AlphaGo for RLers

mooopan
Go as an RL problem

- The complete environment model is available
  - state transition function $s' = f(s,a)$
  - reward function $r(s,a) = 0, 1$ or $-1$
  - terminal condition
  - we can self-play as many times as time allows
  - we can use lookahead search as long as time allows
- The opponent model is not available, but we can assume each player tries to maximize his or her return
Models in AlphaGo

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

Data

- Human expert positions
- Self-play positions

Neural network

- Classification
- Self-play
- Regression
SL policy network

- $p_\sigma(s,a)$
- Maximize the log-likelihood of human moves

$$\Delta \sigma = \frac{\alpha}{m} \sum_{k=1}^{m} \frac{\partial \log p_\sigma(a^k|s^k)}{\partial \sigma}$$

- $p_\sigma$ is computed by a softmax layer (Gibbs policy)
- 30M positions from KGS
- Rollout policy $p_\pi$ is trained in a similar way with a different dataset
RL policy network

- $p_\rho$ ($\rho$ is initialized with $\sigma$)

- Maximize the expected return by Williams’ REINFORCE

$$\Delta \rho = \frac{\alpha}{n} \sum_{i=1}^{n} \sum_{t=1}^{T_i} \frac{\partial \log p_\rho(a_t^i|s_t^i)}{\partial \rho} (z_t^i - v(s_t^i))$$

- $v(s)$ is a baseline, added to reduce the variance of policy gradient estimation

- zero in the first pass, $v_\theta$ in the second pass

- Opponents are randomly sampled from old parameters
Value network

- $v_{\theta}$

- Training a value network = policy evaluation e.g. TD($\lambda$)
  - Given a policy $p_{\rho}$, compute $v^{[p_{\rho}]}(s) = \text{expected return from } s$
  - Assume both players follow $p_{\rho}$

- We can use Monte-Carlo estimation by repeating simply following $p_{\rho}$ and observing a return
  - They observed that training on every position in a game lead to overfitting because they are strongly correlated
  - So, they created a new dataset by sampling only one state from each game
Dataset creation

- Repeat 30M times:
  - Start from an empty board s_1
  - Follow p_σ at s_1,⋯, s_{U-1}, U ~ unif(1,450)
  - Select a move uniformly at random at s_U
  - From s_{U+1}, follow p_ρ till the end and we can observe a return r
  - Add (s_{U+1}, r) to the dataset

- Now we have 30M i.i.d. datasets for training a value function by minimizing squared errors

\[
\Delta \sigma = \frac{\alpha}{m} \sum_{k=1}^{m} (z^k - v_{\theta}(s^k)) \frac{\partial v_{\theta}(s^k)}{\partial \theta}
\]
Why not TD learning?

- They don’t mention it
- Monte-Carlo estimation (=TD(1)) has no bias but high variance
- TD learning (=TD(\(\lambda\)) with \(\lambda < 1\)) has low variance but some bias
  - Maybe they found this bias problematic
Why not Actor-Critic?

- They don’t mention it either :( 

- Actor-Critic can simultaneously learn $p(a|s)$ and $v(s)$, $v(s)$ helping training of $p(a|s)$ 

- Is it partly because $v(s)$ is difficult to learn compared to $p(a|s)$?
Monte-Carlo Tree Search

- Nodes and/or edges in a search tree hold statistics about how good they are
- Statistics are updated by backpropagating returns of simulations, so-called rollouts
- The search tree grows so that it can search promising portion of the state space more deeply
**APV-MCTS**

- APV stands for asynchronous policy and value
- Each node in a search tree contains edges \((s,a)\) for all legal actions
- Each edge stores a set of statistics
  - \(P(s,a)\): prior probability \((\sim p\_\sigma(s,a))\)
  - \(N\_v(s,a)\): number of calls of \(v\_\theta\) below this edge
  - \(W\_v(s,a)\): sum of returns of \(v\_\theta\) below this edge
  - \(N\_r(s,a)\): number of rollouts below this edge
  - \(W\_r(s,a)\): sum of returns of rollouts below this edge
- \(Q(s,a)\):
  \[
  Q(s,a) = (1 - \lambda) \frac{W_v(s,a)}{N_v(s,a)} + \lambda \frac{W_r(s,a)}{N_r(s,a)}
  \]
Selection

- Descend from root to leaf following:

$$\arg\max_a Q(s, a) + u(s, a)$$

$$u(s, a) = c_{\text{puct}} P(s, a) \frac{\sqrt{\sum_b N_r(s, a)}}{1 + N_r(s, a)}$$

- Assign virtual loss $n_{\text{vl}}$ to the statistics of selected edges so that they are not favored in other threads’ selection phases
  
  - $N_r(s, a) += n_{\text{vl}}, W_r(s, a) -= n_{\text{vl}}$

- Similar but different from UCT, which is guaranteed to converge to optimal values

$$u_{\text{uct}}(s, a) = c_{\text{uct}} \frac{\sqrt{\log \sum_b N_r(s, a)}}{N_r(s, a)}$$
Expansion

- If a leaf edge’s $N_r(s,a)$ exceeds a certain threshold, add a new node
  - $N_r$, $W_r$, $N_v$, $W_v$ of its edges are all initialized with 0
  - $P(s,a)$ is initialized with outputs of SL policy network $p_\sigma$
    - They SL works better than RL here
Evaluation

- If the leaf node’s position $s_L$ is not yet evaluated by $v_\theta$, added to a queue so that it will be evaluated by $v_\theta$ asynchronously.

- Simulate from $s_L$ following $p_\pi$ and observe a return $r$. 
Backup

· Update the statistics from leaf to root:
  
  · $N_r(s,a) += 1$
  
  · $W_r(s,a) += r$

· If evaluation of $v_\theta$ is complete then another backup starts asynchronously

  · $N_v(s,a) += 1$

  · $W_r(s,a) += v_\theta(s_L)$
AlphaGo is strong

- As you know
- SOTA
- Beat Lee Sedol 4-1
Lessons

- Don’t fear high variance (if you have massive computation resources)

- REINFORCE can scale surprisingly well

- Using every state in an episode for training can lead to overfitting

- We still need lookahead search