

Ann Practical 1

ChatGPT 3.5

```
You
Write a Python program to plot a few activation functions that are being used in neural
networks
Sloution:
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-10, 10)
plt.xlabel("Input")
plt.ylabel("Output")
plt.plot(x, x, label='Linear')
plt.grid(True)
plt.legend()
plt.show()
x = np.linspace(10,-10,100)
y = 1/(1+np.exp(-x))
plt.xlabel("Input")
plt.ylabel("Sigmoid(X)")
plt.plot(x,y)
plt.grid(True)
plt.show()
x = np.linspace(-10, 10)
plt.xlabel("Input")
plt.ylabel("Tanh(X)")
plt.plot(x,np.tanh(x))
plt.grid(True)
plt.show()
x = np.linspace(-10, 10, 100)
plt.plot(x, np.maximum(0, x))
plt.xlabel("Input")
plt.ylabel("ReLU(x)")
plt.grid(True)
plt.show()
import numpy as np
import matplotlib.pyplot as plt
def softmax(x):
  exp_values = np.exp(x - np.max(x))
  return exp_values / np.sum(exp_values)
x = np.linspace(-5, 5, 100)
y = softmax(x)
plt.xlabel("Input")
```



```
plt.ylabel("Output")
plt.plot(x, y)
plt.title("Softmax Activation Function")
plt.grid(True)
plt.show()
```

I want you to become my teacher and explain me everything that i need to know about the topics and finally explain code.

S ChatGPT

Of course, I'd be happy to help! Let's start with the basics.

Activation Functions in Neural Networks:

In neural networks, activation functions play a crucial role in introducing non-linearity to the model. They help the network learn complex patterns and relationships in the data. Here are some commonly used activation functions:

- 1. **Linear Activation Function**: The simplest activation function, where the output is directly proportional to the input. It's mostly used in regression problems.
- 2. **Sigmoid Activation Function**: This function squashes the input values between 0 and 1, which makes it suitable for binary classification problems, where the output needs to be in the range [0, 1].
- 3. **Tanh (Hyperbolic Tangent) Activation Function**: Similar to the sigmoid function, but it squashes the input values between -1 and 1. It's often used in hidden layers of neural networks.
- 4. **ReLU (Rectified Linear Unit) Activation Function**: ReLU is a piecewise linear function that outputs the input directly if it's positive, and zero otherwise. It helps in dealing with the vanishing gradient problem and speeds up the training process.
- 5. **Softmax Activation Function**: This function is primarily used in the output layer of multi-class classification problems. It converts raw scores into probabilities, ensuring that the sum of probabilities across all classes is equal to 1.

Understanding the Code:

Now, let's dissect the provided Python code:

```
import numpy as np
import matplotlib.pyplot as plt
```

This imports necessary libraries: `numpy` for numerical computations and `matplotlib` for plotting.

python



```
x = np.linspace(-10, 10)
plt.xlabel("Input")
plt.ylabel("Output")
plt.plot(x, x, label='Linear')
plt.grid(True)
plt.legend()
plt.show()
```

This section plots the linear activation function. `np.linspace()` generates an array of evenly spaced numbers over a specified interval (-10 to 10 in this case). It plots the input against itself, showing a linear relationship.

```
python

x = np.linspace(10,-10,100)
y = 1/(1+np.exp(-x))
plt.xlabel("Input")
plt.ylabel("Sigmoid(X)")
plt.plot(x,y)
plt.prid(True)
plt.show()
```

This section plots the sigmoid activation function. It calculates the sigmoid values for the given range of inputs using the formula $\frac{1}{1+e^{-x}}$.

```
python

x = np.linspace(-10, 10)
plt.xlabel("Input")
plt.ylabel("Tanh(X)")
plt.plot(x,np.tanh(x))
plt.grid(True)
plt.show()
```

Here, the code plots the hyperbolic tangent (tanh) activation function. It uses `np.tanh()` to calculate the tanh values for the given input range.

```
python

x = np.linspace(-10, 10, 100)
plt.plot(x, np.maximum(0, x))
plt.xlabel("Input")
plt.ylabel("ReLU(x)")
plt.grid(True)
plt.show()
```

This section plots the ReLU activation function. It uses `np.maximum()` to ensure that only positive values are returned for the given input range.

```
def softmax(x):
    exp_values = np.exp(x - np.max(x))
    return exp_values / np.sum(exp_values)

x = np.linspace(-5, 5, 100)
y = softmax(x)

plt.xlabel("Input")
```



```
plt.ylabel("Output")
plt.plot(x, y)
plt.title("Softmax Activation Function")
plt.grid(True)
plt.show()
```

Finally, this part defines a softmax function and plots its output. The softmax function converts the raw scores (logits) into probabilities, ensuring that the sum of probabilities across all classes is equal to 1.

