Learning 2048 with Deep Reinforcement Learning

Zachariah Levine
Department of Computer Science, University of Waterloo
Outline

- Motivation
- Reinforcement Learning
- Q-Learning
- Six (Unofficial) Stages of Deep Q-Learning
- 2048-unlimited
- Implementation Overview
- Research and Results
- Demonstration and Interactive Results
- Future Work
- Conclusion
- References
Motivation

- Applications of Deep Reinforcement Learning
  - Games
  - Self Driving Cars
  - Manufacturing
  - Robotics
  - Natural Language Processing
  - Computer Vision
  - Etc...
• How is it different?

- Reinforcement Learning

  - Labeled data
  - Direct feedback
  - Predict outcome/future

  - No labels
  - No feedback
  - “Find hidden structure”

  - Decision process
  - Reward system
  - Learn series of actions

  - Sparse and time-delayed labels

Ref. 10
Reinforcement Learning

Environment

Agent

State

Reward

Action
- Sparse and time-delayed labels
- Credit Assignment Problem
- Explore-Exploit Dilemma: Action Selection
  - Greedy Approach
  - Random Approach
  - Epsilon-Greedy Approach
Markov Decision Process

Ref. 11
Most common way to formalize a reinforcement learning problem

An episode of a Markov decision process is a finite sequence of states, actions, and rewards:

\[ s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n \]

An experience or transition is defined as:

\[ \langle s, a, r, s' \rangle \]

"A Markov decision process relies on the Markov assumption, that the probability of the next state \( s_{i+1} \) depends only on current state \( s_i \) and performed action \( a_i \), but not on preceding states or actions." (3)
Discounted Future Reward

- Total Reward:
  \[ R = r_1 + r_2 + r_3 + \ldots + r_n \]

- Total Future Reward:
  \[ R_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_n \]

- Discounted Future Reward:
  \[ R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-t} r_n \]

- Discounted Future Reward:
  \[ R_t = r_t + \gamma (r_{t+1} + \gamma (r_{t+2} + \ldots )) = r_t + \gamma R_{t+1} \]
"In Q-learning we define a function $Q^*(s,a)$ representing the discounted future reward when we perform action ‘$a$’ in state ‘$s$’, and continue optimally from that point on." (3)

$$Q^*(s_t, a_t) = \max_{\pi} R_{t+1}$$

Rewrite as the Bellman Equation:

$$Q^*(s, a) = r + \gamma \max_{a'} Q^*(s', a')$$

If we have $Q^*(s, a)$ then:

$$\pi(s) = \pi^*(s) = \arg\max_a Q^*(s, a)$$
However, we do not know $Q^*(s,a)$; therefore we must estimate it with a non-optimal function $Q(s,a)$. This enables us to define $Q^*(s,a)$ as

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$$

The whole idea behind Q-learning is that the Bellman equation can be used iteratively to improve our approximation of the optimal Q-function.

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$
Q-Learning

Bellman Equation:

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

Update for simple Q-Learning:

\[ Q(s, a) = Q(s, a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) \]

<table>
<thead>
<tr>
<th></th>
<th>Action 0</th>
<th>Action 1</th>
<th>...</th>
<th>Action n-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 0</td>
<td>Q(0, 0)</td>
<td>Q(0, 1)</td>
<td>...</td>
<td>Q(0, n-1)</td>
</tr>
<tr>
<td>State 1</td>
<td>Q(1, 0)</td>
<td>Q(1, 1)</td>
<td>...</td>
<td>Q(1, n-1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>State n-1</td>
<td>Q(n-1, 0)</td>
<td>Q(n-1, 1)</td>
<td>...</td>
<td>Q(n-1, n-1)</td>
</tr>
</tbody>
</table>
Deep Q Network

Problem? Too many states!
Solution? Use a Neural Network to approximate it!

Ref. 3
Now that we have a DQN all we need for deep reinforcement learning is a loss function,

\[ \mathcal{L} = \frac{1}{|B|} \sum_{(s,a,s',r) \in B} \mathcal{L}(\delta) \]

where \( \delta \) is temporal difference,

\[ \delta = Q(s,a) - \left( r + \gamma \max_a Q(s',a) \right) \]

\( \mathcal{L}(\delta) \) for MSE loss is,

\[ \mathcal{L}(\delta) = \frac{1}{2} \delta^2 \]

\( \mathcal{L}(\delta) \) for Huber Loss is,

\[ \mathcal{L}(\delta) = \begin{cases} 
\frac{1}{2} \delta^2 & \text{for } |\delta| \leq 1, \\
|\delta| - \frac{1}{2} & \text{otherwise} 
\end{cases} \]
Deep Q-Learning: Stage 2

- Add Experience Replay
  - Store transitions and sample batches during training
  - Stabilizes learning
  - Needed because successive experiences are highly correlated
Add a separate target network

\[ \delta = Q(s, a; \theta) - \left( r + \gamma \max_{a'} Q(s', a'; \theta^-) \right) \]

- The problem: “...the max operator uses the same values to both select and evaluate an action. This can therefore lead to overoptimistic value estimates.” (7)
- The target network is used to:
  - Determine \( a' \)
  - Evaluate state-action value of \( Q(s', a') \)
Double Deep Q-Networks
- Mitigates overoptimistic value estimates.

\[
\delta = Q(s, a; \theta) - (r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta); \theta^-))
\]

- Use the online network to determine \(a'\) and then use the target network as a measure of how good that action is \(Q(s', a')\).
Dueling Double Deep Q-Networks

\[ Q^{\pi}(s, a) = \mathbb{E}[R_{t+1} | s_t = s, a_t = a, \pi] \]

\[ V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}[Q^{\pi}(s, a)] \quad A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s) \]
Now we must combine the approximate value and advantage functions to form an approximate state-action value function.

\[ Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \]

\[ Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left( A(s, a; \theta, \alpha) - \max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha) \right) \]

\[ Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left( A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right) \]
Deep Q-Learning: Stage 6

Stage for further extensions such as:
- Prioritized Replay
- Continuous Action Domain
- Continuous target network updates

\[ \theta^- \leftarrow \tau \theta + (1 - \tau) \theta^- \]
What is 2048?
- Demo
- State space: realistically $\sim 15^{16}$, theoretically more.
- Action Space = \{0, 1, 2, 3\} or \{\text{<up>}, \text{<right>}, \text{<down>}, \text{<left>}\}
Implementation Overview

- Show a config file
- Huber Loss & MSE Loss & batch updates
- Gradient Clipping
- Double DQN
- Dueling DQN
  - Average Advantage
  - Max Advantage
- Target Network syncing
- Slow tracking
- Update frequency
- Adaptive Learning Rate
- Replay memory
- Epsilon decay mode = {linear, exponential, sinusoidal}
- Epsilon annealing duration
- Epsilon Explorer
- Agent knows best & unsticking agent
- Various activation functions: ReLU, ELU, SReLU
- Various Networks: Convolutional, Fully Connected, Self Normalizing Fully Connected
PyTorch; my own implementation starting from DQN State 2

Normalize states and rewards:

\[ \text{processed}_s = \frac{\log_2 s}{15} \]
\[ \text{processed}_r = \frac{\log_2 r}{15} \]

Epsilon Decay Modes
Epsilon Explorer
- A novel contribution: modify epsilon within an episode in addition to between episodes
- Goal: increase exploration as you get further in the episode and reduce exploration near the beginning of the episode
- See jupyter notebook

Use smaller epsilon values
Agent Knows Best
Unstick Agent
• ~26 runs with my code
  • We will look at a very small subset
• Exploration of the hyperparameter space was limited by computational constraints
  • See config file and networks module
• Best human performance is ~100,000
Research and Results

- `scspc677:run20170719_01`
- Parameters: 75236
- `epsilon_decay_mode = linear`
- `epsilon_annealing_duration = 20,000`
- `slow_tracking = False`
- `epsilon_explorer = False`
Research and Results

- **ubuntu1404:run20170719_01**
- **Parameters: 75236**
- **Compare to scspc677:run20170719_01**
- **epsilon_decay_mode = exponential**
- **epsilon_annealing_duration = 40,000**
- **slow_tracking = False**
- **epsilon_explorer = True**

**Neural Network: DQN_00**

```
1@4x4 Conv2d(1, 32, (2, 2)) 32@3x3 Conv2d(32, 64, (2, 2)) 64@2x2 fc 256 fc 4
```
- **ubuntu1404:run20170719_02**
- **Parameters**: 75236
- **Compare to**
  - scspc677:run20170719_01
- **slow_tracking = True**
- **epsilon_explorer = False**
Neural Network: DQN_02

- 1@4x4 → Conv2d(1, 128, (2, 2)) → 128@3x3
- Conv2d(128, 128, (2, 2)) → 128@2x2 → fc → 256 → fc → 4

- Parameters: 198660

- scspc675.cs:run20170720_03
Research and Results

- **scspc665:run20170721_01**
- Parameters: 920836
- This is the same run as **scspc675:run20170720_03** except:
  - Network 4 instead of network 2
Research and Results

- scspc675:run20170723_01
- Parameters: 330245
- This is the same run as scspc675:run20170720_03 except:
  - dueling_dqn = True instead of False
  - plateau length changed from annealing_duration to annealing_duration/4
Research and Results

Neural Network: DQN_02

1@4x4 Conv2d(1, 128, (2, 2)) 128@3x3 Conv2d(128, 128, (2, 2)) 128@2x2 fc 256 fc 4

- scspc675:run20170723_02
- Parameters: 198660
- This is the same run as scspc675:run20170720_03 except:
  - no penalty for a reward of 0
  - plateau length changed from annealing_duration to annealing_duration/4
Best results so far:
- Largest Tile = 4096
- Longest Episode Duration = 3127
- Highest Score = 67988
- Largest Mean Total Rewards = 15390
- Largest Mean Duration = 893
- My personal highest Tile = 2048
- My personal highest Score = 27556
• Show the demo and interactive results
We would like to experiment with ways that may increase the speed the model learns while avoiding longer training, longer annealing times, and larger models such as:
- Prioritized Experience Replay
- Epsilon Explorer

We would like to experiment more (in general):
- Larger networks
- Longer training/annealing
- Different Networks
- Wider variety of activation functions
To the best of our knowledge, this is the first successful application of Deep Q-Learning to 2048.

My Deep Learning Model can play better than I can on average.

The model is not yet at superhuman performance.

Agent Knows Best is beneficial.

We hypothesize that performance can be increased by:

- Longer training times
- Longer annealing times
- Larger models
- Prioritized Experience Replay
- Epsilon Explorer
References

1. https://gabrielecirulli.github.io/2048/