Reinforcement Learning

State $s_t$  $ightarrow$ Reward $r_t$  $ightarrow$ Action $a_t$

Agent

Environment

Image from reinforce.io
RL Example - Atari Games

- Observed States - Images.
- Internal state - RAM.
- Actions - Pressing Paddle.
- Reward - points from the game.
RL - Quick recap

Discounted future reward

\[ R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \]
\[ = r_t + \gamma (r_{t+1} + \gamma (r_{t+2} + \cdots)) \]
\[ = r_t + \gamma R_{t+1} \]
RL - Quick recap

Value-Action Function

• We define a $Q(s, a)$ representing the maximum discounted future reward when we perform action $a$ in state $s$:

$$Q(s_t, a_t) = \max R_{t+1}$$

• **Q-function**: represents the "Quality" of a certain action in a given state

• Imagine you have the magical Q-function

$$\pi(s) = \arg\max_a Q(s, a)$$

• $\pi$ is the policy
How DQN aggregates states

Figure 4. Seaquest aggregated states on the t-SNE map, colored by value function estimate.

Image Zahavy et al.
One of the main problems in Deep RL

- They require orders of magnitude more interactions than humans.
- DQN needs 200 hours of gameplay to achieve the similar scores to what a human can get in 2 hours. (on 47 atari games).

Question: What is the reason?
Explanation 1

Gradient descent optimisation requires the use of small learning rates.

(Also a problem for other methods which use deep learning).
Explanation 2

Environments usually have sparse reward signals.

- There is an Imbalance of low and high reward samples.
- Neural Network underperforms predicting larger rewards.
Explanation 3

Reward signal propagates to earlier states slowly.

We don’t update Q-values in reverse order because we want transitions in our mini batch to be uncorrelated.
Neural Episodic Control (NEC) - Overview

- Solves some of these problems - makes training a lot faster.
- Uses semi tabular representation by adding a differentiable memory structure - mapping (state, action) pairs to value estimates.
- The difference from other memory augmented RL is that this memory can be updated much faster than Neural Networks. (like LSTM or DNC).
Differentiable Neural Dictionary (DND)

For each action $a \in A$, NEC has a simple memory module $M_a = (K_a, V_a)$.

$K_a$ and $V_a$ are dynamically sized arrays of vectors, each containing same number of vectors.

This memory is just like a dictionary. Can be used outside RL, so it’s referred as DND.

Only append - no delete is needed.
Differentiable Neural Dictionary (DND)

Two possible operations - Writing and Lookup.
Differentiable Neural Dictionary (DND)

**Lookup** h- key, o- output:

\[ o = \sum_i w_i v_i, \]

\[ w_i = k(h, h_i) / \sum_j k(h, h_j), \]

Kernel, not to confuse with a key.

One module like this for each action.
Differentiable Neural Dictionary (DND)

Sum is in practice limited to the top $p$ nearest neighbours ($p=50$)

$$o = \sum_i w_i v_i,$$

$$w_i = k(h, h_i) / \sum_j k(h, h_j),$$

Nearest neighbours algorithm is approximated by KD-trees.
Differentiable Neural Dictionary (DND)

Writing

Simply append a new (key, value) vector pair.

If a key already exists just update a value.
Neural Episodic control (Agent)

Pixel state $S$ is processed by a ConvNet to produce a key $h$.

$h$ is then used to lookup a value from DND.

In case of NEC agent, values in DND are Q value estimates.

In the end this architecture produces $Q(s,a)$ for a single action $a$. 
NEC architecture
Neural Episodic Control

**Algorithm 1** Neural Episodic Control

- $D$: replay memory.
- $M_a$: a DND for each action $a$.
- $N$: horizon for $N$-step $Q$ estimate.

for each episode do

for $t = 1, 2, \ldots, T$ do

- Receive observation $s_t$ from environment with embedding $h$.
- Estimate $Q(s_t, a)$ for each action $a$ via (1) from $M_a$
- $a_t \leftarrow \epsilon$-greedy policy based on $Q(s_t, a)$
- Take action $a_t$, receive reward $r_{t+1}$
- Append $(h, Q^{(N)}(s_t, a_t))$ to $M_{a_t}$.
- Append $(s_t, a_t, Q^{(N)}(s_t, a_t))$ to $D$.
- Train on a random minibatch from $D$.

end for

end for
N step Q value estimate works better

\[ Q^{(N)}(s_t, a) = \sum_{j=0}^{N-1} \gamma^j r_{t+j} + \gamma^N \max_{a'} Q(s_{t+N}, a') \]

If h is already present in DND we update as:

\[ Q_i \leftarrow Q_i + \alpha (Q^{(N)}(s, a) - Q_i) . \]

If maximum Memory capacity is reached overwrite the keys which recently didn’t show up during KNN search.
NEC - Learning

Parameters are updated by minimizing the L2 loss between predicted Q value and N-step estimate Q(N).

Training is done on randomly sampled mini batches from the replay buffer, where (st, at, Rt) tuples are stored. (Rt is a Q(N) estimate here)

Following kernel is used: $k(h, h_i) = \frac{1}{\|h - h_i\|_2^2 + \delta}$.
NEC - Experiment params

- Compared to 5 variants of A3C and DQN
- All algorithms are trained with discount factor 0.99
- Same convolutional architecture as DQN
- N-step Q estimate, N = 100
- Replay buffer stores the last $10^5$ states as opposed to $10^6$ of DQN.
- Note - rewards aren’t clipped to [-1, 1]
## NEC - results

<table>
<thead>
<tr>
<th>Frames</th>
<th>Nature DQN</th>
<th>Q*(λ)</th>
<th>Retrace(λ)</th>
<th>Prioritised Replay</th>
<th>A3C</th>
<th>NEC</th>
<th>MFEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>-0.7%</td>
<td>-0.8%</td>
<td>-0.4%</td>
<td>-2.4%</td>
<td>0.4%</td>
<td>16.7%</td>
<td>12.8%</td>
</tr>
<tr>
<td>2M</td>
<td>0.0%</td>
<td>0.1%</td>
<td>0.2%</td>
<td>0.0%</td>
<td>0.9%</td>
<td>27.8%</td>
<td>16.7%</td>
</tr>
<tr>
<td>4M</td>
<td>2.4%</td>
<td>1.8%</td>
<td>3.3%</td>
<td>2.7%</td>
<td>1.9%</td>
<td>36.0%</td>
<td>26.6%</td>
</tr>
<tr>
<td>10M</td>
<td>15.7%</td>
<td>13.0%</td>
<td>17.3%</td>
<td>22.4%</td>
<td>3.6%</td>
<td>54.6%</td>
<td>45.4%</td>
</tr>
<tr>
<td>20M</td>
<td>26.8%</td>
<td>26.9%</td>
<td>30.4%</td>
<td>38.6%</td>
<td>7.9%</td>
<td>72.0%</td>
<td>55.9%</td>
</tr>
<tr>
<td>40M</td>
<td>52.7%</td>
<td>59.6%</td>
<td>60.5%</td>
<td>89.0%</td>
<td>18.4%</td>
<td>83.3%</td>
<td>61.9%</td>
</tr>
</tbody>
</table>

Table 1. Median across games of human-normalised scores for several algorithms at different points in training

<table>
<thead>
<tr>
<th>Frames</th>
<th>Nature DQN</th>
<th>Q*(λ)</th>
<th>Retrace(λ)</th>
<th>Prioritised Replay</th>
<th>A3C</th>
<th>NEC</th>
<th>MFEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>-10.5%</td>
<td>-11.7%</td>
<td>-10.5%</td>
<td>-14.4%</td>
<td>5.2%</td>
<td>45.6%</td>
<td>28.4%</td>
</tr>
<tr>
<td>2M</td>
<td>-5.8%</td>
<td>-7.5%</td>
<td>-5.4%</td>
<td>-5.4%</td>
<td>8.0%</td>
<td>58.3%</td>
<td>39.4%</td>
</tr>
<tr>
<td>4M</td>
<td>8.8%</td>
<td>6.2%</td>
<td>6.2%</td>
<td>10.2%</td>
<td>11.8%</td>
<td>73.3%</td>
<td>53.4%</td>
</tr>
<tr>
<td>10M</td>
<td>51.3%</td>
<td>46.3%</td>
<td>52.7%</td>
<td>71.5%</td>
<td>22.3%</td>
<td>99.8%</td>
<td>85.0%</td>
</tr>
<tr>
<td>20M</td>
<td>94.5%</td>
<td>135.4%</td>
<td>273.7%</td>
<td>165.2%</td>
<td>59.7%</td>
<td>121.5%</td>
<td>113.6%</td>
</tr>
<tr>
<td>40M</td>
<td>151.2%</td>
<td>440.9%</td>
<td>386.5%</td>
<td>332.3%</td>
<td>255.4%</td>
<td>144.8%</td>
<td>142.2%</td>
</tr>
</tbody>
</table>

Table 2. Mean human-normalised scores for several algorithms at different points in training
In games like **Ms. Pac-man and Alien** some improvement on final reward seems to be because of **no reward clipping**.

Other methods will tend to collect small rewards, while NEC and MFEC will try to actively make the enemies vulnerable and attack them to get large rewards.

In Ms. Pac-Man eating a single ghost can yield a reward of up to +1600, but DQN clips the reward to +1.
NEC - results
NEC - results
NEC - results

But NEC also outperforms the other algorithms on Pong and Boxing where reward clipping does not affect any of the algorithms as all original rewards are in the range $[-1, 1]$;
NEC - results
NEC - results
Future work suggestions

- This work suggest that non-parametric methods are a promising addition to the deep RL box.
- NEC outperforms Prioritised Replay in terms of speed but in the long term PR wins. Future work is to outperform it in the long run as well.
- Apply NEC on visually more challenging 3D worlds, where it might be even more important.
Final Slide

Thanks for listening!
Discussions (slide not related to the paper)

- "Pure" Reinforcement Learning (cherry)
  - The machine predicts a scalar reward given once in a while.
  - A few bits for some samples

- Supervised Learning (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - 10–10,000 bits per sample

- Unsupervised/Predictive Learning (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - Millions of bits per sample

(Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)