The Dying ReLU Problem: A Mathematical and Experimental Study of Permanent neuron inactivity.

Author: Amila Samaranayake Kulathunga Mudiyansela Gedara (**Student ID** - 24082461)

Course: Machine Learning and Neural Network – MSc Data Science

Reading Time: 10–12 minutes

Word Count: ≈ 2,100 words (excluding references and code)

GitHub Repository: https://github.com/amila-samaranayake/dying-relu-analysis.git

1. Introduction

Activation functions make neural networks non-linear and allowing them to learn complicated patterns. ReLU is extensively used because it is simple, efficient, and avoids the vanishing-gradient difficulties associated with sigmoid and tanh (Nair and Hinton, 2010; Goodfellow, Bengio, and Courville, 2016).

Nevertheless, the ***dying ReLU*** **problem**, in which neurons receive zero gradient and output zero for all inputs, can affect ReLU. These neurons stop updating and become permanently inactive, reducing the network's learning capacity (Maas, Hannun, and Ng, 2013).

The tutorial explains the mathematical cause of this failure and demonstrates it experimentally using a deep MLP trained on MNIST with explicit dead-neuron tracking. Additionally, we illustrate how Leaky ReLU and ELU mitigate the issue by preserving non-zero gradients in the negative input region (Clevert, Unterthiner and Hochreiter, 2016).

2. Mathematical Preliminaries

This section explains the key mathematical principles essential to comprehend the fading ReLU issue. We begin by defining the ReLU activation function, then briefly examining gradient descent and backpropagation, and then explaining why zero gradients are particularly harmful for neural network training.

2.1. The Rectified Linear Unit (ReLU)

The ReLU is one of the most basic and extensively used activation functions in deep learning. It is defined as follows:

$$\mathrm{ReLU}(x) = \max(0, x)$$

Where $x$is the pre-activation input to a neuron, given by:

$$x = \mathbf{w}^\top \mathbf{z} + b$$

Here, $\mathbf{w}$ represents the weight vector, $\mathbf{z}$ is the input feature vector, and $\boldsymbol{b}$ is the bias term.

Once the input is positive ReLU outputs it directly. Otherwise, it outputs zero. This simple rule introduces non-linearity while remaining computationally efficient. Unlike sigmoid and tanh, ReLU avoids saturation in the positive region, allowing stronger gradient flow during training (Nair and Hinton, 2010; Goodfellow, Bengio, and Courville, 2016).

2.2. Gradient Descent and Backpropagation

G**radient-based optimisation**, most frequently **gradient descent** versions, is used to train neural networks. Training aims to minimize a given loss function $\mathcal{L}$ by iteratively changing the network's parameters. For a single weight, a general gradient descent update is provided by:

$$w_{t+1} = w_t - \eta \frac{\partial \mathcal{L}}{\partial w_t}$$

Where:

- $w_t$is the weight at iteration t,
- $\eta$is the learning rate,
- $\frac{\partial \mathcal{L}}{\partial w_t}$is the gradient of the loss with respect to the weight.

The backpropagation method, which uses the calculus chain rule to efficiently transmit error signals from the output layer back through each hidden layer, is used to calculate the gradients. The gradient for a neuron with pre activation $\boldsymbol{x}$ and activation $\boldsymbol{a}$ could be expressed as follows:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial x} \cdot \frac{\partial x}{\partial w}$$

This formula demonstrates how the derivative of the activation function, $\frac{\partial a}{\partial x}$. directly affects the gradient. The whole gradient collapses and learning for that neuron essentially stops if this derivative gets extremely tiny or precisely zero (Goodfellow, Bengio, and Courville, 2016).

2.3. Why Zero Gradients Are Dangerous

Maintaining non-zero gradients is essential for effective neural network training, as vanishing gradients slow or completely halt learning. This problem traditionally affected deep networks using sigmoid and tanh activations. ReLU partially solved this by allowing gradients to pass unchanged for positive inputs, since its derivative is one in this region (Nair and Hinton, 2010). However, ReLU introduces a new risk: gradients become exactly zero for negative inputs.

