RL Traffic Signal Control

David Sanwald

May 9, 2018

Table of contents

Introduction

Reinforcement Learning

Machine Learning

- Supervised Learning
    - Classification
    - Regression
- Unsupervised Learning
    - Clustering
    - ...
- Reinforcement Learning

Agent Environment

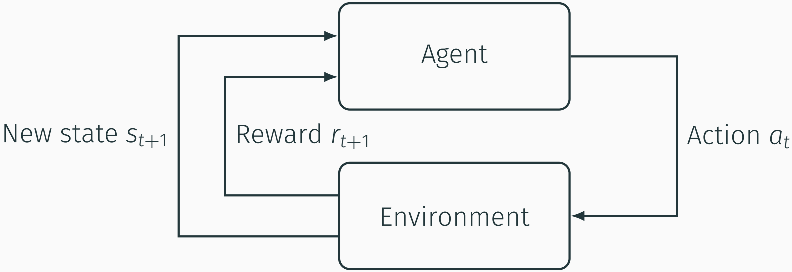


Figure 1: Agent environment interface

Markov Decission Process

MARKOV DECISSION PROCESS is defined by quatuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \mathcal{A} \rangle$

- $\mathcal{S}$, a set of states
- $\mathcal{P}$, a state transition matrix defining the probabilities of some possible next state s' given any state s
  $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- a reward function $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- $\mathcal{A}$, a set of actions

Policy

- specifies agent's behaviour
- mapping of state to action

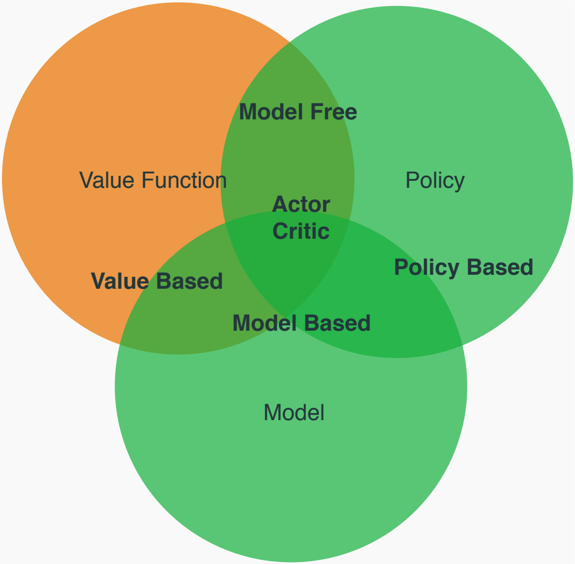
$$\pi(s) = a$$

$$\mathbb{P}(a|s) = \pi(a|s)$$

Markov Property

- The future is conditionally independent of the past given the presence
- implies memorylessnes

$$\mathbb{P}[S_{t+1}|S_1, \ldots, S_t] = \mathbb{P}[S_{t+1}|S_t]$$


Model Free
Value Function
Policy
Actor
Critic
Policy Based
Value Based
Model Based
Model

Value Function

Expected return

- from state s and action a
- given policy π

$$Q^\pi(s,a) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s, a]$$

- decomposable into

$$Q^\pi(s,a) = \mathbb{E}[r + \gamma Q^\pi(s',a') | s, a]$$

Optimal Value Function

- optimal value function

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

- optimal policy

$$\pi^*(s) = \operatorname*{argmax}_{a} Q^*(s,a)$$

- decomposition into

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a')|s,a]$$

TD Learning

Off Policy learning

Q-learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\underbrace{\underbrace{R_{t+1} + \gamma \max_a Q(s_{t+1}, a)}_{\text{target}} - \underbrace{Q(S_t, A_t)}_{\text{prediction}}}_{\text{TD-Error}})]$$

On Policy learning

Sarsa

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, A_{t+1}) - Q(S_t, A_t))]$$

Backup Diagrams

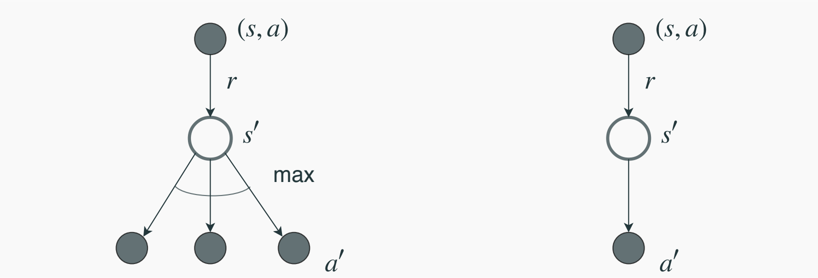


Figure 2: backup diagram for Q-learning and Sarsa

Q-learning

Initialize $Q(s, a)$ arbitrarily
Initialize S
repeat
  Choose A from S using policy derived from Q
  Take action A observe R, S'
  Choose A' from S' using policy derived from Q
  $Q(S, A) \leftarrow Q(SA) + \alpha[R + \gamma \max_a Q(S', a) - Q(S A)]$
  $S \leftarrow S'$
until S is terminal

Demo

Function Approximation

Why Function Approximation?

- large state spaces
- slow learning
- need for generalization

Naive Function Approximation

$$Q(s, a, \theta) \approx Q(s, a)$$

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta)\right)^2\right]$$

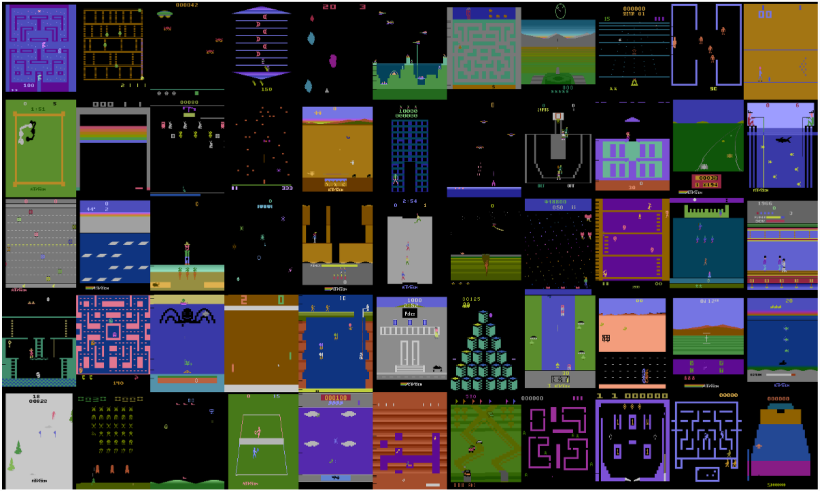
$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta)\right)\frac{\partial Q(s, a, \theta)}{\partial \theta}\right]$$

Deadly Triad

Deadly Triad

- function approximation
- off policy learning
- bootstrapping

atari arcade games



DQN

Human-level control through deep reinforcement learning[1]

- (almost) raw pixel input
- one agent/set of network weights
- comparable to human performance on 29 of 49 games

[1] nature february 2015

experience replay

- decorrelates
- sample efficiency

target network

- inhibits loops

error clipping

- limits gradient magnitude

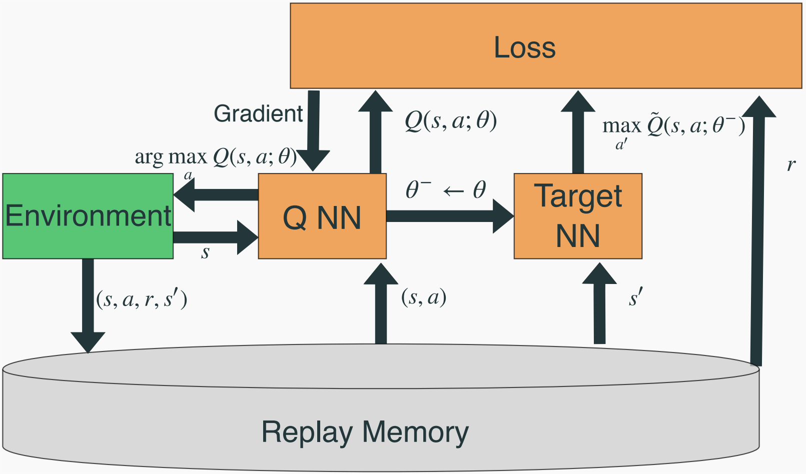
experience replay

- store experience $e_t = (s_t, a_t, r_t, s_{t=1})$ in $D_t = \{e_1, \ldots, e_t\}$
- at timestep t update $(s, a, r, s') \sim U(D)$

fixed target network

- separate target network $\tilde{Q}(s, a, \theta^{-})$ and online network $Q(s, a, \theta)$
- TD error becomes $r + \gamma \max_{a'} Q(s', a', \theta^{-}) - Q(s, a, \theta)$

DQN architecture



DQN conclusion

- generality
- decoupling of learning algorithm and domain
- no manual feature construction
- not as general as it might seem
- closely tied to strengths and weaknesses of NNs

Traffic Light Control

TLC as MDP

RL learns to maximize expected total reward in an MDP (best case)

- construct state signal
- determine reward function
- chose set of actions
- simulate environment dynamics

Markovian Road Users

Markovian Road Users

Markovian Road Users

Markovian Road Users

Markovian Road Users

Markovian Road Users

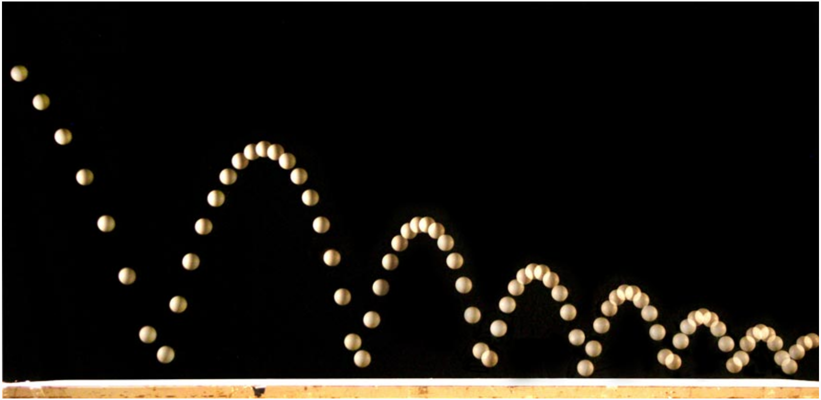


Table 1: My caption

frames	information	order
1	position	0
2	velocity	1
3	acceleration	2
4	jerk	3
5	jounce	4

Markovian Road Users

$$s = \begin{bmatrix} 1 \\ 0 \\ 5 \\ 4 \end{bmatrix}$$

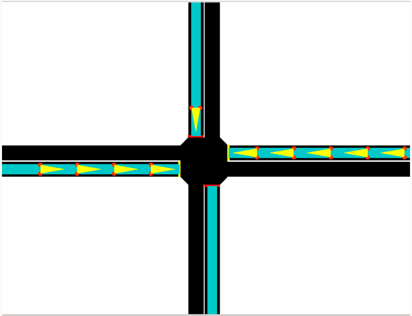


Figure 3: intersection with 4 approaches

1	1	1	0	1	0	0	0	0	0	0	0
0.01	0.07	0	0	0.68	0	0	0	0	0	0	0

Figure 4: Crop used for demonstrating different state representations

1	1	1	1	1	1	0	0	0	0	0	0
0	0.07	0.16	0.1	0.05	0	0	0	0	0	0	0

Figure 5: position and speed matrix for vehicle lengths 5m and 2m

Why is RL TLC hard?

- compound state, hard to extract and process features
- extremely noisy, hard to interpret, difficult to train
  - no Bayesian Optimization etc. for hyperparameter tuning
  - reproducibility problems
- not attractive for researchers from either one area ”beyond the hype”

difficult but maybe still a good idea?

- not scalable for multiple intersections
- does not profit from flexiblity and abstraction
- suffers from abstraction costs

my own experience

- dissapointing attitudes and transparence
- implementation matters
- strenghs and limits of ANNs
- big chances in a short amount of time