Investigating Efficiency of Hybrid Genetic Algorithm-BDNN on Classifying Real and Posed Anger

Lai Hong Chiu

Research School of Computer Science
The Australian National University
u6386686@anu.edu.au

Abstract. We investigated how well Bidirectional Neural Network can predict real and posed anger depicted by videos using summarized pupillary data from corresponding observers and how applying Genetic Algorithm on top of such can help find a good performing hyperparameters. We preprocessed the data by adding extra node to each entry label to enhance input output coupling. We used two training methods: back-forth method involves calculating the loss of a batch as sum of predicting forward and reverse forward, while alternate method involves training model with one fixed direction at a time and switch direction for some criteria until either loss is lower than a tolerance. [7] We splitted the dataset into train, validation and test sets and tuned models for highest validation accuracy. Evaluating validation accuracy and testing accuracy shows that BDNN alone does not perform any better than an ensemble of neural network and unsupervised classifier [1], and BDNN does not generalize well to the testing set unknown to the model. We also tested on GA-ANN hybrid models, which is a automatic parameter tuner for given ANN, for "breeding" a good set of hyperparameter using stochastic uniform selection [20], uniform crossover and elitism on a wide range of parameters and discovered that it can indeed accelerate finding a better solution of model and have considerably higher accuracies comparing to first generation, nevertheless, the generated accuracies still did not outperform results from [1], but the similarity in behaviors of generations from GA combining with both BDNN and FFNN further reinforce the claim that they perform no better than each other.

Keywords: Bidirectional Neural Network, emotion detection, supervised machine learning, feed-forward neural network, class prototypes, classifier, Genetic Algorithm, optimizing hyperparameters, stochastic uniform crossover, elitism

1 Introduction

In this paper we are going to investigate on how Bidirectional Neural Network (BDNN) combined with Genetic Algorithm (GA) could improve classification of acted and posed anger over a hybrid GA-Feed Forward Neural Network (GA-FFNN) classifier over the same dataset. We will also evaluate on pure BDNN and FFNN in terms of abilities to generalize and robustness to unknown data. The dataset is given by paper [1]. The dataset consists of statistical summaries of pupillary responses of different people to different videos. Each of the videos comprises of exactly one person either acting angry or genuinely angry described in [1]. Therefore, the classifiers would take pupillary responses of some videos to predict whether the video depicts genuine anger or not. Details of the methodology is given in methodology section.

Humans convey emotions and feelings about something to each other mainly by verbal or gestural languages. When communicating with others, it is important for us to know how to differentiate acted and genuine expressions to keep us from being unconsciously manipulated. In recent years, supervised deep learning is popular for predicting and classifying patterns within datasets. In particular, neural networks are especially popular for classification tasks because of its ability to arbitrarily approximate perfect predictions given enough amount of training data.

Studies have shown that it is plausibly possible to classify truthfulness of emotions of a person indirectly using receiver's gestural response. Fine-tuned ensemble classifiers comprising FFNN, SVM and KNN has successfully classified statistical summaries of pupillary responses of experiment participants from watching videos depicting person acting or genuinely angry and smiling with both over 90% accuracy. [1][2] This motivates us to do this experiment because of models' high accuracy, indicating underlying statistical relationships exists inside our anger dataset.

Recently, bidirectional learning methods has been proposed and studied. In particular, BDNN is a variant model of classic feed-forward neural networks except that it allows the same set of parameters to be trained in reverse order (i.e.: training "input" from target output). A range of applications of this model or similar learning methods have been explored and has shown some promises. Apart from the famous bidirectional RNN [3], it has been shown to be able to synthesize chemical emission and boiler efficiency models [4]; image classifying facial emotions [5]. With a proper

learning technique it can even be used to improve robustness to image classification and even achieve SOTA accuracies [6]. For BDNN, it has shown the capabilities of improving performance on many-to-one classifications and able to find centers of dataset clusters[7]. Therefore, it is definitely worth investigating on these methods.

To produce a highly performed ANN, it is necessary to fine tune its hyperparameters. Methods such as Grid-search is useful for minimizing the range of the parameters such that a model with good result can be produced. However, to get a globally optimal model (within a large domain) it is required to test over a large combination of parameters and using grid-search is time-consuming. Therefore, we combine GA with ANN to select the best parameters, while exploring given domain as complete as possible. There has been a lot of research on optimizing ANN with GA, and there are good results. For example, optimizing performance of ANN by "evolving" arbitrary connections of each hidden neurons [15], hybrid methods tuning structure and neuron weights devised for hill-climbing function extrema [16] and numerous ways are explored to combine GA and ANN to improve classification [17]. In particular, Gedeon experimented on "evolving" model structure for a hybrid GA-ANN-SVM model on classifying stress features and achieved 89% accuracy [18]. This is like the experiment conducted in [1] and our task in nature. We believe it is worthy to test a similar hybrid GA models to test on its performance for our task as well.

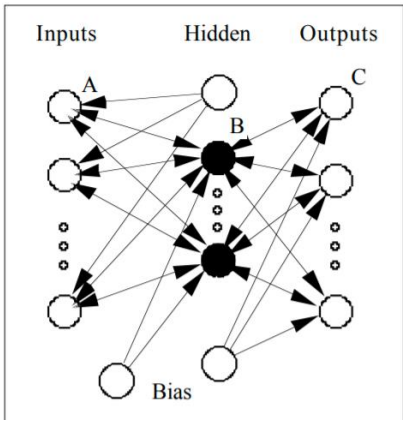
2 Method

2.1 Architectures of models

In this paper we will compare the performance between FFNN and proposed BDNN [7] in terms of ability to generalize and overfitting, then we would compare the performance of hybrid GA-BDNN and GA-FFNN.

