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

David Sliver, ...
Outline

Abstract

Introduction

Supervised learning of policy networks

Reinforcement learning of policy networks

Reinforcement learning of value networks

Searching with policy and value networks
Abstract

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.
Abstract

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.
Introduction

- 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.

- In large games, such as chess (b35, d80) and especially Go (b250, d150), exhaustive search is infeasible, but the effective search space can be reduced by two general principles.

  1. 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 \( v(s) = v^*(s) \) that predicts the outcome from state \( s \).

  2. 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 \( s \).
Introduction

- The strongest current Go programs are based on MCTS, enhanced by policies that are trained to predict human expert moves. These policies are used to narrow the search to a beam of high-probability actions, and to sample actions during rollouts.
Introduction

We train the neural networks using a pipeline consisting of several stages of machine learning (Fig. 1).

1. We begin by training a supervised learning (SL) policy network $p$ directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high-quality gradients. Similar to prior work, we also train a fast policy $P_p$ that can rapidly sample actions during rollouts.

2. Next, we train a reinforcement learning (RL) policy network $p$ 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.

3. Finally, we train a value network $v$ that predicts the winner of games played by the RL policy network against itself.

4. Our program AlphaGo efficiently combines the policy and value networks with MCTS.
Figure 1 | Neural network training pipeline and architecture. a, A fast rollout policy \( p_\alpha \) and supervised learning (SL) policy network \( p_\sigma \) are trained to predict human expert moves in a data set of positions. A reinforcement learning (RL) policy network \( p_\rho \) is initialized to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network. Finally, a value network \( v_\theta \) is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set.

b, Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position \( s \) as its input, passes it through many convolutional layers with parameters \( \sigma \) (SL policy network) or \( \rho \) (RL policy network), and outputs a probability distribution \( p_\sigma(a|s) \) or \( p_\rho(a|s) \) over legal moves \( a \), represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters \( \theta \), but outputs a scalar value \( v_\theta(s') \) that predicts the expected outcome in position \( s' \).
Supervised learning of policy networks

The SL policy network $p(a|s)$ alternates between convolutional layers with weights $w$, and rectifier nonlinearities

- **Objective:** predicting expert moves
- **Input:** randomly sampled state-action pairs $(s, a)$ from expert games
- **Output:** a probability distribution over all legal moves $a$. 
Supervised learning of policy networks

We trained a 13-layer policy network, which we call the SL policy network, from 30 million positions from the KGS Go Server. The network predicted expert moves on a held out test set with an accuracy of 57.0% using all input features, and 55.7% using only raw board position and move history as inputs, compared to the state-of-the-art from other research groups of 44.4% at date of submission.
Supervised learning of policy networks
The RL policy network $p$ is identical in structure to the SL policy network, and its weights are initialized to the same values.

- Objective: improving the policy network
Reinforcement learning of policy networks

We use a reward function $r(s)$ that is zero for all non-terminal time steps $t<T$. The outcome $z_t = \pm r(s_T)$ is the terminal reward at the end of the game from the perspective of the current player at time step $t$: $+1$ for winning and $-1$ for losing.

\[
\Delta \rho \propto \frac{\partial \log p_\rho (a_t | s_t)}{\partial \rho} z_t
\]
Reinforcement learning of policy networks

The RL policy network won more than 80% of games against the SL policy network. Using no search at all, the RL policy network won 85% of games against Pachi. In comparison, the previous state-of-the-art, based only on supervised learning of convolutional networks, won 11% of games against Pachi and 12% against a slightly weaker program, Fuego.
Reinforcement learning of value networks

Estimating a value function $v^p(s)$ that predicts the outcome from position $s$ of games played by using policy $p$ for both players.

This neural network has a similar architecture to the policy network, but outputs a single prediction instead of a probability distribution. We train the weights of the value network by regression on state-outcome pairs $(s, z)$, using stochastic gradient descent to minimize the mean squared error (MSE) between the predicted value $v_\theta(s)$, and the corresponding outcome $z$. 
Reinforcement learning of value networks

- Data Set: self-play data set consisting of 30 million distinct positions, each sampled from a separate game. Each game was played between the RL policy network and itself until the game terminated.
Reinforcement learning of value networks
Searching with policy and value networks