LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Praktikum Autonome Systeme

History & Applications I

Prof. Dr. Claudia Linnhoff-Popien
Thomy Phan, Fabian Ritz, Andreas Sedlmeier
http://www.mobile.ifi.lmu.de

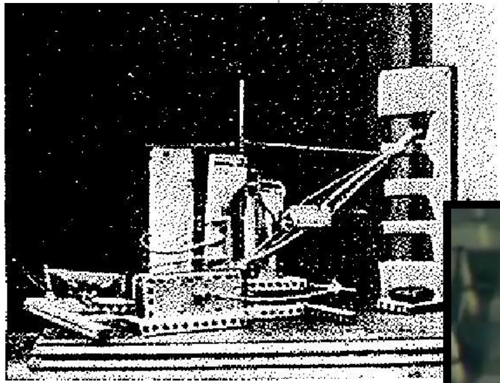
WiSe20

History of Learning Machines

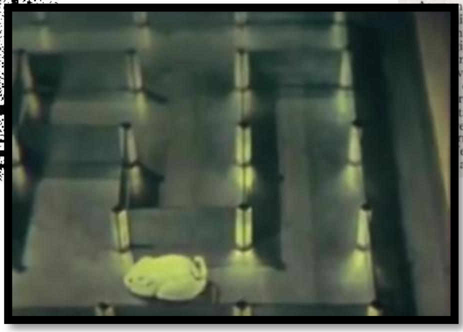
Electro-mechanical learning machines

Maze Solving Machines

http://cyberneticzoo.com/

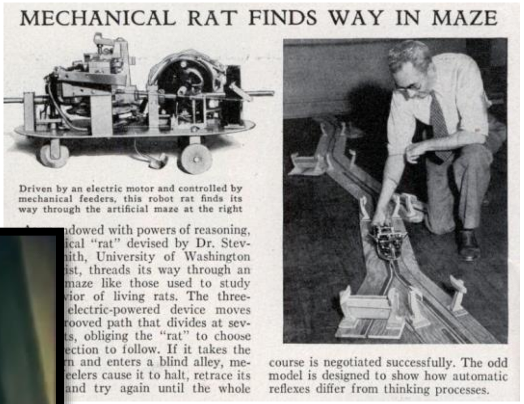


Thomas Ross (1933)



Claude Shannon (1952)

http://cyberneticzoo.com/



MECHANICAL RAT FINDS WAY IN MAZE

Driven by an electric motor and controlled by mechanical feeders, this robot rat finds its way through the artificial maze at the right

ndowed with powers of reasoning, ical "rat" devised by Dr. Stev- ith, University of Washington ist, threads its way through an maze like those used to study vior of living rats. The three- electric-powered device moves rooved path that divides at sev- ts, obliging the "rat" to choose ection to follow. If it takes the n and enters a blind alley, me- eelers cause it to halt, retrace its and try again until the whole course is negotiated successfully. The odd model is designed to show how automatic reflexes differ from thinking processes.

Stevenson Smith (1935)

- Inspired by learning theories like Thorndike‘s „Law of Effect“ (1911) or „reinforcement“ ideas à la Pavlov(1927) / Skinner (1938)
- Alan Turing: Design for a „pleasure-pain system“ (1948)
- → trial-and-error learning

Electro-mechanical learning machines

Claude Shannon „Theseus“ (1952)

Demonstration of Early Machine Learning with "Theseus"
By: Claude Shannon
"Father of Information Theory" AT&T Bell Labs

https://www.youtube.com/watch?v=_9_AEVQ_p74

Programming Digital Computers

- Farley and Clark (1954): Digital Simulation of a neural-network learning machine
- 60s & 70s: Confusion of the Terms:
  Reinforcement Learning / Supervised Learning

„Modern" Definition of trial-and-error learning:
Selecting actions on the basis of evaluative feedback that does not rely on knowledge of what the correct action should be.
- Sutton 2018

- Marvin Minsky „Steps Toward Artificial Intelligence" (1961):
  Discusses issues in trial-and-error learning:
  prediction, expectation, credit-assignment problem

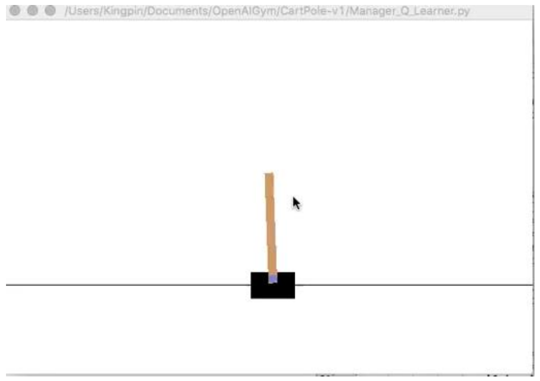
www.kurzweilai.net

First Applications to „Games“

- (1950) Shannon: Suggests that a computer could learn to play chess by using an evaluation function (and modify it online). Also this function need not be exact.
- (1959, 1967): Checkers player (using function approximation for learning value function)
- (1961) Michie: tic-tac-toe learning system

