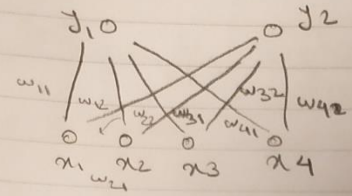
سوال یک

①



در ابتدا وزن های اولیه را initiate می کنیم

$$w = \begin{bmatrix} 0.2 & 0.9 \\ 0.4 & 0.3 \\ 0.6 & 0.5 \\ 0.8 & 0.7 \end{bmatrix} \qquad x = [1 \; 0 \; 0 \; 0]$$

با این نورون شروع میکنیم

در ابتدا فاصله نورون x را تا هر کدام از y_1 و y_2 حساب میکنیم

← فاصله تا y_1 :

$$\sqrt{(1-0.2)^2+(-0.4)^2+(-0.6)^2+(-0.8)^2} = \sqrt{0.64+0.16+0.36+0.64}$$

$$= \sqrt{1.8} = \boxed{\sqrt{1.8}}$$

← فاصله تا y_2 :

$$\sqrt{(1-0.9)^2+(-0.3)^2+(-0.5)^2+(-0.7)^2} = \sqrt{0.01+0.09+0.25+0.49}$$

$$= \boxed{\sqrt{0.84}}$$

نورون $[1\;0\;0\;0]$: y_2 نزدیک تر هست به همین علت وزن های y_2 را باید آپدیت کنیم

$$w = w + \Delta w = \boxed{w + \alpha(x - w)}$$

$$\Rightarrow \begin{bmatrix} 0.9 \\ 0.3 \\ 0.5 \\ 0.7 \end{bmatrix} + 0.5\left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0.9 \\ 0.3 \\ 0.5 \\ 0.7 \end{bmatrix}\right) = \begin{bmatrix} 0.9 \\ 0.3 \\ 0.5 \\ 0.7 \end{bmatrix} + 0.5\begin{bmatrix} 0.1 \\ -0.3 \\ -0.5 \\ -0.7 \end{bmatrix}$$

$$= \begin{bmatrix} 0.95 \\ 0.15 \\ 0.25 \\ 0.35 \end{bmatrix}$$

الان ماتریس وزن ها به صورت زیر است:

$$w = \begin{bmatrix} 0.2 & 0.95 \\ 0.4 & 0.15 \\ 0.6 & 0.25 \\ 0.8 & 0.35 \end{bmatrix}$$

در ادامه برای نورون $[0\ 1\ 1\ 0]$ مراحل را انجام می‌دهیم

$$w = \begin{bmatrix} 0.2 & 0.95 \\ 0.4 & 0.15 \\ 0.6 & 0.25 \\ 0.8 & 0.35 \end{bmatrix} \qquad x = [0\ 1\ 1\ 0]$$

فاصله را با هر کدام y_1 و y_2 می‌سنجیم

فاصله تا y_1:

$$\sqrt{(0-0.2)^2 + (1-0.4)^2 + (1-0.6)^2 + (0-0.8)^2} =$$

$$\sqrt{0.04 + 0.36 + 0.16 + 0.64} = \sqrt{1.2}$$

فاصله تا y_2:

$$\sqrt{(0-0.95)^2 + (1-0.15)^2 + (1-0.25)^2 + (0-0.35)^2} =$$

$$\sqrt{(0.95)^2 + (0.85)^2 + (0.75)^2 + (0.65)^2} = \sqrt{0.9025 + 0.7225 + 0.5625 + 0.4}$$

$$= \sqrt{2.61}$$

نورون $x=[0\ 1\ 1\ 0]$ تا y_1 فاصله نزدیک‌تری داریم حالا وزن‌های آن آپدیت می‌کنیم

$$\begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \\ 0.8 \end{bmatrix} + 0.5\left(\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \\ 0.8 \end{bmatrix}\right) = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \\ 0.8 \end{bmatrix} + 0.5 \begin{bmatrix} -0.8 \\ 0.6 \\ 0.4 \\ -0.8 \end{bmatrix}$$

