AlphaGo Revisited

4/20/18
AlphaGo Concepts

When we talked about AlphaGo two months ago, you knew:

• Monte Carlo tree search
  • We therefore highlighted variations on the MCTS tree policy and default policy as illustrated by AlphaGo.

Now you also know:

• reinforcement learning
  • deep neural networks

  • We can therefore dive into the details.
Review: Monte Carlo tree search

Figure from Chaslot (2006)
AlphaGo Key Insights

Original AlphaGo
• Improve MCTS using neural networks.
  • Accurate policy network (SL) helps with selection.
  • Value network (RL) evaluates expanded states.
  • Fast policy network (SL) helps with simulation.
  • RL policy network used to generate self-play data.

AlphaGo Zero
• Improve deep reinforcement learning using MCTS.
  • Two-headed network learns value and policy.
  • Value head trained to predict game outcomes.
  • Policy head trained to predict MCTS visit probabilities.
Original AlphaGo Neural Networks

- Rollout policy: $p_\pi$
- SL policy network: $p_\sigma$
- RL policy network: $p_\rho$
- Value network: $v_\theta$

Data sources:
- Human expert positions
- Self-play Positions

Methods:
- Policy Gradient
- Classification
- Regression
- Self-play
Step 1: Supervised Learning

Data Set:

- **$X$** (inputs): board positions from top human games.
- **$Y$** (targets): the move the human played.
- Learns a probability distribution.

Two different networks learned from this data set:

- $p_{\pi}$: small network, fast to evaluate
- $p_{\sigma}$: large network, much higher accuracy
\( p_\pi \) and \( p_\sigma \) in MCTS

• The fast network, \( p_\pi \) is used to guide simulations.
  • Normal simulation phase: pick moves uniformly at random.
  • AlphaGo simulation phase: pick moves with the learned probabilities from \( p_\pi \)

• The accurate network, \( p_\sigma \) is used to guide selection.
  • Normal selection phase: UCB explore/exploit tradeoff
  • AlphaGo selection phase:
    • Exploit term is the same (estimated win probability).
    • Explore term proportional to prior (from \( p_\sigma \)) over visits.

\[
Q(s, a) + C \frac{p_\sigma(s, a)}{1 + \text{Visits}(s, a)}
\]
Step 2: Reinforcement Learning

- Gather data from self-play games.
- Train a new policy network to predict the moves made in these games: $p_\rho$
- Use this network in place of $p_\sigma$ to improve the selection phase.

- Train a value network to predict win probability: $v_\theta$
- Combine these values with simulation results.

$$v(s) = \lambda \times \text{outcome} + (1 - \lambda) \times \text{eval}$$
These were all convolutional NNs
AlphaGo Zero Neural Networks

Two-headed network

Making a Prediction using Head #1

Body

Head #1

Loss

Making a Prediction using Head #2

Body

Head #1

Loss

Head #2
Residual Network Architecture
MCTS as an RL operator

• Self-play games starting from random initial NN.

• No simulation phase. When a node is expanded:
  • Get a value from the value head of the network.
  • Get priors for each move from the policy head.

• Train the policy head to predict MCTS moves.
  • Probability for each move after MCTS rollouts.

• Train the value head to predict game outcomes.
  • Probability of winning the game (in self-play).
Other Notes

• Both versions choose the max-weight move in the selection phase.

• Both versions deliberately randomize their play in self-play games, choosing a move with probability proportional to MCTS visits.

Many of you are replicating these. What questions do you have?