MLNC – Assessed Coursework 2: Human Activity Recognition

Objective

In this coursework an MLP is used to classify data on human activity, and its performance then compared with another form of classifier performing the same task. The human activity dataset contains accelerometer data reflecting the average accelerations over a 10-second-interval in 3 dimensions (x,y,z), and a corresponding class of one out of four possibilities: walking, jogging, walking upstairs, walking downstairs.

1 Design of the MLP

a) Learning rate

For this first part of question 1 an MLP is used that employs a gradient descent backpropagation algorithm to train the neural network. Using this algorithm, the effect that the learning rate had on training speed and performance was evaluated. The number of layers and neurons were kept constant at 1 and 5 respectively. Static and dynamic learning rates were compared, and all trained over a constant of 500 epochs.

i. Static

Fig.1.1(a) shows the classification accuracy over 500 training epochs for MLPs with static learning rates of 0.00005, 0.0001, 0.0005, 0.001, 0.005 and 0.01. Generally a trend is seen where an increase in learning rate results in a higher classification accuracy after 500 epochs, however convergence isn't always apparent. Between learning rates of 0.00005-0.001 classification accuracy after 500 epochs increases from roughly 0.42-0.5, with most converging after the number of epochs, or close to converging without any noticeable fluctuations. Above 0.001 (0.005 and 0.01) fluctuations in accuracy are seen, preventing effective convergence. Such rapid fluctuations are often the result of gradient descent overshooting in the optimal direction, suggesting that the learning rate is too high. The smallest and largest magnitudes of learning rate tested (0.00005 and 0.01) produced the lowest classification accuracies after 500 epochs, suggesting that choosing a learning rate within this range is likely to be an effective decision. Finally, Fig.1.1(c) shows the effect that the varied static learning rate had on training speed. This proves to be negligible with a variance of less than 8 seconds between all training speeds.

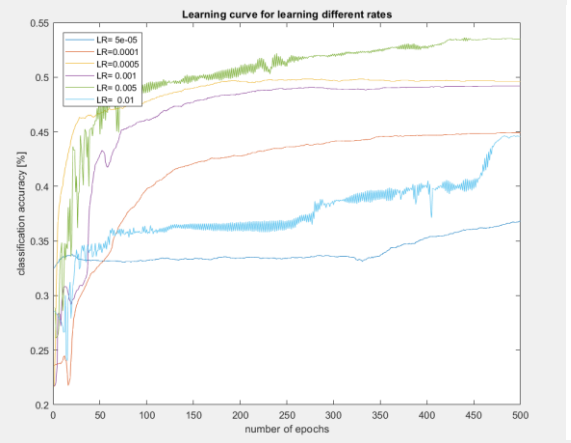
ii. Dynamic

Fig.1.1(b) shows the classification accuracy over 500 training epochs for MLPs with dynamic learning rates. The initial learning rates set at the first epoch were the same as for the static learning rate testing. However, after each epoch the learning rate would decrease by an increment equal to the initial learning rate / 500 (number of epochs). There was also a minimum learning rate of 0.00001, for which it would not fall below. The result shows a similar outcome to the experiments with static learning rates, except for one important difference: By decreasing learning rate over the training process, fluctuations have become less apparent in MLPs with initial learning rates of 0.005 and 0.01, especially towards the end of training. This means convergence now occurs where it previously did not. Fig.1.1(d) shows the variance in training speed. Once again there is a similar pattern to the static learning rate testing, with mostly negligible change in training speed among the different learning rates. For a dynamic learning rate of 0.01 however, time for computation increases to 195s, compared with 168s for 0.005, suggesting 0.01 is too high.

