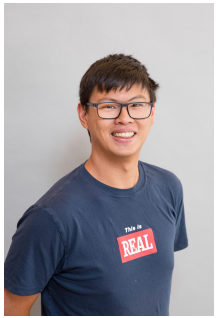
Reinforcement Learning for Games

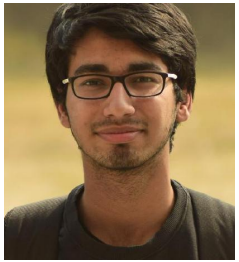
Timothy Lillicrap & Blake Richards

Who made this possible?



Raymond Chua
McGill University and Mila

Mandana Samiei
McGill University and Mila



Kushaan Gupta
University of Lethbridge

The tutorial is based on the AlphaZero/MCTS repo created by Surag Nair and colleagues.

Outline

1. Conceptual & concrete goals.
2. A bit of history.
3. What is RL? What is planning?
4. From AlphaGo to AlphaZero
5. Background for tutorials
6. Tutorial #1-8
7. Bonus Tutorials #1-2
8. What can't our algorithms do?
9. Planning with a learned model

Conceptual Goals

1. Understand how to use reinforcement learning and planning to solve 2-player zero-sum games.

2. Understand how to train policy and value functions for games.

3. Understand how value and policy functions can be used to play games and plan.

Along the way, build appreciation for how these machine learning and game theory concepts map onto ideas in neuroscience.

Concrete Goals

1. Create an Othello player that uses planning, in combination with value and policy networks, to play games.

2. Show that when pitted head to head, planning offers an advantage over players that does not plan.

A bit of history

In the 1990s researchers created chess playing AI that took games from the best chess players in the world.

Deep Blue made use of custom hardware to run the alpha-beta planning algorithm and famously bested Garry Kasparov, the world champion at the time.

However, Deep Blue could not easily be generalized to solve other complex board games.

It took almost 20 years more before AI was able to master the game of Go. Why?

Limitations of Deep Blue

Deep Blue relied on two key features of chess:

1. Effective board position value judgements formalized by expert human players.
2. An action space that was small enough that planning algorithms could be exhaustive enough.

These features limit the applicability of Deep Blue's algorithms.
For example, the board game Go has a *much* larger action space, and the value of board positions are less easily made explicit by experts.

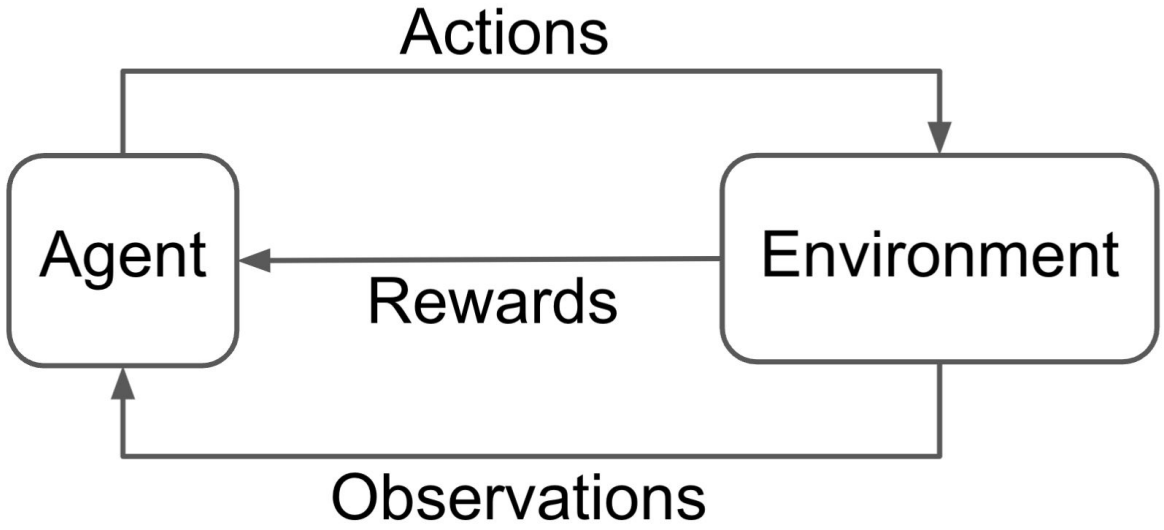
Combining planning with learning

In 2016 AlphaGo overcame both of these problems to best a top player, Lee Sedol, in a full game of Go.

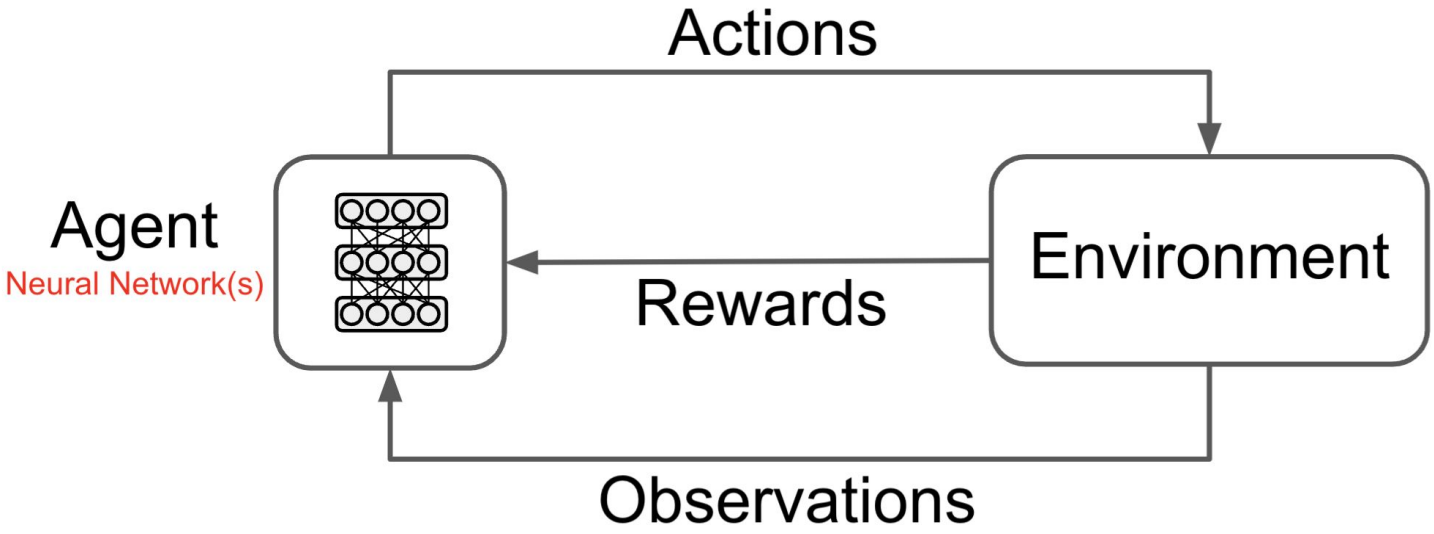
To do so, it combined a planning algorithm with reinforcement learning in order to overcome the two limitations of Deep Blue:

1. Instead of asking experts, AlphaGo learned how to value board positions by looking at games and their outcomes.
2. AlphaGo learned a "policy prior" by predicting expert moves, allowing it to massively shrink the necessary search space for planning.

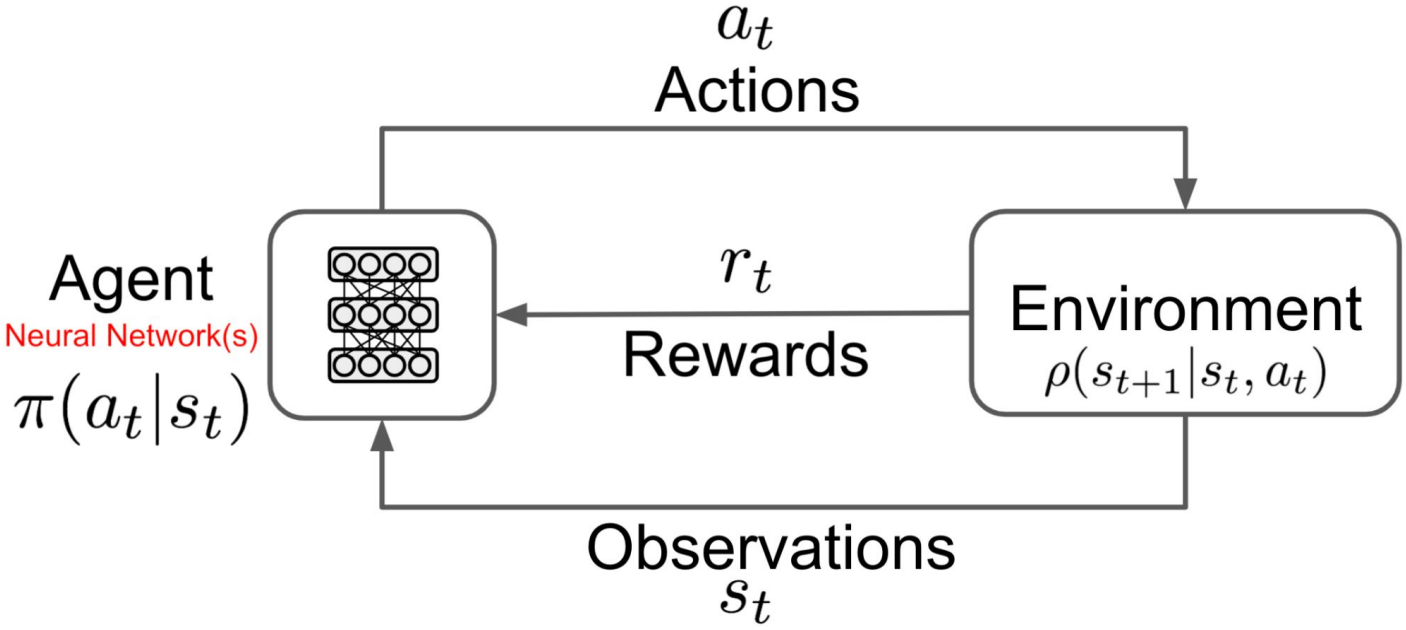
What is reinforcement learning?



What is deep reinforcement learning?



Formalizing the agent-environment loop



A single trial

$$
\begin{array}{ccccc}
r_0, & r_1, & r_2, & \ldots, & r_T \\
a_0, & a_1, & a_2, & \ldots, & a_T \\
\pi(a_0|s_0), & \pi(a_1|s_1), & \pi(a_2|s_2), & \ldots, & \pi(a_T|s_T) \\
s_0, & s_1, & s_2, & \ldots, & s_T \\
 & \rho(s_1|s_0,a_0), & \rho(s_2|s_1,a_1), & \ldots, & \rho(s_T|s_{T-1},a_{T-1})
\end{array}
$$

Time ⟶

Probability of trajectory τ

$$p_\theta(\tau) = \rho(s_0) \prod_{t=0}^{T} \rho(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$

Measuring outcomes

Return for a single trial:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$$

Objective function:

$$J(\theta) = \int_{\mathbb{T}} p_\theta(\tau) R(\tau) d\tau$$

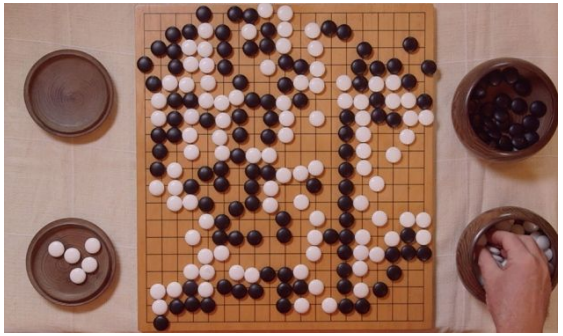
What is planning?

In machine learning, planning is the process of predicting some aspect of the future by thinking ahead.

Planning can take myriad forms, but perhaps the clearest examples involve using a single-step model of the environment to simulate actions and state changes in "imagination".

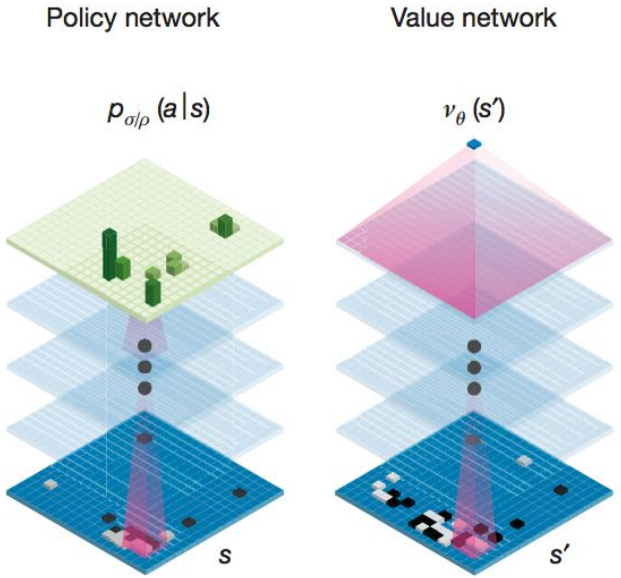
$$\rho\left(s_{t+1} \middle| s_t, a_t\right)$$

Combining Deep Networks with Planning



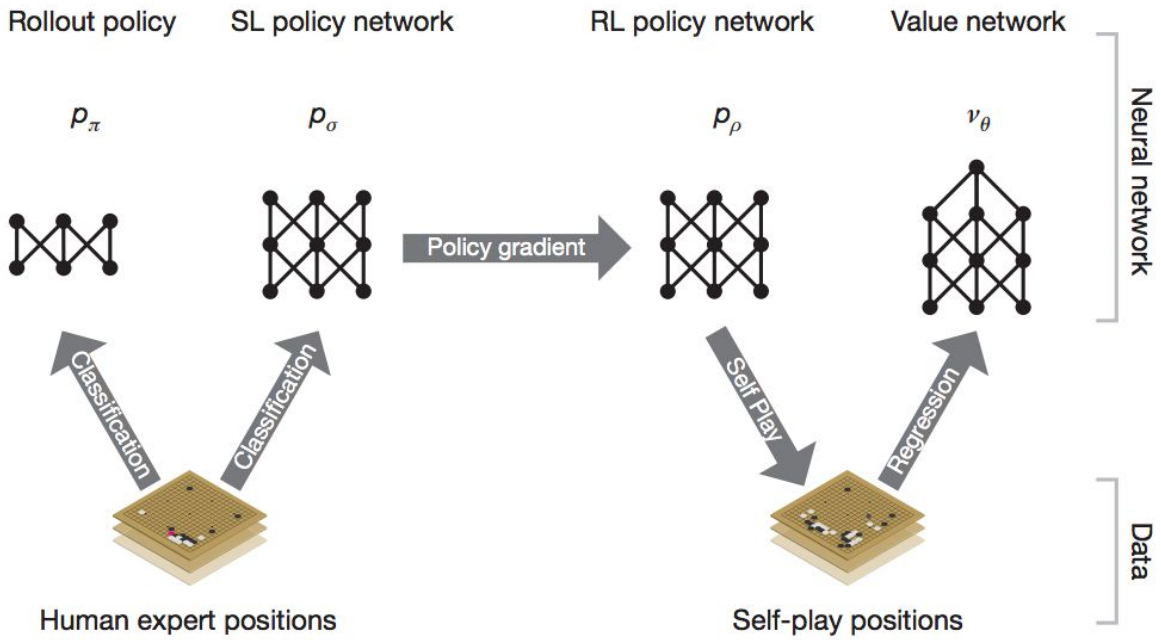
$$\rho(s_{t+1}|s_t, a_t)$$

Use environment model to plan!



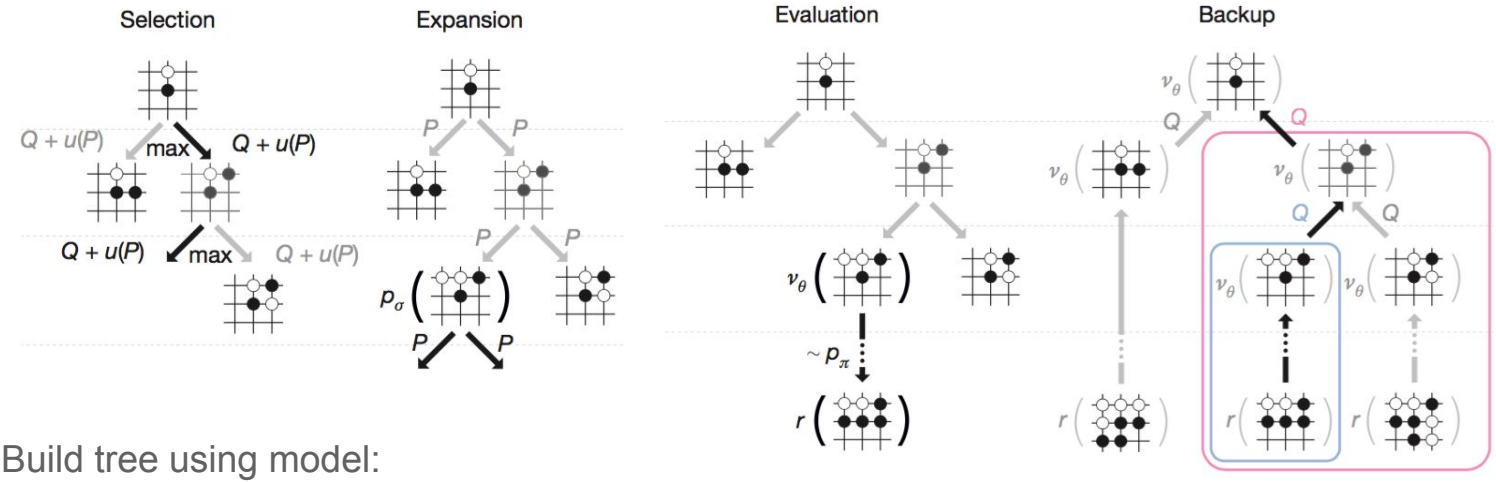
Silver, Huang et al., *Nature*, 2016

Training Policy and Value Networks



Silver, Huang et al., *Nature*, 2016

Plan with an environment model & MCTS



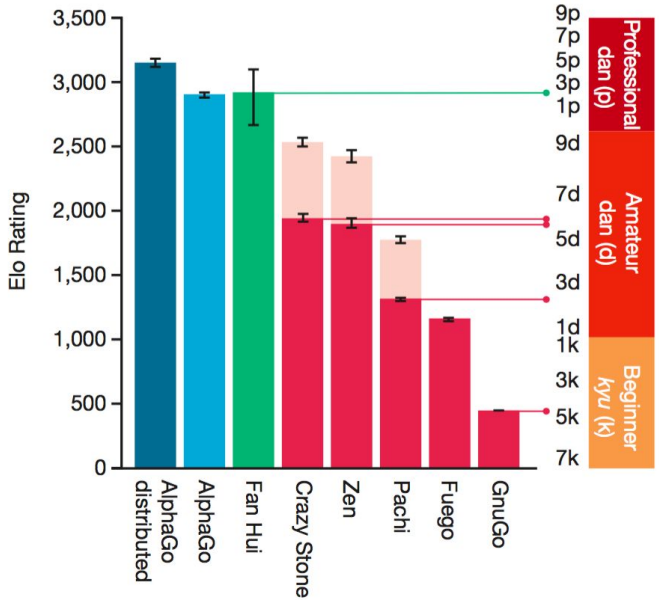
MCTS
=
Monte Carlo
Tree Search

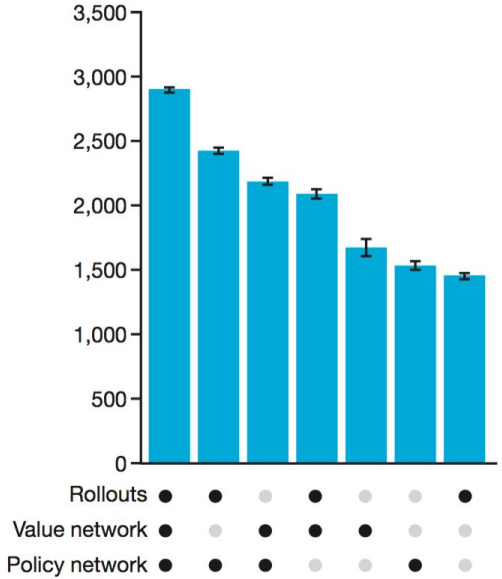
Build tree using model:

$$\rho(s_{t+1}|s_t, a_t)$$

Silver, Huang et al., *Nature*, 2016

Training Policy and Value Networks





Elo Rating Definition

Silver, Huang et al., *Nature*, 2016

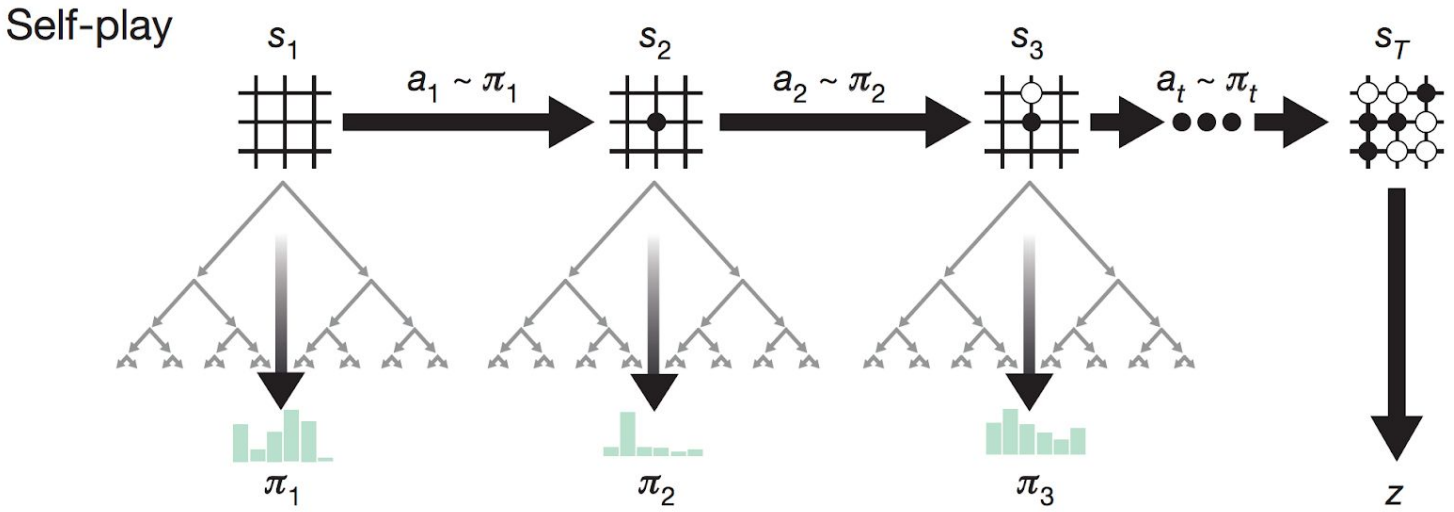
From AlphaGo to AlphaZero

The AlphaGo algorithm quickly evolved from an algorithm that used engineers in the design and training loop, to an algorithm that learned to play Go automatically from self-play. This algorithm was called AlphaGoZero.

Then, since AlphaGo was based on learning rather than expert opinions, it was possible to quickly generalize further.

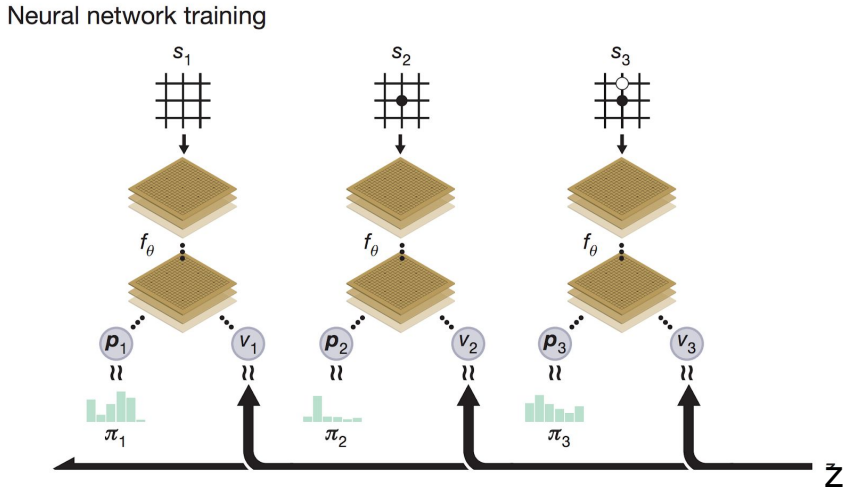
AlphaZero was able to learn Go, chess, and shogi from scratch with no human knowledge about the game except for the rules of the game.

Learning from self-play



Silver, Schrittwieser, Simonyan, et al. *Nature*, 2017

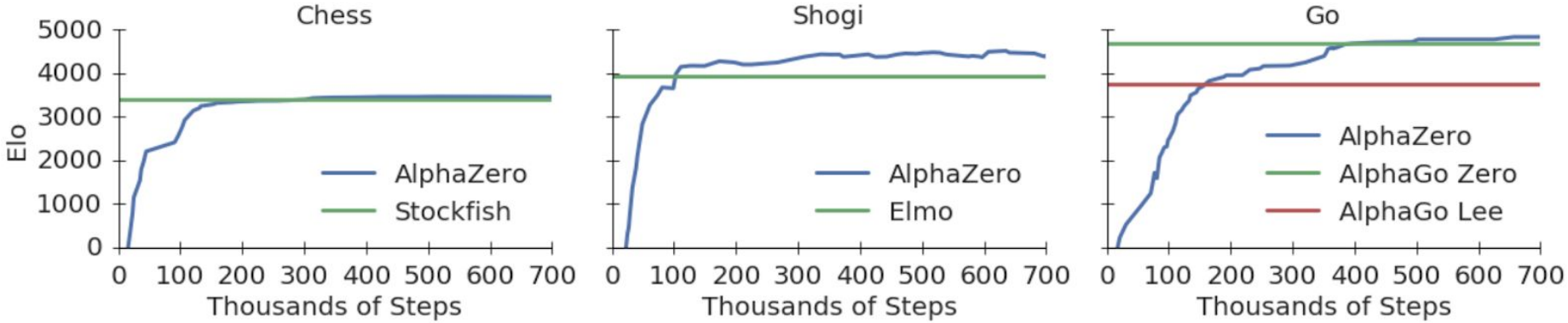
Learning from self-play



$$(\boldsymbol{p}, v) = f_\theta(s) \text{ and } l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2$$

Silver, Schrittwieser, Simonyan, et al. *Nature*, 2017

Learning from self-play: chess, Go, shogi



Silver, Hubert, Schrittwieser, et al. *Nature*, 2017

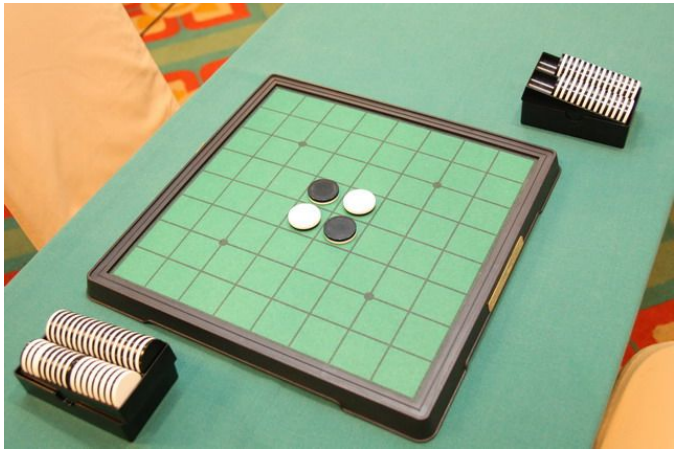
Background for tutorials: Othello/Reversi

We're going to build an Othello player. Othello is:

1. Simpler than Go or Chess.

2. Still a challenging and interesting game with a huge number of possible states.

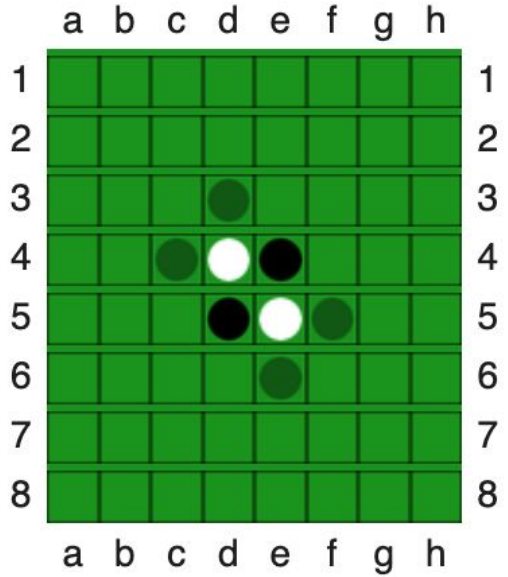
3. Can be played on smaller (6x6) board configuration for speed of simulations and learning.



The rules of Othello/Reversi

1. Each turn a player (black or white) places a single piece on the board.
2. Each piece must be placed on the board and so that there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another piece of the same color, with one or more contiguous pieces of the opponent between them.
3. The pieces in between are flipped over.
4. The player with the most pieces on the board when there are no more legal moves to play wins.

You can play here: https://www.othelloonline.org/



Section #1
A game loop for RL

Let's play Othello!

Goal: Setup a game loop with two players for reinforcement learning experiments.

What do we need to learn to play a turn based game?

Game loop for a single episode:

```
while (not finished):
    Player 1 chooses move
    Step board state
    Player 2 chooses move
    Step board state

compute episode return R
```

A single episode can be viewed from 2 perspectives. From the perspective of Player 1 we have the following data:

$s_0, a_0, s_1, a_1, s_2, a_2, s_3, a_3, ..., s_T, a_T, R$

Creating a random player

For a game with a discrete set of possible moves:

1. Query the game to find which actions are legal and how many there are (*N*).
2. Set the probability of each of these actions to 1/*N*.
3. Sample a move from a multinomial on *N* possible actions.
4. Return the chosen action.

$$\pi_\theta(a_t|s_t)$$

Exercise

- Call a game loop for Othello.
- Complete an agent that plays *random moves*.
- Generate games including wins and losses.

Section #2
Train a value function

What moves are good?

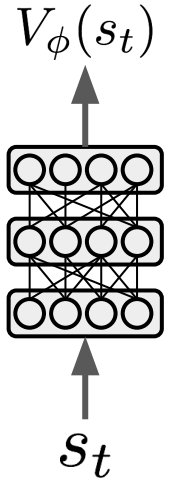
Goal: Learn how to train a value function from a dataset of games played by an expert.

Train a value function

Our value function $V(s_t)$ returns a number between -1 and 1, representing the predicted outcome of the game:
-1 is a loss, 1 is a win from the current player's perspective.

We put the linear output of the network through the tanh() function to bound it's outputs between [-1, 1].

We train via: $\nabla_\phi \mathcal{L} = \sum_{t=0}^{T} \nabla_\phi (R_t - V_\phi(s_t))^2$



Exercise

- Load a dataset of expert generated games.
- Train a network to minimize MSE for win/loss predictions given board states sampled from the dataset.

Section #3
Play games using a value function

Beat the random player!

Goal: Learn how to use a value function in order to create a player that works better than random.

Use the value function to choose an action

Using a model of the environment, value functions can be used to rank potential actions. Then, actions can be chosen according to their rank. For example:

for i in 1 to k:

Choose a random legal action a^i for the current state s_t.
Step the environment: s_{t+1} = env.step(s_t, a^i)
Estimate the value: V(s_{t+1})
Build an array of [V(s_{t+1}), a^i] pairs.

To act, choose the action associated with the highest value.

Exercise

- Sample some random moves and use the value function to rank them.
- Choose the best move as the action and play it.
- Show that doing so beats the random player.

Section #4
Train a policy network

Imitate the expert!

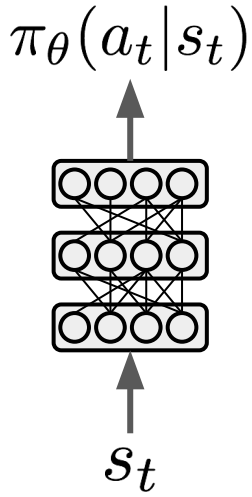
Goal: Learn how to train a policy network via supervised learning / behavioural cloning.

Train a policy network from expert games

- Our policy function will produce a categorical distribution over all possible discrete actions.
- We train on the moves provided in the expert dataset and the output is the softmax function.

- We train to maximize the likelihood of the actions:

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau \sim D_{\pi^*}} \left[\sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) \right]$$



Exercise

- Train a network to predict the next move in an expert dataset
- Do so by minimizing the negative log likelihood of the next action given the current board position.

Section #5
Play games using a policy network

Beat the value function player!

Goal: Learn how to use a policy network to play games.

Creating an agent using a policy function

Now that we have a policy that has been trained via imitation learning on expert data, we can use it to play games.
For the current state, we can sample actions according to:

$$\pi_\theta(a_t|s_t)$$

1. The action with the highest probability (the "greedy" action).
2. An action according to the categorical distribution determined by the softmax output:

$$p(a^i|s) = \frac{\exp(-z_i/T)}{\sum_{j=1}^{K}\exp(-z_j/T)}$$

Exercise

- Use the policy network to determine the next move for a player.
- Build a player that takes the move given the maximum probability by the network.
- Compare this to another player that samples moves according to the probability distribution output by the network.

Section #6
Plan using Monte Carlo rollouts

Where will imagination take us?

Goal: Learn the core idea behind using simulated rollouts to understand the future.

Imagining the future using a model

We can string together calls to our policy function with calls to a model of the game to imagine states in the future.

Then, we can call our value function to estimate how good they are:

$$s_t \xrightarrow[\pi_\theta(a|s)]{} a_t \xrightarrow[p(s'|s,a)]{} s_{t+1} \xrightarrow[\pi_\theta(a|s)]{} a_{t+1} \xrightarrow[p(s'|s,a)]{} s_{t+2} \rightarrow V(s_{t+2})$$

Exercise

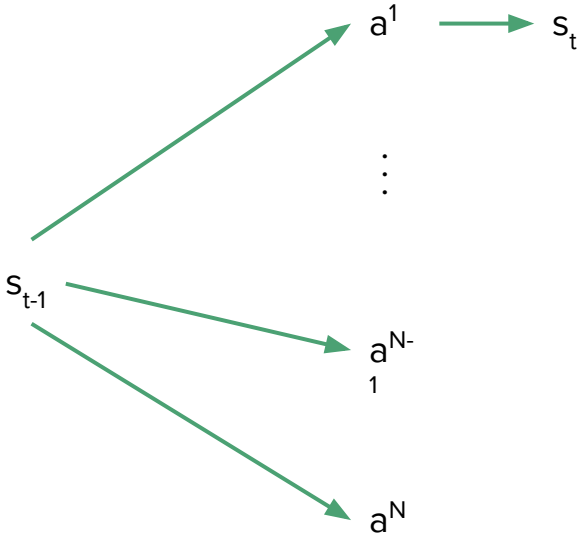
- Complete a loop to run a Monte Carlo simulation using the policy network.
- Compute the value function at the end of *k=3* steps for a given board position.

Section #7
Play with planning

Plan to win!

Goal: Learn how to use simple Monte Carlo planning to play games.

Use Monte Carlo imaginations to play



$$s_t \xrightarrow[p(s'|s,a)]{\pi_\theta(a|s)} a_t \rightarrow s_{t+1} \xrightarrow[p(s'|s,a)]{\pi_\theta(a|s)} a_{t+1} \rightarrow s_{t+2} \rightarrow V(s_{t+2})$$

$$s_t \xrightarrow[p(s'|s,a)]{\pi_\theta(a|s)} a_t \rightarrow s_{t+1} \xrightarrow[p(s'|s,a)]{\pi_\theta(a|s)} a_{t+1} \rightarrow s_{t+2} \rightarrow V(s_{t+2})$$

Average Values

Use Monte Carlo imaginations to play

Now that we can imagine the future and estimate its value, we'll use this capacity to create an agent that plans:

for i in 1 to *k*:

Choose the *i*th ranked action a^i for the root state s_t according to our policy.
Run *N* Monte Carlo rollouts from s_{t+1} follows from the application of a^i
Average the estimated values for each rollout to get: V_i^{AVG}
Build an array of $[V_i^{AVG}, a^i]$ pairs.

To act, choose the action associated with the highest value.

Exercise

- Incorporate Monte Carlo simulations into an agent.
- Do this by choosing the best 3 actions suggested by the policy function and running 10 MC simulations for a depth of 3 from each of these.
- Choose the best of these 3 moves according to the MC rollouts (i.e. which one has the highest estimated value).
- Run the resulting player versus the random, value-based, and policy-based players.

Section #8
Unbeatable Opponents

Will we still play games?

Goal: Engage with the social and ethical ramifications of having unstoppable computer players for games.

Exercise

Read a recent article on technology based cheating in chess:

Chess's cheating crisis: 'paranoia has become the culture'

Exercise

Reflect on related questions:

- Why do humans play games?
- What kinds of games can AI be applied to to win / what kinds of games remain out of reach and why?
- What happens if top human play hinges increasingly on access to compute?
- What will happen to fairness for other intellectual pursuits?
    - https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3

Where to next?

A glimpse of the future

Goal: Understand some of the open problems for the next phase of learning+planning algorithms.

What can't our algorithms do?

The last half decade has given rise to a growing set of powerful reinforcement learning algorithms that are capable of competing with top human players for a widening set of games.

Variations of these are being applied throughout game playing, and we're beginning to see strong solutions to Poker, Starcraft, DotA, Hanabi, and others.

There are, however, clear limitations to the AlphaX algorithms.

What can't our algorithms do?

Limitations of the AlphaX algorithms include:

1. They use a perfect model of the environment.
2. MCTS plans step-by-step.
3. MCTS plans in a strictly forward mode.

Humans appear to have the capacities to plan in ways that AlphaX algorithms cannot:

1. They appear to learn models of the environment.
2. They appear to plan in temporally abstract jumps.
3. They appear to be able to plan in mixes of forward and backward modes.

We have made some progress on #1

Limitations of the AlphaX algorithms include:

1. ***They use a perfect model of the environment.***
2. MCTS plans step-by-step.
3. MCTS plans in a strictly forward mode.

Humans appear to have the capacities to plan in ways that AlphaX algorithms cannot:

1. ***They learn models of the environment.***
2. They appear to plan in temporally abstract jumps.
3. They appear to be able to plan in mixes of forward and backward modes.

Learning models of the world

Reinforcement learning algorithms are increasingly making use of learned models of the environment. The essential idea is to train a model that makes predictions about about how actions lead to future states, values, actions, or rewards.

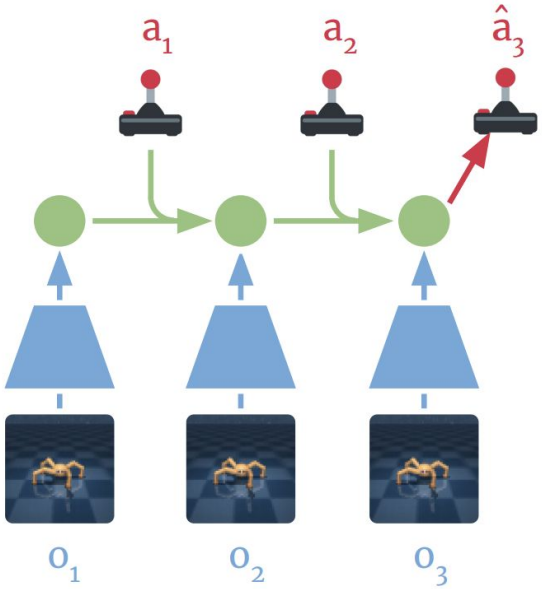
Such models can be used in a variety of ways. We highlight two:

1. *Dreamer* learns a model of the world in order to be able to rollout experience in imagination and then learn behaviour from these fictitious experiences.
2. *MuZero* builds a model of the world that is tailored to the requirements of MCTS, enabling planning in a learned state space.

$$\rho\left(s_{t+1} \middle| s_t, a_t\right)$$

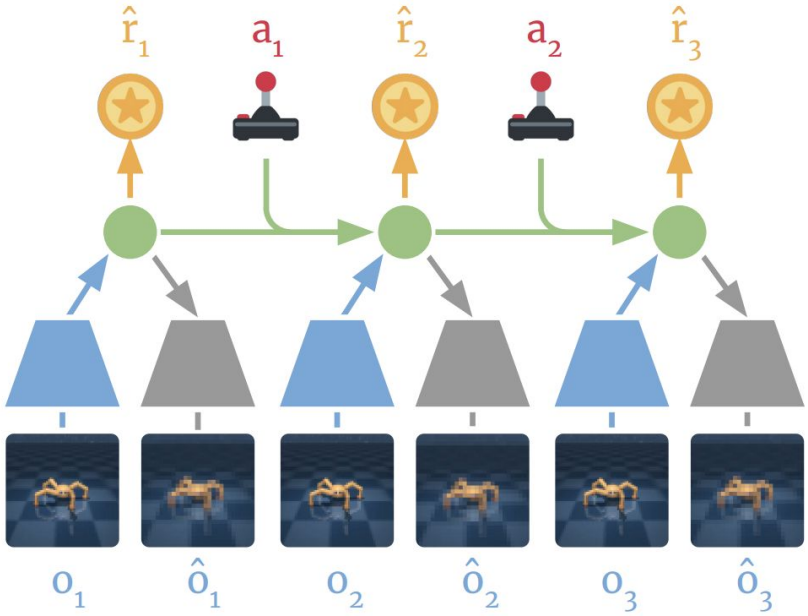
Hafner, Lillicrap, Ba, Norouzi *arXiv*, 2020

Dreamer: Act in the environment



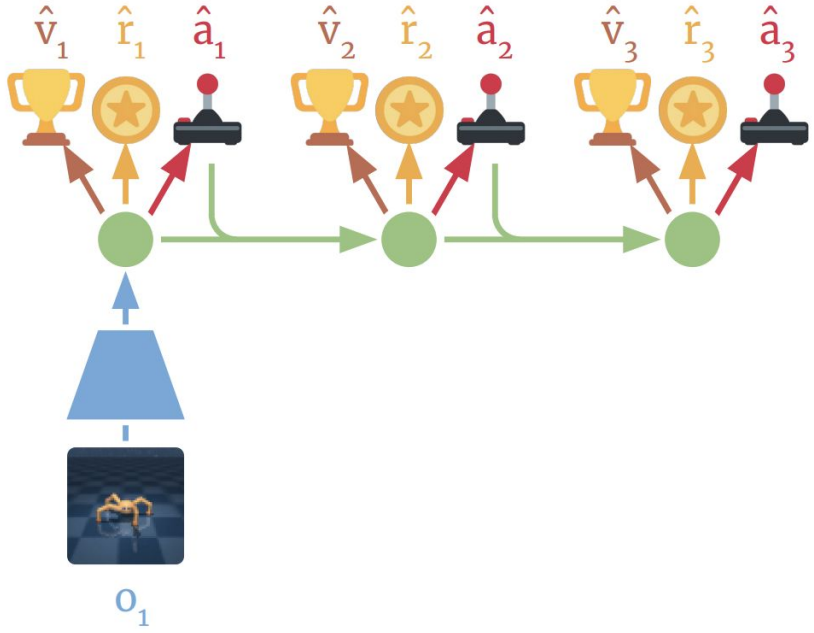
Hafner, Lillicrap, Ba, Norouzi *arXiv*, 2020

Dreamer: Learn dynamics from experience



Hafner, Lillicrap, Ba, Norouzi *arXiv*, 2020

Dreamer: Learn behaviour in imagination



Learn behaviour in imagination

Hafner, Lillicrap, Ba, Norouzi *arXiv*, 2020

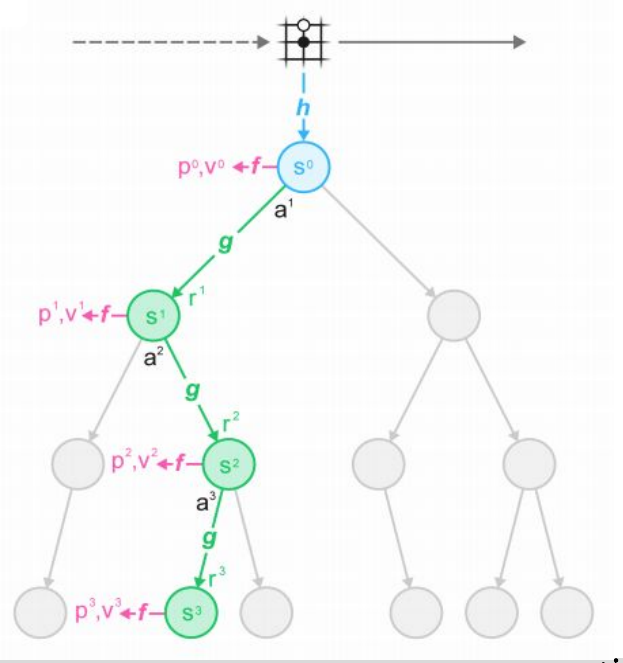
MuZero: Learning a model for planning

AlphaGo, AlphaGo Zero & AlphaZero use perfect models of the environment to plan.

- MuZero instead builds a model of the environment that can be used to search.
- Unlike most model-based work MuZero does not try to model environment transitions.
- Rather, it focuses on predicting the future reward, value, and policy.
- These are the essential components required by MCTS.

Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

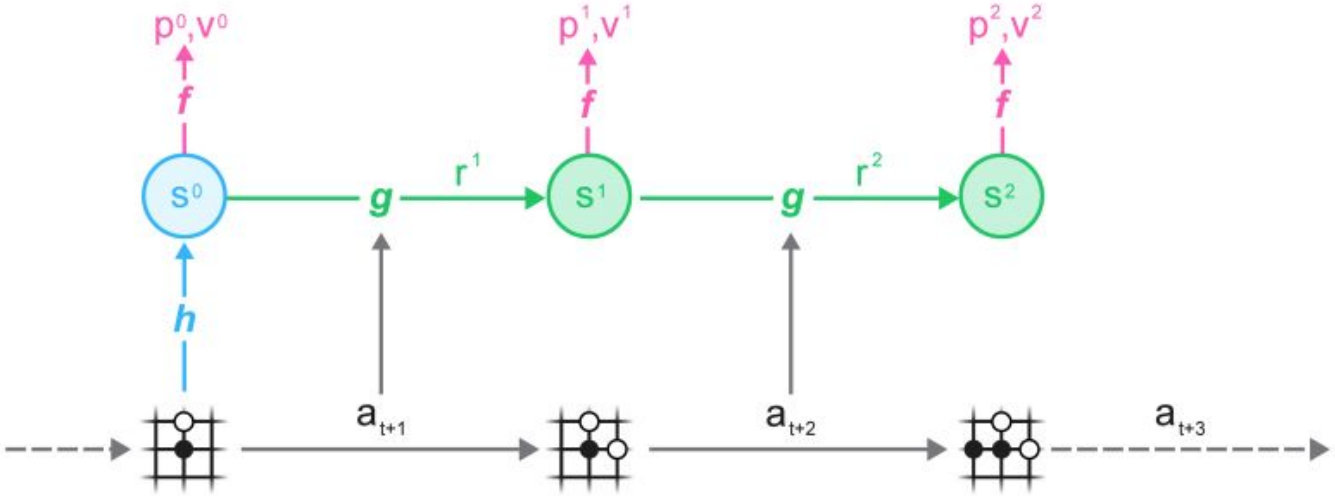
MuZero: Model design



Note that the transposition tables used in *AlphaGo* and *AlphaZero* are not possible in *MuZero*.

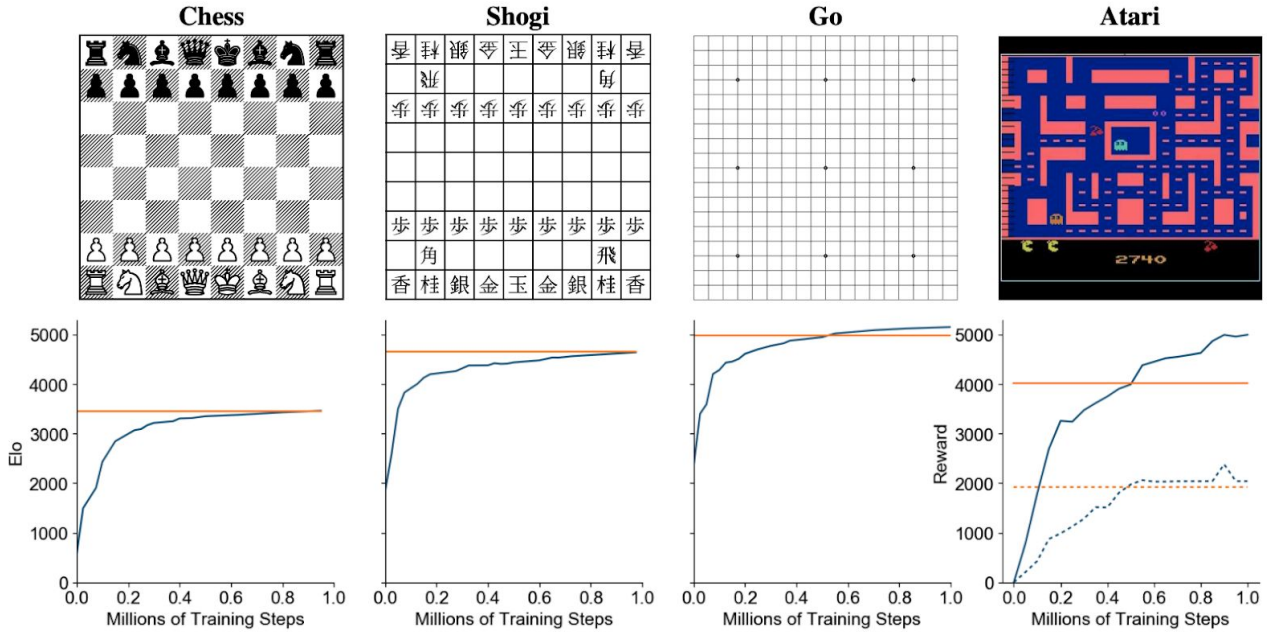
Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

MuZero: How it learns



Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

MuZero: Results



Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

Some lessons learned

1. There is are no neat distinction between model-free and model-based reinforcement learning. There's a huge space of possibilities and we're just beginning to explore what can be done.

2. It's unclear precisely why model-based algorithms are starting to work well. Larger networks appear to be part of the equation.

3. The environment/problem studied is crucial for results. Planning at run-time has been convincingly demonstrated to be important for games like Go, chess, and shogi. The advantage of planning is less clear for games like Atari.

Where to next?

We're still missing algorithms that have the flexibility of human planning.

Most of our algorithms still rely on forward-mode planning and rollouts.

And, none of them exhibit the kinds of state, action, and temporal abstraction exhibited during human planning and behaviours.

These are exciting research questions with deep connections to the neurosciences!