Genetic Algorithm Hyperparameter Optimization on a Reduced Linear Classification Neural Network for SARS-COV-1 Data

Jacqueline Ann Wright

Research School of Computer Science, Australian National University
U5959046@anu.edu.au

Abstract. In this paper, we construct a baseline multi-layer classification neural network which predicts the labels of Normal, SARS, Pneumonia and High Blood Pressure from patient data and then prune this model using Gedeon's *distinctiveness* method of pruning [6]. The baseline model's learning rate and number of hidden neurons hyperparameters are tuned by a genetic algorithm and the pruned model's learning rate and angle boundaries hyperparameters are tuned by a separate genetic algorithm. The aim of this paper is to evaluate the efficacy of genetic algorithms in selecting hyperparameters to create an optimal reduced network. We find that genetic algorithm hyperparameter optimization is highly successful in this task and is successful even when many features are removed from the dataset by picking a higher than hypothesized learning rate for the dataset, almost always predicting with 100% accuracy with a cross-entropy loss curve which indicates that our model is learning well. It is determined that a more complex dataset and different pruning techniques could produce more convincing results, as even with few features, the genetic algorithms tend to choose individuals with hyperparameters that are on the edge of our user-defined boundaries and creates difficulties in defining an effective individual evaluation formula.

Keywords: SGD, Back-Propagation, Distinctiveness Pruning, Classification, Genetic Algorithm, Hyperparameter Optimization, DEAP

1 Introduction

1.1 Background

Machine learning (ML) algorithms have been highly successful in solving problems within numerous areas of application and interest in ML continues to grow alongside the increase in computing power. Classification, regression, clustering, and dimensionality reduction tasks are some of the many tasks in which ML algorithms excel [1]. This success has resulted in the need for experts to continuously construct many models which are tailored for exponentially increasing sets of data to keep up with demand in the growing field of data science [2]. A data scientist must make many choices when building a ML model such as the type of algorithm, the hyper-parameters and how the model's performance is evaluated [2]. While it is suitable to choose the model's learning technique and evaluation technique by hand, an optimal classification model can have a potentially infinite search space when determining optimal hyperparameter combinations. There can be significant differences between even small changes in values in the parameter combinations of the learning rate, number of hidden neurons and the features of the chosen technique. Due to this, hyperparameter optimization can be a highly time-consuming task and it is worthwhile to consider heuristic methods which allows experts to construct optimal models more efficiently and with less human effort, such as types of algorithms known as *Evolutionary Algorithms*. Additionally, automated hyperparameter optimization may improve the performance of ML algorithms as it allows for hyperparameter solutions which are tailored to the specific problem at hand [3]. For these reasons, the capabilities of algorithms which automates the hyperparameter selection process should be explored.

Pruning techniques which reduce the size of the network by removing neurons may be applied on top of automated hyperparameter optimization and fine-tuned by a separate genetic algorithm, which reduces the cost of building an effective model further. As back-propagation can be slow to train networks, it is beneficial to remove excess neurons which do not provide meaningful functionality to the model [6]. The *distinctiveness* technique, proposed by Gedeon (1991) is one such method which removes neurons based on their angular separation; vectors of weights which have angles of lower than 15

degrees or higher than 165 degrees are removed [6]. This paper focuses on the application of both techniques; a suitable type of evolutionary algorithm and the *distinctiveness* pruning technique.

1.2 Task Description

In this paper, we focus on the metaheuristic *genetic algorithm* (GA) optimization technique on a simple multi-label classification problem. A genetic algorithm is a type of evolutionary algorithm which operates by creating and evolving a population of individuals, and individuals are evaluated until the optimum is reached. The best individual is saved with the *hall of fame* function, which holds the individual with the best fitness value throughout the whole evolutionary process [4]. We utilize a pruning method based on *distinctiveness* to ultimately construct an optimal reduced network which is computationally efficient [5]. For such a problem, manually testing hyperparameters is often tedious despite its simplicity, and may be subject to human biases regarding the data; this raises a reproducibility issue, as different experts may select different hyperparameters under time constraints. A GA not only speeds this process up but allows for reproducibility as the same results are yielded from models which are tuned in the same fashion [3]. We aim to fine-tune both a base model and a *distinctiveness* pruned model which performs multi-label classification of SARS-COV-1 symptoms with stochastic gradient descent and use GAs to select hyperparameters for the model. This results in an optimal reduced network which is time-efficient and computationally efficient. We evaluate the performance of the GA for these tasks with its role in aiding the pruning process and discuss the results and limitations for its application with the goal of constructing an optimal reduced network which can predict as well as a base model with many more hidden neurons. This includes a comparison of the results from our last paper, where hyperparameters were manually selected and the same base model is used (2021) [5]. It may be necessary to attempt testing with fewer features, as our dataset has simple patterns and it may be difficult to evaluate the performance of the GA with the full dataset.

