# Enhancing Artificial Neural Networks for Thyroid Disease Diagnosis

William Shen

Research School of Computer Science, The Australian National University
william.shen@anu.edu.au

**Abstract.** Thyroid disease is a disease that restricts our ability to maintain body metabolism, and affects over 12% of Americans over their lifetimes [1]. Hence, it is important to develop general techniques for accurately and effectively predicting and preventing such diseases.
Artificial Neural Networks (ANNs) have been widely used in the past decades for tackling problems in pattern recognition and prediction. ANNs are powerful in that they can generalise and identify patterns in the data despite having no prior knowledge or assumptions about the data itself.
We investigate several methods and improvements for training a feed-forward neural network in predicting binding protein levels for thyroid disease - including pre-processing, evaluation techniques, network design using genetic algorithms, and network reduction techniques.
Our proposed neural network show a large improvement in prediction accuracy of over 3% versus the instance-based learning algorithms that Wilson and Martinez demonstrate in [6].

**Keywords:** artificial neural networks, network reduction, thyroid disease, activation function, cross validation, genetic algorithms

## 1 Introduction

Thyroid disease is a non-fatal, yet serious problem caused by the low (hypothyroidism) or high function (hyperthyroidism) of the thyroid gland. It is usually quite difficult to diagnose thyroid disease, as there is a multitude of symptoms which may also be seen in other diseases. In fact, over half of those with thyroid dysfunction remain undiagnosed [2]. Thus, it is extremely important to provide physicians with clinical decision support systems (CDSS) that assist them in diagnosing all kinds of diseases.

Machine learning techniques, such as Artificial Neural Networks (ANNs) are recognised as non-knowledge based CDSS as they make decisions by evaluating and eventually learning from examples. This is in comparison to knowledge based CDSS which explicitly encode the rules and associations between data, usually supplied by a domain expert. [3].

ANNs have several advantages over traditional machine learning algorithms, such as logistic regression or support vector machines, the most notable being that they can learn the functional form of the data themselves rather than the form being explicitly specified through a basis function, or a kernel, for example. One major disadvantage of ANNs, however, is that they can be slow and unreliable to train. ANNs are usually trained through the back-propagation method, where we feed the data through the network and adapt the weights by propagating the error gradients backward. Moreover, we cannot determine the best architecture for a neural network, as Gedeon and Harris states, a priori [4].

There are several techniques we can apply to improve the generalisation and performance of our ANNs to overcome these challenges. In section 2, we discuss the structure of the data set and the pre-processing techniques we can apply to assist the network to learn better. In section 3, we discuss the model design of our network, including activation and loss functions, and also discuss evaluation techniques for unbalanced data sets. In section 4, we discuss a genetic algorithm for determining a good choice of hyperparameters for training our network. In section 5 we consider network reduction methods aimed at improving the generalisation and performance of our neural network. Finally, in section 6, we discuss the results of the final implementation of the neural network and compare the results to those achieved the instance-based learning algorithms [6] that Wilson and Martinez demonstrate.

Although our main goal is to improve binding protein level prediction for thyroid disease diagnosis using ANNs, the methods we discuss are generally applicable to a data set in any domain. Hence, we have attempted not to tailor our design and evaluation choices very specifically to the case of thyroid disease.

## 2 Data Set and Pre-Processing Techniques

We use the binding protein data found in the Thyroid Disease Data Set on the UCI Machine Learning Repository. The data represents a classification problem where we must identify whether a patient has increased, decreased, or normal binding protein levels (3 classes) based on 22 discrete features and 7 continuous features.

We have chosen this data set for many reasons. Firstly, there is a mix of discrete and continuous variables – leading to interesting pre-processing techniques. Secondly, the data set has a large proportion (≈33%) of missing values. This means that we must either discard any data with missing values, which is undesirable, or investigate methods to impute the missing values.

Finally, there are limited examples for diagnoses with increased or decreased binding protein levels. This means that we must fully learn the patterns from the limited data we have for these classes. Moreover, we must

have a good distribution of classes when training the network to avoid it from generalising samples to one class. Since the data set is unbalanced, we also need to consider different evaluation methods instead of accuracy such as recall (sensitivity), F1, and Cohen's kappa.

### 2.1 Unbalanced Data Set

Table 1: Class Distribution of Training and Test Set

| | Train | Test |
|---|---|---|
| Increased | 117 | 23 |
| Decreased | 9 | 5 |
| Normal | 2637 | 928 |
| Total | 2800 | 956 |

As shown in Table 1, our data set is very imbalanced, as there is a limited number of examples for increased and decreased binding proteins in comparison to normal levels.

