MACHINE LEARNING WORKINGS

Charles Ndung'u

July 2024

Derivation of Sigmoid Activation Function Using Reciprocal Method

The Sigmoid activation function, often used in neural networks, is defined as:

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

Step-by-Step Derivation

1. **Initial Form:**

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

2. **Multiply by e^x:**
To simplify, we multiply both the numerator and the denominator by e^x:

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

3. **Simplify the Denominator:**

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

The denominator $(1 + e^{-x}) \cdot e^x$ simplifies as follows:

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

4. **Reciprocal of the Sigmoid Function:**
Taking the reciprocal of the function:

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

5. **Simplify the Reciprocal:**
Multiply both the numerator and the denominator by e^x:

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

Thus, the sigmoid function in terms of its reciprocal representation is:

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

Properties of the Sigmoid Function

1. **Range:**
   The output of the Sigmoid function lies between 0 and 1:

$$0 < \sigma(x) < 1$$

2. **Derivative:**
   The derivative of the Sigmoid function is useful for backpropagation in neural networks:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Graph of the Sigmoid Function

The Sigmoid function produces an S-shaped curve, which can be plotted as follows:

x	$\sigma(x)$
-2	≈ 0.1192
-1	≈ 0.2689
0	0.5
1	≈ 0.7311
2	≈ 0.8808

Derivation of ReLU Activation Function

The Rectified Linear Unit (ReLU) activation function is defined as:

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

This function can be expressed using the Heaviside step function $H(x)$, which is defined as:

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

Using the Heaviside step function, the ReLU activation function can be written as:

$$\text{ReLU}(x) = x \cdot H(x)$$

To derive this, consider the piecewise definition of the ReLU function and the Heaviside step function:

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

When $x < 0$:

$$\text{ReLU}(x) = 0 = x \cdot H(x) = x \cdot 0 = 0$$

When $x \geq 0$:

$$\text{ReLU}(x) = x = x \cdot H(x) = x \cdot 1 = x$$

Thus, we have shown that:

$$\text{ReLU}(x) = x \cdot H(x)$$

Derivation of Hyperbolic Functions and Euler's Formula from Taylor Series

Hyperbolic Functions

The hyperbolic sine function $\sinh(x)$ and hyperbolic cosine function $\cosh(x)$ are defined as follows:

$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

1. **Taylor Series Derivation:**
For e^x:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Derivation of $\sinh(x)$ and $\cosh(x)$:
From Euler's formula:

$$e^x = \cosh(x) + \sinh(x)$$

$$e^{-x} = \cosh(x) - \sinh(x)$$

Adding these two equations:

$$e^x + e^{-x} = 2\cosh(x)$$

Solving for $\cosh(x)$:

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

Subtracting the second equation from the first:

$$e^x - e^{-x} = 2\sinh(x)$$

Solving for $\sinh(x)$:

$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

Derivation of $\cosh(x)$**:**
From Euler's formula:

$$e^{ix} = \cos(x) + i\sin(x)$$

Taking the real part:

$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$

Using $e^{ix} = \cosh(ix)$:

$$\cos(x) = \cosh(ix)$$

So,

$$\cosh(x) = \cos(ix)$$

Derivation of $\tanh(x)$**:**

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

Substituting $\sinh(x)$ and $\cosh(x)$:

$$\tanh(x) = \frac{\frac{e^x - e^{-x}}{2}}{\frac{e^x + e^{-x}}{2}}$$

Simplifying gives:

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

2. **Using Taylor Series for e^x and e^{-x}:**

$$\sinh(x) = \frac{e^x - e^{-x}}{2} = \frac{\left(1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots\right) - \left(1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots\right)}{2}$$

$$\sinh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots$$

$$\cosh(x) = \frac{e^x + e^{-x}}{2} = \frac{\left(1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots\right) + \left(1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots\right)}{2}$$

$$\cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \cdots$$

3. **Tanh Function Derivation:**
The hyperbolic tangent function $\tanh(x)$ is defined as:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

Substituting the Taylor series expansions of $\sinh(x)$ and $\cosh(x)$:

$$\tanh(x) = \frac{x + \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots}{1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \cdots}$$

Euler's Formula from Taylor Series

Euler's formula relates the complex exponential function to trigonometric functions:

$$e^{ix} = \cos(x) + i\sin(x)$$

1. **Taylor Series for e^{ix}:**

$$e^{ix} = \sum_{n=0}^{\infty} \frac{(ix)^n}{n!} = 1 + ix - \frac{x^2}{2!} - i\frac{x^3}{3!} + \frac{x^4}{4!} + i\frac{x^5}{5!} - \cdots$$

