به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین سری چهارم

نام و نام خانوادگی	محسن فیاض
شماره دانشجویی	810100524
تاریخ ارسال گزارش	24 خرداد 1400

فهرست گزارش سوالات

سوال 1 – SOM **1**

الف) دیتاست fashion_mnist را لود کرده و1000 تصویر اول را به عنوان داده آموزش و 3000 داده را به عنوان داده تست در نظر بگیرید. اگر تعداد 225(=15*15) نورون خروجی با چینش مربعی برای SOM در نظر بگیریم با توجه به ابعاد تصاویر ورودی، ماتریس وزن ها دارای چه ابعادی خواهد بود. همچنین مشخص کنید نوع چینش نورون های خروجی در کدام یک از مراحل زیر تاثیرگذار است؟ به طور خلاصه توضیح دهید. 1

ب) الگوریتم را با شعاع همسایگی (ثابت) R = 1 پیاده سازی کرده و شبکه را به ازای مقادیر مختلف Initial-Learning Rate و decay factor آموزش داده و سپس نتایج بهترین حالت به دست آورده شده را گزارش دهید. گزارش باید شامل مقادیر هایپرپارامترها و چند تصویر از وزن های شبکه در اپیاک های مختلف که به صورت یک ماتریس با ابعاد 420x420 در آمده است، باشد. (در واقع وزن های مربوط به هر کدام از نورون های خروجی یک قسمت از این تصویر را تشکیل می دهند که دارای ابعاد 28*28 است و از کنار هم قرار گرفتن این تصاویر کوچک یک ماتریس به ابعاد 28*420*420 = 15x28*15 تشکیل می شود). 3

(اختیاری :تعداد ورودی های مپ شده و تعداد کلاس های منحصر به فرد مپ شده به هر نورون خروجی را نیز به صورت یک (یا دو) هیستوگرام رسم کنید. به چند نورون بیشتر از یک کلاس مپ شده است؟) 4

(اختیاری: به ازاء شعاع همسایگی (ثابت) R=0 تصویر وزن های شبکه چگونه می شود؟ توضیح دهید.) 5

پ) مرحله قبل را با شعاع همسایگی اولیه R = 3 تکرار کنید و در هر اپیاک شعاع همسایگی را یک عدد کاهش دهید تا به صفر برسد (و سپس بر روی صفر ثابت بماند!) ، همچنین مقادیر هایپرپارامتر ها را دوباره تنظیم کنید. نتایج را همانند مرحله قبل گزارش کنید. (اختیاری :تعداد ورودی های مپ شده و تعداد کلاس های منحصر به فرد مپ شده به هر نورون خروجی را نیز به صورت یک (یا دو) هیستوگرام رسم کنید.به چند نورون بیشتر از یک کلاس مپ شده است؟) 6

ت) آیا بهبودی در نتایج به دست آمده ایجاد شده است؟ به طور مختصر توضیح دهید شعاع همسایگی متغیر چگونه می تواند به بهبود نتایج کمک کند؟ 8

سوال 2 – MaxNet **9**

به ازای x = (1.2, 1.1, 1 , 0.9 , 0.95 , 1.15) و با ε = 0.15 شبیه سازی انجام دهید و مراحل بروز رسانی واحد ها را نشان دهید و سپس سوالات زیر پاسخ دهید 9

توضیح دهید و پیاده سازی کنید تحت چه شرایطی وبا چه اصلاحاتی می توان شبکه Maxnet برای پیدا کردن بزرگترین عدد در بین اعدادی که می دانیم همه این اعداد از $R\epsilon\beta$ بزرگتر هستند استفاده کرد؟ 10

در چه شرایطی وبا چه اصلاحاتی می توان مکانیزم شبکه maxnet را برای مرتب کردن اعداد از بزرگ به کوچک کاربرد؟ 10

در چه شرایطی وبا چه اصلاحاتی می توان مکانیزم شبکه maxnet را برای مرتب کردن اعداد از کوچک به بزرگی کاربرد؟ 11

سوال 3 – Mexican Hat **12**

با استفاده از شبکه Mexican Hat واحدی که مقدار ماکزیمم دارد را از بردار زیر پیدا کنید. 12

[0.27, 0.35, 0.44, 0.58, 0.66, 0.77, 0.4, 0.32, 0.20, 0.15, 0.08] 12

الف: مقدار R1= 0 مقدار R2= ∞ در نظر بگیرید. 12

ب: مقدار R1 =1 مقدار R2 = 3 در نظر بگیرید. 13

در هر بار تکرار نمودار Index اعضای آرایه و مقدار سیگنال خروجی را رسم کرده و در نهایت نتایج دو قسمت را مقایسه کنید tmax (و سایر پارامترها را مقادیر مناسب در نظر بگیرید). 13

سوال 4 – Hamming Net 15

الف) تصاویر زیر را در چند آرایه از نوع numpy ذخیره و فاصله Hamming را برای هر جفت از آنها محاسبه کنید. خانه های سیاه 1 و خانه های سفید 1- در نظر بگیرید! 15

ب) 4 تصویر قسمت اول را به عنوان نمونه ها (exemplars) شبکه Hamming-net در نظر گرفته و با فرض n = 12 (ابعاد داده ورودی شبکه)، ماتریس وزن ها و بایاس شبکه Hamming-net را به دست آورید. 16

پ) با استفاده از ماتریس وزن و بایاس به دست آمده در مرحله قبل خروجی های شبکه را برای ورودی زیر محاسبه کنید. بر اساس خروجی های به دست آمده ورودی به کدام یک از تصاویر (X,Y,A,C) نزدیک تر است. 17

سوال 1 – SOM

الف) دیتاست fashion_mnist را لود کرده و 1000 تصویر اول را به عنوان داده آموزش و 3000 داده را به عنوان داده تست در نظر بگیرید. اگر تعداد 225(=15*15) نورون خروجی با چینش مربعی برای SOM در نظر بگیریم با توجه به ابعاد تصاویر ورودی، ماتریس وزن ها دارای چه ابعادی خواهد بود. همچنین مشخص کنید نوع چینش نورون های خروجی در کدام یک از مراحل زیر تاثیرگذار است؟ به طور خلاصه توضیح دهید.