If we were to train our network on all of the training data, it would be very possible for the network to generalise to the 'Normal' class, especially if we were to apply an optimisation algorithm such as stochastic gradient descent with a single sample or mini-batches. This is not very desirable, as we want to learn the subtle patterns and classify the increased and decreased binding protein levels correctly. As we shall see later, we can improve the predictive accuracy of the network by only training on a fraction of the data from the 'Normal' class.

### 2.2 Missing Values

Approximately 33% of the data in the training set have at least one missing value. Since it is unfeasible to train a neural network on samples that have missing data, one method we could use to remedy this issue is to simply remove all samples with missing values. This is called listwise deletion. However, if we were to do so with the training set, we would see a significant decrease in the number of examples for each class by 18.8% (increased), 22.2% (decreased) and 31.3% (normal).

We ideally do not want to delete these samples, as they could provide useful information for further pre-processing or for training our network.

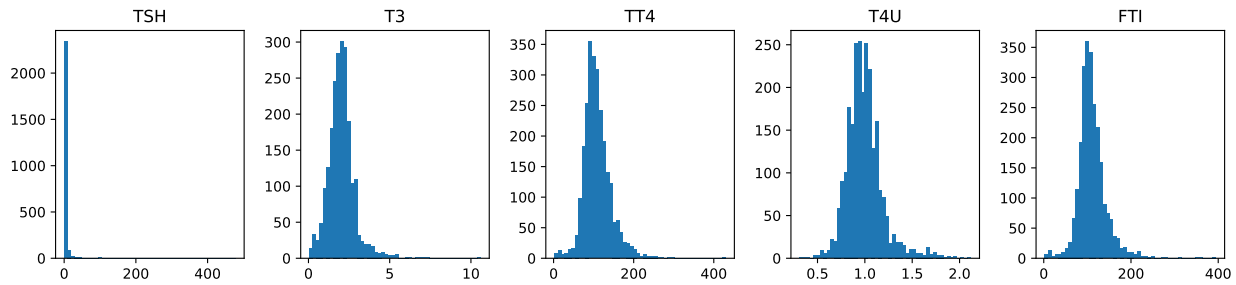
### 2.3 Imputation

Imputation is the process in which we replace (i.e. 'impute') missing data values with substituted values [7].

1. **Mean/Median/Mode substitution**: We replace any missing value with the mean/median/mode of the given variable for samples of the given class. These are examples of single value imputation. This can be problematic for variables where the distribution is not unimodal. Moreover, this increases the bias of our training data towards a specific value and may lead us to overfit. However, we can add random noise to soften this issue.
2. **Multiple Imputation**: In single imputation, imputed values are treated as equal to non-imputed data - potentially leading to a strong bias in the data. Multiple imputation solves this by introducing variability by averaging the single imputations across several imputed data sets. Evidently, multiple imputation is costly and complex.

We chose to use mean imputation, as the distributions of the relevant features, as shown in Figure 1, were generally unimodal, Gaussian, and not very skewed, though there were several outliers. If the distribution of the features were skewed, then it could be a better idea to use median imputation. Moreover, a simple experiment as described in Appendix 9.2, shows that mean imputation leads to the best network predictive performance.

Fig. 1: Distribution of Continuous Variables in Thyroid Data Set



### 2.4 Outlier Removal

We noticed that there were several outliers in the data set which could disturb the neural network during training. Some of these values were blatantly incorrect - e.g. the age of patient was 455. We removed samples of a variable $x$ which were more than $c$ standard deviations away from the mean.

$$x_i - \overline{x} \geq c \cdot \text{std}(x) \tag{1}$$

Where $x = \{x_1, \ ... \ , x_n\}$, $\overline{x}$ is the mean of $x$, $\text{std}(x)$ is the standard deviation of $x$, $c$ is a constant which indicates how far away from the std we remove samples.

We decided to only remove outliers for samples that were in the 'normal' class and set $c = 2.5$. By doing so, we encompass 98.7% of the values that lie around the mean of each variable and retain the very limited training data that we have for the other classes. The removed samples were then added to the test set.

### 2.5 Feature Scaling for Continuous Variables

The range of the continuous features in the data set vary significantly. Without normalising or scaling, we cannot guarantee the stable and quick convergence of our ANN. There are several methods we can use to solve this problem:

- **Max-Min Scaling**: $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$, where $x$ is the original value and $x'$ is the normalized value. This rescales all $x_i$ to be in $[0, 1]$.
- **Mean Normalisation**: $x' = \frac{x - \overline{x}}{\max(x) - \min(x)}$, where $\overline{x}$ is the mean of the variable $x$. This normalises the feature to have a mean of 0. Thus, we will have a mix of positive and negative values after normalising.
- **Standardization**: $x' = \frac{x - \overline{x}}{\text{std}(x)}$, this standardizes the feature to have zero-mean and unit-variance and can be very effective when the distribution of the feature is relatively Gaussian. However, outliers can cause the majority of the values to 'scale' to a very small range.

