Playing Atari with Deep Reinforcement Learning

RL Challenges

- Reward signal is often sparse, noisy and delayed
- Huge amount of possible states
- Data distribution changes as the algorithm learns new behaviors

RL Challenges

- Reward signal is often sparse, noisy and delayed

  Q-Learning (Temporal Difference)

- Huge amount of possible states

- Data distribution changes as the algorithm learns new behaviors

RL Challenges

- Reward signal is often sparse, noisy and delayed

  Q-Learning (Temporal Difference)

- Huge amount of possible states

  Use CNN to learn the underlying distribution

- Data distribution changes as the algorithm learns new behaviors

RL Challenges

- Reward signal is often sparse, noisy and delayed

  Q-Learning (Temporal Difference)

- Huge amount of possible states

  Use CNN to learn the underlying distribution

- Data distribution changes as the algorithm learns new behaviors

  Experience Replay

Modelling the Network

Loss function:

$$L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q\left(s,a;\theta_i\right)\right)^2\right]$$

Target value at iteration i:

$$y_i \;=\; \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a\right]$$

Optimize the loss function by stochastic gradient descent

Experience Replay

- Instead of updating weights using only the current iteration, we sample a random mini-batch of previous iterations. These are stored in memory data-set D.

- After performing gradient descent on this mini-batch, we execute an action according to ε-greedy policy. I. e., choose the greedy strategy $a = \max_a Q(s,a;\theta)$, with probability 1-ε and choose a random action with probability ε

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 $\phi_1 = \phi(s_1)$ e.g. Append last 5 iterations to 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 $\mathcal{D}$

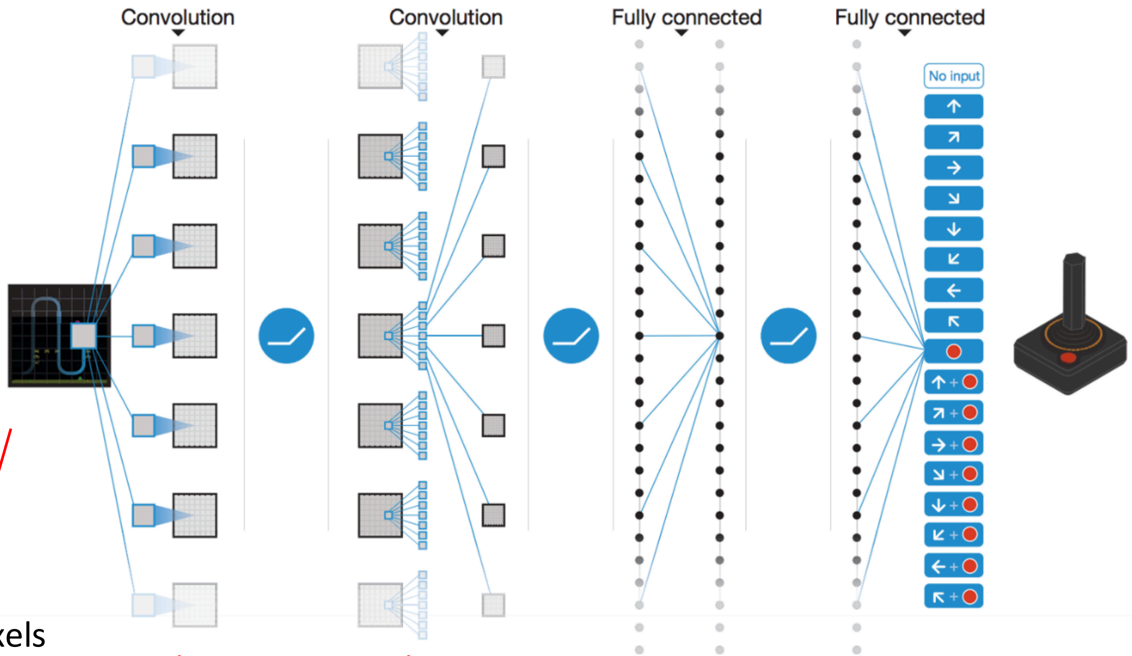
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

  end for

end for

Network architecture



84x84x4 pixels

16 8x8 filters with stride 4

32 4x4 filters with stride 2

256 fully - connected rectifier units

An output for every valid action

References

- [2013] "Playing Atari with Deep Reinforcement Learning”