Neuro-Genetic Evolution for Bidirectional Stochastic Weight Sharing

Parisa Kasaeian

Research School of Computer Science
The Australian National University
U6194195@anu.edu.au

Abstract. Neuro-evolution techniques employ evolutionary algorithms to optimize the architecture and hyperparameters of artificial neural networks. Auto-associative neural networks (autoencoders) are an important type of neural nets, aiming at the reconstruction of the input by consecutive coding-encoding processes. In this paper, we present a novel bidirectional stochastic network which employs weight sharing under a Gaussian random distribution. Building upon this, we employ the genetic algorithm to fine-tune the network hyperparameters. Our experiments confirm that this neuro-genetic evolution highly outperforms the classification accuracy over other architectures, while applied to the both binary and multi-class scenarios.

Keywords: Autoencoder, Bidirectional Stochastic Weight Sharing, Neuro-Evolution, Genetic Algorithm.

1 Introduction

Autoencoders are one of the most inspiring architectures in neural networks with variety of applications in different areas of artificial intelligence from denoising and dimensionality reduction (compression) to learning of generative models and artistic style transfer (adversarial networks). In this work, we implement a shallow standard autoencoder, including an input layer, a hidden layer and an output layer. Inspiring by [1], this extends to a shared-weight architecture [3] and latter, adds reverse optimization pass to deploy a stochastic bidirectional learning [2][4]. Then, we will move it forward by combining the weight sharing and bidirectional optimization ideas to introduce our bidirectional weight sharing autoencoder. This make use of advantages of both networks to achieve a better model generalization. The main contribution of our work is built upon the fact that rigid sharing schemes [3][4], generally lead to a poor generalization at the test time.

Evolutionary algorithms have always been of interest of computer society. Genetic algorithms as important optimizers have been successfully applied to training neural networks [9]. Recently their extension for deep learning improves the performance of a deep autoencoder, producing a sparser neural network [10]. The genetic algorithms mimic the evolution in the nature. The idea is to create a population of species, which would be a collection of bidirectional autoencoders in our case, and subject them to evolution. Hence, they can mutate and reproduce, but only the fittest ones survive and are carried over to the next generation [11]. Our experiments show that standard classification algorithms perform better when our strategy is implemented for the training.

The paper is organized as follows. In section 2, we first formulate the standard, shared-weight and bidirectional algorithms and then, introduce our bidirectional weight sharing architecture added to its genetic version. Section 3 presents our extensive experiments with the discussions around the outcomes. Finally, we conclude in section 4 and offer some novel ideas for future works.

2 Method

To formulate our bidirectional weight sharing autoencoder, we start from a standard autoencoder and expand it to a shared-weight strategy, followed by a stochastic bidirectional learning algorithm.

Suppose an input layer $X = \{x_1, \cdots, x_N, 1\} \in R^{N+1}$ that generates an output layer $Y = \{y_1, \cdots, y_N, y_{N+1}\} \in R^{N+1}$ in an autoencoder network. Let consider a hidden layer $H = \{h_1, \cdots, h_{N1}\} \in R^{N1}$ with $N1 \leq N$ neurons which is fully connected to both input and output layers. This implies two weight matrices $W_1 \in R^{(N+1)\times N1}$ and $W_2 \in R^{N1\times(N+1)}$ that connect the input-hidden and hidden-output layers, respectively.

In a standard autoencoder, we first calculate the hidden layer as

$$H = f\left(W_1{}^T X\right) \tag{1}$$

such that $f(t) = \frac{1}{1+e^{-t}}$ is the sigmoid function. It worth mentioning that the bias weights are included in the above equation by appending a unit vector $\{1\}$ to the input layer X. The next step is the calculation of the output layer.

