**TRƯỜNG ĐẠI HỌC CẦN THƠ**

KHOA CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

🙠🙟🕮🙝🙢

**BÁO CÁO**

**MÁY HỌC NÂNG CAO**

**Tiêu đề**

**ỨNG DỤNG PERCEPTRON PHÂN LOẠI HOA IRIS**

**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](#_Toc40969574)

[I. Mô tả bài toán 4](#_Toc40969575)

[II. Thiết kế và cài đặt 5](#_Toc40969576)

[1. Đồ thị tính toán 5](#_Toc40969577)

[2. Cài đặt với Tensorflow v2.x 5](#_Toc40969578)

[III. Kiểm thử và đánh giá 7](#_Toc40969579)

[1. Train Perceptron qua 1.000 lần với tốc độ học 7](#_Toc40969580)

[2. Train Perceptron qua 10.000 lần với tốc độ học 8](#_Toc40969581)

[IV. Kết luận 8](#_Toc40969582)

# DANH MỤC HÌNH

[Hình 1. Mô hình Perceptron 4](#_Toc40969594)

[Hình 2. Đồ thị tính toán 5](#_Toc40969595)

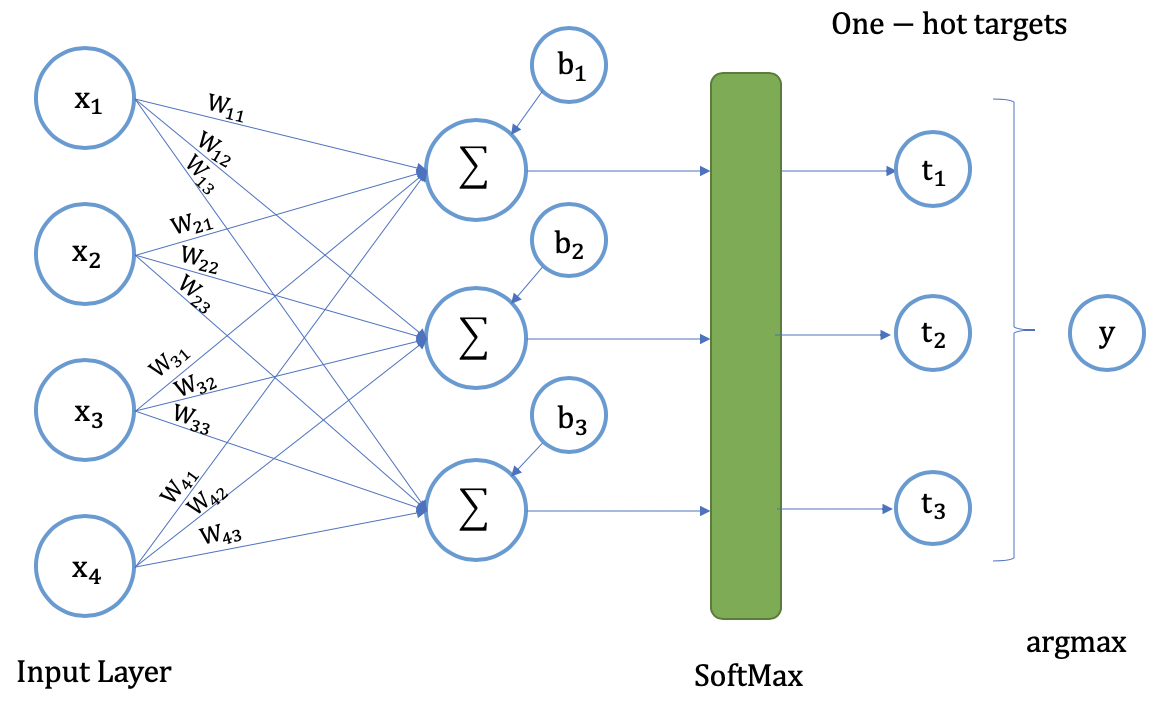
[Hình 3. Ví dụ SoftMax 6](#_Toc40969596)

[Hình 4. Đồ thị hàm loss qua 1.000 lần train 7](#_Toc40969597)

[Hình 5. Đồ thị hàm loss qua 10.000 lần train 8](#_Toc40969598)

## Mô tả bài toán

* Xét 1 Perceptron có 4 đầu vào như **Hình 1**.



