TDL Q-Learning in Connect 4 with Structured and Deep Neural Network

Anupama Anupama
anan2946@colorado.edu

Harshini Muthukrishnan
hamu4143@colorado.edu

Laxmi Naga Santosh Attaluri
laat0836@colorado.edu

Prashil Bhimani
prbh9460@colorado.edu

Abstract

It is still a very difficult task to learn complex game functions. We are applying Temporal Difference Learning along with Deep Q-Learning which is a well known variant of the reinforcement learning approach.We have used both Deep Neural Network as well as structured neural network for developing the model. Our model is trained by strategic, random, minimax of depth 5, montecarlo player as well as self play. The models which were trained on strategic and random player performed very well. We believe that the architecture chosen for the model development is the key ingredient for the overall success and high quality results.

1 Introduction

Every child plays a lot of games, be it board games, atari games, outdoor sports or video games. These games are considered as a very important part of a child's mental development as there is not a fixed strategy followed in these games but more on a reward based system where every move gives certain rewards in terms of points, scores or just more similarity to the end goal. More and more they play the game, the better they become at it. This type of learning is basically known as reinforcement learning.

With the advancement in AI field where scientists are trying to mimic a human brain it was obvious that reinforcement learning has gained traction. Reinforcement learning is learning to analyze a current state and take an action that maximizes a future reward, through continuous interaction. That is, the model takes different actions and awards itself when it does something well. In this type of learning , the model tries to improve by retro-feeding itself i.e, model is trained by self play. The goal is to maximize the reward over the lifetime of the task of the agent or model.

Connect Four is one of the games that is widely played by children and hence we are planning to train a bot to play Connect Four game using reinforced learning.

2 TDL Q-Learning

It is evident that Q learning is an effective variant of reinforcement learning for modeling a game environment. However it becomes troublesome if the state space is huge. The connect 4 game has gigantic state space, a standard 6x7 board has 4,531,985,219,092 possible positions. Thus creating a Q- Table would be very inefficient, instead we can use Neural Network which mimics Q table (a mix of deep learning and reinforcement learning). In general, the reward is directly proportional to the score of the game. Imagine a situation wherein the Player 1 can win by placing the disk in the 7th column. The expected future reward for placing the disk at the correct position will be higher when

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

compared to the reward associated with any other move. In this case, the loss is a measure of how far the prediction is from the actual goal. The motive is to decrease the gap between the prediction and the target i.e., loss. The loss function can be defined as

$$Loss = 1/2\,[R + \gamma.\max_{a`} Q(s`, a`) - Q(s, a)]^2$$

We first carry out an action a, resulting new state s`. We push the state and action into the memory (stack). At the end of the episode, based on the reward obtained, the previous actions are given the discounted reward i.e., the recent actions are given higher rewards whereas the initial actions are given lesser rewards(discounted rewards) which is the temporal difference learning. Squaring this value allows us to penalize the large loss value more and treat the negative values same as the positive values.

3 Architecture Proposed

We have used Deep Neural Network as well as Structural NN to implement the Connect 4 game using reinforcement learning. Later we are comparing both the architectures w.r.t. the parameters like number of features, training speed, accuracy etc.

3.1 Structured Neural Network Architecture

Our model takes in 49 input nodes corresponding to 42 inputs and 7 possible action states. The input layer is sparsely connected to the first hidden layer which consists of 69 neurons. The next layer is a densely connected hidden layer of 20 neurons which is then finally connected to output layer which consist of a single neuron. This output neuron is a single value approximation of the Q function. We have used tanh activation function as its value ranges between -1 and 1. If we use sigmoid function, then we cannot have negative values which would not be suitable for this game. Relu is also ineffective for the same reason. Also, Relu activation function would heavily penalize since it is a linear function when the input is positive. We have chosen Adam optimizer for this architecture as it realizes the benefits of both Adaptive Gradient and Root Mean Square Propagation (RMSProp). Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.This optimizer calculates an exponential moving average of the gradient and the squared gradient. The parameters beta1 and beta2 control the decay rates of these moving averages. Adam helps in achieving good results faster. The tuning parameter values used for the Adam optimizer are alpha=0.001, beta1=0.9, beta2=0.999, epsilon=10e-8. Adam is a replacement optimization algorithm for stochastic gradient descent for training deep neural network learning models. We are using L2 regularization methods as it is computationally efficient due to having analytical solutions. The initial weights are randomly set by the random uniform distribution.

