Deep QWOP Learning

Hung-Wei Wu

Submitted under the supervision of Maria Gini and James Parker to the University Honors Program at the University of Minnesota-Twin Cities in partial fulfillment of the requirements for the degree of Bachelor of Sciences *cum laude* in Computer Science.

12/2/2017

Abstract

We apply a deep learning model to an unconventional control task of keeping a simulated human runner alive as long as possible. The model is a convolutional neural network trained with Q-learning. We show that our model is capable of successfully learn a control policy associated with playing QWOP. We find that it is possible to train the model only using raw pixel data. We successfully applied this model to a non-deterministic environment in the form of a ragdoll physics flash game.

Introduction

QWOP

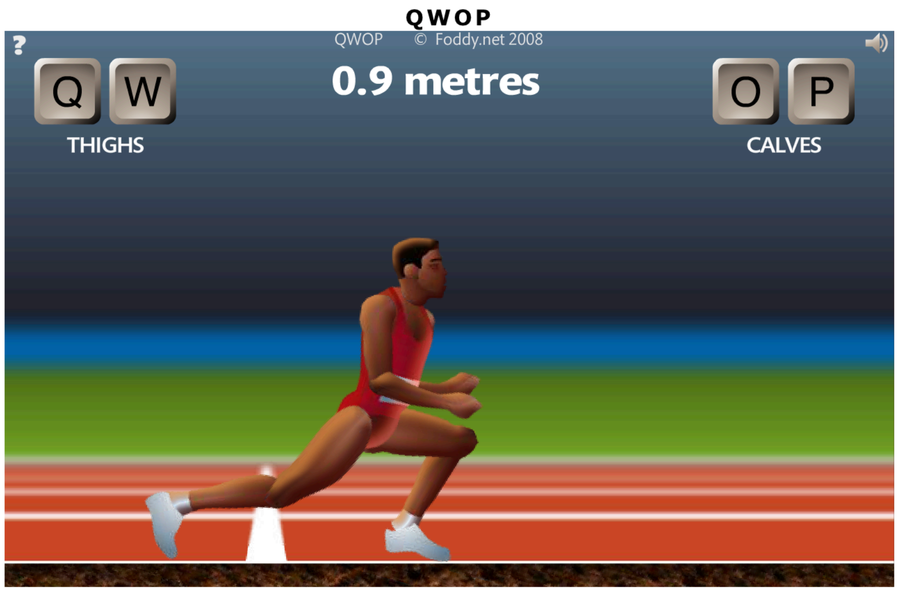

Figure 1. QWOP game play

QWOP is a flash game created by Bennet Foddy infamous for being ridiculously frustrating to play. It is free to play on the Internet. In QWOP, the user controls a ragdoll sprinter using the four keys: “Q”, “W”, “O”, and “P”. Each key controls the left thigh, left calf, right thigh, right calf respectively. With the right inputs and timing, this can be used so simulate real-world human-like running. However, this is not how we, as humans are used to running. We don’t think about how specific muscles have to move in order to maintain balance and move forward. This means that in the context of QWOP, the player’s collective knowledge on balance and movement is essentially ignored. The goal is for the user to attempt to move the ragdoll figure 100 meters without falling over. The game is reset when any section of the upper torso touches the ground. The game implements a ragdoll physics environment where normal physical interactions are greatly simplified. In particular, this means that any limbs that isn’t being directly simulated is dormant, meaning it just falls in the direction that it is already traveling. If the runner gets slightly out of balance, he falls without the user being able to help at all. The articulated figure has little to zero joint stiffness, often leading to it collapsing into comically improbably or compromising positions. The game is notoriously difficult and achieving any sort of forward movement is considered a significant achievement.

