CS 2750 Machine Learning
Lecture 22

Reinforcement learning

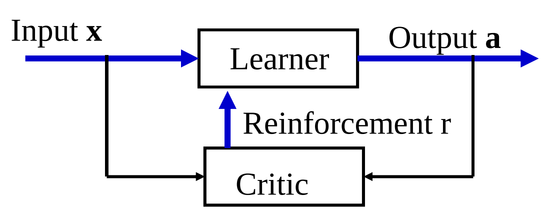
Milos Hauskrecht

milos@cs.pitt.edu

5329 Sennott Square

Reinforcement learning

- **We want to learn the control policy:** $\pi : X \rightarrow A$
- We see examples of **x** (but outputs *a* are not given)
- Instead of *a* we get a feedback *r* (reinforcement, reward) from a **critic** quantifying how good the selected output was



- The reinforcements may not be deterministic
- **Goal:** find $\pi : X \rightarrow A$ with the best expected reinforcements

Gambling example.

- **Game:** 3 different biased coins are tossed
  - The coin to be tossed is selected randomly from the three options and I always see which coin I am going to play next
  - I make bets on head or tail and I always wage $1
  - If I win I get $1, otherwise I lose my bet
- **RL model:**
  - **Input:** X – a coin chosen for the next toss,
  - **Action:** A – choice of head or tail,
  - **Reinforcements:** {1, -1}
- **A policy** $\pi : X \rightarrow A$

Example: $\pi :$ | **Coin1** → ***head*** | **Coin2** → ***tail*** | **Coin3** → ***head*** |

Gambling example

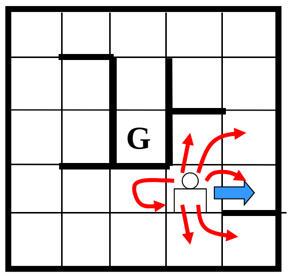
- **RL model:**
  - **Input:** X – a coin chosen for the next toss,
  - **Action:** A – choice of head or tail,
  - **Reinforcements:** {1, -1}
  - **A policy** $\pi :$ | **Coin1** → ***head*** | **Coin2** → ***tail*** | **Coin3** → ***head*** |
- **Learning goal: find** $\pi : X \rightarrow A$ $\pi :$ | **Coin1** → ***?*** | **Coin2** → ***?*** | **Coin3** → ***?*** |

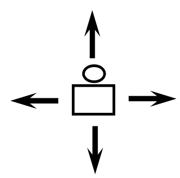
maximizing future expected profits

$E(\sum_{t=0}^{\infty} \gamma^t r_t)$ γ a discount factor = present value of money

Agent navigation example.

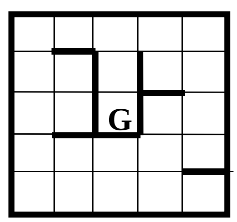
- **Agent navigation in the Maze**:
  - 4 moves in compass directions
  - Effects of moves are stochastic – we may wind up in other than intended location with non-zero probability
  - **Objective:** reach the goal state in the shortest expected time

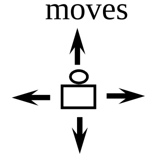




Agent navigation example

- **The RL model:**
  - **Input:** X – position of an agent
  - **Output:** A –a move
  - **Reinforcements:** R
    - -1 for each move
    - +100 for reaching the goal
  - **A policy:** $\pi : X \rightarrow A$





$\pi:$
Position 1 → ***right***
Position 2 → ***right***
…
Position 20 → ***left***

- **Goal: find the policy maximizing future expected rewards**

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

Objectives of RL learning

- **Objective:**
  Find a mapping $\pi^* : X \to A$
  That maximizes some combination of future reinforcements (rewards) received over time
- **Valuation models (quantify how good the mapping is):**
  - **Finite horizon model**

    $$E(\sum_{t=0}^{T} r_t) \qquad \text{Time horizon: } T > 0$$

  - **Infinite horizon discounted model**

    $$E(\sum_{t=0}^{\infty} \gamma^t r_t) \qquad \text{Discount factor: } 0 < \gamma < 1$$

  - **Average reward**

    $$\lim_{T \to \infty} \frac{1}{T} E(\sum_{t=0}^{T} r_t)$$

Exploration vs. Exploitation

- The (learner) actively interacts with the environment:
  - At the beginning the learner does not know anything about the environment
  - It gradually gains the experience and learns how to react to the environment
- **Dilemma (exploration-exploitation):**
  - After some number of steps, should I select the best current choice (**exploitation**) or try to learn more about the environment (**exploration**)?
  - **Exploitation** may involve the selection of a sub-optimal action and prevent the learning of the optimal choice
  - **Exploration** may spend to much time on trying bad currently suboptimal actions

Effects of actions on the environment

Effect of actions on the environment (next input **x** to be seen)

- No effect, the distribution over possible **x** is fixed; action consequences (rewards) are seen immediately,
- Otherwise, distribution of **x** can change; the rewards related to the action can be seen with some delay.

