Machine Learning Engineer Nanodegree

Capstone Project

Joseph Friedman

I. Definition

Project Overview

The goal of this project is to create an AI agent capable of playing and winning at the game, 3D Tic Tac Toe. It is played by two players on a 3x3x3 grid. Players alternate turns; choosing one previously unchosen space on each turn. The winner is the first player to mark three spaces in a “row” in any direction. This project is similar to, but in no way based on, M. van de Steeg, M.M. Drugan and M.A. Wiering Temporal Difference Learning for the Game Tic-Tac-Toe 3D: Applying Structure to Neural Networks . IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (IEEE ADPRL'15), 2015.(http://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/TTT3D_FINAL.pdf)

Problem Statement

In order to achieve my goal of training a neural network to play and win at 3 dimensional tic tac toe i will explore two different types of neural networks:

- Deep Q Learning
- Deep Deterministic Policy Gradient

I will also explore several approaches to training. I will consider training with a environment that provides reward for each move taken as well as an environment that only gives a reward other than 0 upon winning (or losing.) Another aspect that I will be exploring is what the agent will be playing against during training. It can play against:

- Random actions
- Manual programed
- Adversarial Networks

The goal of this project is not necessarily to compare the combinations of these various approaches, rather it is to train a neural network to play and win. If, in the course of solving this problem, data is obtained to compare these techniques, that will be a welcome bonus.

Metrics

The Neural Network will be measured by :

1. Immediate Completions -- did the agent immediately place a third in a row when it already has two
2. Immediate Blocks -- did the agent block when its opponent has two in a row
3. Wins - did the agent win the game.

These metrics were chosen because they are indicative of the NN learning the skills needed to successfully play the game

