Transparent Processing of Neural Networks in Industrial Control

Jiaren Xu [a], Chun Dong [b, *]

School of Electrical Engineering, Beijing Jiaotong University, Beijing 100044, China

[a]17121515@bjtu.edu.cn, [b] chdong@bjtu.edu.cn

*Corresponding author

Keywords: DRL, Linear interpolation, Control.

Abstract: This paper introduces a method that can approximate the neural network as a variable-gain linear feedback controller, which enables the neural network to achieve parameter transparency and parameter interpretability in the application phase. The approximation method is independent of the structure of the neural network, and the learning process is completely consistent with the deep reinforcement learning. Only the learned neural network is approximated in multiple intervals, and the number and range of approximate intervals can be arbitrarily selected.

1. Introduction

The linear feedback controller is a commonly used control method in industrial control because it has the advantages of less parameters and transparency of the control process. However, in some more complicated systems, in order to improve the control effect of the controller, a variable gain linear feedback controller is usually used. It inherits the advantage of linear feedback control process transparency, but due to the increase of parameters, the adjustment process becomes more difficult.

Although deep reinforcement learning also performs well in the field of automatic control, one of the biggest obstacles to applying neural networks to industrial systems is its black box properties. This means that it is difficult for engineers to interpret the relationship between input characteristics and their inputs, which makes it impossible for engineers to know if a neural network is vulnerable. Even during the experiment, some loopholes in the neural network were discovered. At present, there are no effective repair methods. Only by learning more data in the neural network, it is not necessarily effective. Therefore, in the industrial control system, the transparency and interpretability of the model is very important. Only the model can explain the engineers to effectively carry out the vulnerability investigation to ensure the security of the model.

In fact, opacity and uninterpretation are generated by neural networks, so we tried to use linear models instead of neural networks, and then combined with reinforcement learning to learn parameters. This depends on the advantages of both parameter transparency and interpretability, as well as the advantages of automatic tuning. But in Chapter 3 we can see that after replacing the neural network with a linear model, the learning speed will become very slow and even impossible to learn the desired effect. In the method proposed in this paper, we still use neural network combined with reinforcement learning to learn. But after the learning is completed, we use the integral gradient method to extract multiple feedback matrices in the neural network, and then use linear interpolation to integrate the matrices to obtain a transparent and interpretable model (such as figure 1). Since the matrix used in the linear interpolation process corresponds to the feedback matrix, the meaning of the parameters of this method is very clear, so we can also make artificial improvements to the parameters.

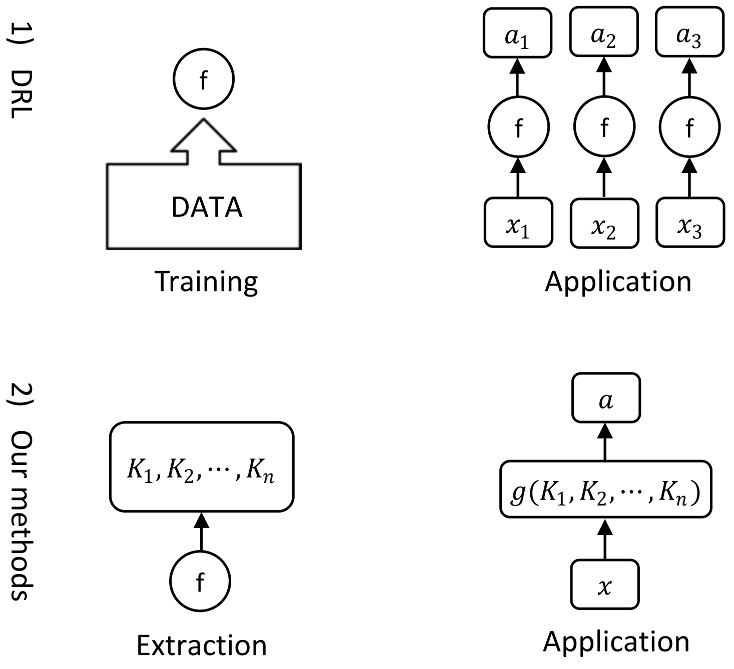


Figure 1. 1) DRL. 2) Our methods.

