International Journal of
Science and Engineering Investigations vol. 6, issue 65, June 2017
ISSN: 2251-8843

Neuroevolution of Autoencoders by Genetic Algorithm

Hidehiko Okada
Faculty of Computer Science and Engineering, Kyoto Sangyo University
(hidehiko@cc.kyoto-su.ac.jp)

Abstract-In this paper, the author experimentally evaluates the ability of a genetic algorithm in evolutionary training of autoencoders. An autoencoder is a component of a deep neural network known as a stacked autoencoder. Optimization of neural networks by means of evolutionary algorithms is called neuroevolution. Weights and biases in an autoencoder are optimized by the genetic algorithm so that the autoencoder can precisely reproduce its input data. A dataset of handwritten digits is used in the experiment. Results showed that the genetic algorithm could evolve autoencoders that reproduced the training and test data better as the autoencoders included more hidden units. A clear trade-off was observed between the reproduction accuracy and the encoding efficiency.

Keywords- *Neural Network, Evolutionary Algorithm, Optimization*

I. INTRODUCTION

Deep neural networks and their learning algorithms have been actively researched recently [1-14]. A stacked autoencoder is a kind of the deep neural network, where an autoencoder is a kind of layered feedforward neural networks [1,7,8]. An autoencoder can be trained by the well-known backpropagation (BP) algorithm [15], but the training of neural networks by the BP algorithm are likely to get trapped in an undesirable local minimum because the algorithm is based on a gradient decent method. Besides, several methods are proposed for training neural networks by using evolutionary algorithms, known as neuroevolution and evolutionary neural networks [16,17]. An advantage of evolutionary algorithms over the BP in training neural networks is that evolutionary algorithms can globally search solutions well and thus the trained neural networks are less likely trapped in an undesirable local minimum [18-23]. Therefore, we can expect that evolutionary algorithms can contribute well to the training of autoencoders (and thus stacked autoencoders). In this paper, the author applies an evolutionary algorithm to the neuroevolution of autoencoders. As the evolutionary algorithm the author selects a genetic algorithm, because genetic algorithms [24,25] are the most well-known evolutionary algorithms.

II. AUTOENCODER

An autoencoder [1,7,8] is a layered feedforward neural network where the number of units in the output layer is the same as the number of units in the input layer. An autoencoder is trained to output the same values as input values, in other words, to reproduce their input data. Fig. 1 shows the topology of an autoencoder adopted in this research. It has a single hidden layer. Usually, the number of hidden units is smaller than those of input (output) layer: N dimensional input real vectors are encoded (compressed) to $M(< N)$ dimensional real vectors between the input and hidden layers, and the M dimensional real vectors are decoded (decompressed) to the N dimensional real vectors between the hidden and output layers. Note that the compression/decompression process is not lossless but lossy so that the output values are not exactly be the same as the input values. An autoencoder is trained to make the error between the input and output values smaller.

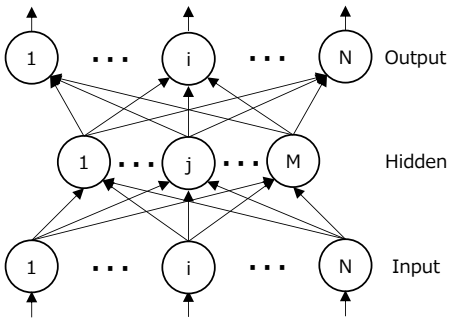

Figure 1. Topology of an autoencoder in this research.

The feedforward calculations in this autoencoder are the same as those in the traditional three layered perceptron. The following equations (1)-(5) show the calculations.

