University of Hertfordshire UH

Name :- Tiasa Malitya

Student ID :- 24005709

Course :- M.Sc. Data Science

Module :- **7PAM2021-0105-2024 – Machine Learning and Neural Networks**

Title :- **Individual Assignment**

Date :- 27/03/2025

Github**:** **https://github.com/Tiasa24/ML_tutorial25_5709**

The necessity of choosing the Right Activation Function in neural networks:

Sigmoid vs. ReLU

Source: Internet

Overview

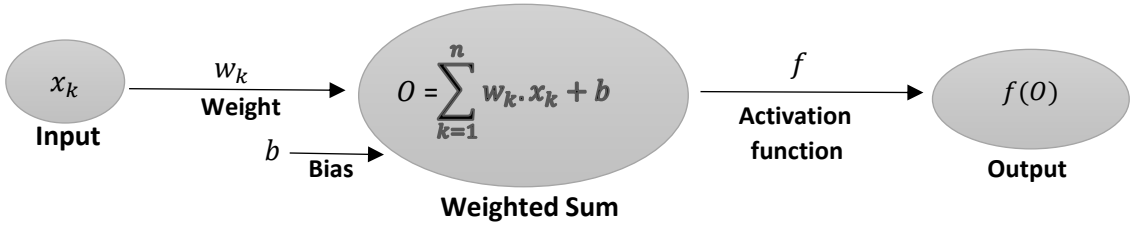
The main focus of this tutorial is to get a clear concept of

- ✓ **Reasons Neural Networks Need Activation Functions**
- ✓ **Sigmoid activation**
- ✓ **ReLU activation**
- ✓ **Practical comparison between Sigmoid and ReLU**
- ✓ **Train CNNs with Sigmoid and ReLU on CIFAR-10 to analyse their performance**.
- ✓ **When to use which?**

From Linear Models to Deep Learning

The concept of *activation functions* lies at the core of neural networks. A neural network is designed to mimic the human brain, with layers of interconnected neurons where activation function is a switch which helps to make the decision what information should be shared next and what should be ignored. However, if we don't introduce activation functions, this process would be nothing more than a basic linear regression model that actually limits the capability of model to learn advanced structure.

Mathematically, activation functions take the weighted sum of a neuron's inputs and apply a non-linear transformation to determinate the neuron's output.



As non-linearity is crucial for models like feedforward neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs) to learn and solve complex tasks in areas like image recognition, natural language processing and time-series forecasting, Activation functions are the one making it possible for these models.

1. Sigmoid vs ReLU

Early neural networks used step functions as activation, similar to the biological neurons where they either fully active or remains inactive depending on whether the input exceed a certain threshold. Over the time, as we needed to create more advanced machine learning model, these simple functions were replaced by more sophisticated ones like sigmoid, ReLU, SoftMax, tanh and many others each offering unique benefits.

In this tutorial we will be exploring mainly ***Sigmoid*** and ***ReLU*** function, how differently they work and the specific scenarios where each is most relevant. By understanding their role, we can understand the power of these in deep learning and how they can be used to solve a wide variety of problems.

1.1 Understanding the Sigmoid Activation

The Sigmoid function is one of the most fundamental activation functions used in neural networks. It introduces non-linearity by transforming input values into a smooth, S-shaped curve and mapping them to a range between 0 and 1.

Q. What is the mathematical formula?

The sigmoid function is defined as

$$\sigma(A) = \frac{1}{1+e^{-A}} \qquad \text{...(i)}$$

where,

A is the input (weighted sum of neuron inputs in a neural network).

e is Euler's number (=2.718), a mathematical constant.

