Convolutional Neural Networks for Reinforcement Learning (Ch.18)

dr. Wojtek Kowalczyk

wojtek@liacs.nl

Leiden Institute of Advanced Computer Science

Universiteit Leiden

Overview

- Learning to play the Atari Breakout game
  - the network
  - key ideas
  - basics of Reinforcement Learning
  - the training algorithm
  - results

- AlphaGo Zero and Alpha Zero
  - Monte Carlo Tree Search (MCTS)
  - the network
  - the training algorithm
  - results

Resources

- Learning to play the Atari Breakout game
  - demo:
    www.youtube.com/watch?v=V1eYniJ0Rnk
  - papers:
    www.cs.toronto.edu/~vmnih/docs/dqn.pdf
    www.nature.com/articles/nature14236
  - presentation of David Silver (DeepMind):
    www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/deep_rl.pdf

- AlphaGo Zero and Alpha Zero
  - AlphaGo Zero and Alpha Zero papers:
    www.nature.com/articles/nature24270
    https://arxiv.org/abs/1712.01815
  - the network
    applied-data.science/static/main/res/alpha_go_zero_cheat_sheet.png
  - Monte Carlo Tree Search (MCTS):
    https://www.sciencedirect.com/science/article/pii/S000437021100052X
  - You Tube explanation:
    www.youtube.com/watch?v=MgowR4pq3e8

https://en.wikipedia.org/wiki/Atari_2600

CPU: 8-bit @ 1.19 MHz

RAM: 128 bytes

ROM: 2kB

Release Year: 1977

Key Ideas:

- train convolutional network to play Breakout

- the network would be taking as input 4 consecutive frames (preprocessed to 4x84x84 pixels) + “reward”;

- 4 frames are needed to contain info about ball direction, speed, acceleration, etc.

- the output consists of 18 nodes that correspond to all possible positions of the joystick (left-right, up-down, 4 diagonals, neutral; plus “red button pressed”)

How could such a network be trained?
What data could be used for training?

DQN for Atari 2600 Games

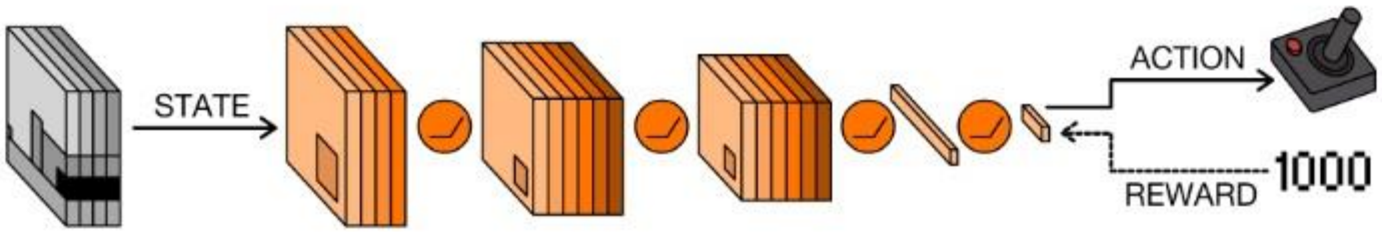
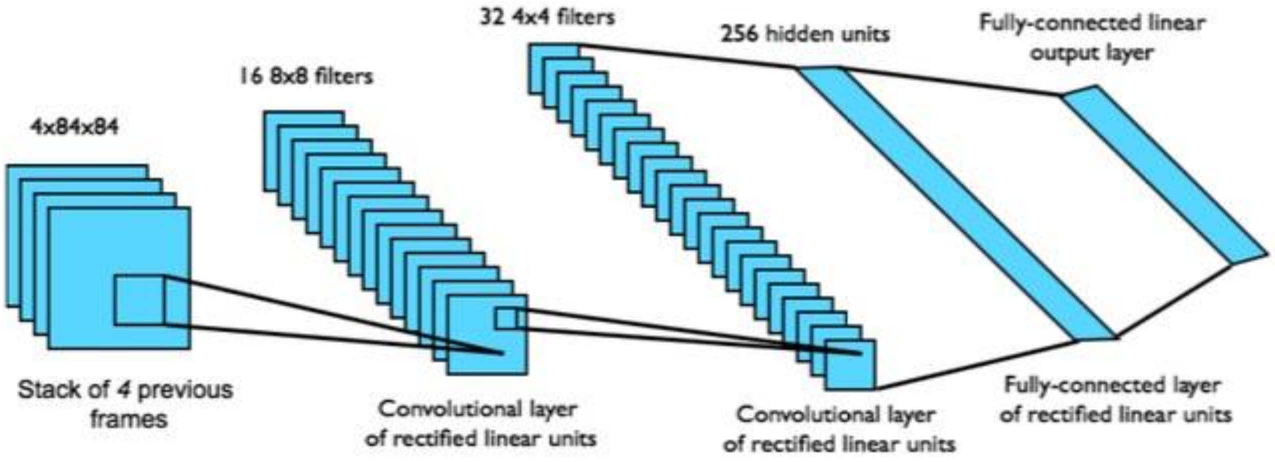
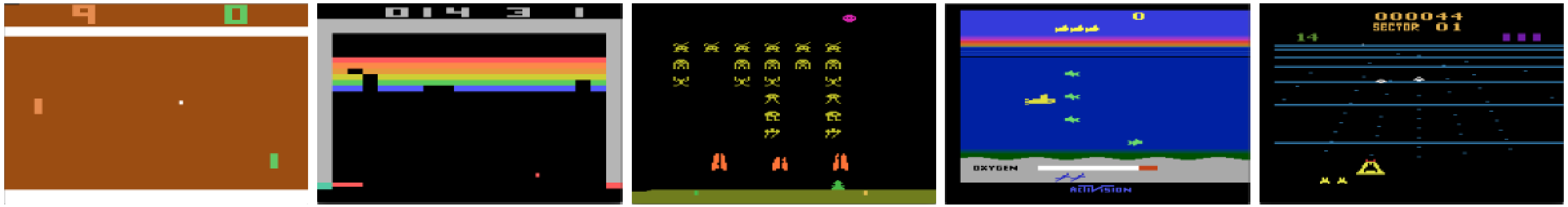


Fig. 4. The DQN [71]. The network takes the state—a stack of greyscale frames from the video game—and processes it with convolutional and fully connected layers, with ReLU nonlinearities in between each layer. At the final layer, the network outputs a discrete action, which corresponds to one of the possible control inputs for the game. Given the current state and chosen action, the game returns a new score. The DQN uses the reward—the difference between the new score and the previous one—to learn from its decision. More precisely, the reward is used to update its estimate of Q, and the error between its previous estimate and its new estimate is backpropagated through the network.

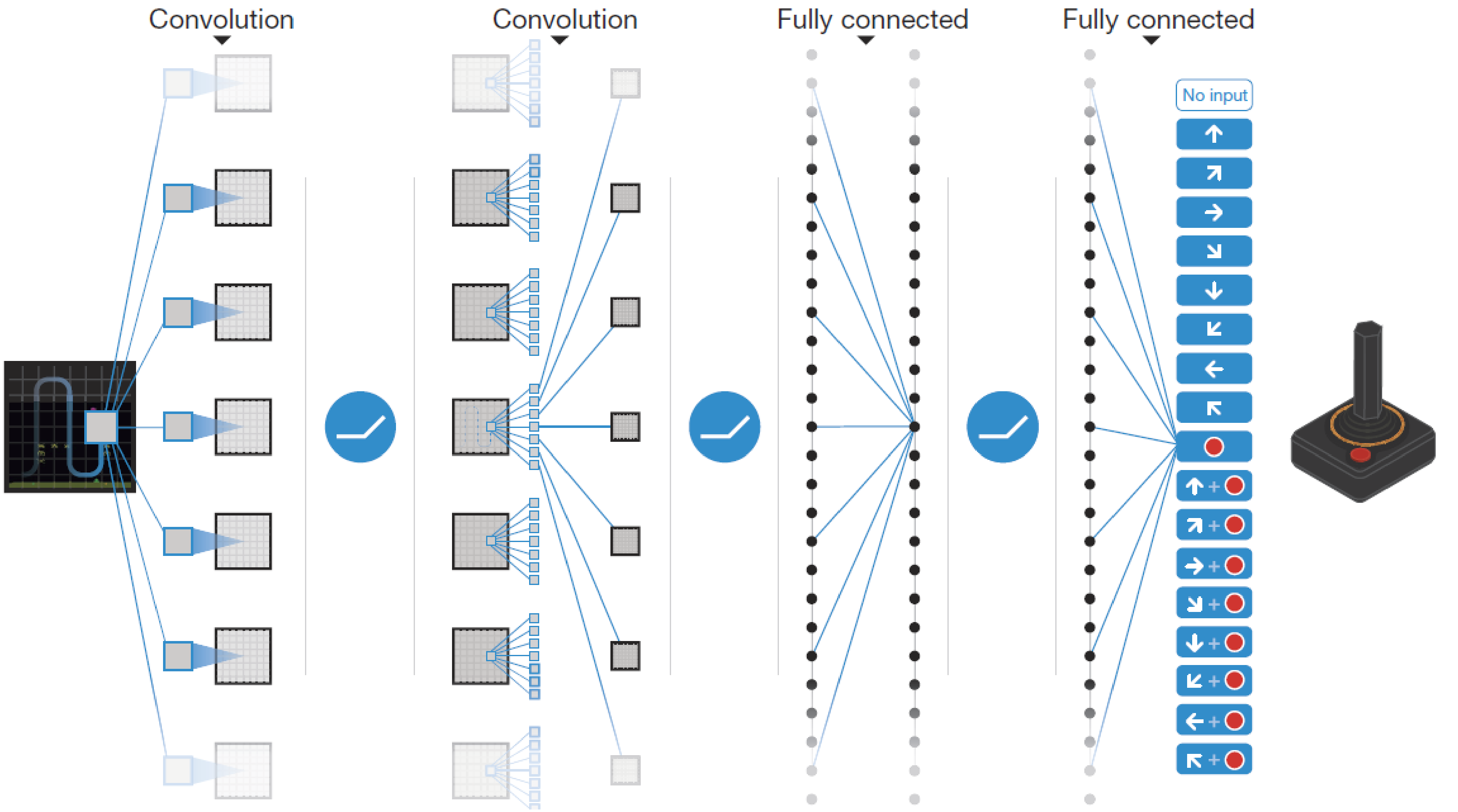




The Reinforcement Learning Approach:

- view it as a Reinforcement Learning Problem:
  states, actions, rewards, policies, value, ...
- assume that the network can estimate the “quality” of possible actions
- initialize the network at random and use it to play many games => generate some training data
- “learn from experience” => use the generated data to improve the network (with help of the Bellman’s equation)
- use the improved network to generate “better data” and return to the previous step; iterate till optimum reached

Will it work for other Atari games?


Convolution
Convolution
Fully connected
Fully connected
No input

Slides 1-19 from D. Silver

www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/deep_rl.pdf

Deep Reinforcement Learning

David Silver, Google DeepMind

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

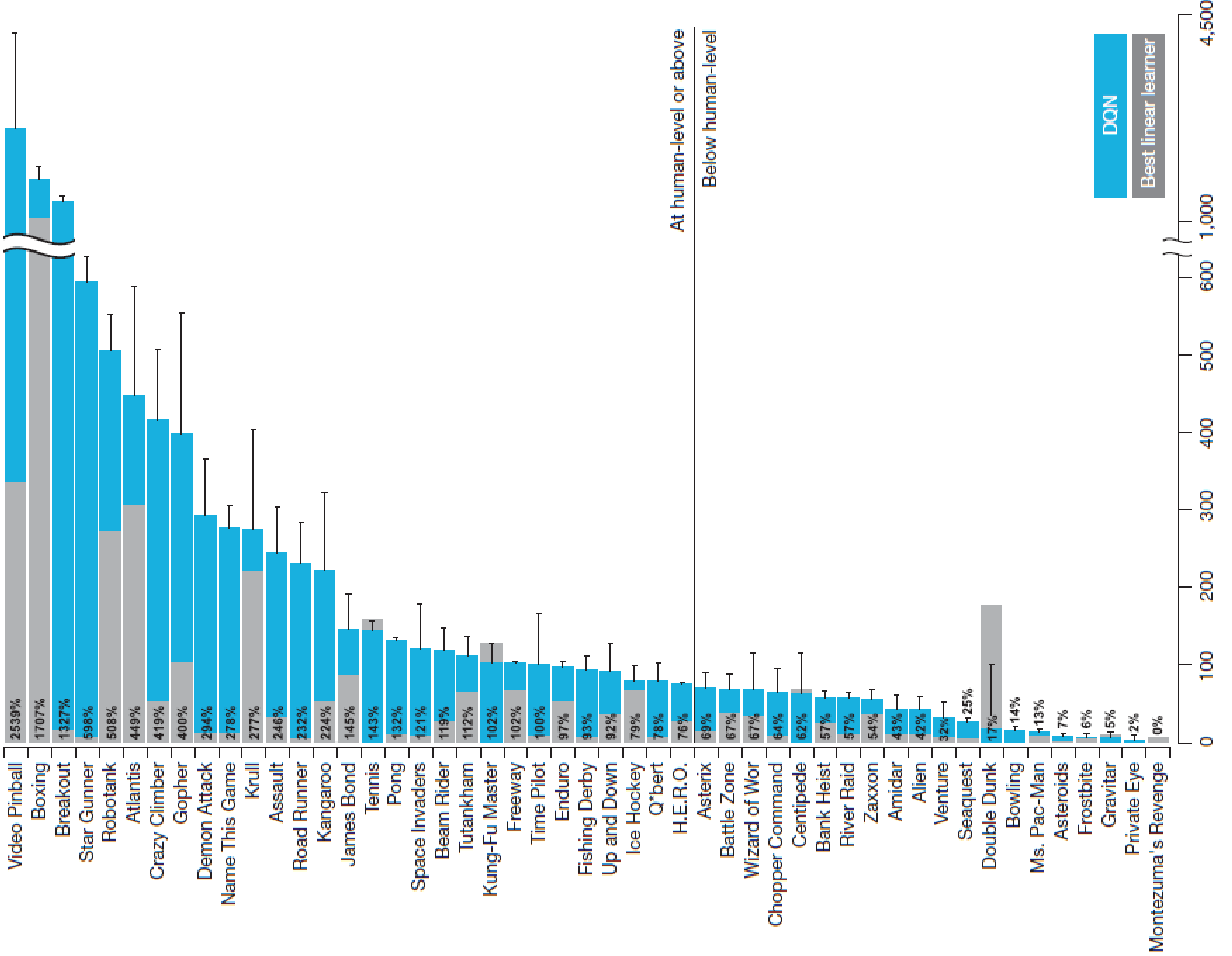
Every C steps reset $\hat{Q} = Q$

End For

End For

Extended Data Table 1 | List of hyperparameters and their values

Hyperparameter	Value	Description
minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed.
replay memory size	1000000	SGD updates are sampled from this number of most recent frames.
agent history length	4	The number of most recent frames experienced by the agent that are given as input to the Q network.
target network update frequency	10000	The frequency (measured in the number of parameter updates) with which the target network is updated (this corresponds to the parameter *C* from Algorithm 1).
discount factor	0.99	Discount factor gamma used in the Q-learning update.
action repeat	4	Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.
update frequency	4	The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.
learning rate	0.00025	The learning rate used by RMSProp.
gradient momentum	0.95	Gradient momentum used by RMSProp.
squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
min squared gradient	0.01	Constant added to the squared gradient in the denominator of the RMSProp update.
initial exploration	1	Initial value of ε in ε-greedy exploration.
final exploration	0.1	Final value of ε in ε-greedy exploration.
final exploration frame	1000000	The number of frames over which the initial value of ε is linearly annealed to its final value.
replay start size	50000	A uniform random policy is run for this number of frames before learning starts and the resulting experience is used to populate the replay memory.
no-op max	30	Maximum number of "do nothing" actions to be performed by the agent at the start of an episode.


DQN
Best linear learner
At human-level or above
Below human-level
0
100
200
300
400
500
600
1,000
4,500
Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%

Other Results

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

AlphaGo, AlphaGo Zero, Alpha Zero

- October 2015: first version of AlphaGo beats European Go Champion Fan Hui 5:0

- March 2016: AlphaGo beats World Go Champion, Lee Sedol AlphaGo was pre-trained on a huge database of historical games, and then improved by playing against itself; a few TPUs

- 19 October 2017: AlphaGo Zero, trained solely on self-played games surpasses AlphaGo, beating it 100:0, hardware costs estimated on $25 million

- December 5, 2017: Alpha Zero announced: a generic architecture learning to play Go, Chess, Shogi in hours (4 hours to learn Chess on a superhuman level); hardware: 5000 TPUs

What NEXT?

Key Ideas behind AlphaGo Zero

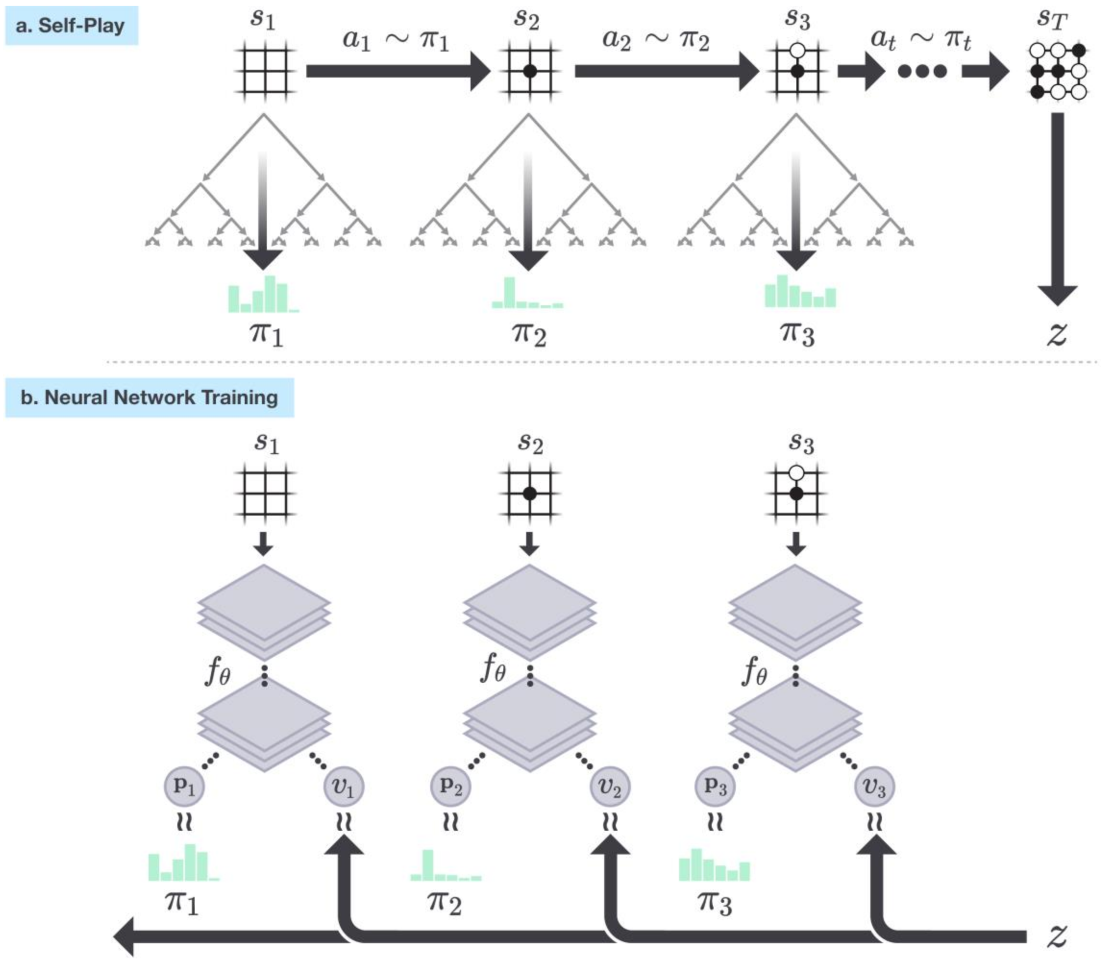
- trained solely by self-play generated data (no human knowledge!)

- use a single Convolutional ResNet with two “heads” that model policy and value estimates:
  - policy = probability distribution over all possible next moves
  - value = probability of winning from the current position

- extensive use of Monte Carlo Tree Search to get “better estimates”

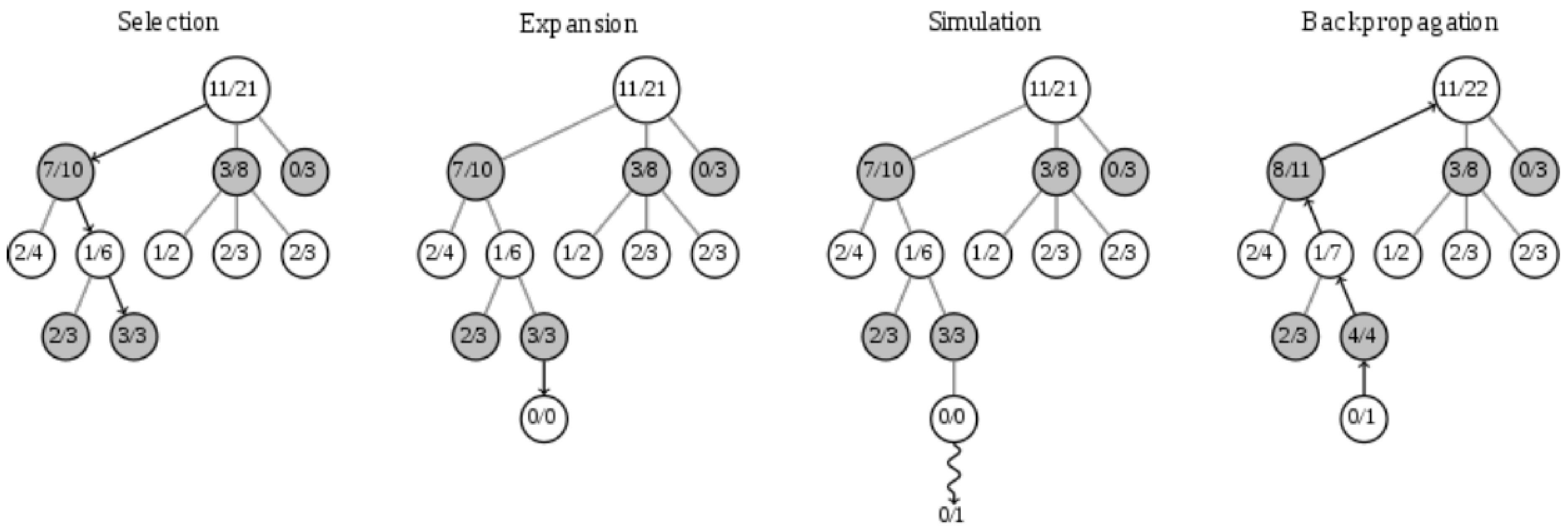
- a tournament to select the best network to generate fresh training data

AlphaGo Zero: Self-Play and the Loss Function



$$(\mathbf{p}, v) = f_\theta(s), \qquad l = (z - v)^2 - \boldsymbol{\pi}^\top \log \mathbf{p} + c||\theta||^2$$

MCTS


Selection
Expansion
Simulation
Backpropagation
11/21
7/10
3/8
0/3
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
0/1
11/22
8/11
1/7
4/4

MCTS

Monte Carlo Tree Search (MCTS)

Upper Confidence Bounds for Trees (UCT)

- UCT = MCTS + UCB [*]
- Selecting a child node c which

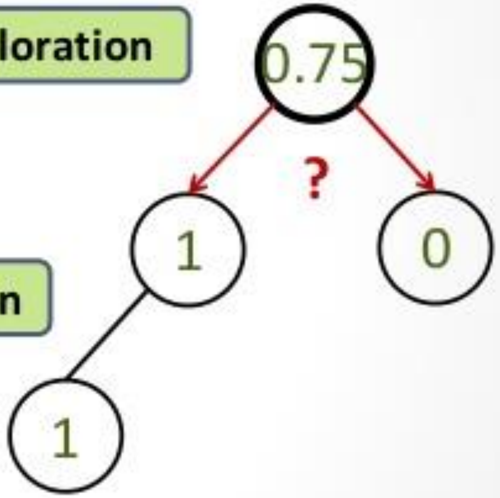
$$c \in \operatorname{argmax}_{i \in I}\left(v_i + C_p \times \sqrt{\frac{\ln n_p}{n_i}}\right)$$

Exploration

Exploitation

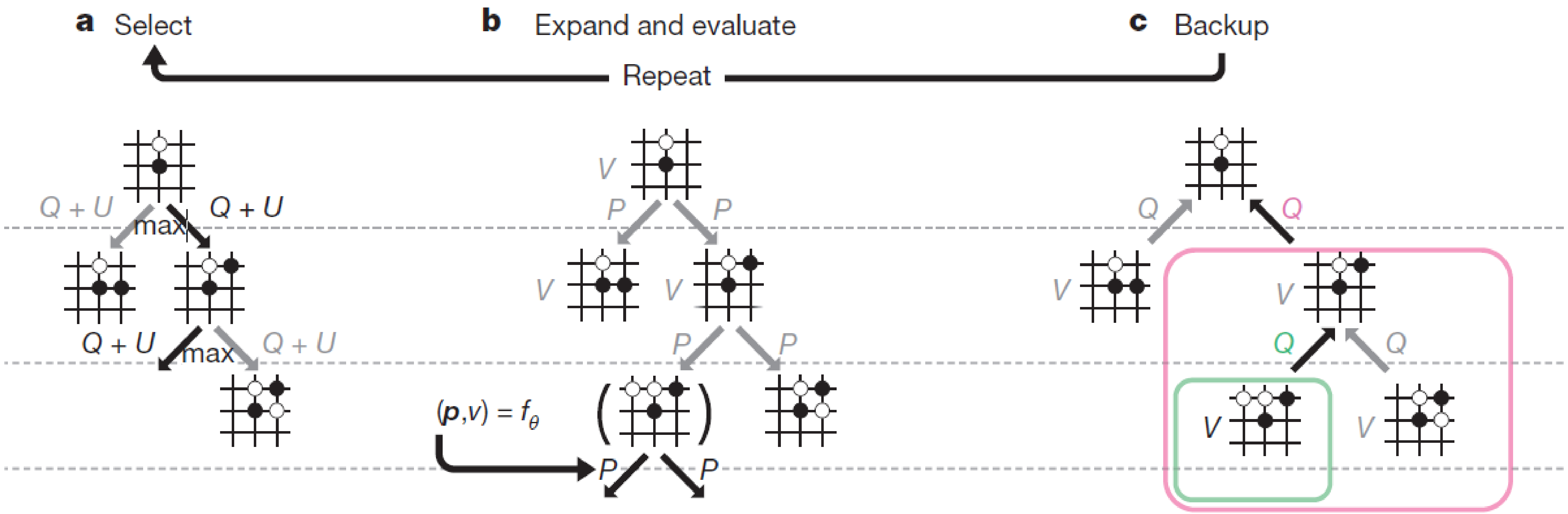
  - p : c's parent node
  - I : the set of p's children
  - v_i : i's approximate utility
  - n_i : i's visit count
  - n_p : p's visit count
  - C_p : a tunable constant

- Example : 4th loop



[*] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning.
In Proceedings of the 17th European Conference on Machine Learning, 2006.

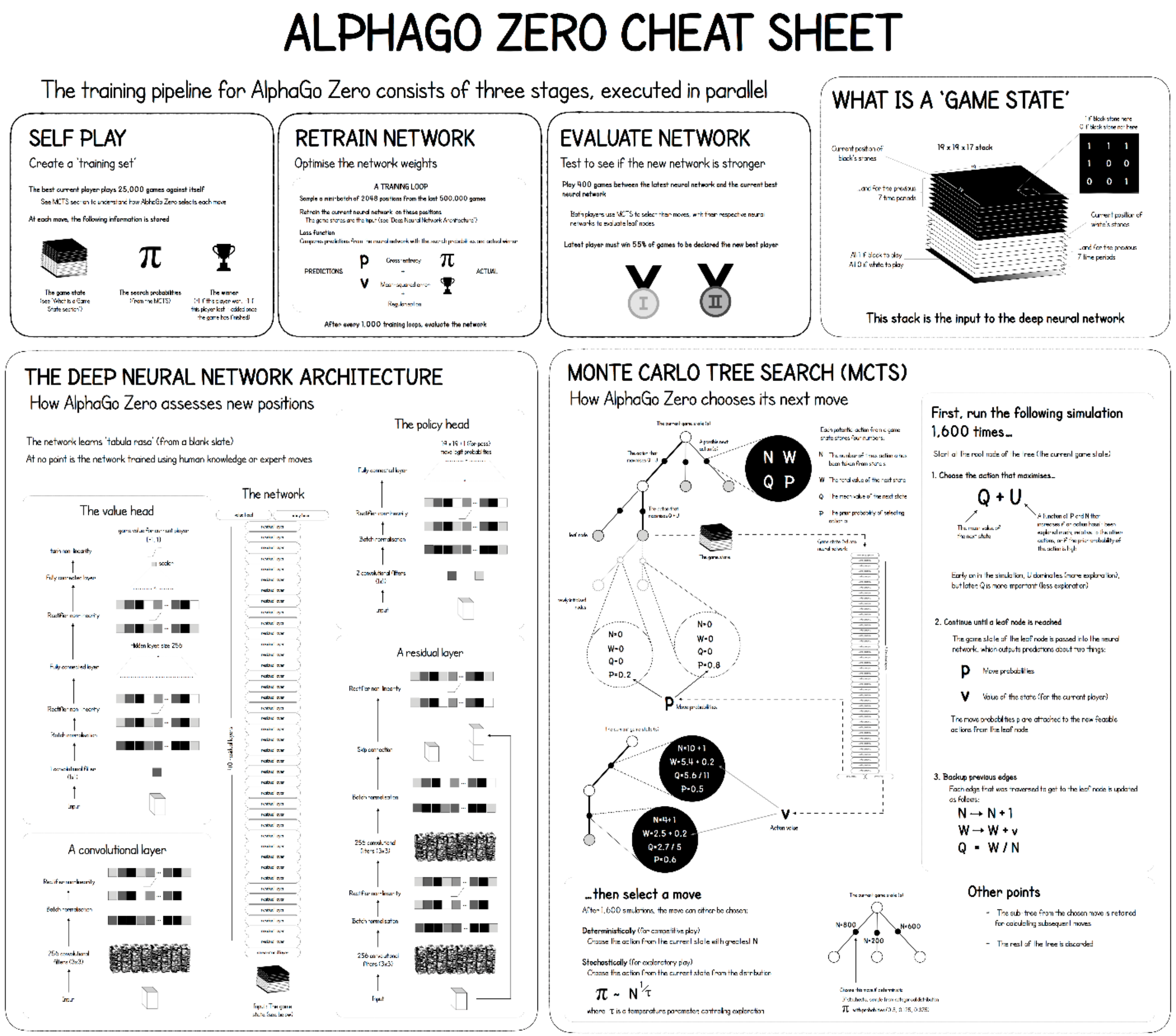
MCTS



a Select
b Expand and evaluate
c Backup
Repeat
Q + U
max
Q + U
Q + U
max
Q + U
V
P
P
V
V
P
P
(p,v) = fθ
P
P
Q
Q
V
V
Q
Q
V
V

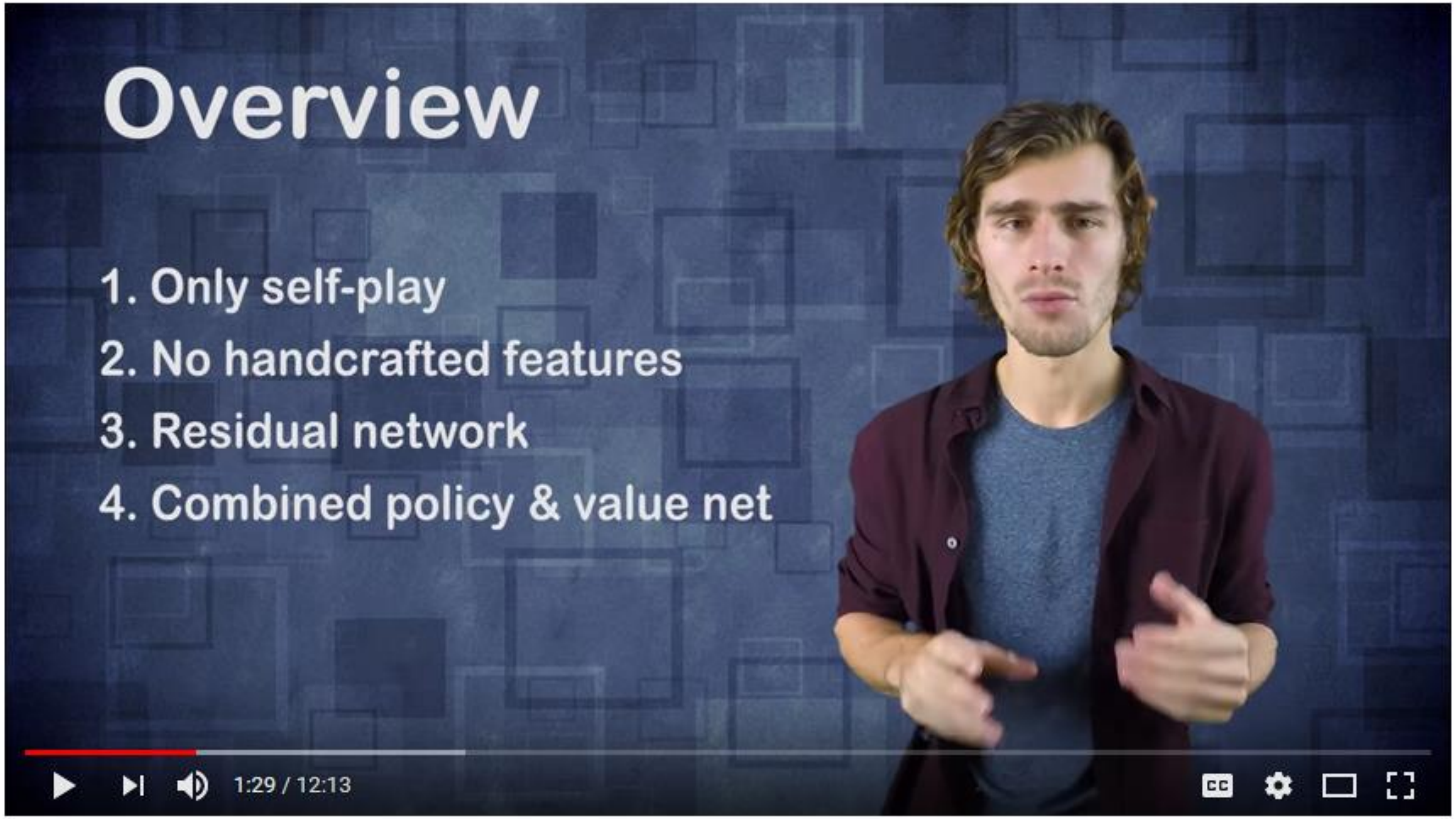
Details

applied-data.science/static/main/res/alpha_go_zero_cheat_sheet.png



Summary:

https://www.youtube.com/watch?v=MgowR4pq3e8



How AlphaGo Zero works - Google DeepMind