Input layer:

$$out_i^{(1)} = x_i, i = 1,2,\dots,N \quad (1)$$

Hidden layer:

$$in_j^{(2)} = \theta_j^{(2)} + \sum_i w_{i,j}^{(2)}\, out_i^{(1)}, j = 1,2,\dots,M \quad (2)$$

$$out_j^{(2)} = f(in_j^{(2)}), j = 1,2,\dots,M \quad (3)$$

Output layer:

$$in_i^{(3)} = \theta_i^{(3)} + \sum_j w_{j,i}^{(3)}\, out_j^{(2)}, i = 1,2,\dots,N \quad (4)$$

$$out_i^{(3)} = f(in_i^{(3)}), i = 1,2,\dots,N \quad (5)$$

The symbols in (1)-(5) denote as follows:

x_i	Input value to i-th input unit.
$out_i^{(1)}$	Output value from i-th input unit.
$in_j^{(2)}$	Input value to j-th hidden unit .
$w_{i,j}^{(2)}$	Weight value from i-th input unit to j-th hidden unit.
$\theta_j^{(2)}$	Bias value of j-th hidden unit.
$out_j^{(2)}$	Output value from j-th hidden unit.
$in_i^{(3)}$	Intput value to i-th output unit.
$w_{j,i}^{(3)}$	Weight value from j-th hidden unit to i-th output unit.
$\theta_i^{(3)}$	Bias value of i-th output unit.
$out_i^{(3)}$	Output value from i-th output unit.

$f()$ is a unit activation function, where the sigmoidal one is adopted in this research: $f(x) = 1/(1 + e^{-x})$.

Suppose the training data are N dimensional real vectors and the number of the data is D.

$$\boldsymbol{X} = \{\boldsymbol{x}_d\}, d = 1,2, \dots, D \quad (6)$$

$$\boldsymbol{x}_d = (x_{d,1}, x_{d,2}, \dots, x_{d,N}) \quad (7)$$

In (6), $\boldsymbol{X}$ is the set of training data. Each training data ($\boldsymbol{x}_d$ in (7)) is the N-dimensional real vector. An autoencoder is trained (i.e., values of $w_{i,j}^{(2)}$, $\theta_j^{(2)}$, $w_{j,i}^{(3)}$, and $\theta_i^{(3)}$ are optimized) so that its output values ($out_i^{(3)}, i = 1,2, \dots, N$) become closer to its inputs ($x_{d,i}, i = 1,2, \dots, N$). In other words, the input value $x_{d,i}$ is the target for the output value $out_i^{(3)}$. Thus, the error between $x_{d,i}$ and $out_i^{(3)}$ becomes smaller by optimizing the value of weights and biases.

$$e_d = \frac{1}{N}\sum_{i=1}^{N}(out_i^{(3)} - x_{d,i})^2 \quad (8)$$

$$e = \frac{1}{D}\sum_{d=1}^{D} e_d \quad (9)$$

e_d in (8) denotes the error for $\boldsymbol{x}_d$ ($0\% \le e_d \le 100\%$), and e in (9) denotes the average error over the entire training data $\boldsymbol{X}$ ($0\% \le e \le 100\%$).

III. Evolutionary Training by Genetic Algorithm

Instead of the BP, a genetic algorithm (GA) is adopted as the training method of the autoencoder in this research. A GA is a population-based multi-point search method, whereas the BP is a single-point search method. Because of this difference, a GA is better than the BP in searching solutions globally. It was reported that an evolutionary algorithm could optimize neural networks better than the BP did [18-23]. Optimization of neural networks by means of evolutionary algorithms is called neuroevolution [16,17]. In this paper, the author reports experimental results on the neuroevolution of autoencoders. There are two types of neuroevolution methods: (A) the topology of a neural network (e.g., the number of hidden layers, the number of units in a hidden layer) is fixed and the weights are optimized, or (B) both of the topology and the weights are simultaneously optimized. In this paper, the author adopts the former method. The autoencoder with the topology shown in Fig.1 includes $2MN$ $(= MN + MN)$ weights and $M + N$ biases. Thus, the autoencoder includes $2MN + M + N$ parameters in total. This parameter consists of a $2MN + M + N$ dimensional real vector and the vector is treated as the genotype in an evolutionary algorithm. The phenotype in the algorithm is the autoencoder in Fig.1. Evolutionary operators are applied to the $2MN + M + N$ real values to optimize them so that the error becomes smaller. The error value e in (9) is calculated with the training data and the output values of an autoencoder.

