Assignment 2 - Perceptron

Neural Networks

Due date: 21-10-2024

1 Perceptron

A perceptron is a foundational building block in the realm of machine learning and artificial intelligence, serving as a binary linear classifier. It is the simplest type of artificial neural network and lays the groundwork for multi-layer perceptrons and deep learning.

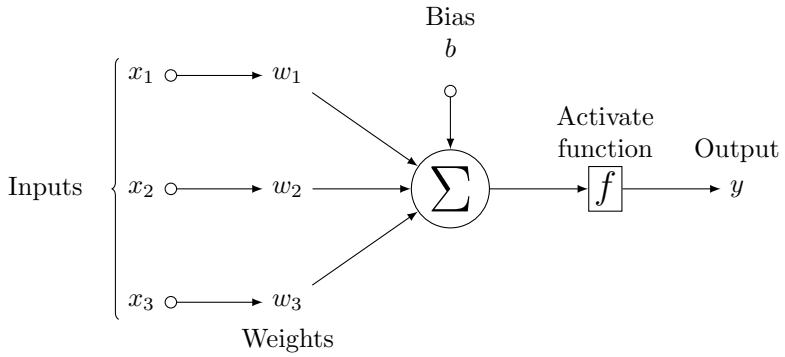

Figure 1: Neuron structure

The perceptron consists of a single layer with multiple input features, each assigned a specific weight. These weights are parameters that the algorithm learns through training. The output is binary, determined by whether the weighted sum of the inputs plus a bias is less than or equal to zero, or greater than zero. Mathematically, this can be represented as:

$$y = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{1}$$

where

- w is the weight vector,
- x is the input vector,

- b is the bias term,
- y is the binary output after applying the activation function to the weighted sum of the inputs.

Forward Propagation is the process where the input data is passed through the perceptron to get the output. It is an essential step in training the neural network.

2 Forward Propagation in a Perceptron

Forward propagation in a perceptron involves these steps:

1. Compute the weighted sum of the inputs and the bias. Initially, the weights and bias are often randomly initialized.

$$z = w \cdot x + b$$

   For multi-class classification, the weights are organized in a matrix W where each column corresponds to a class, and the weighted sum for each class is computed as:

$$z_i = W_i \cdot x + b_i$$

   where z_i is the weighted sum for class i.

2. Apply an activation function to transform the computed weighted sums into class probabilities. For binary classification, a step function or sigmoid can be used. For multi-class classification, the softmax function is commonly applied to produce a probability distribution over the classes:

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

   The predicted class corresponds to the index of the highest probability:

$$\hat{y} = \arg\max(y_i)$$

3. Evaluate the error by comparing the perceptron's output with the actual target value. In the multi-class case, cross-entropy loss is commonly used to compute the error:

$$CrossEntropy(\hat{y}, y) = -\sum_i \mathrm{y}_i \log(\hat{y}_i)$$

3 Training the Perceptron with Gradient Descent

After calculating the error, the next step is to update the weights and bias to minimize this error. This is done using a method called gradient descent. The gradient descent algorithm adjusts the weights and bias in the direction that minimally decreases the overall error. The updates are performed iteratively until the error converges to a minimum value.

The weights and bias are updated using the gradients of the cross-entropy loss with respect to the weights and bias. For multi-class classification, this is done as follows:

$$W = W + \mu \times (Target - y) \times x^T \tag{2}$$

$$b = b + \mu \times (Target - y) \tag{3}$$

where

- μ is the learning rate, a hyperparameter that determines the step size at each iteration while moving towards a minimum of the cost function,
- $Target - y$ is the calculated gradient of the cross-entropy loss between the predicted output and the actual target value,
- w and b are updated to minimize the error.

The perceptron, though simple, lays the foundation for understanding more complex neural network architectures. It introduces key concepts like weights, bias, activation functions, and the process of training a model using an algorithm like gradient descent. Understanding the perceptron is a stepping stone to exploring multi-layer neural networks and deep learning.

4 Homework - 15 points

In this exercise, you are tasked with implementing both the forward and backward propagation processes for a neural network with 784 inputs and 10 outputs using NumPy. This network can be thought of as consisting of 10 perceptrons, each responsible for predicting one of the 10 output classes.

4.1 Problem Statement

Given an input matrix `X` of shape $(m, 784)$, where m is the batch size and 784 is the number of features (input neurons), a weight matrix `W` of shape $(784, 10)$, and a bias matrix `b` of shape $(10,)$, compute the output of the network for each example in the batch, calculate the error, and update the weights and biases accordingly.

Download the MNIST dataset, load the images, and propagate them through your network. Record the initial prediction accuracy before training and the prediction accuracy after training for a specified number of epochs.

1. Load the MNIST dataset

```
import numpy as np
from torchvision.datasets import MNIST


def download_mnist(is_train: bool):
    dataset = MNIST(root='./data',
                    transform=lambda x: np.array(x).flatten(),
                    download=True,
                    train=is_train)

    mnist_data = []
    mnist_labels = []
    for image, label in dataset:
        mnist_data.append(image)
        mnist_labels.append(label)

    return mnist_data, mnist_labels


train_X, train_Y = download_mnist(True)
test_X, test_Y = download_mnist(False)
```

2. Normalize the data and convert the labels to one-hot-encoding.
3. Train the perceptron for 50-500 epochs.
    - For each epoch, split the training data and training labels into batches of 100 elements.
    - For each batch, compute the output of the network using the softmax function. **(3 points)**
    - Implement the gradient descent algorithm to update the weights and biases. **(7 points)**
    - Have an efficient implementation. **(2 points)**
    - Achieve at least 90% accuracy on the testing data. **(3 points)**