facebook
Artificial Intelligence Research

How to solve an MDP incrementally: Approximate algorithms - Value Based

Pirotta Matteo
Facebook AI Research

Acknowledgments

Special thanks to Alessandro Lazaric for providing these slides from the RL class we teach in Paris.

Approximate Value-based Algorithms

Policy Evaluation

Approximate Monte-Carlo As *Supervised Learning*

- Distribution over the state space $\mathcal{D}$
- Function approximation $V_\theta : S \to \mathbb{R}$, $\theta \in \mathbb{R}^d$ *[e.g., linear, deepNet]*
- Build training set of n *samples*

$$s_i \sim \mathcal{D} \qquad R_i = \sum_{t=0}^{H} \gamma^t r_{t,i} = V^\pi(s_i) + \epsilon_i \quad (\mathbb{E}[\epsilon_i] = 0)$$

- Training (batch)

$$\widehat{\theta}_n = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} L(s_i, R_i; \theta) = \frac{1}{n} \sum_{i=1}^{n} \big(V_\theta(s_i) - R_i\big)^2$$

- Testing (aka *generalization* error)

$$L(\widehat{\theta}_n) = \mathbb{E}_{\mathcal{D}}\Big[\big(V^\pi(s) - V_{\widehat{\theta}_n}(s)\big)^2\Big]$$

Approximate Monte-Carlo As *Supervised Learning*

Proposition (qualitative)

Let n be the number of samples used to build the Monte-Carlo training set. Let also $r(s,a) \in [0, r_{\max}]$ and trajectories to be as long as $H = 1/(1-\gamma)$, then approximate Monte-Carlo has a generalization error

$$L(\widehat{\theta}_n) \leq \min_{\theta} L(\theta) + O\left(\frac{1}{1-\gamma}\sqrt{\frac{d}{n}}\right)$$

- Tends to the best possible approximation as n tends to infinity
- Variance may be big

Approximate Monte-Carlo As *Supervised Learning*

- Distribution over the state space $\mathcal{D}$
- Function approximation $V_\theta : S \to \mathbb{R}$, $\theta \in \mathbb{R}^d$ *[e.g., linear, deepNet]*
- Build training set of n samples

$$s_i \sim \mathcal{D} \qquad R_i = \sum_{t=0}^{T_i} \gamma^t r_{t,i} = V^\pi(s_i) + \epsilon_i \quad (\mathbb{E}[\epsilon_i] = 0)$$

- Training *(batch)*

$$\widehat{\theta}_n = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} L(s_i, R_i; \theta) = \frac{1}{n} \sum_{i=1}^{n} \big(V_\theta(s_i) - R_i\big)^2$$

- Testing (aka generalization error)

$$L(\widehat{\theta}_n) = \mathbb{E}_{\mathcal{D}}\Big[\big(V^\pi(s) - V_{\widehat{\theta}_n}(s)\big)^2\Big]$$

Approximate Monte-Carlo As *Supervised Learning*

- Distribution over the state space $\mathcal{D}$
- Function approximation $V_\theta : S \to \mathbb{R}$, $\theta \in \mathbb{R}^d$ *[e.g., linear, deepNet]*
- Build training set of n samples

$$s_i \sim \mathcal{D} \qquad R_i = \sum_{t=0}^{T_i} r_{t,i} = V^\pi(s_i) + \epsilon_i \quad (\mathbb{E}[\epsilon_i] = 0)$$

- Monte-Carlo with **online** training after each sample (s_i, R_i) with learning rate α_i

$$\begin{aligned}\widehat{\theta}_{i+1} &= \widehat{\theta}_i - \alpha_i \nabla_\theta L(s_i, R_i; \theta_i)\\ &= \widehat{\theta}_i - \alpha_i \big(V_{\theta_i}(s_i) - R_i\big) \nabla_\theta V_{\theta_i}(s_i)\end{aligned}$$

- Testing (aka generalization error)

$$L(\widehat{\theta}_n) = \mathbb{E}_{\mathcal{D}}\Big[\big(V^\pi(s) - V_{\widehat{\theta}_n}(s)\big)^2\Big]$$

Policy Evaluation

Fixed policy π

For $i = 1, \ldots, n$

1. Set $t = 0$
2. Set initial state s_0
3. **While** ($s_{t,i}$ not terminal) *[execute one trajectory]*
    1. Take action $a_{t,i} = \pi(s_{t,i})$
    2. Observe **next state** $s_{t+1,i}$ and **reward** $r_{t,i} = r(s_{t,i}, a_{t,i})$
    3. Set $t = t + 1$

