Session #17

CMSC 409:
Artificial Intelligence

http://www.people.vcu.edu/~mmanic/

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

1

CMSC 409: Artificial Intelligence

Session # 17

Topics for today

- Announcements
- Midterm exam preparation

2

CMSC 409: Artificial Intelligence
Session # 17

Topics for today

- Announcements
- Previous session review
- Midterm exam preparation
- Agent-Environment Interface & RL
  - *Agents*
  - *Returns*
  - *Markov Property*
  - *Markov Decision Processes (MDP)*
  - *Example (recycling robot)*
  - *MDP graph*
- Q Learning, Smart Cab Problem

CMSC 409: Artificial Intelligence
Announcements Session # 17

- IMPORTANT:
  - *Course materials (slides, assignments) are copyrighted by instructor & VCU. Sharing/posting/chatGPT/similar is copyright infringement and is strictly prohibited. Such must be immediately reported.*
- 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 #3
  - *Deadline Oct. 26; Review a week from the deadline.*
- Midterm exam (in-class)
  - *Oct. 19 (Thu); prep examples are posted*
- Paper (optional)
  - *The 3rd draft due Nov. 2 (noon)*

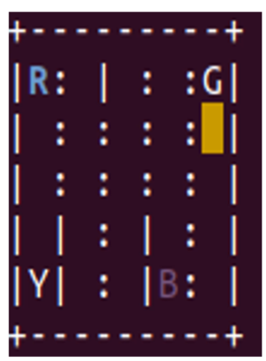
  In addition to previous draft, it should contain a technique (or selection thereof), you plan on using to solve the selected problem (check out the class paper instructions for the 3rd draft)
- Subject line and signature
  - *Please use [CMSC 409] Last_Name Question*

Project 3
Q Learning, Smart Cab Problem

Reinforcement Q-Learning from Scratch in Python with OpenAI Gym:
https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

5

Q-Learning



Smart Cab
- Drops off the passenger to the right location.
- Performs route optimization to minimize the ride time.
- Observe traffic rules and prioritize passenger's safety.

Rewards
- 8 maximum number of time steps
- (+20) for a successful drop off
- (-10) for illegal actions
- (-1) for each time step it takes to get to the destination

Q Learning Approach
- No knowledge about the environment
- Table to represent the value function [500,6]

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha(R + \lambda\, maxQ(s+1,a) - Q_t(s,a))$$

6

Q-Learning

Key parameters

- α (Learning Rate) ▸ Bigger makes convergence faster but may cause convergence difficulties
- λ (Discount Factor) ▸ Close to 1 to take into account future rewards
- ε_{decay}(Drop rate of randomness) ▸ Close to 1 to make sure to explore enough

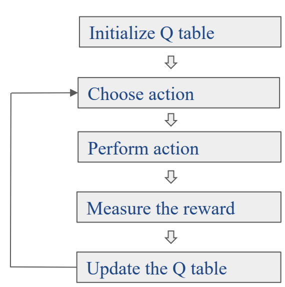
Exploration vs exploitation

- Exploration → Agent will take random decisions, this is useful to learn about the environment.
- Exploitation → Agent will take actions based on what he already knows.(From the Q table)

7

Q-Learning

- Q Learning



8

Q-Learning

- Initialization

Initialize the Q table with zeros

```
q_table = np.zeros([env.observation_space.n, env.action_space.n])
```

Parameters chosen for taxi exercise

```
# Hyper params:

total_ep = 15000
total_test_ep = 1000
max_steps = 100

lr = 0.01
gamma = 0.99

# Exploration Params:

epsilon = 0.5
max_epsilon = 1.0
min_epsilon = 0.01
decay_rate = 0.01
```