2. Deep Reinforcement Learning

Reinforcement learning is an area of machine learning that emphasizes how agents act on the environment to maximize the expected return. At time t, there is state s_t,and the agent selects an action a_t according to the given policy $\pi(a_t|s_t;\theta)$. This action is fed back to the environment in which the agent is located, according to the transition probability $p(s_{t+1}|s_t,a_t)$ of the environment gets a new state s_{t+1}, and the agent gets the reward $r(s_t,a_t)$, after which the process continues. Define R_t to represent the cumulative return from t to infinity and γ as the discount factor: $R_t=\sum_{t'=t}^{\infty}\gamma^{t'-t}r(s_{t'},a_{t'})$, where $\gamma\in(0,1)$. The goal of reinforcement learning is to maximize the expected return $J(\theta)=E_{\pi_\theta}[R_t]$ with θ as the strategic parameter of the agent. In deep learning, the agent's policy $\pi(a_t|s_t;\theta)$ is a neural network. In this chapter, we introduce a parameter optimization method based on the policy gradient method.

2.1 Policy Gradient.

Since the agent's policy $\pi(a_t|s_t;\theta)$ is about the parameter θ, it can optimize the parameters by the gradient rise method on the expected return J(θ). According to the REINFORCE algorithm (Williams, 1992), the gradient of J(θ) with respect to θ is given by:

$$\nabla_\theta J(\theta)=\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\nabla_\theta\log\pi_\theta(a_t|s_t)\,\gamma^t R_t\right]=\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\nabla_\theta\log\pi_\theta(a_t|s_t)\,\gamma^t(R_t-b(s_t))\right]\quad(1)$$

Where $b(s_t)$ is the baseline. Let $\rho_\pi(s)=\sum_{t=0}^{\infty}\gamma^t p(s_t=s)$, (1) can be rewritten as:

$$\nabla_\theta J(\theta)=\mathbb{E}_{s_t\sim\rho_\pi(\cdot),a_t\sim\pi(\cdot|s_t)}[\nabla_\theta\log\pi_\theta(a_t|s_t)\,(R_t-b(s_t))]\quad(2)$$

However, in this method, the update of the gradient takes a long time to affect the state to which the small value is given, and empirically, a poor learning effect is produced. Therefore, in most of the policy gradient methods, the non-discount state distribution is used instead, that is, γ=1 in the equation $\rho_\pi(s)$. The replacement update method is equivalent to maximizing the average return of the strategy, even if R_t is the accumulated discount reward.

The expected calculation is to use Monte Carlo sampling for estimation. In most cases, the state value function $V_\pi(s_t)$ is used as the baseline, and $Q_\pi(s_t,a_t)$ is used instead of R_t, so that R_t-$b(s_t)$ can be replaced by $A_\pi(s_t,a_t)$, which is defined as follows:

$$V_\pi(s_t)=\mathbb{E}_\pi[R_t]=\mathbb{E}_{\pi_\theta(a_t|s_t)}[Q_\pi(s_t,a_t)]\quad(3)$$

$$Q_\pi(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_\pi[R_{t+1}] = r(s_t,a_t) + \gamma\mathbb{E}_{p(s_{t+1}|s_t,a_t)}[V_\pi(s_{t+1})] \tag{4}$$

$$A_\pi(s_t,a_t) = Q_\pi(s_t,a_t) - V_\pi(s_t) \tag{5}$$

In fact, there are many improvements in the policy gradient method. In this paper, we use an improved method proposed by PPO (2017, OpenAI) [7]. The improved gradient is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_{\theta_{old}}}[\nabla_\theta \min(r_t(\theta)A_\pi(s_t,a_t), clip(r_t(\theta),1-\epsilon,1+\epsilon)A_\pi(s_t,a_t))] \tag{6}$$

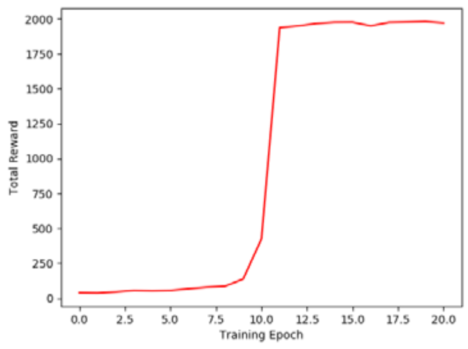
Where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, $\epsilon \in (0,1)$.

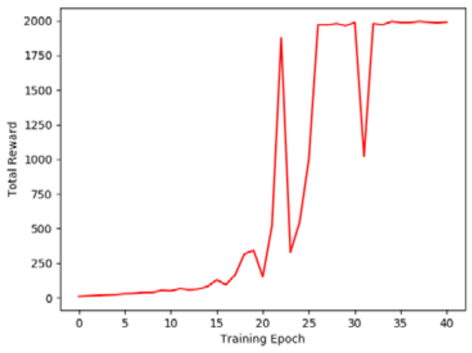
$A_\pi(s_t,a_t)$ is estimated using $\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T\text{-}t+1}\delta_{T\text{-}1}$, where T is the terminal time, $\lambda \in (0,1)$. And $\delta_t = r(s_t,a_t) + \gamma V_w(s_{t+1}) - V_w(s_t)$, where $V_w(s_t)$ is an estimate of $V_{\pi_{\theta_{old}}}(s_t)$. In most cases $V_w(s_t)$ is a neural network whose parameters are updated using the gradient descent method. The gradient formula is as follows:

$$\nabla_w(V_w(s_t) - \sum_{t'=t}^{\infty}\gamma^{t'-t}r(s_{t'},a_{t'}))^2 \tag{7}$$

