An Empirical Analysis of Non-linearities in Deep Networks

Jose Alves*
jalves7@uwo.ca
University of Western Ontario

ABSTRACT

Artificial neural networks have a fixed function for each unit within the network. Such functions, referred to as activation functions, are a non-linear mathematical function that impacts learning and performance of neural networks; making them a core component and consideration in a network design. Currently, the rectified linear unit (ReLu) activation is ubiquitous in Deep Learning as its drop in performance is unrivaled. We perform an empirical evaluation of activation functions to understand if the usage of ReLU still remains justified. Our work examines the performance of a state-of-the-art model, with varied activation functions, against MNIST - a benchmark dataset. We find that ELU activation function has the better speed of convergence and exceeds the performance of the ReLU function.

Index Terms: CS [Artificial Intelligence]: Computer Vision—Deep Learning;

1 INTRODUCTION

Neural networks (NN) have gone through a renaissance over the past half decade. NNs are extremely flexible function approximators but were notoriously difficult to train. Recent progress has decreased the difficulty of training such networks. As a result, outstanding achievements have occurred across various domains, such as image recognition [1], speech synthesis [2], and the defeat of human professionals at the game of Go [3]. The primary driver of this progress was the usage of Deep Learning (DL), where many layers are used in NNs trained on GPUs [4] – earning the name of a "deep neural network" (DNN).

A fundamental part of these DNNs is the choice of non-linear mathematical function, referred to as an activation function, used on each unit within the network. An activation function affects the function complexity a network can approximate, the speed of learning, training convergence, and overall quality of the final network predictions. Thus, selecting an activation function suited to the specific problem is an important consideration.

An activation function known as a Rectified Linear Unit (ReLU) has been the dominant function used; ReLUs have an empirically validated performance across varied domains and network configurations making it a safe first choice. Therefore, we wish to experimentally examine the performance of ReLUs and other activation functions on a standard benchmark dataset with a state of the art (SOTA) DNN architecture. We examine various properties of the learning process across each activation function such as rate of convergence, smoothness of convergence, final test, and training performance, and the quality of the learned parameters in the first layer of the network. Our main contribution is the examination of various axes of performance for various activation function and if the ReLU activation is still a reliable default choice.

The paper is organized as follows: Section 2 discusses related work, Section 3 provides background knowledge on neural networks and activation functions, Section 4 discusses the particular benchmark dataset chosen, Section 5 provides an overview of the experimental methodology, Section 6 covers experimental implementation details, and in Section 7 the experimental results are examined and discussed.

2 RELATED WORK

Xu *et al.* [5] evaluate the performance of four variations of rectified activation functions (ReLU, Leaky ReLU, Randomized ReLU, and Parametric ReLU) on a standard Convolutional Neural Network (CNN). Where Parametric ReLU (PReLU) adaptively learn the negative slope, Randomized ReLU (RReLU) randomly sample a negative slope between a bound, and Leaky ReLU has a fixed negative slope [6]. As both PReLU and RReLU are both closely related variations of LReLU and time constraints they are not evaluated in this paper.

Several papers that introduce new activation functions perform evaluations of different functions against a proposed function in their work [6–11]. In contrast, our work systematically investigates a broad range of activation functions including bounded functions such as sigmoid and tanh. Additionally, we discuss convergence properties throughout training – this is usually ignored in the previous works.

3 BACKGROUND

3.1 Neural Networks

Neural network refers to a statistical model that learns connection strengths between layers of units from data. A neural network is typically a feed forward graph, where each unit (node) applies a non-linear activation function to the weighted combination of its inputs. Equation 1 is the networks output of layer l of k units where $W \in \mathbb{R}^{M \times K}$ is the learned weight

*signifies equal contribution.

