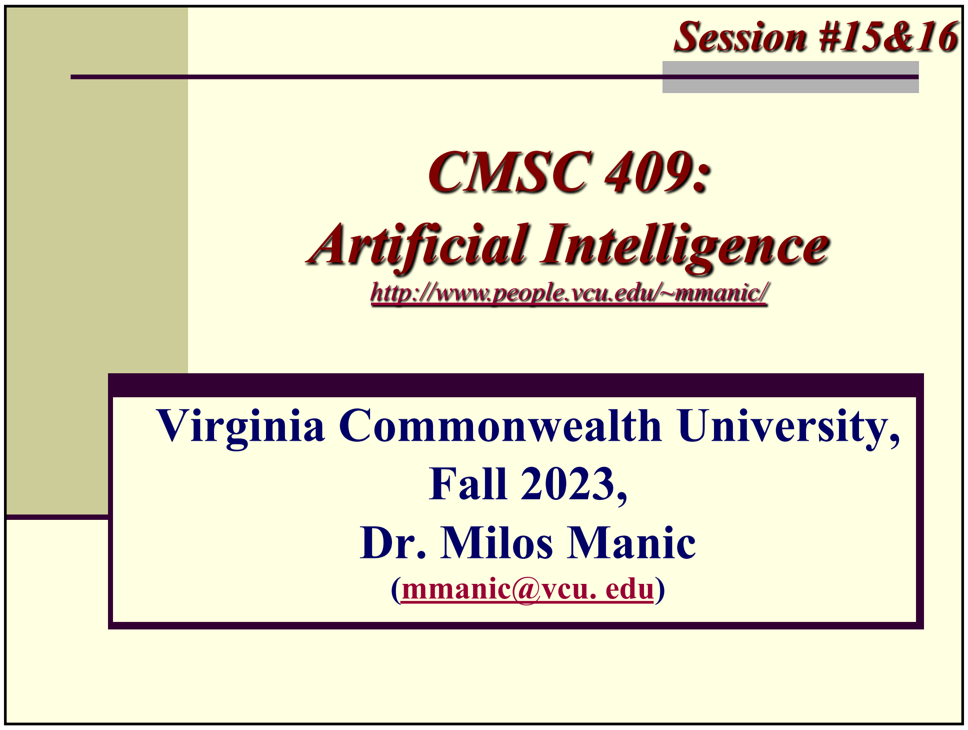


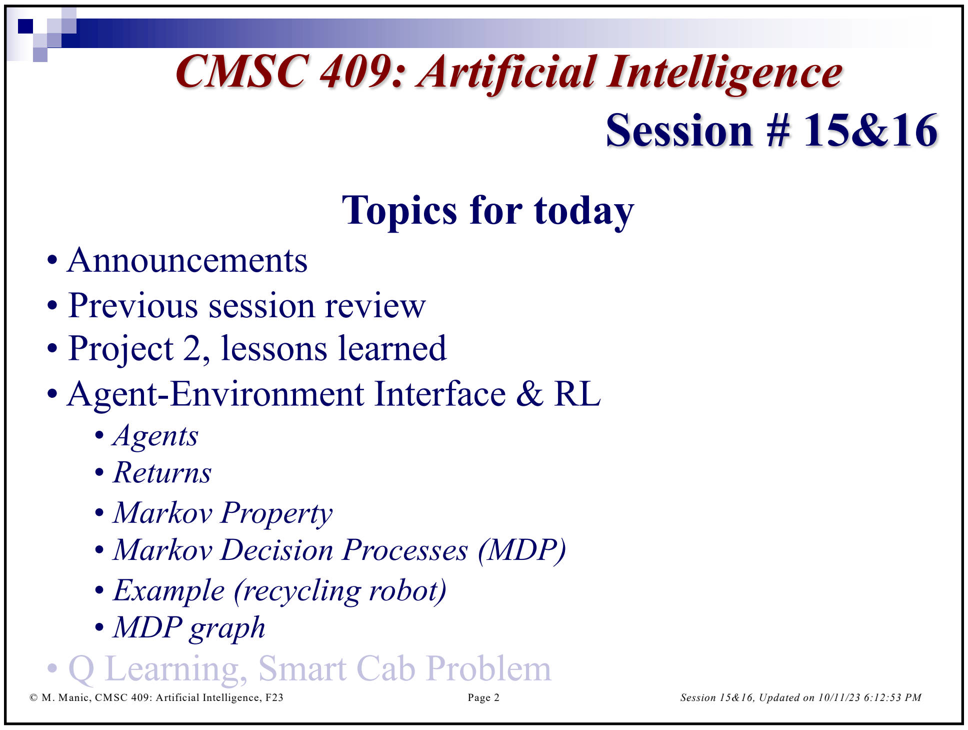
Session #15&16

CMSC 409: Artificial Intelligence

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

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

1

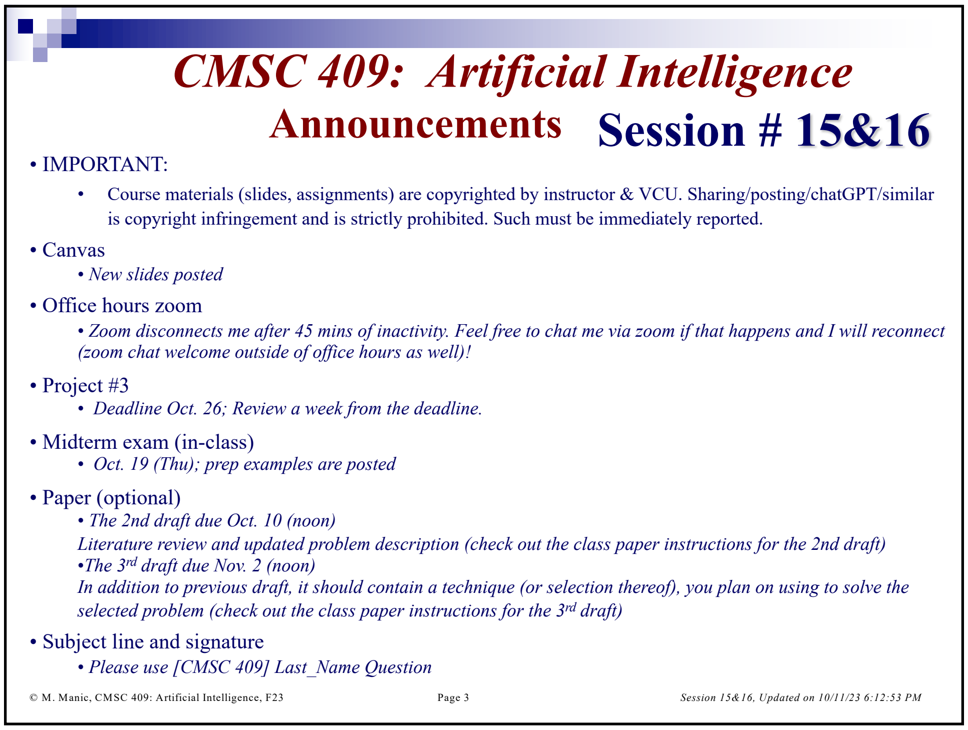


CMSC 409: Artificial Intelligence
Session # 15&16

Topics for today

- Announcements
- Previous session review
- Project 2, lessons learned
- Agent-Environment Interface & RL
  - *Agents*
  - *Returns*
  - *Markov Property*
  - *Markov Decision Processes (MDP)*
  - *Example (recycling robot)*
  - *MDP graph*
- Q Learning, Smart Cab Problem

