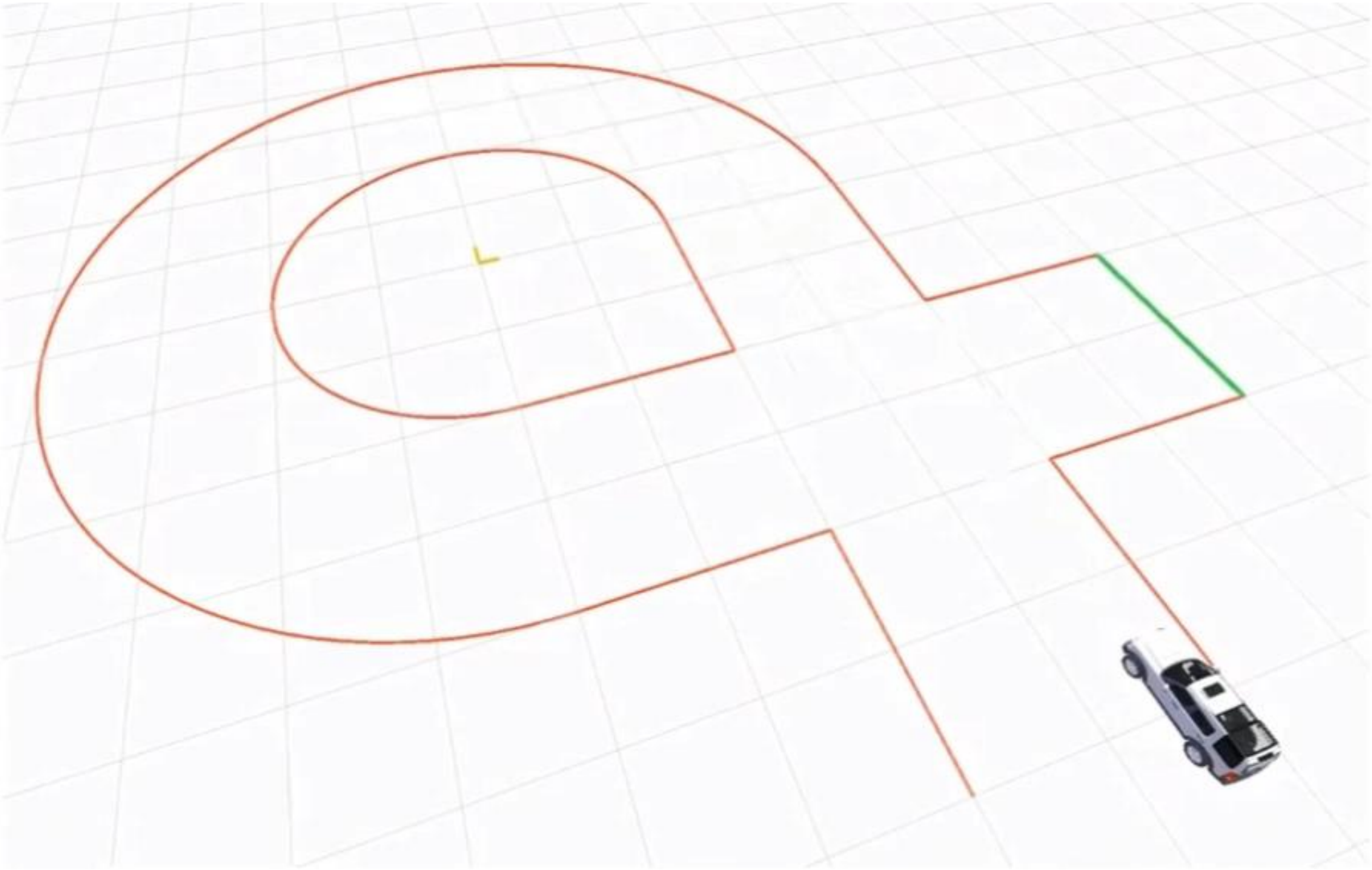
REINFORCEMENT LEARNING

<table>
<tr><td colspan="5" rowspan="4"></td><td colspan="6">Tasks</td></tr>
<tr><td colspan="5">ADAS</td><td></td></tr>
<tr><td colspan="3">Self Driving</td><td></td><td></td><td></td></tr>
<tr><td>Localization</td><td>Perception</td><td>Planning/ Control</td><td>Driver state</td><td>Vehicle Diagnosis</td><td>Smart factory</td></tr>
<tr><td rowspan="8">Methods</td><td rowspan="2">Traditional</td><td colspan="3">Non-machine Learning</td><td>GPS, SLAM</td><td></td><td>Optimal control</td><td></td><td></td><td></td></tr>
<tr><td rowspan="6">Machine-Learning based method</td><td rowspan="4">Supervised</td><td>SVM MLP</td><td></td><td>Pedestrian detection (HOG+SVM)</td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="5">Deep-Learning based</td><td>CNN</td><td></td><td>Detection/ Segmentation/Classification</td><td>End-to-end Learning</td><td></td><td></td><td></td></tr>
<tr><td>RNN (LSTM)</td><td></td><td>Dry/wet road classification</td><td>End-to-end Learning</td><td>Behavior Prediction/ Driver identification</td><td></td><td>*</td></tr>
<tr><td>DNN</td><td></td><td></td><td></td><td></td><td>*</td><td>*</td></tr>
<tr><td colspan="2">Reinforcement</td><td></td><td></td><td>*</td><td></td><td></td><td></td></tr>
<tr><td colspan="2">Unsupervised</td><td></td><td></td><td></td><td></td><td></td><td>*</td></tr>
</table>

Planning



Planning

Hope for Reinforcement Learening

- Supervised Learning
  - Neural networks are great at memorization and not (yet) great at reasoning

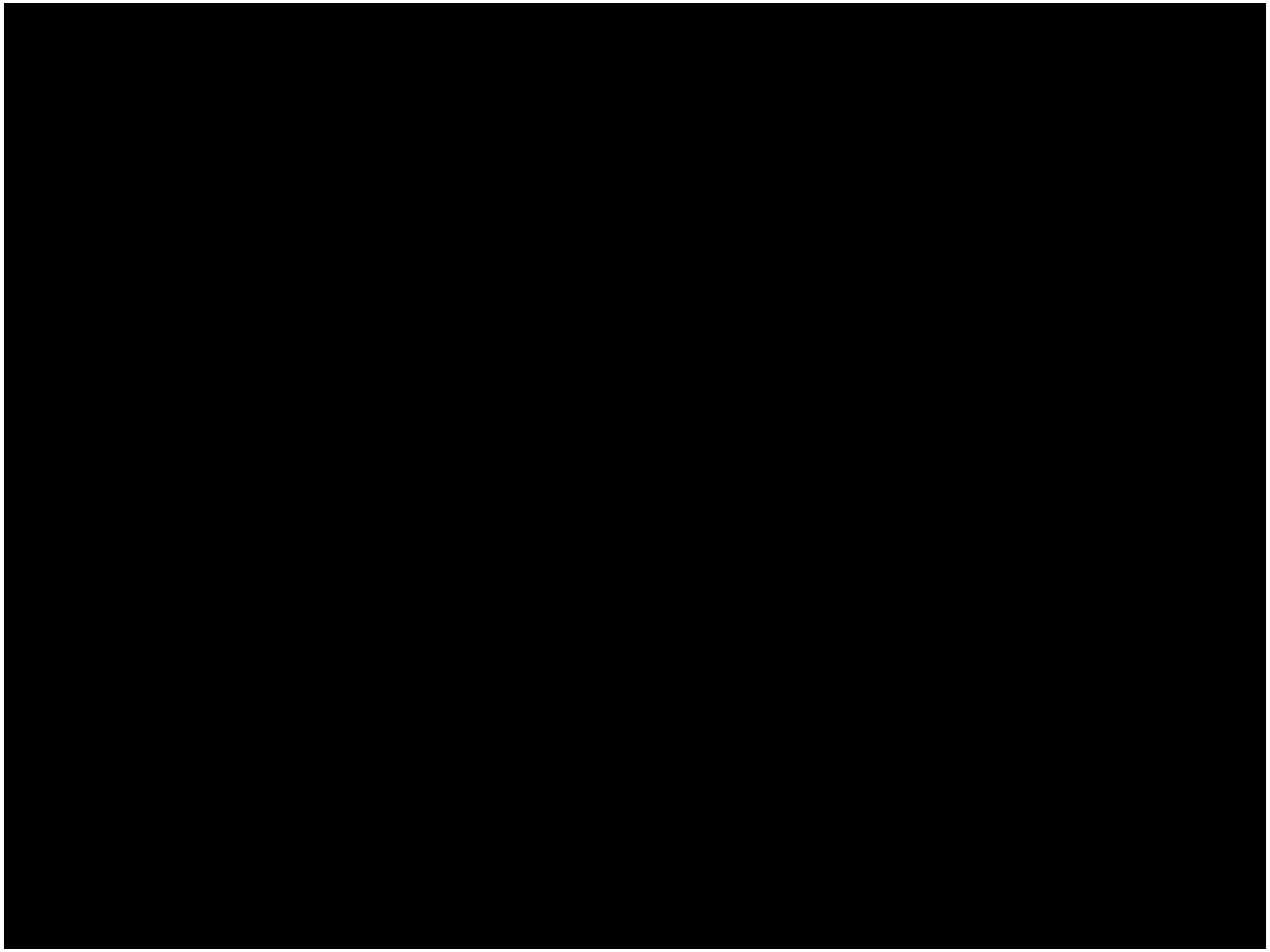
- Reinforcement Learning
  - Brute-force propagation of outcomes to knowledge about states and actions.

- Hope for Deep Learning + Reinforcement Learning
  - General purpose artificial intelligence through efficient generalizable learning of the optimal thing to do given a formalized set of actions and states

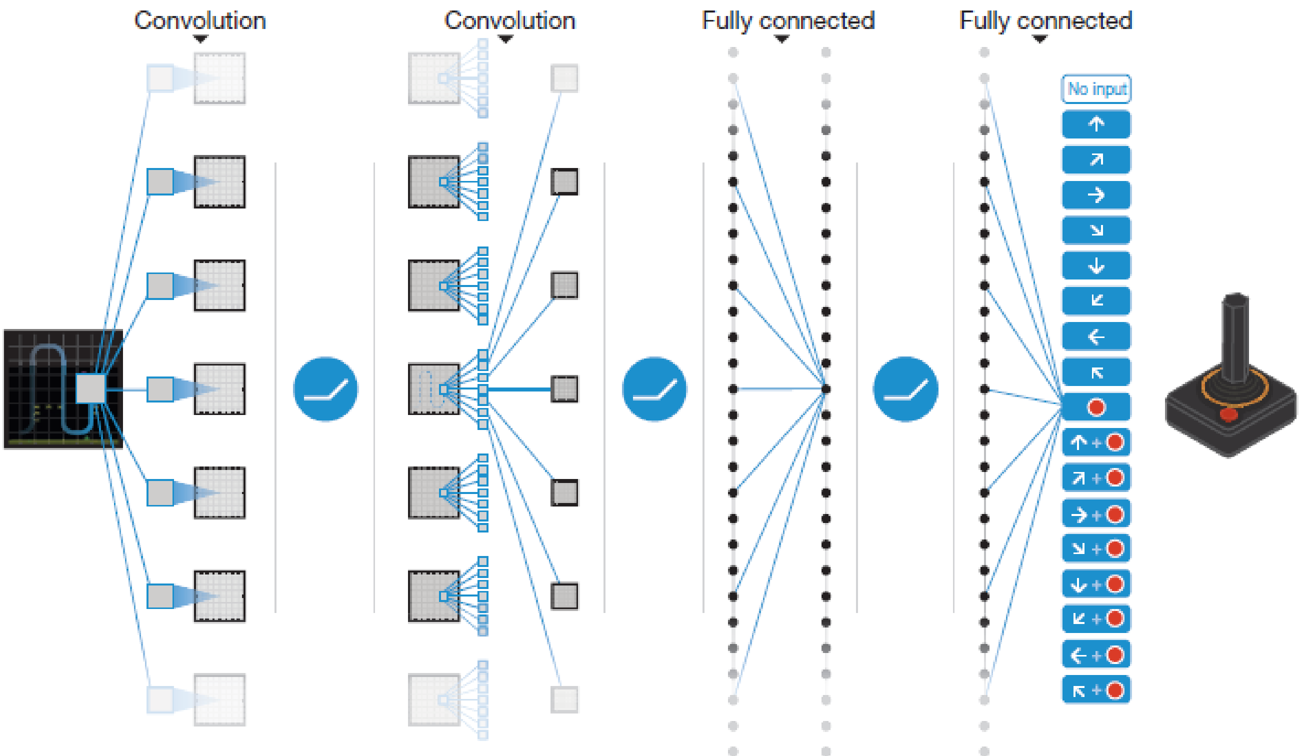
INTRODUCTION TO REINFORCEMENT LEARNING

					Tasks					
					ADAS					
					Self Driving					
					Localization	Perception	Planning/ Control	Driver state	Vehicle Diagnosis	Smart factory
Methods	Traditional	Non-machine Learning			GPS, SLAM		Optimal control			
Methods	Traditional	Machine-Learning based method	Supervised	SVM MLP		Pedestrian detection (HOG+SVM)				
Methods	Deep-Learning based	Machine-Learning based method	Supervised	CNN		Detection/ Segmentation/Classification	End-to-end Learning			
Methods	Deep-Learning based	Machine-Learning based method	Supervised	RNN (LSTM)		Dry/wet road classification	End-to-end Learning	Behavior Prediction/ Driver identification		*
Methods	Deep-Learning based	Machine-Learning based method	Supervised	DNN					*	*
Methods	Deep-Learning based	Machine-Learning based method	Reinforcement				*			
Methods	Deep-Learning based	Machine-Learning based method	Unsupervised							*

DeepMind's DQN playing Breakout



Deep Q-network



How to train?

- In the supervised learning setting, we have to collect training samples and train the network!
  - Training samples: $\{(x_i, y_i)\}$

$(x_i, y_i) = ($  ,  $)$

Game state

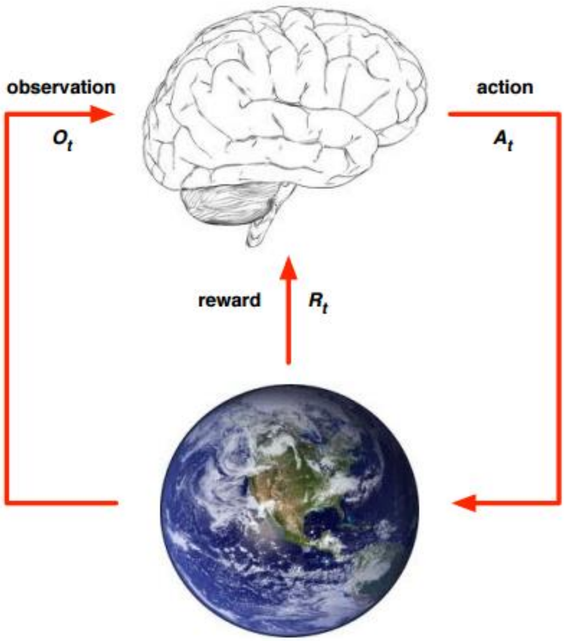
Joystick control

- In the reinforcement setting,
  - ???

INTRODUCTION TO REINFORCEMENT LEARNING

Reinforcement Learning

- Reinforcement learning is an area of machine learning concerned with how software **agents** ought to take actions in an **environment** so as to maximize some notion of **cumulative reward**.



Atari Example

Reinforcement Learning

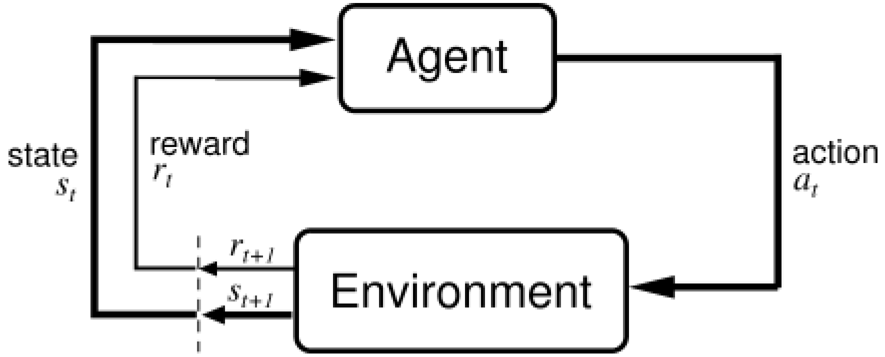
- Learning from interaction
- Goal-oriented learning
- Learning about, from, and while interacting with an external environment

Key Features of RL

- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward (sacrifice short-term gains for greater long-ter m gains)
- The need to **explore and exploit**

Reinforcement Learning Setting

- S – set of states
- A – set of actions
- $R : S \times A \rightarrow R$ – reward for given state and action.

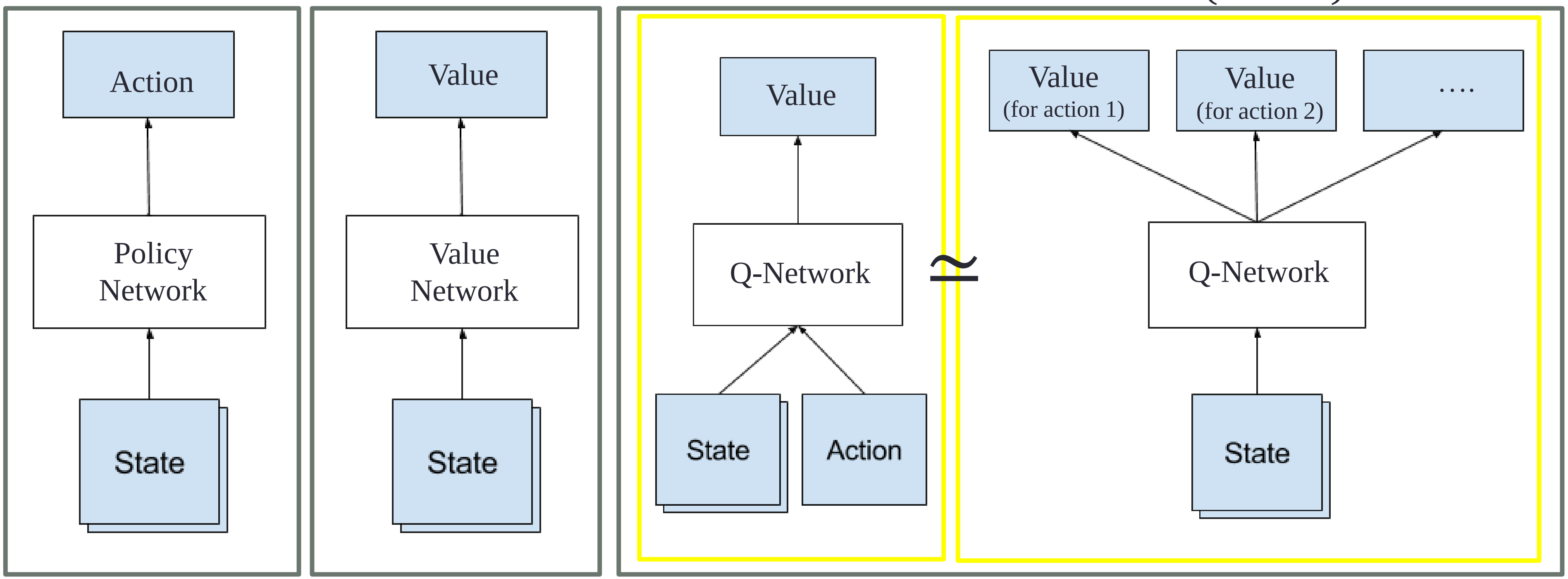


Reinforcement Learning Terms

- Policy: $a = \pi(s)$
  - A policy π is a mapping from each state, $s \in S$, to an action $a \in A(s)$
- (State-) Value function: $V^{\pi}(s)$
  - the expected future reward given a current state $s \in S$ and policy π
- Q-function (Action-value function): $Q^{\pi}(s, a)$
  - the expected future reward given a state action pair, (s, a), and policy π

Reinforcement Learning Terms

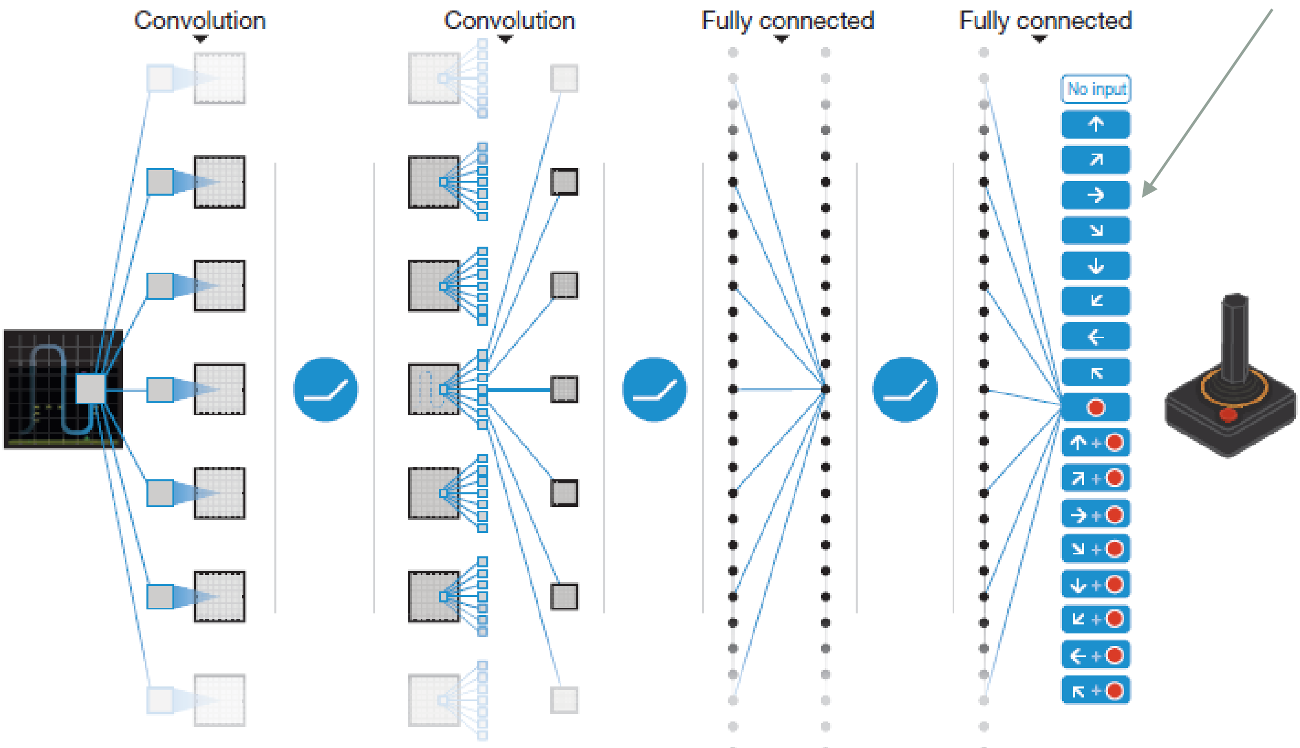
If $\#(action)$ is small



Value = expected reward

Deep Q-network

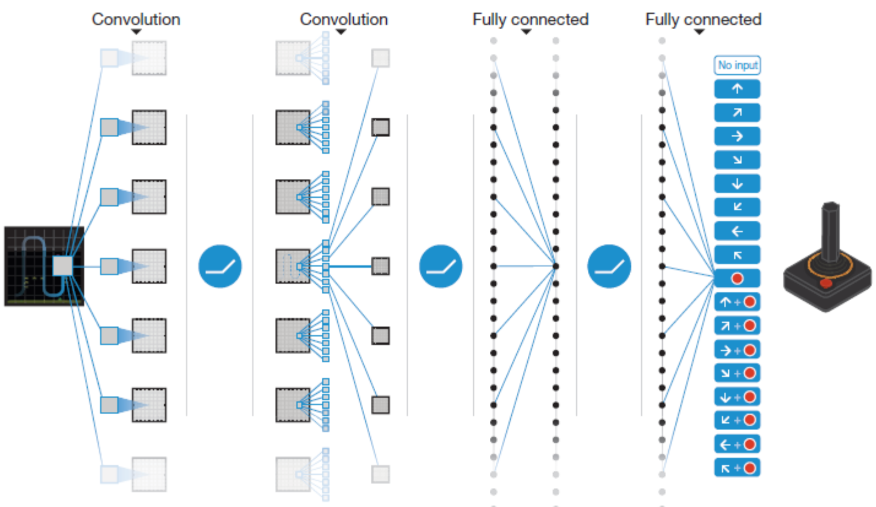
Network output: expected future reward when taking each action

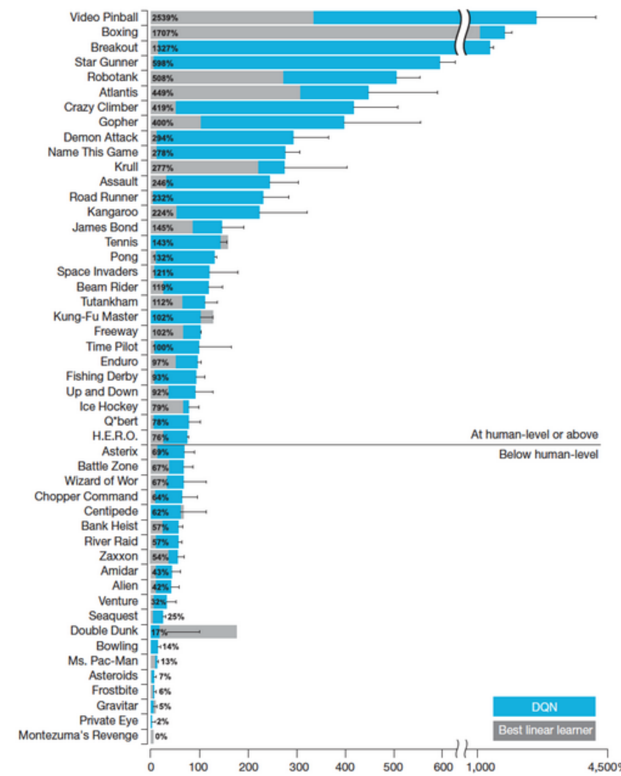


LEARNING METHOD: DEEP Q-LEARNING

Deep Q-network

- From pixels to Actions: Human-level control through Deep Reinforcement Learning