2.1.1 Bidirectional Neural Network

We implement a bidirectional neural network as described by paper [7].
BDNN has a similar architecture with feed forward version, except that we can interchange between inputs and targets (in the same dataset) and train in the same model. For each X corresponding to each Y, we can think of two models: one predicting Y given X, and another predicting X given Y. These two tasks can be a different prediction task (e.g.: one a classification, one regression) or the same, and BDNN is a model where these two models share the same weights (between pairwise neurons). It can be trained and optimized in either direction and have bias just for one direction, but weights must be updated at the same time, which means training one side may affect the loss or accuracy of another side. See Figure 2 for a general architecture of BDNN:



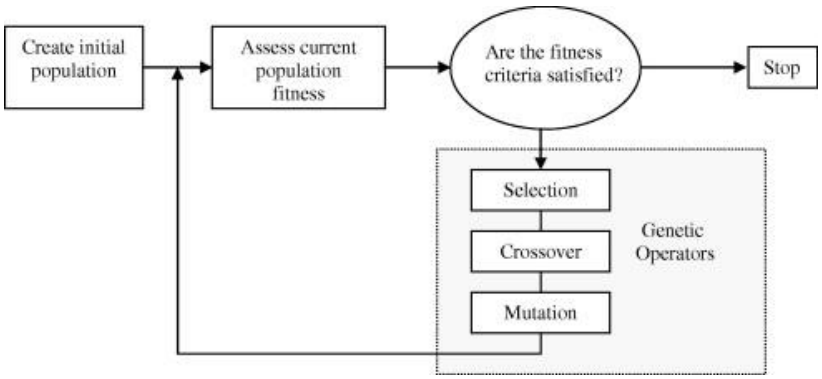


Fig. 1. BDNN sample architecture [9] **Fig. 2.5** General GA architecture [8]

2.1.2 Genetic Algorithm and Hybrid GA-ANN

Our genetic algorithm is built for tuning the hyperparameters of the model. (So populations are hyperparameters) We will follow the basic architecture of Fig 2.5 and inspired from [18], but some modifications would be made, described in 2.4. What we mean by hybrid GA-FFNN / GA-BDNN is that we use GA to find a "best" fit of FFNN/ BDNN. Hence GA and FFNN can be thought of two components of a single model, since ANN itself are also an optimizer in its own sense.

2.2 Dataset

2.2.1 Our Dataset

Our dataset is fetched from that associated with [1]. The original dataset comprises of 400 data entries, each having a total of 8 features. (observer id feature omitted)
Since the entries are statistical summaries of observer's pupillary response over time associated to a video, each observer would only be associated to exactly one video. All observers had responded to all videos exactly once. [1]

The dataset are from 20 Observers (originally 22 participants but some excluded [1]) and 10 videos for each genuine and posed. Their unique id is depicted in the first column. Here list what features (column) we would use for training all models:

Video (V)	Mean (M)	Std (S)	Diff1 (D1)	Diff2 (D2)	PCAd1 (P1)	PCAd2 (P2)	Label (L)
Video id: if it starts with "T" then it has Genuine anger, otherwise posed anger	Mean pupillary size for two eyes over the entire observing process	Standard deviation of pupillary response	Change in left pupillary size after video	Change in right pupillary size after video	Orthogonal linear transformation with the first principal component	Orthogonal linear transformation with the second principal component	Values = 'Genuine' or 'Posed'

2.2.2 Dataset exploration

The dataset has all the values of pupillary size data normalized [1], which means the non-PCA data entries are normalized as well, which means the normalization is already done.
Looking at the Diff statistics, it has already been proved that distributions of this data between genuine anger and posed anger are very different [1], therefore we do not need to preprocess these categories either.

2.2.3 Dataset preparation

We train the feed forward classifier using the original data. However, for the BDNN, we add an extra node to each data entry such that each node is uniformly separated, and no entries contains the same value of node. This is to make sure the data is suitable for bidirectional training and should increase generalization.[7] We first split videos for each category in half, then we shuffle the dataset randomly before training, in our case we used numpy.random.shuffle and we set our seed to 30 for all models (including hybrids) to make a fair comparison. We split the dataset into three parts after shuffling and the size percentage would be in (train: 70%, valid: 10%, test: 20%). We tune for the best model using some evaluation metric on validation set and use the test sets to evaluate on how well a model generalizes to unknown data. We do not want to choose the model based on the test set because we do not want the distributions of large dataset affect the choice of model. Note that this method is modified from [10].
For each set of the partition of the dataset, we have ensured that it contains equal portion of Genuine videos and Posed videos (so validation set would have one video for the two category) and have video sources be unique from other partition to ensure the credibility of model evaluations (independent from choice of videos) and increases model's robustness when training. This method is inspired from leave-one-video-out method. [2]

2.3 Our settings

2.3.1 ANN component

Setting	FFNN	BDNN
Hidden layers	2	
Input -> output (forward dir). Loss function is Cross-entropy: classification (C) / MSE: regression (R)	M,S,D1,D2,P1,P2 -> L (C)	M,S,D1,D2,P1,P2 -> L(C) + extra node (R)
Input -> output (reverse dir)		L + extra node -> M,S,D1,D2,P1,P2 (R)
Activation function (all layers except output)	Relu	
Forward (output) prediction/ reverse input i.e.: L (+ extra node) encodings	L: One-hot. (2 dim). First index = Posed anger (appended by extra node) Largest index of output = predicted class	

For FF and BDNN we would fix two layers of hidden neurons because it would be sufficient for us to learn complex distributions without having too many potential neurons to learn relative to the dataset size [11]. For BDNN, unlike described in paper [7], we will implement separate bias to each direction to allow more parameters to be trained. For both models we will use AdamW as the optimizer. [12] It is known to converge faster than other common optimizers such as (stochastic) gradient descent.

2.3.2 GA component

For the genetic algorithm, we have the following settings. Most of them are directly from that described in [18].

