Optimizing Hyperparameters for Bidirectional Neural Networks using Evolutionary Algorithms

Septian Razi [1]

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

Abstract. A novel application of a Bidirectional Neural Network (BDNN) on Eye Gaze data on Manipulated and Unmanipulated Images was developed. As BDNN's possess a highly complex hyperparameter space, we propose the use of an evolutionary algorithm to search for the optimal hyperparameter space. We develop an EA to do as such, employing two different fitness functions *Accuracy* and *Accuracy with Compression Penalty.* We found that the optimal hyperparameters of the two were different in significant ways, especially in terms of bidirectional training switch occurrence and autoencoder hidden neurons. Applying these optimal hyperparameters, our model achieves an accuracy of 73%.

Keywords: Bidirectional Neural Network, Auto-Encoders, Eye Gaze, Image Manipulation, Hyperparameter Search, Evolutionary Algorithm, Fitness

1 Introduction

1.1 Motivation

We live in a world populated with billions of images. But with the prevalence of image editing software, manipulation of these images has never been easier and increasingly common. There is a burgeoning research in forensics and computer science that is uncovering best practices to determining whether an image has been manipulated, yet there is a lack of research on how we humans perceive, or fail to perceive, these fraudulent images [6]. Using eye gaze data from an image manipulation prediction experiment, combined with a novel technique of bidirectional neural network, we hope to gain additional insights on the use of this technology in this problem domain.

1.2 Related Work

Previous work has been conducted on the development of neural networks for eye gaze data [6][5]. Caldwell et al. achieves an accuracy of 65.7% [6], whereas Tan et al. achieves an accuracy of 67.82% on their neural network by implementing a network reduction technique coupled with an evolutionary algorithm [5].

The idea of a bidirectional neural network (BDNN) is simple; allow the network to be fed and trained traditionally (input to output and error backpropagated) and in its reverse order (output to input and error 'forward-propagated'). Training a neural network in this manner offers benefits from further enforcing inversibility of a function and reduction in free parameters [3]. It has seen many applications from image compression [4], inverse functions, and temporal data [7].

There has also been extensive work on the use of evolutionary algorithms for hyperparameter optimization [8][9][11]. Young employs a bespoke evolutionary algorithm to optimize deep neural networks [8]. In contrast, Cantu-Paz [9] and Martinez [11] apply an evolutionary algorithm to shallow neural networks and find that neural networks generated in this manner can be found in less time and generally perform better than their manually/grid searched counterparts. In essence, evolutionary algorithms have been shown to be an effective solution at finding optimal hyperparameters for neural networks.

Yet, so far there has been limited research on the use of EA's in combination with shallow BDNN's. BDNN's in this paper are in a unique position in that there are two neural networks operating – a bidirectional autoencoder and a classifier. Searching the hyperparameter space for one neural network using conventional techniques already consumes a significant amount of computational resources [10], therefore two neural networks would likely require more. Thus, the use of an EA with BDNN is promising, and this paper seeks to fill this void in research.

1.3 Previous Experiment

In a previous iteration of this paper, the author experimented with the use of BDNNs trained with the Eye Gaze on Manipulated Images dataset [1] in creating Class Prototypes for our problem domain. Our neural network trained in this previous work performed relatively well, achieving an accuracy of around 70%. The class prototypes it generated brought forth some level of insight to our problem domain despite its numerical instability. Furthermore, the author had conducted a manual search of the hyperparameters of our neural network, making it highly likely that there are more optimal settings that can further improve our BDNN and Class Prototypes [10].

1.3 Research Question

Noting the aforementioned limitations, this paper seeks to further improve our previous approach in creating Bidirectional Neural Networks for this dataset. That is, we seek to answer *what the optimal architecture and hyperparameters for a Bidirectionally trained neural network* are.

We use an Evolutionary Algorithm to answer this question. We explore our complex hyperparameter search space and search for the possible architectures and hyperparameters needed for an optimal model with higher accuracy. We also explore the optimal hyperparameters for a BDNN's whose metric for evaluation differs from accuracy.

2 Method

2.1 Dataset Description

In this paper we use the dataset obtained from [1] pertaining to an experiment with eye gaze on manipulated images relevant to the previous work. There are 372 datapoints in the dataset, all of which have numerical features. Table 1 shows a description of the dataset's and its corresponding feature descriptions, including our additional target feature 'prediction'.

