به نام خدا

تمرین سری سوم
درس مبانی هوش محاسباتی
دکتر ناصر مزینی

فرزان رحمانی
۹۹۵۲۱۲۷۱

سوال اول

با توجه به مطالب اسلاید های زیر به حل سوال می پردازیم:

Formation of SOM

Essential processes of a SOFM(self-organizing feature map) are:

1. Initialization
2. Competition
3. Cooperation
4. Synaptic Adaptation

1- Initialization of the synaptic weights is done by assigning ***random small values*** in order to impose no prior order on the map

2- Competition

- An input vector ***x*** selected at random and presented to the network

Have same dimension

- Input vector: $X=[x_1, x_2,..., x_m]^T$
- Weights vector: $W_j=[w_{j1}, w_{j2},..., w_{jm}]^T$

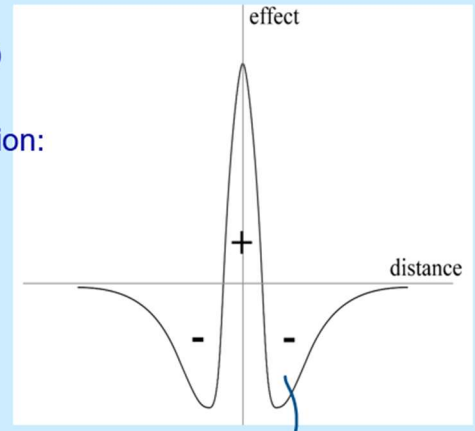
- A neuron is selected as the winner based on a best matching criterion

The Euclidian distance: $i(x)=\arg\min ||X-W_j||$

3- Cooperation

- A topological neighborhood of the winning neuron is selected $h_{j,i(x)}$

The best form is a Mexican hat function:



lateral inhibition
مهار جانبی

A simpler typical choice can be a Gaussian function:

$$h_{j,i(x)} = e^{\frac{-d_{j,i}^2}{2\sigma^2}}$$

4- Adaptation

Synaptic weights are modified according to :

$$\Delta w_j = \eta y_j x - g(y_j) w_j$$

Forgetting term

Hebb Rule

By choosing :

$$g(y_j) = \eta h_{j,i(x)}$$

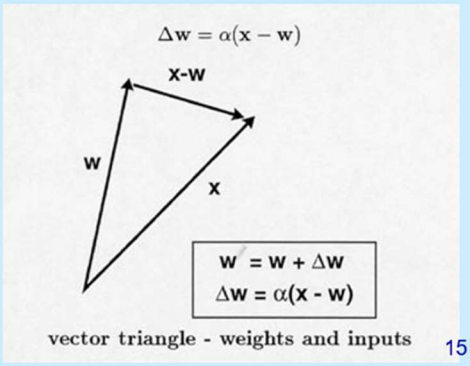
and:

$$y_j = h_{j,i(x)}$$

We obtain :

$$\Delta w_j = \eta h_{j,i(x)} (x - w_j)$$

w ro nazdik be x kon



vector triangle - weights and inputs

15

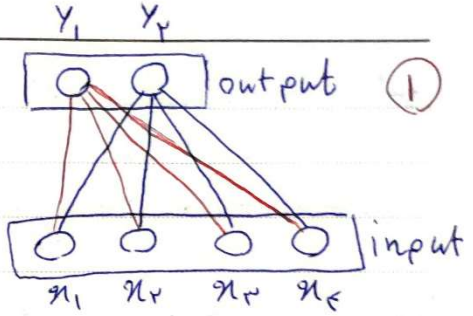
جواب سوال در ادامه آمده است:

Subject
Date

$x_1 = [0, 1, 1, 0]$
$x_2 = [1, 1, 0, 0]$
$x_3 = [1, 0, 0, 0]$
$x_4 = [0, 0, 0, 1]$

همسایگی = 0

$\alpha = \eta = 0,6$



(1)

$$W = \begin{bmatrix} 0,2 & 0,9 \\ 0,4 & 0,7 \\ 0,6 & 0,5 \\ 0,8 & 0,3 \end{bmatrix} \Rightarrow \begin{matrix} W_1 = [0,2 \ 0,4 \ 0,6 \ 0,8] \\ W_2 = [0,9 \ 0,7 \ 0,5 \ 0,3] \end{matrix}$$

W_1 W_2

$$\|z\| = \sqrt{\sum_{k=1}^{n} z_k^2}$$ euclidian distance

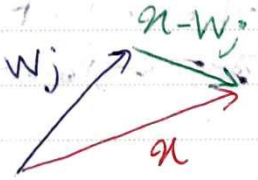
Formation of SOM =
1. Initialization
2. Competition → $i(x) = \arg\min \| X - W_j \|$
3. Cooperation → در نظر می گیریم چون فرض کردیم 0 است
4. Synaptic Adaptation

$$W_j := W_j + \Delta W_j$$

$$\Delta W_j = \eta\, y_j\, x - g(y_j) W_j = \eta\, h_{j,i(x)}(x - W_j)$$

$h_{j,i(x)}$ $\eta\, h_{j,i(x)}$ برای نورون برنده یک برای بقیه صفر است

$$\Delta W_j = \eta(x - W_j) = 0,6(x - W_j)$$



$x_1 = [0, 1, 1, 0]$ با نمونه اول شروع می کنیم.

$x_1 - W_1 = [-0,2 \ 0,6 \ 0,4 \ -0,8] \rightarrow \|x_1 - W_1\| = \sqrt{0,04 + 0,36 + 0,16 + 0,64}$

$x_1 - W_2 = [-0,9 \ 0,3 \ 0,5 \ -0,3] \rightarrow \|x_1 - W_2\| = \sqrt{0,81 + 0,09 + 0,25 + 0,09}$

$\|x_1 - W_1\| \simeq 1,095$
$\|x_1 - W_2\| \simeq 1,114$
→ W_1 is winner → cooperation چون همسایگی صفر است نداریم پس فقط Synaptic Adaptation داریم.

