BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC CẦN THƠ**

**KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**

**🙞 🕮 🙜**



**BÀI TẬP NHÓM**

**MÁY HỌC NÂNG CAO**

**ĐỀ TÀI**

**NEURAL NETWORK**

**Giảng viên hướng dẫn:**

**Phạm Nguyên Khang**

**Sinh viên thực hiện:**

|  |  |
| --- | --- |
| Nguyễn Thành Hoàng Hải | B1812339 |
| Võ Ngọc Long | B1812282 |

Cần Thơ, 11/2021

# MỤC LỤC

[MỤC LỤC 1](#_Toc87431257)

[DANH MỤC HÌNH 3](#_Toc87431258)

[MÔ TẢ YÊU CẦU BÀI TOÁN 4](#_Toc87431259)

[I. YÊU CẦU 1 4](#_Toc87431260)

[II. YÊU CẦU 2 4](#_Toc87431261)

[BÁO CÁO KẾT QUẢ 5](#_Toc87431262)

[I. Yêu cầu 1 5](#_Toc87431263)

[1. Vấn đề 5](#_Toc87431264)

[2. Thư viện sử dụng 5](#_Toc87431265)

[3. Khởi tạo các biến cần thiết 5](#_Toc87431266)

[4. Cài đặt hàm predict 5](#_Toc87431267)

[5. Cài đặt hàm lỗi 6](#_Toc87431268)

[6. Cài đặt hàm huấn luyện 6](#_Toc87431269)

[7. Đọc tập dữ liệu iris 6](#_Toc87431270)

[8. Chia tập dữ liệu 7](#_Toc87431271)

[9. Vòng lặp dùng cập nhật và ghi nhận các giá trị accuracy, loss 7](#_Toc87431272)

[10. Hiển thị ma trận confusion của model trên tập test 8](#_Toc87431273)

[11. Kết quả 8](#_Toc87431274)

[11.1. Kết quả độ chính xác 8](#_Toc87431275)

[11.2. Kết quả độ lỗi 9](#_Toc87431276)

[11.3. Mô tả chi tiết phần dự đoán test của Model 10](#_Toc87431277)

[II. Yêu cầu 2: 10](#_Toc87431278)

[1. Vấn đề: 10](#_Toc87431279)

[2. Giới thiệu phép XOR 10](#_Toc87431280)

[3. Thư viện sử dụng 11](#_Toc87431281)

[4. Khởi tạo các biến cần thiết 11](#_Toc87431282)

[5. Cài đặt hàm predict 11](#_Toc87431283)

[6. Cài đặt hàm lỗi 12](#_Toc87431284)

[7. Cài đặt hàm huấn luyện 12](#_Toc87431285)

[8. Vòng lặp dùng cập nhật và ghi nhận loss 13](#_Toc87431286)

[9. Kết quả 13](#_Toc87431287)

[9.1. Kết quả độ lỗi 13](#_Toc87431288)

[9.2. Kết quả dự đoán phép XOR 14](#_Toc87431289)

[TÀI LIỆU THAM KHẢO 15](#_Toc87431290)

[PHỤ LỤC 16](#_Toc87431291)

[I. Mã nguồn của yêu cầu 1 16](#_Toc87431292)

[II. Mã nguồn của yêu cầu 2 18](#_Toc87431293)

# DANH MỤC HÌNH

[Hình 1: Đồ thị hàm predict trong mô hình iris 5](#_Toc87431251)

[Hình 2: Kết quả độ chính xác mô hình iris 7](#_Toc87431252)

[Hình 3: Kết quả độ lỗi mô hình iris 8](#_Toc87431253)

[Hình 4: Mô tả chi tiết phần dự đoán test của mô hình iris 9](#_Toc87431254)

[Hình 5: Đồ thị hàm dự đoán của model XOR 11](#_Toc87431255)

[Hình 6: Kết quả độ lỗi của mô hình XOR 2 tầng 12](#_Toc87431256)

# MÔ TẢ YÊU CẦU BÀI TOÁN

## YÊU CẦU 1

Xây dựng Mạng Noron 1 tầng với nhiều ngõ ra để phân lớp tập dữ liệu IRIS.

