

Introduction to Reinforcement Learning

E. Le Pennec



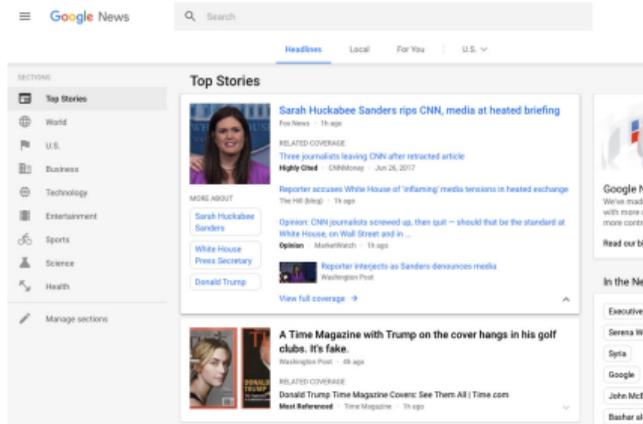
M2 DS - Fall 2021

Outline



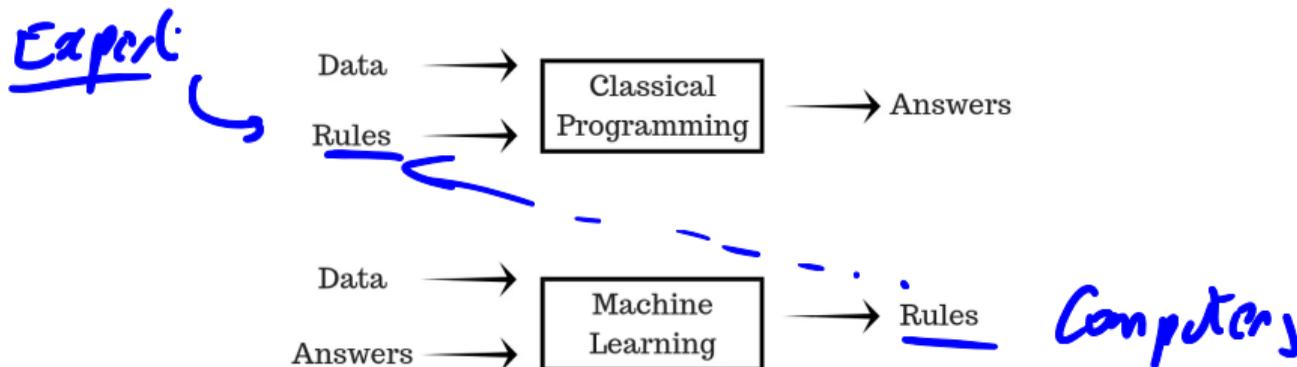
- 1 Machine Learning
- 2 Reinforcement Learning
- 3 Markov Decision Processes
- 4 Dynamic Programming
- 5 Reinforcement Setting
- 6 Reinforcement and Approximation
- 7 AlphaGo
- 8 References

- 1 Machine Learning
- 2 Reinforcement Learning
- 3 Markov Decision Processes
- 4 Dynamic Programming
- 5 Reinforcement Setting
- 6 Reinforcement and Approximation
- 7 AlphaGo
- 8 References



Google News interface showing top stories and related coverage. The main headline is "Sarah Huckabee Sanders rips CNN, media at heated briefing" with a sub-headline "Reporter accuses White House of 'inflaming' media tensions in heated exchange". Other stories include "A Time Magazine with Trump on the cover hangs in his golf clubs. It's fake." and "Reporter interviews as Sanders denounces media".





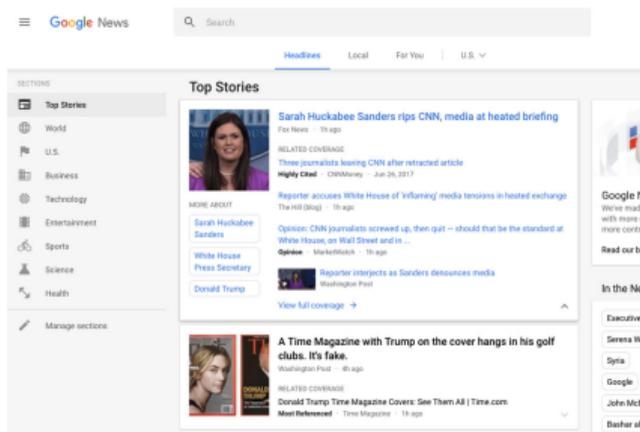
A definition by Tom Mitchell (<http://www.cs.cmu.edu/~tom/>)

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.



