Playing Atari with Deep Reinforcement Learning

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

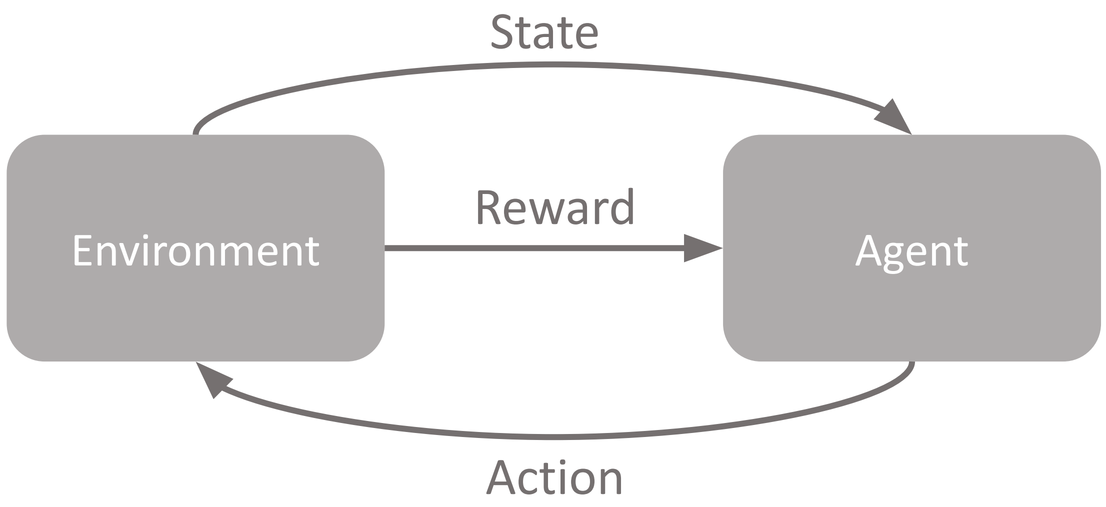
NIPS Deep Learning Workshop 2013

Yeonji Lim

2020.7.16

Basic Knowledge
Reinforcement Learning

- Like the way people learn by themselves by interacting with the environment.
- Applies to the problem of making decisions sequentially
- Goal : Finding optimal policy (maximize sum of rewards)



Basic Knowledge
MDP & Action Value Function

- Markov Decision Processes : Mathematical definition of information about the environment
  - State $\boldsymbol{S}$, Action $\boldsymbol{A}$: finite set of possible states or actions
  - Reward function $\boldsymbol{R}$: $R_s^a = E[R_{t+1}|S_t = s, A_t = a]$, expected value of R_{t+1} which is result of state and action in time t
  - State transition probability $\boldsymbol{P}$: $P_{ss'}^a = P[S_{t+1} = s'|S_t = s, A_t = a]$ probability to go s to s' by action a
  - Discount factor $\boldsymbol{\gamma}$: make future reward less valuable
- Action value function $Q(s,a)$: which action is more valuable

$$Q(s,a) = E[\underbrace{R_{t+1} + \gamma R_{t+2} + \ldots}_{\text{Sum of future rewards}}|S_t = s, A_t = a] = E[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

Basic Knowledge
Q-Learning

- Algorithm which learn action value function Q
- Model-free : don't need model information(State transition probability, Reward function)
- Update Q

$$Q(s_t, a_t) \leftarrow R_{t+1} + \gamma Q(s_{t+1}, a')$$

- Off-policy : Use two policy
  - ε-greedy to choose action a_t to do
  - greedy to choose action a' used as update goal

$$Q(s_t, a_t) \leftarrow R_{t+1} + \max_a \gamma Q(s_{t+1}, a)$$

Deep Reinforcement Learning
DQN & Policy Gradient

Deep Q Network

Policy Gradient

Play Atari with Deep Reinforcement Learning

DQN
Challenges, Goal, Introduction

- What we want : Learning to control agents directly from high-dimensional sensory inputs like vision and speech
  - Deep Learning can make it possible! -> DL could also be beneficial for RL with sensory data?
- Challenges

	Deep Learning	Reinforcement Learning
Training data	Labeled training data	Reward Signal (sparse, noisy, delayed)
Data dependency	Independent	Dependent
About data distribution	Assume a fixed underlying distribution	The data distribution changes as the algorithm learns new behaviors

DQN
Can solve these!

- CNN can overcome these challenges to learn successful control policies from raw video data in complex RL environments
- Model : CNN(Convolutional Neural Network)
- Trained with variant of Q-learning
  - Input : raw pixels
  - Output : value of function estimating future rewards
- Stochastic Gradient Descent
- Experience Replay Memory
- Apply method to many games with no adjustment of the architecture or learning algorithm

DQN
Q-Network

$$R_t = \sum_{t'=t}^{T} r^{t'-t} r_{t'}$$

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

Policy function

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

- Iteratively update Bellman equation to estimate Q. So Value Iteration algorithm iteratively do this procedure

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma Q_i(s',a') \,\middle|\, s,a\right] \qquad \boxed{Q_i \rightarrow Q^* \text{ as } i \rightarrow \infty}$$

- But it is impractical!

  ->action-value function is estimated separately for each sequence, without any generalization.

DQN
Q-Network

- Q-network : Q-learning + neural network function approximator with weights θ

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

- Loss function

Target value for iteration i

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim \boxed{\rho(\cdot)}}\left[\left(\boxed{y_i} - Q(s, a; \theta_i)\right)^2\right], \text{where, } y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \,\middle|\, s, a\right]$$

Behavior distribution

$\rho(s, a)$: probability distribution over sequences s and actions a

- The parameters from the previous iteration θ_{i-1} are held fixed when optimizing the loss function $L_i(\theta_i)$
- Gradient

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

DQN

Preprocessing with CNN(Convolution Neural Network)

- Why?

  Raw Atari frames : 210 × 160 pixel images with a 128 color palette

  -> computationally demanding

- Convert their RGB representation to gray-scale & Down-sample it to a 110×84 image.
- Crop an 84 × 84 region of the image that roughly captures the playing area
- Preprocess function φ
  - applies this preprocessing to the last 4 frames of a history
  - stacks them to produce the input to the Q-function

DQN
Algorithm

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 $\boxed{3}$
  end for
end for

Deep Reinforcement Learning
Result

Close to Human Performance

Better than other algorithms

	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

Better than Human Performance

Far from Human Performance

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
HNeat Best	3616	52	106	19	1800	920	**1720**
HNeat Pixel	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

HNeat : produce deterministic policies that always get the same score

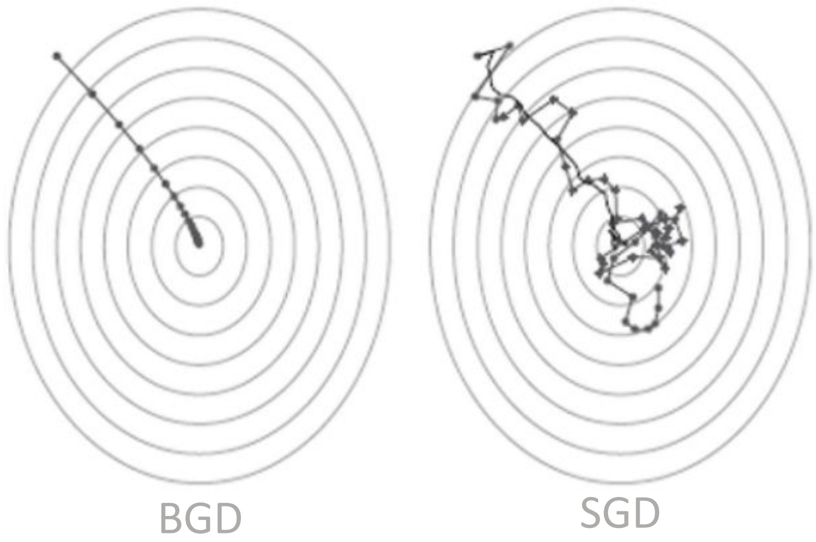
- Best : hand-engineered object detector algorithm (outputs the locations and types of objects on the Atari screen)

- Pixel : the special 8 color channel representation of the Atari emulator (represents an object label map at each channel)

DQN : ε-greedy policy with ε = 0.05.

DQN
Characteristic of DQN - 1

- SGD(Stochastic Gradient Descent)
  - If we use batch update(BGD, Batch Gradient Descent), it will be proportional to the size of data set
  - SGD have low constant cost per iteration and scale to large data-sets
- Advantage of using SGD
  - More steps can be made at the same time
  - If repeated several times, converge as a result of batch processing.
  - High possibility of converging in a better direction without falling into Local Minima(BGD can fall)



DQN
Characteristic of DQN - 2

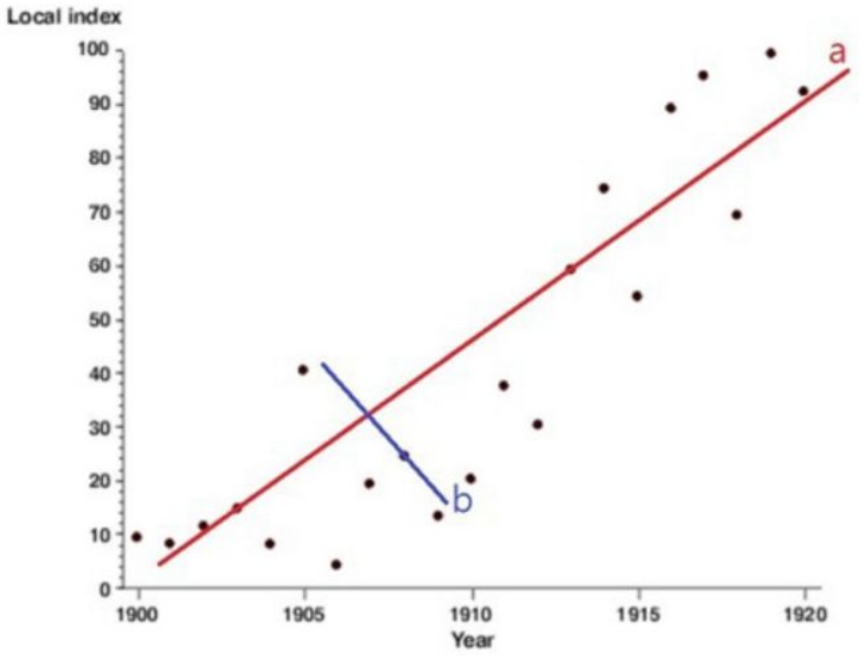
- Approximate Q with neural network
  - Can use image pixel information, not hand-craft feature
  - If it implemented by array, It would have been difficult to achieve correlated results under similar states
  - Because CNN automatically extracts important information from the game and then calculates each Q value again based on those features, you can also expect robust calculations for small state changes.

DQN

Characteristic of DQN - 3

- Experience Replay
  - store the agent's experiences at each time-step, pooled over many episodes into a replay memory

    $e_t = (s_t, a_t, r_t, s_{t+1})$ **in a data-set** $\mathcal{D} = e_1, \ldots, e_N$
  - During the inner loop of the algorithm, we apply Q-learning updates, or minibatch updates, to samples of experience, $e \sim \mathcal{D}$, drawn at random from the pool of stored samples
  - a is the correct answer in the whole, but b can be the answer in the vicinity of b. Experience Replay prevents this situation in the RL environment.



Deep Reinforcement Learning

Conclusion(In my opinion)

- Learn with raw data, not processed data.
- learn anything compared to the previous algorithm.

DQN means a lot.

References

- Volodymyr Mnih, “Playing Atari with Deep Reinforcement Learning”, NIPS Deep Learning Workshop 2013
- 이웅원, 『파이썬과 케라스로 배우는 강화학습』, 위키북스(2017)
- https://github.com/deephoony/RL-Lecture
- https://mangkyu.tistory.com/60
- https://blog.lgcns.com/1692
- http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html
- https://wwiiiii.tistory.com/entry/Deep-Q-Network
- https://steemit.com/deep-learning/@backhoing/deep-reinforcement-learning-with-dobule-q-learning
- http://ddanggle.github.io/demystifyingDL
- https://brunch.co.kr/@kakao-it/73
- https://cding.tistory.com/64
- https://eatch.net/105

Thank You