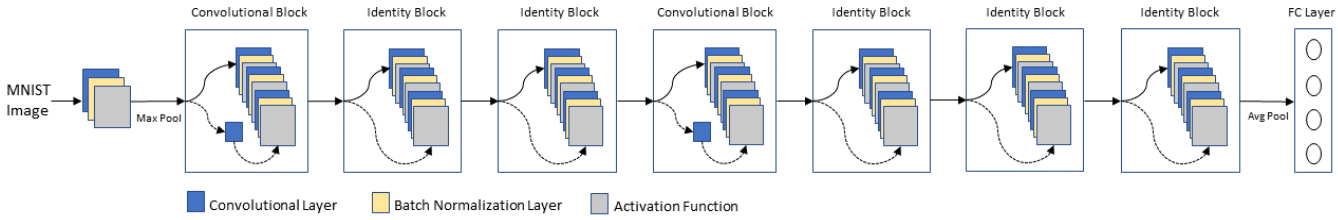

Figure 1: ResNet architecture with 25 convolutional layers implemented for this paper. In each block there are 3 convolutional layers in the main path. Skip connections are represented by dotted lines.

parameters, $b \in \mathbb{R}^{K\times 1}$ is a learned bias parameter, $x \in \mathbb{R}^{M\times 1}$ is the input data with dimension of M, and f is the activation function.

$$y_l = f(W^T x + b) \tag{1}$$

A network is constructed by feeding the output of a previous layer, x_{l-1}, into the next layer as its input. The network is trained using gradient descent and backpropagation to optimize a loss function $\mathscr{L}(x,\theta)$ where θ is all of the networks parameters.

3.2 Deep Learning

Deep learning (DL) has a long history which dates back to the 1940s, despite the term being used to represent an emerging technology. DL refers to neural networks with many layers each containing a high number of units. Due to its depth, a deep network is able to exploit patterns within the data and learns to build a feature hierarchy within the network [12]. This allows automatic feature discovery from data instead of relying on handcrafted features. The recent successes of deep learning are due to several factors: increase in computation power (GPUs), large amounts of data, improvements to parameter initialization methods, and activation functions.

3.3 Activation Functions

Activation functions are responsible for introducing nonlinearity in DNNs allowing complex functions to be represented. There are different types of activation functions and the selection of the best function for a model is an essential step in the implementation of a deep neural network.

3.3.1 Sigmoid

The sigmoid function is widely applied in machine learning: it is a nonlinear function and the simplicity of calculating its derivative. Sigmoid is a mathematical function that accepts all real numbers as input and returns simple probabilities between 0 and 1 [13]. One important example of a classic sigmoidal function is the logistic regression function defined by the formula,

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{2}$$

3.3.2 Hyperbolic tangent Function - tanh

Hyperbolic tangent function, or simply tanh, is also sigmodal with the same characteristics of a logistic regression function but has an important distinctive feature: its normalized range is in between $[-1,1]$. Generally, tanh is preferred to the sigmoid because tanh can deal better with strong-negative inputs. Also, note that the hyperbolic tangent function is a rescaled form of the sigmoid function and it can be expressed by,

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

3.3.3 Rectified Linear Unit - ReLU

Currently, the Rectified Linear Unit function (ReLU) is the most popular activation function with a simple mathematical definition:

$$relu(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{4}$$

ReLU functions require less computational power and have been shown to perform better than either sigmoid or tanh activations functions in DNN architectures [14–16]. Additionally, another distinguishing aspect of the ReLU function is that it solves the vanishing gradient problem – a common problem with sigmoid functions. Vanishing gradient problem occurs in DNNs when gradients approach zero causing the learning process to slow or causing stability issues.

Currently, many variations of the ReLU family of functions exist. We only describe variation used experimentally within our work.

