Applications of Reinforcement Learning

Ist künstliche Intelligenz gefährlich?

Table of contents

Playing Atari with Deep Reinforcement Learning

Motivation

- most successful RL applications
  - handcrafted features
  - linear value function or policy representation

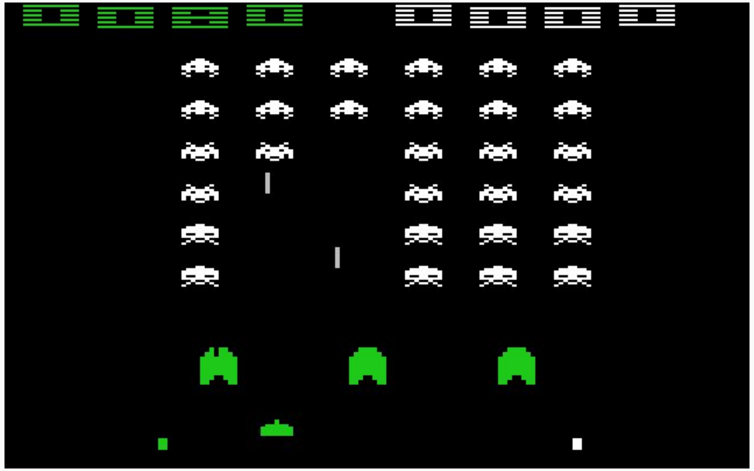
➜ performance relies on quality of features

- advances in deep learning
  - high-level features from raw sensory data

➜ **Deep Reinforcement Learning**

Goal

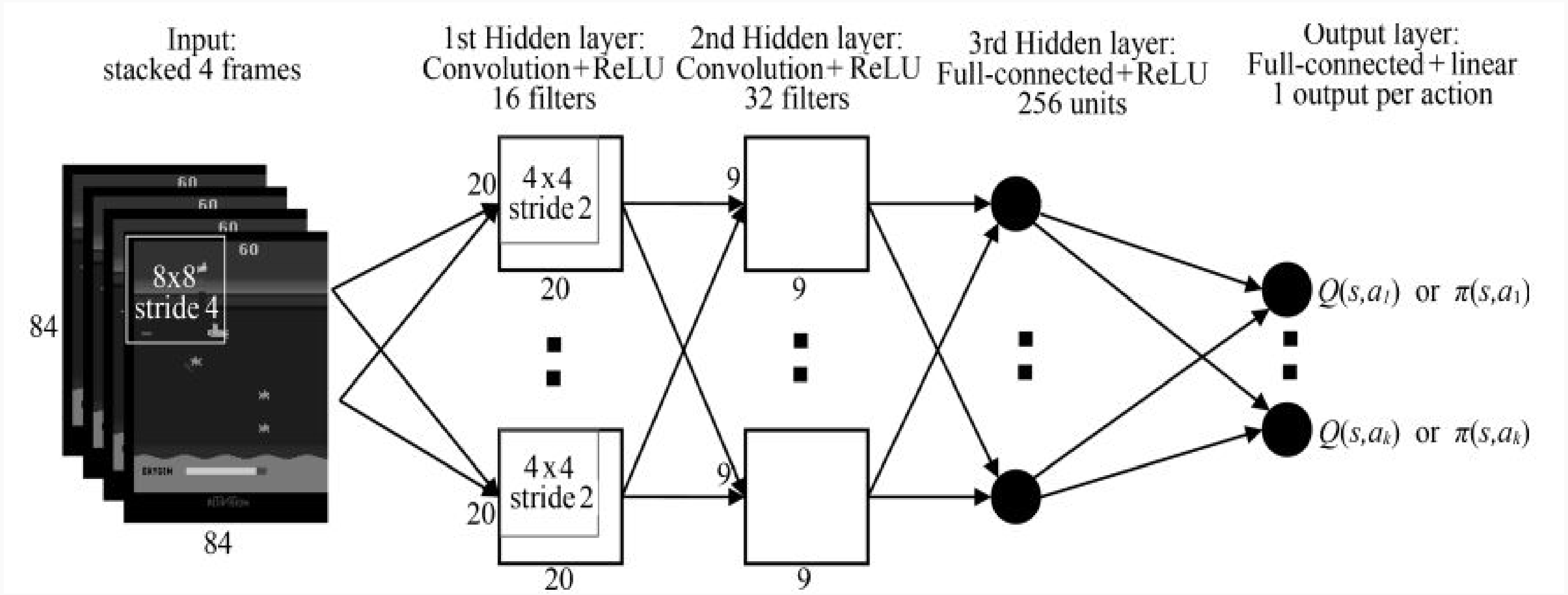
- play Atari with only raw pixels as input
- reward: game score
  - can be delayed
- connect RL algorithm to deep CNN
  - directly working on RGB images
  - downsampled
  - grayscale



