Deep reinforcement learning in Keras

Juan Guillermo Llanos

Computational
Intelligence Laboratory

Introduction

Recent years have seen a surge of interest and research in machine learning, a field that considers the study and construction of algorithms that can learn from and make predictions on data. In this work, we are interested in a specific machine learning technique known as "deep reinforcement learning", which can be applied to teach an agent how to achieve a goal (maximize their reward) by performing the appropriate actions. The goal will be to achieve a high score in a computer game; the agent will be an idealized game player with an internal strategy that can be described by a Markov decision process. In order to carry out experiments, the game, the agent, and the game-playing will be generated and monitored within Keras, a Python neural network library.

Reinforcement Learning

Reinforcement learning is one of the three main types of learning; it is concerned with how agents should act in a dynamic environment in order to maximize a perceived reward. It is inspired by animal learning, for example how we could train a dog to sit on command by rewarding it with a treat every time he sits. In our case, the agent will learn how to play a video game. It will observe the state of the video game, respond with an action according to an internal strategy, and adjust the internal strategy in response to failure or success. The agent will not be provided with any game-specific information, and has no knowledge of the video game's internal state. It will only learn from the video input, the set of possible actions it can perform, and the reward it receives. This is similar to how a human player would learn to play the video game, so the agent can be considered to be a model of a human game player. In machine learning, the most common method to formalize reinforcement learning is as a Markov decision process.

Markov Decision Processes

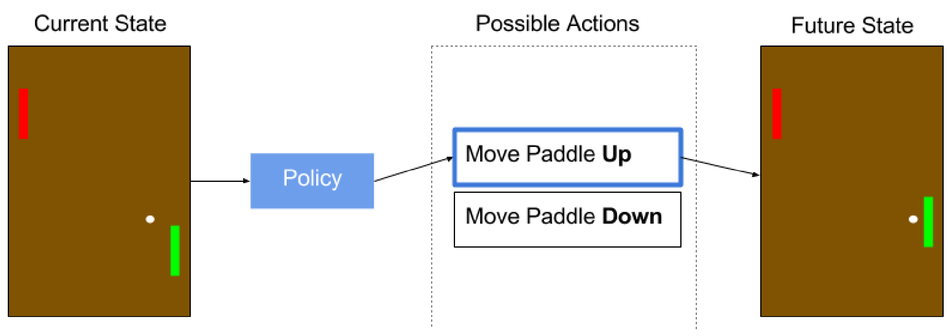

Figure 1: The ideal pong policy maps the current state to the most desirable action (moving the green paddle up). The chosen action is desirable because it points to a future state which avoids a negative reward (the ball getting past our paddle).

A Markov decision process (MDP) is a type of formalism that allows us to model decision making under stochasticity and uncertainty. For example, assume that you are an agent, and you've been placed in an environment (e.g. a pong game). The environment is under a particular state s_t at time t. (e.g. location of your paddle, location and velocity of the ball). The agent can perform a number of actions a in the environment (e.g. move the paddle left). Actions transform the environment to a new state s_{t+1}. Sometimes these actions result in a reward r_{t+1} (e.g. your score increases). Whether or not you transition from s_t to s_{t+1} by performing a is not determined by any previous state or action (more commonly known as the Markov property). The core problem of MDPs is to find a policy (or strategy) for the agent. In other words, we seek to find a function π that maps the current state s_t into the action $a = \pi(s_t)$ the agent ought to make in order to maximize future rewards. Figure 1 shows the ideal policy for the game of pong.

Deep Neural Networks

An artificial neural network (ANN) is a mathematical/computational model inspired by the brains of animals. As they are inspired by the interconnected networks in the brain, ANNs consist of a set of nodes and the connections between said nodes. Each node is seen as a single processing unit that uses the connections to receive input and send output. The combined efforts of all neurons allow the ANN to learn. The core task of the network is to take an input x and use the weights w to propagate it neuron to neuron until it can produce an output $\hat{y}$. Thus, the ANN computes a function $f_w(x)$ parameterized by the weight vector w. Ultimately, we want to find the appropriate weights that will map the input to the correct output. Thus, we want to find the optimal w that will minimize the difference between the predicted output of the neural network $\hat{y}$, and the correct output y. For example, in an image classification problem we want to pass an image to the ANN, and have it return the label of the image (e.g. cat, dog, plane).

There are many kinds of neural networks, but they can all be specified by their structure, their neural model, and the learning algorithm they use. An example of a simple feed forward neural network can be seen in figure 2.

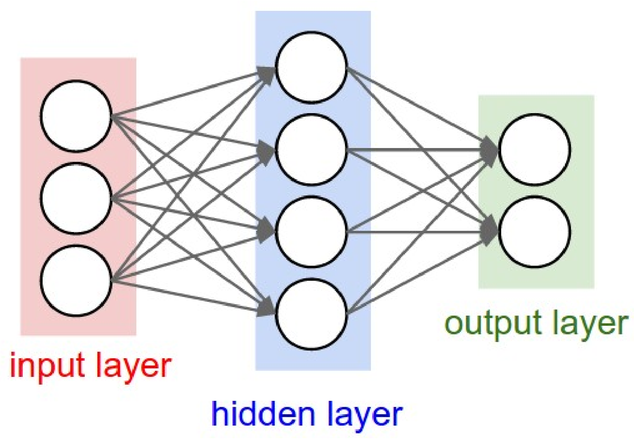

Figure 2: Single layer feed forward neural network

Keras

Keras is a neural networks library written in Python. The library allows the user to choose between two different machine learning libraries to use as backend engines: Theano and TensorFlow. It was built with simplicity and fast prototyping in mind, without sacrificing processing speed. These features facilitated the creation of the neural network and allowed us to focus on the other aspects of the project.

Implementation

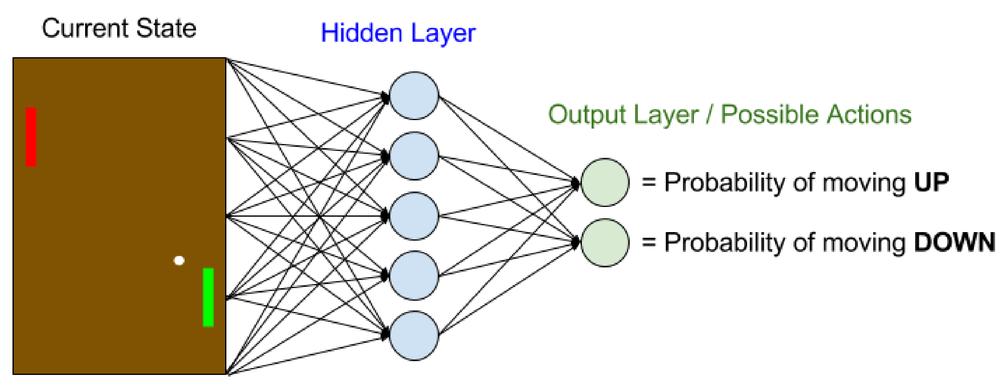

Figure 3: Neural network pong policy

Since ANNs compute arbitrary functions $f_w(x)$, and our reinforcement learning problem required us to find a policy $\pi(s_t)$, we used a neural network to approximate the desired policy function. The game outputs an image of size $210 \times 160 \times 3$ every frame, which we preprocess into a an array of size 6400 (a flattened image of size 80×80). This flattened array is what the ANN receives as input. The network outputs an array of probabilities (of size 2×1), and we sample from this array to determine our next action. This action is then sent back to the game, and the next frame we receive another image with the new state, and possibly a reward. We get a $+1$ reward if we get the ball past our opponent's paddle, and we get a -1 reward if the ball gets past our paddle. This process continues iteratively until either player reaches a score of 21, at which point the game is over.

Results and Discussion

We used a neural network with a single hidden layer of 200 neurons, with a rectified linear unit activation function. The model was trained for 10 hours. During that time, it played $200,000$ pong games. It made a total of 800 updates to the weights of the neural network. After training our agent is able to defeat the opponent (the default AI that is provided with the game) 80 out of every 100 games. Our agent's average score is 19.8, while the opponent's average score is 10.6.

Our agent learned to play Pong from raw pixels with deep reinforcement learning. Since we did not use any specific domain knowledge about Pong, our agent should also be able to learn other video games with the same level of complexity. However, the fact that it takes the agent $200,000$ Pong games to learn indicates that it might not be a very good model of how a human would learn. Humans are able to utilize prior knowledge, such as their model of physics (e.g. the ball bounces, it is not likely to teleport, it will continue to move until it meets another object), and intuitive psychology (e.g. our opponent wants to win, thus his strategy must be a reasonable starting point). This reveals glimpses as to how far away we are from understanding of human learning, but it also points towards possible research directions.

Future Work

We would like to expand the work in this paper in the following ways:

- We want to implement better ways to measure the performance of the agent, and compare it to the implementations mentioned in the related work.
- Tuning the hyper-parameters of the model (e.g. number of neurons, number of layers, time trained) will provide insight into what configuration gives the best performing agent.
- The current implementation runs on the CPU, but Keras has functionality that allows you to run the model on the GPU, which should shorten the time required to train significantly.
- Convolutional neural networks are a type of ANN that specializes in image recognition. Since our agent receives images as input, implementing this ANN variant should increase the performance of the agent.