فاطمه توکلی _400131016

تمرین سوم شبکه عصبی

.1

شبکه خودسازمانده کوهنن دارای دو لایه ورودی و لایه خروجی می باشد،لایه ورودی به تعداد ویژگی های موجود در داده های مسئله نورون دارد و لایه خروجی نورون های نقشه می باشند که میتوان به صورت یک بعدی دو بعدی تا اندازه ابعاد داده های در نظر گرفت.نحوه اتصال این شبکه fully connected می باشد و هر ورودی با قاعده یادگیری رقابتی به خروجی وزن داری که فاصله (اقلیدسی)کمتری با آن دارد متصل می شود.این ویژگی ها به صورت فشرده تر هستند و اطلاعات بهتری برای تفکیک به ما می دهند.

.2

هنگام آموزش شبکه عصبی خود سازمانده کوهونن، مشکل به اصطلاح "نورون های مرده" رخ می دهد. نورون هایی که دارای ضرایب وزن اولیه بسیار دور از الگوهای ورودی هستند، با وجود طولانی بودن یادگیری، هرگز نمی توانند در رقابت پیروز شوند. در این پیاده سازی مشکل سیستمی الگوریتم آموزشی کوهونن با دو اصلاحیه حل شد.اول از همه، عملکرد انتخاب الگوهای تصادفی از یک مجموعه آموزشی تجدید نظر شد. این انتخاب الگوی تصادفی به طور مساوی را تضمین می کند. در مرحله دوم، وزن نورون ها با مقادیر تصادفی، مقداردهی اولیه می شوند، اما فقط از محدوده ای که در الگوها با آن مواجه می شوند.به این ترتیب، حتی اگر با نورون‌هایی مواجه شویم که هرگز در مسابقه‌ای برنده نمی‌شوند. " نورون‌های مرده"، حداقل در همسایگی برنده باقی می‌مانند و تحت آموزش قرار می‌گیرند و نقش پل بین الگوها را در طول طبقه‌بندی بازی می‌کنند.

.3

در این دیتاست مجموعه آموزش و آزمون جدا شده اند و به طور تقریبی 70% مجموعه آموزش هستند و 30% مجموعه آزمون می باشد که داده اعتبارسنجی را از این قسمت جدا میکنیم تا نسبت کلی به طور تقریبی با صورت سئوال مطابقت داشته باشد

در این قسمت ابتدا داده آموزش را با استفاده ازکتابخانه pandas میخوانیم و نام هرستون را از feature گرفته و انتساب میدهیم، سپس هرنوع فعالیت را با برچسب نوع فعالیت مشخص کرده و y داده آموزش را میگیریم:

```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv('X_train.txt', delim_whitespace=True, header=None)
X_train.columns = [features]

# get y labels from the txt file
y_train = pd.read_csv('y_train.txt', names=['Activity'], squeeze=True)
y_train_labels = y_train.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})

X_train
```

Output:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-meanFreq()	fBodyBodyGyroJerkMag-skewness()	fBodyBodyGyroJerkMag-kurtosis()	angle(tBodyAccMean,grav
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.074323	-0.298676	-0.710304	-0.11
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.158075	-0.595051	-0.861499	0.05
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	0.414503	-0.390748	-0.760104	-0.11
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	0.404573	-0.117290	-0.482845	-0.03
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.087753	-0.351471	-0.699205	0.12
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7347	0.299665	-0.057193	-0.181233	-0.195387	0.039905	0.077078	-0.282301	0.043616	0.060410	0.210795	...	-0.070157	-0.588433	-0.880324	-0.19
7348	0.273853	-0.007749	-0.147468	-0.235309	0.004816	0.059280	-0.322552	-0.029456	0.080585	0.117440	...	0.165259	-0.390738	-0.680744	0.06
7349	0.273387	-0.017011	-0.045022	-0.218218	-0.103822	0.274533	-0.304515	-0.098913	0.332584	0.043999	...	0.195034	0.025145	-0.304029	0.05
7350	0.289654	-0.018843	-0.158281	-0.219139	-0.111412	0.268893	-0.310487	-0.068200	0.319473	0.101702	...	0.013865	0.063907	-0.344314	-0.10
7351	0.351503	-0.012423	-0.203867	-0.269270	-0.087212	0.177404	-0.377404	-0.038678	0.229430	0.269013	...	-0.058402	-0.387052	-0.740738	-0.28

7352 rows × 561 columns

در این قسمت داده تست را خوانده و تقسیم کرده:

```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv('X_test.txt', delim_whitespace=True, header=None)
X_test.columns = [features]

# get y labels from the txt file
y_test = pd.read_csv('y_test.txt', names=['Activity'], squeeze=True)
y_test_labels = y_test.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})

X_test
```

```
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.3, random_state=42)
```

```
print('Train shape : ' + str(X_train.shape))
print('Test shape : ' + str(X_test.shape))
print('Validation shape : ' + str(X_val.shape))
```

```
Train shape : (2103, 561)
Test shape : (2062, 561)
Validation shape : (885, 561)
```

