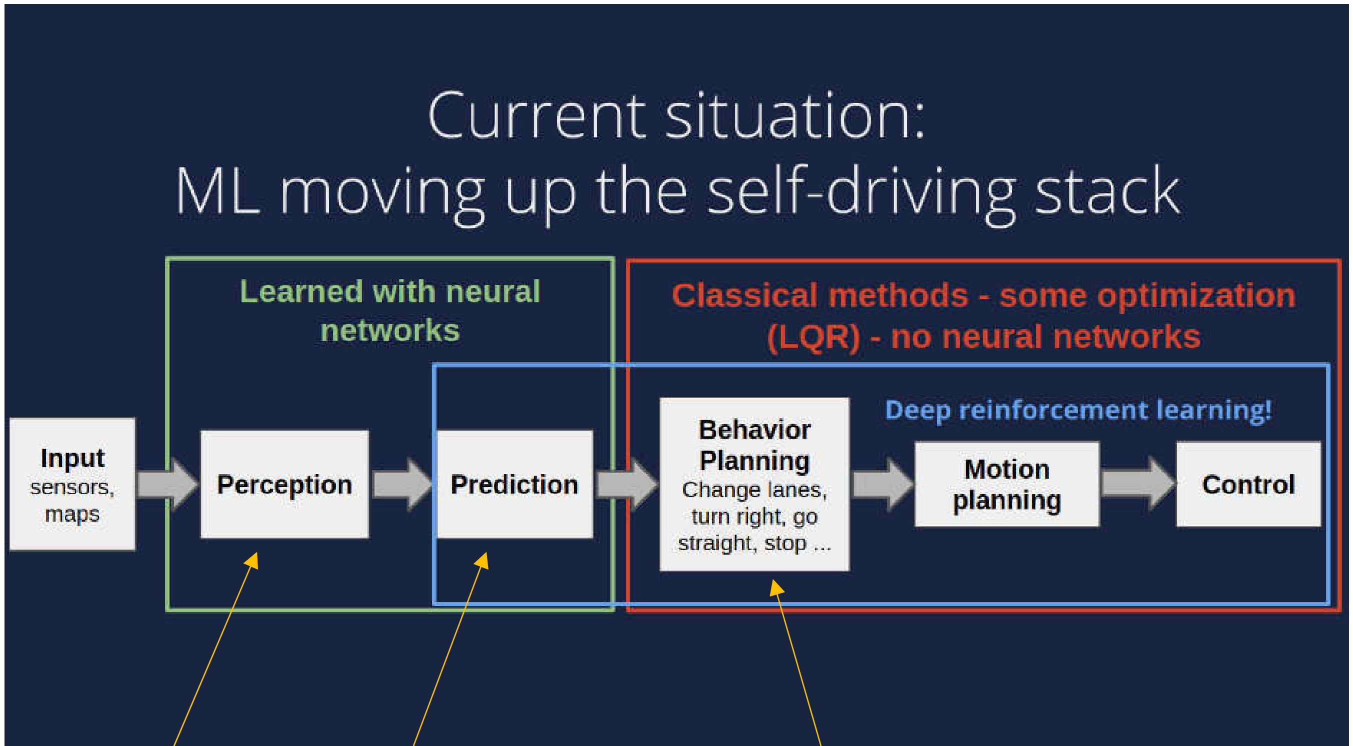
Reinforcement Learning for Autonomous Driving

Cheng-Zhong Xu
FST & IoTSC
czxu@um.edu.mo

Outline

- Overview of RL
- Markov Decision Process
  - Bellman Optimality Equation
  - Value Iteration
  - Policy Iteration
- RL for unknown environment
  - Q-learning
- Deep RL for Large State Space
  - DQN algorithm

Machine Learning for Autonomous Driving



Convolutional Neural Networks for detection and classification

Recurrent Neural Networks for vehicle trajectory prediction

Reinforcement Learning for Vehicle Maneuvers and Path Planning

Types of Learning

- Supervised learning: CNN and RNN
  - Learning from a “teacher”
  - Training data includes desired outputs
- Unsupervised learning
  - Discover structure in data
  - Training data does not include desired outputs
- Reinforcement learning
  - Learning to act under evaluative feedback (rewards)

Supervised LearningLearning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map
x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

→ Cat

Classification

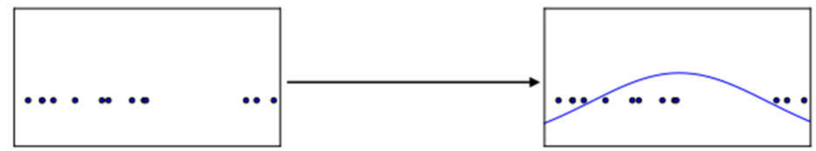
Unsupervised Learning Learning

Data: x

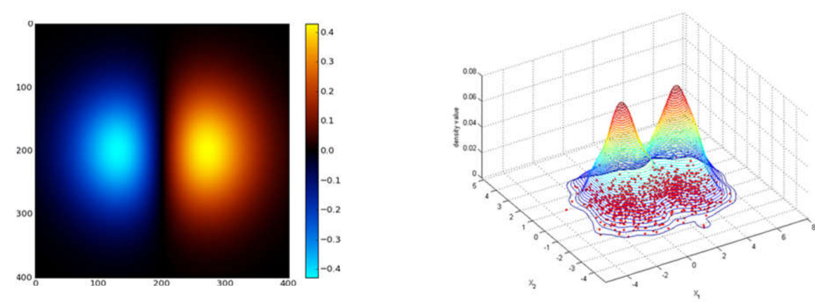
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



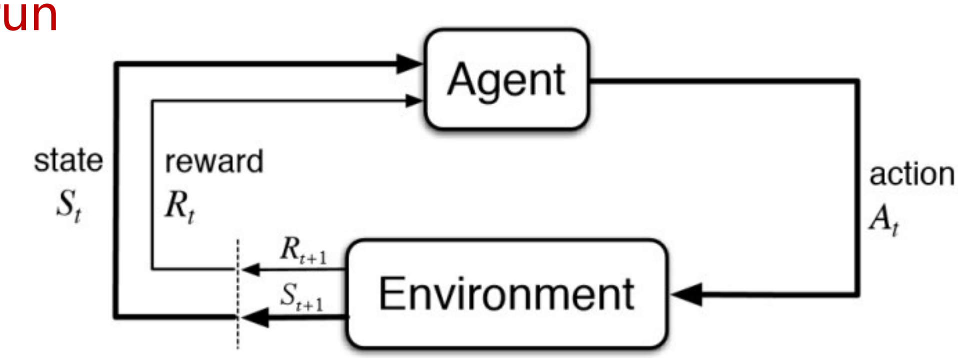
1-d density estimation



2-d density estimation

Reinforcement Learning

- Learning by interacting with environment to make good sequences of decisions under uncertainty
  - Environment may be unknown, nonlinear, stochastic and complex
  - Agent learns a policy mapping states to actions
    - learning by trial and error
  - Goal is to take actions so as to maximize expected but delayed cumulative reward in the long run



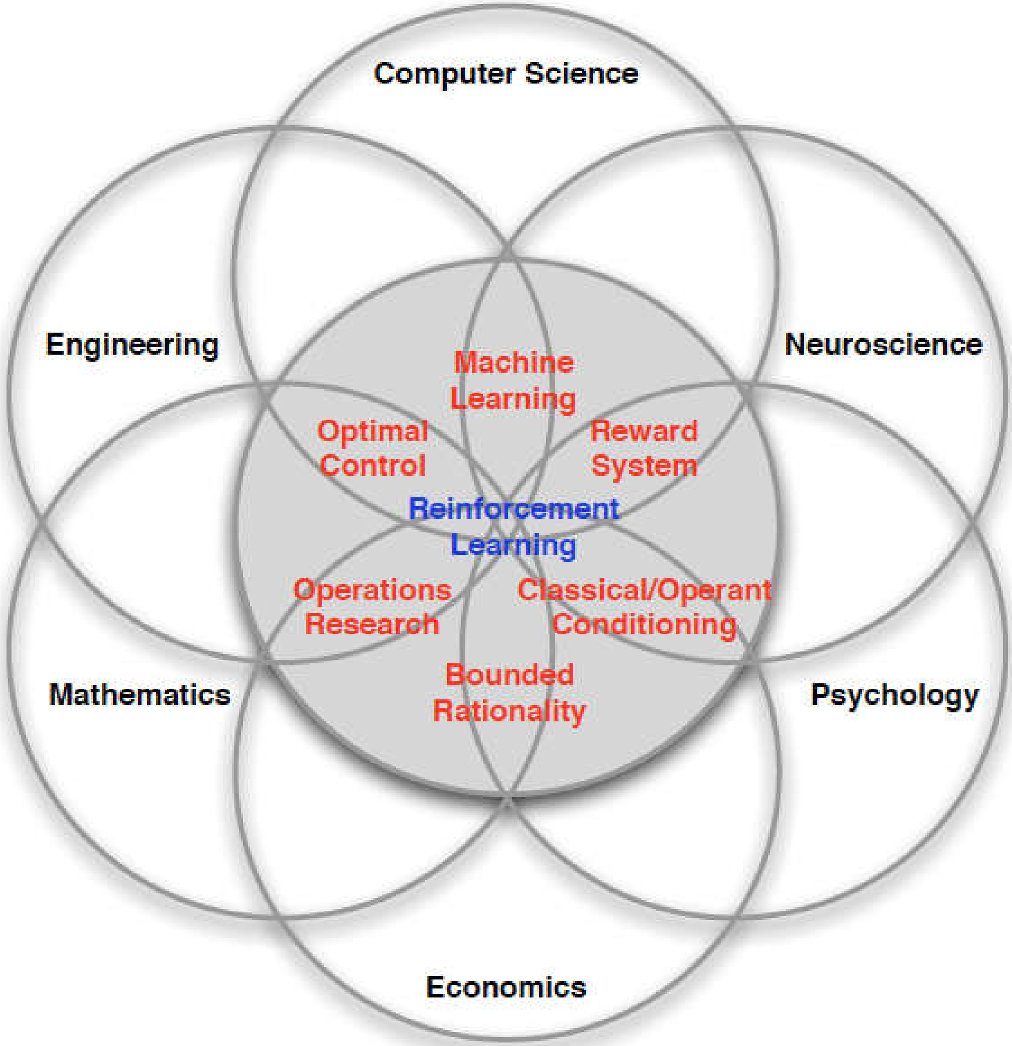
- Examples
  - Self-driving: accelerate, decelerate, turn left/right, etc
  - NLP: summarization, question answer, translation, etc
  - Healthcare: patient treatment
  - Recommendation system to track the change of user behaviors
  - Resource management in cloud datacenters

