Advanced Q Learning

Aaron Lou

June 2018

1 Correlation Problem

There is a correlation problem in our Q-learning algorithm. At each step, our model overfits. For example, a sinusoidal graph shows that our policy is overfitting at each period.

1.1 Synchronization

Similar to actor-critic algorithms, we can use both synchronized and asynchronized methods to update the gradient. This helps deal with the correlation problem by removing that aspect.

1.2 Replay Buffer

We can exploit the fact that Q-learning is off policy by first conglomerating a huge amount of data and training our Q-learning on this data. This has the benefit of being not correlated and having a larger batch size (to reduce variance). Our process is given by

1. Collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ and add it to our total dataset $\mathcal{B}$.
2. sample a batch (s_i, a_i, s'_i, r_i) from $\mathcal{B}$.
3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)]$.

where the 2 and 3 steps are repeated K times.

2 Target network

In order to deal with the "moving target" for gradient descent, we save a value of ϕ' and use that for our gradient descent algorithm. Our Q-learning with replay buffers and a target network is

1. Save target parameters $\phi' \leftarrow \phi$
   2. Collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy; add it to $\mathcal{B}$.
   3. Sample a batch (s_i, a_i, s'_i, r_i) from $\mathcal{B}$.

4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s_i', a_i')]$
where we iterate the outer loop is N times for updating before updating ϕ', and K times for a sample and gradient update.

3 "Classic" deep Q-learning algorithm- DQN

Our DQN algorithm is given as follows

1. Take some action a_i, observe (s_i, a_i, s_i', r_i) and add to $\mathcal{B}$
2. Sample minibatch $\{s_i, a_i, s_i', r_i\}$ from $\mathcal{B}$
3. Compute $y_i = r_I + \gamma \max_{a_i'} Q_{\phi'}(s_i, a_i))$ using $Q_{\phi'}$.
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i)$
5. Update $\phi' \leftarrow \phi$ every N steps.

Alternatively, we can remove the sense of "lag" by updating (similar to Polyak averaging)
$\phi' \leftarrow \tau\phi' + (1-\tau)\phi$ where $\tau = 0.999$

an overview of all methods is broken into three steps.

1. Data collection
2. Update target
3. Q - function regression

where Online Q-learning is all three same speed, DQN is where 2 is slow, and fitted Q-learning is where 3 is in a loop of 2 in a loop of 1.

4 Q-learning tricks

4.1 Double Q-learning

Oftentimes, $r_j + \max_{a_j'} Q_{\phi'}(s_j', a_j')$ overestimates the Q value because noise propagates the error. This comes from the max value, and we rewrite it as

$$\max_{a'} Q_{\phi'}(s', a') = Q_{\phi'}(s', \arg\max_{a'} Q_{\phi'}(s', a'))$$

Double Q-learning uses two networks to minimize the error:

$$Q_{\phi_A}(s, a) \leftarrow r + \gamma Q_{\phi_B}(s', \arg\max_{a'} Q_{\phi_A}(s'))$$

$$Q_{\phi_B}(s, a) \leftarrow r + \gamma Q_{\phi_A}(s', \arg\max_{a'} Q_{\phi_B}(s'))$$

The two Q are differently noisy, which means that the error doesn't propagate. In practice, we use ϕ' and ϕ as our two networks and have

$$y = r + \gamma Q_{\phi'}(s', \arg\max_{a'} Q_\phi(s', a'))$$

4.2 Multi-step Returns

Our Q-learning target is $y_{j,t} = r_{j,t} + \gamma \max_{a_{j,t+1}} Q_{\phi'}(s_{j,t+1}, a_{j,t+1})$. We construct multi-step targets, in particular, N-step is given by

$$y_{j,t} = \sum_{t'=t}^{t+N-1} r_{j,t'} + \gamma^N \max_{a_{j,t+1}} Q_{\phi'}(s_{j,t+N}, a_{j,t+N})$$

This only works when we are on-policy. We need the previous values, and so we fix this by

1. Ignoring the problem (often works).
2. Cut the trace, and dynamically choose N to get on-policy data
3. Importance sampling.

4.3 Q-learning with continuous actions

We with to find $\max_{a'_j} Q_{\phi'}(s'_j, a'_j)$. We have a couple of optimization techniques.

1. Gradient based optimization (SGD), but this is a bit slow. Stochastic Optimization also has many methods. The simple solution is we sample a bunch of times and find the max

$$\max_a Q(s,a) \approx \max\{Q(s,a_1)\dots, Q(s,a_N)\}$$

This works for up to 40 dimensions. More accurate solutions are CEM(cross-entropy method) and CMA-ES.

2. Use a function class that is easy to optimize. For example, if we train a neural network to output μ, P, V from input S. Our output is

$$Q_\phi(s,a) = -\frac{1}{2}(a - \mu_\phi(s))^T P_\phi(s)(a - \mu_\phi(s)) + V_\phi(s)$$

these are called NAF(Normalized Advantage Functions) and we know that

$$\arg\max_a Q_\phi(s,a) = \mu_\phi(s) \;\; \max_a Q_\phi(s,a) = V_\phi(s)$$

3. Learn an approximate maximizer. This is called DDPG, and the idea is to train another network $\mu_\theta(s)$ s.t. $\mu_\theta(s) \approx \arg\max_a Q_\phi(s,a)$.

We can solve $\theta \leftarrow \arg\max_\theta Q_\phi(s, \mu_\theta(s))$ using $\frac{dQ_\phi}{d\theta} = \frac{da}{d\theta}\frac{dQ_\phi}{da}$. Our new target is $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, \mu_\theta(s'_j))$. Our full algorithm is

1. Take some action a_i and observe (s_i, a_i, s'_i, r_i) and add it $\mathcal{B}$.
2. Sample a mini-batch $\{s_j, a_j, s'_j, r_j\}$ from $\mathcal{B}$.
3. Compute $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, \mu_{\theta'}(s'_j))$ using target nets $Q_{\phi'}$ and μ_θ.
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_j, a_j)(Q_\phi(s_j, a_j) - y_j)$.
5. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(s_j)\frac{dQ_\phi}{da}(s_j, a)$.
6. update ϕ' and θ' using Polyak averaging.

4.4 Simple tips for Q-learning

- Q-learning is hard to stabilize, test on reliable tests before moving on to harder examples.
- Having a large replay buffers help improve stability.
- These methods take a lot of time to train.
- Start with a high exploration (epsilon) and reduce as time advances

4.5 Advanced tips for Q-learning

- The Bellman error gradient is very large. Clip gradients or use Huber loss (which approximates absolute values).

$$L(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| \leq \delta \\ \delta|x| - \frac{\delta^2}{2} & \text{otherwise} \end{cases}$$

- Double Q-learning is good and pretty easy to incorporate.
- N-step return is normally good, but has downsides.
- Have exploration and training rates reduce from high to low. The Adam optimizer works for this, as well.
- Run multiple random seeds, since our model is inconsistent between runs.