۱-کدهای مربوط به این سوال در فایل Q1.ipynb نوشته شده اند.

الف) ابتدا کتابخانه و ابزار لازم را فراخوانی کردیم.

```
import numpy as np
import math
```

سپس طرح های داده شده برای ذخیره شدن در شبکه ی هاپفیلد را به صورت آرایه ی numpy تعریف کردیم و همه را در یک آرایه ی P قرار دادیم.

```
x1 = np.array([-1, -1, 1, -1, 1, -1, -1, 1])
x2 = np.array([-1, -1, -1, -1, -1, 1, -1, -1])
x3 = np.array([-1, 1, 1, -1, -1, 1, -1, 1])
P = np.array([x1, x2, x3])
```

بعد از آن کلاس Hopfield را طوری تعریف کردیم که تعداد طرح ها و تعداد عضوهای هر پترن و ماتریس وزن با مقداردهی اولیه ی صفر داشته باشد.

```
class Hopfield:
    def __init__(self, N):
        self.N = N
        self.P = 0
        self.W = np.zeros((N, N), dtype=np.int32)
```

در مرحله ی بعد تابع training را برای آموزش شبکه و بدشت آوردن وزن ها با توجه به طرح های ذخیره شده در آن نوشتیم. طوری که طبق فرمول و ویژگی های هاپفیلد کار میکند.

```
def training(Hopfield, P):
      for Xi in P:
          sta = np.vstack(Xi)
          Hopfield.W += np.dot(sta, sta.T)
      np.fill_diagonal(Hopfield.W, 0.0)
      return Hopfield.W
```

این تابع هر پترن را در ترانهاده اش ضرب برداری میکند و حاصل این عملیات برای همه ی پترن ها را جمع میکند در نهایت درایه هایی که طول و عرض برابر دارند برابر با ۰ قرار میدهد.

بعد از آن تابع prediction را به صورت زیر نوشتیم:

```
def prediction(Hopfield, start):
      epochs = 10
      is_stable = False
      result = [(start, math.inf)]
      for i in range(epochs):
          start = np.sign(np.dot(start, Hopfield.W))
          h = np.count_nonzero(np.equal(input, start))
          d = Hopfield.P - h
          result.append((start,d))

          if np.array_equal(result[-1][0], result[-2][0]):
              if i == 0:
                  is_stable=True
              return start, is_stable

      return min(result, key = lambda t: t[1])[0], is_stable
```

با کمک این تابع چک میکنیم که شبکه ی ما میتواند الگوی ورودی را به خاطر بیاورد یا خیر. باید با تعداد مشخصی الگو را در وزن های بدست آمده ضرب کنیم و حاصل را چک کنیم. البته میتوانیم از دقت هم استفاده کنیم اما ما epoch مشخصی را برای پایان دادن به کار در نظر گرفتیم.

وزن های بدست آمده برای الگو ها:

```
HN = Hopfield(8)
W = training(HN, P)
print("Weight : ")
print(W)
```

```
Weight :
[[ 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]]
```

نتایج بدست آمده برای چک کردن پایداری شبکه برای الگوهای وورودی:

```
#x1 = np.array([-1, -1, 1, -1, 1, -1, -1, 1])
out1, res1 = prediction(HN, x1)
print(out1, res1)
# x2 = np.array([-1, -1, -1, -1, -1, 1, -1, -1])
out2, res2 = prediction(HN, x2)
print(out2, res2)
# x3 = np.array([-1, 1, 1, -1, -1, 1, -1, 1])
out3, res3 = prediction(HN, x3)
print(out3, res3)
```

```
[-1 -1  1 -1  1 -1 -1  1] True
[-1 -1 -1 -1 -1  1 -1 -1] True
[-1  1  1 -1 -1  1 -1  1] True
```

همانطور که مشاهده میکنیم این شبکه به ازای هر پترن ورودی به خود آن پترن همگرا شده برای همین در نهایت پایداری آن به ازای هرکدام True شده است.