Deep Q Learning

Deep Q Learning was made famous by DeepMind in a 2013 paper “Playing Atari with Deep Reinforcement Learning” describing a deep reinforcement learning system combines Deep Neural Networks with Reinforcement Learning that is able to master a diverse range of Atari 2600 games with only the raw pixels and score as inputs. Until this point, it had only been possible to create individual algorithms capable of mastering a single specific domain. Deep Q Learning represents the first demonstration of a general-purpose agent that as able to continually adapt its behavior without human intervention. However, it has only been applied to deterministic tasks, tasks where a given action produces a given reward that can be inferred from the environment. The task of playing QWOP is significantly more difficult due to the ragdoll physics environment. Each key press is not guaranteed to have the same results or effects on the environment.

Related Work

DeepMind Atari

DeepMind published a paper in 2013 describing the first deep learning model to successfully learn control policies directly from sensory input using reinforcement learning. The input is raw pixels and the output is a value function estimating future rewards. Their method was able to learn to play seven Atari 2600 games and even surpass a human expert on three of the games. These games include Pong, Breakout, Space Invaders, Seaquest, Beam Rider. Their model is a convolutional neural network trained with a variant of Q-learning, using stochastic gradient descent to update the weights. They also implement an experience replay mechanism which randomly samples previous actions and state transitions to smooth out the training distribution over past behaviors. This will be the primary basis of our model.

OpenAI Gym

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. This platform provides many environments that agents can interact with in a unified way. It provides an interface that allows agents to step the environment by one timestep and return new observations, reward, and exit status. Example environments include the "CartPole-v0" environment. A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track.

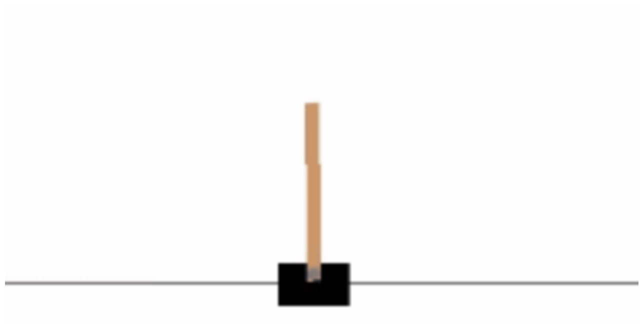

Figure 2. OpenAI CartPole environment

The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves move than 2.4 units from the center. CartPole is one of the simplest environments in OpenAI gym. An agent can move the cart by performing a series of actions of 0 or 1 to the cart, pushing it left or right. The QWOP game interface is written to follow a similar environment architecture where an agent has access to reset and step methods. An example agent implements a simple three-layer neural network and is trained using Q-learning. After around 500 episodes, the agent learned how to maximize the score by keeping the pole upright and the cart in the center of the environment.

Stanford CS229

Gustav Brodman and Ryan Voldstad used reinforcement learning to play QWOP. Methods included discretization of state spaces with regular value iteration and fitted value iteration using a set of reward features. Instead of using raw pixel inputs, other variables were used to better quantify the QWOP runner's state. Distance alone is not enough to determine the state of the runner so other variables such as number of feet on the ground, left and right knee angles, angle between the left and right legs, and thigh rotational velocities were used to represent the state. Through some experimentation, they settled on a feature mapping using the difference between thigh angles, the angles of each knee, the overall “tilt” of the runner, and the runners horizontal speed. Evaluating their model resulted fairly good results. The QWOP sprinter was able to travel around 30000 units (arbitrary distance units.) Initially, a shuffling gait was observed. However, after 10 iterations, a gait that resembled actually walking was observed.

