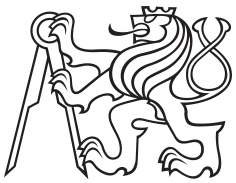
Bachelor Project



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Online Adaptive Control Using Neural Networks

Jan Švrčina

Supervisor: Ing. Teymur Azayev
Field of study: Cybernetics and Robotics
August 2020

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Švrčina Jan** Personal ID number: **474447**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Online Adaptive Control Using Neural Networks

Bachelor's thesis title in Czech:

Adaptivní řízení pomocí neuronových sítí

Guidelines:

The environment in which a system controller is deployed often changes due to inaccurate modeling, parameter drift or external disturbances. One way to counter this is to design a robust controller which works for a variety of system parameters. Another way is to adapt the controller on the fly. The advantage of the latter approach is the potential to handle larger system parameter deviations and a superior performance in the long term.
The student is expected to learn an adaptive neural network control policy using reinforcement learning. The neural network has to be a temporal model which can adapt to system changes by observing the recent episode history. The student can either use an existing state of the art implementation or provide his/her own. The method will be tested on a simple system such as an inverted pendulum and demonstrated in simulation. The results should be compared with several classical controllers such as linear control, MPC or H-inf.

Bibliography / sources:

[1] F.L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," IEEE Circuits & Systems Magazine, Invited Feature Article, pp. 32-50, Third Quarter 2009.
[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov , Proximal Policy Optimization Algorithms, arxiv

Name and workplace of bachelor's thesis supervisor:

Ing. Teymur Azayev, Vision for Robotics and Autonomous Systems, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **07.01.2020** Deadline for bachelor thesis submission: **14.08.2020**

Assignment valid until: **30.09.2021**

Ing. Teymur Azayev	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I thank the CTU for being such a good *alma mater*, my supervisor Ing. Teymur Azayev, for the provided consultations, materials and information and my mother, for lifelong support.

Děkuji ČVUT, že mi je tak dobrou *alma mater*, svému vedoucímu Ing. Teymurovi Azayevovi, za poskytné konzultace, materiály a informace, a své mamince, za celoživotní podporu.

Declaration

I declare that I have written the submitted work independently and that I have listed all the used literature.

In Prague, 14. August 2020

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 14. srpna 2020

Abstract

The online adaptive control of parameter varying systems is a persisting problem in the field of control engineering. Parameter varying systems are dynamic systems, whose properties depend on unobservable variables, unknown to the controller. Classic control methods often fail on this task, as they are too inaccurate or too computationally demanding. Deep reinforcement learning [1] offers a way of acquiring fast online-adaptive policies, very closely approximating the optimal policy. It becomes even more useful, when the dimensionality of the system, such as in robotic manipulation, becomes too big for classic methods to handle. This thesis demonstrates the adaptability of neural network policies [2] in the control of parameter varying systems, on its simulation of a non-linear system. It also benchmarks these policies against three fully non-linear MPC controllers, which should achieve almost optimal behaviour in this setting. In addition, an LSTM network estimator [3] of the unobservable variables is created and used in the control policy.

Keywords: adaptive control, neural networks, reinforcement learning

Supervisor: Ing. Teymur Azayev
ČVUT v Praze
Fakulta elektrotechnická
Katedra kybernetiky
Karlovo náměstí 13
121 35 Praha 2

