

Artificial Intelligence

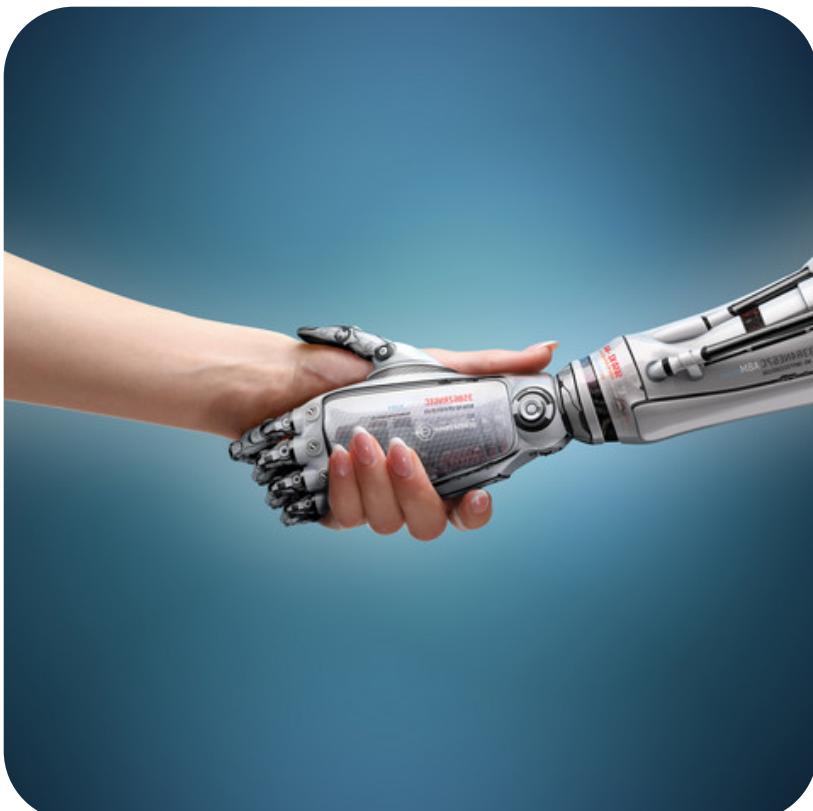
V11: Reinforcement Learning for Game Play

Learning to act

Example: DeepMind's Alpha Zero
Training the policy/value network

Based on material by

- David Silver, DeepMind
- David Foster, Applied Data Science
- Surag Nair, Stanford University



Teaser



AlphaGo Zero Trailer: <https://youtu.be/tXIM99xPQC8>



AlphaGo The Movie: <https://youtu.be/WXuK6gekU1Y>

Teaser (See <https://youtu.be/tXIM99xPQC8>)



Educational objectives for today



- **Know** what **reinforcement learning** is and how it differs from supervised learning
- **Know** real-world applications of **reinforcement learning**
- **Explain** how **Alpha Zero** works in principle, apart from the neural network details
- **Be able** to **start working** on a simple **self-play example** yourself





1. LEARNING TO ACT

Reinforcement learning (RL)

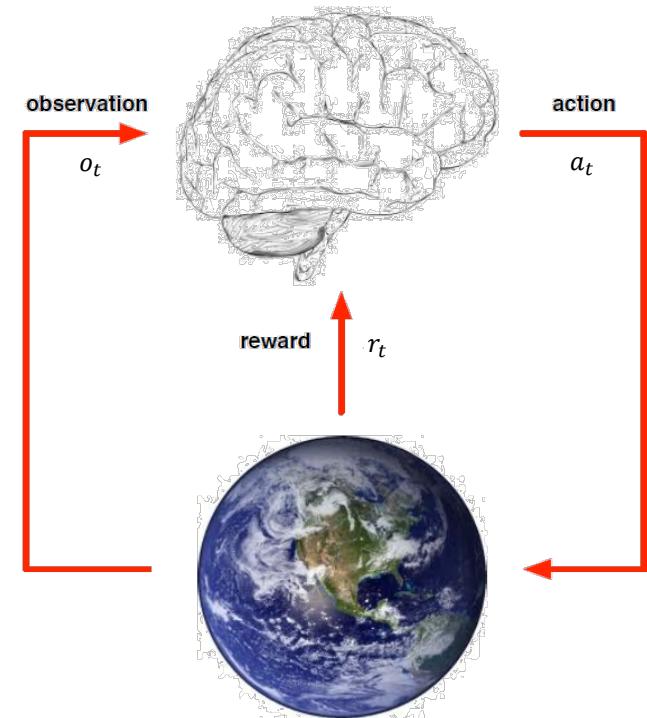
Agent learns by **interacting** with a stochastic environment
→ Science of **sequential decision making**

