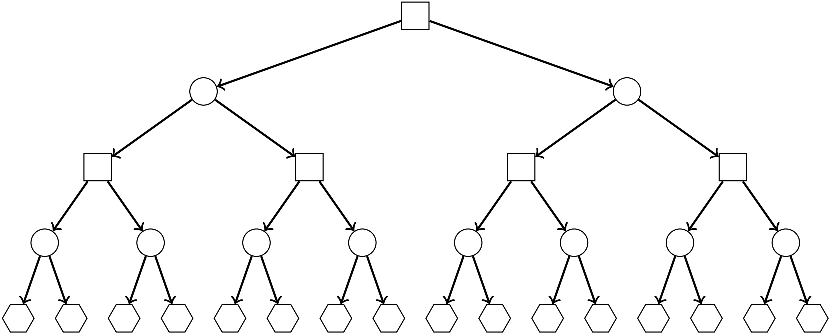
Artificial Intelligence for Perfect Information Games

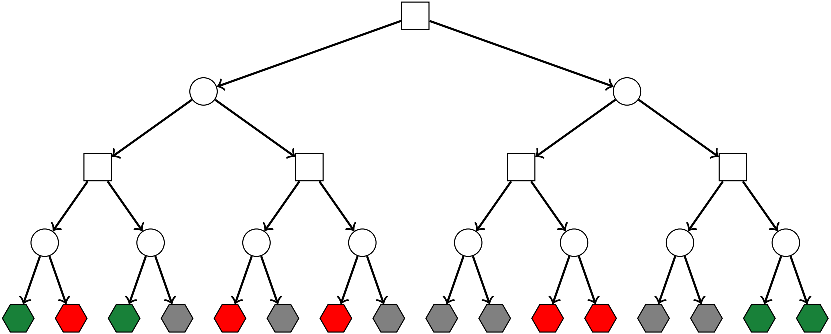
Thibault Gauthier

November 8, 2017

Solving a game

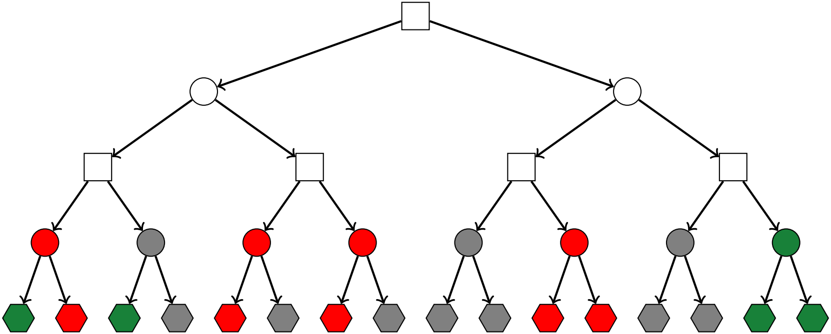


Solving a game



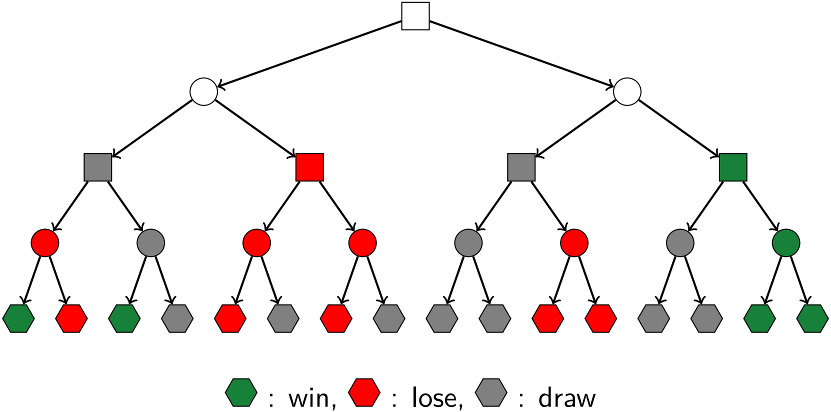
: win, : lose, : draw

Solving a game

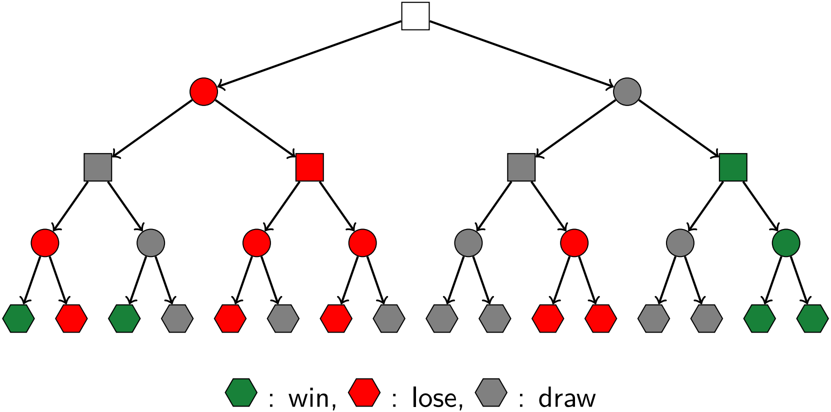


: win, : lose, : draw

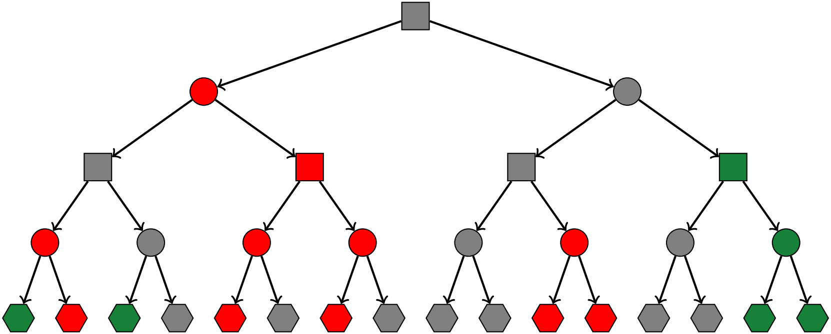