Abstrakt

Online adaptivní řízení systémů s proměnlivými parametry je přetrvávající problém v odvětví řídící techniky. Systémy s proměnlivými parametry jsou dynamické systémy, jejichž vlastnosti jsou závislé na nepozorovatelné proměnné, neznámé pro řídící systém. Klasické řídící metody často selhávají na tomto problému, protože jsou buď příliš nepřesné, nebo příliš výpočetně náročné. Hluboké posilované učení [1] nabízí způsob jak získat rychlou adaptivní kontrolní strategii, která se blíží strategii optimální. Hluboké posilované učení je ještě užitečnější, pokud je dimenzionalita systému pro klasické metody příliš velká, například u robotické manipulace. Tato práce demonstruje adaptivitu neuronových sítí [2] v řízení systémů s proměnlivými parametry, na vlastní simulaci nelineárního systému. Také porovnává tyto strategie s třemi plně nelineárními MPC kontrolery, které by v tomto úkolu měly dosahovat téměř optimálního chování. Navíc je také vytvořena LSTM síť [3] odhadující tyto nepozorovatelné proměnné, která je poté použita v řídící strategii.

Klíčová slova: adaptivní řízení, neuronové sítě, posilované učení

Překlad názvu: Adaptivní řízení pomocí neuronových sítí

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

Neural networks [4] and *reinforcement learning* [1] have proven very capable in solving problematic tasks in control engineering, such as robotic manipulation [5] [6], as well as non-linear control [7] and system identification [8]. Classical control theory approaches often struggle in these tasks, because methods based on a linearized model are too inaccurate and those based on non-linear models are too computationally demanding for *real-time applications*. Neural networks are accurate even on non-linear systems and are computationally efficient, even applicable to embedded systems [4].

Deep reinforcement learning [1] shows great promise in *robust* and *adaptive control* of *parameter varying systems*. Parameter varying systems are dynamic systems, whose properties depend on unobservable variables, unknown to the controller. There are 2 existing approaches to control these systems. The first one is a robust control approach, which aims to create a controller that works ideally on all, but usually on most variances of the system, without any information about the unobservable variables. The second is an adaptive control approach, where the unobservable variables are treated as latent variables and thus their values can be estimated. Directly solving these tasks usually requires a solution of a large numerical optimization problem, which is computationally infeasible for online control. Classical approximative methods, like extended Kalman filter, often fail, because their initial guess is too far off and the estimator cannot correct itself. This is where *deep reinforcement learning* is very useful since it can learn a very good approximation of the optimal

solution, which is computed much faster than the direct solution.

1.2 Goals

The goal of this thesis is to compare a neural network control approach with a model predictive control approach for controlling a parameter varying dynamic system and to create a neural network to estimate this varying parameter and use this information in the control policy. Neural networks should prove more adaptable to incomplete information setups and also be much faster. Neural networks will also be compared among themselves, to see how additional information improves their performance. All this is demonstrated in a simple non-linear environment, where the MPC is computationally feasible and should be very close to the optimal policy.

1.3 Completion plan

This section outlines the plan to complete the thesis specification and goal. In addition, the expected results are presented.

1.3.1 Steps

Environment. The first step is to create an environment on which all methods and policies could be tested and evaluated. For this thesis, the cart-pole swing-up task has been chosen, as it is non-linear, its dimensionality is feasible for the creation of a good mathematical model, and whose dynamics change significantly depending on the current state and system parameters.

Benchmark controller. The next step is to create a controller based on classic control engineering methods. The fully non-linear version of model predictive control (MPC) has been chosen, as it performs well on non-linear systems and the required mathematical model of the system is available. The MPC controller optionally takes information about the parameters of the model.

Neural networks. Multiple neural network architectures are created and then trained using reinforcement learning algorithms. All networks except for one are feed-forward, they differ in their inputs. The first input option is simply the current observation and trained only on fixed parameters. The second option is the current observation with information about the parameters (ground truth or estimation). The third option is a recent history of observations and actions. Last option is a recurrent network with input composed of the current observation, the last action and the reward, inspired by [9].

Estimator. A recurrent neural network is created to estimate the parameters of the system. This estimator is then used as information for controllers, which utilize parameter information. Controllers with information from the estimator and controllers with ground truth information are then compared.

