Deeper Convolution Neural Network for Poker

In the last few years, deep learning networks have made tremendous progress. Deep learning networks has beaten the top players in complex games of GO, Chess and Checkers. We chose to do our project on Poker as the game of Poker is more challenging due to the existence of randomness and hidden information, where the opponent's card is not known to the players.

Literature Survey

Poker is easy to learn and play, but hard to master. Players need to have strategic and logical thinking and adept analysis in order to play it well. The deep learning model used by the authors of DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker[1] claim to have beaten professional poker players. The neural network model used in the paper is simply 7 fully connected layers, and there is not much discussion on the effectiveness of different network architectures on this task.

Motivation

After learning the neural network used in DeepStack in detail, we would like to explore further possibilities as we believe that a more complex architecture may be more effective. In particular, Recurrent Neural Network (RNN) may be a better architecture here because it can memorize previous decisions made and also possibly take the opponent's move into account instead of just the table scenario as the input. Furthermore, DeepStack has sparse lookahead trees which only make use of a small subset of the game's possible actions. We want to explore expanding this subset to include more of the possible actions to see if performance improves.

Problem Definition/ Implementation

We aim to train a model that can play Poker well, hopefully being able to beat all the matches against the DeepStack model in the literature survey aforementioned. Similar to the paper, we will train our model by playing a large number (millions) of randomly generated Poker games. We will randomly generate the pot size, cards and public card, and the player ranges, which will be the input to the model. The output from the connected hidden layers (or whichever architecture we find to be best) will be pre-processed such that the values would satisfy the constraint (zero-sum constraint), after which, the value would be mapped back into a vector of counterfactual values, which would finally give the output. We would take into account all the possible actions (call, raise, bet, check, fold) from all the players. We will train the model with Adam Optimizer with a Huber Loss Function. If successful and time allows, we will also try to train it in reinforcement learning style that learn during playing. This is because we believe best decision shall be made not only based on current situation, but also depending on opponents move and style.

In order to perform recursive player, we will use Counterfactual Regret Minimisation (CRM)[2]. We will iteratively maintain a regret value for each action to define a strategy for each of the players and the regret values would be updated accordingly. The expected value to p at node h assuming that the players play according to the predicted strategy profile, σ, from that point is formally denoted as $v^{\sigma}_{p}(h)$.

Evaluation

To evaluate our deep learning model on poker, we will use our Deeper Convolution Neural Network (DCNN) model as well as Recurrent Neural Network (RNN) to play against the deepstack model for a

number of random games. Win ratio will be used to judge the performance. A satisfactory outcome is defined as our DCNN being able to win 50% of all the matches against DeepStack model while a good outcome will be defined as our DCNN being able to win 75% of all the matches against DeepStack model, and lastly an excellent outcome would be when our DCNN can will 100% of matches against the DeepStack model.

Dataset

We will randomly generate various Poker situations. It is easy to generate these situations and we do not need to find a dataset from other sources, as Poker inputs can be easily created.

Experiment Setup

We will be using Google Cloud Platform (GCP) - https://cloud.google.com/compute as it is easy to use, requires less setup and they give free credits. Most importantly, GCP allows easy deployment of Deep Learning VM with pre-packaged pre-configured OS and software needed for Deep Learning - https://console.cloud.google.com/marketplace/details/ click-to-deploy-images/deeplearning?pli=1.

Hardware Setup

We will be using a VM with NVIDIAR TeslaR K80: nvidia-tesla-K80. We have chosen a GPU accelerated VM as our compute workload (neural network) can benefit from GPU (Nvidia) acceleration. Moreover, K80 images is generally available and cost effective compared to other Nvidia accelerated visual machines (V100, etc...). We generally do not need the extra speed in our experiments.

Schedule / Role Assignment

Timeline	Part	Assignment
Week 8 - 9	1. Create the code/ rule for Poker in Python, 2. Replicate the implementation of DeepStack, 3. Create the model for Deeper Neural Network	1. Wu Biao 2. Hyma, Ying Kiat 3. Wei Qing
Week 10	● Observe the result after adding many hidden layers to DeepStack ● Try out different Deep Learning Architectures and note down the result	● All
Week 11	● Play our model against DeepStack and evaluate the results to compare performance	● All
Week 12	● Prepare for presentation	● All

References

[1] Matej Moravc, Martin Schmid, Neil Burch, Viliam Lisy, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, Michael Bowling, 3 March 2017. DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker, arXiv:1701.01724v3. https://arxiv.org/pdf/1701.01724.pdf

[2] Noam Brown, Adam Lerer, Sam Gross, Tuomas Sandholm (13 Nov 2018). Deep Counterfactual Regret Minimisation, arXiv:1811.00164v2. https://arxiv.org/pdf/1811.00164.pdf

Group Members (Group 32)
Wu Biao, Hyma, Ying Kiat, Wei Qing