Subject
Date

$\Delta W_1 = 0,5(x_1 - W_1) = 0,5[-0,2 \;\; 0,4 \;\; 0,4 \;\; -0,1] =$
$[-0,1 \;\; 0,2 \;\; 0,2 \;\; -0,4]$

$W_1 := W_1 + \Delta W_1 = [0,2 \;\; 0,4 \;\; 0,6 \;\; 0,1] + [-0,1 \;\; 0,2 \;\; 0,2 \;\; -0,4]$

$W_1 = [0,1 \;\; 0,7 \;\; 0,8 \;\; 0,4]$
جدید

$W_1 = [0,1 \;\; 0,7 \;\; 0,8 \;\; 0,4]$ نمونه دوم $x_2 = [1 \;\; 1 \;\; 0 \;\; 0]$
$W_2 = [0,9 \;\; 0,7 \;\; 0,5 \;\; 0,3]$

$x_2 - W_1 = [0,9 \;\; 0,3 \;\; -0,8 \;\; -0,4]$ $\|x_2 - W_1\| = 1,3064$
$x_2 - W_2 = [0,1 \;\; 0,3 \;\; -0,5 \;\; -0,3]$ $\|x_2 - W_2\| = 0,693$

W_2 is winner

$\Delta W_2 = 0,5(x_2 - W_2) = [0,05 \;\; 0,15 \;\; -0,25 \;\; -0,15]$

$W_2 := W_2 + \Delta W_2 = [0,95 \;\; 0,85 \;\; 0,25 \;\; 0,15]$
جدید

$W_1 = [0,1 \;\; 0,7 \;\; 0,8 \;\; 0,4]$ همین روند را برای نمونه سوم
$W_2 = [0,95 \;\; 0,85 \;\; 0,25 \;\; 0,15]$ یعنی $x_3 = [1,0,0,0]$ ادامه می‌دهیم.
W_2 برنده می‌شود.
W_2 جدید $= [0,975 \;\; 0,425 \;\; 0,125 \;\; 0,075]$

$W_1 = [0,1 \;\; 0,7 \;\; 0,8 \;\; 0,4]$ اگر همین کار را برای $x_4 = [1 \;\; 0 \;\; 0 \;\; 0]$
$W_2 = [0,975 \;\; 0,425 \;\; 0,125 \;\; 0,075]$ تکرار کنیم W_1 برنده می‌شود. سپس مقدار
جدید می‌گیرد.
W_1 جدید $= [0,05 \;\; 0,35 \;\; 0,4 \;\; 0,7]$

$W_1 = [0,05 \;\; 0,35 \;\; 0,4 \;\; 0,7]$ این مراحل را تا جایی انجام می‌دهیم تا به همگرایی
$W_2 = [0,975 \;\; 0,425 \;\; 0,125 \;\; 0,075]$ برسیم یعنی نورون‌های برنده تغییر نکنند و
تغییرات وزن‌ها بسیار کوچک شود و تقریباً به صفر برسد.

سوال دوم

بله، لیست [۱،۱،۱،۱]، [-۱،-۱،-۱،-۱]، [-۱،۱،-۱،۱]، [۱،۱،-۱،-۱] در شبکه هاپفیلد قابل ذخیره است. این بدان معنی است که شبکه می تواند با شروع از هر حالت اولیه دیگر به هر یک از این چهار حالت همگرا شود.(در حقیقت به الگویی همگرا می‌شود که شباهت بیشتری با آن دارد) یعنی این چهار حالت پایدار(stable) هستند و نقاط مینیمم محلی تابع انرژی می باشند. همچنین چون تعداد الگو ها کوچکتر مساوی تعداد نورون ها هستند با توجه اسلاید ها مشکلی وجود ندارد. برای مقدار دهی اولیه وزن های ماتریس از فرمول زیر که در اسلاید ها آمده است استفاده می‌کنیم. بعد از تشکیل ماتریس وزن از هر یک از pattern ها شروع می‌کنیم و اگر در همان حالت ماند و تغییر نکرد(ثابت ماند) نشان دهنده پایداری state آن الگو است. در واقع اگر از state الگو تغییری نکند نشان می دهد که که الگو نقطه مینیمم محلی تابع انرژی است و پایدار است. (برای تحقیق بیشتر می توانیم هر یک از ۱۶ حالت موجود را به شبکه دهیم و ببینیم که در نهایت به یکی از ۴ الگوی موجود همگرا خواهد شد.)

How Does It Work?

-The number of independent patterns or words that can be remembered is less than or equal to the number of nodes in the net.

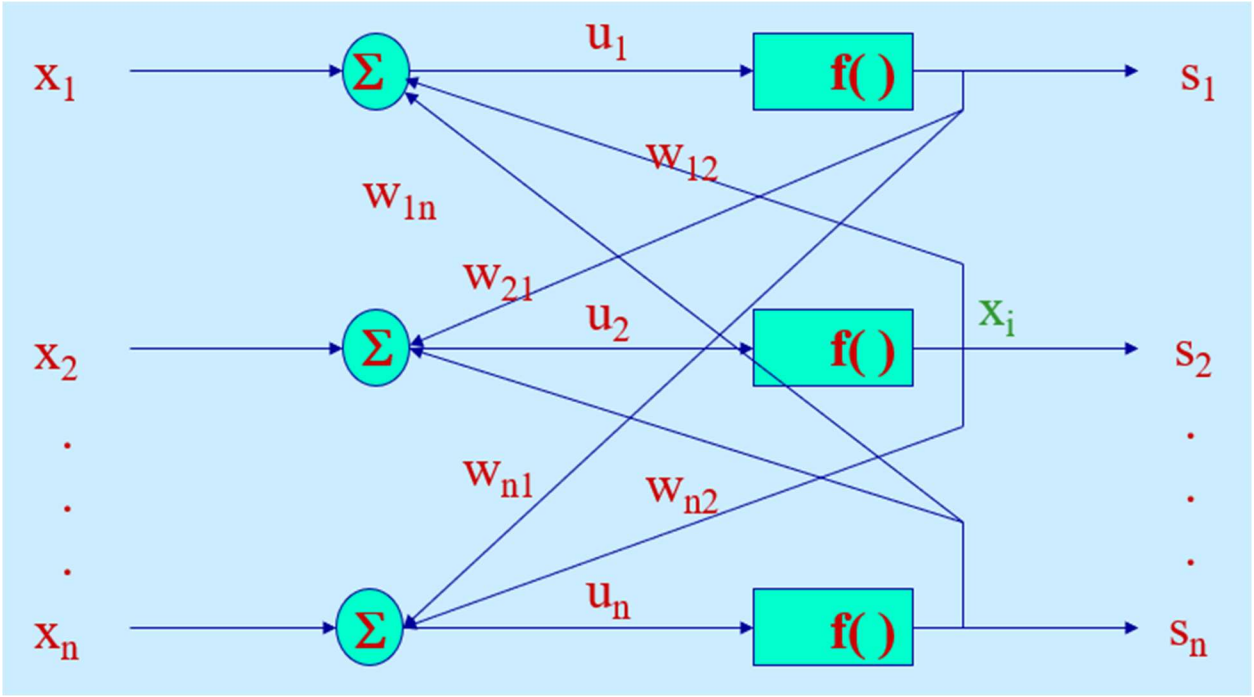
-Memory recall corresponds to a trajectory taking the system from some initial state (input) to the local energy minimum (closest association) .