We chose use standardization as the distributions of our continuous features were relatively Gaussian as shown in Figure 1, especially after imputation and outlier removal. Moreover, through evaluating our model using K-Fold stratified cross-validation, as shown in Appendix 9.3, we found that standardization gave the best prediction performance.

### 2.6 Encoding for Discrete Variables

For the true and false variables (e.g. sick, pregnant, goitre), we encode false as 0 and 1 as true. For 'sex', we encode female as 0, male as 1 and unknown as 0.5, representing a neutral value between female and male. We encode the target classes increased binding protein, decreased binding protein, and normal as 0, 1 and 2 respectively.

## 3 Model Design

### 3.1 Neural Network Architecture

With all redundant features removed, there are 21 features for each data point. Thus, we model our network with 21 input neurons each taking a feature from our sample. Since we must classify each sample to one of three classes, our network will have 3 output neurons. We will take the neuron with the maximum output to be our predicted class. To determine a 'good' number of hidden layers and hidden neurons we will use a genetic algorithm, as we discuss in Section 4.

### 3.2 Activation Functions

We consider the logistic sigmoid, hyperbolic tan (tanh), rectified linear unit (ReLU) and the leaky rectified linear unit (Leaky ReLU) activation functions. We theorize that using the leaky ReLU will give us the best results, as it represents no 'information loss' and solves the problem of 'dying' ReLUs when the gradient of the ReLU is 0 for input less than 0. However, given that the size of our network is relatively small, we may not encounter dying ReLUs.

Our initial experiments, as seen in Appendix 9.1, showed that using the (leaky) ReLU activation function lead to better results than if we used the logistic sigmoid and helped our network converge faster.

### 3.3 Loss Function

We use the cross-entropy loss as our loss function as we are tackling a multi-class classification problem. Cross-entropy measures the 'distance' between what the neural network's belief of the output class is, and what the actual target class is. Cross-entropy loss is a much better choice than mean squared error for classification problems as the computed gradients stay relatively large in comparison to the gradient for squared error as the network converges.

### 3.4 Evaluation Function

**The Paradox of Accuracy** The accuracy paradox [10] states that "predictive models with a given level of accuracy may have greater predictive power than models with higher accuracy". This is especially true for unbalanced data sets. Consider the following confusion matrices for 1085 samples:

Table 2: Accuracy ≈ 95.8%

| | increased | decreased | normal |
|---|---|---|---|
| increased | 20 | 0 | 5 |
| decreased | 0 | 9 | 1 |
| normal | 40 | 0 | 1010 |

Table 3: Accuracy ≈ 97.05%

| | increased | decreased | normal |
|---|---|---|---|
| increased | 1 | 0 | 24 |
| decreased | 0 | 2 | 8 |
| normal | 0 | 0 | 1050 |

Clearly the predictor that produced the results in Table 2 is more desirable than the predictor that produced the results given by Table 3, as there are far less false negatives and hence far more true positives for the 'increased' and 'decreased' classes. Hence, we must come up with a more informative measure than accuracy to evaluate our neural network.

**Precision, Recall, F1** Precision defines the exactness of our classifier (ratio of true positive predictions to total number of positive predictions), while recall defines the completeness of our classifier (ratio of positive predictions to total number of positives). We can combine these together as the F1 score which conveys the balance between precision and recall. $F_1 = 2 \cdot \frac{precision \times recall}{precision + recall}$.

Although these measures provide a much better measure of model performance for unbalanced data sets than accuracy, we would be left with a F1 score for each class, making it more difficult to evaluate the overall performance of a network with a single value. Although we could take the mean of these scores for each class, this could give a skewed result as the precision, recall and F1 score for the 'normal' class would be much higher/lower than that of the other classes.

**Cohen's Kappa** [11], $\kappa$, measures the agreement between two 'judges' which classify samples into one of several classes. In our case, one judge is our trained neural network, and the other is us, the 'teacher'. where we provide the true target class. In more rigid terms, Cohen's kappa describes the number of agreements correct against the number of agreements expected by chance through a single value $\kappa$.

$$\kappa = \frac{f_0 - f_c}{N - f_c} \tag{2}$$

$f_0$ is the number of agreements between the 'judges', $f_c$ is the total expected frequency for the number of agreements by chance over each class, and $N$ is the total number of decisions. Clearly, the larger the $\kappa$ is, the better. We can trivially show $\kappa = 1$ for 100% accuracy, and $\kappa = 0$ if all samples have been classified to one class.

We have used Cohen's Kappa along with accuracy in our experiments to determine the performance of the neural network across different pre-processing and network architecture decisions. Although we could have introduced our own measure for determining the effectiveness of our classifier for the thyroid data set by penalising false negatives harshly, we use Cohen's kappa as it is more applicable to any unbalanced data set.