Experience replay and preprocessing

- replay memory
  - 1 million most recent frames
- one experience
  - $(\phi_j, a_j, r_j, \phi_{j+1})$
- preprocessing function $\phi(s)$
  - stacks history of 4 images
  - crops 84x84 region of image
- initialise $s_1 = \{ x_1 \}$ and $\phi_1 = \phi(s_1)$

Network Architecture



Experience generation

- every k-th frame
  - with probability **ε** select random action a_t
    - **ε** = 1
    - anneals to 0.1
  - otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
  - execute a_t, observe reward r_t and image x_{t+1}
  - set $s_{t+1} = s_t, a_t, x_{t+1}$
  - preprocess $\phi_{t+1} = \phi(s_{t+1})$
  - store transition $(\phi_j, a_j, r_j, \phi_{j+1})$ in replay memory

Deep Q-learning

- sample random experience $(\phi_j, a_j, r_j, \phi_{j+1})$
- set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
- perform gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$

Super Mario World

Goal

- first level of Super Mario World
- deep Q-learning with replay memory and spatial transformer
- emulator: lsnes (using LUA API)
- neural net framework: Torch
- training on random parts of the level

Inputs and Outputs

- inputs:
  - last 4 actions as two-hot-vectors (A/B/X/Y and an arrow button)
  - last 4 screenshots, downscaled to 32x32 (grayscale, slightly cropped)
  - current screenshot, 64x64 (grayscale, slightly cropped)

- state captured every 5th frame ➜ 12 times per second
- replay memory size: 250.000 entries

- output:
  - Q-Values for every action in current state (8-dimensional vector)
  - choosing highest button and arrow value

Rewards

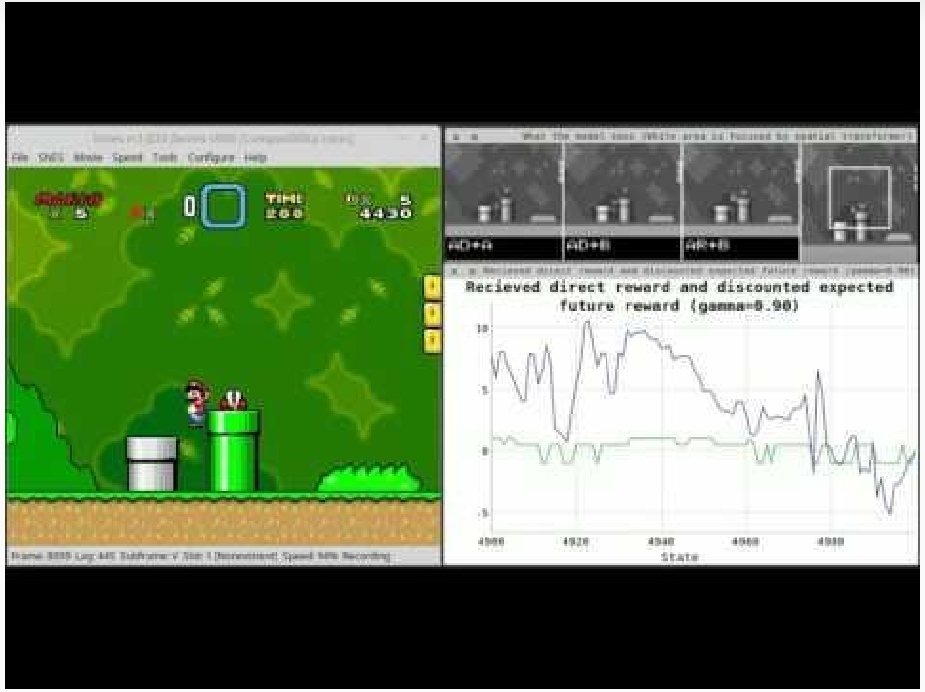
- +0.5 moving **right**
- +1.0 moving **fast right** (≥8 pixels)
- −1.0 moving **left**
- −1.5 moving **fast left** (≥8 pixels)
- +2.0 during **level-finished-animation**
- −3.0 during **death animation**