Settings	FFNN	BDNN
Chromosome (bit length for each parameter are hyperparameter for the hybrid model. Determined after model comparison.)	Binary, format in order: (amsgrad = 0, then not use amsgrad in optim) [hidden layer 1 (h1), h2, learning rate (lr), weight decay (wd), amsgrad (1 bit)] Given bits [b0, b1,…, bn] then: H1 / H2 = max(1*b0 + 2*b1 + … + 2^(n)*bn, 1) Lr / wd = max(2^-t *b0 + 2^-(t+1) *b1 + … + 2^-(t+n)* bn , eps). There are separate sets of t, eps for each parameter would be fixed and determined after model comparison.	
Initialization	Uniformly random bit initialization of a number of chromosomes	
Fitness (>=0 and <= 1.0)	Feed the parameters into training a new model, and the "best" validation accuracy among given epochs would be the fitness. See section 2.5	
Recombination	Uniform crossover (one un-ordered pair produce exactly one offspring). [19] random seed is 30. Crossover all pairs of distinct parents.	
Mutation	Random bit flip. For each bit chance is 0.5% (random seed = 30)	
Selection	Stochastic uniform selection [20]. Similar to proportional selection [19] but allows weak models to be selected as well for more exploration of solution space. Think of it as roulette wheel selection but with more than 1 pointer. If there are parents' fitness higher than the best fitness amongst the selected offsprings, then carry over some parents' genes to next generation as well (Elitism) [19].	
Termination criteria	Either all chromosome fitness has diff to minimum fitness <= 0.0005 or a fixed number of generations reached.	

Additionally, to ensure we explore new combinations for each gen, we forced the recombination to not produce a parent by re-rolling offspring, and we force the offspring to mutate until it is different from all other offsprings and parents. To accelerate computation time until it produces a new chromosome the mutation rate would be (mutation rate)*(# attempts). Unlike [18], we do not include features as part of chromosome because model from [1] used all features. (So a good evidence that all features are useful, given its high accuracies)

2.4 Evaluation metrics

For both models, we will optimize model based on best validation accuracies (label prediction). Note that for BDNN, we do not evaluate the loss of predicted extra node because firstly it is meaningless for classification and, secondly the loss is automatically minimized by the training process defined below. **After** choosing "good" model over a wide but sparse range of set of hyperparameters, the model's general performance, extrapolation capabilities and robustness would be evaluated by looking at the test accuracy. Then, we will try to improve the accuracies by using GA hybrids to tune over the range of hyperparameters around that of the top model (and increase parameter domains) and determine whether accuracies can be improved from the original "best" one and see which one produce a better result. That way we can access whether GA algorithm is useful for this task and further evaluate on performance on BDNN vs FFNN on a larger domain of hyperparameters and thus make a stronger conclusion.

2.5 Experimental and implementation pipeline

The steps can be split into two steps: 1. Model comparison and 2. Hybrid-GA-ANN model comparison. ANN are all implemented from Pytorch in conventional way and GA framework is written using numpy and vanilla Python library. Data preprocessing is done in 1. and with pandas and numpy automatically. Generated preprocessed dataset are attached for reference.

There are two separate training method for BDNN and a conventional one for FFNN. All of them would be trained over a limited number of epochs and over re-shuffled batches. Partition is split into batches only when training. Loss is calculated by mean loss over each batch before backpropagation. Average train accuracy across all batches would be calculated for each epoch. Validation and test accuracies are calculated from the average of the whole partition.

The two BDNN models differs from mainly the loss calculation: One method is fundamentally the same to FFNN, except that the loss for each data entry is the sum loss of forward (given features, predict label and extra node) and backward (given label and extra node, regress over features). We call this method BDNN-backforth (BDNN-bf). The other method trains over one direction at a time for a given dataset instead of calculating loss of both directions altogether. It is very similar to method described in [7]. Given number of epochs n, max number of rounds and error tolerance, it initially trains at most n epochs in forward direction (predict labels), then iteratively interleaves direction of training. Direction is switched when the loss is smaller than the loss of previous direction or n epochs are trained. The whole training ends when loss of either direction is smaller than the tolerance or it has been trained for max rounds. (1 round = one forward one backward). We will call this method BDNN-alternate (BDNN-alt).

For losses, cross entropy and MSE are all implemented in built-in functions of Pytorch. Note that the cross entropy function already assumed that the outputs are un-normalized [14] (since it would be passed through a softmax layer when this is executed), so our model do not have a softmax layer.

For each training (for BDNN-alt, initial forward direction only) We have implemented early stopping. Which means given n epochs, if model has trained more than n/2 times and validation accuracy did not go better than all previously recorded accuracies, then training will stop and will select the model weights giving the highest accuracy amongst the epochs trained. For example, if trained 30 epochs has higher accuracy than trained 40 epochs, then it will save the model with 30 epochs trained.

For step 1, we first tune FFNN with a wider range of parameters to estimate where the optimal parameters may lie. The details and the data extracted will be in the appendix. Initially, we chose batch size 1 because it has the most stochastic behavior to the weight updating, but then we realized that there are too many random and unreasonable spike and drops in validation accuracy. Therefore, we slightly increase the batch size to 5. Then we choose a new set of parameters and retrain the feed forward network, and we also use these set of parameters to choose a good model for BDNN with both methods: n_h1_choice = [5,10,20,30,40,50]/n_h2_choice = [5,10,20,30,40,50]/lr_choice = [0.001, 0.0001, 0.00001, 0.005, 0.01, 0.0005]/ weight_decay_choice = [0.05]/ amsgrad_choice = [True, False]/ epochs_choice = [100]/ batch_size_choice = [5]. For BDNN_alt method we fix error tolerance to 1e-4 and max_rounds to 4.

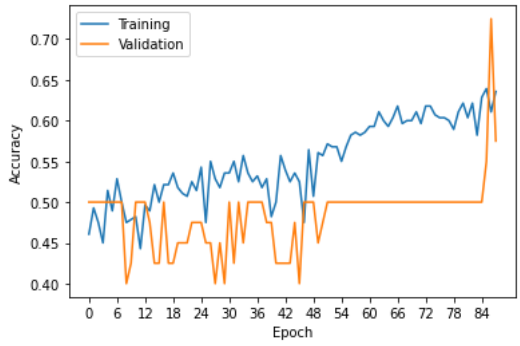
For step 2, from the result of step 1, we will choose the number of bits and range of values included for lr, wd, h1, and h2 so that it is possible to have chromosomes that is equal to or very similar in value to the hyperparameters producing the top validation accuracies for both FFNN and one of BDNN models. We will decide which BDNN to use with hybrid GA given the performance from step 1. Initially we used a smaller range of hyperparameters, offsprings and generations to test the GA hybrids However, for a more accurate comparison we extend range/ increased parameters and this would be the result we use for evaluation.

3 Predictions and Results

3.1 Step 1: Initial Predictions

We think that since BDNN has increased coupling between input and output, it would learn the underlying patterns more tightly to the actual situation. Therefore, BDNN with either loss calculation method should perform considerably better than feed forward. Secondly, both BDNN and feed forward models should perform no better than the model described in [1] because that model is a more complex architecture and it learns both supervised and unsupervised methods, so it should capture information more accurately.

3.2 Step 1: Result



We have moved some of the data to appendix 7.1. We will use only some of the data to discuss.

We have also plotted accuracy changes for the top settings over epochs. The accuracies data corresponding to the figure is the point whose validation accuracy is the highest. For BDNN alternate method, one iteration mean change of direction of training. For general BDNN, 0-1 iteration is the initial training in the forward direction. The epochs are the total epochs counting from first epoch of initial forward train

Fig. 2. FF: {'n_h1': 5, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True}

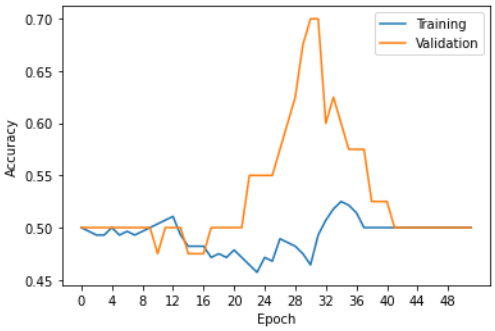
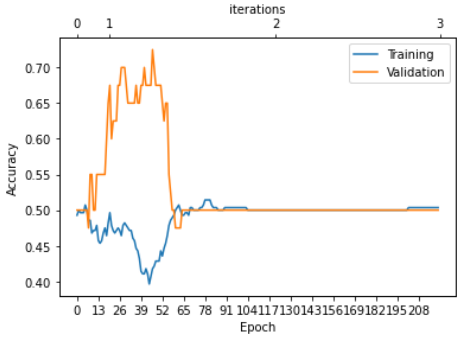


Fig. 3. (left) BDNN_bf: {'n_h1': 10, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': False}

Fig. 4. BDNN_alt: {'n_h1': 10, 'n_h2': 30, 'num_epochs':100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}



3.3 Step 2: Initial experiment of hybrid-ANN

The initial experiment is to get a feel of whether GA can really help improve accuracies given some small range of variables. It showed that they can indeed improve accuracies from the initial agents. The variables used and results are in Appendix 7.2.1

3.4 Step 2: Extended experiment of hybrid ANN

From the maximum possible validation accuracy, we concluded that BDNN_alt performs slightly better than BDNN_bf. (Exact accuracy in Section 4.3). Therefore, we would use GA-FFNN as baseline, and then compare with GA-BDNN_alt. We increased number of generation and size and parents to keep to get a fairer comparison. In the end, we choose the following parameters: n_h1_bindim=5, n_h2_bindim=5, lr_bindim=9, wd_bindim=4, num_initial_agent=30, n_input=n_input, n_classes=n_classes, num_epochs = 60, batch_size = 15, err_tolerance = 1e-1, max_rounds = 1, max_num_gen=22, offspring_size = 30, keep_parents_num=5, t_lr = 9, t_wd = 5, eps_lr = 0.00005, eps_wd = 0.0005. The bit sizes are tailored for BDNN_alt, but it also covers a close enough value to the optimal set found in FFNN Fig 3. For example, since lr is 1e-5 in Fig 5, then we set t_lr = 9 and lr_bindim = 9, so smallest nonzero value is 2^(-18), and biggest value is ~0.0038, which sufficiently covers around 1e-5. We slightly increased the batch size for faster computation, decrased max_rounds and epochs because we found out their effect on improving accuracies is minimal. GA-FFNN has 60 epochs because So for each generation we will need to train minimum 30C2 = 435 ANNs to get their fitness. The running time for each hybrid algorithm is ~8-9 hours. Below figures shows the mean and max (valid) accuracies from each generation. (generation 0 is the randomly initialized generation)

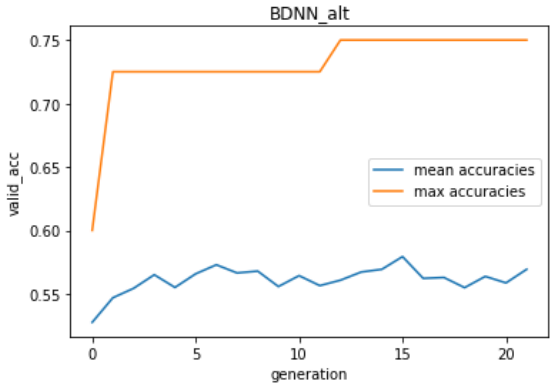
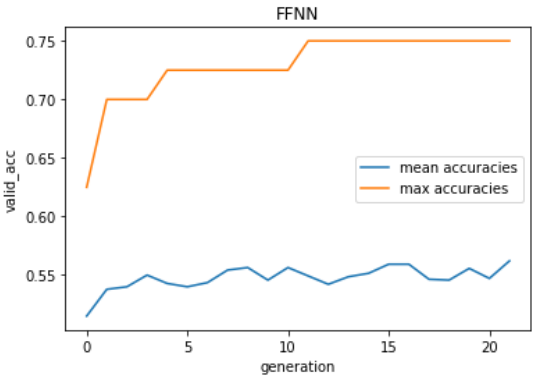


Fig. 6. (left): BDNN_alt

Fig. 7. (right) FFNN



4 Discussion

Now we will analyze on the result to see if BDNN is good for this classification task, and see if our performance is of any credibility, after that we will analyze the performance and usefulness of hybrid GA models on this task.

4.1 Comparison to the original paper

Comparing to the original model that classifies the same labels at 95% [1], even using the overall accuracy (weighted mean of accuracy between training, test, and validation depending on sizes) and accuracy from GA hybrids the original model still surpasses our either models with either methods of training used. This is expected because the model used in [1] is a ensemble of different structures of models and with unsupervised methods as well, which can learn more complex patterns and trends within data.

