BDNN for Classification with Genetic Algorithm Hyperparameters Optimization

Dan Yang u6165740@anu.edu.au

Research School of Computer Science, Australian National University

Abstract. This paper creates a backpropagation trained BDNN (Bidirectional Neural Network) to predict the status of a patient given the target's temperatures, blood pressure and nausea level. It illustrates methods of creating bijection between input and output instances for model construction and shows how BDNN is implemented in a classification problem. The paper then investigates the potential improvements to BDNN including hyperparameters optimization using genetic algorithm. It later investigates input neurons pruning and compares BDNN with baseline multiclass classification neural network, where it deduces the suitable situation to apply BDNN. Lastly the paper discusses potential retrievable data knowledge from the BDNN model.

Keywords: Neural network, BDNN, backpropagation, classification, genetic algorithm, hyperparameters optimization

1 Introduction

The Bidirectional Neural Network is an intuitive structure from the neural science since the synaptic transmission can be either electrical or chemical and where it is electrical the transmission is usually bidirectional [1]. It enables the model to remember both the input patterns and output vectors as associative memories and can extract meaning from the trained neural networks, which is seen to improve the acceptability of neural networks [2]. The training of the model iterates between training from inputs to outputs and training inversely with weights in both directions remained the same waiting for update. Therefore, this structure requires a one-to-one mapping between inputs and outputs as otherwise the inverse training is impossible. The construction of such bijective correspondence is one of the main challenges of BDNN and several methods confronting this have been discussed by Nejad and Gedeon [1]. This paper will implement one of them and discuss the others.

Once a BDNN classification model is created, we will further optimize hyperparameters using genetic algorithm to have an optimal model. After that, we will investigate input neurons pruning and from that deduce the suitable situation for using BDNN for classification.

Due to the association of inputs and outputs, it is natural to believe that BDNN would have insight of the data representation and can be used to retrieve missing input data. Details of such feature will be investigated later in the paper.

1.1 Dataset

The dataset used in this paper is from B.S. Mendis et al.'s Investigation of Aggregation in Fuzzy Signatures Paper [3]. It includes four sub datasets from normal people and patients suffering from SARS, High Blood Pressure and Pneumonia with each dataset recording the target's temperatures over time, systolic and diastolic blood pressure and nausea levels. The goal is to predict the status of a patient (Normal, SARS, HighBP, Pneumonia) based on his medical information (23 patterns) recorded in the dataset. In this manner, the problem could be treated as a classification problem.
The later part of the paper will also discuss pruning of certain input patterns at the minimal sacrifice of the accuracy of the model.

Table 1. Snapshot of the Raw Data.

Temp 8am				Temp 8pm			BP Systolic			BP Diastolic			Nausea			Abdominal Pain	
Slight	*Mod*	*High*		*Slight*	*Mod*	*High*	*Slight*	*Med*	*High*	*Slight*	*Med*	*High*	*Slight*	*Med*	*High*	*No*	*Yes*
0.1013	0.929	0.842	…	0	0.6884	0.8575	0	0.7626	0.8238	0.0352	0.8114	0.6855	0.0652	0.5279	0.8177	0	0.9594
0.8827	0.1286	0		0.8639	0.152	0	0.9144	0.0285	0	0.9047	0.0287	0	1	0	0	1	0
1	0	0		1	0	0	0	0.2427	0.8081	0	0.2431	0.9252	1	0	0	1	0
0.0827	0.8573	0.8759		0.0639	0.904	0.8846	1	0	0	1	0	0	1	0	0	1	0

2 Model Methodology and Design

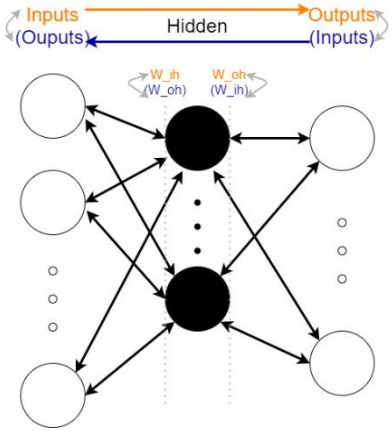
This section covers the preprocessing of dataset, BDNN classification model construction (including methods of creating a one-to-one mapping between input and output, classification strategy and training) and measurement.