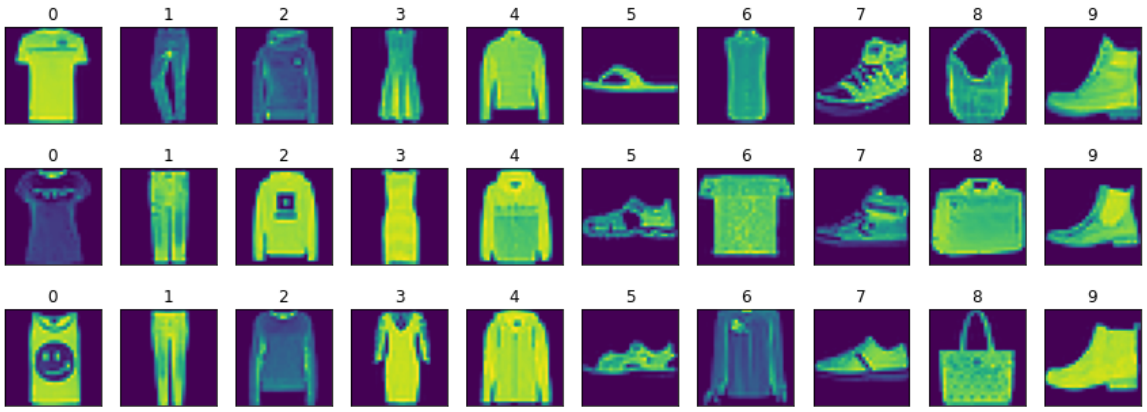
ابتدا دیتاست را بارگذاری کرده و همانطور که خواسته شده 1000 تصویر اول آموزش را برای آموزش و 3000 داده اول تست را برای تست در نظر گرفتیم. ضمنا ورودی‌ها را به جای 0 تا 255 به 0 تا 1 منتقل کردیم. همچنین تصاویر در کلاسهای متفاوت را نمایش دادیم تا در آینده با خروجی های SOM مقایسه کنیم.

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
x_train, y_train = x_train[:1000]/255, y_train[:1000]
x_test, y_test = x_test[:3000]/255, y_test[:3000]
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
(1000, 28, 28) (1000,) (3000, 28, 28) (3000,)
```

```
def plot_pics(x: np.ndarray, y: np.ndarray):
    fig = plt.figure(figsize=(15, 4))
    for class_num in range(10):
        class_indices = np.where(y == class_num)[0]
        class_x = x[class_indices]
        rnd_idx = np.random.randint(len(class_x))
        image = class_x[rnd_idx]
        ax = fig.add_subplot(1, 10, class_num + 1, xticks=[], yticks=[])
        ax.set_title(class_num)
        plt.imshow(image)
    plt.show()

for i in range(3):
    plot_pics(x_train, y_train)
```



شکل 1 - بارگذاری و نمایش دیتاست

از آنجا که ورودی سایز 28*28=784 و خروجی 225 است، ماتریس وزنها از ورودی‌ها به خروجی دارای ابعاد 784 در 225 خواهد بود.

در SOM، در هر مرحله پس از انتخاب نزدیک‌ترین نورون خروجی به ورودی یا همان برنده، وزن‌های نورون‌ای خروجی اطرافش نیز تا یک شعاع مشخص آپدیت می‌شوند. این کار در step 5 انجام می‌شود و اینکه چینش به چه شکل باشد، مثلا در اینجا مربعی، مشخص می‌کند که چه نورون‌هایی در این همسایگی می‌افتند , همسایگی به چه شکل باشد که در صفحه 171 کتاب هم نمونه هایی از آن آمده است.

ب) الگوریتم را با شعاع همسایگی (ثابت) R = 1 پیاده سازی کرده و شبکه را به ازای مقادیر مختلف Initial- Learning Rate و decay factor آموزش داده و سپس نتایج بهترین حالت به دست آورده شده را گزارش دهید. گزارش باید شامل مقادیر هایپرپارامترها و چند تصویر از وزن های شبکه در ایپاک های مختلف که به صورت یک ماتریس با ابعاد 420x420 در آمده است، باشد. (در واقع وزن های مربوط به هر کدام از نورون های خروجی یک قسمت از این تصویر را تشکیل می دهند که دارای ابعاد 28*28 است و از کنار هم قرار گرفتن این تصاویر کوچک یک ماتریس به ابعاد 28*= 15x28*15 420*420 تشکیل می شود).

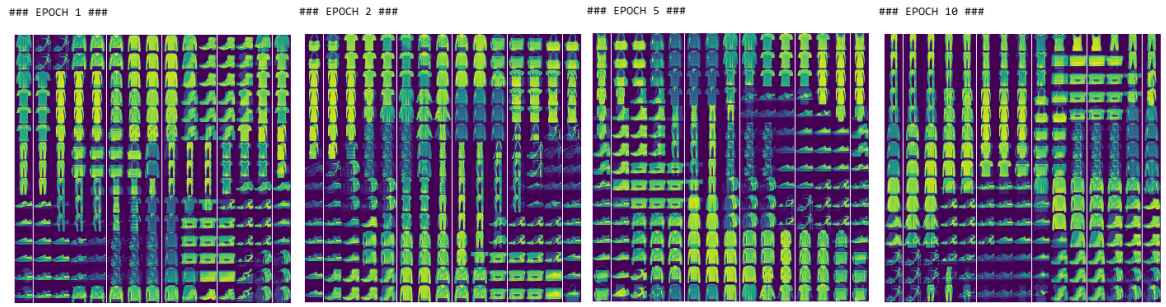
مرحله اول الگوریتم مقداردهی اولیه وزن‌ها است. در صفحه 172 کتاب آمده است:

Random values may be assigned for the initial weights. If some information is available concerning the distribution of clusters that might be appropriate for a particular problem, the initial weights can be taken to reflect that prior knowledge. In Examples 4.4-4.9, the weights are initialized to random values (chosen from the same range of values as the components of the input vectors).

بنابراین چون مقادیر ورودی را بین 0 و 1 گذاشتیم، وزن‌ها را نیز به مقادیر رندوم در این محدوده مقدار دهی اولیه می‌کنیم.

سپس مقادیر هایپرپارامتر مختلف تست شد که اولا نباید خیلی بزرگ باشد lr مثلا چون باعث می‌شود یادگیری دچار مشکل شود و پس از انتخاب دو هایپرپارامتر لرنینگ ریت 0.7 و decay برابر 0.99، نتیجه زیر دیده شد.