Leads to two forms of **reinforcement learning:**

- **Learning with immediate rewards**
  - **Gambling example**
- **Learning with delayed rewards**
  - **Agent navigation example**; move choices affect the state of the environment (position changes), a big reward at the goal state is delayed

RL with immediate rewards

- **Game:** 3 different biased coins are tossed
  - The coin to be tossed is selected randomly from the three options and I always see which coin I am going to play next
  - I make bets on head or tail and I always wage $1
  - If I win I get $1, otherwise I lose my bet
- **RL model:**
  - **Input:** X – a coin chosen for the next toss
  - **Action:** A – head or tail bet
  - **Reinforcements:** {1, -1}
- **Learning goal: find** $\pi : X \rightarrow A$

maximizing the future expected profits over time

$E(\sum_{t=0}^{\infty} \gamma^t r_t)$ γ a discount factor = present value of money

CS 2750 Machine Learning

RL with immediate rewards

- **Expected reward**

  $E(\sum_{t=0}^{\infty} \gamma^t r_t)$ γ - a discount factor = present value of money

- **Immediate reward case:**
  - Reward for the choice becomes available immediately
  - Our choice does not affect environment and thus future rewards

    $$E(\sum_{t=0}^{\infty} \gamma^t r_t) = E(r_0) + E(\gamma r_1) + E(\gamma^2 r_2) + \ldots$$

    $r_0, r_1, r_2 \ldots$ Rewards for every step

  - Expected one step reward for input $\mathbf{x}$ and the choice a : $R(\mathbf{x}, a)$

CS 2750 Machine Learning

RL with immediate rewards

Immediate reward case:

- Reward for the choice *a* becomes available immediately
- Expected reward for the input $\mathbf{x}$ and choice *a*: $R(\mathbf{x}, a)$
  - For the gambling problem it can be defined as:

    $$R(\mathbf{x}, a_i) = \sum_j r(\omega_j \mid a_i, \mathbf{x}) P(\omega_j \mid \mathbf{x}, a_i)$$

  - ω_j - a future outcome of the coin toss
  - Recall the definition of the expected loss
- **Expected one step reward for a strategy** $\pi : X \rightarrow A$

  $$R(\pi) = \sum_x R(\mathbf{x}, \pi(\mathbf{x})) P(\mathbf{x})$$

  $R(\pi)$ is the expected reward for $r_0, r_1, r_2 \ldots$

CS 2750 Machine Learning

RL with immediate rewards

- **Expected reward**

$$E(\sum_{t=0}^{\infty}\gamma^t r_t) = E(r_0) + E(\gamma r_1) + E(\gamma^2 r_2) + \ldots$$

- **Optimizing the expected reward :**

$$\max_{\pi} E(\sum_{t=0}^{\infty}\gamma^t r_t) = \max_{\pi}\sum_{t=0}^{\infty}\gamma^t E(r_t) = \max_{\pi}\sum_{t=0}^{\infty}\gamma^t R(\pi) = \max_{\pi} R(\pi)(\sum_{t=0}^{\infty}\gamma^t)$$

$$= (\sum_{t=0}^{\infty}\gamma^t)\max_{\pi} R(\pi)$$

$$\max_{\pi} R(\pi) = \max_{\pi}\sum_{x} R(\mathbf{x},\pi(\mathbf{x}))P(\mathbf{x}) = \sum_{x} P(\mathbf{x})[\max_{\pi(\mathbf{x})} R(\mathbf{x},\pi(\mathbf{x}))]$$

- **Optimal strategy:** $\pi^* : X \rightarrow A$

$$\pi^*(\mathbf{x}) = \arg\max_{a} R(\mathbf{x}, a)$$

RL with immediate rewards

- **We know that** $\pi^*(\mathbf{x}) = \arg\max_{a} R(\mathbf{x}, a)$
- **Problem:** In the RL framework we do not know $R(\mathbf{x}, a)$
  – The expected reward for performing action *a* at input **x**
- **How to get** $R(\mathbf{x}, a)$ **?**