-Each step along the (recall) trajectory results in the same or lower energy.

-Since energy is bounded from below, a solution is guaranteed for every input.

- Network Model and Operation
  - *The formal McCulloch-Pits neuron*
  - *One of the two definite states "**on**" or "**off**"*
  - *Synaptic connection w_{ji} between neuron i and j, determines the contribution of output x_i of neuron i to activation potential of neuron j*
  - *Weights determined according to a Hebbian rule:*

$$w_{ij} = \sum_{k=1}^{P} x_i^k x_j^k$$



جواب کامل و راه حل در ادامه آمده است:

Subject
Date

چون هر pattern داری ۴ ورودی است پس ۴ خروجی و درنتیجه ۴ نورون در شبکه داریم. همچنین با ۴ الگوی موجود وزن ها را تأیین می کنیم.

patterns

$x_1 = [1, 1, -1, -1]$

$x_2 = [-1, -1, 1, 1]$

$x_3 = [-1, -1, -1, -1]$

$x_4 = [1, 1, 1, 1]$

بله قابل ذخیره سازی است.

$$W_{ij} = \sum_{k=1}^{P} x_i^k x_j^k = \sum_{k=1}^{4} x_i^k x_j^k$$

$W_{ji} = W_{ij}$, $W_{ii} = 0$

$W_{00} = W_{11} = W_{22} = W_{33} = 0$, $W_{12} = W_{21} = 1+1+1+1 = 4$

$W_{13} = W_{31} = -1-1+1+1 = 0$, $W_{14} = W_{41} = -1-1+1+1 = 0$

$W_{23} = W_{32} = -1-1+1+1 = 0$, $W_{24} = W_{42} = -1-1+1+1 = 0$, $W_{34} = W_{43} = 1+1+1+1 = 4$

neuron i \ neuron 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

Weight Matrix

threshold = 0

bipolar valued hard limiter

$$f_0(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases}$$

حال با شروع از state هریک از pattern ها می بینیم که stable هستند یا نه.

الگوی x_4

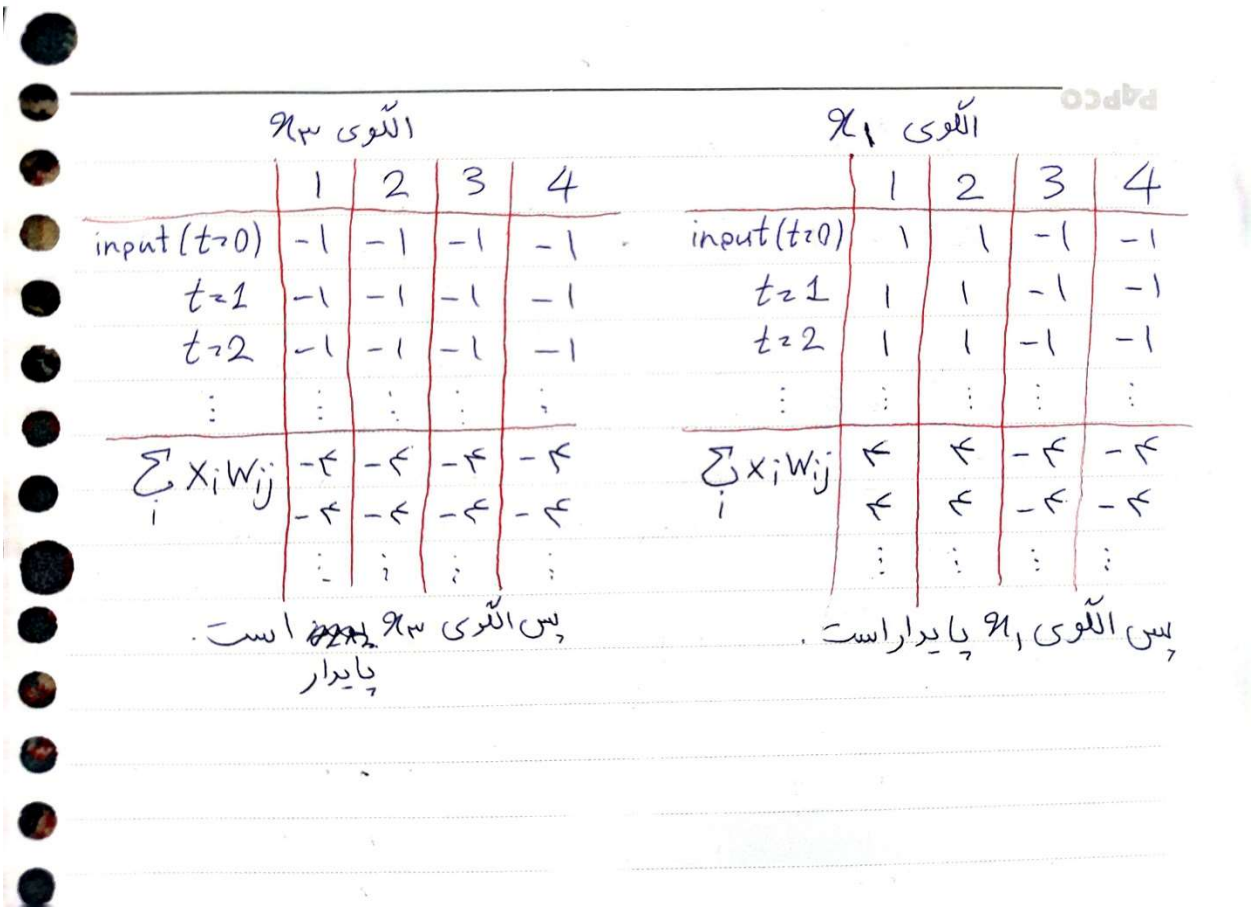
	1	2	3	4
input ($t=0$)	1	1	1	1
$t=1$	1	1	1	1
$t=2$	1	1	1	1
$\sum_i x_i w_{ij}$	4	4	4	4
	4	4	4	4

پس الگوی x_4 پایدار و نقطه min انرژی است.