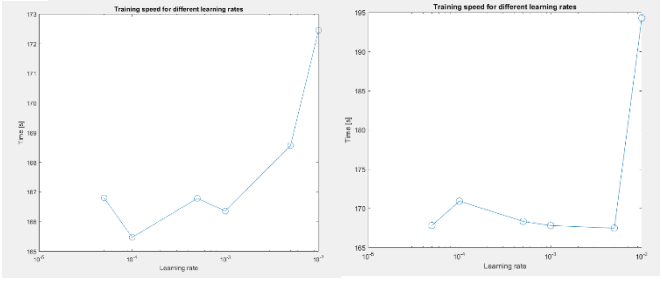
Fig.1.1: Investigations into different static and dynamic learning rates.

(A)

(B)



(C) (D)



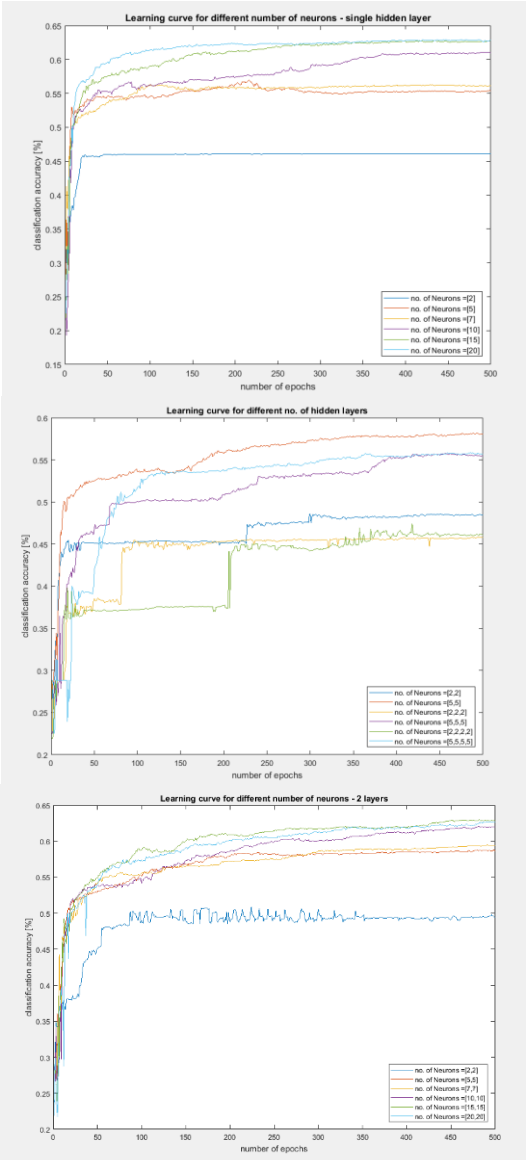
Taking all into consideration the MLP with a dynamic learning rate of 0.005 is probably the most effective choice. It produces the highest classification accuracy, 54%, has a relatively fast training speed, and through changing to a dynamic learning rate also converges after 500 epochs.

b) Network configuration

For the rest of the questions an MLP was used that implements a resilient back propagation algorithm (RPROP). This form of algorithm is not influenced by the learning rate, and therefore the parameter is not taken into consideration for the remainder of the coursework. When investigating different configurations of the neural network, the two parameters to be altered are the number of hidden layers and number of neurons in each hidden layer. There are an infinite amount of possible combinations for these two parameters, and so it is impossible to find the optimum configuration. However given the limited time and computational power a simple approach was used with the following method: First the number of neurons in a single hidden layer was altered and the resulting classification accuracies recorded. Fig.1.2(a) shows the classification accuracy over 500 epochs for a single hidden layer network with 2, 5, 7, 10, 15 and 20 neurons. Classification accuracy after the 500 epochs is shown to increase with the number of neurons in the hidden layer. However past 15 neurons this growth becomes negligible; the difference in classification accuracy after convergence between 15 and 20 neurons is only 1%, and that increase of 5 neurons leads to an extra 25 seconds of training time. Table.6.1 in the appendix shows the results for all network configurations investigated, including an MLP with 25 and 30 neurons in a single layer. Results show that this trend of negligible increase in accuracy with a considerable increase in training time continues as the number of neurons increase.

After this, the effect that the number of layers had on accuracy was investigated, keeping the number of neurons in each layer to a relatively low value so as to keep computation time at a reasonable level. Fig.1.2(b) shows the classification accuracy over 500 epochs for MLPs with 2,3 or 4 hidden layers, and with either 2 or 5 neurons in each hidden layer. Results show that in fact, increasing the number of layers often had a detrimental effect on classification accuracy. For instance, two layers of 5 neurons produced a higher accuracy than three or four layers of 5 neurons. Increasing the number of layers also increased computation time quite drastically, as can be seen in Table.6.1. Consequently increasing the number of layers did not seem to be the most effective method. Despite this, the double layered networks from Fig.1.2(b) performed well and therefore one final investigation was done to look at double hidden layered networks with 2, 5, 7, 10, 15 and 20 neurons in both layers. Results in Fig.1.2(c) show very similar outcomes to that of the single-layered networks with only a marginal increase in accuracy. This also led to a considerable increase in run-time.

(A)



(B)

(C)

Fig.1.2: Investigations into network geometry.

c) Final chosen configuration

It was decided that when choosing the best MLP configuration a maximum cut-off for code run-time was set at 200s. This arbitrary time was chosen because classification accuracy could always be improved by increasing the number of neurons or number of hidden layers, however after a certain point this increase is not worth the added run-time especially considering how small this increase in accuracy was. Taking the results from all three experiments into account it was decided the best configuration would be a single layer of 20 neurons. This is because it produced one of the highest classification accuracies (62.5%) with a run-time of only 175s. Comparing to other configurations that achieved similar or slightly higher accuracies their run-times increase above the cut-off of 200s. Table.1.1 shows the chosen parameters and the total number of weights for this configuration, calculated using Eq.1.1. Using these parameters, a frequency-orientated confusion matrix was constructed shown in Table.1.2. From this the precision was then calculated for each class shown in Table.1.3. This informs us of the fraction

of times where the algorithm correctly predicted class *i* out of all instances where the algorithm declared class *i*.

No. of hidden layers	No. of neurons in each layer	No. of weights
1	20	164

Table 1.1: Table of chosen parameters for MLP.

$$Nw = (I + 1) * H + (H + 1) * O$$

Eq.1.1: Calculating the total number of weights. Input nodes, I = 3, Hidden layer nodes, H = 20, Output nodes, O = 4. The addition of one accounts for biases in the input and hidden layers.

		Predicted class			
	Class	1	2	3	4
Actual class	1	**953**	103	136	141
	2	172	**1030**	96	145
	3	181	88	**607**	253
	4	175	69	312	**539**

Precision (%)			
Class 1	Class 2	Class 3	Class 4
71.49	71.38	53.76	49.22

Table 1.2: Confusion matrix and precisions using MLP.

2 MLP for binary classification

To run the MLP for binary classification first two classes had to be chosen. Looking at the precision for each class suggests that classes 1 and 2 (walking and jogging) would be the easiest to distinguish considering that they are both predicted correctly more often than the other two classes. The data was then split to only include classes 1 and 2 and the MLP run with the same network configuration. The classification accuracy over 500 training epochs was found to converge at a considerably higher value of **85%**. The confusion matrix for binary classification can be seen in Table.2.1. Both this and the precision for class 1 and class 2, calculated as 90.3% and 80.9% respectively, further indicate that the MLP is much more effective at binary classification that 4-way classification.

		Predicted class	
	Class	1	2
Actual class	1	**1204**	129
	2	275	**1168**

Table 2.1: Confusion matrix for binary classification using MLP.