Table 1. Eye Gaze on Image Manipulation Dataset Description

Feature Name	Description
participant	ID number of the experiment participant
num_fixs	Total number of fixations by the participants spent looking at the image
fixs_duration	Total amount of time (in seconds) that the participant spent looking at the image
num_man_fixs	Total number of fixations by the participant when looking within target area
man_fixs_dur	Total amount of time (in seconds) the participant spent looking within the target area
image	The image the participant was viewing
image_manipulated	Whether the image is manipulated (1) or not (0)
votes	Verbal opinion of participants on whether they deem the image to be unmanipulated (0), manipulated (1) or unsure (2)
predicted	Whether the participant classifies the image correctly. (0), true positive, (1) true negative, (2) false positive, (3) false negative

We have opted for the continued use of this dataset from our previous work with it, allowing us to measure our improvement from a standardized baseline. This paper is also exploring the efficacy of BDNN's and EA's in this problem domain of shallow neural networks with limited data, in contrast to deep neural networks and plentiful data [8].

2.2 Data Pre-processing

We omit column's that will not be useful for this classification task or might leak information about the target variable, namely "image".

We also slightly alter our target variable. We removed all data instances where the participant answered "unsure" on their vote of the image. This was done as these rows are naturally outliers in our data and their eye gaze behavior may only mislead other classifications. Furthermore, we removed 'vote' and added a new target feature named 'predicted', which is nominal in nature corresponding to whether the participant correctly classified the experiment's techniques.

Normalization is applied to the data. We chose to use a Standard / Z-score normalization method to all our columns, reducing the standard deviation significantly.

2.3 Autoencoder and Bidirectional Neural Network

We first create an Autoencoder neural network as our BDNN, and train it in both directions using a Stochastic Gradient Descent optimiser.

To create an Autoencoder Neural Network, we create neural network with the same number of inputs as outputs. We also ensure that the hidden layers in between the two are less than that of the input size. This is because, the purpose of using an Autoencoder for our model to learn a simplified version of our input data that is sufficient enough to still accurately represent it [4]. Thus, when trained correctly, the output of the hidden layer is a compressed or encoded representation of our original input pattern.

Implementing this in modern framework such as Pytorch involves creating two symmetrical neural networks. One of the models represents the forward training direction, while the other represents the backward training direction. When transitioning between a forward training and backward training in the BDNN, we simply switch the weights of the previously trained model to the other direction's model using the state_dict. This approach essentially emulates a Bidirectional Neural Network. To improve invertibility of the autoencoder, we also remove all biases from our model.

It is important to note that there are numerous ways of training our BDNN, depending on when the training direction swap occurs. In our implementation, we allow three types; cycle-wise, epoch-wise and batch-wise. A cycle-wise training switches the training direction only after one direction has completed a full training cycle for the given number of epochs. In epoch-wise, we switch the training direction every epoch, and for batch-wise we switch after every batch. This distinction is important, as Gedeon [4] has shown these approaches can affect the performance of our BDNN.

2.4 Classification Neural Network

After training our BDNN, we then train our classifier neural network that will classify our target data *predicted*.

Figure 1 below offers a simplified visualization of our approach. The first two layers of our BDNNs in our classifier neural network would be taken from the first two layers of our trained BDNN. From this, we add multiple layers of hidden neurons from these two layers and converge them into our classification output neurons. We use a CrossEntropyLoss function for our classifier as it is appropriate for our multiclass classification neural network. We then train our classifier net using the Stochastic Gradient Descent optimizer.

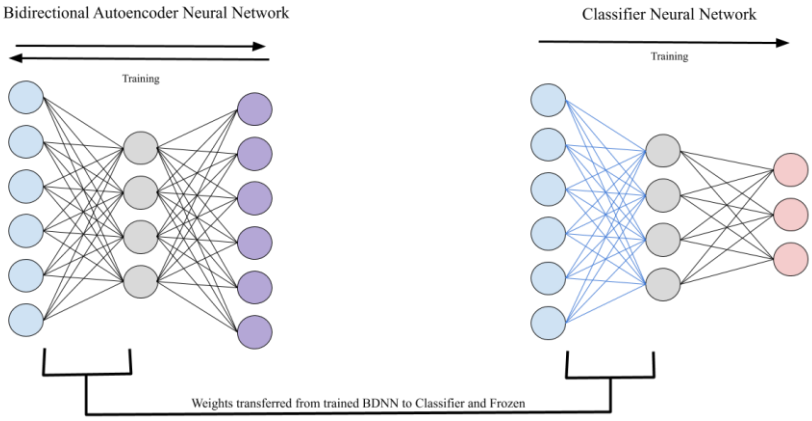

Fig. 1. Diagram of how the classifier neural network operates. Once the BDNN is trained in both directions, we transfer the first layer of weights to our classifier neural network. We then train the classifier neural network to classify our data correctly, ensuring that the imported weights are not affected by our optimizer's backpropagation.

We then evaluate our classifier using our train test split. We split our dataset such that 80% was used for training, and the other 20% used for testing. We retrieve the accuracy of our classifier model on this testing set as an evaluation metric.

2.5 Evolutionary Algorithm for Hyperparameter Search

Search Space

We first define the search space for our hyperparameters. This is important as numerical hyperparameters can have an infinite amount of values, thus making our search infinitely large [10]. Furthermore, there are countless many other hyperparameters that are part of a given neural network model that contribute in varying amounts to the performance of the model. Thus, we have chosen a set of hyperparameter values to search and we restrain each hyperparameter to a select amount of values. Table 2 below shows the hyperparameter search space, with 13 different hyperparameters.

Table 2. Hyperparameter Search Space Defined

Feature Name	Possible Values	Description
Autoencoder Hyperparameters		
encoding_size_decrease	1, 2, 3, 4	Difference between input size and hidden layer size
activation	Tanh, sigmoid, relu, leaky relu, tanhshrink, softsign	Activation function used in neural network
learning_rate	0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001	Learning rate for model's SGD optimizer
momentum	0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0	Momentum for model's SGD optimizer

bidirection_change	'batch', 'epoch', 'cycle'	Specification of when training direction switches in Bidirectional Training
num_epochs	100, 500, 1000, 1500, 2000	Number of epochs to train
batch_size	100, 50, 10, 1	Number of datapoints in each batch
Classifier Hyperparameters		
hidden_layers_class	10, 20, (10, 5), (10, 20), (20, 10)	Hidden Layer structure of layers following second layer (from BDNN)
activation	Tanh, sigmoid, relu, leaky relu, tanhshrink, softsign	Activation function used in all hidden layers neural network
learning_rate	0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001	Learning rate for model's SGD optimizer
momentum	0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0	Momentum for model's SGD optimizer
num_epochs	100, 500, 1000, 1500, 2000	Number of epochs to train
batch_size	100, 50, 10, 1	Number of datapoints in each batch

Given the above search space, there are over 6×10^{10} possible combinations of hyperparameter values. Clearly, a grid search among this space would be incredibly inefficient and time consuming.

Chromosomes and DNA

Our next step is to define our DNA and Chromosome for a given agent. We use a binary encoding for the chromosome to simplify the other EA function implementations. To encode the possible values for each 13 possible variables, we use a 52 bit binary array where every 4bits represents a value in the hyperparameter search space. Because the largest number a 4 bit digit can represent is 15, and not all of our hyperparameters have 15 different possibilities, we normalize the binary value to fit within each hyperparameter's possibilities. Figure 2 below shows an example.

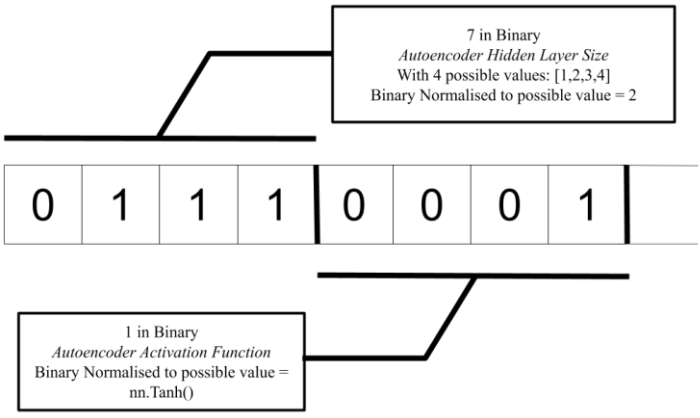

Fig. 2. Diagram of the Chromosome Layout. One chromosome will contain 52 bits. Every 4 bits represents an index of a hyperparameter's possible value array space. In the figure above, the first 4 bits represents the hyperparameter of "Autoencoder Hidden layer Size" and represents the hyperparameter value 2. This pattern continues for all 13 hyperparameters.

