**INTRODUCTION MACHINE LEARNING**

**EXERCISE 4**

A red sign with white text

Description automatically generated

|  |
| --- |
| **TEACHER:** |
| Johannes Kiesel |

|  |
| --- |
| **GROUP:** |
| Group 16 |
|  |
| **SUBMITTED BY:** |
| Aaron Perez Herrera |
| Cesar Fernando Gamba Tiusaba |
| Chun Ting Lin |
| Olubunmi Emmanuel Ogunleye |

Exercise 1 : Gradient Descent (1+1+1=3 Points)

(a) Name one difference between the perceptron training rule and the gradient descent method.

The Perceptron training rule is not based on residuals in the (p+1) dimensional input- output-space) but refers to the input space only, where it simply evaluates the side of the hyperplane as a binary feature (correct side or not). Gradient descent is a regression approach that exploits the residuals provided by a loss function of choice, whose differential is evaluated to guide hyperplane search

(b) What are the main requirements for the hypothesis space and error function for gradient descent to be successfully applied?

In batch gradient descent, the weight vector is determined by calculating the differences in weights for each example in the training data and subsequently computing the weight vector for the entire batch in a single step. Incremental gradient descent involves computing the weight vector immediately after calculating the difference in weights for each example in the training dataset, leading to a more iterative and continuous update process. Compared to batch gradient descent, the example-based weight adaptation of incremental gradient descent can better avoid getting stuck in a local minimum of the loss function.

(c) Name the key difference between the algorithm of batch gradient descent (BGD) and the algorithm of incremental gradient descent (IGD).

The perceptron y1(), with a non-zero bias term wo. xo, is more general than yo(), which has a bias term of zero. The bias term acts as an additional flexibility factor, allowing y1() to represent all the functions yo()'s weights can capture, plus additional functions that arise from the non-zero bias. In essence, y1() can model a broader range of relationships in the data, making it more versatile and general than yo(). The bias term allows models to fit the data better by providing additional flexibility.

Exercise 2 : Perceptron Learning (2 Points)

Consider two perceptrons, yo () and y1 (), both defined by the function heaviside(j\_owjj). Both perceptrons have identical weights except for the bias term: wo = 0 for yo() and wo = 1 for y1(). Determine if one of the perceptrons, yo() or y1 (), is more general than the other (as defined in the lecture units on concept learning). If one is more general, specify which one and explain your answer.

We have two perceptrons, y0() and y1(), which are defined using the Heaviside function (a step function) and both have identical weights except for the bias term. Specifically.

* W0 = 0 for y0()
* W0 = 1 for y1()

And each perceptron are represented as:

The bias between y0() and y1() is 0 and 1, respectively. This bias shifts the decision boundary of the perceptron. The general decision rule for a perceptron is:

For perceptron y0(), the output is determined by the sum of the weighted inputs, and since the bias is 0, the decision boundary is where:

For perceptron y1(), the decision boundary is where:

* The perceptron y0() has the decision boundary at
* The perceptron y1() has the decision boundary shifted by -1, i.e., at

Since the bias in y0() is 0, it has a decision boundary at the origin. On the other hand, the bias in y1() shifts the decision boundary. This shift in the decision boundary means that y1() is more general than y0() because it allows for a broader range of inputs to be classified as positive, due to the shift in the threshold.

In conclusion y1() is more general than y0() because the bias in y1() shifts the decision boundary, allowing for a greater variety of inputs to be classified as 1. Therefore, y1() can classify a broader range of inputs, making it more general.

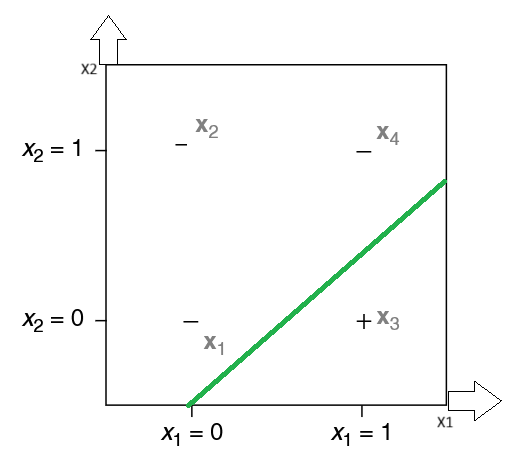
Exercise 3 : Perceptron Learning (1+1+2+2+1=6 Points)

In this exercise, you design a single perceptron with two inputs a1 and x2. This perceptron shall implement the boolean formula A A -B with a suitable function y(x1, x2). Use the values 0 for false and 1 for true.

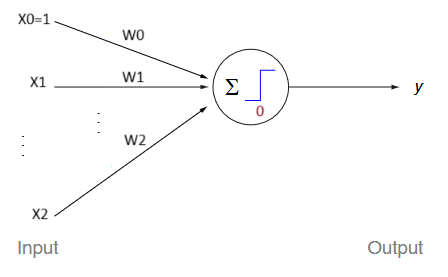
1. Draw all possible examples and a suitable decision boundary in a coordinate system.

First of all, we build the table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Example | A | B | ┐B | A ˄┐B | C |
| X1 | 0 | 0 | 1 | 0 | - |
| X2 | 0 | 1 | 0 | 0 | - |
| X3 | 1 | 0 | 1 | 1 | + |
| X4 | 1 | 1 | 0 | 0 | - |



1. Draw the graph of the perceptron. The schematic must include a1, x2, and all model weights.



1. Manually determine the weights w = (wo, w1, w2) for the decision boundary you drew in (a).

A suitable weight values would be by introducing a negative bias term W0

Considering we are using Heaviside function.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Example | X1 | X2 | y | C |
| X1 | 0 | 0 | -0.5 < 0 | - |
| X2 | 0 | 1 | -1.5 < 0 | - |
| X3 | 1 | 0 | 0.5 > 0 | + |
| X4 | 1 | 1 | -0.5 < 0 | - |

The results reflect the behavior of the perceptron.

1. Now determine w using the perceptron training algorithm (PT). Use a learning rate n of 0.3 and initialize the weights with wo = -0.5 and w1 = w2 = 0.5. Instead of selecting examples randomly, use the following examples in the given order (stop after those four examples):

|  |  |  |
| --- | --- | --- |
| X1 | X2 | C |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Example | x1 | x2 | c | y | ∆W0 | ∆W1 | ∆W2 |
| X1 | 0 | 0 | 0 | 0 | -0.5 | 0.5 | 0.5 |
| X2 | 0 | 1 | 0 | 1 | -0.8 | 0.5 | 0.2 |
| X3 | 1 | 0 | 1 | 0 | -0.5 | 0.8 | 0.2 |
| X4 | 1 | 1 | 0 | 1 | -0.8 | 0.5 | -0.1 |

Final weights

W0 = -0.8

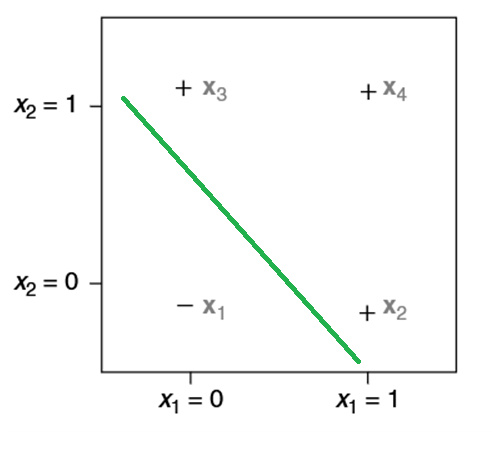
W1 = 0.5

W2 = -0.1

Draw the decision boundary after every weight update into the coordinate system of (a).

First calculation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Example | x1 | x2 | c | y |
| X1 | 0 | 0 | 0 | 0 |
| X2 | 0 | 1 | 0 | 1 |
| X3 | 1 | 0 | 1 | 1 |
| X4 | 1 | 1 | 0 | 1 |



First update

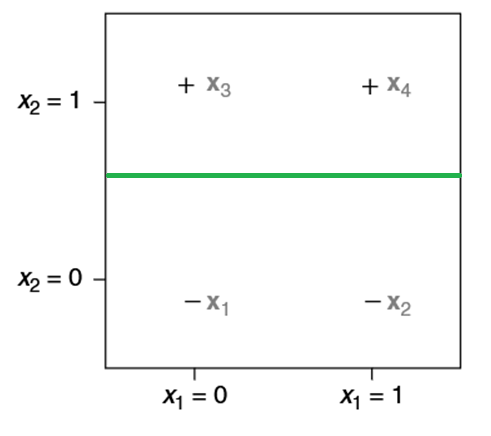
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Example | x1 | x2 | c | y |
| X1 | 0 | 0 | 0 | 0 |
| X2 | 0 | 1 | 0 | 0 |
| X3 | 1 | 0 | 1 | 0 |
| X4 | 1 | 1 | 0 | 0 |

A square with numbers and symbols

Description automatically generated

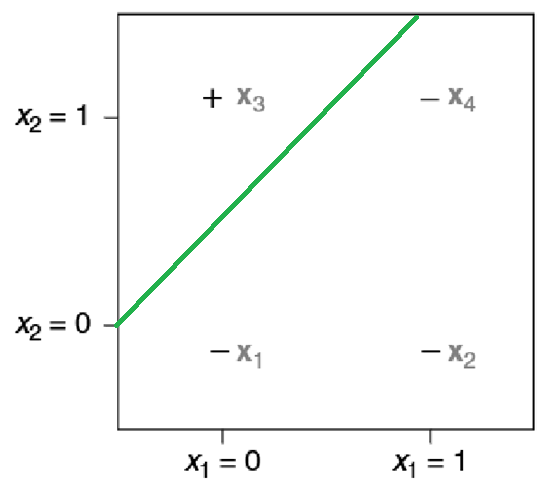
Second update

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Example | x1 | x2 | c | y |
| X1 | 0 | 0 | 0 | 0 |
| X2 | 0 | 1 | 0 | 0 |
| X3 | 1 | 0 | 1 | 1 |
| X4 | 1 | 1 | 0 | 1 |



Third / last update

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Example | x1 | x2 | c | y |
| X1 | 0 | 0 | 0 | 0 |
| X2 | 0 | 1 | 0 | 0 |
| X3 | 1 | 0 | 1 | 1 |
| X4 | 1 | 1 | 0 | 0 |



1. Briefly describe one effect of changing the learning rate n on the learning progress.

A large learning rate means we are making larger changes to the weights every time we do a learning update. However, if the changes are too large, we might oscillate back and forth rather than focusing on a solution.

* Higher η: Faster convergence but risks overshooting optimal weights.
* Lower η: Slower convergence but smoother and more stable learning.