Lecture 3: The Role of Activation Functions

Justo E. Karell

September 18, 2024

CS 583: Deep Learning

Introduction

Activation functions are a fundamental component of neural networks, playing a crucial role in enabling the network to learn and model complex relationships. This lecture focuses on understanding what activation functions are, their mathematical significance, and how they change the nature of the model. We will explore different types of activation functions and demonstrate their impact by computing the output of neural networks with one layer and two layers using these functions.

1. What is an Activation Function?

An activation function is a mathematical function applied to the output of a neuron in a neural network. It introduces non-linearity into the network, allowing it to model complex, non-linear relationships. Without activation functions, neural networks would be limited to modeling only linear transformations, no matter how many layers they have.

Mathematical Representation

In a neural network, the output of a neuron before the activation function is applied is given by the weighted sum:

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

Where:

- $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$ is the input vector of size n.

- $\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \in \mathbb{R}^n$ is the weight vector associated with the input.
- $b \in \mathbb{R}$ is the bias term.
- $z \in \mathbb{R}$ is the weighted sum (inner product) of the input and weight vectors, plus the bias.

The output of the neuron after applying the activation function is:

$$y = \sigma(z)$$

where σ is the activation function that introduces non-linearity.

Example:

Neural Network Without an Activation Function

Let's consider a simple single-layer perceptron with the following parameters:

- Input vector: $\mathbf{x} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$
- Weights: $\mathbf{w} = \begin{bmatrix} 0.5 \\ -0.4 \end{bmatrix}$
- Bias: $b = 1$

The weighted sum z is calculated using the inner product (dot product) of $\mathbf{w}^T\mathbf{x}$:

$$z = \mathbf{w}^T\mathbf{x} + b = (0.5)(2) + (-0.4)(-1) + 1 = 1.0 + 0.4 + 1 = 2.4$$

Thus, without an activation function, the output is simply $z = 2.4$. This is a linear transformation of the input.

Now, let's apply a sigmoid activation function $\sigma(z) = \frac{1}{1+e^{-z}}$ to the output z:

$$y = \sigma(2.4) = \frac{1}{1 + e^{-2.4}} \approx 0.916$$

In this case, the activation function "squashes" the output into a range between 0 and 1, making it suitable for tasks where outputs represent probabilities.

2. The Use of Activation Functions

Activation functions serve several key purposes in neural networks:

- **Introducing Non-linearity**:

  Without an activation function, each layer of the neural network would only perform a linear transformation of the inputs. Mathematically, if we define the operation of a neuron (without an activation function) as:

  $$f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

  where $\mathbf{W} \in \mathbb{R}^{m\times n}$ is a weight matrix, $\mathbf{x} \in \mathbb{R}^n$ is the input vector, and $\mathbf{b} \in \mathbb{R}^m$ is a bias vector, then applying multiple layers would result in:

  $$\mathbf{y} = \mathbf{W}_L\mathbf{W}_{L-1}\ldots\mathbf{W}_1\mathbf{x} + \mathbf{b}_L + \mathbf{b}_{L-1} + \ldots + \mathbf{b}_1$$

  which is still a linear transformation, regardless of the depth of the network. Activation functions $\sigma(\cdot)$ are essential for introducing non-linearity:

  $$\mathbf{y} = \sigma(\mathbf{W}_L\sigma(\mathbf{W}_{L-1}\ldots\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_{L-1}) + \mathbf{b}_L)$$

  Non-linearity enables the network to approximate complex functions and relationships, making it suitable for tasks involving non-linear patterns in the data.

- **Enabling Stacking of Layers**:

  When activation functions are applied between layers, the overall model becomes a composition of non-linear functions. Without non-linearity, stacking layers would offer no added representational power, as the entire model would collapse into a linear transformation. However, with activation functions, we can apply a series of non-linear transformations:

  $$\mathbf{y} = \sigma_L(\mathbf{W}_L\sigma_{L-1}(\mathbf{W}_{L-1}\ldots\sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \ldots) + \mathbf{b}_L)$$

  This allows the network to progressively capture more abstract and complex features at each layer, enabling the model to learn intricate patterns in the data.

