Assignment - 10
Neural Networks

HARSH SIDDHAPURA

1230169813

03/02/2024

The Perceptron

A **perceptron** stands at the crossroads of machine learning and artificial neural networks. It's the elementary building block, akin to a digital neuron, that plays a pivotal role in binary classification tasks. Imagine it as a decision-maker: given input features, it determines whether an instance belongs to one specific class or another. Simplicity is its hallmark, yet its impact reverberates through more complex neural network architectures.

Mathematical representation, y = sign [wdxd + wd-1.xd-1 + ...] = sign[w.x]

The perceptron comprises three essential components:

- **Input Features:** These are the raw data points, represented as a vector.
- **Weights:** Each input feature is assigned a weight, reflecting its importance.
- **Threshold:** A predefined threshold value. The perceptron computes a weighted sum of the input features, compares it to the threshold, and makes a binary decision. If the sum exceeds the threshold, it outputs a positive class; otherwise, it signals a negative class. This linear decision boundary neatly separates the input space.

Activation Functions

Sign() function:

- The sign() function is straightforward: it maps input values to either 1 or -1. If the input is positive, it yields 1; if negative, it gives -1. It's a step function, abruptly transitioning between these two values.
- Not Differentiable: The critical limitation of the sign function lies in its lack of differentiability. Its derivative is zero almost everywhere, except at the origin (where it's undefined).
- Backpropagation Challenge: During neural network training, gradients are essential for weight updates. Since the sign function's derivative is zero almost everywhere, the weights won't adjust effectively. Consequently, learning becomes impossible [1].

Sigmoid() function:

- The sigmoid() function, also known as the logistic function, smoothly maps input values to the range (0, 1).
- Differentiable: Unlike the sign function, the sigmoid function is differentiable everywhere. Its derivative is never zero, although it does become small for extreme inputs.
- Continuous Transition: The sigmoid function provides a gradual transition from 0 to 1, making it suitable for modeling probabilities or confidence scores.

The sigmoid activation allows neural networks to work with non-linearly separable data. By stacking multiple layers of sigmoid-activated neurons, neural networks can learn intricate decision boundaries, enabling them to tackle complex tasks [2].

The Multi-Layer Perceptron (MLP)

Sol-1:

A **Perceptron** serves as the fundamental building block in artificial neural networks. Its simplicity lies in having just two layers: an input layer and an output layer. When faced with binary classification tasks, it determines whether input features belong to one specific class or another. Picture it as a digital decision-maker. However, perceptrons have limitations—they can only handle linearly separable patterns and lack the ability to learn complex relationships.

A **Multi-Layer Perceptron (MLP)** extends beyond the perceptron's boundaries. It introduces hidden layers - one or more layers sandwiched between the input and output layers. These hidden layers contain neurons that compute weighted sums and apply activation functions (like sigmoid or ReLU). The magic lies in non-linearity: MLPs can model intricate decision boundaries. They thrive on non-linearly separable data, capturing patterns that defy simple linear separation.

While perceptrons are elementary, MLPs empower neural networks to tackle real-world complexities. By stacking hidden layers, they learn and adapt. Training an MLP involves backpropagation and gradient descent, making it more challenging than perceptrons but infinitely more powerful.

Sol-2:

Effect of Adding Hidden Layers:

1. Shape of Decision Boundaries: As we add hidden layers, the shape of decision boundaries becomes more intricate. Shallow networks (few hidden layers) tend to produce simple linear or piecewise linear boundaries. Deeper architectures (more hidden layers) allow for curved, non-linear decision boundaries. These complex boundaries adapt better to intricate data distributions.

2. Weight Parameters: More hidden layers mean an increased number of weight parameters. Each layer introduces additional weights that need to be learned during training. Longer training times are required due to the higher dimensionality of weight space.

Effect of Adding Neurons Within Hidden Layers:

1. Shape of Decision Boundaries: More neurons within a layer enhance the expressive power of the MLP. Abundant neurons allow the network to represent a larger set of non-linear transformations. Decision boundaries become more flexible and capable of capturing intricate patterns.

2. Weight Parameters: Increasing the number of neurons directly impacts the total weight parameters. Each neuron contributes to the weight space, affecting the model's capacity. However, too many neurons can lead to overfitting, where the model fits noise in the data.

References

[1] Detailed insights into sigmoid vs. tanh: Sigmoid vs. Tanh Activation Functions
https://www.baeldung.com/cs/sigmoid-vs-tanh-functions

[2] Keras documentation on activation functions: Keras Layer Activation Functions
https://keras.io/api/layers/activations/