Policy Gradient Methods

Spring 2021

Case Study 5, Option 2: Tic-Tac-Go

Your task it to write a function

```
moveX(blocked, Xmoves, Omoves) {
 ...
 return(nextX)
}
```

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Language templates

```r
# R
# blocked: vector of integers
# Xmoves: vector of integers
# Ymoves: vector of integers
moveX <- function(blocked, Xmoves, Ymoves) {
  ...
  return(nextX)
}

```

```python
# Python
# blocked: list of integers
# Xmoves: list of integers
# Ymoves: list of integers
def moveX(blocked, Xmoves, Ymoves):
  ...
  return nextX

```

```matlab
% Matlab
% blocked: array of integers
% Xmoves: array of integers
% Ymoves: array of integers
function [nextX] = moveX(blocked, Xmoves, Ymoves)
  ...
end

```

Implicit vs. Explicit Policy Methods

- So far we've used an *implicit* policy for selecting actions.
- Given a value function $V(s)$, the optimal policy at state s_i selects

$$\arg\max_a \mathbb{E}\left\{r_i + V(s_{i+1})\right\}$$

or, for Q-factors,

$$\arg\max_a Q(s_i, a).$$

Implicit vs. Explicit Policy Methods

- So far we've used an *implicit* policy for selecting actions.
- Given a value function $V(s)$, the optimal policy at state s_i selects

  $$\arg\max_a \mathbb{E}\left\{r_i + V(s_{i+1})\right\}$$

  or, for Q-factors,

  $$\arg\max_a Q(s_i, a).$$

- This approach requires that we know V or Q, or at least can approximate them online.
- Sometimes it is easier to learn an optimal policy directly.

Policies

- Recall that a policy $\pi(a, s)$ returns the probability of selecting action a.
- Formally, the implicit policy on the previous slide was

$$\pi(a_i, s_i) = \begin{cases} 1 & a_i = \arg\max_a Q(s_i, a) \\ 0 & \text{otherwise} \end{cases}$$

Policies

- Recall that a policy $\pi(a, s)$ returns the probability of selecting action a.
- Formally, the implicit policy on the previous slide was

$$\pi(a_i, s_i) = \begin{cases} 1 & a_i = \arg\max_a Q(s_i, a) \\ 0 & \text{otherwise} \end{cases}$$

- Other policies return a distribution over all possible actions, and the agent selects an action based on these probabilities.

Policies

- Recall that a policy $\pi(a, s)$ returns the probability of selecting action a.
- Formally, the implicit policy on the previous slide was

$$\pi(a_i, s_i) = \begin{cases} 1 & a_i = \arg\max_a Q(s_i, a) \\ 0 & \text{otherwise} \end{cases}$$

- Other policies return a distribution over all possible actions, and the agent selects an action based on these probabilities.
- Often policies are *parameterized* by a vector of tunable parameters θ, i.e. $\pi = \pi(a, s|\theta)$.

Policy example: The newspaper problem

- Imagine you run a small newspaper that sells papers daily at newsstands (no subscriptions).
- Each night you need to decide how many papers to print for the next day. If you print too many, you may lose money. If you print too few, you lose sales.

Policy example: The newspaper problem

- Imagine you run a small newspaper that sells papers daily at newsstands (no subscriptions).
- Each night you need to decide how many papers to print for the next day. If you print too many, you may lose money. If you print too few, you lose sales.
- Definitions:
  - The price of a newspaper is p.
  - Each paper costs c to print. (Assume $c < p$.)
  - Each day the demand for your papers is D. You do not know this value ahead of time; it is a random variable.
  - The policy to learn is θ, the number of papers to print each day.

Learning a value for θ.

The *reward* is the profit for each day:

$$\begin{aligned} r &= \text{revenue} - \text{expenses} \\ &= p \min\{\theta, D\} - c\,\theta \end{aligned}$$

Learning a value for θ.

The *reward* is the profit for each day:

$$\begin{aligned} r &= \text{revenue} - \text{expenses} \\ &= p\min\{\theta, D\} - c\,\theta \end{aligned}$$

We can compute the stochastic gradient of the daily reward:

$$\frac{dr}{d\theta} = \begin{cases} p - c & \text{if } \theta < D \\ -c & \text{if } \theta > D \end{cases}$$

Learning a value for θ.

The *reward* is the profit for each day:

$$\begin{aligned} r &= \text{revenue} - \text{expenses} \\ &= p \min\{\theta, D\} - c\,\theta \end{aligned}$$

We can compute the stochastic gradient of the daily reward:

$$\frac{dr}{d\theta} = \begin{cases} p - c & \text{if } \theta < D \\ -c & \text{if } \theta > D \end{cases}$$

Over time we can learn an optimal policy by stochastic steepest ascent

$$\theta^{(k+1)} = \theta^{(k)} + \alpha \frac{d}{d\theta} r(\theta^{(k)})$$

for some learning rate α.

The REINFORCE algorithm

One of the earlier policy gradient methods was REINFORCE (Williams 1992).
It is still used today for finding policies from experience.

The REINFORCE algorithm

One of the earlier policy gradient methods was REINFORCE (Williams 1992). It is still used today for finding policies from experience.

1. Initialize the policy parameters θ, possibly to random values.
2. Generate a trajectory $s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$.
3. foreach $i \in \{0, \ldots, T\}$:
    - Calculate the return $R_i = r_i + r_{i+1} + \cdots + r_T$.
    - $\theta \leftarrow \theta + R_i \nabla_\theta \log \pi(a_i, s_i|\theta)$
4. Go to step $\#2$ and repeat.

The REINFORCE algorithm

One of the earlier policy gradient methods was REINFORCE (Williams 1992). It is still used today for finding policies from experience.

1. Initialize the policy parameters θ, possibly to random values.
2. Generate a trajectory $s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T, r_T$.
3. foreach $i \in \{0, \ldots, T\}$:
   - Calculate the return $R_i = r_i + r_{i+1} + \cdots + r_T$.
   - $\theta \leftarrow \theta + R_i \nabla_\theta \log \pi(a_i, s_i | \theta)$
4. Go to step $\#2$ and repeat.

Note that the policy π can be any parameterized function, including a neural network. The details of the parameter update change based on the structure of π, e.g. by using backpropagation.

Summary

- Policy gradient methods learn policies directly from experience.
- Stochastic search methods can find simple policies, and REINFORCE works well for complex policies.

Summary

- Policy gradient methods learn policies directly from experience.
- Stochastic search methods can find simple policies, and REINFORCE works well for complex policies.
- **Next time:** AlphaGo has a value network, a policy network, and tree search (rollout). How do these all fit together?