

# Implementing and Evaluating Neural Networks

SCC.461 Programming for Data Scientists - Final Assignment

Harry Baines

35315878

`h.baines3@lancaster.ac.uk`

## Abstract

In this paper we implement and evaluate a neural network classifier from scratch using Python and compare it against a library implementation (Keras) capable of solving binary classification problems. Neural networks rely on a large set of preset hyperparameters which can directly influence classification performance. Hence we explore how changing these parameters and testing different neural network architectures affects classification results. We utilise k-fold cross-validation, a common re-sampling technique used to evaluate the skill of both classifiers. We use the R statistical package to facilitate data analysis and comparison of both classifiers. We find that the choice of optimisation algorithm and activation function have a significant influence on classification accuracy, with the Adam optimiser and ReLU activation function achieving the greatest validation accuracy for a dataset of sonar observations, with the learning rate, number of hidden layers and number of neurons in each layer also having varying effects on classification performance.

## 1 Introduction

### 1.1 Background

Neural networks are a set of mathematical algorithms loosely modelled after the human brain with the aim of recognising patterns in datasets. They are organised in layers of neurons which take inputs, perform some useful computation and output the result to the next layer of neurons. A neural network with a single hidden layer is commonly known as a shallow neural network, whereas a neural network with  $\geq 2$  hidden layers is considered to be a deep neural network. As the number of hidden layers is increased the number of layers of abstraction is also increased. For example in object recognition, layer 1 could represent individual pixel values, layer 2 could represent edges, layer 3 could represent shapes and so on until an entire object is formed. It is easy to notice an increase in the number of layers increases computational time and complexity.

These algorithms rely on a wide number of preset variables more commonly known as hyperparameters. This includes the choice of optimisation algorithm, activation functions and learning rates which must be tuned to the specific dataset under consideration. It is of paramount importance to explore the range of hyperparameter values such that an optimal model accuracy can be obtained following model training.

In this paper we use neural networks as a method to solve binary classification problems, that is, training a neural network on a set of labelled observations such that the model can produce reasonable predictions for binary class labels on unseen data. We explore different configurations of hyperparameters and evaluate the effect they have on classification performance. We utilise the Keras library [1], an open-source neural network library to facilitate the development of deep neural networks, to develop a library version of the classifier. Following implementation of both versions of these classifiers, we analyse the classification results using R to evaluate and draw conclusions of the effect of the model's hyperparameter values.

The neural network being developed from scratch in this paper is an object-oriented implementation of code based on a deep neural network for image classification found in the first course of the Deep Learning Specialisation on Coursera [2].

### 1.2 Sonar Dataset

Due to the mathematical and technical complexity of these algorithms as well as the analysis we will undertake, we consider a single dataset of sonar signals, consisting of 208 observations and 60 features

each with a single class label. The dataset contains 111 patterns obtained by bouncing sonar signals off a metal cylinder at various angles under various conditions, and 97 patterns obtained from rocks under similar conditions.

Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time. The label associated with each record contains the letter “R” if the object is a rock and “M” if it is a mine (metal cylinder) [3].

### 1.3 Research Questions

In this paper we aim to answer the following questions for an implementation and library version of a neural network:

1. How does the choice of optimiser and activation function affect validation accuracy across different learning rates?
2. How does the number of hidden layers and the number of neurons in each layer affect validation accuracy?
3. Do the classifier evaluation metrics (accuracy, precision, recall, F1-score) differ significantly for the implemented and library versions of the classifier?
4. Is there enough statistical evidence to suggest there exists a difference between cross-validation means collected from the implemented and library versions of the classifier?

## 2 Methods

Artificial neural networks are a set of algorithms inspired by the human brain which endeavour to recognize patterns in sets of data. A neural network consists of multiple layers of neurons, with connections between neurons in different layers representing weights with biases also connected to each neuron.

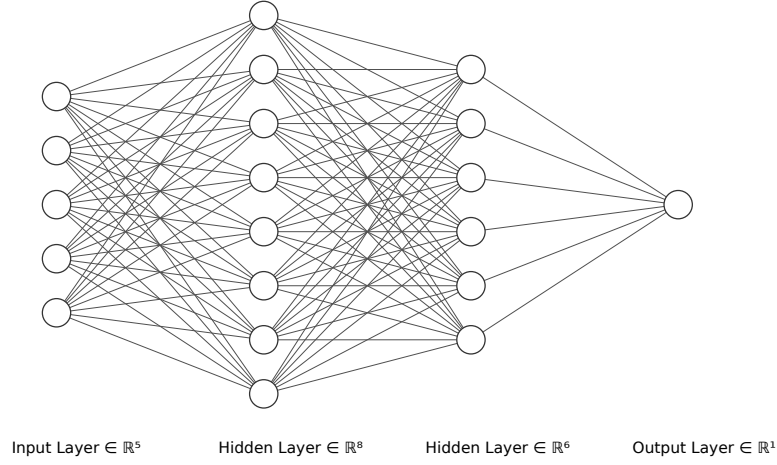


Figure 1: A 3-layer Neural Network

The number of neurons in the input layer is determined by the number of features present in the input dataset (i.e. a constant value). A single neuron is present in the output layer when binary classification problems are considered, with multiple neurons enabling multi-class classification problems to be solved. The number of hidden layers and the number of neurons in each hidden layer are hyperparameters one can tune to improve classification accuracy. Hidden layers are generally required if the data can only be separated by a non-linear decision boundary. As expected, increasing the number of hidden layers and neurons increases computational time and complexity due to a larger number of mathematical calculations.

