A crash course in machine learning

Florent Krzakala & Antoine Baker

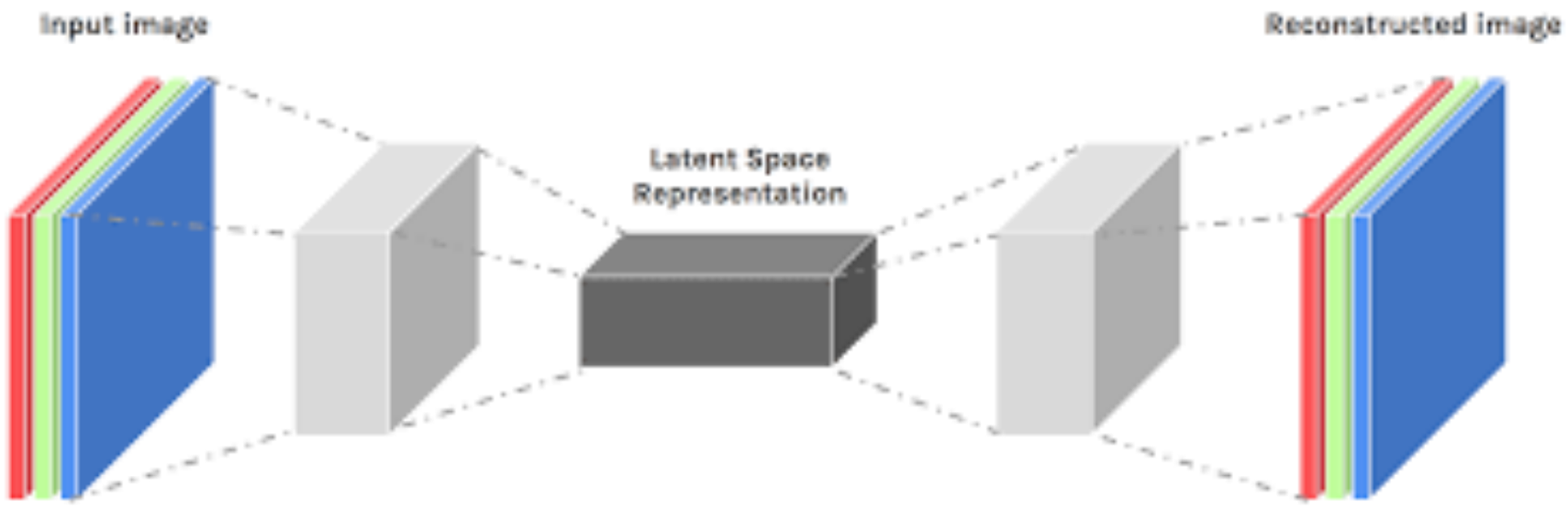
https://sphinxteam.github.io/mlcourse_2019/

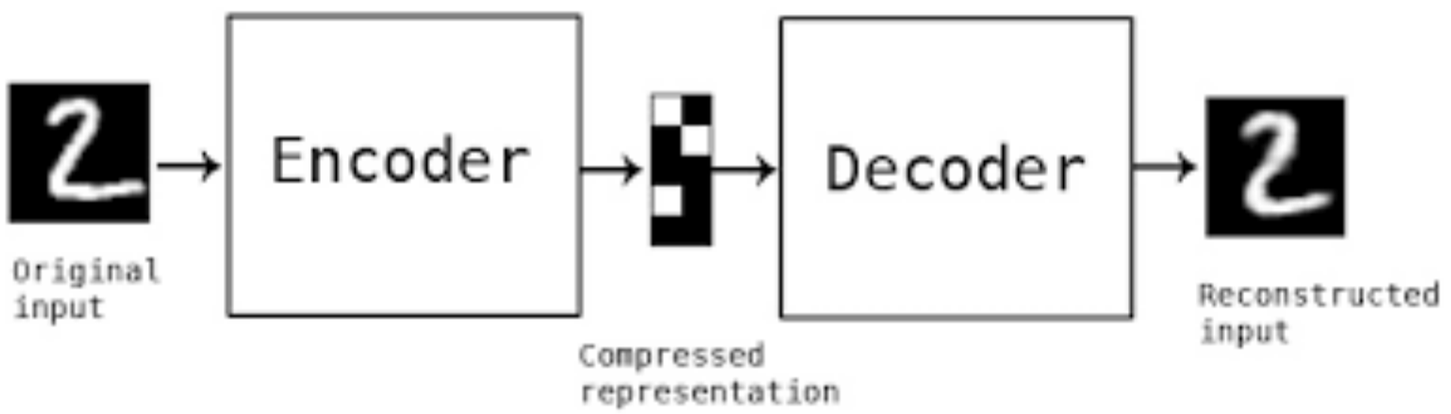
florent.krzakala@gmail.com **antoine.baker@gmail.com**

Achitectures

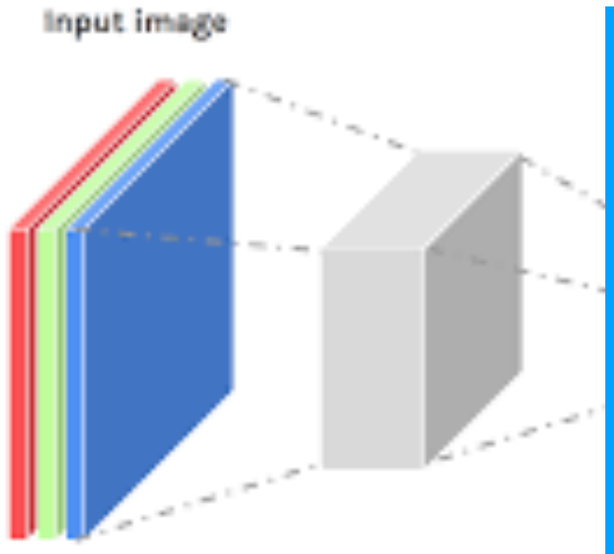
auto-encoders

What can you learn if you do not have labels?



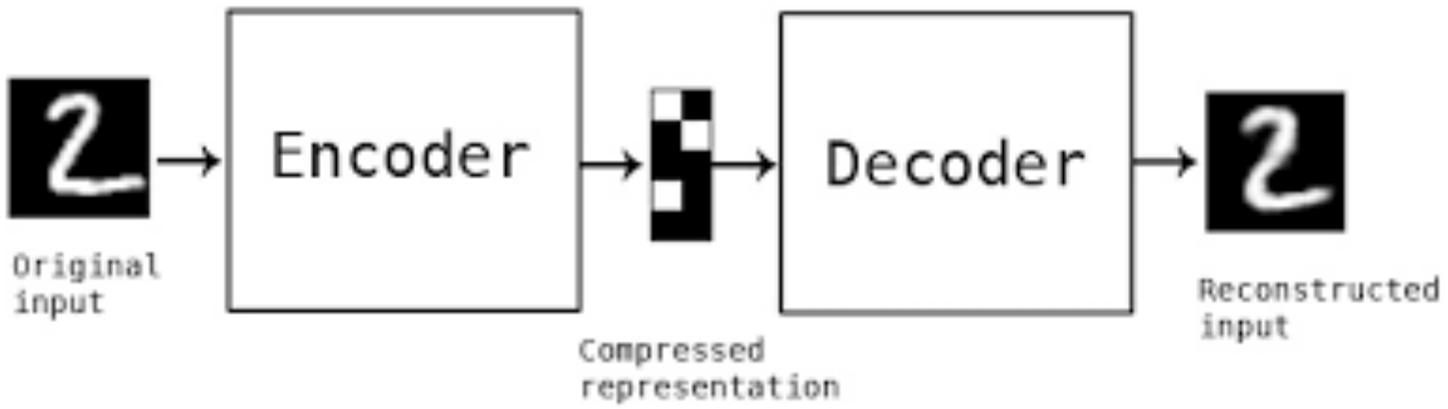


Learn representation without labels



Cute here and add a single layer!

Pre-training without labels!

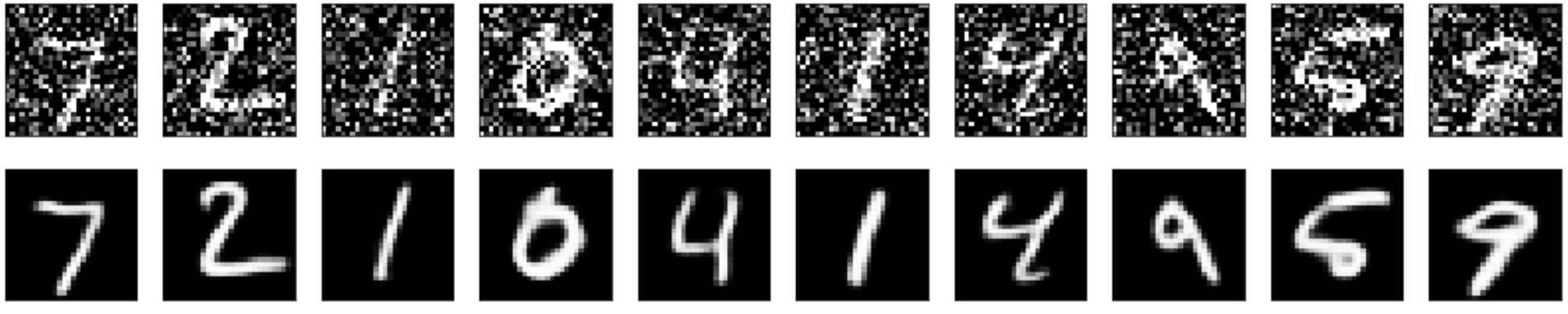


Learn representation without labels

Fundamental in the early day of deep learning for pre-training

Denoising auto-encoder

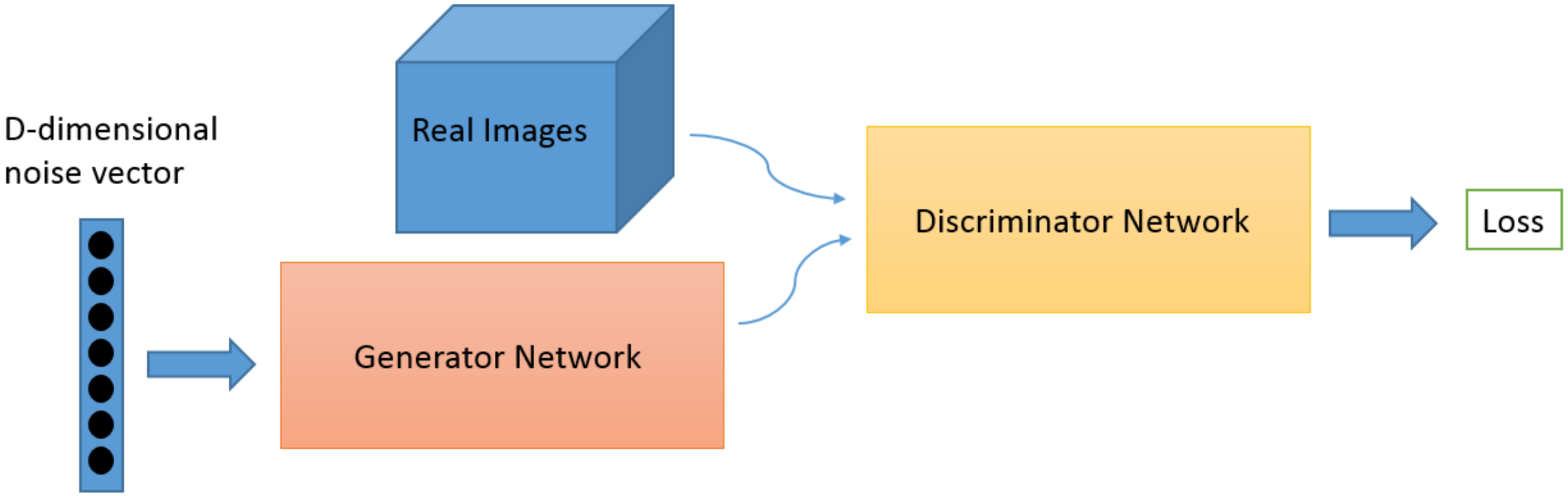
```
decoded_imgs = denoiser_cnn.predict(x_test_noisy)

n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



Denoising auto-encoder

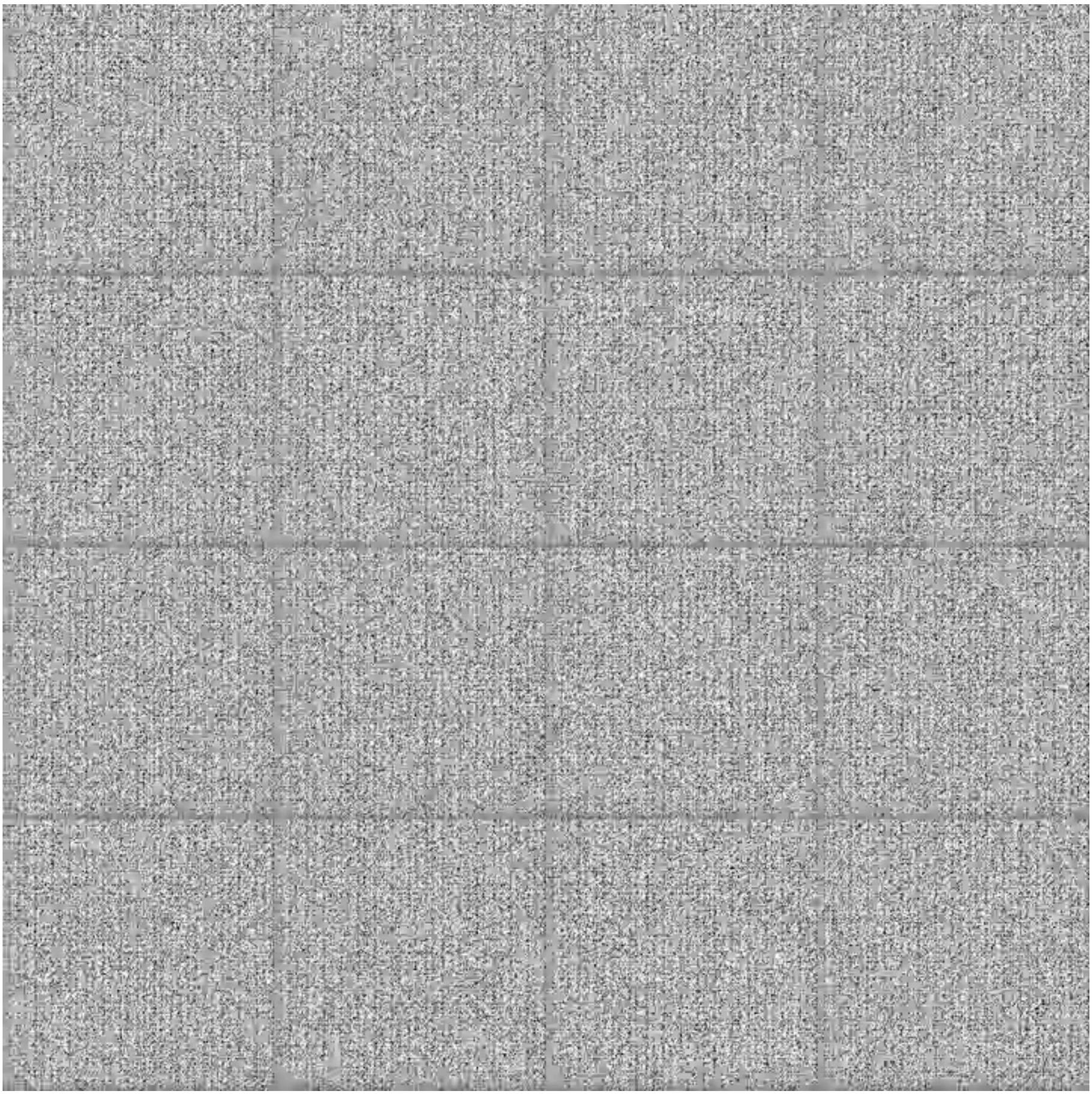
GANs

Generative Adversarial Networks

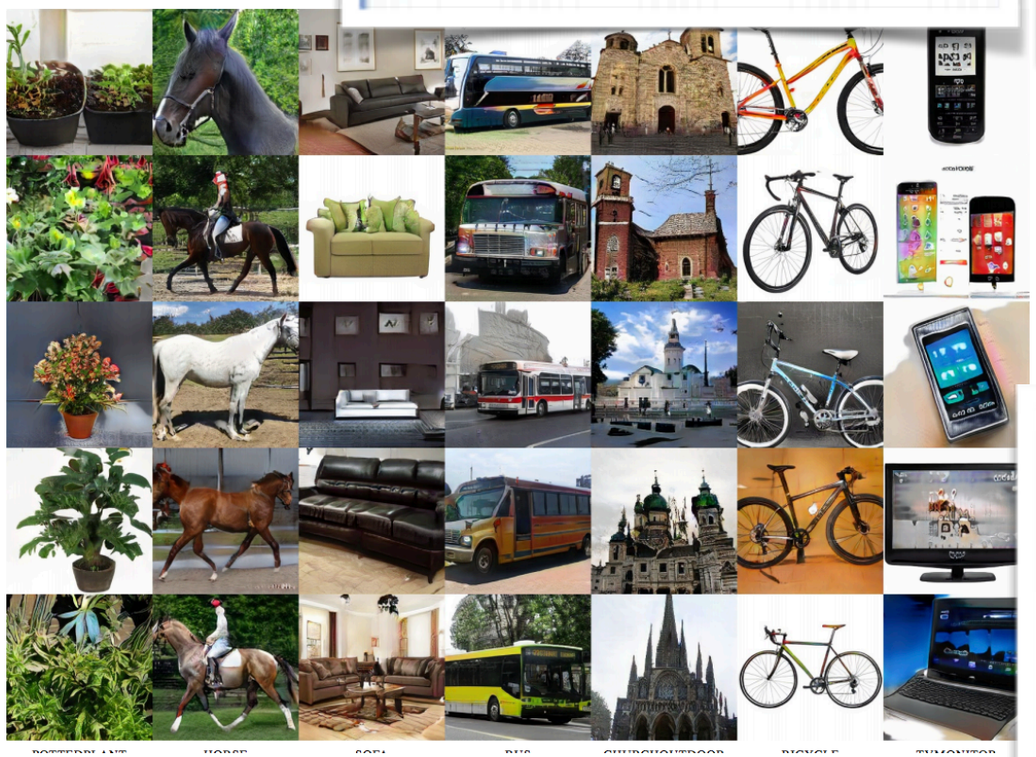


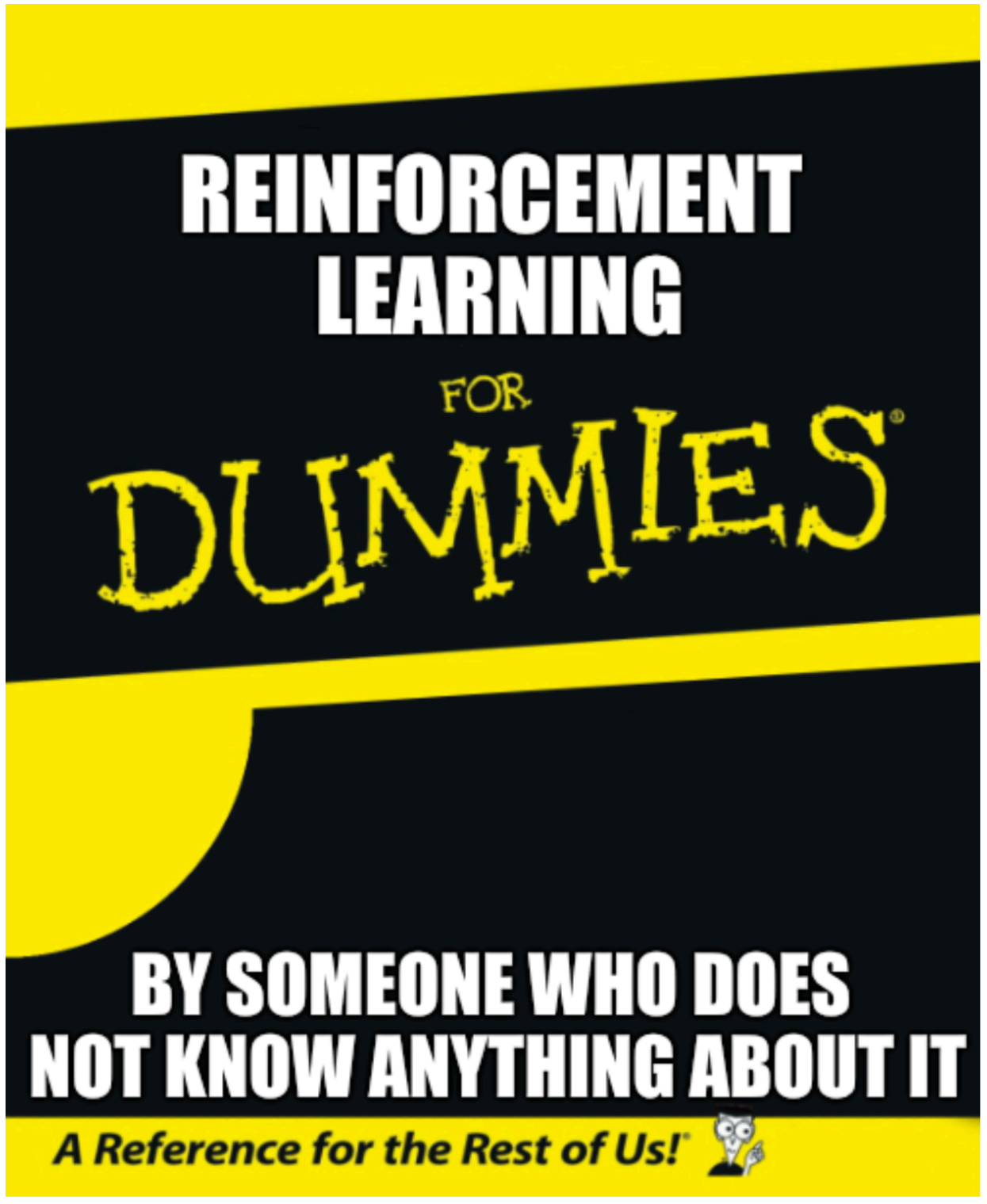
The loss function is measuring how good the discriminator can distinguish between real and generated images!

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$



Nvidia @NIPS 2017




REINFORCEMENT
LEARNING
FOR
DUMMIES
BY SOMEONE WHO DOES
NOT KNOW ANYTHING ABOUT IT
A Reference for the Rest of Us!

This tutorial

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou
Daan Wierstra Martin Riedmiller
DeepMind Technologies
{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

1 Introduction

Learning to control agents directly from high-dimensional sensory inputs like vision and speech is one of the long-standing challenges of reinforcement learning (RL). Most successful RL applications that operate on these domains have relied on hand-crafted features combined with linear value functions or policy representations. Clearly, the performance of such systems heavily relies on the quality of the feature representation.

Recent advances in deep learning have made it possible to extract high-level features from raw sensory data, leading to breakthroughs in computer vision [11, 22, 16] and speech recognition [6, 7]. These methods utilise a range of neural network architectures, including convolutional networks, multilayer perceptrons, restricted Boltzmann machines and recurrent neural networks, and have exploited both supervised and unsupervised learning. It seems natural to ask whether similar techniques could also be beneficial for RL with sensory data.

However reinforcement learning presents several challenges from a deep learning perspective. Firstly, most successful deep learning applications to date have required large amounts of hand-labelled training data. RL algorithms, on the other hand, must be able to learn from a scalar reward signal that is frequently sparse, noisy and delayed. The delay between actions and resulting rewards, which can be thousands of timesteps long, seems particularly daunting when compared to the direct association between inputs and targets found in supervised learning. Another issue is that most deep learning algorithms assume the data samples to be independent, while in reinforcement learning one typically encounters sequences of highly correlated states. Furthermore, in RL the data distribution changes as the algorithm learns new behaviours, which can be problematic for deep learning methods that assume a fixed underlying distribution.

This paper demonstrates that a convolutional neural network can overcome these challenges to learn successful control policies from raw video data in complex RL environments. The network is trained with a variant of the Q-learning [26] algorithm, with stochastic gradient descent to update the weights. To alleviate the problems of correlated data and non-stationary distributions, we use

1

nature
International journal of science

Article | Published: 18 October 2017

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) | Download Citation ⬇

Abstract

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by

RL 101

States & Actions

Rewards, Policy & Value

States & Actions

Curent state of the system

Actions of the player

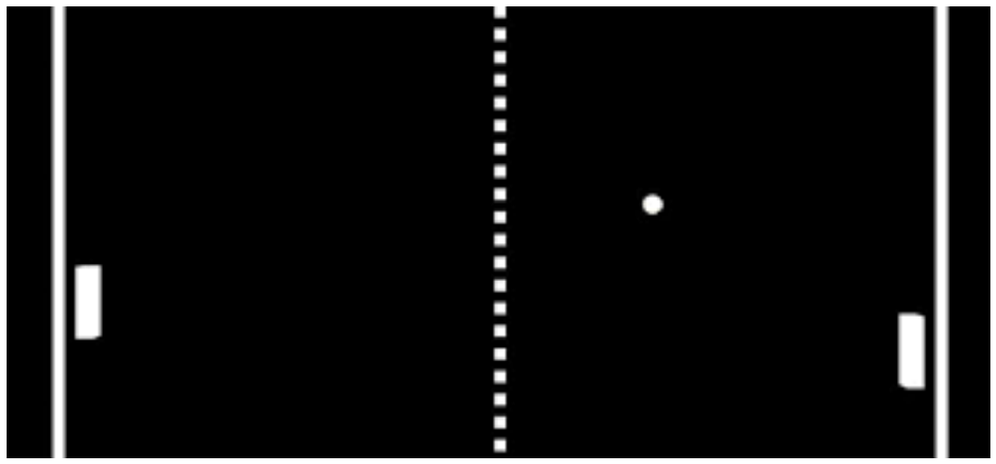
States & Actions

Curent state of the system

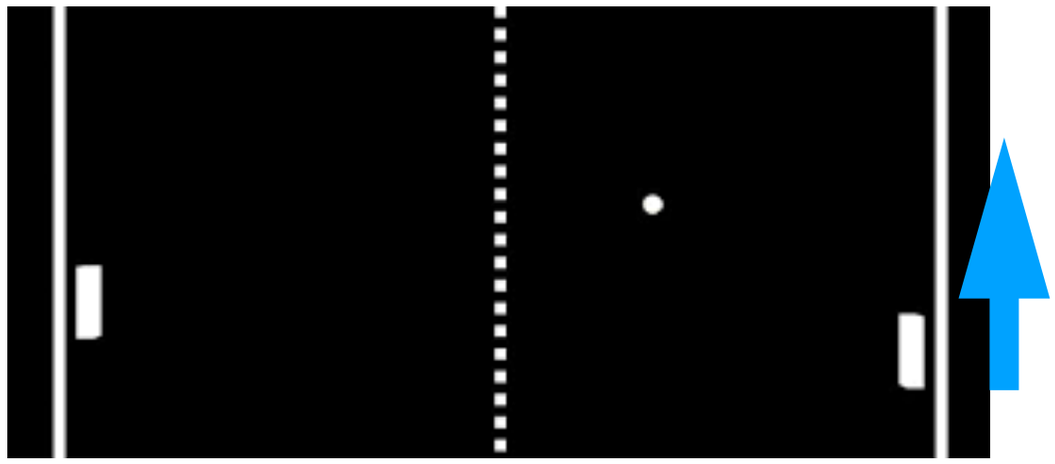
Actions of the player

States & Actions

Curent state of the system



Actions of the player



States & Actions

Curent state of the system

Actions of the player

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$

More precisely, it includes future rewards under perfect play

$$V(\{a_t, s_t\}) = r(t) + \sum_{t=\tau+1}^{\infty} \gamma^{\tau} r^{\mathrm{BEST}}(\tau)$$

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$

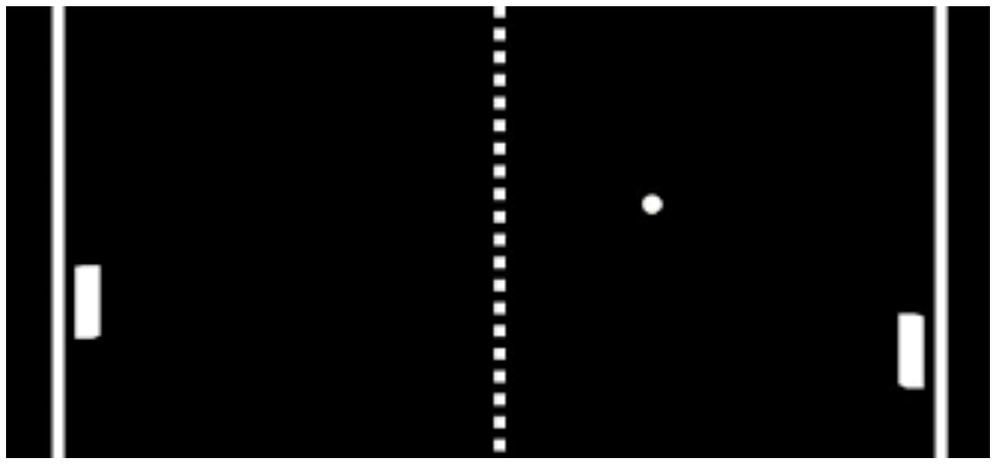
Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$



Score = 1 if ball still in the game

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$

Score = ???

Policy

What action are you playing given the state ?

$$\pi(a \mid s) \equiv \mathbb{P}(a \mid s)$$

Policy $\pi(a \mid s) \equiv \mathbb{P}(a \mid s)$

Reward r^t = score at time t as a result of the action

Value $V(\{a_t, s_t\}) = r(t) + \sum_{t=\tau+1}^{\infty} \gamma^{\tau} r^{\mathrm{BEST}}(\tau) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$

Q-Learning

Imagine one has access to a “cheat” table Q *(s,a) giving,
for every state of the system s*, the <u>values</u> *for each action a under perfect play

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t)$$

Q-Learning

Imagine one has access to a “cheat” table Q *(s,a) giving,
for every state of the system s*, the <u>values</u> *for each action a under perfect play

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t)$$

(s) -> action a|s —> (s’)

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t) = r(a) + \gamma V(s')$$

Q-Learning

Imagine one has access to a "cheat" table Q *(s,a) giving,
for every state of the system s*, the <u>values</u> *for each action a under perfect play

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t) = r(a) + \gamma V(s')$$

Q-Learning

Imagine one has access to a "cheat" table Q *(s,a) giving,
for every state of the system s*, the values *for each action a under perfect play

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t) = r(a) + \gamma V(s')$$

Then the optimal policy is

$$\pi^*(a \mid s) = \delta(a, \text{argmax}_{\text{a}}(Q^*(s,a)))$$

How to find the Q-table?

(i) Start with an estimated one, possibility random $Q^{t=0}(s, a)$

(ii) Iterate the following equation:

$$Q^{t+1}(s, a) = r_a(t) + \gamma \max_{a^{t+1}} \left[Q^t(s^{t+1}, a^{t+1}) \right]$$

(iii) Claim: the table will converge to Q*

Proof!

$$Q^{t+1}(s,a) = r_a(t) + \gamma \max_{a^{t+1}} \left[Q^t(s^{t+1}, a^{t+1}) \right]$$

(a) Q* is a fixed point!

$$Q^*(s,a) = r_a + \sum_{t=1}^{\infty} \gamma^t r^{\text{best}|a}(t) = r(a) + \gamma V(s')$$

$$Q^*(s,a) = r_a + \gamma \max_{a'} \left[Q^*(s' \mid a, a') \right]$$

Proof!

$$Q^{t+1}(s,a) = r_a(t) + \gamma \max_{a^{t+1}} \left[Q^t(s^{t+1}, a^{t+1}) \right]$$

(b) the iteration is contractive

$$Q_1^{t+1}(s,a) - Q_2^{t+1}(s,a) = r(s,a) + \gamma \max_{a^{t+1}} \left[Q_1^t(s^{t+1}, a^{t+1}) \right] - r(s,a) - \gamma \max_{a^{t+1}} \left[Q_2^t(s^{t+1}, a^{t+1}) \right]$$

$$Q_1^{t+1}(s,a) - Q_2^{t+1}(s,a) = \gamma \left(\max_{a^{t+1}} \left[Q_1^t(s^{t+1}, a^{t+1}) \right] - \max_{a^{t+1}} \left[Q_2^t(s^{t+1}, a^{t+1}) \right] \right)$$

$$Q_1^{t+1}(s,a) - Q_2^{t+1}(s,a) \leq \gamma \max_{a',s'} \left(\left[Q_1^t(s',a') \right] - \left[Q_2^t(s',a') \right] \right)$$

$$\| Q_1^{t+1}(s,a) - Q_2^{t+1}(s,a) \|_\infty \leq \gamma \| \left[Q_1^t(s',a') \right] - \left[Q_2^t(s',a') \right] \|_\infty$$

Proof!

$$Q^{t+1}(s,a) = r_a(t) + \gamma \max_{a^{t+1}} \left[Q^t(s^{t+1}, a^{t+1}) \right]$$

(a) Q* is a fixed point!

(b) the iteration is contractive

Convergence to Q*

Banach fixed-point theorem

From Wikipedia, the free encyclopedia

In mathematics, the **Banach–Caccioppoli fixed-point theorem** (also known as the **contraction mapping theorem** or **contraction mapping principle**) is an important tool in the theory of metric spaces; it guarantees the existence and uniqueness of fixed points of certain self-maps of metric spaces, and provides a constructive method to find those fixed points. It can be understood as an abstract formulation of Picard's method of successive approximations.[1] The theorem is named after Stefan Banach (1892–1945) and Renato Caccioppoli (1904–1959), and was first stated by Banach in 1922. Caccioppoli independently proved the theorem in 1931.[2]

Bellman equation

$$Q^{t+1}(s,a) = r(t) + \gamma \max_{a^{t+1}} \left[Q^{t}(s^{t+1}, a^{t+1}) \right]$$

Bellman equation

$$Q^{t+1}(s,a) = r(t) + \gamma \max_{a^{t+1}} \left[Q^{t}(s^{t+1}, a^{t+1}) \right]$$

Mostly **useless** in practice: we can never apply the Bellman operator **exactly**, except for very small problems where we can easily iterate over all (s,a)

Instead try out randomly some actions & iterate a damped version of the algorithm

$$Q^{t+1}(s,\ a) = (1-\delta) * Q^{t}(s,\ a) + \delta(R(a) * \gamma \max_{a'} Q^{*}(s'|a,\ a'))$$

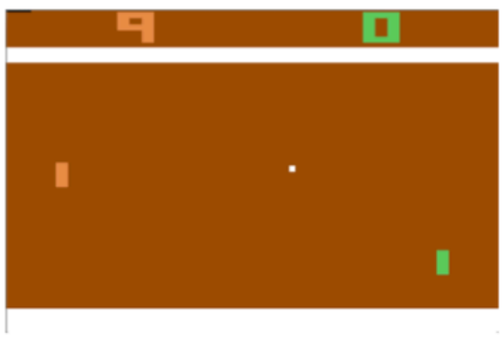
How to try "randomly ?"

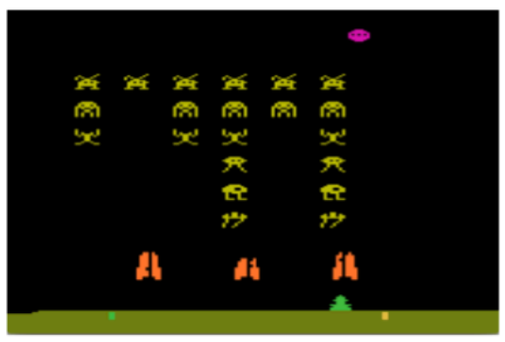
Follow current policy (exploitation "in-policy") p=1-ε

Try random moves (exploration "out-policy") p=ε

Atari games

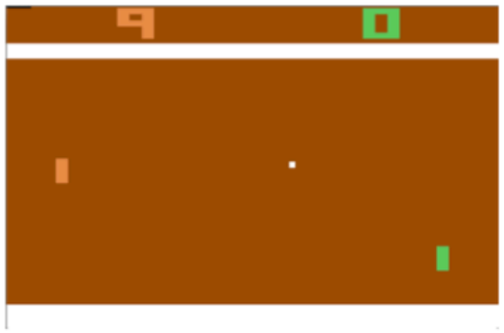
Reinforcement leanring

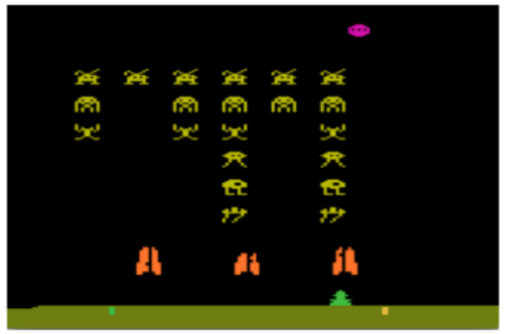





Atari games

Reinforcement leanring






Ask a computer to learn how to play breakout such that the <u>score is good at the end of the game</u>

Rules: The computer “see” the image & control the joystick.
Asked to maximise the end score

Reinforcement learning

Action a^t = action of the joystick

Reward r^t = score at time t as a result of the action

Q-learning

Imagine one has access to a table Q *(s,a) giving, *for every state of the system s*, the (weighted) product of all best score in the future *for each action a*

$$Q^*(s,a) = \sum_{t=0}^{\infty} \gamma^t r^{\text{best}}(t)$$

Reinforcement learning

Action a^t = action of the joystick

Reward r^t = score at time t as a result of the action

Q-learning

Imagine one has access to a table Q *(s,a) giving, *for every state of the system s*, the (weighted) product of all best score in the future *for each action a*

$$Q^*(s,a) = \sum_{t=0}^{\infty} \gamma^t r^{\text{best}}(t)$$

(Bellmann equation)

$$Q^*(s,a) = r(t=0) + \gamma \max_{a^{t+1}} \left[Q^*(s^{t+1}, a^{t+1}) \right]$$

Reinforcement learning

Action a^t = action of the joystick

Reward r^t = score at time t as a result of the action

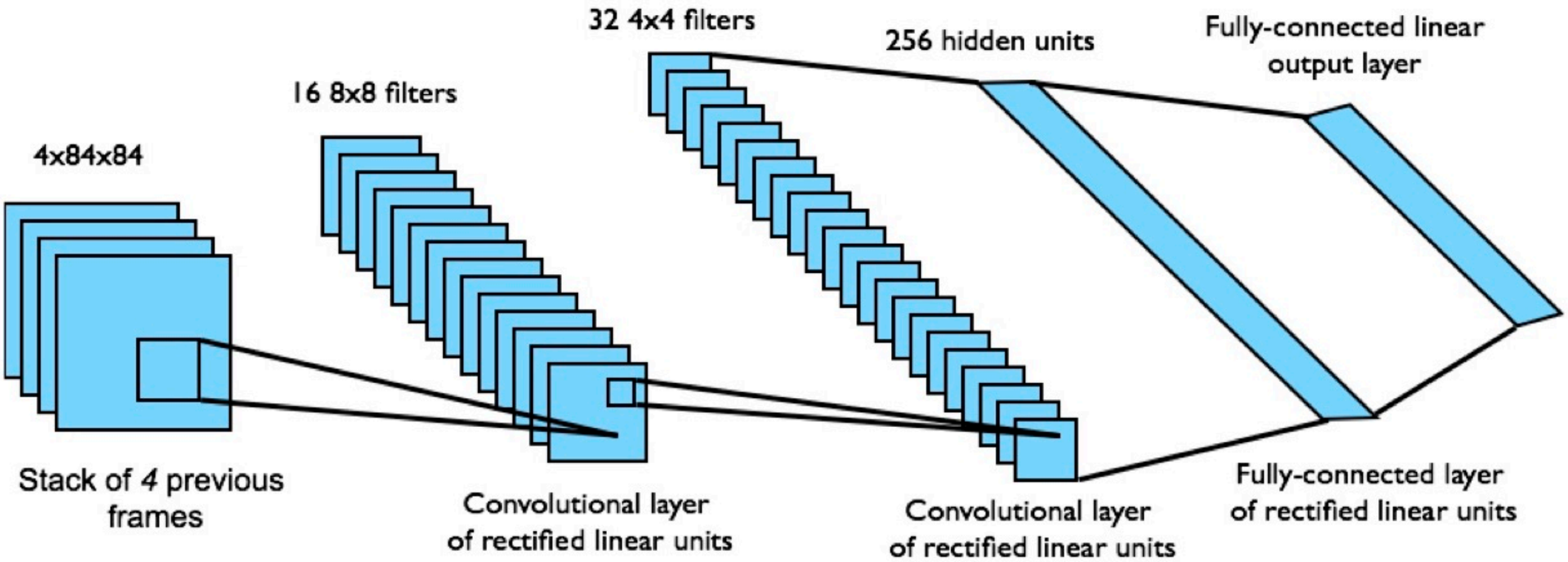
Q-learning

Given an estimate of the ideal Q(s,a), define the cost

$$\text{Cost} = \sum_{s} \left(Q(s, a) - r(t) - \gamma \max_{a'} \left[Q(s^{t+1}, a^{t+1}) \right] \right)^2$$

Learn the function Q(s,a) by gradient descent

Represent Q(s,a) as a convnet



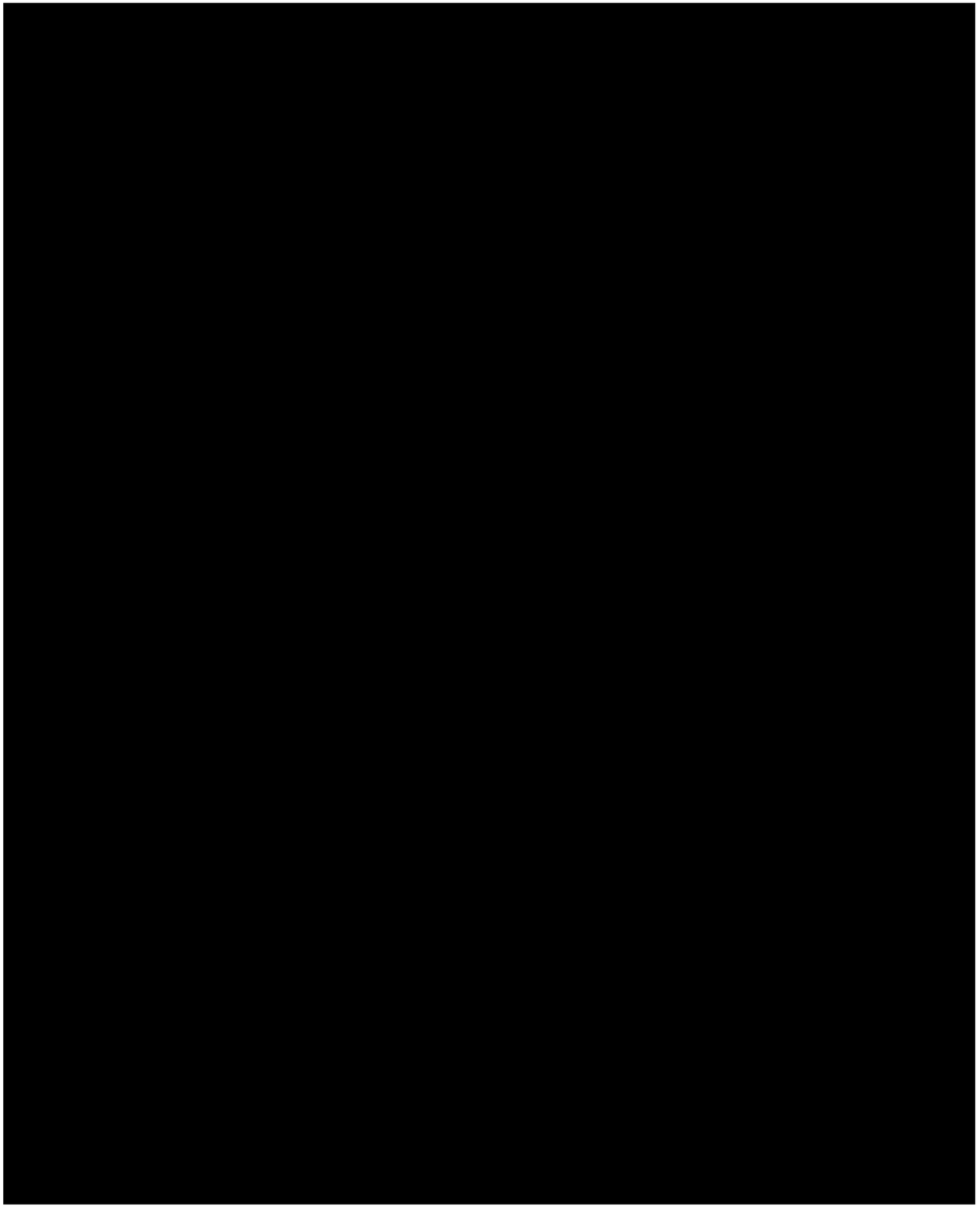
Experience Replay

Create a “replay buffer” that stores current state of the games.
At each step: sample a small batch of “experience” to feed our neural network.

Advantage: more efficient sampling go the q-table, and intuitively more pleasing

Ex: if we are in the first level and then the second (which is totally different), our agent can forget how to behave in the first level.

By learning how to play on water level, our agent will forget how to behave on the first level




Before training
peaceful swimming

Policy Gradients

Optimize directly the policy π(a|s) without using the Q-table
Advantage : can be use for continuous variable, direct optimization

$$J(\theta) = \mathbb{E}_{\pi_\theta, \vec{s}}\left[V(\vec{s})\right]$$

Compute the gradient of J !

REINFORCE

$$J(\theta) = \mathbb{E}[f(X)] = \int_x f(x) p_{\boldsymbol{\theta}}(x)\, dx$$

$$\nabla_\theta J(\theta) = \int_x f(x) \nabla_\theta p_{\boldsymbol{\theta}}(x)\, dx = \int_x f(x) p(x) \nabla_\theta \log(p_{\boldsymbol{\theta}}(x))\, dx = \mathbb{E}[f(x) \nabla_\theta \log(p_{\boldsymbol{\theta}}(x))$$

Now, assume that we are dealing with trajectories generated by a Markov chain:

$$p_{\boldsymbol{\theta}}(\tau) = p(s_0) \prod_{i=0}^{\infty} \pi_{\boldsymbol{\theta}}(a_i|s_i) p(s_{i+1}|s_i)$$

$$\log p_{\boldsymbol{\theta}}(\tau) = \log p(s_0) + \sum_i (\log \pi_{\boldsymbol{\theta}}(a_i|s_i) + \log p(s_{i+1}|s_i))$$

While the transition π depends on theta, the actual dynamics does not 8

$$g = \nabla_\theta J(\theta) = \mathbb{E}\left[f(\tau) \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \right]$$

REINFORCE

$$g = \nabla_\theta J(\theta) = \mathbb{E}\left[f(\tau) \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \right]$$

Approximate the expectation by an empirical sum over many “plays”

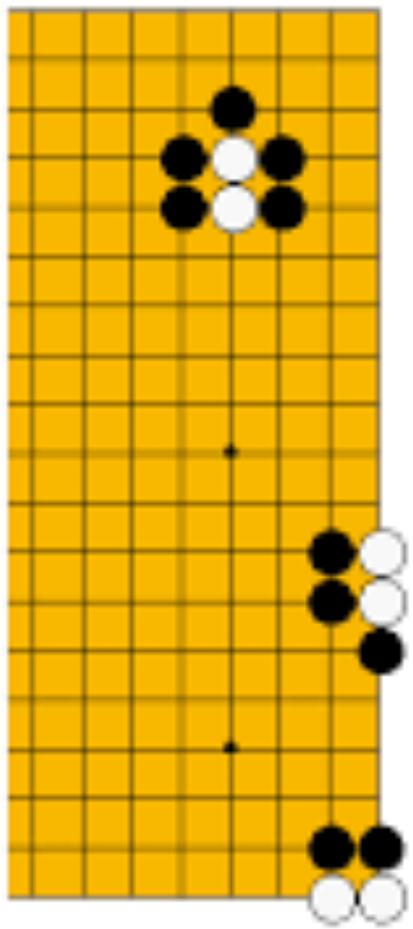
$$g = \nabla_\theta J(\theta) \approx \frac{1}{P} \sum_{p=1}^{P} f(\tau) \sum_i \nabla_\theta \log \pi_\theta(a_i \,|\, s_i)$$

Go

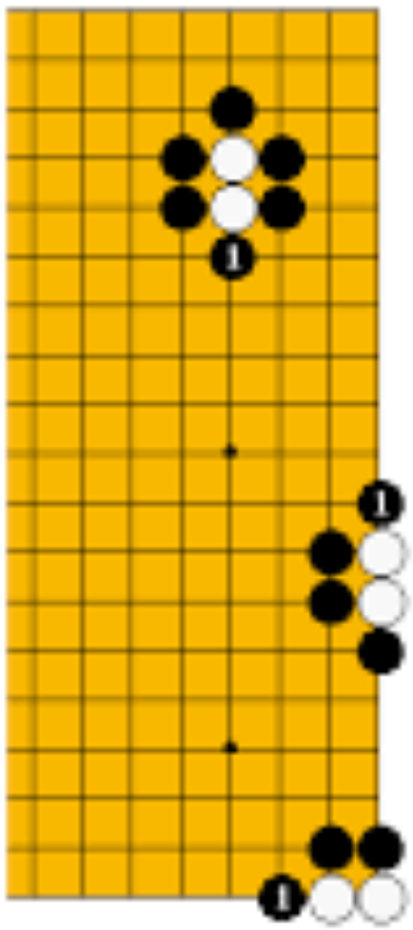
2,500 years old: Oldest game still play today

One type of piece, one type of move

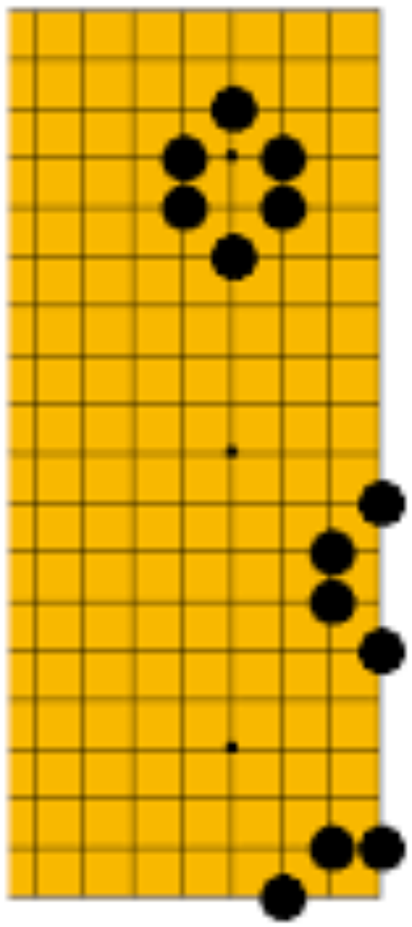
Once placed on the board, stones may not be moved, but stones are removed from the board if "captured". Capture happens when a stone or group of stones is surrounded by opposing stones on all orthogonally-adjacent points



Dia. 18

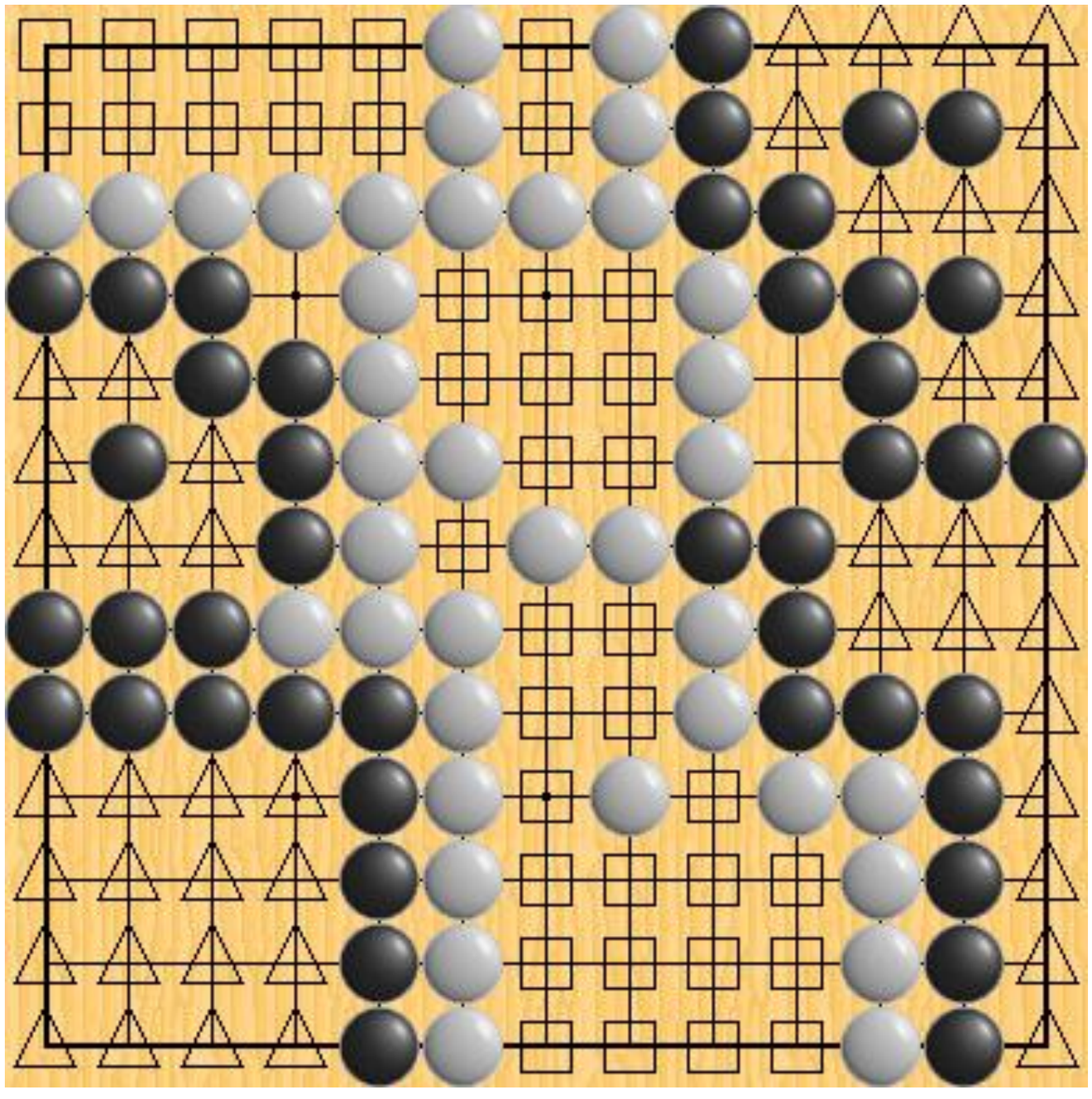


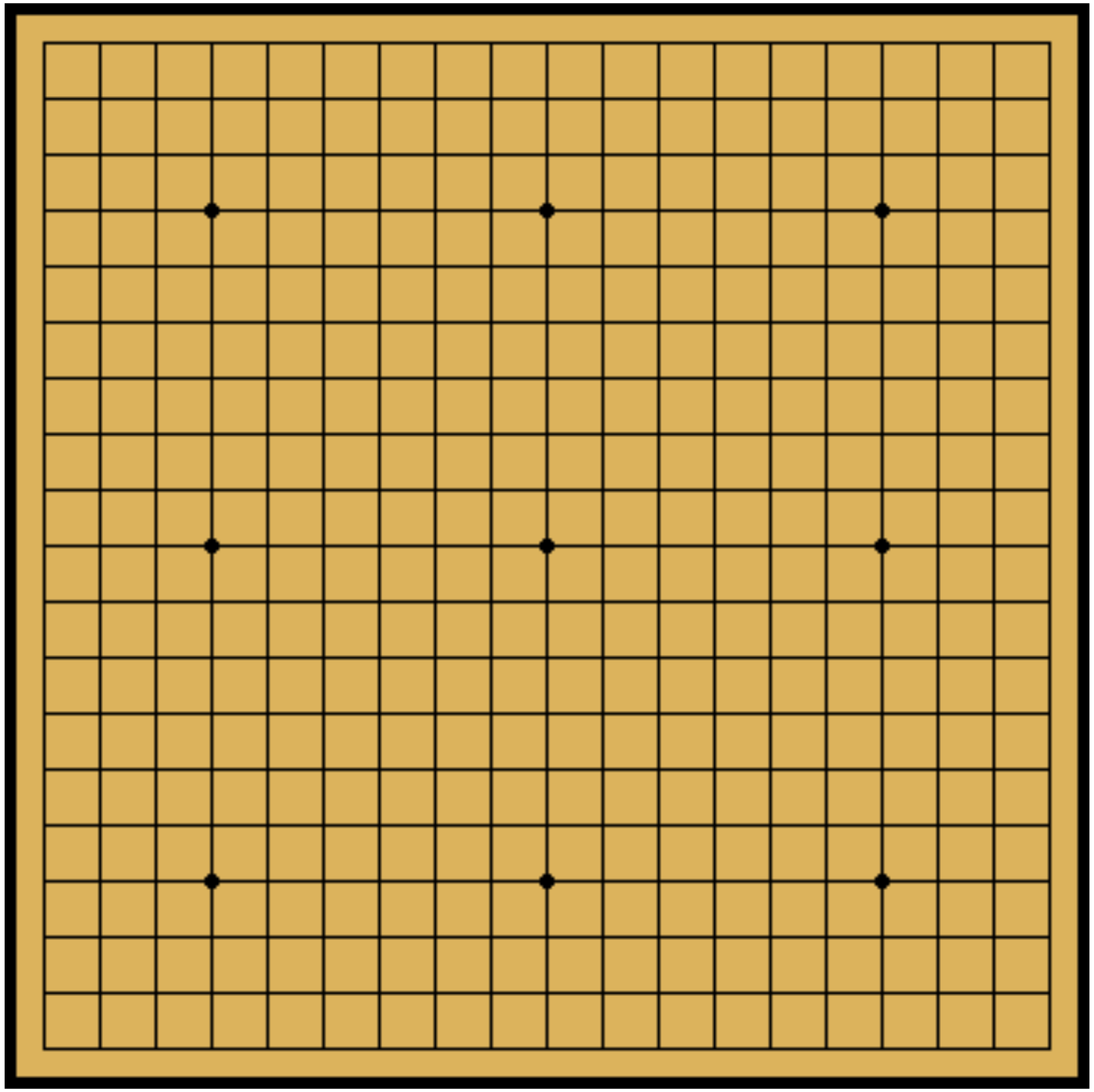
Dia. 19



Dia. 20

A player's score is determined by the number of stones that player has on the board plus the empty area surrounded by that player's stones.






nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
CONSERVATION
SONGBIRDS À LA CARTE
Illegal harvest of millions of Mediterranean birds
PAGE 452
RESEARCH ETHICS
SAFEGUARD TRANSPARENCY
Don't let openness backfire on individuals
PAGE 459
POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
Dawkins's calling card 40 years on
PAGE 462
NATUREASIA.COM
28 January 2016
Vol. 529, No. 7587

STEP 1

Supervised learning of policy networks

We trained a 13-layer policy network, which we call the SL policy network, from 30 million positions from the KGS Go Server. The net- work predicted expert moves on a held out test set with an accuracy of 57.0% using all input features, and 55.7% using only raw board position and move history as inputs,

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$$

STEP 2

Reinforcement learning of policy networks

Play many games

Compute policy gradient, use REINFORCE

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

The outcome $z_t = \pm r(s_T)$ is the terminal reward at the end of the game from the perspective of the current player at time step t: +1 for winning and −1 for losing.

We evaluated the final RL algorithm by sampling each move $a_t \sim p_\varrho(\cdot|s_t)$ from its output probability distribution over actions. When played head-to-head, the RL policy network won more than 80% of games against the SL policy network. We also tested against the strongest open-source Go program, Pachi, a sophisticated Monte Carlo search program, ranked at 2 amateur *dan* on KGS, that executes 100,000 simulations per move. Using no search at all, the RL policy network won 85% of games against Pachi.

STEP 3

Reinforcement learning of value networks

$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t \ldots T} \sim p]$$

Complicated function, approximate by a neural net

Play many games with the policy RL, train the weights of the value network by regression on state-outcome pairs (s, z), using stochastic gradient descent to minimize the mean squared error (MSE) between the predicted value $v_\theta(s)$, and the corresponding outcome z

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

At this pont we have

Policy (supervised) $p_\sigma(a \mid s)$

Policy (RL) $p_\rho(a \mid s)$

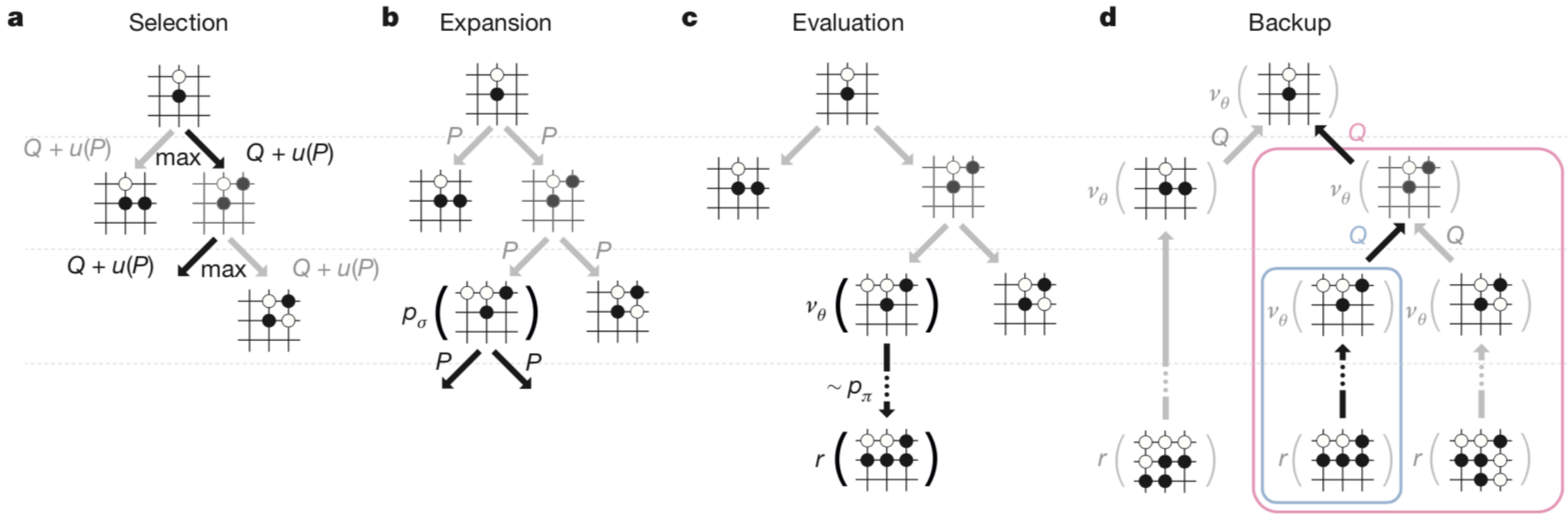
value networks $v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t \ldots T} \sim p]$

How to use this? Monte-Carlo Tree-Search

Guiding the search in the tree of possibilities!

How to use this?

Monte-Carlo Tree-Search



At each time step t of each simulation, an action a_t is selected from state s_t :

$$a_t = \text{argmax}_a \left[Q(s_t, a) + \frac{p_\sigma(s, a)}{1 + N(s, a)} \right]$$

$$N(s, a) = \sum_{i=1}^{n} 1(s, a, i) \qquad Q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{n} 1(s, a, i) V(s)$$

$$V(s) = (1 - \lambda)\nu_\theta(s) + \lambda z_{\text{fast}}$$

Fan Hui

October 2015

Lee Sedol

March 2016

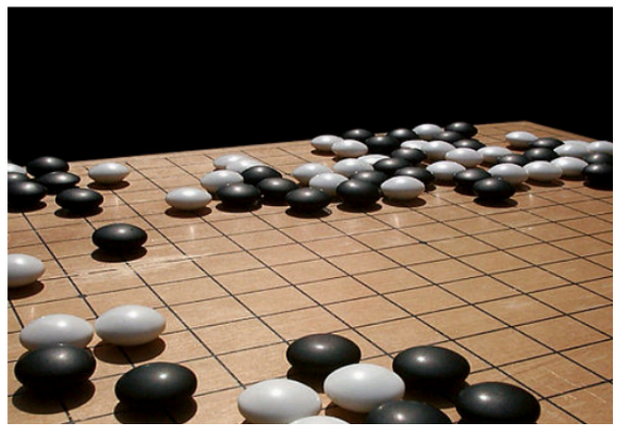
Move 37

See the exact moment the world champion of Go realises DeepMind is vastly superior [GIF]

Jim Edwards
Mar. 12, 2016, 2:30 PM 9,626

HOW GOOGLE'S AI VIEWED THE MOVE NO HUMAN COULD UNDERSTAND

Go fans have noticed a couple of key moments in the match between world champion Lee Sedol and Google's DeepMind AlphaGo



Game 2 [edit]

AlphaGo (black) won the second game. Lee stated afterwards that "AlphaGo played a nearly perfect game",[49] "from very beginning of the game I did not feel like there was a point that I was leading".[50] One of the creators of AlphaGo, Demis Hassabis, said that the system was confident of victory from the midway point of the game, even though the professional commentators could not tell which player was ahead.[50]

Michael Redmond (9p) noted that AlphaGo's 19th stone (move 37) was "creative" and "unique".[30] Lee took an unusually long time to respond to the move.[30] An Younggil (8p) called AlphaGo's move 37 "a rare and intriguing shoulder hit" but said Lee's counter was "exquisite". He stated that control passed between the players several times before the endgame, and especially praised AlphaGo's moves 151, 157, and 159, calling them "brilliant".[51]

Alphago Zero (2017)

Artificial intelligence research has made rapid progress in a wide variety of domains from speech recognition and image classification to genomics and drug discovery. In many cases, these are specialist systems that leverage enormous amounts of human expertise and data.

However, for some problems this human knowledge may be too expensive, too unreliable or simply unavailable. As a result, a long-standing ambition of AI research is to bypass this step, creating algorithms that achieve superhuman performance in the most challenging domains with no human input. In our most recent **paper**, published in the **journal Nature**, we demonstrate a significant step towards this goal.

nature
International journal of science

Article | Published: 18 October 2017

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) | Download Citation

Abstract

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by

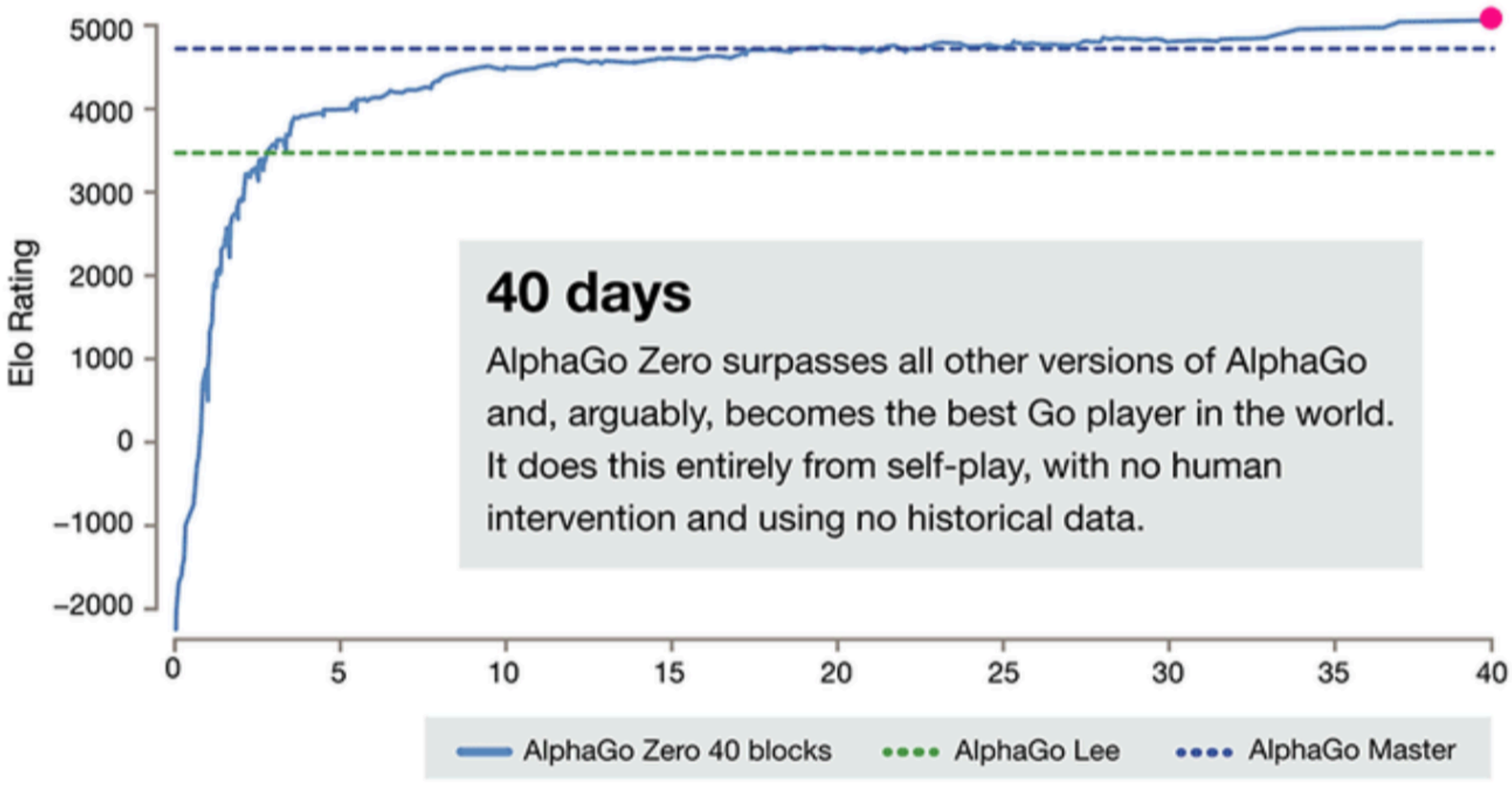
A single network, Learning with RL

1 Reinforcement Learning in *AlphaGo Zero*

Our new method uses a deep neural network f_θ with parameters θ. This neural network takes as an input the raw board representation s of the position and its history, and outputs both move probabilities and a value, $(\mathbf{p}, v) = f_\theta(s)$. The vector of move probabilities $\mathbf{p}$ represents the probability of selecting each move (including pass), $p_a = Pr(a|s)$. The value v is a scalar evaluation, estimating the probability of the current player winning from position s. This neural network combines the roles of both policy network and value network [12] into a single architecture. The neural network consists of many residual blocks [4] of convolutional layers [16,17] with batch normalisation [18] and rectifier non-linearities [19] (see Methods).

AlphaGo Zero

- Beats AlphaGo by 100:0



AlphaGo Zero

- Beats AlphaGo by 100:0

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis

(Submitted on 5 Dec 2017)

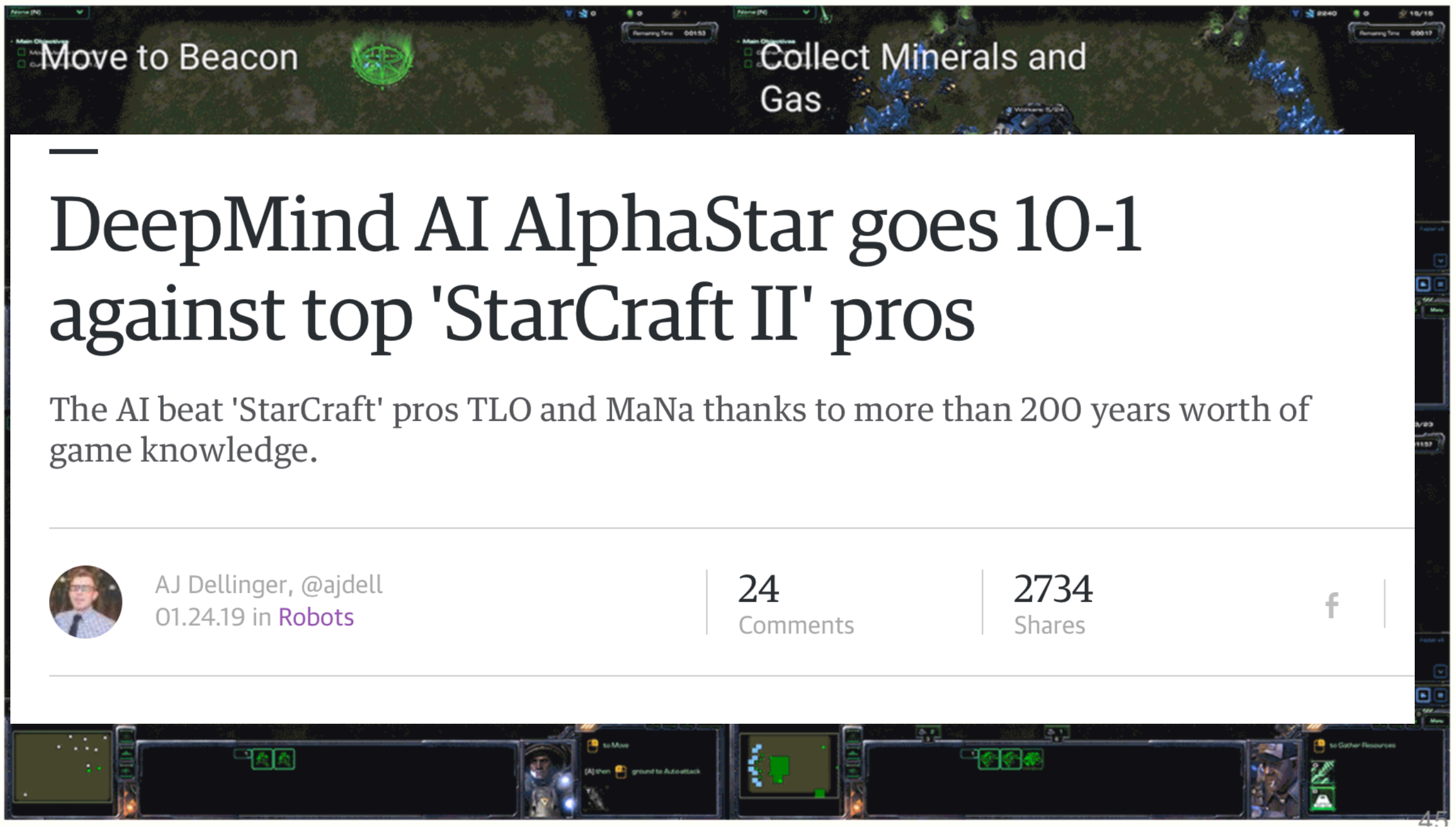
The game of chess is the most widely-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. In contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go, by tabula rasa reinforcement learning from games of self-play. In this paper, we generalise this approach into a single AlphaZero algorithm that can achieve, tabula rasa, superhuman performance in many challenging domains. Starting from random play, and given no domain knowledge except the game rules, AlphaZero achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

What's next for AI?

DeepMind's AI is Struggling to Beat Starcraft II - Bloomberg
https://www.bloomberg.com/.../deepmind-master-of-go-struggles-to-crack-its-next-mi...

What's next for AI?

DeepMind's AI is Struggling to Beat Starcraft II - Bloomberg
https://www.bloomberg.com/.../deepmind-master-of-go-struggles-to-crack-its-next-mi...



DeepMind AI AlphaStar goes 10-1 against top 'StarCraft II' pros

The AI beat 'StarCraft' pros TLO and MaNa thanks to more than 200 years worth of game knowledge.

AJ Dellinger, @ajdell
01.24.19 in Robots

24
Comments

2734
Shares