1.3.2 Expected results

It is expected that the performance of NN would be slightly worse the MPC controller with ground truth information about the parameters, however, the NN would outperform the MPC in the limit cases of the parameter variance with only partial information (estimation or no information at all). It was also expected that the NN with estimator would outperform the other NNs (excluding the one with ground truth parameter information), as it was given additional information in the training phase.

1.4 Thesis structure

This thesis is divided into 9 chapters. In chapter 1 there is a motivation, goal, completion plan and thesis structure and in section 1.5 basic theory is described, formalizing the solved problems. Chapter 2 is a literature review, describing related work. Chapter 3 is the introduction to neural networks (section 3.1) and the introduction to reinforcement learning and used deep reinforcement learning methods (section 3.2). Chapter 5 describes the environment and thus describes the demonstrative problem on which all methods will be tested. Chapter 6 describes the details of method implementations. Chapter 7 presents the results of the thesis and in chapter 8 are these results discussed. Finally, chapter 9 outlines the completion of the thesis specification and subjectively evaluates the thesis.

1.5 Theory

1.5.1 State-space model

State-space model is used to describe dynamic systems, whose dynamics can be represented by a function of its states and actions (inputs).

Definition 1.1. State-space model in discrete time t is defined by $(\mathcal{S}, \mathcal{A}, \mathcal{O}, f, g)$, $\mathcal{S}$ is a set of states, $s \in \mathcal{S}$, $\mathcal{A}$ is a set of actions (inputs), $a \in \mathcal{A}$, $\mathcal{O}$ is a set of observations (outputs), $o \in \mathcal{O}$, $f : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is a dynamics function and $g : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{O}$ is an observation function [10]. Equations describing this model are then

$$s_{t+1} = f(s_t, a_t) \tag{1.1}$$

$$o_t = g(s_t, a_t) \tag{1.2}$$

Optionally, this model can be extended by the introduction of disturbance $d \in \mathcal{D}$ and noise $n \in \mathcal{N}$, which are uncontrollable and often random variables. The model equations are then

$$s_{t+1} = f(s_t, a_t, d_t) \tag{1.3}$$

$$o_t = g(s_t, a_t, n_t) \tag{1.4}$$

This representation allows us to model a large variety of dynamic system or at least approximate them, as continuous-time systems can be discretized.

1.5.2 Markov decision process

State-space model, with condition $o_t = s_t$ can be formalized into a Markov decision process, allowing to evaluate the performance of the selected actions.

Definition 1.2. [1, p. 46] Discrete-time Markov decision process (MDP) is defined by $(\mathcal{S}, \mathcal{A}, T, R, h, \gamma)$, where $\mathcal{S}$ is set of states $s \in \mathcal{S}$, $\mathcal{A}$ is a set of actions $a \in \mathcal{A}$, T is a transition model $T(s_t, a_t, s_{t+1}) : \mathbb{P}[s_{t+1} \mid s_t, a_t]$, $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a reward function where, reward $r_t = R(s_t, a_{t-1})$ is gained by a transition from s_{t-1} to s_t, by taking action a_{t-1}, $h \in \mathbb{N} \cup \{\infty\}$ is a planning horizon and $\gamma \in [0, 1]$ is a discount factor (if $h = \infty$ then $\gamma \neq 1$). The transition model T must have a Markov property, that the transition probability is only based on the current state and action, equation 1.5.

$$\mathbb{P}[s_{t+1} \mid s_t, a_t] = \mathbb{P}[s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_1, a_0, s_0] \tag{1.5}$$

Remark. For a finite set of $\mathcal{S}$ and $\mathcal{A}$ the transition model T is directly a function $T : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \mapsto [0, 1]$, giving the exact probability distribution of the transition, however, for infinite sets the definition becomes much more complicated, unnecessarily for the purpose of this thesis.

Remark. The definition of MDPs can differ slightly. The main differences are in the definition of the reward function R, as to what arguments are used. In some definitions, the only the state s_t is an argument of the reward function so $r_t = R(s_t)$. In other definitions also the initial state s_{t-1} is an argument, besides the final state s_t and action a_{t-1}, so $r_t = R(s_t, a_{t-1}, s_{t-1})$.

