به نام خدا

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

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

گزارش تمرین 3:

سوال اول:

کد این سوال در نوت بوک HW3_Q1 موجود است.

ابتدا شبکه Hopfield را پیاده سازی می کنیم. کلاس Hopfield دارای متد store است که شبکه را روی pattern های داده شده آموزش می دهد و آن ها را ذخیره می کند. متد check_pattern نیز یک pattern را گرفته و stable بودن آن را چک می کند، همچنین نزدیک ترین pattern شبیه به آن را برمی گرداند و مقدار دقت و انرژی را گزارش می کند.

برای store پترن های ورودی، ماتریس وزن با توجه به لیست پترن ها و Hebian Rule محاسبه می‌شود.

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

سپس برای بررسی پایدار بودن ورودی جدید، طی چند Epoch مقادیر a و انرژی را به دست می‌آوریم.

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

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

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

$$E(total) = \sum_{i=1}^{N} E(i,t) = -\sum_{i=1}^{N} a(i,t)\left(\sum_{j=1}^{N} w_{i,j} \, a(j,t)\right) = -\sum_{i=1}^{N}\sum_{j=1}^{N} w_{i,j} a(i,t) a(j,t)$$

این کار را تا زمانی ادامه می‌دهیم که یکی از سه حالت زیر رخ دهد:

- در لحظه اول a به دست آمده با ورودی برابر باشد. در این صورت ورودی پایدار بوده است.
- دو a آخر و پشت سر هم یکسان باشند. در این صورت در میان لیست a های ذخیره شده a ای که کمترین انرژی را دارد خروجی می‌دهیم. (convergence)
- در صورتی که اتفاقات بالا نیفتاد پترن به یک حالت پایدار همگرا نمی شود. چون ما فرض کردیم تغییرات شبکه به صورت همزمان هستند اما در واقع آسنکرون است. در این صورت تمام Epoch ها کامل اجرا می‌شوند و سپس از لیست a ها a ای که کمترین انرژی را دارد برمی‌گردانیم.

1- پس از پیاده سازی، پترن های داده شده را در شبکه ذخیره می کنیم. ماتریس وزن نتیجه:

```
3 patterns stored.

Weight Matrix:

 [[ 0  1 -1  3  1 -1  3 -1]
 [ 1  0  1  1 -1  1  1  1]
 [-1  1  0 -1  1 -1 -1  3]
 [ 3  1 -1  0  1 -1  3 -1]
 [ 1 -1  1  1  0 -3  1  1]
 [-1  1 -1 -1 -3  0 -1 -1]
 [ 3  1 -1  3  1 -1  0 -1]
 [-1  1  3 -1  1 -1 -1  0]]
```

2- پایدار بودن هر کدام از الگو ها را چک می کنیم.

پترن اول:

```
Pattern: [-1 -1  1 -1  1 -1 -1  1]
Pattern is stable.
Nearest pattern found in iteration: 1
Nearest pattern: [-1 -1  1 -1  1 -1 -1  1]
Accuracy: 100.0 %
Energy: -44
```

پترن دوم:

```
Pattern: [-1 -1 -1 -1 -1  1 -1 -1]
Pattern is stable.
Nearest pattern found in iteration: 1
Nearest pattern: [-1 -1 -1 -1 -1  1 -1 -1]
Accuracy: 100.0 %
Energy: -44
```

پترن سوم:

```
Pattern: [-1  1  1 -1 -1  1 -1  1]
Pattern is stable.
Nearest pattern found in iteration: 1
Nearest pattern: [-1  1  1 -1 -1  1 -1  1]
Accuracy: 100.0 %
Energy: -48
```

شبکه هر سه پترن را پایدار تشخیص داد.

3- چک کردن الگو های نویزی:

الگوی اول:

```
Pattern: [1, -1, 1, -1, 1, -1, -1, 1]
converged to same pattern in iteration: 1
Nearest pattern found in iteration: 1
Nearest pattern: [-1 -1  1 -1  1 -1 -1  1]
Accuracy: 87.5 %
Energy: -44
```

پترن نویزی اول که تنها یک بیت نویزی داشت، در iteration = 1 به پترن پایدار اول (X1) همگرا شد.

