Training AlphaGo

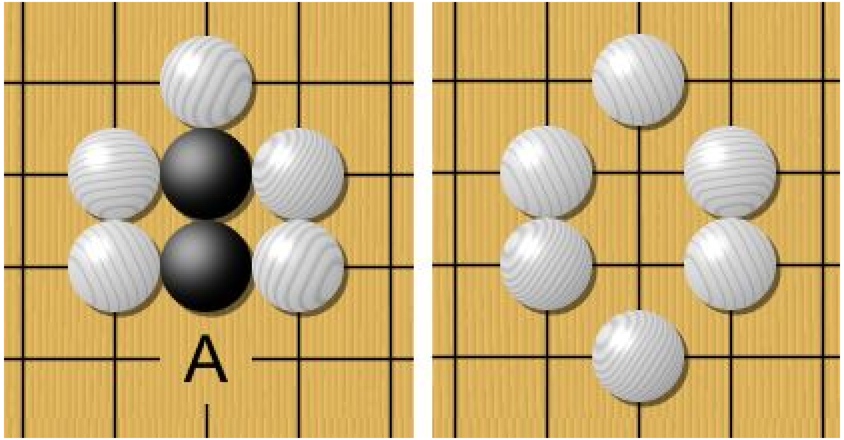
reinforcement learning and deep neural networks

Enrico Guiraud

“Go is exemplary in many ways of the difficulties faced by artificial intelligence: a **challenging decision-making task**, an **intractable search space**, and an **optimal solution so complex** it appears infeasible to directly approximate using a policy or value function”

Mastering the game of Go with deep neural networks and tree search. 2016, Silver, Huang

The game of Go



- black vs white on a 19x19 board (chess is 8x8)
- game ends when both players pass consecutively
- score is based on amount of “captured” territory

Go and AI

Considered much more difficult for computers to win than other games such as chess (most traditional methods do not scale at its size)

Go and AI

Considered much more difficult for computers to win than other games such as chess (most traditional methods do not scale at its size)

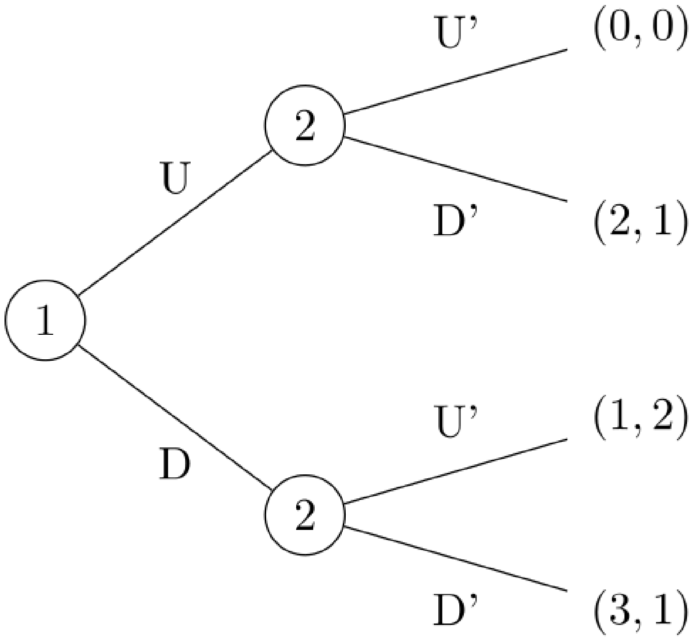
1997: IBM’s Deep Blue wins against Kasparov
2014: AlphaGo project **start**
2015: **win 499** matches against state-of-the-art, **loses 1**
October 2015: win against European Go champion (2-dan)
January 2016: Nature paper [1]
March 2016: win against Lee Sedol (9-dan, world top 5)

AlphaGo's problem 1/2

A game with perfect information is solved by **exploring the tree** of possible games.