الگوی x_2

	1	2	3	4
input ($t=0$)	-1	-1	1	1
$t=1$	-1	-1	1	1
$t=2$	-1	-1	1	1
$\sum_i x_i w_{ij}$	-4	-4	4	4
	-4	-4	4	4

پس الگوی x_2 پایدار و نقطه min انرژی است.



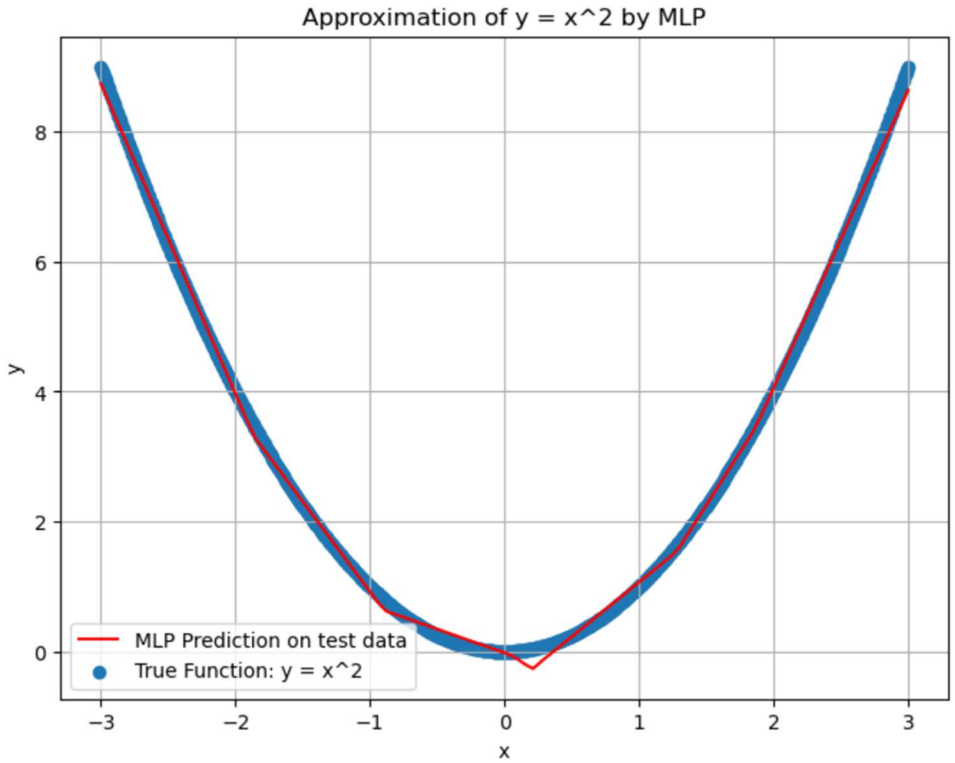
مراجع:

https://bard.google.com/

سوال سوم

این سوال در نوت بوک Q3.ipynb پیاده سازی شده است. با مراجعه به آن می توانید با جزییات بیشتر روند پیاده سازی را مشاهده کنید. کد ها به طور کامل با کامنت ها توضیح داده شده است و همچنین توضیحات تکمیلی در markdown ها قابل دیده شدن است. ابتدا نتیجه مدل را در زیر مشاهده میکنید و سپس به ساختار مدل و توضیح کد به شکل مختصر در اینجا می پردازیم.

```
Epoch [500/5000], Training Loss: 0.1156
Epoch [1000/5000], Training Loss: 0.0649
Epoch [1500/5000], Training Loss: 0.0455
Epoch [2000/5000], Training Loss: 0.0335
Epoch [2500/5000], Training Loss: 0.0255
Epoch [3000/5000], Training Loss: 0.0201
Epoch [3500/5000], Training Loss: 0.0163
Epoch [4000/5000], Training Loss: 0.0137
Epoch [4500/5000], Training Loss: 0.0118
Epoch [5000/5000], Training Loss: 0.0104
Test data loss: 0.0113
```



همان طور که میبینیم ضرر آموزش و تست (Training loss, Test loss) بعد از آموزش به مقدار قابل توجهی کاهش پیدا کرده است و همچنین در نمودار بالا میبینیم که مدل در بازه -۳ تا +۳ تابع y=x^2 را با تقریب خوبی پیش بینی کرده است. پس این مدل برای حل این مسئله نتیجه خوبی دارد.

توضیح در مورد ساختار مدل MLP:

کد ارائه شده یک شبکه عصبی پرسپترون چند لایه (MLP) ساده را برای تقریب تابع \(y = x^2 \) پیاده سازی می کند. بیایید معماری و اجزای آن را تجزیه کنیم:

معماری مدل:
۱. لایه ورودی:

- اندازه لایه ورودی مدل ۱ است، زیرا یک مقدار ورودی واحد x می گیرد.

۲. لایه پنهان:

- لایه پنهان از ۱۰ نورون تشکیل شده است. این لایه از تابع فعال سازی Rectified Linear Unit (ReLU) استفاده می کند.
- تابع فعال سازی ReLU (f(x) = max(0, x)) به دلیل سادگی و کارایی خود در شبکه های عصبی عمیق شناخته شده است. همچنین چون از یک طرف flat است و از طرف دیگر flat نیست نسبت به تابع سیگموئید عملکرد بهتری دارد. همچنین در اینجا ما با مقادیر پیوسته سروکار داریم ولی توابع سیگموئید یا tanh مقادیر باینری دارند به همین دلیل عملکرد بد تری نسبت به ReLU دارند.

۳. لایه خروجی:

- اندازه لایه خروجی ۱ است که یک خروجی واحد تولید می کند که هدف آن تقریبی مقدار (y) برای یک (x) معین است.
- هیچ تابع فعال سازی در لایه خروجی اعمال نمی شود. زیرا نیازی نداریم و مقادیر پیوسته حقیقی در اینجا مطلوب است.