2.1 Data Preprocessing

The given datasets are separated with each file representing a unique label. Therefore, the priority is to attach labels to the data before concatenation. Each dataset was read as a dataframe and four medical conditions 'SARS', 'Normal', 'HighBP', 'Pneumonia' were encoded as labels 0,1,2,3 and attached to the dataframe. The four dataframes were then concatenated and shuffled to be random. After tweaking the data to meet the one-to-one mapping criterion of inputs and outputs (covered in next part), 80% of the data was separated as training data with the remaining being the testing data. Inputs and outputs were further separated accordingly.

2.2 BDNN Construction

The construction of BDNN is an extension to the normal neural network: the network is invertible, meaning it not only predicts the corresponding label from the medical data but also retrieves the medical data given the label. Therefore, there is a bijection between input and output instances. However, since the classification is a many-to-one problem, such one-to-one mapping must be manually constructed.



2.2.1 One-to-one Mapping

Two major techniques for generating one-to-one mapping between input and output instances have been discussed by Nejad and Gedeon [1]: 1. Adding an extra node to the output instances; 2. Subclassification of the input patterns which is similar to a common technique used in Fuzzy Input Patterns [4]. For the simplicity of understanding BDNN, the first method is used.

A natural way of creating an extra node is taking the mean value of all features in the input instance. However, since all values in the input data are between 0 and 1, there are duplicates among such extra nodes. Therefore, we cannot guarantee the bijection between input instances and (label, extra_node) outputs.

A solution was enlightened by the L2-distance between each input instance v_i and mean of all input instances $\bar{v}$. Moreover, to guarantee the uniqueness (thus bijection), from the experiments, instead of taking $\sqrt{\sum_{j=1}^{23}((v_i)_j - \bar{v}_j)^2}$, we use $extra\_node_i = \frac{1}{23}\sum_{j=1}^{23}((v_i)_j - \bar{v}_j)^2$. In this way (label, extra_node) results in 4000 different output instances. Since there are 4000 different input instances, a one-to-one mapping is formed.

2.2.2 Network Layout

Once a bijection is formed between input and output instances, we could construct the network. For simplicity, only one hidden layer is used (for now). A function is created to notate weights W_{ih} (wights on input hidden neurons) as W_{oh} (weights on output hidden neurons) in the inverse network (and notate W_{oh} as W_{ih}) as shown in the above picture. A basic neural network structure is made and then called twice with input dimension and output dimension swapped to have a pair of neural networks with inverse directions. The realization of BDNN is completed during the training process.

2.2.3 Training

With the one-to-one mapping between input instances and (label, extra_node) output instances, the classification problem could be transformed to a regression problem with multiple neurons on both ends. During one epoch of training, backpropagation is applied in both forward and reverse direction with the weight re-notating function called. Since the network model is inversive regression, we use the same optimizer for both forward and backward directions. The choice for such optimizer will be investigated with its hyperparameters in the next chapter. The loss functions are set to MSE which is an L2 norm error usually used for regression.

2.3 Measurement

As our focus is classification, only the forward direction (from medical data to (label, extra_node)) will be measured. Also, since extra_node is used mainly for bijection construction, we are more concerned about the label (the extra_node has its own usage which will be discussed later). Since the model is treated as a regression, the label is treated as a floating number. Therefore, for accuracy, the result will be considered accurate if the predicted label value falls into a range around the real integer label. The allowed range is set to ±0.5 (exclusive) for this specific model which is equivalent to the 'nearest integer'. The loss of the prediction is made by the MSE error between predicted (label, extra_node) and the real value. The code of accuracy measurement could be seen in the supporting document.

3 Hyperparameters Optimization with Genetic Algorithm

This section first gives a unoptimized BDNN model. It then investigates the selection of optimizers with corresponding hyperparameters like learning rate and momentum. The hyperparameters optimization used here is deduced from genetic algorithm.

3.1 Initial Model

We first selected SGD (Stochastic Gradient Descent) with learning rate set to 0.01 and momentum set to 0.9. The number of hidden neurons was first set to 20. The accuracy and loss for both training and testing data during the 300 training epochs (each epoch includes backpropagations on both directions) are shown in Fig. 1.

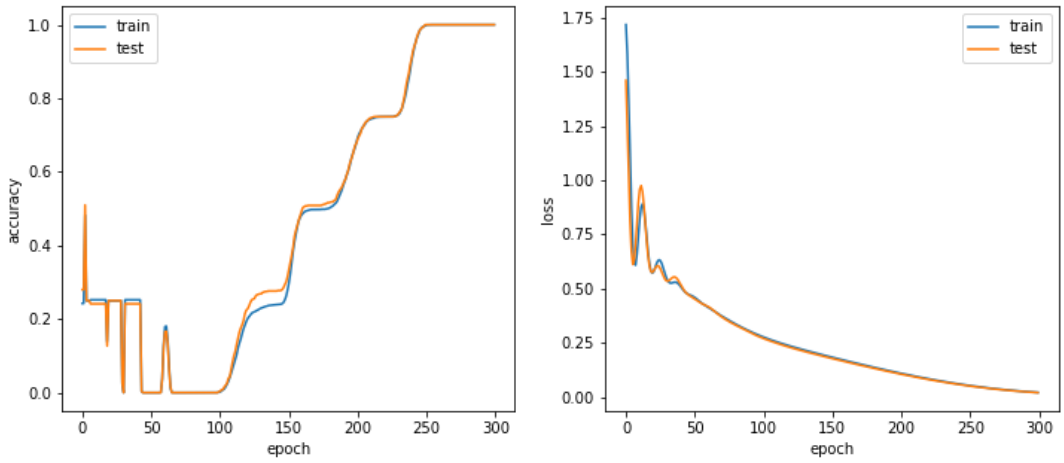


Figure. 1. optimizer=SGD (1r=0.01, momentum=0.9) #hidden_neuron=20

Epoch	Testing Accuracy	Testing Loss
100	0.25%	0.2723
150	31.59%	0.1789
200	67.79%	0.1065
250	100%	0.0514
300	100%	0.0204

Table. 2. Testing Accuracy & Loss

As can be seen from the figure and the table, the training of the model with current hyperparameters converges after 250 epochs, which is relatively slow. Also, in the first 100 epochs, the loss oscillates severely, and the accuracy often gets stuck. This is most likely because the optimizer focuses more on the extra_node in the early stage due to the poor choice of hyperparameters. For the training process to converge faster, hyperparameters in SGD will be optimized using genetic algorithm.

3.2 Genetic Algorithm Hyperparameters Optimization

3.2.1 Differential Evolution Algorithm

Evolutionary Algorithms, inspired by the natural evolution of species, have been successfully applied to solve numerous optimization problems in diverse fields.[8] The specific genetic algorithm used here is a variation of differential evolution algorithm introduced in [8], where parameters are equivalent to genes in biological systems. A population is formed by vectors with each representing a distinct set of parameters. Different combinations of parameters result in different fitness values for a vector. A vector with a higher fitness outlives others and will pass its features to further generation via mutation.
Differential evolution algorithm can be implemented as an iterative process. Generalized steps for our specific model are given below:

1. Initialization: generate an initial population with N vectors filled with randomized hyperparameters' values $[hp_1^1, hp_2^1, \dots, hp_n^1], [hp_1^2, hp_2^2, \dots, hp_n^2], \dots, [hp_1^N, hp_2^N, \dots, hp_n^N]$
2. Initial evaluation: calculate the scores of BDNN model (here we use test accuracy) for each vector of hyperparameter values. Retrieve the best score and the corresponding vector.
3. While population converges:
   For each hyperparameter hp_j^i in each vector:
   Generates a random number $rand \sim Uniform(0,1)$
   If $rand < recombination\_rate$:
   Mutation: $hp_j^i = hp_j^{best} + mutation\_rate * (hp_j^{random1} - hp_j^{random2})$
   Else:
   $hp_j^i = hp_j^i$
   Calculate the scores for the new population, retrieve the new best score with corresponding vector.

