Trends and Observations in Computer Go

Gian-Carlo Pascutto, gcp@sjeng.org

Up until 2006, computer Go had progressed at a snail's pace, with the best programs being as strong as the average amateur. However, the discovery of a new Monte Carlo Tree Search method set things in motion,
and they would not slow down.

Advances in computer vision, neural networks, graphics hardware, reinforcement and supervised learning, pattern scoring and tree search have propelled computer go players convincingly past humans in the 12 years that followed. The time has come where everyone with a personal computer can potentially have a professional lever sparring opponent and analysis partner. We will discuss what is still missing to make the picture complete.

The initial breakthrough was made by the introduction of an efficient Monte Carlo Tree Search method applicable to Go. For the first time, the strength of a Go program scaled with hardware. The ad hoc basis of the algorithm was quickly improved by linking the discovery with existing research on ... playing slot machines. In some ways, humanity was doomed at that point: further, inevitable increases in the speed of hardware would continuously improve the strength of the machines. In 1989 Feng-hsiung Hsu drew a graph plotting the speed vs strength of existing chess computers, and made a prediction about what would be needed to beat the world champion. He delivered with Deep Blue 7 years later. In 2006 the distance to human Go champions was big enough that no-one would draw such a plot, but the end result has turned out to be no different.

A second breakthrough was made by the re-introduction of neural networks into Go. Once a poster child of AI in the 1980's, but later discarded, advances such as convolutional networks, solutions to vanishing gradient problem, residual stacks and optimizations such as Batch Normalization made them powerful imagine recognition techniques, and advances in graphics hardware quickly made very large networks practical. Multiple teams quickly found out that their performance was just as good in Go, and a second jump in performance was achieved, propelling us more quickly to the end goal.

The last jump came from improved reinforcement learning methods. With the programs within striking distance, a last step needed to be made for the apprentice to surpass the master. By finding improved methods to let the programs improve from their own mistakes, computers finally surpassed the best humans.

Being at this point now, it is time to make the results usable to Go players. Leela Zero has replicated the above research in such a way that the needed software and networks are available to anyone. What do we need for good performance of a Go program, how do we make use of it, what have we learned so far, and what parts are still missing or could be improved?