TOPICS IN REINFORCEMENT LEARNING – SPRING 2023

TUTORIAL ON DQN, POLICY GRADIENTS AND PPO

DEEP Q-NETWORK (DQN)

DEEP Q-NETWORKS

- This is a **value-based** method.
- Essentially Q-learning using Neural Networks.
- Introduced in the landmark paper Playing Atari with Deep Reinforcement Learning from DeepMind

ALGORITHM

- Our neural network (also called the Q-network), **Q(s, a),** takes in a state vector **s** and an action vector **a** as inputs and predicts the Q-value of taking action **a** in state **s**.
- During training, tuples of (s, a, r, s') are collected following a policy π(s) = argmax Q(s, a)
- Using these samples, we update the Q network with the following loss calculated for each sample:

$$[Q(s, a) - (r + \gamma \max_{a'} Q(s', a'))]^2$$

ALGORITHM

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

POLICY GRADIENT METHODS

POLICY-BASED METHODS VS. VALUE-BASED METHODS

- Value based methods try to estimate a value function (either V(s) or Q(s,a)) using sample trajectories from the environment and use the estimated value function to obtain a policy, usually as an argmax of the value function.
- Policy based methods however aim to directly estimate the optimal policy from sample trajectories without learning a value function first.
- They have the advantage that they can learn arbitrary policies, even stochastic ones.
- Some mathematical guarantees exist for policy gradient methods which are difficult to show for value-based methods.

POLICY GRADIENT

- We use a neural network $\pi_\theta(a \mid s)$, parameterized by θ, which is called the **policy network.**
- The policy network takes in a state vector sand outputs a probability distribution over actions. $\pi(a \mid s)$ represents the probability of taking action a.
- During training, we collect samples (s, a, r, s') by sampling at each step from the output of $\pi(s)$.
- Let T be trajectory of (s, a, r, s') samples collected over an episode.
- Let R(T) = total discounted reward over the trajectory.
- We update our network so as to maximize R(T).
- That is, we take a **gradient ascent step** with the gradient as follows:

$$\nabla_\theta E[R(T)] = \nabla_\theta \sum p(T)R(T)$$

were p(T) is the probability of a trajectory $p(T) = \prod \pi_\theta(a \mid s)$

POLICY GRADIENT THEOREM

- The policy gradient theorem derives an expression for the policy gradient in terms of gradients of $\log(\pi_\theta(a \mid s))$, i.e. $\nabla_\theta \log(\pi_\theta(a \mid s))$.
- A proof can be found here: https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html

REINFORCE / Vanilla Policy Gradient

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

PROXIMAL POLICY OPTIMIZATION (PPO)

Actor Critic Methods

- Vanilla Policy Gradient (VPG) / REINFORCE is noisy and learning is slow.
- Actor Critic Methods improve over basic policy gradient by estimating the state-action value function Q(s,a) as well along with the policy function.
- This reduces the variance of vanilla policy gradient and makes for more stable learning

Proximal Policy Optimization

- Actor critic methods, despite the improvement over VPG remain unstable as large updates to the policy network can throw off the learning.
- To solve this problem, Trust Region Policy Optimization (TRPO) was introduced which constrains the update to the policy network parameters within a "trust region".
- TRPO however is computationally quite inefficient as it requires calculation of the Fishcher information matrix of the policy network parameters
- PPO provides a more efficient way to constrain the policy update within a trust region.
- More detailed explanation: https://spinningup.openai.com/en/latest/algorithms/ppo.html

Stable Baselines 3

https://stable-baselines3.readthedocs.io/en/master/

Contains implementations of REINFORCE, PPO, DQN and other SOTA algorithms