Evolutionary Algorithm for Input Feature Selection and Network Pruning in Brainwave Classification

Siddharth Sachan

Research School of Computer Science
The Australian National University, Canberra ACT 2601, Australia
siddharth.sachan@anu.edu.au

Abstract. When the amount of data available for training is low, designing the neural network optimally becomes much more important. Designing process includes (1) deciding upon which features to use and (2) what should the size of the hidden layer be. In this paper, we have implemented input feature selection and network pruning through genetic algorithm. For the purpose of training, we use electroencephalograms(EEG) from 24 participants listening to 3 different genres of music. The data was pre-processed and used to train baseline one hidden layer network. Using genetic algorithm we have shown that a subset with smaller number of features can be found with higher validation classification accuracy. We use genetic algorithm to prune the one hidden layer network as well. This is compared with the distinctiveness pruning which uses activations of the hidden neurons as functionality measure. We show good performance of the validation set. However, due to overuse of the validation data by the genetic algorithm we report poor performance on the test set, implying severe overfitting to the validation data. Some possible solutions are also discussed.

Keywords: Artificial Neural Network · Network Pruning · Feature Selection · Genetic Algorithm · Electroencephalogram · Classification

1 Introduction and Background

Since training deeper and wider Artificial Neural Networks (ANNs) has become seamless and fast, these networks are often over-parameterized. Researchers have shown that shallow networks can achieve performance at par with deep and complex networks [1, p. 1]. Over-parameterization happens as there is no standard way to determine the width (number of hidden neurons in a layer) and depth (number of hidden layers) for a given task. Even when designing a fully connected neural network with just one hidden layer, the most challenging task is to set the number of hidden neurons in the layer. We often use many different values and select one based on performance. However, this approach does not alleviate the issue of over-parameterization, which might result in longer run time and more memory costs. One way to resolve this is via network pruning. Broadly, after training a model, the functionality of the hidden units is determined by some criteria, and similar neurons are removed. The goal of network pruning is thus to reduce the size of the network with the least affect on its performance.

Moreover, when the amount of data available is relatively small, training wide and deep models often lead to poor generalization results [3, p. 2]. Input feature selection is often used to restrict the size of the network in such cases. The goal here is to select a subset of features which results in maximum performance. One might argue that training with all the features will eventually lead to less importance (weights) given to less significant features. These less important features act like the noise which our models must learn to ignore. However, when the amount of data is low, a variety of regularization techniques are required to avoid overfitting (model learning the noise).

Both the network pruning and feature selection can be seen as NP hard problems where we try to find a subset which performs the best. A brute force approach will be to evaluate the performance for every possible subset of hidden neurons [7, p. 2-3] or features. However, this becomes intractable as the search space grows exponentially with the number of neurons or features (2^n, where n is the number of number of neurons or features). Researchers have experimented with a variety of evolutionary algorithms to tackle this issue[10]. Evolutionary algorithms are iterative methods, where we choose the subsets for next iteration (generation) smartly. In this work, we use genetic algorithm, the most popular subset of evolutionary algorithm[10]. Genetic Algorithm is a meta-heuristic inspired by the process of natural selection i.e. survival of the best. The basic idea is the subsets which perform the best in a generation, survive to the next generation. We will discuss more about this in the methodology section. Given that we iterate through enough generations, the algorithm should result in good enough locally optimal solution. The solution will not be a globally optimal as we do not search over the whole space.

Additionally for network pruning, *Relevance*[5,9], *contribution*[8] and *distinctiveness*[2] functionality approaches have been experimented with in the past. In this work, we also compare the genetic algorithms and distinctiveness

pruning. The distinctiveness of a hidden neuron is determined from the activation vector over all training examples presented to the model. We use the angular separation between the activation vectors of different neurons to understand their similarity. More on this in the methodology section.

We use brainwave classification task for our experiments. The impact of music on the brain has been studied to understand and predict the emotions that a particular genre of music incites [4]. Electroencephalography is an electrophysiological method to record electrical signals emanating from brain. The resulting electroencephalogram (EEG) is used to understand brain activity and determine conditions like epilepsy, stress, stroke and sleep disorders [6]. Recently, researchers have proposed EEG-based emotion recognition using neural networks [11,6]. In this work, we have EEG data for 24 participants for 8 different songs belonging to 3 music genre (classical, instrumental and pop). The data has been band-pass-filtered into three frequency bands (α: 8-12 Hz, β: 12-30 Hz, γ: >30 Hz). Also, we only use the 25 summary statistics (Table 1 [6, p. 4]) from the F7 electrode to simplify the classification task. The summary statistics (referred to as features) provide a simple way of analysing EEG signals, as raw EEG data also has a time component. F7 electrode is chosen as it was shown to produce the most useful feature for classification [6]. Moreover, using only a single channel data allows us to determine the viability of simpler and cheaper methods of data collections (for example a single channel EEG sensor) for brainwave classification.