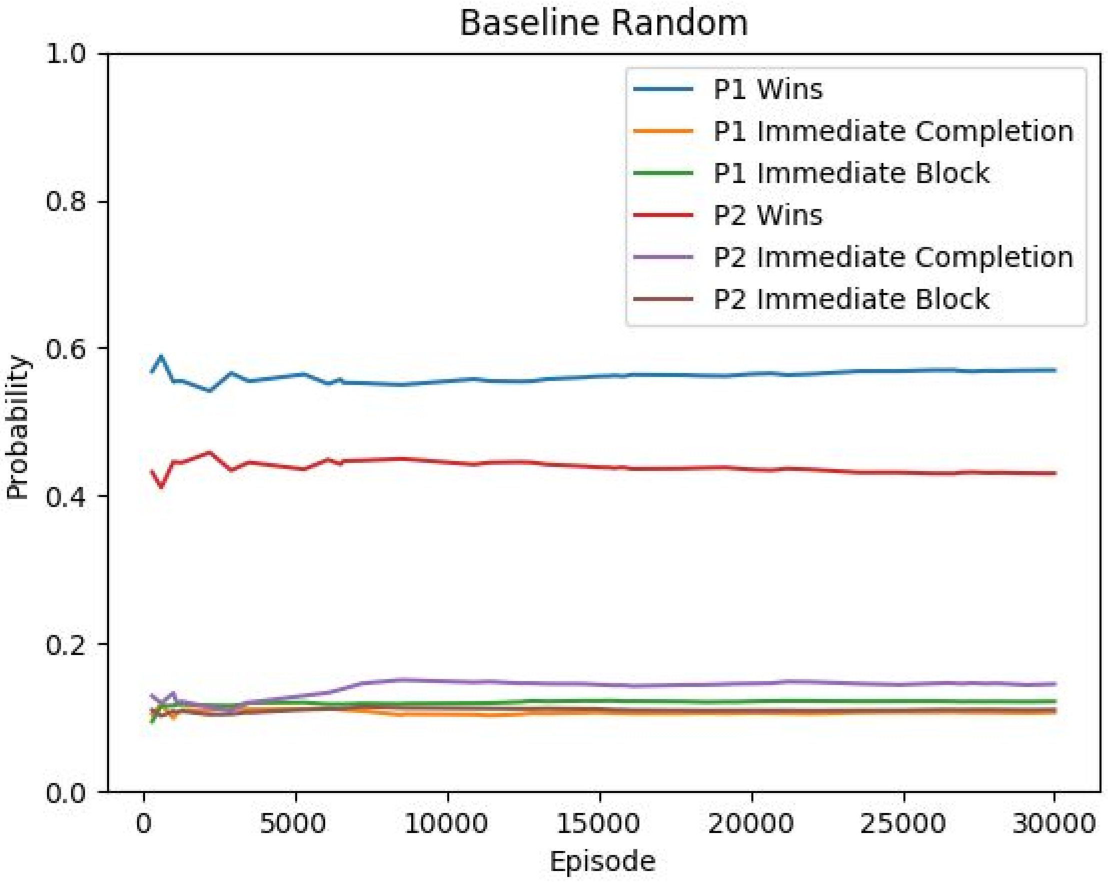
II. Analysis

(approx. 2-4 pages)

Data Exploration

Our environment consists of 27 spaces each of which can be in one of three states, empty, player 1 , or player 2; traditionally these are marked in the US as X and O. There are, potentially, two ways to represent the spaces in the environment, Individually labeled 0-26, or a 3D matrix of coordinates. For simplicities sake I will use a 1D vector of 27 values. The advantage of using coordinates is that it may possible to train a neural network (specifically a deep deterministic policy gradient) that can generalize to other sized games such as 5x5x5. Since that isn't the goal of this project I will start with (one-hot encoding) the spaces labeled as 0 thru 26. There are 49 winning combinations, or "rows."

Exploratory Visualization



This visualization demonstrates a baseline of two random players playing. You can see that player one has an advantage and wins about 57% of the time. Of note, is the lack of precision when the sample size is less than 7500 episodes. As we will later see, one of the challenges in implementing this algorithm is the occurrences of the NN not learning or descending into local minima. At first this was almost 80% of the time and with improvement to the explore/exploit ratio it was reduced to 15%. It appears to me that analysis of this graph could help improve this further as it appears that if during training, random moves are taken for at least 7500 episodes the NN would learn closer to 100% of the time. (Subsequent testing revealed that this is not the case, perhaps increasing the batch size of the experience replay would help.)

Algorithms and Techniques

The two algorithms I intend to explore are:

- Deep Q Network (DQN)
- Deep Deterministic Policy Gradient (DDPG)

I prefer using an environment which only provides a reward at the end of the game. I want the algorithm to learn policies for the game that may be superior to a human player, and as such I want no human (reward) feedback for each move.

The DQN can handle much larger state space than would be reasonable with a traditional Q value table. This project certainly requires that as its state space is very large .

DDPG is an extension of DQN in order to accommodate continuous space states which uses a second NN (called the critic network) to estimate Q values and uses this estimate to update the weights on the first network (called the actor network). This algorithm was developed to solve reinforcement learning problems involving continuous state spaces (real world 3d space) because a continuous state space is virtually infinite and could not be accommodated by a DQN. Both algorithms run into similar problems and I will be using the same techniques to overcome them.

The first issue is, that sequential states are very similar and if trained in order the NN would descend into local minima and never converge. We use a technique call Experience Replay in which we store the states in a buffer and draw random batches from the buffer for training. This allows the network to train on diverse states and converge.

The other issue during training, common to all Q learning, is known as the Explore - Exploit dilemma. This refers to the question of whether the network should try completely new states or future states of previously attempted states. The technique to overcome this is known as ϵ greedy policy in which we select a random move with ϵ probability and with $1-\varepsilon$ probability select the highest Q value already learned for this state.

Benchmark

The RL agent will be benchmarked against four opponents.

1. Random moves
2. A relatively shallow policy gradient network
3. A deeper policy gradient network
4. The deeper policy gradient network with random moves

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

There is no preprocessing necessary, other than one hot encoding. The inputs are simply a list of 27 numbers, representing the 27 spaces, with either a 0, for empty, 1 for player 1, and 2 for player 2. When using the NN for the second player, the environment will return the state and the reward for the second player.

