Deep Reinforcement Learning

Srivatsan Srinivasan

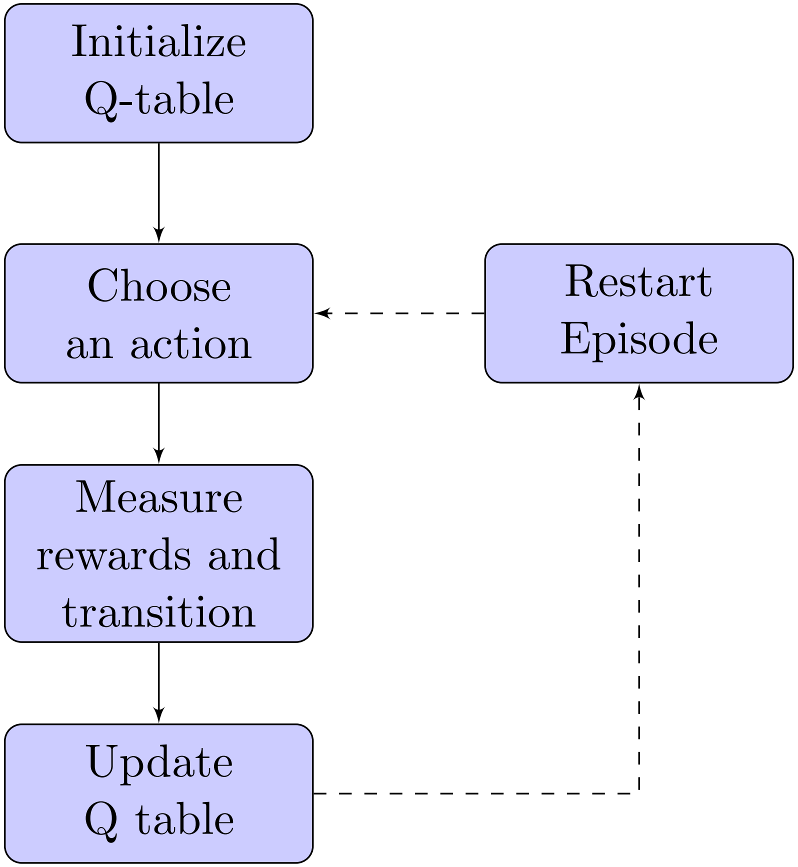
IACS

March 27, 2019

Outline

Q-Learning

Q-Learning in a nutshell



Q-Learning

$$Q(s,a) = Q(s,a) + \alpha\,\overbrace{(\underbrace{R(s,a) + \gamma \max_{a'} Q(s',a')}_{\text{approximates opt. policy directly}} - Q(s,a))}^{\text{TD Update}}$$

- Off-policy, Model-Free
- Tabular Q-Learning not scalable. EXAMPLES ?
- Approximate Q-function

Parametric Q-Learning

Use a function approximator to estimate the action-value function

$$Q^*(s,a) \approx Q(s,a;\theta)$$

θ - parameters of any function approximator

GOAL : Want to find a function approximation for Q satisfying Bellman Equation

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

Parametric Q Learning with Gradient Descent

Training continues as we keep acquiring data.

Define a target $Q^{+}(s,a;\theta|(s,a,s')) \approx R(s,a) + \gamma \max_{a'} Q(s',a';\theta)$

FORWARD PASS

$$\mathcal{L}(\theta^{(i)}) = \frac{1}{2}\sum_j \left[Q^{+}_{\theta}(s_j,a_j,s'_j) - Q_{\theta}(s_j,a_j)\right]^2$$

BACKPROPAGATION

$$\nabla_\theta \mathcal{L}(\theta^{(i)}) = \sum_j \left[Q^{+}_{\theta^{(i)}}(s_j,a_j,s'_j) - Q_{\theta^{(i)}}(s_j,a_j)\right] \cdot \nabla_\theta Q_{\theta^{(i)}}(s_j,a_j)$$

Value based Deep RL

Deep Q Networks

θ parameters of a deep network.

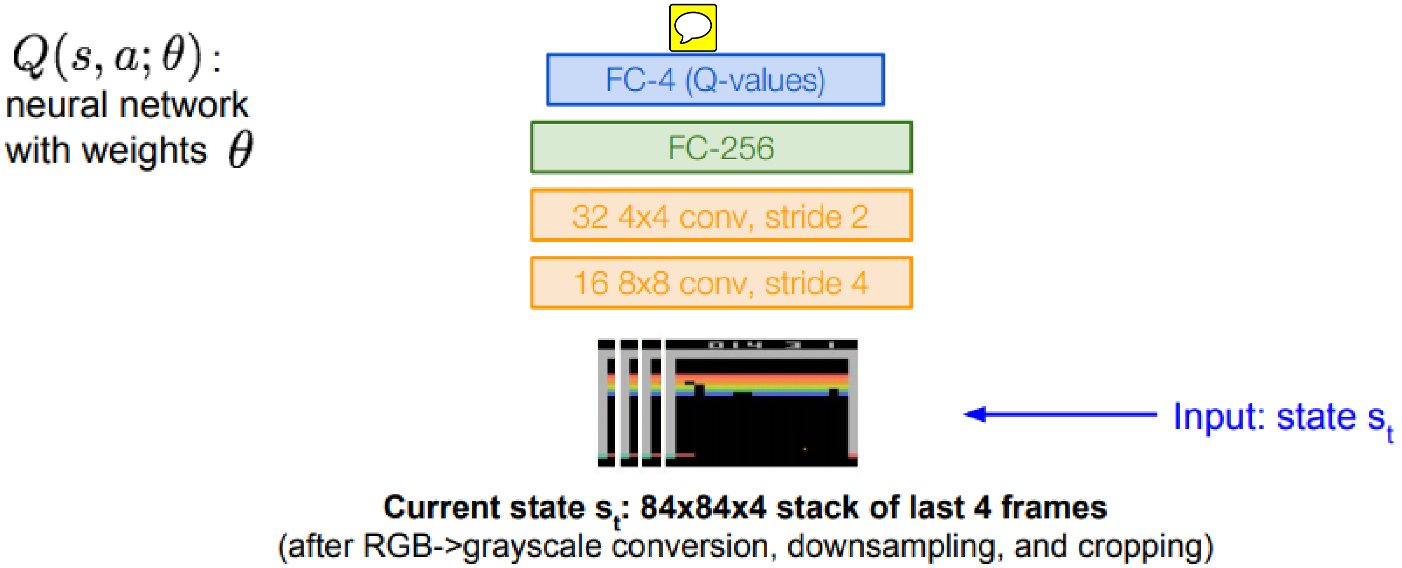


Figure 1: An Example DQN for Atari Games

Potential Issues with simple DQN

Learning from batches of consecutive samples is problematic:

1. Samples are correlated $\implies$ inefficient learning
2. Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) $\implies$ can lead to bad feedback loops.
3. Non-stationarity of targets. Remember targets are treated as pseudo "ground truths".

HACK 1 - Experience Replay Buffer

- Continually update a replay memory table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples.
- Reduces correlations between samples as we don't sample just from a recent current episode batch of transitions alone.
- Reuse allows more efficient use of previous experience as Q-Learning is incremental.
- Solves problems 1 and 2.