Dữ liệu được cho để huấn luyện mô hình: [UCI Machine Learning Repository: Iris Data Set](https://archive.ics.uci.edu/ml/datasets/Iris)

## YÊU CẦU 2

Dùng Tensorflow xây dựng mạng 2 tầng giải bài toán XOR.

# BÁO CÁO KẾT QUẢ

## Yêu cầu 1

### Vấn đề

Xây dựng 1 mô hình mạng nơ ron 1 tầng để phân loại tập dữ liệu iris

### Thư viện sử dụng

Để tiến hành giải quyết yêu cầu với tập dữ liệu iris, trước tiên cần nhập những thư viện cần thiết của Python

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

import pandas as pd

import tensorflow as tf

### Khởi tạo các biến cần thiết

W = tf.Variable(tf.eye(4, 3))

b = tf.Variable(tf.zeros((3,)))

Các biến được khởi tạo:

* W: là một ma trận gồm 4 hàng và 3 cột (4 thuộc tính, 3 nơ ron).

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

* b: là bias của mỗi nơ ron trong W. Trong trường hợp này là một mảng các con số 0.

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |

### Cài đặt hàm predict

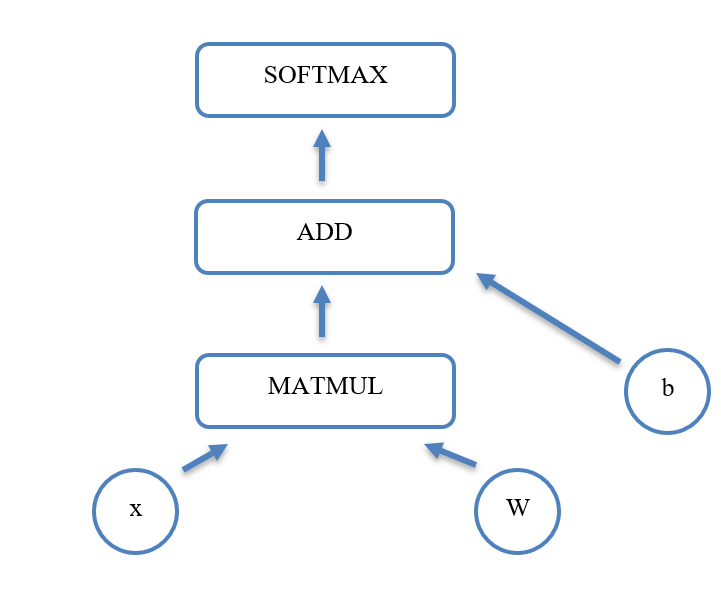
def predict(x: tf.Tensor) -> tf.Tensor:

return tf.nn.softmax(tf.matmul(x, W) + b)

Giá trị x: có 4 thuộc tính.

Ta thực hiện phép toán nhân x và W. Kết quả của phép toán vừa rồi, ta đem cộng với phần tử b.

Sử dùng hàm softmax để trả về giá trị trung bình, là một con số.



Hình 1: Đồ thị hàm predict trong mô hình iris

### Cài đặt hàm lỗi

Thiết kế dựa trên công thức:

* : giá trị mong muốn của lớp j.
* : giá trị dự báo của lớp j.

def loss(y\_true, y\_pred) -> tf.Tensor:

return tf.reduce\_mean(-tf.reduce\_sum(y\_true \* tf.math.log(y\_pred), 1))

### Cài đặt hàm huấn luyện

def train(x\_train, y\_train, lr: float = 0.001) -> None:

with tf.GradientTape() as tape:

train\_loss = loss(y\_train, predict(x\_train))

dW, db = tape.gradient(train\_loss, [W, b])

W.assign\_sub(lr\*dW)

b.assign\_sub(lr\*db)

* Tính giá trị **train\_loss.**
* Sử dụng **GradientTape()** lưu trữ quá trình tính toán train\_loss.
* Lấy đạo hàm của W và b theo hàm lỗi thông qua **GradientTape().**
* Cập nhật lại giá trị W và b với tốc độ học.