RL with immediate rewards

- **Problem:** In the RL framework we do not know $R(\mathbf{x}, a)$
  - The expected reward for performing action *a* at input **x**
- **Solution:**
  - For each input **x** try different actions *a*
  - Estimate $R(\mathbf{x}, a)$ using the average of observed rewards

$$\tilde{R}(\mathbf{x}, a) = \frac{1}{N_{x,a}} \sum_{i=1}^{N_{x,a}} r_i^{x,a}$$

  - Action choice $\pi(\mathbf{x}) = \arg\max_a \tilde{R}(\mathbf{x}, a)$
  - Accuracy of the estimate: statistics (Hoeffding's bound)

$$P\left(\left|\tilde{R}(\mathbf{x}, a) - R(\mathbf{x}, a)\right| \geq \varepsilon\right) \leq \exp\left[-\frac{2\varepsilon^2 N_{x,a}}{(r_{\max} - r_{\min})^2}\right] \leq \delta$$

  - Number of samples: $N_{x,a} \geq \frac{(r_{\max} - r_{\min})^2}{2\varepsilon^2} \ln \frac{1}{\delta}$

RL with immediate rewards

- **On-line (stochastic approximation)**
  - An alternative way to estimate $R(\mathbf{x}, a)$
- **Idea:**
  - choose action *a* for input **x** and observe a reward $r^{x,a}$
  - Update an estimate

$$\tilde{R}(\mathbf{x}, a) \leftarrow (1 - \alpha)\tilde{R}(\mathbf{x}, a) + \alpha\, r^{x,a}$$

α - a learning rate

- **Convergence property:** The approximation converges in the limit for an appropriate learning rate schedule.
- Assume: $\alpha(n(x, a))$ - is a learning rate for *n*th trial of (*x*,*a*) pair
- Then the converge is assured if:

1. $\sum_{i=1}^{\infty} \alpha(i) = \infty$ 2. $\sum_{i=1}^{\infty} \alpha(i)^2 < \infty$

Exploration vs. Exploitation

- In the RL framework
  - the (learner) actively interacts with the environment.
  - At any point in time it has an estimate of $\tilde{R}(\mathbf{x}, a)$ for any input action pair
- **Dilemma:**
  - Should the learner use the current best choice of action (exploitation)

$$\hat{\pi}(\mathbf{x}) = \arg\max_{a \in A} \tilde{R}(\mathbf{x}, a)$$

  - Or choose other action *a* and further improve its estimate (exploration)

- **Different exploration/exploitation strategies exist**

Exploration vs. Exploitation

- **Uniform exploration**
  - Choose the "current" best choice with probability $1-\varepsilon$

$$\hat{\pi}(\mathbf{x}) = \arg\max_{a \in A} \tilde{R}(\mathbf{x}, a)$$

  - All other choices are selected with a uniform probability

$$\frac{\varepsilon}{|A|-1}$$

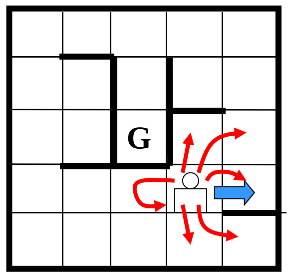
- **Boltzman exploration**
  - The action is chosen randomly but proportionally to its current expected reward estimate

$$p(a \mid \mathbf{x}) = \frac{\exp\left[\tilde{R}(x,a)/T\right]}{\sum_{a' \in A} \exp\left[\tilde{R}(x,a')/T\right]}$$

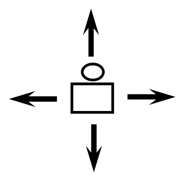
T – is temperature parameter. **What does it do?**

RL with delayed rewards.

- **Agent navigation in the Maze**:
  - 4 moves in compass directions
  - Effects of moves are stochastic – we may wind up in other than intended location with non-zero probability
  - **Objective:** reach the goal state in the shortest time

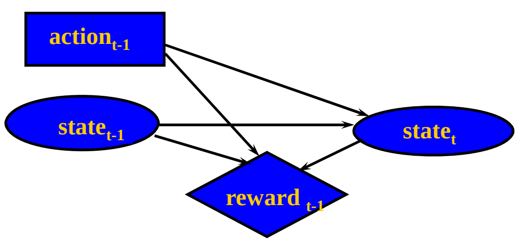


moves

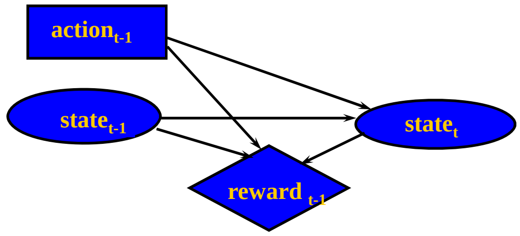


Learning with delayed rewards

- Actions, in addition to immediate rewards affect the next state of the environment and thus indirectly also future rewards
- We need a model to represent environment changes
- The model we use is called **Markov decision process (MDP)**
  - Frequently used in AI, OR, control theory
  - **Markov assumption**: next state depends on the previous state and action, and not states (actions) in the past