Let me know if you have any questions or need further clarification!



for mula of linear activation function

S ChatGPT

The linear activation function, often referred to simply as "identity" function, is one of the simplest activation functions used in neural networks. It essentially performs a linear transformation on the input, where the output is directly proportional to the input.

Mathematically, the formula for the linear activation function is:

$$f(x) = x$$

Where:

- *x* is the input to the function.
- f(x) is the output of the function, which is equal to the input x.

In other words, the output of the linear activation function is the same as the input. This function is primarily used in regression problems, where the goal is to predict continuous values, and the network needs to learn a linear relationship between the input features and the target variable.

In Python code, you can represent the linear activation function simply as:

```
python

def linear_activation(x):
    return x
```

This function takes an input `x` and returns the same value as the output.

You sigmoid



S ChatGPT

The sigmoid activation function, also known as the logistic function, is a widely used activation function in neural networks, particularly in binary classification tasks. It maps any real-valued number to a value between 0 and 1. The sigmoid function's characteristic S-shaped curve ensures that the output is bounded, making it suitable for problems where we need to predict probabilities.

The formula for the sigmoid activation function is:

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

Where:

- *x* is the input to the function.
- *e* is the base of the natural logarithm, approximately equal to 2.71828.
- f(x) is the output of the function, which is a value between 0 and 1.

In Python, you can implement the sigmoid activation function using the NumPy library as follows:

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

This function takes an input `x` and applies the sigmoid transformation to it using the `np.exp()` function from NumPy, which calculates the exponential of each element in the input array.

Here's an example of how to use this function:

```
import matplotlib.pyplot as plt

# Generate input values
x = np.linspace(-10, 10, 100)

# Apply sigmoid activation function
y = sigmoid(x)

# Plot the sigmoid function
plt.plot(x, y)
plt.xlabel('Input')
plt.ylabel('Output')
plt.title('Sigmoid Activation Function')
plt.grid(True)
plt.show()
```

This code will plot the sigmoid function and visualize its characteristic S-shaped curve.

P You

why its popular?



S ChatGPT

The sigmoid activation function is popular for several reasons:

- 1. **Output Range**: The sigmoid function squashes its input to a range between 0 and 1. This property makes it convenient for tasks where we need to predict probabilities, especially in binary classification problems where we want to output the probability of a sample belonging to a particular class. The output can be interpreted as the probability of the input belonging to the positive class.
- 2. **Smooth Gradient**: The sigmoid function's derivative has a smooth and continuous gradient across its entire domain. This smoothness facilitates efficient optimization during the training of neural networks using gradient-based methods like gradient descent. It helps prevent issues such as exploding or vanishing gradients, which can hinder the training process in deeper networks.
- 3. **Non-linearity**: While the sigmoid function is bounded and monotonically increasing, it is still non-linear. This non-linear characteristic allows neural networks to model complex relationships and decision boundaries in the data, enabling them to learn and generalize well to unseen examples.
- 4. **Historical Significance**: The sigmoid function has been historically used in the field of neural networks and artificial intelligence. It was one of the earliest activation functions proposed and has been extensively studied and applied in various neural network architectures. Its familiarity and historical significance contribute to its continued usage in many applications.
- 5. **Differentiability**: The sigmoid function is differentiable everywhere, allowing for the computation of gradients needed for backpropagation, which is crucial for updating the weights of the neural network during training.

Overall, the sigmoid activation function's properties make it a versatile choice for many neural network architectures, particularly in scenarios involving binary classification and where the output needs to be interpreted as probabilities. However, it's worth noting that the sigmoid function has limitations, such as the vanishing gradient problem for very large or very small inputs, which has led to the exploration and adoption of alternative activation functions like ReLU and its variants in certain situations.

P You

There is a point in my text book regarding its popularity that, " It satisfies the property between derivative and itself susch that its computationally easy to perform.

S ChatGPT

Yes, that's a valid point from a computational perspective. One of the advantages of the sigmoid activation function is that its derivative can be expressed in terms of the function itself. This property simplifies the computation of gradients during backpropagation, which is essential for training neural networks using optimization algorithms like gradient descent.

