Activation functions

Contents

[What are the Activation functions: 1](#_Toc173490842)

[How and when and why where we used the activation functions: 3](#_Toc173490843)

[TensorFlow Example 5](#_Toc173490844)

[Explanation 6](#_Toc173490845)

# What are the Activation functions:

Activation functions are a critical component of neural networks, determining the output of a node (neuron) given an input or set of inputs. They introduce non-linearity into the network, allowing it to model complex relationships between inputs and outputs. Here are some commonly used activation functions:

1. **Sigmoid (Logistic) Function**:
   * **Range**: (0, 1)
   * **Pros**: Smooth gradient, output values bound between 0 and 1, traditionally used in binary classification problems.
   * **Cons**: Prone to vanishing gradient problem, outputs not zero-centered.
2. **Hyperbolic Tangent (tanh) Function**:

**The x is power here**

* + **Range**: (-1, 1)
  + **Pros**: Zero-centered output, smooth gradient.
  + **Cons**: Still suffers from vanishing gradient problem for large positive or negative inputs.

1. **Rectified Linear Unit (ReLU)**:
   * **Range**: [0, ∞)
   * **Pros**: Computationally efficient, mitigates vanishing gradient problem, accelerates convergence.
   * **Cons**: Can suffer from "dying ReLU" problem where neurons get stuck during training and only output 0.
2. **Leaky ReLU**:
   * **Range**: (-∞, ∞)
   * **Pros**: Addresses "dying ReLU" problem by allowing a small, non-zero gradient when the unit is inactive.
3. **Parametric ReLU (PReLU)**:
   * **Range**: (-∞, ∞)
   * **Pros**: Similar to Leaky ReLU but with α\alphaα learned during training, offering more flexibility.
4. **Exponential Linear Unit (ELU)**:
   * **Range**: (-α, ∞)
   * **Pros**: Smooth and differentiable, mitigates vanishing gradient problem, output closer to zero mean, which speeds up learning.
5. **Swish**:
   * **Range**: (-∞, ∞)
   * **Pros**: Smooth and non-monotonic, has shown to outperform ReLU in certain cases.

**Choosing an Activation Function**

* **Sigmoid and tanh**: Typically used in the output layer of binary classification problems.
* **ReLU and its variants**: Popular for hidden layers in deep networks due to efficiency and reduced likelihood of vanishing gradients.
* **ELU and Swish**: Can provide slight performance improvements over ReLU, particularly in deeper networks.

In practice, ReLU and its variants are often the default choice for hidden layers due to their simplicity and effectiveness. The choice of activation function can significantly impact the performance and training efficiency of the neural network.

# How and when and why where we used the activation functions:

**How Activation Functions are Used:**

1. **Computing Neuron Output**:
   * Each neuron in a neural network computes a weighted sum of its inputs, adds a bias, and applies an activation function to this sum.

where www are the weights, xxx are the inputs, bbb is the bias, and σ\sigmaσ is the activation function.

1. **Layer-wise Application**:
   * Activation functions are applied to each neuron in a layer, transforming the input before passing it to the next layer.

**When Activation Functions are Used:**

1. **Hidden Layers**:
   * **During Training**: Activation functions are used after each layer's linear transformation during forward propagation.
   * **During Inference**: Activation functions are also applied during the inference phase to compute the final output from the input data.
2. **Output Layer**:
   * **Regression Problems**: Typically use a linear activation function or none at all, as the output needs to represent a continuous value.
   * **Binary Classification**: Use the sigmoid activation function to output probabilities between 0 and 1.
   * **Multiclass Classification**: Use the softmax activation function to produce a probability distribution over classes.

**Why Activation Functions are Used:**

1. **Introduce Non-linearity**:
   * Activation functions introduce non-linear properties to the network, enabling it to learn complex patterns and functions. Without non-linearity, the network would be equivalent to a single-layer linear model.
2. **Enable Deep Learning**:
   * Non-linear activation functions allow networks with multiple layers (deep networks) to learn and model complex data through hierarchical feature extraction.
3. **Control Output Range**:
   * Activation functions like sigmoid and tanh compress outputs to a specific range, which is useful for ensuring stable outputs or probabilities.