الگوی دوم:

```
Pattern: [1, 1, -1, -1, -1, 1, -1, -1]
Nearest pattern found in iteration: 2
Nearest pattern: [ 1  1 -1 -1 -1  1 -1 -1]
Accuracy: 100.0 %
Energy: -12
```

پترن دوم که دو بیت نویزی داشت، به پترن خاصی همگرا نشد. برای این پترن تمام epoch ها اجرا شد و از لیست a ها a ای که کمترین انرژی را دارد به عنوان nearest pattern برگردانده شد.

الگوی سوم:

```
Pattern: [1, 1, 1, -1, 1, 1, -1, 1]
Nearest pattern found in iteration: 2
Nearest pattern: [-1 -1  1 -1  1  1 -1  1]
Accuracy: 75.0 %
Energy: -32
```

پترن سوم نیز به پترن خاصی همگرا نشد. برای این پترن تمام epoch ها اجرا شد و از لیست a ها a ای که کمترین انرژی را دارد به عنوان nearest pattern برگردانده شد.

4- غیر از الگوهایی که شبکه را با آنها آموزش دادیم، الگوهای پایدار دیگری نیز داریم که شبکه به سمت آنها همگرا خواهد شد. برای مثال، حالت‌های زیر نیز دارای مینیمم انرژی هستند و می توانند ذخیره شوند (دسته اول حتما ذخیره خواهد شد.):

- دسته اول: reverse sign (چون معکوس هر پترن پایدار هم دارای انرژی برابر انرژی پترن است)
- دسته دوم: ترکیب خطی از پترن ها (mixture states) (تعداد حالت های این دسته مشخص نیست، ممکن است 0 یا چند تا از این حالت ذخیره شود.)

برای بدست آوردن الگو های پایداری که در این شبکه ذخیره شده اند متد find_stable_patterns را به کلاس Hopfield اضافه می کنیم. این متد با توجه به تعداد نورون ها، همه الگو های ممکن را می‌سازد. و برای هر کدام پایدار بودن آن الگو را توسط متد check_pattern بررسی می کند. سپس پترن های پایدار را بر می گرداند. خروجی این متد برای شبکه ما:

```
stable patterns count: 6
these patterns are stable:

[-1, -1, -1, -1, -1, 1, -1, -1]
[-1, -1, 1, -1, 1, -1, -1, 1]
[-1, 1, 1, -1, -1, 1, -1, 1]
[1, -1, -1, 1, 1, -1, 1, -1]
[1, 1, -1, 1, -1, 1, 1, -1]
[1, 1, 1, 1, 1, -1, 1, 1]
```

همانطور که میبینیم 6 پترن پایدار ذخیره شده است که 3 پترن پایدار اول، پترن های خودمان هستند که شبکه را با آن ها آموزش دادیم. و 3 تای بعدی، معکوس 3 پترن اول هستند.

5- دو مثال را برای این موضوع امتحان می کنیم، یک پترن که دارای 6 بیت نویز است را به شبکه می‌دهیم. (به پترن X3، نویز اضافه کرده ایم)

```
Pattern: [-1, 1, -1, 1, 1, -1, 1, -1]
converged to same pattern in iteration: 1
Nearest pattern found in iteration: 1
Nearest pattern: [ 1 -1 -1  1  1 -1  1 -1]
Accuracy: 75.0 %
Energy: -48
```

این پترن حاصل اضافه شدن نویز به X3 بود اما در iteration = 1 به یک پترن دیگر همگرا شد. (چون نویز آن زیاد بود و از X3 خیلی دور بود و به یک پترن stable دیگر نزدیک تر بود.)
یک مثال دیگر هم امتحان می کنیم. یک پترن که دارای 4 بیت نویز است را به شبکه می‌دهیم. (به پترن X2، نویز اضافه کرده ایم.)

```
Pattern: [1, -1, 1, -1, 1, -1, -1, -1]
Nearest pattern found in iteration: 2
Nearest pattern: [ 1 -1  1  1  1 -1  1 -1]
Accuracy: 75.0 %
Energy: -28
```