The evolutionary processes of a genetic algorithm in this paper is as follows:

Step1: Initialization

Step2: Evaluation

Step3: Conditional Termination

Step4: Crossover and Mutation

Step5: Goto Step2

In Step1, genotype values ($2MN + M + N$ real values) are initialized with random numbers for each of P individuals, where P denotes the population size. The value of P is given. The domain range of each genotype value should be neither too large nor too small in this research, because the value is used as a weight or bias value in a neural network. In Step2, fitness of new individuals (those with new genotype values) are evaluated. In this research, the fitness is based on the error in (9). An individual with a smaller error fits better (and thus ranked higher). In Step3, the loop of evolutionary processes is finished if a given termination condition is met. In this research, the loop is finished if the number of generations is reached to a given limit. In Step4, new offspring individuals are produced by using current individuals as parents. New genotype values for the offspring individuals are determined by using the genotype values of parent individuals. Step4 consists of the following sub-steps:

Step4-1: Elitism

Step4-2: Selection of two parent individuals

Step4-3: Crossover

Step4-4: Mutation

Step4-5: Conditional Termination

Step4-6: Goto Step4-2

In Step4-1, the best L individuals in the parent population are copied to the offspring population, where L is a given number. This process aims at preserving "elite" individuals. The remaining P-L individuals in the offspring population are determined in the loop of Steps 4-2 – 4-6. In Step4-2, two individuals are selected from the parent population. In this

research, the tournament selection is adopted. In Step4-3, a new offspring individual is produced by applying a crossover method to the two parent individuals selected in Step4-2. In this research, the blend crossover [26] is adopted. In Step4-4, each individual newly produced by the crossover (Step4-3) is randomly mutated. Each of the real $2MN + M + N$ values in the genotype vector of an individual is randomly initialized under a given small probability. In Step4-5, the loop of reproduction processes is finished if the number of individuals newly produced in Steps 4-2 – 4-6 reaches to P-L. Thus, the loop of Steps 4-2 – 4-6 is repeated P-L times.

IV. Evolutionary Training by Genetic Algorithm

This section reports an experimental study in which a dataset of handwritten digits is used as training data. The dataset is the Optical Recognition of Handwritten Digits Data Set which is available in the UC Irvine Machine Learning Repository.
(https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+ of +Handwritten+Digits)

For each of the 10 digits (0,1,…,9), 20 data are randomly extracted from the data file optdigits.tra. Thus, the total number of the sampled data is $10 * 20 = 200$. A half of the 200 data is used as the training data, and the remaining half is used as the test data. Each data consists of $8 * 8 = 64$ pixels and a pixel is valued with either of 0,1,…,16 (0: white, 16: black). In this experiment, the pixel values are normalized to a real value within the interval [0.0, 1.0] by dividing the values by 16.0. Figs. 2 and 3 visually show the training and test data respectively.

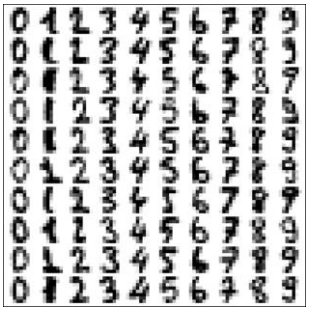

Figure 2. Training data in this experiment.

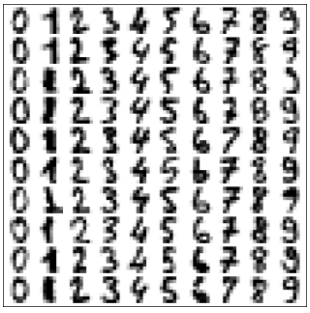

Figure 3. Test data in this experiment.

The numbers of units in the input and output layers are 64, because each training data consists of 64 values. The number of hidden units is experimentally set to 16, 25, 36 and 49. For example, the autoencoder with 36 hidden units has $64 * 36 + 36 * 64$ weights and 36+64 biases in total. Thus, the genotype is a 4708 dimensional real vector for an autoencoder with 36 hidden units.

Parameter values of the genetic algorithm are experimentally set as follows:

- Initial genotype values: randomly sampled from the standard normal distribution.
- Limit of genotype values: within the interval [-5.0, 5.0].
- Population size: 100.
- Limit of generations: 10,000.
- Number of elite individuals: 2.
- Tournament size for the selection of parents: 10.
- α for the blend crossover: 0.5.
- Mutation probability: 1/D, where D is the genotype length. D=2128, 3289, 4708, 6385 for the autoencoder with 16, 25, 36, 49 hidden units respectively.