Crossover, Selection and Mutation

We employ two crossover functions; a random crossover and a one-point crossover. The one-point crossover intersection point is at bit 28, meaning that it splits a chromosome into its autoencoder hyperparameters and it's classifiers hyperparameters. These two functions are chosen at random, as multiple random crossover functions have been shown to improve the performance of the evolutionary algorithm [9]. We initialized our crossover rate to 0.6.

Our mutation function is a random mutation function, that will flip the bits of a random gene in a chromosome. We initialize our mutation rate at 0.002.

Our selection function is a random proportional selection. This means, the agents to be transferred to the next generation will be chosen at random, but agents that have a better fitness function will be slightly favoured.

Fitness Function

We define two fitness functions that we would like to explore

Accuracy

This fitness function simply returns the accuracy of a given agent's classification model on the test data set.

Accuracy with Compression Penalty

This fitness function returns the accuracy of a given agent's classification model on the test data set, but a penalty based on the autoencoders hidden layer size. The equation is as follows:

$$fitness = accuracy + encoding\_size\_decrease. \quad (1)$$

This fitness function aims to increase the emphasis on the autoencoder to achieve the best possible compression, thus rewarding agents whose autoencoder are able to compress the input data in a much better way. Without this, a simple accuracy encoder may just favour the autoencoder to have more neurons in its layer to minimize compression and therefore allow the classifier model to work with more data.

2.6 Experiment Design

We employ all the above initializations of our BDNN, classifier and evolutionary algorithm and compare the optimal parameters between the two fitness functions; *Accuracy* and *Accuracy with Compression Penalty* and find their optimal hyperparameters. We discuss our findings below.

3 Results and Discussion

We ran our evolutionary algorithm for both fitness functions with a population size of 10 and 50 generations. We found that it took approximately 3 hours for the algorithm to finish for both.

For our Accuracy fitness function, we obtained the following optimal hyperparameters:

Table 3. Resulting Optimal Hyperparameter values for *Accuracy* Fitness Function

Feature Name	Optimal Values	Description
Autoencoder Hyperparameters		
encoding_size_decrease	2	Difference between input size and hidden layer size
activation	tanhshrink	Activation function used in neural network
learning_rate	0.1	Learning rate for model's SGD optimizer
momentum	0.3	Momentum for model's SGD optimizer
bidirection_change	'cycle'	Specification of when training direction switches in Bidirectional Training
num_epochs	1000	Number of epochs to train
batch_size	50	Number of datapoints in each batch
Classifier Hyperparameters		
hidden_layers_class	(10, 20)	Hidden Layer structure of layers following second layer (from BDNN)
activation	relu	Activation function used in all hidden layers neural network
learning_rate	0.5	Learning rate for model's SGD optimizer
momentum	0.7	Momentum for model's SGD optimizer
num_epochs	100	Number of epochs to train
batch_size	50	Number of datapoints in each batch

This model achieved an accuracy of 73.1% on our testing set, which is an improvement on previous work [5][6].

One interesting observation we found was that our encoding_size_decrease was higher than the minimum value of 1. We had anticipated that a simple accuracy fitness function would inadvertantly cause the autoencoder to preserve as much of the original input data as they could by specifying a larger hidden layer size to pass on to the classifier function. In contrast, a value of 2 indicates that the autoencoder training has contributed to some level of compression of our input pattern that can be more appropriate to classifier models than simply giving it the entiritiy of the input pattern.

Table 4. Resulting Optimal Hyperparameter values for *Accuracy with Compression Penalty* Fitness Function

Feature Name	Optimal Values	Description
Autoencoder Hyperparameters		
encoding_size_decrease	4	Difference between input size and hidden layer size
activation	tanh	Activation function used in neural network
learning_rate	0.1	Learning rate for model's SGD optimizer
momentum	0.5	Momentum for model's SGD optimizer
bidirection_change	'epoch'	Specification of when training direction switches in Bidirectional Training
num_epochs	1000	Number of epochs to train
batch_size	50	Number of datapoints in each batch

Classifier Hyperparameters		
hidden_layers_class	(10, 20)	Hidden Layer structure of layers following second layer (from BDNN)
activation	Sigmoid	Activation function used in all hidden layers neural network
learning_rate	0.05	Learning rate for model's SGD optimizer
momentum	0.3	Momentum for model's SGD optimizer
num_epochs	1500	Number of epochs to train
batch_size	50	Number of datapoints in each batch

This model also achieved a similar accuracy of 73%, despite the encoding size decrease.

It is clear to see that the optimal hyperparameters for our fitness function that awards compression levels are different in our results. For a higher compression rate with only two hidden nodes, the autoencoder employs the 'epoch'-wise bidirectional training. This may indicate that epoch wise training is more conducive to trainig BDNN's Autoencoders that have a high compression rate compared to 'cycle'-wise as we observed previously.

4 Conclusion

In conclusion, we employed an Evolutionary Algorithm to explore the optimal hyperparameters for a Bidirectional Neural Network. Our Implementation of the BDNN involved training an autoencoder bidirectionally and applying the weights to a classifier. We explored two different metrics of performance for our BDNN translated into a fitness function for our EA. That is, *Accuracy* of the model and *Accuracy with Compression Penalty* of the model. We had found that the optimal hyperparameters for these two fitness functions given by our EA were different. This is in line with research that posits that hyperparameters for a neural network are domain specific [11]. One interesting finding is that some level of compression in our autoencoder can benefit our classifier's accuracy, going against the notion that the autoencoder would preserve as much data as possible for the classifier. Furthermore, the bidirectional training occurrences has an impact on the quality of training an BDNN autoencoder receives, and correlates to the neurons in the hidden layers.

Application of these optimal hyperparameters on our BDNN's resulted in an accuracy over around 73%.

There is significant room for more research in the combination between BDNN and evolutionary algorithms. This paper proves the efficacy of this combination yet includes many limitations. The EA algorithm employed in this paper is far from sophisticated, and thus exploration on altering the EA's crossover, mutation, fitness and selection functions on a BDNN hyperparameter optimization is needed. This is especially true with fitness functions, applying different metrics such as precision, f1 and roc may uncover different optimal hyperparameter settings. Furthermore, our findings were limited by our predefined hyperparameter search space, and thus there is possibility to conduct searches on a larger hyperparameter space. Furthermore, combining them with other techniques such as pruning, dropout and other modern techniques may uncover different optimal hyperparameters and settings.

References

1. Caldwell, S., Gedeon, T., Jones, R., Copeland, L.: Imperfect Understandings: A Grounded Theory And Eye Gaze Investigation Of Human Perceptions Of Manipulated And Unmanipulated Digital Images. In: Proceedings of the World Congress on Electrical Engineering and Computer Systems and Science (EECSS 2015) (2015).
2. Nejad, A.F., Gedeon, T.D.: Bidirectional neural networks and class prototypes. Proceedings of ICNN'95 - International Conference on Neural Networks. (1995).
3. Gedeon, T.D.: Stochastic bidirectional training. SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218). (1998).
4. Gedeon, T.D., Catalan, J.A., Jin, J.: Image Compression using Shared Weights and Bidirectional Networks.
5. Tan, Z., Plested, J.: Classification of Humans' Perception of Manipulated and Unmanipulated Digital Images Using Feedforward Neural Network with Network Reduction Technique. ICONIP2019 Proceedings. (2019).
6. Caldwell, S.: Framing digital image credibility: image manipulation problems, perceptions and solutions, https://openresearch-repository.anu.edu.au/bitstream/1885/133844/1/Caldwell%20Thesis%202017.pdf, (2016).
7. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing. 45, 2673–2681 (1997).
8. Young, S.R., Rose, D.C., Karnowski, T.P., Lim, S.-H., Patton, R.M.: Optimizing deep learning hyper-parameters through an evolutionary algorithm. Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments - MLHPC '15. (2015).
9. Cantu-Paz, E., Kamath, C.: An Empirical Comparison of Combinations of Evolutionary Algorithms and Neural Networks for Classification Problems. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics). 35, 915–927 (2005).
10. Bergstra, J., Bengio, Y.: Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research. (2012).
11. Martinez, G.: Finding Optimal Neural Network Architecture Using Genetic Algorithms. (2007).