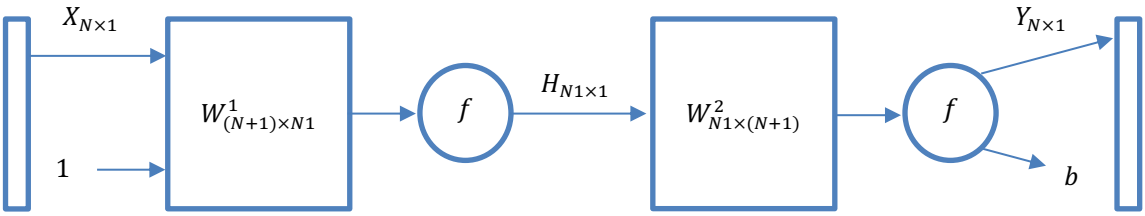

Figure 1 Standard Autoencoder. This consists of fully-connected input, hidden, and output layers with sigmoid non-linear rectifiers. To implement the bias, a unit vector is appended to the end of input vector. The training of input/output weights follows the standard backpropagation practice, employing stochastic gradient descent algorithm.

$$Y = f\left(W_2{}^T H\right) \tag{2}$$

To adapt the weights recursively, we compute the gradient of output layer with respect to the input layer as

$$\Delta_2 = (X - Y)(Y(I_Y - Y)) \tag{3}$$

Here, $I_Y \in R^{N+1}$ is the unity matrix. The updated weight matrix holds

$$W_2 = W_2 + \eta \times \left(H\Delta_2{}^T\right) \tag{4}$$

While η is a learning rate. To update the input weights W_1, we follow the same practice.

$$\Delta_1 = (W_2\Delta_2)(H(I_Y - H)) \tag{5}$$

Again, $I_H \in R^{N1}$ and we have

$$W_1 = W_1 + \eta \times \left(X\Delta_1{}^T\right) \tag{6}$$

The above procedure repeats for a pre-defined number of epochs, until a good convergence between the input X and output Y is granted. Algorithm 1 presents the above formulations in a code snippet.

```
Algorithm 1
Standard Autoencoder

Input: X and η
Output: W1 and W2

1: while epoch < maximum epochs
2:    Calculate the hidden layer H by equation (1)
3:    Calculate the output layer Y by equation (2)
4:    Calculate gradient Δ2 of the output layer by equation (3)
5:    Update the output weights W2 by equation (4)
6:    Calculate gradient Δ1 of the input layer by equation (5)
7:    Update the input weights W1 by equation (6)
```

An extension to the above standard algorithm was proposed in [4] which in each epoch, apply the following formulation to share the weights between updated input/output layers.

$$W_1(i,i) = W_2(i,i) \quad \forall i \in [1, N1] \tag{7}$$

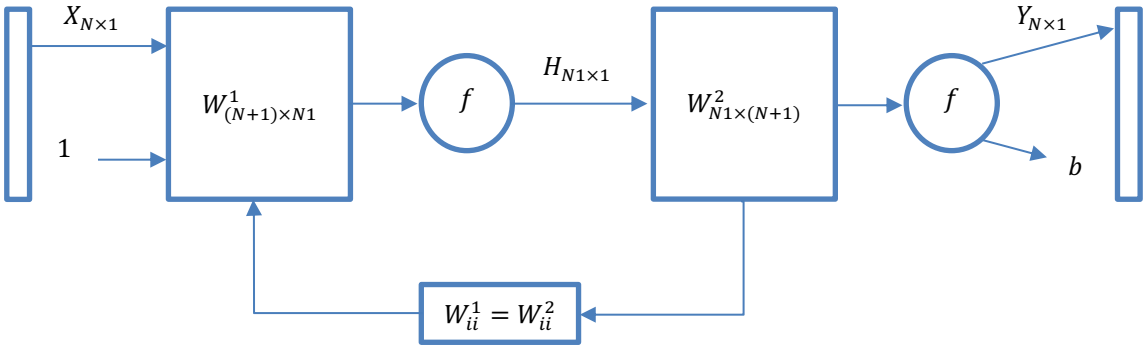

Figure 2 Shared-Weight Autoencoder. This shares the same architecture as standard autoencoder, whilst replaces diagonal entries of the input and output weight matrices after deploying backpropagation in each epochs of the training process.