Figs. 4-16 show the results of this experiment. Fig. 4 shows the error e along with the generations of GA. The graph legend shows the number of hidden units. Figs. 5-16 visually show the output by the best autoencoder in the first/last generations. For example, Figs. 5-7 show the results where the number of hidden units is 16. Fig. 5 shows the output by the best autoencoder among the 100 autoencoders in the first generation. The error between the output in Fig. 5 and the input in Fig. 2 is large (33.07% per pixel), because the weights and biases are just randomly initialized. Fig. 6 shows the output by the best autoencoder in the last generation. The error between the output in Fig. 6 and the input in Fig.2 is smaller (13.30% per pixel), but the output is not similar enough to the input. Fig. 7 shows the output by the best autoencoder in the last generation, where the input is the test data in Fig. 3. The trained autoencoder could, to a certain extent, reproduce the input data that is not used for the training, but the output is not also similar enough to the input. These are because the number of hidden units is not enough. Figs. 8-10, 11-13 and 14-16 show the results in the same manner as Figs 5-7, where the numbers of hidden units are 25, 36 and 49 respectively. Figs. 15 and 16 show that the trained autoencoder with 49 hidden units reproduces the training and test data well.

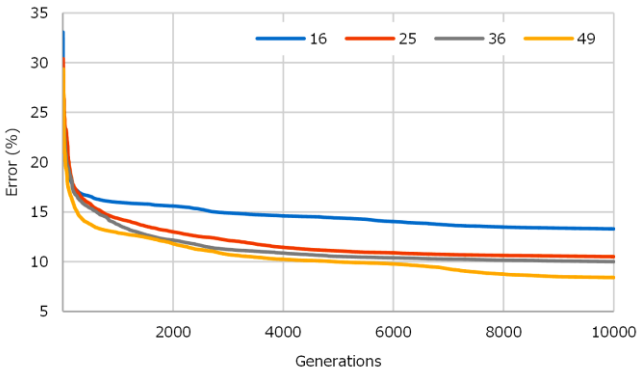

Figure 4. Decrease of the error along with the generations of GA. The legend shows the number of hidden units.

Figure 5. Output by the best autoencoder in the first generation (#Hidden units: 16, Input: training data, Error: 33.07%).

Figure 6. Output by the best autoencoder in the last generation (#Hidden units: 16, Input: training data, Error: 13.30%).

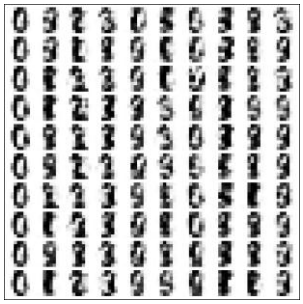

Figure 7. Output by the best autoencoder in the last generation (#Hidden units: 16, Input: test data, Error: 14.89%).

Figure 8. Output by the best autoencoder in the first generation (#Hidden units: 25, Input: training data, Error: 30.40%).

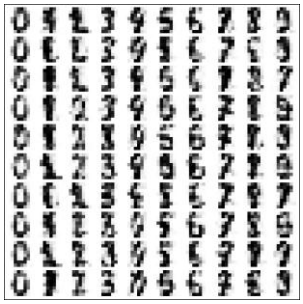

Figure 9. Output by the best autoencoder in the last generation (#Hidden units: 25, Input: training data, Error: 10.50%).

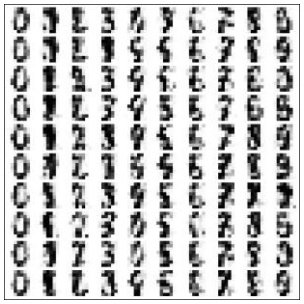

Figure 10. Output by the best autoencoder in the last generation (#Hidden units: 25, Input: test data, Error: 11.76%).