2. **Separating Real and Imaginary Parts:**

$$e^{ix} = \cos(x) + i\sin(x)$$

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$$

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

Activation Functions in The Model

ReLU (Rectified Linear Unit)

Layers:

- The Dense layers with 256 and 128 units use ReLU as the activation function.

```
model.add(Dense(256, input_shape=(max_length,), activation='relu'))
model.add(Dense(128, activation='relu'))
```

Role:

- **ReLU Function:**

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

- ReLU is widely used in neural networks because it helps mitigate the vanishing gradient problem by allowing only positive values to pass through while setting negative values to zero.
- It helps in introducing non-linearity to the model, which enables the network to learn complex patterns.
- ReLU is computationally efficient, making the training process faster.

Softmax

Layer:

- The final Dense layer with the number of units equal to the number of classes (`len(train_y[0])`) uses Softmax as the activation function.

```
model.add(Dense(len(train_y[0]), activation='softmax'))
```

Role:

- **Softmax Function:**

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- The Softmax function converts the output of the final layer into a probability distribution over the classes.

- Each output value lies between 0 and 1, and the sum of all output values equals 1.
- It is used in multi-class classification problems where the model needs to assign probabilities to each class.

Summary of Activation Functions

ReLU

- **Purpose:** Introduces non-linearity, avoids vanishing gradient problem, and improves computational efficiency.
- **Usage:** Used in hidden layers (Dense layers with 256 and 128 units).

Softmax

- **Purpose:** Converts output logits into a probability distribution over classes.
- **Usage:** Used in the final output layer for multi-class classification.

Full Code Context

Here is the relevant section of your model code with activation functions highlighted:

```
model = Sequential()
model.add(Dense(256, input_shape=(max_length,), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))
```

Explanation

- **First Dense Layer:** Uses 256 units with ReLU activation to capture complex patterns from the input.
- **Dropout Layer:** Helps prevent overfitting by randomly setting a fraction of input units to 0 during training.
- **Second Dense Layer:** Uses 128 units with ReLU activation to further process the learned features.
- **Dropout Layer:** Again, helps prevent overfitting.

- **Final Dense Layer:** Uses a number of units equal to the number of classes with Softmax activation to output the probability distribution over the classes.

These activation functions help your model learn effectively from the input data and make accurate predictions for the chatbot responses.

1 Introduction

Diagram Explanation

Input Layer

- Receives the input features (bag of words) of length `max_length`.

Dense Layer 1

- **Neurons**: 256
- **Activation Function**: ReLU
- **Dropout**: 50%

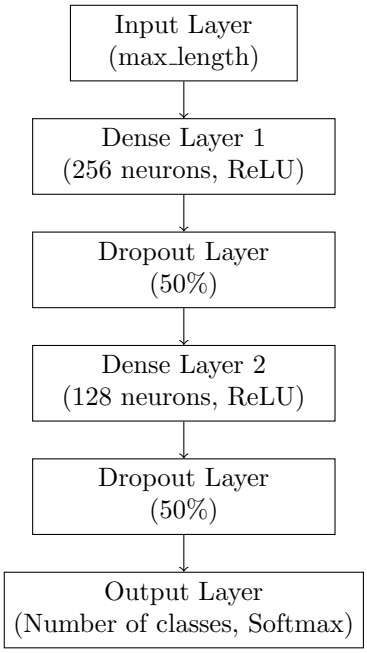
Dense Layer 2

- **Neurons**: 128
- **Activation Function**: ReLU
- **Dropout**: 50%

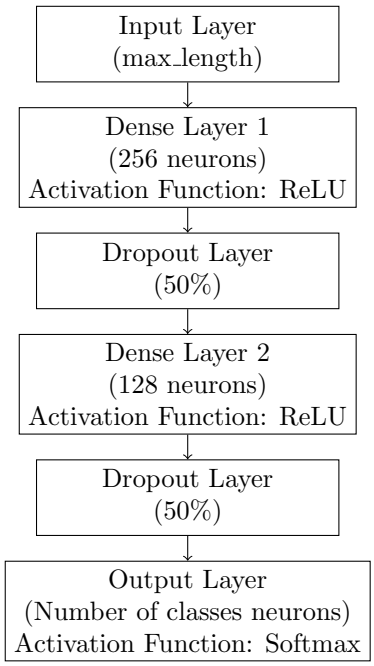
Output Layer

- **Neurons**: Number of classes
- **Activation Function**: Softmax

Diagram


Input Layer
(max_length)
Dense Layer 1
(256 neurons, ReLU)
Dropout Layer
(50%)
Dense Layer 2
(128 neurons, ReLU)
Dropout Layer
(50%)
Output Layer
(Number of classes, Softmax)

Detailed Representation



Description of Each Layer

Input Layer

The input layer takes the preprocessed input data (bag of words) with a length equal to `max_length`.