3.3.4 Leaky Rectified Linear Unit - Leaky ReLU

The Leaky ReLU is an adaptation of the ReLU function to solve the problem of dead neurons. ReLU functions can generate dead neurons which are nodes that not contribute any activation signal to the network. A Leaky ReLU solves this issue by giving a small slope when $x <= 0$, therefore, the gradient will not die and it can be recovered during the training.

$$\text{leaky relu}(x) = \begin{cases} x & \text{for } x > 0 \\ 0.01x & \text{otherwise} \end{cases} \tag{5}$$

3.3.5 Exponential Linear Unit - ELU

Another approach to solving the vanishing problem is the Exponential Linear Unit (ELU) function. Empirical results by Clevert *et al.* [6] show that this function performs well in DNN architectures with more than 5 layers.

$$elu(x,\alpha) = \begin{cases} x & \text{for } x \geq 0 \\ \alpha(exp(x) - 1 & \text{otherwise} \end{cases} \tag{6}$$

3.4 MNIST Dataset

The Modified National Institute of Standards and Technology dataset (MNIST) is a dataset of images composed of handwritten digits from 0 to 9. The handwriting recognition application is a computer vision task that many consider solved, however, it has remained relevant for decades and serves as a basic benchmark. MNIST consists of 70,000 labeled 28x28 pixel grayscale images of handwritten digits. Each digit image is normalized in size and location within the 28x28 domain, so no time is needed to prepare or clean the data. The objective of MNIST is to accurately identify the correct classification of a digit, 0 to 9, from its image.

It should be noted that the performance of NNs on MNIST surpasses other popular statistical methods [17].

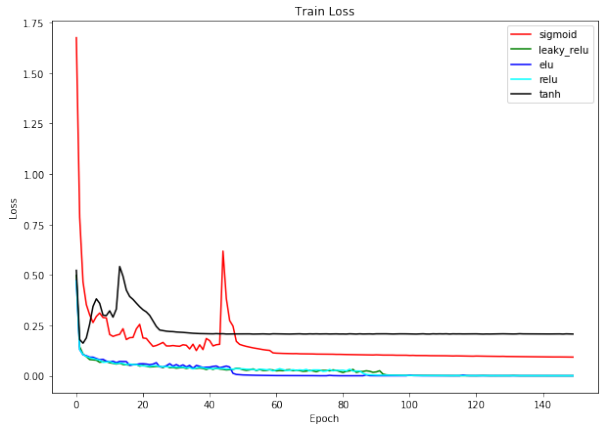

Figure 2: Training loss of the network for each activation function.

4 Methodology

Our experiments vary the activation function while holding all other variables constant in our DNN model and training on MNIST. Within this work we use Residual Networks (Resnet) proposed by He *et al.* [18] as our DNN model. Resnets hold the current state of the art performance on other challenging image datasets such as ImageNet [1]. The structure of Resnets address the degradation problem found in DNNs: adding more layers to DNNs causes the accuracy of the network to saturate, when the loss starts to converge, causing higher training error [19, 20]. Resnets architectures make heavy use of joined convolutional and batch normalization layers in what is refereed to as "blocks" [21]. The distinction between

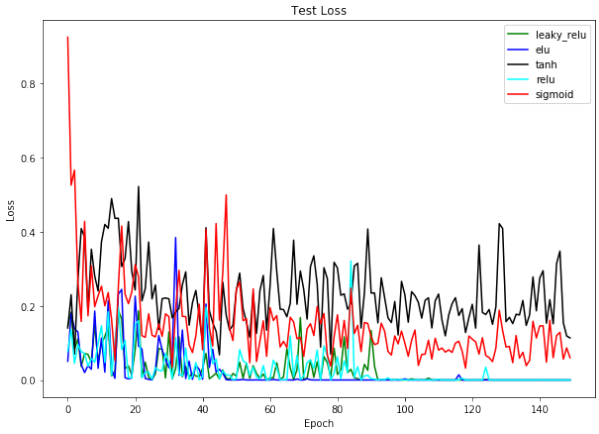

Figure 3: Test loss of the network over the training period.

each block type is based on input to output dimensionality; an identity block keeps dimensionality consistent while a convolutional block reduces the output dimensionality. Each block is comprised of a convolutional layer followed by a batch normalization layer and finally an activation function. An additional architectural modification by He *et al.* was to use short circuits between later and earlier layers, known as skip connections, to improve information flow across deeper layers. Skip connections help improve learning as gradient signals have shortcuts to flow to earlier layers.

An additional factor in our decision of both dataset and model was time: training a larger newer dataset, such as Imagenet [1], against all activation functions is prohibitive as the training time per model is in the range of days.