Algorithm 2 shows the shared-weight process in a procedural order.

```
Algorithm 2
Shared-Weight Autoencoder

Input: X and η
Output: W1 and W2

1: while epoch < maximum epochs
2:    Calculate the hidden layer H by equation (1)
3:    Calculate the output layer Y by equation (2)
4:    Calculate gradient Δ2 of the output layer by equation (3)
5:    Update the output weights W2 by equation (4)
6:    Calculate gradient Δ1 of the input layer by equation (5)
7:    Update the input weights W1 by equation (6)
8:    Share the input/output weights by equation (7)
```

The stochastic bidirectional learning [4] tried to increase the generalization power of the standard autoencoders by adding another reverse pass from the output to the input. To implement this in each epoch, the weights are swapped after final updating of W_1, W_2 as follows

$$\begin{aligned} W_{R1} &= W_2^T \\ W_{R2} &= W_1^T \end{aligned} \tag{8}$$

Now, we recalculate the hidden and input layers by considering Y as the input.

$$\begin{aligned} H &= f\left({W_{R1}}^T Y\right) \\ X &= f\left({W_{R2}}^T H\right) \end{aligned} \tag{9}$$

In the reverse pass, the gradients are defined as

$$\begin{aligned} \Delta_{R2} &= (Y - X)(X(I_X - X)) \\ \Delta_{R1} &= (W_{R2}\Delta R_2)(H(I_H - H)) \end{aligned} \tag{10}$$

To proceed, we compute the updated reverse weights as follows.

$$\begin{aligned} W_{R2} &= W_{R2} + \eta \times \left(H{\Delta_{R2}}^T\right) \\ W_{R1} &= W_{R1} + \eta \times \left(X{\Delta_{R1}}^T\right) \end{aligned} \tag{11}$$

and finally, the weights being transferred to the next epoch are

$$\begin{aligned} W_1 &= W_{R2}^T \\ W_2 &= W_{R1}^T \end{aligned} \tag{12}$$

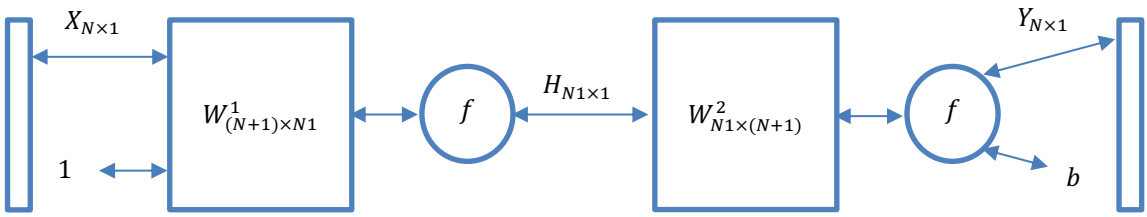

Figure 3 Stochastic Bidirectional Autoencoder. This merges two standard autoencoders which mirror each other. In practice, the output of one autoencoder feeds the input of the other while the weight matrices get updated consecutively. For implementation, the input and output vectors/weights are swapped, before each training epoch.

There are two alternatives for the above bidirectional strategy. The one-pass strategy does the swapping for each input instance separately, while two-pass scheme applies weight-swapping after updating the weights for all the input instances. In practice, the differences between one/two-pass trainings are quite small and they perform almost the same. We believe that swapping the output/input and re-training of the weights in bidirectional autoencoders, can produce better accuracy, when this is implemented for each instance separately (one-pass). Therefore, we deploy the one-pass implementation which also addresses the run-time considerations.

Algorithm 3 presents the implementation of a stochastic bidirectional autoencoder.

```
Algorithm 3
Stochastic Bidirectional Autoencoder

Input: X and η
Output: W1 and W2

1: while epoch < maximum epochs
2:    Calculate the hidden layer H by equation (1)
3:    Calculate the output layer Y by equation (2)
4:    Calculate gradient Δ2 of the output layer by equation (3)
5:    Update the output weights W2 by equation (4)
6:    Calculate gradient Δ1 of the input layer by equation (5)
7:    Update the input weights W1 by equation (6)
8:    Swap the input/output weights by equation (8)
9:    Recalculate the hidden and input layers by equation (9)
10:   Calculate the reverse gradients ΔR2 and ΔR1 by equation (10)
11:   Update the reverse weights WR2 and WR1 by equation (11)
12:   Swap the weights by equation (12)
```