Background

Our QWOP agent implements the Deep Q Learning algorithm using a neural net and reinforcement learning. Q-learning is a model-free reinforcement learning technique. Specifically, Q-learning can be used to find an optimal action-selection policy for any given finite Markov decision process. In Q-learning, there is an action-value function called the Q function, which is used to approximate the reward based on a state. It ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. A policy is a rule that the agent follows in selecting actions. When such an action-value function is learning, the optimal policy can be constructed by simply selected the highest values in each state. We use a neural network to model the Q-function.

$$loss = \left(\underbrace{\overset{\text{Reward}}{\mathrm{r}} + \overset{\text{Decay Rate}}{\gamma} \max_{a`} \hat{Q}(s, a`)}_{\text{Target}} - \underbrace{Q(s, a)}_{\text{Prediction}} \right)^2$$

Figure 1. Q value and loss calculation

An agent first carries out an action a and observe the reward r and the resulting state s. Based on the result, we calculate the maximum target Q value and then discount it so that the future reward is worth less than immediate reward. The most notable features of the Deep Q Learning algorithm are the remember and replay methods. One of the challenges for Deep Q Learning is that the neural network used in the algorithm tends to forget the previous experiences as it overwrites them with new experiences. Thus, methods are needed to remember previous actions and rewards and retrain the neural network to retain previous knowledge. To ensure the agent performs well long term, we need to take into account not only the immediate rewards but also the future rewards. In order to accomplish this, a discount rate is specified. Thus, the agent will learn to maximize the discounted future reward based on the given state.

Initially, the neural network is not trained to maximize the Q-function. Thus, the QWOP agent will then randomly select possible actions a set percentage of the time. This is specified using the exploration rate. It is better for the agent to try different actions and observe the rewards and start converging on the optimal action-value function. However, when the agent is not randomly deciding its actions, it will predict the reward value based on the current state and pick the action that will give the highest reward.

There are also some hyper parameters that has to be specified when the model is being trained. They are listed below in Figure 4.

Episodes	The number of games the agents are going to play.
Gamma	The decay rate used to calculate the future discounted reward.
Epsilon	The percentage that the agent will randomly decide its actions.
Epsilon decay	As the network gradually learns patterns, it will explore less and less.
Learning rate	How much the network learns in each iteration.

Figure 4. Deep Q Learning hyper parameters

The two hidden layers in the neural network used to train the Q-function are composed of rectified linear unit neurons (ReLu). The ReLu is an activation function defined as the positive part of its argument. The function is shown in Figure 5, where x is the input to a neuron. It was first introduced in 2000 with strong biological motivations and mathematical justifications. It has been used in convolutional networks more effectively than the widely used logistic sigmoid function.

$$f(x) = x^{+} = \max(0, x),$$

Figure 5. ReLu activation function

Since there are four possible inputs into the QWOP game interface, and buttons can be pressed concurrently, instead of modeling the actions as four distinct outputs, an alternative key schema is defined. There are now 16 distinct outputs, each representing a combination of four keys. The schema is defined below in Figure 6. Each row represents one of the 16 possible 4-key combinations and the 1s and 0s represent if that corresponding key is pressed or not.

	Q	**W**	**O**	**P**
A	0	0	0	0
B	0	0	0	1
C	0	0	1	0
D	0	0	1	1
E	0	1	0	0
F	0	1	0	1
G	0	1	1	0
H	0	1	1	1
I	1	0	0	0

J	1	0	0	1
K	1	0	1	0
L	1	0	1	1
M	1	1	0	0
N	1	1	0	1
O	1	1	1	0
P	1	1	1	1

Figure 6. Key input schema definition

Methods

In order for the agent to interface with the QWOP game environment, it had to be able to simulate keyboard input as well as read the raw pixels on the screen. This was achieved by creating a virtual environment for the Deep Q Learning agent to get the current state and step through actions. Another variable that was needed was the current distance that the runner is at. However, due to the obfuscated nature of the native JavaScript code, we had to rely on other methods. We utilized OpenCV to find contours of the numbers and wrapping rectangles. The raw pixels at those locations are then cropped and fed into a support vector machine trained to predict which number it was. The Python Imaging Library (PIL) was used to take screenshots of the game and feed it as raw input into the agent. Pyautogui was used to simulate keyboard input.

