B9140 Dynamic Programming & Reinforcement Learning Lecture 7 - 10/30/17

Introduction to Reinforcement Learning

Lecturer: Daniel Russo *Scribe: Nikhil Kotecha, Ryan McNellis, Min-hwan Oh*

0 From Previous Lecture

Last time, we discussed least-squares value iteration with stochastic gradient descent given the history of data $\mathcal{H} = \{(s_n, r_n, s_{n+1}) | n \leq N\}$

Algorithm 1: Least-squares VI with SGD

```
Input: $\theta$, $(\alpha_t\colon t \in \mathbb{N})$
for $k = 0, 1, 2, ...$ do
    $\theta = \theta_k$
    repeat
        Sample $(s, r, s') \sim H$
        $y = r + \gamma V_{\theta_k}(s')$
        $\theta = \theta - \alpha_t \nabla (V_\theta(s) - y)^2$
        $t = t + 1$
    until convergence;
    $\theta_{k+1} = \theta$
end
```

In this lecture, we will be bridging the gap between this and DeepMind's DQN. Note that in summary there are three main differences:

1. Incremental training: θ_k's are updated frequently (perhaps every period) rather than waiting for convergence
2. Learning a *state-action* value function (Q-function)
3. Adapting the policy as data is collected (changes how future data is collected)

1 Incremental Training

1.1 Temporal Difference Learning

A fully-online analogue of least-squares value iteration.

Algorithm 2: Temporal Difference Learning

```
Input: $\mu$, $\theta$, $(\alpha_t\colon t \in \mathbb{N})$ (step-wise sequence)
for $n = 0, 1, 2, ...$ do
    Observe $s_n$, play $a_n = \mu(s_n)$ (See state, play action that policy tells to play in state.)
    Observe $(r_n,\ s_{n+1})$ (Outcome: instantaneous reveal, next state.)
    $y = r_n + \gamma V_\theta(s_{n+1})$ (Under current parameter, one-step look ahead value.)
    $\theta = \theta - \alpha_n \nabla (V_\theta(s_n) - y)^2$ (Gradient step.)
end
```

This mechanism is biologically plausible: instantaneous outcomes can be labeled good or bad with the goal of trying to predict if outcomes are good or bad. The realized y depends on the parameter, akin to trying to predict a moving target.

Temporal Difference (TD) with linear function approximation converges to θ^* solving:

$$\Phi\theta^* = \Pi T_\mu \Phi\theta^* \tag{1}$$

Result: (Tsitsiklis & Van Roy, 1997[1])

-Fixed point of:

$$V_{\theta_{k+1}} = \Pi T_\mu V_{\theta_k} \tag{2}$$

This relies on the theory of stochastic approximation and fact that ΠT_μ is a contraction (recall proof from previous class).

Essence of the result in 3 steps:

Step 1, Calculate gradient:

$$\frac{\partial}{\partial\theta}\frac{(V_\theta(S_n)-y)^2}{2} = \left[\phi(S_n)^\top\theta - (r_n + \gamma\phi(S_{n+1})^\top\theta)\right]\phi(S_n) = g_n(\theta) \tag{3}$$

In words, the gradient of the loss is equal to the predicted value less the reward and discounted predicted value with the feature value in the next state. This can also be expressed in the following equation, in which $g_n(\theta)$ is a random variable that depends on the state and the realized reward and state.

$$\frac{\partial}{\partial\theta}\frac{(V_\theta(S_n)-y)^2}{2} = g_n(\theta) \tag{4}$$

Step 2, Denoise:

$$\mathbb{E}_0[(g_n(\theta)] = \Phi^\top D_\pi\,(T_\mu\Phi\theta - \Phi\theta) \tag{5}$$

Here, the expectation is taken over the steady state. In the RHS, Φ represents the features, D_π is a diagonal matrix with the steady state probabilites on the diagonal. Inside the parenthetical looks like the average Bellman error in prediction as measured in features.

Step 3, Solution to fixed point equation:

$$(\theta-\theta^*)^\top\mathbb{E}_0[g_n(\theta)] > 0 \tag{6}$$

Above is the essence of the result: looking at the convergence point, which is the solution to the fixed point equation.

[1] Tsitsiklis, J. and Van Roy, B. (1997). An Analysis of Temporal-Difference Learning with Function Approximation. IEEE Transactions on Automatic Control, 42(5): 674- 690.