The full tree has size $\textbf{breadth}^{\textbf{depth}}$

Chess: 35^{80}
Go: 250^{150}
A factor $\mathbf{10^{164}}$ between them



AlphaGo's problem 2/2

"challenging decision-making"
impossible to explicitly define winning strategies

"intractable search space"
impossible to perform full tree search

"complex optimal solution"
very hard to reduce breadth and depth effectively

AlphaGo’s solution

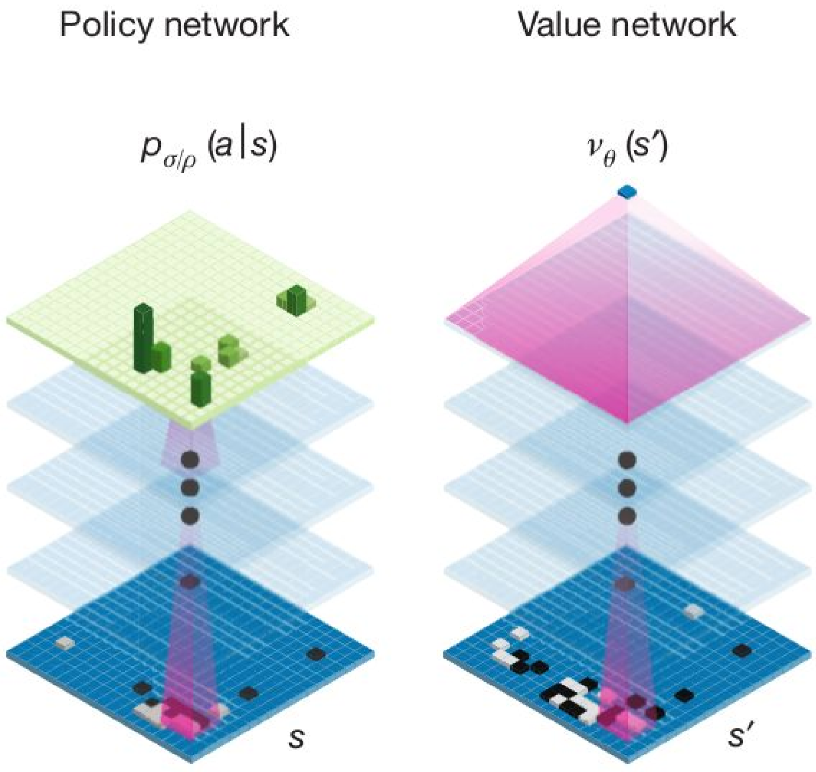
Monte Carlo Tree Search, but:

reduce breadth

- ‘Policy’ CNN network chooses good moves
- novel combination of supervised and reinforcement learning

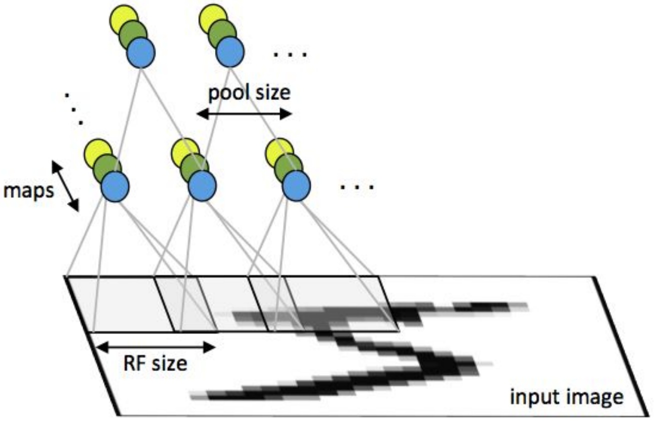
reduce depth

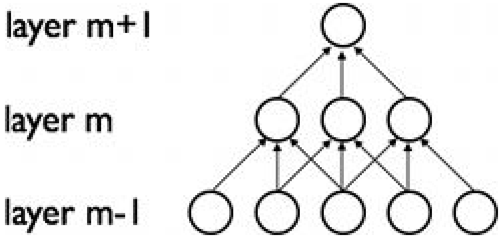
- ‘Value’ CNN network evaluates the board
- supervised learning