We begin by computing a forward pass through the network, computing the weighted input to the neurons in layer  $l$ :

$$z^l = W^l a^{l-1} + b^l \quad (1)$$

where  $w^l$  represents a matrix of weights associated with the  $l^{th}$  layer,  $a^{l-1}$  is a matrix of activation values from the  $(l-1)^{th}$  layer and  $b^l$  is our biases vector in the  $l^{th}$  layer. This vectorised calculation results in a matrix of  $z$  values for each neuron in layer  $l$ , thus improving computational efficiency. We then input  $z$  to an activation function which produces the neuron's output to be passed to the following layer. The activation  $a^l$  of neurons in the  $l^{th}$  layer can be computed using  $z^l$  as calculated previously:

$$a^l = \sigma(z^l) \quad (2)$$

which results in a matrix of activation values for neurons in layer  $l$  [4]. For binary classification problems (2 class labels), the output layer tends to consist of a single neuron with the sigmoid activation function. This enables a probability value to be calculated, with the neuron's output existing in the range  $[0, 1]$ . We then compare the output value to a threshold value, above which a label of 1 will be assigned and 0 should the value be below (or equal to) the threshold.

For binary classification problems we utilise the cross-entropy loss function to predict a target label in the set  $\{0, 1\}$ . We compute the following equation given a predicted label  $\hat{y}^{(i)}$  and the true label  $y^{(i)}$  for a single training example:

$$L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})) \quad (3)$$

We then proceed to compute the cost function  $J(\theta)$ , which calculates the average of losses over all training examples:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) \quad (4)$$

where  $\theta$  is the set of network parameters (i.e.  $w$  and  $b$ ).

The next step is backpropagation, in which we adjust the weights and biases of the network by computing derivatives of the cost function with respect to the current set of weights and biases. We elide specific details of backpropagation due its mathematical complexity, although Andrew Trask gives a good description in his book [5]. Using these computed derivatives we can then update our network parameters using an optimisation algorithm and a set learning rate. We then repeat this iterative process multiple times and over time the network learns the optimal set of values for the weights and biases for the provided dataset. Hence should we provide the network unseen data, it can make a reasonable predictions for its class labels.

We utilised k-fold cross-validation to obtain k separate accuracy values evaluated on test sets during the cross-validation loop. We compute the mean of these values which represents our final validation accuracy value. In our analysis we set  $k$  to 10, although this value can be adjusted accordingly.

## 2.1 Activation Functions

Activation functions are responsible for transforming a value to a smaller range, typically between 0 and 1. Typically, a single sigmoid function in the output layer enables a prediction to be computed as a probability value between 0 and 1 for binary classification problems. The choice of activation functions in the hidden layers can be chosen before training.

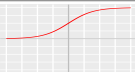
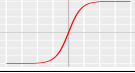
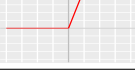
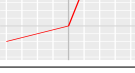
| Activation Func. | Equation  | Derivative   | Plot  |
|------------------|---|--|---|
| Sigmoid          | $f(x) = \frac{1}{1+e^{-x}}$   | $f'(x) = f(x)(1 - f(x))$   |   |
| Tanh             | $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$  | $f'(x) = 1 - f(x)^2$   |  |
| ReLU             | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$        | $f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$      |  |
| Leaky ReLU       | $f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |  |

Table 1: Activation functions used in our analysis.

Derivatives of activation functions are required so that we can compute gradient values of the error curve, such that we can adjust the weights of the network in order to minimise the error function.

## 2.2 Optimisation Algorithms

### 2.2.1 Gradient Descent Optimiser

Gradient descent (batch gradient descent) is the most common machine learning optimisation algorithm. On every iteration of gradient descent we update the network's parameters in the negative direction of the gradient of the cost function w.r.t the parameters, where the gradient gives the direction of steepest ascent [6]. The parameters of the network are updated using the following equations:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta) \quad (5)$$

where  $J(\theta)$  represents the cost function described in Equation 4.

Many variations of gradient descent exist, including stochastic gradient descent, in which we compute parameter updates for each training example and mini-batch gradient descent which computes network parameter updates for each mini-batch. Stochastic gradient descent takes significantly longer to run with larger training datasets compared to mini-batch gradient descent, hence we use the latter in our implementation and analysis. Mini-batch gradient descent uses the following equation for each mini-batch of  $n$  training examples:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \quad (6)$$

The size of each mini-batch,  $n$ , is a tunable hyperparameter configured by the programmer to a value such as 32 or 64. A larger mini-batch size tends to result in degradation of the model's quality in terms of its ability to generalize [7]. Due to the large number of possible hyperparameters we could tune in our analysis, we used a fixed mini-batch size of 64 in our implementation.

### 2.2.2 RMSprop Optimiser

RMSprop is an unpublished adaptive learning rate optimiser developed to resolve the radically diminishing learning rates of another optimiser known as Adagrad [8]:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2 \quad (7)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \quad (8)$$

This algorithm divides the learning rate by an exponentially decaying average of squared gradients, with recommended values of 0.9 and 0.001 for the hyperparameters  $\gamma$  and  $\eta$  respectively.