5 Experiment Implementation Details

The Resnet implementation used within our work is a reduced version of He *et al.* [18]. Our model is comprised of 25 layers with block and layer configurations shown in Figure 1.
The networks weights were initialized with Kaiming Normal [22] with a gain of 0.25. All biases were set to 0.1 as we wanted to ensure all units within the network initially contribute to the networks output. Such a scenario is possible if the weights were to be initialized to 0.0, with a bias of 0.0 – causing the unit to not activate regardless of its input.

The MNIST dataset was split into a training and testing subsets of 60,000 and 10,000 samples respectively. This denomination between training and test subsets is standard within the field of computer vision [1, 23]. Our network was trained for 150 epochs, where an epoch is a full iteration over the training dataset, using the Adam optimizer [24]. For Adam we used an initial learning rate of 0.1, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 1e-8$. The networks object was to minimize the cross-entropy loss, expressed by equation (7), between the image labels and network predictions.

$$H(p,q) = -\sum_{i=1}^{n} p(x_i) \log q(x_i). \tag{7}$$

The learning rate was annealed by a factor of 10.0 if the

training loss of the network had not decreased for 10 epochs. This greatly helps convergence of DNNs [25].

6 Experimental Results

In this section, the experimental results obtained with the performance evaluation of ReLU, Leaky ReLU, ELU, sigmoid, and tanh activation functions on MNIST dataset will be presented.

6.1 Training and Test performance

Activation Function	Training Loss	Test Loss
ELU	2.76e-6	4.15e-9
ReLU	6.08e-5	1.67e-4
Leaky ReLU	3.2e-5	1.05e-5
Sigmoid	0.0925	0.06
Tanh	0.21	0.1134

Table 1: Final values for training and test learning curves on the final epoch.

For both training and test graphs, Figure 2 and 3, it is clear that ELU, ReLU, and Leaky ReLU outperform sigmoid and tanh activation functions. There is no evidence of overfitting occurring on any experimental run. From Table 1 and 2 we see that the ELU function outperforms other activation functions. The precision between ELU, ReLU, and Leaky ReLU are extremely close.

Activation Function	Precision
ELU	99.43
ReLU	99.07
Leaky ReLU	99.20
Sigmoid	96.78
Tanh	93.63

Table 2: Final precision values for test dataset on the final epoch.

6.2 Convergence Properties

We are interested in several properties throughout the learning phase: speed asymptotic convergence of its loss, the smoothness of this convergence, and stability around the asymptotic point. Each property is considered on both the training and test loss graphs seen in Figure 2 and Figure 3 respectively.

The sigmoid and tanh training curves experience large oscillations earlier in training indicating large weight and gradient updates are occurring. The tanh activation function appears to be unstable early in training but gains stability as the number of iterations increase. While the sigmoid function has instability for twice as many iterations before finally settling. This instability of the sigmoid and tanh function could be due to both being bounded functions between $[0, 1]$ and $[-1, 1]$ respectively. Interestingly, the tanh function can be seen as a scaled version of the sigmoid, along with the y-axis, by a scale of $1/2$; such scaling may be an indication of why the sigmoid takes approximately twice as many gradient updates to stabilize – the sigmoid function has weaker gradients than the tanh function [26]. This range affects both the magnitude and direction of gradient updates to the network. If we compare the ReLU activation function, which is bounded between $[0, +\infty)$, we can conclude that range supported by both the sigmoid and tanh function might be too restrictive to allow gradients to flow backward through the network. This affects both the speed and stability of convergence during learning.

Immediately, the observation can be made that ReLU, Leaky ReLU, and ELU functions share the same characteristics during learning. All functions experience small amounts of variance up to some period of time, before finally dropping sharply, and then quickly converging to their asymptotic value. Initially, the ELU activation function experiences oscillations but is the first to drop sharply towards its asymptotic value. We can see that the ELU activation does appear to have small occasionally spikes after convergence, more than any other function including tanh and sigmoid, so we can consider it to be the least stable. The ReLU and Leaky ReLU functions appear to be nearly identical in smoothness and stability of convergence; though the ReLU function dips slightly earlier than the Leaky ReLU.

