به نام خدا

درس مبانی هوش محاسباتی

تمرین سری دوم

مدرس درس:
جناب آقای دکتر مزینی

تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۰۹/۱۱

سوال ۱ :

تابع زیر را در نظر داشته باشید :

$$y = -1 + (\frac{2}{3}) * sin(2x * \pi) + L$$

$$0 < x < 2$$

$$-0.8 < L < 0.8$$

حال با توجه به نمونه برداری زوج‌های x،y را درست کنید.
در این مرحله، از زوج‌هایی که درست کردیم به عنوان داده‌های آموزش و ارزشیابی استفاده می‌کنیم.
شبکه RBF را در نظر بگیرید.
می‌خواهیم شبکه RBF درست شده را روی دیتاستی که در بالا درست کردیم، آموزش دهیم و نتایج داده‌های ارزشیابی را بررسی کنیم.
همانطور که در درس یاد گرفتیم، برای پیدا کردن مرکزها چندین روش معروف داریم. شبکه RBFای که داریم را توسط این ۳ نوع الگوریتم زیر پیاده‌سازی کنید و نتایج هر کدام به همراه نمودارهای آن‌ها را بررسی کنید.

1. K-means
2. GMM
3. Random

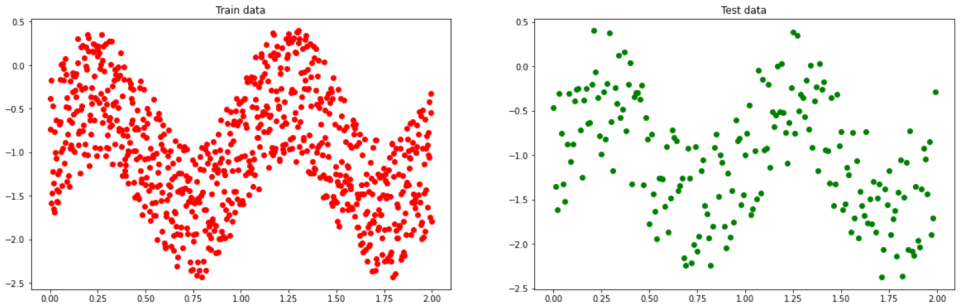
حال، در این قسمت نتایج حالت‌های بالا را با هم مقایسه کنید و نقاط ضعف و قوت هر روش را ذکر کنید. همچنین درباره کاربرد به خصوص هر روش توضیح دهید.
به نظرتون اگر به جای استفاده از یک شبکه RBF از یک شبکه MLP استفاده می‌کردیم، نتایج چگونه می‌شد؟ به طور کامل توضیح دهید.

پاسخ ۱:

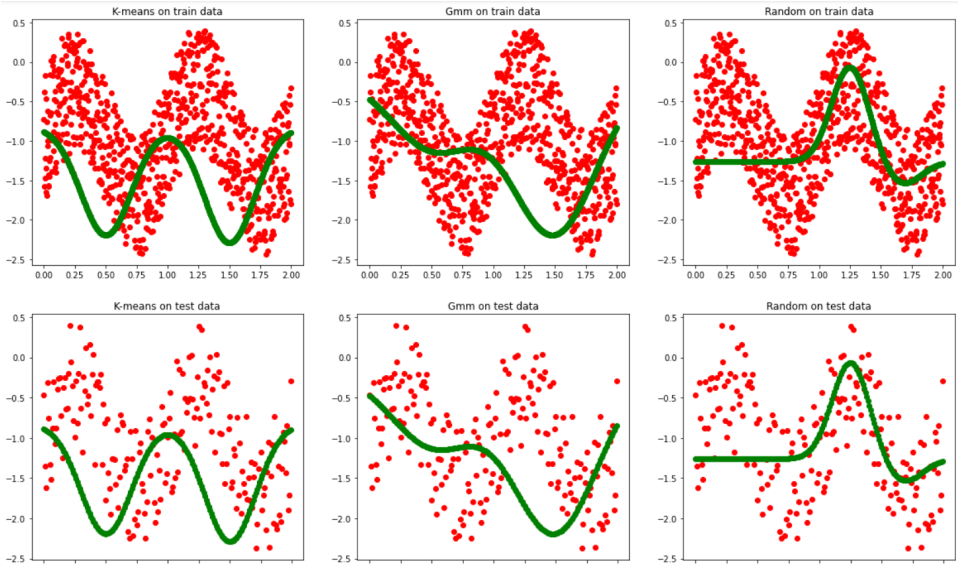
زوج‌های x و y را با استفاده از کد زیر برای داده‌های train و test می‌سازیم.

```python
# train = 800
x_train = np.arange(0, 2, 1/400)
y_train = (-1 + ((2/3) * np.sin(2 * x_train * (np.pi))) + (1.6 * np.random.random_sample((800, )) - 0.8))
x_train_rbf = x_train.reshape(800,1)
y_train_rbf = y_train.reshape(800,1)
# test = 200
x_test = np.arange(0, 2, 0.01)
y_test = (-1 + ((2/3) * np.sin(2 * x_test * (np.pi))) + (1.6 * np.random.random_sample((200, )) - 0.8))
x_test_rbf = x_test.reshape(200,1)
y_test_rbf = y_test.reshape(200,1)
```

