Using Genetic Algorithms to Optimise Hyper-Parameters for a Constructive Cascade Neural Network*

Rhys Frame

[1] u6948741@anu.edu.au
[2] Research School of Computer Science, Australian National University

Abstract. A problem that occurs when constructing artificial neural networks is choosing the size/topology of the network. The success of a neural network heavily relies on the suitability of the architecure used on the given problem, and the optimal network topology is hard to determine a priori for more complex problems. Constructive cascade algorithms are used to progressively create a neural network by adding neurons onto an existing network so that the size of the network can be determined by the network. Although the creation of the network is done by the algorithm, there are still choices to be made on hyper-parameters like the initial size of the network and the learning rate. We can use a genetic algorithm to optimise these parameters, so that the entire network is generated by an algorithm, and nothing is left for the user to choose. A simple neural network with one hidden layer was modified to utilize a cascade algorithm in different ways based on knowledge of the input data. The hyper-parameters of this network were then optimised using a genetic algorithm. The algorithms tested get similar results to a simple neural network structure on a dataset that recorded the effect of different types of music on brain activity.

Keywords: Constructive Cascade Algorithm · Neural Network Architecture · Genetic Algorithm

1 Introduction

Constructive cascade neural networks are feedforward neural networks where the network is constructed in the learning phase. This is predominately seen with the Cascade-Correlation algorithm [1]. This algorithm starts with the input fully connected to the output with no hidden neurons, and trains this network until a plateau in training loss, where a single hidden neuron is added and subsequently trained. This neuron is trained to maximise its *correlation* with the network loss, and once this has been done sufficiently, another neuron is added and the process continues.

This process does not perfectly construct a network for every problem however, because there are still many parameters such as the number of epochs and batch size that have to be chosen, and there is no guarantee that the network will not for instance add too many neurons and create a network that is too complex for the problem. Instead of manually choosing hyper-parameters, we could use a search-based algorithm to find parameters that optimize the performance of the cascade neural network that is constructed.

Genetic algorithms are inspired by the process of natural selection, where an initial population is generated and the fittest individuals in the population are 'mated' with one another, and sometimes mutated to create a new population of even fitter individuals. This process is repeated a number of times and the fittest individual over all populations is taken. We can use a genetic algorithm to optimize our constructive cascade algorithm, where we generate a population of constructive cascade algorithms and evaluate them by their training loss when predicting the dataset. After some number of generations, we can take the neural network (or better, the hyper-parameters that were used to construct it), and use this as our final result.

In [2], the authors construct a cascade algorithm that they use for a face recognition task. They present two algorithmss: Local Feature Constructive Cascade (LoCC), and Symmetry Local Feature Constructive Cascade (SymLoCC). The LoCC algorithm is a cascade algorithm that attempts to extract local features from a 32 x 32 pixel image by having hidden neurons take input from overlapping grids of pixels rather than starting from a completely connected network. By making the receptive fields for each hidden neuron only a square section of the input, smaller features (such as the nose, mouth, eyes) can be learned by different neurons and then combined to obtain an output, rather than each neuron seeing the entire image each time. The cascade layers in this algorithm also only take input from subsets of the original input and from the output of the initial hidden layer.

The SymLoCC algorithm in [2] is a modified version of the LoCC algorithm that sets all weights on the right half of the input grid as mirrored weights from the left half of the input grid. The entire network from input to

* Supported by the Australian National University

the hidden layer is thus mirrored, and only half of the weights from the input layer can be set. The reason for this as stated in [2] is "because human faces possess symmetric characteristcs", thus an image of a face will be roughly symmetric. From this, the network should preserve its generalisation properties with this restriction, and the classification can be done more efficiently.

This paper takes the notion that building *a priori* knowledge of a dataset into a neural network can improve generalisation performance, and I implement a simple constructive cascade algorithm to build a network on brainwave data that was taken when subjects were exposed to different music. This was done in three stages, where I first test a simple neural network, then a simple cascade network, and finally I modify the cascade network to run much faster and also to focus the network on more significant features of the dataset. After this, a genetic algorithm was used to optimize hyper-parameters for the final cascade network, and the results of the genetic algorithm were taken. Finally, the hyper-parameters chosen by the genetic algorithm were used to generate the final cascade network and the results were taken.

2 Method

2.1 Approaches

The three approaches to examine the dataset were a simple neural network, a simple cascade algorithm, and a 'selective' cascade algorithm.

The simple neural network included a network with one hidden layer which was completely connected to both the input and output.

A cascade algorithm was applied to the original neural network, which would create a hidden neuron that was fully connected to both the input and output, along with the output of all hidden neurons added in this way.

The final architecture takes into account the information given by [4] on the frequency of each feature that was observed in the experiment to select the data. The most frequent features were 'highlighted' in this architecture, as each neuron added through the cascading algorithm only took input from the features mentioned in the data set paper [4], along with all previously added cascade neurons. The purpose of this was to force the network to train on these features more than other features, since these features are expected to be correlated with the classification more than others.

After implementing the selective cascade network, a genetic algorithm was developed where individuals in the algorithm were a list of the hyper-parameters required for the selective cascade network. These hyper-parameters were the number of hidden neurons in the original network, the number of epochs used for training, the batch size of each epoch and the learning rate used for training.

2.2 Data set

The data set that was used in this paper was a section of the data set provided by [4]. The original data was collected in an electroencephalogram (EEG) test which recorded brainwave activity in subjects as they were exposed to different genres of music. The data used in this paper was the data taken from the F7 channel in the experiment, which is a channel located on the Frontal Lobe of the brain. Features of this raw data like the mean and max. frequency are included. In total there are 25 features for the F7 channel included in our data set.

In [4], they use feature selection methods to determine the best features for extraction in order to classify the data. They list the 25 most frequent features selected by their feature selection methods, and of these 25 features, 5 of them are from the F7 channel. These features are the ones that are used for the receptive fields of the cascading neurons in the selective cascade algorithm described above. Note that one of the 5 features listed was not included in the data set used for this paper, so each cascading neuron has a receptive field of 4 input features (plus the output of all previous cascade neurons).

2.3 Training

All weights in all of the architectures were initialised uniformly randomly over the interval (-1,1). The activation function used in all networks was the sigmoid activation function. The Adam optimization algorithm [3] was used to train the weights of all 3 networks. All networks were trained in batches of 10 over 800 epochs, with an initial hidden layer of 50 hidden neurons and a learning rate of 0.01.

For the cascade algorithms, every 10 epochs a `checking` parameter was set, making every forward pass record its loss until the 4 most recent recorded losses had a standard deviation of less than 0.05, in which case a new neuron

was added and `checking` was set to `False`. This meant that the network would only add neurons when convergence was met in terms of loss minimising during training, so as to not over complicate the network. The simple network was tested with 50, 100 and 150 hidden neurons to see if this made a significant difference on the result. The final architecture was tested 30 times, and each test the number of cascading neurons created was recorded to see if there was and relationship between the final complexity of the network and the testing accuracy.

The genetic algorithm used was based off an implementation of a genetic algorithm for hyper-parameter optimization [5], and using the DEAP library functions. The population size used was 20, and number of generations was set to 10, due to the large computation times required for higher populations/generations. Each individual in the population was randomly generated using the following bounds for each hyper-parameter:

- Size of initial hidden layer: 5-50
- Number of epochs: 50-300
- Batch size of epoch: 5-50
- Learning rate: 0.01 - 0.1

The crossover method used was one point crossover, with a 0.7 probability of crossover and tournament selection with tournament size equal to 2. The mutation probability was 0.01, and mutation was done by simply randomly generating one of the hyper-parameters in the individual. Cross validation was done by randomly splitting the dataset into 70% training data, 15% testing data and the remaining 15% as a validation set that was used at the end of the genetic algorithm in order to verify if the hyper-parameters found by the genetic algorithm produced a model with high accuracy. The genetic algorithm was run 20 times to achieve a set of optimal parameters, and those parameters were tested on the validation set 20 times, and the accuracy in predicting the validation set was averaged over these 20 times.

3 Results and Discussion

Table 1. shows the testing accuracy results of each architecture over 20 trials, as well as the validation accuracy for 20 trails of the genetic algorithm. The table shows that the mean of every architecture lies within 1% of 42%, which is only marginally better than random chance (there are 3 classes). The genetic algorithm arrived at an initial hidden size of 36 neurons, 56 epochs, a batch size of 48 and a learning rate of 0.0234 (3 d.p.), but was only able to achieve 43% validation accuracy. This very low testing accuracy can most likely be attributed to the data set, as it is only contains the data from one of the fourteen channels where data was collected from in the original data set.

Table 1. Testing Accuracy of Each Network

	Network					
	Simple NN 50	Simple NN 100	Simple NN 150	Basic Cascade	Selective Cascade	Optimised Selective Cascade
	41.38	54.17	40	49.57	51.3	42.35
	37.41	37.84	41.41	46.67	38.71	44.71
	39.32	41.12	35.59	42.98	37.9	47.06
	35.04	37.82	39.69	37.61	43.24	38.82
	44.63	33.93	45.6	38.98	33.96	40.00
	40.91	45.87	44.34	44.96	47.01	49.41
	42.99	42	42.61	44.55	39.13	45.88
	44.55	43.8	43.36	41.23	46.36	36.47
	43.97	41.13	46.27	45.67	42.24	37.65
Testing Accuracy %	35	36.44	38.68	43.14	43.4	44.71
	41.18	42.5	36.21	34.95	42.24	50.59
	39.25	43.97	38.52	39.5	37.4	42.35
	42.62	38.35	42.34	40	37.14	43.53
	46.4	45.61	39.29	42.2	45.54	44.71
	43.69	44.63	44.04	44.96	50	41.18
	40	42.2	41.03	40.71	37.82	43.53
	44.95	42.75	42.28	41.13	29.91	41.18
	45.19	44	41.07	44.72	44.09	44.71
	44.35	44.35	43.7	39.5	45.13	40.00
	37.1	38.46	47.32	37.27	44.92	41.18
Mean	41.497	42.047	41.668	42.015	41.872	43.00
S.D.	3.431	4.345	3.152	3.600	5.341	3.65
Range	11.40	20.24	11.73	14.62	21.39	14.12

The range and standard deviation of the simple neural network with 100 neurons was much higher than its counterparts, which may be significant, as this architecture also achieved the highest testing accuracy of 54.17%.

Selective cascade had the highest variance in its results, with a standard deviation of 5.341%, compared to the standard deviation of 3.6% for the basic cascade. Such deviation was suspected to be strongly correlated to the number of neurons created during cascading. Because of this suspicion, more data was collected and the testing accuracy was plotted against the number of cascade neurons created for 35 executions of the selective cascade algorithm, as seen in Fig. 1.

The data gives a correlation of -0.2, meaning that there is a loose negative correlation between number of cascade neurons added and testing accuracy. The correlation coefficient of 0.2 tells us however that this relationship is not very strong, and may only apply in the range where enough data was collected.

The genetic algorithm found that a much smaller number of epochs, larger batch size and larger learning rate yielded better results for a selective cascade. The genetically optimised selective cascade also found a much smaller variance than the original selective cascade, so these parameters are preferred as they return more reliable results. These parameters still only provide with a guess that is marginally better than a random one, which further points towards a problematic dataset.

Although the results are almost uniform, the selective cascade architecture used may have significant improvements on a more classifiable dataset. The testing in [2] shows that by reducing the receptive fields of cascading neurons (so they aren't fully connected) can still facilitate a working neural network, while being less resource intensive.

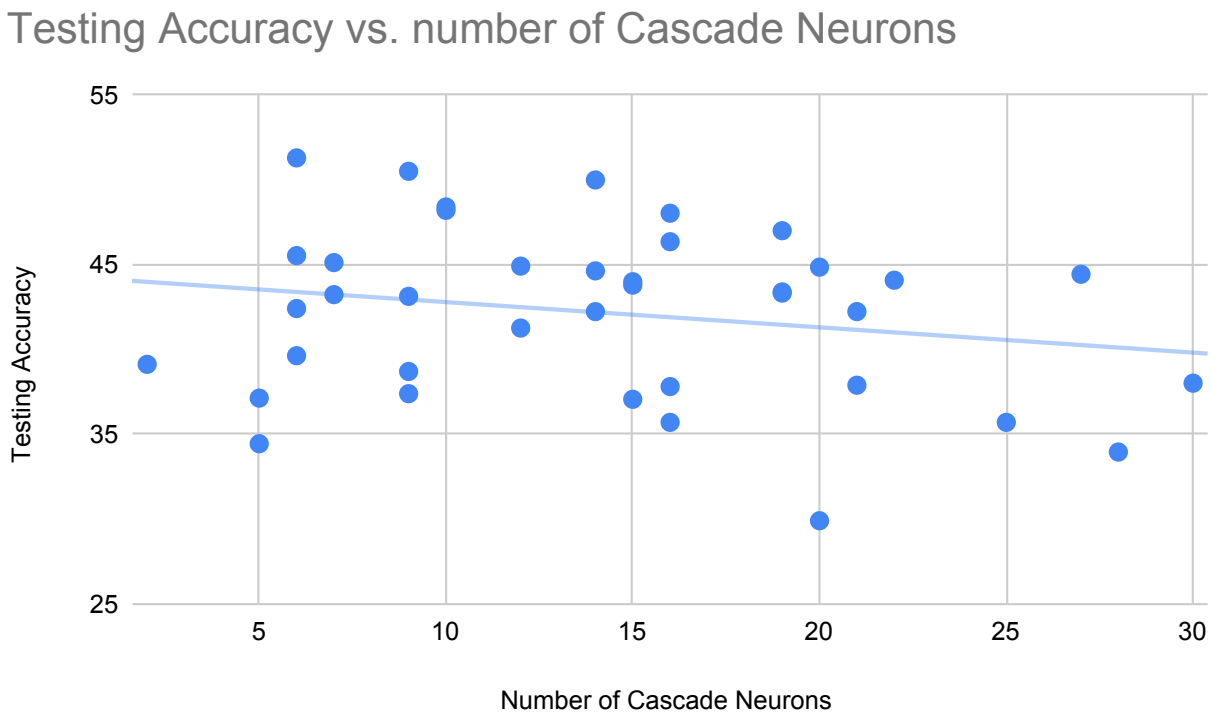

Fig. 1. Data points for testing accuracy by the number of cascade neurons created during training, including a line of best fit generated by google sheets

4 Conclusion and Future Work

This data set is most likely not fit for a classification problem, and because of this not many conclusions can be drawn. The experimental analysis of this dataset, however, provided some insight into the types of evaluations that can be taken out on constructive cascade algorithms. As cascade algorithms partially create the topology of a neural network, different topologies could be generated and tested similar to what was done in this study. A more appropriate data set (not an incomplete data set) could yield testing grounds for an automatic topology generator that could generate topologies that optimize training for a particular data set. Much how in [2] they create a symmetrical network topology, different shapes/parameters could be tested in constructive cascade algorithms and analysis could be done on the effect that a given topology has on testing accuracy (like testing the affect of the number of cascades completed vs the testing accuracy). The genetic algorithm used here could be applied to fine tune different models of cascade architectures to optimise for a given problem.

In future, different data sets will be used and more extensive data collection should be done, as in this project much more time was spent on coding the network than testing it.

References

1. Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE (1990)
2. Khoo, S., Gedeon, T.: Generalisation performance vs. architecture variations in constructive cascade networks. In: International Conference on Neural Information Processing. pp. 236–243. Springer (2008)
3. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
4. 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. IEEE (2020)
5. Rothwell, C.: Hyper-parameter optimisation using genetic algorithms: Classification task (Oct 2018), https://www.linkedin.com/pulse/hyper-parameter-optimisation-using-genetic-algorithms-conor-rothwell