**Where Activation Functions are Used:**

1. **Hidden Layers**:
   * **ReLU (Rectified Linear Unit)**: Commonly used due to its simplicity and efficiency. It allows networks to learn faster and perform better in practice.

**Code:**

**import numpy as np def relu(x): return np.maximum(0, x)**

**Leaky ReLU**: Used to address the "dying ReLU" problem by allowing a small gradient when the unit is inactive.

**Code**:

def leaky\_relu(x, alpha=0.01):

return np.where(x > 0, x, alpha \* x)

**ELU (Exponential Linear Unit)**: Used to improve learning by producing outputs that are closer to zero mean, speeding up training.

**Code**:

def elu(x, alpha=1.0):

return np.where(x > 0, x, alpha \* (np.exp(x) - 1))

**2. Output Layers**:

**Sigmoid:** Used for binary classification problems to output probabilities.

**Code:**

**def sigmoid(x):**

**return 1 / (1 + np.exp(-x))**

**Softmax**: Used for multiclass classification to output a probability distribution.

**Code:**

def softmax(x):

exps = np.exp(x - np.max(x)) return exps / np.sum(exps, axis=0)

**Linear**: Often used in regression problems where the output is a continuous value.

# TensorFlow Example

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, ReLU, Softmax

# Define the neural network model

model = Sequential([

Dense(4, input\_shape=(3,)), # First hidden layer with 4 neurons

ReLU(), # ReLU activation function

Dense(3), # Output layer with 3 neurons

Softmax() # Softmax activation function

])

# Compile the model

model.compile(optimizer='sgd', loss='categorical\_crossentropy', metrics=['accuracy'])

# Example input tensor (batch size of 5)

X = tf.random.normal((5, 3))

# Forward pass (predict)

output = model(X)

print("Output:\n", output)

# Example target tensor (one-hot encoded labels for batch size of 5)

y = tf.constant([[1, 0, 0], [0, 1, 0], [0, 0, 1], [0, 1, 0], [1, 0, 0]])

# Compute loss and print it

loss = model.evaluate(X, y, verbose=0)

print("Loss:", loss)

# Train the model (example with one step)

model.fit(X, y, epochs=1, verbose=1)

# Explanation

1. **Model Definition**:
   * A Sequential model is used to stack layers sequentially.
   * Dense layers are used to create fully connected layers.
   * ReLU() and Softmax() are used to apply ReLU and softmax activation functions, respectively.
2. **Model Compilation**:
   * The model is compiled with the SGD optimizer and categorical\_crossentropy loss function, which is suitable for multiclass classification.
3. **Example Input and Forward Pass**:
   * An example input tensor X is created with a batch size of 5 and input size of 3.
   * The forward pass is performed using model(X), which computes the output of the model for the given input.
4. **Example Target and Loss Calculation**:
   * An example target tensor y is created with one-hot encoded class labels for the batch.
   * The loss is computed using model.evaluate(), which returns the loss and other metrics.
5. **Training**:
   * The model is trained for one epoch using model.fit(), which performs a backward pass and updates the model parameters based on the computed gradients.

# What are the activation Functions:

In artificial intelligence, each neuron forms a weighted sum of its inputs and passes the resulting scalar value through a function referred to as an activation function. If a neuron has n inputs, then the output or activation of a neuron is:

This function g is referred to as the activation function.

In short activation function is a input output gate, the activated function decided which neuron will active.

If we did not apply the activation function in our model training, it will give the result in liner regression, and we will never get nonlinear results

## Why Activation functions are needed:

It adds to you model the power to catch the nonlinear pattern

# Ideal Activation function

* + - Nonlinear:
    - Differential:
    - Computationally:
    - Zero centered: