Scalable Machine Learning

Reinforcement Learning
What is reinforcement learning

A group of machine learning models that are used to train decision making “agents”
Different from many ML methods discussed so far because in its most basic form RL is non-differentiable.
The goals for this class

▶ Get familiar with the terminology.
▶ Recognize if a problem can be stated in terms of RL.
▶ Recognize which algorithms may be applicable

These slides are heavily based on these resources.

▶ https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292
▶ https://towardsdatascience.com/reinforcement-learning-an-introduction-to-the-concepts-app
▶ David Silver’s RL course http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
▶ Emma Brunskill’s RL course. http://web.stanford.edu/class/cs234/index.html has video
RL basics

The basic components of are:

- The agent’s **state** or observations of the environment. Can be any function of the total input received.
- Agent’s **actions**. What is the agent allowed to do?
- **Reward** Feedback from the environment. May be immediate or delayed.
RL examples: playing Atari games

Seminal paper: “Human-level control through deep reinforcement learning”.

- Input/observations. 4 frames of the game.

- Agent’s actions: game controls
- Reward: the score.
RL examples: BipedalWalker

The Bipedal walker is a physics simulator.

- State: hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with ground, and 10 lidar rangefinder measurements.
- Actions space: BipedalWalker has 2 legs. Each leg has 2 joints. The size of our action space is 4 which is torque applied on 4 joints.
- Reward: +300 for walking. -100 for falling.
Medical decision making: sepsis control


- **State:** a discretized patient health state based on clinical (EHR) data.
- **Actions space:** administration of intravenous fluids and vasopressors
- **Reward:** +100 for keeping the patient alive. -100 for failing.
There is a big difference between the BipedalWalker and the sepsis example. What is it?
There is a big difference between the BipedalWalker and the sepsis example. What is it?
One last example


- **State:** A SMILES string of length $\leq T$
- **Actions space:** A character in the SMILES alphabet.
- **Reward:**

![Diagram](image_url)

\[ r_{\text{log}}(P) = \begin{cases} 11, & \text{log} P \in (1, 4) \\ 1, & \text{otherwise} \end{cases} \]

\[ r(pIC_{50}) = \exp(-\frac{pIC_{50}}{3} + 3) \]
Some formal definitions

At each step $t$ the agent
- Executes action $A_t$
- Receives observation $O_t$
- Receives scalar reward $R_t$
History and State

- **history or trajectory** is the sequence of observations, actions, and rewards
  \[ H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t \]  

- What happens next depends on the history. The agent will select the actions and the environment will select observations and rewards.

- Formally, **state** is a function of the history
  \[ S_t = f (H_t) \]
Environment state vs. agent state

- The **environment state** $S^e_t$ is the environment's private representation. That is whatever data the environment uses to pick the next observation/reward.
- The **agent state** $S^a_t$ is the agent's internal representation. That is the information that the agent will use to pick its actions.
- $S^a_t$ can be any function of the history.

$$S^a_t = f(H^a_t) \quad (3)$$
An **information state** (a.k.a. Markov state) contains all useful information from the history. This must have the **Markov** property.

\[
P [S_{t+1} | S_t] = P [S_{t+1} | S_1, \ldots, S_t] \tag{4}
\]

- “The future is independent of the past given the present”
- Once the current state has been computed, the history may be discarded.
- The state is a sufficient statistic of the future.
Partially vs. fully observable environments

- Full observability: agent directly observes environment state.
  \[ S_t^a = S_t^e \]  

- This is known as a **Markov decision process (MDP)**

- Even if the environment is fully observable we can still have \( S_t^a \neq S_t^e \) because the current state doesn’t remember all of the history.

- In practice most RL applications are not true MDPs but all of the calculations/theory make this assumption.
Components of an RL agent

An RL agent may include one or more of these components:

- **Policy**: agents behavior function

- **Value function**: how good is each state and/or action. May be given but most often needs to be estimated.

- **Model**: agents representation of the environment
Policy

- A policy is a function from state to actions

- Can be deterministic: \( a = \pi(s) \)

- or stochastic: \( \pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \)
Value function

- In RL we maximize the **sum of future discounted rewards**

\[ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \]  

(6)

Where \( \gamma \in [0, 1] \) is the discount factor.

