Using the REINFORCE Algorithm to Solve the Cartpole Problem

Michael Boss

An undergraduate thesis advised by Guenter Schneider,
submitted to the Department of Physics, Oregon State University,
in partial fulfillment of the requirements for the degree B.S. in Physics.

Draft submitted on November 4, 2022

Abstract

In reinforcement learning, an agent takes actions based on the state of their environment and receives a reward for their action. The goal is to create a policy, a function which maps from states to actions, that maximizes the expected value of the reward. Neural networks are often used as policy functions because of their ability to find a function that optimizes the reward through training. In this work, fundamental theory about reinforcement learning, neural networks, and using neural networks as a policy functions is introduced. The REINFORCE algorithm and a feedforward neural network are used to solve the cartpole problem. The solution found by the REINFORCE algorithm was compared to a solution to the cartpole problem found by common sense physical problem solving. The REINFORCE algorithm greatly outperformed the physical solution and was competitive with other reinforcement learning algorithms. Performance was best when using a discount factor of one, and worse with lower discount factors. Overall, this experiment suggests that reinforcement learning can outperform old school engineering when trying to solve continuous control problems.

Contents

List of Figures

1 Introduction

A control problem is a problem where a dynamical system needs to be controlled. For example, programming a robotic hand to pick up an object would be a control problem. At Oregon State University Dynamic Robotics Laboratory, researchers aim to control the movement of bipedal robots using reinforcement learning [1]. Other groups have used reinforcement learning to teach robots to do tasks such as pancake flipping, bipedal movement, and archery [2]. Modeling the dynamics of a bipedal robot is a daunting and complicated task, but fortunately such robots have been controlled successfully with the use of reinforcement learning. In fact, reinforcement learning is becoming as integral to robotics as mathematics is to physics [3].

When attempting to solve a difficult control problem, it is often natural for a scientist or engineer to try and model the dynamics of the problem with differential equations. However, sometimes dynamical systems become so complicated it is not viable to create accurate physical models. Fortunately, there is another approach which should be considered. Reinforcement learning can be used to solve many control problem environments. It is a branch of machine learning where the goal is create a set of rules called the policy. An agent carries out the policy in their environment and receives a reward for each action they take. In reinforcement learning, the goal is to find a policy which maximizes the agent's expected reward. Consider a simple game where a character attempts to walk around a grid world. Some of the tiles in the grid world are safe, and others cause the character to fall into water. The character wins by walking from the start of the grid world to the end without falling into water. In this example, the agent is the character, the environment is the grid world, and the reward could be plus one for completing the walk and zero for falling into the water. The goal of reinforcement learning would be to find a policy for the character which maximizes their chances of completing the walk safely. Any algorithm which attempts to solve a game in this manner is a reinforcement learning algorithm. Reinforcement learning can also be applied to complicated control environments with great success. In fact, sometimes reinforcement learning algorithms outperform algorithms which use the underlying physical model of a control environment. Reinforcement learning has been used to solve a variety of

problems including the cartpole problem, driving a car, and legged locomotion [4].

This work aims to show the power of using reinforcement learning to solve control problems compared to engineering solutions from physics. The control problem of interest is the cartpole problem, which is a system comprised of an upright pendulum attached to a cart on a track. A unit force can be exerted on the left or right of the cart at every point in time. The goal is to pick forces which keep the cart on the track and the pendulum up right. This problem is solved with a physical solution derived from the dynamics of the system and with a reinforcement learning algorithm. The techniques are compared to see how the solution engineering by a scientist compare to that of a computer. By studying reinforcement learning on control problems for which solutions can be viably engineered, the ability of reinforcement learning to control systems which present engineering challenges can be inferred.

2 Theory

2.1 Reinforcement Learning

An agent interacts with an environment in a reinforcement learning process. At each time step, t, the agent observes the state of the environment s_t. The space of all possible states is known as the state space. The agent then performs an action, a_t, which is defined by the agents policy function $\pi(a|s)$. The space of all possible actions is called the action space. The policy is denoted $\pi(a|s)$ because it is stochastic, i.e. it returns the probability of taking action a given state s. The action the agent chooses is randomly sampled from the probability distribution found from applying the policy to all the valid actions in the action space. Deterministic policies, $\pi(s)$, can also be used, but are not in this work. Finally, the agent receives a reward, r_t, from the environment. A training episode is defined as a complete trajectory, $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$, which is a sequences of states, actions, and rewards. The cumulative reward, R_t, at any given t, is the sum of all future rewards in τ. It is computed by

$$R_t = \sum_{t'=t}^{T} r_{t'}. \tag{1}$$

The goal of reinforcement learning is to find a policy that optimizes the expected value of the cumulative reward for a whole trajectory, R_0, through the agent's trial and error. In other words, the goal is to find a $\pi(a|s)$ which maximizes

$$\mathbb{E}[R_0] = \sum_{\tau} P(\tau) R_0. \tag{2}$$

The right side of Equation 2 is a sum over all possible τ. There, $P(\tau)$ is the probability of τ occurring, and thus the right side of Equation 2 is a direct calculation of expected value. Sometimes, a discounted cumulative reward, G_t, is used rather than R_t. In this case,

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{3}$$

where $\gamma \in [0, 1]$ is called the discount factor. Lower values of gamma make the rewards at time steps further from t more negligible. Tuning γ allows for the reinforcement learning process to more heavily consider immediate rewards.