### Đọc tập dữ liệu iris

col\_names = ["sepal length in cm",

"sepal width in cm",

"petal length in cm",

"petal width in cm",

"class"]

df = pd.read\_csv("iris.data", names=col\_names)

df\_x = df.drop("class", axis=1)

enc\_y = LabelEncoder()

df\_y = enc\_y.fit\_transform(df["class"])

### Chia tập dữ liệu

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df\_x,df\_y,test\_size=0.2, random\_state=1911)

x\_train = tf.Variable(x\_train , dtype=tf.float32)

x\_test = tf.Variable(x\_test , dtype=tf.float32)

y\_train = tf.one\_hot(y\_train, len(enc\_y.classes\_))

y\_test = tf.one\_hot(y\_test , len(enc\_y.classes\_))

Tập dữ liệu iris được chia theo tỉ lệ:

* Tập train: 80% (120 phần tử)
* Tập test: 20% (30 phần tử)

Với phần tử x\_train, x\_test: chuyển về dạng tf.Variable.

Với phần tử y\_train, y\_test:

Số

one\_hot

Lable

* Chuyển các giá trị train, test sang dạng tf.Variable để phù hợp khi sử dụng tensorflow.

### Vòng lặp dùng cập nhật và ghi nhận các giá trị accuracy, loss

f\_loss\_train = open("iris\_loss\_train.txt", 'w')

f\_loss\_test = open("iris\_loss\_test.txt" , 'w')

f\_accurary\_train = open("iris\_accurary\_train.txt", 'w')

f\_accurary\_test = open("iris\_accurary\_test.txt" , 'w')

for epoch in range(100000):

train(x\_train, y\_train, 0.01)

y\_true\_train = tf.argmax(y\_train , 1)

y\_true\_test = tf.argmax(y\_test , 1)

y\_pred\_train = predict(x\_train)

y\_pred\_test = predict(x\_test)

train\_loss = float(loss(y\_train, y\_pred\_train))

test\_loss = float(loss(y\_test , y\_pred\_test))

train\_accuracy = accuracy\_score(y\_true\_train,tf.argmax(y\_pred\_train, 1))

test\_accuracy = accuracy\_score(y\_true\_test, tf.argmax(y\_pred\_test , 1))

print("train loss: {}\ttrain accuracy: {}\ttest loss: {}\ttest accuracy: {}".format(train\_loss, train\_accuracy, test\_loss, test\_accuracy))

f\_loss\_train.write("{}\n".format(train\_loss))

f\_loss\_test .write("{}\n".format(test\_loss))

f\_accurary\_train.write("{}\n".format(train\_accuracy))

f\_accurary\_test .write("{}\n".format(test\_accuracy))

f\_loss\_train.close()

f\_loss\_test .close()

f\_accurary\_train.close()

f\_accurary\_test .close()

* Mở file để lưu lại các giá trị accuracy và loss trong lúc huấn luyện
* Thực hiện lặp 100000 lần
* Tốc độ học là 0,01
* Mục đích: cập nhật W và b
* Đóng file sau khi huấn luyện

### Hiển thị ma trận confusion của model trên tập test

disp = ConfusionMatrixDisplay(confusion\_matrix(tf.argmax(y\_test, 1),

tf.argmax(predict(x\_test), 1)),

display\_labels=enc\_y.classes\_)

disp.plot()

plt.show()

