به نام خدا

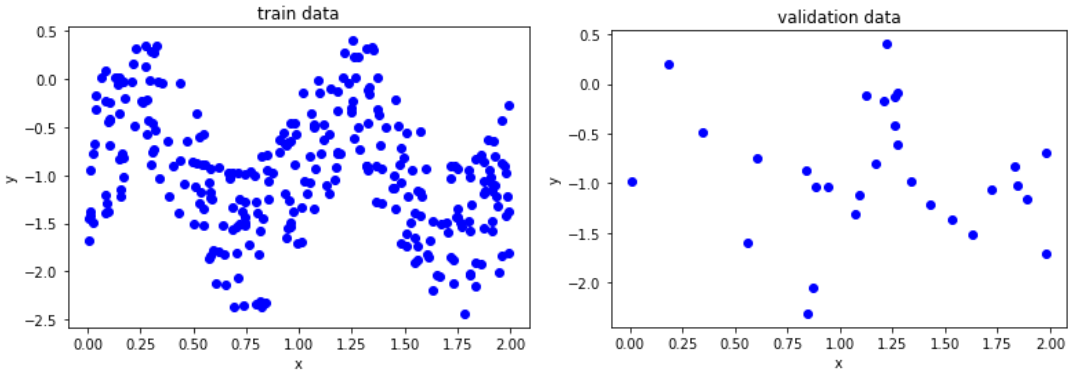
نام: فاطمه زهرا بخشنده **استاد: دکتر ناصر مزینی**

شماره دانشجویی: 98522157 **درس مبانی هوش محاسباتی**

گزارش تمرین 2:

سوال اول:

ابتدا تابع generate_data را می نویسیم که نمونه برداری کرده و x و y را میسازد. به کمک آن، 300 نمونه آموزشی و 30 نمونه تست آماده می کنیم. که به صورت زیر هستند:



1. مدل RBF

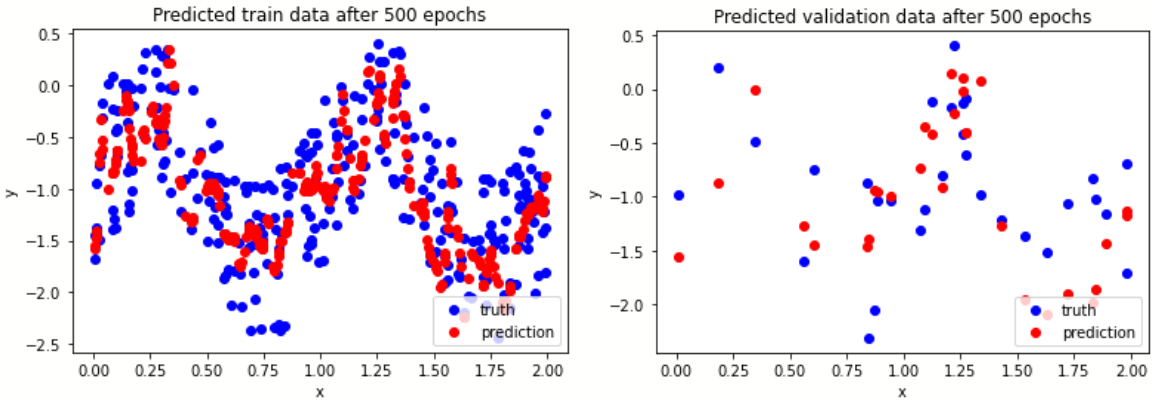
کلاس مدل RBF را میسازیم و در آن متد های مورد نیاز را پیاده سازی می کنیم. وزن ها را به صورت رندوم initialize می کنیم.

ویژگی cluster_method را برای این کلاس قرار می دهیم که در هنگام ساخت کلاس مقدار دهی می شود و می تواند kmeans، gmm یا random باشد. در متد train هنگام آموزش شبکه، برای پیدا کردن center ها ابتدا cluster_method را بررسی می کنیم و طبق آن متد مورد نظر را برای پیدا کردن مراکز، فراخوانی می کنیم. برای روش kmean و gmm از sklearn.cluster استفاده می کنیم. و به کمک آن ها متد هر کدام را پیاده سازی می کنیم. برای حالت random نیز به صورت رندوم مراکز را از دیتا انتخاب می‌کنیم.

