(Deep) Reinforcement Learning

Pattern recognition

May 26, 2020

Outline

- Machine learning paradigms
- (classical) Reinforcement learning
  - problem formulation
  - common approaches
- Deep reinforcement learning
  - methods and architectures
  - challenges

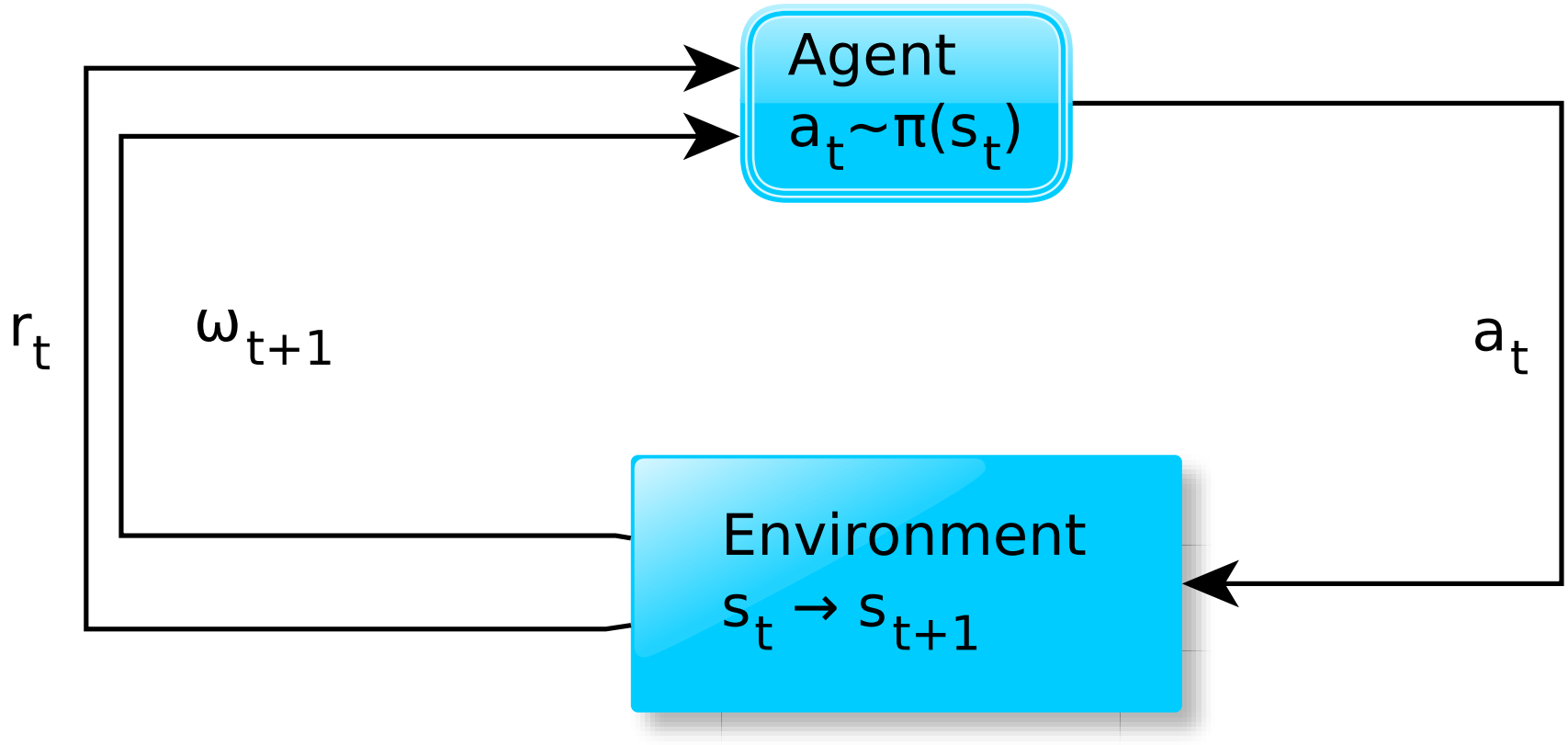
Machine Learning Paradigms

- Supervised learning
  - learn mapping (function) $f : X \to Y$, given pairs $(x \in X, y \in Y) \in D$
  - function approximation
  - classification, regression
- Unsupervised learning
  - learn $g(X)$ – a more compact representation of X (dimensionality reduction, compression)
  - learn the distribution of X – generative models
  - clustering, feature extraction
- Reinforcement learning
  - learn mapping $f : X \to Y$, given pairs $(x \in X, z \in Z)$ – what is Z?
  - (mostly) decision-making problems

Reinforcement Learning (RL)

- RL comes from behavioral psychology
  - an individual (biological agent) explores its environment and learns by interacting with it
- In RL, an *artificial agent* simulates this behavior
  - **interacts with an environment**
  - gathers experience (**rewards**/punishments)
  - tries to optimize certain task/objectives, implicitly given via rewards
- RL agent
  - starts in state $s_0 \in S$ associated with observation $\omega_0 \in \Omega$
  - At each time step t, the agent takes action $a_t \in A$ which results in the environment changing from state s_t to s_{t+1}, modifying the observation to ω_{t+1}
  - as a consequence of entering a state, the agent receives a reward $r_t \in R$
  - goal is to learn optimal control strategy: policy $\pi(s_t) \rightarrow a_t$

RL Agent


Agent
$a_t \sim \pi(s_t)$
r_t
ω_{t+1}
a_t
Environment
$s_t \rightarrow s_{t+1}$

Markov Decision Process (MDP)

- discrete time stochastic control process defined by a tuple (S, A, T, R) and assuming Markovian property

S is the **state space** – all possible states s

- may include distribution of starting states s_0
- and a set of terminal/absorbing states

A is the **action space** - all valid actions

- action and state space can be discrete or continuous

$T : S \times A \times S \to [0, 1]$ **transition probabilities** or transition dynamics (the rules of the environment)

- $p = T(s_t, a_t, s_{t+1}) = \mathcal{P}(s_{t+1} \mid s_t, a_t)$ is the probability of transitioning into state s' by taking action a when in the state s
- deterministic ($p \in \{0, 1\}$) or probabilistic ($p \in (0, 1)$)

$R : S \times A \times S \to \mathbb{R}$ is the instant **reward** function

- $r = R(s_t, a_t, s_{t+1}) < R_{max}$ is the reward of transitioning from s to s' by taking action a

RL set in MDP

- A simplified formulation of RL agent:
  $[\omega_0 \in \Omega \sim s_0 \in S] \rightarrow ... \rightarrow [\omega_t \in \Omega \sim s_t \in S] + a_t \in A \rightarrow$
  $[\omega_{t+1} \in \Omega \sim s_{t+1} \in S] \Rightarrow r_{t+1} \in R$

  ω is an observation (measurement)
- In **regular MDP**: $\omega = s, \forall \omega \in \Omega \wedge \forall s \in S$

- The goal of RL is to find optimal policy:

$\pi(s_t) \rightarrow A$ is the **policy** – decision "function" or strategy that determines the action to be taken in a given state

π^* is the optimal policy that always chooses actions that maximize the total reward

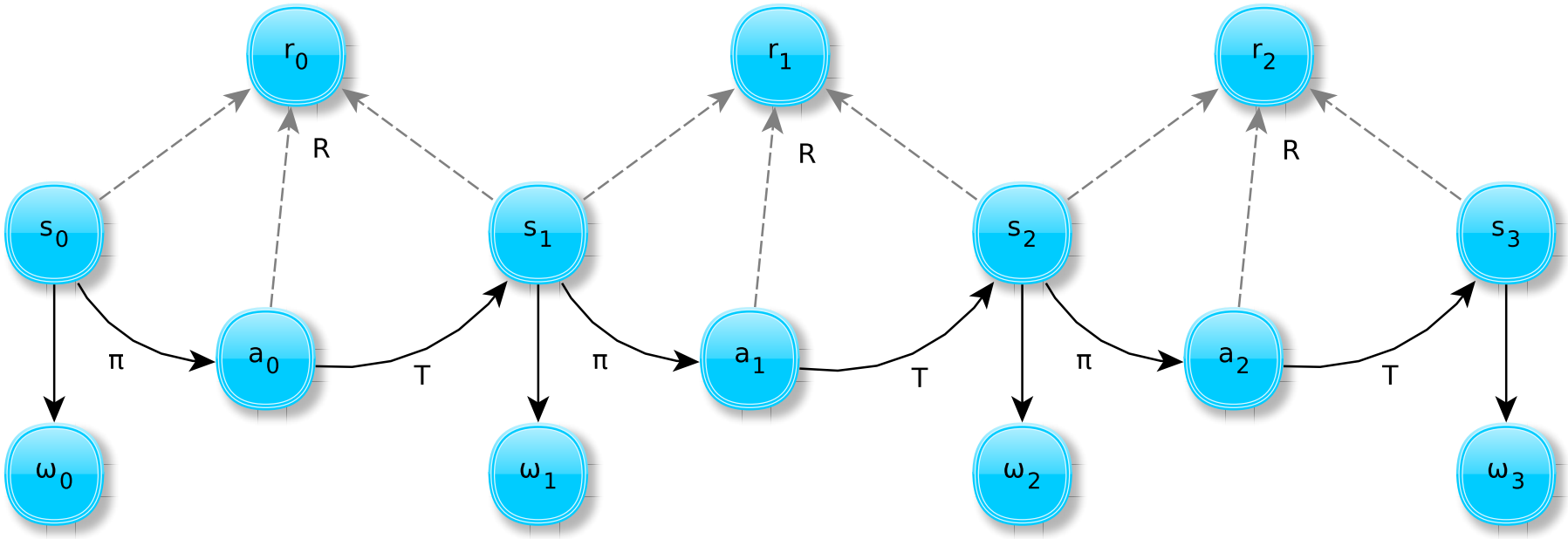
θ are the parameters of the policy, i.e. the policy can be written as $\pi(s_t \mid \theta) \rightarrow A$

- in case of deep RL, a policy network is used (and θ are the parameters of the network)

POMDP

- **Partially Observable MDP** (POMDP):
- not all properties of the environment are observable (e.g., measurement error)
- better approximation of the real world
- probability distribution of observations/measurements:$\mathcal{P}\left(\omega_{t+1} \mid s_{t+1}, a_t\right)$
- in robotics: measurement model
- in RL, *belief over states* is updated from the current state belief and the performed action

POMDP



How to find the optimal policy?

Example world:

			s_+
	X		s_-

where s_+ is a terminal state with reward $= +1$ and s_- is a terminal state with reward $= -1$,

Counting instant reward:

0	0	0	+1
0	X	0	-1
0	0	0	0

Credit assignment problem: the reward is sparse, how to determine "partial rewards" for the other states?

Expected reward

- Expected reward (V-**value** or *utility* of a state):
  $V^{\pi}(s) = \mathbb{E}[\mathfrak{R} \mid \mathfrak{s}, \pi] = \mathbb{E}\left[\sum_{k=0}^{T} \gamma^k r_{t+k} \mid s_t = s, \pi\right]$,
  where $r_t = \mathop{\mathbb{E}}_{a \sim \pi} R(s_t, a, s_{t+1})$ = reward in the current state; T is horizon (could be infinite)

  $V^{\pi}(s)$ is expected reward (return) starting in the state s and following π until the end of the *epoch* (*policy roll-out*)

- *Instant* reward vs *delayed* reward:
  $\sum_{k=0}^{T} \gamma^k r_{t+k} = r_t + \sum_{k=1}^{T} \gamma^k r_{t+k}$
  where $\gamma \in [0, 1)$ is the **discount factor** (for delayed reward)
  - lower values $\rightarrow$ more emphasis on instant reward
  - higher values $\rightarrow$ more emphasis on delayed reward
  - discount factor makes the delayed reward meaningful (i.e. the infinite sum converges)
  - not required in non-stationary/finite environments
- **Optimal expected reward**: $V^*(s) = \max_{\pi \in \Pi} V^{\pi}(s)$

Expected reward

Using discounted reward ($\gamma = 0.9$):

0.73	0.81	0.9	+1
0.66	X	0.81	-1
0.6	0.66	0.73	0.66

Taxonomy of RL methods

- **model-free** methods
  - **value-based** methods $\rightarrow$ determining the policy from state values
    - V-**value/utility-based** agents – learns utility of states
    - Q-**learning** – learning policy based on utility/quality of state-action pairs
  - **policy-based** methods $\rightarrow$ direct policy learning (reflex agent)
- **model-based** methods $\rightarrow$ learning of the model of the environment (T, R)

Value-based methods

- Learning based on estimates of the value/utility of (being in) a given state
- Given $V^{\pi}(s)$, compute optimal policy:
  π^* such that $a = \pi^*(s)$ maximizes utility for all states:
  $V^*(s) = \max_{\pi \in \Pi} V^{\pi}(s), \forall s \in S$
  - remember that $R(s_t, a_t, s_{t+1})$, where $a_t = \pi(s_t)$ and $\mathcal{P}(s_{t+1} \mid s_t, a_t) = T(s_t, a_t, s_{t+1})$
- *Problem*: T (**transition probabilities**) are usually **not known** (would require a model of the environment)
- *Solution*: Instead of learning the utility of a state, learn directly the **utility of an *action in a given state***
- This is called Q-**value** function (or sometimes Q-function; "quality"):
  $Q^{\pi}(s, a)$
  - state-action value pairs

Q-value function

- State-action quality (value) function:
  $$Q^{\pi}(s,a) = \mathbb{E}[\mathfrak{R} \mid \mathfrak{s}, \mathfrak{a}, \pi] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right]$$
- Using the Markovian property and dynamic programming, Q can be rewritten as *Bellman equation*(s):
  $$Q^{\pi}(s_t, a_t) = \begin{cases} r_t & \textit{iff } s_t \textit{ is a terminal state} \\ \mathbb{E}\left[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1} \sim \pi(s_{t+1}))\right] & \textit{otherwise} \end{cases}$$
- **Optimal Q-value**: $Q^*(s,a) = \max_{\pi \in \Pi} Q^{\pi}(s,a)$
- **Optimal policy**: $\pi^* = \arg\max_{a \in A} Q^*(s,a), \forall s \in S$

Learning the Q-value function

- Learning the Q-value function – how to find the optimal Q-values?
- Two main approaches:
  - sampling
  - bootstrapping
- V-values can be learned similarly
- For off-policy methods: ϵ-greedy policy (to experience sub-optimal states)

Learning the Q-value function: sampling

- Monte Carlo method
  - initial estimates for the Q-values
  - adjust the initial estimates by randomly sampling and averaging policy roll-outs at every state
- Cannot be applied in large (or infinite) state-action spaces (roll-out computation is intractable)
- Can be combined with bootstrapping (estimates for the roll-out)

Learning the Q-value function: bootstrapping

- Learning by bootstrapping:
  - initialize Q-values
  - update values according to rule (**temporal difference learning**):

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha [Y - Q^{\pi}(s_t, a_t)], \quad (1)$$

    where Y is a *target* value, α is learning rate, and $[Y_k - Q^{\pi}(s_t, a_t)]$ is called **temporal difference** error
  - the target Y can be defined as:
    1. $Y = r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})$: *on-policy learning* (SARSA algorithm)
    2. $Y = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$: *off-policy learning* – this is called Q-**learning**
- The Q-value function can be described by a set of **parameters** θ: $Q^{\pi}(s, a, \theta_k)$; the parameters are updated at every iteration
- Update rule can be derived from Eq. (1):

$$\theta_{k+1} \leftarrow \theta_k + \alpha [Y_k - Q^{\pi}(s_t, a_t, \theta_k)] \nabla_{\theta_k} Q^{\pi}(s_t, a_t, \theta_k) \quad (2)$$

  - *fitted* Q-learning

V & Q Recap

- In simple words:
  - $V(s)$ is the value of a state s, i.e. how good is it to be in the state s
    - how much reward can we (in the best case) get, starting in this state
  - $Q(s, a)$ is the "quality" of an action a whilst being in a specific state s – how good is to take the action a, given the current state is s
- On-policy: V & Q (more precisely, rewards from which V and Q can be computed) are computed using the current policy (online learning)
  - the quality of the current policy can greatly influence the learning convergence (quality of the training "dataset" depends on the policy)
- Off-policy: V & Q are computed without the current policy
  - usually prerecorded data (offline learning)
  - approximating optimal policy: choosing maximal action in a given state (computed from previous experience)

Direct policy search

- For large or continuous action spaces, computing optimal Q may be impossible (or at least impractical)
- Instead, learn the policy $\pi(s \mid \theta)$ directly by finding the optimal parameters parameters $\theta^* = \arg\max_{\theta} \mathbb{E}[\mathfrak{R} \mid s, \theta], \forall s \in S$
- Approaches:
  - **policy gradient methods**
    - idea: if an action turns out to be better than expected, increase the likelihood of choosing it (and vice versa)
    - necessary to compute estimated returns of the roll-out of the current policy
    - deterministic (requires model of the environment) or stochastic approximation (sampling)
    - stochastic: gradients cannot be propagated – needs gradient estimator (REINFORCE algorithm)
  - **actor-critic methods**

    actor learns the policy

    critic feedback – learns the value function
  - evolutionary algorithms
    - useful when the optimized function is not differentiable

Model-based approaches

- Previous approaches assumes no knowledge of the model of the environment (reward and transition functions)
- Model of the environment is usually learned but can be given in the form of some prior knowledge (e.g. rules of a game)
  - in complex environments, some heuristics can be given
- Model is used to predict/simulate future steps – faster learning without direct interaction with the environment
- Can be (usually is) combined with the other approaches
- Cons:
  - specific to a given problem
  - requires (good) knowledge of the optimal strategy in the given setup

Deep Reinforcement Learning

- RL seems good at solving complex task, why do we need *deep* RL?
- *Classical* RL algorithms share the same issues as all ML methods:
  - computational & memory complexity
  - also sample complexity but this is often more prevalent in deep learning
  - problems in environments with many states and actions (e.g., not possible to have a look-up table for the Q-values)
- **Deep Reinforcement Learning** (DRL)
  - uses the "universal function approximator" property of ANNs to approximate optimal policy/Q-value function
    - environments with many states could be learned
  - learns to compress high-dimensional input space into a low-dimensional representation ($\sim$features)
    - inputs such as images could be used
  - trade-off for standard DL problems (approximate solution, convergence)

Architectures for DRL

- Value-based
  - DQN
  - Double DQN
  - Dueling DQN
- Policy Gradients
- Actor-critic

Deep Q-networks

- Deep Q-learning using **deep Q-networks** (DQN)
- Q-value function approximated by a neural network (FCN/CNN)
- Therefore, parameters θ of the function $Q(s, a, \theta)$ are the parameters of the neural network
- Loss function can be similar to traditional Q-learning:

$$L(\theta_k) = (Q(s_t, a_t, \theta_k) - Y_k)^2,$$

where $Y_k = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta_k)$

- Modifications
  - update parameters θ' only every $C \in \mathbb{N}$ iterations (higher stability)
  - replay memory stores a set of $< s_t, a_t, r_t, s_{t+1} >$ tuples, i.e. a batch of data – updates on minibatches (lower variance)
  - weight clipping (limits the scale of error derivatives but introduces bias)
  - other "traditional" modifications from ANN are used

DQN variants

- Double DQN
  - *max* operation in the target overestimates expected reward – selection and evaluation of action is computed the same way
  - target in update rule is replaced with

$$Y_k^{DDQN} = r_t + \gamma Q\left(s_{t+1}, \arg\max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}, \theta_k\right), \theta'_k\right)$$

- Dueling DQN
  - not all state-action values are know (e.g. some actions in some states are irrelevant)
  - solution: use two value streams and combine them:
    - estimate V-value of a state separately
    - estimate $A(s,a) = Q(s,a) - V(s)$, where A is the **advantage function**
    - final output:

$$Q(s,a,\theta,\alpha,\beta) = V(s,\alpha) + A(s,a,\beta)$$

Other DRL approaches

- DQN suffers from the same problems as classical Q-learning – large or continuous action spaces
- Policy optimization
  - actor-critic approach:
    - actor and critic represented by two networks
- Model-based methods
  - learning the model (reward and transition functions) via ANN
- In general, similar approaches as in classical RL but functions are approximates using ANNs

Challenges

- Temporal credit assignment problem
- Exploration vs. exploitation
- Curse of dimensionality
- Partially observable environments
- Non-stationary environments

Recommended reading

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G. and Pineau, J., 2018. *An introduction to deep reinforcement learning*. Foundations and Trends in Machine Learning, 11(3-4), pp.219-354.

- A comprehensive introduction to Deep RL and an overview of (nearly) current techniques and applications

- More current approaches/modifications:
  - Trust region policy optimization (TRPO)
  - Proximal policy optimization (PPO)
  - Curiosity driven exploration
  - Hindsight experience replay (HER)
  - Soft actor-critic (SAC)

Thank you for your attention