### 2.2.3 Adam Optimiser

As explained in the original paper [9], Adam (Adaptive Moment Estimation) is a stochastic optimisation algorithm which computes adaptive learning rates for different parameters from estimates of the first and second moments of gradients. The Adam optimiser is based on the following equations:

$$v_t = \beta_1 \times v_{t-1} - (1 - \beta_1) \times g_t \quad (9)$$

$$s_t = \beta_2 \times s_{t-1} - (1 - \beta_2) \times g_t^2 \quad (10)$$

$$\Delta\omega_t = \eta \frac{v_t}{\sqrt{s_t + \epsilon}} \times g_t \quad (11)$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t \quad (12)$$

where  $\eta$  is the initial learning rate,  $g_t$  is the gradient at time  $t$  along  $\omega^j$ ,  $v_t$  is the exponential average of gradients along  $\omega_j$ ,  $s_t$  is the exponential average of squares of gradients along  $\omega_j$ , and  $\beta_1, \beta_2$  are hyperparameters with typical values of 0.9 and 0.999 respectively [10].

The advantages of this optimiser are that it is relatively easy to implement, requires little tuning to the hyperparameters of the algorithm and is suitable for problems with noisy or sparse gradients.

## 2.3 Statistical Testing

Given we utilised 10-fold cross-validation in our analysis to evaluate the skill of the implemented and library versions of the classifier, an unmodified paired Student t-test can be used to compare the mean validation accuracies obtained following cross-validation. The null and alternative hypotheses for this test are given as follows:

$$H_0 : \mu_D = \mu_1 - \mu_2 = 0$$

$$H_1 : \mu_D = \mu_1 - \mu_2 \neq 0$$

The null hypothesis states there is no difference between the means, whereas the alternative hypothesis states there is a difference between the means. In order to test these hypotheses, we calculate a test statistic using the following equation:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}} \quad (13)$$

where  $\bar{d}$  is the mean of the differences between each pair of means,  $s_d$  is the standard deviation of the differences and  $n$  is the total number of means. Under the null hypothesis, the t statistic follows a t

distribution with  $n - 1$  degrees of freedom. Using a t-distribution table enables a  $p$ -value to be obtained for the test. A small  $p$ -value (i.e. less than 0.05 if a 95% confidence level is used) leads to rejection of the null hypothesis. In our analysis we utilise this test to evaluate if there is a significant difference between the validation accuracy means obtained following 10-fold cross-validation for all implemented optimisers as described in section 2.2.

Despite this method having a high type 1 error (i.e. rejection of a true null hypothesis), it has good repeatability with a reasonable type 2 error (i.e. non-rejection of a false null hypothesis). Another option not explored in this paper could be to use McNemar’s test, which uses a 2x2 contingency table of classification results obtained from a single run [11]. Non-parametric paired tests, such as the Wilcoxon signed-rank test (non-parametric version of the Student’s t-test) could also be used if the independence assumption in the original t-test is violated [12].

Given the large number of statistical methods available, there exists no single ideal procedure which satisfies all required constraints. In practice, many statistical models prove not to be perfect when data is limited. Although, the results of the tests can provide confidence intervals which become useful when interpreting and comparing different learning procedures [13].

## 3 Results and Discussion

### 3.1 Analysing Optimisers, Activation Functions and Learning Rates

In this section we analyse and compare the effect optimisation algorithms, activation functions and the learning rate have on the validation accuracy of neural networks following cross-validation. In our analysis we utilise 10-fold cross validation, meaning the sonar data under consideration will be split into 10 groups, with a single group on each iteration representing the validation set and the others comprising the training set. Following 10 iterations each group will have been part of the training and validation sets, hence we reduce the chance of overfitting the model. We compute the mean validation accuracy from the 10 accuracies obtained from 10-fold cross-validation and construct error bars using a 95% confidence interval of the mean validation accuracies to enable comparisons between means.

The optimisers under consideration, mini-batch gradient descent, Adam and RMSprop, will be analysed alongside Leaky ReLU, ReLU, Sigmoid and Tanh activation functions for neurons in the hidden layers. The learning rate, a key hyperparameter that must be tuned to the specific problem at hand, will be adjusted from 0.001 to 1 at 4 distinct values. A fixed mini-batch size of 64 for computing parameter updates over a total of 2000 epochs will be used. We consider a single hidden layer containing 5 neurons with varying activation functions as previously described.

#### 3.1.1 Mini-batch Gradient Descent Optimiser

In the following plot we compare the validation accuracies obtained following 10-fold cross validation for both the implemented and library neural networks using the mini-batch gradient descent optimiser:

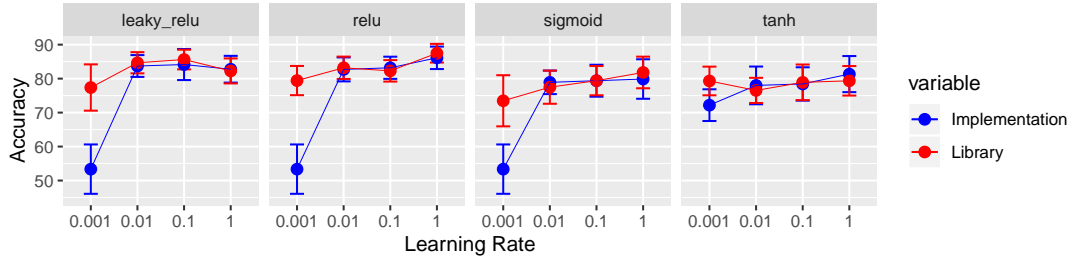


Figure 2: Mini-batch gradient descent optimiser accuracies following 10-fold CV against learning rates for different activation functions.

In Figure 2 we notice significantly lower validation accuracies for the implemented version with a learning rate of 0.001 compared to the library implementation. We notice that the validation accuracy either improves or remains constant at larger learning rates across activation functions for both versions. The implemented version generally showed larger error margins at smaller learning rates indicating the true mean value lies within a larger range. We notice at larger learning rates the ReLU activation function achieves the greatest validation accuracy with narrower error margins.

#### 3.1.2 RMSprop Optimiser

In the following plot we compare the validation accuracies obtained following 10-fold cross validation for both the implemented and library neural networks using the RMSprop optimiser:

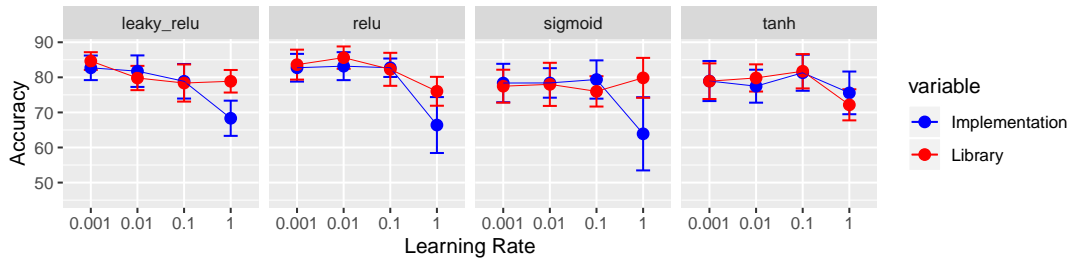


Figure 3: RMSprop optimiser accuracies following 10-fold CV against learning rates for different activation functions.

In Figure 3 we notice validation accuracies decrease with larger learning rates particularly for the implemented version. The ReLU activation function achieved a peak validation accuracy with a learning rate of 0.01 for both the implemented and library versions. The implemented version generally showed larger error margins at larger learning rates indicating the true mean value lies within a larger range.

### 3.1.3 Adam Optimiser

In the following plot we compare the validation accuracies obtained following 10-fold cross validation for both the implemented and library neural networks using the Adam optimiser:

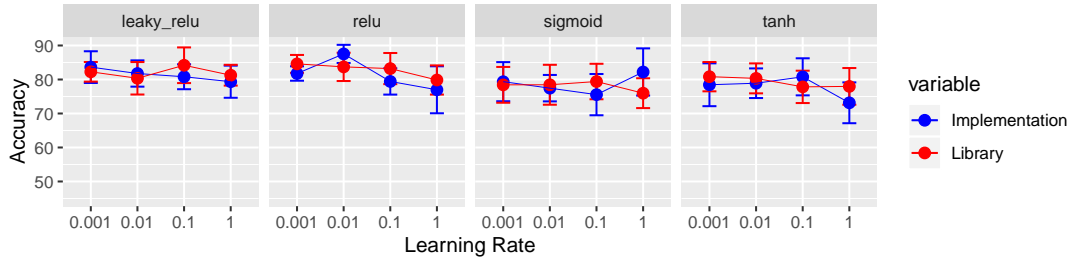


Figure 4: Adam optimiser accuracies following 10-fold CV against learning rates for different activation functions.

In Figure 4 we generally notice validation accuracies decrease by small amounts when the learning rate is increased. Again, the ReLU activation function achieves the highest validation accuracy compared to the other activation functions at a learning rate of 0.01. Both the implemented and library versions are similar in terms of the computed cross-validation accuracies. Interestingly the Adam optimiser showed superior validation accuracies across all tested learning rates and activation functions compared to the RMSprop and mini-batch gradient descent optimisers.

### 3.1.4 Summary

From these results we can conclude the Adam optimiser with a learning rate of 0.01 and the ReLU activation function achieves the highest validation accuracy for the sonar dataset. In the following sections we use these hyperparameter configurations for further analysis. The analysis conducted here clearly emphasizes the fact that the effect a learning rate has on a given optimiser and activation function can significantly affect the validation accuracy of the developed model and must therefore be chosen carefully in order to maximise validation accuracy.

## 3.2 Analysing Hidden Layers

In this section we analyse how adjusting the number of neurons in hidden layers affects validation accuracy. Increasing the number of neurons in hidden layers increases the total number of mathematical computations, hence we analyse a neural network with a single hidden layer containing 1 to 10 neurons (10 separate models), followed by a neural network with 2 hidden layers, with the first hidden layer containing 10 neurons and the second hidden layer containing 1 to 10 neurons (10 separate models). In our previous analysis we found the ReLU activation function with a learning rate of 0.01 achieved peak validation accuracy across all optimisers, hence we use this function for neurons in the hidden layers and 0.01 as a fixed learning rate. We continue to use a fixed mini-batch size of 64 over 2000 epochs.

