Activation Functions & Non-Linearity: A Quick Guide

Understanding the Need for Activation Functions – A Hiring Example

Imagine you're building a system to help HR departments filter job candidates. You have data on potential hires, including:

- **CGPA:** Their academic performance.
- **Internship Length (Months):** How long they've interned for.
- **Projects:** The number of projects they've completed.
- **Communication Skills (Rating 0–10):** A score based on their communication abilities.

Based on these factors, you want to categorize each candidate into one of three labels:

- **Strong Fit**
- **Maybe**
- **Not a Fit**

Now, you could use a simple linear equation to combine these factors. You might assign weights to each factor and sum them up. For example:

$$Z = (w_1 \cdot \text{CGPA}) + (w_2 \cdot \text{Internship Length}) + (w_3 \cdot \text{Projects}) + (w_4 \cdot \text{Communication Skills}) + b$$

Let's say you randomly assign the following values: $w_1 = 0.3$, $w_2 = 0.2$, $w_3 = 0.5$, $w_4 = 0.1$, and $b = -3$. For a candidate (Candidate A) with CGPA of 8.9, Internship Length of 6, Projects of 5, and Communication Skills of 9:

$$Z = (0.3 \cdot 8.9) + (0.2 \cdot 6) + (0.5 \cdot 5) + (0.1 \cdot 9) - 3 = 4.27$$

The problem is, the Z value you get might be on a completely different scale from your input features. CGPA is between 0 and 10, communication skills between 0 and 10, projects might be a small number, and internship length might be a small number; but the Z value has now become 4.27. They don't have an upper bound or lower bound.

If this Z value is passed to another layer in your neural network, it will again become difficult to interpret, especially if your parameters have vastly different ranges. This is where activation functions come in! They help compress the values and put them on a common scale so they're easier to work with and help the model learn better.

What are Activation Functions?

Activation functions are mathematical functions that introduce non-linearity to the output of a neuron. In essence, they decide whether a neuron should "fire" or not based on its input. They take the result of your linear equation (the Z value we calculated earlier) and transform it into a more manageable output.

Without activation functions, your neural network would simply be a series of linear transformations. This limits its ability to learn complex patterns. Activation functions help introduce complexity into the model.

The key here is to map this value (Z) to a range. Let's explore the most common activation functions:

Sigmoid: Squashing Values Between 0 and 1

The sigmoid function is defined as:

$$f(Z) = \frac{1}{1 + e^{-Z}}$$

where e is Euler's number (approximately 2.71828). The sigmoid function takes any real value as input and outputs a value between 0 and 1. This is very useful for tasks where you need to predict a probability, like classifying an email as spam or not spam.

Let's consider Candidate A again and calculate its sigmoid:

$$f(4.27) = \frac{1}{1 + e^{-4.27}} \approx 0.986$$

Advantages:

- Outputs a value between 0 and 1, which can be interpreted as a probability.
- Easy to understand and implement.

Disadvantages:

- **Vanishing Gradient:** For very large or very small inputs, the gradient of the sigmoid function becomes very small, which can slow down learning.

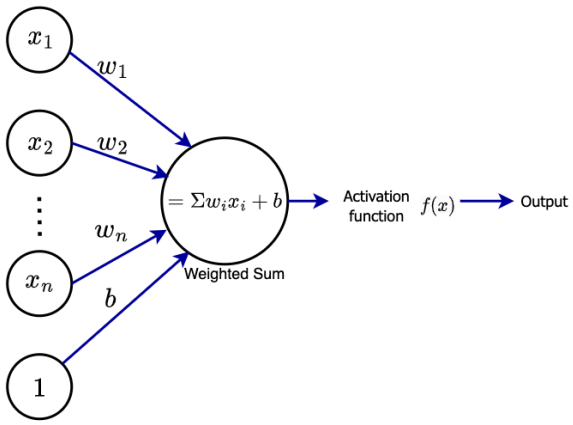

Figure 1: Activation function

ReLU (Rectified Linear Unit): The Simple Yet Effective Choice

The ReLU function is defined as:

$$f(Z) = \max(0, Z)$$

This means that if the input is positive, the output is the input itself. If the input is negative, the output is zero.

So, if the input Z is -10, the output is 0; if the input is 5, the output is 5. Back to Candidate A, the ReLU score is:

$$f(4.27) = \max(0, 4.27) = 4.27$$

Advantages:

- Simple and computationally efficient.
- Less prone to vanishing gradients than sigmoid in many cases.

Disadvantages:

- **Dying ReLU:** Neurons can "die" if their inputs are consistently negative, meaning they always output zero and stop learning.

