A BRIEF INTRODUCTION TO REINFORCEMENT LEARNING

Nicholas Kiefer

Outline

- From Supervised and Unsupervised to Reinforcement Learning
- Markov Decision Process
- Value-based Reinforcement Learning
- Other approaches to RL
- Application example
- Problems
- Summary

From SL and UL to RL: A Motivation

- Task 1: classify pictures into categories cats and dogs
  - Use SL and a big labeled dataset to train a NN
- Task 2: generate pictures of handwritten numbers
  - Use UL and an unlabeled dataset to train a NN
- Task 3: play Tic-Tac-Toe against an imperfect player
  - SL and UL are not very good for this!

→ Let's formalize the reinforcement learning problem

MARKOV DECISION PROCESS

Markov Decision Process

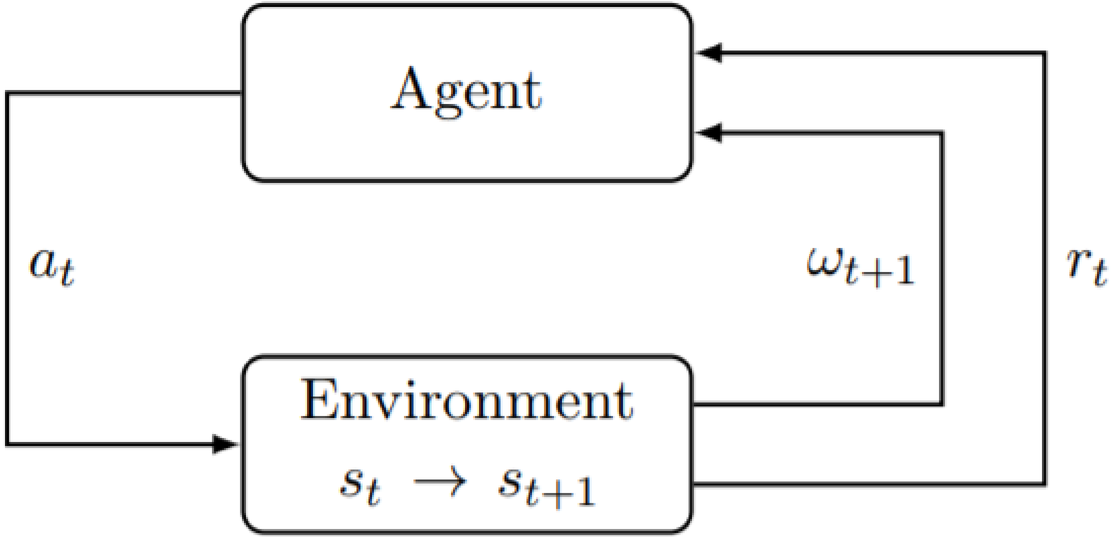


Figure 1: The agent-environment interaction [1]

Markov Decision Process

- 5-tuple of (S, A, T, R, γ) with
  - S the state space
  - A the action space
  - T the transition function
  - R the reward function
  - γ the discount factor

