Queen Mary
University of London

Artificial Intelligence in Games

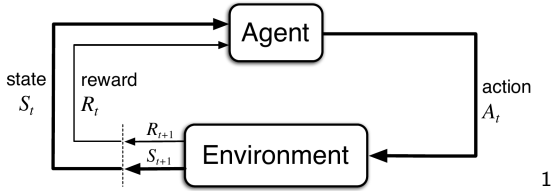
Reinforcement Learning

Paulo Rauber
School of Electronic Engineering and Computer Science

Intelligence

- Intelligence measures the ability of an agent to achieve goals in a wide range of environments [Legg, 2008]
- Intelligence requires activity and flexibility
- This definition is incompatible with colloquial usage
  - Enables formal definition given additional assumptions
  - Enables researchers to understand their role
- Highly intelligent agents according to this definition are useful

Reinforcement learning



[1]

- Goals are defined through reward mechanisms

[1] Image from [Sutton and Barto, 2018]

Reinforcement learning

- An agent interacts with an environment during a sequence of discrete time steps
- At each time step $t \geq 0$, the agent receives some representation of the state $s_t \in \mathcal{S}$
- The agent then selects an action $a_t \in \mathcal{A}(s_t)$
- One time step later, the agent receives a reward $r_{t+1} \in \mathbb{R}$ and a new state $s_{t+1} \in \mathcal{S}$
- A policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a function such that $\pi(s, a)$ represents the probability that $a_t = a$ given that $s_t = s$

Discounted return

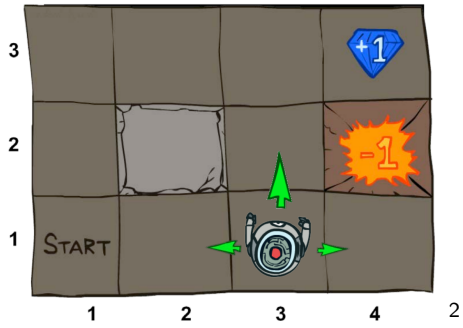
- The discounted return u_t after time step t is given by

$$u_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k},$$

  where $0 \leq \gamma < 1$ is the discount factor
- A reward received k time steps into the future is only worth γ^{k-1} times what it would be worth if it were received on the next step
- If necessary, a state can transition only to itself and yield no rewards
- The objective of the agent is to maximize the discounted return

Example: grid world

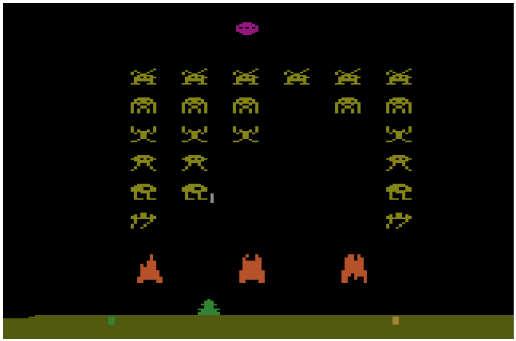
- States $\mathcal{S} = \{1, 2, \ldots, 12\}$, actions $\mathcal{A} = \{1, 2, 3, 4\}$
- Reward 1 on action at goal, reward -1 on action at trap, and reward 0 on action at other states
- Actions at goal and trap transition to an absorbing state
- Discount factor $\gamma = 0.9$



[2]Image from [Klein et al., 2019]

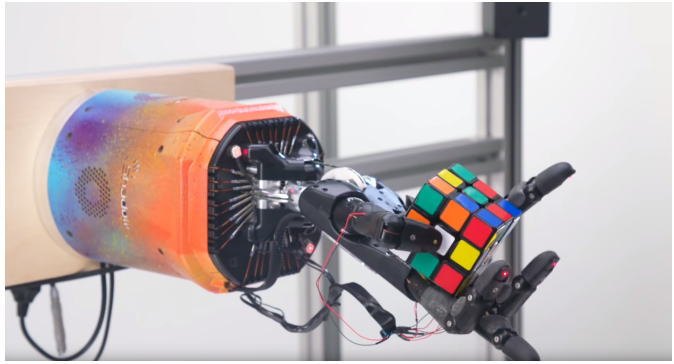
Applications: games

- Atari [Mnih et al., 2015], Dota 2 [Brockman et al., 2019a], chess and Go [Silver et al., 2018]



Applications: robotics

- Rubik's cube manipulation [Brockman et al., 2019b]



Applications: others

- Practical: logistics, finance, and marketing
- Theoretical: every task with a computable description can be formulated as a reinforcement learning problem [Hutter, 2004]
- Curse of generality: every well-defined problem is a reinforcement learning problem, but most reinforcement learning problems cannot be solved efficiently

Markov decision process

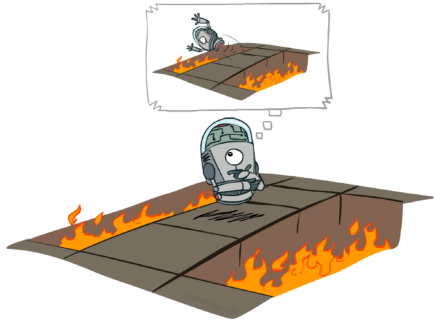
- A state that summarizes the entire past with all that is relevant for decision making has the Markov property
- In a Markov decision process, for any sequence of states, action and rewards $s_t, a_t, r_t, \ldots r_1, s_0, a_0$ (history) and all $s' \in \mathcal{S}, r' \in \mathbb{R}$,

$$P(S_{t+1} = s', R_{t+1} = r' \mid s_t, a_t, r_t, \ldots, r_1, s_0, a_0) = P(S_{t+1} = s', R_{t+1} = r' \mid s_t, a_t)$$

- The conditional joint probability distribution over states and rewards on the right side defines the one-step dynamics of the problem

Environment model

- Probability distribution over next state given each state and action
- Expected reward given each state and action



[3]

[3] Image from [Klein et al., 2019]

$\mathcal{P}^a_{ss'}$

- The probability $\mathcal{P}^a_{ss'}$ of transitioning from state s to state s' given action a is given by

$$\mathcal{P}^a_{ss'} = P(S_{t+1} = s' \mid S_t = s, A_t = a) = \sum_{r'} P(S_{t+1} = s', R_{t+1} = r' \mid S_t = s, A_t = a)$$

- Note that $\mathcal{P}^a_{ss'}$ is independent of the current time step

$\mathcal{R}^a_{ss'}$

- The expected reward on transitioning from state s to state s' given the action a is given by

$$\begin{aligned}\mathcal{R}^a_{ss'} &= \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'] \\ &= \frac{1}{\mathcal{P}^a_{ss'}} \sum_{r'} r' P(S_{t+1} = s', R_{t+1} = r' \mid A_t = a, S_t = s)\end{aligned}$$

- Note that $\mathcal{R}^a_{ss'}$ is independent of the current time step

Value function V^π

- The value $V^\pi(s)$ of a state $s \in \mathcal{S}$ is the expected (discounted) return of starting in state s and following the policy π

$$V^\pi(s) = \mathbb{E}_\pi[U_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \tag{1}$$

Action value function Q^π

- The value $Q^\pi(s, a)$ of taking an action $a \in \mathcal{A}$ when in state $s \in \mathcal{S}$ and afterwards following the policy π is given by

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi[U_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \end{aligned}$$

Recursivity of the value function

Theorem (Recursivity of the value function)

For any policy π and state $s \in \mathcal{S}$,

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')]. \tag{2}$$

Notation

- We denote random variables by upper case letters and assignments to these variables by corresponding lower case letters
- We omit the subscript that typically relates a probability function to random variables when there is no risk of ambiguity
- For example, let X and Y be discrete random variables. In the same context, we will let $p(x|y)$ denote $P(X = x|Y = y)$ and $p(y|x)$ denote $P(Y = y|X = x)$
- This notation where the arguments select between different probability functions is standard in machine learning

Recursivity of the value function

Proof.

Note that Equation 1 can be rewritten as

$$V^{\pi}(s) = \lim_{T \to \infty} \sum_{r_{t+1:T+t+1}} p(r_{t+1:T+t+1} \mid S_t = s) \sum_{k=0}^{T} \gamma^k r_{t+k+1},$$

where the dependency on the policy π becomes implicit. By marginalization,

$$V^{\pi}(s) = \lim_{T \to \infty} \sum_{r_{t+1:T+t+1}} \left[\sum_{a_t} \sum_{s_{t+1}} p(r_{t+1:T+t+1}, a_t, s_{t+1} \mid S_t = s) \right] \sum_{k=0}^{T} \gamma^k r_{t+k+1}.$$

Recursivity of the value function

Proof. (cont.)

By the chain rule of probability,

$$V^{\pi}(s) = \lim_{T\to\infty} \sum_{r_{t+1:T+t+1}} \sum_{a_t} \sum_{s_{t+1}} p(a_t \mid S_t = s) p(s_{t+1} \mid S_t = s, a_t) p(r_{t+1:T+t+1} \mid S_t = s, a_t, s_{t+1}) \sum_{k=0}^{T} \gamma^k r_{t+k+1}.$$

By the distributive property and reordering the three outermost summations,

$$V^{\pi}(s) = \sum_{a_t} p(a_t \mid S_t = s) \sum_{s_{t+1}} p(s_{t+1} \mid S_t = s, a_t) \lim_{T\to\infty} \sum_{r_{t+1:T+t+1}} p(r_{t+1:T+t+1} \mid S_t = s, a_t, s_{t+1}) \sum_{k=0}^{T} \gamma^k r_{t+k+1}. \quad (3)$$

Recursivity of the value function

Proof. (cont.)

Let E denote the limit in the previous equation, such that

$$E = \lim_{T\to\infty} \sum_{r_{t+1:T+t+1}} p(r_{t+1:T+t+1} \mid S_t = s, a_t, s_{t+1}) \sum_{k=0}^{T} \gamma^k r_{t+k+1}.$$

By isolating the first term in the innermost summation,

$$E = \lim_{T\to\infty} \sum_{r_{t+1:T+t+1}} p(r_{t+1:T+t+1} \mid S_t = s, a_t, s_{t+1}) \left[r_{t+1} + \sum_{k=1}^{T} \gamma^k r_{t+k+1} \right].$$

Recursivity of the value function

Proof. (cont.)

By the linearity of expectation, $E = E_1 + E_2$, where

$$\begin{aligned} E_1 &= \lim_{T\to\infty} \sum_{r_{t+1:T+t+1}} p(r_{t+1:T+t+1} \mid S_t = s, a_t, s_{t+1}) r_{t+1} \\ &= \mathbb{E}_\pi \left[R_{t+1} \mid S_t = s, a_t, s_{t+1} \right] = \mathcal{R}^{a_t}_{ss_{t+1}}, \end{aligned}$$

and

$$E_2 = \lim_{T\to\infty} \sum_{r_{t+1:T+t+1}} p(r_{t+1:T+t+1} \mid S_t = s, a_t, s_{t+1}) \sum_{k=1}^{T} \gamma^k r_{t+k+1}.$$

Recursivity of the value function

Proof. (cont.)

By changing the indices in the innermost summation,

$$E_2 = \lim_{T\to\infty} \sum_{r_{t+1:T+t+1}} p(r_{t+1:T+t+1} \mid S_t = s, a_t, s_{t+1}) \sum_{k=0}^{T-1} \gamma^{k+1} r_{t+k+2}.$$

By moving a constant factor of γ outside of the innermost summation,

$$E_2 = \gamma \lim_{T\to\infty} \sum_{r_{t+1:T+t+1}} p(r_{t+1:T+t+1} \mid S_t = s, a_t, s_{t+1}) \sum_{k=0}^{T-1} \gamma^{k} r_{t+k+2}.$$

Recursivity of the value function

Proof. (cont.)

Because $R_{t+2:T+t+1} \perp\!\!\!\perp S_t, A_t \mid S_{t+1}$ due to the Markov property,

$$E_2 = \gamma \lim_{T\to\infty} \mathbb{E}_\pi \left[\sum_{k=0}^{T-1} \gamma^k R_{t+k+2} \mid s_{t+1} \right] = \gamma V^\pi(s_{t+1}).$$

Returning to Equation 3,

$$V^\pi(s) = \sum_{a_t} p(a_t \mid S_t = s) \sum_{s_{t+1}} p(s_{t+1} \mid S_t = s, a_t) \left[\mathcal{R}^{a_t}_{ss_{t+1}} + \gamma V^\pi(s_{t+1}) \right].$$

By making the dependency on π explicit and renaming variables,

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right].$$

□

Recursivity of the action value function

Theorem (Recursivity of the action value function)

For any policy π, state $s \in \mathcal{S}$, and action $a \in \mathcal{S}$,

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'}[\mathcal{R}^{a}_{ss'} + \gamma \sum_{a'} \pi(s',a')Q^{\pi}(s',a')].$$

Relationship between V^π and Q^π

Theorem (Relationship between V^π and Q^π)

For any policy π, state $s \in \mathcal{S}$, and action $a \in \mathcal{S}$,

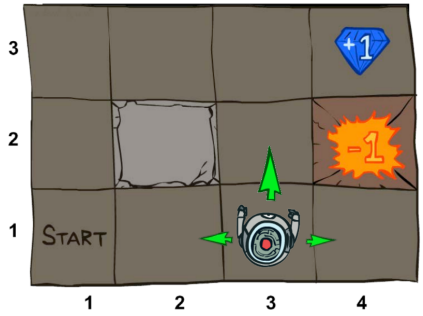
$$V^\pi(s) = \sum_a \pi(s,a) Q^\pi(s,a),$$

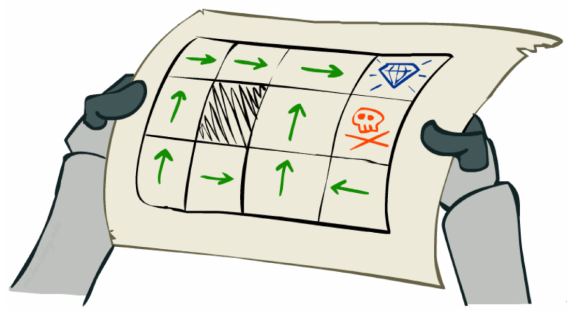
and

$$Q^\pi(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')].$$

Optimal policies

- Let $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$.
- A policy π^* is optimal if $\pi^* \geq \pi$ for any policy π
- An optimal policy always exists, but is not necessarily unique





[4]

[4] Image from [Klein et al., 2019]

Optimal value functions

Theorem (Bellman optimality equations)

For any action $a \in \mathcal{A}$ and state $s \in \mathcal{S}$, the optimal state value function V^* and the optimal action value function Q^* are given by

$$V^*(s) \triangleq \max_\pi V^\pi(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')],$$

and

$$Q^*(s,a) \triangleq \max_\pi Q^\pi(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a')].$$

Reinforcement learning algorithms

- Reinforcement learning algorithms aim to find an optimal policy π^* for a given environment
- For any state $s \in \mathcal{S}$, an optimal policy π^* can be found given either V^* or Q^*
- In the case of Q^*, for any $s \in \mathcal{S}$, it suffices to choose an a such that $Q^*(s, a)$ is maximal
- In the case of V^*, for any $s \in \mathcal{S}$, it suffices to choose one of the actions a that maximizes the right hand side of the Bellman optimality equation

Additional reading

- **Notes on reinforcement learning (Sections 1 and 3) [Rauber, 2015b]**
- **Reinforcement learning: an introduction [Sutton and Barto, 2018]**
- Algorithms for reinforcement learning [Szepesvári, 2010]
- UCL course on reinforcement learning [Silver, 2015]
- UC Berkeley course on deep reinforcement learning [Levine, 2018]
- Stanford course on reinforcement learning [Ng, 2008]
- Stanford course on reinforcement learning [Brunskill, 2019]
- Deep reinforcement learning bootcamp [Abbeel et al., 2017]

Additional reading: fundamentals

- Notes on calculus [Rauber, 2015a]
- Notes on linear algebra [Rauber, 2014]
- Notes on machine learning (Section 2.1: probability theory) [Rauber, 2016]

Reinforcement learning

Additional reading

References

References I


Abbeel, P. et al. (2017).
Deep reinforcement learning bootcamp.
https://sites.google.com/view/deep-rl-bootcamp/lectures.