1.3 Dataset Description

The dataset used is Gedeon's (2005) SARS-COV-1 dataset [8]. There are 4000 rows of data within the dataset, and this is split into 1000 for each of four labels, SARS, HighBP, Normal and Pneumonia. Doctors must check for certain symptoms when determining if a patient has SARS; *fever*, *blood pressure*, presence of *nausea*, and *abdominal pains* [8]. The *fever* and *blood pressure* symptoms are numerical values of temperature and blood pressure readings, and are divided into sets of low, medium, and high. These are taken from the patient at four-hour intervals starting from 8am and ending at 8pm. The *nausea* and *abdominal pains* features each have two columns for *yes* and *no*. There are 23 features altogether in the dataset and some features are removed during testing to observe its implications on our results.

2 Method

2.1 Data Inspection and Pre-processing

The data was given in .csv format in four separate files corresponding to their label with 1000 rows of data in each file. To prepare the data to be input into the model, these files were consolidated into one .csv file and a 24^{th} column was added which is the corresponding output label. Headers were added for clarity in handling the values. The dataset given by Gedeon (2005) is already normalized to values between 0 and 1, so no normalization of the values is necessary. Each of the 23 features represents one input, and the 4 labels SARS, HighBP, Normal and Pneumonia are the output classes of the network. This .csv file data is read and converted into a DataFrame. The final column of output class labels is converted from a String to an Integer value between 0 to 3 to represent each class, and the rest of the data is also converted into Integer values from string numeric values. The DataFrame is now comprised of Float and Integer values, including the output classes. With the data sorted in this manner, it is easy to alter the model if needed and add or remove features to test the performance of the GA in classification. We alter the data from its initial state of 23 features (input vectors) by reducing the number of features in response to our results from the full dataset to better evaluate the performance of GA hyperparameter optimization and to further understand the task of GA-aided classification for our dataset. These reduced sets of features are saved as separate files and the code is set to read from any of these files without the need to be altered.

We extend the work done in our last paper [5] and include a validation set when performing the train-test-split for the base model. This is because the hyperparameters will be extensively tuned with the GA, and so a validation set is necessary and validation loss is used as a variable in our individual evaluation formula for the baseline model. To improve upon reliability of the results, we also use a *stratified* split which randomly and equally distributes the sets between the classes where values are chosen and ordered at random. The data is split into half so that it can be partitioned into a base set and a pruned set. The base set is further split into a training, validation, and testing set, where the training set has 1280 rows, the validation set has 320 rows, and the testing set has 400 rows of data. For the pruned model, there is only a train-test split as we load the base model to continue training, prune the hidden neurons from the base model and only adjust the learning rate and angular separation boundaries. The training data for the pruned set has 1600 rows and the testing data has 400 rows of data.

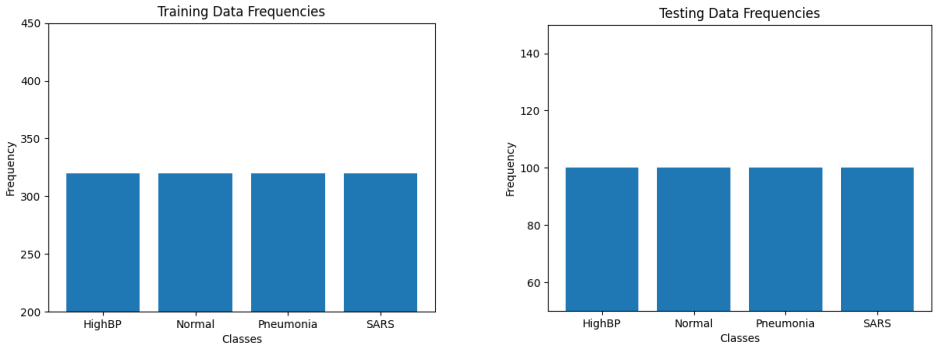


Fig. 1. Example frequency distribution of target labels for training and testing data generation by the Scikit-Learn train_test_split() stratified method.

After the data is sorted into their respective sets, the DataFrame is converted into arrays which allows for further conversion into torch tensors. These tensors hold the inputs and outputs for the models. Reproducibility can be guaranteed in the data splits by altering the random state value.

2.2 Baseline Model Design and Hyperparameters

The baseline model used is of the same structure used in our previous paper [5] to allow for comparison of results. The baseline model is a multi-layer sigmoid neural network consisting of three layers; the input layer with neuron size corresponding to the number of features, the hidden layer with GA-adjusted hidden neurons, and the output layer of 4 neurons. PyTorch is used to transform the layer outputs for the linear model and its sigmoid activation function is applied in the hidden layer during the forward pass. We use cross-entropy as the loss function.

