BITS F464: Machine Learning

29 Reinforcement Learning (RL)

Dr. Kamlesh Tiwari
Assistant Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

April 05, 2021 ONLINE (Campus @ BITS-Pilani Jan-May 2021)

http://ktiwari.in/ml

Trial and Error

- In 1956 **Arthur Samuel** wanted to develop a Checkers playing program that could beat him
- Idea was to let the computer play lot of games against itself and learn
- In 1962 the computer won over human player
- Father of ML

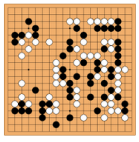
He defined ML as a field of study that gives computers the ability to learn without being explicitly programmed.

Learning A Game

Consider playing game of Atari, Chess or Go.

- They all requires intuition, creative and strategic thinking





DeepBlue won 2 chess game over *Gary Kasparov* (3D+1L) in 1997
AlphaGo in March 2016, beat *Lee Sedol* a professional GO player.

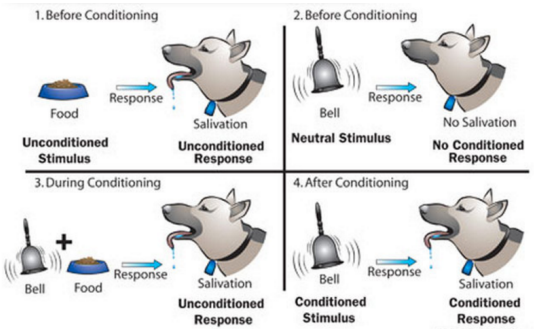
What one want to learn?

- Rule of game, or
- Tactic/skill (policy) to play

Reinforcement Learning

- Reinforcement learning works on generate and test methodology
- An agent automatically enumerate solution and compute reward/penalty
- Task of agent is to learn from indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward
- Agent may have no prior knowledge of the effects of its actions on the environment
- Neither supervised nor unsupervised, it is trial and error
- Reinforcement learning algorithms are related to dynamic programming
- Example includes learning to control a mobile robot, learning to optimize operations in factories, and learning to play board games

Reinforcement Learning

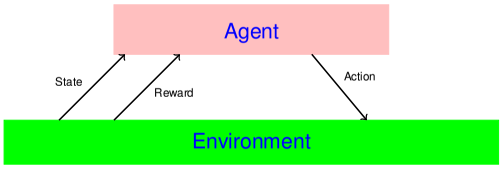


Rewards can lead to learning[1]

[1] Image from: http://www.krigolsonteaching.com/reinforcement-learning.html

State, Action, and Reward

Problem of learning a control policy to maximize cumulative reward is very general



$$S_0 \xrightarrow[r_0]{a_0} S_1 \xrightarrow[r_1]{a_1} S_2 \xrightarrow[r_2]{a_2} S_3 \xrightarrow[r_3]{a_3} \dots$$

Goal is to choose a policy that maximizes

$$r_o + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \dots \qquad \textit{where } 0 \le \gamma < 1$$

Reinforcement Learning

- Target function to be learned is a policy $\pi : S \rightarrow A$ where S and A are set of states and actions. Important points are
  - Delayed reward: $r(s, \pi(s))$ is unavailable
  - Exploration: agent influences the distribution of training examples
  - Partially observable states: forward camera cannot see behind
  - Life-long learning: using previous experience to learn new tasks
- Assumption is *Markov decision process* (MDP), Therefore, reward $r_t = r(s_t, a_t)$ and next state $s_{t+1} = \delta(s_t, a_t)$
- **Discounted cumulative reward** of a policy π from initial state s_t is

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

here $0 \leq \gamma < 1$ is a constant determining relative value of delayed versus immediate rewards

Other Reward Function

- Finite horizon reward

$$\sum_{i=0}^{h} r_{t+i}$$

- Average reward

$$\lim_{h \rightarrow \infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$$

Agent's learning task is

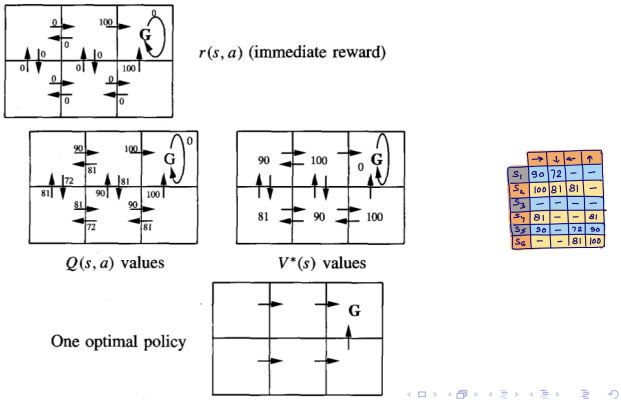
to determine optimal policy

$$\pi^* = argmax_{\pi} V^{\pi}(s), \forall s$$

$V^{\pi^*}(s)$ is also represented as $V^*(s)$ for simplicity

An Example

Let $\gamma = 0.9$



Q Learning

- Specifies a method for an agent to learn an optimal policy
- Agent should prefer state s_1 over state s_2 when $V^*(s_1) > V^*(s_2)$
- Optimal action in state s is

$$\pi^*(s) = \underset{a}{\text{argmax}} \left(r(s, a) + \gamma V^*(\delta(s, a)) \right)$$

problem is that r and δ are not known

- Define a function $Q(s, a)$ representing maximum discounted cumulative reward that can be achieved starting from state s and applying action a as the first action

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

- Then

$$\pi^*(s) = \underset{a}{\text{argmax}} Q(s, a)$$

Q Learning

- One can choose globally optimal action sequences by reacting repeatedly to the local values of Q for the current state
- Value of Q can be found through iterative approximation
- As $V^*(s) = max_{a'} Q(s, a')$ it is easy to see

$$Q(s, a) = r(s, a) + \gamma max_{a'} Q(\delta(s, a), a')$$

- In terms of iterative approximation

$$\hat{Q}(s, a) = r + \gamma max_{a'} \hat{Q}(s', a')$$

where immediate reward $r = r(s, a)$ and $s' = \delta(s, a)$

Q Learning Algorithm

Algorithm for Q Learning can be specified as below

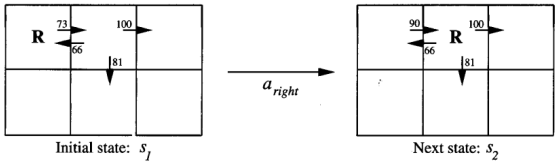
Algorithm 1: Q Learning

```
For each s and a pair; initialize table entry Q̂(s, a) = 0
Observe the current state s
while True do
    Select an action a and execute
    Receive immediate response/reward r
    Observe new state s'
    Update the Q̂(s, a) = r + γ max_a' Q̂(s', a')
    s = s'
```

- Produces the, Q Table
- Guess expected running time?

Q Learning Example

Consider a single action for the example below (let $\gamma = 0.9$)



$$\begin{aligned}\hat{Q}(s_1, a_{right}) &= r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &= 0 + 0.9 \max_{a'}\{66, 81, 100\} \\ &= 90\end{aligned}$$

Does it converge?

- Yes, under certain assumptions
  1. Deterministic MDP
  2. Rewards are bounded $|r(s, a)| < c$
  3. Agent visits every possible state-action pair infinitely often
- Note that $\hat{Q}(s, a)$ does never decreases in any iteration

$$\hat{Q}_n(s, a) \leq \hat{Q}_{n+1}(s, a)$$

- And is upper bounded by $Q(s, a)$ *i.e.*

$$0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

- Consider consecutive intervals during which each state-action transition occurs at least once[2]

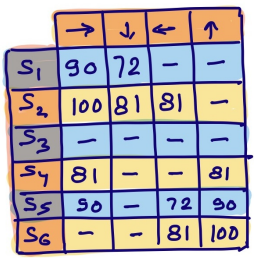
[2] Random Walk takes $n \log n$ time to traverse fully connected graph with n nodes

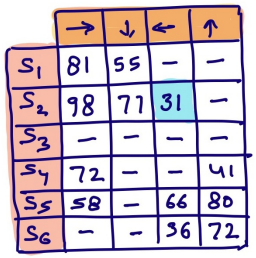
Does it converge (contd)

- Maximum error in $\hat{Q}$

$$\triangle_n = max_{s,a}|\hat{Q}(s, a) - Q(s, a)|$$

is maximum cell wise difference [3]





[3] $\triangle_n = 50$ in the above example

Does it converge (contd)

- Let $\triangle_n$ be the maximum error in $\hat{Q}$ i.e.

$$\triangle_n = max_{s,a}|\hat{Q}(s, a) - Q(s, a)|$$

- For any $\hat{Q}_n(s, a)$ that is updated on iteration $n + 1$, the magnitude of the error in the revised estimate $\hat{Q}_{n+1}(s, a)$ is

$$\begin{aligned}|\hat{Q}_{n+1}(s, a) - Q(s, a)| &= |(r + \gamma max_{a'} \hat{Q}_n(s', a')) - (r + \gamma max_{a'} Q(s', a'))| \\ &= \gamma |max_{a'} \hat{Q}_n(s', a') - max_{a'} Q(s', a')| \\ &\leq \gamma max_{a'} |\hat{Q}_n(s', a') - Q(s', a')| \\ &\leq \gamma max_{s'', a'} |\hat{Q}_n(s'', a') - Q(s'', a')| \\ &= \gamma \triangle_n\end{aligned}$$

- Starting from $\triangle_0$ error reduces to $\gamma \triangle_0, \gamma^2 \triangle_0, ..., \gamma^n \triangle_0, ..., 0$

Therefore, $\hat{Q}$ converges to Q after infinitely many steps ...

Experimentation Strategies

- Can we select the action that maximizes $\hat{Q}(s, a)$
  NO, as one have to visit each state infinitely often for convergence
- For some $k > 0$

$$Pr(a_i|s) = \frac{k^{\hat{Q}(s,a_i)}}{\sum_j k^{\hat{Q}(s,a_j)}}$$

Nondeterministic Rewards and Actions

- Reward and transition function may have probabilistic nature
- Policy π is then based on expected value. Discounted cumulative reward is taken as $V^{\pi}(s_t) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$
- Definition of Q becomes

$$\begin{aligned}Q(s, a) &= E[r(s, a) + \gamma V^*(\delta(s, a))] = E[r(s, a)] + \gamma E[V^*(\delta(s, a))] \\ &= E[r(s, a)] + \gamma \sum_{s'} P(s'|s, a) V^*(s') \\ &= E[r(s, a)] + \gamma \sum_{s'} P(s'|s, a) max_{a'} Q(s', a')\end{aligned}$$

- Earlier training rule (for the deterministic) fails to converge here. Therefore, decaying weight average is taken as

$$\hat{Q}_n(s, a) = (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n[r + \gamma max_{a'} \hat{Q}_{n-1}(s', a')]$$

where $\alpha_n = \frac{1}{1 + visits_n(s,a)}$ where $visits_n(s, a)$ is number of times (s, a) is visited till nth iteration.

There is a problem

- The method is not scalable.
- As one have to compute $\hat{Q}(s, a)$ for every pair and
- infinitely many times.
- It is huge problem as the state space for typical games is large.

What is the solution?

We can use a function approximator, that would estimate the action value function $Q(s, a)$.

- One such an approximator could be a Neural Network.
- When it is a Deep Neural Network, we call it a deep Q-learning. [4]

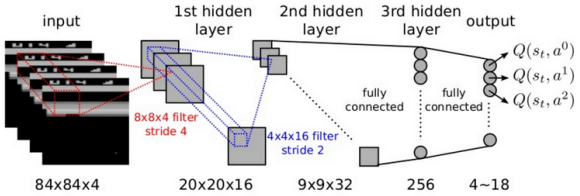
[4] Human-level control through deep reinforcement learning, *Mnih, Volodymyr and Kavukcuoglu, Koray and Silver, David and Rusu, Andrei A and Veness, Joel and Bellemare, Marc G and Graves, Alex and Riedmiller, Martin and Fidjeland, Andreas K and Ostrovski, Georg and others*, 518(7540) pp 529–533, Nature 2015

Deep Reinforcement Learning for Atari Game

- Objective is to complete the game with highest score
- States are the raw pixel values in the game screen
- Actions are moving controls left/right/up/down
- Reward is the increase/decrease of score in each timestamps

Deep Reinforcement Learning for Atari Game

The network that we are going to use is this[5]



- Preprocessing: Grayscale conversion and scale down
- Network takes a stack of four frames as an input to exploit spatial relationships in images
- Convolution layer uses ELU (exponential linear unit) activation[6]
- Experience Replay is used to make more efficient use of observed experience

[5] https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/deep_q_learning.html

[6] https://arxiv.org/pdf/1511.07289.pdf

Deep Q-Learning with Experience Replay

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

Thank You!

Thank you very much for your attention!

Queries ?

(Reference[7])

[7] Book - *Machine Learning*, ch-9/13, Tom Mitchell.