در این سوال، ۸۰۰ نمونه به عنوان داده train و ۲۰۰ نمونه، برای داده آموزشی استفاده شده است. نمونه‌های ما، به صورت شکل زیر شدند.

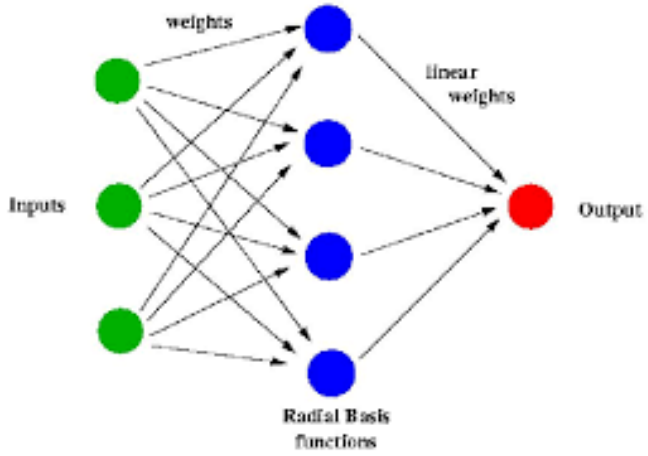


در ادامه با استفاده از شبکه RBF، مدل خود را آموزش می‌دهیم. طبق صورت سوال، این شبکه را با استفاده از الگوریتم‌های k-means، GMM و Random train می‌کنیم. نتایج هر سه الگوریتم در تصویر زیر آورده شده است.



حال به طور مختصر توضیحی در مورد RBF داده و سپس به بررسی هر یک از این الگوریتم‌های آن می‌پردازیم.

در این روش با استفاده از تغییری بعد یک ورودی سعی در دسته‌بندی کردن اطلاعات داریم، در واقع همان کاری که در MLP توسط چندین لایه میانی اتجام می‌شد ، حاال می‌توان با استفاده از تعدادی Radial Basis Function اتجام شود. ساختار کلی این روش دارای ۳ لایه است ، لایه ورودی، یک لایه میانی شامل تعدادی RBF و در نهایت هم لایه خروجی را داریم. لایه خروجی عملکرد بسیار ساده‌ای دارد و در واقع یک Perceptron است. به شکل زیر توجه کنید:



حال سوال که پیش می‌آید این است که هر یک از این RBFها چه عملکردی دارند. برای این مورد، چند الگوریتم متفاوت وجود دارد که در ادامه به معرفی هر یک می‌پردازیم.

۱. K-means: الگوریتم K-means بر اساس پارامتر چگالی برای تعیین مرکز خوشه‌بندی با هدف بهبود نرخ تمرین RBF معرفی شد. برای جلوگیری از overfitting در RBF، باید چند نمونه اولیه (نقاط نماینده) را به جای همه نقاط انتخاب کنیم. K-means یک الگوریتم مفید برای انتخاب آن نمونه های اولیه است. بنابراین می توانیم شبکه K-means و RBF را به صورت زیر ترکیب کنیم:

- مقداردهی اولیهk نقطه تصادفی به عنوان مرکز
- optimize کردن خطا به طور مکرر

- بهینه‌سازی زیرمجموعه: هر x با استفاده از نزدیک‌ترین u به‌طور بهینه تقسیم‌بندی می‌شود
- بهینه‌سازی مرکزها
- تکرار مراحل ۲ به بعد تا رسیدن به converge نهایی