Finally, when considering the test dataset the majority of characteristics observed on the training dataset hold. Though, the stability of the ReLU function appears to have much higher variance and instability before convergence. In comparison to the Leaky ReLU function we can conclude that while the ReLU function may converge slightly faster the slight negative slope of the Leaky ReLU allows better weight corrections to be made such that there is greater stability – with a small cost to speed of convergence (roughly 4% faster). It appears that all the activation functions have little impact on the stability of the network around its asymptotic point. In conclusion, the ELU function appears to have the best properties overall. The only apparent downside is it has mild instability after convergence to its asymptotic value.

6.3 Weight Parameter Distributions

Before training begins the network parameters are initialized by sampling from a Gaussian distribution with zero mean and standard deviation set according to He *et al.* [22]. Within this section, we are interested in understanding how the distribution of parameter values changes over time as learning progresses. This gives us insight into how the network learns over time and how each activation function affects the initial parameter values after repeated batch gradient updates. We focus only on the first convolutional layer's weights as there are simply too many other layers to examine within the scope of this paper.

In Figure 4 the distribution of the parameters are shown over time as learning progresses. Upon inspection, it becomes

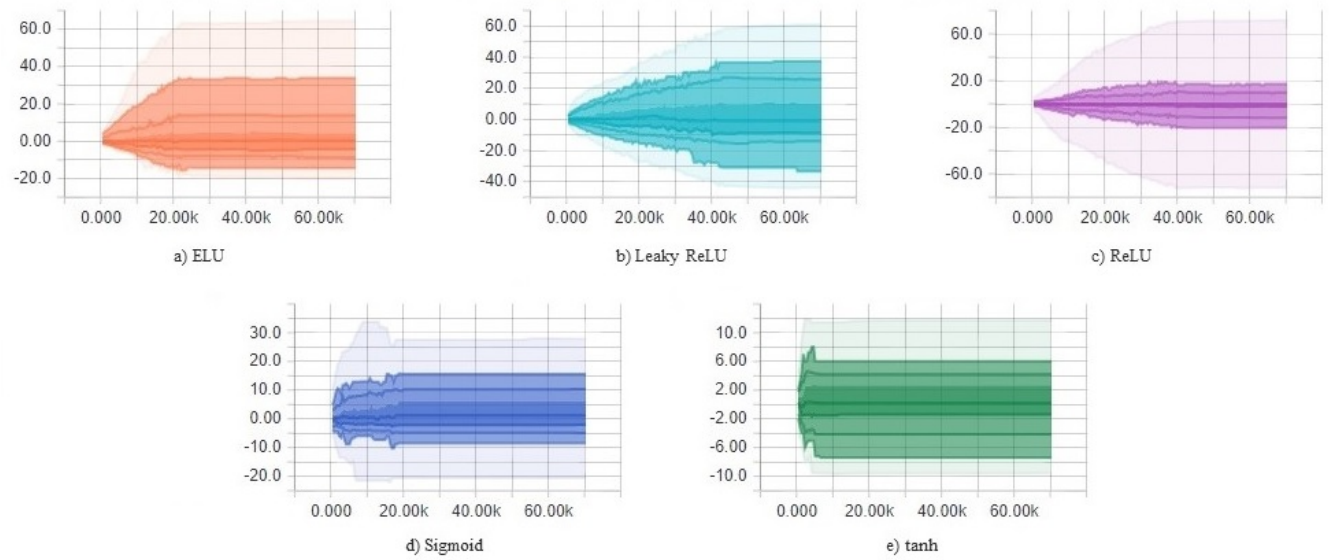

Figure 4: Distribution of weight parameters over time from the first convolutional layer in the network.