2.2 Neural Networks

A common technique in reinforcement learning is to use a neural network for the policy function. The neural network is often called a policy network in this context. Neural networks are a good choice for a policy function because any function can be approximated by a neural network [5]. Also, a policy network can be trained to optimized the expected value of the agent's reward. There are several different kinds of neural networks, but this work focuses solely on feedforward neural networks. A feedforward neural network is made up of layers of neurons. Each layer in the network has its own mathematical function called the activation function. Every neuron in a layer applies the activation function to the input received from the previous layer. The neuron then sends the output of the activation function, known as the activation, to every neuron in the next layer. So, the input for a neuron is a linear combination of the activations from the previous layer of neurons plus a constant called the bias [6]. The number of layers and number of neurons in each layer is called the architecture of a neural network. Neural network architectures can be visualized with graphs where the nodes represent neurons, and the arrows represent the outputs from one neuron being sent to another neuron.

Figure 1 shows an annotated neural network architecture. The neural network shown in Figure 1 takes a vector $\boldsymbol{x} \in \mathbb{R}^{m^{[1]}}$, and outputs a vector $\boldsymbol{y} \in \mathbb{R}^{m^{[L]}}$. The activations of the input layer of the network are set to $\boldsymbol{x}$, and the elements of $\boldsymbol{y}$ are set to the activations of the output layer. The activation of neuron i in layer $l+1$ is denoted $a_i^{[l+1]}$, and is computed by

$$a_i^{[l+1]} = \sigma_{l+1}(\sum_{k=1}^{m^{[l]}} w_{ik}^{[l+1]} a_k^{[l]} + b_i^{[l+1]}). \tag{4}$$

Here, $m^{[l]}$ is the number of neurons in layer l, $b_i^{[l+1]}$ is the bias of neuron $a_i^{[l+1]}$, $w_{ik}^{[l+1]}$ is the weight multiplied from the output sent from neuron k in layer l to neuron i in layer $l+1$. Lastly, σ_{l+1} is the activation function of layer $l+1$. Equivocally, the activations and biases

in layer $l+1$ can be represented by vectors $\boldsymbol{a}^{[l+1]}$ and $\boldsymbol{b}^{[l+1]}$. The activations in layer l will thus be represented by $\boldsymbol{a}^{[l]}$. In this case, the activations of all neurons in layer $l+1$ can be computed by

$$\boldsymbol{a}^{[l+1]} = \sigma(\boldsymbol{W}^{[l+1]}\boldsymbol{a}^{[l]} + \boldsymbol{b}^{[l+1]}), \tag{5}$$

where $\boldsymbol{W}^{[l+1]}$ is the matrix whose elements are $w_{ik}^{[l]}$.

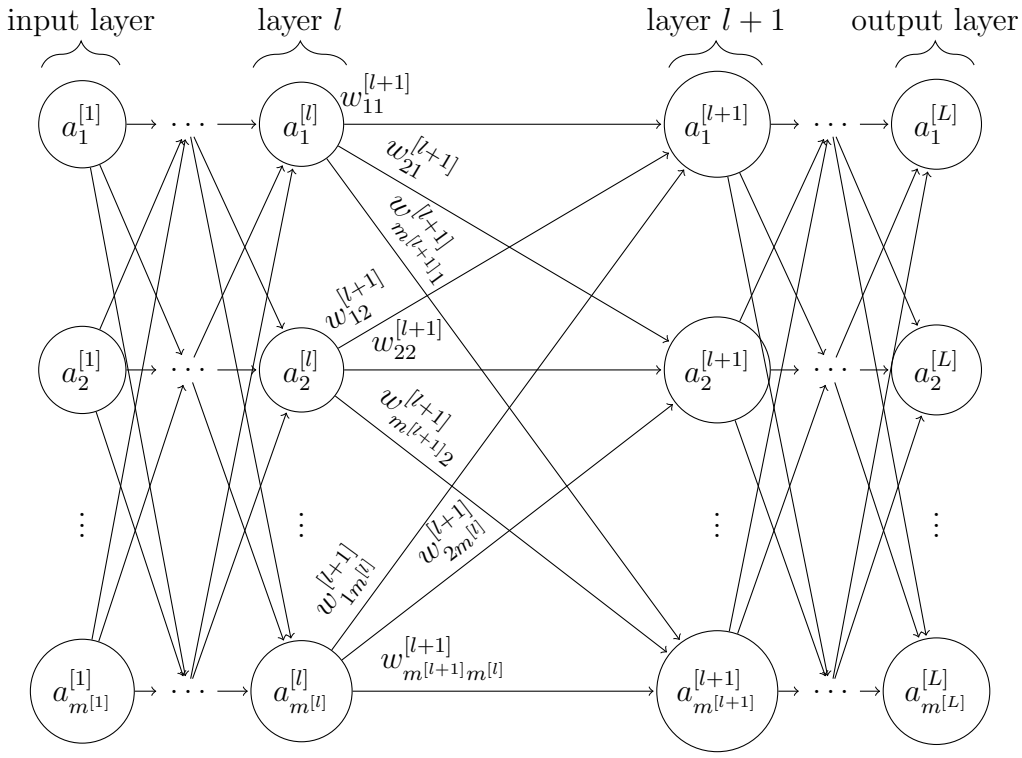


Figure 1: The architecture of an arbitrary feedforward neural network which maps from $\mathbb{R}^{m^{[1]}}$ to $\mathbb{R}^{m^{[L]}}$. The weights of the network are annotated as $w_{ij}^{[l+1]}$, and the activations are labeled $a_i^{[l]}$. The arrows represent how outputs from one layer are moved to the next layer as inputs.

In a physical control environment, the space of all possible states of the environment is called the state space and the space of all possible actions is called the action space. The input layer of the policy network used to solve the environment must accept vectors from the state space and the output layer must output vectors in the action space. In the cartpole problem the state space is $\mathbb{R}^4$, where the elements of vectors in this space correspond to the position of the cart, velocity of the cart, the angular position of the pendulum, and the angular velocity of the pendulum.

When a neural network is used in reinforcement learning, the learning process begins by initializing the weights and biases of the network randomly. Then, a trial simulation is performed using the network. At each time step, the network inputs the state of the environment into the network, which outputs a probability distribution of actions for the agent to take. The agent then takes an action randomly sampled from the probability distribution and receives a reward. After using the policy network to determine an entire trajectory, the network's weights and biases are updated to maximize the agent's expected reward via a process called gradient ascent.

2.3 Gradient Ascent

An objective function, $U(\boldsymbol{\theta})$, can be defined as

$$U(\boldsymbol{\theta}) = \mathbb{E}[G_0] = \mathbb{E}[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}], \tag{6}$$

in the case where a discounted reward is used. Here, $\boldsymbol{\theta}$ is a vector containing all the weights and biases of the network. Using this definition, the gradient of the objective function can be used to maximize the expected reward. This is accomplished by taking a gradient ascent of the objective function. This means to update the weights of the network at the end of each training episode by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}). \tag{7}$$

In Equation 7, α is called the learning rate and it determines how quickly the network will find a local maximum of the objective function. It may seem tempting to pick a very large learning rate, but this can lead to the network overshooting the local maximum.

2.4 Backpropagation

In order to take a gradient ascent, the gradient needs to be computed in the first place. Calculating the gradient of the objective function, $\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$, is accomplished using a process called backpropagation. Backpropagation works by computing the partial derivatives with respect to the weights and biases of the last layer first. Then these derivative are used to

compute the derivatives for the previous layer by the chain rule. All the partial derivatives with respect to the weights and biases are computed in this manner moving backwards thought the network layer by layer. The backpropagation algorithm is derived as follows.

Remember, $\boldsymbol{\theta}$ is a vector containing all the weights, $w_{ij}^{[l]}$, and biases, $b_i^{[l]}$, of the network. To start, the derivative $\frac{\partial U}{\partial w_{ij}^{[L]}}$ is derived using the chain rule. Here, L is the output layer and is natural to start with,

$$\frac{\partial U}{\partial w_{ij}^{[L]}} = \frac{\partial U}{\partial a_i^{[L]}} \frac{a_i^{[L]}}{\partial w_{ij}^{[L]}}. \tag{8}$$

Remember that, $a_i^{[L]} = \sigma_L(\sum_{k=1}^{m^{[L-1]}} w_{ik}^{[L]} a_k^{[L-1]} + b_i^{[L]})$. We can let $z_i^{[L]} = \sum_{k=1}^{m^{[L-1]}} w_{ik}^{[L]} a_k^{[L-1]} + b_i^{[L]}$, then

$$\frac{\partial a_i^{[L]}}{\partial w_{ij}^{[L]}} = \frac{d\sigma_L}{dz_i^{[L]}} \frac{\partial z_i^{[L]}}{\partial w_{ij}^{[L]}} = \sigma_L'(z_i^{[L]}) a_j^{[L-1]}. \tag{9}$$

Finally, by substitution

$$\frac{\partial U}{\partial w_{ij}^{[L]}} = \frac{\partial U}{\partial a_i^{[L]}} \sigma_L'(z_i^{[L]}) a_j^{[L-1]}. \tag{10}$$

Now, the derivative of U with respect to the bias, $b_i^{[L]}$, is derived. Due to the chain rule,

$$\frac{\partial U}{\partial b_i^{[L]}} = \frac{\partial U}{\partial a_i^{[L]}} \frac{\partial a_i^{[L]}}{\partial b_i^{[L]}}, \tag{11}$$

and

$$\frac{\partial a_i^{[L]}}{\partial b_i^{[L]}} = \frac{d\sigma_L}{dz_i^{[L]}} \frac{\partial z_i^{[L]}}{\partial b_i^{[L]}} = \sigma_L'(z_i^{[L]}). \tag{12}$$

So by substitution,

$$\frac{\partial U}{\partial b_i^{[L]}} = \frac{\partial U}{\partial a_i^{[L]}} \sigma_L'(z_i^{[L]}). \tag{13}$$

Now consider arbitrary layer l. The derivative $\frac{\partial U}{\partial w_{ij}^{[l]}}$ is derived once again using the chain rule. So,

$$\frac{\partial U}{\partial w_{ij}^{[l]}} = \frac{\partial U}{\partial a_i^{[l]}} \frac{\partial a_i^{[l]}}{\partial w_{ij}^{[l]}}. \tag{14}$$

