CS4104 Applied Machine Learning

Reinforcement Learning

Reinforcement learning

- In the case of the agent acts on its environment, it receives some evaluation of its action (reinforcement), but is not told of which action is the correct one to achieve its goal

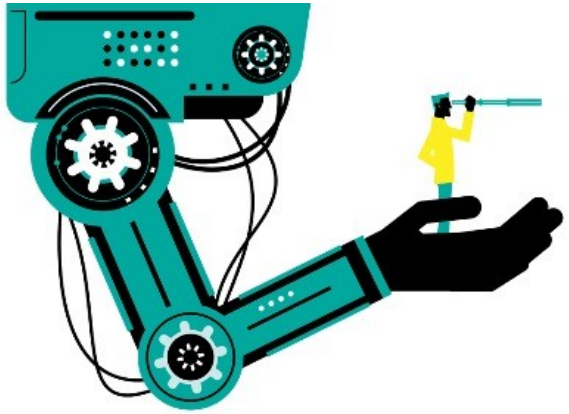
What is Reinforcement Learning?

- Reinforcement learning requires a learning agent to learn from the environment rather than being guided what to do.
- With the lack of designated guidelines and uncertain chances of matching the right actions, the reinforcement learning is indeed a trial-and-error learning.
- A numerical reward is used as a reinforcement signal to encourage the learning agent to successfully keep matching the expected outcomes.
- The learning agents will be guided by the problems they have encountered in past experiences and try to avoid them.

zeshan.khan@nu.edu.pk

Uses for Reinforcement Learning

- Control Systems
- Robotics
- Game Development
- Automation
- Advertisement
- E-commerce (e.g., Amazon)
- Industrial Automation (e.g., Stocks)



zeshan.khan@nu.edu.pk

Reinforcement learning

- Task

  Learn how to behave successfully to achieve a goal while interacting with an external environment

  - *Learn via experiences!*

- Examples
  - Game playing: player knows whether it win or lose, but not know how to move at each step
  - Control: a traffic system can measure the delay of cars, but not know how to decrease it.

zeshan.khan@nu.edu.pk

Paradigm	 Supervised Learning	 Unsupervised Learning	 Reinforcement Learning
Objective	$p_\theta(y\|x)$	$p_\theta(x)$	$\pi_\theta(a\|s)$
Applications	→ Classification → Regression	→ Inference → Generation	→ Prediction → Control

zeshan.khan@nu.edu.pk

Reinforcement Learning

Advantages

- No need for large labeled data sets
- Avoid human error and bias
- Can outperform humans
- Used to solve complex problems
- Error is corrected as there are occurrences
- Learns from experience

Disadvantages

Trial/Error can be time consuming and costly

Not preferred for solving methods

Needs a lot of data and computation especially when the environment has a lot of states

zeshan.khan@nu.edu.pk

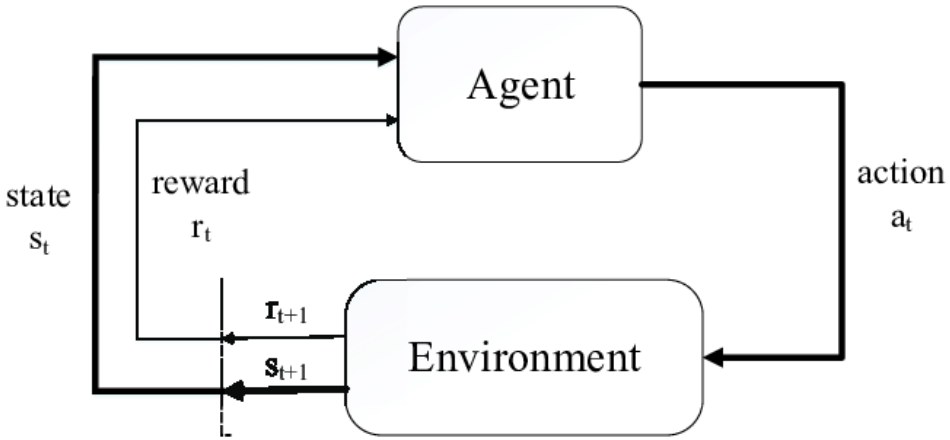
Prediction

Control

zeshan.khan@nu.edu.pk

Terminology

- Agent: entity that is performing the actions to accumulate reward
- Policy: decision making the agent employs to decide
- Reward: immediate return given to the agent when it performs a task
- Environment: scenario that an agent must face
- Value: long term return with discount
- State: current situation returned by the environment



zeshan.khan@nu.edu.pk

RL model

- Each percept(e) is enough to determine the State(the state is accessible)
- The agent can decompose the Reward component from a percept.
- The agent task: to find a optimal policy, mapping states to actions, that maximize long-run measure of the reinforcement
- Think of reinforcement as reward
- Can be modeled as MDP model!

