How the Sigmoid Activation Function Causes Vanishing Gradients in Deep Neural Networks

Archit Roy

Abstract: Backpropagation is the fundamental algorithm that enables neural networks to learn. Neural networks rely on it to update their parameters. However, the effectiveness of this algorithm depends on maintaining a sufficiently large gradient across layers. Some activation functions significantly reduce the gradient flow. The sigmoid in particular causes this weak gradient flow. This review looks closely at how the derivative of the sigmoid function contributes to the vanishing gradient problem, analyses the mathematical form, shows that the derivative flattens out to zero over a large range of input values and makes gradient propagation inefficient in deep neural networks. A small experiment is presented to show how networks using sigmoid train slower than networks using the ReLU. This review highlights why the sigmoid has largely been replaced in modern neural networks.

1.Introduction

Deep neural networks have often experienced problems in training due to vanishing gradients. Training deep neural networks effectively requires the gradients to be propagated back from output layers to the starting layers. If these gradients become extremely small, the process of learning slows down drastically or halts altogether. This is known as the vanishing gradient problem. This made training of deep neural networks a difficult task before the adoption of more robust activation functions like the hyperbolic tangent or ReLU.

One of the primary reasons for the vanishing gradients is the choice of activation function. Initially the sigmoid function was widely used due to its smoothness and simplicity. However, studies have indicated that its mathematical properties lead to shrinking gradients during backpropagation in deep networks. This review analyses how the derivative of the sigmoid function leads to the problem and why other activation functions are more suitable for deep networks.

2.Methodology

This review synthesizes findings from peer-reviewed journals, government publications, and institutional reports published till 2025. Articles were selected based on relevance to vanishing

gradients, activation functions in neural networks and backpropagation using sources like geeks for geeks, lunatech, arxiv.

3.Background

3.1 The Sigmoid function

The sigmoid is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

It is a mathematical function with a distinctive S-shaped curve. If x is a large negative value, the function approaches 0. If x is a large positive value, the function approaches 1. Due to this binary nature, it allows networks to learn non linear relationships of data and the outputs of the network can be interpreted as probabilities.

3.2 The derivative of the Sigmoid function

The derivative of the sigmoid is defined as:

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

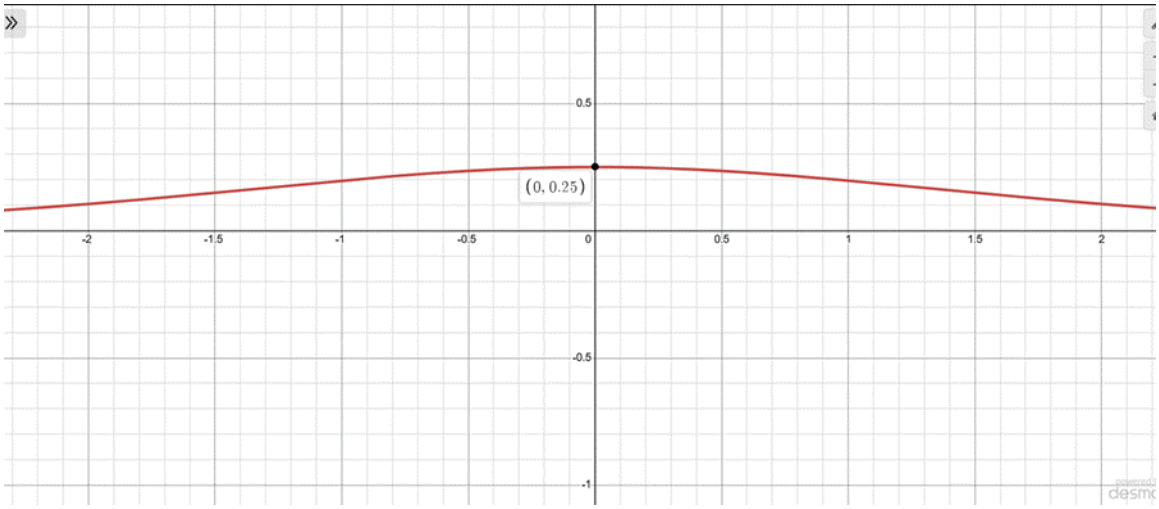

Figure 1.1 The graphical representation of the derivative of the sigmoid (Desmos)

As shown in Figure 1.1 the derivative of the sigmoid function has a maximum value at (0,0.25). The sections ahead explain this value significantly weakens the gradient flow.

3.3 Backpropagation and gradient flow

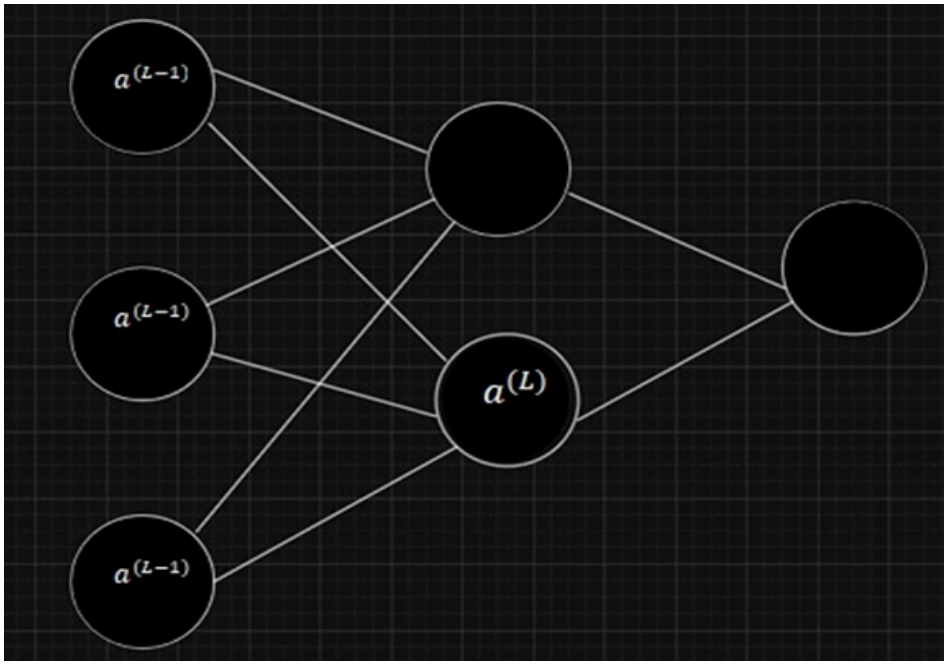

Figure 1.2 shows a small neural network

The calculation for a backpropagation in terms of the cost function is done as follows:

$$\frac{\partial C_0}{\partial W^{(L)}} = \frac{\partial Z^{(L)}}{\partial W^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial Z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}}$$

Where C_0 is the cost function defined by C_0 = ($a^{(L)}$-y)^2 where y is the expected output of the network and $W^{(L)}$ is the weight for each node. $a^{(L)}$ is the activation of each neuron as shown by Figure 1.1 and $Z^{(L)}$ is defined as: $Z^{(L)}$ = σ($W^{(L)}a^{(L-1)}$ + b) where b is the bias. The equation after calculating each derivative can be rewritten as:

$$\frac{\partial C_0}{\partial W^{(L)}} = a^{(L-1)} \cdot \sigma'\left(Z^{(L)}\right) \cdot 2(a^{(L)}\text{-y})$$

As shown above, the derivative of the sigmoid appears in the equation we need to calculate the rate of change of the cost function. This means that it affects the process of backpropagation.

Keep in mind that this calculation only applies for one node. The calculation for all nodes is given as follows:

$$\frac{\partial C_0}{\partial W^{(L)}} = \frac{1}{n}\sum_{k=0}^{n-1}\frac{\partial C_k}{\partial W^{(L)}}$$

This means that the derivative of the sigmoid is used multiple times in this calculation.

4.1 Values of $\sigma'(x)$

As discussed earlier the derivative of the sigmoid is:

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

This expression has a maximum value of 0.25 at $x = 0$ and the function $\sigma'(x) \to 0$ as x moves to 1 or -1. This means that the gradient is usually 0 or very close to 0 for relatively large positive and negative values of $Z^{(L)}$. This makes the gradient flow highly inefficient as there is a relatively low rate of change in the cost function, slowing down the training process.

4.2 Mathematical comparison of the tanh and Sigmoid functions

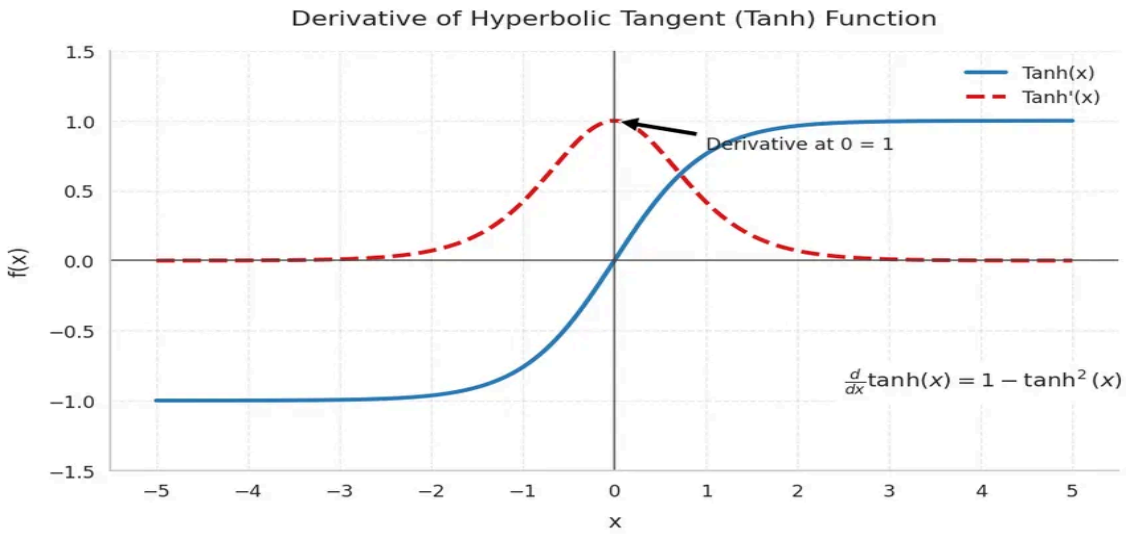

Figure 1.3 The graph of derivative of tanh (source: GeeksforGeeks)

As illustrated by Figure 1.3 the derivative of tanh has a max value of 1 at $x = 0$ and has a value of approximately 0.42 at 1 and -1. This means that tanh will maintain sufficiently high values for all values of $Z^{(L)}$. This is beneficial as the rate of change of the cost function is relatively high. This maintains a strong gradient flow throughout the backpropagation process resulting in training times that are significantly better than the sigmoid.

4.3 Implication for gradient flow

During backpropagation the derivative of the sigmoid is multiplied in various layers. This means that this gradient shrinks rapidly across layers. Let's demonstrate this shrink through a small example. We assume that the derivatives of both the sigmoid and the tanh activation functions are at their peak values. The gradients after passing through 10 layers would be:

Sigmoid: $(0.25)^{10} \approx 9.54 \times 10^{-7}$

tanh: $(1)^{10} = 1$

This small comparison shows that by passing through just 10 layers, the gradient for the sigmoid function shrinks to nearly zero. This makes deep neural networks inefficient and inhibits their ability to learn. The sigmoid saturates and is not useful for real world inputs. It is practically zero at all values. On the contrary, the tanh function maintains a strong gradient flow across these 10 layers for the same value of $Z^{(L)}$ making it the better alternative to the sigmoid in deep neural networks.

5. Comparison between Sigmoid and ReLU

The following comparison uses the Sequential model from keras and works on the MNIST dataset for getting the required results.

Figure 1.4. Code snippet using the Sigmoid activation (Google Colab)

Figure 1.5.Code snippet using the ReLU activation (Google Colab)

The results are as follows:

```
model.fit(Xtrain,Ytrain,epochs=10,batch_size=128)

Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.1060 - loss: 4.5765
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.1053 - loss: 2.3062
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.1009 - loss: 2.3078
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.1048 - loss: 2.3073
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.1047 - loss: 2.3096
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.1047 - loss: 2.3088
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.1052 - loss: 2.3079
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.1019 - loss: 2.3091
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.1044 - loss: 2.3085
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.1031 - loss: 2.3103
<keras.src.callbacks.history.History at 0x7d20078259d0>
```

Figure 1.6 Results of the Sigmoid Activation function (Google Colab)

```
model.fit(Xtrain,Ytrain,epochs=10,batch_size=128)

Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.2977 - loss: 553.4570
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7491 - loss: 12.7156
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8211 - loss: 8.1894
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8461 - loss: 6.4879
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8619 - loss: 5.4374
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8692 - loss: 4.7533
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8765 - loss: 4.2339
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8808 - loss: 3.8367
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8820 - loss: 3.5284
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8842 - loss: 3.3873
<keras.src.callbacks.history.History at 0x7a6b39d366c0>
```

Figure 1.7 Results for the ReLU Activation function (Google Colab)

As shown by Figure 1.6 the calculated loss for the model using the sigmoid function as activation becomes stagnant after the second Epoch. The loss remains constant to two decimal

places at approximately 2.31 for the entirety of the test after the second Epoch. This demonstrates that the gradient is shrinking through the layers. As for the ReLU, the Figure 1.7 clearly shows that the calculated loss is varying throughout the test. This shows that the ReLU is not undergoing the gradient shrinking problem that the sigmoid function faces. The loss keeps rapidly changing throughout the entirety of the test showing why it has largely replaced the sigmoid in deep network architectures.

6. Conclusion

This paper examined how the derivative of the sigmoid function causes the vanishing gradient problem in deep neural networks. The gradient having a maximum value of 0.25 and having a sharp decline towards 0 for higher input values makes it incompetent for use and creates a chain of shrinking gradients across layers during backpropagation. As a result, the networks that use the sigmoid activation function struggle to learn effectively. This mathematical analysis provides a clear explanation on why more advanced networks deter from using the sigmoid function. Instead, activation functions that preserve gradient values like ReLU have largely replaced the sigmoid for deep neural networks.

References

[1] Glorot, Xavier, and Yoshua Bengio. “Understanding the Difficulty of Training Deep Feedforward Neural Networks.” *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, https://proceedings.mlr.press/v9/glorot10a.html.

[2] Van, Leni, and Johannes Lederer. “Regularization and Reparameterization Avoid Vanishing Gradients in Sigmoid-Type Networks.” *arXiv*, 2021, https://arxiv.org/abs/2106.02260.

[3] Zeng, Jinshan, et al. “On ADMM in Deep Learning: Convergence and Saturation-Avoidance.” *arXiv*, 2019, https://arxiv.org/abs/1902.02060.

[4] Hu, Z., J. Zhang, and Y. Ge. “Handling Vanishing Gradient Problem Using Artificial Derivative.” *IEEE Access*, 2021, https://doi.org/10.1109/ACCESS.2021.3055358.

[5] Gullipalli, T., K. Murali, and S. Peri. “An Improved Taylor Hyperbolic Tangent and Sigmoid Activations for Avoiding Vanishing Gradients in Recurrent Neural Nets.” *International Arab Journal of Information Technology*, 2023, https://www.iajit.org/paper/5282.

[6] Alzubaidi, S., et al. “Vanishing Gradient Problem.” *Journal of Big Data*, 2021, https://doi.org/10.1186/s40537-021-00444-8.