**TRƯỜNG ĐẠI HỌC CẦN THƠ**

KHOA CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



**BÁO CÁO**

**MÁY HỌC NÂNG CAO**

**Tiêu đề**

**BÀI TOÁN XOR VỚI MẠNG NƠ-RON ĐA TẦNG**

**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](#_Toc40982839)

[I. Mô tả bài toán 4](#_Toc40982840)

[II. Thiết kế và cài đặt 6](#_Toc40982841)

[1. Đồ thị tính toán 6](#_Toc40982842)

[2. Cài đặt với Tensorflow v2.x 6](#_Toc40982843)

[III. Kiểm thử và đánh giá 8](#_Toc40982844)

[1. Train Perceptron qua 3.000 lần với tốc độ học 8](#_Toc40982845)

[2. Train Perceptron qua 10.000 lần với tốc độ học 9](#_Toc40982846)

[IV. Kết luận 10](#_Toc40982847)

# DANH MỤC HÌNH

Hình 1.Biểu diễn dữ liệu cho bài toán XOR 4

Hình 2. Mô hình Perceptron 4

Hình 3. Đồ thị tính toán 6

Hình 4. Đồ thị hàm Sigmoid 7

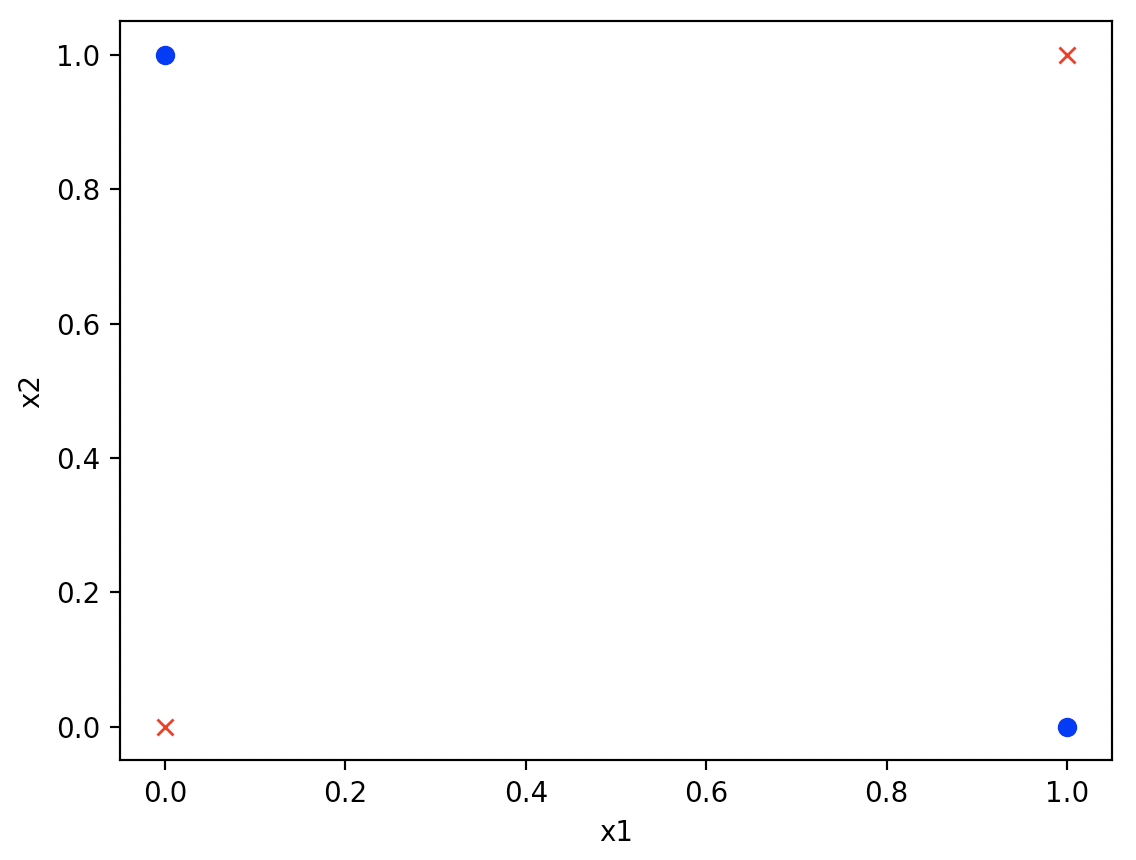
Hình 5. Đồ thị hàm loss qua 3.000 lần train 8

Hình 6. Đồ thị hàm loss qua 10.000 lần train 9

## Mô tả bài toán

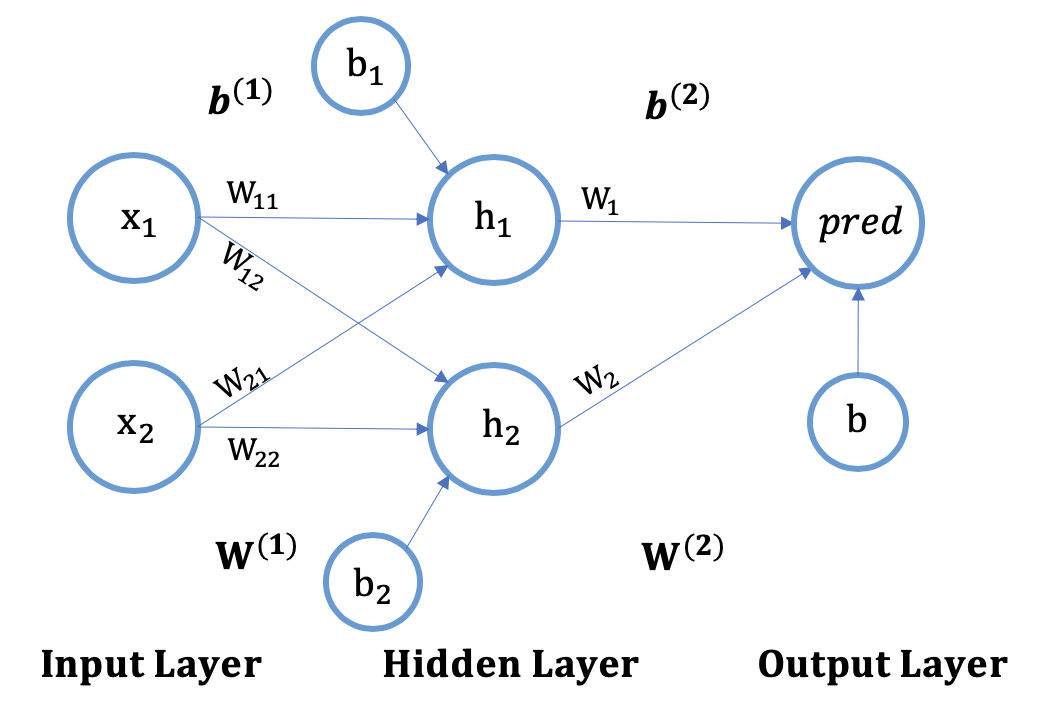
* Bảng chân trị cho toán tử XOR ():

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



Hình 1.Biểu diễn dữ liệu cho bài toán XOR

* Đối với bài toán tuyến tính sẽ tìm được 1 đường thẳng phân chia giữa các điểm. Tuy nhiên đối với bài toán XOR – bài toán phi tuyến sẽ không thể thực hiện theo cách như thế. Để giải quyết bài toán XOR ta cần thêm 1 hidden layer với 2 nơ-ron.
* Xét 1 Perceptron có 2 đầu vào như **Hình 2**.



Hình 2. Mô hình Perceptron

* Mô hình trên có 2 layer (số lượng layer của mô hình không tính input layer)
* Mô hình: 2-2-1, nghĩa là 2 node trong input layer, 1 hidden layer có 2 node và output layer có 1 node.
* Input layer và hidden layer luôn thêm node 1 để tính b cho layer sau, nhưng không tính vào số lượng node trong layer
* Ở mỗi node trong hidden layer và output layer đều thực hiện 2 bước: tính tổng linear () và áp dụng hàm kích hoạt (Sigmoid).
* Với dữ liệu đầu vào:

|  |  |  |
| --- | --- | --- |
|  | |  |
|  |  |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

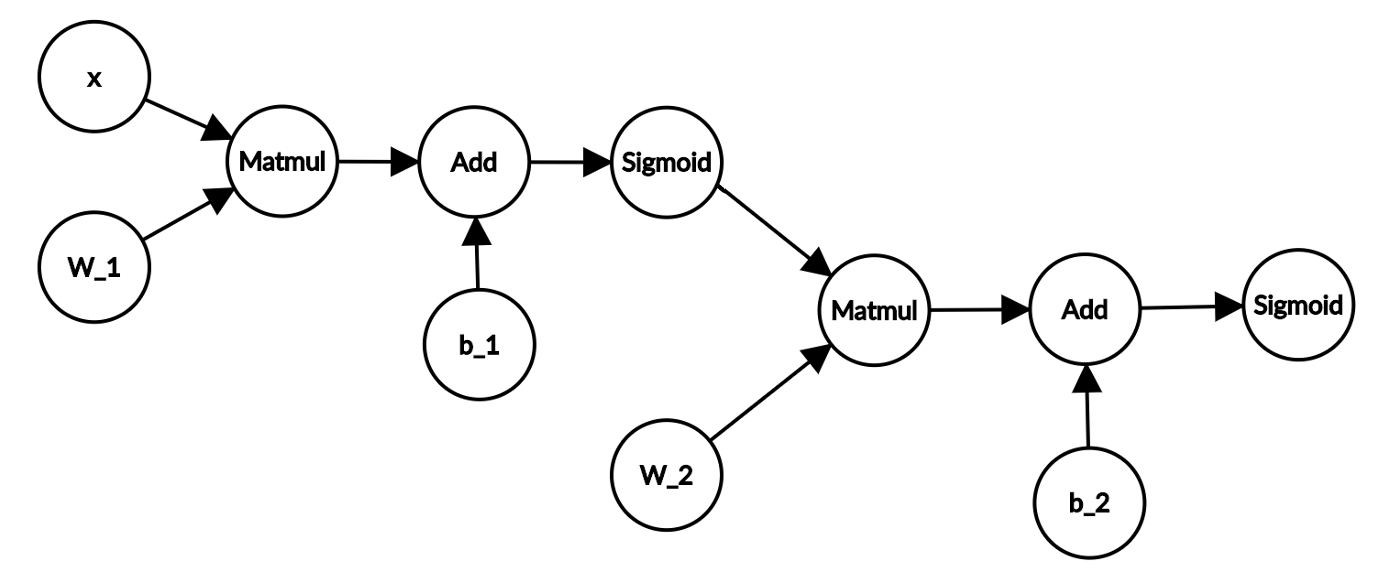
* Tìm sao cho tối ưu hàm mất mát (hay giá trị hàm mất mát nhỏ nhất có thể). Hàm mất mát phản ánh độ chênh lệnh giữa giá trị thực tế với giá trị dự đoán.
* Hàm mất mát:
* Hàm dự đoán:

Với là vector cột kích thước (shape) - số dòng sẽ đúng bằng “số chiều” của là vector cột kích thước (shape) - số dòng sẽ đúng bằng số lớp của tập huấn luyện.

## Thiết kế và cài đặt

### Đồ thị tính toán

Từ mô hình Perceptron trên **(Hình 1)** 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:

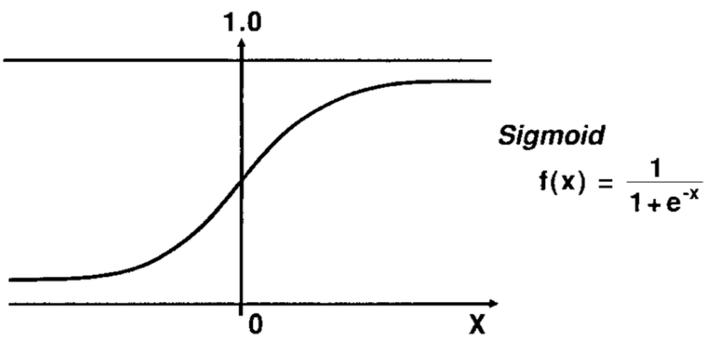
|  |
| --- |
| W1 = tf.Variable(tf.random.uniform((2, 2), -1, 1))  b1 = tf.Variable(tf.zeros((2,)))  W2 = tf.Variable(tf.random.uniform((2, 1), -1, 1))  b2 = tf.Variable(tf.zeros((1,))) |

* Xây dựng hàm dự đoán

|  |
| --- |
| def predict(x):  hidden = tf.nn.sigmoid(tf.matmul(x, W1) + b1)  return tf.nn.sigmoid(tf.matmul(hidden, W2) + b2) |

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.reduce\_sum(  y\_true\*tf.math.log(y\_pred)+  (tf.constant(1.0)-y\_true)\*tf.math.log(tf.constant(1.0)-y\_pred),1)) |

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))  dW1, db1, dW2, db2 = tape.gradient(current\_loss,  [W1, b1, W2, b2])  W1.assign\_sub(learning\_rate \* dW1)  b1.assign\_sub(learning\_rate \* db1)  W2.assign\_sub(learning\_rate \* dW2)  b2.assign\_sub(learning\_rate \* db2) |

* Ở đây sử dụng Gradient Descent bằng tf.GradientTape() để tính đạo hàm riêng của hàm loss theo . Quy tắc cập nhật :
* 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).

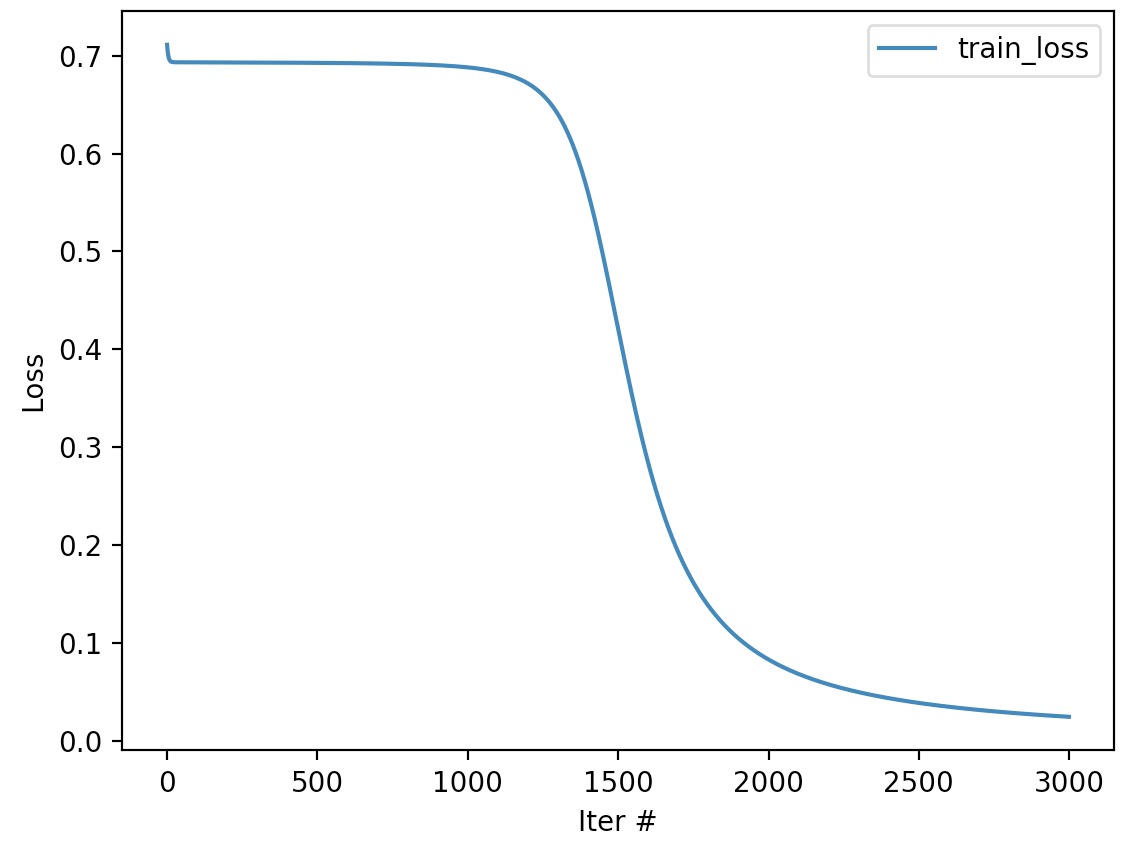
## Kiểm thử và đánh giá

### Train Perceptron qua 3.000 lần với tốc độ học

|  |
| --- |
| print("\nTraining ...")  epochs, lr = 3000, 0.3  for epoch in range(epochs):  loss = model.train(X, y, learning\_rate=lr)  print("Epoch %d: Loss=%.6f" % (epoch+1, loss)) |

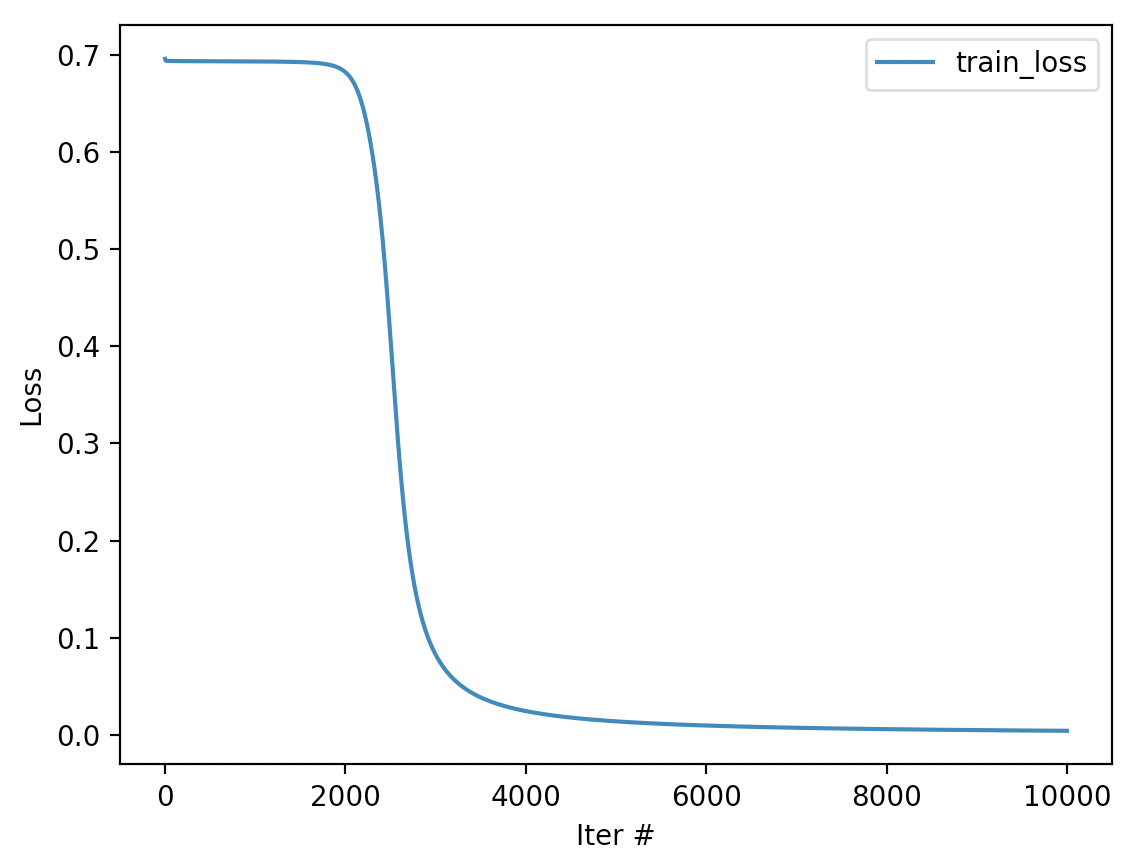
***Dự báo kết quả trên tập huấn luyện:***

|  |
| --- |
| W1 = <tf.Variable 'Variable:0' shape=(2, 2) dtype=float32, numpy=  array([[ 5.535559 , 5.8883457],  [-5.762126 , -5.647575 ]], dtype=float32)>  b1 = <tf.Variable 'Variable:0' shape=(2,) dtype=float32, numpy=array([-2.9975612, 2.8343213], dtype=float32)>  W2 = <tf.Variable 'Variable:0' shape=(2, 1) dtype=float32, numpy=  array([[ 9.070498],  [-8.497471]], dtype=float32)>  b2 = <tf.Variable 'Variable:0' shape=(1,) dtype=float32, numpy=array([3.8937726], dtype=float32)>  tf.Tensor(  [[0.02410506]  [0.9681434 ]  [0.97820187]  [0.02020412]], shape=(4, 1), dtype=float32) |



Hình 5. Đồ thị hàm loss qua 3.000 lần train

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



Hình 6. Đồ thị hàm loss qua 10.000 lần train

|  |
| --- |
| W1 = <tf.Variable 'Variable:0' shape=(2, 2) dtype=float32, numpy=  array([[ 6.7487874, 6.9158216],  [-6.953611 , -6.6455245]], dtype=float32)>  b1 = <tf.Variable 'Variable:0' shape=(2,) dtype=float32, numpy=array([-3.5954645, 3.3104675], dtype=float32)>  W2 = <tf.Variable 'Variable:0' shape=(2, 1) dtype=float32, numpy=  array([[ 12.266032],  [-11.770539]], dtype=float32)>  b2 = <tf.Variable 'Variable:0' shape=(1,) dtype=float32, numpy=array([5.5611243], dtype=float32)>  tf.Tensor(  [[0.00420509]  [0.99427223]  [0.99614537]  [0.00360394]], shape=(4, 1), dtype=float32) |

## Kết luận

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