The Applicability of Pruning by Distinctiveness on Different Artificial Neural Networks

A Examination of Pruning by Distinctiveness on Multilayer Perceptron and Recurrent Neural Networks on Brainwave-Music Effect Data

Yichen Wang[0000−0001−9887−0247]

Research School of Computer Science
The Australian National University
Canberra, Australia
u6345072@anu.edu.au

Abstract. Network pruning - the task of removing redundant or less important hidden neurons - has been demonstrated beneficial in trained neural networks, including simple multilayer perceptrons and deep neural networks, for various reasons: less training time, better generalisation performance, or realise some specific task such as image compression, natural language processing, etc. To date, various pruning techniques have been proposed and demonstrated effective in reducing neural networks size but maintaining similar networks' performances. However, studies also show that pruning techniques' performances may vary based on certain neural network architectures or datasets. Pruning by distinctiveness is one of which falls into such category: this technique has presented promising pruning results in several multilayer perceptrons models in image compression applications. Yet few works were attempted on applying such technique into other neural network architectures. That being said, a more critical analysis regarding the applicability of this pruning technique requires further investigation. In this work, we attempt to address this issue by examining pruning by distinctness's applicability on different neural network architectures. We first train a simple multilayer perceptron and recurrent neural networks on a brainwave-music effect classification problem. We then apply pruning by distinctness on these neural network models and observe their pruning performances. We also additionally observe hidden neurons distinctness determined by this pruning technique in relation to our models' performance. Our results show that pruning by distinctness provides a stable and effective performance on the multilayer perceptron neural network, yet the recurrent neural network falls short of such. We conclude that pruning techniques do vary based on neural network architectures, and pruning by distinctness is more applicable to simple multilayer perceptrons to our best knowledge.

Keywords: Neural Network · Network Reduction · Electroencephalogram · Affective Neuronscience · Music Therapy · Recurrent Neural Network · Time-series Data

1 Introduction

Artificial Neural Networks (ANNs) are computational models inspired by human brains neurons system to process information[12]. An ANN normally consists of a large number of artificial neurons which are interconnected by direct links and which cooperate to perform a desired task using mathematical or computational model. During the process, the ANN learns certain patterns from input data and such result can be utilised for other practices. Each neuron in the network extracts specific features based on the content it receives and the position in the network. One of the classical ANNs is multilayer perceptrons (MLPs), also known as "vanilla" ANNs, consisting of at least three layers: an input, a hidden and an output layer, feed-forwarding information from the input for certain computations.

To date, ANNs have evolved with different architectures for solving various tasks, ranging from classification problem, natural language processing to image pattern recognition[1]. In particular, recurrent neural networks (RNNs) is one kind of ANNs that can model sequences of arbitrary lengths of input and output data.

A RNN's architecture is similar to the MLP including hidden layers, feed-forward computation, but with an additional feedback connection linking from output to input. By this way, the RNN can allow the current state of the network (i.e hidden neurons' weight or activation parameters) influence its future state. This is of great help for applications dealing with timeseries or natural language data, which require the previous state information for the computation of current state[7].

However, although ANNs have been widely applied across different fields, literature has noted that many of these networks, particularly those with good performance[6], require time-consuming computation and significantly rely on hardware's computing capability. These requirements result in increase infrastructure costs and restriction on performing ANNs on mobile or small devices.

One popular approach to solving these issues is network pruning, which removes redundant or less important hidden neurons to reduce the network size and computation cost[2]. In particular, pruning by distinctness proposed by Gedeon and Harris has been used in detecting this kind of neurons[5]. This is achieved by using hidden neurons output activation vectors over the input pattern space as hidden neurons distinctness to *represent their functionalities*. The application of this technique has shown its effectiveness across several works in MLPs[3,4]. For example, an image quality compression task can be achieved by removing redundant neurons and maintaining a desired level of image quality. Such promising pruning results have revealed the possibility

of its application into other ANNs. Yet few practices were done to validate such proposition. Furthermore, as addressed in [2], pruning techniques' performances may vary based on certain neural network architectures or datasets. That being said, thorough work is required in empirically validating the effectiveness of pruning by distinctness on different ANNs.

In this paper, we attempt to investigate the applicability of pruning by distinctness on various network architectures based on a brainwave-music effect classification problem. We specifically select this dataset given that the original work deals with time-series electroencephalograms(EEGs) data, and has an outstanding performance using a rather complicated MLP. We desire a more compact solution to this that could achieve similar performance. We also want to explore an alternative recurrent neural network architecture as well as pruning in this classification problem, to examine pruning by distinctness applicability across different various neural network architectures.

2 Method

There are four distinct parts to our method: data production/acquisition and processing; implementation of ANNs architecture, including simple multilayer perceptron and recurrent neural networks, and training; network pruning by distinctness, as well as the evaluation for our models. We discuss details as follows.

2.1 Data and processing

Data production and acquisition The dataset described in [10] was used in this work, which was provided by its author. The provided dataset, including raw time-series EEGs signals and extracted features, is a subset of the original EEGs brainwave signals collected from 24 participants using all 14 channels of the related headset equipment. The raw EEGs signals were collected at the sampling rate of 128 Hz while playing a total eight out of twelve selected music pieces to participants.

For the provided raw time-series EEGs signals, they are grouped by song numbers. Within each group, it contains time-series EEGs signals from any 16 of 24 participants who listened to one of the 12 songs, all from F7 channel of the headset. Each song was about 4-minute length and the sampling rate is 8 Hz. That being said, there are 8 signal points for each second and each sequence has a length of approximate 2000-3000 data points. The total number of sequences in each group is 16 * 12 = 192, and each sequence is associated with a label. To best balance our training and testing datasets due to the impact of various lengths of time-series EGGs signals, we randomly selected 75% for training data and the left for testing, for each time-series length group.

For the provided extracted features, they are statistical features extracted by 26 linear and non-linear operations, which were originally from raw EEGs signals and has been preprocessed including a median smoothing filtering, band-pass-filtered, and then were segmented into the lengths of music pieces. The provided dataset has 576 instances in total with 25 features and one label class. Each data instance is associated with a label,

Both provided raw EEGs data and extracted features shared the same property of labels, which are the categories of the song pieces that participants listened to. The label class contains three categories: classical, instrumental and pop, indicating the kind of song pieces participants listened to while collecting the original brainwave signals.

Furthermore, we can notice that both provided datasets are all from channel F7. This may due to the fact that data collected from this channel has been demonstrated as one of the most effective features channels to the classification task in the original work[10].

Data prepossessing For the provided raw time-series EEGs signals, inspections of the original dataset indicated that some default preprocessing might have gone through by device SDK. Given such, a median smoothing filtering was applied to smooth out noisy signals produced from the default preprocessing. This process also removed some outliers without loss of data points.

For the provided extracted features, we left them intact after the inspection of the original dataset showing that they have gone through necessary processing and additional feature extractions.

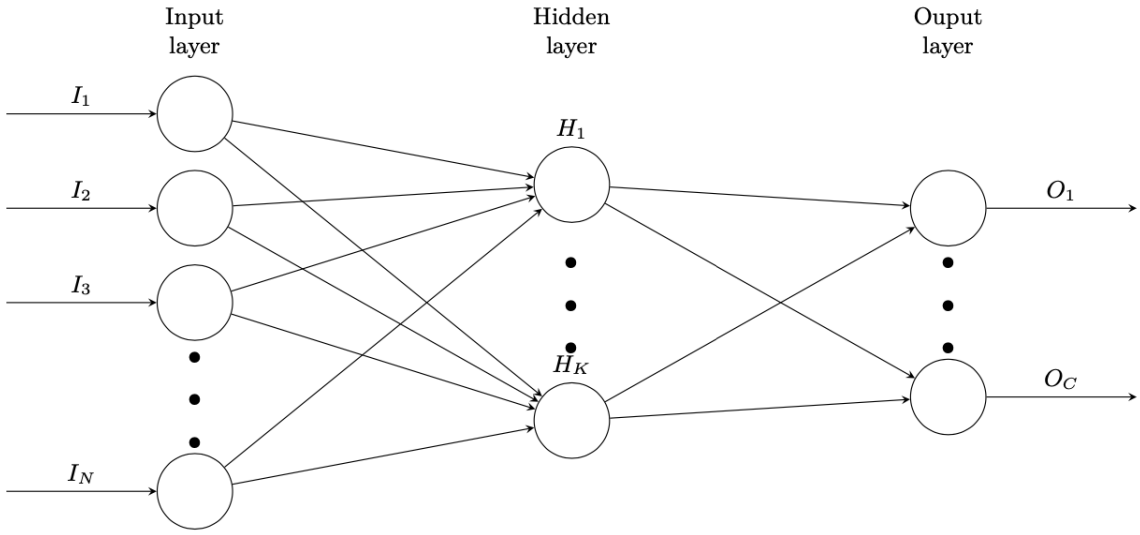
Both dataset were then preprocessed using the Min-Max normalisation technique to reduce the impacts of over-fitting issues to certain features before feeding into our ANN models for training. This is done by shifting each feature into the range of minimum 0 to maximum 1[9]. In addition, the output class in time-series EEGs signals dataset was encoded to one-hot for the classification task.

2.2 Neural Network Architecture

In our work, we selected two different ANNs architectures: a multilayer perceptron, and a reccurent neural network, which fit our datasets as well as goals. We given details as follows.

Multilayer Perceptron Neural Network (MLP) The choice of the MLP in this paper was mainly based on that used in [10] due to the high reported classification accuracy. A standard two layer fully connected feed-forward ANN was constructed (Figure 1), containing one hidden layer consisting of 30 units with a Sigmoid activation function and an output layer consisting of 3 units with a Softmax activation function, each corresponding to a category of music pieces (classical, instrumental and pop).

Fig. 1: An illustration of the Neural Network architecture in this work: 25 input features (N = 25) with 30 hidden neurons (K = 30) to calssify three music categories (C =3). Each input was fed into the hidden layer with Sigmoid activation function, and then fed into the output layer with Softmax activation function.



Recurrent Neural Network(RNN) An alternative recurrent neural network was also constructed for the original work which we can improve upon. We specifically selected many-to-one RNN model (Figure 2), which suits our classification problem that having participants' time-series EEGs signals sequences as input and the matched song categories as the output.

The RNN was designed with one hidden layer contains 50 units and use tanh activation function to achieve nonlinearity. The output layer is a vector with size of 3 * 1 for each pattern and uses Softmax activation function to predict the classification results.

Training MLP and RNN Both models were trained using back-propagation algorithm with the Cross-entropy loss as loss function and Adam optimisation algorithm for optimiser. Furthermore, mini-batch training was utilised in order to address the potential of slow convergence that stuck in local minima, given the problem is expected to not be fully convex in nature. All the implementation was done in Pytorch.

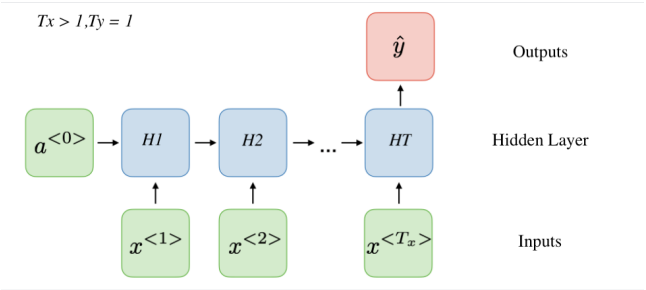


Fig. 2: An illustration of a many-to-one RNN in this work: X refers to the total numbers of input EEGs fed in to the model. The hidden layer contains 50 hidden states. The output contains a vector with the size 3*1 encoding the label class. T refers to timesteps of a given timeseries EGGs signal.

Image Source

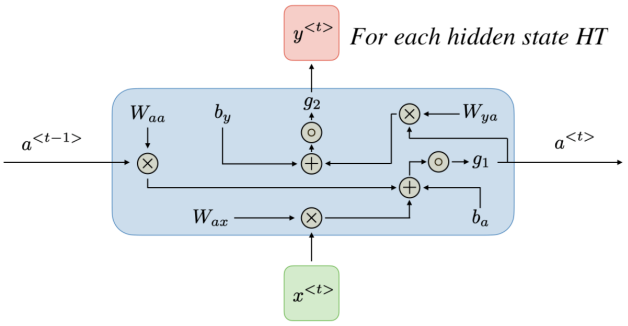


Fig. 3: Details of a hidden state in the hidden layer of a RNN model: for each timestep t, the activation $a^{<t>} = g_1(W_{aa}a^{<t>} + W_{ax}X^{<t>} + b_a)$ and $y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$ where $W_{aa}a^{<t>}$, $W_{ax}X^{<t>}$,$W_{ya}a^{<t>}$ are weights and b_a , b_y are bias.

Image Source

2.3 Network Reduction: Pruning by Distinctiveness

Pruning by distinctness[5] network reduction technique was then applied to explore its pruning performance on different trained models. We specifically selected this technique given its effectiveness on several MLPs tasks. We hoped to further investigate its applicability on other ANNs such as RNNs. We describe the original principle of this technique and then discuss how this method was utilised on the RNN in this work.

In this technique, hidden neurons' distinctiveness is determined from their output activation vectors over the input pattern space to represent hidden neurons' functionalities. Neurons that share similar distinctness are recognised as similar units, and one of them can be removed without reducing the model's performance. In other words, we calculate the angle of the hidden neurons output activation vector over input space. The angular separations less than 15° can be seen effectively similar, and one can be removed. For situation where more than two units performing similar functionalities (angular separations up to about 15°) in our model, n-1 units can be removed and add their weight values to the remaining one. The weight of the removed neurons should be added to the remaining ones, indicating its weighted effectiveness among other hidden neurons. Noted, the vector angle are normalised to the range -0.5 to 0.5 so that the angular separation ranges from 0°to 180°rather than 0°to 90°.

In addition, for the RNN model, such pruning method was alternatively performed by removing redundant weight vectors in the weight matrix based on output activation vectors between each hidden state in the hidden layer, rather than **actually reducing network size**. That being said, weight vectors from weight matrix between input and the hidden state, as well as between hidden states, which were identified as less important or redundant in this method, were removed. The reason for such is due to the complicated structure in RNNs hidden states. As illustrated in Figure 3: the computation of a hidden state is not only associated with the weights W_{ax} and bias b_a between input and hidden states, but also with the weights W_{aa} between hidden states (among previous and current hidden states). That being said, when a weight matrix dimension is reduced, other associated parameters dimensions, including weights and bias, are also required to be reduced accordingly. This also applies to output vectors between hidden layers in two and more hidden layers RNNs. Furthermore, the hidden state should not be removed in any sense, as doing so will violate the RNNs functionality.

2.4 Evaluation

For evaluation, the method proposed in [2] was used, including model's classification performance and efficiency (the model's classification execution time), to evaluate our pruning technique's performance on different models. Four additional measures were also used to support our analysis in models performance, as addressed in [11] that accuracy does not always provide complete information of the model performance. These measures are:

- Precision (Fraction of the predicted labels matched)
- Recall/Sensitivity (True Positive Rate)
- Specificity (True Negative Rate)
- F-measure (Harmonic mean of Precision and Recall)

Furthermore, we also observed the relationship between hidden neurons distinctness and its functionalities, which may contribute to our models' performance, mainly following the process in [3] and [5]. This was done by reporting on the minimum angle between remaining hidden neurons while pruning every two neurons along with the model's training losses.

Due to the nature of having multiple data instances from each participant, a random split for training and test datasets was not viable as bias will be introduced if a participant's data was in both the training and test sets. As such, the k-fold cross-validation was employed during models training, where the data was divided into k pieces to train k models of the same type, each using a different piece of data as validation to the trained model's performance.

That being said, the evaluation process of our work was formalised to: First, we reported the classification accuracy and losses of the unpruned model in classifying the categories of music pieces. We then pruned our MLP and RNN model, and reported on the minimum angle between remaining hidden neurons while pruning every two neurons along with the losses and efficiency. We then compared pruning performance based on our models' performance.

3 Results and Discussion

Overall, our unpruned classification models maintained a stable test accuracy around 40% and 50% on MLP and RNN respectively, as shown in Table 1. Pruning by distinctness was effective at the beginning with improved accuracy around 44.17 % and 52.08% on MLP and RNN respectively. However, further pruning on our models performed worse than the unpruned one shown in Table 2 and 3. Furthermore, pruning by distinctness on RNN model presented questioning evaluation results, although it was shown effective solely based on classification accuracy. Last but not least, we were not able to reproduce the results described in [10] with a more compact model in any way.

3.1 Baseline

Different hyper-parameters were explored in this work to establish a stable performance baseline for our MLP and RNN classification models. We also used this baseline as a point of comparison to observe the pruning performances on these two models. Furthermore, we recorded the execution time on testing our models, which were used to examine the pruning technique's applicability.

MLP For MLP, we started by trying to reproduce the results described in [10] as the baseline for our later work. [10] reported that a 97.5% classification accuracy was achieved using a simple two layer neural network of 30 hidden neurons in the hidden layer. The Levenberg—Marquardt method and mean squared normalised error (MSE) were used as performance function.

In our model, we altered to back-propagation algorithm and Cross-entropy as performance function given our aim here is to examine hidden neurons functionalities based on its *distinctness*. We constructed a similar neural network model with 30 hidden neurons and Sigmoid activation. Initial attempt with learning rates of 0.01, and a max epoch of 1000 gave a very poor result of training accuracy 62.72 % and testing accuracy 33%. This indicate our model performs no better than random guessing as the 3 classes in the dataset are balanced. The high reported training accuracy also suggests that our model is over-fitting to the training dataset after observing the confusion matrix and average losses.

As such, we applied the mini-batch method with 10-fold cross-validation to prevent over-fitting issue as well as the concern of our comparatively small dataset sample. We then trained our model in batch sizes 16, 32 and 64 with 0.01 learning rates and 10-fold cross-validation respectively. We also increased neurons sizes and explored learning rates 0.001, 0.01 and 0.1 respectively.

The final architecture of our model in reporting results used 30 hidden neurons with a learning rate of 0.01. This model was trained for 1500 epochs with 32 batch sizes to obtain our baseline results. Overall our model test accuracy maintained stable around 40% using 1.034 ms to execute. The baseline model's final test results are illustrated in Table 1.

Table 1: Our baseline model's final test performance

Model		(Average) Loss	(Average) Accuracy	Execution Time (ms)
MLP	10-fold training	0.861	51.99%	Not applicable
	10-fold validation	1.136	41.25%	Not applicable
	Final model training (baseline)	0.935	60.46%	Not applicable
	Final model test (baseline)	1.143	40.00 %	1.034
RNN	5-fold training	1.0409	53.47%	Not applicable
	5-fold validation	1.0059	49.20%	Not applicable
	Final model training (baseline)	0.969	51.58%	Not applicable
	Final model test (baseline)	0.870	50.00%	108.01

RNN For RNN, we constructed one hidden layers contains 50 units and use tanh activation function to achieve nonlinearity. Different hyperparameters were explored in the model with learning rates 0.0001, 0.003, 0.005, max epochs 100, 200, 500, batch sizes 16 and 32 respectively using 5-fold cross-validation.

The final architecture of our model in reporting results was trained using 5-fold cross-validation with learning rate 0.003, batch size 32 and a max epoch of 200, which gave a very steady performance with training accuracy 53.47% and validation accuracy 49.20%. The final model training accuracy was 51.58% and testing accuracy 50 %, using 108.01 ms to execute. Examining the confusion matrix and average losses, it was found that the mode was accurate at predicting 2 out of 3 label classes, with rare correct classification on the third label. It was also found that some average training accuracies were subtlety higher than validation accuracy in some folds, although data was randomly shuffled before training and testing. This may suggest that the model was sensitive to certain lengths of time-series EGGs signal in making classification.

3.2 Network Reduction

Overall, pruning by distinctness was found to have different behaviours on MLP and RNN models. For the MLP, pruning was effective while removed the first two neurons. The model scored a relatively high performance compared to the unpruned network. Yet further pruning provided worse performance. For the RNN, pruning has shown effective in removing some of weight vectors while pruning the first four. Although pruning performances vary based on the specific weight vectors that were removed. We give pruning details for each model as follows.

MLP To prune our MLP model, we computed vectors angles among all hidden neurons weights and manually labelled neurons whose angular separations up to about 15°. We then ranked neurons based on the number of other similar neurons identified by our technique in descending order. We then pruned every two neurons based on the ranking. For the neurons with more than similar neurons, we selected the one with the smallest angular separations, which indicate these two neurons vectors are very most close in the input pattern space compared to other identified similar neurons.

Table 2 and Figure 4 illustrated our model classification performance during the pruning. Overall, network pruning by distinctness was effective in our model while pruning the first two neurons. The model scored a relatively high performance compared to the unpruned network with an accuracy 44.17%. In contrast, we got worse performance when continuing the pruning process. This may due to further pruned neurons have complicated relationships with other neurons. Removing such results in the loss of important neurons, which

incorporate important information or features to our model. What's more, we can see that the execution time increased while pruning the first two neurons, and then decreased subsequently (Fig 6).

Table 2: Table showing various evaluation metrics that demonstrate our pruned model performance. Loss is calculated using the Cross Entropy loss function. In the pruned neuron pairs, the one after the arrow refers to the pruned one, while the one before the arrow refers to the remained one.

Pruned Neurons Numbers	Pruned Neuron Pair	Loss	Accuracy	Precision	Recall	F1 Score	Execution Time (ms)
0 (baseline)	Not applicable	1.143	40.00 %	0.318	0.344	0.304	1.034
2	(5->12) (6->12)	1.105	44.17 %	0.310	0.359	0.330	1.199
4	(2->7)(12->18)	1.134	40.83 %	0.335	0.357	0.333	1.073
6	(3->27)(14->25)	1.136	39.17 %	0.329	0.348	0.338	1.059

We then visualised the pruned model's performance versus minimum angle among remaining hidden neurons to observe the relationship between hidden neurons' distinctness and their functionalities. As illustrated in Figure 5, we can see that the minimum angle among remaining neurons is coarsely indicating our hidden neurons behaviours, particularly while pruning the first two neurons. Further pruning every two neurons lead to an increasing loss in our model.

Fig. 4: Bar plots showing our pruned model performance by pruning every two neurons including accuracy, precision, recall and f1 score.

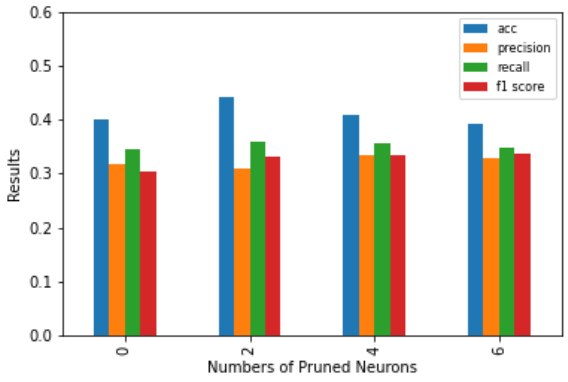


Fig. 5: Line plots showing minimal angle between remained hidden neurons versus pruned models performance.

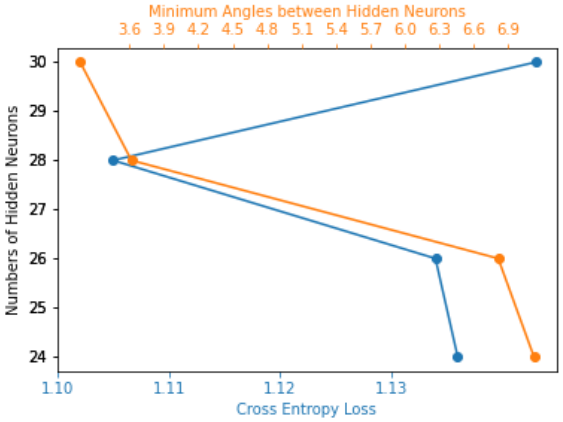


Fig. 6: The change of execution time on MLP: matching Fig 6, we can see that improved performance led to more execution time, and less execution time with decreased performance.

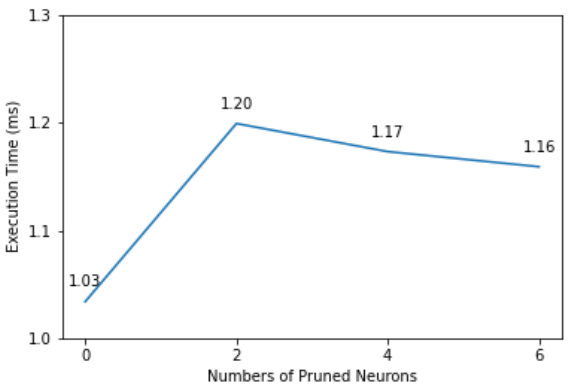
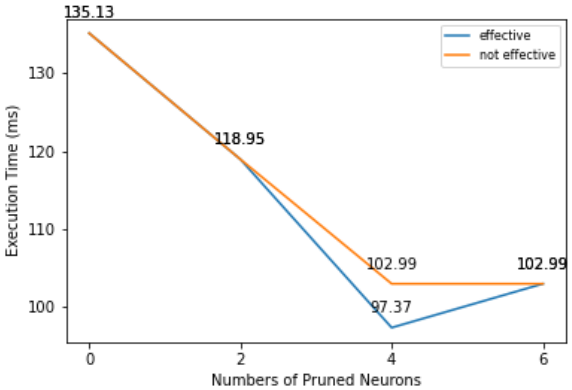


Fig. 7: The change of execution time on RNN. The effective line refers to pruning four weight vectors pairs which retained decent classification accuracy, not effective line refers to prune weight vectors pairs with dismay performance.



RNN To pruning our RNN model, the same process of computing output activation vectors and angular separations in MLP pruning was followed. We then filtered the weight vectors indexes which angular separations up to 15° or no less 165 °. We ranked weight vectors based on angular separations in descending order, indicating they are identified with similar functionalities in this pruning method. We then removed these similar weight vectors progressively in the weights matrix on both input-hidden and hidden-hidden layers. During the pruning process, we found that only removing one weight vector, which was recognised as the most prominent in having least angular separations with lots other similar weight vectors, experienced a large drop in performance with accuracy 33%. Few attempts were further made to remove different prominent weight vectors, and encountered the similar situation described above. Given such, we alternated to a more gentle pruning process by removing weight vectors that were prominent in descending order, hoping to observe the pruning performance on our RNN model. In other words, we remove weight vectors from ones that have the least numbers of similar vectors identified by their distinctness.

Table 3 presents our RNN model performance while pruning every two weight vectors. We found that pruning different weight vectors, although they were identified similar by their distinctness, would lead to different model performances. As we can see that pruning by distinctness was effective while pruning the first two weight vectors, with improved classification accuracy 52.08% and less time for execution. However, we also found that the pruned model experienced a half drop in precision and F1 scores.

Table 3: Table showing our RNN performance while pruning every two weight vectors. It is interesting to note that we found pruning different weight vectors, although they are identified similar by their distinctness, led to different performance. The highlighted ones demonstrated such. We also provided minimum angle among the remaining weight vectors.

Num	Pruned Weight Vectors Index	Loss	Accuracy	Precision	Recall	F1 Score	Execution Time(ms)	Minimal Angle
0	None	0.87080	0.5000	0.5218	0.4329	0.3172	108.01	0.428
2	(40->44) (41->48)	1.03498	0.5208	0.2168	0.4528	0.2899	118.954	0.428
4	(39->46) (48->49)	1.04860	0.5000	0.2191	0.4375	0.2864	97.366	0.428
	(27->28)(43->49)	1.09330	0.3333	0.1111	0.3333	0.1667	102.993	0.428
6	Pair (36->42)(30->26) or Pair (4->6)(3->26)	1.09140	0.3333	0.1111	0.3333	0.1667	98.064/181.329	0.428

Continuing to prune the third and fourth weight vectors, we can see that removing weight vector 46 and 49 remained a decent classification performance with accuracy 50% and less execution time. While removing weight vector 28 and 49 instead, which only replaced one weight vector, introduced a huge drop in the model's performance and increased execution time, which was no better than random guessing. What's more, further pruning showed ineffective at all, regardless of different pruned weight vectors were explored based on their similar distinctness. However, we did notice that the execution time has decreased more compared to pruning the first weight vectors pair(Fig 7). Furthermore, it was also worth to note that the whole pruning process in our RNN model experienced a relatively large increase in loss, with more than 0.2 difference values compared to the unpruned one, as well as in precision and F1 scores. Even though the recall score didn't fluctuate much until the model's performance decreased to one same as random guessing.

In addition, we also recorded the minimum angle among the remaining weight vectors to examine whether such can imply RNN hidden state functionalities (Figre 3), following the same process in MLP. However, given the dismal model performances encountered in pruning prominent weight vectors, we failed to obtain any significant result. This is mainly because the minimum angle among remaining weight vectors remains the same if we do not remove prominent ones, while removing prominent ones would directly disrupt the model's performance in this specific task. That being said, there is a trade-off between model efficiency and quality in pruning.

3.3 Discussion and Observations

There are several interesting observations regarding our MLP and RNNs models performance:

First, we can see our model achieved a better performance while pruning the first two neurons on both ANNs architectures. For the MLP, the reason for such may come from the removal of neuron 6 and 12. As we can see from Table 2, the identified neurons 12 were associated with both neurons 5 and 6. Removing both of them and adding their weights to neuron 5 resulted in extracting important or features that were effective for our model. However, further pruning two neurons reported not significantly improved classification performance, although one of pruned neurons were identified as similar to neuron 12 by "distinctness". While for the RNN, the behaviours and reason that contributed to a better performance seem rather complicated. As mentioned above, the precision of the model in classification had a huge drop, along with an increase in loss. It was not surprising that loss increased when precision decreased, indicating that model was not precise in classifying this testing set, and experienced great losses. While what should be drawn attention to is the relatively high accuracy but low precision, indicating that the model performs well in this specific testing set, but may not perform well in other testing sets. This suggests that our pruned RNN model may not be generalised well.

Second, our experimental results revealed that pruning by distinctness performance varied based on different models: it appeared to be more stable in pruning MLP, which model's accuracy and loss were consistent: high accuracy less loss and vice versa. As for the RNN, our experiment revealed rather complicated behaviours in classification. The model experienced a huge drop in precision and an increase in the loss while pruning first weight vectors, although the classification accuracy improved with less execution time. As we have discussed above, this implies that our pruned RNN model may not be generalised well on other datasets. Further pruning on RNNs continued to show drops in precision. Furthermore, it is also worth noting that the pruning appears to affect the classification accuracy. We can see that our unpruned model's precision is 0.5218. In contrast, all pruned models' precision scores were all under 0.22, indicating the pruning may decrease model's precision on RNNs. This also further confirms that there is a trade-off between the model's quality and efficiency during pruning.

Third, we also observed the relationship between hidden neurons distinctness and their functionalities, which may contribute to our models' performance. Our result showed that neurons *distinctness* is not sufficient in indicating hidden neurons functionalities, and do not have a direct impact on the performance of MLP. Furthermore, our experiment also revealed that it was rather complicated and may not applicable for RNN in such case. For the MLP, we can found there was a huge drop of loss in our model while pruning the first two neurons illustrated in Figure 5, while the minimum angle between hidden neurons didn't have much change. This shows that our model was still able to achieve a better performance in pruning even though there were similar neurons existed over the input pattern space. Further reduction of these similar neurons conversely introduced more loss on our model without improved classification performance. For RNN, the minimum angle among remained weight vectors remained same as we failed to follow the process by starting removing most prominent weight vectors like we did on MLP. As our experiment showed that any removal of prominent weight vectors resulted gloomy classification accuracies, not better than random guessing. However, our experiment did provide insights showing that it was rather complicated to apply pruning by distinctness on RNNs. Reasons for such may due to RNNs' fundamental principles. As we have briefly addressed in subsection 2.2, RNNs are designed with hidden states that handling weights matrix both from input to hidden state, and between hidden states, so that to achieve certain computations. That being said, pruning on RNNs are not removing hidden states, but rather reducing the weights matrix. Such alternation may violate advantages presented in pruning by distinctness, which was originally used for a simple MLP. Furthermore, existing literature has posed the problem whether specific pruning method offers similar benefits across architectures, as well as whether the weights matrix after pruning are critical for obtaining the final efficient model[2,8].

Last but not least, we can notice that both models experienced an increase in execution time with improved performance, and less execution time with the cost of worse models performances in further pruning. This further confirms the trade-off between the model's efficiency and performance in pruning.

4 Future works

Based on our discussion, several works can be made for future work:

First, as discussed in Subsection 3.3, some unusual model performances after pruning are largely due to the removal of hidden neurons or weight vectors determined by distinctness, which were calculated over the input patterns. As such, it is important to have high quality as well as considerable amounts of input data so that our work could build upon in obtaining statistically valid result. However, as addressed in 2.1, datasets used in this work were all from one channel. A more integrated dataset would be desired in thoroughly addressing our mythologies of examining pruning by distinctness applicability.

Second, more factors may need to be considered in the removal of weight vectors on RNNs using pruning by distinctness. As we have addressed that operation on weights matrix may not be able to obtain effective results. That being said, an alternative of pruning by distinctness require further investigation if we want to apply such method on RNNs.

Third, other ANNs models such as conventional neural networks(CNNs) can be explored in investigating pruning by distinctness pruning performance. As mentioned in the discussion, the uniqueness of RNN's architecture that handles both input-hidden, hidden-hidden weights may not be suitable for pruning using this method. Classical CNNs (without recurrent connections) instead may produce alternative results.

5 Conclusion

In this work, we investigated the applicability of pruning by distinctness on various network architectures based on a brainwave-music effect classification problem. We constructed MLP and RNN models to classify three types of music based on human EEGs while listening to the music. We then pruned our model based on pruning by distinctness technique and observed the models' performance and efficiency. We obtain statistically insignificant results indicating that our pruned model could not have a compact solution with competent performance. Our results show that pruning by distinctness provided a stable and effective performance on the MLP, yet the recurrent neural network deems questioning. We conclude that pruning techniques do vary based on choices of neural network architectures, and pruning by distinctness is more applicable to simple multilayer perceptrons.

References

1. Abiodun, O.I., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, N.A., Arshad, H.: State-of-the-art in artificial neural network applications: A survey. Heliyon **4**(11), e00938 (2018)
2. Blalock, D., Ortiz, J.J.G., Frankle, J., Guttag, J.: What is the state of neural network pruning? arXiv preprint arXiv:2003.03033 (2020)
3. Gedeon, T.D.: 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. IEEE (1995)
4. Gedeon, T., Harris, D.: Creating robust networks. In: [Proceedings] 1991 IEEE International Joint Conference on Neural Networks. pp. 2553–2557. IEEE (1991)
5. Gedeon, T., Harris, D.: Network reduction techniques. In: Proceedings International Conference on Neural Networks Methodologies and Applications. vol. 1, pp. 119–126 (1991)
6. Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, M.X., Chen, D., Lee, H., Ngiam, J., Le, Q.V., Wu, Y., et al.: Gpipe: Efficient training of giant neural networks using pipeline parallelism. arXiv preprint arXiv:1811.06965 (2018)
7. Hüsken, M., Stagge, P.: Recurrent neural networks for time series classification. Neurocomputing **50**, 223–235 (2003)
8. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270 (2018)
9. Patro, S., Sahu, K.K.: Normalization: A preprocessing stage. arXiv preprint arXiv:1503.06462 (2015)
10. 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)
11. Valverde-Albacete, F.J., Peláez-Moreno, C.: 100% classification accuracy considered harmful: The normalized information transfer factor explains the accuracy paradox. PloS one **9**(1), e84217 (2014)
12. Wang, S.C.: Artificial neural network. In: Interdisciplinary computing in java programming, pp. 81–100. Springer (2003)