### Kết quả

#### Kết quả độ chính xác

Hình 2: Kết quả độ chính xác mô hình iris

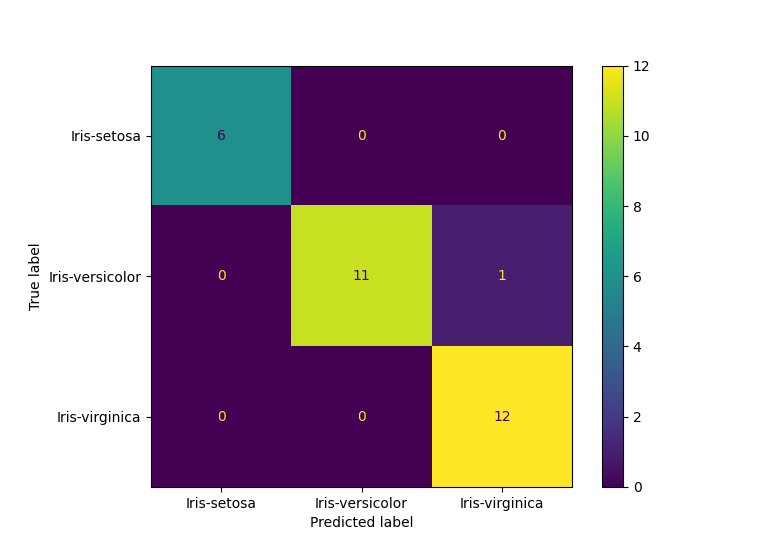
Biểu đồ cho thấy khi càng huấn luyện thì độ chính xác càng tăng.

#### Kết quả độ lỗi

Hình 3: Kết quả độ lỗi mô hình iris

Biểu đồ cho thấy khi càng huấn luyện thì lỗi càng giảm.

#### Mô tả chi tiết phần dự đoán test của Model



Hình 4: Mô tả chi tiết phần dự đoán test của mô hình iris

* Từ hình ảnh cho thấy:
  + Model dự đoán khá chính xác
  + Chỉ sai 1 phần tử ở Iris-versicolor trong khi model dự đoán là Iris-virginica

## Yêu cầu 2:

### Vấn đề

Xây dựng mạng nơ ron 2 tầng để giải bài toán XOR

### Giới thiệu phép XOR

Phép XOR (eXclusive OR) là 1 phép toán thực hiện trên bit với 2 đối số

Dưới đây là bảng chân trị của phép XOR

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A XOR B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

### Thư viện sử dụng

Để tiến hành giải quyết yêu cầu xây dựng mô hình phép XOR, trước tiên cần nhập những thư viện cần thiết của Python

import tensorflow as tf

### Khởi tạo các biến cần thiết

W1 = tf.Variable(tf.eye(2, 2))

b1 = tf.Variable(tf.zeros((2,)))

W2 = tf.Variable(tf.eye(2, 1))

b2 = tf.Variable(tf.zeros((1,)))

Các biến được khởi tạo:

* W1: tầng nơ ron đầu tiên, một ma trận gồm 2 hàng và 2 cột (2 bit, 2 nơ ron).

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

* b1: là bias của mỗi nơ ron trong W1.

|  |  |
| --- | --- |
| 0 | 0 |

* W2: tầng nơ ron thứ 2, một ma trận gồm 2 hàng và 1 cột (2 input, 1 nơ ron).

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

* b2: là bias của mỗi nơ ron trong W2.

|  |
| --- |
| 0 |

### Cài đặt hàm predict

def h(x: tf.Tensor) -> tf.Tensor:

return tf.nn.relu(tf.matmul(x, W1) + b1)

def predict(x: tf.Tensor) -> tf.Tensor:

return tf.sigmoid(tf.matmul(h(x), W2) + b2)

Hình 5: Đồ thị hàm dự đoán của model XOR

* h(x): kết quả dự đoán của tầng 1
  + Giá trị X: có 2 bit.
  + Thực hiện phép toán nhân X và W1. Kết quả của phép toán vừa rồi, ta đem cộng với b1.
  + Sử dụng hàm ReLU trả về kết quả.
* predict(x): kết quả dự đoán của mô hình
  + Thực hiện phép nhân h(x) với W2 sau đó cộng với b2
  + Sử dụng hàm Sigmoid trên kết quả vừa nhận và trả về kết quả dự đoán

### Cài đặt hàm lỗi

Thiết kế dựa trên công thức:

* : giá trị mong muốn của .
* : giá trị dự báo .

def loss(y\_true, y\_pred) -> tf.Tensor:

return tf.reduce\_mean(tf.math.squared\_difference(y\_true, y\_pred))

### Cài đặt hàm huấn luyện

def train(x\_train, y\_train, lr: float = 0.001) -> tf.Tensor:

with tf.GradientTape() as tape:

train\_loss = loss(y\_train, predict(x\_train))

dW1, db1, dW2, db2 = tape.gradient(train\_loss, [W1, b1, W2, b2])

W1.assign\_sub(lr\*dW1)

b1.assign\_sub(lr\*db1)

W2.assign\_sub(lr\*dW2)

b2.assign\_sub(lr\*db2)

return train\_loss

* Tính giá trị **train\_loss.**
* Sử dụng **GradientTape()** lưu trữ quá trình tính toán **train\_loss**.
* Lấy đạo hàm của W1, b1, W2 và b2 theo hàm lỗi thông qua **GradientTape().**
* Cập nhật lại giá trị W1, b1, W2 và b2 với tốc độ học.
* Trả về kết quả **train\_loss**.

### Vòng lặp dùng cập nhật và ghi nhận loss

with open("xor\_loss.txt", 'w') as f:

for epoch in range(100000):

train\_loss = float(train(X, Y, 0.01))

print(train\_loss)

f.write("{}\n".format(train\_loss))

* Mở file để lưu lại giá trị loss trong lúc huấn luyện
* Thực hiện lặp 100000 lần
* Tốc độ học là 0,01
* Mục đích: cập nhật W1, b1, W2 và b2
* Đóng file sau khi huấn luyện

### Kết quả

#### Kết quả độ lỗi

Hình 6: Kết quả độ lỗi của mô hình XOR 2 tầng

* Biểu đồ cho thấy khi càng huấn luyện thì lỗi càng giảm.
* Ở những lần huấn luyện đầu thì có thể thấy độ lỗi giảm khá đáng kể.
* Tuy nhiên, càng về sau độ lỗi gần như không có sự thay đổi lớn.

#### Kết quả dự đoán phép XOR

Kết quả thu được sau khi cho mô hình huấn luyện

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **A XOR B** | **OUTPUT MODEL** |
| 0 | 0 | 0 | 0.02948206 |
| 0 | 1 | 1 | 0.98458105 |
| 1 | 0 | 1 | 0.9845771 |
| 1 | 1 | 0 | 0.02948206 |

Từ kết quả trên có thể nhận xét là mô hình cho kết quả dự đoán rất gần với phép XOR. Tuy vẫn còn có lỗi ở kết quả dự đoán nhưng với độ lỗi rất nhỏ (xấp xỉ 0.02) so với kết quả đúng.

# TÀI LIỆU THAM KHẢO

[**https://www.tensorflow.org/api\_docs/python/tf/Tensor**](https://www.tensorflow.org/api_docs/python/tf/Tensor)

[**https://www.tensorflow.org/api\_docs/python/tf/GradientTape**](https://www.tensorflow.org/api_docs/python/tf/GradientTape)

# PHỤ LỤC

## Mã nguồn của yêu cầu 1

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

import pandas as pd

import tensorflow as tf

W = tf.Variable(tf.eye(4, 3))

b = tf.Variable(tf.zeros((3,)))

def predict(x: tf.Tensor) -> tf.Tensor:

return tf.nn.softmax(tf.matmul(x, W) + b)

def loss(y\_true, y\_pred) -> tf.Tensor:

return tf.reduce\_mean(-tf.reduce\_sum(y\_true\*tf.math.log(y\_pred), 1))

def train(x\_train, y\_train, lr: float = 0.001) -> None:

with tf.GradientTape() as tape:

train\_loss = loss(y\_train, predict(x\_train))

dW, db = tape.gradient(train\_loss, [W, b])

W.assign\_sub(lr\*dW)

b.assign\_sub(lr\*db)

def main() -> None:

col\_names = ["sepal length in cm",

"sepal width in cm",

"petal length in cm",

"petal width in cm",

"class"]

df = pd.read\_csv("iris.data",

names=col\_names)

df\_x = df.drop("class", axis=1)

enc\_y = LabelEncoder()

df\_y = enc\_y.fit\_transform(df["class"])

