Generals Exam: Enhancing AlphaZero to Accomodate a Larger Policy Space and Applications

Zachary Hervieux-Moore

18/05/18

Overview

1. Review & Preliminaries
   - Markov Decision Processes and Dynamic Programming
   - Multi-armed Bandit Theory
   - Monte Carlo Tree Search
2. AlphaZero
3. Current Work
   - Motivation
   - Progress
4. Demo
5. Next Steps and Future Directions

Markov Decision Processes

MDPs are 5-tuples consisting of $(S, A, P.(\cdot), R.(\cdot), \gamma)$

- $\mathcal{S}$ is the set of all states, $S_t \in S$ is a state at time t
- $\mathcal{A}$ is the set of all states, $a_t \in A$ is an action performed at time t
- $P_a(S_{t+1}|S_t)$ is the transition probability from state S_t to the next state under action a_t
- $R_a(S_{t+1}, S_t)$ is the reward received after performing action a_t in state S_t and ending up in S_{t+1}
- $\gamma \in [0, 1]$ is the discount factor on future rewards (how greedy you are)

Dynamic Programming

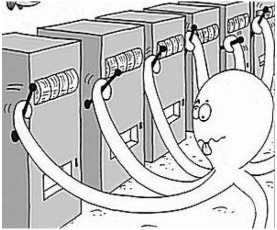
- Dynamic programming is the process of solving a complex problems by solving many simpler problems
- Backwards induction can be used to solve MDP finite horizon problem (finding best policy π that maximizes reward)

$$V_\pi(s) = \mathbb{E}[\sum_{t=0}^{k} R_\pi(s)|s,\pi]$$

- Rich history of solving MDP infite horizon problems using Bellman equation and backwards induction on stochastic MDPs to get value iteration

$$V_{i+1}(s) = \max_a \left\{ \sum_{s'} P_a(s'|s)(R_a(s',s) + \gamma V_i(s')) \right\}$$

Multi-armed Bandit Problems



- K machines that give out rewards according to some distribution in $[0, 1]$
- $X_{i,n}$ is a random variable of the reward for pulling machine i for the n^{th} time
- Let $T_j(n)$ be the number of times machine j is pulled after n turns
- Regret:

$$\mu^* n - \sum_i K \mu_j T_j(n)$$

Upper Confidence Bound

- Algorithm that balances exploration vs. exploitation to achieve optimal asymptotic lower bound for regret
- Regret after n rounds:

$$O\left(\sqrt{Kn\log(n)}\right)$$

- Algorithm:

$$\arg\max_j \bar{X}_j + \sqrt{\frac{2\ln n}{n_j}}$$

- Variant: if $P_1, \ldots, P_K$ are probabilities of arms being optimal then regret becomes

$$O\left(\sqrt{n\log(n)}(\sum_i \sqrt{(P_i)})^2\right)$$

EXP3

- Algorithm that also achieves the optimal regret lower bound but for the much more general case of adversarial multi-armed bandits
- Algorithm:

$\gamma \in [0,1]$, $w_i(0) = 1$ for $i \in \{1, \dots, K\}$, $n = 0$
while $n <$ max_iterations **do**

1) $p_i(n) \leftarrow= (1-\gamma)\frac{w_i(n)}{\sum_{j=1}^{K} w_j(n)} + \frac{\gamma}{K}$
2) $a_n \sim$ distribution of $p_i(n)$'s
3) get reward $X_{a_n}(n)$
4) $\bar{X}_{a_n}(n) = X_{a_n}(n)/p_i(n)$
5) $w_i(n+1) = w_i(n)e^{\gamma \bar{X}_{a_n}(n)/K}$
6) $w_j(n+1) = w_j(n)$ for all other $j \neq i$

end while

Monte Carlo Tree Search

- MCTS is a heuristic search through a tree but can be thought of as a decision process. It is used to sample the rewards through a path to get a lookahead policy.

