Introduction to Reinforcement Learning

References:

- https://github.com/dalmia/David-Silver-Reinforcement-learning
- https://deeplearningindaba.com/2020/
- http://introtodeeplearning.com

Content

1. What is reinforcement learning?
2. Markov Decision Process (MDP)
3. Policy
4. Value Functions and Bellman Equation
5. Approaches to Reinforcement Learning
   A. Value-based RL: SARSA and Q-Learning
   B. Model-based Methods
      I. Deep Q Networks
      II. Policy Gradients: REINFORCE and Actor-Critic
6. Reinforcement Learning Applications

1. What is reinforcement Learning

Classes of Learning Problems

Supervised Learning

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

Goal: Learn function to map

$$x \rightarrow y$$

Apple example:

This thing is an apple.

Classes of Learning Problems

Supervised Learning

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

Goal: Learn function to map
$$x \rightarrow y$$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

This thing is like the other thing.

Classes of Learning Problems

Supervised Learning

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

Goal: Learn function to map
$$x \rightarrow y$$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

This thing is like the other thing.

Reinforcement Learning

Data: state-action pairs

Goal: Maximize future rewards over many time steps

Apple example:

Eat this thing because it will keep you alive.

What is reinforcement learning?

We've seen how to solve many cool problems around supervised and unsupervised learning

But a major component of intelligence is decision making

What is reinforcement learning?

Reinforcement learning is the branch of machine learning relating to learning in **sequential decision making settings**

Behaviour learning

From supervised to reinforcement

Supervised learning, single decision point

Multiple decision points

- How do I know if I'm doing the right thing?
- How do my decisions now impact the future?
- Actions affect the environment!

Interacting with an environment

Decision maker (agent) exists within an environment

Interacting with an environment

Decision maker (agent) exists within an environment

Agent takes **actions** based on the environment **state**

Interacting with an environment

Decision maker (agent) exists within an environment

Agent takes **actions** based on the environment **state**

Environment **state** updates

Agent receives feedback as **rewards**

2. Markov Decision Process (MDP)

A model for decision making

Markov Decision Process (MDP)

$M = \langle S, A, T, R, \gamma \rangle$

A model for decision making

Markov Decision Process (MDP)

M = $\langle S, A, T, R, \gamma \rangle$

- States: encode world configurations
- Actions: choices made by agent

Questions:
- What are the states and actions in the following examples?
- Are they discrete or continuous?

Board Game

PacMan

States?

Actions?

Inventory Replenishment

Marketing

A model for decision making

Markov Decision Process (MDP)

M = $\langle S, A, T, R, \gamma \rangle$

Markov: ***Future is independent of the past, given the present***

$$\mathcal{P}_{ss'} = P(S_{t+1} = s' \mid S_t = s)$$

$$= P(S_{t+1} = s' \mid S_0, S_{t-1}, \dots, S_t = s)$$

A model for decision making

Markov Decision Process (MDP)

M = $\langle S, A, T, R, \gamma \rangle$

Transition function: how the world evolves under actions

$$\mathcal{P}_{ss'}^{a} = P(S_{t+1} = s' \mid S_t = s, A_t = a)$$

A model for decision making

Markov Decision Process (MDP)

M = ⟨S, A, T, R, γ⟩

Rewards: feedback signal to agent

Questions:
- What are the rewards received by each action/state in our previous examples?
- Does every action receive a reward?
- Immediate or long-term reward? Positive or negative?

Reward

- Reward may be associated with states or state-action pairs
  - Reward received when arriving at a state: $\mathcal{R}_s = E[R_{t+1}|S_t = s]$
  - Reward received when taking an action at a state: $\mathcal{R}_s^a = E[R_{t+1}|S_t = s, A_t = a]$
- Note some notations used here:
  - Given state S_t and action A_t, reward is received at time t+1, i.e. R_{t+1}
  - An episode can be shown as (_, S_1 A_1), (R_2, S_2, A_2), …, (R_t, S_t, A_t), …

$$\mathcal{R}_{s_0} = p_1\mathcal{R}_{s_0}^{a_0} + p_2\mathcal{R}_{s_0}^{a_1}$$

- Reward R_{t+1} is a scalar feedback signal
- Markov Reward Process (MRP): A Markov reward process is a Markov chain with values.
- It indicates how well agent is doing at step t
- The agent's job is to ***maximize cumulative reward***
- Reinforcement learning is based on the reward hypothesis:

Definition (Reward Hypothesis)

All goals can be described by the **maximization of expected cumulative reward**

Examples of Rewards

- Play Atari games
  - + increase score
  - – decrease score
- Play chess or board game Go
  - + win a game
  - – lose a game
- Make a humanoid robot walk
  - + reward for forward motion
  - – reward for failing over
- Manage an investment portfolio
  - + each $ gained in bank
  - – each $ lost
- Manage inventory:
  - – Order loss due to out-of-stock, storage cost
- Questions:
  - Does every action receive a reward in each of these examples?
  - Immediate or long-term reward?