1.2 Stochastic Approximation

History: Robbins and Monro, 1951 wrote a paper entitled "A Stochastic Approximation Method."[2] These ideas are widely used in control systems, signal processing, stochastic simulation, time series, and Machine Learning (today).

Incremental Mean: observe $X_1, X_2, ... i.i.d$ with mean θ

$$\hat{\theta}_n = \frac{1}{n}\sum_{i}^{n} X_i = \frac{1}{n}\left(\sum_{i}^{n-1} X_i + X_n\right) = \hat{\theta}_{n-1} - \frac{1}{n}\left(\hat{\theta}_{n-1} - X_n\right) = \hat{\theta}_{n-1} - \alpha_n(g_n) \tag{7}$$

Here time is represented by n. In the first step the empirical average is taken, then rewritten to exactly compute the mean. This results in the new mean being equal to the last mean less the difference of the mean estimate and the observation.

Some key observations:

Observation 1, the sum of squares is finite:

$$\sum_{n=1}^{\infty} \alpha_n = \infty, \qquad \sum_{n=1}^{\infty} \alpha_n^2 < \infty \tag{8}$$

Observation 2, average updates go in the right direction:

$$\mathbb{E}[g_n | \mathcal{F}_{n-1}] = (\hat{\theta}_{n-1} - \theta) \tag{9}$$

To put these observations together, use the Martingale Convergence Theorem to show $\hat{\theta}_n \to \theta$.

As it turns out, the above procedure is equivalent to applying Stochastic Gradient Descent (SGD) to the objective function $\mathbb{E}[(\theta - X)^2/2]$ using step size $\alpha_n = 1/n$. More generally, SGD is used to find the parameter θ which minimizes the *nonnegative* loss function $\ell(\theta) = \mathbb{E}_{\xi}[f(\theta, \xi)]$ for some $f(\cdot, \cdot)$. It is assumed that this expectation cannot be computed directly. However, we can use SGD to optimize over the loss function with respect to *i.i.d.* samples of the random variable ξ. The SGD algorithm is as follows:

Algorithm 3: Stochastic Gradient Descent (SGD)

```
Input: step size $\alpha_t$, starting parameter $\theta_1$
for $t = 1, 2, ...$ do
    Sample $\xi_t$ (i.i.d.)
    Compute $g_t = \nabla_\theta f(\theta, \xi_t)|_{\theta=\theta_t}$
    $\theta_{t+1} = \theta_t - \alpha_t g_t$
end
```

Let $\nabla \ell(\theta_t)$ be shorthand notation for $\nabla \ell(\theta)|_{\theta=\theta_t}$. If the step size α_t is chosen appropriately, then SGD converges to a locally-optimal solution:

Theorem 1. *$||\nabla \ell(\theta_t)|| \to 0$ as $t \to \infty$ if the following conditions are satisfied:*

1. $\sum_{t=1}^{\infty} \alpha_t = \infty$
2. $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

[2]Robbins, H., & Monro, S. (1951). A stochastic approximation method. The annals of mathematical statistics, 400-407.

Proof (sketch): Let $\mathcal{F}_t = \{\xi_s : s \leq t\}$. Then, $\mathbb{E}[g_t|\mathcal{F}_{t-1}] = \nabla\ell(\theta_t)$: given the entire history of data $\mathcal{F}_{t-1}$, the expected value of the noisy gradient g_t is equal to the true gradient $\nabla\ell(\theta)|_{\theta=\theta_t}$. One can also show that:

$$\ell(\theta_{t+1}) = \ell(\theta_t) - \alpha_t \nabla\ell(\theta_t)^\top g_t + O(\alpha_t^2)$$

Combining these two observations,

$$\mathbb{E}[\ell(\theta_t) - \ell(\theta_{t+1})|F_{t-1}] = \alpha_t ||\nabla\ell(\theta_t)||^2 + O(\alpha_t^2)$$

Assume by contradiction that $\liminf_{t\to\infty} ||\nabla\ell(\theta_t)||^2 = c > 0$. Then, for large t, $\mathbb{E}[\ell(\theta_t)]$ decreases by $\alpha_t c + O(\alpha_t^2)$ each iteration. This implies that $\mathbb{E}[\ell(\theta_t)]$ approaches $-\infty$, given the theorem conditions $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$. However, this violates the non-negativity of $\ell(\theta)$. Thus, it must be the case that $\liminf_{t\to\infty} ||\nabla\ell(\theta_t)||^2 = 0$.