With this algorithm, hyperparameters that result in a higher accuracy will pass to the next generation (with probability recombination_rate) in the form of mutant. Through evolution, the future generation will converge to an optimal solution.

3.2.2 SGD Hyperparameter Optimization

There are several details for the implementation of the differential evolution algorithm given above: 1. The number of epochs is limited to 100 as we seek faster convergence; 2. In order to get a simpler model, the number of hidden neurons is set to 10; 3. The hyperparameters in SGD for optimizing are the learning rate and momentum; 4. Initialization of population (a set of combinations of hyperparameters) is made by randomization where learning rate follows an exponential distribution with $\frac{1}{\lambda} = 10$ and momentum follows a uniform distribution in (0,1); 5. The population convergence is determined by MSE of the population; 6. Population size was set to 10, recombination rate was set to 0.7 and mutation rate was set to 0.5.
From the genetic algorithm, an optimal solution was found as (learning_rate=0.3, momentum=0.06). As can be seen from Fig. 2, the training converges much faster (after epoch 75) than the previous (epoch 250) and the final accuracy of BDNN model is also 100%.

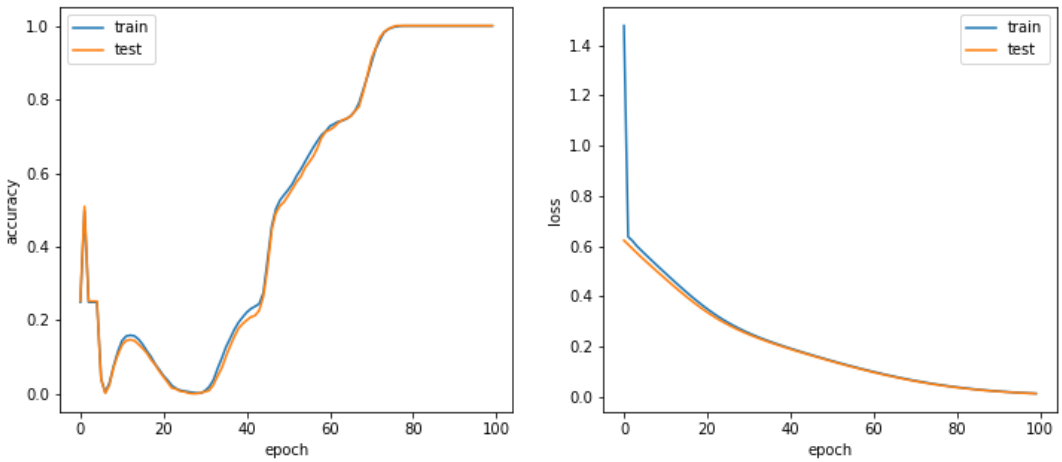


Figure. 2. optimizer=SGD (1r=0.3, momentum=0.06) #hidden_neuron=10

3.2.3 Limitation of Differential Evolution Algorithm

Another hyperparameter worth investigating is the number of hidden layers. We have tried to use differential evolution algorithm to optimize this hyperparameter together with the learning rate and momentum, where the mutation for the number of hidden neurons is projected to a nearest integer. The result was poor. A possible reason is that for integer parameters, the mutation in differential algorithm hardly preserves the feature from the previous fittest hyperparameter, and projecting to a nearest integer further enlarges such problem. Therefore, we could conclude that differential evolution algorithm is better suited for floating hyperparameters, and it will be even better if those hyperparameters are completely continuous.
A more compatible genetic algorithm for hyperparameters optimization will be left for future investigation.

3.3 Adam

While the learning rate in SGD could be optimized for faster convergence, it has an equivalent type of effect for all the weights/parameters of the model, which is not quite ideal. An alternative is using adaptive learning rate per parameter. One commonly used optimizer with such feature is Adam (Adaptive Moment Estimation). Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments [5]. Parameters that would ordinarily receive smaller or less frequent updates receive larger updates with Adam (the reverse is also true). Therefore, it speeds up the training.
However, due to the fast speed of Adam, for the current model with given data, Adam could easily skip the optimal solution. If we reduce its learning rate, the number of epochs will be sacrificed. With a balance between learning rate and number of epochs hard to achieve, Adam fails to show advantage over SGD under the current structure. However, it will have usage in the next section.

