Evolutionary Neural Networks on Predicting Preferred Personality for Friend Recommendation in Social Networks - A Comparative Study

Nafis Neehal

Department of CSE, Bangladesh University of Engineering and Technology
Dhaka, Bangladesh.

Abstract

Neural Networks are one of the most powerful classification models till date. But, it suffers from several problems like - getting stuck in the local minima, or static structure selection, or slow convergence of synaptic weights, etc. Development of Hybrid Evolutionary Neural Networks using Evolutionary Algorithms and Swarm Algorithms is becoming more and more popular to avoid these kind of problems. In this paper, I have developed 6 different Evolutionary Neural Networks and conducted a comparative study on their performance on a particular learning problem. I've also proposed a novel architecture of a evolutionary neural network which shows potential to perform well in complex and large learning problems.

Keywords: Evolutionary Neural Networks, Neuroevolution, PSO, Evolution Strategies, Artificial Neural Networks

1. Introduction

The use of evolutionary algorithms (EAs) to aid in artificial neural network (ANNs) learning is nothing new. It has been a popular approach for a long time. But mostly, these algorithms have been used to cover for the shortcomings of backpropagation, which is - it often gets stuck in the local minima [1]. In this

*Corresponding author
Email address: `nafisneehal@iut-dhaka.edu` (Nafis Neehal)

study that I've conducted, I've developed 6 different evolutionary neural networks, each have different combinations of Particle Swarm Optimization (PSO), Evolution Strategies (ES) and Backpropagation (BP) to depict the structure of the network (choose number of hidden layer nodes and activation function used in hidden layer), initialize the weights of the network and optimize it.

1.1. motivation

While studying neural networks, I've always felt that it is somewhat being forced to underperform despite having the potential to do more. I believe, the fixed structure of neural network, that we determine beforehand, and the random initial weights that we use when instantiating a neural network object are the main two reasons for neural nets underperforming. And, to solve this two problems altogether using a single neural network was my main motivation.

1.2. Research Contribution

My research contributions are as follows -

- Developing 6 different hybrid evolutionary neural networks, among which 4 are complete new as far as I know. Because, ES has not been used (maybe in one or two, although I couldn't find any) in neural networks, and also I didn't find any neural networks which solely had PSO for weight optimization.
- Comparing the performances of these 6 networks on a specific learning problem, on which much work not have been done also.
- Pointing out the limitations of each of the network, by developing which the network's performance can be increased to an excellent level.
- Proposed a novel neural network (ESPSB_Net) which can select its structure by itself using ES, use PSO to curate initial weights and then optimize those weights using typical backpropagation. I believe, this neural network enforces a balanced application of exploration (PSO, ES) and exploitation

(Backpropagation) which shows a very good potential of becoming a powerful self-evolving neural network.

1.3. Literature Review

As I stated earlier, uses of Evolutionary Algorithms and Swarm Algorithms to optimize its structure and weights is nothing new (but implementing both in the same network has not been done yet which is my major contribution). Palmes et. al. [1] proposed a mutation based genetic neural network to address the problem of backpropagation getting stuck in local optima. Lam et. al. [2] presented a neural network where they tuned the structure and parameters of neural network using an improved version of GA. Settles et. al. [3] presented a Hybrid GA/PSO Neural Network where they used both GA and PSO for weight optimization, but not for defining structure.

Mohammad et. al. [4] proposed a neural network, similar to PB_Net that I've developed. They had initialized a neural network with weights optimized by PSO for a few rounds, and then used those weights to initialize the final network and optimize those weights later on using backpropagation. Juang et.al. [5] proposed an evolutionary recurrent neural network, where he used and hybrid of PSO and GA for structure optimization only. Garro et. al. [6] showed a comparison of performance between backpropagation and PSO in optimizing synaptic weights of an ANN and showerd that PSO performed better most of the time than backpropagation. Meissner et. al. [7] used an Optimized version of PSO for training neural networks and achieved a time decrease by a factor of four and two while comparing with other methods. Gudise et. al. [8] also compared PSO and Backpropagation for training neural networks and showed that PSO converges faster than Backpropagation.

2. Methodology

Evolutionary Neural Networks have become a cutting edge research tool. Many well-known research groups have started developing optimized evolutionary neural networks and incorporating them in their research works where the hyper-parameter space is huge. Particle Swarm Optimization is a Stochastic Optimization Method which recently has become very popular for fusing with Artificial Neural Networks. Also, among typical Evolutionary Algorithms (where re-sampling occurs), Genetic Algorithm is the most popular one to use with Neural Networks.

2.1. Problem Definition

In this work, I've used a fusion of (μ, λ) Evolution Strategies, Particle Swarm Optimization and Backpropagation in different combinations for building 6 different evolutionary neural networks and assess their performance on a particular learning problem.

2.2. The Learning Problem Description

The learning problem I've used here is - "Prediction of Preferred Personality for Friend Recommendation in Social Networks". Normally, social networks, like Facebook, recommends friends to a user based on number of mutual friends, match in Geo-location (same office, same educational institution etc.), common interests etc. But, in real life, we typically become friends with the person who has similar personality like us or has the personality that we like. So, being motivated from that, in my B.Sc thesis I started to work on developing a Friend Recommendation Framework that will recommend friends to users based on personality preference. In my earlier thesis work, I applied a plain vanilla form of Artificial Neural Network. And in this work, I applied the different evolutionary neural network models on this particular learning problem and assessed their performance.

