Deep RL

(Slides by Svetlana Lazebnik, B Ravindran, David Silver)

Function approximation

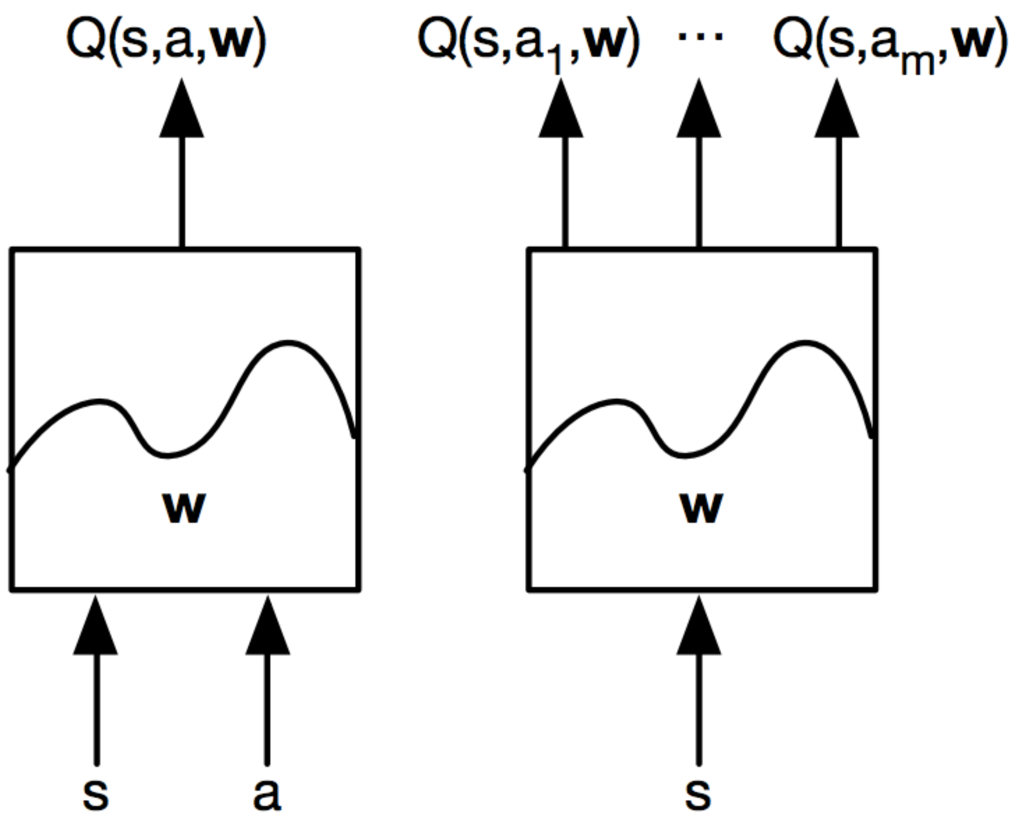
- So far, we've assumed a *lookup table* representation for utility function U(s) or action-utility function Q(s,a)
- This does not work if the state space is really large or continuous
- Alternative idea: approximate the utilities or Q values using parametric functions and automatically learn the parameters:

$$V(s) \approx \hat{V}(s; w)$$

$$Q(s,a) \approx \hat{Q}(s,a; w)$$

Deep Q learning

- Train a deep neural network to output Q values:



Source: [D. Silver]

Deep Q learning

- Regular TD update: “nudge” Q(s,a) towards the target

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(\boxed{R(s) + \gamma \max_{a'} Q(s',a')} - Q(s,a)\right)$$

- Deep Q learning: encourage estimate to match the target by minimizing squared error:

$$L(w) = \left(\underbrace{\boxed{R(s) + \gamma \max_{a'} Q(s',a';w)}}_{\text{target}} - \underbrace{\boxed{Q(s,a;w)}}_{\text{estimate}}\right)^2$$

Deep Q learning

- Regular TD update: "nudge" Q(s,a) towards the target

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(\boxed{R(s) + \gamma \max_{a'} Q(s',a')} - Q(s,a)\right)$$

- Deep Q learning: encourage estimate to match the target by minimizing squared error:

$$L(w) = \left(\underbrace{\boxed{R(s) + \gamma \max_{a'} Q(s',a';w)}}_{\text{target}} - \underbrace{\boxed{Q(s,a;w)}}_{\text{estimate}}\right)^2$$

- Compare to supervised learning:

$$L(w) = \left(y - f(x;w)\right)^2$$

  – Key difference: the target in Q learning is also moving!

Online Q learning algorithm

- Observe experience (s,a,s', r)
- Compute target $y = r + \gamma \max_{a'} Q(s', a'; w)$
- Update weights to reduce the error

$$L = \left(y - Q(s, a; w)\right)^2$$

- Gradient: $\nabla_w L = \left(Q(s, a; w) - y\right) \nabla_w Q(s, a; w)$

- Weight update: $w \leftarrow w - \alpha \nabla_w L$

- This is called *stochastic gradient descent* (SGD)

Dealing with training instability

- Challenges
  - Target values are not fixed
  - Successive experiences are correlated and dependent on the policy
  - Policy may change rapidly with slight changes to parameters, leading to drastic change in data distribution
- Solutions
  - Freeze target Q network
  - Use *experience replay*

Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

Experience replay

- At each time step:
  - Take action a_t according to epsilon-greedy policy
  - Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay memory buffer*
  - Randomly sample *mini-batch* of experiences from the buffer

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$s_t, a_t, r_{t+1}, s_{t+1} \quad \rightarrow$

Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

Experience replay

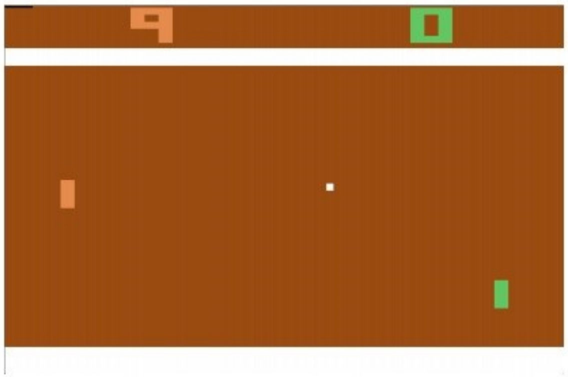
- At each time step:
  - Take action a_t according to epsilon-greedy policy
  - Store experience (s_t, a_t, r_{t+1}, s_{t+1}) in *replay memory buffer*
  - Randomly sample *mini-batch* of experiences from the buffer
  - Perform update to reduce objective function

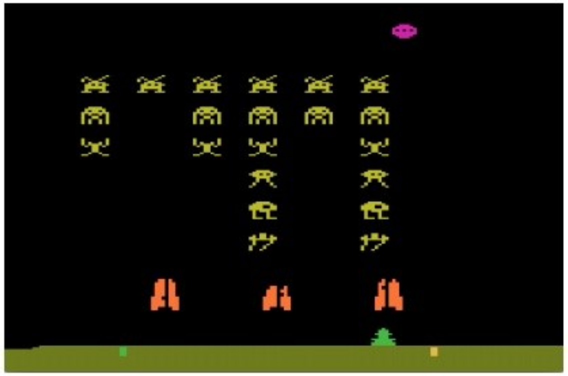
$$\mathbf{E}_{s,a,s'}\left[\left(R(s)+\gamma \max_{a'} \boxed{Q(s',a';w^{-})} - Q(s,a;w)\right)^2\right]$$