4 Further Investigation

4.1 Input Neurons Pruning

Due to the association between input and output, a potential advantage of BDNN is to allow fewer input neurons. The experiment was carried by using only the ‘temperature over time’ data as input to predict the label. With fewer input neurons, the training is expected to be harder to converge. Therefore, Adam is preferred here. With a relatively high learning rate 0.04 for Adam, the training still took many more epochs to converge (430 epochs), with a converged model having test accuracy only around 90%, which is significantly lower than before.

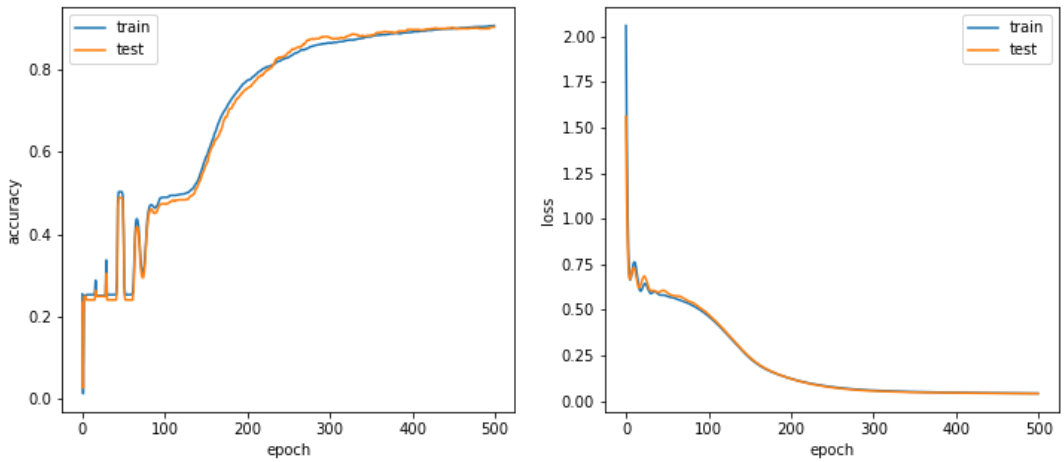


Fig. 3. optimizer=Adam (1r=0.04) #hidden_layers=1 #hidden_neuron=10 #input_neurons=12

4.2 Input Neurons Pruning with Multiple Layers

A potential strategy to resist the effect of fewer input neurons is increasing the number of hidden layers. The experiment was carried based on the above case where the learning rate remained the same. Two hidden layers were used in this case with corresponding numbers of hidden neurons being 8 and 4. Weights on hidden neurons were swapped properly during training. The BDNN model now converges much faster during training (100 epochs) with a greatly improved testing accuracy at 99%.