First Applications to „Games“

- (1968) Michie & Chamber: RL Controller for pole balancing
- „Cartpole“ still relevant today as classic RL benchmark task: e.g openai-gym:


/Users/Kingpin/Documents/OpenAIGym/CartPole-v1/Manager_Q_Learner.py

First Applications to „Games“

- (NeurIPS 2019) Freeman, Metz, Ha: Learning to Predict Without Looking Ahead: World Models Without Forward Prediction

  Research Question: Does an RL agent learn that it can be useful to predict the future?

https://learningtopredict.github.io/

First Applications to „Games“

- (1960s) Tsetlin: Learning automata: k-armed bandits

- (1973) Widrow, Gupta, Maitra: Modified Least-Mean-Square algorithm, can learn from success and failure signals: „learning with a critic“: i.e. critic evaluates performance
  → Allows learning Blackjack

First Applications in Economics & Game Theory

- (1973) Bush: Apply learning theory to economic models
- (1991) Arthur: Study artificial agents that behave more like real humans
- (1991-) RL in the context of game theory, multi-agent systems

Temporal Difference Learning

- (1983) Barto, Sutton: Actor-Critic architecture applied to pole-balancing
- (1986) Anderson: Extension to backprop in neural networks
- (1989) Watkins: Q-learning
- (1992) Tesauro: TD-Gammon (backgammon)

Applications of Reinforcement Learning

Representation issues […] are so often critical to successful applications
–Sutton 2018

Applications of reinforcement learning are still far from routine and typically require as much art as science. Making applications easier and more straightforward is one of the goals of current research in reinforcement learning.

–Sutton 2018

Samuel‘s Checkers Player (1955)

- **Lookahead search** from the current position
- **Heuristic search** methods to **decide how to expand the search tree** and when to stop
- Shannon‘s **MiniMax procedure** to find the best move:
  - Each position is given the value of the position that would result from the best move
    Assumption: Program trys to maximize the score, oponent tries to minimize it

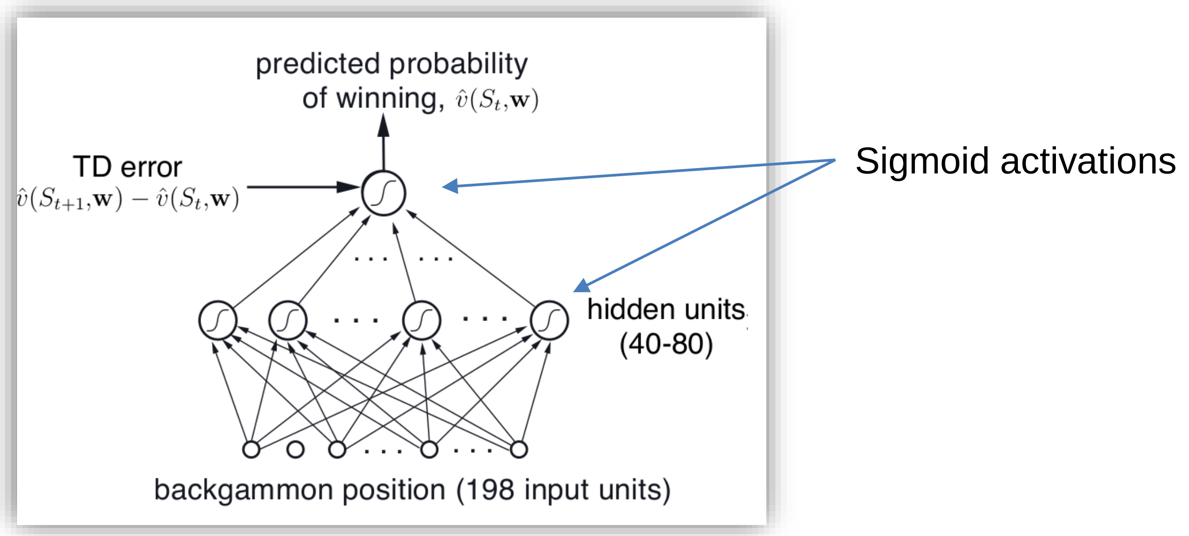
- Terminal positions scored by a **value function** using **linear function approximation**
- No explicit rewards, instead measure the number of pieces relative to the opponent

TD-Gammon (1992)

- Algorithm: **TD-λ**
- Combined with **non-linear Function Approximation**:
  - Multilayer Neural Network
  - Trained by backpropagating TD errors
- Large state space:
  - 30 pieces and 24 possible locations
    → Lookup table not possible
- Large number of possible moves from each position
- Dice rolls & possible opponent moves lead
  to an effective branching factor of the game tree of: ~400
- Highly stochastic (dice) + Opponent actions can be seen as uncertainty
- Fully observable

TD-Gammon

- Rewards: always 0, except on steps when the game is won
- DNN predicting the probability of winning: $\hat{v}(S_t)$


predicted probability of winning, $\hat{v}(S_t,\mathbf{w})$
TD error $\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})$
Sigmoid activations
hidden units (40-80)
backgammon position (198 input units)

Sutton (2018)

- State Encoding: 4 Units for each point on the board = 192
- +2 Units: #black / white pieces on the bar
- +2 Units: #black / white pieces already removed
- +2 Units showing who‘s turn it is (black or white)

TD-Gammon

- Training Data: **Self-Play**
  → TD-Gammon plays against itself
- Compute value of all possible dice roll + state outcomes
- Choose the move that leads to the state with the highest estimated value
- After ~300.000 games, beats the previous best Backgammon program

→ TD-Gammon 0.0: No expert knowledge used

- TD-Gammon 1.0: Uses special Backgammon features
  Level of best human players
- TD-Gammon 2.0-3.0: Further improvements, combination with MCTS

- TD-Gammon influenced the way, the best players now play Backgammon

IBM WATSON Jeopardy! (2011)

https://spectrum.ieee.org

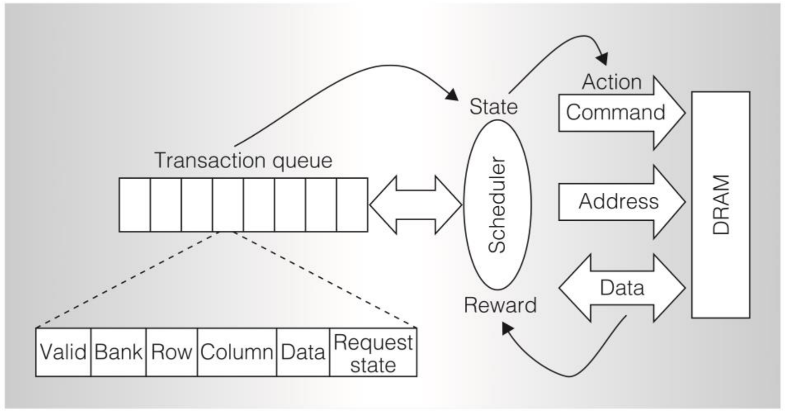
- **Natural Language processing** etc.
  but also **RL components**:
- Decision making: **Adaption of TD-Gammon system** (TD-λ + NN)
  - Trained offline on many simulated runs
  - Calculation of action values $\hat{q}(s, bet)$ (Probability of a win)
    Action values are a combination of 2 value functions:
    - State value function
      + Confidence of a correct answer in the category
  - Choose maximum action value
- **No Self-Play** (Watson's style was too different to humans)
  → Self-Play would have explored the state space at regions not typical in human matches
- Instead: **Expert Models used as opponents** in training
- Only End-Games used MC-trials, because of required speed

mobile and distributed systems group

Reinforcement Learning Memory Controller for DRAM Access (2008)

Ipek, Engin, et al. "Self-optimizing memory controllers: A reinforcement learning approach."
ACM SIGARCH Computer Architecture News. Vol. 36. No. 3. IEEE Computer Society, 2008.

- Computer Memory Controller solves a scheduling problem
- RL Memory Controller improves speed of program execution
- Previous state-of-the-art controllers used policies that did not take advantage of **past scheduling experience** and did not account for long-term consequences of scheduling decisions



Sutton (2018)

- Scheduler is the reinforcement learning agent
- Environment is represented by features of the transaction queue
- Actions are commands to the DRAM system

Reinforcement Learning Memory Controller for DRAM Access (2008)

- **SARSA** to learn an action-value function
- State: 6 integer-valued features
- Linear function approximation
- Exploration: ε-greedy (0.05)
- Optimized for on-chip implementation

Human-Level Video Game Play (2013)

DQN

https://www.youtube.com/watch?v=xN1d3qHMIEQ

Human-Level Video Game Play (2013)

- Previous Examples: Network-Input are Hand-crafted features!
- Google DeepMind:

  Deep Multi-Layer NN can automate the feature design process
- DQN (Deep Q-Network): Q-Learning combined with convolutional NN
- Learned to play 49 different ATARI games
- Different policy for each game

Human-Level Video Game Play (2013)

- Same raw input, architecture, parameters for all games
- Experience replay
- Reduced ‚resolution‘ from 210x160x3 (RGB) to 84x84x1 (b/w)
- Full state is not observable from single frame

  → stack 4 most recent frames → Input: 84x84x4
- 4-18 discrete actions
- Reward: Game score increased = +1, decrease = -1, else 0
- ε-greedy: Decreasing linearly over 1e6 frames

Human-Level Video Game Play (2013)

- Modifications to standard Q-Learning:
  - Experience replay
  - Target-Network as clone every N time-steps of the Online-Network
  - Error-Clipping: Constrain error term
    $R_{t+1} + \gamma \max_a \tilde{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$
    to [-1,1], which increases stability

- Not all games ‚solved' – e.g. Montezuma's Revenge:
  Only skill of a random player