3.2 Deep Neural Network Architecture

Similar to the structured neural network proposed above, deep NN model takes in 49 input nodes corresponding to 42 inputs and 7 possible action states.It consists of 3 hidden layers. The output layer consist of a single neuron which approximates the Q function. We have used same activation function, optimization function and regularization which was used in the Structured neural network. However dense neural networks are prone to over-fitting when compared to structured neural network. So we have used dropout layers with the dropping rate of 0.3. The deep neural network consists of more than 13000 features whereas the structured neural network has around 1200 features which is almost one tenth of the number of features in the dense neural network.

4 Training and Testing Models Used

We explored a set of players to train our Neural Network models and test against it. All the models we developed had their own advantages and disadvantages which resulted in a better understanding of the Neural Network model we built. The following are the various models used as part of the training and testing process.

4.1 Random Player

As the name suggests, this player was programmed to make random moves without any logic. This was designed to be a baseline model against which we would compare, train and test against.

4.2 Strategic Player

In this player we had encoded the human nature of playing the game. The strategy is as follows:

- By placing the move in a particular column, find if the current player can win. If so, place the winning move and finish the game.
- Else, find a particular column in which if the opponent places the move in the next turn then he/ she will win. If found, block the position so that the opponent cannot win in the next possible iteration.
- Otherwise, find a longest possible combination you can make by placing in a particular column. At first, if multiple columns gave the same highest streak, the last one found was chosen. When plotting this particular strategy, we found that the columns started filling from the rightmost end. So a random column was chosen thus randomizing the moves in case of highest streak collisions.

4.3 MiniMax Player

The minimax is supposed to be the best 2 player game algorithm where the players play alternatively. This works on the principle of minimizing the potential loss for the worst case (maximum loss) scenario. When dealing with gains, it is referred to as "maximin"—to maximize the minimum gain. The heuristic in minimax is used to evaluate the relative "goodness" of a board configuration. The heuristic is as follows:

$$Score = (myFourInARows) * 100000 + (myThreeInARows) * 100 + (myTwoInARows)$$

If the opponent has one or more four in a rows, then

$$Score = -100000$$

since such a condition would lead to the opponent's win.

But for a game like connect4, the state space increases exponentially as the depth of the search tree increases. In order to tackle the problem of exponential growth, we had to tweak the minimax player. The minimax player, when implemented, would result in the highest accuracy. In order for the model to not over-fit, we have to include some randomness in the player. Also the time taken to build the player by training the model would be really high. These combined reasons led us to implementing the player for only up-to a depth of 5.

4.4 Monte Carlo Player

It is a computation process that uses random numbers to produce an outcome(s). Instead of having fixed inputs, probability distributions are assigned to some or all of the inputs. This will generate a probability distribution for the output after the simulation is run. It combines the precision of tree search with the generality of random sampling. It does not depend on heuristic function to evaluate the best next move to make, rather it just considers the game mechanics to play random roll-outs and get an expected reward after a fixed number of iterations. For every potential move the computer can do, it does it and then plays random games, sampling the number of games won and lost. Then the decision on what move to make is made based on the move that appears to have the best won/lost ratio. Monte Carlo was built thinking it would be a better player than the strategic player since it is based on a rigid algorithmic logic. But once we trained and tested, we realized that it performed worse than the random player. This is because Monte Carlo method is episodic and since connect 4 isn't, the model performed poorly for training.

4.5 Self Play

In this, the model was made to play against itself. We invert the game state and both work on the same model. As the model gets better, the chances of either of the players winning becomes slight. We expect most of the matches to get to a draw which actually did.