   EndWhile

EndFor

Return: Estimate of the value function $\widehat{V}^\pi(\cdot)$

Approximate TD As *Pseudo*-Gradient Descent

- Run π over a single trajectory $(s_0, r_0, s_1, r_1, s_2, r_2, \ldots, s_n, r_n)$
- TD loss using **bootstrapped** target

$$\widetilde{L}(s_t, \widetilde{R}_t; \theta) = (V_\theta(s_t) - \widetilde{R}_t)^2 = (V_\theta(s_t) - r_t - \gamma V_{\theta_t}(s_{t+1}))^2$$

- TD *online* update with learning rate α_t

$$\begin{aligned}\widehat{\theta}_{t+1} &= \widehat{\theta}_t - \alpha_t \nabla_\theta \widetilde{L}(s_t, \widetilde{R}_t; \theta_t) \\ &= \widehat{\theta}_t - \alpha_t \big(V_{\theta_t}(s_t) - r_t - \gamma V_{\theta_t}(s_{t+1})\big) \nabla_\theta V_{\theta_t}(s_t)\end{aligned}$$

Approximate TD As *Pseudo*-Gradient Descent

- Run π over a single trajectory $(s_0, r_0, s_1, r_1, s_2, r_2, \dots, s_n, r_n)$
- TD loss using **bootstrapped** target

$$\widetilde{L}(s_t, \widetilde{R}_t; \theta) = (V_\theta(s_t) - \widetilde{R}_t)^2 = (V_\theta(s_t) - r_t - \gamma V_{\theta_t}(s_{t+1}))^2$$

- TD *online* update with learning rate α_t

$$\begin{aligned}\widehat{\theta}_{t+1} &= \widehat{\theta}_t - \alpha_t \nabla_\theta \widetilde{L}(s_t, \widetilde{R}_t; \theta_t) \\ &= \widehat{\theta}_t - \alpha_t \big(V_{\theta_t}(s_t) - r_t - \gamma V_{\theta_t}(s_{t+1})\big) \nabla_\theta V_{\theta_t}(s_t)\end{aligned}$$

Not really a gradient method...

Linear TD

Linear space to approximate value functions

$$\mathcal{F} = \left\{ V_\theta(s) = \sum_{j=1}^{d} \theta_j \varphi_j(s), \;\; \theta \in \mathbb{R}^{d} \right\} \qquad \text{with features } \varphi_j : S \to [0, L]$$

Compact notation

$$\phi(s) = [\varphi_1(s) \dots \varphi_d(s)]^\top \in \mathbb{R}^d \Rightarrow V_\theta(s) = \phi(s)^\top \theta$$

$$\Phi = \left[\phi(s_1)^\top; \phi(s_2)^\top; \cdots \phi(s_S)^\top\right] \in \mathbb{R}^{S \times d} \Rightarrow V_\theta = \Phi\theta$$

Linear TD

Linear TD update equation

$$\begin{aligned}\widehat{\theta}_{t+1} &= \widehat{\theta}_t - \alpha_t\big(V_{\theta_t}(s_t) - r_t - \gamma V_{\theta_t}(s_{t+1})\big)\nabla_\theta V_{\theta_t}(s_t)\\ &= \widehat{\theta}_t - \alpha_t\big(\phi(s_t)^\top\theta_t - r_t - \gamma\phi(s_{t+1})^\top\theta\big)\phi(s_t)\\ &= \widehat{\theta}_t - \alpha_t\big(\phi_t^\top\theta_t - r_t - \gamma\phi_{t+1}^\top\theta\big)\phi(s_t)\end{aligned}$$

Linear TD

Theorem

Let $D \in \mathbb{R}^{S\times S}$ be the matrix of the stationary distribution of π, i.e., $D = \text{diag}(\rho^\pi(s_1), \rho^\pi(s_2), \dots, \rho^\pi(s_S))$. Then the linear TD estimate converges to θ^*, which is the fixed point of the projected Bellman operator

$$\Phi\theta^* = \Pi_D T^\pi \Phi\theta^*$$

and it has error

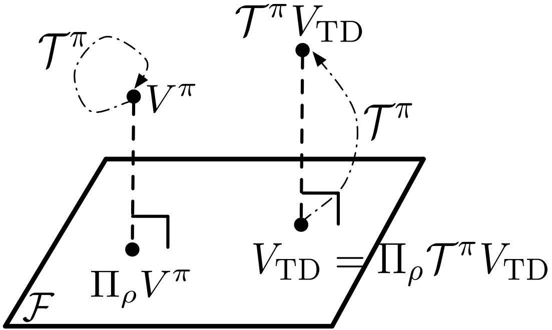
$$L_D(\theta^*) \leq \frac{1}{\sqrt{1-\gamma^2}} \min_\theta L_D(\theta)$$

where Π_D is the orthogonal projection in D norm and L_D is the expected loss w.r.t. the stationary distribution D.