When a gradient becomes exactly zero, the corresponding weight update becomes:

$$\Delta w = -\eta \cdot 0 = 0$$

This indicates that there is no longer any updating of the parameter. A neuron's derivative stays 0, and it is theoretically removed from the learning process if it reaches a condition in which it generates zero output for all inputs. The dying ReLU issue, which is explicitly analyzed in the next section, is based on this condition.

3. Mathematical Mechanism of the Dying ReLU Problem

This section provides a mathematical explanation for why ReLU neurons may become permanently dormant following training. After analyzing the ReLU function's derivative and connecting it to gradient-based learning, we determine the training conditions that cause permanent neuronal death.

3.1. ReLU Derivative and Gradient Flow

The function that activates ReLU is defined as follows:

$$\mathrm{ReLU}(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Where $x = \mathbf{w}^\top \mathbf{z} + b$is the pre-activation of a neuron. The derivative of ReLU with regard to its input is:

$$\frac{d}{dx}\mathrm{ReLU}(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Learning is directly impacted by this derivative. Because the gradient of a neuron operating in the **active area** (x>0) is exactly one, mistakes can propagate backward without attenuation. However, the neuron's gradient becomes precisely zero as it moves into the **inactive area** (x≤0). No gradient signal can go through the neuron during backpropagation in this condition (Goodfellow, Bengio, and Courville, 2016).

3.2. Gradient Descent and the Onset of Neuron Death

Gradient-based optimization is used to train neural networks. For a single parameter, a general gradient descent update is given by:

$$w_{t+1} = w_t - \eta \frac{\partial \mathcal{L}}{\partial w_t}$$

Where η is the learning rate, and $\mathcal{L}$ is the loss function. Using the chain rule, the gradient of the loss with respect to the weight can be written as:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial x} \cdot \frac{\partial x}{\partial w}$$

For a ReLU neuron operating in the inactive regime, we have:

$$\frac{\partial a}{\partial x} = 0$$

And therefore:

$$\frac{\partial \mathcal{L}}{\partial w} = 0$$

Substituting this into the gradient descent update rule yields:

$$w_{t+1} = w_t$$

This depicts that **once a ReLU neuron enters the inactive regime, its weights and bias stop updating completely**. The neuron becomes permanently inactive if this state is true for every training sample. The idea of a **dead neuron** is therefore formalized.

3.3. Conditions That Trigger the Dying ReLU State

A neuron becomes permanently dead when its pre-activation satisfies:

$$x = \mathbf{w}^\top \mathbf{z} + b \leq 0 \forall \mathbf{z}$$

Several training elements might cause this condition:

(i) Large Learning Rates

A single gradient update may force the neuron's weights and bias from the active regime into the negative regime for all subsequent inputs if the learning rate η is too high. After that, the neuron's output is always 0, making it unable to learn anything else.

(ii) Negative Bias Drift

During the early stages of training, bias terms can drift towards large negative values. Since the bias is added to each pre-activation, a sufficiently negative bias ensures:

$$\mathbf{w}^\top \mathbf{z} + b < 0$$

For all inputs **z**, forcing the neuron into permanent inactivity.

(iii) Poor Weight Initialisation

When training begins, the majority of pre-activations may lie in the negative domain due to improper weight initialization. ReLU networks need variance-preserving initialization techniques to avoid early activation collapse, as demonstrated by He et al. (2015). A sizable portion of neurons may start training around the dead state in the absence of such initialization.

3.4. Why the Dying ReLU State Is Permanent

Once a neuron satisfies:

$$a(x) = 0 \text{ and } \frac{d}{dx}\text{ReLU}(x) = 0$$

For all training samples, it loses its mathematical connection to the optimization process. Neither the neuron's weights nor its bias can be altered by additional training since no gradient passes backward through it. Thus, the neuron is **permanently dead** under ordinary gradient descent.

Since this behavior is specific to activation functions with an exactly zero derivative over a finite input region, modified rectifiers like Leaky ReLU and ELU effectively prevent neuron death by maintaining non-zero gradients for negative inputs (Maas, Hannun, and Ng, 2013; Clevert, Unterthiner, and Hochreiter, 2016). Sigmoidal activations do not exhibit this permanent failure mode.