The derivative of the sigmoid function $\sigma(x)$ with respect to its input x can be expressed as:

$$\frac{d}{dx}\sigma(x) = \sigma(x) \cdot (1 - \sigma(x))$$



This derivative can be directly computed using the output of the sigmoid function $\sigma(x)$ without needing to recompute the sigmoid itself, which saves computational resources. This property is beneficial for training neural networks efficiently, especially in deep architectures where the computational cost of computing gradients can become significant.

Additionally, the sigmoid function has a simple and straightforward mathematical form, which further contributes to its computational ease. These factors collectively make the sigmoid activation function computationally convenient and contribute to its popularity, particularly in the early days of neural network research and development.



Explain in simple language

S ChatGPT

Sure, let's break it down into simpler terms.

When we train a neural network, we adjust its parameters (weights and biases) to minimize the difference between the predicted outputs and the actual outputs. To do this, we use an optimization algorithm called gradient descent, which requires calculating the gradients (derivatives) of the loss function with respect to the parameters.

Now, the sigmoid function, which is often used as an activation function in neural networks, has a nice property: its derivative can be expressed in terms of the function itself. In other words, if we know the value of the sigmoid function at a certain point, we can easily calculate its derivative at that point without having to redo the whole sigmoid calculation again.

This property is handy because during training, we need to compute a lot of derivatives for many points in the network. If we had to redo the sigmoid calculation every time we wanted to find its derivative, it would be computationally expensive. But since we can use the sigmoid value we already calculated, it makes the computation much simpler and faster.

So, this property of the sigmoid function, where its derivative can be expressed using the function itself, makes it computationally easy to work with during training. That's one of the reasons why it's popular in neural networks.



tahn

S ChatGPT

Certainly! The hyperbolic tangent function, commonly known as tanh, is another popular activation function used in neural networks. It shares some similarities with the sigmoid function but has a different range and shape.



The formula for the tanh activation function is:

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

Where:

- *x* is the input to the function.
- e is the base of the natural logarithm, approximately equal to 2.71828.
- f(x) is the output of the function, which is a value between -1 and 1.

Here's why tanh is popular:

- 1. **Zero-Centered**: Unlike the sigmoid function, which is bounded between 0 and 1, tanh is centered around zero and ranges from -1 to 1. This property makes it suitable for situations where the input may have negative values, as it can capture both positive and negative relationships in the data.
- 2. **Non-linearity**: Similar to the sigmoid function, tanh is also non-linear. It allows neural networks to model complex relationships and decision boundaries in the data, enabling them to learn and generalize well to unseen examples.
- 3. **Gradient Magnitude**: The tanh function has steeper gradients compared to the sigmoid function, especially around zero. This property can accelerate learning during training, as it provides stronger signals for weight updates, particularly in deeper architectures.
- 4. **Smoothness**: Like the sigmoid function, tanh has a smooth and continuous derivative across its entire domain. This smoothness facilitates efficient optimization during training using gradient-based methods.

In Python, you can implement the tanh activation function using the NumPy library:

```
import numpy as np

def tanh(x):
    return np.tanh(x)
```

This function takes an input `x` and applies the tanh transformation to it using the `np.tanh()` function from NumPy.

Overall, the tanh activation function is a popular choice in neural networks due to its zero-centeredness, non-linearity, and suitability for capturing both positive and negative relationships in the data.



NameError Traceback (most recent call last)

Cell In[13], line 5

2 x = np.linspace(-10, 10, 100)

4 # Apply sigmoid activation function



```
----> 5 y = sigmoid(x)
7 # Plot the sigmoid function
8 plt.plot(x, y)
```

NameError: name 'sigmoid' is not defined

S ChatGPT

It seems like there's an issue with the code. It's trying to use a function called `sigmoid`, but it hasn't been defined yet. Let's correct that.

Here's how you can define the sigmoid function in Python using the NumPy library:

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

Now, you can use this `sigmoid` function to plot the sigmoid activation function:

```
import matplotlib.pyplot as plt

# Generate input values
x = np.linspace(-10, 10, 100)

# Apply sigmoid activation function
y = sigmoid(x)

# Plot the sigmoid function
plt.plot(x, y)
plt.xlabel('Input')
plt.ylabel('Output')
plt.title('Sigmoid Activation Function')
plt.grid(True)
plt.show()
```

With this corrected code, you should be able to plot the sigmoid activation function successfully. If you have any further questions or encounter any issues, feel free to ask!

P You

x = np.linspace(10,-10,100) what does this line mean?