→ Exploration is still a challenge for DQN

Mastering the Game of Go: AlphaGo (2016)

https://www.youtube.com/watch?v=8tq1C8spV_g

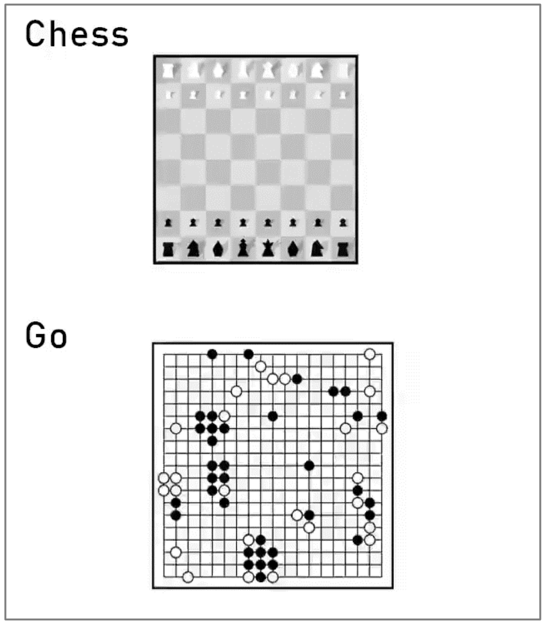
Mastering the Game of Go: AlphaGo (2016)

- Previously, no Go program could play comparable to Go masters
- AlphaGo: Combines Deep NNs, Supervised Learning, MCTS & RL
- Won 4/5 matches against 18x world champion Lee Sedol

- Like TD-Gammon:
  RL over simulated games of Self-Play

Mastering the Game of Go: AlphaGo (2016)

- Like TD-Gammon:
  RL over simulated games of Self-Play
- Go Search Space: A lot larger than chess
  - Legal moves (Go: ~250 / Chess: ~ 35)
  - More moves per game (Go: ~150 / Chess: ~80)



Why is AlphaGo stronger than classical MCTS?

- AlphaGo‘s MCTS is guided by both policy & value function learned via RL
- Function approximation: Deep Convolutional NN
- 2-Stage Training-Process:
  Policy-Network-Weights not started randomly but results of previous supervised learning from human expert moves

Mastering the Game of Go: AlphaGo (2016)

What action to explore next? i.e. Tree Expansion:

- MCTS: expand search tree based on the stored action-values
- AlphaGo: expand based on probabilities predicted by SL-policy network (which is trained to predict human expert moves)

How to evaluate the new state node?

- **MCTS:** Return of a rollout initiated from it
- **AlphaGo:** Return of the rollout & value-function learned previously via RL

$$v(s) = (1 - \eta)v_\theta(s) + \eta G,$$

G: Return of the rollout
η : Mixing Parameter (Best performance with 0.5)

Mastering the Game of Go: AlphaGo (2016)

How to choose actions in the rollout?

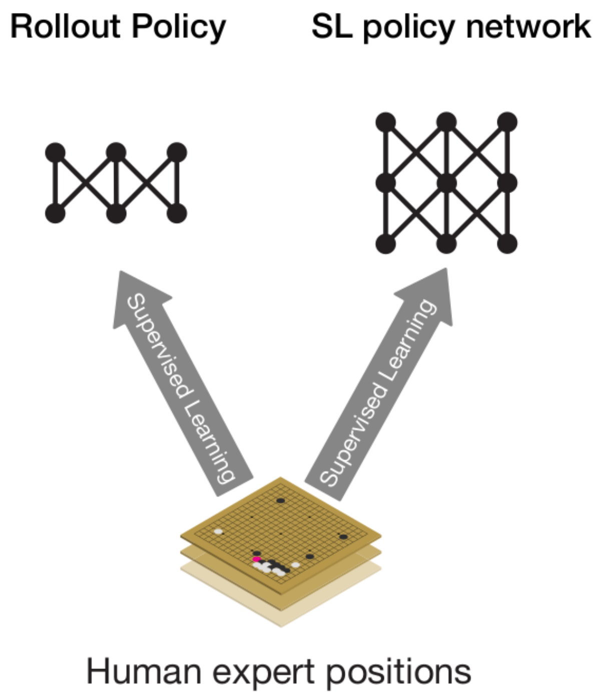
– **AlphaGo:** Fast rollout policy
  (simple linear network: pre-trained supervised from human moves)

How to choose the action to really take after the MCTS simulations?

– **AlphaGo:**
  - Keep track how many simulations choose each action from the current state
  - Choose the action most often taken in simulation

Mastering the Game of Go: AlphaGo (2016)

How are the AlphaGo networks trained?