Convolutional neural networks (CNNs)

- feed-forward neural network (like MLPs)
- each neuron has a limited **receptive field**
  - less weights → higher scalability
  - correlations between pixels close to each other are taken into account
- trained with **stochastic gradient ascent** + **back-prop**
- **local** filters become increasingly **global** with the depth of the network





images from http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/ and http://deeplearning.net/tutorial/lenet.html

Reinforcement learning (RL) 1/2

Software agents learn to take the right *actions* to maximise a *reward* over *time*.
Successful applications in video-game AIs, robot AIs, helicopter control...

Reinforcement learning (RL) 1/2

Software agents learn to take the right *actions* to maximise a *reward* over *time*.
Successful applications in video-game AIs, robot AIs, helicopter control...

General setting:

- state of the system
- possible actions
- reward for the actions
  - not known to the agent
  - requires exploration
- policy function
  → which action to take
- value function
  → expected long-term reward

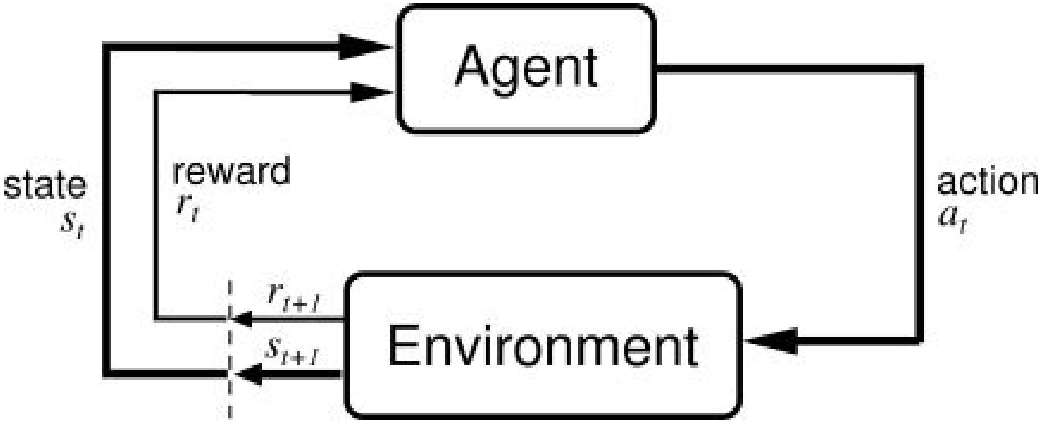


image from http://wiki.ubc.ca/Course:CPSC522/Reinforcement_Learning_with_Function_approximation