Q. How Does the Function Look Graphically?

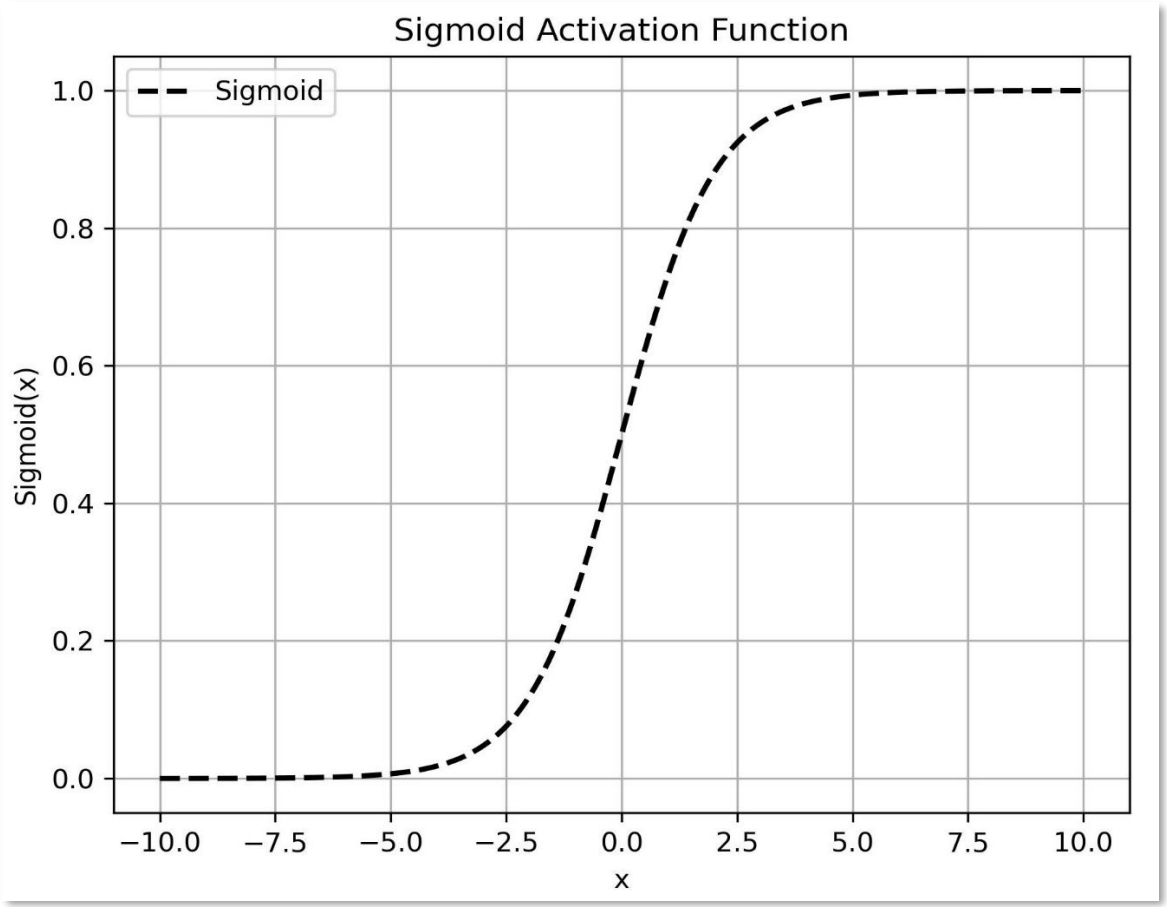
Below is the python code which plots the Sigmoid function.

```
# Define the Sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Generate values for x
x = np.linspace(-10, 10, 400)

# Plot Sigmoid function
plt.figure(figsize=(6,5))
plt.plot(x, sigmoid(x), label='Sigmoid', linestyle="--", color ="black", linewidth=2)
plt.title('Sigmoid Activation Function')
plt.xlabel('x')
plt.ylabel('Sigmoid(x)')
plt.grid(True)
plt.legend()
```

here we get the graph,



Plotting the function gives an S-shaped curve:
for positive values of **x,** e^{-x} approaches 0, making $\sigma(x)$ tends to 1,
for negative values of x, e^{-x} approaches 1, making $\sigma(x)$ tends to 0.

Q. Why Sigmoid Is commonly used in Neural Networks?

A. Beyond Linearity

The Sigmoid function helps amplify important information while filtering irrelevant inputs. Although neural networks primarily perform only matrix multiplications and additions, the Sigmoid transforms them to non-linear, allowing the network to learn complex decision boundaries.

Example

Try to classify handwritten digits **'0'** and **'1'**. A linear approach may struggle if some '0's are slightly deformed or some '1's are slanted. By applying the Sigmoid activation function in the hidden layers, the network uses neurons to gradually bend and adjust the decision boundary making it better at distinguishing between the two digits.

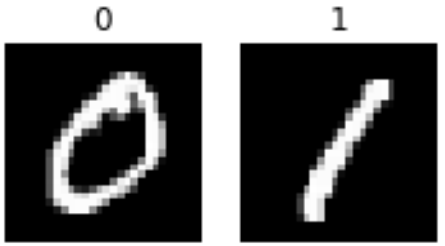


Figure: handwritten digit 0 and 1 (image source)

B. Normalized Output (0 to 1) ensuring meaningful Probability Scores

In binary classification tasks output of sigmoid function is always between 0 and 1, which can be interpreted as the probability of a sample belonging to a particular class. This is crucial for clear decision-making. It's ability to map input values into a probability range ensures that all outputs remain within a reasonable scale preventing extremum. This makes the Sigmoid function a great choice when you need to estimate the likelihood or confidence score.

Example:

In a spam classifier for emails, the Sigmoid function takes the features of an email and outputs a value between 0 and 1. This probabilistic output helps to inform and interpret decisions whether an email should be classified as spam or not.

C. Smooth and Differentiable for Optimization

Neural networks learn to reduce prediction errors by gradient descent that helps optimize the weights by calculating gradients and adjusting the weights in the direction that reduces the loss. For this process to work, the network needs to know how much each weight should be updated, which is done through backpropagation. To make sure this learning process works smoothly, the activation function used in the network must be differentiable as the gradients are calculated using the derivative of the activation function.

In case of sigmoid function, it's simple to compute the derivative which is given by:

$$\sigma'(A) = \sigma(A)[1 - \sigma(A)] \qquad \text{..... (ii)}$$

The fact that the derivative is expressed in terms of itself simplifies the calculations and allows the gradients to flow easily through the network during backpropagation helping the network to gradually minimize prediction errors and improve performance. However, while sigmoid is useful, it also has some limitations that we will explore next.

Q. Does the Sigmoid Function Have Any Drawbacks?

A. Vanishing Gradient Problem

We have seen the derivative of the Sigmoid function is depend on its output. When the input is very large or very small, the Sigmoid function's output approaches either 1 or 0 respectively. As a result, the derivative becomes very small.

From the equation (ii), we get

$If\ \sigma(A) \approx 1, \sigma'(A) \approx 0$ and similarly, $If\ \sigma(A) \approx 0, \sigma'(A) \approx 0$

This small gradient leads to the vanishing gradient problem, where the gradients during backpropagation become so tiny that the learning process slows down significantly. In deep neural networks, this can cause difficulty in training the model as the weight updates is minimal.

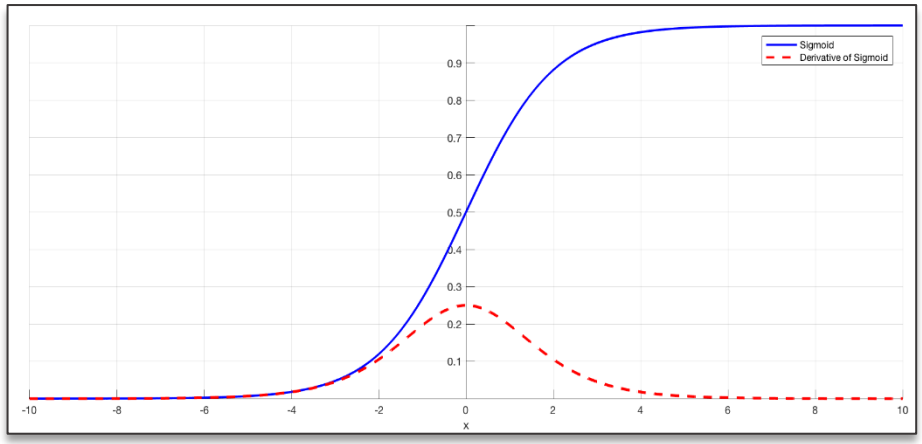


Figure: Vanishing gradient

B. Not Zero-Centred

The Sigmoid function always outputs values between 0 and 1, which means its gradient (or derivative) is always positive. This results in weight updates during training being either consistently positive or negative, slowing down the learning process. The lack of negative gradients limits the network's ability to adjust the weights in all directions effectively and can lead to slower convergence during training.