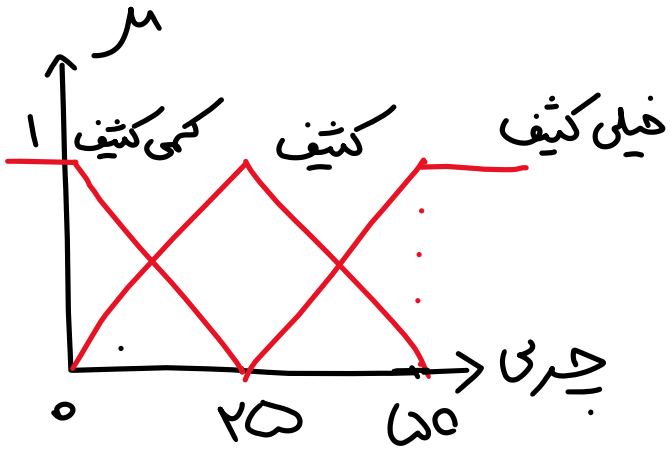
این پترن همگرا نشد. چون ما فرض کردیم تغییرات شبکه به صورت همزمان هستند اما در واقع آسنکرون است. به همین دلیل، برای این پترن تمام epoch ها اجرا شد و از لیست a ها a ای که کمترین انرژی را دارد به عنوان nearest pattern برگردانده شد.

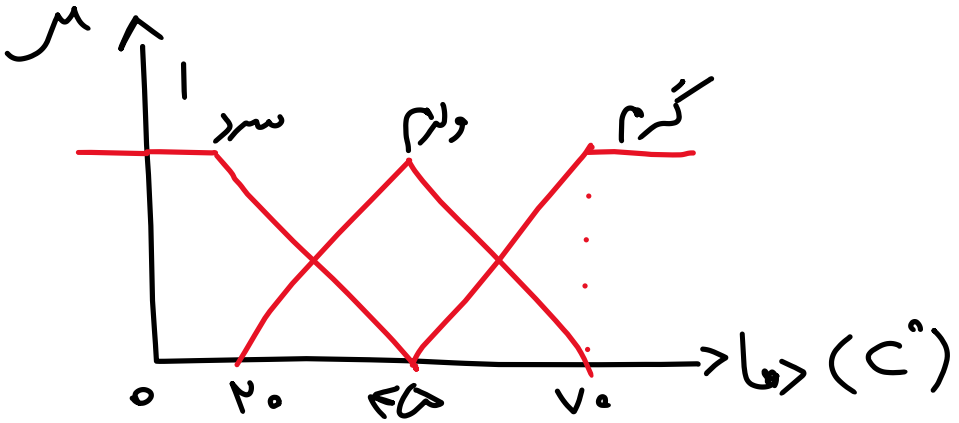
پس در حالتی که ورودی نویزی به شبکه بدهیم این دو حالت ممکن است پیش بیاید (قبلا در ابتدای سوال، راجب 3 حالت کلی که پیش می آید نوشته بودم.)

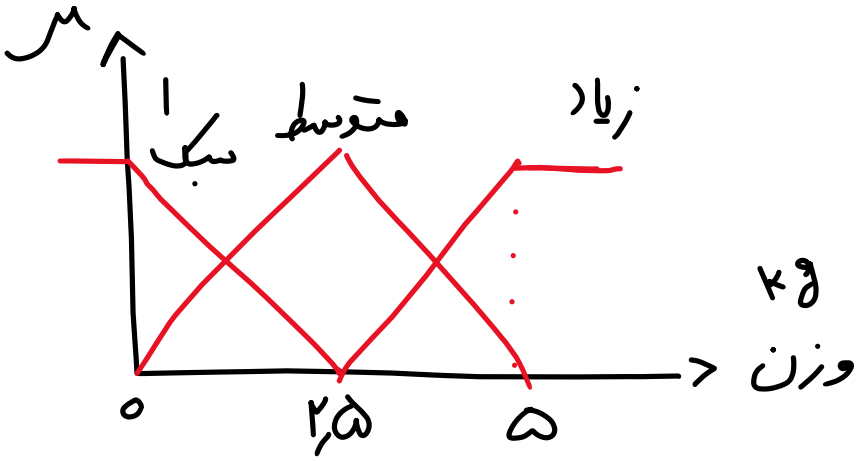
هرچه نویز پترن کمتر باشد و به پترن های stable نزدیک تر باشد امکان converge شدن بیشتر است و زود تر این اتفاق می افتد. اگر از پترن های stable دور باشد باید برای آن epoch بیشتری اجرا شود و در نهایت اگر همگرا نشود نزدیک ترین a به آن برگردانده می شود.