S ChatGPT

The line x = np.linspace(10, -10, 100) creates an array of 100 evenly spaced numbers between 10 and -10. Let's break it down:



- `np.linspace(start, stop, num)`: This NumPy function generates `num` evenly spaced numbers over the closed interval `[start, stop]`. In other words, it divides the interval between `start` and `stop` into `num` equally spaced segments and returns an array containing these points.
- `start`: The starting value of the sequence.
- `stop`: The end value of the sequence.
- `num`: The number of samples to generate.

So, in the line x = np.linspace(10, -10, 100), it generates an array (x) of 100 numbers starting from 10 and ending at -10. The sequence is evenly spaced, meaning the difference between consecutive numbers is constant. In this case, it will generate numbers decreasing from 10 to -10 because the start value (10) is greater than the stop value (-10).



relu

S ChatGPT

ReLU, which stands for Rectified Linear Unit, is a widely used activation function in neural networks, particularly in deep learning architectures. It introduces non-linearity to the network by outputting the input directly if it is positive, and zero otherwise.

Mathematically, the ReLU activation function is defined as:

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

Where:

- *x* is the input to the function.
- f(x) is the output of the function.

In simple terms, if the input x is greater than zero, the ReLU function outputs the input value itself. If the input is negative, the output is zero. This makes ReLU a piecewise linear function.

ReLU has become popular for several reasons:

- 1. **Sparsity**: ReLU's output is zero for all negative inputs, which introduces sparsity into the network. This sparsity can help in reducing overfitting by preventing the network from memorizing noise in the training data.
- 2. **Computational Efficiency**: ReLU is computationally efficient to compute and differentiate. Unlike sigmoid and tanh functions, which involve expensive exponential calculations, ReLU involves only simple thresholding operations.
- 3. Gradient Vanishing: ReLU mitigates the vanishing gradient problem encountered with sigmoid and tanh functions, particularly in deep neural networks. The gradient of ReLU is constant for



positive inputs, ensuring that gradients do not vanish as the network depth increases.

In Python, you can implement the ReLU activation function as follows:

```
import numpy as np

def relu(x):
    return np.maximum(0, x)
```

This function takes an input `x` and applies the ReLU transformation to it using the `np.maximum()` function from NumPy, which returns the element-wise maximum of the inputs.

Here's an example of how to use this function:

```
import matplotlib.pyplot as plt

# Generate input values
x = np.linspace(-10, 10, 100)

# Apply ReLU activation function
y = relu(x)

# Plot the ReLU function
plt.plot(x, y)
plt.xlabel('Input')
plt.ylabel('Output')
plt.title('ReLU Activation Function')
plt.grid(True)
plt.show()
```

This code will plot the ReLU function, showing its characteristic linear behavior for positive inputs and zero output for negative inputs.



softmax

⑤ ChatGPT

The softmax activation function is primarily used in the output layer of neural networks, particularly in multi-class classification problems. It converts raw scores (also known as logits) into probabilities, ensuring that the sum of probabilities across all classes is equal to 1.

Mathematically, the softmax function is defined as:

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

Where:

- x_i is the raw score (logit) for class i.
- e is the base of the natural logarithm, approximately equal to 2.71828.



- The denominator $\sum_{i} e^{x_{i}}$ represents the sum of the exponential scores over all classes.
- The numerator e^{x_i} represents the exponential score for the current class.

The softmax function essentially transforms the logits into a probability distribution over multiple classes, with each class having a probability between 0 and 1. This enables the neural network to output probabilities that can be interpreted as the likelihood of the input belonging to each class.

In Python, you can implement the softmax function as follows:

```
import numpy as np

def softmax(x):
    exp_values = np.exp(x - np.max(x)) # Subtracting the maximum value for numerical stability
    return exp_values / np.sum(exp_values, axis=0)
```

This function takes an array of raw scores `x` as input and applies the softmax transformation to it. It subtracts the maximum value of `x` to improve numerical stability and then calculates the exponential of each score. Finally, it normalizes the exponentials by dividing by their sum, resulting in a probability distribution.

Here's an example of how to use this function:

```
import matplotlib.pyplot as plt

# Generate input values
x = np.linspace(-5, 5, 100)

# Apply softmax activation function
y = softmax(x)

# Plot the softmax function
plt.plot(x, y)
plt.xlabel('Input')
plt.ylabel('Output (Probability)')
plt.title('Softmax Activation Function')
plt.grid(True)
plt.show()
```

This code will plot the softmax function, showing how it transforms the raw scores into a probability distribution over multiple classes.