1.2 Deep Dive into ReLU

To get rid of the major drawback of vanishing gradient problem, researchers introduced the *ReLU (Rectified Linear Unit)* making it a more efficient choice for modern neural networks. Let's explore how ReLU works and why it has become the default activation function in deep learning.

ReLU is a widely used activation function in deep learning due to its effectiveness. Unlike sigmoid, which squeezes values into a narrow range, ReLU allows the network to retain more information and speeds up training.

Q. What is the mathematical formula?

ReLU is defined as,

$f(A) = \max(0, A)$ ..... (iii)

This means that for any input A, if A is positive, the function outputs A. Otherwise, it outputs 0.

Q. How Does the ReLU Look Graphically?

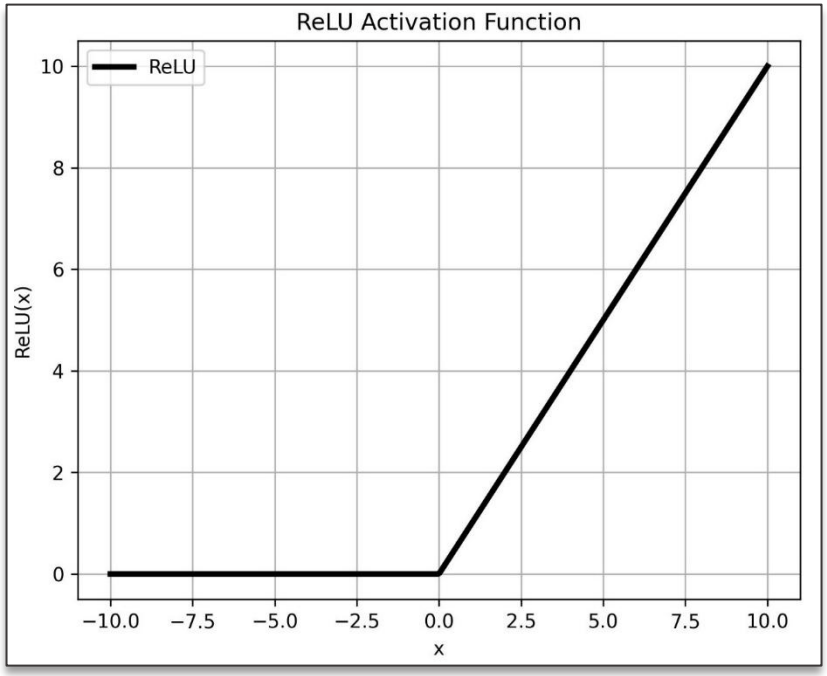
To visualize ReLU, let's plot

```
# Define the ReLU function
def relu(x):
    return np.maximum(0, x)

# Generate values for x
x = np.linspace(-10, 10, 400)

# Plot ReLU function
plt.figure(figsize=(6, 5))
plt.plot(x, relu(x), label='ReLU', linestyle = "solid", linewidth=3, color = "black")
plt.title('ReLU Activation Function')
plt.xlabel('x')
plt.ylabel('ReLU(x)')
plt.grid(True)
plt.legend()
```

the graph



Q. What ReLU is powerful?

A. Solve Vanishing Gradient

Unlike the sigmoid function, ReLU does not saturate for positive values helps gradients remain optimised during backpropagation, leading to faster and better convergence making ReLU particularly well-suited for deep networks where multiple layers need to be optimized efficiently.

B. Computational Efficiency

The ReLU function is simple to compute it just replaces negative values with zero. This means fewer expensive operations.

C. Sparse Activation (Efficiency in Large Networks)

ReLU introduces sparsity in the neural network by outputting zero for all negative input values, which means some neurons remain inactive. This leads to fewer active neurons during each computation, reducing unnecessary calculations and improving computational efficiency, making the learning process faster. By activating only, the neurons with positive inputs, ReLU helps to prevent overfitting by simplifying the network's complexity.

2. Sigmoid vs. ReLU: A Practical Comparison

Now that we have a solid understanding of both the **Sigmoid** and **ReLU** activation functions, let's compare their performance by training a CNN model on the CIFAR-10 dataset containing 60,000 colour images (size 32×32 pixels) across 10 different classes like Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck.

- Let's start with importing necessary dependencies

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
```

- Load the dataset and preprocessing

```
# Load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Normalize pixel values (0 to 1 range)
X_train, X_test = X_train / 255.0, X_test / 255.0
```

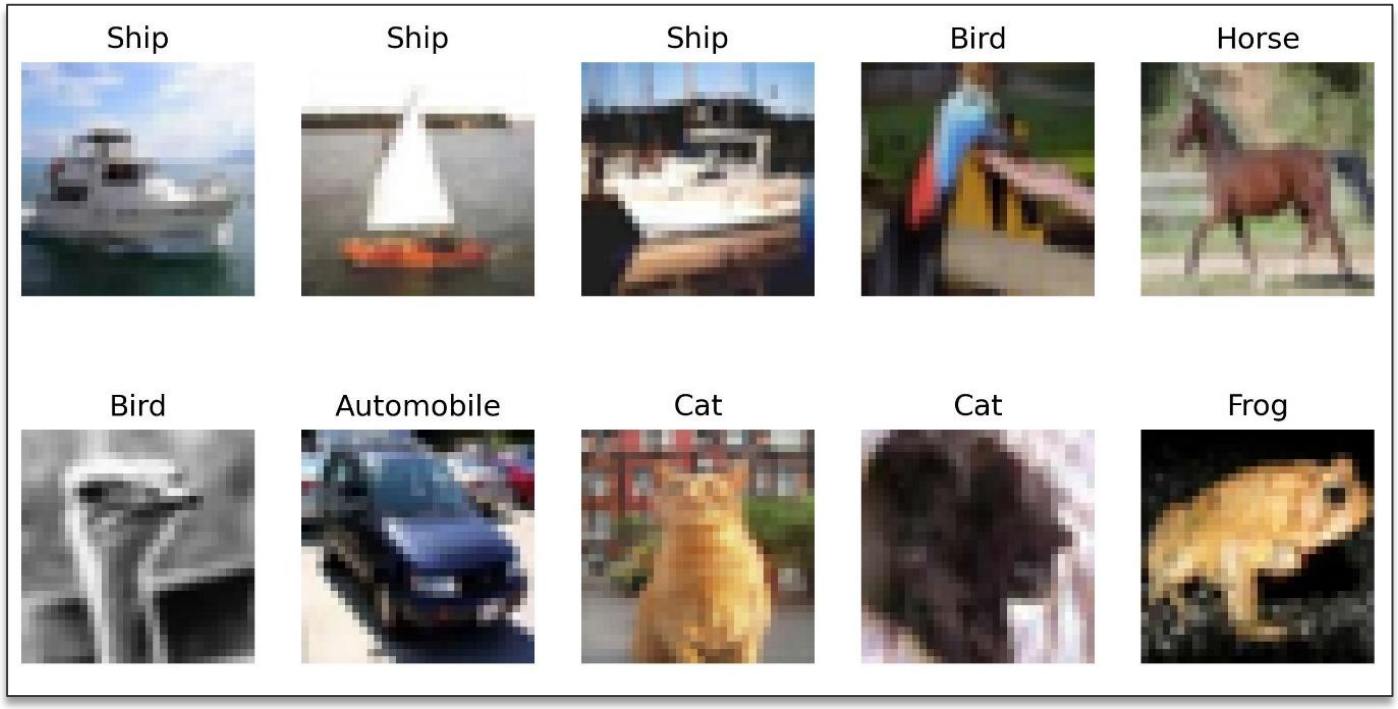
❖ Turn the labels to categorical format

```
# Convert labels to categorical format
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Why One-Hot Encoding

Instead of using a single number for each class (0-9), we convert it into a vector to treat each class independently without implying any order or relationship between them.

❖ let's explore some images from the CIFAR-10 dataset to understand the variety of objects it contains.



What is Validation Accuracy and Loss?

let's quickly refresh the concepts

Validation Accuracy measures how well our model generalizes to unseen data. Higher is better!

