DONDERS INSTITUTE

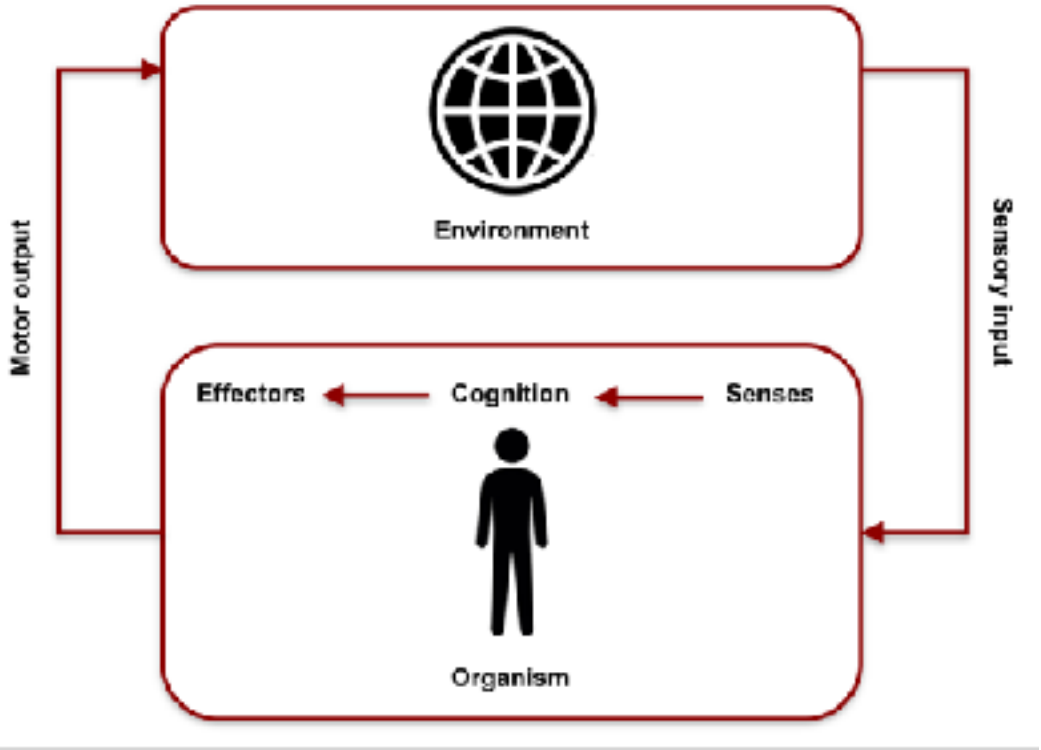
Neural Information Processing Systems

Intelligent Agents

Marcel van Gerven

Radboud University Radboudumc

Beyond sensations: Closing the perception-action cycle



Outline

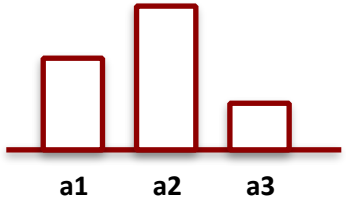
- Learning to behave
- RL algorithms
  - Policy gradient
  - Value-based RL
- New developments

Learning to behave

Policy

A policy π specifies the probability

$$\pi(\mathbf{a} \mid \mathbf{s})$$



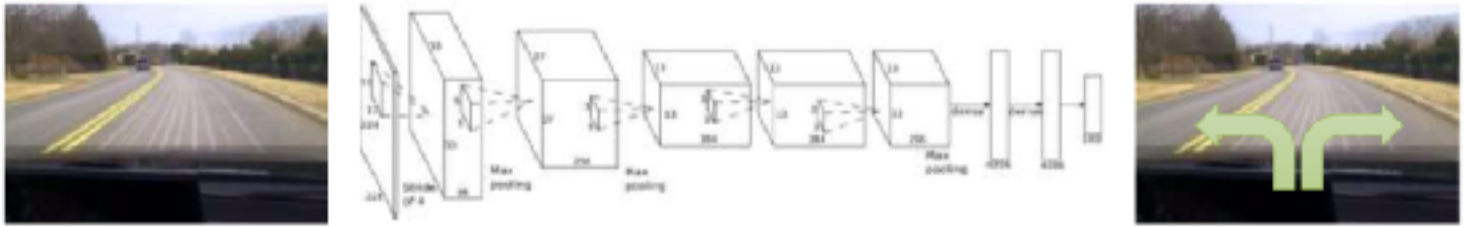
of selecting action(s) **a** in state **s**

A policy can be deterministic or stochastic

A policy can be discrete or continuous

How to learn this policy?

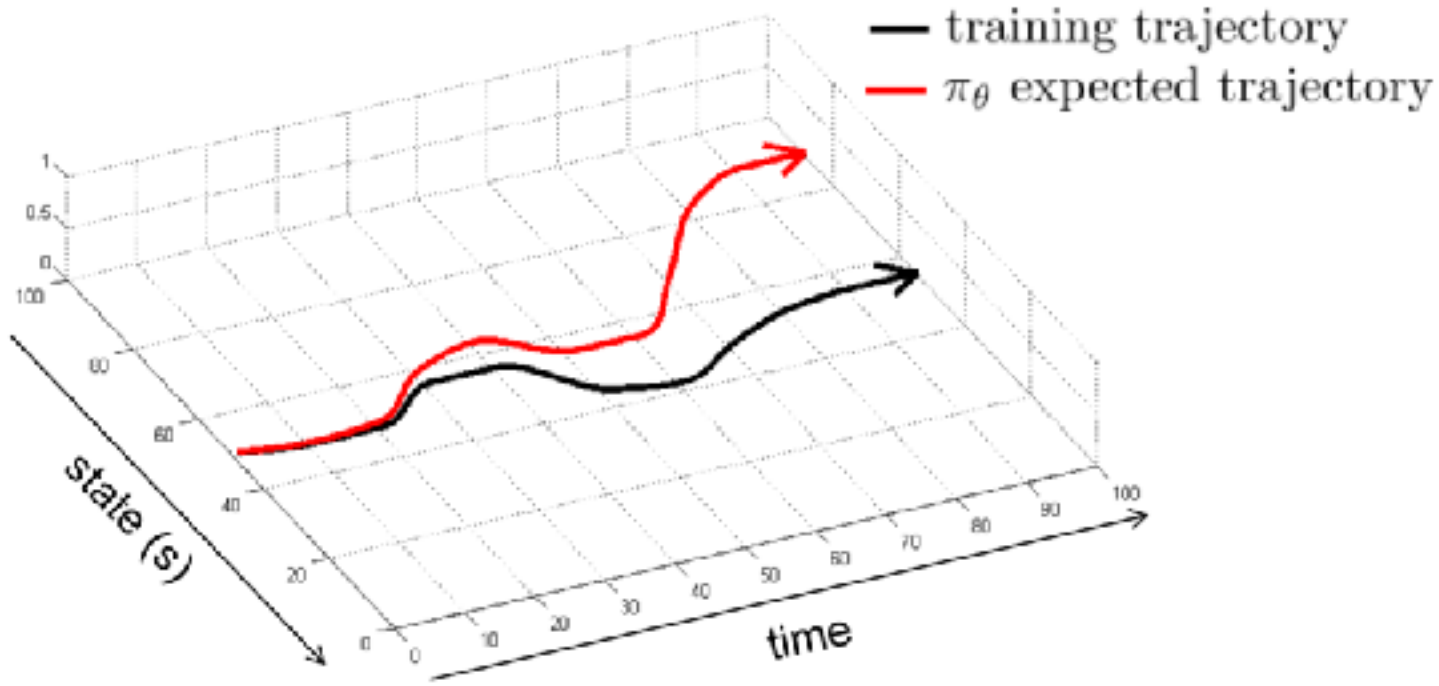
Imitation learning



$\mathbf{s}$ $\pi(\mathbf{a} \mid \mathbf{s})$ $\mathbf{a}$

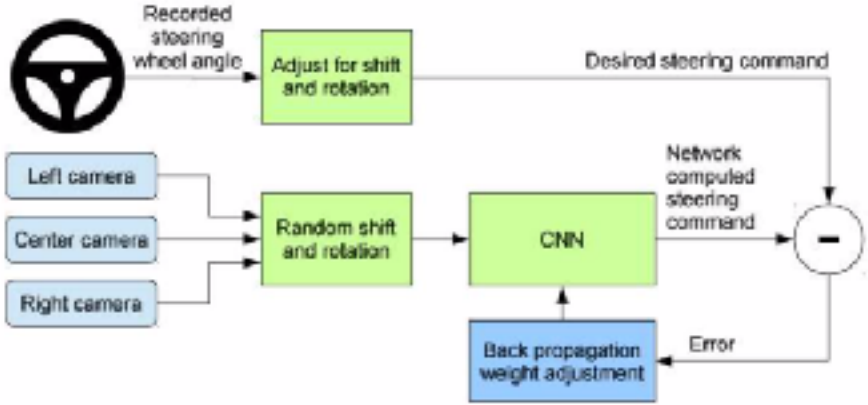
https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

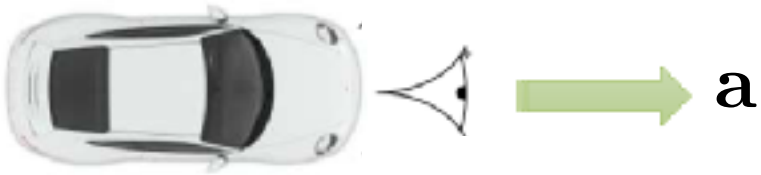
Stability issues



http://rail.eecs.berkeley.edu/deeprlcourse/

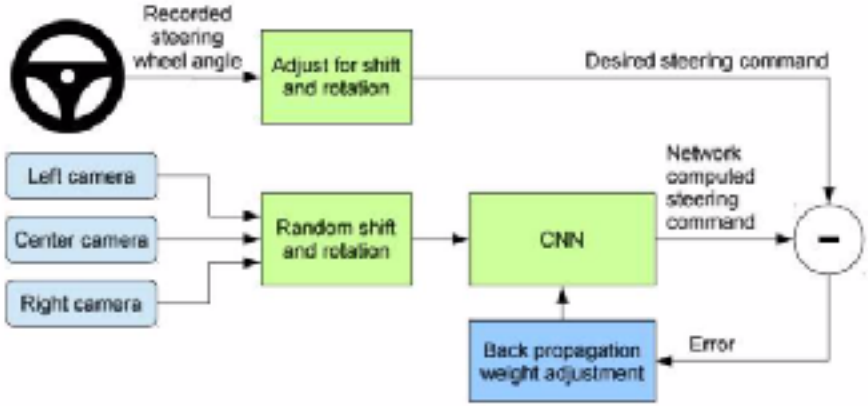
Possible solution





https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

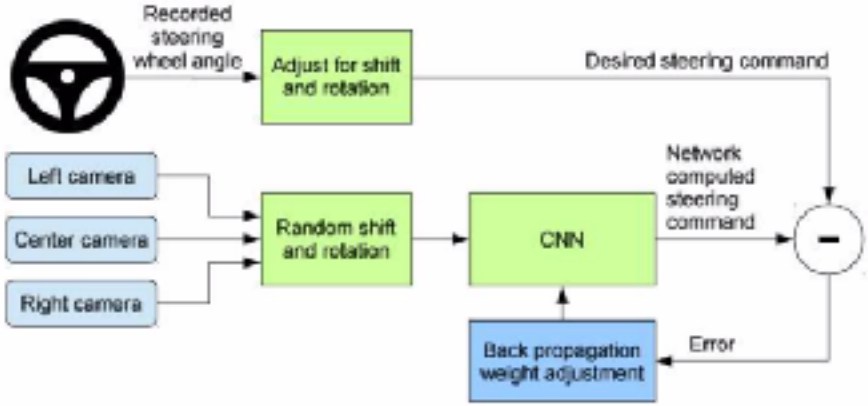
Possible solution



bias $\mathbf{a}$ to the right

https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

Possible solution



bias a to the right

https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

Reinforcement learning

Suppose we don’t know the correct action but just how desirable an action is?

Answer: Reinforcement learning (RL)

RL is a general-purpose framework for artificial intelligence

- RL can be used by an agent with the capacity to act
- Each action influences the agent's future state
- Success is measured by a scalar reward signal
- RL selects actions to maximise future reward
- The essence of an intelligent agent

Reward

Given a state $\mathbf{s}_t$ and an action $\mathbf{a}_t$ selected by the policy the environment produces reward r_{t+1}

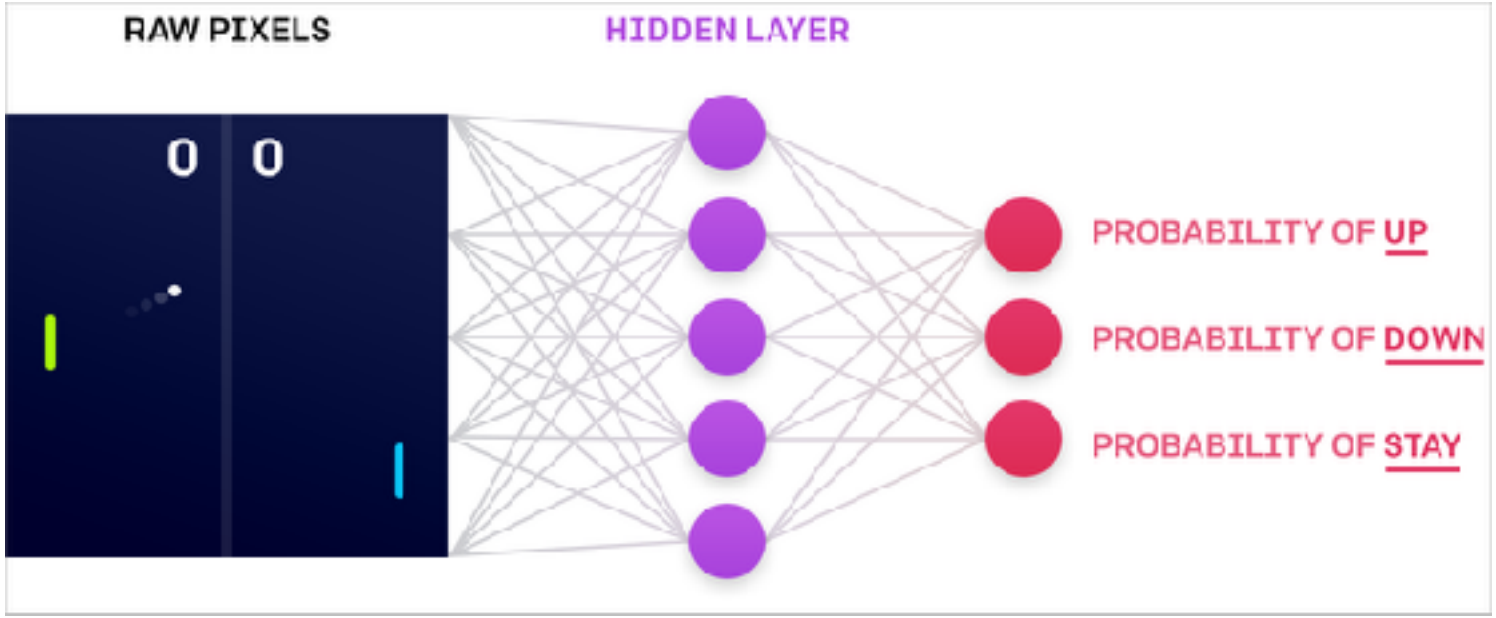
The total reward is defined by:

$$r(\tau) = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$$

where $\tau = (\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)$ is some trajectory

Policy gradient

Policy gradient approach



Assume a parameterized policy: $\pi_{\theta}(\mathbf{a} \mid \mathbf{s})$

Optimal policy

Define the probability of a trajectory under a policy:

$$\pi_{\boldsymbol{\theta}}(\tau) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

Define expected return: $J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}(\tau)} [r(\tau)]$

Find policy that maximizes the expected return:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Policy gradient approach

We write:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}(\tau)} \left[r(\tau) \right] = \int \pi_{\boldsymbol{\theta}}(\tau) r(\tau) d\tau$$

We optimize the parameters via gradient ascent:

$$\nabla J(\boldsymbol{\theta}) = \int \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\tau) r(\tau) d\tau$$

a convenient identity

$$\pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) = \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau)$$

Score function

Policy gradient approach

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}(\tau)} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau) r(\tau) \right]$$

We have:

$$\log \pi_{\boldsymbol{\theta}}(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) + \log p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

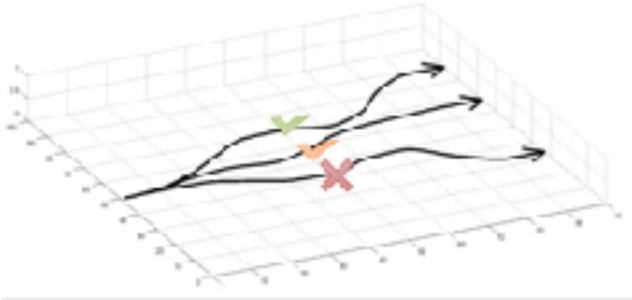
Therefore:

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

Evaluating the policy gradient

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

Sample over episodes



Therefore:

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

REINFORCE algorithm

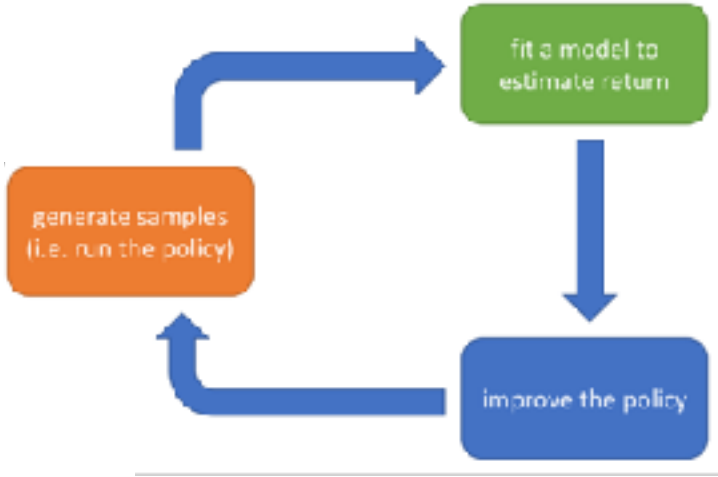
$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t}\mid\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T}r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

Allows gradient ascent steps of the form: $\Delta\boldsymbol{\theta} = \eta\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$

1. Sample $\{\tau^i\}$ from $\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)$ (run the policy)

2. $\nabla J(\boldsymbol{\theta}) \approx \sum_i \left(\sum_t \nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t}\mid\mathbf{s}_{i,t})\right)\left(\sum_t r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$

3. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$



Reducing variance: Cost to go

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

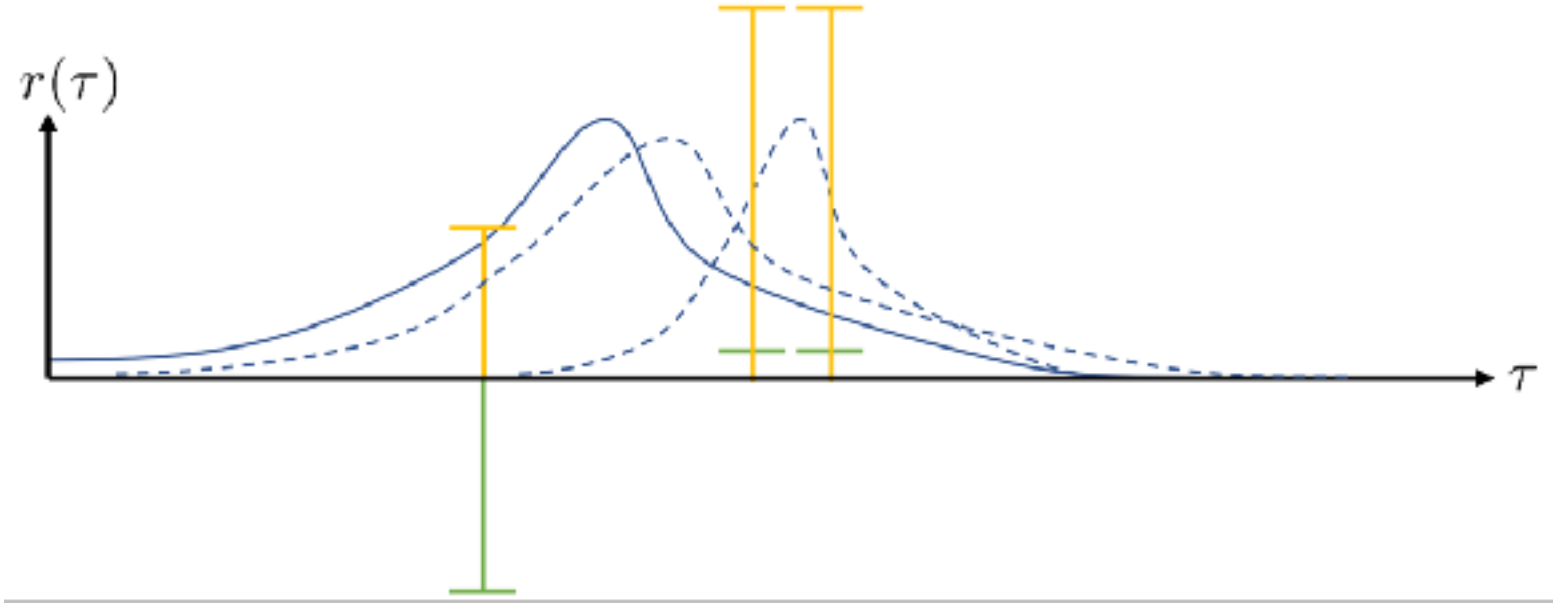
$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}) \left(\sum_{t'=1}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

Causality: policy at time t' cannot affect reward at time t < t'

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

where $\hat{Q}_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$ is known as the 'cost to go'

Reducing variance: Adding a baseline



Reducing variance: Adding a baseline

Policy gradient:

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau_i) \left[r(\tau_i) - b \right]$$

with $b = \frac{1}{N} \sum_{i=1}^{N} r(\tau_i)$

Reduces the variance

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau_i) = \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})$$

$$r(\tau_i) = \sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

Subtracting a baseline is unbiased in expectation:

$$\mathbb{E}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau) b] = \int \pi_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau) b d\tau = \int \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\tau) b d\tau = b \nabla_{\boldsymbol{\theta}} \int \pi_{\boldsymbol{\theta}}(\tau) d\tau = b \nabla_{\boldsymbol{\theta}} 1 = 0$$

Policy gradient with automatic differentiation

We want to compute the following using automatic differentiation:

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

Compare with standard maximum likelihood learning in supervised setting:

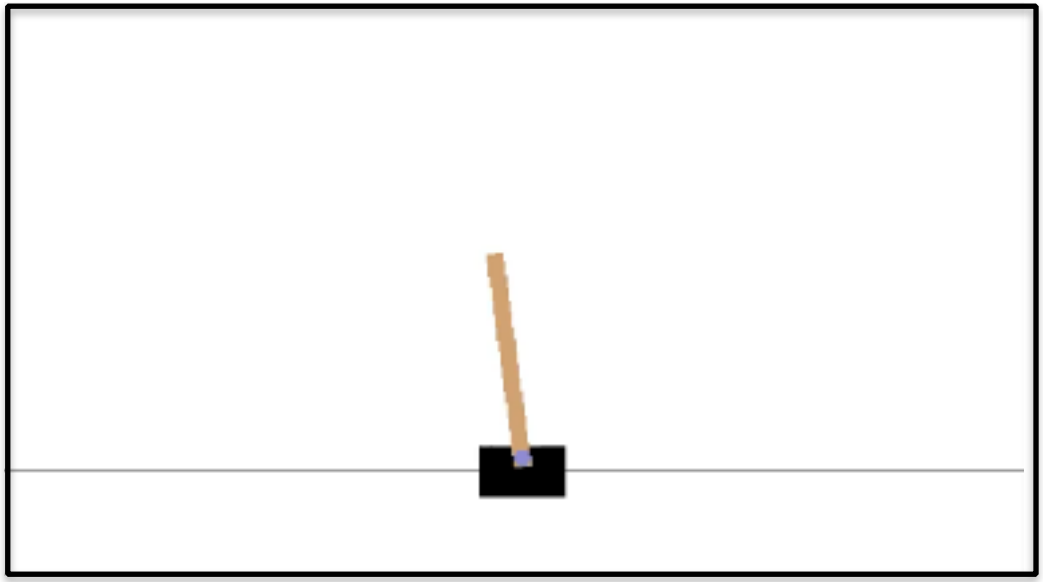
$$\nabla J_{\mathrm{ML}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}) \quad J_{\mathrm{ML}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})$$

Implement pseudo-loss as weighted maximum likelihood:

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

Practical assignment

Solve balancing pole problem using REINFORCE algorithm.



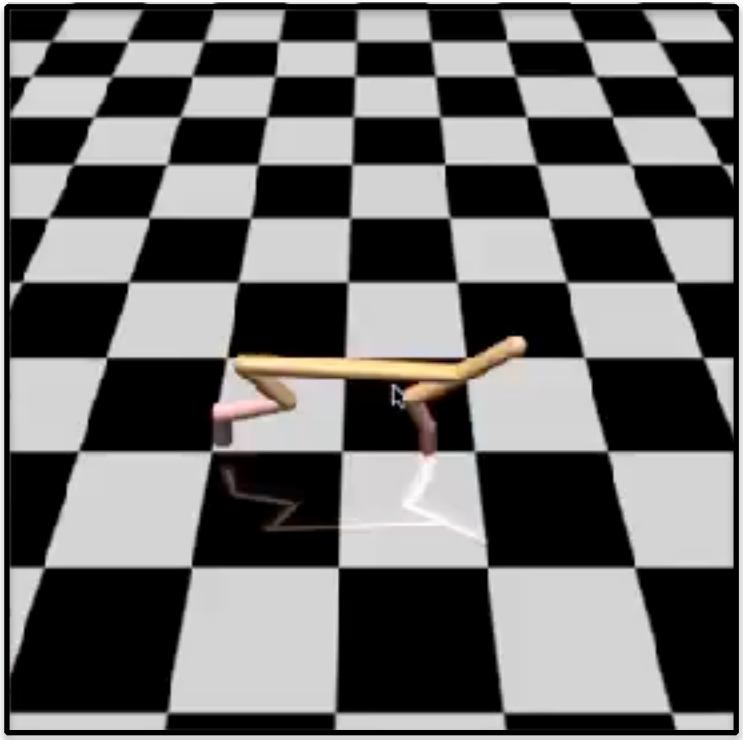
We use OpenAI Gym for this: https://gym.openai.com

Requires implementation of functions that compute the loss and score.

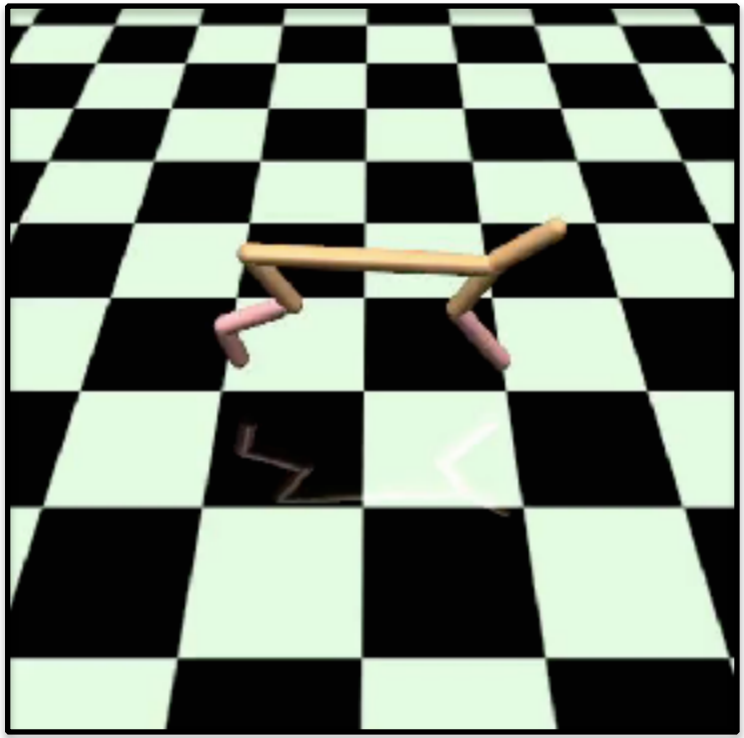
Policy gradient suggested readings

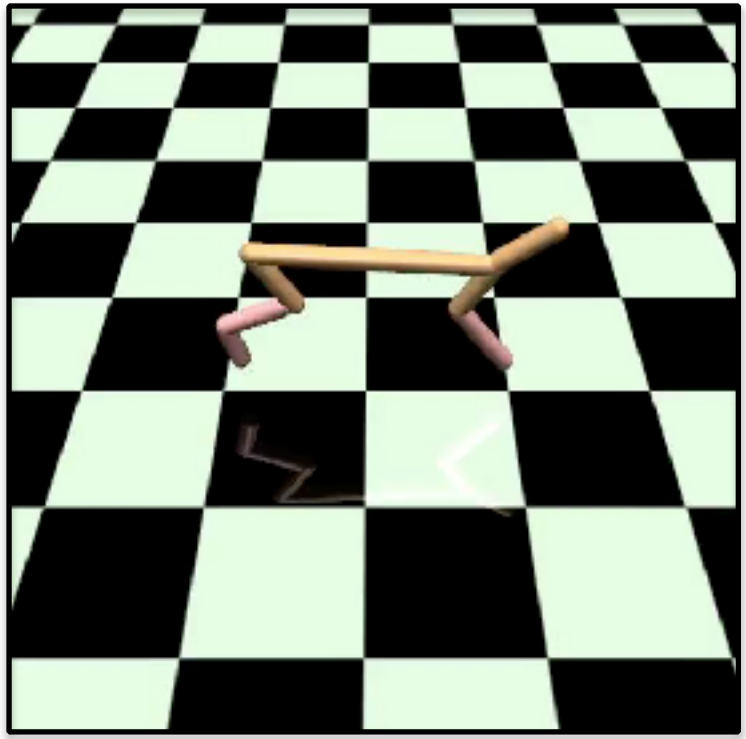
- Classic papers
  - Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning: introduces REINFORCE algorithm
  - Baxter & Bartlett (2001). Infinite-horizon policy-gradient estimation: temporally decomposed policy gradient (not the first paper on this! see actor-critic section later)
  - Peters & Schaal (2008). Reinforcement learning of motor skills with policy gradients: very accessible overview of optimal baselines and natural gradient
- Deep reinforcement learning policy gradient papers
  - Levine & Koltun (2013). Guided policy search: deep RL with importance sampled policy gradient (unrelated to later discussion of guided policy search)
  - Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
  - Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient

RL on half-cheetah

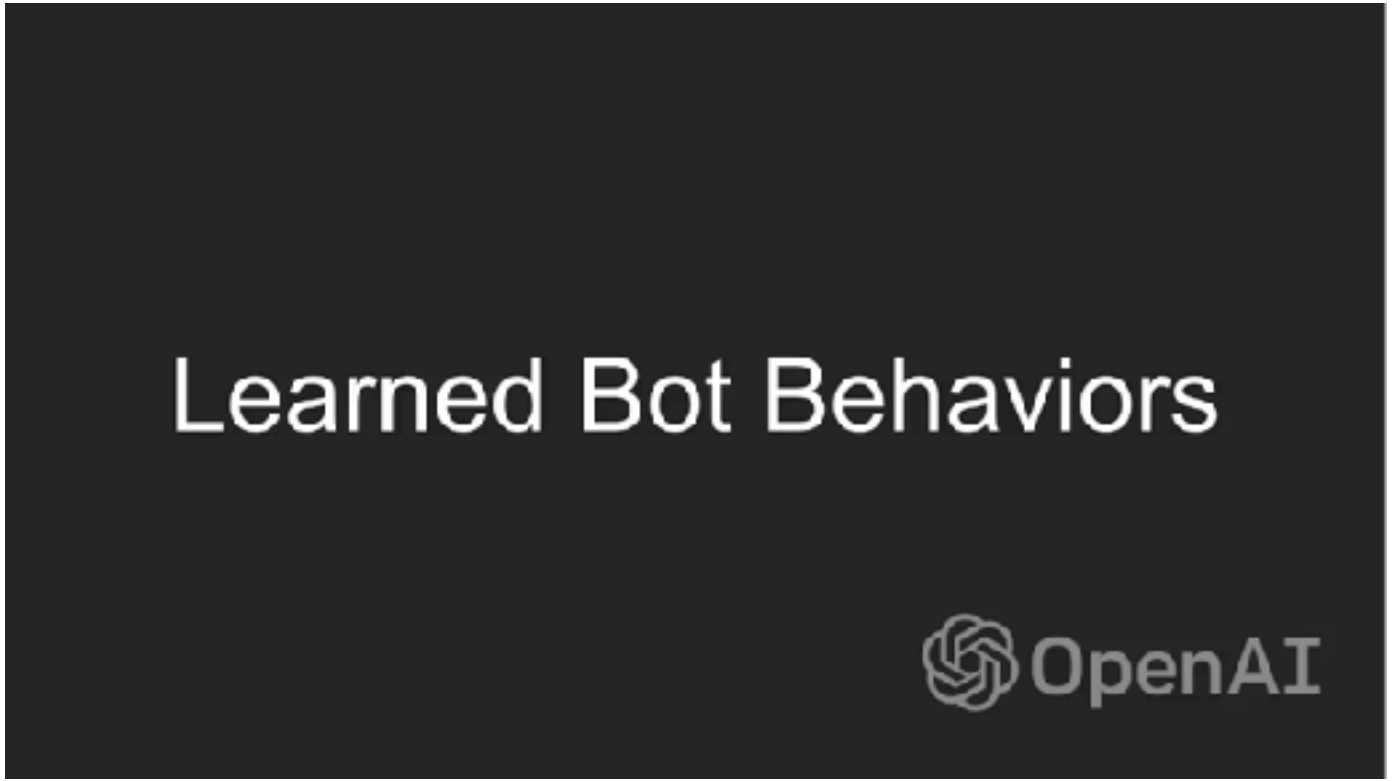


RL on half-cheetah





Playing DOTA 2 using Proximal Policy Optimization



https://blog.openai.com/openai-baselines-ppo/

Value-based RL

Action-value function

Q function (aka action-value function) expected reward to go when starting in state ***s*** and choosing action ***a*** under policy π:

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \mid \mathbf{s}_t, \mathbf{a}_t\right]$$

Measures the quality of an action ***a*** when being in a state ***s***

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Value-based RL: Q learning

Q function can be learned using a procedure called Q-learning:

- Before learning has started, *Q* returns an (arbitrary) fixed value
- Each time the agent selects an action, and observes a reward and a new state that may depend on both the previous state and the selected action, *Q* is updated.
- The core of the algorithm is a simple value iteration update.
- It assumes the old value and makes a correction based on the new information.

Q learning

Update equation:

TD error

$$Q_{t+1}(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q_t(\mathbf{s}_t, \mathbf{a}_t) + \eta \left(r_{t+1} + \gamma \max_{\mathbf{a}} Q_t(\mathbf{s}_{t+1}, \mathbf{a}) - Q_t(\mathbf{s}_t, \mathbf{a}_t) \right)$$

where $r_{t+1} = r(\mathbf{a}_t, \mathbf{s}_t)$, η is the learning rate and γ is a discounting factor

Q learning

The Q function is in essence one huge lookup table.

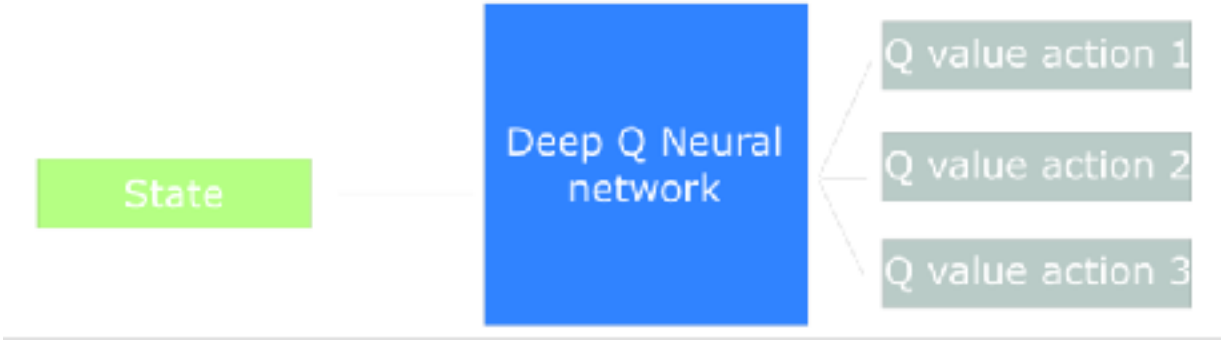
Becomes impossible to maintain for large (real-world) inputs (most states are never seen)

Solution: Represent action-value function by a function parameterized by θ

$$Q_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a}) \approx Q(\mathbf{s}, \mathbf{a})$$

Suitable parameter values may allow us to generalize to states we never encountered.

Deep Q learning



We can choose the parameterized function to be a deep neural network

Network takes states as input and produces value for each possible action as output

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Deep Q learning

Define objective function by mean-squared TD error:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}\left[\left(r_{t+1} + \gamma \max_{\mathbf{a}} Q_{\boldsymbol{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}) - Q_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}_t)\right)^2\right]$$

Leads to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta} = \mathbb{E}\left[\left(r_{t+1} + \gamma \max_{\mathbf{a}} Q_{\boldsymbol{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}) - Q_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}_t)\right) \frac{\partial Q_{\boldsymbol{\theta}}(\mathbf{s}_t, \mathbf{a}_t)}{\partial \theta}\right]$$

Optimise objective end-to-end by stochastic gradient descent.

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Stability issues

Naive Q-learning oscillates or diverges with neural nets:

1. Data is sequential

- Successive samples are correlated, non-iid

2. Policy changes rapidly with slight changes to Q-values

- Policy may oscillate
- Distribution of data can swing from one extreme to another

3. Scale of rewards and Q-values is unknown

- Naive Q-learning gradients can be large unstable when backpropagated

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Deep Q networks

DQN provides a stable solution to deep value-based RL:

1. Use experience replay

- Break correlations in data, bring us back to iid setting
- Learn from all past policies

2. Freeze target Q-network

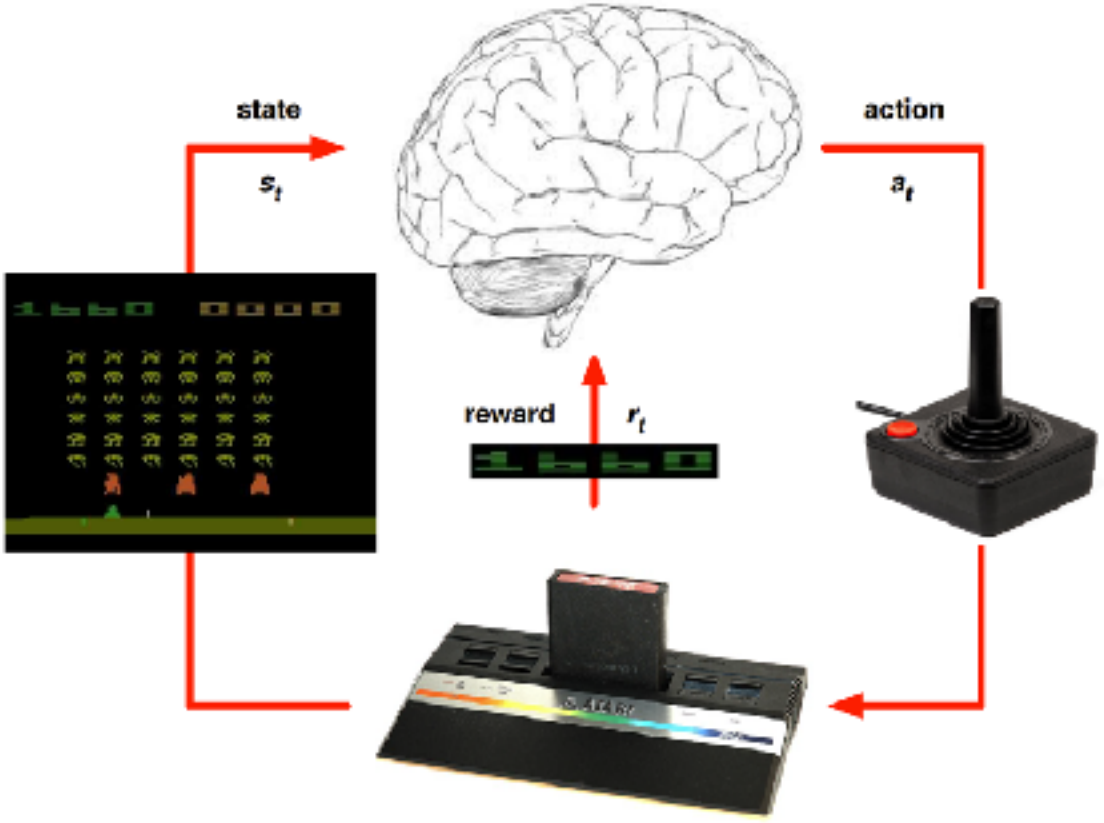
- Avoid oscillations
- Break correlations between Q-network and target

3. Clip rewards or normalize network adaptively to sensible range

- Robust gradients

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Playing Atari games with deep Q learning

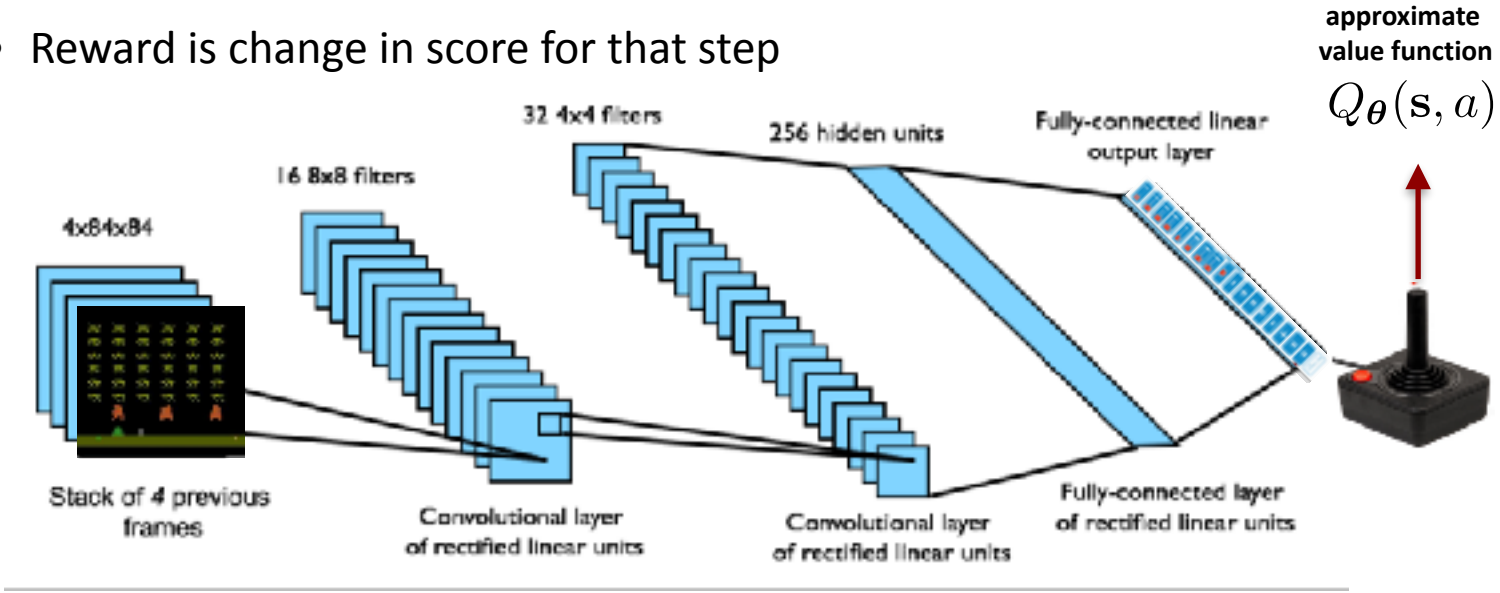


http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

DQN in Atari games

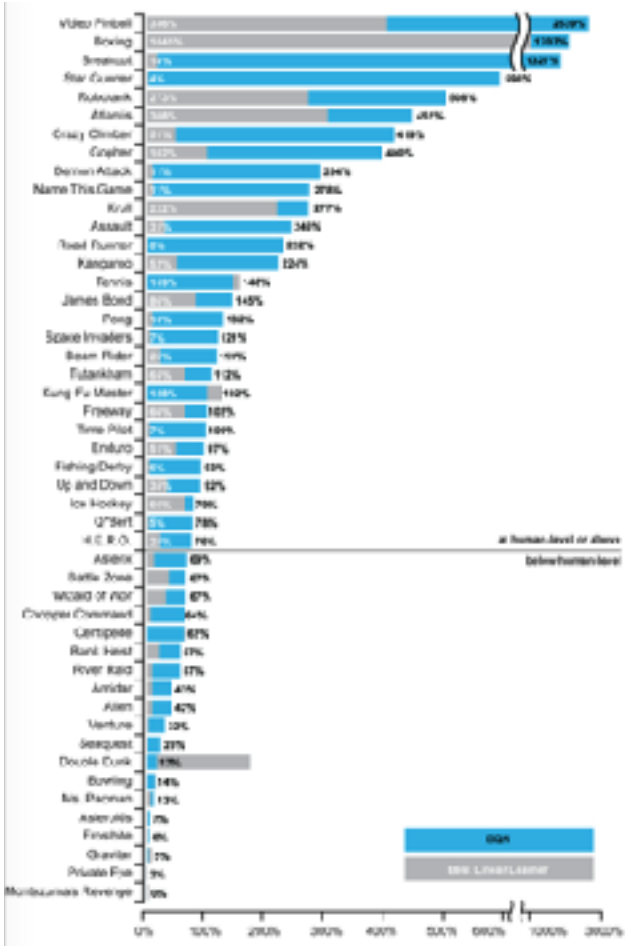
End-to-end learning of values *Q(s; a)* from pixels *s*

- Input state *s* is stack of raw pixels from last 4 frames
- Output is *Q(s; a)* for 18 joystick/button positions
- Reward is change in score for that step



http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

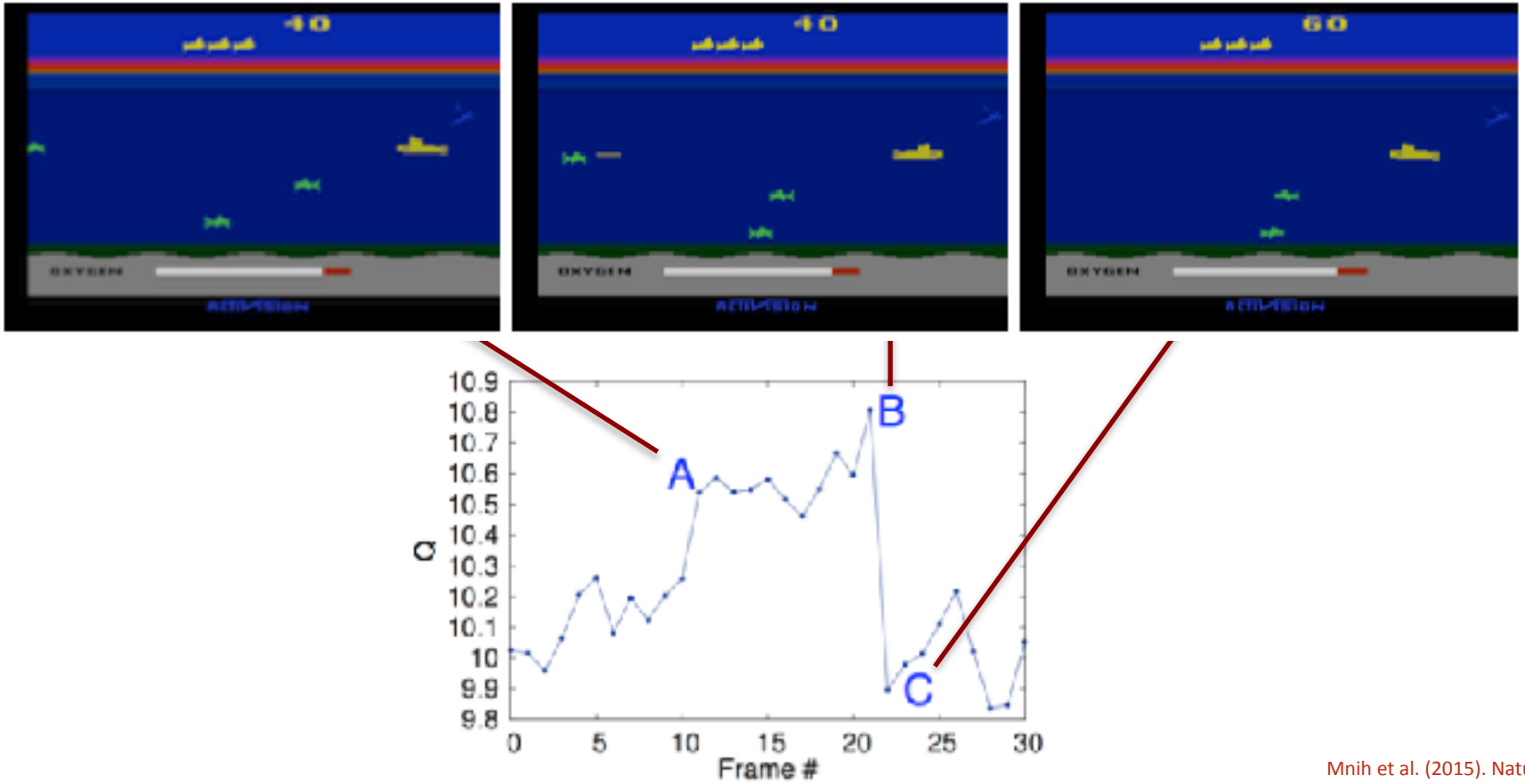
DQN results in Atari



Example: DQN playing Breakout

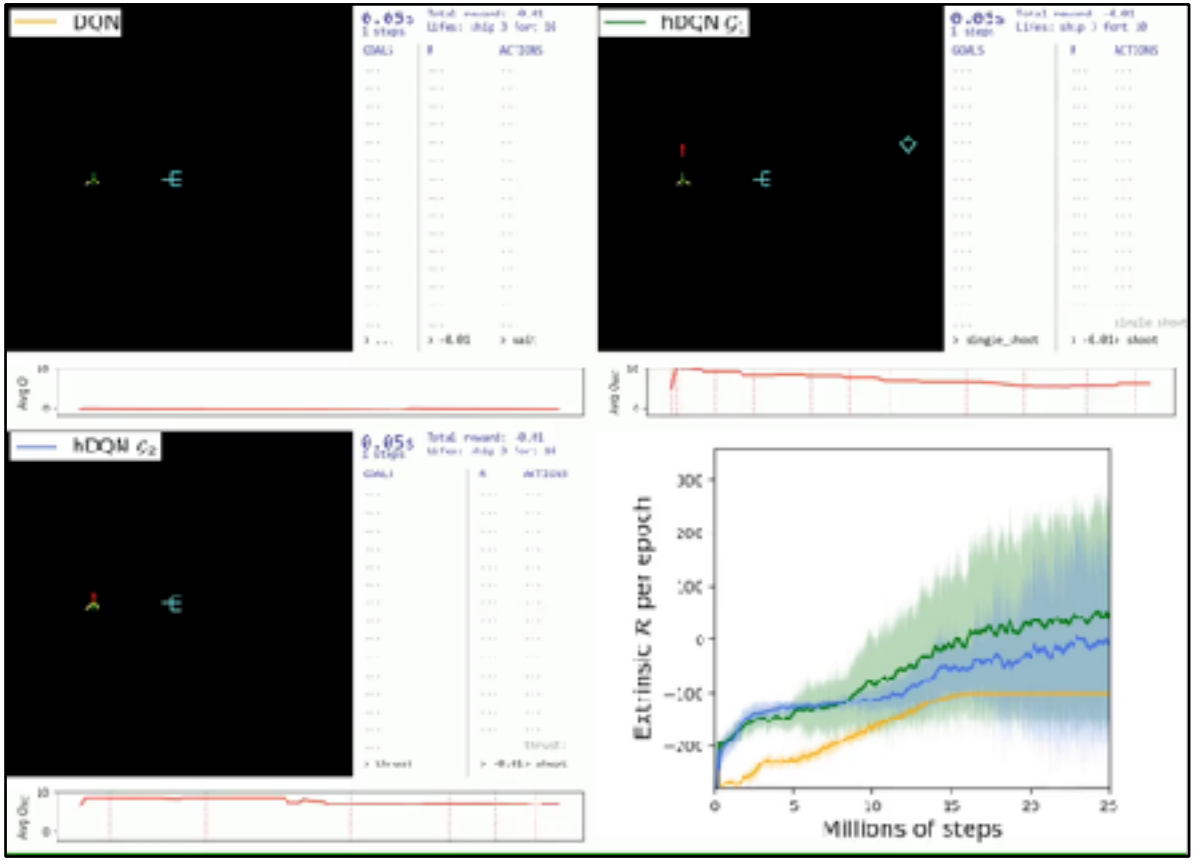
Mnih et al. (2015). Nature

Example: DQN playing Seaquest



Mnih et al. (2015). Nature

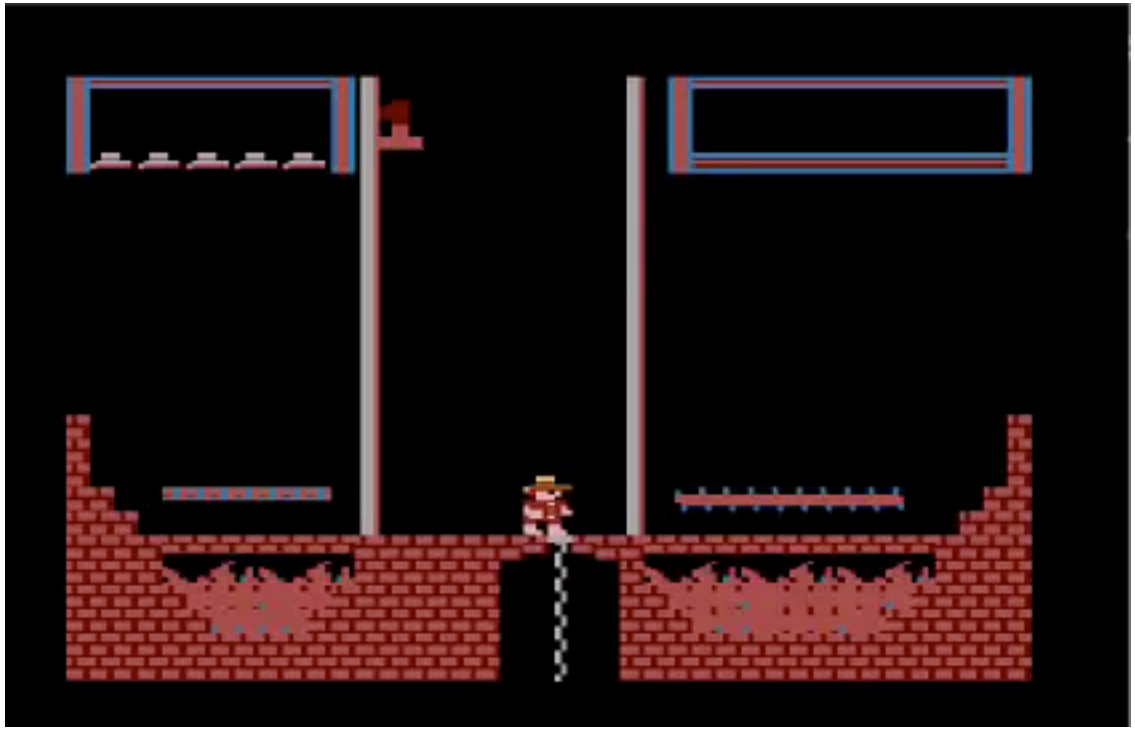
Hierarchical DQN



https://github.com/hipoglucido/Hierarchical-DRL-Tensorflow

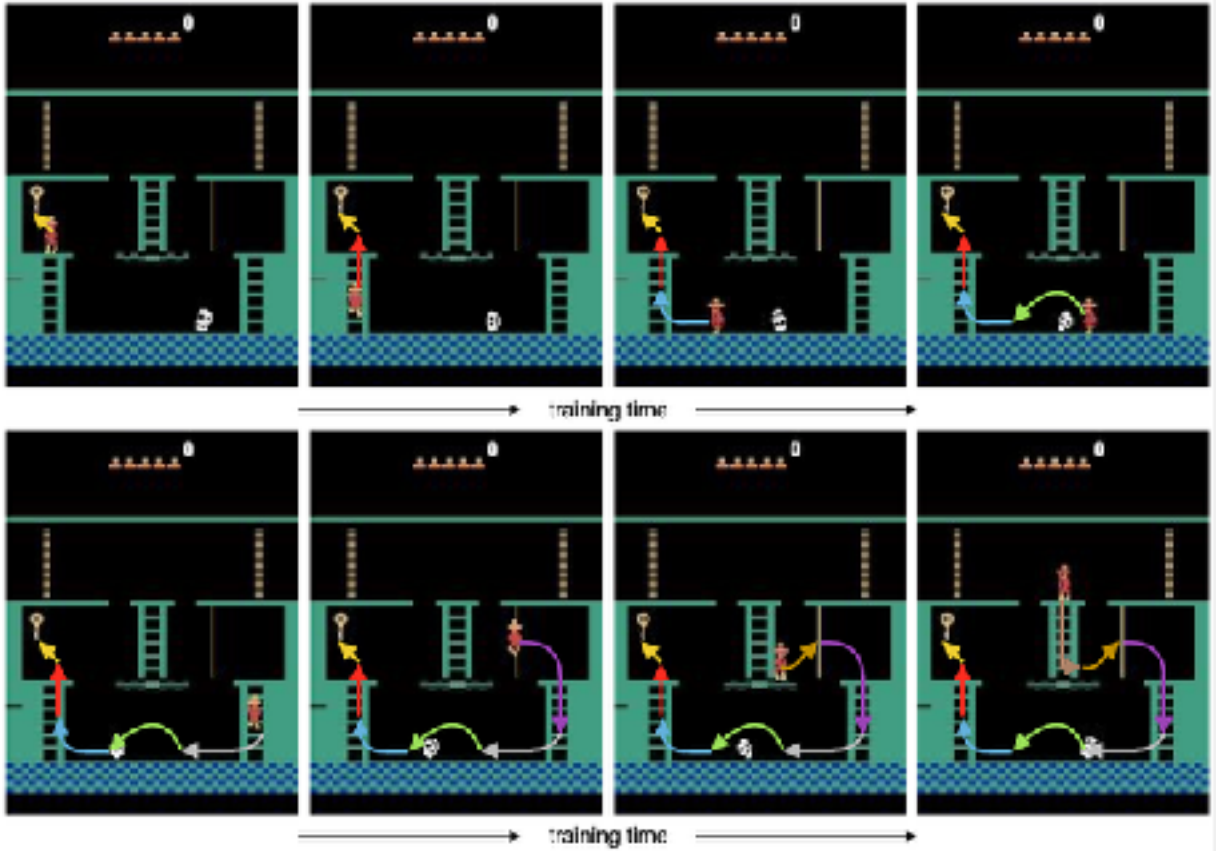
Montezuma's revenge

- Vanilla DQN fails on Montezuma's revenge



https://www.engadget.com/2016/06/09/google-deepmind-ai-montezumas-revenge/

Learning Montezuma's Revenge from a single demonstration



https://blog.openai.com/learning-montezumas-revenge-from-a-single-demonstration/

Approaches to reinforcement learning

Actor only

- Search directly for the optimal policy π^*
- This is the policy achieving maximum future reward

Critic only

- Estimate the optimal action-value function Q^*
- Can be used to determine the optimal action at each time step
- Naturally suited for discrete actions

Actor-critic

- Uses both an actor and a critic (as a baseline) to find the optimal policy

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Evolution strategies

Evolution strategies

In ES, we forget entirely that there is an agent, an environment, that there are neural networks involved, or that interactions take place over time, etc.

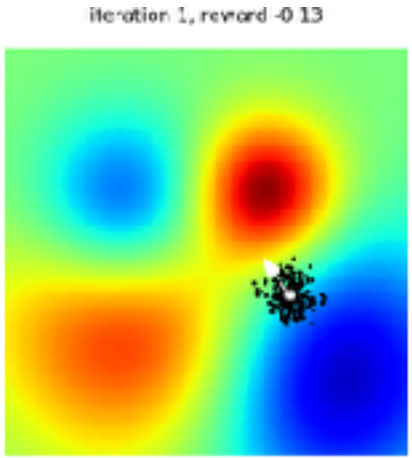
https://blog.openai.com/evolution-strategies/

ES algorithm

Intuitively, the optimization is a “guess and check” process, where we start with some random parameters and then repeatedly 1) tweak the guess a bit randomly, and 2) move our guess slightly towards whatever tweaks worked better.

https://blog.openai.com/evolution-strategies/

ES optimization process



https://blog.openai.com/evolution-strategies/

Tradeoffs between ES and RL

ES enjoys multiple advantages over RL algorithms:

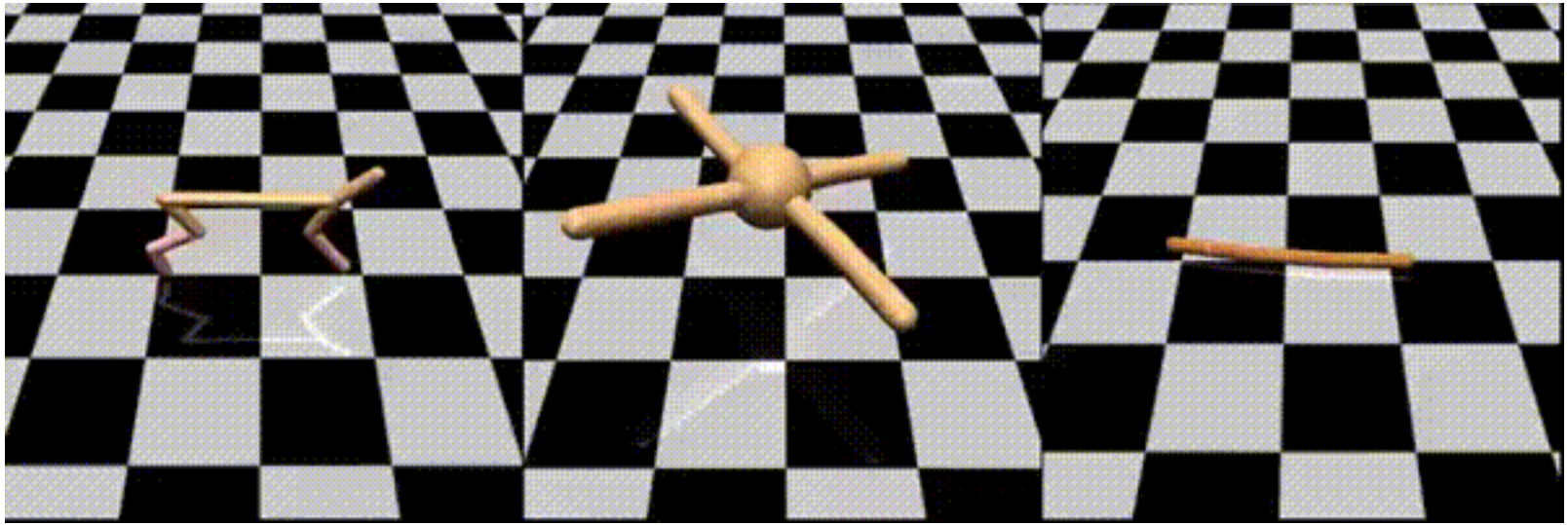
- No need for backpropagation.
- Highly parallelizable.
- Higher robustness.
- Structured exploration.
- Credit assignment over long time scales.

Challenge:

- Adding noise in parameters must lead to different outcomes to obtain some gradient signal.

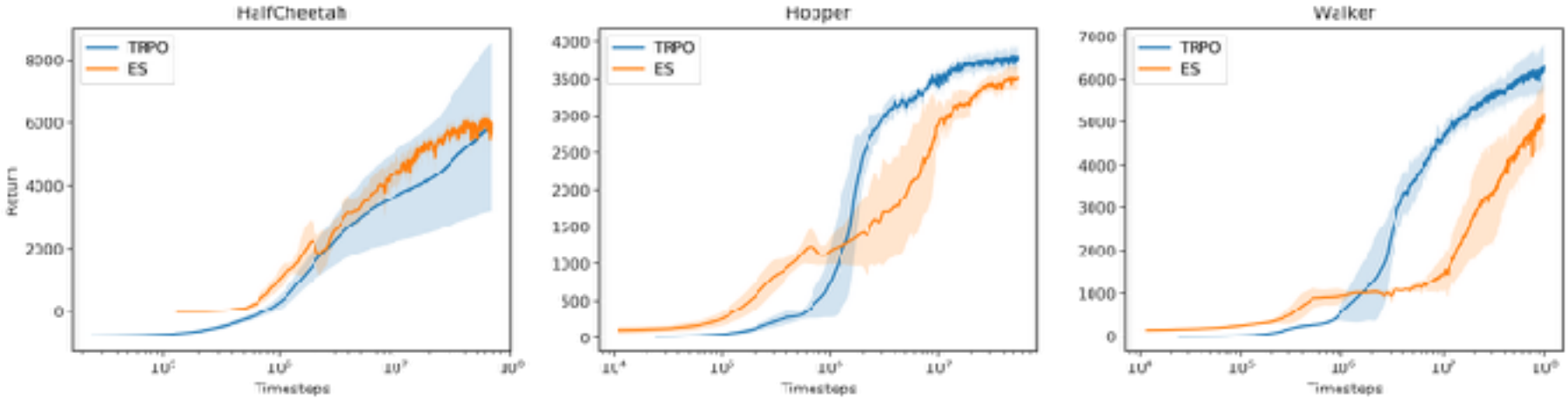
https://blog.openai.com/evolution-strategies/

ES on control tasks



https://blog.openai.com/evolution-strategies/

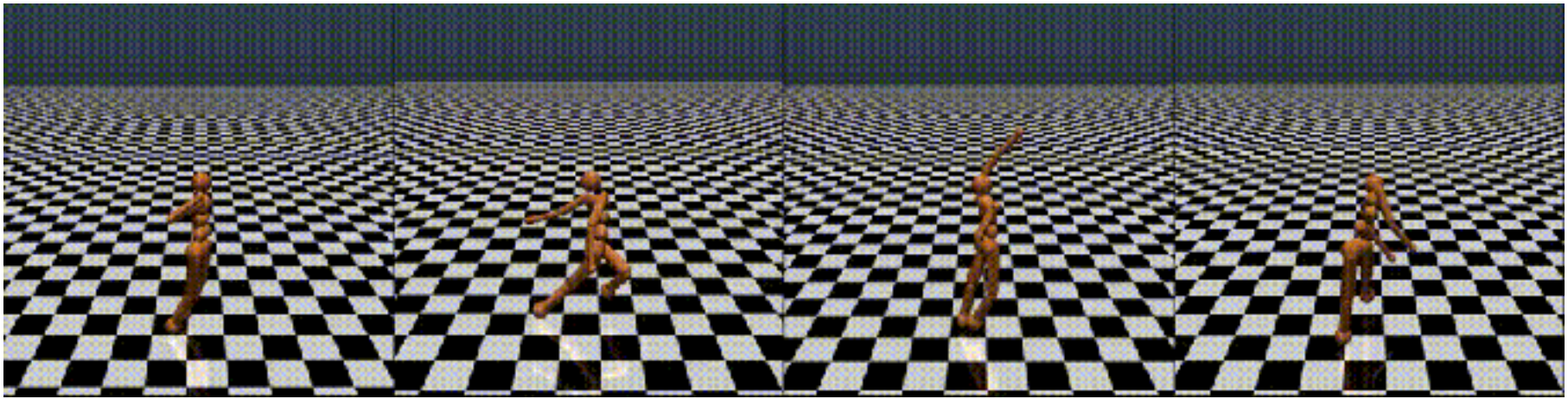
Performance comparison



Wall clock comparison: Running on a computing cluster of 80 machines and 1,440 CPU cores, ES is able to train a 3D MuJoCo humanoid walker in only 10 minutes (A3C on 32 cores takes about 10 hours)

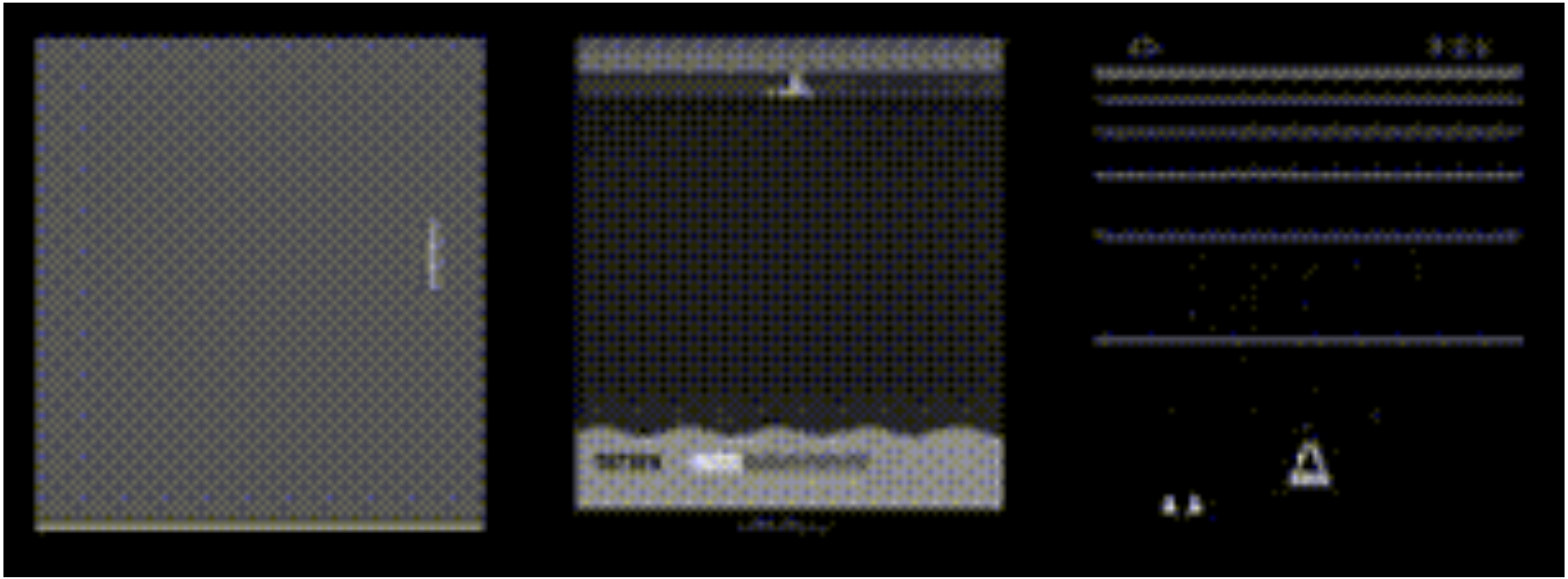
https://blog.openai.com/evolution-strategies/

ES on 3D humanoid walker task



https://blog.openai.com/evolution-strategies/

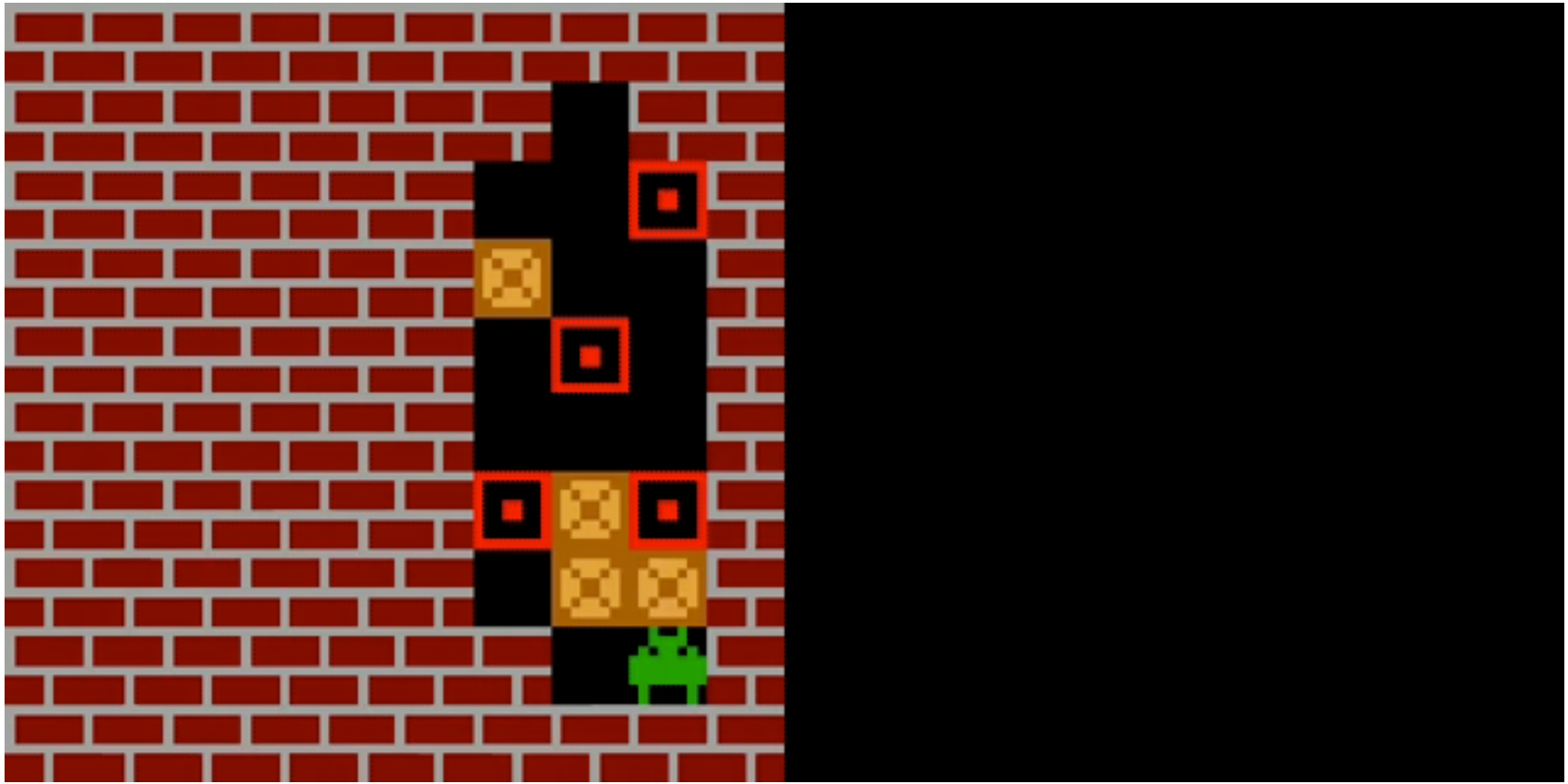
ES on Atari games



https://blog.openai.com/evolution-strategies/

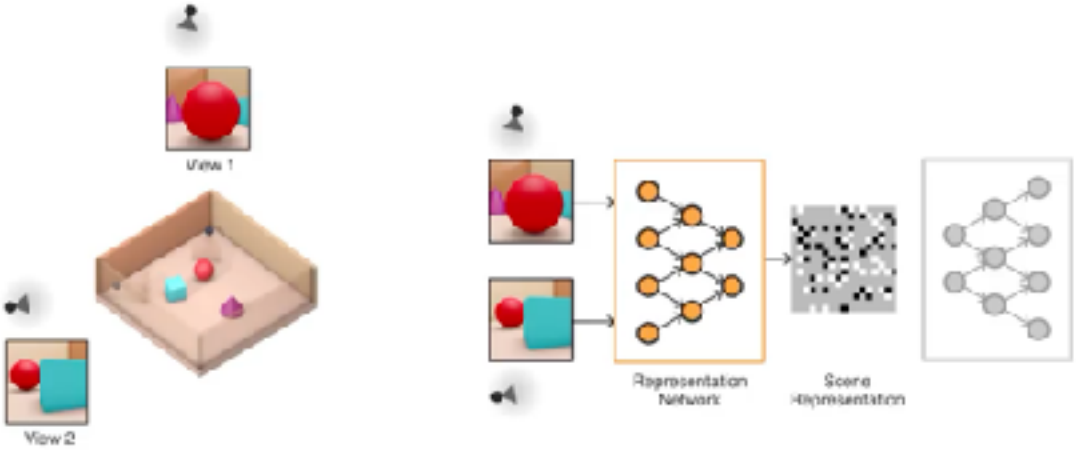
Other developments

Agents that imagine and plan



https://deepmind.com/blog/agents-imagine-and-plan/

Neural scene representation and rendering



https://deepmind.com/blog/neural-scene-representation-and-rendering/

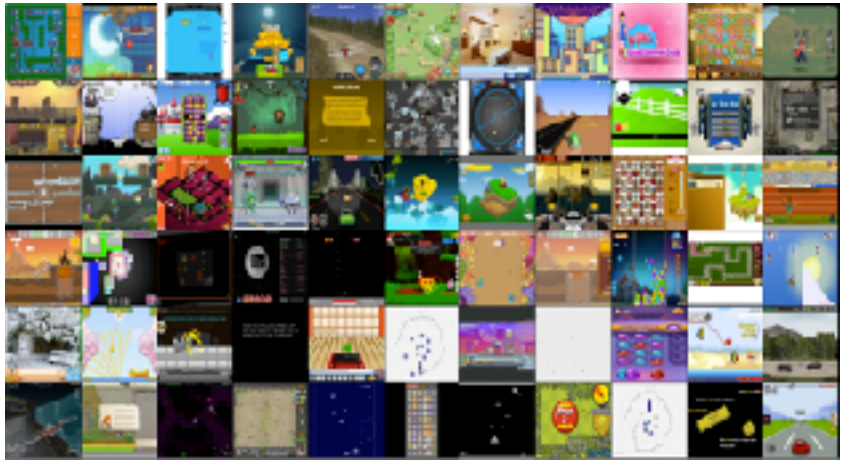
Capture the Flag: the emergence of complex cooperative agents

Agent observation raw pixels

Indoor map overview

https://deepmind.com/blog/capture-the-flag/

OpenAI Universe



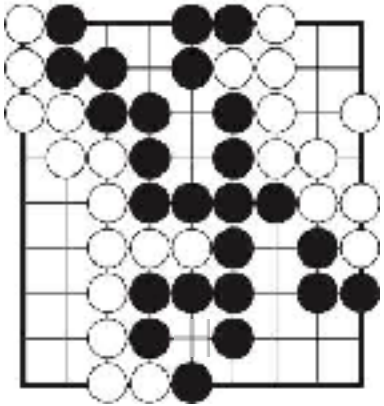
```
import gym
import universe # register Universe environments into Gym

env = gym.make('flashgames.DuskDrive-v0') # any Universe environment ID here
observation_n = env.reset()

while True:
  # agent which presses the Up arrow 60 times per second
  action_n = [[('KeyEvent', 'ArrowUp', True)] for _ in observation_n]
  observation_n, reward_n, done_n, info = env.step(action_n)
  env.render()
```

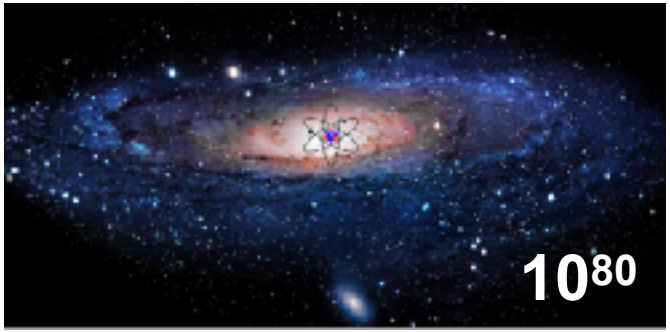
https://openai.com/blog/universe/

Mastering the game of Go



Played on 19 x 19 board. Aim is to capture territory and opponent's stones by surrounding areas.

The number of possible games of Go (~10^{761}) is much larger than that of chess (~ 10^{120})



AlphaGo

Different versions of AlphaGo:

AlphaGo: Defeated professional Go players

- Fan Hui: European Go champion
- Lee Sedol: One of the world's best players

AlphaGo Zero:

Learned to outperform AlphaGo Fan/Lee and play at superhuman performance without any human intervention

(the system is instructed about the rules of the game)

Analytical solutions to game-play: The game of Nim

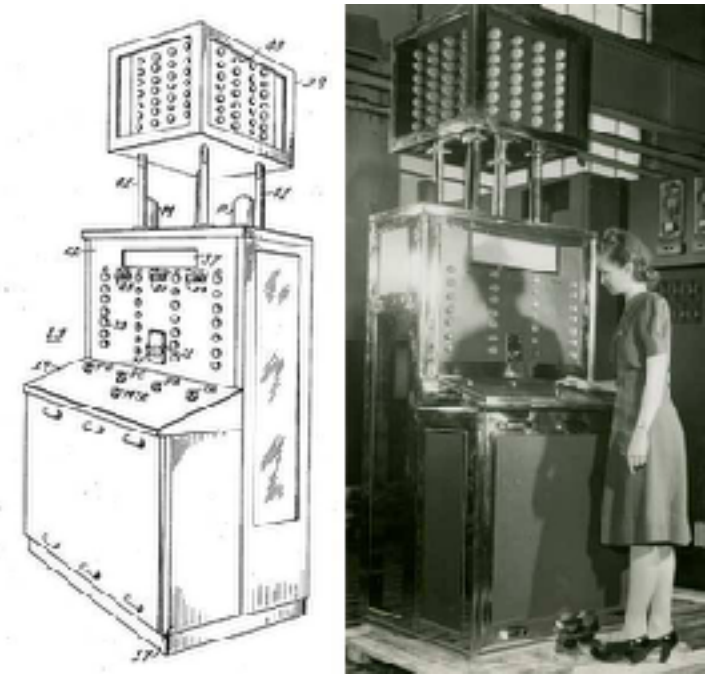
(1) Take at least one token
(2) Take from only one stack
(3) If you take the last token, you win

Solved: outcome (win, lose, or draw) can be predicted from any position, given that both players play perfectly.

➢ Proves whether the first player will win, lose, or draw from the initial position

➢ Provides an algorithm that can produce perfect play (moves) from any position, even if mistakes have already been made on one or both sides.

The game of Nim



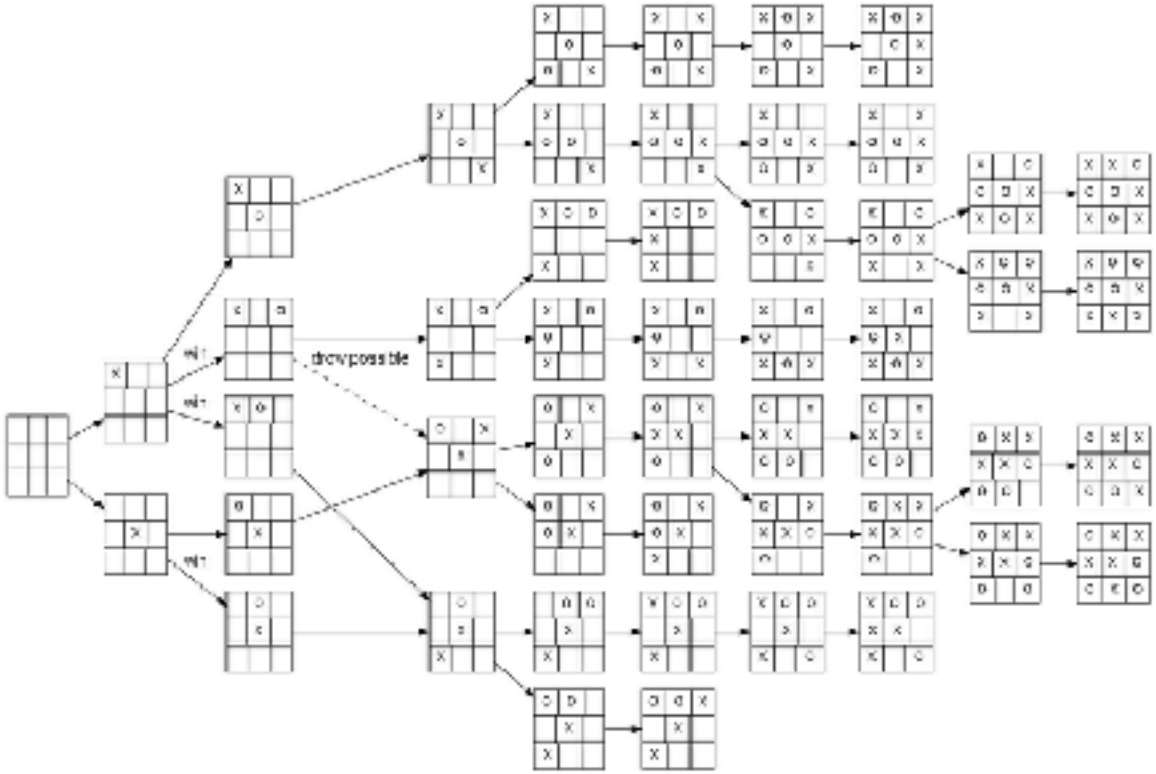
Nimatron: One of the first computer games (1940)

“NIM Playing Machine from Tinkertoy”,1978-79 http://harveycohen.net/nim/

Tree search

In general, to solve games, we need to perform tree search

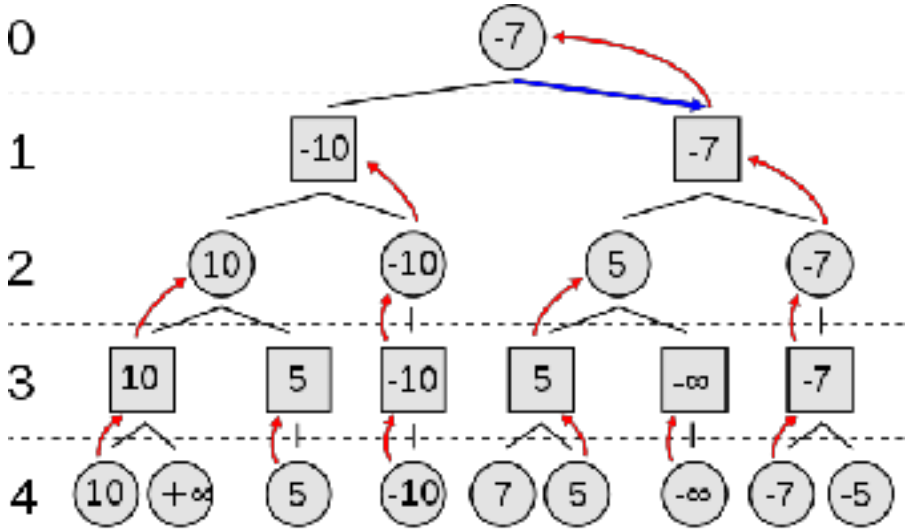
game tree for tic-tac-toe



Minimax

Knowing the complete game tree is useful for a game playing AI. It allows the program to pick the best possible move at a given game state.

Can be done with the minimax algorithm: At each game turn, the AI figures out which move would minimize the maximum loss



Deep Blue strategy

Minimax traverses the whole game tree down to nodes representing end-of-game states. This does not work for complex games!

Deep Blue searched the game tree as far as possible, usually to a depth of six moves or more. It would then use a hand-designed evaluation function to evaluate the quality of the nodes at that depth.

Evaluation function replaces the subtree below that node with a single value summarizing this subtree.

Then, Deep Blue would proceed similarly to the minimax algorithm: The move that leads to the least bad worst-case scenario at this maximum depth is chosen.

Deep Blue strategy

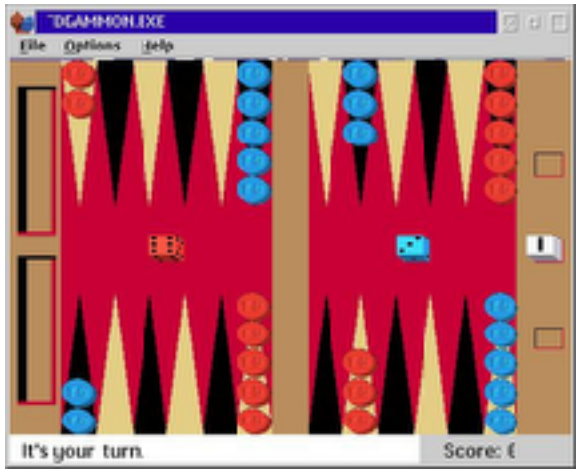
Go cannot be tackled effectively with the Deep Blue approach:

- Go has a wider branching factor (more possible moves at each state) than chess.
- Games tend to be longer.
- Hence, it is more difficult to search the game tree to a sufficient depth.
- It is more difficult to design evaluation functions for Go than for chess.

Learning from self-play

AlphaGo learns by playing games against itself. It combines tree search with neural networks.

Rich tradition in AI:



TD-Gammon: backgammon using neural networks and playing against itself

Monte Carlo Tree Search (MCTS)

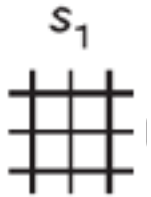
AlphaGo uses Monte Carlo Tree Search (MCTS):

- Run many game simulations starting at the current game state and stopping when the game is won by one of the two players
- At first, the simulations are completely random
- At each simulation, some values are stored, such as how often each node has been visited, and how often this has led to a win.
- These numbers guide the later simulations in selecting actions
- The more simulations are executed, the more accurate these numbers become at selecting winning moves.
- As the number of simulations grows MCTS converges to optimal play

Browne et al. (2012). A Survey of Monte Carlo Tree Search Methods, 4(1), 1–49.
https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/

Monte Carlo Tree Search (MCTS)

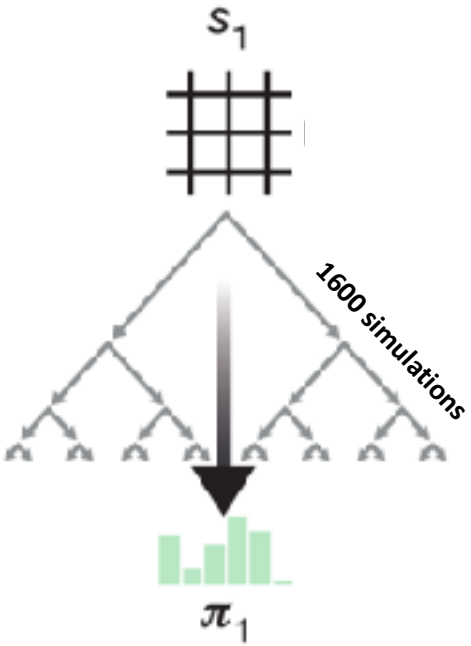
AlphaGo uses MCTS to play games against itself:



Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Monte Carlo Tree Search (MCTS)

AlphaGo uses MCTS to play games against itself:

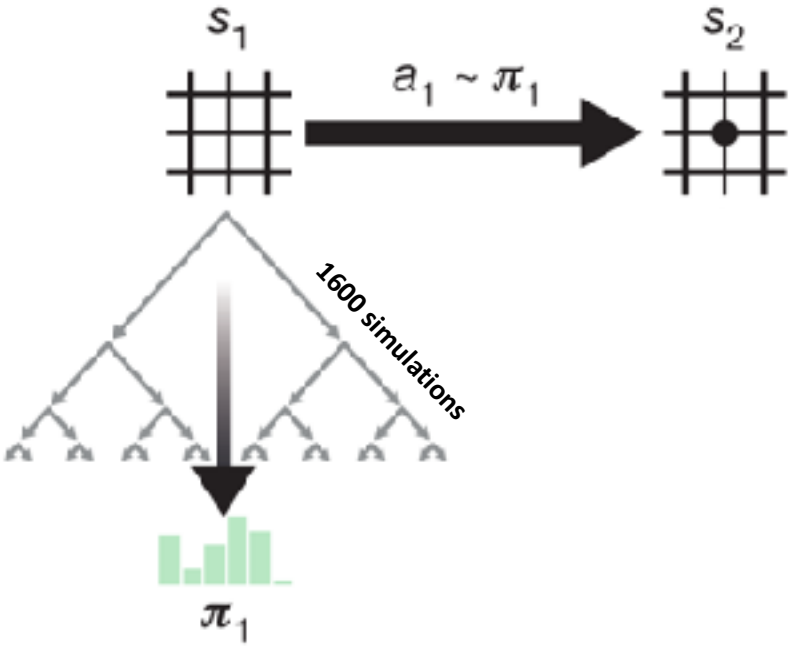


Search probabilities

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Monte Carlo Tree Search (MCTS)

AlphaGo uses MCTS to play games against itself:

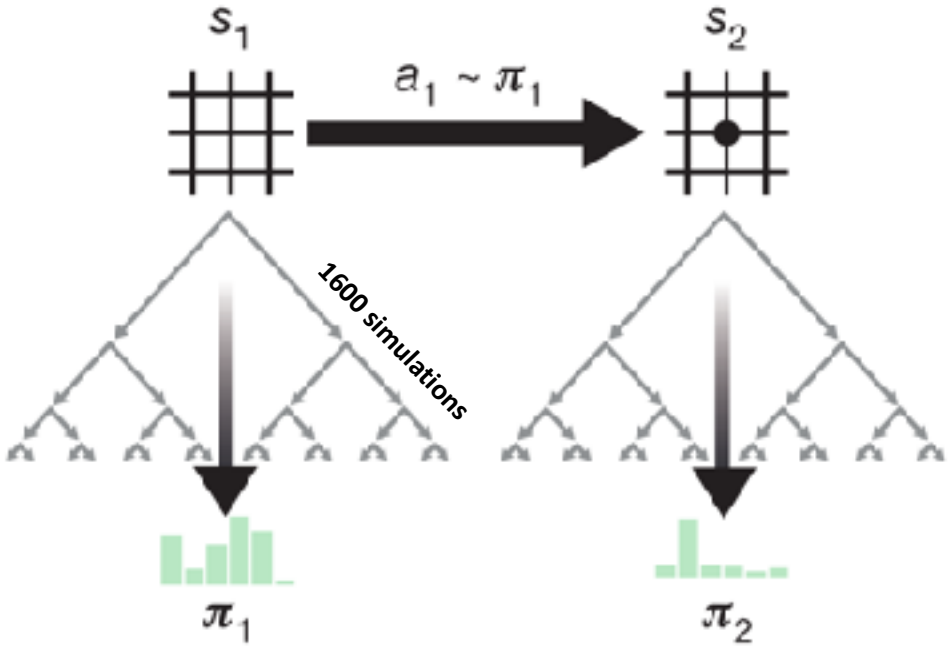


Search probabilities

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Monte Carlo Tree Search (MCTS)

AlphaGo uses MCTS to play games against itself:

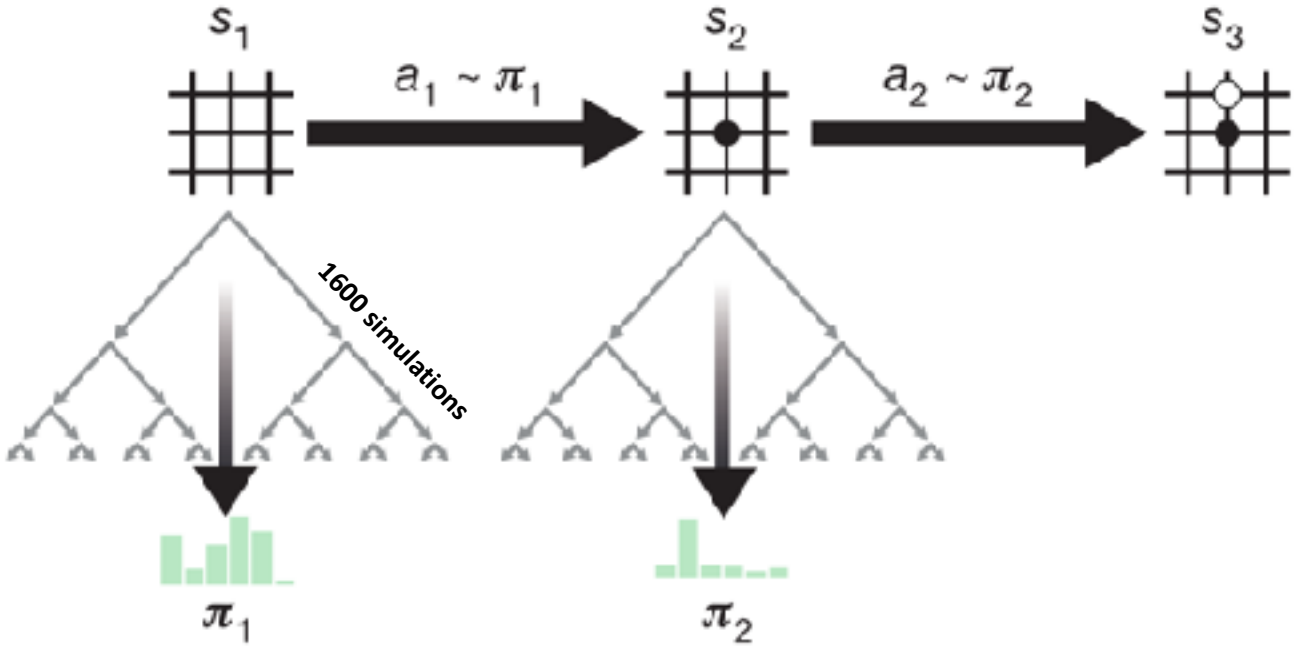


Search probabilities

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Monte Carlo Tree Search (MCTS)

AlphaGo uses MCTS to play games against itself:

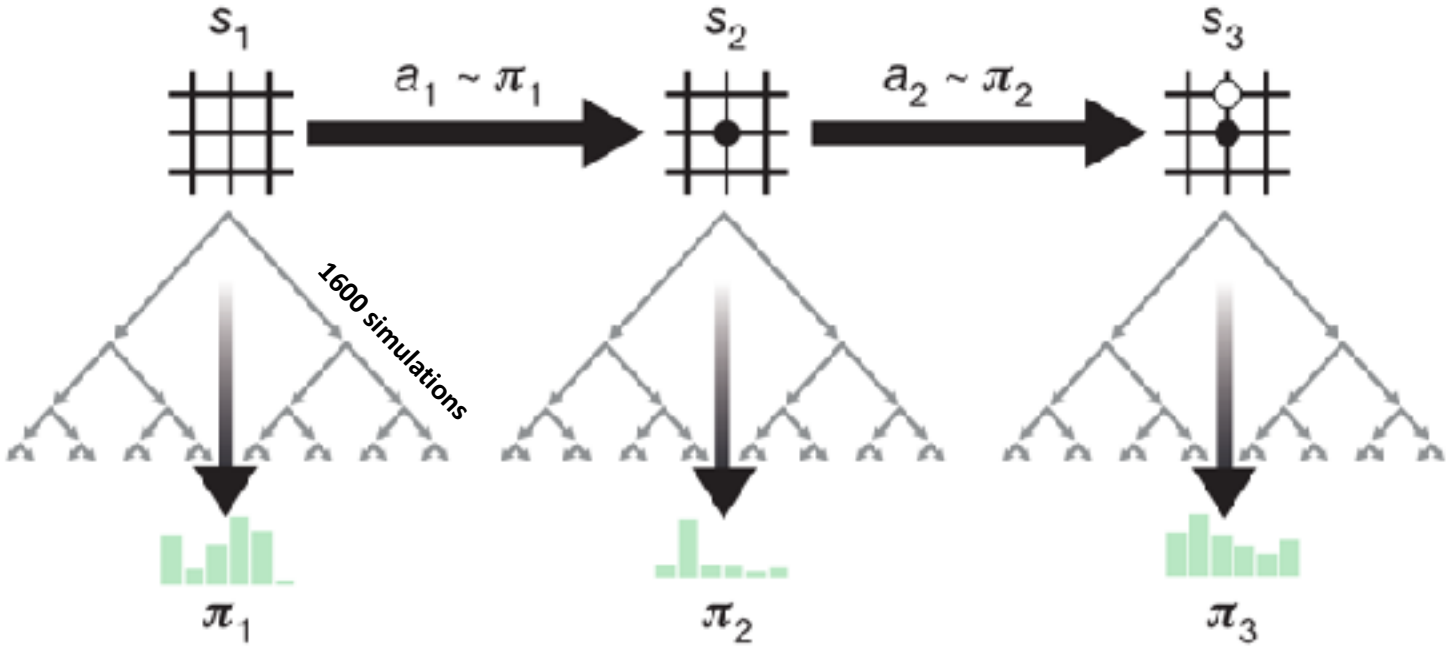


Search probabilities

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Monte Carlo Tree Search (MCTS)

AlphaGo uses MCTS to play games against itself:

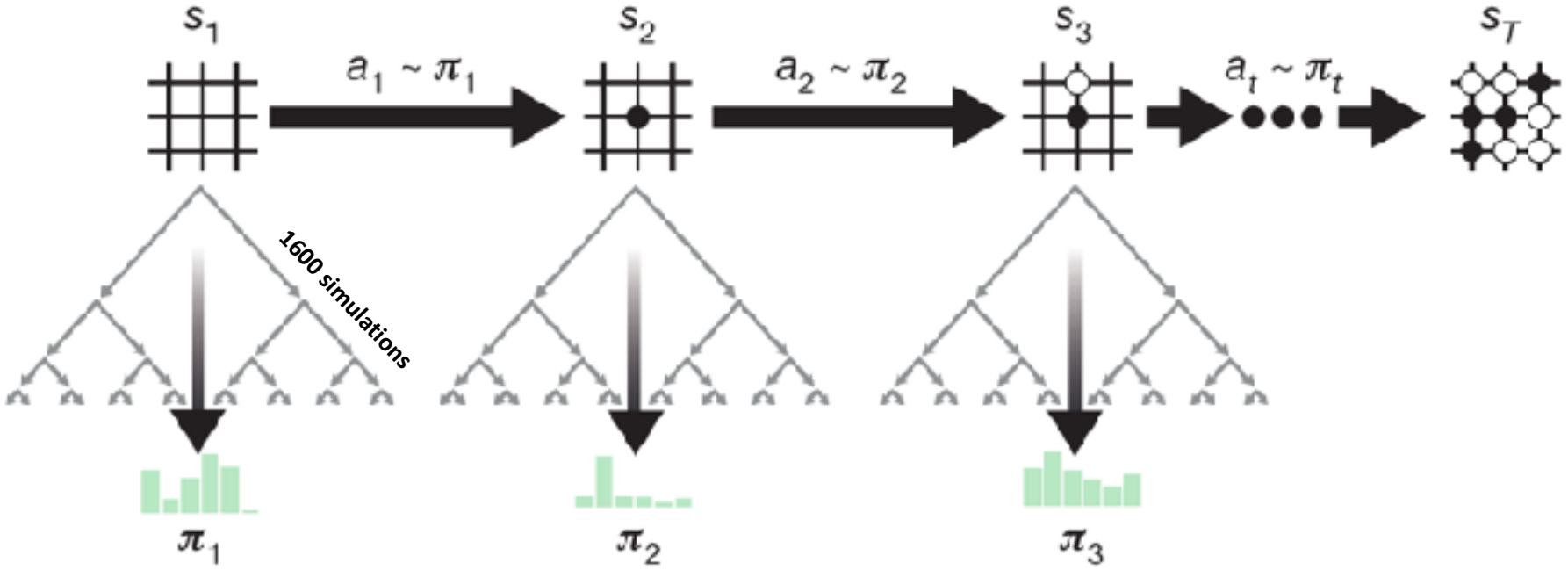


Search probabilities

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Monte Carlo Tree Search (MCTS)

AlphaGo uses MCTS to play games against itself:

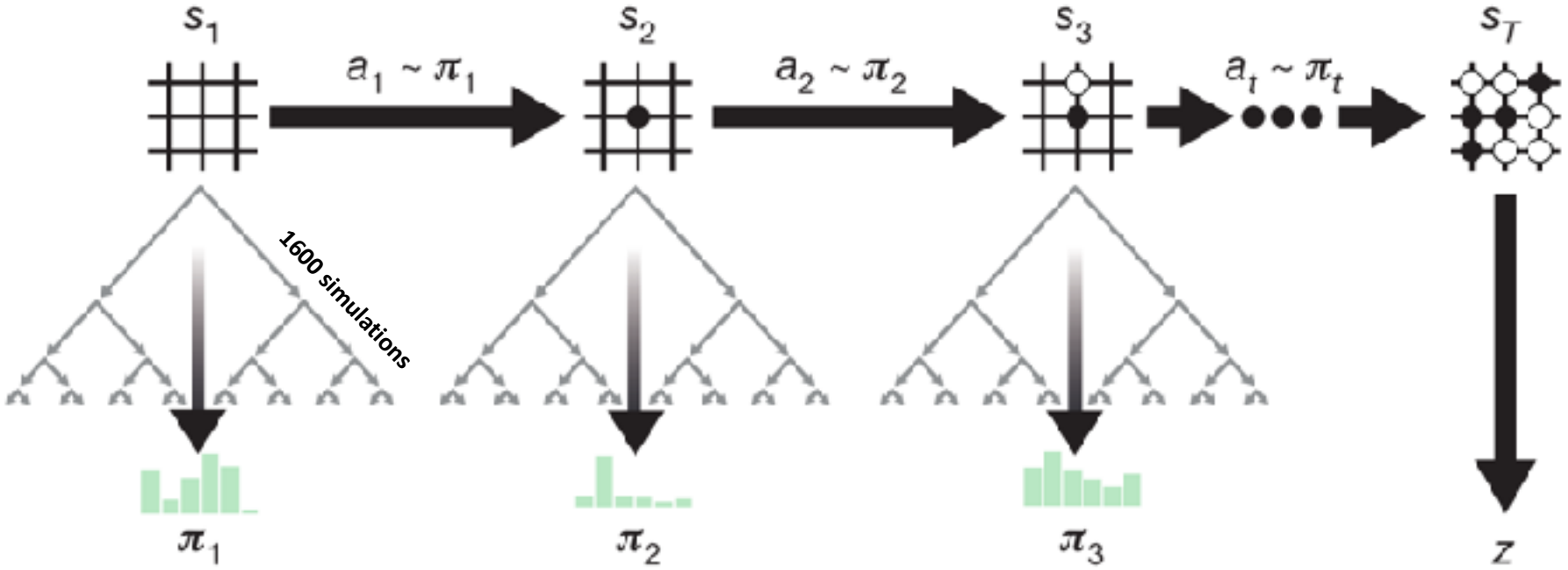


Search probabilities

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Monte Carlo Tree Search (MCTS)

AlphaGo uses MCTS to play games against itself:



Search probabilities

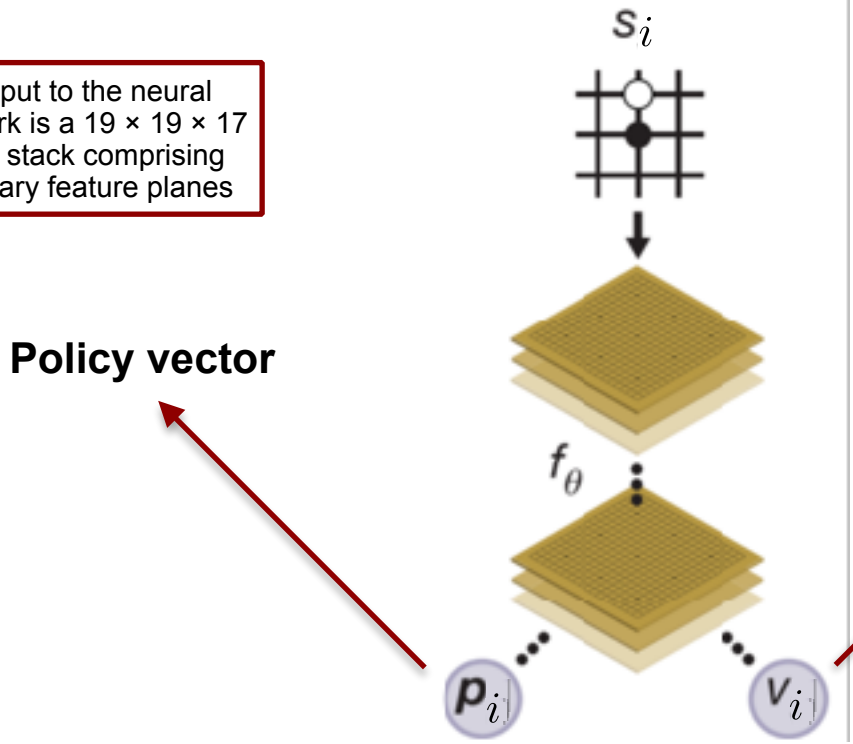
Black or White wins

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Deep neural network

MCTS uses a deep neural network to guide traversal through the search tree

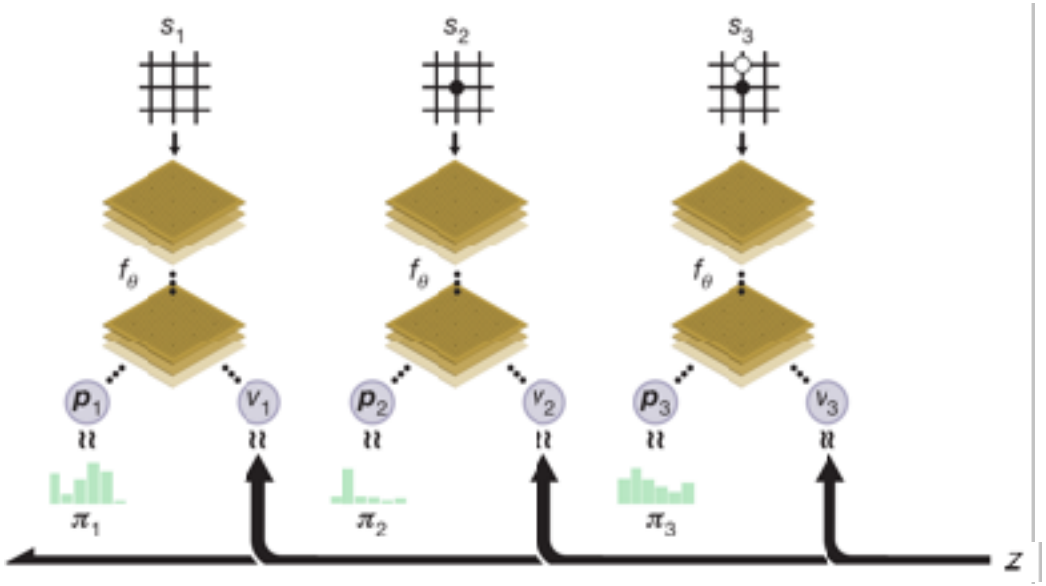
The input to the neural network is a 19 × 19 × 17 image stack comprising 17 binary feature planes



The overall network depth, in the 20- or 40-block network, is 39 or 79 parameterized layers, respectively, for the residual tower, plus an additional 2 layers for the policy head and 3 layers for the value head.

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Neural network training

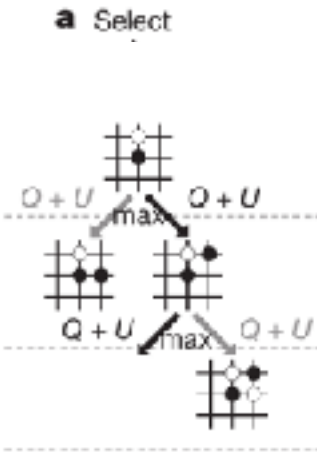


Neural network parameters θ are updated to maximize the similarity between **p** and **π** and to minimize the difference between v and z:

$$\ell = (z - v)^2 - \boldsymbol{\pi}^T \log \mathbf{p} + c||\boldsymbol{\theta}||^2$$

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

How MCTS selects an action



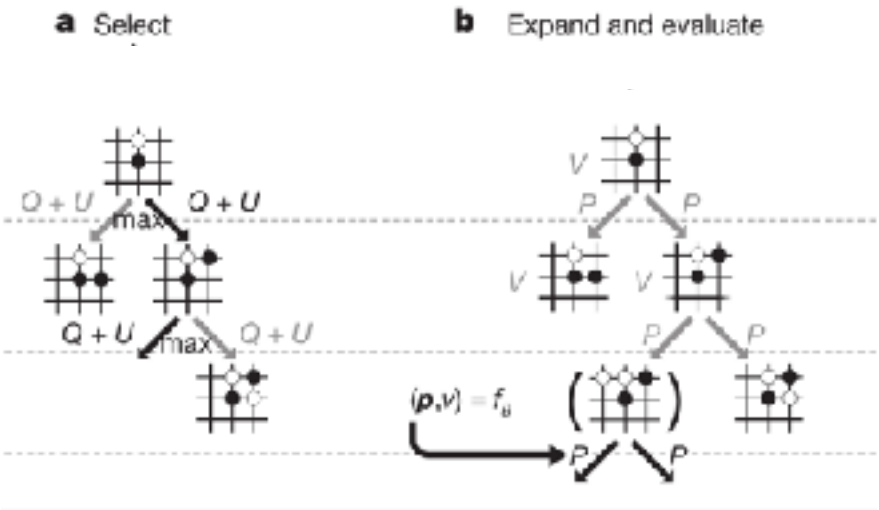
Each edge (s, a) in the search tree stores a prior probability P(s, a), a visit count N(s, a), and an action value Q(s, a)

Each simulation starts from a root state and iteratively selects moves that maximize an upper confidence bound Q + U where

$$U(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

How MCTS selects an action

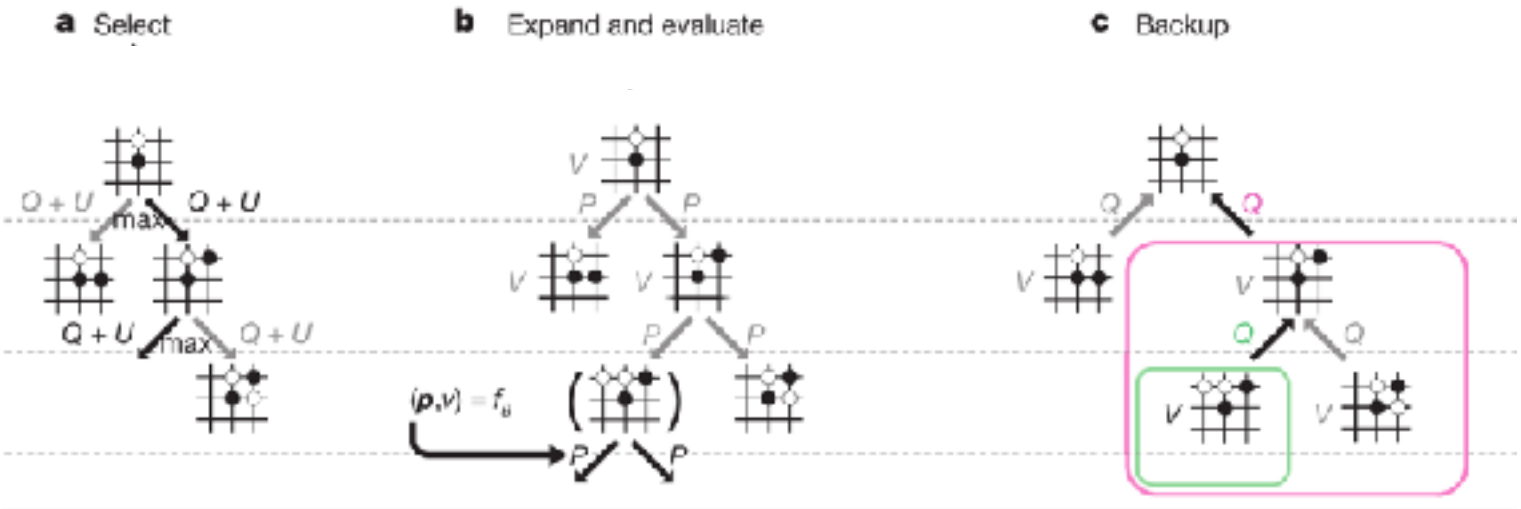


When a new edge (s,a) is selected, N is increased and the neural network is invoked to compute **p** and v for that edge

The prior P is set to be equal to **p**

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

How MCTS selects an action

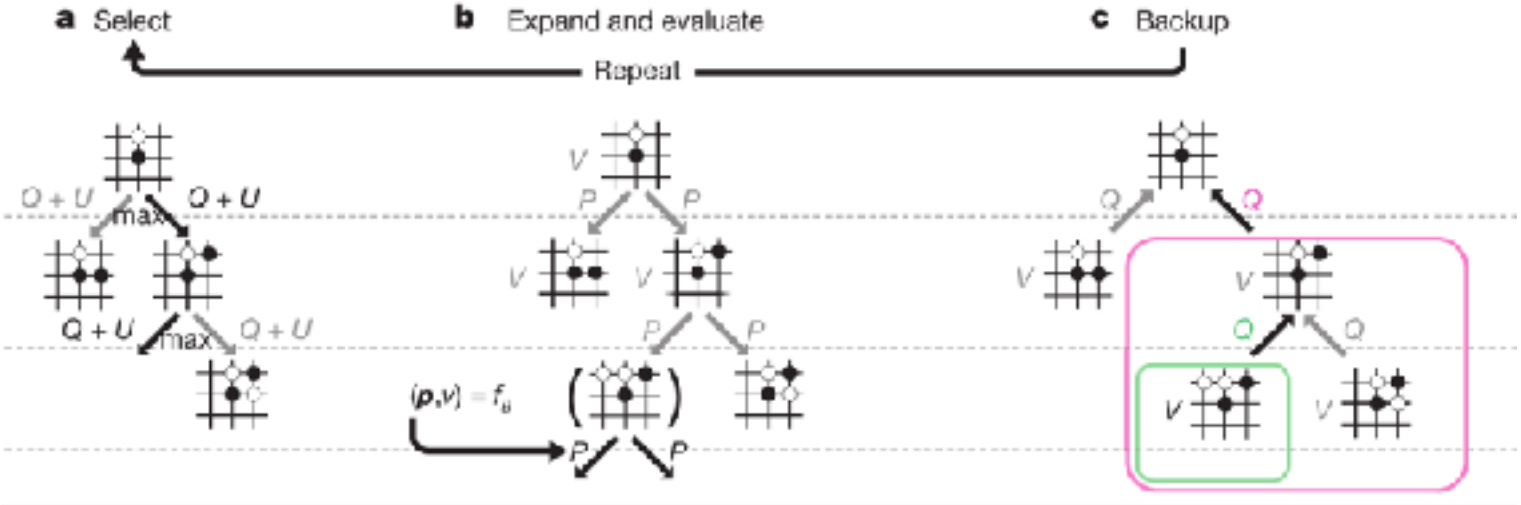


Backup step recomputes Q based on the averages of values V(s') of the final states s' that are eventually reached over all simulations:

$$Q(s,a) = 1/N(s,a) \sum_{s'|s,a\to s'} V(s')$$

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

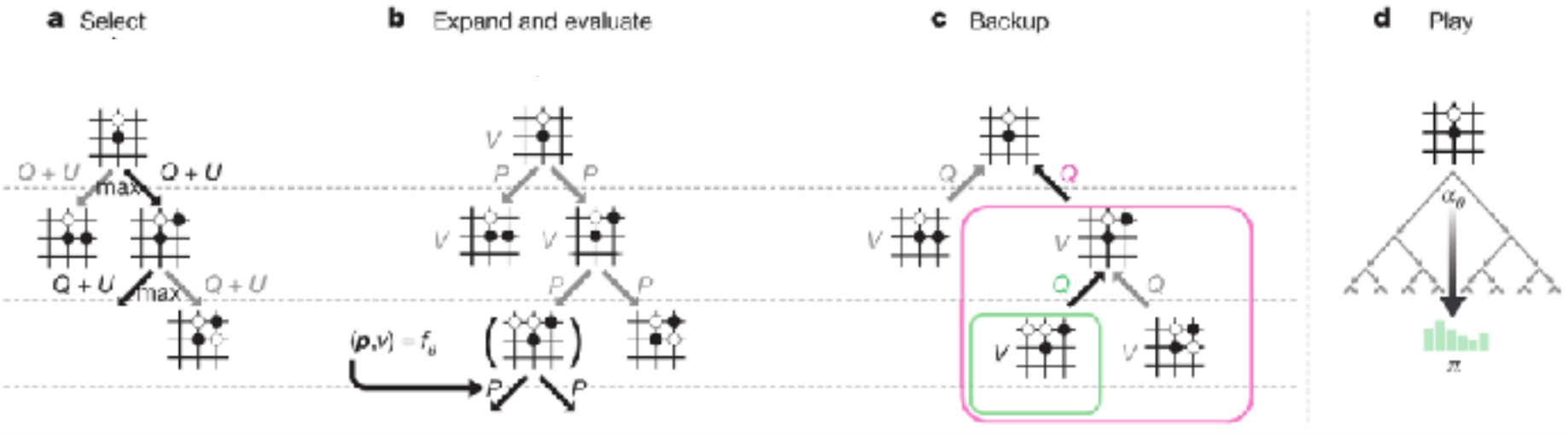
How MCTS selects an action



Simulations from a root state are repeated many times (1600 simulations per move)

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

How MCTS selects an action



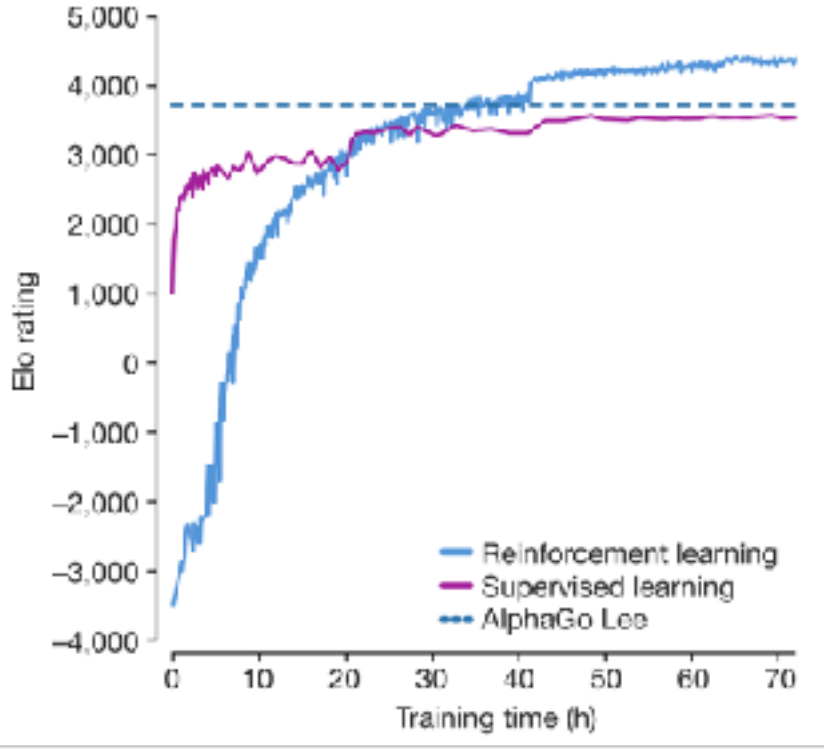
Search probabilities **π** are computed based on the number of times an action has been taken during simulation:

$$\pi_a \propto N(s,a)^{1/\tau}$$

An action is sampled from this vector where tau is a temperature parameter. This is the next move.

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Results



AlphaGo Zero outperformed AlphaGo Lee after just 36 h

AlphaGo Zero used a single machine with 4 tensor processing units (TPUs), whereas AlphaGo Lee was distributed over many machines and used 48 TPUs.

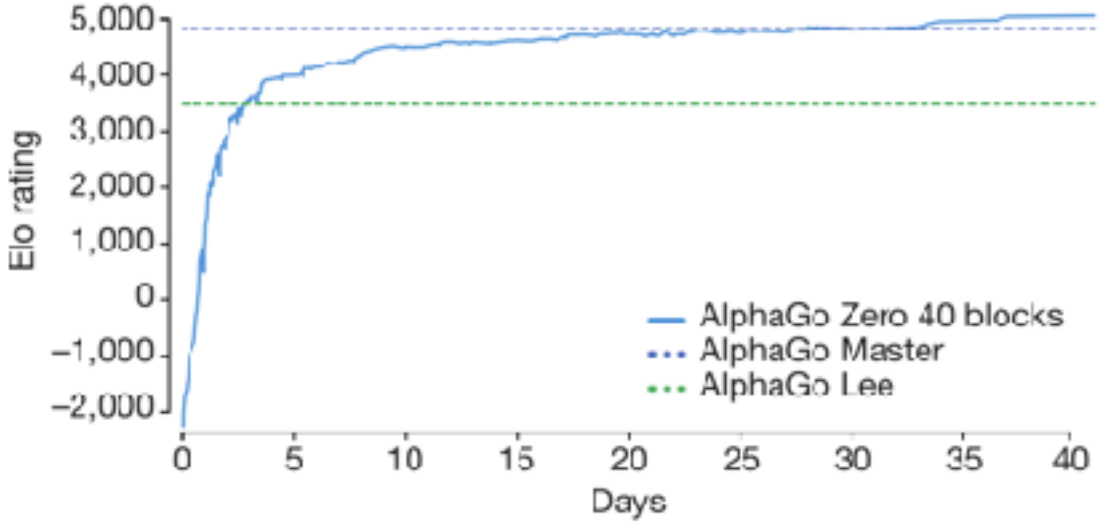
Defeated system trained on human games after 24 h

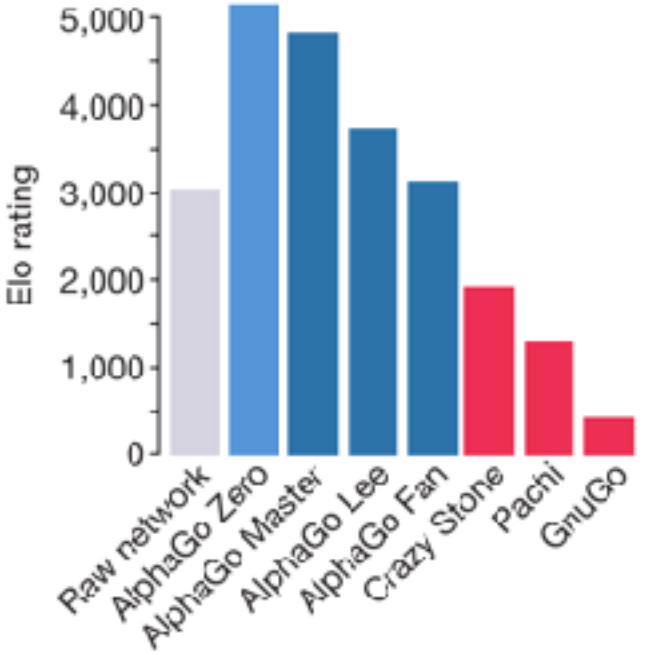
No handcrafted features used

Much more elegant than AlphaGo Lee (used several training stages and multiple neural networks)

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

Results using larger model trained for 40 days





Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Nature; 2017;550(7676):354–9.

AlphaGo Zero

AlphaGo Zero is a major leap forward in AI

AlphaZero system recently solved chess and shogi

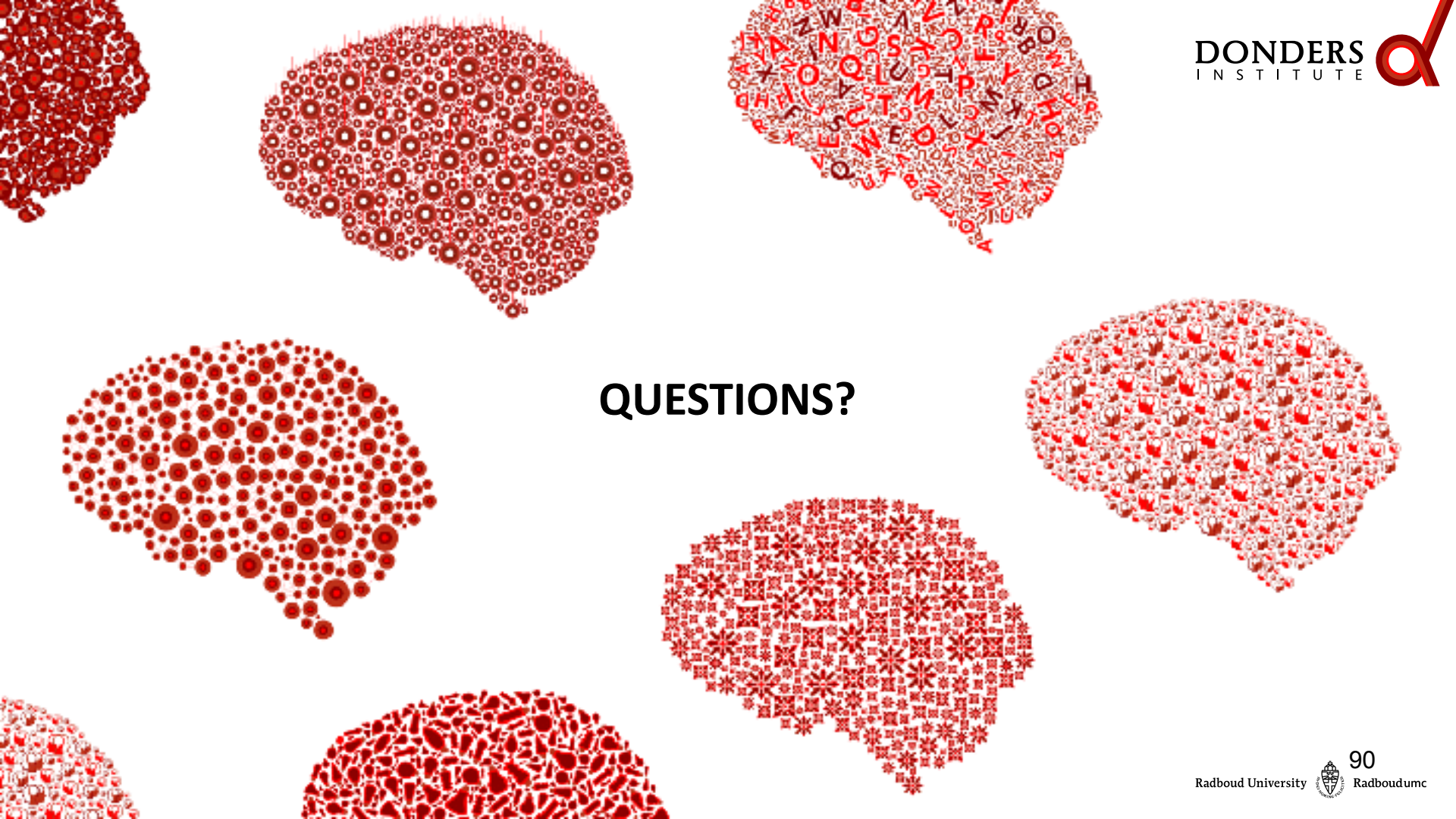
General algorithm for solving large range of problems

Example: protein folding prediction for drug discovery

Pitched as a step in achieving **artificial general intelligence (AGI)**

Sources

- http://cs234.stanford.edu
- http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- https://www.youtube.com/playlist?list=PLqYmG7hTraZDNJre23vqCGIVpfZ_K2RZs


QUESTIONS?