```
som = SOM(input_size=784, output_shape=(15, 15),
          radius_of_neighborhood=1, radius_of_neighborhood_reduction=0,
          learning_rate=0.7, learning_rate_decay=0.99)
som.train(x_train, epochs=10, plot_weights=True)
som.evaluate(x_test, y_test)
```



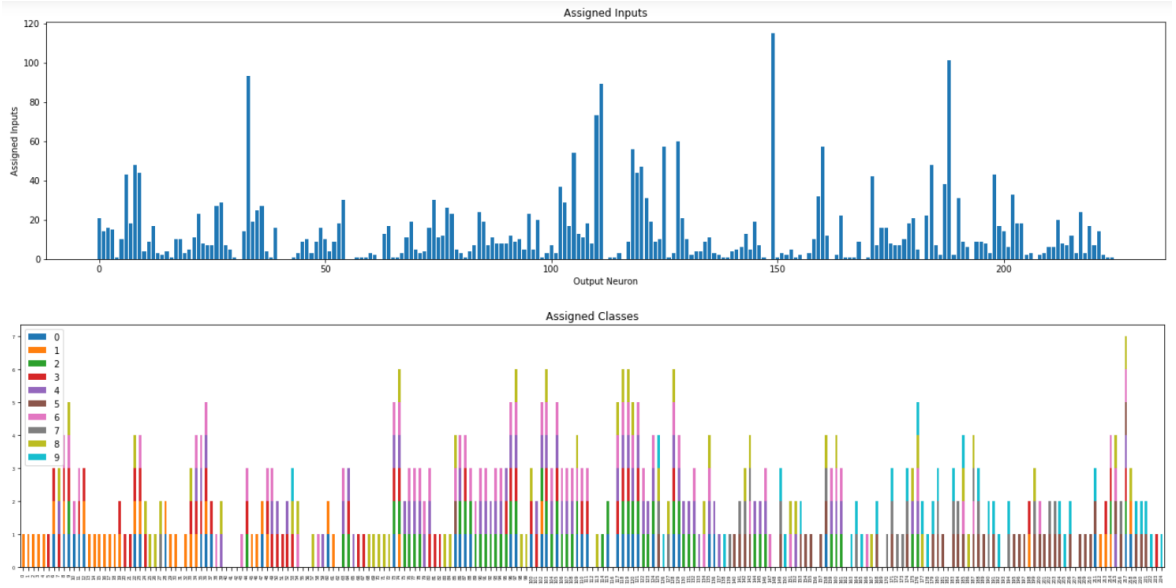
شکل 2 - هایپرپارامترها و نتایج R=1

در شکل بالا نکته جالبی که دیده می‌شود این است که چون شعاع همسایگی برابر 1 بود، در وزن‌ها هم مربع های 3 در 3 دیده می‌شود که شبیه به هم شده‌اند و چون شعاع را کم

نمی‌کنیم همین موضوع تا آخر هم دیده می‌شود.

(اختیاری :تعداد ورودی های مپ شده و تعداد کلاس های منحصر به فرد مپ شده به هر نورون خروجی را نیز به صورت یک (یا دو) هیستوگرام رسم کنید. به چند نورون بیشتر از یک کلاس مپ شده است؟)

برای این کار یک تابع evaluate هم در کلاس som نوشتم که در کدهای کنار این فایل آمده است. این کد روی بخش تست اجرا شده و مواردی که خواسته شده را خروجی می‌دهد.



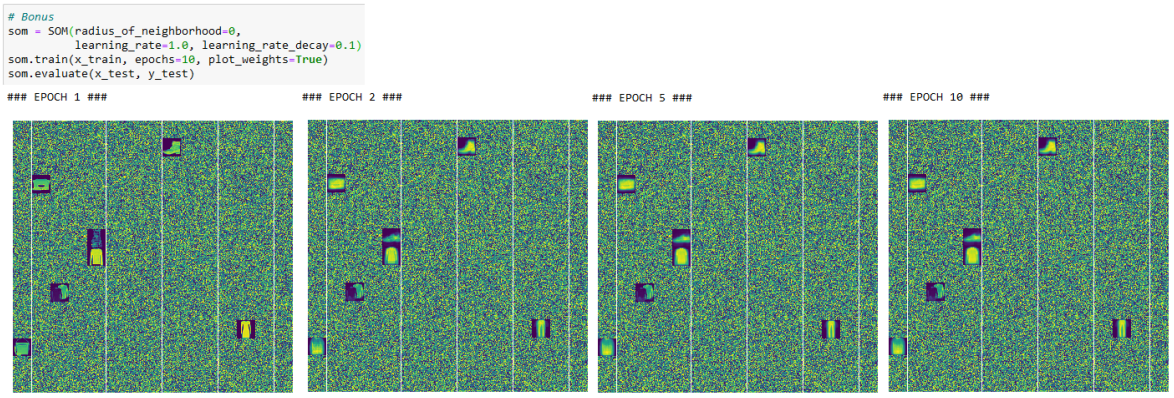

```
More than 1 class #neurons: 130
```

شکل 3 - هیستوگرام R=1

همانطور که دیده می‌شود هر نورون توانسته بخشهایی از دیتا را به خود اختصاص دهد و ضمنا در شکل پایین می‌بینیم که هر نورون حدودا 1 کلاس را در بر گرفته و در بعضی موارد بیشتر که عدد دقیق نورونهایی که بیشتر از یک کلاس دارند 130 است. ضمنا از رنگها مشخص است که انواع مختلف کلاسها هم یادگیری شده است. ضمنا مثلا می‌شود دید که در بعضی موارد که چند کلاس هست مثلا کلاس 7 و 9 که هر دو شکل کفش هستند مثلا با هم اشتباه شده‌اند که معنادار است.

(اختیاری: به ازاء شعاع همسایگی (ثابت) R=0 تصویر وزن های شبکه چگونه می شود؟ توضیح دهید.)

نتیجه اجرا به شکل زیر شد.



شکل 4 - هایپرپارامترها و نتایج R=0

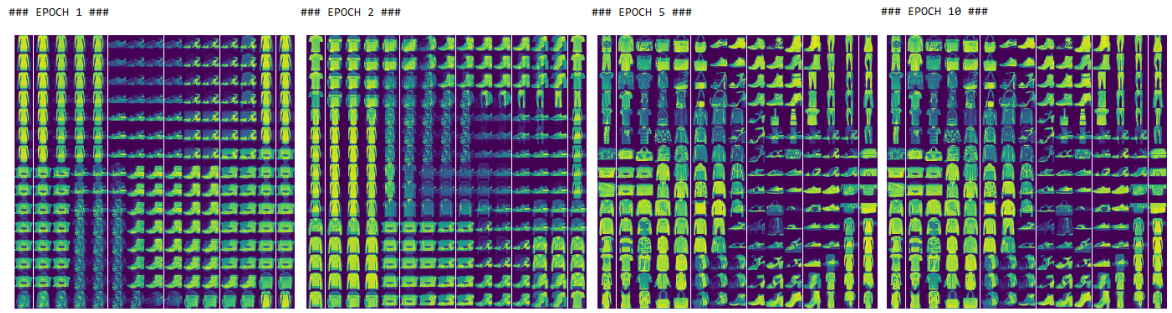
در این بخش می‌بینیم که فقط یک سری نورون خاص آموزش می‌بینند. توضیح این اتفاق این است که با R=0 یعنی فقط همان نورونی که برنده شده است آپدیت می‌شود. مثلا اگر اول کار یکی از نورونها که وزن رندوم دارد به یکی از شکل ها نزدیکتر باشد، همان آپدیت می‌شود و نزدیکتر می‌شود و دفعه های بعدی هم طبیعتا همان به ورودی ها نزدیکتر است و باز همان برنده می‌شود و بقیه نورون‌ها چون کمکی بخاطر شعاع همسایگی نمی‌گیرند، کاملا شانس خود برای برنده شدن را از دست می‌دهند. بنابراین ناحیه جذب های پراکنده در نورون های خروجی شکل می‌گیرد.

پ) مرحله قبل را با شعاع همسایگی اولیه R = 3 تکرار کنید و در هر ایپاک شعاع همسایگی را یک عدد کاهش دهید تا به صفر برسد (و سپس بر روی صفر ثابت بماند!) ، همچنین مقادیر هایپرپارامتر ها را دوباره تنظیم کنید. نتایج را همانند مرحله قبل گزارش کنید. (اختیاری :تعداد ورودی های مپ شده و تعداد کلاس های منحصر به فرد مپ شده به هر نورون خروجی را نیز به صورت یک (یا دو) هیستوگرام رسم کنید.به چند نورون بیشتر از یک کلاس مپ شده است؟)

امکان کاهش شعاع در هر ایپاک هم در پیاده سازی که کنار این فایل است نوشته شد.

نتیجه اجرا با موارد خواسته شده به شکل زیر است.

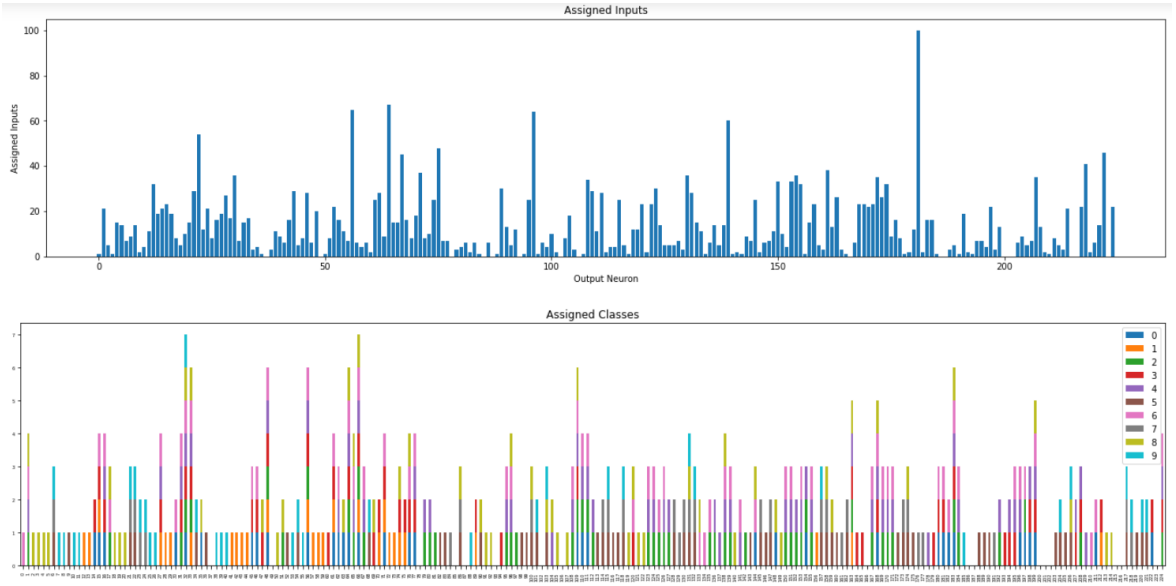
```
som = SOM(radius_of_neighborhood=3, radius_of_neighborhood_reduction=1,
          learning_rate=0.6, learning_rate_decay=0.99)
som.train(x_train, epochs=10, plot_weights=True)
som.evaluate(x_test, y_test)
```



شکل 5 - هایپرپارامترها و نتایج R=3 و کاهش 1

در اینجا می‌بینیم که اول که شعاع همسایگی بزرگ است، دسته‌ها در وزنها هم کنار هم می‌افتند و شعاع بزرگی دارند. در ادامه که شعاع کم می‌شود، داخل هر دسته هم نزاع ایجاد می‌شود و کم کم هر نورون از بقیه مستقل می‌شود و از حالت cooperative به حالت competitive می‌رسیم. که می‌بینیم در انتها مثلا در همان شعاع بزرگی که همه کفش یک مدل بودند، حالا هر کدام به یک شکل کفش خاص رسیده است.

ضمنا بخش اختیاری نیز در زیر آمده است.




```
More than 1 class #neurons: 120
```

شکل 6 - هایپرپارامترها و نتایج R=3 و کاهش 1

در اینجا هم می‌بینیم که هر نورون خاص منظوره تر شده است و تعداد نورونهای با بیشتر از 1 کلاس به تعداد 120 شده است. در بخش کامل تر توضیح می‌دهیم.

ت) آیا بهبودی در نتایج به دست آمده ایجاد شده است؟ به طور مختصر توضیح دهید شعاع همسایگی متغیر چگونه می تواند به بهبود نتایج کمک کند؟

بله همانطور که دیده شد تعداد نورونهای با بیش از یک کلاس از 130 به 120 رسید و ضمنا تعداد ورودی‌هایی که به هر نورون اختصاص می‌یابد نیز بین نورون‌ها پخش شده‌اند. همانطور که گفته شد شعاع همسایگی متغیر باعث می‌شود در ابتدا نورونها با هم همکاری cooperative داشته باشند و همدیگر را کمک کنند و ناحیه جذب های بزرگ برای دسته‌های کلی تر ایجاد شود مثل دسته های شبیه به کفش به صورت کلی. اما در ادامه که شعاع متغیر کاهش یابد، باعث می‌شود همان نورون‌ها هم با هم به نزاع competitive بپردازند و مثلا دسته کلی کفش‌ها بتواند به زیر دسته های خودش شکسته شود و داده بهتر تشخیص داده شود که طبق نتایج هم این فایده مشاهده شد و دسته بندی ها بهتر انجام شدند.

سوال 2 – MaxNet

به ازای x = (1.2, 1.1, 1 , 0.9 , 0.95 , 1.15) و با ε = 0.15 شبیه سازی انجام دهید و مراحل بروز رسانی واحد ها را نشان دهید و سپس سوالات زیر پاسخ دهید

الگوریتم پیاده شده و نتیجه شبی‌سازی در زیر آمده است.

```
class MaxNet:
    def __init__(self, epsilon=0.15):
        self.epsilon = epsilon

    def f(self, x):
        return max(0, x)

    def max(self, x):
        a = np.array(x)
        print(a)
        while (a > 0).sum() > 1:
            a_old = a.copy()
            for i in range(len(a)):
                a[i] = self.f(a_old[i] - self.epsilon * np.sum(np.delete(a_old, i)))
            print(a)
        return np.argmax(a), x[np.argmax(a)]

maxnet = MaxNet(epsilon=0.15)
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
maxnet.max(x=[1.2, 1.1, 1, 0.9, 0.95, 1.15])
```

```
[ 1.200  1.100  1.000  0.900  0.950  1.150]
[ 0.435  0.320  0.205  0.090  0.148  0.377]
[ 0.264  0.132  0.000  0.000  0.000  0.198]
[ 0.215  0.062  0.000  0.000  0.000  0.139]
[ 0.184  0.010  0.000  0.000  0.000  0.097]
[ 0.168  0.000  0.000  0.000  0.000  0.068]
[ 0.158  0.000  0.000  0.000  0.000  0.043]
[ 0.152  0.000  0.000  0.000  0.000  0.019]
[ 0.149  0.000  0.000  0.000  0.000  0.000]

(0, 1.2)
```

شکل 7 - الگوریتم و نتیجه مکس نت

همانطور که دیده می‌شود، اعداد با هم به نزاع می‌پردازند و کاهش می‌یابند تا در آخر تنها عددی که به 0 نرسیده است ایندکس ماکسیمم است و آن ایندکس و مقدارش را خروجی می‌دهیم.

- **توضیح دهید و پیاده سازی کنید تحت چه شرایطی وبا چه اصلاحاتی می توان شبکه Maxnet برای پیدا کردن بزرگترین عدد در بین اعدادی که می دانیم همه این اعداد از $R\epsilon\beta$ بزرگتر هستند استفاده کرد؟**

چون MaxNet با اعداد منفی مشکل دارد نباید ورودی منفی باشد. ولی چون فرض مسئله این است که همه اعداد از بتا بزرگتر هستند، بنابراین ابتدا همه را با بتا جمع می‌کنیم تا همه اعداد مثبت شوند و سپس روی آنها maxnet را اجرا می‌کنیم.

