Advanced Q-Function Learning Methods

February 22, 2017

Review: Q-Value iteration

Algorithm 1 Q-Value Iteration

Initialize $Q^{(0)}$
for $n = 0, 1, 2, \ldots$ until termination condition **do**
  $Q^{(n+1)} = \mathcal{T} Q^{(n)}$
end for

$$[\mathcal{T} Q](s, a) = \mathbb{E}_{s_1}\left[r_0 + \gamma \max_{a_1} Q(s_1, a_1) \,\middle|\, s_0 = s, a_0 = a \right]$$

Q-Value Iteration with Function Approximation: Batch Method

- Parameterize Q-function with a neural network Q_θ
- Backup estimate $\widehat{\mathcal{T}Q}_t = r_t + \max_{a_{t+1}} \gamma Q(s_{t+1}, a_{t+1})$
- To approximate $Q \leftarrow \widehat{\mathcal{T}Q}$, solve minimize$_\theta \sum_t \left\| Q_\theta(s_t, a_t) - \widehat{\mathcal{T}Q}_t \right\|^2$

Algorithm 2 Neural-Fitted Q-Iteration (NFQ)[1]

- Initialize $\theta^{(0)}$.

  for $n = 0, 1, 2, dots$ **do**

  Run policy for K timesteps using some policy $\pi^{(n)}$.

  $\theta^{(n+1)} = \text{minimize}_\theta \sum_t \left(\widehat{\mathcal{T}Q_{\theta^{(n)}}}_t - Q_\theta(s_t, a_t) \right)^2$

  end for

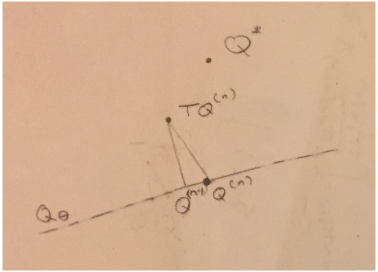
[1] M. Riedmiller. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". *Machine Learning: ECML 2005*. Springer, 2005.

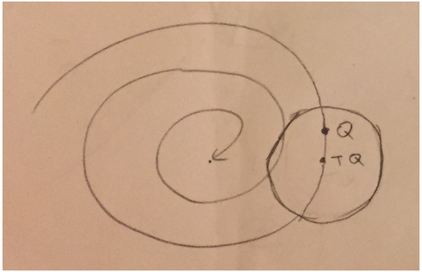
Q-Value Iteration with Function Approximation: Online/Incremental Method

Algorithm 3 Watkins' Q-learning / Incremental Q-Value Iteration

Initialize $\theta^{(0)}$.
for $n = 0, 1, 2, dots$ **do**
  Run policy for K timesteps using some policy $\pi^{(n)}$.
  $g^{(n)} = \nabla_\theta \sum_t \left(\widehat{\mathcal{T}Q}_t - Q_\theta(s_t, a_t) \right)^2$
  $\theta^{(n+1)} = \theta^{(n)} - \alpha g^{(n)}$ (SGD update)
end for

Q-Value Iteration with Function Approximation: Error Propagation

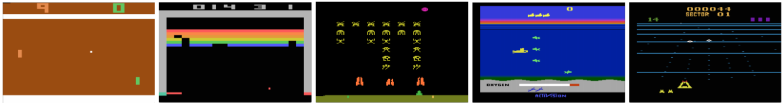




- Two sources of error: approximation (projection), and noise
- Projected Bellman update: $Q \to \Pi \mathcal{T} Q$
  - $\mathcal{T}$: backup, contraction under $\|\cdot\|_\infty$, not $\|\cdot\|_2$
  - Π: contraction under $\|\cdot\|_2$, not $\|\cdot\|_\infty$

DQN (overview)

- Mnih et al. introduced Deep Q-Network (DQN) algorithm, applied it to ATARI games
- Used deep learning / ConvNets, published in early stages of deep learning craze (one year after AlexNet)
- Popularized ATARI (Bellemare et al., 2013) as RL benchmark

