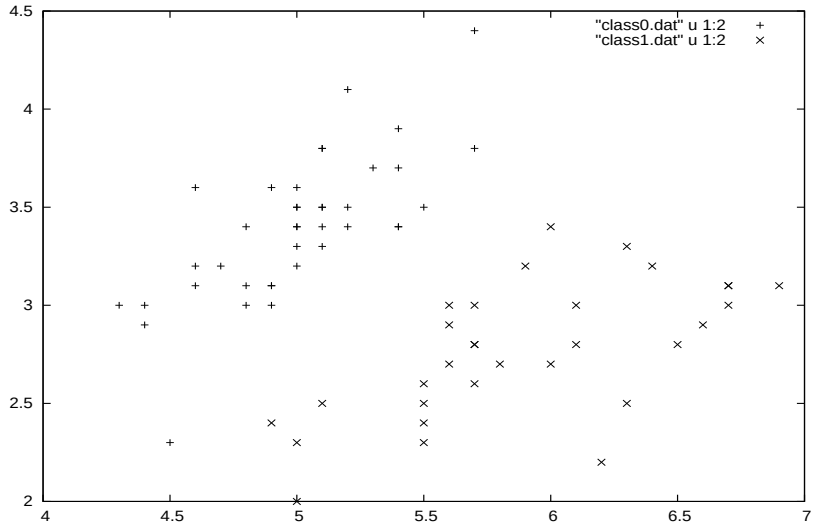
Assignment 2 solution

1.1

I first split the dataset into two. The first file contains all data points belonging to class 1, the second one containing datapoints belonging to class 0. I name this class0.dat and class1.dat respectively. Then I plot these using gnuplot (you can also use EXCEL, OpenOffice, or other plotting tools).

```
grep '1.0000000e+00$' assignment2_train.txt  >class1.dat
grep '0.0000000e+00$' assignment2_train.txt  >class0.dat
gnuplot
gnuplot> plot "class0.dat" u 1:2, "class1.dat" u 1:2
```



Modifies mymlp.c as follows (only three lines needed to be modified, these are highlighted):

```
//#define AAMMODE
#define LINEARMLP

…

void RunForward(struct NET net, struct ENTRY *entry)
{
  int i, j;
  REAL wsum;

  //Input to hidden
  for (i = 0; i < net.hdim; i++){
    wsum = net.hbias[i];
    for (j = 0; j < entry->npoints; j++){
      wsum += net.iweights[i][j] * entry->points[j];
    }
    net.hact[i] = hact(wsum);
  }

#ifdef LINEARMLP
  memcpy(net.hact, entry->points, 2*sizeof(REAL));
#endif

  //Hidden to output
  for (i = 0; i < net.odim; i++){
    net.oout[i] = net.obias[i];
  }
  for (j = 0; j < net.hdim; j++){
    for (i = 0; i < net.odim; i++){
      net.oout[i] += net.hact[j] * net.oweights[j][i];
    }
  }
}

  entries = ReadEntries(argv[1], &nentries);
  net.idim = entries->npoints;
#ifdef LINEARMLP
  net.idim = 2;
#endif
#ifdef AAMMODE
  fprintf(stderr, "Training MLP in AAM mode\n");
#endif
```

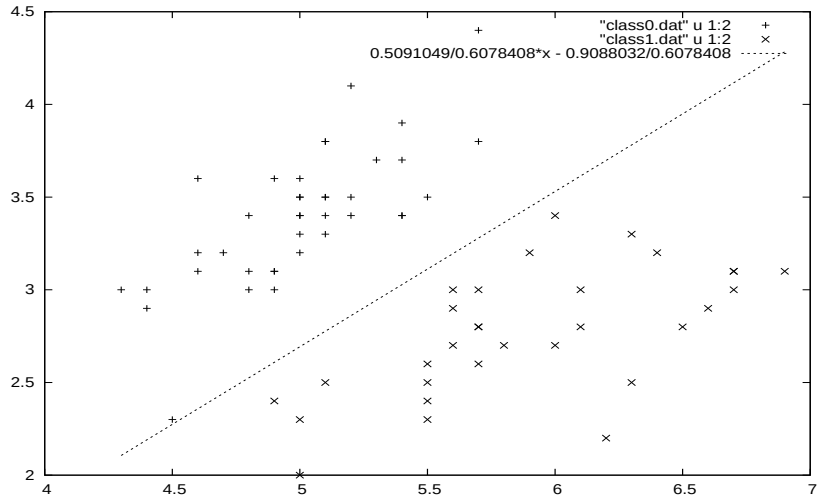
```
  net.odim = entries->ntargets;

…

  Save(1, net, entries, nentries);
  printf("%E %E %E\n", net.oweights[0][0], net.oweights[1][0], net.obias[0]);
```

Now compile the code:
`gcc mymlp.c -o mymlp -lm`

Then train the network:
`mymlp assignment2_train.txt 100000 2 0.00005`

Output produced:
`5.091049E-01 -6.078408E-01 -4.088032E-01`

Plot the result:
`plot "class0.dat" u 1:2, "class1.dat" u 1:2, 0.5091049/0.6078408*x – 0.9088032/0.6078408`



The classes are linearly separable. The MLP found a decision boundary which separates the two classes. Note that the decision boundary is optimal.

1.2

Long term dependency is a problem by which parameters, the further they are away from the output layer, received less and less of the error signal (the error induced gradient becomes smaller and smaller). The problem gets worse exponentially with the distance of a parameter to the output layer. This means that hidden layers “learn” slower than the output layer, and that there is a limit to the length of a sequence or depth of a graph that a recursive or recurrent neural network can learn. Long term dependency is caused by the back-propagation of the gradient descent method. The gradients are “filtered” by the layers as the gradient is propagated back. This means that more and more information about the network error gets lost.

1.3

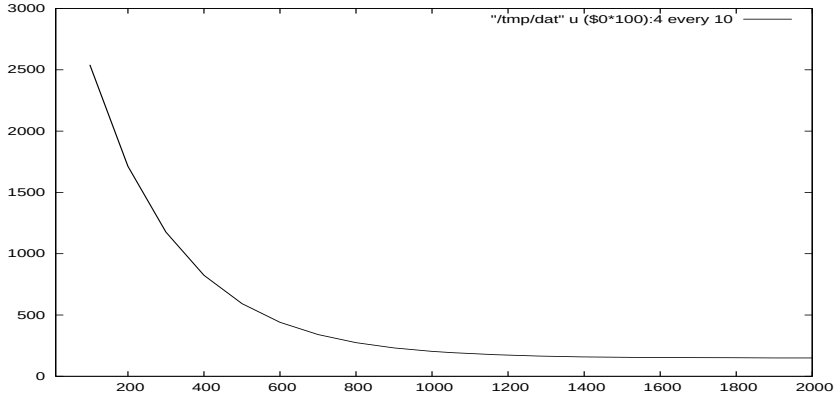
Modify the code as follows:

```
#define AAMMODE
//#define LINEARMLP
```

Then compile the code:
`gcc mymlp.c -o mymlp -lm`

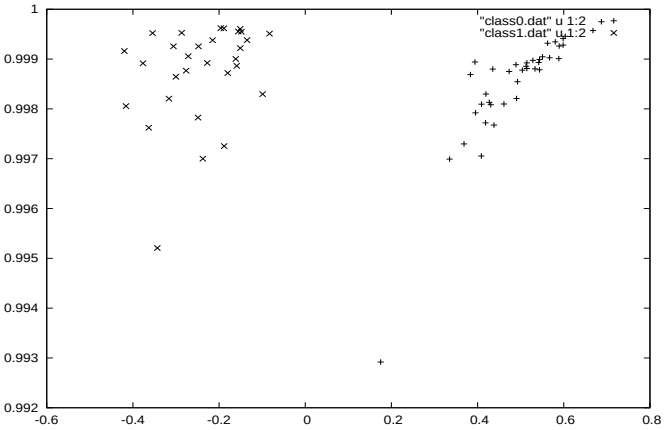
Then train the network:
`mymlp assignment2_train.txt 100000 2 0.00005`

Plot of summed square error:



Smallest SSE was: 145.6302

The output of the hidden layer is saved to a file named hiddenout1.dat. Plotting the content of the file (and indicating class membership so as to determine linear separability) gives the following:



Observation: The projection is clearly linearly separable. However, the vertical value range is very small when compared to the horizontal value range, and when compared to the value range of the original input vectors. The reasons are two-fold: Firstly, the output values of the hidden layer are confined to be within [-1:1]. Secondly, the hidden layer contains no measure to spread the projection points across [-1:1]. It is sufficient to ensure that the projection differentiates data points which are different in the input space, and that this differentiation can be described by a linear function. The value range plays no role since the magnitude of the weights in the output weights can compensate the size of the output in the hidden layer.

2.1

Assume that the data is stored in a file named animals.txt, and that the content of the file is as follows:

13 hexa 10 10 gaussian

```
1 0 0 1 0 0 0 0 1 0 0 1 0 Dove
1 0 0 1 0 0 0 0 1 0 0 0 0 Hen
1 0 0 1 0 0 0 0 1 0 0 0 1 Duck
1 0 0 1 0 0 0 0 1 0 0 1 1 Goose
1 0 0 1 0 0 0 0 1 1 0 1 0 Owl
1 0 0 1 0 0 0 0 1 1 0 1 0 Hawk
0 1 0 1 0 0 0 0 1 1 0 1 0 Eagle
0 1 0 0 1 1 0 0 0 1 0 0 0 Fox
0 1 0 0 1 1 0 0 0 0 1 0 0 Dog
0 1 0 0 1 1 0 1 0 1 1 0 0 Wolf
1 0 0 0 1 1 0 0 0 1 0 0 0 Cat
0 0 1 0 1 1 0 0 0 1 1 0 0 Tiger
0 0 1 0 1 1 0 1 0 1 1 0 0 Lion
0 0 1 0 1 1 1 1 0 0 1 0 0 Horse
0 0 1 0 1 1 1 1 0 0 1 0 0 Zebra
0 0 1 0 1 1 1 0 0 0 0 0 0 Cow
```

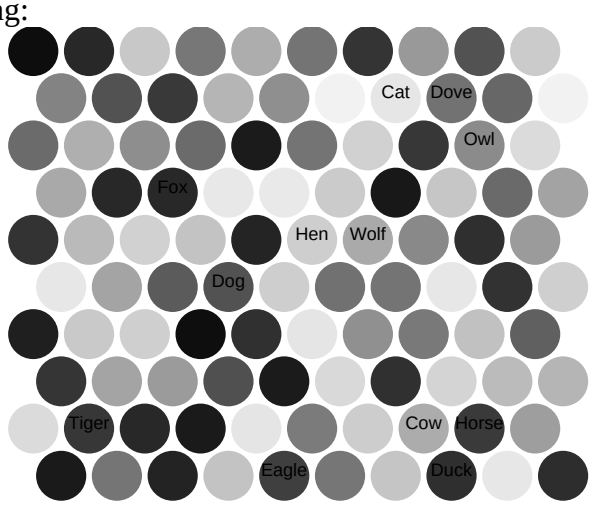
First step: Initialize a new SOM

```
randinit.exe -din animals.txt -cout init.map -xdim 10 -ydim 10 -topol hexa
-neigh gaussian -rand 7
```

Second step plot the random mappings:

```
vcal.exe -din animals.txt -cin init.map -cout tmp.map
planes.exe -cin tmp.map
```

This produced the following:

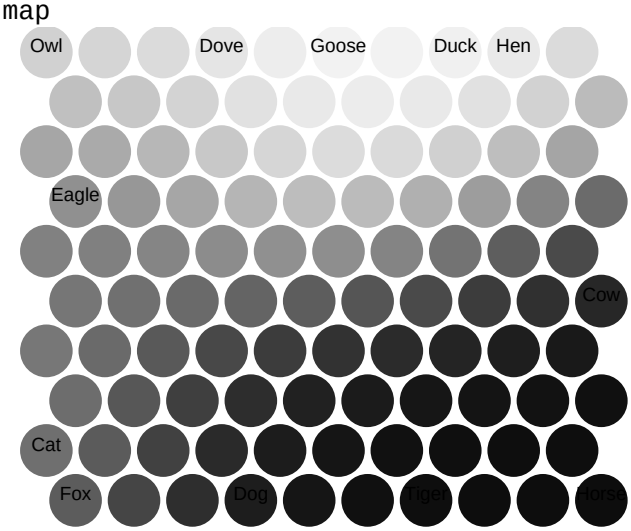


Third, train the map:

```
/vsom.exe -cin init.map -din animals.txt -cout trained.net -rlen 100000 -alpha
0.001 -radius 5
```

Forth, display the trained map:

```
vcal.exe -din animals.txt -cin trained.net -cout tmp.map
planes.exe -cin tmp.map
```



Explanation of the plots: In the plot of the randomly initialized network we find mappings such as Dove and Cat mapped nearby. Hence, in its initial stage, the SOM does not exhibit any topology preserving mapping. After training, animals which are similar in the input vector are mapped to nearby regions. The trained map maps Dove and Cat in distant areas which illustrates that the map has learnt to map similar patterns together while mapping dissimilar patterns far apart. This is a typical property of clustering algorithms, and hence, a SOM is a clustering algorithm which can be used for visualization and dimension reduction of high dimensional data.

Kohonen's result can not be reproduced because of the following: The input vectors for the animals Owl and Hawk as well as for Horse and Zebra are identical. This means that it is impossible that two different codebooks will be activated for Owl and Hawk, and for Horse and Zebra. Thus, Kohonen's result must be wrong.

2.2

Unsupervised learning implies that we do not have any target values available. Hence, there is no mechanism which can be used to evaluate whether a clustering result is any good. Nevertheless, there are a number of evaluation meassures which assume either that

- a target value is available anyway (external index)
- relative indexes are computed which work under the assumption that the clusters are correct and useful. Under this assumption, we compute the “wellness” of this clustering.

Common evaluation meassures:

Intra cluster SSE, Inter cluster SSE, SSE, External index, and relative indexes based on the former.

3.1

The MLP will be quicker. In fact, the MLP will be quicker for almost all learning problems (except very small, practically useless learning problems). This is because the number of operations that need to be carried out is much larger for the SOM. See discussion in 3.2 for more details.

3.2

Both, MLP and SOM exhibit a linear computational complexity. Both MLP and SOM are massively parallel systems. However, the number of operations that need to be carried out is typically much larger (by some constant) for a SOM than for the MLP. The main reason is that the SOM contains N times N codebook (neurons) each of which are of dimension k. In comparison, a MLP contains P neurons each of which is of dimension 1. Thus, in a fair comparison, when P = N * M, the MLP will be at most k times faster. However, the operations that need to be carried out in a SOM are simple additions and multiplications whereas the MLP requires more complex functions (such as tanh or exp) which are more expensive to execute. Nevertheless, the MLP remains the faster algorithm by some constant value.

-END-