Applying the chain rule once again,

$$\frac{\partial U}{\partial a_i^{[l]}} = \sum_{k=1}^{m^{[l+1]}} \frac{\partial U}{\partial a_k^{[l+1]}} \frac{\partial a_k^{[l+1]}}{\partial a_i^{[l]}} = \sum_{k=1}^{m^{[l+1]}} \frac{\partial U}{\partial a_k^{[l+1]}} w_{ki}^{[l+1]} \sigma'_{l+1}(z_k^{[l+1]}). \tag{15}$$

Then, the partial derivatives of U with respect to the weights in layer l can be written as

$$\frac{\partial U}{\partial w_{ij}^{[l]}} = \sum_{k=1}^{m^{[l+1]}} \frac{\partial U}{\partial a_k^{[l+1]}} w_{ki}^{[l+1]} \sigma'_{l+1}(z_k^{[l+1]}) \frac{\partial a_i^{[l]}}{\partial w_{ij}^{[l]}}, \tag{16}$$

which after evaluating the derivative of the activation in layer l, $a_i^{[l]}$, with respect to the weight $w_{ij}^{[l]}$ can be simplified to

$$\frac{\partial U}{\partial w_{ij}^{[l]}} = a_j^{[l-1]} \sigma'_l(z_i^{[l]}) \sum_{k=1}^{m^{[l+1]}} \frac{\partial U}{\partial a_k^{[l+1]}} w_{ki}^{[l+1]} \sigma'_{l+1}(z_k^{[l+1]}). \tag{17}$$

A similar process can be done for the bias, $b_i^{[l]}$ to show that

$$\frac{\partial U}{\partial b_i^{[l]}} = \sigma'_l(z_i^{[l]}) \sum_{k=1}^{m^{[l+1]}} \frac{\partial U}{\partial a_k^{[l+1]}} w_{ki}^{[l+1]} \sigma'_{l+1}(z_k^{[l+1]}). \tag{18}$$

Now, let $\boldsymbol{z}^{[l]}$ be a vector with elements $z_i^{[l]}$, let $\nabla_{\boldsymbol{W}^{[l]}} \boldsymbol{U}$ be a matrix with ij entry $\frac{\partial U}{\partial w_{ij}^{[l]}}$, let $\nabla_{\boldsymbol{b}^{[l]}} \boldsymbol{U}$ be a vector with elements $\frac{\partial U}{\partial b_i^{[l]}}$, and let $\nabla_{\boldsymbol{a}^{[l]}} U$ be a vector with elements $\frac{\partial U}{\partial a_k^{[l]}}$. Then, equations 17 and 18 can be written in the vector form

$$\nabla_{\boldsymbol{W}^{[l]}} \boldsymbol{U} = \sigma'_l(\boldsymbol{z}^{[l]}) \odot [\boldsymbol{W}^{[l+1]^T} (\nabla_{\boldsymbol{a}^{[l+1]}} U \odot \sigma'_{l+1}(\boldsymbol{z}^{[l+1]})] \boldsymbol{a}^{[l-1]^T} \tag{19}$$

and

$$\nabla_{\boldsymbol{b}^{[l]}} \boldsymbol{U} = \sigma'_l(\boldsymbol{z}^{[l]}) \odot [\boldsymbol{W}^{[l]^T} (\nabla_{\boldsymbol{a}^{[l+1]}} U \odot \sigma'_{l+1}(\boldsymbol{z}^{[l+1]})]. \tag{20}$$

Here, $\odot$ is element-wise multiplication. Thus all the derivatives in $\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$ can be computed by first computing the derivatives for the weights and biases in the last layer. Then, the derivatives for the weights and biases in the preceding layers are calculated layer by layer moving backwards via equations 19 and 20. Algorithm 1 shows how to perform backpropa-

gation in detail with inspiration from [6].

Algorithm 1: Backpropagation Algorithm

Input: state vector $\boldsymbol{s}$

Output: gradient of objective function $\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$

set the activation of the input layer, $\boldsymbol{a}^{[1]}$, to $\boldsymbol{s}$

for *each* $l = 2, 3, \ldots, L$ **do**

  compute $\boldsymbol{z}^{[l]} = \boldsymbol{w}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}$ and $\boldsymbol{a}^{[l]} = \sigma_l(\boldsymbol{z}^{[l]})$

end

compute $\boldsymbol{\delta}^{[L]} = \nabla_{\boldsymbol{a}^{[L]}} U \odot \sigma_L{}'(z_i^{[L]})$

for *each* $l = L-1, L-2, \ldots, 2$ **do**

  compute $\boldsymbol{\delta}^{[l]} = \sigma_l{}'(\boldsymbol{z}^{[l]}) \odot ({\boldsymbol{W}^{[l]}}^T \boldsymbol{\delta}^{[l+1]})$

  compute elements of $\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$ via $\nabla_{\boldsymbol{W}^{[l]}} \boldsymbol{U} = \boldsymbol{\delta}^{[l]} {\boldsymbol{a}^{[l-1]}}^T$ and $\nabla_{\boldsymbol{b}^{[l]}} \boldsymbol{U} = \boldsymbol{\delta}^{[l]}$

end

3 Methods

Both a physics based solution to the cartpole environment as well as a solution found from reinforcement learning are implemented. OpenAI Gym, a toolkit for comparing reinforcement learning algorithms, is used for simulating the cartpole problem. Gym includes a simulation of the cartpole problem which integrates Equation 21 and Equation 22 using Euler's method to find the position of the cart, velocity of the cart, angular position of the pendulum, and angular velocity of the pendulum at the next time step.

