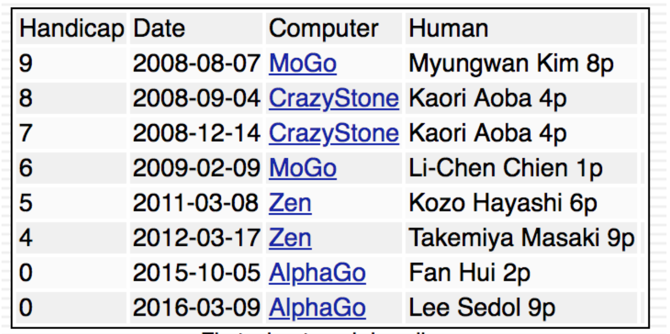
CMPUT 396: AlphaGo

The race to build a superhuman Go-bot is over! AlphaGo won, about 10 years earlier than people though.

Even before AlphaGo there was a rich computer Go ecosystem. However, AlphaGo was the first to win against a human player with no handicap.



Handicap	Date	Computer	Human
9	2008-08-07	MoGo	Myungwan Kim 8p
8	2008-09-04	CrazyStone	Kaori Aoba 4p
7	2008-12-14	CrazyStone	Kaori Aoba 4p
6	2009-02-09	MoGo	Li-Chen Chien 1p
5	2011-03-08	Zen	Kozo Hayashi 6p
4	2012-03-17	Zen	Takemiya Masaki 9p
0	2015-10-05	AlphaGo	Fan Hui 2p
0	2016-03-09	AlphaGo	Lee Sedol 9p

First win at each handicap

Post-AlphaGo, computer Go research will continue mostly in the commercial world as people try to replicate AlphaGo with less hardware (for the competition, AlphaGo played on reasonable hardware but the training cost millions of dollars).

Brief History

2006: Monte Carlo Tree Search first introduced by Remi Coulom by applying the Monte Carlo method (which dates back to the 1940s and is basically the idea of using randomness for deterministic problems that are difficult to solve otherwise) to game-tree search.

Pre-AlphaGo: the top pros could beat the top programs with a 3 stone handicap

2015: AlphaGo vs. Fan Hui (2p), Fan Hui lost 5-0 in the formal matches and 3-2 in the blitz version

2016: AlphaGo vs. Lee Sedol (9p), Lee Sedol lost 4-1

2017: AlphaGo vs. Ke Jie (9p, #1 in the world in 2017), Ke Jie lost 3-0

2017: AlphaGo retires

Computer Olympiad

Since 1989, the Computer Olympiad has been a main venue for computer-computer board game competition (so people who have built programs to play a certain game can compete with other programs). UAlberta groups have competed in Chess, Checkers, Othello, Go, Hex, Amazons, and other games.

2006: this is the first time MCTS with Crazystone (Remi Coulom’s program) taking the gold in 9x9 Go (at the same competition, GNU Go won the 19x19 Go gold)

2009: Fuego (Martin Mueller at U of A) won 9x9 Go gold

2008 to present: MoHex (also UAlberta) has won gold in 11x11 Hex

Martin Mueller wrote Fuego to be game independent, so MoHex is built on those game independent parts. This means that every time Fuego gets stronger, MoHex also gets stronger.

Neural Nets and Image Classification

A **neural net** is a machine-constructed function. Neural nets are interconnected groups of nodes, inspired by the human brain and nervous system. The neural net itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. In this way, neural nets "learn" to perform tasks without being programmed with any task-specific rules.

(https://en.wikipedia.org/wiki/Artificial_neural_network).

All we really need to know about neural nets: it's a kind of function that is machine constructed, originally modelled to resemble the architecture of how we think the human brain works. It takes in some inputs, and gives outputs.

Supervised learning is learning from (usually human-supplied) training data. "I give you problems with answers to help you learn."

Example:

What is this?

It's a kitten!

How do we know this is a kitten? Someone told us that this is a kitten.

How can we teach a computer to know that this is a kitten too?

Image classification is the process of learning how to recognize pixel images of things (e.g. kittens).

Research in image classification generally works like this:

- Humans gather large collections of human-labelled images
  - E.g. say that we are given 1000 pictures of birds and we want to learn to recognize the species. A human will label all the images with the species of the bird.

