Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih, et al.
DeepMind Technologies
Atari 2600 games
Problem Statement

• Build a single agent that can learn to play any of the 7 atari 2600 games.

• Input:
  – 210 X 60 RGB video at 60hz (or 60 frames per second)
  – Game score
  – Set of game commands

• Output:
  – A command sequence to maximize the game score
Reinforcement learning

- Agent: To be build using deep
- Action set: Set of valid game controls
- Environment: Atari 2600
- Reward: Difference in the game score.
  - \( R(t) = \text{Score}(t-1) - \text{Score}(t) \)
- State: Every 3rd frame
On the nature of reward (1)

- Reward signal depends on the past actions not just the current action.
On the nature of reward (2)

• Immediate rewards are more valuable than future rewards.
  – $100 today is more valuable than $100 after 10 years
• Is it always true?
On the nature of reward (3)

• This paper makes the following assumption about rewards:
  – Goal of the agent is to maximize $R_t$

\[
R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}
\]

  – Where gamma is the discount factor, T is the time step at which the game terminates
Definitions

- **Policy** is a function which chooses the right action for every state.
- **State** is defined as a sequence of observation and action: \( s_t = x_1, a_1, x_2, a_2, \ldots, x_t, a_t \)
- **Action-value function**:
  \[
  Q^*(s, a) = \max_{\pi} \mathbb{E} [R_t | s_t = s, a_t = a, \pi]
  \]
- **Bellman equation**:
  \[
  Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} [r + \gamma \max_{a'} Q^*(s', a') | s, a]
  \]
Q-learning

• Agent maintains a table of Q[S,A] where S is the set of states and A is the set of actions.
• Q[S,A] is the current estimate of Q*(s,a) and is initialized by the designer.
• Agent picks an action and receives a reward.
  – Experience : <s,a,r,s’>
• This experience is used to update the Q[S,A] using:
  – Q[s,a] ← Q[s,a] + α(r + γmax_a' Q[s',a'] - Q[s,a])
• Alpha is called the Q-learning rate.
• It can be shown that Q[S,A] --> Q*(s,a)
### Example

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$Q[s_0, right]$</th>
<th>$Q[s_1, upC]$</th>
<th>$Q[s_3, upC]$</th>
<th>$Q[s_5, left]$</th>
<th>$Q[s_4, left]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>4.5</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>0.35</td>
<td>6.0</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-0.92</td>
<td>1.36</td>
<td>6.75</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>0.03</td>
<td>0.51</td>
<td>4</td>
<td>8.1</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>2.54</td>
<td>4.12</td>
<td>6.82</td>
<td>9.5</td>
<td>11.34</td>
</tr>
<tr>
<td>1000</td>
<td>4.63</td>
<td>5.93</td>
<td>8.46</td>
<td>11.3</td>
<td>13.4</td>
</tr>
<tr>
<td>10,000</td>
<td>6.08</td>
<td>7.39</td>
<td>9.97</td>
<td>12.83</td>
<td>14.9</td>
</tr>
<tr>
<td>100,000</td>
<td>7.27</td>
<td>8.58</td>
<td>11.16</td>
<td>14.02</td>
<td>16.08</td>
</tr>
<tr>
<td>1,000,000</td>
<td>8.21</td>
<td>9.52</td>
<td>12.1</td>
<td>14.96</td>
<td>17.02</td>
</tr>
<tr>
<td>10,000,000</td>
<td>8.96</td>
<td>10.27</td>
<td>12.85</td>
<td>15.71</td>
<td>17.77</td>
</tr>
<tr>
<td>$\infty$</td>
<td>11.85</td>
<td>13.16</td>
<td>15.74</td>
<td>18.6</td>
<td>20.66</td>
</tr>
</tbody>
</table>
Q-learning

• This is very slow as Q* is estimated for every state which in this case is a sequence
• Instead of learning the entire function, use function approximators.

\[ Q(s, a; \theta) \approx Q^*(s, a). \]

• Where Q is a parameterized non-linear function approximator like neural-network.
• The goal is to find the right set of parameters such that neural network approximates Q*(s,a)
Q-network

- Loss function:
  \[ L_i (\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right] , \]

- Where:
  \[ y_i = \mathbb{E}_{s' \sim \mathcal{E}} [r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a] \]

- Differentiating the loss function with respect to the weights gives the following gradient:
  \[ \nabla_{\theta_i} L_i (\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right] \]
Q-network is 14 years old

- TD-gammon is a backgammon player with superhuman performance.
- Developed by Gerald Tesauro from IBM in 1992
Whats new?

- In contrast to the TD-gammon Q-network, this paper used a technique called “experience replay”
- Agent’s experience $e_t = <s,a,r,s'>$ at each time step is stored.
- Data set $D = \{ e_1, e_2, \ldots, e_n \}$ is a collection of experiences
- During Q-learning instead of learning from a sequence of experience, learn from a randomly drawn sample of experiences.
Deep Q-learning

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 $\epsilon$ 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
Advantages of experience replay

1. Each step of experience is used in many weight updates => greater data efficiency

2. When learning on-policy current parameters determine the next data sample to be trained on.
   - If maximizing action is to move left then training samples will be dominated by samples from the left hand side.
   - Might cause unwanted feedback loop and parameters could get stuck in local minima.
   - Experience replay on the other hand randomized experiences
Experiment

- Raw image is downsampled cropped to produce a 84 X 84 image
- Agent is expected to produce output for once every 4 frames, and hence input to the neural network is 4 X 84 X 84
- The agent was trained for 10 million frames using a replay memory of 1 million most recent frames.
Results

- **Average Reward on Breakout**
  - Graph shows the reward per episode over training epochs.

- **Average Reward on Seaquest**
  - Graph shows the reward per episode over training epochs.

- **Average Q on Breakout**
  - Graph shows the Q value over training epochs.

- **Average Q on Seaquest**
  - Graph shows the Q value over training epochs.
Results

DQN performs better than human for 3 games and provides state of the art performance for 6 games

<table>
<thead>
<tr>
<th></th>
<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>
<td>179</td>
</tr>
<tr>
<td>Contingency [4]</td>
<td>1743</td>
<td>6</td>
<td>159</td>
<td>-17</td>
<td>960</td>
<td>723</td>
<td>268</td>
</tr>
<tr>
<td>DQN</td>
<td>4092</td>
<td>168</td>
<td>470</td>
<td>20</td>
<td>1952</td>
<td>1705</td>
<td>581</td>
</tr>
<tr>
<td>Human</td>
<td>7456</td>
<td>31</td>
<td>368</td>
<td>-3</td>
<td>18900</td>
<td>28010</td>
<td>3690</td>
</tr>
<tr>
<td>HNet Best [8]</td>
<td>3616</td>
<td>52</td>
<td>106</td>
<td>19</td>
<td>1800</td>
<td>920</td>
<td>1720</td>
</tr>
<tr>
<td>HNet Pixel [8]</td>
<td>1332</td>
<td>4</td>
<td>91</td>
<td>-16</td>
<td>1325</td>
<td>800</td>
<td>1145</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>
<td>1075</td>
</tr>
</tbody>
</table>

Table 1: The upper table compares average total reward for various learning methods by running an $\epsilon$-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNet and DQN. HNet produces deterministic policies that always get the same score while DQN used an $\epsilon$-greedy policy with $\epsilon = 0.05$. 
Thank you