Many faces of reinforcement learning

- Optimal control (Engineering)
- Dynamic Programming (Operations Research)
- Reward systems (Neuroscience)
- Classical/Operant Conditioning (Psychology)

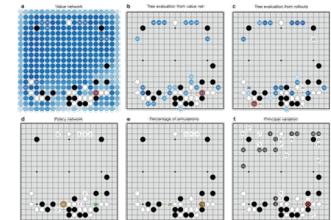
Characteristics

- No supervisor, no goals – only **reward** signals
- Feedback is **delayed**
- Objective: maximize cumulative reward
- Trade-off between **exploration & exploitation**
- Sequential decisions: actions effect observations (non i.i.d.)

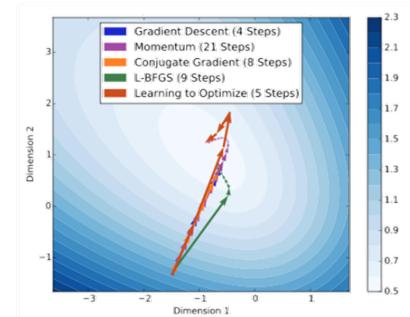


Application areas

- Automated vehicle control
 - An unmanned helicopter learning to fly and perform stunts
- Chat bots
 - Agent figuring out how to make a conversation
- Medical treatment planning
 - Planning a sequence of treatments based on the effect of past treatments
- **Game playing**
 - Playing backgammon, Atari Breakout, Tetris, Tic Tac Toe
- Data Center Cooling
 - <https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/>
- Database query optimization
 - J. Ortiz et al., “Learning State Representations for Query Optimization with Deep Reinforcement Learning”, DEEM’2018
- Learning new machine learning algorithms
 - <https://bair.berkeley.edu/blog/2017/09/12/learning-to-optimize-with-rl/>



AlphaGo



...and more

→ see <https://www.oreilly.com/ideas/practical-applications-of-reinforcement-learning-in-industry>,
<https://www.meetup.com/de-DE/Reinforcement-Learning-Zurich/> (currently less active)

2. EXAMPLE: DEEPMIND'S ALPHA ZERO



The game of Go

Properties

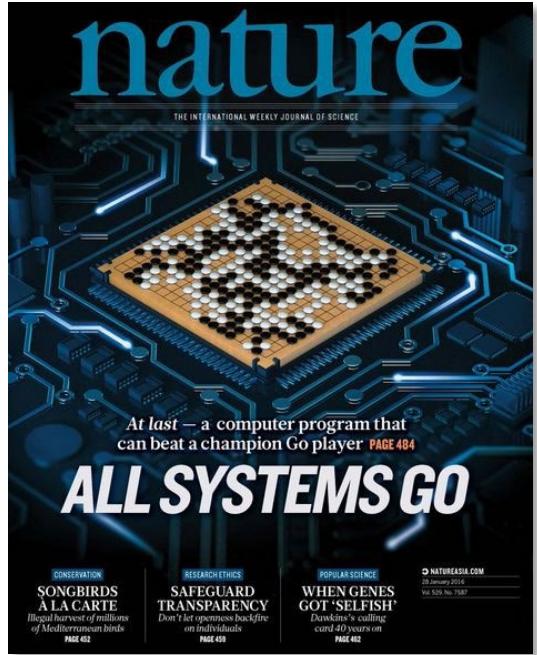
- **Perfect-information, deterministic**, two-player, turn-based, zero-sum game
- Played on a 19x19 board, alternate moves between black and white
- Two possible results: win or loss
- Considered a grand challenge for AI due to **vast search space** ($\sim 10^{170}$ states; chess: 10^{50})

Rules

- The player having **more “area” wins**
- Each turn, a **stone** of the player’s color is **put on an intersection** of the board (or “pass”)
- A stone (or **connected group** of stones) fully and **directly surrounded** by stones of the other color **is removed** from the board (“captured”)
- It is not allowed to recreate a former board position
- **Two consecutive passes end the game**