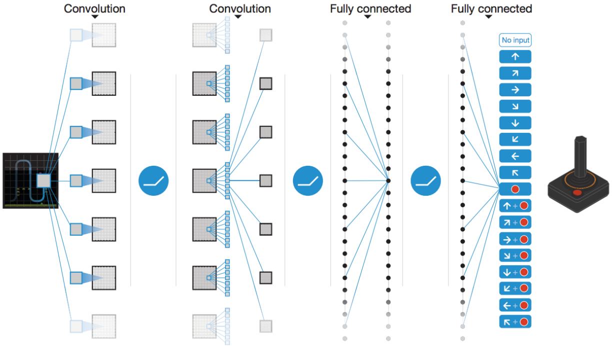


- Outperformed baseline methods, which used hand-crafted features

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, et al. "Playing Atari with Deep Reinforcement Learning". (2013)
M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. "The Arcade Learning Environment: An Evaluation Platform for General Agents". *Journal of Artificial Intelligence Research* (2013)

DQN (network)



Convolution
Convolution
Fully connected
Fully connected
No input

DQN (algorithm)

- Algorithm is hybrid of online and batch Q-value iteration, interleaves optimization with data collection
- Key terms:
  - Replay memory $\mathcal{D}$: history of last N transitions
  - Target network: old Q-function $Q^{(n)}$ that is fixed over many ($\sim 10,000$) timesteps, while $Q \Rightarrow \mathcal{T}Q^{(n)}$

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

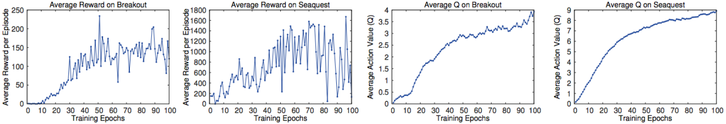
-

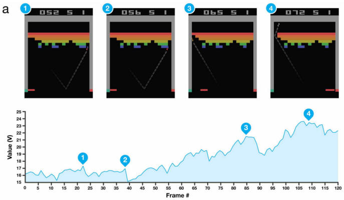
DQN Algorithm: Key Concepts

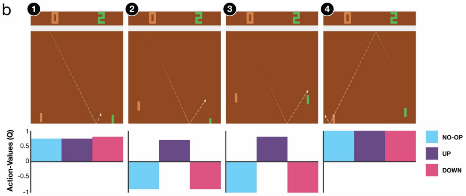
- Why replay memory?
  - Why it's valid: Q-function backup $Q \Rightarrow \mathcal{T}Q^{(n)}$ can be performed using off-policy data
  - Each transition (s, a, r, s') seen many times $\Rightarrow$ better data efficiency, reward propagation
  - History contains data from many past policies, derived from $Q^{(n)}, Q^{(n-1)}, Q^{(n-2)}, \ldots$ and changes slowly, increasing stability.
  - Feedback: $Q \Leftrightarrow \mathcal{D}$
- Why target network? Why not just use current Q as backup target?
  - Resembles batch Q-value iteration, fixed target $\mathcal{T}Q^{(n)}$ rather than moving target
  - Feedback: $Q \Leftrightarrow Q^{(\text{target})}$

Are Q-Values Meaningful

Yes:



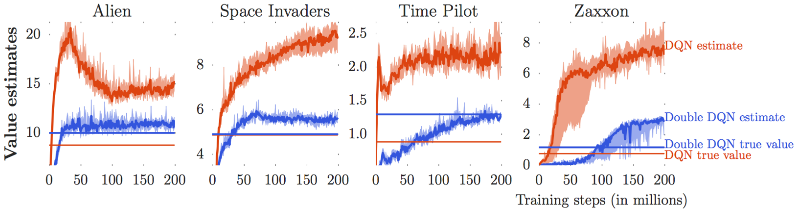




From supplementary material of V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, et al. "Human-level control through deep reinforcement learning". *Nature* (2015)

Are Q-Values Meaningful

But:



From H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double Q-learning". *CoRR abs/1509.06461* (2015)

Double Q-learning

- $\mathbb{E}_{X_1,X_2}[\max(X_1, X_2)] \geq \max(\mathbb{E}_{X_1,X_2}[X_1], \mathbb{E}[X_2])$
- Q-values are noisy, thus $r + \gamma \max_{a'} Q(s', a')$ is an overestimate
- Solution: use two networks Q_A, Q_B, and compute argmax with the *other* network

$$Q_A(s, a) \leftarrow r + \gamma Q(s', \arg\max_{a'} Q_B(s', a'))$$
$$Q_B(s, a) \leftarrow r + \gamma Q(s', \arg\max_{a'} Q_A(s', a'))$$

"$\leftarrow$" means "updates towards"

H. V. Hasselt. "Double Q-learning". *NIPS*. 2010

Double DQN

- Standard DQN:

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q^{(\text{target})}(s',a')$$

$$Q(s,a) \leftarrow r + \gamma Q^{(\text{target})}(s', \arg\max_{a'} Q^{(\text{target})}(s',a'))$$

- Double DQN:

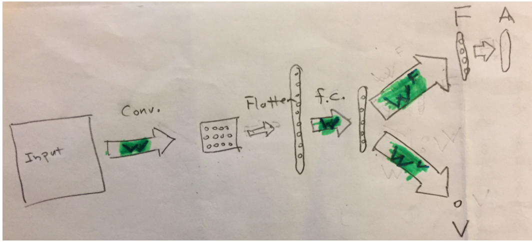
$$Q(s,a) \leftarrow r + \gamma Q^{(\text{target})}(s', \arg\max_{a'} Q(s',a'))$$

- Might be more accurately called "Half DQN"

H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double Q-learning". *CoRR, abs/1509.06461* (2015)

Dueling net

- Want to separately estimate value function and advantage function $Q(s,a) = V(s) + A(s,a)$
  - $|V|$ has larger scale than $|A|$ by $\approx 1/(1-\gamma)$
  - But small differences $A(s,a) - A(s,a')$ determine policy
- Parameterize Q function as follows: $Q_\theta(s,a) = V_\theta(s) + \underbrace{F_\theta(s,a) - \operatorname{mean}_{a'} F_\theta(s,a')}_{\text{“}Advantage\text{” part}}$

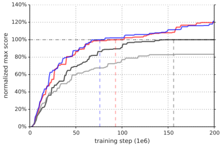


- Separates value and advantage parameters, whose gradients have different scale. Poor scaling can be fixed by RMSProp / ADAM

Z. Wang, N. de Freitas, and M. Lanctot. “Dueling network architectures for deep reinforcement learning”. arXiv preprint arXiv:1511.06581

Prioritized Replay

- Bellman error loss: $\sum_{i \in \mathcal{D}} \left\| Q_\theta(s_i, a_i) - \hat{Q}_t \right\|^2 / 2$
- Can use importance sampling to favor timesteps i with large gradient. Allows for faster backwards propagation of reward information
- Use last Bellman error $|\delta_i|$, where $\delta_i = Q_\theta(s_i, a_i) - \hat{Q}_t$ as proxy for size of gradient
  - Proportional: $p_i = |\delta_i| + \epsilon$
  - Rank: $p_i = 1/\operatorname{rank}_i$
- Yields substantial speedup across ATARI benchmark



T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". *arXiv preprint arXiv:1511.05952* (2015)

Practical Tips (I)

- DQN is more reliable on some tasks than others. Test your impementation on reliable tasks like Pong and Breakout: if it doesn't achieve good scores, something is wrong.

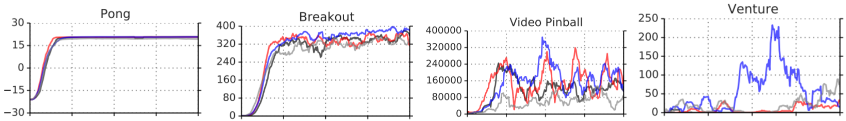


Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". *arXiv preprint arXiv:1511.05952* (2015), Figure 7

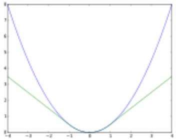
- Large replay buffers improve robustness of DQN, and memory efficiency is key.
  - Use uint8 images, don't duplicate data
- Be patient. DQN converges slowly—for ATARI it's often necessary to wait for 10-40M frames (couple of hours to a day of training on GPU) to see results significantly better than random policy

Credit: Szymon Sidor

Practical Tips (II)

- Use Huber loss on Bellman error

$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \leq \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$



- Do use Double DQN—significant improvement from 3-line change in Tensorflow.
- To test out your data preprocessing, try your own skills at navigating the environment based on processed frames.
- Always run at least two different seeds when experimenting
- Learning rate scheduling is beneficial. Try high learning rates in initial exploration period.
- Try non-standard exploration schedules.

Credit: Szymon Sidor

That's all. Questions?