Title: Write a python program to plot a few activation functions that are being used in neural network

Outcome:

The Python program demonstrate and plots some commonly used activation functions used in neural networks. It will plot the Sigmoid, Tanh, ReLU and Softmax activation functions for visualization purposes. This program will generate a plot showing the activation functions of Sigmoid, Tanh, ReLU and Softmax over a range of input values. one can observe the behavior and characteristics of each activation function from the plot.

Software and Hardware Requirements:

- Python 3.0

- NumPy library

- Matplotlib library

- Jupyter Notebook

Theory: An activation function in a neural network is a mathematical function applied to the output of a neuron or a layer of neurons. It determines the output or activation level of a neuron based on the weighted sum of its inputs. The purpose of an activation function is to introduce non-linear transformations to the network's computations. Without activation functions, the network would be limited to performing only linear transformations. Commonly used activation functions include the sigmoid, tanh, ReLU (Rectified Linear Unit), Leaky ReLU, and softmax. Each activation function has its own properties and characteristics, making it suitable for different types of problems and architectures.

Sigmoid Function:

The sigmoid activation function, also known as the logistic function, is a mathematical function commonly used in machine learning. It transforms an input value into a range between 0 and 1. The sigmoid function has an S-shaped curve and is defined as f(x) = 1 / (1 + e^(-x)), where f(x) represents the sigmoid output for a given input x and e is the mathematical constant approximately equal to 2.71828. The sigmoid activation function is often used in binary classification tasks to produce probabilities that an input belongs to a certain class. It is differentiable and its output range makes it suitable for modeling nonlinear relationships and mapping inputs to probabilities.

Tanh Function:

The hyperbolic tangent (tanh) activation function is a mathematical function commonly used in neural networks. It is an extension of the sigmoid function and also maps the input to a range between -1 and 1. The tanh activation function is symmetric around the origin, providing both positive and negative outputs. It is useful for capturing nonlinear relationships and is often used in recurrent neural networks (RNNs) and hidden layers of feedforward neural networks.

Relu Function:

The Rectified Linear Unit (ReLU) activation function is a commonly used mathematical function in neural networks. It applies a simple thresholding operation to the input values, transforming negative values to zero and leaving positive values unchanged. Mathematically, the ReLU function can be defined as $f(x) = \max(0, x)$, where $f(x)$ represents the ReLU output for a given input x. The ReLU activation function is known for its simplicity and computational efficiency. ReLU has become widely popular and is often used in deep learning architectures due to its ability to alleviate the vanishing gradient problem and promote faster convergence during training.

Softmax:

The softmax function is a mathematical function used in machine learning and neural networks. It takes a vector of real numbers as input and outputs a probability distribution over multiple classis. The softmax function normalizes the input values and transforms them into probabilities that sum up to 1. It is commonly used as the activation function in the output layer of a neural network for multi-class classification tasks. The softmax function exponentiates each input value, divides it by the sum of all exponentiated values, and produces a probability value for each class

Conclusion:

Activation functions are crucial components of neural networks that enable them to learn complex mappings from input data to output predictions. Understanding the properties and behaviors of different activation functions is essential for designing effective neural network architectures and achieving optimal performance in various machine learning tasks.

Title: Generate ANDNOT function using McCulloch-Pitts neural network by a python program.

Outcome: The Python program demonstrate The McCulloch-Pitts neuron model which is a simplified mathematical model of a neuron. It's primarily used for binary classification tasks. The ANDNOT function is a logical operation that returns true if the first input is true and the second input is false.

Software and Hardware Requirements:

- Python 3.0
- NumPy library
-Text editor or Integrated Development Environment (IDE)

Theory:

The McCulloch-Pitts neural network model is a simplified mathematical abstraction of a biological neuron. It was introduced by Warren McCulloch and Walter Pitts in 1943. While it is not as sophisticated as modern neural network models, it provides a foundation for understanding neural computation. The ANDNOT function is a logical operation that takes two binary inputs and returns true (1) if the first input is true (1) and the second input is false (0). Otherwise, it returns false (0). This function can be implemented using a McCulloch-Pitts neuron by appropriately setting the weights and threshold.