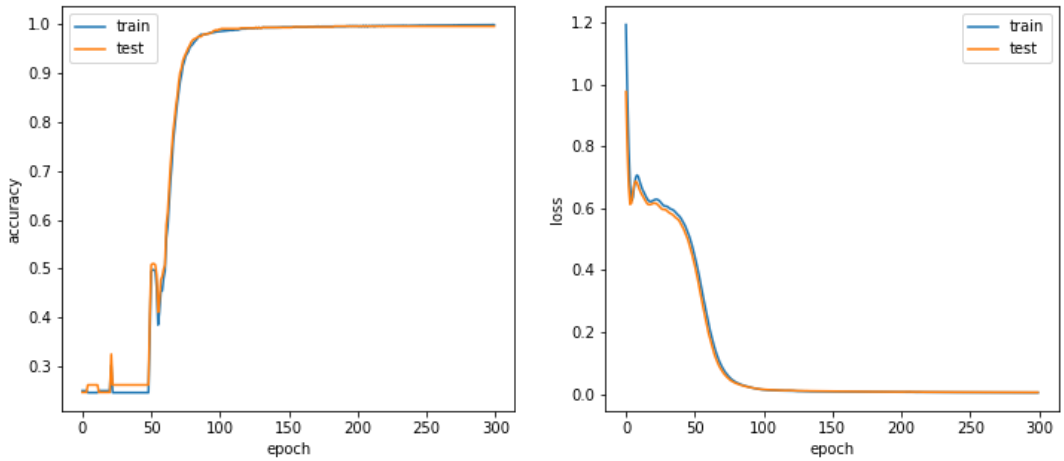


Fig. 4. optimizer=Adam (1r=0.04) #hidden_layers=2 #hidden_neuron=8,4 #input_neurons=12

4.3 Comparison with Baseline Classification Neuron Network

The Baseline Multiclass Classification Neuron Network is an extension to the normal Binary Classification Neural Network with input being the data features and output being a 4-element-vector for softmaxing and choosing the index of maximum.
There are 2 sets of comparisons between BDNN and Baseline Multiclass Classification.

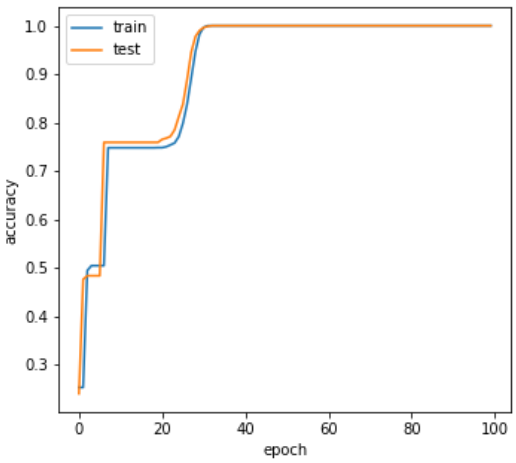


Fig. 5. Baseline with full input neuron optimizer=Adam (1r=0.01) #hidden_neuron=10

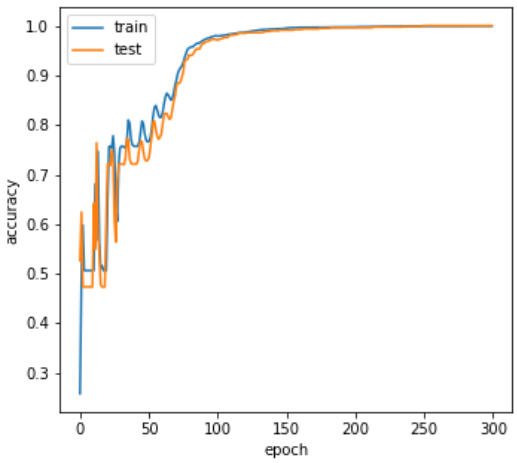


Fig. 6. Baseline with pruned input neurons optimizer=Adam (1r=0.04) #hidden_neuron=10

The first comparison is based on the same input condition as 3.2.2 where the input is complete. From the comparison between Fig. 2 and Fig. 5, it can be deduced that for the simple classification issue BDNN suffers from low training efficiency due to its multiple learning which is like the problem of bidirectional associative memory [6].
The second comparison is based on the same condition as 4.2 where several neurons of input are pruned. From the comparison between Fig. 4 (training converges around epoch 100 with test accuracy 99%) and Fig. 6 (training converges around epoch 150 with test accuracy 100%), it can be deduced that when the input patterns are missing critical information, BDNN with multiple layers could perform as good as, or even better than the Baseline Multiclass Classification Neuron Network due to its ability to establish an association between input patterns and output vectors thus having an intuition of the data relation.

