Lecture 11: Deep Q Learning

1 Introduction

Admin:

- Midterm on Thur. Will be in Friend 101. If you haven't received an email about accommodations, please email me or ODS.

1.1 Course Project

1.2 Review: Q-learning

While SARSA and expected SARSA give us Q^π, Q-learning gives us $Q^\star$.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r(s,a) + \gamma \max_{a'} Q(s',a')\right). \tag{1}$$

Q-learning is off-policy: it is trying to estimate $Q^\star$ even when transitions are from suboptimal policy. It is not for evaluating a fixed policy.

Deep Q-learning.

- can't represent all states and action in a big table, so we'll need function approximation.
- we'll want $Q_\theta(s,a) = Q^\star(s,a)$.
- instead of trying to learn all the entries in this table, we're trying to learn these parameters
- architecture: state is input, output is $Q(s,a)$ for all actions a.

Using θ_i to denote the weights at iteration i, our updates are:

$$\theta_{i+1} \leftarrow \arg\min_{\theta_{i+1}} \frac{1}{2}\left(\underbrace{Q_{\theta_{i+1}}(s,a)}_{\text{prediction}} - \underbrace{(r(s,a) + \gamma \max_{a'} Q_{\theta_i}(s',a'))}_{\text{target / label}}\right)^2. \tag{2}$$

Challenges with deep Q-learning: the deadly triad Sutton and Barto (2018, Chapter 11.10): "*The potential for off-policy learning remains tantalizing, the best way to achieve it still a mystery.*"

- Bootstrapping
- Function approximation
- Off-policy

1.3 Goals for today

By the end of today's class, you'll be able to

- explain why deep q-learning tends to overestimate Q-values, and how to combat it
- why deep q-learning can be unstable, and how to combat it.

2 DQN [2]

Context: 2015 paper from DeepMind got deep Q-learning to work well.

- One of the first successes of RL on high-dimensional inputs. Outperformed many humans on Atari video games (breakout, pong).
- Convergence of deep learning and reinforcement learning (hence, this area is now called deep RL).

Architecture:

- 3 conv layers, 2 fc layers
- frame stacking (4 frames), raw pixels
- actions are 18 joystick positions/buttons
- reward is change in score

Loss:

$$\min(Q_\theta(s,a) - y)^2 \qquad \text{where} \quad Y = r + \gamma \max_{a'} Q_{\theta_{\text{target}}}(s', a'). \tag{3}$$

Update $\theta_{\text{target}} \leftarrow \theta$ periodically, or $\theta_{\text{target}} \leftarrow \eta\theta_{\text{target}} + (1-\eta)\theta$.

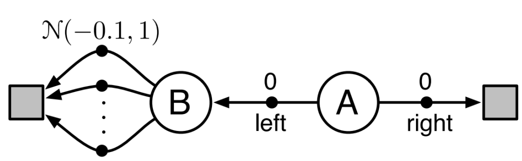
Exploration: Just like in MABs, we need a mechanism for exploration. DQN does epsilon-greedy.

Experience replay.

- building a dataset from the agent's own experience. Note that this is qualitatively different from supervised learning (e.g., behavioral cloning), where data is from expert
- circular buffer.
- updates are done on random mini-batches of data sampled uniformly.
- benefits: more data efficiency (each transition used in multiple gradient updates, not just thrown away after its first use) and decorrelates transitions in each batch
- there's a lot of work on non-uniform sampling strategies (e.g., prioritized experience replay); unclear how important this is in most settings.

2.1 Two Additional Tricks

Combatting maximization bias. We'll inherently have some noise in our Q value estimates, sometimes overestimate sometimes understimates. But, when we take the max in Q-learning, we're going to get an overestimate.



The Double Q-learning [1] trick was introduced in 2010. The key idea is to learn two separate value functions: one to select a', and another to estimate $Q(s', a')$.

$$a' \leftarrow \arg\max_{a'} Q^A(s', a') \qquad \text{(action selection)}$$
$$y \leftarrow r(s,a) + \gamma Q^B(s', a'). \qquad \text{(action evaluation)}$$

You randomly choose which Q to use for arg max, and use the other for the max Note that the argmax and max used in the TD target are now decorrelated, which helps a lot.

After the DQN paper came out in 2015, double DQN was applied on top [3]. Recall that DQN uses target networks, so we now use the target network as the second network:

$$a' \leftarrow \arg\max_{a'} Q_\theta(s', a') \qquad \text{(action selection)}$$
$$y \leftarrow r(s, a) + \gamma Q_{\theta_{\text{target}}}(s', a'). \qquad \text{(action evaluation)}$$

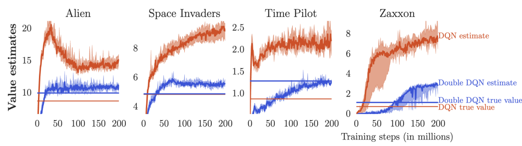


Figure 1: Double DQN

Multi-step returns.

$$y = R^n + \gamma^n \max_{a'} Q(s_{t+n}, a'). \tag{4}$$

3 Policy vs value-based methods.

- Q-learning is value based – directly updates Q/v, only reads out policy at the end
- policy gradient / REINFORCE is policy-based, directly updates a policy without estimating values
- both are model-free – don't need simulator/model
- Policy gradient
  - Pro: unbiased gradient estimate
  - Pro: handles high-dim actions
  - Con: on-policy, high-variance (hence poor sample efficiency)
- Q-earning
  - Pro: low variance updates, more sample efficient
  - Pro: can use off-policy data (not that we used a replay buffer with data from different policies, but we couldn't do this with REINFORCE (we had to sample data from policy itself)).
- Moving forward, after spring break, we'll look at hybrid methods that retain the sample effiency of Q-learning while also scaling to high-dim actions.

References

[1] Hasselt, H. (2010). Double q-learning. *Advances in neural information processing systems*, 23.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[3] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.