2 Using State-Action Value Functions

Up to this point in class, we have focused on on the estimation of the value functions $V^*(s)$ corresponding to the optimal policy. However, the reinforcement-learning literature instead focuses on estimating the "Q-functions" $Q^*(s,a)$, which can be thought of as the "value" of a state-action pair. This shift in focus is due to the fact that reinforcement-learning algorithms need to do more than simply evaluate a fixed policy – they also need to control the data-collection process through the actions they take (this will be emphasized in the next section). However, as we will show, the methodology we have studied for estimating value functions extends easily to estimating Q-functions.

First, note that the value function $V^*(s)$ can be computed from the Q-function $Q^*(s,a)$ in the following way:

$$V^*(s) = \max_{a\in A} Q^*(s,a) \tag{10}$$

The Q-functions obey the following system of equations:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s')V^*(s') \tag{11}$$

In words, $Q^*(s,a)$ represents the reward from taking action a in state s plus the expected cost-to-go from taking actions according to the optimal policy. As it turns out, the optimal policy can be derived from knowing Q^*:

$$\mu^*(s) = \arg\max_{a\in A} Q^*(s,a)$$

Thus, if we can estimate Q^*, then we can simply read off the optimal policy. Note that we can define a Q-function with respect to any policy μ, rather than simply the optimal policy:

$$Q_\mu(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s')V_\mu(s')\,,$$

or, in words, the reward from taking action a in state s plus the expected cost-to-go from taking actions according to the policy μ.

As mentioned previously, much of the theory about value functions extends to Q-functions. For example, Q^* obeys its own Bellman equations:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_{a\in A} Q^*(s',a)$$

Note that the above equation was obtained from simply plugging equation (10) into equation (11).

Another slight change from our usual formulation is to consider stochastic policies. When we studied dynamic programming theory, we did not consider such policies because they are never optimal for Markov Decision Processes. However, in reinforcement learning, stochastic policies are useful for exploration – for allowing the algorithm to see new states. We define the stochastic policy μ as follows:

$$\mu(s,a) = \text{probability of choosing action } a \text{ when in state } s$$

Note in particular that $\sum_a \mu(s,a) = 1$. The "Bellman equations" for Q_μ with respect to a stochastic μ are as follows:

$$Q_\mu(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \left[\max_{a' \in A} \mu(s',a') Q_\mu(s',a') \right] \tag{12}$$

We can define a "Bellman operator" F_μ with respect to the above equation. Using F_μ, we can express equation (12) using the shorthand notation $Q_\mu = F_\mu Q_\mu$.

Given the Bellman equations above, we can apply similar techniques to learning Q-functions as we used when learning value functions. For example, below is a policy-iteration algorithm for learning Q-functions:

```
Algorithm 4: Policy Iteration (for estimating Q*)
  Input: starting policy $\mu_0$
  for $k = 0, 1, 2, ...$ do
      Evaluate $Q_{\mu_k}$ from solving the Bellman equation $Q_{\mu_k} = F_{\mu_k} Q_{\mu_k}$
      Improve the policy: $\mu_{k+1}(s) = \arg\max_a Q_{\mu_k}(s,a)$
  end
```

In settings where the state-action space is large, we can approximate Q_μ through a linear function $Q_\theta = \Phi\theta$, where $Q_\theta(s,a) = \phi(s,a)^\top \theta$. To learn the best value of θ, one can apply an algorithm called SARSA, which is very similar to temporal difference learning:

```
Algorithm 5: SARSA
  Input: starting parameter $\theta$, policy $\mu$, step sizes $\alpha_t$
  for $n = 0, 1, 2, ...$ do
      Observe $s_n$ and sample $a_n \sim \mu(s_n, \cdot)$
      Observe $(r_n, s_{n+1})$ and sample $a_{n+1} \sim \mu(s_{n+1}, \cdot)$
      $y = r_n + \gamma Q_\theta(s_{n+1}, a_{n+1})$
      $\theta = \theta - \alpha_t \nabla (Q_\theta(s_n, a_n) - y)^2$
  end
```

Note that in temporal difference learning, we perform updates with respect to the tuple (s_n, a_n, r_n). In SARSA, we now perform updates with respect to the vector $(s_n, a_n, r_n, s_{n+1}, a_{n+1})$.

