Improving the Q Learning Algorithm with Rudimentary AI

Burke Brockelbank
(Dated: August 5, 2018)

Neural networks have effectively been taught to play video games. These neural nets have historically been trained through a method of reinforcement learning called Q learning. One of the major challenges with this method occurs in the early stages of training when the quality function is most inaccurate. Here we show a method for skipping past this initial training phase, decreasing the time to convergence. In this method, the neural net first trains on labelled data generated by a sub-optimal, but theoretically simple, artificial intelligence until it is capable of playing the game to a similar effectiveness. It is then trained with Q learning to refine the performance of the AI. This method is applicable to any Q learning scenario where it is easy to concieve an AI to prepare labelled data.

I. INTRODUCTION

The method of training explored in this report is Q learning. Q learning has always had the problem of slow convergence near the beginning of its training, an issue that is inherent its formulation. Q learning is shown to theoretically converge in infinite time with some restrictions on the learning rate in [1] and has proven capable of converging with no assistance in practice, however the amount of computation power and time needed to do this may not always be feasible. Additionally, it is better to avoid the problems with Q learning rather than trust in the theory in case a specific implementation makes it impossible to use.

Another issue with reinforcement learning is that its behavior is highly dependent on how the reward function is specified. Poor choices of reward functions can lead the model to make decisions which are not desired. In [2] the authors describe an interesting example where the specific reward function they used lead to a difference between how they wanted their player to act and how their player ended up maximizing reward.

A. The Quality Function

The formulation of DQ begins with specifying the reward of an action. Suppose that there is a player in a state s who takes an action a. This action will move the player from the state s to s' and also give the player some immediate reward $r(s,a)$. For example, if the player is a robo-advisor that is trading stocks then the state is the current price of all stocks and the action is either to wait, sell a stock, or buy a stock. The reward may be specified as the amount of value gained in moving from the state s to s'. After this the player will take the action a' from the state s', followed by taking the action a'' from the consequent state s'' and so on. The cumulative reward R will include all of these rewards, but to reflect the uncertainty about future actions, a discount factor $\gamma \in (0,1)$ is introduced.

$$R = r_{s,a} + \gamma r_{s',a'} + \gamma^2 r_{s'',a''} + \dots, \tag{1}$$

Suppose we had some ideal policy function π^* that we could consult at every step to give us the ideal action to maximize total reward given a state. Then we define our quality function as,

$$Q^*(s,a) = r_{s,a} + \gamma r_{s',\pi^*(s')} + \gamma^2 r_{s'',\pi^*(s'')} + \dots. \tag{2}$$

Thus, $Q(s,a)$ represents the maximal cumulative reward we can get from the state s given that we are going to perform the action a from that state. We can rewrite this as,

$$Q^*(s,a) = r_{s,a} + \gamma Q^*(s', \pi^*(s')). \tag{3}$$

which is known as the Bellman equation. Additionally, π^* can then be written in terms of Q^* as,

$$\pi^*(s) = \arg\max_a Q^*(s,a). \tag{4}$$

The objective of Q learning is to use a neural net to model this quality function. Let the approximation be Q. Define a policy function analogously to (4),

$$\pi(s) = \arg\max_a Q(s,a). \tag{5}$$

Then there will be some difference between the two sides of (3) called δ.

$$Q(s,a) = r_{s,a} + \gamma Q(s', \pi(s')) + \delta. \tag{6}$$

The loss is then calculated as a function of δ.

In order to train the neural net effectively, some degree of exploration is used. In the case of this report, π is replaced with π_ϵ where $\epsilon \in [0,1]$. π_ϵ has a probability ϵ of returning a random action and a $(1-\epsilon)$ probability of returning π.

B. Monkey Want Banana

This report considers the performance of a neural net playing a simple game where the player controls a monkey in a square grid trying to collect bananas while traversing walls and avoiding lava. Every action the monkey takes expends some of its food so it must constantly

FIG. 1. A demonstration of the monkey game showing all the block types and the movement interface. Block types are barriers `#`, monkeys `m`, bananas `b`, and lava `d` which is symbolized by a `d` for danger. The line beginning with `>>>` is the movement interface. In this case, the player has chosen to move downwards. These controls are defined with the common `wasd` movements. Additionally the player can choose not to move by entering a space key which we will symbolize as _. The numbers on the perimeter of the map show the position of objects on the map by splitting it into a number modulo 10 and the ones digit. In this image, the monkey is at position (239,251).

find new bananas to eat. The game is written with a text-based ASCII art interface. One screen of the game is shown in Figure 1.

A natural choice of reward function is the food level. Specifically, the immediate reward of some action a taking the game state from s to s' is the food level in s' minus the food level in s. Doing an action takes one food away from the monkey, so if no banana is eaten, the immediate reward is -1. Eating a banana give the monkey 5 food, so if the monkey moves to collect a banana, it gets an immediate reward of 4. The monkey also may starve to death, or move into lava, in which case it will get a reward of -50. The discount factor was set to 0.8 since this fairly accurately describes the lack of knowlege the monkey has outside of its $11x11$ vision range.