P You

what does relu do?

⑤ ChatGPT

ReLU, which stands for Rectified Linear Unit, is an activation function commonly used in neural networks. It introduces non-linearity to the network by outputting the input directly if it is positive, and zero otherwise. In simpler terms, ReLU does the following:



1. **Positive Input**: If the input x is greater than zero, ReLU returns the input value itself.

$$f(x) = x$$

2. **Negative Input**: If the input *x* is less than or equal to zero, ReLU returns zero.

$$f(x) = 0$$

In summary, ReLU effectively acts as a thresholding function, activating neurons only if their input is positive.

Here's what ReLU does in practical terms:

- **Sparsity**: ReLU introduces sparsity into the network by setting negative values to zero. This sparsity helps in reducing overfitting by preventing the network from memorizing noise in the training data.
- **Non-linearity**: Although ReLU is a simple function, it introduces non-linearity to the network, allowing it to learn complex patterns and relationships in the data.
- **Efficiency**: ReLU is computationally efficient to compute and differentiate compared to other activation functions like sigmoid and tanh. It involves only simple thresholding operations, making it faster to train neural networks.

Overall, ReLU is a popular choice in neural networks due to its simplicity, efficiency, and effectiveness in training deep learning models. It is often used in hidden layers of neural networks and has been instrumental in the success of deep learning in various applications.

P You

So, in the line x = np.linspace(10, -10, 100), it generates an array (x) of 100 numbers starting from 10 and ending at -10. The sequence is evenly spaced, meaning the difference between consecutive numbers is constant. In this case, it will generate numbers decreasing from 10 to -10 because the start value (10) is greater than the stop value (-10).

Explian me how the numbers are generated and then what happens in this:y = (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))

S ChatGPT

Certainly! Let's break down each part of the code:

1. Generating the array `x`:

```
python
x = np.linspace(10, -10, 100)
```

Here, `np.linspace()` is used to generate an array `x` containing 100 evenly spaced numbers starting from 10 and ending at -10. Since the start value (10) is greater than the stop value (-10), the



sequence will be decreasing from 10 to -10. The `linspace()` function evenly divides the interval between 10 and -10 into 100 points, so the difference between consecutive numbers is constant.

2. Calculating the tanh function:

```
python

y = (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))
```

This line calculates the hyperbolic tangent (tanh) function for each element in the array x. The formula for the tanh function is:

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

Here's how the calculation works:

- o `np.exp(x)` calculates the exponential of each element in the array `x`.
- o `np.exp(-x)` calculates the exponential of the negative of each element in the array `x`.
- \circ `(np.exp(x) np.exp(-x))` computes the difference between the exponentials of `x` and `-x`.
- \circ `(np.exp(x) + np.exp(-x))` computes the sum of the exponentials of `x` and `-x`.
- Finally, (np.exp(x) np.exp(-x)) / (np.exp(x) + np.exp(-x)) divides the difference by the sum, giving us the tanh function for each element in the array x.

So, in summary, the code generates an array \mathbf{x} containing 100 evenly spaced numbers from 10 to -10, and then calculates the tanh function for each element in \mathbf{x} using the formula provided. The resulting array \mathbf{y} will contain the values of the tanh function corresponding to each value of \mathbf{x} .

P You

what i mean is are the 100 numbers randomly generated and fed to y at once are one by one? Or its completely something else?

⑤ ChatGPT

I see, let me clarify the process.

In the line x = np.linspace(10, -10, 100), the np.linspace() function generates 100 evenly spaced numbers starting from 10 and ending at -10. These numbers are not randomly generated; they are evenly distributed along the interval from 10 to -10. Each of these numbers represents a point on the x-axis.

Then, in the line y = (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x)), the np.exp() function calculates the exponential of each element in the array x. This means that the exponential function is applied to every number generated by np.linspace() individually.

So, the array $\mathbf{\hat{x}}$ containing 100 numbers is not fed to $\mathbf{\hat{y}}$ all at once. Instead, each number in $\mathbf{\hat{x}}$ is processed one by one to calculate the corresponding value in $\mathbf{\hat{y}}$. The calculation is done element-wise,



meaning that each number in `x` is independently used to calculate the corresponding value in `y`.

In summary, the numbers in the array \mathbf{x} are generated once, and then each number is used individually to calculate the corresponding value in the array \mathbf{y} . This process ensures that the values of the hyperbolic tangent (tanh) function are computed for each input value in \mathbf{x} according to the specified formula.

ChatGPT can make mistakes. Consider checking important information.