Comparison of Auto-Encoder Training Algorithms

Teodor Boyadzhiev[1], Stela Dimitrova[1] and Simeon Tsvetanov[1]

[1] Sofia University, Faculty of Mathematics and Informatics,
Sofia, Bulgaria
teodorvb@gmail.com, stelasd@uni-sofia.bg, tsvetanov@fmi.uni-sofia.bg

Abstract. Training of deep neural networks is difficult due to vanishing gradients. Therefore, a pre-training procedure based on restricted Boltzmann machines is suggested to resolve this problem. However, new developments in deep learning aim to resolve the problem with vanishing gradients by using rectifier linear units (ReLU). This study compares the performance of a RBM pre-trained auto-encoder with sigmoid activations to the performance of auto-encoder with ReLU activation. The results showed that the ReLU auto-encoder achieved better reconstruction and saved training time, since it doesn't require pre-training.

Keywords: Deep Learning · Restricted Boltzmann Machine· ReLU · Auto-Encoder

1 Introduction

Auto-Encoders are deep feed forward neural networks used for dimensionality reduction [1]. The advantage of the auto-encoder over methods such as Principal Component Analysis (PCA) is that they can capture non-linear dependencies.

Since the auto-encoders have many hidden layers, it is hard to train using error back-propagation [2] [3]. The reason for such difficulties is that the gradients are vanishing. Therefore, unsupervised pre-training procedure based on restricted Boltzmann machines (RBM) is used [3] [4]. The RBMs are stochastic neural networks which can learn the probability distribution of input data [5] [6]. RBMs are used for feature extraction [7] [8] [9] [10], and anomaly detection [11].

A study by Tan et al. [12] compared a stacked auto-encoder, a stacked auto-encoder with RBM pre-training, and an auto-encoder with RBM pre-training. The results showed that the best performance was achieved by the stacked RBM pre-trained auto-encoder. Nuha et al. [13] showed pre-training with RBM increases the training time even for auto-encoder with just one hidden layer.

This RBM pre-training technique is used in different applications [14] [15] [16] [17] [18] [19]. Also, RBMs are used as part of the error back-propagation training at each iteration [20].

Recent developments in deep learning, however, have resolved the problem with vanishing gradients by using rectifier linear unit (ReLU) [21] and weight initialization techniques such as Xavier [22]. This allowed for very deep networks to be trained such as MobileNetV2 [23], Xception [24], UNet [25], SegNet [26]. A study by Zhou et al.

[27] in which stacked auto-encoders are compared with auto-encoders trained with only back-propagation also shows that the back-propagation algorithm results in better models than the greedy layer-wise training without fine-tuning.

The previous studies by Hinton [3], Tan [12], Nuha [13] and Zhou [27] use sigmoid activations. The sigmoid activation has derivative which has maximum value of $\frac{1}{4}$. This can cause the gradients to vanish during back-propagation.

This study aims to compare a deep auto-encoder trained by back-propagation with RBM pre-training to an auto-encoder of the same architecture trained directly with by back-propagation. It will try to address the problem of vanishing gradients by using ReLU activations. It will try to push the number of hidden layers in the auto-encoder further than the previous studies. Hinton and Tan used 7 hidden layers, Zhou 1 to 3, and Nuha just 1.

2 Network Architecture and Datasets

The auto-encoders are first compared on the MNIST dataset. This dataset consists of hand-written digits represented as gray-scale images with resolution of 28 by 28 pixels. It has 60000 training images and 10000 testing images. The comparison of the auto-encoders is done on the 10000 testing images.

The auto-encoders are fully connected and symmetric from the middle layer. Each auto-encoder has the same architecture, 784-512-256-128-64-32-64-128-256-512-784, which has 10 hidden layers.

With this architecture 3 auto-encoders are trained:

- **RBM + sigmoid** RBM pre-trained auto-encoder which has sigmoid activations.
- **sigmoid** Auto-encoder which is not RBM pre-trained and has sigmoid activations.
- **ReLU** Auto-encoder which is not RBM pre-trained and has ReLU activations.

The hidden layer with 32 units is called bottleneck. The part of the auto-encoder before the bottleneck is called encoder and the part after is called decoder. The auto-encoder in this study maps 784-dimensional space into 32-dimensional space.

2.1 RBM Pre-Training

The restricted Boltzmann machine is a stochastic neural network, which can learn the probability distribution of data. It has visible units and hidden units, where each visible unit corresponds to a dimension of the input data and each hidden unit is latent variable.

It is a special case of the Boltzmann machine in which the visible units only depend on the hidden units, and the hidden units only depend on the visible units.