apparent that ELU, ReLU, and Leaky ReLU functions distribution continues to change overtraining; conversely the sigmoid and tanh functions appear to stop shortly after training begins. More generally, each distribution rapidly increases in variance centered around zero and then stabilizes. From Figure 4d) and 4e) we can see that approximately the largest magnitudes for both sigmoid and tanh functions are $\{-20, 30\}$ and $\{-10, 10\}$ respectively. While ELU, ReLU, and Leaky ReLU functions are $\{-20, 60\}$, $\{-60, 60\}$, and $\{-40, 60\}$ respectively. The large difference in magnitudes is most likely due to the nature of each function grouping; tanh and sigmoid are bounded functions with limited gradient magnitudes while ELU, ReLU, and Leaky ReLU functions are singly or doubly unbounded allowing much greater gradient magnitudes. With deeper networks bounded activation functions will quickly saturate at higher layers as all the gradient information from lower layers passes backward. As sigmoid and tanh functions are bounded they quickly saturate causing slower convergence.

The opposite happens with the unbounded functions as the gradient returning from lower layers cannot saturate enabling faster learning and continual weight adjustments. The nature of the bounded and unbounded functions, in conjunction to our observations of distribution drift discussed in Section 6.2, indicate that unbounded functions are preferred for deep networks as they allow gradient information to be propagated from deeper layers as training progresses.

7 Future Work

We believe it would be fruitful to pursue such an analysis in other subfields where DNNs are prolific such as Reinforcement Learning, Natural Language Processing, and Speech Synthesis. The activation function chosen affects the convergence properties of DNNs and in fields such as Reinforcement Learning having a better understanding of stability would be valuable.

Additionally, future work should perform multiple training runs with different random seeds such that the parameters of the network vary between training iterations. Doing so would provide confidence bounds on the activation functions performance. This was omitted from our work as it is extremely time intensive.

8 Conclusion

We have shown that activation functions impact the performance of a DNNs on various axes. Through our analysis we have the following conclusions: for greater stability during training it is best to use a singly or doubly unbounded function, such as ELU or ReLU; the greatest speed of convergence occurred by using the ELU activation function; there is only a small difference between Leaky ReLU and ReLU activation functions, using the ReLU activation might be advantageous as its faster to compute the derivative; and finally we see that bounded functions cause slower learning in very deep networks.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[2] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. "Wavenet: A generative model for raw audio". *arXiv preprint arXiv:1609.03499*, 2016.

[3] Fei-Yue Wang, Jun Jason Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Xiaoxiao Dai, Jie Zhang, and Liuqing Yang. "Where does AlphaGo go: from church-turing thesis to AlphaGo thesis and beyond". *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120, 2016.

[4] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". *Neural networks*, 61:85–117, 2015.

[5] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. "Empirical evaluation of rectified activations in convolutional network". *arXiv preprint arXiv:1505.00853*, 2015.

[6] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". *arXiv preprint arXiv:1511.07289*, 2015.

[7] Xiaoheng Jiang, Yanwei Pang, Xuelong Li, Jing Pan, and Yinghong Xie. "Deep neural networks with Elastic Rectified Linear Units for object recognition". *Neurocomputing*, 2017.

[8] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. "Learning activation functions to improve deep neural networks". *arXiv preprint arXiv:1412.6830*, 2014.

[9] Suo Qiu and Bolun Cai. "Flexible Rectified Linear Units for Improving Convolutional Neural Networks". *arXiv preprint arXiv:1706.08098*, 2017.

[10] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Swish: a Self-Gated Activation Function". *arXiv preprint arXiv:1710.05941*, 2017.

[11] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In *Proc. ICML*, volume 30, 2013.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Jun Han and Claudio Moraga. "The influence of the sigmoid function parameters on the speed of backpropagation learning". *From Natural to Artificial Neural Computation*, pages 195–201, 1995.

[14] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] Li Deng. "The MNIST database of handwritten digit images for machine learning research [best of the web]". *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[20] Kaiming He and Jian Sun. "Convolutional neural networks at constrained time cost". In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5353–5360, 2015.

[21] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In *International Conference on Machine Learning*, pages 448–456, 2015.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[24] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980*, 2014.

[25] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[26] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient backprop". In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.