2

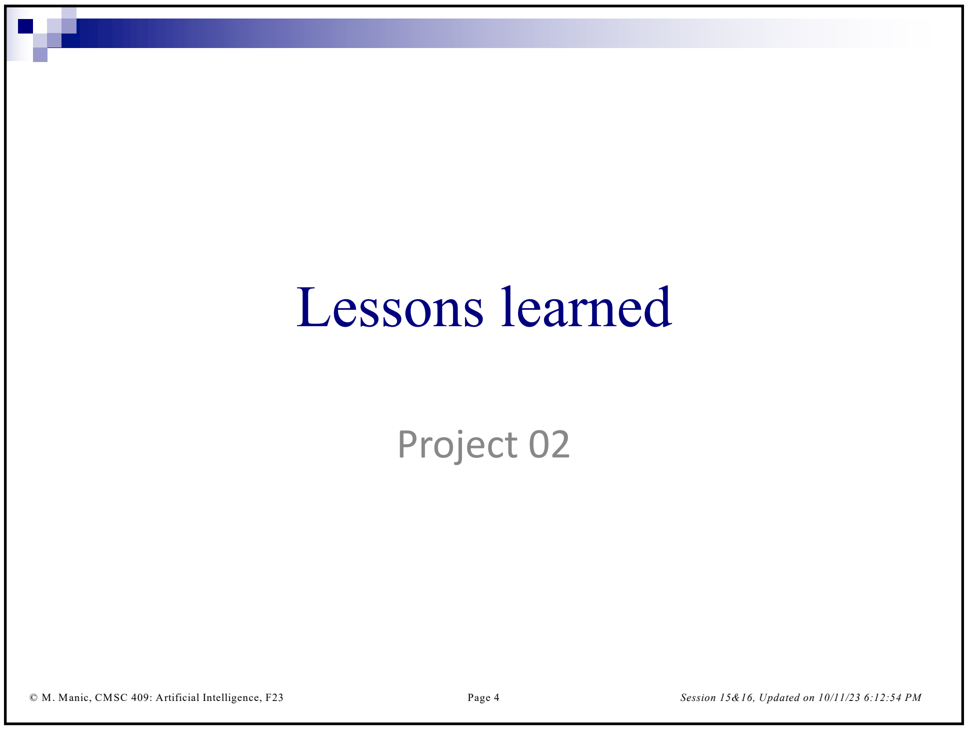


CMSC 409: Artificial Intelligence

Announcements Session # 15&16

- 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 2nd draft due Oct. 10 (noon)*
    Literature review and updated problem description (check out the class paper instructions for the 2nd draft)
  - *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*

3



Lessons learned

Project 02

4

Project 2 statistics

STATISTICS

COUNT	27
Minimum Value	0
Maximum Value	16
Range	16
Average	11.037
Median	13
Standard Deviation	5.853
Variance	34.267

GRADE DISTRIBUTION

Greater than 100	8
90 - 100	5
80 - 89	6
70 - 79	0
60 - 69	1
50 - 59	0
40 - 49	2
30 - 39	0
20 - 29	0
10 - 19	0
0 - 9	6

Project 2 review

PR 2.1:
- All requested plots should be included in the report
- Missing in some cases:
  - TE of training
  - Confusion matrix for testing
  - Comparison of confusion matrix between project 1 & 2
  - Normalization - refer to the one shown in class
  - Discussion on data distribution (25% vs. 75% training), discussion for activation function (hard & soft)

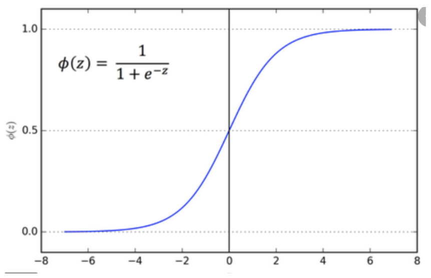
- Activation function
  - The provided dataset is driving the choice of activation function; i.e. the desired outputs are 0 & 1 => should use unipolar activation function (soft or hard).
  - Unipolar soft activation is similar to sigmoid (but NOT the same).

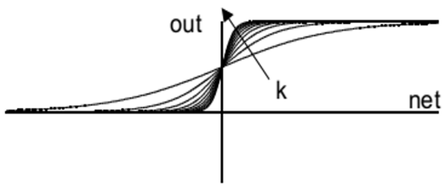
$$o = f(knet) = \frac{1}{1 + \exp(-knet)}$$

soft vs. sigmoid

$$S(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid vs. unipolar soft activation





