Human-level control through deep reinforcement learning

Jake Hoare, UNSW
May 2016

Objective

- Create a single agent that can play a range of different Atari games better than a human
- Inputs are screen pixels and score
- Actions are valid moves in the game
- General AI

Challenges

- Natural solution – reinforcement learning
- High dimensionality of input – 210 x 160 screen with 128 colours at 60 Hz
- No domain-specific information or heuristics
- Delayed rewards
- Instability of reinforcement learning with non-linear approximator

Q-learning

- Learn the policy (what action to take in a given state) that maximises the sum of future discounted rewards

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

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

Dimensionality

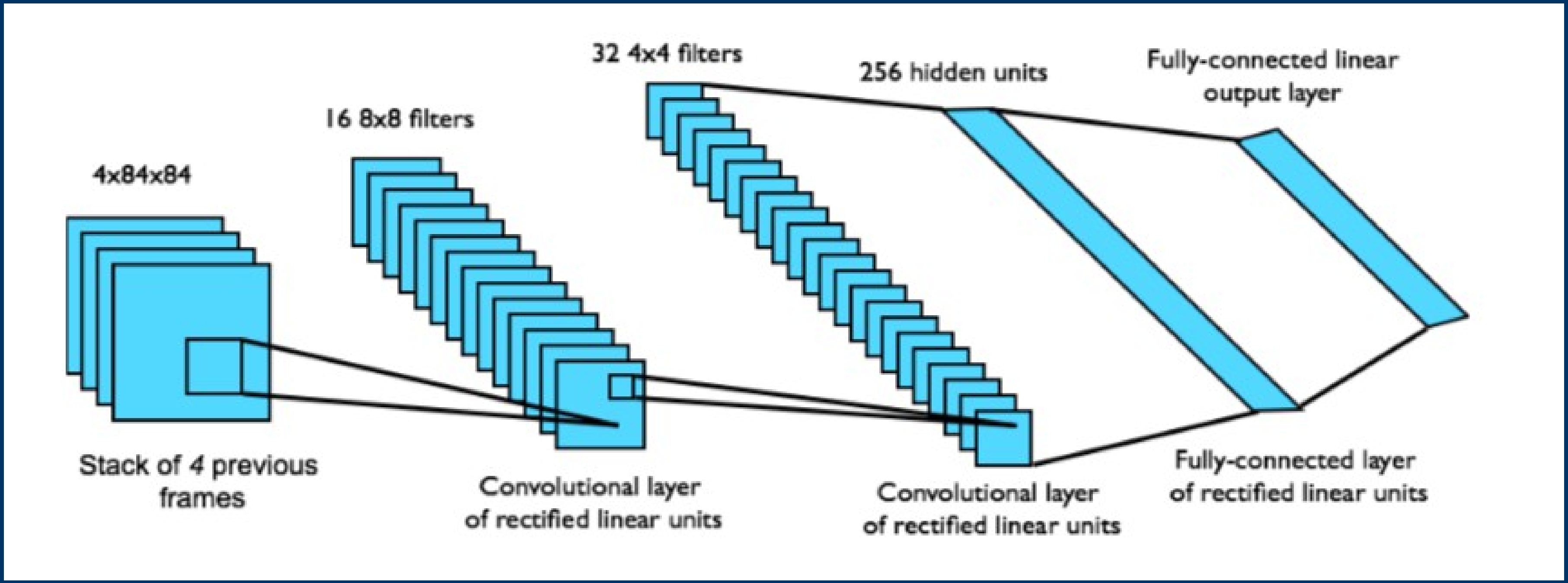
- State becomes sequence of 4 frames
- $10^{283,000}$ states
- Need to generalise to reduce dimensions
- Instead of learning discrete Q(s,a), approximate with convolutional neural network

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

Initial data compression

- Colour 210 x 160 to 256-grayscale 84 x 84
- Max(frame, previous frame)

Network architecture



- Learn a separate Q for each action – find the best action by choosing the largest

Experience replay

- Instability of network learning caused by temporal correlations in sequence of observations
- Network trained on random minibatches of previous observations
- Efficient reuse of experiences

Freeze target

- Training example is a tuple (s,a,r,s')
- Error function uses target value also based on network output
- Poor convergence if output and target move together so adjust target network only periodically

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

rmsProp

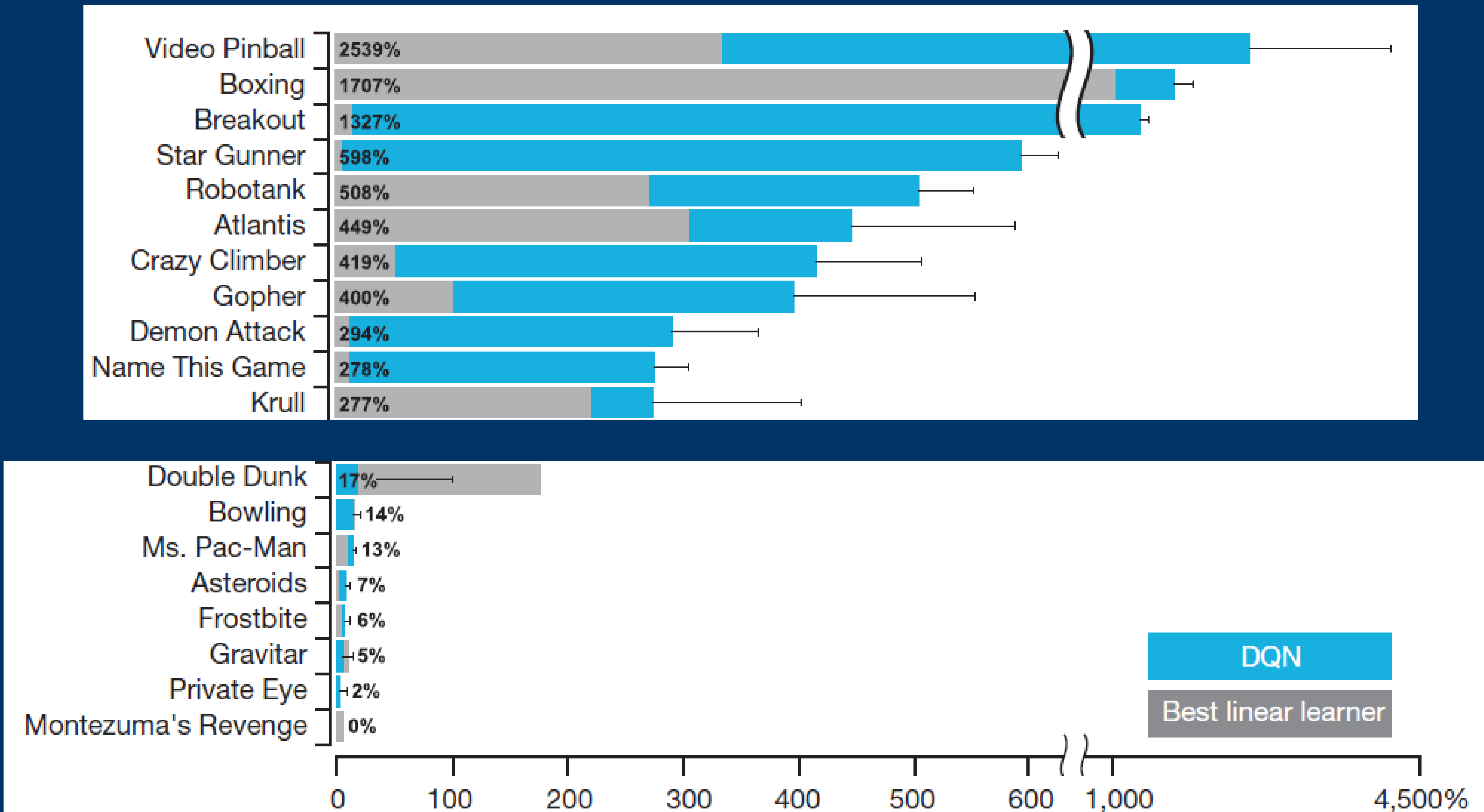
- Problem of same gradient descent learning rate for all weights
- rprop – for full batch learning, increase the step size multiplicitavely if gradients agree
- rmsProp – for minibatch learning, normailise update by maintaining a decaying mean square average of each gradient

Other tricks

- Reward clipping – normalise to +/-1, loses ability to differentiate between big and small gains
- Skipping frames – fast-forward by selecting actions every 4th frame
- Exploration vs exploitation – epsilon greedy policy to explore initially but then take best actions

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Results



Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
2539%
1707%
1327%
598%
508%
449%
419%
400%
294%
278%
277%
Double Dunk
Bowling
Ms. Pac-Man
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge
17%
14%
13%
7%
6%
5%
2%
0%
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

Strengths and weaknesses

- Space Invaders https://youtu.be/Dds_yDJFhvI
- Breakout https://youtu.be/cjpEIotvwFY
- Montezuma https://www.youtube.com/watch?v=Klxxg9JM5tY

Double Q learning

- Improve upon target freezing by unpacking the target expression

- Network used twice – once to determine the best next action and once to find the Q value of the next state

$$Y_t^{\mathrm{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

Prioritsed experience replay

- Improve upon experience replay by replaying large error examples more often
- Can overfit and may not replay if error was initially low – add randomness