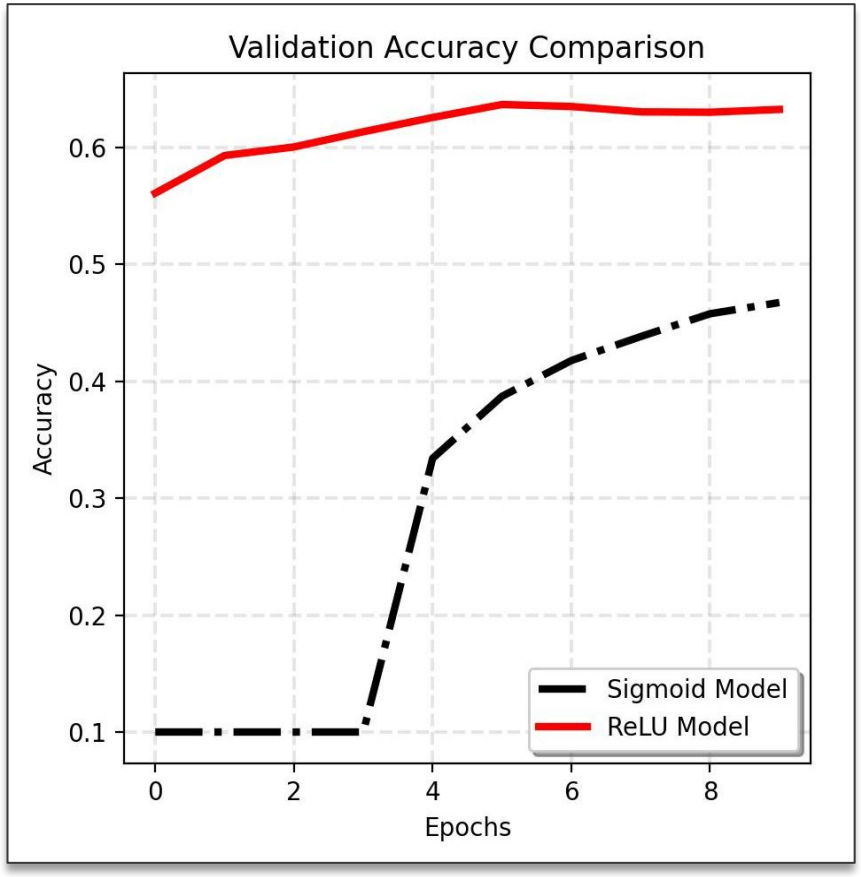
Validation Loss indicates how well our model's predictions match the true values. Lower is better!

Let’s compare the performance of the Sigmoid and ReLU activation functions using validation accuracy and loss over 10 epochs. This will give us insights into how each activation function behaves in practice and how the models learned over time whether they overfitted or underperformed.

Point 1: Validation Accuracy Comparison

Sigmoid starts off with 10% accuracy and improves gradually, reaching 46.72% by the end. The slow improvement in validation accuracy suggests the model is struggling with underfitting, meaning it isn’t capturing the complexities of the CIFAR-10 dataset as well as it could.

ReLU, on the other hand, starts much better at 56.07%, and its accuracy steadily increases to 63.23%. helps the model learn faster, likely avoiding the underfitting issue seen with Sigmoid. Its smoother and faster improvement indicates better generalization to unseen data.



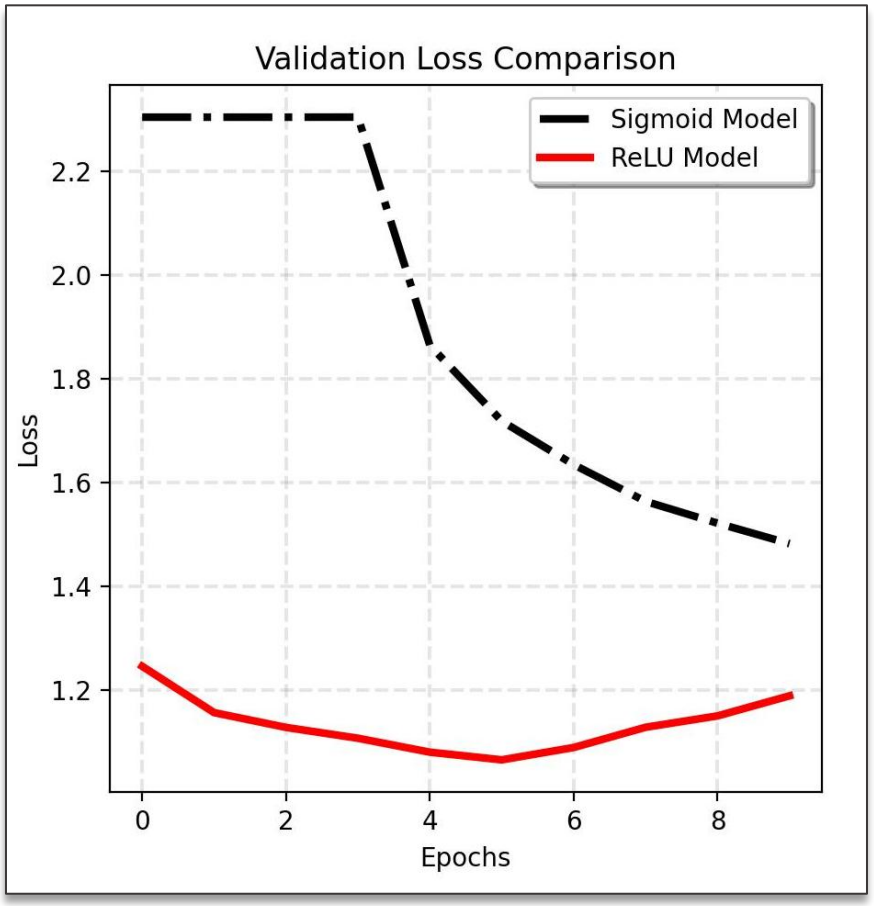
Q. What does this tell us?

ReLU seems to help the model learn faster and achieve better performance. The key here is that ReLU doesn't suffer from the vanishing gradient problem as much as Sigmoid. It’s easier for the network to update its weights and learn from the data.

Point 2: Validation Loss Comparison

Sigmoid begins with a validation loss of 2.3047 and reduces to 1.4827 by the end. While this is a steady reduction, the loss does not decrease as quickly as ReLU's, which suggests that Sigmoid might be encountering some difficulty in optimizing—potentially a result of underfitting.

ReLU starts with a lower loss of 1.2465 and gradually drops to 1.1895. ReLU shows a faster and more consistent reduction in loss, indicating that it is more capable of learning from the data and generalizing to new examples.

