Machine Learning Nanodegree
Capstone Proposal

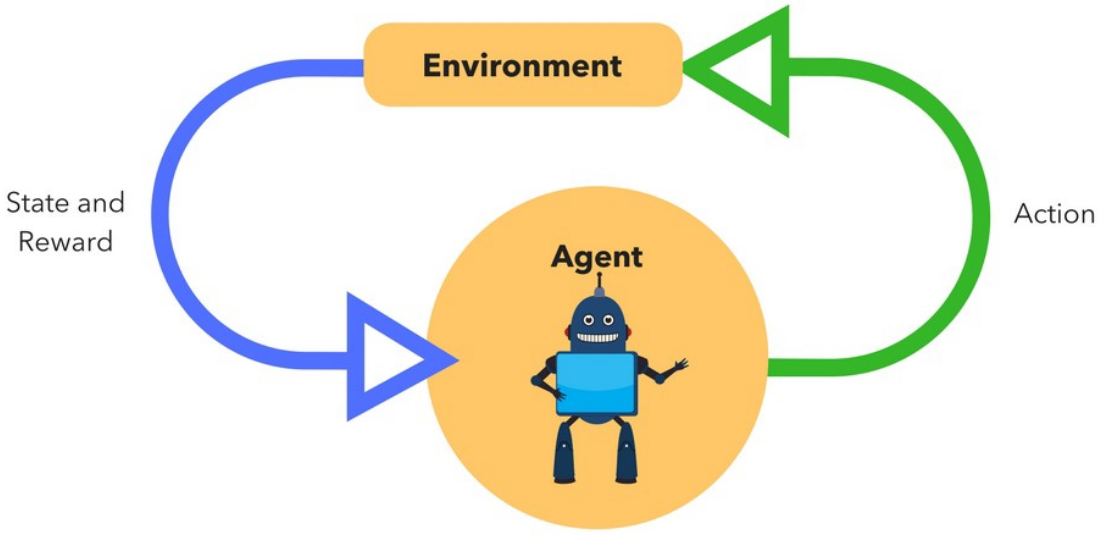
Peng Xu

February 13, 2016

1 Domain Background

Reinforcement learning addresses the problem of how agents should learn to take actions to maximize cumulative reward through interactions with the environment. The traditional approach for reinforcement learning algorithms requires carefully chosen feature representations, which are usually hand-engineered. Recently, significant progress has been made by combining advances in deep learning for learning feature representations.

Deep Reinforcement Learning has proven its ability in a varity of classic problems among video games and robotics. In some extent, a video game is equivelent to a simulation of real case, such as racing cars, drones etc. By solving a similar problem in a video game or a simulation setting, the final successful model could be applied on the countpart problem in real case using transfer learning, which saves a lot of time and money or even reverse mission impossible. Among those challenging problems how to train a vehicle to be a self-learner interests me most.



In other words, a general pipeline is to be established along with this practice of building a game bot. In the future, the pipeline could be applied on similar problems both in simulation setting and in field cases which is the motivation for me.

2 Problem Statement

In openAI-Universe, several racing car games are provided, for example, the Coaster Racer flash game, in which a vehicle is simply controlled by 3 inputs, left, right, forward. It is easily to be redesigned in a Reinoforcement Learning setting. The vehicle interacts with its environment in the way as shown in below. It is expected that the racing car can learn a smart driving behavior after a series of training steps leading to a maximal reward or namely score here.

This is starting from the "How to Install OpenAI's Universe and Make a Game Bot" live session by Siraj Raval on Youtube. The bot is for the Coaster Racer flash game and determines the answer to 2 questions – should I turn, and which way should I turn? It answers the first by checking if it's been receiving a reward for a certain interval (no crashes), and if it has it will turn. It answers the second by randomly picking a turn between left and right. This bot is able to complete the race. So it's a bot that uses reinforcement learning to determine when to turn and each turn is a random direction.

- **Task:** Playing Coaster Racer.
- **Performance:** Percent of games won against other agents.
- **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.

Therefore, I seek to build a Q-learning agent trained via a deep convolutional neural network to approximate the optimal action-value function:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{1}$$

which is the maximum sum of rewards achievable by a behaviour policy π.

3 Datasets and Inputs

By using a Deep Q-Learning model, there is no need to use an external dataset since the data are captured along with training.

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 by 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.

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.

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.

7 Project Design

7.1 Programming Language and Libraries

- **Python 2.**
- **openai-gym**. Open source machine learning library for Reinforcement Learning.
- **openai-universe**. Provide the game environment.
- **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

- **Coaster Racer flash game.** 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.

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:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2}$$

References

L. V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Maastricht University, 1994.

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.