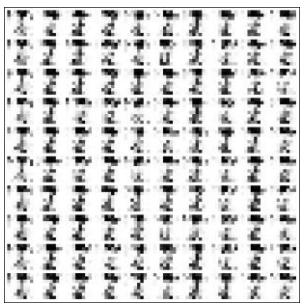

Figure 11. Output by the best autoencoder in the first generation (#Hidden units: 36, Input: training data, Error: 26.76%).

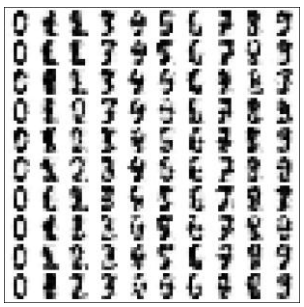

Figure 12. Output by the best autoencoder in the last generation (#Hidden units: 36, Input: training data, Error: 10.01%).

As these figures show, the GA could evolve autoencoders that reproduced the training and test data better as the autoencoders was equipped with more hidden units. A clear trade-off was observed between the reproduction accuracy and the encoding efficiency. Table 1 shows the ratio of the number of hidden units to the number of input units. An autoencoder with smaller hidden units includes smaller parameters and encodes the 64 dimensional input data to less dimensional code.

Some techniques such as the dropout [27-29] and the sparse encoding [30-31] are proposed to train deep neural networks by the BP. These techniques may also contribute to the evolutionary training. Future work includes the application and evaluation of these techniques.

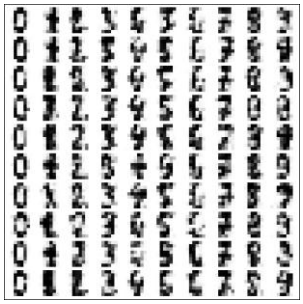

Figure 13. Output by the best autoencoder in the last generation (#Hidden units: 36, Input: test data, Error: 11.30%).

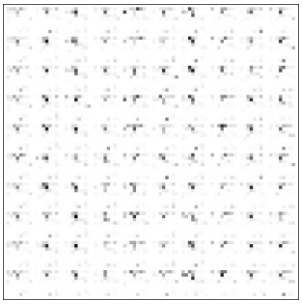

Figure 14. Output by the best autoencoder in the first generation (#Hidden units: 49, Input: training data, Error: 29.36%).

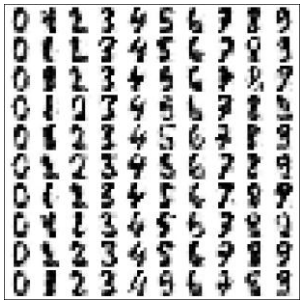

Figure 15. Output by the best autoencoder in the last generation (#Hidden units: 49, Input: training data, Error: 8.41%).

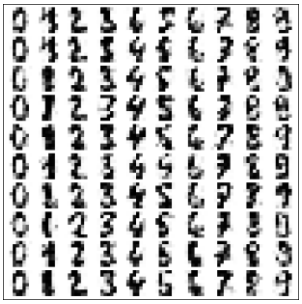

Figure 16. Output by the best autoencoder in the last generation (#Hidden units: 49, Input: test data, Error: 9.59%).

TABLE I. THE RATIO OF THE NUMBER OF HIDDEN UNITS TO THE NUMBER OF INPUT UNITS

#Hidden units	16	25	36	49
Ratio	16/64 = 25.0%	25/64= 39.1%	36/64= 56.1%	49/64= 76.6%

V. CONCLUSION AND FUTURE WORK

The author adopted a genetic algorithm to the evolutionary training of an autoencoder. The experimental results with the data of handwritten digits showed that the genetic algorithm could evolve autoencoders so that it reconstructed both of the training and test data well if the number of hidden units was

enough. Because an autoencoder is the component of a deep neural network known as a stacked autoencoder, the results reported in this paper indicate that a genetic algorithm will contribute to the training of stacked autoencoders so that they can work well on tasks such as pattern recognition. Evolutionary algorithms other than genetic algorithms have been proposed, e.g., evolution strategies and differential evolutions. Some swarm intelligence algorithms can also be adopted to the stochastic training of deep neural networks, e.g., particle swarm optimizations and artificial bee colony algorithms. The author will evaluate, compare and improve the capabilities of these evolutionary/swarm algorithms in the training of deep neural networks.