How to train: Q-Learning

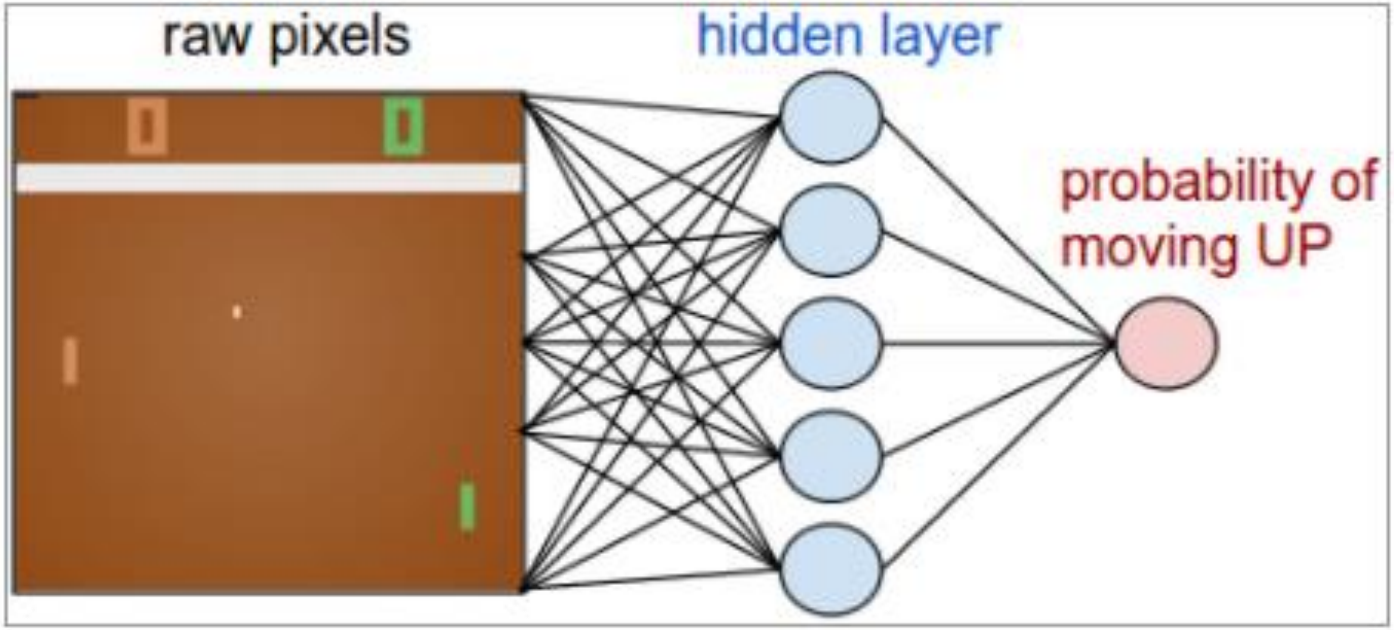
- Optimal Q-values should obey Bellman equation
  - Bellman equation for $Q^*(s,a)$
    - $Q^*(s,a) = \sum_{s'} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma \max_{a'} Q^*(s',a') \right]$
    - $Q^*(s,a;w) = \sum_{s'} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma \max_{a'} Q^*(s',a';w) \right] = r + \gamma \max_{a'} Q^*(s',a';w)$
- Treat right hand side $r + \gamma \max_{a'} Q^*(s',a';w)$ as a target
- Minimize MSE loss by stochastic gradient descent

$$l = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \right)^2$$

- Converges to Q^* using table lookup representation
- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

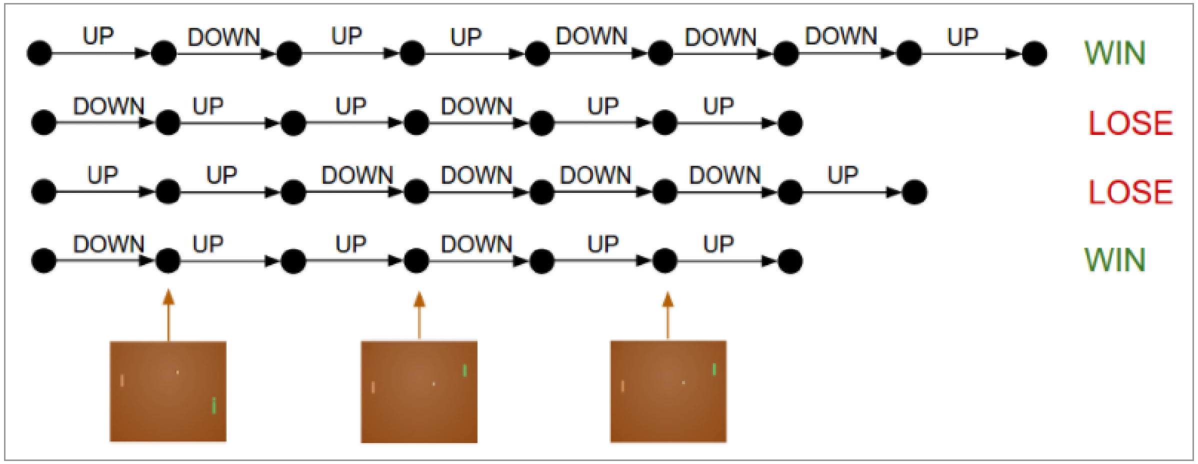
LEARNING METHOD: POLICY GRADIENT

Policy Network



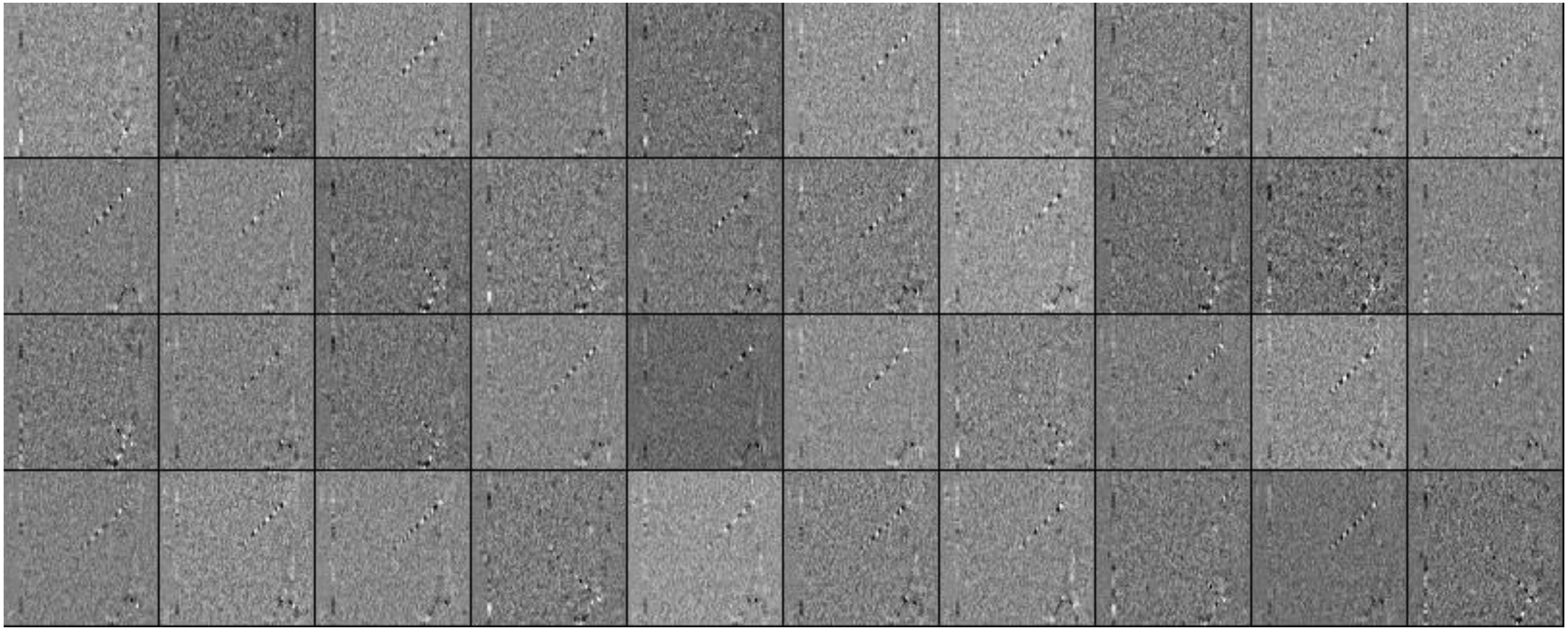
Policy Gradient Method

- Random Initialization
- Repeat
  - Generate samples (run the policy)
  - Policy improvement
    - Reward-weighted gradient learning (similar to the supervised learning)



Cartoon diagram of 4 games. Each black circle is some game state (three example states are visualized on the bottom), and each arrow is a transition, annotated with the action that was sampled. In this case we won 2 games and lost 2 games. With Policy Gradients we would take the two games we won and slightly encourage every single action we made in that episode. Conversely, we would also take the two games we lost and slightly discourage every single action we made in that episode.

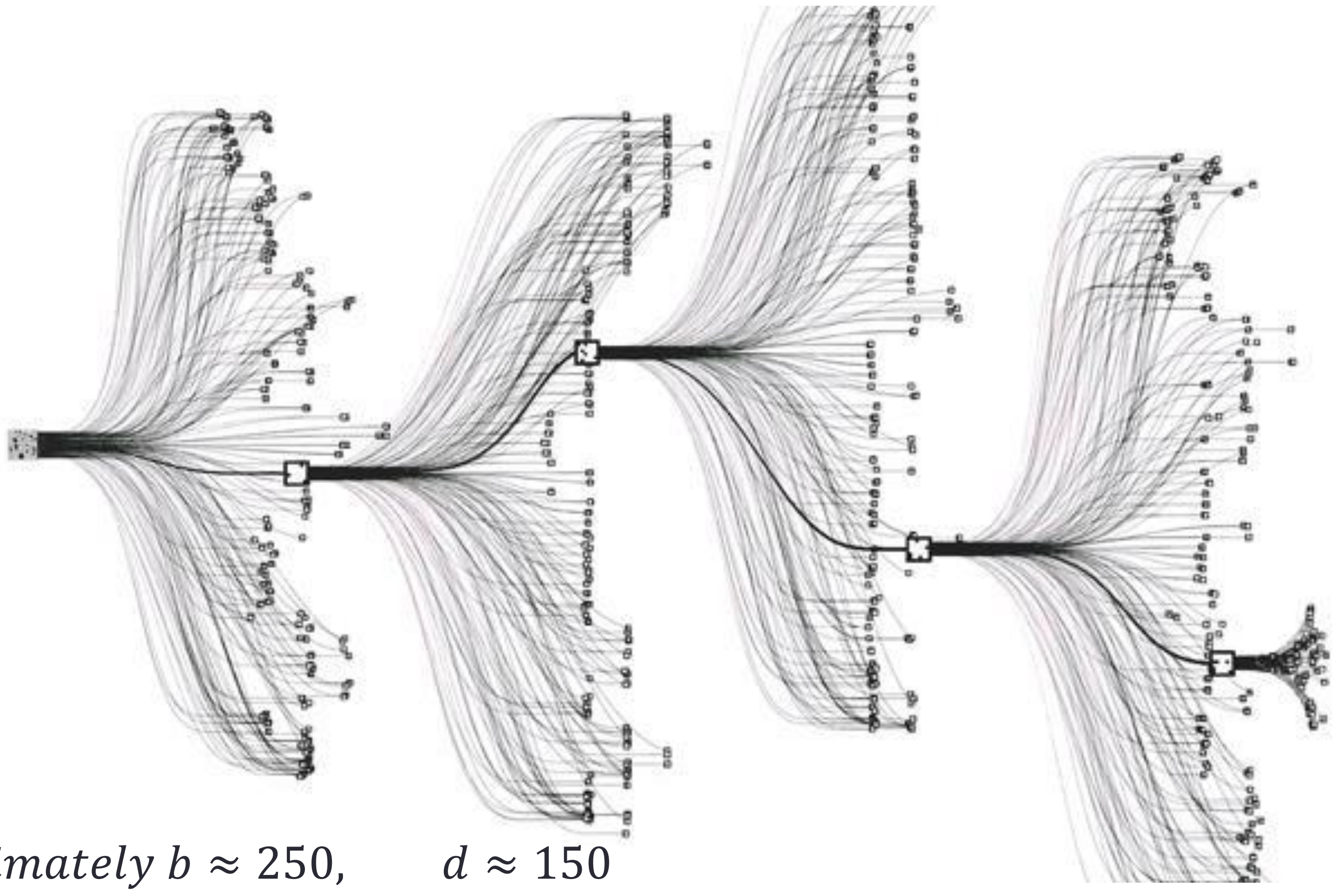
40 (out of 200) neurons



CASE STUDY: ALPHAGO

바둑

Search space



$Approximately\ b \approx 250, \qquad d \approx 150$

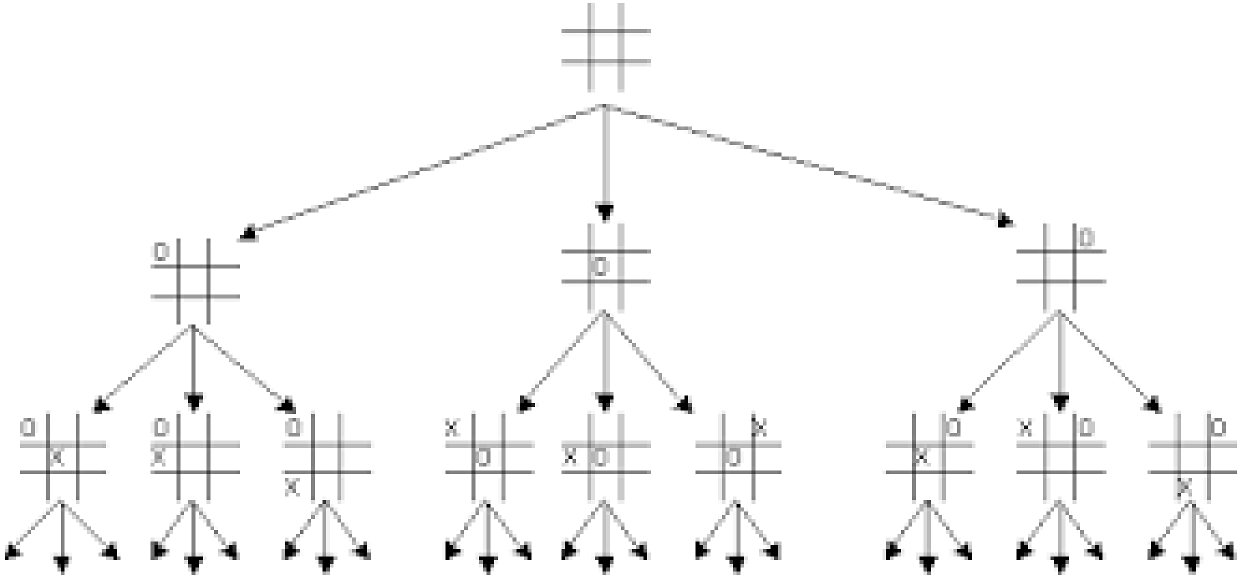
$\rightarrow\ 250^{150} \approx 5 \times 10^{359}$

$Chess:\ 35^{80} \approx 3 \times 10^{123}$

GAME STRATEGY

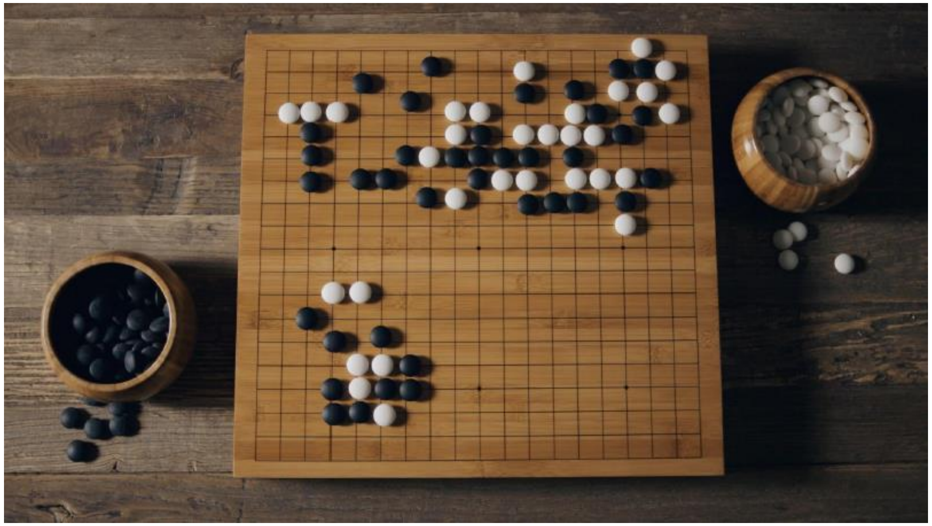
Game Strategy

- To win the game, we only need to build a game tree



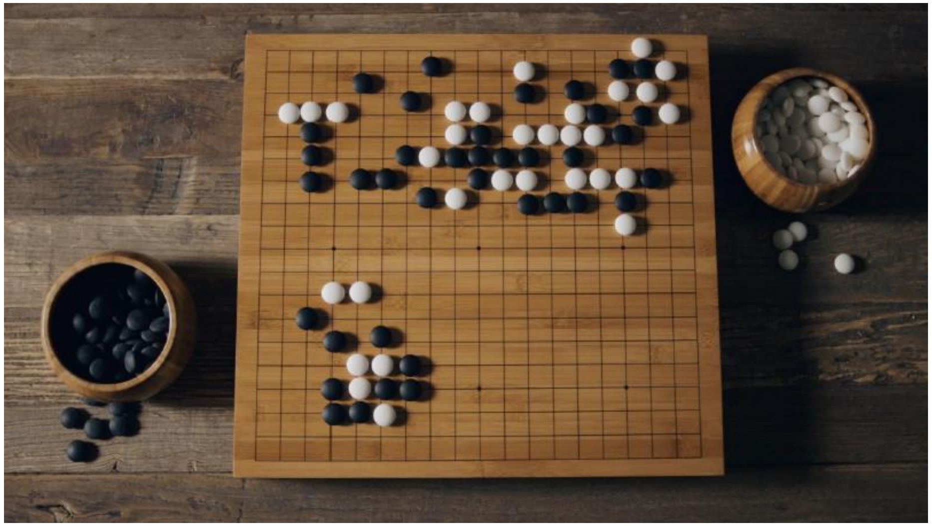
Game Strategy

- To win the game, we need to find $p^*(a|s)$
  - $p^*(a|s)$: Optimal action value function
  - Which action should I take?



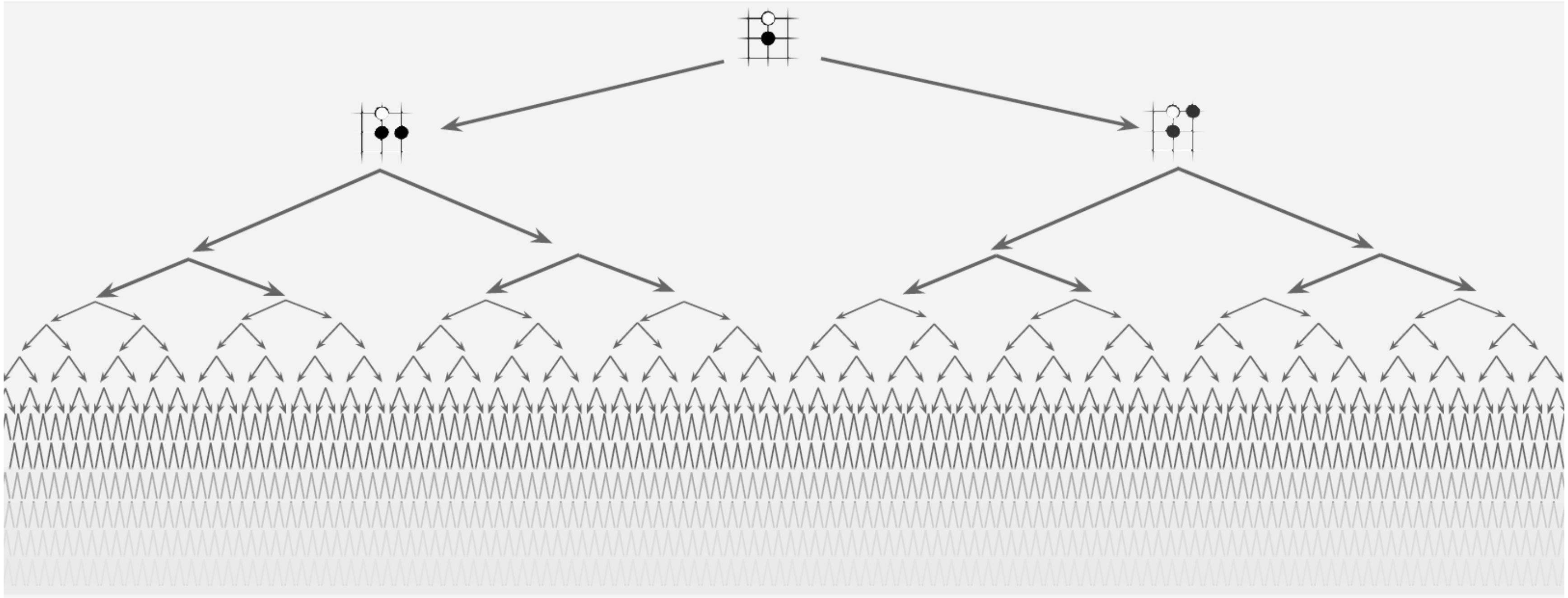
Game Strategy

- To win the game, we need to find $v^*(s)$
  - $v^*(s)$: Optimal Value Function

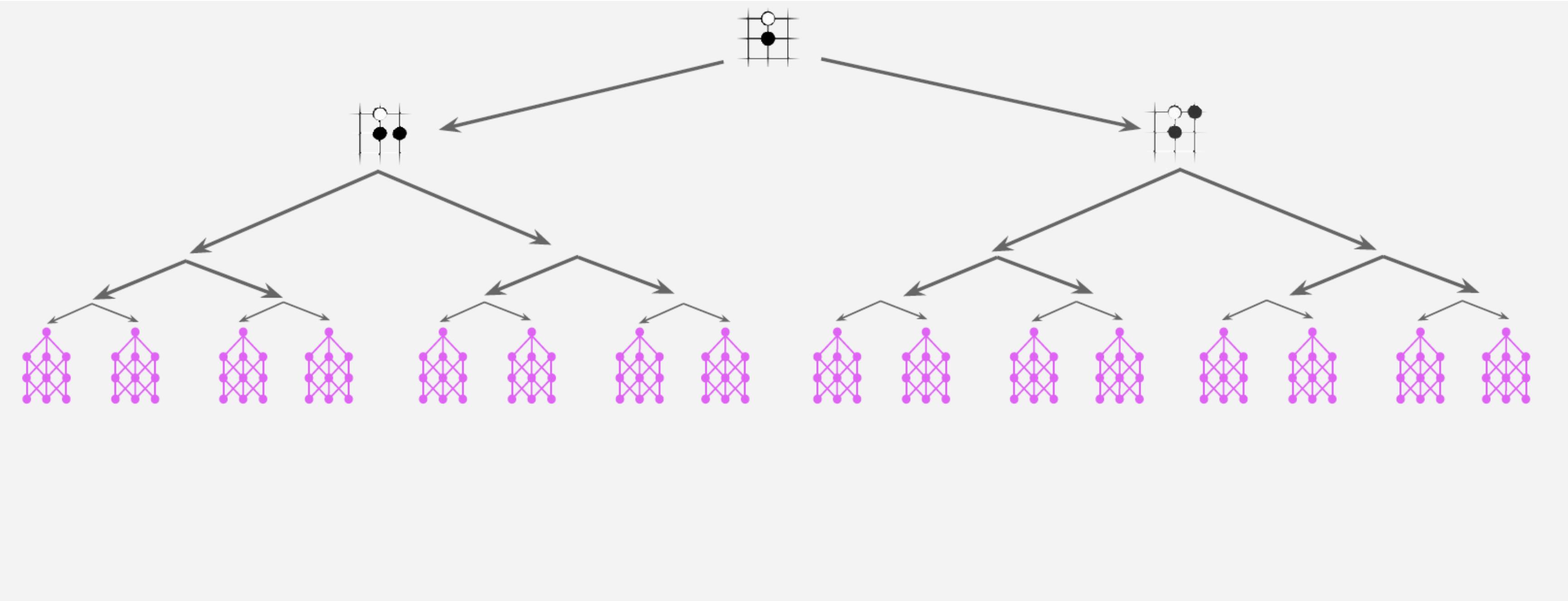


THREE COMPONENTS OF ALPHAGO

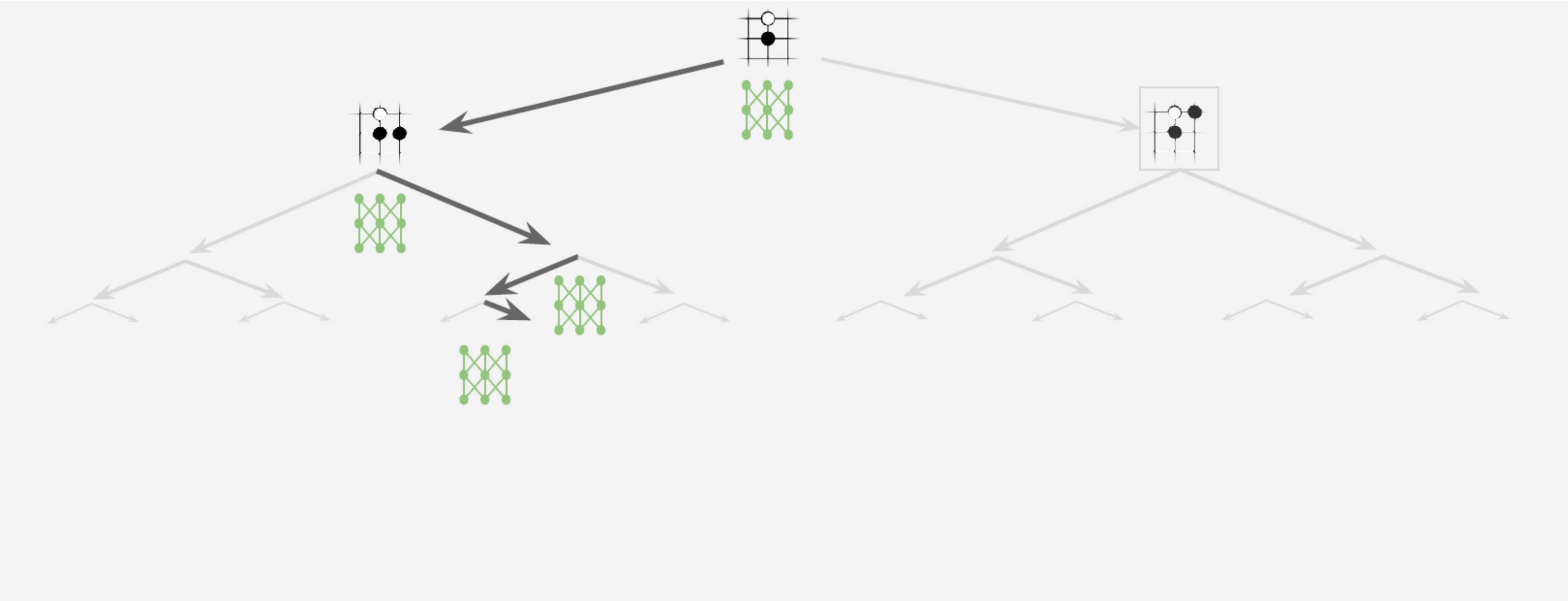
Monte Carlo Tree Search



Reducing depth search with value network



Reducing breadth search with policy network

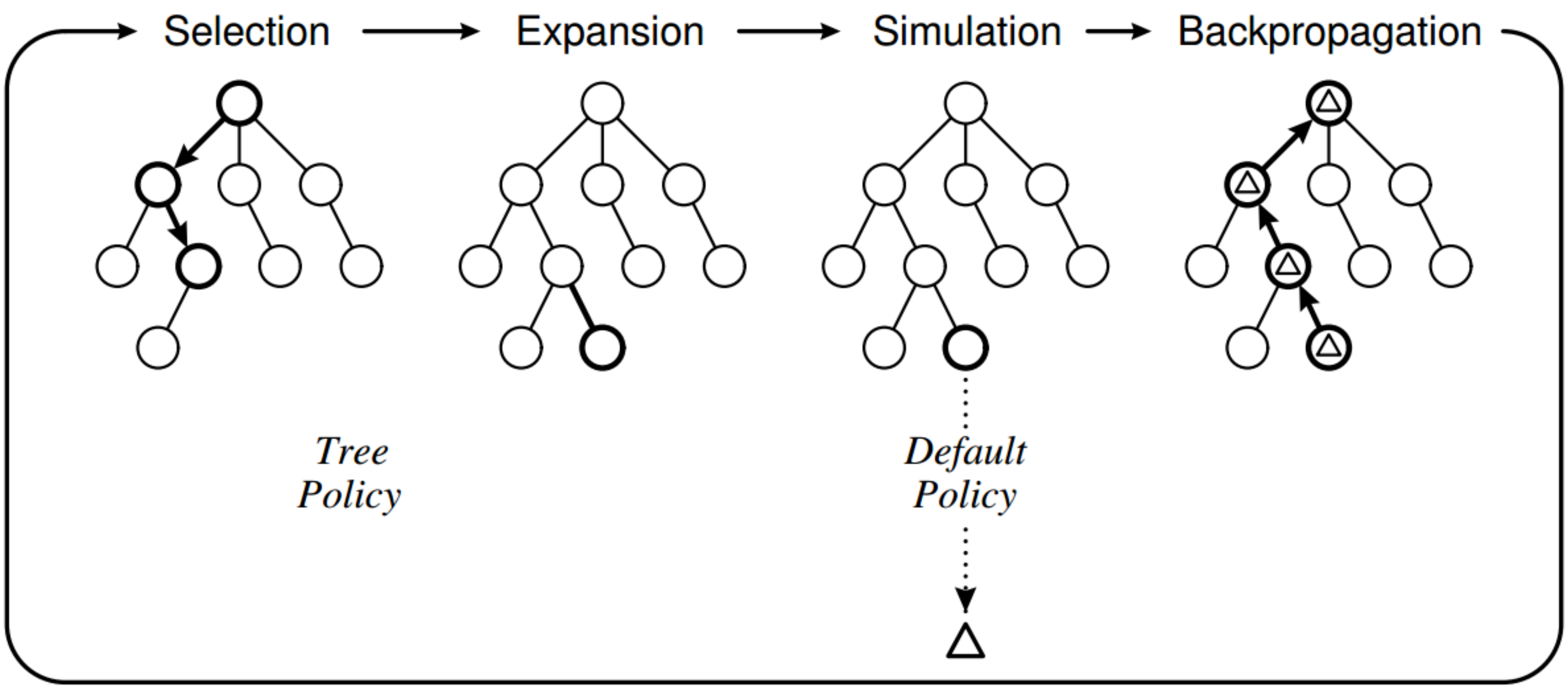


MONTE CARLO TREE SEARCH

Monte Carlo Tree Search

- a method for finding optimal decisions in a given domain by **taking random samples in the decision space** and **building a search tree** according to the results

One iteration of the general MCTS approach



General MCTS approach

- **Selection**: Starting at the root node, a child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached.
- **Expansion**: One (or more) child nodes are added to expand the tree, according to the available actions.
- **Simulation**: A simulation is run from the new node(s) according to the default policy to produce an outcome
- **Backpropagation**: The simulation result is “backed up” through the selected nodes to update their statistics.

Algorithm 1 General MCTS approach.

function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$) reward
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0)) action

General MCTS approach

- Playout, rollout, simulation
  - playing out the task to completion according to the default policy

- Four criteria for selecting the winning action
  - Max child: Select the root child with the highest reward.
  - Robust child: Select the most visited root child.
  - Max-Robust child: Select the root child with both the highest visit count and the highest reward. If none exist, then continue searching until an acceptable visit count is achieved
  - Secure child: Select the child which maximizes a lower confidence bound.

HOW TO DESIGN TREE POLICY? MULTI-ARMED BANDIT

Multi-armed bandit

- The K-armed bandit problem may be approached using a policy that determines which bandit to play, based on past rewards.

UCT (Upper Confidence Bounds for Trees) algorithm

Algorithm 2 The UCT algorithm.

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

function BESTCHILD(v, c)

$$\textbf{return} \quad \underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$$

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BACKUPNEGAMAX(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta$
    $\Delta \leftarrow -\Delta$
    $v \leftarrow$ parent of v

Exploration vs Exploitation

function BESTCHILD(v, c)

$$\textbf{return} \quad \underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$$

encourages the exploitation of higher-reward choices

encourages the exploration of less visited choices

ALPHAGO

3 key components in AlphaGo

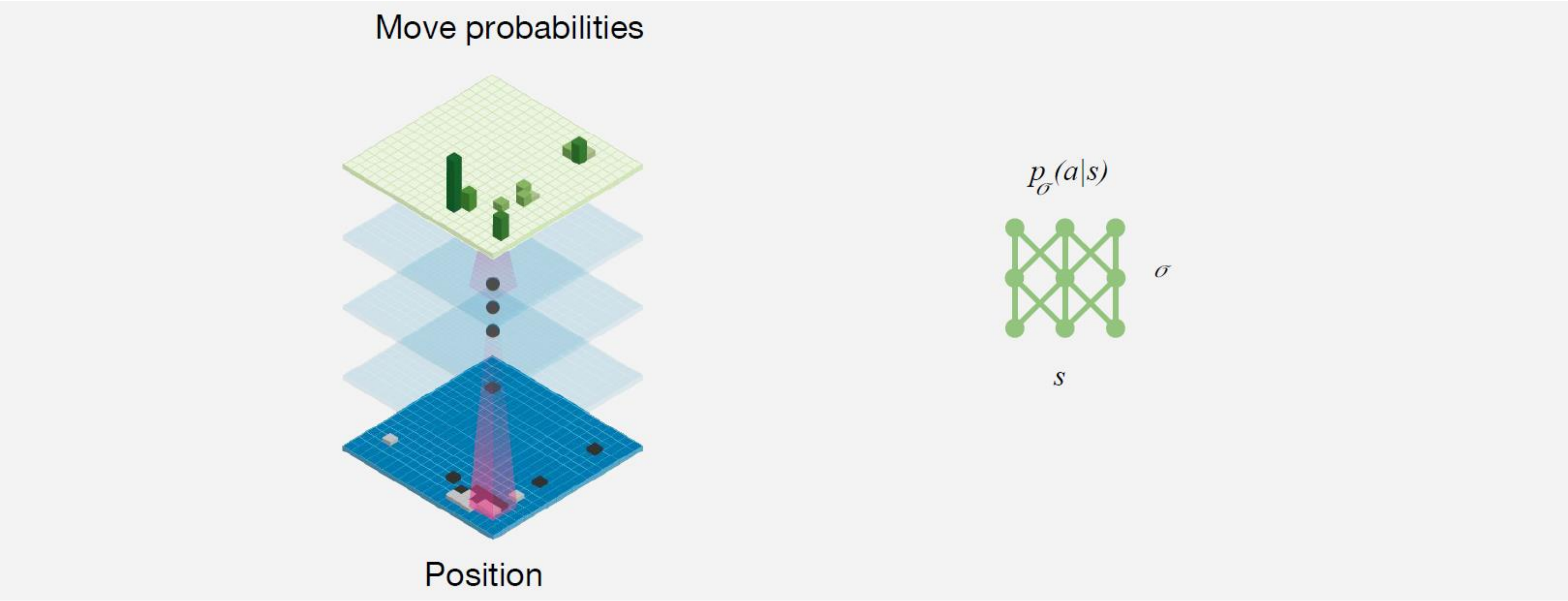
- MCTS
- Policy network
- Value network

POLICY NETWORK

Policy network

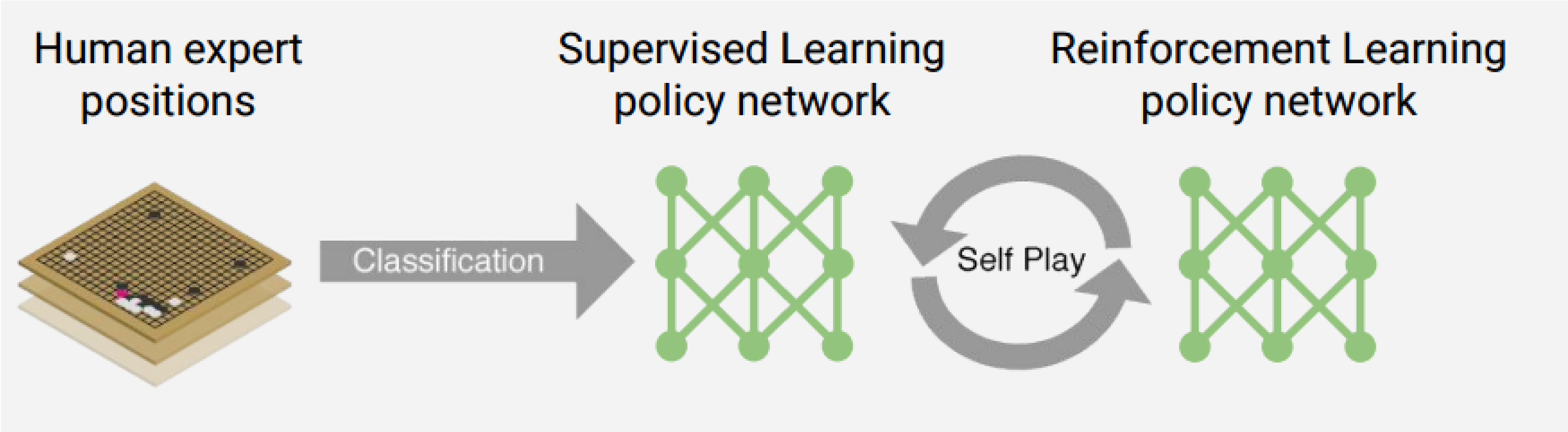
- To imitate expert moves
  - There are 19^2 possible actions (with different probabilities)

Policy network



3 Policy networks

- Supervised learning policy network
- Reinforcement learning policy network
- Roll-out policy network



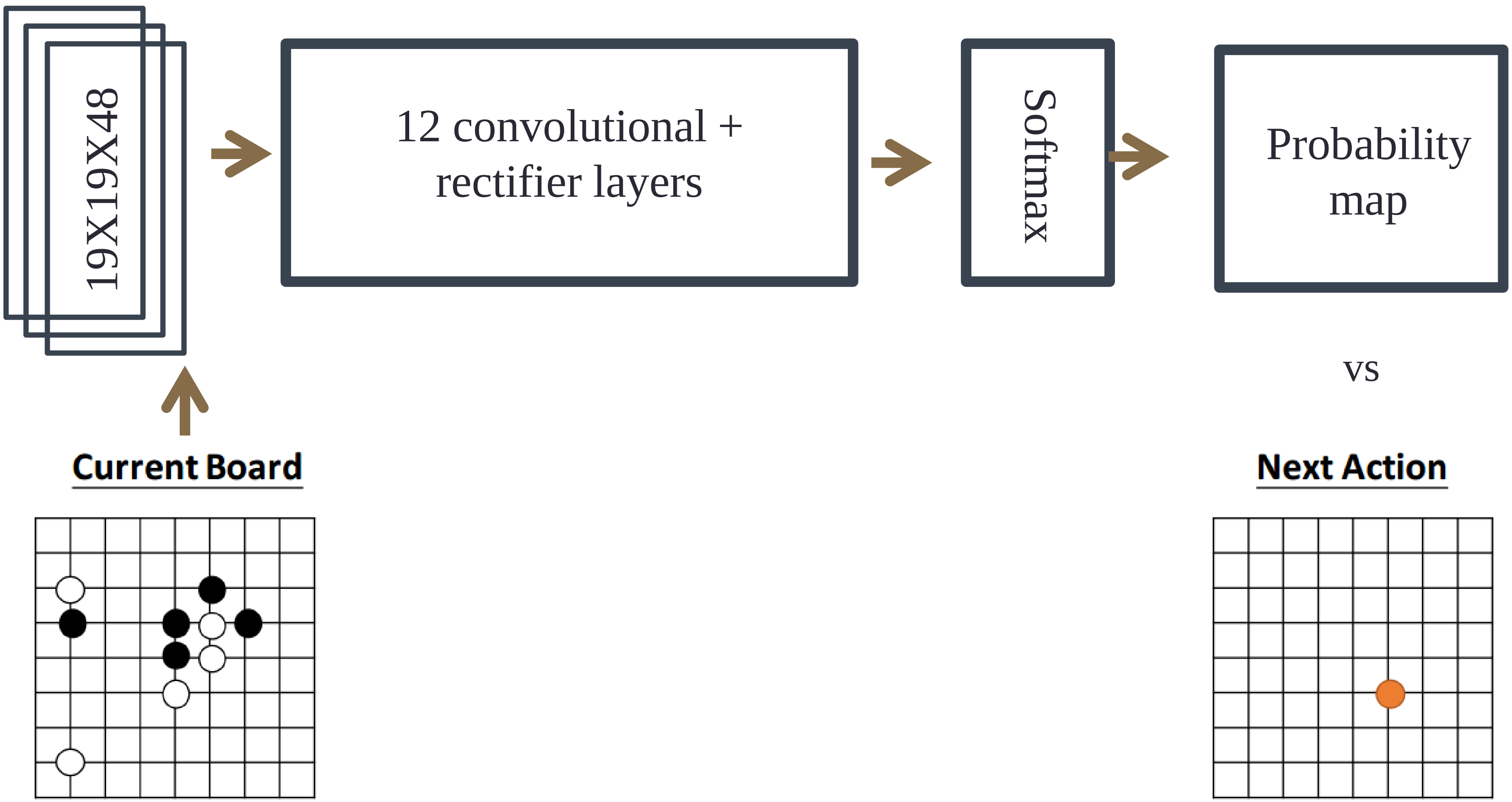
Supervised learning of policy networks

- Policy network: 12 layer convolutional neural network
- Training data: 30M positions from human expert games (KGS 5+ dan)
- Training algorithm: maximize likelihood by stochastic gradient descent

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

- Training time: 4 weeks on 50 GPUs using Google Cloud
- Results: 57% accuracy on held out test data (state-of-the art was 44%)

Supervised learning of policy networks



Training: $$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$$

Played by
Human Expert

Reinforcement learning of policy networks

- Policy network: 12 layer convolutional neural network
- Training data: games of self-play between policy network
- Training algorithm: maximize wins z by policy gradient reinforcement learning

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z$$

- Training time: 1 week on 50 GPUs using Google Cloud
- Results: 80% vs supervised learning. Raw network ~3 amateur dan.

Training the RL Policy Network P_ρ

- Refined version of SL policy (P_σ)
- Initialize weights to $\rho = \sigma$
- $\{\rho^- | \rho^-$ is an old version of $\rho\}$
- P_ρ vs $P_{\{\rho^-\}}$

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{T-1}} s_T \Longrightarrow z_t = r\left(s_T\right) \in \{\pm 1\}$$

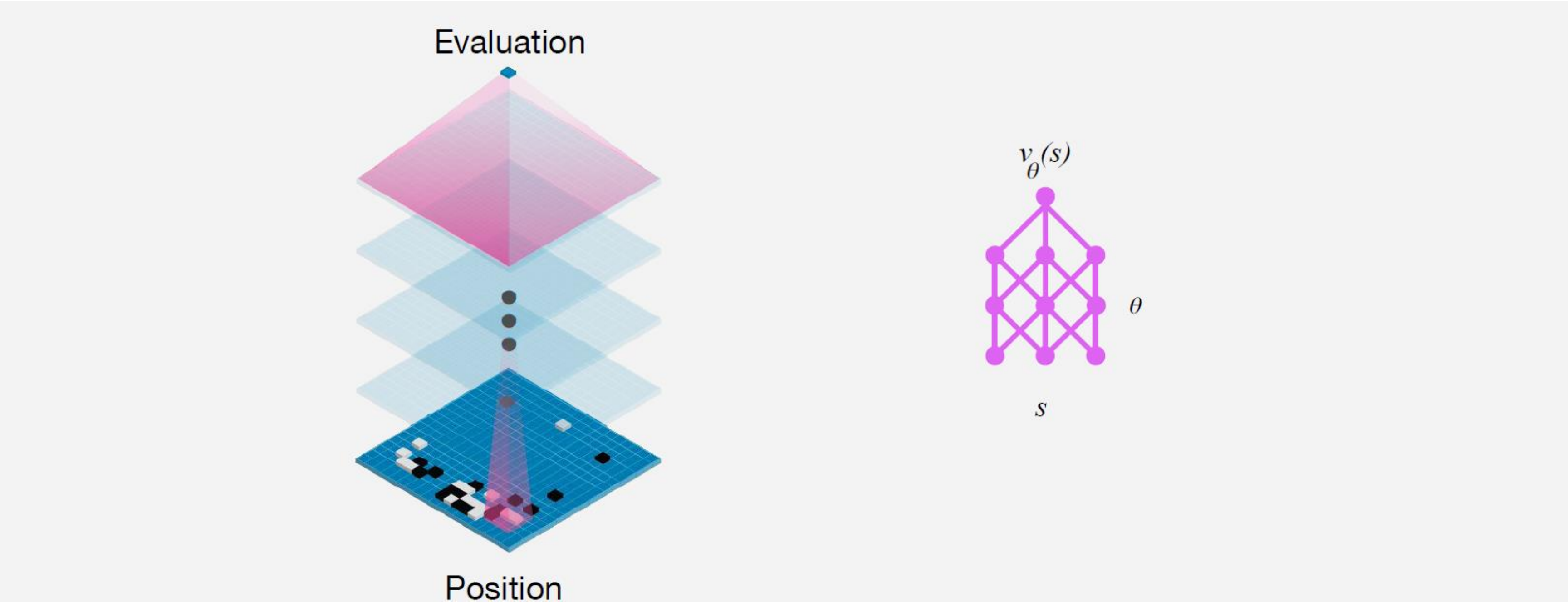
$$\Delta\rho \propto \sum_{t=1}^{T} \frac{\partial log P_\rho(a_t|s_t)}{\partial\rho} z_t$$

Roll-out policy network

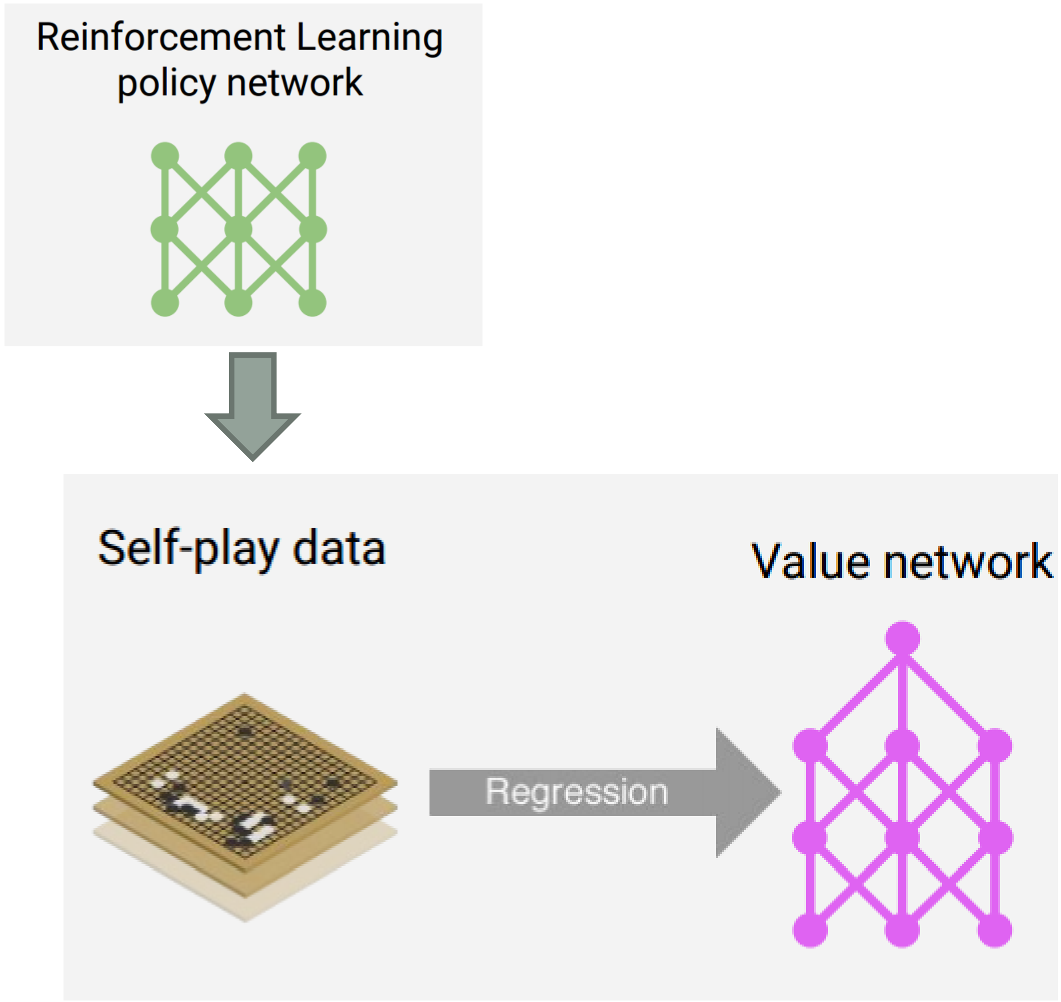
- Faster version of supervised learning policy network $p(a|s)$ with shall networks (3 ms→ 2us)

VALUE NETWORK

Value network



Value network



Reinforcement learning of value networks

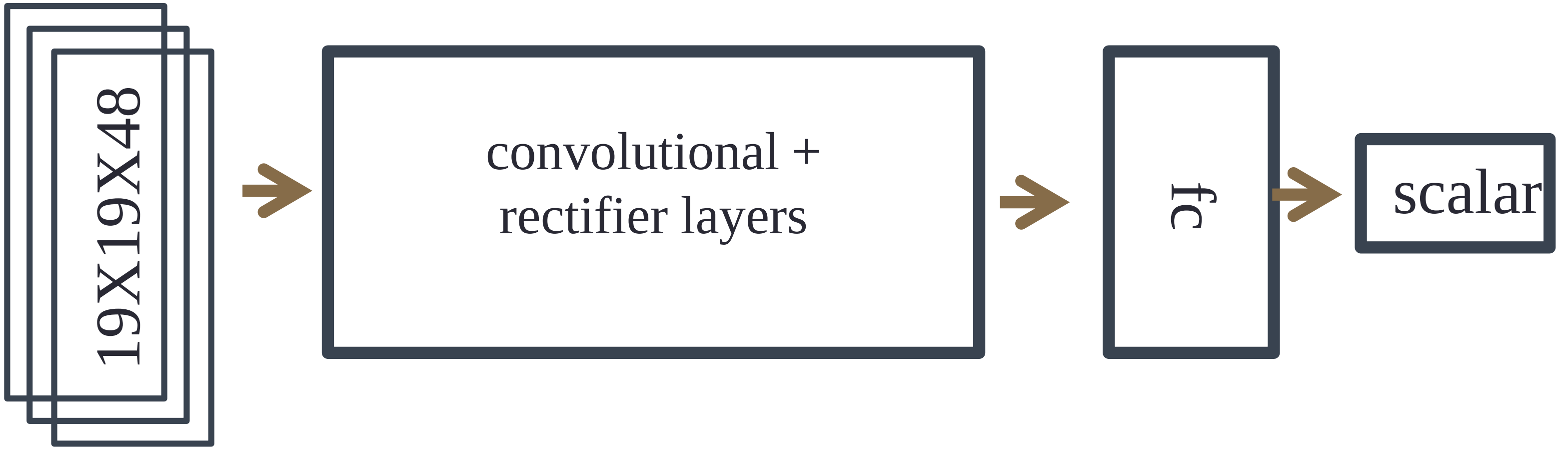
- Value network: 12 layer convolutional neural network
- Training data: 30 million games of self-play
- Training algorithm: minimize MSE by stochastic gradient descent

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

- Training time: 1 week on 50 GPUs using Google Cloud
- Results: First strong position evaluation function - previously thought impossible

Training the Value Network V_θ

- Position evaluation
- Approximating optimal value function
- Input: state, output: probability to win
- Goal: minimize MSE



TRAINING

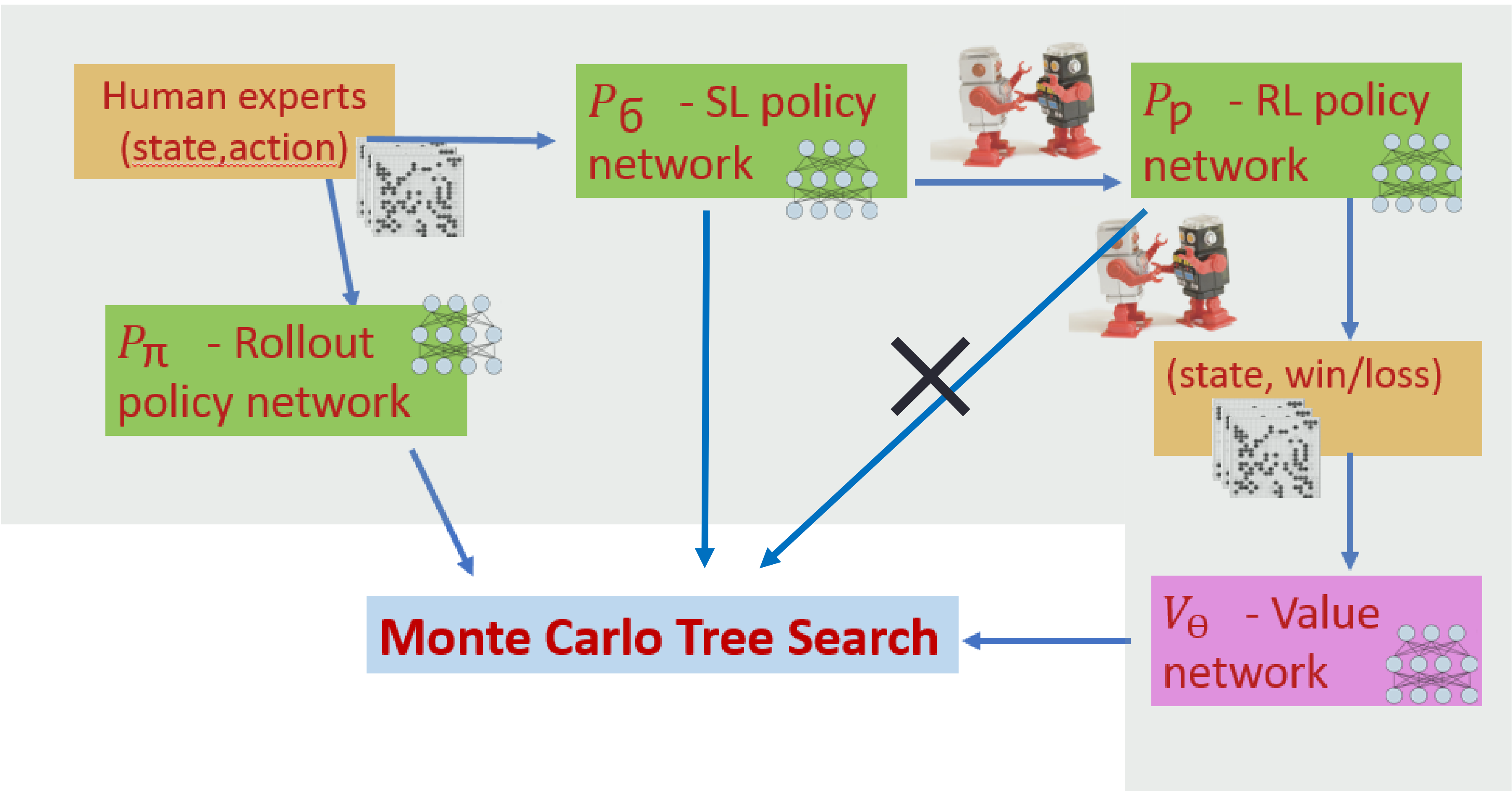
Input Features

Extended Data Table 2 | Input features for neural networks

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

Feature planes used by the policy network (all but last feature) and value network (all features).

Training the Deep Neural Networks



Summary: Training the Deep Neural Networks

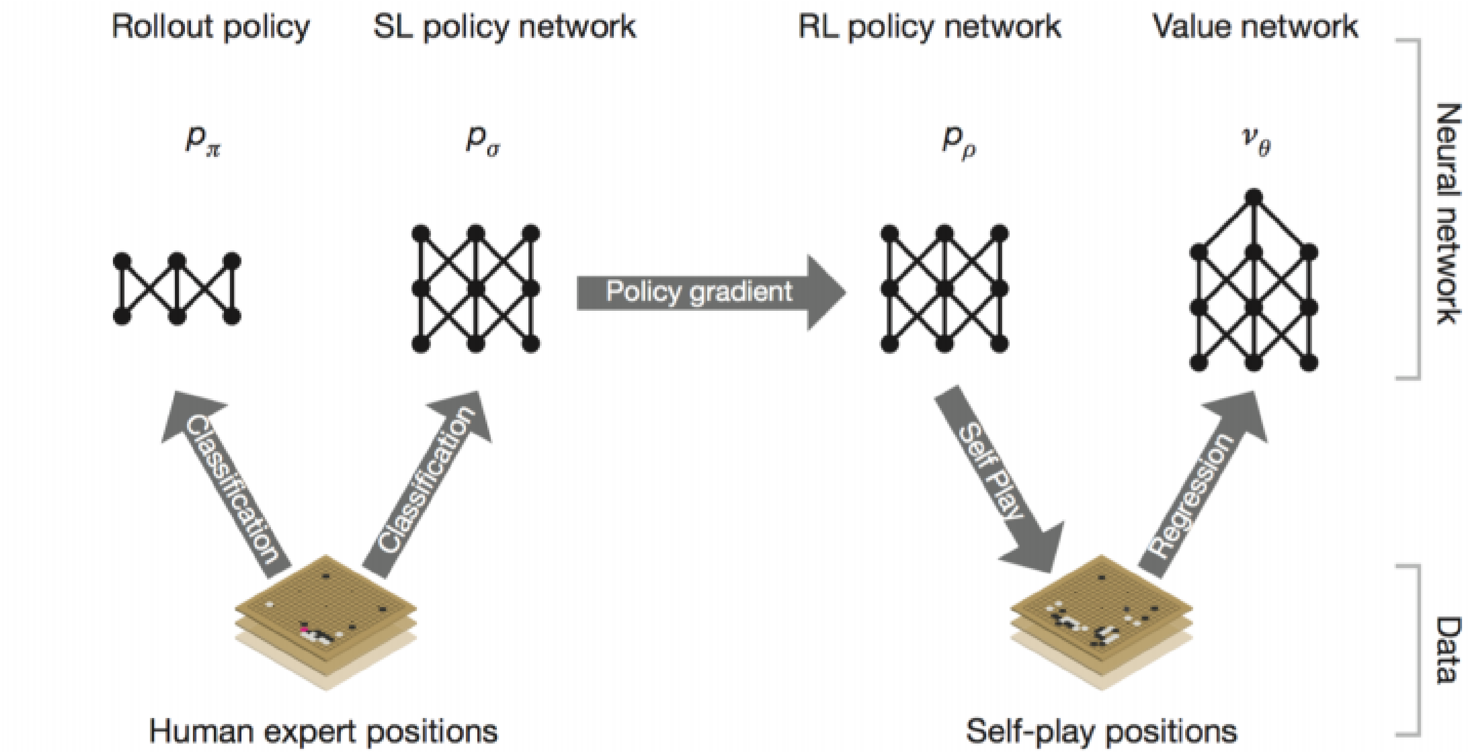


Figure: Neural Network Training Pipeline and Architecture

MCTS

Monte Carlo Tree Search

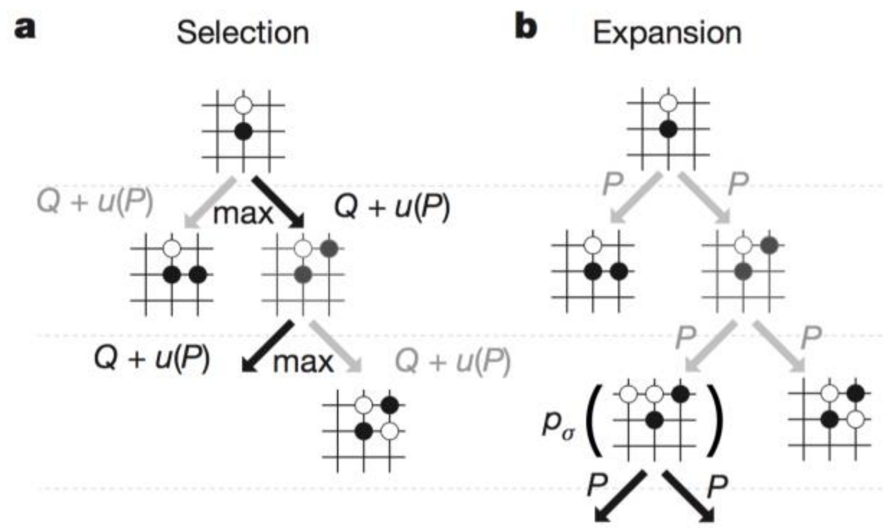


Figure: Monte Carlo Tree Search in AlphaGo

Edge storing statistics

- $\{P(s,a), N_v(s,a), N_r(s,a), W_v(s,a), W_r(s,a), Q(s,a)\}$
  - $P(s,a)$: prior probability
  - $N_v(s,a)$: # of leaf evaluation
  - $W_v(s,a)$: Monte Carlo estimated action value accumulated over $N_v(s,a)$
  - $N_r(s,a)$: # of roll-out evaluation
  - $W_r(s,a)$: Monte Carlo estimated action value accumulated over $N_r(s,a)$

Monte Carlo Tree Search: selection

- Each edge (s,a) stores:
  - $Q(s,a)$ - action value (average value of sub tree)
  - $N(s,a)$ – visit count
  - $P(s,a)$ – prior probability

$$p(s,a) = p_\sigma(a|s)$$

$$a_t = \underset{a}{argmax}\left(Q(s_t,a) + u(s_t,a)\right)$$

$$u(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N_r(s,b)}}{1 + N_r(s,a)}$$

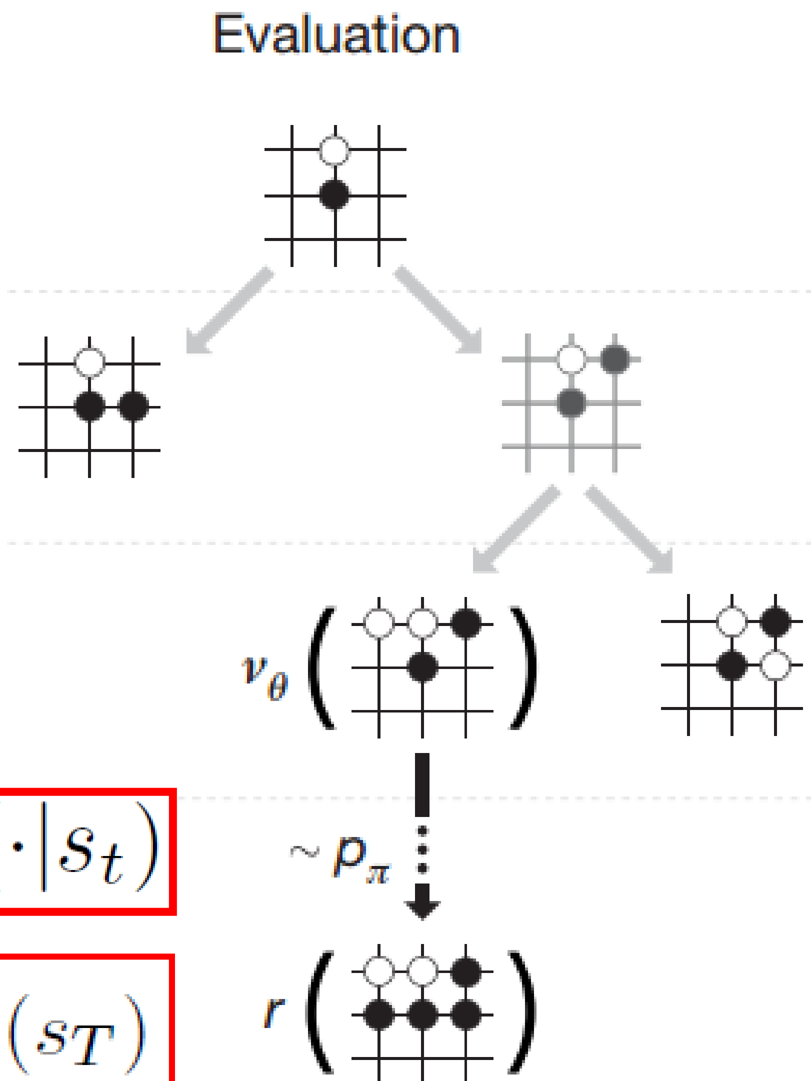
Monte Carlo Tree Search: evaluation

- Leaf evaluation:
  - Value network
  - Random rollout

$$V(s_L) = (1-\lambda)V_\theta(s_L) + \lambda z_L$$

$$a_t \sim p_\pi(\cdot|s_t)$$

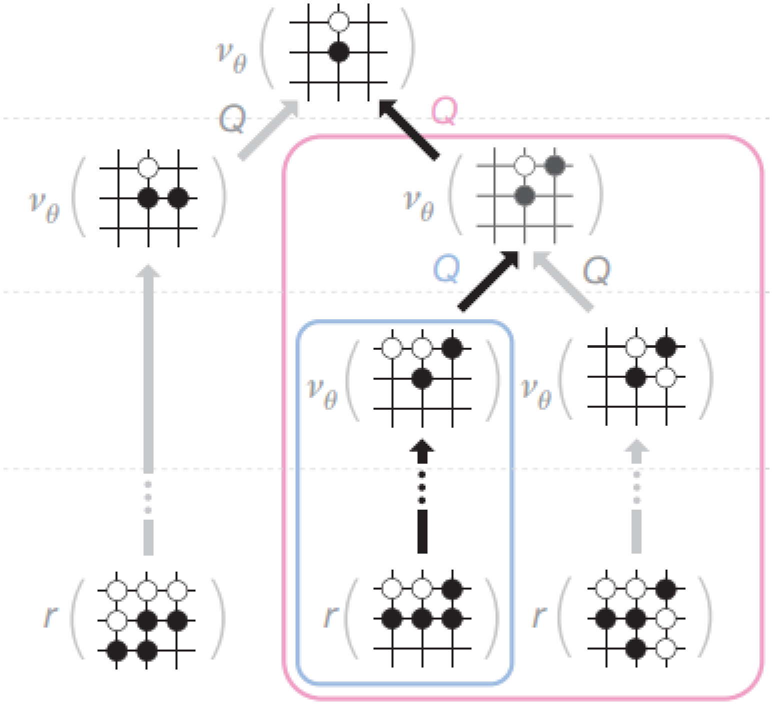
$$z_T = \pm r(s_T)$$



Monte Carlo Tree Search: backup

$$Q(s,a) = (1-\lambda)\frac{W_v(s,a)}{N_v(s,a)} + \lambda\frac{W_r(s,a)}{N_r(s,a)}$$

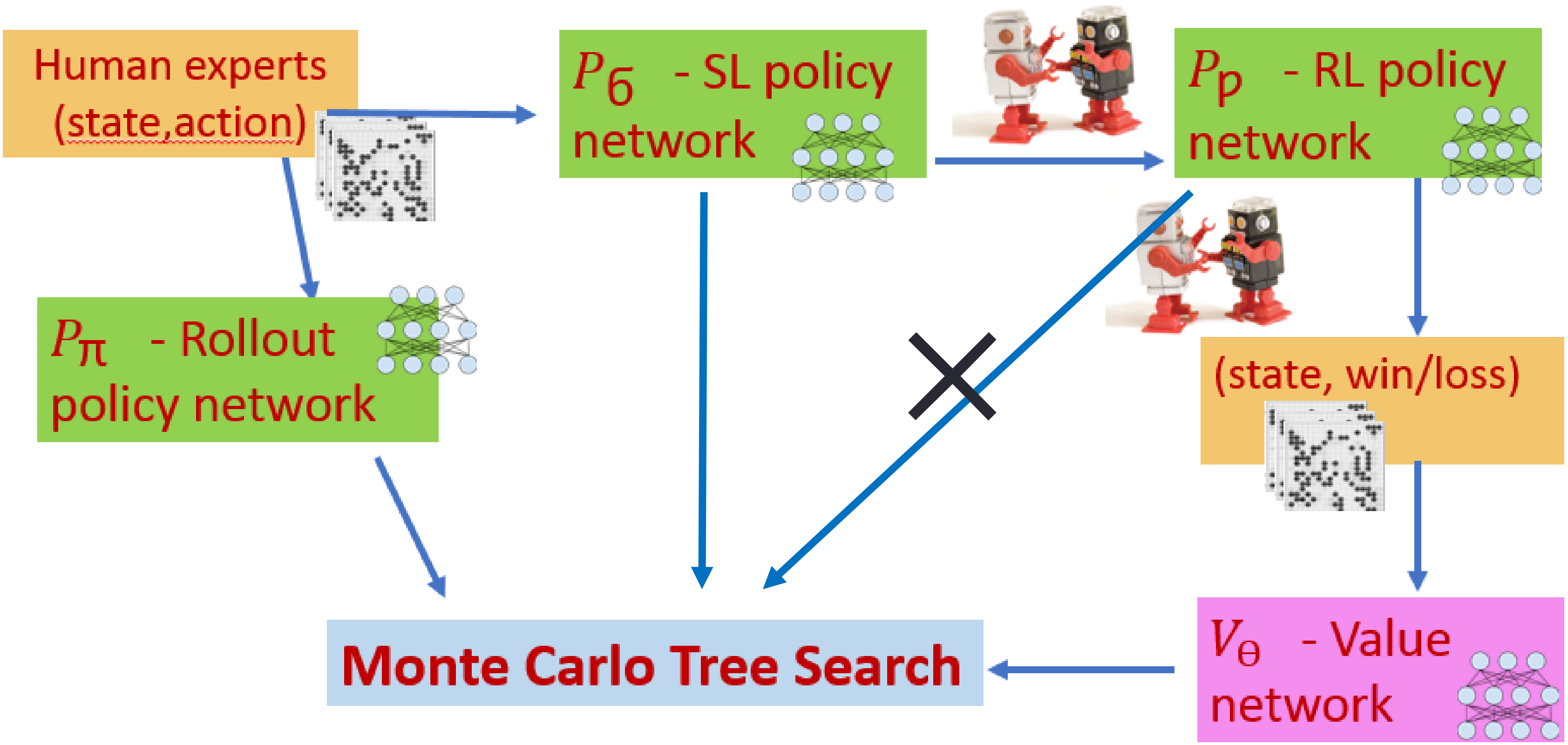
Value network Roll-out



How to choose the next move?

- Maximum visit count
    - Less sensitive to outliers than maximum action value

Training the Deep Neural Networks



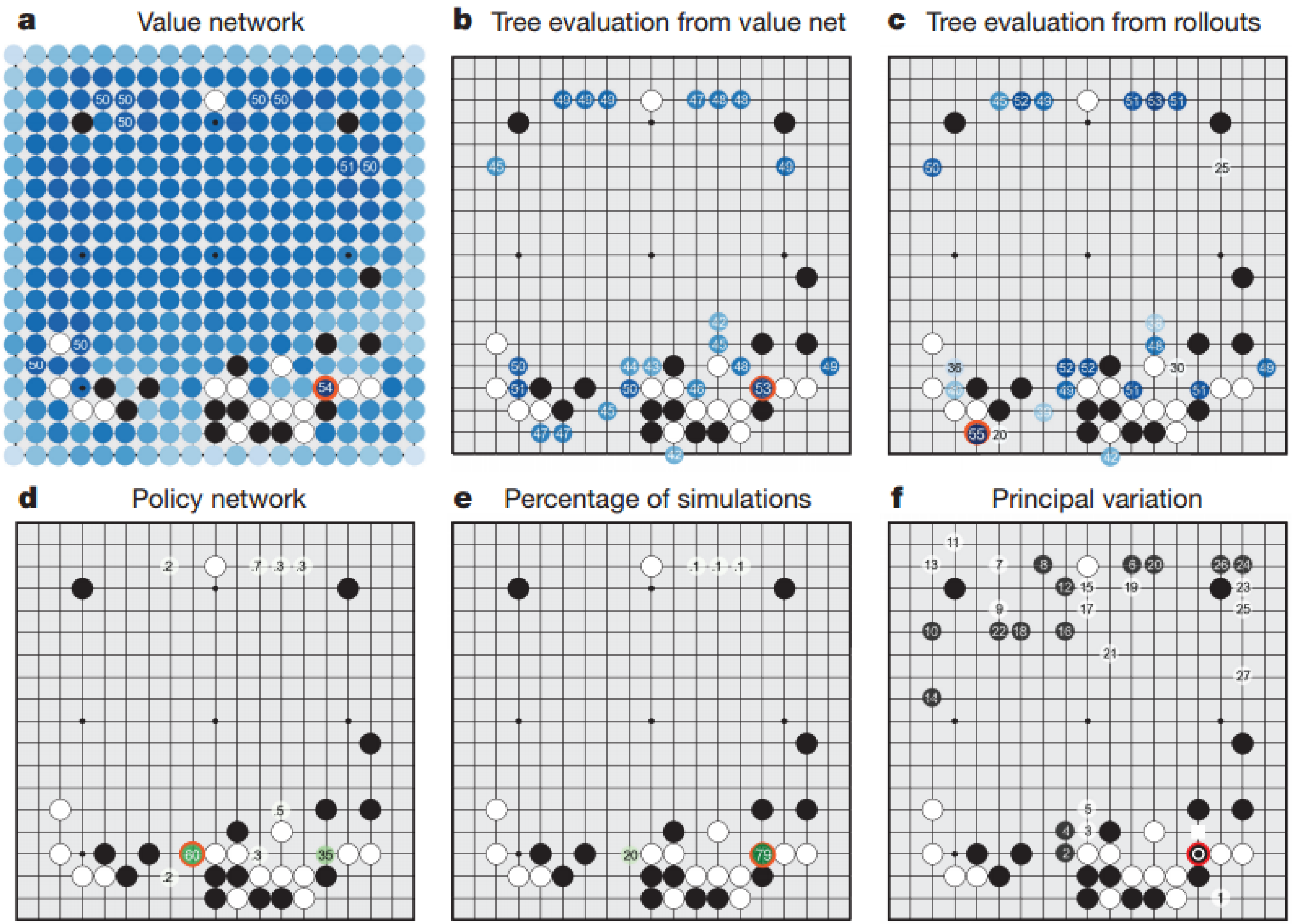


Figure 5 | How AlphaGo (black, to play) selected its move in an informal game against Fan Hui. For each of the following statistics, the location of the maximum value is indicated by an orange circle. **a**, Evaluation of all successors s' of the root position s, using the value network $v_\theta(s')$; estimated winning percentages are shown for the top evaluations. **b**, Action values $Q(s, a)$ for each edge (s, a) in the tree from root position s; averaged over value network evaluations only ($\lambda = 0$). **c**, Action values $Q(s, a)$, averaged over rollout evaluations only ($\lambda = 1$). **d**, Move probabilities directly from the SL policy network, $p_\sigma(a|s)$; reported as a percentage (if above 0.1%). **e**, Percentage frequency with which actions were selected from the root during simulations. **f**, The principal variation (path with maximum visit count) from AlphaGo's search tree. The moves are presented in a numbered sequence. AlphaGo selected the move indicated by the red circle; Fan Hui responded with the move indicated by the white square; in his post-game commentary he preferred the move (labelled 1) predicted by AlphaGo.


KBA
Google DeepMind
Challenge Match
8 - 15 March 2016
AlphaGo
Google DeepMind
Challenge Match


phaGo
KBA
Google DeepMind
Challenge Match
8 - 15 March 2016

AlphaGo VS Experts

4:1