Machine Learning Nanodegree
Capstone Proposal

David A. Robles

December 13, 2016

1 Domain Background

Reinforcement learning is the area of machine learning concerned with the idea of learning by interacting with an environment (Sutton and Barto, 1998). It has a long and notable history in games. The checkers program written by Samuel (1959) was the first remarkable application of temporal difference learning, and also the first significant learning program of any kind. It had the principles of temporal difference learning decades before it was described and analyzed. However, it was another game where reinforcement learning reached its first big success, when TD-Gammon (Tesauro, 1995) reached world-class gameplay in Backgammon by training a neural network-based evaluation function through self-play.

Deep Learning (LeCun et al., 2015) is another branch of machine learning that allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. Deep learning techniques have dramatically improved the state-of-the-art in areas such as speech recognition (Hinton et al., 2012), image recognition (Krizhevsky et al., 2012) and natural language processing (Colbert et al., 2011).

Reecently, there have been several breakthroughs when combining reinforcement learning and deep learning. Mni et al. (2015) used a convolutional neural network trained with a variant of Q-learning and learned to play Atari 2600 games at a human-level. This year, one of the biggest challenges for artificial intelligence was solved, when Google's DeepMind (Silver et al., 2016) created AlphaGo to beat the world's greatest Go player. AlphaGo used deep neural networks trained by a combination of supervised learning from human expert games, and reinforcement learning from games of self-play.

Developing strong game-playing programs for classic two-player games (e.g. Chess, Checkers, Go) is important in two respects: first, for humans that play the games looking for an intellectual challenge, and second, for AI researchers that use the games as testbeds for artificial intelligence. In both cases, the task for game programmers and AI researchers of writing strong game AI is a hard and tedious task that requires hours of trial and error adjustments and human expertise. Therefore, when a strong game-playing algorithm is created to play a specific game, it is rarely useful for creating an algorithm to play another game, since the domain knowledge is non-transferable. For this reason, there is enormous value in using machine learning to learn to play these games without using any domain knowledge.

1.1 Motivation

My personal motivation for investigating this area more in depth is that we have plans for developing mobile games that require strong AI. So far we have developed one mobile game, Tic Tac Toe Clash[1,2], but Tic Tac Toe is such a simple game that does not even require an evaluation function, since is possible to search the full game tree using Alpha-Beta pruning to find the best possible move. But we are in the process of developing more games, and some of them will require stronger AI, and hand-coding different heuristics for

[1] https://itunes.apple.com/us/app/tic-tac-toe-clash/id1041863034?mt=8
[2] https://play.google.com/store/apps/details?id=com.zerosumstudios.tictactoeclash

each particular game is a hard and tedious task. So, my personal goal is to use this project as a first step to create a generic reinforcement learning framework that uses deep learning to learn evaluation functions for abstract strategy games without using any domain knowledge.

2 Problem Statement

In this project, we will use reinforcement learning with deep learning to make an agent learn to play the game of Connect 4[3] by playing games against itself. In other words, using the formalism used by Mitchell (1997) to define a machine learning problem:

- **Task:** Playing Connect 4.
- **Performance:** Percent of games won against other agents, and accuracy of the predictions on a Connect 4 dataset.
- **Experience:** Games played against itself.
- **Target function:** $Q^{\pi} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, where $\mathcal{S}$ is the set of *states* (board positions) and $\mathcal{A}$ is the set of *actions* (moves), and $\mathbb{R}$ represents the value of being in a state $s \in \mathcal{S}$, applying a action $a \in \mathcal{A}$, and following policy π thereafter.
- **Target function representation:** Deep neural network.

3 Datasets and Inputs

This project will not use a dataset as part of the training, since an agent will use reinforcement learning to learn to play Connect 4 by playing games against itself. To generate the episodes a Connect 4 environment written in Python will be used. This environment is explained more in detail in Section 7.

However, as part of the testing phase, we will use the *Connect 4 Data Set*[4] that is available from the UCI Machine Learning Repository (Hettich and Merz, 1998). This dataset has a total of 67,557 instances, representing all legal 8-ply positions in Connect 4 in which neither player has won yet, and which the next move is not forced. Each instance is described by 42 features, one for each space in the 6×7 board, and can belong to one of the classes $\{\text{x}, \text{o}, \text{b}\}$, where x is the first player, o is the second player, and b is empty. The outcome class is the game theoretical value for the first player, and can belong to one of the classes $\{\text{Win}, \text{Loss}, \text{Draw}\}$. There are 44,473 wins, 16,635 losses and 6,449 draws. In Section 5 we discuss how this dataset will be used as a benchmark model.

