DeepMind

Introduction to OpenSpiel

To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

DeepMind

Introduction to OpenSpiel

Edward Lockhart

Joint work with Marc Lanctot, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, Jonah Ryan-Davis, and several external contributors!

Many, many great collaborators!

DeepMind

1 Introduction

To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

What We'll Cover

1. Introduction & Motivation
2. API Overview
3. Counterfactual Regret Minimization
4. Reinforcement Learning

To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

Intro to OpenSpiel (Released Aug '19)

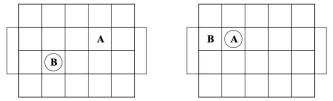
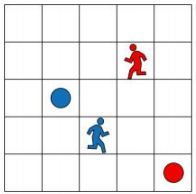
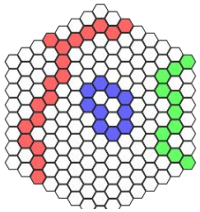
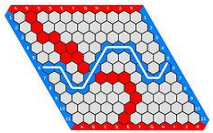


Figure 2: An initial board (left) and a situation requiring a probabilistic choice for A (right).

- Open source framework for research on RL, search, and planning in games
- Main impl in C++ and Python. Also:
  - Go
  - Julia (contributed post-release)
- > 25 games
- > 10 algorithms







To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

OpenSpiel

Supports:

- n-player games
- Zero-sum, coop, general-sum
- Perfect / imperfect info
- Simultaneous-move games

To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

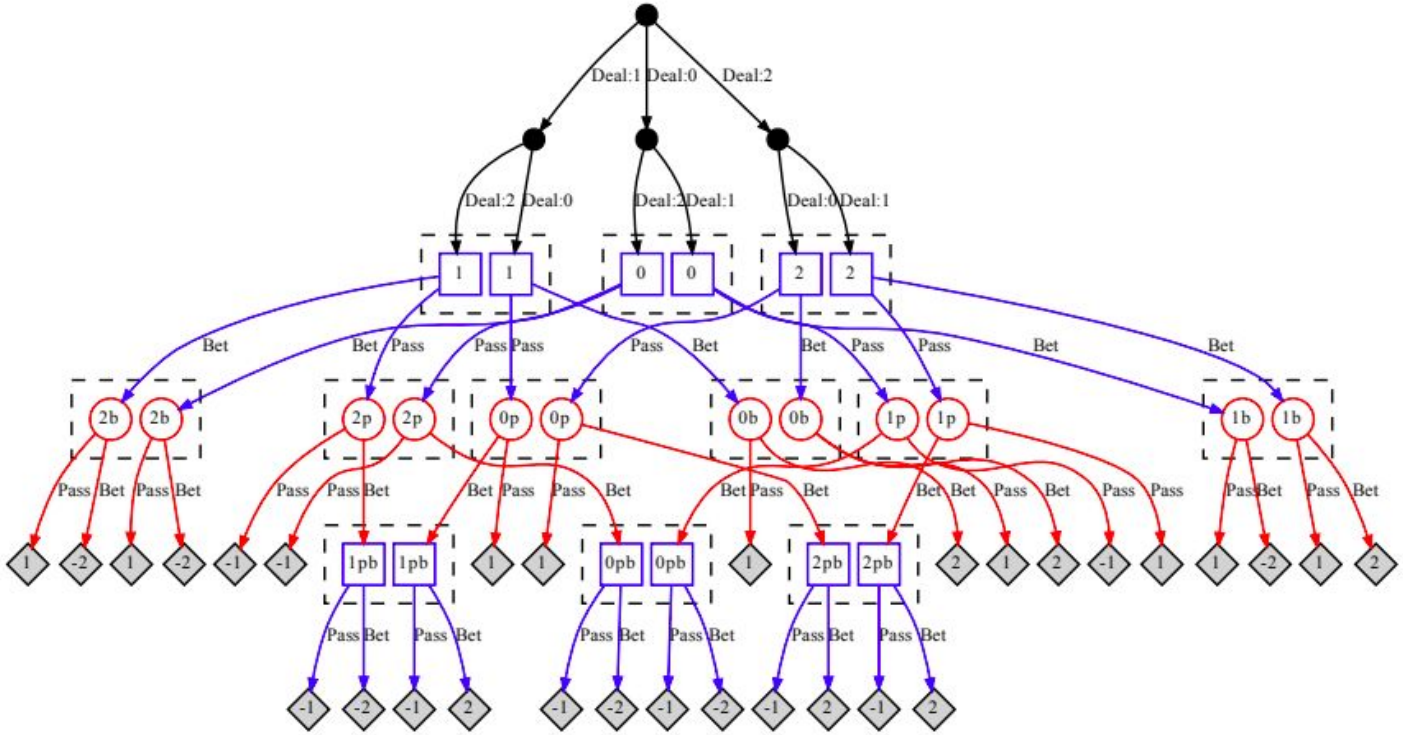
Tour of OpenSpiel

github.com/deepmind/open_spiel/

- Contributors
- Games
- Algorithms

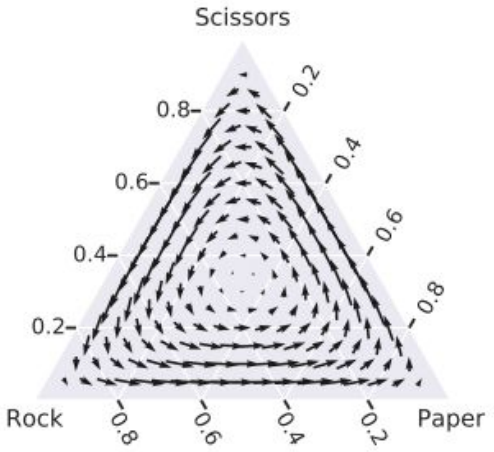
To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

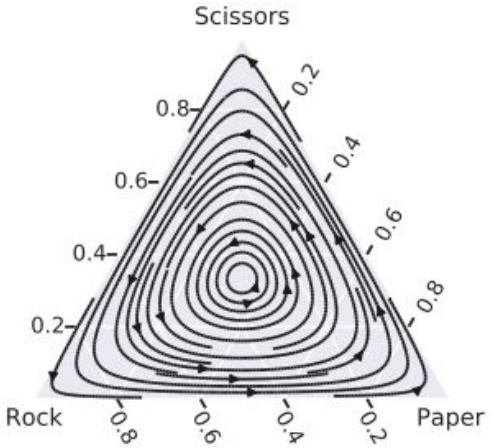
OpenSpiel: Example Viz (Kuhn Poker)

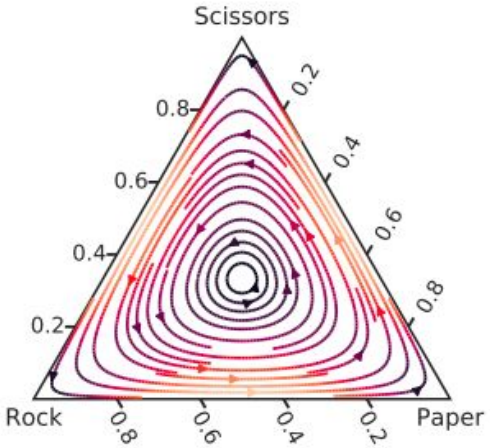


To work hands-on, open the [colab notebook] and run the first three cells now (it takes ~10 mins)

OpenSpiel: Example Viz (Replicator dynamics)

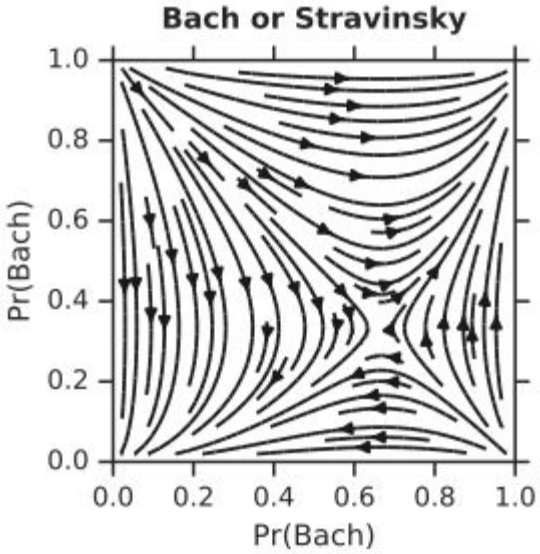


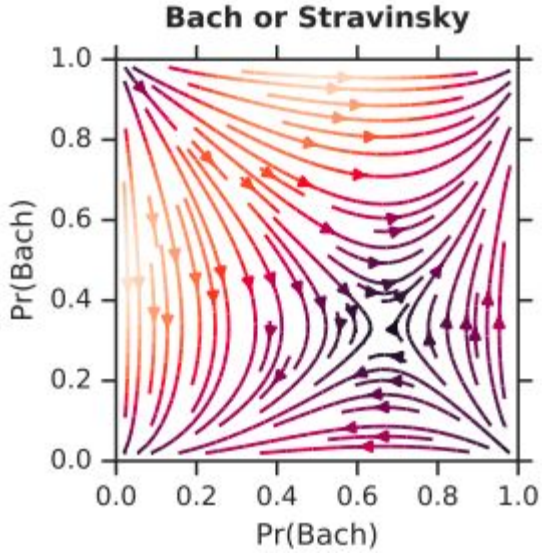




To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

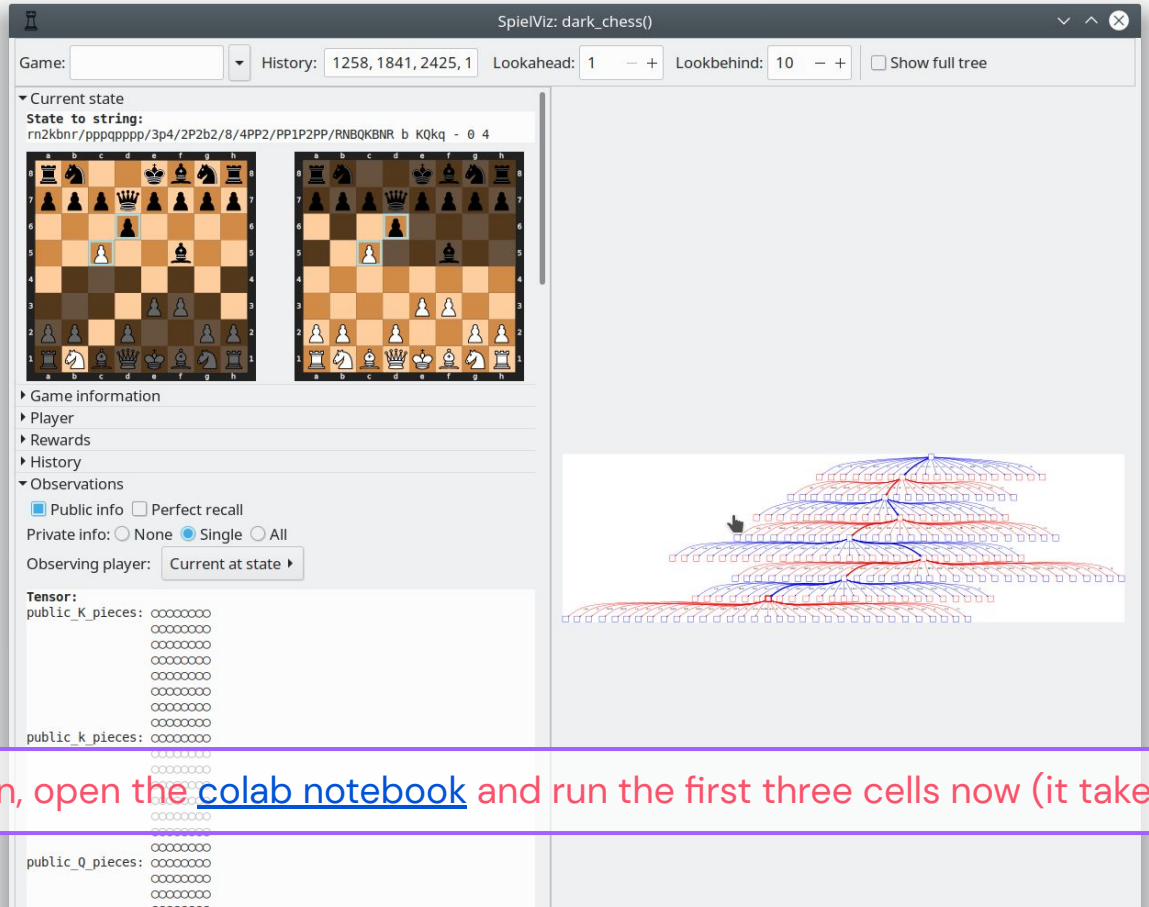
OpenSpiel: Example Viz (Replicator dynamics)





To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

OpenSpiel: Example Viz (Dark Chess)



To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

Motivation: Why another games / RL library?

1. Promote work on **general** multiagent RL
   a. “Atari Learning Environment” of multiagent/games
   b. General game-**learning**
2. **Games** have specific requirements and use cases:
   a. Illegal moves, turn-based, etc.
3. Connecting research communities!
4. Open code, metrics, communication, progress
5. Reproducibility in research

To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

DeepMind

2 API Overview

To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

OpenSpiel: Design & Code

Design Philosophy

1. **Keep it simple.**
2. **Keep it light.**

Main structure:

- C++ core + Python API
- Julia API
- Go API
- Games in C++ (or Python)
- Algs in C++ and Python
- Many examples / colab

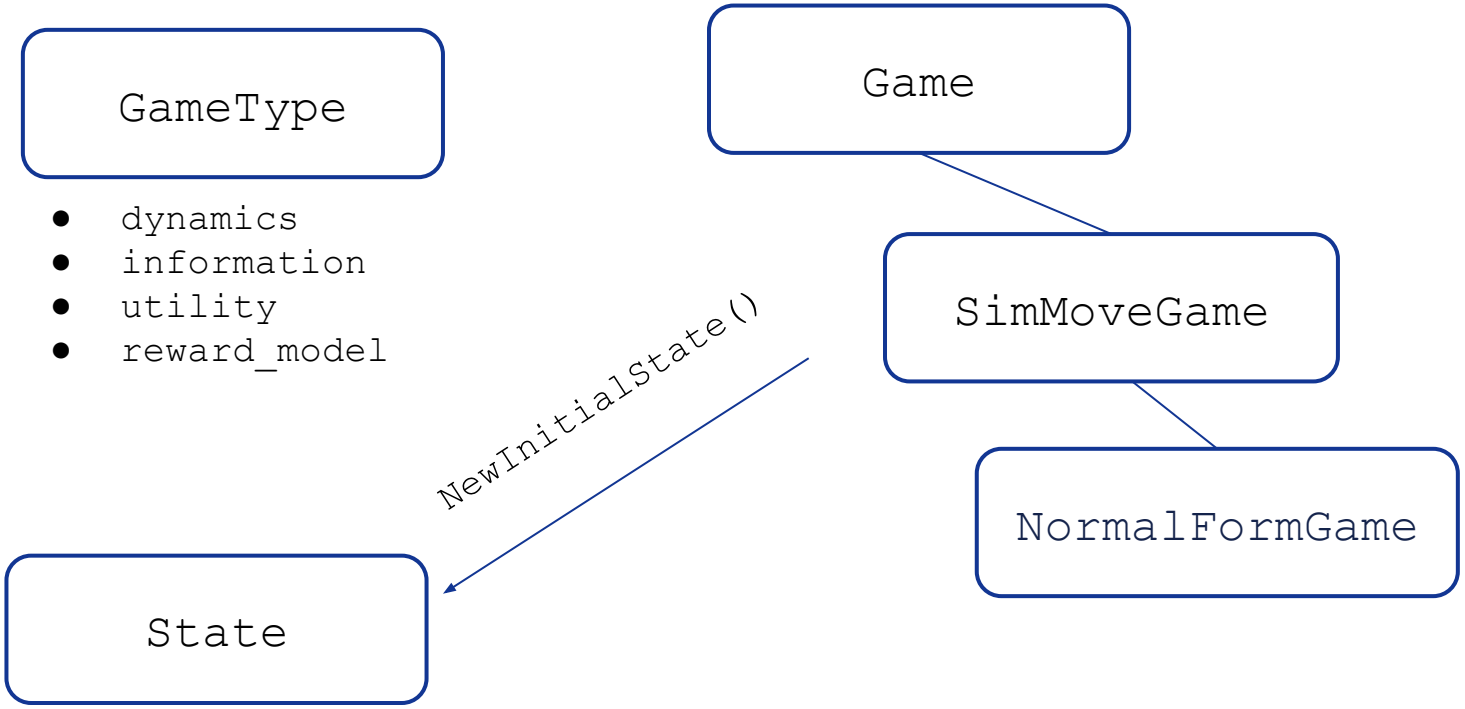
Example

```python
import random
import pyspiel
import numpy as np

game = pyspiel.load_game("kuhn_poker")
state = game.new_initial_state()
while not state.is_terminal():
  legal_actions = state.legal_actions()
  if state.is_chance_node():
    # Sample a chance event outcome.
    outcomes_with_probs = state.chance_outcomes()
    action_list, prob_list = zip(*outcomes_with_probs)
    action = np.random.choice(action_list, p=prob_list)
    state.apply_action(action)
  else:
    # The algorithm can pick an action based on an observation (fully observable
    # games) or an information state (information available for that player)
    # We arbitrarily select the first available action as an example.
    action = legal_actions[0]
    state.apply_action(action)
```

To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

Object-Oriented API



To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

OpenSpiel Games

Playthroughs are in open_spiel/integration_tests/playthroughs

Example games:

- Two-player zero-sum perfect-information: tic_tac_toe
- Simultaneous move game: matrix_rps
- Chance & multi-player: pig_4p
- Chance & imperfect-information: kuhn_poker_2p

To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

DeepMind

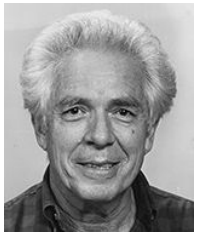
3 Kuhn Poker and CFR

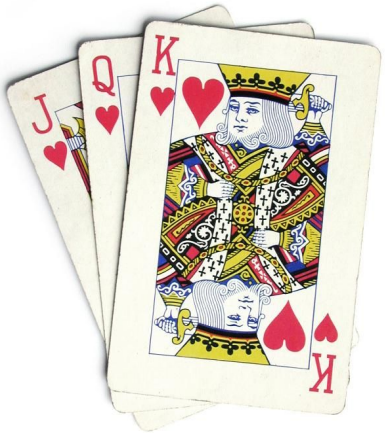
To work hands-on, open the colab notebook and run the first three cells now (it takes ~10 mins)

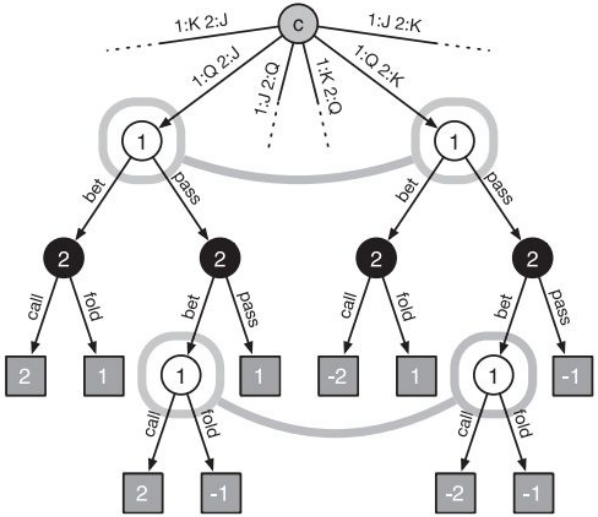
Partially Observable Zero-Sum Games

Kuhn (simplified) poker

- Players start w/ 2 chips
- Each: ante 1 chip
- 3-card deck
- 2 actions: pass, bet
- Reward: money diff







Terminology

- An **information state,** s, corresponds to a *sequence of observations*
  - with respect to the player to play at s

Ante: 1 chip per player, 

, P1 bets (raise)

Terminology

- An **information state,** s, corresponds to a *sequence of observations*
  - with respect to the player to play at s

private observation

Ante: 1 chip per player,  , P1 bets (raise)

Terminology

- An **information state,** s, corresponds to a *sequence of observations*
  - with respect to the player to play at s

private observation

Ante: 1 chip per player,  , P1 bets (raise)

Environment is in one of many **world states** $h \in s$

Terminology

- An **information state,** s, corresponds to a *sequence of observations*
  - with respect to the player to play at s

private observation

Ante: 1 chip per player,  , P1 bets (raise)

Environment is in one of many **world states** $h \in s$

full **history** of actions (including nature's!!)

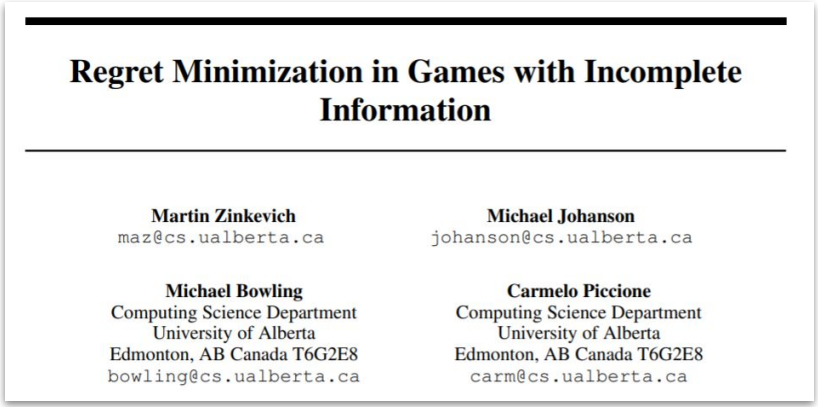
Kuhn Poker in Open Spiel

Have a look at the playthrough: kuhn_poker_2p

CFR Algorithm

1. Initialize the current policy to uniform random
2. Repeatedly
    a. Compute the **counterfactual** regret for each action at each decision point when both players follow the current policy
    b. Update the current policy at each decision point to be proportional to the total counterfactual regret so far (if positive)

The policy to evaluate is the average policy used across all iterations of the loop.



Regret Minimization in Games with Incomplete Information

Martin Zinkevich
maz@cs.ualberta.ca

Michael Johanson
johanson@cs.ualberta.ca

Michael Bowling
Computing Science Department
University of Alberta
Edmonton, AB Canada T6G2E8
bowling@cs.ualberta.ca

Carmelo Piccione
Computing Science Department
University of Alberta
Edmonton, AB Canada T6G2E8
carm@cs.ualberta.ca

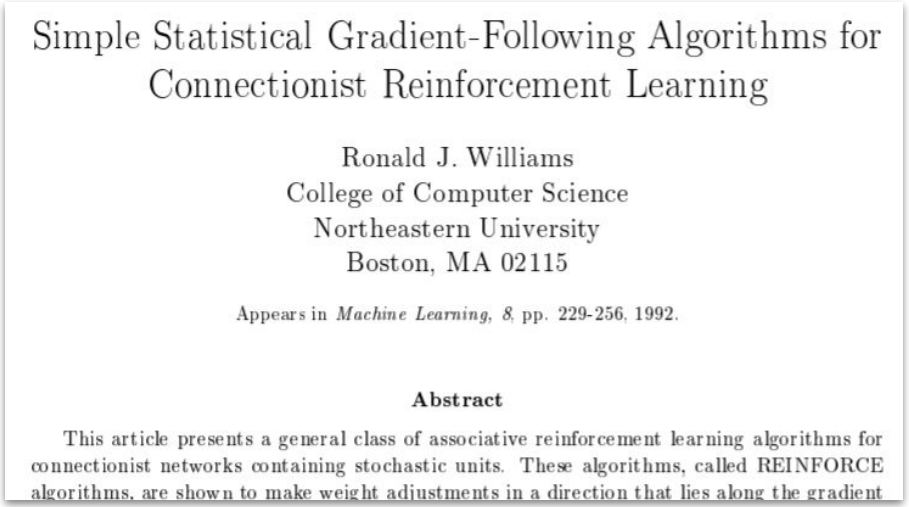
DeepMind

4 REINFORCE

REINFORCE Algorithm

1. Initialize the current policy to uniform random
2. Repeatedly
    a. Play a full episode of the game
    b. Update the policy in the direction of the reward received for the actions chosen

This directly optimizes the policy.



Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning

Ronald J. Williams
College of Computer Science
Northeastern University
Boston, MA 02115

Appears in *Machine Learning*, *8*, pp. 229-256, 1992.

Abstract

This article presents a general class of associative reinforcement learning algorithms for connectionist networks containing stochastic units. These algorithms, called REINFORCE algorithms, are shown to make weight adjustments in a direction that lies along the gradient

DeepMind

5 Resources

Best File References

First example and API references: (**bold** = playable via console)

- `examples/example.cc`
- `python/examples/example.py`
- **`python/examples/poker_fcpa_example.py`**
- `python/examples/matrix_game_example.py`
- `python/egt/dynamics_test.py`
- **`python/examples/mcts.py`**
- `python/examples/kuhn_policy_gradient.py`
- `python/examples/tic_tac_toe_qlearner.py`
- `python/examples/independent_tabular_qlearning.py`

Thank You!

OpenSpiel: A Framework for Reinforcement Learning in Games

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, Jonah Ryan-Davis

(Submitted on 26 Aug 2019 (v1), last revised 31 Dec 2019 (this version, v5))

OpenSpiel is a collection of environments and algorithms for research in general reinforcement learning and search/planning in games. OpenSpiel supports n-player (single- and multi- agent) zero-sum, cooperative and general-sum, one-shot and sequential, strictly turn-taking and simultaneous-move, perfect and imperfect information games, as well as traditional multiagent environments such as (partially- and fully- observable) grid worlds and social dilemmas. OpenSpiel also includes tools to analyze learning dynamics and other common evaluation metrics. This document serves both as an overview of the code base and an introduction to the terminology, core concepts, and algorithms across the fields of reinforcement learning, computational game theory, and search.

- Paper: https://arxiv.org/abs/1908.09453
- Github: github.com/deepmind/open_spiel/

DeepMind

The end and thank you

Edward Lockhart
locked@google.com

March 2021