- They design an algorithm that distinguishes among the different kinds of images
    - E.g. they design an algorithm that can distinguish between the different species of birds in the images
- They train their algorithm, say a neural net, by taking something like 90% of the images.
    - E.g. train the algorithm with 900 of the 1000 human-labelled images that we have
- They use the remaining 10% of the images to test the accuracy of their algorithm
    - E.g. use the remaining 100 pictures of birds, ask the algorithm which species each one is, compare to the human label to know if we are right (how accurate our algorithm is)

In 2012, there was a breakthrough in image classification by Krizhevsky, Sutskever, and Hinton (from the U of T). They trained a large, deep convolutional neural net for image classification and showed that a neural net (if deep enough and trained long enough) can be better than humans at image classification.

How can we use the ideas of image classification in Go? We can use it to pick moves.

AlphaGo Evolution

DeepMind was founded in 2010 by Demis Hassabis and two others. DeepMind is an unusual company because the goal was just to do AI. Demis Hassabis figured that if they specialized in AI they would be of value to big companies like Google.

David Silver was there from the beginning (worked as a consultant even when he was still working on his PhD). After he finished his PhD in 2013, he started to work there full-time as the head of reinforcement learning.

In 2014, Google bought DeepMind, reportedly for over $500 million.

DeepMind realized that they could use ideas from image classification to look at positions in Go, so they embedded the ideas of image classification into MCTS (December 2014). So instead of trying to figure out which species of bird the image was, they could ask "given this current board position, what do humans usually do in this situation?" to predict moves in Go.

The University of Edinburgh independently did the same thing (Storkey-Clark. February 2015) by creating a Go move-policy network. They have a web applet with a neural net that they trained so that when you give it a board position, it tells you what move humans would make. For example, humans usually open in the corners at 3-4 or 4-4. Their neural net trained from records of games from pro Go players.
(https://chrisc36.github.io/deep-go/)

These trained neural nets, even with no search, can beat a lot of human players (at least average players). However, it's probably not strong enough to beat really good players.

Let's say that we're going to use one of these neural nets in tree search, say MCTS. Performing selection as usual is probably fine and when we get to expansion we can ask the neural net to pick a child for us. What's the problem?

It's not really a problem that it's deterministic, but it *is* a problem that it is too slow (about 1/3rd of a second on the applet). Ideally, we would like this step to be 1/1000th of a second (or less).

In DeepMind's 2014 paper, they cited a lot of possible concerns like these, which suggested that perhaps computers beating humans in Go was still 10 years away.

However, in October 2015, AlphaGo played Fan Hui in a secret match and won 5-0. In January of 2016, this was revealed to the public through a paper published through Nature. They used Fan Hui in these initial matches because he was the European champion, a professional Go player, and he was already in Europe (he lived in France) so it was convenient.

The version of AlphaGo that beat Fan Hui was a fairly primitive version of the program. The basic idea was to go crazy with neural nets, use image classification on thousands of moves, and use supervised learning to figure out what the most common move would be. Then, they used reinforcement learning so when they made a move, they would get a reward of 1 when they won and a reward of -1 when they lost. The reinforcement learning algorithm gets the reward ad adjusts the neural net appropriately.

KGS is an internet Go server where AlphaGo got a lot of its records for the image classification part. One downside is that a lot of the moves played may not have been the best moves (humans aren't perfect and probably many of the players weren't top professionals). However, this is okay. We're using it for move selection so it's probably valuable to us to look at lots of moves, not just the moves that we would think the "best' professional move would be.

AlphaGo used shallow neural nets so it didn't take quite as long (in contrast, the "slow" Storkey-Clark neural net uses a super deep neural net).

Another great idea was that they transformed the "pick the best move" policy to be a "give a winning probability of this move" policy.

Policy net: for picking moves
Value net: knows for position X what the winning probability is

AlphaGo MCTS:

1) Select a leaf: call the shallow neural net, ask what is the best move here?
2) Expand
3) Simulate
4) Backpropagate

At the beginning, they would do simulations half of the time and call the value net the other half of the time so it would be faster. Later on, however, they got fast enough that they were able to throw this out.

However, they still had problems with latency, the calls to the neural net were taking too long. The algorithm couldn't wait.

So they modified their MCTS:

1) Select
2) Expand
3) Simulate
4) Backpropagate: start this *before* you get the result, then keep going. When you get the answer back, you might need to adjust a bit. (This is super complicated, but they figured it out.)

An interesting thing about AlphaGo is that they started with human data, but ended up with something superhuman. One concern was that the value nets started with human data and they didn't want this to be a limiting factor. What if the humans missed something? We don't want AlphaGo to miss something that the humans also missed.