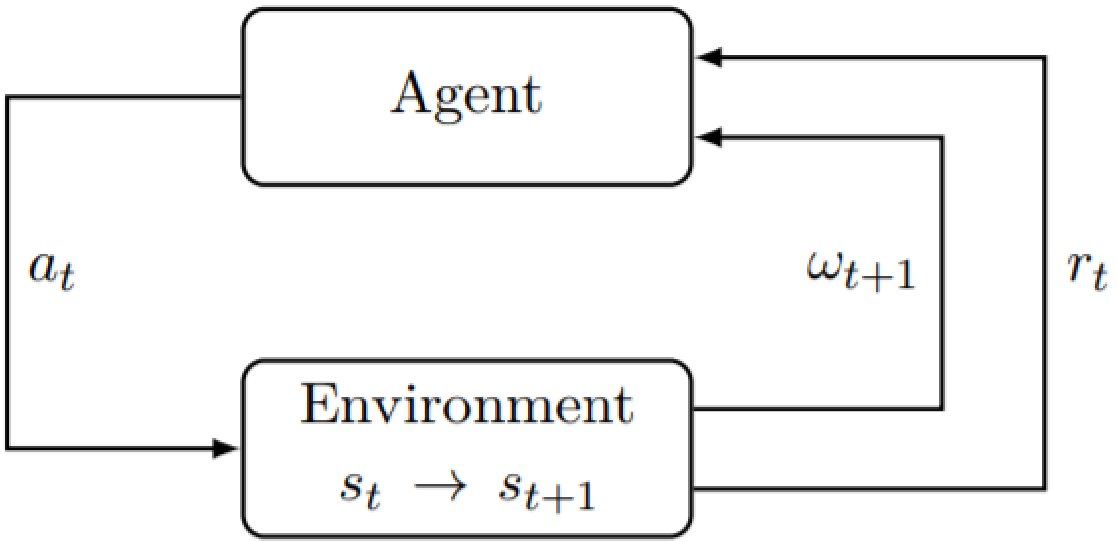


Figure 1: The agent-environment interaction [1]

→ Theoretical Framework of Reinforcement Learning

→ The agent wants to maximize his rewards

Reward, Policy and Value

- Reward function R maps from (state,action) to $\mathbb{R}$
  - Has upper bound: $|R(s, a\ )| \leq R_{max}$
  - Needs to be predefined
- The total discounted reward is $\mathcal{R} = \sum_{t=0}^{\infty} \gamma^t R_{t+1}$, $\gamma \in [0,1]$

Reward, **Policy** and Value

- Reward function *R* maps from (state,action) to $\mathbb{R}$
- Policy function $\pi(s, a) : \mathcal{S} \times \mathcal{A} \to [0,1]$
  - Gives transitional probability to choose action *a* in state *s*

Reward, Policy and **Value**

- Reward function *R* maps from (state, action) to $\mathbb{R}$

- Policy function $\pi(s,a) : \mathcal{S} \times \mathcal{A} \to [0,1]$

- The optimal value function maps to each state-action pair the largest expected reward achievable by any policy
- Value-based approach: Policy should be Ɛ-greedy
  - Any policy that is greedy w.r.t. the optimal values is optimal

Convergence Motivation

The optimal value function can be written as

$$V^*(s,a) = \max_a \mathbb{E}[r_{t+1} + \gamma V^*(s_{t+1},a) | s_t = s, a_t = a] \tag{1}$$

Starting out from a random value function V we can iteratively update the function following

$$V_{k+1}(s,a) = \mathbb{E}_\pi[r_{t+1} + \gamma V_k(s_{t+1},a) | s_t = s] \tag{2}$$

and get the optimal value function V^*.

Q-learning (Value-based RL)

Algorithm parameters: learning rate α, small $\epsilon > 0$
Initialize $V(s, a)$, for all $s \in S, a \in A$ arbitrarily except that $V(Goal, .) = 0$
Loop for each episode:
    Initialize s
    Loop for each step of episode:
        Choose a in s using ϵ-greedy policy
        Take action a, observe r, s'
        $V(s, a) \leftarrow (1 - \alpha)V(s, a) + \alpha[r + \gamma \max_a V(s', a)]$
        $s \leftarrow s'$
    Until s is goal

Figure 2: Pseudo-code for Q-learning [2,modified]

Example: Gridworld

- Episodic, undiscounted task, discrete state space
- Each step gives reward of -1 except if:
    - Agent moves into target state, $R = 0$
    - Agent falls off cliff, $R = -100$
- Values of state-action pairs should reflect shortest path

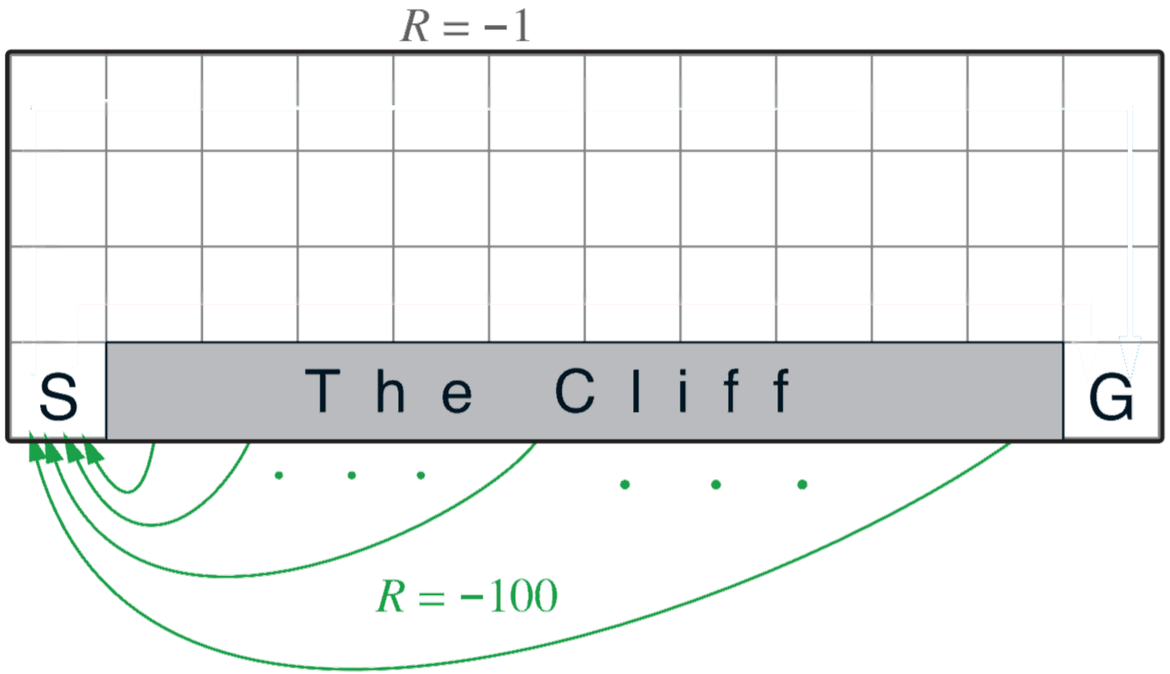


Figure 3: Gridworld with start (S), goal (G) and a cliff region [2]

Example: Gridworld

- Episodic, undiscounted task, discrete state space
- Each step gives reward of -1 except if:
  - Agent moves into target state, $R = 0$
  - Agent falls off cliff, $R = -100$
- Values of state-action pairs should reflect shortest path
- Use Ɛ-greedy policy

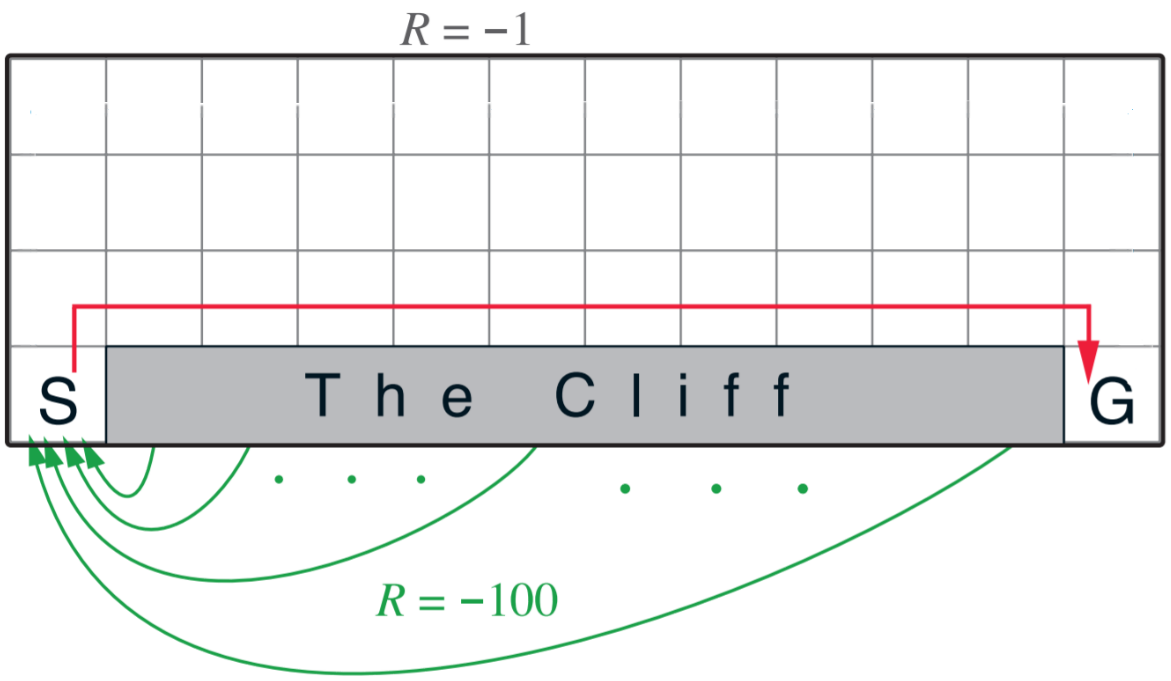


Figure 4: Gridworld with start (S), goal (G), a cliff region and optimal path [2]

Example: Gridworld

- Episodic, undiscounted task, discrete state space
- Each step gives reward of -1 except if:
  - Agent moves into target state, $R = 0$
  - Agent falls off cliff, $R = -100$
- Values of state-action pairs should reflect shortest path
- Use $\mathcal{E}$-greedy policy

```
    Up    Down   Left   Right
[[ 0.779  0.75  -0.389  0.914]
 [-0.103 -0.107 -1.005 -0.025]
 [ 0.492 -0.528  1.051  1.023]
 [-0.296  0.646  0.812  0.913]
 [ 0.992 -0.047 -0.242  0.177]
 [ 0.649  1.007 -0.409 -0.741]
 [-0.09   1.044  0.538 -0.134]
 [ 0.683 -0.935 -0.939  0.586]
 [ 1.075 -0.344 -1.067 -1.066]
 [-0.164  0.544  0.943  0.599]
 [-0.888 -0.097  0.932 -0.344]
 [-0.492 -0.933 -0.372  0.828]]
```

Figure 5: Initial Values along the way

```
[-12.     -12.829 -12.004 -63.781]
[-11.086 -12.071 -11.895 -11.   ]
[-10.001 -77.813 -11.022 -10.   ]
[ -9.664 -64.253 -10.479  -9.   ]
[ -8.938 -77.785  -8.589  -8.   ]
[ -7.666 -39.636  -7.411  -7.   ]
[ -7.165 -63.306  -6.431  -6.   ]
[ -6.111 -39.577  -6.447  -5.   ]
[ -5.566 -64.305  -4.429  -4.   ]
[ -3.057 -86.316  -3.926  -3.   ]
[ -3.621 -63.78   -2.632  -2.   ]
[ -2.152 -78.166  -2.26   -1.   ]
[-1.426 -0.    -1.195 -1.103]
```

Figure 6: Final Values along the way

Example: Gridworld

- Optimal path is quickly found (<1s)
- Ɛ -greedy strategy introduces random deterioration of cumulative reward

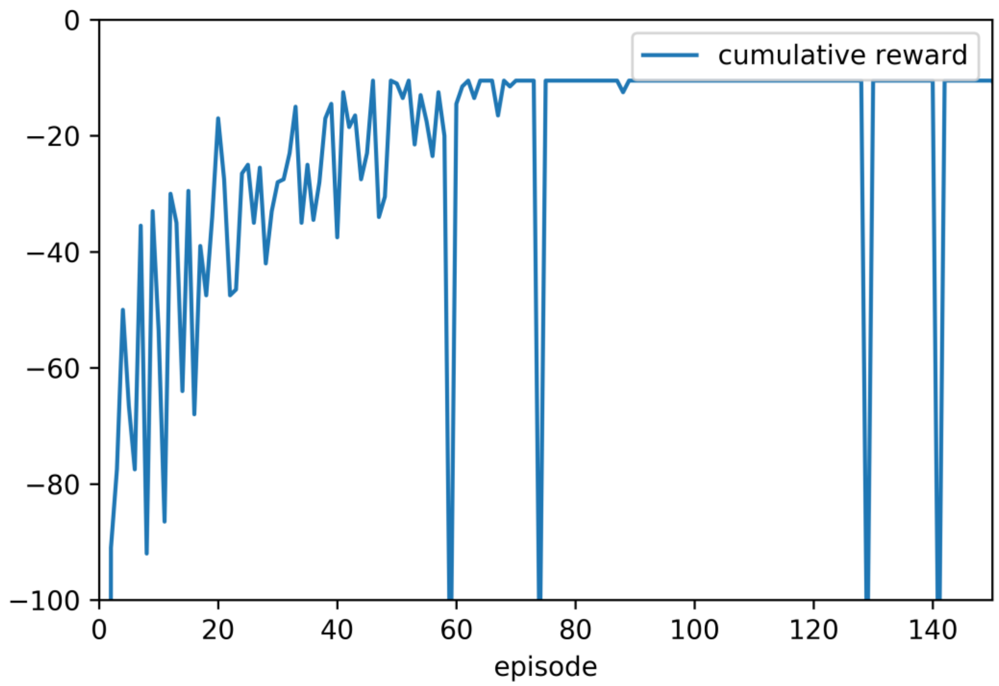


Figure 7: Cumulative reward during training

APPROACHES TO RL

Approaches – Function Approximation

- Problems are high-dimensional but have low complexity
- Every continuous function can be approximated to an arbitrary degree with NNs [3]
  - http://neuralnetworksanddeeplearning.com/chap4.html

→ So why not use NNs to approximate Value and Policy Function?

- Approximation needs to be well-suited for the task for convergence to an optimal solution

Approaches

- **Value-based**
  - State-action value pairs are build up through experience
  - Simplest method is lookup table
- **Policy-based**
  - No value function required
  - continuous action spaces are accessible

→ Methods are combined in Actor-Critic approach

Value- and Policy-based methods are **Model-free**

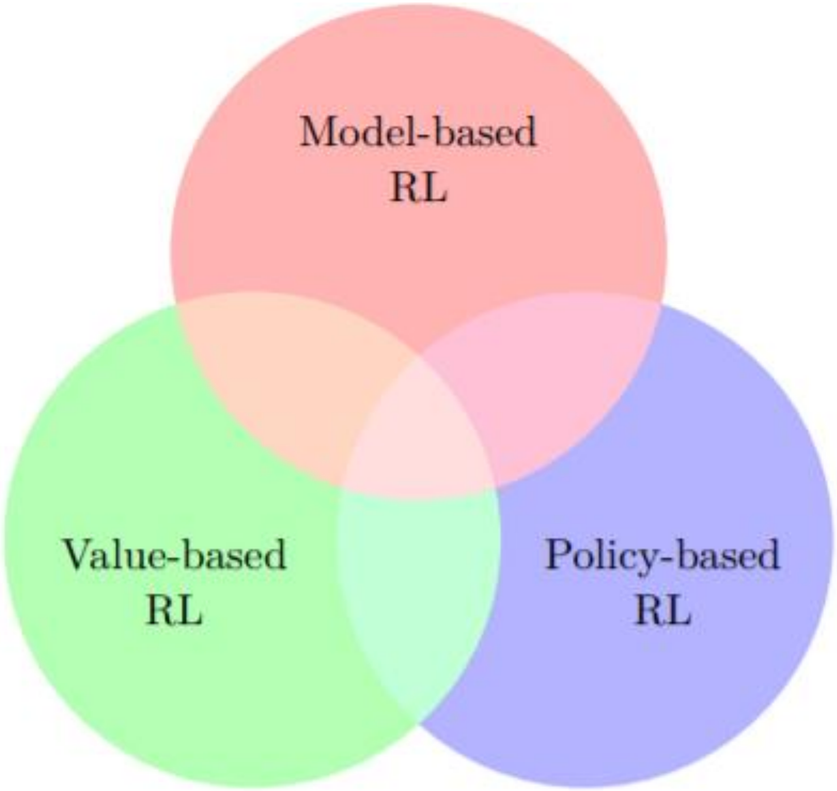


Figure 8: Venn diagram of different methods in RL [1]

Approaches

- **Model-based**
  - Model of environment is given
  - Actions can be planned beforehand on model
  - Most common method are lookahead searches
  - Good judgement of trajectories
  - Stopping criteria are hard to define

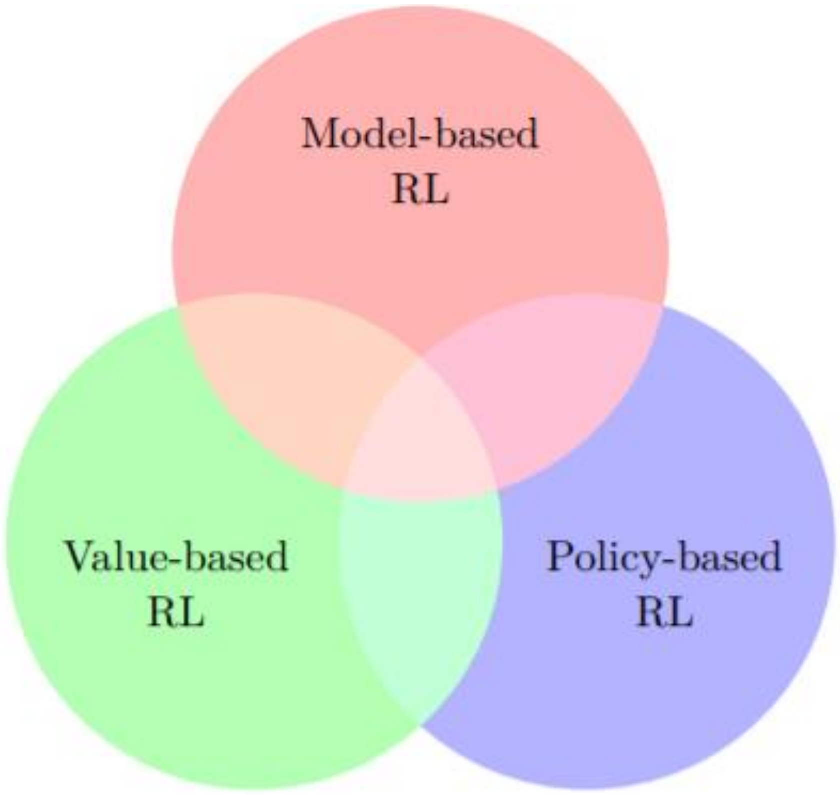


Figure 8: Venn diagram of different methods in RL [1]

The Game Of Go

Rules

- 2 Players place stones on a grid taking turns
- Stones with no freedom are dead or belong to a group

Goal

- Surround biggest territory possible

- Possible sequences $\sim 10^{800}$

→ Curse of Dimensionality

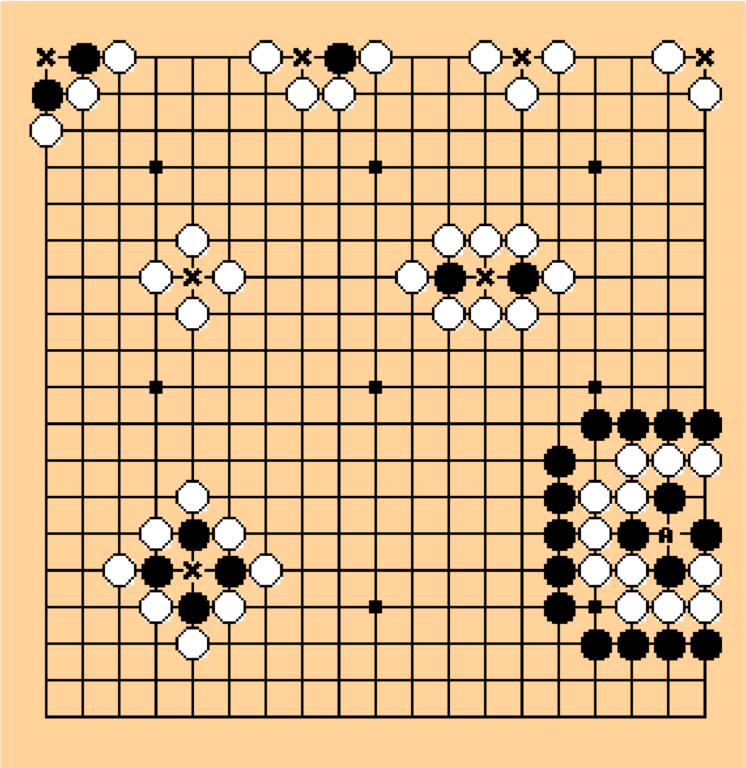


Figure 9: situations on a Go board [4]