Markov decision process



Formal definition: 4-tuple (S, A, T, R)

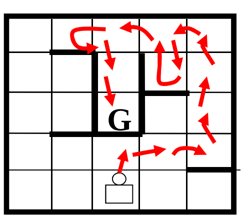
• **A set of states** S (X)	locations of a robot
• **A set of actions** A	move actions
• **Transition model** $S \times A \times S \rightarrow [0,1]$	where can I get with different moves
• **Reward model** $S \times A \times S \rightarrow \Re$	reward/cost for a transition

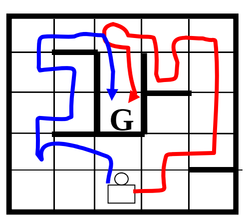
MDP problem

- We want to find the best policy $\pi^* : S \rightarrow A$
- **Value function** (V) for a policy, quantifies the goodness of a policy through, e.g. infinite horizon, discounted model

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

It: 1. combines future rewards over a trajectory
2. combines rewards for multiple trajectories (through expectation-based measures)





Value of a policy for MDP

- Assume a fixed policy $\pi : S \rightarrow A$
- How to compute the value of a policy under infinite horizon discounted model?

Fixed point equation:

$$V^{\pi}(s) = \underbrace{R(s,\pi(s))}_{\text{expected one step reward for the first action}} + \underbrace{\gamma \sum_{s' \in S} P(s' | s, \pi(s)) V^{\pi}(s')}_{\text{expected discounted reward for following the policy for the rest of the steps}}$$

$$\mathbf{v} = \mathbf{r} + \mathbf{U}\mathbf{v} \quad \Longrightarrow \quad \mathbf{v} = (\mathbf{I} - \mathbf{U})^{-1}\mathbf{r}$$

- For a finite state space– we get a set of linear equations

Optimal policy

- **The value of the optimal policy**

$$V^{*}(s) = \max_{a \in A} \left[\underbrace{R(s,a)}_{\text{expected one step reward for the first action}} + \underbrace{\gamma \sum_{s' \in S} P(s' | s, a) V^{*}(s')}_{\text{expected discounted reward for following the opt. policy for the rest of the steps}} \right]$$

Value function mapping form:

$$V^{*}(s) = (HV^{*})(s)$$

- **The optimal policy:** $\pi^{*} : S \rightarrow A$

$$\pi^{*}(s) = \arg\max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' | s, a) V^{*}(s') \right]$$

Computing optimal policy

Dynamic programming. Value iteration:

- computes the optimal value function first then the policy
- iterative approximation
- converges to the optimal value function

Value iteration (ε **)**

initialize **V** ;; V is vector of values for all states

repeat

  set $\mathbf{V}' \leftarrow \mathbf{V}$

  set $\mathbf{V} \leftarrow \mathbf{HV}$

until $\|\mathbf{V}' - \mathbf{V}\|_\infty \leq \varepsilon$

output $\pi^*(s) = \arg\max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V(s') \right]$

Reinforcement learning of optimal policies

- **In the RL framework we do not know the MDP model !!!**
- **Goal:** learn the optimal policy

$$\pi^* : S \rightarrow A$$

- **Two basic approaches:**
  - **Model based learning**
    - Learn the MDP model (probabilities, rewards) first
    - Solve the MDP afterwards
  - **Model-free learning**
    - Learn how to act directly
    - No need to learn the parameters of the MDP
  - A number of clones of the two in the literature

Model-based learning

- We need to learn **transition probabilities** and **rewards**
- **Learning of probabilities**
  - ML or Bayesian parameter estimates
  - Use counts $\tilde{P}(s'|s,a) = \frac{N_{s,a,s'}}{N_{s,a}}$ $\qquad N_{s,a} = \sum_{s' \in S} N_{s,a,s'}$
- **Learning rewards**
  - Similar to learning with immediate rewards

$$\tilde{R}(s,a) = \frac{1}{N_{s,a}} \sum_{i=1}^{N_{s,a}} r_i^{s,a}$$

- **Problem: on-line update of the policy**
  - **would require us to solve an MDP after every update !!**

Model free learning

- **Motivation:** value function update (value iteration):

$$V(s) \leftarrow \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right]$$

- Let

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$$

- Then $\quad V(s) \leftarrow \max_{a \in A} Q(s,a)$

- Note that the update can be defined purely in terms of Q-functions

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q(s',a')$$

Q-learning

- **Q-learning** uses the Q-value update idea
  - **But** relies on a stochastic (on-line, sample by sample) update

