به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

گزارش پروژه درس سیستم های فازی و شبکه های عصبی

عنوان :

حل مسئله فروشنده دوره گرد با استفاده از شبکه عصبی SOM

استاد درس :

دکتر مهدی جلیلی

تهیه کننده :

تینا خواجه - 93210761

فهرست

مقدمه:

در این پروژه که پیاده سازی یکی روش های ارائه شده در یکی از مقالات مرتبط است، به حل مسئله فروشنده دوره گرد با استفاده از شبکه عصبی Self Organized Map (SOM) می پردازیم. هدف دیگر این پروژه علاوه بر حل این مسئله استفاده از متد های مقدار دهی اولیه و تعریف پارامترها و قوانین انطباق با هدف دستیابی به نتایج بهتر و سریع تر می باشد. مواردی همچون انتخاب تعداد نورون ها ، تاثیر نحوه ی چینش اولیه نورون ها، مقادیر وزن آنها، به روز رسانی پارامتر های مرتبط و... مورد مطالعه قرار می گیرد و در نهایت بعد از بررسی پیچیدگی الگوریتم ارائه شده با استفاده از شبکه عصبی SOM نتایج این روش در مقایسه با دیگر روش ها را برای نقشه های معروف به عنوان ورودی به مسئله فروشنده دوره گرد خواهیم دید.

تعریف مسئله فروشنده دوره گرد:

در این مسئله داده هایی که داریم عبارت هستند از لیست شهر ها و فاصله بین آنها و هدف پیدا کردن مسیری است که از تمام شهر ها بگذرد، از هر شهر نیز تنها یک بار عبور کند و در نهایت دوباره به شهر اولیه خود باز گردد.این مسئله از دسته مسائل NP_Hard محسوب می شود. مسئله فروشنده دوره گرد یک حالت خاص از مسئله ی Traveling Purchaser Problem محسوب می گردد.

مسئله فروشنده دوره گرد برای اولین بار در سال 1930 فرموله بندی شد و جزو مسائلی می باشد که تحقیقات و مطالعات بسیاری روی آن انجام شده است. همچنین یکی مسائلی است که در بهینه سازی به عنوان benchmark می توان آن را در نظر گرفت . هر چند که این مسئله از نظر محاسباتی سخت و پیچیده است تعداد زیادی روش برای حل آن ارائه شده است.

در این پروژه به حل مسئله فروشنده دوره گرد با استفاده از شبکه عصبی Self_Organized Map می پردازیم.

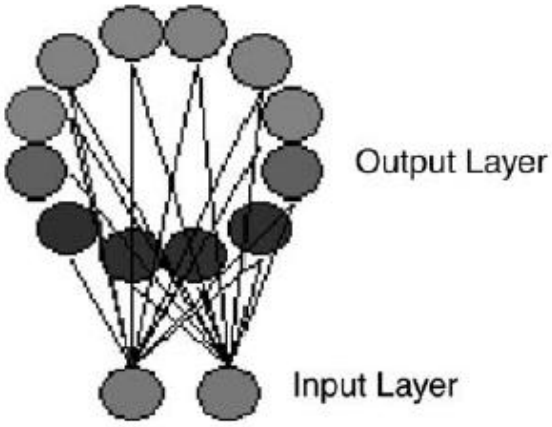
حل مسئله فروشنده دوره گرد با استفاده از شبکه عصبی SOM

از نظر تاریخی اولین روشی که برای TSP با استفاده از شبکه های عصبی ارائه شد با استفاده از شبکه Hopfield بود. این روش بر پایه ی کمینه کردن تابع انرژی استوار بود. روشی که در این پروژه ارائه می دهیم بر پایه ی شبکه عصبی Self Organized Map استوار خواهد بود.

انواع مختلفی روش ارائه شده با عنوان SOM وجود دارد که همه آنها ویژگی مشترکی های مشترکی دارند. در کل یادگیری Self Organized و یا یادگیری unsupervised شامل تغییرات بسیار در وزن های سیناپسی شبکه در پاسخ به مجموعه الگو های ورودی است. این تغییرات وزن ها بر اساس مجموعه قوانین یادگیری صورت می گیرند.

Kohonen self-organizing map به کلاسی خاص از شبکه های عصبی مربوط هستند که شبکه های عصبی رقابتی نام دارند. در این شبکه ها نورون هابرای داشتن فعالیت بیشتر با همدیگر رقابت می کنند. نتیجه این رقابت فعال شدن یک نورون در لایه خروجی خواهد بود. هدف Kohonen self-organizing map عبارت است از کسب شباهت و نحوه توزیع احتمال داده های ورودی.

جهت استفاده از این روش برای حل مسئله فروشنده دوره گرد یک شبکه دو لایه که ورودی دو بعدی به همراه m نورون در لایه خروجی دارد احتیاج داریم که نورون های لایه خروجی با فاصله شعاعی برابر بر روی دایره قرار گرفته اند. بر اثر گذر زمان و تکمیل شبکه می توان آن را به صورتی در نظر گرفت که به مرور زمان حلقه اولیه کشیده شده و بر روی شهر ها در نظر گرفته خواهد شد. شکل 2 ساختار گفته شده را ترسیم می کند. همچنین سعی شده است تا در حین توضیحات کد های پروژه مرتبط با هر قسمت را نیز اضافه نماییم.



شکل 1 : ساختار دو بعدی SOM

داده های ورودی در این حالت مختصات شهر ها هستند که بعد از ورود scale می شوند تا مقادیری بین 0 و یک داشته باشند.
وزن نورون های لایه خروجی مختصات آنها بر روی دایره واحد محسوب می گردند. (کد مربوط به قرار گیری نورون ها بر روی دایره به شعاع 0.5 و مرکز (0.5 , 0.5) که در فایل TSP.m قرار دارد به صورت زیر است)

```
for i=1 :neuronNum
    x=0.5*cos((i-1)*(2*pi/neuronNum))+0.5;
    y=0.5*sin((i-1)*(2*pi/neuronNum))+0.5;
    neorunCordinations(i,1)=x;
    neuronWeights(i,1)=x;
    neorunCordinations(i,2)=y;
    neuronWeights(i,2)=y;
end
```

داده های ورودی به صورت تصادفی به شبکه نشان داده می شوند و رقابتی بر اساس فاصله اقلیدسی بین نورون ها صورت می گیرد. در این حالت نورون برنده نورونی است مانند J که در کمترین فاصله را تا شهر نشان داده شده دارد.

$$J = \text{Argmin}_j\{\|x_i - y_j\|_2\}$$

که در فرمول بالا xi مختصات شهر، yi مختصات نورون و ||.|| فاصله اقلیدسی می باشد.

کد مربوط به این قسمت از پروژه در فایل findWinner.m قرار دارد که در زیر قابل مشاهده می باشد.

```
function winnerIndex = findWinner(input, weights,selctedNeurons)
%input - > is a sample input
%weights -> is a matrix containing all weights
 [rowW,colW]=size(weights);
 min=10000;
 minWeightIndex=0;
for i=1:rowW
    if(selctedNeurons(i,1)~=1)
        dif=norm(input-weights(i,:));

        if(dif<min)
            %distances are garantuee to be less than 1 because all cordinations are scaled
            min=dif;
            minWeightIndex = i;
        end
    end
end
 winnerIndex = minWeightIndex;
end
```

علاوه بر آن در هر حلقه نورون برنده و همسایگان با تعریف و در نظر گرفتن تابع همسایگی f مطابق با زیر به شهر نشان داده شده شبیه خواهند شد.

$$f(\sigma,d) = \mathrm{e}^{\left(-d^2/\sigma^2\right)},$$

. فاصله d که فاصله ی بین دو نورون j و J تعریف می شودبه صورت زیر اندازه گیری می شود:

$$d = \min\{\|j - J\|, m - \|j - J\|\}$$

و m در آن تعداد نورون های خروجی است.

کد مربوط به تابع همسایگی در فایل neighborhoodFunction.m قرار دارد و مطابق با توضیحات بالا به صورت زیر است :

```
function h=neighborhoodFunction(winer,point,var,m)
val=sum(abs(winer-point));
dis=min(val,m-val);
h=exp(-1*power(dis,2)/power(var,2));
end
```

برای به روز رسانی وزن ها از قانون یادگیری زیر استفاده می نماییم.

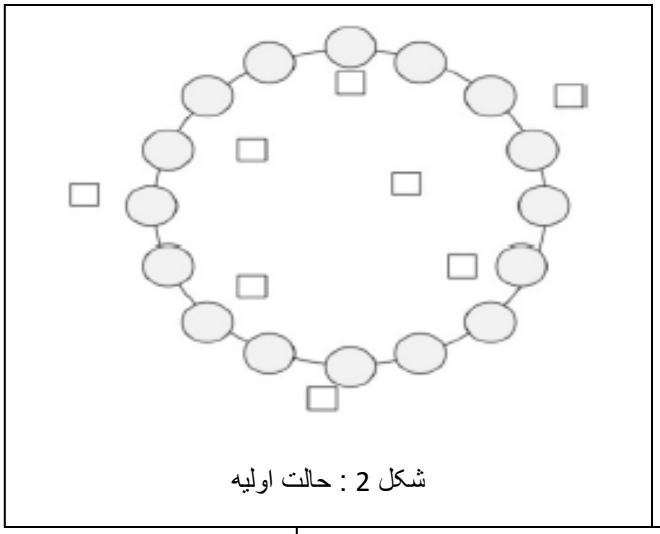
$$y_j^{\text{new}} = y_j^{\text{old}} + \alpha f(\sigma, d)\left(x_i - y_j^{\text{old}}\right),$$

که در آن alpha و sigma به ترتیب نرخ یادگیری و واریانس تابع همسایگی هستند.

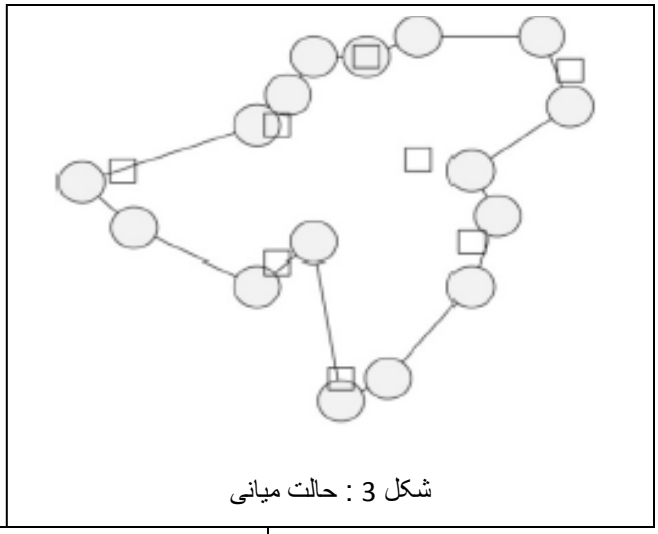
کد مربوط به این قسمت و به روز رسانی وزن ها نیز در فایل TSP.mقرار دارد و به صورت زیر می باشد :

```
for neuronIndex=1:size(neuronWeights,1)
    h=neighborhoodFunction(neorunCordinations(winnerIndex,:),neorunCordinations(neuronIndex,:),var,neuronNum);
    %update weights
    tmp(neuronIndex,:)=neuronWeights(neuronIndex,:)+LR*h*(input-neuronWeights(neuronIndex,:));
end
```

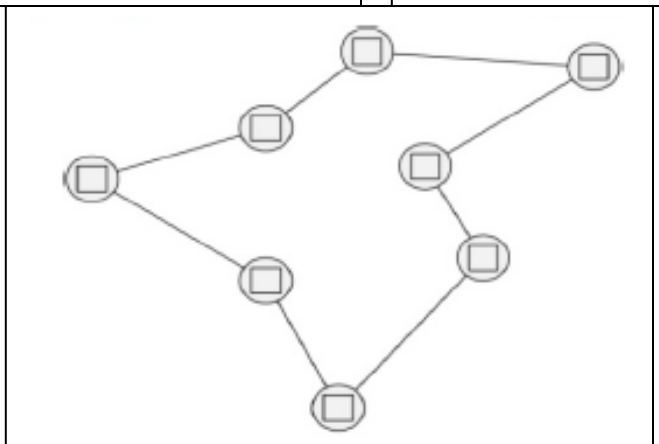
پس از توضیحات درباره نحوه کار الگوریتم و حل مسئله فروشنده دوره گرد، می توان تکامل این الگوریتم را به صورت زیر در نظر گرفت:



شکل 2 : حالت اولیه



شکل 3 : حالت میانی



شکل4 : حالت پایانی(جواب مسئله)

تطبیق پارامترها :

همانطور که در بخش گذشته مشاهده نمودیم الگوریتم SOM دارای دو پارامتر می باشد که عبارتند از alpha که نرخ یادگیری محسوب می شود و دیگری sigma که واریانس تابع همسایگی است. روش پیشنهاد شده از سوی kohonen برای تکامل این پارامتر ها به مرور زمان به صورت تابع نمایی به فرم زیر می باشد.

$$\sigma_k = \sigma_0 \times \exp\left(-\frac{k}{\tau_2}\right),$$

$$\alpha_k = \alpha_0 \times \exp\left(-\frac{k}{\tau_1}\right),$$

که در آنها k شماره تکرار است و t1,t2ثابت های زمانی هستند. لازم به ذکر است که نحوه انتخاب و بهبود پارامتر ها در همگرایی الگوریتم بسیار مهم و تاثیر گذار می باشد.

بر اثر انجام آزمایشاتی که در آنها الگوریتم SOM را برای حل مسئله فروشنده دوره گرد استفاده شده است در مقاله یک روش جدید برای پیدا کردن مقادیر جدید پارامتر هابه مرور زمان پیدا شد که منجر به همگرایی سریع تر الگوریتم می شود. این قوانین به صورت زیر خواهد بود :

$$\alpha_k = \frac{1}{\sqrt[4]{k}},$$

$$\sigma_k = \sigma_{k-1} \times (1 - 0.01 \times k), \quad \sigma_0 = 10.$$

که در این پیاده سازی برای حل مسئله فروشنده دوره گرد از این قوانین استفاده کرده ایم و کد مربوط به این قسمت ها در فایل TSP.m قرار دارد و به صورت زیر می باشد.

```
LR=1/power(iteration,1/4);
var = var*(1-0.01*iteration);
```

الگوریتم MGSOM ارائه شده در مقاله

در مقاله ای که اساس این پروژه قرار گرفت الگوریتمی جهت حل مسئله فروشنده دوره گرد بر پایه شبکه عصبی SOM ارائه شد که در زیر به شرح آن می پردازیم

انواع روش های مقدار دهی اولیه:

روش هایی که برای مقدار دهی های اولیه الگوریتم لازم می باشد در نحوه همگرایی الگوریتم بسیار موثر است. در زیر چگونگی مقدار دهی اولیه در مواردی که در این پیاده سازی لازم است را بررسی می نماییم.

انتخاب تعداد نورون ها و طول همسایگی:
کمترین تعداد نورون هایی که می توان در لایه خروجی قرار داد عبارت خواهد بود از تعداد شهر های موجود در مسئله، اما این تعداد باعث می شود که نتایج بسیار حساس به نحوه قرار گیری اولیه نورون ها باشد. برای انتخاب تعداد نورون های لایه خروجی پیشنهاد شده است که این تعداد را برابر با دو برابر تعداد شهر ها در نظر بگیریم. که در پیاده سازی انجام شده در پروژه نیز تعداد نورون های لایه خروجی دو برابر تعداد شهر ها می باشد.

تابع همسایگی نیز که مشخص کننده تاثیر ورودی نشان داده شده به شبکه بر روی سایر نورون های همسا یه است نیز یک فاکتور مهم دیگر محسوب می شود. این تاثیر برای نورون برنده و اطرافیان نزدیک آن زیاد است و همین طور با افزایش فاصله این تاثیر گذاری کاهش می یابد. با در نظر گرفتن چنین موضوعی پیشنهاد می کنیم که طول تاثیر گذار که عبارت است از تعداد نورون هایی که باید بعد از برنده شدن یک نورون به روز رسانی شوند به 40 درصد کل نورون ها محدود شود. و این تاثیر گذاری با افزایش زمان کاهش می یابد.

انتخاب شاخص برای نورون های خروجی و همچنین نحوه چینش اولیه نورون ها :
برای اینکه جلوی انتخاب هر نورون به عنوان نورون برنده را برای بیش از یک شهر در هر epoch بگیریم برای هر نورون خروجی شاخصی را معرفی می کنیم که شاخص بازدارنده نام دارد. این شاخص به این صورت عمل می کند که هر بار که در هر iteration یک نورون به عنوان برنده مشخص شد این شاخص برای آن به مقداری تغییر می کند که معرف این موضوع است که این نورون قبلا به عنوان برنده یک بار انتخاب شده است و به این ترتیب جلوی انتخاب دوباره آن به عنوان برنده را در iteration های بعدی می گیرد. به این ترتیب شانس انتخاب برای دیگر نورون ها افزایش می یابد.

در این روش پیشنهاد شده برای حل مسئله فروشنده دوره گرد با استفاده از شبکه عصبی SOM در هر iteration هر شهر تنها یک بار انتخاب می شود و ترتیب انتخاب شهر ها و نشان دادن آنها به شبکه در هر iteration به صورت رندم خواهد بود.

در پیاده سازی انجام شده آرایه selectedNourouns که در فایل TSP.m تعریف شده است با این هدف آمده است عناصر آن نشان دهنده آن هستند که آیا آن نورون قبلا به عنوان نورون برنده انتخاب شده است یا نه. کد این قسمت به صورت زیر است :

```
winnerIndex = findWinner(input, neuronWeights,selctedNeurons);
selctedNeurons(winnerIndex,1)=1;
```

مقدار دهی اولیه وزن های سیناپسی نورون ها:
یکی از مسائلی که با آن روبه رو هستیم در حل مسئله فروشنده دوره گرد با استفاده از شبکه عصبی TSP نحوه ی چینش نورون ها در ابتدا و قبل از شروع الگوریتم است . اینکه نورون ها را چگونه و بر روی چه منحنی در کنار همدیگر قرار دهیم تا کمینه زمان محاسبه و طول گردش شهر ها را داشته باشیم. منحنی های بسیاری وجود دارند که می توان نورون ها را در ابتدا بر روی آنها قرار داد اما سوالی که مطرح می شود این است که کدام یک از این منحنی ها و حالات اولیه چینش نورون ها از بقیه بهتر است؟ در واقع می توان گفت که هیچ جواب قطعی برای پاسخ به این سوال وجود ندارد اما در مقاله پیاده سازی شده تلاش های بسیاری برای پیدا کردن حالت بهینه صورت گرفته است و در نهایت بهترین آن پیاده سازی شده است.

اینکه بخواهیم تمام حالات ممکن را بررسی کرده و از میان آنها بهترین را انتخاب کنیم غیر ممکن است.در این مقاله 4 مورد به صورت زیر بررسی شده است:

روش اول:

مقدار دهی اولیه وزن های نورون ها بر روی فضای داده های ورودی (شهر های TSP)مهمترین مشکل این روش این است که زمان پردازش برای رسیدن به یک چنش بر اساس شباهت بین نورون ها و شهر ها احتیاج است.