۱-ب)

الگوهای نویزی را طبق خواسته ی سوال به شبکه دادیم و خروجی به صورت زیر شد:

```
X1n = [1, -1, 1, -1, 1, -1, -1, 1]
out1, res1 = prediction(HN, X1n)
print(out1)
X2n = [1, 1, -1, -1, -1, 1, -1, -1]
out2, res2 = prediction(HN, X2n)
print(out2)
X3n = [1, 1, 1, -1, 1, 1, -1, 1]
out3, res3 = prediction(HN, X3n)
print(out3)
```

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

الگوی نویزی اول یک بیت اختلاف با الگوی اصلی اش داشت و در نهایت به همان همگرا شد اما الگوی اول و دوم نویزی به الگوی اصلیشان همگرا نشدند.

۱-ج) همانطور که میدانیم یک شبکه ی هاپفیلد آموزش یافته نسبت به پترن های ذخیره شد در خود و و پترن های معکوس آنها پایدار است. یعنی در اینجا پترن های

[-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]

حالا باید این ادعا را ثابت کنیم. میتوانیم تمام الگوهای ۸ بیتی ممکن را به شبکه بدهیم و آن پترن هایی که به ازای آنها شبکه پایدار میشود را نمایش دهیم.

```
results = []
pattern_size = 8
for i in range(2**8):
    j = bin(i)[2:].zfill(pattern_size)
    patern_i = [2*int(j[k])-1 for k in range(pattern_size)]
    out,res = prediction(HN, patern_i)
    if res == True:
        print(out)
```

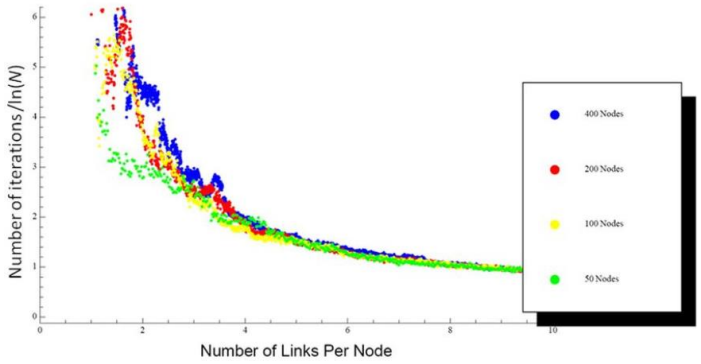
```
[-1 -1 -1 -1 -1  1 -1 -1]
[-1 -1  1 -1  1 -1 -1  1]
[-1  1  1 -1 -1  1 -1  1]
[ 1 -1 -1  1  1 -1  1 -1]
[ 1  1 -1  1 -1  1  1 -1]
[ 1  1  1  1  1 -1  1  1]
```

دقیقا همان پترن ها پیدا شدند.

۱-ث) نکته ای که در مورد شبکه ی هاپفیلد وجود دارد این است که این شبکه حتما خروجی دارد و حتما به یک پترن مشابه همگرا میشود حالا این پترن ممکن است دقیقا از پترن های از قبل ذخیره شده در شبکه نباشد و معکوس یا ترکیبی از آن ها باشد یا حتی به اشتباه تشخیص داده شود.

در مثالی که در اسلاید مربوط به این مبحث دیدیم اگر هنگام محاسبه مقادیر را به صورت سنکرون آپدیت کنیم ممکن است شبکه در یک حلقه گیر کند. مثلا در مثال اسلاید شبکه به ازای یکی از پترن ها بین دو مقدار تکرار میشد و به یک پترن همگرا نمیشد اما طبق گفته ی استاد اگر به صورت آسنکرون حاصل ضرب وزن در پترن را بدست آوریم (یعنی وقتی درایه ی اول سزر اول را حساب کردیم قبل از تمام شدن سطر اول درایه ی اول سطر دوم را حساب کنیم) این مشکل پیش نخواهد آمد و شبکه همگرا خواهد شد.

