

Foundations of Artificial Intelligence

46. AlphaGo and Outlook

Thomas Keller

Universität Basel

May 30, 2016

Foundations of Artificial Intelligence

May 30, 2016 — 46. AlphaGo and Outlook

46.1 Introduction

46.2 MCTS in AlphaGo

46.3 Neural Networks

46.4 Summary

Board Games: Overview

chapter overview:

- ▶ 41. Introduction and State of the Art
- ▶ 42. Minimax Search and Evaluation Functions
- ▶ 43. Alpha-Beta Search
- ▶ 44. Monte-Carlo Tree Search: Introduction
- ▶ 45. Monte-Carlo Tree Search: Advanced Topics
- ▶ 46. AlphaGo and Outlook

46.1 Introduction

Go

- ▶ more than 2500 years old
- ▶ considered the hardest classical board game
- ▶ played on 19×19 board
- ▶ simple rules:
 - ▶ players alternately place a stone
 - ▶ surrounded stones are removed
 - ▶ player with more territory wins



Monte-Carlo Methods in Go: Brief History

- ▶ 1993: Brüggemann applies **Monte-Carlo methods** to Go
- ▶ 2006: **MoGo** of Gelly et al. is the first Go AI based on **Monte-Carlo Tree Search**
- ▶ 2008: Coulom's **CrazyStone** player beats 4 dan professional Kaori Aobai with handicap of 8 stones
- ▶ 2012: Ojima's **Zen** player beats 9 dan professional Takemiya Masaki with handicap of 4 stones
- ▶ 2015: **AlphaGo** beats the European Go champion Fan Hui, a 2 dan professional, 5-0
- ▶ 2016: AlphaGo beats one of the worlds best Go players, 9 dan professional Lee Sedol, with 4-1

46.2 MCTS in AlphaGo

MCTS in AlphaGo: Overview

- ▶ based on Monte-Carlo Tree Search
- ▶ search nodes annotated with:
 - ▶ utility estimate $\hat{Q}(n)$
 - ▶ visit counter $N(n)$
 - ▶ a constant **prior probability** $p_0(n)$ from **SL policy network**

MCTS in AlphaGo: Tree Policy

- ▶ selects successor n that maximizes $\hat{Q}(n) + \hat{U}(n)$
- ▶ computes bonus term $\hat{U}(n)$ for each node **proportionally to prior and number of visits** as $\hat{U}(n) \propto \frac{p_0(n)}{1+N(n)}$

⇒ computes an **upper confidence bound** with a bonus term that resembles **Boltzmann exploration**

MCTS in AlphaGo: Iteration Evaluation

- ▶ Utility of an iteration is made up of two parts:
 - ▶ the result of a simulation $u_{\text{sim}}(n)$ with a default policy from a **rollout policy network**
 - ▶ a heuristic value $h(n)$ from a **value network**
- ▶ combined via a **mixing parameter** $\lambda \in [0, 1]$ by setting the utility of the iteration to

$$\lambda \cdot u_{\text{sim}}(n) + (1 - \lambda) \cdot h(n)$$

- ▶ mixing parameter in final version is $\lambda = 0.5$, which indicates that **both parts are important** for the playing strength

MCTS in AlphaGo: Other

expansion phase:

- ▶ ignores restriction that unvisited successors must be created
- ▶ stores annotations **in the parent node**

final recommendation:

- ▶ return successor that has been **visited most often** rather than the one with highest utility estimate

46.3 Neural Networks

Neural Networks

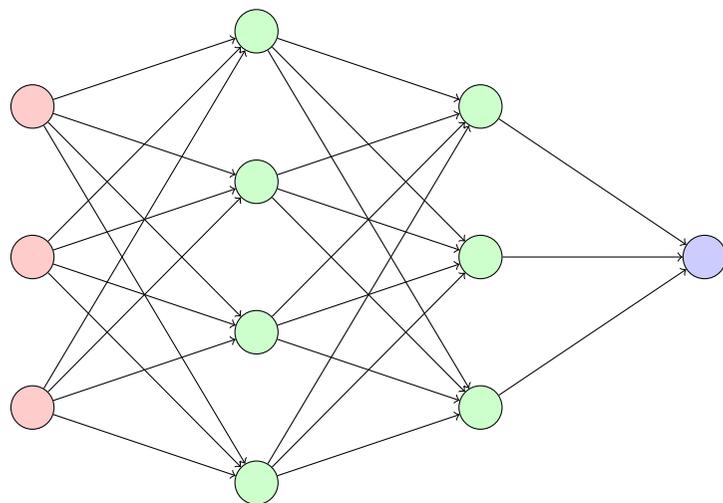
AlphaGo computes four neural networks:

- ▶ rollout policy network
 - ⇒ for **initialization**
- ▶ supervised learning (SL) policy network
 - ⇒ for **prior probabilities**
- ▶ reinforcement learning (RL) policy network (intermediate step only)
- ▶ value network
 - ⇒ for **initialization**

Neural Network

- ▶ used to approximate an unknown function
- ▶ layered graph of three types of nodes:
 - ▶ input nodes
 - ▶ hidden nodes
 - ▶ output nodes
- ▶ iteratively learns function by adapting **weights** of connections between nodes

Neural Networks: Example



input layer 1st hidden layer 2nd hidden layer output layer

SL Policy Network: Architecture

input nodes:

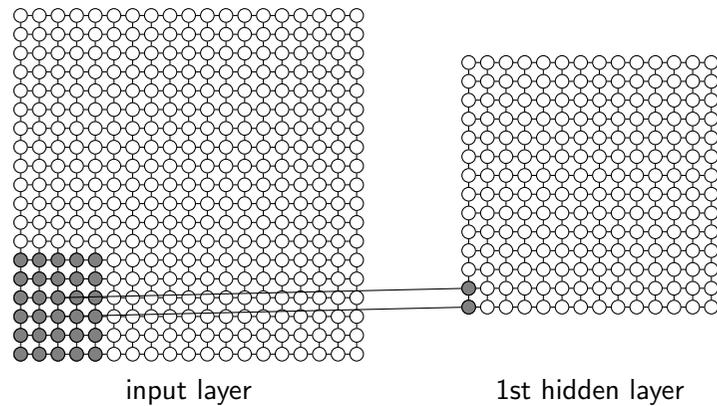
- ▶ the current **position**
- ▶ **move history**
- ▶ additional **features** (e.g., number of captured stones)

hidden layer:

- ▶ several **convolutional layers**:
 - ▶ **combine local information**
 - ▶ allow **less connections** between layers
 - ▶ weights are shared between connections of the same type
- ▶ final **linear softmax** layer
 - ▶ converts weights to **probabilities**

output nodes: a **probability distribution** over all legal moves

SL Policy Network: Convolutional Layers



SL Policy Network

- ▶ uses 30 million positions from strong human players on KGS
- ▶ uses **supervised learning**: the network learns to match given input to **given** output (i.e., the given position to the selected move)
- ▶ most **“human-like”** part of AlphaGo: aims to **replicate human choices**, not to win
- ▶ prediction accuracy: 57 %
- ▶ 3 ms per query

well-informed results with variance \Rightarrow good for **priors**

Rollout Policy Network: Architecture

input nodes:

- ▶ only **small set of features** from small window around own and opponent's previous move
- ▶ does not look at the entire 19×19 board

hidden layer: a single **linear softmax** layer

output nodes: a **probability distribution** over all legal moves

Rollout Policy Network

- ▶ uses supervised learning with the same data as the SL policy network
- ▶ lower prediction accuracy: 24.2 %
- ▶ but allows fast queries: just $2 \mu\text{s}$ (more than 1000 times faster than SL policy network)

reasonably informed yet cheap to compute

\Rightarrow well-suited as **default policy**

Value Network: RL Policy Network

first create sequence of RL policy networks with **reinforcement learning**

- ▶ **initialize** first RL policy network to SL policy network
- ▶ in each iteration, **pick a former RL policy network** uniformly at random \Rightarrow prevents overfitting to the current policy
- ▶ play with the current network against the picked one:
 - ▶ **compute the probability distribution** over all legal moves for the current state
 - ▶ **sample** a move according to the probabilities
 - ▶ **play** that move
 - ▶ repeat alternately until a final position is reached
- ▶ create new RL policy network by **updating weights** in the direction that maximizes expected outcome

Value Network: Architecture

then transform RL policy network to value network

input nodes: same as in SL and RL policy network

hidden layer: similar to RL policy network

output node: **utility estimate** that approximates Q^*
 \Rightarrow the value network computes a heuristic

Value Network

- ▶ using state-outcome pairs from KGS Server leads to **overfitting**
- ▶ using too many positions from same game introduce bias (not enough data to use only a few)
- ▶ create a **new dataset** with 30 million self-plays of standalone RL policy network and itself
- ▶ each game only introduces **a single state-outcome pair** into the new dataset
- ▶ only **minimal overfitting**
- ▶ slightly worse **win percentage** than using RL Policy Network as default policy
- ▶ but **15000 times faster**

very well informed and reasonably fast

\Rightarrow good **heuristic**

46.4 Summary

Summary: This Chapter

- ▶ AlphaGo combines Monte-Carlo Tree Search with **neural networks**
- ▶ uses **priors** to guide selection strategy
- ▶ priors are learned from **human players**
- ▶ learns a reasonably informed yet **cheap to compute** default policy
- ▶ iterations are additionally evaluated with **utility estimates**, which are learned from humans and intensive self-play

Summary: Board Games

- ▶ board games are a topic that has traditionally been important in AI research
- ▶ in most board games, computers are able to beat human experts
- ▶ **optimal strategy** can be computed with minimax
- ▶ $\alpha - \beta$ pruning often **speeds up minimax** significantly
- ▶ introduction of Monte-Carlo Tree Search led to **tremendous progress** in many games
- ▶ combination with **neural networks** allowed to beat a human professional in Go