Markov Decision Processes (MDP)

esgl Hu

November 23, 2017

1 Playing Atari with Deep Reinforcment Learning [1]

Firstly, most successful deep learning applications to date have required large amounts of hand-labelled training data. ① RL algorithms, on the other hand, must be able to learn from **a scalar reward signal** that is frequently ***sparse, noisy and delayed***. The delay between actions and resulting rewards, with can be thousands of timesteps long, seems particularly daunting when compared to the direct association between inputs and tagets found in supervised learning. Another issue is that most ② deep learning algorithms assume the data samples to be ***independent***, while in reinforcement learning one typically encounters sequences of ***high correlated states***. Furthermore, ③ in RL the ***data distribution changes*** as the algorithm learns new behaviours, which can be problematic for deep learning methods that assume ***a fixed underlying distrbituion***.

④ To avaeviate the problems of **correlated data and non-stationary distributions**, it uses an **experience replay mechanism** which randomly samples previous transitions, and thereby smotths the training distribution over many past behaviors.

It was shown that combining model-free reinforcement learning algorithms such as Q-learning with non-linear function approximators, or indeed with off-policy learning could cause the Q-network to ***diverge***. Subsequently, the majority of work in reinforcement learning focused on linear function approximators with better convergence guarantees. [2]

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}[r + \gamma max_{a'} Q^*(s^{'}, a^{'})|s,a] \tag{1}$$

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2] \tag{2}$$

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}[(r + \gamma max_{a'} Q(s^{'}, a^{'};\theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)], \tag{3}$$

where $y_i = \mathbb{E}_{s'\sim\mathcal{E}}[r + \gamma max_{a'} Q(s^{'}, a^{'};\theta_{i-1})|s,a]$

The disadvatages of Q-learning

- Overestimate
- Underestimate

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory 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 do$t = 1, T$
    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 D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set
$$y_j = \begin{cases} r_j & \text{for terminal} \quad \phi_{t+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a^{'}; \theta) & \text{for non-terminal} \quad \phi_{j+1} \end{cases}$$
    Perform a gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$ according to equation (3)
  end for
end for

Algorithm 2 deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value Q with random weights θ
Initialize target action-value function Q with weights $\theta^- = \theta$
for $t = 1, T$ **do**
  With probability ϵ select a random action a_t
  otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
  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 D
  Sample random minibatch of transitons $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
  set
$$y_j = \begin{cases} r_j & \text{if episode terminates at step} j+1 \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a^{'}; \theta^-) & \text{otherwise} \end{cases}$$
  Perform a gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
  Every C step reset $\hat{Q} = Q$
end for

Algorithm 3 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.

Initialize replay memory $\mathcal{H} = \Phi, \Delta = 0, p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi(S_0)$
for $t = 1 to T$ **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = max_{i<t} p_i$
  if t $\equiv$ 0 mod K **then**
    for $j = 1 to k$ **do**
      Sample transition $j \sim P(j) = {p_j}^\alpha / \sum_i {p_i}^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{target}(S_j, argmax_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{target} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. Computer Science, 2013.

[2] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. IEEE Transactions on Automatic Control, 42(5):674–690, 2002.