Implementation and Refinement

The first step was to create the environment for 3d TicTacToe. I initially attempted to create an OpenAI Gym Environment but I was hampered by the lack of documentation and was unable to complete it compatible with openai. The game and benchmarking code was improved on throughout the project. I adapted a DQN in tensorflow and trained it as player 1 against a player doing random moves and the initial results were surprising. I the NN would lose >80% of the time!! I then carefully re implemented the the DQN only to find the same result. I then drastically lowered the future reward discount factor, γ, and trained for 100,000 episodes. This showed improvement, to about 60% wins , and I retrained it for 1,000,000 episodes, and achieve 95% wins!! The skill that this NN learned is to put three in a row. This exercise does serve two purposes. It proves that the algorithm is correctly implemented and the trained network is will be the opponent of the NN that will learn to block. While I was able to implement a DQN that defeated an opponent playing random moves greater than 85% of the time, the NN always completed 2 in a row several times before finally winning. This NN proved ineffective as an opponent to train another network to block. I believed that the reason is as follows. The state space is too large and would require way more training than would be feasible for me at this time. In addition, the explore/exploit algorithm should be modified to skew the likelihood of exploit earlier on in each episode to improve results. At this point I felt confident that the state space is large enough to justify using the DDPG algorithm. What I did not realize at the time was that a **two layer** network was insufficient to handle the state space involved but it is likely that a deeper network would be able to.

I then began to implement a (relatively shallow) DDPG algorithm and immediately saw a vast improvement. After running several experiments I began to realize that the network, more often than not, would get stuck in local minima and not learn at all. In these instances its performance would be worse than random. In the 10-20% of the time it did learn at all, it learned very well (>90%).

I understood that the explore/exploit decay that I had been using was inappropriate for the 3d tic tac toe game. The multistep nature of the game requires that earlier steps begin to have more exploit and less exploring while the later steps still require more exploration. In order to accomplish this I used a new (as far as I know) technique which I called “Noisy Max-Q”. During training, the agent would (roughly) scale the Q values for the current state into probabilities and select an action based on these probabilities. My expectation was that as the network gained confidence in the (earlier) steps it would progressively gain confidence in the later steps of the useful early steps and not waste time exploring later steps of bad early states. This greatly reduced the occurrences of the network being stuck at local minima and not learning (from almost 90% to <15%.)
While even the shallowest DQN showed some learning and improvement in the winning benchmark; most learning required a deeper network.

When the earlier algorithms showed significant learning on rare occasions it prompted me to more clearly define success for this project. The loosest definition of success for this project would include an algorithm that could learn even on rare occasions. Being a reinforcement learning problem, we have the luxury of retraining as many times as necessary to produce an agent that can effectively play the game. I suppose this could be said, to some extent, to other domains as well. At the other extreme, success would be defined as an algorithm that always trains effectively. In the end I decided to take a balanced approach and improve the NN until it was more likely to effectively train and had increased effectiveness.

IV. Results

(approx. 2-3 pages)

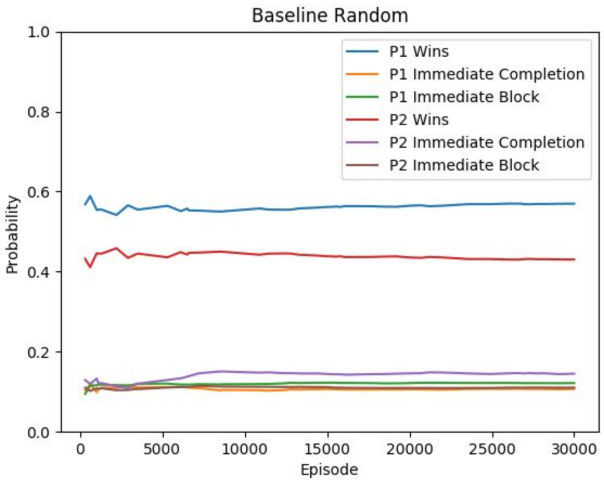
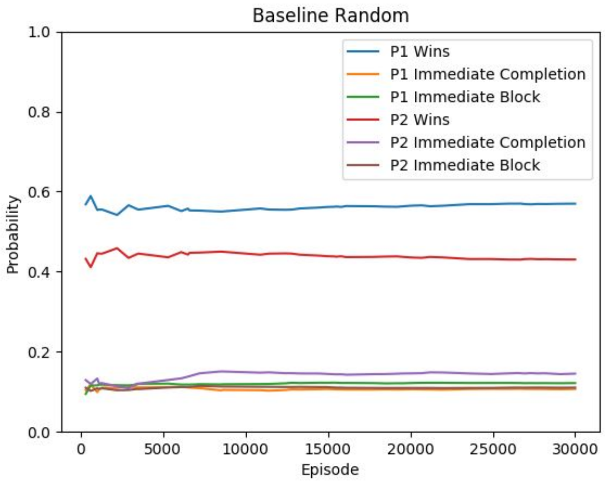
Model Evaluation ,Validation and Justification

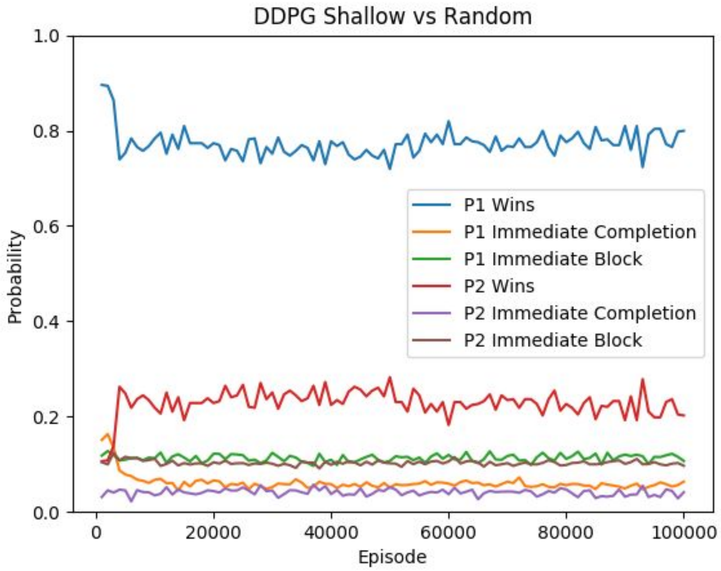
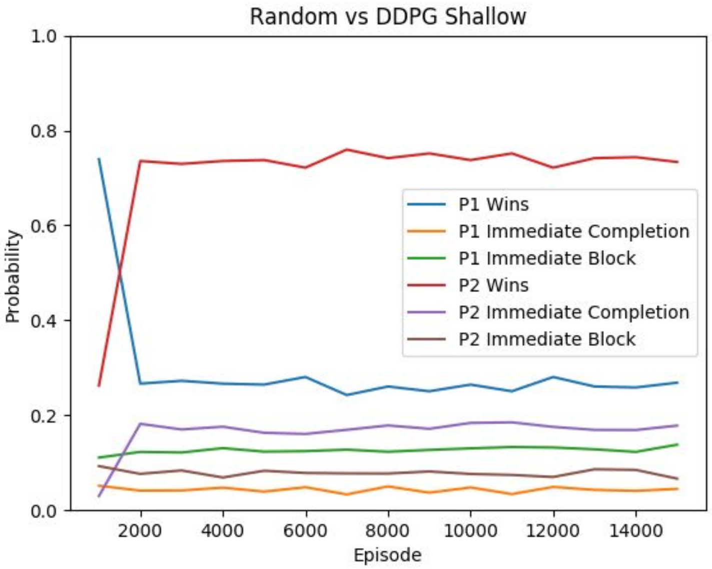
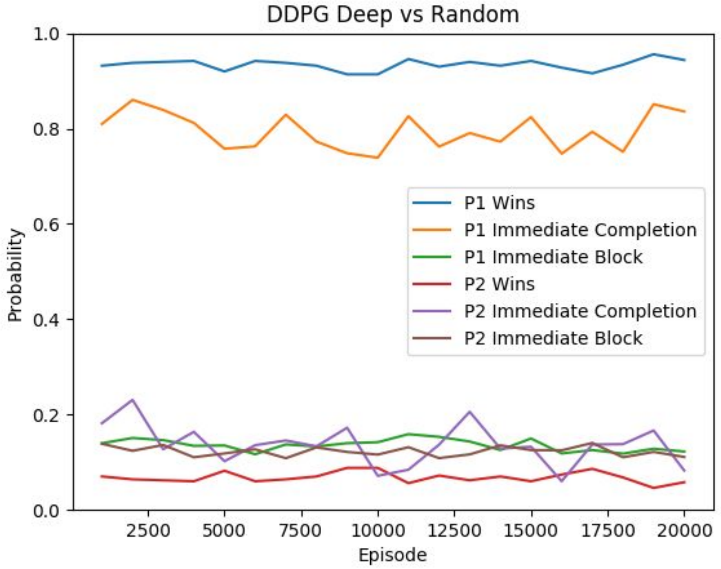
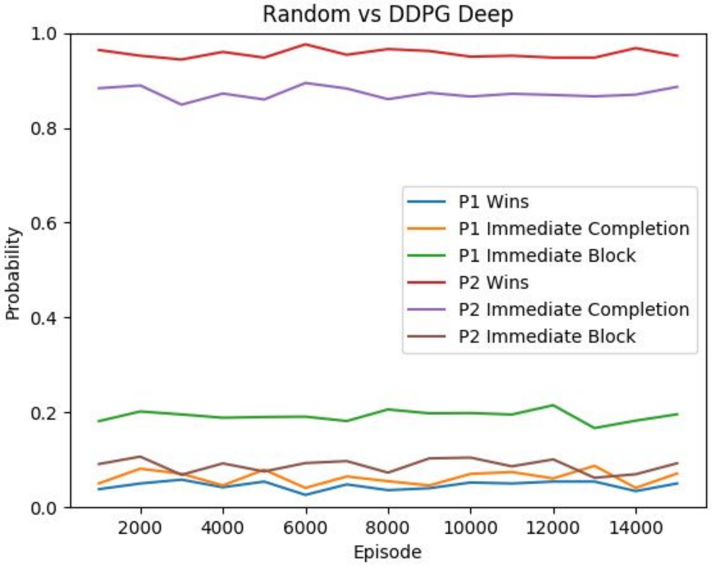
The final model, based on the Deep Deterministic Policy Gradient algorithm, with a 10 layer Actor and 10 layer Critic network successfully solves the problem of playing 3d Tic Tac Toe. The training is effective ~90% of the time and learns to win between 80-98% of the time. Some improvement above random is seen in immediate completions and blocking. In rare cases, the combination of early states and actions can lead to the network being “stuck” in a local minima. The model successfully converges 80% of the time, the model converges and shows improvement in all three metrics when compared to baseline against a random player, an adversarial network and most importantly an trained network with random moves mixed in. This would indicate that the model generalizes well against unseen “data” (states.) The explore exploit technique used greatly improved the robustness of the architecture from 90% not converging to 20% not converging. In the context of reinforcement learning I believe this is acceptable.