AlphaGo, AlphaGo Zero & Alpha Zero / MuZero

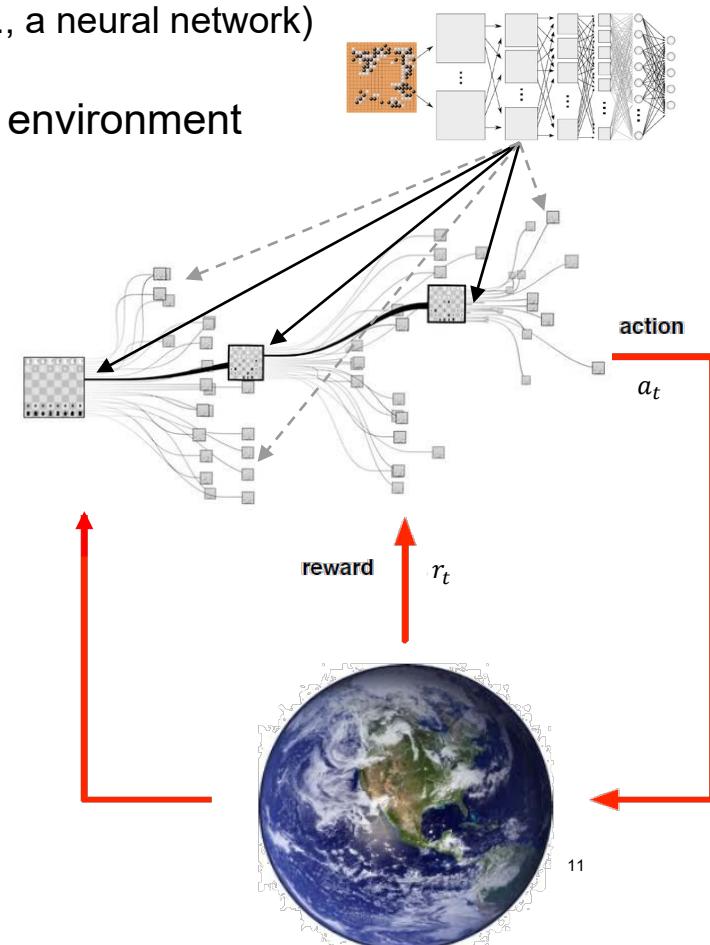


Interesting: playing strength ↑, generality ↑, complexity ↓ (over time)

Goal: a policy

Policy

- Policy $\pi = p(a_t|s_t)$ maps (probabilistically) from the current state s_t to action a_t
 ➔ can be represented by a **function approximator** (ML system, e.g., a neural network)
- Given the optimal policy π^* , one can behave optimally in the environment
 ➔ but optimality in complex strategic situations is difficult to achieve
 ➔ **lookahead search** makes **tactical behavior** easier



Using a learned policy in Alpha Zero

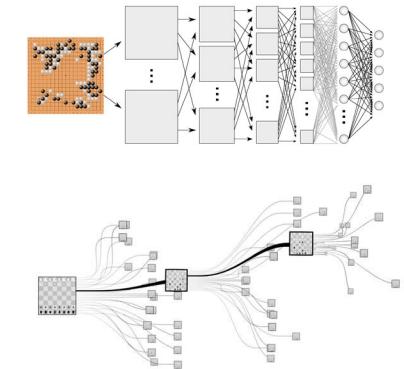
I.e., play a move given a policy

Goal

- In state s_t , chose next move a_t

Ingredients

- **Neural network** $\vec{p}, v = f_\theta(s_t)$ that outputs two quantities
 - Policy vector \vec{p} (a distribution over all possible actions)
 - Value v (an estimate of the probability of winning from this state)
- **Monte Carlo Tree Search (MCTS)** to build ad hoc search tree
 - MC: tree not fully grown → only **likely branches** get explored
 - (Chosen branch can be reused for next move for computational savings)



How to chose each move

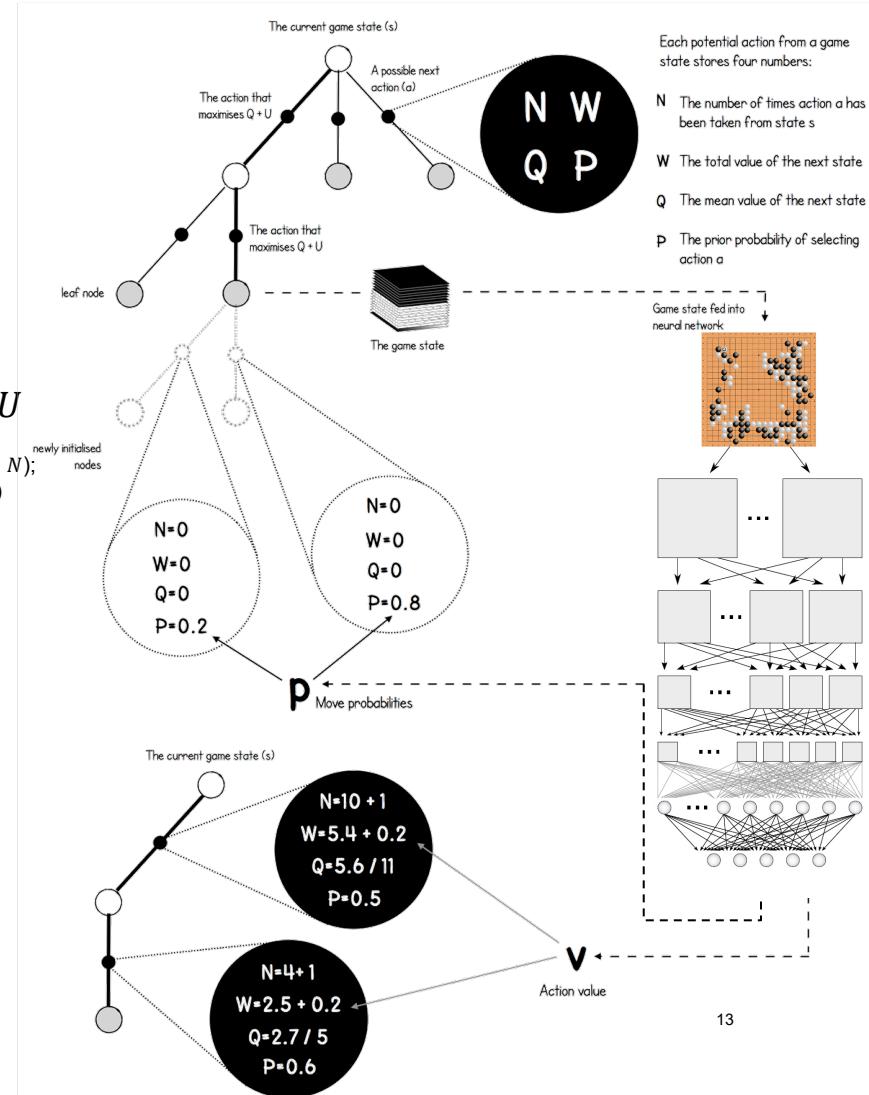
- Perform MCTS search on ad-hoc built tree
(using neural network for initial intuition if a move is good → see next slide)
- Play move most often taken by search ($\max(N)$)