Initially, an environment representing the QWOP game was instantiated. Then an agent was created. Based on the number of episodes, the agent will step through 500 predicted actions and subsequent training until it falls over and the game resets or all 500 actions are executed. That concludes one episode. Every tenth episode, the current weight and biases in the neural networks are cached in a backup file. We limit the input to be a small section of the runner's legs in an effort to reduce the time to train the neural network. The reward is defined as how long the agent stays alive. Thus, the reward is greater the longer the agent is alive. And the Q-function is incentivized to choose actions that correspond with stability.

Results

Through personal experimentation and gaming experience, one reliable way to stay alive is to either hold zero keys down or press the keys that will result in the runner with its legs as spread apart as possible. Initial trials with 1000 episodes of 500 steps each yielded promising results. The hyper parameters were set as follows in Figure 7. The agent will start off by guess 100% of the time and every episode will decrease the guessing rate by 0.5%. For fear of overshooting, the learning rate was defined to be as small as possible. However, one trade off what that it takes a significant time for the neural network to converge on the optimal Q-function.

Episodes	1000
Gamma	0.95
Epsilon	1.0
Epsilon decay	0.995
Learning rate	.0001

Figure 7. Hyper parameters for initial trials

As more episodes were executed, the agent learned to press the same key over and over again. The particular key combination was "J", which corresponds to holding the "Q" and "P" key down. This configuration allowed the runner to get in a position similar to someone doing the lunges. This position proved to be the most stable, as repeated presses of "J" after entering the lunge position could not make the agent fall over. Due to the low learning rate and low epsilon decay rate, each training session took upwards of eight hours. However, given the hyper parameters, the Deep Q Learning agent learned to start pressing the same keys around episode 300. Then around episode 500, the key pressed becomes “J”. The key combination that provides the most stability.

With a working Deep Q Learning agent, we attempted to shorten the training time by increasing the learning rate and epsilon decay. Meaning that the agent will guess less and find the global minima faster. However, it is important to note that a very small learning rate will cause the network to train extremely slowly, and if it is too high, we risk overshooting and never finding the global minima. By changing these hyper parameters, the agent was able to learn to press “J” repeatedly by episode 200. However, this did not significantly decrease our experimentation time. Since the agent is staying alive longer, each episode would last longer as well.

Plotting the most common key schema over episode number, a trend is easily noticed. The agent initially starts to randomly guess actions and occasionally survives over five episodes. And as we cross 250 episodes, the network learns that pressing the same keys results in a higher reward. This plot is shown below in Figure 8.