HACK 2 - Target Network

- Target network - Another neural network (θ^-) that is more stationary than the current Q-Network(θ) .
- Use θ^- as a lagged version of θ which is updated periodically after specified number of iterations τ (remains constant meanwhile)

$$Q^+(s_i, a_i, s_i') = R(s_i, a_i) + \gamma * \max_{a'} Q(s_i', a'; \theta^-)$$

- Solves problem 3

VISIT PSEUDO-CODE FROM NOTES

Over-Optimism in DQN

Remember TD-Update for Q-Learning Remember how we calculate the TD-target

$$\underbrace{Q^{+}(s,a;\theta)}_{\text{target}} = \underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma \max_{a'} Q(s',a';\theta \text{ or } \theta^{-})}_{\text{Discounted max possible Q-value from s'}}$$

ISSUE : Over-estimation of Q-values because of combined action selection and evaluation by the same network and propagates to other states too.

$\rightarrow$ This over-estimation/self-serving bias slows down network training.

Double DQN (DDQN-1)

Decompose the max operation in TD update into action selection and action evaluation.

- Q-Network selects action
- Target Network evaluates the value.

$$Q^+(s,a,s') = R(s,a,s') + \gamma\, Q_{\theta^-}\underbrace{(s', \underbrace{\arg\max_{a'}(Q_\theta(s',a'))}_{\text{selection with Q net}}}_{\text{evaluation with target net}}$$

Might not always improve performance but definitely helps with stability.

Dueling DQN

- Advantage Function $A(s,a) = Q(s,a) - V(s)$ gives a relative measure of utility of an action a from a state s.
- Models state value V(s) better, independent of actions $\rightarrow$ Learn which states are valuable and learn well the effect of each action (via advantage) only in those states.
- Training procedure similar to DDQN.

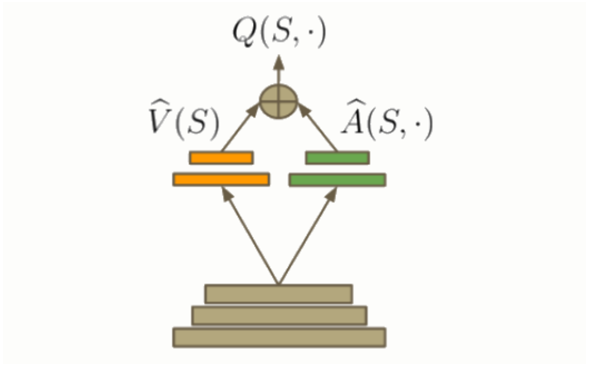


Figure 2: Dueling splits the Q into V and A and then aggregates.

End of Deep Value Models

VIDEO OF DEEPMIND ATARI

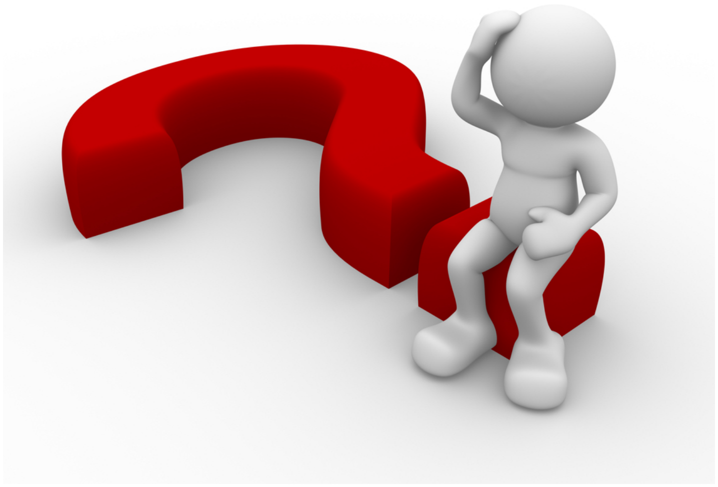


Figure 3: Any Questions on value deep RL (essential for HW) ?

Policy based Deep RL

Policy Networks

- Ultimate Goal : Find best policy - $\pi^*(a|s)$
- Directly parametrize policy by a neural network $\pi(a|s) \approx \pi(a|s;\theta)$
- Take gradients on some loss function on the policy to learn the best policy.
- WHY ? Specifying values might be more complex, policy might be easier.

  Example : a robot grasping an object has a very high-dimensional state $\implies$ hard to learn exact value of every (state, action) pair. But the policy can be much simpler: just close your hand.

Policy Gradients

Define returns of a policy.

$$J(\theta) = \sum_{s \in \mathcal{S}} d^{\pi}(s) V^{\pi}(s) = \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) Q(s, a)$$

where $d^{\pi}(s) = \lim_{t \to \infty} P(s_t = s | s_0, \pi_{\theta})$ is the stationary distribution of states under our policy $\pi(\theta)$.

Use gradient ascent to maximize $J(\theta)$

Policy Gradients, REINFORCE

Theorem (Policy Gradient Theorem)

If we define $\mathbb{E}_\pi = \mathbb{E}_{s \sim d^\pi, a \sim \pi_\theta}$, then we can rewrite the gradient of the returns under a policy π_θ as $\nabla_\theta J(\theta) = \mathbb{E}_\pi[Q_\pi(s,a)\nabla_\theta \log \pi_\theta(a|s)]$

PROOF - Switch to Lecture Notes.

- Key to understand expectations under a policy : $\mathbb{E}_\pi$ means the expectation over joint state action pairs induced by a given policy - the joint distribution of the stationary state distribution and the actions under π from those states.
- REINFORCE uses Monte-Carlo estimates using episode samples to calculate the expectation $\mathbb{E}_\pi$

Actor Critic Methods

- Two key components in policy gradients : Q, π
- Learn both of them together - leads to reduced variance in PG.
- **Critic** updates value function parameters V(s;w) or Q(s,a;w)
- **Actor** updates the policy parameters θ for $\pi_\theta(a|s)$ in the direction suggested by the critic. PSEUDO-CODE in Notes.
- A3C/A2C (distributed) - critics learn value, multiple actors trained in parallel asynchronously/synchronously.

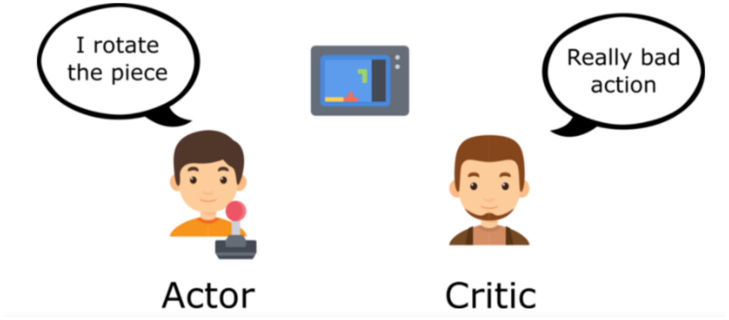


Figure 4: Actor-Critic Setup

Off-Policy Policy Gradients

- REINFORCE, A3C - On policy learning
- Off policy - better/safer exploration, does not require sequential trajectories and can use samples from a buffer.

$$\begin{aligned}
J(\theta) &= \sum_{s\in\mathcal{S}} d^{\beta}(s) \sum_{a\in\mathcal{A}} Q^{\pi}(s,a)\pi_\theta(a|s) \\
&= \mathbb{E}_{s\sim d^{\beta}}\left[\sum_{a\in\mathcal{A}} Q^{\pi}(s,a)\pi_\theta(a|s)\right] \\
\nabla_\theta J(\theta) &\approx \mathbb{E}_{s\sim d^{\beta}}\left[\sum_{a\in A} Q^{\pi}(s,a)\nabla_\theta\pi_\theta(a|s)\right] \\
&= \mathbb{E}_{\beta}\left[\underbrace{\frac{\pi_\theta(a|s)}{\beta(a|s)}}_{\text{Importance weight}} \cdot Q^{\pi}(s,a)\nabla_\theta \log\pi_\theta(a|s)\right]
\end{aligned} \tag{1}$$

- Simply adjust usual PG training with a weighted sum.