از کاربردهای این روش، می‌توان به تقسیم بندی بازار، خوشه بندی اسناد، تقسیم بندی تصویر و فشرده سازی تصویر و غیره اشاره کرد.
مزایای این روش عبارتند از:

- اجرای نسبتا ساده
- مناسب برای مجموعه داده‌های بزرگ
- تضمین همگرایی
- سازگاری آسان با نمونه‌های جدید
- به خوشه هایی با اشکال و اندازه های مختلف، مانند خوشه های بیضی تعمیم می یابد

از معایب این روش نیز می‌توان موارد زیر را بیان کرد:

- انتخاب k به صورت دستی
- وابسته بودن به مقادیر اولیه
- خوشه بندی داده ها با اندازه ها و چگالی های مختلف
- خوشه بندی نقاط پرت

۲. GMM: همانطور که از نام آن پیداست، هر خوشه بر اساس توزیع گاوسی متفاوتی مدل‌سازی می‌شود. این رویکرد انعطاف‌پذیر و احتمالی برای مدل‌سازی داده‌ها به این معنی است که به جای داشتن تخصیص سخت در خوشه‌هایی مانند k-means، ما تخصیص‌های نرم داریم. این بدان معنی است که هر نقطه داده می تواند توسط هر یک از توزیع ها با احتمال مربوطه تولید شود. در واقع، هر توزیع مسئولیتی برای تولید یک نقطه داده خاص دارد. مراحل این الگوریتم، به شرح زیر است:

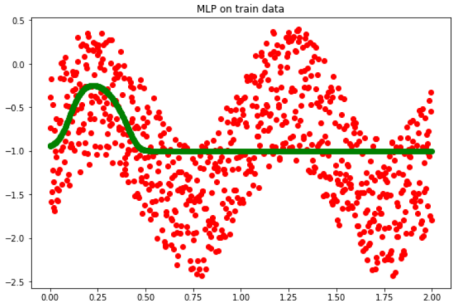
- تعیین یک ماتریس covariance برای تعریف ارتباط هر Guassian با یکدیگر (هر چه دو گاوسی شبیه‌تر باشند، میانگین آنها نزدیک‌تر خواهد بود)
- تعیین تعداد خوشه‌ها با تعیین Guassian در هر گروه
- انتخاب هاییhyperparameter که نحوه جداسازی بهینه داده‌ها را با استفاده از مدل‌های mixture Gaussian تعریف می‌کنند.

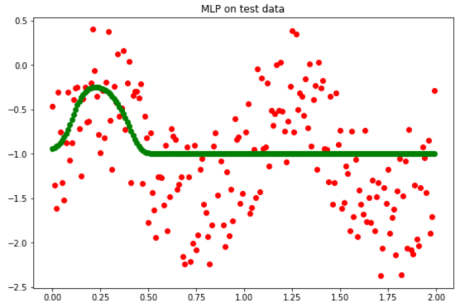
کاربرد این روش نیز در تشخیص ناهنجاری، پیش‌بینی قیمت سهام، استفاده برای مسائل دسته‌بندی و غیره می‌باشد.

۳. Random: در این روش به صورت رندوم دسته بندی را انجام می‌دهیم و از method خاصی استفاده نمی‌کنیم. کاربرد این روش نیز، برای زمان‌هایی است که اطلاعات دقیقی از شبکه نداریم.

حال به توضیح نتایج حاصل شده می‌پردازیم. k-means از تمامی الگوریتم‌های دیگر عملکرد بهتری داشت و مقادیر مشابهی با تابع اصلی دارد. GMM هم عملکرد نسبتا خوبی داشت اما Random فقط برای یک سری از داده‌ها خوب عمل کرد. این نتایج، با توجه به توضیحات داده شده در بالا، منطقی به نظر می‌رسد.

حال مدل خود را با استفاده از MLP تعریف می‌کنیم. MLP، یک لایه ورودی ،یک لایه خروجی و تعدادی لایه میان (Hidden Layers) دارد. خروجی هر لایه به لایه بعدی می‌رود (For-ward Propagation) و در انتها به محاسبه Error نهایی در لایه آخر، این Error به شکل Back Propagation به لایه‌های پیشین باز می‌گردد و به همین شکل وزن‌ها را Update می‌کنیم.