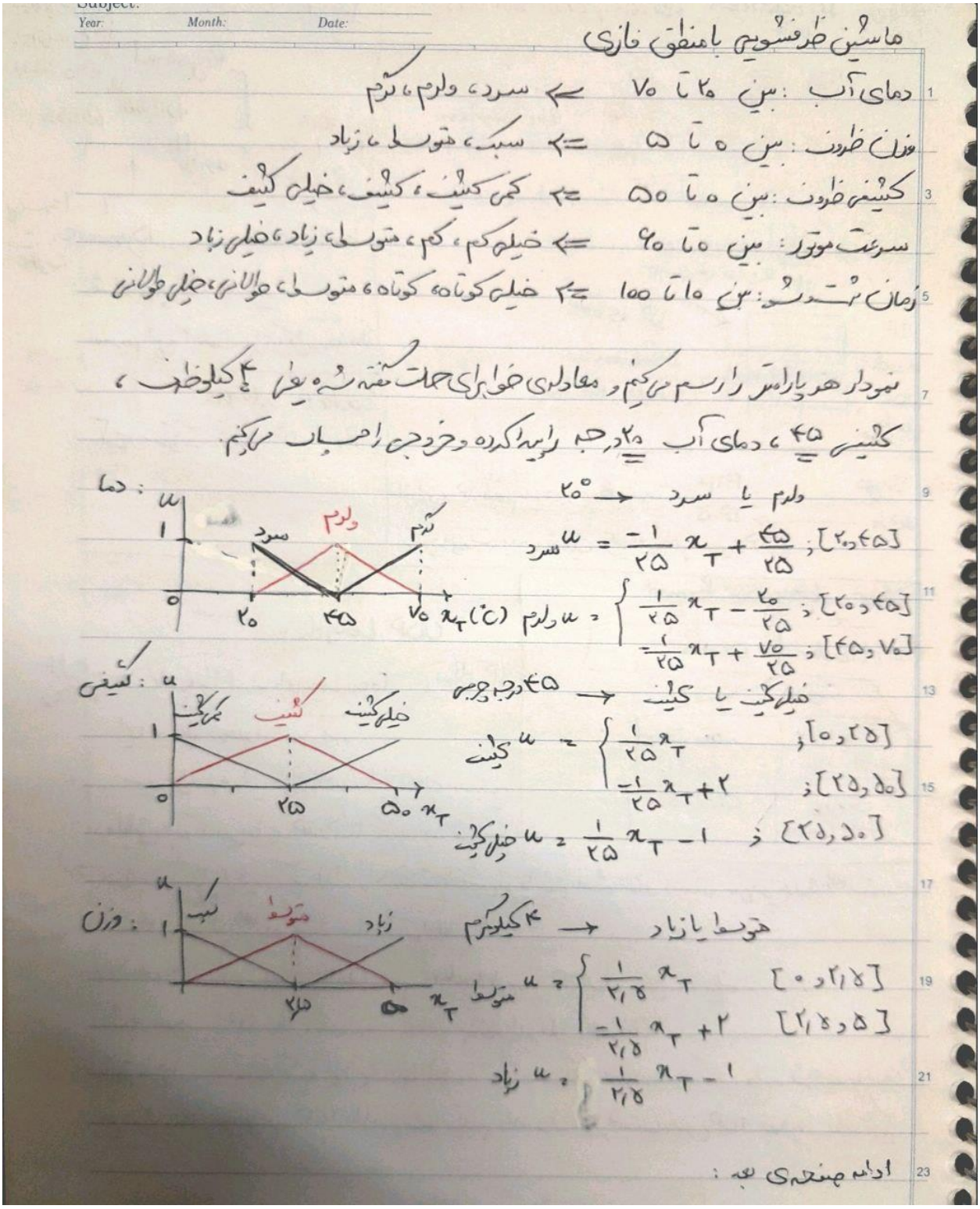
تعداد تکرارهای شبکه هاپفیلد برای دستیابی به همگرایی با جذب کننده تعیین شده از یک حالت شروع دلخواه به عنوان تابعی از تعداد پیوندها در هر گره در شبکه مورد نیاز است.




همانطور که در شکل مشخص است هرچه تعداد اتصال بین نود ها در یک شبکه (تعداد همسایه های هر نود) بیشتر باشد تعداد تکرار کمتری لازم است. مثلا در همین سوال چون ما آرایه ای از بیت ها داشتیم و پترن های ورودی یک بعدی بودند حداکثر هر نود دو همسایه داشت و ما با ۵ تکرار خواسته های سوال را بدست آوردیم. اگر پترن ها دو بعدی بودند هر نود ۴ همسایه میداشت و با تعداد تکرار کمتر جواب های خوبی بدست می آمد.

منبع عکس: https://www.researchgate.net/figure/Number-of-iterations-of-the-Hopfield-net-required-to-achieve-convergence-to-the_fig7_49726300

-۲


ماشین ظرفشویی با منطق فازی
دمای آب: بین ۲۰ تا ۷۰ ⇐ سرد، ولرم، گرم
وزن ظروف: بین ۰ تا ۵ ⇐ سبک، متوسط، زیاد
کثیفی ظروف: بین ۰ تا ۵۰ ⇐ کمی کثیف، کثیف، خیلی کثیف
سرعت موتور: بین ۰ تا ۹۰ ⇐ خیلی کم، کم، متوسط، زیاد، خیلی زیاد
زمان شستشو: بین ۱۰ تا ۱۰۰ ⇐ خیلی کوتاه، کوتاه، متوسط، طولانی، خیلی طولانی
نمودار هر پارامتر را رسم می‌کنیم و معادله‌ی خواسته‌ی حالت گفته شده یعنی ۴ کیلو ظرف، کثیفی ۴۵، دمای آب ۲۵ درجه را پیدا کرده و خروجی را حساب می‌کنیم.
دما:
سرد
ولرم
گرم
۲۰
۴۵
۷۰
$x_T$(°C)
$\mu_{\text{سرد}} = \frac{-1}{25} x_T + \frac{45}{25}$ ; [۲۰, ۴۵]
$\mu_{\text{ولرم}} = \begin{cases} \frac{1}{25} x_T - \frac{20}{25} & ; [20, 45] \\ \frac{-1}{25} x_T + \frac{70}{25} & ; [45, 70] \end{cases}$
۲۵ → سرد یا ولرم
کثیفی:
کمی کثیف
کثیف
خیلی کثیف
۲۵
۵۰
$x_T$
۴۵ → خیلی کثیف یا کثیف
$\mu_{\text{کثیف}} = \begin{cases} \frac{1}{25} x_T & ; [0, 25] \\ \frac{-1}{25} x_T + 2 & ; [25, 50] \end{cases}$
$\mu_{\text{خیلی کثیف}} = \frac{1}{25} x_T - 1$ ; [۲۵, ۵۰]
وزن:
سبک
متوسط
زیاد
۲/۵
۵
$x_T$
۴ → متوسط یا زیاد
$\mu_{\text{متوسط}} = \begin{cases} \frac{1}{2.5} x_T & [0, 2.5] \\ \frac{-1}{2.5} x_T + 2 & [2.5, 5] \end{cases}$
$\mu_{\text{زیاد}} = \frac{1}{2.5} x_T - 1$
ادامه صفحه‌ی بعد:

Subject:
Year: Month: Date:

استفاده از روش ممدانی (max(min)

دما: { سرد $u(20^\circ) = \frac{-20+45}{25} = 1$ ، ولرم $u(25^\circ) = 0$

وزن: { کبیف $u^{(45)} = \frac{-45}{25} + \frac{50}{28} = \frac{1}{5} = 0.2$ ، خیلی کبیف $u^{(45)} = \frac{45}{25} - \frac{28}{28} = \frac{20}{28} = 0.8$