4.2 Model parameters

From step 1:

FF: {'n_h1': 5, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True}
BDNN-BF: {'n_h1': 10, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': False} BDNN-Alt: {'n_h1': 10, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}

Additionally, for reference here is the parameters for the best performing model found by hybrid GA models. (step 2) Note their accuracy is (slightly) higher than that in step 2, and they have the same accuracy as well:

FF: {'n_h1': 16, 'n_h2': 22, 'num_epochs': 60, 'batch_size:': 15, 'lr'~ 0.00504, 'weight_decay'~ 0.0469, 'amsgrad': False}
BDNN_alt: {'n_h1': 15, 'n_h2': 22, 'num_epochs': 20, 'batch_size:': 15, 'lr'~ 0.0008, 'weight_decay'~ 0.0195, 'amsgrad': False}
Looking at the best model of validation accuracy of all three models, it has number of hidden layer of the first layer less than number of second one. This violates the rule-of-thumb conventions for selecting a neural network model that performs generally well [13]. However, violating rule of thumb does not mean the model is not credible. The random shuffling when splitting partition dataset into batches creates stochastic effect to the update schemes of parameters. It is possible that some parameters are updated so that it fits the model better than other choice of parameters.

The two BDNN models have higher number of hidden neurons for the first layer than that of feed forward. This phenomenon should be normal because BDNN has to learn the weights to cater for input regression as well, also the dimension of output is 3 (output label + extra node) instead of 2 in feed forward task.

Moreover, the learning rate choice of feed forward model is higher than that of BDNN model. The reason why could be because feed forward model has less things to learn, thus the learning process is easier for feed forward, but observing that the result from step 2 indicates that learning rate in BDNN-alt from step 1 may very well be a coincidence for the accuracy to be that high. Higher in accuracy for higher batch size may be indicating that the stochastic effect of small batch size may be instead adversely affecting performance or the regularization effect is too strong for this task.

4.3 Model performance, overfitting and generalization ability (step 1: pure ANN)

Now we are going to evaluate on whether BDNN is in any sense better than Feed forward model for classification task for this dataset and the set of parameters tuned, and how well they generalize to unknown data and robustness.

	FFNN	BDNN-bf	BDNN-alt
Train	0.611	0.725	0.525
Valid	0.493	0.700	0.400
Test	0.418	0.725	0.5

Comparing to uniformly distributed dataset blind guess (50%), BDNN methods fit poorly on the training dataset, but they performed drastically better on validation dataset. The potential reason of why this may be true will be addressed in the next section. Now we will talk about how well the model performs.

Nevertheless, high accuracy in validation set shows that BDNN models' parameters can be tuned to better fit some specific arbitrary dataset. For feed forward methods, the training accuracy performed better than BDNN methods, but it is only little bit better than guessing. Its validation accuracy performed as good as that in BDNN-alternate methods, and BDNN methods do not considerably perform way better than feed forward. Therefore, in terms of the validation accuracy as an evaluation baseline, BDNN-alternate model performs as good as feed forward model, but BDNN-alternate model is suspected to overfit the model because it poorly fits the training data. BDNN performs similarly but slightly worse than the other two, but it also has suspected overfitting problem to BDNN-alternate model. Therefore, one may conclude that in terms of this task, feed forward performed as good as BDNN alternate model, given enough hyperparameter tuning,

(we will further reinforce this claim in section 4.5) but it is very much prone to overfitting, while FFNN is more robust in this sense.

In terms of further generalization ability, we look at the testing accuracy. Only feed forward and BDNN-alternate model has accuracies equivalent to guessing. BDNN-back forth model, however, has lower than 50% performance, meaning that BDNN-back forth model is not really good at generalizing patterns out of training as good as other did. The interesting thing is, for BDNN alternate, it has both validation accuracy and testing accuracy higher than training accuracy. The reason of why it has poor training accuracy can also be explained in the below section.

4.4 Effects of Epochs on performance

Looking at Figure 3, the accuracy of the training model is rising steadily, while there were some minor drops in training accuracy for earlier epochs. However, for validation accuracy, the accuracy keeps fluctuating at largest 50% and at least 40%, while after epoch 50 it keeps staying at 50% until a sudden rise in accuracy for ~80 epochs. This may be because the model is slowly tuning the weights and bias of the model, so that the output prediction gets closer to the desired label, but since accuracy is absolute, "close-enough" data does not mean it is correct. Overall, accuracies of training and validation shows that epochs have a positive effect on tuning feed forward model as expected.

Looking at Figure 4, the validation accuracy rise drastically after 24 epochs and then dropped back to usual 50% accuracy after 40 epochs. The rise and drop further supports the claim that stochastic effect of shuffled batches have a major impact on the performance especially on BDNN. The regularization effect induced by small batch size may be too much to train and induces too much randomness. This claim can also be supported by Figure 5 as well. It can be a reason to explain why there is a big difference in validation and training accuracy.

Looking at Figure 5, when trained forward initially (iteration 0 to 1), there is a drastic increase in validation accuracy, when direction reversed, the validation accuracy continued to rise until after ~20-30 epochs it drops significantly back to 50%. Different from BDNN-back forth model (Figure 4) however, the training accuracy drops considerably when the validation accuracy increased. The drop of accuracy of training data in iteration 1 (reverse training) suggests that when model is regressing over the "input features", the weight change when trying to fit the input data do not have a good effect on fitting labels. This claim can also be supported by iteration 2, the model does not have much change of accuracies after fitting over the inputs.

4.5 Performance of Hybrid models on improving model performance (Step 2)

Studying Figure 6 and 7, we see that the max validation accuracies of both hybrid GA-ANN models reaches 75%, which are higher than the parameters that we tuned. The fact that we extended the range of parameters and finding a "better" one in only a few generations suggest that GA hybrid models when applied to this task, can find a model with reasonable performance easily, and it suggests that our implementation of elitism and selection method have a positive effect on tuning parameters in terms of balancing exploration and exploitation.