### 3.5 Training

To train our network, we use batch gradient descent. Although batch gradient descent is much more computationally expensive compared to stochastic gradient descent, it leads to a smoother traversal of the error manifold towards a minima, which could be beneficial. Moreover, it is less susceptible to the noise of the data feed to the optimiser at each step. Since our dataset is unbalanced, using an optimiser such as stochastic gradient descent could also lead the network to generalise to one class.

To evaluate the initial design choices of our network, we used stratified K-Fold cross validation, an extension to vanilla K-Fold cross validation. The folds in stratified K-Fold are generated in a way such that each is a good representative of the whole data set. This is vital when we have unbalanced data sets, as it ensures that each fold contains at least one sample of every class.

We train on the entire pre-processed training set, giving a test set ratio of 83%. The main reason for this high percentage is that we must limit the number of samples of the 'normal' class, as discussed in Section 2.1, to prevent the network from generalising to one class.

## 4 Genetic Algorithm for determining Network Hyperparameters

As Peck et al. discuss in [12], genetic algorithms are "well suited for searching in a large parameter space" as they "explore the parameter space and exploit the similarities between highly fit candidate solutions". Thus, genetic algorithms are applicable to several choices we must make when designing a neural network, such as input feature and hyperparameter selection.

We consider using a genetic algorithm to determine the number of hidden layers, number of hidden neurons, activation function, and learning rate for our neural network. These hyperparameters are usually very difficult to select, and often much experimentation is required to determine a good choice. As Gedeon and Harris state [4], we cannot determine the best architecture for a network 'a priori'.

### 4.1 Chromosome and Gene Encoding

Our chromosome contains four genes: the number of hidden layers, number of hidden neurons, activation function, and learning rate in order. Note that we represent the learning rate, which we limit to $(0, 1]$, as an integer for simplicity's sake. Our initial experiments representing the learning rate using the IEEE Standard for Floating-Point Arithmetic (IEEE 754) lead to undesirable learning rates when performing crossover and mutation.

We encode these values using Gray coding. Gray coding alleviates the Hamming Cliff issues encountered when using traditional binary coding, as it "ensures the Hamming distance between the representation of successive numerical values is one" [13]. By using Gray coding, we can assure that only a small change in the chromosome/gene is required for a small change in fitness.

### 4.2 Initialisation, Fitness and Training

Following our initial experiments training the neural network and for ease of binary transformations, we decided to limit the allele of each gene to the following domains:

- Number of hidden layers $\in \{1, 2, 3\}$
- Number of hidden neurons $\in \{5, 6, \ \ldots\ , 63\}$
- Activation function $\in \{\text{Sigmoid}, \tanh, \text{ReLU}, \text{Leaky ReLU}\}$
- Learning rate $\in \{0.01, 0.02, \ \ldots\ , 1.0\}$

We set our population size for our problem to be 30 to ensure we can encapsulate enough variety in the hyperparameter selection for each individual. To initialise the population, we randomly sample from the domain of each hyperparameter.

An appropriate measure for fitness for our problem of predicting thyroid disease is the Cohen's kappa on the test set. Accuracy would not be an appropriate measure as discussed in Section 3.4, due to the paradox of accuracy.

To train each individual network, we adopt early stopping (i.e. we stop training when the test loss is increasing for $n$ epochs, we set $n = 15$) and limit the maximum number of epochs to 1500. This is more ideal than setting a hard limit for the number of epochs as it reduces the chance for the network to overfit to the training data.

### 4.3 Crossover, Mutation, and Selection

We considered using one-point, two-point and uniform crossover to produce offspring. We decided to use uniform crossover as it is more suited to the transformations to the genes we require, and because it is usually more ideal for smaller populations as De Jong and Spears suggest in [14], because it can be 'less' disruptive to the genes. We produce 9 new individuals (children) in each generation.

Mutation is important to maintain the diversity of the characteristics within the population. It allows us to avoid local minima by preventing the population from being too similar. We randomly flip the bits within the chromosome of each individual with a probability of 0.01.

When selecting which individuals to produce offspring, we use proportional selection. This gives a higher probability of individuals with a higher fitness to reproduce leading to offspring with similar/increased fitness, whilst also giving those with lower fitness a chance to reproduce, keeping the population diverse.

We maintain our population size of 30 by using elitism and tournament selection. We ensure that the top 10 individuals in the population survive to the next generation (elitism), and use tournament selection with a tournament size of 5 to choose the remaining 20 individuals. This ensures there is generally a good mix of characteristics within our population, and that we avoid populations that have similar chromosomes.