The Big Five personality traits, also known as the five-factor model (FFM), and the OCEAN model, gives 5 different categories for human personalities, namely -

- Openness to experience (inventive/curious vs. consistent/cautious)
- Conscientiousness (efficient/organized vs. easy-going/careless)
- Extraversion (outgoing/energetic vs. solitary/reserved)
- Agreeableness (friendly/compassionate vs. challenging/detached)
- Neuroticism (sensitive/nervous vs. secure/confident)

For personality assessment, a 50 Question Survey form was distributed among surveyees. From the answer to the questions of this survey, a surveyee's personality score in each of the 5 personality categories can be derived. These 5 score values in 5 different personality categories are used as 5 different feature values for our learning problem. This looks like a float vector like [2.3, 4.4, 3.2, 4.1, 2.4].

In the survey, it was also questioned that what type of personality among the five would the user choose to be friends with. Users could choose one or more than one personalities of their preference for friendship. This is a size 5 boolean vector like [0, 1, 0, 0, 1].

The survey was done by, David Stillwell of The Psychometrics Centre, University of Cambridge. [9]. But, unfortunately, this dataset is now closed after the Cambridge Analytical event [10]. However, I started working on the dataset way before they closed it.

2.3. Dataset Description

The dataset contains 9917 samples and 11 columns. The first column depicts the sample number (a facebook user), column [2-6] depicts the float vector of feature values (five personality scores), and column [7-11] depicts the boolean vector for friendship preference.

2.4. Github Repository for Code and Dataset

This is the Github Repository link -

https://github.com/Nafis-Neehal-IUT/Neuroevolution-BUET

It contains the following files -

- Iris Neural Network.ipynb - A draft version of (5,20) ES on Neural Network [not part of the main project, just saving it here for any future reference]
- Main.ipynb - The main notebook, contains all the 6 Evolutionary Neural Network implementations that I build from scratch
- NeuralNetwork.py - The NeuralNetwork module I wrote from scratch.
- PSO.py - The Particle Swarm Optimizaiton module I wrote from scratch.
- Readme.md - Contains installation guidelines
- input.csv - The Personality Dataset (9917 x 11)

[If you face difficulty to Render the Main.ipynb notebook in Github, please refer to this link in NBViewer - `http://bit.ly/nbviewerneuro`]

2.5. Overview of the Approach

A brief outline of my approach -

- At first, I imported the data and all other necessary modules and packages into my program
- Then I did the necessary data preprocessing. I converted all the data which were "object" type into "float" type. Also, omitted the first column containing the sample number, and splitted the remaining dataset (dimension 9917 x 10) into train, validation and test set. The training set contained 8000 samples, validation set contained 1000 samples and test set contained 917 samples. All these samples were randomly taken from the main dataset. Also wrote a function for threshold mapping. This is because, Sigmoid function always compresses the output between [0,1]. So, if the final layer output of Neural Network is greater than or equal to 0.5 in any of the output node, then I converted it to 1, and if it was less than 0.5 then converted it to 0.

- Then, I designed the outline for 6 Neural Network Models that I was going to test -

    (1) **N_Net:** Plain Vanilla Neural Network with randomly initialized weights which will be optimized using Backpropagation only
    (2) **NP_Net:** Neural Network with randomly initialized weights which will be optimized using PSO only, No Backpropagation
    (3) **PB_Net:** Neural Network with initial weights optimized by PSO, and then further optimized using Backpropagation
    (4) **ES_Net:** Neural Network with structure selected by (5,20) Evolution Strategies and randomly initialized weights which will be optimized using Backpropagation
    (5) **ESPS_Net:** Neural Network with structure selected by (5,20) Evolution Strategies and randomly initialized weights which will be optimed using PSO only, No Backpropagation
    (6) **ESPSB_Net:** Neural Network with with structure selected by (5,20) Evolution Strategies and initial weights optimized by PSO (not randomly initialized) and finally updated using Backpropagation

  I applied each of those nets on the personality dataset and plotted training and validation loss and also calculated precision, recall, f-measure and accuracy acquired by using those nets on test dataset.

3. Results

Detailed Result of my experiments are described in the following sections.

3.1. Experimental Setup

My experimental setup details are given below (Basic Neural Network's Structure in Fig. 1) -