The ANDNOT Function: The ANDNOT function is true (1) if the first input is 1 and the second input is 0; otherwise, it returns 0.

McCulloch-Pitts Neuron Model: In this model, the neuron output y is computed as:

1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq \text{threshold} \\
0 & \text{otherwise}
\end{cases} \]

where \(x_i \) are inputs, \(w_i \) are weights, and the threshold is a constant.

Implementing the ANDNOT Function:

McCulloch-Pitts neuron with specific weights and a threshold to implement the ANDNOT logic.

- mcculloch_pitts_neuron computes the output of a McCulloch-Pitts neuron given inputs, weights, and a threshold.
- andnot_function implements the ANDNOT logic by defining specific weights ([1, -1]) and a threshold (0.5).
- The andnot_function uses this neuron to compute the ANDNOT of two binary inputs (x1 and x2). The threshold and weights are set such that:

- If x1 = 1 and x2 = 0, the weighted sum (1 * 1 + (-1) * 0 = 1) exceeds the threshold (0.5), resulting in an output of 1.
- For all other combinations (x1 = 0 or x2 = 1 or both), the output will be 0.

Conclusion:

In conclusion, the ANDNOT function can be effectively implemented using the McCulloch-Pitts neural network model. Through appropriate selection of weights and threshold, the McCulloch-Pitts neuron can mimic the logic of ANDNOT operation. The process involves assigning weights to inputs such that the neuron activates when the first input is 1 and the second input is 0, while remaining inactive otherwise. This implementation showcases the capability of simple neural network models, like the McCulloch-Pitts model, to perform basic logical operations. However, it's important to note that the McCulloch-Pitts model is a simplified abstraction and lacks the sophistication of modern neural network architectures.

Title: Write a suitable example to demonstrate the perceptron learning law with its decision regions using python. Give the output in graphical form.

Outcome:

The Python program demonstrates the implementation of the perceptron learning law with its decision regions.

Software and Hardware Requirements:

- Python 3.0

- NumPy library

- Matplotlib library

- Sklearn library

Theory:

Perceptron learning law:

The perceptron learning law is an iterative algorithm used to train a single-layer perceptron for binary classification tasks. It works by adjusting the weights and bias of the perceptron to minimize the error between the predicted class labels and the actual class labels.

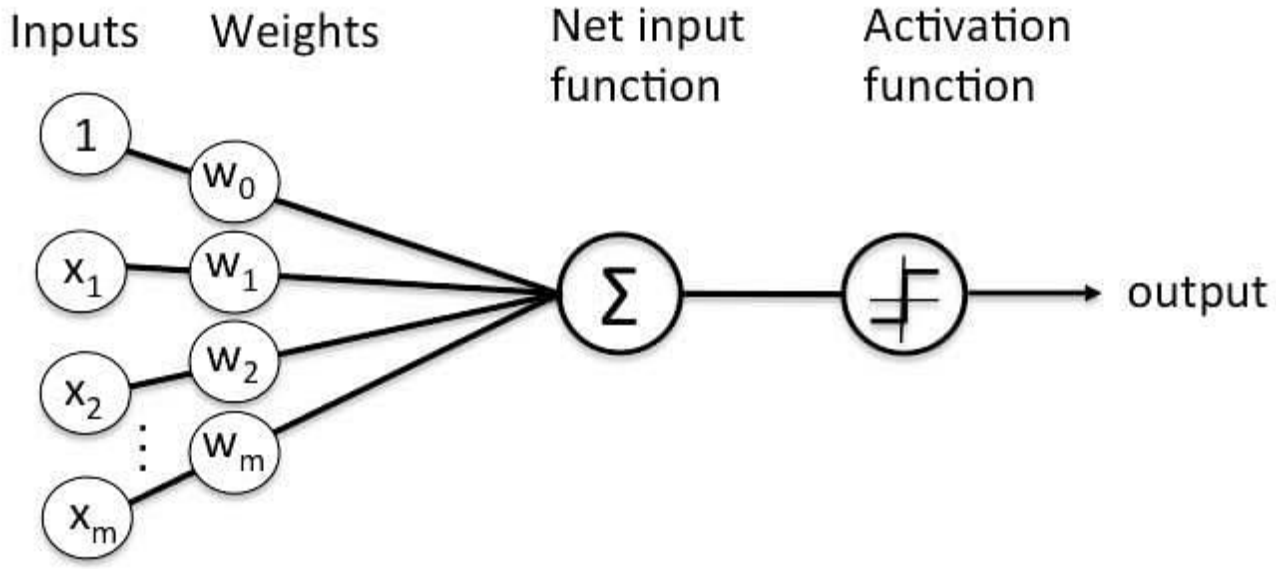
Core Concepts:

Perceptron: A simple artificial neural network unit with one or more input features, weights, a bias term, and an activation function (usually the step function in this case).

Weights: Values associated with each input feature that determine their influence on the output.

Bias: A constant term that shifts the decision boundary learned by the perceptron.

Activation Function: A function applied to the weighted sum of inputs and bias to produce the output class label. The step function outputs 1 for a non-negative value and 0 otherwise, creating a binary classification.



Algorithm:

1. **Initialization:**
    - Initialize weights and bias to zeros or small random values.
    - Define learning rate and number of epochs.
2. **Define Perceptron Function:**
    - Compute the weighted sum of inputs and apply an activation function to obtain the output.
3. **Training:**
    - Iterate through the dataset for a fixed number of epochs.
    - Update weights and bias based on the perceptron learning rule.
4. **Decision Boundary:**
    - Generate a mesh grid of points covering the feature space.
    - Classify each point using the trained perceptron to plot the decision boundary.
5. **Output:**
    - Visualize decision regions and decision boundary to analyse perceptron's performance.

Conclusion:

In this assignment, we studied to demonstrate the perceptron learning law with its decision regions using python.

Title: Write a python program for Bidirectional Associative Memory with two pairs of vectors.

Outcome:

The Python program demonstrates the Bidirectional Associative Memory with two pairs of vectors.

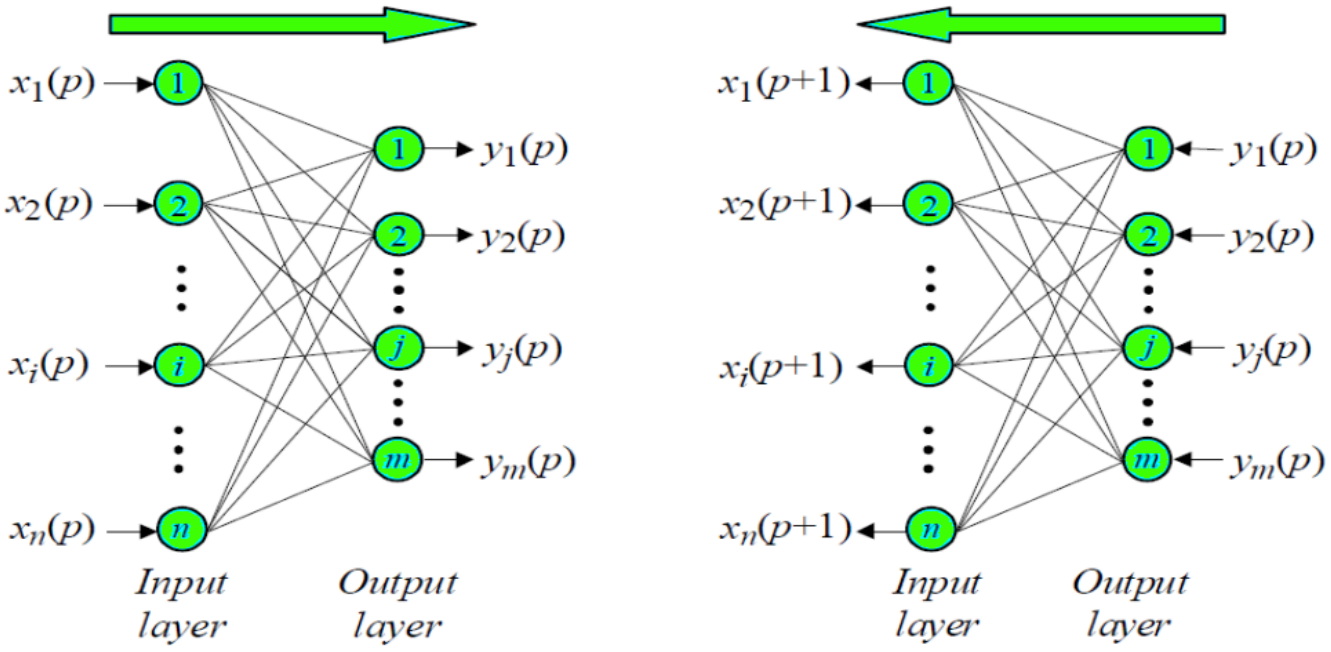
Software and Hardware Requirements:

- Python 3.0
- NumPy library
- Matplotlib library

Theory:

Bidirectional Associative Memory (BAM) is a supervised learning model in Artificial Neural Network. This is hetero-associative memory, for an input pattern, it returns another pattern which is potentially of a different size. This phenomenon is very similar to the human brain. Human memory is necessarily associative. It uses a chain of mental associations to recover a lost memory like associations of faces with names, in exam questions with answers, etc. In such memory associations for one type of object with another, a Recurrent Neural Network (RNN) is needed to receive a pattern of one set of neurons as an input and generate a related, but different, output pattern of another set of neurons.

The main objective to introduce such a network model is to store hetero-associative pattern pairs. This is used to retrieve a pattern given a noisy or incomplete pattern. BAM Architecture: When BAM accepts an input of n-dimensional vector X from set A then the model recalls m-dimensional vector Y from set B. Similarly, when Y is treated as input, the BAM recalls X.



(a) Forward direction. (b) Backward direction.

Conclusion: The above python program demonstrate application for Bidirectional Associative Memory with two pairs of vectors.

Title: Implement Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation.

Outcome:

The Python program demonstrates the Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation.

Software and Hardware Requirements:

- Python 3.0

- NumPy library

- Matplotlib library

Theory:

To implement the training process of an Artificial Neural Network (ANN) in Python, it is required to cover both forward propagation and backpropagation. create a class for the neural network and implement methods to perform these steps using a simple feedforward architecture with a single hidden layer and sigmoid activation functions. We will also use gradient descent to update the network parameters during backpropagation.

In this implementation:

We define a Neural Network class representing a simple feedforward neural network with one hidden layer.

The constructor (__init__) initializes the network's weights (W1, W2) and biases (b1, b2) randomly using a normal distribution. learning_rate is the hyperparameter controlling the step size of weight updates during training.

The sigmoid method implements the sigmoid activation function.

The sigmoid_derivative method computes the derivative of the sigmoid activation function, which is used in backpropagation.

The forward_propagation method performs forward propagation through the network:

It computes the weighted sum and applies the sigmoid activation for both the hidden and output layers.

The backward_propagation method performs backpropagation to update weights and biases based on the computed errors (delta2 and delta1).

The train method trains the neural network using the specified number of epochs:
It iteratively performs forward propagation, backpropagation, and updates the weights and biases.
It prints the loss (mean squared error) at regular intervals during training.
In the example usage block:
We create an instance of Neural Network with specified input size, hidden layer size, output size, and learning rate.
We generate random training data (XOR example) and use it to train the neural network.
Finally, we test the trained network by making predictions on the training data (X) using forward propagation.

Conclusion:

The above Implementation demonstrates Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation.

Title: Write a python program to illustrate ART neural network.

Outcome:

The Python program demonstrates the illustrate ART neural network. these models leverage the capabilities of neural networks to generate or manipulate artistic content such as images, music, text, or other forms of creative output.

Software and Hardware Requirements:

- Python 3.0
- NumPy library
- Matplotlib library

Theory: ART is a class of neural network models that was developed by Stephen Grossberg and Gail Carpenter in the 1980s. The ART model is unique in its ability to perform unsupervised learning and pattern recognition while exhibiting properties of stability and adaptability.

Overview of the ART model and its key components:

Components of ART Model:

Input Layer (F1 Layer): This layer receives input patterns, which are typically binary or continuous values. The F1 layer neurons represent the input feature space.

Comparison Layer (F2 Layer): Neurons in the F2 layer perform competitive matching between the input pattern and prototype memories (stored patterns) in the network. The F2 neurons use a matching rule to determine similarity between the input and stored memories.

Recognition (Reset) Mechanism: The ART model uses a recognition mechanism to identify the closest prototype memory that matches the input pattern. If a match is found above a certain threshold (determined by vigilance parameter), the corresponding prototype memory is activated.

Learning Rule: ART networks use a learning rule to update the prototype memories based on the input patterns. When a new input is presented and no match is found (below the vigilance threshold), a new prototype memory is created in the network.

Key Concepts:

Vigilance Parameter: This parameter controls the sensitivity of the ART model to input patterns. Higher vigilance values lead to stricter matching criteria, resulting in fewer matches and more distinct prototype memories.

Resonance: Resonance occurs when an input pattern sufficiently matches a stored prototype memory, causing activation of the corresponding neuron in the F2 layer.

Conclusion:

The above Implementation demonstrates Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation.

Title: Write a python program to design a Hopfield Network which stores 4 vectors.

Outcome:

The Python program will display the stored patterns and the retrieval results for the specified input patterns using the trained Hopfield Network. Hopfield Networks are useful for associative memory tasks and can recall stored patterns even when presented with noisy or incomplete inputs. You can modify the input patterns and observe how the network retrieves or recognizes similar patterns based on the stored memories.

Software and Hardware Requirements:

- Python 3.0
- NumPy library
- Jupyter Notebook

Conclusion: We have designed a Hopfield network that stores 4 vectors, the network is able to recall the stored vectors even when they are perturbed.

Title: MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

Outcome:

The aim of this assignment is to Implement MNIST handwritten character detection using PyTorch, Keras with TensorFlow backend, or TensorFlow would yield similar outcomes in terms of accuracy and performance in terms of use, familiarity, and specific project requirements.

Hardware Requirements

Software Requirements

- Python 3.0

- NumPy library

- Matplotlib library

Theory:

The MNIST dataset serves as a foundational benchmark for evaluating various classification algorithms, particularly in the domain of handwritten character recognition. This dataset comprises 28x28 grayscale images of digits (0-9), making it ideal for introductory tasks due to its manageable size and simplicity. Deep learning frameworks such as PyTorch, Keras with TensorFlow backend, and TensorFlow offer robust tools for building and training neural networks on the MNIST dataset, each with its unique features and advantages.

These frameworks provide a range of functionalities tailored to different user needs and expertise levels. PyTorch, favored by researchers and experienced developers, boasts a dynamic computation graph and intuitive debugging capabilities, enabling flexible model development and experimentation. Keras, built on top of TensorFlow, offers a user-friendly API suitable for beginners and rapid prototyping, while TensorFlow provides scalability and deployment capabilities, making it ideal for large-scale production deployments across various platforms.

When implementing MNIST handwritten character detection, the choice of framework often hinges on factors such as ease of use, familiarity, and specific project requirements. Despite their differences in syntax and implementation details, all three frameworks enable users to achieve high accuracy by employing convolutional neural networks (CNNs) and training methodologies such as data augmentation, regularization, and hyperparameter tuning. By understanding the theoretical foundations and practical implementation aspects of these frameworks, developers can effectively leverage their strengths to tackle MNIST classification tasks with confidence and efficiency.

Algorithm:

1. Data Preparation:

    - Load the MNIST dataset, which typically consists of training and testing sets of 28x28 grayscale images of handwritten digits (0-9) along with their corresponding labels.
    - Normalize the pixel values of the images to a range between 0 and 1 to aid convergence during training.
    - Convert the labels into one-hot encoded vectors for categorical classification.

2. Model Definition:

    - Define a neural network architecture suitable for image classification, commonly based on convolutional neural networks (CNNs) due to their effectiveness in image tasks.
    - Specify the layers of the model, including convolutional layers, pooling layers, and fully connected layers.
    - Choose appropriate activation functions such as ReLU for hidden layers and softmax for the output layer to obtain class probabilities.

3. Training:

    - Initialize the model parameters and select a loss function, such as cross-entropy loss, to measure the disparity between predicted and actual labels.
    - Choose an optimizer, such as stochastic gradient descent (SGD) or Adam, to update the model parameters based on the gradient of the loss function.
    - Iterate over the training set in mini-batches, forward propagating the input data through the network, computing the loss, and backward propagating the gradients to update the model parameters.
    - Validate the model performance on a separate validation set to monitor for overfitting and adjust hyperparameters accordingly.
    - Repeat the training process until convergence or a predefined number of epochs.

4. Evaluation:

    - Evaluate the trained model on the test set to assess its generalization performance using metrics such as accuracy, precision, recall, and F1-score.
    - Visualize the model predictions alongside the ground truth labels to gain insights into its strengths and weaknesses.
    - Fine-tune the model architecture and hyperparameters as needed based on the evaluation results to improve performance.

5. Deployment:

- Save the trained model parameters for future use or deployment in production environments.
- Implement inference functionality to classify new handwritten digit images using the trained model.
- Integrate the model into applications or services for real-world usage, ensuring scalability, efficiency, and reliability.

Conclusion:

MNIST handwritten character detection using PyTorch, Keras, and TensorFlow showcases the effectiveness of these frameworks in image classification. Despite differences, they all offer robust tools for model development, training, and deployment. By understanding their nuances, developers can harness their strengths to achieve accurate results in recognizing handwritten digits.

Title:

Train a neural network with Tensorflow / pytorch and evaluation of logistics regression using tensorflow.

Outcome:

Conducting this experiment provides valuable insights into the strengths, weaknesses, and trade-offs associated with different machine learning models, facilitating informed decision-making in model selection and deployment.

Hardware Requirements

Software Requirements

- Python 3.0

- NumPy library
- Matplotlib library

Theory:

Neural networks, built upon the structure of biological neurons, comprise layers of interconnected nodes that process input data through weighted connections. These connections enable the network to learn complex patterns and relationships in the data. TensorFlow and PyTorch are prominent deep learning frameworks, offering tools and libraries for constructing, training, and deploying neural network models efficiently. Logistic regression, a fundamental statistical technique for binary classification, models the probability of a binary outcome using a logistic function. Despite its simplicity, logistic regression can provide interpretable results and serves as a baseline model for comparison.

During model evaluation, performance metrics such as accuracy, precision, recall, and F1-score are computed using a separate test dataset to assess the model's effectiveness in making predictions. Model selection involves comparing the performance of different algorithms or architectures and selecting the most suitable one based on predefined criteria such as accuracy, computational resources, and interpretability.

Optimization techniques, including hyperparameter tuning and architecture adjustments, aim to improve the model's performance further. These methods involve exploring the hyperparameter space using techniques such as grid search, random search, or Bayesian optimization to find optimal configurations.

The interpretability-complexity trade-off is an essential consideration in model selection. Simple models like logistic regression are easier to interpret but may lack the capacity to capture complex patterns in the data. In contrast, neural networks, particularly deep architectures, offer greater complexity and

predictive power but may be more challenging to interpret due to their black-box nature.

By understanding these principles and considerations, researchers and practitioners can design and conduct experiments effectively, leading to the development of robust and accurate machine learning models.

Conclusion:

In this project, we explored the implementation of two different machine learning techniques: training a neural network using TensorFlow or PyTorch and evaluating logistic regression using TensorFlow.