- Value function predicts the **expected** future discounted reward starting in state \( s \) and following a policy \( \pi \)

\[ v_\pi(s) = \mathbb{E}_\pi \left[ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s \right] \]  

(7)

- Can be used to evaluate the current state in terms of its future potential.

- It is a function of both policy and state.
A model of the environment gives as the distributions over rewards $r_{t+1}$ and next states $s_{t+1}$ from a specific state $s_t$ and action $a_t$.

- $T$ predicts the next state.
- $R$ predicts the next reward.

$$
T(s, s', a) = \mathbb{P} \left[ S_{t+1} = s' | S_t = s, A_t = a \right] \\
R(s, s', a) = \mathbb{E} \left[ R_{t+1} | S_t = s, A_t = a \right]
$$

- $T$ and $R$ may be given, if we are playing a game with known rules.
- $T$ and $R$ may be learned otherwise.
Categorizing RL agents

- **Value Based**
  - No Policy (Implicit)
  - Value Function

- **Policy Based**
  - Policy
  - No Value Function

- **Actor Critic**
  - Policy
  - Value Function

- **Model free**
  - Policy and/or Value Function
  - No model

- **Model Based**
  - Policy and/or Value Function
  - Model
RL Agent Taxonomy

- Model-Free
- Policy-Based
- Policy
- Model-Based
- Value-Based
- Value Function
- Actor
- Critic

Model
Problems in reinforcement learning

- Policy evaluation. This is a prediction task. Predict the expected reward from each state given some policy $\pi$.

- Policy optimization. Optimization task. Optimize $\pi$ such that the expected reward is maximized.

For a finite state space with a given model the value function and the optimization can be computed exactly.
The state space

The main questions in any RL application is how to represent the state/action space. Are the states and/or actions sufficiently discrete that we can:

- Fit them in a table.
- Have enough observations for each entry in our table in order to compute expectations.

For discrete or discretizable problems we can use “simple” iterative algorithms to update value and policy functions. There are no gradients and no deep learning. This is simple RL.
Policy-value iteration

In a model-based RL two simple update steps will converge to the correct solution (under mild assumptions)

Value iteration:

\[ V(s) := \sum_{s'} P_{\pi(s)}(s, s') \left( R_{\pi(s)}(s, s') + \gamma V(s') \right) \]  \hspace{1cm} (9)

If the policy doesn’t change the value iteration will converge to the true value function.

Policy iteration

\[ \pi(s) := \arg\max_a \left\{ \sum_{s'} P(s'|s, a) \left( R(s'|s, a) + \gamma V(s') \right) \right\} \]  \hspace{1cm} (10)

If the value function doesn’t change the policy iteration will converge to the true best policy for that value function.
Value functions vs $Q$ functions

- Value functions are defined for a given policy $\pi_i$.
- We can ask about the value function for the best policy. What is the maximum expected reward under any policy? For policy-value iteration the final value function is the maximum value function.
- In model-free RL the “maximum value” function is not sufficient. The policy also depends on is $P$ and $R$.

$$\pi(s) = \operatorname{argmax}_a \left\{ \sum_{s'} P(s'|s, a) \left( R(s'|s, a) + \gamma V(s') \right) \right\} \quad (11)$$

- Instead we can fold all of these into a single function: the $Q$ function.

$$Q : S \times A \to \mathbb{R} \quad (12)$$

This function evaluates the quality of state, action pairs in terms of maximum expected reward.
Q-learning

Q-learning is simple iterative update.

\[ Q^{new}(s_t, a_t) = \]

\[
(1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot \left( r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \right)
\]

(13)

- Lots of theory about the learning rate. In practice some constant like \( \alpha = 0.1 \) is used.
- Initialization matters. What happens if all the Q’s are initialized to 0 and all the rewards are positive?
Exploration and Exploitation

- Exploration finds more information about the environment
- Exploitation exploits known information to maximize reward
- Agent needs to explore as well as exploit

Example:
- Goal: have the best possible dinner.
- Exploitation: go the your favorite restaurant
- Exploration: Try a new one
When the state space is not discrete

- We may have too many states to represent them in a table. Possibly infinitely many of the states involve real values.

- Each encountered state may be quite rare but we would like to generalize across “similar” states.

- Solution: function approximations.
Function approximations

In RL we make use of various functions

- Value function is a function of state.
- Policy is a function of state.
- Q-function is a function of state, action pairs.

Instead of having a lookup table for these we define parametrized functions that take as input state features $f_s$ so that.

- Value function is $V(f_s)$
- Policy is a function of state $P(a|s) = T(f_s)$. Usually will involve a softmax.
- Q-function is a function of state, action pairs. Here there are options.

The functions can be simple linear or complex DNNs but they are differentiable and parametrized by a vector of parameters $\theta$!
Reinforcement Learning Objective: Maximize the expected reward following a parametrized policy

\[ J(\theta) = \mathbb{E}_{\pi}[r(\tau)] \] (14)

Where \( \tau \) is a trajectory (a sequence of states, actions, and rewards). We can do this using gradient ascent.

\[ \theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \] (15)

The problem: our objective function contains an expectation.
The policy gradient theorem

\[ \nabla \mathbb{E}_\pi [r(\tau)] = \nabla \int \pi(\tau) r(\tau) d\tau \]

\[ = \int \nabla \pi(\tau) r(\tau) d\tau \]

\[ = \int \pi(\tau) \nabla \log \pi(\tau) r(\tau) d\tau \]

\[ \nabla \mathbb{E}_\pi [r(\tau)] = \mathbb{E}_\pi [r(\tau) \nabla \log \pi(\tau)] \] (16)

The Policy Gradient Theorem: The derivative of the expected reward is the expectation of the product of the reward and gradient of the log of the policy \( \pi_\theta \).
Why is this useful

\[ \nabla \mathbb{E}_{\pi_\theta}[r(\tau)] = \mathbb{E}_{\pi_\theta}[r(\tau)\nabla \log \pi_\theta(\tau)] \]  

(17)

We can write out \(\pi_\theta(\tau)\).

\[ \pi_\theta(\tau) = P(s_0) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}, r_{t+1}|s_t, a_t) \]

model of the environment  

(18)
We can get rid of the unknown environment model!

\[
\log \pi_\theta(\tau) = \log \mathcal{P}(s_0) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \log p(s_{t+1}, r_{t+1}|s_t, a_t)
\]

\[
\nabla \log \pi_\theta(\tau) = \sum_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t)
\]

\[
\nabla \mathbb{E}_{\pi_\theta}[r(\tau)] = \mathbb{E}_{\pi_\theta} \left[ r(\tau) \left( \sum_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t) \right) \right]
\]

(19)

In practice the expectation is computed by “rolling out” trajectories with the current policy. The more the better.
Examples in detail: the sepsis model

- The data was discretized to 750 states and 25 possible actions.
- The state transition probability \( T(s'|s, a) \) was computed directly from a sample of the data. Actions that were never observed are not evaluated.
- The systems is discrete and there are no functions or derivatives.
- The authors applied Q-learning and policy iteration.
Some results
The model consists of 3 parts.

- A generative network $G$ that generates valid SMILES strings. This is aspirin \([\text{CC(O)OC1CCCCC1C(O)O}]\).
- A predictive network $P$ that predicts chemical properties and takes SMILES strings as input.
- $G$ and $P$ are trained separately so that $G$ produces valid strings and $P$ has some accuracy.
- $G$ can be viewed as a policy approximation model and thus can be trained further with reinforcement learning.
  - The actions are adding a character to the SMILES string
  - The reward is computed only at the end by $P$. The intermediate rewards are 0.
  - Can be trained with REINFORCE via policy gradient.
Some results

A. Distribution of melting temperature ($T_m$, °C) for minimized, maximized, and baseline conditions.

B. Distribution of JAK2 inhibition (pIC50).

C. Distribution of partition coefficient (logP).

D. Distribution of number of benzene rings.

E. Distribution of number of substituents.