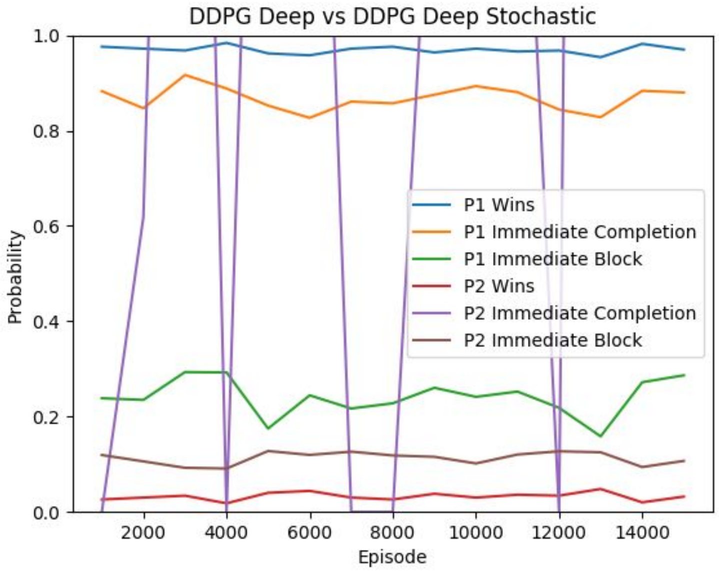
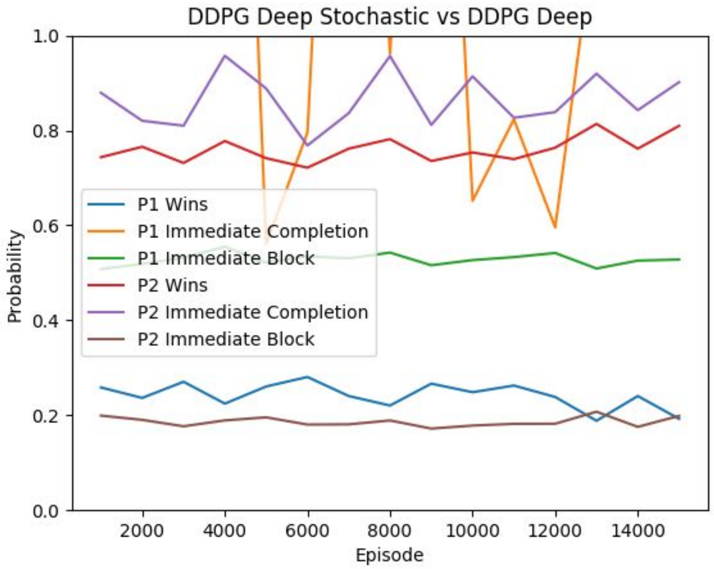
V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