- Discount for future rewards: $\gamma = 0.9$

Policy

- **ε**-greedy policy
    - start: **ε** = 0.8
    - decreases to 0.1 over 400.000 actions
- random action: coin flip
    - randomize one out of two actions
    - randomize both actions

Demonstration


File SNES Movie Speed Tools Configure Help
TIME 288
4430
AD+A
AD+B
AR+B
Recieved direct reward and discounted expected future reward (gamma=0.90)
10
5
0
-5
4900
4920
4940
4960
4980
State

Source: https://youtu.be/L4KBBAwF_bE

Stanford University Autonomous Helicopter

Motivation

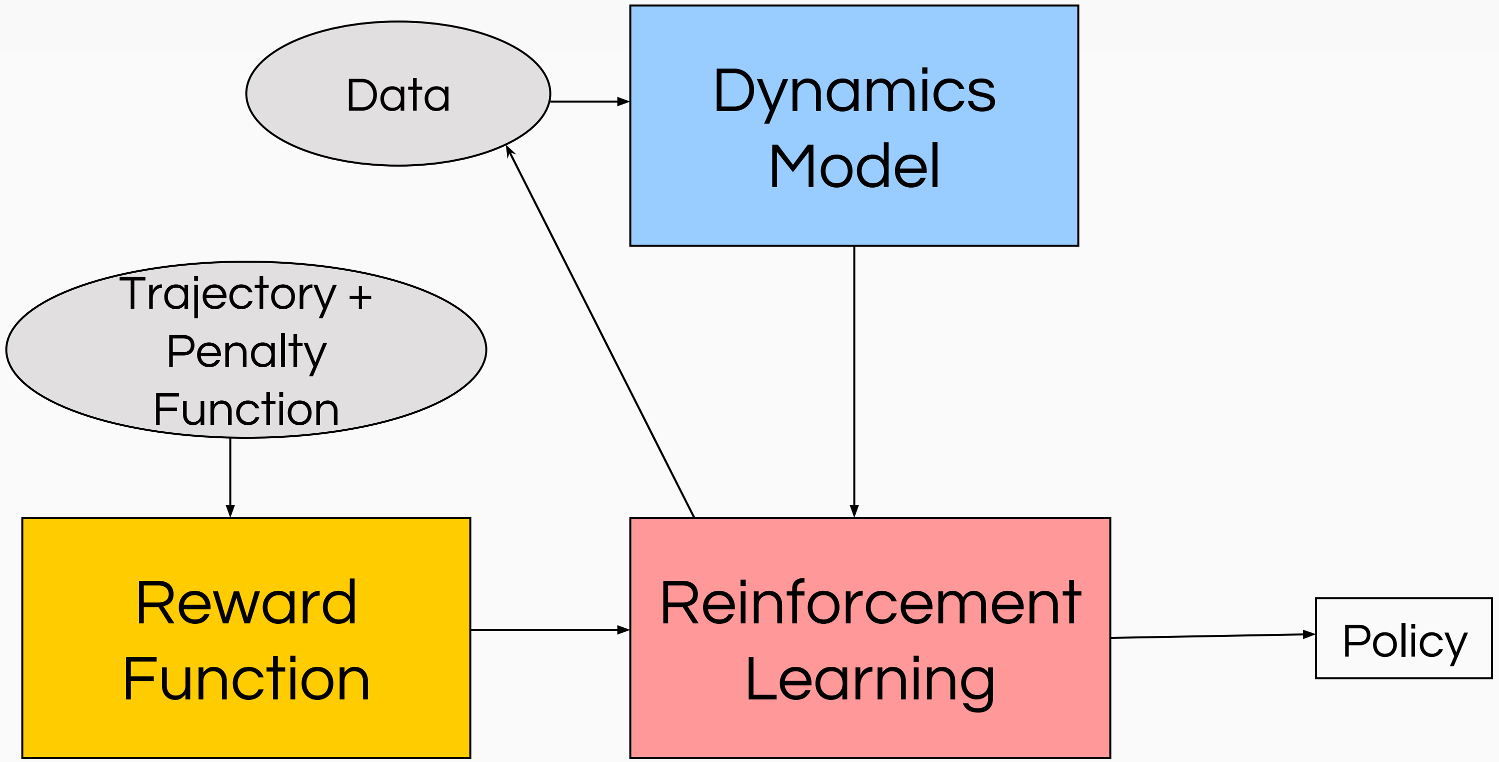
- challenging control problem
  - complex dynamics model
- exploration can cause crashes
  - expensive

➜ **Apprenticeship learning**

What is needed to fly autonomous?

- trajectory
  - desired path for the helicopter to follow
  - hand-coded
- dynamics model
  - learned from flying data
  - input: current state and controls
  - output: prediction where helicopter will be
- controller
  - feeds controls to fly trajectory
  - policy

Overview


Data
Dynamics Model
Trajectory + Penalty Function
Reward Function
Reinforcement Learning
Policy

Algorithm

1. start with an example flight
2. compute a **dynamics model** and **reward function** based on the target trajectory and sample flight
3. find a **controller** (policy) that maximizes this reward
4. fly the helicopter with the current controller and **add this data** to the **sample flight data**
5. if we flew the target trajectory stop, otherwise go to step 2

Problems

- quick learning
- only simple maneuvers
- can’t hand-code **complex** trajectories
  - should obey system dynamics
  - unable to explain how task is performed

➜ **Apprenticeship learning of trajectory**

Learning the trajectory

- multiple demonstrations of the same maneuver

$$y_j^k = \begin{bmatrix} s_j^k \\ u_j^k \end{bmatrix}, \text{ for } j = 0..N^k - 1, k = 0..M - 1$$

  - s: sequence of states
  - u: control inputs

- goal: find “hidden” target trajectory of length T

$$z_t = \begin{bmatrix} s_t^\star \\ u_t^\star \end{bmatrix}, \text{ for } t = 0..T - 1$$

Graphical Model

- intended trajectory

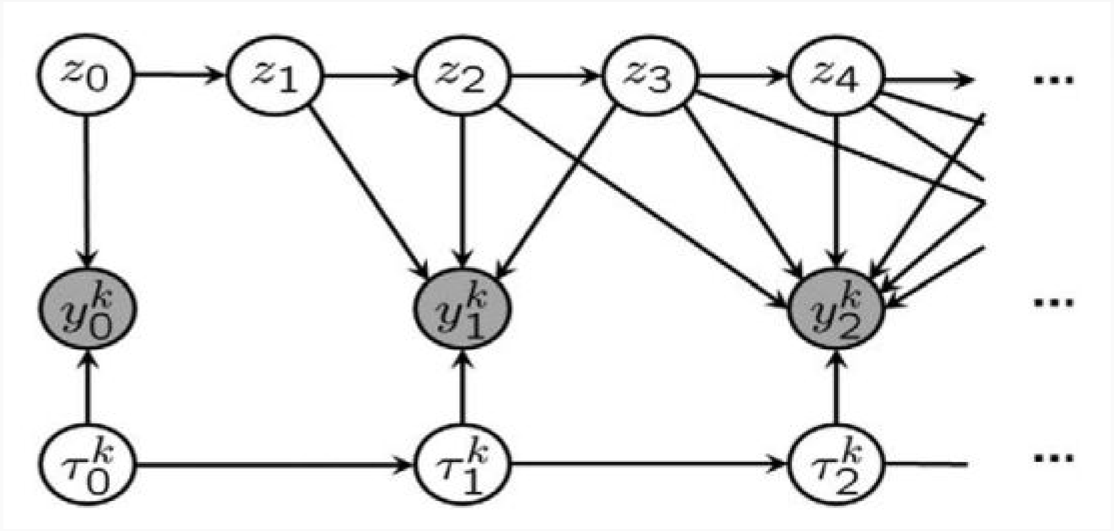
$$z_{t+1} \quad = \quad f(z_t) + \omega_t$$

- expert demonstration

$$y_j \quad = \quad z_{\tau_j} + \nu_j$$

- time indices

$$\tau_j^k \quad \sim \quad \mathbb{P}(\tau_{j+1}^k | \tau_j^k).$$

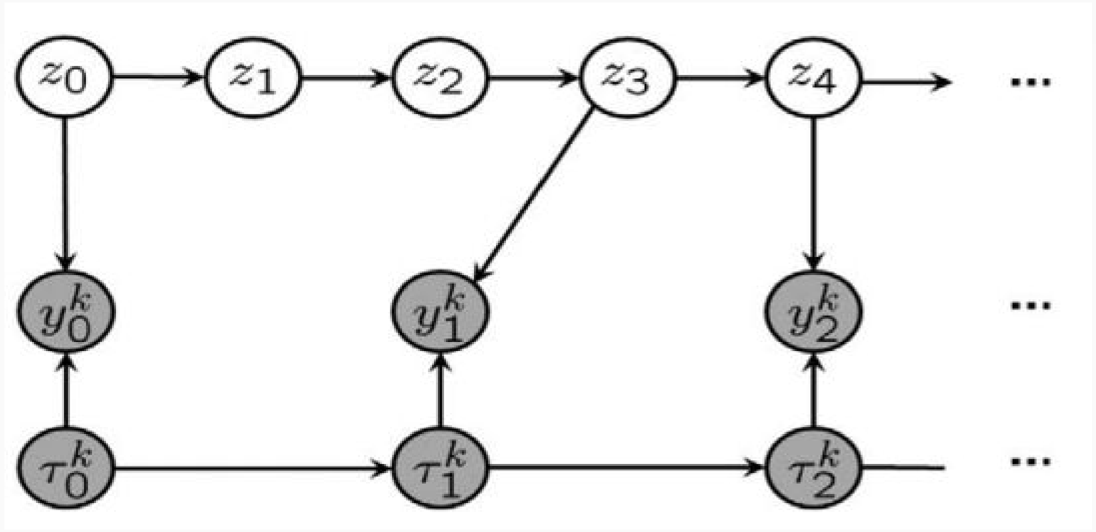


- intended trajectory satisfies dynamics, but **τ** unknown

Learning Algorithm

- unknown **τ**
    - inference is hard

- known **τ**
    - standard HMM



Algorithm

- make initial guess for **τ**
- alternate between:
    - fix **τ** , run Baum-Welch algorithm on resulting HMM
    - choose new **τ** using dynamic time warping

Further adjustments

- time varying dynamics model

$$z_{t+1} = f_t(z_t) + \omega_t^{(z)} \equiv f(z_t) + \beta_t^{\star} + \omega_t^{(z)}$$

  - f: crude model
  - β: difference between crude estimation and target
  - w: gaussian noise

- incorporation of prior knowledge
  - loops on plane in space
  - flips with center fixed

Demonstration

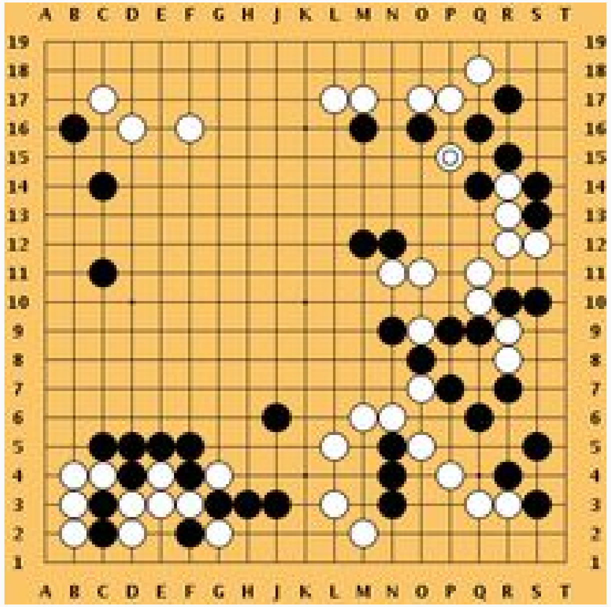
Source: https://youtu.be/VCdxqn0fcnE

AlphaGo

Motivation

- Go
  - 19x19 board
  - goal: dominate the board
  - surrounded area
  - captured stones
  - 4.6x10^{70} possible states

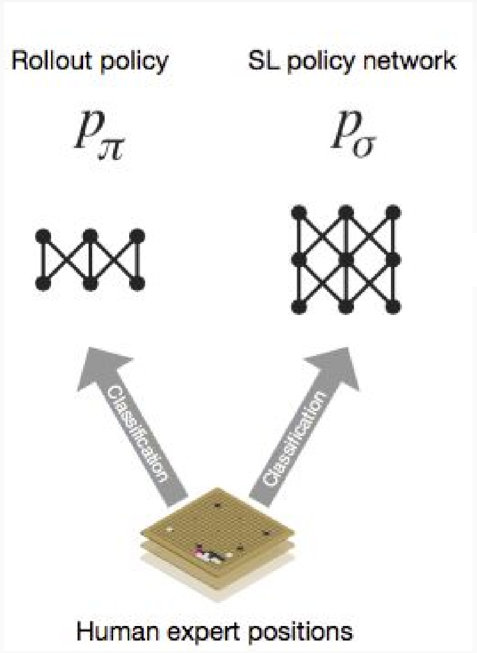
- previous AIs: amateur level



First stage

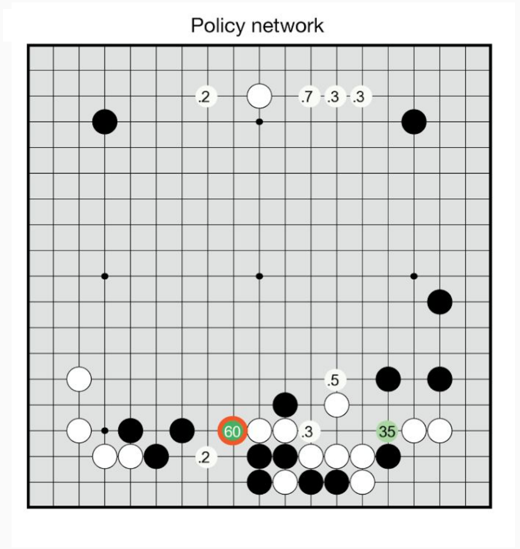
- Supervised Learning Policy Network p_σ
  - input: board state s
  - output: distribution over legal moves
  - 30 million positions
  - 57% accuracy
  - 3 ms

- Fast Rollout Policy Network $p_{\square}$
  - faster
  - 24% accuracy
  - 2 µs



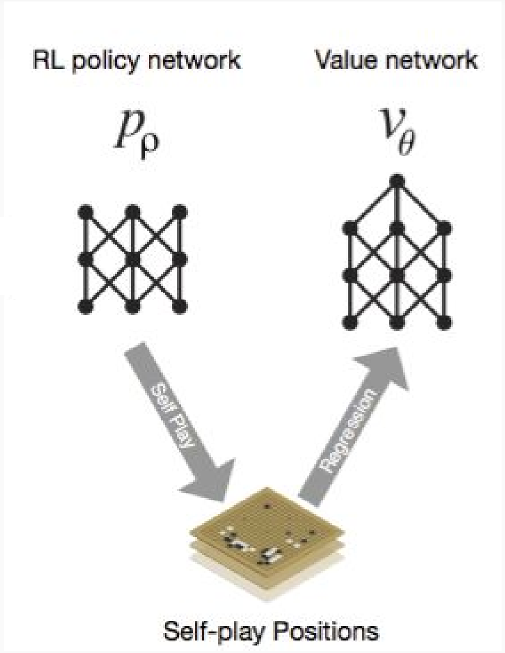
Second stage

- Reinforcement Learning Policy Network p_ρ
  - initialised with weights of p_σ
  - plays against random previous iterations
  - rewards:
    - +1 win
    - −1 lost
    - 0 else