Reinforcement learning (RL) 2/2

Value-based RL

Learn estimates for the long-term rewards of actions

Policy is implicit

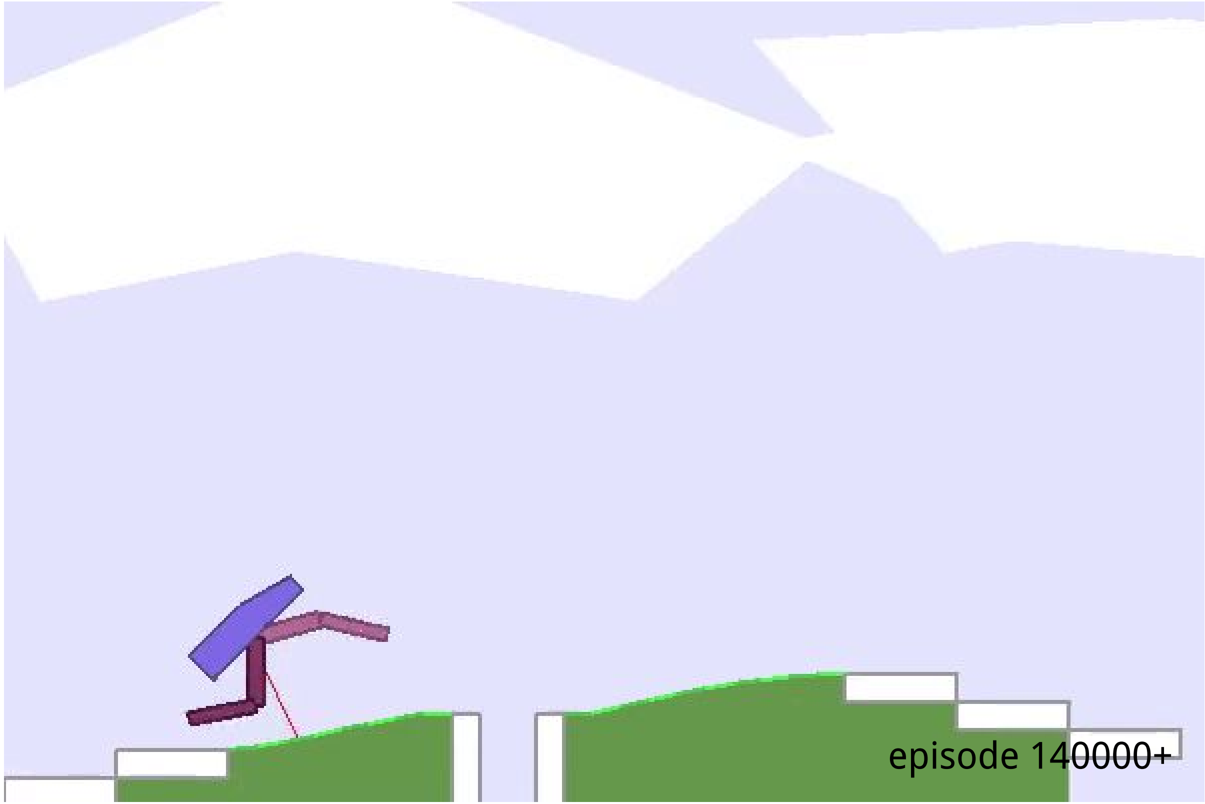
Policy-based RL

Define the policy as a parametric function of the state

Define a performance function for the policy

Vary parameters until policy has been sufficiently improved

Reinforcement learning: example



State:
velocity
angular velocity
cockpit angular velocity
position of joints
legs' contact with ground
10 distance measurements

Rewards:
+300 for reaching the end
+1 for moving forward
-100 for falling
-1 for applying motor torque

source: https://gym.openai.com

Value-based RL: Q-learning

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Value-based RL: Q-learning

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

- guaranteed to converge to optimum
- does not require modeling of the environment
- actions and states can be exponentially many (did anyone say Go?)
- slow learning if rewards only come after a long sequence of actions (*cough* Go *cough*)

Policy-based RL: policy gradient and REINFORCE

optimise long-term reward with respect to the policy parameters
→ (stochastic) gradient descent.

Objective: $J(\theta) = E\{r(\tau)\} = \int_{\mathbb{T}} p_\theta(\tau) r(\tau) d\tau$

Gradient: $\nabla_\theta J(\theta) = \left\langle \left(\sum_{k=0}^{H} \nabla_\theta \log \pi_\theta(\mathbf{u}_k | \mathbf{x}_k) \right) \left(\sum_{l=0}^{H} r_l \right) \right\rangle$

Policy-based RL: policy gradient and REINFORCE

optimise long-term reward with respect to the policy parameters
$\rightarrow$ (stochastic) gradient descent.

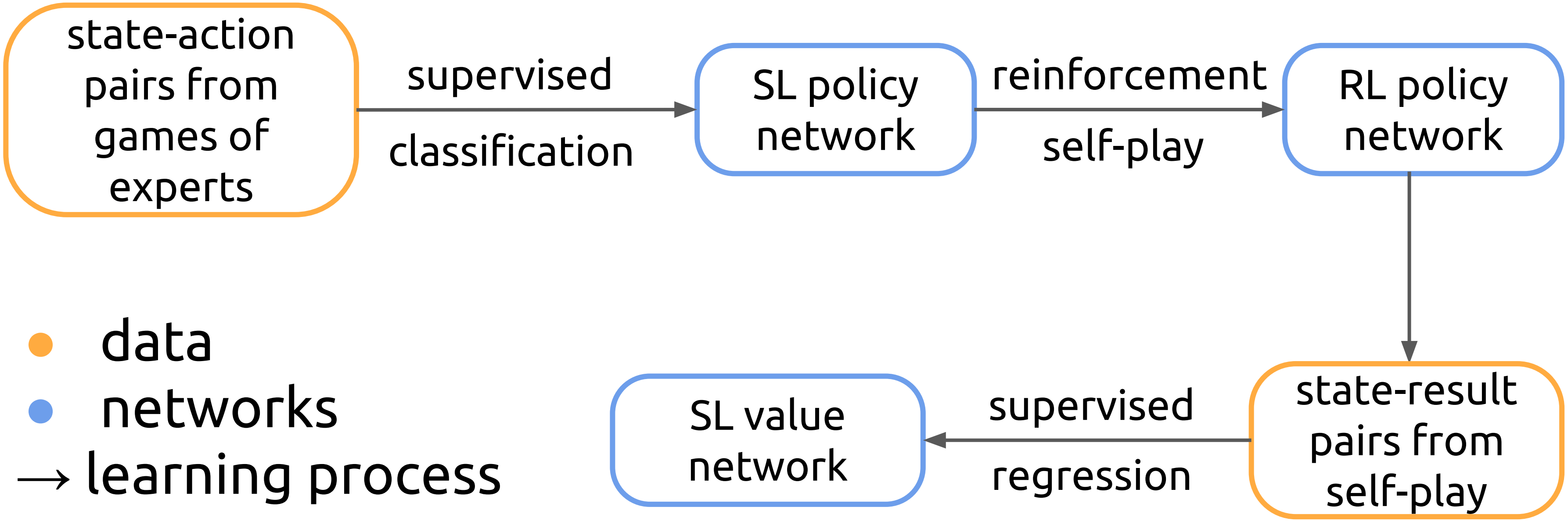
Objective: $J(\theta) = E\{r(\tau)\} = \int_{\mathbb{T}} p_\theta(\tau) r(\tau) d\tau$

Gradient: $\nabla_\theta J(\theta) = \left\langle \left(\sum_{k=0}^{H} \nabla_\theta \log \pi_\theta(\mathbf{u}_k | \mathbf{x}_k) \right) \left(\sum_{l=0}^{H} r_l \right) \right\rangle$

- policy may integrate the knowledge of experts
- policy may integrate information on the environment
- continuous states and action spaces are not an issue
- can get stuck in local optima
- requires to model a parametrised policy
- can improve the “style” of the teacher, cannot invent the Fosbury flop

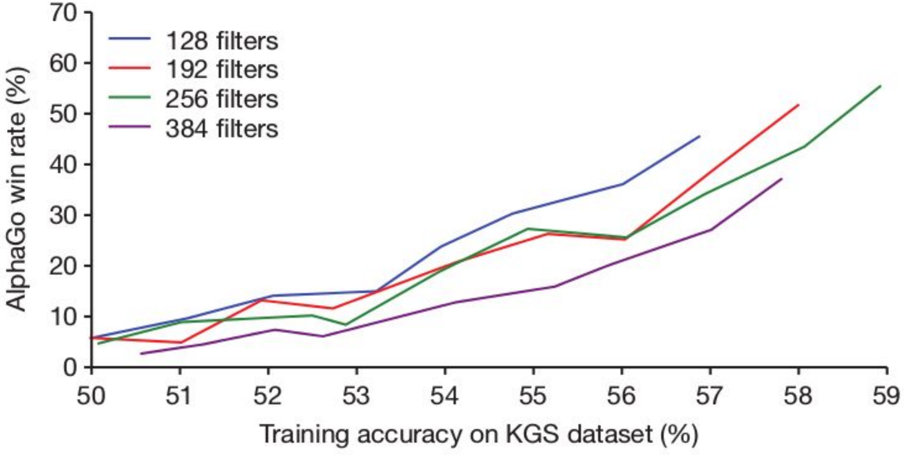
(see [5] for a very nice explanation of the math)

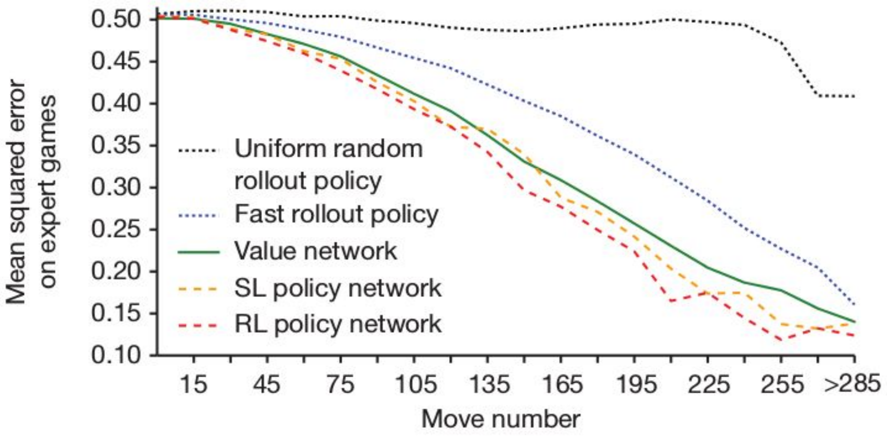
AlphaGo’s training pipeline



Results of AlphaGo’s training

- SL policy net has 57% accuracy (state-of-the-art was 44%)
- RL policy net won 80% of matches against SL policy net
- using no tree search, RL policy net won 85% of matches against state-of-the-art Go software





For the curious: hard numbers

AlphaGo
40 search threads
48 CPUs
8 GPUs

Distributed AlphaGo
40 search threads
1202 CPUs
176 GPUs

SL learning of policy net
8 million state-action pairs
340 million training steps
50 GPUs
3 weeks training time

RL learning of policy net
10’000 mini-batches of 128 games
50 GPUs
1 day training time

SL learning of value net
50 million mini-batches of 32 state-result pairs
50 GPUs
1 week training time

References

[1] Mastering the game of Go with deep neural networks and tree search. 2016, Silver, Huang
[2] Wikipedia, Wikipedia, Wikipedia
[3] Rectifier Nonlinearities Improve Neural Network Acoustic Models. 2013, Maas, Hannun, Ng
[4] Efficient BackProp. 1998, Yann LeCun et al.
[5] www.scholarpedia.org/article/Policy_gradient_methods

Thank you!

Stochastic gradient ascent (SGA)

Objective function: $Q(w) = \sum_{i=1}^{n} Q_i(w)$

Classic gradient ascent: $w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)$

Stochastic gradient ascent: $w := w - \eta \nabla Q_i(w)$

- Guaranteed to converge under loose conditions
- Much faster than the classic (exact) version
- Combined with back-prop, it is the standard for training of large NN [4]
- Shares many of the issues of classic gradient ascent

For the curious: tricks of the trade

This was not the whole story...

- rectifier nonlinearities (see [3])
- reward was 0 during the whole game,
  1 for winning, -1 for losing
- training the value network on full games instead of single board states resulted in strong overfitting
- symmetries: AlphaGo actually processes mini-batches of 8 board states, in parallel

How AlphaGo sees the board

- stone colour
- liberties
- legality of action
- turns since stone was played
- few other tactical features

Food for thought

"During the match against Fan Hui, AlphaGo evaluated thousands of times fewer positions than Deep Blue did in its chess match against Kasparov, compensating by selecting those positions more intelligently, using the policy network, and evaluating them more precisely, using the value network, an approach that is perhaps closer to how humans play."

Performance of different AlphaGo setups