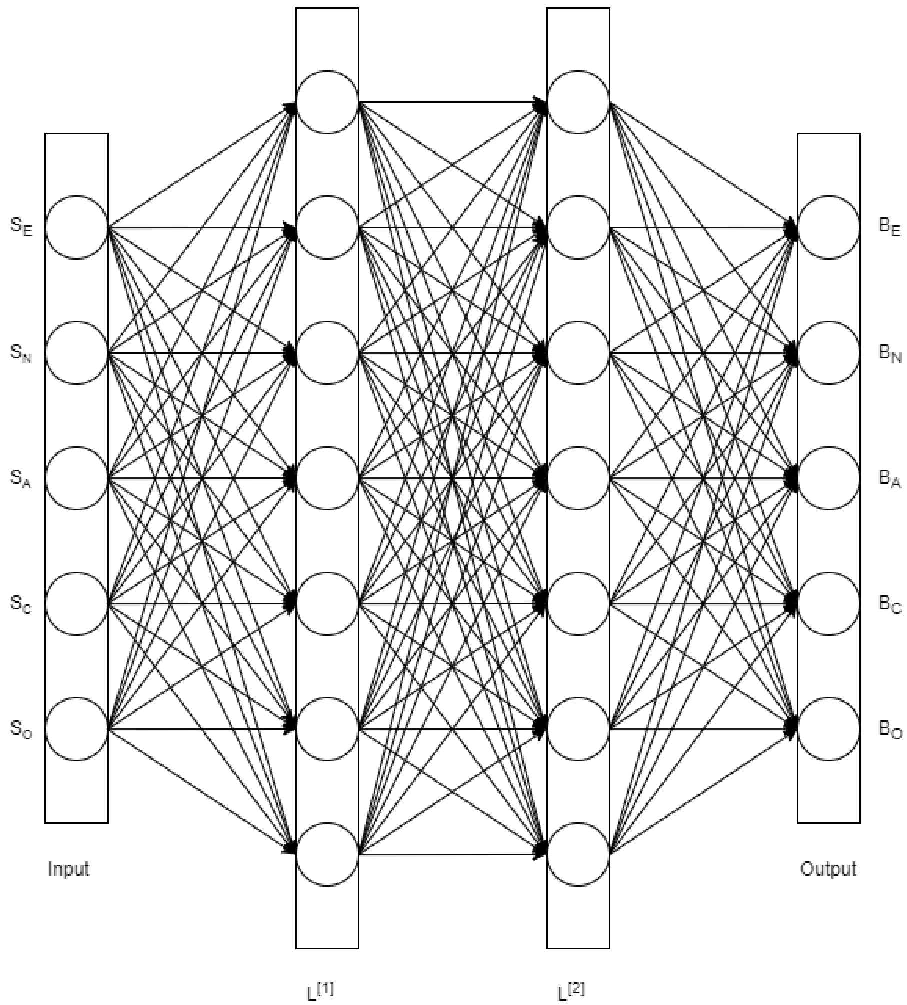


Figure 1: Basic Structure of My Neural Network

3.1.1. PC Configuration

- RAM: 8GB
- OS: 64 Bit (Windows 8.1)
- Processor: Core i5 (5th Gen)
- Processor Speed: 2.20 GHz (No Overclocking)

3.1.2. Neural Network, PSO and ES Configurations

- **N_Net:** 5 Input Nodes (float vector), 5 Output Nodes (boolean vector), 2 hidden layers, 8 hidden nodes each, Output activation: Sigmoid, Hidden Activation: ReLU. PSO and ES not used. Neural Net Epoch: 2000.
- **NP_Net:** 5 Input Nodes (float vector), 5 Output Nodes (boolean vector), 2 hidden layers, 16 hidden nodes each, Output activation: Sigmoid, Hidden Activation: ReLU. PSO used for 1000 Generations, 50 particles each.
- **PB_Net:** 5 Input Nodes (float vector), 5 Output Nodes (boolean vector), 2 hidden layers, 8 hidden nodes each, Output activation: Sigmoid, Hidden Activation: ReLU. PSO used for 100 Generations 30 Particles, each. Neural Net Epoch: 2000.
- **ES_Net:** 5 Input Nodes (float vector), 5 Output Nodes (boolean vector), 2 hidden layers, 17 hidden nodes each (determined by ES), Output activation: Sigmoid, Hidden Activation: Tanh (determined by ES). (5,20) ES used for 30 generations, PSO not used. Neural Net Epoch: 2000.
- **ESPS_Net:** 5 Input Nodes (float vector), 5 Output Nodes (boolean vector), 2 hidden layers, 17 hidden nodes each (determined by ES), Output activation: Sigmoid, Hidden Activation: Tanh (determined by ES). (5,20) ES used for 30 generations.PSO used for 1000 Generations, 50 particles each.
- **ESPSB_Net:** 5 Input Nodes (float vector), 5 Output Nodes (boolean vector), 2 hidden layers, 17 hidden nodes each (determined by ES), Output

activation: Sigmoid, Hidden Activation: Tanh (determined by ES). (5,20) ES used for 30 generations. PSO used for 100 Generations, 20 particles each. Neural Net Epoch: 2000.

3.2. Experimental Results

Experimental Results of my models are given below -

3.2.1. Training-Validation Loss VS Epoch

Figure 1 contains Training-Validation Loss for all the 6 Neural Nets. Among them NP_Net and ESPS_Net - in which only PSO was used to optimize the neural network's weights are trained on 1000 Epochs. The rest four neural networks were trained on 2000 epochs each.

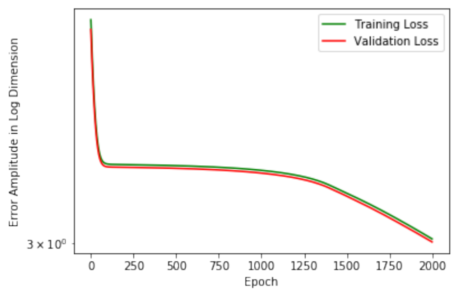
3.2.2. Performance on Test Dataset

Figure 2 contains all the performance measurements data of all 6 nets on the test dataset. All of these values are the average values for all 917 test samples.