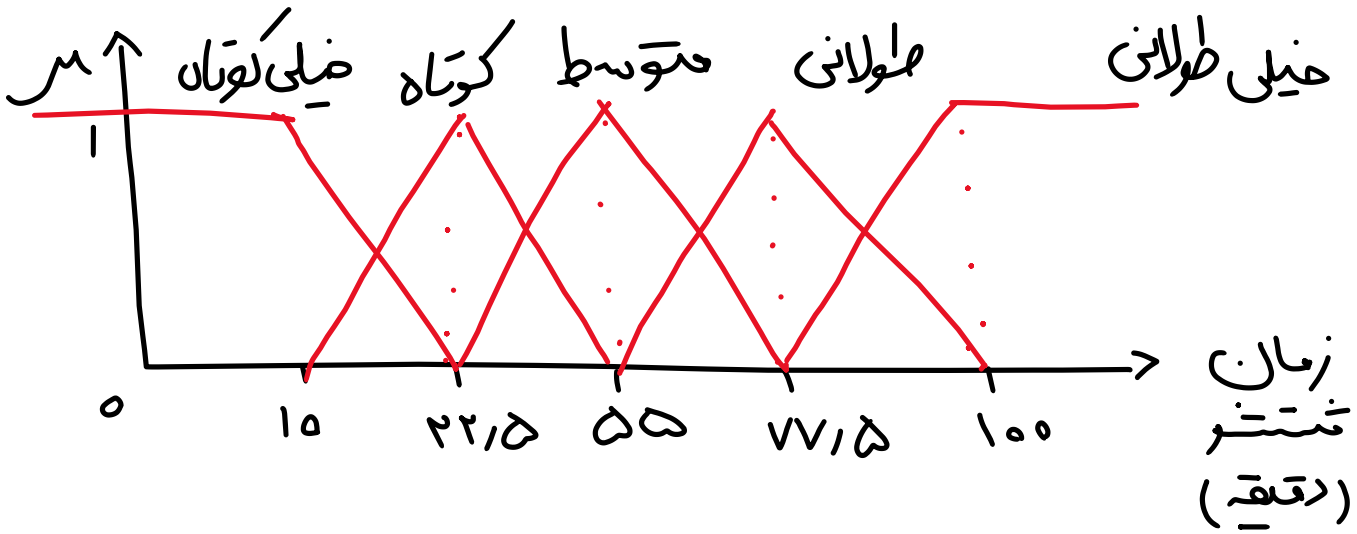
سوال دوم:

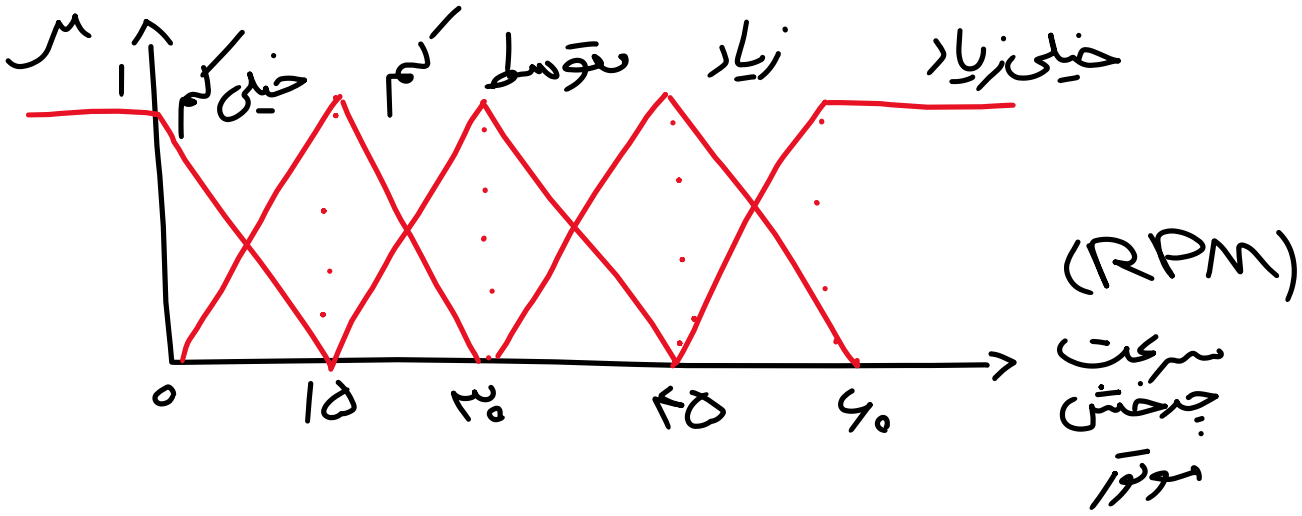
ابتدا کنترلر فازی خود را طراحی می کنیم. نمودار های متغیر های فازی را رسم می کنیم.











- چربی: [0, 50] ← (کمی کثیف، کثیف، خیلی کثیف)
- دما: [20, 70] ← (سرد، ولرم، گرم)
- وزن: [0, 5] ← (سبک، متوسط، زیاد)
- زمان شستشو: [10, 100] ← (خیلی کوتاه، کوتاه، متوسط، طولانی، خیلی طولانی)
- سرعت چرخش موتور: [0, 60] ← (خیلی کم، کم، متوسط، زیاد، خیلی زیاد)

ورودی ها: چربی، دما، وزن

خروجی ها: زمان شستشو، سرعت چرخش موتور

جداول روابط ورودی ها با خروجی ها به صورت زیر هستند:

وقتی دما سرد است:

وزن / چربی	کمی کثیف	کثیف	خیلی کثیف
سبک	زمان کوتاه، سرعت زیاد	زمان خیلی طولانی، سرعت زیاد	
متوسط	زمان متوسط، سرعت خیلی زیاد		
زیاد			زمان خیلی طولانی، سرعت خیلی زیاد

وقتی دما ولرم است:

وزن / چربی	کمی کثیف	کثیف	خیلی کثیف
سبک			
متوسط			
زیاد	زمان طولانی، سرعت متوسط		زمان خیلی طولانی، سرعت خیلی کم

وقتی دما گرم است:

وزن / چربی	کمی کثیف	کثیف	خیلی کثیف
سبک			
متوسط			
زیاد		زمان نسبتا طولانی، سرعت کمی زیاد	

شرایط داده شده:

وزن = 4 kg → $\frac{1}{2.5} = 0.4$ متوسط, $\frac{1.5}{2.5} = 0.6$ زیاد → وزن زیاد

چربی = 45 → $\frac{5}{25} = 0.2$ کثیف, $\frac{20}{2.5} = 0.8$ خیلی کثیف → خیلی کثیف

وزن = 20 ℃ → سرد

طبق جدول ترم های فازی داریم: اگر ظرف خیلی کثیف، وزن زیاد و دمای آب سرد باشد، باید سرعت موتور خیلی زیاد و زمان شستشو خیلی طولانی باشد.

از قائده max min ممدانی استفاده می کنیم.

$$\min(0.6, 0.8) = 0.6, \min(0.6, 1) = 0.6, \min(0.8, 1) = 0.8 \rightarrow \max(0.6, 0.6, 0.8) = 0.8$$

در نتیجه خروجی ها را بدست می آوریم:

سرعت چرخش موتور: $\frac{[45+(0.6\times(60-45)+60]}{2} = 57\ RPM$

زمان شستشو: $\frac{[77.5+(0.6\times(100-77.5)+100]}{2} = 95.5\ min$

سوال سوم:

کد این سوال در نوت بوک HW3_Q3 موجود است.