3.1 Physical Approach

This problem has two degrees of freedom: θ is the angular position of the pendulum and x is the position of the cart. The additional parameters of the problem are l, the length of the pendulum, m_c, the mass of the cart, and m_p, the mass of the pendulum. A free body diagram of the problem is show in Figure 2. The forces exerted on the cart are $\boldsymbol{F}$, the externally applied force, $\boldsymbol{N}$, the normal force between the cart and the pendulum, $\boldsymbol{N}_c$, the normal force between the cart and the track, and $\boldsymbol{G}_c$, the gravitational pull of the Earth on the cart. The forces exerted on the pendulum are $\boldsymbol{N}$, and $\boldsymbol{G}_p$, the gravitational pull of Earth on the pendulum.

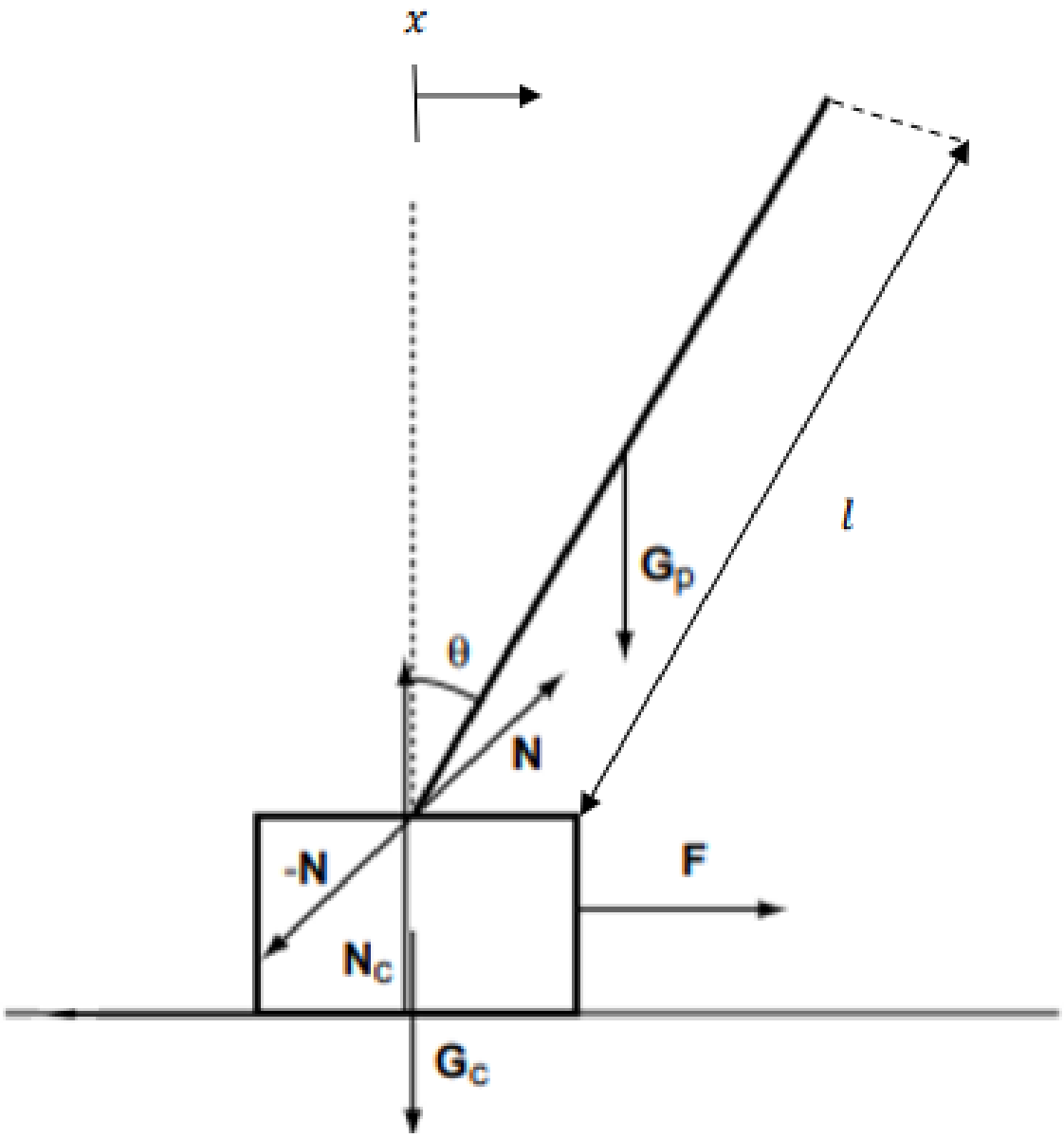


Figure 2: A free body diagram of the cartpole problem.

The dynamics of the cartpole problems without friction can be described with the following two differential equations as shown in [7]:

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta(\frac{-F - m_p l \dot{\theta}^2 \sin\theta}{m_c + m_p})}{l(\frac{4}{3} - \frac{m_p \cos^2\theta}{m_c + m_p})} \tag{21}$$

$$\ddot{x} = \frac{F + m_p l(\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta)}{m_c + m_p} \tag{22}$$

The physical solution implemented solves for the angular velocity by integrating Equation 21 using Euler's method. This is computed for both a rightward and leftward force, then the force which minimizes the angular velocity is executed.

3.2 Reinforcement Learning Approach

3.2.1 REINFORCE Algorithm

The physical solution to the cartpole problem was then used as a benchmark for solutions found through reinforcement learning. The REINFORCE algorithm was used to solve the cartpole problem. Algorithm 2 shows the REINFORCE algorithm. REINFORCE relies on the fact that the gradient of the objective function can by calculated by

$$\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) = \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log[\pi(a_t|s_t, \boldsymbol{\theta})] G_t, \tag{23}$$

as shown in [8]. Here, G_t is the discounted cumulative reward from Equation 3. Remember that $\pi(a_t|s_t)$ is the probability of taking action a_t given s_t. Thus $\pi(a_t|s_t)$ is equal to the value of the activation of the neuron in the output layer of the policy network that corresponds to action a_t. The REINFORCE algorithm was implemented in Python using PyTorch, a library for constructing and training neural networks.

Algorithm 2: REINFORCE Algorithm

Input: policy $\pi(a|s, \boldsymbol{\theta})$

Output: policy $\pi(a|s, \boldsymbol{\theta})$

Parameters: learning rate α, discount factor γ

initialize policy parameter $\boldsymbol{\theta}$

for *true* **do**

  generate a trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_2, \ldots, s_T, a_T, r_T)$ from $\pi(a|s, \boldsymbol{\theta})$

  for *each step* t *of* τ **do**

    $G_t \leftarrow \sum_{i=t}^{T} \gamma^{i-t} r_t$

  end

  $\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) \leftarrow \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log[\pi(a_t|s_t, \boldsymbol{\theta})] G_t$

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$

end

3.2.2 Policy Network

The chosen policy network consists of an input layer, one hidden layer, and an output layer. The input layer has four neurons, as the state space of the cartpole problem is $\mathbb{R}^4$. The hidden layer has 128 neurons, and the output layer has two neurons as the action space consists of only a unit leftward or rightward force on the cart. The activation function for the hidden layer is the ReLU function, $f(x) = \max[x, 0]$, and the activation function for the output layer is the softmax function, $f(\boldsymbol{x})_i = \frac{e^{\boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{x}_j}}$. The softmax function guarantees that the sum of the activations in the output layer is 1. The activations of the output layer need to sum to one because the output layer outputs a probability distribution of the action space. A diagram of the policy network is shown in Figure 3.