Rao, et al, VCONF: a reinforcement learning approach to virtual machines auto-configuration, ICAC 2009

Elements of RL

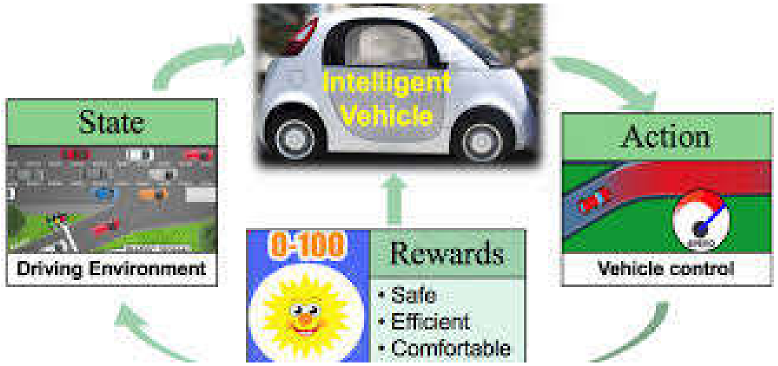
- Agent state:
  - fully observable env (e.g. chess)
  - Partially observable env, indirectly observes env (e.g. porker)
    - Beliefs of env state
  - Discrete vs Continuous
- Major components:
  - Policy is about agent's behavior, concerning about how an agent should behave
  - Value function in state s and action a is prediction of future award. It is about how good is each state and/or state-action pair
    - Optimal state-action value function → optimal actions
  - Model : predict what env will do next
    - Transition Probability
    - Reward R to predict the next (immediate) reward
- Environment is unknown in reality
  - Interact with environment
  - Agent improves policy

Inter-disciplinary Studies



RL for Autonomous Driving

- RL for Behavior Planning in self-driving
  - Agent observes state env (How to represent a state?)
  - Agent takes an action to achieve a benefit (stop, left, right, accelerate, decelerate, etc)
  - Agent receives a reward based on the results of that action (positive or negative)
  - RL enables the agent to learn the optimal behavior that will maximize the reward



- Why RL matters…
  - Range of maneuvers: it can learn an extremely large number of roadway maneuvers
  - Complexity of maneuvers: it can learn complex maneuvers involving many contextual parameters
  - Decision: it can make decisions akin to a human driver

DeepTraffic: Example of RL for Highway Driving

See https://github.com/lexfridman/deeptraffic for online car racing

Markov Decision Process

- **Almost all RL problems can be formulated as MDP**
- Markov property
  - The future is independent of the past, given the present
- Defined as 5 tuple <S, A, R, P, r>
  - S: set of possible states [start state = $s_{0,}$ optional terminal / absorbing state]
  - A: set of possible action
  - R : immediate reward given (state, action, next state) tuple
  - P : transition probability distribution from one state to another
  - r: discount factor to reflect the uncertainty about the future

    (r=0: short sight; r=1 far sight)
- Policy is defined as a map from state to action
  - Deterministic policy: $\pi(s) = a$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$
- What is a good policy?

  Maximizes current reward? Sum of all future reward?

  Discounted future rewards!
- Optimal policy

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$$

Markov Decision Process (cont')

- State Value Function **(Value Function)**
  - How good is a state? Am I screwed or winning this game?
  - Value function of state s under policy π

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

- Action-Value Function **(Q-function)**
  - How good is a state action-pair? Should I do this now?
  - State action value func of state s, action a, under policy pi

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- Optimal Q-value function is the expected cumulative reward from taking action a in state s and acting optimally thereafter

$$Q^{*}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi^*\right]$$

Bellman Optimality Equations

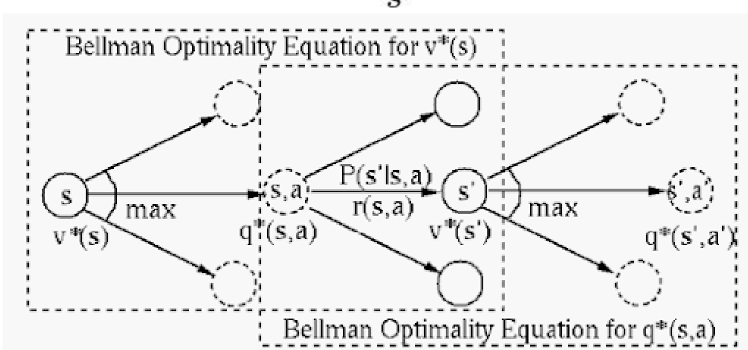
- Extracting optimal value / policy from Q-values:

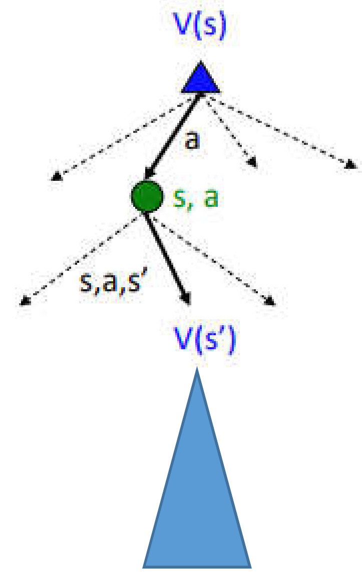
$$V^*(s) = \max_a Q^*(s,a) \qquad\qquad \pi^*(s) = \arg\max_a Q^*(s,a)$$

- Recursive optimality equations:

$$V^*(s) = \max_a \sum_{s'} p\left(s'|s,a\right)\left[r(s,a) + \gamma V^*\left(s'\right)\right]$$

$$\begin{aligned} Q^*(s,a) &= \mathop{\mathbb{E}}_{s' \sim p(s'|s,a)}\left[r\left(s,a\right) + \gamma V^*(s)\right] \\ &= \sum_{s'} p\left(s'|s,a\right)\left[r\left(s,a\right) + \gamma V^*(s)\right] \\ &= \sum_{s'} p\left(s'|s,a\right)\left[r\left(s,a\right) + \gamma \max_a Q^*(s',a')\right] \end{aligned}$$





Value Iteration

- Based on the Bellman Optimality Equation

$$V^*(s) = \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^*(s')\right]$$

- VI Algorithm
  - Initialize values of all states $V^0(s) = 0$
  - While not converged:
    - For each state:

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^i(s')\right]$$

  - Repeat until convergence (no change in values)

$$V^0 \rightarrow V^1 \rightarrow V^2 \rightarrow \quad \cdots \rightarrow V^i \rightarrow \ldots \quad \rightarrow V^*$$

- Time complexity per iteration $O(|S|^2|A|)$

Q-Value Iteration

- Value Iteration Update:

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a) \left[r(s,a) + \gamma V^i(s') \right]$$

- Q-Value Iteration Update:

$$Q^{i+1}(s,a) \leftarrow \sum_{s'} p\left(s'|s,a\right) \left[r\left(s,a\right) + \gamma \max_{a'} Q^i(s',a') \right]$$

Same algorithm as value iteration, but it loops over actions as well as states

Policy Iteration

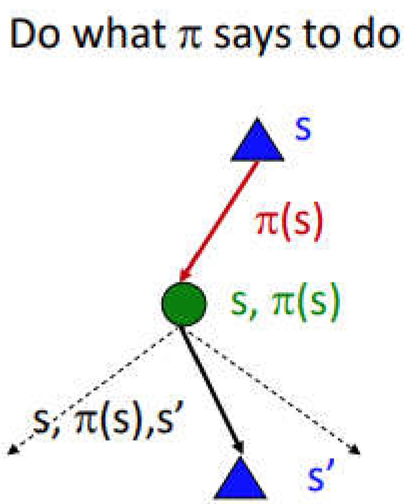
- Policy iteration: Start with arbitrary π_0 and refine it.

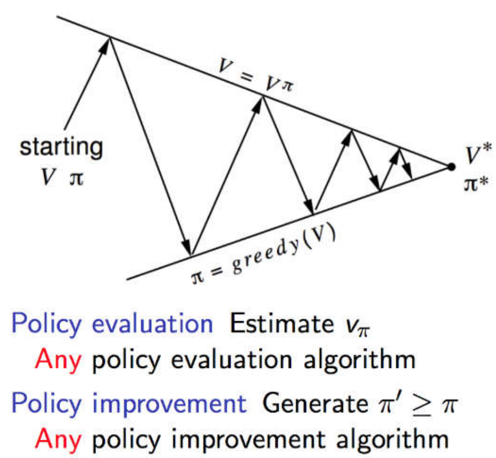
$$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \quad \ldots \quad \rightarrow \pi^*$$

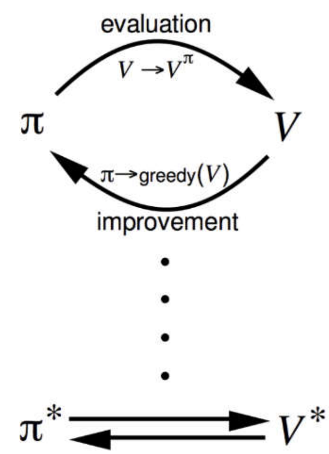
- Involves repeating two steps:
  - Policy Evaluation: Compute V^π (similar to Value Iteration)
  - Policy Refinement: Greedily change actions as per V^π

    $\pi'(s) = \text{argmax}_a V^\pi(s)$

$$\pi_0 \longrightarrow V^{\pi_0} \longrightarrow \pi_1 \longrightarrow V^{\pi_1} \longrightarrow \quad \ldots \quad \longrightarrow \pi^* \longrightarrow V^{\pi^*}$$