- **Controlling Output Range**:

  Certain activation functions, such as the sigmoid or tanh functions, squash the output into a defined range. For example, the sigmoid function squashes output values into the range $(0, 1)$, and the tanh function maps output values into the range $(-1, 1)$. These transformations can be beneficial for specific tasks, such as binary classification, where the output represents a probability. Mathematically, the sigmoid function transforms the weighted sum $z \in \mathbb{R}$ into a value within $(0, 1)$:

  $$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \text{where} \quad z \in \mathbb{R}, \quad y = \sigma(z) \in (0, 1)$$

Similarly, the tanh function maps outputs into the range $(-1, 1)$:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad \text{where} \quad z \in \mathbb{R}, \quad y = \tanh(z) \in (-1, 1)$$

This ability to control the output range is essential in many machine learning tasks where predictions need to be bounded.

3. Types of Activation Functions

Activation functions play a key role in transforming the output of each neuron, and each function has different properties. In this section, we will cover several common activation functions, explain how they transform input values, and define their domain and range.

1. Sigmoid Function

The sigmoid function is defined as:

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

- **Domain**: $\mathbb{R}$ (all real numbers)
- **Range**: $(0, 1)$

Explanation:
For any input $z \in \mathbb{R}$, the sigmoid function squashes it into the range $(0, 1)$, making it suitable for tasks where the output represents probabilities.

- For large positive inputs, as $z \to +\infty$, $\sigma(z) \to 1$. - For large negative inputs, as $z \to -\infty$, $\sigma(z) \to 0$. - For $z = 0$, $\sigma(0) = 0.5$.

The sigmoid function compresses large positive and negative inputs into the extremes of the output range, but it can suffer from the vanishing gradient problem, as the derivative of the sigmoid approaches zero for large $|z|$.

2. Tanh Function (Hyperbolic Tangent)

The tanh function is defined as:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- **Domain**: $\mathbb{R}$ (all real numbers)
- **Range**: $(-1, 1)$

Explanation:

The tanh function transforms real input values into the range $(-1, 1)$, which means it centers the output around zero, unlike the sigmoid function.

- For large positive inputs, as $z \to +\infty$, $\tanh(z) \to 1$. - For large negative inputs, as $z \to -\infty$, $\tanh(z) \to -1$. - For $z = 0$, $\tanh(0) = 0$.

The tanh function is symmetric around the origin, making it more suitable than sigmoid when negative values are useful in the model.

3. ReLU (Rectified Linear Unit)

The ReLU function is defined as:

$$\sigma(z) = \max(0, z)$$

- **Domain**: $\mathbb{R}$ (all real numbers)
- **Range**: $[0, +\infty)$

Explanation:

The ReLU function outputs zero for all negative inputs and passes positive inputs unchanged. This introduces sparsity in the network (i.e., some neurons output zero), which helps reduce complexity and speeds up learning.

- For any $z > 0$, $\sigma(z) = z$. - For any $z \leq 0$, $\sigma(z) = 0$.

One drawback of ReLU is that it can lead to dead neurons when neurons output zero consistently and never recover (due to the zero gradient for negative inputs).

4. Leaky ReLU

The Leaky ReLU function is a variation of the ReLU function that allows small, non-zero outputs for negative inputs, defined as:

$$\sigma(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases}$$

where α is a small positive constant (e.g., $\alpha = 0.01$).

- **Domain**: $\mathbb{R}$ (all real numbers)
- **Range**: $(-\infty, +\infty)$

Explanation:

Leaky ReLU addresses the problem of dead neurons in ReLU by allowing a small gradient for negative inputs.

- For $z > 0$, $\sigma(z) = z$. - For $z \leq 0$, $\sigma(z) = \alpha z$, where α is a small constant, preventing the gradient from becoming zero for negative values.

5. Softmax Function

The softmax function is typically used in the output layer of a neural network for multi-class classification problems. It converts the raw output of each neuron into a probability distribution over all classes. The function is defined as:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

where $\mathbf{z} = [z_1, z_2, \ldots, z_n]$ is the vector of raw neuron outputs.

- **Domain**: $\mathbb{R}^n$ (a vector of real numbers)
- **Range**: $(0, 1)$ (for each element in the output vector)

Explanation:

The softmax function transforms a vector of real numbers into a probability distribution. The sum of all outputs equals 1, making it useful for classification tasks where the network needs to output probabilities over multiple classes.

- For large positive values of z_i, the corresponding probability $\sigma(\mathbf{z})_i$ approaches 1. - For small values of z_i, the corresponding probability $\sigma(\mathbf{z})_i$ approaches 0.

6. Swish Function

The Swish function is a smooth, non-monotonic activation function, defined as:

$$\sigma(z) = z \cdot \sigma_{\text{sigmoid}}(z) = \frac{z}{1 + e^{-\beta z}}$$

- **Domain**: $\mathbb{R}$ (all real numbers)
- **Range**: $(-0.278, +\infty)$

Explanation:

Swish allows the input to be scaled by the sigmoid of the input. It has shown to work better than ReLU in some tasks, offering smoother gradients. Here, β is either constant or a trainable parameter depending on the model. When $\beta = 1$, this becomes a Sigmoid Linear Unit.

- For large positive values, Swish behaves like ReLU ($z \to +\infty$, $\sigma(z) \to z$). - For large negative values, Swish saturates to a small negative value ($z \to -\infty$, $\sigma(z) \to 0$).

4. Example 1: Neural Network with One Layer and Sigmoid Activation Function

Network Parameters:

- Input vector: $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
- Weights: $\mathbf{w} = \begin{bmatrix} 0.5 \\ -0.4 \end{bmatrix}$
- Bias: $b = 1$
- Activation function: Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

Step 1: Compute the Weighted Sum (Dot Product)

The output z before applying the activation function is calculated as the dot product of $\mathbf{w}$ and $\mathbf{x}$, plus the bias b:

$$z = \mathbf{w}^T \mathbf{x} + b = (0.5)(1) + (-0.4)(2) + 1 = 0.5 - 0.8 + 1 = 0.7$$

Step 2: Apply the Sigmoid Activation Function

The sigmoid activation function is given by:

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

Applying this to $z = 0.7$:

$$y = \sigma(0.7) = \frac{1}{1 + e^{-0.7}} \approx 0.668$$

Result

Thus, the output of the neuron after applying the sigmoid activation function is $y \approx 0.668$.

5. Example 2: Neural Network with One Layer and ReLU Activation Function

Network Parameters:

- Input vector: $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
- Weights: $\mathbf{w} = \begin{bmatrix} 0.3 \\ -0.8 \end{bmatrix}$
- Bias: $b = 0.2$
- Activation function: ReLU $\sigma(z) = \max(0, z)$

Step 1: Compute the Weighted Sum (Dot Product)

The output z before applying the activation function is calculated as the dot product of $\mathbf{w}$ and $\mathbf{x}$, plus the bias b:

$$z = \mathbf{w}^T \mathbf{x} + b = (0.3)(1) + (-0.8)(2) + 0.2 = 0.3 - 1.6 + 0.2 = -1.1$$

Step 2: Apply the ReLU Activation Function

The ReLU activation function is defined as:

$$\sigma(z) = \max(0, z)$$

Applying this to $z = -1.1$:

$$y = \max(0, -1.1) = 0$$

Result

Thus, the output of the neuron after applying the ReLU activation function is $y = 0$.

6. Example 3: Neural Network with Two Layers and Sigmoid Activation Function

Network Parameters:

- Input vector: $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
- Weights for hidden layer: $\mathbf{W}_1 = \begin{bmatrix} 0.2 & 0.4 \\ -0.5 & 0.3 \end{bmatrix}$
- Biases for hidden layer: $\mathbf{b}_1 = \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$
- Weights for output layer: $\mathbf{W}_2 = \begin{bmatrix} 0.7 & -0.3 \end{bmatrix}$
- Bias for output layer: $b_2 = 0.05$
- Activation function: Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

Step 1: Compute Hidden Layer Output

The hidden layer output is computed as:

$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 = \begin{bmatrix} 0.2 & 0.4 \\ -0.5 & 0.3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix} = \begin{bmatrix} 1.1 \\ -0.1 \end{bmatrix}$$

Apply the sigmoid activation function:

$$\mathbf{h} = \sigma(\mathbf{z}_1) = \begin{bmatrix} \sigma(1.1) \\ \sigma(-0.1) \end{bmatrix} = \begin{bmatrix} 0.750 \\ 0.475 \end{bmatrix}$$

Step 2: Compute Output Layer

The output layer is computed as:

$$z_2 = \mathbf{W}_2\mathbf{h} + b_2 = \begin{bmatrix} 0.7 & -0.3 \end{bmatrix} \begin{bmatrix} 0.750 \\ 0.475 \end{bmatrix} + 0.05 = 0.4825$$

Apply the sigmoid activation function:

$$y = \sigma(z_2) = \frac{1}{1 + e^{-0.4825}} \approx 0.618$$

Result

Thus, the final output of the network is $y \approx 0.618$.

