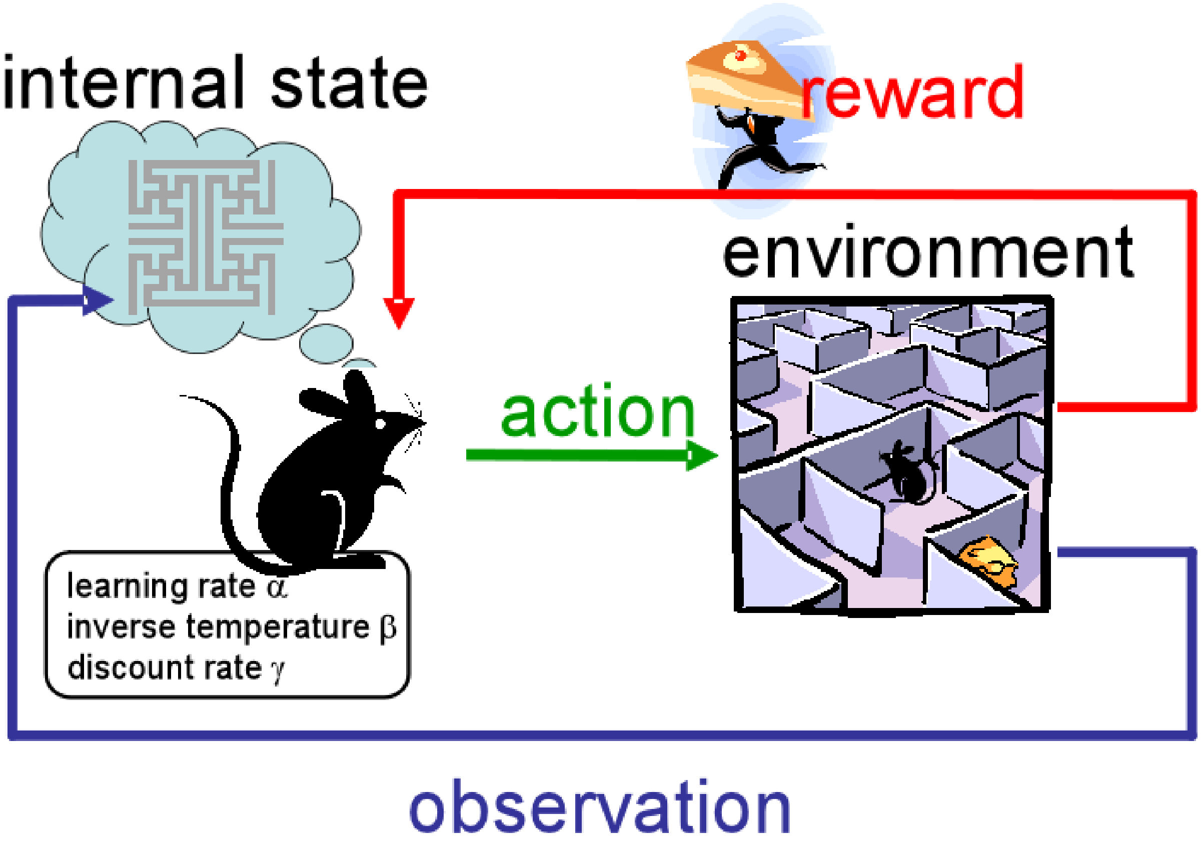
Reinforcement Learning with Policy Gradients



Cyber Rodent Project

Some slides from:

David Silver, Radford Neal

ECE324, Winter 2022

Michael Guerzhoy

Reinforcement Learning

- Supervised learning:
  - The training set consists of inputs and outputs. We try to build a function that predicts the outputs from the inputs. The cost function is a *supervision signal* that tells us how well we are doing
- Unsupervised Learning
  - The training set consists of data (just the inputs). We try to build a function that models the inputs. There is no supervision signal
- Reinforcement Learning
  - The *agent* performs *actions* that change the *state* and receives *rewards* that depend on the state
  - Trade-off between exploitation (go to states you already discovered give you high reward) and exploration (try going to states that give even higher rewards)

Reinforcement Learning

- The world is going through a sequence of states $s_1, s_2, s_3, \dots, s_n$ and times $t_1, t_2, \dots, t_n$
- At each time t_i, the agent performs **action** a_i, moves to **state** s_{i+1} (depending on the action taken) and receives **reward** r_i (the reward could be 0)
- Goal: maximize the total reward over time
    - Total reward: $r_1 + r_2 + \cdots + r_n$
    - Total reward with discounting, so that rewards for away in the future count for less: $r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{n-1} r_n$
        - Getting a reward now is better than getting the same reward later on

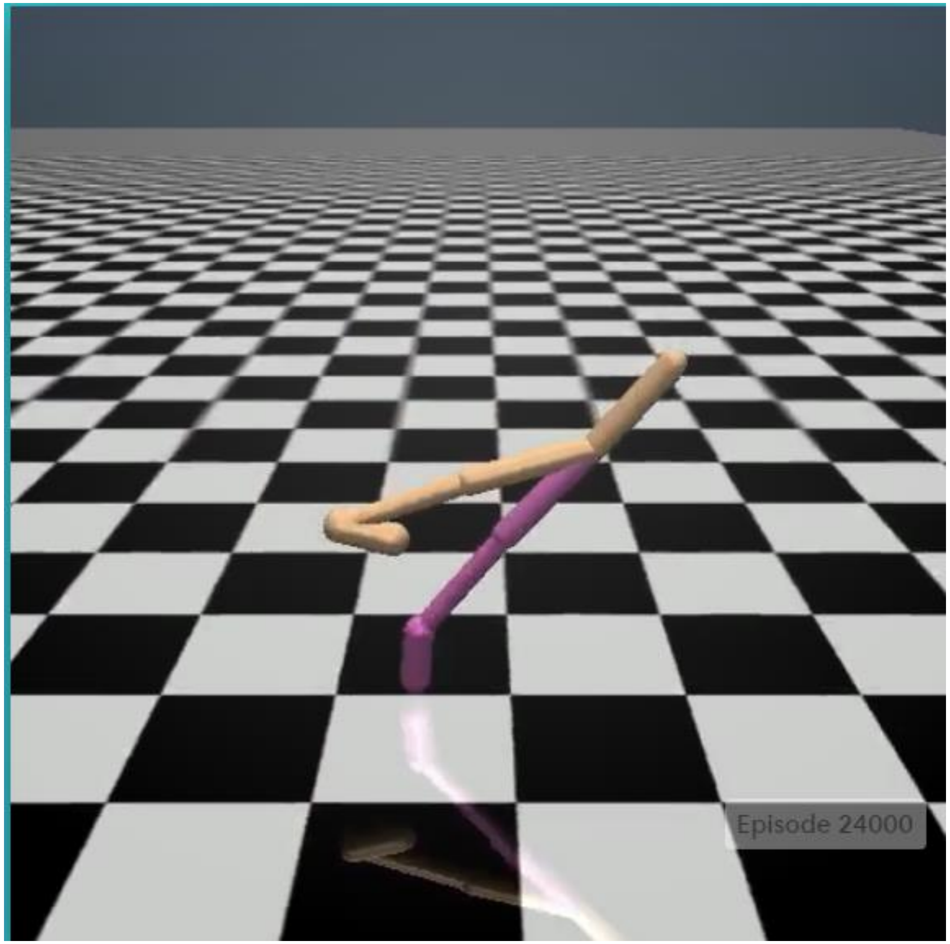
Reinforcement Learning: Go

AlphaGo defeats Lee Sedol (2016)

Reinforcement Learning: Go

- **State**: the position on the board
- **Reward**: 0 if the game hasn’t ended, 1 if the agent wins, -1 if the opponent wins
- **Action**: make a legal Go move (place a stone on a free point)
- Goal: make a function that, given the state (position on the board), finds an optimal move
  - Note: we could have intermediate goals as well, like learning a function that evaluates every state
- *Exploitation vs. Exploration*
  - Make moves the function already thinks will lead to a good outcome *vs*
  - Try making novel moves and see if you discover a way to adjust the function to get even better outcomes

Reinforcement Learning: Walking



https://gym.openai.com/envs/Walker2d-v1

Reinforcement Learning: Walking

- **State**: the positions of all the joints
- **Reward**: if we haven’t walked to the destination yet, 0. If we reached the destination, 1
- **Action**: apply a force to a joint in a particular direction
- Goal: learn a function that applies a particular force to a particular joint at every time-step t so that the walker reaches the destination

Policy Learning

- A policy function π takes in the current **state** s, and outputs the move (**action**) the agent should take
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = P(A_t = a|S_t = s)$
    - Must have for things like playing poker
    - But also good for exploration in general!
- Just like for other functions we approximate, we can parametrize π using a parameter vector θ
  - Initialize θ randomly
  - Follow the policy π_θ, and adjust θ based on the rewards we receive

Softmax Policy (discrete actions)

- Compute features $\phi(a, s)$ for each action-state tuple
  - Some kind of representation that makes sense
  - Could be something very complicated
    - E.g. something computed using a deep neural network
  - In general, we can think of the features as the last layer of the neural network, before it's passed into the softmax
- $\pi_\theta(s, a) \propto \exp(\phi(s, a)^T \theta)$

Gaussian Policy (continuous actions)

- For continuous actions, it makes sense to use a Gaussian distribution for the actions, centered around $\phi(s)^T\theta$
- $a \sim N(\phi(s)^T\theta, \sigma^2)$

How good is policy π_θ?

- $V^{\pi_\theta}(s)$ is the (expected) total reward if we start from state s
  - Start from state s at time 0
  - Follow policy π_θ, and compute $r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$
- $q^{\pi_\theta}(a|s)$ is the total expected reward for performing action a in state s, and then following π_θ
  - $V^{\pi_\theta}(s) = \sum_a \pi_\theta\,(a|s)\,q^{\pi_\theta}(a|s)$

How good is policy π_θ?

- $d^{\pi_\theta}(s)$ is the probability of the agent being in state s if we follow policy π_θ for a long time
  - Not easily computed at all!
  - But we can simply follow policy π_θ for a long time and record how often we find ourselves in each state
  - For continuous states, do some approximation of that
- $J_{avV}(\theta) = \sum_s \, d^{\pi_\theta}(s) V^{\pi_\theta}(s)$
  - $V^{\pi_\theta}(s)$ is the (expected) total reward if we start from state s
  - We want states that lead to high rewards to have high probability
  - We want to take actions that lead to high rewards
- Larger $J_{avV}(\theta)$ means better θ

Policy Gradient

- $J_{avV}(\theta) = \sum_s \ d^{\pi_\theta}\,(s) V^{\pi_\theta}(s)$

$$= \sum_s d^{\pi_\theta}\,(s) \sum_a \pi_\theta(a|s) q^{\pi_\theta}(a|s)$$

- $\nabla J = \begin{pmatrix} \partial J/\partial\theta_1 \\ \cdots \\ \ldots \\ \partial J/\partial\theta_n \end{pmatrix}$
- Idea: $\theta \leftarrow \theta + \alpha\nabla J(\theta)$
  - Increase J

Policy Gradient: Finite Differences

- For each k in $1..n$

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta+u_k)-J(\theta)}{\epsilon}$$

(u_k is all 0’s except the k-th coordinate is ϵ)

- Approximate $J(\theta)$ by following policy π_θ for a while and keeping track of the rewards you get!
- Has actually been used to make physical robots that walk
  - The policy function had about 12 parameters
  - Vary each parameter in turn, have the robot run, measure how fast it ran, and compute the gradient based on that

Policy Gradient Theorem

- $J_{avV}(\theta) = \sum_s \ d^{\pi_\theta}\,(s)V^{\pi_\theta}\,(s)$, so
- $J_{avV}(\theta) = \sum_s \ d^{\pi_\theta}\,(s) \sum_a \pi_\theta\,(a|s)\, q^{\pi_\theta}(a|s)$
  - $\pi_\theta\,(a|s)$ is the probability of taking action a starting from state s, following policy $\pi_\theta\,(a|s)$
  - $q^{\pi_\theta}(\,a|s)$ is the total expected reward for performing action a in state s, and then following π_θ

- $\nabla_\theta J_{avV}(\theta) = \sum_s \ d^{\pi_\theta}\,(s) \sum_a q^{\pi_\theta}(a|s)\nabla_\theta \pi_\theta\,(a|s)$
  - Not obvious! We are differentiating an expression involving both d^{π_θ} and V^{π_θ}

Policy Gradient Theorem

- $\nabla_\theta J_{avV}(\theta) = \sum_s \ d^{\pi_\theta}(s) \sum_a \ q^{\pi_\theta}(a|s) \nabla_\theta \pi_\theta \ (a|s)$
  - Weighted sum over $\sum_a \ q^{\pi_\theta}(s,a) \nabla_\theta \pi_\theta \ (a|s)$
  - If it looks like we should take action *a* in state *s* (i.e., $q^{\pi_\theta}(s,a)$ is high):
    - Care more about $\nabla_\theta \pi_\theta \ (a|s)$, which tells us how to change θ to make it more likely that we take action *a* in state *s*
  - Take the weighted average over the gradients for all states, weighing the states that we are more likely to visit more

Policy Gradient: Gaussian Policy

- $a \sim N(\phi(s)^T\theta, \sigma^2)$
- $\nabla_\theta \log \pi_\theta(a|s) = \nabla_\theta \log \exp\left(-\frac{\left(a-\phi(s)^T\theta\right)^2}{2\sigma^2}\right) =$

$$\nabla_\theta - \frac{(a - \phi(s)^T\theta)^2}{2\sigma^2} =$$

$$\frac{(a - \phi(s)^T\theta)\phi(s)}{\sigma^2}$$

- (How to make it more likely that we take action a in state s?)
- (Aside: $\nabla \log (f) = (\nabla f)/f$)

Expectation trick

- At time t, starting from state S_t:
- $\nabla_\theta J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a q^{\pi_\theta}(a|s) \nabla_\theta \pi_\theta(a|s) =$

$$E_{\pi_\theta}\left[\gamma^t \sum_a q^{\pi_\theta}(a|S_t) \nabla_\theta \pi_\theta(a|S_t)\right]$$

- (Just follow policy π_θ, and in the long term, will encounter states in proportions d^{π_θ})

Using expectations

- $E_{\pi_\theta}[f(A)] = \sum_a \pi_\theta(a) f(a)$
- $E_{\pi_\theta}[f(A)] = E_{\pi_\theta}[E_{\pi_\theta}[f(A)] = E_{\pi_\theta}\left[\sum_a \pi_\theta(a) f(a)\right]$

Expectation trick, again

- $\nabla_\theta J_{avV}(\theta) = E_{\pi_\theta}\left[\gamma^t \sum_a q^{\pi_\theta}(a|S_t) \nabla_\theta \pi_\theta(a|S_t)\right]$

$$= E_{\pi_\theta}\left[\gamma^t \sum_a \pi_\theta(a|S_t) q^{\pi_\theta}(a|S_t) \frac{\nabla_\theta \pi_\theta(a|S_t)}{\pi_\theta(a|S_t)}\right]$$

  - Multiply and divide again by $\pi_\theta(a|S_t)$

- Now, replace the sum over actions a by a single action A_t that we actually take – can do that inside an expectation!

$$= E_{\pi_\theta}\left[\gamma^t q^{\pi_\theta}(A_t|S_t) \frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}\right]$$

Expectation trick, again

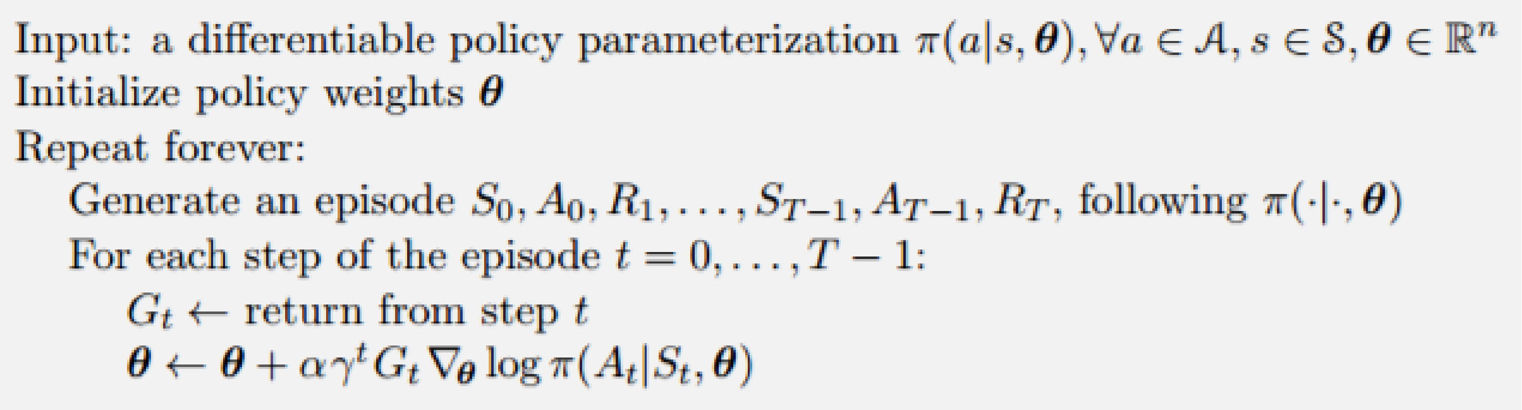
- $\nabla_\theta J_{avV}(\theta) = E_{\pi_\theta}[\gamma^t q^{\pi_\theta}(A_t|S_t)\ \frac{\nabla_\theta \pi_\theta\,(A_t|S_t)}{\pi_\theta\,(A_t|S_t)}]$
- Now, replace $q^{\pi_\theta}(A_t|S_t)$ by the actual total reward we get by following policy π_θ, G_t -- again, can do that inside the expectation
- $\nabla_\theta J_{avV}(\theta) = E_{\pi_\theta}\left[\gamma^t G_t \frac{\nabla_\theta \pi_\theta\,(A_t|S_t)}{\pi_\theta\,(A_t|S_t)}\right] = E_{\pi_\theta}[\gamma^t G_t \nabla_\theta \log \pi_\theta(A_t|S_t)]$
- Note: $E[G_0] = V^{\pi_\theta}(S_0)$

REINFORCE: Intro

- $\nabla_\theta J_{avV}(\theta) = E_{\pi_\theta}\left[\gamma^t G_t \frac{\nabla_\theta \pi_\theta\,(A_t|S_t)}{\pi_\theta\,(A_t|S_t)}\right] = E_{\pi_\theta}[\gamma^t G_t \nabla_\theta \log \pi_\theta(A_t|S_t)]$

REINFORCE

- $\nabla_\theta J_{avV}(\theta) = E_{\pi_\theta}\left[\gamma^t G_t \frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}\right]$
- Estimate the expectation by simply following policy π_θ and recording the rewards you get!



```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G_t \leftarrow$ return from step $t$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$
```

- Note: G_t is the total (discounted) reward starting from time t

REINFORCE

- $\nabla_\theta J_{avV}(\theta) = E_{\pi_\theta}\left[\gamma^t G_t \frac{\nabla_\theta \pi_\theta\,(A_t|S_t)}{\pi_\theta\,(A_t|S_t)}\right]$
- Overall idea: follow the policy, if it seems that starting from time t we’re getting a big reward, make state A_t more likely

Case Study: AlphaGO

- Go is a remarkably difficult game
  - Lots of possible moves
  - At least $10^{(10^{48})}$ possible games
  - Very hard to tell if a position is good or bad

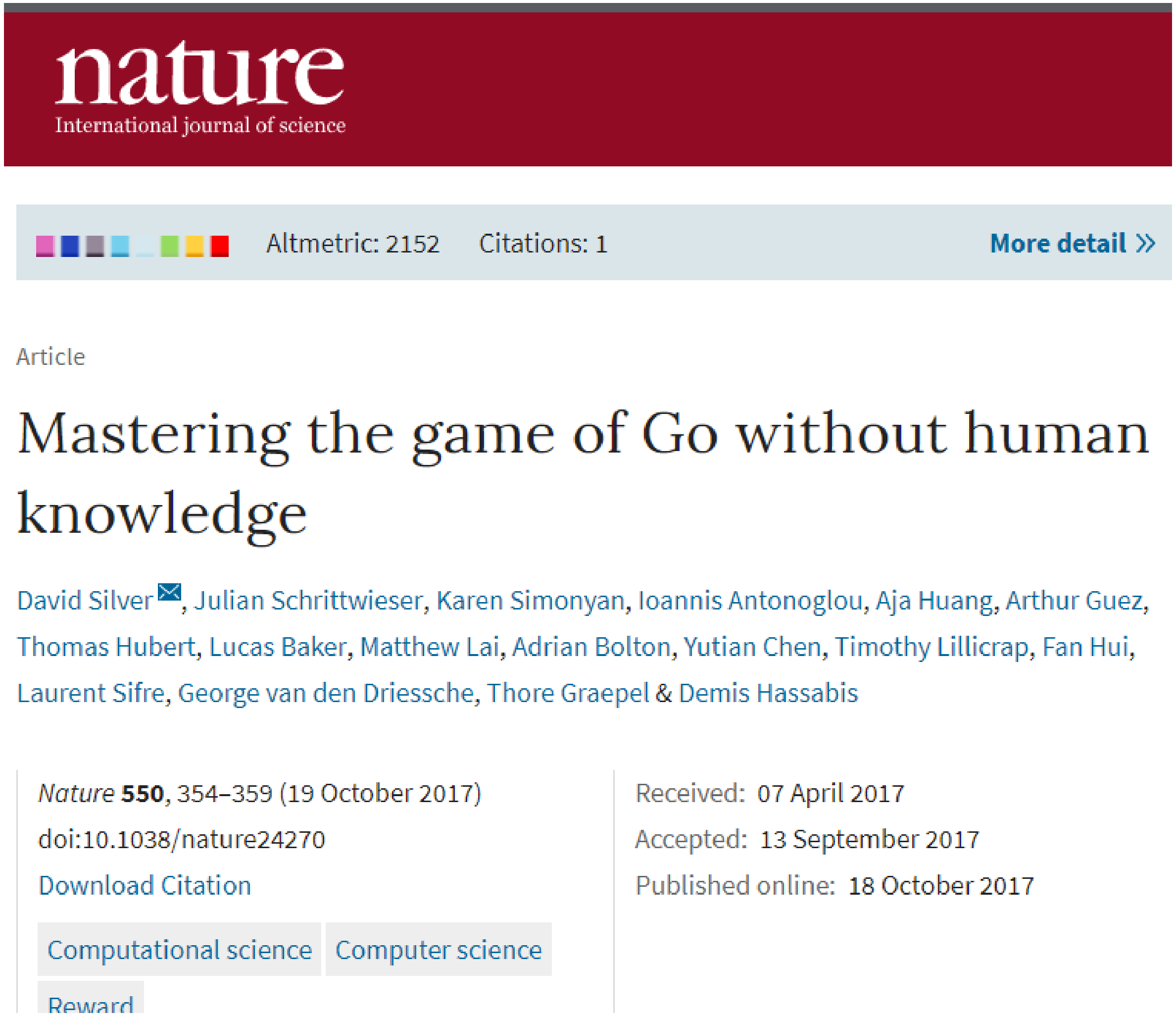
Google Brain’s AlphaGo

- Defeated Lee Sedol, one of the world’s top Go professionals
- The first time a computer program managed to do that
- Highly engineered system with multiple moving parts

AlphaGo’s policy network

- Stage A: a deep convolutional network trained by trying using supervised learning to predict human moves in a game database
    - A ConvNet makes sense since Go “shapes” – configurations of stones – are local, and might be detectable with convolutional layers
- Stage B: use Reinforcement Learning to learn the policy network by making the policy network play against a previous iteration of the policy network
    - Reward: winning a game
    - Train using Policy Gradient
- Use a sophisticated game tree search algorithm together with the Policy Network to actually play the game

AlphaGo Zero



nature
International journal of science

Altmetric: 2152 Citations: 1 More detail »

Article

Mastering the game of Go without human knowledge

David Silver ✉, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis

Nature **550**, 354–359 (19 October 2017)
doi:10.1038/nature24270
Download Citation

Computational science Computer science Reward

Received: 07 April 2017
Accepted: 13 September 2017
Published online: 18 October 2017

AlphaGo Zero

- Does not use a database of human moves to train the initial network that evaluates positions
- Does not use "rollouts"
  - At test time, just evaluate all the possible positions one move ahead
- Used $25 million of hardware