ابتدا سیستم فازی خود را می‌سازیم، و ترم های خواسته شده را به صورت triangle functions تعریف می‌کنیم.

```
from simpful import *
```

```
FS = FuzzySystem()

temperature = AutoTriangle(
    3, terms=['cold', 'normal', 'hot'],
    universe_of_discourse=[0, 70]
)
humidity = AutoTriangle(
    5, terms=['very_low', 'low', 'normal', 'high', 'very_high'],
    universe_of_discourse=[0, 100]
)
raining = AutoTriangle(
    3, terms=['a_little', 'normal', 'a_lot'],
    universe_of_discourse=[0, 10000]
)
height = AutoTriangle(
    5, terms=['very_low', 'low', 'normal', 'high', 'very_high'],
    universe_of_discourse=[0, 5000]
)
```

```
 ____  __  _  _  ____  ____  _  _  __
/ ___)(  )( \/ )(  _ \(  __)/ )( \(  ) v2.9.0
\___ \ )( / \/ \ ) __/ ) _) ) \/ (/ (_/\
(____/(__)\_)(_/(__)  (__)  \____/\____/

Created by Marco S. Nobile (m.s.nobile@tue.nl)
and Simone Spolaor (simone.spolaor@unimib.it)
```

سپس variable های خود را به سیستم فازی اضافه می کنیم تا آن ها را بشناسد.

```
FS.add_linguistic_variable("today_temperature", temperature)
FS.add_linguistic_variable("yesterday_temperature", temperature)
FS.add_linguistic_variable("two_days_ago_temperature", temperature)
FS.add_linguistic_variable("three_days_ago_temperature", temperature)
FS.add_linguistic_variable("humidity", humidity)
FS.add_linguistic_variable("raining", raining)
FS.add_linguistic_variable("height", height)
```

پس از آن، rule های فازی خود را اضافه می کنیم. (rule 10)

```
Rules = [
    "IF (humidity IS low) THEN (today_temperature IS hot)",
    "IF (yesterday_temperature IS cold) AND (two_days_ago_temperature IS cold) AND (three_days_ago_temperature IS cold) THEN (today_temperature IS cold)",
    "IF (yesterday_temperature IS normal) AND (height IS low) THEN (today_temperature IS normal)",
    "IF (three_days_ago_temperature IS hot) AND (raining IS normal) THEN (today_temperature IS hot)",
    "IF (height IS high) AND (humidity IS very_high) THEN (today_temperature IS cold)",
    "IF (yesterday_temperature IS hot) AND (two_days_ago_temperature IS hot) AND (three_days_ago_temperature IS hot) THEN (today_temperature IS hot)",
    "IF (yesterday_temperature IS cold) OR (height IS very_high) THEN (today_temperature IS cold)",
    "IF (two_days_ago_temperature IS hot) OR (raining IS a_little) THEN (today_temperature IS hot)",
    "IF (humidity IS normal) OR (raining IS normal) THEN (today_temperature IS normal)",
    "IF (yesterday_temperature IS normal) AND (raining IS a_little) THEN (today_temperature IS normal)"
]
FS.add_rules(Rules, verbose=True)
```

Variable هایی که تعریف کردیم را با اعداد، مقدار دهی می کنیم.

```
FS.set_variable("yesterday_temperature", 50)
FS.set_variable("two_days_ago_temperature", 35)
FS.set_variable("three_days_ago_temperature", 20)
FS.set_variable("humidity", 30)
FS.set_variable("raining", 5000)
FS.set_variable("height", 2300)
```

در نهایت، از متد inference استفاده می کنیم. و با توجه به خروجی فازی که تعریف کرده بودیم (today_temperature)، قانون هایی که تعیین کردیم و مقدار variable ها، سیستم فازی خروجی مورد نظر (دمای هوای امروز) را پیش بینی می کند.

```
yesterday temperature: 50
two days ago temperature: 35
three days ago temperature: 20
humidity: 30
raining: 5000
height: 2300

today temperature is: 40.416303354171006
```