Rewards

Scalar feedback signal

Encode (un)desirable features of behaviours:

Winning/losing, collisions, taking expensive actions, ...

- Sparse
- Delayed
- Only have relative value

A model for decision making

Markov Decision Process (MDP)

M = ⟨S, A, T, R, γ⟩

$\gamma \in [0,1]$ discounting for future rewards

Question: Why reward dicounts?

Reward Discounts

- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behavior shows preference for immediate reward
  - $\gamma \rightarrow 0$: "myopic" evaluation
  - $\gamma \rightarrow 1$: "far–sighted" evaluation
- If there is no discount, what would happen in a cyclic Markov process?

Sequential Decision Making

- Goal: *select actions to maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
  - A financial investment (may take months to mature)
  - Refuelling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

Example: Student MRP

Return

Definition

The *return* G_t is the total discounted reward from time-step t.

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

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Example: Student Markov Chain Episodes

Sample episodes for Student Markov Chain starting from $S_1 = \text{C1}$

$$S_1, S_2, ..., S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Example: Student MRP Returns

Sample **episodes** for Student Markov Chain

Starting from $s_1 = \text{C1}$ with $\gamma = \frac{1}{2}$

$G_1 = R_2 + \gamma R_3 + \ldots + \gamma^{T-2} R_T$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	$=$	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	$=$	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	$=$	-3.41
C1 FB FB C1 C2 C3 Pub C1 … FB FB FB C1 C2 C3 Pub C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	$=$	-3.20

3. Policy

Policy

- A policy is the agent's behavior
- It is a map from states to actions
- Two types of Policy:
  - **Deterministic** policy: $a = \pi(s)$
  - **Stochastic** policy:
    $\pi(a|s) = \mathrm{P}(A_t = a|S_t = s)$
- **Optimal Policy** (π^*): A policy return the maximal accumulated rewards
  $G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \ldots$
- This is what we want to learn!

$\pi(Pass)$ = Go to sleep

$\pi(a = Take\ Class\ 1|s = Pub)$ = 0.2

$\pi(a = Take\ Class\ 2|s = Pub)$ = 0.4

…

Start from "Class 1", what is the optimal policy? What if we don't know the transition probability, i.e. $\pi(a = Pass|s = Class\ 3)$? How to find it?

An example

Cleaning Robot

Actions:

Reward:

- +1 for finding dirt
- -1 for falling into hole
- -0.001 for every move

An example

States:

- Position on grid e.g.
  - S is (1,1), goal (4,3)

Actions:

Reward:

- +1 for finding dirt
- -1 for falling into hole
- -0.001 for every move

What is the optimal policy?

Is this an optimal policy?

How to search for the optimal policy?

4. Value Functions and Bellman Equations

Value Function

The value function $v(s)$ gives the long-term value of state s

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

Example: State-Value Function for Student MRP

What if γ = 0.9?

Bellman Equation for MRPs

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

Bellman Equation for MRPs (2)

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

Transition probability from s to s'

Example: Bellman Equation for Student MRP

Example: Bellman Equation for Student MRP

- What if γ = 0.9?

- What if transition probabilities between states (i.e. policy) are changed? Will state values change?

- Which policy is optimal?

Value Function under a Policy

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Bellman Expectation Equation

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]$$

The action-value function can similarly be decomposed,

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \underbrace{\gamma q_\pi(S_{t+1}, A_{t+1})}_{\text{discounted value of successor action}} \mid S_t = s, A_t = a \right]$$

Relationship between $V_\pi(s)$ and $q_\pi(s,a)$

1 $$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

2 $$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

3 $$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Example: Student MDP

Note, rewards are associated with actions

Example: Bellman Expectation Equation in Student MDP

Assume π(a|s) = 0.5, γ = 1

*7.4 = 0.5 * (1 + 0.2* -1.3 + 0.4 * 2.7 + 0.4 * 7.4) + 0.5 * 10*

Example: State-Value Function for Student MDP

GridWorld: Dynamic Programming Demo

- GridWorld: Dynamic Programming Demo (http://www.cs.toronto.edu/~lcharlin/courses/80-629/reinforcejs/gridworld_dp.html)

Random Policy
γ=0.9

One sweep
of update

- Why the state values become 1.0, 0.3, 0.3?
  - 2* 0.5 * (1 + 0) = 1.0
  - 1/3 * (0.9 * 1.0) = 0.3 (use Equation 3)

What will be its next state value ?

Reward 1 received only at this state

What are the two state-action values?

Optimal Value Function

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_{\pi} v_\pi(s)$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

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

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.

Finding an Optimal Policy

An optimal policy can be found by maximising over $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_*(s, a) \\ 0 & \textit{otherwise} \end{cases}$$

Move in direction of greatest value

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Finding Q* (or V*) is equivalent to finding π*

GridWorld demo: http://www.cs.toronto.edu/~lcharlin/courses/80-629/reinforcejs/gridworld_dp.html

Example: Optimal Policy for Student MDP

What if we don't know V* or q*? Can we find the optimal poilcy incrementally?

Principle of Optimality

Any optimal policy can be subdivided into two components:

- An optimal first action A_*
- Followed by an optimal policy from successor state S'

Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if

- *For any state s' reachable from s*
- *π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$*

Bellman Optimality Equation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

$$\Downarrow$$

$$\mathrm{v}^*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \, \mathrm{v}^*(s') \right)$$

Take the first optimal action a^* (i.e. $\max_a q_\pi(s, a)$)

followed by an optimal policy from successor state s'

The Goal of Reinforcement Learning

How to learn an optimal policy based on Bellman Optimality Equation?

$$\mathrm{v}^*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \, \mathrm{v}^*(s') \right)$$

The Goal of Reinforcement Learning

How to learn an optimal policy based on Bellman Optimality Equation?

$$\mathrm{v}^*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \, \mathrm{v}^*(s') \right)$$

Value Iteration: Iteratively update V

- At each iteration i :
  - For all states s in S :
    - Update $v_{i+1}(s) := \max_a (\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_i(s'))$

Try the GridWorld demo!

Challenges in searching $\mathrm{v}^*(s)$:

- Often $\mathcal{P}_{ss'}^a$ and sometimes $\mathcal{R}_s^a$ are unknown in advance
- There may be a large number of possible states and actions
- Exhaustive search is often impossible and ineffective

Example: Bellman Optimality Equation in Student MDP

5. Approaches to Reinforcement Learning

Taxonomy of RL Algorithms

- Value-based vs. Policy-based RL
  - Value-based: Estimate the optimal value function $v^*(s)$ or $q^*(s,a)$, the maximum value achievable under any policy
  - Policy-based: Search directly for the optimal policy π^*, the policy achieving maximum future reward
- Model-based vs. Model-free RL
  - Model-based RL build a model of the environment and plan (e.g. by lookahead) using model

https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Data generation

Transition probabilities and rewards are often unknown in advance

Instead, generate samples of training data ***(s,a,r,s')*** from environment

Learning from Experience

We need

- **A method to choose actions**

- Some model to keep track of and learn
  - *Value Function*

The Bandit Problem

Consider a row of one-arm bandit machines in a casino

Set of “arms” (actions) that each generate rewards from different distributions

Exploration vs exploitation

Action Selection: Exploration vs. Exploitation

Maximizing expected returns means balancing between:

- **Exploitation**: Exploiting gained knowledge (greedy)
  – Take the best known action
- **Exploration**: Exploring new actions/states (random)
  – Try something new

https://medium.com/deep-math-machine-learning-ai/ch-12-1-model-free-reinforcement-learning-algorithms-monte-carlo-sarsa-q-learning-65267cb8d1b4

Action selection strategies

ε-Greedy ($0 < \varepsilon \leq 1$):

- With probability $1-\varepsilon$ exploit
  - Choose the best action for a state
- With probability ε explore
  - Randomly choose action

