Session #08

CMSC 409: Artificial Intelligence

http://

Virginia Commonwealth University,
Fall 2023,
Dr. Milos Manic
(mmanic@vcu. edu)

1

CMSC 409: Artificial Intelligence

Session # 08

Topics for today

- Announcements
- Previous session review
- Perceptron learning rule
  - *Perceptron training*
  - *Graphical illustration*

2

CMSC 409: Artificial Intelligence

Announcements

Session # 08

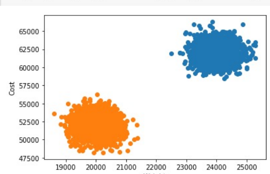
- Canvas
  - *New slides posted*
- Office hours zoom
  - *Zoom disconnects me after 45 mins of inactivity. Feel free to chat me via zoom if that happens and I will reconnect (zoom chat welcome outside of office hours as well)!*
- Project #2
  - *Deadline Oct. 3 (noon)*
- Paper (optional)
  - *The 2nd draft due Oct. 10 (noon)*
  - *Literature review and updated problem description (check out the class paper instructions for the 2nd draft)*
- Subject line and signature
  - *Please use [CMSC 409] Last_Name Question*

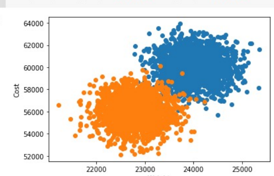
3

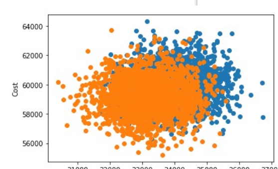
Project 1 data sets

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import kde
import pandas as pd

DATA_LOCATION = "AGroup.csv"
FA = "X"
SA = "Y"
TA = "Z"
datas = pd.read_csv(DATA_LOCATION)
x = datas[FA]
y = datas[SA]
z = datas[TA]
plt.xlabel('Weight')
plt.ylabel('Cost')
data = pd.DataFrame({"X Value": x, "Y Value": y, "Category": z})
groups = data.groupby("Category")

for name, group in groups:
    plt.plot(group["X Value"], group["Y Value"], marker="o", linestyle="", label=name)
```

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import kde
import pandas as pd

DATA_LOCATION = "BGroup.csv"
FA = "X"
SA = "Y"
TA = "Z"
datas = pd.read_csv(DATA_LOCATION)
x = datas[FA]
y = datas[SA]
z = datas[TA]
plt.xlabel('Weight')
plt.ylabel('Cost')
data = pd.DataFrame({"X Value": x, "Y Value": y, "Category": z})
groups = data.groupby("Category")

for name, group in groups:
    plt.plot(group["X Value"], group["Y Value"], marker="o", linestyle="", la
```





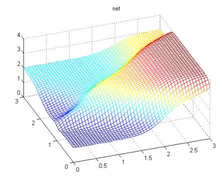


4

Perceptron learning rule

- *Perceptron training*
- *Learning example*
- *Graphical illustration*
  - *Learning constant & hard activation function*
- *Learning example in Perl*
- *Hard vs. soft activation function*
  - *Soft activation function*

Perceptron training (supervised training)

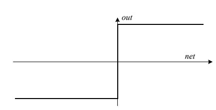


$$\Delta \mathbf{w}_i = \alpha\, \delta\, \mathbf{x}$$

Perceptron learning rule:

$$\delta = d - o$$

$$\Delta \mathbf{w}_i = \alpha\, \mathbf{x}\left(d - \operatorname{sign}(net)\right)$$



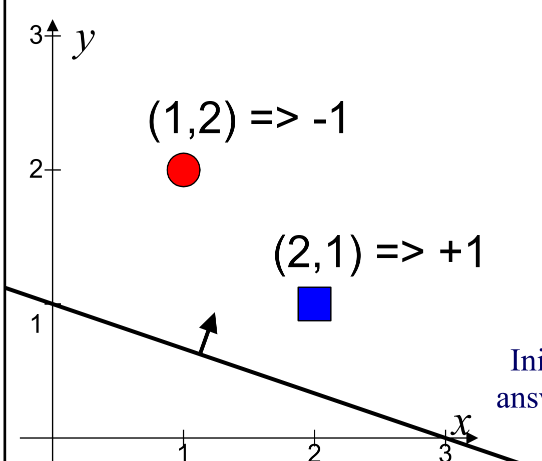
Assuming bipolar neurons:
output = ±1, and $\Delta \mathbf{w}_i = \pm\alpha\, \mathbf{x}\, 2$

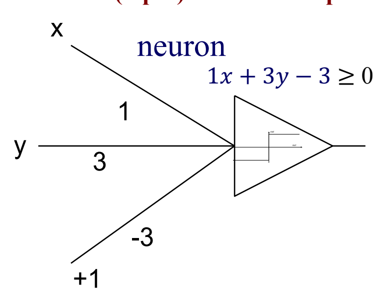
Perceptron learning rule

- Perceptron training
- *Learning example*
- Graphical illustration
  - Learning constant & hard activation function
- Learning example in Perl
- Hard vs. soft activation function
  - Soft activation function

Simple example of training one neuron
(perceptron learning rule)

Training patterns in form of:
(input)=>desired output



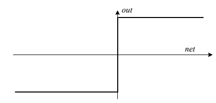


Initial weight set produces wrong answers: both patterns belong to +1 category!

And, as always….start with definition of net & out!

Simple example of training one neuron (cont.)
(perceptron learning rule)

				Desired output
Weights:	1	3	-3	
Pattern 1:	1	2	+1	-1
Pattern 2:	2	1	+1	+1



$$net = \sum_{i=1}^{n} w_i x_i$$ Actual output

for pattern 1: *net* = 1*1+3*2-3*1= 4 => +1

for pattern 2: *net* = 1*2+3*1-3*1= 2 => +1

Simple example of training one neuron (cont.)

Assuming learning constant: $\alpha = 0.3$

weights: $\mathbf{w} = [1 \quad 3 \quad -3]$

pattern 1: $\mathbf{x} = [1 \quad 2 \quad 1]$

$$net = \sum_{i=1}^{n} w_i x_i \qquad \Delta\mathbf{w} = \alpha\, \mathbf{x}(d - o)$$

$$net = 1 \cdot 1 + 2 \cdot 3 + 1 \cdot (-3) = 4 \quad => +1$$

net = 4 => out =+1

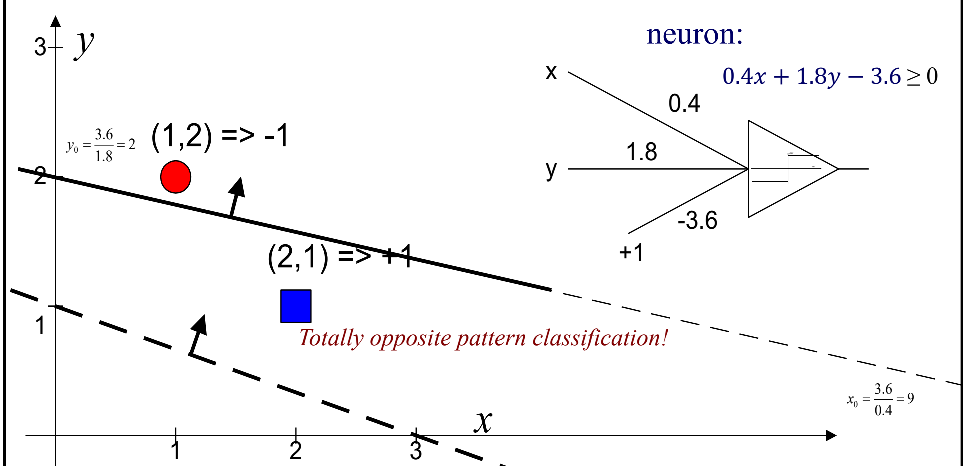
$$\Delta\mathbf{w} = 0.3\, \mathbf{x}(-1 - 1) = -0.6\mathbf{x}$$

$$\Delta\mathbf{w} = [-0.6 \quad -1.2 \quad -0.6]$$

$$\mathbf{w} = [0.4 \quad 1.8 \quad -3.6]$$ modified weights

Simple example of training one neuron (cont.)

