DeepMind Self-Learning Atari Agent

“Human-level control through deep reinforcement learning”
– Nature Vol 518, Feb 26, 2015

“The Deep Mind of Demis Hassabis” – Backchannel / Medium.com – interview with David Levy

“Advanced Topics: Reinforcement Learning” – class notes
David Silver, UCL & DeepMind

Nikolai Yakovenko
3/25/15 for EE6894
Motivations

“automatically convert unstructured information into useful, actionable knowledge”

“ability to learn for itself from experience”

“and therefore it can do stuff that maybe we don’t know how to program”

- Demi Hassabis
“If you play bridge, whist, whatever, I could invent a new card game...”

“and you would not start from scratch... there is transferable knowledge.”

Explicit 1st step toward self-learning intelligent agents, with transferable knowledge.
Why Games?

• Easy to create more data.

• Easy to compare solutions.

• (Relatively) easy to transfer knowledge between similar problems.

• But not yet.
“idea is to slowly widen the domains. We have a prototype for this – the human brain. We can tie our shoelaces, we can ride cycles & we can do physics, with the same architecture. So we know this is possible.”

- Demis Hassbis
What They Did

• An agent, that learns to play any of 49 Atari arcade games
  – Learns strictly from experience
  – Only game screen as input
  – No game-specific settings
DQN

- Novel agent, called deep Q-network (DQN)
  - Q-learning (reinforcement learning)
    - Choose actions to maximize “future rewards” Q-function
  - CNN (convolution neural network)
    - Represent visual input space, map to game actions
  - Experience replay
    - Batches updates of the Q-function, on a fixed set of observations

- No guarantee that this converges, or works very well.

- But often, it does.
DeepMind Atari -- Breakout
DeepMind Atari – Space Invaders
CNN, from screen to Joystick
The Recipe

• Connect game screen via CNN to a top layer, of reasonable dimension.
• Fully connected, to all possible user actions
• Learn optimal Q-function $Q^*$, maximizing future game rewards
• Batch experiences, and randomly sample a batch, with experience replay
• Iterate, until done.
Obvious Questions

• State: screen transitions, not just one frame
  – Four frames

• Actions: how to start?
  – Start with no action
  – Force machine to wiggle it

• Reward: what it is??
  – Game score

• Game AI will totally fail... in cases where these are not sufficient...
Peek-forward to results.

Space Invaders  |  Seaquest
But first... Reinforcement Learning in *One Slide*
Markov Decision Process

Fully observable universe
State space $S$, action space $A$
Transition probability function $f: S \times A \times S \rightarrow [0, 1.0]$
Reward function $r: S \times A \times S \rightarrow \text{Real}$
At a discrete time step $t$, given state $s$, controller takes action $a$:
- according to control policy $\pi: S \rightarrow A$ [which is probabilistic]
Integrate over the results, to learn the (average) expected reward.
Control Policy <-> Q-Function

• Every control policy \( \pi \) has corresponding \( Q \)-function
  
  \( Q: S \times A \rightarrow \text{Real} \)
  
  Which gives reward value, given state \( s \) and action \( a \), and assuming future actions will be taken with policy \( \pi \).

• Our goal is to learn an optimal policy
  
  This can be done by learning an optimal \( Q^* \) function
  
  Discount rate \( \gamma \) for each time-step \( t \)

\[
Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi],
\]

(maximum discount reward, over all control policies \( \pi \).)
Q-learning

• Start with any $Q$, typically all zeros.
• Perform various actions in various states, and observe the rewards.
• Iterate to the next step estimate of $Q^*$
  – $\alpha = \text{learning rate}$

\[
Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k [r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k)]
\]
Dammit, this is a bit complicated.
Let’s steal excellent slides from David Silver, University College London, and DeepMind
Observation, Action & Reward

At each step $t$ the agent:
- Executes action $A_t$
- Receives observation $O_t$
- Receives scalar reward $R_t$

The environment:
- Receives action $A_t$
- Emits observation $O_{t+1}$
- Emits scalar reward $R_{t+1}$

$t$ increments at env. step
Measurable Progress

Examples of Rewards

- Fly stunt manoeuvres in a helicopter
  - +ve reward for following desired trajectory
  - −ve reward for crashing
- Defeat the world champion at Backgammon
  - +/−ve reward for winning/losing a game
- Manage an investment portfolio
  - +ve reward for each $ in bank
- Control a power station
  - +ve reward for producing power
  - −ve reward for exceeding safety thresholds
- Make a humanoid robot walk
  - +ve reward for forward motion
  - −ve reward for falling over
- Play many different Atari games better than humans
  - +/−ve reward for increasing/decreasing score
(Long-term) Greed is Good?

**Rewards**

- A reward $R_t$ is a scalar feedback signal
- Indicates how well agent is doing at step $t$
- The agent’s job is to maximise cumulative reward

Reinforcement learning is based on the **reward hypothesis**

**Definition (Reward Hypothesis)**

*All* goals can be described by the maximisation of expected cumulative reward

Do you agree with this statement?
Markov State = Memory not Important

Information State

An information state (a.k.a. Markov state) contains all useful information from the history.

Definition

A state $S_t$ is Markov if and only if

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, …, S_t]$$

- “The future is independent of the past given the present”
  $$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:∞}$$
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future
- The environment state $S^e_t$ is Markov
- The history $H_t$ is Markov
Rodentus Sapiens: Need-to-Know Basis

Lecture 1: Introduction to Reinforcement Learning

- The RL Problem
- State

Rat Example

- What if agent state = last 3 items in sequence?
- What if agent state = counts for lights, bells and levers?
- What if agent state = complete sequence?
MDP: Policy & Value

• Setting up complex problem as Markov Decision Process (MDP) involves tradeoffs

• Once in MDP, there is an optimal policy for maximizing rewards

• And thus each environment state has a value
  – Follow optimal policy forward, to conclusion, or $\infty$

• Optimal policy <-> “true value” at each state
Chess Endgame Database

If value is known, easy to pursue optimal policy.
Policy: Simon Says

- A **policy** is the agent’s behaviour
- It is a map from state to action, e.g.
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$
Value: Simulate Future States, Sum Future Rewards

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

\[ v_\pi(s) = \mathbb{E}_\pi \left[ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right] \]

Familiar to stock market watchers: discounted future dividends.
Simple Maze

Lecture 1: Introduction to Reinforcement Learning
Inside An RL Agent

Maze Example

- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent’s location
Maze Policy

Maze Example: Policy

Arrows represent policy \( \pi(s) \) for each state \( s \)
Maze Example: Value Function

- Numbers represent value $v_\pi(s)$ of each state $s$
OK, we get it. Policy & value.
Back to Atari

**Atari Example: Reinforcement Learning**

- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores
How Game AI Normally Works

- Rules of the game are known
- Can query emulator
  - perfect model inside agent’s brain
- If I take action $a$ from state $s$:
  - what would the next state be?
  - what would the score be?
- Plan ahead to find optimal policy
  - e.g. tree search

Heuristic to evaluate game state; tricks to prune the tree.
These seem radically different approaches to playing games...
...but part of the Explore & Exploit Continuum

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is usually important to explore as well as exploit
RL is Trial & Error

- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way
E&E Present in (most) Games

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Restaurant Selection</td>
</tr>
<tr>
<td>Exploitation  Go to your favourite restaurant</td>
</tr>
<tr>
<td>Exploration  Try a new restaurant</td>
</tr>
<tr>
<td>■ Online Banner Advertisements</td>
</tr>
<tr>
<td>Exploitation  Show the most successful advert</td>
</tr>
<tr>
<td>Exploration  Show a different advert</td>
</tr>
<tr>
<td>■ Oil Drilling</td>
</tr>
<tr>
<td>Exploitation  Drill at the best known location</td>
</tr>
<tr>
<td>Exploration  Drill at a new location</td>
</tr>
<tr>
<td>■ Game Playing</td>
</tr>
<tr>
<td>Exploitation  Play the move you believe is best</td>
</tr>
<tr>
<td>Exploration  Play an experimental move</td>
</tr>
</tbody>
</table>
Back to Markov for a second...
Markov Reward Process (MRP)

A Markov reward process is a Markov chain with values.

**Definition**

A *Markov Reward Process* is a tuple \( \langle S, \mathcal{P}, \mathcal{R}, \gamma \rangle \)

- \( S \) is a finite set of states
- \( \mathcal{P} \) is a state transition probability matrix, \( \mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s] \)
- \( \mathcal{R} \) is a reward function, \( \mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s] \)
- \( \gamma \) is a discount factor, \( \gamma \in [0, 1] \)
MRP for a UK Student

Example: Student MRP
Discounted Total Return

Lecture 2: Markov Decision Processes

Markov Reward Processes

Return

Definition

The return $G_t$ is the total discounted reward from time-step $t$.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The discount $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward $R$ after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - $\gamma$ close to 0 leads to "myopic" evaluation
  - $\gamma$ close to 1 leads to "far-sighted" evaluation
Discounting the Future – We do it all the time.