This claim can be further supported by observing that mean accuracies both increased comparing to beginning at the end of 22th generation. For the max accuracies to be staying for almost 10 generations, suggesting that this performance may be very close to a local maximal on the decision surface, but whether this is true or not is unknown. Although it stayed at 75% accuracy, it does not mean it cannot increase anymore because we have only tested for 22 generations, which is relatively small comparing to other genetic algorithms. However, trying over an extend range of parameters further supports the claim that both models perform very similarly on this task, and so BDNN-alt do not perform any better than FFNN in this sense, and we can surely conclude that using GA to tune parameters are good for improving model performance on this task for both BDNN-alt and FFNN, regardless of how much it can improve.

Moreover, given that there are 2^24 distinct chromosomes for our case and using grid search may need to start from training relatively bad performing parameters, GA hybrid models definitely is faster to find relatively good parameters, or at least can give us an insight on where the optimal set of hyperparameters can be after "breeding" for some generations instead of random grid search, or at least for this random seed setting (=30).

This hybrid model nevertheless suffers from the downside that we need to give a fixed domain (number of bits and t, eps) for it to optimize, meaning we need to still get to know a big picture of where a "good" value may lie in order to make it perform well. In order to know where the global optimal is, we need to give some mathematical proof (perhaps on tedious calculus), because it is impossible to try over all the non-negative real number for the weights. For this case it is meaningless to do so because of the small size of dataset and may cause extreme overfit even if we have found it. Therefore, we believe that tuning initial models over a large sparse set of parameters is enough credibility before using GA hybrids to "believe" that we have found a local optimal solution.

5 Conclusion and future work

The accuracies from BDNN and feed forward network models are merely a bit better than blind guess on uniform distribution, suggesting that BDNN may not be enough to predict posed and genuine anger.

In terms of how well BDNN performs over classic feed forward model in this context, BDNN-alternate model performs as well as feed forward in terms of validation accuracy, but BDNN generally do not perform drastically better than feed forward given the same sets of parameters choices to tune. Together with having low accuracies on testing set for BDNN-alternate may be suggesting that coupling inputs and outputs do not have much effect on increasing performance and generalization ability in this context. BDNN-back-forth method performs worse than other two models, suggesting that it is not ideal to calculate loss together for each batch.

The phenomenon of coupling inputs and outputs definitely worth investigating further nevertheless. The impact of stochastic effect of batch size and epochs affecting BDNN networks comparing to feed forward neural network need to be investigated further by increasing the overall dataset size and try some other tasks such as making both direction of training the same.

Suggestion proposed above needs to be verified further by testing BDNN models on multiple dataset that has varying noises so as to find out how noise-less input helps learn output.

Our experiments on GA hybrids suggests that it definitely has some potential on (even slightly) improving model performance (in terms of validation accuracy).

The genetic algorithm may need to be experimented more in order to find a better solution. The simplest way would be to increase generation and offspring size to further confirm the effect of our implementation of selection and crossover method. Alternatively, we can choose to try different selection and crossover method and trying different exploration and exploitation strategy on how to accelerate generating “good enough” models. The mentioned downside from 4.5 applies to all GA in nature. Therefore, the easiest way to solve this is to expand our dataset. Neural symbolic reasoning or fuzzy logic may also helps in this task for its dataset size.

<<ABSTRACT>>

6 Reference

1. Chen, L., Gedeon, T., Hossain, M.Z. and Caldwell, S., 2017, November. Are you really angry? Detecting emotion veracity as a proposed tool for interaction. In Proceedings of the 29th Australian Conference on Computer-Human Interaction (pp. 412-416).
2. Hossain, M.Z. and Gedeon, T., 2017, May. Classifying posed and real smiles from observers' peripheral physiology. In Proceedings of the 11th EAI International Conference on Pervasive Computing Technologies for Healthcare (pp. 460-463).
3. Schuster, M. and Paliwal, K.K., 1997. Bidirectional recurrent neural networks. IEEE transactions on Signal Processing, 45(11), pp.2673-2681.
4. Li, G.Q., Qi, X.B., Chan, K.C. and Chen, B., 2017. Deep bidirectional learning machine for predicting NO x emissions and boiler efficiency from a coal-fired boiler. Energy & Fuels, 31(10), pp.11471-11480.
5. Jiaxu, L.I.U., Facial Emotion Classification in SFEW Database Based on Hierarchical CNN with Bidirectionality and Data Augmentation.
6. Pontes-Filho, S. and Liwicki, M., 2019, July. Bidirectional learning for robust neural networks. In 2019 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.
7. Nejad, A.F. and Gedeon, T.D., 1995. Bidirectional neural networks and class prototypes. In Proceedings of ICNN'95-International Conference on Neural Networks (Vol. 3, pp. 1322-1327). IEEE.
8. Clarke, A. and Miles, J.C., 2012. Strategic Fire and Rescue Service decision making using evolutionary algorithms. Advances in Engineering Software, 50, pp.29-36.
9. Gedeon, S1 2021, NN7: Compression, lecture notes, Neural Networks, Deep Learning and Bio-inspired Computing COMP4660, Australian National University
10. Gedeon, S1 2021, NN10a: Cross-validation, lecture notes, Neural Networks, Deep Learning and Bio-inspired Computing COMP4660, Australian National University
11. Gedeon, S1 2021, NN3-Backprop, lecture notes, Neural Networks, Deep Learning and Bio-inspired Computing COMP4660, Australian National University
12. Loshchilov, I. and Hutter, F., 2018. Fixing weight decay regularization in adam.
13. Gedeon, S1 2021, NN5: Hidden units, lecture notes, Neural Networks, Deep Learning and Bio-inspired Computing COMP4660, Australian National University
14. Pytorch.org. 2021. CrossEntropyLoss — PyTorch 1.8.1 documentation. [online] Available at: <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html> [Accessed 30 April 2021].
15. Miller, G.F., Todd, P.M. and Hegde, S.U., 1989, June. Designing Neural Networks using Genetic Algorithms. In ICGA (Vol. 89, pp. 379-384).
16. Tsai, J.T., Chou, J.H. and Liu, T.K., 2006. Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm. IEEE Transactions on Neural Networks, 17(1), pp.69-80.
17. Cantú-Paz, E. and Kamath, C., 2005. 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(5), pp.915-927.
18. Sharma, N. and Gedeon, T., 2013, April. Hybrid genetic algorithms for stress recognition in reading. In European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (pp. 117-128). Springer, Berlin, Heidelberg.
19. Gedeon, S1 2021, EA2: Genetic Algorithms with Technical Details, lecture notes, Neural Networks, Deep Learning and Bio-inspired Computing COMP4660, Australian National University
20. Baker, J.E., 1987, July. Reducing bias and inefficiency in the selection algorithm. In Proceedings of the second international conference on genetic algorithms (Vol. 206, pp. 14-21).

7 Appendix: More neural net parameter tuning

We first tune the parameters of hidden neurons, then tune the parameters of learning weight, weight decay and amsgrad choices.
A: tuning hidden neurons: We choose the smallest batch size to increase its regularization ability. All models are trained in cuda.

n_h1_choice : [5,10,15,…, 70];
n_h2_choice: [5,10,15,…, 70];
epochs: 100 (note: this is the max epochs each model will train itself before early stopping);
lr: 0.001; weight decay: 0.01 (these are the default values in pytorch);
batch size: 1

n_h1': 15, 'n_h2': 10	0.7
n_h1': 70, 'n_h2': 65	0.625
n_h1': 60, 'n_h2': 60	0.625
n_h1': 55, 'n_h2': 30	0.625
n_h1': 55, 'n_h2': 25	0.625
n_h1': 50, 'n_h2': 35	0.625
n_h1': 50, 'n_h2': 25	0.625
n_h1': 45, 'n_h2': 15	0.625
n_h1': 70, 'n_h2': 55	0.6
n_h1': 15, 'n_h2': 15	0.6
n_h1': 70, 'n_h2': 70	0.575

10 settings with different validation accuracy. One with highest accuracy is at the top.

B: tuning weights.

Choices: (chosen n_h1 : 15, n_h2 : 10, rest of weights stays the same unless specified below)

lr = [0.001, 0.0001, 1e-05, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0];

amsgrad = [True, False];

weight_decay = [0.01, 0.001, 0.0001, 1e-05, 0.05, 0.1, 0.5]

lr': 0.0001, 'weight_decay': 0.5, 'amsgrad': True}	0.675
lr': 0.0001, 'weight_decay': 1e-05, 'amsgrad': False}	0.6
lr': 0.005, 'weight_decay': 1e-05, 'amsgrad': True}	0.6
lr': 0.0001, 'weight_decay': 0.01, 'amsgrad': True}	0.575
lr': 0.0001, 'weight_decay': 0.1, 'amsgrad': True}	0.575
lr': 0.001, 'weight_decay': 0.001, 'amsgrad': False}	0.55
lr': 0.001, 'weight_decay': 0.01, 'amsgrad': True}	0.55
lr': 0.001, 'weight_decay': 0.0001, 'amsgrad': True}	0.55
lr': 0.001, 'weight_decay': 0.05, 'amsgrad': False}	0.55
lr': 0.001, 'weight_decay': 0.5, 'amsgrad': True}	0.55

Top 10 settings.

7.1 Actual training: From this point in this paper onwards we are tuning the parameters using the params defined in 2.5

Top 20 settings for feed forward:

choice	v_accs	Accs (train, valid, test)
{'n_h1': 5, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True}	0.725	(0.6107142567634583, 0.7250000238418579, 0.5250000357627869)
{'n_h1': 20, 'n_h2': 40, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False}	0.65	(0.4821428656578064, 0.6500000357627869, 0.5)
{'n_h1': 20, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': True}	0.625	(0.4571429491043091, 0.625, 0.42500001192092896)
{'n_h1': 20, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': False}	0.625	(0.4714285433292389, 0.625, 0.4375)
{'n_h1': 20, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True}	0.625	(0.6642856597900391, 0.625, 0.5)
{'n_h1': 20, 'n_h2': 50, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False}	0.625	(0.5178571343421936, 0.625, 0.48750001192092896)
{'n_h1': 40, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': False}	0.625	(0.44999995827674866, 0.625, 0.45000001788139343)
{'n_h1': 5, 'n_h2': 40, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True}	0.6	(0.6249998807907104, 0.6000000238418579, 0.42500001192092896)
{'n_h1': 10, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': True}	0.6	(0.5142858028411865, 0.6000000238418579, 0.4375)
{'n_h1': 20, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': True}	0.6	(0.535714328289032, 0.6000000238418579, 0.4375)
{'n_h1': 40, 'n_h2': 50, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': True}	0.6	(0.48928579688072205, 0.6000000238418579, 0.4749999940395355)
{'n_h1': 5, 'n_h2': 40, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': True}	0.575	(0.4750000238418579, 0.574999988079071, 0.4000000059604645)
{'n_h1': 5, 'n_h2': 40, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': False}	0.575	(0.5035714507102966, 0.574999988079071, 0.5)
{'n_h1': 5, 'n_h2': 40, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': True}	0.575	(0.46071434020996094, 0.574999988079071, 0.5625)
{'n_h1': 5, 'n_h2': 40, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': False}	0.575	(0.4571428894996643, 0.574999988079071, 0.574999988079071)
{'n_h1': 5, 'n_h2': 50, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False}	0.575	(0.48928576707839966, 0.574999988079071, 0.4375)
{'n_h1': 5, 'n_h2': 50, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': True}	0.575	(0.5142856836318970, 0.574999988079071, 0.5250000357627869)
{'n_h1': 10, 'n_h2': 50, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': True}	0.575	(0.5, 0.574999988079071, 0.5875000357627869)
{'n_h1': 20, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': True}	0.575	(0.47857147455215454, 0.574999988079071, 0.625)
{'n_h1': 20, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': True}	0.575	(0.5035714507102966, 0.574999988079071, 0.48750001192092896)

Top 20 for BDNN—back and forth method:

choice	v_accs	accs
{'n_h1': 10, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': False}	0.7	(0.49285727739334106, 0.699999988079071, 0.4000000059604645)
{'n_h1': 30, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': True}	0.675	(0.43571436405181885, 0.675000011920929, 0.4000000059604645)
{'n_h1': 5, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': True}	0.65	(0.4750000834465027, 0.6500000357627869, 0.550000011920929)
{'n_h1': 10, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': True}	0.65	(0.4607143700122833, 0.6500000357627869, 0.5625)
{'n_h1': 10, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': False}	0.65	(0.4571428596973419, 0.6500000357627869, 0.550000011920929)
{'n_h1': 20, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': True}	0.65	(0.4357143044471741, 0.6500000357627869, 0.5625)
{'n_h1': 50, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': True}	0.65	(0.49642857909202576, 0.6500000357627869, 0.512499988079071)
{'n_h1': 5, 'n_h2': 50, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': False}	0.625	(0.4928571283817291, 0.625, 0.6000000238418579)
{'n_h1': 10, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': True}	0.625	(0.4607143998146057, 0.625, 0.5875000357627869)
{'n_h1': 10, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': True}	0.625	(0.4285714626312256, 0.625, 0.550000011920929)
{'n_h1': 10, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': False}	0.625	(0.48928573727607727, 0.625, 0.4375)
{'n_h1': 20, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': True}	0.625	(0.4500001072883606, 0.625, 0.574999988079071)
{'n_h1': 50, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': True}	0.625	(0.45357146859169006, 0.625, 0.550000011920929)
{'n_h1': 50, 'n_h2': 40, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': True}	0.625	(0.49642857909202576, 0.625, 0.574999988079071)
{'n_h1': 5, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': False}	0.6	(0.4857143759727478, 0.6000000238418579, 0.5875000357627869)
{'n_h1': 5, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True}	0.6	(0.47857144474983215, 0.6000000238418579, 0.625)
{'n_h1': 5, 'n_h2': 50, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': True}	0.6	(0.4500000476837158, 0.6000000238418579, 0.625)
{'n_h1': 5, 'n_h2': 50, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': True}	0.6	(0.5000001788139343, 0.6000000238418579, 0.574999988079071)
{'n_h1': 10, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0005, 'weight_decay': 0.05, 'amsgrad': False}	0.6	(0.48928573727607727, 0.6000000238418579, 0.5625)
{'n_h1': 10, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False}	0.6	(0.5035713911056519, 0.6000000238418579, 0.550000011920929)

Top 20 settings for BDNN – alternate method:

choice	v_accs	accs
{'n_h1': 10, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.725	(0.4178571105003357, 0.7250000238418579, 0.5)
{'n_h1': 10, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.7	(0.4392857849597931, 0.699999988079071, 0.5)
{'n_h1': 10, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.675	(0.4892857074737549, 0.675000011920929, 0.42500001192092896)
{'n_h1': 40, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True, 'err_tolerance': 0.1, 'max_rounds': 20}	0.675	(0.6178571581840515, 0.675000011920929, 0.6000000238418579)
{'n_h1': 10, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.65	(0.4571428894996643, 0.6500000357627869, 0.5250000357627869)
{'n_h1': 10, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': True, 'err_tolerance': 0.1, 'max_rounds': 20}	0.65	(0.514285683631897, 0.6500000357627869, 0.48750001192092896)
{'n_h1': 20, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': True, 'err_tolerance': 0.1, 'max_rounds': 20}	0.65	(0.5071429014205933, 0.6500000357627869, 0.5)
{'n_h1': 20, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.65	(0.4642857313156128, 0.6500000357627869, 0.5)
{'n_h1': 20, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True, 'err_tolerance': 0.1, 'max_rounds': 20}	0.65	(0.6285713911056519, 0.6500000357627869, 0.75)
{'n_h1': 30, 'n_h2': 30, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': True, 'err_tolerance': 0.1, 'max_rounds': 20}	0.65	(0.5, 0.6500000357627869, 0.375)
{'n_h1': 30, 'n_h2': 40, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True, 'err_tolerance': 0.1, 'max_rounds': 20}	0.65	(0.625, 0.6500000357627869, 0.3499999940395355)
{'n_h1': 50, 'n_h2': 50, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.01, 'weight_decay': 0.05, 'amsgrad': True, 'err_tolerance': 0.1, 'max_rounds': 20}	0.65	(0.4535714089870453, 0.6500000357627869, 0.5)
{'n_h1': 5, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.625	(0.5071429014205933, 0.625, 0.4124999940395355)
{'n_h1': 5, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.625	(0.44999995827674866, 0.625, 0.5)
{'n_h1': 10, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.001, 'weight_decay': 0.05, 'amsgrad': True, 'err_tolerance': 0.1, 'max_rounds': 20}	0.625	(0.4357142448425293, 0.625, 0.45000001788139343)
{'n_h1': 10, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.005, 'weight_decay': 0.05, 'amsgrad': True, 'err_tolerance': 0.1, 'max_rounds': 20}	0.625	(0.485714316368103, 0.625, 0.36250001192092896)
{'n_h1': 10, 'n_h2': 40, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.625	(0.5571427345275879, 0.625, 0.4124999940395355)
{'n_h1': 20, 'n_h2': 10, 'num_epochs': 100, 'batch_size:': 5, 'lr': 1e-05, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.625	(0.5107143521308899, 0.625, 0.5)
{'n_h1': 20, 'n_h2': 20, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.625	(0.44285720586776733, 0.625, 0.512499988079071)
{'n_h1': 30, 'n_h2': 5, 'num_epochs': 100, 'batch_size:': 5, 'lr': 0.0001, 'weight_decay': 0.05, 'amsgrad': False, 'err_tolerance': 0.1, 'max_rounds': 20}	0.625	(0.4392857551574707, 0.625, 0.4749999940395355)

7.2 Summaries of results of hybrid GA-ANN.

Initially we have made some experimental parameters to see how well it work. We have tested them over BDNN_alt, FFNN and FFNN. The results may show that BDNN_alt performs better than FFNN, but to confirm, we extend the parameters described in the results section.

Settings:

100 epochs, 10 offsprings and initial chromosomes, keep max 3 parents, max_rounds = 3, hidden_bits = 4, lr_bits = 5, wd_bits = 4 (So for each generation, at most have 13 chromosomes), 10 generations, (BDNN_alt: err_tol: 1e-1, max_rounds = 3)

7.2.1 Results (initial toy experiment)