Deep Reinforcement Learning

1. Learning to play Atari arcade games

2. Learning to play Go
Learning to play Atari video games with Deep Reinforcement Learning

Mnih et al., DeepMind Technologies, 2013

• Deep learning:
  – Requires large amount of hand-labeled data
  – Assumes data samples are iid, with stationary distribution

• Reinforcement learning:
  – Must learn from sparse, noisy, delayed reward function
  – “Samples” are not independent
  – Data distribution can change as system learns “online”.
Learning to play Atari video games with Deep Reinforcement Learning

Mnih et al., DeepMind Technologies, 2013

- https://www.youtube.com/watch?v=Up-a5x3coC0

- Uses convolutional neural network:
  - Input is raw pixels of video frames (~ the “state”)

- Output is estimated $Q(s,a)$ for each possible action
• Their system learns to play Atari 2600 games.

• 210x160 RGB video at 60 Hz.

• Designed to be difficult for human players

• “Our goal is to create a single neural network agent that is able to successfully learn to play as many of the games as possible.”

• No game-specific info provided. No hand-designed visual features.

• Learns exclusively from video input, the reward, and terminal signals. Network architecture and hyperparameters kept constant across all games.
- **CNN:** Input is set of 4 most recent video frames, output vector is $Q$-value of each possible action, given the input state.

- At each time step $t$, agent selects action $a_t$ from set of legal actions $A = \{1, \ldots, K\}$.

- Action $a_t$ is passed to emulator.

- Game state and current score are updated.

- Agent observes image $x_t$ from emulator (vector of raw pixels representing current screen).

- Agent receives reward $r_t$ representing change in game score. (Usually 0.)
Loss function:

\[
L_t(\theta_t) = \frac{1}{2} \left( r + \gamma \max_{a'} Q(s', a') - Q(s, a_t) \right)^2
\]
Three possible sources of training instability / non-convergence:

1. Temporally adjacent “experiences” \((s, a, r, s')\) are clearly not independent or identically distributed.

2. Small updates to \(Q\) can significantly change the policy

3. Correlations between the action values \(Q\) and “target” values
\[
r + \gamma \max_{a'} Q(s', a')
\]

Proposed methods to address these:

1. Use “experience replay”

2. Use two networks: one for action values, \(Q(s, a ; \theta)\), and one for targets, \(Q(s, a ; \theta^-)\), and only update the target network periodically.
Experience Replay

During training, to alleviate problem of correlated data and non-stationary distributions, use “experience reply” mechanism, which randomly samples previous transitions, and “thereby smooths the training distribution over many past behaviors.”

“During learning, we apply Q-learning updates, on samples (or minibatches) of experience \((s,a,r,s')\) drawn uniformly at random from the pool of stored samples.”
Separating action-value and target networks

Use two networks:

\( Q(s, a; \theta) \) is the network that estimates the current estimated value of taking action \( a \) in state \( s \)

\( Q(s', a'; \theta^-) \) is the network used for estimating the target:

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

New loss function:

\[
    L_t(\theta_t) = \left( r + \gamma \max_{a'} Q(s', a' | \theta^-) - Q(s, a | \theta_t) \right)^2
\]
Updating networks

Every time step, $Q(s, a; \theta_t)$ is updated using the target computed using $Q(s', a'; \theta_{t-})$.

(Minimize loss function via stochastic gradient descent on $\theta_t$ (weights).)

Every $C$ time steps, $\theta_{t-}$ is replaced by $\theta$. 
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

$$\nabla_{\theta_i} L_i (\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right].$$ (3)
Experiments

• Used same CNN architecture, learning algorithm, and parameters across a suite of Atari games.
Video

https://www.youtube.com/watch?v=Ih8EfvOzBOY
More detailed results

Space Invaders

Seaquest

Epsilon-greedy action section, with epsilon = 0.05
Extended Data Table 4 | Comparison of DQN performance with linear function approximator

<table>
<thead>
<tr>
<th>Game</th>
<th>DQN</th>
<th>Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakout</td>
<td>316.8</td>
<td>3.00</td>
</tr>
<tr>
<td>Enduro</td>
<td>1006.3</td>
<td>62.0</td>
</tr>
<tr>
<td>River Raid</td>
<td>7446.6</td>
<td>2346.9</td>
</tr>
<tr>
<td>Seaquest</td>
<td>2894.4</td>
<td>656.9</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>1088.9</td>
<td>301.3</td>
</tr>
</tbody>
</table>

The performance of the DQN agent is compared with the performance of a linear function approximator on the 5 validation games (that is, where a single linear layer was used instead of the convolutional network, in combination with replay and separate target network). Agents were trained for 10 million frames using standard hyperparameters, and three different learning rates. Each agent was evaluated every 250,000 training frames for 135,000 validation frames and the highest average episode score is reported. Note that these evaluation episodes were not truncated at 5 min leading to higher scores on Enduro than the ones reported in Extended Data Table 2. Note also that the number of training frames was shorter (10 million frames) as compared to the main results presented in Extended Data Table 2 (50 million frames).
“The performance of DQNs is normalized with respect to a professional human games tester (that is, 100% level) and random play (that is, 0% level).

Note that the normalized performance of DQN, expressed as a percentage, is calculated as:

$$100 \times \frac{\text{DQN score} - \text{random play score}}{\text{human score} - \text{random play score}}$$
Mastering the game of Go with deep neural networks and tree search

Silver et al., 2016

https://www.youtube.com/watch?v=53YLZBSS0cc