Intro to Machine Learning

Augustin Cosse.

Fall 2021

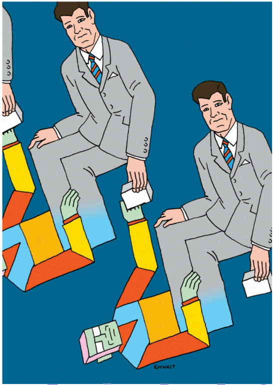
November 30, 2021

Reinforcement learning

- Reinforcement learning is learning what to do so as to maximize a numerical reward signal
- "The learner is not told which action to take but instead must discover which action yield the most reward by trying them"
- Ex.1.: "A chess player makes a move. The choice is informed by planning (anticipation of possible replies and counterreplies) and by immediate intuitive judgements of the desirability of possible positions and moves"
- Ex.2. "A mobile robot decides whether it should enter a room in search of a target or start to find its way back to its battery charging station"

source: R. Sutton, A.G. Barto, Reinforcement learning: An introduction

10 BREAKTHROUGH TECHNOLOGIES



Reinforcement Learning

By experimenting, computers are figuring out how to do things that no programmer could teach them.

Availability: 1 to 2 years

by Will Knight

Reinforcement learning: constitutive elements

- The policy defines the learning agent's way of behaving at any given time
- On each time step, the evironment sends to the reinforcement learning agent a single number called the reward which specifies what are good and bad events in an immediate sense.
- To know what is good in the long run, we use a value function which is the total amount of reward the agent can expect to accumulate over the future, starting from that state.
- Finally, the last element is a model for the environment which enables inferences to be made on how the environment will react w.r.t a particular action. The role of the model is essentially to predict the next state and next reward given the current state and action.

source: R. Sutton, A.G. Barto, Reinforcement learning: An introduction

Reinforcement learning

- Actions are usually taken to maximize the value, not the reward, because high value actions are those that lead to the highest level of reward in the long run. Finding such actions is however hard as those have to be constantly re-estimated based on the decisions of the agent.

- An important instance of reinforcement learning in which there is a single state is the bandit problem

source: R. Sutton, A.G. Barto, Reinforcement learning: An introduction

Reinforcement learning

- We usually distinguish two types of feedbacks
  - Instructive feedback which indicates the correct action to take, given the action taken (this action is the basis of supervised learning)
  - Evaluative feedbacks indicate whether an action that was taken was good or bad but does not indicate whether it was the best or worst possible action
- Reinforcement learning as opposed to supervised learning evaluates (not in an absolute sense) the action taken rather than instructing by giving correct actions.
- Reinforcement learning is usually studied in a simplified framework (non associative = change in the response to a stimulus based on repeated exposure to that stimulus) known as multi-armed bandit problem

source: R. Sutton, A.G. Barto, Reinforcement learning: An introduction

Reinforcement learning

- The Multi-armed bandit problem is an instance of non associative learning.
- Non associative learning = no more than one situation or no feedback on the situation (i.e no way to associate the reward to a particular situation or state of the environment)
- >< Associative learning = different actions are best in different environements

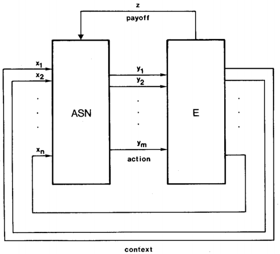


Fig. 1. An ASN interacting with an environment E. The ASN receives context signals $x_1, \ldots, x_n$ and a payoff or reinforcement signal z from E and transmits actions to E via output signals $y_1, \ldots, y_m$

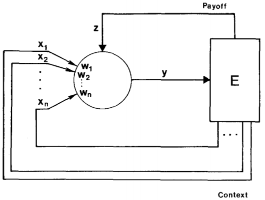
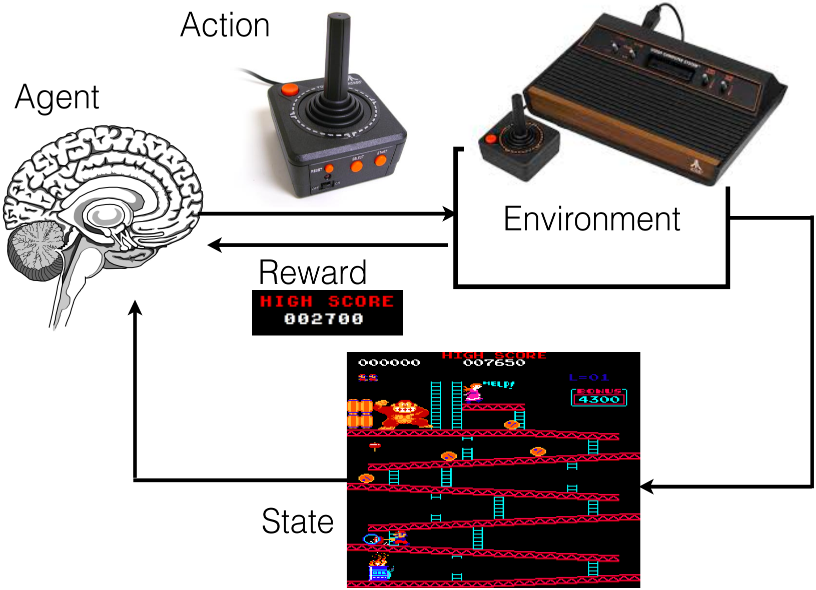


Fig. 2. The simplest ASN: A single adaptive element interacting with and environment E

source: Barto, Sutton, and Brouwer, Biological Cybernetics, 1981.

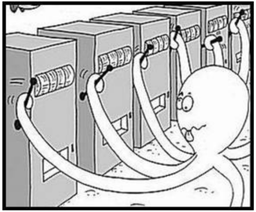


Inspired from https://keon.io/deep-q-learning/, Deep Q-Learning with Keras and Gym

Reinforcement learning

- The multi-armed bandit is a simplified version of non associative feedback problem

- In the k-armed bandit problem, you are faced repeatedly with a choice among k different possible options or actions. After each choice, you receive a numerical reward chosen from some stationnary probability distribution that depends on the action you selected.



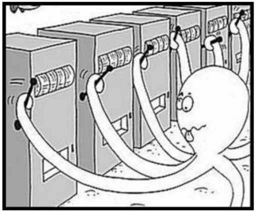
source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning

- You can think of the k-armed bandit problem as the problem of playing one of the k levers of a slot machine. You choose which lever you play and the reward is the payoff for hitting the jackpot

- The value of an arbitrary action, a, which we denote $v(a)$ is the expected reward given that you selected a

$$v(a) = \mathbb{E}\left\{R_t | A_t = a\right\}$$



source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning

- We don't know the exact value $v^*(a)$ (because we don't know the distribution). So we would like an estimate $v_{\text{est},t}(a)$ (estimated value at time t) that would be as close as possible to $v^*(a)$

- When you keep track of the estimated action values through time, then at each time step, there is always at least one action whose estimated value is best. We call this greedy actions.

- When you select one of these actions, we say that you are exploiting your current knowledge of the values of the actions

- When you select one of the non greedy actions, then we say you are exploring. In particular, exploring enables you to improve your estimates of the non greedy action's values.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning

- Exploitation will maximize your expected reward on the one step but exploration may lead to greater total reward in the long run.
- Intuitively, if you have many time steps ahead, it may be better to explore.
- How do we balance exploration and exploitation when dealing with the k-armed bandit problem?

source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning: Action value estimates

- The first thing we want to do is get an estimate of the value of an action at time t.
- The natural approach is to average over the rewards received in the past

$$v_{\text{est},t}(a) = \frac{\text{sum of rewards when } a \text{ taken}}{\text{number of times } a \text{ taken}} = \frac{\sum_{i=1}^{t-1} R_i \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t} \mathbb{1}_{A_i=a}}$$

Here $\mathbb{1}_{\text{predicate}}$ is used to denote the indicator function for the predicate. $\mathbb{1}_p = 1$ if the predicate is verified and 0 otherwise.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning: Action value estimates

- Then the simplest action selection procedure (known as greedy action selection) is to select (one of) the action(s) with the highest estimated value,

$$A^* = \underset{a}{\operatorname{argmax}}\, v_{\mathrm{est},t}(a)$$

- Greedy action selection always exploits current knowledge to maximize immediate reward (i.e it does not spend time investigating inferior actions to see if they might be better)

- A group of alternative methods known as ε-greedy methods consist in behaving greedily most of the time, but once in a while (with probability ε) select an action randonly (with uniform probability) from the list of all possible actions.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning: Action value estimates

- ε-greedy methods ensure that every action is sampled an infinite number of times. Which in turns implies that the estimator $v_{\text{est}}(a)$ converges to the v^* (the true expected value)

- To avoid keeping each reward in memory independently, typical implementations of greedy and ε-greedy only update the averaged reward (a.k.a value). If Q_n is used to denote the value of a given action after the n^{th} step,

$$Q_n = \frac{R_1 + R_2 + \dots R_{n-1}}{n-1}$$

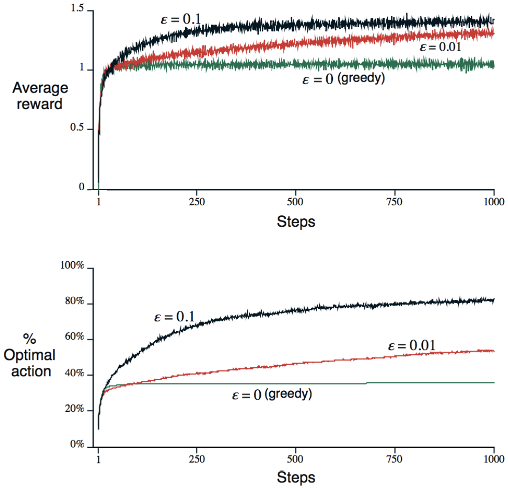
we compute Q_{n+1} as

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right)$$

$$= Q_n + \frac{1}{n}\left[R_n - Q_n\right]$$

source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning: greedy vs ε-greedy



source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning: Simple Bandit algorithm

1. Initialize, for every action $a = 1$, to k
   1.1 $v(a) \leftarrow 0$
   1.2 $n(A) \leftarrow 0$ (number of times A has been chosen)
2. Repeat
   2.1 $A \leftarrow \begin{cases} \underset{a}{\text{argmax}}\, v(a) & \text{with probability } 1-\varepsilon \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
   2.2 $R \leftarrow \text{bandit}(a)$
   2.3 $n(A) \leftarrow n(A) + 1$
   2.4 $q(A) \leftarrow v(A) + \frac{1}{N(A)}[R - v(A)]$

source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning: Simple Bandit algorithm

- So far we have focused on stationnary Bandit problems (Problems for which the reward probabilities do not change with time).

- When the problems are not stationnary, the choice of an action will depend on the instant at which the action is taken. In particular we will want to give more weight to the rewards associated to more recent actions. One way to achieve this is to add a weight in the update rule for the value V_{n+1},

$$v_{n+1} = v_n + \alpha(R_n - v_n)$$

  Developing, we get

$$v_{n+1} = (1-\alpha)^n v_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$$

source: Sutton & Barto, Reinforcement Learning: An Introduction.

Reinforcement learning: Q-learning

- How can we extend this idea to a more complex framework in which we face multiple states (e.g. the best action on the stock market is highly dependent on the state of the market)
- Under an important assumption (called stationnarity for preferences), the utility associated to a sequence of states $s_0, s_1, \ldots$ is known to be defined as the sequence of discounted rewards

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

- Given this definition, and provided that the agent chooses the best action each time (the action that maximizes the expected utility), our utility satisfies the Bellman equation

$$U[s] = R[s] + \gamma \max_a \sum_{s'} P(s'|s, a) U[s']$$

Reinforcement learning: Q-learning

- In fact one can extend this equation to the associative framework by introducing the notion of Q-table
- A Q-table is a way to store the value of a pair (s, a) (corresponding to being in state s and taking action a). The corresponding framework is known as Q-learning
- One approach in Q-learning consists in requiring the Q-table to satisfy the equality

$$Q[s, a] = R[s] + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q[s', a']$$

That is to say, we encode the value of a state-action pair as the immediate reward of the state plus the best possible value we will be able to achieve in the subsequent state (i.e. we take an optimistic viewpoint)

Reinforcement learning: Q-learning

- Given the Bellman update on Q-table, we can define what is known as a time difference update to learn the Q-table

$$Q[s,a] \leftarrow Q[s,a] + \eta \left(R[s] + \gamma \max_{a'} Q[s',a'] - Q[s,a] \right)$$

  This update corresponds to adding to our current guess for Q a correction (based on the difference between the right handside and the left handside of the Q-table) for the sample s, a, s' that is acquired by the agent

- Note that we replace the average on the RHS of the update by an estimate based on a single sample (this is because we assume that for a sufficiently long simulation, we will ultimately add corrections corresponding to each of the terms in the average). In short we approximate the average with an empirical estimate based on the samples we get through the iterations.

Reinforcement learning: Generalization

- A major problem with the Q-table approach introduced before is the space needed to store the table and the fact that for a large environment, most of the states will be unexplored so that when the agent will find itself in those states, it will not know what to do.

- A solution to this problem consists in learning a parametric model for the Q-table

- We can then consider the learning problem as a sequential where the objective is to reduce the gap between our current value estimate stored in the Q-table and the right handside of the Bellman equation

$$\min_{\theta} \left\| \hat{Q}[s, a] - (R[s] + \gamma \max_{a'} \hat{Q}[s', a']) \right\|^2$$

Reinforcement learning: Generalization

- Considering a small learning rate η and taking a stochastic viewpoint, we get the sequential gradient updates

$$\theta_i \leftarrow \theta_i + \eta \left[R[s] + \gamma \max_{a'} \hat{Q}[s', a'] - \hat{Q}[s, a] \right] \frac{\partial \hat{Q}_\theta[s, a]}{\partial \theta_i}$$

- Within this framework, we can then use any of the models that were introduced in the supervised part of the course

- A popular approach (known as deep Q-learning) stores the Q-table as a neural network and update the weights of the network each time a new sample s, s', a is obtained

- See *Playing Atari with Deep Reinforcement Learning* by DeepMind