Introduction to Reinforcement Learning

E. Le Pennec

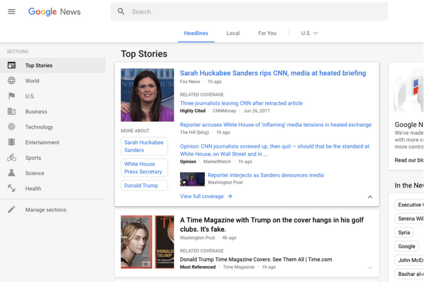
ÉCOLE POLYTECHNIQUE

M2DS - Fall 2022

Outline

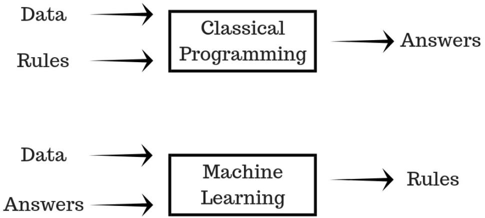
Outline

Machine Learning



Sources: MyCarDoesWhat.org/theverge.com/Rutgers Prep School

Machine Learning



A definition by Tom Mitchell (`http://www.cs.cmu.edu/~tom/`)

A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E.

Source: C. Barabasi

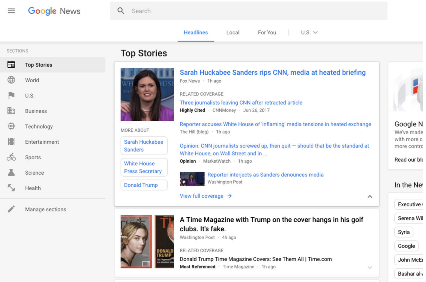
Object Detection

A detection algorithm:

- **Task**: say if an object is present or not in the image
- **Performance**: number of errors
- **Experience**: set of previously seen labeled images

Source: MyCarDoesWhat.org

Article Clustering



An article clustering algorithm:

- **Task**: group articles corresponding to the same news
- **Performance**: quality of the clusters
- **Experience**: set of articles

Source: theverge.com

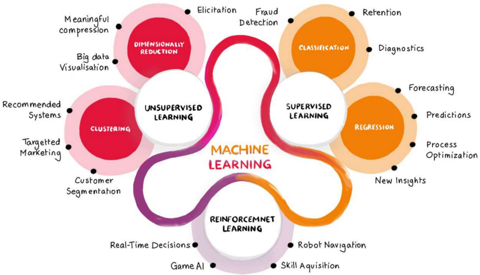
A Robot that Learns

Source: Rutgers Prep School

A robot endowed with a set of sensors playing football:

- **Task**: play football
- **Performance**: score evolution
- **Experience**:
  - past games
  - current environment and action outcome,

Three Kinds of Learning



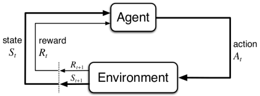
Unsupervised Learning	Supervised Learning	Reinforcement Learning
Task: Clustering/DR	**Task:** Prediction/Classification	**Task:** Action
Performance: Quality	**Performance:** Average error	**Performance:** Total reward
Experience: Raw dataset (No Ground Truth)	**Experience:** Good Predictions (Ground Truth)	**Experience:** Reward from env. (Interact. with env.)

- **Timing:** Offline/Batch (learning from past data) vs Online (continuous learning)

Source: BCG

Outline

Reinforcement Learning



Reinforcement Learning Setting

- **Env.:** provides a reward and a new state for any action.
- **Agent policy π:** choice of an action A_t from the state S_t.
- **Total reward:** (discounted) sum of the rewards.

Questions

- **Policy evaluation:** how to evaluate the expected reward of a policy knowing the environment?
- **Planning:** how to find the best policy knowing the environment?
- **Reinforcement Learning:** how to find the best policy without knowing the environment?

Outline

The Agent-Environment Interface

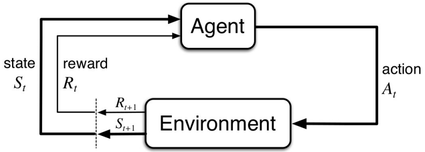


Figure 3.1: The agent–environment interaction in a Markov decision process.

MDP

- At time step $t \in \mathbb{N}$:
  - State $S_t \in \mathcal{S}$: representation of the environment
  - Action $A_t \in \mathcal{A}(S_t)$: action chosen
  - Reward $R_{t+1} \in \mathcal{R}$: instantaneous reward
  - New state S_{t+1}
- Dynamic entirely defined by

$$\mathbb{P}(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a) = p(s', r | s, a)$$

- Finite MDP: $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ are finite.

Returns and Episodes

Return

- (Discounted) Return:

$$G_t = \sum_{t'=t+1}^{T} \gamma^{t'-(t+1)} R_{t'}$$

- Recursive property

$$G_t = R_{t+1} + \gamma G_{t+1}$$

- Finiteness if $|R| \leq M$

$$|G_t| \leq \begin{cases} (T-(t+1))M & \text{if } T < \infty \\ M\frac{1}{1-\gamma} & \text{otherwise} \end{cases}$$

- Not well defined if $T = \infty$ and $\gamma = 1$.

Policies and Value Functions

Policy and Value Functions

- Policy: $\pi(a|s)$
- Value function:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]$$

- Action value function:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

Two natural problems

- Policy evaluation: compute v_π given π.
- Planning: find $\pi^\star$ such that $v_{\pi^\star}(s) \geq v_\pi(s)$ for all s and π.

- Those objects may not exist in general!
- Can be traced back to the 50's!

Outline

Policy Evaluation by Bellman Backup

$G_t = R_{t+1} + \gamma G_{t+1}$

Fixed Point Property

- Bellman Equation
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_\pi(s')\right] = \mathcal{T}_\pi(v_\pi)(s)$$

- Linear equation that can be solved.

Policy Evaluation by Dynamic Programming

- Fixed point iterative algorithm: $v_{k+1}(s) = \mathcal{T}_\pi(v_k)(s)$

- Converge if $T < \infty$ or $\gamma < 1$.

Planning by Policy Improvement

Policy Improvement Property

- If π' is such that $\forall s, q_\pi(s, \pi'(s)) \geq v_\pi(s)$ then $v_{\pi'} \geq v_\pi$.

- ϵ-greedy improvement among ϵ-policy: classical improvement degraded by picking uniformly the action with probability ϵ

Policy Iteration Algorithm

- Compute v_{π_k}
- Greedy update:

$$\begin{aligned}\pi_{k+1}(s) &= \operatorname*{argmax}_a q_{\pi_k}(s, a) \\ &= \operatorname*{argmax}_a \sum_{s', r} p(s', r|s, a) \left(r + \gamma v_{\pi_k}(s')\right)\end{aligned}$$

- If $\pi' = \pi$ after a greedy update $v_{\pi_{k+1}} = v_{\pi_k} = v_*$.

- Convergence in finite time in the finite setting.

Planning by Bellman Backup

Fixed Point Property

- Bellman Equation

$$v_*(s) = \max_a \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_*(s') \right] = \mathcal{T}_*(v_*)(s)$$

- Linear programming problem that can be solved.

Policy Evaluation by Dynamic Programming

- Iterative algorithm: $v_{k+1}(s) = \mathcal{T}_*(v_k)(s)$

- Converge if $T < \infty$ or $\gamma < 1$.
- Amount to improve the policy after only one step of policy evaluation.

Planning by Bellman Backup

Q-value and enhancement

- Q-value:

$$q_\pi(s,a) = \sum_{s'} \sum_{r} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a') \right]$$

- Easy policy enhancement: $\pi'(s) = \operatorname*{argmax}_a q(s,a)$

Fixed Point Property

- Bellman Equation

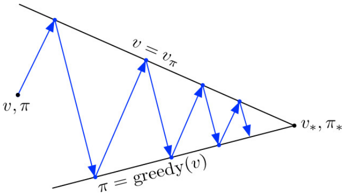
$$q_*(s,a) = \sum_{s'} \sum_{r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s',a') \right] = \mathcal{T}_*(q_*)(s,a)$$

- Linear programming problem that can be solved.

Policy Evaluation by Dynamic Programming

- Iterative algorithm: $q_{k+1}(s,a) = \mathcal{T}_*(q_k)(s,a)$

Generalized Policy Iteration



Generalized Policy Iteration

- Consists of two simultaneous interacting processes:
  - one making a value function consistent with the current policy (policy evaluation)
  - one making the policy greedy with respect to the current value function (policy improvement)

- Stabilizes only if one reaches the optimal value/policy pair.
- Asynchronous update are possible provided every state(/action) is visited infinitely often.
- Very efficient but requires the knowledge of the transition probabilities.

Outline

Reinforcement Learning

Reinforcement Learning - Sutton (98)

- An agent takes actions in a sequential way, receives rewards from the environment and tries to maximize his long-term (cumulative) reward.

Reinforcement Learning

- MDP setting with cumulative reward.
- Planning problem.
- Environment known only through interaction, i.e. some sequences $\cdots S_t A_t R_{t+1} S_{t+1} A_{t+1} \cdots$.

MC Methods

- Back to $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$.
- Monte Carlo:
  - Play several episodes using policy π.
  - Average the returns obtained after any state s.

- Good theoretical properties provided every states are visited asymptotically *infinitely often*.

Extensions

- Extension to off-policy setting (behavior policy $b \neq$ target policy π) with importance sampling.
- Extension to planning with policy improvement steps (estimating q_π instead of v_π)

- No theoretical results for the last case.
- Need to wait until the end of an episode to update anything...

Bootstrap and TD

- Rely on

$$\begin{aligned} v_\pi(s) &= \mathcal{T}_\pi v_\pi(s) \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \end{aligned}$$

- Temporal Difference: stochastic approximation scheme

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

- Update occurs at each time step.
- Can be proved to converge (under some assumption on α).

- Combine the best of Dynamic Programing and MC.
- Can be written in term of Q:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right)$$

SARSA and Q Learning

- How to use this principle to obtain the best policy?

SARSA: Planning by Prediction and Improvement (online)

- Update Q following the current policy π
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right)$$
- Update π by policy improvement.

- May not converge if one use a greedy policy update

Q Learning: Planning by Bellman Backup (off-line)

- Update Q following the behavior policy b
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$
- No need to use importance sampling correction for depth 1 update.

- Proof of convergence in both cases.

Variations

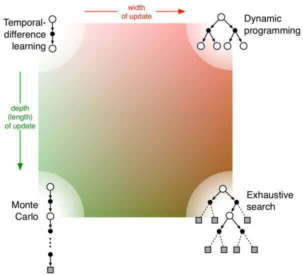


Figure 8.11: A slice through the space of reinforcement learning methods, highlighting the two of the most important dimensions explored in Part I of this book: the depth and width of the updates.

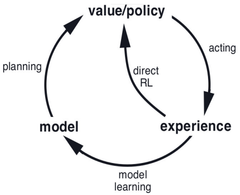
Depth

- Number of steps in the update.

Width

- Number of states/actions considered at each step.

Planning and Learning



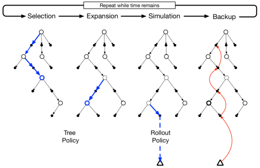


Figure 8.10: Monte Carlo Tree Search. When the environment changes to a new state, MCTS executes as many iterations as possible before an action needs to be selected, incrementally building a tree whose root node represents the current state. Each iteration consists of the four operations **Selection**, **Expansion** (though possibly skipped on some iterations), **Simulation**, and **Backup**, as explained in the text and illustrated by the bold arrows in the trees. Adapted from Chaslot, Bakkes, Szita, and Spronck (2008).

Planning and Models

- Planning can combine a model estimation (DP) and direct learning (RL).

Real Time Planning

- Planning can be made online starting from the current state.

- Curse of dimensionality: methods are hard to use when the cardinality of the states and the actions are large!

Outline

Value Function Approximation

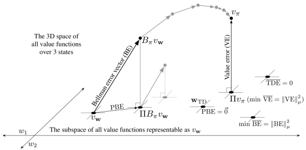
Value Function Approximation

- **Idea:** replace $v(s)$ by a parametric $\hat{v}(s, \boldsymbol{w})$.
- **Issues:**
  - Which approximation functions?
  - How to define the quality of the approximation?
  - How to estimate $\boldsymbol{w}$?

Approximation functions

- Any parametric (or kernel based) approximation could be used.
- Most classical choice:
  - Linear approximation.
  - Deep Neural Nets...

Approximation Quality



- How define when $\hat{v}(\cdot, \boldsymbol{w})$ is close to v_π (or v_*)

Prediction(/Control)

- Prediction objective:
$$\sum_s \mu(s)(v_\pi(s) - \hat{v}(s, \boldsymbol{w}))^2$$
- Bellman Residual:
$$\sum_s \mu(s)(\mathcal{T}_\pi \hat{v}(s, \boldsymbol{w}) - \hat{v}(s, \boldsymbol{w}))^2$$
or its projection...

- **Issue:** Neither v_π or $\mathcal{T}_\pi$ are known...

Online Gradient and Semi-Gradient

Online Prediction

- SGD algorithm on $\boldsymbol{w}$:
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t) \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$
- MC approximation (still SGD):
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(G_t - \hat{v}(S_t, \boldsymbol{w}_t) \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$
- TD approximation (not SGD anymore):
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t) \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$

- Deeper or wider scheme possible.

Online Control

- SARSA-like algorithm:
  - Prediction step as previously with the current policy
  $$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t) - \hat{q}(S_t, A_t, \boldsymbol{w}) \right) \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$
  - ϵ-greedy update of the current policy

Offline Control with Approximation

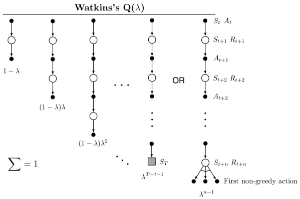


Figure 12.12: The backup diagram for Watkins's Q(λ). The series of component updates ends either with the end of the episode or with the first nongreedy action, whichever comes first.