The loss function was decided as the smooth L1 loss,

$$\text{loss}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \|\delta\| \leq 1 \\ \|\delta\| - \frac{1}{2} & \|\delta\| > 1 \end{cases}. \tag{7}$$

C. Challenges with Q Learning

One of the main issues with Q learning is evident in (6). Calculating the loss for performing the action a from the state s, it depends on the next state. Hypothetically,

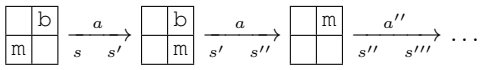

FIG. 2. A series of hypothetical states in the monkey game. The actions taken in this series represent one of the possible sets of actions that would be given by consulting π^* at every state.

δ could be very small or even zero even if Q does not accurately model Q^* as long as $Q(s,a)$ and $\gamma Q(s', \pi(s'))$ differ from their respective Q^* predictions by the same amount. If Q is inaccurate, like at the beginning of its training, it will converge slowly.

For an example of this challenge in the monkey game, consider the series of states in Figure 2. First we will derive Q^* from the states in Figure 2 and then show how with some issues in Q can cause slow learning. We can work backwards from the final state to derive Q^*. Assuming that after s''' the monkey cannot find any more bananas and does not die, it will just eat one food per turn and thus have each immediate reward be -1. The quality can be determined via (2) as,

$$Q^*(s''', a''') = (-1) + \gamma(-1) + \gamma^2(-1) + \ldots = -5. \tag{8}$$

From this we calculate all previous qualities via (3).

$$Q^*(s'', a'') = (-1) + \gamma Q^*(s''', a''') = -5, \tag{9}$$
$$Q^*(s', a') = (4) + \gamma Q^*(s'', a'') = 0, \tag{10}$$
$$Q^*(s, a) = (-1) + \gamma Q^*(s', a') = -1. \tag{11}$$

Suppose that Q misevaluates $Q(s'', a'')$ as 0 and $Q(s', a')$ as 4. Then (6) becomes,

$$\begin{aligned} Q(s', a') &= r_{s,a} + \gamma Q(s'', \pi(s'')) + \delta \\ 4 &= 4 + \gamma 0 + \delta \\ \delta &= 0. \end{aligned}$$

This gives a loss of zero so no learning will be done from the evaluation from state s' until $Q(s'', a'')$ changes. This same effect occurs during training with the effect of making delta either too large or too small. This arises from the fact that, at the beginning of training, Q does not come close to satisfying the Bellman equation. However, if Q is already close to Q^* then it will be close to the Bellman equation and the effect will not be as pronounced.

II. METHODS

The training of the neural net came in three major forms. In the first form, the neural net was trained on labelled data generated by the AI. In the second form, the neural net was trained in some curated situations to fix some deficiencies likely caused by issues in the labelled

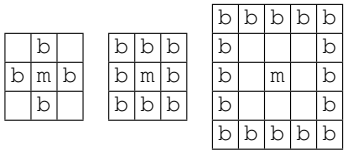

FIG. 3. Three different levels of curated training are show. Places where bananas are allowed to be placed are marked with a b while the monkey's position is marked with an m.

data. This form will be called curated training. Different levels of curated training are shown in Figure 3. In the third form, the neural net was allowed to explore and learn on its own in the same environment that the AI ran in. This form will be called random reinforcement training

In the first part of training labelled data had to be created for the AI to play in. Data was created in game boards which had monkeys, bananas, and lava but no barriers. It was determined that adding in barriers caused the AI to get stuck far too often so data was generated without them. Even without barriers, the AI often got stuck in infinite loops which added noise to the dataset. This is likely the reason that the model didn't converge well on the AI and was incapable of replicating it. Despite this, the first phase of training proved invaluable to the overall training. When we attempted to skip this step, the model could not be trained to ouperform the AI.

The second part of training was meant to address some obvious shortcomings that came from training on such a poor data set. This part included both curated training and random reinforcement. Initially, the model underwent curated training with bananas directly adjacent (above, below, left or right) followed by a distance of one block away This part of training had When a banana was placed in close proximity to the monkey with nothing else in its surroundings, the model would often not move the monkey to the banana. The model trained with the Q learning algorithm on successive stages of putting a banana close to the monkey at increasing distance at each stage. These stages ran until the monkey could consistently approach bananas up to a distance of two blocks away.

III. RESULTS

The model was then tested against the AI in a room with a random distribution of 2% barriers, 1% lava, and 5% bananas

emphasize how the AI was so simple and really insufficient for the task at hand but was necessary in creating a much more effective neural net

IV. DISCUSSION

talk about how the convolutional branch of the neural net was not sufficient for dealing with far-away objects alone and the neural net would likely benefit breatly by expanding the non-convolutional branch out to the full 11x11 grid.

[1] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction* (MIT press, 1998).
[2] J. Clark and D. Amodei, "Faulty reward functions in the wild," (2017).