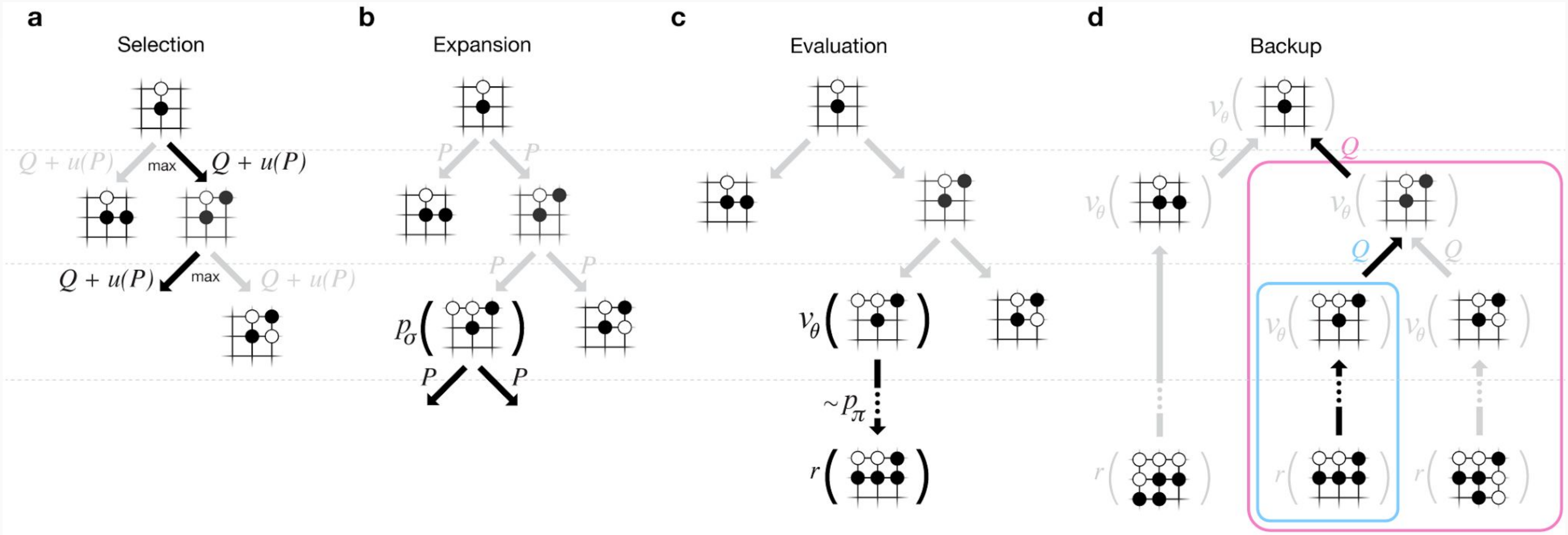


Third stage

- Value Network v_θ
  - value function for strongest policy $v^p(s)$
  - predicts outcome from position s
  - outputs single prediction
  - 30 million games of self-play as input



Monte Carlo Tree Search

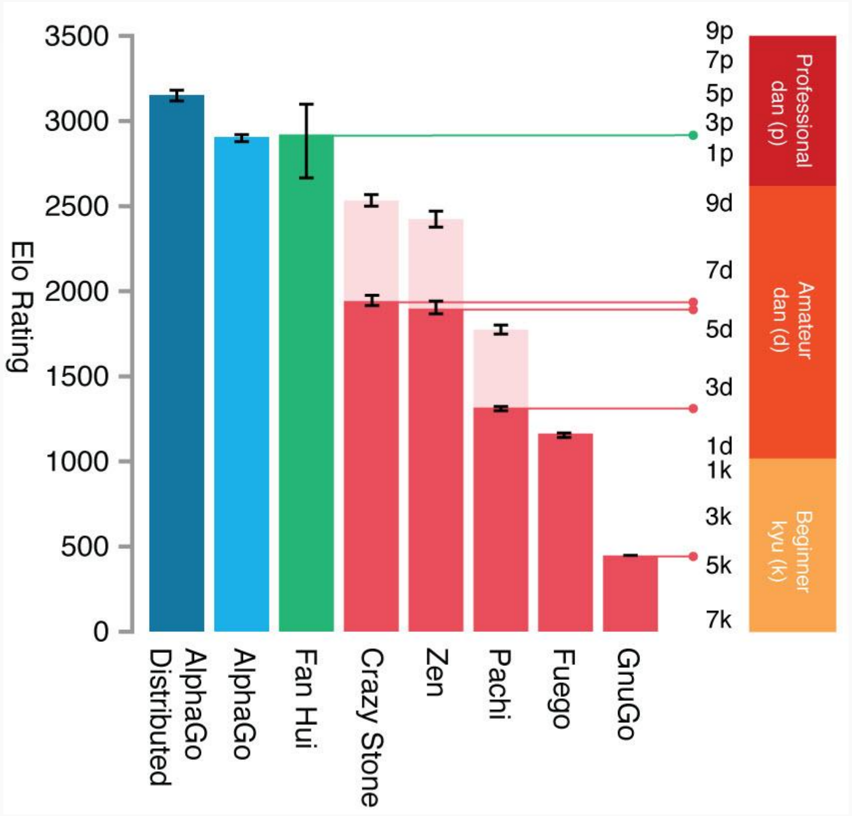


$$V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L$$

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

Summary

- tournament against other AIs
  - 5 seconds per turn
  - 99.8% winrate overall
- handicapped games (4 stones)
  - 77% against Crazy Stone
  - 86% against Zen
  - 99% against Pachi
- AlphaGo distributed
  - 77% against single machine
  - 100% against other AIs
- 5:0 against Fan Hui
- 4:1 against Lee Sedol



Thanks for your attention!

Sources

Atari
https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
https://wiki.tum.de/display/lfdv/Convolutional+Neural+Network+for+Game

Mario
https://github.com/aleju/mario-ai

Helicopter
http://cs.stanford.edu/groups/helicopter/papers/nips06-aerobatichelicopter.pdf
https://people.eecs.berkeley.edu/~pabbeel/papers/AbbeelCoatesNg_IJRR2010.pdf

AlphaGo
https://gogameguru.com/i/2016/03/deepmind-mastering-go.pdf
https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/