Tanh (Hyperbolic Tangent): Centering the Data

The tanh function is defined as:

$$f(Z) = \frac{e^{Z} - e^{-Z}}{e^{Z} + e^{-Z}}$$

This function outputs a value between -1 and 1. It's similar to sigmoid but centered around zero, which can sometimes lead to faster learning.

Back to Candidate A:

$$f(4.27) = \frac{e^{4.27} - e^{-4.27}}{e^{4.27} + e^{-4.27}} \approx 0.999$$

Advantages:

- Output is centered around zero, which can improve learning.
- Strong gradients.

Disadvantages:

- Still susceptible to vanishing gradients, although less so than sigmoid.

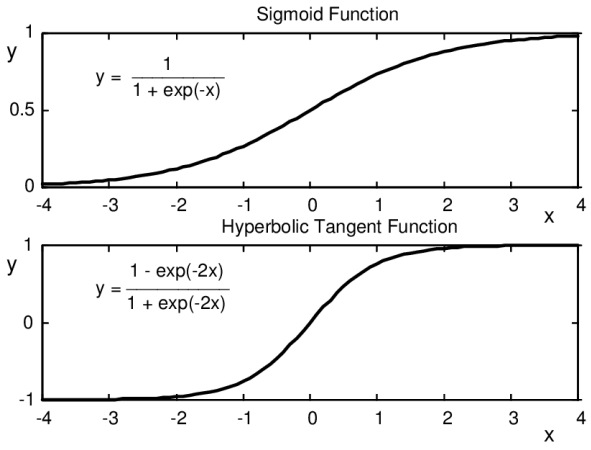

Figure 2: Sigmoid vs Tanh

Vanishing Gradients: A Learning Hurdle

Vanishing gradients occur when the gradients during backpropagation become extremely small. This happens because neural networks learn through a process of iteratively adjusting their weights to minimize the error in their predictions. The gradient tells the network how to adjust these weights. When the gradient is very small, the weights barely change, and learning slows down or even stops.

Sigmoid and tanh are particularly prone to vanishing gradients, especially in deep networks. ReLU is less susceptible because its gradient is either 0 or 1, which helps to prevent the gradient from becoming too small.

Think of a mountain climber taking tiny steps up a steep hill. They aren't making much progress and might get discouraged!

Choosing the Right Activation Function

So, how do you choose the right activation function for your task? Here's a general guideline:

- **Output Layer (Binary Classification):** Sigmoid is often a good choice for the output layer when you need to predict a probability (0 or 1). This classification requires that the values be normalized between 0 to 1.
- **Hidden Layers:** ReLU is a common choice for hidden layers due to its simplicity and resistance to vanishing gradients. Tanh can also be used, especially if you want to center the data.
- **Output Layer (Multi-Class Classification):** Softmax is a common output layer choice that scales all the values to probabilities.

The Softmax Function: Probabilities for Multiple Classes

Softmax is another activation function, typically used in the output layer for multi-class classification problems. This is the perfect choice when we need to pick one out of several classes. It transforms a vector of real numbers into a probability distribution, where each value represents the probability of belonging to a particular class.

The softmax function is defined as:

$$f(Z_i) = \frac{e^{Z_i}}{\sum e^{Z}}$$

where:

- Z_i is the input value for the i-th class.
- The sum is over all the classes.
- e is Euler's Number.

Back to the hiring example, we need to decide whether they were fit, maybe a fit, or not a fit. Our neural network spits out an array of values $[2.0, 1.0, 0.1]$. If these were used as input values to a softmax function, the output will be $[0.665, 0.244, 0.091]$.

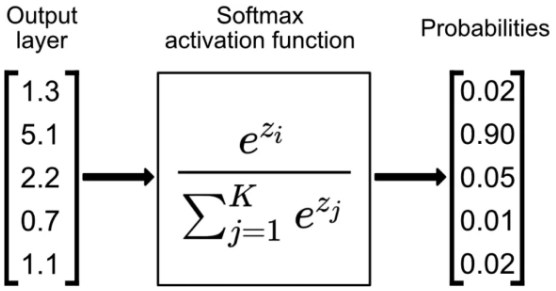

Figure 3: What Softmax does

This converts it to easy-to-understand probability. The candidate is most likely (0.665) a strong fit!

Conclusion

Activation functions are a crucial component of neural networks. They introduce non-linearity, enabling the model to learn complex patterns. By understanding the properties of different activation functions and their impact on training, you can build more effective and robust AI models. Experiment with different activation functions in your own projects and see how they affect performance!