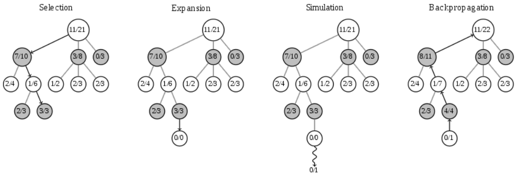


Figure: The four main components of MCTS

Bandit Algorithms on MCTS

- In MCTS, sampling of the paths through the tree can be thought of as a multi-armed bandit problem
- MCTS is modeled like this because the finite/infinite horizon in MDPs is not enforced in certain decision trees
- This leads to the selection step following schemes like UCB, called UCT, which work well in practice

Dynamic Programming Formulation of UCT

- If we assume that the process is finite horizon, i.e. the process cannot loop through the same state ad infinitum, we can model MCTS as a lookahead policy from an MDP
- Recall the Bellman equation

$$V_t(s) = \max_a R_a(S_{t+1}, S_t) + V(S_{t+1})$$

- S_t is the node in the tree, a_t is an edge of the node, we can estimate $V_t(s)$ by sampling via UCT to get the lookahead policy

$$\tilde{V}_t(\tilde{S}_t) = \max_a R_a(\tilde{S}_{t+1}, \tilde{S}_t) + \tilde{V}(\tilde{S}_{t+1}) + c_{uct}\sqrt{\frac{\ln N_(\tilde{S}_t)}{N(\tilde{S}_t, a)}}$$

History of AlphaZero

- 3 different versions each improving on itself
- AlphaGo (January 2016)
  - Used supervised learning to learn an initial neural network then used self play for further training
- AlphaGo Zero (October 2017)
  - Removed the supervised learning part and engineering tricks of AlphaGo
  - Went from two neural networks to just one
- Alpha Zero (December 2017)
  - Removes evaluation component of AlphaGo Zero and other engineering tweaks
- In what follows, I will say Alpha Zero to mean both of the last two

Overview of AlphaZero

- I break AlphaZero into three main components
- Policy and Value Learning
  - Comprises the neural network and the training part of the algorithm
- Self Play
  - The way that the algorithm generates data for the previous step
- Evaluator
  - Consists of the logic that determines which neural networks to keep and throw away
  - Not in Alpha Zero

Policy and Value Learning

- Structure of neural network:
  - Residual block tower consisting of 40 blocks
  - Output of tower is fed into two heads
  - Value head produces a single value representing probability of winning
  - Policy head outputs a vector of probability logits
- Loss:

$$(p, v) = f_\theta(S), \quad \ell = (z - v) - \pi^T \log p + c\|\theta\|^2$$

- Uses batches of size 4,096 turns sampled from the previous 500,000 games

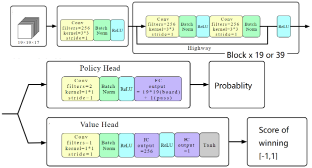


Figure: Diagram of network architecture

Reinforcement Through Self Play

- Highly distributed set of games played through MCTS
- 800 rounds of MCTS performed using the following version of PUCT:

$$a_{t'} = \max_a \frac{V(S_L)}{N(S_{t'})} + c_{puct} P(S_{t'}, a) \frac{\sqrt{\sum_a N(S_{t'}, a)}}{N(S_{t'})}$$

- Final selection of action, if $t < 30$:

$$\pi_a(S_t) = \frac{N(S_t, a)}{\sum_b N(S_t, b)}$$

Otherwise,

$$\max_a \frac{N(S_t, a)}{\sum_b N(S_t, b)}$$

- To encourage exploration, add Diriclet noise $\eta \sim \text{Dirichlet}(0.03)$ to $P(S_0, a)$

$$P(S_t, a) = (1 - \epsilon) P(S_t, a) + \epsilon\eta$$

Evaluating

- After a 1,000 training iterations, it is time to evaluate the network to see if it is better
- Using the checkpoint, play 400 games with reduced exploration
  - if the checkpoint beats the previous best 55% of the time, it becomes the new best and update the self play networks
  - Otherwise do nothing
- This part is removed in AlphaZero but is still useful to calculate ELO at this stage

Putting it All Together

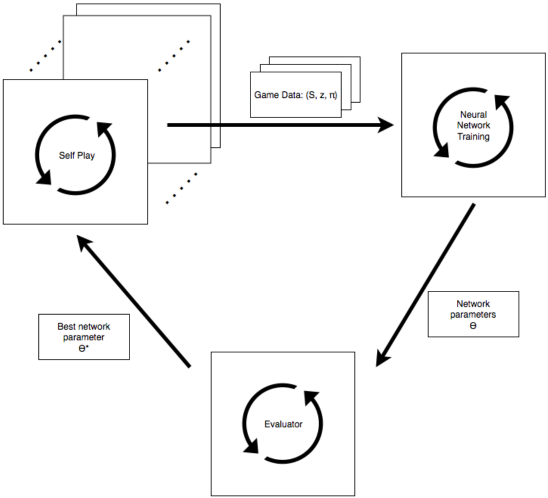


Figure: High level block diagram of AlphaZero

Mathematical Formulation of AlphaZero

We can model AlphaZero as an MDP with the following parameters:

- $S_t = (X_t, B_t)$, the state consists of the physical game state X_t and the belief state of the correct policy $(\bar{P}_{X_t}(\cdot))$ and value $(\bar{V}_{X_t})$ at node X_t which is given by our neural network $(\bar{P}_{X_t}(\cdot), \bar{V}_{X_t}) = f_\theta(X_t)$
- Action space is all possible moves, possibly encoded
- No transition probabilities, everything is deterministic
- Reward $R_a(S_{t+1}, S_t)$ is 0 if nothing or a draw occurs, 1 if you win, and -1 if you lose
- You can then view the MCTS in AlphaZero as a stochastic lookahead policy for the stochastic dynamic program of playing the game

Motivation: AlphaZero and Action Space

- AlphaZero handles large state spaces really well due to the neural network
- However, network outputs a policy vector which can get quite large
- UCT exploration policy is determined by value so they seem intrinsically tied

Motivation: Chess Example

Input States Tensor of 8x8x119

- 6 - positions for white pieces, 6 - positions for black pieces, 2 - keeps track of repetitions
- Repeated 8 times for a 8-step history
- 7 of more planes to keep track of player, castling, etc.

Output Policy Tensor of 8x8x73 which represents picking a square and moving it to one of 73 possible different moves

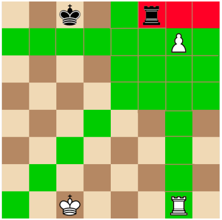


Figure: How AlphaZero encodes chess moves

Motivation: Scrabble

- Scrabble represents an interesting case. On one hand, MCTS with classical AI heuristics became super human in the early 2000's. Now, a more sophisticated algorithm based on MCTS could not even load into memory
- Why? Must encode for every possible Scrabble move possible, with 100,000 English words and 15x15 board, intractable

Figure: Scrabble Board

Motivation: Potential Solution

- Must get rid of the policy component of the neural network

$$(v, \cancel{p}) = f_\theta(s)$$

- Couple of different ways:
  - Use a MCTS lookahead policy that does not rely on a prior distribution (what I've done)
  - Induce a prior probability distribution based on the values of the next states
- Added benefit of much more simple computations in the neural network

Framework Overview

- The framework that I am developing to experiment is modular in several different aspects:
  - Games can be easily added via a standard interface
  - Different MCTS variations can be added via a standard interface
- Allows for quicker prototyping and experimentation
- The core of the framework interfaces with the above, does not use game/mcts specific implementation

Framework Progress

- Current implementation: $>$2,000 LOC
- Over 5,000 LOC written during development
- Everything runs on a local machine: self play using MCTS framework and training through tensorflow
- Distributive interface implemented through a ZMQ server with database storage

Game Class

```
class Game:

def generate_moves(self):
  raise NotImplementedError("Generate moves not implemented")

def transition(self):
  raise NotImplementedError("Transition not implemented")

def set_state(self):
  raise NotImplementedError("Set state not implemented")

def visualize(self):
  raise NotImplementedError("Visualize not implemented")
```

MCTS Class

```
class MCTS:

def __init__(self, root, game, node_config):
    self.tree = Tree(root, node_config)
    self.game = game
    self.node_config = node_config

def selection(self):
  raise NotImplementedError("Selection not implemented")

def expansion(self):
  raise NotImplementedError("Expansion not implemented")

def simulation(self):
  raise NotImplementedError("Simlation not implemented")

def backpropagation(self):
  raise NotImplementedError("Backpropagation not implemented")

def selection_final(self):
  raise NotImplementedError("Final selection not implemented")

def run(self):
  raise NotImplementedError("Run not implemented")
```

Parallel Architecture

For distributive performance, the framework breaks the tasks into workers for the major components of AlphaZero

- Self Play Worker
- Training Worker
- Oracle Worker
- Evaluator Worker

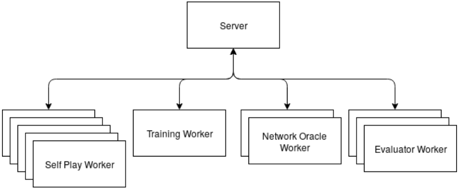


Figure: Block Diagram of Architecture

Technical Difficulties: Distributive Computing

- Distributive computing is hard in general when the processes must communicate
- The ZMQ architecture works very well but is the communication overhead worth it?
- Might be requried due to the Global Interpreter Lock in CPython