After applying the 1st pattern for the 1st time: $\mathbf{w} = [0.4 \quad 1.8 \quad -3.6]$



11

Simple example of training one neuron

Applying the 2nd pattern for the 1st time: $\mathbf{w} = [0.4 \quad 1.8 \quad -3.6]$

weights: $\mathbf{w} = [0.4 \quad 1.8 \quad -3.6]$

pattern 2: $\mathbf{x} = [2 \quad 1 \quad 1]$

$$net = \sum_{i=1}^{n} w_i x_i \qquad \Delta\mathbf{w} = \alpha\, \mathbf{x}(d - o)$$

$$net = 2 \cdot 0.4 + 1 \cdot 1.8 - 1 \cdot 3.6 = -1 \quad => \quad -1$$

net = -1 => out =-1

$$\Delta\mathbf{w} = 0.3\,\mathbf{x}(+1 - (-1)) = 0.6\mathbf{x}$$

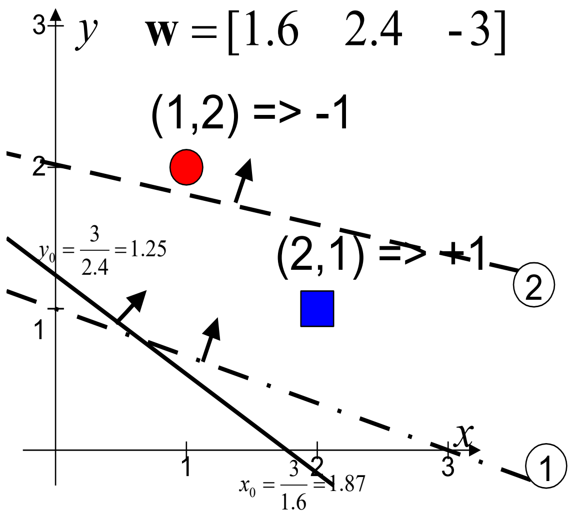
$$\Delta\mathbf{w} = [1.2 \quad 0.6 \quad 0.6]$$

$\mathbf{w} = [1.6 \quad 2.4 \quad -3.0]$ modified weights

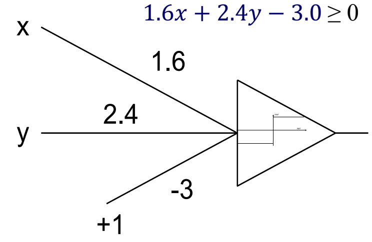
12

Simple example of training one neuron

After applying the 2nd pattern for the 1st time: $\mathbf{w} = [1.6 \quad 2.4 \quad -3.0]$



neuron:
$1.6x + 2.4y - 3.0 \geq 0$



13

Simple example of training one neuron

Applying the 1st pattern for the 2nd time: $\mathbf{w} = [1.6 \quad 2.4 \quad -3]$

weights: $\mathbf{w} = [1.6 \quad 2.4 \quad -3]$

pattern 1: $\mathbf{x} = [1 \quad 2 \quad 1]$

$$net = \sum_{i=1}^{n} w_i x_i \qquad \Delta\mathbf{w} = \alpha\, \mathbf{x}(d - o)$$

$$net = 1 \cdot 1.6 + 2 \cdot 2.4 - 1 \cdot 3 = 3.4 \quad => \quad +1$$

net = 3.4 => out =+1

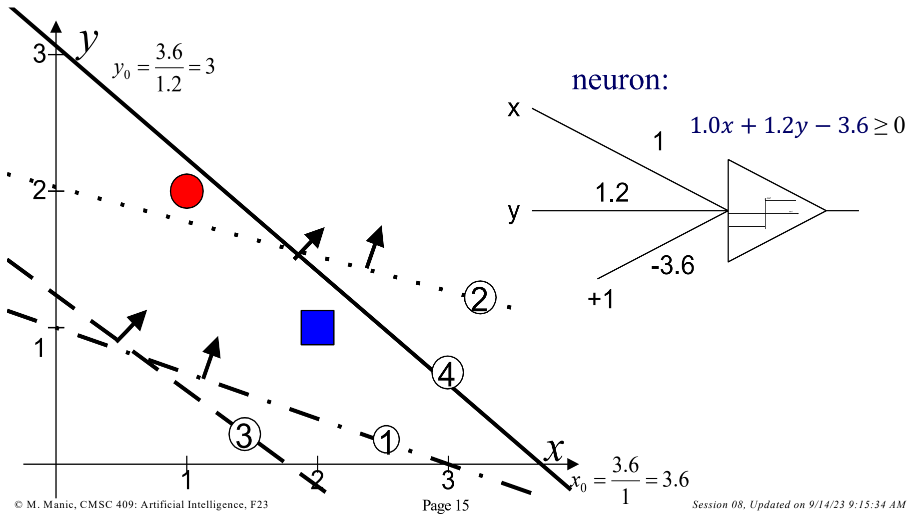
$$\Delta\mathbf{w} = 0.3\,\mathbf{x}(-1 - (+1)) = -0.6\mathbf{x}$$
$$\Delta\mathbf{w} = [-0.6 \quad -1.2 \quad -0.6]$$

$\mathbf{w} = [1 \quad 1.2 \quad -3.6]$ modified weights

14

Simple example of training one neuron

After applying the 1st pattern for the 2nd time: $\mathbf{w} = [1.0 \quad 1.2 \quad -3.6]$



© M. Manic, CMSC 409: Artificial Intelligence, F23 Page 15 *Session 08, Updated on 9/14/23 9:15:34 AM*

15

Simple example of training one neuron

Applying the 2nd pattern for the 2nd time: $\mathbf{w} = [1 \quad 1.2 \quad -3.6]$

weights: $\mathbf{w} = [1 \quad 1.2 \quad -3.6]$
pattern 2: $\mathbf{x} = [2 \quad 1 \quad 1]$

$$net = \sum_{i=1}^{n} w_i x_i \qquad \Delta\mathbf{w} = \alpha\, \mathbf{x}(d - o)$$

$$net = 2 \cdot 1 + 1 \cdot 1.2 - 1 \cdot 3.6 = -0.4 \quad => \quad -1$$

net = -0.4 => out =-1

$$\Delta\mathbf{w} = 0.3\, \mathbf{x}(+1 - (-1)) = 0.6\mathbf{x}$$
$$\Delta\mathbf{w} = [1.2 \quad 0.6 \quad 0.6]$$

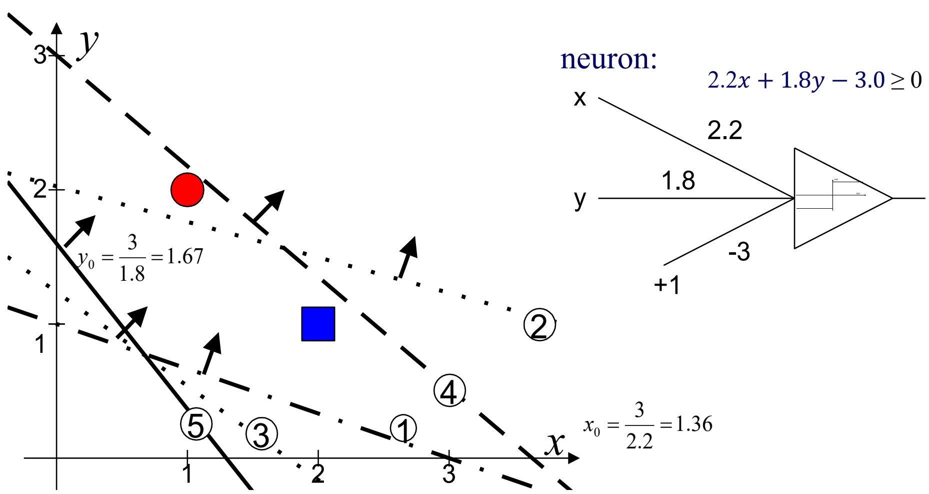
$\mathbf{w} = [2.2 \quad 1.8 \quad -3.0]$ modified weights

© M. Manic, CMSC 409: Artificial Intelligence, F23 Page 16 *Session 08, Updated on 9/14/23 9:15:34 AM*

16

Simple example of training one neuron

After applying the 2nd pattern for the 2nd time: $\mathbf{w} = [2.2 \quad 1.8 \quad -3.0]$



Simple example of training one neuron

Applying the 1st pattern for the 3rd time: $\mathbf{w} = [2.2 \quad 1.8 \quad -3.0]$

weights: $\mathbf{w} = [2.2 \quad 1.8 \quad -3.0]$
pattern 1: $\mathbf{x} = [1 \quad 2 \quad 1]$

$$net = \sum_{i=1}^{n} w_i x_i \qquad \Delta\mathbf{w} = \alpha\, \mathbf{x}(d - o)$$

$$net = 1 \cdot 2.2 + 2 \cdot 1.8 - 1 \cdot 3 = 2.8 \quad => \quad +1$$

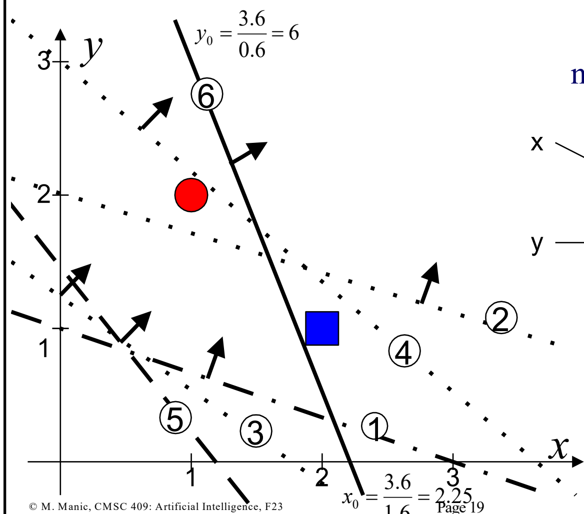
net = 2.8 => out =+1

$$\Delta\mathbf{w} = 0.3\, \mathbf{x}(-1 - (+1)) = -0.6\mathbf{x}$$
$$\Delta\mathbf{w} = [-0.6 \quad -1.2 \quad -0.6]$$

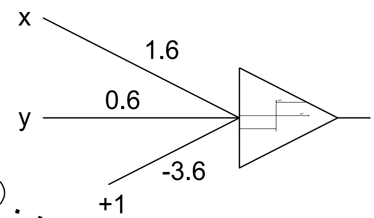
$$\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$$ modified weights

Simple example of training one neuron

After applying the 1st pattern for the 3rd time: $\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$



neuron: $1.6x + 0.6y - 3.6 \geq 0$



19

Simple example of training one neuron

Applying the 2nd pattern for the 3rd time: $\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$

weights: $\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$
pattern 2: $\mathbf{x} = [2 \quad 1 \quad 1]$

$$net = \sum_{i=1}^{n} w_i x_i \qquad \Delta\mathbf{w} = \alpha\, \mathbf{x}(d - o)$$

$$net = 2 \cdot 1.6 + 1 \cdot 0.6 - 1 \cdot 3.6 = 0.2 \quad => \quad +1$$

net = 0.2 => out =+1

$$\Delta\mathbf{w} = 0.3\, \mathbf{x}(+1 - (+1)) = 0 \cdot \mathbf{x} = 0$$
$$\Delta\mathbf{w} = [0 \quad 0 \quad 0]$$

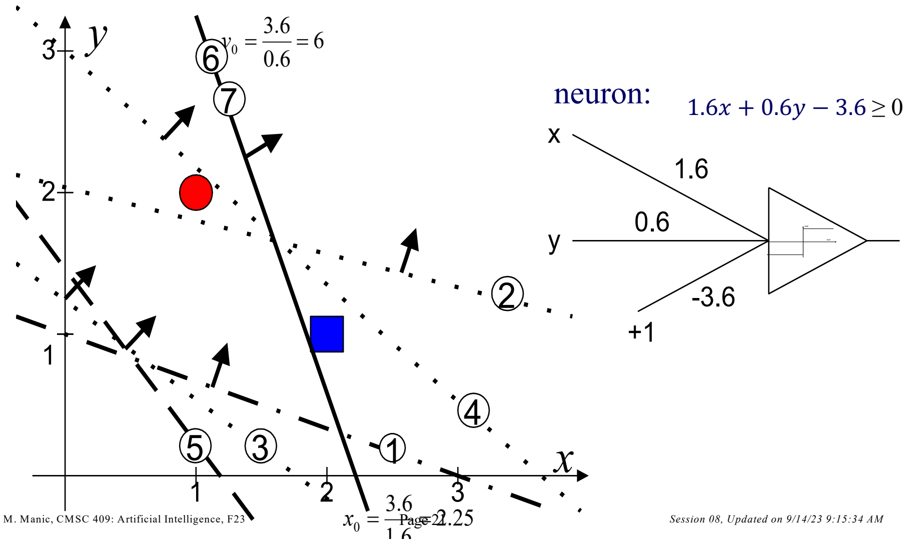
(weights haven't changed, d=o!)

$\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$ modified weights

20

Simple example of training one neuron

After applying the 2nd pattern for the 3rd time: $\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$



21

Simple example of training one neuron

Applying the 1st pattern for the 4th time: $\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$

weights: $\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$
pattern 1: $\mathbf{x} = [1 \quad 2 \quad 1]$

$$net = \sum_{i=1}^{n} w_i x_i \qquad \Delta\mathbf{w} = \alpha\, \mathbf{x}(d - o)$$

$$net = 1 \cdot 1.6 + 2 \cdot 0.6 - 1 \cdot 3.6 = -0.8 \quad => \quad -1$$

net = -0.8 => out =-1

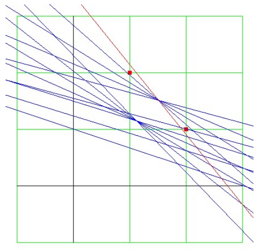
$$\Delta\mathbf{w} = 0.3\, \mathbf{x}(-1 - (-1)) = 0 \cdot \mathbf{x} = 0$$
$$\Delta\mathbf{w} = [0 \quad 0 \quad 0]$$

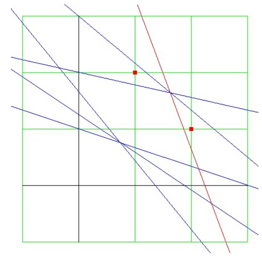
(weights haven't changed, d=o!)

$\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$ modified weights

22

This was hard threshold function…

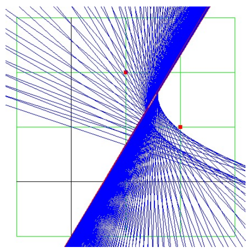


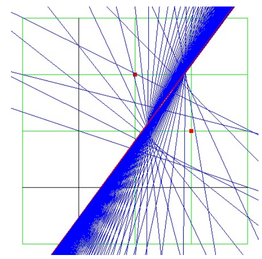


learning $\alpha = 0.1$

… how about some other?

learning $\alpha = 0.3$





23

Unsupervised vs. supervised learning

$$\boxed{\Delta \mathbf{w}_i = \alpha\, \delta\, \mathbf{x}}$$

$$\boxed{\mathbf{w}_{k+1} = \mathbf{w}_k + \Delta \mathbf{w}}$$

Hebb Rule *(unsupervised):* $\delta = o$

Correlation Rule *(supervised):* $\delta = d$

Perceptron Fixed Rule: $\delta = d - o$

24

Things to remember…

- **Neuron "representation"**
  - *We have represented the same neuron in 3 ways: inequality, drawn decision line, or drawn neuron*
- **Learning = adjusting weights!**
  - *Through learning, we train i.e adjust neuron parameters (weights, including bias/threshold)*
  - *Learning (training) is "driven" by a learning signal δ*
- **Activation function matters!**
  - *Hard activation function*
    - *for non-overlapping data sets may be sufficient, but may not be optimal*
    - *once the error is zero (δ in case of perceptron), Δw becomes zero, nothing gets learned any more – solution* ***may not be optimal****!*
    - *i.e, if error* $\to 0$*, then* $\delta \to 0$, consequently $\Delta w \to 0$
  - *Linear or soft activation function*
    - *error likely never becomes zero, i.e. you can continue optimizing solution until stopping criterion is met*
- **Training…**
  - *this was "incremental" training, we added new knowledge based on every pattern in every iteration…one should be aware of pros and cons…*