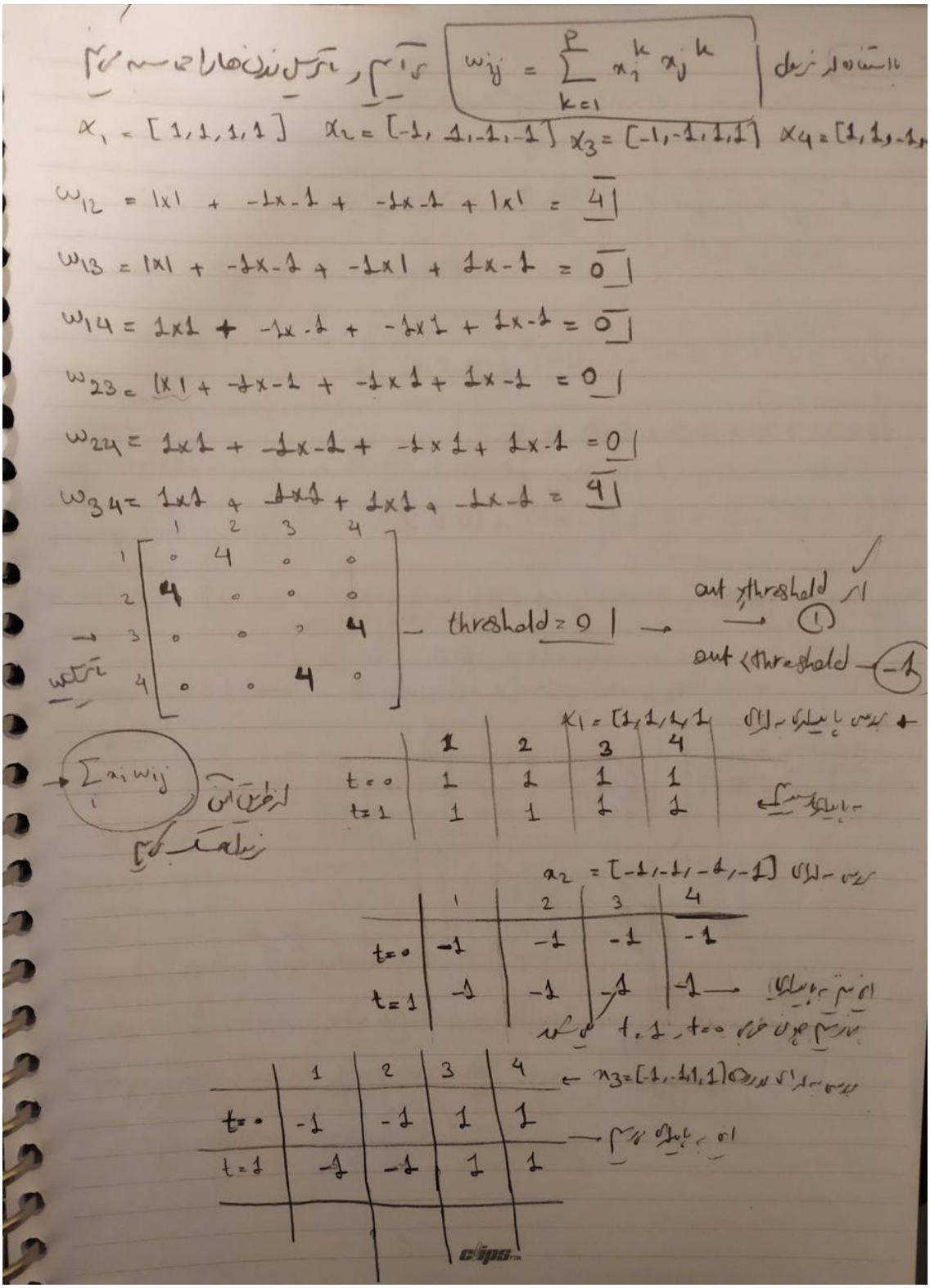
$$= \begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \\ 0.8 \end{bmatrix} + \begin{bmatrix} -0.4 \\ 0.3 \\ 0.2 \\ -0.4 \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.7 \\ 0.8 \\ 0.4 \end{bmatrix}$$

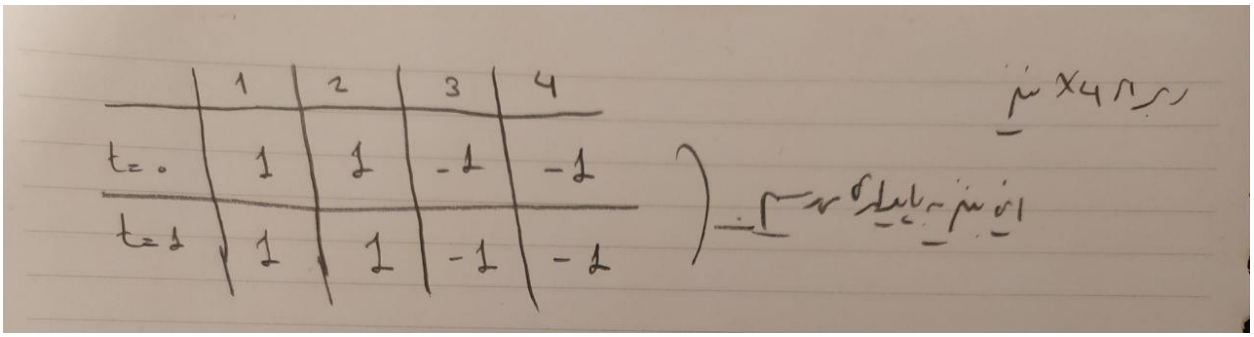
$$w = \begin{bmatrix} -0.2 & 0.95 \\ 0.7 & 0.15 \\ 0.8 & 0.25 \\ 0.4 & 0.35 \end{bmatrix}$$