zeshan.khan@nu.edu.pk

Genetic algorithm and Evolutionary programming

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    parents ← SELECTION(population, FITNESS-FN)
    population ← REPRODUCTION(parents)
  until some individual is fit enough
  return the best individual in population, according to FITNESS-FN
```

- Start with a set of individuals
- Apply selection and reproduction operators to “evolve” an individual that is successful (measured by a fitness function)

zeshan.khan@nu.edu.pk

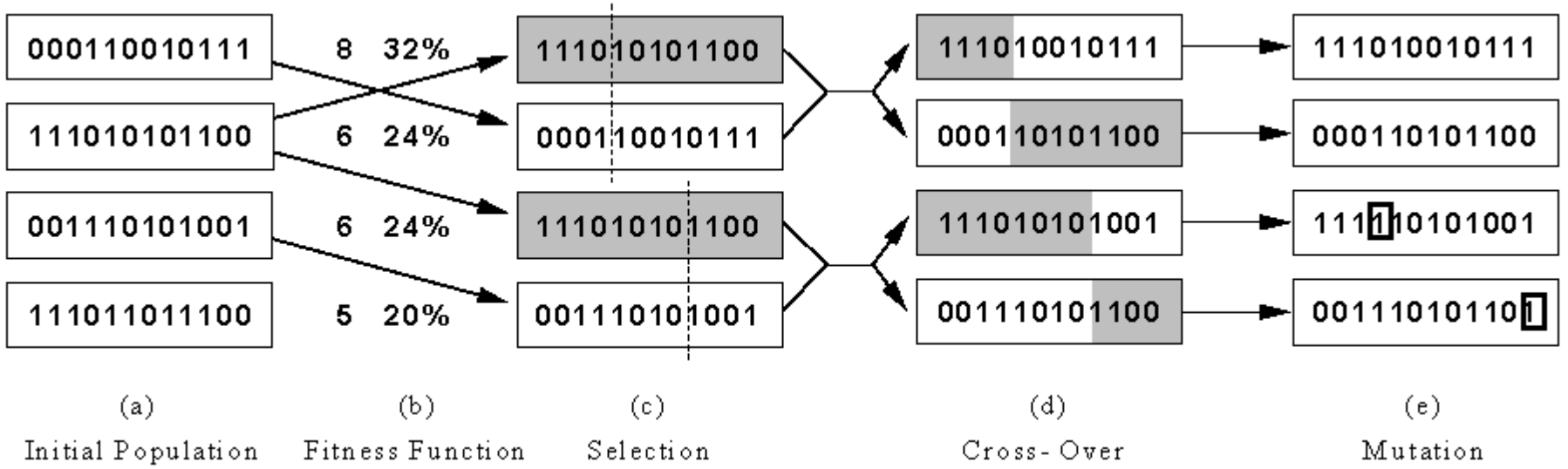
Genetic algorithm and Evolutionary programming

- Imagine the individuals as agent functions
- Fitness function as performance measure or reward function
- No attempt made to learn the relationship the rewards and actions taken by an agent
- Simply searches directly in the individual space to find one that maximizes the fitness functions

zeshan.khan@nu.edu.pk

Genetic algorithm and Evolutionary programming

- Represent an individual as a binary string(each bit of the string is called a gene)
- Selection works like this: if individual X scores twice as high as Y on the fitness function, then X is twice likely to be selected for reproduction than Y is
- Reproduction is accomplished by cross-over and mutation



zeshan.khan@nu.edu.pk

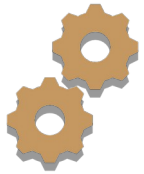
Markov Decision Processes

State space

$$s_t \in \mathcal{S}$$

Action space

$$a_t \in \mathcal{A}$$



Transition function

$$\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$$

$$s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$$

$$s_0 \sim \mathcal{T}_0$$

Reward function

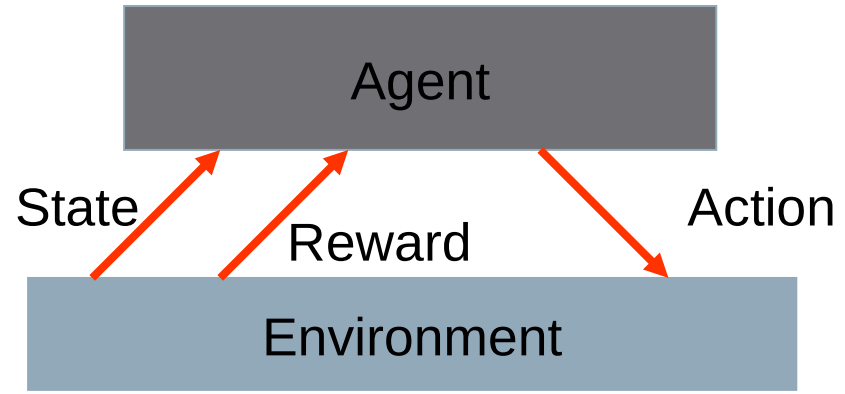
$$\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$$

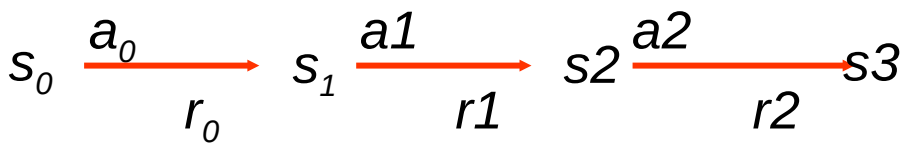
$$r_t \sim \mathcal{R}(s_t, a_t)$$

zeshan.khan@nu.edu.pk

Review of Markov Decision Process (MDP) model

- MDP model <S,T,A,R>





- S– set of states
- A– set of actions
- T(s,a,s') = P(s'|s,a)– the probability of transition from *s* to *s'* given action *a*
- R(s,a)– the expected reward for taking action *a* in state *s*

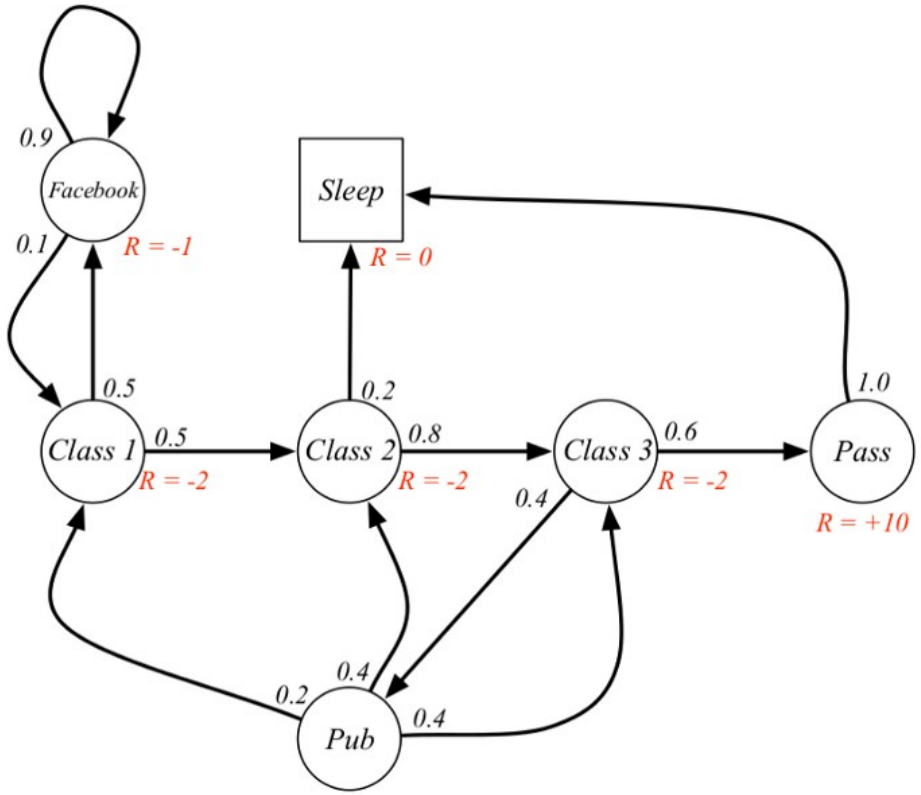
$$R(s,a) = \sum_{s'} P(s'|s,a) r(s,a,s')$$

$$R(s,a) = \sum_{s'} T(s,a,s') r(s,a,s')$$

zeshan.khan@nu.edu.pk

Markov Decision Process

- How the environment is declared