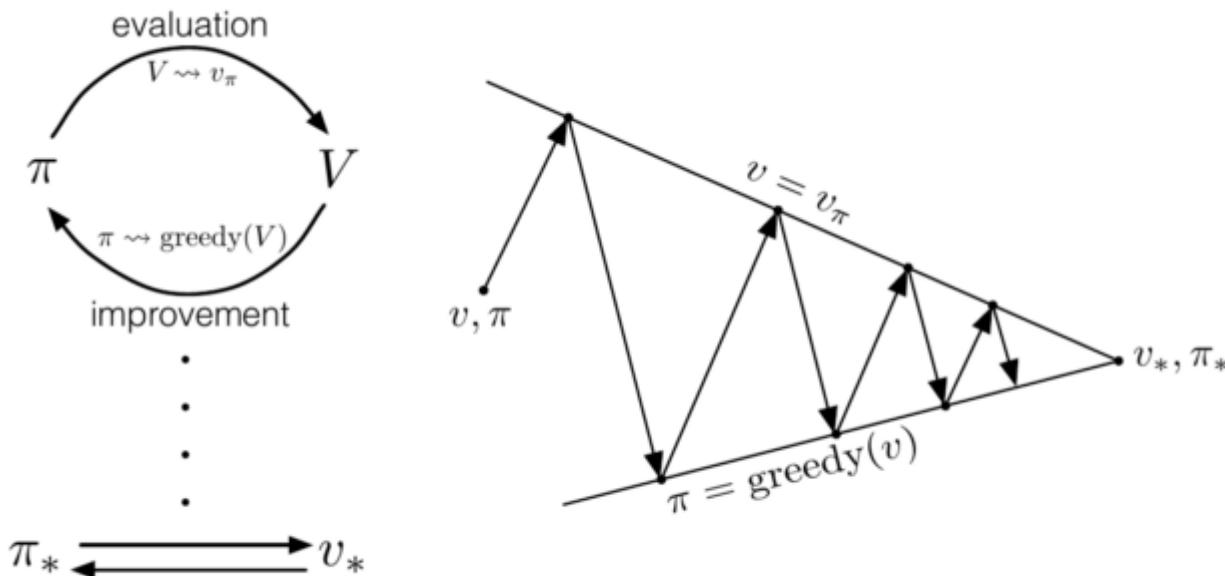
Perform a MCTS search

i.e., provide the basis for a move

- Create (empty or partly re-used) tree with root s_t
- Perform 1,600 simulations:
 - (one simulation = one traversal of current tree until yet unexpanded leaf node or terminal node is hit)
 - 1. Start at $s = s_t$
 - 2. Traverse tree:
while s is not a leaf node: choose a that maximizes $Q + U$
(Q is the current mean value of s over all simulations in this search;
 U is high if s has high prior probability p from the neural net, or hasn't been explored much (small N);
→ U dominates at the beginning of a search; as the branch gets explored, Q becomes important)
 - 3. Expand tree: query neural net for $\vec{p}, v = f_\theta(s)$
 $N = 0, W = 0, Q = 0, p = \vec{p}_a$
 - 4. Backup: update statistics of each visited node:
 $N = N + 1, W = W + v, Q = W/N$



3. TRAINING THE POLICY/VALUE NETWORK BY POLICY ITERATION

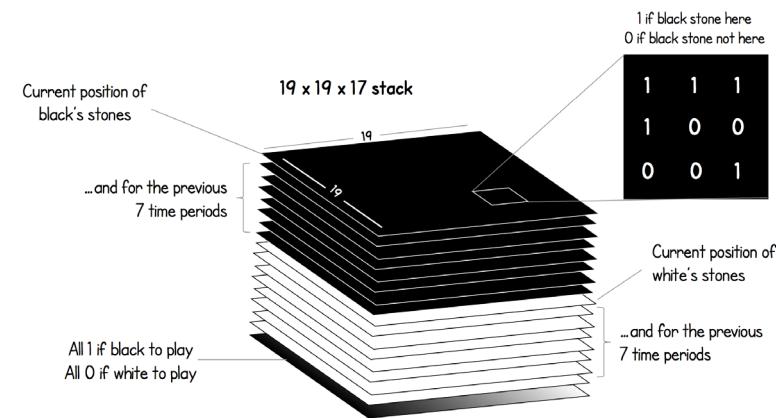
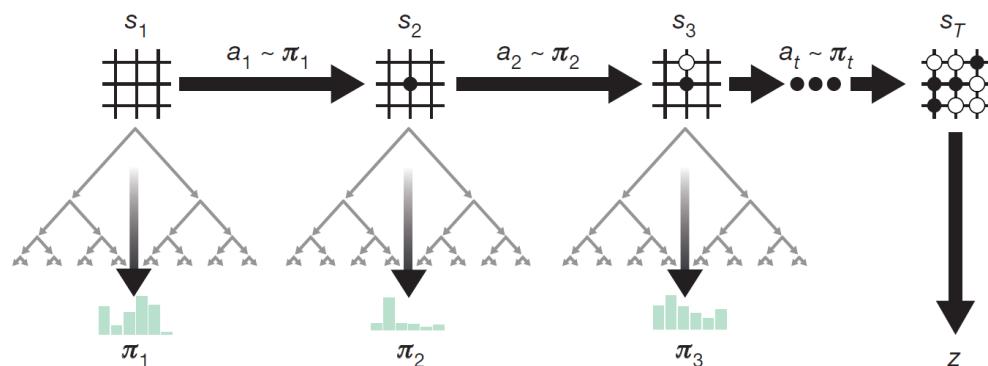


Source: <http://incompleteideas.net/book/ebook/node46.html>

Create experience by selfplay

(=Evaluate the current policy)

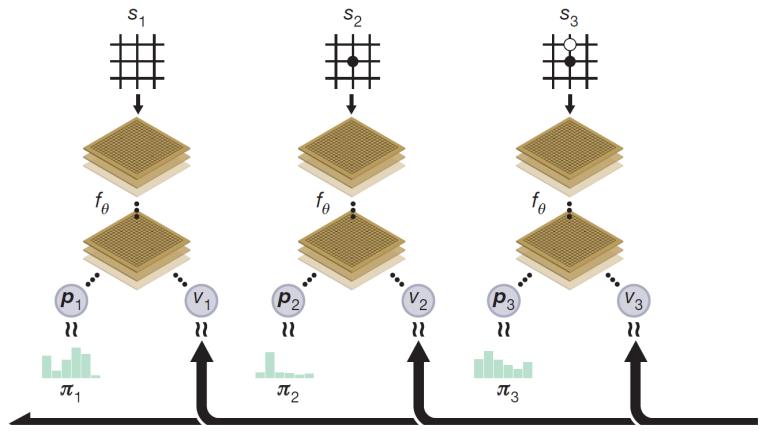
1. Initialize f_θ randomly
2. Play 25,000 games against yourself
 - Use MCTS and current best f_θ for both player's moves
 - **For each move, store**
 - **game state** (see figure →),
 - **search probabilities** from MCTS ($\pi_t \sim N$ for all actions of s_t),
 - **winner** ($z = \pm 1$ from perspective of current player)



3. Trigger retraining (→ see next slide), goto 2

(Re-)train neural network (=Improve the current policy)

1. Experience replay: sample mini-batch of 2,048 positions from last 500,000 self-play games
2. Retrain f_θ on this batch using **supervised learning**:
 - **Input:** game states
 - **Output:** move-probabilities p (dropping vector notation for simplicity), value v
 - **Labels:** search-probabilities π , actual winner z
 - **Loss:** cross-entropy between p, π + MSE between v, z + L_2 -regularization of θ



3. Trigger evaluation (→ see next slide) after 1,000 training loops, goto 1

Evaluate current network

1. Play 400 games between current best vs. latest f_θ
 - Choose each move by MCTS and respective network
 - Play deterministically (no additional exploration → see below)

After 1,600 simulations, the move can either be chosen:

Deterministically (for competitive play)

Choose the action from the current state with greatest N

Stochastically (for exploratory play)