Hình 1. Mô hình Perceptron

* Mô hình trên có 2 layer (số lượng layer của mô hình không tính input layer)
* Mô hình: 4-3, nghĩa là 4 node trong input layer và output layer có 3 node.
* Node output layer thực hiện 2 bước: tính tổng linear () và áp dụng hàm kích hoạt ().
* Với dữ liệu đầu vào là dữ liệu từ tập IRIS (thư viện ). Tập dữ liệu huấn luyện gồm n dòng, m=5 cột. Mỗi dòng của tập biểu thị một điểm dữ liệu (được phân cách bởi dấu “,”). 4 cột đầu là các đặc trưng và cột còn lại là nhãn của tập dữ liệu.
* Tìm W, b sao cho tối ưu hàm mất mát (hay giá trị hàm mất mát nhỏ nhất có thể). Hàm mất mát là đọ đo so sánh 2 phân phối xác suất.
* Hàm mất mát (cross entropy) so sánh 2 phân phối xác suất:
* 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.

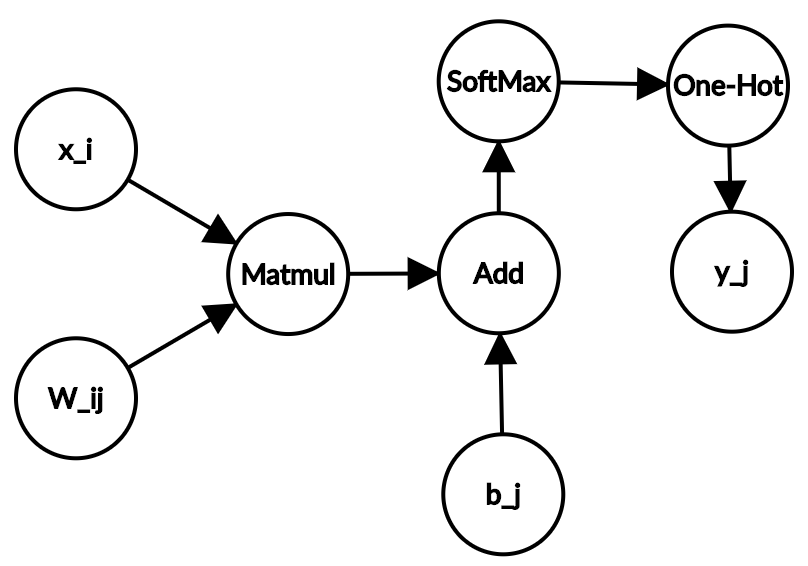
* Tuy nhiên với bài toán đa lớp (multi-classes), ta cần thông qua một lớp xử lý đầu ra sao cho giá trị trả về là ma trận 0 1 kích thước . **One-hot endcoding** là quá trình biến đổi từng giá trị thành các đặc trưng nhị phân chỉ chứa giá trị 1 hoặc 0. Mỗi mẫu trong đặc trưng phân loại sẽ được biến đổi thành một véc-tơ có kích thước chỉ với một trong các giá trị là 1 (biểu thị nó là nhãn). Ví dụ cho tập dữ liệu IRIS:

|  |  |  |
| --- | --- | --- |
|  | → | [[1,0,0]  [0,1,0]  [0,1,0]  [0,0,1]] |

## 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 2. 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 time  import numpy as np  import tensorflow as tf  from sklearn import datasets  import matplotlib.pyplot as plt  from sklearn.preprocessing import OneHotEncoder  from sklearn.metrics import accuracy\_score, precision\_score, f1\_score, recall\_score |

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

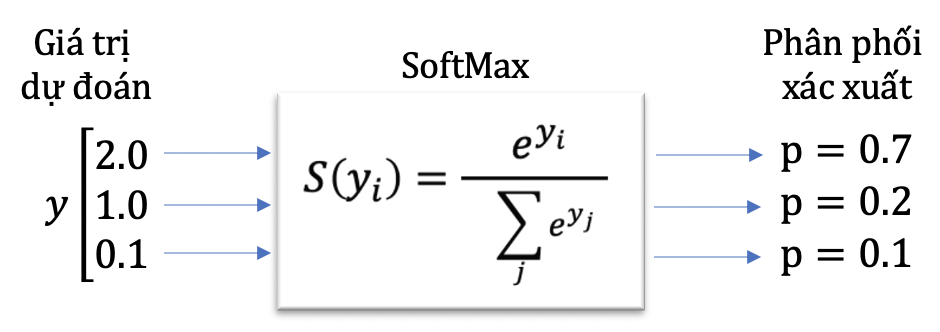
|  |
| --- |
| W = tf.Variable(tf.random.uniform((4, 3), -1, 1))# 4 “chiều” 3 cột  b = tf.Variable(tf.zeros((3,))) # 3 nơ-ron |

* Xây dựng hàm dự đoán

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

với tf.matmul nhân 2 ma trận

* Hàm được sử dụng để chuẩn hoá một phân phối xác suất Ví dụ:



Hình 3. Ví dụ SoftMax

* Giá trị dự đoán của một dòng dữ liệu sẽ là 1 tại vị trí có xác suất lớn nhất (argmax) ngược sẽ trả về 0 cho các vị trí còn lại (sử dụng chiến thuật “1 chống lại tất cả” - chọn 1 lớp dương còn lại là lớp âm).
* 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):  cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y\_true \*  tf.math.log(y\_pred), 1))  return cross\_entropy |

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 ...")

epochs, lr = 10000, 0.2

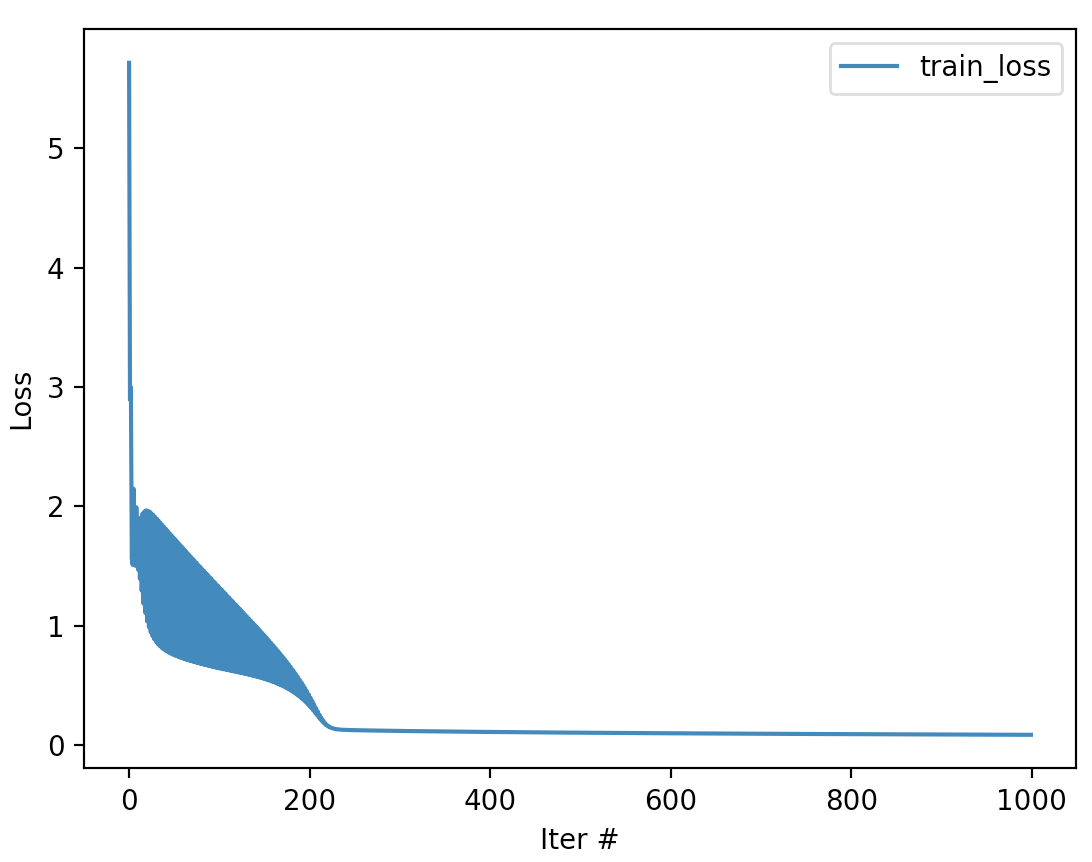
for epoch in range(epochs):

loss = model.train(X, y, learning\_rate=lr)

print("Epoch %d: Loss=%.6f" % (epoch+1, loss))

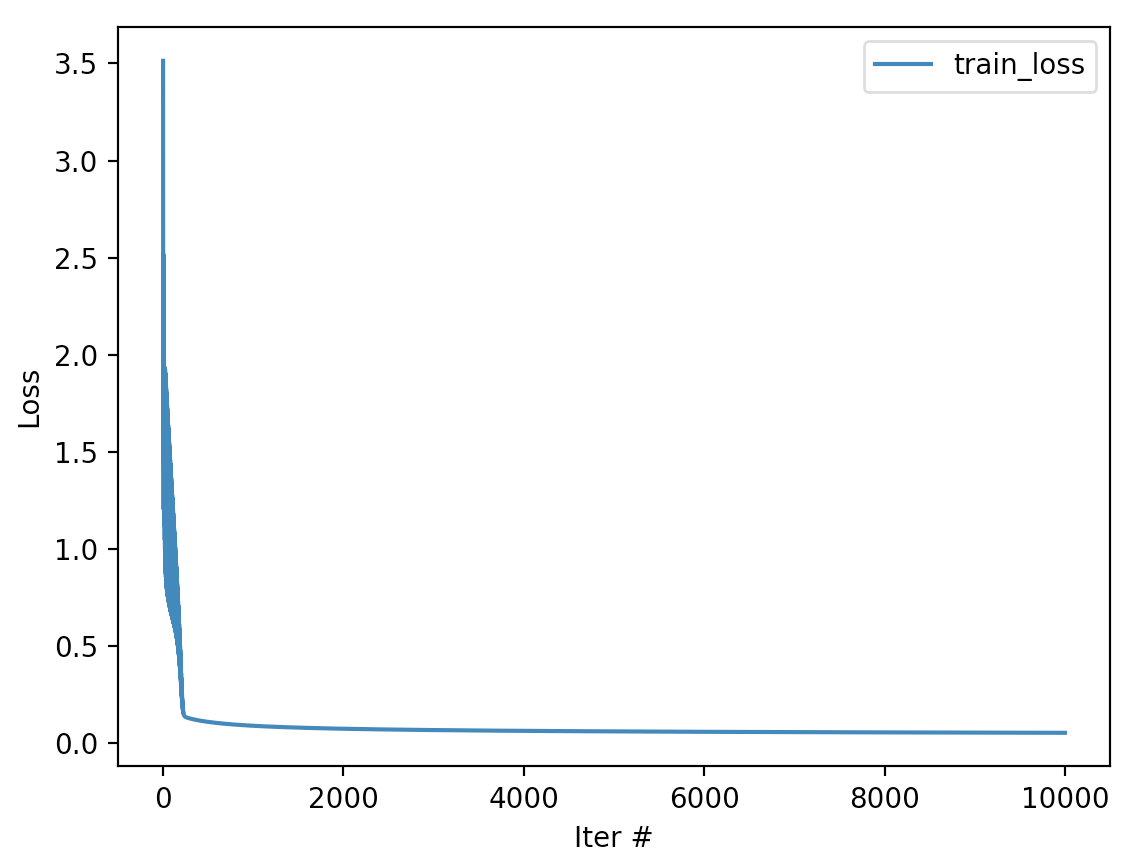
***Dự báo kết quả trên tập huấn luyện:***

|  |
| --- |
| W = <tf.Variable 'Variable:0' shape=(4, 3) dtype=float32, numpy=  array([[ 1.5116544 , 0.61015606, -2.3082743 ],  [ 2.1916397 , 0.27243236, -2.6108851 ],  [-3.1376963 , 0.25526303, 4.255493 ],  [-2.6608162 , -1.646165 , 4.0829167 ]], dtype=float32)>  b = <tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([ 0.43501133, 1.0828398 , -1.517854 ], dtype=float32)>  Accuracy = 98.0%  Precision = 98.01253834867279%  F1\_Score = 97.99979997999799%  Recall = 98.0% |



Hình 4. Đồ thị hàm loss qua 1.000 lần train

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



Hình 5. Đồ thị hàm loss qua 10.000 lần train

|  |
| --- |
| W = <tf.Variable 'Variable:0' shape=(4, 3) dtype=float32, numpy=  array([[ 2.2698095 , 1.1273406 , -2.6577358 ],  [ 4.700799 , 0.60752493, -4.0038223 ],  [-6.0546603 , -0.3020501 , 5.9684043 ],  [-2.071966 , -2.8469336 , 6.4087663 ]], dtype=float32)>  b = <tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([ 0.8740186, 4.281873 , -5.1558914], dtype=float32)>  Accuracy = 98.0%  Precision = 98.01253834867279%  F1\_Score = 97.99979997999799%  Recall = 98.0% |

## Kết luận

Phân loại đa lớp sử dụng Perceptron bản chất không khác mấy so với phân loại nhị nhân sử dụng Perceptron. Có 2 điểm quan trọng đó là dữ liệu đầu ra và hàm mất mát. Đối với dữ liệu đầu ra của phân loại đa lớp là làm sao để biến đổi về dạng ma trận 0 1 kích thước (với là số lượng mẫu của tập dữ liệu và là số lượng nhãn). **One-hot endcoding** là quá trình biến đổi từng giá trị thành các đặc trưng nhị phân chỉ chứa giá trị 1 hoặc 0. Mỗi mẫu trong đặc trưng phân loại sẽ được biến đổi thành một véc-tơ có kích thước chỉ với một trong các giá trị là 1 (biểu thị nó là nhãn). Tiếp theo là xây dựng hàm mất mát sử dụng hàm . Là 1 hàm để chuẩn hoá một phân phối xác suất. Nhãn dự đoán của 1 dòng dữ liệu sẽ chính là vị trí có xác suất lớn nhất.

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