- The goal is to follow the path with the highest sum of rewards
- Low discount factor, immediate reward
- High discount factor, long term reward



$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

zeshan.khan@nu.edu.pk

Discount Factor

- We want to be greedy but not impulsive
- Implicitly takes uncertainty in dynamics into account
- Mathematically: $\gamma<1$ allows infinite horizon returns
- Return

$$G(s_t, a_t) = \sum_{\tau=0}^{T} \gamma^{\tau} \mathcal{R}(s_{t+\tau}, a_{t+\tau})$$

zeshan.khan@nu.edu.pk

Solving an MDP

Objective

$$J(\pi) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t), s_0 \sim \mathcal{T}_0} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t) \right]$$

Goal: $\hat{\pi} = \arg\max_{\pi} J(\pi)$

zeshan.khan@nu.edu.pk

Function Approximation

Model:	$Q_\theta(s_t, a_t)$
Training data:	$\langle s_t, a_t, r_t, s_{t+1} \rangle$
Loss function:	$\mathcal{L}(\theta) = \|\|y_t - Q_\theta(s_t, a_t)\|\|_2^2$
where	$y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$

zeshan.khan@nu.edu.pk

Model based v.s.Model free approaches

- But, we don't know anything about the environment model —the transition function *T(s,a,s')*
- Here comes two approaches
  - *Model based approach RL*:
    learn the model, and use it to derive the optimal policy.
    e.g Adaptive dynamic learning(ADP) approach
  - *Model free approach RL*:
    derive the optimal policy without learning the model.
    e.g LMS and Temporal difference approach
- Which one is better?

zeshan.khan@nu.edu.pk

Passive v.s. Active learning

Passive learning

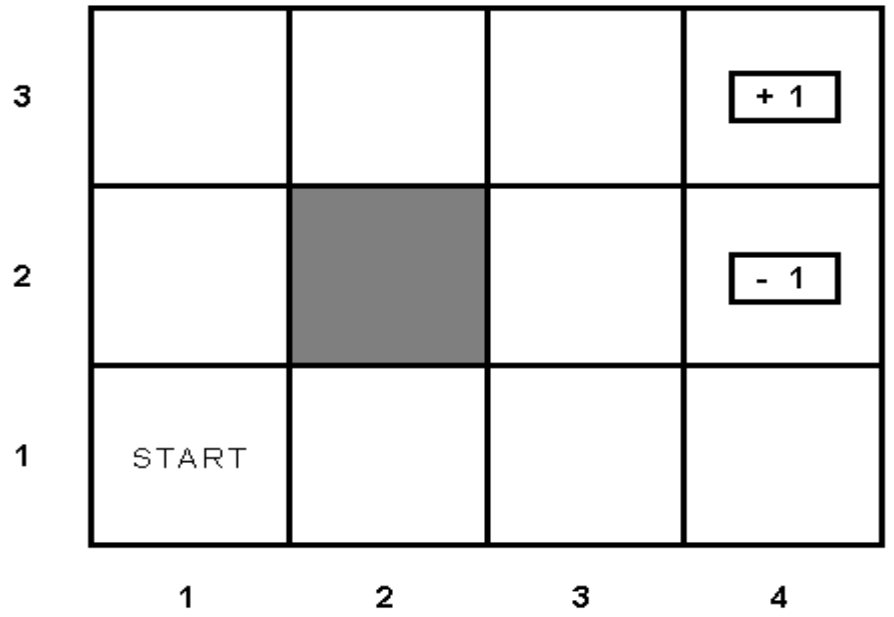
- The agent imply watches the world going by and tries to learn the utilities of being in various states

Active learning

- The agent not simply watches, but also acts

zeshan.khan@nu.edu.pk

Example environment



zeshan.khan@nu.edu.pk

Passive learning scenario

- The agent see the the sequences of state transitions and associate rewards
  - The environment generates state transitions and the agent perceive them

e.g (1,1) ␣(1,2) ␣(1,3) ␣(2,3) ␣(3,3) ␣(4,3)[**+1**]

(1,1)␣(1,2) ␣(1,3) ␣(1,2) ␣(1,3) ␣(1,2) ␣(1,1) ␣(2,1) ␣(3,1) ␣(4,1) ␣(4,2)[**-1**]

- Key idea: updating the utility value using the given training sequences.

zeshan.khan@nu.edu.pk

Passive leaning scenario