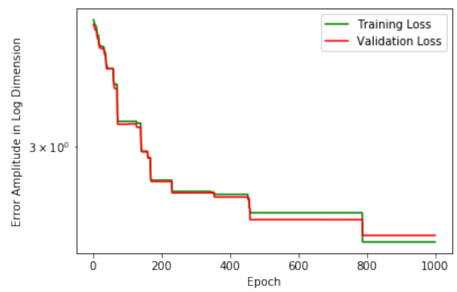
For testing, at first output for all the 917 values were predicted using a single forward pass of the neural network. And then, those predictions were converted to 0 or 1 based on - if any of the 5 output nodes emitted a value less than 0.5, it was converted to 0 and if it emitted a value greater than or equal to 0.5 then it was converted to 1. It is noteworthy, Sigmoid function was used at the output layer which always squeezes outputs to a float number between [0,1]

3.3. Discussions

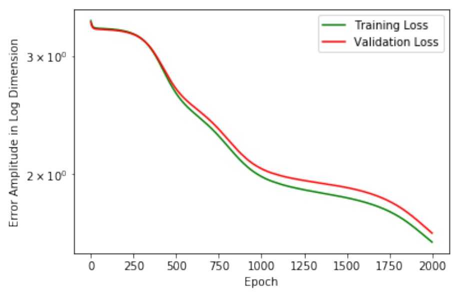
N_Net. N_Net was the typical plain vanilla form of Neural Network. I trained it for 2000 epochs on training set (8000 samples) without using any kind of added parameters like momentum, weight decreasing, dropout, regularization etc. for avoiding overfitting, or more optimization. From the training-validation loss result, it can be seen, that after around 120 epochs, the loss decrease graph got flattend for around 1100 epochs, and then slowly started going downwards again. It maybe the case that it was stuck in a local optima for all those epochs and couldn't find the right weights because I set the learning rate to 0.01. It



