CHAPTER 1

DEEP REINFORCEMENT LEARNING

1 Neural Networks

...

2 Distributional Reinforcement Learning

Remember that we defined $\mathbb{P}^\pi_{s,a} := \mathbb{P}^\pi \otimes \delta_{S_0}(s) \otimes \delta_{A_0}(a)$ as the probability measure of the Markov reward process (S, A, R) started in (s, a). We define the random variable of the return under policy π as

$$Z^\pi := \sum_{t=0}^{\infty} \gamma^t R_t, \quad \gamma \in (0, 1).$$

Unlike the methods before, where we were interested in the expected reward $Q^\pi(s, a) = \mathbb{E}^\pi_{s,a}[Z^\pi]$, we are now interested in the distribution of these cumulative rewards. For that define

$$\eta^\pi_{s,a}(B) := \mathbb{P}^\pi_{s,a}(Z^\pi \in B), \quad B \in \mathcal{B}(\mathbb{R}).$$

In analogy to weak solutions for PDEs, i.e. in sense of distribution, the return distribution $\eta^\pi_{s,a}$ satisfies the Bellman equation in sense of distribution:

$$\forall \phi \in C_b(\mathbb{R}): \quad \int_{\mathbb{R}} \phi(z) d\eta^\pi_{s,a}(z) = \mathbb{E}^\pi_{s,a}[\int_{\mathbb{R}} \phi(R + \gamma z) d\eta_{S',A'}(z)].$$

We define $f_{r,\gamma}(z) := r + \gamma z$ and the push forward

$$((\eta^\pi_{s,a})_{f_{r,\gamma}})(B) := \eta^\pi_{s,a}(f^{-1}_{r,\gamma}(B)), \quad B \in \mathcal{B}(\mathbb{R}).$$

then, the above can be written as

$$\begin{aligned} &\forall \phi \in C_b(\mathbb{R}): \quad \int_{\mathbb{R}} \phi(z) d\eta^\pi_{s,a}(z) = \mathbb{E}^\pi_{s,a}[\int_{\mathbb{R}} \phi(R + \gamma z) d\eta_{S',A'}(z)] = \mathbb{E}^\pi_{s,a}[\int_{\mathbb{R}} \phi(z) d(\eta_{S',A'})_{f_{R,\gamma}}(z)] \\ \iff &\forall \phi \in C_b(\mathbb{R}): \quad \langle \phi, \eta^\pi_{s,a} \rangle = \mathbb{E}^\pi_{s,a}[\langle \phi, (\eta^\pi_{S',A'})_{f_{R,\gamma}} \rangle]. \end{aligned}$$

For any $\phi \in C_b(\mathbb{R})$ and any measure ν we have

$$\begin{aligned} (\nu * \delta_r)(\phi) &= \int_{\mathbb{R}} \phi(z) d(\nu * \delta_r)(z) = \int_{\mathbb{R}} \int_{\mathbb{R}} \phi(x + y) d\nu(x) d\delta_r(y) \\ &= \int_{\mathbb{R}} \phi(x + r) d\nu(x) = (\nu(\cdot - r))(\phi). \end{aligned}$$

Next, define $D_\gamma(x) := \gamma x$, then if $Z \sim \eta^\pi(x,a)$ then $\gamma Z \sim (\eta^\pi(s,a))_{D_\gamma}$. In total

$$Z\gamma + r \sim (\eta^\pi(s,a))_{D_\gamma}(\cdot - r) = ((\eta^\pi(s,a))_{D_\gamma} * \delta_r)$$

holds in sense of distribution. Now define the distributional Bellman operator in sense of distribution as

$$\mathcal{T}^\pi Z^\pi(s,a) = R(s,a) + \gamma Z^\pi(S', A'),$$

i.e. the distribution

$$\mathcal{T}^\pi : (\mathcal{P}(\mathbb{R}))^{\mathcal{S}\times\mathcal{A}} \to (\mathcal{P}(\mathbb{R}))^{\mathcal{S}\times\mathcal{A}}, (\nu_{s,a})_{(s,a)\in\mathcal{S}\times\mathcal{A}} \mapsto ((\mathcal{T}^\pi\nu)_{s,a})_{(s,a)\in\mathcal{S}\times\mathcal{A}}$$

is point wise defined for all $\phi \in C_b(\mathbb{R})$ as

$$(\mathcal{T}^\pi\nu)_{s,a}(\phi) := \mathbb{E}^\pi_{s,a}[((\nu(S',A'))_{D_\gamma} * \delta_R)(\phi)] = \sum_{s',a',r} \pi(a';s')p(s',r;s,a)((\nu_{s',a'})_{D_\gamma} * \delta_r)(\phi)$$