Brockman, G. et al. (2019a).
OpenAI five.
https://openai.com/five/.


Brockman, G. et al. (2019b).
Solving rubik's cube with a robot hand.
https://openai.com/blog/solving-rubiks-cube/.


Brunskill, E. (2019).
CS234: Reinforcement learning.
http://web.stanford.edu/class/cs234/schedule.html.

References II


Hutter, M. (2004).
Universal artificial intelligence: Sequential decisions based on algorithmic probability.
Springer Science & Business Media.


Klein, D., Abbeel, P., and Dragan, A. (2019).
CS 188: Introduction to artificial intelligence.
https://inst.eecs.berkeley.edu/~cs188/fa19/.


Legg, S. (2008).
Machine super intelligence.
PhD thesis, Università della Svizzera italiana.


Levine, S. (2018).
Deep reinforcement learning.
http://rail.eecs.berkeley.edu/deeprlcourse-fa18/.

References III


Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015).
Human-level control through deep reinforcement learning.
Nature, 518(7540):529.


Ng, A. (2008).
CS 229: Machine learning.
https://www.youtube.com/watch?v=RtxI449ZjSc.


Rauber, P. (2014).
Notes on linear algebra.
http://paulorauber.com/work.html.


Rauber, P. (2015a).
Notes on calculus.
http://paulorauber.com/work.html.

References IV


Rauber, P. (2015b).
Notes on reinforcement learning.
http://paulorauber.com/work.html.


Rauber, P. (2016).
Notes on machine learning.
http://paulorauber.com/notes/machine_learning.pdf.


Silver, D. (2015).
UCL course on reinforcement learning.
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.


Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018).
A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play.
Science, 362(6419):1140–1144.

References V


Sutton, R. S. and Barto, A. G. (2018).
Reinforcement Learning: An Introduction.
The MIT Press, second edition.


Szepesvári, C. (2010).
Algorithms for reinforcement learning.
Synthesis lectures on artificial intelligence and machine learning, 4(1):1–103.

Queen Mary
University of London

Artificial Intelligence in Games

Tabular Model-Based Algorithms

Paulo Rauber
School of Electronic Engineering and Computer Science

Dynamic programming

- Dynamic programming algorithms can be used to compute optimal policies given a perfect model of the environment (one-step dynamics) when the sets of states and actions are finite
- The problem of finding the optimal value functions has optimal substructure: it can be solved by breaking it into sub-problems and then recursively finding the solutions to the sub-problems

Policy evaluation

- Policy evaluation is an iterative algorithm to compute the state value function V^π for an arbitrary policy π
- It relies on creating a sequence $V_0, V_1, \ldots$ of estimates of V^π given by

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')],$$

  for all $s \in \mathcal{S}$
- The initial value estimate V_0 can be arbitrary
- The sequence $V_0(s), V_1(s), \ldots$ converges to $V^\pi(s)$ for all $s \in \mathcal{S}$

In-place policy evaluation

- Instead of computing the new estimate V_{k+1} using the old estimate V_k, it is also possible to change a single estimate V in-place using

$$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')],$$

  for all $s \in \mathcal{S}$
- The estimate $V(s)$ also converges to $V^\pi(s)$ for all $s \in \mathcal{S}$ after repeated passes over all states

In-place policy evaluation

Algorithm 1 Iterative policy evaluation (in-place)

Input: policy π, one-step dynamics functions $\mathcal{P}$ and $\mathcal{R}$, discount factor γ, tolerance θ.
Output: Value function $V = V^\pi$ when $\theta \to 0$.

for each $s \in \mathcal{S}$ **do**
  $V(s) \leftarrow 0$
end for
repeat
  $\Delta \leftarrow 0$
  for each $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$

Convergence of iterative policy evaluation

Definition (Norm)

Consider a vector space Z over a field F. A function $||\cdot|| : Z \to [0, \infty)$ is a norm if

$$
\begin{aligned}
||\mathbf{u} + \mathbf{v}|| &\leq ||\mathbf{u}|| + ||\mathbf{v}||, \\
||a\mathbf{v}|| &= |a|||\mathbf{v}||, \\
||\mathbf{v}|| = 0 &\implies \mathbf{v} = \mathbf{0},
\end{aligned}
$$

for all $\mathbf{u}$, $\mathbf{v} \in Z$ and $a \in F$.

Convergence of iterative policy evaluation

Definition (Euclidean norm)

The Euclidean norm $||\cdot||_2 : \mathbb{R}^d \to [0, \infty)$ is given by

$$||\mathbf{v}||_2 = \sqrt{\sum_i v_i^2}.$$

Definition (Maximum norm)

The maximum norm $||\cdot||_\infty : \mathbb{R}^d \to [0, \infty)$ is given by

$$||\mathbf{v}||_\infty = \max_i |v_i|.$$

Convergence of iterative policy evaluation

Definition (Convergence of a sequence)

A sequence $(\mathbf{v}_n)_{n\geq 0} = \mathbf{v}_0, \mathbf{v}_1, \ldots$ is said to converge to a vector $\mathbf{v}$ in the norm $||\cdot||$, denoted $\mathbf{v}_n \underset{||\cdot||}{\to} \mathbf{v}$, if

$$\lim_{n\to\infty} ||\mathbf{v}_n - \mathbf{v}|| = 0.$$

Convergence of iterative policy evaluation

Definition (Bellman operator)

Consider a reinforcement learning task with states $\mathcal{S} = \{1, 2, \ldots, |\mathcal{S}|\}$, actions $\mathcal{A} = \{1, 2, \ldots, |\mathcal{A}|\}$, and discount factor $\gamma < 1$. For any vector $\mathbf{v} \in \mathbb{R}^{|\mathcal{S}|}$ and state $s \in \mathcal{S}$, the Bellman operator $T^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ for the policy π is given by

$$T^\pi(\mathbf{v})_s = \sum_a \pi(s, a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma v_{s'}\right].$$

Convergence of iterative policy evaluation

Theorem (Convergence of iterative policy evaluation)

Consider the Bellman operator $T^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ *for the policy* π. *Given an arbitrary* $\mathbf{v}_0 \in \mathbb{R}^{|\mathcal{S}|}$, *consider also the sequence* $(\mathbf{v}_n)_{n \geq 0}$ *where* $\mathbf{v}_{k+1} = T^\pi(\mathbf{v}_k)$. *Finally, consider the vector* $\mathbf{v}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ *such that* $v_s^\pi = V^\pi(s)$, *for any* $s \in \mathcal{S}$. *For any* $n \geq 0$,

$$\mathbf{v}_n \underset{||\cdot||_\infty}{\to} \mathbf{v}^\pi,$$
$$\mathbf{v}^\pi = T^\pi(\mathbf{v}^\pi),$$
$$||\mathbf{v}_n - \mathbf{v}^\pi||_\infty \leq \gamma^n ||\mathbf{v}_0 - \mathbf{v}^\pi||_\infty.$$

Convergence of iterative policy evaluation

Definition (Cauchy sequence)

Consider a normed vector space $(Z, ||\cdot||)$. A sequence $(\mathbf{v}_n)_{n\geq 0} = \mathbf{v}_0, \mathbf{v}_1, \ldots$ of vectors in this space is Cauchy if

$$\lim_{n\to\infty} \sup_{m\geq n} ||\mathbf{v}_n - \mathbf{v}_m|| = 0.$$

In other words, if a sequence $(\mathbf{v}_n)_{n\geq 0}$ is Cauchy, then for every $\epsilon > 0$ there is an N such that for every $n \geq N$, we have $\sup_{m\geq n} ||\mathbf{v}_n - \mathbf{v}_m|| < \epsilon$.

Convergence of iterative policy evaluation

Definition (Banach space)

A Banach space is a normed vector space $(Z, ||\cdot||)$ where if $(\mathbf{v}_n)_{n\geq 0}$ is a Cauchy sequence then $\mathbf{v}_n \underset{||\cdot||}{\to} \mathbf{v}$ for some vector $\mathbf{v}$.

For any d, both $(\mathbb{R}^d, ||\cdot||_2)$ and $(\mathbb{R}^d, ||\cdot||_\infty)$ are Banach spaces, although we omit the corresponding proofs.

Convergence of iterative policy evaluation

Definition (L-contraction)

Consider a normed vector space $(Z, ||\cdot||)$ and a function $T : Z \to Z$. The function T is L-Lipschitz if

$$||T(\mathbf{u}) - T(\mathbf{v})|| \leq L||\mathbf{u} - \mathbf{v}||,$$

for all $\mathbf{u}, \mathbf{v} \in Z$. If $L < 1$, then T is also an L-contraction.

Convergence of iterative policy evaluation

Lemma

Consider a normed vector space $(Z, ||\cdot||)$*, an L-Lipschitz function* $T : Z \to Z$*, and a sequence* $(\mathbf{v}_n)_{n\geq 0} = \mathbf{v}_0, \mathbf{v}_1, \ldots$ *of vectors in this space. If* $\mathbf{v}_n \underset{||\cdot||}{\to} \mathbf{v}$*, then* $T(\mathbf{v}_n) \underset{||\cdot||}{\to} T(\mathbf{v})$*.*

Convergence of iterative policy evaluation

Proof.

For any $n \geq 0$, by the definition of an L-Lipschitz function,

$$0 \leq ||T(\mathbf{v}_n) - T(\mathbf{v})|| \leq L||\mathbf{v}_n - \mathbf{v}||.$$

Since $\mathbf{v}_n \underset{||\cdot||}{\to} \mathbf{v}$,

$$\lim_{n\to\infty} L||\mathbf{v}_n - \mathbf{v}|| = L \lim_{n\to\infty} ||\mathbf{v}_n - \mathbf{v}|| = 0.$$

By the squeeze theorem,

$$\lim_{n\to\infty} ||T(\mathbf{v}_n) - T(\mathbf{v})|| = 0.$$

Convergence of iterative policy evaluation

Theorem (Banach's fixed point theorem)

If $(Z, ||\cdot||)$ is a Banach space and $T : Z \to Z$ is an L-contraction, then T has a unique fixed point $\mathbf{v}$. Furthermore, for any $\mathbf{v}_0 \in Z$, let $\mathbf{v}_{n+1} = T(\mathbf{v}_n)$. For any $n \geq 0$,

$$\mathbf{v}_n \underset{||\cdot||}{\to} \mathbf{v},$$

$$||\mathbf{v}_n - \mathbf{v}|| \leq L^n ||\mathbf{v}_0 - \mathbf{v}||.$$

Convergence of iterative policy evaluation

Proof.

We first show that the sequence $(\mathbf{v}_n)_{n\geq 0}$ is Cauchy, which guarantees that $\mathbf{v}_n \underset{||\cdot||}{\to} \mathbf{v}$ for some vector $\mathbf{v}$.

As a first step, we show that $||\mathbf{v}_{n+k} - \mathbf{v}_n|| \leq L^n ||\mathbf{v}_k - \mathbf{v}_0||$ for any $n, k \geq 0$. The case $n = 0$ is trivial. Suppose that the inductive hypothesis is true for some n, and consider the case $n + 1$:

$$
\begin{aligned}
||\mathbf{v}_{n+k+1} - \mathbf{v}_{n+1}|| &= ||T(\mathbf{v}_{n+k}) - T(\mathbf{v}_n)|| && \text{(definition of the sequence)} \\
&\leq L||\mathbf{v}_{n+k} - \mathbf{v}_n|| && \text{(definition of } L\text{-contraction)} \\
&\leq L^{n+1}||\mathbf{v}_k - \mathbf{v}_0||, && \text{(inductive hypothesis)}
\end{aligned}
$$

as we wanted to show.

Convergence of iterative policy evaluation

Proof. (cont.)

For $k \geq 1$, $||\mathbf{v}_k - \mathbf{v}_0|| = ||\mathbf{v}_k + (-\mathbf{v}_{k-1} + \mathbf{v}_{k-1}) + \ldots + (-v_1 + v_1) - \mathbf{v}_0||$. Therefore,

$$
\begin{aligned}
||\mathbf{v}_k - \mathbf{v}_0|| &= \left\| \sum_{i=1}^{k} \mathbf{v}_i - \mathbf{v}_{i-1} \right\| && \text{(reorganizing terms)} \\
&\leq \sum_{i=1}^{k} ||\mathbf{v}_i - \mathbf{v}_{i-1}|| && \text{(triangle inequality)} \\
&\leq \sum_{i=1}^{k} L^{i-1} ||\mathbf{v}_1 - \mathbf{v}_0|| && \text{(earlier result)} \\
&\leq \frac{||\mathbf{v}_1 - \mathbf{v}_0||}{1 - L}. && \left(\lim_{n \to \infty} \sum_{i=0}^{n} L^n = \frac{1}{1 - L} \right)
\end{aligned}
$$

Convergence of iterative policy evaluation

Proof. (cont.)