The goals of this work are:

1. Implement a baseline one hidden layer network for music genre classification.
2. Implement genetic algorithm for input feature selection to determine the subset of features which results in the best performing model.
3. Implement genetic algorithm for network pruning to determine the subset of hidden neurons which results least drop in performance.
4. Implement distinctiveness pruning to remove similar neurons in the hidden layer.
5. Compare the pruning results of pruning through genetic algorithm and distinctiveness.

In the following sections, we first describe the methodology used to achieve these goals. Then, we discuss the results and limitations of the methods to gain a deeper understanding of the problem. Finally, we conclude the paper with some interesting future research possibilities.

2 Data and Methodology

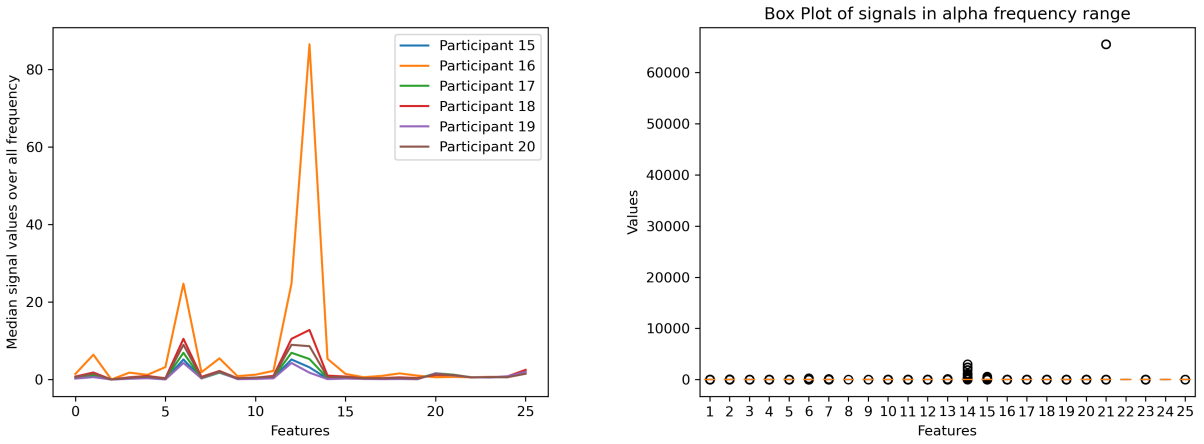
Here we discuss the pre-processing we apply to the data to make it viable for training and testing. We also discuss the methods used to achieve the goals mentioned above.

2.1 Data Pre-processing

We have 576 rows of data which means 192 rows for each α, β and γ waves. Each line has 25 features and a target music genre value in the set {1=classical, 2=instrumental, 3=pop}. First, we plot the median signal values (over the three frequency bands) to understand the participants' effect on the EEG data. Fig. 1a shows this plot for participant 15-20. We observe that the median values are consistently higher for participant 16 as compared to others. So if we normalize the signal column-wise (feature-wise) for all the participants, then in the normalized data, signals for some participants (for example, participant 19 here)can become too low. Hence, we will normalize each feature w.r.t. each participant and not the whole column (feature).

Before, we can apply the normalization we separate the α, β and γ data and make box-plots to determine the presence of outliers. Fig. 1b shows the box plot for the 25 features for α waves. We clearly observe an outlier point for the 21^{st} feature. Looking at the raw data, we find that the value of 63535 (which is interestingly $2^{16} - 1$, might be because of underflow) is present for four observations. We replace these with the median value of the 21^{st} feature for α range. Then, we apply the min-max normalization for each feature w.r.t. each participant. For the target value we subtract 1 from all the rows so as to make the class labels = {0,1,2}. This gives us 3 sets of normalized data, each having 192 rows where each row has 25 features and one target class. We can combine the three sets to get a maximum of 75 features for the input.

Finally, we shuffle the data (the 192 rows) and split it into training, validation and test set with 80/10/10 split. We do this without taking participant ids into consideration as researchers have shown that the behaviour of brain waves has commonalities that transcends participants and sessions [11]. We also check if the validation and test splits have roughly the same number of examples from each class (to check for the bias in these sets).



(a) (b)

Fig. 1: Distributions of the features. (a) Median EEG signal for the 25 summary statistics (features) from F7 electrode for different participants. We observe that the EEG signals are consistently higher for participant 16 as compared to other participants. If normalization is done over all the participants, then in the normalized data, signals for some participants might become too small. (b) Box plot of the 25 summary statistics (features) for α frequency range. We observe a clear outlier for feature number 21. This value is 65535 (interestingly $= 2^{16} - 1$) in a column where the median value is around 0.9.