	Player 1	Player 2
Baseline	 Baseline Random 	 Baseline Random

DDPG Shallow		
DDPG Deep		

DDPG Generalizing		

These plots demonstrate the learning that has been performed as player 1 and player 2. The rows are as follows:

1. Baseline - The results of both players making random moves
2. DDPG Shallow / Random- a shallow NN vs a random player and vice versa
3. DDPG Deep / Random - a deep NN vs a random player and vice versa
4. DDPG Deep vs DDPG Deep Generalizing - DDPG Deep vs DDPG Deep w 50% random moves and vice versa

Reflection

This project was an amazing experience. Over the course of the nanodegree (s) I began to wonder if the skills we practiced in the structured projects would 'generalize’ to new problems. Applying the skills learned in this nanodegree as well as the DLND to a new problem demonstrates to me, the value of what we learned.

I chose a reinforcement learning problem simply because it was a type of deep neural network that I had not done in the DLND. At the time, the DDPG variation of deep q learning was the latest and greatest deep rl algorithm. I didn't need to implement it from scratch, directly from the white paper, as someone had already implemented it in TFLayers, but I did reimplement it in vanilla tensorflow. The example implementation was for the cart-pole environment in openAI, a relatively simple Atari game. Adapting it to fit 3d tic tac toe involved changing two aspects, the depth and width of the network and most importantly the explore/exploit decay. I'm most proud of this innovation. The original examples used an exponentially decreasing explore rate. This works in the context of the cart pole game with a small action space and almost immediate feedback. The previous actions make no difference at each point. In 3d tic tac toe, the previous steps are entirely relevant. Therefore it is necessary to increase the exploit of the earlier moves whilst still exploring the later moves. The training getting stuck in local minima most of the time further demonstrated the need for a less simple explore decay rate. My solution was to (roughly) use the q values as probabilities, assuming that q values for unexplored states would be random and as the q values were updated, the best early steps would be “exploited” and the later states would still be random. (I am referring to the q values as if there is a q table even though that's not actually the case.)

Improvement

One aspect that can be improved upon is the representation of the state space as a one hot encoded list. Alternatively, the state space could be represented as cartesian coordinates. This could enable the this neural network to generalize to larger sized game boards such as 5x5x5x5x5x5. Failing that, it could improve accuracy. As is, it would seem that making the NN deeper would improve learning. A more important improvement would be to adjust the network to improve the likelihood of convergence, currently at 80%.

Much to my dismay, I did not read the paper cited above prior to completing this project. Two things of note are their representation of state and benchmark player. They did represent the states as a 3D matrix as opposed to a 1D vector. While I consider this superior from a puritan perspective, I do think we should strive to solve problems with the simplest solution. Their benchmarking technique, using a simply programmed opponent which “incorporates the rule-based knowledge of not making stupid mistakes” they were able to use a fourth and more significant metric of seeing if the NN could play a “double trap” to force a win. I do believe that using this opponent also avoids the problem of local minima. I am, nevertheless satisfied with my project and it explore/exploit technique.