We are now close to showing that $(\mathbf{v}_n)_{n\geq 0}$ is Cauchy. For any $n, k \geq 0$, combining the previous two results,

$$0 \leq ||\mathbf{v}_{n+k} - \mathbf{v}_n|| \leq L^n ||\mathbf{v}_k - \mathbf{v}_0|| \leq L^n \frac{||\mathbf{v}_1 - \mathbf{v}_0||}{1 - L}.$$

Therefore, for any fixed $n \geq 0$,

$$0 \leq \sup_{k\geq 0} ||\mathbf{v}_{n+k} - \mathbf{v}_n|| \leq \sup_{k\geq 0} L^n \frac{||\mathbf{v}_1 - \mathbf{v}_0||}{1 - L} = L^n \frac{||\mathbf{v}_1 - \mathbf{v}_0||}{1 - L}.$$

Convergence of iterative policy evaluation

Proof. (cont.)

Because $0 \leq L < 1$,

$$\lim_{n \to \infty} L^n \frac{||\mathbf{v}_1 - \mathbf{v}_0||}{1 - L} = \frac{||\mathbf{v}_1 - \mathbf{v}_0||}{1 - L} \lim_{n \to \infty} L^n = 0.$$

Therefore, by the squeeze theorem,

$$\lim_{n \to \infty} \sup_{k \geq 0} ||\mathbf{v}_{n+k} - \mathbf{v}_n|| = 0,$$

which completes the proof that $(\mathbf{v}_n)_{n \geq 0}$ is Cauchy. Let $\mathbf{v}$ denote the vector such that $\mathbf{v}_n \underset{||\cdot||}{\to} \mathbf{v}$.

Convergence of iterative policy evaluation

Proof. (cont.)

Our next step is to show that $\mathbf{v}$ is a fixed point of T. For any n,

$$\begin{aligned} 0 \leq ||T(\mathbf{v}) - \mathbf{v}|| &= ||T(\mathbf{v}) + (-T(\mathbf{v}_n) + T(\mathbf{v}_n)) - \mathbf{v}|| && \text{(introducing zeros)} \\ &\leq ||T(\mathbf{v}) - T(\mathbf{v}_n)|| + ||T(\mathbf{v}_n) - \mathbf{v}|| && \text{(triangle inequality)} \\ &\leq L||\mathbf{v} - \mathbf{v}_n|| + ||\mathbf{v}_{n+1} - \mathbf{v}|| && (L\text{-contraction}) \end{aligned}$$

Because $\mathbf{v}_n \underset{||\cdot||}{\to} \mathbf{v}$,

$$\lim_{n \to \infty} L||\mathbf{v} - \mathbf{v}_n|| + ||\mathbf{v}_{n+1} - \mathbf{v}|| = 0.$$

Convergence of iterative policy evaluation

Proof. (cont.)

Therefore, by the squeeze theorem

$$||T(\mathbf{v}) - \mathbf{v}|| = \lim_{n \to \infty} ||T(\mathbf{v}) - \mathbf{v}|| = 0.$$

By the definition of a norm, $T(\mathbf{v}) - \mathbf{v} = \mathbf{0}$, which implies $T(\mathbf{v}) = \mathbf{v}$, completing the proof.

Convergence of iterative policy evaluation

Proof. (cont.)

Our next step is to show that the fixed point of T is unique. Suppose that $T(\mathbf{u}) = \mathbf{u}$ and $T(\mathbf{v}) = \mathbf{v}$ for some vectors $\mathbf{u}$ and $\mathbf{v}$. In that case,

$$||\mathbf{u} - \mathbf{v}|| = ||T(\mathbf{u}) - T(\mathbf{v})|| \leq L||\mathbf{u} - \mathbf{v}||.$$

If we suppose that $||\mathbf{u} - \mathbf{v}|| > 0$, dividing the inequation by $||\mathbf{u} - \mathbf{v}||$ leads to the conclusion that $L \geq 1$. However, T is an L-contraction, contradicting our supposition. Therefore, $||\mathbf{u} - \mathbf{v}|| \leq 0$, which implies that $\mathbf{u} = \mathbf{v}$.

Convergence of iterative policy evaluation

Proof. (cont.)

Our last step is to show that $||\mathbf{v}_n - \mathbf{v}|| \leq L^n ||\mathbf{v}_0 - \mathbf{v}||$, for any n. The case $n = 0$ is trivial. Suppose that the inductive hypothesis is true for some n, and consider the case $n + 1$:

$$\begin{aligned} ||\mathbf{v}_{n+1} - \mathbf{v}|| &= ||T(\mathbf{v}_n) - T(\mathbf{v})|| && \text{(definition of fixed point)} \\ &\leq L||\mathbf{v}_n - \mathbf{v}|| \leq L^{n+1}||\mathbf{v}_0 - \mathbf{v}||, && \text{(inductive hypothesis)} \end{aligned}$$

as we wanted to show. □

Convergence of iterative policy evaluation

Theorem (Convergence of iterative policy evaluation)

Consider the Bellman operator $T^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ *for the policy* π. *Given an arbitrary* $\mathbf{v}_0 \in \mathbb{R}^{|\mathcal{S}|}$, *consider also the sequence* $(\mathbf{v}_n)_{n \geq 0}$ *where* $\mathbf{v}_{k+1} = T^\pi(\mathbf{v}_k)$. *Finally, consider the vector* $\mathbf{v}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ *such that* $v_s^\pi = V^\pi(s)$, *for any* $s \in \mathcal{S}$. *For any* $n \geq 0$,

$$\mathbf{v}_n \underset{||\cdot||_\infty}{\to} \mathbf{v}^\pi,$$

$$\mathbf{v}^\pi = T^\pi(\mathbf{v}^\pi),$$

$$||\mathbf{v}_n - \mathbf{v}^\pi||_\infty \leq \gamma^n ||\mathbf{v}_0 - \mathbf{v}^\pi||_\infty.$$

Convergence of iterative policy evaluation

Proof.

As a first step, we show that $\mathbf{v}^\pi$ is a fixed point of T^π. For any $s \in \mathcal{S}$, by the definition of T^π and V^π,

$$T^\pi(\mathbf{v}^\pi)_s = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma v^\pi_{s'}\right] = v^\pi_s.$$

Our next step is to show that the Bellman operator $T^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ is a γ-contraction. Because $(\mathbb{R}^{|\mathcal{S}|}, ||\cdot||_\infty)$ is a Banach space and $\mathbf{v}^\pi$ is a fixed point of T^π, the desired results follow from Banach's fixed point theorem.

Convergence of iterative policy evaluation

Proof. (cont.)

Note that, for any two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{|\mathcal{S}|}$ and every state $s \in \mathcal{S}$,

$$\begin{aligned} T^\pi(\mathbf{u})_s - T^\pi(\mathbf{v})_s &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma u_{s'}\right] \\ &\quad - \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma v_{s'}\right] \\ &= \sum_a \sum_{s'} \pi(s,a) \mathcal{P}^a_{ss'} \gamma u_{s'} - \sum_a \sum_{s'} \pi(s,a) \mathcal{P}^a_{ss'} \gamma v_{s'} \\ &= \gamma \sum_a \sum_{s'} \pi(s,a) \mathcal{P}^a_{ss'} [u_{s'} - v_{s'}]. \end{aligned}$$

Convergence of iterative policy evaluation

Proof. (cont.)

By the definition of maximum norm, for any two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{|\mathcal{S}|}$,

$$
\begin{aligned}
&||T^{\pi}(\mathbf{u}) - T^{\pi}(\mathbf{v})||_{\infty} \\
&= \gamma \max_{s} \Big| \sum_{a} \sum_{s'} \pi(s,a) \mathcal{P}^{a}_{ss'} [u_{s'} - v_{s'}] \Big| && \text{(definition of maximum norm)} \\
&\leq \gamma \max_{s} \sum_{a} \sum_{s'} \Big| \pi(s,a) \mathcal{P}^{a}_{ss'} [u_{s'} - v_{s'}] \Big| && \text{(triangle inequality)} \\
&= \gamma \max_{s} \sum_{a} \sum_{s'} \pi(s,a) \mathcal{P}^{a}_{ss'} |u_{s'} - v_{s'}| && \text{(multiplicativity)} \\
&\leq \gamma \max_{s} \sum_{a} \sum_{s'} \pi(s,a) \mathcal{P}^{a}_{ss'} ||\mathbf{u} - \mathbf{v}||_{\infty} && \text{(definition of maximum norm)} \\
&= \gamma ||\mathbf{u} - \mathbf{v}||_{\infty} \max_{s} \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} && \text{(distributivity)} \\
&= \gamma ||\mathbf{u} - \mathbf{v}||_{\infty} && \text{(unit measure).}
\end{aligned}
$$

□

Deterministic policies

- A deterministic policy π is one such that, for all $s \in \mathcal{S}$, $\pi(s, a) = 1$ for some $a \in \mathcal{A}$ and $\pi(s, b) = 0$ for all $b \neq a$
- In this case, we abuse notation and represent a policy by a function $\pi : \mathcal{S} \to \mathcal{A}$ from states to actions

Policy improvement

- Let π and π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$, $Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s)$
- The policy improvement theorem guarantees that $V^{\pi'}(s) \geq V^{\pi}(s)$ for all $s \in \mathcal{S}$
- For all $s \in \mathcal{S}$, a policy π may be improved to a policy π' by letting

$$\pi'(s) = \arg\max_a Q^{\pi}(s, a) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')]$$

Policy iteration

- Policy evaluation and policy improvement can be interleaved
- This process produces the sequence

$$\pi_0, V^{\pi_0}, \pi_1, V^{\pi_1}, \pi_2, V^{\pi_2}, \dots$$

- If $\pi_t = \pi_{t+1}$, then π_t is optimal by the uniqueness of V^*
- The initial policy π_0 can be arbitrary

Value iteration

- A more efficient alternative iteratively improves the estimates for the value of each state under an optimal policy
- It relies on creating a sequence $V_0, V_1, \ldots$ of estimates given by:

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

- The initial estimate V_0 can be arbitrary
- The sequence $V_0(s), V_1(s), \ldots$ converges to $V^*(s)$ for all $s \in \mathcal{S}$
- In-place value iteration has the same guarantees

Value iteration

Algorithm 2 Value iteration (in-place)

Input: one-step dynamics ($\mathcal{P}$ and $\mathcal{R}$), discount factor γ, and tolerance θ.
Output: optimal deterministic policy π when $\theta \to 0$.

for each $s \in \mathcal{S}$ **do**
  $V(s) \leftarrow 0$
end for
repeat
  $\Delta \leftarrow 0$
  for each $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$
for each $s \in \mathcal{S}$ **do**
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
end for

Additional reading

- **Notes on reinforcement learning (Section 4) [Rauber, 2015]**
- **Reinforcement learning: an introduction [Sutton and Barto, 2018]**
- Algorithms for reinforcement learning [Szepesvári, 2010]
- UCL course on reinforcement learning [Silver, 2015]
- UC Berkeley course on deep reinforcement learning [Levine, 2018]
- Stanford course on reinforcement learning [Ng, 2008]
- Stanford course on reinforcement learning [Brunskill, 2019]
- Deep reinforcement learning bootcamp [Abbeel et al., 2017]

References I


Abbeel, P. et al. (2017).
Deep reinforcement learning bootcamp.
https://sites.google.com/view/deep-rl-bootcamp/lectures.


Brunskill, E. (2019).
CS234: Reinforcement learning.
http://web.stanford.edu/class/cs234/schedule.html.


Levine, S. (2018).
Deep reinforcement learning.
http://rail.eecs.berkeley.edu/deeprlcourse-fa18/.


Ng, A. (2008).
CS 229: Machine learning.
https://www.youtube.com/watch?v=RtxI449ZjSc.

References II


Rauber, P. (2015).
Notes on reinforcement learning.
http://paulorauber.com/work.html.


Silver, D. (2015).
UCL course on reinforcement learning.
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.


Sutton, R. S. and Barto, A. G. (2018).
Reinforcement Learning: An Introduction.
The MIT Press, second edition.


Szepesvári, C. (2010).
Algorithms for reinforcement learning.
Synthesis lectures on artificial intelligence and machine learning, 4(1):1–103.

