Alpha Go Nature Paper

Perfect Information Game

perfect information games + perfect play - optimal value function $v^*(s)$
go b~250 d~150 => exhaustive search infeasible

Reducing search space

<u>reduce depth</u>

- position evaluation where subtree is replaced by $v(s) \sim v^*(s)$
- s = board state

<u>reduce breadth</u>

- use policy $p(a|s)$ probability distribution over move 'a' in 's'
- Monte Carlo rollouts (READUP)
  - weak amateur
  - search to maximum depth by sampling long sequences of actions for both players for policy p
  - Averaging over roll outs yields position evaluation
- Monte Carlo Tree Search (MCTS)
  - strong amateur
  - Uses Monte Carlo rollout to estimate value of each state in a search tree
  - More simulations => more accuracy
  - Asymptotically the policy converges to optimal play and evaluations converge to v*
- Deep CNN
  - 19*19 image for board position
  - Convolutional Layers to represent the position
  - Value network
  - policy network

First Stage Policy Networks

<u>Policy Network $p_\sigma(a|s)$ SL</u>

- supervised learning from expert human moves
- s = board state, a = legal moves over s
- $p_\sigma(a|s)$ = weights σ, [128-256 filter convolution layer + rectifier layer]*n + softmax layer => probablity distribution over 'a'
- trained on samples (s,a) pairs using stochastic gradient ascent to maximize likelihood of human move
  - $\Delta\sigma \propto \partial \log p(a|s) / \partial\sigma$
- 13 layer policy network using 30 million KGS Go Server positions
- Accuracy of 57% using all input features

- 55.7% using board position and move history in 3ms
- Larger networks => better accuracy | Slower evaluation
- Fast efficient learning learning updates, immediate feedback, high quality gradients

Fast policy $p\pi(a|s)$

- faster, less accurate compared to $p\sigma$
- $p\pi(a|s)$ = weights π, [linear softmax]*n using small features => 24.3% accuracy in 2μs
- Used to sample actions during roll outs

Second Stage Policy Networks

Policy network $p\rho(a|s)$ RL

- Policy Gradient Reinforcement learning, For goal of winning rather than maximizing predictive accuracy
- identical to $p\sigma(a|s)$, including initial weights $\rho = \sigma$
- play against random previous iteration (to stabilize and prevent overfitting)
- reward function, $r(s) = 0$ for non-terminal time steps $t<T$ - ***understand?***
- outcome/terminal reward, $zt = \pm r(sT)$ => win/loss ~ +1/-1
- weights are updated at each time step 't' by stochastic gradient ascent in the direction of max outcome
- $\Delta\rho \propto (\partial \log p\ (at\ |st)/\partial\rho) zt$
- move $a@t \sim p\ (\cdot|s@t)$
- win 80% of games against SL
- win 85% of games against pachi (strongest open source MCST)

Value Network $v\theta(a|s)$

- Reinforcement Learning to estimate value function $vp(s)$ that predicts outcome from position 's' using policy 'p'
  - $vp(s)=E[zt|st=s, a@t\ldots T \sim p]$
- similar to policy nw but single prediction instead of probability distribution
- value
  - $v^*(s)$ perfect play
  - $vp\rho(s)$ RL $p\rho$ network
  - $v\theta(s)$ = value network with weight θ
- trained by regression on state-outcome pairs (s.z) using stochastic gradient descent to minimize MSE between prediction and actual outcome
- $\Delta\theta \propto (\partial v\ (s)/\partial\theta)\ (z - v\theta(s))$
- To avoid overfitting RL policy nw was played with itself.
- $v\theta(s)$ ~ accuracy of MonteCarle rollouts using RL $p\rho$ network using 15000 times less computation
- Predicts winner of games played by RL against itself

Searching Policy and Value networks

- node (s, a)
    - action value Q(s,a)
    - visit count N(spa)
    - prior probability P(spa)
- traversal by simulation (descending without backup)
    - at time step 't' 'at' is selected from 'st'
    - $a@t$ =argmax.a(Q(st,a)+u(st,a))
    - *where* $u(s, a) \propto P(s, a) / (1+N(s,a))$, decays with repeated visits
- leaf sL at step L is processed once by pσ
    - output probabilities for each a, P(s,a) = pσ(a|s)
- leaf node evaluated in 2 ways
    - $v\theta(sL)$
    - zL - outcome of random rollout until terminal step T using fast rollout policy pπ
    - $V(sL) = (1 - \lambda)v\theta(sL) + \lambda zL$
- at end of simulation
    - visit count , $N(s,a)=\sum\mathrm{l}(s,a,i)$
    - action value $Q(s,a)= 1/N(s,a) \sum\mathrm{l}(s,a,i)V(siL)$
- SL policy pσ performed better than stronger RL policy pρ ~ diverse beam of promising moves
- However, $v\theta(s)$ ~ vpρ(s) performed better

AlphaGo efficiently combines policy and value networks with MCTS to select actions by lookahead search.

- async multi-threaded search on CPUs, policy and value networks in parallel on GPUs
- Single node 48CPU, 8GPU alphaGO
- Distributed 1202 CPU, 176 GPU alpha go

Performance

- with mixed networks (*λ=0.5)* performed best winning 95% of games
- value nw approximated outcomes of games played by strong but slow pρ
- rollouts evaluate games played by weaker but faster pπ