```
function PASSIVE-RL-AGENT(e) returns an action
  static: U, a table of utility estimates
          N, a table of frequencies for states
          M, a table of transition probabilities from state to state
          percepts, a percept sequence (initially empty)

  add e to percepts
  increment N[STATE[e]]
  U ← UPDATE(U, e, percepts, M, N)
  if TERMINAL?[e] then percepts ← the empty sequence
  return the action Observe
```

zeshan.khan@nu.edu.pk

LMS updating

- ***Reward to go*** of a state

  the sum of the rewards from that state until a terminal state is reached

- Key: use observed ***reward to go*** of the state as the direct evidence of the actual expected utility of that state

- Learning utility function directly from sequence example

zeshan.khan@nu.edu.pk

LMS updating

function LMS-UPDATE (*U*, *e*, *percepts*, *M*, *N*) **return** an updated *U*

if TERMINAL?[*e*] **then**

{ *reward-to-go* = 0

for each *ei* in percepts (starting from end) do

s = STATE[*ei*]

reward-to-go = *reward-to-go* + REWARS[*ei*]

U[s] = RUNNING-AVERAGE (*U[s]*, *reward-to-go*, *N[s]*)

end

}

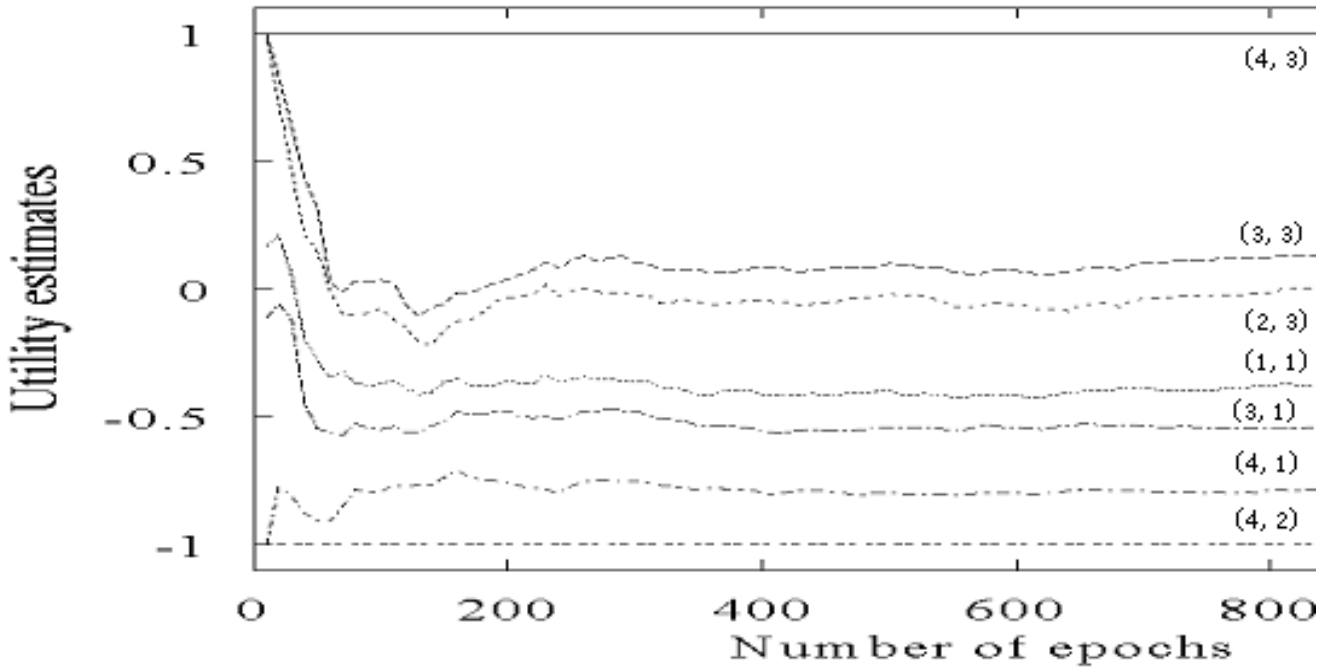
function RUNNING-AVERAGE (*U[s]*, *reward-to-go*, *N[s]*)

U[s] = [*U[s]* * *(N[s]* – *1)* + *reward-to-go]* / *N[s]*

zeshan.khan@nu.edu.pk

LMS updating algorithm in passive learning

- Drawback:
  - The actual utility of a state is constrained to be probability- weighted average of its successor's utilities.
  - Converge very slowly to correct utilities values (requires a lot of sequences)
    - *for our example, >1000!*



zeshan.khan@nu.edu.pk

Temporal difference method in passive learning

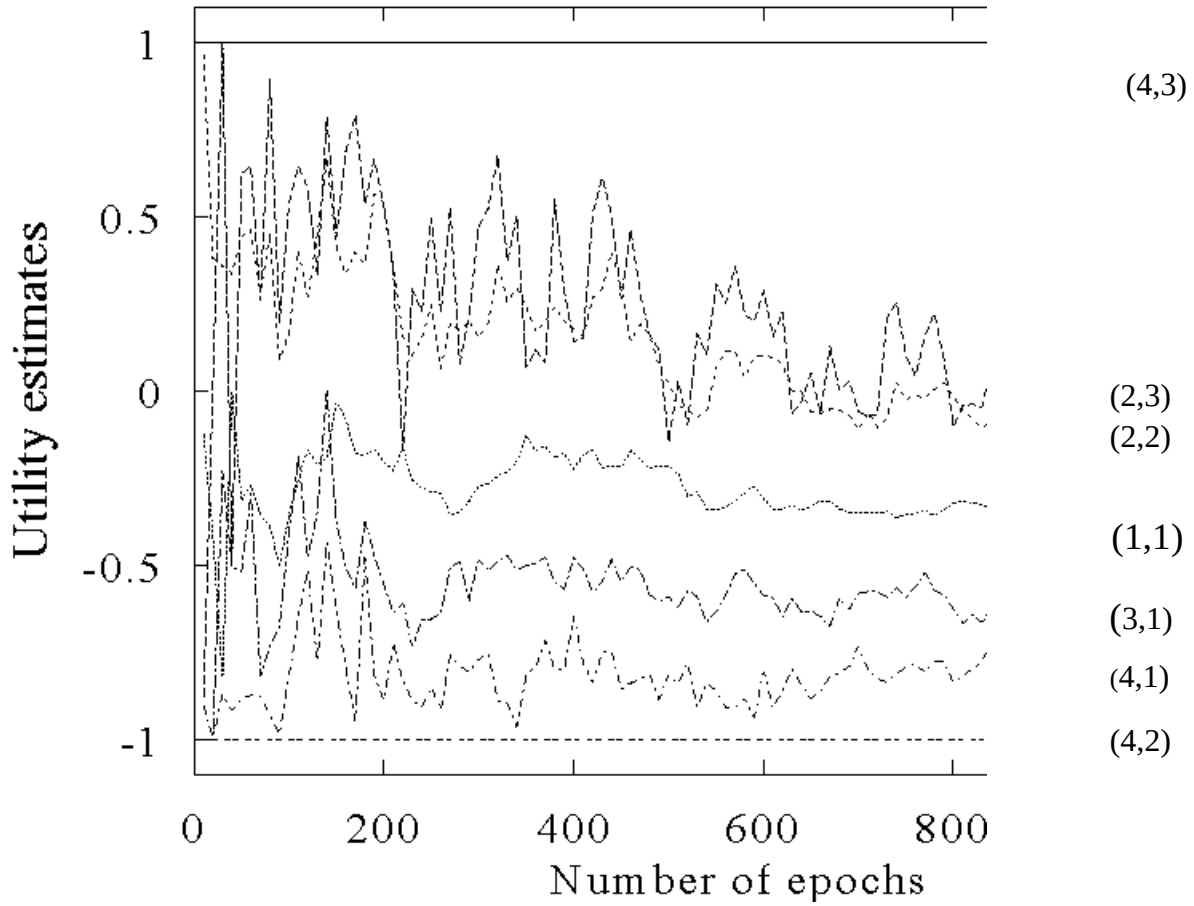
- TD(0) key idea:
  - adjust the estimated utility value of the current state based on its immediately reward and the estimated value of the next state.
- The updating rule

$$U(s) = U(s) + \alpha(R(s) + U(s') - U(s))$$

  - α is the learning rate parameter
  - Only when α is a function that decreases as the number of times a state has been visited increased, then can *U(s)* converge to the correct value.

zeshan.khan@nu.edu.pk

The TD learning curve



zeshan.khan@nu.edu.pk

Adaptive dynamic programming(ADP) in passive learning