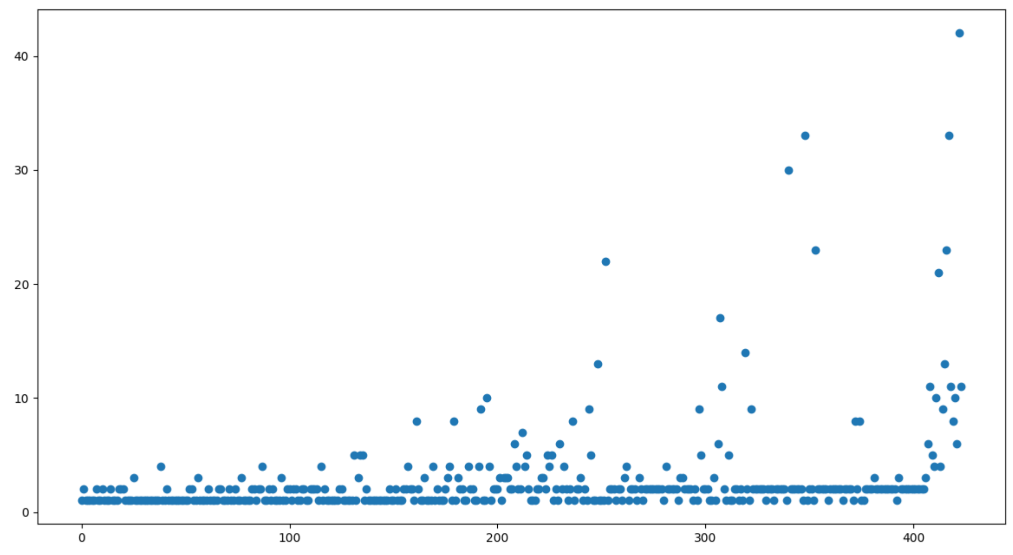

Figure 8. Most common key count every episode

Plotting the time alive over episode number, a similar trend to the one shown in Figure 8 can be observed. We can see that after episode 500, the agent was able to stay alive consistently through the 500 actions. This plot is shown below in Figure 9. Both plots show a slight exponential growth trend. Which is expected, as the network learns the correct sequence of actions to take, they are predicted more often and result in higher reward, creating a positive feedback loop.

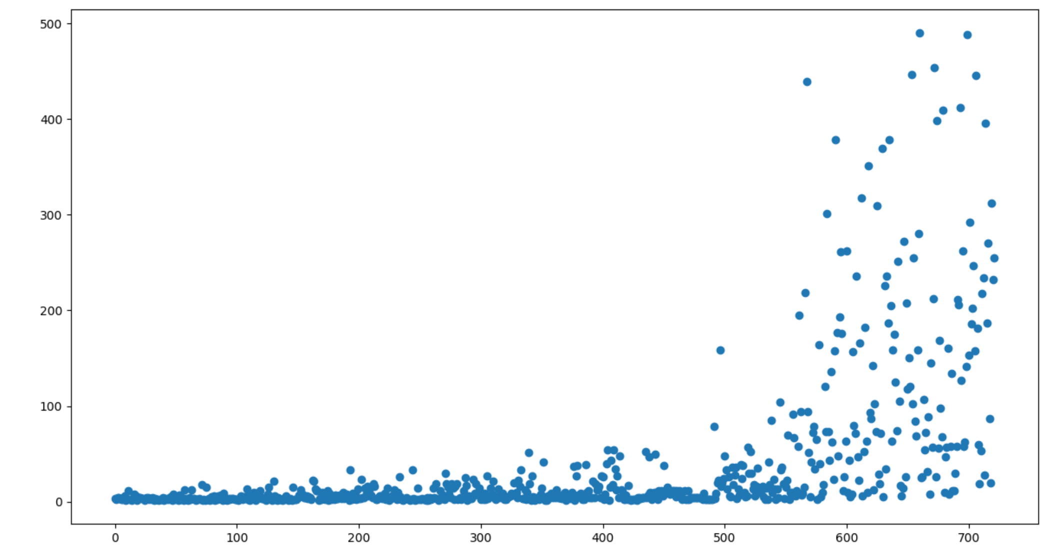

Figure 9. Time alive every episode

Conclusion

In this paper, we discussed applying Deep Q Learning to a nonconventional control task of keeping the QWOP running alive as long as possible. This is in contrast to the traditional way success is measured in QWOP. Typically, success is defined as distance traveled, however, we redefined the problem and was able to successfully apply Deep Q Learning. We have shown that with only raw pixel inputs, a neural network can converge to the optimal value of the Q-function. After roughly half of the expected training episodes, the agent learned to stay alive by repeatedly “Q” and “P”.

References

Strehl, Li, Wiewiora, Langford, Littman. “PAC model-free reinforcement learning”, 2006.

Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, Riedmiller. “Playing Atari with Deep Reinforcement learning”, 2013.

Brockman, Cheung, Pettersson, Schneider, Schulman, Tang, Zaremba. OpenAI Gym, https://github.com/openai/gym, 2016.

Fran, Others. Keras. https://github.com/fchollet/keras, 2015.

Brodman, Voldstad. “QWOP Learning”, 2012.