Đại học quốc gia TP.HCM
Trường đại học công nghệ thông tin

Khoa khoa học máy tính

Lớp IT003.M21.KHTN

BÁO CÁO ĐỒ ÁN

XÂY DỰNG MODEL NHẬN DIỆN CHỮ SỐ VIẾT TAY SỬ DỤNG NEURAL NETWORK 3 LỚP

Học viên:
Lê Hoài Thương
21520474

Giảng viên:
Nguyễn Thanh Sơn

Mục lục

Danh sách hình vẽ

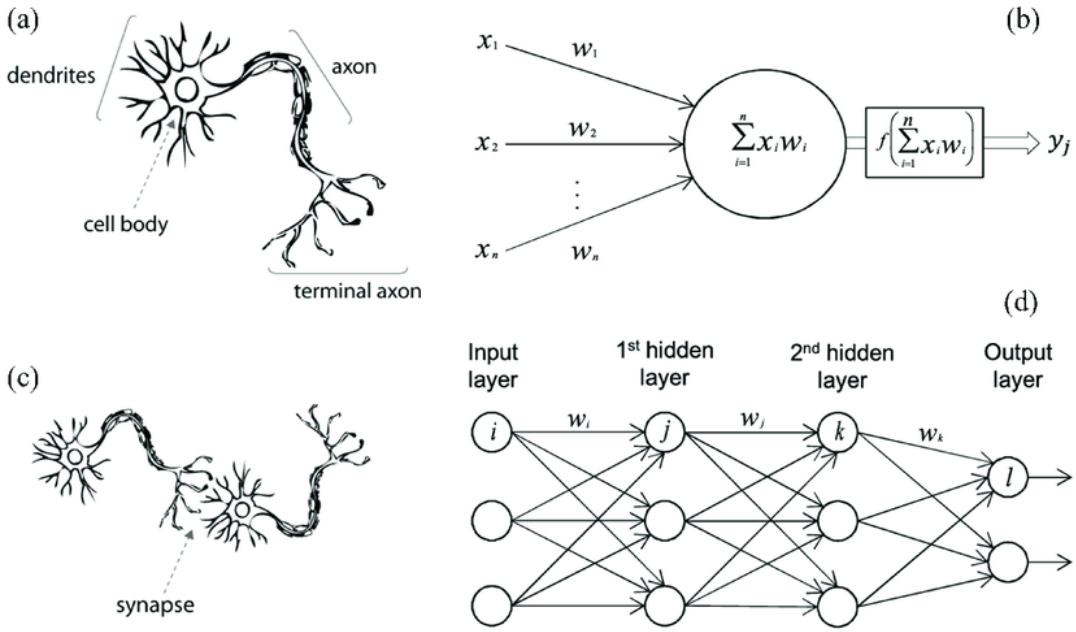
1 Giới thiệu bài toán

Xây dựng một mô hình neural network có nhiệm vụ nhận diện và phân biệt được 10 chữ số từ 0 đến 9 được viết tay. Dữ liệu đầu vào là hình ảnh 1 chữ số viết tay bất kì và đầu ra là chữ số đó.

Bài toán này được áp dụng nhiều trong các lĩnh vực như nhận dạng chữ số trên chi phiếu ngân hàng, mã số trên phong bì thư của dịch vụ bưu điện hay trên các biểu mẫu nói chung.

2 Cơ sở lí thuyết

2.1 Cấu trúc Neural Network



Hình 1: So sánh mạng nơ ron nhân tạo và mạng nơ ron sinh học

Mạng nơ ron nhân tạo (Artificial Neural Network - ANN) là một chuỗi các thuật toán được xây dựng dựa trên cấu trúc và hoạt động của mạng neural sinh học, nhằm tìm ra mối liên hệ sâu xa giữa dữ liệu đầu vào X và dữ liệu đầu ra

y. Neural Network đang trở thành 1 công cụ mạnh mẽ để giải quyết các bài toán khó như nhận dạng ảnh, nhận dạng giọng nói,.. một cách hiệu quả.

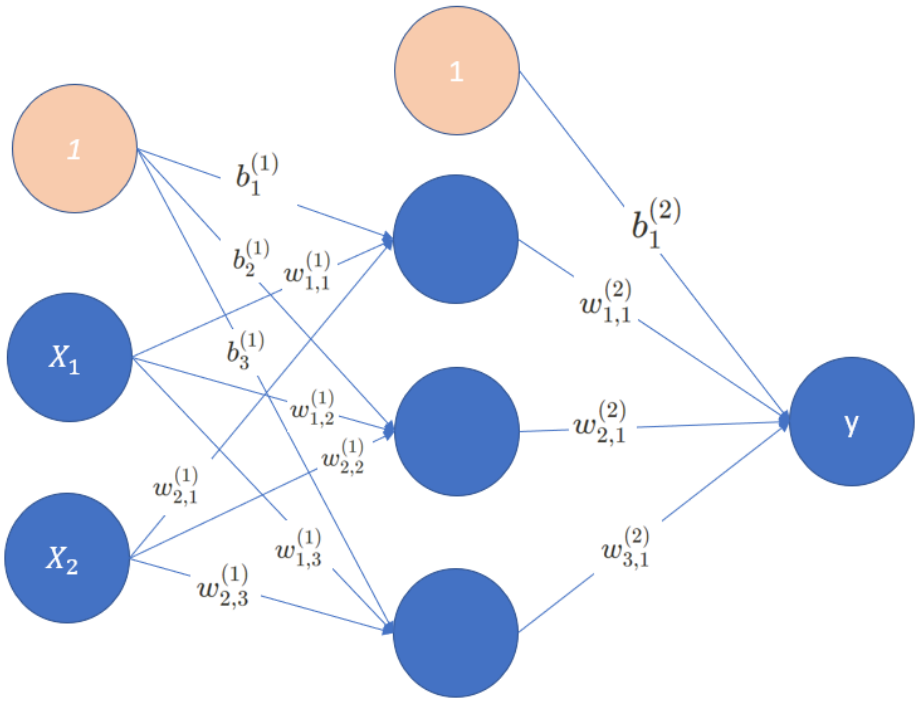
Một ANN gồm nhiều lớp, mỗi lớp được tạo thành bởi các nodes - nơi thực hiện các phép tính toán. Tại mỗi node, thông tin đầu vào (x) sẽ được nhân với một hệ số - hay trọng số (w), làm biến đổi thông tin đầu vào thành những đặc trưng mà thuật toán đang cố gắng học. Tổng của các tích xw này sẽ được đi qua activation function để xác định xem những thông tin này có ích cho kết quả cuối cùng của mạng hay không. Nếu được xác định là có ích, những thông tin này sẽ được activated - được đi tiếp vào các lớp sâu hơn của mạng.

Một lớp là một tập hợp các node. Đầu vào của các node trên 1 lớp chính là đầu ra của các node ở lớp kề trước (trừ lớp đầu tiên- lớp input). Một ANN gồm các lớp cơ bản như sau:

- Input layer: Các node chứa thông tin đầu vào của dữ liệu, số node ở layer này sẽ bằng số chiều của dữ liệu.
- Hidden layer: Các lớp chứa các node thực hiện những tính toán
- Output layer: Kết quả cuối cùng của thuật toán

Mỗi một node trong hidden layer và output layer đều có đặc điểm:

- Liên kết với tất cả các node ở layer trước đó với hệ số w riêng (được gọi là fully connected)
- Mỗi node có 1 hệ số bias riêng (hay chính là w_0)
- Diễn ra 2 bước tính toán: tính tổng linear và đi qua activation function



Hình 2: Một neural network đơn giản với 3 lớp: 2 - 3 - 1

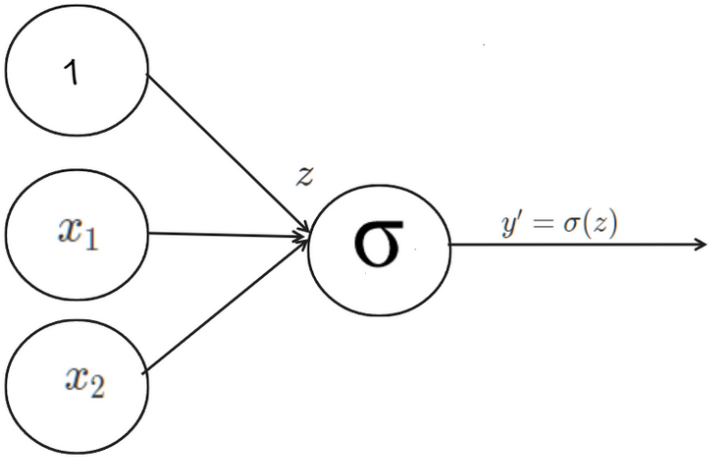
2.2 Logistic regression

Logistic regression là một mô hình neural network đơn giản với duy nhất 1 input layer và 1 output layer.

Quá trình tính toán gồm 2 bước:

1. Tính tổng linear: $z = 1 * w_0 + x_1 * w_1 + x_2 * w_2$
2. Áp dụng sigmoid function: $y' = \sigma(z)$

Mô hình của logistic regression được mô tả như sau:



Hình 3: Mô hình logistic regression

2.3 Kí hiệu

Số node trong layer thứ k là $l^{(k)}$. Ma trận $W^{(k)}$có kích thước $l^{(k)} * l^{(k-1)}$ là ma trận trọng số giữa layer thứ k và layer thứ $k+1$, trong đó $W_{i,j}^{(k)}$ là trọng số từ node thứ i của layer k nối với node thứ j ở layer $k+1$

Vector $b^{(k)}$ là vector chứa bias của lớp thứ k. $b_i^{(k)}$ là bias của node thứ i ở layer thứ k.

1. Tính tổng linear theo công thức: $z_i^k = \sum_{j=1}^{l^{(k-1)}} a_j^{(k-1)} * w_{i,j}^{(k)} + b_i^{(k)}$
2. Áp dụng dụng activation function: $a_i^{(l)} = \sigma(z_i^{(k)})$

Với vector z là vector kích thước $l^{(k)} * 1$ là giá trị của các node trong layer k sau khi áp dụng hàm activation function

Vector $a^{(k)}$ kích thước $l^{(k)} * 1$ là giá trị các node trong layer k sau khi áp dụng hàm activation function

2.4 Quá trình feed forward

Lần lượt thực hiện 2 bước tính toán qua mỗi lớp: tính tổng linear và đi qua activation function.

Công thức tính toán vector z và a ở layer thứ i:

- $z^{(i)} = a[i-1].W[i]$
- $a^{(i)} = \sigma(z^{(i)})$

$a^{(L)}$ chính là kết quả dự đoán của model, với L là số layer có trong model

2.5 Loss function

Ta dùng hàm mean square error(MSE) để tính hàm mất mát

$$\boxed{J = \frac{1}{2n}\sum_{i=0}^{n}(target_i - prediction_i)^2} \tag{1}$$

Với *target* là kết quả thực của tập đầu vào, và *prediction* là kết quả đầu ra của model sau khi feed forward và n là số điểm dữ liệu.

2.6 Back propagation

Để có thể huấn luyện model hiệu quả cho ra kết quả tốt thì ta phải tối ưu hàm Loss bằng thuật toán Scholastic Gradient Descent. Cần phải tính toán được đạo hàm của loss function theo từng ma trận trọng số $W^{(i)}$ và vector bias $b^{(i)}$. Tuy nhiên, việc tính toán các giá trị này trực tiếp là rất khó khăn vì hàm mất mát không phụ thuộc trực tiếp vào các hệ số.

Back propagation là thuật toán giúp tính toán các đạo hàm ngược từ layer cuối đến layer đầu.

Việc thực hiện tính toán chi tiết như sau:

1. Thực hiên feed forward , lưu lại các giá trị $A^{(i)}$ tại mỗi layer
2. Với output layer, tính: $E^{(L)} = \frac{\partial J}{\partial Z^{(L)}}$ với L là số layer trong model

3. Với $i = L-1, L-2, L-3, \ldots 1$, tính: $E^{(i)} = (W^{(i+1)}E^{(i)}).f'(z^{(i)})$
4. Từ đó suy ra: $\frac{\partial J}{\partial W^{(L)}} = A^{(L-1)} \otimes E^{(L)T}$
5. Cập nhật lại đạo hàm cho ma trận trọng số:
   - $\frac{\partial J}{\partial W^{(i)}} = A^{(i-1)} * E^{(i)T}$
   - $W+ = -\frac{\partial J}{\partial W^{(i)}} * learningrate$

   Tuy việc tính toán bằng back propagation khá đơn giản, quá trình này mất kha khá thời gian vì độ phức tạp lớn.

3 Xây dựng mô hình neural network nhận diện chữ số viết tay

3.1 Thiết lập thông số của mạng

Model là 1 ANN gồm 3 lớp:

- Input layer có kích thước là 28*28, là dữ liệu của ảnh đầu vào, mỗi node có giá trị từ 0 đến 255. Để có thể tính toán tốt hơn, ta nén dữ liệu xuống còn trong khoản [0,1] bằng cách chia cho 225
- 1 Hidden layer: gồm 86 nodes tính toán
- Ouput layer: gồm 10 node output, giá trị mỗi node trong khoảng $[0, 1]$ có ý nghĩa là xác suất điểm dữ liệu x là hình ảnh chữ số i.

3.2 Import các thư viện cần thiết

```
# import the necessary packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn import datasets
import matplotlib.pyplot as plt
```

```
import numpy as np
import random
from tqdm import tqdm
from sklearn.utils import shuffle
```

3.3 Tiền xử lí dữ liệu

Dùng bộ dữ liệu MNIST gồm hình ảnh chữ số viết tay có kích thước $28 * 28$, trong đó 60000 bức ảnh dùng để train model và 10000 mẫu khác dùng để đánh giá model.

Hình 4: Mẫu dữ liệu MNIST

Tải bộ dữ liệu MNIST về từ data set của keras. Nén giá trị của dữ liệu xuống trong khoảng $[0, 1]$. Dùng hàm **LabelBinarizer().fit_transform** để chuyển tập target thành các vector one-hot
Reshape lại dữ liệu đầu vào từ ma trận 28*28 thành 784*1

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = (X_train - X_train.min())/(X_train.max()-X_train.min())
X_test = (X_test - X_test.min())/(X_test.max()-X_test.min())
y_train= LabelBinarizer().fit_transform(y_train)
y_test = LabelBinarizer().fit_transform(y_test)

X_train = np.reshape(X_train,(X_train.shape[0],28*28))
```

```
X_test = np.reshape(X_test,(X_test.shape[0],28*28))
X_test.shape
```

3.4 Viết class Neural Network

Class neural network dùng để thể hiện model neural network. Đầu tiên, ta cần khởi tạo các thông số cho model: số lớp, số nodes trên mỗi lớp và learning rate.

Mỗi lớp được khởi tạo ngẫu nhiên một ma trận trọng số kích thước MxN với M là số node ở lớp hiện tại và N là số node ở lớp tiếp theo. Tuy nhiên, để có thể dễ dàng tính toán, ma trận trọng số được thêm 1 hàng và 1 cột cho bias (trừ output layer không cần bias). Như vậy, việc tính toán sẽ dễ dàng hơn vì bias lúc này được coi như 1 weight huấn luyện được mà không cần phải tính toán riêng.

```
class NeuralNetwork:
    def __init__(self, layers, learning_rate):
        # initialize the list of weights matrices, then store the
        # network architecture and learning rate
        self.W = []
        self.layers = layers
        self.learning_rate = learning_rate
        for i in (np.arange(0, len(layers) - 2)):
          w = np.random.randn(layers[i]+1,layers[i+1]+1)
          self.W.append(w/np.sqrt(layers[i]))
          w = np.random.randn(layers[-2]+1,layers[-1])
          self.W.append(w/np.sqrt(layers[-2]))
```

Dùng activation function là hàm sigmoid. Hàm sigmoid phù hợp với bài toán vì đạo hàm đẹp dễ dàng tính toán. Hơn nữa, hàm sigmoid nhận đầu vào là 1 số thực và trả về đầu ra là 1 số thực trong khoảng $[0, 1]$, phù hợp để tính các xác suất đầu ra.

Viết các hàm tính hàm sigmoid(x) và đạo hàm sigmoid(x) theo công thức:

- Hàm sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Công thức đạo hàm sigmoid : Đặt $z = \sigma(x)$ Ta có: $z' = z * (1 - z)$

```
def sigmoid(self,x):
    return 1.0/(1+np.exp(-x))

def sigmoid_deriv(self,z):
    return z*(1-z)
```

Model được huấn luyện thông qua phương thức .fit, với X_train là tập dữ liệu huấn luyện và y_train là tập label tương ứng. Epochs là số epochs mà model được huấn luyện.

Trong mỗi epoch sẽ diễn ra 2 quá trình chính: feed forward để lấy được và backpropagation để cập nhật lại weight và bias. Lưu lại loss trong quá trình train để có thể quan sát sự thay đổi của loss.

```
def fit(self, X_train,y_train,epochs=1000):
    #add a 1s column into X in order to treat bias
    #as a trainable param
    X_train = np.c_[X_train,np.ones((X_train.shape[0]))]
    #loop over epochs
    for epoch in (np.arange(0,epochs)):
      #shuffle training data
      X_train,y_train = shuffle(X_train,y_train)
      for(x,target) in zip(X_train,y_train):
        #feed forward, save a history of activaton
        A = self.feed_forward(x,target)
        #back propagation
        self.back_propagation(A,target)
      loss,acc = self.calculate_loss_acc(X_train,y_train,addBias = False)
      loss_history.append(loss) #save loss history
      #print loss and accuracy through training time
      print("Epoch {0:5d}: Loss: {1:15.9f}  Acc:
      {2:15.9f}%".format(epoch+1,loss,acc))
```

Quá trình feed forward tính toán lần lượt các $z^{(i)}$ và $a^{(i)}$ theo công thức: $z^{(i)} = a[i-1].W[i]$ $a^{(i)} = \sigma(z^{(i)})$ Các giá trị $a^{(i)}$ được lưu lại thành 1 vector A để dùng cho quá trình back propagation

```
    def feed_forward(self,x,y):
        A = [np.atleast_2d(x)]
        for layer in np.arange(0,len(self.W)):
          z = A[layer].dot(self.W[layer])
          a = self.sigmoid(z)
          A.append(a)
        return A
```

Quá trình tính toán và cập nhật các tham số weight và bias được thực hiện chi tiết như sau:

```
def back_propagation(self,A,y):
    error = A[-1]-y # calculate the error between target and model's result
    D = [error*self.sigmoid_deriv(A[-1])] #the maxtrix of gradient
    #calculate the gradient backward
    for layer in np.arange(len(A)-2,0,-1):
      delta = D[-1].dot(self.W[layer].T)
      delta = delta*self.sigmoid_deriv(A[layer])
      D.append(delta)
    #reverse the D matrix
    D = D[::-1]
    #update the W matrix
    for layer in np.arange(0,len(self.W)):
      self.W[layer] +=  - self.learning_rate*A[layer].T.dot(D[layer])
```

Model được huấn luyện dựa trên tập dữ liệu huấn luyện có sẵn. Hàm predict được viết để model có thể cho kết quả dự đoán dựa trên một tập dữ liệu đầu vào bất kì. Tham số x là dữ liệu đầu vào, tham số addBias để kiểm soát việc có thêm cột bias vào hay không.

```
def predict(self , X , addBias = True):
    p = np.atleast_2d(X)
    if addBias:
      p = np.c_[p,np.ones((p.shape[0]))]
    for layer in np.arange(0, len(self.W)):
```

```
            p = self.sigmoid(np.dot(p,self.W[layer]))
        return p
```

Sau khi hoàn thành huấn luyện model, hàm evaluate giúp đánh giá model trên tập test.

```
def evaluate(self,X_test,y_test):
    print("Evaluating model.....")
    loss,acc = self.calculate_loss_acc(X_test,y_test,addBias=True)
    print("Accuracy: {0:7f}%\n".format(acc))
    print("Loss: {0:7f}".format(loss))
```

3.5 Kết quả

Source code:

- Github
- Colab

Sau khi đánh giá model trên tập dữ liệu test, mô hình có độ chính xác khá cao - 97.600000%

4 Kết luận

Bài báo cáo đã trình bày về Neural Network cũng như áp dụng mô hình Neural Network 3 lớp đơn giản lên bài toán Nhận dạng chữ số viết tay. Từ đó, ta thấy được những ưu nhược điểm của mô hình trên:

- Ưu điểm:
  - Cho kết quả khả quan trên tập test (97.6%)
  - Các thông số có thể tự điều chỉnh tổng quá trình học, làm tăng tính hiệu quả cho model
  - Dễ cài đặt và được ứng dụng nhiều trong thực tế

  - Có thể thực hiện mô hình hóa và tính toán các mối quan hệ phức tạp giữa các dữ liệu.

- Nhược điểm:
  - Mất nhiều thời gian và tài nguyên để train model
  - Mô hình cho bài toán trên chỉ có 1 hidden layer dẫn đến việc tính toán những quan hệ phức tạp và sâu xa hơn không hiệu quả

5 Tài liệu tham khảo

- Multi-layer Perceptron và Backpropagation