MLND Capstone Proposal

A Reinforcement Learning approach to Dots-and-Boxes

Matthew Deakos | mwoolnerdeakos@gmail.com

May 1st, 2017

1. Domain Background

Reinforcement learning is an area of machine learning that is concerned with how an agent can learn to act in some optimal way based on its interactions with the environment. This differs from other forms of machine learning in several ways. Namely, other forms of supervised learning require labeled data, which is fed into a machine learning algorithm to find a predictive model. Reinforcement learning, however, does away with explicit pairs of data, and instead simply allows the agent to collect information on its own. This means that correct and incorrect actions are not explicitly defined, and the agent must learn which is which.[1]

Game environments are well-suited to reinforcement learning algorithms because they have a clearly defined reward structure (victory and loss, or score). In fact, several new developments in reinforcement learning have come out of game playing agents. Most notably perhaps is AlphaGo, which was a Go bot that beat Lee Sedol This was a revolutionary accomplishment in reinforcement learning[2]. Other interesting developments include DeepMind's work on the Atari games, mastering most of them beyond the level of a human.[3]

1.1 Motivation

I have found reinforcement learning to be incredibly interesting because of the unique class of problems that is seeks to solve, and the remarkable advancements that have been made with it in the past few years. Watching AlphaGo beat Lee Sedol, in fact, was one of my primary drivers in pursuing machine learning at all. It was not clear to me until that moment how the tasks that we had always viewed as uniquely human would be quickly withered by the oncoming advancements in technology, and how much I wanted to play a part in that revolution. In that spirit, I wanted to build something of my own that would learn to play a game (and hopefully, to play better than me).

---

1 Reinforcement learning. (2017, April 15). Retrieved May 02, 2017, from https://en.wikipedia.org/wiki/Reinforcement_learning

2 Hassabis, Demis (). . , 529, 484-489. Article

3 Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013)

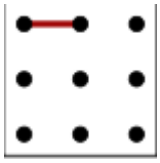
2. Problem Statement

This project seeks to use reinforcement learning and deep learning to allow an agent to learn a policy for the game 'Dots and Boxes'[4]. This game is simple in its rules, but it is strategically complex enough to warrant a machine learning approach to the problem. The model should learn to reliably win against the basic model, and other models will be provided for further evaluation. Details are outlined in section 5.

3. Dataset and Inputs

Due to the nature of the problem, no dataset is being used. In its place a game environment will be built and games will be simulated. For this project, we will use a 5 x 5 model (5 dots by 5 dots). A board of this size is complex enough to warrant an ML agent; the state space has been calculated at 2^40 possible boards, which is far more than can be reasonably explored. The action space is also quite high, with 40 possible action moves from the starting position.

The proposed input is a three dimensional array that is somewhat analogous to typical image representation. The first two dimensions will indicate a 'cell' and the third dimension will have one-hot encoded values indicating if a wall exists. For example, the first cell of a game below would be represented by the array [1, 0, 0, 0]. This translates to [North Wall, No South Wall, No East Wall, No West Wall]. It is my hypothesis that this input format will help keep local patterns contained, which may be capitalized on by a convolutional net. Games will be a series of these input states, and a reward will be granted on reaching the end of an episode. This reward will be in the set of {1, 0, -1}, corresponding to a win, draw, or loss.



4. Solution Statement

The proposed solution for this problem is a reinforcement learning agent that uses a convolutional neural net to approximate the value for a given state. This information can be gained by having the agent play many games against itself. The number of games necessary cannot be easily approximated before implementation, but the initial proposition is one million iterations. Then, when the agent has a good approximation of state values, the optimal policy is simply one which maximizes that Q-Value.

5. Benchmark Models

5.1 Random Agent

The most basic benchmark is a purely random player. This player will randomly take any legal moves during their turn. The learning agent should, at the very least, be able to beat the random agent a

4 "Dots and Boxes." Wikipedia. Wikimedia Foundation, 30 Apr. 2017. Web. 01 May 2017.

significant proportion of the time. If the win-to-loss ratio hovers around 1-1, we can be sure that our agent is not learning.

5.2 Naive Agents

Two rule-based agents will also be used to evaluate the performance. The first will be an agent that searches for a potential scoring move, and if one does not exist will move randomly. This is approximately how you would expect someone who just learned about the game to play.
The second will be more difficult. This agent will also search for scoring moves, but will also actively avoid making any moves that lead to a scoring move for the opponent. That is to say, this agent will avoid putting the third line on a box.

6. Evaluation Metrics

Win Rate
This statistic will be the proportion of the number of games that the agent can win against a test agent. A high number of games will be played to ensure accuracy (>10 thousand).

Loss Rate
This statistic measure the proportion of games that the agent loses against a test agent. This is necessary, because there is a third outcome for this game, so losses should be explicitly measured.

Draw Rate
This statistic measures the proportion of games that the agent plays against a test agent that end in a draw.

7. Project Design

7.1 Requirements

Python 3
Tensorflow 1.1 – An open source library for deep learning.
Numpy – A computational library.

7.2 Environment and Agent API

Shown below is the planned interface for the agents and the environment.

```
class Agent:
    """The basic agent class"""
    def act(self,state):
        """Act in the environment"""
    def observe(self,state,reward):
        """Observe the environment and collect a reward"""


class Environment:
    """Environment Class"""
    def step(self,action):
    """Take the action in the environment and allocate rewards"""
    def score_action(self, action):
    """Returns the score of an action (0, 1 or 2)"""
    def play(self):
    """Play an entire game and return results"""
```

7.3 Training Process

To improve, the agent will play games against a previous version of itself. This can be done simply by adding the players to the environment and running environment.play().

We should also ensure that the training agent gets practice as first and second player (though this will tend to matter less once boxes start being scored).

7.4 Learning algorithm

Q-Learning is the proposed learning method for this problem. The algorithm is shown below:

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

Q-Learning gives a value for a state-action pair in the environment. By repeatedly exploring the environment, and updating according to the algorithm above, we can find the optimal action for any state by examining the Q-value of state-action pairs.[5]

In addition to simple Q-Learning, we can implement a DQN with experience replay, as has been implemented by DeepMind for learning to play Atari games[6]. The pseudocode for this is shown below.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

5 Q learning. (n.d.). Retrieved May 01, 2017, from http://cse-wiki.unl.edu/wiki/index.php/Q_learning#Minimax_Q-Learning

6 Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013)

Bibilography

Reinforcement learning. (2017, April 15). Retrieved May 02, 2017, from https://en.wikipedia.org/wiki/Reinforcement_learning

"Dots and Boxes." *Wikipedia*. Wikimedia Foundation, 30 Apr. 2017. Web. 01 May 2017.

Reinforcement Learning. Richard S. Sutton and Andrew G. Barto. (1998)

Q learning. (n.d.). Retrieved May 01, 2017, from http://cse-wiki.unl.edu/wiki/index.php/Q_learning#Minimax_Q-Learning

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013)