TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN**

**MÔN NHẬP MÔN HỌC MÁY**

**TÌM HIỂU VỀ THUẬT TOÁN LAN TRUYỀN NGƯỢC (BACKPROPAGATION)**

*Người hướng dẫn*: **GV LÊ ANH CƯỜNG**

*Người thực hiện*: **LÊ HOÀNG LONG – 518H0035**

Lớp **: 18H50205**

Khoá  **: 22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN MÔN NHẬP MÔN HỌC MÁY**

**TÌM HIỂU VỀ THUẬT TOÁN LAN TRUYỀN NGƯỢC (BACKPROPAGATION)**

Người hướng dẫn: **GV LÊ ANH CƯỜNG**

Người thực hiện: **LÊ HOÀNG LONG – 518H0035**

Lớp **: 18H50205**

Khoá  **: 22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

LỜI CẢM ƠN

Đầu tiên, em xin bày tỏ lòng biết ơn sâu sắc đến Trường Đại học Tôn Đức Thắng và khoa Công Nghệ Thông Tin đã tạo điều kiện để lớp học được mở cũng như cung cấp mọi điều kiện tốt nhất cho khóa học. Đặc biệt, em xin chân thành cảm ơn Giảng viên Lê Anh Cường đã quan tâm, giúp đỡ, hướng dẫn em mọi lúc để em có thể hoàn thành bài báo cáo trong thời gian qua.

Bài báo cáo này là sản phẩm của riêng em, nên không thể tránh khỏi những sai sót. Kính mong nhận được sự chỉ bảo đóng góp ý kiến của các thầy cô, cũng như mọi người để em ngày càng hoàn thiện thêm kiến thức.

Em xin chân thành cảm ơn!

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi / chúng tôi và được sự hướng dẫn của TS Nguyễn Văn A;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 25 tháng 4 năm 2021*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Lê Hoàng Long*

*Trần Văn C*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

Bài tập lớn này tập trung bày lý thuyết về thuật toán lan truyền ngược (backpropagation) để học tham số của mạng neural truyền thẳng (feed forward neural network) với giả thuyết ( Giả thuyết tầng đầu ra của mạng neural là 1 nút. Các hàm kích hoạt (activation function) là hàm sigmoid Hàm loss là Mean Square Error ) và viết chương trình để thực hiện thuật toán đó bằng Python.

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc70277471)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii](#_Toc70277472)

[TÓM TẮT iv](#_Toc70277473)

[MỤC LỤC 1](#_Toc70277474)

[CHƯƠNG 1 2](#_Toc70277475)

[LÝ THUYẾT VỀ THUẬT TOÁN LAN TRUYỀN NGƯỢC (BACKPROPAGATION) ĐỂ HỌC THAM SỐ CỦA MẠNG NEURAL TRUYỀN THẲNG (FEED FORWARD NEURAL NETWOR) 2](#_Toc70277476)

[1.1 Mạng Neural là gì và các đặc trưng của nó: 2](#_Toc70277477)

[1.2 Kiến trúc mạng Neural: 2](#_Toc70277478)

[1.3 Hàm Sigmoid: 3](#_Toc70277479)

[1.4 Logistic regression: 4](#_Toc70277480)

[1.5 Mean Square Error: 4](#_Toc70277481)

[1.6 Ký hiệu: 5](#_Toc70277482)

[1.7 Các bước tính toán forwar computation: 6](#_Toc70277483)

[1.8 Các bước tính toán gradient descent: 8](#_Toc70277484)

[CHƯƠNG 2 10](#_Toc70277485)

[VIẾT CHƯƠNG TRÌNH THEO THUẬT TOÁN Ở CHƯƠNG 1 10](#_Toc70277486)

CHƯƠNG 1

LÝ THUYẾT VỀ THUẬT TOÁN LAN TRUYỀN NGƯỢC (BACKPROPAGATION) ĐỂ HỌC THAM SỐ CỦA MẠNG NEURAL TRUYỀN THẲNG (FEED FORWARD NEURAL NETWOR)

Ở đây chúng ta sẽ xem xét trên giả thuyết:

* Tầng đầu ra của mạng neural là 1 nút
* Các hàm kích hoạt (activation function) là hàm sigmoid
* Hàm loss là Meam Square Error

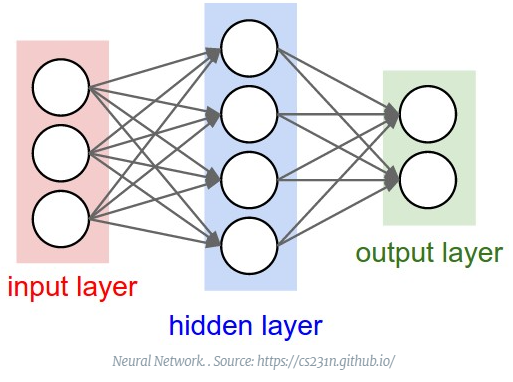
1.1 Mạng Neural là gì và các đặc trưng của nó:

Mạng thần kinh truyền thẳng (hay còn gọi là mạng nơ-ron truyền thẳng, tiếng Anh: feedforward neural network, viết tắt là FNN) là một mạng thần kinh nhân tạo trong đó kết nối giữa các nút không hình thành một chu trình. Mạng neuron truyền thẳng là loại mạng neuron nhân tạo đầu tiên và đơn giản nhất được phát minh ra. Trong mạng này, thông tin di chuyển chỉ một chiều - hướng đến - từ các nút đầu vào, thông qua các nút ẩn (nếu có) và đi đến các nút đầu ra. Không có chu trình (chu kỳ) hoặc vòng lặp trong mạng.

* Các đặc trưng cơ bản của mạng neuron:
  + Liên kết giữa các đơn vị. Xét tổng quát, mỗi liên kết được định nghĩa bởi một trọng số wjk cho ta biết hiệu ứng mà tín hiệu của đơn vị j có trên đơn vị k
  + Gồm một tập các đơn vị xử lý (các noron nhân tạo)
  + Một luật lan truyền quyết định cách tính tín hiệu ra của từng đơn vị từ đầu vào của nó.

1.2 Kiến trúc mạng Neural:

Mạng Neuron là sự kết hợp của các tầng bằng percepreon hay còn được gọi là perceptron đa tầng (multilayer perceptron)

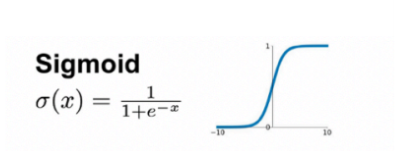


* Một mạng Neuron sẽ có 3 kiểu tầng:
  + Tầng vào (input layer): Là tầng bên trái cùng của mạng thể hiện cho các đầu vào của mạng.
  + Tầng ra (output layer): Là tầng bên phải cùng của mạng thể hiện cho các đầu ra của mạng.
  + Tầng ẩn (hidden layer): Là tầng nằm giữa tầng vào và tầng ra thể hiện cho việc suy luận logic của mạng.

Một mạng Neuron chỉ có 1 tầng vào và 1 tầng ra như có thể có nhiều tầng ẩn

* Mỗi node trong hidden layer và output layer:
  + Liên kết với tất cả các node ở layer trước đó và có hệ số w riêng
  + Mỗi node có 1 hệ số bias riêng
  + Đều diễn ra 2 bước tính: tính tổng linear và áp dụng activation function (ở đây ta dùng hàm sigmoid)

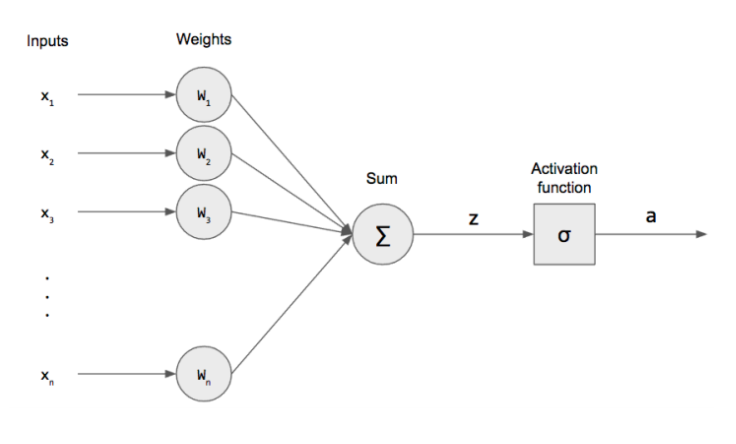
1.3 Hàm Sigmoid:



Hàm sigmoid là hàm số nếu đầu vào lớn hàm số sẽ cho đầu ra gần với giá trị 1. Còn đầu vào nhỏ (rất âm), hàm số sẽ cho đầu ra gần với giá trị 0

1.4 Logistic regression:

Logistic regression là mô hình Neural network đơn giản nhất chỉ với input layer và output layer.



Mô hình của logistic regression có dạng *y*^​=*σ*(*w*0+*w*1​∗*x*1​+*w*2​∗*x*2​). Ở đây ta có thể nhìn thấy 2 bước:

* Tính tổng linear: *z*=1∗*w*0​+*x*1​∗*w*1+*x*2​∗*w*2
* Áp dụng hàm sigmoid function: *y*^​=*σ(z)*

1.5 Mean Square Error:

Mean Square Error (Sai số bình phương) là hàm mất mát được sử dụng phổ biến nhất để hồi quy. Sự mất mát là dữ liệu giám sát trung bình của sự khác biệt bình phương giữa giá trị thực và giá trị dự đoán, hoặc viết nó dưới dạng công thức.



Trong đó:

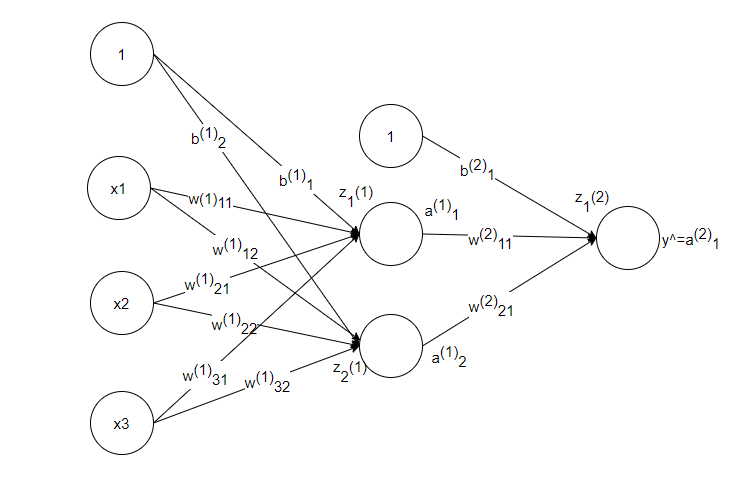
* Y^ là một vector của n dự đoán
* Y là vector của các giá trị quan sát được

1.6 Ký hiệu:



Ta ký hiệu:

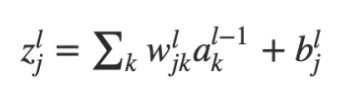
* + Số node trong hidden layer thứ j là l(j)
  + Ma trận W(l) kích thức l(l-1) \* ll là ma trận hệ số giữa layer (l-1) và layer l trong đó wjkl là hệ số kết nối từ node thứ j thuộc layer l-1 đến node thứ k của layer l.
  + blj là hệ số bias của các node j trong layer thứ l
  + zlj là giá trị các node trong layer l sau bước tính tổng linear
  + akl-1 là giá trị của node trong layer l sau khi áp dụng activation function



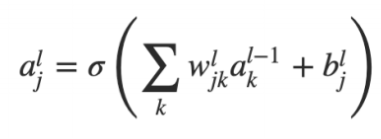
* Nhìn vào mô hình trên ta thấy:
  + Mô hình trên có 2 layer (số lượng layer không tính input layer)
  + Mô hình: 3-2-1, nghĩa là 3 node trong input layer, 2 node trong hidden layer, và output layer có 1 node
  + Input layer và hidden layer luôn thêm 1 node để tính bias cho layer sau. Nhưng không tính vào số lượng node trong layer
  + Ở mỗi node trong hidden layer và output layer đề thực hiện 2 bước: tính tổng linear và áp dụng activation function
  + Các hệ số bias ký hiệu trong hình

1.7 Các bước tính toán forwar computation:

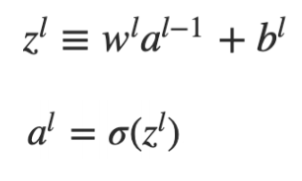
* Đầu tiên ta cần tính tổng linear của từng node:



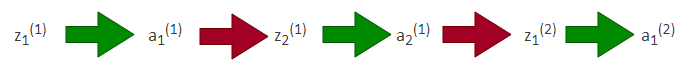
* Tiếp đến ta áp dụng activation function (ở đây là hàm sigmoid) với tham số là tổng linear



* Sau đó ta tính output

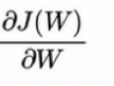


* Ví dụ Forward Computation dựa theo hình vẽ ở trên:
  + z1(1) = b1(1) + x1 \* w11(1) + x2 \* w21(1) + x3 \* w31(1)
  + a1(1) = *σ*(z1(1) ​)
  + z2(1) = b2(1) + x1 \* w12(1) + x2 \* w22(1) + x3 \* w32(1)
  + a2(1) = *σ*(z2(1) ​)
  + z1(2) = b1(2) + a1(1) \* w11(2) + a2(1) \* w21(2)
  + a1(2) = *σ*(z1(2) ​) = y^

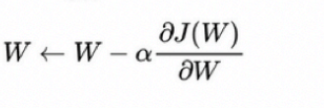


1.8 Các bước tính toán gradient descent:

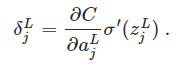
* Các bước tính toán của Gradient descent:
  + Bước khởi tạo: Bắt đầu bằng các hệ số nhỏ, hoặc hệ số ngẫu nhiên hoặc bằng 0: Quá trình này gọi là thiết lập vị trí ban đầu (starting point – điểm khởi đầu, hay initial weight – hệ số khởi tạo).
  + Bước 1: Tính đạo hàm từng phần của Loss function ở vị trí hiện tại đối với từng biến



* + Bước 2: Cập nhật hệ số mới theo hàm cập nhật sau :



* Bốn phương trình cơ bản của lan truyền ngược:
  + Phương trình cho lỗi trong lớp đầu ra δL:



Trong đó ∂C/ ∂aLj, chỉ đo lường mức độ thay đổi nhanh chóng của như một hàm của vị trí node j kích hoạt đầu ra. σ′(zLj)σ đo lường mức độ nhanh chóng của chức năng kích hoạt σ đang thay đổi của zLj

* + Phương trình lỗi δl về lỗi trong lớp tiếp theo δl + 1:



Ta có (wl+1)T là chuyển vị ma trận trọng lượng wl+1 cho (l+1) lớp. Giả sử chúng ta biết lỗi δl+1 tại l +1 lớp. Khi chúng ta áp dụng ma trận trọng lượng chuyển vị, (wl+1)T , chúng ta có thể nghĩ về điều này một cách trực quan khi chuyển lỗi về phía sau qua mạng, cho chúng ta một số loại thước đo lỗi ở đầu ra của l lớp.

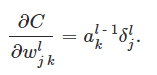
Sau đó chúng ta lấy Hadamard (Tích nguyên tố của hai vector) ⊙ σ′(zl). Điều này di chuyển lỗi trở lại thông qua chức năng kích hoạt trong lớp l, cho chúng ta lỗi δl trong đầu vào có trọng số cho lớp l.

* + Phương trình cho tốc độ thay đổi đối với bất kỳ sai lệch nào trong mạng:



Lỗi δlk là chính xác bằng với tốc độ thay đổi ∂C/ ∂blj

* + Phương trình cho tốc độ thay đổi với bất kỳ trọng số nào trong mạng:



* Thuật toán trong Gradient descent:
  + Đầu vào x: Đặt bias tương ứng a1 cho lớp đầu vào
  + Feedforward: Với mỗi l=2,3, … L tính toán zl= wlal-1 + bl và al=σ(zl)
  + Lỗi ở tầng ra δL= Tính toán vectơ δL=∇aC⊙σ′(zL)
  + Sao chép lại lỗi :  Đối với mỗi l = L - 1 , L - 2 , … , 2 tính toán δl= ( (wl+1)T δl+1)  ⊙ σ′(zl)
  + Kết quả là độ dốc của hàm được tính bởi

CHƯƠNG 2

VIẾT CHƯƠNG TRÌNH THEO THUẬT TOÁN Ở CHƯƠNG 1

*Tổng quan*: Chạy trương trình trên với một ví dụ cụ thể bằng python

* Đầu tiên ta cần import các thư viện

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

* Tiếp đến ta sẽ upload file:



from google.colab import files

uploaded = files.upload()

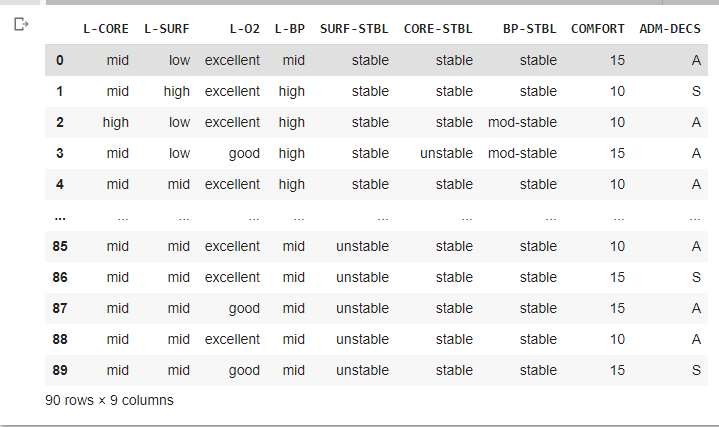
* Đọc data và đặt tên cho từng cột:

import pandas as pd

import io

dataframe = pd.read\_csv(io.StringIO(uploaded['post-operative.data'].decode('utf-8')), names=["L-CORE", "L-SURF", "L-O2", "L-BP", "SURF-STBL", "CORE-STBL", "BP-STBL", "COMFORT", "ADM-DECS"])

dataframe



* Kiểm tra missing data:

index=0

print('Các dòng bị missing data: ')

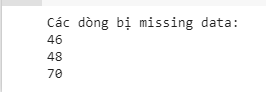
for row in dataframe.itertuples():

  for column in row:

    if column=='?':

      print(index)

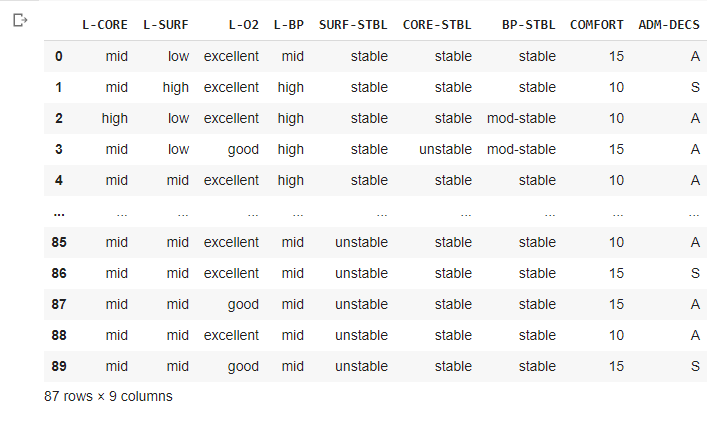
  index = index+1



* Xóa các dòng bị missing data:

dataframe1=dataframe1.drop(labels=[46,48,70], axis=0)

dataframe1



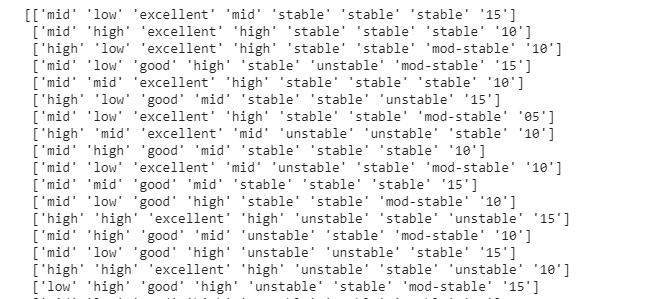
* Tách X, y:

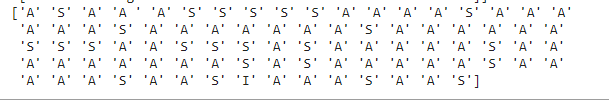
X = dataframe1.values[:,:-1]

y = dataframe1.values[:,-1]

print(X)

print(y)





* Encoder dữ liệu

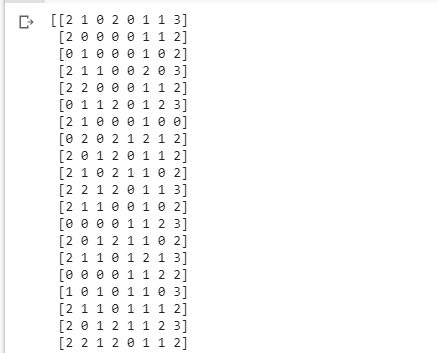
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

for i in range(X.shape[1]):

    X[:,i] = encoder.fit\_transform(X[:,i], )

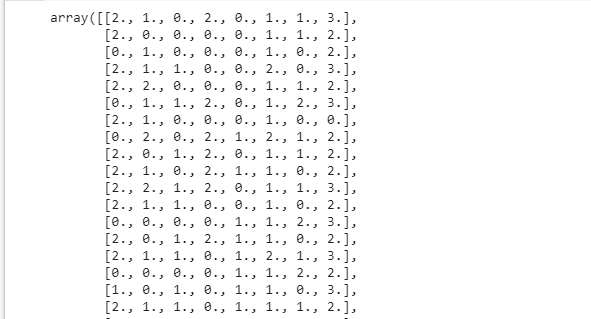
print(X)



* X với dữ liệu là float để tránh bị lỗi:

X=np.array(X, dtype=np.float32)

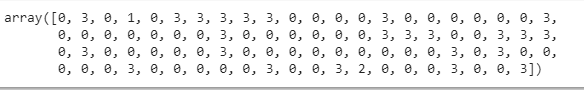
X



* Mã hóa y:

y=encoder.fit\_transform(y)

y



* Thực hiện thuật toán:

from math import exp

def sigmoid(z):

  return 1.0/(1.0 + np.exp(-z))

def sigmoid\_prime(z):

  return sigmoid(z)\*(1-sigmoid(z))

class NN:

  def \_\_init\_\_(self, l):

    self.weights = []

        # Set giá trị weight tương ứng cho input layer và hidden layer random

    for i in range(1, len(l) - 1):

      w = 2\*np.random.random((l[i-1] + 1, l[i] + 1)) -1

      self.weights.append(w)

    # Set giá trị weight tương ứng của out layer random

    w = 2\*np.random.random( (l[i] + 1, l[i+1])) - 1

    self.weights.append(w)

  def fit(self, X, y, text\_size=0.2, epochs=10000):

    # Một Epoch được tính là khi chúng ta đưa tất cả dữ liệu vào mạng neural network 1 lần. đem toàn bộ dữ liệu qua mạng một vài lần để tìm được kết quả tối ưu

    #Thêm giá trị bias vào input layer

    bias = np.atleast\_2d(np.ones(X.shape[0]))

    X = np.concatenate((bias.T, X), axis=1)

    for i in range(epochs):

      k = np.random.randint(X.shape[0])

      v = [X[k]]

      for l in range(len(self.weights)):

        dot\_value = np.dot(v[l], self.weights[l])

        activation = sigmoid(dot\_value)

        v.append(activation)

      #Tầng output

      error = y[k] - v[-1]

      delta = [error \* sigmoid\_prime(v[-1])]

      #Bắt đầu từ lớp thứ 2 tới lớp cuối cùng

      for l in range(len(v) - 2, 0, -1):

        delta.append(delta[-1].dot(self.weights[l].T)\*sigmoid\_prime(v[l]))

      # reverse [leve43(output)->leve42(hidden)]  => [leve42(hidden)->leve43(output)]

      delta.reverse()

      #Gradient descent:

        # 1. Nhân delta đầu ra của nó và kích hoạt đầu vào để lấy gradient của trọng lượng.

        # 2. Trừ một tỷ lệ (phần trăm) của gradient khỏi trọng lượng

      for i in range(len(self.weights)):

        layer = np.atleast\_2d(v[i])

        deltai = np.atleast\_2d(delta[i])

        self.weights[i] += text\_size \* layer.T.dot(deltai)

  def predict(self, x):

    b = np.concatenate((np.ones(1).T, np.array(x)))

    for l in range(0, len(self.weights)):

      b = sigmoid(np.dot(b, self.weights[l]))

    return b

nn = NN([8,50,1])

nn.fit(X, y)

y\_predict=[]

for e in X:

  y\_predict.append(nn.predict(e))

  print(e,nn.predict(e))

* Và đây là kết quả:

[2. 1. 0. 2. 0. 1. 1. 3.] [0.99569658]

[2. 0. 0. 0. 0. 1. 1. 2.] [0.57336408]

[0. 1. 0. 0. 0. 1. 0. 2.] [1.]

[2. 1. 1. 0. 0. 2. 0. 3.] [1.]

[2. 2. 0. 0. 0. 1. 1. 2.] [0.99999999]

[0. 1. 1. 2. 0. 1. 2. 3.] [1.]

[2. 1. 0. 0. 0. 1. 0. 0.] [1.]

[0. 2. 0. 2. 1. 2. 1. 2.] [1.]

[2. 0. 1. 2. 0. 1. 1. 2.] [1.]

[2. 1. 0. 2. 1. 1. 0. 2.] [0.99912755]

[2. 2. 1. 2. 0. 1. 1. 3.] [1.]

[2. 1. 1. 0. 0. 1. 0. 2.] [0.99998906]

[0. 0. 0. 0. 1. 1. 2. 3.] [0.51121274]

[2. 0. 1. 2. 1. 1. 0. 2.] [0.99999762]

[2. 1. 1. 0. 1. 2. 1. 3.] [1.]

[0. 0. 0. 0. 1. 1. 2. 2.] [0.96001512]

[1. 0. 1. 0. 1. 1. 0. 3.] [0.52807343]

[2. 1. 1. 0. 1. 1. 1. 2.] [0.94996428]

[2. 0. 1. 2. 1. 1. 2. 3.] [0.99607879]

[2. 2. 1. 2. 0. 1. 1. 2.] [1.]

[1. 0. 1. 2. 1. 1. 1. 3.] [0.99760052]

[1. 2. 0. 0. 1. 1. 2. 2.] [1.]

[2. 2. 1. 2. 1. 1. 2. 3.] [1.]

[2. 2. 1. 2. 1. 1. 1. 2.] [1.]

[0. 0. 1. 2. 0. 1. 0. 2.] [1.]

[1. 2. 1. 2. 1. 1. 1. 2.] [1.]

[0. 2. 1. 1. 0. 1. 0. 2.] [1.]

[1. 2. 0. 0. 0. 1. 0. 2.] [1.]

[2. 2. 0. 2. 0. 1. 2. 3.] [1.]

[2. 2. 1. 2. 1. 1. 2. 2.] [1.]

[2. 2. 1. 0. 1. 1. 1. 2.] [1.]

[1. 1. 1. 2. 1. 1. 2. 2.] [1.]

[2. 2. 0. 0. 1. 1. 0. 2.] [0.75340275]

[2. 1. 1. 2. 0. 1. 1. 2.] [1.]

[1. 2. 0. 0. 0. 1. 0. 2.] [1.]

[2. 2. 1. 2. 0. 1. 1. 2.] [1.]

[1. 2. 0. 2. 0. 1. 1. 2.] [1.]

[1. 1. 1. 2. 1. 1. 2. 2.] [1.]

[1. 1. 1. 2. 0. 1. 1. 1.] [1.]

[2. 2. 1. 0. 1. 1. 0. 2.] [1.]

[1. 1. 1. 2. 1. 1. 1. 2.] [1.]

[1. 2. 1. 2. 0. 1. 1. 3.] [1.]

[0. 0. 1. 0. 1. 1. 1. 3.] [0.5682467]

[2. 2. 1. 2. 0. 1. 1. 2.] [1.]

[1. 1. 0. 2. 0. 1. 1. 2.] [1.]

[1. 2. 1. 2. 1. 1. 1. 2.] [1.]

[2. 2. 0. 2. 1. 1. 1. 2.] [1.]

[2. 0. 1. 1. 1. 1. 1. 2.] [0.99802644]

[2. 0. 1. 2. 1. 0. 0. 2.] [0.50781158]

[1. 0. 0. 2. 1. 1. 1. 2.] [0.99999289]

[2. 1. 0. 0. 1. 1. 2. 2.] [0.60378804]

[2. 2. 1. 2. 1. 1. 0. 2.] [1.]

[0. 0. 0. 2. 1. 1. 0. 2.] [1.]

[2. 2. 1. 2. 1. 1. 1. 3.] [1.]

[0. 2. 1. 0. 0. 1. 2. 3.] [1.]

[2. 1. 1. 0. 1. 1. 0. 2.] [0.79254719]

[1. 1. 1. 0. 0. 1. 1. 2.] [1.]

[2. 0. 1. 2. 0. 1. 0. 2.] [1.]

[2. 0. 1. 2. 1. 1. 2. 2.] [1.]

[2. 1. 0. 0. 0. 1. 1. 2.] [0.77181766]

[2. 2. 1. 2. 0. 1. 2. 2.] [1.]

[2. 1. 0. 2. 0. 1. 2. 2.] [1.]

[0. 2. 0. 2. 1. 2. 2. 2.] [1.]

[2. 2. 1. 0. 0. 1. 1. 2.] [1.]

[2. 1. 0. 2. 1. 1. 1. 2.] [0.99999866]

[2. 2. 0. 2. 1. 1. 1. 2.] [1.]

[2. 2. 0. 0. 0. 1. 1. 2.] [0.99999999]

[2. 2. 0. 1. 0. 1. 1. 2.] [1.]

[1. 1. 0. 2. 0. 1. 1. 2.] [1.]

[2. 2. 0. 2. 0. 1. 0. 2.] [1.]

[2. 2. 0. 0. 0. 1. 1. 2.] [0.99999999]

[2. 1. 0. 0. 0. 1. 0. 2.] [0.68175585]

[1. 2. 1. 2. 0. 1. 2. 2.] [1.]

[2. 2. 0. 2. 0. 1. 0. 2.] [1.]

[2. 2. 0. 2. 0. 1. 2. 2.] [1.]

[2. 2. 0. 2. 1. 2. 1. 2.] [1.]

[2. 2. 1. 0. 0. 1. 1. 2.] [1.]

[2. 2. 0. 2. 0. 1. 1. 3.] [0.9999929]

[2. 2. 0. 2. 0. 1. 1. 2.] [1.]

[2. 1. 1. 2. 0. 1. 2. 2.] [1.]

[0. 2. 0. 2. 1. 1. 2. 0.] [1.]

[2. 2. 0. 2. 0. 1. 2. 2.] [1.]

[2. 2. 0. 2. 1. 1. 1. 2.] [1.]

[2. 2. 0. 2. 1. 1. 1. 3.] [0.99608783]

[2. 2. 1. 2. 1. 1. 1. 3.] [1.]

[2. 2. 0. 2. 1. 1. 1. 2.] [1.]

[2. 2. 1. 2. 1. 1. 1. 3.] [1.]

**TÀI LIỆU THAM KHẢO**

1. <http://neuralnetworksanddeeplearning.com/chap2.html>
2. StackOverflow
3. <https://viblo.asia/p/cach-xay-dung-mot-mang-no-ron-don-gian-chi-bang-python-1VgZv1nRKAw?fbclid=IwAR1tLsSV3ONM5-PZJnNgiNdQKTXi1BfsZ7WcufVg7GEbMmRIwQSOVU97p8g>
4. <https://www.python-course.eu/neural_networks_backpropagation.php>
5. <https://visualstudiomagazine.com/articles/2017/06/01/back-propagation.aspx>
6. <https://www.section.io/engineering-education/linear-regression-code/>
7. <https://www.programmersought.com/article/11595562932/>