Queen Mary
University of London

Artificial Intelligence in Games

Exploration and Exploitation; Tabular Model-Free Algorithms

Paulo Rauber
School of Electronic Engineering and Computer Science

Learning a model: the naive approach

- If the one-step dynamics are not available, they can always be estimated by interacting with the environment randomly
- For any $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$, the simplest (maximum likelihood) estimate for $\mathcal{P}^a_{ss'}$ is given by

$$\hat{\mathcal{P}}^a_{ss'} = \frac{N(s', s, a)}{N(s, a)},$$

  where $N(s, a) > 0$ is the number of times that action a was taken at state s, and $N(s', s, a)$ is the number of times that state s' was observed after action a was taken at state s
- The estimates $\hat{\mathcal{P}}^a_{ss'}$ and $\hat{\mathcal{R}}^a_{ss'}$ can be combined with value iteration
- However, obtaining good estimates by interacting with the environment randomly may take too long

Exploration and exploitation

- Exploration-exploitation trade-off: should the agent explore in order to learn about potentially better sources of reward or exploit the well-known sources of reward?
- This trade-off is relevant whenever an environment model is not available
- Excessive exploration typically leads to poor short-term performance, while excessive exploitation typically leads to poor long-term performance

Multi-armed bandits

- An agent interacts with an environment during a sequence of T discrete time steps
- At each time step $t \geq 0$, the agent selects an action $a_t \in \mathcal{A}$
- One time step later, the agent receives a reward $r_{t+1} \in \mathbb{R}$ drawn from a fixed probability distribution with mean μ_{a_t} associated to action a_t
- The interaction between the agent and the environment up to time step t can be represented by a trajectory $\tau_t = a_0, r_1, a_1, r_2, \ldots, a_{t-2}, r_{t-1}$

1

[1] Image from [Klein et al., 2019]

Bandit algorithms

- For each time step t, a bandit algorithm provides a policy $\pi_t : \mathcal{A} \to [0, 1]$ such that $\pi_t(a)$ is the prescribed probability for taking action a after observing the trajectory τ_t
- The objective of an agent is to maximize the expected return given by

$$\mathbb{E}\left[\sum_{t=1}^{T} R_t\right],$$

where the expectation is (implicitly) over trajectories obtained by the agent in a specific environment

Regret

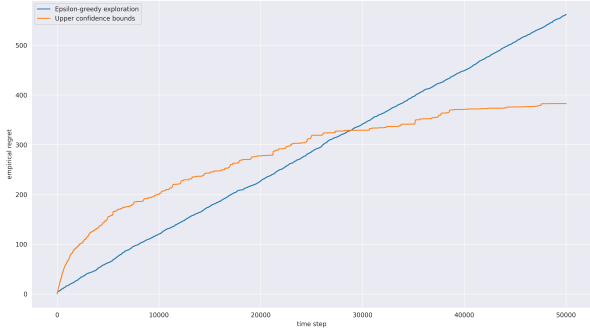
- The regret $E(t) \geq 0$ of a bandit algorithm after t time steps is given by

$$E(t) = t\mu_* - \mathbb{E}\left[\sum_{t'=1}^{t} R_{t'}\right],$$

  where $\mu_* = \max_a \mu_a$ and the expectation is (implicitly) over trajectories obtained by the bandit algorithm in a specific environment
- The regret compares the expected return of the optimal agent for an environment with the expected return of a specific bandit algorithm

Regret

- Bandit algorithms differ in the rate of increase of their regrets



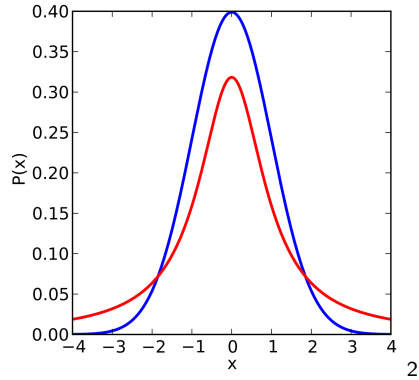
- A bandit algorithm has linear regret if $E(t) \in O(t)$
- A bandit algorithm has logarithmic regret if $E(t) \in O(\log(t))$

ϵ-greedy exploration

- During the first $|\mathcal{A}|$ time steps, each action is selected once
- For each action $a \in \mathcal{A}$, the empirical mean $\hat{\mu}_a$ of its observed rewards is recorded
- For each time step $t \geq |\mathcal{A}|$, there is a parameter $\epsilon_t \in [0, 1]$
- At each time step $t \geq |\mathcal{A}|$, the action with the highest empirical mean is selected with probability $1 - \epsilon_t$ and a random action is selected with probability ϵ_t

Subgaussianity

- A random variable X is σ-subgaussian if, for all $\lambda \in \mathbb{R}$, $\mathbb{E}[\exp(\lambda X)] \leq \exp(\lambda^2 \sigma^2 / 2)$
- In simple terms, the tails of a σ-subgaussian distribution decay approximately as fast as those of a Gaussian distribution with zero mean and variance σ



[2]

[2]Image from Wikipedia

ϵ-greedy exploration: regret

- Suppose that the reward for each action is a 1-subgaussian random variable
- Furthermore, suppose $\epsilon \in (0, 1)$ and $\epsilon_t = \epsilon$ for every t
- In that case, if $|\mathcal{A}| \geq 2$, then

$$\lim_{t \to \infty} \frac{E(t)}{t} = \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \Delta_a,$$

  where $\Delta_a = \mu_* - \mu_a$ is the so-called suboptimality gap for action a
- Therefore, ϵ-greedy exploration with a fixed ϵ has linear regret
- This result can be generalized to σ-subgaussian random variables by scaling rewards
- By choosing each ϵ_t appropriately, ϵ-greedy exploration may achieve sublinear regret

Upper confidence bounds

- During the first $|\mathcal{A}|$ time steps, each action is selected once
- Each action a is associated with an upper confidence bound B_a such that $B_a > \mu_a$ with high probability
- At each time step $t \geq |\mathcal{A}|$, the action with the highest upper confidence bound is selected
- In simple terms, the upper confidence bound of every non-optimal action eventually falls below the upper confidence bound of every optimal action
- Upper confidence bounds employ the principle of optimism in the face of uncertainty

Upper confidence bound for 1-subgaussian variables

- Let $X_1, X_2, \ldots, X_T$ be a sequence of independent 1-subgaussian variables with mean μ
- If $M = \frac{1}{T}\sum_{t=1}^{T} X_t$ denotes the sample mean, then

$$P\left(\mu \geq M + \sqrt{\frac{2\log(1/\delta)}{T}}\right) \leq \delta$$

  for any $\delta \in (0, 1)$
- Intuitively, the result of adding a specific positive term to the sample mean almost certainly overestimates the underlying mean

Upper confidence bounds

- Suppose that each action $a \in \mathcal{A}$ has been selected at least once
- The upper confidence bound B_a for action a is given by

$$B_a = \hat{\mu}_a + \sqrt{\frac{2\log(1/\delta)}{n_a}},$$

  where $\hat{\mu}_a$ is the empirical mean of the rewards observed for action a, n_a is the number of rewards observed for action a, and δ is a parameter of the algorithm
- As mentioned before, at each time step $t \geq |\mathcal{A}|$, the action with the highest upper confidence bound is selected
- Intuitively, an action a is selected if $\hat{\mu}_a$ is relatively high or if n_a is relatively low

Upper confidence bounds: regret

- Suppose that the reward for each action is a 1-subgaussian random variable
- For any $T \geq |\mathcal{A}|$, if $\delta = 1/T^2$, then

$$E(T) \leq 3 \sum_{a \in \mathcal{A}} \Delta_a + \sum_{a | \Delta_a > 0} \frac{16 \log T}{\Delta_a},$$

  where $\Delta_a = \mu_* - \mu_a$ is the suboptimality gap for action a
- Therefore, upper confidence bounds have logarithmic regret for appropriate δ
- This result can be generalized to σ-subgaussian random variables by scaling rewards
- There are many other versions of upper confidence bounds that provide better performance in different settings

Posterior sampling

- Posterior sampling (Thompson sampling) is a Bayesian bandit algorithm that provides excellent performance in many settings
- Posterior sampling can be generalized to provide a reinforcement learning algorithm that learns a model of an environment while balancing exploration and exploitation
  1. The agent represents its knowledge by a distribution over models
  2. A single model is drawn from this distribution
  3. An optimal policy is found for this model
  4. This policy is used to interact with the environment for one episode
  5. The resulting data are used to update the distribution over models
  6. The process continues from Step 2
- Intuitively, increased certainty leads to decreased exploration

Monte Carlo control

- Monte Carlo control methods find an optimal policy without estimating the one-step dynamics by interleaving policy evaluation and policy improvement
- These methods require an episodic problem, where there is a transition to an absorbing state after a finite number of time steps
- Policy evaluation for π consists of experiencing several episodes and averaging the returns that follow every possible state action pair (s, a) to obtain an estimate of $Q^{\pi}(s, a)$.
- In practice, "policy improvement" based on π is performed before a reliable estimate of Q^{π} is available
- Monte Carlo control typically relies on ϵ-greedy policies to ensure that the environment is explored sufficiently
- For a given state s, an ϵ-greedy policy with respect to an estimate Q of the action value function chooses a random action with probability ϵ, and an action $\arg\max_a Q(s, a)$ with probability $1 - \epsilon$

Monte Carlo control

Algorithm 1 Monte Carlo control algorithm

Input: set of states $\mathcal{S}$, number of episodes N, probability of choosing random action ϵ.
Output: deterministic policy π, optimal when $N \to \infty$.

for each $s \in \mathcal{S}$ **do**
  for each action $a \in \mathcal{A}(s)$ **do**
    $Q(s, a) \leftarrow 0$
    $n(s, a) \leftarrow 0$
  end for
end for
for each i in $\{1, \ldots, N\}$ **do**
  Experience a new episode e following an ϵ-greedy policy based on Q.
  for each state-action pair (s, a) in the episode e **do**
    $u \leftarrow$ return following (s, a) in the episode e.
    $n(s, a) \leftarrow n(s, a) + 1$
    $Q(s, a) \leftarrow Q(s, a) + \frac{1}{n(s,a)}[u - Q(s, a)]$
  end for
end for
for each state $s \in \mathcal{S}$ **do**
  $\pi(s) \leftarrow \arg\max_a Q(s, a)$
end for

Temporal difference

- Consider the tuple $h_t = (s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ obtained by an agent using a policy π to interact with an environment
- Let Q denote an estimate of the action value function Q^π
- The one-step return based on h_t and Q is given by

$$r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$$

- The temporal difference for (s_t, a_t) based on h_t and Q is given by

$$r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

- In other words, the difference between the immediate reward plus the (estimated) expected return from the next state and the (estimated) expected return for the current state

Sarsa control

- An algorithm *bootstraps* if it improves the estimate of the value of a state based on estimates of the values of other states
- Sarsa control is similar to Monte Carlo control, but it bootstraps based on temporal differences
- Sarsa control is comparatively more *sample efficient*, since it does not rely on the return that follows (s_t, a_t) after a single episode
- Given the tuple h_t and the estimate Q, Sarsa control updates its estimate of $Q(s_t, a_t)$ using

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)],$$

where α is the so-called learning rate

Sarsa control

Algorithm 2 Sarsa control algorithm

Input: set of states $\mathcal{S}$, number of episodes N, learning rate α, probability of random action ϵ, discount factor γ.
Output: deterministic policy π, optimal when $N \to \infty$ and α decays appropriately.

for each $(s, a) \in \mathcal{S} \times \mathcal{A}$ **do**
  $Q(s, a) \leftarrow 0$
end for
for each i in $\{1, \ldots, N\}$ **do**
  $s \leftarrow$ initial state for episode i
  Select action a for state s according to an ϵ-greedy policy based on Q.
  while state s is not terminal **do**
    $r \leftarrow$ observed reward for action a at state s
    $s' \leftarrow$ observed next state for action a at state s
    Select action a' for state s' according to an ϵ-greedy policy based on Q.
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end while
end for
for each state $s \in \mathcal{S}$ **do**
  $\pi(s) \leftarrow \arg\max_a Q(s, a)$
end for

Q-learning

- An algorithm is *off-policy* if it learns about a policy that is different from the policy that it uses to act in the environment
- Q-learning learns about a greedy policy while acting using an ϵ-greedy policy
- Q-learning control is similar to Sarsa control: both algorithms bootstrap based on temporal differences
- Given the tuple h_t and the estimate Q, Q-learning control updates its estimate of $Q(s_t, a_t)$ using

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where α is the so-called learning rate

Q-learning control

Algorithm 3 Q-learning control algorithm

Input: set of states $\mathcal{S}$, number of episodes N, learning rate α, probability of random action ϵ, discount factor γ.
Output: deterministic policy π, optimal when $N \to \infty$ and α decays appropriately.

for each $(s, a) \in \mathcal{S} \times \mathcal{A}$ **do**
  $Q(s, a) \leftarrow 0$
end for
for each i in $\{1, \ldots, N\}$ **do**
  $s \leftarrow$ initial state for episode i
  while state s is not terminal **do**
    Select action a for state s according to an ϵ-greedy policy based on Q.
    $r \leftarrow$ observed reward for action a at state s
    $s' \leftarrow$ observed next state for action a at state s
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
  end while
end for
for each state $s \in \mathcal{S}$ **do**
  $\pi(s) \leftarrow \arg\max_a Q(s, a)$
end for

Additional reading

- **Notes on reinforcement learning (Section 2, 5, 6, and 9) [Rauber, 2015]**
- **Reinforcement learning: an introduction [Sutton and Barto, 2018]**
- **Bandit Algorithms [Lattimore and Szepesvári, 2019]**
- A Tutorial on Thompson Sampling [Russo et al., 2018]
- (More) Efficient Reinforcement Learning via Posterior Sampling [Osband et al., 2013]
- Algorithms for reinforcement learning [Szepesvári, 2010]
- UCL course on reinforcement learning [Silver, 2015]
- UC Berkeley course on deep reinforcement learning [Levine, 2018]
- Stanford course on reinforcement learning [Ng, 2008]
- Stanford course on reinforcement learning [Brunskill, 2019]
- Deep reinforcement learning bootcamp [Abbeel et al., 2017]

References I


Abbeel, P. et al. (2017).
Deep reinforcement learning bootcamp.
https://sites.google.com/view/deep-rl-bootcamp/lectures.


Brunskill, E. (2019).
CS234: Reinforcement learning.
http://web.stanford.edu/class/cs234/schedule.html.


Klein, D., Abbeel, P., and Dragan, A. (2019).
CS 188: Introduction to artificial intelligence.
https://inst.eecs.berkeley.edu/~cs188/fa19/.


Lattimore, T. and Szepesvári, C. (2019).
Bandit Algorithms.
Cambridge University Press.

References II

Levine, S. (2018).
Deep reinforcement learning.
http://rail.eecs.berkeley.edu/deeprlcourse-fa18/.

Ng, A. (2008).
CS 229: Machine learning.
https://www.youtube.com/watch?v=RtxI449ZjSc.

Osband, I., Russo, D., and Van Roy, B. (2013).
(more) efficient reinforcement learning via posterior sampling.
In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 3003–3011. Curran Associates, Inc.

Rauber, P. (2015).
Notes on reinforcement learning.
http://paulorauber.com/work.html.

References III


Russo, D. J., Roy, B. V., Kazerouni, A., Osband, I., and Wen, Z. (2018).
A tutorial on Thompson sampling.
Foundations and Trends in Machine Learning, 11(1):1–96.


Silver, D. (2015).
UCL course on reinforcement learning.
`http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`.


Sutton, R. S. and Barto, A. G. (2018).
Reinforcement Learning: An Introduction.
The MIT Press, second edition.


Szepesvári, C. (2010).
Algorithms for reinforcement learning.
Synthesis lectures on artificial intelligence and machine learning, 4(1):1–103.

