Introduction to Artificial Intelligence

Fall 2018

Prof. Gilles Louppe
g.louppe@uliege.be

LIÈGE université

Logistics

This course is given by:

- Theory: Prof. Gilles Louppe [g.louppe@uliege.be]
- Exercises: Antoine Wehenkel [antoine.wehenkel@uliege.be]
- Projects: Samy Aittahar [saittahar@uliege.be]

Feel free to contact any of us for help!



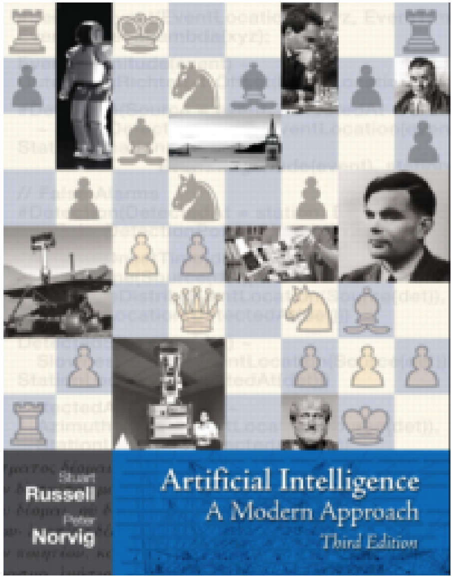
Slides

Slides

- are available at https://github.com/glouppe/info8006-introduction-to-ai.
- are available in HTML and in PDFs.
- will be posted online the day before the lesson.
- will be slightly different from previous years.

Some lessons are partially adapted from [CS188] Introduction to Artificial Intelligence, by Dan Klein and Pieter Abbeel from UC Berkeley.

Textbook



The core content of this course is based on the following textbook:

> *Stuart Russel, Peter Norvig. "Artificial Intelligence: A Modern Approach", Third Edition, Global Edition.*

This textbook is strongly recommended, although not required.

Philosophy

Thorough and detailed

- Understand the landscape of artificial intelligence.
- Be able to write from scratch, debug and run (some) AI algorithms.

Well established algorithms and state-of-the-art

- Well-established algorithms for building intelligent agents.
- Introduction to materials new from research ($\leq$ 5 years old).
- Understand some of the open questions and challenges in the field.

Practical

- Fun and challenging course project.

Lectures

- Theoretical lectures
- Exercise sessions

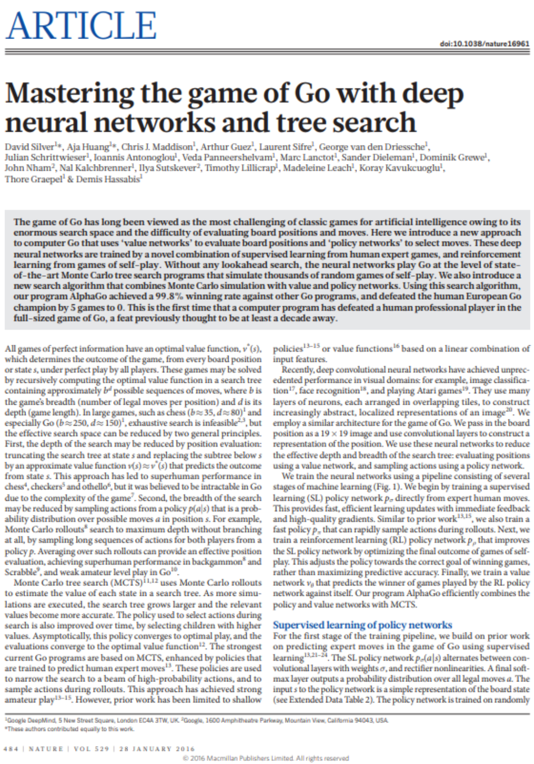
Outline

- Lecture 1: Foundations
- Lecture 2: Solving problems by searching
- Lecture 3: Constraint satisfaction problems
- Lecture 4: Adversarial search
- Lecture 5: Representing uncertain knowledge
- Lecture 6: Inference in Bayesian networks
- Lecture 7: Reasoning over time
- Lecture 8: Making decisions
- Lecture 9: Learning
- Lecture 10: Communication
- Lecture 11: Artificial General Intelligence and beyond

Projects

Reading assignment

Read, summarize and criticize a major scientific paper in Artificial Intelligence. (Paper to be announced later.)



ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35$, $d \approx 80$)[1] and especially Go ($b \approx 250$, $d \approx 150$)[1], exhaustive search is infeasible[2,3], but the effective search space can be reduced by two general principles. First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s. This approach has led to superhuman performance in chess[4], checkers[5] and othello[6], but it was believed to be intractable in Go due to the complexity of the game[7]. Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s. For example, Monte Carlo rollouts[8] search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon[8] and Scrabble[9], and weak amateur level play in Go[10].

Monte Carlo tree search (MCTS)[11,12] uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate. The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function[12]. The strongest current Go programs are based on MCTS, enhanced by policies that are trained to predict human expert moves[13]. These policies are used to narrow the search to a beam of high-probability actions, and to sample actions during rollouts. This approach has achieved strong amateur play[13–15]. However, prior work has been limited to shallow policies[13–15] or value functions[16] based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification[17], face recognition[18], and playing Atari games[19]. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image[20]. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a representation of the position. We use these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network.

We train the neural networks using a pipeline consisting of several stages of machine learning (Fig. 1). We begin by training a supervised learning (SL) policy network p_σ directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high-quality gradients. Similar to prior work[13,15], we also train a fast policy p_π that can rapidly sample actions during rollouts. Next, we train a reinforcement learning (RL) policy network p_ρ that improves the SL policy network by optimizing the final outcome of games of self-play. This adjusts the policy towards the correct goal of winning games, rather than maximizing predictive accuracy. Finally, we train a value network v_θ that predicts the winner of games played by the RL policy network against itself. Our program AlphaGo efficiently combines the policy and value networks with MCTS.

Supervised learning of policy networks

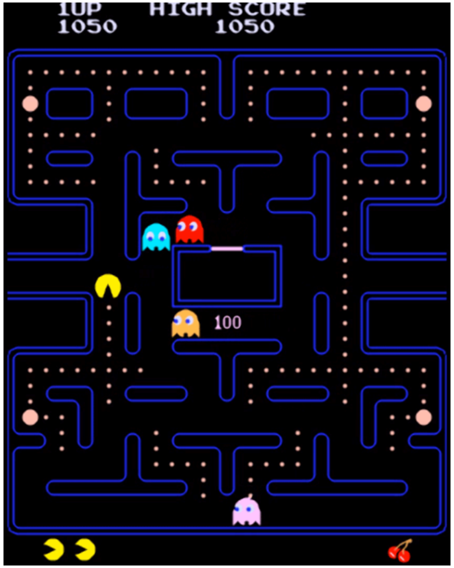
For the first stage of the training pipeline, we build on prior work on predicting expert moves in the game of Go using supervised learning[13,21–24]. The SL policy network $p_\sigma(a|s)$ alternates between convolutional layers with weights σ, and rectifier nonlinearities. A final softmax layer outputs a probability distribution over all legal moves a. The input s to the policy network is a simple representation of the board state (see Extended Data Table 2). The policy network is trained on randomly

[1]Google DeepMind, 5 New Street Square, London EC4A 3TW, UK. [2]Google, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA.
*These authors contributed equally to this work.

Programming project

Implement an intelligent agent for playing Pacman. The project will be divided into three parts, with increasing levels of difficulty:

- Eat as much dots as possible
- Eat as much dots as possible, while not getting killed by ghosts (deterministic)
- Eat as much dots as possible, while not getting killed by ghosts (stochastic)



Evaluation

- Oral exam (60%)
- Reading assignment (10%)
- Programming project (30%)

Projects are mandatory for presenting the exam.

Going further

This course is meant as an introduction to many other courses available at ULiège, including:

- ELEN0062: Introduction to Machine Learning
- INFO8004: Advanced Machine Learning
- INFOXXXX: Deep Learning (Spring 2019)
- INFO8003: Optimal decision making for complex problems
- INFO0948: Introduction to Intelligent Robotics
- INFO0049: Knowledge representation
- ELEN0016: Computer vision
- DROI8031: Introduction to the law of robots

Let's start!