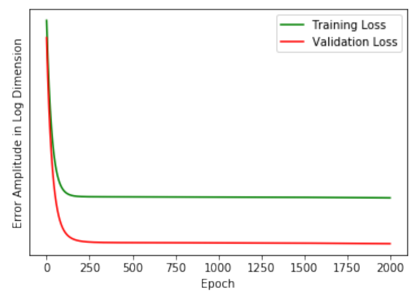
(a) N_Net



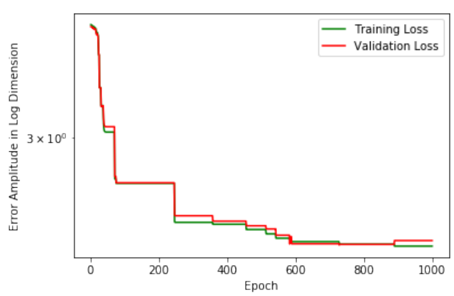
(b) NP_Net



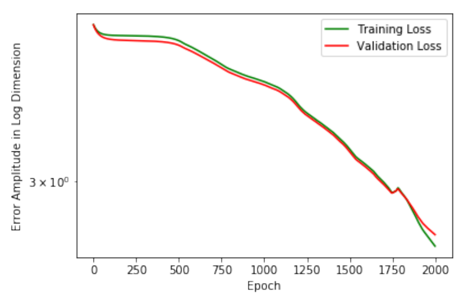
(c) PB_Net



(d) ES_Net

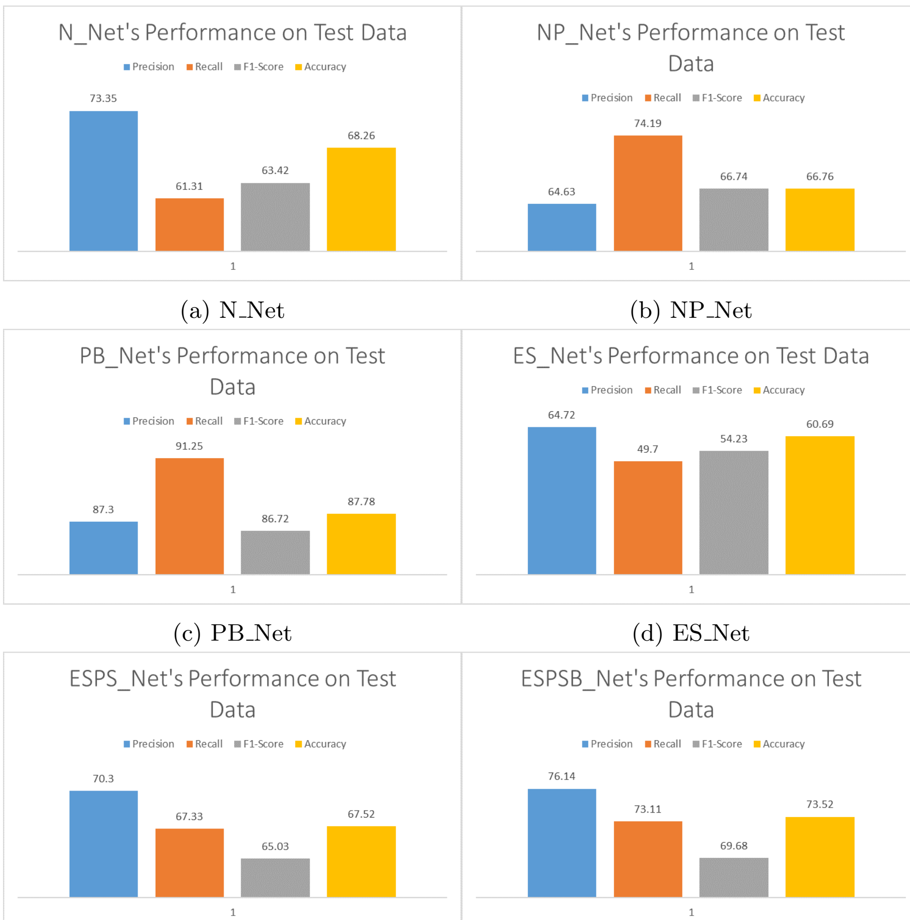


(e) ESPS_Net



(f) ESPSB_Net

Figure 2: Training-Validation Loss VS Epoch Plot for all 6 Nets



(a) N_Net (b) NP_Net

(c) PB_Net (d) ES_Net

(e) ESPS_Net (f) ESPSB_Net

Figure 3: Performance of All 6 Nets on Test Data

can be tuned a little bit avoid this flattening. Although, this network only shows a very small sign of overfitting (huge dataset helped in this regard), but as it flatteneded after 120 epochs for quite a long time, I think I could have adopted the early stopping method here to avoid this overfitting because the later decrease in weights were somewhat negligible. Its performance on the test dataset was also not that much satisfactory (depicts the overfitting) as I didn't use any extra optimization techniques. Adopting those will certainly increase the overall performance on test data and reduce overfitting.

NP_Net. In NP_Net, I didn't use any kind of backpropagation at all and solely depended on PSO for weight optimization. From the loss figure, it is prominent, that there are frequent flattening for certain period of epochs in between. As, PSO is a global optimization algorithm and enforces exploration, I guess, hence the periodic irregular flattened areas. It can also be seen, that upto 200 generations, the weight decrease was significant, but after that, the weight decrease got somewhat slowed down. So, here also, I could have adopted early stopping after around the first 200 epochs. It also shows some sign of overfitting (depicted from test performance), which could be avoided using regularization or dropout.

PB_Net. PB_net, depicted the best performance on test data in terms of all the performance metrics. In PB_Net, I initialized the neural network with optimized by PSO for a specific number of rounds, and then optimized those weights using typical backpropagation. Most probably, due to the initial use of curated weights this network performed the best. Also, from its training-validation loss graph, it shows a general trend of going downwards without any significant flattening. However, the rate of decrease for validation data was slight lower than training data. But, as validation graph is generally trending downwards, so it is safe to say that the model is not overfitted (this is also because the model is performing equally well on test data).

ES_Net. The ES_Net's performance was moderate. From training-validation loss grpah, it can be seen that weight decrease stopped strictly after around 130 epochs. This early convergence can be due to the earlier structure selection of neural network by ES. As the model does not perform very well on the test data, then either this model is overfitted or it is stuck in local optima. The second one has better chance of happening, because for the model to be overfitted, the training accuracy needs to be very good and continuously decreasing or at least showing a downward trend, but here it flattened. So, it is very likely, that the model got stuck in a local optima which is very typical for gradient based optimizations.

Chromosome and Gene Details: In (5,20) ES, each chromosome will be a vector like this [4, [1, 0, 0]]. Here, the first element will be an integer between 4-32 inclusive. This will depict the number of nodes in each of the hidden layer. And the second element is a boolean vector which can be mapped - [1, 0, 0] = ReLU, [0, 1, 0] = tanh and [0, 0, 1] = Sigmoid.

Selection for Mutation: Coin toss will be done for selecting whether any gene of the chromosome will be mutated or not. For gene A (number of nodes in hidden layer), selection threshold is set to 30% and for Gene B (boolean vector for activation function selection in hidden layer) is set to 50

Mutation Operators: Mutation operator for A is selecting a random number within [4, 32]. Used this instead of random walk to enforce exploration a bit to avoid possible local minima. Mutation operator for Gene B is simple bitwise rightshift. For instance, [1,0,0] -¿ [0,1,0]; [0,1,0]-¿[0,0,1] and [0,0,1]-¿[1,0,0]

Taboo List: I've also maintained a Taboo list of λ (population size = 20) size. Whenever a new child is created, it is first checked whether it is in the taboo list. If yes, then another new child is created. If not, then the first element is popped from the taboo list and this new child is appended at the last of the taboo list besides being added as the new member of the main population. So, each new candidate will get a fair lifetime of 19 new not-present-in-taboo birth.

Fitness Assesment: is done using a simple forward pass without any backward pass. For each candidate structure, a neural network with random weight is created. Then a single forward pass is made and then cost is calculated. This is the measurement of fitness. Lower the cost means fitter the candidate.

ESPS_Net. ESPS_Net performed slightly better than its predecessor ES_Net. Although, it had the same structure, the training and validation loss decrease trend of ESPS_Net is generally downwards which shows that it is probably not stuck any local optima (also, as PSO was used for weight optimization, not backprop, so very few chance of getting stuck in local optima). However, it still didn't perform well on the test set which depicts the sign of being slightly overfitted. This can be omitted by adopting early stopping after around 250 epochs (from train-validation loss graph) and use regularization or dropout.

ESPSB_Net. ESPSB_Net was my target neural network. It performed the second best on test dataset in terms of performance metrices. It also has a very good training-validation loss decreasing trend without any flattening or discontinuity. Performance on test data still has room for improvement as it still shows very small sign of overfitting which can be improved by applying regularization or dropout (early stopping will not work well on this model - depicted from training-validation loss graph).

