

# CSC321 Lecture 23: Go

Roger Grosse

# Final Exam

- Friday, April 20, 9am-noon
  - Last names A–Y: Clara Benson Building (BN) 2N
  - Last names Z: Clara Benson Building (BN) 2S
- Covers all lectures, tutorials, homeworks, and programming assignments
  - 1/3 from the first half, 2/3 from the second half
  - If there's a question on Lectures 22 or 23, it will be easy
- Emphasis on concepts covered in multiple of the above
- Similar in format and difficulty to the midterm, but about 3x longer
- Practice exams are posted

# Overview

- Most of the problem domains we've discussed so far were natural application areas for deep learning (e.g. vision, language)
  - We know they can be done on a neural architecture (i.e. the human brain)
  - The predictions are inherently ambiguous, so we need to find statistical structure
- Board games are a classic AI domain which relied heavily on sophisticated search techniques with a little bit of machine learning
  - Full observations, deterministic environment — why would we need **uncertainty**?
- This lecture is about AlphaGo, DeepMind's Go playing system which took the world by storm in 2016 by defeating the human Go champion Lee Sedol
- Combines ideas from our last two lectures (policy gradient and value function learning)

# Overview

Some milestones in computer game playing:

- 1949 — Claude Shannon proposes the idea of game tree search, explaining how games could be solved algorithmically in principle
- 1951 — Alan Turing writes a chess program that he executes by hand
- 1956 — Arthur Samuel writes a program that plays checkers better than he does
- 1968 — An algorithm defeats human novices at Go  
...silence...
- 1992 — TD-Gammon plays backgammon competitively with the best human players
- 1996 — Chinook wins the US National Checkers Championship
- 1997 — DeepBlue defeats world chess champion Garry Kasparov

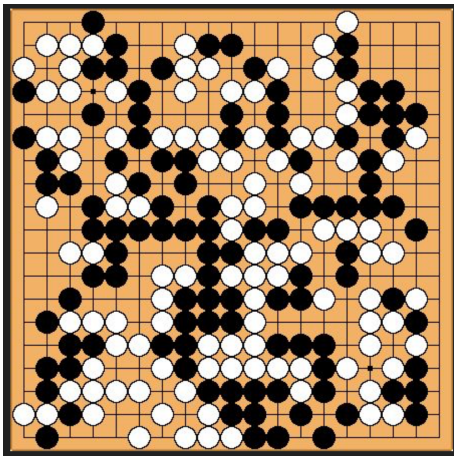
After chess, Go was humanity's last stand

- Played on a  $19 \times 19$  board
- Two players, black and white, each place one stone per turn
- Capture opponent's stones by surrounding them



# Go

- Goal is to control as much territory as possible:

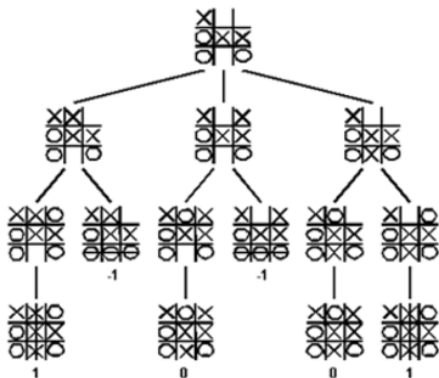


What makes Go so challenging:

- Hundreds of legal moves from any position, many of which are plausible
- Games can last hundreds of moves
- Unlike Chess, **endgames are too complicated** to solve exactly (endgames had been a major strength of computer players for games like Chess)
- Heavily dependent on pattern recognition

# Game Trees

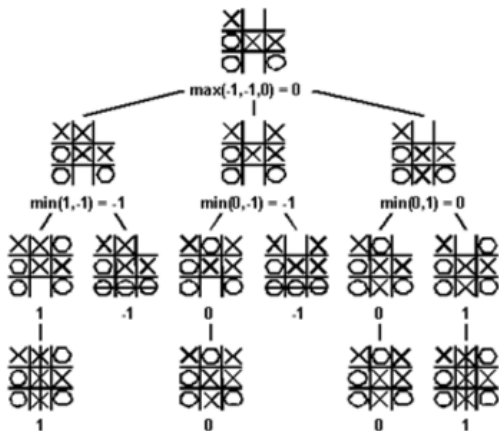
- Each node corresponds to a legal state of the game.
- The children of a node correspond to possible actions taken by a player.
- Leaf nodes are ones where we can compute the value since a win/draw condition was met





# Game Trees

- To label the internal nodes, take the max over the children if it's Player 1's turn, min over the children if it's Player 2's turn

