به نام خدا

تمرین پنجم مباحث ویژه

ساخت یک کلاسیفایر با استفاده از الگوریتم شبکه عصبی دو لایه

استاد دولتشاهی

پریسا مبارک

40211415006

شرح تمرین:

میخواهیم با استفاده از الگوریتم شبکه عصبی دو لایه، یک مدل بر روی دیتاستی که در این تمرین داریم بسازیم ودقت را هم برای مجوعه آموزشی و هم برای مجموعه تست گزارش کنیم.

الگوریتم شبکه عصبی دو لایه :

در این روش یک لایه ورودی، یک لایه پنهان و یک لایه خروجی داریم.(اگر سه لایه بود، دو لایه پنهان داشتیم و ...). هرلایه، وزن ها و بایاس مخصوص خود را دارد و چون شبکه عصبی یک شبکه فولی کانکتد هست برای هر لایه یک ماتریس وزن و یک بردار بایاس داریم.

بنابراین در این مثال دو ماتریس وزن و دو بردار بایاس داریم.یکی برای لایه پنهان و دیگری برای لایه لایه خروجی.

به طور کلی در این الگوریتم دو فرایند محاسبات پیش رو ومحاسبات پس رو، بصورت تکراری صورت میگیرد تا شبکه عصبی به یک نقطه بهینه همگرا شود.

محاسبات پیش رو:

ورودی های شبکه را از لایه ورودی به سمت لایه خروجی انتقال میدهیم. در هرلایه از وزن ها و بایاس های آن لایه برای محاسبه خروجی استفاده میکنیم. خروجی هرلایه به عنوان ورودی لایه بعد استفاده میشود. برای لایه پنهان داریم: a1= f(w1.x +b1) و برای لایه خروجی داریم: w2.a1 +b2. در نهایت نیز یک تابع سافت مکس رو خروجی لایه خروجی اعمال میکنیم.

محاسبات پس رو:

بعد از اینکه بصورت کامل از لایه ورودی به لایه خروجی رسیدیم و خروجی لایه خروجی بدست امد، محاسبات پس رو را انجام میدهیم. به این صورت که خطای خروجی شبکه را محاسبه میکنیم(مثلا تفاوت بین خروجی و برچسب واقعی). این خطا را به سمت عقب از لایه خروجی به لایه های قبلی انتشار میدهیم. در هرلایه وزن ها وبایاس هارا به روز میکنیم تا خطای شبکه کاهش یابد.

پس هدف اصلی الگوریتم شبکه عصبی دولایه، تنظیم مقادیر وزن ها و بایاس های شبکه عصبی است تا زمانی که مدل به خوبی آموزش ببیند و دقت بیشینه شود.

ابتدا همانند کدهای قبل دیتاست را میخوانیم و ورودی ها و خروجی های آموزش و تست را مشخص میکنیم. طبق توضیحات بالا لایه پنهان از طریق یک تابع فعال ساز مثلا سیگموید(یا relu یا tanh) بدست می آید. و در خروجی لایه خروجی نیز یک تابع سافت مکس اعمال میکنیم. بنابراین توابع سیگمویدو سافت مکس را همانند کد زیر تعریف میکنیم. همچنین تابع فرضیه برای لایه پنهان را تعریف میکنیم یعنی f(w1.x +b1). به طور خلاصه برای لایه پنهان ماتریس وزن های لایه پنهان را در ورودی ضرب کرده و به علاوه بردار بایاس میکنیم و سپس سیگموید را روی حاصل آن اعمال میکنیم. توابع سیگموید ، سافت مکس و تابع فرضیه لایه پنهان به شکل زیر تعریف شده اند:

```python
# تابع فعال‌سازی سیگموید
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# تابع فرضیه با یک لایه پنهان
def h_neural_network(X, w1, b1):
    z1 = X.dot(w1.T) + b1
    a1 = sigmoid(z1)
    return a1

# تابع softmax
def softmax(z):
    exp_z = np.exp(z)
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)
```

همچنین پارامترهای لازم را مقداردهی میکنیم. ماتریس های وزن ها و بردارهای بایاس را بصورت تصادفی انتخاب میکنیم:

```python
# مقداردهی اولیه پارامترها برای شبکه عصبی دو لایه
num_hidden_units = 50  # تعداد نورون‌ها در لایه پنهان
num_classes = len(np.unique(y_train))  # تعداد کلاس‌ها
w1 = np.random.randn(num_hidden_units, xn_train.shape[1])  # وزن‌ها برای لایه پنهان
w2 = np.random.randn(num_classes, num_hidden_units)  # وزن‌ها برای لایه خروجی
b1 = np.random.randn(1, num_hidden_units)  # بایاس برای لایه پنهان
b2 = np.random.randn(1, num_classes)  # بایاس برای لایه خروجی
```

سپس تابع آپدیت را تعریف میکنیم. برای اینکار ابتدا محاسبات پیش رو انجام میشود یعنی از لایه ورودی به لایه خروجی میرسیم. به این صورت که ابتدا لایه پنهان را به دست می آوریم(a1). سپس لایه خروجی را بدست می آوریم(z2). و در لایه خروجی نیز تابع سافت مکس اعمال میکنیم(a2). تا اینجا محاسبات پیش رو را انجام دادیم.

حال نوبت ب محاسبات پس رو یعنی آپدیت پارامترها(از لایه خروجی به سمت عقب) میباشد. ابتدا پارامترهای لایه خروجی یعنی w2 و b2 را آپدیت میکنیم. به این صورت که برای هرکلاس خطای

خروجی را محاسبه و از آن برای آپدیت وزن ها و بایاس ها استفاده میکنیم. سپس نوبت به آپدیت پارامترهای لایه پنهان میرسد.ابتدا خطای لایه خروجی را محاسبه میکنیم و سپس بااستفاده از این خطا و مشتق تابع فعال سازی لایه پنهان، خطای لایه پنهان را محاسبه میکنیم. در نهایت از این خطای لایه پنهان برای آپدیت وزن ها وبایاس های لایه پنهان استفاده میکنیم:

```
# تابع به‌روزرسانی پارامترها
def update_step_neural_network():
    global w1, b1, w2, b2
    a1 = h_neural_network(xn_train, w1, b1)
    z2 = a1.dot(w2.T) + b2
    a2 = softmax(z2)
    m = len(y_train)
    # به‌روزرسانی پارامترهای لایه خروجی
    for k in range(num_classes):
        gradient = a1.T.dot(a2[:, k] - (y_train == k + 1))
        w2[k,:] -= (alpha / m) * gradient
        gradient_b2 = np.sum(a2[:, k] - (y_train == k + 1))
        b2[k] -= (alpha / m) * gradient_b2
    # به‌روزرسانی پارامترهای لایه پنهان
    delta2 = (a2 - (y_train[:, np.newaxis] == np.arange(1, num_classes+1)))
    gradient1 = delta2.dot(w2) * (a1 * (1 - a1))
    w1 -= (alpha / m) * np.dot(gradient1.T, xn_train)
    b1 -= (alpha / m) * np.sum(gradient1, axis=0)
```

در ادمه آموزش را برای 700مرحله تکرار میکنیم تا مدل به خوبی آموزش ببیند و در هرمرحله نیز دقت آموزشی و تست را محاسبه میکنیم :

```
# آموزش شبکه عصبی
alpha = 0.01
accuracy_list_train = []
accuracy_list_test = []
for i in range(700):
    update_step_neural_network()
    # محاسبه دقت مدل برای داده‌های آموزش
    a1_train = h_neural_network(xn_train, w1, b1)
    z2_train = a1_train.dot(w2.T) + b2
    y_probs_train = softmax(z2_train)
    y_pred_train = np.argmax(y_probs_train, axis=1) + 1
    accuracy_train = np.mean(y_pred_train == y_train)
    accuracy_list_train.append(accuracy_train)
    # محاسبه دقت مدل برای داده‌های تست
    a1_test = h_neural_network(xn_test, w1, b1)
    z2_test = a1_test.dot(w2.T) + b2
    y_probs_test = softmax(z2_test)
    y_pred_test = np.argmax(y_probs_test, axis=1) + 1
    accuracy_test = np.mean(y_pred_test == y_test)
    accuracy_list_test.append(accuracy_test)
```

درنهایت دستور چاپ دقت نهایی و نمایش نمودار دقت برای آموزش و تست را مینویسیم:

```
# چاپ دقت نهایی
print("Final Training Accuracy:", accuracy_list_train[-1])
print("Final Test Accuracy:", accuracy_list_test[-1])

# نمودار دقت مدل
plt.plot(accuracy_list_train, label='Training Accuracy')
plt.plot(accuracy_list_test, label='Test Accuracy')
plt.xlabel('iteration')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

خروجی های این کد:

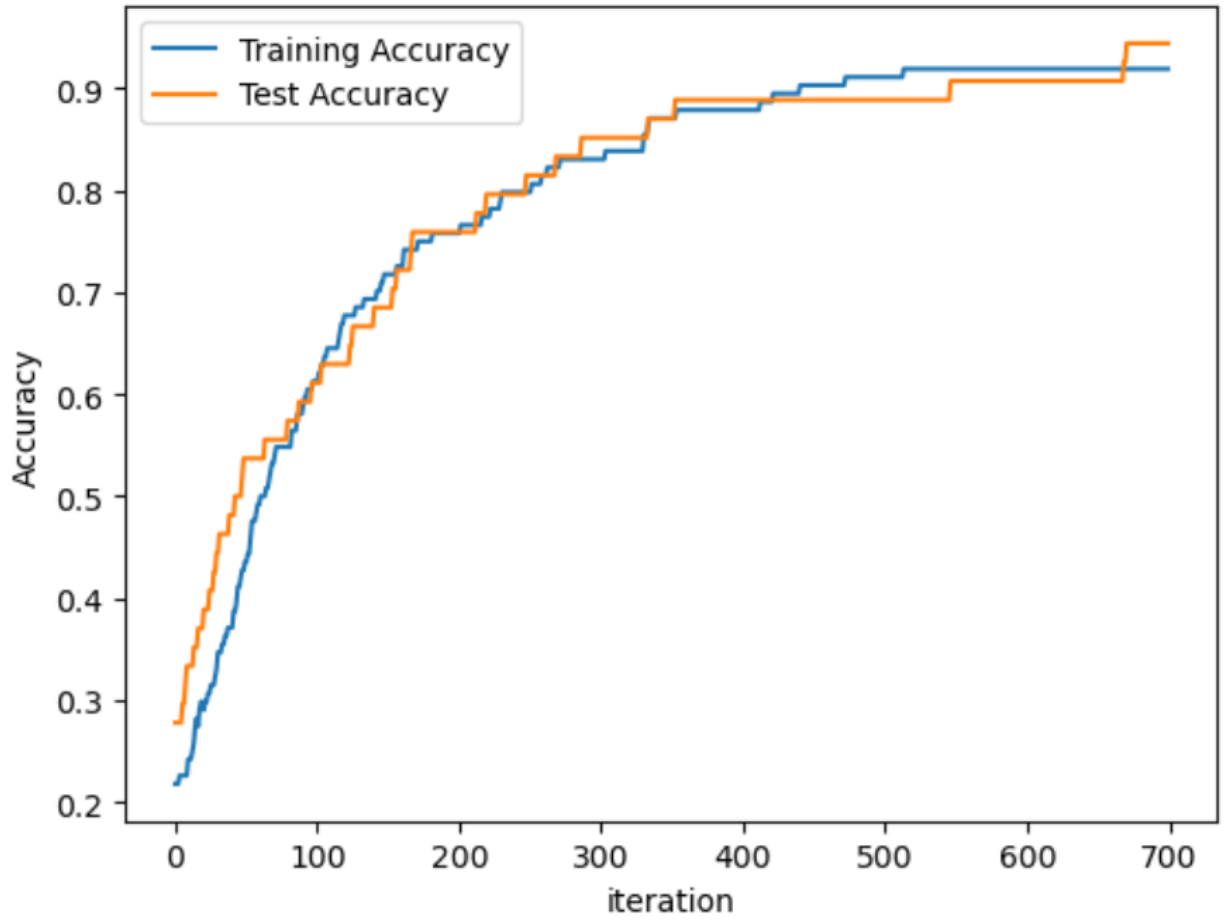
چاپ داده های ورودی:

```
dataset:
         0     1     2     3    4     5     6     7     8      9     10    11  \
0    14.23  1.71  2.43  15.6  127  2.80  3.06  0.28  2.29   5.64  1.04  3.92
1    13.20  1.78  2.14  11.2  100  2.65  2.76  0.26  1.28   4.38  1.05  3.40
2    13.16  2.36  2.67  18.6  101  2.80  3.24  0.30  2.81   5.68  1.03  3.17
3    14.37  1.95  2.50  16.8  113  3.85  3.49  0.24  2.18   7.80  0.86  3.45
4    13.24  2.59  2.87  21.0  118  2.80  2.69  0.39  1.82   4.32  1.04  2.93
..     ...   ...   ...   ...  ...   ...   ...   ...   ...    ...   ...   ...
173  13.71  5.65  2.45  20.5   95  1.68  0.61  0.52  1.06   7.70  0.64  1.74
174  13.40  3.91  2.48  23.0  102  1.80  0.75  0.43  1.41   7.30  0.70  1.56
175  13.27  4.28  2.26  20.0  120  1.59  0.69  0.43  1.35  10.20  0.59  1.56
176  13.17  2.59  2.37  20.0  120  1.65  0.68  0.53  1.46   9.30  0.60  1.62
177  14.13  4.10  2.74  24.5   96  2.05  0.76  0.56  1.35   9.20  0.61  1.60

       12 13
0    1065  A
1    1050  A
2    1185  A
3    1480  A
4     735  A
..    ... ..
173   740  C
174   750  C
175   835  C
176   840  C
177   560  C
```

چاپ دقت نهایی و نمایش نمودارها برای آموزش و تست:

```
[178 rows x 14 columns]
Final Training Accuracy: 0.9193548387096774
Final Test Accuracy: 0.9444444444444444
```



طبق خروجی ها، مشاهده میکنیم که مدل به خوبی آموزش دیده است.