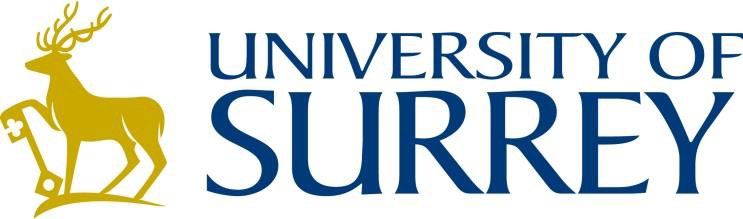
Neural Network Simulation using Matlab for Understanding how to solve pattern recognition problems using backpropagation

Vaishali Shivakumar (6738581)

*Assignment Report for AI and AI Programming (EEEM005)*



Computer Vision, Robotics and Machine Learning

*Department of Electronic Engineering* Faculty of Engineering and Physical Sciences University of Surrey

Guildford, Surrey, GU2 7XH, UK

Table of Contents

[Executive Summary 3](#_bookmark0)

[Introduction 3](#_bookmark1)

1. [Experiment 1 4](#_bookmark2)
   1. [Overview 4](#_bookmark3)
   2. [The procedure 4](#_bookmark4)
   3. [Results and Analysis 4](#_bookmark5)
2. [Experiment 2 8](#_bookmark6)
   1. [Overview 8](#_bookmark7)
   2. [The procedure 8](#_bookmark8)
   3. [Results and Analysis 9](#_bookmark9)
3. [Experiment 3 12](#_bookmark10)
   1. [Overview 12](#_bookmark11)
   2. [Procedure 12](#_bookmark12)
   3. [Results and Analysis 13](#_bookmark13)
4. [Experiment 4 18](#_bookmark14)
   1. [Overview 18](#_bookmark15)
   2. [Procedure 18](#_bookmark16)
   3. [Results & Analysis 18](#_bookmark17)

[Conclusion 19](#_bookmark18)

[Citations 19](#_bookmark19)

[Appendix 19](#_bookmark20)

# Executive Summary

This report focuses on using a multi-layer perceptron (MLP) to solve pattern recognition problems, specifically using the cancer dataset. The experiments in this report vary the number of hidden nodes and epochs to determine their effect on the MLP's learning capacity and robustness. After testing 21 different combinations, it was found that the best combination was 32 hidden layers and 16 epochs. This combination was then used as a basic classifier in experiments 2 and 3, which investigated ensemble learning and the effect of optimizers on MLP learning, respectively. It was found that ensembles with higher base classifiers and the trainrp optimizer performed better. Experiment 4 explored the use of an MLP to produce nonlinear decision boundaries for overlapping bivariate classes. The results showed that the plotted decision boundary was close to the optimal decision boundary with a small Euclidean distance. Overall, this report highlights the importance of varying the number of hidden nodes and epochs in an MLP and the effectiveness of ensemble learning and different optimizers in improving MLP performance.

# Introduction

The process of pattern recognition is essential to machine learning, which involves analyzing and understanding patterns and correlations among the data to extract new information. Neural networks are powerful tools for pattern recognition, capable of learning complex mathematical functions and mapping input data to output values. Neural networks use a single hidden layer to learn these functions, and the universal approximation theorem proves that these networks can learn any non-linear function. The backpropagation algorithm is used to train these neural networks, which involves feeding data through the network in a forward direction, calculating the loss, and then backpropagating the error to update the weights and biases of the network.

In this assignment, we focus on how neural networks can solve pattern recognition problems in a cancer dataset. The first part of the assignment involves creating an optimal neural network with one layer by experimenting with the number of epochs and hidden nodes. The second part builds upon the first by using the optimal network to experiment with ensemble techniques. The third part of the assignment repeats the experiments with different optimizers. Finally, the fourth part focuses on distinguishing between two equiprobable, overlapping classes in two dimensions using neural networks. Throughout the assignment, MATLAB is used to implement and analyze the neural networks, experimenting with different parameters to optimize the performance of the networks. Overall, this assignment provides a comprehensive overview of how neural networks can be used for pattern recognition and the importance of experimentation to optimize the performance of these networks.

# Experiment 1:

## Overview

The experiment is about training neural networks with a cancer dataset available in the MATLAB repository. This network is built using the nntools library of MATLAB, which is a multi-layer perceptron with one hidden layer. The nntools is a flexible platform to build and train neural networks with less coding. The nntools app enables to experiment with different parameters. After satisfactory parameters are obtained, the script for the neural network can be obtained using the script generation function in the tool and later optimized in the MATLAB workspace. The dataset is randomly split into 50/50 ratios for training and testing the neural network. The model experiments against different epochs-hidden layer combinations.

## The procedure

The experiment is done for a combination of 3 different hidden layers with neurons count of 2, 8 and 32 and with different epochs values as 1, 2, 4, 8, 16, 32 and 64. Thus a total of 21 combinations of results can be obtained. The early stopping is set to 0 so that the model can run for the specified number of epochs. Each of these combinations is run 30 times with different values of train-test split ratios. After running 30 times for each of these combinations the mean and standard deviation of the classification error are computed which is then used to evaluate the performance of the model. This procedure is repeated for all the other combinations as well and finally, 21 values of the mean and standard deviation of the classification error are obtained. Finally, the mean values are plotted against the number of epochs for visualization and analysis.

## Results and Analysis

The first model with 2 hidden layers starts with the same test and train classification error rates of 0.552818 and however the train and test error rates start to deviate from the 8th epoch.

Then from the 16th epoch the error rates become stable, but the test error rate is found to be above the train error rate. Overall, the model has the least test error rate of 0.08538 at the 16th Epoch and least train Classification error rate of 0.06857 at the 16th Epoch.

The model with 8 hidden layers, it starts off with classification error rate of 0.346896 however the both the train Classification error rate and the test classification error rates dips sharply at the 4th Epoch indicating the fact that model can learn quickly as the nodes are increased. From the graph it can be ascertained that the error rates become stable from the 4th epoch and the test classification error rate beings to deviate from the tarin classification error rate and lies above the train classification error rate indicating overfitting of the model. As the epoch increases there is a clear difference in train and test classification error rates.

In the model with 32 hidden layers the dip in error rate is gradual and no sharp dip is found. However, from the 8th epoch the difference between the classification error rates of the train and test data can be clearly seen and the test data lies well above the train data indicating overfitting of the model from the 8th epoch. But the model reached the global minimum value of 0.0577143 for the train classification error rate at the 16th epoch and 0.0782741 for the test classification error rate at the 16th epoch.

Overfitting occurs when the model learns too much from the training data which causes poor performance on the test data. This happens because the model learns the noise present in the training data. Thus, the model begins to show high variance when overfitting occurs.

The graph above and the one below shows the standard deviations for each epoch node combination for 30 iterations. Both the train data and the test data follow a similar pattern.

From the first experiment, the optimum hidden layer epoch combination is 32-16. This is because the training classification error rate has the global minimum value of 0.0577 and the test classification error rate has a minimum value of 0.0782. Using the above-identified combination. Experiment 2 is carried on.

# Experiment 2:

## Overview

This experiment is about using ensemble technique to study the variations in accuracy of the prediction. In ensemble learning multiple individual models are aggregated and the aggregated model output is used for prediction. The prediction is done based on majority voting. The majority voting is a technique where the predicted output will be the output class which majority of the classifiers in the ensemble had predicted. A base classifier is defined with the optimal values of epoch and hidden layers obtained from experiment 1. Then the same base classifier is used to create an ensemble of 15 models with random initial weights. They are treated as both individual classifiers and then as ensembles. Further, the experiment is carried out by varying the count of base classifiers, then with a different ensemble of different node epoch combinations to study the variations in the accuracy of the model.

## The procedure

This experiment is done in different stages. In the first stage, the individual classifiers are trained, and their individual accuracies are identified. Then the cumulative accuracy of the ensemble model is found using the ensemble technique. In the second stage, the number of base classifiers count is varied, the experiment is repeated, and the accuracy of individual ensemble models is calculated. In the third stage, the number of base classifiers count is kept constant, but the experiment is repeated for different node epoch combinations below the optimal combination and the accuracy of the ensemble model is computed. In the fourth stage, the number of base classifiers count is kept constant, but the experiment is repeated for different node epoch combinations above the optimal combination and the accuracy of the ensemble model is computed. In the final stage, the number of hidden layers and the number of base classifier count is kept constant, but the epoch is varied and then the accuracy of the ensemble model is computed. Later the accuracy of the ensemble model from the second stage to the final stage is plotted for visualization. In order to ascertain that any training data biases are alleviated, each model is trained 30 different times with 30 different training and test data combinations.

## Results and analysis

The above Bar graph depicts the accuracy of individual models in an ensemble to Train and test accuracies. Most of the models other than 1, 2 and 6 have similar train accuracy scores because the weights of the models are initialized randomly, and the random value seed each time depends on the iteration value. The average Mean train accuracy and Average Mean Test accuracy for this ensemble are 96.959% and 96.674% respectively. So, from the accuracy score it can be confirmed that despite the weights of the models being initialized randomly the ensemble has learned in a similar way. The learning rate is not affected by initializing the weights randomly.

In the second stage of the experiment, the epoch and node values are kept constant but now the experiment is conducted with an ensemble of classifiers. An ensemble of 10 classifiers in a range of 3-25 is randomly used for training the data and thereafter the accuracies of the individual ensemble are plotted, and the overall mean accuracy of all the ensembles is calculated. Here each of the ensembles is considered an individual model.

The accuracy varies as the number of counts of base classifiers in the ensemble. The ensemble performed well for higher count values and performed same as the base classifier at lower values for the test data. Beyond 15, the ensemble shows overfitting as the test accuracy value starts decreasing.

In the third stage, the number of base Classifier count is kept constant, but the experiment is repeated for different nodes, epoch combinations which are below optimum node-epoch combination of 32-16. The graph above represents the accuracy value of the ensemble of 15 Classifiers for 10 different Node Epoch Combinations which are lower than the optimum 32-16 combination. It can be seen from the graph that the ensemble model.

shows high variation. The lower values show poor performance. Thus, the ensemble has poor performance if both the hidden layers and epoch values are less. Node values 26 and beyond have shown much better performance complemented with similar epoch values. Also, most of the combinations where node epoch values were closer had similar train and test accuracies.

In the fourth stage, the experiment was repeated with different Epoch, Node combinations higher than optimal value with same number of base classifiers in the ensemble. The pairs with different epoch and hidden layer combinations exhibit the lowest performance, while the rest of the other combinations performed better. The combination with 78 nodes and 67 epochs has performed well, but the rest of the combinations show overfitting of the data.

In the fifth stage the experiment is repeated with constant number of nodes but different epoch values for this ensemble. In a recent experiment, the number of hidden neurons was fixed at 32, and the range of epochs was set between 1 and 100. It was observed that the model struggled to learn with only 2 epochs, leading to high error rates and lower accuracy. However, as the number of epochs increased, the model began to learn better.

Nevertheless, it was noted that after the optimal epoch was reached, the model began to overfit by learning the noise in the data. Each model was trained individually with a different epoch number, so the variation in train loss across epochs was due to random weight variability. Finally, an ensemble of these models was used to calculate accuracies for both train and test data.

In the experiment, an ensemble was created with multiple base classifiers, outperformed other ensemble. Furthermore, other ensembles also outperformed their respective individual classifiers. This highlights the effectiveness of ensemble techniques in improving accuracy and reducing classification errors.

# Experiment 3

## Overview

## Experiment 3 follows the same methodology as Experiment 2 but involves a change in optimizers. The optimizer 'trainscg' used in Experiment 2 is replaced with 'trainlm' and 'trainrp', and the training process is repeated.

## Procedure

The backpropagation algorithm updates the weights of neural networks, but it needs to choose the optimal weights to enable learning. Inappropriate weights can prevent the model from learning. The weights are chosen to minimize the loss, and this is accomplished using optimizers. Optimizers are algorithms that change the parameters of neural networks, such as learning rate, to minimize loss [1]. There are various optimizers available, including Gradient Descent, Stochastic Gradient Descent, Adam, Levenberg–Marquardt, RProp, RMS Prop, and Conjugate gradient.

Gradient Descent is one of the most important optimization algorithms and is considered the backbone of machine learning and deep learning. It is a first-order optimization algorithm that finds the direction of the steepest descent to minimize the loss function. In gradient descent, the goal is to reach the global minimum of the loss function based on several parameters, such as current weights, learning rate, and loss function. The algorithm decides how weights should be updated using the equation[2]

A(n+1) = A(n) − η⋅∇L(A)

where A(n+1) is a new weight, A(n) is the current weight, η is the learning rate, and L(A) is the loss function.

The negative sign in the equation represents how much the loss function is subtracted from the current weights so that the algorithm could go against the gradient, towards the local minima. If the gradients are steeper, the steps are larger, and if the gradients are flatter, the algorithm takes smaller steps, which is taken care of by the learning rate. In this experiment, the current learning rate is 0.01.

In Experiment 3, the optimizers used in Experiment 2 (trainscg) are replaced with Levenberg–Marquardt (trainlm) and RProp (trainrp) algorithms. The Levenberg–Marquardt algorithm is an iterative algorithm used to solve least-square problems where the systems are non-linear. It can find global minima if there are fewer local minima [3]. On the other hand, RProp algorithm is a popular algorithm that uses the sign and magnitude of gradients to update the weights. It can adapt well to each step and handle weights individually [4]. Using these algorithms changes the learning pattern of the base classifiers, which can be analyzed to evaluate the performance of the optimizers.

## results & analysis

In the beginning, the ensemble of models is trained using the trainlm optimizer. Upon analyzing the train and test errors, it is observed that the accuracy is much better than what was obtained using the trainscg optimizer. This trend is consistent across all the ensemble models with varying parameters. In particular, all base classifiers that have different epochs and hidden layers and lower than optimal layers and epochs have performed better than all the base classifiers with trainscg optimizer.

To compare the performance of the trainlm and trainrp optimizers on the ensemble of 15 classifiers with the same epoch and hidden layers, their graphs were analyzed. It was observed that trainlm resulted in higher train accuracy, but trainrp showed better test accuracy.

The previous set of graphs demonstrated a consistent pattern where trainlm performed better in terms of train accuracy, while trainrp performed better in terms of test accuracy, implying a slight overfitting. In contrast, the following graphs, which maintain a constant number of hidden nodes, show that trainrp consistently outperforms trainlm. However, there are a few instances where trainlm has a slight edge over trainrp, but they are isolated. Additionally, the gap between train and test data is smaller in the model trained using trainrp than in the model

trained using trainlm, indicating that the model trained with trainlm has more overfitting issues.

In the graph presented below, where an odd number of classifiers were used for different models, we see the same pattern as before. The overall best performance occurs when the number of ensemble models is higher. In both trainlm and trainrp, there is a performance improvement in test accuracy around the 21-25 range. In terms of learning, the same pattern continues, where trainrp shows signs of overfitting while trainlm has train and test accuracies closer to each other.

The above experiment suggests that trainrp optimizer outperforms trainlm and trainscg in terms of minimizing error and maximizing accuracy in test data with less overfitting. It can be considered as evidence that trainrp can adjust to the training data and converge better than other optimizers for this dataset.

# Experiment 4

## Overview

The experiment involves differentiating between two equi-probable classes of overlapping two-dimensional Gaussians. Class 1 has a mean 𝜇1 of [0,0] and variance 𝜎1, while Class 2 has a mean 𝜇2 of [2,0] and variance 𝜎2 of 4. It is known that the optimum boundary between the two classes is a circle centred at [−2/3,0] with a radius of 2.34. To solve the problem, random data points are generated for each class, and neural networks are utilized for model training.

## Procedure

To generate a dataset with two different classes and different mean and variance, the 'mvnrnd' function is used. Each class has 1650 two-dimensional arrays, and when concatenated, it creates a dataset of 3300 records. Two labels are generated for each class, concatenated, and one-hot encoded. The data is then randomly rearranged to ensure a normal distribution. The dataset is split into a 10% training set and a 90% testing set. The 'patternnet'

The algorithm is used to train the model with various epoch and hidden unit combinations. After experimentation, it is found that the best combination is 60 hidden units and 38 epochs. This combination is then used to generate an ensemble of base classifiers as in experiment 2.

## Results & Analysis

The data has been predicted and the mean and variance of class 2 have been calculated, resulting in a mean of [-0.1713, -0.0215] and a variance of 0.7162. Using these values, a circle has been plotted on the scatter plot of the predicted values, with red indicating class 1 and green indicating class 2. The decision boundary, represented by a small circle, has been calculated as [-2/3,0], with an optimal Euclidean distance between the origin and the obtained origin of 0.4671. This distance indicates that the values within the decision boundaries belong to the same class, providing evidence to support the theory that the decision boundary exists between [-2/3,0].

# Conclusion

From the four experiments we conducted and the data we analyzed, we have come to several conclusions:

1. Using a lower pair of hidden units and epochs takes more time to learn.

2. A higher pair of hidden units and epochs can achieve better learning.

3. Training a model beyond a certain point can lead to overfitting.

4. Using the same base classifier with different initial weights results in convergence after some time, indicating that the initial weights do not significantly impact model performance.

5. Ensemble techniques can improve the accuracy of both training and testing data.

6. Choosing an appropriate optimizer for the dataset can enhance the model's learning.

7. We were able to demonstrate that for a dataset with two equi-probable classes having mean [0,0] and [2,0] with variances of 1 and 4, respectively, the decision boundary is closer to [-2/3,0] with a Euclidean distance of 0.4671.

These observations highlight the ability of neural networks to adapt to different problem statements and learn non-linear boundaries, which is supported by the Universal Approximation Theorem.

# Citations

1. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function,overall%20loss%20and%20improving%20accuracy.
2. <https://en.wikipedia.org/wiki/Gradient_descent>
3. <https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm>
4. <https://en.wikipedia.org/wiki/Rprop>
5. https://in.mathworks.com/help/deeplearning/ug/cancer-detection.html

# Appendix

Code of Experiment 1: