Reinforcement Learning: Game Benchmarks

Data Science Indy April 2019

Reinforcement Learning

- Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning is considered one of three machine learning paradigms, along with supervised learning and unsupervised learning
- Focus is finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge)
- RL does not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible

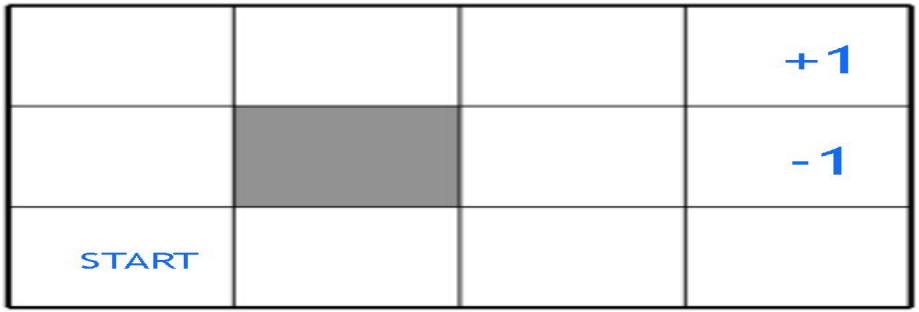
Reinforcement Learning

- Environment is typically formulated as a Markov Decision Process (MDP) with:

- a set of environment and agent states, S;
- a set of actions, A, of the agent;
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability of transition from state s to state s' under action a.
- $R_a(s, s')$ is the immediate reward after transition from s to s' with action a.
- rules that describe what the agent observes

Classic RL problems

Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

- Pole-balancing
- TD-Gammon
- Helicopter
- Resource allocation in datacenters
- ...

End-to-end RL and Deep RL

- End-to-end process extends RL from learning only for actions to learning the entire process from sensors to motors including higher-level functions that are difficult to develop independently from other functions
- Approach originated in TD-Gammon (1992) by Gerald Tesauro at IBM's Thomas J. Watson Research Center
- Work extended to game of Go (AlphaGo) using a deep convolutional neural network (or more specifically a Deep-Q network) that showed superior results in image recognition and that competed well on 49 different Atari games

End-to-end RL and Deep RL

- End-to-end process extends RL from learning only for actions to learning the entire process from sensors to motors including higher-level functions that are difficult to develop independently from other functions
- Approach originated in TD-Gammon (1992) by Gerald Tesauro at IBM's Thomas J. Watson Research Center
- Work extended to game of Go (AlphaGo) using a deep convolutional neural network (or more specifically a Deep-Q network) that showed superior results in image recognition and that competed well on 49 different Atari games

What is pyGame?

A set of Python modules to make it easier to write games.

home page: http://pygame.org/

documentation: http://pygame.org/docs/ref/

pyGame helps you do the following and more:

- Sophisticated 2-D graphics drawing functions
- Deal with media (images, sound F/X, music) nicely
- Respond to user input (keyboard, joystick, mouse)
- Built-in classes to represent common game objects

pyGame

- Python module developed for writing games in python
- Cross platform
- Written on top of the Simple DirectMedia Layer, wrappers around OS functions
- Windows, Unix, Linux, Android, iOS
- Can be used for multimedia development
- Distributed with python

pyGame

pyGame consists of many **modules** of code to help you:

```
cdrom     cursors     display     draw     event
font     image     joystick     key     mouse
movie     sndarray     surfarray     time
   transform
```

sprites: Onscreen characters or other moving objects.

collision detection: Seeing which pairs of sprites touch.

event: An in-game action such as a mouse or key press.

event loop: Many games have an overall loop that:

waits for events to occur, **updates** sprites, **redraws** screen

Arcade Learning Environment

ALE provides an interface to hundreds of Atari 2600 game environments

ALE presents significant research challenges for reinforcement learning, model learning, model-based planning, imitation learning, transfer learning, and intrinsic motivation

ALE is released as free, open-source software under the terms of the GNU General Public License

Arcade Learning Environment

ALE is built on top of Stella, an open-source Atari 2600 emulator.

It allows the user to interface with the Atari 2600 by receiving joystick motions, sending screen and/or RAM information,and emulating the platform.

ALE also provides a game-handling layer which transforms each game into a standard reinforcement learning problem by identifying the accumulated score and whether the game has ended

Arcade Learning Environment

ALE has been receiving increasing attention from the scientific community, leading to some high-prole success stories such as the much publicized Deep Q-Networks (DQN)

Motivations for ALE in RL problems:

1) they are varied enough to provide multiple different tasks, requiring general competence

2) they are interesting and challenging for humans,

3) they are free of experimenter's bias, having been developed by an independent party.

Playing Atari with Deep Reinforcement Learning

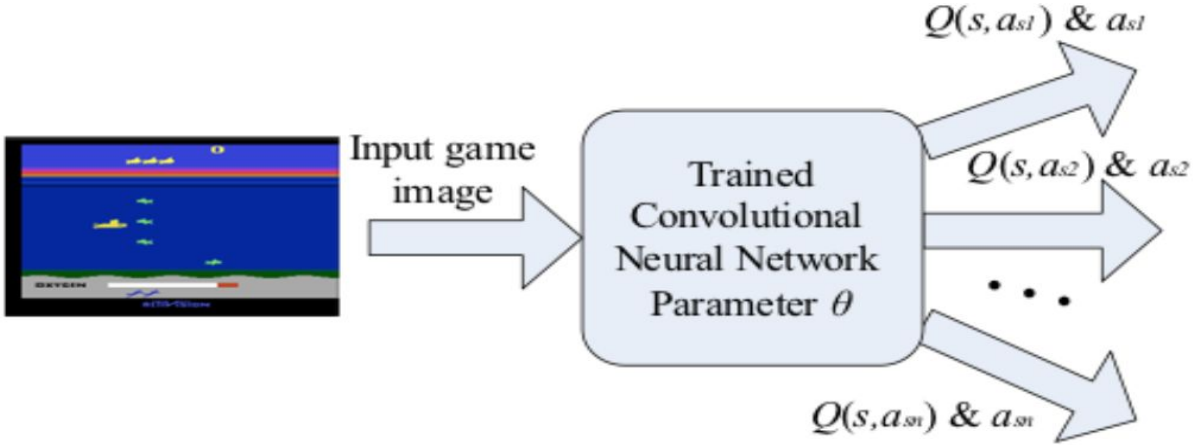
- The Atari 2600 is a video game console released in September 1977 by Atari, Inc.
- Atari emulator: Arcade Learning Environment (ALE)

Process

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 ϵ 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 $(y_j - Q(\phi_j, a_j; \theta))^2$
    end for
end for

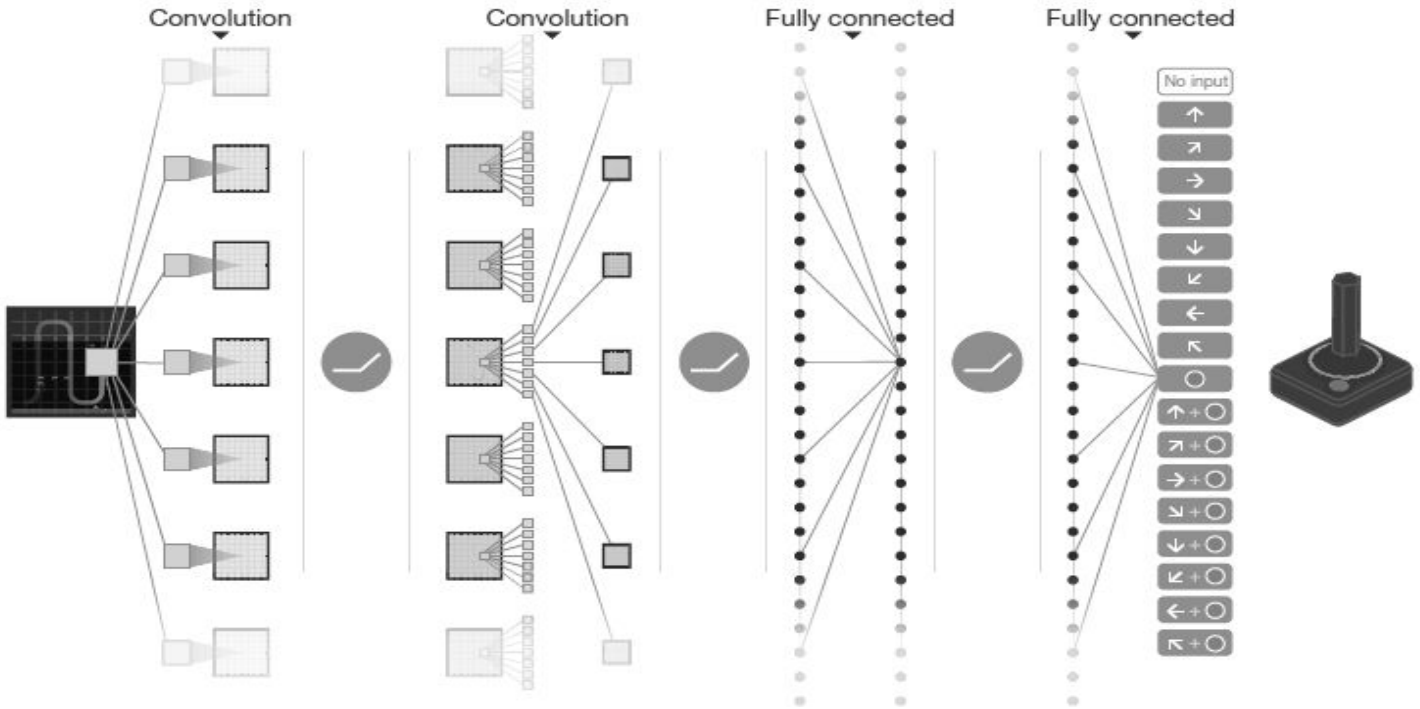
- Input is image state, outputs are legal actions and corresponding Q(s,a) values

Visualization of Process



Play the game

Alternative illustration of process



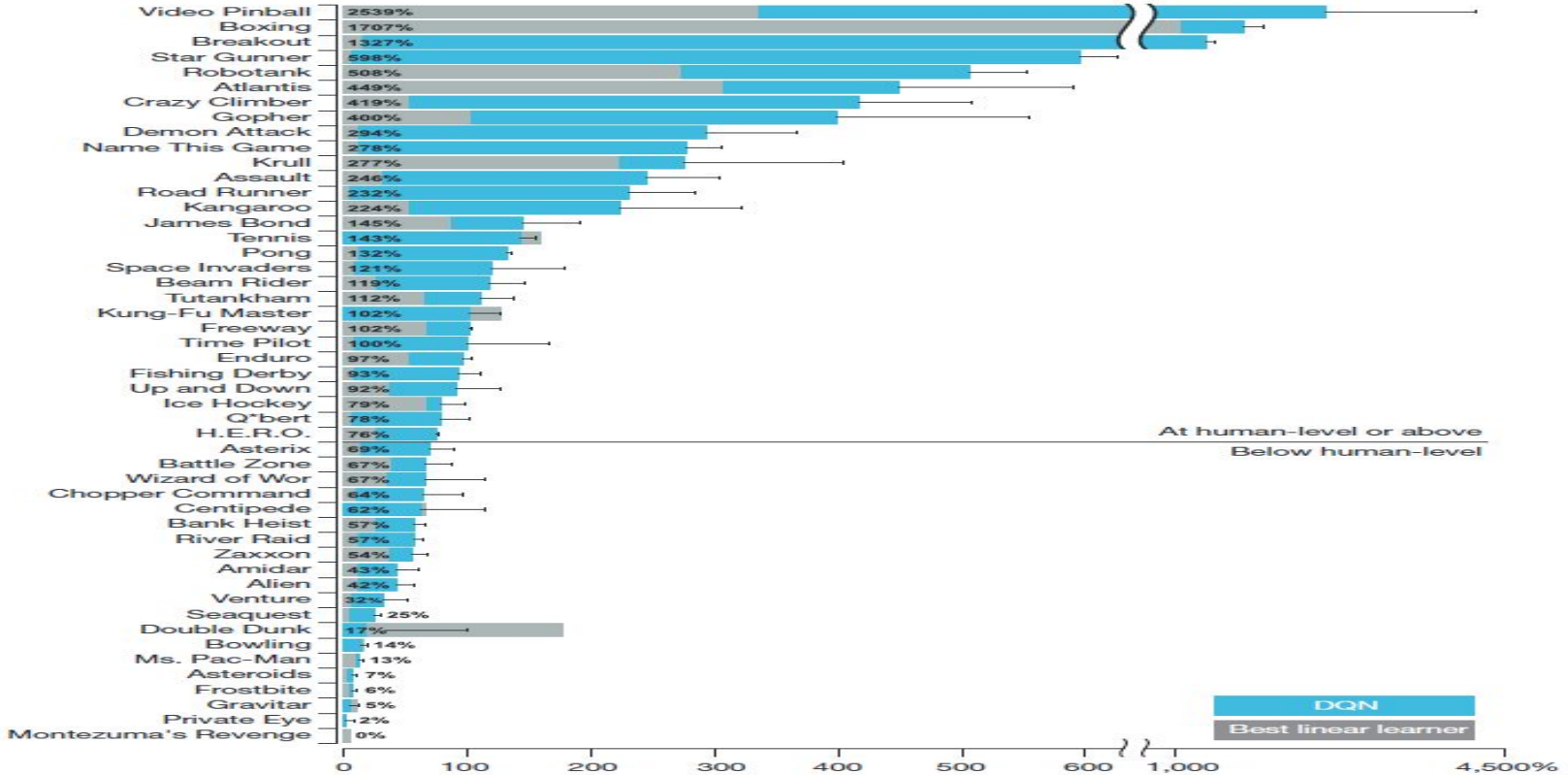
Results

Screen shots from five Atari 2600 games: (Left-to-right) Beam Rider, Breakout, Pong, Seaquest, Space Invaders

Comparison of average total reward for various learning methods by running an ε-greedy policy with $\varepsilon = 0.05$ for a fixed number of steps

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa	996	5.2	129	−19	614	665	271
Contingency	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690

Alternative Results



What do hyperparameters look like?

Hyperparameter	Value	Description
minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed.
replay memory size	1000000	SGD updates are sampled from this number of most recent frames.
agent history length	4	The number of most recent frames experienced by the agent that are given as input to the Q network.
target network update frequency	10000	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).
discount factor	0.99	Discount factor gamma used in the Q-learning update.
action repeat	4	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.
update frequency	4	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.
learning rate	0.00025	The learning rate used by RMSProp.
gradient momentum	0.95	Gradient momentum used by RMSProp.
squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
min squared gradient	0.01	Constant added to the squared gradient in the denominator of the RMSProp update.
initial exploration	1	Initial value of ε in ε-greedy exploration.
final exploration	0.1	Final value of ε in ε-greedy exploration.
final exploration frame	1000000	The number of frames over which the initial value of ε is linearly annealed to its final value.
replay start size	50000	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.
no-op max	30	Maximum number of "do nothing" actions to be performed by the agent at the start of an episode.

Techniques for RL training

- Brute force/naive
- Apprenticeship (pretrain from human player)
- Transfer learning (pretrain from another model)
- Play against another programmed CPU
- Self-play (multi-player games, high performance in: chess, checkers, backgammon, othello, Scrabble, Poker, Go, Starcraft)

Current open areas

- Representation learning
- Exploration
- Transfer learning
- Model learning
- Off-policy learning

Links

https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

https://www.nature.com/articles/nature14236

https://www.nature.com/articles/nature16961

https://arxiv.org/abs/1207.4708

https://deepmind.com/documents/119/agz_unformatted_nature.pdf

https://arxiv.org/abs/1710.02298

https://github.com/google/dopamine

https://www.pygame.org/news

https://github.com/mgbellemare/Arcade-Learning-Environment