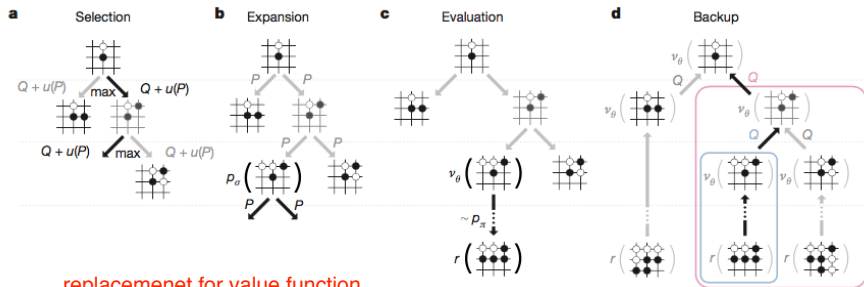


# Game Trees

- As Claude Shannon pointed out in 1949, for games with finite numbers of states, you can solve them in principle by drawing out the whole game tree.
- Ways to deal with the exponential blowup
  - Search to some fixed depth, and then estimate the value using an **evaluation function**
  - Prioritize exploring the most promising actions for each player (according to the evaluation function)
- Having a good evaluation function is key to good performance
  - Traditionally, this was the main application of machine learning to game playing
  - For programs like Deep Blue, the evaluation function would be a learned linear function of carefully hand-designed features

# Monte Carlo Tree Search

- In 2006, computer Go was revolutionized by a technique called Monte Carlo Tree Search.



Silver et al., 2016

- Estimate the value of a position by simulating lots of **rollouts**, i.e. games played randomly using a quick-and-dirty policy
- Keep track of number of wins and losses for each node in the tree
- Key question: how to select which parts of the tree to evaluate?

# Monte Carlo Tree Search

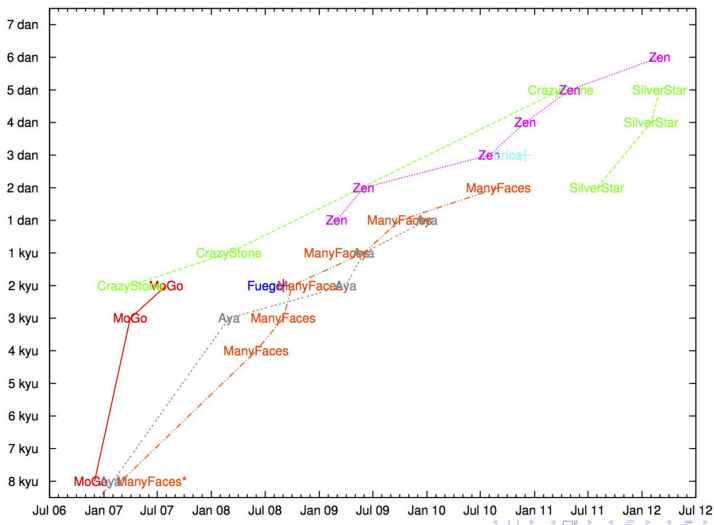
- The selection step determines which part of the game tree to spend computational resources on simulating.
- This is an instance of the **exploration-exploitation tradeoff** from last lecture
  - Want to focus on good actions for the current player
  - But want to explore parts of the tree we're still uncertain about
- **Uniform Confidence Bound (UCB)** is a common heuristic; choose the **node** which has the largest frequentist upper confidence bound on its value:

$$\mu_i + \sqrt{\frac{2 \log N}{N_i}}$$

- $\mu_i$  = fraction of wins for action  $i$ ,  $N_i$  = number of times we've tried action  $i$ ,  $N$  = total times we've visited this node

# Monte Carlo Tree Search

Improvement of computer Go since MCTS (plot is within the amateur range)



Now for DeepMind's computer Go player, AlphaGo...

# Predicting Expert Moves

- Can a computer play Go without any search?
- Ilya Sutskever's argument: experts players can identify a set of good moves in half a second
  - This is only enough time for information to propagate forward through the visual system — not enough time for complex reasoning
  - Therefore, it ought to be possible for a conv net to identify good moves

# Predicting Expert Moves

- Can a computer play Go without any search?
- Ilya Sutskever's argument: experts players can identify a set of good moves in half a second
  - This is only enough time for information to propagate forward through the visual system — not enough time for complex reasoning
  - Therefore, it ought to be possible for a conv net to identify good moves
- **Input:** a  $19 \times 19$  ternary (black/white/empty) image — about half the size of MNIST!
- **Prediction:** a distribution over all (legal) next moves
- **Training data:** KGS Go Server, consisting of 160,000 games and 29 million board/next-move pairs
- **Architecture:** fairly generic conv net
- When playing for real, choose the highest-probability move rather than sampling from the distribution



# Predicting Expert Moves

- Can a computer play Go without any search?
- Ilya Sutskever's argument: experts players can identify a set of good moves in half a second
  - This is only enough time for information to propagate forward through the visual system — not enough time for complex reasoning
  - Therefore, it ought to be possible for a conv net to identify good moves
- **Input:** a  $19 \times 19$  ternary (black/white/empty) image — about half the size of MNIST!
- **Prediction:** a distribution over all (legal) next moves
- **Training data:** KGS Go Server, consisting of 160,000 games and 29 million board/next-move pairs
- **Architecture:** fairly generic conv net
- When playing for real, choose the highest-probability move rather than sampling from the distribution
- This network, which just predicted expert moves, could beat a fairly strong program called GnuGo 97% of the time.
  - This was amazing — basically all strong game players had been based on some sort of search over the game tree

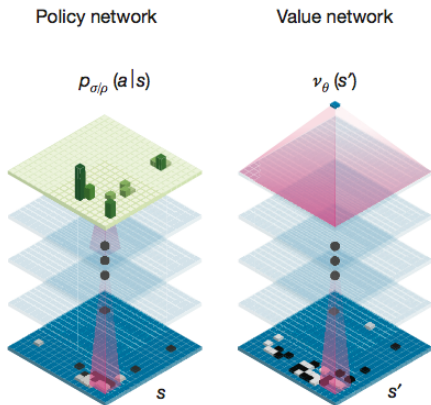
# Self-Play and REINFORCE

- The problem from training with expert data: there are only 160,000 games in the database. What if we overfit?
- There is effectively infinite data from **self-play**
  - Have the network repeatedly play against itself as its opponent
  - For stability, it should also play against older versions of itself
- Start with the **policy** which samples from the predictive distribution over expert moves
  - The network which computes the policy is called the **policy network**
- **REINFORCE** algorithm: update the policy to maximize the expected reward  $r$  at the end of the game (in this case,  $r = +1$  for win,  $-1$  for loss)
- If  $\theta$  denotes the parameters of the policy network,  $a_t$  is the action at time  $t$ , and  $s_t$  is the state of the board, and  $z$  the **rollout** of the rest of the game using the current policy

$$R = \mathbb{E}_{a_t \sim p_{\theta}(a_t | s_t)}[\mathbb{E}[r(z) | s_t, a_t]]$$

# Policy and Value Networks

- We just saw the policy network. But AlphaGo also has another network called a **value network**.
- This network tries to predict, for a given position, **which player has the advantage**.
- This is just a vanilla conv net trained with least-squares regression.
- Data comes from the board positions and outcomes encountered during self-play.



Silver et al., 2016

# Policy and Value Networks

- AlphaGo combined the policy and value networks with Monte Carlo Tree Search
- Policy network used to simulate rollouts
- Value network used to evaluate leaf positions

# AlphaGo Timeline

- **Summer 2014** — start of the project (internship project for UofT grad student Chris Maddison)
- **October 2015** — AlphaGo defeats European champion
  - First time a computer Go player defeated a human professional without handicap — previously believed to be a decade away
- **January 2016** — publication of Nature article “Mastering the game of Go with deep neural networks and tree search”
- **March 2016** — AlphaGo defeats gradmaster Lee Sedol
- **October 2017** — AlphaGo Zero far surpasses the original AlphaGo without training on any human data
- **Decemter 2017** — it beats the best chess programs too, for good measure

# AlphaGo

- Most of the Go world expected AlphaGo to lose 5-0 (even after it had beaten the European champion)
- It won the match 4-1
- Some of its moves seemed bizarre to human experts, but turned out to be really good
- Its one loss occurred when Lee Sedol played a move unlike anything in the training data

## Further reading:

- Silver et al., 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* <http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html>
- Scientific American: <https://www.scientificamerican.com/article/how-the-computer-beat-the-go-master/>
- Talk by the DeepMind CEO:  
[https://www.youtube.com/watch?v=a iwQsa\\_7ZIQ&list=PLqYmG7hTraZCGIymT8wVVIXLWkKPNBoFN&index=8](https://www.youtube.com/watch?v=a iwQsa_7ZIQ&list=PLqYmG7hTraZCGIymT8wVVIXLWkKPNBoFN&index=8)