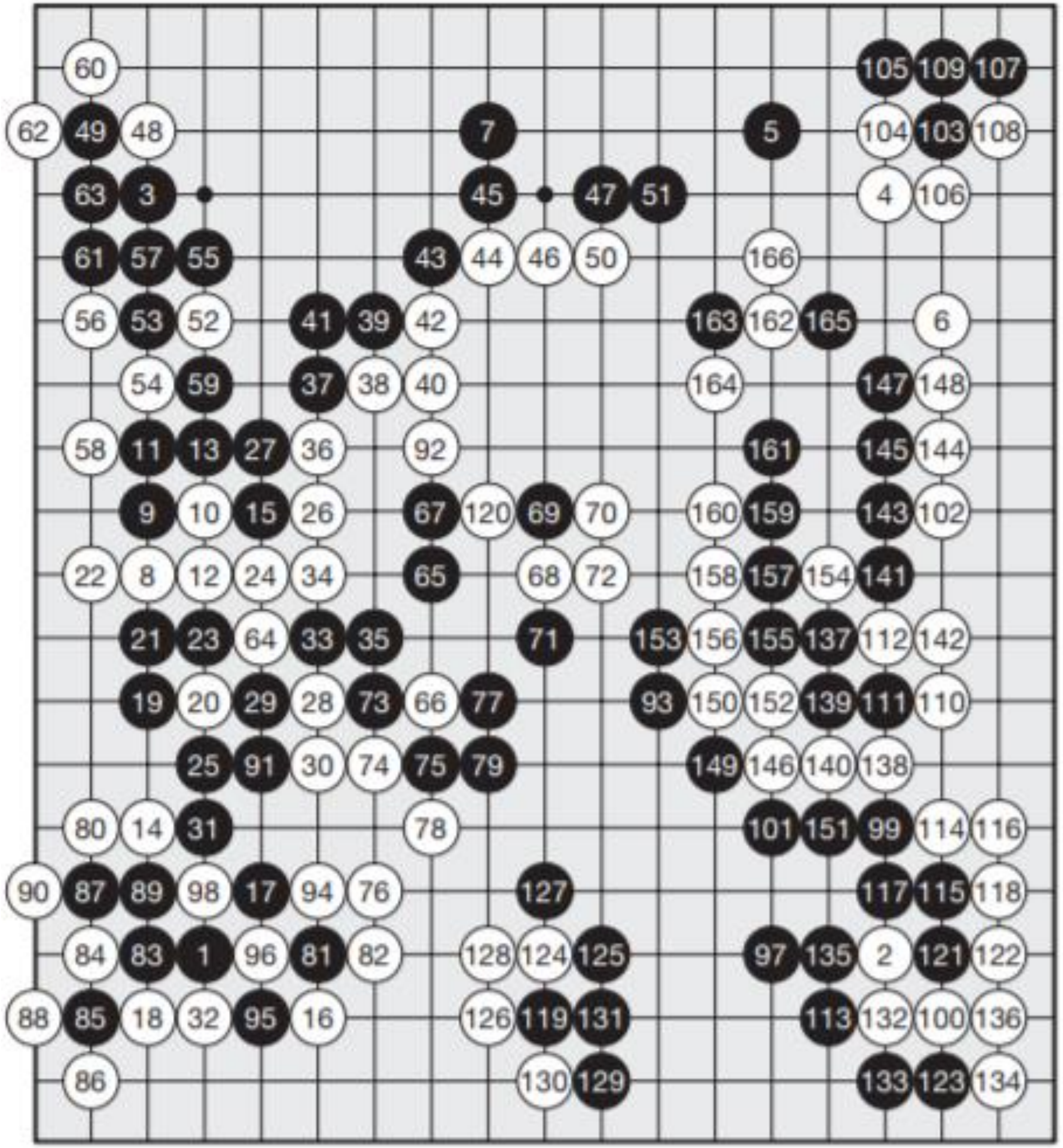


Figure 10: AlphaGo (white) vs. Fan Hui (AlphaGo won) [5]

Example: AlphaGo [5]

SL policy function → **RL policy function** → **RL value function** → **MCTS for best move**

SL policy function
- Based on 30 million played positions
- SGD
- 13 convolutional layers
- Input: 19x19x48

RL policy function
- Policy gradient RL
- Play games with previous iteration of network
- Input: 19x19x48

RL value function
- Value NN with similar architecture as policy NN
- Outputs single prediction
- Trained on 30 million generated positions

