Reinforcement Learning (cont.)

Nan Ye

School of Mathematics and Physics
The University of Queensland

Reinforcement Learning

Planning vs Reinforcement Learning

- Planning
  - an MDP model is given for the environment,
  - we need to solve the MDP to find an optimal policy.
- Reinforcement learning (RL)
  - the MDP is only partially known, but interaction with the environment is allowed,
  - we want to find an optimal policy using as few interactions and as little computational resources as possible

planning

I know I'll get stuck in a traffic jam w.p. 0.2 if I go right, w.p. 0.5 if I go down

RL

Hmm, which route is more likely to have a traffic jam? I'll try it out...

w.p. = *with probability*

RL examples in life

- Shopping: we try several sellers first, then we'll stick to the one which we have best experience with.
- This also happens when we find our favourite authors, singers, sports, ...

General form of an RL algorithm

- An RL algorithm generally iterates between the following two steps
  - execute a behavior policy to interact with the environment and collect experience
  - learn from experience
- The *behavior policy* may be some fixed policy or may evolve as we have more experience.
  - this is an important design decision in RL algorithms
  - involves exploration-exploitation tradeoff (next slide)
- The learning step may directly learn a *target policy* π or indirectly learn it by learning its value function (or action-value function).

Exploration-exploitation Tradeoff

- When we interact with the environment in RL, how should we act?
- We can *explore* less explored actions to see whether they are more rewarding.
- We can *exploit* current information to take the best action based on current information.
- Dilemma
  - too much exploration $\Rightarrow$ can't sufficiently exploit actions which are found to be useful
  - too much exploitation $\Rightarrow$ can't explore unexplored optimal actions
- A good behavior policy should balance exploration and exploitation.

RL Approaches

Model-based vs. model-free

- Model-based: involves learning the environment model
- Model-free: learn an optimal policy without learning the environment model

Off-policy vs. on-policy

- Off-policy: evaluate or improve a non-behavior policy
  - target policy $\neq$ behavior policy
  - experience data is "off" target policy
- On-policy: evaluate or improve the behavior policy
  - target policy $=$ behavior policy
  - experience data is "on" target policy

Temporal Difference (TD) Methods

- TD methods learn value function approximations by performing updates using current estimates
  - TD methods are model-free: they directly estimate value functions without learning a model.
  - TD methods are bootstrapping methods as the update is partly based on existing estimates.
- We cover an off-policy TD method known as Q-learning, and an on-policy TD method known as SARSA.

Q-learning

MDPs with finitely many states

- Q-learning tries to directly estimate the optimal Q-function by solving the Bellman optimality equation

$$Q^*(s,a) = \sum_{s'} T(s' \mid s,a)(R(s,a) + \max_{a'} Q^*(s',a'))$$

- Key idea: if we experience a transition (s,a,s',r), then we can use it to perform an update

$$Q(s,a) \leftarrow Q(s,a) + \alpha\underbrace{(\overbrace{r + \gamma \max_{a'} Q(s',a')}^{\text{TD target}} - Q(s,a))}_{\text{TD}},$$

where $\alpha > 0$ is the learning rate, s is the current state, and s' and r are the next state and the reward obtained after executing a.

Algorithm 1 Tabular Q-learning

Initialise the state-action value function Q
while termination condition not met **do**
    Execute the behavior policy to obtain a new experience (s, a, s', r)
    Perform TD update for Q using the new experience

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)).$$

- Various termination criteria can be used
    - e.g. little change over recent updates, maximum number of interaction, maximum computation time
- A commonly used behavior policy is the ϵ-greedy policy, which executes a random action w.p. $\epsilon > 0$, and the greedy action $\arg\max_a Q(s, a)$ w.p. $1 - \epsilon$.

- Q-learning is an off-policy algorithm. Why?
  - The (implicit) target is the optimal policy, but the experience comes from a (non-optimal) behavior policy.
  - However, the experience s, a, r, s' is not generated by the optimal policy $\Rightarrow$ it is "off" the target policy.
  - To learn from the off-policy data, we update using an *imagined* experience s, a, r, s', a^*, where $a^* = \arg\max_{a'} Q(s', a')$.
    - We use the current Q to help us imagine how the optimal policy would behave when adding a^*.

MDPs with infinitely many states

- If we have infinitely many states, we can't use a table to store the Q-function.
- Typically, we use a parametric representation $Q_\theta(s, a)$ in this case.
- The update step in the Q-learning algorithm becomes

$$\theta \leftarrow \theta - \alpha\big(Q_\theta(s, a) - r - \gamma \max_{a'} Q_\theta(s', a')\big) \nabla Q_\theta(s, a).$$

Why? This performs a gradient descent on the squared TD error

$$\big(Q_\theta(s, a) - r - \gamma \max_{a'} Q_{\theta^-}(s', a')\big)^2,$$

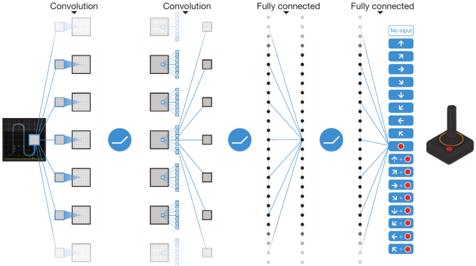
where $\theta^- = \theta$ is treated as fixed parameters.

Deep Q-Networks (DQN) for Atari Games

Key ideas

- A deep CNN approximation $Q_\theta(s, a)$
  - state consists of the last 4 frames, and Q_θ is a CNN that takes in a preprocessed representation $\phi(s)$, and outputs the action probabilities.
- Experience replay
  - instead of using current observed transition to update model, use a randomly sampled minibatch from the experience memory
- Separate target Q-network
  - A separate Q-network Q_{θ^-} is used to compute the TD target, and Q_{θ^-} is updated to Q_θ after a given number of steps

Mnih et al., Human-level control through deep reinforcement learning, 2015



A schematic illustration of a CNN for Q_θ

Architecture for Q_θ in (Mnih et al., 2015)

- Input: $84 \times 84 \times 4$ stack of last 4 frames (after an RGB-to-gray conversion and scaling)
- 1st hidden layer: Conv(8x8, 32, S=4) + ReLU
- 2nd hidden layer: Conv(4x4, 32, S=2) + ReLU
- 3rd hidden layer: Conv(3x3, 64, S=1) + ReLU
- 4th hidden layer: FC-512 + ReLU
- output layer: softmax over actions

Algorithm 2 DQN

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function Q with weights $\theta^- = \theta$
for episode=1 to M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$.
  for $t = 1$ to T **do**
    Select a_t randomly w.p. ϵ and as $\arg\max_a Q_\theta(\phi(s_t), a)$ w.p. $1 - \epsilon$
    Execute 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 D
    Sample random minibatch of transitions $\{(\phi_j, a_j, r_j, \phi_{j+1}) : j \in J\}$ from D
    For $j \in J$, set $y_j = \begin{cases} r_j, & \text{if episode terminates at step } j+1, \\ r_j + \gamma \max_{a'} Q_{\theta^-}(\phi_{j+1}, a'), & \text{otherwise.} \end{cases}$.
    Perform a gradient descent step on $\frac{1}{|J|} \sum_{j \in J} (y_j - Q_\theta(\phi_j, a_j))^2$ wrt θ.
    Set $\theta^- = \theta$ if t is a multiple of C

SARSA

- Q-learning is an off-policy model-free RL algorithm.
- SARSA is an on-policy variant of Q-learning.
- It is the same as Q-learning, except that for each update, it first observes a sequence s, a, r, s', a' (that's why the name SARSA), then update

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)).$$

- Why is SARSA on-policy?
  - $q = r + \gamma Q(s', a')$ is a bootstrap estimate of the behavior policy's value $Q(s, a)$ using its experience s, a, r, s', a'.
  - The update aims to provide an improved estimate using the convex combination $\alpha q + (1 - \alpha) Q(s, a)$.
  - Thus we are using the behavior policy's experience to improve itself.
- If the behavior policy is a fixed policy π, Q would converge to Q_π, thus we can use the algorithm for policy evaluation.