در این دیتاست y از یک شروع شده است و برای آموزش شبکه عصبی دچار اشکال میشویم پس آن را اصلاح میکنیم:

```
y_train = y_train.replace(6,5)
```

```
np.unique(y_train)
```

```
array([0, 1, 2, 3, 4, 5])
```

```
y_test = y_test.replace(6,5)
```

```
np.unique(y_test)
```

```
array([0, 1, 2, 3, 4, 5])
```

4.

شبکه دارای 3 لایه می باشد که لایه اول 32 نورون با تابع فعالسازی sigmoid می باشد، لایه دوم 10 نورون با تابع فعالسازی sigmoid و اخرین لایه به تعداد خروجی های مسئله 6 نورون دارد و تابع آن softmax می باشد :

```
model = keras.models.Sequential([
                                 keras.layers.Dense(units=32,activation='sigmoid'),
                                 keras.layers.Dense(units =10, activation = 'sigmoid'),
                                 keras.layers.Dense(6, activation = 'softmax')

                                ])
```

```
model.compile(optimizer='adam', loss = keras.losses.SparseCategoricalCrossentropy(),metrics=['accuracy'])
```

```
model.fit(X_train,y_train,epochs=30)
```

نتیجه ده مرحله یادگیری اخر و دقت به دست آمده با پارامترهای تعیین شده:

```
Epoch 20/30
230/230 [==============================] - 0s 2ms/step - loss: 0.0553 - accuracy: 0.9827
Epoch 21/30
230/230 [==============================] - 0s 2ms/step - loss: 0.0530 - accuracy: 0.9833
Epoch 22/30
230/230 [==============================] - 0s 1ms/step - loss: 0.0501 - accuracy: 0.9839
Epoch 23/30
230/230 [==============================] - 0s 1ms/step - loss: 0.0533 - accuracy: 0.9815
Epoch 24/30
230/230 [==============================] - 0s 2ms/step - loss: 0.0449 - accuracy: 0.9852
Epoch 25/30
230/230 [==============================] - 0s 2ms/step - loss: 0.0444 - accuracy: 0.9857
Epoch 26/30
230/230 [==============================] - 0s 1ms/step - loss: 0.0404 - accuracy: 0.9869
Epoch 27/30
230/230 [==============================] - 0s 2ms/step - loss: 0.0426 - accuracy: 0.9864
Epoch 28/30
230/230 [==============================] - 0s 2ms/step - loss: 0.0388 - accuracy: 0.9868
Epoch 29/30
230/230 [==============================] - 0s 2ms/step - loss: 0.0387 - accuracy: 0.9864
Epoch 30/30
230/230 [==============================] - 0s 2ms/step - loss: 0.0368 - accuracy: 0.9875
<keras.callbacks.History at 0x7fb21d2274d0>
```

نتیجه به دست آمده بر روی داده اعتبارسنجی:

```
test_loss,test_accuracy = model.evaluate(X_test,y_test,verbose=2)
```

```
65/65 - 0s - loss: 0.1965 - accuracy: 0.9355 - 323ms/epoch - 5ms/step
```

5.

پیاده سازی self-organizing map:

در ابتدا یک mapبه اندازه شبکه ایی که میخاهیم ایجاد کنیم میسازیم و رندوم مقدار دهی میکنیم:

مقادیر اولیه پارامتر یادگیری، سایز شبکه و داده ورودی را می دهیم، شعاع اولیه را نصف اندازه طول شبکه در نظر میگیریم

```
class SOM:
    def __init__(self,data, map_size,dir_map="not load from file", lr = 0.2):

        self.map = np.random.random(size=(map_size[0], map_size[1], map_size[2]))
        self.lr0 = lr
        self.lr = self.lr0
        self.R0 = map_size[0]//2
        self.R = self.R0
        if dir_map != "not load from file":
          self.load_map(dir_map)
```

سپس تابع آموزش را تعریف میکنیم که در هرمرحله به صورت رندوم یک ایندکس در بازه تعریف شده انتخاب میکند و تمامی ویژگی های آن نمونه را در نظر میگیرد و تابع winner را روی آن اعمال می کند و همسایه های برنده را در شعاع تعریف شده فرامیخواند و وزن آن ها به روز رسانی میکند js را برای بررسی تغییرات و نشان دادن آنها در نظر میگیریم:

```
def train(self, X, y, T=1000, error_threshold=10**-50):
    # Loss history
    Js = []
    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])
        for i in range(len(X)):
            x = X.iloc[i, :]
            # Neuron with most compatibilty with x
            winner = self.find_winner(x)
            # 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, NS, len(X))

        # 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}")
            self.visualize(X,y)

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

    return Js
```

فاصله اقلیدسی نمونه iام با 651 ویژگی را از شبکه ایی که تعریف کردیم حساب میکند و ایندکس کمترین آن ها را برمیگرداند

```
def find_winner(self, x):
    rep_x = np.tile(x, (self.map.shape[0], self.map.shape[1], 1))
    dists = np.sum((self.map - rep_x) ** 2, axis=2)

    winner = np.unravel_index(np.argmin(dists, axis=None), shape=dists.shape)
    return winner
```

همسایه ها در شعاع R را پیدا کرده، اگر در بازه تعریف شده بودند مقدار آن ها را به صورت زیر میگذارد:

```
1/ ( 1+ np.sqrt(ri**2 + rj**2))
```

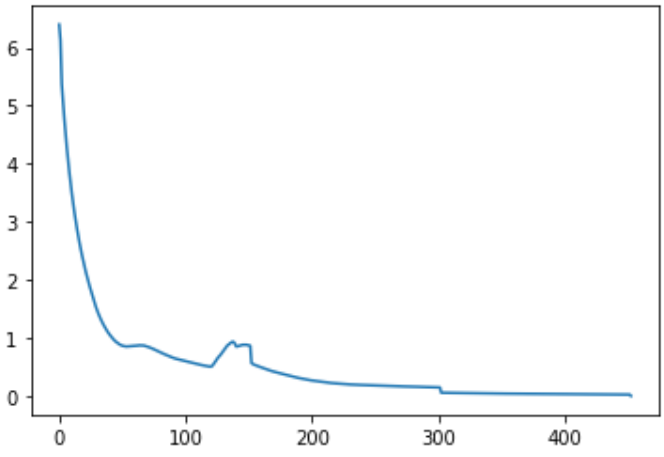
```
def get_NS(self, winner):
    # not neighbor = 0 , neighbor = 1/sqrt(euclidean_distance)
    NS = np.zeros(shape= (self.map.shape[0], self.map.shape[1]))
    iw, jw = winner[0], winner[1]
    R = int(self.R)
    for ri in range(-R, R):
        for rj in range(-R, R):
            if (0 <= iw + ri < self.map.shape[0]) and  (0 <= jw + rj < self.map.shape[1]):
                NS[iw + ri, jw + rj] = 0 if np.sqrt(ri**2 + rj**2) > R else 1/ ( 1+ np.sqrt(ri**2 + rj**2))
    return NS
```

سپس وزون همسایه ها را با این شیوه اصلاح می کند:

```
def update_weights(self, x, n_strength, X_len):
    NS = np.tile(n_strength, [self.map.shape[2],1,1]).transpose()
    rep_x = np.tile(x, [self.map.shape[0], self.map.shape[1], 1])
    Delta = rep_x - self.map
    self.map = self.map + (self.lr/X_len) * np.multiply(NS, Delta)

def visualize(self, X, y):
    scores = np.zeros(shape=(self.map.shape[0], self.map.shape[1],len(list(set(y)))))
    self.scores = np.zeros(shape=(self.map.shape[0], self.map.shape[1]))
    for i in range(len(X)):
        x = X.iloc[i, :]
        x = np.asarray(x)
        winner = self.find_winner(x)
        iw, jw = winner[0], winner[1]
        scores[iw, jw][y[i]] += 1
    for i in range(len(scores)):
        for j in range(len(scores[0])):
          self.scores[i,j] = np.argmax(scores[i,j])
    print(self.scores)
    c = plt.imshow(self.scores, cmap='jet')
    plt.colorbar(c)
    for i in range(len(scores)):
      for j in range(len(scores[0])):
        plt.text(j,i, self.scores[i, j],ha="center", va="center", color="w")
    plt.title("Class of each Neurons")
    plt.show()



def extract_feature(self, x): # here we give a data of n feature and take a matrix of size map as output (e.g, 9*9)
    x = np.asarray(x)
    rep_x = np.tile(x, [self.map.shape[0], self.map.shape[0], 1])
    dists = np.sum((self.map - rep_x)**2, axis=2)
    return 1/ (1 + dists)
```

خروجی شبکه بر روی داده های ما به صورت زیر می باشد:

```
m_size = 9
som_net = SOM(data = X_train ,map_size = [m_size,m_size,X_train.shape[1]])
Js = som_net.train(X_train, y_train , T = 600)
plt.plot(Js)
plt.show()
```



همانطور که مشاهده میشود تغییرات به تدریج کم شده و شبکه ما در itr = 350 همگرا میشود و ثابت می ماند.