7. Example 4: Neural Network with Two Layers and Tanh Activation Function

Network Parameters:

- Input vector: $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
- Weights for hidden layer: $\mathbf{W}_1 = \begin{bmatrix} 0.2 & 0.4 \\ -0.5 & 0.3 \end{bmatrix}$
- Biases for hidden layer: $\mathbf{b}_1 = \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$
- Weights for output layer: $\mathbf{W}_2 = \begin{bmatrix} 0.7 & -0.3 \end{bmatrix}$
- Bias for output layer: $b_2 = 0.05$
- Activation function: Tanh $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Step 1: Compute Hidden Layer Output

The hidden layer output is computed as:

$$\mathbf{z}_1 = \begin{bmatrix} 1.1 \\ -0.1 \end{bmatrix}$$

Apply the tanh activation function:

$$\mathbf{h} = \tanh(\mathbf{z}_1) = \begin{bmatrix} \tanh(1.1) \\ \tanh(-0.1) \end{bmatrix} = \begin{bmatrix} 0.800 \\ -0.100 \end{bmatrix}$$

Step 2: Compute Output Layer

The output layer is computed as:

$$z_2 = \mathbf{W}_2\mathbf{h} + b_2 = \begin{bmatrix} 0.7 & -0.3 \end{bmatrix} \begin{bmatrix} 0.800 \\ -0.100 \end{bmatrix} + 0.05 = 0.635$$

Apply the tanh activation function:

$$y = \tanh(z_2) = \tanh(0.635) \approx 0.561$$

Result

Thus, the final output of the network is $y \approx 0.561$.

8. Example 5: Neural Network with Two Layers and ReLU Activation Function

Network Parameters:

- Input vector: $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
- Weights for hidden layer: $\mathbf{W}_1 = \begin{bmatrix} 0.2 & 0.4 \\ -0.5 & 0.3 \end{bmatrix}$
- Biases for hidden layer: $\mathbf{b}_1 = \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$
- Weights for output layer: $\mathbf{W}_2 = \begin{bmatrix} 0.7 & -0.3 \end{bmatrix}$
- Bias for output layer: $b_2 = 0.05$
- Activation function: ReLU $\sigma(z) = \max(0, z)$

Step 1: Compute Hidden Layer Output

The hidden layer output is computed as:

$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 = \begin{bmatrix} 0.2 & 0.4 \\ -0.5 & 0.3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix} = \begin{bmatrix} 1.1 \\ -0.1 \end{bmatrix}$$

Apply the ReLU activation function:

$$\mathbf{h} = \max(0, \mathbf{z}_1) = \begin{bmatrix} \max(0, 1.1) \\ \max(0, -0.1) \end{bmatrix} = \begin{bmatrix} 1.1 \\ 0 \end{bmatrix}$$

Step 2: Compute Output Layer

The output layer is computed as:

$$z_2 = \mathbf{W}_2\mathbf{h} + b_2 = \begin{bmatrix} 0.7 & -0.3 \end{bmatrix} \begin{bmatrix} 1.1 \\ 0 \end{bmatrix} + 0.05 = 0.82$$

Apply the ReLU activation function:

$$y = \max(0, z_2) = \max(0, 0.82) = 0.82$$

Result

Thus, the final output of the network is $y = 0.82$.

Conclusion

Activation functions are critical in neural networks for introducing non-linearity and enabling the network to learn complex patterns. Different activation functions offer various trade-offs. Sigmoid provides smooth probabilities for binary tasks, tanh handles symmetric data more effectively, and ReLU ensures faster learning by mitigating gradient issues with positive inputs. Choosing the appropriate activation function can significantly impact the model's performance and training efficiency.