Technical Difficulties: Differences with AlphaZero

- Google has access to far greater computational resources than Princeton and so their algorithm has a slightly different architecture
- Google has access to TPUs further enhancing their performance
- Each self play worker has their own copy of the network
    - No need for oracle worker
    - Much faster for MCTS iterations
- Google most likely implemented it in C++ for better speed

Technical Difficulties: Hyperparameters

- Many different hyperparameters to tune and might be game/network specific
- MCTS hyperparameters:
  - C_{uct} - for UCT algorithm which is the exploration-exploitation tradeoff
  - γ - for EXP3 defines mixture of sampling distributions
- Exploration hyperparameters
  - τ - number of turns that the final MCTS selection will sample randomly
  - ϵ - Amount of Dirichlet noise to add to root to encourage exploration/non-deterministic play
- Loss weights

Proof of Concept: Pawns

- Simplified version of chess where there are only pawns and the goal is to be the first one to get a pawn to the end
- Ran both AlphaZero and my version with EXP3 for 1,000,000 training iterations
- Both develop more complicated structures as time progresses

Demo Time

Next Steps: Low Hanging Fruit

- Make code more performant in the MCTS loop
  - AlphaZero 0.4s vs. 4s
  - AlphaZero 800 rollouts vs. 100
- Optimize the parallelization
- Add more complicated games and validate
- Refactor code to make more modular

Next Steps: Compute Cluster

- Write slurm wrapper for the framework
- Need to refactor optimization code to allow splitting of batches across multiple GPUs
- Predict that I will be able to get the framework doing 500 games per second on 1,000 CPUs which matches the production of AlpheZero

Next Steps: ELO Evaluation

- Standard in all of the AlphaZero papers
- Two ways ELO calculation can be done:
  1. Using the formula:

$$\mathbb{P}(a \text{ beats } b) = \frac{1}{1 + e^{c_{elo}(R_b - R_a)}}$$

  2. Using techniques that take into account cross-play

Next Steps: Scrabble

- The ultimate goal of the project
- Generating moves presents a much harder challenge than the other games
- Pre existing libraries written in C++ through Quackle

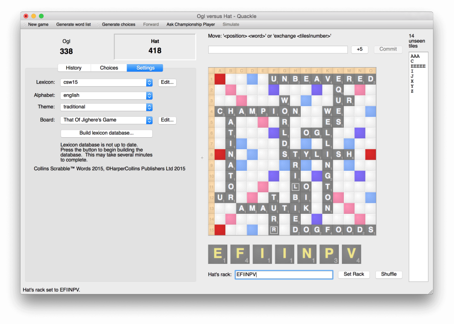


Figure: Quackle Environment

Future Directions: Robotics

- The algorithm is well suited for the world of robotics - large state space and many actions
- MuJoCo is an open source environment from the University of Washington that is widely used to benchmark control algorithms

Figure: MuJoCo physics environment

Future Directions: Multiplayer Games

- Prior work on UCT applied to multiplayer games show that the the strategy extracted from UCT is a mixed strategy equilibrium
- Avenue of research, enhancing AlphaZero with online learning to exploit a suboptimal player
- Halite II would be a good game to tackle

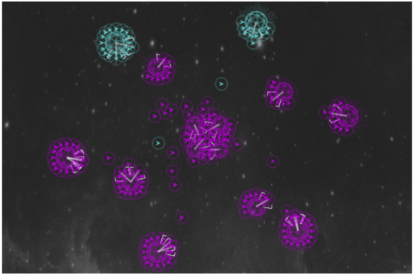


Figure: A game of Halite II in progress

Future Directions: SmartDriving Cars

- Following the work done by Chenyi, this could be extended to learning how to drive from the use of the TORCS environment
- Chenyi had to manually drive car simulator to generate his data
- Chenyi showed that the results work surprisingly well when transfering from simulator to real world

(a) Autonomous driving in TORCS

(b) Testing on real video

Figure: Chenyi's work in a simulator and real environment

Future Directions: Real World Applications

- Method is capable of being applied to anything with a simulator that can generate “self-play” data
- Interested in working with Warren in any domain that he sees the benefit of using a AlphaZero like algorithm
- Examples: optimizing electrical network flow, telecom networks, automatic circuit design

Goals for PhD

- A mixture of applications and theory in the realm of reinforcement learning/deep reinforcement learning
- Working on how to reduce the dependence on the size of the action space
- Hierarchichal learning - how to group actions together to form a hierarchy of decisions/goals