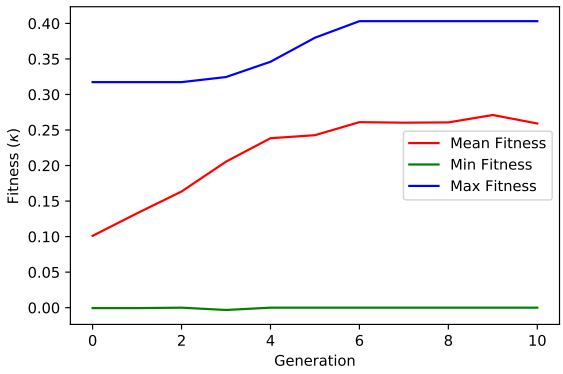
We terminated the genetic algorithm once we did not see any more improvements in the maximum fitness in the population. We found that 10 generations was sufficient in most situations.

### 4.4 Results

We observed $\kappa \approx 0.4$ and an accuracy of 98.4% in the most fit individual in our population following ten generations of our genetic algorithm. The selected hyperparameters were 2 hidden layers, 32 hidden neurons in each hidden layer, the ReLU activation function, and a learning rate of 0.329. Figure 2 shows the overall fitness of the population over this specific run of the genetic algorithm.

This represents a moderately substantial $\kappa$ and accuracy increase over our best network trained with our hand-picked hyperparameters (2 hidden layers, 12 hidden neurons, ReLU, lr=0.4) which achieved $\kappa \approx 0.35$ and an accuracy of 97.4%. In most situations, our genetic algorithm found a choice of hyperparameters that lead to similar or better performance than our network trained with hand-picked hyperparameters.

Fig. 2: Fitness of Population over Time




## 5 Network Reduction Techniques

It is often very difficult to determine the ideal number of neurons for the hidden layers in our neural networks. Even if our network performs well with a large number of neurons, it could be the case that many of them are performing the same function and hence can be pruned. Moreover, to train networks successfully and efficiently, we usually need to use more hidden neurons than the minimum number actually required.

Network reduction has many advantages. It reduces the amount of memory our network takes up and leads to less computational power required to make new predictions. More importantly, network reduction helps our network generalise better as by removing excess units, we can reduce any overfitting to the training data.

We discuss Gedeon and Harris' notion of distinctiveness [4], and investigate how we could apply pruning using distinctiveness for networks trained for unbounded activation functions (e.g. ReLU, leaky ReLU).

### 5.1 Distinctiveness

Distinctiveness measures how similar two hidden units are in the input space, by computing the angle between the functionality of the two units in input space using an angular range of 0-180°. That is, we must first normalise the weight vectors of the hidden neurons such that we use an angular range of 0-180°, then we can calculate the angle between each pair of weight vectors.

Gedeon and Harris propose that an angular separation of up to about 15° or over 165° between two hidden units is considered 'sufficiently similar' and hence one of the corresponding units can be pruned. The weight vector of the hidden unit which is being pruned is added to the weight vector of the unit that remains.

**Normalising** Recall that the logistic sigmoid squashes all activations between 0 and 1, and has a 'center' of 0.5. Hence, to use an angular range of 0-180° we can simply subtract 0.5 element-wise from all the weight vectors. Similarly, the hyperbolic tangent tanh squashes all activations between -1 and 1 and has a 'center' of 0 and hence we do not need to normalise. On the other hand, we cannot say the same for unbounded activation functions such as the ReLU – there is no 'center' point for these activations.

To solve this issue, we propose that sampling and using the mean of the activations of the hidden units during training is a sufficient metric for determining the 'center' of unbounded activation functions.

### 5.2 Results

For the logistic sigmoid and hyperbolic tan, pruning non-distinct hidden neurons usually improved the generality of the network as expected – $\kappa$ increased following pruning as shown in Tables 3 and 4.

Table 4: Before pruning, accuracy = 96.3%, $\kappa = 0.27$

| | increased | decreased | normal |
|---|---|---|---|
| increased | 20 | 0 | 5 |
| decreased | 0 | 0 | 5 |
| normal | 94 | 0 | 2715 |

Table 5: After pruning, accuracy = 96.5%, $\kappa = 0.282$

| | increased | decreased | normal |
|---|---|---|---|
| increased | 20 | 0 | 5 |
| decreased | 0 | 0 | 5 |
| normal | 84 | 0 | 2721 |

However, for the ReLU and leaky ReLU, our experiments showed mixed results as shown in Tables 5 and 6. In most situations, pruning hidden units decreased the predictive performance and generalisation of our network. Sometimes, after pruning, the network classified all samples to one class leading to $\kappa = 0$. However, in rare scenarios, pruning helped the network classify more precisely to each class by removing false positives and false negatives leading to a higher Cohen's kappa.

