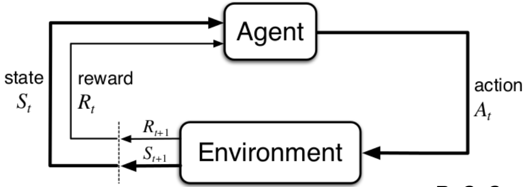
Deep Reinforcement Learning

Martin Matyášek

Artificial Intelligence Center
Czech Technical University in Prague

October 27, 2016

Reinforcement Learning in a picture



R. S. Sutton and A. G. Barto 2015

- learning what to do to maximize *future* reward
- general-purpose framework extending sequential decision making when the model of the environment is unknown

RL background

- let's assume MDP $\langle S, A, P, R, s_0 \rangle$
  - RL deals with situation where the environment model P and R is unknown
  - can be generalized to *Stochastic Games* $\langle S, N, A, P, R \rangle$

- RL agent includes:
  - **policy** $a = \pi(s)$ (deterministic), $\pi(a \mid s) = \mathbb{P}(a \mid s)$ (stochastic)
  - **value function** $Q^\pi(a \mid s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s, a]$
    - Bellman Eq.: $Q^\pi(a \mid s) = \mathbb{E}_{s',a'}[r + \gamma Q^\pi(a' \mid s') \mid s, a]$
    - opt. value funcions: $Q^*(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a]$
    - opt. policy: $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$
  - **model** - learned proxy for environment

RL Types

1. **value-based** RL
   - estimate the opt. value function $Q^*(s, a)$
   - max. value achievable under *any* policy
2. **policy-based** RL
   - search directly for the opt. policy π^*
   - i.e. policy yielding max. future reward
3. **model-based** RL
   - build a model of the environment
   - plan using this model

Q-learning

Algorithm 1 Q-learning

```
initialize the Q-function and V values (arbitrarily)
repeat
    observe the current state s_t
    select action a_t and take it
    observe the reward R(s_t, a_t, s_{t+1})
    Q_{t+1}(s_t, a_t) ← (1 − α_t)Q_t(s_t, a_t) + α_t(R(s_t, a_t, s_{t+1}) + γV_t(s_{t+1}))
    V_{t+1}(s) ← max_a Q_t(s, a)
until convergence
```

$$Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha_t)Q_t(s_t, a_t) + \alpha_t(R(s_t, a_t, s_{t+1}) + \gamma V_t(s_{t+1}))$$

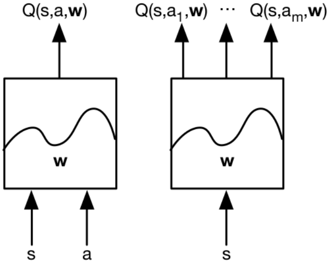
$$V_{t+1}(s) \leftarrow \max_a Q_t(s, a)$$

Q-learning

- **model-free** method
- *temporal-difference* version:
  - $Q(s,a) \leftarrow Q(s,a) + \alpha(\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{value based on next state}} - Q(s,a))$
- converges to Q^*, V^* iff $0 \leq \alpha_t < \infty$, $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$
- *zero-sum Stochastic Games*:
  - cannot simply use $Q_i^\pi : S \times A_i \rightarrow \mathbb{R}$ but rather $Q_i^\pi : S \times A \rightarrow \mathbb{R}$
  - **minimax-Q** converges to NE
  - **R-max**: converge to $\epsilon -$ Nash with prob. $(1-\delta)$ in poly. # steps (*PAC learn*)

Q-Networks

- $Q^*(s,a) \approx Q(s,a,\mathbf{w})$
- treat right hand side $r + \gamma \max_{a'} Q(s', a', \mathbf{w})$ of Bellman's Eq. as target
- minimize **MSE loss** $l = (r + \max_{a'} Q(s', a', \mathbf{w}) - Q(s,a,\mathbf{w}))^2$ by **stochastic gradient descent**



David Silver, Google DeepMind

Q-learning summary

+ converges to Q^* using table lookup representation
– diverges using NN:
  - correlations between samples
  - non-stationary targets