تعداد لایه ها و نورون ها:

- مدل علاوه بر لایه ورودی دارای دو لایه است: یک لایه پنهان با ۱۰ نورون و یک لایه خروجی.
- تعداد نورون های لایه پنهان (۱۰ نورون) تا حدودی خودسرانه و دلخواه انتخاب شد و ممکن است اندازه متوسطی برای این مسئله در نظر گرفته شود. این یک هایپرپارامتر است و می تواند بر اساس آزمایش و الزامات تنظیم شود. بر اساس تجربه بنظر مقدار خوبی است که اعتدال را بین overfit و underfit حفظ کند.
- همچنین چون در اینجا مسئله نسبتاً ساده ای داریم بنظر استفاده از یک لایه پنهان کافی است.

تابع ضرر(loss function):

- تابع ضرر استفاده شده در این MLP خطای میانگین مربعات (MSE) است.
- \(\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_{\text{true}} - y_{\text{pred}})^2 \)
- میانگین اختلاف مجذور بین مقادیر پیش بینی شده و مقادیر واقعی را اندازه گیری می کند.

تابع فعال سازی:

- تابع فعال سازی ReLU در لایه پنهان اعمال می شود.
- ReLU غیر خطی بودن را معرفی می کند و به شبکه اجازه می دهد تا روابط پیچیده بین ورودی ها و خروجی ها را بیاموزد.
- سادگی آن در محاسبات و توانایی برای کاهش مشکل گرادیان ناپدید شدن، آن را به یک انتخاب محبوب تبدیل کرده است.

چرایی این معماری و مناسب بودن آن برای پیش‌بینی تابع (x^2):

- Non-Linearity Handling: فعال سازی ReLU در لایه پنهان مدل را قادر می سازد تا رابطه غیرخطی بین ورودی ها (x) و خروجی های مربعی (y) را ثبت کند.
- Expressiveness of Model: یک MLP ساده با یک لایه پنهان و فعال سازی ReLU می تواند توابع مختلف غیرخطی را به طور موثر تقریب بزند.
- Scalability: این معماری می تواند نقطه شروع خوبی برای وظایف تقریب تابع ساده باشد. نسبتا سبک است اما قادر به یادگیری الگوهای پیچیده است.

چرا برای پیش بینی تابع x^2 خوب است:

- شباهت در شکل: تابع x^2 یک رابطه غیر خطی است. سادگی و غیرخطی بودن ارائه شده توسط فعال سازی ReLU در لایه پنهان به مدل اجازه می دهد تا ماهیت درجه دوم x^2 را به تصویر بکشد.
- انعطاف پذیری و سازگاری: شبکه می تواند وزن های خود را در حین تمرین از طریق پس انتشار تنظیم کند تا افت میانگین مربعات خطا را به حداقل برساند، که به آن امکان می دهد به تدریج عملکرد درجه دوم را با دقت بیشتری تقریب کند.
- این معماری تعادلی بین پیچیدگی و سادگی برقرار می‌کند و آن را به نقطه شروع مناسبی برای تقریب توابع غیرخطی مانند (x^2) با یادگیری از داده‌ها و تنظیم وزن‌های آن بر این اساس از طریق آموزش تبدیل می‌کند.

توضیح کد:
این کد یک پرسپترون چندلایه (MLP) ساده را با استفاده از NumPy برای تقریب تابع y = x^2 پیاده سازی می کند. بیایید گام به گام کد را تجزیه کنیم:

وارد کردن کتابخانه ها
numpy برای عملیات عددی وارد شده است.
matplotlib.pyplot برای رسم نمودارها وارد شده است.

Seed را تنظیم میکنیم.
seed تصادفی را برای تکرارپذیری در تولید اعداد تصادفی تنظیم می کند.np.random.seed(85)

تابع True را تعریف میکنیم.
true_function(x) تابع y = x^2 را تعریف می کند.

کلاس لایه را تعریف میکنیم.
کلاس لایه یک لایه را در شبکه عصبی مقداردهی اولیه می کند.
__init__: وزن‌ها و بایاس‌ها را با مقادیر تصادفی برای اندازه ورودی و خروجی معین، مقداردهی می‌کند.
forward: با محاسبه خروجی لایه با استفاده از ورودی، وزن ها و بایاس ها، پاس رو به جلو را انجام می دهد.

کلاس MLP را تعریف میکنیم
کلاس MLP یک پرسپترون چند لایه ساده را نشان می دهد.
__init__: مدل MLP را با اندازه های لایه های ورودی، مخفی و خروجی همراه با نرخ یادگیری راه اندازی می کند.
relu: تابع فعال سازی Rectified Linear Unit (ReLU) را پیاده سازی می کند.
mean_squared_error: میانگین مربعات خطای خطا را محاسبه می کند.
Train: مدل MLP را با استفاده از گذر رو به جلو، محاسبه ضرر، و پس انتشار با نزول گرادیان، آموزش می‌دهد.
predict: با استفاده از مدل آموزش داده شده برای داده های آزمون، پیش بینی می کند.

تولید داده های آموزشی
x_train به عنوان داده ورودی در محدوده [-۳، ۳] تولید می شود.
y_train با اعمال تابع true در x_train تولید می شود.

ایجاد مدل MLP
نمونه‌ای از مدل MLP با اندازه‌های لایه ورودی، مخفی و خروجی خاص ایجاد می‌شود.

مدل را آموزش میدهیم
مدل با استفاده از داده های x_train و y_train آموزش داده شده است.

تولید داده های آزمایشی برای رسم
x_test برای اهداف ترسیم در همان محدوده x_train ایجاد شده است.
predicted_test پیش بینی های مدل را برای داده های x_test نگه می دارد.

نتایج را ترسیم میکنیم
یک نمودار برای تجسم تابع واقعی (y = x^2) و پیش‌بینی‌های MLP ایجاد می‌شود.

نمایش نمودار
در نهایت، نمودار نمایش داده می شود تا مقایسه بین تابع واقعی و تقریب MLP را نشان دهد.