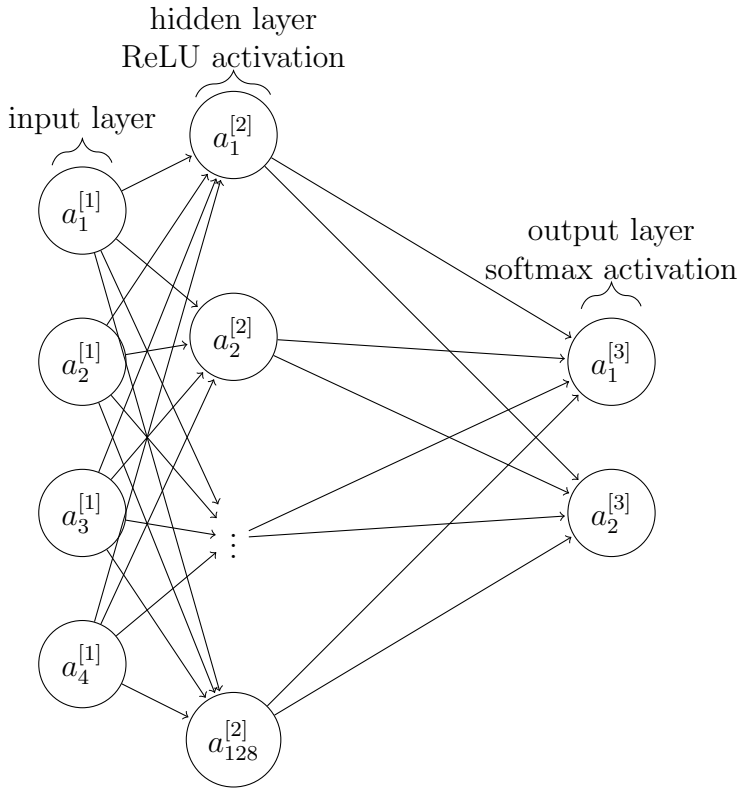


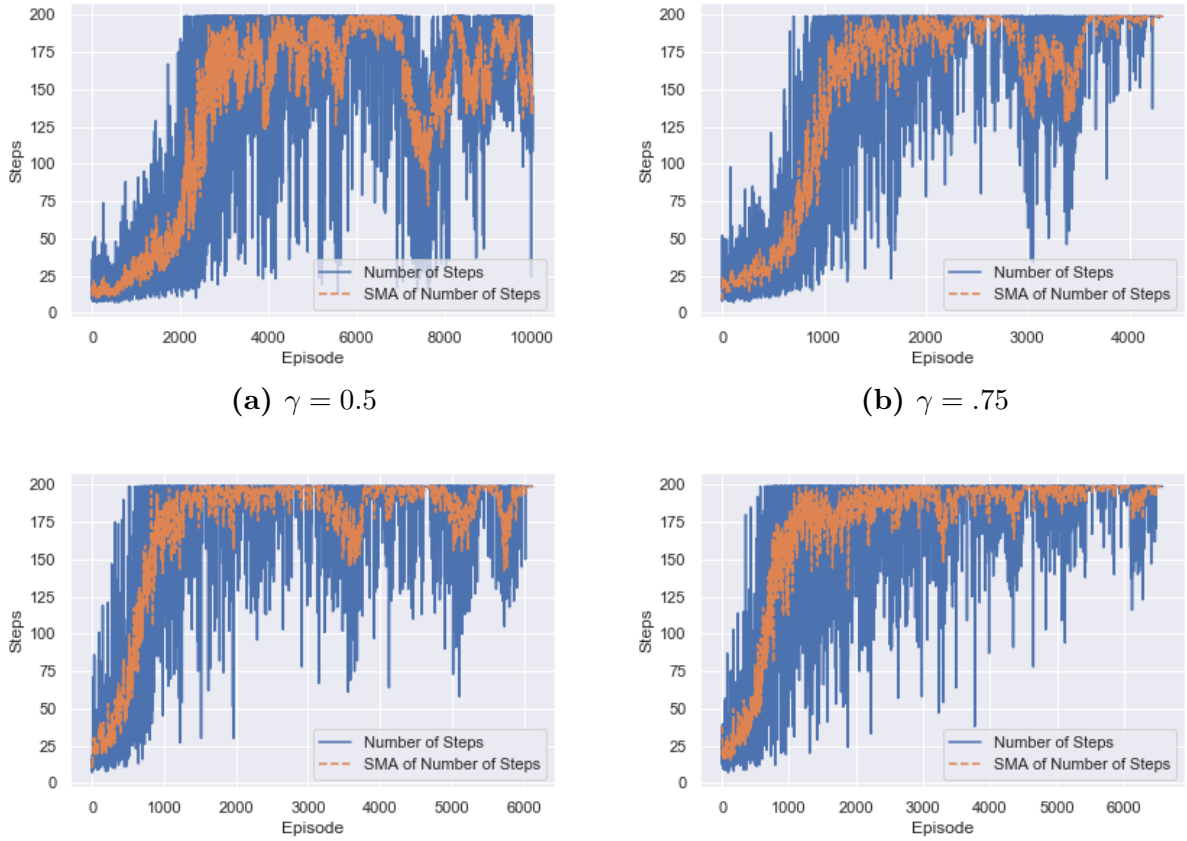
Figure 3: Architecture of the policy network used in solving the cartpole problem. Notice that the input layer and output layer have the same dimensions as the state space and action space respectively.

4 Results and Discussion

OpenAI Gym considers their cartpole problem environment to be solved when an algorithm can achieve an average reward of 195.0 over 100 trials. The physical approach to the cartpole problem which minimized the angular velocity of the cartpole pendulum achieved an average reward of 193.17 $\pm$ 12.90. The REINFORCE algorithm achieved an average reward of 200.0 $\pm$ 0.0 after training for 6,579 episodes. It took the REINFORCE algorithm 8.86 minutes to train.

4.1 Visualizations of Learning Curves

Figure 4 shows the number of time steps the cartpole was successfully stabilized during each episode of the learning process. It also shows a simple moving average of the number of steps achieved during each episode. A simple moving average is the average number of steps achieved over an arbitrary number of most recent training episodes. Training episodes never last longer than 200 steps by design. Therefore, 200 is the maximum reward that can be achieved during a single training episode.

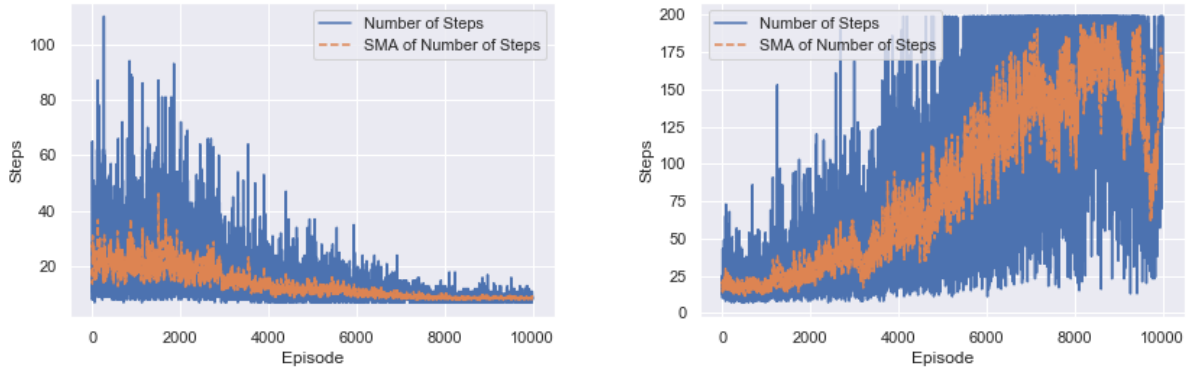


(a) $\gamma = 0.5$ **(b)** $\gamma = .75$

(c) $\gamma = 0.9$ **(d)** $\gamma = 1.0$

Figure 4: The learning curves of the REINFORCE algorithm are shown above. The blue curve is the number of steps the algorithm achieved during each episode of training. The orange curve is a 10 episode long simple moving average of the number of steps achieved. Each graph shows the learning curve for a different discount factor, γ. The higher discount factors achieve a higher number of steps over fewer episodes of training.

As seen in Figure 4, the discount factor, γ, has a great affect on the learning process. Lower discount factors result in the REINFORCE algorithm taking more training episodes to achieve near perfect rewards (200 steps). In fact, when using small enough discount factors, it appears the training process will never stabilize the pendulum for more than a couple steps. It seems that the REINFORCE algorithm must consider the effect of future rewards and immediate rewards fairly equally to have the most success. Figure 5 shows the learning curves of the REINFORCE algorithm for lower discount factors. It can be seen that the training process fails with small enough discount factors.



(a) $\gamma = 0.1$ **(b)** $\gamma = .25$

Figure 5: The learning curves of the REINFORCE algorithm are shown above for small γ. When $\gamma = 0.1$, the REINFORCE algorithm fails to stabilize the pendulum even for a few time steps. For $\gamma = .25$, the REINFORCE algorithm does not achieve a perfect score even after 10,000 training episodes.

4.2 Comparison to Leading Algorithm

The leading algorithm for the cartpole problem on OpenAI Gym achieved an average reward of 200.00 ± 0.00 after 306 episodes. This algorithm took 12 minutes to train. The REINFORCE algorithm competes well with the leading algorithm in terms of average reward, and takes only 74% of the time to train. Still, the leading algorithm takes less than 5% of the training episodes of REINFORCE to achieve the perfect average reward.

5 Conclusions

A priori physical solutions do not appear to solve the cartpole problem to OpenAI Gym's standards. Solutions found through reinforcement learning can achieve better average rewards than an algorithm which solely aims to minimize the angular velocity of the cartpole pendulum. It should be noted that the a priori physical solution used made no attempts to minimize the velocity of the cart. Perhaps the success of reinforcement learning over the physical approach can be attributed to the physical solution minimizing the angular velocity of the pendulum, but not the velocity of the cart. A physical solution which aims to minimize both the angular velocity of the pole and the velocity of the cart should be used as a more competitive benchmark in future research.

It should be noted that reinforcement learning takes some tweaking and experimenting to properly solve an environment. A reinforcement learning algorithm and policy network architecture need to be chosen during the process of solving the cartpole problem. Altering the algorithm used or architecture of the policy network could affect the reward achieved after training. In addition, hyper parameters such as the discount factor and learning rate need to be tuned to optimize the reward. The ability to tune hyper parameters to better solve a problem with reinforcement learning is one of the strengths of using the class of techniques. It was found that higher discount factors lead to higher average rewards, with a discount factor of one yielding the perfect average reward. This suggests that considering the trajectory of the cartpole many time steps in the future is just as important as considering time steps in the immediate future when looking to obtain a higher average reward.

Investigating the sensitivity of the REINFORCE algorithm's success in solving the cartpole problem to the discount factor suggests something about the physical nature of the cartpole problem. It suggests that understanding how a sequence of forces of many time steps into the future affects the long term trajectory of the cartpole. Since the REINFORCE algorithm fails to stabilize the cartpole when using low discount factors, clearly the rewards at time steps in the future cannot be disregarded. Perhaps the success or failure of a reinforcement learning algorithm can indicate someting about the nature of the problem it is being applied to. The result is an intriguing juxtaposition to the physical solution used on

the cartpole problem which only relies on information from the underlying physical model one step into the future. The REINFORCE algorithm needs information on the system for many time steps into the future while an engineered solution only needs information from one time step into the future.

Nonetheless, it is astonishing that a trained neural network can control the cartpole better than an algorithm created by a scientist. For more complicated control problems, such as bipedal motion, it is likely that any control process created by a scientist or engineer will fall short of what can be found by reinforcement learning. As the dynamics of a system to control become more complicated, reinforcement learning should be greatly considered to address the control problem.

6 Acknowledgements

I want to thank Professor Guenter Schneider for advising me in this research project.

7 References

References

1. Siekmann, J. *et al. Learning Memory-Based Control for Human-Scale Bipedal Locomotion* 2020. arXiv: 2006.02402 [cs.RO].

2. Kormushev, P., Calinon, S. & Caldwell, D. G. Reinforcement Learning in Robotics: Applications and Real-World Challenges. *Robotics* **2,** 122–148. ISSN: 2218-6581. https://www.mdpi.com/2218-6581/2/3/122 (2013).

3. Kober, J., Bagnell, J. A. & Peters, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* **32,** 1238–1274. eprint: https://doi.org/10.1177/0278364913495721. https://doi.org/10.1177/0278364913495721 (2013).

4. Lillicrap, T. P. *et al. Continuous control with deep reinforcement learning* 2019. arXiv: 1509.02971 [cs.LG].

5. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)* **2,** 303–314. ISSN: 0932-4194. http://dx.doi.org/10.1007/BF02551274 (Dec. 1989).

6. Nielsen, M. A. *Neural Networks and Deep Learning* misc. 2018. http://neuralnetworksanddeeplearning.com/.

7. Florian, R. Correct equations for the dynamics of the cart-pole system (Aug. 2005).

8. Li, Y. *Deep Reinforcement Learning* 2018. arXiv: 1810.06339 [cs.LG].