The probability of visible values $\boldsymbol{v}$ and the hidden features $\boldsymbol{h}$ is

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \tag{1}$$

$$Z = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}, \tag{2}$$

where E is the energy function.

The RBMs used in the pre-training algorithm have continuous visible units and discrete hidden units. This type of RBM is known as Gaussian-Bernoulli RBM [28] and its energy function is

$$E(\boldsymbol{v},\boldsymbol{h}) = \sum_{i}^{V} \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_{j=1}^{H} b_j h_j - \sum_{i=1}^{V}\sum_{j=1} \frac{v_i}{\sigma_i} w_{ij} h_j \quad , \tag{3}$$

where $W \in R^{VxH}$ is the weight matrix, $\boldsymbol{a} \in R^V$ are the visible biases, $\boldsymbol{b} \in R^H$ are the hidden biases, V is the number of visible units, and H is the number of hidden units. Due to the conditional independence assumption in the RBM, the visible and hidden units can be updated simultaneously according to

$$p(v_i|\boldsymbol{h}) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp - \frac{1}{2\sigma_i^2}\left(v_i - a_i - \sigma_i \sum_j w_{ij} h_j\right)^2 \tag{4}$$

$$p(h_j|\boldsymbol{v}) = \frac{1}{1 + \exp - \left(\sum_i \frac{v_i}{\sigma_i} w_{ij} + b_j\right)} \quad . \tag{5}$$

For each layer of the encoder part a RBM is trained with visible units representing the input data for the layer and the hidden units the activation. Once the RBM is trained the activation probabilities of the hidden units are calculated for each input data-point, and these probabilities become the input data for the next RBM.

Finally the auto-encoder is initialized by setting the weights, and biases of the encoder part to the weights and the hidden biases of the corresponding RBM, and the weights and biases of the decoder part are set to the transposed weights and visible biases of the corresponding RBM.

In the last step the auto-encoder is trained with the error back-propagation algorithm.

2.2 Training of RBMs

Restricted Boltzmann machines are trained by minimizing the negative log-likelihood of the data, given the parameters of the RBM,

$$-\frac{1}{N}\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \sum_{\boldsymbol{h}} \frac{1}{Z} \exp - E\left(\boldsymbol{v}^{(n)}, \boldsymbol{h}\right) \quad , \tag{6}$$

where N is the size of the input data, and $\boldsymbol{v}^{(n)}$ is the nth sample of the input data. To minimize this function the derivative with respect to each parameter has to be calculated,

$$-\frac{1}{N}\frac{\partial}{\partial\theta_i}\mathcal{L}(\boldsymbol{\theta}) = < -\mathbb{E}\left[-\frac{\partial}{\partial\theta_i} E(\boldsymbol{v}^{(n)}, \boldsymbol{h}) | \boldsymbol{v}^{(n)}\right] + \mathbb{E}\left[-\frac{\partial}{\partial\theta_i} E(\boldsymbol{v},\boldsymbol{h})\right] >_{n=1}^{N} \quad , \tag{7}$$

where $\mathbb{E}[f(x,y)|x] = \sum_y p(y|x)f(x,y)$, and $<\bullet>$ is average. Since the two expectations are computationally difficult to estimate contrastive divergence [?] is used to approximate the derivative,

$$-\frac{1}{N}\frac{\partial}{\partial\theta_i}\mathcal{L}(\boldsymbol{\theta}) =< \frac{\partial}{\partial\theta_i}E(\boldsymbol{v}^{(n)}(0), \boldsymbol{h}^{(n)}(0)) - \frac{\partial}{\partial\theta_i}E(\boldsymbol{v}^{(n)}(k), \boldsymbol{h}^{(n)}(k)) >_{n=1}^{N} \tag{8}$$

where $\boldsymbol{v}^{(n)}(k)$ are the visible units sampled from $\boldsymbol{h}^{(n)}(k-1)$, and $\boldsymbol{h}^{(n)}(k)$ are the hidden units sampled from $\boldsymbol{v}^{(n)}(k)$. The visible units $\boldsymbol{v}^{(n)}(0)$ are set to the values of the nth data point.

Following [3], $k = 1$ and the values of the visible and simple units are not sampled from the RBM, but instead the values of the hidden units are set to the activation probabilities and the values of the visible units are set to the mean value of the Gaussian distribution. The standard deviation is set to 1 for each visible unit, and only the weights and biases are updated.

3 Methods

Each RBM was trained using the AdaMax algorithm with the default parameters and a batch size of 250. The initial parameters of the weights of the RBMs were drawn from Gaussian distribution, $N\left(0, \frac{2}{V+H}\right)$, (Xavier) and the biases are set to zero.

The first two RBMs (784x512 and 512x256) were trained for 400 iterations, the third (256x128) for 600, fourth (128x64) 1000, and the last (64x32) for 3000. The error function used is the mean squared error,

$$\mathcal{E} = \frac{1}{NV}\sum_{n=1}^{N}\sum_{i=1}^{V}(y_i(n) - \hat{y}_i(n))^2 \quad , \tag{9}$$

where N is the number of data points, $y_i(n)$ is the nth target, and $\hat{y}_i(n)$ is the output of the network for the nth input.

Then the auto-encoders were trained with the AdaMax algorithm with default parameters, for 20 iterations with batch size of 32. The auto-encoder **RBM + sigmoid** was initialized to the weights of the RBMs, and the other two auto-encoders were initialized with the Xavier initialization drawn from Gaussian distribution. The ReLU units have a maximum value of 1.

For the RBMs was used custom implementation of the AdaMax algorithm and the calculation of the derivatives. For the fine tuning of the auto-encoders was used TensorFlow 2.4.

4 Results

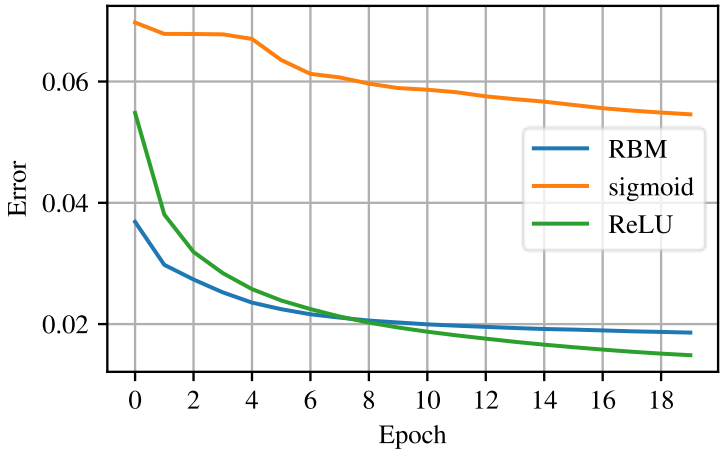

Fig. 1. Training error of the auto-encoders - RBM + sigmoid (Blue), sigmoid (Orange), and ReLU (Green)

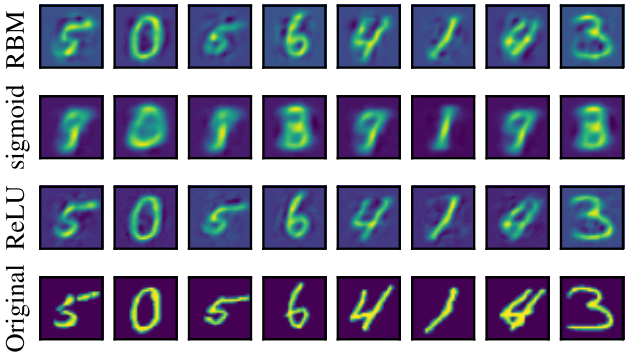

Fig. 2. An example of the reconstruction error of 8 samples of the MNIST dataset.

Figure 1 shows the training error for each auto-encoder. The network **RBM + sigmoid** is trained starts from the lowest error, due to the pre-training, but the error declines slower than **ReLU** and converges at higher error. The auto-encoder **sigmoid** starts at much higher error and the error declines very slow. It would require a lot more iterations to achieve good performance.

Figure 2 shows 8 randomly chosen samples of the MNIST dataset compressed and reconstructed with each of the auto-encoders. Best performing is the **ReLU** auto-encoder, and the worst is the **sigmoid** auto-encoder. The pre-trained auto-encoder (**RBM + sigmoid**) has similar performance as the **ReLU**.

5 Conclusion

The results showed that the RBM pre-training significantly improves the performance of the auto-encoder when using sigmoid activations, however the benefits of the ReLU activation offset the benefits of the RBM pre-training. The **ReLU** auto-encoder performed better than the **RBM + sigmoid** when trained the same number of iterations, but didn't need pre-training which is quite expensive. The pre-training required approximately $4.93 * 10^{12}$ multiplications.

The reason that the ReLU activation is better is that it has derivative of exactly 1 or 0, which helps with the problem of vanishing gradients. The maximum value of the sigmoid derivative is $\frac{1}{4}$ which causes the network to either be impossible to train or take too long time when many layers are used.

Using ReLU activations and Xavier initialization outperforms the RBM pre-training. Provided the additional computational cost, the pre-training becomes inefficient.