Solving a game



Solving a game



Solving a game



: win, : lose, : draw

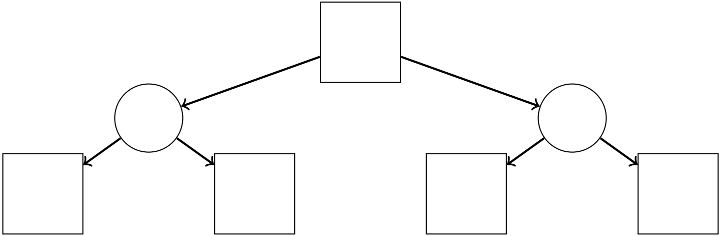
Game complexity

Game	Moves	Length	Positions	Solved
Tic-Tac-Toe	4	9	10^3	
Checkers	2.8	30	10^{20}	
Chess	35	70	10^{47}	
Go	250	150	10^{170}	

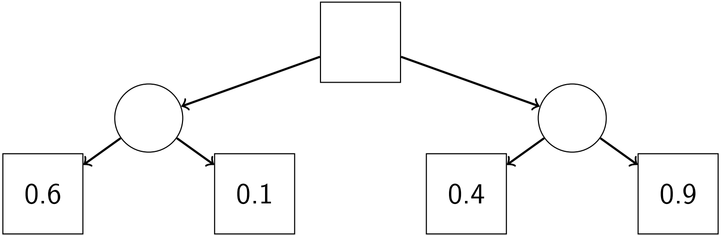
Game complexity

Game	Moves	Length	Positions	Solved
Tic-Tac-Toe	4	9	10^3	Yes
Checkers	2.8	30	10^{20}	Yes
Chess	35	70	10^{47}	No
Go	250	150	10^{170}	No