روش دوم:

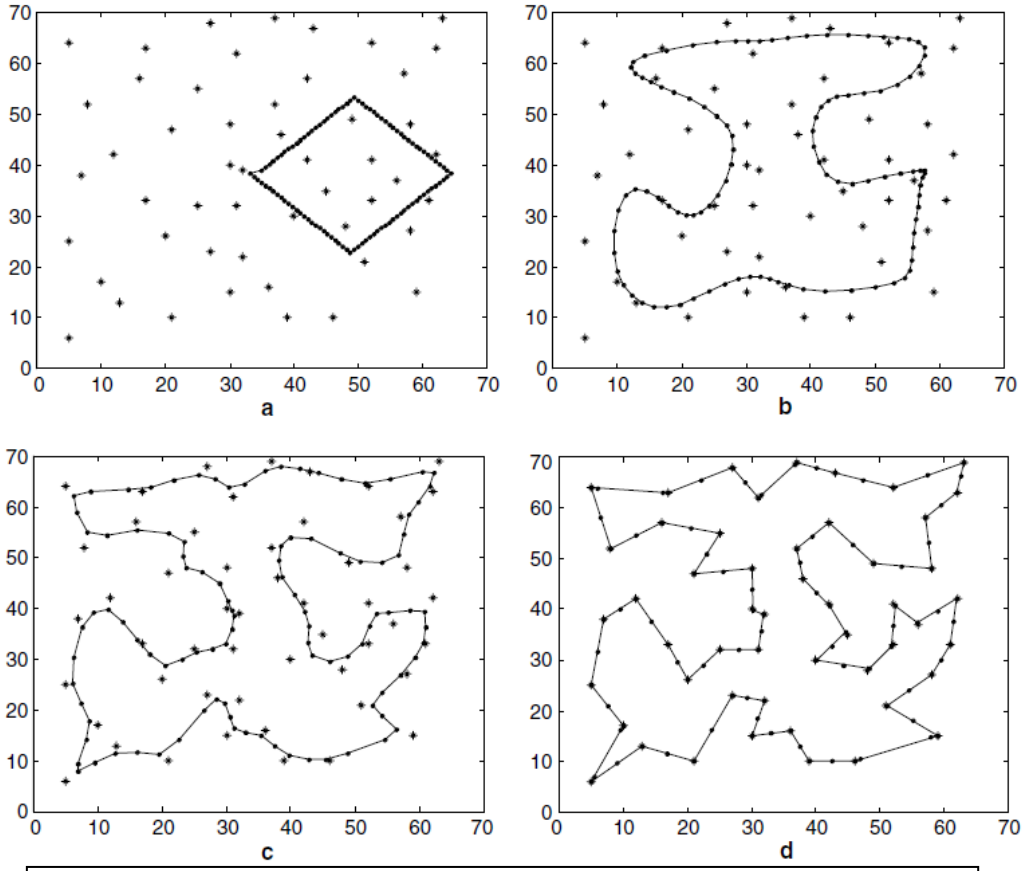
نورون ها را به صورت تصادفی بر روی دایره ای به مرکز شهر ها قرارا دهیم. که در نهایت این روش تاثیر اندکی بر روی طول مسیر مشخص شده دارد.

روش سوم:

روش سوم به این صورت است که نورون ها را بر روی یک تور(مسیری که از شهر ها می گذرد) قرار دهیم. در این حالت از روش نزدیک ترین همسایه استفاده می کنیم. تجربیان نشان داده است که این روش تاثیر اندکی بر روی زمان اجرای الگوریتم و همچنین طول مسیر نهایی پیدا شده دارد.

روش چهارم:

این روش به گونه ای خواهد بود که نورون های در ابتدا بر روی یک لوزی که در سمت راست مرکز شهر ها قرار دارند قرار می گیرند. بعد از شروع الگوریتم و به مرور زمان لوزی که متشکل از نورون ها ایت کش آمده و بر روی شهر ها قرار می گیرند. در این روش شاهد سرعت بالاتر رسیدن به جواب خواهیم بود و همچنین به نتایج بهتری خواهیم رسید. در شکل زیر حالت اولیه چینش نورون ها، چینش آنها بعد از چند بار تکرار و همچنین حالت نهایی را مشاهده می کنیم.



شکل5: موارد a تا d نحوه چینش نورون ها از ابتدا شوع الگوریتم تا انتها را مشخص می نماید

الگوریتم MGSOM

در مقاله مورد بررسی و پیاده سازی الگوریتمی بر پایه توضیحات قبلی ارائه شده است با نام MGSOM که در زیر به تشریح آن می پردازیم :

1_مقدار دهی اولیه :

فرض نمایید nتعداد شهر ها باشد ورودی این الگوریتم مختصات دکارتی n تا شهر خواهد بود. همچنین فرض کنید m(t) تعداد نورون ها بر روی لوزی باشدکه در سمت راست n شهر واقع شده است. در ابتدا و برای شروع به کار m(0)را برابر با n در نظر می گیریم. مقدار اولیه sigma را 10 و tرا برابر با 1 در نظر می گیریم. Tmin را نیز برابر با تعداد تکرار های مجاز در نظر می گیریم.

2_ترتیب های تصادفی :

محاسبات را با ترتیب های مختلفی از شهر ها که از 1,…,n برچسب خورده اندآغاز نمایید. همچنین در ابتدای هر حلقه لازم است که شاخص بازدارنده نورون ها را که در بالا معرفی کردیم را false قرار دهید.

3_تطبیق پارامتر ها :

دو پارارمتر alpha که نرخ یادگیری است و همچنین sigma که واریانس تابع همسایگی محسوب می شودبا استفاده از دو فرمول زیر که قبلا نیز آنها را معرفی نموده بودیم محاسبه می کنیم.

$$\alpha_k = \frac{1}{\sqrt[4]{k}},$$

$$\sigma_k = \sigma_{k-1} \times (1 - 0.01 \times k), \quad \sigma_0 = 10.$$

4_ رقابت :

به ازای هر ورودی که به شبکه داده می شود، طی یک فرایند رقابتی نزدیک ترین نورون به شهر داده شده به عنوان ورودی انتخاب می شود.(معیار نزدیکی در این حالت فاصله اقلیدسی می باشد) این رقابت تنها بین نورون هایی صورت خواهد گرفت که در iteration حال حاضر به عنوان نورون برنده انتخاب نشده باشند. بنابراین نورون برنده J به صورت زیر انتخاب خواهد شد.

$$J = \text{Argmin}_j \|x_i - y_j\| \quad \text{for } \{j|\text{inhibit}[j] = \text{false}\}$$

و بعد از آن شاخص بازدارنده را برای این نورون به true تغییر خواهیم داد.

5_ به روز رسانی وزن نورون ها :

بر اساس معادلات گفته شده در قبل وزن های سیناپسی نورون برنده و همسایگان آن به روز رسانی می شوند و بقیه وزن ها در زمان t ثابت باقی می مانند. طول همسایگی نورون ها محدود به m(t) %40 که برابر با تعداد نورن ها در زمان t است می باشد.
همچنین یه شمارنده Cj(t) در نظر می گیریم تا بوسیله آن تاریخچه یادگیری را تا حدی نگه داری نماییم.

این شمارنده برای نورون برنده j یکی افزایش می یابد و در غیر این صورت بدون تغییر باقی می ماند.

$$C_j(t+1) = \begin{cases} C_j(t)+1, & \text{if } i = J \\ C_j(t), & \text{otherwise} \end{cases}$$

6_ اضافه کردن یک نورون جدید :

در هر تعداد تکرار مشخص یک بار یک نورون جدید را به نورون های قبلی اضافه می نماییم تا وقتی که تعداد نورون ها به اندازه 2*n برسد. هر بار که قرار است یک نورون مانند r را به نورون های جدید اضافه می نماییم. در این حالت نورونی مانند p را که شمارنده Cp(t) آن از همگی بیشتر است را انتخاب می نماییم .

$$C_p(t) \geqslant C_j(t), \quad \text{for all } j$$

اگر چند نورون وجود داشته باشد مقدار شمارنده آنها ماکزیمم باشد در این صورت یکی را به صورت رندم انتخاب می نماییم. همسایه نورون p را به صورت f=p-1 در نظر می گیریم، و نورون جدید را که r نامیدیم بین p و f اضافه می کنیم و وزن سیناپسی نورون rرا به صورت زیر مقدار دهی می کنیم :

$$y_r = 0.5(y_p + y_f)$$

همچنین مقدار شمارنده نورون p و r را دوباره به صورت زیر مقدار دهی می کنیم.

$$C_p(t+1) = 0.5C_p(t), \qquad C_r(t+1) = 0.5C_p(t)$$

بعد از این اضافه کردن تعداد نورن ها یکی افزایش خواهد یافت m(t+1) = m(t)+1

7_ اگر همه شهر ها به عنوان ورودی نشان داده نشده اند (یک iteration به پایان نرسیده است) به مرحله 4ام بروید

8_ تا زمانی که تعداد تکرار ها به می نیمم تعداد تکرار ها نرسیده است به گام دوم بروید.

نتایج ارائه شده در مقاله:

برای ارزیابی الگوریتم ارائه شده این الگوریتم بر روی نقشه های معروف موجود برای مسئله فروشنده دوره گرد اجرا شده است و در مقایسه با جواب بهینه و همچنین جواب های پیدا شده توسط دیگر الگوریتم ها نتایج به صورت زیر خواهد بود.

Instances	No. of cities	Optimum length
Bier127	127	118,282
Eil51	51	426
Eil76	76	538
kroA200	200	29,368
Lin105	105	14,379
pcb442	442	50,778
Pr107	107	44,303
Pr6	6	96,772
Pr152	152	73,682
Rat195	195	2323
ed100	100	7910
st70	70	675

جدول شماره1 : نمونه نقشه های استفاده شده برای مسئله فروشنده دوره گرد جواب کمینه

Instances	PKN	GN	AVL	MGSOM
Bier127	122211.7	155163.2	122673.9	119,580
Eil51	443.9	470.7	443.5	431.9569
Eil76	571.2	614.3	571.3	556.2061
kroA200	30927.8	39370.2	30994.9	29,947
Lin105	15374.2	15469.5	15311.7	14,383
pcb442	–	–	59649.8	55,133
Pr107	44504.3	80481.3	45096.4	44,379
Pr136	103878.0	5887.7	103442.3	98,856
Pr152	74804.2	105230.1	74641.0	74,228
Rat195	–	–	2681.2	2462
rd100	87.9	8731.2	8265.8	8002.7
st70	692.8	755.7	693.3	682.9886

جدول شماره2 : نتایج دیگر الگوریتم های ارائه شده برای حل مسئله فروشنده دوره گرد و الگوریتم MGSOM

Instances	KL	KG	SETSP	MGSOM
Bier127	121548.7	121923.7	120470	119,580
Eil51	438.2	438.2	435.46	431.9569
Eil76	564.8	567.5	560.78	556.2061
kroA200	30200.8	30444.9	30,284	29,947
Lin105	14664.4	14564.6	14,566	14,383
pcb442	56399.9	56082.9	55,937	55,133
Pr107	44628.3	44491.1	44,484	44,379
Pr136	101156.8	101752.4	101,030	98,856
Pr152	74395.5	74629.0	74,543	74,228
Rat195	2607.3	2599.8	2583	2462
rd100	8075.7	8117.4	8115.7	8002.7
st70	685.2	690.7	685.8	682.9886

جدول شمار3: نتایج دیگر الگوریتم های ارائه شده برای حل مسئله فروشنده دوره گرد و الگوریتم MGSOM

Instances	KL	KG	SETSP	MGSOM
Bier127	2.76	3.08	1.85	1.0936
Eil51	2.86	2.86	2.22	1.3983
Eil76	4.98	5.48	4.23	3.3840
kroA200	2.84	3.67	3.12	1.9715
Lin105	1.98	1.29	1.30	0.0278
pcb442	11.07	10.45	10.16	8.5770
Pr107	0.73	0.42	0.41	0.1719
Pr136	4.53	5.15	4.40	2.1530
Pr152	0.97	1.29	1.17	0.7412
Rat195	12.24	11.92	11.19	5.9847
rd100	2.09	2.62	2.60	1.1718
st70	1.51	2.33	1.60	1.1835
Total average	4.05	4.21	3.69	2.3215

جدول شمار4: میانگین انحراف معیار از طول بهینه راه حل ارائه شده توسط روش های مختلف

جدول شماره 4 میزان انحراف معیار از جواب های بهینه را برای هر یک از نمونه نقشه های TSP با تعداد مختلف شهر نشان می دهد. همانطور که مشخص است نتایج بدست آمده برای الگوریتم MGSOM از بقیه الگوریتم ها بهتر می باشد.

پیاده سازی و نتایج بدست آمده در پروژه:

پیاده سازی انجام شده به عنوان پروژه این درس با الگوریتم ارائه شده در بالا کمی متفاوت است. به عنوان مثال در ابتدا مختصات همه شهر ها scale می شوند و نورون ها با تعداد ثابت دو برابر تعداد شهر ها از ابتدا بر روی دایره ای به شعاع واحد و مرکز مختصات شهر ها قرار می گیرند.

بقیه موارد و قوانین به روز رسانی وزن ها و پارامتر ها دقیقا مشابه روند ارائه شده در الگوریتم و توضیحات ابتدایی گزارش پروژه است.

نتایج بدست آمده از پیاده سازی پروژه توسط اینجانب به صورت زیر می باشد که در آن کمینه طول پیدا شده توسط الگوریتم در جدول ذکر شده است. با توجه به اینکه زمان لازم برای اجرا تا رسیدن به پاسخ در حالتی که تعداد نورون های لایه خروجی زیاد باشد طولانی می شود و همچنین عدم امکان استفاده از کامپیوتر رو میزی با امکانات بهتر از لپ تاپ برای بهبود سرعت اجرا ناچار به اجرای کد برای نمونه دیتا ست های آماده با تعداد نورون های کمتر از 30 شده که در زیر نتایج را مشاهده می نماییم (تعداد نورون ها برابر است با عدد جلوی نام نقشه).

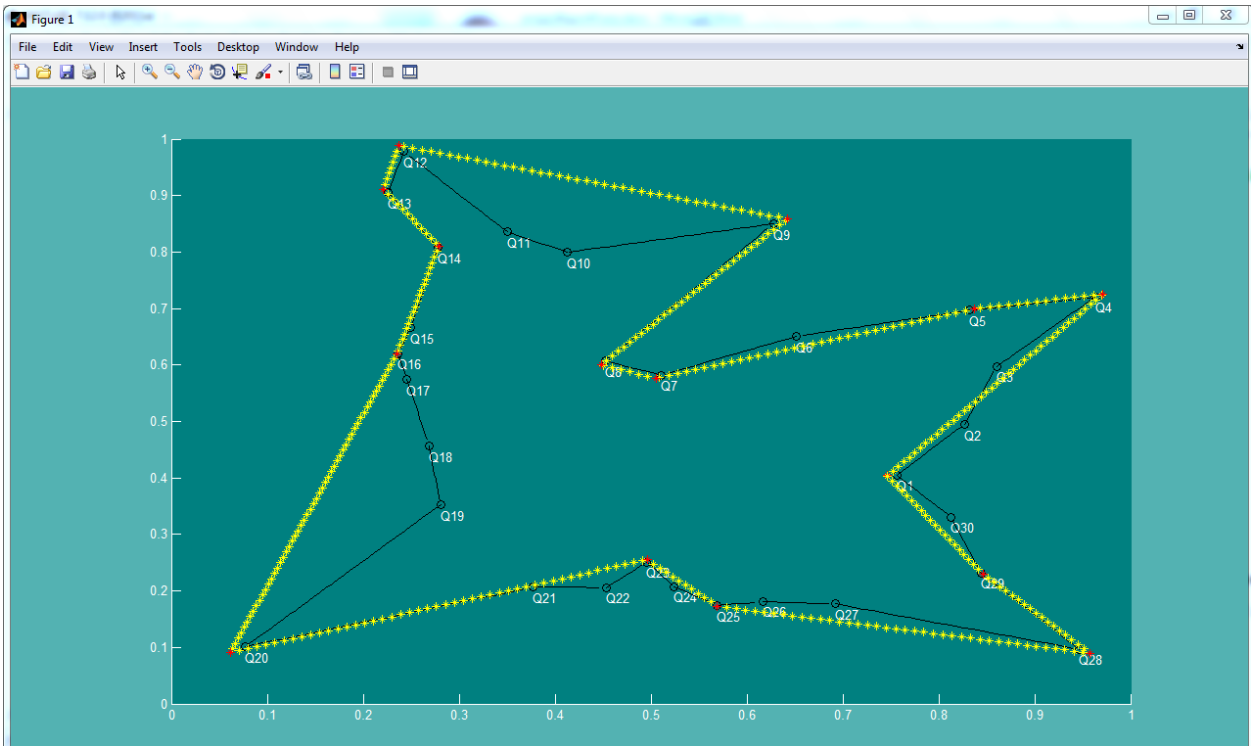
به همراه کد ارسال شده این چهار نقشه نیز ارسال گردید. که می توان با تغییربه صورت دستی در فایل TSP.m هر یک از این نقشه ها را به عنوان وردی به شبکه داد.

MAP	Optimal	Implementation
Burma14	3323	3412
Ulysses16	6859	6859
Bays29	2020	2131
Fry26	937	1102

کد ارسال شده در حال حاضر در فایل TSP.m که فایل main برنامه محسوب می شود تنظیم شده است که تعداد15 شهر به صورت رندم بر روی نقشه قرار دهد و راه حل را که عبارت است از کوتاه ترین مسیر موجود به گونه ای که از تمام شهر ها عبور می کند را در نهایت پیدا کرده و با رنگ زرد نمایش می دهد و طول آن را نیز چاپ می نماید. در صورتی که تمایل به تغییر تعداد شهر ها دارید، می توانید این تعداد را در فایل TSP.m که در متغیر cityNum قرار دارد به صورت دستی تغییر دهید و یا از نقشه های نام برده در بالا استفاده نمایید. در ابتدای فایل TSP.m کد مربوط به استفاده از هر یک نقشه ها آمده است اما به صورت کامنت شده قرار دارد کافیست علامت کامنت را حذف کنید و خط 16 که متغیر cityCordinations را مقدار دهی می کنیم مقدار درست را قرار دهید.

به همراه گزارش ارسال شده یک ویدئو از اجرای برنامه نیز ارسال شده است.

همچنین یک نمونه خروجی برنامه را در شکل زیر می بینیم که مسیر زرد رنگ نشان دهنده کوتاه ترین مسیر پیدا شده توسط پروژه است :



شکل 6: خروجی برنامه برای نقشه ای با تعداد 15 شهر

منابع :

maps strategy for Bai, Yanping, Wendong Zhang, and Zhen Jin. "An new selforganizing solving the traveling salesman problem." Chaos, Solitons & Fractals 28.4 (2006): 1082-1089.