Table 6: Before pruning, accuracy = 97.1%, $\kappa = 0.30$

| | increased | decreased | normal |
|---|---|---|---|
| increased | 17 | 0 | 8 |
| decreased | 0 | 1 | 4 |
| normal | 68 | 2 | 2739 |

Table 7: After pruning, accuracy = 98.8%, $\kappa = 0.22$

| | increased | decreased | normal |
|---|---|---|---|
| increased | 4 | 0 | 21 |
| decreased | 0 | 1 | 4 |
| normal | 5 | 4 | 2800 |

We theorize that adjusting the conditions (tolerance) of $\leq 15°$ and $\geq 165°$ for the angular separations could solve this issue, as by using the original conditions we prune out approximately 50% of the hidden neurons when using a unbounded activation function. However, it is more likely that using the mean of the activations as an approximation of the 'center' of the activation function is not an ideal metric. Our experiments show that the activation means shift around constantly between different training runs.

## 6 Results and Discussion

To train our final network we applied mean imputation and standardization with a outlier constant of 2.5 as described in the pre-processing section. For our hand-picked hyperparameters, we used two hidden layers with 12 hidden neurons in each layer, the ReLU activation function, a learning rate of 0.4, and a L2 regularisation of 0.03. We decided this through experimentation with different hyperparameters as seen in the Appendix. Moreover, using the ReLU allowed our network to converge must faster than if we had used the logistic sigmoid, and also gave marginally better results as seen in Figures 3 and 4. This network achieved a best accuracy of 97.4% and a Cohen's kappa of 0.35 on the test set.

Pruning on this specific network lead it to classify all test samples to a single class giving $\kappa = 0$ due to the reasons as outlined in section 5.2. However, in most situations, our network achieved an accuracy of about 97%, kappa of about 0.28 and very insubstantial improvements, if any, through pruning using distinctiveness. However, this represents a significant improvement over the baseline accuracy of 95% and Cohen's kappa of 0.22.

Fig. 3: Typical graphs for training with ReLU

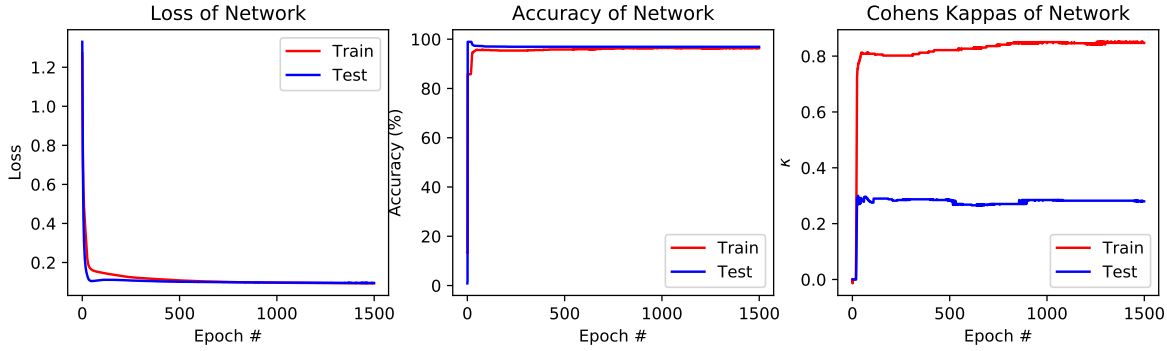
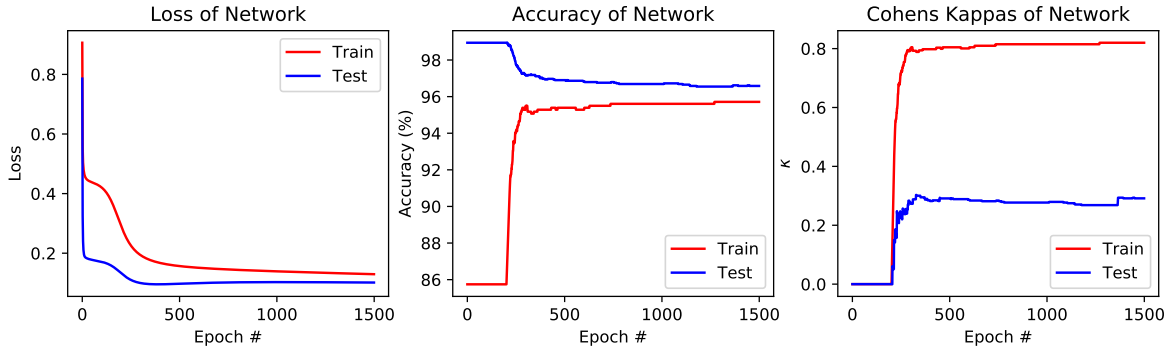



Fig. 4: Typical graphs for training with Sigmoid




As described in section 4, the best individual trained using our genetic algorithm achieved an accuracy of 98.4% and a Cohen's Kappa of 0.40. In most situations though, our genetic algorithm would find a network that had at least the same performance as that of the networks with hand-picked hyperparameters. No hidden units were pruned on the networks trained using the genetic algorithm. We believe this is because we applied early stopping rather than training with a fixed number of epochs, but leave this for further investigation.

