

MACHINE LEARNING: BÀI THỰC HÀNH SỐ 5

MÔ HÌNH ANN (MULTI LAYER CLASSIFIER), CNN VÀ ỨNG DỤNG PCA

Trong phần thực hành này, chúng ta sẽ bắt đầu với việc xây dựng mô hình ANN bằng các bước như lý thuyết, chỉ với các công cụ cơ bản (numpy) và thực hiện trên tập dữ liệu nhân tạo. Sau đó chúng ta sẽ dùng PCA để giảm số chiều của một vài tập dữ liệu thực tế và áp dụng mô hình ANN của chúng ta để phân lớp.

Ví dụ 1. Ý tưởng ví dụ này tham khảo từ <https://cs231n.github.io/neural-networks-case-study/>, với dữ liệu nhân tạo để minh họa (toy example).

Trước hết chúng ta tạo một tập dữ liệu được chia vào 03 phân lớp khác nhau trong không gian $d=2$ chiều (để có thể hiển thị ra mặt phẳng). Các phân lớp sẽ được tạo sao cho không có hai phân lớp nào là tách được tuyến tính (linearly separable). Đoạn lệnh dưới đây để tạo dữ liệu theo ý tưởng như vậy:

```
# import the necessary libraries
from __future__ import division, print_function, unicode_literals
import math
import numpy as np
import matplotlib.pyplot as plt

N = 200 # number of points per class
d = 2 # dimensionality
C = 3 # number of classes
X = np.zeros((d, N*C)) # data matrix (each row = single example)
y = np.zeros(N*C, dtype='uint8') # class labels

for j in range(C):
    ix = range(N*j, N*(j+1))
    r = np.linspace(0.0, 1, N) # radius
    t = np.linspace(j*4, (j+1)*4, N) + np.random.randn(N)*0.2 # theta
    X[:, ix] = np.c_[r*np.sin(t), r*np.cos(t)].T
    y[ix] = j
```

Trong đoạn lệnh trên, ta đã tạo dữ liệu (là các điểm – 2 chiều) bằng cách tạo các điểm theo tọa độ cực: từ một tâm điểm, ta vừa tăng dần bán kính r , đồng thời tăng dần góc t theo chỉ số của điểm, dẫn đến các điểm

sẽ được phân bố xa dần tâm và tạo thành vệt xoáy. Ngoài ra ta bổ sung yếu tố ngẫu nhiên $np.random.randn(N)*0.2$ để tọa độ các điểm có sự xê dịch xung quanh vệt xoáy.

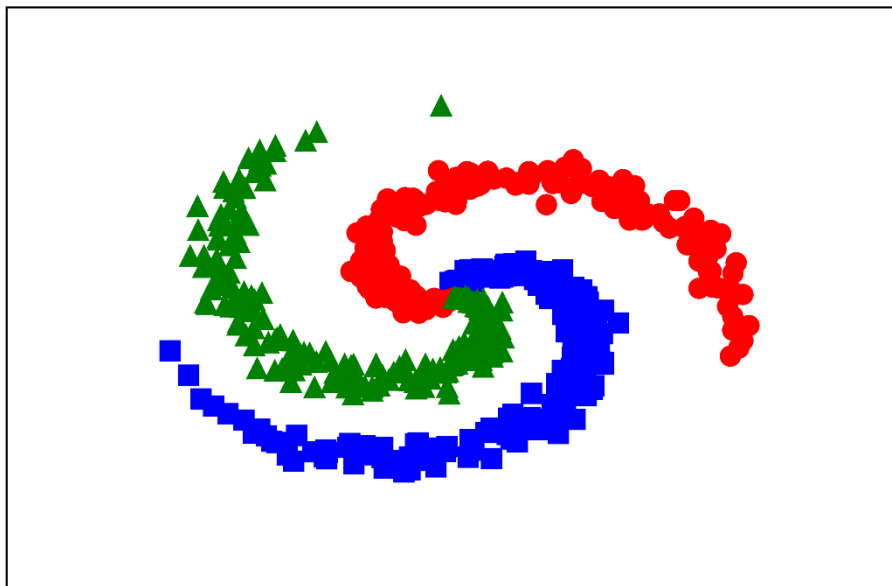
Đoạn chương trình dưới đây sẽ vẽ các điểm để ta có thể quan sát trực quan:

```
# Lets visualize the data:
plt.plot(X[0, :N], X[1, :N], 'bs', markersize = 7);
plt.plot(X[0, N:2*N], X[1, N:2*N], 'ro', markersize = 7);
plt.plot(X[0, 2*N:], X[1, 2*N:], 'g^', markersize = 7);

plt.xlim([-1.5, 1.5])
plt.ylim([-1.5, 1.5])
cur_axes = plt.gca()
cur_axes.axes.get_xaxis().set_ticks([])
cur_axes.axes.get_yaxis().set_ticks([])

plt.savefig('example_points.png', bbox_inches='tight', dpi = 600)
plt.show()
```

Khi thực hiện, đoạn lệnh sẽ cho tập điểm phân bố có dạng như hình dưới:



Do ở đây dữ liệu không tách được tuyến tính, nên chúng ta không thể sử dụng phương pháp hồi quy Softmax. Dưới đây chúng ta sử dụng một kiến trúc mạng ANN với 01 tầng ẩn bổ sung (tức số Layer $L = 2$), và ở tầng đầu ra (output) chúng ta sử dụng hàm kích hoạt là SoftMax, do đây là bài toán phân loại nhiều lớp.

Chúng ta bắt đầu với việc xây dựng các hàm cơ sở như Softmax

```
def softmax(V):  
    e_V = np.exp(V - np.max(V, axis = 0, keepdims = True))  
    Z = e_V / e_V.sum(axis = 0)  
    return Z
```

Tiếp theo chúng ta xây dựng hàm tổn thất, ở đây được tính theo công thức ứng với trường hợp phân loại nhiều lớp bằng softmax:

$$J \triangleq J(\mathbf{W}, \mathbf{b}; \mathbf{X}, \mathbf{Y}) = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(\hat{y}_{ij})$$

Đoạn chương trình xây dựng hàm tổn thất như sau:

```
# cost or loss function  
def cost(Y, Yhat):  
    return -np.sum(Y*np.log(Yhat))/Y.shape[1]
```

Tiếp theo, chúng ta chuyển đầu ra Y từ dạng số nguyên trong tập $\{1, 2, \dots, C\}$ - ứng với C classes – sang dạng onehot-coding, tức là vector C thành phần, nếu Y thuộc lớp thứ c thì tọa độ thứ c tương ứng là 1, các tọa độ khác là 0:

```
## One-hot coding  
from scipy import sparse  
def convert_labels(y, C = 3):  
    Y = sparse.coo_matrix((np.ones_like(y),  
        (y, np.arange(len(y)))), shape = (C, len(y))).toarray()  
    return Y
```

Tiếp theo ta khởi tạo các giá trị cho bộ trọng số và các số kiến trúc mạng. Ở đây ta sẽ bổ sung 01 tầng hidden với số units của tầng này là 100. Các bạn có thể tăng hoặc giảm con số này và quan sát kết quả. Ở tầng đầu ra, chúng ta cần vector 03 thành phần là xác suất để điểm tương ứng rơi vào một trong các lớp 0, 1, 2 (thay cho 1, 2, 3). Vậy số chiều của các tầng trong mạng ANN sẽ là: d0 = 2 (tầng input); d1 = 100 (tầng hidden) và d2 = 3 (tầng output). Đoạn lệnh khởi tạo như sau:

```
d0 = 2  
d1 = h = 100 # size of hidden layer
```

```

d2 = C = 3

# initialize parameters randomly
W1 = 0.01*np.random.randn(d0, d1)
b1 = np.zeros((d1, 1))
W2 = 0.01*np.random.randn(d1, d2)
b2 = np.zeros((d2, 1))

Y = convert_labels(y, C)
N = X.shape[1]

eta = 1 # learning rate

```

Tiếp theo là đoạn chương trình thực hiện thuật toán lặp theo phương pháp Gradient Descent

```

for i in range(10000):
    ## Feedforward

    Z1 = np.dot(W1.T, X) + b1
    A1 = np.maximum(Z1, 0)
    Z2 = np.dot(W2.T, A1) + b2
    Yhat = softmax(Z2)

    # print loss after each 1000 iterations
    if i % 1000 == 0:
        # compute the loss: average cross-entropy loss
        loss = cost(Y, Yhat)
        print("iter %d, loss: %f" % (i, loss))

    # backpropagation

    E2 = (Yhat - Y) / N
    dW2 = np.dot(A1, E2.T)
    db2 = np.sum(E2, axis = 1, keepdims = True)
    E1 = np.dot(W2, E2)
    E1[Z1 <= 0] = 0 # gradient of ReLU
    dW1 = np.dot(X, E1.T)
    db1 = np.sum(E1, axis = 1, keepdims = True)

```

```
# Gradient Descent update
```

```
W1 += -eta*dW1
```

```
b1 += -eta*db1
```

```
W2 += -eta*dW2
```

```
b2 += -eta*db2
```

Dưới đây chúng ta tính độ chính xác cho trường hợp áp dụng lên chính dữ liệu training

```
Z1 = np.dot(W1.T, X) + b1
```

```
A1 = np.maximum(Z1, 0)
```

```
Z2 = np.dot(W2.T, A1) + b2
```

```
predicted_class = np.argmax(Z2, axis=0)
```

```
acc = 100*np.mean(predicted_class == y)
```

```
print('training accuracy: %.2f %%' % (acc))
```

Đoạn chương trình dưới đây thực hiện việc vẽ dữ liệu cùng với đường phân cách để ta có thể quan sát kết quả một cách trực quan:

```
# Visualize results
```

```
xm = np.arange(-1.5, 1.5, 0.025)
```

```
xlen = len(xm)
```

```
ym = np.arange(-1.5, 1.5, 0.025)
```

```
ylen = len(ym)
```

```
xx, yy = np.meshgrid(xm, ym)
```

```
# xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

```
# xx.ravel(), yy.ravel()
```

```
print(np.ones((1, xx.size)).shape)
```

```
xx1 = xx.ravel().reshape(1, xx.size)
```

```
yy1 = yy.ravel().reshape(1, yy.size)
```

```
# print(xx.shape, yy.shape)
```

```
# XX = np.concatenate((np.ones((1, xx.size)), xx1, yy1), axis = 0)
```

```
X0 = np.vstack((xx1, yy1))
```

```
# print(X.shape)
```

```
Z1 = np.dot(W1.T, X0) + b1
```

```
A1 = np.maximum(Z1, 0)
```

```
Z2 = np.dot(W2.T, A1) + b2
```

```
# predicted class
```

```

Z = np.argmax(Z2, axis=0)

Z = Z.reshape(xx.shape)
CS = plt.contourf(xx, yy, Z, 200, cmap='jet', alpha = .1)

# Plot also the training points
# plt.scatter(X[:, 1], X[:, 2], c=Y, edgecolors='k', cmap=plt.cm.Paired)
# plt.xlabel('Sepal length')
# plt.ylabel('Sepal width')

# X = X.T
N = 100
print(N)

plt.plot(X[0, :N], X[1, :N], 'bs', markersize = 7);
plt.plot(X[0, N:2*N], X[1, N:2*N], 'g^', markersize = 7);
plt.plot(X[0, 2*N:], X[1, 2*N:], 'ro', markersize = 7);
# plt.axis('off')
plt.xlim([-1.5, 1.5])
plt.ylim([-1.5, 1.5])
cur_axes = plt.gca()
cur_axes.axes.get_xaxis().set_ticks([])
cur_axes.axes.get_yaxis().set_ticks([])

plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.xticks(())
plt.yticks(())
plt.title('#hidden units = %d, accuracy = %.2f %%' % (d1, acc))
# plt.axis('equal')
# display(X[1:, :], original_label)
fn = 'ex_res'+ str(d1) + '.png'
# plt.savefig(fn, bbox_inches='tight', dpi = 600)
plt.show()

```

Bài tập tự thực hành

Hãy sử dụng đoạn mã lệnh giảm số chiều bằng phương pháp phân tích thành phần chính đã học (tham khảo mã lệnh áp dụng cho dữ liệu ảnh đã được gửi):

Ví dụ 2. Áp dụng vào dữ liệu ảnh chữ số viết tay:

- giảm số chiều dữ liệu xuống còn 100 chiều (từ $28 \times 28 = 784$ chiều ban đầu) với tập dữ liệu 5000 ảnh bất kỳ.
- Áp dụng các phương pháp phân loại nhiều lớp: Multinomial Logistic Regression và ANN (đã có code) để phân loại, tỷ lệ train:test là 0.7:0.3.

Ví dụ 3. Giải nén tệp face_data.zip đính kèm sẽ có 165 ảnh các khuôn mặt của 15 người, mỗi người chụp ở 11 trạng thái khác nhau, dưới định dạng tệp png. Có một số thư viện Python hỗ trợ việc đọc ảnh và trả về

mảng số ứng với các điểm ảnh (xem lại khái niệm ảnh số trong phần đọc dữ liệu chữ số viết tay). Tên ảnh được ghép bởi prefix 'subject', chỉ số của người tương ứng ghi theo kiểu 01, 02 ... đến 15 và dấu chấm ".".

Tiếp theo là các trạng thái, gồm 11 trạng thái

['centerlight', 'glasses', 'happy', 'leftlight', 'noglasses', 'normal', 'rightlight', 'sad', 'sleepy', 'surprised', 'wink']

Cuối cùng là phần mở rộng trong tên tệp, .png.

Đoạn chương trình dưới đây cho phép sử dụng phương thức của thư viện OpenCV (cv2) để đọc tệp ảnh và lấy ra ma trận ứng với các điểm ảnh, sau đó duỗi thẳng thành 1 vector với số chiều $D = \text{height} * \text{width}$.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# path to the database - change it if needed
path = 'D:\\Teach_n_Train\\Machine Learning\\code\\yale\\yalefaces_data\\'

ids = range(1, 16) # 15 people
states = ['centerlight', 'glasses', 'happy', 'leftlight',
          'noglasses', 'normal', 'rightlight', 'sad',
          'sleepy', 'surprised', 'wink']
prefix = 'subject'
surfix = '.png' #file extension is png

# open one picture to get the image's size
fn = prefix + '01.' + states[0] + surfix
im = cv2.imread(path + fn, 0)

h = im.shape[0] # hight
w = im.shape[1] # width

D = h * w
N = len(states)*15
print(N, D, h, w)

X = np.zeros((D, N))

# collect all data
count = 0

# there are 15 people
for person_id in range(1, 16):
    for state in states:

        # get name of each image file
        fn = path + prefix + str(person_id).zfill(2) + '.' + state + surfix

        # open the file and read as grey image
        tmp = cv2.imread(fn, cv2.IMREAD_GRAYSCALE)

        # then add image to dataset X
        X[:, cnt] = tmp.reshape(D)
        count += 1
```

Áp dụng cho dữ liệu ảnh khuôn mặt (của 15 người, mỗi người có 11 trạng thái):

- c. Giảm số chiều dữ liệu xuống còn 500.
- d. Áp dụng các phương pháp phân loại nhiều lớp: Multinomial Logistic Regression và ANN (đã có code) để phân loại, tỷ lệ train:test là 0.7:0.3
- e. Coi toàn bộ 165 ảnh đầu vào là train, tìm 5 ảnh chân dung (tùy ý) sau đó đưa về cùng kích thước như trong ảnh dữ liệu ($H = 320$; $W = 243$), với phần chân dung lệch sang tay phải theo hướng người nhìn vào. Thử dụng mô hình đã huấn luyện, chạy test xem 5 ảnh dữ liệu mới sẽ thuộc nhóm nào.