ε usually higher at beginning of learning, decay later

Softmax

- Sample action given softmax

$$P_t(a) = \frac{\exp(Q_t(a)/\tau)}{\sum_{i=1}^{n} \exp(Q_t(i)/\tau)},$$

ϵ-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

5 (A). Value-based RL: SARSA and Q-Learning

Basic Idea: Bellman Equations

Under a policy π, at time t, if Q-value ($q_\pi(S_t, A_t)$) estimates are accurate, the following must hold:

$$q_\pi(S_t, A_t) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

If not, then there is an error:

$$\delta = q_\pi(S_t, A_t) - \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

Then we learn better Q-values to minimize the δ

- **SARSA**: Follow a policy π to determine the next action A_t(**On Policy**)
- **Q-Learning**: Reuse experience from other policies to determine the next action A_t (**Off Policy**)

SARSA Algorithm for On-Policy Control

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
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    $\boxed{\text{Choose } A' \text{ from } S' \text{ using policy derived from } Q \text{ (e.g., } \varepsilon\text{-greedy)}}$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\underbrace{\left[R + \gamma Q(S',A') - Q(S,A)\right]}_{\text{learned value}}$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

new value

old value

learning rate

expected target by Bellman Equation

old value

Pick A'

A' may not be the action with max $Q_t(S',A')$ over all possible A's because of exploration (ε)

Q-Learning Algorithm for Off-Policy Control

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'

    $$\boxed{Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \boxed{\max_a Q(S',a)} - Q(S,A)\right]}$$

    $S \leftarrow S'$;

  until S is terminal

new value

old value

learning rate

the most optimistic estimate so far

learned value

old value

- Estimate $Q_t(S,A)$ using $\max_{A'} Q_t(S,A')$, which is learned from past experience, but do not pick A' at Step t.
- Directly go to Step $t+1$ and then pick A based on updated Q values.

SARSA vs. Q-Learning

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
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figure 6.9: Sarsa: An on-policy TD control algorithm.

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'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$; what about A'?
  until S is terminal

this is like following a greedy policy (e.g. ε=0, NO exploration)

Figure 6.12: Q-learning: An off-policy TD control algorithm.

Sutton & Barto, 2018

Q-Learning demo

(*T*,*R*,*S*,*A*)

Shreyas Skandan: https://www.youtube.com/watch?v=RTu7G0y4Os4

The Game of Go

Aim: Get more board territory than your opponent.

Board Size $n \times n$	Positions 3^{n^2}	% Legal	Legal Positions
1×1	3	33.33%	1
2×2	81	70.37%	57
3×3	19,683	64.40%	12,675
4×4	43,046,721	56.49%	24,318,165
5×5	847,288,609,443	48.90%	414,295,148,741
9×9	$4.434264882 \times 10^{38}$	23.44%	$1.03919148791 \times 10^{38}$
13×13	$4.300233593 \times 10^{80}$	8.66%	$3.72497923077 \times 10^{79}$
19×19	$1.740896506 \times 10^{172}$	1.20%	$2.08168199382 \times 10^{170}$

Greater number of legal board positions than atoms in the universe.

It is impossible to store legal positions (states) in memory

Source: Wikipedia.

Large-Scale Reinforcement Learning

Reinforcement learning can be used to solve *large* problems, e.g.

- Backgammon: 10^{20} states
- Computer Go: 10^{170} states
- Helicopter: continuous state space

Large-Scale Reinforcement Learning

Reinforcement learning can be used to solve *large* problems, e.g.

- Backgammon: 10^{20} states
- Computer Go: 10^{170} states
- Helicopter: continuous state space

How can we scale up the model-free methods for *prediction* and *control* ? **Use models to approximate v(s) and q(s, a)!**

5 (B). Model-Based Methods

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Value Function Approximation

- So far we have represented value function by a *lookup table*
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - *Generalise* from seen states to unseen states
  - *Update* parameter $\mathbf{w}$ using gradient descent

Types of Value Function Approximation

Deep neural Networks can be used as function approximators

What about cost function?

Least Squares Prediction

- Given value function approximation $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
- And *experience* $\mathcal{D}$ consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Which parameters $\mathbf{w}$ give the *best fitting* value fn $\hat{v}(s, \mathbf{w})$?
- Least squares algorithms find parameter vector $\mathbf{w}$ minimising sum-squared error between $\hat{v}(s_t, \mathbf{w})$ and target values v_t^π,

$$\begin{aligned} LS(\mathbf{w}) &= \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2 \\ &= \mathbb{E}_\mathcal{D} \left[(v^\pi - \hat{v}(s, \mathbf{w}))^2 \right] \end{aligned}$$

Stochastic Gradient Descent with Experience Replay

Given experience consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

Repeat:

1. Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha(v^\pi - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

Stochastic Gradient Descent with Experience Replay

Given experience consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

Repeat:

1. Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha(v^\pi - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

Converges to least squares solution

The question is how to get V^π

$$\mathbf{w}^\pi = \underset{\mathbf{w}}{\text{argmin}}\ LS(\mathbf{w})$$

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimise mean-squared error between approximate action-value fn $\hat{q}(S, A, \mathbf{w})$ and true action-value fn $q_\pi(S, A)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

Similarly, how to calculate $q_\pi(S, A)$?