Offline Control

- Q-Learning like algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \boldsymbol{w}_t) - \hat{q}(S_t, A_t, \boldsymbol{w}_t) \right) \times \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

  with an arbitrary policy b.
- Deeper formulation using importance sampling possible.

- **Issue:** Hard to make it converge in general!

Deadly Triad

Sutton-Barto's Deadly Triad

- **Function Approximation**
- **Bootstrapping**
- **Off-policy training**

Deep Q-Learning Stabilization Tricks

- Memory replay: sample from a set of episodes (sampled model)
- Frozen Q: use previous weights in the max (two scale algorithms)
- Amount to an approximate value iteration algorithm!

Policy Based Approach

- Other approach with a **parametric policy**.

Parametric Policy Setting

- New goal:

$$J(\theta) = \sum_s \mu_{\pi_\theta}(s) v_{\pi_\theta}(s)$$
$$= \sum_s \mu_{\pi_\theta}(s) \sum_a \pi_\theta(a|s) q_{\pi_\theta}(s, a)$$

- Stochastic gradient:

$$\widehat{\nabla} J(\theta) = \sum_t \gamma^t \nabla \log \pi_\theta(A_t|S_t)(q_{\pi_\theta}(S_T, A_T) - v_{\pi_\theta}(S_t))$$
$$\sum_t \gamma^t \nabla \log \pi_\theta(A_t|S_t) a_{\pi_\theta}(S_T, A_T)$$

- On policy algorithm if we can estimate $a_{\pi_\theta}(S_T, A_T) = q_{\pi_\theta}(S_T, A_T) - v_{\pi_\theta}(S_t)$ for instance by MC.

Actor-Critic

- Simultaneous parameterization of
  - the policy π by $\boldsymbol{\theta}$,
  - the value function q by $\boldsymbol{w}$
- Simultaneous update:

$$\delta_t = R_t + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w})$$

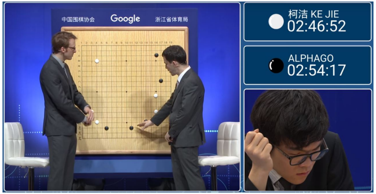
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} + \beta \delta_t \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_{t+1} + \alpha \delta_t \nabla \hat{q}(S_t, A_t, \boldsymbol{w})$$

- Can be adapted to continuous actions.
- Basis for SOTA algorithm.
- Can be hybrided with deep Q-learning.

Outline

AlphaGo



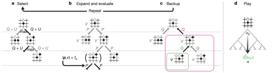
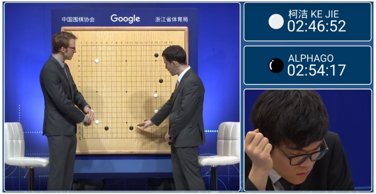


Figure 2 | MCTS in AlphaGo Zero. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). **b,** The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s,\cdot),V(s)) = f_\theta(s)$; the vector of P values are stored in the outgoing edges from s. **c,** Action value Q is updated to track the mean of all evaluations V in the subtree below that action. **d,** Once the search is complete, search probabilities π are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

AlphaGo

- Enhanced MCTS technique using a Deep NN for both the value function and the policy.
- Rollout policy and initial value network by supervised learning on a huge database.
- Enhancement of the value network using Actor/Critic RL on self-play.

AlphaGo



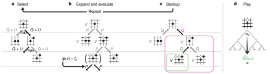


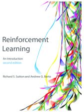
Figure 2 | MCTS in AlphaGo Zero. **a**, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). **b**, The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s,\cdot),V(s)) = f_\theta(s)$; the vector of P values are stored in the outgoing edges from s. **c**, Action value Q is updated to track the mean of all evaluations V in the subtree below that action. **d**, Once the search is complete, search probabilities π are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

AlphaGo Zero

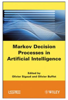
- No supervised initialization but only self-play.
- Alternate
  - MCTS with a current policy.
  - Gradient descent toward the resulting MCTS policy
- Much shorter training time and better performance!

Outline

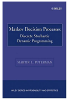
References



R. Sutton and A. Barto.
Reinforcement Learning, an Introduction (2nd ed.)
MIT Press, 2018

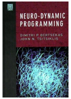


O. Sigaud and O. Buffet.
Markov Decision Processes in Artificial Intelligence.
Wiley, 2010

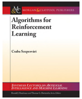


M. Puterman.
Markov Decision Processes. Discrete Stochastic Dynamic Programming.
Wiley, 2005

T. Lattimore and Cs. Szepesvári.
Bandit Algorithms.
Cambridge University Press, 2019



D. Bertsekas and J. Tsitsiklis.
Neuro-Dynamic Programming.
Athena Scientific, 1996



Cs. Szepesvári.
Algorithms for Reinforcement Learning.
Morgan & Claypool, 2010

Licence and Contributors

Contributors

- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.