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



An article clustering algorithm:

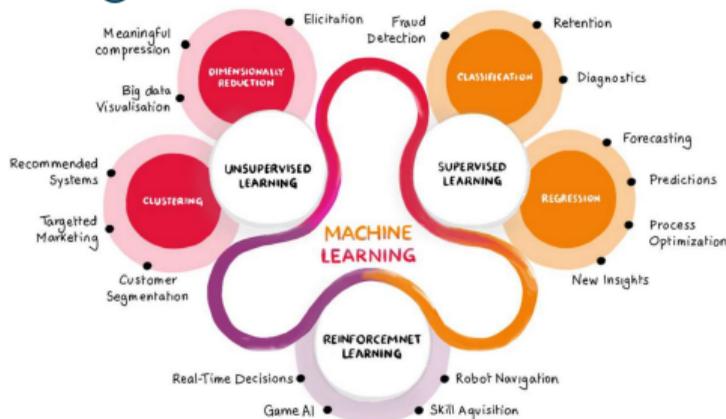
- **Task:** group articles corresponding to the same news
- **Performance:** quality of the clusters
- **Experience:** set of articles



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

- **Task:** Clustering/DR
- **Performance:** Quality
- **Experience:** Raw dataset (No Ground Truth)

Supervised Learning

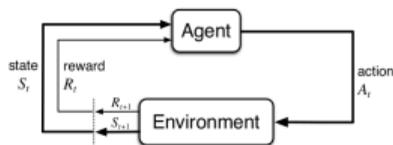
- **Task:** Prediction/Classification
- **Performance:** Average error
- **Experience:** Good Predictions (Ground Truth)

Reinforcement Learning

- **Task:** Action
- **Performance:** Total reward
- **Experience:** Reward from env. (Interact. with env.)

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

- 1 Machine Learning
- 2 Reinforcement Learning**
- 3 Markov Decision Processes
- 4 Dynamic Programming
- 5 Reinforcement Setting
- 6 Reinforcement and Approximation
- 7 AlphaGo
- 8 References



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?

- 1 Machine Learning
- 2 Reinforcement Learning
- 3 Markov Decision Processes**
- 4 Dynamic Programming
- 5 Reinforcement Setting
- 6 Reinforcement and Approximation
- 7 AlphaGo
- 8 References

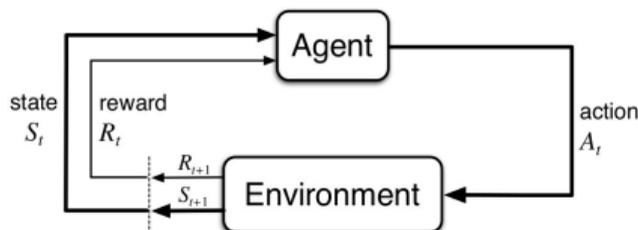


Figure 3.1: The agent–environment interaction in a Markov decision process.

MDP

- At time step $t \in \mathcal{N}$:
 - 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.

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 $\gamma = 1$.

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 π^* such that $v_{\pi^*}(s) \geq v_{\pi}(s)$ for all s and π .
- Those objects may not exist in general!
- Can be traced back to the 50's!

- 1 Machine Learning
- 2 Reinforcement Learning
- 3 Markov Decision Processes
- 4 Dynamic Programming**
- 5 Reinforcement Setting
- 6 Reinforcement and Approximation
- 7 AlphaGo
- 8 References

Fixed Point Property

- Bellman Equation

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

- 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 $\gamma < 1$.

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:

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

- If $\pi' = \pi$ after a greedy update $v_{\pi_{k+1}} = v_{\pi_k} = v_*$.
- Convergence in finite time in the finite setting.

Fixed Point Property

- Bellman Equation

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

- 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.

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

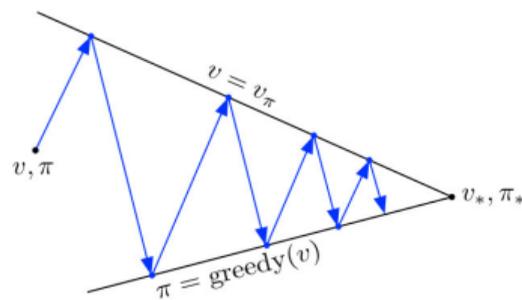
- Bellman Equation

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

- Linear programming problem that can be solved.

Policy Evaluation by Dynamic Programming

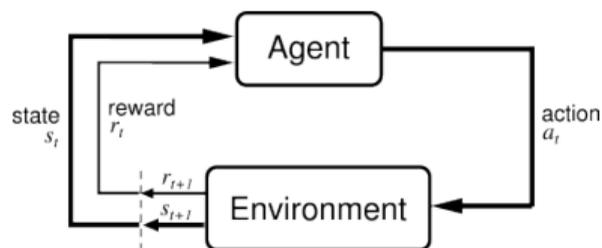
- Iterative algorithm: $q_{k+1}(s, a) = \mathcal{T}_*(q_k)(s, a)$



Generalized Policy Iteration

- 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.

- 1 Machine Learning
- 2 Reinforcement Learning
- 3 Markov Decision Processes
- 4 Dynamic Programming
- 5 Reinforcement Setting**
- 6 Reinforcement and Approximation
- 7 AlphaGo
- 8 References



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 $\dots S_t A_t R_{t+1} S_{t+1} A_{t+1} \dots$.

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
- 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

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

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

- Combine the best of Dynamic Programming and MC.

- Can be written in term of Q :

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

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

SARSA: Planning by Prediction and Improvement (online)

- Update Q following the current policy π

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

- 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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

- No need to use importance sampling correction for depth 1 update.

- Proof of convergence in both cases.

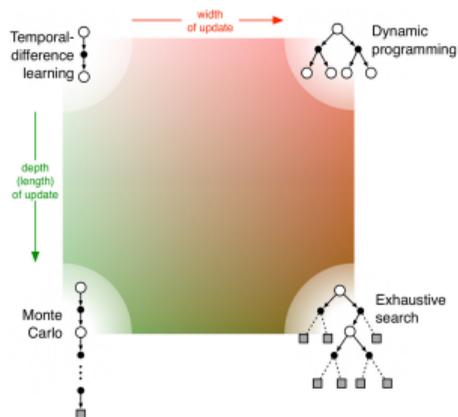


Figure 8.11: A slice through the space of reinforcement learning methods, highlighting the two of the most important dimensions explored in Part I of this book: the depth and width of the updates.

Depth

- Number of steps in the update.

Width

- Number of states/actions considered at each step.

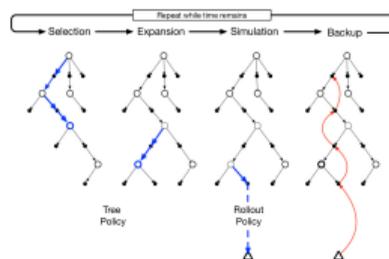
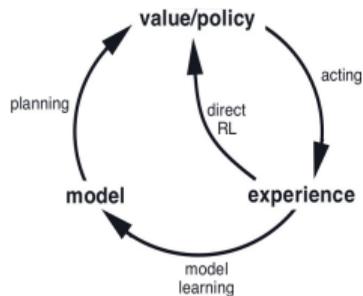


Figure 8.10: Monte Carlo Tree Search. When the environment changes to a new state, MCTS occurs as many iterations as possible before an action needs to be selected, incrementally building a tree whose root node represents the current state. Each iteration consists of the four operations **Selection**, **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, Bonet, 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!

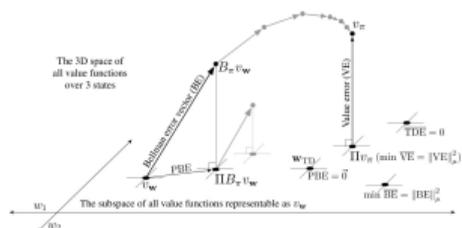
- 1 Machine Learning
- 2 Reinforcement Learning
- 3 Markov Decision Processes
- 4 Dynamic Programming
- 5 Reinforcement Setting
- 6 Reinforcement and Approximation**
- 7 AlphaGo
- 8 References

Value Function Approximation

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

Approximation functions

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



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

Prediction(/Control)

- Prediction objective:

$$\sum_s \mu(s) (v_\pi(s) - \hat{v}(s, \mathbf{w}))^2$$

- Bellman Residual:

$$\sum_s \mu(s) (\mathcal{T}_\pi \hat{v}(s, \mathbf{w}) - \hat{v}(s, \mathbf{w}))^2$$

or its projection...

- **Issue:** Neither v_π or \mathcal{T}_π are known...

Online Prediction

- SGD algorithm on \mathbf{w} :

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha (v_\pi(S_t) - \hat{v}(S_t, \mathbf{w})) \nabla \hat{v}(S_t, \mathbf{w})$$

- MC approximation (still SGD):

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha (G_t - \hat{v}(S_t, \mathbf{w})) \nabla \hat{v}(S_t, \mathbf{w})$$

- TD approximation (not SGD anymore):

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha (R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w})) \nabla \hat{v}(S_t, \mathbf{w})$$

- Deeper or wider scheme possible.

Online Control

- SARSA-like algorithm:

- Prediction step as previously with the current policy

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha (R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \nabla \hat{q}(S_t, A_t, \mathbf{w})$$

- ϵ -greedy update of the current policy

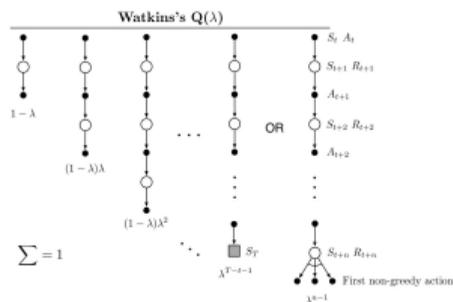


Figure 12.12: The backup diagram for Watkins's $Q(\lambda)$. The series of component updates ends either with the end of the episode or with the first nongreedy action, whichever comes first.

Offline Control

- Q-Learning like algorithm:

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

with an arbitrary policy b .

- Deeper formulation using importance sampling possible.
- **Issue:** Hard to make it converge in general!

Sutton-Barto's Deadly Triad

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

Stabilization Tricks

- (Back to policy iteration),
- Memory replay: sample from a set of episodes
- Frozen Q : use the previous weights in the max
- Clip/normalize rewards. . .

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

Actor-Critic

- Simultaneous parameterization of
 - the policy π by θ ,
 - the value function s by w
- Simultaneous update:

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

$$\theta_{t+1} = \theta_t + \alpha \delta_t \frac{\nabla \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \nabla \hat{v}(S_t, \mathbf{w})$$

- Online approach
- Can be adapted to continuous actions.

- 1 Machine Learning
- 2 Reinforcement Learning
- 3 Markov Decision Processes
- 4 Dynamic Programming
- 5 Reinforcement Setting
- 6 Reinforcement and Approximation
- 7 AlphaGo**
- 8 References

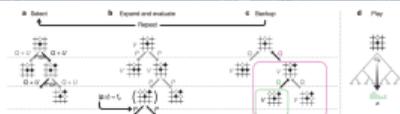


Figure 21: MCTS in AlphaGo. a. Each simulation traverses the tree by selecting the edge with maximum action value Q . b. Q plus an upper confidence bound U that depends on a neural prior probability P and visit count. If the best edge (which is incremented once traversed), the leaf node is expanded and the associated position is evaluated by the neural network V_{π} . Q is the mean of V_{π} over n simulations. c. The outgoing edges from a . Action value Q is updated to track the mean of all evaluations V for the subtree below that action. d. Once the search is complete, search probability P are updated, proportional to Q^n , where n is the visit count of each node from the root state and n 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.

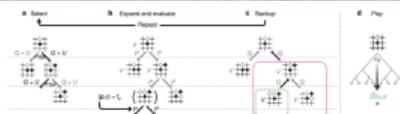


Figure 21 MCTS in AlphaGo Zero. a. Each simulation traverses the tree by selecting the edge with maximum action value Q ; this is upper confidence bound (UCB) that depends on a neural prior probability P and visit count. If the last edge (which is incremented once traversed), the leaf node is expanded and the associated position is evaluated by the neural network. $P(a_i | s)$, $Q(a_i | s)$ the mean of P values are stored in the outgoing edges from a_i . Action value Q is updated to track the mean of all evaluations V for the states below that action. d. Once the search is complete, search probabilities π are returned, proportional to Q^k , where k is the visit count of each move from the root state and k 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!

- 1 Machine Learning
- 2 Reinforcement Learning
- 3 Markov Decision Processes
- 4 Dynamic Programming
- 5 Reinforcement Setting
- 6 Reinforcement and Approximation
- 7 AlphaGo
- 8 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



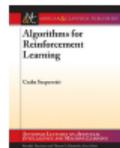
M. Puterman.
Markov Decision Processes. Discrete Stochastic Dynamic Programming.
Wiley, 2005



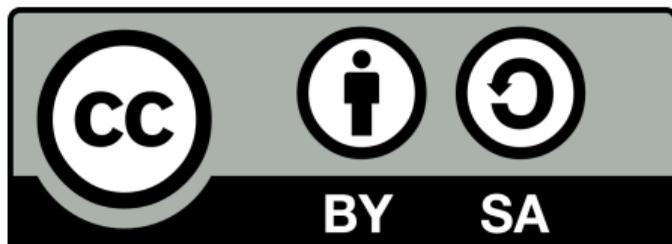
T. Lattimore and Cs. Szepesvári.
Bandit Algorithms.
Cambridge University Press, 2019



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



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



Creative Commons Attribution-ShareAlike (CC BY-SA 4.0)

- You are free to:
 - **Share:** copy and redistribute the material in any medium or format
 - **Adapt:** remix, transform, and build upon the material for any purpose, even commercially.
- Under the following terms:
 - **Attribution:** You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **ShareAlike:** If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
 - **No additional restrictions:** You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

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.