Deep Reinforcement Learning

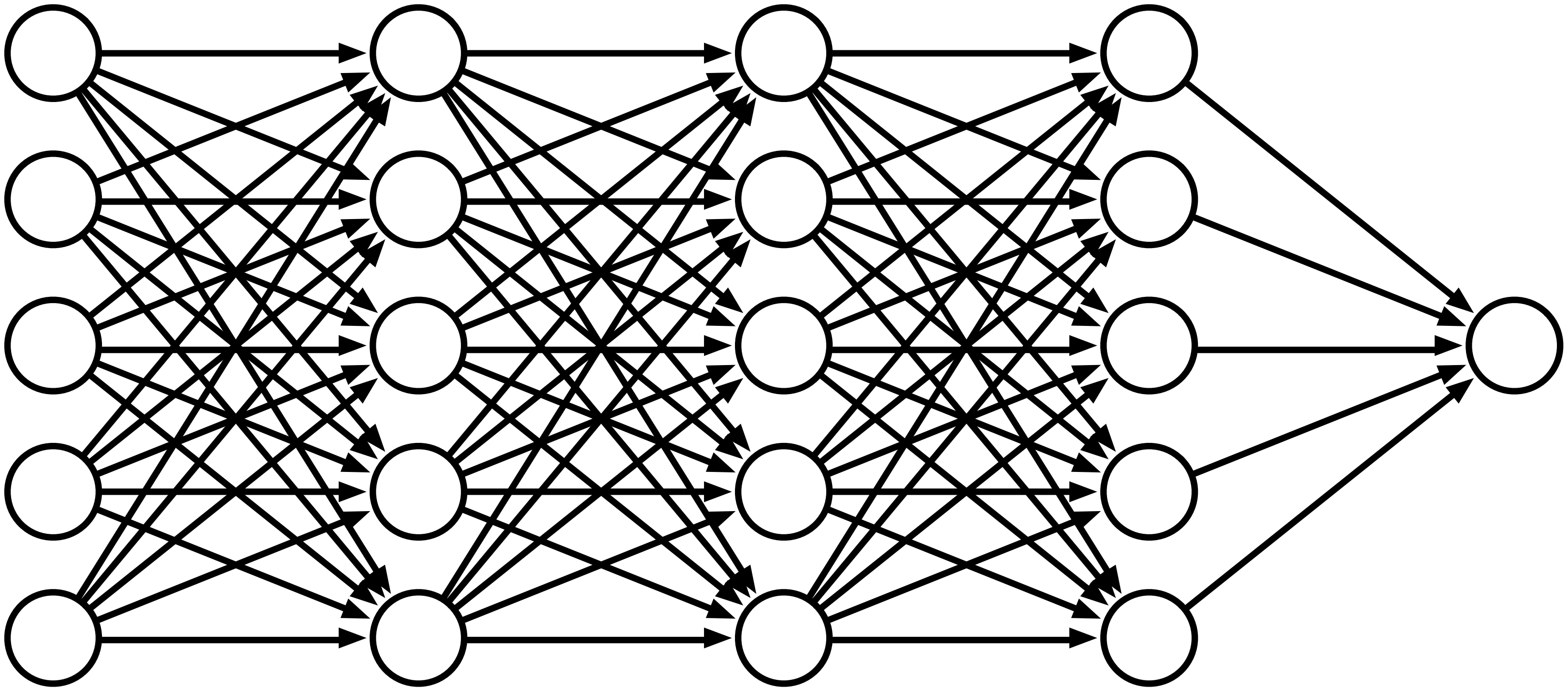
Philipp Koehn

15 April 2025

Reinforcement Learning

- Sequence of actions
  - moves in chess
  - driving controls in car
- Uncertainty
  - moves by component
  - random outcomes (e.g., dice rolls, impact of decisions)
- Reward delayed
  - chess: win/loss at end of game
  - Pacman: points scored throughout game
- Challenge: find optimal policy for actions

Deep Learning



- Mapping input to output through multiple layers
- Weight matrices and activation functions

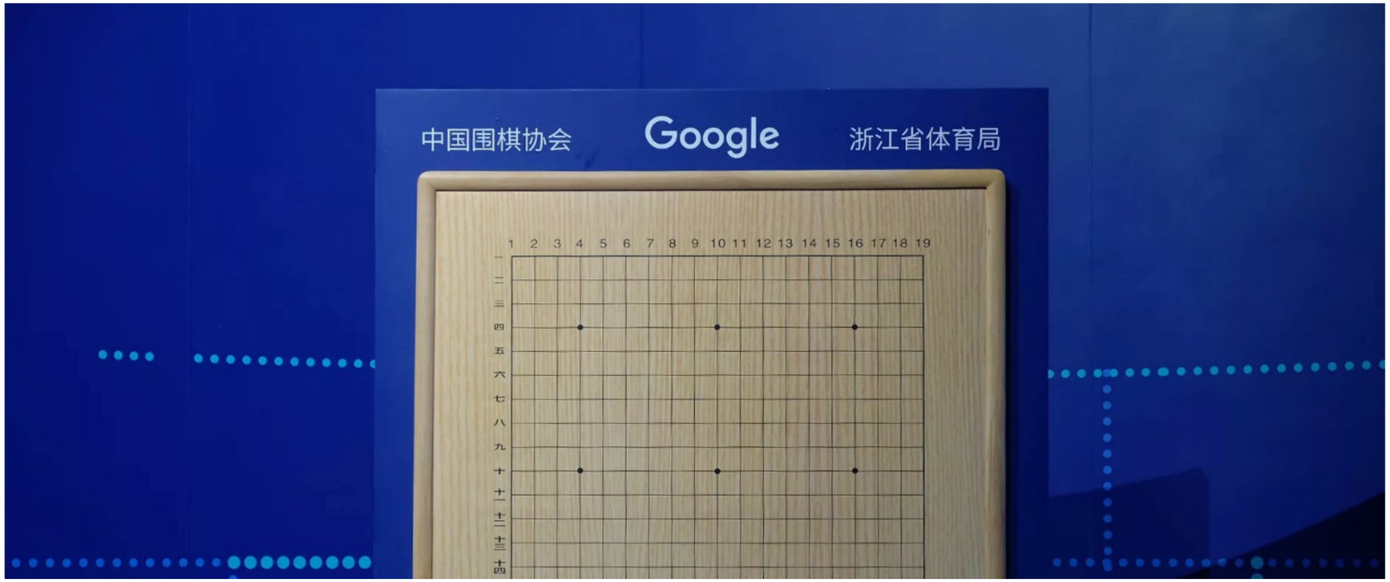
AlphaGo

GOOGLE \ TECH \ ARTIFICIAL INTELLIGENCE

AlphaGo retires from competitive Go after defeating world number one 3-0

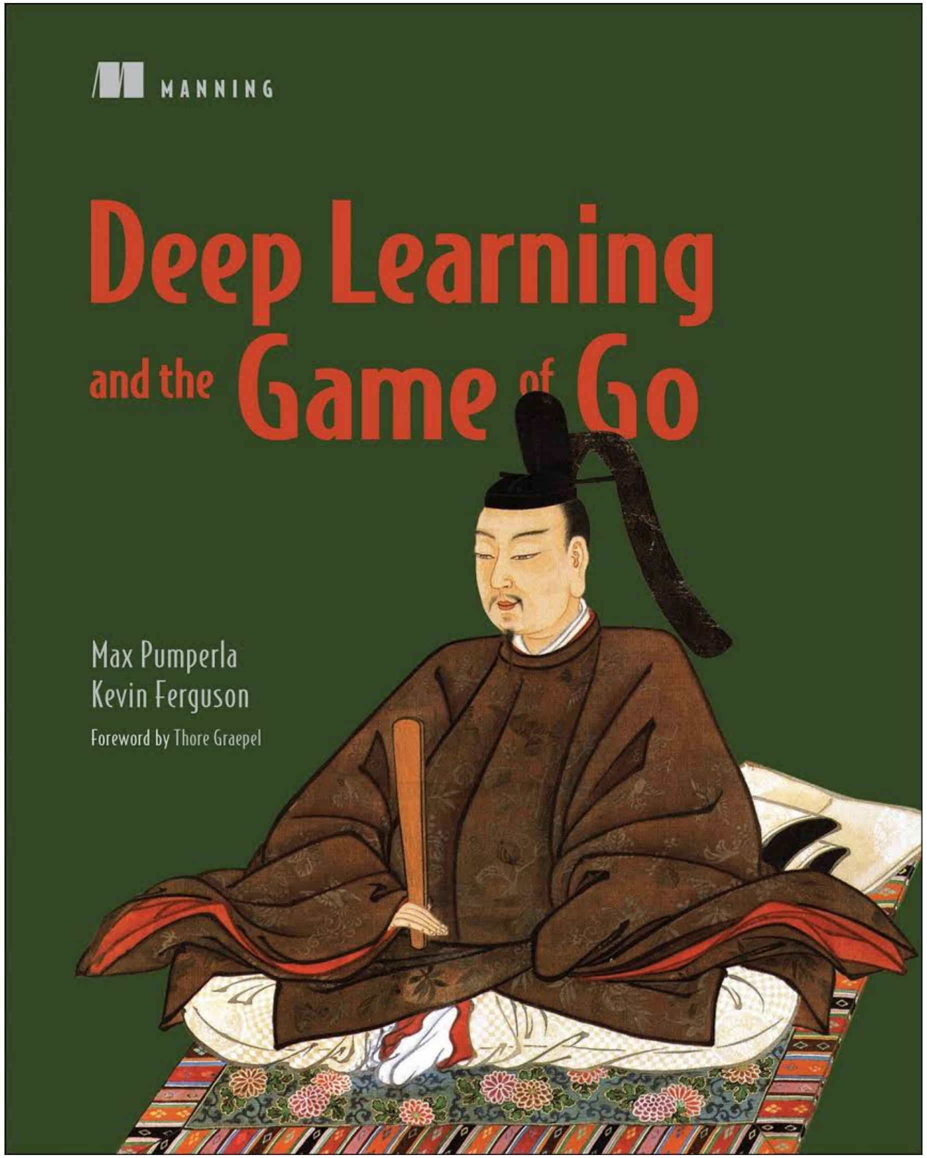
By Sam Byford | @345triangle | May 27, 2017, 5:17am EDT

SHARE



Book

- Lecture based on the book
  Deep Learning and the Game of Go
  by Pumperla and Ferguson, 2019
- Hands-on introduction to game playing and neural networks
- Lots of Python code

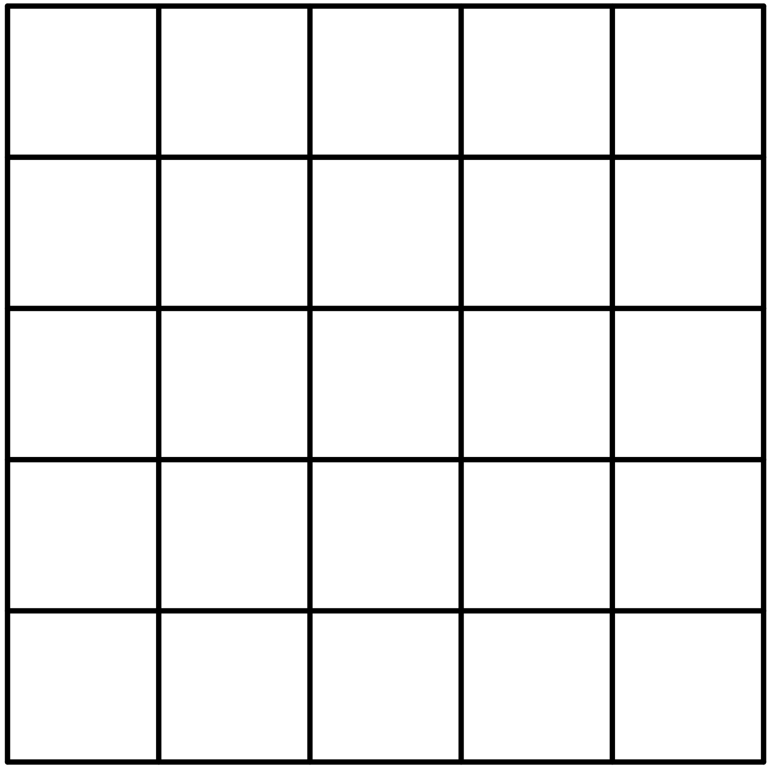


go

Go

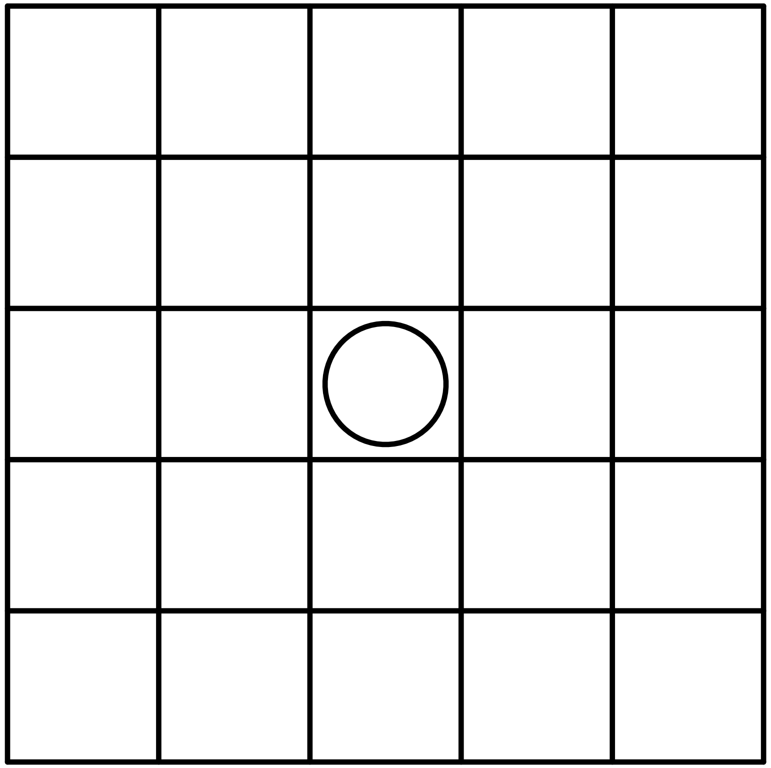
- Board game with white and black stones
- Stones may be placed anywhere
- If opponents stones are surrounded, you can capture them
- Ultimately: you need to claim territory
- Player with most territory and captured stones wins

Go Board



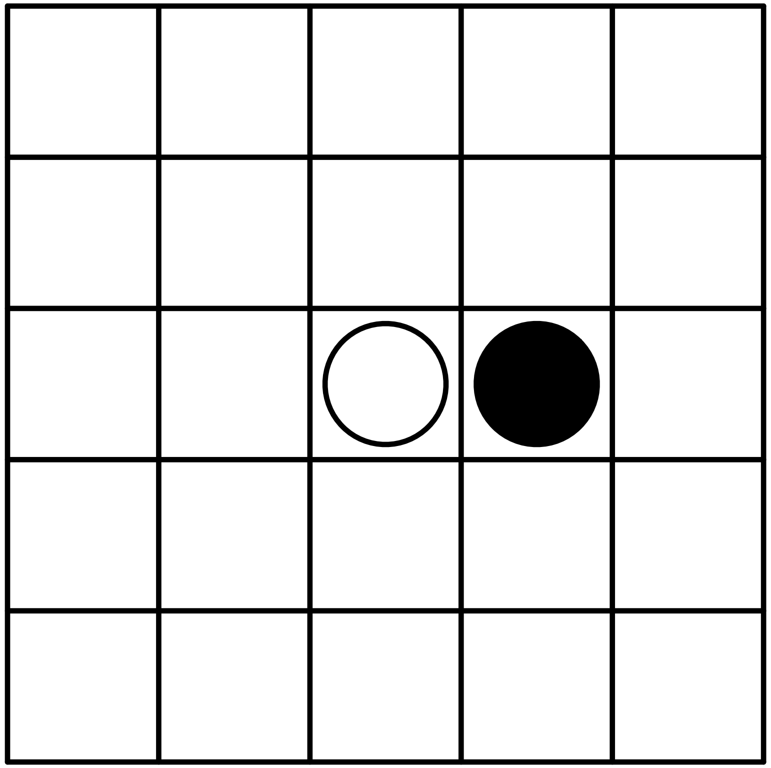
- Starting board, standard board is 19x19, but can also play with 9x9 or 13x13