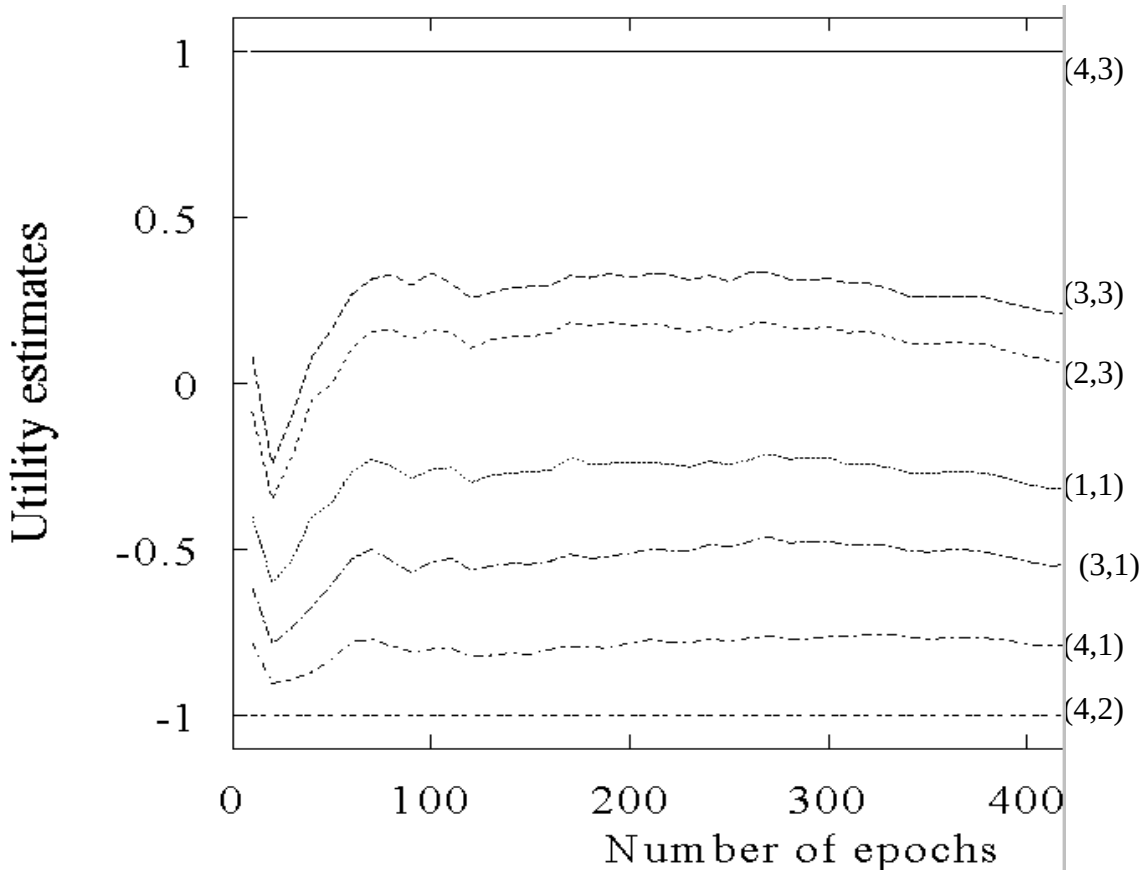
- Different with LMS and TD method(model free approaches)
- ADP is a model based approach!
- The updating rule for passive learning

$$U(s) = \sum_{s'} T(s,s')(r(s,s') + \gamma U(s'))$$

- However, in an unknown environment, T is not given, the agent must learn T itself by experiences with the environment.
- How to learn T?

zeshan.khan@nu.edu.pk

ADP learning curves



zeshan.khan@nu.edu.pk

Active learning

- An active agent must consider
  - what actions to take?
  - what their outcomes maybe(both on learning and receiving the rewards in the long run)?
- Update utility equation

$$U(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') U(s') \right)$$

- Rule to chose action

$$a = \arg\max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') U(s') \right)$$

zeshan.khan@nu.edu.pk

Active ADP algorithm

For each s, initialize *U(s)* , *T(s,a,s')* and *R(s,a)*

Initialize *s* to current state that is perceived

Loop forever

{

Select an action ***a*** and execute it (using current model ***R*** and ***T***) using

$$a = \arg\max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s')U(s')\right)$$

Receive immediate reward ***r*** and observe the new state ***s'***

Using the transition tuple <*s,a,s',r*> to update model ***R*** and ***T*** (see further)

For all the sate ***s***, update ***U(s)*** using the Bellman update rule

$$U(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s')U(s')\right)$$

s = s'

}

zeshan.khan@nu.edu.pk

How to learn model?

- Use the transition tuple <s, a, s', r> to learn T(s,a,s') and R(s,a). That's supervised learning!
  - Since the agent can get every transition (s, a, s',r) directly, so take (s,a)/s' as an input/output example of the transition probability function T.
  - Different techniques in the supervised learning(see further reading for detail)
  - Use r and $T(s,a,s')$ to learn $R(s,a)$

$$R(s,a) = \sum_{s'} T(s,a,s')r$$

zeshan.khan@nu.edu.pk

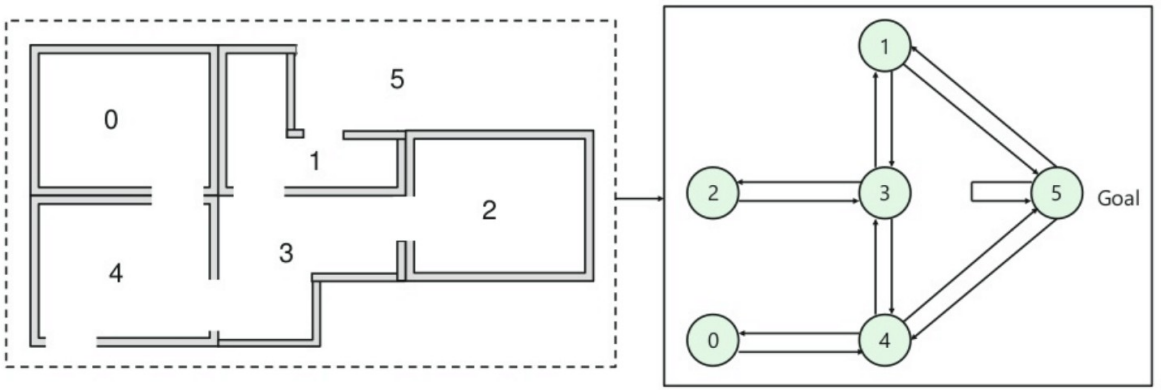
ADP approach pros and cons

- Pros:
  - ADP algorithm converges far faster than LMS and Temporal learning. That is because it use the information from the the model of the environment.
- Cons:
  - Intractable for large state space
  - In each step, update U for all states
  - Improve this by *prioritized-sweeping* (see further reading for detail)

zeshan.khan@nu.edu.pk

Q-learning

- Value-based method of supplying information to inform which action an agent should take.
- 'Q' means quality which represents the gain of a future reward given a situation.
- Seeks the best course of action to ta and maximizes the total reward.



zeshan.khan@nu.edu.pk

Q Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma Q(s_{t+1}, \pi(s_{t+1})) - Q(s_t, a_t) \right)$$

$$\pi(s) \leftarrow \arg\max_a Q(s, a)$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$;
  until S is terminal

zeshan.khan@nu.edu.pk

Model free method– TD-Q learning

- Define Q-value function

- Q-value function updating rule

- Key idea of TD-Q learning
  - Combined with temporal difference approach
  - The updating rule

- Rule to chose the action to take

zeshan.khan@nu.edu.pk

TD-Q learning agent algorithm

For each pair (s, a), initialize $Q(s,a)$

Observe the current state s

Loop forever {

Select an action ***a*** and execute it

Receive immediate reward ***r*** and observe the new state ***s'***

Update

}

zeshan.khan@nu.edu.pk

Exploration problem in Active learning

- An action has two kinds of outcome
  - Gain rewards on the current experience tuple (s,a,s')
  - Affect the percepts received, and hence the ability of the agent to learn

zeshan.khan@nu.edu.pk

Exploration problem in Active learning

- A trade off when choosing action between
  - its immediately good(reflected in its current utility estimates using the what we have learned)
  - its long term good(exploring more about the environment help it to behave optimally in the long run)
- Two extreme approaches
  - “wacky”approach: acts randomly, in the hope that it will eventually explore the entire environment.
  - “greedy”approach: acts to maximize its utility using current model estimate

  See Figure 20.10
- Just like human in the real world! People need to decide between
  - Continuing in a comfortable existence
  - Or striking out into the unknown in the hopes of discovering a new and better life

zeshan.khan@nu.edu.pk

