Asynchronous Methods for Deep Reinforcement Learning

Hardware Acceleration for Data Processing
Fall 2017
Ingredients of Reinforcement Learning

- Set of States $S$
- Set of Actions $A$

Agent observes $s_t \in S$ and...
...takes action $a_t \in A$ according to its policy...
...receiving a new $s_{t+1} \in S$ and a reward $r_{t+1} \in \mathbb{R}$.

- Total Reward:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \quad \gamma \in (0, 1)$$
What makes RL difficult?

- Delayed rewards
- Vast state spaces
- Dynamic environment / Actions influence training data
Idea: Tabulate “quality” of state-action-pairs

\[ Q^\pi(s, a) = \mathbb{E}[R_t|s_t = s, a, \pi] \]

Choose action that maximizes Q (\(\varepsilon\)-greedily)

For an optimal policy it holds that:

\[
Q^*(s, a) = \begin{cases} 
    r + \gamma \max_{a'} Q^*(s', a') & \text{if } s' \text{ terminal} \\
    r + \gamma \mathbb{E}[R_t|s_t = s, a, \pi] & \text{else}
\end{cases}
\]
Solving RL: Q-Learning (2)
[Watkins 89]

Iterate Two Steps:

1. Choose action that maximizes Q (ε-greedily)

2. Update table according to

   \[ Q^*(s, a) = \begin{cases} 
   r & \text{if } s' \text{ terminal} \\
   r + \gamma \max_{a'} Q^*(s', a') & \text{else} 
   \end{cases} \]

3. Repeat until convergence
Problem: Q-Tables Do Not Scale

<table>
<thead>
<tr>
<th>Input State</th>
<th>Action 1</th>
<th>...</th>
<th>Action 361</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>1</td>
<td>...</td>
<td>0.5</td>
</tr>
<tr>
<td>State 2</td>
<td>0</td>
<td>...</td>
<td>0.6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>State $10^{170}$</td>
<td>0.7</td>
<td>...</td>
<td>2</td>
</tr>
</tbody>
</table>

Q-values
Reinforcement Learning with Neural Networks

\[ Q(s, a, \theta) \approx Q^*(s, a) \]
Reinforcement Learning with *Deep* Neural Networks

[Mnih et al. 2015]

Input State Representation → 

Taken from “Human-level control through deep reinforcement learning” (Mnih et al. 2015)
Deep Q-Learning

Network is parameterized by $\theta$

$\theta$ can be trained by minimizing the loss

$$L(\theta) = \mathbb{E}[(targetQ - estimatedQ)^2]$$

$$= \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta) \right)^2 \right]$$
Issues with Deep Q-Learning

- Delayed rewards
- Vast state spaces
- Dynamic environment / Actions influence training data
- Feedback loops
- Updates affect Q-function globally
**Breaking Feedback Loops**

- **State** $s'$ (after $a$)
- **Target Network** ($\theta^-$)
- **Action** $a$
- **Trained Network** ($\theta$)
- **Q-values for** $s$
- **Reward for** $a$
- $\max_{a'} Q(s', a', \theta^-)$
- $\frac{\partial L}{\partial \theta}$
- $Q(s, a, \theta)$
Breaking Feedback Loops

State $s'$ (after $a$)

Target Network ($\theta^-$)

State $s$

Q-values for $s$

Action $a$

$\theta$
Issues with Deep Q-Learning

- Delayed rewards
- Vast state spaces
- Dynamic environment / Actions influence training data
- Feedback loops
- Updates affect Q-function globally
Experience Replay

**Problem**
Learning from new experiences might override previous progress!

**Idea**
Store state transitions during exploration
Sample from past experience during learning to maintain diversity
Issues with Deep Q-Learning

- Delayed rewards
- Vast state spaces
- Dynamic environment / Actions influence training data
- Feedback loops
- Updates affect Q-function globally
N-Step Methods

**Problem**
Rewards propagate through the DQN slowly!

**Idea**
Take multiple actions in sequence before updating the network
Parallel Deep Q-Learning

Parameter Server

Agent 1

Agent ...

Agent N

\( \theta \)
Massively Parallel Methods of Deep RL
[Nair et al. 2015]

- Distributed version of DQN
- (Then) State-of-the-art results on the Atari
- Training takes **days on 100+ machines**

Taken from “Massively Parallel Methods of Deep RL” (Nair et al. 2015)
Going HOGWILD!

Single Machine

Agent / Thread 1

Agent / Thread ...

Agent / Thread N

θ

θ

θ
Asynchronous one-step Q-learning
[Mnih et al. 2016]

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

```
// Assume global shared \( \theta, \theta^- \), and counter \( T = 0 \).
Initialize thread step counter \( t \leftarrow 0 \)
Initialize target network weights \( \theta^- \leftarrow \theta \)
Initialize network gradients \( d\theta \leftarrow 0 \)
Get initial state \( s \)
repeat
    Take action \( a \) with \( \epsilon \)-greedy policy based on \( Q(s, a; \theta) \)
    Receive new state \( s' \) and reward \( r \)
    \[ y = \begin{cases} 
    r & \text{for terminal } s' \\
    r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' 
    \end{cases} \]
    Accumulate gradients wrt \( \theta \): \( d\theta \leftarrow d\theta + \frac{\partial(y - Q(s, a; \theta))^2}{\partial \theta} \)
    \( s = s' \)
    \( T \leftarrow T + 1 \) and \( t \leftarrow t + 1 \)
    if \( T \mod I_{target} == 0 \) then
        Update the target network \( \theta^- \leftarrow \theta \)
    end if
    if \( t \mod I_{AsyncUpdate} == 0 \) or \( s \) is terminal then
        Perform asynchronous update of \( \theta \) using \( d\theta \).
        Clear gradients \( d\theta \leftarrow 0 \).
    end if
until \( T > T_{max} \)
```
What about Experience Replay?

Experience Replay was scrapped!

Intuition
Parallel learners explore different paths anyway!
Parallelism similar effect to experience replay.

Diversity can be enforced by different policies
Policy Gradient Methods

Policy is **parametrized directly** with

\[ \pi(a_t|s_t; \theta) \]

and estimated using gradient ascent for

\[ \nabla_\theta \log \pi(a_t|s_t; \theta) R_t \]

which is an unbiased estimate of

\[ \nabla_\theta \mathbb{E}[R_t] \]
Asynchronous Advantage Actor-Critic (A3C)

**Advantage Actor-Critic**

\[ V_\theta \log \pi(a_t|s_t; \theta) (R_t - V(s_t, \theta_v)) \]

**Note:**

- \( R_t \) depends on action taken \( a_t \)
- \( V(s_t, \theta_v) \) depends on the state only

\( R_t - V(s_t, \theta_v) \) is called the advantage of \( a_t \)
Asynchronous Advantage Actor-Critic (A3C)  
[Mnih et al. 2016]

\[ \pi(a_t | s_t; \theta) \]

\[ V(s_t, \theta_v) \]

---

**Algorithm S3** Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors \( \theta \) and \( \theta_v \) and global shared counter \( T = 0 \)
// Assume thread-specific parameter vectors \( \theta' \) and \( \theta'_v \)

```
Initialize thread step counter \( t \leftarrow 1 \)

repeat
  Reset gradients: \( d\theta \leftarrow 0 \) and \( d\theta_v \leftarrow 0 \).
  Synchronize thread-specific parameters \( \theta' = \theta \) and \( \theta'_v = \theta_v \)
  \( t_{\text{start}} = t \)
  Get state \( s_t \)
  repeat
    Perform \( a_t \) according to policy \( \pi(a_t | s_t; \theta') \)
    Receive reward \( r_t \) and new state \( s_{t+1} \)
    \( t \leftarrow t + 1 \)
    \( T \leftarrow T + 1 \)
  until terminal \( s_t \) or \( t - t_{\text{start}} = t_{\text{max}} \)

  \( R = \begin{cases} 
    0 & \text{for terminal } s_t \\
    V(s_t, \theta'_v) & \text{for non-terminal } s_t \end{cases} \) // Bootstrap from last state

  for \( i \in \{t-1, \ldots, t_{\text{start}}\} \) do
    \( R \leftarrow r_i + \gamma R \)
    Accumulate gradients wrt \( \theta' \): \( d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i | s_i; \theta') (R - V(s_i, \theta'_v)) \)
    Accumulate gradients wrt \( \theta'_v \): \( d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i, \theta'_v))^2 / \partial \theta'_v \)
  end for

  Perform asynchronous update of \( \theta \) using \( d\theta \) and of \( \theta_v \) using \( d\theta_v \).

until \( T > T_{\text{max}} \)
## Evaluation on the Atari Domain

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Time</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>8 days on GPU</td>
<td>121.9%</td>
<td>47.5%</td>
</tr>
<tr>
<td>Gorilla</td>
<td>4 days, 100 machines</td>
<td>215.2%</td>
<td>71.3%</td>
</tr>
<tr>
<td>D-DQN</td>
<td>8 days on GPU</td>
<td>332.9%</td>
<td>110.9%</td>
</tr>
<tr>
<td>Dueling D-DQN</td>
<td>8 days on GPU</td>
<td>343.8%</td>
<td>117.1%</td>
</tr>
<tr>
<td>Prioritized DQN</td>
<td>8 days on GPU</td>
<td>463.6%</td>
<td>127.6%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>1 day on CPU</td>
<td>344.1%</td>
<td>68.2%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>4 days on CPU</td>
<td>496.8%</td>
<td>116.6%</td>
</tr>
<tr>
<td>A3C, LSTM</td>
<td>4 days on CPU</td>
<td>623.0%</td>
<td>112.6%</td>
</tr>
</tbody>
</table>

Normalized scores on 57 Atari Games [Mnih et. al 2016]
Evaluation
Asynchronous Methods vs. DQN

Averaged scores of different algorithms for 5 games over time [Mnih et. al 2016]
(Super-) Linear Speedup

Average speedup of different algorithms for different thread counts [Mnih et. al 2016]

<table>
<thead>
<tr>
<th>Method</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-step Q</td>
<td>1.0</td>
<td>3.0</td>
<td>6.3</td>
<td>13.3</td>
<td>24.1</td>
</tr>
<tr>
<td>1-step SARSA</td>
<td>1.0</td>
<td>2.8</td>
<td>5.9</td>
<td>13.1</td>
<td>22.1</td>
</tr>
<tr>
<td>n-step Q</td>
<td>1.0</td>
<td>2.7</td>
<td>5.9</td>
<td>10.7</td>
<td>17.2</td>
</tr>
<tr>
<td>A3C</td>
<td>1.0</td>
<td>2.1</td>
<td>3.7</td>
<td>6.9</td>
<td>12.5</td>
</tr>
</tbody>
</table>

Average speedup of A3C for different thread counts [Mnih et. al 2016]
Evaluation with TORCS

Averaged scores of different algorithms for TORCS racing simulator [Mnih et. al 2016]
Stability & Robustness

Scores of A3C with different learning rates for 5 games [Mnih et. al 2016]
## Stability & Robustness

Raw scores of different algorithms for a selection of Atari Games [Mnih et. al 2016]

<table>
<thead>
<tr>
<th>Game</th>
<th>DQN</th>
<th>Gorila</th>
<th>Double</th>
<th>Dueling</th>
<th>Prioritized</th>
<th>A3C FF, 1 day</th>
<th>A3C FF</th>
<th>A3C LSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowling</td>
<td>41.2</td>
<td>54.0</td>
<td>69.6</td>
<td>65.7</td>
<td>65.8</td>
<td>36.2</td>
<td>35.1</td>
<td>41.8</td>
</tr>
<tr>
<td>Boxing</td>
<td>25.8</td>
<td>74.2</td>
<td>73.5</td>
<td>77.3</td>
<td>68.6</td>
<td>33.7</td>
<td>59.8</td>
<td>37.3</td>
</tr>
<tr>
<td>Breakout</td>
<td>303.9</td>
<td>313.0</td>
<td>368.9</td>
<td>411.6</td>
<td>371.6</td>
<td>551.6</td>
<td>681.9</td>
<td>766.8</td>
</tr>
<tr>
<td>Centipede</td>
<td>3773.1</td>
<td>6296.9</td>
<td>3853.5</td>
<td>4881.0</td>
<td>3421.9</td>
<td>3306.5</td>
<td>3755.8</td>
<td>1997.0</td>
</tr>
<tr>
<td>Chopper Comman</td>
<td>3046.0</td>
<td>3191.8</td>
<td>3495.0</td>
<td>3784.0</td>
<td>6604.0</td>
<td>4669.0</td>
<td>7021.0</td>
<td>10150.0</td>
</tr>
<tr>
<td>Crazy Climber</td>
<td>50992.0</td>
<td>65451.0</td>
<td>113782.0</td>
<td>124566.0</td>
<td>131086.0</td>
<td>101624.0</td>
<td>112646.0</td>
<td>138518.0</td>
</tr>
<tr>
<td>Defender</td>
<td>27510.0</td>
<td>33996.0</td>
<td>21093.5</td>
<td>36242.5</td>
<td>56533.0</td>
<td>233021.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Demon Attack</td>
<td>12835.2</td>
<td>14880.1</td>
<td>69803.4</td>
<td>56322.8</td>
<td>73185.8</td>
<td>84997.5</td>
<td>113308.4</td>
<td>115201.9</td>
</tr>
<tr>
<td>Double Dunk</td>
<td>-21.6</td>
<td>-11.3</td>
<td>-0.3</td>
<td>-0.8</td>
<td>2.7</td>
<td>0.1</td>
<td>-0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Enduro</td>
<td>475.6</td>
<td>71.0</td>
<td>1216.6</td>
<td>2077.4</td>
<td>1884.4</td>
<td>-82.2</td>
<td>-82.5</td>
<td>-82.5</td>
</tr>
<tr>
<td>Fishing Derby</td>
<td>-2.3</td>
<td>4.6</td>
<td>3.2</td>
<td>-4.1</td>
<td>9.2</td>
<td>13.6</td>
<td>18.8</td>
<td>22.6</td>
</tr>
<tr>
<td>Freeway</td>
<td>25.8</td>
<td>10.2</td>
<td>28.8</td>
<td>0.2</td>
<td>27.9</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Takeaway Message

- Deep Reinforcement Learning is hard
- Requires techniques like experience replay
- Deep RL is easily parallelizable
- Parallelism can replace experience replay
- Dropping experience replay allows on-policy methods like actor-critic
- A3C surpasses state-of-the-art performance
QUESTIONS?
One-step Q-learning vs. N-step Q-learning

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared \( \theta, \theta' \) , and counter \( T = 0 \).
Initialize thread step counter \( t \leftarrow 0 \)
Initialize target network weights \( \theta^- \leftarrow \theta \)
Initialize network gradients \( d\theta \leftarrow 0 \)
Get initial state \( s \)
repeat
  Take action \( a \) with \( \epsilon \)-greedy policy based on \( Q(s,a;\theta) \)
  Receive new state \( s' \) and reward \( r \)
  \( y = \begin{cases} 
  r & \text{for terminal } s' \\
  r + \gamma \max_{a'} Q(s',a';\theta^-) & \text{for non-terminal } s' 
  \end{cases} \)
  Accumulate gradients wrt \( \theta \): \( d\theta \leftarrow d\theta + \frac{\partial(y - Q(s,a;\theta))^2}{\partial \theta} \)
  \( s = s' \)
  \( T \leftarrow T + 1 \) and \( t \leftarrow t + 1 \)
  if \( T \mod I_{\text{target}} == 0 \) then
    Update the target network \( \theta^- \leftarrow \theta \)
  end if
  if \( t \mod I_{\text{AsyncUpdate}} == 0 \) or \( s \) is terminal then
    Perform asynchronous update of \( \theta \) using \( d\theta \).
    Clear gradients \( d\theta \leftarrow 0. \)
  end if
until \( T > T_{\text{max}} \)

Algorithm 2 Asynchronous n-step Q-learning - pseudocode for each actor-learner thread.

// Assuming global shared parameter vector \( \theta \).
// Assuming global shared target parameter vector \( \theta^- \).
// Assuming global shared counter \( T = 0 \).
Initialize thread step counter \( t \leftarrow 1 \)
Initialize target network parameters \( \theta^- \leftarrow \theta \)
Initialize thread-specific parameters \( \theta' = \theta \)
Initialize network gradients \( d\theta \leftarrow 0 \)
repeat
  Clear gradients \( d\theta \leftarrow 0 \)
  Synchronize thread-specific parameters \( \theta' = 0 \)
  \( l_{\text{start}} = l \)
  Get state \( s_t \)
  repeat
    Take action \( a_t \) according to the \( \epsilon \)-greedy policy based on \( Q(s_t,a_t;\theta') \)
    Receive reward \( r_t \) and new state \( s_{t+1} \)
    \( t \leftarrow t + 1 \)
    \( T \leftarrow T + 1 \)
    if terminal \( s_t \) or \( t - l_{\text{start}} == l_{\text{max}} \) then
      Update the target network \( \theta^- \leftarrow \theta \)
    end if
    for \( s \in \{l - 1, \ldots, l_{\text{start}}\} \) do
      \( R = \begin{cases} 
      0 & \text{for terminal } s_t \\
      \max_{a} Q(s_t,a;\theta^-) & \text{for non-terminal } s_t 
      \end{cases} 
      \)
      \( R = r_t + \gamma R \)
      Accumulate gradients wrt \( \theta' \): \( d\theta \leftarrow d\theta + \frac{\partial(R - Q(s,a;\theta'))^2}{\partial \theta'} \)
    end for
    Perform asynchronous update of \( \theta \) using \( d\theta \).
    if \( T \mod I_{\text{target}} == 0 \) then
      \( \theta^- \leftarrow \theta \)
    end if
  until \( T > T_{\text{max}} \)
Q-Learning: A Toy Example

Current Q-Table

<table>
<thead>
<tr>
<th></th>
<th>Backwards</th>
<th>Forwards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Street</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Goal</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Hyperparameters: $\epsilon = 0.999 \quad \gamma = 0.5$

Next Action: Go Forward
Q-Learning: A Toy Example

Previous Action: Go Forwards
No Reward

Next Action: Go Forwards
### Q-Learning: A Toy Example

**Current Q-Table**

<table>
<thead>
<tr>
<th></th>
<th>Backwards</th>
<th>Forwards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Street</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goal</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Previous Action: Go Forwards
Reward: +1

Update Table According to $Q^*(s, a) = \begin{cases} r & \text{if } s' \text{ terminal} \\ r + \gamma \max_{a'} Q^*(s', a') & \text{else} \end{cases}$
Q-Learning: A Toy Example

Current Q-Table

<table>
<thead>
<tr>
<th></th>
<th>Backwards</th>
<th>Forwards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Street</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goal</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Game has restarted!

Next Action: Go Forwards
Q-Learning: A Toy Example

### Current Q-Table

<table>
<thead>
<tr>
<th></th>
<th>Backwards</th>
<th>Forwards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>Street</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goal</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Previous Action: Go Forwards,
No reward, but rewarding follow-up action

Update Table According to $Q^*(s, a) = \begin{cases} r + \gamma \max_{a'} Q^*(s', a') & \text{if } s' \text{ terminal} \\ r & \text{else} \end{cases}$

Next Action: Go Backwards
Q-Learning: A Toy Example

Current Q-Table

<table>
<thead>
<tr>
<th></th>
<th>Backwards</th>
<th>Forwards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>Street</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>Goal</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Previous Action: Go Backwards
No reward, but rewarding follow-up action

Update Table According to $Q^*(s, a) = \begin{cases} r & \text{if } s' \text{ terminal} \\ r + \gamma \max_{a'} Q^*(s', a') & \text{else} \end{cases}$

Next Action: Go Backwards
Q-Learning: A Toy Example

Current Q-Table

<table>
<thead>
<tr>
<th></th>
<th>Backwards</th>
<th>Forwards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0.25</td>
<td>0.5</td>
</tr>
<tr>
<td>Street</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>Goal</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Done