سرعت: { متوسط $u(4) = \frac{-4}{2.5} + 2 = \frac{1}{2.5} = 0.4$ ، زیاد $u(4) = \frac{4}{2.5} - 1 = 0.6$

* قوانین: ۱) اگر دما خیلی کبیف، وزن زیاد و آب ولرم ⇒ سرعت موتور خیلی کم، زمان شستشو خیلی طولانی
0 0.6 0.8

۲) اگر دما خیلی کبیف، وزن زیاد و آب سرد ⇒ سرعت خیلی زیاد، زمان خیلی طولانی
1 0.6 0.8

۱) $\min(0, 0.6, 0.8) = 0$ ⟶ $\max(0, 0.6) = 0.6$

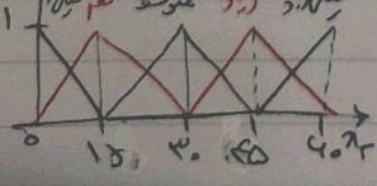
۲) $\min(1, 0.6, 0.8) = 0.6$

نمودار سرعت و زمان را هم رسم می‌کنیم

دیفازی 0.6 درجه عضویت را برای زمان خیلی زیاد و سرعت خیلی کم و خیلی زیاد حساب می‌کنیم

چون جواب max حاصل قانون دوم شد سرعت خیلی زیاد که برای قانون ۲ ام حساب شده

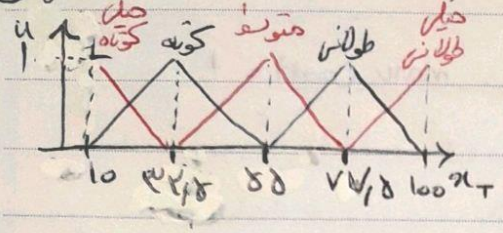
سرعت موتور:




$\mu_{\text{خیلی زیاد}} = \frac{+1}{15}x_2 - 4$

$\mu_{\text{خیلی کم}} = \frac{-1}{15}x_2 + 1$

زمان:




$$\mu_{\text{خیلی طولانی}} = \frac{x_T - 77.5}{22.5}$$

برحسب آمدن زمان در سرعت برازای مقادیر ورودی داده شده:

سرعت:
$$\frac{x_T - 40}{15} = 0.4 \Rightarrow \frac{x_T}{15} = 3.4 \rightarrow x_T = 54$$

زمان:
$$\frac{x_T}{22.5} - \frac{77.5}{22.5} = 0.4 \Rightarrow x_T - 77.5 = 13.5$$
$$\Rightarrow x_T = 91$$

باتوجه به قوانین باشرایط گفته شده سرعت موتور بایه ۵۴ دور بر دقیقه
و زمان گشت سگ 41 دقیقه باشه

۳- منبع: simpful · PyPI

کدهای مربوط به این سوال در فایل Q3.ipynb نوشته شده اند.

ابتدا کتابخانه ی موردنیاز را نصب و فراخوانی کردیم:

```
!pip install simpful
import simpful as sf
```