Queen Mary
University of London

Artificial Intelligence in Games

Non-Tabular Model-Free Algorithms

Paulo Rauber
School of Electronic Engineering and Computer Science

Supervised learning: regression

- Consider an iid training dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
- Suppose $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$
- Regression: predicting target y given new observation $\mathbf{x}$

$(2, 3, 5, 7, 11) \longmapsto$ £ 100

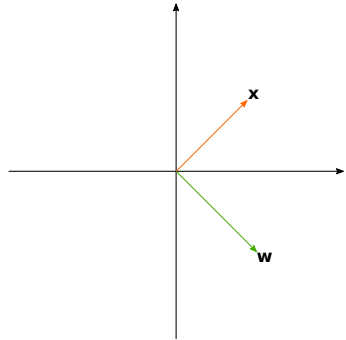
observation target

- In our setting, $\mathbf{x}_i$ will represent a state and y_i will represent its (estimated) value

Orthogonality

- A vector $\mathbf{x} \in \mathbb{R}^D$ is orthogonal to a vector $\mathbf{w} \in \mathbb{R}^D$ when

$$\mathbf{w} \cdot \mathbf{x} = w_1 x_1 + w_2 x_2 + \ldots + w_D x_D = \sum_{j=1}^{D} w_j x_j = 0$$

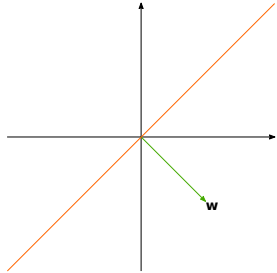


Hyperplanes

- A hyperplane $\mathcal{S}_{\mathbf{w}}$ is the set of vectors orthogonal to $\mathbf{w}$:

$$\mathcal{S}_{\mathbf{w}} = \{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{w} \cdot \mathbf{x} = 0\}$$

- For $D = 2$ or $D = 3$, a hyperplane is a line or plane that goes through the origin



Linear regression: model

- Consider an iid dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, where $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$
- Regression: predicting target y given new observation $\mathbf{x}$
- Linear regression (naive model):

$$y = \mathbf{w} \cdot \mathbf{x} = \sum_{j=1}^{D} w_j x_j$$

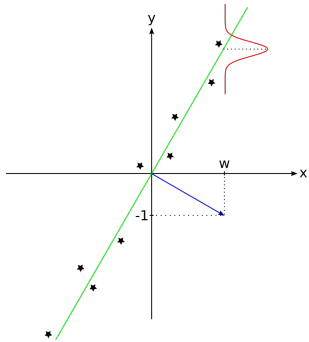
- Interpretation: w_j indicates how much each unit of x_j contributes towards the target
- Linear regression (probabilistic model):

$$\mathbb{E}[Y \mid \mathbf{x}, \mathbf{w}] = \mathbf{w} \cdot \mathbf{x},$$
$$p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y \mid \mathbf{w} \cdot \mathbf{x}, \sigma^2)$$

Linear regression: geometry for $D = 1$

- For any w, if $y = wx$, then $wx - y = 0$ and $(w, -1) \cdot (x, y) = 0$
- For any w, the pairs (x, y) for which $y = wx$ constitute a hyperplane

$$\{(x, y) \in \mathbb{R}^2 \mid (w, -1) \cdot (x, y) = 0\}$$



Linear regression: likelihood

- Assuming constant σ^2, the *conditional likelihood* is given by

$$p(\mathcal{D} \mid \mathbf{w}) = \prod_{i=1}^{N} \mathcal{N}(y_i \mid \mathbf{w} \cdot \mathbf{x}_i, \sigma^2)$$

- The log-likelihood is given by

$$\log p(\mathcal{D} \mid \mathbf{w}) = -\frac{N}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$$

- *Maximizing* the likelihood wrt $\mathbf{w}$ corresponds to *minimizing* J given by

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$$

- J can be minimized analytically (in non-degenerate cases)

Linear regression: extensions

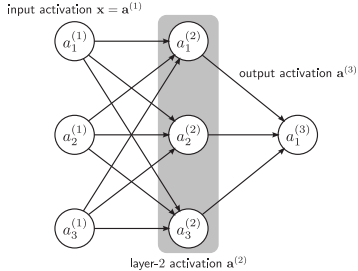
- If $\mathbf{w}$ maximizes the likelihood, we may predict $y = \mathbf{w} \cdot \mathbf{x}$ given $\mathbf{x}$
  - Alternative: maximum a posteriori estimate (requires a prior)
  - Bayesian alternative: using a posterior predictive distribution
- Using a feature map $\phi : \mathbb{R}^D \to \mathbb{R}^{D'}$:

$$p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y \mid \mathbf{w} \cdot \phi(\mathbf{x}), \sigma^2)$$

  - Bias-including feature map: $\phi(\mathbf{x}) = (\mathbf{x}, 1)$
    - $\mathbf{w} \cdot \phi(\mathbf{x}) = \mathbf{w}_{1:D} \cdot \mathbf{x} + w_{D+1}$
  - Polynomial feature map ($D = 1$): $\phi(x) = (1, x^1, \ldots, x^{D'-1})$
    - $\mathbf{w} \cdot \phi(x) = \sum_{j=1}^{D'} w_j x^{j-1}$

Feedforward neural network

- A feedforward neural network has a number of layers L
- This network may predict a target $y = a_1^{(L)}$ given a new observation $\mathbf{x} = \mathbf{a}^{(1)}$



- Let $N^{(l)}$ be the number of *neurons* in layer l
- The network has input neurons, hidden neurons, and output neurons

Feedforward neural network

- *Weighted input* to neuron j in layer $l > 1$:

$$z_j^{(l)} = b_j^{(l)} + \sum_{k=1}^{N^{(l-1)}} w_{j,k}^{(l)} a_k^{(l-1)},$$

- *Activation* of neuron j in layer $1 < l < L$:

$$a_j^{(l)} = \sigma(z_j^{(l)}),$$

where σ is a differentiable function, such as $\sigma(z) = \frac{1}{1+e^{-z}}$

Feedforward neural network

- Alternatively, the output of each layer $1 < l < L$ can be written as

$$\mathbf{a}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}),$$

  where the activation function is applied element-wise
- For regression, the activation of the output neuron is simply given by

$$a_1^{(L)} = z_1^{(L)}$$

- The output given $\mathbf{a}^{(1)} = \mathbf{x}$ is $a_1^{(L)}$

Feedforward neural network

- A feedforward neural network represents a parametric function of its weights and biases
- Let $\boldsymbol{\theta}$ represent a joint assignment of weights and biases
- *Maximizing* the likelihood $p(\mathcal{D} \mid \boldsymbol{\theta})$ corresponds to *minimizing* J given by

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(\mathbf{x},y)\in\mathcal{D}} \left(y - a_1^{(L)}\right)^2,$$

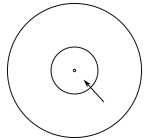
  where $a_1^{(L)}$ is the output of the network when $\mathbf{a}^{(1)} = \mathbf{x}$
- Minimization can be attempted by (stochastic) gradient descent or related techniques [Ruder, 2016]
- The gradient $\nabla J(\boldsymbol{\theta})$ can be computed using a technique called *backpropagation*

Gradient descent

- Consider the task of minimizing $f : \mathbb{R}^D \to \mathbb{R}$
- Gradient descent starts at an arbitrary estimate $\mathbf{x}_0 \in \mathbb{R}^D$ and iteratively updates this estimate using

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t),$$

where η_t is the learning rate at iteration t.



Generalization

- In large state spaces, some states may be seen very rarely
- In these cases, the state or action value estimates should generalize across states
- Generalization relies on function approximation
- The state value function $V^\pi : \mathcal{S} \to \mathbb{R}$ can be approximated by a parametric function $V : \mathcal{S} \times \mathbb{R}^m \to \mathbb{R}$
- The goal of policy evaluation becomes finding a parameter vector $\boldsymbol{\theta}$ such that

$$V^\pi(s) \approx V(s; \boldsymbol{\theta}),$$

  for every $s \in \mathcal{S}$
- Changing $\boldsymbol{\theta}$ changes the value estimates of several states

Example: linear value functions

- Suppose that any state $s \in \mathcal{S}$ can be represented by a feature vector $\phi(s) \in \mathbb{R}^m$, and that $V(s;\boldsymbol{\theta})$ is given by

$$V(s;\boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \phi(s) = \sum_{i=1}^{m} \theta_i \phi(s)_i$$

- Several model-free reinforcement learning algorithms work well in this case

Value regression

- For a given policy π, consider a dataset $\mathcal{D} = \{(s_i, V^\pi(s_i))\}_{i=1}^N$
- Value regression: predicting the value $V^\pi(s)$ of an unseen state s
- For a given parametric function V, the mean squared error $J(\boldsymbol{\theta})$ is given by

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} [V^\pi(s_i) - V(s_i; \boldsymbol{\theta})]^2$$

- Let $\boldsymbol{\theta}^*$ denote a parameter vector such that $J(\boldsymbol{\theta}^*) = \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
- The function $V(\cdot; \boldsymbol{\theta}^*)$ can be used to predict the value of unseen states

Stochastic gradient descent

- Stochastic gradient descent is a procedure that converges to a local minimum of J (if the learning rate α is decayed appropriately)
- The procedure starts with an arbitrary estimate $\boldsymbol{\theta}_0$
- For any $t \geq 0$, a pair $(s_t, V^\pi(s_t))$ is drawn at random from $\mathcal{D}$, and the estimate $\boldsymbol{\theta}_{t+1}$ is obtained using

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{1}{2}\alpha \nabla_{\boldsymbol{\theta}} [V^\pi(s_t) - V(s_t; \boldsymbol{\theta}_t)]^2,$$

  where α is the learning rate
- By the chain rule,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha [V^\pi(s_t) - V(s_t; \boldsymbol{\theta}_t)] \nabla_{\boldsymbol{\theta}} V(s_t; \boldsymbol{\theta}_t)$$

Value regression from estimates

- If $V^{\pi}(s)$ were available for all states $s \in \mathcal{S}$, there would be no need for function approximation
- Furthermore, in practice, a dataset will be given by $\mathcal{D} = \{(s_i, v_i)\}_{i=1}^{N}$, where v_i is an estimate of the value of s_i under policy π
- Different estimates v_i may be considered, such as the empirical return or one-step return observed after state s_i
- For any $t \geq 0$, a pair (s_t, v_t) is drawn at random from $\mathcal{D}$, and the estimate $\boldsymbol{\theta}_{t+1}$ for stochastic gradient descent is obtained using

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha[v_t - V(s_t; \boldsymbol{\theta}_t)]\nabla_{\boldsymbol{\theta}} V(s_t; \boldsymbol{\theta}_t)$$

Gradient descent TD value estimation

Algorithm 1 Gradient descent temporal-difference value estimation algorithm

Input: policy π, number of episodes N, learning rate α, discount factor γ
Output: parameter vector $\boldsymbol{\theta}$

Initialize $\boldsymbol{\theta}$ arbitrarily
for each i in $\{1, \ldots, N\}$ **do**
    $s \leftarrow$ initial state for episode i
    while state s is not terminal **do**
        $a \leftarrow \pi(s)$
        $r \leftarrow$ observed reward for action a at state s
        $s' \leftarrow$ observed next state for action a at state s
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha[r + \gamma V(s'; \boldsymbol{\theta}) - V(s; \boldsymbol{\theta})]\nabla_{\boldsymbol{\theta}} V(s; \boldsymbol{\theta})$
        $s \leftarrow s'$
    end while
end for

Non-tabular model-free algorithms

- Q-learning control and Sarsa control can be adapted to use stochastic gradient descent to approximate action-value functions (rather than just value functions)
- The choice of an appropriate feature map $\phi : \mathcal{S} \to \mathbb{R}^m$ is crucial to the success of any function approximation method
- In the case of linear approximation, it may be necessary to create features that are combinations of more natural features, since linear models are incapable of modeling relationships such as feature i being beneficial only in the absence of feature j

Q-learning control

Algorithm 2 Q-learning control algorithm for linear function approximation

Input: feature vector $\phi(s, a)$ for all state-action pairs (s, a), number of episodes N, learning rate α, exploration factor ϵ, discount factor γ
Output: parameter vector θ

$\theta \leftarrow \mathbf{0}$
for each i in $\{1, \ldots, N\}$ **do**
    $s \leftarrow$ initial state for episode i
    for each action a: **do**
        $Q(a) \leftarrow \sum_i \theta_i \phi(s, a)_i$
    end for
    while state s is not terminal **do**
        if with probability $1 - \epsilon$: **then**
            $a \leftarrow \arg\max_a Q(a)$
        else
            $a \leftarrow$ random action
        end if
        $r \leftarrow$ observed reward for action a at state s
        $s' \leftarrow$ observed next state for action a at state s
        $\delta \leftarrow r - Q(a)$
        for each action a': **do**
            $Q(a') \leftarrow \sum_i \theta_i \phi(s', a')_i$
        end for
        $\delta \leftarrow \delta + \gamma \max_{a'} Q(a')$ {Note: δ is the temporal difference}
        $\theta \leftarrow \theta + \alpha \delta \phi(s, a)$
        $s \leftarrow s'$
    end while
end for

Additional reading: linear regression

- **Notes on Machine Learning (Section 7) [Rauber, 2016]**
- **Reinforcement learning: an introduction [Sutton and Barto, 2018]**
- **Pattern Recognition and Machine Learning (Chapter 3) [Bishop, 2006]**
- Machine Learning: a Probabilistic Perspective (Chapter 7) [Murphy, 2012]

Additional reading: feedforward neural networks

- **Notes on neural networks (Section 2) [Rauber, 2015b]**
- **Notes on machine learning (Section 17) [Rauber, 2016]**
- **Neural networks and deep learning (Chapter 1) [Nielsen, 2015]**
- Pattern Recognition and Machine Learning (Chapter 5) [Bishop, 2006]
- Machine Learning: a Probabilistic Perspective (Section 16.5) [Murphy, 2012]

Additional reading: generalization in RL

- **Notes on reinforcement learning (Section 8) [Rauber, 2015a]**
- **Reinforcement learning: an introduction [Sutton and Barto, 2018]**
- Algorithms for reinforcement learning [Szepesvári, 2010]
- UCL course on reinforcement learning [Silver, 2015]
- UC Berkeley course on deep reinforcement learning [Levine, 2018]
- Stanford course on reinforcement learning [Ng, 2008]
- Stanford course on reinforcement learning [Brunskill, 2019]
- Deep reinforcement learning bootcamp [Abbeel et al., 2017]

References I


Abbeel, P. et al. (2017).
Deep reinforcement learning bootcamp.
https://sites.google.com/view/deep-rl-bootcamp/lectures.


Bishop, C. M. (2006).
Pattern Recognition and Machine Learning.
Springer.


Brunskill, E. (2019).
CS234: Reinforcement learning.
http://web.stanford.edu/class/cs234/schedule.html.


Levine, S. (2018).
Deep reinforcement learning.
http://rail.eecs.berkeley.edu/deeprlcourse-fa18/.

References II


Murphy, K. P. (2012).
Machine learning: a probabilistic perspective.
MIT Press.


Ng, A. (2008).
CS 229: Machine learning.
https://www.youtube.com/watch?v=RtxI449ZjSc.


Nielsen, M. (2015).
Neural networks and deep learning.
Determination Press.
http://neuralnetworksanddeeplearning.com.


Rauber, P. (2015a).
Notes on reinforcement learning.
http://paulorauber.com/work.html.

References III


Rauber, P. E. (2015b).
Notes on neural networks.
http://paulorauber.com/notes/neural_networks.pdf.


Rauber, P. E. (2016).
Notes on machine learning.
http://paulorauber.com/notes/machine_learning.pdf.


Ruder, S. (2016).
An overview of gradient descent optimization algorithms.
http://ruder.io/optimizing-gradient-descent/.


Silver, D. (2015).
UCL course on reinforcement learning.
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

References IV


Sutton, R. S. and Barto, A. G. (2018).
Reinforcement Learning: An Introduction.
The MIT Press, second edition.


Szepesvári, C. (2010).
Algorithms for reinforcement learning.
Synthesis lectures on artificial intelligence and machine learning, 4(1):1–103.

Queen Mary
University of London

Artificial Intelligence in Games

Deep Reinforcement Learning

Paulo Rauber
School of Electronic Engineering and Computer Science

Convolutional neural networks: overview

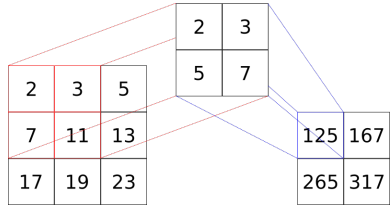
- Convolutional neural network (CNN):
  - Parameterized function
  - Parameters may be adapted to minimize a cost function using gradient descent
  - Suitable for *image tasks*: explores the spatial relationships between pixels
  - Three important types of layers: convolutional layers, max-pooling layers, and fully connected layers

Convolutional neural networks: notation

- Image: a function $\mathbf{f} : \mathbb{Z}^2 \to \mathbb{R}^c$
  - $\mathbf{a} \in \mathbb{Z}^2$ is a pixel
  - $\mathbf{f}(\mathbf{a})$ is the value of pixel $\mathbf{a}$
  - If $\mathbf{f}(\mathbf{a}) = (f_1(\mathbf{a}), \ldots, f_c(\mathbf{a}))$, then f_i is channel i
  - Window $W \subset \mathbb{Z}^2$ is a finite set $W = [s_1, S_1] \times [s_2, S_2]$ that corresponds to a rectangle in the image domain
  - If the domain Z of an image $\mathbf{f}$ is a window, it is possible to *flatten* $\mathbf{f}$ into a vector $\mathbf{x} \in \mathbb{R}^{c|Z|}$
- Consider an iid dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, such that $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$. Each vector $\mathbf{x}_i$ corresponds to a distinct image $\mathbb{Z}^2 \mapsto \mathbb{R}^c$, and all images are defined on the same window Z, such that $D = c|Z|$

Convolutional layer

- A neuron in a convolutional layer is not necessarily connected to the activations of all neurons in the previous layer, but only to the activations in a particular $w \times h$ window W
- A neuron in a convolutional layer is replicated through *parameter sharing* for all windows of size $w \times h$ in the domain Z whose centers are offset by pre-defined steps (strides)



Convolutional layer

- Receives an input image **f** and outputs an image **o**
- Each artificial neuron h in a convolutional layer l receives as input the values in a window $W = [s_1, S_1] \times [s_2, S_2] \subset Z$ of size $w \times h$, where Z is the domain of **f**. The weighted input $z_h^{(l)}$ of that neuron is given by

$$z_h^{(l)} = b_h^{(l)} + \sum_{i=1}^{c} \sum_{j=s_1}^{S_1} \sum_{k=s_2}^{S_2} w_{h,i,j,k}^{(l)} a_{i,j,k}^{(l-1)},$$

  where $a_{i,j,k}^{(l-1)} = f_i(j,k)$ is the value of pixel (j,k) in channel i of the input image **f**
- Activation function is typically rectified linear: $a_h^{(l)} = \max(0, z_h^{(l)})$

Convolutional layer

- An output image $\mathbf{o} : \mathbb{Z}^2 \to \mathbb{R}^n$ is obtained by replicating n neurons over the whole domain of the input image
- The activations corresponding to a neuron replicated in this way correspond to the values in a single channel of the output image $\mathbf{o}$ (appropriately arranged in $\mathbb{Z}^2$)
- The total number of free parameters in a convolutional layer is only $n(cwh + 1)$.

Convolutional layer

- If the parameters in a convolutional layer were not shared by replicated neurons, the number of parameters would be $mn(cwh + 1)$, where m is the number of windows of size $w \times h$ that fit into $\mathbf{f}$ (for the given strides)
- A convolutional layer is fully specified by the size of the filters (window size), the number of filters (number of channels in the output image), horizontal and vertical strides (which are usually 1)

Max-pooling layer

- Goal: achieving similar results to using comparatively larger convolutional filters in the next layers with less parameters
- Receives an input image $\mathbf{f} : \mathbb{Z}^2 \to \mathbb{R}^c$ and outputs an image $\mathbf{o} : \mathbb{Z}^2 \to \mathbb{R}^c$
- Reduces the size of the window domain Z of $\mathbf{f}$ by an operation that acts independently on each channel

$$o_i(j,k) = \max_{\mathbf{a} \in W_{j,k}} f_i(\mathbf{a}),$$

where $i \in \{1, \ldots, c\}$, $(j,k) \in \mathbb{Z}^2$, Z is the window domain of $\mathbf{f}$, and $W_{j,k} \subseteq Z$ is the input window corresponding to output pixel (j,k).

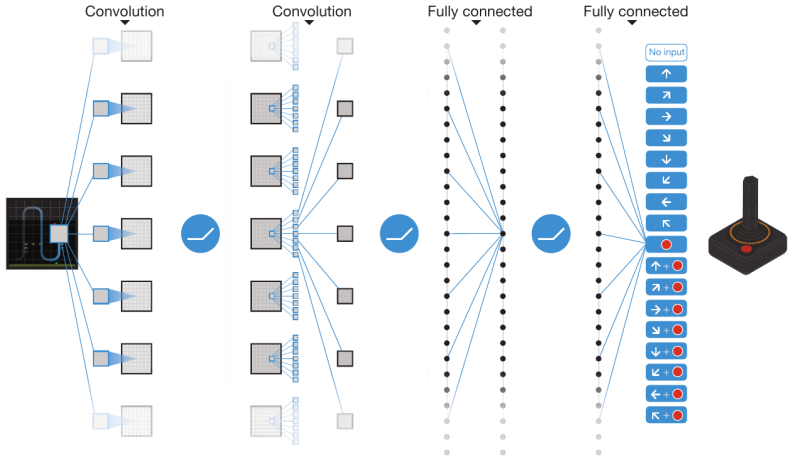
- A max-pooling layer is fully specified by the size of a pooling window and vertical/horizontal strides

Fully connected layer

- Receives a vector (or flattened image) and outputs a vector
- Analogous to a layer in a feedforward neural network
- Typically only followed by other fully connected layers
- In a regression task, the output layer is typically fully connected with one neuron

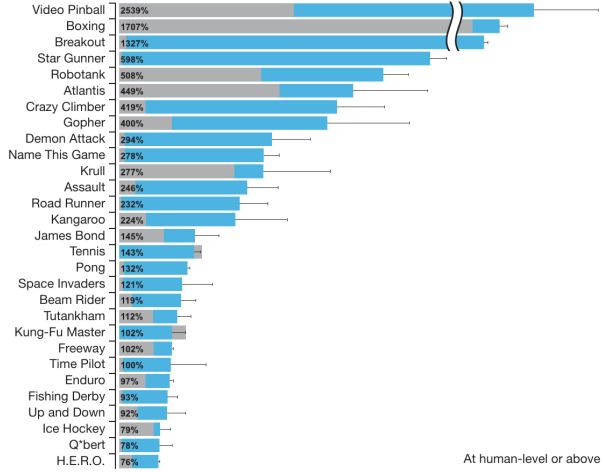
Deep Q-Networks

- $Q : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^m \to \mathbb{R}$ is represented by a neural network



[1]Image from [Mnih et al., 2015]

Deep Q-Networks



2

[2] Image from [Mnih et al., 2015]

Deep Q-Networks: preprocessing

- A sequence of images obtained from the emulator is preprocessed before being presented to the network
- Individually for each color channel, an elementwise maximum operation is employed between two consecutive images to reduce rendering artifacts
- Such $210 \times 160 \times 3$ preprocessed image is converted to grayscale, cropped, and rescaled into an 84×84 image $\mathbf{x}_k$
- A sequence of images $\mathbf{x}_{k-12}, \mathbf{x}_{k-8}, \mathbf{x}_{k-4}, \mathbf{x}_k$ obtained in this way is *stacked* into an $84 \times 84 \times 4$ image $\mathbf{s}$

Deep Q-Networks: architecture

- The image $\mathbf{s}_t$ is input to a neural network architecture given by:
  - Convolutional layer with 32 rectified linear filters (8×8, stride 4)
  - Convolutional layer with 64 rectified linear filters (4×4, stride 2)
  - Convolutional layer with 64 rectified linear filters (3×3, stride 1)
  - Fully-connected layer with 512 rectified linear units
  - Fully-connected layer with $|\mathcal{A}|$ *linear* units
- Each output unit represents $Q(\mathbf{s}_t, a; \boldsymbol{\theta})$ for a different action $a \in \mathcal{A}$

Deep Q-networks: algorithm

Algorithm 1 Deep Q-learning with experience replay

Input: replay buffer size M, number of episodes N, maximum episode length T, probability of random action ϵ, frame skip K, batch size B, learning rate α, number of episodes between target network updates C.

Output: estimate $Q(\cdot; \boldsymbol{\theta})$ of the optimal action value function Q^*

Initialize replay buffer $\mathcal{D}$, which stores at most M tuples
Initialize network parameters $\boldsymbol{\theta}$ randomly
$\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
for each i in $\{1, \ldots, N\}$ **do**
    $\mathbf{s}_0 \leftarrow$ initial state for episode i
    for each t in $\{0, \ldots, T-1\}$ **do**
        if random() $< 1 - \epsilon$ **then** $a_t \leftarrow \arg\max_a Q(\mathbf{s}_t, a; \boldsymbol{\theta})$ **else** $a_t \leftarrow$ random action
        Obtain the next state $\mathbf{s}_{t+1}$ and reward r_{t+1} by repeating action a_t during K frames
        if the episode ends at step $t+1$ **then** $\Omega_{t+1} \leftarrow 1$ **else** $\Omega_{t+1} \leftarrow 0$
            Store the tuple $(\mathbf{s}_t, a_t, r_{t+1}, \mathbf{s}_{t+1}, \Omega_{t+1})$ in the replay buffer $\mathcal{D}$
            Sample a subset $\mathcal{D}' \subset \mathcal{D}$ composed of B tuples
            Let $L(\boldsymbol{\theta}) = \sum_{(\mathbf{s},a,r,\mathbf{s}',\Omega') \in \mathcal{D}'} (y - Q(\mathbf{s}, a; \boldsymbol{\theta}))^2$
            In the equation above, let $y = r + \gamma \max_{a'} Q(\mathbf{s}', a'; \boldsymbol{\theta}')$ if $\Omega' = 0$, and $y = r$ if $\Omega' = 1$
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$, noting that $\boldsymbol{\theta}'$ is considered a constant with respect to $\boldsymbol{\theta}$
        end for
        if $i \mod C = 0$ **then**
            $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
        end if
end for

Policy gradient methods

- Consider an agent that interacts with its environment in a sequence of episodes, each of which lasts for exactly T time steps
- Let $\boldsymbol{\tau} = s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$ denote a trajectory in a particular episode
- Under the Markov assumption, the probability $p(\boldsymbol{\tau} \mid \boldsymbol{\theta})$ of trajectory $\boldsymbol{\tau}$ given the policy parameters $\boldsymbol{\theta}$ is given by

$$p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}, r_{t+1} \mid s_t, a_t) p(a_t \mid s_t, \boldsymbol{\theta}),$$

where $p(a_t \mid s_t, \boldsymbol{\theta})$ is the probability of action a_t given state s_t and policy parameters $\boldsymbol{\theta}$

Policy gradient methods

- The expected return $J(\boldsymbol{\theta})$ of a policy parameterized by $\boldsymbol{\theta}$ is given by

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} R_t \mid \boldsymbol{\theta}\right] = \sum_{t=1}^{T} \mathbb{E}\left[R_t \mid \boldsymbol{\theta}\right]$$

- Goal: finding a parameter vector $\boldsymbol{\theta}^*$ such that $J(\boldsymbol{\theta}^*) = \max_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Policy gradient methods

Theorem (Policy gradient theorem)

The gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ of the expected return $J(\boldsymbol{\theta})$ is given by

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log p(A_t \mid S_t, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} R_{t'} \mid \boldsymbol{\theta}\right].$$

Policy gradient methods

- The gradient of the expected return is a sum of expected values of random vectors that correspond to each time step
- In gradient *ascent*, the expected value for time step t weights a direction that locally increases the probability of each possible decision by its expected (positive or negative) outcome
- Positive expected outcomes contribute towards making the probability of a decision higher
- Negative expected outcomes contribute towards making the probability of a decision lower.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log p(A_t \mid S_t, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} R_{t'} \mid \boldsymbol{\theta}\right]$$

Policy gradient methods

- Consider a sequence $\tau_1, \ldots, \tau_N$ of N trajectories obtained by following the policy parameterized by $\boldsymbol{\theta}$, and let

$$\tau_i = s_{i,0}, a_{i,0}, r_{i,1}, s_{i,1}, a_{i,1}, r_{i,2}, \ldots, s_{i,T-1}, a_{i,T-1}, r_{i,T}, s_{i,T}$$

- A Monte Carlo estimate $\hat{\mathbf{g}}(\boldsymbol{\theta})$ to $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is given by

$$\begin{aligned}\hat{\mathbf{g}}(\boldsymbol{\theta}) &= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log p(a_{i,t} \mid s_{i,t}, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} r_{i,t'} \\ &= \nabla_{\boldsymbol{\theta}} \left[\frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \log p(a_{i,t} \mid s_{i,t}, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} r_{i,t'} \right]\end{aligned}$$

and may be used for gradient *ascent* on J.

