Outline

- Review: Q-Learning
- Policy Optimization
- Actor-Critic
(Tabular) Q-Learning

Algorithm:

Start with $Q_0(s, a)$ for all $s, a$.
Get initial state $s$
For $k = 1, 2, ...$ till convergence
    Sample action $a$, get next state $s'$
    If $s'$ is terminal:
        $\text{target} = R(s, a, s')$
        Sample new initial state $s'$
    else:
        $\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
        $Q_{k+1}(s, a) \leftarrow (1 - \alpha)Q_k(s, a) + \alpha [\text{target}]$
        $s \leftarrow s'$
Approximate Q-Learning

- Instead of a table, we have a parametrized Q function:  $Q_\theta(s, a)$
  - Can be a linear function in features:
    $$Q_\theta(s, a) = \theta_0 f_0(s, a) + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$
  - Or a complicated neural net

- Learning rule:
  - Remember:  $\text{target}(s') = R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$
  - Update:
    $$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[ \frac{1}{2} (Q_\theta(s, a) - \text{target}(s'))^2 \right]_{\theta=\theta_k}$$
Connection to Tabular Q-Learning

- Suppose $\theta \in \mathbb{R}^{|S| \times |A|}$, $Q_\theta(s, a) \equiv \theta_{sa}$

$$\nabla_{\theta_{sa}} \left[ \frac{1}{2} (Q_\theta(s, a) - \text{target}(s'))^2 \right]$$

$$= \nabla_{\theta_{sa}} \left[ \frac{1}{2} (\theta_{sa} - \text{target}(s'))^2 \right]$$

$$= \theta_{sa} - \text{target}(s')$$

- Plug into update: $\theta_{sa} \leftarrow \theta_{sa} - \alpha (\theta_{sa} - \text{target}(s'))$

$$= (1 - \alpha)\theta_{sa} + \alpha [\text{target}(s')]$$

- Compare with Tabular Q-Learning update:

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha)Q_k(s, a) + \alpha [\text{target}(s')]$$
Policy Optimization?

- Often the policy can be simpler than Q or V
  - E.g., Robotic grasp
- V: doesn’t prescribe actions
  - We need the dynamic model (+ compute 1 Bellman back-up)
- Q: need to be able to efficiently find the best action for every Q state
  - Challenge: What happens when actions are high-dimensional or continuous
Policy Optimization

- Solution: learn policies that maximize rewards, not the values that predict them
  - On policy learning – learn directly from actions
  - Any model that can be trained, could be a policy: Allows continuous action spaces, learning a stochastic policy

- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights
Policy Search

- **Simplest policy search:**
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before

- **Problems:**
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical

- Better methods exploit lookahead structure, sample wisely, change multiple parameters...
RL: Learning Manipulation

Levine*, Finn*, Darrell, Abbeel, JMLR 2016
Policy Optimization Notation

- Consider control policy parameterized by parameter vector $\theta$

$$\max_{\theta} \mathbb{E}\left[ \sum_{t=0}^{H} R(s_t) | \pi_\theta \right]$$

- Stochastic policy class (smoothes out the problem):

$$\pi_\theta(u|s) : \text{probability of action } u \text{ in state } s$$

Figure source: Sutton & Barto, 1998]
Example (Playing Pong)

Suppose we had the training labels…
(we know what to do in any state)

(x1, UP)
(x2, DOWN)
(x3, UP)
...

\[
\sum_i \log p(y_i | x_i)
\]

maximize:

![Diagram of raw pixels and hidden layer with probability of moving UP]
Reinforcement Learning

Let’s just act according to our current policy...

Rollout the policy and collect an episode
Pretend every action we took here was the correct label.

maximize: \( \log p(y_i \mid x_i) \)

Pretend every action we took here was the wrong label.

maximize: \( -(1) \cdot \log p(y_i \mid x_i) \)
Supervised Learning

maximize:

$$\sum_i \log p(y_i | x_i)$$

For images $x_i$ and their labels $y_i$.

---

Reinforcement Learning

1) we have no labels so we sample:

$$y_i \sim p(\cdot | x_i)$$

2) once we collect a batch of rollouts: maximize:

$$\sum_i A_i \ast \log p(y_i | x_i)$$

We call this the advantage, it’s a number, like +1.0 or -1.0 based on how this action eventually turned out.
**Supervised Learning**

maximize:

\[ \sum_i \log p(y_i | x_i) \]

For images \( x_i \) and their labels \( y_i \).

---

**Reinforcement Learning**

1) we have no labels so we sample:

\[ y_i \sim p(\cdot | x_i) \]

2) once we collect a batch of rollouts:

maximize:

\[ \sum_i A_i \ast \log p(y_i | x_i) \]

+ve advantage will make that action more likely in the future, for that state.

-ve advantage will make that action less likely in the future, for that state.
Vanilla Policy Optimization Algorithm

Initialize the policy
For iterations=1,2,...
    Collect a set of trajectories by executing the current policy
    At each time step of the trajectory, compute the advantage $A_i$
    Update the policy using a policy gradient estimate, which is
    $\nabla_\theta A_i.\log P(y_i|x_i, \theta)$
End for
Policy Optimization vs. Dynamic Programming

- Policy Optimization
  - Policy Gradients
  - Actor-Critic Methods

- Dynamic Programming
  - Policy Iteration
  - Value Iteration
  - Q-Learning
  - modified policy iteration
Asynchronous Advantage Actor-Critic (A3C)

- **Asynchronous:**
  - Multiple workers (agents)

- **Actor-Critic:**
  - Policy optimization + Value Iteration
  - Networks for Policy and V function
A3C Actor-Critic

- **Asynchronous:**
  - Multiple workers to increase efficiency and diversity
    - Unlike Q-Learning and Policy gradient with single agent
  - Global Network for policy
    - Independent local networks for each worker

- **Actor-Critic:**
  - Actor: computes policy(s): probability over actions
  - Critic: Computes $V(s)$: how good a certain state is to be in
A3C

- **Constructing the global network**
  - convolutional layers to process spatial dependencies
  - LSTM layer to process temporal dependencies
  - value and policy output layers.
Iteration 0
RL: Learning Soccer
[Bansal et al, 2017]
Conclusion

- Done with Search and Control
- Move on to Probabilistic Inference