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \max_{a'} Q(s',a')$$

is replaced with

$$\hat{Q}(s,a) \leftarrow (1-\alpha)\hat{Q}(s,a) + \alpha \left(r(s,a) + \gamma \max_{a'} \hat{Q}(s',a') \right)$$

$r(s,a)$ - reward received from the environment after performing an action *a* in state *s*

s' - new state reached after action *a*

α - learning rate, a function of $N_{s,a}$ - a number of times *a* executed at *s*

Q-learning

The on-line update rule is applied repeatedly during direct interaction with an environment

Q-learning
initialize *Q(s,a) =0* for all *s,a* pairs
observe current state *s*
repeat
  select action *a* ; use some exploration/exploitation schedule
  receive reward *r*
  observe next state *s'*
  update $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right)$
  set *s* to *s'*
end repeat

Q-learning convergence

The **Q-learning is guaranteed to converge** to the optimal Q-values under the following conditions:

- Every state is visited and every action in that state is tried infinite number of times
  - This is assured via exploration/exploitation schedule
- The sequence of learning rates for each *Q(s,a)* satisfies:

1. $\sum_{i=1}^{\infty} \alpha(i) = \infty$ 2. $\sum_{i=1}^{\infty} \alpha(i)^2 < \infty$

$\alpha(n(s,a))$ - Is the learning rate for the *n*th trial of (*s*,*a*)

Exploration vs. Exploitation

- In the RL with the delayed rewards
  - At any point in time the learner has an estimate of $\hat{Q}(\mathbf{x}, a)$ for any state action pair
- **Dilemma:**
  - Should the learner use the current best choice of action (exploitation)

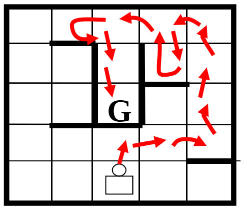
$$\hat{\pi}(\mathbf{x}) = \arg\max_{a \in A} \hat{Q}(\mathbf{x}, a)$$

  - Or choose other action *a* and further improve its estimate of $\hat{Q}(\mathbf{x}, a)$ (exploration)
- **Exploration/exploitation strategies**
  - **Uniform exploration**
  - **Boltzman exploration**

Q-learning speed-ups

- The basic Q-learning rule updates may propagate distant (delayed) rewards very slowly

Example:



- **Goal:** a high reward state
- To make the correct decision we need all Q-values for the current position to be good
- **Problem:**
  - in each run we back-propagate values only 'one-step' back. It takes multiple trials to back-propagate values multiple steps.

Q-learning speed-ups

- **Remedy:** Backup values for a larger number of steps

Rewards from applying the policy

$$q_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

We can substitute (immediate rewards with n-step rewards):

$$q_t^n = \sum_{i=0}^{n} \gamma^i r_{t+i} + \gamma^{n+1} \max_{a'} Q_{t+n}(s', a')$$

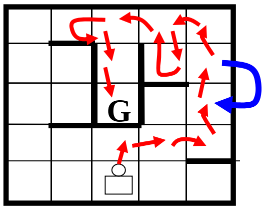
Postpone the update for *n* steps and update with a longer trajectory rewards

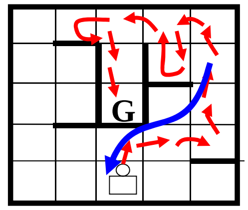
$$Q_{t+n+1}(s,a) \leftarrow Q_{t+n}(s,a) + \alpha \left(q_t^n - Q_{t+n}(s,a) \right)$$

Problems: - larger variance
- exploration/exploitation switching
- wait n steps to update

Q-learning speed-ups

- One step vs. n-step backup



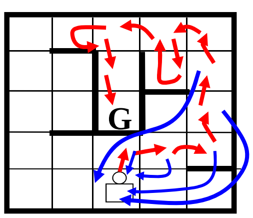


Problems with n-step backups:

- larger variance
- exploration/exploitation switching
- wait n steps to update

Q-learning speed-ups

- **Temporal difference (TD) method**
  - Remedy of the wait n-steps problem
  - Partial back-up after every simulation step
    - Similar idea: weather forecast adjustment



Different versions of this idea has been implemented

RL successes

- Reinforcement learning is relatively simple
  - On-line techniques can track non-stationary environments and adapt to its changes

- **Successful applications**:
  - TD Gammon – learned to play backgammon on the championship level
  - Elevator control
  - Dynamic channel allocation in mobile telephony
  - Robot navigation in the environment