COMP7703 Machine Learning
Assignment 3

Thanat Chokwijitkul 44522328

Question 7.1

Question 7.1(a)

Compare the performance of the different optimisers using the MLP topology chosen in Prac 6 Q3.

As an exhaustive experimental exploration of optimisers, learning rates, etc. is not required, the design choice and hyperparameters are mostly based on intuition and experience in the field of artificial neural networks and deep learning (thesis project), along with results and diagnostic information from the Tensorboard output.

Summary of the MLP topology chosen in Prac 6 Q3

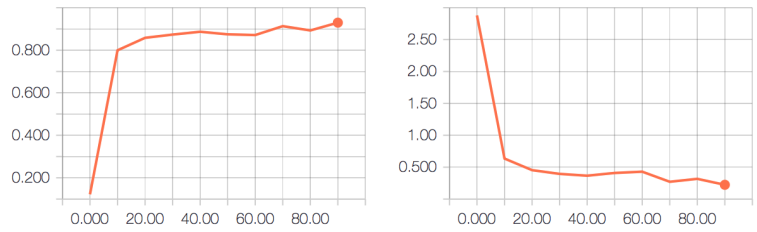
Input layer: 784 neurons
Hidden layer: 512 neurons with rectified linear units (ReLUs)
Output layer: 10 neurons with the softmax function

Training algorithm: Adaptive Moment Estimation (Adam)
Cost function: Cross-Entropy

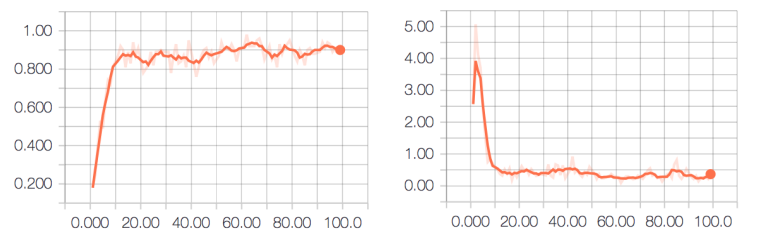
Experimental Results

Note: For question 7.1(a), the maximum number of training steps is 100 epochs.

Adam (Original Topology)



(a) Training accuracy (b) Training error



(c) Test accuracy (d) Test error

Figure 1: Performance of the MLP topology chosen in Prac 6 Q3 with Adam as the optimiser

Adaptive Moment Estimation (Adam) is a stochastic gradient-based optimisation technique with adaptive learning rates and momentum on each parameter, designed to efficiently deal with machine learning problems with high-dimensional parameter space or large datasets [1]. Figure 1 shows the performance of the MLP topology chosen in Prac 6 Q3 with Adam as the optimiser in terms of training accuracy, training error, test accuracy and test error (Figure 1(a), 1(b), 1(c) and 1(d) respectively). With learning rate of 0.01, beta_1 of 0.5 and beta_2 of 0.999, the network achieved a training accuracy of 0.9296 and a training error of 0.2235. The trained model achieved an accuracy of 0.90 and a cross entropy error of 0.3685 on the test set. These results are quite impressive considering such small amount of training time.

Gradient Descent

(a) Training accuracy (b) Training error

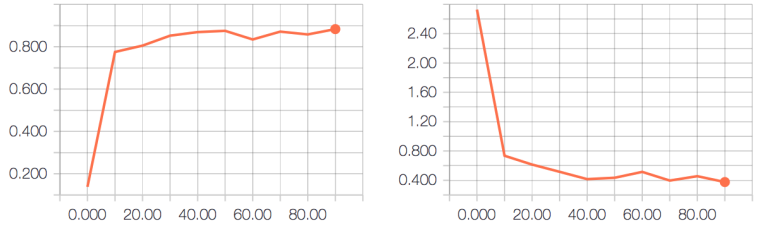
(c) Test accuracy (d) Test error

Figure 2: Performance of the MLP topology chosen in Prac 6 Q3 with Gradient Descent as the optimiser

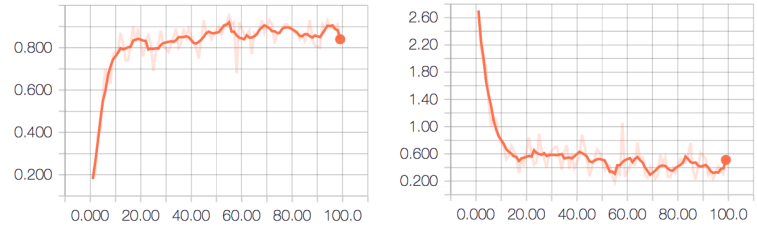
Gradient descent is one of the most well-known algorithms and the most common approach to optimise MLPs [2]. Figure 2 shows the performance of the MLP topology chosen in Prac 6 Q3 with gradient descent as the optimiser in terms of training accuracy, training error, test accuracy and test error (Figure 2(a), 2(b), 2(c) and 2(d) respectively). With learning rate of 0.1, the network achieved a training accuracy of 0.8736 and a training error of 0.4336. The trained model achieved an accuracy of 0.80 and a cross entropy error of 0.6079 on the test set.