Dense Layer 1

- **Neurons**: 256
- **Activation Function**: ReLU
- Applies the ReLU function to the weighted sum of inputs.
- Output is passed to the next layer.

Dropout Layer

- Randomly sets 50% of the input units to 0 at each update during training time, which helps prevent overfitting.

Dense Layer 2

- **Neurons**: 128
- **Activation Function**: ReLU
- Applies the ReLU function to the weighted sum of inputs.
- Output is passed to the next layer.

Dropout Layer

- Randomly sets 50% of the input units to 0 at each update during training time, which helps prevent overfitting.

Output Layer

- **Neurons**: Number of classes (equal to the number of possible responses).
- **Activation Function**: Softmax
- Converts the outputs into a probability distribution over the classes, indicating the model's confidence in each class.

ReLU (Rectified Linear Unit) is considered one of the best activation functions in neural networks due to several advantages it offers over other activation functions:

1. **Simplicity:** ReLU is computationally efficient and easy to implement, involving simple mathematical operations (max function).
2. **Avoids Vanishing Gradient:** Unlike activation functions like sigmoid or tanh that saturate at high or low values, causing gradients to vanish, ReLU does not saturate in the positive region (outputting zero for negative inputs), thereby mitigating the vanishing gradient problem.
3. **Faster Convergence:** ReLU accelerates the convergence of stochastic gradient descent (SGD) compared to saturating activation functions because its derivative is either 0 or 1, simplifying the gradient computation.
4. **Sparsity:** ReLU can lead to sparsity in neural networks by zeroing out negative values, which can be beneficial in certain types of networks, reducing the computational load and overfitting.
5. **Better Representational Power:** ReLU allows neural networks to learn complex representations more effectively due to its non-linear nature and ability to model intricate relationships in data.
6. **Biological Plausibility:** ReLU is loosely inspired by the behavior of biological neurons, where they either fire (output a signal) or do not fire based on their input.

7. **State-of-the-Art Performance:** In practice, ReLU has been found to perform well across a wide range of tasks and is a default choice in many deep learning models.

Despite its advantages, ReLU also has limitations, such as the "dying ReLU" problem where neurons can permanently output zero if their weights are updated in a way that keeps their output always negative. Techniques like Leaky ReLU and variants such as Parametric ReLU (PReLU) and Exponential Linear Unit (ELU) have been developed to address some of these issues while retaining the benefits of ReLU.

The main purpose of using activation functions in deep learning, especially nonlinear activation functions like ReLU (Rectified Linear Unit), sigmoid, tanh, etc., is to introduce nonlinearity into the network. Neural networks need to learn complex patterns and relationships in data that are often nonlinear in nature. Here are the key reasons why activation functions are crucial:

1. **Introducing Nonlinearity:** Without activation functions, neural networks would simply be linear transformations of their inputs, regardless of the number of layers. This would severely limit their ability to learn and represent complex patterns in data.

2. **Learning Complex Functions:** By applying nonlinear activation functions, neural networks can learn and approximate complex, nonlinear mappings between input and output data. This enables them to model intricate relationships that exist in real-world datasets.

3. **Capturing Higher-Level Features:** Activation functions allow neural networks to capture and represent higher-level features in data, which are often not linearly separable. For example, in image recognition tasks, networks need to detect edges, textures, and shapes—tasks that require nonlinear transformations of the input data.

4. **Gradient Propagation:** Activation functions play a critical role in backpropagation, the algorithm used to update the weights of neural networks during training. They ensure that gradients can be calculated and propagated efficiently through the network layers, facilitating learning and convergence.

5. **Stabilizing Outputs:** Certain activation functions, like sigmoid and tanh, normalize the outputs of neurons to a specific range (e.g., [0, 1] for sigmoid), which can be beneficial for certain types of tasks, such as binary classification.