ماتریس وزن نهایی که باید نشان داده می‌شد:

هر بار این فرآیند را برای هر نورون تکرار کردیم برای هر نورون دیگر هم اینکار را انجام دهیم در این صورت ① epoch را انجام داده‌ایم. و این کار را می‌توانیم تا زمانی ادامه دهیم که ماتریس وزن ما دیگر تغییری نکند.

سوال دوم


$w_{ij} = \sum_{k=1}^{p} x_i^k x_j^k$
$x_1 = [1,1,1,1]$ $x_2 = [-1,-1,-1,-1]$ $x_3 = [-1,-1,1,1]$ $x_4 = [1,1,-1,-1]$
$w_{12} = 1x1 + -1x-1 + -1x-1 + 1x1 = 4$
$w_{13} = 1x1 + -1x-1 + -1x1 + 1x-1 = 0$
$w_{14} = 1x1 + -1x-1 + -1x1 + 1x-1 = 0$
$w_{23} = 1x1 + -1x-1 + -1x1 + 1x-1 = 0$
$w_{24} = 1x1 + -1x-1 + -1x1 + 1x-1 = 0$
$w_{34} = 1x1 + -1x-1 + 1x1 + -1x-1 = 4$
threshold = 0
out > threshold → 1
out < threshold → -1
$\sum_i x_i w_{ij}$
t=0
t=1
$x_2 = [-1,-1,-1,-1]$
$x_3 = [-1,-1,1,1]$
clips



سوال سوم

در این سوال شبکه MLPکه طراحی کردم به این صورت که دوتا hidden layer داریم که تعداد نورون ها در ان برابر با ۸ است و هم چنین برای تابع ضرر از MSE استفاده کردم و بعد با استفاده از gradient descent وزن ها را اپدیت می کنیم activation function که بین لایه استفاده کردیم نیز relu هست در واقع معماری به صورت زیر است:

```
class MLP:
    def __init__(self, input_size, hidden_size, output_size):
      self.W1 = np.random.uniform(0, 1, size=(input_size, hidden_size[0]))
      self.b1 = np.random.uniform(0, 1, size=(hidden_size[0],))
      self.W2 = np.random.uniform(0, 1, size=(hidden_size[0], hidden_size[1]))
      self.b2 = np.random.uniform(0, 1, size=(hidden_size[1],))
      self.W3 = np.random.uniform(0, 1, size=(hidden_size[1], output_size))
      self.b3 = np.random.uniform(0, 1, size=(output_size,))
```

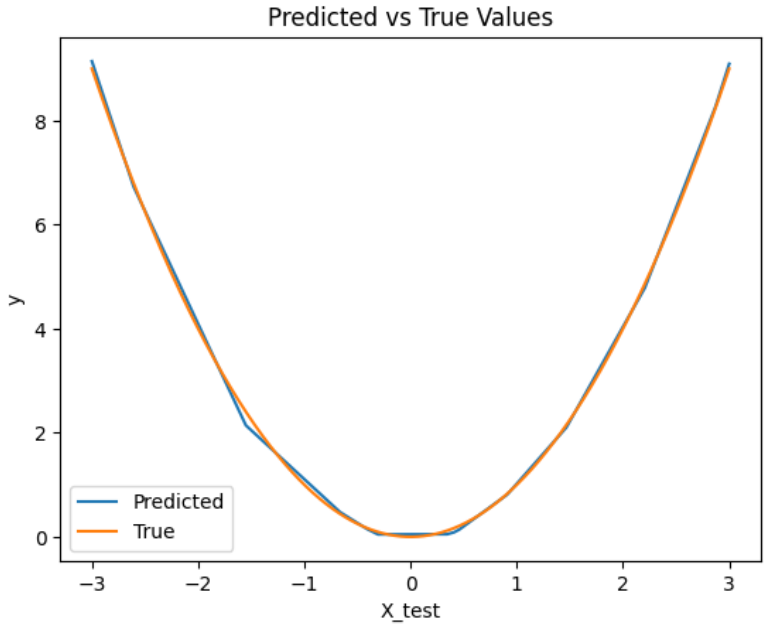
و هم چنین پارامترهای ما نیز به صورت زیر است:

```
input_size = 1
hidden_size = [8,8]
output_size = 1
learning_rate = 0.01
num_epochs = 10000
```

و در ادامه در یک حلقه for ما مراحل forward و محاسبه تابع ضرر و backward را طی می کنیم و نتایج تابع ضرر در چند اپک اخر به صورت زیر است:

```
Epoch: 9700/10000, Loss: 0.0104769164888899677
Epoch: 9800/10000, Loss: 0.0116700195904874 14
Epoch: 9900/10000, Loss: 0.0118759810051110 01
Epoch: 10000/10000, Loss: 0.0107471147251486 08
```

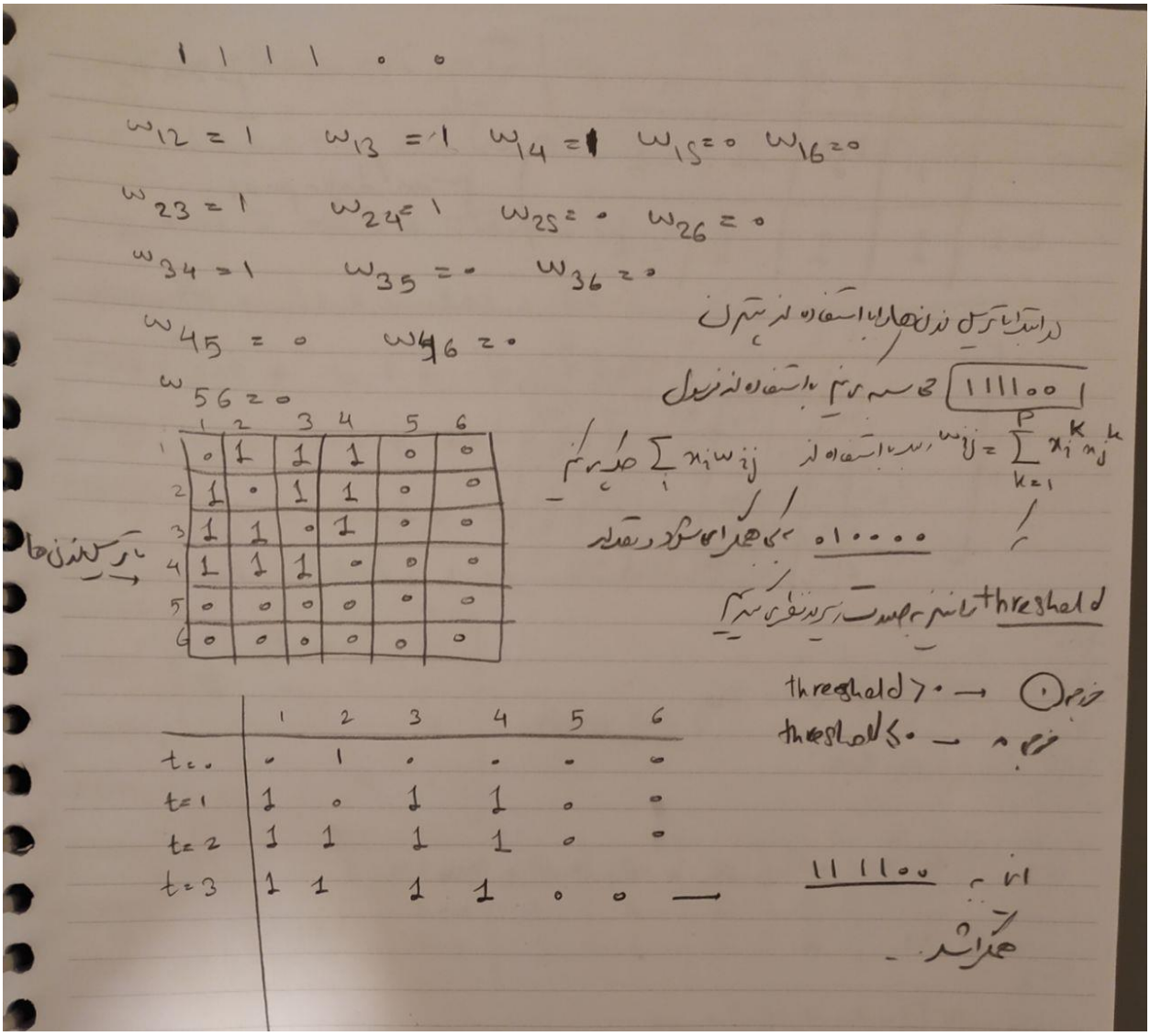
و بعد برای پیش بینی تابع در بازه -۳ و۳ اومدیم دونمودار که یکی مربوط به predict هست و دیگری مربوط به خود تابع هست را رسم کردم که نتیجه به صورت زیر شد:



تمام کد در نوتبوک q3 موجود است.

سوال چهارم

پاسخ در عکس زیر است:



سوال پنجم

یکی از بهترین شبکه ها برای مسائل بهینه سازی SOM هست مسئله TSP را می توانیم به این صورت مدل کنیم:

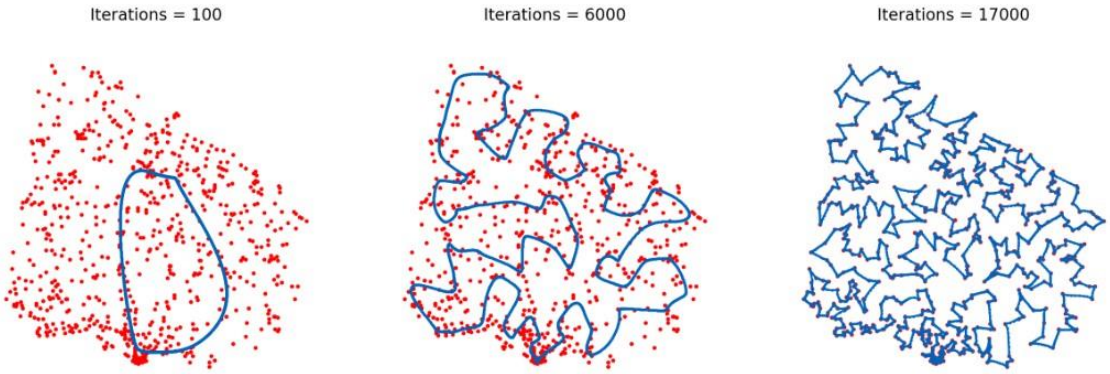
در ابتدا یک شبکه با تعداد نورون های برابر با تعداد شهرهایی که درمسئله داریم در نظر می گیریم برای بردار وزن هر نورون، ۲ عنصر داریم. ورودی های شبکه در اینجا، مختصات شهرها هستند که بهتر است نرمالیزه شوند.وزن های شبکه را در ابتدا به صورت تصادفی initialize می کنیم. در هر بار یک نورون انتخاب میشود. بنابراین برای هر شهر، یک نورون برنده داریم که مکان ان شهر را در مسیر مشخص میکند.

در واقع هر دفعه که یک نورون برنده میشود وزن ها به صورت زیر اپدیت میشود:

$$n_{t+1} = n_t + h(w_e) \cdot \Delta(e, n_t)$$

که h همان تابع همسایگی هست و این مراحل تا جایی ادامه میدهیم که دیگر وزن ها زیاد تغییری نکنند

شکل زیر دید خوبی از همگرا شدن به پاسخ بهینه را نشان میدهد:



برای استفاده از SOM برای حل TSP از سایت زیر استفاده کردم:

https://diego.codes/post/som-tsp

این مسئله با استفاده از Hopfield قابل حل هست. به این صورت که برای هر شهر،یک نورون در شبکه هاپفیلد در نظر میگیریم. همچنین از فاصله بین شهرها نیز به عنوان وزن اولیه استفاده میکنیم. سپس شبکه د را train کرده و اگر یک وزن ۱ بشود، جای آن شهر را در مسیر نشان میدهد. زمانی که یکی از نورون های شبکه در جایگاه یک نورون متفاوت ۱ بشود نشان دهنده رسیدن به مسیر می باشد و هم چنین ان مسیر بهترین مسیر میشود .

به طور کلی، استفاده از شبکه Hopfield در حل مسئله TSP می‌تواند یک روش تقریبی باشد و به صورت محدود می‌تواند به نتایج بهینه نزدیک شود. با این حال، باید توجه داشت که شبکه Hopfield ممکن است در مقایسه با الگوریتم‌های بهینه‌سازی کلاسیک و الگوریتم‌های دقیق و همچنین SOM، دقت کمتری داشته باشد و در مسئله TSPبا تعداد شهرهای زیاد، به دلیل پیچیدگی محاسباتی، نتایج به دست آمده ممکن است ناکارآمد باشد.