Momentum

As Stochastic Gradient Descent (SGD) tends to have unstable per iteration update and takes more time to converge at the local minima, the idea of momentum can be used to solve this issue [2]. Figure 3 shows the performance of the MLP topology chosen in Prac 6 Q3 with momentum as the optimiser in terms of training accuracy, training error, test accuracy and test error (Figure 3(a), 3(b), 3(c) and 3(d) respectively). With learning rate of 0.05 and momentum of 0.9, the network achieved a training accuracy of 0.8831 and a training error of 0.3761. The trained model achieved an accuracy of 0.84 and a cross entropy error of 0.5108 on the test set. The performance of momentum is considered similar to the performance of the standard gradient descent with a slight increase in accuracy and cross-entropy error.



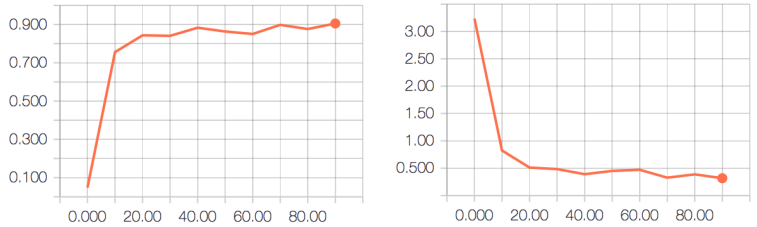
(a) Training accuracy (b) Training error



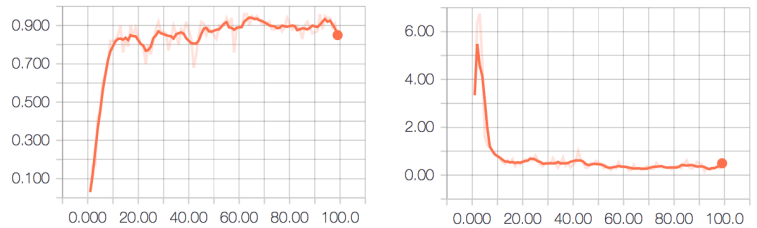
(c) Test accuracy (d) Test error

Figure 3: Performance of the MLP topology chosen in Prac 6 Q3 with momentum as the optimiser

Adagrad



(a) Training accuracy (b) Training error

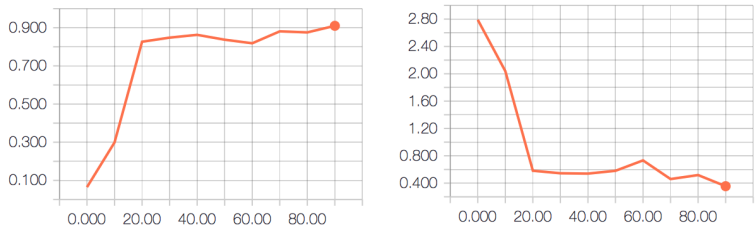


(c) Test accuracy (d) Test error

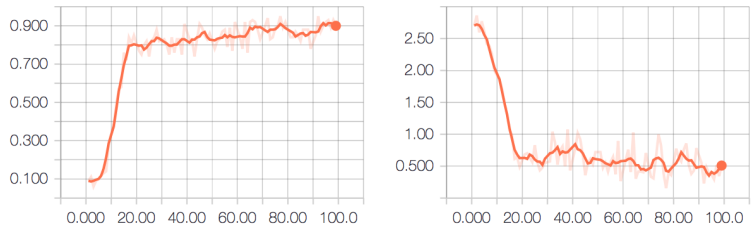
Figure 4: Performance of the MLP topology chosen in Prac 6 Q3 with Adagrad as the optimiser

Adaptive Gradient (Adagrad) is a gradient-based optimiser that adjusts the learning rate to the parameters, performing the smaller update for frequent parameters and larger update for infrequent parameters [3]. Figure 4 shows the performance of the MLP topology chosen in Prac 6 Q3 with Adagrad as the optimiser in terms of training accuracy, training error, test accuracy and test error (Figure 4(a), 4(b), 3(c) and 4(d) respectively). With learning rate of 0.15 and initial accumulator value of 0.1, the network achieved a training accuracy of 0.9055 and a training error of 0.3167. The trained model achieved an accuracy of 0.85 and a cross entropy error of 0.4959 on the test set.

RMSProp



(a) Training accuracy (b) Training error



(c) Test accuracy (d) Test error

Figure 5: Performance of the MLP topology chosen in Prac 6 Q3 with RMSProp as the optimiser

Root Mean Square Propagation (RMSProp) is another gradient-based optimiser that adapts the learning rate for each parameter. The main idea is to utilise the magnitude of recent gradients to normalise the gradients [4]. Figure 5 shows the performance of the MLP topology chosen in Prac 6 Q3 with RMSProp as the optimiser in terms of training accuracy, training error, test accuracy and test error (Figure 5(a), 5(b), 5(c) and 5(d) respectively). With learning rate of 0.001, decay rate of 0.5 and momentum of 0.9, the network achieved a training accuracy of 0.91 and a training error of 0.3551. The trained model achieved an accuracy of 0.90 and a cross entropy error of 0.5097 on the test set. The performance of RMSProp is better than the standard gradient descent, momentum and Adagard but inferior comparing with the performance of Adam.

Summary of Performance

Table 1: Performance of the MLP topology chosen in Prac 6 Q3 using different optimisers

Optimiser	**Training Accuracy**	**Training Error**	**Test Accuracy**	**Test Error**
Gradient Descent	0.8736	0.4336	0.8000	0.6079
Momentum	0.8831	0.3761	0.8400	0.5108
Adagrad	0.9055	0.3167	0.8500	0.4959
RMSProp	0.9100	0.3551	0.9000	0.5097
Adam	0.9296	0.2235	0.9000	0.3685

As mentioned in the practical sheet, the number of training steps has been set to a small number. For question 7.1(a), each model was trained with the maximum number of training steps of 100 epochs (since question 7.1(b) asks to increase 'max_steps'). Therefore, one assumption regarding the results is that each model has not reached its optimal performance yet and can be improved with more training time by increasing the number of epochs. However, the obtained results as shown in Table 1 are sufficient to demonstrate the difference of these optimisers regarding their performance.

