به نام خدا

مهدی فیروزبخت

۴۰۰۱۳۱۰۲۷

تمرین سوم شبکه عصبی

-۱

یکی از کار هایی که میشود با این ساختار انجام داد کاهش بعد داده ها میباشد بدین صورت که میتوانیم در اموزش شبکه ، به ازاء هر ورودی اعمال شده به شبکه برای مثال بعد داده ی ورودی به شبکه ۱۰۰۰ می باشد. اگر سایز نقشه برابر ۱۰۰ در نظر گرفته شود ، بعد در لایه اول ۱۰۰۰ ویژگی به عنوان ورودی در نظر گرفته میشود و سپس در لایه دوم یک نقشه ۲ بعدی از ۱۰۰ ویژگی یعنی ۱۰ * ۱۰ میباشد که این داده ها با یک ضریبی به این نقشه نگاشت میشود. در هر مرحله یکی از این ویژگی ها به عنوان ویژگی برنده انتخاب میشود و در مجموع پس از گذشت چندین تکرار تعداد زیادی ویژگی یا نرون به عنوان برنده انتخاب میشوند. میتوانیم فاصله بردار وزن تمام نرون های لایه خروجی(map) با بردار ورودی را محاسبه کرده و این فواصل را به عنوان ویژگی جدید در نظر بگیریم که در این جا چون ۱۰۰ فاصله وجود دارد پس بردار ورودی جدید به جای این که ۱۰۰۰ ویژگی داشته باشد ، ۱۰۰ ویژگی برایش در نظر میگیریم و از این ویژگی ها میتوانیم به عنوان ورودی برای یک مدل دسته بندی استفاده کنیم . این ویژگی ها به صورت فشرده تر هستند و حاوی اطلاعات بهتری نسبت به ویژگی های قبلی میباشند .

-۲

نرون های مرده نرون هایی هستند که در تمام مدت آموزش در هیچ مرحله ای به پیروزی نمیرسند و هیچ وقت به عنوان نرون انتخاب نمیشوند. نرون های مرده سبب کوچک شدن نقشه میشوند. وقتی نقشه کوچک باشد هر نرون میتواند برای دسته های مختلف آموزش ببیند و این کار باعث بروز اشتباه در محاسبات شده و کار شبکه را خراب میکند.

اقای کوهونن اعلام کرد برای برطرف کردن مشکل نرون های مرده در ابتدای اموزش این شعاع همسایگی را بزرگ در نظر بگیریم به گونه ای که تمام نرون های لایه خروجی را (بردار وزن ان ها) ، برای نرون برنده در ابتدای کار به عنوان همسایه در نظر بگیرد و این گونه وزن همه ی نرون ها در ابتدای کار شروع به اصلاح شدن میکند و فاز ordering به درستی شکل میگیرد .سپس میتوان با

تابعی بر حسب زمان شعاع همسایگی را کاهش داد جوری که فقط همسایگان مجاور نرون برنده دچار اصلاح وزن شوند . روش دیگری که میتوان برای جلوگیری از ایجاد نرون های مرده در شبکه استفاده کنیم این است که در هنگام وزن دهی اولیه به بردار وزن نرون های لایه خروجی ، برخی از داده های ورودی را به عنوان رندوم انتخاب کرده و از مقادیر ویژگی های ان ها به عنوان وزن های نرون لایه خروجی در نظر بگیریم این کار باعث میشود تا نرون های لایه خروجی خیلی از الگوی توزیع داده های ورودی فاصله نداشته باشد و نرون مرده خیلی کمتر احتمال شکل گیری دارد .

-۳

```
#Load data
X = load_dataset('D:/HomeWork/NN/HW3/UCI HAR Dataset/train/X_train.csv')
y = load_dataset('D:/HomeWork/NN/HW3/UCI HAR Dataset/train/y_train.csv')
X_test = load_dataset('D:/HomeWork/NN/HW3/UCI HAR Dataset/test/X_test.csv')
y_test = load_dataset('D:/HomeWork/NN/HW3/UCI HAR Dataset/test/y_test.csv')
y = y-1
y_test = y_test-1

#split data to train and validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.10, random_state=42)
```

در این قسمت کد ، داده های آموزش را به عنوان ورودی گرفته و ۱۰٪ آن را برای داده های اعتبارسنجی انتخاب میکنیم. داده های تست و آزمون نیز وارد کرده ایم.

-۴

در ابتدا کتابخانه های مورد نظر را وارد میکنیم. سپس داده های لازم را لود میکنیم. داده هایی که در قسمت سوم نشان داده شده است. سپس داده های آموزش برای داده های هدف را به صورت categorical تبدیل کرده ایم.

```
x_tr = pd.DataFrame(new_X_train)
y_tr = pd.DataFrame(y_train)
Y_train = to_categorical(y_tr,num_classes=6)

X_valid = pd.DataFrame(X_valid)
y_valid = pd.DataFrame(y_valid)
y_valid = to_categorical(y_valid,num_classes=6)
```

سپس یک کالبک برای TensorBoard قرار داده ایم. در مرحله بعد مدل خود را ایجاد کرده ایم.

```
model  = keras.models.Sequential([
                                  keras.Input(shape =(561,),name = 'input_layer'),
                                  keras.layers.Dense(units = 64 , activation='relu', name= 'hidden_layer1'),
                                  keras.layers.Dense(units = 128 , activation='relu', name= 'hidden_layer2'),
                                  keras.layers.Dense(6,activation='softmax',name='output_layer')
])
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.009),loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(x_tr,Y_train,epochs= 20, validation_data =(X_valid, y_valid) , callbacks=[tb_callBack])
```

در این مدل با ۲۰ تکرار مدل را fit میکنیم و مدل های مختلف بررسی میکنیم:

۱ لایه مخفی ، ۱۲۸ نرون : دقت ۰/۹۸۳۵ برای آموزش و ۰/۹۲۷۴ برای تست

۱ لایه مخفی ، ۶۴ نرون : دقت ۰/۹۷ برای آموزش و ۰/۹۰ برای تست

۱ لایه مخفی ، ۳۲ نرون : دقت ۰/۹۷ برای آموزش و ۰/۹۳ برای تست

۱ لایه مخفی ، ۱۶ نرون: دقت ۰/۹۸۵۲ برای آموزش و ۰/۹۵ برای تست

۲ لایه مخفی ، ۱۶ نرون و ۵۰ نرون : دقت ۰/۹۸۰۳ برای آموزش و ۰/۹۵۲۸ برای تست

۲ لایه مخفی ، ۳۲ نرون و ۵۰ نرون : دقت ۰/۹۸۱۶ برای آموزش و ۰/۹۲۵۴ برای تست

۲ لایه مخفی ، ۶۴ نرون و ۱۲۸ نرون : دقت ۰/۹۷۸۲ برای آموزش و ۰/۹۴۸۱ برای تست

با بررسی تنسوربرد و همگرایی این مدل ها ، مدل آخر ، ۲ لایه مخفی با ۶۴ نرون و ۱۲۸ نرون در ، و دقت ذکر شده بهترین حالت را ایجاد کرده است.

-۵

ابتدا کتابخانه های مورد نیاز را وارد میکنیم. سپس داده های اصلی را وارد میکنیم. سپس کلاس SOM را ایجاد میکنیم. در قسمت train یک لوپ برای تعداد تکرار ها ایجاد میکنیم. سپس در این لوپ یک لوپ دیگر برای تکرار روی داده ها ایجاد میکنم. سپس در این قسمت ابتدا برای هر داده نرون برنده را تشخیص میدهیم. سپس مقدار آن را ذخیره میکنیم و همسایه های آن را و همچنین وزن های آن را اپدیت میکنیم. سپس با تکرار روی داده ها میانگین وزن نرون های برنده را تشخیص میدهیم. همچنین تعداد نرون های مرده را محاسبه میکنیم. برای محاسبه ضریب یادگیری و شعاع از فرمول (1 - t / T) استفاده میکنیم.

```
    for t in range(T):
        prev_map = self.map.copy()
        # Shuffle X in every iteration
        shuffle_ind = np.random.randint(0, X.shape[0], X.shape[0])
        avg = (0 , 0)
        scores = np.zeros(shape=(self.map.shape[0], self.map.shape[1], 1))

        for i in range(len(X)):
            x = X[shuffle_ind[i], :]
            # Neuron with most compatibilty with x
            winner = self.find_winner(x)
            scores[winner[0], winner[1]] += 1
            neuos[winner[0] , winner[1]] += to_categorical(y[i],num_classes=6)[0]
            avg = tuple(map(sum, zip(avg, winner)))
            avg = tuple(ti/2 for ti in avg)
            # Get all neurons in the neighborhood of winner
            NS = self.get_NS(winner)
            # Update weights of all neurons in the neighborhood of winner
            self.update_weights(x, winner, NS, len(X))


        dead_neurons.append(self.find_dead_neurons(scores))
        winners.append(np.linalg.norm(avg[0] - avg[1]))
        # Update learning rate and neighborhood radius (linear decay)
        self.lr = self.lr0 * (1 - t / T)
        self.R = self.R0 * (1 - t / T)


        Js.append(np.linalg.norm(prev_map - self.map))

        if t%10 == 0 or t == T-1:
            print(f"Iteration: {t}, Loss: {Js[-1]:.4f}, lr: {self.lr:.4f}, R: {self.R:.4f}")

        if Js[-1] < error_threshold:
            print("MIN CHANGE")
            break

    return Js , winners , dead_neurons , neuos
```

همچنین برای محاسبه get_umatrix میانگین همسایه های هر داده را محاسبه کرده و برای داده قرار میدهیم. باقی کد های قرار داده شده از روی کد آموزش داده شده در ویدیو کلاس است.

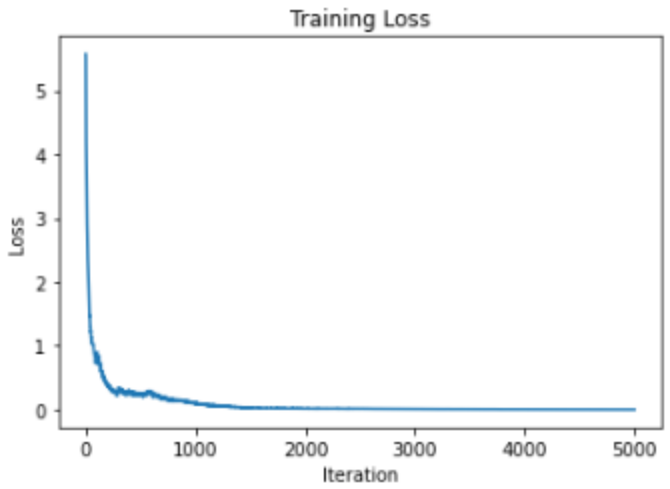
```
def get_umatrix(self):

    umatrix = np.zeros((self.map.shape[0], self.map.shape[1]))



    for map_x in range(self.map.shape[0]):
        for map_y in range(self.map.shape[1]):

            n_dist = 0
            total_dist = 0

            # Circular neighborhood
            for ri in range(-1, 2):
                for rj in range(-1, 2):
                    if 0 <= map_x + ri < self.map.shape[0] and 0 <= map_y + rj < self.map.shape[1]:
                        dist = np.linalg.norm(self.map[map_x, map_y] - self.map[map_x + ri, map_y + rj])
                        if dist != 0:
                            total_dist += dist
                            n_dist += 1

            avg_dist = total_dist / n_dist
            umatrix[map_x, map_y] = avg_dist

    return umatrix
```

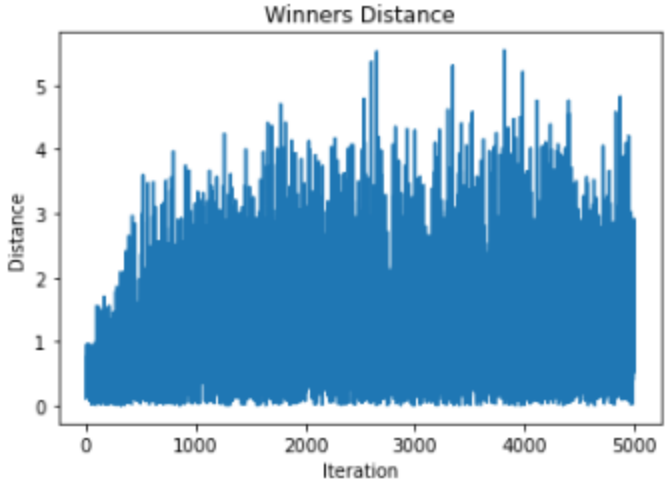
مقادیر بهینه پیدا شده برای این محاسبه به شرح زیر است:

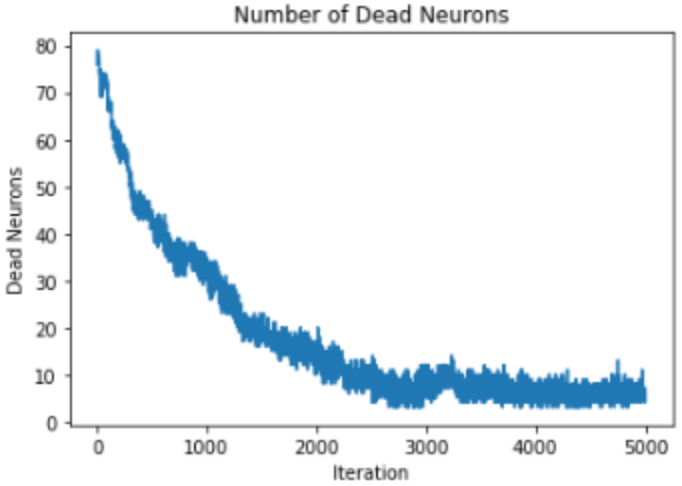
شعاع : ۴

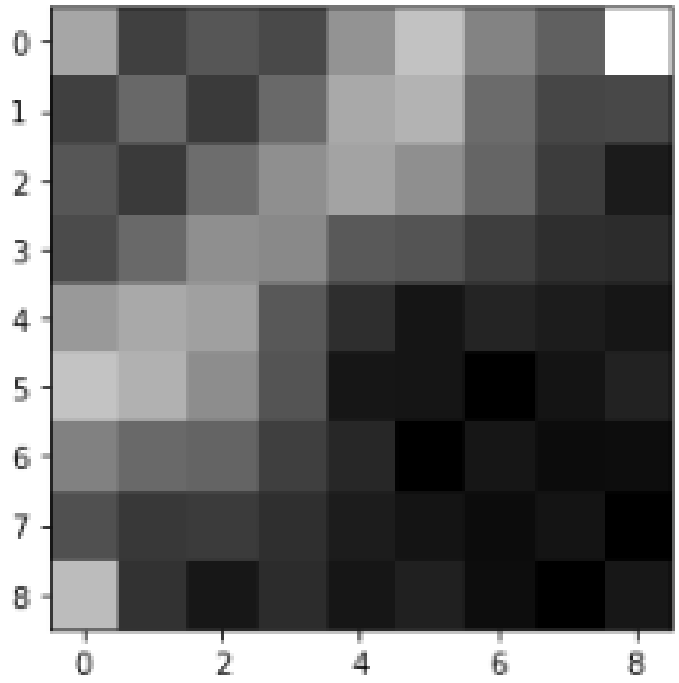
تعداد نرون های خروجی : ۸۱

ضریب کاهش : 5000 / t - 1 است که t برابر مقدار در هر تکرار است.


Training Loss
Loss
Iteration
0
1
2
3
4
5
0
1000
2000
3000
4000
5000


Winners Distance
Distance
Iteration
0
1
2
3
4
5
0
1000
2000
3000
4000
5000


Number of Dead Neurons
Dead Neurons
Iteration
0
10
20
30
40
50
60
70
80
0
1000
2000
3000
4000
5000



مقادیر خواسته شده در حالت بهینه در تصاویر بالا قرار داده شده است.

-۶

در ادامه سوال قبل تنسورفلو را لود میکنیم. سپس با استفاده از تابع درون کلاس SOM برای تبدیل از حالت عادی داده ها به مقدار کاهش یافته توسط کلاس SOM از Feature Extraction استفاده کرده سپس تمام داده ها را به حالت جدید می آوریم و درون متغیر new_X_train قرار میدهیم. برای استفاده از این داده ها ، از داده های اصلی درون متغیر X استفاده میکنیم و سپس دوباره مانند قبل داده ها را به ۲ دسته آموزش و اعتبارسنجی تبدیل میکنیم.

```
#split data to train and validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.10, random_state=42)

x_tr = pd.DataFrame(new_X_train)
y_tr = pd.DataFrame(y_train)
Y_train = to_categorical(y_tr,num_classes=6)

X_valid = pd.DataFrame(X_valid)
y_valid = pd.DataFrame(y_valid)
y_valid = to_categorical(y_valid,num_classes=6)


```

همانند کد های سوال چهارم یک مدل برای داده های جدید ایجاد میکنیم.

```
model  = keras.models.Sequential([
                                  keras.Input(shape =(561,),name = 'input_layer'),
                                  keras.layers.Dense(units = 64 , activation='relu', name= 'hidden_layer1'),
                                  keras.layers.Dense(units = 128 , activation='relu', name= 'hidden_layer2'),
                                  keras.layers.Dense(6,activation='softmax',name='output_layer')
])
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.009),loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(x_tr,Y_train,epochs= 20, validation_data =(X_valid, y_valid) , callbacks=[tb_callBack])
```

سپس مدل های مختلف را تست میکنیم . برای تست کردن نیز از همان مقادیر سوال چهارم استفاده میکنیم زیرا بتوانیم با نتیجه آن سوال مقایسه ای کنیم :

در این مدل با ۲۰ تکرار مدل را fit میکنیم و مدل های مختلف بررسی میکنیم:

۱ لایه مخفی ، ۱۲۸ نرون : دقت ۰/۸۸ برای آموزش و ۰/۹۲ برای تست

۱ لایه مخفی ، ۶۴ نرون : دقت ۰/۸۷ برای آموزش و ۰/۹۰ برای تست

۱ لایه مخفی ، ۳۲ نرون : دقت ۰/۸۷ برای آموزش و ۰/۹۰ برای تست

۱ لایه مخفی ، ۱۶ نرون: دقت ۰/۸۸ برای آموزش و ۰/۹۱ برای تست

۲ لایه مخفی ، ۱۶ نرون و ۵۰ نرون : دقت ۰/۸۸ برای آموزش و ۰/۹۲ برای تست

۲ لایه مخفی ، ۳۲ نرون و ۵۰ نرون : دقت ۰/۸۹ برای آموزش و ۰/۹۲ برای تست

۲ لایه مخفی ، ۶۴ نرون و ۱۲۸ نرون : دقت ۰/۸۹ برای آموزش و ۰/۹۳ برای تست

با بررسی تنسوربرد و همگرایی این مدل ها ، مدل آخر ، ۲ لایه مخفی با ۶۴ نرون و ۱۲۸ نرون در ، و دقت ذکر شده بهترین حالت را ایجاد کرده است. با وجود کاهش مقدار دقت در این مثال ها اگر تعداد تکرار ها را زیاد کنیم نتیجه به دست آمده بهتر از نتیجه به دست آمده در سوال ۴ خواهد بود. از نکات مثبت این کاهش بعد این است که با وجود کاهش میتوان به نتیجه بهتری رسید و چون تعداد بعد داده کمتر است مقدار مصرف حافظه و سرعت بهتر خواهد بود .