! **go deep**

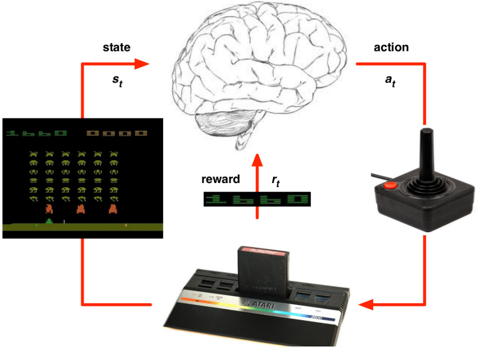
Deep Q-Networks (DQN)

- basic approach is called **experience replay**
- *idea*: remove correlations by building data-set from agent's experience $e = (s, a, r, s')$
  - sample experiences from $D_t = \{e_1, e_2, \dots e_t\}$ and apply update
- deal with non-stationarity by fixing $\mathbf{w}^-$ in $l = (r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}))^2$

Algorithm 2 Deep Q-learning algorithm

init. replay memory D, init. Q with random weights
observe initial state s
repeat
    with prob. ϵ select random a, select $a = \text{argmax}_{a'} Q(s, a')$
    carry out a, observe (r, s') and store (s, a, r, s') in D
    sample random transition (ss, aa, rr, ss') from D
    calculate target for each minibatch transition:
    if ss' is terminal state **then**
        $tt \leftarrow rr$
    else
        $tt \leftarrow rr + \gamma \max_{a'} Q(ss', aa')$
    end if
    train the Q-network using $(tt - Q(ss, aa))^2$ as loss
    $s \leftarrow s'$
until convergence

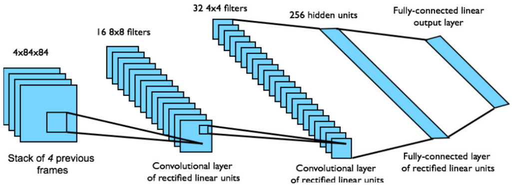
DQN in Atari



David Silver, Google DeepMind

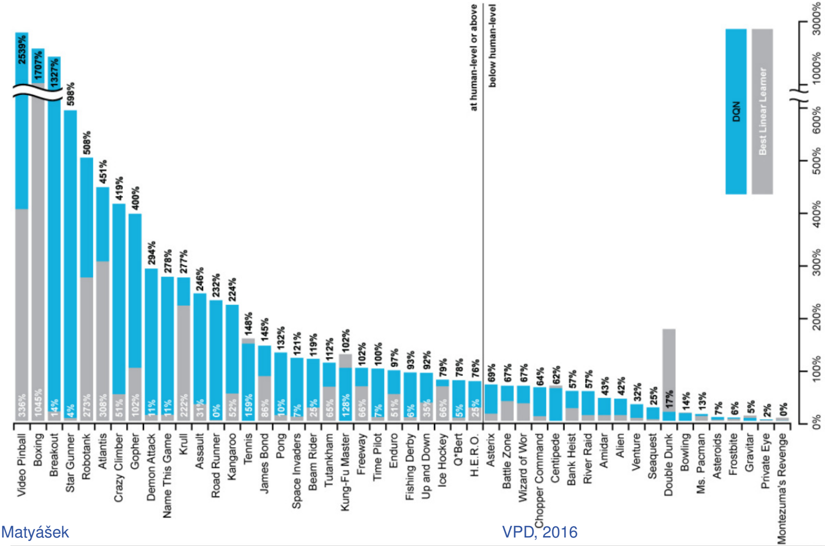
DQN in Atari - setting

state	stack of raw pixels from last 4 frames
actions	18 joystick/button positions
reward	delta in score
learn	$Q(s, a)$



David Silver, Google DeepMind

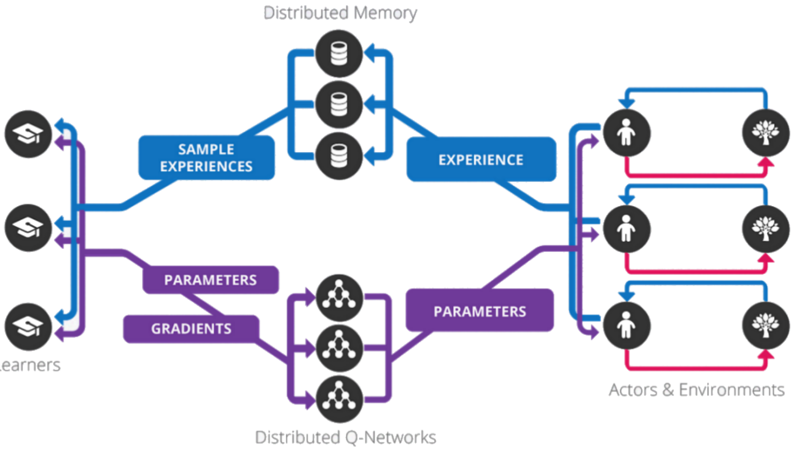
DQN in Atari - results



DQN improvements

- **Double DQN** removes bias caused by $\max_a Q(\cdot)$
  - current QN - $\mathbf{w}$ used to select actions
  - old QN - $\mathbf{w}^-$ used to evaluate actions
- **Prioritized Replay** weight experience according to DQN error (stored in PQ)
- **Duelling Network** split Q-Network into:
  - action-independent *value* function
  - action-dependent *advantage* function

General Reinforcement Learning Architecture (GORILA)



Deep Policy Network

- parametrize the policy π by a DNN and use SGD to optimize weights $\mathbf{u}$
  - $\pi(a \mid s, \mathbf{u})$ or $\pi(s, \mathbf{u})$
  - $\max_{\mathbf{u}} L(\mathbf{u}) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi(\cdot, \mathbf{u})]$
- policy gradients:
  - $\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}[\frac{\partial \log \pi(a|s,\mathbf{u})}{\partial \mathbf{u}} Q^{\pi}(s, a)]$ for *stochastic* policy $\pi(a \mid s, \mathbf{u})$
  - $\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}[\frac{\partial Q^{\pi}(s,a)}{\partial a} \frac{\partial a}{\partial \mathbf{u}}]$ for *deterministic* policy $a = \pi(s)$ where a is cont. and Q diff.
- variations: *Actor-Critic alg.*, *A3C*, *Fictitious Self-Play*

Game Theory 101

normal-form game $G = (\mathcal{N}, \mathcal{A}, u)$

- $\mathcal{N} = \{1, 2, \ldots, n\}$ - players
- $\mathcal{A} = \times_{i \in \mathcal{N}} A_i$ - pure strategies (actions)
  - $\Pi = \times_i \Pi_i = \times_i \Delta(A_i)$ - mixed stratedies
- $u = (u_1, \ldots, u_n)$, $u_i : \mathcal{A} \rightarrow \mathbb{R}$ - utilities

best response

$$BR_i(\pi_{-i}) = \{\pi_i \in \Pi_i \mid \forall \pi'_i \neq \pi_i : u_i(\pi_i, \pi_{-i}) \geq u_i(\pi'_i, \pi_{-i})\}$$

Nash equlibrium

$$NE(G) = \{\pi \in \Pi \mid \forall i \in \mathcal{N} : \pi_i \in BR_i(\pi_{-i})\}$$

Fictitious Self Play (FSP)

- **fictitious play (FP)**
  1. initialize beliefs about the opponent's strategy
  2. play a best response to the assessed strategy of the opponent
  3. observe the opponent's actual play and update beliefs accordingly, *goto 2*
- **fictitious self play (FSP)**
  - DQN with experience replay learns "BR" to opponent policies
  - policy network learns an average of BRs

$$\frac{\partial l}{\partial \mathbf{w}} = \frac{\partial \log \pi_{\mathbf{w}}(a \mid s)}{\partial \mathbf{w}}$$

  - actions a sample mix of policy network and best response

Deep RL Summary

- RL steps in when the model of the environment is unknown
- NN employed when the state space is too large
- RL sets the learning objective, NN then approximates:
  - *value function* $V^* \approx V_{\mathbf{w}}$, $Q^* \approx Q_{\mathbf{w}}$
  - *policy* $\pi^* \approx \pi_{\mathbf{u}}$
- use DNN to deal with convergence and correlations

What about some applications?

MvM History

Man vs Machine:

1992 backgammon *Tesauro* very close to top human experts

1996 chess Kasparov loses 2.5-3.5 vs *Deep Blue*

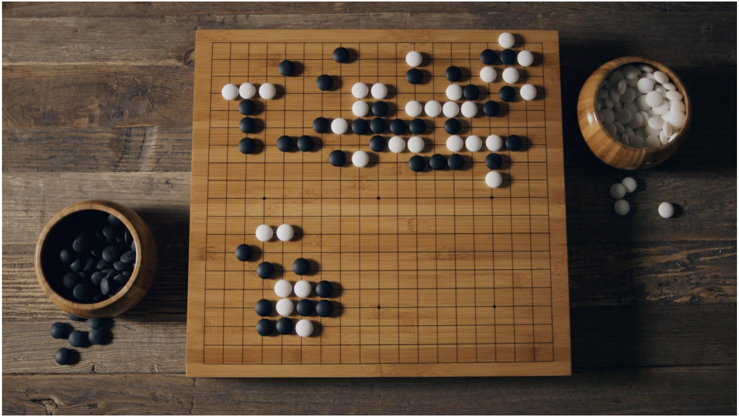
1997 othello Logistello vs Murakami 6-0

2007 checkers solved

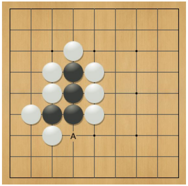
2008 poker Polaris wins vs poker pros

- *2015* - Heads-up limit Texas hold'em solved

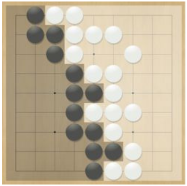
Go



The Rules of Go



(a) capture



(b) territory

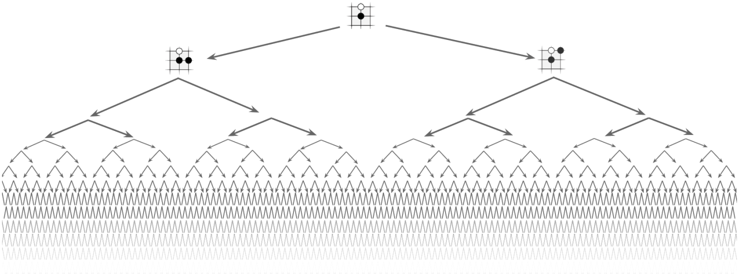
- **start** empty board
- **move** place one stone (of your color)
- **goal** surround
- **win** control more than half of the board

Go in Theory

- *2-players* (just me and the opponent)
- *zero-sum game* (win for me = loss for opponent)
- *perfect information*
- *finite* (the game rules ensure this)

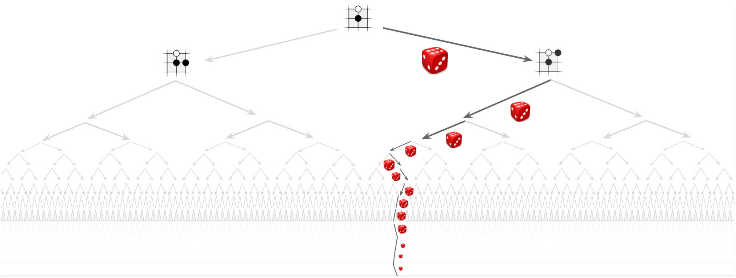
- NE exists and we can search for it:
  - *minimax*
  - *alpha-beta*
  - **negascout**

Go in Reality



there is $\mathcal{O}(b^d)$ game states where $b \approx 250$ and $d \approx 150$

Alternative to Exhaustive Search



Monte Carlo Tree Search (MCTS):

- popular heuristic search algorithm for game play
- Monte Carlo rollouts to estimate $v(s) \approx v^*(s)$ (reduces d)
- sampling actions from $p(a \mid s)$ reduces b
  - MC rollouts search to max. depth without branching at all

MCTS

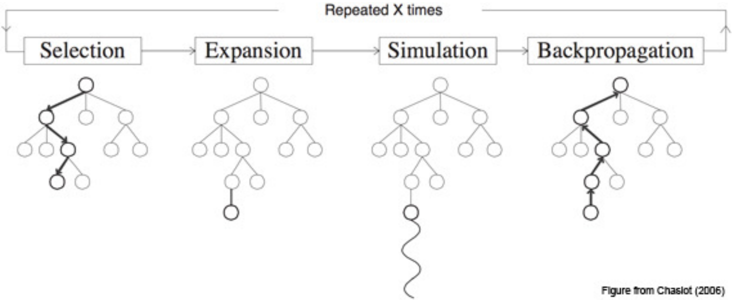


Figure from Chaslot (2006)

$t \leq 2015$

- the strongest Go programs are based on MCTS
- enhanced by policies that are trained to predict human expert moves
  - early rules hand-made
  - later ML based on simple features (lin. comb. of inputs)
- knowledge learned:
  (i) fast (simple) knowledge used for move selection in simulation (*rollout policy*)
  (ii) slower (better) knowledge used for move ordering in tree search (*SL policy*)

2016: Lee Sedol vs. AlphaGo

March 9-15, 2016

name	*Lee Sedol*	*AlphaGo*
age	33	2
rank	9 dan prof.	none
titles	18	0
power	1 brain	1200 CPU and 200 GPU
results	loss, loss, loss, **win**, loss	**win**, **win**, **win**, loss, **win**
experience	$c \cdot 1$k games	$c \cdot 1$M self-play games

How?

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

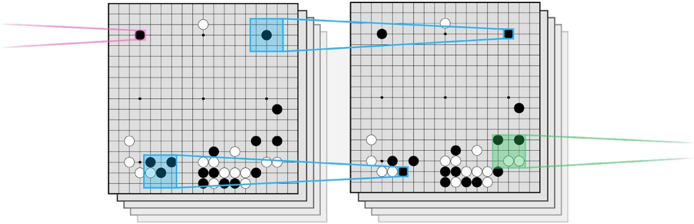
Science

AI CENTER

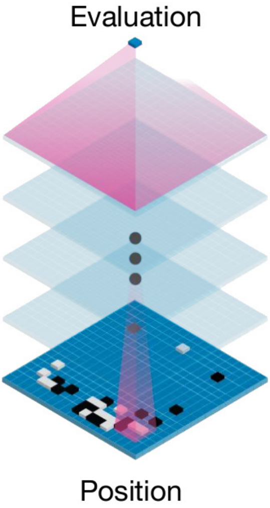
AlphaGo Design

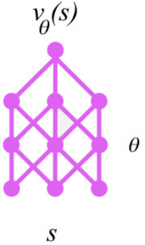
- normal search - MCTS
- simulation (rollout) policy - relatively normal
- supervised learning (SL) policy from master games - improved in details, more data
- **RL from self-play for value network**
- **RL from self-play for policy network**

Convolutional Neural Network

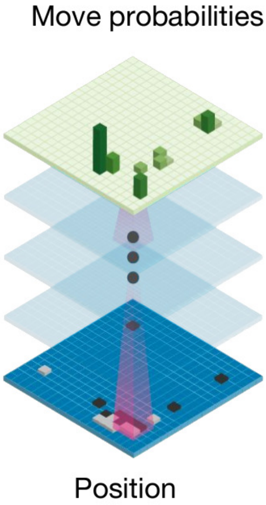


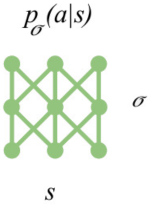
Value Network



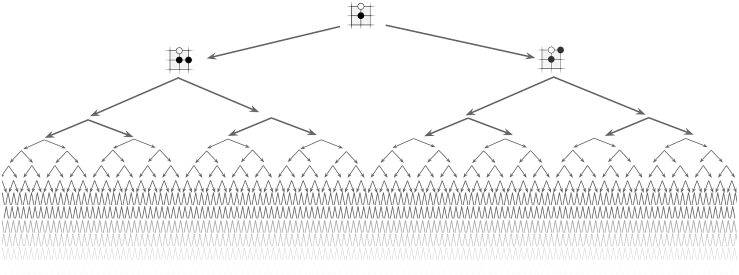


Policy Network

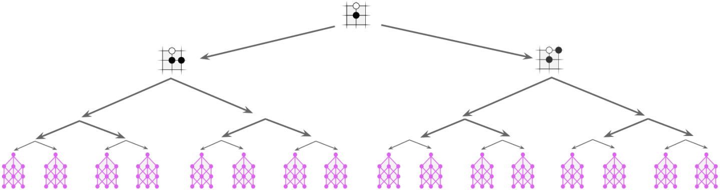




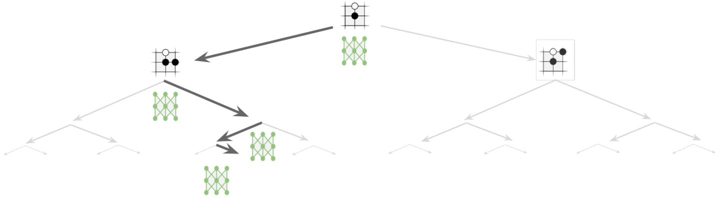
Reducing d with Value Network



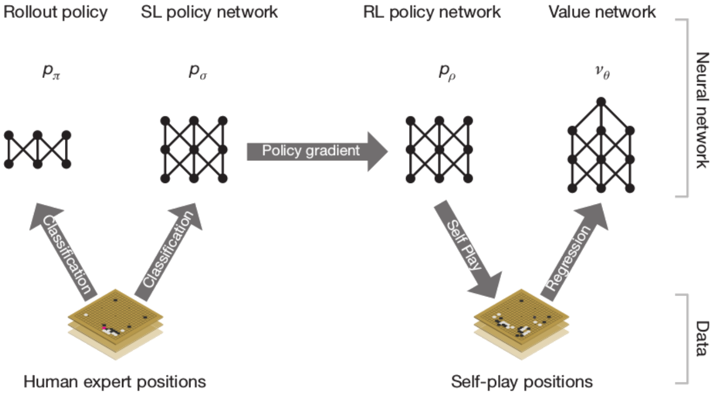
Reducing *d* with Value Network



Reducing b with Policy Network



Deep RL in AlphaGo



SL of Policy Networks

network 12 layer convolutional NN

training data 30M position from human experts (KGS 5+ dan)

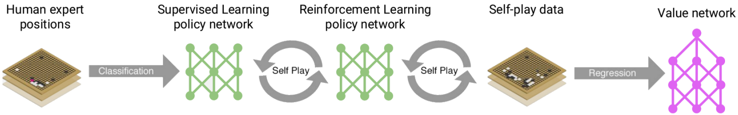
training alg. max. likelihood by SGD

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a \mid s)}{\partial \sigma}$$

training time 4 weeks on 50 GPUs (Google Cloud)

results 57% accuracy on held out test data

- state-of-the art was 44%



RL of Policy Networks

network 12 layer convolutional NN

training data games of self-play between policy networks

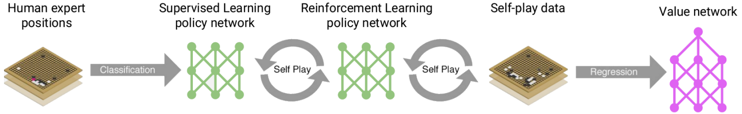
training alg. max. wins z by policy gradient RL

$$\Delta\rho \propto \frac{\partial \log p_\rho(a \mid s)}{\partial \rho} z$$

training time 1 week on 50 GPUs (Google Cloud)

results 80% vs. SL network p_σ

- raw network $\sim$ 3 amateur dan



RL of Value Networks

network 12 layer convolutional NN

training data 30M games of self-play

idea

$$v^p(s) = \mathbb{E}[z_t \mid s_t = s, a_{t,\dots,T} \sim p]$$

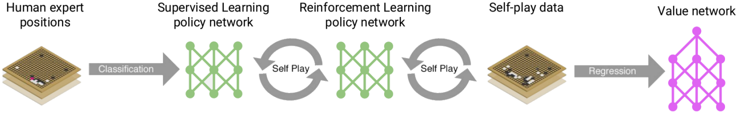
$$v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$$

training alg. min. MSE by SGD

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}(z - v_\theta(s))$$

training time 1 week on 50 GPUs (Google Cloud)

results **first strong position eval. function**



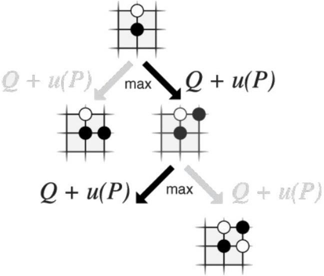
MCTS in AlphaGo

- each edge stores:
  action value $Q(s, a)$
  visit count $N(s, a)$
  prior prob. $P(s, a)$ (initialized to $P(s, a) = p_\sigma(a \mid s)$)
- at each step t we select in state s_t:

$$a_t = \mathrm{argmax}_a(Q(s_t, a) + u(s_t, a))$$

- $u(s, a) \propto \frac{P(s,a)}{1+N(s,a)}$ decays with repeated visits (encourages exploration)
- leaf node s_L is evaluated in two different ways:
  $V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L$
  1. by value network $v_\theta(s_L)$
  2. by outcome $z_L \sim^* p_\pi$