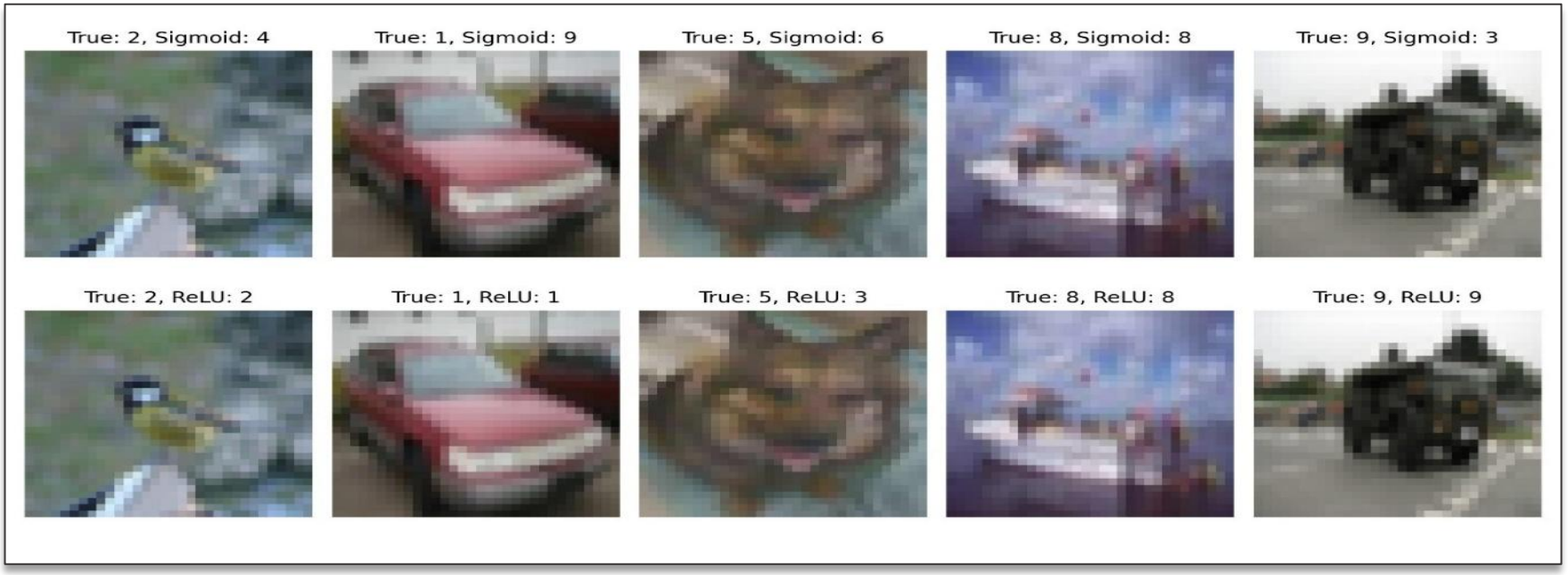


Q. What can we learn here?

ReLU not only gives us a higher accuracy but also a lower validation loss, meaning it's making better predictions earlier in training.

3. Visual Comparison

Predictions of both models on these 10 test images



As you can see, the ReLU model generally produces more accurate results, correctly identifying more images compared to the Sigmoid model. While both models make some mistakes, the difference in their performance is evident. By seeing these examples, we get a clearer picture of why ReLU tends to outperform Sigmoid, especially for complex task.

4. When to Use Sigmoid or ReLU?

Choosing the right activation function is essential for optimizing neural network performance, and the decision depends on factors like task type, network depth, and computational efficiency. **Use Sigmoid** when working on binary classification tasks where outputs need to represent probabilities between 0 and 1, in shallow networks where vanishing gradients have minimal impact, or in the output layer for probability estimation. **Use ReLU** for deeper networks to mitigate vanishing gradients, in models trained on large datasets where computational efficiency is crucial, and in deep learning architectures where ReLU is the standard for hidden layers. **Best practice** suggests using ReLU in hidden layers due to its efficiency, while Sigmoid is mainly reserved for the output layer in binary classification. For multi-class classification, SoftMax is commonly used in the output layer to ensure probability values sum to 1.

5. Learning Outcome

Through this comparison, we've seen how activation functions can significantly impact a neural network's learning ability. ReLU demonstrated faster convergence, higher validation accuracy, and lower validation loss, making it a better choice for deep networks. Sigmoid, while useful in specific cases, struggled with slow learning and vanishing gradients, limiting its effectiveness in deep architectures.

The key takeaway? Choosing the right activation function isn't just a technical detail, it directly affects how well and how efficiently a model learns. If you're working with deep networks, ReLU is usually the smarter pick. However, always consider the nature of your problem before deciding. The key is understanding your model's needs and choosing the activation function that best supports learning.

References

Activation Functions in Neural Networks – A Blog by Enjoy Algorithms.
Available at: https://www.enjoyalgorithms.com/blog/activation-functions-in-neural-networks [Accessed 24 March 2025].

Activation Functions in Neural Networks: A Comprehensive Guide. Active Tech Systems.
Available at: https://activetechsystems.com/blog/machine-learning/activation-functions-in-neural-networks-a-comprehensive-guide/ [Accessed 24 March 2025].

Activation Functions in Neural Networks. Warga Net Tech.
Available at: https://warganet.tech/2024/10/activation-functions-in-neural-networks/ [Accessed 24 March 2025].

Kamalov, F., Cherukuri, A. K., Nazir, A., Safaraliev, M., & Zgheib, R. (2021) 'Comparative analysis of activation functions in neural networks'.
28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS). DOI: 10.1109/ICECS53924.2021.9665646.

Nwankpa, C. E., Ijomah, W. L., Gachagan, A., & Marshall, S. (2018) 'Activation functions: Comparison of trends in practice and research for deep learning'.
University of Strathclyde. Available at: https://arxiv.org/abs/1811.03378 [Accessed 24 March 2025].

Sharma, S., Sharma, S., & Athaiya, A. (2021) 'Activation functions in neural networks'.
Global Institute of Technology, Jaipur.

Szandała, T. (2021) 'Review and comparison of commonly used activation functions for deep neural networks'.
Wroclaw University of Science and Technology.