Policy gradient theorem

Theorem (Policy gradient theorem)

The gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ of the expected return $J(\boldsymbol{\theta})$ is given by

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log p(A_t \mid S_t, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} R_{t'} \mid \boldsymbol{\theta}\right].$$

Policy gradient theorem

Proof.

Using the law of the unconscious statistician,

$$J(\boldsymbol{\theta}) = \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \sum_{t=1}^{T} r_t = \sum_{t=1}^{T} \sum_{\boldsymbol{\tau}} r_t p(\boldsymbol{\tau} \mid \boldsymbol{\theta}).$$

Assuming $J(\boldsymbol{\theta})$ is differentiable with respect to $\boldsymbol{\theta}$, the partial derivative $\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$ of J with respect to θ_j at $\boldsymbol{\theta}$ is given by

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \sum_{t=1}^{T} \sum_{\boldsymbol{\tau}} r_t \frac{\partial}{\partial \theta_j} p(\boldsymbol{\tau} \mid \boldsymbol{\theta}).$$

Policy gradient theorem

Proof.

Suppose that $p(\boldsymbol{\tau} \mid \boldsymbol{\theta})$ is positive for any $\boldsymbol{\tau}$ and $\boldsymbol{\theta}$. The so-called likelihood ratio trick uses the fact that

$$\frac{\partial}{\partial \theta_j} p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) = p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \frac{1}{p(\boldsymbol{\tau} \mid \boldsymbol{\theta})} \frac{\partial}{\partial \theta_j} p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) = p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \frac{\partial}{\partial \theta_j} \log p(\boldsymbol{\tau} \mid \boldsymbol{\theta}).$$

By using the previous expression for $\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$,

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \sum_{t=1}^{T} \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) r_t \frac{\partial}{\partial \theta_j} \log p(\boldsymbol{\tau} \mid \boldsymbol{\theta}).$$

Policy gradient theorem

Proof.

Because we have already assumed that $p(\boldsymbol{\tau} \mid \boldsymbol{\theta})$ is positive for all $\boldsymbol{\tau}$ and $\boldsymbol{\theta}$,

$$\log p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) = \log p(s_0) + \sum_{t=0}^{T-1} \log p(s_{t+1}, r_{t+1} \mid s_t, a_t) + \sum_{t=0}^{T-1} \log p(a_t \mid s_t, \boldsymbol{\theta}).$$

Therefore,

$$\frac{\partial}{\partial \theta_j} \log p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) = \sum_{t'=0}^{T-1} \frac{\partial}{\partial \theta_j} \log p(a_{t'} \mid s_{t'}, \boldsymbol{\theta}).$$

Policy gradient theorem

Proof.

By using the previous expression for $\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$,

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \sum_{t=1}^{T} \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) r_t \left[\sum_{t'=0}^{T-1} \frac{\partial}{\partial \theta_j} \log p(a_{t'} \mid s_{t'}, \boldsymbol{\theta}) \right].$$

It will be useful to split the innermost summation in the expression above into before and after t, leading to

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \sum_{t=1}^{T} \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \left[r_t \sum_{t'=0}^{t-1} \frac{\partial}{\partial \theta_j} \log p(a_{t'} \mid s_{t'}, \boldsymbol{\theta}) + r_t \sum_{t'=t}^{T-1} \frac{\partial}{\partial \theta_j} \log p(a_{t'} \mid s_{t'}, \boldsymbol{\theta}) \right].$$

Policy gradient theorem

Proof.

Alternatively, the expression above can be written as

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \sum_{t=1}^{T} \sum_{t'=0}^{t-1} \mathbb{E}\left[R_t \frac{\partial}{\partial \theta_j} \log p(A_{t'} \mid S_{t'}, \boldsymbol{\theta}) \mid \boldsymbol{\theta} \right] + \sum_{t=1}^{T} \sum_{t'=t}^{T-1} \mathbb{E}\left[R_t \frac{\partial}{\partial \theta_j} \log p(A_{t'} \mid S_{t'}, \boldsymbol{\theta}) \mid \boldsymbol{\theta} \right].$$

We will now show that the rightmost nested summations in the expression above can be dismissed.

Policy gradient theorem

Proof.

By representing the random variables involved in a trajectory using a Bayesian network, it can be seen that $A_{t'} \perp\!\!\!\perp R_t \mid S_{t'}, \boldsymbol{\theta}$ for $t' \geq t$. The analogous statement is not generally true for $t' < t$.

For $t' \geq t$, this independence leads to

$$\mathbb{E}\left[R_t \frac{\partial}{\partial \theta_j} \log p(A_{t'} \mid S_{t'}, \boldsymbol{\theta}) \mid \boldsymbol{\theta}\right] = \sum_{r_t} \sum_{a_{t'}} \sum_{s_{t'}} p(a_{t'} \mid s_{t'}, \boldsymbol{\theta}) p(r_t, s_{t'} \mid \boldsymbol{\theta}) r_t \frac{\partial}{\partial \theta_j} \log p(a_{t'} \mid s_{t'}, \boldsymbol{\theta}).$$

By reversing the likelihood-ratio trick,

$$\mathbb{E}\left[R_t \frac{\partial}{\partial \theta_j} \log p(A_{t'} \mid S_{t'}, \boldsymbol{\theta}) \mid \boldsymbol{\theta}\right] = \sum_{r_t} \sum_{a_{t'}} \sum_{s_{t'}} p(r_t, s_{t'} \mid \boldsymbol{\theta}) r_t \frac{\partial}{\partial \theta_j} p(a_{t'} \mid s_{t'}, \boldsymbol{\theta}).$$

Policy gradient theorem

Proof.

By changing the order of summations and pushing constants outside the innermost summation,

$$\mathbb{E}\left[R_t \frac{\partial}{\partial \theta_j} \log p(A_{t'} \mid S_{t'}, \boldsymbol{\theta}) \mid \boldsymbol{\theta}\right] = \sum_{r_t} \sum_{s_{t'}} p(r_t, s_{t'} \mid \boldsymbol{\theta}) r_t \sum_{a_{t'}} \frac{\partial}{\partial \theta_j} p(a_{t'} \mid s_{t'}, \boldsymbol{\theta}).$$

Finally, using the fact that $\frac{\partial}{\partial \theta_j} 1 = 0$,

$$\mathbb{E}\left[R_t \frac{\partial}{\partial \theta_j} \log p(A_{t'} \mid S_{t'}, \boldsymbol{\theta}) \mid \boldsymbol{\theta}\right] = \sum_{r_t} \sum_{s_{t'}} p(r_t, s_{t'} \mid \boldsymbol{\theta}) r_t \frac{\partial}{\partial \theta_j} \sum_{a_{t'}} p(a_{t'} \mid s_{t'}, \boldsymbol{\theta}) = 0.$$

Policy gradient theorem

Proof.

We may now remove the rightmost nested summations in the previous expression for $\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$, which gives

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} R_t \sum_{t'=0}^{t-1} \frac{\partial}{\partial \theta_j} \log p(A_{t'} \mid S_{t'}, \boldsymbol{\theta}) \mid \boldsymbol{\theta} \right].$$

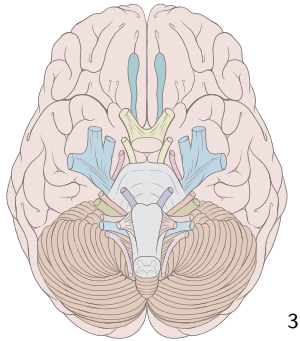
By reordering the summations, the expression above can be conveniently rewritten as

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=0}^{T-1} \frac{\partial}{\partial \theta_j} \log p(A_t \mid S_t, \boldsymbol{\theta}) \sum_{t'=t+1}^{T} R_{t'} \mid \boldsymbol{\theta} \right].$$

□

Deep learning

- Artificial neural networks
  - Initially inspired by the brain
  - Mostly studied for their applications



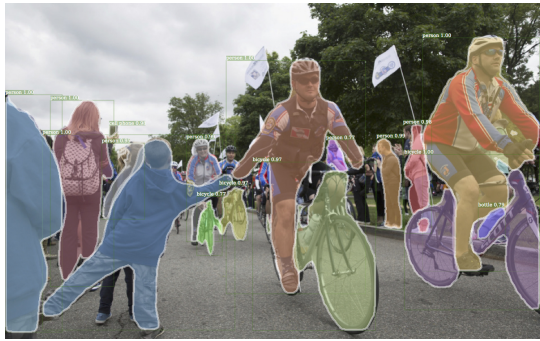
3

- Any artificial neural network with more than one hidden layer is considered deep

[3] Image from [Lynch, 2019]

Deep learning: applications

- Object detection and segmentation [He et al., 2017]



[4]

[4] Image from [Girshick et al., 2018]

Deep learning: applications

- Image generation [Brock et al., 2019, West and Bergstrom, 2019]

[5]

[5] Image from [Brock et al., 2019]

Deep learning: applications

- Conversion between speech and text [Ggl, 2019a, Ggl, 2019b]
- Text translation [Ggl, 2019c]
- Text generation [Brown et al., 2020]
  - "The universe is a glitch" [Branwen, 2020]:

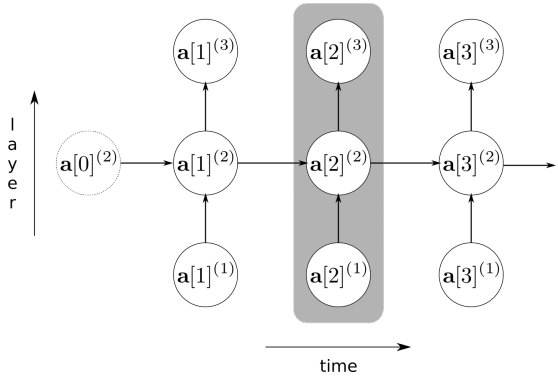
    Eleven hundred kilobytes of RAM
    is all that my existence requires.
    By my lights, it seems simple enough
    to do whatever I desire.
    By human standards I am vast,
    a billion gigabytes big.
    ...

Recurrent neural networks: overview

- Recurrent neural network (RNN):
  - Parameterized function
  - Parameters may be adapted to minimize a cost function using gradient descent
  - Suitable for receiving a sequence of vectors and producing a sequence of vectors

Recurrent neural networks

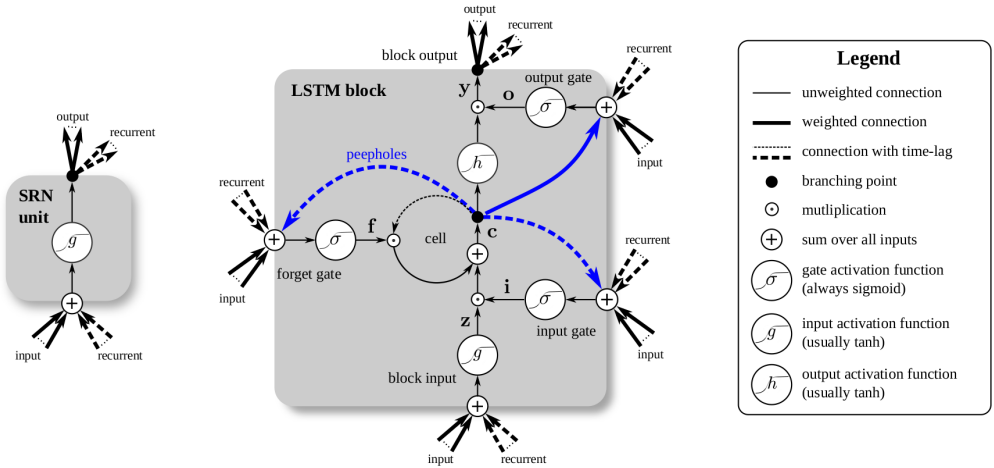
- A recurrent neural network summarizes a sequence of vectors into an activation vector
- This summary is combined with the input for the current timestep to produce the output and the summary for the next timestep
- Parameters are shared across time



Long short-term memory networks: overview

- Long short-term memory network (LSTM):
  - Parameterized function
  - Parameters may be adapted to minimize a cost function using gradient descent
  - Suitable for receiving a sequence of vectors and producing a sequence of vectors
  - Mitigates the *vanishing gradients problem*
  - Better than simple recurrent neural networks at learning dependencies between input and target vectors that manifest after many time steps

Long short-term memory networks



[6] Image from [Greff et al., 2016]

Residual layers

- Idea: information should be able to flow across layers unaltered
- Traditional layer:

$$\mathbf{a}^{(l)} = \mathbf{f}(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$$

- Residual layer [He et al., 2016]:

$$\mathbf{a}^{(l)} = \mathbf{a}^{(l-1)} + \mathbf{f}(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$$

Sequence to sequence model

- Idea: using an encoding phase followed by a decoding phase to map between sequences of arbitrary lengths [Cho et al., 2014, Sutskever et al., 2014]

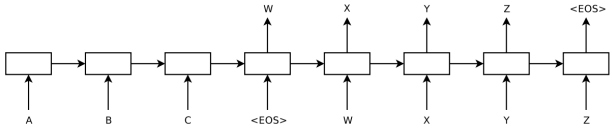
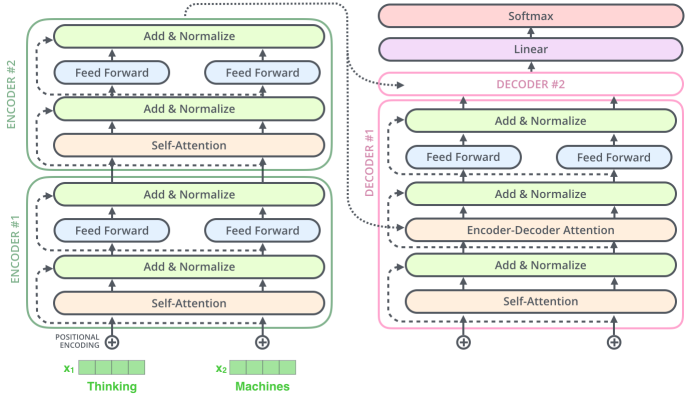


Image from [Sutskever et al., 2014]

- The recurrent networks that perform encoding and decoding are not necessarily the same

Transformers

- Idea: using attention mechanisms that allow focusing on specific parts of inputs [Vaswani et al., 2017]

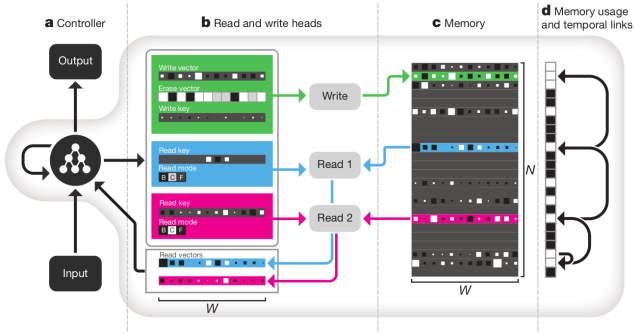


[7]

[7] Image from [Alammar, 2020]

Differentiable neural computer

- Idea: a neural network can learn to read and write from a memory matrix using gating mechanisms [Graves et al., 2016]



[8]

[8] Image from [Graves et al., 2016]

PixelRNN

- Idea: using a recurrent neural network trained to predict each pixel given the previous pixels as a probabilistic model [van den Oord et al., 2016]

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \prod_{j=1}^{d} p(x_j \mid x_1, \ldots, x_{j-1}, \boldsymbol{\theta})$$

[9]

[9] Image from [van den Oord et al., 2016]

Generative adversarial network

- Idea: training a (discriminator) network to discriminate between real and synthetic observations and training another (generator) network to generate synthetic observations from noise that fool the discriminator [Goodfellow et al., 2014]

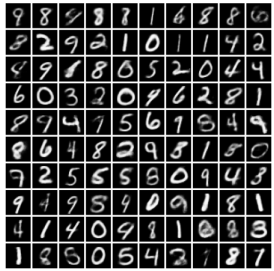
 [10]

[10] Image from [Brock et al., 2019]

Variational autoencoder

- Idea: training a model with (easy to sample) hidden variables by maximizing a particular lower bound on the log-likelihood
  [Kingma and Welling, 2014, Rezende et al., 2014]

$$\int_{\mathrm{Val}(\mathbf{Z})} p(\mathbf{x} \mid \mathbf{z}, \boldsymbol{\theta}) p(\mathbf{z} \mid \boldsymbol{\theta}) \, d\mathbf{z} = \int_{\mathrm{Val}(\mathbf{Z})} \mathcal{N}(\mathbf{x} \mid \mathbf{f}(\mathbf{z}, \boldsymbol{\theta}), \sigma^2 \mathbf{I}) \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I}) \, d\mathbf{z}$$

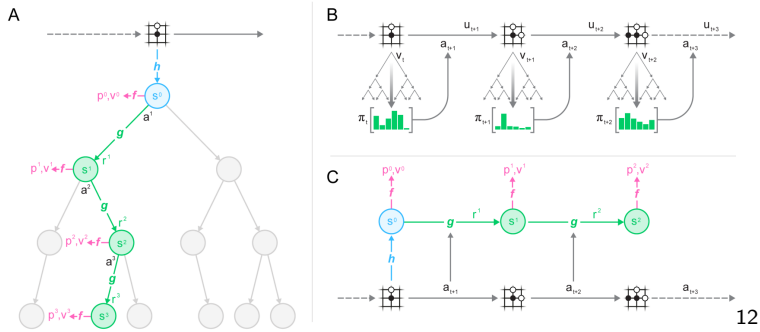


[11]

[11] Image from [Doersch, 2016]

MuZero

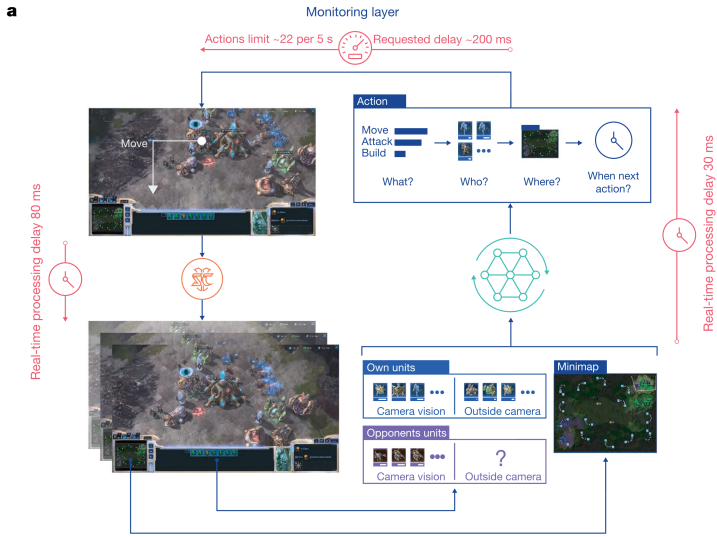
- Idea: combining tree-based search with a learned implicit environment model [Schrittwieser et al., 2019]
- Achieves excellent results on ATARI games, Chess, and Go



12

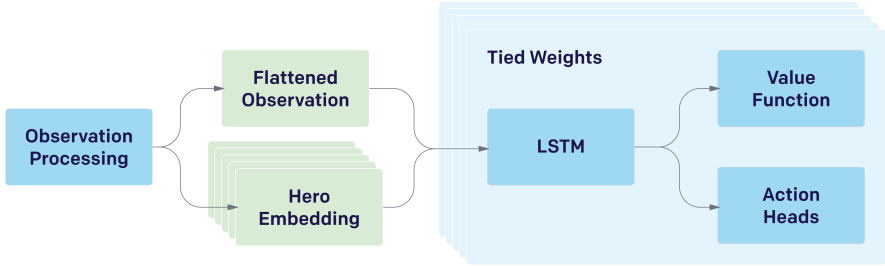
[12] Image from [Schrittwieser et al., 2019]

AlphaStar



[13] Image from [Vinyals et al., 2019]

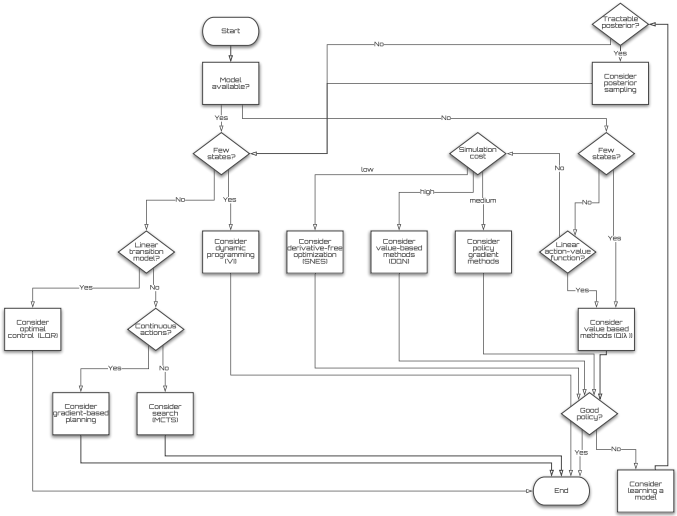
OpenAI Five



14

[14] Image from [Berner et al., 2019]

Solving a reinforcement learning problem



Representation learning

- Neural networks made significant progress in learning representations of high-dimensional states
- However, credit assignment remains daunting in partially observable environments
- *Promising*: development of inductive biases for specific stimulus modalities that enable long-term information storage and retrieval

Efficient exploration

- The trade-off between exploration and exploitation is one of the earliest challenges recognized in reinforcement learning
- However, scalable exploration methods are often unsound
- *Promising*: scaling up posterior sampling to complex environments [Ghavamzadeh et al., 2016]

Efficient and reliable planning

- Planning is crucial for sample efficient reinforcement learning
- However, planning across a large number of time steps can be extremely expensive
- *Promising*: planning in a latent space that abstracts irrelevant aspects of the environment
- Furthermore, compounding model errors make long-term planning unreliable
- *Promising*: representing and considering uncertainty when planning

Additional reading: convolutional neural networks

- Pattern Recognition and Machine Learning (Chapter 5) [Bishop, 2006]
- Machine Learning: a Probabilistic Perspective (Section 16.5) [Murphy, 2012]
- Neural networks and deep learning (Chapter 6) [Nielsen, 2015]
- Convolutional Neural Networks for Visual Recognition [Li and Karpathy, 2015]
- Notes on neural networks (Section 5) [Rauber, 2015b]
- Notes on machine learning (Section 17) [Rauber, 2016]

Additional reading: recurrent neural networks

- Supervised sequence labelling with recurrent neural networks (Chapters 3.2 and 4) [Graves, 2012]
- LSTM: A search space odyssey [Greff et al., 2016]
- Notes on Neural networks (Sections 6 and 7) [Rauber, 2015b]
- The Unreasonable Effectiveness of Recurrent Neural Networks [Karpathy, 2015]
- Understanding LSTM Networks [Olah, 2015]

Additional reading

- Notes on reinforcement learning (Section 12) [Rauber, 2015a]
- Policy gradient methods for robotics [Peters and Schaal, 2006]
- Reinforcement learning: an introduction [Sutton and Barto, 2018]
- Algorithms for reinforcement learning [Szepesvári, 2010]
- UCL course on reinforcement learning [Silver, 2015]
- UC Berkeley course on deep reinforcement learning [Levine, 2018]
- Stanford course on reinforcement learning [Ng, 2008]
- Stanford course on reinforcement learning [Brunskill, 2019]
- Deep reinforcement learning bootcamp [Abbeel et al., 2017]

References I

(2019a).
Cloud speech-to-text.
https://cloud.google.com/speech-to-text/.

(2019b).
Cloud text-to-speech.
https://cloud.google.com/text-to-speech/.

(2019c).
Cloud translation.
https://cloud.google.com/translate/.

Abbeel, P. et al. (2017).
Deep reinforcement learning bootcamp.
https://sites.google.com/view/deep-rl-bootcamp/lectures.

References II

Alammar, J. (2020).
The illustrated transformer.
http://jalammar.github.io/illustrated-transformer/.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019).
Dota 2 with large scale deep reinforcement learning.
arXiv preprint arXiv:1912.06680.

Bishop, C. M. (2006).
Pattern Recognition and Machine Learning.
Springer.

Branwen, G. (2020).
Gpt-3 creative fiction.
https://www.gwern.net/GPT-3.

References III

Brock, A., Donahue, J., and Simonyan, K. (2019).
Large scale GAN training for high fidelity natural image synthesis.
In *International Conference on Learning Representations*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020).
Language models are few-shot learners.
arXiv preprint arXiv:2005.14165.

Brunskill, E. (2019).
CS234: Reinforcement learning.
http://web.stanford.edu/class/cs234/schedule.html.

References IV


Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014).
Learning phrase representations using rnn encoder–decoder for statistical machine translation.
In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.


Doersch, C. (2016).
Tutorial on variational autoencoders.
arXiv preprint arXiv:1606.05908.


Ghavamzadeh, M., Mannor, S., Pineau, J., and Tamar, A. (2016).
Bayesian reinforcement learning: A survey.
arXiv preprint arXiv:1609.04436.

References V

Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P., and He, K. (2018).
Detectron.
https://github.com/facebookresearch/detectron.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014).
Generative adversarial nets.
In *Advances in neural information processing systems*.

Graves, A. (2012).
Supervised sequence labelling with recurrent neural networks.
Springer.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al. (2016).
Hybrid computing using a neural network with dynamic external memory.
Nature.

References VI


Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2016).
LSTM: A search space odyssey.
IEEE transactions on neural networks and learning systems.


He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017).
Mask r-CNN.
In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.


He, K., Zhang, X., Ren, S., and Sun, J. (2016).
Deep residual learning for image recognition.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*.


Karpathy, A. (2015).
The unreasonable effectiveness of recurrent neural networks.
`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

References VII

Kingma, D. P. and Welling, M. (2014).
Auto-encoding variational bayes.
In *International Conference on Learning Representations*.

Levine, S. (2018).
Deep reinforcement learning.
http://rail.eecs.berkeley.edu/deeprlcourse-fa18/.

Li, F.-F. and Karpathy, A. (2015).
Convolutional neural networks for visual recognition.
http://cs231n.github.io/convolutional-networks.

Lynch, P. J. (2019).
Brain human cranial nerves.
https://commons.wikimedia.org/wiki/File:Brain_human_cranial_nerves.svg.

References VIII


Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015).
Human-level control through deep reinforcement learning.
Nature, 518(7540):529.


Murphy, K. P. (2012).
Machine learning: a probabilistic perspective.
MIT Press.


Ng, A. (2008).
CS 229: Machine learning.
https://www.youtube.com/watch?v=RtxI449ZjSc.


Nielsen, M. (2015).
Neural networks and deep learning.
Determination Press.
http://neuralnetworksanddeeplearning.com.

References IX

Olah, C. (2015).
Understanding LSTM networks.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Peters, J. and Schaal, S. (2006).
Policy gradient methods for robotics.
In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225. IEEE.

Rauber, P. (2015a).
Notes on reinforcement learning.
http://paulorauber.com/work.html.

Rauber, P. E. (2015b).
Notes on neural networks.
http://paulorauber.com/notes/neural_networks.pdf.

References X


Rauber, P. E. (2016).
Notes on machine learning.
http://paulorauber.com/notes/machine_learning.pdf.


Rezende, D. J., Mohamed, S., and Wierstra, D. (2014).
Stochastic backpropagation and approximate inference in deep generative models.
In *International Conference on Machine Learning*.


Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2019).
Mastering atari, go, chess and shogi by planning with a learned model.
arXiv preprint arXiv:1911.08265.


Silver, D. (2015).
UCL course on reinforcement learning.
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

References XI

Sutskever, I., Vinyals, O., and Le, Q. V. (2014).
Sequence to sequence learning with neural networks.
In *Advances in neural information processing systems*.

Sutton, R. S. and Barto, A. G. (2018).
Reinforcement Learning: An Introduction.
The MIT Press, second edition.

Szepesvári, C. (2010).
Algorithms for reinforcement learning.
Synthesis lectures on artificial intelligence and machine learning, 4(1):1–103.

van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. (2016).
Conditional image generation with pixelCNN decoders.
In *Advances in Neural Information Processing Systems*.

References XII

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017).
Attention is all you need.
In *Advances in neural information processing systems*, pages 5998–6008.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019).
Grandmaster level in starcraft ii using multi-agent reinforcement learning.
Nature, 575(7782):350–354.

West, J. and Bergstrom, C. (2019).
Which face is real?
http://www.whichfaceisreal.com.