4.6 Human Player

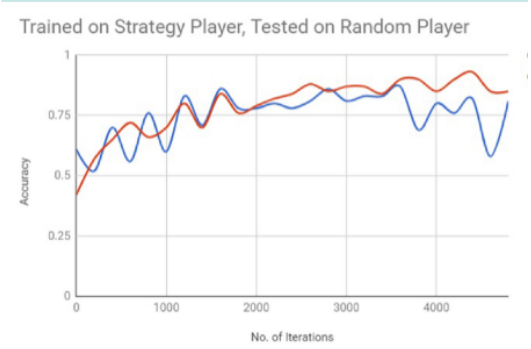
Finally, we tested our model by playing against it. Doing so, we were able to discern how the model interpreted our moves. This resulted in a better understanding of our model and helped with fine tuning the parameters.

5 Performance of the models

The number of features in a structured neural network were approximately 1700 whereas the numbers of features in the deep neural network were 17000.The accuracy of the model is 93% when trained on a strategic player and tested on random player.The accuracy of the model was also more than 90% when trained on random player and tested on strategic player.There is a reduction of 90% in the number of parameters but it does not affect the accuracy but the training time is reduced significantly. The models which were trained on montecarlo player did not perform well as it is episodic, not strategic.

Figure 1: Trained on Random Player(Structured vs Deep Neural Network)
(Red line represents Structured neural network, Blue represents Deep Neural Network)

Figure 2: Trained on Test Player(Structured vs Deep Neural Network)
(Red line represents Structured neural network, Blue represents Deep Neural Network)

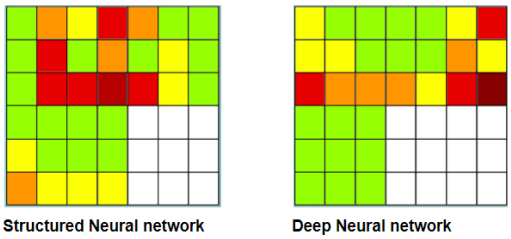


6 Analysis

Below two figures depicts heat maps of structured and deep neural network.The deep neural network model was over fitting at particular winning moves making the model more prone to new kinds of

players. The deep neural network model was fitting at the edges which are technically the least favorable positions. The structured neural network dealt with this problem in a better way.The structured neural network provided the winning moves at the center which are more favorable to winning the game.

Figure 3: Heat Map depicting the frequency of win at a certain location



7 Individual Contributions

- Anupama Anupama : Worked on the game architecture and initial modelling of Q learning algorithm for analysis. Helped in implementation of the structured neural network.
- Harshini Priya Muthukrishnan : Found and modified the limited depth Min-Max player to play against modelled. Modified the Strategic player sent by Chris Ketelson.
- Laxmi Naga Santosh Attaluri : Worked on testing and comparing models and found ways to improve the model by identifying the ways model was overfitting the data. Developed the Monte Carlo Player to test against player. Researched about optimizers and losses which will be appropriate for the model.
- Prashil Bhimani : Worked on the game architecture and modelling for Q learning algorithm. Implemented the dense and structured neural network. Developed the self play and Random algorithm for the model to play against itself. Worked on improving the model and finding cases after overfitting was detected. Worked on testing and comparing the models.

8 Results

Structured neural network take less time to train as compared to the deep neural network.However, same accuracy can be achieved with the structured and deep neural network, if the required features to be used as input are well known in advance.

References

[1]Michiel Van De Steeg, Madalina M. Drugan and Marco Wiering Temporal Difference Learning for the Game Tic-Tac-Toe 3D: Applying Structure to Neural Networks

[2]Michiel Van De Steeg, Madalina M. Drugan and Marco Wiering Temporal Difference Learning for the Game Tic-Tac-Toe 3D: Applying Structure to Neural Networks

[3] V. Allis. A knowledge-based approach of Connect-4. The game is solved: White wins. Master's thesis, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, 1988.

[4] D. Curran and C. O'Riordan. Evolving Connect-4 playing neural networks using cultural learning. NUIG-IT-081204, National University of Ireland, Galway, 2004.

[5] K. Krawiec and M. G. Szubert. Learning n-tuple networks for Othello by coevolutionary gradient search. In Proc. GECCO'2011, Dublin, pages 355–362. ACM,New York, 2011