Deep Q Networks (DQN)

How can we use deep neural networks to model Q-functions?

Deep Q Networks (DQN)

How can we use deep neural networks to model Q-functions?

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?

Sounds familiar? This is used in Q-Learning

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

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?

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

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?

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

Experience Replay in Deep Q-Networks (DQN)

DQN uses experience replay and fixed Q-targets

- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right]$$

- Using variant of stochastic gradient descent

DQN: Atari

Convolution
Convolution
Fully connected
Fully connected
No input

DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

Network architecture and hyperparameters fixed across all games

DQN Demo

- ConvNetJS Deep Q Learning Demo, https://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html

- Atari:
  - https://www.youtube.com/watch?v=MKtNv1UOaZA
  - https://becominghuman.ai/lets-build-an-atari-ai-part-1-dqn-df57e8ff3b26

- WaterWorld DQN (http://www.cs.toronto.edu/~lcharlin/courses/80-629/reinforcejs/waterworld.html)

Downsides of Q-learning

Complexity:

- Can model scenarios where the action space is discrete and small
- Cannot handle continuous action spaces

IMPORTANT:
Imagine you want to predict steering wheel angle of a car!

Flexibility:

- Cannot learn stochastic policies since policy is deterministically computed from the Q function

To overcome, consider a new class of RL training algorithms: Policy gradient methods

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Advantages of Policy-Based RL

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

Example: Aliased Gridworld (1)

- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

$$\phi(s, a) = \mathbf{1}(\text{wall to N}, a = \text{move E})$$

i.e. Whether there is a walls to N and it can move to E

- Compare value-based RL, using an approximate value function

$$Q_\theta(s, a) = f(\phi(s, a), \theta)$$

- To policy-based RL, using a parametrised policy

$$\pi_\theta(s, a) = g(\phi(s, a), \theta)$$

Example: Aliased Gridworld (2)

- Under aliasing, an optimal deterministic policy will either
  - move W in both grey states (shown by red arrows)
  - move E in both grey states
- Either way, it can get stuck and *never* reach the money
- Value-based RL learns a near-deterministic policy
  - e.g. greedy or ϵ-greedy
- So it will traverse the corridor for a long time

Example: Aliased Gridworld (3)

- An optimal stochastic policy will randomly move E or W in grey states

$$\pi_\theta(\text{wall to N and S, move E}) = 0.5$$
$$\pi_\theta(\text{wall to N and S, move W}) = 0.5$$

- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Policy Gradient (PG) : Key Idea

DQN (before): Approximating Q and inferring the optimal policy,

Policy Gradient (PG): Key Idea

DQN (before): Approximating Q and inferring the optimal policy,

Policy Gradient: Directly optimize the policy!

Policy Gradient (PG): Key Idea

DQN (before): Approximating Q and inferring the optimal policy,

Policy Gradient: Directly optimize the policy!

$$\sum_{a_i \in A} \pi(a_i|s) = 1$$

Policy Gradient (PG): Key Idea

DQN (before): Approximating Q and inferring the optimal policy,

Policy Gradient: Directly optimize the policy!

$$\sum_{a_i \in A} \pi(a_i|s) = 1$$

$$\pi(a|s) = P(action|state)$$

Policy Gradient (PG): Key Idea

DQN (before): Approximating Q and inferring the optimal policy,

Policy Gradient: Directly optimize the policy!

$$\sum_{a_i \in A} \pi(a_i|s) = 1$$

$$\pi(a|s) = P(action|state)$$

But how to compute gradients?

Policy Objective Functions

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ
- But how do we measure the quality of a policy π_θ?
- In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

Score Function

Optimization Objective: Find θ that maximizes J(θ)

- We now compute the policy gradient *analytically*
- Assume policy π_θ is differentiable whenever it is non-zero
- and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- Likelihood ratios exploit the following identity

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

- The score function is $\nabla_\theta \log \pi_\theta(s, a)$

One-Step MDPs

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = \mathcal{R}_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\
&= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \mathcal{R}_{s,a} \\
\nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \underline{\pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) \mathcal{R}_{s,a}} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r]
\end{aligned}
$$

Remember, $\pi_\theta(s, a)$ is a function of s

The gradient does not contain d(s) or transition probabilities any more

Policy Gradient Theorem

- The policy gradient theorem generalises the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $Q^{\pi}(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem

For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, or $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ Q^{\pi_\theta}(s, a)\right]$$

Now we only need to figure out how to approximate Q values under a policy

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- For an episode $[1, 2, \ldots, T]$, at each step t, using return:
  $G_t = \gamma^t R_t + \gamma^{t+1} R_{t+1} + \ldots + \gamma^T R_T$
  as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

- If G_t is high, push up the probabilities of the actions seen
- If G_t is low, push down the probabilities of the actions seen

Monte-Carlo Policy Gradient (REINFORCE) Architecture

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Reducing Variance in REINFORCE

- Rewards received during trajectories may have high variance
- Different estimates of $Q^{\pi_\theta}(s_t, a_t)$ have been used, i.e., R_t, G_t, …
- Sometimes, a move without receiving a reward (or even receiving a negative reward) is not necessary a bad move
- How to reduce variance?
  - ***Critic***: model $Q^{\pi_\theta}(s, s)$ using a function $Q^w\ (s, a)$
  - ***Advantage***: Check whether a reward is better or worse than what is expected
    - Introduce a baseline $b(s) = v^{\pi_\theta}(s)$, the expected value of s
    - Again, model $v^{\pi_\theta}(s)$ using a function $V^w(s)$
    - Advantage $A^w(s, a) = Q^w\ (s, a) - V^w(s)$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\, G_t\right] \qquad \text{REINFORCE}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ Q^w(s, a)\right] \qquad \text{Q Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ A^w(s, a)\right] \qquad \text{Advantage Actor-Critic}$$

Reducing Variance Using a Critic

- Monte-Carlo policy gradient still has high variance
- We use a critic to estimate the action-value function,

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain *two* sets of parameters

  Critic Updates action-value function parameters w ***similar to Deep Q-Learning!***

  Actor Updates policy parameters θ, in direction suggested by critic ***similar to REINFORCE!***

- Actor-critic algorithms follow an *approximate* policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ Q_w(s, a)\right]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a)\ Q_w(s, a)$$

Actor-Critic Algorithm

- Critic: Estimate the value function
  - Action-value (the Q value) or state-value (the V value)
  - Updates parameter w by minimizing the temporal difference (TD) between action value $Q_w(s,a)$ and $[r_t + \gamma Q_w(s',a')]$ (updated value after a')
- Actor: Updates θ by policy gradient

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \ldots T$: **do**
  Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
  $\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$
  and use it to update the parameters of Q function:
  $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

REINFORCE

Similar to Deep Q-Learning

Actor-Critic Architecture Design

Separate Networks for Policy and Value

Shared Network Design

6. Reinforcement Learning Applications

AlphaGo Beats Top Human Player at Go (2016)

Silver et al., *Nature* 2016.

AlphaGo Beats Top Human Player at Go (2016)

1) Initial training: human data

Silver et al., *Nature* 2016.

AlphaGo Beats Top Human Player at Go (2016)

1) Initial training: human data

2) Self-play and reinforcement learning
→ super-human performance

Silver et al., *Nature* 2016.

AlphaGo Beats Top Human Player at Go (2016)

1) Initial training: human data

2) Self-play and reinforcement learning
→ super-human performance

3) "Intuition" about board state

Silver et al., *Nature* 2016.

AlphaGo Beats Top Human Player at Go (2016)

Silver et al., *Nature* 2016.

Application Areas

Mobile Health Interventions

Heartsteps

Experimental Design & Machine Learning Opportunities in Mobile Health: Susan Murphy

Application Areas

Robotics:

learning behaviours

RL Application Areas

Games

- Standardised testbeds
- Long decision horizons

Application Areas

Automated Trading

1:

2: ???

3:

Questions?

Maze Example

- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Can you find the optimal policy π(a|s)?

Maze Example: Policy

Can you calculate the action values under this deterministic policy?

- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function

- Numbers represent value $v_\pi(s)$ of each state s