REINFORCE

- Q-learning and SARSA learn value function approximations, but we can also directly learn a policy.
- REINFORCE tries to directly optimise a parametric policy $\pi_\theta(a \mid s)$ by maximizing its value function

$$V(\theta) = \sum_\tau p(\tau \mid \theta) R(\tau) = \mathbb{E}_{\tau \sim p} R(\tau),$$

  where

  - $\tau = (s_1, a_1, s_2, a_2, \ldots)$ is a trajectory (state-action sequence),
  - $p(\tau \mid \theta)$ is the distribution of trajectory τ when playing π_θ, and
  - $R(\tau)$ is the total (discounted) reward collected along τ.
- It computes a stochastic gradient of $V(\theta)$ at each iteration, and then performs gradient ascent.

- Usually, it is often computationally intractable to evaluate $V(\theta)$ first, and then evaluate its gradient,
  - in the discrete state case, $V(\theta)$ involves summing over a large number of trajectories.
  - in the continuous state case, computing $V(\theta)$ involves evaluating a complex integral.

A stochastic gradient

- REINFORCE uses the following important observation

$$\nabla V(\theta) = \mathbb{E}_{\tau \sim p} R(\tau) \nabla \ln p(\tau \mid \theta).$$

  Why? Because $\nabla \ln p(\tau \mid \theta) = \frac{\nabla p(\tau|\theta)}{p(\tau|\theta)}$.
- This gives us a Monte Carlo estimate of the gradient

$$\nabla V(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau^{(i)}) \nabla \ln p(\tau^{(i)} \mid \theta),$$

  where the trajectories $\tau^{(1)}, \ldots, \tau^{(N)}$ are randomly sampled from $p(\cdot \mid \theta)$.
- We need to relate $\nabla \ln p(\tau \mid \theta)$ back to the policy π_θ.

- Note that

$$p(\tau \mid \theta) = p(s_1) \prod_{t=1}^{|\tau|} \pi(a_t \mid s_t, \theta) p(s_{t+1} \mid s_t, a_t),$$

  where $|\tau|$ denotes the length of a trajectory (number of state-action pairs).
- While $p(\tau^{(i)} \mid \theta)$ depends on the transition probabilities, the gradient of the log probability does not,

$$\nabla \ln p(\tau^{(i)} \mid \theta) = \sum_{t=1}^{|\tau_i|} \nabla \ln \pi_\theta(a_t^{(i)} \mid s_t^{(i)}).$$

Putting things together

- REINFORCE repeatedly improves $V(\theta)$ as follows
  - Simulate π_θ to collect trajectories $\tau^{(1)}, \dots, \tau^{(N)}$.
  - Update θ using

$$\theta \leftarrow \theta + \alpha \left(\frac{1}{N} \sum_{i=1}^{N} R(\tau^{(i)}) \left(\sum_{t=1}^{|\tau^{(i)}|} \nabla \ln \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \right) \right).$$

REINFORCE is an on-policy method.

More RL Algorithms

- Value function fitting methods
  - *estimate value function or Q-function of the optimal policy*
  - *e.g. temporal difference learning (SARSA, Q-learning, DQN), fitted value iteration*
- Policy gradient methods
  - *use gradient-based method to maximize the value of a policy*
  - *e.g. REINFORCE, TRPO (trust region policy optimization), PPO*
- Actor-critic algorithms
  - *estimate value function or Q-function of current policy, use it to imporve policy*
  - *e.g. A3C (asynchronous advantage actor-critic), SAC (soft actor-critic)*
- Model-based RL algorithms
  - *estimate an environment model for policy learning*
  - *e.g. Dyna, guided policy search*

What You Need to Know

- Reinforcement learning
  - algorithm: exploration using a behavior policy + update on the target policy
  - exploration-exploitation tradeoff
- Approaches: model-based vs model-free, off-policy vs on-policy
- Temporal different learning: Q-learning, DQN, SARSA
- Policy gradient methods: REINFORCE