Table 8: Neural Network Typical Performance on Test Set

| | Base NN | NN + Limited Pre-processing | NN + Pre-processing | NN + Pre-processing + Genetic |
|---|---|---|---|---|
| Accuracy | 98.94% | 95% | 97% | 98% |
| $\kappa$ | 0 | 0.22 | 0.28 | 0.35 |

Note: NN = neural network, we have not included pruning as it did not improve the results in almost all scenarios. Moreover, no neurons were pruned with the final network taken from the genetic algorithm.

### 6.1 Comparison

Wilson and Martinez propose heterogeneous distance functions that handle both discrete (categorical) and continuous variables for instance-based learning algorithms [6]. They used a K-nearest-neighbour classifier with $k = 1$ and compared the results achieved on several data sets, including binding protein for thyroid disease, using different distance functions such as the Euclidean distance and the Heterogeneous Value Difference Metric (HVDM). All 'pre-processing' to the data set was achieved through the distance functions themselves.

Table 9: Average Accuracy using 10-Fold Cross-Validation [6]

| | Euclidean | HOEM | HVDM | DVDM | IVDM | WVDM |
|---|---|---|---|---|---|---|
| Average Accuracy | 94.89 | 94.89 | 95.00 | 94.86 | 95.32 | 95.29 |

The highest accuracy of 95.32% was achieved using the Interpolated Value Difference Metric (IVDM) distance function where continuous values are discretised into several bins. We note that these accuracies are very similar to our baseline accuracy of 95% – this may suggest that neural networks are better suited to the task of predicting binding protein levels for thyroid disease.

Our final neural network trained using a genetic algorithm achieved a classification accuracy of 98.4% on the raw test set, representing a 3.08% and 3.51% improvement over nearest neighbours with IVDM and the Euclidean distance respectively.

We believe that our neural network is a better choice than instance based learning algorithms with improved distance functions for many reasons. First of all, there is the improvement in classification accuracy for our neural network which we believe is due to the richer feature space of the data set (21 features). Instance-based learning algorithms often suffer when the dimensionality of the data is high (curse of dimensionality), mainly due the difficulty of effectively determining distance in multiple dimensions.

Moreover, instance-based learning algorithms make predictions by comparing a new sample with all training instances. Although these algorithms do not require any training, it is evident that making predictions is very computationally expensive. On the other hand, although neural networks take a significant time to train, they are very quick at making predictions. Thus, it is much more achievable to deploy a trained neural network, especially on devices with limited computational power or applications that require predictions on a very frequent basis.

## 7 Future Work

More pre-processing techniques, evaluation methods, and network reduction experiments should be investigated and carried out in the future. We only considered single imputation methods, which could lead to a strong bias in the dataset. Future work should experiment with multiple imputation methods leading to more noise in the imputed data.

In this paper, we briefly discussed precision and recall as potential metrics for evaluating our model, but decided against them as they did not give us a single value to work with. Future work should consider how we could combine precision, recall and the F1 score into a single metric, perhaps by considering increased and decreased binding proteins as a single class or weighing each class more than the normal class.

We believe that better results could be discovered if feature selection and input pre-processing were to be considered as parameters in our genetic algorithm. However, this would greatly increase the search space and increase computational requirements whilst only seeing very moderate improvements in performance.

Finally, we were relatively unsuccessful in finding a good metric for applying distinctiveness in pruning hidden units. We will need to consider and evaluate if it is possible or feasible to come up with a better metric or investigate how using bounded ReLUs [8] could help us reduce the size and hence improve the generalisation of our network.

## 8 Conclusion

In this paper, we have shown the process for designing, training, improving, and evaluating a neural network. Firstly, we motivated the thyroid disease dataset and discussed methods to pre-process the variables, including imputation, outlier removal and normalisation techniques. We also decided the pre-processing techniques we would use through an analysis of the distribution of the variables and simple experiments using K-Fold Stratified Cross Validation.

Next, we considered the design of our neural network including the architecture, loss function and evaluation mechanisms. We also considered and discussed different activation functions, demonstrating each's advantages and disadvantages. More importantly, we presented the paradox of accuracy and introduced Cohen's kappa as a more reliable evaluation technique than accuracy which allowed us to evaluate the design decisions we made more effectively.

Then, we demonstrated a genetic algorithm for determining a good choice of hyperparameters (number of hidden layers and neurons, activation function, learning rate) for our neural network. We briefly discussed the structure of the chromosome, crossover, mutation and selection operations used, along with the results we achieved.