After DeepBlue won against Kasparov, IBM dismantled it. The DeepMind team didn't want the same thing to happen to AlphaGo so they have put a lot of resources on AlphaGo online to not leave it behind (in the webnotes).

After Fan Hui's game against AlphaGo in October 2015, he started working with the AlphaGo team. He would play it 9 hours a day, then talk to Aja to let him know about any weaknesses he discovered. The team would take these and decide if they were a big deal and if they could be fixed.

Fan Hui discovered that he was able to beat AlphaGo with AlphaGo's help. He knew the weaknesses so he could get it into a certain position, and then let AlphaGo take over for him and win the game.

Hayward thought it might have been better to prolong it, let Lee Sedol play AlphaGo v13 first (the version that beat Fan Hui), but when he asked someone at DeepMind about it they said that their hand was forced because they had competition from people like Facebook to be the first Go program to beat a human professional player.

In Game 1, move 7 by Lee Sedol was interesting because it is not a typical move (Q8). He has never spoken about this, but it seems like he was playing a calculated risk. If he's playing the same version Fan Hui did, he thinks he could outsmart it. Lee Sedol usually goes to tournaments with two other strong players (they're called "seconds"). Probably someone from his team read the Nature paper, so they know that AlphaGo started by looking at the data from professional Go games. This could be why he played this move, which is not entirely "pro-like".

After AlphaGo beat Lee Sedol, some people said that Lee Sedol was only ranked 6th in the world, there's no way AlphaGo could beat the top pro…

So in March 2017, AlphaGo played Ke Jie, the top-ranked player at that time, and it was a slaughter. Ke Jie lost 3-0.

Then came AlphaGo Zero… Someone asked David Silver at the AlphaGo press conference during the Lee Sedol match if it would be better to start with a blank slate, rather than with the human data that they started with. At the time (March 2016) he said he didn't know.

In October 2017, DeepMind released a paper on AlphaGo Zero, which did just that. It removed the supervised learning part, their algorithm was "based solely on reinforcement learning without human data, guidance, or domain knowledge beyond game rules". AlphaGo Zero became its own teacher and achieved superhuman performance by *tabula rasa* (blank slate) reinforcement learning through self-play.

Then in December 2017 came AlphaZero, a more generalized approach that is not specific to Go like AlphaGo Zero. The AlphaZero algorithm can achieve superhuman performance from a blank slate in many different domains (within 24 hours achieved superhuman level of play in chess, shogi (Japanese chess), and Go). They played Stockfish, the world champion chess program, in a 100-game match, winning 28 times, drawing 72 times, and never losing.

There has been some criticism of this match because it may not have been a fair fight because of the differences in hardware between AlphaZero and Stockfish. But regardless, AlphaZero was able to become a very strong chess program.

Zugzwang: when you're in a position in chess where it's your turn to play but you're out of good moves (you almost don't want to move at all).

- AlphaZero was able to get Stockfish into this position

DeepMind claimed that AlphaZero took only 4 hours to learn chess. How accurate is this though? DeepMind spent three years on getting their neural net learners to the best configurations. It may have taken four hours, but there was a lot of other work that went into them getting to this point.

Remember that you can also stop the function at any point and use that version (the longer you train the better you will be, but you can take as much or as little time as you want to).

AlphaGo retired after playing against Ke Jie. Alpha going, going, gone. Why? It's too expensive to maintain. The people are the most expensive part, and those who worked on AlphaGo have since moved to other projects.

While all of this was happening…

There are many commercial Go programs that started using the same ideas as AlphaGo. One such program, Deep Zen, the top commercial Go program, played Cho Chikun, an old 9p player. Cho Chikun won the match 2-1.

Deep Zen used a single $15K computer requiring 1-2 Kw of power.

How often did the moves made in the Deep Zen Go vs. Cho Chikun match the prediction made by Storkey-Clark's DCNN? Quite a few actually! The branching factor of Go is huge in that there are a lot of legal moves, but the number of top moves is often very small.

A more in-depth look at some of the important games…

AlphaGo vs. Lee Sedol

Game 1: AG: white, LSD: black
AlphaGo wins

Move 7: Lee Sedol made a very unusual move for move 7, which was probably to try to throw off his opponent (spoiler: it didn't work).

AlphaGo learned on human game records, built a neural net that could predict what a human would do, then trained using reinforcement learning (if you won, then the moves you made along the way are good moves, and if you lost, they were bad moves). Then, AlphaGo used self-play to get better and better.

Lee Sedol's attempt to throw off AlphaGo didn't work (in fact it backfired, because in the end it wasn't a good move).

Remember that AlphaGo uses a policy net, which is basically a move picker, and a value net, which looks at the board position and can tell you the expected win rate. You can't use supervised learning to build the value net because it doesn't tell you the win rate for the move, it only gives you one path to the win. Building a policy net is a little easier, you can always get the next move for each move that you select.

At the time of the AlphaGo vs. Lee Sedol match people didn't know a lot about how AlphaGo was built outside of what was in the Nature paper. Some people thought that it might start with trees for certain openings, so playing an unlikely move early on might not have been covered by the trees with limited breadth at the top (this maybe explains why Lee Sedol played such an unusual move so early on in the game). As we know, this was not the case. AlphaGo is flexible and was able to adapt.

After game one, a lot of pros still thought that Lee Sedol would win the match because he took a risk in this game and played a suboptimal move that cost him in the end. However, after the first game, media coverage increased a lot and it got a lot of attention from regular people.

Game 2: LSD: white, AG: black
AlphaGo wins

At move 36, Lee Sedol went to take a smoke break. AlphaGo usually took 2-3 minutes to make a move so AlphaGo made move 37 and then the time on Lee Sedol's clock was going until he came back. Move 37, played by AlphaGo, was a very strange move. It was a shoulder hit in the middle of the edge of board that no human pro would ever play. When the move was first played people thought that AlphaGo made a mistake.

Hollywood couldn't write a better script. It was the worst time to take a smoke break.

Despite the huge branching factor of the game of Go, top-level Go is actually fairly narrow. Usually there are a couple obvious moves that people don't deviate from. However, move 27 went beyond that.

The conclusion from this game was that AlphaGo is beyond human. At this point, it was still unclear if AlphaGo would win the match or not, but it had played a game against a top human pro, made a move that no top human pro would make, and won. It went beyond the human game records that AlphaGo started learning on.

Game 3: AG: white, LSD: black
AlphaGo wins again, and wins the match with 3 wins

Lee Sedol is a talented Go player in that he can play many different styles of Go (the general split is passive vs. fighting style, but it's more complicated than that). In the match, he tried many different playing styles to take on AlphaGo. However, AlphaGo is also talented at playing with different styles.

Lee Sedol started the game aggressively with a high Chinese fuseki, a fighting stance that seeks to take the initiative in the game. At move 14, AlphaGo played a two-space jump above one of its other stones in the corner. Lee Sedol's reply was perhaps overlyaggressive, right below the stone AlphaGo had just placed (a move that is too extreme according to Go fundamentals).

Fan Hui described Lee Sedol's playing style in this match as a wolf ambushing his prey, concealing himself until the best opportunity presents itself. "Patience and keen senses are his weapons. The moment he detects his chance, he strikes swiftly and fatally." Fan Hui commented that Lee Sedol betrayed his style and showed his fangs too early in this game, starting with this move.

By move 28, AlphaGo's win rate was 59%, than 62% at move 32, 72% at move 48, and 74% after move 54 (to reach such a high number of quickly meant that the game was essentially decided).

There had been a rumour since the beginning of the match that AlphaGo could not handle ko correctly, especially since it had avoided a large ko near the end of game 2, seemingly preventing ko. In this game with black move 61, there was a dangerous possibility of ko so perhaps Lee Sedol was testing this.

At black move 77, it seemed that Lee Sedol admitted he could not win by normal means so he adopted his famous "zombie" playing style where a player who is doomed thrashes about in an attempt to catch its opponent off guard. It has been successful in the past, but in this game it was futile.

By white move 84, AlphaGo's win rate was 84%.

By the end of the game, there was finally a ko fight at the top of the board, shattering the notion that AlphaGo had any problems with ko and destroying Lee Sedol's last hope of survival.

Lee Sedol resigned at move 176.

At the postgame press conference, Lee Sedol stated that his new objective was to win at least one game, but after three losses in a row even this seemed unlikely.

Game 4: LSD: white, AG: black
Lee Sedol wins! A win for the humans!

Lee Sedol actually asked to play as white for this game and because the match was already won, the AlphaGo team obliged, AlphaGo actually thinks that white has a better chance to win with a komi of 7.5.

What we learned from the first three games:

- AlphaGo is, in a sense, crude. It doesn't play elegant Go all the time even if it wins, for example, it makes Lee Sedol reach across the table to make his move when human players would usually let the opponent play on their side.
- AlphaGo is very flexible, it was playing to win and to do that it played some moves that pros would not play. Pros often engage in the battle if they are invaded, but AlphaGo would often not engage, ignoring the invader and playing somewhere else instead.
- AlphaGo would sometimes throw away points. It doesn't care about the margin of winning, just that it wins.

By black move 77 of this game, AlphaGo's win rate was 70%. It looked like a lot of the middle might be black's so the only for white to win would be to not let black get most of the middle.

White move 78 was the game changer. It was an unpredictable move for Lee Sedol to play. He was asked after the match about why he made this move, he said that he looked at every possible move and he knew he would lose with every one of them. This was the last move he looked at, so he played it and hoped for the best. He said that he played what felt right. And it worked. This move cast AlphaGo into complete confusion.

AlphaGo started to make mistakes. Lee Sedol threw AlphaGo into such unexpected territory that it couldn't figure out what to do next. It was in trouble by move 85 and its win rate was in freefall.

Lee Sedol was in byo-yomi by move 103, but victory was on the horizon.

At move 180, AlphaGo resigned.

So what happened to AlphaGo after move 78 ("the divine move")? Move 78 was so unexpected that AlphaGo didn't have enough of a tree built up under it to make good moves. If it had had a more sophisticated time management strategy, it might have said "I don't have enough information, let's take 20 minutes of our time to build up our tree and then play." The time management strategy of AlphaGo at the time was fairly simple, only 2-3 minutes per move.

If Lee Sedol had played AlphaGo a few months later, AlphaGo wouldn't have made mistakes like this. If you go back and study what happened, move 78 wasn't actually a winning move, but it was enough to throw off AlphaGo.

Conclusions from this game? Lee Sedol is an amazing Go player, and AlphaGo is human. It makes mistakes too.

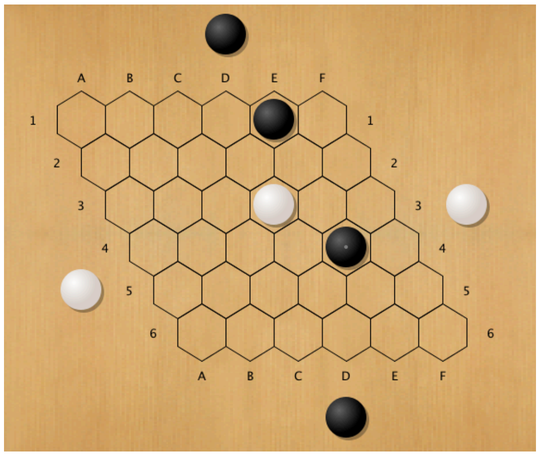
Game 5: AG: white, LSD: black
AlphaGo wins

Lee Sedol requested to be black in this match. Again, usually you wouldn't get to pick your colour but the match was already won so DeepMind said it was okay. Although AlphaGo prefers white because it has a better chance of winning, Lee Sedol had won as white so he wanted to prove he was capable of winning with Black as well.

However, he was in trouble by move 80 and lost the game.

Common MCTS Flaw: Optimistic Simulations

In games where the winner often has a narrow line of play, e.g. sequence with unique winning move(s), randomized simulation often has the wrong result.



For example, look at this Hex board. From this position with white to play, who wins?

MoHex with RAVE and 17 500 simulations in the tree says that move D5 has a win rate of 53%. But in fact, D5 loses. Actually all white moves lose. Black wins from this position.

Deep Zen Go vs. Cho Chikun

Remember earlier, Cho Chikun won this match 2-1 against the top commercial Go program.

Among commercial Go programs, Zen has strong playouts (simulations), but Zen's plyaouts missed Cho Chikun's best defensive sequences. In game 3, black (CCK) move 137, the win rate was estimated to be 64% (MCTS below 55% usually loses). Zen estimated that white, their colour, had a large group in the middle right. Zen's simulations were too optimistic, so the win rate started dropping.

By moves 156-158, Zen moves were small gains so Cho Chikun guessed that the computer was in trouble. After move 64, white's (Zen's) win rate dropped down to 48%.

Black move 167 cut into the white middle right group, making it a dead group. On move 168 the Zen operator Hideki Kato resigns.

More on the AlphaGo Nature Paper

Go is “easy” to solve because it is a game of perfect information, all you have to do is traverse the game tree. The problem is the size of the tree, with a depth of 150, breadth of 250 and a search space of 10^{360}.

Demis Hassabis always says that there are more ways for the game to playout than there are atoms in the universe, only about 10^{80}.

How can we make a strong player?

Reduce the search space!

- Truncate the tree at depth d, for subtree s, replace the true value function $v^*(s)$ with an estimate value function $v'(s)$

Reduce the breadth!

- Sample actions from policy $p(a|s)$, that is generate the inputs randomly from a the probability distribution over the domain, which is the possible moves a from s (this sampling is what MCTS does)

The biggest difference has been made in neural net breakthroughs. Namely, in image classification using deep convolutional neural nets, using many layers of neurons, each arranged in overlapping tiles to represent an image.

AlphaGo uses neural nets to evaluate positions using its *value net* and uses neural nets to estimate $p(a|s)$ using its *policy net*.

The AlphaGo network training pipeline started with ~30 000 000 game board positions from 160 000 KGS games to train the supervised learning policy net, resulting in a strong move picker, or policy net.

They also trained a fast, weaker policy net that was a bit shallower. Why have a shallow neural net when you can have a deep one?

- The deeper your neural net, the longer it takes to make the call. The time to get the output signal from the input signal is proportional to the number of layers in the neural net. Shallower also can give you a better breadth of search.

For AlphaGo they also trained a value net to predict to win rate at a given position. In the end, they combined the fast policy net and the value net with MCTS.

Input Features for AlphaGo Neural Nets

What do we need to give the neural net as input? In the most simple case, we could just give it the game board position. However, to do better we can also give it additional properties of the current game board position, “hints” to help it make better decisions.

These include:

a) Cell (black, white, or empty)
b) Liberties (for each group on the board)
c) Capture size (if the group was captured, how many stones would go off the board?)
d) Self-atari size
e) Liberties after move (how many liberties for groups *after* the move)
f) Ladder capture (if the ladder plays out in the obvious way, will they run into a wall? Ladder dies)
g) Ladder escape (if the ladder plays out in the obvious way, will they run into their own stone? Ladder survives!)
h) Legal and does not fill own eye

Policy Nets (Deep and Shallow) via Supervised Learning

Goal: use human data and supervised learning to create a policy net.

For an arbitrary state s and all possible actions a, the deep convolutional neural net will estimate function $f = \text{prob}(a\,|s)$.

- Input: board position s
- Output: from s, for each empty cell a, what is the probability (from the human data) that a human selects action a?

net	response time (10^{-6} s)	response accuracy
deep	3 000	.57
shallow	2	.24

We measure the results of the DCNN accuracy by withholding a fraction of the human data (for which we have the input and the output already) to test the results and see if it matches (if we give the input to the DCNN, the output should match what the expected output we have in the human data).

The human data AlphaGo used was 30 000 000 game positions from 160 000 KGS games.

Why did they add all these additional features to the input into the neural net? Because they got better results when they did, the function was more accurate.

Supervised learning policy net accuracy (% of guessing right):

- 0.557 (board input)
- 0.57 (board + extra features input) *this is actually a huge increase in strength

Strengthen policy net via reinforcement learning

A move that clearly leads to a loss is not likely to be played, so adding information about whether a move leads to a loss should make the DCNN more accurate.

Starting from a supervised learning policy net, train a new neural net using reinforcement learning: for each move, self-play game to the end, find the result

- Move that leads to win gets reward 1
- Move that leads to loss gets reward -1

The architecture of the reinforcement learning DCNN is similar to the supervised learning version, with just one extra final layer. The weights of the reinforcement learning DCNN are initialized to the final weights of the supervised learning DCNN. The final weights of the RL DCNN are then trained through reinforcement learning.

Compared to the SL policy net, the RL policy net has a win rate of 80%.

From RL Policy Net to RL Value Net

The policy net estimates the probability of making a move, so it ranks its children, but it cannot rank non-siblings. However, for a tree search like minimax or MCTS, we need to be able to compare non-siblings.

For each node, we need to be able to estimate the value. What is the probability that a given move *wins*?

For an arbitrary state s, DCNN estimates function $\mathrm{v}(s) = \mathrm{prob}(s \text{ is winning position})$.

- Input: board position s

The problem is that using 30 000 000 game positions from only 160 000 games leads to overfitting (*overfitting* is a modelling error which occurs when a function is too closely fit to a limited set of data points, it may therefore fail to fit additional data or predict future observations reliably). The value net learns to mimic move sequences in 160 000 games, but it players poorly in other games.

Solution: use 30 000 000 game positions from 30 000 000 self-play dataset games.

The RL value net and MCTS will be used at the leaf nodes of the search tree. One RL value net is comparable in accuracy to about t MCTS simulations using the fast RL policy net, but requires 15 000 times less computation time.

AlphaGo Search: Integrate Policy / Value Nets into MCTS

They didn't make just one version of AlphaGo, they made 1000s of versions using self-play to build the neural nets. They would update the parameters and have a new version. They would have tournaments between all these different versions of AlphaGo to choose their new benchmark version (this helped refine the parameters, because the one that beat all the others probably had the best settings for the parameters).

One of the great ideas in the AG Nature Paper was to take the policy net and convert it to the value net using some fancy mathematics (a very beautiful but complicated algorithm).

This is where the beauty of DeepMind comes in. They had 20 people who were technically strong to be dealing with all the tricky details of the algorithm together with the help of Google's computing power.

They found that the SL policy net works better in MCTS than the stronger RL policy net, presumably because humans select a diverse beam of promising moves to explore, whereas RL optimizes for the single best move. Also, the RL value net (derived from the RL policy net) works better in AlphaGo than the similar value net derived from the SL policy net.

Evaluating the policy and value nets is slow, so they used asynchronous multi-threaded search, they used CPUs for simulation, and GPUs for policy / value net calls (GPUs were created to reduce lag on video games, but basically their highly parallel structure makes them more efficient than general-purpose CPUs for algorithms that process large blocks of data in parallel).

AlphaGo specs:
- single-machine: 40 search threads, 48 CPUs, 8 GPUs
- distributed: 40 search threads, 1202 CPUs, 176 GPUs

Simulations

Like RAVE, but more general (also used in Crazystone and Erica). Each move and the estimated win probability in the tree is cached, then the simulations used these cached moves and win probabilities.

RAVE uses these probabilities only for children of each node on the path to the root.

Simulation policy features:
Keeping track of and updating at each a little more in the simulation.
- response (1 pattern)
- save Atari (1)

- neighbour to previous move (8) (look at 8 surrounding cells of previous move)
- nakade (inside move (8192)
- response pattern (12-point diamond around previous move) (32 207)
- non-response pattern (8-points around cell) (69 338)

(Basically we're looking for patterns or certain kinds of shapes and if we see those, make a response, like the save-bridge in Hex.)

Nakade example:

```
if x group has no outside liberties ,
  where should x/o play ?

 x x x x x
 x - - - x
 x x x x x
```

Should play in the middle to make 2 eyes, or block the opponent from making 2 eyes. If there are no liberties for black outside, white can kill black by playing in the middle.

Tree Policy Features

Child selection policy (for selecting a leaf in MCTS, roughly speaking, we go best first).

All the simulation policy features plus:
- self-atari (1), move allows stone to be captured
- last move distance (34), Manhattan distance to previous 2 moves
    - is the value we get good or bad? Who knows! The NN will decide the importance of the feature.
- non-response pattern (32 307)

Discussion

AlphaGo was using a lot of things that other Go programs before it used too, but there's also a lot of new elements that it introduced. AlphaGo doesn't use RAVE, which is interesting because David Silver was one of the first people to use this (sometimes you have to throw out your ugly baby if it's not working).

Compared to DeepBlue, which used a handcrafted evaluation function, AlphaGo picked positions more wisely using the policy net and MCTS and evaluated positions more accurately, using the value net and simulations.

AlphaGo used neural nets trained using general purpose supervised/reinforcement learning methods. It trained with no-suicide, Tromp-Taylor rules (close to Chinese rules) with a komi of 7.5 (changing the ruleset or the komi would require retraining all of its neural nets).

A couple other interesting things not in the paper…

AlphaGo’s time management policy was found by ad hoc self-play machine learning. AlphaGo ponders (thinks during its own and its opponent’s clock time).

In the Lee Sedol match, AlphaGo’s time per move was close to constant. Compared to AG, Lee Sedol was often behind on time. AG pondering exaggerated this effect, putting Lee Sedol under more time pressure. 9-dan pros like Lee Sedol are used to playing with very little time.