### 3.2.1 Single Hidden Layer

The following plot shows the results of increasing the number of neurons from 1 to 10 in a single hidden layer for different optimisers:



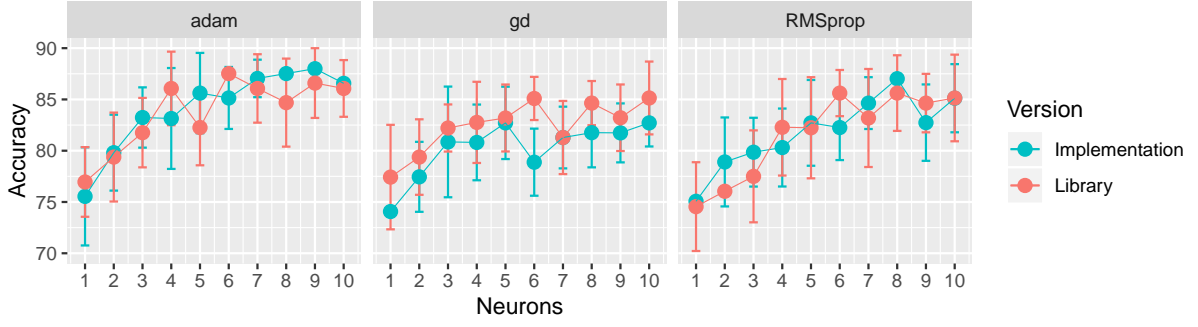


Figure 5: Validation accuracy against number of neurons in a single hidden layer (implemented and library versions).

In Figure 5 we notice a correlation between the number of neurons in the hidden layer and the validation accuracy - having a larger number of neurons in the hidden layer achieves greater validation accuracies for both the implemented and library versions of the neural network. The Adam optimiser with the previously mentioned hyperparameters achieves greater validation accuracies when neurons are increased, with a peak validation accuracy of 87.5% for the library implementation with 6 neurons, and 88% for the implemented version with 9 neurons. Interestingly, the validation accuracy decreases for the library implementation after 6 neurons which could potentially indicate overfitting. The RMSprop optimiser performed worse with a small number of neurons compared to mini-batch gradient descent and Adam optimisers, although validation accuracies were relatively similar.

### 3.2.2 2 Hidden Layers

The following plot shows the results of increasing the number of neurons from 1 to 10 in the second hidden layer (with 10 neurons in the first hidden layer) for different optimisers:

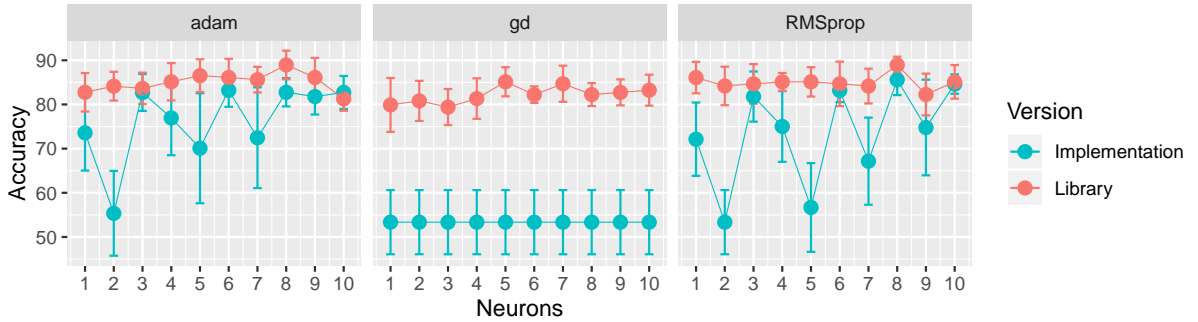


Figure 6: Validation accuracy against number of neurons in the second hidden layer (implemented and library versions).

In Figure 6 we notice a very slight increase in validation accuracy for the library version when the number of neurons in the second hidden layer is increased. The accuracy for the implemented version varied significantly as the number of neurons increased for both the Adam and RMSprop optimisers. Interestingly, the validation accuracy remained constant at around 53% for the mini-batch gradient descent optimiser in the implemented version which means increasing the number of neurons had no effect on the final validation accuracy. From these results we notice the Adam optimiser achieves a higher validation accuracy compared to the other optimisers for the implemented and library versions. We can also conclude the effect of the second hidden layer for the library version did not significantly improve validation accuracy when compared just a single hidden layer containing 10 neurons (see Figure 5).

### 3.2.3 Summary

In the previous analysis we saw how increasing the number of neurons in hidden layers of a neural network affects validation accuracy. For the sonar dataset under consideration, a single hidden layer with

10 neurons may be sufficient to achieve a reasonable validation accuracy. Adding an extra hidden layer had little effect on the library version, including worse validation accuracies for the implemented version.

### 3.3 Classification Performance

In this section we analyse the evaluation metric scores obtained following classification for both a single hidden layer and 2 hidden layers in the neural network. The following metrics will be used to analyse classification performance for both the implemented and library versions of the neural network:

- Accuracy ( $\frac{TP+TN}{TP+TN+FP+FN}$ ): a ratio of correctly predicted observations to the total number of observations
- Precision ( $\frac{TP}{TP+FP}$ ): a ratio of correctly predicted positive observations to the total number of predicted positive observations
- Recall ( $\frac{TP}{TP+FN}$ ): a ratio of correctly predicted positive observations to all observations in actual positive class
- F1-score ( $2 \times \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$ ): a weighted average of precision and recall

Following 10-fold cross-validation we compute the average of these collected metrics over the 10 iterations.

#### 3.3.1 Single Hidden Layer

The following plot shows the average evaluation metrics for a neural network with a single hidden layer with 10 neurons each with the ReLU activation function, and the Adam optimiser with a learning rate of 0.01 over 2000 epochs:

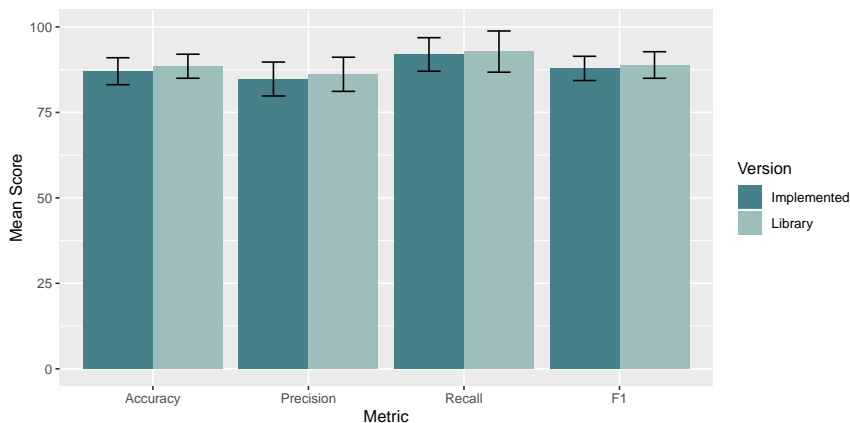


Figure 7: Bar plot of average classification metric scores for a single hidden layer.

In Figure 7 we notice overlapping error bars for both the implemented and library versions across all metrics. Although this doesn't mean the means between versions are statistically significant, it is important to note we obtain similar average evaluation metrics meaning the implemented version is a good approximation of the more sophisticated library version.

#### 3.3.2 2 Hidden Layers

The following plot shows the average evaluation metrics for a neural network with a hidden layer with 10 neurons followed by a hidden layer with 10 neurons each with the ReLU activation function, and the Adam optimiser with a learning rate of 0.01 over 2000 epochs:

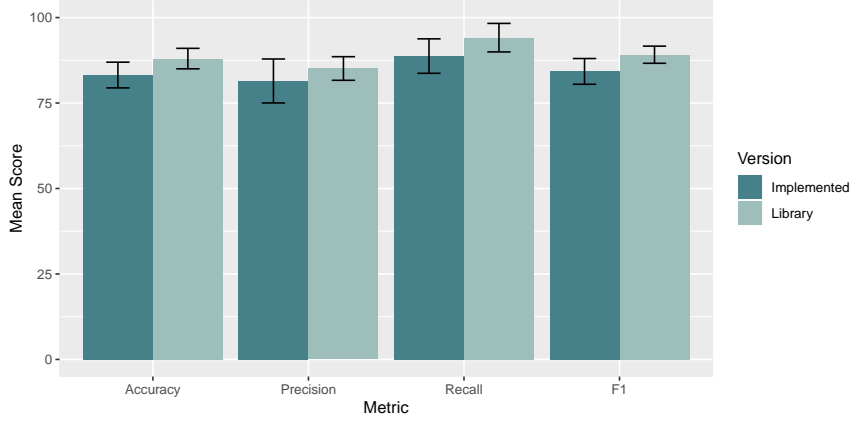


Figure 8: Bar plot of average classification metric scores for 2 hidden layers.

In Figure 8 we notice a similar result as shown in Figure 7, although in this result we notice the difference between each average evaluation metric for the implemented and library versions are generally further apart.

### 3.3.3 ROC Plots

An ROC curve (receiver operating characteristic curve) shows the performance of a classification model at all classification thresholds and plots two parameters against each other: True Positive Rate ( $\frac{TP}{TP+FN}$ ) and False Positive Rate ( $\frac{FP}{FP+TN}$ ). Increasing the classification threshold classifies more items in the negative class, which causes the number of true positives (TP) to decrease or remain constant, and the number of false negatives (FP) to increase or remain constant. Therefore, recall will either decrease or remain constant as recall is dependent on both true positives and false negatives. Precision will increase as the number of false positives decreases. Decreasing the classification threshold classifies more items in the positive class, which increases the number of false positives (FP) and true positives (TP). The AUC (area under the ROC curve) enables us to compute all points on the ROC curve, and provides an aggregate measure of performance across all classification thresholds. When comparing classification models, the ideal model has an ROC close to the top left corner, with a perfect classifier having an AUC of 1.0.

The following ROC plot shows how TPR and FPR vary over different classification thresholds for a single hidden layer with 10 neurons all using the ReLU activation function, the Adam optimiser with a learning rate of 0.01, over 2000 epochs:

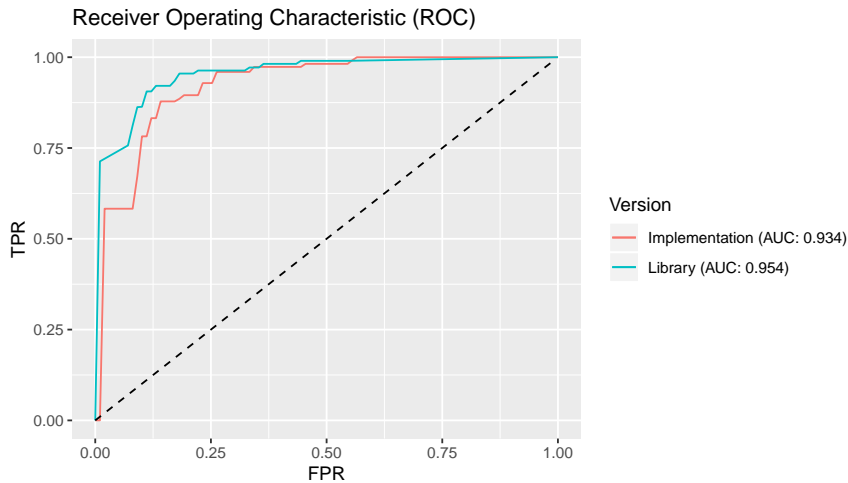


Figure 9: ROC plot for a neural network with a single hidden layer.

In Figure 9 we see the library version of the classifier is marginally better than the implemented

version as the AUC value of the library version is 0.95 and the AUC of the implemented version is 0.93.

The following ROC plot shows how TPR and FPR vary over different classification thresholds for a hidden layer with 10 neurons followed by another hidden layer with 10 neurons all using the ReLU activation function, the Adam optimiser with a learning rate of 0.01, over 2000 epochs:

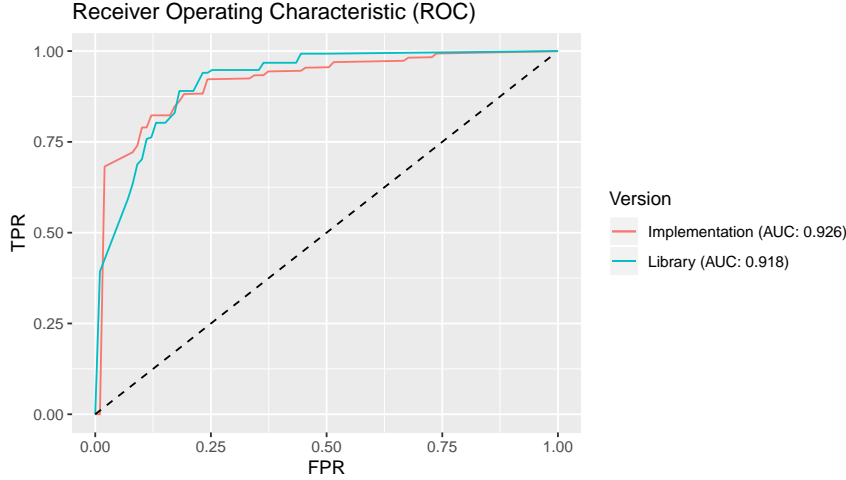


Figure 10: ROC plot for a neural network with 2 hidden layers.

Interestingly in Figure 10 both the implemented and library versions of the classifier have smaller AUC values compared to when a single hidden layer was used (i.e. both classifiers performed better with only a single hidden layer).

### 3.4 Comparing Implementation and Library Neural Networks

In this section we run a paired Student t-test to evaluate if there is a difference between the set of validation accuracy means collected from 10-fold cross-validation for all of the implemented optimisers for both the implemented and library versions of the neural network. We use the results obtained from our hidden layers analysis in section 3.2 to perform our analysis.

#### 3.4.1 Mini-batch Gradient Descent Optimiser

Following a paired t-test for the cross-validation accuracy means for the implemented and library versions of the classifier using the mini-batch gradient descent optimiser we obtain a  $p$ -value of 0.003. Given a 95% confidence level, we can say we have sufficient evidence to reject the null hypothesis  $H_0$  and accept the alternative hypothesis  $H_1$  (that is, our  $p$ -value is smaller than 0.05). This means there is enough evidence to suggest there is a difference between cross-validation accuracy means for the implemented and library versions of the classifier for this optimiser (see Appendix A for R results).

#### 3.4.2 RMSprop Optimiser

Following a paired t-test for the cross-validation accuracy means for the implemented and library versions of the classifier using the RMSprop optimiser we obtain a  $p$ -value of 0.7835. This value is relatively large, hence we fail to reject the null hypothesis  $H_0$ . This means that there is not enough evidence to suggest there is a significant difference of the cross-validation accuracy means for the implemented and library versions of the classifier for this optimiser (see Appendix B for R results).

#### 3.4.3 Adam Optimiser

Following a paired t-test for the cross-validation accuracy means for the implemented and library versions of the classifier using the Adam optimiser we obtain a  $p$ -value of 0.5347. This value is relatively large, hence we fail to reject the null hypothesis  $H_0$ . This means that there is not enough evidence to suggest there is a significant difference of the cross-validation accuracy means for the implemented and library versions of the classifier for this optimiser (see Appendix C for R results).

## 4 Conclusion

In this paper we evaluated the performance of an implemented deep neural network classifier against a library implementation, including analysis of how different configurations of hyperparameters affects the final classification accuracy.

### 4.1 Aims and Objectives

1. *How does the choice of optimiser and activation function affect validation accuracy across different learning rates?*

In section 3.1 we explored different optimisers, notably mini-batch gradient descent, RMSprop and Adam optimisers. We analysed the effect that changing the learning rate, activation function and optimiser had on k-fold cross-validation accuracy. We found the Adam optimiser, the ReLU activation function and a learning rate of 0.01 were the optimal set of hyperparameters for the sonar dataset under consideration.