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df\_x,

df\_y,

test\_size=0.2,

random\_state=1911)

print(len(x\_train))

print(len(x\_test))

'''

x\_train = tf.Variable(x\_train , dtype=tf.float32)

x\_test = tf.Variable(x\_test , dtype=tf.float32)

y\_train = tf.one\_hot(y\_train, len(enc\_y.classes\_))

y\_test = tf.one\_hot(y\_test , len(enc\_y.classes\_))

f\_loss\_train = open("iris\_loss\_train.txt", 'w')

f\_loss\_test = open("iris\_loss\_test.txt" , 'w')

f\_accurary\_train = open("iris\_accurary\_train.txt", 'w')

f\_accurary\_test = open("iris\_accurary\_test.txt" , 'w')

for epoch in range(100000):

train(x\_train, y\_train, 0.01)

y\_true\_train = tf.argmax(y\_train , 1)

y\_true\_test = tf.argmax(y\_test , 1)

y\_pred\_train = predict(x\_train)

y\_pred\_test = predict(x\_test)

train\_loss = float(loss(y\_train, y\_pred\_train))

test\_loss = float(loss(y\_test , y\_pred\_test))

train\_accuracy = accuracy\_score(y\_true\_train , tf.argmax(y\_pred\_train, 1))

test\_accuracy = accuracy\_score(y\_true\_test , tf.argmax(y\_pred\_test , 1))

print("Epoch {}:".format(epoch))

print("\ttrain loss:\t{}\ttrain accuracy:\t{}" .format(train\_loss , train\_accuracy))

print("\ttest loss:\t{}\ttest accuracy:\t{}" .format(test\_loss , test\_accuracy))

f\_loss\_train.write("{}\n".format(train\_loss))

f\_loss\_test .write("{}\n".format(test\_loss))

f\_accurary\_train.write("{}\n".format(train\_accuracy))

f\_accurary\_test .write("{}\n".format(test\_accuracy))

f\_loss\_train.close()

f\_loss\_test .close()

f\_accurary\_train.close()

f\_accurary\_test .close()

disp = ConfusionMatrixDisplay(confusion\_matrix(tf.argmax(y\_test , 1),

tf.argmax(predict(x\_test), 1)),

display\_labels=enc\_y.classes\_)

disp.plot()

plt.show()

'''

if \_\_name\_\_ == "\_\_main\_\_":

main()

## Mã nguồn của yêu cầu 2

import tensorflow as tf

W1 = tf.Variable(tf.eye(2, 2))

b1 = tf.Variable(tf.zeros((2,)))

W2 = tf.Variable(tf.eye(2, 1))

b2 = tf.Variable(tf.zeros((1,)))

def h(x: tf.Tensor) -> tf.Tensor:

return tf.nn.relu(tf.matmul(x, W1) + b1)

def predict(x: tf.Tensor) -> tf.Tensor:

return tf.sigmoid(tf.matmul(h(x), W2) + b2)

def loss(y\_true, y\_pred) -> tf.Tensor:

return tf.reduce\_mean(tf.math.squared\_difference(y\_true, y\_pred))

def train(x\_train, y\_train, lr: float = 0.001) -> tf.Tensor:

with tf.GradientTape() as tape:

train\_loss = loss(y\_train, predict(x\_train))

dW1, db1, dW2, db2 = tape.gradient(train\_loss, [W1, b1, W2, b2])

W1.assign\_sub(lr\*dW1)

b1.assign\_sub(lr\*db1)

W2.assign\_sub(lr\*dW2)

b2.assign\_sub(lr\*db2)

return train\_loss

def main() -> None:

X = tf.Variable([[0.0, 0],

[0, 1],

[1, 0],

[1, 1]])

Y = tf.Variable([[0.0],

[1],

[1],

[0]])

with open("xor\_loss.txt", 'w') as f:

for epoch in range(100000):

train\_loss = float(train(X, Y, 0.01))

print(train\_loss)

f.write("{}\n".format(train\_loss))

print(W1)

print(b1)

print(W2)

print(b2)

print(predict(X))

if \_\_name\_\_ == "\_\_main\_\_":

main()