3-4 K-NN classifier

The classification algorithm chosen for comparison was a k-NN classifier. The reason for this being that the dataset is small, has relatively low dimensionality, and is all within a similar scale. For bigger or more complex datasets k-NN is not as effective of a tool due to the fact that the entire training dataset needs to be stored, resulting in very long/expensive computations. In this case however run-time was not a problem, and with this very non-linear spread of data k-NN is a good method due to its flexibility as a model and inherent nature to optimise locally.

Since K-NN is a non-parametric method, and the model grows with the training data, there are no parameters. K however acts as a hyperparameter as it cannot be learned from the training process, it must be defined separately. To fit with the design of the functions given, the TrainClassifierX.m was set up to produce two parameters; the training data and the training labels. These were then fed into ClassifyX along with the test data, where the k-NN method was implemented. The model was experimented with different values of k, ranging from 5-10, showing little change in the resulting accuracy and so a value of 5 was chosen as final. Using this model an accuracy of **86.9%** was achieved for binary classification, showing that it performed well. The confusion matrix for this k-NN model is shown in Fig.3.1 below, further confirming its effectiveness. The precision for class 1 and class 2 was calculated as 91.7% and 82.5% respectively.

Fig.3.1: Confusion matrix using k-NN.

		Predicted class	
	Class	1	2
Actual class	1	**1223**	110
	2	252	**1191**

5 Comparing MLP and k-NN classifiers

Comparison of the MLP with the k-NN classifier in terms of classification accuracy and confusion matrices shows that they perform quite similarly. The k-NN classifier shows to be a slightly more accurate model with roughly a two percent increase in classification accuracy. Confusion matrices are also remarkably similar, with class 1 (walking) predictions having roughly a 10% higher precision than class 2 (jogging) for both classifiers. On the contrary, where they do differ is run-time. The MLP classifier takes less than a third of the time to compute, with run-time amounting to 141s compared to 475s for the k-NN classifier when ran on the same computer. Consequently if 475s was to be taken as a limit for run-time, the MLP network configuration could be modified to include more neurons/more hidden layers which would, in some form, likely increase accuracy above that of the k-NN classifier. This highlights the previously mentioned flaw of using k-NN classifiers: The fact that it is a lazy classifier and has to store the entire training data can make it difficult to use for prediction in real time as computation takes too long. Despite this, if you consider the simplicity of the code used to implement the k-NN classifier compared to the MLP it shows the k-NN model does have some benefits. Conclusively if a classifier was needed that can be used repetitively and continually adapt to changes/increases in the dataset then the MLP is likely to be more efficient. In cases where the dataset remains small and unchanged a k-NN classifier could be more appropriate due to its simplicity when it comes to coding whilst producing high accuracy

Appendix A: Source code

```
%% %% %% Q1 MULTI-LAYER PERCEPTRON %%%%%%%%%%
clear all
close all
%% Load Data
load Activities.mat
%% 3D scatter plot
train_comb=[train_data(:,1) train_data(:,2) train_data(:,3)
train_labels(:,1)];
%splitting data into classes for visualisation
train_class1=train_comb((train_comb(:,4)==1),:);
train_class2=train_comb((train_comb(:,4)==2),:);
train_class3=train_comb((train_comb(:,4)==3),:);
train_class4=train_comb((train_comb(:,4)==4),:);

figure
scatter3(train_class1(:,1),train_class1(:,2),train_class1(:,3),9,
'r','.');
hold on
scatter3(train_class2(:,1),train_class2(:,2),train_class2(:,3),9,
'b','.');
hold on
scatter3(train_class3(:,1),train_class3(:,2),train_class3(:,3),9,
'g','.');
hold on
scatter3(train_class4(:,1),train_class4(:,2),train_class4(:,3),9,
'm','.');
xlabel('x [unit-distance/s^2]')
ylabel('y [unit-distance/s^2]')
zlabel('z [unit-distance/s^2]')
legend('walking','jogging','walking upstairs','walking
downstairs')
%% %% Q1(b) Investigating network geometry
%This section shows the code used to investigate different
network
%geometries; number of layers/number of neurons in each layer.
The
%algorithm is set up using a for loop so that multiple
configurations can
%be investigated at once and the resulting accuracies plotted on
a single
%graph. Here the code is set up for the final chosen
configuration: a single layer of 20 neurons.
nbrOfEpochs_max = 500;
nbrOfEpochs=[(1:nbrOfEpochs_max)'];
% initialise variables
```

```
% "multi" variables are to allow data to be stored from multiple
configurations
nbrNeuronsMulti=[];
accuracyMulti=[];
best_prediction_multi=[];
accuracy_max=[];
j={[20]};%j can be set as multiple configurations
e.g.{[2],[5],[10,10]}
for i=1:size(j,2);
    tic;%recording time taken for each configuration to run
nbrOfNeuronsInEachHiddenLayer = j{1,i};

[accuracy, best_prediction] = MLP_REST(train_data, train_labels,
test_data, test_labels, nbrOfNeuronsInEachHiddenLayer,
nbrOfEpochs_max);

nbrNeuronsMulti{1,i}= nbrOfNeuronsInEachHiddenLayer;
accuracyMulti(nbrOfEpochs,i)=accuracy;
best_prediction_multi((1:length(best_prediction)),i)=best_predict
ion;
accuracy_max(1,i)=sum(accuracyMulti((476:500),i))/length(accuracy
Multi(476:500));%maximum accuracy is taken as the average over
the last 25 epochs
toc;
time(1,i)=toc;
end

% Plot of accuracy over number of epochs for multiple
configurations
figure
plot(nbrOfEpochs,accuracyMulti);
xlabel('number of epochs')
ylabel('classification accuracy [%]')
title('Learning curve for 20 neurons single-layer')
legend(strcat('no. of Neurons =','[20]'),'Location','southeast')

%% %% Q1(c)Final configuration & confusion matrix
% Best MLP config: 1 layer, 20 neurons
% -> accuracy = 0.627, comp time = 175 seconds
%%% Confusion matrix %%%
%number of weights
nbrOfweights=((3+1)*nbrOfNeuronsInEachHiddenLayer)+((1+nbrOfNeuro
nsInEachHiddenLayer)*4);
%confusion matrix
label_compare=[test_labels(:,1),best_prediction(:,1)];
mat=zeros(4);

for i=1:4
    for j=1:4
    mat(i,j)=sum(label_compare(:,1)==i & label_compare(:,2)==j)
```

```
    end
end

%Precision for each class - probabilities for each class being
predicted correctly
prob=(diag(mat)'./sum(mat'))*100;
%%%%%%%%%%%%% Class 1 and 2 are easiest to distinguish
%%%%%%%%%%%%%

%% %% %% Q2 BINARY CLASSIFICATION %%%%%%%%%%%
clear all
close all
%% Load Data
load Activities.mat
%% Split into two classes (1 & 2)
train_data_comb=[train_data,train_labels];
train_data_binary=train_data_comb((train_data_comb(:,4)==1 |
train_data_comb(:,4)==2),1:3);%binary training data accelerations
train_labels_binary=train_data_comb((train_data_comb(:,4)==1 |
train_data_comb(:,4)==2),4);%binary training data classes

test_data_comb=[test_data,test_labels];
test_data_binary=test_data_comb((test_data_comb(:,4)==1 |
test_data_comb(:,4)==2),1:3);%binary test data accelerations
test_labels_binary=test_data_comb((test_data_comb(:,4)==1 |
test_data_comb(:,4)==2),4);%binary test data classes
%% MLP for binary classification
%Here the code is the same as Q1(b) and with the same
configuration. Inputs
%are just changed to the binary data variables.
nbrOfEpochs_max = 500;
nbrOfEpochs=[(1:nbrOfEpochs_max)'];

nbrNeuronsMulti=[];
accuracyMulti=[];
best_prediction_multi=[];
accuracy_max=[];
j={[20]};
for i=1:size(j,2);
    tic;
nbrOfNeuronsInEachHiddenLayer = j{1,i};

[accuracy, best_prediction] = MLP_REST(train_data_binary,
train_labels_binary, test_data_binary, test_labels_binary,
nbrOfNeuronsInEachHiddenLayer, nbrOfEpochs_max);

nbrNeuronsMulti{1,i}= nbrOfNeuronsInEachHiddenLayer;
accuracyMulti(nbrOfEpochs,i)=accuracy;
best_prediction_multi((1:length(best_prediction)),i)=best_predict
ion;
```

```
accuracy_max(1,i)=sum(accuracyMulti((476:500),i))/length(accuracy
Multi(476:500));
toc;
time(1,i)=toc;
end

figure
plot(nbrOfEpochs,accuracyMulti);
xlabel('number of epochs')
ylabel('classification accuracy [%]')
title('Learning curve for 20 neurons single-layer - Binary
classification')
legend(strcat('no. of Neurons =','[20]'),'Location','southeast')

%% Confusion matrix - binary classification
label_compare_binary=[test_labels_binary(:,1),best_prediction(:,1
)];
mat_binary=zeros(2);

for i=1:2
    for j=1:2
    mat_binary(i,j)=sum(label_compare_binary(:,1)==i &
label_compare_binary(:,2)==j)
    end
end

%probabilities for each class being predicted correctly
prob_binary=(diag(mat_binary)'./sum(mat_binary'))*100;

%% %% %% Q3 & 4 MY CLASSIFIER %%%%%%%%%%%%%%

%% Train my classifier
% *Training does not actually occur here as using k-NN, therefore
training
% occurs within ClassifyX. Here the function is just used to set
parameters as the training
% datapoints and their respective classes.
parameters =
TrainClassifierX(train_data_binary,train_labels_binary);

%% Classification using k-NN
tic;
class = ClassifyX(test_data_binary, parameters)
accuracy_kNN =
sum(class(:,1)==test_labels_binary)/length(class);%Accuracy of my
classifier
toc;
time_kNN=toc;
%% Confusion matrix - k-NN
label_compare_binary_kNN=[test_labels_binary(:,1),class(:,1)];
```

```
mat_binary_kNN=zeros(2);

for i=1:2
    for j=1:2
    mat_binary_kNN(i,j)=sum(label_compare_binary_kNN(:,1)==i &
label_compare_binary_kNN(:,2)==j)
    end
end

%probabilities for each class being right
prob_binary_kNN=(diag(mat_binary_kNN)'./sum(mat_binary_kNN'))*100
;

%% Sanity check
SanityCheck()
```

Appendix B: Table 6.1 – Results from all network configurations tested for Q1(b).

No. of hidden layers	No. of neurons/neuron arrangement	Max accuracy (average last 25 epochs) [%]	Time for computation (s)
Single layer arrangement			
1	2	0.46092	95.07782987
1	5	0.552936	110.0150481
1	7	0.560728	118.7401371
1	10	0.609968	127.3690636
1	15	0.616336	151.7008867
1	20	0.625848	175.7460885
1	25	0.631368	200.9787856
1	30	0.634096	228.3144754
Multi-layer arrangement			
2	2,2	0.484464	129.8264912
2	5,5	0.580904	159.4116765
3	2,2,2	0.457408	166.5987301
3	5,5,5	0.55536	209.3441182
4	2,2,2,2	0.460792	203.0999138
4	5,5,5,5	0.557392	260.3815978
Double-layer arrangement			
2	2,2	0.495128	129.8264912
2	5,5	0.58708	159.4116765
2	7,7	0.594088	178.2466233
2	10,10	0.619752	206.915314
2	15,15	0.628896	244.808691
2	20,20	0.625776	281.1876321