همانطور که در شکل بالا نیز مشخص است، MLP عملکرد خوبی ندارد. همچنین سرعت training در MLP کمتر است؛ دلیل این اتفاق، این است که MLP نویز را یاد نمی‌گیرد و به خاطر همین، هم بیشتر طول می‌کشد تا یاد بگیرد و هم عملکرد ضعیفی دارد. شبکه RBF به دلیل استفاده از توابع radial، نویز را یاد گرفته و عملکرد سریع‌تری دارند.

نکته: در حل این سوال، از لینک‌های زیر استفاده شد.

https://github.com/mohammadym/Computational_Intelligence/blob/main/2/CI992-HW2/CI992_HW2.ipynb

https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

https://medium.com/geekculture/rbf-network-and-k-means-8c4466c32100 https://towardsdatascience.com/gaussian-mixture-modelling-gmm-833c88587c7f

https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages

سوال ۲:

فرض کنید ورودی‌های $x_1, x_2, ..., x_n$ قابل ذخیره کردن باشند. اگر مینیمم‌های محلی شبکه‌ی هاپفیلد دقیقا همین ورودی‌ها باشند،

آیا لیست [(1,1,1,1), (-1,-1,-1,-1), (1,1,-1,-1), (-1,-1,1,1)] قابل ذخیره‌سازی است؟ توضیح دهید و اگر امکان‌پذیر است، وزن‌های شبکه را محاسبه کنید.

پاسخ ۲:

برای حل این سوال، ابتدا وزن متناسب با هر pattern را به دست آورده و سپس آن‌ها را جمع می‌کنیم.

$$w_{i,j}^k = x_i^k x_j^k$$

$pattern1 = (1, 1, 1, 1)$

i/j	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

$pattern2 = (-1, -1, -1, -1)$

i/j	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

$pattern3 = (1, 1, -1, -1)$

i/j	1	2	3	4
1	0	1	-1	-1
2	1	0	-1	-1
3	-1	-1	0	1
4	-1	-1	1	0

$pattern4 = (-1, -1, 1, 1)$

i/j	1	2	3	4
1	0	1	-1	-1
2	1	0	-1	-1
3	-1	-1	0	1
4	-1	-1	1	0

برای به دست آوردن وزن کلی، کافیست از رابطه زیر استفاده کنیم:

$$w_{i,j} = \sum_{k=1}^{k} w_{i,j}^{k}$$

بنابراین، مقادیر وزن‌ها، به صورت روبرو در می‌آید.

i/j	1	2	3	4
1	0	4	0	0
2	4	0	0	0
3	0	0	0	4
4	0	0	4	0

حال، برای به دست آوردن انرژی هر pattern، از رابطه زیر استفاده می‌کنیم.

$$E = -\sum_{i,j} w_{i,j} o_i o_j$$

طبق فرمول بالا داریم:

$$E(pattern1) = -(4 + 4 + 4 + 4) = -16$$
$$E(pattern2) = -(4 + 4 + 4 + 4) = -16$$
$$E(pattern3) = -(4 + 4 + 4 + 4) = -16$$

$$E(pattern4) = -(4+4+4+4) = -16$$

با توجه به مقادیر به دست آمده، نتیجه می‌گیریم می‌توانیم لیست داده شده را ذخیره کنیم؛ چرا که برای ذخیره‌سازی، کافیست پایین‌ترین سطح انرژی را داشته باشیم و برای ماتریس وزن بالا، پایین‌ترین سطح انرژی، ۱۶- می‌باشد.

سوال ۳:

۱. مسئله فروشنده دوره‌گرد یا Traveling Salesman Problem را در نظر بگیرید. در این مسئله تعدادی شهر داریم و هزینه‌ی رفتن مستقیم از یکی به دیگری را می‌دانیم. مطلوب است کم هزینه‌ترین مسیری که از یک شهر شروع می‌شود و از تمامی شهرها دقیقا یکبار عبور کند و به شهر شروع بازگردد. توضیح دهید این مسئله با کدام یک از شبکه‌های عصبی که تا کنون خوانده‌اید قابل حل است. در صورتی که این مسئله با شبکه‌ای قابل حل بود، الگوریتم، ساختار شبکه و سایر توضیحات را ارائه دهید. در صورت غیر قابل حل بودن نیز دلیل خود را توضیح دهید.

Hopfield, RBF, SOM, MLP

۲. فایل cities.csv درون پوشه تمرین شامل مختصات تعدادی شهر است. شما می‌بایست این سوال را با یکی از روش‌های قسمت قبل پیاده‌سازی کنید. در انتها پس از یافتن مسیر بهینه توسط این الگوریتم مسیر را روی یک نمودار نمایش دهید. لازم است هنگام آموزش از مسیر یافته شده در Epochهای مختلف حداقل پنج تصویر قرار دهید.

فایل دیتا در فایل پروژه شامل مختصات تعدادی شهر است. هدف در این سوال این است که تمامی این شهر ها بازدید شود به شرط اینکه نقطه شروع و پایان یکی باشد و از هر شهر فقط یک بار عبور کند. شما می‌بایست این سوال را با الگوریتم Kohonen پیاده سازی کنید. در انتها پس از یافتن مسیر بهینه توسط این الگوریتم مسیر را روی یک نمودار نمایش دهید. در شکل نمونه‌ای از مسیر یافت شده را می‌توانید ببینید.

پاسخ ۳:

۱. این مسئله با استفاده از Hopfield قابل حل می‌باشد. به این صورت که برای هر شهر، یک نورون در شبکه هاپفیلد در نظر می‌گیریم. همچنین برای مقدار دهی اولیه وزن‌ها نیز از فاصله بین شهرها استفاده می‌کنیم. سپس شبکه خود را train کرده و اگر یک وزن ۱ بشود، جایگاه آن شهر را در مسیر نشان می‌دهد. زمانی که یکی از نورون‌های شبکه در جایگاه یک نورون متفاوت ۱ شود، نشان دهنده رسیدن به مسیر می‌باشد و همچنین آن مسیر، بهترین مسیر می‌شود.

حل این سوال با استفاده از RBF نیز ممکن است. به این صورت که در ابتدا کمترین و بیشترین فاصله بین دو شهر را پیدا کرده تا طول حداقلی و حداکثری تور بین شهرها را پیدا کنیم. سپس این فاصله را به بازه‌های satisfactory تقسیم کرده و مرکز و عرض آن را مشخص می‌کنیم. در ادامه در هر فاصله، تورهای satisfactory با کمترین فاصله بین شهرها را پیدا کرده و می‌توانیم به مسیر مورد نظرمان برسیم.

از SOM نیز می‌توانیم برای حل این سوال استفاده کنیم. نخست، یک شبکه با تعداد نورون‌های برابر با تعداد شهرها می‌سازیم. برای بردار وزن هر نورون، ۲ عنصر داریم. ورودی‌های شبکه در اینجا، مختصات شهرهایمان هستند که بهتر است نرمالیزه شوند. وزن‌های شبکه را نیز در ابتدا به صورت تصادفی initialize می‌کنیم. در هر بار، یک نورون انتخاب می‌شود و شبکه شکل می‌گیرد. بنابراین برای هر شهر، یک نورون برنده داریم که مکان آن شهر در مسیر را مشخص می‌کند.

حل این سوال با استفاده از MLP غیر ممکن است. برای استفاده از این روش، باید label برای هر شهر داشته باشیم که چون در اینجا مکان شهرها در مسیر را نمی‌دانیم، پس نمی‌توانیم این مسئله را با این روش حل کنیم.

۲. برای حل این سوال، ابتدا فایل csv داده شده را خوانده و مختصات شهرها را لیست coordinates ذخیره می‌کنیم.

Find coordinates

```
data = open('Cities.csv', "r")
coordinates = []
x = []
y = []
for d in data:
    coordinate = list(map(float, d.split()[1:]))
    x.append(coordinate[1])
    y.append(coordinate[0])
    coordinates.append([coordinate[1], coordinate[0]])
coordinates = np.array(coordinates)
```

تعداد شهرها را با استفاده از قطعه کد زیر به دست می‌آوریم.

Find the number of cities

```
[5]  cities_number = coordinates.shape[0]
```

مختصات شهرها را normalize می‌کنیم تا مقداری بین ۰ و ۱ به ما بدهد.

Normalize the data

```
r =(max(x)-min(x)) / (max(y)-min(y)) , 1
ratio = np.array(r) / max(r)
normalized_coordinates = (coordinates - np.array([min(x), min(y)])) / ratio
```

تابعی برای پیدا کردن نورون برنده با توجه به شبکه و مختصات داده شده می‌نویسیم.

Select the nearest city

```
[7]  def nearest_city(network, coordinate):
       distance = np.linalg.norm(network - coordinate, axis=1)
       return np.where(distance == np.amin(distance))
```

سپس، با استفاده از تابع زیر، همسایگی را به دست می‌آوریم. o نورون برنده، r شعاع، و d نیز تعداد شهرها را مشخص می‌کند.

Find neighbors of a city

```
def find_neighbors(o, p, d):
    if p < 1:
      p = 1

    sigma = np.absolute(o - np.arange(d))
    distance = np.minimum(sigma, d - sigma)

    return np.exp(-(pow(distance, 2)) / (2*(pow(p, 2))))
```

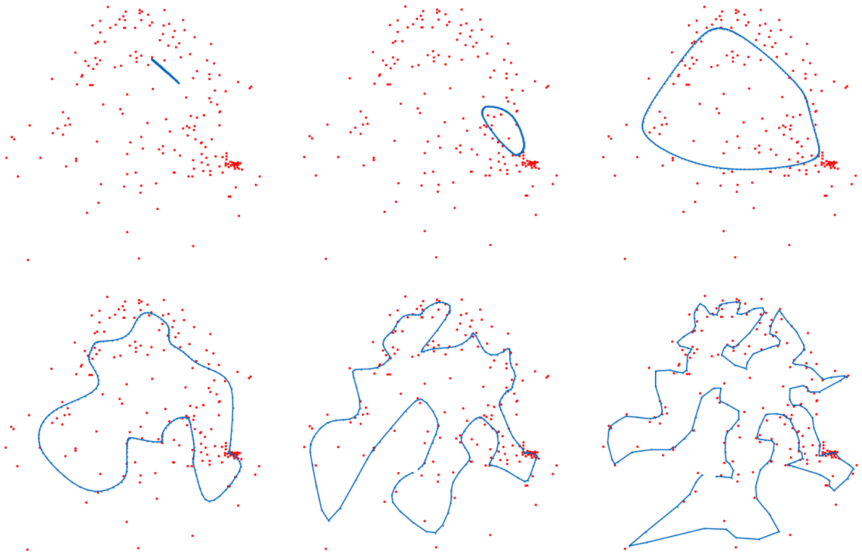
از تابع زیر نیز برای plot کردن شبکه استفاده می‌کنیم.

```
[9] def plot_city_network(network, coordinates):
      fig = plt.figure(figsize=(5, 5), frameon = False)
      axis = fig.add_axes([0,0,1,1])
      axis.set_aspect('equal', adjustable='datalim')
      plt.axis('off')
      axis.scatter(coordinates[:, 0], coordinates[:, 1], color='red', s=4)
      axis.plot(network[:,0], network[:,1], 'r.', ls='-', color='#0063ba', markersize=2)
      plt.show()
```

سپس، با استفاده از قطعه کد زیر، شبکه خود را آموزش می‌دهیم. در اینجا learning rate را ۰.۸ در نظر گرفتیم.

```
lr = 0.8
r = cities_number * 8

# Kohonen network
kohonen_network = np.random.rand(cities_number, 2)

for i in range(100000):
  if not i % 100:
    print('\t> Iteration {}/{}'.format(i, 100000), end="\r")
  random_city = np.random.randint(0, cities_number)
  winner = nearest_city(kohonen_network, normalized_coordinates[random_city])[0][0]
  neighbours = find_neighbors(winner, r//10, cities_number)
  kohonen_network += neighbours[:,np.newaxis] * lr * (normalized_coordinates[random_city] - kohonen_network)
  lr = lr * 0.99997
  r = r * 0.9997

  if not i % 1000:
    print(f'epoch {i+1} : Cities and Network')
    plot_city_network(kohonen_network, normalized_coordinates)

  if r < 1:
    print('Radius has completely decayed at {} iterations'.format(i))
    break

  if lr < 0.001:
    print('Learning rate has completely decayed at {} iterations'.format(i))
    break
```

بخشی از تصاویر حاصل از آموزش شبکه در ادامه آورده شده‌اند.



نکته: برای حل این سوال، از لینک زیر کمک گرفته شد:

https://github.com/diego-vicente/ntnu-som/tree/f28a218b7f30627c8e767b2308b5b40773785387/src