MCTS for best move
- Lookahead search
- **Reward is zero until terminal state**

PROBLEMS IN RL

Problems

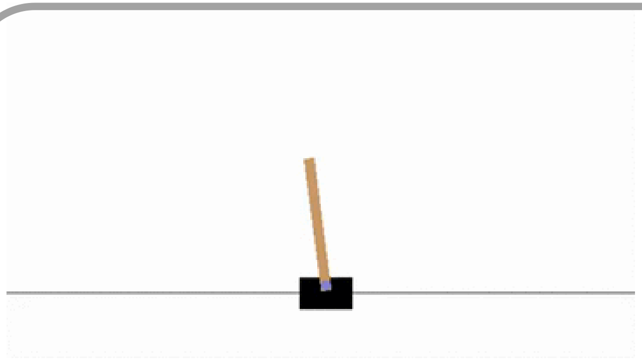
- Sparse rewards
  - Many steps in a very particular order are necessary to receive one reward.
  - a solution are self-defined or intermediate rewards
- Sampling efficiency
  - slow learning compared to humans
  - Lots of data is needed to train an agent
    - MNIST dataset: 70,000 pictures
    - AlphaGo first stage: 30 million positions

Problems

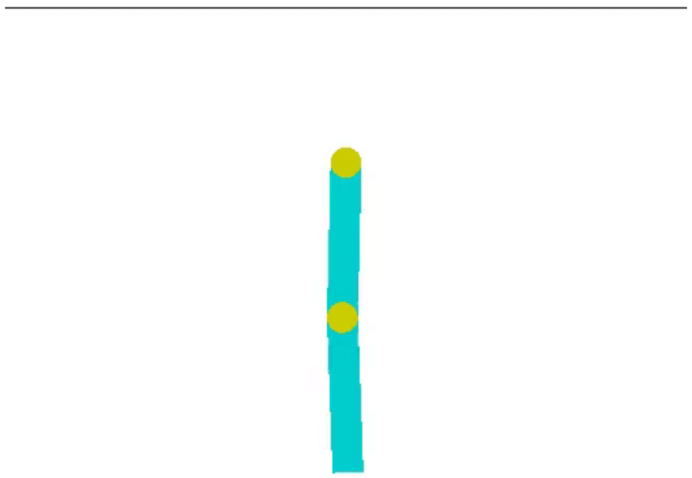
- Exploitation vs. Exploration
  - exploration is gathering more observation/knowledge about the environment
  - exploitation is maximizing the reward given the current knowledge
  - Neither can be pursued exclusively
  - Easiest approach is ϵ-greedy policy
- Credit assignment problem
  - intrinsic in policy gradient methods
  - what exact action justifies the reward?

Problems

- Benchmarking
  - some standard problems [2]:



Video 1: Cartpole demonstration [6]



Video 2: Acrobot demonstration [7]

  - Games are a preferred choice
  - In general it is hard to quantize the quality of an algorithm

Summary

- Keywords: Reward, Value, Policy, Function approximation, Q-learning, Model-based
- Not explained: Overfitting, Reward shaping, Auxiliary tasks, Imitation learning, Bellman equations, Integration of model-based and model-free methods, Double Q-learning, Double DQN, Inverse reinforcement learning, zero-shot learning

→ **RL is a big field!**

→ Recommend literature: (Sutton, Barto), (François-Lavet et. al), (Csaba Szepesvari)

References

Thank you for your attention!

- [1] An Introduction to Deep Reinforcement Learning
  https://arxiv.org/pdf/1811.12560.pdf
- [2] Sutton, Barto: Reinforcement Learning: An Introduction
  http://incompleteideas.net/book/bookdraft2017nov5.pdf
- [3] http://neuralnetworksanddeeplearning.com/chap4.html
- [4] Go board: https://www.cs.cmu.edu/~wjh/go/rules/Japanese.2.gif
- [5] Mastering the game of Go with deep neural networks and tree search
  https://www.nature.com/articles/nature16961.pdf
- [6] Cartpole video: https://www.youtube.com/watch?v=46wjA6dqxOM
- [7] openai: https://gym.openai.com/envs/#classic_control
- [8] Algorithms for Reinforcement Learning
  https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf