Reinforcement Learning

Eilif Solberg

28.09.2018

Outline

Section 1

Introduction

Why reinforcement learning?

Promises of reinforcement learning (vs supervised learning)

- Less detailed instructions/annotations needed
  - task rather than implementation
- Supervised learning is about *imitating* behaviour
- Reinforcement learning is about *optimal* behaviour

Reinforcement learning is not new - but still in its infancy

Agent-environment interaction

Interactive play where at each 'iteration'

1. Agent do an action a_t according to its policy π
2. The environment responds with
    - observation o_{t+1}
    - reward r_{t+1}

The goal of the agent is maximize reward.

- $\max_\pi E_\pi[\sum_t \gamma^t R_t]$, $\gamma \in (0, 1]$ is called the *discount factor*

What is the goal of the environment?

Agent-environment interaction II

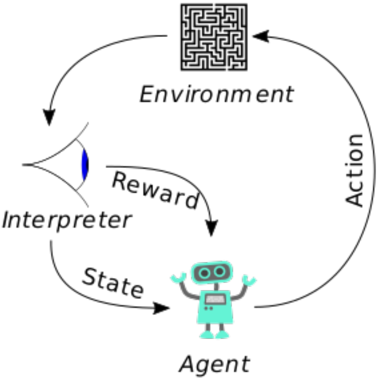

Figure: Illustration: By Megajuice [CC0], from Wikimedia Commons

Agent policy and agent state

The agent should choose action based on the information available

$$a_t \sim \pi(o_0, a_0, r_1, o_1, \ldots, a_{t-1}, r_t, o_t) \tag{1}$$

- Will assume that we have "enough" information.

Often tries to simplify

$$s_t = f(o_0, a_0, r_1, o_1, \ldots, a_{t-1}, r_t, o_t) \tag{2}$$
$$a_t \sim \pi(s_t) \tag{3}$$

Agent state is the information the agent uses to choose actions

Environment state

- The information the environment uses to base its response on.
- Usually some *unknown* distribution

$$(o_{t+1}, r_{t+1}) \sim \mathcal{P}(o_0, a_0, r_1, o_1, \ldots, a_{t-1}, r_t, o_t, a_t) \quad (4)$$

We often assume

$$s_t = f(o_0, a_0, r_1, o_1, \ldots, a_{t-1}, r_t, o_t) \quad (5)$$

$$(o_{t+1}, r_{t+1}) \sim \mathcal{P}(s_t, a_t) \quad (6)$$

Interplay revisited

$$s_t = f(o_0, a_0, r_1, o_1, \ldots, a_{t-1}, r_t, o_t) \tag{7}$$

$$a_t \sim \pi(s_t) \tag{8}$$

$$(o_{t+1}, r_{t+1}) \sim \mathcal{P}(s_t, a_t) \tag{9}$$

We write $\pi(a|s)$ for probability/density of choosing action a given state s.

Section 2

Prediction - evaluating policy

Evaluation in supervised learning

For some loss function g

$$L = \frac{1}{N} \sum_{i=1}^{N} g(f(X_i), Y_i) \tag{10}$$

In reinforcement learning we have no fixed dataset or loss!

State-value function

Define the *return*

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{11}$$

i.e. discounted future reward from time t and onwards.
We define state-value function as

$$v_\pi(s) = E_\pi[G_0 | S_0 = s] \tag{12}$$

- Tells us how *good* a state is

Action-value function

Expected future reward from state s when taking action a

$$q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a] \tag{13}$$

- Tells us how *good* it is to take an action from state s

What's the relation between v_π and q_π?

$$v_\pi(s) = \int \pi(a|s) q_\pi(s,a) da \tag{14}$$

Subsection 1

Monte-Carlo - full lookahead

Monte Carlo

Want to estimate e.g. state-value function

$$v_\pi(s) = E_\pi[G_t | S_t = s] \tag{15}$$

- Can we calculate this expectation?
- We can sample it!

Sample N episodes and then get estimates

$$V[s] \leftarrow \frac{1}{N[s]} A[s] \tag{16}$$

- N[s] is visit count and A[s] accumulated rewards
- Does this converge?

Monte Carlo - incremental update

Incremental update after each episode:

$$N[s_t] \leftarrow N[s_t] + 1 \tag{17}$$

$$V[s_t] \leftarrow V[s_t] + \frac{1}{N[s_t]}(G_t - V[s_t]) \tag{18}$$

Later also updates of the form

$$V[s_t] \leftarrow V[s_t] + \alpha(G_t - V[s_t]) \tag{19}$$

for $\alpha > 0$

Incremental average derivation

$$\bar{x}_n := \frac{1}{n}\sum_{i=1}^{n} x_i \tag{20}$$

$$= \frac{1}{n}((n-1)(\frac{1}{n-1}\sum_{i=1}^{n-1} x_i) + x_n) \tag{21}$$

$$= \frac{1}{n}((n-1)\bar{x}_{n-1}) + x_n) \tag{22}$$

$$= \bar{x}_{n-1} + \frac{1}{n}(x_n - \bar{x}_{n-1}) \tag{23}$$

Update with function approximation

$v_\eta(s_t)$ are our previous estimate with our function approximator with parameters η. Define loss for each prediction

$$l(\eta) = \frac{1}{2}\big(G_t - v_\eta(s_t)\big)^2 \tag{24}$$

- G_t are our "supervised" targets

Taking gradients

$$\nabla_\eta l(\eta) = -(G_t - v_\eta(s_t))\nabla_\eta v_\eta(s_t) \tag{25}$$

Update in steepest descent direction

$$\eta = \eta + \alpha\big(G_t - v_\eta(s_t)\big)\nabla_\eta v_\eta(s_t) \tag{26}$$

Note that $\nabla_\eta v_\eta(s_t)$ is direction which *increases* value estimate

Subsection 2

Temporal Difference - one step lookahead

Temporal difference learning

Value estimates are not independent of each other!

- Assume you are in state s_t, estimated future reward is $v_\pi(s_t)$
- When we go one step ahead, estimate usually *changes* due to
  - randomness in our action
  - randomness in environment state transition and reward
- We should *on average* get the same expected future reward.

Bellman expectation equations:

$$v_\pi(s_t) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})] \quad (27)$$

$$= \int\int\int \pi(a|s_t)p(r, s'|s_t, a)(r + \gamma v_\pi(s'))ds'drda \quad (28)$$

Update equations

Assume value estimates stored in array

$$V[s] = \sum_a \sum_r \sum_{s'} \pi(a|s)p(r, s'|s, a)(r + \gamma V[s']) \tag{29}$$

We can iteratively update *value estimates* by

$$V[s_t] \leftarrow V[s_t] + \alpha((R_{t+1} + \gamma V[S_{t+1}]) - V[s_t]) \tag{30}$$

With function approximation, update *function parameters* η

$$\eta = \eta + \alpha\big((R_{t+1} + v_{\eta'}(S_{t+1})) - v_\eta(s_t))\big)\nabla_\eta V_\eta(s_t) \tag{31}$$

Subsection 3

TD(λ) - "intermediate" lookahead

TD(λ)

- Monte Carlo: No bias, high variance
- Temporal difference learning: lower variance, some bias

TD(λ) - continuous spectrum of models between MC and TD, $\lambda \in (0, 1)$

Section 3

Control - optimizing policy

Control - optimizing policy

Find the *optimal* policy π

$$v_\pi(s) = max_{\pi'} v_{\pi'}(s) \tag{32}$$

- How can we improve a given policy π?
- Do more of the good actions and less of the bad
- How do we measure the *goodness* of an action?

Subsection 1

Policy gradient - policy based control

Policy gradient

Defined a parametrized family of policies $\pi_\theta,\ \theta \in \Theta$. Reduced problem to

$$\pi_{\theta^*} = argmax_\theta E_{\pi_\theta}(G_0) \tag{33}$$

- Recall that G_0 is *return*, expected discounted reward

We know how to do parameter-optimization, right?

Supervised learning training

Assume differentiable loss function l and f differentiable in θ.

$$\min_{\theta} E(l(f(X;\theta),Y)) \tag{34}$$

Find gradient with

$$\begin{aligned}
\nabla_\theta E(l(f(X;\theta),Y)) &= \nabla_\theta \int\int p(x,y) l(f(x;\theta),y) dxdy && (35)\\
&= \int\int \nabla_\theta (p(x,y) l(f(x;\theta),y)) dxdy && (36)\\
&= \int\int p(x,y) \nabla_\theta l(f(x;\theta),y) dxdy && (37)\\
&\approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta l(f(x_i;\theta),y_i) && (38)
\end{aligned}$$

- Can we do something similar in RL?

RL imitation

Let z denote an episode, i.e. $z = (s_0, a_0, r_1, s_1, , a_{\tau-1}, r_\tau, s_\tau)$. $r(z) = \sum_{t=1}^{\tau} r_t$. Want to optimize

$$E_{\pi_\theta}(G_0) = \int p(z;\theta)r(z)dz \tag{39}$$

Let's see if we can get the gradient

$$\nabla_\theta E_{\pi_\theta}(G_0) = \nabla_\theta \int p(z;\theta)r(z)dz \tag{40}$$

$$= \int \nabla_\theta p(z;\theta)r(z)dz \tag{41}$$

- Are we stuck?

Log-derivative trick

For a variable of one parameter x

$$\frac{d}{dx}\log f(x) = \frac{\frac{d}{dx}f(x)}{f(x)} \tag{42}$$

For a policy of several variables θ this generalizes to

$$\nabla_\theta \log f(\theta) = \frac{\nabla_\theta f(\theta)}{f(\theta)} \tag{43}$$

and thus

$$\int \nabla_\theta p(z;\theta) r(z) dr = \int p(z;\theta) \nabla_\theta \log p(z;\theta) r(z) dr$$

and can be sampled with

$$\nabla_\theta E_{\pi_\theta}(G_0) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log p(z_i;\theta) r(z_i) \tag{44}$$

Gradient

Remains to figure out expression for $\nabla_\theta \log p(z;\theta)$. Turns out that it is

$$\nabla_\theta \log p(z;\theta) = \sum_{t=0}^{\tau-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{45}$$

and thus our full gradient estimate is

$$\nabla_\theta E_{\pi_\theta}(G_0) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{\tau^{(i)}} \left(\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) r^{(i)} \right) \tag{46}$$

and our update becomes

$$\theta \leftarrow \theta + \alpha \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{\tau^{(i)}} \left(\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) r^{(i)} \right) \tag{47}$$

Each action $a_t^{(i)}$ contributes $\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) r^{(i)}$

Subsection 2

Policy iteration and value iteration - value based control

Policy iteration - idea

We saw before that our value function was given by

$$v_\pi(s) = \int \pi(a|s) q_\pi(s,a) da$$

- Expected reward an average of between good and bad actions
- Why not just choose the best action?

$$\pi'(s) := argmax_a q_\pi(s,a) \tag{48}$$

That this works, i.e. $\pi' \geq \pi$, is know as the *policy improvement theorem*.

Policy iteration - algorithm I

For $i = 0, 1, 2, \ldots$ repeat the following two steps

1. Policy evaluation Estimate the value function for policy π_i.
2. Policy improvement Define a policy π_{i+1} by acting *greedily* with respect to the value function estimated in the previous step.

- Usually only crudely approximate each step
- Incomplete knowledge of environment –> need to ensure we keep exploring

For $i = 0, 1, 2, \ldots$ repeat the following two steps

1. Policy evaluation Estimate the value function $\hat{q}_{\pi_i}$
2. Policy improvement Define a policy π_{i+1} by acting ϵ*-greedily* with respect to $\hat{q}_{\pi_i}$

$$\pi_{i+1}(a|s) = \begin{cases} 1 - \epsilon + \epsilon/K & \text{for a } = argmax_{a'} \hat{q}_{\pi_i}(s, a') \\ \epsilon/K & else \end{cases}$$

Value iteration - idea

Bellman optimization equations

$$q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \tag{49}$$

$$= \int \int p(r, s' | s, a)(r + \gamma \max_{a'} q_*(s', a')) ds' dr \tag{50}$$

Value iteration - algorithm (Q-learning)

Without function approximation

$$Q[s,a] \leftarrow Q[s,a] + \alpha((R + \max_{a'} Q[S',a']) - Q[s,a]) \tag{51}$$

With function approximation we have the update

$$\lambda \leftarrow \lambda + \alpha\big((R + \max_{a'} q_\eta(S',a')) - q_\eta(s,a)\big)\nabla_\eta q_\eta(s,a) \tag{52}$$

Subsection 3

Actor-critic - policy and value based control

Actor-critic

- Combine policy and value based control
- In policy gradient, scale the gradient by something "smarter" than observed reward G_t.
- Critic which judges how good each action is

Section 4

Conclusion

Conclusion

- + Very general and powerful framework
- + Some great success stories in games and robotics
- - Computationally expensive
- - May need simulated environment to learn
- - May get undesirable solution to problem
    - Think "Dilbert" (or worse!)