4. Simulation and Experimental Setup

The experimental setup for simulating and analyzing the dying ReLU issue is described in this section. The goal is to provide training settings that increase the likelihood of neuron loss and to quantify it.

4.1. Dataset

The **MNIST dataset,** which consists of 60,000 training and 10,000 test pictures of handwritten digits (0–9), each measuring 28 by 28 pixels, is used in all experiments (Goodfellow, Bengio, and Courville, 2016). Images were flattened into 784-dimensional input vectors after being normalized to the interval [0, 1].

4.2. Network Architecture

To emphasize fading ReLU behavior, a **deep multilayer perceptron (MLP)** with five hidden layers was employed. There are 256 neurons in each buried layer. For multi-class classification, the output layer employs a softmax activation. In the baseline experiment, ReLU activations are used in all hidden layers.

Because dying ReLU effects worsen with increasing network depth, this depth was used (Maas, Hannun, and Ng, 2013).

4.3. Training Configuration

Training was performed using the **Adam optimiser** with a purposefully **high learning rate of 0.005** to increase the likelihood of neuron death. **Sparse categorical cross-entropy** was used as the loss function. A **batch size of 128** was used to train the models across **12 epochs**. To prevent hiding unstable activation behaviour, no batch normalization nor dropout were used.

4.4. Detection of Dead Neurons

A specified subset of 1,000 training samples was used to record activations from all hidden layers following each training period. A neuron was considered **dead** if it produced no output for any of the observable inputs:

$$a_j^{(l)}(x_k) = 0 \forall k$$

The dead neuron percentage for each layer was computed as:

$$\text{Dead Percentage}_l = \frac{\text{Dead neurons in layer } l}{\text{Total neurons in layer } l} \times 100$$

This provides a direct quantitative measure of internal capacity loss.

4.5. Comparative Activation Functions

To verify that neuron death is caused by the zero-gradient behaviour of ReLU, two different activations were assessed using the same setup:

- **Leaky ReLU** with $\alpha = 0.01$(Maas, Hannun and Ng, 2013)
- **Exponential Linear Unit (ELU)** with $\alpha = 1.0$(Clevert, Unterthiner and Hochreiter, 2016)

Identical monitoring procedures were applied for fair comparison.

All simulation code, activation monitoring procedures, and figure generation scripts are available at: https://github.com/amila-samaranayake/dying-relu-analysis.git

5. Results & Discussion

Figure 1 presents the training loss and validation accuracy for the ReLU, Leaky ReLU, and ELU networks.

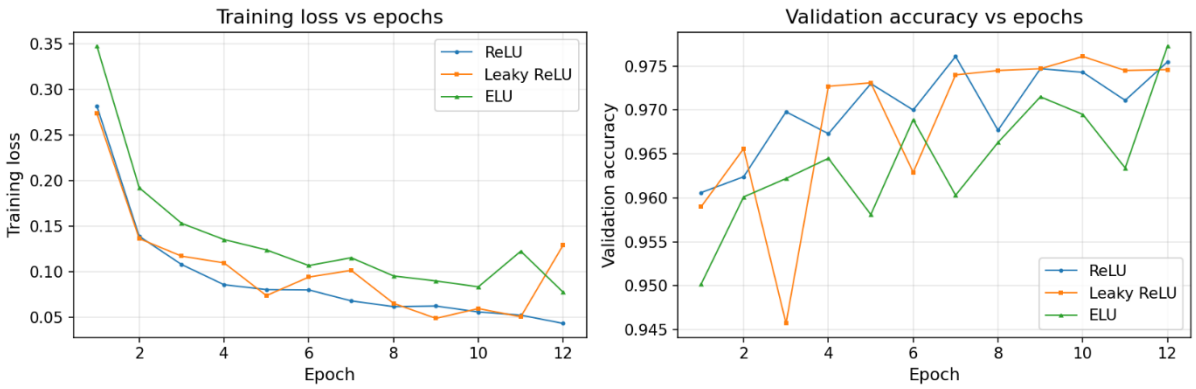

Figure 1: Training loss and validation accuracy across epochs for ReLU, Leaky ReLU, and ELU networks.

Across epochs, all three models show steady and seamless convergence. In this experiment, Leaky ReLU and ELU converge a little more slowly but stay stable, whereas ReLU obtains the lowest final training loss. All models' validation accuracy ranges from **96% to 98%,** demonstrating robust and similar generalization performance. These findings show that **intrinsic structural breakdown inside the network cannot be detected by typical performance indicators alone.**

The ReLU network exhibits considerable internal deterioration despite comparable exterior performance. The **percentages of dead neurons** by layer over training epochs are displayed in Figure 2.

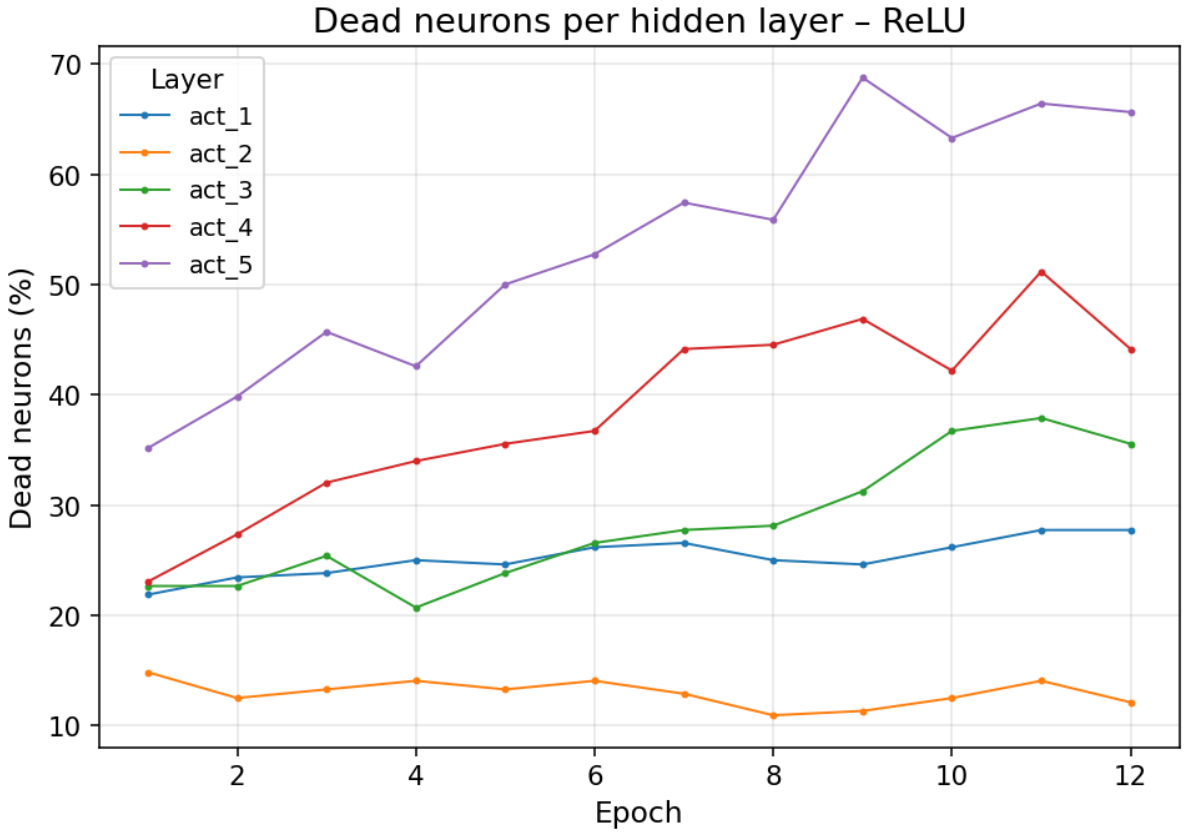

Figure 2 : Percentage of dead neurons per hidden layer in the ReLU network across training epochs.

There is a clear **depth-dependent pattern.** While deeper layers see ever higher neuron loss, the first buried layer stabilizes at around **25–28% dead neurons.** A significant loss of representational ability is indicated by the deepest hidden layer having more than **65% permanently deactivated neurons** by the end of the training cycle. The **permanent nature of neuron loss under ReLU** is confirmed by the fact that no recovery is seen after neurons become inactive.