In Section 5, we examined network reduction techniques – specifically, the notion of distinctiveness. We showed that pruning non-distinct hidden units for bounded activation functions (sigmoid, tanh) improved the generalisation of the network and increased the accuracy and Cohen's kappa. We proposed to take the mean of the activations as the 'center' for normalisation for the unbounded activation functions (ReLU, leaky ReLU) and demonstrated that it was not a very good metric as it decreased the accuracy and Cohen's kappa in most situations.

Finally, we presented the results of our final neural network and compared them against the improved distance functions Wilson and Martinez demonstrated for instance-based learning algorithms. Our initial results are very promising and demonstrate the importance of effective pre-processing and evaluation techniques along with the power of genetic algorithms.

## 9 Appendix

For all of the following experiments, we evaluated the model using 5-Fold Stratified Cross Validation and reported the mean loss, accuracy and Cohen's Accuracy.

### 9.1 Activation Functions

To compare activation functions, we used mean imputation, an outlier constant of 2.5, and standardization.

| Activation Function | Mean Loss | Mean Accuracy | Mean Cohen's Kappa |
|---|---|---|---|
| Sigmoid | 0.081 | 97.85% | 0.75 |
| ReLU | 0.066 | 98.22% | 0.80 |
| Leaky ReLU | 0.069 | 98.02% | 0.79 |

In general, the ReLU and leaky ReLU performed better than the logistic sigmoid.

### 9.2 Imputation

To compare imputation methods, we used max-min scaling for the data (which scales continuous variables between [0, 1]), an outlier constant of 2.5, and the ReLU activation function.

| Imputation | Mean Loss | Mean Accuracy | Mean Cohen's Kappa |
|---|---|---|---|
| Mean | 0.09 | 97.64% | 0.73 |
| Median | 0.10 | 97.43% | 0.71 |
| Mode | 0.09 | 97.50% | 0.69 |

Mean imputation gave the highest accuracy and Cohen's kappa, which is not unexpected as the distribution of the continuous variables is relatively Gaussian as shown in Figure 1.

### 9.3 Scaling/Normalisation

To compare scaling/normalisation methods, we used mean imputation for the data, an outlier constant of 2.5, and the ReLU activation function.

| Normalisation | Mean Loss | Mean Accuracy | Mean Cohen's Kappa |
|---|---|---|---|
| Max-Min | 0.09 | 97.64% | 0.72 |
| Mean Normalisation | 0.09 | 97.68% | 0.73 |
| Standardization | 0.07 | 97.95% | 0.77 |

Using standardization improved the predictive performance of our network quite significantly over the other methods of scaling/normalisation.

## References


1. The American Thyroid Association.: About Hypothyroidism: https://www.thyroid.org/media-main/about-hypothyroidism/
2. Cooper, DS. and Ridgway, EC.: Thoughts on prevention of thyroid disease in the United States, Thyroid, vol. 12, pp. 925–929. (2002)
3. Alther, M. and Reddy, CK.: Clinical Decision Support Systems, Healthcare Data Analytics (2015)
4. Gedeon, TD. and Harris, D.: Network Reduction Techniques, Proceedings International Conference on Neural Networks Methodologies and Applications, AMSE, vol. 1, pp. 119–126, San Diego, (1991)
5. Gedeon, TD.: Indicators of Hidden Neuron Functionality: the Weight Matrix versus Neuron Behaviour, Proceedings 1995 Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, pp. 26–29, (1995)

6. Wilson, DR. and Martinez, TR.: Improved Heterogeneous Distance Functions, Journal of Artificial Intelligence Research, vol. 6, pp. 1–34, (1997)
7. Gelman, A. and Hill, J.: Data Analysis Using Regression and Multilevel/Hierarchical Models, Chapter 25, pp. 529–543, (2007)
8. Liew, SS., Khalil-Hani M. and Bakhteri R.: Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems, Neurocomputing, vol. 216, pp. 718–734 (2016)
9. PyTorch Documentation: http://pytorch.org/docs/master/nn.html
10. Wikipedia contributors: Accuracy paradox, Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Accuracy_paradox&oldid=814373733
11. Cohen, J.: A Coefficient of Agreement for Nominal Scales, Educational and Psychological Measurement, vol. 20, pp. 37–46 (1960)
12. Peck, CC., Dhawan ,AP., Meyer, CM.: Genetic Algorithm based Input Selection for a Neural Network Function Approximator with Applications to SSME Health Monitoring, IEEE International Conference on Neural Networks, vol. 2 , pp. 1115-1122 (1993)
13. Engelbrecht, AP.: Computational Intelligence: An Introduction (2007)
14. De Jong, KA., Spears, WM.: An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms, Parallel problem solving from nature, pp. 38-47 (1991)