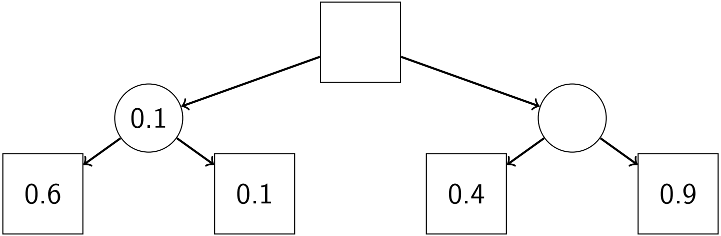
Minimax algorithm



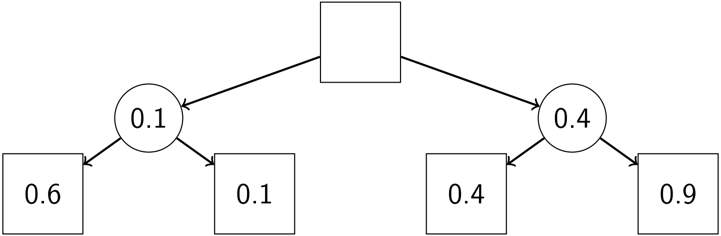
Minimax algorithm



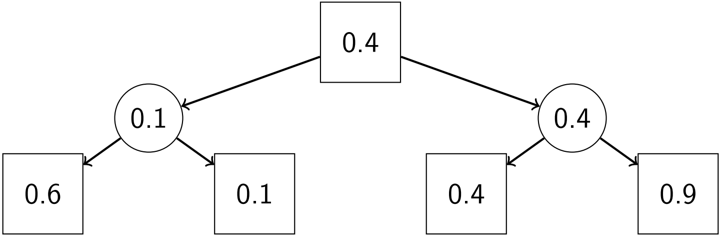
Minimax algorithm



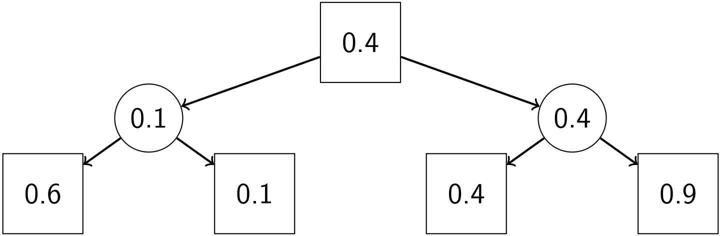
Minimax algorithm



Minimax algorithm



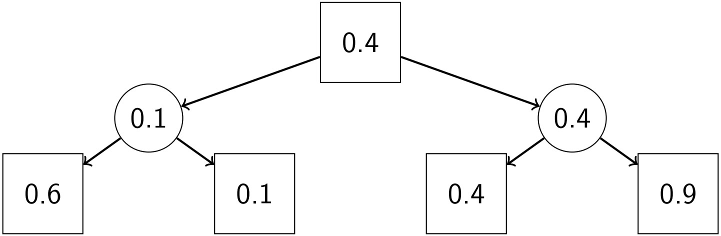
Evaluation and policy



$$Evaluation: \quad Position \to \mathbb{R}$$
$$Policy: \quad Position \to \mathbb{R}^{Move}$$

$$Evaluation(root) =$$
$$Policy(root) =$$

Evaluation and policy



$Evaluation: \ \ Position \rightarrow \mathbb{R}$
$Policy: \ \ Position \rightarrow \mathbb{R}^{Move}$

$Evaluation(root) = 0.4$
$Policy(root) = Left\ 0\%\ Right\ 100\%$

Heuristics

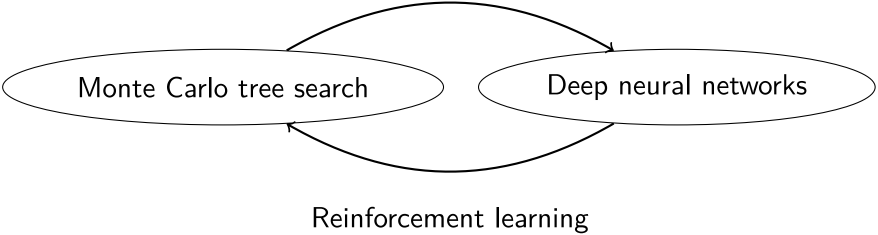
In chess:

- Evaluation: value of pieces, king's safety, pawn doubling, . . .
- Policy: derived from evaluation
- Search: minimax + alpha-beta pruning + . . .

In go (before 2016):

- Evaluation: random playouts
- Policy: statistics on good shapes
- Search: Monte Carlo tree search

AlphaGo



[2] Mastering the Game of Go without Human Knowledge
David Silver, Julian Schrittwieser, Karen Simonyan et al.
Nature 550, 354–359, 2017

Monte Carlo tree search with priors

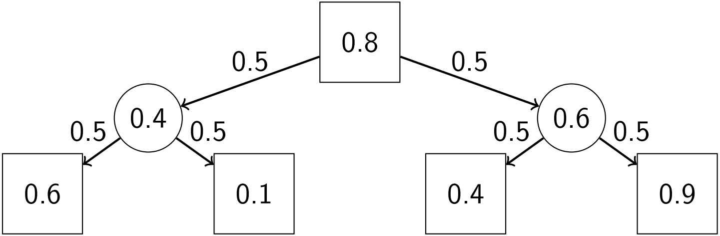
Let P_{prior} be a fixed policy and E_{prior} be a fixed evaluation.
MCTS computes a "better" evaluation $E_{average}$ and policy $P_{average}$.