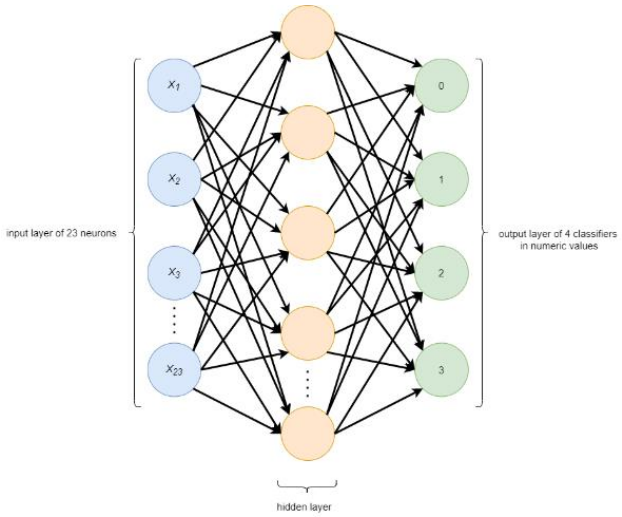


Fig. 2. Diagram of the multi-layer neural network with the sigmoid function for the hidden layers. 23 input neurons, GA-specified number of hidden neurons in the hidden layer and 4 output neurons.

Performance is observed and measured using a confusion matrix and classification report. Predictions are converted into one column for comparison and normalized with the SoftMax function.

Cross-entropy loss plots are extensively used in determining the correct number of epochs for training. The number of epochs parameter is set at 150 epochs for the base model, as the cross-entropy loss plot shows that convergence occurs with well-performing GA selected hyperparameters from between 60 to 150 epochs. We do not train it further for concerns of overfitting due to the simplicity of the dataset. We choose to prune the model only one time after the base model is complete, as the data is simple, and we expect the resulting pruned model to have very few neurons.

To facilitate usage of a genetic algorithm for hyperparameter optimization, the base model is created within a function. The hyperparameters that will be tested for the base model are the hidden neurons and the learning rate; this function takes these two values from the GA to run a network with the GA-presented hyperparameters. Each time the function is run, the state

dictionary of the model is saved and used when the model is retrained with the pruned network. The function returns the final training accuracy, validation accuracy and testing accuracy as well as the final validation loss and a penalty for poor performance regarding the validation loss. This penalty adds 0.20 to the count for each time the validation loss in the current epoch is higher than the previous epoch and is referred to in the GA to ensure that the GA prioritizes individuals which result in a smooth validation loss curve.

2.3 Pruned Model Design, Distinctiveness and Hyperparameters

Many modern pruning methods use a trained model as an input, prune connections, then re-train and re-test the network [9]. This is the method that we use. The pruned model is constructed in the same way as the base model with 3 layers, input vectors of the number of features and 4 output vectors; however, the retrained model will have different parameters. The pruned model will have a new GA-selected learning rate and use the number of hidden neurons according to the pruning method. The base model is pruned by the *distinctiveness* method, which removes neurons in the weight matrix based on their angular separation [6]. To achieve this, every pair of vectors must have the angles between them calculated. In Gedeon's work, angles are deemed sufficiently similar if they are less than 15 degrees. If they have an angular separation of over 165 degrees, they are complementary, and both can be removed [6]. The angle between two vectors is as follows:

$$\theta = \cos^{-1}\left[\frac{\vec{a}\cdot\vec{b}}{|\vec{a}|\ |\vec{b}|}\right]$$

The output vector which processes the distinctiveness pruning are normalized to values between 0 and 1 by the sigmoid activation function. Instead of using a set minimum angle and maximum angle parameter as in Gedeon's paper [6], we are tuning it as a hyperparameter with the GA. This is to ensure a minimal network while maintaining maximum accuracy. As the goal of *distinctiveness* pruning is to create a reduced network which removes undesirable units that are deemed unnecessary, it would be optimal to remove as many of these neurons as possible without sacrificing model performance. Our model is simple, and it is possible that the final vectors have an angle greater than the maximum-angle, so we can only perform pruning if we have more than 2 vectors left in the model during the pruning stage. After the pruning, the model is re-trained on a new set of unseen data.

2.4 Genetic Algorithm Design for Baseline Model

The GA is constructed using the DEAP evolutionary computation framework. The population and individual chromosomes are generated using DEAP's toolbox. We use a separate GA for both phases of training, as we are tuning different hyperparameters. The base model GA has two chromosomes containing the two hyperparameters that we wish to tune. The hidden neurons parameter is a randomly generated Integer and the learning rate is a randomly generated Float.