3 Deep Q-Networks

Deep Q-Networks (DQN) is a Q-learning variant that utilizes deep neural networks to approximate the Q-value. This algorithm addresses how to deal with large, continuous state action spaces $(\mathcal{S}, \mathcal{A})$. The idea is to reduce dimensionality by plugging in the state space $\mathcal{S}$ (e.g. pixels in an Atari game) into a convolutional neural layer, which is the fed into a feed forward neural network that outputs the Q-value for each action $a \in \mathcal{A}$. Problems with simple neural networks are

- They can forget
- Are unstable: small changes in Q-values can lead to big changes in the policy, i.e. the action distribution
- Correlation: In a trajectory τ_i of a rollout, the consecutive states s_t, s_{t+1} are highly correlated, which violates the i.i.d. assumption for estimating the expectation.
- The loss is non-stationary, since the target values depend on the parameters of the network itself, meaning
  $$\mathcal{L}(\theta) := \mathbb{E}[(\mathbb{E}[R + \gamma \max_{a'} Q^\theta(S', a')] - Q^\theta(S, A))^2],$$
  where both Q terms depend on θ.

In DQN these issues where solved by the following

- Experience Replay: A behavoiral policy π^b samples Rollouts, that are stored in a buffer $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^N$. In order to break the correlation between consecutive states, mini batches are sampled uniformly from this buffer to update the Q-network.
- Target Network: A second network with parameters θ^- is introduced, which is bootstrapped, i.e. every C steps the parameters of the Q-network are copied to the target network $\theta^- \leftarrow \theta$. This delay stabilizes the training.

Tabular Q-learning can be interpreted in the following two ways:

- ML view: Fit a target Q_n to the data Q^{π^*}. Define $Y_n := R + \gamma \max_{a'\in\mathcal{A}} Q_n(S', a')$ and
  $$\tilde{L}_n^Q := \mathbb{E}[(\mathbb{E}_{S,A}[Y_n] - Q_n(S,A))^2] = \mathbb{E}[(Y_n - Q_n(S,A))^2] + \mathbb{E}[\mathbb{V}_{S,A}(Y_n)] =: L_n^Q + \mathbb{E}[\mathbb{V}_{S,A}(Y_n)].$$

- RL view: We want to approximate the Bellman optimality operator

$$T^*Q(s,a) := \mathbb{E}_{s,a}[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')].$$

  Because it is a contraction in $\|\cdot\|_\infty$, the fixed point iteration converges.

Because the expectation is the minimizer of the L^2 error (variance), it holds that $\mathbb{E}[X] = \arg\min_\theta \mathbb{E}[(X-\theta)^2]$ which leads to

$$T^*Q(s,a) = \mathbb{E}_{s,a}[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')] = \arg\min_\theta \mathbb{E}_{s,a}[(R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') - \theta)^2].$$

Thus doing one step SGD (with step size $\frac{\alpha}{2}$???) on L_n^Q or doing the iteration for Q-learning yields the same scheme:

$$Q_{n+1}(s,a) = Q_n(s,a) + \alpha(y - Q_n(s,a)).$$

Remark 3.1

When doing Q-learning with neural networks, we no longer do the gradient on $(s,a) \in \mathcal{S} \times \mathcal{A}$, but instead on a smaller parameter space $\Theta \subset \mathcal{S} \times \mathcal{A}$. Additionally, parameterizing the target with a different set of parameters θ^- the gradient of the loss becomes

$$\nabla_\theta L_n^Q = \mathbb{E}[\nabla_\theta (Y_n^{\theta^-} - Q_n^\theta(S,A))^2] = -\mathbb{E}[2(Y_n^{\theta^-} - Q_n^\theta(S,A)) \nabla_\theta Q_n^\theta(S,A)].??? -$$

So the gradient step for SGD becomes

$$Q_{n+1}^\theta(s,a) = Q_n^\theta(s,a) + \alpha(y_n^{\theta^-} - Q_n^\theta(s,a)) \nabla_\theta Q_n^\theta(s,a).$$

4 Proximal Policy Gradient

5 Advantage Estimation

6 Variants to PPO

7 Stable Baselines3