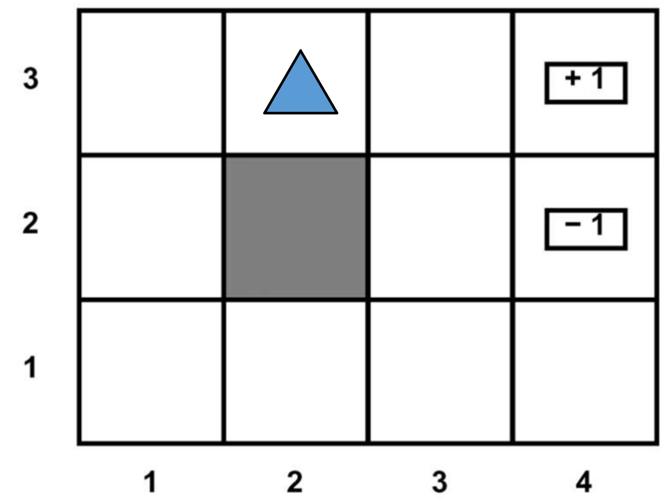






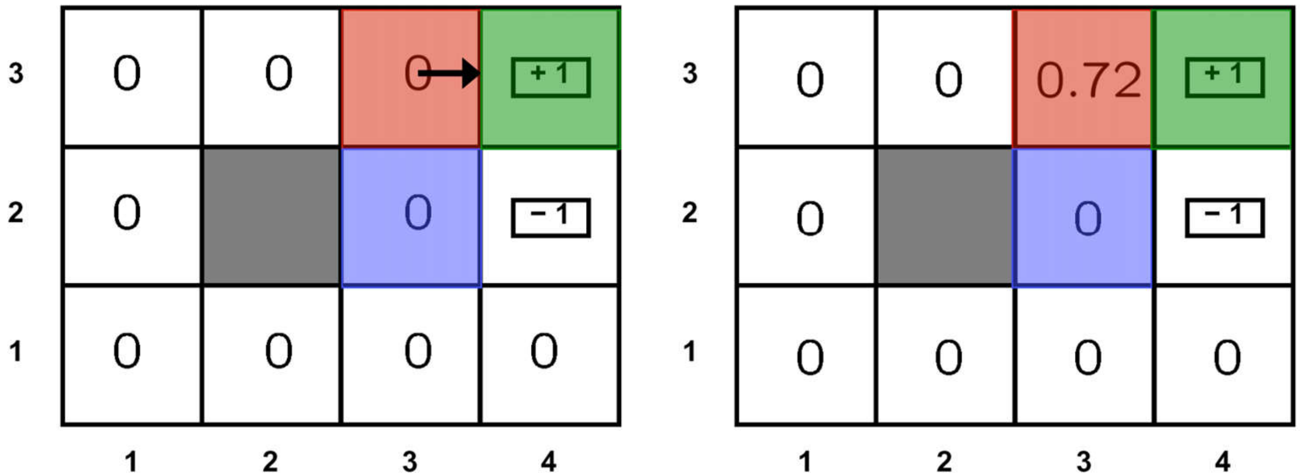
Canonical Example: Grid World

- Agent lives in a grid
- Walls block the agent's path
- Actions do not always go as planned
  - 80% of the time, go to desired direction
  - 10% of the time, West of desired direction; 10% East of desired action
  - If there is a wall, the agent stays put



- State: Agent's location
- Actions: N, E, S, W
- Rewards: +1 / -1 at absorbing states

Value Iteration (VI)



Discount r=0.9

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} p\left(s'|s,a\right)\left[r(s,a) + \gamma V^i\left(s'\right)\right]$$

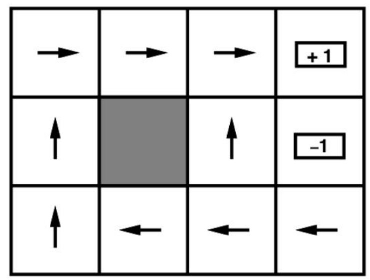
$$V^2(\langle 3,3\rangle) = \sum_{s'} P\left(s'|\,\text{right}, \langle 3,3\rangle\right)\left[r(\langle 3,3\rangle) + \gamma V^1\left(s'\right)\right]$$

$$= 0.9[0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$$

The optimal policy **π***

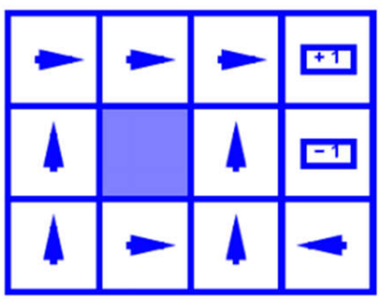
- Computing Actions from Q values
  - Easy job than value function

