New York University: Machine Learning in Economics

Lecture 12: Reinforcement Learning

Dr. George Lentzas

Introduction

What is Reinforcement Learning

- Reinforcement Learning (RL) is a branch of machine learning that deals with how agents should act in an unknown, stochastic environment when they are trying to maximize some long term goal they care about.
- In this context "learning" is by trial and error, that is by direct interaction with the environment through agent action and perception.
- RL is different from supervised learning in two important ways:
  1. There is no clear separation between training and testing data; instead learning is "on-line" with data generated by the agent interacting with the environment.
  2. Feedback for an action chosen by the agent is 'evaluative' but not 'instructive'; the agent know how good he action was but not if it was the best action.

Introduction

Formalization

- More formally we have an agent who observes the state of the world s_t in $\mathscr{S}$ and chooses an action α_t in $\mathscr{A}$ to then receive a reward r_{t+1} and end up in the next state s_{t+1}.
- The environment is expected to be non-deterministic: the same action in the same state may result in different next states and/or reinforcement values in different occasions.
- The probabilities of state transitions are however constant over time (environment is stationary) but not necessarily known.

Introduction

A Simple Case

- Let us looks at a motivating example known as the n-arm bandit problem. This is actually used in practice very often!
- You are faced with a choice between n different actions; after each choice you receive a reward from a stationary probability distribution that depends on the action taken and your objective is to maximize the expected total reward over $1,000$ actions. What do you do?
- A number ways to do this, here we will look at "Action-Value-Greedy" strategy. The idea is to keep estimates of the values of actions and do take the best action most of the time. Formally, we use estimates of values of actions

$$Q_t(\alpha) = \frac{r_1 + \cdots + r_{k_\alpha}}{k_\alpha}$$

  where k_α is the number of times action α has been taken.
- Then the action taken is

$$\alpha^* = \arg\max_\alpha \left[Q_t(\alpha) \right]$$

  with probability $1 - \epsilon$ and a random action with probability ϵ.

Introduction

A Simple Case

- This example illustrates a key challenge of RL: "the exploration-exploitation" trade-off. How to optimally balance these two needs is a key aspect of successful learning.
- It also illustrates a key idea, to keep tabular estimates of the values of actions (or states-actions later).
- Action-Value Greedy is not the only way to go here; alternative methods include Softmax selection, reinforcement comparison, or Pursuit methods.
- In practice Action-Value Greedy is implement incrementally to avoid memory problems,
$$Q_{k+1}(\alpha) = Q_k(\alpha) + w_k \left[r_{k+1} - Q_k(\alpha) \right]$$
- From stochastic approximation theory if $\sum_{k \to \infty} |w_k| \to \infty$ and $\sum_{k \to \infty} (w_k)^2 \to 0$ this will converge. Two common choices for w_k are $w_k = \frac{1}{k+1}$ (converges) and $w_k = w$ (does not converge, but not necessarily a bad thing).

Introduction

A Simple Case

- Where do you start the above iteration? Set $Q_0(\alpha)$ at high values.
- This is called "optimistic starting values" and it encourages exploration in the beginning of the process which is desirable.
- When is RL a good idea? When observing the result is not a good indication if the action chosen was optimal.
- In other words when observing success does not mean you did the right thing (and/or similar for failure). This is called instructive feedback breakdown.
- So why is this a simple case? Because there is only 1 state of the world and choosing the best action does not depend on the state of the world.
- This is of course very restrictive; in most interesting applications the best action depends on the state of the world and such problems are called "associative".

Formalization

Markov Decision Processes

- Having defined the action $\mathscr{A}$ and state $\mathscr{S}$ sets we need a toolkit to describe the environment. Without much loss of generality this is assumed to be Markovian, namely to satisfy:

$$P^{\alpha}_{ss'} := \Pr\left(s_{t+1} = s' | s_t = s, \alpha_t = \alpha\right) = \Pr\left(s_{t+1} = s' | s_t, \ldots, s_0, \alpha_t, \ldots, \alpha_0\right)$$

$$R^{\alpha}_{ss'} := \mathrm{E}\left(r_{t+1} | s_{t+1} = s', s_t = s, \alpha_t = \alpha\right) = \mathrm{E}\left(r_{t+1} | s_{t+1}, \ldots, s_0, \alpha_t, \ldots, \alpha_0\right)$$

- The one step state-action and reward dynamics along with $\mathscr{A}$ and $\mathscr{S}$ define a Markov Decision Process (MDP).

Formalization

Value Functions and Policies

- What does it mean for an agent to maximize some long term goal? The first thing to consider is how does the agent takes the future into account.
- There are three main approaches here (in all the r_t is the reward received at time t:
  1. *The Finite Horizon Model*: at any given moment in time the agent optimizes the expected reward for the next h steps
  $$E\left(\sum_{t=0}^{h} r_{t+1}\right)$$
  This model can be used in either of two ways: in one the agent always looks at h time steps ahead (so the end horizon is constantly receding) and in the other the end horizon is fixed hence the agent looks at h steps, then $h-1$ and so on.
  2. *The Infinite Horizon Model*: the long term reward of the agent is considered but rewards received in the future are discounted with a factor γ to give
  $$\mathrm{E}\left(\sum_{t=0}^{\infty} \gamma^{t} r_{t+1}\right)$$
  3. *The Average Reward Model*: the average long term reward is considered
  $$\lim_{h\to\infty} \mathrm{E}\left(\frac{1}{h}\sum_{t=0}^{h} r_{t+1}\right)$$

Formalization

Value Functions and Policies

- We will proceed with the Infinite Horizon Model and denote the long term reward as
$$\mathrm{E}\left(R_t\right)$$
where
$$R_t = \left(\sum_{t=0}^{\infty} \gamma^t r_{t+1}\right)$$
- The two main building blocks of RL to answer the question of how do we think about maximizing this long term reward are the *value function* and the *policy*.
- A policy $\pi_t\left(s, \alpha\right)$ is a temporal mapping from states to actions, i.e. the probability that the agent will choose action α in state s at time t.

Formalization

Value Functions and Policies

- In order to quantify what a 'good' policy is we introduce the value function. A value function maps states to a number (long term worth) assuming the agent follows a policy. This is given by

$$V^{\pi}(s) = \mathrm{E}_{\pi}(R_t | s_t = s)$$

- Intuitively, $V^{\pi}(s)$ captures the long term reward the agent expects to earn starting at state s if they follow policy π.
- Given the above, solving the RL problem comes down to finding the 'optimal' policy π^* which is the (not necessarily unique) policy that is always better or at least as good as all other policies

$$V^{\pi^*}(s) := V^*(s) = \max_{\pi} V^{\pi}(s)$$

Bellman Equations

Value Function Bellman Equations

- Value functions satisfy a number of consistency conditions known as Bellman Equations

$$V^*(s) = \max_{\alpha} \left[\mathrm{E}\left(r_{t+1} + \gamma V^*(s_{t+1}) \,|\, s_t = s, \alpha_t = \alpha \right) \right]$$

and

$$V^*(s) = \max_{\alpha} \left[\sum_{s'} P^{\alpha}_{ss'} \left(R^{\alpha}_{ss'} + \gamma V^*(s') \right) \right]$$

Bellman Equations

Action - Value Function Bellman Equations

- We can also look at the Action Value Function Q given by

$$Q^{\pi}\left(s,\alpha\right)=\mathrm{E}_{\pi}\left(R_t|s_t=s,\alpha_t=\alpha\right)$$

- This gives the value of starting at state s, taking action α and then following policy π. It also satisfies Bellman Equations

$$Q^{*}\left(s,\alpha\right)=\mathrm{E}\left(r_{t+1}+\gamma\max_{\alpha'}Q^{*}\left(s_{t+1},\alpha'\right)|s_t=s,\alpha_t=\alpha\right)$$

and

$$Q^{*}\left(s,\alpha\right)=\sum_{s'}P_{ss'}^{\alpha}\left(R_{ss'}^{\alpha}+\gamma\max_{\alpha'}Q^{*}\left(s',\alpha'\right)\right)$$

Dynamic Programming

Iterative Updating

- A popular way to solve RL type of problems is to use Dynamic Programming.
- The core idea is to turn the Bellman Equations into iterative update rules until convergence.
- Using the Value function an iterative rule can be written as

$$V_{k+1}(s) = \mathrm{E}_{\pi}\left[r_{t+1} + \gamma V_k(s_{t+1} | s_t = s)\right]$$

- Dynamic Programming assumes that the environment ($R^{\alpha}_{ss'}$, and $P^{\alpha}_{ss'}$) is known.
- This allows us to calculate the expectation analytically at each iteration and update all states.

Dynamic Programming

Iterative Updating

- The above algorithm is called *policy estimation* as it involves estimating the value function of a particular policy.
- It is usually complemented by a *policy improvement* step; once the value of a policy has been calculated the policy is improved by taking a greedy action at each state, i.e.

$$\pi'(s) = \arg\max_{\alpha} \mathrm{E}_{\pi}\left[r_{t+1} + \gamma V^{\pi}(s_{t+1}) \,|\, s_t = s, \alpha_t = \alpha\right]$$

- Policy estimation and policy improvement are usually combined and repeated until a policy has been estimated that is greedy w.r.t to itself.

Dynamic Programming

Policy / Value Iteration and Shortcomings

- Two specific ways to do this are Policy Iteration (alternate between the two steps) and Value Iteration where we combine the two steps in one at each iteration and update the value function given the greedy action using

$$V_{k+1}(s) = \mathrm{E}_{\pi}\left[r_{t+1} + \gamma V_k\left(s_{t+1}|s_t = s\right)\right]$$

- Dynamic Programming has two obvious limitations:
- It assumes that the environment (transition probabilities and conditional reward distributions) is known; this is unrealistic in most interesting applications.
- It is not easily applicable to large state space problems, where full updating of all states at each iteration can lead to polynomial time solutions.

