Human-level Control Through Deep Reinforcement Learning

Google DeepMind: Mnih et al. 2015

CSC2541
Nov. 4th, 2016

Intro

- Policy π maps states (observation) to actions: $\pi(s) = a$
- Action-Value Function Q gives expected total reward from a state and action from some policy

$$Q^{\pi}(s, a) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... | s_t = s, a_t = a] \quad (1)$$

- Optimal Action-Value Function Q^* gives best value possible from any policy

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... | s_t = s, a_t = a, \pi] \quad (2)$$

$$= \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s', a') | s, a] \quad (3)$$

Deep Q Networks (DQN)

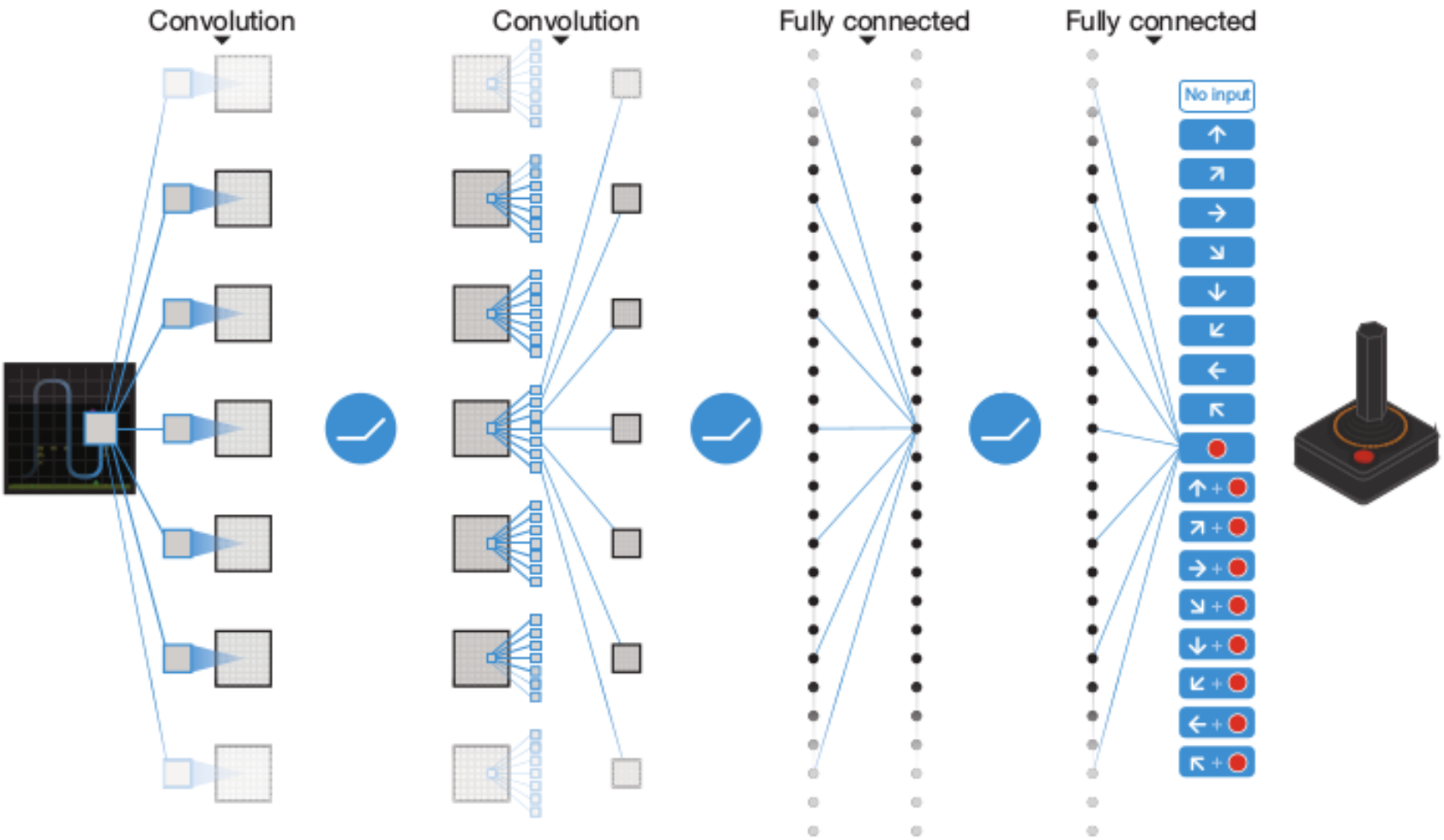
- Idea: want to replicate successes of Supervised Learning in Reinforcement Learning.
- Q^* is a function, so we can approximate with a Deep Network.
- Loss Function for Q-learning updates (MSE)

$$\mathcal{L}(w) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)^2\right]$$

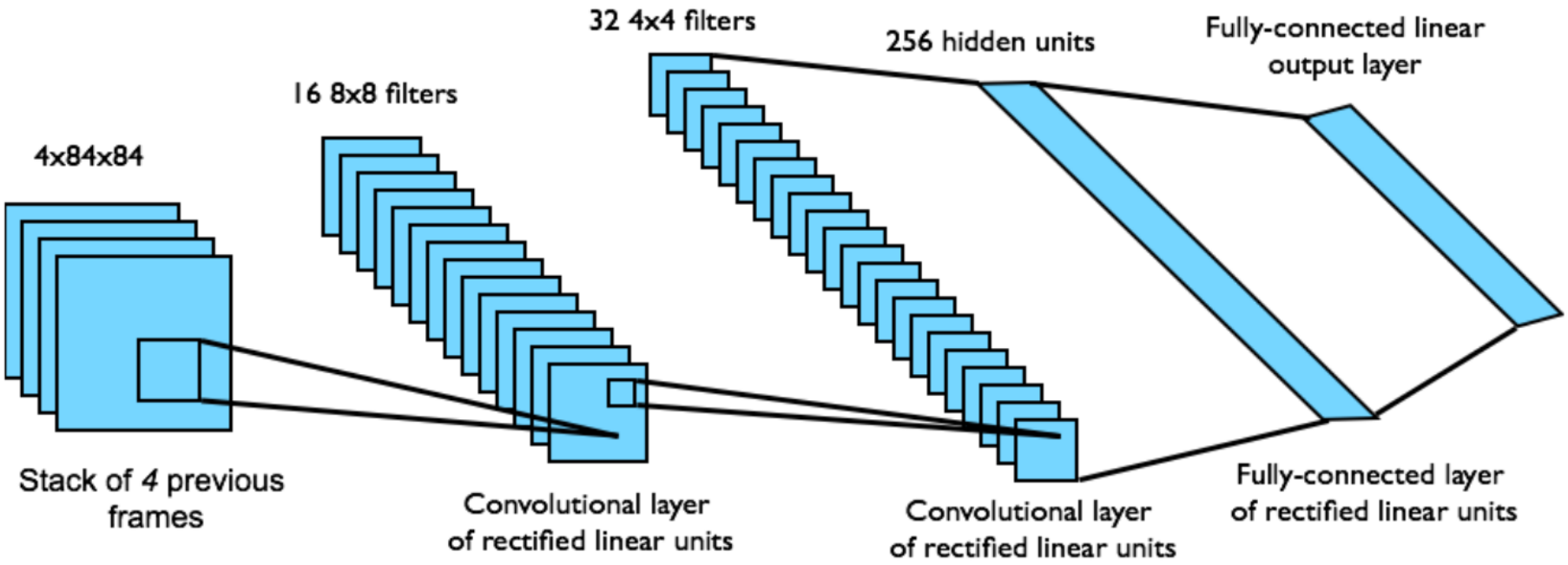
- Loss is difference between target value (fixed) and current estimate of Q.
- Gradient of Loss

$$\nabla_w \mathcal{L}(w) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \nabla_w Q(s, a, w)\right]$$

Architecture of DQN



Architecture of DQN cont.



Architecture of DQN cont..

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.

Issues

1. Observations are sequential and correlated (non iid).
   - How about destroying the temporal structure?
2. Data distribution depends on policy (action), which may change drastically with small changes in Q.
   - Good policy at some situations will be irrelevant at other situations.
3. Gradients are sensitive to scale of Rewards
   - Gradient Clipping, restrict reward e.g. $r \in [-1, 1]$, batch normalization

Experience Replay [Lin 1993]

- Helps with the first two issues (correlated observations, non-stationary data distribution).
- First, choose action from ϵ-greedy policy, then store (s_t, a_t, r_t, s_{t+1}) to memory $\mathcal{D}$.
- Sample a mini-batch of $(s, a, r, s') \sim \mathcal{D}$ and use that to optimize loss.

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

- By doing batch learning, we gained some protection from correlated observations and non-stationary data distribution.
- However, we have lost the temporal structure of the data.
- Instead of looking at individual experience samples, we could replay sequences of experiences (lessons).

Fixed Target Q Network

- Often, whenever we update $Q(s_t, a_t)$ by increasing it, $Q(s_{t+1}, a_t)$ is increased for all actions.
- This means our target $r + \gamma \max_a Q(s, a, w)$ is also increased.
- Our updates to Q and our targets are correlated.
- To fix this correlation, we add a delay between updates to Q and computation of targets.
- This is done by computing targets using older sets of parameters, not the most recent.

Q-Learning vs DQN

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Linear vs. Nonlinear Function Approximator

Effect of replacing a Deep Network with a shallow network with one linear hidden layer.
Note: Previous work using linear function approximation will sometimes perform better than shallow network e.g. 129.1 in Enduro.

Game	DQN	Linear
Breakout	316.8	3.00
Enduro	1006.3	62.0
River Raid	7446.6	2346.9
Seaquest	2894.4	656.9
Space Invaders	1088.9	301.3

The learning algorithm

Algorithm 1: deep Q-learning with experience replay.

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

References

1. Mnih, V. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015)
2. Lin, L.-J. Reinforcement learning for robots using neural networks. Technical Report, DTIC Document (1993)