سپس سیستم فازی کنترلی را که طبق سوال ۳ پارامتر دمای دیرور و دو روز گذاشته و ۳ روز گذشته از جنس دما برحسب سلسیوس دارد و یک پارامتر رطوبت بر حسب درصد و یک پارامتر بارش باران برحسب میلیمتر و یک پارامتر ارتفاع شهر برحسب متر دارد. همچنین یک پارامتر کنترل شونده ی دمای امروز دارد که برحسب پارامترهای قبلی کنترل میشود. حدود هر متغیر را بر اساس دانش عمومی در نظر گرفتیم مثلا بلند ترین شهر جهان۵۱۰۰ متر ارتفاع دارد که برای رند شدن و تقسیم بهتر بین ۵ محدوده ۵۵۰۰ متر را حدبالای ارتفاع شهر درنظر گرفتیم.

```
FS = sf.FuzzySystem()

#برحسب سیلسیوس
temp = sf.AutoTriangle(4, terms=['very_cold', 'cold', 'hot', 'very_hot'], universe_of_discourse=[-5, 100])

FS.add_linguistic_variable("1_day_ago_tmp", temp)
FS.add_linguistic_variable("2_days_ago_tmp", temp)
FS.add_linguistic_variable("3_days_ago_tmp", temp)
FS.add_linguistic_variable("today_tmp", temp)

#برحسب درصد
hmdt = sf.AutoTriangle(5, terms=['very_low', 'low', 'normal', 'high', 'very_high'], universe_of_discourse=[0, 100])
FS.add_linguistic_variable("humidity", hmdt)
#برحسب میلیمتر
rain_fall = sf.AutoTriangle(5, terms=['very_low', 'low', 'normal', 'much', 'very_much'], universe_of_discourse=[0, 2500])
FS.add_linguistic_variable("rain_fall", rain_fall)
#برحسب متر
height = sf.AutoTriangle(5, terms=['very_low', 'low', 'normal', 'high', 'very_high'], universe_of_discourse=[0, 5500])
FS.add_linguistic_variable("height", height)
```

۱۲ قانون زیر را به طور فرضی به سیستم کنترلی دادیم.

```
RULE1 = "IF (1_day_ago_tmp IS cold) AND (rain_fall IS much) THEN (today_tmp IS cold)"
RULE2 = "IF (1_day_ago_tmp IS very_cold) AND (rain_fall IS very_much) THEN (today_tmp IS very_cold)"
RULE3 = "IF (1_day_ago_tmp IS hot) OR (3_days_ago_tmp IS hot) THEN (today_tmp IS hot)"
RULE4 = "IF (1_day_ago_tmp IS very_hot) AND (2_days_ago_tmp IS very_hot) THEN (today_tmp IS very_hot)"
RULE5 = "IF (1_day_ago_tmp IS cold) OR (rain_fall IS very_low) THEN (today_tmp IS hot)"
RULE6 = "IF (1_day_ago_tmp IS very_cold) OR (3_days_ago_tmp IS very_cold) THEN (today_tmp IS very_cold)"
RULE7 = "IF (3_days_ago_tmp IS cold) OR (2_days_ago_tmp IS cold) THEN (today_tmp IS cold)"
RULE8 = "IF (3_days_ago_tmp IS very_hot) AND (2_days_ago_tmp IS hot) AND (1_day_ago_tmp IS cold) THEN (today_tmp IS very_cold)"
RULE9 = "IF (humidity IS high) AND (rain_fall IS normal) THEN (today_tmp IS hot)"
RULE10 = "IF (height IS very_low) AND (rain_fall IS very_much) THEN (today_tmp IS very_cold)"
RULE11 = "IF (height IS normal)  OR (rain_fall IS very_low) THEN (today_tmp IS hot)"
RULE12 = "IF (height IS high) AND (rain_fall IS normal) THEN (today_tmp IS hot)"

FS.add_rules([RULE1, RULE2, RULE3, RULE4, RULE5, RULE6, RULE7, RULE8, RULE9, RULE10, RULE11, RULE12], verbose=True)
```

در نهایت برای آزمون سیستم فازی، مقادیر زیر را به سیستم فازی دادیم تا دمای امروز را پیشبینی نماید.

```
FS.set_variable("1_day_ago_tmp", -1)
FS.set_variable("2_days_ago_tmp", 6)
FS.set_variable("3_days_ago_tmp", 14)
FS.set_variable("humidity", 35)
FS.set_variable("rain_fall", 750)
FS.set_variable("height", 20)
```

```
result = FS.inference()
print(result)
```

```
{'today_tmp': 28.828796721262986}
```