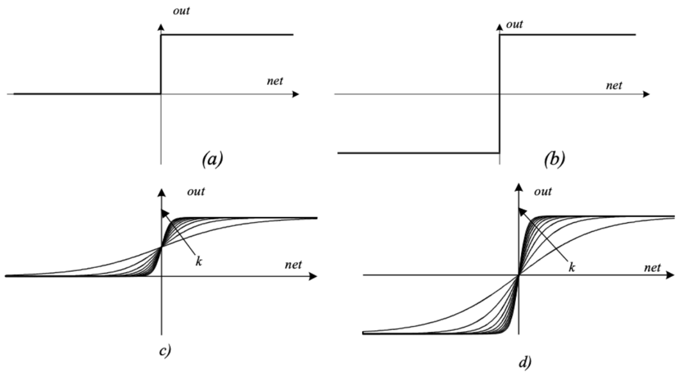
$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

$$o = f(knet) = \frac{1}{1 + \exp(-knet)}$$

Reminder on hard vs soft act. function (Session 09 & 10); This project was using unipolar (not bipolar act. fun.).

Learning Agents – transfer function

(for now, note the shape of the function only)



$$o = f(knet) = \frac{1}{1 + \exp(-knet)}$$

$$o = f(knet) = \tanh(knet) = \\ = \frac{2}{1 + \exp(-2knet)} - 1$$

Total Error

- **Total Error**

  The true positives, false positives, their rates, and corresponding accuracy/misclassification error represent metrics typical for confusion matrix (yes/no decisions) - Session 6.

  For evaluating the success of neural network training, we typically use TE (Total Error, i.e. sum of squares of *(d-o)*, as in Session 9&10.

  The TE is calculated for one iteration (epoch), for all training/testing patterns (training/testing TE), and that is the value we compare against the given *Epsilon*. In other words, the TE is our stopping criterion (again, for training).

9

Project 2 review

Pr. 2.2 Soft vs. hard activation function (5 pts)

Compare and discuss results when hard unipolar activation was used, vs. when soft unipolar activation function was used. You should include plots and be specific (provide quantitative comparisons when comparing). Comment on each training/testing distribution and each data set.

- Results should be discussed separately for each dataset (A, B, C).
- Should discuss the difference between TE for hard and soft act. function;

  The soft act. function yields more "optimized" separation, for ex.:

  For dataset A, TE should be near 0 (soft act. fun.), and 0 (hard act. fun.)

  The TE should be worse (larger) for sets A->B->C (either act. fun)
    - but especially for hard act. fun

  When algorithm is converging, hard act. function may take fewer iterations (less time), and might be sufficient for simpler datasets (with clear class/behavior separation).

  Learning rate alpha & gain
    - smaller alpha/gain is "safer" (slower training) but may lead to smaller error (and in case of gain, more optimized solution)
    - smaller gain effectively means more "linear" act. fun; larger gain means more "hard act. fun." like activation function

10

Python code snippets (1/2)

```
# Normalize Data
def prep_data(df):
    df['Cost'] = (df['Cost'] - df['Cost'].min()) / (df['Cost'].max() - df['Cost'].min())
    df['Weight'] = (df['Weight'] - df['Weight'].min()) / (df['Weight'].max() - df['Weight'].min())
    return df
```

above code snippet illustrates a way of doing data normalization

....

```
# Hard Activation Function
def hard_activation_function(x):
    if x >= 0:
        return 1
    else:
        return 0
```

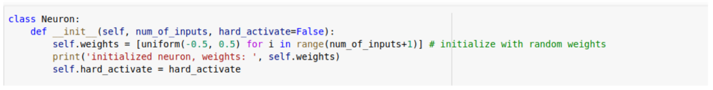
```
# Soft Activation Function
def soft_activation_function(x, y):
    return 1/(1+math.exp(-y*x))
```

hard and soft activation functions can be implemented as follows

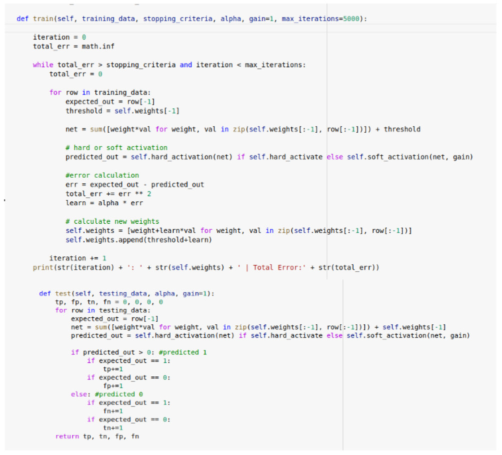
...

Python code snippets (2/2)

```
# Train Neuron
```



```
class Neuron:
    def __init__(self, num_of_inputs, hard_activate=False):
        self.weights = [uniform(-0.5, 0.5) for i in range(num_of_inputs+1)] # initialize with random weights
        print('initialized neuron, weights: ', self.weights)
        self.hard_activate = hard_activate
```



```
  def train(self, training_data, stopping_criteria, alpha, gain=1, max_iterations=5000):

      iteration = 0
      total_err = math.inf

      while total_err > stopping_criteria and iteration < max_iterations:
          total_err = 0

          for row in training_data:
              expected_out = row[-1]
              threshold = self.weights[-1]

              net = sum([weight*val for weight, val in zip(self.weights[:-1], row[:-1])]) + threshold

              # hard or soft activation
              predicted_out = self.hard_activation(net) if self.hard_activate else self.soft_activation(net, gain)

              #error calculation
              err = expected_out - predicted_out
              total_err += err ** 2
              learn = alpha * err

              # calculate new weights
              self.weights = [weight+learn*val for weight, val in zip(self.weights[:-1], row[:-1])]
              self.weights.append(threshold+learn)

          iteration += 1
      print(str(iteration) + ': ' + str(self.weights) + ' | Total Error:' + str(total_err))

  def test(self, testing_data, alpha, gain=1):
      tp, fp, tn, fn = 0, 0, 0, 0
      for row in testing_data:
          expected_out = row[-1]
          net = sum([weight*val for weight, val in zip(self.weights[:-1], row[:-1])]) + self.weights[-1]
          predicted_out = self.hard_activation(net) if self.hard_activate else self.soft_activation(net, gain)

          if predicted_out > 0: #predicted 1
              if expected_out == 1:
                  tp+=1
              if expected_out == 0:
                  fp+=1
          else: #predicted 0
              if expected_out == 1:
                  fn+=1
              if expected_out == 0:
                  tn+=1
      return tp, tn, fp, fn
```

During submission

- Deliverable
  - Send the report in the pdf format
  - Single file (code attached separately)
  - All deliverables should be included in the zip file (Do not attach report separately)
- Naming of the zip file
  - Name the zip file as asked in the Project specification
- Team work:
  - State the workload distribution clearly or it will be considered as equal

Agent-Environment Interface & RL

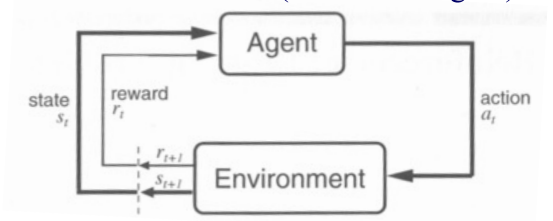
- ☐ *Returns*
- ☐ *Markov Property*
- ☐ *Markov Decision Processes (MDP)*
- ☐ *Example (recycling robot)*
- ☐ *MDP graph*

Reinforcement Learning by Sutton & Barto, A Bradford book, 1998.

Agent-Environment Interface

Agent

- Agent is both learner and decision-maker
- RL – learning from interaction to achieve a goal
- Agent interacts with **environment** (outside the agent)



- At each time step t, agent learns about the **environment's state** s_t, and selects an action, a_t.
- As a consequence of its action, agent receives a numerical reward r, and finds itself in a new state, s_{t+1}.

Agent-Environment & RL

Returns

- **Goals of an Agent:**
  - To maximize the **expected return** R_{t+1} in the long run (agent-environment)
- **Tasks can be…**
  - **Episodic tasks** (return in cases of "episodes" or plays of a game, trips through a maze; T-final time step)

$$R_t = r_{t+1} + r_{t+2} + ... + r_T$$

  - **Continuing tasks** (continual process-control tasks), maximize discounted return
  - **Discounting**

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

  - **Discount rate (γ)**:

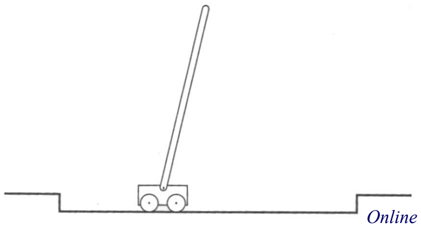
$$0 \le \gamma \le 1$$

    - *if $\gamma < 1$, return finite (if reward sequence r_k is bounded)*
    - *if $\gamma = 0$, the agent is myopic (maximizing only immediate rewards)*
    - *if $\gamma \rightarrow 1$, the agent is more farsighted (future rewards weighted more strongly)*

Agent-Environment & RL

Returns

- **Pole balancing (inverted pendulum)**
  - *Goal – keep the pole from falling*
  - *Could be treated as episodic or continuing task*
    - ***episodic** task – each attempt an episode (+1 for each step with no failure)*
      - *return is the number of steps until failure*
    - ***continuing** task – discounting (-1 on each failure, 0 otherwise)*
      - *return related to -γ^k, k – number of steps before failure*



Online version of the pole balancing game (Unity):
https://rajatbakshi.itch.io/wobble-pole

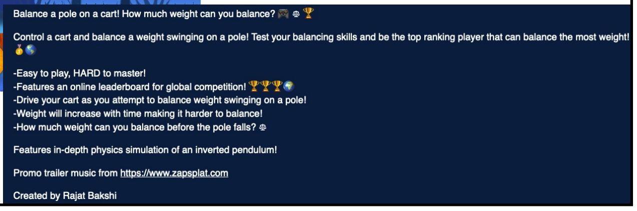
17

Agent-Environment & RL

Pole balancing (inverted pendulum)

Online version of the pole balancing game (Unity):
https://rajatbakshi.itch.io/wobble-pole

With time the weight increases making it harder to balance (you can use keyboard if easier…)


Balance a pole on a cart! How much weight can you balance? 🎮 ⚖ 🏆

Control a cart and balance a weight swinging on a pole! Test your balancing skills and be the top ranking player that can balance the most weight! 🥇🌎

-Easy to play, HARD to master!
-Features an online leaderboard for global competition! 🏆🏆🏆🌍
-Drive your cart as you attempt to balance weight swinging on a pole!
-Weight will increase with time making it harder to balance!
-How much weight can you balance before the pole falls? ⚖

Features in-depth physics simulation of an inverted pendulum!

Promo trailer music from https://www.zapsplat.com

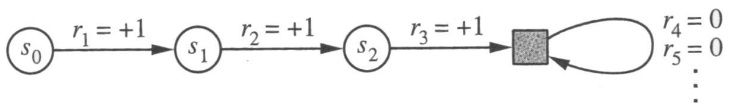
Created by Rajat Bakshi

18

Agent-Environment & RL

Markov Property

- Goal – keep the pole from falling
  - *Episodic tasks with finite sequence of time steps (starting from 0)*
  - *For state s at time t of episode i ($s_{t,i}$), and similarly for the action, reward, probability, final time step ($a_{t,i}$, $r_{t,i}$, $\pi_{t,i}$, $T_{t,i}$).*



(where absorbing state - episode termination, transitions to itself only with rewards=0)

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad => \quad \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

(continuing tasks reduced to episodic, with $T=\infty$ or $\gamma=1$, but not both).

Agent-Environment & RL

Markov Property for RL

- *Agent-environment*
  - Agent makes decision based on a signal from the environment
    - ***Markov property*** *– property of environments and their state signals*
    - *Current state maintains that "knowledge", not a sequence of events that preceded it*
    - *Idea that one can make predictions based on the current state alone (just as well if one knew complete past history)*
  - State signal
    - *Sensory measurement; original sensations, processed; complex structures of the sequence of sensations over time*
  - Example
    - *Peripheral vision; one hard drive failing after previous sequence of fail/repair; checkers or chess position, position/velocity of cannonball*

State signal that summarizes past sensations in a way that all relevant information is retained – is a state signal having the Markov property!

Agent-Environment & RL

Markov Property for RL

- *General probability distribution of an action at time **t+1** (for all past events):*

$$P\{s_{t+1}=s',\ r_{t+1}=r \mid s_t, a_t, r_t,\ s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0, r_0\}$$

- *State signal has the Markov property and is a Markov state:*

$$P\{s_{t+1}=s',\ r_{t+1}=r \mid s_t, a_t\}$$

iff both equations are equal for all s', r, and histories s_t, a_t, r_t, …, s_0, a_0, r_0.

*Decisions and values in reinforcement learning are functions **of the current state only**.*

*Markov Chains do **not** have the memory (future state is based on the current state). However, current state is based on the previous, and so on, so in that sense the current state is really based on a whole previous history.*

Agent-Environment & RL

Markov Decision Processes (MDP) for RL

- *MDP are RL tasks that satisfy Markov property*
- *If state and action spaces are finite – finite Markov decision process (finite MDP)*

- **Transition probabilities**
  - *Given any state and action **s** and **a**, the **probability of next state s' occurring** is:*

$$P^{a}_{ss'} = \Pr\{s_{t+1}=s' \mid s_t = s, a_t = a\}$$

  - *Given the current state and action **s** and **a**, and the next state **s'**, the **expected value of the next reward** is:*

$$R^{a}_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

Agent-Environment & RL

Markov Decision Processes (MDP) for RL

- **Example - Recycling Robot**
  - *Robot collecting empty soda cans in an office*
  - *Robot makes a decision whether it should*
    1) actively search for a can, 2) wait stationary for someone to bring a can, 3) recharge itself
  - *Action set **A** based on State of Charge (SoC, or energy levels) of the battery:*

$$\mathbf{A}(high) = \{search, wait\}$$
$$\mathbf{A}(low) = \{search, wait, recharge\}$$

(possible SoC is high or low only)

  - *if battery level is **high**, robot can search*
    - *with probability α of staying in SoC **high** (and **1-** α of changing to **low**)*
  - *if battery level is **low**, robot can still search*
    - *with probability β of staying in SoC **low** (and **1-** β of depleting the battery)*
  - *Rewards*
    - *each can collected +1, if rescued -3*

Agent-Environment & RL

Markov Decision Processes (MDP) for RL

- **Example - Recycling Robot (cont.)**
  - *expected number of cans collected (i.e. expected reward) while searching (waiting) is R^{search} (R^{wait}), where $R^{search} > R^{wait}$.*
  - *Transition probabilities for the finite MDP:*

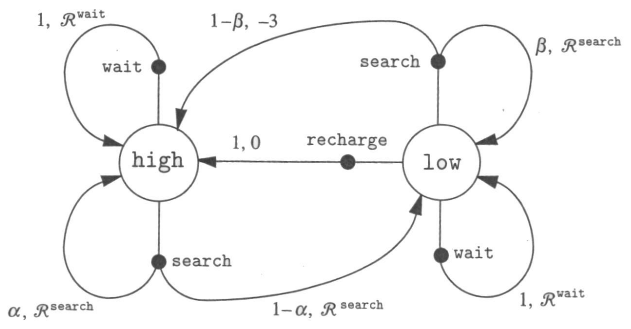
s	s'	a	$\mathcal{P}^a_{ss'}$	$\mathcal{R}^a_{ss'}$
high	high	search	α	$\mathcal{R}^{\texttt{search}}$
high	low	search	$1-\alpha$	$\mathcal{R}^{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$\mathcal{R}^{\texttt{search}}$
high	high	wait	1	$\mathcal{R}^{\texttt{wait}}$
high	low	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	high	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	low	wait	1	$\mathcal{R}^{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0

- *if battery level is **high**, robot can search; with probability α of staying with **high** (and **1-** α of changing to **low**)*
- *if battery level is **low**, robot can search; with probability β of staying with **low** (and **1-** β of depleting the battery)*

Agent-Environment & RL

Markov Decision Processes (MDP) for RL

- **Example - Recycling Robot (cont.)**
  - *Transition graph for the finite MDP (state nodes **high** & **low**; action nodes - solid circles; probabilities of arrows leaving an action node sum=1)*


1, $\mathcal{R}^{\text{wait}}$
wait
1−β, −3
search
β, $\mathcal{R}^{\text{search}}$
high
1, 0
recharge
low
search
wait
α, $\mathcal{R}^{\text{search}}$
1−α, $\mathcal{R}^{\text{search}}$
1, $\mathcal{R}^{\text{wait}}$