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward
- It is sometimes possible to use undiscounted Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.
Example: State-Value Function for Student MRP (1)

$v(s)$ for $\gamma = 0$
Long Term View

Example: State-Value Function for Student MRP (2)
Back to $Q^*$
Q-Learning in One Slide

We now consider off-policy learning of action-values $Q(s, a)$

No importance sampling is required

Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$

But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$

And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left( R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$

Each step: we adjust $Q$ toward observations, at learning rate $\alpha$. 
Q-Learning Control: Simulate every Decision

Q-Learning Control Algorithm

\[ Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right) \]

Theorem

Q-learning control converges to the optimal action-value function, \( Q(s, a) \to q_*(s, a) \)
Q-Learning Algorithm

Q-Learning Algorithm for Off-Policy Control

Initialize $Q(s, a), \forall s \in S, a \in A(s)$, arbitrarily, and $Q(terminal-state, \cdot) = 0$

Repeat (for each episode):
  Initialize $S$

  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $A$, observe $R, S'$
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  until $S$ is terminal

Or learn on-policy, by choosing states non-randomly.
Think Back to Atari Videos

• By default, the system takes default action (no action).

• Unless rewards are observed (a few steps) from actions, the system moves (toward solution) very slowly.
Back to the CNN...
CNN, from screen ($S$) to Joystick ($A$)
Four Frames $\rightarrow$ 256 hidden units

DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

Network architecture and hyperparameters fixed across all games
Experience Replay

• Simply, batch training.

• Feed in a bunch of transitions, compute new approximating of $Q^*$, assuming current policy

• Don’t adjust $Q$, after every data point.

• Pre-compute some changes for a bunch of states, then pull a random batch from the database.
Experience Replay (Batch train): DQN

DQN uses experience replay and fixed Q-targets

- Take action $a_t$ according to $\epsilon$-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions $(s, a, r, s')$ from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters $w^-$
- Optimise MSE between Q-network and Q-learning targets

$$L_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i} \left[ \left( r + \gamma \max_{a'} Q(s', a'; w^-_i) - Q(s, a; w_i) \right)^2 \right]$$

- Using variant of stochastic gradient descent
Experience Reply with SGD

Stochastic Gradient Descent with Experience Replay

Given experience consisting of \(\langle \text{state}, \text{value} \rangle\) pairs

\[ \mathcal{D} = \{ \langle s_1, v_1^{\pi} \rangle, \langle s_2, v_2^{\pi} \rangle, \ldots, \langle s_T, v_T^{\pi} \rangle \} \]

Repeat:

1. Sample state, value from experience

\[ \langle s, v^{\pi} \rangle \sim \mathcal{D} \]

2. Apply stochastic gradient descent update

\[ \Delta w = \alpha (v^{\pi} - \hat{v}(s, w)) \nabla_w \hat{v}(s, w) \]
Do these methods help?

How much does DQN help?

<table>
<thead>
<tr>
<th>Game</th>
<th>Replay Fixed-Q</th>
<th>Replay Q-learning</th>
<th>No replay Fixed-Q</th>
<th>No replay Q-learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakout</td>
<td>316.81</td>
<td>240.73</td>
<td>10.16</td>
<td>3.17</td>
</tr>
<tr>
<td>Enduro</td>
<td>1006.3</td>
<td>831.25</td>
<td>141.89</td>
<td>29.1</td>
</tr>
<tr>
<td>River Raid</td>
<td>7446.62</td>
<td>4102.81</td>
<td>2867.66</td>
<td>1453.02</td>
</tr>
<tr>
<td>Seaquest</td>
<td>2894.4</td>
<td>822.55</td>
<td>1003</td>
<td>275.81</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>1088.94</td>
<td>826.33</td>
<td>373.22</td>
<td>301.99</td>
</tr>
</tbody>
</table>

Yes. Quite a bit.

Units: game high score.
Finally... results... it works! (sometimes)

- **Space Invaders**
- **Seaquest**
Some Games Better Than Others

• Good at:
  – quick-moving, complex, short-horizon games
  – Semi-independent trails within the game
  – Negative feedback on failure
  – Pinball

• Bad at:
  – long-horizon games that don’t converge
  – Ms. Pac-Man
  – Any “walking around” game
Montezuma: Drawing Dead

Can you see why?
Can DeepMind learn from chutes & ladders?

How about Parcheesi?
Actions & Values

- Value is in expected (discount) score from state
- Breakout: value increases as closer to medium-term reward
- Pong: action values differentiate as closer to ruin
Frames, Batch Sizes Matter

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minibatch size</td>
<td>32</td>
<td>Number of training cases over which each stochastic gradient descent (SGD) update is computed.</td>
</tr>
<tr>
<td>replay memory size</td>
<td>1000000</td>
<td>SGD updates are sampled from this number of most recent frames.</td>
</tr>
<tr>
<td>agent history length</td>
<td>4</td>
<td>The number of most recent frames experienced by the agent that are given as input to the Q network.</td>
</tr>
<tr>
<td>target network update frequency</td>
<td>10000</td>
<td>The frequency (measured in the number of parameter updates) with which the target network is updated (this corresponds to the parameter $C$ from Algorithm 1).</td>
</tr>
<tr>
<td>discount factor</td>
<td>0.99</td>
<td>Discount factor gamma used in the Q-learning update.</td>
</tr>
<tr>
<td>action repeat</td>
<td>4</td>
<td>Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.</td>
</tr>
<tr>
<td>update frequency</td>
<td>4</td>
<td>The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.</td>
</tr>
<tr>
<td>learning rate</td>
<td>0.00025</td>
<td>The learning rate used by RMSProp.</td>
</tr>
<tr>
<td>gradient momentum</td>
<td>0.95</td>
<td>Gradient momentum used by RMSProp.</td>
</tr>
<tr>
<td>squared gradient momentum</td>
<td>0.95</td>
<td>Squared gradient (denominator) momentum used by RMSProp.</td>
</tr>
<tr>
<td>min squared gradient</td>
<td>0.01</td>
<td>Constant added to the squared gradient in the denominator of the RMSProp update.</td>
</tr>
<tr>
<td>initial exploration</td>
<td>1</td>
<td>Initial value of $\epsilon$ in $\epsilon$-greedy exploration.</td>
</tr>
<tr>
<td>final exploration</td>
<td>0.1</td>
<td>Final value of $\epsilon$ in $\epsilon$-greedy exploration.</td>
</tr>
<tr>
<td>final exploration frame</td>
<td>1000000</td>
<td>The number of frames over which the initial value of $\epsilon$ is linearly annealed to its final value.</td>
</tr>
<tr>
<td>replay start size</td>
<td>50000</td>
<td>A uniform random policy is run for this number of frames before learning starts and the resulting experience is used to populate the replay memory.</td>
</tr>
<tr>
<td>no-op max</td>
<td>30</td>
<td>Maximum number of “do nothing” actions to be performed by the agent at the start of an episode.</td>
</tr>
</tbody>
</table>
Bibliography

• Demis Hassabis interview: https://medium.com/backchannel/the-deep-mind-of-demis-hassabis-156112890d8a
• Wonderful Reinforcement Learning Class (David Silver, University College London): http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
• Readable (kind of) paper on Replay Memory: http://busoniu.net/files/papers/smcc11.pdf
• Chute & Ladders: an ancient morality tale: http://uncyclopedia.wikia.com/wiki/Chutes_and_Ladders
• ALE (Arcade Learning Environment): http://www.arcadelearningenvironment.org/
• Stella (multi-platform Atari 2600 emulator): http://stella.sourceforge.net/faq.php
• Deep Q-RL with Theano: https://github.com/spragunr/deep_q_rl
Addendum: Atari Setup w/ Stella
Addendum: ALE Atari Agent

Game will be controlled by an internal agent.

- Episode 1 ended, score: 120
- Episode 2 ended, score: 55
- Episode 3 ended, score: 225
- Episode 4 ended, score: 270
- Episode 5 ended, score: 155
- Episode 6 ended, score: 240
- Episode 7 ended, score: 315
- Episode 8 ended, score: 65
- Episode 9 ended, score: 155
- Episode 10 ended, score: 250

compiled agent | I/O pipes | saves frames
Addendum: (Video) Poker?

• Can input be fully connected to actions?

• Atari games played one button at a time.

• Here, we choose which cards to keep.

• Remember Montezuma’s Revenge!
Addendum: Poker Transition

How does one encode this for RL?

OpenCV easy for image generation.