Choose the action from the current state from the distribution

$$\pi \sim N^{1/\tau}$$

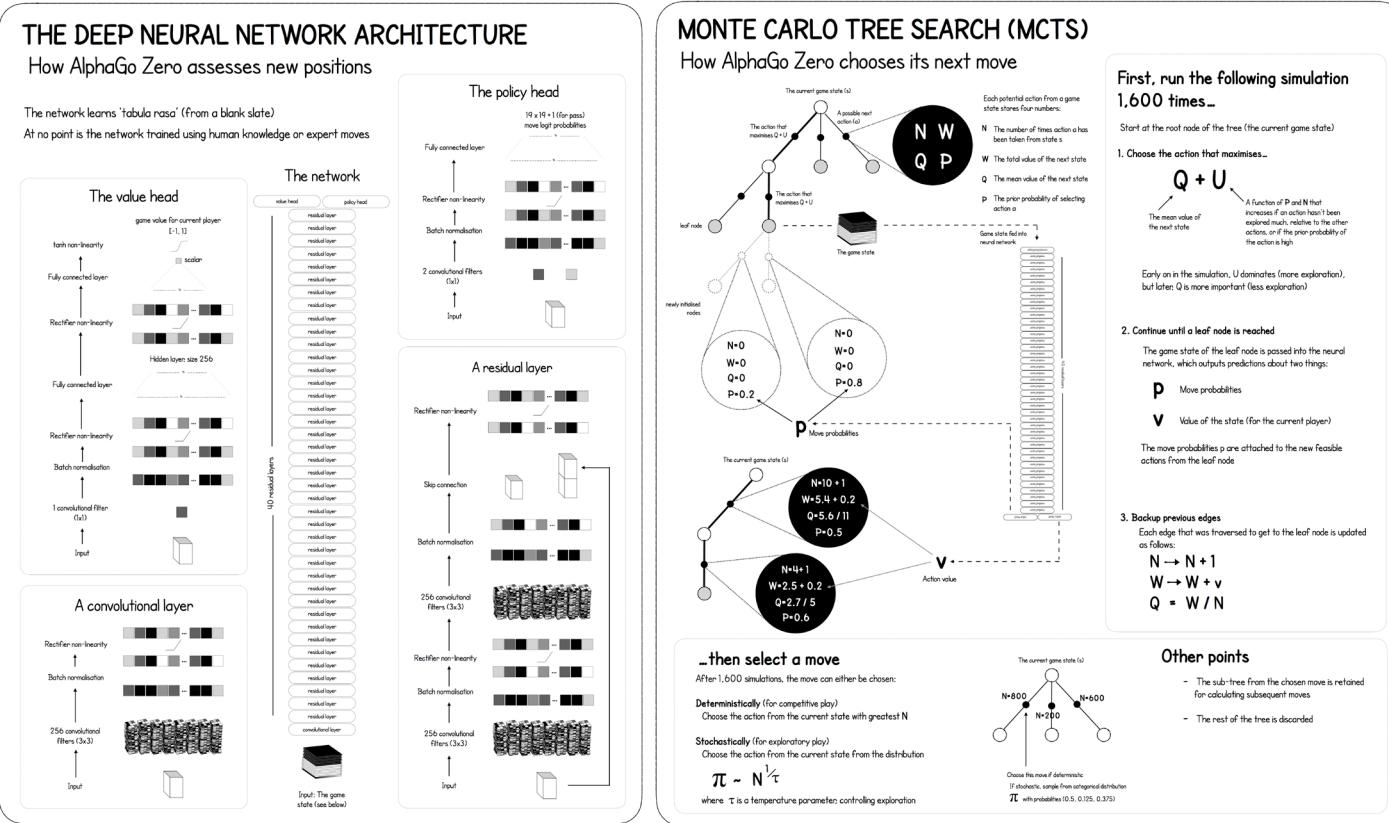
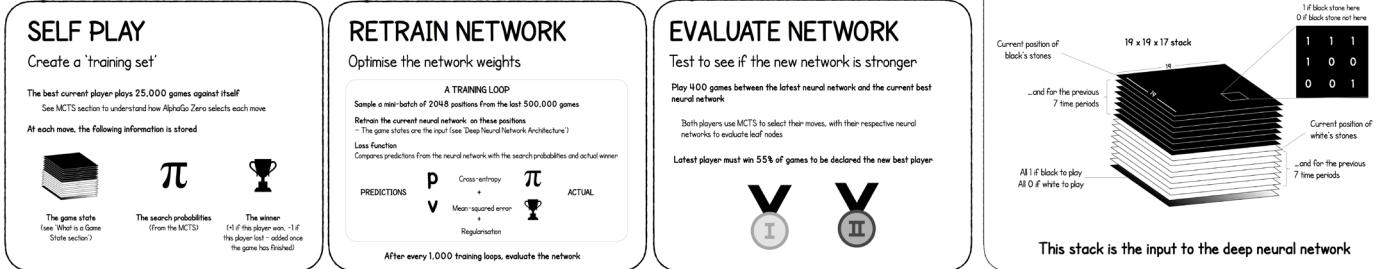
where τ is a temperature parameter; controlling exploration

2. Replace best network with latest f_θ if the latest wins $\geq 55\%$ of matches

Alpha Zero overview

Source: <https://medium.com/applied-data-science/alphago-zero-explained-in-one-diagram-365f5abf67e0>

The training pipeline for AlphaGo Zero consists of three stages, executed in parallel



Important RL concepts showcased here

To be detailed elsewhere

- Formal framework: **Markov decision processes** (MPDs)
- The RL problem: observations vs. **states**, learning vs. **planning, prediction & control**
- Ingredients to a solution: **model, value function** (v : state-value / q : action-value), **policy**
- Methods: **dynamic programming (policy iteration)**, **Monte Carlo**, temporal difference learning
- RL & function approximation: general instability, **experience replay, target networks**
- Exploration vs. exploitation: **optimistic initialization (upper confidence bounds)**, noise on priors



Where's the intelligence? Man vs. machine

- Alpha(Go) Zero **learns** without human intervention **from scratch** (pure selfplay & the rules)
→ strong point for capabilities of RL
 - Alpha(Go) Zero is considerably more **simple/principled** than previous approaches
→ good ideas are usually simple and intuitively right (the reverse is not necessarily true!)
 - Recently*, OpenAI showed similar **fascinating performance** on Dota2, and DeepMind on Quake III Arena** as well as on Atari & board games combined***
→ RL has made big progress and seems fit for real applications beyond simulations
-
- Yet****, solutions are still hand-crafted per use case and suffer from extreme **sample inefficiency** and **training instabilities**
→ Training takes very long even on server hardware, debugging is frustrating, success is fragile
 - And: Go was an easy and exceptional target, difficult to repeat elsewhere*****

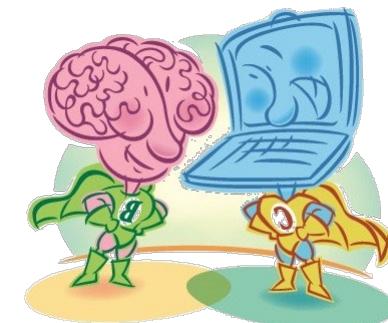
*) See <https://blog.openai.com/openai-five/> and <https://blog.openai.com/learning-dexterity/>

**) See <https://deepmind.com/blog/capture-the-flag/>

***) See <https://arxiv.org/abs/1911.08265>

****) See <https://www.alexirpan.com/2018/02/14/rl-hard.html> and <http://amid.fish/reproducing-deep-rl>

*****) See <https://medium.com/@karpathy/alphago-in-context-c47718cb95a5>

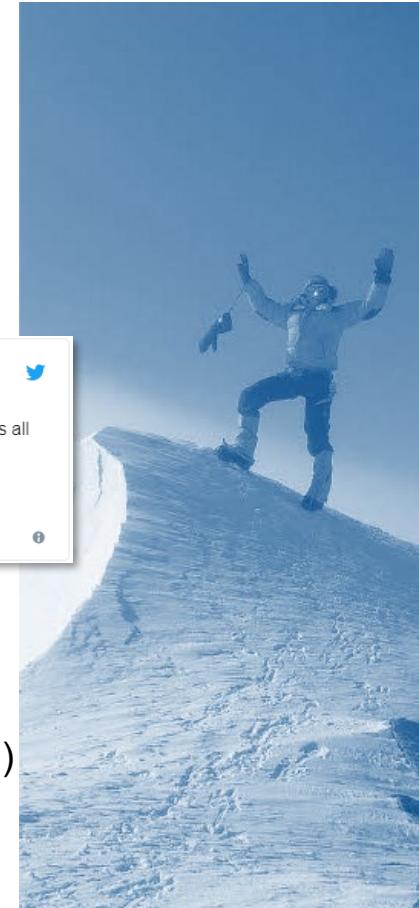


Review

- **Reinforcement learning (RL)** is “**learning to act**” – a general method for “**sequential decision making**”
- Most notable **differences** from unsupervised & supervised **ML**:
 - **no “data set”**
 - agent learns from interaction with environment and sparse rewards
 - less learning signal
 - **experience** is highly correlated and **not i.i.d.!**; but:



- **Alpha Zero** uses an elegant RL algorithm based on
 - **Selfplay** (for experience generation)
 - **MCTS** tree search (to plan ahead in a principled way)
 - Function approximation using **deep learning** (to use intuition as guide for tree search)
- Read the original publication, it is worth it (clear, concise, precise, complete):
<https://www.nature.com/articles/nature24270>





APPENDIX

Pseudo code – training π

Source: <https://web.stanford.edu/~surag/posts/alphazero.html>

```
def policyIterSP(game):
    nnet = initNNNet() #initialise random neural network
    examples = []
    for i in range(numIters):
        for e in range(numEps):
            #collect examples from this game
            examples += executeEpisode(game, nnet)
        new_nnet = trainNNNet(examples)
        #compare new net with previous net
        frac_win = pit(new_nnet, nnet)
        if frac_win > threshold:
            nnet = new_nnet #replace with new net
    return nnet

def executeEpisode(game, nnet):
    examples = []
    s = game.startState()
    mcts = MCTS() #initialise search tree

    while True:
        for _ in range(numMCTSSims):
            mcts.search(s, game, nnet)
        #rewards can not be determined yet
        examples.append([s, mcts.pi(s), None])
        #sample action from improved policy
        a = random.choice(len(mcts.pi(s)), p=mcts.pi(s))
        s = game.nextState(s,a)
        if game.gameEnded(s):
            examples = assignRewards(examples, game.gameReward(s))
    return examples
```

```
def search(s, game, nnet):
    if game.gameEnded(s): return -game.gameReward(s)

    if s not in visited:
        visited.add(s)
        P[s], v = nnet.predict(s)
        return -v

    max_u, best_a = -float("inf"), -1
    for a in range(game.getValidActions(s)):
        u = Q[s][a] + c_puct*P[s][a]*sqrt(sum(N[s]))/(1+N[s][a])
        if u>max_u:
            max_u = u
            best_a = a
    a = best_a

    sp = game.nextState(s, a)
    v = search(sp, game, nnet)

    Q[s][a] = (N[s][a]*Q[s][a] + v)/(N[s][a]+1)
    N[s][a] += 1
    return -v
```