- Linear TD converges
- The error is related to the best possible error

Linear TD

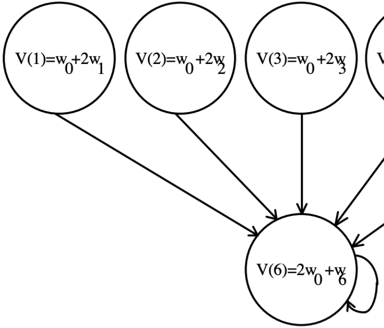
$$V_{TD} = \Phi\theta^* = \Pi_D T^\pi \Phi\theta^* = \Pi_D T^\pi V_{TD}$$



Approximate TD

Approximate TD **may not converge** (i.e., it might diverge) if

- Linear approximation but states s_i are obtained by following a different policy *(off-policy learning)*
- *Non-linear approximation* and states s_i are obtained by following π



Approximate TD – Extensions

- Approximate TD(λ)
- GTD, GTD2, TDC (and others): *convergence* guarantees for off-policy and "mildly" non-linear approximators
- *Averagers* are specific stable approximators (mostly interpolators)
- Approximate TD is a *true* gradient method in reversible Markov chains
- Many *variance reduction* techniques can be applied

Reinforcement Learning Algorithms

Policy Learning

Approximate QL As *Pseudo*-Gradient Descent

- Run π over a single trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_n, a_n, r_n)$
- QL loss using **bootstrapped** target

$$\widetilde{L}(s_t, a_t, \widetilde{R}_t; \theta) = (Q_\theta(s_t, a_t) - \widetilde{R}_t)^2 = \Big(Q_\theta(s_t, a_t) \underbrace{- r_t - \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a')}_{\text{target}}\Big)^2$$

- QL *online* update with learning rate α_t

$$\begin{aligned}\widehat{\theta}_{t+1} &= \widehat{\theta}_t - \alpha_t \nabla_\theta \widetilde{L}(s_t, a_t, \widetilde{R}_t; \theta_t) \\ &= \widehat{\theta}_t - \alpha_t \big(Q_{\theta_t}(s_t, a_t) - r_t - \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a')\big) \nabla_\theta Q_{\theta_t}(s_t, a_t)\end{aligned}$$

Approximate QL As *Pseudo*-Gradient Descent

- Run π over a single trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_n, a_n, r_n)$
- QL loss using **bootstrapped** target

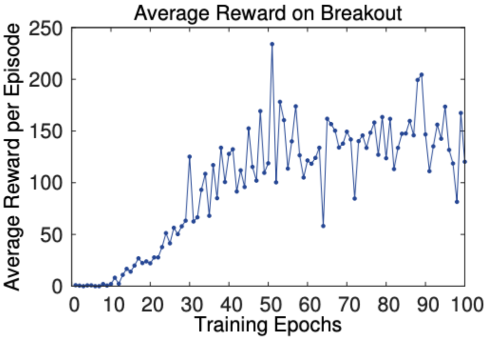
$$\widetilde{L}(s_t, a_t, \widetilde{R}_t; \theta) = (Q_\theta(s_t, a_t) - \widetilde{R}_t)^2 = \Big(Q_\theta(s_t, a_t) \underbrace{- r_t - \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a')}_{\text{target}}\Big)^2$$

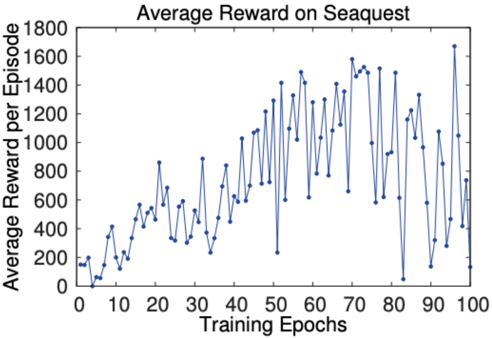
- QL *online* update with learning rate α_t

$$\begin{aligned}\widehat{\theta}_{t+1} &= \widehat{\theta}_t - \alpha_t \nabla_\theta \widetilde{L}(s_t, a_t, \widetilde{R}_t; \theta_t) \\ &= \widehat{\theta}_t - \alpha_t \big(Q_{\theta_t}(s_t, a_t) - r_t - \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a')\big) \nabla_\theta Q_{\theta_t}(s_t, a_t)\end{aligned}$$

It may diverge even with linear function approximation...

