Reinforcement Learning

Alvin Li

December 2020

1 SARSA

- On-Policy
- Diff between Q-Learning: uses weighted average instead of max in Q-Learning
- $R + \gamma Q(S', A') - Q(S, A)$

2 Q-Learning

- Off-policy
- Learn policy to maximize total reward
- Q-table to store value at [state,action], updated using epsilon
- Algorithm:
  (1) Agent at s_n, uses a_n, gets r_1
  (2) Pick action using either max value or random based on ϵ
  (3) Update q

  ```
  Q[state, action] = Q[state, action] + lr * (reward + gamma *
      np.max(Q[new_state, :])  Q[state, action])
  ```

- Markov Decision Process - outputs (rewards) are partly random (maybe due to malfunctioning) and partly under control, stochasticity. Also each state depends only on previous state.
- Bellman's Eqn
  $$V(s) = max_a(R(s, a) + \gamma V(s'))$$
- Bellman's Eqn with stochasticity
  $$V(s) = max_a(R(s, a) + \gamma \Sigma s' P(s, a, s')V(s'))$$
  P(s,a,s') is the probability of going from state s to s' given a

- Q Eqn (quality of action)

$$Q(s,a) = R(s,a) + \gamma \Sigma_{s'} P(s,a,s') V(s')$$

$$Q(s,a) = R(s,a) + \gamma \Sigma_{s'} P(s,a,s') max_{a'}(s',a')$$

- Temporal Difference

$$TD = R(s,a) + \gamma \Sigma_{s'} (P(s,a,s') max_{a'}(s',a')) - Q(s,a)$$

- Update

$$Q(s,a) = Q_{t-1}(s,a) + \alpha TD$$

3 Double Deep Q-Learning

- Algorithm:
  (1) Complete S, A, R, S_{t+1}, a is chosen by argmax(Q) or epsilon greedy, store SARS in memory
  (2) Sample a number of previous SARS
  (3) Loss is MSE of pred Q vs target Q, using target Q (Q') for action selection, primary Q for evaluation
  ie. $(r + \gamma Q(s_{t+1}, argmax_a Q'(s_{t+1}, a; \theta_t)) - Q(s, a, \theta))^2$
  (4) Gradient Descent
  (5) After N its, copy pred (or primary) network weights to target network weights

- Two NNs are used. One to estimate the target, one to estimate the pred (or primary). Target network is only updated every N its, using the prediction network.

- Experience Replay: sample a few [S,A,R,S] from previous X iterations to train

- Alternative: Update target network using Polyak Averaging, τ is usually very small. θ' are the weights of primary network, θ is for the target network

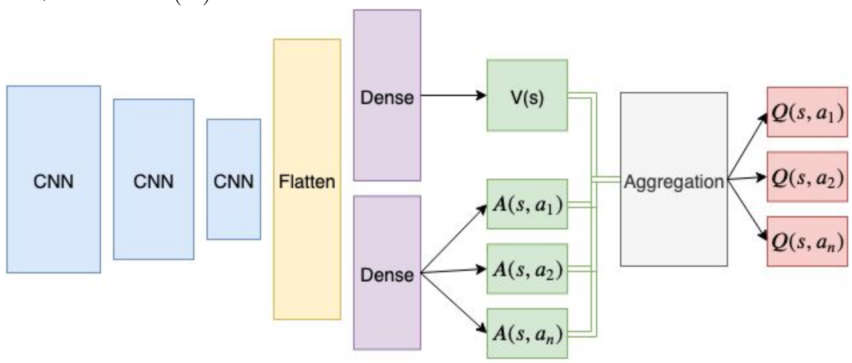
$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

- Double: To solve moving target problem, unstable nets

- Clipped: Use the minimum of Q,Q' for value evaluation, both using their own Q for action selection. To solve overestimation problem

$$r + \gamma min(Q, Q')$$

- Dueling: Using Value and Advantage functions for Q. Aggregation is Q = V + A - mean(A)



4 Deep Deterministic Policy Gradients

- $\theta^Q, \theta^\mu, \theta^{Q'}$ and $\theta^{\mu'}$ represent Q network, deterministic policy func, target Q and target policy network weights respectively. Target nets, as before, are time-delayed versions of primary nets.

- Algorithm:
  Q(s,a) is critic, $\mu(s|\theta^\mu)$ is actor, N is noise

  ```
  for m episodes:
    for T timesteps:
      Select action based on μ(s|θ^μ) + N
      Store SARS
      Sample SARS batch for experience replay
      Update Critic using loss of:
        L = MSE(r + γQ'(μ'|θ^Q') − Q(θ^Q))
      Update Actor using policy gradient:
        Δ_θ^μ J = mean(ΔQ(θ^Q)Δ_θ^μ μ(θ^μ))
      Update target nets using soft updates (τ)
  ```

- OU Process: Generate noise

5 Important Notes

Symbol or Vocab	Meaning
A or a	Action
R	Reward
α	Step Size
$\pi_t(a)$	Policy (the probability of selecting a at t)
Q_π	State Action Value Function (expected return given state and action)

6 Sources Consulted

[1] https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56
[2] https://arxiv.org/pdf/1509.06461.pdf
[3] https://arxiv.org/abs/1509.02971