Table 1. Example of a chromosome in the baseline model's GA.

Hidden Neurons	Learning Rate
13	0.1746473627483726

We define custom mutate and evaluate functions, as it is necessary to pre-define boundaries for our hyperparameters with disparate values. The minimum boundary for the hidden neurons is set to 10 and the maximum boundary is set to 50. These values are chosen as our previous paper uses 50 hidden neurons for the base model pre-pruning, and this minimum boundary is chosen as we would like enough neurons remaining to observe our techniques on the pruning method in the case that the GA determines that it may effectively select very few hidden neurons [5]. For the learning rate, we set the lowest boundary to 0.001 and the maximum boundary to 1.0. This creates a large search space of potential effective learning rates ranging from very slow to very fast. A learning rate of 1.0 would normally result in overfitting and overshoot the goal, however, we would like to explore potential unexpected solutions [10].

The GA is set to maximize the fitness function value of 1.0, so the evaluate function will prioritize individuals which are the closest to reaching this value of 1.0. Our evaluation function is as follows:

$$Evaluation = CombinedAccuracy - \left(\frac{HiddenNeurons}{1000}\right) - \left(\frac{FinalValLoss}{10}\right) - ValLossPenalty$$

This equation is the combined accuracy of the training, validation and test set minus the hidden neurons and validation loss scaled to the fitness function value and minus the custom penalty function which reduces an individual's fitness if it has a

rough loss curve. This evaluation formula aims to reward the individuals which result in the best overall accuracy, with the least hidden neurons, lowest final validation loss and has a smooth curve. A hidden neuron number of 30, for example, would penalize the fitness by 0.03, which is 3 percent of the overall fitness. We hoped to choose a penalty that is not too harsh but would still encourage the model to select fewer hidden neurons if it is able to maintain accuracy.

We use a population size of 50 and set the number of generations to 50. Larger sizes result in exponentially longer running time, as the model must run for every individual and generation. It has been observed that these parameters lead to good results for our search space, so we do not need more generations or population. Crossover rate is set to 0.8 and the mutation rate is high at 0.2, as since we have a relatively small population, we would like a higher mutation rate to explore more of the search space within the population and number of generations. The *hall of fame* function is used to store the best performing individual chromosomes across all the individuals and generations. The GA runs our base model repeatedly with its chosen parameters. When the GA is completed, the HOF individual's hyperparameter values are saved to a text file.

2.5 Genetic Algorithm Design for Pruned Model

Post-pruning, we use a different genetic algorithm to optimize hyperparameters which retrains the pruned model. It is constructed in the same fashion using DEAP, however, there are different parameters for the GA and a different evaluation method. As we will be using the number of hidden neurons after distinctiveness pruning is performed, we do not need to use number of hidden neurons as a hyperparameter. The hyperparameters are learning rate and the degree of angular separation for pruning.

Table 2. Example of a chromosome in the pruned model's GA.

Learning Rate	Minimum Angle Boundary	Maximum Angle Boundary
0.283	28	153

The learning rate is a random float with the same boundary 0.001 to 1.0. The minimum angle boundaries have a range from [1, 89]. The maximum angle boundaries have a range from [90, 180] and these are both randomly generated Integer values. These ranges are chosen to cover the maximum percentage of neurons which could be deemed similar and complementary, and to potentially prune less if the pruning is significantly impacting the model's performance. As we are tuning different hyperparameters in the pruned model, a different evaluation formula must be constructed. The formula we use is as follows:

$$Evaluation = CombinedAccuracy - \left(\frac{AngleMax - AngleMin}{1500}\right)$$

This takes the final combined accuracy for the training and testing phases and introduces a penalty for having a larger difference between the maximum angle boundary and minimum angle boundary, which would mean that the individual is deemed most fit if it can maintain the highest accuracy while removing the most neurons. The difference between angle boundaries is divided by 100 to scale it to within the FitnessMax range, and further divided by 15 as we do not want the penalty to be too harsh and have adverse effects on model accuracy for the best individual. The same population size and number of generations as the base model is used, as we are using the same amount of data as for the base model. The crossover and mutation rate are also the same at 0.8 and 0.2 respectively, to explore a higher number of potential solutions within our population and generations. The best performing individual is again saved by the *hall of fame* function into a text file. After the best performing hyperparameters have been saved, they may be used to re-run the model numerous times manually to evaluate performance.

3 Results and Discussion

3.1 GA Optimized Hyperparameters on Base Model

The GA for the base model hyperparameter optimization is run 5 times to examine reproducibility in the tuning of the chosen hyperparameters over different data splits, as we have noted that reproducibility in the tuning stage is one of the advantages GA optimizations provides. The learning rate is truncated at three decimal places for clarity. The model is first tested with the full dataset of 23 features.

Table 3. Suggested GA selected parameter values for the base model with 23 input vectors.

Test	Hidden Neurons	Learning Rate
1	10	0.987

2	10	0.978
3	10	0.972
4	10	0.983
5	10	0.994

It can be observed in these tests that the GA is consistently picking the same number of hidden neurons and the same learning rate to one decimal point for our full dataset. However, the issue is that it is picking the lowest boundary we have selected for hidden neurons and is pushing the highest boundary for the learning rate.

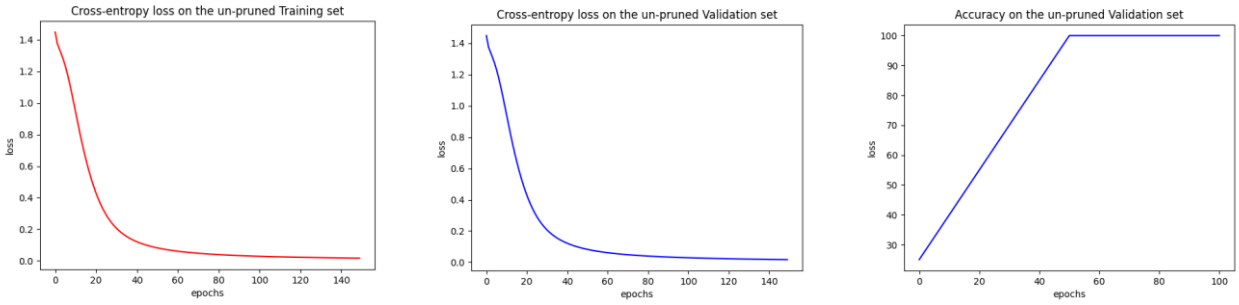


Fig. 3. Cross-entropy loss plots for the base model's training and validation set and accuracy plot of the un-pruned validation set using the GA-selected hyperparameters of 10 hidden neurons and a learning rate of 0.98 with 23 input vectors.

Despite evidence that our GA would like to push the boundaries of our hidden neurons and learning rate parameters, we observe that the model is performing very well, predicting at 100% accuracy, and converging at only approximately 60 epochs into training. This is in stark contrast to the findings in our last paper where convergence did not occur until nearly 2000 epochs [5]. The far higher learning rate was not previously considered as an option due to the consideration of overfitting, however, the GA has determined that this is not an issue for this dataset, and we can achieve optimal performance with these hyperparameters.

Although our base model is performing exceptionally well with the GA-selected hyperparameters, our dataset is too simple to fully explore the capabilities of GA optimization, and upon distinctiveness pruning and re-tuning we observe the same issue.

Table 4. Suggested GA selected parameter values for the pruned model. Only three tests were done to illustrate the issue. Learning rate is truncated to 3 decimal points.

Test	Learning Rate	Minimum Angle	Maximum Angle
1	0.483	88	91
2	0.744	89	90
3	0.347	89	90

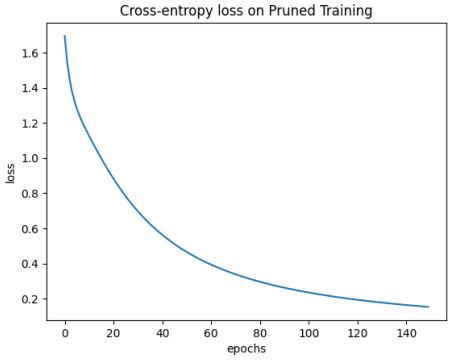


Fig. 4. Cross-entropy loss plot for one test of pruned training with the hyperparameters of 0.5 learning rate, 89 minimum angle and 90 maximum angle.

The model begins to converge around the 150th epoch, and in the test shown in Fig. 4., the model predicted with 100% accuracy with only 2 hidden neurons left. The hidden neurons were reduced by 10 and the model continues to perform optimally due to the simplicity of the dataset. We are not able to deduce the efficacy of the GA on our hyperparameters, as the model is easily able to perform optimally with only 2 neurons and the GA will select angular separations which result in the most pruning possible regardless of their similarity. However, the learning rate appears to have a wider range and is no longer at the boundary. In the three test runs above, the learning rates range from approximately 0.35 to 0.74 and can perform optimally in all these cases.

As our goals are to evaluate the performance of GA hyperparameter optimization as well as creating an optimal reduced network, we decide to remove some features and apply the above methods with reduced features where it is more difficult to correctly classify the labels.

Table 5. Testing the number of features and its effect on the performance of the GA. LR truncated at 3 decimal points. Validation and test accuracy are the results from running the model one time with the chosen parameters. 150 epochs.

Features	Hidden Neurons	Learning Rate	Validation Accuracy	Test Accuracy
Only Blood Pressure (6 inputs)	15	0.998	98.12%	99.50%
Only Temperature (12 inputs)	19	0.885	95.93%	94.50%
Only Abdominal Pains and Nausea (5	10	0.983	50%	50%
Only BP Systolic (3 inputs)	25	0.984	98.12%	98.25%
Only BP Diastolic (3 inputs)	28	0.988	97.81%	98.75%
One Temp Medium, BP Sys Medium, Nausea Medium (3 inputs)	11	0.971	99.68%	100%
One Temp Medium (1 input)	24	0.968	71.25%	73%

In these results, the learning rate is not affected apart from when the features are only *temperature*, where the learning rate is 0.1 lower. However, the number of hidden neurons ranges from 10 to 28 for the above tests. All features appear to be able to allow to model to accurately predict the label apart from *abdominal pains* and *nausea*. This is because there is no difference in the data between 'Normal' and 'HighBP' patients and neither of these experience nausea or abdominal pains, so the model is only able to differentiate between Pneumonia and SARS patients from Normal and HighBP patients. The other noticeable issue the model has in classifying the correct label for using only one input of medium temperature is seen from the confusion matrix – it is often misclassifying the temperature between HighBP and Normal patients and the temperature between SARS and Pneumonia patients, as the values within these two sets are similar. More combinations were tested other than those listed in the table above, and the fewest features that may consistently result in 100% accuracy are to use a combination - three features of 'temperature', 'BP' and 'nausea or abdominal pains'.

Understanding the contribution of the features for predictions allows us to select the most important features to construct an optimal model with less of a data requirement. We run the model with only one column of *medium temperatur*e, *systolic blood pressure* and *presence of nausea* and find that perfect testing accuracy can be achieved with only these three features and 11 hidden neurons. While *systolic BP* and *diastolic BP* could achieve similar accuracy with only 3 input vectors, the GA determined that many more hidden neurons are required to achieve this accuracy. When the number of features is reduced to 1 input of only temperature, validation and testing accuracy fall greatly.

The GA-selected hyperparameters perform far better in terms of prediction accuracy and converge at a much lower epoch on average than the manually selected parameters in our last paper [5]. Observing these results, it appears to be a combination of the columns which aids in correct classification, and every tested feature may be used to correctly classify data besides the use of only abdominal pains and nausea.

3.2 GA Optimized Hyperparameters on Pruned Model

To prune and retrain an optimal reduced network which can correctly classify with less features, we will use the dataset with 3 features, *temperature medium* for one time of the day, *BP systolic medium* and *nausea medium*. The number of epochs is increased to 500 for this stage, as the model with this number of features with fewer neurons requires more epochs to converge and begins to converge around 500 epochs.

Table 6. GA selected hyperparameters for the pruned model using only the above mentioned 3 features with 11 hidden neurons and 0.971 learning rate for the base model with 500 epochs. Final HN is the final hidden neuron number.

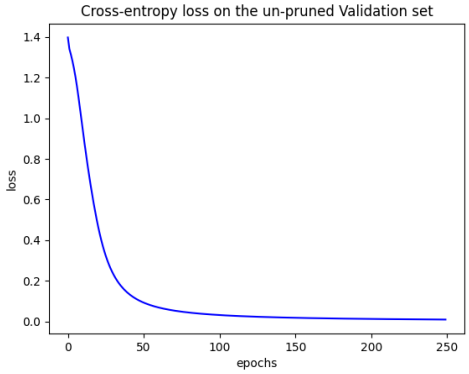
Test	Learning Rate	Minimum Angle	Maximum Angle	Final HN	Training Accuracy	Testing Accuracy
1	0.499	88	92	2	100%	100%
2	0.640	88	91	2	100%	99.75%
3	0.836	89	91	2	100%	100%
4	0.910	87	91	2	100%	100%
5	0.944	89	91	2	100%	100%

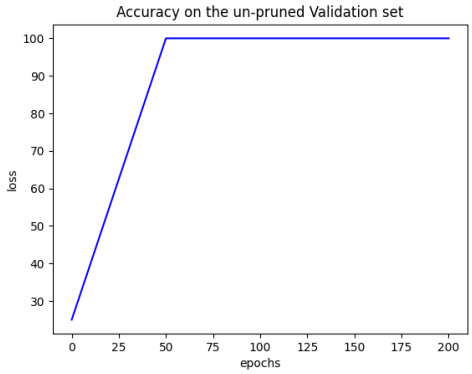
These results show that the GA selects a larger range of potential learning rates on the pruned model. This is because the model can achieve 100% with this range of learning rates, again, due to the simplicity of prediction despite there being only 3 features and it has already learned on half of the data, so many types of individuals are deemed to have the best fitness over the different tests. The GA has discovered that it can achieve optimal performance by pruning as many neurons as possible, so it has selected the largest possible minimum angle and smallest possible maximum angle boundaries to simply prune as much as it can to reach the smallest number of hidden neurons due to our penalty formula. While it is difficult to analyze the efficacy of GA hyperparameter selection in general with these results, this achieves our secondary goal of constructing an optimal reduced network, as it is predicting with 100% accuracy with only 3 features and 2 hidden neurons.

