Deep Q-Learning, AlphaGo and AlphaZero

Lyle Ungar

With slides from Eric Eaton

DQN
AlphaGo
AlphaZero, MuZero

Remember Q-Learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma Q\left(s', \pi(s')\right) - Q(s,a)\right)$$

Converges when this is zero

where

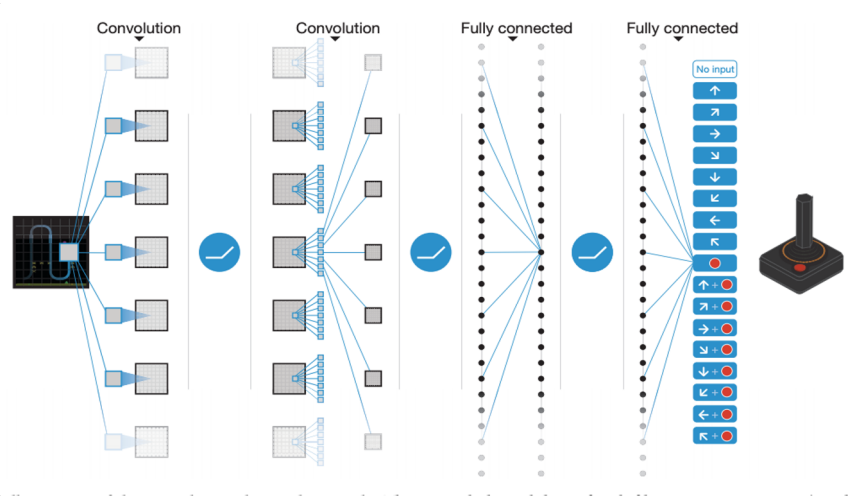
$$Q\left(s', \pi(s')\right) = \max_{a'} Q(s', a')$$

Deep Q-Learning (DQN)

Input: *s*

Output: *Q(s,a)*

Learning:
gradient descent with the following loss function:



$$\left(\left(R(s,a,s') + \gamma \max_{a'} Q(s',a')\right) - Q(s,a)\right)^2$$

The policy, $\pi(a)$, is then given by maximizing the predicted Q-value

Separate Q- and Target Networks

Issue: **Instability (e.g., rapid changes) in the Q-function can cause it to diverge**

Idea: **use two networks to provide stability**

- **The Q-network is updated regularly**
- **The target network is an older version of the Q-network, updated occasionally**

$$\left(\left(R(s,a,s') + \gamma \max_{a'} \underbrace{Q(s',a')}_{\text{computed via target network}}\right) - \underbrace{Q(s,a)}_{\text{computed via Q-network}}\right)^2$$

Deep Q-Learning (DQN) Algorithm

Initialize replay memory $\mathcal{D}$
Initialize Q-function weights θ
for episode $= 1 \ldots M$, do
  Initialize state s_t
  for $t = 1 \ldots T$, do
    $a_t \leftarrow \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \max_a Q^*(s_t, a; \theta) & \text{with probability } 1 - \epsilon \end{cases}$ ε-greedy
    Execute action a_t, yielding reward r_t and state s_{t+1}
    Store $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in $\mathcal{D}$
    $s_t \leftarrow s_{t+1}$
    Sample random minibatch of transitions $\{\langle s_j, a_j, r_j, s_{j+1} \rangle\}_{j=1}^{N}$ from $\mathcal{D}$
    $y_j \leftarrow \begin{cases} r_j & \text{for terminal state } s_{j+1} \\ r_j + \gamma \max_{a'} Q\left(s_{j+1}, a'; \theta\right) & \text{for non-terminal state } s_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
  end for
end for

Based on https://arxiv.org/pdf/1312.5602v1.pdf

DQN on Atari Games

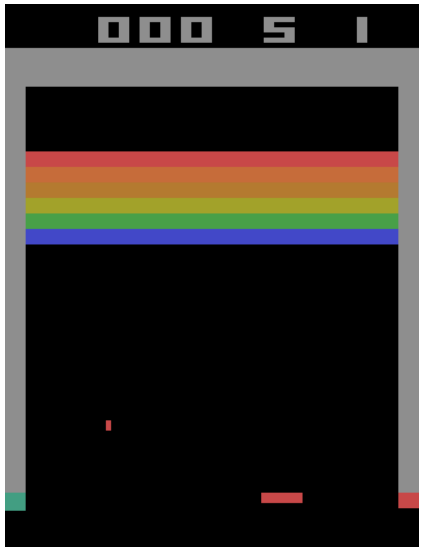
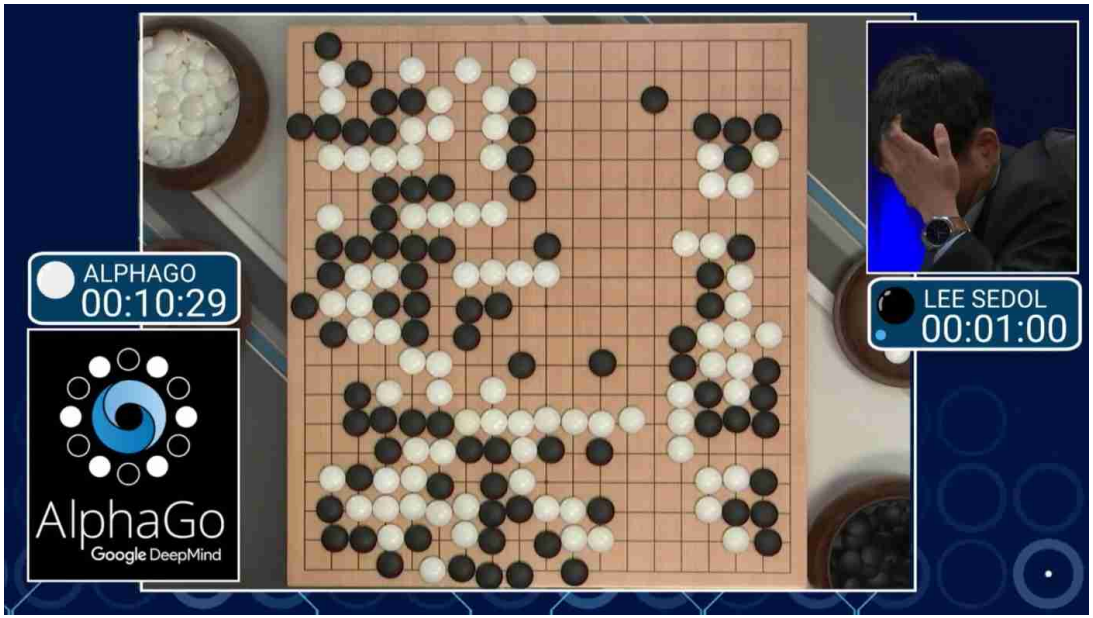


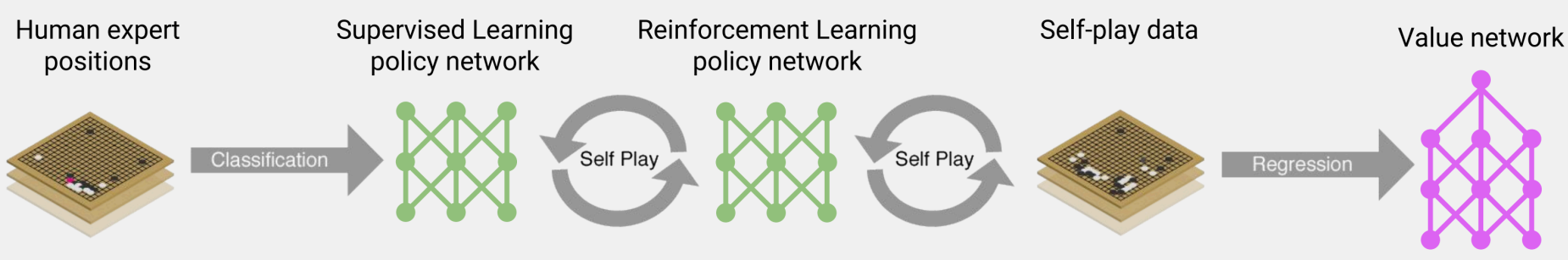
Image Sources:
https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756
https://deepmind.com/blog/going-beyond-average-reinforcement-learning/
https://jaromiru.com/2016/11/07/lets-make-a-dqn-double-learning-and-prioritized-experience-replay/

AlphaGo



https://medium.com/@jonathan_hui/alphago-how-it-works-technically-26ddcc085319 **2016**

AlphaGo - Complicated



1. Train a CNN to predict (supervised learning) moves of human experts

2. Use as starting point for policy gradient (self-play against older self)

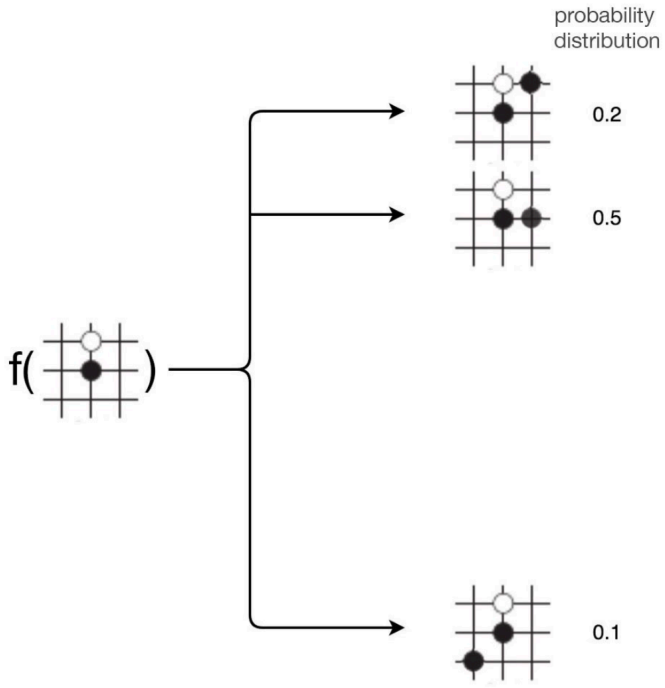
3. Train value network with examples from policy network self-play

4. Use Monte Carlo tree search to explore possible games

Image from DeepMind's ICML 2016 tutorial on AlphaGo: https://icml.cc/2016/tutorials/AlphaGo-tutorial-slides.pdf

Learn policy

- ***a*** = *f(**s**)*
- ***a*** = where to play (19*19)
- ***s*** = description of board



State ~ 19*19*48

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

AlphaGo – lots of hacks

- **Bootstrap**
  - Initialize with policy learned from human play
- **Self-play**
- **Speed matters**
  - Rollout network (fast, less accurate game play)
  - Monte Carlo search
- **It still needed fast computers**
  - > 100 GPU weeks

AlphaZero

- **Self-play with a single, continually updated neural net**
  - No annotated features - just the raw board position
- **Uses Monte Carlo Tree Search**
- **Beat AlphaGo (100-0) after just 72 hours of training**
  - On 5,000 TPUs

https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go

2017

Monte Carlo Tree Search (MCTS)

- **In each state s_{root}, select a move, $a_t \sim \pi_t$ either proportionally** (exploration) **or greedily** (exploitation)
- **Pick a move a_t with**
  - low visit count (not previously frequently explored)
  - high move probability (under the policy
  - high value (averaged over the leaf states of MC plays that selected *a* from *s*) according to the current neural net
- **The MCTS returns an estimate *z* of $v(s_{root})$ and a probability distribution over moves, $\pi = p(a|s_{root})$**

AlphaZero loss function

NNet: $(\mathbf{p}, v) = f_\theta(s)$

- Minimizes the error between the predicted outcome – the value function: *v(s)* and the actual game outcome *z*
- Maximizes the similarity of the policy vector ***p****(s)* to the MCTS probabilities $\pi(s)$.
- L2 regularize the weights θ

$$l = (z - v)^2 - \pi^{\mathrm{T}} \log \mathbf{p} + c\|\theta\|^2$$

MuZero

- **Slight generalization of AlphaZero**
  - Also uses Monte Carlo Tree Search (MCTC), self-play
- **Learns Go, Chess, Atari …**
- **Learns 3 CNNs**
  - **Representation model**: observation history $\longrightarrow$ $state_t$
  - **Dynamics model:** $state_t$ * $action_t$ $\longrightarrow$ $state_{t+1}$
  - **Policy**: $state_t$ $\longrightarrow$ $action_t$

StarCraft

- StarCraft-playing AI model consists of 18 agents, each trained with 16 Google v3 TPUs for 14 days.
- Thus, at current prices ($8.00 / TPU hour), the company spent $774,000 on this model

Applied RL Summary

- Superhuman game playing using DQL
- Why is DeepMind losing $500 million/year?