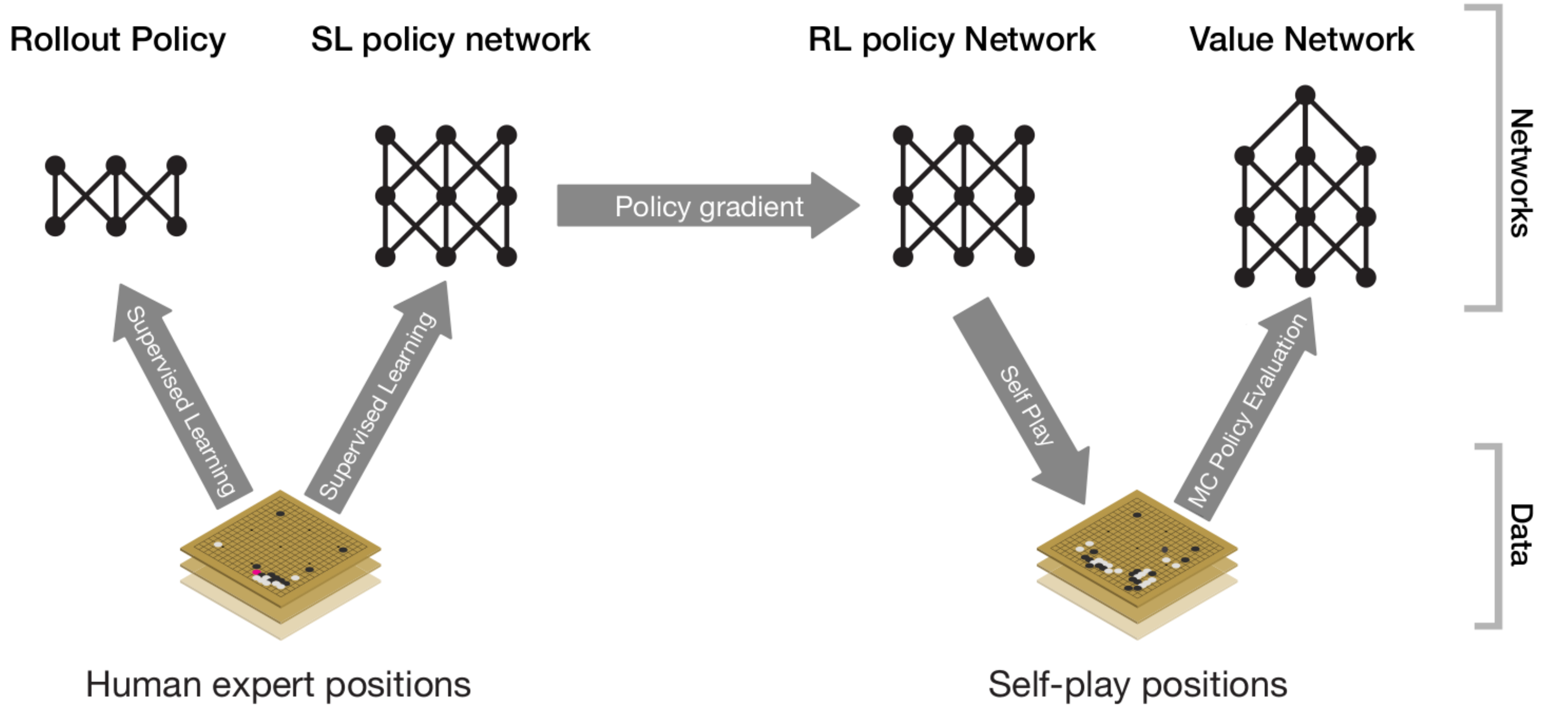


→ „Imitate Human Players“

https://www.nature.com/articles/nature16961

Mastering the Game of Go: AlphaGo (2016)

How are the AlphaGo networks trained?

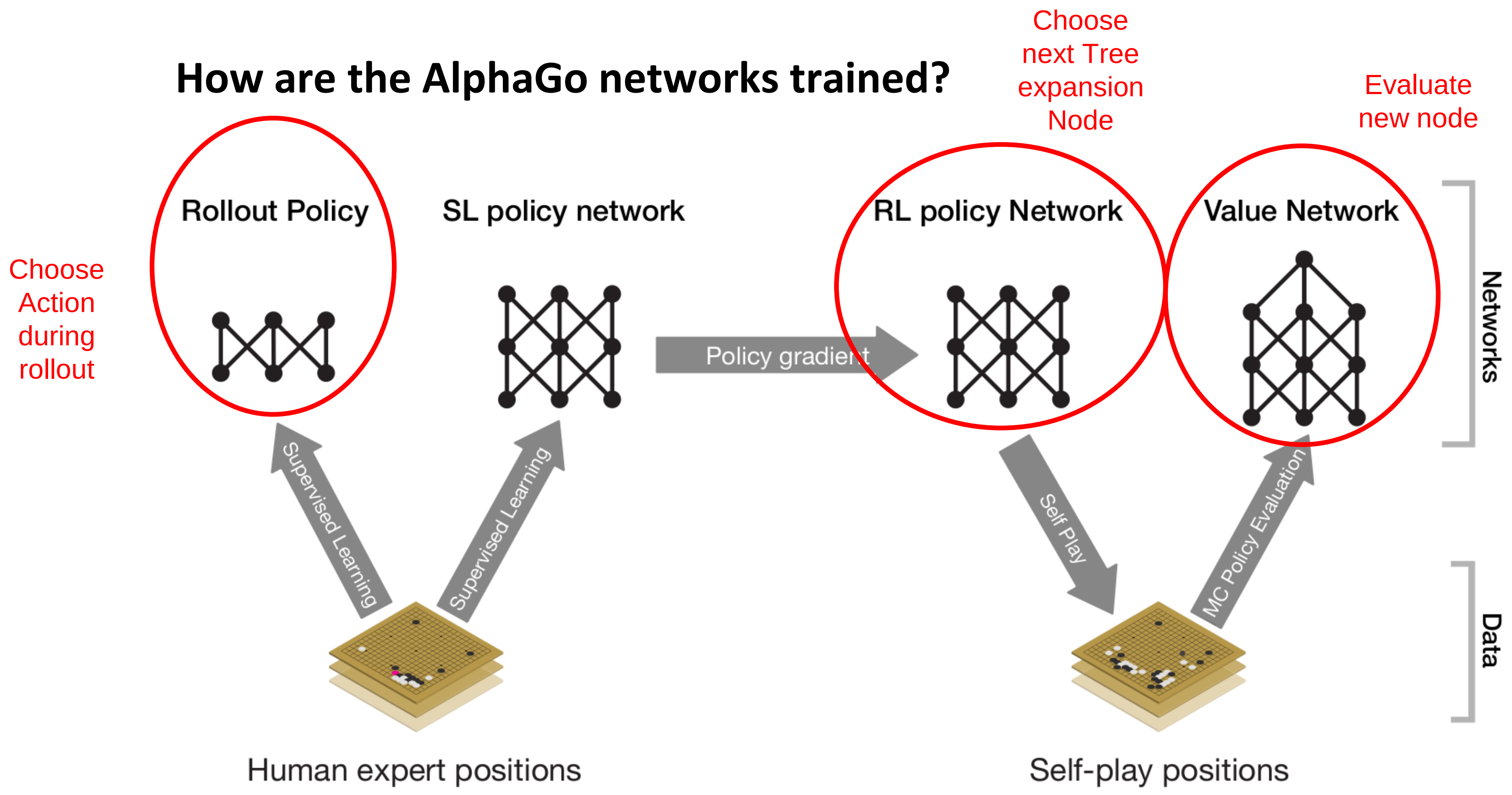


→ „Imitate Human Players“ „Beat the teacher“ via self-play

https://www.nature.com/articles/nature16961

Mastering the Game of Go: AlphaGo (2016)

How are the AlphaGo networks trained?



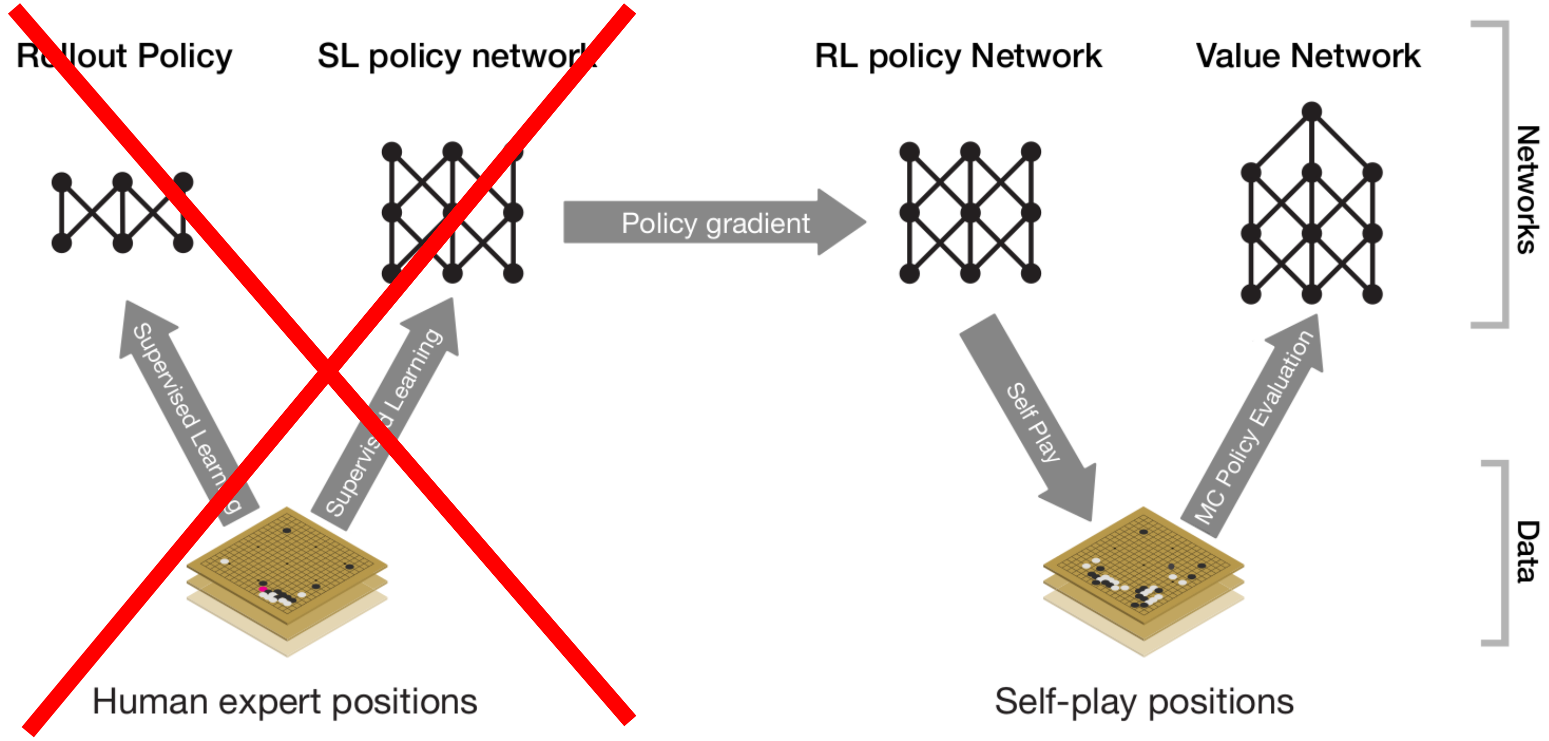
https://www.nature.com/articles/nature16961

AlphaGo Zero (2017)

https://www.youtube.com/watch?v=tXlM99xPQC8

AlphaGo Zero (2017)

– No Human data (!) Starts from scratch (→ random actions)



https://deepmind.com/documents/119/agz_unformatted_nature.pdf

AlphaGo Zero (2017)

- No Human data (!) Starts from scratch (→ random actions)
- Only Self-Play Reinforcement Learning
- MCTS seems to massively stabilize the learning

How/why is Zero's training so stable?
[…] deep RL is notoriously unstable and prone to forgetting, self-play is notoriously unstable and prone to forgetting, the two together should be a disaster without a good (imitation-based) initialization & lots of historical checkpoints to play against.

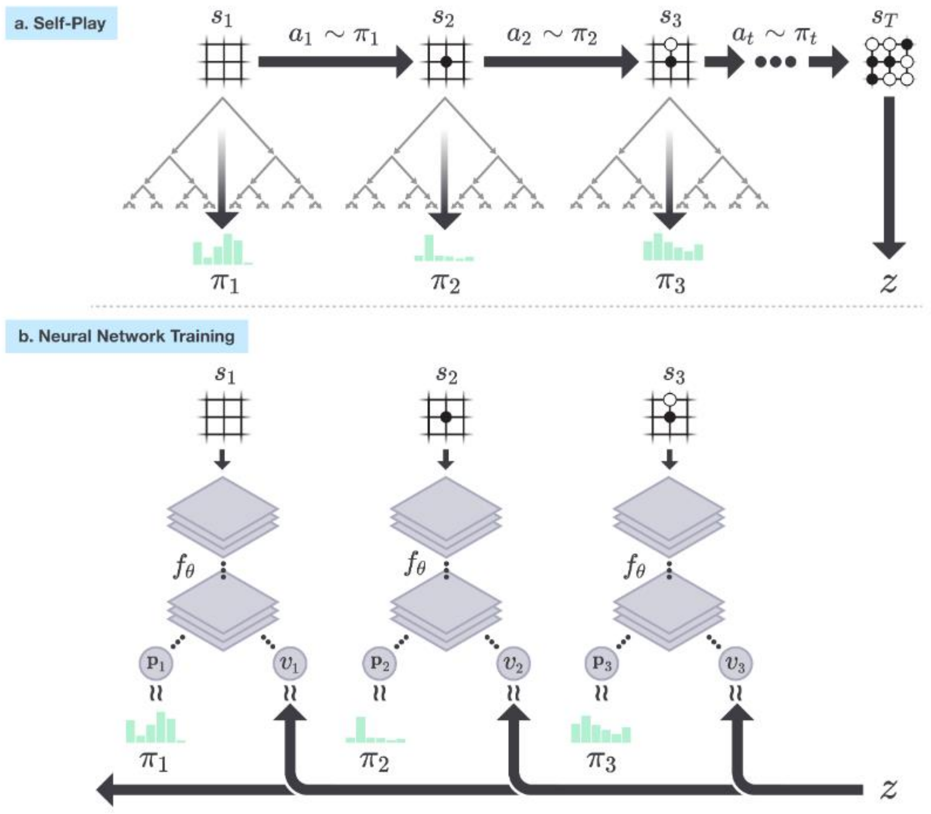
David_Silver:
AlphaGo Zero uses a quite different approach to deep RL than typical (model-free) algorithms such as policy gradient or Q-learning. By using AlphaGo search we massively improve the policy and self-play outcomes - and then we apply simple, gradient based updates to train the next policy + value network. This appears to be much more stable than incremental, gradient-based policy improvements that can potentially forget previous improvements.

https://www.reddit.com/r/MachineLearning/comments/76xjb5/ama_we_are_david_silver_and_julian_schrittwieser/dolgji4/?context=8&depth=9

https://deepmind.com/documents/119/agz_unformatted_nature.pdf

AlphaGo Zero (2017)

- No Human data (!)
  Starts from scratch
  (→ random actions)
- Only Self-Play Reinforcement Learning
- MCTS seems to massively stabilize the learning
- AlphaGo Zero: Uses MCTS during RL Self-Play (training)
- AlphaGo: Uses MCTS during live-play (but not in training)



https://discovery.ucl.ac.uk/id/eprint/10045895/1/agz_unformatted_nature.pdf

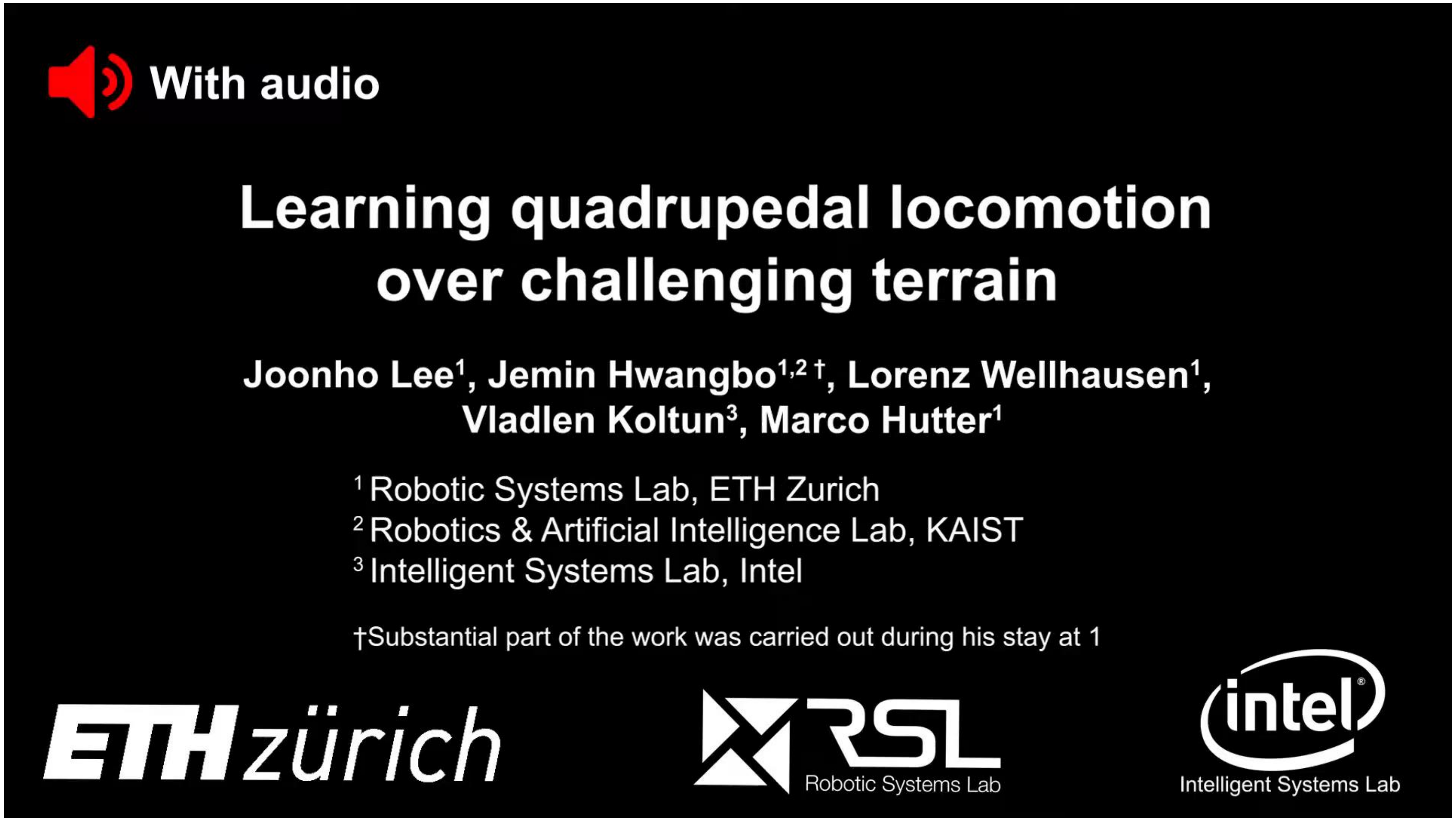
mobile and distributed systems group

Other (Real-World / Non-Games) Applications

- Facebook improves its **push notifications** with RL
  (Facebook Horizon – E2E RL Platform)
- JPMorgan: LOXM - Deep RL based **Trading Application**
- Concepts for **Fleet-Management** Optimization (Ride-Sharing)
- **A/B Testing** Optimization using Multi-Armed-Bandits
- Siemens & bons.ai: **CNC Machine Calibration** Optimization

Robot Locomotion

Lee, Joonho, et al. "Learning quadrupedal locomotion over challenging terrain.„ Science robotics 5.47 (2020).



With audio

Learning quadrupedal locomotion over challenging terrain

Joonho Lee[1], Jemin Hwangbo[1,2 †], Lorenz Wellhausen[1], Vladlen Koltun[3], Marco Hutter[1]

[1] Robotic Systems Lab, ETH Zurich
[2] Robotics & Artificial Intelligence Lab, KAIST
[3] Intelligent Systems Lab, Intel

†Substantial part of the work was carried out during his stay at 1

ETH zürich

RSL Robotic Systems Lab

intel Intelligent Systems Lab

https://robotics.sciencemag.org/content/5/47/eabc5986

https://arxiv.org/abs/2010.11251

Thank you!