Many combinations are further tried to derive results from the base model which could allow us to explore a wider selection of pruned hyperparameters where the GA does not prune all the hidden neurons. When the pruned model tends to perform poorly with less than 70% accuracy, the GA selects in the opposite manner by selecting the minimum boundaries. For this dataset with our model, selecting angular separations appears to be overly simplified by the GA and instead of determining degree of pruning, the GA is determining whether the model should be pruned at all. This may only be the case due to the nature of our data, however, we are unable to find features that could show different results. A different dataset and model may be necessary for this.

3.3 Performance of the GA

We reach the goal of determining the performance of GA hyperparameter optimization for our dataset. Compared to our previous paper [5], our automatically selected hyperparameters outperforms the manually selected hyperparameters by a large margin both for the base model and the pruned model. Showing the contrast with 250 epochs:





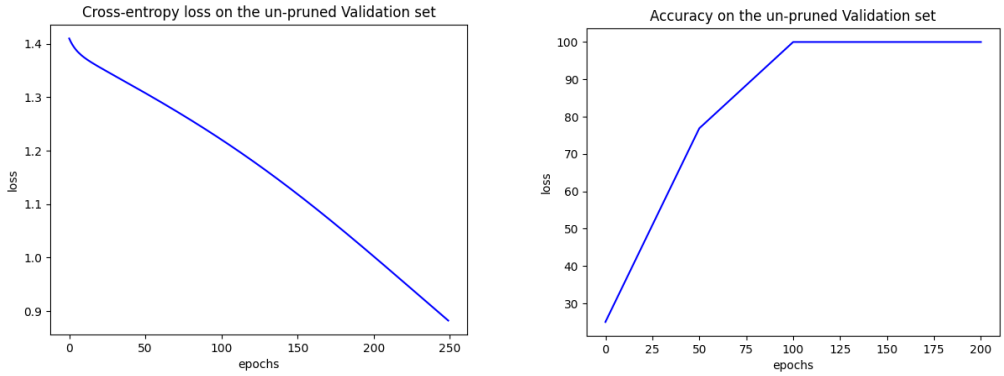


Fig. 5. Top: cross-entropy loss plot and accuracy on the validation set for GA-selected hyperparameters of 11 hidden neurons and 0.98 learning rate on the base model over 250 epochs. Bottom: cross-entropy loss plot and accuracy on the validation set for the manually selected hyperparameters of 50 hidden neurons and 0.03 learning rate from the previous paper on the base model over 250 epochs [5].

The GA optimized model learns exceptionally well with the chosen high learning rate, converging and reaching maximum accuracy by the 50th epoch on the base model. The manually selected hyperparameters from our last paper [5] is not close to convergence even by 250 epochs and only reaches 100% accuracy at 100 epochs. Although these results demonstrate the potential of GA hyperparameter optimization, better results could be achieved on a more complex dataset and model. Regarding the pruning technique, percentage-based pruning of an ordered list of angular separations ranked by importance may also be a more suitable technique when using a GA for pruning hyperparameter optimization, rather than pruning all which are above or below the angle boundaries. The pruned model cannot be compared with the pruned model in our previous paper, as a different pruning technique is used, although our GA is easily capable of consistently identifying if the model can be fully pruned down to 2 hidden neurons and still perform optimally.

Regarding the creation of a reduced model, a GA can identify the fewest number of hidden neurons necessary for a baseline model, lessening the need for a pruning technique. However, a pruning technique applied on top of a base model and retraining allows for more neurons to be pruned without sacrificing accuracy. We can show this by setting our base model GA parameters to a minimum hidden neuron boundary of 2, instead of 10. When this is done, the results are as shown:

Table 7. GA selected hyperparameters for the base model when the hidden neuron minimum boundary is set to 2 with 150 epochs, where the model has reached convergence.

Test	Hidden Neurons	Learning Rate
1	4	0.999
2	4	0.985
3	4	0.988

Further pruning and retraining allows for fewer hidden neurons in an optimal model.

It should be noted that there was some difficulty with creating a suitable evaluation function for our two genetic algorithm models. Some trial-and-error was necessary to ensure that our function was selecting the correct individuals while maintaining our goal of creating a network which was as reduced as possible and not overfitting.

4 Conclusion and Future Work

This work constructed a baseline network and a re-trained pruned network using a linear classification model with two separate genetic algorithms used to optimize hyperparameters. We have found that GA hyperparameter optimization is highly effective for tuning both our base model and our pruned model and can create an optimal reduced network. The learning rate required for our dataset to achieve low validation loss, a smooth loss curve and 100% accuracy was much higher than expected and had not been tried in our last paper. For our dataset, we managed to reduce the network down to only 2 hidden neurons. With all cases and with all numbers of features, our GA could find hyperparameters which consistently resulted in our model learning at a good rate and predicting with 100% accuracy. The distinctiveness pruning method was not very compatible with the GA as a hyperparameter, as it tended to stick to the minimum or maximum bounds rather than utilize a range to determine

the number of neurons to prune. It is possible that percentage-based distinctiveness pruning or implementing an angular separation priority function would remedy this issue. While the GA allowed us to find a learning rate which was a big improvement to previous attempts, the boundary we set to run the tests appeared to be too low. It was unexpected that the model could learn so well with such a large learning rate, although there is concern that it would overfit. Further tests could be done with adding in more data or features to diagnose any issues with overfitting. As noted, there were minor issues with defining an evaluation function for our models and this was a case of trial-and-error. On a case-by-case basis, there may be some models on some datasets where defining the evaluation function may be just as time-consuming as manually tuning hyperparameters, and this is a consideration to make when deciding whether GA hyperparameter optimization is suitable for the task at hand.

These results may not represent the capabilities of GA hyperparameter optimization for all types of models with all types of datasets. Our model is shallow, with only one hidden layer, and our dataset is simple – meaning that the patterns are easily discernible for the network. This may not always be the case and the patterns may not be so obvious in more complex datasets. We have only tested the tuning of hidden neurons, learning rate and angular separations. Different types of models have many different hyperparameters which may be tuned, and these were not explored in this paper. Determining the efficiency of GA hyperparameter tuning may be more suitable on Deep Neural Networks with large configuration spaces where hyperparameters such as optimizer types, activation types and drop-out rate can be tuned [4]. Future work on this technique must include different types of models and a variety of datasets, especially on DNNs and tree-based algorithms with many potential hyperparameters outside of hidden neuron number and learning rate. We do not explore using a GA as a feature selection tool, and this could be a consideration in the future.

References

1. Schmidt, J., Marques, M. R. G., Botti, S. & Marques, M. A. L. (2019) “Recent advances and applications of machine learning in solid-state-materials science,” Npj Computational Materials, 5(1). doi: 10.1038/s41524-019-0221-0
2. Shawi, R. E., Maher, M., Sakr, S. (2019) “Automated Machine Learning: State-of-the-art and open challenges” arXiv preprint arXiv: 1904.12054, https://arxiv.org/abs/1904.12054
3. Feurer M., Hutter F. (2019) Hyperparameter Optimization. In: Hutter F., Kotthoff L., Vanschoren J. (eds) Automated Machine Learning. The Springer Series on Challenges in Machine Learning. Springer, Cham. https://doi.org/10.1007/978-3-030-05318-5_1
4. Yang, L., Shami, A. (2020) “On Hyperparameter optimization of machine learning algorithms: Theory and practice,” Neurocomputing, 415, 295-316. doi: 10.1016/j.neucom.2020.07.061
5. Wright, J. (2021) “Investigating the Results of Hidden Neuron Pruning in the Classification of SARS Symptoms,” 4th ANU Bioinspired Computing Conference
6. Gedeon, T. D., Harris, D., (1991). “Network Reduction Techniques,” Proc. Int. Conf. on Neural Networks Methodologies and Applications, AMSE, San Diego, vol. 2, pp. 25-34, 1991.
7. Gedeon, T. D., (1995). “Indicators of Hidden Neuron Functionality: the Weight Matrix Versus Neuron Behaviour,” Proc. Int. Conf on Artificial Neural Networks and Expert System (ANNES ’95), IEEE Computer Society, USA, pp. 26, 1995.
8. Mendis, B. S., Gedeon, T. D., & Koczy, L. T. (2005). “Investigation of aggregation in fuzzy signatures”, Proc. Int. Conf. on Computational Intelligence, Robotics and Autonomous Systems, Singapore, 2005.
9. Louizos, C., Welling, M. and Kingma, D. P. (2018) “Learning sparse neural networks through l0 regularization,” in International Conference on Learning Representations, 2018.
10. Thimm, G., & Fiesler, E. (1997). High-order and multilayer perceptron initialization. Neural Networks, IEEE Transactions on, 8(2), 349-359.