```
class MaxNetBeta:
    def __init__(self, epsilon=0.15, min_value=0):
        self.epsilon = epsilon
        self.min_value = min_value

    def f(self, x):
        return max(0, x)

    def max(self, x):
        a = np.array(x) - self.min_value
        print(a)
        while (a > 0).sum() > 1:
            a_old = a.copy()
            for i in range(len(a)):
                a[i] = self.f(a_old[i] - self.epsilon * np.sum(np.delete(a_old, i)))
            print(a)
        return np.argmax(a), x[np.argmax(a)]

maxnet = MaxNetBeta(epsilon=0.15, min_value = -1)
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
maxnet.max(x=[-1, -0.5, 1, 0.9, 0.95, 1.15])
```

```
[ 0.000  0.500  2.000  1.900  1.950  2.150]
[ 0.000  0.000  1.025  0.910  0.967  1.197]
[ 0.000  0.000  0.564  0.432  0.498  0.762]
[ 0.000  0.000  0.310  0.158  0.234  0.538]
[ 0.000  0.000  0.171  0.000  0.083  0.433]
[ 0.000  0.000  0.093  0.000  0.000  0.395]
[ 0.000  0.000  0.034  0.000  0.000  0.381]
[ 0.000  0.000  0.000  0.000  0.000  0.376]

(5, 1.15)
```

شکل 8 - الگوریتم مکس نت با بتا

همانطور که دیده می‌شود می‌بینیم که اول کار همه اعداد مثبت شده اند و سپس مکس نت توانسته ماکسیمم را به خوبی پیدا کند.

- **در چه شرایطی وبا چه اصلاحاتی می توان مکانیزم شبکه maxnet را برای مرتب کردن اعداد از بزرگ به کوچک کاربرد؟**

برای مرتب کردن اعداد از بزرگ به کوچک، هر بار ماکسیمم را پیدا کرده و سپس آن را حذف می‌کنیم و دوباره روی باقی اعداد مکس نت را اجرا می‌کنیم تا تمام لیست مرتب شود.

```
x = [1.2, 1.1, 1, 0.9, 0.95, 1.15, 2]
sorted_x = []
for i in range(len(x)):
    maxnet = MaxNet(epsilon=0.15, verbose=False)
    i, v = maxnet.max(x)
    x.pop(i)
    sorted_x.append(v)
sorted_x
```

```
[2, 1.2, 1.15, 1.1, 1, 0.95, 0.9]
```

شکل 9 - الگوریتم مکس نت برای مرتب سازی بزرگ به کوچک

همانطور که دیده می‌شود، لیست مرتب شده در نهایت به دست آمد. البته شرایط اینکه اعداد باید بیشتر از 0 باشند موجود است.

- **در چه شرایطی وبا چه اصلاحاتی می توان مکانیزم شبکه maxnet را برای مرتب کردن اعداد از کوچک به بزرگی کاربرد؟**

مثل بخش قبل تک تک ماکسیمم را پیدا می‌کنیم و فقط به جای append کردن در بخش قبل، اینجا در ابتدای لیست عدد جدید را وارد می‌کنیم.

```
x = [1.2, 1.1, 1, 0.9, 0.95, 1.15, 2]
sorted_x = []
for i in range(len(x)):
    maxnet = MaxNet(epsilon=0.15, verbose=False)
    i, v = maxnet.max(x)
    x.pop(i)
    sorted_x.insert(0, v)
sorted_x
```

```
[0.9, 0.95, 1, 1.1, 1.15, 1.2, 2]
```

شکل 10 - الگوریتم مکس نت برای مرتب سازی کوچک به بزرگ

و لیست مرتب از کوچک به بزرگ را داریم و البته باید حتما شرایط مکس نت که اعداد همه مثبت باشند باید رعایت شود.

سوال 3 – Mexican Hat

با استفاده از شبکه Mexican Hat واحدی که مقدار ماکزیمم دارد را از بردار زیر پیدا کنید.

[0.27, 0.35, 0.44, 0.58, 0.66, 0.77, 0.4, 0.32, 0.20, 0.15, 0.08]

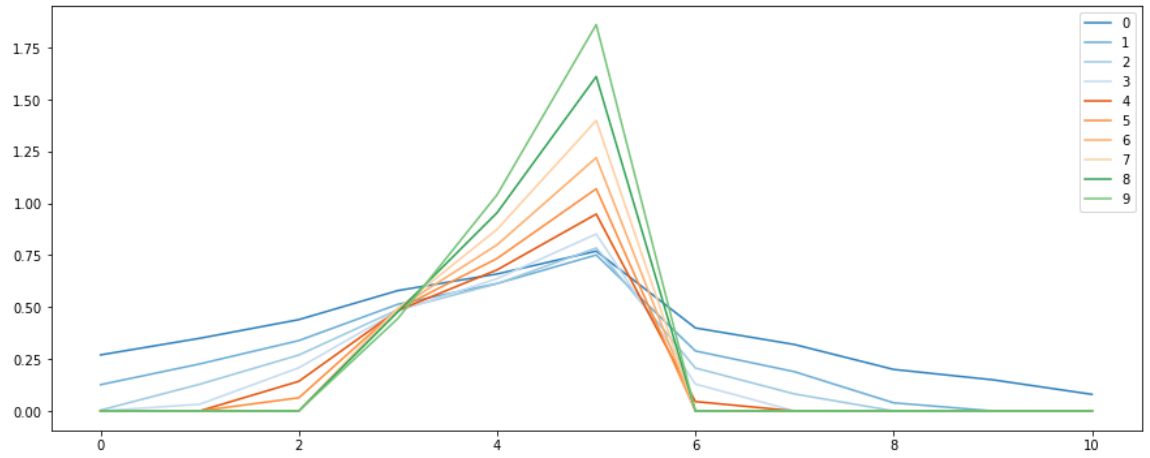
الف: مقدار R1= 0 مقدار R2= ∞ در نظر بگیرید.

پیاده سازی الگوریتم مورد نظر در فایل کنار این گزارش آمده است. نتیجه اجرا با مقادیر الف در زیر آمده است.