9

Q-Learning

- Train Algorithm

```
# Implementing the Q Learning Algorithm:

for episode in range(total_ep):

  # Reset Environment:
  state = env.reset()
  step = 0
  done = False

  for step in range(max_steps):

    # Choose an action a in the current world state(s) (step 3)
    # First we randomize a number
    exp_exp_tradeoff = random.uniform(0, 1)

    # If this number > greater than epsilon --> exploitation (taking the biggest q value for the current state):
    if exp_exp_tradeoff > epsilon:
      action = np.argmax(q_table[state, :])

    # Else, doing random choice:
    else:
      action = env.action_space.sample()

    # Take the action (a) and observe the outcome state (s') and the reward (r)
    new_state, reward, done, info = env.step(action)

    # Update Q(s,a):= Q(s,a) + lr [R(s,a) + gamma * max Q(s',a') - Q(s,a)]
    q_table[state, action] = q_table[state, action] + lr * (reward + gamma *
                                np.max(q_table[new_state, :]) - q_table[state, action])

    # Our new state:
    state = new_state

    # If done True, finish the episode:
    if done == True:
      break

  # Increment number of episodes:
  episode += 1


print("Training finished.\n")
```

- Pick a random value action or exploit computed Q values
- Execute the chosen action -> obtain next state and reward
- Calculate the max Q value and update the Q table

10

Q-Learning

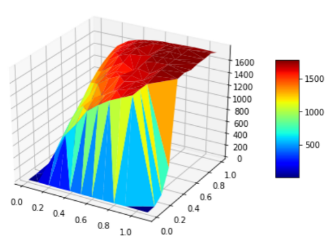
- Evaluate Algorithm

```python
"""Evaluate agent's performance after Q-learning"""
# Using Q Table:

env.reset()
rewards = []
tot_penalties = []

for episode in range(total_test_ep):
  state = env.reset()
  step = 0
  done = False
  total_rewards = 0
  penalties = 0
  print('========================')
  print('EPISODE: ', episode)

  for step in range(max_steps):

    env.render()

    # Take the action based on the Q Table:
    action = np.argmax(q_table[state, :])

    new_state, reward, done, info = env.step(action)

    if reward == -10:
        penalties += 1

    total_rewards += reward

    # If episode finishes:
    if done:
      rewards.append(total_rewards)
      tot_penalties.append(penalties)
      print('Reward: ', total_rewards)
      print('Penalty: ', penalties)
      break

    state = new_state

env.close()
print('Reward Over Time: {}'.format(sum(rewards)/total_test_ep))
print('Penalty Over Time: {}'.format(sum(tot_penalties)/total_test_ep))
```

- Agent switch from exploration to exploitation
- Next action is selected using the best Q- value
- Penalty is updated for (-10) reward
- If no penalty -> Smart cab agent performed correct pick up/drop off actions with 1000 passengers

Q-Learning

- Parameters study
  - Different results for α, λ, epsilon between [0,1]
  - High Learning Rate could cause convergence problems with more complex problems or with more noise
  - No universal rules, parameter study is recommended



Q-Learning

- Results for chosen parameters
  - Around 1,000 episodes to converge
  - Takes longer to converge than with bigger Learning Rate but ensures stability

```
# Hyper params:

total_ep = 15000
total_test_ep = 1000
max_steps = 100

lr = 0.01
gamma = 0.99

# Exploration Params:

epsilon = 0.5
max_epsilon = 1.0
min_epsilon = 0.01
decay_rate = 0.01
```

13

Clustering

14

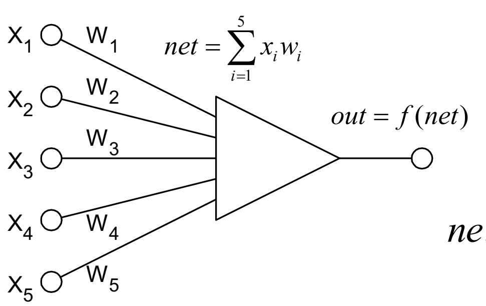
Clustering as competitive learning

- *Kohonen Networks (WTA)*
- *Derivation*
- *Steps*
- *Example*
- *Problems & remedies*

Kohonen, T. (1988) Self-Organization and Associative Memory, 2nd Ed. New York, Springer-Verlag.
Kohonen, T. (1982) Self-organized formation of topologically correct feature maps. Biological Cybernetics, 43:59-69.
Kohonen, T. (1990) The Self-Organizing Map. Proceedings of the IEEE, 78:1464-1480.
Kohonen, T. (1995) Self-Organizing Maps. Springer, Berlin.
Kohonen, T., Oja, E., Simula, O., Visa, A., and Kangas, J. (1996). Engineering applications of the self-organizing map. Proceedings of the IEEE, 84:1358-1384.

15

Kohonen Network



$$net = \sum_{i=1}^{5} x_i w_i = \mathbf{XW}^T$$

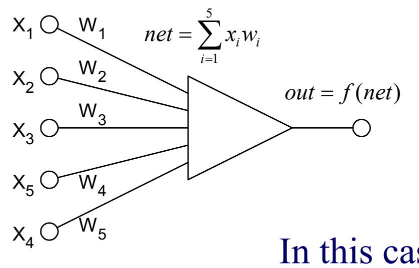
Remember:

*If inputs are binaries, for example **X**=[1, -1, 1, -1, -1] then the maximum net value is when weights are identical to the input pattern:*

***W**=[1, -1, 1, -1, -1]*

16

Kohonen Network



Also….

In this case $net = 5$.

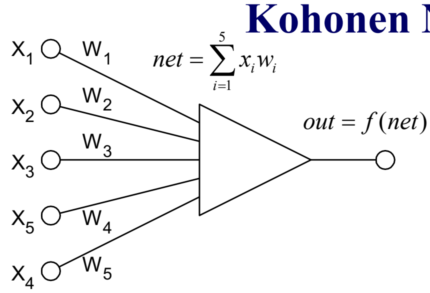
For binary weights and patterns net value can be found using equation:

$$net = \sum_{i=1}^{n} x_i w_i = \mathbf{X}\mathbf{W}^T = n - 2HD$$

where n is the number of inputs and HD is the Hamming distance between input vector **X** and weight vector **W**.

17

Kohonen Network



This concept can be extended to weights and patterns with analog values as long as both lengths of the weight vector and input pattern vectors are the same.

The Euclidean distance between weight vector **W** and input vector **X** is:

$$\|\mathbf{W}-\mathbf{X}\| = \sqrt{(w_1 - x_1)^2 + (w_2 - x_2)^2 + \cdots + (w_n - x_n)^2}$$

Also can be written as:

$$\|\mathbf{W}-\mathbf{X}\| = \sqrt{\sum_{i=1}^{n}(w_i - x_i)^2} = \sqrt{\sum_{i=1}^{n}(w_i w_i - 2w_i x_i + x_i x_i)}$$

$$\|\mathbf{W}-\mathbf{X}\| = \sqrt{\mathbf{W}\mathbf{W}^T - 2\mathbf{W}\mathbf{X}^T + \mathbf{X}\mathbf{X}^T} \quad \textit{(matrix form)}$$

18

Kohonen Network

$$\|\mathbf{W}-\mathbf{X}\| = \sqrt{\mathbf{W}\mathbf{W}^T - 2\mathbf{W}\mathbf{X}^T + \mathbf{X}\mathbf{X}^T}$$

Now, if the lengths of both the weight and input vectors are normalized to value of one:

$\|\mathbf{X}\| = 1$ and $\|\mathbf{W}\| = 1$

then the equation simplifies to:

$$\boxed{\|\mathbf{W}-\mathbf{X}\| = \sqrt{2 - 2\mathbf{W}\mathbf{X}^T}}$$

NOTE: the maximum value of net value (net=1), is when **W** and **X** are identical…

Kohonen Networks (WTA)

- *Derivation*
- *Steps*
- *Example*
- *Problems & remedies*

Kohonen Network

(unsupervised training process)

1. All patterns are normalized (the lengths of the pattern vectors are normalized to unity).

2. Weights are chosen randomly for all neurons

$$\begin{cases} z_1 = \dfrac{x_1}{\sqrt{\sum_{i=1}^{n} x_i^2}} \\ \cdots\cdot \\ z_n = \dfrac{x_n}{\sqrt{\sum_{i=1}^{n} x_i^2}} \end{cases}$$

© M. Manic, CMSC 409: Artificial Intelligence, F23 Page 21 Session 17, Updated on 10/11/23 6:13:12 PM

21

Kohonen Network

(unsupervised training process)

3. Lengths of the weight vectors are normalized to unity.

4. A pattern is applied to an input and *net* values are calculated for all neurons

$$net = \sum_{i=1}^{5} z_i v_i = \mathbf{Z}\mathbf{V}^T$$

$$\begin{cases} v_1 = \dfrac{w_1}{\sqrt{\sum_{i=1}^{n} w_i^2}} \\ \cdots\cdot \\ v_n = \dfrac{w_n}{\sqrt{\sum_{i=1}^{n} w_i^2}} \end{cases}$$

© M. Manic, CMSC 409: Artificial Intelligence, F23 Page 22 Session 17, Updated on 10/11/23 6:13:12 PM

22

Kohonen Network
(unsupervised training process)

5. A winning neuron is chosen (neuron with largest *net* value).

6. Weights for the winner k are modified using a weighted average:

$$\mathbf{W}_k = \mathbf{V}_k + \alpha \mathbf{Z}$$

where:
α - is the learning constant,
k – index of winning neuron

weights of other neurons are not modified.

Kohonen Network
(unsupervised training process)

7. Weights for the winning neuron are normalized.

8. Another pattern is applied (go to step 4.).

$$\begin{cases} v_1 = \dfrac{w_1}{\sqrt{\sum_{i=1}^{n} w_i^2}} \\ \quad\dots. \\ v_n = \dfrac{w_n}{\sqrt{\sum_{i=1}^{n} w_i^2}} \end{cases}$$

Kohonen Network
(unsupervised training process)

During pattern applications **some neurons** are frequent winners and other never take part in the process. The latter ones are eliminated and the **number of recognized clusters** is equal to the **number of surviving neurons**.

NOTE: *number of clusters might not be known upfront. You can start with larger network and eliminate neurons as you go. However, there is a danger of misclassification in this case.*

Things to keep in mind:

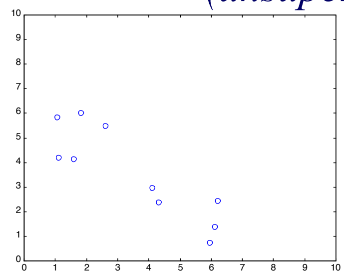
• Clusteing is strongly dependent on the initial set of randomly chosen weights, and order of updating.

• During the normalization process important information about the length of input patterns is lost.

Kohonen Networks (WTA)

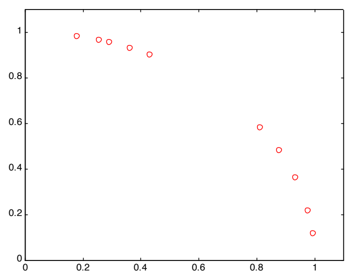
- *Derivation*
- *Steps*
- *Example*
- *Problems & remedies*

Kohonen Network
(unsupervised training process)



Original patterns

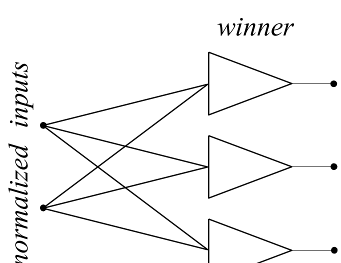
5.9630	0.7258
4.1168	2.9694
1.8184	6.0148
6.2139	2.4288
6.1290	1.3876
1.0562	5.8288
4.3185	2.3792
2.6108	5.4870
1.5999	4.1317
1.1046	4.1969



Normalized patterns

0.9927	0.1208
0.8110	0.5850
0.2894	0.9572
0.9314	0.3640
0.9753	0.2208
0.1783	0.9840
0.8759	0.4825
0.4296	0.9030
0.3611	0.9325
0.2545	0.9671

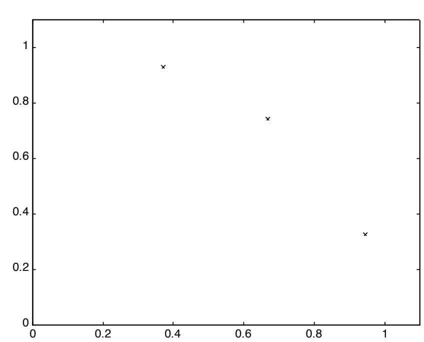
Kohonen Network
(unsupervised training process)



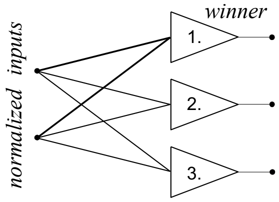
$$\begin{bmatrix} .9459 & .3243 \\ .669 & .7433 \\ .3714 & .9285 \end{bmatrix}$$

normalized initial weights

randomly chosen number of neurons…



Kohonen Network
(unsupervised training process)



applying the first pattern $Z_1 = \begin{bmatrix} 0.9927 & 0.1208 \end{bmatrix}$

initial weights net

$$\begin{bmatrix} .9459 & .3243 \\ .6690 & .7433 \\ .3714 & .9285 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.9782 \\ 0.7539 \\ 0.4809 \end{bmatrix}$$

From here, neuron #1 is the winner. Therefore, weights for neuron #1 are updated and normalized:

$\left(\frac{1.2437}{\sqrt{1.2437^2+0.3606^2}}\right)=0.96044$

$\mathbf{W}_k = \mathbf{V}_k + \alpha \mathbf{Z} =$ (0.9459 0.3243) + alpha (0.9927 0.1208)=

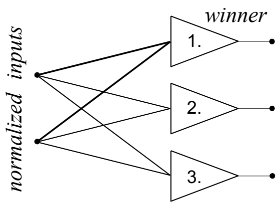
= (0.9459 0.3243) + 0.3 (0.9927 0.1208) = (1.2437 0.3606) ==> (0.9605 0.2784) *normalization*

$$\begin{bmatrix} .9459 & .3243 \\ .6690 & .7433 \\ .3714 & .9285 \end{bmatrix} \xrightarrow[\text{weight update}]{} \begin{bmatrix} .9605 & .2784 \\ .6690 & .7433 \\ .3714 & .9285 \end{bmatrix}$$

1st neuron updated

29

Kohonen Network
(unsupervised training process)



applying the second pattern $\begin{bmatrix} 0.8110 & 0.5850 \end{bmatrix}$

initial weights net

$$\begin{bmatrix} .9605 & .2784 \\ .6690 & .7433 \\ .3714 & .9285 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.9418 \\ 0.9774 \\ 0.8444 \end{bmatrix}$$

From here, neuron #2 is the winner. Therefore weights for neuron #2 are updated and normalized:

$\mathbf{W}_k = \mathbf{V}_k + \alpha \mathbf{Z} =$ (0.6690 0.7433) + alpha (0.8110 0.5850)

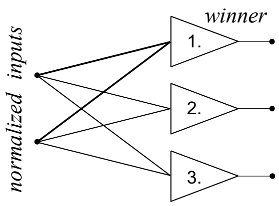
= (0.6690 0.7433) + 0.3 (0.8110 0.5850) = (0.9123 0.9188) ==> (0.7046 0.7096) *normalization*

$$\begin{bmatrix} .9605 & .2784 \\ .6690 & .7433 \\ .3714 & .9285 \end{bmatrix} \xrightarrow[\text{weight update}]{} \begin{bmatrix} .9605 & .2784 \\ .7046 & .7096 \\ .3714 & .9285 \end{bmatrix}$$

2nd neuron updated

30

Kohonen Network
(unsupervised training process)



applying the third pattern $Z_3 = [0.2894 \quad 0.9572]$

initial weights net

$$\begin{bmatrix} .9605 & .2784 \\ .7046 & .7096 \\ .3714 & .9285 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.5445 \\ 0.8832 \\ 0.9962 \end{bmatrix}$$

From here, neuron #3 is the winner. Therefore weights for neuron #3 are updated and normalized:

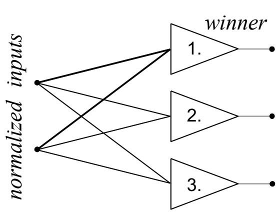
$$\mathbf{W}_k = \mathbf{V}_k + \alpha\mathbf{Z} = (0.3714 \quad 0.9285) + \text{alpha}\ (0.2894 \quad 0.9572)$$

$$= (0.3714 \quad 0.9285) + 0.3\ (0.2894 \quad 0.9572) = (0.4582 \quad 1.2156) \xRightarrow{\text{normalization}} (0.3527 \quad 0.9357)$$

$$\begin{bmatrix} .9605 & .2784 \\ .7046 & .7096 \\ .3714 & .9285 \end{bmatrix} \xRightarrow{\text{weight update}} \begin{bmatrix} .9605 & .2784 \\ .7046 & .7096 \\ .3527 & .9357 \end{bmatrix}$$

3rd neuron updated

Kohonen Network
(unsupervised training process)



applying the fourth pattern $Z_4 = [0.9314 \quad 0.3640]$

initial weights net

$$\begin{bmatrix} .9605 & .2784 \\ .7046 & .7096 \\ .3527 & .9357 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.996 \\ 0.915 \\ 0.669 \end{bmatrix}$$

From here, neuron #1 is the winner. Therefore weights for neuron #3 are updated and normalized:

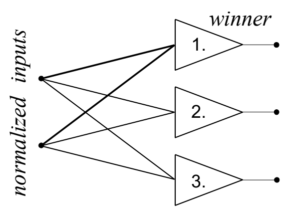
$$\mathbf{W}_k = \mathbf{V}_k + \alpha\mathbf{Z} = (0.9605 \quad 0.2784) + \text{alpha}\ (0.9314 \quad 0.3640)$$

$$= (0.9605 \quad 0.2784) + 0.3\ (0.9314 \quad 0.3640) = (1.2399 \quad 0.3876) \xRightarrow{\text{normalization}} (0.9544 \quad 0.2983)$$

$$\begin{bmatrix} .9605 & .2784 \\ .7046 & .7096 \\ .3527 & .9357 \end{bmatrix} \xRightarrow{\text{weight update}} \begin{bmatrix} .9544 & .2983 \\ .7046 & .7096 \\ .3527 & .9357 \end{bmatrix}$$

1st neuron updated

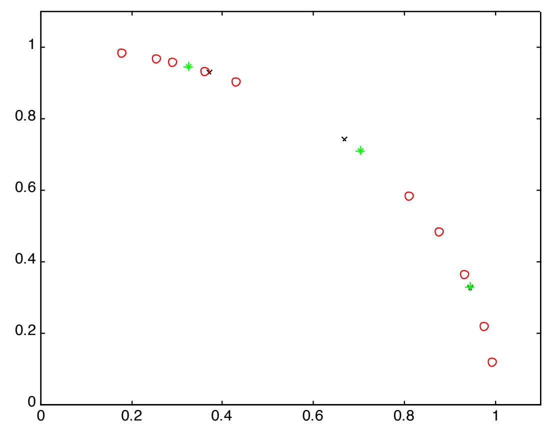
Kohonen Network
(unsupervised training process)



Process continues for all patterns

weights after the first iteration

$$\begin{bmatrix} .9605 & .2784 \\ .6690 & .7433 \\ .3714 & .9285 \end{bmatrix}$$

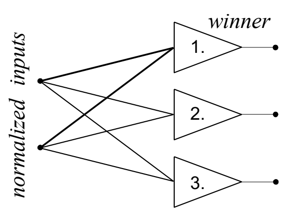


o - normalized patterns

x - initial weights

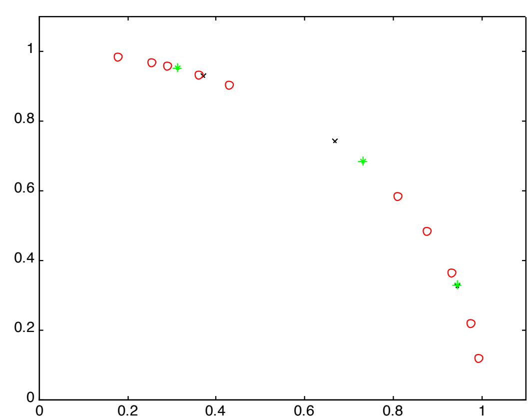
* - current weights

Kohonen Network
(unsupervised training process)



weights after the second iteration

0.9439	0.3302
0.7309	0.6825
0.3119	0.9501

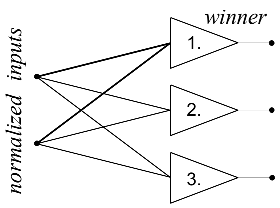


o - normalized patterns

x - initial weights

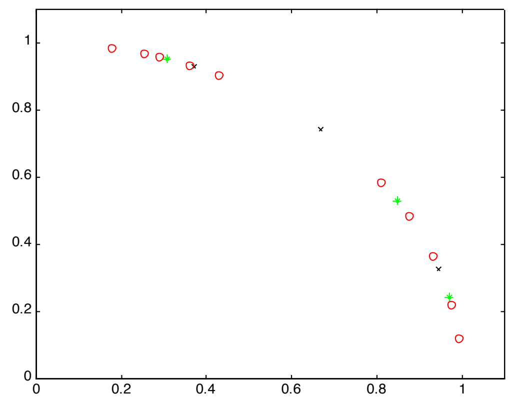
* - current weights

Kohonen Network
(unsupervised training process)



weights after the 30 iterations

0.9699 0.2435
0.8492 0.5281
0.3072 0.9516



o - normalized patterns

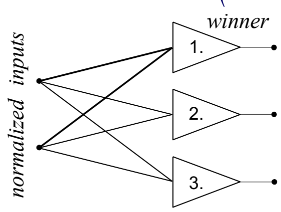
x - initial weights

* - current weights

weights are representing center of clusters

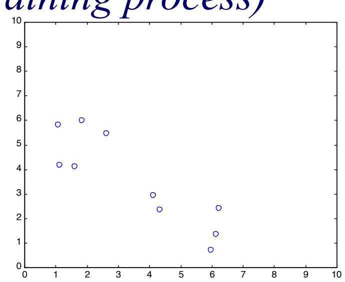
35

Kohonen Network
(unsupervised training process)



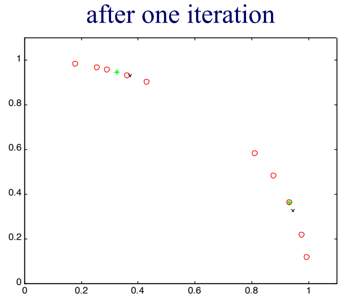
Initial weights

$$\begin{bmatrix} .9605 & .2784 \\ .7046 & .7096 \\ .3527 & .9357 \end{bmatrix}$$



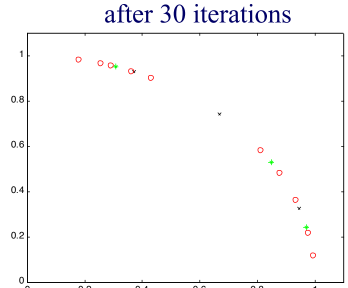
Patterns (initially)

after one iteration



Patterns (after one iterat.)

after 30 iterations



Patterns (after 30 iterat.)

36

Kohonen Networks (WTA)

- *Derivation*
- *Steps*
- *Example*
- *Problems & remedies*

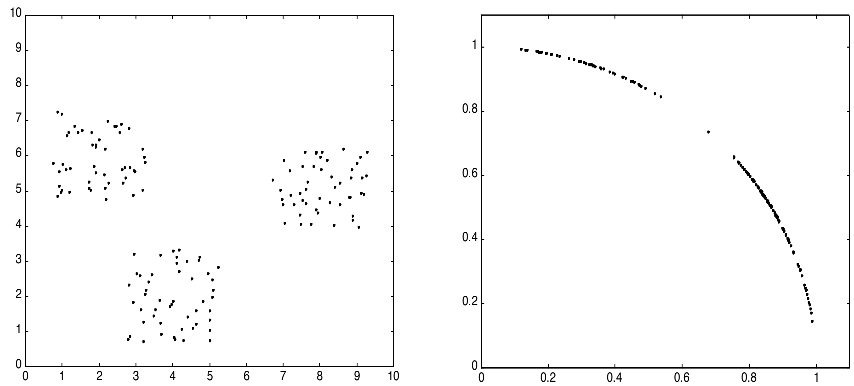
Kohonen Networks
Problems & Remedies

1. Important information about the length of the vector is lost during the normalization process

2. Clustering may depend on:
   a) Order patterns are applied
   b) Number of initial neurons
   c) Initial weights

Kohonen Networks
Problems & Remedies

Important information about length of the vector is lost during the normalization process



Kohonen Networks
Problems & Remedies

Problem:
Important information about length of the vector is lost during the normalization process.

Possible remedies:
• The problem can be solved by increasing a dimension by one and usage of vector angles as variables. Lengths are the same.
This approach (used by Kohonen) leads to complex trigonometric computations

• Other way to approach the problem is to project patterns into hypersphere of higher dimensionality. This way all patterns have the same length but important information is not lost.