3. Methods and Simulation

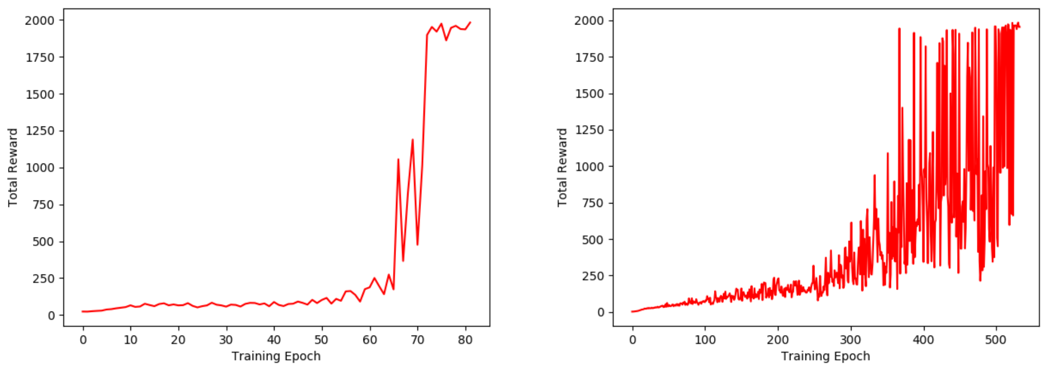
On the first-order pendulum and the second-order pendulum, we use the neural network as the policy and the linear model as the policy. Figure 2 shows the learning effect of the first-order pendulum of the neural network and the second-order pendulum. Figure 3 is the first-order pendulum of the linear model, and the learning effect diagram on the second-order pendulum. The horizontal axis represents the number of actions of the agent, and the vertical axis represents the round score of the agent. The higher the score, the better the control effect. As can be seen from Figure 2 and Figure 3, if the neural network is directly replaced by a linear model, it will take more time to learn, and even cannot learn a policy to make the system stable. In order to maintain the efficiency of the process and to achieve interpretability, our method is to use the neural network in the learning process. After the neural network has been learned, the characteristics of the neural network (corresponding to the feedback matrix in the control system) are extracted. Linear interpolation is performed to obtain a linear feedback model with variable gain. Since the coefficients in the matrix have a clear physical meaning, we can combine the engineering experience to make some small manual adjustments to the feedback matrix extracted by the neural network.





(a) (b)

Figure 2. a) 1st-order with NN. b) 1st-order with linear.



(a) (b)

Figure 3. a) 2nd-order with NN. b) 2nd-order with linear.

3.1 Feature Extraction

In this paper we use a method called integral gradient (Mukund, 2017) [10] for feature extraction, the formula is as follows:

$$f(s) = \int_0^1 \nabla_s f(s)|_{s=s_0+t(s-s_0)} \cdot (s - s_0)\, dt + f(s_0) \tag{8}$$

$$f(s) \approx \frac{1}{m}\sum_{k=1}^{m} \nabla_s f(s)|_{s=s_0+\frac{k}{m}(s-s_0)} \cdot (s - s_0) + f(s_0) \tag{9}$$

This method was first proposed to analyze the extent to which each pixel of the input image affects neural network prediction. It can be seen that the above equation has the structure of $f(s) = -K(s)(s_0 - s) + b$, where $K(s) = \int_0^1 \nabla_s f(s)|_{s=s_0+t(s-s_0)}\, dt$, $b = f(s_0)$.

It is worth noting that as long as s_0 is within the definition domain, the equation $f(s) = K(s)s + b$ holds. In all of the following, we assume that $s = s_0$ is the equilibrium point, let $e = s_0 - s$, and get $f(s) = -K(s)e + b$.

From the above formula, we can construct $f_1(s)$ and $f_2(s)$ as approximation functions of f(s) in the vicinity of $s = s_1$ and $s = s_2$, as follows:

$$f_1(s) = -K(s_1)e + b \tag{10}$$

$$f_2(s) = -K(s_2)e + b \tag{11}$$

Regardless of the point at which the approximation is made, $K(s_1)$ and $K(s_2)$ represent the degree of influence of the function on the variation of the error e, ie the feedback matrix of the system. In engineering, we can easily determine how the controller is controlled by the feedback matrix, because each coefficient in the feedback matrix corresponds to the weight of the controller's change to this state. The feedback matrix can be approximated by $K(s) \approx \frac{1}{m}\sum_{k=1}^{m} \nabla_s f(s)|_{s=s_0+\frac{k}{m}(s-s_0)}$, where The larger the m, the higher the degree of approximation.

In order to get a better approximation, we will approximate in multiple intervals. However, the approximated function is discontinuous. This discontinuity may cause jitter in the system during control. In this paper, we will use linear interpolation methods to eliminate this discontinuity, and linear interpolation does not add opacity to the approximation process.

3.2 Linear Interpolation

Linear interpolation is a commonly used method for processing the scaling of an image. Unlike the general interpolation method, we interpolate $K(s_1), K(s_2), \cdots$, instead of $f(s_1), f(s_2), \cdots$.

Assume $s = [s^1, s^2, \cdots, s^n]^T \in R^{n\times 1}$,and $s^i \in [s^i_1, s^i_2]$, $s^i_2 \geq s^i_1$, where $i = 1,2,\cdots$, we have:

$$\widetilde{K}(s) = \sum_{j_1=1,j_2=1,\cdots,j_n=1}^{2} K(s^1_{3-j_1}, s^2_{3-j_2}, \cdots, s^n_{3-j_n}) \prod_{i=1}^{n} \frac{(s^i - s^i_{j_i})(3-2j_i)}{s^i_2 - s^i_1} \tag{12}$$

$$b = f(s_0) \tag{13}$$

$$\tilde{f}(s) = \widetilde{K}(s)s + b \tag{14}$$

$\tilde{f}(s)$ is an estimate of $f(s)$. In the application, we need to divide several intervals in each dimension of the state s in advance, that is, several groups s_2^i, s_1^i. When calculating $\widetilde{K}(s)$, the closest s_2^i, s_1^i from s^i is used.

Considering the case where the stable point is $s = s_0$, in most stable balanced systems (such as inverted pendulum), there will be $f(s_0) = 0$, but since $f(s)$ is the result of neural network, $f(s_0)$ does not necessarily equal 0, but we can force $b = 0$ so that $\tilde{f}(s_0) = 0$.

3.3 Simulation

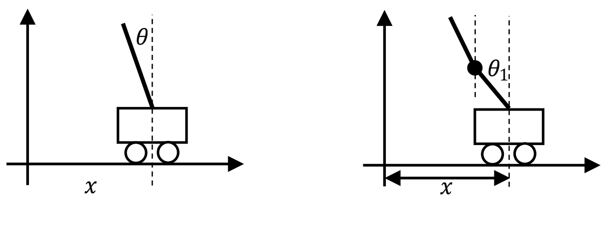
We compare the first- and second-order pendulums with the well-known neural network and the methods described in this article. A schematic diagram of the first-order pendulum and the second-order pendulum is shown in Figure 4 In the first-order pendulum, the state variable is $s = [x, \theta, \dot{x}, \dot{\theta}]^T$, the reward $r_t = 1\text{-}x_t^2\text{-}\theta_t^2\text{-}0.01\dot{x}_t^2\text{-}0.01\dot{\theta}_t^2$. The structure of the neural network is: [4, 256], [256, 2], the feedback matrix and the comparison of control effects are shown in Figures 3.5 and 3.6. The division of the interval is as follows:

$$x = \{-1, -0.5, 0, 0.5, 1\}, \theta = \{-16, -8, 0, 9, 16\}, \dot{x} = \{-2, -1, 0, 1, 2\}, \dot{\theta} = \{-36, -18, 0, 18, 36\}$$

In the second-order pendulum, the state variable is $s = [x, \theta_1, \theta_2, \dot{x}, \dot{\theta}_1, \dot{\theta}_2]^T$, reward $r_t = 1\text{-}x_t^2\text{-}\theta_{1,t}^2\text{-}\theta_{2,t}^2\text{-}0.01\dot{x}_t^2\text{-}0.01\dot{\theta}_{1,t}^2\text{-}0.01\dot{\theta}_{2,t}^2$. The structure of the neural network is: [6, 256], [256, 128], [128, 2], the feedback matrix and the comparison of control effects are shown in Figures 3.7 and 3.8. The division of the interval is as follows:

$$x = \{0\}, \theta_1 = \{-20, -16, -12, -8, -4, 0, 4, 8, 12, 16, 20\},$$
$$\theta_2 = \{-30, -24, -18, -12, -6, 0, 6, 12, 18, 24, 30\},$$
$$\dot{x} = \{0\}, \dot{\theta}_1 = \{0\}, \dot{\theta}_2 = \{0\}$$

In the system of the second-order inverted pendulum, we only interpolate θ_1, θ_2, which is designed according to engineering experience. Such an approach can greatly reduce the number of parameters and the amount of calculation, which is convenient for analysis.



(a) (b)

Figure 4. a) 1st-order pendulum. b) 2nd-order pendulum.

As can be seen from Figure 5 and Figure 7, the feedback matrix obtained by linear interpolation differs from the feedback matrix extracted directly from the neural network, but the general trend is the same. As can be seen from Figures 6 and 8, the control effect of the proposed method is comparable to that of the neural network. When the neural network is stable, it is not guaranteed to stop at $x = 0$ (may be achieved by adjusting the parameters of the reward function). In our method, since we have performed manual tuning, let $b = 0$, so we can stop at $x = 0$ when the system is stable.

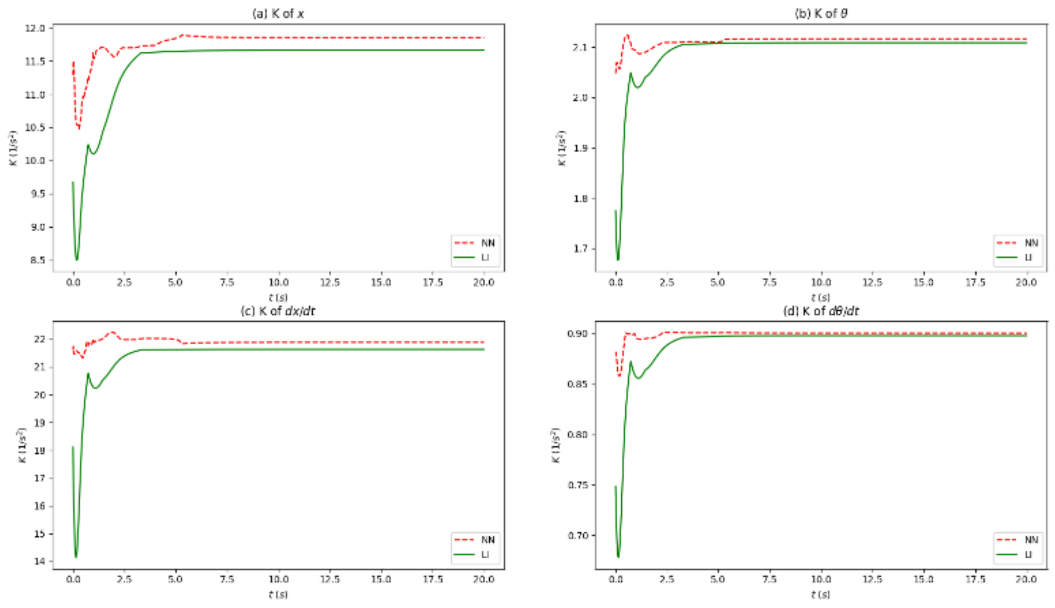


Figure 5. Feedback matrix of 1st-order

In the control data of the same set of first-order inverted pendulum, the feedback matrix K extracted by the neural network using the integral gradient is compared with the image obtained by linear interpolation. The red dotted line is the neural network, and the green solid line is the method of this paper.

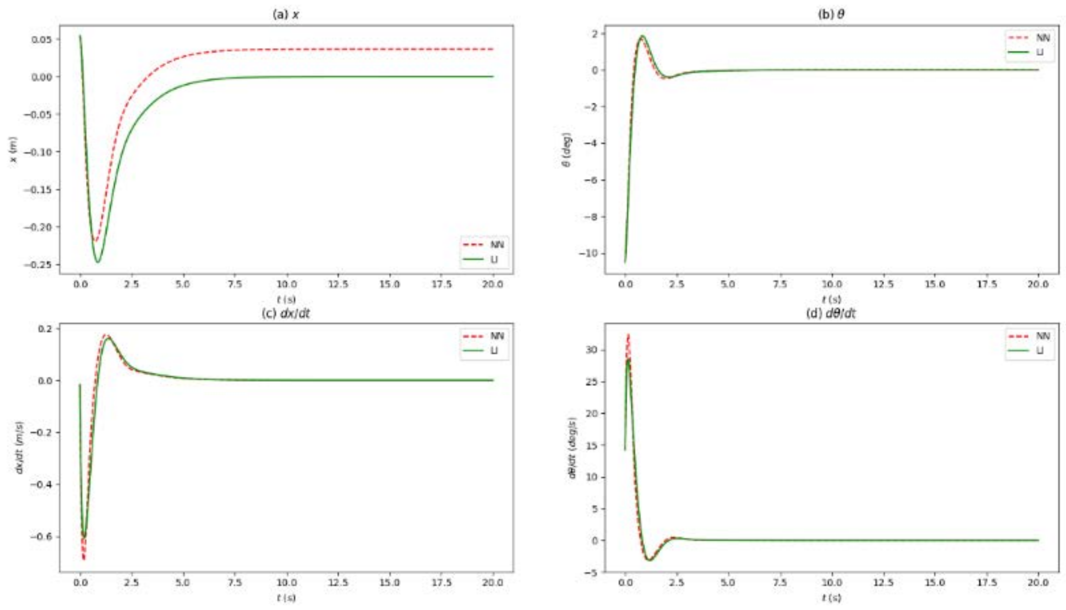


Figure 6. State of 1st-order

In the first-order inverted pendulum system, starting with the same initial state, the control effect of the neural network is compared with the control effect of the method in this paper. The red dotted line is the neural network, and the green solid line is the method of this paper.

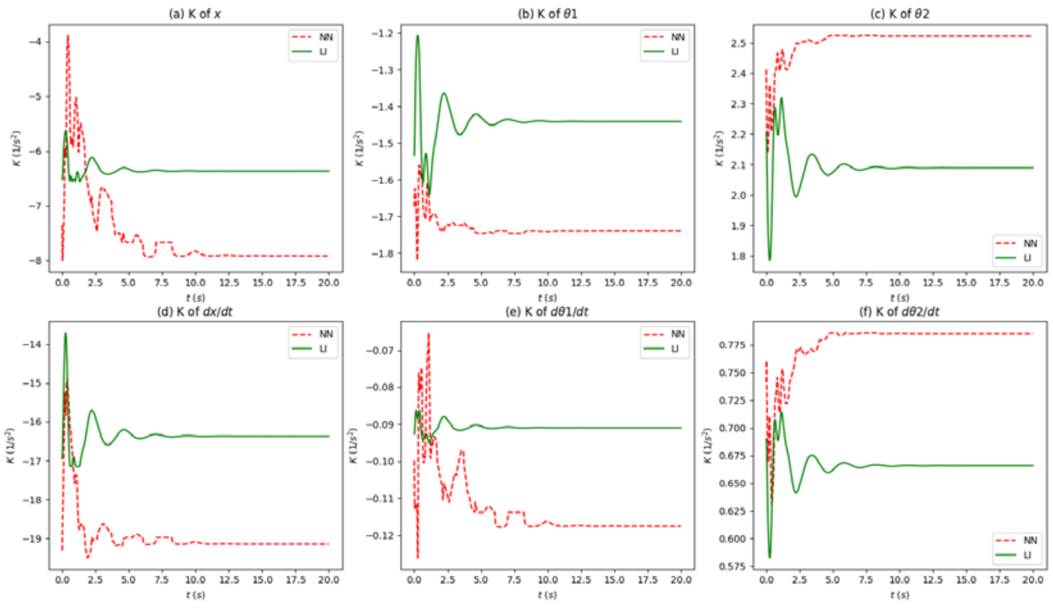


Figure 7. Feedback matrix of 2nd-order

In the control data of the same set of second-order inverted pendulum, the feedback matrix K extracted by the neural network using the integral gradient is compared with the image obtained by linear interpolation. The red dotted line is the neural network, and the green solid line is the method of this paper.

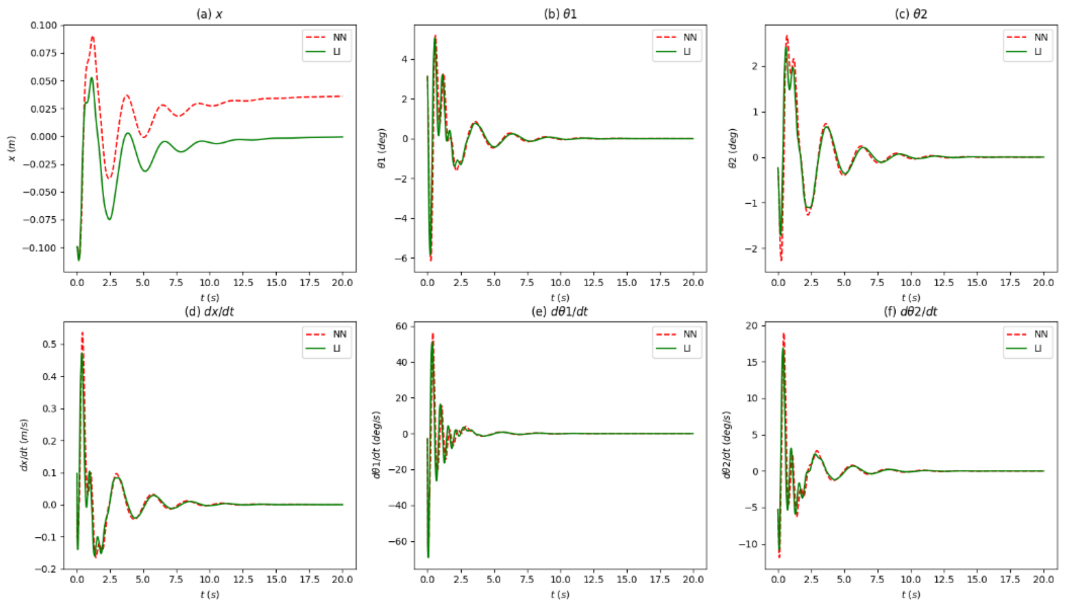


Figure 8. State of 2nd-order

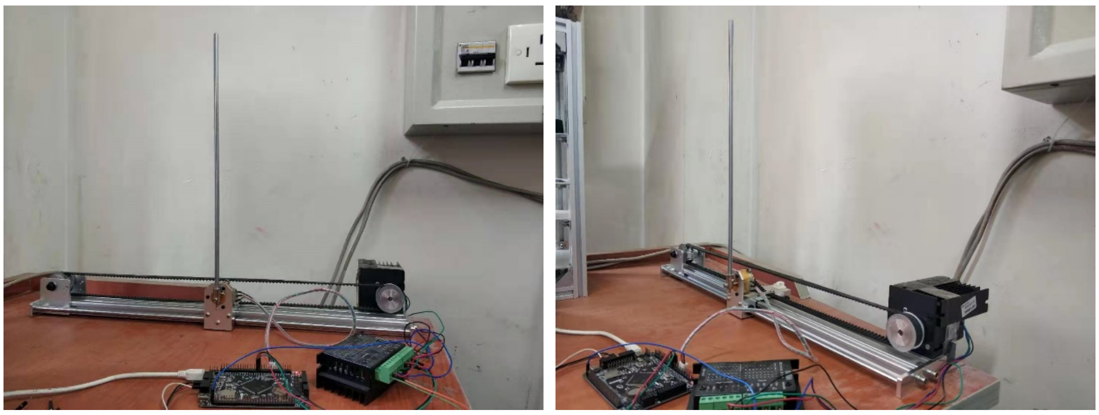
In the second-order inverted pendulum system, starting with the same initial state, the control effect of the neural network is compared with the control effect of the method in this paper. The red dotted line is the neural network, and the green solid line is the method of this paper.

4. Experiment

In this chapter, we apply the controller from the method of this article to the real inverted pendulum. Since the parameters of the modeling are based on the actual inverted pendulum, we directly control the inverted pendulum directly from the controller obtained from the simulation environment.

4.1 Hardware

The motor we use is a 42-stepper motor, model 42HZ260-1684D5. The angle sensor used is a potentiometer, model number WDD35D4-5K. The controller module used is STM32 and the model number is STM32F407ZG. The overall physical map is shown in Figure 9.



(a) (b)

Figure 9. a) Front view of the inverted pendulum. b) Side view of the inverted pendulum

4.2 Waveform

Since the potentiometer can only measure the angle, the angular velocity cannot be measured, and the data measured by the potentiometer has a large noise. So, in actual use, we added a Kalman filter with the inverted pendulum as a model. The state $(x, \theta, \dot{x}, \dot{\theta})$ input to the controller is the state after Kalman filtering. The waveform of the state during the control process is shown in Figure 10.

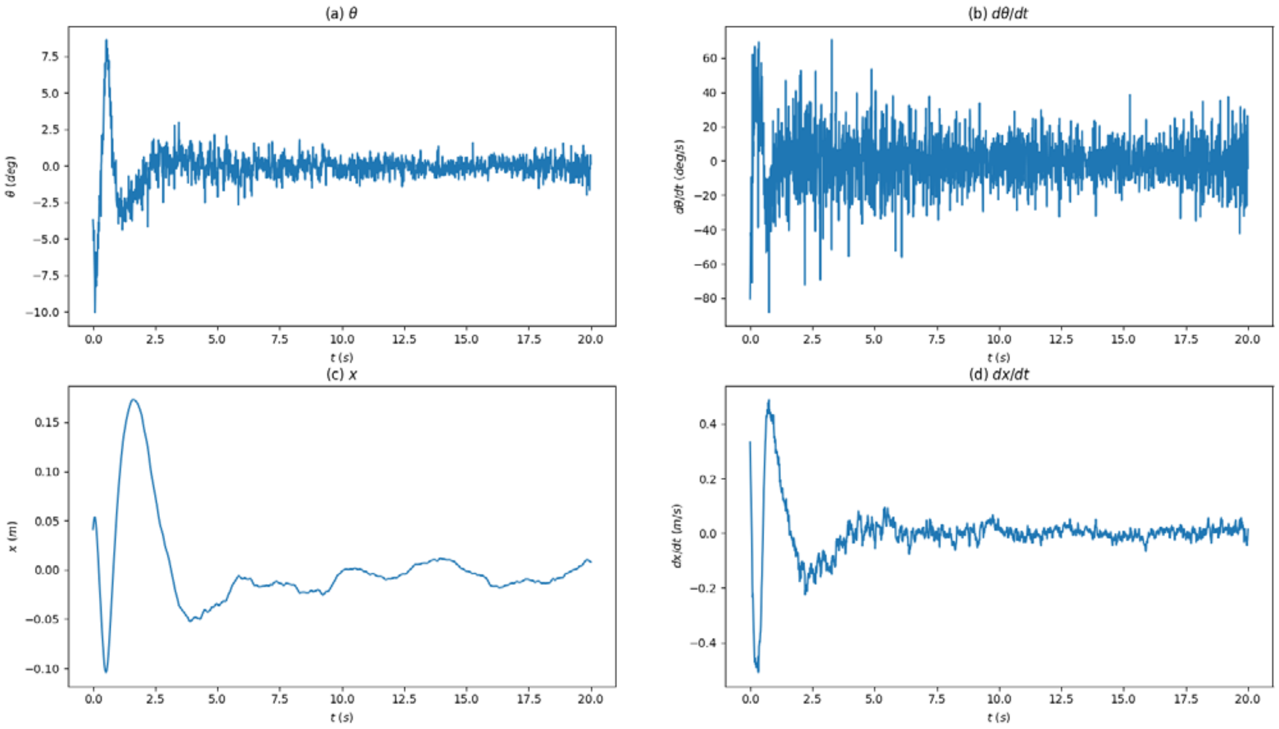


Figure 10. Control waveform.

As can be seen from Figure 10, our controller is able to achieve stable balance control. After a disturbance to the angle of the inverted pendulum, the controller can return the inverted pendulum back to the balance point. And as we get in the simulation, we can stop at x=0.

5. Summary

Through the simulation and physical control waveforms of the previous chapters, we can see that our method can achieve stable balance control. And we have inherited the advantages of deep reinforcement learning and have the ability to interpret. Our approach allows manual tuning and parameters have clear physical and engineering implications. The most straightforward example is that in Chapter 3 we removed the offset so that our controller could stop the inverted pendulum at x=0, and the neural network corresponding to the method could not stop at x=0. And the control effect of our method is close to that of the original neural network, which shows that our method has a higher approximation to the original neural network. More notably, since our method is equivalent to a piecewise linear controller, we can directly analyze its stability theoretically, and the neural network directly obtained by using deep reinforcement learning cannot be analyzed. Stability. Of course, our method requires some manual experience when the dimension is high, mainly in the division of state. However, this has greatly saved our development time compared to manual tuning parameters.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver. et al. Playing Atari with Deep Reinforcement Learning. 2013.

[2] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015) doi:10.1038/nature14236.

[3] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]// International Conference on International Conference on Machine Learning. JMLR.org, 2014.

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel. et al. Continuous control with deep reinforcement learning. 2015.

[5] Mnih V, Badia, Adrià Puigdomènech, Mirza M, et al. Asynchronous Methods for Deep Reinforcement Learning [J]. 2016.

[6] Schulman J, Moritz P, Levine S, et al. High-Dimensional Continuous Control Using Generalized Advantage Estimation[J]. Computer Science, 2015.

[7] Schulman J, Levine S, Moritz P, et al. Trust Region Policy Optimization[J]. 2015.

[8] Schulman, John, Wolski, Filip, Dhariwal, Prafulla. et al. Proximal Policy Optimization Algorithms [J].

[9] Gu S, Lillicrap T, Ghahramani Z, et al. Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic[J]. 2016.

[10] Mukund Sundararajan, Ankur Taly, Qiqi Yan. Axiomatic Attribution for Deep Networks. 2017.

[11] Fazel M, Ge R, Kakade S M, et al. Global Convergence of Policy Gradient Methods for the Linear Quadratic Regulator[J]. 2018.