The statement "Adam seems to be the most popular at the moment, followed by RMSProp" mentioned in the practical sheet seems to be convincing considering the results from Table 1. The Adam optimiser achieved the highest performance regarding both accuracy and cross-entropy error, followed by RMSProp. On the other hand, gradient descent, momentum achieved inferior performance with the same amount of training time. The reason is perhaps because of the learning algorithm itself e.g. stochastic gradient descent-based optimisers tend to perform better than the

standard gradient descent, or the choice of hyperparameters e.g. learning rate. If the latter is the case, the performance can be improved by parameter tuning using a set of parameters specific to that optimiser and the training data. Nevertheless, it can be concluded that the network topology designed for Prac 6 Q3 is considered optimal among other topologies regarding the choice of the optimiser.

Question 7.1(b)

Compare the performance of the network with ReLU, tanh, and sigmoid functions as the activation function.

Summary: Activation Functions

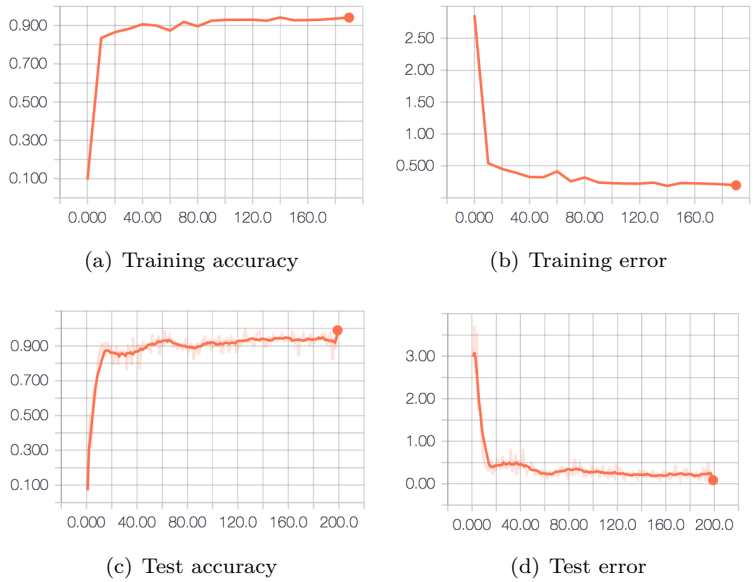
Table 2: Summary of ReLU, sigmoid and tanh functions [5]

Activation Function	Definition	Range
Rectified Linear Unit (ReLU)	$f(x) = max(0, x)$	$[0, \infty)$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$
Hyperbolic Tangent (Tanh)	$f(x) = \frac{2}{1+e^{-2x}} - 1$	$(-1, 1)$

Experimental Results

Note: For question 7.1(b), the maximum number of epochs has been increased to 200 epochs.

ReLU



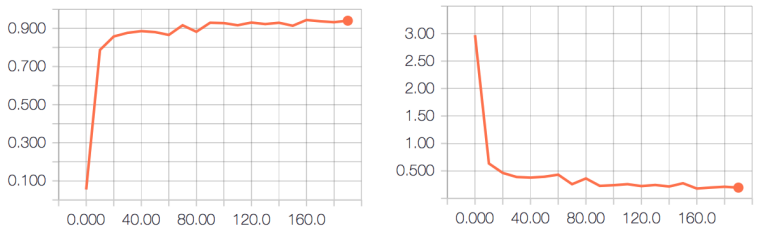
(a) Training accuracy (b) Training error

(c) Test accuracy (d) Test error

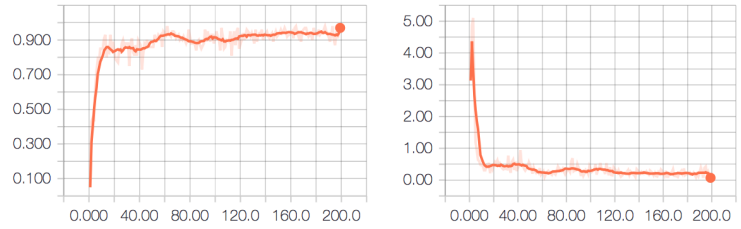
Figure 6: Performance of the MLP topology chosen in Prac 6 Q3 with ReLU as the activation function

Figure 6 shows the performance of the MLP topology chosen in Prac 6 Q3 with Adam as the optimiser and rectified linear unit (ReLU) as the activation function in terms of training accuracy, training error, test accuracy and test error (Figure 6(a), 6(b), 6(c) and 6(d) respectively). Using the same hyperparameters described for Adam in question 7.1(a), the network achieved a training accuracy of 0.9416 and a training error of 0.1961. The trained model achieved an accuracy of 0.99 and a cross entropy error of 0.0865 on the test set, which is considered a huge improvement after increasing the maximum training steps.