A noteworthy finding is that a significant proportion of dead neurons already appears in the **first training epoch**. This demonstrates that dying ReLU may be caused by early weight updates, bias drift, and improper initialization in addition to being a long-term optimization problem. Repeated exposure to the zero-gradient condition increases this impact as training goes on, especially in deeper layers.

Figure 3 compares the **average dead neuron percentage across all hidden layers** for the three activation functions.

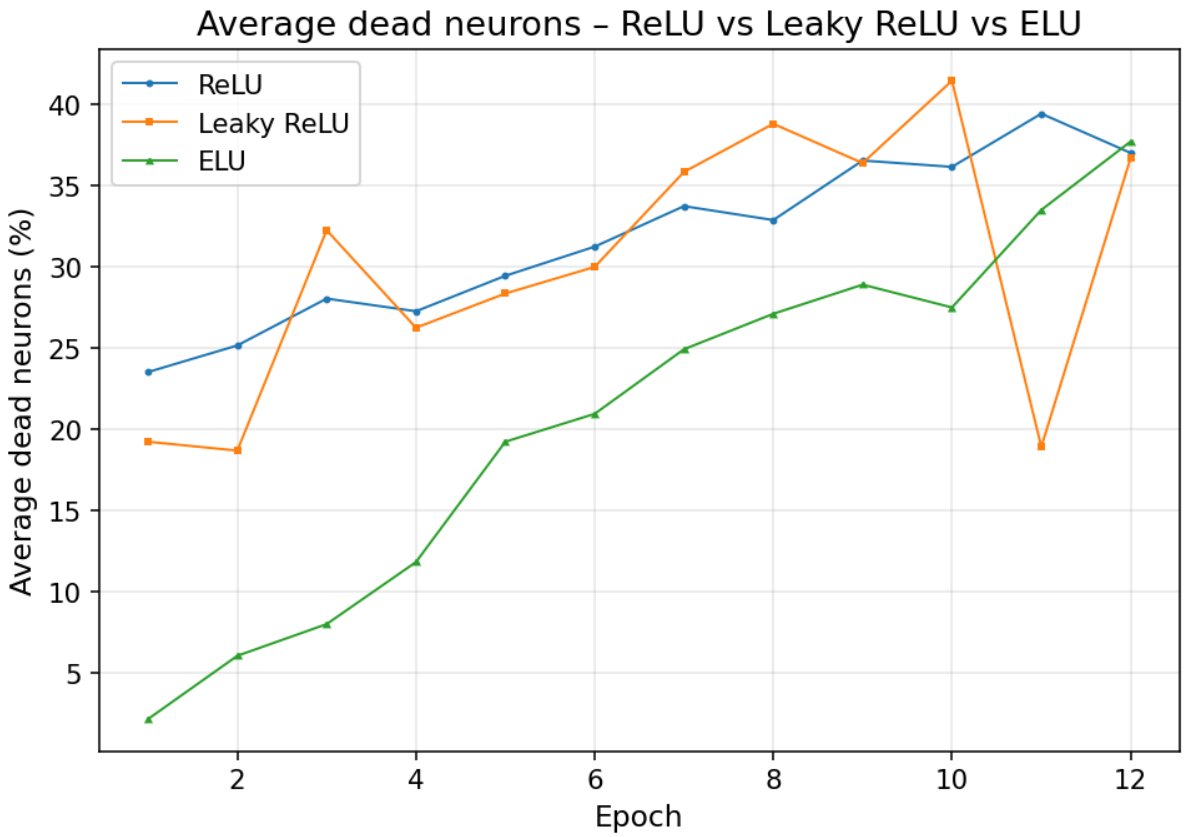

Figure 3: Average percentage of dead neurons across hidden layers for ReLU, Leaky ReLU, and ELU.

Neuron loss in the ReLU network grows steadily, reaching nearly **40% in the last epoch.** Leaky ReLU and ELU, on the other hand, keep **dead neuron numbers low and steady** during training. This demonstrates that dying ReLU is not an inevitable characteristic of deep neural networks, but rather is particularly brought on by **ReLU's rigorous zero-gradient behavior for negative inputs.**

Figure 4 provides visual confirmation through an activation heatmap of the first hidden layer of the ReLU network.