2.1 Bidirectional Weight Sharing

To improve the generalization of the stochastic bidirectional learning, we propose a novel algorithm which combines the shared-weight and bidirectional autoencoders using a Gaussian random distribution to share the input/output weights. The problem of sharing the weights by formula (6) lies in its rigid assignment. Imposing a harsh constraint on this assignment prevents the network from proper generalization at the test time. Inspired by the idea of drop-out in the stacked autoencoders, we define two random binary matrices $R_1 \in R^{(N+1)\times N1}$ and $R_2 \in R^{N1\times(N+1)}$ which have similar sizes as W_1, W_2 weight matrices. To share the weights after applying formula (12), we combine the weights as

$$\begin{aligned} W_1 &= R_1 W_1 + (1 - R_1) W_2^T \\ W_2 &= (1 - R_2) W_1^T + R_2 W_2 \end{aligned} \quad \textbf{(13)}$$

This means that we hold some input original weights, replace the rest with the output weights, and vice-versa. It is in contrast with the conventional shared-weight algorithm [3], because our strategy employs random indices to assign shared weights. This leads to better results on the test time, because the training phase learns how to deal with random weights.

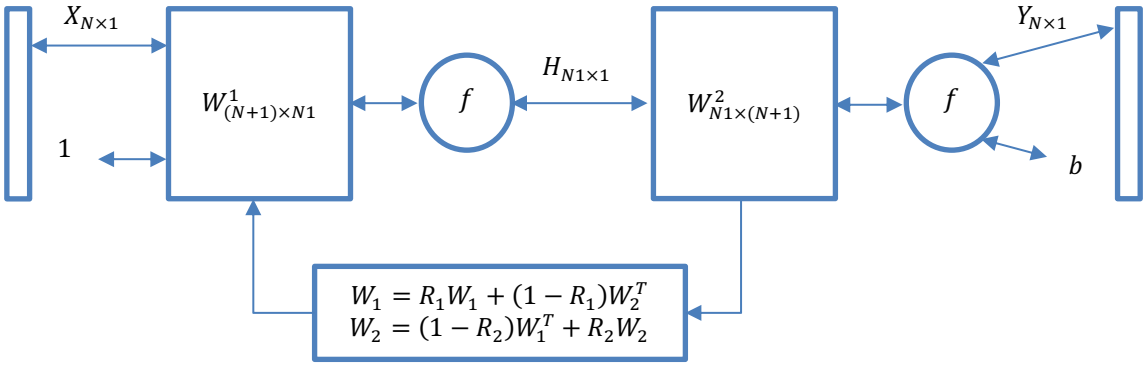

Figure 4 Bidirectional Weight Sharing Autoencoder. This combines the bidirectional and shared-weight autoencoders. Inspired by drop-out technique in modern autoencoders which randomly ignores some neurons to improve the generalization at the test time, we replace some input weights with output ones and vice versa using a Gaussian random distribution. This imposes a controlled randomness in weight sharing at the training time.

Algorithm 4 summarizes the implementation of our proposed bidirectional weight sharing method. The trick is to replace the calculated bias with unit vector after each swapping. Although there should be other strategies for sharing of biases, our experiments show that the above trick produces better results.

```
Algorithm 4
Bidirectional Weight sharing Autoencoder

Input: X and η
Output: W1 and W2

1: while epoch < maximum epochs
2:    Calculate the hidden layer H by equation (1)
3:    Calculate the output layer Y by equation (2)
4:    Calculate gradient Δ2 of the output layer by equation (3)
5:    Update the output weights W2 by equation (4)
6:    Calculate gradient Δ1 of the input layer by equation (5)
7:    Update the input weights W1 by equation (6)
8:    Swap the input/output weights by equation (8)
9:    Recalculate the hidden and input layers by equation (9)
10:   Calculate the reverse gradients ΔR2 and ΔR1 by equation (10)
11:   Update the reverse weights WR2 and WR1 by equation (11)
12:   Swap the weights by equation (12)
13:   Share the input/output weights by equation (13)
```

2.2 Classification

Autoencoders are not designed as classifiers. To employ above algorithms for the sake of binary or multi-class classification, we introduce a Softmax classifier network which employs the autoencoder code (hidden layer) as its input. To implement the classifier, we define a three-layer fully-connected network followed by a Softmax layer.