Sigmoid



(a) Training accuracy (b) Training error

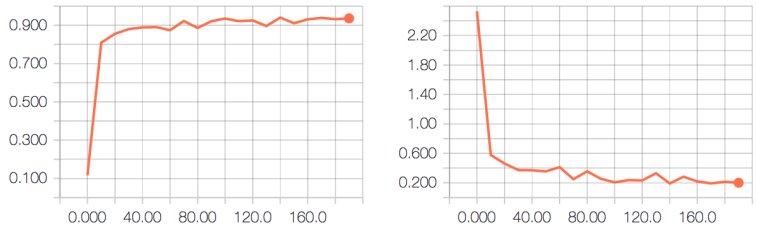


(c) Test accuracy (d) Test error

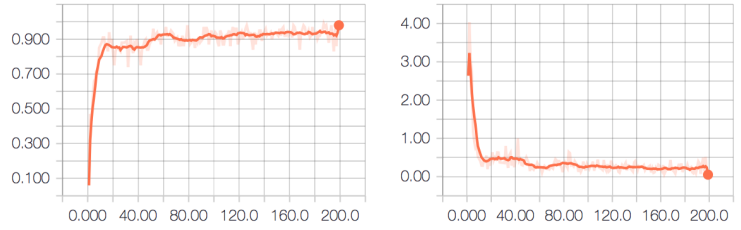
Figure 7: Performance of the MLP topology chosen in Prac 6 Q3 with sigmoid as the activation function

Figure 7 shows the performance of the MLP topology chosen in Prac 6 Q3 with Adam as the optimiser and sigmoid as the activation function in terms of training accuracy, training error, test accuracy and test error (Figure 7(a), 7(b), 7(c) and 7(d) respectively). After training, the network achieved a training accuracy of 0.9405 and a training error of 0.1956. The trained model achieved an accuracy of 0.97 and a cross entropy error of 0.0708 on the test set. These results are almost identical to the case of ReLU. However, the accuracy on the test set is slightly lower.

Tanh



(a) Training accuracy (b) Training error



(c) Test accuracy (d) Test error

Figure 8: Performance of the MLP topology chosen in Prac 6 Q3 with tanh as the activation function

Figure 8 shows the performance of the MLP topology chosen in Prac 6 Q3 with Adam as the optimiser and tanh as the activation function in terms of training accuracy, training error, test accuracy and test error (Figure 8(a), 8(b), 8(c) and 8(d) respectively). After training, the network achieved a training accuracy of 0.9361 and a training error of 0.2007. The trained model achieved an accuracy of 0.98 and a cross entropy error of 0.0513 on the test set. These results are almost same as the results of sigmoid, but with lower error rate on the test set.

Summary of Performance

Table 3: Performance of the MLP topology chosen in Prac 6 Q3 with Adam as the optimiser and three different activation functions

Activation Function	Training Accuracy	Training Error	Test Accuracy	Test Error
ReLU	0.9416	0.1961	0.9900	0.0865
Sigmoid	0.9405	0.1956	0.9700	0.0708
Tanh	0.9361	0.2007	0.9800	0.0513

Table 3 outlines the performance of the MLP topology chosen in Prac 6 Q3 with Adam as the optimiser and three different activation functions, including ReLU, Sigmoid and Tanh. In this particular case, different activation functions do not seem to yield much difference in terms of training performance but seem to slightly impact the generalisation capability when performing classification on the test set. According to Table 3, the results yielded that ReLU achieved the highest test accuracy, followed by tanh and sigmoid. However, tanh achieved the lowest test error rate, followed by sigmoid and ReLU.

Even though ReLU yielded the best results, one may assume that any of these three activation functions is applicable for this particular task. However, if the structure of MLP is more complex i.e. increase in the number of hidden layers, ReLU seems to be a more appropriate choice. According to [6], ReLU has range $[0, \infty)$ while the sigmoid and tanh functions have the range of (0,1) and (-1,1) respectively. The gradient of the sigmoid and hyperbolic tangent function tend to vanish in a deep neural network architecture which does not occur in the case of ReLU. The rectifier does not alter the gradient value during the backpropagation process, which alleviates the vanishing gradient problem. Additionally, even though tanh is quicker at converging than sigmoid or logistic function, ReLU has been proved to be much faster than tanh when training a network [6], [7].

Question 7.2

Using the link to the CS231n CNN theory [8], calculate the volume for the weight matrices of each layer for a convolutional network that has two conv-pool layers:

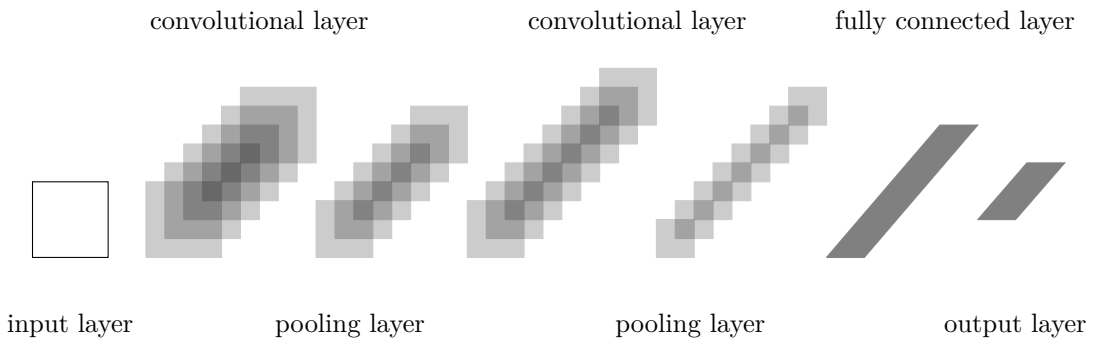


Figure 9: A convolutional neural network with two conv-pool layers

For the conv layer, assume:

- A padding of 1
- A stride length of 1
- A spatial extent of 3

Remember, the number of filters in a particular layer is given by the expression: $i \times 32$, where i is the layer number.