Move 1



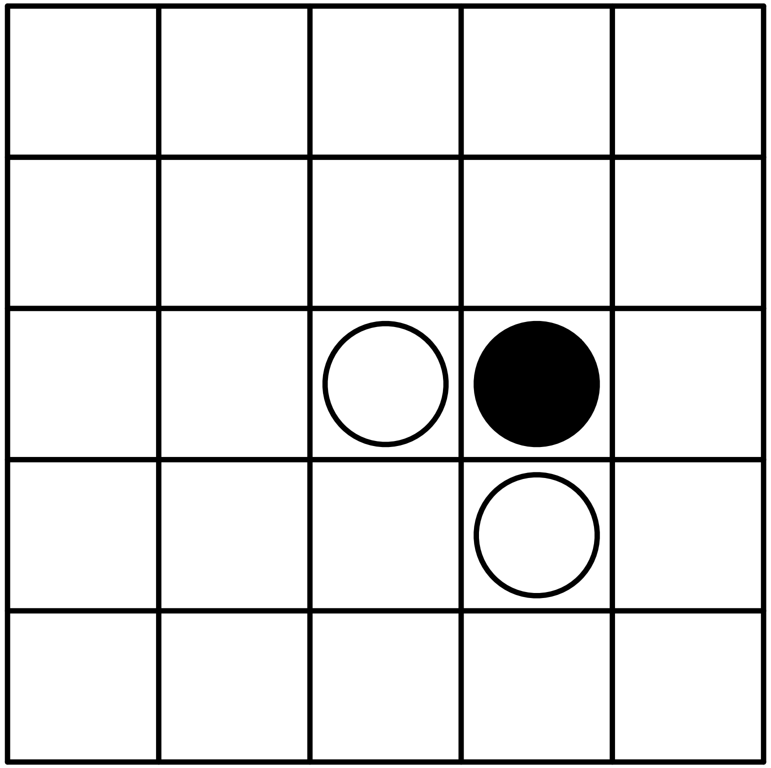
- First move: white

Move 2



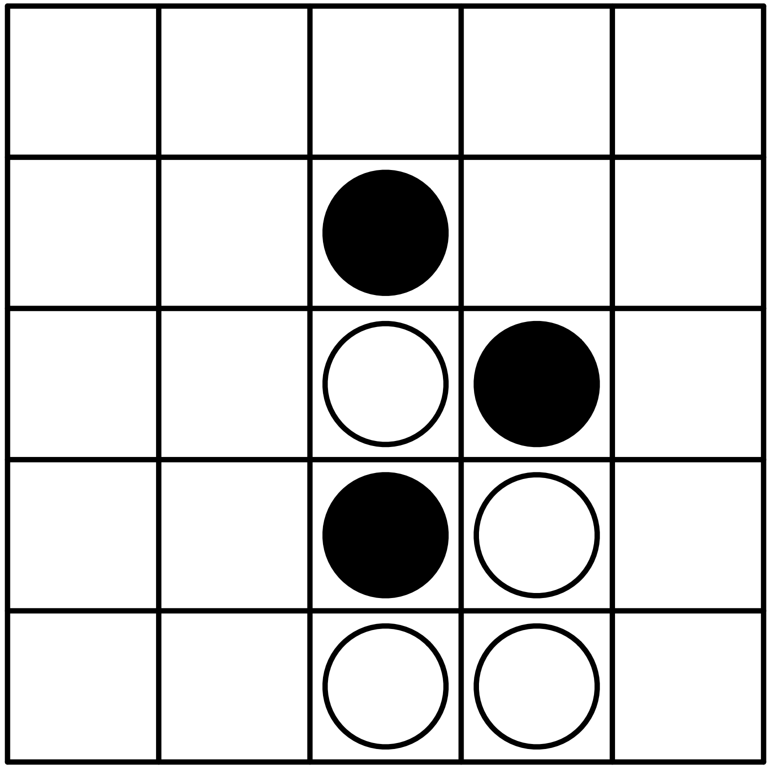
- Second move: black

Move 3



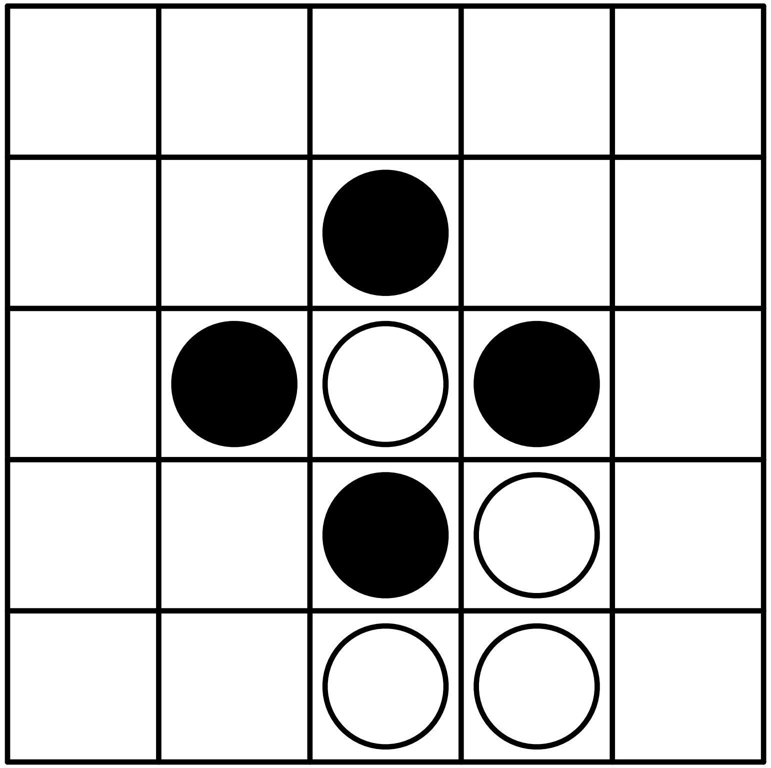
- Third move: white

Move 7



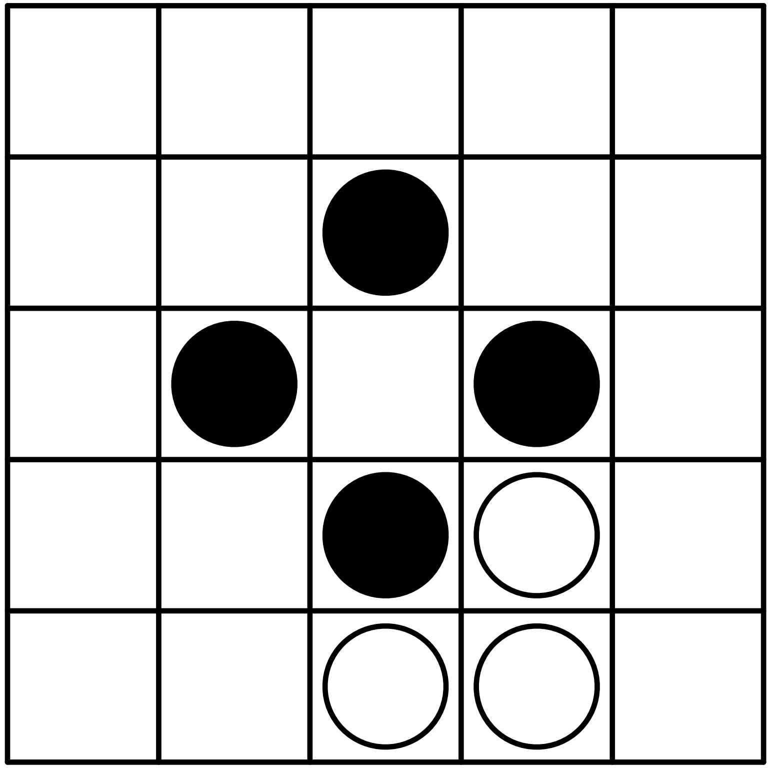
- Situation after 7 moves, black's turn

Move 8



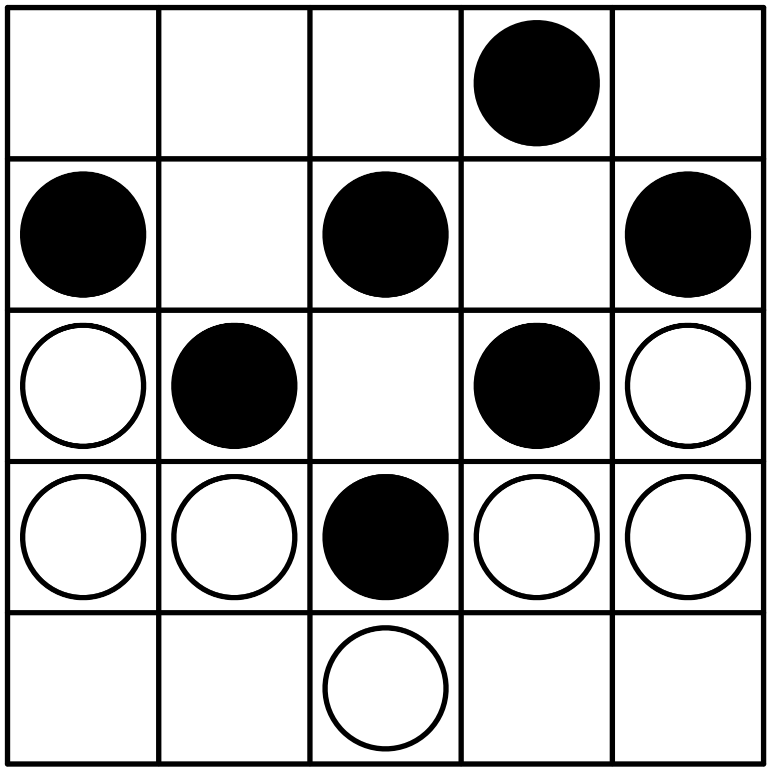
- Move by black: surrounded white stone in the middle

Capture



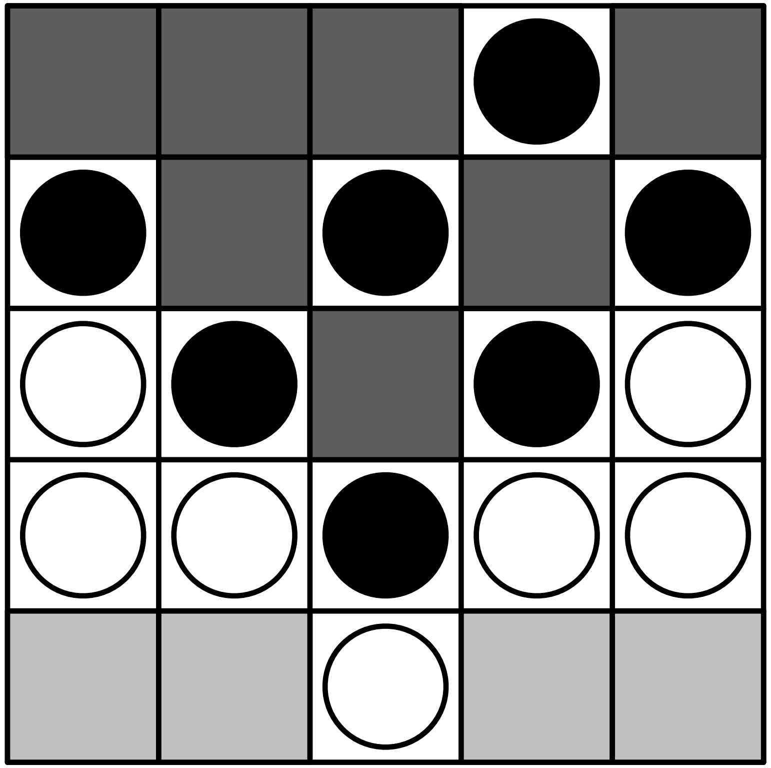
- White stone in middle is captured

Final State



- Any further moves will not change outcome

Final State with Territory Marked



- Total score: number of squares in territory + number of captured stones

Why is Go Hard for Computers?

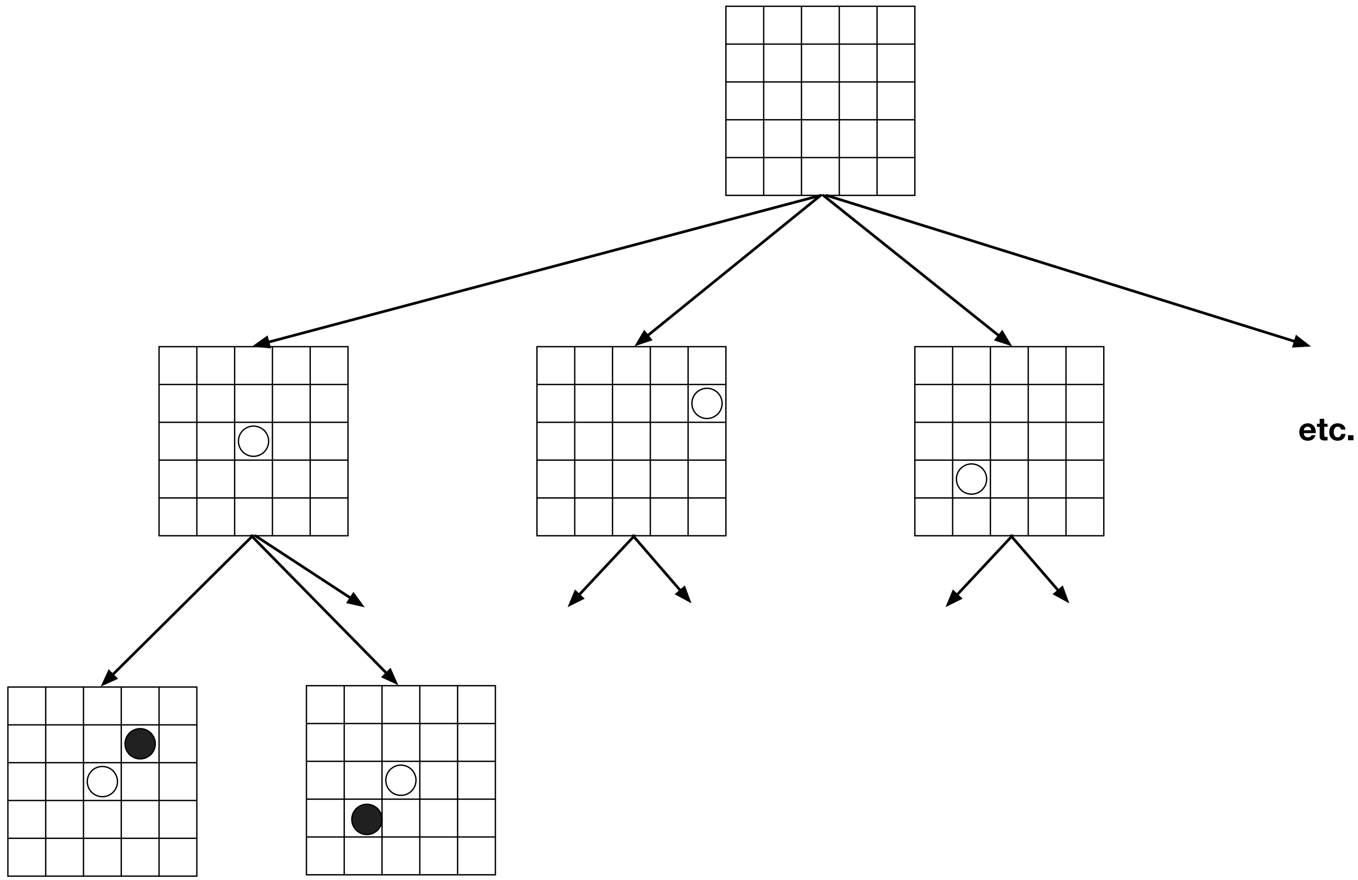
- Many moves possible
  - 19x19 board
  - 361 moves initially
  - games may last 300 moves

⇒ Huge branching factor in search space

- Hard to evaluate board positions
  - control of board most important
  - number of captured stones less relevant

game playing

Game Tree



- Recall: game tree to consider all possible moves

Alpha-Beta Search

- Explore game tree depth-first
- Exploration stops at win or loss
- Backtrack to other paths, note best/worst outcome
- Ignore paths with worse outcomes

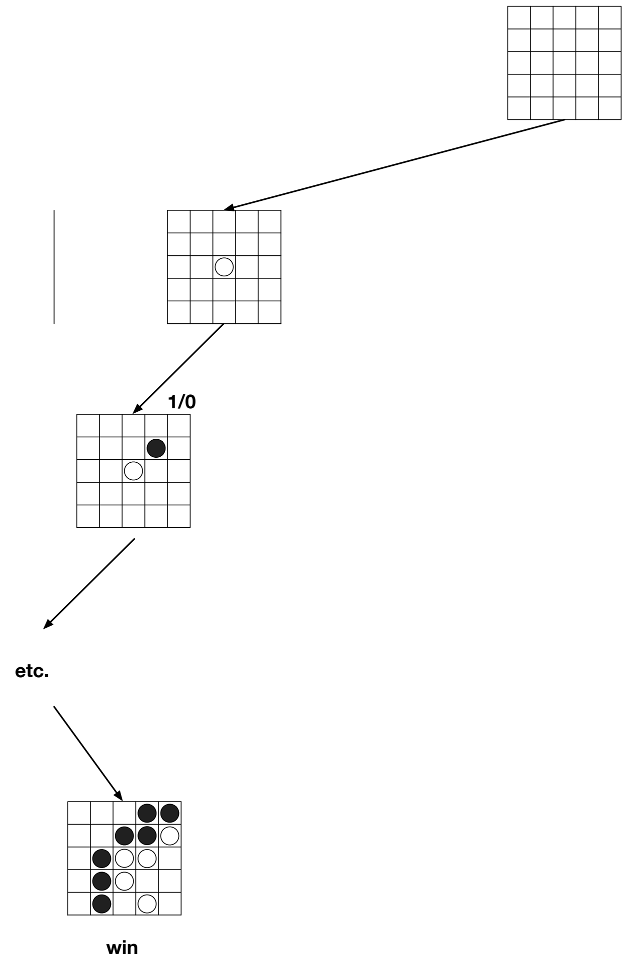
- This does not work for a game tree with about 361^{300} states

Evaluation Function for States

- Explore game tree up to some specified maximum depth
- Evaluate leaf states
  - informed by knowledge of game
  - e.g., chess: pawn count, control of board
- This does not work either due
  - high branching factor
  - difficulty of defining evaluation function

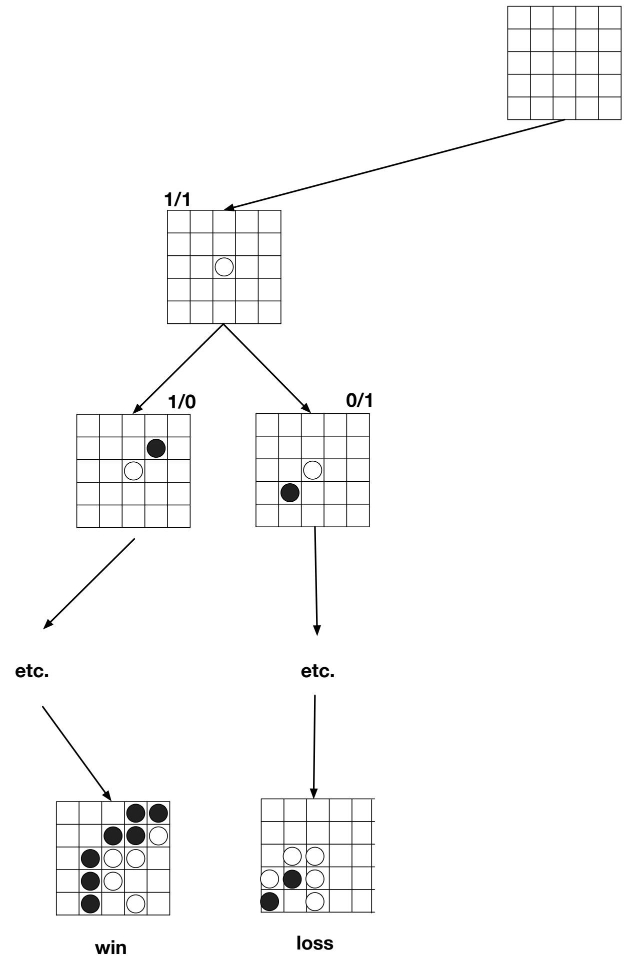
monte carlo tree search

Monte Carlo Tree Search



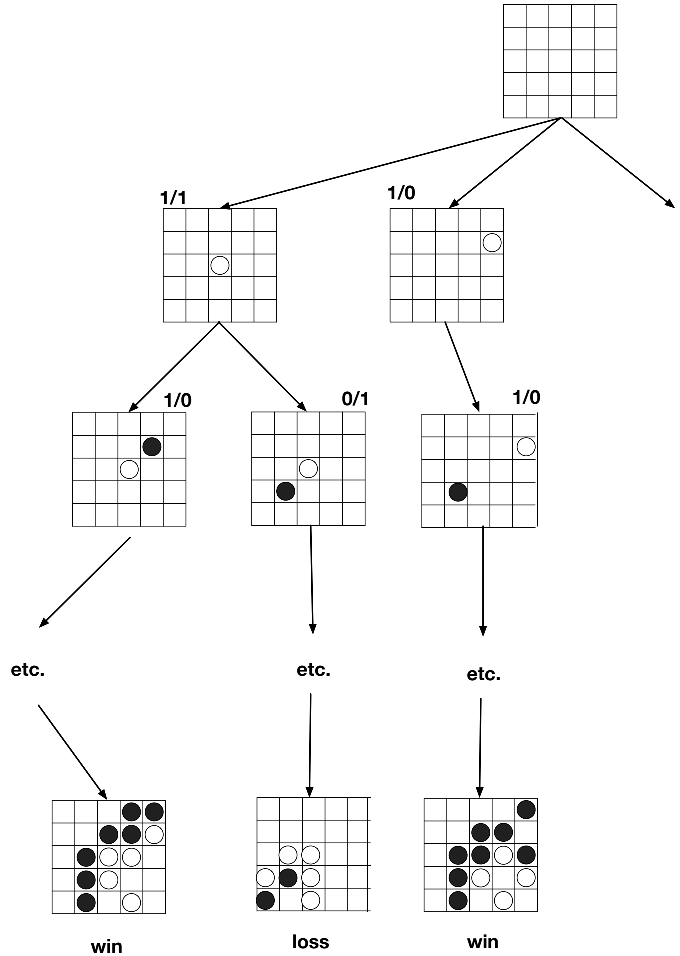
- Explore depth-first randomly (”roll-out”), record win on all states along path

Monte Carlo Tree Search



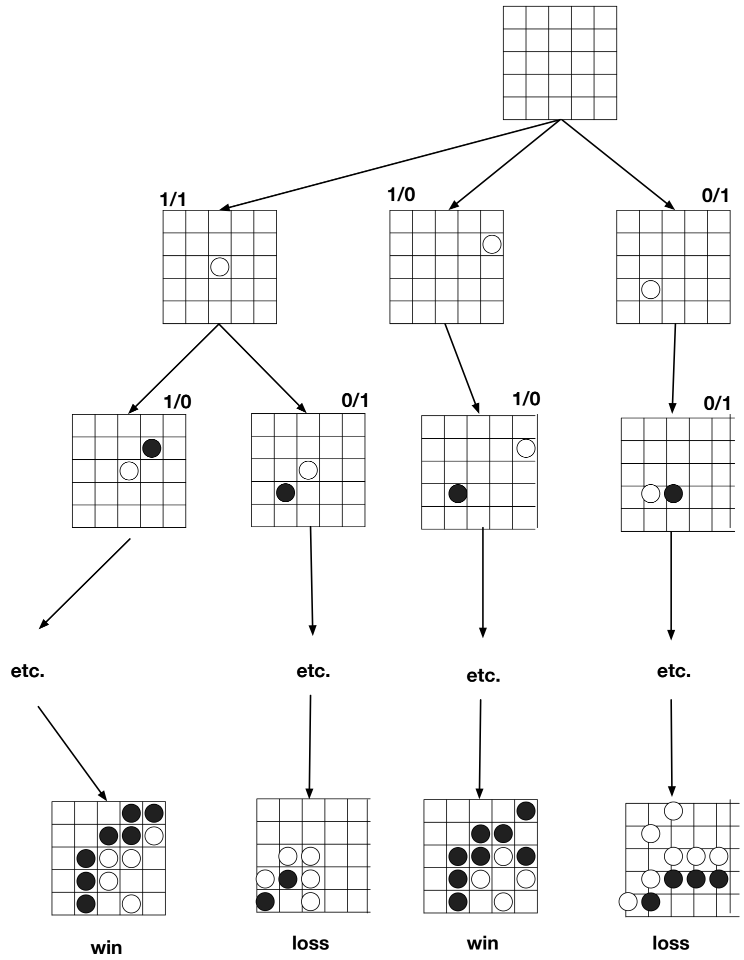
- Pick existing node as starting point, execute another roll-out, record loss

Monte Carlo Tree Search



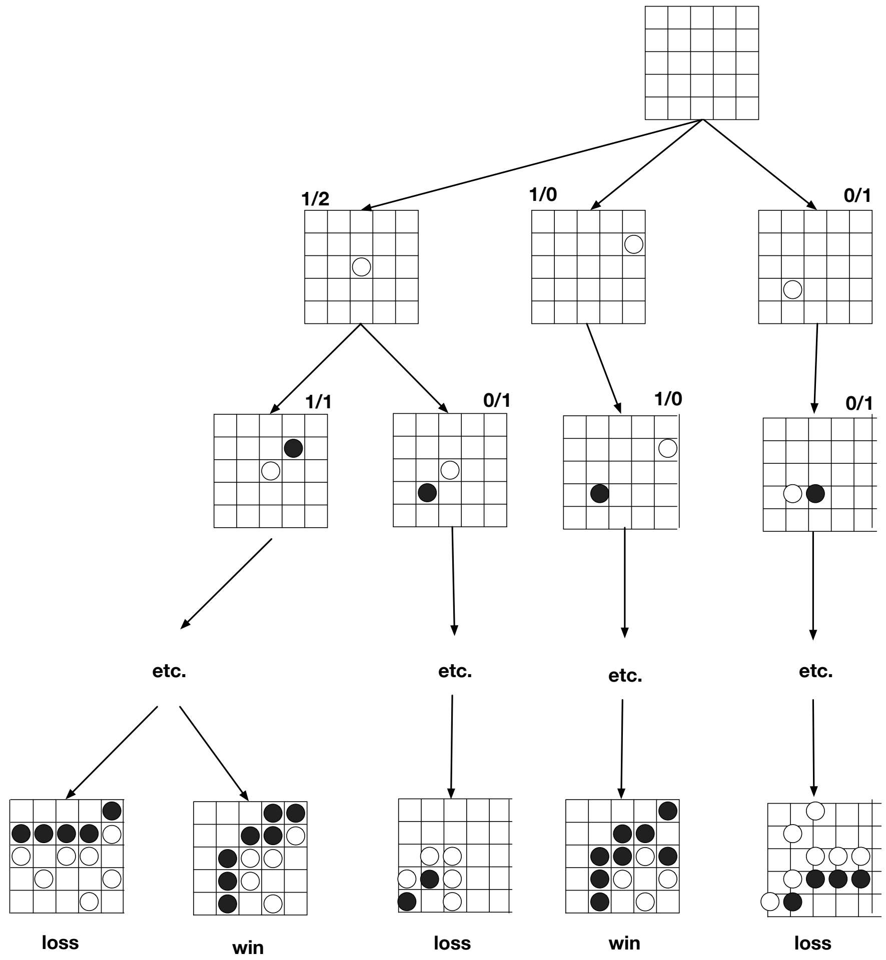
- Pick existing node as starting point, execute another roll-out

Monte Carlo Tree Search



- Pick existing node as starting point, execute another roll-out

Monte Carlo Tree Search



- Increasingly, prefer to explore paths with high win percentage

Monte Carlo Tree Search

- Which node to pick?

$$w + c\sqrt{\frac{\log N}{n}}$$

  - N total number of roll-outs
  - n number of roll-outs for this node in the game tree
  - w winning percentage
  - c hyper parameter to balance exploration

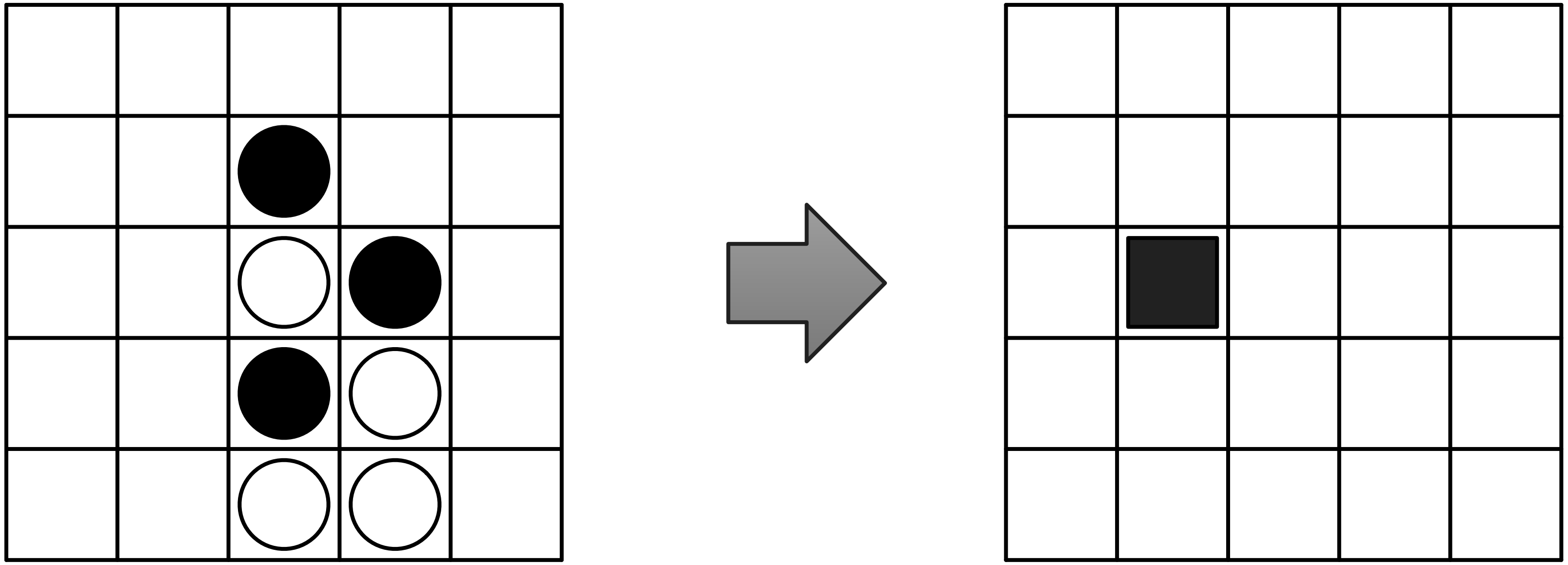
- This is an inference algorithm
  - execute, say, 10,000 roll-outs
  - pick initial action with best win percentage w
  - can be improved by following rules based on well-known local shapes

action prediction

with

neural networks

Learning Moves



- We would like to learn actions of game playing agent
- Input state: board position
- Output action: optimal move

Learning Moves

0	0	0	0	0
0	0	1	0	0
0	0	-1	1	0
0	0	1	-1	0
0	0	-1	-1	0

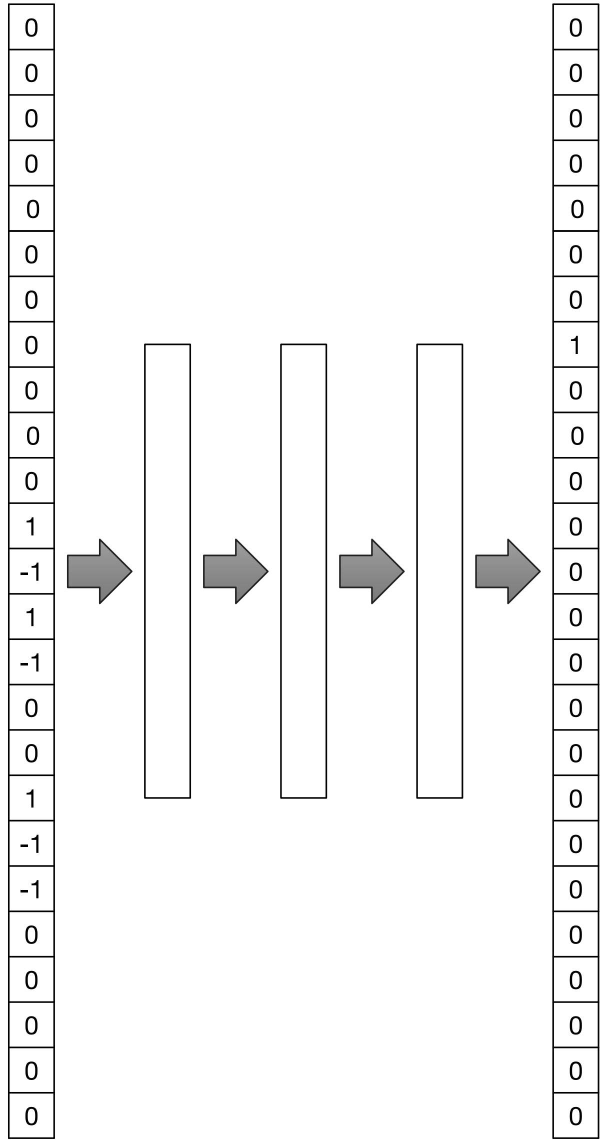
$\Rightarrow$

0	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	0	0	0	0
0	0	0	0	0

- Machine learning problem
- Input: 5x5 matrix
- Output: 5x5 matrix

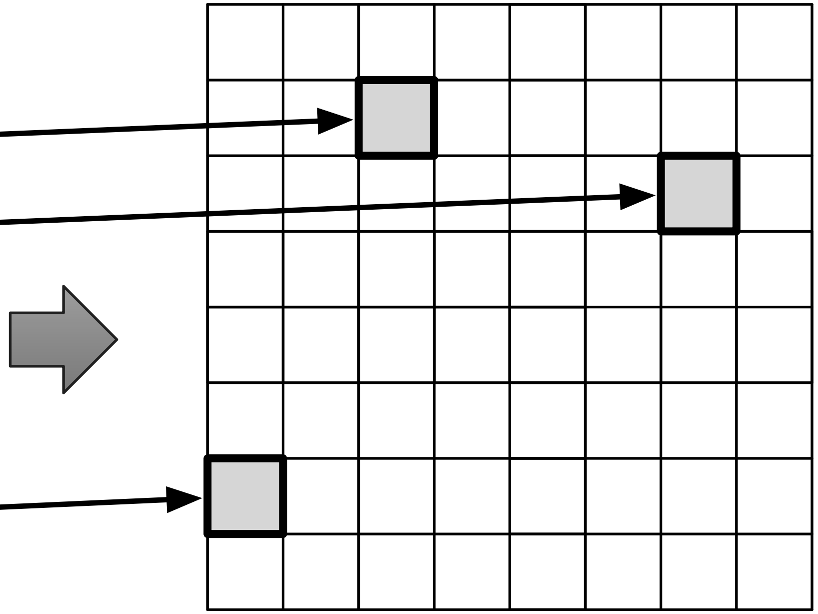
Neural Networks

- First idea: feed-forward neural network
  - encode board position in $n \times n$ sized vector
  - encode correct move in $n \times n$ sized vector
  - add some hidden layers
- Many parameters
  - input and output vectors have dimension 361 (19x19 board)
  - if hidden layers have same size $\rightarrow$ 361x361 weights for each
- Does not generalize well
  - same patterns on various locations of the board
  - has to learn moves for each location
  - consider everything moved one position to the right



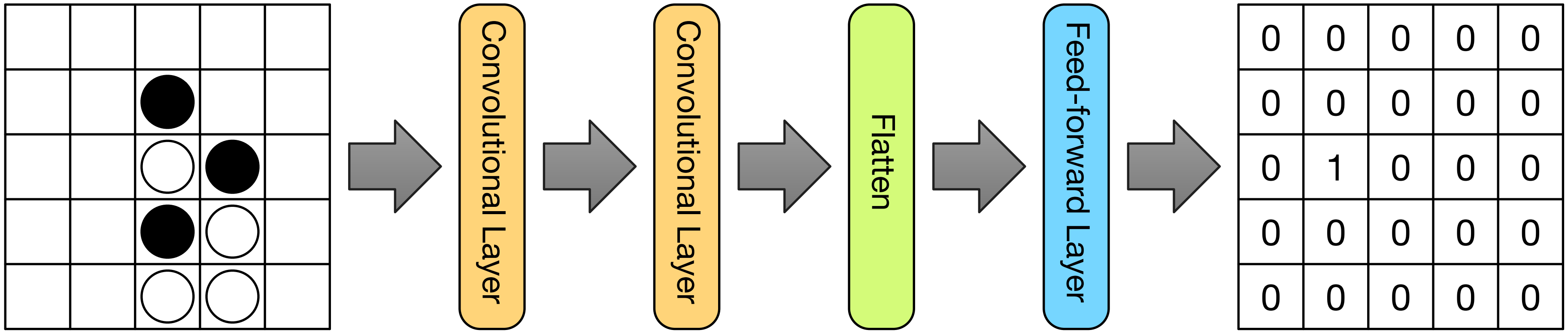
Convolutional Neural Networks

0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	-1	1	1	1	0	-1	1	0
0	0	1	-1	0	0	0	1	-1	0
0	0	-1	-1	0	0	0	-1	-1	0
0	0	0	0	-1	0	0	0	0	0
0	0	1	0	0	-1	0	1	0	0
0	0	-1	1	0	0	0	-1	1	0
0	0	0	0	0	0	0	0	-1	0
0	0	0	0	0	0	0	-1	0	0



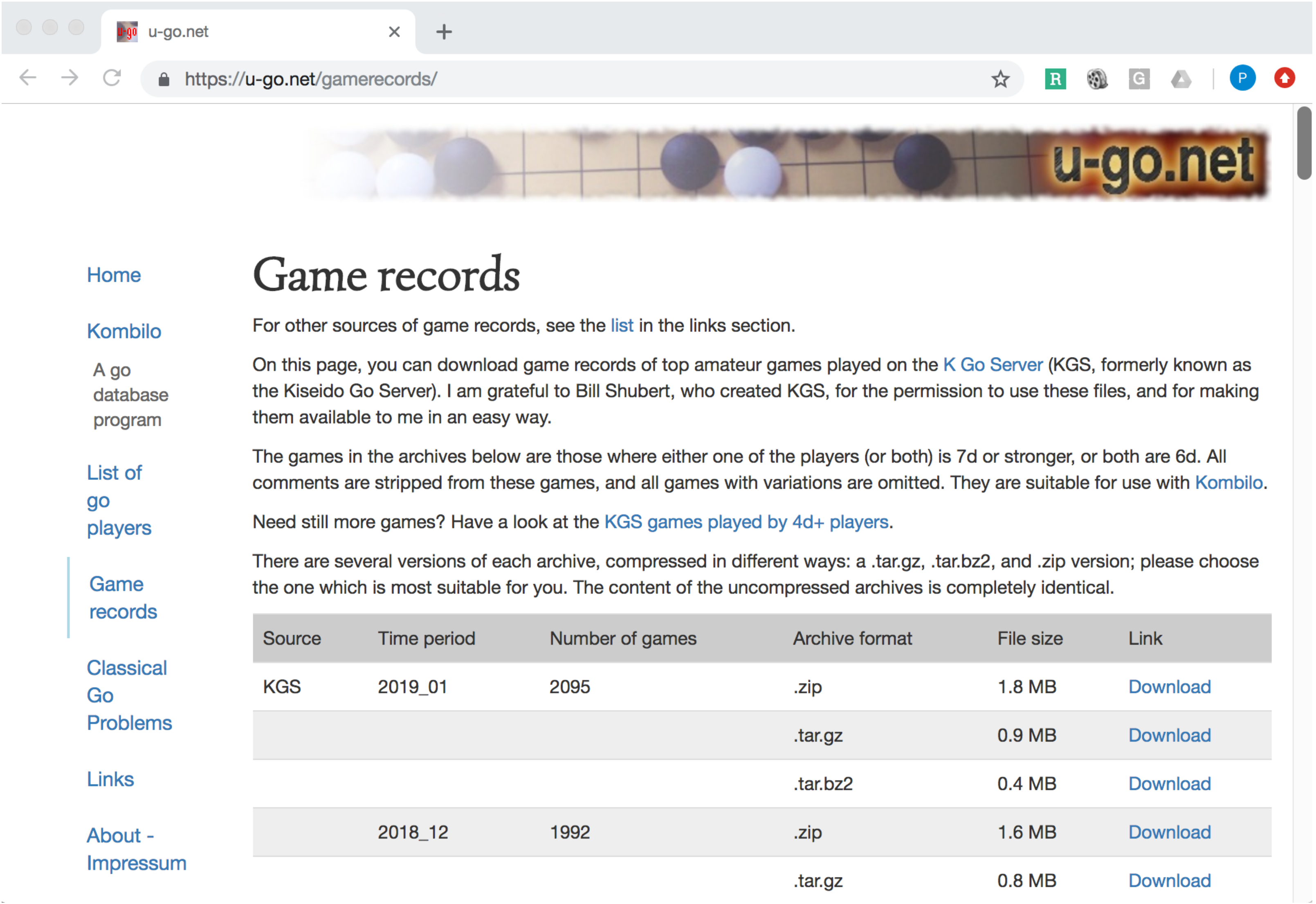
- Convolutional kernel: here maps 3x3 matrix to 1x1 value
- Applied to all 3x3 regions of the original matrix
- Learns local features

Move Prediction with CNNs



- May use multiple convolutional kernels (of same size)
  → learn different local features
- Resulting values may be added or maximum value selected (max-pooling)
- May have several convolutional neural network layers
- Final layer: softmax prediction of move

Human Game Play Data



u-go.net

https://u-go.net/gamerecords/

u-go.net

Home

Kombilo

A go database program

List of go players

Game records

Classical Go Problems

Links

About - Impressum

Game records

For other sources of game records, see the list in the links section.

On this page, you can download game records of top amateur games played on the K Go Server (KGS, formerly known as the Kiseido Go Server). I am grateful to Bill Shubert, who created KGS, for the permission to use these files, and for making them available to me in an easy way.

The games in the archives below are those where either one of the players (or both) is 7d or stronger, or both are 6d. All comments are stripped from these games, and all games with variations are omitted. They are suitable for use with Kombilo.

Need still more games? Have a look at the KGS games played by 4d+ players.

There are several versions of each archive, compressed in different ways: a .tar.gz, .tar.bz2, and .zip version; please choose the one which is most suitable for you. The content of the uncompressed archives is completely identical.

Source	Time period	Number of games	Archive format	File size	Link
KGS	2019_01	2095	.zip	1.8 MB	Download
			.tar.gz	0.9 MB	Download
			.tar.bz2	0.4 MB	Download
	2018_12	1992	.zip	1.6 MB	Download
			.tar.gz	0.8 MB	Download

Human Game Play Data

- Game records
  - sequence of moves
  - winning player
- Convert into training data for move prediction
  - one move at a time
  - prediction $+1$ for move if winner
  - prediction -1 for move if loser
- learn winning moves, avoid losing moves

Playing Go with Neural Move Predictor

- Greedy search
- Make prediction at each turn
- Selection move with highest probability

reinforcement learning

Self-Play

- Previously: learn policy from human play data
- Now: learn policy from self-play
- Need to have an agent that plays reasonably well to start

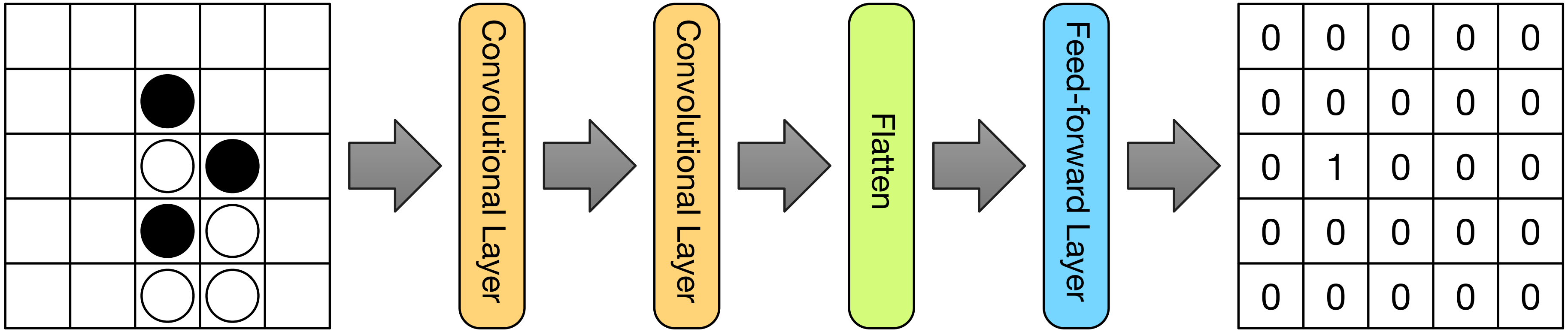
  → learn initial policy from human play data
- Greedy move selection with same policy will result in the same game each time
  - stochastic moves:
    move predicted with 80% confidence → select it 80% of the time
  - may have to clip probabilities that are too certain (e.g., 99.9% to 80%)

Experience from Self-Play

- Self play will generate self play data (”experience”)
  - sequence of moves
  - winner at the end

- Can be used as training data to improve model
  - first train model on human play data
  - then, run 1 epoch over self-play data

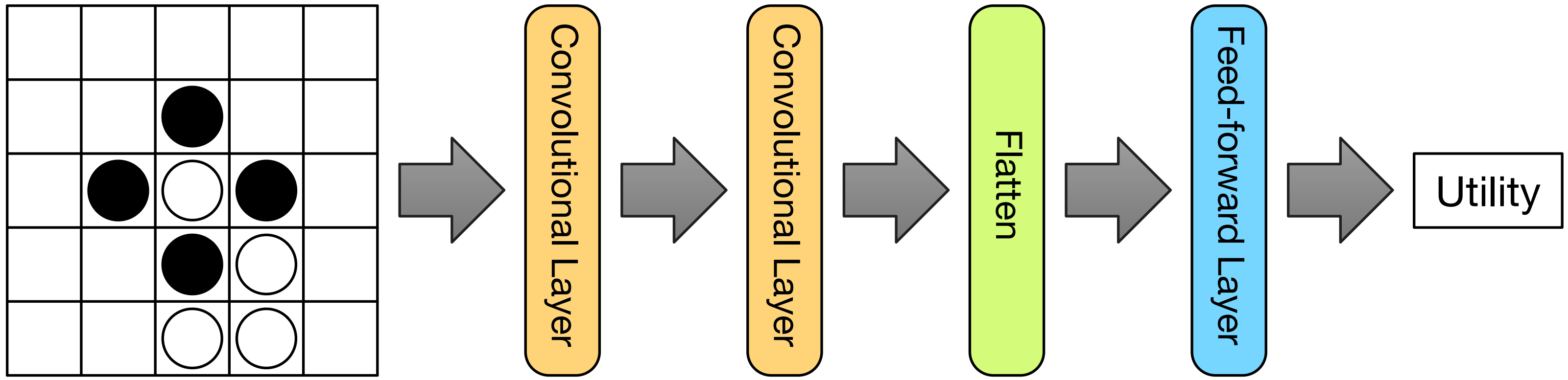
Policy Search



- Reminder: policy informs which action to take in each state

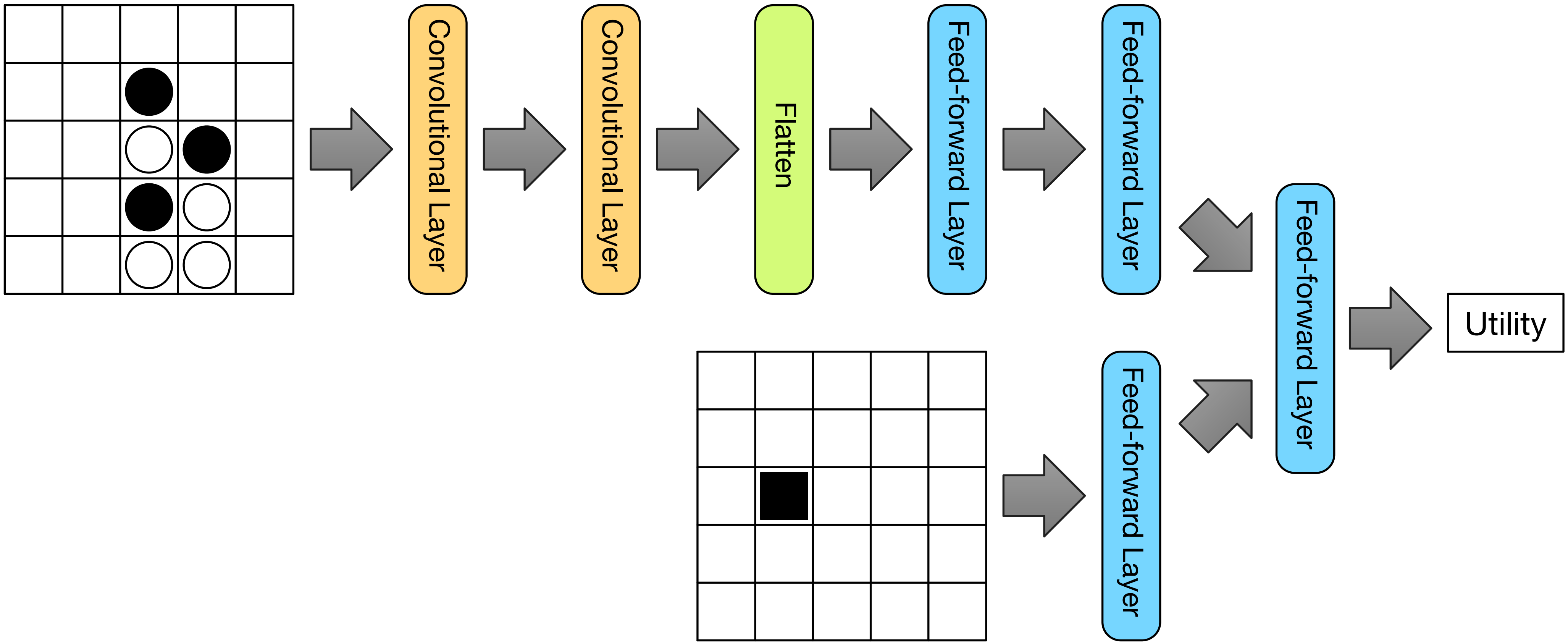
- Learning move predictor = learning policy

Q Learning



- Learn utility value for each state = likelihood of winning
- Training on game play data, utility=1 for win, 0 for loss
- Game play with utility predictor
  - consider all possible actions
  - compute utility value for resulting state
  - choose action with maximum utility outcome

Q Learning



- Alternative architecture
- Explicitly modeling the last move: Q(state,action)

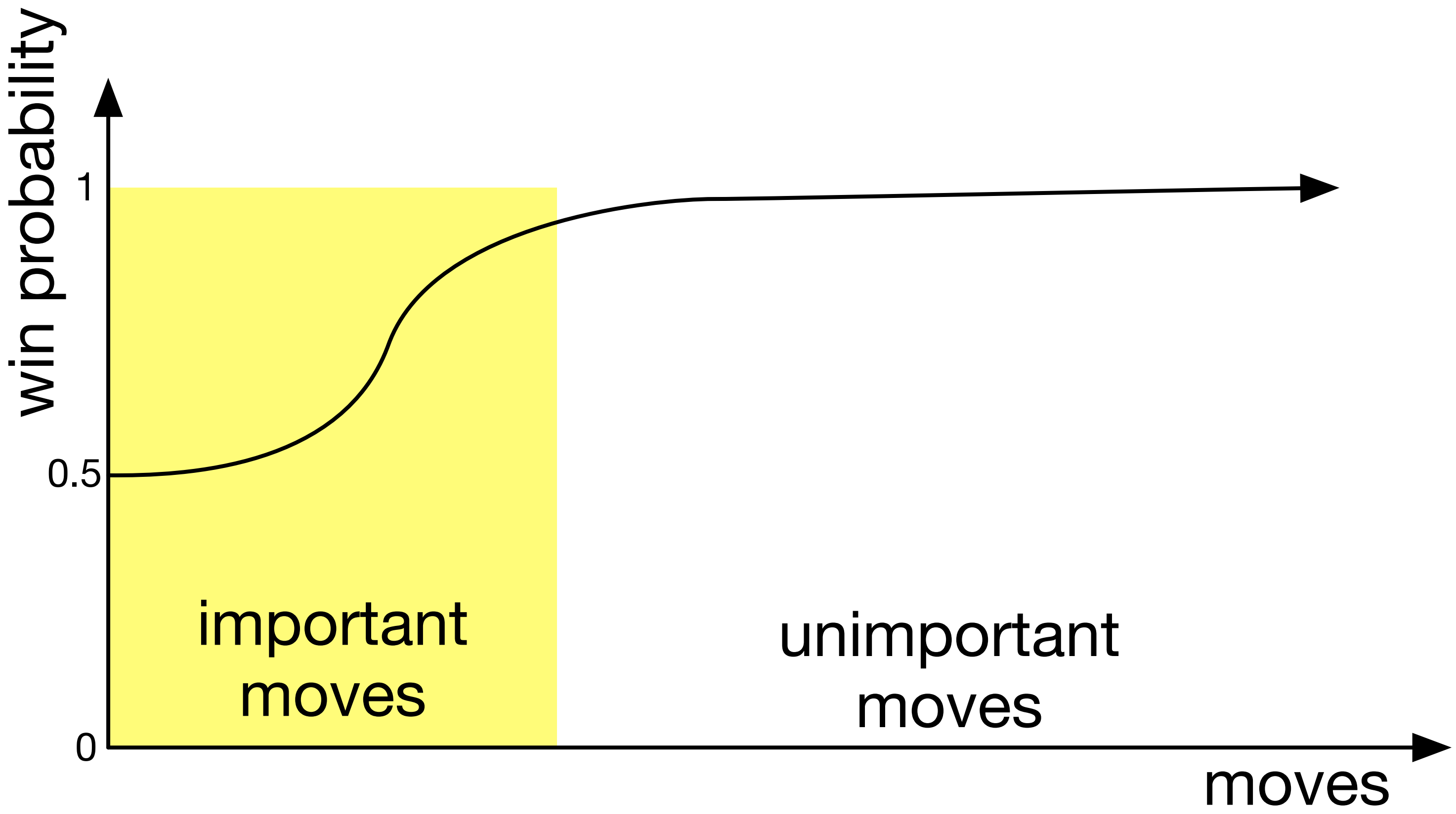
actor-critic learning

Credit Assignment Problem

- Go game lasts many moves (say, 300 moves)
  - some of the moves are good
  - some of the moves are bad
  - some of the moves make no difference

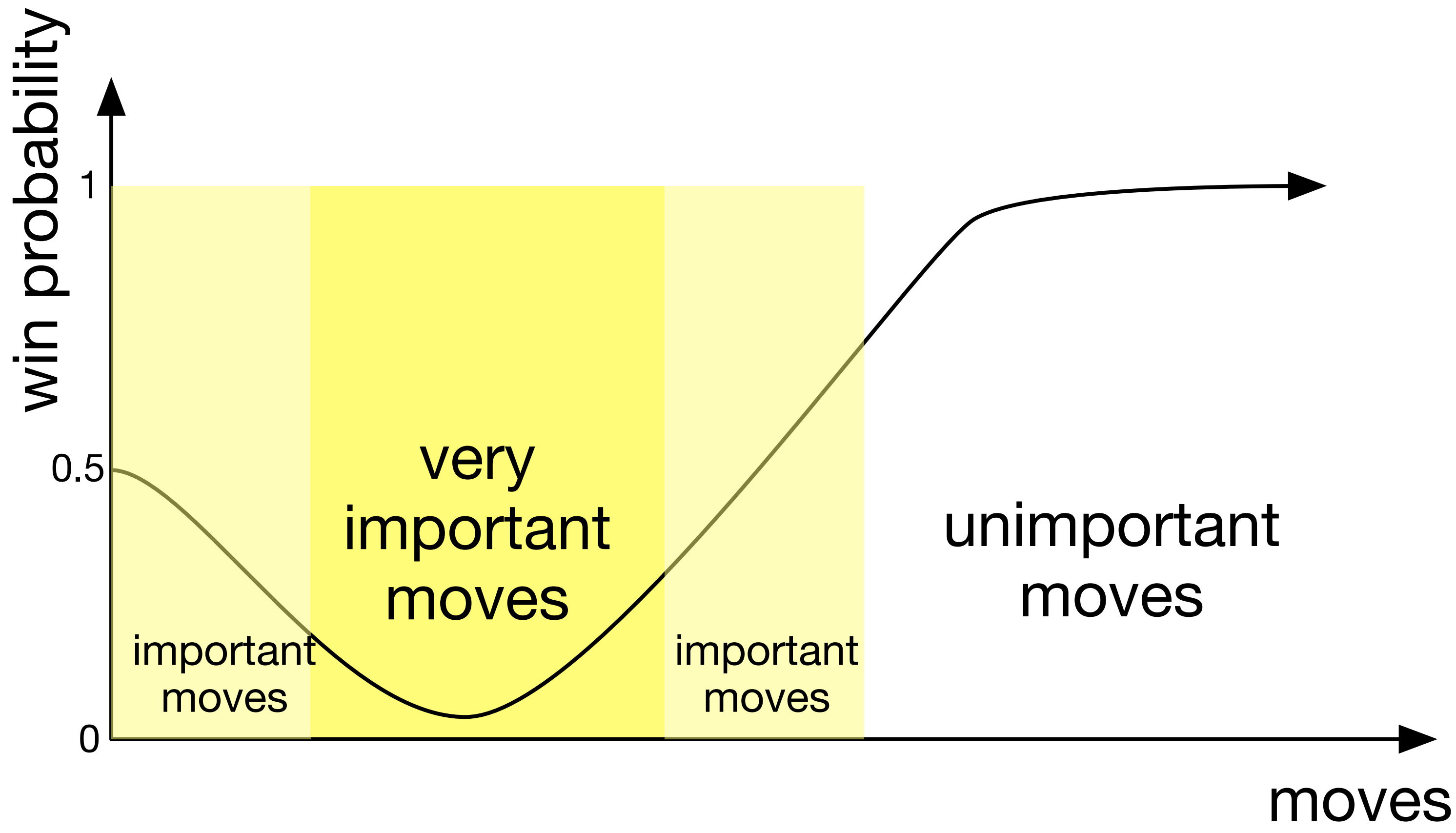
- We want to learn from the moves that made a difference
  - before: low chance of winning
  - move
  - at the end → win

Consider Win Probability



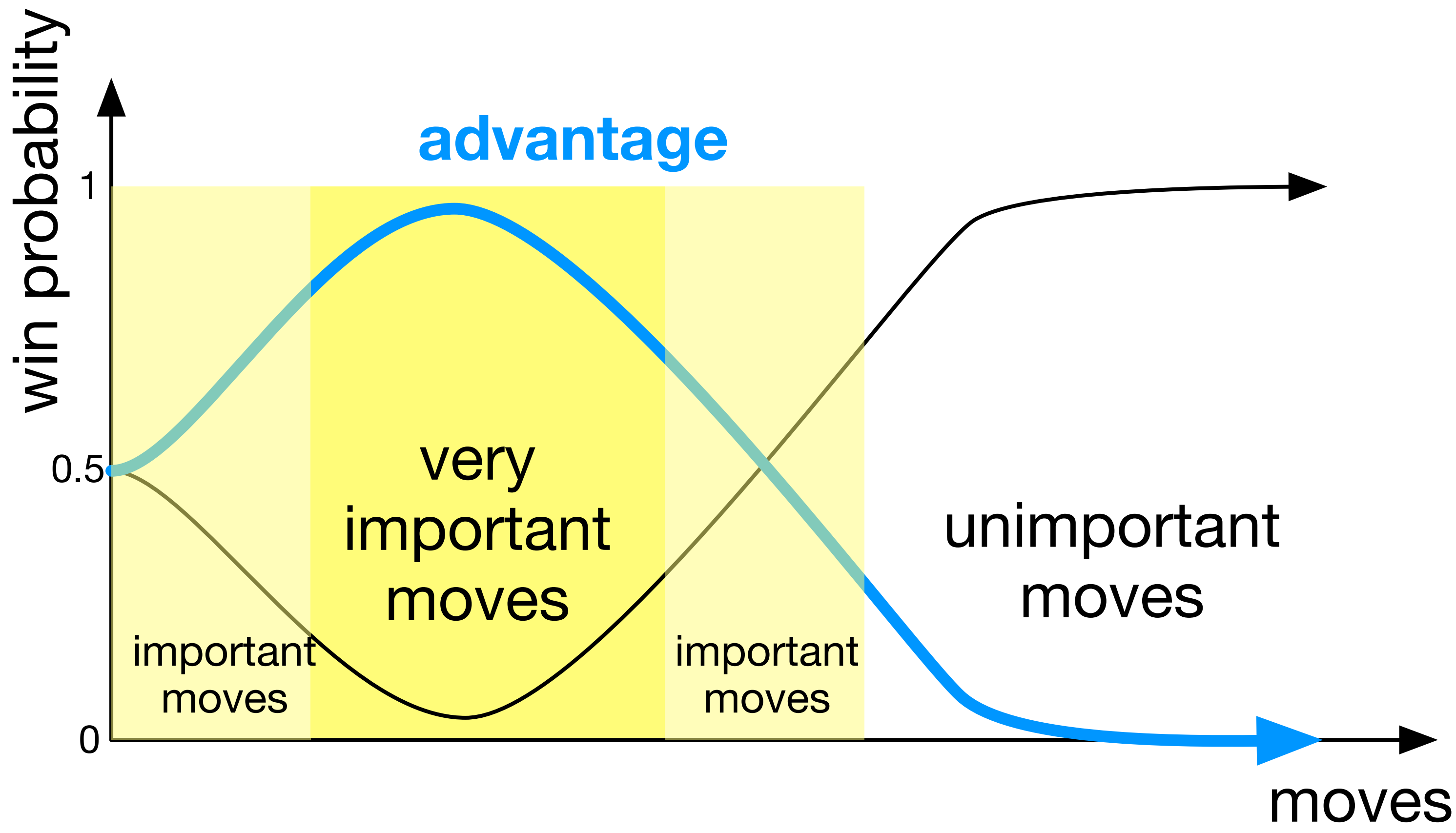
- Moves that pushed towards win matter more

Consider Win Probability



- Especially important moves: change from losing position to winning position

Advantage



Compute utility of state $V(s)$
Definition of advantage: $A = R - V(s)$
(R = final reward)

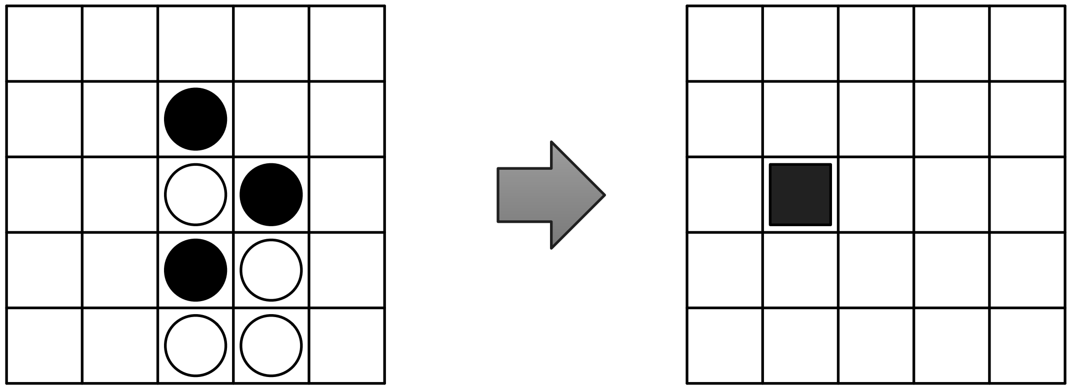
Actor-Critic Learning

- Combination of policy learning and Q learning
  - actor: move predictor (as in policy learning) $s \rightarrow a$
  - critic: value of state (as in Q learning) $V(s)$

- We use this setup to influence how much to boost good moves
  - advantage $A = R - V(s)$
  - good moves when advantage is high

Policy Learning with Advantage

- Before: predict win

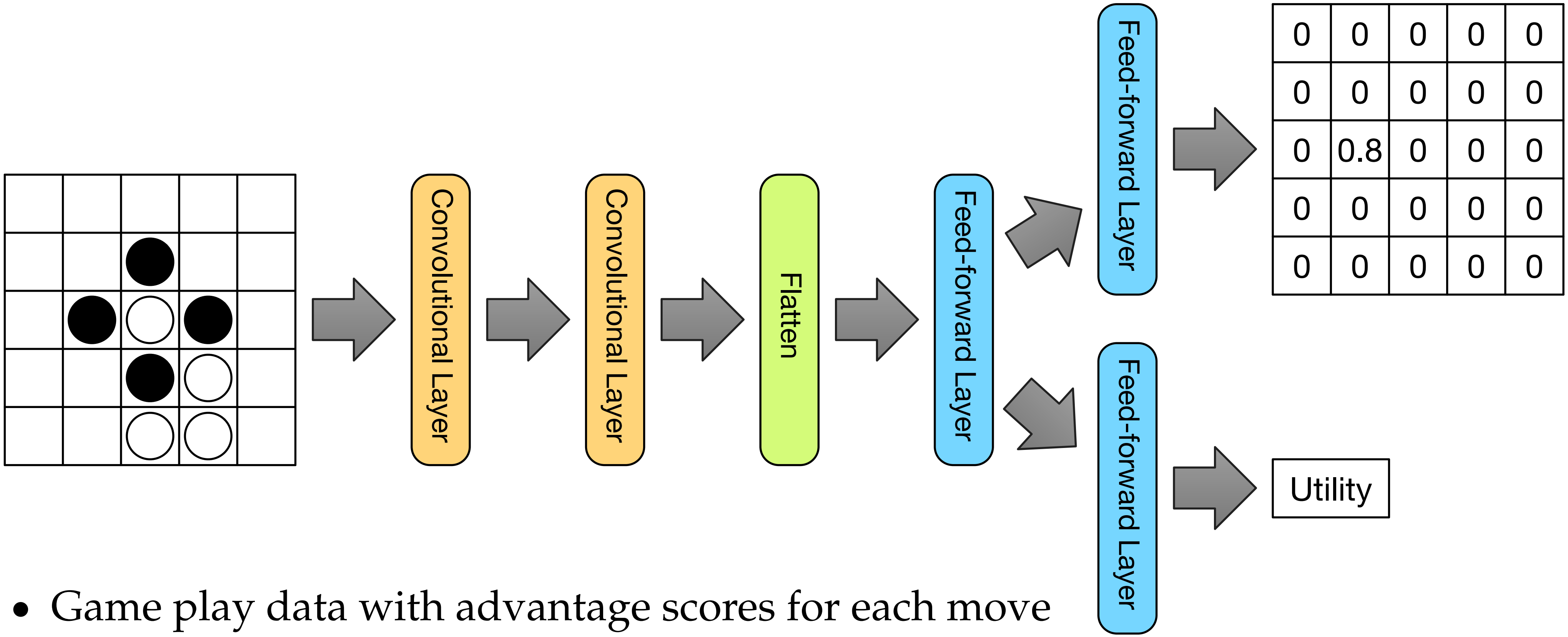


- Now: predict advantage

0	0	0	0	0
0	0	1	0	0
0	0	-1	1	0
0	0	1	-1	0
0	0	-1	-1	0

0	0	0	0	0
0	0	0	0	0
0	0.8	0	0	0
0	0	0	0	0
0	0	0	0	0

Architecture of Actor-Critic Model



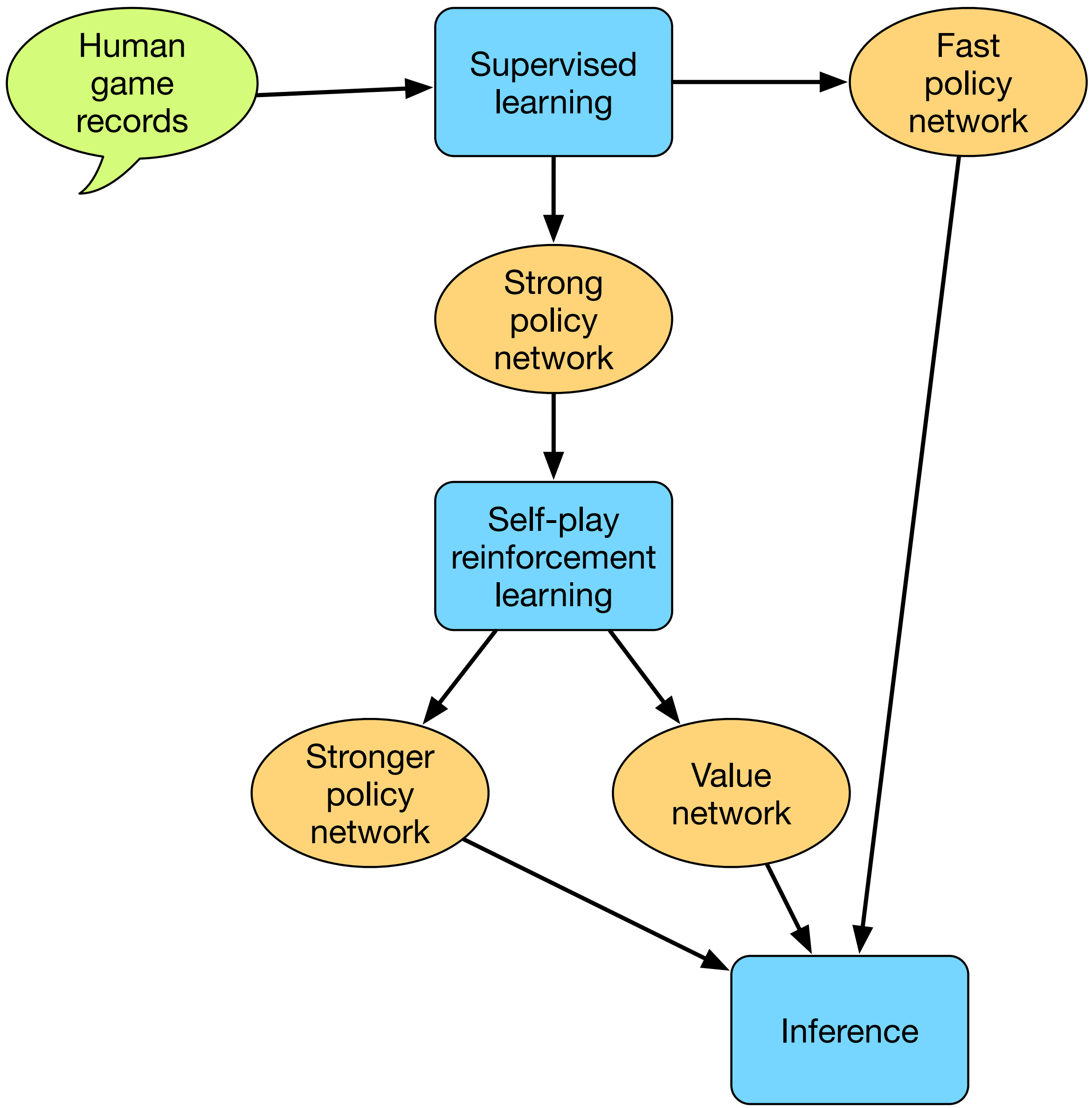
- Game play data with advantage scores for each move
- Training of actor and critic similar

⇒ Share components, train them jointly

- Multi-task learning helps regularization

alpha go

Overview


Human game records
Supervised learning
Fast policy network
Strong policy network
Self-play reinforcement learning
Stronger policy network
Value network
Inference

Encoding the Board

- We encoded each board position with a integer (+1=white, –1=black, 0=blank)

- AlphaGo uses a 48-dimensional vector that encode knowledge about the game
  - 3 booleans for stone color
  - 1 boolean for legal and fundamentally sensible move
  - 8 boolean to record how far back stone was placed
  - 8 booleans to encode *liberty*
  - 8 booleans to encode *liberty* after move
  - 8 booleans to encode *capture* size
  - 8 booleans to encode how many of your own stones will be placed in jeopardy because of move
  - 2 booleans for *ladder* detection
  - 3 booleans for technical values

- Note: *ladder*, *liberty*, and *capture* are basic concepts of the game

Policy and Value Networks

- Policy network: $s \rightarrow a$
- Value network: $s \rightarrow V(s)$
- These networks are trained as previously described
- Fairly deep networks
  - 13 layers for policy network
  - 16 layers for value network

Monte Carlo Tree Search

- Inference uses a refined version of Monte Carlo Tree Search (MCTS)
- Roll-out guided by fast policy network (greedy search)
- When visiting a node with some unexplored children ("leaf")
  → use probability distribution from strong policy network for stochastic choice
- Combine roll-out statistics with prediction from value network

MCTS with Value Network

- Estimate value of a leaf node l in the game tree where a roll-out started as

$$V(l) = \frac{1}{2}\,\text{value}(l) + \frac{1}{2}\,\text{roll-out}(l)$$

  – $\text{value}(l)$ is prediction from value network
  – $\text{roll-out}(l)$ is win percentage from Monte Carlo Tree Search

- This is used to compute Q values for any state-action pair given its leaf nodes l_i

$$Q(s,a) = \frac{\sum_i V(l_i)}{N(s,a)}$$

- Combine with the prediction of the strong policy network $P(s,a)$

$$a' = \text{argmax}_a Q(s,a) + \frac{P(s,a)}{1 + N(s,a)}$$

alpha go zero

Less and More

- Less
  - no pre-training with human game play data
  - no hand crafted features in board encoding
  - no Monte Carlo rollouts
- More
  - 80 convolutional layers
  - tree search also used in self-play

Improved Tree Search

- Tree search adds one node in each iteration (not full roll-out)

- When exploring a new node
  - compute its Q value
  - compute action prediction probability distribution
  - pass Q value back up through search tree

- Each node in search tree keeps record of
  - P prediction for action leading to this node
  - Q average of all terminal Q values from visits passing through node
  - N number of visits of parent
  - n number of visits of node

- Score of node (c is hyper parameter to be optimized)

$$Q + cP\frac{\sqrt{N}}{1+n}$$

Inference and Training

- Inference
  - choose action from most visited branch
  - visit count is impacted by both action prediction and success in tree search
  - → more reliable than win statistics or raw action prediction
- Training
  - predict visit count

and more...

Google's AlphaZero Destroys Stockfish In 100-Game Match

FM MikeKlein
Dec 6, 2017, 12:50 PM | 351 | Chess Event Coverage

English

Chess changed forever today. And maybe the rest of the world did, too.

StarCraft is a deep, complicated war strategy game. Google's AlphaStar AI crushed it.

DeepMind has conquered chess and Go and moved on to complex real-time games. Now it's beating pro gamers 10-1.

By Kelsey Piper | Updated Jan 24, 2019, 7:04pm EST