The convergence theory is identical when considering linear approximations of Q_θ. In the limit,

$$\Phi\theta^* = \Pi F_\mu \big(\Phi\theta^* \big) .$$

3 Adaptive Control with Value Function Approximation

We are now going to look at adaptive control with value function approximation or Q-function approximation. Where does policy μ come from? If we are given a Q-function, we can provide a policy that we think is better with respect to the given Q-function. We can gather data using this policy. Then we can re-evaluate our Q-function, and then get a better policy and so on.

Algorithm 6: Approximate Policy Iteration

```
Input: starting policy $\mu_0$
for $k = 0, 1, 2, ...$ do
    Approximate $Q_{\mu_k} = Q_{\theta_k}$ (Run SARSA until convergence to $\theta_k$ with $\Phi\theta_k = \Pi F_{\mu_k}(\Phi\theta_k)$)
    Improve the policy: $\mu_{k+1}(s) = \arg\max_a Q_{\mu_k}(s,a)$
end
```

This is very similar to policy iteration. But it is not going to behave very well in general due to the nature of the approximation. Note that by following μ_k, we evaluate Q_{μ_k}. We approximate Q-functions by minimizing the predictive loss on the states we visit under the current policy. So we prioritize being accurate on those states and actions that we visit. And we might accidentally introduce errors in the evaluations of states and actions that we don't visit under current policy.

Policy "chattering" in approximate polity iteration

Example: Suppose we have 1 state with 2 actions. So we write $Q(a) = Q(s,a)$ for $a = 1, 2$, i.e. we get rid of s since there is only 1 state. We approximate $Q(a) \approx Q_\theta(a) = \Phi(a)\theta$, $\theta \in \mathbb{R}$.

Suppose we have $Q = (Q(1), Q(2)) = (-1, 1)$ i.e. if we play action 1, we get a reward of -1. If we play action 2, we get a reward of 1. Obviously, we want to play action 2. Now, suppose we have $\Phi = (2, 1)$, i.e. if $\theta > 0$, we think we should play action 1. And if $\theta < 0$, we think we should play action 2.

Let's look at what approximate policy iteration algorithm does.

Algorithm 7: Approximate Policy Iteration with Linear Approximation and 1 state

```
Input: $\theta_0$, $a_0 = \arg\max_a(\Phi\theta_0)(a)$
for $k = 0, 1, 2, ...$ do
    $\theta_{k+1} = \arg\min_\theta ((\Phi\theta)(a_k) - Q(a_k))^2$
    $a_{k+1} = \arg\max_a(\Phi\theta_k)(a)$
end
```

So what happens with $(\theta_0 = -\frac{1}{2})$?

$$\theta_0 = -\frac{1}{2} \Rightarrow Q_{\theta_0} = (-1, -\frac{1}{2}) \Rightarrow a_0 = 2$$
$$\Phi(2)\theta_1 = Q(2) \Rightarrow \theta_1 = 1 \Rightarrow a_1 = \arg\max_a Q_{\theta_1}(a) = 1$$
$$\Phi(1)\theta_2 = Q(1) \Rightarrow \theta_2 = -\frac{1}{2} \Rightarrow a_2 = 2$$

We can't represent the true value function well everywhere, i.e. fail to globally approximate the value function. When we gather data from the current policy, we fit the experience/episode well but end up fitting wrong on the states we are not currently visiting. When we go visit those states that we did poorly on now we may forget what we have fitted well previously. This goes in a loop as what happens in the example above. Policy chattering really happens!

4 A High Level View of DQN

- Action: 18 button / joystick control
- State: Stack of four 84×84 pixel frames (most recent 4 frames)
- Q_θ: Convolutional neural network with 4 hidden layers
- Clip reward: map to $(-1, 0, 1)$
- In training: ϵ-greedy action selection $\rightarrow$ 5 million frames of training data (38 days)

Two changes to Q-learning

1. Fixed target network and update every 10k frames (looks more like approximate policy iteration)
2. Experience replay (buffer of 1 million frames) $\rightarrow$ breaks correlation