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

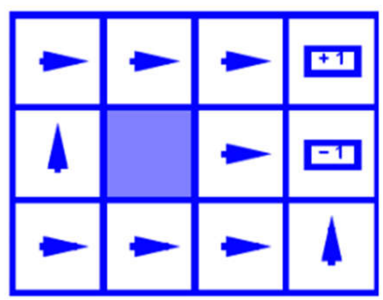


r(s) = -0.04

- Policy is dependent on r(s) as well



r(s) = 0.4



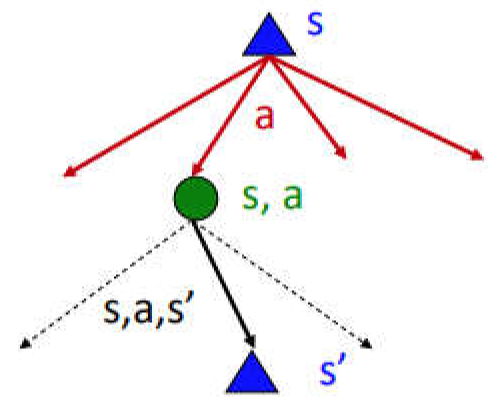
r(s) = 2.0

Limitations of MDP

- Typically, we don't know the environment
  - Transition probability p(s'|s,a) is unknown, how actions affect the environment?
  - Immediate reward function r(s, a, s') is unknown, what/when are the good actions?
- But, we can learn by trial and error
  - Gather experience (data) by performing actions.

$$\{s, a, s', r\}_{i=1}^{N}$$

  - Approximate unknown quantities from data.



Reinforcement Learning

Deep Learning Based Methods

- In addition to not knowing the environment, sometimes the state space is too large.

- A value iteration updates takes $O(|S|^2|A|)$
  - Not scalable to high dimensional states e.g.: RGB images.

- Solution: Deep Learning!
  - Use deep neural networks to learn low-dimensional representations.

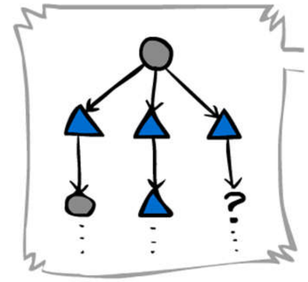
Deep Reinforcement Learning

Outline

- Overview of RL
- Markov Decision Process
  - Bellman Optimality Equation
  - Value Iteration
  - Policy Iteration
- RL for unknown environment
  - Q-learning
- Deep RL for Large State Space
  - DQN algorithm

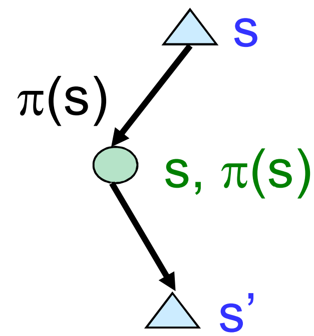
Reinforcement Learning

- We want to evaluate V, without known transition probability P and reward function R



- Sample-based Monte-Carlo

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V_k^{\pi}(s'_n)$$
$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i sample_i$$



- Temporal Difference (for values, not policy)
  - learn from every experience
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + (\alpha) sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Q-Learning

- SARSA: sample-based Q-value iter

$$Q(a,s) \leftarrow Q(a,s) + \alpha \cdot \left(r_s + \gamma \cdot Q(a',s') - Q(a,s)\right)$$

- Q-Learning: sample-based

$$Q(a,s) \leftarrow Q(a,s) + \alpha \cdot \left(r_s + \gamma \max_{a'} Q(a',s') - Q(a,s)\right)$$

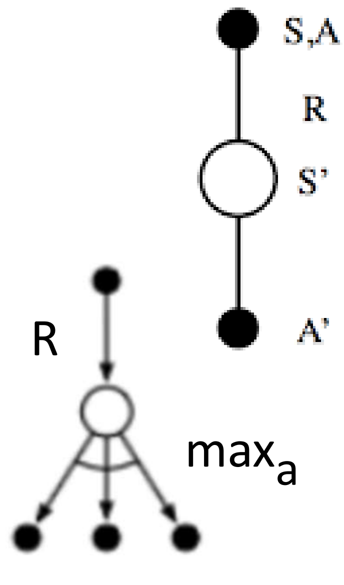
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate Q(s,a)
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate new estimate into a running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

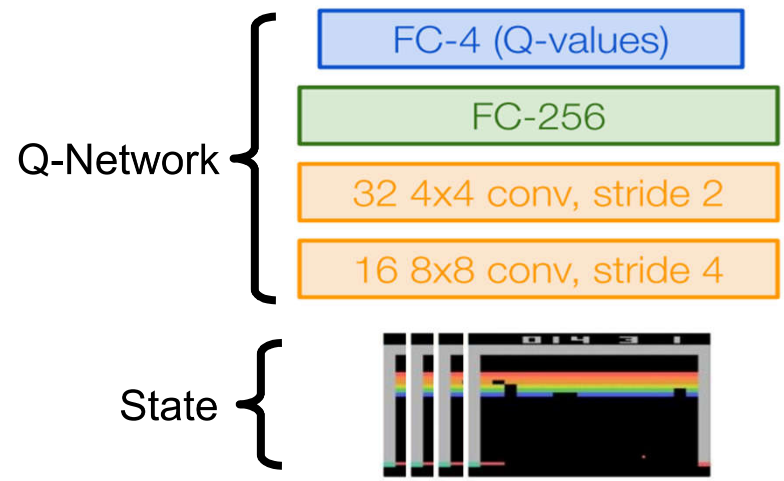
- Off policy learning: update policy is different behavior policy

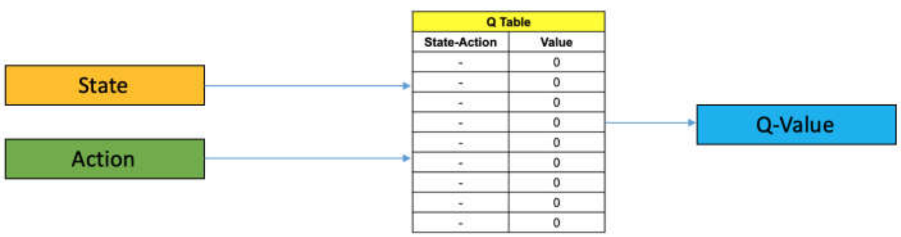


Q-learning

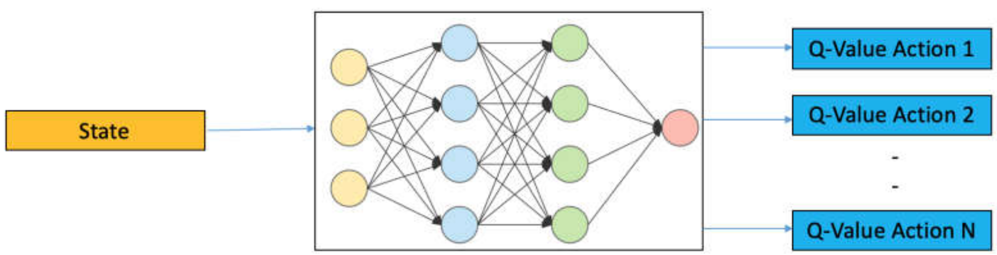
Deep Q Learning

- Q table Q(s,a) is too big for large problems
- Use NN to approximate Q table





Q Learning



Deep Q Learning

DQN Learning
(Mnih, et al, Playing Atari with Deep Reinforcement Learning, 2013)

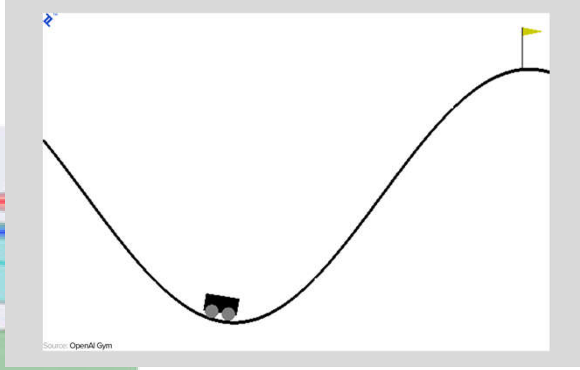
- Approximation of NN for Q(s, a)
  - Define Loss Function for Mean Square Error of a single data point

$$\text{MSE Loss} := \Big(\underbrace{Q_{new}(s,a)}_{\text{Predicted Q-Value}} - \underbrace{(r + \gamma \max_a Q_{old}(s',a))}_{\text{Target Q-Value}}\Big)^2$$

  - Gradient Descent is to minimize its loss function $\frac{\partial Loss}{\partial \theta_{new}}$
    - Calculation over the full training set at each step
    - Computational expensive
  - Stochastic Gradient Descent to use a "random" set of data for gradient calculation: minibatch $\{(s,a,s',r)_i\}_{i=1}^{B}$

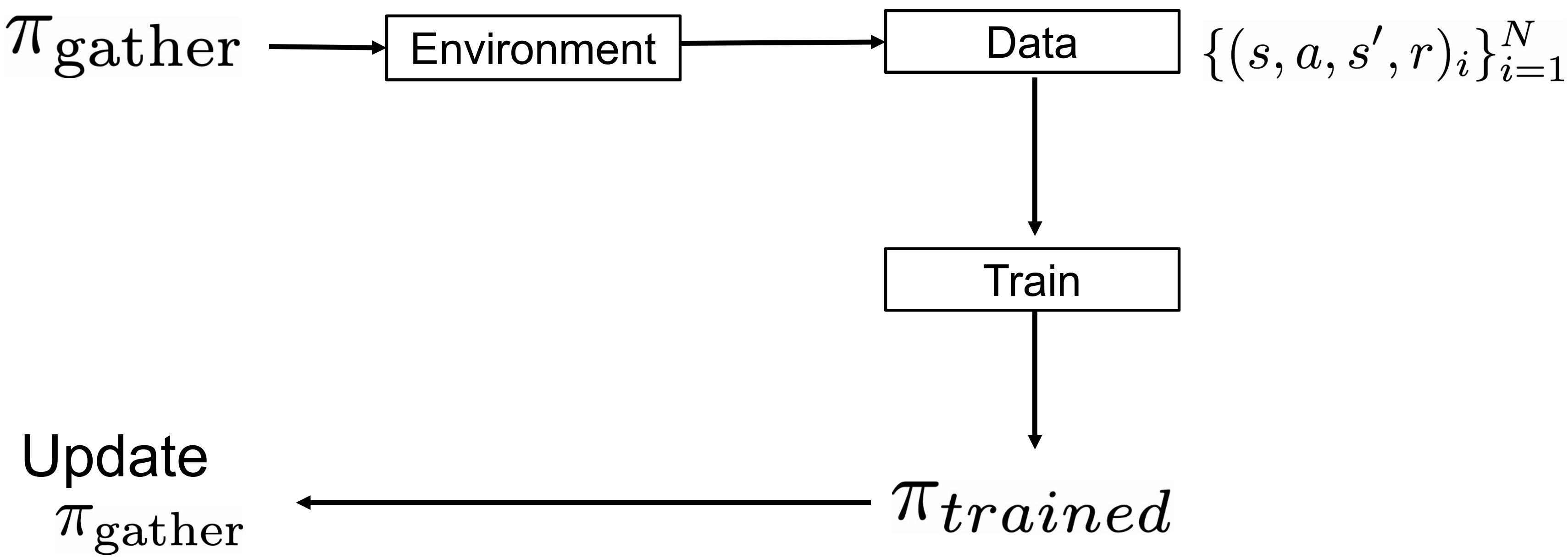
Performance of DQN vs Q Learning

- Mountain Car



Ohnishi, et al, Constrained Deep Q-Learning Gradually Approaching Ordinary Q-Learning, Neurorobot, Dec 2019

How To Gather Experience?



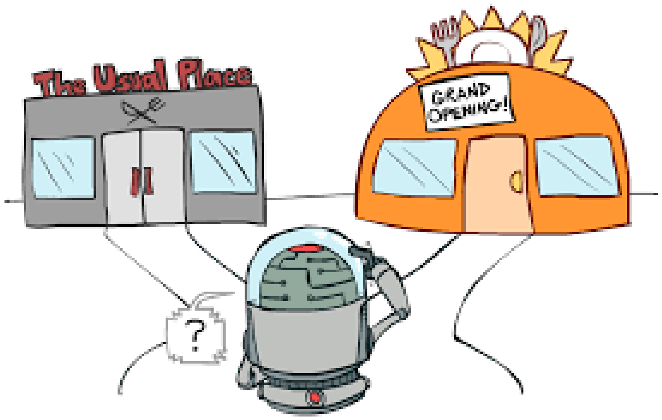
Challenge 1: Exploration vs Exploitation

Challenge 2: Non i.i.d, highly correlated data

Exploration and Exploitation

- What should π_{gather} be?
  - Greedy? → Exploit local minimal,

$$\arg\max_a Q(s, a; \theta)$$



Eat in “best” restaurant (exploitation)
Explore “new” restaurant (exploration)

- An exploration strategy:
  - ϵ-greedy

$$a_t = \begin{cases} \arg\max_a Q(s, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

Correlated Data Problem

- Samples are correlated, not i.i.d

  => high variance gradients

  => inefficient learning

- Experience Replay
  - A replay buffer stores transitions
  - Continually update replay buffer as game (experience) episodes are played, older samples discarded
  - Train Q-network on random mini-batches of transitions from the replay memory, instead of consecutive samples

Putting together: DQN Algorithm

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

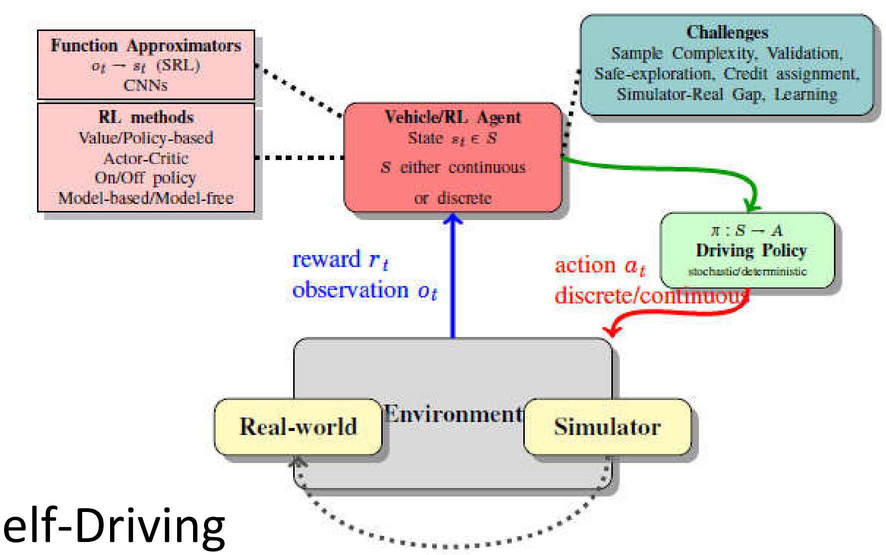
Experience Replay

Epsilon-greedy

Q Update

In Summary: RL for Self-Driving

- Value-based RL
  - (Deep) Q-Learning, approximating Q*(s,a), with a deep Q-network
- Policy-based RL
  - Directly approximate optimal policy π^*_θ with a parametrized policy π^*
- Model-based RL
  - Approximate transition function p(s, a, s') and reward function r(s,a)
  - Plan by looking ahead in the (approx.) future!



RL in Self-Driving