4. Limitations of the Work

- My Laptop is not that much high-end, so couldn't perform all the simulations.
- Didn't take any measurement for overfitting reduction like - Regularization, Dropout, Early Stopping etc.
- Didn't incorporate runtime measurement in performance assessment. Need to work on that aspect as well.
- Also, I need to refactor my code. I've directly implemented ES which I shouldn't have done. Instead, I should have implemented the ES code as a class, like I did with NeuralNetwork.py and PSO.py.

5. Conclusion

This study depicts the potential of my desired network, which was the last one ESPSB_Net. Although it performed second best, but I believe it's performance can be enhanced by taking measures to prevent overfitting, as well as tune the hyper-parameters a little bit. Further performance assessment of this network on benchmark optimization problems like Rastrigin Function, Matyas Function, Ackley Function, Peaks Function etc can be added. Also, instead of PSO, other swarm algorithms like Firefly algorithm can be used here, as well as other EAs like Genetic Algorithm can be experimented with too. Overall, building these hybrid networks and applying them on different learning problems can be a new research domain.

References

[1] P. Palmes, T. Hayasaka, S. Usui, Mutation-Based Genetic Neural Network, IEEE Transactions on Neural Networks 16 (3) (2005) 587–600. doi:10.1109/TNN.2005.844858.
URL http://www.ncbi.nlm.nih.gov/pubmed/15940989http://ieeexplore.ieee.org/document/1427764/

[2] F. Leung, H. Lam, S. Ling, P. Tam, Tuning of the structure and parameters of a neural network using an improved genetic algorithm, IEEE Transactions on Neural Networks 14 (1) (2003) 79–88. doi:10.1109/TNN.2002.804317.
URL http://ieeexplore.ieee.org/document/1176129/

[3] M. Settles, T. Soule, Breeding swarms, in: Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05, ACM Press, New York, New York, USA, 2005, p. 161. doi:10.1145/1068009.1068035.
URL http://portal.acm.org/citation.cfm?doid=1068009.1068035

[4] E. T. Mohamad, D. J. Armaghani, E. Momeni, A. H. Yazdavar, M. Ebrahimi, Rock strength estimation: a PSO-based BP approach, Neu-

ral Computing and Applications 30 (5) (2018) 1635–1646. doi:10.1007/s00521-016-2728-3.
URL http://link.springer.com/10.1007/s00521-016-2728-3

[5] C.-F. Juang, A Hybrid of Genetic Algorithm and Particle Swarm Optimization for Recurrent Network Design, IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics) 34 (2) (2004) 997–1006. doi:10.1109/TSMCB.2003.818557.
URL http://ieeexplore.ieee.org/document/1275532/

[6] B. A. Impr. Bourg-offset), H. Sossa, R. A. Vázquez, Communication, BEP terminale professionnelle : corrige., Vol. 7, Fontaine Picard, 1997.
URL http://www.ceser.in/ceserp/index.php/ijai/article/view/2260

[7] M. Meissner, M. Schmuker, G. Schneider, Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training, BMC Bioinformatics 7 (1) (2006) 125. doi:10.1186/1471-2105-7-125.
URL http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-7-125

[8] V. Gudise, G. Venayagamoorthy, Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks, in: Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706), IEEE, pp. 110–117. doi:10.1109/SIS.2003.1202255.
URL http://ieeexplore.ieee.org/document/1202255/

[9] David Stillwell, MyPersonality Database from The Psychometrics Centre, University of Cambridge.
URL https://www.psychometrics.cam.ac.uk/productsservices/mypersonality

[10] M. Kosinski, My Personality Project.
URL https://sites.google.com/michalkosinski.com/mypersonality

6. Appendix

6.1. *Neural Network Equations*

6.1.1. *Forward Propagation*

Forward pass is basically the prediction of neural networks. The notations that have been used in forward propagation is described below:

- $Z^{[i]}$ = Output of i^{th} Layer of Neural Network
- $A^{[i]}$ = Activation of i^{th} Layer of Neural Network
- X = input
- $W^{[i]}$ = Weight from $(i-1)^{th}$ to i^{th} Layer of Neural Network
- $b^{[i]}$ = Biases for i^{th} Layer of Neural Network

Equations for Forward Pass is shown in Algorithm 1.

Algorithm 1 Forward Propagation in Neural Network

```
procedure ForwardPropagation(X)                ▷ Forward Propagation
    Z[1] = W[1].X + b[1]
    A[1] = ReLU(Z[1])
    Z[2] = W[2].A[1] + b[2]
    A[2] = ReLU(Z[2])
    Z[3] = W[3].A[2] + b[3]
    A[3] = Sigmoid(Z[3])
end procedure
```

6.1.2. *Cross-Entropy Loss Function*

Cross-Entropy is the most popular loss function for neural networks. It is shown in Algorithm 2.