2. *How does the number of hidden layers and the number of neurons in each layer affect validation accuracy?*

In section 3.2 we explored how increasing the number of hidden layers and the number of neurons in each layer affected cross-validation accuracy. We found the Adam optimiser was generally the best optimiser in this case.

3. *Do the classifier evaluation metrics (accuracy, precision, recall, F1-score) differ significantly for the implemented and library versions of the classifier?*

In section 3.3 we computed classification metrics and plotted ROC curves to evaluate the performance of the implemented and library versions of the classifier and found the metrics differed more with 2 hidden layers compared to a single hidden layer.

4. *Is there enough statistical evidence to suggest there exists a difference between cross-validation means collected from the implemented and library versions of the classifier?*

In section 3.4 we carried out a paired Student's t-test to compare the cross-validation means collected for both versions of the classifier using different optimisers, and we found only the mini-batch gradient descent optimiser showed the means were statistically different.

### 4.2 Future Work

We envisage future analysis could involve any of the following:

- Implementation of further optimisation algorithms, such as AdaGrad, AdaDelta or Stochastic Gradient Descent with Momentum
- Implementation of further activation functions, such as Randomized Leaky ReLU, Radial Basis Functions, Softmax or Exponential Linear Units
- Implementation of further initialisation techniques, such as zero, He or Xavier initialisation as opposed to simple random initialisation of network parameters
- Implementation of further types of cross-validation, such as leave-one-out cross-validation or exhaustive cross-validation
- Developing the implemented neural network model further to facilitate multi-class classification of class labels
- Neural architecture search (NAS) analysis to facilitate the design of the neural network's architecture
- Regularisation methods to prevent over-fitting of the model, such as L2 or dropout
- A more preferable statistical test to compare cross-validation accuracy means collected for the implemented and library versions of the classifier such as McNemar's test
- Apply both implemented and library versions of the classifier to datasets containing a large number of observations

## References

- [1] “Keras: The python deep learning library.” [Online]. Available: <https://keras.io/>
- [2] “Neural networks and deep learning.” [Online]. Available: <https://www.coursera.org/learn/neural-networks-deep-learning?specialization=deep-learning>
- [3] “Uci machine learning repository: Connectionist bench (sonar, mines vs. rocks) data set.” [Online]. Available: [https://archive.ics.uci.edu/ml/datasets/ConnectionistBench\(Sonar,Minesvs.Rocks\)](https://archive.ics.uci.edu/ml/datasets/ConnectionistBench(Sonar,Minesvs.Rocks))
- [4] Nielsen and M. A., “Neural networks and deep learning,” Jan 2018. [Online]. Available: <http://neuralnetworksanddeeplearning.com/chap2.html>
- [5] A. Trask, *Grokking Deep Learning*, 1st ed. USA: Manning Publications Co., 2019.
- [6] I. Dabbura, “Gradient descent algorithm and its variants,” Sep 2019. [Online]. Available: <https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3>
- [7] Keskar, N. Shirish, Jorge, Mikhail, Tang, and P. T. Peter, “On large-batch training for deep learning: Generalization gap and sharp minima,” Feb 2017. [Online]. Available: <https://arxiv.org/abs/1609.04836>
- [8] S. Ruder, “An overview of gradient descent optimization algorithms,” *CoRR*, vol. abs/1609.04747, 2016. [Online]. Available: <http://arxiv.org/abs/1609.04747>
- [9] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.
- [10] A. Kathuria, “Intro to optimization in deep learning: Momentum, rmsprop and adam,” Aug 2018. [Online]. Available: <https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/>
- [11] T. G. Dietterich, “Approximate statistical tests for comparing supervised classification learning algorithms,” *Neural Computation*, vol. 10, no. 7, pp. 1895–1923, 1998. [Online]. Available: <https://doi.org/10.1162/089976698300017197>
- [12] Janez, “Statistical comparisons of classifiers over multiple data sets,” Jan 2006. [Online]. Available: <http://www.jmlr.org/papers/v7/demsar06a.html>
- [13] T. M. Mitchell, “Machine learning (mcgraw-hill international editions computer science series),” Jan 1997. [Online]. Available: <https://www.abebooks.com/9780071154673/Machine-Learning-McGraw-Hill-International-Editions-0071154671/plp>

## Appendix

### A Mini-batch Gradient Descent Optimiser Paired t-test R Results

```
> t.test(df2$AccuracyMean1, df2$AccuracyMean2, paired = TRUE, alternative = "two.sided")

Paired t-test

data: df2$AccuracyMean1 and df2$AccuracyMean2
t = -4.0185, df = 9, p-value = 0.003025
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.4496079 -0.9646778
sample estimates:
mean of the differences
 -2.207143
```

### B RMSprop Optimiser Paired t-test R Results

```
> t.test(df2$AccuracyMean1, df2$AccuracyMean2, paired = TRUE, alternative = "two.sided")

Paired t-test

data: df2$AccuracyMean1 and df2$AccuracyMean2
t = 0.28306, df = 9, p-value = 0.7835
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.265183 1.627088
sample estimates:
mean of the differences
 0.1809524
```

## C Adam Optimiser Paired t-test R Results

```
> t.test(df2$AccuracyMean1, df2$AccuracyMean2, paired = TRUE, alternative = "two.sided")

Paired t-test

data: df2$AccuracyMean1 and df2$AccuracyMean2
t = 0.64545, df = 9, p-value = 0.5347
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.067514  1.919895
sample estimates:
mean of the differences
      0.4261905
```