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

\[ Q^{(n+1)} = T Q^{(n)} \]

end for

\[
[TQ](s, a) = E_{s_1} \left[ r_0 + \gamma \max_{a_1} Q(s_1, a_1) \mid s_0 = s, a_0 = a \right]
\]
Q-Value Iteration with Function Approximation: Batch Method

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

Algorithm 2 Neural-Fitted Q-Iteration (NFQ)$^1$

- Initialize $\theta^{(0)}$.
  
  for $n = 0, 1, 2, \ldots$ do
  
  Run policy for $K$ timesteps using some policy $\pi^{(n)}$.
  
  $\theta^{(n+1)} = \text{minimize}_\theta \sum_t \left( \hat{T}Q_{\theta^{(n)}} - Q_\theta(s_t, a_t) \right)^2$

  end for

---

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

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

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

for $n = 0, 1, 2, \ldots$ do

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

$g^{(n)} = \nabla_{\theta} \sum_t \left( \widehat{T}Q_t - Q_{\theta}(s_t, a_t) \right)^2$

$\theta^{(n+1)} = \theta^{(n)} - \alpha g^{(n)}$ (SGD update)

end for
Q-Value Iteration with Function Approximation: Error Propagation

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

<table>
<thead>
<tr>
<th>B. Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S. Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>354</td>
<td>1.2</td>
<td>0</td>
<td>-20.4</td>
<td>157</td>
<td>110</td>
</tr>
<tr>
<td>Contingency [4]</td>
<td>1743</td>
<td>6</td>
<td>159</td>
<td>-17</td>
<td>960</td>
<td>722</td>
</tr>
<tr>
<td>DQN</td>
<td>4092</td>
<td>168</td>
<td>470</td>
<td>20</td>
<td>1952</td>
<td>1705</td>
</tr>
<tr>
<td>Human</td>
<td>7456</td>
<td>31</td>
<td>368</td>
<td>-3</td>
<td>18900</td>
<td>28010</td>
</tr>
<tr>
<td>HNeat Best [8]</td>
<td>3616</td>
<td>52</td>
<td>106</td>
<td>19</td>
<td>1800</td>
<td>920</td>
</tr>
<tr>
<td>HNeat Pixel [8]</td>
<td>1332</td>
<td>4</td>
<td>91</td>
<td>-16</td>
<td>1325</td>
<td>800</td>
</tr>
<tr>
<td>DQN Best</td>
<td>5184</td>
<td>225</td>
<td>661</td>
<td>21</td>
<td>4500</td>
<td>1740</td>
</tr>
</tbody>
</table>

DQN (network)
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 $Q \Rightarrow TQ^{(n)}$

---

**Algorithm 1** Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights

```plaintext
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 $\epsilon$ 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 $Q \Rightarrow TQ^{(n)}$ 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: $Q \Leftrightarrow D$

▶ Why target network? Why not just use current $Q$ as backup target?
  ▶ Resembles batch $Q$-value iteration, fixed target $TQ^{(n)}$ rather than moving target
  ▶ Feedback: $Q \Leftrightarrow Q^{(target)}$
Are Q-Values Meaningful

Yes:

Are Q-Values Meaningful

But:

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

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

“←” means “updates towards”
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”

---

**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:

\[ Q_\theta(s, a) = V_\theta(s) + F_\theta(s, a) - \text{mean}_{a'} F_\theta(s, a') \]

“Advantage” part

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

Prioritized Replay

- Bellman error loss: \( \sum_{i \in D} \left| Q_\theta(s_i, a_i) - \hat{Q}_t \right|^2 / 2 \)
- 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: \( p_i = 1 / \text{rank}_i \)
- Yields substantial speedup across ATARI benchmark.

---

Practical Tips (I)

▶ DQN is more reliable on some tasks than others. Test your implementation 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

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

▶ 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?