Deterministic Policy Gradients - DPG

- In all previous algorithms, policy was stochastic $\pi(s) = P(a|s)$.
- Policy as a deterministic decision $a = \mu(s)$

$$
\begin{aligned}
J(\theta) &= \int_{\mathcal{S}} \rho^{\mu}(s) Q(s, \mu_\theta(s)) ds \\
\nabla_\theta J(\theta) &= \mathbb{E}_{s \sim \rho^{\mu}} \left[\nabla_a Q^{\mu}(s, a) \nabla_\theta \mu_\theta(s) \mid_{a=\mu_\theta(s)} \right] \\
&\approx \frac{1}{N} \sum_i \nabla_a Q(s_i, a; \theta) \mid_{a=\mu_\theta(s_i)} \nabla_\theta \mu_\theta(s)
\end{aligned}
$$

- This version of deterministic policy gradient can be plugged into any algorithm.

Deep Deterministic Policy Gradients - DDPG

- Extends DQN and its ideas to continuous action space using policy gradients in an actor-critic framework.
- Can work in off-policy settings too like DQN since it uses Q-Learning TD Update and Importance weights to update the networks.
- Promotes exploration with a noisy deterministic policy $\mu'(s) = \mu(S) + \mathcal{N}$.
- Actor-Critic Training Framework
  - Select action from μ', execute and get transitions into a buffer.
  - Train the Q-Network with DQN TD Error
  - Update the policy networks using DPG
  - Update both the target networks whenever necessary.

Trust Region Optimization

Decide step size before deciding direction. Use an easier-to-optimize approximation to decide the direction to move on.

$$x^{new} = \arg\max_x m(x) \quad \text{such that} \quad ||x|| < \delta$$

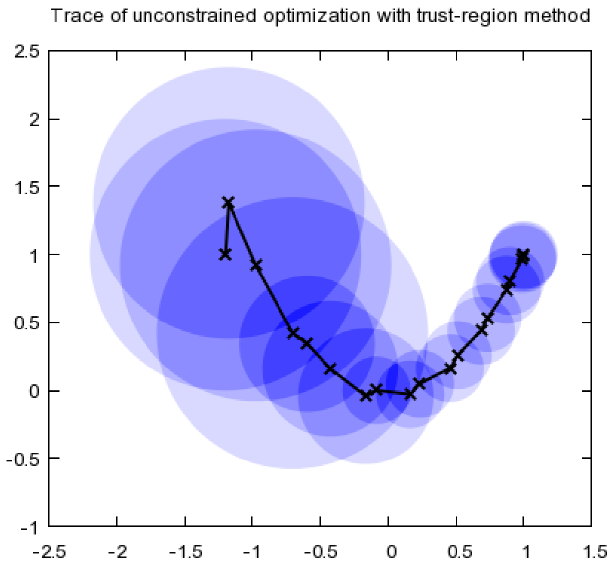


Figure 5: Trust Region Optimization

Minorize-Maximization Algorithm

- Find an approximated lower bound of the original objective as the surrogate objective
- Maximize the surrogate function (easy to optimize) continuously in each iteration.
- Guaranteed to converge to optima for convex functions.

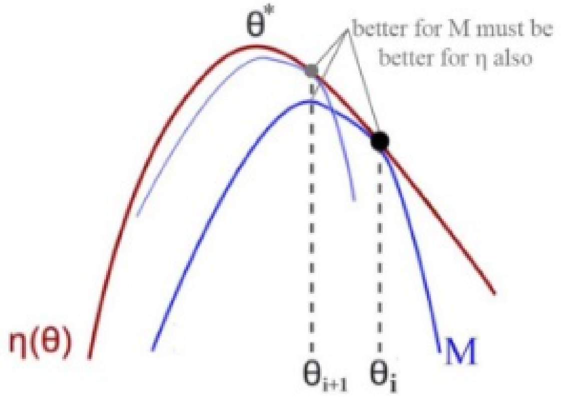


Figure 6: Minorize Maximization Example

Trust Region Policy Optimization - TRPO

- Safe-RL : Drastic policy updates undesirable.
- Make policy updates in the trust-region only.

$$J(\hat{\pi}) = J(\pi) + \sum_s \rho_{\hat{\pi}}(s) \sum_a \hat{\pi}(a|s) A_\pi(s,a)$$

$$L(\hat{\pi}) = J(\pi) + \sum_s \rho_{\pi}(s) \sum_a \hat{\pi}(a|s) A_\pi(s,a)$$

$$L(\pi_\theta) = J(\pi_{\theta_{old}}) + \sum_{s \in \mathcal{S}} \rho^{\pi_{old}} \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \hat{A}_{\theta_{old}}(s,a)$$

$$= \mathbb{E}_\pi \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \hat{A}_{\theta_{old}}(s,a)$$

TRPO objective : $\max L(\pi_\theta)$

such that $\quad D_{KL}(\pi_{\theta_{old}}(.|s) || \pi_\theta(.|s)) < \delta$

Proximal Policy Optimization - PPO

- KL divergence is sometimes hard to compute
- Proposes a simpler update scheme for the policy.
- Usual TRPO loss

$$J^{TRPO}(\theta) = \mathbb{E}\left[\underbrace{\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}}_{r(\theta)} \hat{A}_{\theta_{old}}(s,a)\right]$$

- Proposal to clip the ratio of policies.

$$J^{CLIP}(\theta) = \mathbb{E}\Bigg[\min\Bigg(r(\theta)\hat{A}_{\theta_{old}}(s,a), \quad \text{clip}(1-\epsilon, r(\theta), 1+\epsilon)\hat{A}_{\theta_{old}}(s,a)\Bigg)\Bigg]$$

- Overall loss

$$J^{PPO}(\theta) = \mathbb{E}\Bigg[J^{CLIP}(\theta^\pi) - \underbrace{c_1 \sum_k (V^+(s) - V_{\theta^V}(s))^2}_{\text{usual TD loss}} + \underbrace{c_2 H(s, \pi_\theta(}_{\text{Promotes exploration}}$$

REST....

Due to paucity of time, you can find additional and deeper material on the ones we talked about and other policy gradient variants in the references provided in the end of the notes.

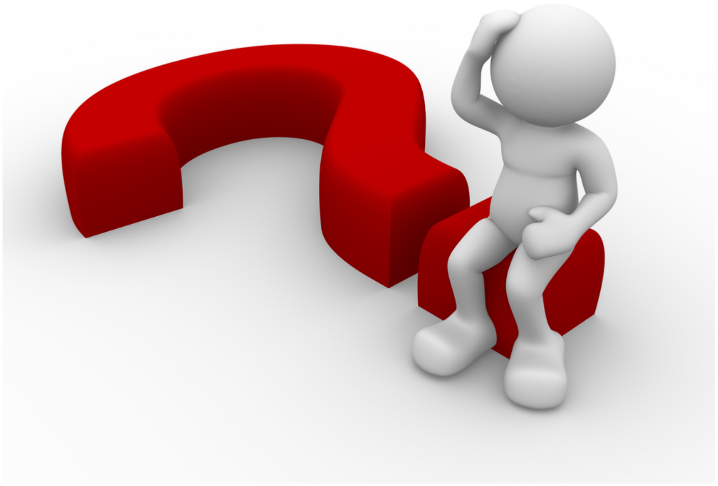


Figure 7: Any Questions on policy deep RL

Lessons

General Lessons

- Variance reduction in gradient estimates is important.
- Off-policy - exploratory and data efficient.
- Target Networks to stabilize decisions of online networks.
- Entropy regularization to promote exploration.
- Shared parameter networks are useful.
- Deterministic policy as a single point approximation of stochastic policy.
- Ideas from convex and approximate convex optimization.