REFERENCES

[1] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507.

[2] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.

[3] Boureau, Y. L., & Cun, Y. L. (2008). Sparse feature learning for deep belief networks. In Advances in neural information processing systems (pp. 1185-1192).

[4] Sutskever, I., & Hinton, G. E. (2008). Deep, narrow sigmoid belief networks are universal approximators. Neural Computation, 20(11), 2629-2636.

[5] Bengio, Y. (2009). Learning deep architectures for AI. Foundations and trends in Machine Learning, 2(1), 1-127.

[6] Larochelle, H., Bengio, Y., Louradour, J., & Lamblin, P. (2009). Exploring strategies for training deep neural networks. Journal of Machine Learning Research, 10(Jan), 1-40.

[7] Tan, C. C., & Eswaran, C. (2010). Autoencoder Neural Networks: A Performance Study Based on Image Reconstruction, Recognition and Compression. LAP Lambert Academic Publishing.

[8] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11(Dec), 3371-3408.

[9] Salakhutdinov, R., & Hinton, G. (2012). An efficient learning procedure for deep Boltzmann machines. Neural computation, 24(8), 1967-2006.

[10] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[11] Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8), 1798-1828.

[12] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[13] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural Networks, 61, 85-117.

[14] Zhang, S., Choromanska, A. E., & LeCun, Y. (2015). Deep learning with elastic averaging SGD. In Advances in Neural Information Processing Systems, 685-693.

[15] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1988). Learning representations by back-propagating errors. In Neurocomputing: foundations of research, James A. Anderson and Edward Rosenfeld (Eds.). MIT Press, Cambridge, MA, USA, 696-699.

[16] Williams, D. R. G. H. R., & Hinton, G. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-538.

[17] Yao, X. (1999). Evolving artificial neural networks. Proceedings of the IEEE, 87(9), 1423-1447.

[18] Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. Evolutionary Intelligence, 1(1), 47-62.

[19] Montana, D. J., & Davis, L. (1989). Training feedforward neural networks using genetic algorithms. IJCAI, 89, 762-767.

[20] Sexton, R. S., Dorsey, R. E., & Johnson, J. D. (1998). Toward global optimization of neural networks: a comparison of the genetic algorithm and backpropagation. Decision Support Systems, 22(2), 171-185.

[21] Sexton, R. S., & Gupta, J. N. (2000). Comparative evaluation of genetic algorithm and backpropagation for training neural networks. Information Sciences, 129(1), 45-59.

[22] Örkcü, H. H., & Bal, H. (2011). Comparing performances of backpropagation and genetic algorithms in the data classification. Expert systems with applications, 38(4), 3703-3709.

[23] Joy, C. U. (2011). Comparing the Performance of Backpropagation Algorithm and Genetic Algorithms in Pattern Recognition Problems. International Journal of Computer Information Systems, 2(5), 7-12.

[24] Che, Z. G., Chiang, T. A., & Che, Z. H. (2011). Feed-forward neural networks training: A comparison between genetic algorithm and back-propagation learning algorithm. International Journal of Innovative Computing, Information and Control, 7(10), 5839-5850.

[25] Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, *3*(2), 95-99.

[26] Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. Machine learning, 3(2), 95-99.

[27] Eshelman Larry, J., & Schaffer David, J. Real-coded Genetic Algorithms and Interval-Schemata. Foundations of Genetic Algorithms 2, 187-202, 1993

[28] Dahl, G. E., Sainath, T. N., & Hinton, G. E. (2013, May). Improving deep neural networks for LVCSR using rectified linear units and dropout. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on (pp. 8609-8613). IEEE.

[29] Gal, Y., & Ghahramani, Z. (2016, June). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In International Conference on Machine Learning (pp. 1050-1059).

[30] Ba, J., & Frey, B. (2013). Adaptive dropout for training deep neural networks. In Advances in Neural Information Processing Systems (pp. 3084-3092).

[31] Boureau, Y. L., & Cun, Y. L. (2008). Sparse feature learning for deep belief networks. In Advances in neural information processing systems (pp. 1185-1192).

[32] Ng, A. (2011). Sparse autoencoder. CS294A Lecture notes, 72(2011), 1-19.