Approximate QL As *Pseudo*-Gradient Descent





Towards DQN

Practical challenges in making approximate QL "more" stable

- Sequential updates $\Rightarrow$ *correlated samples*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*
- QL update using $\max_{a'} Q(s, a')$ $\Rightarrow$ *over-estimation*

Towards DQN

Practical challenges in making approximate QL “more” stable

- Sequential updates $\Rightarrow$ *correlated samples*
  $\Rightarrow$ *experience replay*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*
- QL update using $\max_{a'} Q(s, a')$ $\Rightarrow$ *over-estimation*

Towards DQN

Practical challenges in making approximate QL "more" stable

- Sequential updates $\Rightarrow$ *correlated samples*
  $\Rightarrow$ *experience replay*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
  $\Rightarrow$ *target network*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*

- QL update using $\max_{a'} Q(s, a')$ $\Rightarrow$ *over-estimation*

Towards DQN

Practical challenges in making approximate QL "more" stable

- Sequential updates $\Rightarrow$ *correlated samples*
  $\Rightarrow$ *experience replay*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
  $\Rightarrow$ *target network*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*
  $\Rightarrow$ *reward normalization*
- QL update using $\max_{a'} Q(s, a')$ $\Rightarrow$ *over-estimation*

Towards DQN

Practical challenges in making approximate QL "more" stable

- Sequential updates $\Rightarrow$ *correlated samples*
  $\Rightarrow$ *experience replay*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
  $\Rightarrow$ *target network*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*
  $\Rightarrow$ *reward normalization*
- QL update using $\max_{a'} Q(s, a')$ $\Rightarrow$ *over-estimation*
  $\Rightarrow$ *double Q-learning*

Experience Replay

- To help remove correlations, store dataset $\mathcal{D}$ from prior experience
- QL online with *replay buffer*
  - Sample experience from the dataset

$$(s, a, r, s') \sim \mathcal{D}$$

  - Online update

$$\widehat{\theta}_{t+1} = \widehat{\theta}_t - \alpha_t \Big(Q_{\theta_t}(s,a) \underbrace{- r - \gamma \max_{a'} Q_{\theta_t}(s', a')}_{\text{target}} \Big) \nabla_\theta Q_{\theta_t}(s,a)$$

  - Execute policy (e.g., ϵ-greedy or softmax)
  - Add new sample to dataset

Target Network

Issue: weights are updated and the target changes $\Longrightarrow$ non-stationarity

- To help improve stability, fix the target weights used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights being updated
- Let $\overline{\theta}$ be the parameters of the target network

Target Network

- QL online with *replay buffer* and *target network*
  - Sample experience from the dataset

$$(s, a, r, s') \sim \mathcal{D}$$

  - Compute target

$$y_t = r + \gamma \max_{a'} Q_{\overline{\theta}}(s', a')$$

  - Online update

$$\widehat{\theta}_{t+1} = \widehat{\theta}_t - \alpha_t \big(Q_{\theta_t}(s, a) - y_t\big) \nabla_\theta Q_{\theta_t}(s, a)$$

  - Execute policy (e.g., ϵ-greedy or softmax)
  - Add new sample to dataset
- *Update target network* $\overline{\theta}$ every C steps

* it is possible to do also a smooth update of the target network $\overline{\theta} = \tau\overline{\theta} + (1-\tau)\theta_t$ with $\tau \approx 1$. Less used than full updates.

Mini-batch Update

Issue: online update is inefficient with modern tools (e.g., NN)

Perform update on a *mini-batch* $\mathcal{D}_{\text{mini}}$ sampled from $\mathcal{D}$

- Let $\overline{\theta}$ the *target function*
- Mini-batch loss

$$\widetilde{L}_{\mathcal{D}_{\text{mini}}}(\theta) = \mathbb{E}_{(s_i,a_i,s_{i+1},r_i)\sim\mathcal{D}}\left[\left(Q_\theta(s_i,a_i) - r_i - \gamma \max_{a'} Q_{\overline{\theta}}(s_{i+1},a')\right)^2\right]$$

- Update θ using SGD on $\widetilde{L}_{\mathcal{D}_{\text{mini}}}(\theta)$

Mini-Batch Update

- Sample m transitions from replay buffer $\mathcal{D}$

$$\Lambda_t = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^m$$

- Compute *loss*

$$L(\theta|\Lambda_t, \overline{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(Q_\theta(s,a) - r_i - \gamma \max_{a'} Q_{\overline{\theta}}(s'_i, a')\right)^2$$

- Update by SGD

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta|\Lambda_t, \overline{\theta})$$

Target Network

Learn optimal policy π^*

Initialize θ **and** $\overline{\theta}$

For $i = 1, \ldots, n$

1. Set $t = 0$
2. Set initial state s_0
3. **While** ($s_{t,i}$ not terminal) *[execute one trajectory]*
   1. Take action $a_{t,i}$ [using ϵ-greedy or softmax]
   2. Observe next state $s_{t+1,i}$ and reward $r_{t,i} = r(s_{t,i}, a_{t,i})$
   3. Set $t = t + 1$
   4. Store transition $(s_{t,i}, a_{t,i}, s_{t+1,i}, r_{t,i})$ into an experience replay buffer $\mathcal{D}$
   5. Perform update *of* θ on a mini-batch $\mathcal{D}_{\text{mini}}$ sampled from $\mathcal{D}$ *using target* $\overline{\theta}$
   6. **Every** C **steps** $\overline{\theta} \leftarrow \theta$

   EndWhile

EndFor

Return: Estimate of the optimal policy $\widehat{\pi}^*$

DQN – Atari

Image preprocessing: grey-scale, crop to 84x84

DQN – Atari

State definition

$s_t = \{$

$a_{t-1}, r_{t-1},$

$\cdots$

$a_{t-4}, r_{t-4} \}$

DQN – Atari

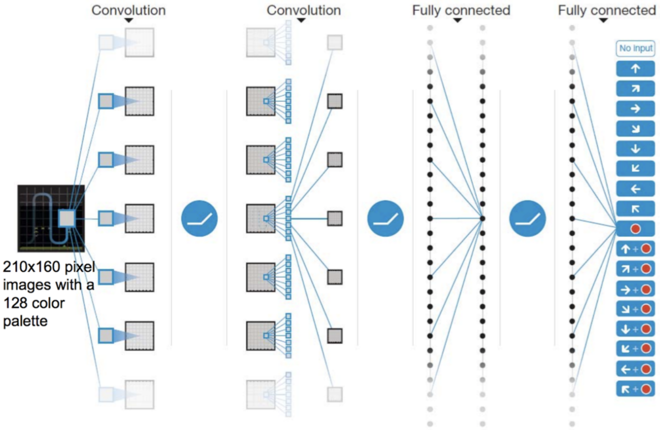
Time definition: 4 last frames

$s_t = \{$

$a_{t-1}, r_{t-1}, \dots, a_{t-4}, r_{t-4} \}$

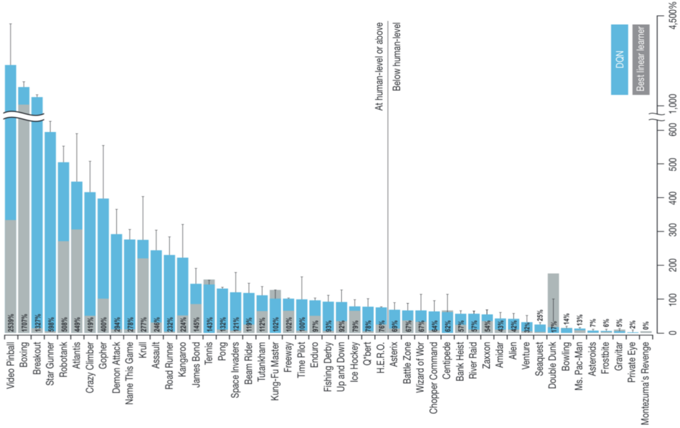
DQN – Atari

Action-value function: deepNet with as many heads as actions



DQN – Atari

Performance



DQN – Atari

Ablation

DQN

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Summary

- Update rule of approximate TD and the properties of its linear version
- Update rule of approximate QL and the DQN algorithm

Bibliography

Thank you!

facebook
Artificial Intelligence Research