Keep parameters of *target network* fixed, update every once in a while

Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

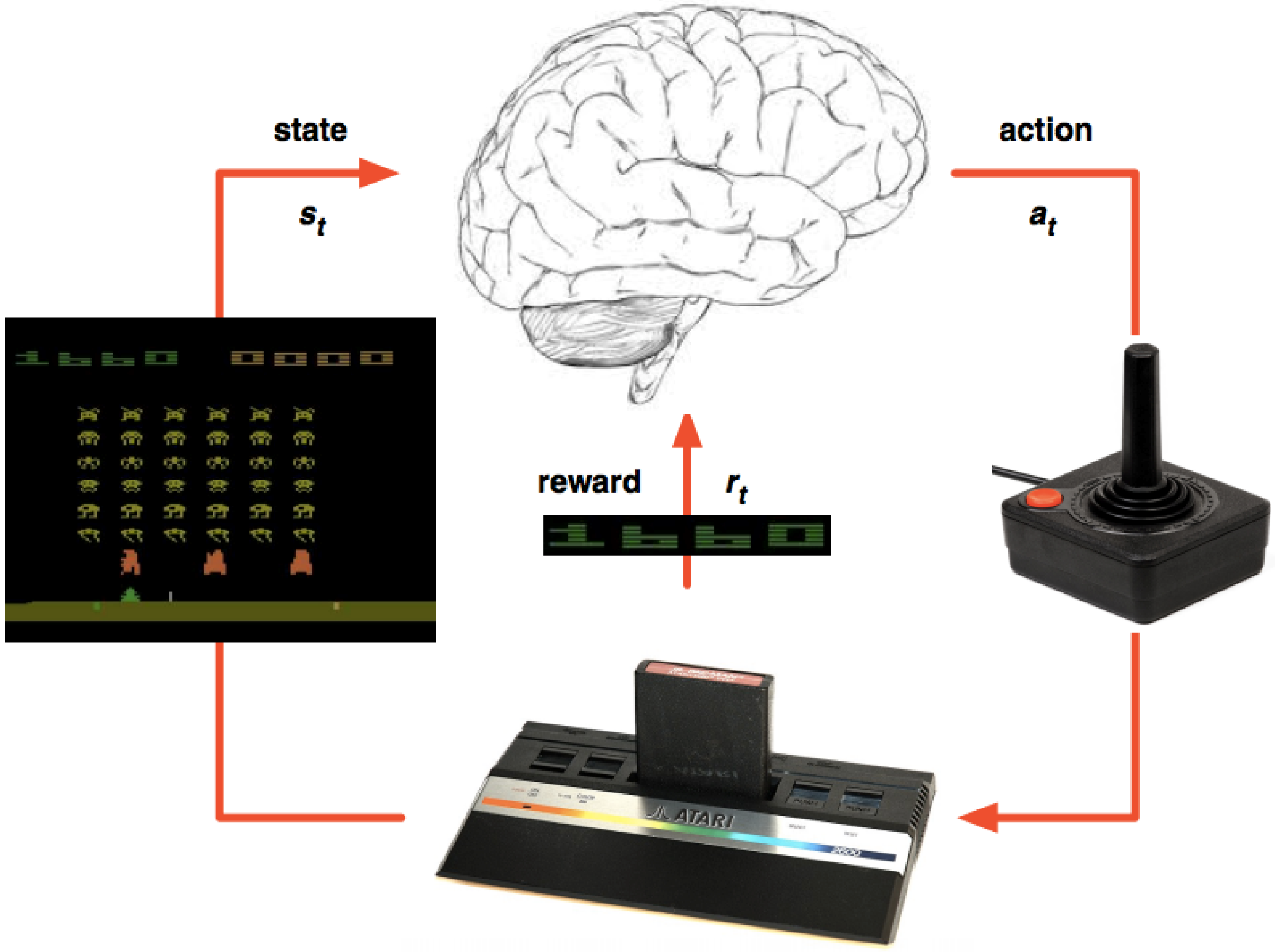
Atari





- Learnt to play from video input
  - from scratch
- Used a complex *neural network!*
  - Considered one of the hardest learning problems solved by a computer.
- More importantly *reproducible!!*

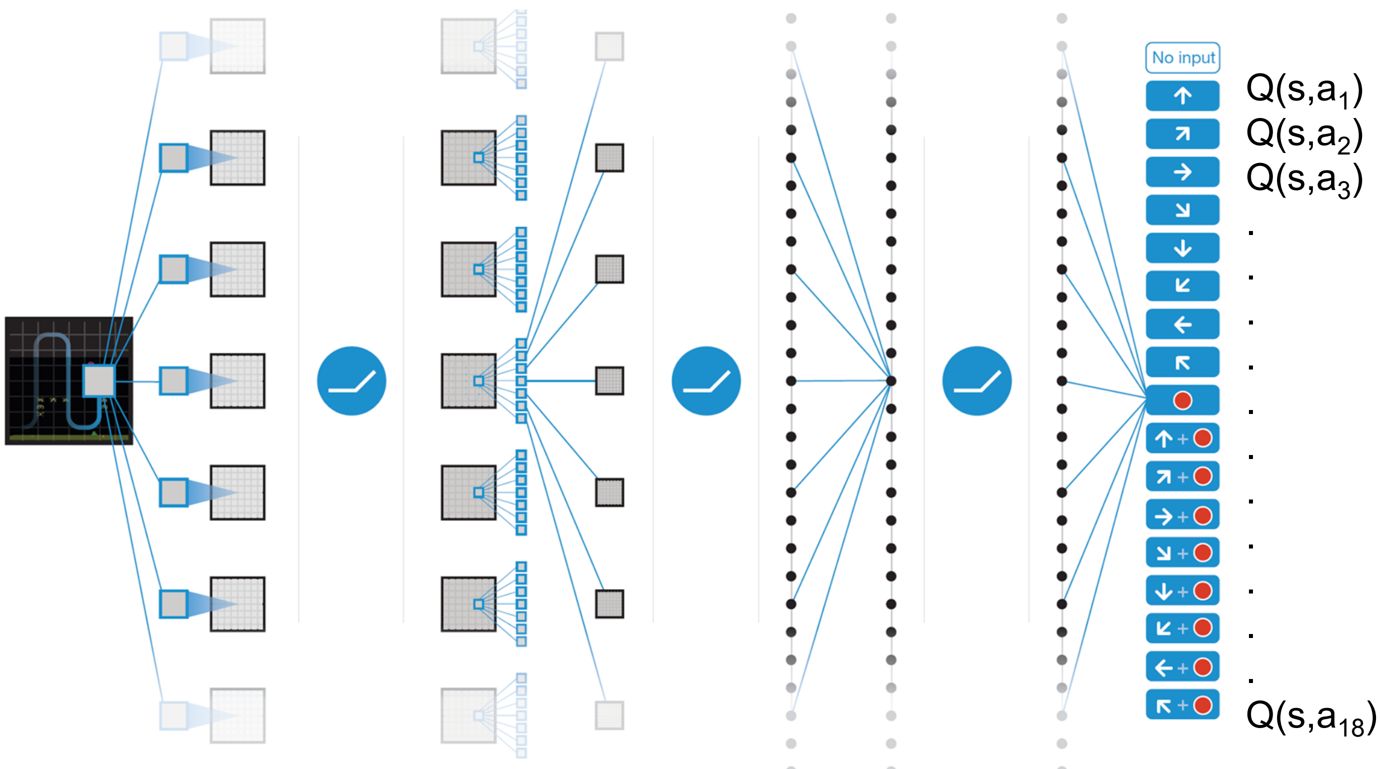
Deep Q learning in Atari



Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

Deep Q learning in Atari

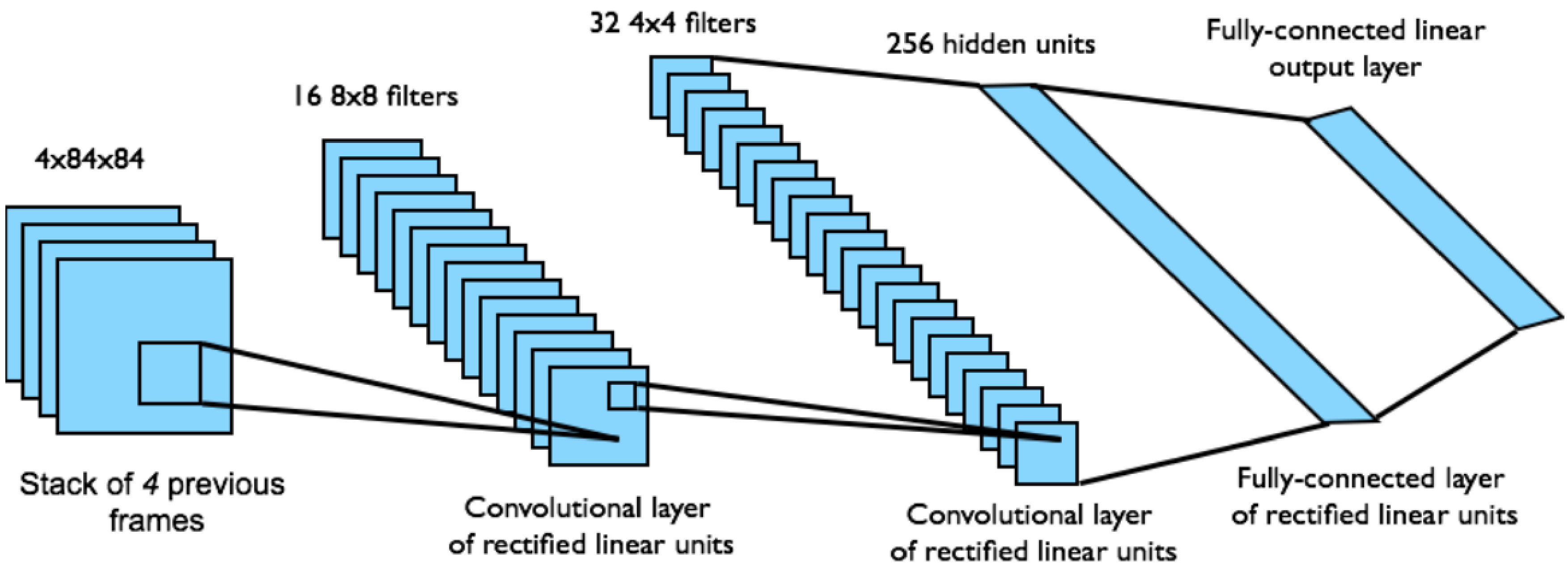
- End-to-end learning of Q(s,a) from pixels s
- Output is Q(s,a) for 18 joystick/button configurations
- Reward is change in score for that step



Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

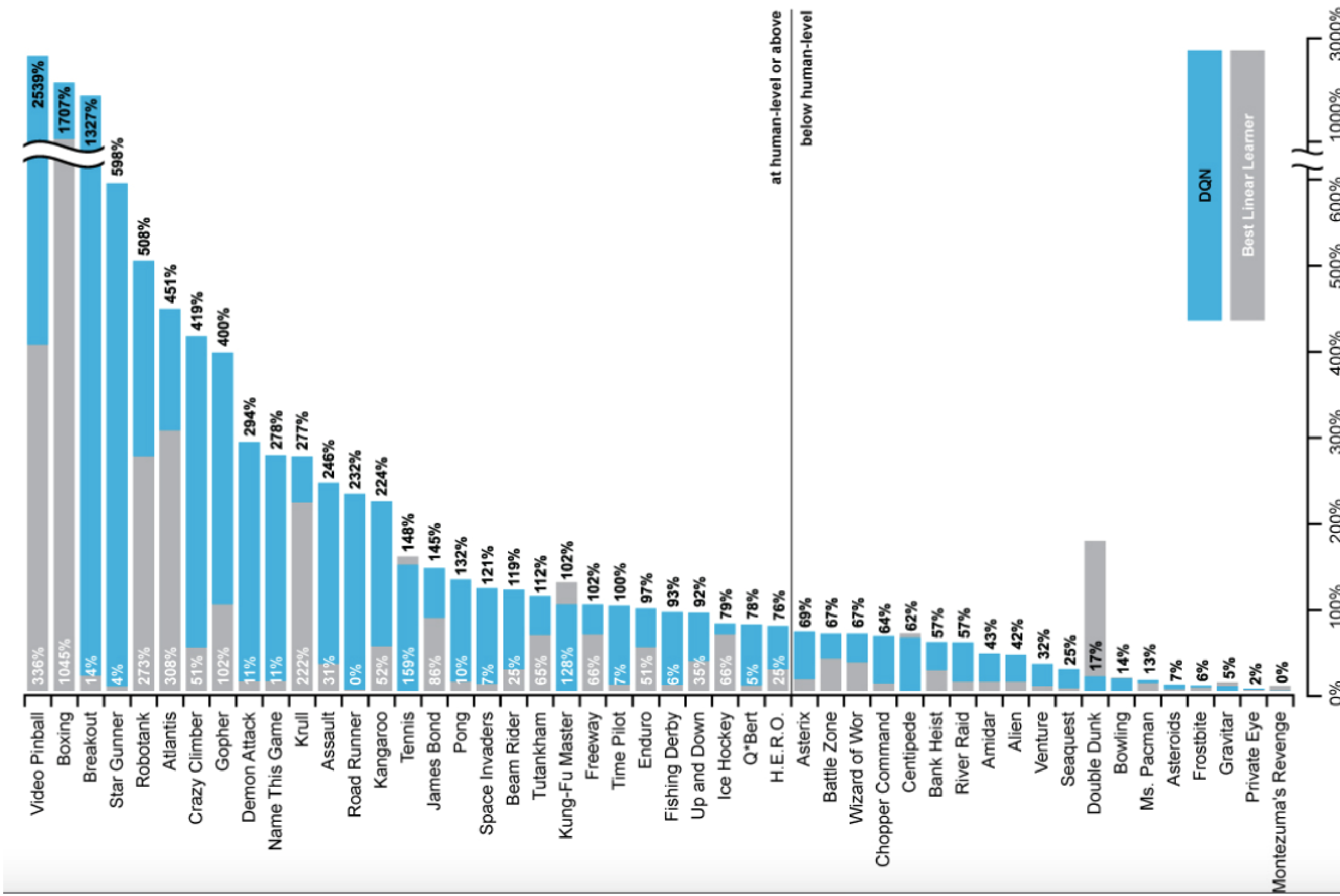
Deep Q learning in Atari

- Input state s is stack of raw pixels from last 4 frames
- Network architecture and hyperparameters fixed for all games



Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

Deep Q learning in Atari



Mnih et al. Human-level control through deep reinforcement learning, *Nature* 2015

Breakout demo

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

Policy gradient methods

- Learning the policy directly can be much simpler than learning Q values
- We can train a neural network to output *stochastic policies*, or probabilities of taking each action in a given state
- *Softmax* policy:

$$\pi(s,a;u) = \frac{\exp\left(f(s,a;u)\right)}{\sum_{a'} \exp\left(f(s,a';u)\right)}$$

Asynchronous Advantage Actor Critic

Mnih et al, 2016

A3C is a recent DRL algorithm for learning policies for sequential decision making on CPUs

It consists of an actor or policy $\pi_{\theta_a}(a_t|s_t)$ which maps states to probability distribution over actions

And a critic or value function $V_{\theta_c}(s_t)$ which evaluates the cumulative expected discounted return from state s_t

The critic tracks the actor and is used to identify *better* actions for a given state.

Avoids the use of replay memory

Actor-critic algorithm

- Define objective function as total discounted reward:

$$J(u) = \mathbf{E}\left[R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots \right]$$

- The gradient for a stochastic policy is given by

$$\nabla_u J = \mathbf{E}\left[\nabla_u \log \underbrace{\boxed{\pi(s,a;u)}}_{\text{Actor network}} \underbrace{\boxed{Q^{\pi}(s,a;w)}}_{\text{Critic network}} \right]$$

- Actor network update: $u \leftarrow u + \alpha \nabla_u J$
- Critic network update: use Q learning (following actor's policy)

Advantage actor-critic

- The raw Q value is less meaningful than whether the reward is better or worse than what you expect to get
- Introduce an *advantage function* that subtracts a baseline number from all Q values

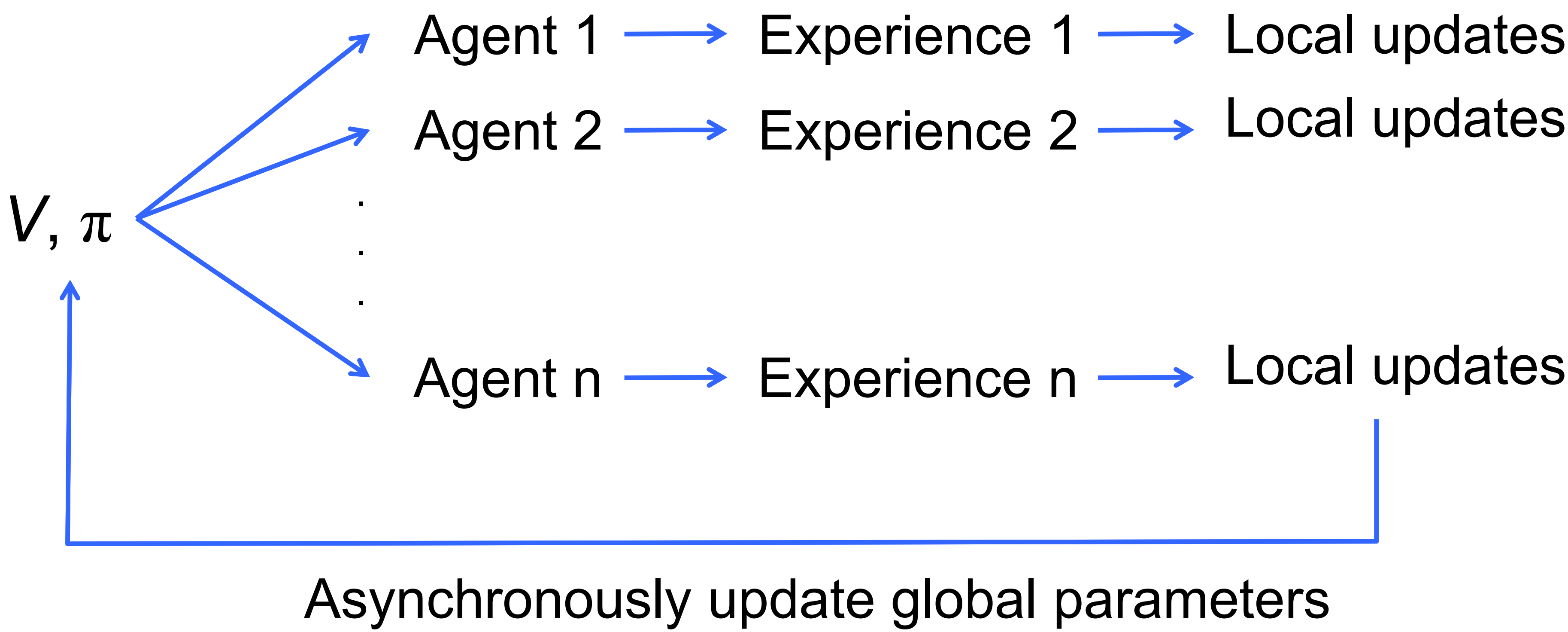
$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

Computed by trajectory

  - Estimate *V* using a *value network*
- Advantage actor-critic:

$$\nabla_u J = \mathbf{E}\left[\nabla_u \log \pi(s,a;u) A^{\pi}(s,a;w)\right]$$

Asynchronous advantage actor-critic (A3C)



Mnih et al. [Asynchronous Methods for Deep Reinforcement Learning](). ICML 2016

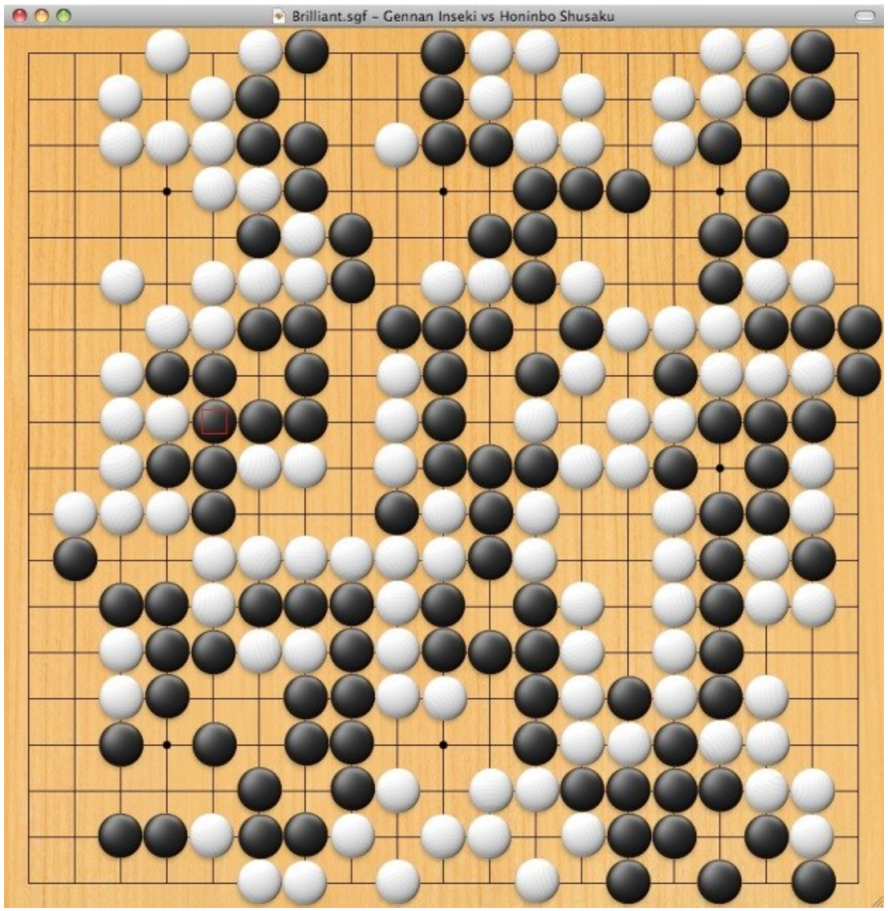
Asynchronous advantage actor-critic (A3C)

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Mean and median human-normalized scores over 57 Atari games

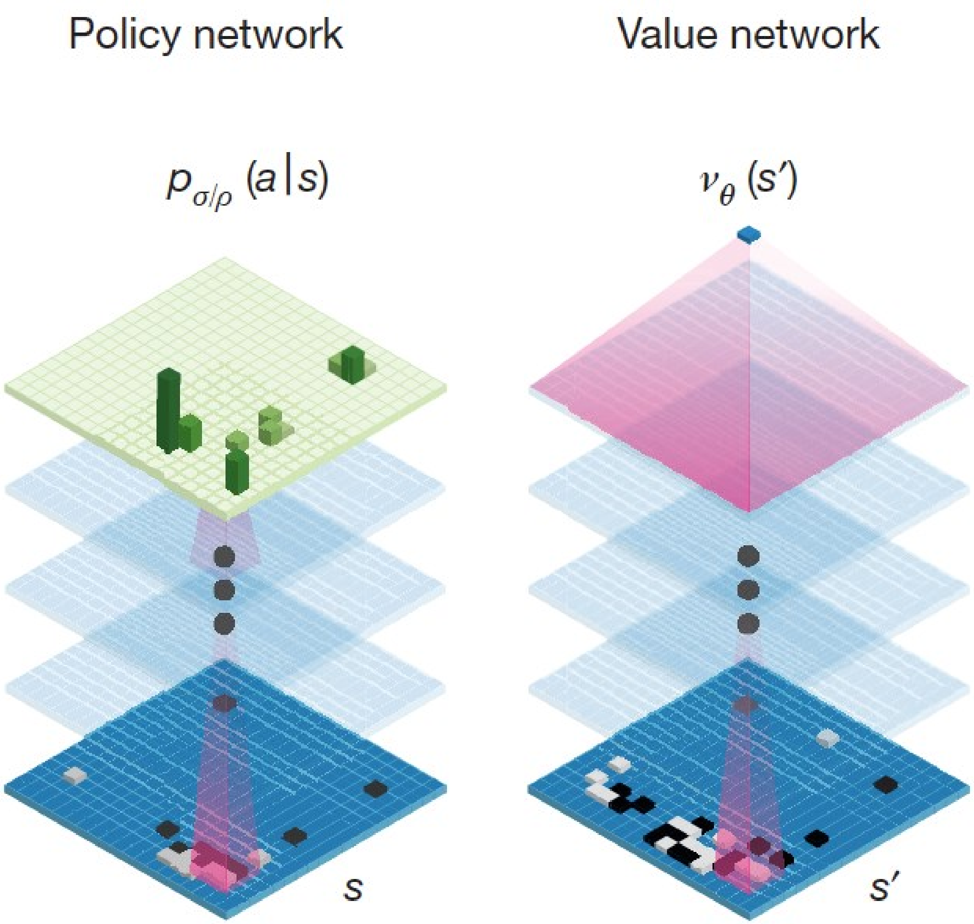
Mnih et al. [Asynchronous Methods for Deep Reinforcement Learning](). ICML 2016

Playing Go



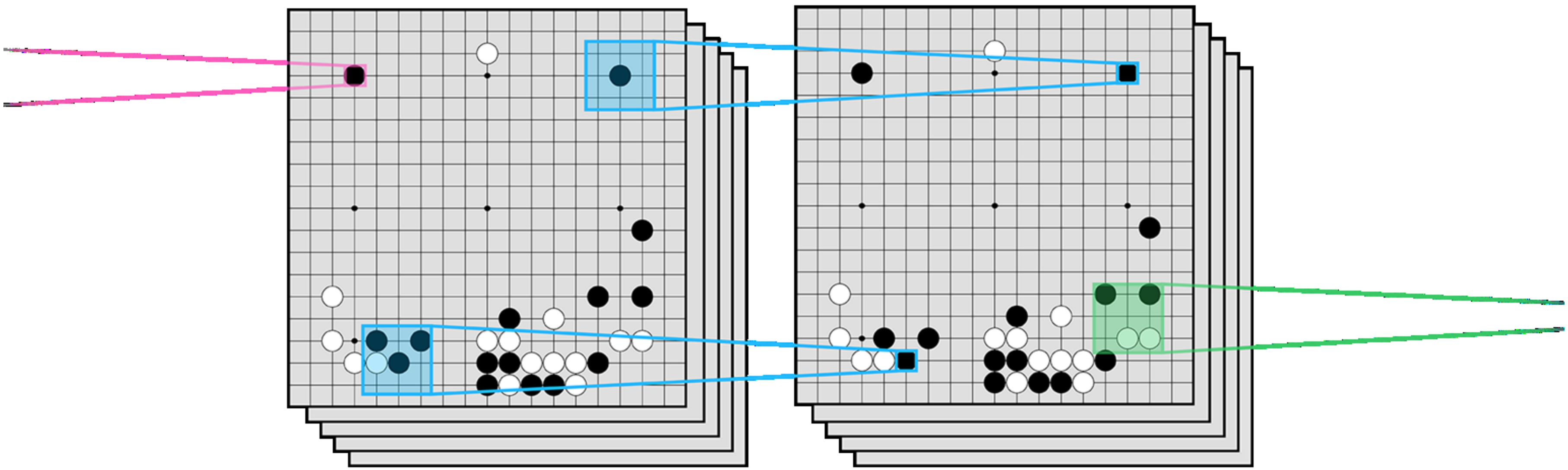
- Go is a known (and deterministic) environment
- Therefore, learning to play Go involves solving a known MDP
- Key challenges: huge state and action space, long sequences, sparse rewards

Review: AlphaGo



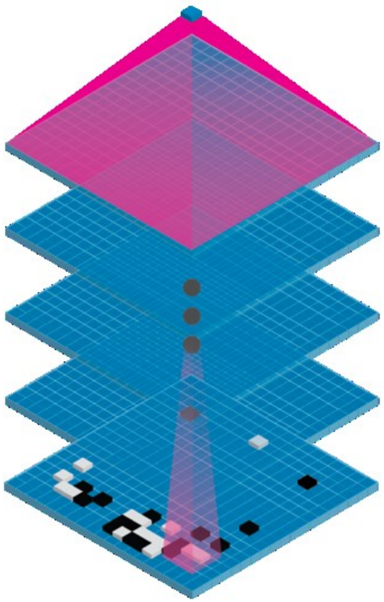
- **Policy network:** initialized by supervised training on large amount of human games
- **Value network:** trained to predict outcome of game based on self-play
- Networks are used to guide Monte Carlo tree search (MCTS)

D. Silver et al., [Mastering the Game of Go with Deep Neural Networks and Tree Search](), Nature 529, January 2016

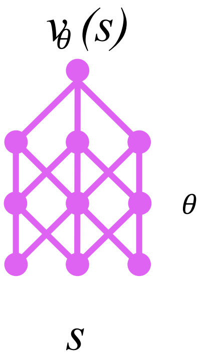


Value network

Evaluation

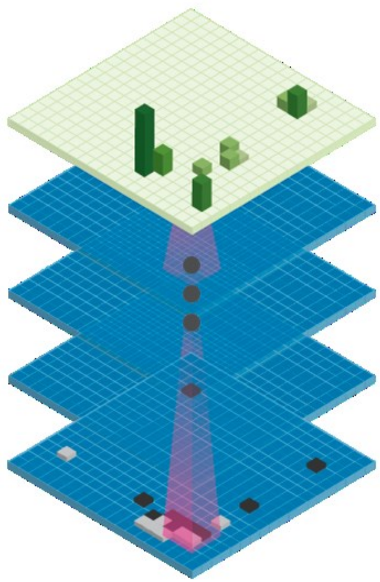


Position

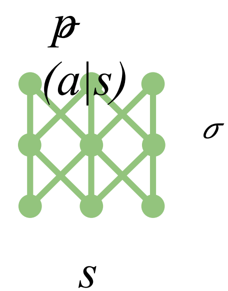


Policy network

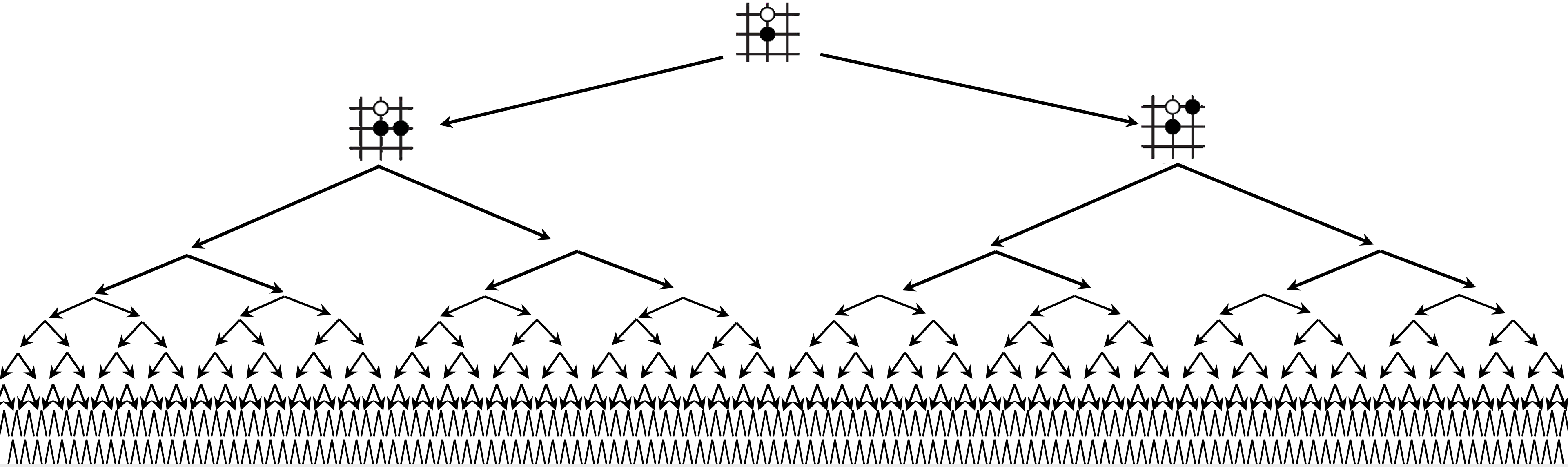
Move probabilities



Position

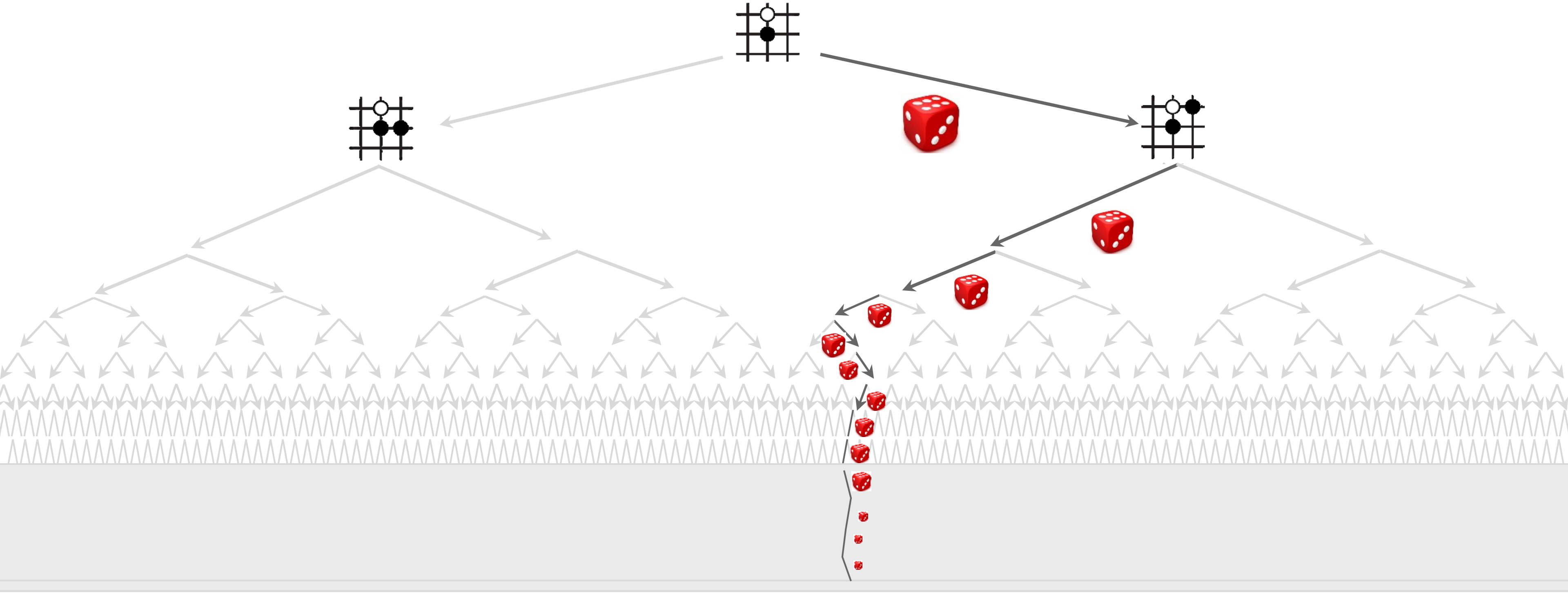


Exhaustive search

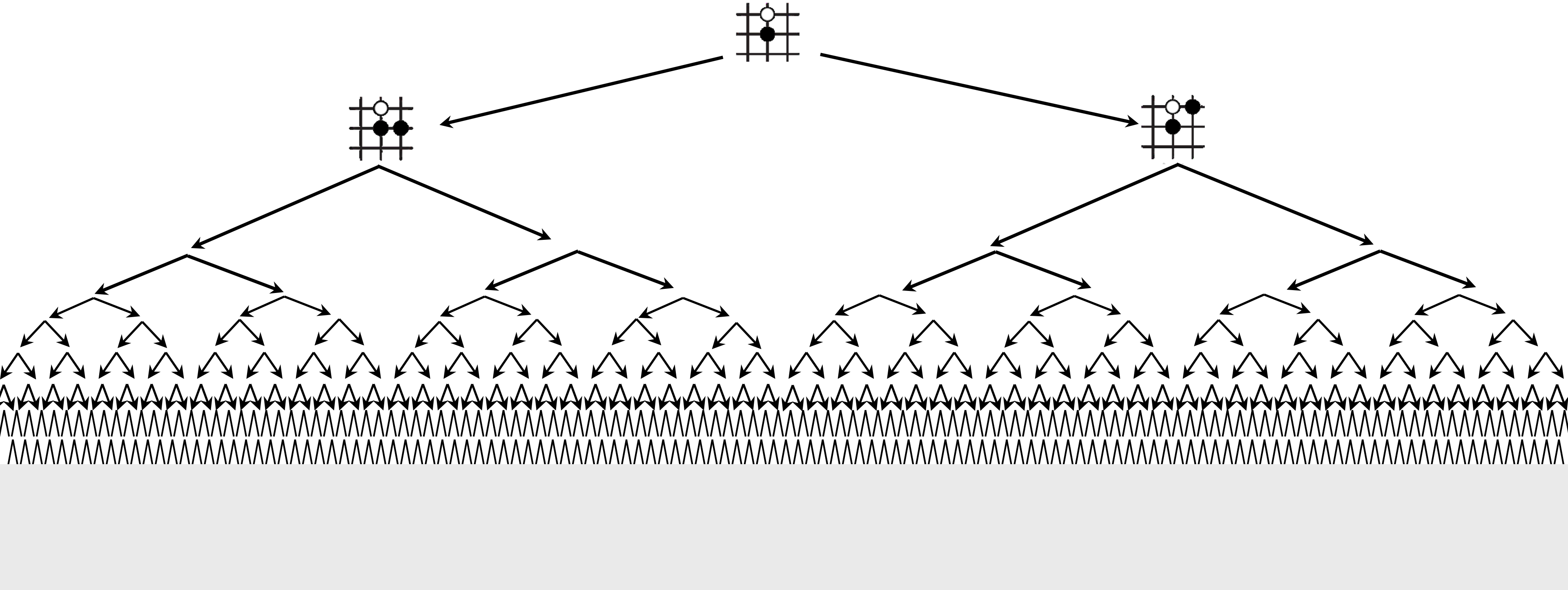


Google DeepMind

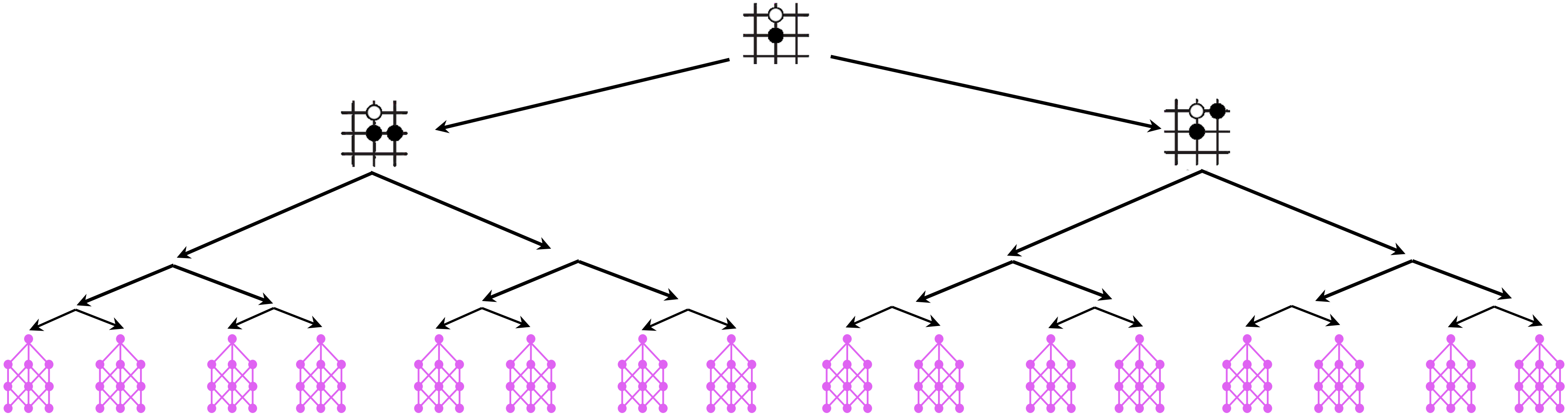
Monte-Carlo rollouts



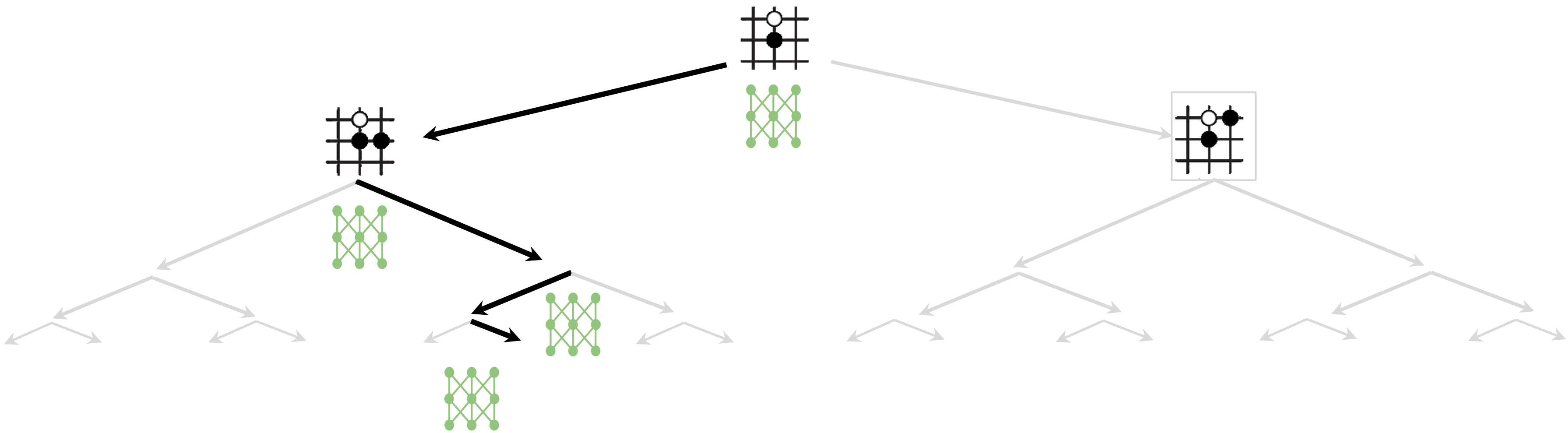
Reducing depth with value network



Reducing depth with value network

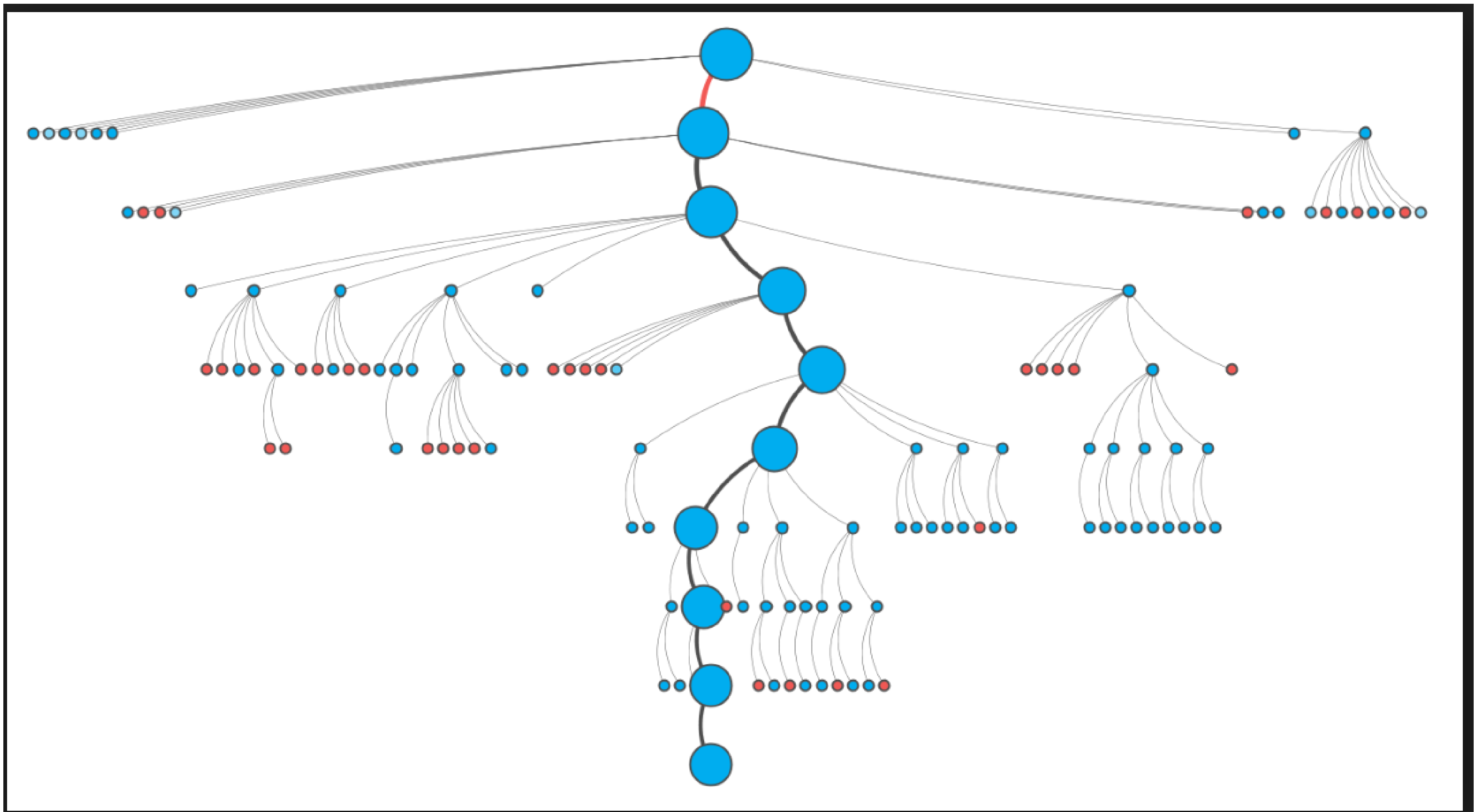


Reducing breadth with policy network

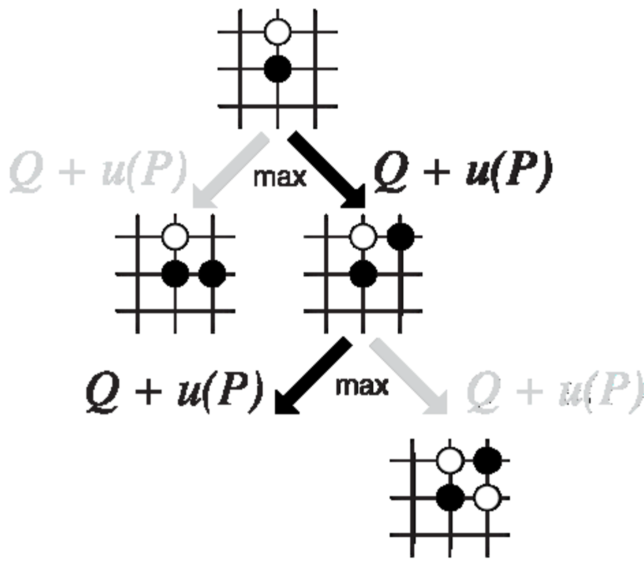


Monte Carlo Tree Search

- Figure from https://codepoke.net/2015/03/03/walk-the-line-search-techniques-evaluation-functions/



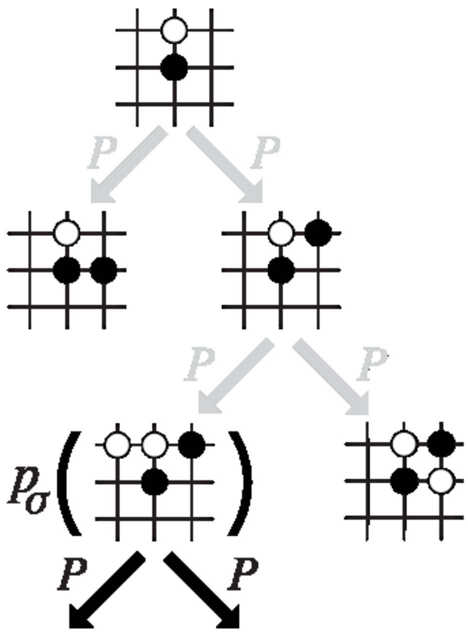
Monte-Carlo tree search in AlphaGo: **selection**



P prior probability
Q action value

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

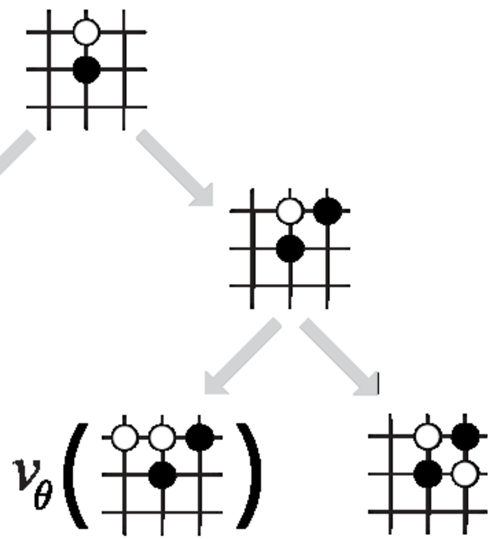
Monte-Carlo tree search in AlphaGo: **expansion**



p_σ Policy network
P prior probability

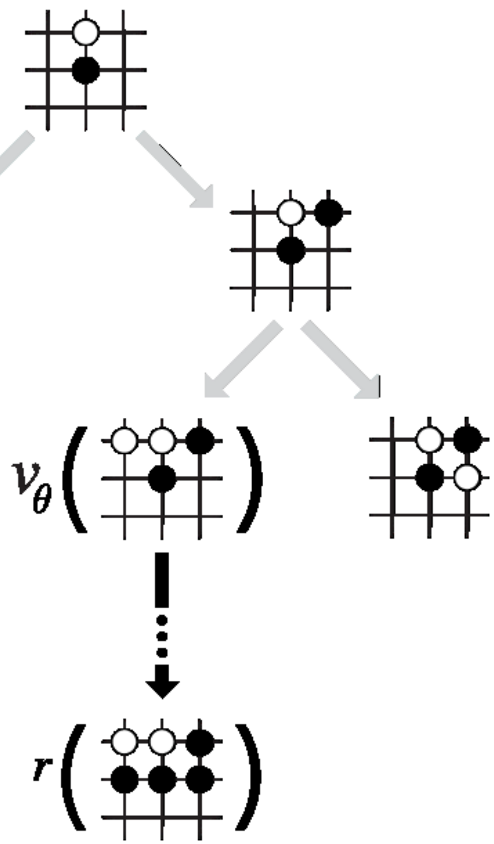
Google DeepMind

Monte-Carlo tree search in AlphaGo: **evaluation**



v_θ Value network

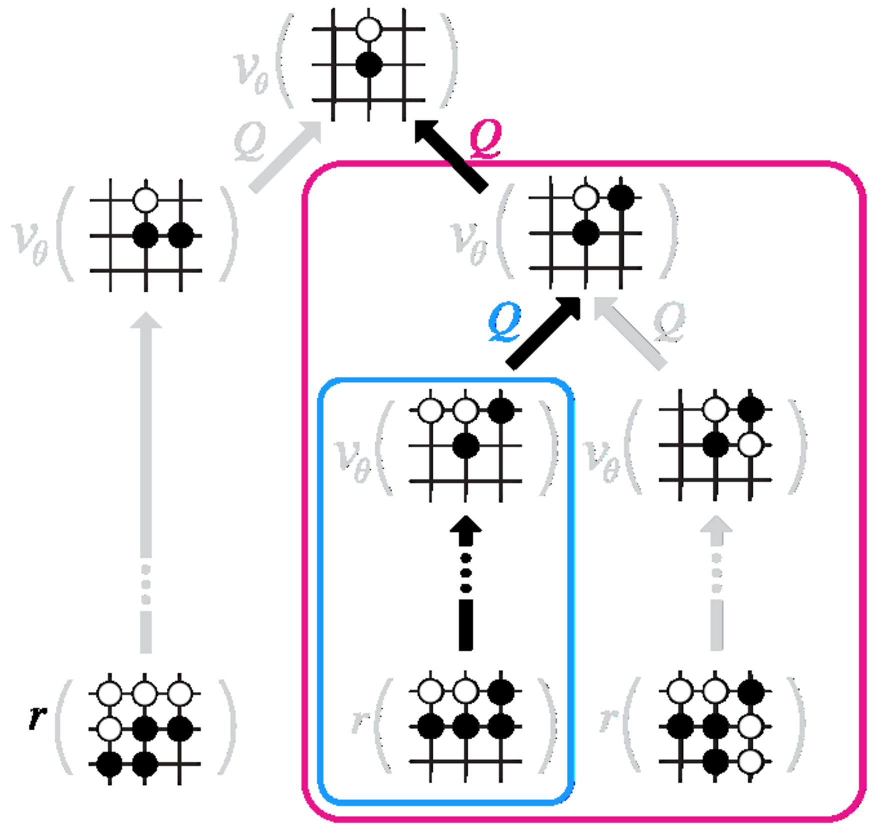
Monte-Carlo tree search in AlphaGo: **rollout**



v_θ Value network
r Game scorer

Google DeepMind

Monte-Carlo tree search in AlphaGo: **backup**



Q Action value

v_θ Value network

r Game scorer

Google DeepMind

AlphaGo Zero

- A fancier architecture (deep residual networks)
- No hand-crafted features at all
- A single network to predict both value and policy
- Train network entirely by self-play, starting with random moves
- Uses MCTS inside the reinforcement learning loop, not outside

D. Silver et al., Mastering the Game of Go without Human Knowledge, Nature 550, October 2017

https://deepmind.com/blog/alphago-zero-learning-scratch/

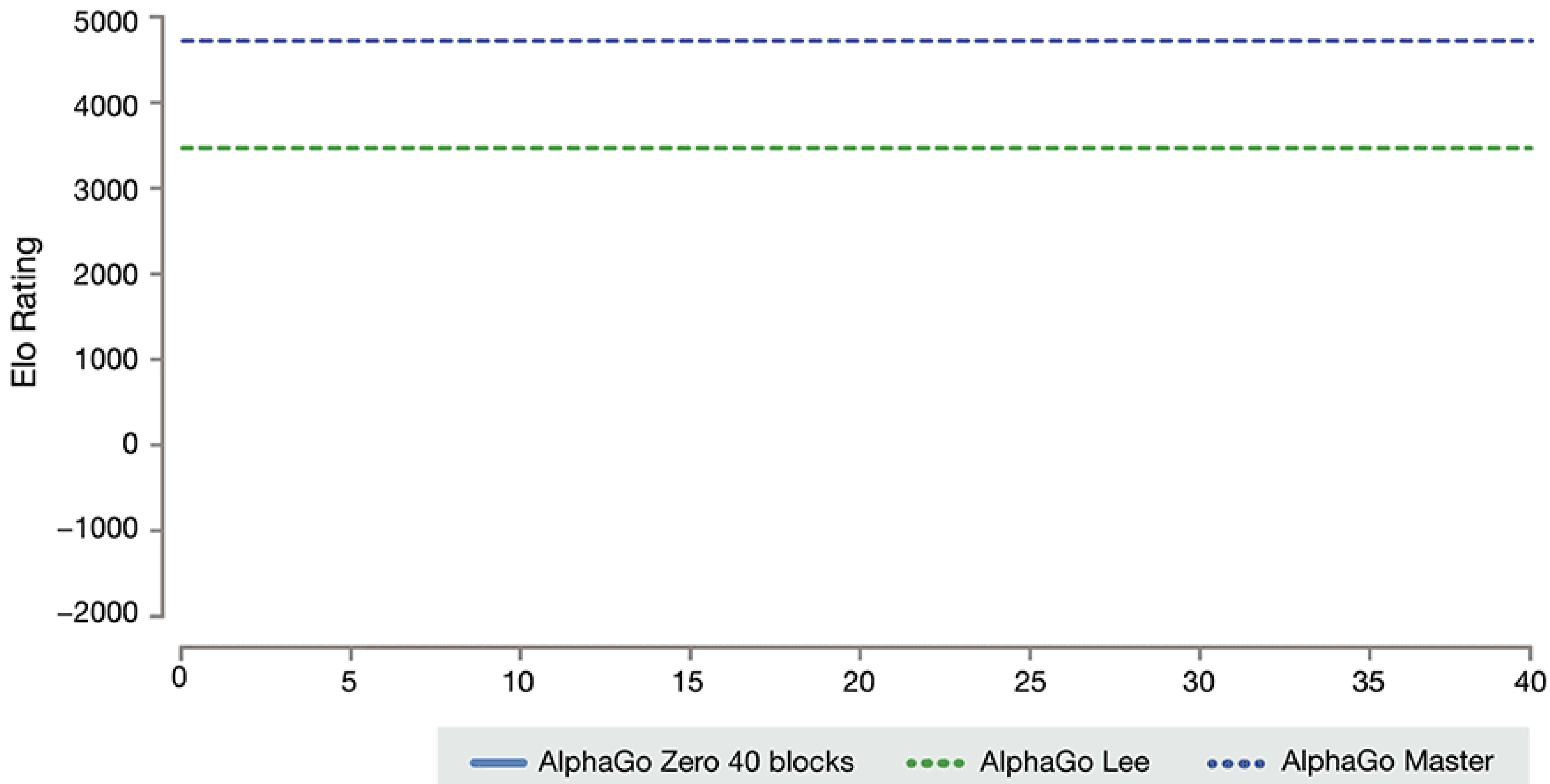
AlphaGo Zero

- Given a position, neural network outputs both move probabilities *P* and value *V* (probability of winning)
- In each position, MCTS is conducted to return refined move probabilities π and game winner *Z*
- Neural network parameters are updated to make *P* and *V* better match π and *Z*
- Reminiscent of **policy iteration**: self-play with MCTS is *policy evaluation*, updating the network towards MCTS output is *policy improvement*

D. Silver et al., [Mastering the Game of Go without Human Knowledge](https://deepmind.com/blog/alphago-zero-learning-scratch/), Nature 550, October 2017

https://deepmind.com/blog/alphago-zero-learning-scratch/

AlphaGo Zero

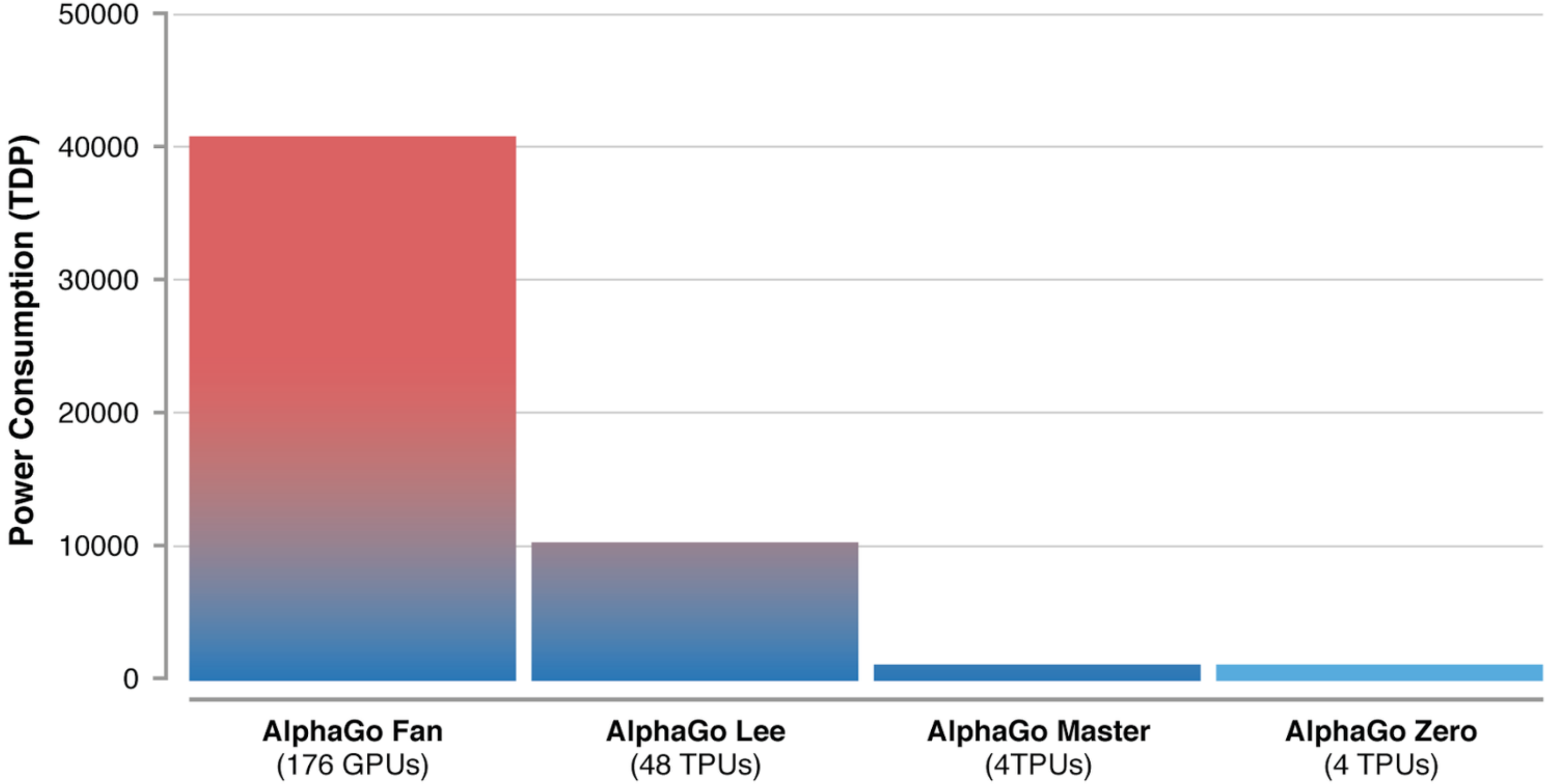


D. Silver et al., Mastering the Game of Go without Human Knowledge, Nature 550, October 2017

https://deepmind.com/blog/alphago-zero-learning-scratch/

AlphaGo Zero

It's also more efficient than older engines!



D. Silver et al., Mastering the Game of Go without Human Knowledge, Nature 550, October 2017

https://deepmind.com/blog/alphago-zero-learning-scratch/

Summary

- Deep Q learning
- Policy gradient methods
  - Actor-critic
  - Advantage actor-critic
  - A3C
- Policy iteration for AlphaGo
- Imitation learning for visuomotor policies