1. Choose a sequence of moves leading to a position s based on:

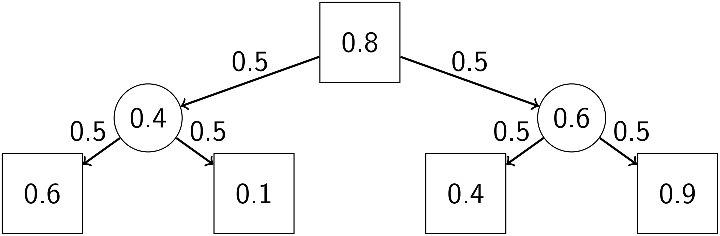
$$E_{average} + \frac{P_{prior}}{P_{average}}$$

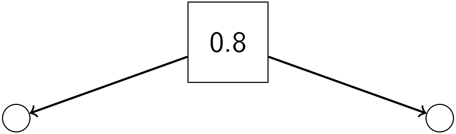
2. Extend the tree by applying all possible moves from s.
3. Evaluate s using E_{prior}.
4. Update $E_{average}$ and $P_{average}$ for all ancestors of s.

Monte Carlo tree search with priors

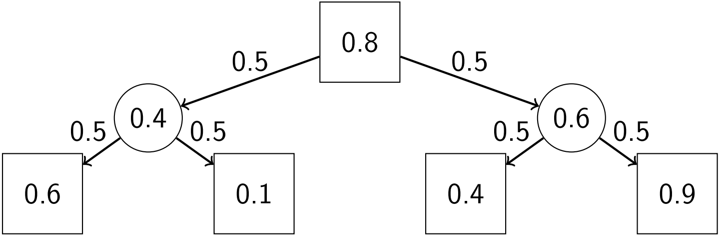


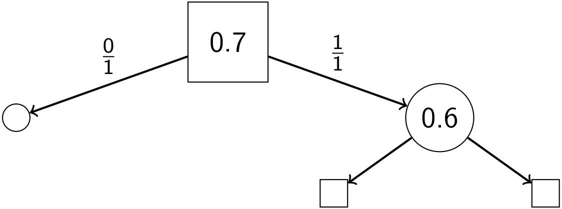
Monte Carlo tree search with priors



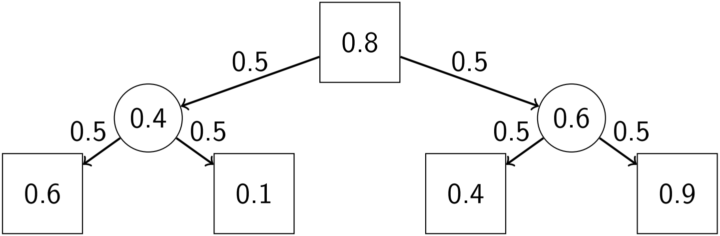


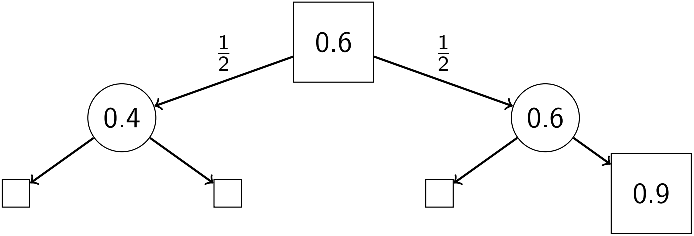
Monte Carlo tree search with priors



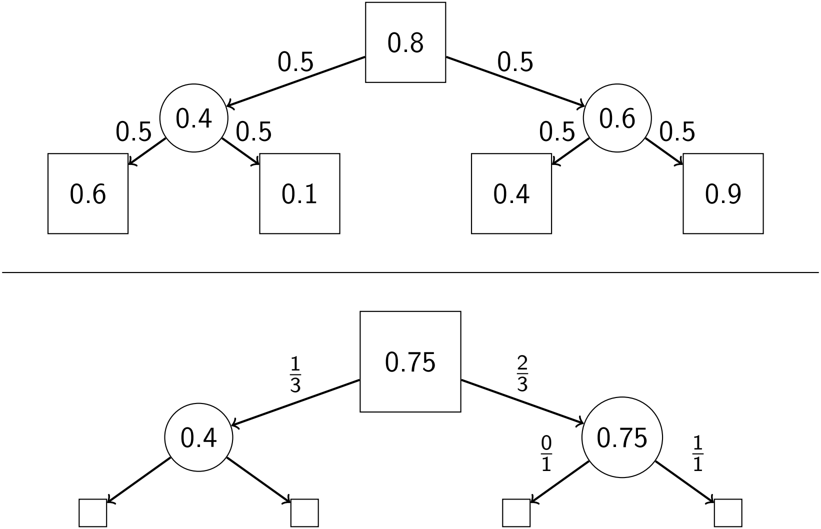


Monte Carlo tree search with priors





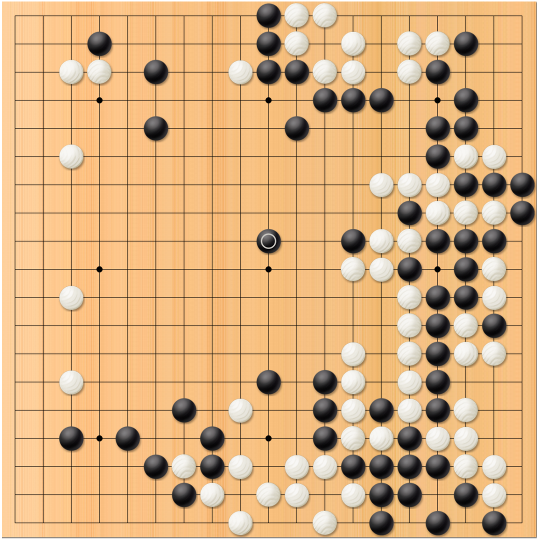
Monte Carlo tree search with priors



Monte Carlo tree search with priors: summary

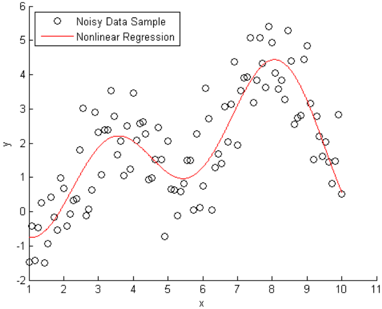
1. Input: Prior evaluation, policy.
2. Simulations from a starting position s_{start}
3. Output: Example of a better evaluation and policy for s_{start}
4. Repeat the process with different starting position for more examples.

Go is mostly a pattern recognition game



Machine learning

From data points (training examples), to generalization and compression of information.



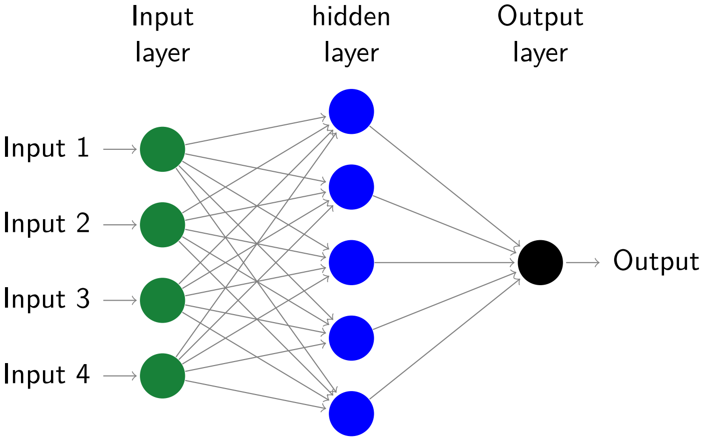
`https://datascience.stackexchange.com/questions/9529/how-to-select-regression-algorithm-for-noisy-scattered-data/9535`

Machine learning for pictures input: Convolutional neural networks

- 2012: Distinguishing cats from dogs in pictures.
- 2015: Predicting the next played move with 57% accuracy high amateur games at the game of Go.

Neural networks

Each arrow is associated with a weight use to multiply its input.



The hidden layer is fully-connected.

Training a neural network

Type of examples:

neural network	input-output
policy network	position-policy
value network	position-evaluation

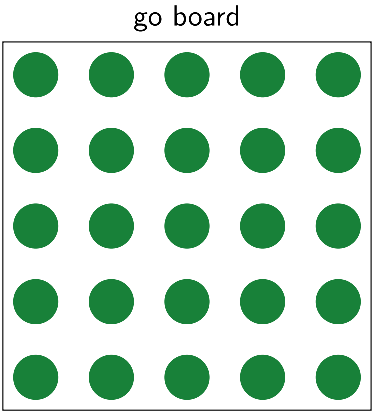
Generating examples:

- from professional games
- from self-play

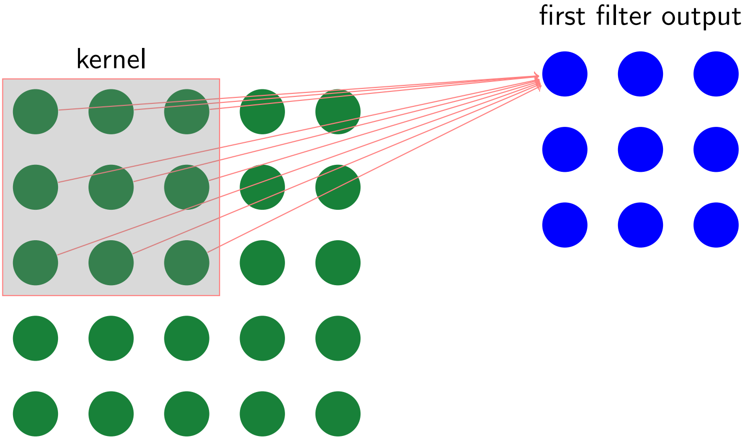
Training algorithm:

- modification of the weights by gradient descent (backpropagation)

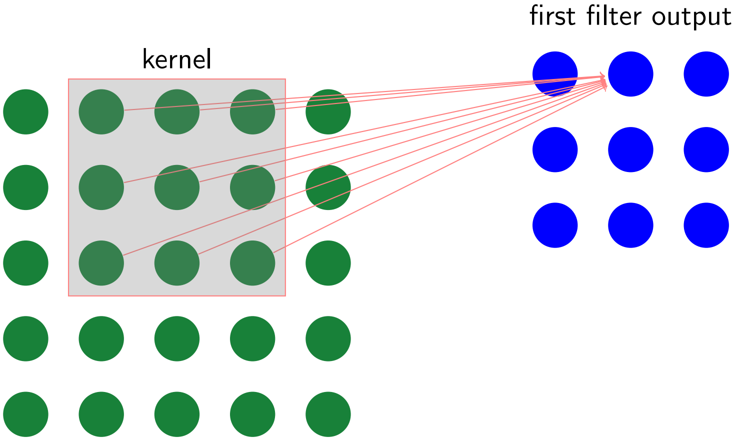
A convolution of 256 filters of kernel size 3x3 with stride 1



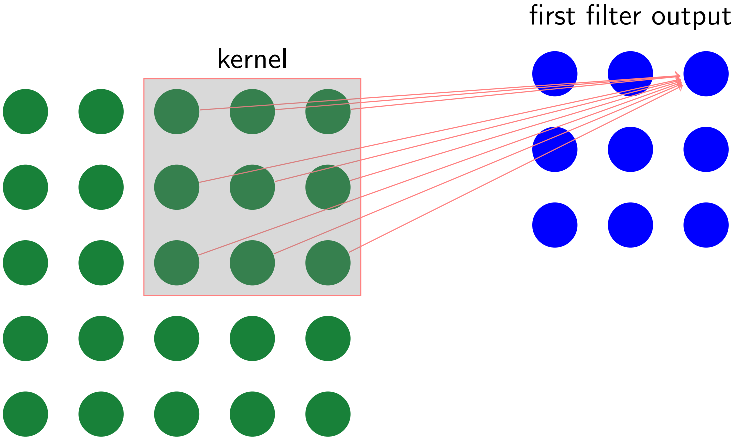
A convolution of 256 filters of kernel size 3x3 with stride 1



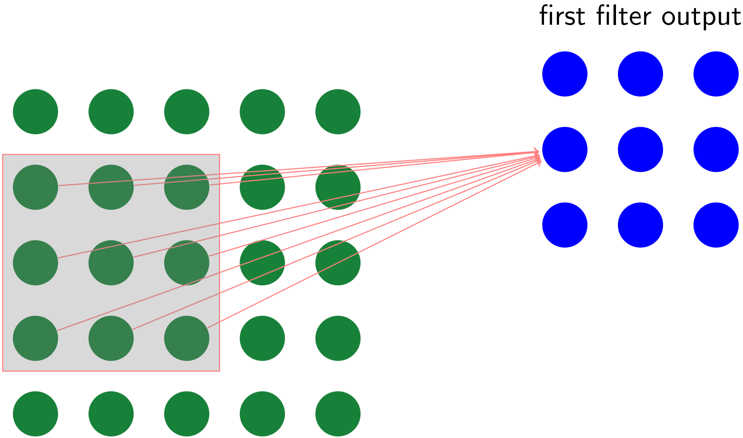
A convolution of 256 filters of kernel size 3x3 with stride 1



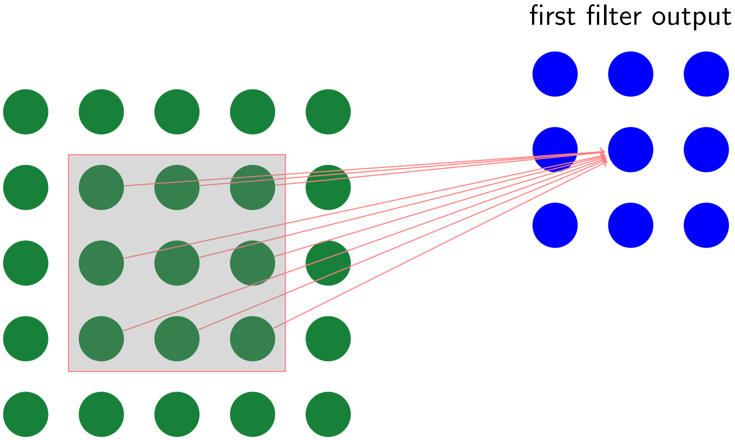
A convolution of 256 filters of kernel size 3x3 with stride 1



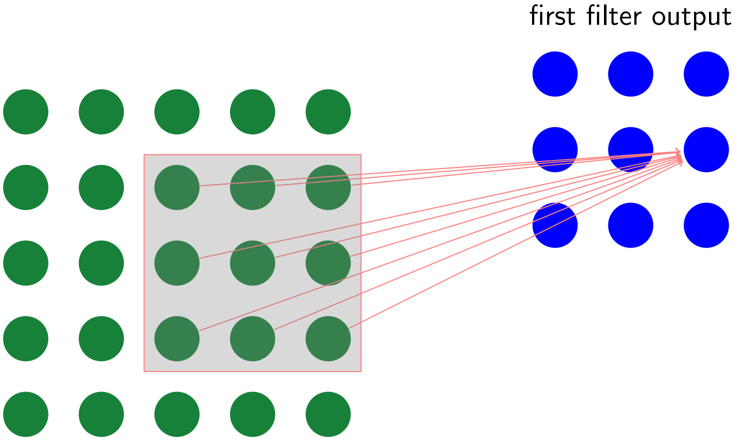
A convolution of 256 filters of kernel size 3x3 with stride 1



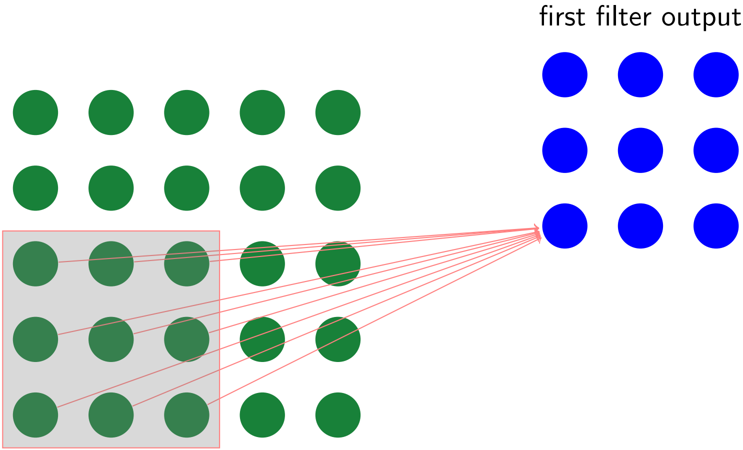
A convolution of 256 filters of kernel size 3x3 with stride 1



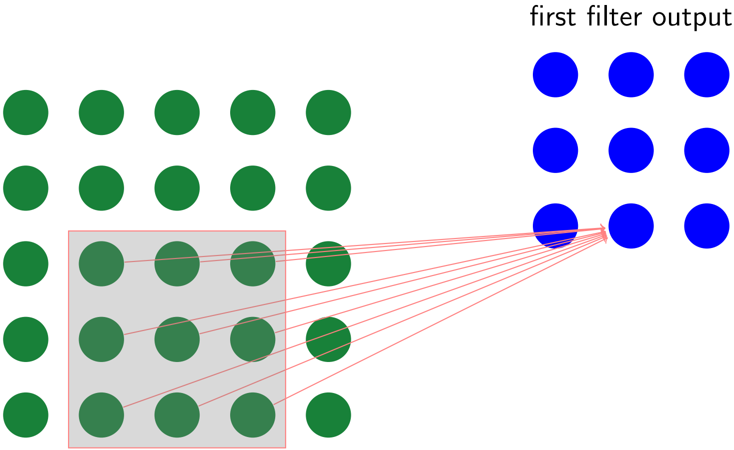
A convolution of 256 filters of kernel size 3x3 with stride 1



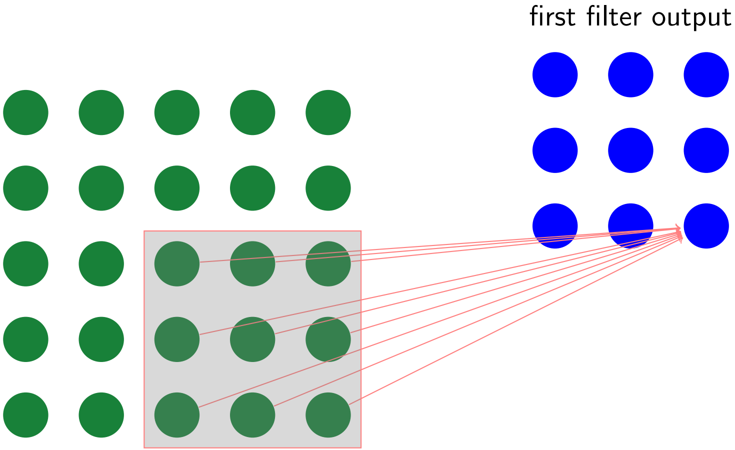
A convolution of 256 filters of kernel size 3x3 with stride 1



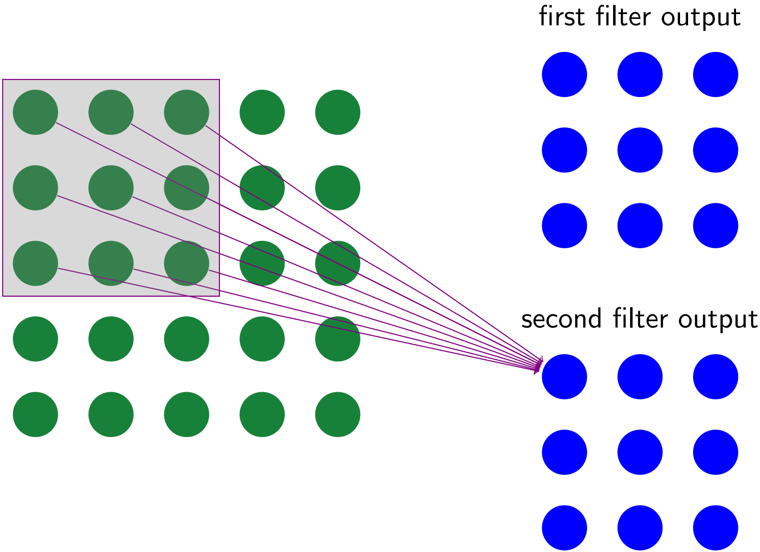
A convolution of 256 filters of kernel size 3x3 with stride 1



A convolution of 256 filters of kernel size 3x3 with stride 1



A convolution of 256 filters of kernel size 3x3 with stride 1



Neural network architecture

- Input: Go position.
- Neural network
  - 79 convolutional layers
  - Batch normalizations
  - Residual connections
- Output
  - policy: 2-fully connected layer
  - evaluation: 3 fully-connected layer

Self-learning system

1. Start with random prior policy and prior evaluation.
2. Self-play using Monte-Carlo Tree Search with 1600 simulations.
3. Create better examples position-policy and position-evaluation.
4. Generalize and compress by training the neural network architecture.
5. Better policy and evaluation becomes prior policy and evaluation.
6. Loop to 2.

Performance and legacy

- Beat the best human player at go:
  Go is easy.
- Rediscover human go knowledge through self-learning:
  With suitable algorithms, complex strategies can be build by accumulating small improvements.