In future research can be explored whether the RBM pre-training can improve the performance of the convolutional auto-encoders.

Acknowledgement. The experiments described in this paper became possible thanks to the computing resources and technical support provided by UNITe project: https://unite-bg.eu/

References

1. M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," AIChE journal, vol. 37, p. 233–243, 1991.
2. S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber and others, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, A field guide to dynamical recurrent neural networks. IEEE Press, 2001.
3. G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," science, vol. 313, p. 504–507, 2006.
4. G. E. Hinton, S. Osindero and Y.-W. Teh, "A fast learning algorithm for deep belief nets," Neural computation, vol. 18, p. 1527–1554, 2006.
5. S. S. Haykin and others, Neural networks and learning machines/Simon Haykin., New York: Prentice Hall,, 2009.
6. P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," 1986.
7. N. Jaitly and G. Hinton, "Learning a better representation of speech soundwaves using restricted boltzmann machines," in 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2011.
8. H. B. Sailor, D. M. Agrawal and H. A. Patil, "Unsupervised Filterbank Learning Using Convolutional Restricted Boltzmann Machine for Environmental Sound Classification.," in InterSpeech, 2017.
9. J. Yang, J. Deng, S. Li and Y. Hao, "Improved traffic detection with support vector machine based on restricted Boltzmann machine," Soft Computing, vol. 21, p. 3101–3112, 2017.

10. J. Vrábel, P. Pořízka and J. Kaiser, "Restricted Boltzmann Machine method for dimensionality reduction of large spectroscopic data," Spectrochimica Acta Part B: Atomic Spectroscopy, vol. 167, p. 105849, 2020.
11. Y. Zhang, P. Peng, C. Liu and H. Zhang, "Anomaly detection for industry product quality inspection based on Gaussian restricted Boltzmann machine," in 2019 IEEE international conference on systems, man and cybernetics (SMC), 2019.
12. C. C. Tan and C. Eswaran, "Performance comparison of three types of autoencoder neural networks," in 2008 Second Asia International Conference on Modelling & Simulation (AMS), 2008.
13. H. Nuha, M. Mohandes and B. Liu, "Seismic data compression using auto-associative neural network and restricted Boltzmann machine," in SEG Technical Program Expanded Abstracts 2018, Society of Exploration Geophysicists, 2018, p. 186–190.
14. A. Pumsirirat and L. Yan, "Credit card fraud detection using deep learning based on auto-encoder and restricted boltzmann machine," International Journal of advanced computer science and applications, vol. 9, p. 18–25, 2018.
15. D. He, Q. Qiao, Y. Gao, J. Zheng, S. Chan, J. Li and N. Guizani, "Intrusion detection based on stacked autoencoder for connected healthcare systems," IEEE Network, vol. 33, p. 64–69, 2019.
16. A. M. Mahmoud, F. Alrowais and H. Karamti, "A Hybrid Deep Contractive Autoencoder and Restricted Boltzmann Machine Approach to Differentiate Representation of Female Brain Disorder," Procedia Computer Science, vol. 176, p. 1033–1042, 2020.
17. J. Li, Z. L. Yu, Z. Gu, W. Wu, Y. Li and L. Jin, "A hybrid network for ERP detection and analysis based on restricted Boltzmann machine," IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 26, p. 563–572, 2018.
18. T. Kuremoto, S. Kimura, K. Kobayashi and M. Obayashi, "Time series forecasting using restricted boltzmann machine," in International Conference on Intelligent Computing, 2012.
19. J. Qiao and L. Wang, "Nonlinear system modeling and application based on restricted Boltzmann machine and improved BP neural network," Applied Intelligence, vol. 51, p. 37–50, 2021.
20. M. Ma, C. Sun and X. Chen, "Deep coupling autoencoder for fault diagnosis with multimodal sensory data," IEEE Transactions on Industrial Informatics, vol. 14, p. 1137–1145, 2018.
21. X. Glorot, A. Bordes and Y. Bengio, "Deep sparse rectifier neural networks," in Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011.
22. X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the thirteenth international conference on artificial intelligence and statistics, 2010.
23. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018.
24. F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017.
25. O. Ronneberger, P. Fischer and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in International Conference on Medical image computing and computer-assisted intervention, 2015.

26. V. Badrinarayanan, A. Kendall and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," IEEE transactions on pattern analysis and machine intelligence, vol. 39, p. 2481–2495, 2017.
27. Y. Zhou, D. Arpit, I. Nwogu and V. Govindaraju, "Is joint training better for deep auto-encoders?," 2014.
28. A. Krizhevsky, G. Hinton and others, "Learning multiple layers of features from tiny images," 2009.