Q Learning

The Action Value Function

- Q Learning is a powerful RL algorithm that combines Dynamic Programming with so called Monte Carlo methods.
- The idea is to still use the iterative update rules but now learning can happen without a model, simply from experience. This idea gives rise to a number of RL algorithms.
- Recall that the Action Value Function is given by

$$Q^{\pi}(s,\alpha) = \mathrm{E}_{\pi}\left(R_t | s_t = s, \alpha_t = \alpha\right)$$

and satisfies the Bellman Equations

$$Q^{*}(s,\alpha) = \mathrm{E}\left(r_{t+1} + \gamma \max_{\alpha'} Q^{*}\left(s_{t+1},\alpha'\right) | s_t = s, \alpha_t = \alpha\right)$$

and

$$Q^{*}(s,\alpha) = \sum_{s'} P^{\alpha}_{ss'}\left(R^{\alpha}_{ss'} + \gamma \max_{\alpha'} Q^{*}\left(s',\alpha'\right)\right)$$

Q Learning

The Algorithm

- We will use the Action Value Function Q now because, without a model of the environment, we cannot any longer evaluate the expected rewards given an action.
- Using Q we can evaluate state-action pairs explicitly and improve the policy by taking the greedy action.
- The Q Learning Algorithm is given by

$$Q\left(s_t, \alpha_t\right) \leftarrow Q\left(s_t, \alpha_t\right) + \omega\left[r_{t+1} + \gamma \max_{\alpha} Q^*\left(s_{t+1}, \alpha\right) - Q\left(s_t, \alpha_t\right)\right]$$

- For finite states and under the step convergence conditions mentioned above, Q Learning will converge to the Optimal Value Function Q^*.
- Convergence happens irrespective of how the agent actually generates experience, as long as all action-state pairs continue to be updated.

Q Learning

Large State Spaces

- Up to now we have made the assumption the value function (V or Q) can be represented as a table.
- In most interesting RL problems this is not realistic; the state space is either very large or infinite. Then we cannot use a table anymore (and even if we could it would be inefficient).
- To solve this we use function approximation in order to generalize experience. In other words, we express Q as a function of some vector of features $\boldsymbol{\theta}$.
- Each experience update is now a training example for the functional approximation

$$Q(s_t, \alpha_t) = Q(s_t, \alpha_t; \boldsymbol{\theta})$$

Q Learning

Semi Gradient Q Learning

- We want to estimate a parameter vector $\boldsymbol{\theta}$ which will minimize

$$MSE(\boldsymbol{\theta}) = \sum_s \Pr(s) \left(Q^\pi(s, \alpha) - Q_t(s, \alpha) \right)$$

- $\Pr(s)$ is the distribution under which we will experience states under the policy π.
- We can adapt the usual Gradient Descent algorithm

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \omega \left[\underbrace{r_{t+1} + \gamma Q_t(s_{t+1}, \alpha_{t+1})}_{v_t} - Q_t(s_t, \alpha_t) \right] \nabla_{\boldsymbol{\theta}} Q_t(s_t, \alpha_t)$$

- As long as v_t is an unbiased estimate of Q^π then convergence to some local optimum is guaranteed.

Q Learning

Semi Gradient Q Learning

- Two methods commonly used for function approximation in RL: (i) Linear models and (ii) Multilayer Artificial Networks
- In a linear model, Q is a linear function of the features

$$Q\left(s_t, \alpha_t\right) = \boldsymbol{\theta}_t' \boldsymbol{\phi}_s$$

where ϕ_s is the vector of features.

- In the linear case there is only one local maximum so global maximum is guaranteed in cases of convergence.
- Central to the above approach is that the states are sampled according to the "on-policy" distribution (the distribution of states encountered given while the agent is interacting with the environment and selecting actions according to the policy whose value function we are approximating).
- For other distributions, this method may diverge.

Q Learning

Control with Function Approximation

- If the action set is discrete and not too large then we can use a grid search method

$$\alpha^* = \arg\max_{\alpha} Q_t\left(s_t, \alpha\right)$$

- If the action set is continuous this is more complicated; a topic of ongoing research.
- The interaction of bootstrapping, function approximation and off policy learning is the biggest challenge for Q-Learning and is also a topic of research.
- Off-policy control does not create experience with the same distribution with which states would be encountered following the estimation policy. This can lead to divergence of the algorithm.
- For Q-Learning this is an issue because the experience is usually generated by an ϵ-greedy policy.

Reading and Homework

Reading

- Reinforcement Learning: A Survey, L.P Kaelbling et. al, Journal of Artificial Intelligence Research 4 (1996).
- https://webdocs.cs.ualberta.ca/ sutton/book/the-book.html (Sutton, 1998) Chapters 1, 2.1, 3, 6, 9.
- https://www.cis.upenn.edu/ mkearns/papers/rlexec.pdf
- Q-Learning-Based Financial Trading Systems with Applications, Corazza et. al.
- These notes follow closely the excellent book "Reinforcement Learning: an Introduction" by A. Sutton and R.S. Barro.