Mohammed V Universty
National higher school of computer science and system analysis

Learning Algorithms implementation in C++

Author:
Oussama BARKOUKI

artificial intelligence engineering

college year
2020-2021

Contents

List of Figures

1 Introduction

A basic implementation of different learning algorithms, and a brief comparison between them. The following sections highlight three algorithms PLA, Pocket, and Adaline with their respective hypothesis visualization as well as the evolution of the loss function over time.

2 Learning algorithms

2.1 Data without Noise

For each of those algorithms, the implementation is a bit different as well as their use cases. For example, the PLA cannot handle noises on the data We'll go into details later about why.

2.1.1 PLA

A data without noise represents clean data that can easily be separated by a line. The algorithm can easily converge and hit the condition of the loss being 0.

The following figures highlight the hypothesis and the evolution of loss function over time.

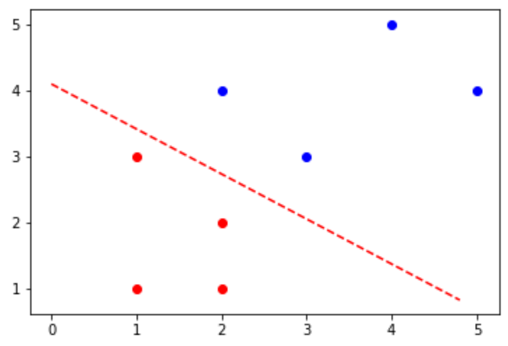

Figure 1: Perceptron Hypothesis

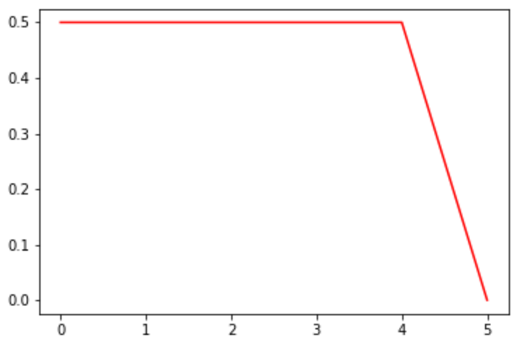

Figure 2: PLA loss evolution

2.2 Data with Noise

2.2.1 Pocket

The pocket algorithm works in a way that restricts the model into a specific number of iterations each, while for each iteration only keeping the weights that resulted in the minimal loss. The algorithm keeps changing the weights but only storing those that were able to give a minimal loss.

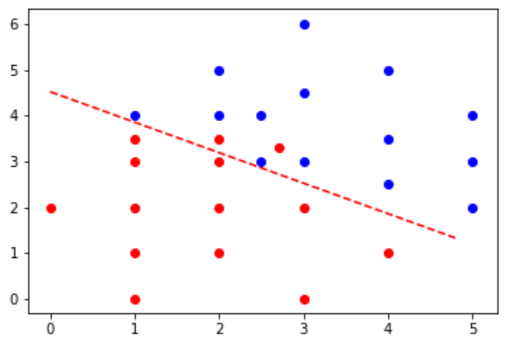

Figure 3: Pocket Hypothesis

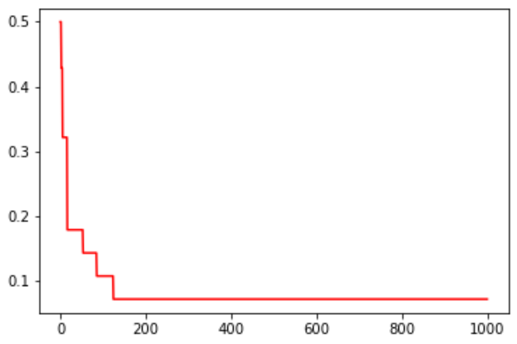

Figure 4: Pocket loss evolution

2.2.2 Adaline

Adaline uses a different direction algorithm it uses gradient descent, the same for it's loss function that uses the mean squared error. This model is a bit different from the other ones as it has two different hypotheses one for training and that is used for the prediction that applies the sign activation function.

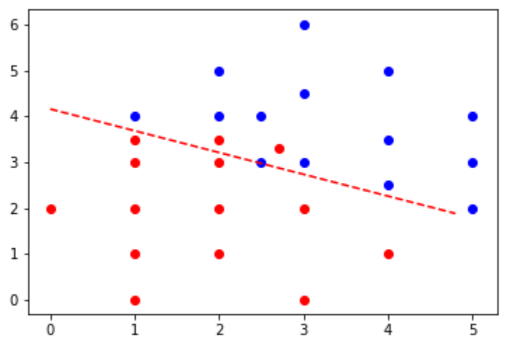

Figure 5: Adaline Hypothesis

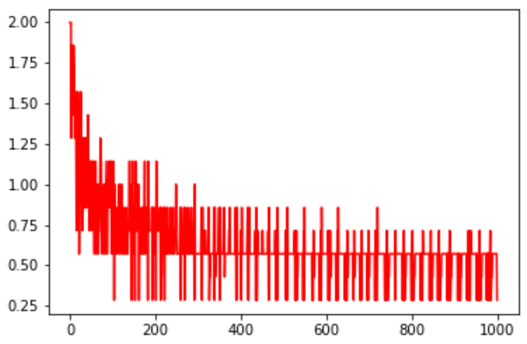

Figure 6: PLA loss evolution

3 Activation functions Examples

3.1 Relu

In the context of artificial neural networks, the rectifier is an activation function defined as the positive part of its argument:

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

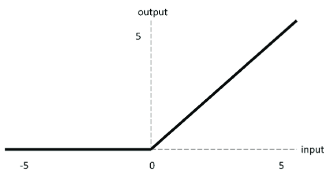

Figure 7: Relu Graph

3.2 Tanh

In the context of artificial neural networks, the tanh is used to map the output of the model to an interval of [-1, 1]:

$$f(x) = tanh(x)$$

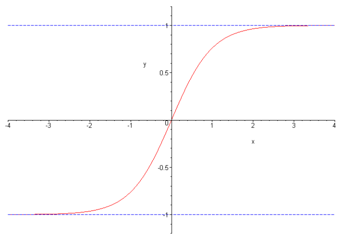

Figure 8: Tanh Graph