4 Solution Statement

To tackle the problem described in Section 2, we will use Reinforcement learning with Deep Learning to automatically learn evaluation functions by playing games against itself. Unlike other approaches that need a very large dataset, this approach will try to learn to play games without any domain knowledge (no dataset will be used). This is a promising approach for creating game-playing algorithms for playing other two-player games of perfect information.

[3] https://en.wikipedia.org/wiki/Connect_Four
[4] https://archive.ics.uci.edu/ml/datasets/Connect-4

5 Benchmark Model

- **Random agent**. This benchmark consists in playing against an agent that takes uniformly random moves. This is the most basic benchmark, but first we have to be sure that our learned evaluation function can play better than a random agent before moving into a harder benchmark. Also, this will help us to detect bugs in the code and algorithms: if a learned value function does not play significantly better than a random agent, is not learning. The idea is to test against this benchmark using Alpha-beta pruning at 1, 2 and 4-ply search.
- **Connect 4 Data Set**. This dataset will be used as the main benchmark. The learned value function will be used to predict the game-theoretic outcomes (win, loss or draw) for the first player in the 67,557 instances of the dataset.

6 Evaluation Metrics

- **Winning percentage.** This metric consists in playing a high number of games (e.g. 100,000) against another agent (e.g. a random agent), and calculating the average of games won by the agent that uses the learned value function.
- **Prediction accuracy.** The learned value function will be used to predict the game-theoretic outcomes (win, loss or draw) of the board positions in the Connect 4 Data Set.

7 Project Design

7.1 Programming Language and Libraries

- **Python 2.**
- **scikit-learn**. Open source machine learning library for Python.
- **Keras.** Open source neural network library written in Python. It is capable of running on top of either Tensorflow or Theano.
- **TensorFlow.** Open source software libraries for deep learning.

7.2 Environments and Agents

This project will use two environments:

- **Tic Tac Toe.** Will be used to verify that the reinforcement learning algorithms are working as expected, since it is a simple game where an agent can easily search the full game tree to find the optimal actions.
- **Connect 4.** The main environment for which we will learn value functions using reinforcement learning with deep neural networks.

Both environments will follow a common Game interface:

```
class Game(object):
    def copy(self):
        '''Returns a copy of the game'''
    def getBoard(self):
        '''Returns the board of the game'''
```

```
    def getCurrentPlayer(self):
        '''Returns the player whose turn it is'''
    def getLegalMoves(self):
        '''Returns a list of legal moves for the player in turn'''
    def getOutcomes(self):
        '''Returns the outcome for each player at the end of the game'''
    def isGameOver(self):
        '''Returns true if the game is over, false otherwise'''
    def makeMove(self, move):
        '''Makes a move for the player whose turn it is'''
```

Also, we will use a simple Agent interface:

```
class Agent(object):
    def chooseMove(self, game):
        '''Returns one of the legal moves from game.getLegalMoves()'''
    def learn(self, game):
        '''
        Called at the end of each episode. It should use game.getBoard()
        and game.getOutcomes() to update the deep neural network
        '''
```

These interfaces will allow us to run multiple trials relatively easy:

```
n_trials = 100
agent = Agent()
for trial in range(n_trials):
    game = Connect4()
    while not game.isOver():
        chosenMove = agent.chooseMove(game)
        game.makeMove(chosenMove)
    agent.update(game)
```

7.3 Machine Learning Design

- **Type of training experience:** Games against self
- **Target function:** $Q^{\pi} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, where $\mathcal{S}$ is the set of *states* (board positions) and $\mathcal{A}$ is the set of *actions* (moves), and $\mathbb{R}$ represents the value of being in a state $s \in \mathcal{S}$, applying a action $a \in \mathcal{A}$, and following policy π thereafter.
- **Representation of learned function:** Deep neural network
- **Learning algorithm:** Q-learning, a model-free online off-policy algorithm, whose main strength is that it is able to compare the expected utility of the available actions without requiring a model of the environment:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

7.4 Additional Algorithms

- **Alpha-beta Pruning.** The most common game tree search algorithm for two-player games of perfect information. It will be used as part of the evaluation of the learned value functions.

References

R. Colbert et al. Natural language processing (almost) from scratch. In *Journal of Machine Learning Research*, volume 12, pages 2493–2537, 2011.

C. B. S. Hettich and C. Merz. UCI repository of machine learning databases, 1998.

G. Hinton et al. Deep neural networks for acoustic modeling in speech recognition. In *IEEE Signal Processing Magazine*, volume 29, pages 82–97, 2012.

A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. Advances in Neural Information Processing Systems*, volume 25, pages 1090–1098, 2012.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, pages 436–444, 2015.

T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

V. Mni et al. Human-level control through deep reinforcement learning. *Nature*, pages 529–533, 2015.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal*, 3(3):210–229, 1959.

D. Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3), 1995.