"به نام یزدان پاک"

گزارش پروژه اول مبانی هوش محاسباتی

محمد چوپان۹۸۳۱۱۲۵

تاریخ گزارش : ۱۴۰۰/۰۸/۲۸

قدم اول: دریافت دیتاست

در ابتدا نیاز است دیتاست هایی را که در اختیار شما قرار دادیم، لود کنید. دیتاست ها به فرمت ".pkl" هستند که می‌توانید با استفاده از کتابخانه "Pickle" آن ها را بخوانید.

چهار فایل زیر دیتاست هایی هستند که برای انجام این پروژه (بجز بخش امتیازی) به آن‌ها نیاز دارید.

در این قسمت داده ها را با استفاده از کد قرار داده شده خوانده و به ۲ قسمت train و تست تقسیم بندی میکنیم.

قدم دوم: محاسبه خروجی (Feedforward)

همونطور که می‌دونید، برای محاسبه‌ی خروجی از روی ورودی در شبکه‌های عصبی، در هر لایه عملیات زیر انجام میشه:

$$a^{(L+1)} = \sigma(W^{(L+1)} \times a^{(L)} + b^{(L+1)})$$

در نتیجه، توی پیاده‌سازی شبکه عصبی، برای وزن‌های بین هر دو لایه، یک ماتریس k در n در نظر می‌گیریم که k، تعداد نورون‌های لایه‌ی بعدی و n، تعداد نورون‌های لایه‌ی فعلی. در نتیجه، هر سطر ماتریس W، وزن‌های مربوط به یک نورون خاص در لایه‌ی بعدی هستش. همچنین، برای بایاس‌های بین هر دو لایه هم، یک بردار جداگانه در نظر می‌گیریم که ابعادش برابر با تعداد نورون‌های لایه بعدی هستش.

سپس ۲۰۰ داده اول را برداشته و به صورت رندوم وزن ها را تولید میکنیم.

```
np.random.seed(1)
# number of first input layer perceptrons
n_x = 102
# perecptron number of first hidden layer
n_h_1 = 150
##perecptron number of second hidden layer
n_h_2 = 60
# last layer
n_y = 4
# make weights and bias for all layers
W1 = np.random.normal(size=(n_h_1, n_x))
b1 = np.zeros((n_h_1, 1))
W2 = np.random.normal(size=(n_h_2, n_h_1))
b2 = np.zeros((n_h_2, 1))
W3 = np.random.normal(size=(n_y, n_h_2))
b3 = np.zeros((n_y, 1))
```

تابعتی تحت عنوان sigmod برای محاسبه کردن تابع فعالیت تعریف میکنیم.

```
def sigmoid(x):
    ans = 1 / (1 + np.exp(-x))
    return ans
```

در نهایت در طول ۲۰۰ داده خود پیمایش میکنیم و برای هر داده خروجی را محاسبه کرده و اگر با برچسب داده شده یکی بود به شمارنده خود اضافه میکنیم.

```
counter = 0
# feeddorward for all datas
for i in range(len(minimize_train_set)):
    # label and fetures of one input
    reshape_train = minimize_train_set[i][0]
    reshape_train_label = minimize_train_set[i][1]
    # calculate precptron output with activiation sigmoid function
    S1 = sigmoid(W1 @ reshape_train + b1)
    S2 = sigmoid(W2 @ S1 + b2)
    S3 = sigmoid(W3 @ S2 + b3)
    # find  index of maximum  of output
    index = np.where(S3 == np.amax(S3))
    # find index of maximum of input label
    max_index = np.where(reshape_train_label == np.amax(reshape_train_label))
    if index == max_index:
        # if guess is correct counter++
        counter += 1
# print accuracy and time of execution
print("Accuracy is : " + str(counter / 200))
end_time = datetime.now()
print('Duration: {}'.format(end_time - start_time))
```

و دقت نهایی را محاسبه میکنیم.

که برابر است با:

```
Accuracy is : 0.29
Duration: 0:00:00.119024
```

قدم سوم: پیاده‌سازی Backpropagation

همونطور که می‌دونید، فرآیند یادگیری شبکه‌ی عصبی به معنی مینیمم کردن تابع Cost هستش:

$$Cost = \sum_{j=0}^{n_L-1} (a_j^{(L)} - y_j)^2$$

که این‌کار به کمک روش Gradient Descent انجام میشه که در اون با بدست آوردن مشتقات جزئی تابع Cost نسبت به تمامی پارامترها (یعنی همان گرادیان)، تغییرات مورد نظر بر روی پارامترها رو انجام می‌دیم:

$$(W,\ b) = (W,\ b) - \alpha \nabla Cost$$

در این قدم مراحل ۱ و ۲ را طی کرده و مانند قبل داده ها را خوانده و ماتریس وزن ها را تعریف میکنیم.

همچنان تعداد epoch ها و سایز batch

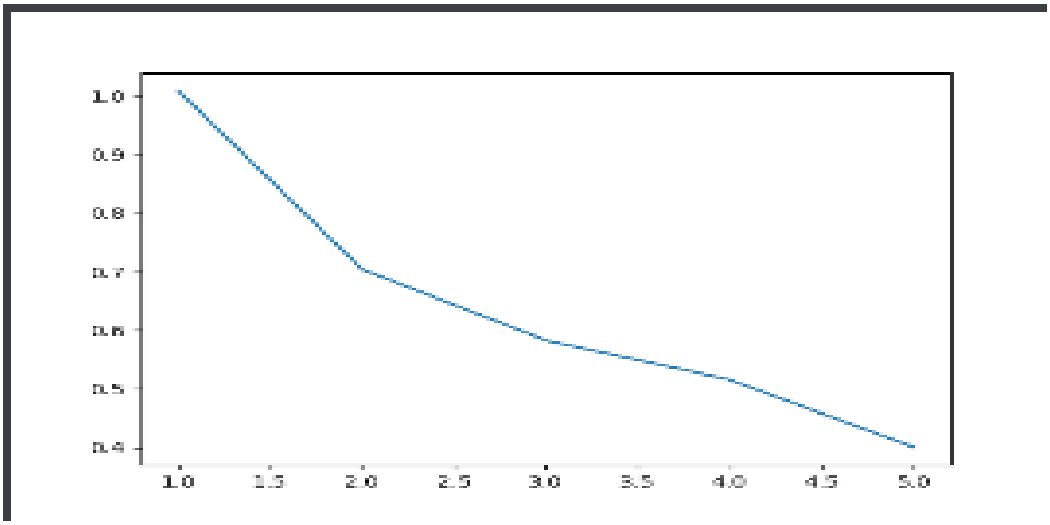
```
np.random.seed(1)
# defime epoch size
epoch = 5
#batch size
batch_size = 10
#each batch data number
batch_num = int(200 / 10)
#learning rate for calculating new W B
learning_rate = 0.3
np.random.seed(1)
#size of layer first to last
n_x = 102
n_h_1 = 150
n_h_2 = 60
n_y = 4
# # make weights and bias for all layers
W1 = np.random.randn(n_h_1, n_x)
b1 = np.zeros((n_h_1, 1))
W2 = np.random.randn(n_h_2, n_h_1)
b2 = np.zeros((n_h_2, 1))
W3 = np.random.randn(n_y, n_h_2)
b3 = np.zeros((n_y, 1))
```

سپس برای هر mini batch مانند قبل feed forward میزنیم با این تفاوت که در هر مرحله هزینه را محاسبه کرده و تابع گرادیان هزینه را نیز محاسبه میکنیم. در نهایت تغییرات را برای وزن ها عامل میکنیم.

و سپس روی داده ها یک feed forward برای محاسبه نتیجه و دقت نهایی میزنیم. و زمان و هزینه و دقت را گزارش میکنیم.

```
average cost : 3.212307798550221
Accuracy is : 0.735
Duration: 0:02:18.408122
```

نمودار هزینه ها :



کد ها :

```
for epoch_count in range(epoch):
    # shuffle
    #cost off each epoch
    total_cost = 0
    print("epoch_count  " + str(epoch_count + 1))
    #shuffle train set
    random.shuffle(train_set)
    minimize_train_set = train_set[:200]
    for batch_count in range(batch_size):
        print("batch count " + str(batch_count + 1))
        #initialize gradians for weights and bias off all layers
        grad_W1 = np.zeros((n_h_1, n_x))
        grad_W2 = np.zeros((n_h_2, n_h_1))
        grad_W3 = np.zeros((n_y, n_h_2))
        grad_b1 = np.zeros((n_h_1, 1))
        grad_b2 = np.zeros((n_h_2, 1))
        grad_b3 = np.zeros((n_y, 1))

        for i in range(batch_num):
            print("mini batch num is " + str(i + 1))
            #just feed forward
            reshape_train = minimize_train_set[batch_count * 20 + i][0]
            reshape_train_lables = minimize_train_set[batch_count * 20 + i][1]
            S0 = reshape_train
            S1 = sigmoid(W1 @ S0 + b1)
```

```
S0 = reshape_train
S1 = sigmoid(W1 @ S0 + b1)
S2 = sigmoid(W2 @ S1 + b2)
S3 = sigmoid(W3 @ S2 + b3)
temp_cost = 0
# calculate cost of each data
for s in range(len(S3)):
    temp_cost += pow(S3[s][0] - reshape_train_lables[s][0], 2)
total_cost += temp_cost
#calculate all gradians
#wight for W3
for j in range(grad_W3.shape[0]):
    for k in range(grad_W3.shape[1]):
        grad_W3[j, k] += 2 * (S3[j, 0] - reshape_train_lables[j, 0]) * S3[j, 0] * (1 - S3[j, 0]) * S2[k, 0]

# bias
for j in range(grad_b3.shape[0]):
    grad_b3[j, 0] += 2 * (S3[j, 0] - reshape_train_lables[j, 0]) * S3[j, 0] * (1 - S3[j, 0])

#  third layer

delta_3 = np.zeros((n_h_2, 1))
for k in range(n_h_2):
    for j in range(n_y):
        delta_3[k, 0] += 2 * (S3[j, 0] - reshape_train_lables[j, 0]) * S3[j, 0] * (1 - S3[j, 0]) * W3[j, k]
```

```
            W3 = W3 - (learning_rate * (grad_W3 / batch_size)
            W2 = W2 - (learning_rate * (grad_W2 / batch_size)
            W1 = W1 - (learning_rate * (grad_W1 / batch_size)
            b3 = b3 - (learning_rate * (grad_b3 / batch_size)
            b2 = b2 - (learning_rate * (grad_b2 / batch_size)
            b1 = b1 - (learning_rate * (grad_b1 / batch_size)

        # calculate costs
        costs.append(total_cost / 200)
        print("cost of this epoch is " + str(total_cost))
print("average cost : " + str(sum(costs)))
#plot costs per epoch diagram
epoch_list = [c + 1 for c in range(epoch)]
plt.plot(epoch_list, costs)
plt.show()
```

و در نهایت یک feed forward برای محاسبه دقت

```
for i in range(len(minimize_train_set)):
      reshape_train=minimize_train_set[i][0]
      reshape_train_label=minimize_train_set[i][1]
      S0 = reshape_train
      S1 = sigmoid(W1 @ S0 + b1)
      S2 = sigmoid(W2 @ S1 + b2)
      S3 = sigmoid(W3 @ S2 + b3)
      index = np.where(S3 == np.amax(S3))
      max_index = np.where(reshape_train_label == np.amax(reshape_train_label))
      if index == max_index:
          counter += 1
#show resluts
print("Accuracy is : " +str(counter / 200))
end_time = datetime.now()
print('Duration: {}'.format(end_time - start_time))
```

قدم چهارم: Vectorization

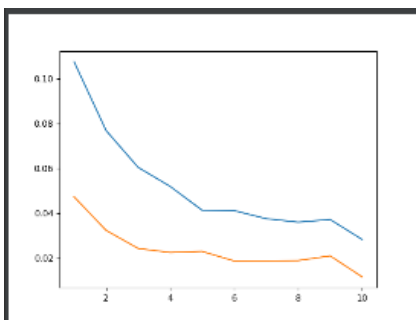
دلیل اینکه تا اینجا فقط با ۲۰۰ داده اول دیتاست کار کردیم اینه که زمان اجرای پیاده‌سازی فعلی‌مون خیلی زیاد هستش. برای اینکه این مشکل رو برطرف کنیم، از مفهومی تحت عنوان **Vectorization** استفاده می‌کنیم. این مفهوم به این معنیه که به جای اینکه بیایم و توی کار با دیتامون، for بزنیم روی درایه‌ها، سعی کنیم عملیاتی که می‌خوایم انجام بدیم رو به شکل عملیات ماتریسی (ضرب و جمع ماتریسی و برداری، ضرب داخلی، تراتهاده کردن و اعمال توابع روی تک‌تک عناصر ماتریس‌ها) پیاده‌سازی کنیم.

تنها تفاوت این قدم با قدم قبلی در زمان اجرای آن است که ما محاسبه گرادیان ها را به شکل زیر تغییر داده ایم.

```
grad_W3 += (2 * (S3 - reshape_train_lables) * S3 * (1 - S3)) @ np.transpose(S2)
# bias
grad_b3 += 2 * (S3 - reshape_train_lables) * S3 * (1 - S3)
# third layer
# activation
delta_3 = np.zeros((n_h_2, 1))
delta_3 += np.transpose(W3) @ (2 * (S3 - reshape_train_lables) * (S3 * (1 - S3)))
# weight
grad_W2 += (S2 * (1 - S2) * delta_3) @ np.transpose(S1)
# bias
grad_b2 += delta_3 * S2 * (1 - S2)
# second layer
# activation
delta_2 = np.zeros((n_h_1, 1))
delta_2 += np.transpose(W2) @ (delta_3 * S2 * (1 - S2))
# weight
grad_W1 += (delta_2 * S1 * (1 - S1)) @ np.transpose(reshape_train)
# bias
grad_b1 += delta_2 * S1 * (1 - S1)
```

خروجی ما :

```
average cost epochs : 0.5183532585822295
average cost all of epoch : 0.23847256043383916
Accuracy is : 0.8
Duration: 0:00:02.454640
```



قدم پنجم: تست کردن مدل

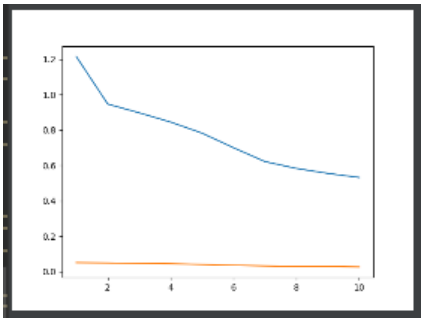
حالا که الگوریتم رو تا حد خوبی بهینه کردیم، می‌تونیم بریم و روی کل ۱۹۶۲ داده مجموعه Train، فرآیند یادگیری رو انجام بدیم. مقدار batch_size رو برابر با ۱۰، ضریب یادگیری رو برابر با ۱ و همچنین تعداد epoch ها رو ۱۰ در نظر بگیرید.

در این مرحله تنها تعداد داده ها متفاوت است و در نهایت آموزش هم بروی داده های تست و هم داده های اصلی feed forward میزنیم.

```
for i in range(len(test_set)):
    reshape_test = test_set[i][0]
    reshape_test_label = test_set[i][1]
    S0 = reshape_test
    S1 = sigmoid(W1 @ S0 + b1)
    S2 = sigmoid(W2 @ S1 + b2)
    S3 = sigmoid(W3 @ S2 + b3)
    temp_cost = 0
    for s in range(len(S3)):
        temp_cost += pow(S3[s][0] - reshape_test_label[s][0], 2)
    total_cost += temp_cost

    index = np.where(S3 == np.amax(S3))
    max_index = np.where(reshape_test_label == np.amax(reshape_test_label)
    if index == max_index:
        counter += 1
print("Accuracy is for real values  : " + str(counter / 662))
costs.append(total_cost / 662)
print("cost of this time  " + str(total_cost))
```

نتایج:



```
average cost all of epoch : 0.37439444126775606
Accuracy is : 0.7390417940876657
Accuracy is for real values  : 0.6102719033232629
cost of this time  428.1042566197772
average cost : 0.6466831670993612
Duration: 0:00:16.191201
```