Algorithm 2 Loss Function in Neural Network

procedure LossFunction(Y)

RETURN $\frac{1}{m}\sum_{i=1}^{m}(-Y^{(i)}\log(\hat{Y^{(i)}}) - (1 - Y^{(i)})\log(1 - \hat{Y^{(i)}})$

end procedure

Algorithm 3 Back Propagation in Neural Network

procedure BACKPROPAGATION(X, Y)
  dZ$^{[3]}$ = A$^{[3]}$ - Y ▷ Derivatives Calculation
  dW$^{[3]}$ = (1/m) dZ$^{[3]}$.A$^{[2].\mathrm{T}}$
  dB$^{[3]}$ = (1/m) np.sum(dZ$^{[3]}$, axis=1, Keepdims = True)
  dZ$^{[2]}$ = W$^{[3].\mathrm{T}}$.dZ$^{[3]}$ * DReLU(Z$^{[2]}$)
  dW$^{[2]}$ = (1/m) dZ$^{[2]}$.A$^{[1].\mathrm{T}}$
  dB$^{[2]}$ = (1/m) np.sum(dZ$^{[2]}$, axis=1, Keepdims = True)
  dZ$^{[1]}$ = W$^{[2].\mathrm{T}}$.dZ$^{[2]}$ * DReLU(Z$^{[1]}$)
  dW$^{[1]}$ = (1/m) dZ$^{[1]}$.X$^{\mathrm{T}}$
  dB$^{[1]}$ = (1/m) np.sum(dZ$^{[1]}$, axis=1, Keepdims = True)

  W$^{[1]}$ = W$^{[1]}$ - 0.01 * dW$^{[1]}$ ▷ Parameter Update
  b$^{[1]}$ = b$^{[1]}$ - 0.001 * dB$^{[1]}$
  W$^{[2]}$ = W$^{[1]}$ - 0.01 * dW$^{[2]}$
  b$^{[2]}$ = b$^{[1]}$ - 0.001 * dB$^{[2]}$
  W$^{[3]}$ = W$^{[1]}$ - 0.01 * dW$^{[3]}$
  b$^{[3]}$ = b$^{[1]}$ - 0.001 * dB$^{[3]}$
end procedure

6.1.3. Backward Propagation

Backpropagation is basically a gradient based learning and optimization method which is prone to local optima. The notations that have been used in back propagation is described below:

- dZ$^{[i]}$ = Derivative of Z of i$^{\mathrm{th}}$ Layer of Neural Network
- A$^{[i].\mathrm{T}}$ = Transpose of A$^{[i]}$
- dW$^{[i]}$ = Derivative of W of i$^{\mathrm{th}}$ Layer of Neural Network
- dB$^{[i]}$ = Derivative of B of i$^{\mathrm{th}}$ Layer of Neural Network

Equations for Backprop is shown in Algorithm 3.

6.2. PSO Pseudocode

In computational science, particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions. Fig 4 and Fig 5 presents my implementation of PSO.

Main Equation of Particle Swarm Optimization -

- $\mathrm{WV_{ij}(t)}$ = Intertia Component
- $\mathrm{r_1c_1(P_{ij}(t) - x_{ij}(t))}$ = Cognitive Component
- $\mathrm{r_2c_2(g_j(t) - x_{ij}(t))}$ = Social Component
- $\mathrm{c_1}$, $\mathrm{c_2}$ = Acceleration Coefficients
- W = Inertia Coefficient
- t, t(t+1) = timestamps
- $\mathrm{V_{ij}(t)}$ = $\mathrm{j^{th}}$ velocity component of $\mathrm{i^{th}}$ particle
- $\mathrm{P_{ij}(t)}$ = $\mathrm{j^{th}}$ component of $\mathrm{i^{th}}$ particle's best position
- $\mathrm{x_{ij}(t)}$ = $\mathrm{j^{th}}$ component of $\mathrm{i^{th}}$ particle's current position
- $\mathrm{r_1}$, $\mathrm{r_2}$ = $N(\mu, \sigma^2)$

For determining the value of W, $\mathrm{c_1}$ and $\mathrm{c_2}$, I've used Constriction coefficients proposed by Clerc and Kennedy, 2002.

Let,

$$\phi = \phi_1 + \phi_2 \geq 4$$

$$0 \leq \kappa \leq 1$$

$$\chi = \frac{2\kappa}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}$$

Commonly, $\kappa = 1$, $\phi_{\ 1} = 2.05$, $\phi_2 = 2.05$

So, Finally,

W $= \chi$, C$_1 = \kappa\phi_1$, C$_2 = \kappa\phi_2$

And for Velocity and Position Update,

$$V_{\mathrm{ij}}(t+1) = WV_{\mathrm{ij}}(t) + r_1c_1(P_{\mathrm{ij}}(t) - x_{\mathrm{ij}}(t)) + r_2c_2(g_{\mathrm{j}}(t) - x_{\mathrm{ij}}(t))$$

$$x_{\mathrm{ij}}(t+1) = x_{\mathrm{ij}}(t) + V_{\mathrm{ij}}(t+1)$$

Algorithm 4 Particle Swarm Optimization in Neural Network

procedure PSOLOOP($NeuralNetwork$)
  for each epoch$_{\mathrm{(i)}}$ in range(1, maxIt) **do**
    for each particle$_{\mathrm{(j)}}$ in range(1, nPop) **do**
      $V_{\mathrm{j}}(t+1) \leftarrow WV_{\mathrm{j}}(t) + r_1c_1(P_{\mathrm{j}}(t) - x_{\mathrm{j}}(t)) + r_2c_2(g(t) - x_{\mathrm{j}}(t))$
      $x_{\mathrm{j}}(t+1) \leftarrow x_{\mathrm{j}}(t) + V_{\mathrm{j}}(t+1)$
      particle$_{\mathrm{(j)}}$.cost $\leftarrow$ COSTPSO(NeuralNetwork,particle$_{\mathrm{j}}$)
      if particle$_{\mathrm{(j)}}$.cost <particle$_{\mathrm{(j)}}$.bestCost **then**
        particle$_{\mathrm{(j)}}$.bestPoition $\leftarrow$ particle$_{\mathrm{(j)}}$.position
        particle$_{\mathrm{(j)}}$.besCost $\leftarrow$ particle$_{\mathrm{(j)}}$.cost
        if particle$_{\mathrm{(j)}}$.bestCost <globalBestCost **then**
          globalBestPosition $\leftarrow$ particle$_{\mathrm{(j)}}$.bestPosition
          globalBestCost $\leftarrow$ particle$_{\mathrm{(j)}}$.cost.bestCost
        end if
      end if
    end for
  end for
end procedure

Algorithm 5 Particle Swarm Optimization in Neural Network

```
procedure InitializeParticle(nVar, nPop)
    globalBestPosition ← 0⃗ of dimension (nVar x 1)
    globalBestCost ← None
    for each particle_(i) in range(1, nPop) do
        particle_(i).position ← N(μ, σ^2) of dimension (nVar x 1)
        particle_(i).velocity ← 0⃗ of dimension (nVar x 1)
        particle_(i).cost ← None
        particle_(i).bestPosition ← particle_(i).position
        particle_(i).bestCost ← particle_(i).cost
    end for
end procedure
procedure InitializeParameters(NeuralNetwork, X, Y, maxIt, nPop)
    nVar ← NeuralNetwork.numWeights + NeuralNetwork.numBiases
    κ ← 1
    φ_1 ← 2.05
    φ_2 ← 2.05
    φ ← φ_1 + φ_2
    χ ← 2κ / |2 − φ − sqrt(φ^2 − 4φ)|
    maxIt ← 100
    nPop ← 50
    w ← χ
    wdamp ← 0.99
    c_1 ← χ.φ_1
    c_2 ← χ.φ_2
end procedure
procedure COSTPSO(NeuralNetwork, particle)
    NeuralNetwork.weights ← particle.position.slice
    NeuralNetwork.biases ← particle.position.slice
    NeuralNetwork.ForwardPropagation(X)
    RETURN NeuralNetwork.Loss(Y)
end procedure
procedure InitializeSwarm(NeuralNetwork)
    for each particle_(i) in range(1, nPop) do
        particle_(i).cost ← COSTPSO(NeuralNetwork, particle_(i))
        if particle_(i).bestCost < globalBestCost then
            globalBestPosition ← particle_(i).bestPosition
            globalBestCost ← particle_(i).bestCost
        end if
    end for
end procedure
```

6.3. Big 5 Personality Questionnaire

The most frequently used measures of the Big Five comprise either items that are self-descriptive sentences or, in the case of lexical measures, items that are single adjectives. Research has suggested that some methodologies in administering personality tests are inadequate in length and provide insufficient detail to truly evaluate personality. Usually, longer, more detailed questions will give a more accurate portrayal of personality. The five factor structure has been replicated in different peer reports. However, many of the substantive findings rely on self-reports. Much of the evidence on the measures of the Big 5 relies on self-report questionnaires

Test

Rating	I....	Rating	I.....
	1. Am the life of the party.		26. Have little to say.
	2. Feel little concern for others.		27. Have a soft heart.
	3. Am always prepared.		28. Often forget to put things back in their proper place.
	4. Get stressed out easily.		29. Get upset easily.
	5. Have a rich vocabulary.		30. Do not have a good imagination.
	6. Don't talk a lot.		31. Talk to a lot of different people at parties.
	7. Am interested in people.		32. Am not really interested in others.
	8. Leave my belongings around.		33. Like order.
	9. Am relaxed most of the time.		34. Change my mood a lot.
	10. Have difficulty understanding abstract ideas.		35. Am quick to understand things.
	11. Feel comfortable around people.		36. Don't like to draw attention to myself.
	12. Insult people.		37. Take time out for others.
	13. Pay attention to details.		38. Shirk my duties.
	14. Worry about things.		39. Have frequent mood swings.
	15. Have a vivid imagination.		40. Use difficult words.
	16. Keep in the background.		41. Don't mind being the center of attention.
	17. Sympathize with others' feelings.		42. Feel others' emotions.
	18. Make a mess of things.		43. Follow a schedule.
	19. Seldom feel blue.		44. Get irritated easily.
	20. Am not interested in abstract ideas.		45. Spend time reflecting on things.
	21. Start conversations.		46. Am quiet around strangers.
	22. Am not interested in other people's problems.		47. Make people feel at ease.
	23. Get chores done right away.		48. Am exacting in my work.
	24. Am easily disturbed.		49. Often feel blue.
	25. Have excellent ideas.		50. Am full of ideas.

Figure 4: Questionnaire for Big Five Personality Score

6.4. Evolution Strategies Pseudocode

Algorithm 18 *The* (μ, λ) *Evolution Strategy*

```
μ ← number of parents selected
λ ← number of children generated by the parents

P ← {}
for λ times do
    P ← P ∪ {new random individual}
Best ← □
repeat
    for each individual Pᵢ ∈ P do
        AssessFitness(Pᵢ)
        if Best = □ or Fitness(Pᵢ) > Fitness(Best) then
            Best ← Pᵢ
    Q ← the μ individuals in P whose Fitness( ) are greatest
    P ← {}
    for each individual Qⱼ ∈ Q do
        for λ/μ times do
            P ← P ∪ {Mutate(Copy(Qⱼ))}
until Best is the ideal solution or we have run out of time
return Best
```

Figure 5: (μ,λ) Evolution Strategies