```
mexican_hat = MexicanHat(r1=0, r2=np.inf, c1=1.2, c2=-0.05)
mexican_hat.iterate([0.27, 0.35, 0.44, 0.58, 0.66, 0.77, 0.4, 0.32, 0.20, 0.15, 0.08])
```

```
100% ████████████████████ 10/10 [00:00<00:00, 144.93it/s]

(5, 0.77)
```



شکل 11 - الگوریتم و نتیجه کلاه مکزیکی الف

مشاهده می‌شود که پس از اجرای الگوریتم، ایندکس 5ام با مقدار 0.77 بیشترین مقدار را دارد.

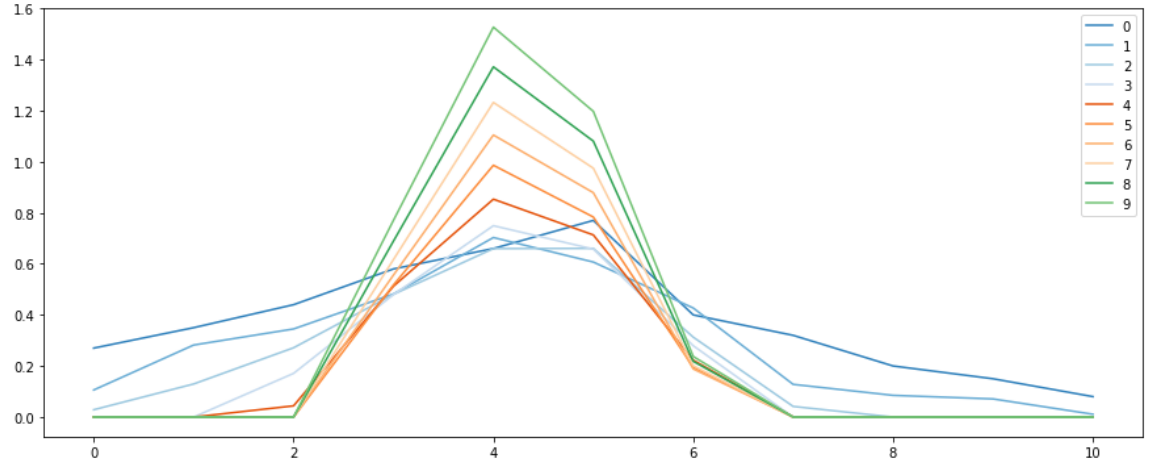
ب: مقدار R1 =1 مقدار R2 = 3 در نظر بگیرید.

نتیجه اجرای ب در زیر آمده است.

```
mexican_hat = MexicanHat(r1=1, r2=3, c1=0.5, c2=-0.2)
mexican_hat.iterate([0.27, 0.35, 0.44, 0.58, 0.66, 0.77, 0.4, 0.32, 0.20, 0.15, 0.08])
```

```
100% 10/10 [00:00<00:00, 135.13it/s]

(4, 0.66)
```



شکل 12 - الگوریتم و نتیجه کلاه مکزیکی ب

با این مقادیر ب، و همچنین پس از تنظیم هایپرپارامترهای دیگر، در این بخش، ایندکس 4ام با مقدار 0.66 بیشترین شد.

در هر بار تکرار نمودار Index اعضای آرایه و مقدار سیگنال خروجی را رسم کرده و در نهایت نتایج دو قسمت را مقایسه کنید tmax (و سایر پارامترها را مقادیر مناسب در نظر بگیرید).

همانطور که دیدیم، در حالت اولی مقدار 0.77 بیشترین شد ولی در دومی 0.66. روش کلاه مکزیکی به دنبال یافتن بیشینه نرم (به جای سخت در maxnet) است. به این صورت که تا شعاع r1 اطراف نودهای کمک کننده و cooperative هستند و از r1 تا r2 در تنازع competitive هستند. به این صورت عین مثال استاد، دنبال ناحیه ای می‌گردیم که بیشترین‌ها در آن هستند و لزوما این بخش بیشترین مقدار ندارد.

در قسمت الف که r2 را بی نهایت گذاشتیم، و r1 را 0 گذاشتیم، یعنی هر مقدار فقط خودش به خودش کمک می‌کند و از تمام باقی مقادیر ضرر می‌بیند. به این ترتیب ناحیه به 0 رسیده و تنها مقداری که بیشترین در کل است پیدا می‌شود که همان 0.77 که دیدیم است.

اما در قسمت ب که 1 و 3 گذاشتیم ناحیه نرم‌تری را پیدا می‌کنیم. همانطور که در شکل هم

دیده می‌شود، در ابتدای کار، مقادیر کمی چولگی دارند و سمت چپ در مجموع مقادیر نسبت به چپ دارد. به همین دلیل کلاه مکزیکی که مقدار ماکسیمم نرم را پیدا می‌کند، سمت ایندکس 4 با مقدار 0.66 را پیدا کرده است. این بیشینه مقدار نیست، اما طبق شکل می‌توان دید که نقطه‌ای است که می‌توان گفت اکثر مقادیر بزرگ در اطراف آن هستند.

در مجموع، با اجرای الگوریتم، ناحیه های با مقدار بیشتر بالاتر می‌روند و ناحیه های با مقدار کم و مخصوصا وقتی یکم دورترشان مقادیر بزرگ باشد کوچکتر می‌شود که همان شکل کلاه مکزیکی که نام الگوریتم است را می‌سازد. ضمنا بخاطر تابع فعالسازی که استفاده کردیم، مقادیر هیچگاه از 0 تا 2 خارج نمی‌شوند.

سوال 4 – Hamming Net

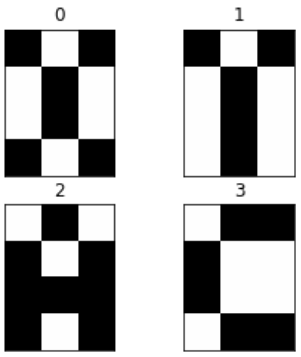
الف) تصاویر زیر را در چند آرایه از نوع numpy ذخیره و فاصله Hamming را برای هر جفت از آنها محاسبه کنید. خانه های سیاه 1 و خانه های سفید 1- در نظر بگیرید!

ابتدا آرایه‌ها را متناظر با تصاویر می‌سازیم و نمایش می‌دهیم.

```
exemplars = np.array([
    [+1, -1, +1,
     -1, +1, -1,
     -1, +1, -1,
     +1, -1, +1,],
    [+1, -1, +1,
     -1, +1, -1,
     -1, +1, -1,
     -1, +1, -1,],
    [-1, +1, -1,
     +1, -1, +1,
     +1, +1, +1,
     +1, -1, +1,],
    [-1, +1, +1,
     +1, -1, -1,
     +1, -1, -1,
     -1, +1, +1,],
])

def draw(image):
    image = (image.reshape(4, 3) * -1 + 1) * 127
    img = Image.fromarray(image)
    plt.imshow(img)

plt.figure(figsize=(4, 4))
for i in range(len(exemplars)):
    plt.subplot(2, 2, i + 1)
    plt.gca().set_title(i)
    plt.xticks([])
    plt.yticks([])
    draw(exemplars[i])
plt.show()
```



شکل 13 - ذخیره‌سازی تصاویر در آرایه

سپس فاصله hamming را محاسبه می‌کنیم.

```
def hamming_distance(a, b):
    return (a != b).sum()

for i in range(len(exemplars)):
    for j in range(len(exemplars)):
        print(f"HammingDistance({i}, {j}): {hamming_distance(exemplars[i], exemplars[j])}")
```

```
HammingDistance(0, 0): 0
HammingDistance(0, 1): 3
HammingDistance(0, 2): 8
HammingDistance(0, 3): 8
HammingDistance(1, 0): 3
HammingDistance(1, 1): 0
HammingDistance(1, 2): 11
HammingDistance(1, 3): 7
HammingDistance(2, 0): 8
HammingDistance(2, 1): 11
HammingDistance(2, 2): 0
HammingDistance(2, 3): 6
HammingDistance(3, 0): 8
HammingDistance(3, 1): 7
HammingDistance(3, 2): 6
HammingDistance(3, 3): 0
```

شکل 14 - فاصله همینگ نمونه‌ها

طبیعتا فاصله hamming بین x,y و y,x تفاوتی ندارد و برابر شده است. و ضمنا فاصله هر x با خودش نیز طبیعتا 0 است.

ب) 4 تصویر قسمت اول را به عنوان نمونه ها (exemplars) شبکه Hamming-net در نظر گرفته و با فرض n = 12 (ابعاد داده ورودی شبکه)، ماتریس وزن ها و بایاس شبکه Hamming-net را به دست آورید.

ابتدای الگوریتم را با محاسبه و نمایش ماتریس وزنها و بایاس آغاز می‌کنیم.

```
class HammingNet:
    def __init__(self, exemplars):
        self.weights = exemplars / 2  # [4, 12]
        self.biases = np.ones(len(exemplars)) * len(exemplars[0]) / 2
        print("weights", self.weights)
        print("biases ", self.biases)


hamming_net = HammingNet(exemplars)
```

```
weights [[ 0.5 -0.5  0.5 -0.5  0.5 -0.5 -0.5  0.5 -0.5  0.5 -0.5  0.5]
 [ 0.5 -0.5  0.5 -0.5  0.5 -0.5 -0.5  0.5 -0.5 -0.5  0.5 -0.5]
 [-0.5  0.5 -0.5  0.5 -0.5  0.5  0.5  0.5  0.5 -0.5  0.5  0.5]
 [-0.5  0.5  0.5  0.5 -0.5 -0.5  0.5 -0.5 -0.5 -0.5  0.5  0.5]]
biases  [6. 6. 6. 6.]
```

شکل 15 - محاسبه وزنها و بایاس

پ) با استفاده از ماتریس وزن و بایاس به دست آمده در مرحله قبل خروجی های شبکه را برای ورودی زیر محاسبه کنید. بر اساس خروجی های به دست آمده ورودی به کدام یک از تصاویر (X,Y,A,C) نزدیک تر است.

پیاده سازی کامل الگوریتم و نتیجه آن برای ورودی خواسته در زیر آمده است.

```
x_test = np.array(
    [+1, +1, +1,
     -1, +1, -1,
     -1, +1, -1,
     -1, +1, -1,],
)
idx_to_letter = {0: "X", 1: "Y", 2: "A", 3: "C"}
```

```
class MaxNet:
    def __init__(self, epsilon=0.15):
        self.epsilon = epsilon

    def f(self, x):
        return max(0, x)

    def max(self, x):
        a = np.array(x)
        while (a > 0).sum() > 1:
            a_old = a.copy()
            for i in range(len(a)):
                a[i] = self.f(a_old[i] - self.epsilon * np.sum(np.delete(a_old, i)))
        return np.argmax(a), x[np.argmax(a)]

class HammingNet:
    def __init__(self, exemplars):
        self.weights = exemplars / 2  # [4, 12]
        self.biases = np.ones(len(exemplars)) * len(exemplars[0]) / 2

    def best_match(self, x):
        y = self.weights @ x
        maxnet = MaxNet(epsilon=0.1)
        max_i, max_v = maxnet.max(y)
        return idx_to_letter[max_i]


hamming_net = HammingNet(exemplars)
hamming_net.best_match(x_test)
```

```
'Y'
```

شکل 16 - الگوریتم hamminnet و نتیجه آن

همانطور که دیده می‌شود مدل تشخیص داده است که ورودی داده شده بیشتر از همه به Y شبیه است که تشخیص درستی است. این کار با انجام یک ضرب داخلی بین ورودی و تمام وزنهای مربوط به هر نورون انجام شده و سپس یک maxnet آن نورونی که بیشتر از همه مشابهت داشته را انتخاب کرده است که همان نورون مربوط به Y بوده است.