Understanding AlphaGo

Machine Learning papers

Bitdefender ML Team
eburceanu@bitdefender.com

June 6, 2016

Intro

NN - Concepts

AlphaGo

Conclusions

Presentation Purpose

- brief NN introduction
- what are the components of AlphaGo (Nature magazine paper, January 2016)
- how do they link
- speculate about mistakes in games 3, 4

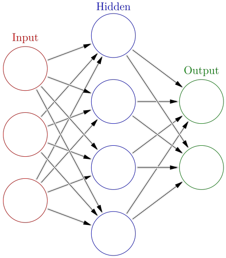
Neural Networks I

- NN purpose: given some data, approximate a good function of the input
  - $y_{nn} = output_{nn} = f_{W_f}(g_{W_g}(h_{W_h}(...(input))))$
  - $y_{correct} = output_{correct} =$ known from data
  - define how different is y_{nn} from the $y_{correct}$

Neural Networks I

- NN purpose: given some data, approximate a good function of the input
  - $y_{nn} = output_{nn} = f_{W_f}(g_{W_g}(h_{W_h}(...(input))))$
  - $y_{correct} = output_{correct} =$ known from data
  - define how different is y_{nn} from the $y_{correct}$

  - 150 vs 300
  - cat vs dog
  - $(y_{nn} - y_{correct})^2$
  - small vs big penalties



Neural Networks II

- minimize cost = "difference" between y_{nn} and $y_{correct}$
- find proper parameters $f = f_{W_f}(X) = W_f * X + b_f$
- each time we make a mistake, find what caused it and make small adjustments in the network (fancy name: back-propagation)
- **Q**: How W influences the *Cost*?

Neural Networks II

- minimize cost = "difference" between y_{nn} and $y_{correct}$
- find proper parameters $f = f_{W_f}(X) = W_f * X + b_f$
- each time we make a mistake, find what caused it and make small adjustments in the network (fancy name: back-propagation)
- **Q**: How W influences the *Cost*?
  - iteratively, $W = W - \alpha * \nabla_W Cost$ (Taylor)
  - end: W will have a value for which $Cost$ is minimum
  - **Q**: How else?

Why Go?

- perfect information game and zero-sum game
  - V_{optim} exists (state value, under optim play by all players)
- solving it:
- exhaustive
  - $branches^{depth}$ sequences of moves
  - chess: $35^{8}0$
  - go: $250^{1}50$ ($3^{3}61$ table states)
- search tree
  - truncate (alpha-beta pruning)
  - sampling (Monte Carlo Tree Search - MCTS)
  - **Q**: Pro and cons?

AlphaGo paper

- **Q**: Who read the paper?

AlphaGo paper

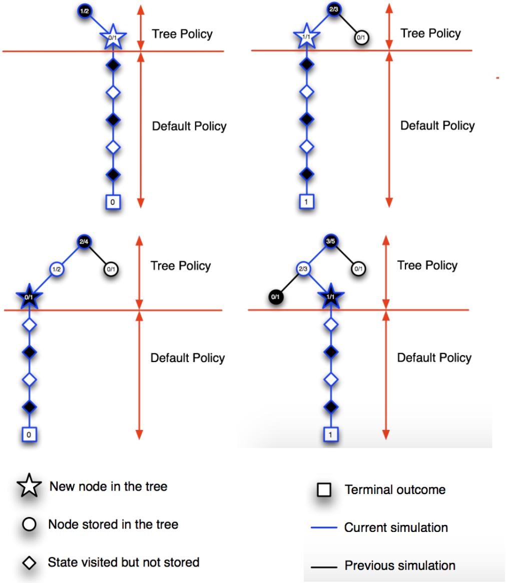
- **Q**: Who read the paper?
- "Mastering the game of Go with deep neural networks and tree search"
- David Silver - first author, maybe the best RL...human expert? PhD in AI on Go
- in: dataset of expert moves (KGS)
- out: first program that defeats a professional player
- components:
  - Monte Carlo Tree Search (MCTS)
  - NN for $value(state)$: $V_\theta(s)$
  - NN for "strategy" (policy), $policy(action|state)$: p_σ, p_π, p_ρ

BlackBox AlphaGo

Bitdefender

- $p_\sigma(action|state)$ policy
  - learns to do expert human moves
- $p_\pi(action|state)$ policy
  - learns to do **fast** expert human moves
- $V_\theta(state)$ value function
  - evaluates a state
- tree search algorithm
- **Q**: To do what?

Monte Carlo Tree Search I



Monte Carlo Tree Search II

- simulate and find the best move
- statistics
  - $P(s, a)$ - p_σ prior probability
  - $N_v(s, a), N_r(s, a)$ - count
  - $W_v(s; a), W_r(s, a)$ - additive score for all simulations ($V_\theta(s)$ and $r(s)$)
  - $Q(s, a)$ - mixed score
  - for all legal actions. **Q**: How?
- $+$asynchronous, $+$PUCT (exploration factor)

Monte Carlo Tree Search III

- Steps
  1. Selection
     - statistics $t = argmax_a(Q(s_t, a) + u(s_t, a))$ (best edge)
     - $u(s, a) = c_{puct} P(s, a) \frac{\sqrt{\sum_b Nr(s,b)}}{1+Nr(s,a)}$
     - until no more expanded states (L - we need to evaluate it)
  2. Evaluation
     - $V_\theta(state_L)$ or
     - reward of rollout with $p_\pi(a|state_L)$, for both players
  3. Backup
     - $N_r + 1, N_v + 1, W_v + V_\theta(state_L), W_r + reward$, async, n_{VL}
     - $Q(s, a) = (1 - \lambda) \frac{W_v(s,a)}{N_v(s,a)} + \lambda * \frac{W_r(s,a)}{N_r(s,a)}$
  4. Expansion
     - $N_r(s, a) > n_{thr}$, init all to 0
- run multiple times to build the tree

Neural Nets - tools for MCTS I

Bitdefender

- $p_\sigma(action|state)$ policy
  - supervised learning (55%, SotA 44%, eval in 3 ms)
  - expert dataset (30 mil, in: Go board, out: move = action)
  - 48 planes - 19x19, locally preprocessed Go features
  - 13 layers, conv 5, 3 + ReLU + softmax legal moves
  - used as prior for P(s, a) in MCTS (u(s,a) component)
  - $\delta p = \frac{\partial log(p(a|s))}{\partial p}$
- $p_\pi(action|state)$ policy
  - supervised, feature-engineered input
  - conv + relu + softmax (24%)
  - simple and very fast (eval in 2 μs)
  - used for rollout

Neural Nets - tools for MCTS II

- $p_\rho(action|state)$ policy
  - same structure, init with p_σ
  - 80% vs p_σ
  - play against itself (reinforcement learning, ± 1 reward)
  - not exactly itself, but sampling from previous versions
  - $\delta p = \frac{\partial log(p(a|s))}{\partial p} * r(s_T)$
- $V_\theta(state)$ value function
  - expected value of reward
  - supervised learning on p_ρ (playing history)
  - sample 1 state per game, 30 mil (**Q**: Why?)
  - similar architecture with p_σ (without softmax)
  - regression, RL, unbiased estimator of the reward
  - $\delta\theta = \frac{\partial log(V_\theta(s))}{\partial\theta} * (z - V_\theta(s))$
  - average for all symmetries (8)

Conclusions

- **Q**: Why it is not maximizing the margin? (game 3)
- **Q**: Time management?
- **Q**: Why did it figure out so late that it is loosing? (game 4)
- Conclusions
  - great things are simple
  - lots of engineering challenges (not addressed today)
  - insights and experience in the field (David Silver, Aja Huang)

Other Questions?

References

- **AlphaGo paper**, payed from Nature, just request it from the ML team, https://deepmind.com/alpha-go.html
- **Reinforcement Learning course**, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- **Neural Nets course**, http://cs231n.stanford.edu/
- **Reinforcement Learning Book**, Algorithms for Reinforcement Learning, Csaba Szepesvari
- **MCTS survey**, http://www.cameronius.com/cv/mcts-survey-master.pdf