Starting by $H \in R^{N1}$, the first layer of 16 hidden neurons compresses this code and feeds it to the next layer of 8 hidden neurons, which maps it to the third layer. The number of hidden neurons in the third layer equals the number of classes in the dataset under study. Finally, a Softmax layer converts this to the class probabilities. We select the class with the maximum probability as the predicted class. Here, we report the accuracy of classifier defined as the ratio of correct predictions (true accept) to the whole predictions.

We conduct several experiments on a variety of datasets and classification settings. To better generalize the outcomes, datasets are selected to be in different shapes (number of instances vs. number of features) and number of classes (binary vs. multi-class). In all the experiment, we set the number of hidden neurons $N1 = 32$, learning rate $\eta = 0.01$, and the number of epochs to10^4. We also conduct each experiment with 10 different random seeds for weight initialization and report the mean and standard deviation as the classification accuracy.

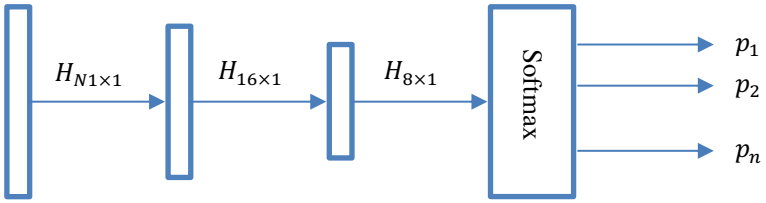

Figure 5 Softmax Classifier Network. This is fed by the hidden layer (code) of an autoencoder, which then projected to a Softmax layer to gives the probability of n different classes as $\{p_1, \cdots, p_n\} \in R^n$ such that the highest probability corresponds to the target predicted class.

2.3 Genetic Algorithm

To improve the precision of our bidirectional weight sharing network, we employ the evolutionary techniques to optimize the autoencoder hyperparameters *i.e.* number of hidden neurons, activation function, drop-out percentage, and learning rate. For implementation, we first create a population of bidirectional autoencoders by assigning the hyperparameters to randomly selected parameters from a predefined acceptable range of values. Then, we train the population and calculate their accuracy as a measure of fitness.

The bidirectional autoencoders with higher accuracies have better chance to be picked as parents to produce the next generation. Two high-accuracy networks are randomly selected for breading and making a child autoencoder. This child inherits the hyperparameters form the parents in a random order. The next step is the mutation of the child network by replacing one of its hyperparameters with values form the predefined acceptable range. This guarantees a good diversity in the consequent generations. Repeating this procedure for several number of generations, the best setting of hyperparameters will be produced for the bidirectional weight sharing network.

Figure 6 Genetic Algorithm. This procedure finds the optimal hyperparameters for the bidirectional weight sharing autoencoder.

Algorithm 5 provides a summary of above process to find the optimal hyperparameters for the bidirectional weight sharing autoencoders.

```
Algorithm 5
Genetic Bidirectional Weight sharing Autoencoder

Input: Size of population, number of generation, N1, R1, f, and η
Output: Set of optimal hyperparameters

1: Create a population of autoencoders with random hyperparameters
2: Train and evaluate the population
2: In each generation except the last generation
2:    Randomly pick parent autoencoders with high classification accuracy
3:    Bread these parents to make child autoencoders
4:    Inherit hyperparameters to the children
5:    Mutate children by replacing one of their parameters with the input
6:    Train and evaluate the children as the new generation
7:  Search the last generation to find the optimal hyperparameters
```

3 Experiments

To benchmark our proposed bidirectional weight sharing scheme with other algorithms, we select three different classification tasks from UCI machine learning repository [5]. The corresponding datasets are Single Proton Emission Computed Tomography (Heart) [6], Soybean (Large) [7], and Molecular Biology (Promoter Gene Sequences) [8]. The detailed specifications of the datasets are described as follows. We select SPECT and SOYBEAN as typical binary and multi-class classification scenarios. The GENE dataset is a special case of binary classification, since the features are non-numeric. This is a hard challenge because the order of features in each instance is also an important factor that the autoencoder should consider learning at the training phase. Table 1 summarizes the specifications of these datasets.

Table 1. Datasets information.

Dataset	# Instance	# Class	# Feature
SPECT	267	2	23
SOYBEAN	307	19	35
GENE	106	2	57

The SPECT dataset is about diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images of the heart. Each of the patients is classified into two normal or abnormal categories. The database contains 267 instances of 23 features, divided to 80 training and 187 test samples. Regarding class distribution, 55 samples belong to the normal and 212 to the abnormal classes, respectively.

The SOYBEAN dataset includes 307 instances of 35 features, categorized in 19 classes. The last four classes have few examples and the class distribution is not uniform. In other words, the distribution of classes is highly imbalanced. The literature around balancing of class distribution is vast, but we do not apply any balancing strategy in our experiments.

Finally, the GENE dataset has been developed to help evaluate a hybrid learning algorithm. This consists of 106 samples with 57 non-numeric features, i.e. the sequential nucleotide ("base-pair") positions. They are divided to promoter and non-promoter classes with 50 percent class distribution. To interpret the DNA sequences of AGTC nucleotides, we assign a number to each nucleotide and follow the rule of thumb (80-20) to divide the dataset to training/test sets.

3.1 Results

Table 2 shows the mean and standard deviation of classification accuracies for standard, shared-weight, bidirectional, and bidirectional weight sharing autoencoders on three different datasets.

Table 2. Mean and standard deviation of classification accuracies.

Algorithms	SPECT (%)	SOYBEAN (%)	GENE (%)
Standard	65.35 ± 4.38	86.71 ± 1.90	58.87 ± 6.82
Shared-Weight	65.78 ± 3.36	86.81 ± 1.41	57.92 ± 4.88
Bidirectional	65.40 ± 4.58	86.30 ± 1.77	59.06 ± 6.78
Bidirectional Weight Sharing	66.04 ± 4.33	87.10 ± 1.46	69.43 ± 3.65
Genetic Bidirectional Weight Sharing	$\mathbf{72.37 \pm 1.41}$	$\mathbf{88.62 \pm 1.23}$	$\mathbf{81.13 \pm 1.87}$

Table 3 lists the range of network hyperparameters from whom genetic algorithm picks the optimal settings. Increasing the valid range of parameters widens the opportunity to find out better hyperparameters, whilst makes the processing time infeasible for large datasets.

Table 3. The range of network hyperparameters for bidirectional weight sharing autoencoders.

Hyperparameter	
Hidden Neurons	32, 64, 128, 256, 512, 1024
Activation	Hyperbolic Tangent, Sigmoid
Drop-out	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
Learning Rate	0.001, 0.01, 0.1, 1.0

Table 4 presents the optimal hyperparameters for bidirectional weight sharing network, derived by genetic algorithm. We set the size of population to 10 bidirectional autoencoders and evolve them in 10 generations. For implementation, we get some ideas from an open source repository called evolution of the neural network with genetic algorithm [12].

Table 4. The optimal hyperparameters for bidirectional weight sharing autoencoders.

Dataset	Hidden Neurons	Activation	Drop-out	Learning Rate
SPECT	32	Sigmoid	0.8	0.01
SOYBEAN	1024	Sigmoid	0.2	0.001
GENE	1024	Sigmoid	0.3	0.001

3.2 Discussions

Table 2 confirms that our proposed architecture (bold) outperforms on both binary (SPECT, GENE datasets) and multi-class (SOYBEAN dataset) classifications. This shows an improvement over the stochastic bidirectional by a two-digit margin. An interesting observation is that the shared-weight autoencoder outperforms stochastic bidirectional on SPECT and SOYBEAN datasets, but falls behind on the GENE dataset.

Looking at Table 4, it turns out that Sigmoid is the best activation function for the proposed bidirectional weight sharing network. The high variations in the number of hidden neurons, drop-out, and learning rate are mostly due to various number of classes and features of the datasets under study. For example, we perform binary classification for both SPECT and GENE datasets, but the complexity of GENE dataset regarding the order of features, forces the need for longer code length (number of hidden neurons). The strength of neuro-genetic algorithm lies in its ability to match the network architecture with the requirement of classification problem.

Table 5 represents the best overall accuracies reported in literature, using different machine learning algorithms. It seems that the shallow neural networks employed in our experiments cannot perform well compared to the other proposed algorithms. The reason is that the number of learning parameters are not sufficient to draw proper separating hyperplanes on the data manifolds. In other words, we need far more parameters to outperform the best accuracies.

Table 5. Best accuracies in literature.

Dataset	Accuracy (%)
SPECT	90.4 (CLIP4)
SOYBEAN	97.1 (IWN)
GENE	96.2 (KBANN)

To improve our performance, we need to increase the number of layers and move towards deep learning architectures. Employing of modern deep learning techniques like drop-out, rank pooling, stacking and boosting, will highly improve the efficiency of the above architectures. Since the number of instances are small, applying deep learning strategies may lead to overfitting problem, which means perfect training precision and poor test performance. To prevent overfitting in the experiments, our proposed Gaussian weight sharing method would be highly beneficial. In addition, employing other evolutionary techniques might help to fine-tune the hyperparameters in a more optimal way.

4 Conclusion and Future Work

Inspiring by the concept of drop-out in deep learning, we introduce a novel bidirectional stochastic autoencoder which shares the coding-encoding weights under a gaussian random distribution. This highly generalizes the trained model for binary and multi-class classifications. Using genetic algorithm, we extract the optimal hyperparameters of our proposed bidirectional autoencoder. Our experiments on three different datasets confirm that our proposed technique outperforms other algorithms in accuracy with a big margin. For the future work, we consider other random regimes like uniform, lognormal, negative exponential, gamma, and beta distributions. We can also try the other evolutionary techniques or extend the use of genetic algorithm to tune the hyperparameters of the classifier inside the bidirectional autoencoder.

5 References

1. Gedeon, T. D., & Harris, D. (1992, June). Progressive image compression. In Neural Networks,1992. IJCNN., International Joint Conference on (Vol. 4, pp. 403-407). IEEE.
2. Nejad, A. F., & Gedeon, T. D. (1995, November). Bidirectional neural networks and class prototypes. In Neural Networks, 1995. Proceedings., IEEE International Conference on (Vol.3, pp. 1322-1327). IEEE.
3. Gedeon, T. D., Catalan, J. A., & Jin, J. Image Compression using Shared Weights and Bidirectional Networks. In Proceedings 2nd International ICSC Symposium on Soft Computing (SOCO'97) (pp. 374-381).
4. Gedeon, T. D. (1998, October). Stochastic bidirectional training. In Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on (Vol. 2, pp. 1968-1971). IEEE.
5. UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/index.php
6. Kurgan, L.A. and Cios, K.J. and Tadeusiewicz, R. and Ogiela, M. and Goodenday, L.S.: Knowledge Discovery Approach to Automated Cardiac SPECT Diagnosis, Journal of Artificial Intelligence in Medicine, vol. 23, no. 2, pp. 149--169 (2001)
7. Tan, M. and Eshelman, L.: Using Weighted Networks to Represent Classification Knowledge in Noisy Domains, Proceeding of the Fifth International Conference on Machine Learning Machine Learning, pp. 121--134, Elsevier (1988)
8. Towell, G.G. and Shavlik, J.W. and Noordewier, M.O.: Refinement of Approximate Domain Theories by Knowledge-based Neural Networks, Proceedings of the Eighth National Conference on Artificial intelligence, Boston, MA, vol. 861866, (1990)
9. Schaffer J.D., Whitley D., and Eshelman L.J.: Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art, International Workshop on Combinations of Genetic Algorithms and Neural Networks pp. 1--37 (1992)
10. David, Omid E and Greental, Iddo: Genetic Algorithms for Evolving Deep Neural Networks, Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 1451--1452, ACM (2014)
11. Ruehle, F.: Evolving Neural Networks with Genetic Algorithms to Study the String Landscape, Journal of High Energy Physics, no. 8, p. 38 (2017)
12. Harvey, M.: Evolve a neural network with a genetic algorithm, GitHub repository, https://github.com/harvitronix/neural-network-genetic-algorithm (2017)