4.4 Missing Data Retrieval

BDNN could be used for assigning the most likely value for an unknown data. This characteristic could be illustrated in the following example where the first line of data in **Table 1** will be used:
This piece of data belongs to a SARS patient. Suppose the Nausea-High value 0.8177 (the 21st column) is missing. Back-engineering could be performed once the method of generating extra_node is checked. In this paper's model, the generation of the extra_node is shown in 2.2: $extra\_node_i = \frac{1}{23}\sum_{j=1}^{23}((v_i)_j - \bar{v}_j)^2$ where $\bar{v}$ is a known value. Therefore, extra_node for this line could be approximated by following the same formula except ignoring the missing entry, i.e., $extra\_node = \frac{1}{22}(\sum_{j=1}^{20}(v_j - \bar{v}_j)^2 + \sum_{j=22}^{23}(v_j - \bar{v}_j)^2)$ in this input.
Since the label of this piece of data is also known, with the pair (label, extra_node), the original full data could be approximated by the inverse direction network and the missing value could therefore be approximated.
Due to the relatively poor structure from output to input in this model, the approximated missing value is 0.77. While not quite accurate, it could still be considered as a plausible guess.
For a better structured BDNN model, the value assigned would be more accurate. This will be further investigated in the future work.

5 Conclusion and Future Work

BDNN is a useful neural network with the potential of showing the relation between patterns. It can be used for classification problems especially when the given data is incomplete. This paper has shown a step-by-step implementation of BDNN with optimizations and analysis on extreme situations.
Genetic algorithm, like BDNN, is also bio-inspired. It preserves characteristics of parameters that contribute to the model by passing them to next generations via mutation. Therefore, it could be used for optimization. This paper has shown an implementation of using one of the genetic algorithms to perform hyperparameters optimization for a neural network model.

5.1 Limitations

This paper focused on the adding extra node method to generate a one-to-one mapping between input and output instances. Compared to subclassification of the input patterns method, this method is comprehensive and easy to implement with the cardinal principles of BDNN preserved. However, such method has its limitations. The added extra_node, especially for our dataset, does not reflect much useful information of the data. Therefore, the model gives little information in the backward direction and is unable to be trained as cluster center finders.
The Differential Evolution algorithm used for hyperparameters optimization in this paper also has its limits. The mutation method restricts that the algorithm is only suitable for floating number whereas integer hyperparameters like number of hidden neurons and number of hidden layers cannot be optimized by this genetic algorithm.

5.2 Future Work

The future work involves investigating subclassification of the input patterns as preparation of BDNN. The BDNN model using subclassification of input patterns could be experimented on this paper's dataset since the structure of this dataset is close to SFM (Gedeon and Turner's Student Final Marks [7]). Also, with that model, more features of BDNN could be explored and generalized.
Another potential future work given this dataset is to use fuzzy signatures as hidden neurons for a classification neural network as patients' medical data is recorded as fuzzy values in our dataset.
Furthermore, more genetic algorithms will also be learned for a hyperparameters optimizer with better compatibility.

References

1. Nejad, A.F., Gedeon, T.D. (1995) BiDirectional neural networks and class prototypes, *Proceedings of ICNN'95 - International Conference on Neural Networks*, pp. 1322-1327 vol.3, doi: 10.1109/ICNN.1995.487348.
2. Gallant, S.I. (1988) Connectionist expert systems, *Communications of the ACM*, vol. 31, no. 2
3. Mendis, B.S., Gedeon, T.D., & Koczy, L.T. (2005) Investigation of aggregation in fuzzy signatures, *Proceedings of 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems, Singapore*.
4. Ishibuchi, H., Morioka, K. (1995) Calssification of fuzzy input patterns by neural networks, Proceedings of *ICNN'95 - International Conference on Neural Networks*, 1995, pp. 3118-3123 vol.6, doi: 10.1109/ICNN.1995.487282.
5. Kingma D.P., Ba J. (2015) Adam: a method for stochastic optimization, ICLR Poster 2015
6. Kosko, B. (1988) Bidirectional Associative Memories, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-18
7. Gedeon, T.D., Turner, H. (1993) Explaining student grades predicted by a neural network, *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, 1993, pp. 609-612 vol.1, doi: 10.1109/IJCNN.1993.713989.
8. Qin, A.K., Huang, V.L., & Suganthan, P.N. (2009) Differential evolution algorithm with strategy adaptation for global numerical optimization, *IEEE Transactions on Evolutionary COmputation*, 2009, doi: 10.1109/TEVC.2008.927706.