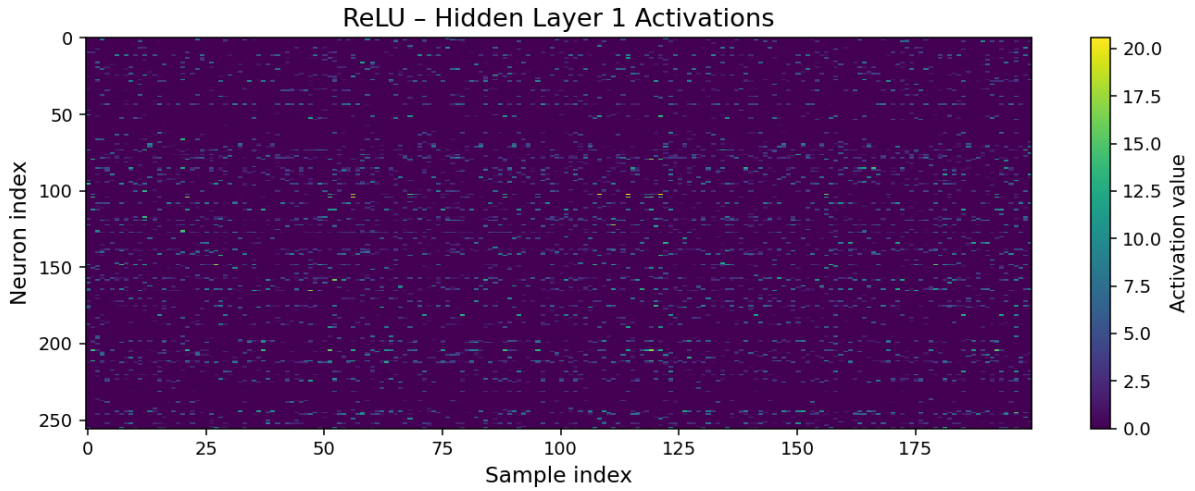

Figure 4: Activation heatmap of the first hidden layer in the ReLU network showing permanently inactive neurons.

Neurons that produce zero across all examined samples are shown by several **horizontal black bars.** Learning and the spread of information are no longer aided by these dead neurons. Even when classification performance is still excellent, the activation heatmap and quantified dead neuron measures together offer compelling empirical evidence of **internal representational collapse in ReLU networks.**

5.1. Further Mitigation Strategies for ReLU

The chance of dying ReLU can be decreased by several other tactics in addition to alternate activation functions. Initialisation of variance- preserving weights, as Early activation collapse is avoided because to the initialisation. Large updates that could force neurons into the negative regime permanently are avoided by carefully choosing the learning rate. While residual connections enhance gradient flow in deep designs, batch normalisation stabilizes activation distributions and lowers the chance of neurons going dormant. To keep deep networks robust and trainable, these methods function in tandem with activation design.

6. Summary

- The dying ReLU problem was analysed using mathematical derivation and controlled simulation.
- A deep MLP was trained on the MNIST dataset using ReLU, Leaky ReLU, and ELU activations.
- All three activation functions achieved similar classification accuracy.
- Only the ReLU network exhibited significant internal neuron collapse, particularly in deeper layers.
- Neuron death was observed as early as the first training epoch.
- Leaky ReLU and ELU maintained stable neuron activity due to their non-zero gradients in the negative input region.

- The results confirm that the dying ReLU problem is caused by ReLU's zero-gradient behaviour, not by deep learning itself.
- Silent internal neuron collapse presents potential reliability risks in real-world AI systems.

7. References

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv*. https://arxiv.org/abs/1511.07289

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. https://arxiv.org/pdf/1502.01852

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*. https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*. https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf

TensorFlow Developers. (2024). *Keras activation functions documentation*. https://www.tensorflow.org/api_docs/python/tf/keras/activations

TensorFlow Developers. (2024). *tf.keras.layers.ELU documentation*. https://www.tensorflow.org/api_docs/python/tf/keras/layers/ELU

TensorFlow Developers. (2024). *tf.keras.layers.LeakyReLU documentation*. https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU