**TRƯỜNG ĐẠI HỌC CẦN THƠ**

KHOA CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



**BÁO CÁO**

**MÁY HỌC NÂNG CAO**

**Nội dung**

**CÀI ĐẶT PERCEPTRON VỚI TENSORFLOW V2.x**

**GVHD:**

**PGS. TS Phạm Nguyên Khang Thành viên**

|  |  |  |
| --- | --- | --- |
| **STT** | **MSSV** | **HỌ VÀ TÊN** |
| 1 | B1709618 | Tạ Đặng Vĩnh Phúc |
| 2 | B1709632 | Đào Công Tính |

**Học kỳ 02, Năm học: 2019-2020**

**MỤC LỤC**

[DANH MỤC HÌNH 3](#_Toc40875443)

[I. Mô tả bài toán 4](#_Toc40875444)

[II. Thiết kế và cài đặt 5](#_Toc40875445)

[1. Đồ thị tính toán 5](#_Toc40875446)

[2. Cài đặt với Tensorflow v2.x 5](#_Toc40875447)

[III. Kiểm thử và đánh giá 7](#_Toc40875448)

[1. Train Perceptron qua 1.000 lần với tốc độ học 7](#_Toc40875449)

[2. Train Perceptron qua 10.000 lần với tốc độ học 8](#_Toc40875450)

[IV. Kết luận 9](#_Toc40875451)

# DANH MỤC HÌNH

Hình 1. Bài toán Perceptron 4

Hình 2. Perceptron với 1 nơ-ron 4

Hình 3. Đồ thị tính toán 5

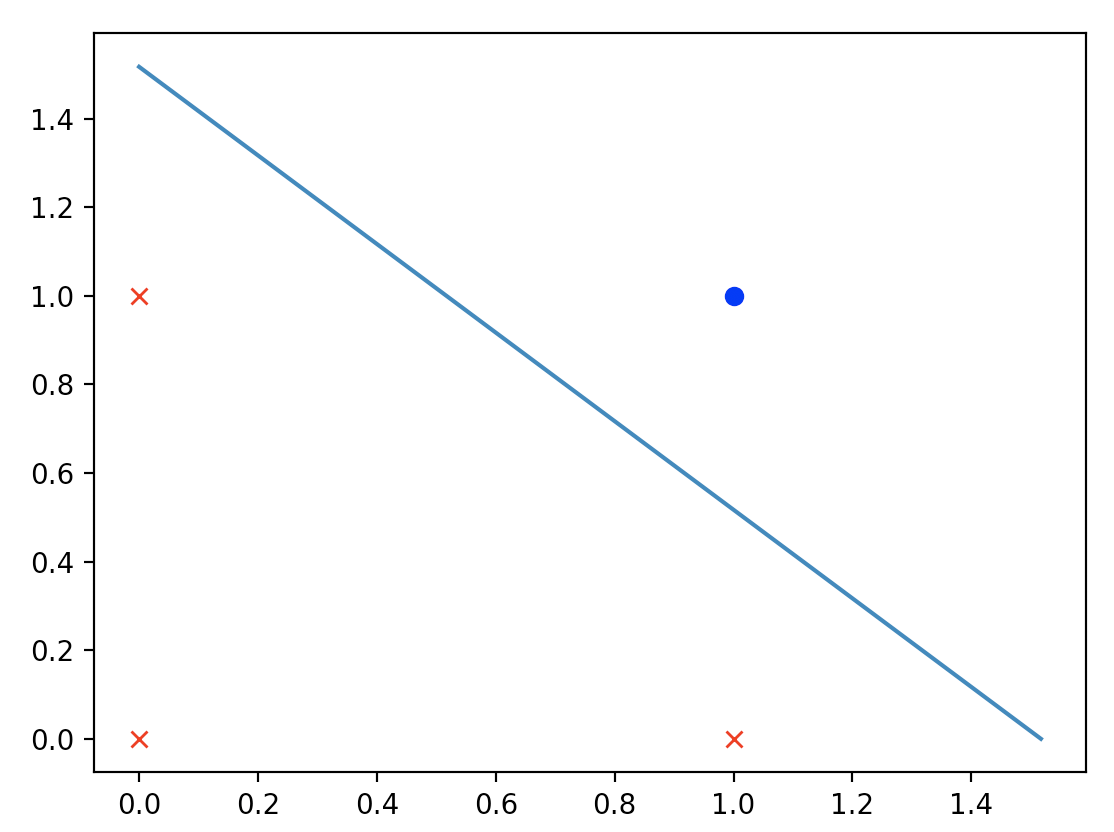
Hình 4. Đồ thị hàm Sigmoid 6

Hình 5. Đồ thị hàm loss qua 1.000 lần train 7

Hình 6. Đồ thị hàm loss qua 10.000 lần train 8

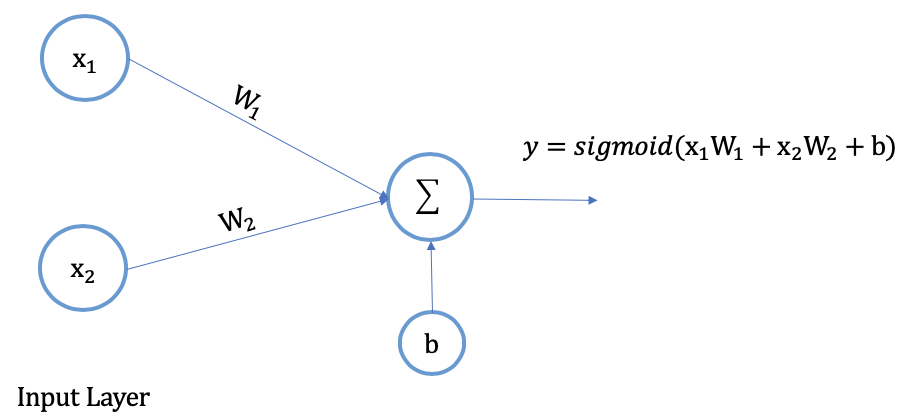
## Mô tả bài toán

**Phát biểu:** Cho hai lớp được gán nhãn, hãy tìm một đường phẳng sao cho toàn bộ các điểm thuộc lớp 0 nằm về 1 phía, toàn bộ các điểm thuộc lớp 1 nằm về phía còn lại của đường thẳng đó. Giả sử tồn tại một đường thẳng như thế (**Hình 1**).



Hình 1. Bài toán Perceptron

Xét 1 Perceptron có 2 đầu vào như **Hình 2**.



Hình 2. Perceptron với 1 nơ-ron

Với dữ liệu đầu vào:

|  |  |  |
| --- | --- | --- |
| X | | y |
| x1 | x2 |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Tìm W, b sao cho tối ưu hàm mất mát (hay giá trị hàm mất mát nhỏ nhất có thể). Hàm mất mát phản ánh độ chênh lệnh giữa giá trị thực tế với giá trị dự đoán.

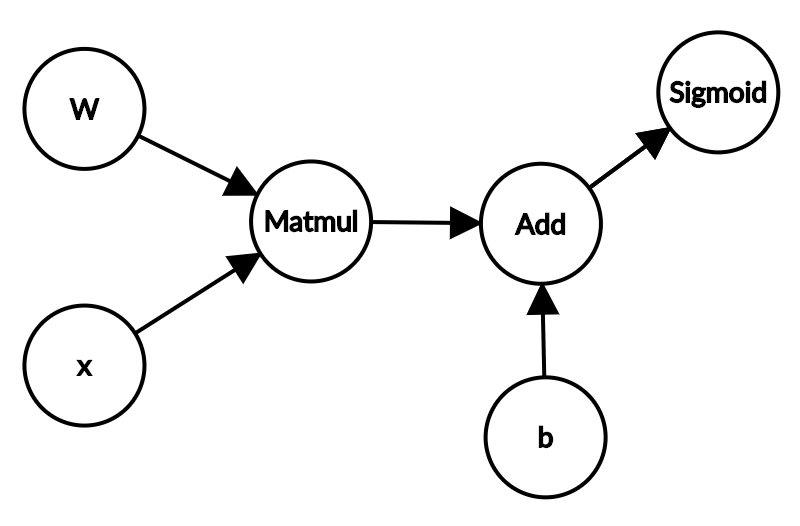
* Hàm mất mát:
* Hàm dự đoán:

Với là 1 vector cột kích thước (shape) , số dòng sẽ đúng bằng “số chiều” của là 1 scalar

## Thiết kế và cài đặt

### Đồ thị tính toán

Từ mô hình Perceptron trên **(Hình 2)**, ta xây dựng được đồ thị tính toán tương ứng:



Hình 3. Đồ thị tính toán

### Cài đặt với Tensorflow v2.x

#### Xây dựng các hàm cần thiết

* Trước tiên import những thư viện cần thiết

|  |
| --- |
| import numpy as np  import tensorflow as tf |

* Bài toán đặt ra là đi tìm W, b nên khi này W, b sẽ là biến (Variable) và được khai báo trong Tensorflow:

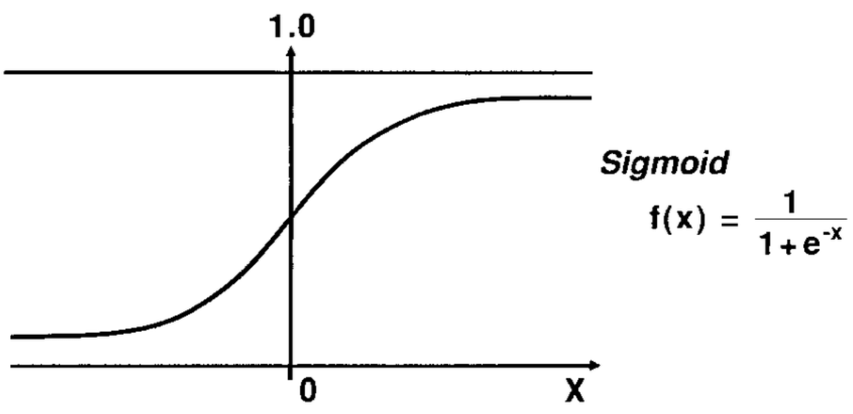
|  |
| --- |
| W = tf.Variable(tf.random.uniform((2, 1), -1, 1))  b = tf.Variable(tf.zeros((1,))) |

* Xây dựng hàm dự đoán

|  |
| --- |
| def predict(x):  return tf.sigmoid(tf.matmul(x, W) + b) |

với tf.matmul nhân 2 ma trận

* Để xác định 1 phần tử thuộc lớp 0 hoặc 1 thì cần phải đặt ra 1 ngưỡng (threshold). Giả sử đặt . Nếu kết quả dự đoán sẽ thuộc lớp 1, ngược lại kết quả sẽ là phần tử thuộc lớp 0.
* Hàm được sử dụng rộng rãi vì nó bị chặn trong khoảng



Hình 4. Đồ thị hàm Sigmoid

* Trong hàm , một sự thay đổi nhỏ trong input dẫn đến một kết quả output không mấy thay đổi. Vì vậy, nó đem lại một đầu ra "mượt" hơn và liên tục hơn so với input.
* Trong bài toán này ta quan tâm đến hàm loss – được xem là hàm mấu chốt và được thực hiện như sau:

|  |
| --- |
| def loss(y\_true, y\_pred):  return tf.reduce\_mean(tf.square(y\_true - y\_pred)) |

với y\_true, y\_pred là tham số vector cột

#### Huấn luyện Perceptron

|  |
| --- |
| def train(X, y, learning\_rate):  with tf.GradientTape() as tape:  current\_loss = loss(y, predict(X))  dW, db = tape.gradient(current\_loss, [W, b])  W.assign\_sub(learning\_rate \* dW)  b.assign\_sub(learning\_rate \* db) |

* Ở đây sử dụng Gradient Descent bằng tf.GradientTape() để tính đạo hàm riêng của hàm loss theo W, b. Quy tắc cập nhật W, b:
* Với là tốc độ học (learning\_rate). Dấu trừ thể hiện việc chúng ta phải đi ngược với đạo hàm (descent). Để cập nhật lại 1 Tensor thì trong Tensorflow có hàm .
* Để tìm được W, b tối ưu hàm mất mát, trong lập trình có một kỹ thuật để tìm 1 giá trị gần bằng tối ưu. Đó là kỹ thuật lặp (loop).

## Kiểm thử và đánh giá

### Train Perceptron qua 1.000 lần với tốc độ học

|  |
| --- |
| print("\nTraining ...")  Ws, bs = [], []  epochs, lr = 10000, 0.2  for epoch in range(epochs):  Ws.append(W)  bs.append(b)  loss = model.train(X, y, learning\_rate=lr)  print("Epoch %d: W1=%.6f, W2=%.6f, b=%.6f, Loss=%.6f" %  (epoch+1, Ws[-1][0], Ws[-1][1], bs[-1], loss)) |

***Dự báo kết quả trên tập huấn luyện:***

W1 = 2.5790522 W2 = 2.576913

b = -3.9937687

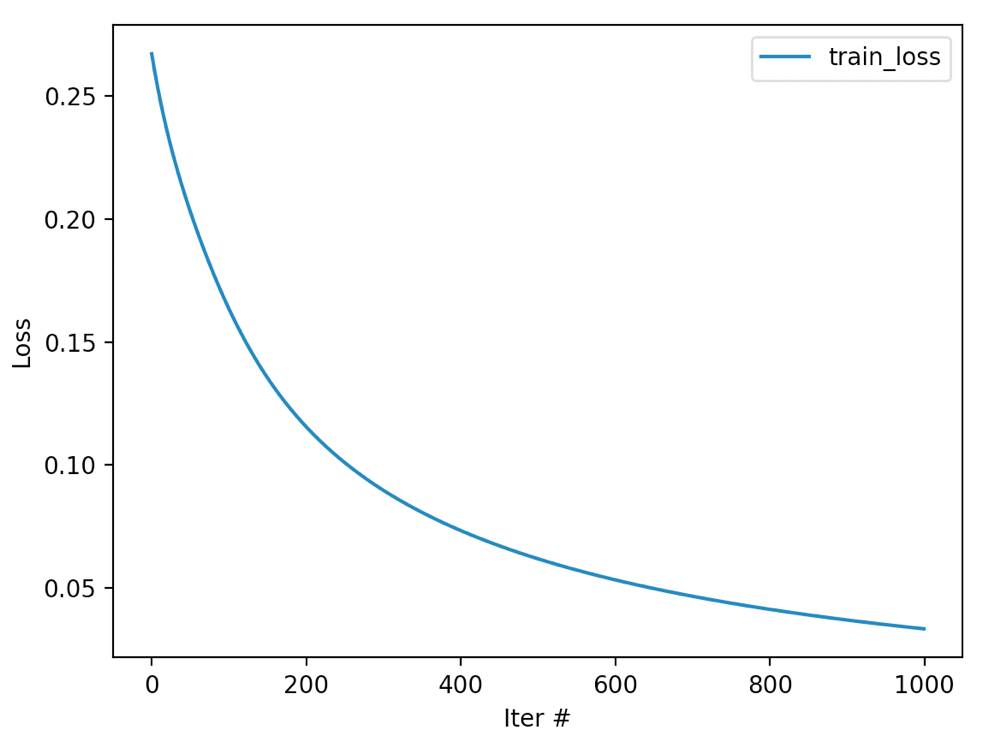
tf.Tensor(

[[0.0180966]

[0.195155 ]

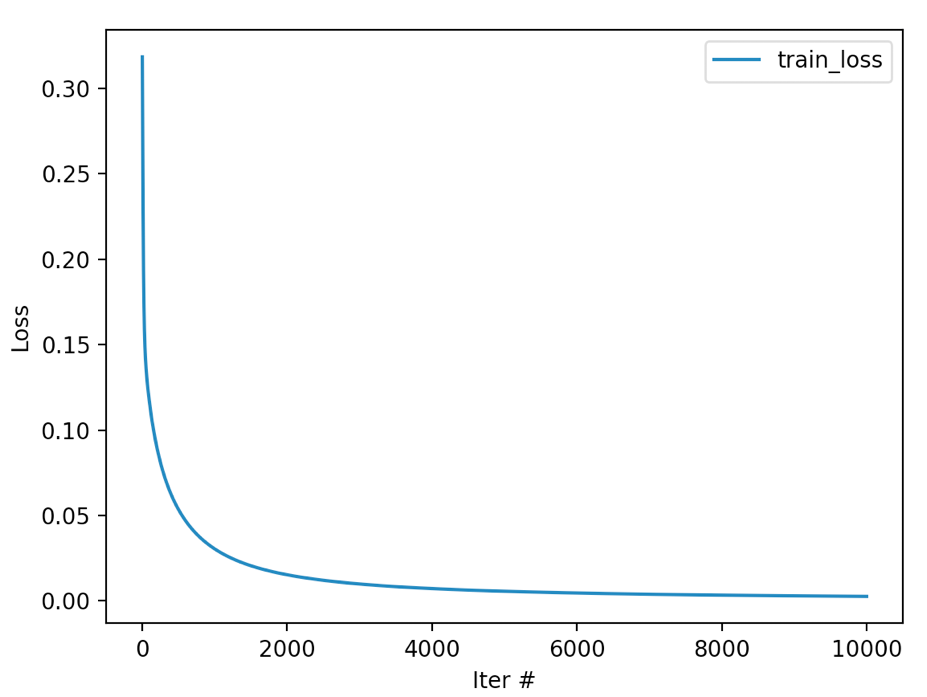
[0.1954912]

[0.7617317]], shape=(4, 1), dtype=float32)



Hình 5. Đồ thị hàm loss qua 1.000 lần train

### Train Perceptron qua 10.000 lần với tốc độ học



Hình 6. Đồ thị hàm loss qua 10.000 lần train

W1 = 5.4843764 W2 = 5.4843764

b = -8.319235

tf.Tensor(

[[2.4372291e-04]

[5.5469304e-02]

[5.5469304e-02]

[9.3398130e-01]], shape=(4, 1), dtype=float32)

Ta nhận xét khi này kết quả dự đoán đã được cải thiện. Hàm loss thực sự giảm đáng kể.

## Kết luận

“Cứ làm đi, sai đâu sửa đấy, cuối cùng sẽ thành công!” là ý tưởng chính của giải thuật học sâu nói chung. Điểm mấu chốt là hàm mất mát sẽ tính độ chênh lệnh giữa giá trị thực tế với giá trị dự đoán. Tuỳ theo dữ liệu mà sử dụng hàm kích hoạt , hay … Huấn luyện mô hình qua một số lần lặp nhất định cùng với một tốc độ học vừa đủ nhỏ. Việc huấn luyện qua nhiều lần sẽ “học” được tốt hơn hay hàm loss sẽ được tối ưu hơn qua việc lấy đạo hàm riêng của hàm loss và cập nhật lại với tốc độ học (learning\_rate).

***\* CODE****:* [*https://github.com/dcongtinh/tensorflow/blob/master/TensorFlow.py*](https://github.com/dcongtinh/tensorflow/blob/master/TensorFlow.py)