Assume that the first FC layer has 1,024 neurons and the second has the number of classes in MNIST. Once you have done this calculate the number of parameters in this network.

Variables

W: Input volume size (width)
H: Input volume size (Height)
D: Depth
F: Filter size (spatial extent or receptive field)
S: Stride length
P: Amount of zero-padding
K: Number of filters (depth)

Summary: Convolutional layer

A convolutional layer:

- accepts a volume of size $W_1 \times H_1 \times D_1$
- produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- with parameter sharing:
  - $F \times F \times D_1$ is the number of weights per filter
  - $(F \times F \times D_1) \times K + K$ biases is the total number of weights

Summary: Pooling layer

A pooling layer:

- accepts a volume of size $W_1 \times H_1 \times D_1$
- produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$

Solution

Convolutional Layer: $F = 3, S = 1, P = 1$
Pooling Layer: $F = 2, S = 1$

Keywords

INPUT: Input layer
CONV-N: Convolutional layer with N filters
POOL: Pooling layer
FC: Fully connected layer

Total Number of Parameters With Parameter Sharing

Table 4: Calculation details of the number of parameters in the network

Layer	**Shape**	**Number of Neurons**	**Number of Parameters**
INPUT	$[28 \times 28 \times 1]$	$28 \times 28 \times 1 = 784$	0
Output dimension of CONV-32: $(28 - 3 + 2(1))/1 + 1 = 28$			
CONV-32	$[28 \times 28 \times 32]$	$28 \times 28 \times 32 = 25088$	$((3 \times 3 \times 1) + 1) \times 32 = 320$
Output dimension of POOL: $(28 - 2)/1 + 1 = 27$			
POOL	$[27 \times 27 \times 32]$	$27 \times 27 \times 32 = 23328$	0
Output dimension of CONV-64: $(27 - 3 + 2(1))/1 + 1 = 27$			
CONV-64	$[27 \times 27 \times 64]$	$27 \times 27 \times 64 = 46656$	$((3 \times 3 \times 32) + 1) \times 64 = 18496$
Output dimension of POOL: (27 - 2) / 1 + 1 = 26			
POOL	$[26 \times 26 \times 64]$	$26 \times 26 \times 64 = 43264$	0
FC	$[1 \times 1 \times 1024]$	1024	$(43264 + 1) \times 1024 = 44303360$
FC	$[1 \times 1 \times 10]$	10	$(1024 + 1) \times 10 = 10250$
Total: 320 + 18496 + 44303360 + 10250 = 44332426 parameters			

The number of parameters in each layer is the volume of weight matrix of that layer including the bias unit. According to Table 4, the total number of parameters in this network is **44,332,426 parameters**.

Question 7.4

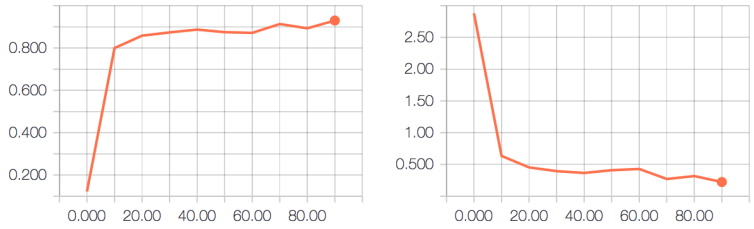
Compare the performance of the MLP and the CNN that you created.

Experimental Results

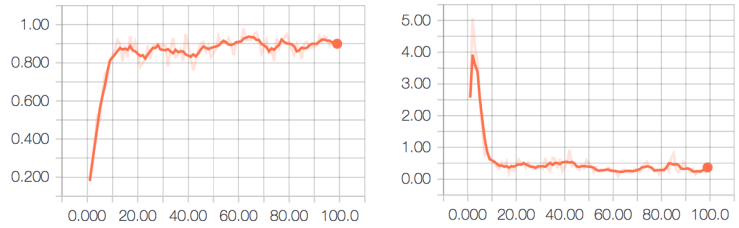
Note: For question 7.4, the maximum number of training steps is 100 epochs.

Multilayer Perceptron

Figure 10 shows the performance of the MLP created for question 7.1(a) based on the topology selected in Prac 6 Q3 in terms of training accuracy, training error, test accuracy and test error (Figure 10(a), 10(b), 10(c) and 10(d) respectively). The network uses Adam as its optimiser and ReLu as the activation function. After training, the network achieved a training accuracy of 0.9296 and a training error of 0.2235. The trained model achieved an accuracy of 0.90 and a cross entropy error of 0.3685 on the test set.



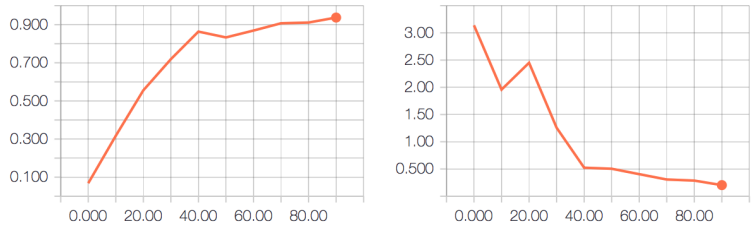
(a) Training accuracy (b) Training error



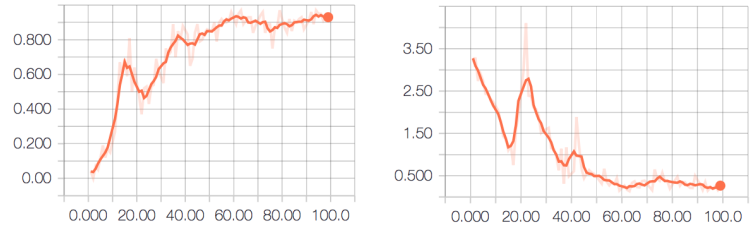
(c) Test accuracy (d) Test error

Figure 10: Performance of the MLP topology chosen in Prac 6 Q3 with Adam as the optimiser

Convolutional Neural Network



(a) Training accuracy (b) Training error



(c) Test accuracy (d) Test error

Figure 11: Performance of a convolutional neural network

Figure 11 shows the performance of a convolutional neural network in terms of training accuracy, training error, test accuracy and test error (Figure 11(a), 11(b), 11(c) and 11(d) respectively). The network uses RMSProp as the optimiser and ReLu as the activation function as this design choice tends to result in a better gain in performance observed from multiple experimental trials. The network follows the architecture illustrated in Figure 12.

The convolutional layer of this network has 32 filters with a filter size of 3 and stride of 2. The pooling layer has a filter size of 2 and stride of 2. The fully connected layer consists of 512 neurons and the output layer consists of 10 neurons (number of classes). After training, the network achieved a training accuracy of 0.9374 and a training error of 0.2037. The trained model achieved an accuracy of 0.93 and a cross entropy error of 0.2667 on the test set.

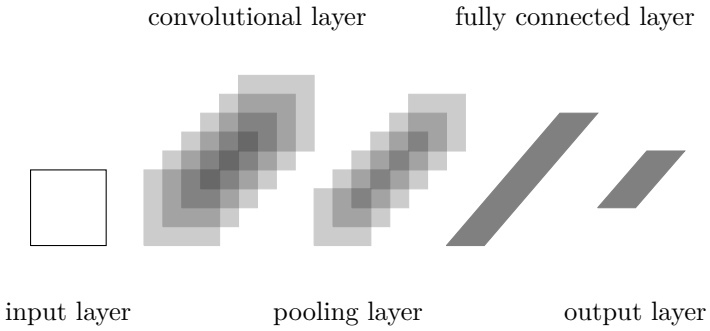


Figure 12: A convolutional neural network with a single conv-pool layer

Summary of Performance

Table 5: Performance of the MLP topology chosen in Prac 6 Q3 and a CNN

Activation Function	Training Accuracy	Training Error	Test Accuracy	Test Error
MLP	0.9296	0.2235	0.9000	0.3685
CNN	0.9374	0.2037	0.9300	0.2667

Table 5 compares the performance of the MLP based on the topology selected in Prac 6 Q3 and a convolutional neural network. The CNN archived better performance than the standard MLP even though this MLP is considered the optimal choice among other standard MLP topologies according to question 7.1(a), which explains why the differences are relatively small. This level of performance is expected since CNN was designed especially for pattern/image recognition problems. One can design a standard MLP to be equivalent to a CNN, but since the total number of parameters would be much higher, the training time would also rise accordingly. In the case of a CNN with parameter sharing, the total number of parameters is drastically reduced, and the training time is reduced accordingly. Another observation is that the performance of a standard MLP equivalent to a CNN is usually inferior since it is bounded by a large number of parameters, which leads to more noise during the training process [9].

Question 8.3

Using the same dataset with the parameter/kernel setting that gave the best performance in Prac8 Q2, experiment by varying the C parameter (which has value 1.0 by default). Try C = 0.01, 0.1, 10 and 100. Use Matlab to plot the performance (percentage correct) on the test set.

The ionosphere dataset has been selected for question 8.3. The details of this dataset is listed below:

- Number of input features: 34
- Number of classes: 2
- Number of samples: 351 (264 training samples and 87 test samples)

According to the experiment with the Support Vector Machine (SVM) kernel function by trying different parameter value settings for both polynomial and radial basis function (RBF) kernels in question 8.2, the RBF kernel with the gamma value of 1.0 yielded the best results. The parameter 'gamma' can be regarded as the spread of the kernel or the decision region. When the value of gamma is small, the decision boundary is very board. Oppositely, when the value of gamma is high, it forms areas of decision boundaries around data points [10].

Although the gamma value of 1.0 is optimal when using handout cross validation with the default training/test set percentage split in Weka (66%), it appears that the gamma value of 0.6 is

optimal in the case of 10-fold cross validation. Since this the cross-validation method used in this question is the default training/test set percentage split, the best gamma value is 1.0.

This question experiments with different values of the C parameter, a penalty for misclassification. When C is high, the classifier is heavily penalised for misclassification (low bias, high variance). Oppositely, when C is low, misclassification is allowed and the classifier may not try to avoid some misclassified data points (high bias, low variance) [10].

Experimental Results

Note: The results also include the default value of the C parameter in Weka (1.0).

Table 6: Performance with different values of the C parameter

C	Accuracy (%)	Error (%)	Precision	Recall	F-measure
0.01	55.462	44.538	0.555	1.0	0.714
0.1	55.462	44.538	0.555	1.0	0.714
1.0	94.958	5.042	0.941	0.970	0.955
10.0	94.958	5.042	0.941	0.970	0.955
100.0	95.798	4.202	0.942	0.985	0.963

Table 6 outlines the experimental results when varying the C parameters in terms of accuracy, misclassification error, precision, recall and f-measure.

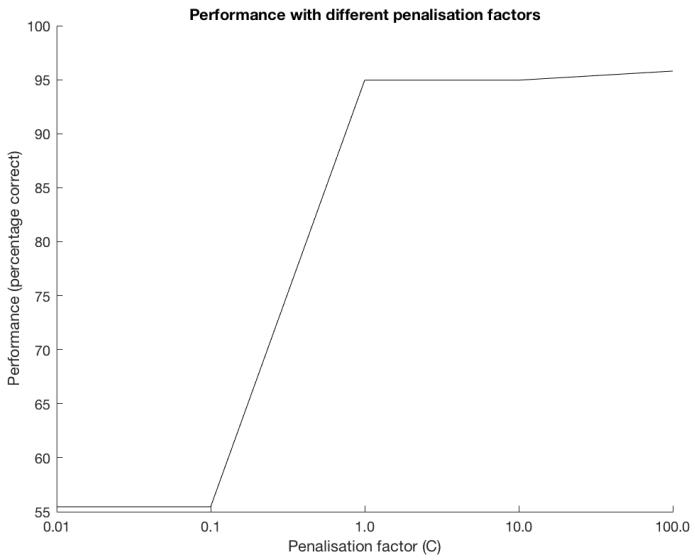


Figure 13: Performance with different penalisation factors

Figure 13 is a plot of performance (percentage correct) on the test set when C = 0.01, 0.1, 1.0, 10.0 and 100.0. Even though the performance tends to get better as the value of the C parameter increases up to 100.0, from an extra experiment, the performance remains unchanged when the C value is greater than 100.0.

Question 9.5

For the sonar, diabetes, ionosphere and glass datasets, attempt to reproduce the results reported in the paper [11] – specifically in the first column of Table 2. You should be able to find most of the experimental details used by the authors in the paper – provide comments if you have to make any assumptions for your experiments. Record your results and comment briefly on any difference between them and the results in the paper.

The main task for this question is to attempt to reproduce the results reported in [11] i.e. the test set error rates for the diabetes, glass, ionosphere and sonar data sets using a single neural network classifier. The results are averaged over five 10-fold cross-validation trials. The neural networks are trained using the standard backpropagation algorithm. Parameter settings for the neural networks include:

- Learning rate of 0.15
- Momentum term of 0.9

The number of hidden neurons is based on the criteria of having at least one hidden neuron per output, at least one hidden neuron for every ten inputs, and five hidden neurons being a minimum.

The number of epochs is based on the problem size. Let N be the number of data points:

- If $N < 250$ then train the network for 60 - 80 epochs
- If $250 \leq N \leq 500$ then train the network for 40 epochs
- If $N > 500$ then train the network for 20 - 40 epochs

Table 7: Summary of datasets

Dataset	Cases	Class	Input	Output	Hidden	Epochs
Diabetes	768	2	8	1	5	30
Glass	214	6	9	6	10	80
Ionosphere	351	2	34	1	10	40
Sonar	208	2	60	1	10	60

Table 7 is the summary of data sets used in this experiment based on the information from [11]. The details include the number of data points, the number of classes, the number of input, output, and hidden units used in the neural networks, and the number of epochs.

Experimental Results

Table 8: Test-set error rates obtained from [11]

Dataset	Error Rate
Diabetes	23.9
Glass	38.6
Ionosphere	9.7
Sonar	16.6

Table 9: Reproduced test-set error rates

Dataset	Error Rate
Diabetes	23.9
Glass	37.8
Ionosphere	9.1
Sonar	16.6

Table 8 and 9 show the original test-set error rates from [11] and the reproduced error rates for the data sets described in Table 7 respectively. Each table outlines the test-set error rates for the diabetes, glass, ionosphere and sonar data sets using a single neural network classifier. Each error rate is an average of five 10-fold cross validation experiments. Using the described neural network architecture and parameter settings, the reproduced error rates after performing classification on the test sets of the diabetes and sonar datasets are 23.85 and 16.63 respectively, which are equal to the original results when the values are rounded to a single decimal point. However, the reproduced results of the glass and ionosphere datasets are different from the original results as the error rates of both datasets are slightly lower, which are still considered similar since the differences are not large (the difference of 0.8 and 0.6 for the glass and ionosphere respectively). One assumption is that this is perhaps due to the limitation of Weka for random weight initialisation. Opitz and Maclin [11] mentions that weights in the network should be randomly initialised to be in the range $[-0.5, 0.5]$, which is not feasible for Weka.

The data and the calculation details of the error rates in Table 9 are listed as follows:

Diabetes

$avg(24.6094, 23.5677, 23.8281, 24.0885, 23.1771) = 23.85416$

Glass

$avg(36.4486, 37.8505, 39.2523, 36.9159, 38.3178) = 37.75702$

Ionosphere

$avg(9.6866, 9.1168, 8.8319, 8.547, 9.4017) = 9.1168$

Sonar

$avg(15.8654, 15.3846, 16.8269, 17.3077, 17.7885) = 16.63462$

Question 9.6

For the datasets used in Prac 9 Q5, attempt to reproduce the results in columns 3 and 5 of the paper [11]. Provide comments if you have to make any assumptions for your experiments. Record your results and comment briefly on any difference between them and the results in the paper.

The main task for this question is to attempt to reproduce the results reported in [11] i.e. the test set error rates for the diabetes, glass, ionosphere and sonar data sets using an ensemble where the MLPs are trained using randomly resampled training sets (Bagging) and an ensemble where the MLPs are trained using weighted resampled training sets (Boosting). In the case of Boosting, the resampling is based on the Adaptive Boosting (Ada) method, a meta-algorithm which can be used to boost the performance of any machine learning algorithm but is best for weak learners [12]. The assumptions for this question are based on the assumptions in question 9.5, including:

- Result computation method - averaged over five 10-fold cross-validation experiments
- Learning algorithm - backpropagation
- Parameter settings - learning rate = 0.15, momentum = 0.9
- Network structure - details listed in Table 7
- Number of training epochs - details listed in Table 7

Experimental Results

Table 10: Test-set error rates from [11]

Dataset	Bagging	Ada-boosting
Diabetes	22.8	23.3
Glass	33.1	31.1
Ionosphere	9.2	8.3
Sonar	16.8	13.0

Table 11: Reproduced test-set error rates

Dataset	Bagging	Ada-boosting
Diabetes	22.7	23.4
Glass	33.1	31.2
Ionosphere	9.0	8.3
Sonar	15.4	13.5

Table 10 and 11 show the original test-set error rates from [11] and the reproduced error rates for the datasets described in Table 7 respectively. Each table outlines the test set error rates for the diabetes, glass, ionosphere and sonar data sets using the Bagging and Boosting approaches. By strictly following the neural network architecture and parameter settings described in Table 7, most of the reproduced error rates are similar to the original error rates. For Bagging, the reproduced error rates after performing classification on the test sets of the glass, diabetes and ionosphere datasets are either equal or very close to the results in Table 10 (with the minimum difference of 0.1 and the maximum difference of 0.2). However, the error rate for the sonar dataset seems to slightly differ from the original error rate.

In [11], it can be observed that the error rates produced by a single neural network classifier are always higher than the error rates produced by Bagging for all datasets except for the sonar dataset. In the case of the sonar dataset, the error rate increases by 0.2 (from 16.6 to 16.8) when applying Bagging. However, the reproduced error rate decreases by 1.2 comparing with the result of a single neural network classifier (from 16.6 to 15.4).

For Boosting, the reproduced error rates after performing classification on the test sets of the ionosphere is equal to the original error rate, and the reproduced error rates for other datasets are considered similar (with the minimum difference of 0.1 and the maximum difference of 0.5).

The data and the calculation details of the error rates in Table 11 are listed as follows:

Diabetes

Bagging: $avg(22.526, 22.9167, 22.7865, 23.1771, 22.2656) = 22.73438$
Ada-boosting: $avg(23.6979, 22.0052, 23.8281, 24.349, 22.9167) = 23.35938$

Glass

Bagging: $avg(32.7103, 32.243, 33.6449, 34.1121, 33.1776) = 33.17758$
Ada-boosting: $avg(31.3084, 31.7757, 32.7103, 30.8411, 29.4393) = 31.21496$

Ionosphere

Bagging: $avg(9.4017, 9.1168, 9.6866, 8.8319, 7.9772) = 9.00284$
Ada-boosting: $avg(8.2621, 7.9772, 8.547, 7.6923, 8.8319) = 8.2621$

Sonar

Bagging: $avg(15.3846, 16.3462, 15.8654, 14.4231, 14.9038) = 15.38462$
Ada-boosting: $avg(13.4615, 12.5, 12.9808, 14.4231, 13.9423) = 13.46154$

Opitz and Maclin [11] concludes the empirical evaluation of Bagging and Boosting for neural networks that a Bagging ensemble nearly always outperforms a single neural network classifier. This is due to the case of the sonar dataset mentioned above since it breaks the error rate decrement pattern when applying Bagging. However, according to Table 9 and 11, the experiments for question 9.5 and 9.6 can be concluded that a Bagging ensemble always outperforms a single neural network classifier since the error rates always decrease after applying Bagging.

In the case of Boosting, a Boosting ensemble can significantly outperform both a single neural network classifier and Bagging. Nevertheless, it may produce no gain or a decrease in performance for some datasets, such as the diabetes dataset in this case. According to [11], Boosting may suffer from overfitting caused by the presence of noise in the dataset, which may explain an increase in error rate.

Overall, according to the experimental results, one can draw a conclusion that Bagging is probably suitable for most problems. However, for some problems, Boosting may be more appropriate as it tends to yield larger gains in overall performance.

Note: The conclusion applies to both question 9.5 and 9.6.

References

[1] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[2] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[3] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[4] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, 2012.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016.

[6] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[7] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the 30th International Conference on Machine Learning*, vol. 30, no. 1, 2013.

[8] A. Karpathy, "Convolutional Neural Networks (CNNs/ConvNets)," http://cs231n.github.io/convolutional-networks/, 2017, accessed: 2017-05-23.

[9] S. Hijazi, R. Kumar, and C. Rowen, "Using convolutional neural networks for image recognition," 2015.

[10] "SVC parameters when using RBF kernel," https://chrisalbon.com/machine-learning/svc_parameters_using_rbf_kernel.html, 2016, accessed: 2017-05-23.

[11] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999.

[12] Y. Freund and R. Schapire, "A short introduction to boosting," *Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[13] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[14] S. Marsland, *Machine learning: an algorithmic perspective*. CRC press, 2015.