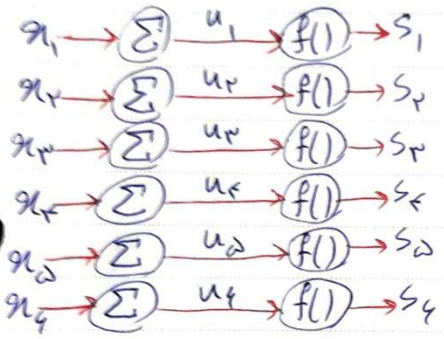
حلقه اصلی آموزشی شامل forward pass، محاسبه ضرر و پس انتشا برای به‌روزرسانی وزن‌ها و بایاس‌های شبکه است. مدل به تدریج یاد می گیرد که تابع واقعی y = x^2 را تقریب بزند.

مراجع:

https://chat.openai.com/
https://bard.google.com/

چون ۶ بیت داریم پس ۶ نورون برای شبکه در نظر می‌گیریم. رشته بیت ۱۱۱۱۰۰

باید ذخیره شود پس با این الگو شبکه را آموزش می‌دهیم. ۰۱۰۰۰۰ → ۱۱۱۱۰۰



$$u_i = \sum_{i=1}^{6} w_{ij} s_j \;,\; w_{ij} = w_{ji} \;,\; w_{ii} = 0$$

$$threshold = \frac{1}{2}$$

$$f_{\frac{1}{2}}(x) = \begin{cases} 1 & x > \frac{1}{2} \\ 0 & x \le \frac{1}{2} \end{cases}$$

binary valued hard limiter

Weight Matrix

neuron i

neuron j	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	1	1	0	0
3	1	1	0	1	0	0
4	1	1	1	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

فقط یک الگو داریم. pattern = 111100

$$W_{ij} = \sum_{k=1}^{1(P)} x_i^k x_j^k = x_i^1 x_j^1$$

$$W_{11} = W_{22} = W_{33} = W_{44} = W_{55} = W_{66} = 0$$

$$W_{12} = W_{21} = 1 \;,\; W_{13} = W_{31} = 1 \;,\; W_{14} = W_{41} = 1$$
$$W_{15} = W_{51} = 0 \;,\; W_{16} = W_{61} = 0 \;,\; W_{23} = W_{32} = 1$$
$$W_{24} = W_{42} = 1 \;,\; W_{25} = W_{52} = 0 \;,\; W_{26} = W_{62} = 0$$
$$W_{34} = W_{43} = 1 \;,\; W_{35} = W_{53} = 0 \;,\; W_{36} = W_{63} = 0$$
$$W_{45} = W_{54} = 0 \;,\; W_{46} = W_{64} = 0 \;,\; W_{56} = W_{65} = 0$$

	۱	۲	۳	۴	۵	۶
input ($t=0$)	۰	۱	۰	۰	۰	۰
$t=1$	۱	۰	۱	۱	۰	۰
$t=2$	۱	۱	۱	۱	۰	۰
$t=3$	۱	۱	۱	۱	۰	۰
$\sum_i x_i w_{ij}$	۱	۰	۱	۱	۰	۰
	۲	۳	۲	۲	۰	۰
	۳	۳	۳	۳	۰	۰

به نقطه min انرژی رسیدیم. بعد از آن تغییری نداریم و الگو بازیابی شد به عبارتی دیگر به الگوی ۱۱۱۱۰۰ همگرا شده‌ایم.

Subject :

سوال پنجم

مسئله فروشنده دوره گرد یک مسئله شناخته شده در علوم کامپیوتر است: این مسئله شامل یافتن کوتاه ترین مسیر ممکن است که تمام شهرها را در یک نقشه معین فقط یک بار طی می کند. اگرچه توضیح ساده آن، این مسئله در واقع NP-Complete است. این نشان می‌دهد که دشواری حل آن با تعداد شهرها به سرعت افزایش می‌یابد و ما در واقع راه‌حل کلی که مسئله را حل کند نمی‌دانیم. به همین دلیل، هر روشی که بتواند راه‌حل sub-optimal را پیدا کند، عموماً به اندازه کافی خوب است.

در بین سه شبکه عصبی ذکر شده، شبکه های هاپفیلد(Hopfield networks) و SOM (Kohonen) پتانسیل حل مسئله فروشنده دوره گرد (TSP) را دارند. مورد دیگر، یعنی MLP (پرسپترون چند لایه)، به دلیل محدودیت‌های ذاتی برای این کار مناسب نیست.

*** منظور از تور(tour) در جواب های زیر مسیری است که از یک شهر شروع می شود و از تمامی شهر ها دقیقا یکبار عبور کند و به شهر شروع بازگردد.

شبکه هاپفیلد و TSP

شبکه های هاپفیلد(Hopfield networks) نوعی شبکه عصبی بازگشتی هستند که قادر به ذخیره و بازیابی خاطرات(memories) هستند و آنها را برای مسائل بهینه سازی ترکیبی(combinatorial optimization problems) مانند TSP مناسب می کند. در زمینه TSP، یک شبکه هاپفیلد می تواند برای رمزگذاری(encode) مسئله به عنوان یک تابع انرژی استفاده شود، که در آن انرژی هزینه کل یک تور خاص(particular tour) را نشان می دهد. سپس شبکه به‌طور مکرر وضعیت نورون‌های خود را به‌روزرسانی می‌کند تا این انرژی را به حداقل برساند، و در نهایت به راه‌حلی همگرا می‌شود که نشان‌دهنده یک تور کم‌هزینه است.

الگوریتم برای حل TSP با استفاده از شبکه های هاپفیلد

۱. رمزگذاری مسئله(Encoding the Problem): هر شهر را به عنوان یک نورون در شبکه هاپفیلد نشان دهید. وزن بین نورون ها نشان دهنده فاصله بین شهرها است.
۲. تابع انرژی(Energy Function): تابع انرژی را تعریف کنید که مسافت های طولانی را جریمه می کند و دقیقاً یک بار بازدید از همه شهرها را پاداش می دهد.
۳. قانون به روز رسانی شبکه(Network Update Rule): وضعیت هر نورون را بر اساس اصل کمینه سازی انرژی محلی با هدف کاهش انرژی کلی شبکه به روز کنید.
۴. همگرایی(Convergence): شبکه به روز رسانی ادامه می دهد تا زمانی که به وضعیت پایدار برسد، که نشان دهنده یک تور کم هزینه است.

ساختار شبکه

شبکه هاپفیلد برای TSP دارای یک ساختار کاملاً متصل است که در آن هر نورون به هر نورون دیگر متصل است. هر نورون یک شهر را مدل می‌کند و وزن های بین نورون ها فاصله بین شهر ها را مدل می‌کند. همچنین تابع انرژی نیز هزینه تور را مدل می‌کند. این به شبکه اجازه می دهد تا روابط جهانی بین شهرها و فواصل آنها را به تصویر بکشد.

در نتیجه، شبکه های هاپفیلد به دلیل توانایی آنها در ذخیره و بازیابی خاطرات(memories) و توانایی آنها برای به حداقل رساندن توابع انرژی، رویکرد امیدوارکننده ای را برای حل TSP ارائه می دهند. با این حال، توجه به این نکته مهم است که شبکه های هاپفیلد ممکن است همیشه راه حل بهینه را پیدا نکنند(مثلاً در مینیمم محلی گیر کند) و عملکرد آنها می تواند به انتخاب پارامترها حساس باشد.

حل TSP با استفاده از شبکه های خود سازمانده (SOM)

SOM(Kohonen) در درجه اول برای کارهای یادگیری بدون نظارت (unsupervised learning tasks) مانند خوشه‌بندی(clustering) و کاهش ابعاد (dimensionality reduction) استفاده می شود. برای حل مسائل بهینه‌سازی مانند TSP که به استراتژی‌های جستجوی صریح نیاز دارد، باید آن را کمی تغییر دهیم. برای حل مسئله، می‌توانیم سعی کنیم اصلاحی از تکنیک نقشه خودسازمانده (SOM) اعمال کنیم. اجازه دهید نگاهی به این تکنیک بیندازیم، و پس از درک بهتر آن را در TSP اعمال کنیم.

برای استفاده از شبکه برای حل TSP، مفهوم اصلی برای درک چگونگی تغییر تابع همسایگی است. اگر به جای یک شبکه(gird)، یک آرایه دایره ای از نورون ها(a circular array of neurons) را اعلام کنیم، هر گره فقط از نورون های جلو و پشت خود آگاه خواهد بود. یعنی شباهت درونی فقط در یک بعد کار خواهد کرد. با انجام این اصلاح جزئی، نقشه خودسازماندهی مانند یک حلقه الاستیک رفتار می کند و به شهرها نزدیکتر می شود اما به لطف تابع همسایگی سعی می کند محیط(perimeter) آن را به حداقل برساند.

اگرچه این اصلاح ایده اصلی پشت این تکنیک است، اما آنطور که هست کار نمی کند: الگوریتم به سختی در هیچ یک از زمان ها همگرا می شود. برای اطمینان از همگرایی آن، می‌توانیم نرخ یادگیری، آلفا را برای کنترل کاوش(exploration) و بهره‌برداری (exploitation) از الگوریتم در نظر بگیریم. برای به دست آوردن اکتشاف(exploration) بالا در ابتدا، و پس از آن بهره‌برداری (exploitation) بالا در اجرا، ما باید هم در تابع همسایگی و هم در نرخ یادگیری یک کاهش(decay) را لحاظ کنیم. کاهش نرخ یادگیری، جابجایی تهاجمی کمتری(less aggressive displacement) از نورون‌های اطراف مدل را تضمین می‌کند و کاهش همسایگی (decaying the neighborhood) منجر به بهره‌برداری متوسط‌تر از حداقل محلی هر بخش از مدل می‌شود. سپس، رگرسیون ما را می توان به صورت زیر بیان کرد:

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

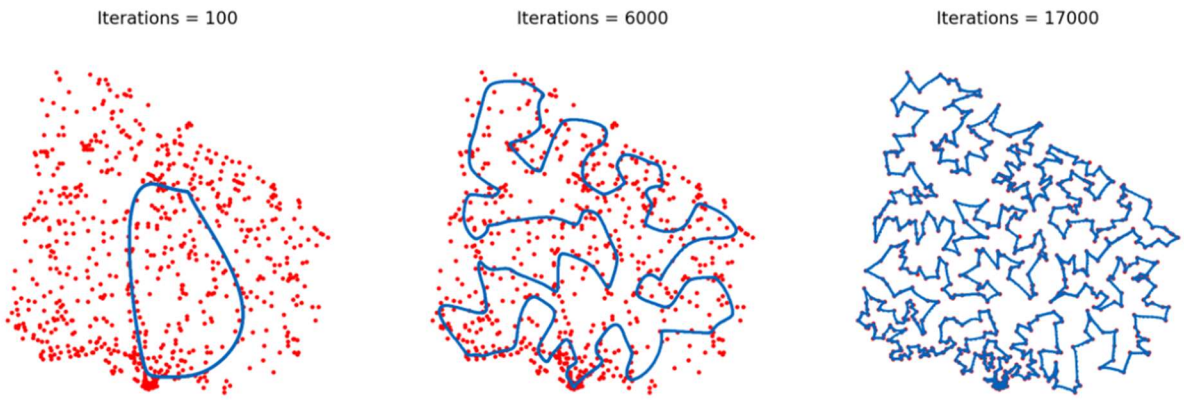
که در آن آلفا نرخ یادگیری در یک زمان معین است و (g) تابع گاوسی است به مرکزیت وزن های برنده و با پراکندگی همسایگی h متمرکز شده است. تابع کاهش(decay) شامل ضرب دو تخفیف داده شده، گاما، برای نرخ یادگیری و فاصله همسایگی است.

$$\alpha_{t+1} = \gamma_\alpha \cdot \alpha_t, \quad h_{t+1} = \gamma_h \cdot h_t$$

این عبارت در واقع کاملاً شبیه به Q-Learning است و همگرایی به روشی مشابه با این تکنیک جستجو می شود. تحلیل رفتن پارامترها می تواند در کارهای یادگیری بدون نظارت مانند موارد فوق مفید باشد. همچنین شبیه به عملکرد تکنیک Quantization بردار یادگیری است که توسط Teuvo Kohonen توسعه یافته است.

در نهایت، برای به دست آوردن مسیر از SOM، فقط لازم است یک شهر را با نورون برنده اش مرتبط کنیم، حلقه را از هر نقطه شروع کنیم و شهرها را بر اساس ترتیب ظاهر شدن نورون برنده آنها در حلقه مرتب کنیم. اگر چندین شهر به یک نورون نگاشت می شوند، به این دلیل است که ترتیب عبور از چنین شهرهایی توسط SOM در نظر گرفته نشده است (به دلیل عدم ارتباط با فاصله نهایی یا به دلیل عدم دقت کافی). در آن صورت می توان هرگونه ترتیب ممکن را برای چنین شهرهایی در نظر گرفت.

شکل زیر نحوه اجرای الگوریتم در تکرار های متوالی را نشان می دهد که در لینک مرجع پیاده سازی شده است.



محدودیت های MLP

MLP (پرسپترون چند لایه): MLP برای وظایف یادگیری تحت نظارت (supervised learning tasks) مانند طبقه بندی (calssification) و رگرسیون (regression) طراحی شده است، جایی که یاد می گیرد ورودی ها را به خروجی ها بر اساس داده های برچسب گذاری شده (labeled data) نگاشت کند. ولی ما در اینجا داده های برچسب گذاری شده نداریم و صرفا شهر ها و فاصله بین آنها را داریم نه مسیر های بهینه که بتوانیم برای آموزش از آنها استفاده کنیم. در حالی که MLP می تواند برای مسائل تقریبی(approximation problems) استفاده شود، با مسائل بهینه سازی ترکیبی مانند TSP تقلا می کند و نمی‌تواند آن ها را حل کند.

همچنین در زیر روش حل این مسئله به شکل کامل تری با Hopfield آمده است که لینک مقاله آن در مراجع ذکر شده است.

In 1985, Hopfield designed the fully connected network which is later known as Hopfield neural network [14]. He simulated the TSP of 10 cities and 30 cities respectively. Moreover, it was the first algorithm that used neural network to solve the optimization problem of TSP. As shown in Fig. 1, the algorithm idea is to convert the objective function into the energy function of the neural network, and minimize the energy function in the process of running the neural network so as to obtain the local optimal solution. Hopfield defined the energy function and the equations of motion of the TSP problem in Eqs. (5) and (6).

$$E = \frac{A}{2}\sum_{x=1}^{N}\sum_{i=1}^{N}\sum_{j=1}^{N} X_{xj}X_{xj} + \frac{B}{2}\sum_{i=1}^{N}\sum_{x=1}^{N}\sum_{y=x}^{N} X_{xi}X_{yi} + \frac{C}{2}\left(\sum_{x=1}^{N}\sum_{i=1}^{N} X_{xi} - N\right)^2 + \frac{D}{2}\sum_{x=1}^{N}\sum_{y=1}^{N}\sum_{i=1}^{N} d_{xy}X_{xi}\left(X_{y,i+1} + X_{y,i-1}\right) \quad (5)$$

$$\frac{dU_{xi}}{dt} = -\frac{\partial E}{\partial X_{xi}} = -A\left(\sum_{i=1}^{N} X_{xi} - 1\right) - B\left(\sum_{y=1}^{N} X_{yi} - 1\right) - D\sum_{y=1}^{N} d_{xy}X_{y,i+1} \quad (6)$$

Here A, B and C are all positive, which is equivalent to the three penalty terms plus the target function.
Here is a two-layer Hopfield neural network in Fig. 1. The zero layer is the input of the network, and the first layer is the node of the neuron. The neuronal input includes the external input and the feedback of neuronal output in Eq. (7).

$$z_j = \sum_{i}^{N} w_{ij}y_i + x_j \quad (7)$$

Here w_{ij} is weight, $y_i = [y_1, y_2, \cdots, y_N]$ is the neuronal output and $x_j = [x_1, x_2, \cdots, x_N]$ is the external input.
The difference between Hopfield network and other neural networks is that its network weights are not determined through repeated learning, but are directly given, and then the state of the network is updated according to the operating equation of the network, and finally the local optimal solution is reached. Hopfield found that this neural network worked very well for problems smaller than 30 cities. However, It did not apply when the TSP is larger than 30 cities.
Although Hopfield network solves TSP almost perfectly, it also has some shortcomings, which have been optimized by many researchers. Luo [15] improved the third term of the energy function in view of its poor convergence; In addition, Qiao et al [16] also improved the energy function by adding a correction term to the energy function and establishing a new hysteretic noisy frequency conversion sinusoidal chaotic neural network (HNFCSCNN) to solve TSP which received a satisfactory result.

Li et al [17] improved the connection weight according to its shortcoming that it was easy to fall into the local minimum. They changed the connection weights based on the performance of the objective function. García [18] adopted the divide-and-conquer method to improve the performance of Hopfield network in TSP.

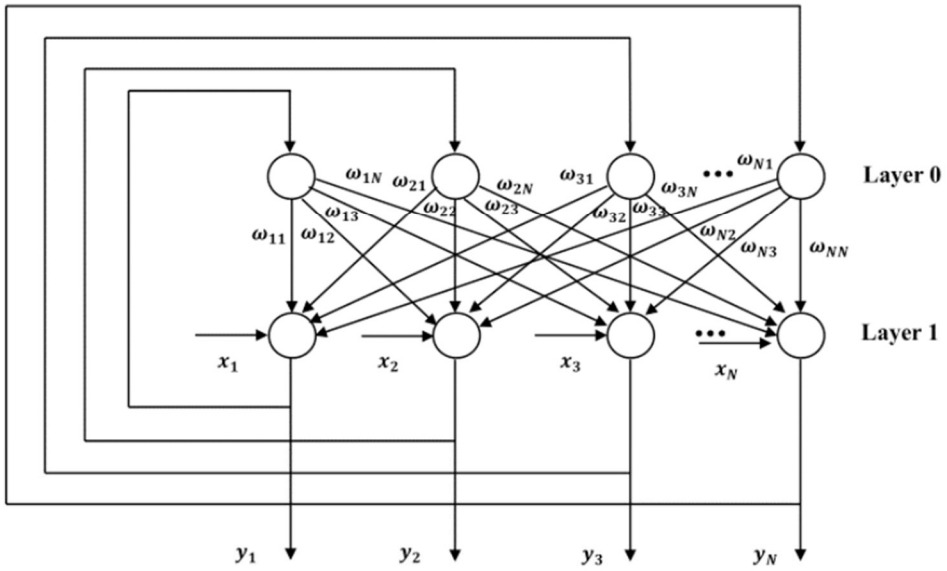


Fig. 1. Two layer of Hopfield network

مراجع:

https://chat.openai.com/
https://bard.google.com/
https://www.researchgate.net/publication/231761748_Comparison_of_Neural_Networks_for_Solving_the_Travelling_Salesman_Problem
https://diego.codes/post/som-tsp/

پایان