Take-home message

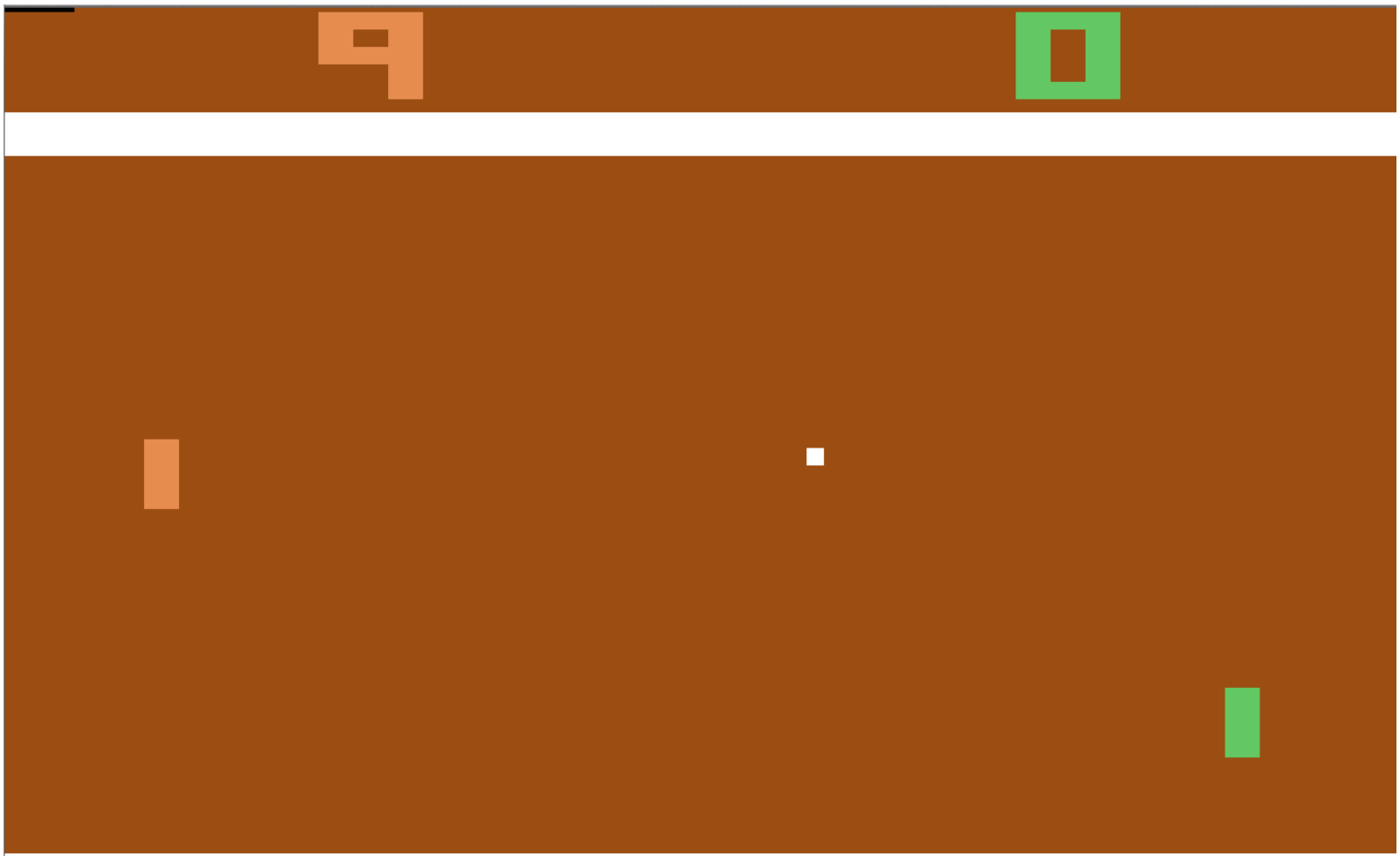
1. Always use a good quality tea
2. Always use freshly drawn water (stale water is stale tea)
3. Remember to warm the tea pot
4. Measure the right amount of tea for the amount of water in the pot
5. Water must reach boiling point
6. Let the tea brew for 3–5 minutes before serving

Let every cup you make be a cup that cheers!

Tea talk #XI

February 13, 2014

Playing Atari 2600 games

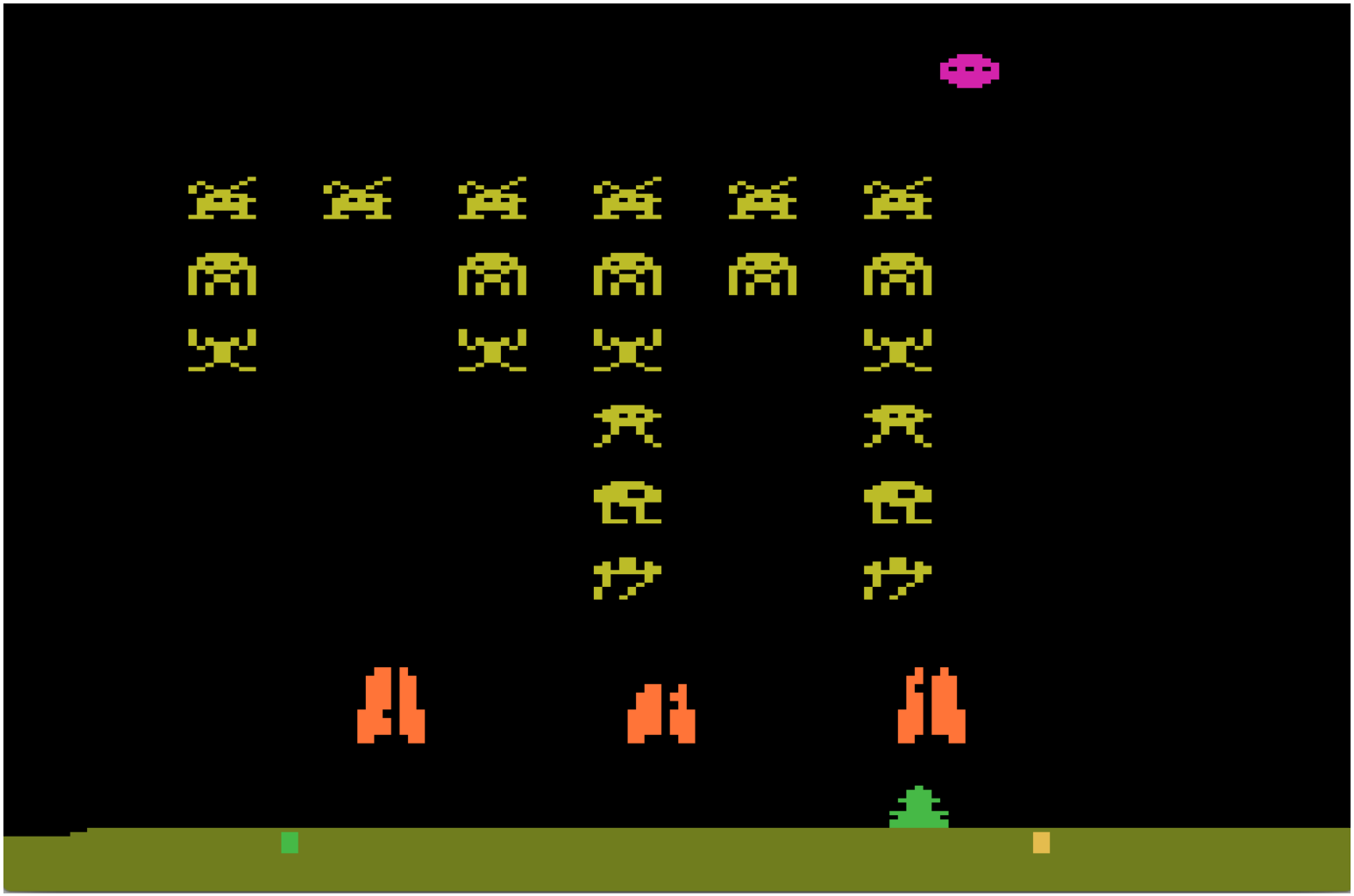


Pong

Playing Atari 2600 games

Breakout

Playing Atari 2600 games



Space Invaders

Playing Atari 2600 games

Seaquest

Playing Atari 2600 games

Beam Rider

Playing atari games with deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller

Deep Learning workshop @NIPS, 2013

Playing atari games with deep RL

- first deep learning model to successfully learn control policies directly from high-D sensory input using RL
- input: raw pixels,
  output: value function estimating future reward
- same architecture and learning algorithm for every game
- outperform all previous approaches on 6 (out of 7) games, surpasses a human expert on 3 of them.

Methods

- convolutional network that takes states as input and produces values of actions $Q(s, a)$ as an output;
- states s_t: a stack of 4 raw screen frames (84x84, at 3-bit);
- actions a_t: move right, left, shoot, etc. (4 to 18)
- rewards r_t: changes in game score clipped to (-1, 1)

- ϵ-greedy stochastic gradient Q-learning
- experience replay (up to 10^6 frames)

Learning

Discounted return:

$$R_t = r_t + \sum_{\tau=1}^{T-t} \gamma^\tau r_{t+\tau}$$

Optimal value

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

Bellman equation

$$Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a') | s, a]_{s'}$$

Q-learning

$$Q_{i+1}(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s', a') | s, a]_{s'}$$

Learning

Q-learning

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s',a')|s,a]_{s'}$$

Function approximator

$$Q^*(s,a) \sim Q(s,a;\theta)$$

Loss function

$$L_i = \mathbb{E}[(y_i - Q(s,a;\theta_i))^2|s]_a$$

Gradient descent

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}[(r + \gamma \max_{a'} Q_i(s',a') - Q(s,a;\theta_i))\nabla\theta_i Q(s,a;\theta_i)]_{a,s'}$$

Deep Learning

Gradient descent

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}[(r + \gamma \max_{a'} Q_i(s', a') - Q(s, a; \theta_i))\nabla\theta_i Q(s, a; \theta_i)]_{a,s'}$$

- Replace expectations by samples: 'stochastic gradient descent'
- Resample samples: 'experience replay'

$$e_t = (s_t, a_t, r_t, a_{t+1})$$

- ϵ-greedy: allow for some randomness
  (annealed from 100% to 1% over the first million frames)
- Frame skipping (4, or 3)

Deep Architecture

1. convolves 16 8x8 filters with stride 4 with the input image
   + a rectifier nonlinearity
2. convolves 32 4x4 filters with stride 2
   + a rectifier nonlinearity.
3. fully-connected layer of 256 rectifier units
4. fully-connected linear layer with a single output for each valid action

Methods

Algorithm 1 Deep Q-learning with Experience Replay

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Results

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.