Exploration problem in Active learning

- One kind of solution: the agent should be more wacky when it has little idea of the environment, and more greedy when it has a model that is close to being correct
  - In a given state, the agent should give some weight to actions that it has not tried very often.
  - While tend to avoid actions that are believed to be of low utility

- Implemented by ***exploration function f(u,n)***:
  - assigning a higher utility estimate to relatively unexplored action state pairs
  - Chang the updating rule of value function to

$$U^+(s) = \max_a (r(s,a) + \gamma f(\sum_{s'} T(s,a,s')U^+(s'), N(a,s))$$

  - $U+$ denote the **optimistic estimate** of the utility

zeshan.khan@nu.edu.pk

Exploration problem in Active learning

- One kind of definition of *f(u,n)*

  - is an **optimistic estimate** of the **best possible reward** obtainable in any state
  - The agent will try each action-state pair(*s,a*) at least *Ne* times
  - The agent will behave initially as if there were wonderful rewards scattered all over around– **optimistic** .

zeshan.khan@nu.edu.pk

Generalization in Reinforcement Learning

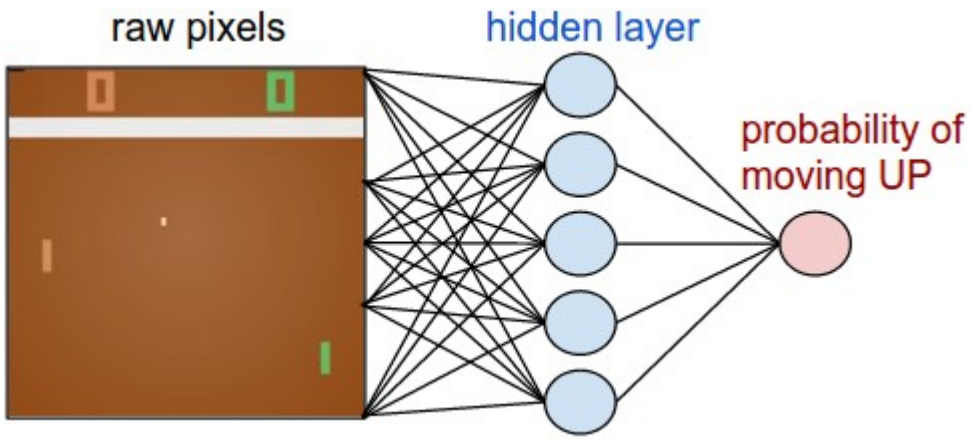
- So far we assumed that all the functions learned by the agent are *(U, T, R,Q)* are tabular forms—

  i.e.. It is possible to enumerate state and action spaces.

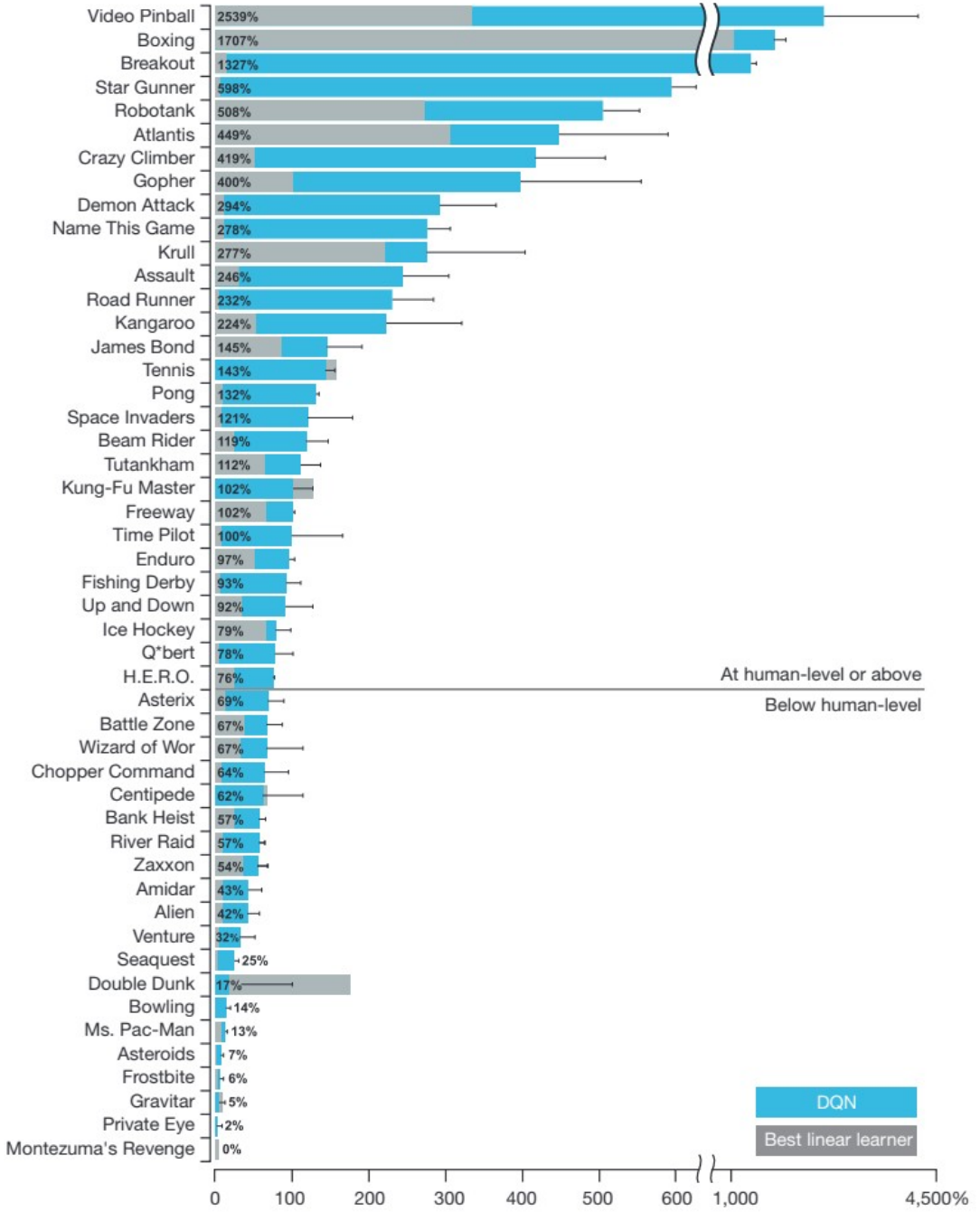
- Use generalization techniques to deal with large state or action space.
  - Function approximation techniques

zeshan.khan@nu.edu.pk

Example: Atari Games



zeshan.khan@nu.edu.pk



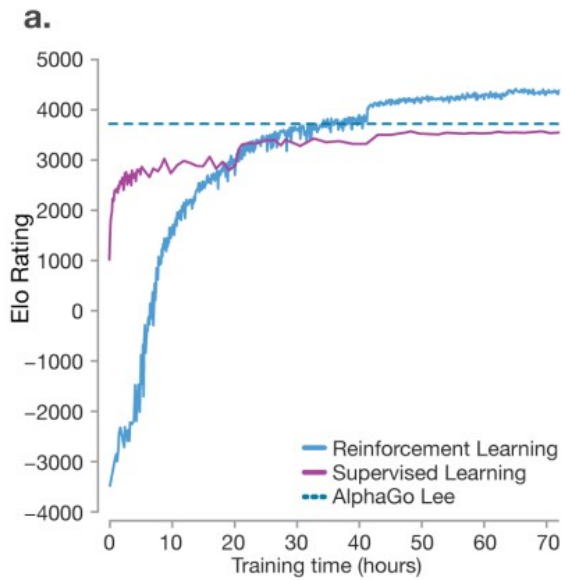
State: Raw Pixels
Actions: Valid Moves
Reward: Game Score

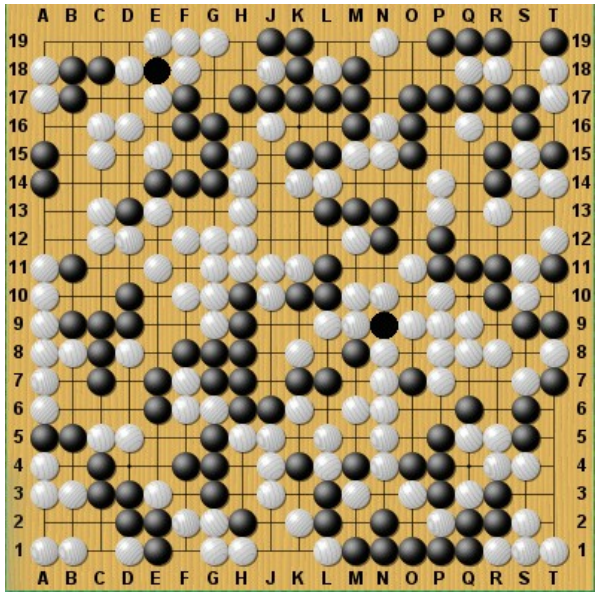
zeshan.khan@nu.edu.pk

Alpha Go

- Learning how to beat humans at ‘hard’ games (search space too big)
- Far surpasses (Human) Supervised learning
- Algorithm learned to outplay humans at chess in 24 hours

State: Board State
Actions: Valid Moves
Reward: Win or Lose

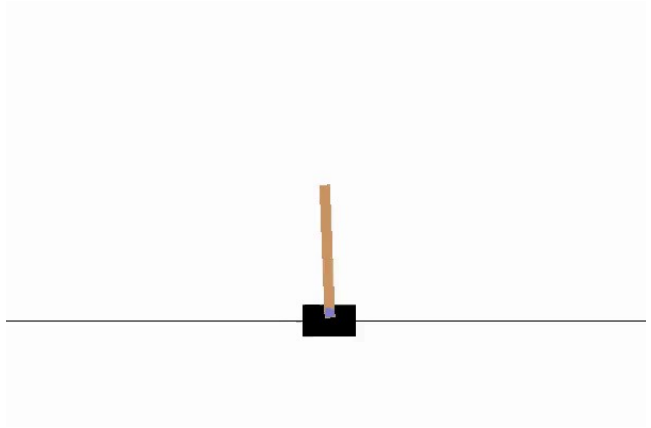




zeshan.khan@nu.edu.pk

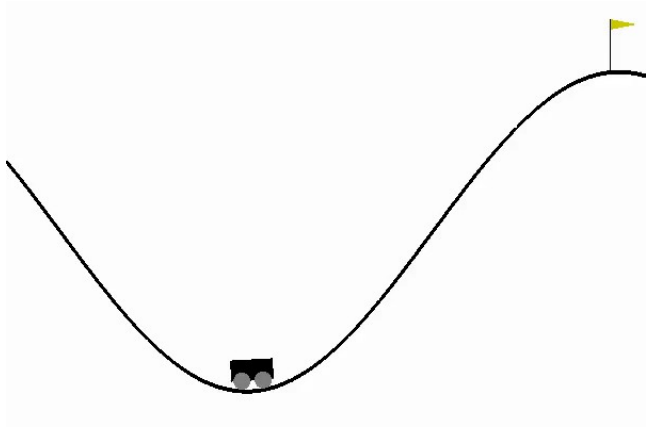
Gym – toolkit for reinforcement learning

CartPole



Reward +1 per step the pole remains up

MountainCar



Reward 200 at flag -1 per step

```
import gym

env = gym.make('CartPole-v0')

import random
import QLearning # Your implementation goes here...
import GymSupport

trainingIterations = 20000

qlearner = QLearning.QLearning(<Parameters>)

for trialNumber in range(trainingIterations):
    observation = env.reset()
    reward = 0
    for i in range(300):
        env.render() # Comment out to make much faster...

        currentState = ObservationToStateSpace(observation)
        action = qlearner.GetAction(currentState, <Parameters>)

        oldState = ObservationToStateSpace(observation) # Old state =
current
        observation, reward, isDone, info = env.step(action)
        newState = ObservationToStateSpace(observation)

        qlearner.ObserveAction(oldState, action, newState, reward, …)

        if isDone:
            if(trialNumber%1000) == 0:
                print(trialNumber, i, reward)
            break

# Now you have a policy in qlearner – use it...
```

zeshan.khan@nu.edu.pk

https://gym.openai.com/docs

Summary

Reinforcement Learning:
- Goal: Maximize
- Data:

Many (awesome) recent successes:
- Robotics
- Surpassing humans at difficult games
- Doing it with (essentially) zero human knowledge

Challenges:
- When the episode can end without reward
- When there is a 'narrow' path to reward
- When there are many states and actions

(Simple) Approaches:
- Q-Learning -> discounted reward of action
- Policy Gradients -> Probability distribution over
- Reward Shaping
- Memory
- Lots of parameter tweaking…

zeshan.khan@nu.edu.pk