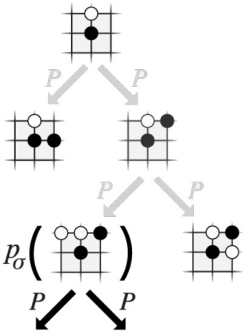
MCTS in AlphaGo: selection



P prior probability
Q action value

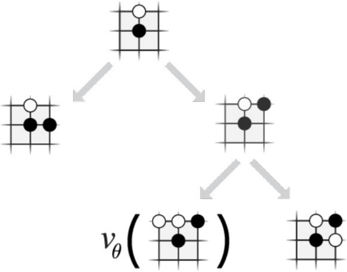
$$u(P) \propto P/N$$

MCTS in AlphaGo: expansion



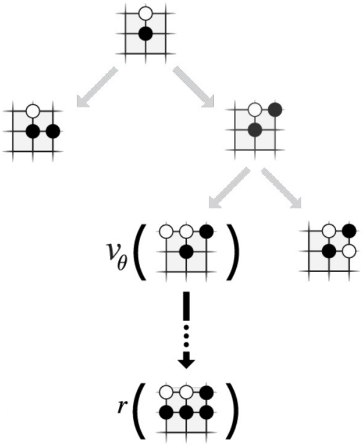
p_σ Policy network
P prior probability

MCTS in AlphaGo: evaluation



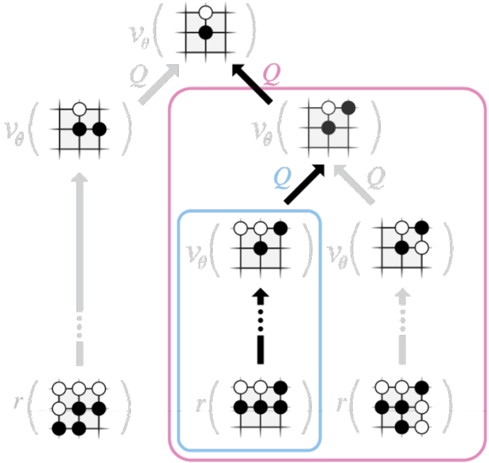
v_θ Value network

MCTS in AlphaGo: rollout



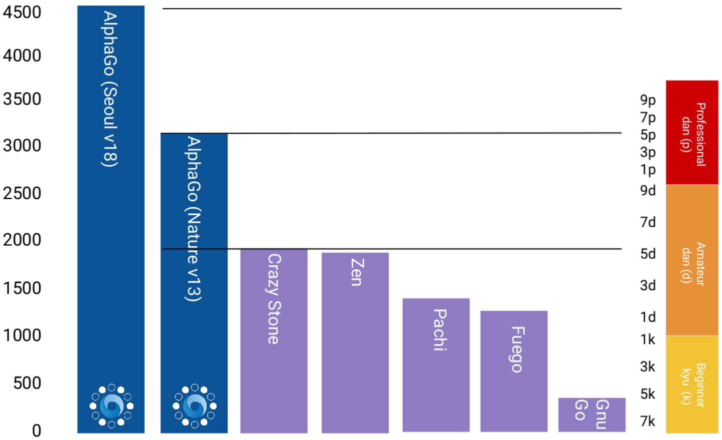
v_θ Value network
r Game scorer

MCTS in AlphaGo: backup



Q Action value
v_θ Value network
r Game scorer

AlphaGo: results



References

- Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.
- Silver, David and Huang, Aja and Maddison, Chris J and others. *Mastering the game of Go with deep neural networks and tree search*. Nature Publishing Group, 2016.
- Shoham, Yoav and Leyton-Brown, Kevin. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.
- Silver, David. *Tutorial: Deep Reinforcement Learning*. http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf
- Silver, David. *AlphaGo slides*. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/AlphaGo_IJCAI.pdf