Assignment 2 - Perceptron

Advanced Topics in Neural Networks

10 October 2023

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.

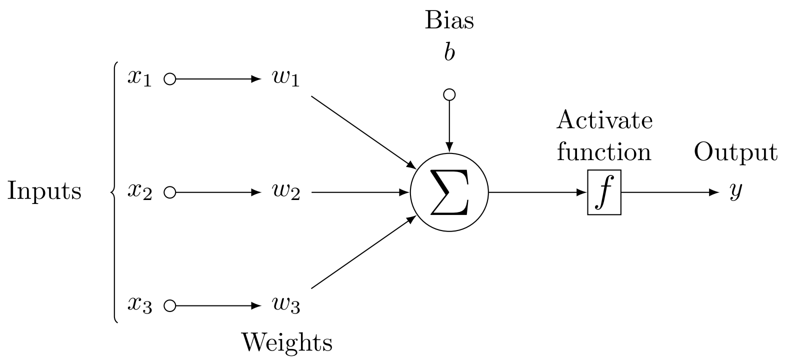

Figure 1: Neuron structure

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:

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

where

- $\mathbf{w}$ is the weight vector,
- $\mathbf{x}$ 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.
$$z = \mathbf{w} \cdot \mathbf{x} + b$$
2. Apply an activation function to transform the computed weighted sum into the binary output. The Heaviside step function is a common choice for binary classification tasks (the sigmoid can also be used).
$$y = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases}$$
3. Evaluate the error by comparing the perceptron's output with the actual target value.
$$\text{Error} = \text{Target} - y$$

The perceptron is a linear model; hence it can only model linearly separable functions. For more complex, non-linearly separable problems, multi-layer perceptrons or other types of neural networks are typically used.

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 following equations:

$$\mathbf{w} := \mathbf{w} + \mu \times \text{Error} \times \mathbf{x} \tag{2}$$

$$b := b + \mu \times \text{Error} \tag{3}$$

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,
- $\text{Error} = \text{Target} - y$ is the calculated error between the predicted output and the actual target value,
- $\mathbf{w}$ 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 Exercise

Due date: End-of-Day Monday, 16.10.2023

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 PyTorch. This network can be thought of as consisting of 10 perceptrons, each responsible for predicting one of the 10 output classes. Implement these processes manually, using only basic PyTorch operations. Do **not** use built-in layers like `nn.Linear` or similar high-level APIs.

4.1 Problem Statement

Given an input tensor `X` of shape $(m, 784)$, where m is the number of examples and 784 is the number of features (input neurons), a weight tensor `W` of shape $(784, 10)$, and a bias tensor `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.

4.2 Constraints

- Use only basic PyTorch operations like matrix multiplication, element-wise addition, and indexing. Don't use APIs such as `torch.nn` or `torch.functional`.
- Assume the activation function to be the sigmoid function, defined as

$$y = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- The bias tensor `b` should be added to the weighted sum before the activation function is applied.
- Implement the gradient descent algorithm to update the weights and biases.

4.3 Expected Outcome

Write a Python function named `train_perceptron` that takes `X`, `W`, `b`, `y_true` (the true labels), and `mu` (the learning rate) as inputs and returns the updated weights and biases after applying both forward and backward propagation steps.

Function Signature:

```
def train_perceptron(X: Tensor, W: Tensor, b: Tensor, y_true: Tensor, mu: float):
    # Return the updated W and b
    pass  # Your implementation here
```

Input:

- `X`: A 2D PyTorch tensor of shape $(m, 784)$ containing the input features.
- `W`: A 2D PyTorch tensor of shape $(784, 10)$ containing the initial weights for the 10 perceptrons.
- `b`: A 1D PyTorch tensor of shape $(10,)$ containing the initial biases for the 10 perceptrons.
- `y_true`: A 2D PyTorch tensor of shape $(m, 10)$ containing the true labels for each of the m examples.
- `mu`: A float representing the learning rate.

Output:

- A tuple containing two elements: the updated weight tensor `W` of shape $(784, 10)$ and the updated bias tensor `b` of shape $(10,)$ after training.

Evaluation: Your implementation will be considered correct if the weights and biases are updated correctly according to the forward and backward propagation steps described earlier in this document, adhering to the specified constraints.

Extra 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.