Total Loss for Epoch 1000 = L1000
p11 p12 p13 … p1M
Predicted Class = x1 Actual Class = y1
p21 p22 p23 … p2M
Predicted Class = x2 Actual Class = y2
          .
          .
pTestL1 pTestL2 pTestL3 … pTestLM
Predicted Class = xTestL Actual Class = yTestL
Accuracy = Z%
```

The Classes labels are 0-indexed integers. Input points can be double floating point values.

Implementation:
After taking input from the user and storing the training dataset and testing dataset, the datasets are normalized using the min-max method. This can also be modified to standardize the data instead of normalizing, by calculating the mean and variance of the dataset for each dimension. The testing dataset and training dataset are normalized separately.

Following this, the weights for each layer of neurons are initialized according to the activation function selected. For sigmoid function, Normalized Xavier Distribution is used to randomly assign values to the weights. The range for the weights is defined as [-Sqrt(6/(X+Y)), Sqrt(6/(X+Y))], where X is the number of nodes in the previous layer, and Y is the number of nodes in the current layer. For the ReLU activation function, He Distribution is used to randomly select weight values. The range of the weights in [0, Sqrt(2/X)], where X is the number of nodes in the previous layer.

After initialization of weights, the MLP is trained using backpropagation and Stochastic Gradient Descent. In each epoch, the MLP takes as input each point in the training set and calculates the loss. Since the activation function in the output layer is the Softmax function, the loss function used is the Cross Entropy Softmax Loss Function, $L =- \sum_{1}^{M} y_i * ln(p_i)$, where y_i is the target output for the input, and p_i is the calculated output for the output. y is a 1-hot vector. Next the backpropagation algorithm is run, where Gradient for each weight in the network is calculated w.r.t. the loss function. The formula used is $\delta^L = (((W^{L+1})^T * \delta^{L+1}) \odot \sigma'(z^L)$where z^L is the weighted sum of the outputs of the neurons in the previous layer, delta is the derivative of the loss function w.r.t the weighted sum in each neuron, sigma is the activation function used and W is the weight matrix. To calculate the derivative w.r.t. some weight, $\frac{\partial L}{\partial W_{ij}^L} = a_j^{L-1} * \delta_i^L$. Using the gradient for each weight, the weights are updated by subtracting the product of gradient and the learning rate from the current weight. The learning rate is fixed but can be changed and the program can be rerun to check the effects of the change. The number of epochs are also fixed but can be changed and the program can be rerun to see the change. Also the number of neurons in the hidden layers are fixed as well and can be changed and the program can be rerun to note the effects of such a change.

In each epoch, the total loss is calculated by calculating the loss for each input data point. Finally, after the training is over, the test dataset is fed forward and the network outputs the probability of each class for a particular input (output of the Softmax function). The class with the highest probability is selected as the predicted output and compared to the actual output. After running over all points in the test set, the accuracy of the network is calculated and output as a percentage.

Results:
The accuracy of the network was calculated over a few datasets with varying number of hidden layers

Iris Dataset:
- Training set size = 100
- Test set size = 50
- Input dimensions = 4
- Classes = 3
  - Activation Function: Sigmoid + Softmax
    - 0 Hidden layers: 86% accuracy
    - 1 Hidden layer: 94% accuracy
    - 2 Hidden layers: 90% accuracy
  - Activation Function: ReLU + Softmax
    - 0 Hidden layers: 86% accuracy
    - 1 Hidden layer: 32% accuracy
    - 2 Hidden layers: 32% accuracy

Seeds Dataset:
- Training set size = 150
- Test set size = 60
- Input dimensions = 7
- Classes = 3
  - Activation Function: Sigmoid + Softmax
    - 0 Hidden layers: 90% accuracy
    - 1 Hidden layer: 91.667% accuracy
    - 2 Hidden layers: 90% accuracy
  - Activation Function: ReLU + Softmax
    - 0 Hidden layers: 90% accuracy
    - 1 Hidden layer: 28.333% accuracy
    - 2 Hidden layers: 28.333% accuracy

Wine Dataset:
- Training set size = 4000
- Test set size = 898
- Input dimensions = 11
- Classes = 11
  - Activation Function: Sigmoid + Softmax
    - 0 Hidden layers: 32.739% accuracy
  - Activation Function: ReLU + Softmax
    - 0 Hidden layers: 32.739% accuracy

XOR Dataset:
- Training set size = 4

Test set size = 4
Input dimensions = 2
Classes = 2

- Activation Function: Sigmoid + Softmax
  - 0 Hidden layers: 25% accuracy
  - 1 Hidden layer: 100% accuracy
  - 2 Hidden layers: 100% accuracy
- Activation Function: ReLU + Softmax
  - 0 Hidden layers: 25% accuracy
  - 1 Hidden layer: 50% accuracy
  - 2 Hidden layers: 50% accuracy