ارور را با روش MSE محاسبه می کنیم. پس از ساخت کلاس نوبت به آموزش داده ها با مدل RBF می رسد.

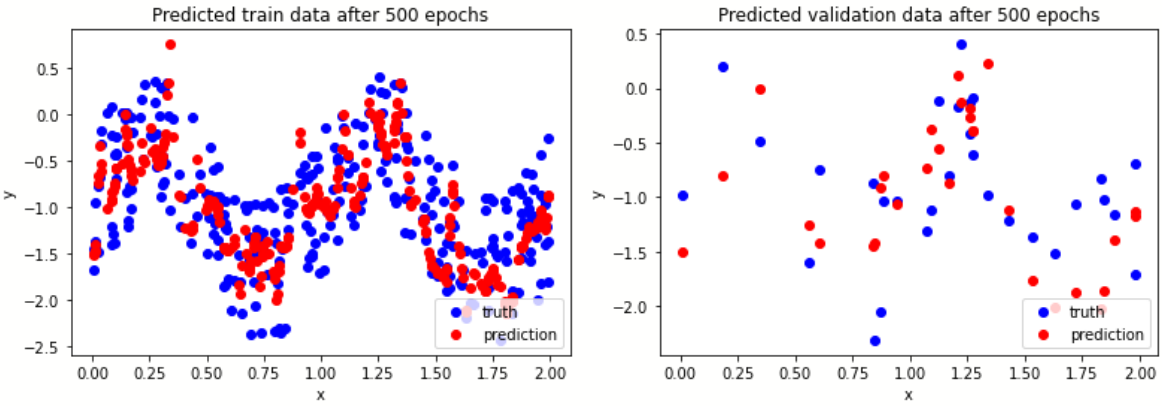
1.1. روش K-Means

ابتدا یک آبجکت از کلاس میسازیم که cluster_method برای آن kmeans باشد. این مدل را روی داده‌های train با 500 تا epoch آموزش داده، سپس متد predict را روی داده آموزشی و ارزیابی فراخوانی می‌کنیم. نتایج:



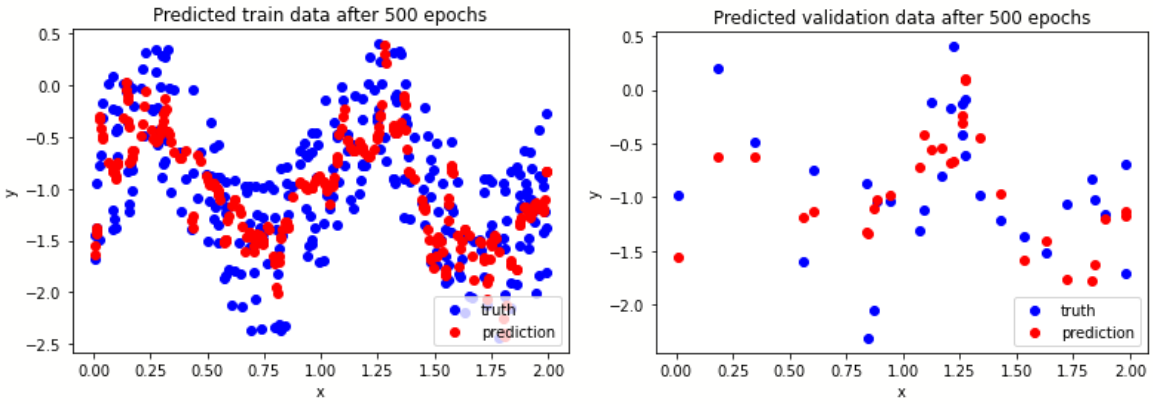
1.2. روش GMM

این بار یک آبجکت از این کلاس با cluster_method برابر gmm می‌سازیم. این مدل را روی داده‌های train با 500 تا epoch آموزش داده، سپس متد predict را روی داده آموزشی و ارزیابی فراخوانی می‌کنیم. نتایج:



1.3 روش Random

این دفعه cluster_method را برابر random قرار می دهیم. این مدل را روی داده‌های train با 500 تا epoch آموزش داده، سپس متد predict را روی داده آموزشی و ارزیابی فراخوانی می‌کنیم. نتایج:



مقایسه: روش random که مشخص است به صورت رندوم centers را مشخص می کند. همانطور که از شکل ها نیز مشخص است، دو روش دیگر robust تر بوده و پیشرفته تر عمل می کنند. بین این دو روش به نظر می‌رسد که Gaussian mixtures قوی‌تر هستند. با این حال، GMM ها معمولاً نسبت به K-Means کندتر هستند، زیرا برای رسیدن به همگرایی، تکرارهای بیشتری از الگوریتم EM نیاز است. آنها همچنین می توانند به سرعت به local minimun همگرا شوند که چندان مطلوب نیست.

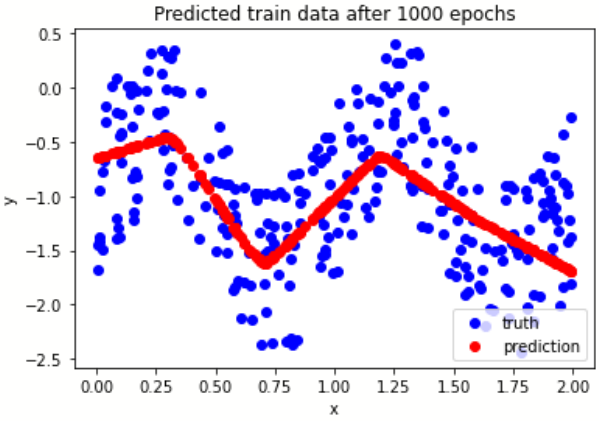
به همین دلیل نیز در روشی پیشرفته تر، از K-Means + Gaussian Mixtures استفاده می شود. به طوری که GMM ها با K-Means مقداردهی اولیه می شوند. این روش معمولاً به خوبی کار می کند و خوشه های تولید شده با K-Means را بهبود می بخشد.

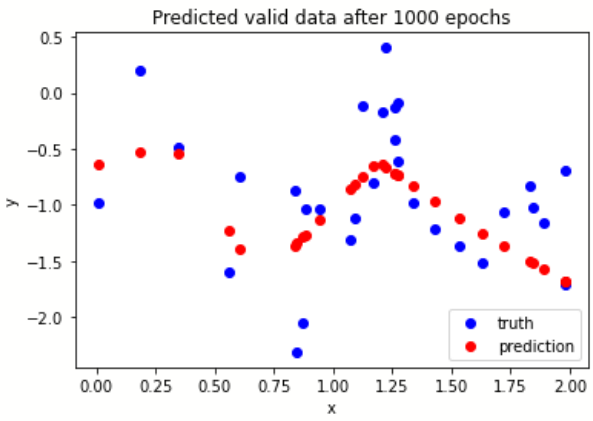
2. مدل MLP

مدل خود را به صورت زیر میسازیم:

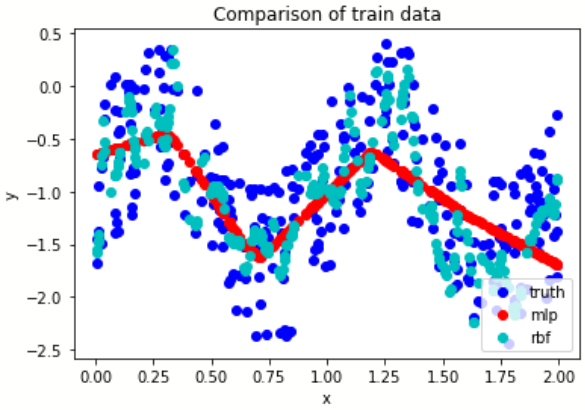
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, 1)]         0

 dense (Dense)               (None, None, 60)          120

 activation (Activation)     (None, None, 60)          0

 dense_1 (Dense)             (None, None, 40)          2440

 activation_1 (Activation)   (None, None, 40)          0

 dense_2 (Dense)             (None, None, 30)          1230

 activation_2 (Activation)   (None, None, 30)          0

 dense_3 (Dense)             (None, None, 20)          620

 activation_3 (Activation)   (None, None, 20)          0

 dense_4 (Dense)             (None, None, 1)           21

=================================================================
Total params: 4,431
Trainable params: 4,431
Non-trainable params: 0
```

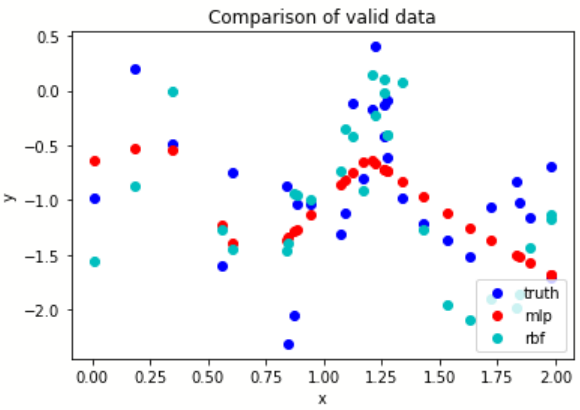
سپس آن را روی داده train با 1000 تا epoch آموزش می دهیم. و متد predict را روی داده آموزشی و ارزیابی فراخوانی می‌کنیم. نتایج:





3. مقایسه:





در کل نتیجه RBF دقیقتر است. برای چنین توابعی (غیر خطی) مدل Radial به علت استفاده از متد گاوسی بهتر عمل می کند.

داده تولید شده ما بخاطر L مقداری نویز دارد. با توجه به نتایج، MLP نویز را یاد نمی گیرد و فقط تابع سینوسی را یاد گرفته است. اما شبکه RBF بخاطر استفاده از توابع Radial میتواند نویز را یاد بگیرد. همچنین بخاطر وجود نویز طول می کشد تا MLP تابع را یاد بگیرد اما این مشکل در RBF وجود ندارد و آموزش سریع است.

تفاوت های دیگر: اگر در MLP چند نورون را حذف کنیم عملکرد دچار اختلال بزرگ نمی شود. MLP با تعداد نورون های زیاد به خوبی کار می کند.

منابع: لینک

سوال دوم:

ابتدا ماتریس وزن این 4 پترن را محاسبه می کنیم که با توجه به فرمول زیر بدست می‌آید.

$$w_{i,j} = \sum_{k=1}^{K} w_{i,j}^{k} = \sum_{k=1}^{4} x_i^k x_j^k$$

$$w_{i,i} = 0$$

$$w_{i,j} = w_{j,i}$$

$$K: number\ of\ patterns = 4$$

ماتریس وزن ها:

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

ورودی جدید را به مدل وارد می‌کنیم و پایدار بودن آن را بررسی می‌کنیم.

$$u(i, t+1) = \sum_{j=1}^{N} w_{i,j}\, a(j,t)$$

$$a(i, t+1) = sign(u(i, t+1))$$

طبق اسلاید 12 از sign به عنوان activation function استفاده می کنیم که به صورت زیر است.

•**Bipolar valued hard limiter {-1, 1}**

$$f_t(x) = \begin{cases} 1 & x > t \\ -1 & x \le t \end{cases}$$

چون ورودی های ما 1 و 1- هستند sign آن ها برابر خودشان است.

	1	2	3	4
$\boldsymbol{Input}(t = 0)$	x_1	x_2	x_3	x_4
$t = 1$	x_2	x_1	x_4	x_3
$t = 2$	x_1	x_2	x_3	x_4
$\sum_i \boldsymbol{X_i W_{ij}}$	$4x_2$	$4x_1$	$4x_4$	$4x_3$
	$4x_1$	$4x_2$	$4x_3$	$4x_4$

چون 4 عددی مثبت است وقتی از جدول پایین می‌خواهیم برای مثال $sign(4x_2)$ را در بالا قرار دهیم، این مقدار برابر با $sign(x_2)$ است که آن هم با توجه به توضیحات قبل، برابر با خود x_2 است.

قرار بود مینیمم‌های محلی شبکه‌ی هاپفیلد دقیقا همین ورودی‌ها باشند. در جدول هم مقادیر $t = 0$ با $t = 2$ برابر شد. پس در $t = 0$ و همچنین $t = 2$ شبکه پایدار است. پس این لیست قابل ذخیره سازی است.

منابع: اسلاید

سوال سوم:

در مسئله TSP تعدادی شهر داریم و می‌خواهیم کم هزینه ترین مسیری که از یک شهر شروع شود و از تمامی شهر ها دقیقا یکبار عبور کند و به شهر شروع بازگردد را پیدا کنیم.

- این مسئله با استفاده از mlp قابل حل نیست چون داده ما خروجی و لیبل مشخصی ندارد و جایگاه شهرها در مسیر را در اختیار نداریم پس نمی‌توانیم از آموزش supervised استفاده کنیم.
- این مسئله به کمک RBF قابل حل است. به این صورت که ابتدا کمترین و بیشترین فاصله میان دو شهر را پیدا می کنیم، تا حداقل و حداکثر طول توری که در نهایت میان شهرها پیدا میکنیم مشخص شود. حالا فاصله ای که داریم را به بازه های satisfactory تقسیم میکنیم و center و width آن را مشخص میکنیم. سپس در هر interval تورهای satisfactory که کمترین فاصله میان شهرها را دارند پیدا می‌کنیم تا در نهایت به مسیر مورد نظرمان برسیم. (منبع: لینک)
- این مسئله را با استفاده از Hopfield نیز می توانیم حل کنیم. ابتدا یک شبکه هاپفیلد با 194*194 یعنی 37636 نورون (به تعداد شهر ها به توان 2) میسازیم که هر نورون متناظر با یک المان در جدول بین شهر ها است. مانند شکل زیر: (منبع: لینک)

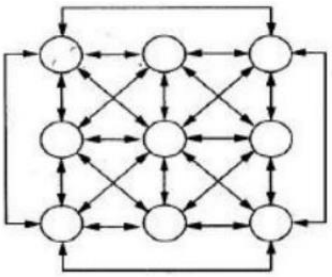


Figure 2 : Fully Connected Hopfield Network for 3 cities.

مقدار هر ورودی xi تعیین می شود و مجموع وزنی $\sum_i W_{ij}X_i$ همه ورودی ها محاسبه می شود. اگر این مقدار بزرگتر یا مساوی 0 باشد، خروجی نورون 1 می شود و اگر کوچکتر از 0 باشد، خروجی نورون 1- می شود. یک نورون حالت خروجی خود را تا زمانی که دوباره به روز شود حفظ می کند. به این ترتیب شبکه train می‌شود تا زمانی که انرژی به حداقل برسد و کوتاه ترین path را بیابیم.

- TSP با استفاده از SOM نیز قابل حل است. ابتدا یک شبکه SOM با 194 نورون (به تعداد شهر ها) پیاده سازی می کنیم. هر نورون 2 فیچر x و y شهر ها را دارد. ابتدا وزن های شبکه را به صورت رندوم تعریف می کنیم. ورودی های شبکه مختصات شهرها خواهند بود که بهتر است آنها را نرمال کنیم. پس از آموزش، هر شهر به یک نورون اختصاص داده می شود. سفر باید از شهر نورون شماره 0 شروع شود و به نورون 193 برسد. همچنین بازگشت به نورون اول را هنگام محاسبه فاصله ها باید در نظر بگیریم.

پیاده سازی با Kohonen:

روش حل با این روش در بخش قبل توضیح داده شد. ابتدا فایل را در کولب آپلود کرده، فایل csv را می خوانیم و مختصات x و y شهر ها را ذخیره می کنیم.

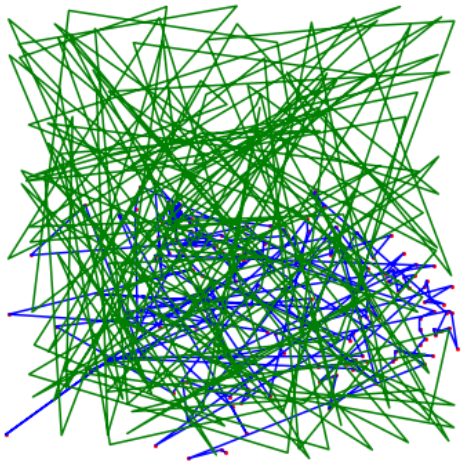
```
number of cities:  194

cities 1 to 4:
             x           y
1  24748.3333  50840.0000
2  24758.8889  51211.9444
3  24827.2222  51394.7222
4  24904.4444  51175.0000
```

سپس داده های مختصات را با کم کردن min و map کردن به بازه [0, 1] نرمال می کنیم.

کلاس مدل Kohonen را ساخته و متد های مورد نیاز را برای آن پیاده سازی می کنیم. شبکه را با شعاع همسایگی 5 و learning rate = 0.5 آموزش می دهیم. با توجه به اینکه شعاع همسایگی را در هر اپاک کمی کاهش می دهیم، شبکه می تواند زودتر به ثبات برسد. اگر تعداد اپاک مورد نظر تمام شود، یا شعاع همسایگی خیلی کوچک شود، آموزش به اتمام می رسد. پس از هر 200 اپاک تصویر شبکه را رسم می کنیم. 10 شکل رسم شده که 5 تای آن ها را اینجا گذاشتم:

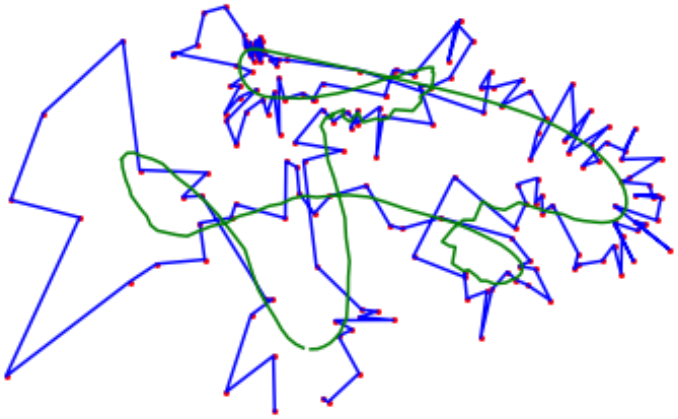
در حالت اولیه:

```
initialization complete
current path distance:  27.013821634049258
```



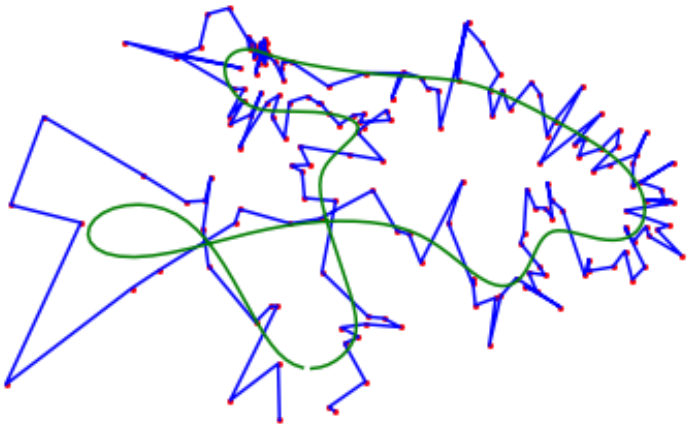
بعد از 200 epoch:

```
epoch  1   radius:  5
epoch  101   radius:  4.523960735568544
current path distance:  9.275041153977368
```



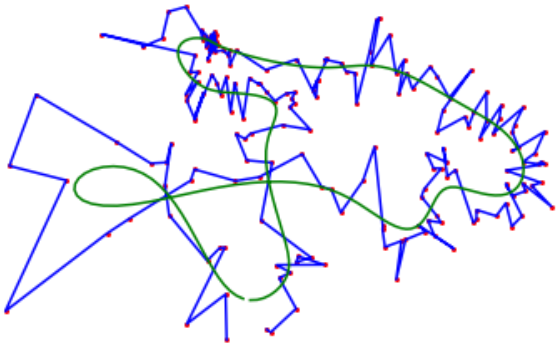
بعد از 600 epoch:

```
epoch  401   radius:  3.350929530033697
epoch  501   radius:  3.0318947243059187
current path distance:  8.714667845321426
```



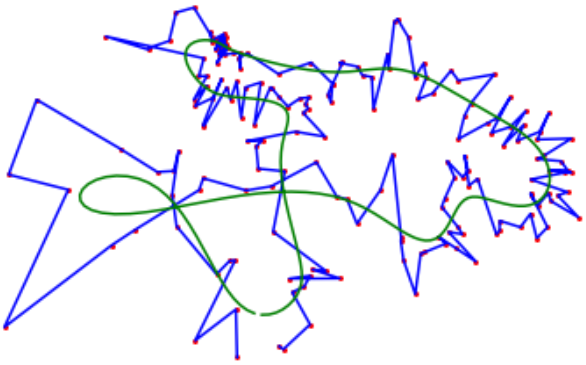
بعد از 1000 epoch:

```
epoch  801   radius:  2.2457457430503736
epoch  901   radius:  2.031933112726019
current path distance:  8.53071788396907
```



بعد از 1200 epoch:

```
epoch  1001   radius:  1.8384771238548159
epoch  1101   radius:  1.6634396643120348
current path distance:  8.4157995710166
```



بعد از 2000 epoch:

```
epoch  1801   radius:  0.8257504349184881
epoch  1901   radius:  0.7471325089899775
current path distance:  8.304764075993441
```