2.2 Baseline Network and Training

Implementation of the network and training is very straightforward in PyTorch. Following are some key steps:

- We implement 1 hidden layer fully connected neural network, with Sigmoid activation in the hidden layer. So, the only hyper-parameter in the network structure is the number of hidden neurons.
- We use Adam optimizer (learning rate = 0.003) as it is faster than SGD.
- Cross-entropy loss is used as this is a classification task with three classes.
- After every epochs of training, we run the model on validation set and if the validation accuracy increases, we save the model as a checkpoint.
- We use batch size of 2 for training as the data set is relatively small.
- For validation and testing we use batch size of 8 to improve the speed.
- We run the training for 200 epochs and plot average training and validation loss over epochs.
- We train the models 20 times and report the best performance to account for the uncertainty due to initialization.

We also use F1 score as a performance measure to get some idea about the false positive and false negatives. Moreover, for the present task of multi-class classification, we weight the scores (as F1 score is for binary classification) as per the presence of class in the set. These weights accounts for possible class imbalance in the validation and test sets.

2.3 Genetic Algorithm

In a genetic algorithm, a population of candidate solutions (called individuals or phenotypes) is evolved toward better overall solution. Each phenotype has a genotypic encoding called chromosomes for which we use a string of binary values, where 1 is for the features (or neurons) included in the candidate solution and 0 is for the features (or neurons) not included in the candidate solution. The general outline of the algorithm used (inspired from this review paper [10])is as follows:

- First step is the the **initialization** of the population, where we randomly generate candidate solutions. Here we sample from uniform distribution which implies that in the first generation (population in each iteration is called a generation), the approximate number of features (or neurons) selected in the candidate solutions is half the total number of features (or neurons).

- In each generation, the **fitness** of every individual in the population is evaluated. Here, we use a combination of classification accuracy (on validation set) and the number of features selected by the individual as fitness score.

$$\text{fitness score} = \text{validation accuracy} - (\text{length factor}) * \text{size} \tag{1}$$

  Eq. 1 shows our desire of high validation accuracy with less size. Here, size refers to the number of 1s in the chromosome i.e. size of the input layer (input feature selection) or the size of the hidden layer (network pruning). Hence, we want smaller networks with good validation accuracy. Length factor controls the importance given to the size. We will see how it impacts the average size of the population as the number of generations increase.
- Then comes the **selection** step where more fit individuals are selected from the population. We implement probabilistic and top selection. In probabilistic approach, we choose individuals with probability proportional to the fitness score of the individual (if fitness score is less than 0, then 0 probability is assigned to the individual). This is done with replacement so more probable individuals can be selected multiple times. However, this can create issues when the performance of the whole generation is similar. In top selection, we select the top few individuals in a population. These are repeated so that the resulting population is of the same size as the initial population.
- The selected individuals are recombined randomly in the process called **cross-over**. For each individual in the population resulting after selection, we decide whether or not to cross over with a cross-rate. If the cross rate is 0.8, the 80% of the population is chosen for crossover. This ensures that some of the individuals from previous generation are present and the performance does not drop. If an individual (A) is selected, we randomly select another individual (B) from the population for uniform cross-over. For uniform cross-over, random points are selected from B and copied to A to get a single chromosome.
- Finally, random **mutations** are added to the chromosomes to ensure exploration. We flip all the element values (0 becomes 1 and 1 becomes 0) with a fixed probability (mutation rate). If the mutation rate is high, more exploration is done over the course of the optimization. The population generated after mutation is ready for next generation.
- This process of fitness, selection, cross-over and mutation is repeated over a specified number of generations or until a satisfactory fitness level has been reached for the population.

Keeping the general approach in mind, the genetic algorithm implementations for input feature selection and network pruning are as follows:

Feature Selection

1. We **initialize** a population of random chromosomes of size 75 (α, β and γ combined).
2. For each chromosome in the population we select the features as represented by the chromosome. A simple network (similar to baseline) is trained using these features. The validation accuracy and the number of input features (size) is used to calculate the **fitness** (Eq. 1) of the model.
3. We do top or probabilistic **selection** after calculating the fitness.
4. Then, **cross-over** and **mutation** is done according to cross rate and mutation rate (hyper-parameters).
5. For each generation we display (and log) the best, average and variance in the validation accuracy. We also record the average size over the generation. We also display the best performing chromosome which is logged separately (as the run time is significant).
6. Steps 2-5 are repeated for a specified number of generations.

Network Pruning

1. We **initialize** a population of random chromosomes of size 20 (the number of hidden neurons).
2. For each chromosome in the population we copy the weights for selected neurons (as represented by the chromosome) to an new network (of appropriate size). The validation accuracy of the new network and the number of hidden neurons selected (size) is used to calculate the **fitness** (Eq. 1) of the model.
3. We do top or probabilistic **selection** after calculating the fitness.
4. Then, **cross-over** and **mutation** is done according to cross rate and mutation rate (hyper-parameters).
5. For each generation we record the best and average validation accuracy and F1 scores. We also record the average number of hidden neurons in the new model. We also display the best performing chromosome.
6. Steps 2-5 are repeated for a specified number of generations. The best performing pruned model (based on validation accuracy) is saved.

2.4 Network Pruning through distinctiveness functionality

We use cosine similarity between activations as a measure of neuron functionality. We perform sequential pruning, where only one neuron is removed in one step. This is much easier to implement than working the logic to remove multiple neurons given the set of similar neurons, but less efficient. For pruning, we implement the following steps:

1. Stack all the activations from the training set to get a h×n tensor. Here, h is the number of hidden units and n is the number of samples in the training set.
2. Next, we subtract the mean, so that approximately half of the activations are negative and angles are in range 0 to 180 degrees.
3. These activations are used to calculate a h×h cosine similarity matrix.
4. Then depending upon the threshold (angle in degrees), we compute all the similar and complementary pairs of neurons.
5. We take the first similar pair of neurons (say [a,b], such that a>b). We remove the neuron with higher index (a) and make the weights and biases associated with the other neuron (b) as the average of both the the neurons.
6. If there are no similar pairs, we do same thing with complementary pairs. However when taking average we subtract the weights (and biases).
7. Evaluate the pruned model's accuracy on training and validation sets.
8. Step 1 to 7 are repeated for this pruned model, until no more similar or complement pairs are found. The best performing pruned model (based on validation accuracy) is saved.

3 Results and Discussion

In this section, we first discuss the results obtained upon training models with α, β and γ wave data. This forms our baseline. Then, we use genetic algorithm to find a subset of features which gives the best validation performance. Finally, we compare the results of the two approaches to pruning.

3.1 Baseline

We train 6 models, taking 3 data-sets (α, β and γ waves) and 2 different hidden units count (h ={10, 20}). The best (according to validation accuracy, for 20 runs) training and validation performance are tabulated in Tab. 1. We also show the validation and training losses for α (Fig. 2a), β, γ (Fig. 2b) and α, β, γ (Fig. 2c) features. Looking at the evaluation measures and losses for validation sets, we can see that the models do well on training set but do not generalize well for any case. However, we also see that the models using only the α wave data perform the poorest. α waves mostly correspond to relaxed state of mind and are boosted during meditation. Thus, one would expect co-relation between α waves and music only when the participant finds the music relaxing (we do not know this). Both β and γ correspond to cognitive activity, alertness and activity and hence have better chances of co-relation with what participant feels/thinks during the experiment. This is shown in our experiments as the models using β and γ wave data outperform the models using α wave data.

Model	Training Accuracy, F1 score	Validation Accuracy, F1 score
1 (α, $h = 10$)	0.6732, 0.6682	0.3684, 0.3392
2 (α, $h = 20$)	0.6797, 0.6781	0.4211, 0.4152
3 (β, $h = 10$)	0.6275, 0.6227	0.5789, 0.5684
4 (β, $h = 20$)	0.7124, 0.71	0.5263, 0.5252
5 (γ, $h = 10$)	0.6405, 0.6407	0.4737, 0.5097
6 (γ, $h = 20$)	0.6471, 0.6463	0.4737, 0.4907
7 (β, γ, $h = 20$)	0.9934, 0.9935	0.5263, 0.5627
8 (α, β, γ, $h = 20$)	0.7758, 0.7731	0.4263, 0.4347

Table 1: The best accuracy and F1 Score evaluation on the training and validation set for different models.

We know that wider models have more trainable parameters. With these observations in mind, we train two more model v.i.z. Model 7: using β and γ wave data and $h = 20$ and Model 8: using α, β and γ wave data and $h = 20$. Model 8 is the most general model, it uses all the features available for training. If enough data was available to model 8, it should be able to figure out which features are important by learning weights accordingly. But, we see that model 8 performs worse than other models using less number of features. Hence, just adding more features does not guarantee better performance. This shows the need for a feature selection operation before training, which is addressed by the genetic algorithm for input feature selection.

(a) α wave data, 25 input features (b) β, γ wave data, 50 input features (c) α, β, γ wave data, 75 input features

Fig. 2: Plots of average training (blue) and validation (orange) loss during training where h=20. The models generalises poorly for both the cases. However, we see that for the model using α wave data, the validation losses increase more sharply than that for the one with β, γ wave data.

3.2 Input feature selection by genetic algorithm

From our baseline experiments we observe that the accuracy and F1 score (Table 1) agree with each other in most cases. This justifies our choice of using only the validation accuracy during selection (Eq. 1). For training, we use number of hidden neurons = 20, batch size = 2, Adam optimizer (learning rate =0.003) and train for 200 epochs. The best validation accuracy achieved during genetic algorithm is shown in Table 2 along with the hyper-parameter settings of the genetic algorithm. Referring to the experiments in Table 2, we make the following observations:

S. No.	Population size	Number of generations	Length factor	Selection method	Cross rate	Mutation rate	Validation accuracy	Number of selected features
1	20	5	0	Top 5	0.8	0.1	0.7368	33
2	30	5	0	Top 5	0.8	0.1	0.7368	33
3	10	10	0	Top 3	0.8	0.1	0.7368	34
4	10	10	0.1	Top 3	0.8	0.1	0.7895	40
5	10	10	0.2	Top 3	0.8	0.1	0.7368	33
6	10	10	0.25	Top 3	0.8	0.1	0.7368	37
7	10	10	0.5	Top 3	0.8	0.1	**0.8421**	40
8	10	20	0.1	Top 3	0.8	0.2	0.7894	34
9	10	20	0.1	Top 3	0.8	0.05	0.7368	33
10	10	10	2	Top 3	0.8	0.1	0.7368	32
11	10	10	0.5	Probabilistic	0.8	0.1	0.6316	33
12	10	10	1	Probabilistic	0.8	0.1	0.6842	26
13	10	10	2	Probabilistic	0.8	0.1	0.6316	36

Table 2: Experiments with different hyper-parameter settings for input feature selection using genetic algorithm. The best validation accuracy achieved is in bold.

1. Impact of *population size* - From 1,2 and 3: we are able to achieve same performance level even with smaller population. Since, larger population implies longer run time, we stick to population size of 10 for the rest of the experiments.
2. Impact of *number of generations* - From 4,8: There is no improvement in running the algorithm for 20 generations instead of 10 generations. As more generations need more run time, we run the algorithm for 10 generations.
3. Impact of *mutation rate* - From 4, 8, 9: With lower mutation rate (0.05), we observed that the best chromosome would often not change from generation to generation. This results in low exploration, variance in the validation accuracy across the population lower as compared to mutation rate 0.1 and 0.2. It is harder for the algorithm to find solutions which match the performance with higher mutation rate. For mutation rate of 0.2, although we are able to achieve similar best performance (as compared to mutation rate 0.1) the mean accuracy stays around 45%-50% (Fig. 3). This show that as a whole population, the evolution is not working as the average performance stays stagnant. For mutation rate of 0.1, we observe the mean accuracy goes up to 63% for some generations, which shows that the whole population gets better. It still fluctuates between generation which

shows some exploration is still done. Hence we see that mutation rate of 0.1 keeps enough variance while still becoming better as a population. Hence, we choose mutation rate of 0.1 for most of the experiments.

4. Impact of *selection method* - In the probabilistic selection, we observed that for some generations the probability distribution across the generation becomes flat (for population size 10, they all become close to 0.1). This essentially makes the selection process random for that generation. We expect this phenomena to slow down the evolution as better individuals are not guaranteed to be passed on. From 11-13 on the table, we observe that the best validation accuracy achieved by probabilistic selection is worse than that by the top selection.
5. Impact of *length factor* - We expected length factor to regulate the average number of the features selected. As the evolution goes on (generation count increases), we expect the average number of the features selected (referred to as the size) to decrease if the length factor is increased. However, we did not observe a clear relationship. Moreover, the best performing has similar size across all the experiments.
6. *Comparison with the baseline* - All the experiments perform better than any of the baseline models. We observed that some generations might have poor performance (the validation accuracy of the best chromosome in that generation is less than 0.45), but running through multiple generations we are able to consistently achieve around 75% validation accuracy. The best validation accuracy we achieved in our experiments is 84.21%. It is not clear why it occurs in the particular parameter setting (factors involved - the chromosome and the network initialization), but the corresponding chromosome is used for pruning studies.

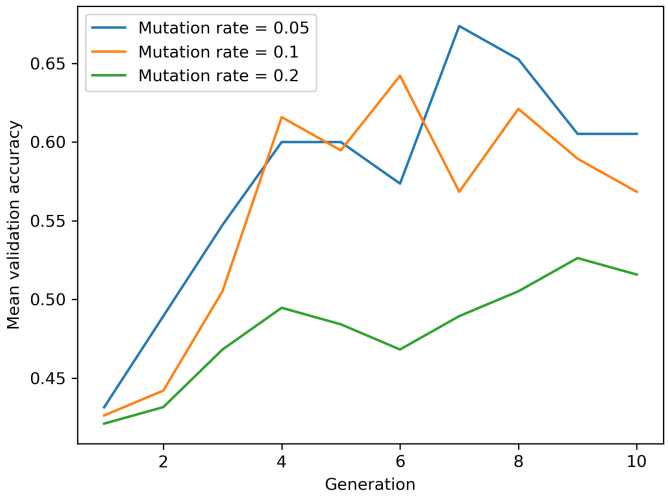
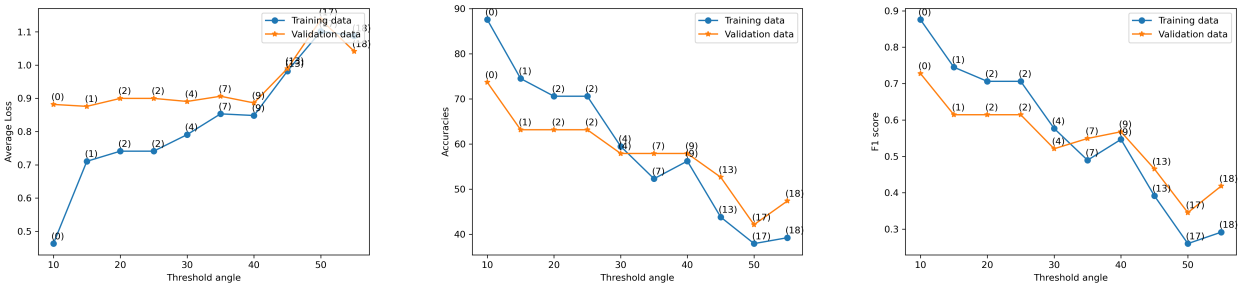


Fig. 3: Impact of mutation rate on the general performance across the generation. We see for high mutation rate (0.2) the population as whole performs worse as compared to lower mutation rates (0.05, 0.1).

3.3 Pruning: Distinctiveness functionality vs Genetic Algorithm

We take the chromosome corresponding to the best validation accuracy achieved in input feature selection. Then we train 20 models using the features as represented by the chromosome. The best model among these is selected as the baseline for pruning studies. It has 40 input features, 20 hidden neurons and achieves validation accuracy of 73.68%.

To study the affect of distinctiveness pruning on the model, we remove all the similar or complementary neurons within a threshold and evaluate the resulting model. The threshold is varied to understand the impact of removing nearly similar neurons as compared to distinct enough neurons from the model. We use the average cross-entropy loss, accuracy and F1 score on training and validation set as the evaluation measures (Fig. 4). The average loss plot (Fig. 4a) shows that the validation losses do not increase much even after removing 9 neurons, while the training loss almost doubles and becomes similar to the validation loss. This shows that removing the neurons which over-fit the training data can be removed without much impact on the evaluation measures. However, the accuracy (Fig. 4b) and F1 score (Fig. 4c) show that there is bigger drop for the first neuron removed than the next one. This implies that the first removed neuron has more functionality than the second removed neuron. This is contradictory to the distinctiveness assumption where the neurons with smaller angle between activations are less distinct. This shows that there is no one to one simple relation between the angle between activations and functionality (in terms of accuracy and F1 score).



(a) Average loss (b) Accuracy (c) F1 score

Fig. 4: Performance after distinctiveness pruning on training (blue) and validation (orange) sets plotted against the threshold angle (in degrees). The number inside the braces (.) shows the number of neurons pruned. Except for the first neuron, we observe only a little loss in performance even for threshold of 40°.

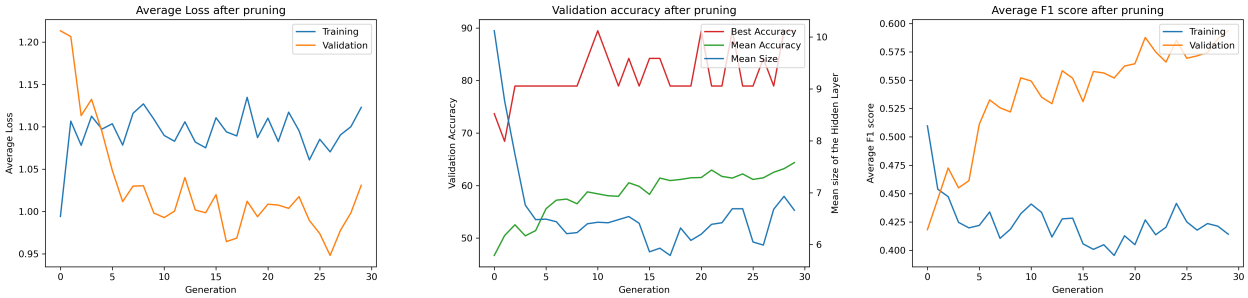
In distinctiveness pruning we remove the similar or complementary neurons and adjust the weights. In pruning through genetic algorithm, we simply use a subset of hidden layer neurons; no adjustment in the weights is required. The hyper-parameters available for tuning are similar to the input feature selection. However, as the process is much faster than training networks, we do not focus much on population size and number of generation. We set high enough population size (100) and number of generation (30). Table 3 shows the results for different hyper-parameters. Referring to the serial numbers in Table 3, we make the following observations:

S. No.	Length factor	Selection method	Cross rate	Mutation rate	Validation accuracy	Number of selected neurons (Size)	Average size in last generation
1	0	Top 25	0.8	0.05	0.8948	9	10.79
2	0	Top 25	0.8	0.1	0.8948	9	10.98
3	0	Top 25	0.8	0.2	0.8421	10	11.02
4	0	Top 25	0.5	0.1	0.8948	9	11.94
5	0	Top 25	0.9	0.1	0.8948	9	11.42
6	0	Probabilistic	0.8	0.1	0.8421	10	10.12
7	1	Probabilistic	0.8	0.1	0.8948	11	9.89
8	2	Probabilistic	0.8	0.1	0.8948	10	9.53
9	1	Top 25	0.8	0.1	0.8948	10	10.36
10	2	Top 25	0.8	0.1	0.8948	7	9.13
11	3	Top 25	0.8	0.1	0.8948	7	7.84
12	4	Top 25	0.8	0.1	0.8948	7	6.66

Table 3: Experiments with different hyper-parameter settings for pruning using genetic algorithm.

1. Impact of *selection method* and *cross rate* - From 2, 4-8: We observe that there is little impact of these hyper-parameters on the performance. The locally optimal solutions obtained in these experiments are equivalent.
2. Impact of *mutation rate* - From 1-3: We see that for higher mutation rate (0.2), the validation accuracy achieved is less than that for that for 0.1 or 0.05 mutation rate. Mutation rate of 0.1 achieves the best accuracy while maintaining more diversity as compared to mutation rate of 0.05. Hence, we choose mutation rate of 0.1 for the rest of the experiments.
3. Impact of *length factor* - From 9-12: The average size of the population in the last generation decreases with increase in the length factor. This is expected as increasing the length factor puts more importance on less wide networks (less hidden neurons) which are good enough. We also see that, the best network is also less wide for high length factor. Hence, by increasing the length factor we can guide the genetic algorithm to find solutions where less number of neurons remain after pruning.
4. How does the evolution look? - Fig. 5 shows the loss, accuracy, size and F1 scores averaged over all the individuals in a generation. We see as the evolution proceeds the losses decrease and the accuracy and F1 score increase. This shows that the population becomes better as a whole as the evolution goes on. The blue curve in 5b shows that the mean size settles around 6.5 in the first few generations implying the size regulation happens quickly and further decreasing the size results in worse models.

5. *Comparison* with baseline (unpruned) and distinctiveness pruning - Baseline model had validation accuracy of 73.68% which distinctiveness pruning is unable to improve upon. However, through genetic algorithm we are able to prune the model while increasing the validation accuracy for all the hyper-parameter settings, where the best validation accuracy achieved is 89.48%. This shows that the baseline model is severely over-fitting to training data and we **may be over-fitting to the validation data now** as we have run huge number of experiments based on the validation set. We run the model on the test to determine if this is the case.



(a) Average loss (b) Average accuracy and size (c) Average F1 score

Fig. 5: Performance during pruning through genetic algorithm for hyper-parameter setting on S. No. 12 in Table 3. From (a) and (c) we observe that the average validation loss (orange) goes down and average F1 score increases as evolution proceeds. This shows that the population as a whole becomes better with evolution. The blue curve in (b) shows that the mean size settles around 6.5 in the first few generations.

Finally, we tabulate the evaluation of the (1) baseline model, (2) model achieved after distinctiveness pruning (threshold =40, 9 neurons removed) and (3) the best model after genetic algorithm pruning to determine if we are over-fitting to the validation data. This is essential, especially for the **genetic algorithm as we run huge number of experiments based on the validation data which might bias our results**. Table 4 shows that the model obtained after genetic algorithm pruning suffers from over-fitting to the validation data. Even our baseline model, is only as good as a random guess on the test set. This implies that somewhere our **assumption about the data are wrong**. We need more experiments with different validation strategies like leave two-participants out, one for validation, one for testing. However, implementing this with genetic algorithm will lead to quite long run times.

Model	Validation Accuracy	Validation F1 score	Test Accuracy	Test F1 score
1	0.7368	0.7278	0.35	0.3534
2	0.6315	0.6144	0.35	0.3252
3	0.8948	0.8948	0.20	0.2016

Table 4: Accuracy and F1 Score on the validation and test set for models before and after pruning. We see that the test performance of the model achieved after genetic algorithm is poor. This implies over-fitting to the validation set.

4 Conclusion and Future Work

In this work we have implemented genetic algorithm for input feature selection and network pruning. Both these problems are basically NP hard, where the search space grows exponentially with the number of features/neurons involved. First, we analysed the data to remove outliers and normalize in the best way possible. Then, baseline models were trained using different predefined sets of the features. For the most general case (using 75 input features), we are able to get the best validation accuracy of 42.63%. Upon using genetic algorithm we are able to increase this to 84.21% for input size of 40 features. Finally, we prune this model using genetic algorithm where the size of hidden layer is reduced from 20 to 7, with a further increase in the validation accuracy to 89.48%. We also compare the pruning results to the pruning obtained by distinctiveness functionality. We note that the **genetic algorithm uses the validation data for huge number of experiments** and hence expect some over-fitting to

the validation set. However, we see that there is severe over-fitting to validation data, leading to models which are not viable for use.

This failure implies that some of our assumptions about the data are not correct. One reason could be that our train-validation-test split strategy is wrong. We can try a different one, like leave two-participants out strategy, one for validation and the other one for testing. However, this requires to requires to train model repeatedly (like leave one out), which increases the run time by several times. Moreover, the test example left out will not be reflective of the data, as each participant only listens to songs from 2 genres, which means one target will be missing from the test set. Another assumption we implicitly made is that genre classification is possible from the summary statistics of the F7 electrode. This can go wrong in two ways v.i.z. (1) summary statistics are not enough or (2) only F7 electrode signals are not enough. To address (1), we can try to extract features from the time series data and for (2) we can work with EEG data from all the receptors and figure out important electrodes for classification task. In both cases more data and processing is required.

Acknowledgement The author would like to thank the participants who took part in the original experiment and Professor Tom Gedeon, Research School of Computer Science, The Australian National University Canberra, Australia for providing the dataset for this study.

References

1. Ba, L.J., Caruana, R.: Do deep nets really need to be deep? In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. p. 2654–2662. NIPS'14, MIT Press, Cambridge, MA, USA (2014)
2. Gedeon, T.: Indicators of hidden neuron functionality: the weight matrix versus neuron behaviour. In: Proceedings 1995 Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems. pp. 26–29 (1995). https://doi.org/10.1109/ANNES.1995.499431
3. Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R.P., Tang, J., Liu, H.: Feature selection: A data perspective. ACM Computing Surveys (CSUR) **50**(6), 1–45 (2017)
4. McCraty, R., Barrios-Choplin, B., Atkinson, M., Tomasino, D.: The effects of different types of music on mood, tension, and mental clarity. Alternative therapies in health and medicine **4**(1), 75–84 (1998)
5. MOZER, M.C., SMOLENSKY, P.: Using relevance to reduce network size automatically. Connection Science **1**(1), 3–16 (1989). https://doi.org/10.1080/09540098908915626
6. Rahman, J.S., Gedeon, T., Caldwell, S., Jones, R.: Brain melody informatics: Analysing effects of music on brainwave patterns. In: 2020 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2020). https://doi.org/10.1109/IJCNN48605.2020.9207392
7. Reed, R.: Pruning algorithms-a survey. IEEE Transactions on Neural Networks **4**(5), 740–747 (1993). https://doi.org/10.1109/72.248452
8. SANGER, D.: Contribution analysis: A technique for assigning responsibilities to hidden units in connectionist networks. Connection Science **1**(2), 115–138 (1989). https://doi.org/10.1080/09540098908915632
9. Segee, B., Carter, M.: Fault tolerance of pruned multilayer networks. In: IJCNN-91-Seattle International Joint Conference on Neural Networks. vol. ii, pp. 447–452 vol.2 (1991). https://doi.org/10.1109/IJCNN.1991.155374
10. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. IEEE Transactions on Evolutionary Computation **20**(4), 606–626 (2015)
11. Zheng, W.L., Lu, B.L.: Investigating critical frequency bands and channels for eeg-based emotion recognition with deep neural networks. IEEE Transactions on Autonomous Mental Development **7**(3), 162–175 (2015). https://doi.org/10.1109/TAMD.2015.2431497