Policy. With given MDP, the goal is to find a policy π which maximizes its value function V^π for every state $s \in \mathcal{S}$. Because of Markov property of T, the only information the policy needs for selection of action a_t is the current state s_t.

$$V^\pi(s_t) = \mathbb{E}[\sum_{k=0}^{h-1} \gamma^k r_{t+k+1} \,|\, a_{t+k} = \pi(s_{t+k})] \tag{1.6}$$

1.5.3 Parameter identification and parameter drift

In most practical uses, it is impossible to get the exact dynamics function f in def. 1.1 or transition model T in def. 1.2, so it is necessary to take this information into account and do not blindly rely on our approximation of the real model. For example, if we have a drone, for which we can measure its lift force relatively accurately and we want to design a controller that functions both for when the drone has a camera attached to it and when it has not. If we can identify how much weight the drone has to lift, the controller will be better. This is a problem of parameter identification. If the drone could drop the camera, then the weight would change during the flight and it would be a problem of parameter drift. Another example of parameter drift is the damping coefficient in dampeners. Because of material fatigue, the dampener becomes more and more plastic over time and its dynamics (damping coefficient ζ) can change drastically, as shown in figure 1.1. Another example of a parameter drift problem is the docking of a space shuttle into a refuelling station, as described in [11]. The problem is that during the refuelling procedure, the mass of the shuttle, besides other parameters, changes and the controller might not be able to compensate.

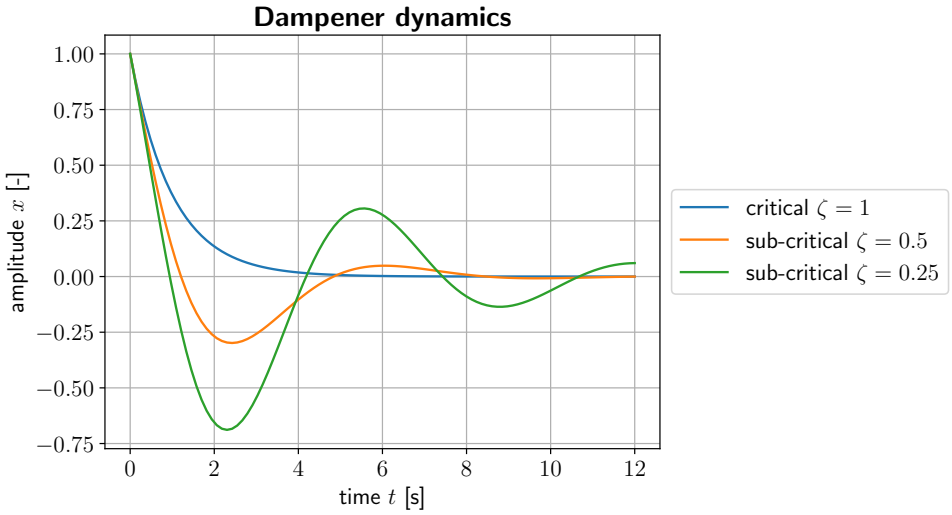


Figure 1.1: Dampener dynamics

1.5.4 Partially Observed Markov Decision Process

To deal with the problem of parameter drift or identification, the partially observed Markov decision process (POMDP) is defined, which is a generalization of MDP (def. 1.2), where the state s is not fully observable.

Definition 1.3. [12] Discrete-time partially observed Markov decision process is defined by $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, \Omega, R, h, \gamma)$, where $\mathcal{S}$ is set of states $s \in \mathcal{S}$, $\mathcal{A}$ is a set of actions $a \in \mathcal{A}$, $\mathcal{O}$ is a set of observations $o \in \mathcal{O}$, T is a transition model with probability distribution $\mathbb{P}[s_{t+1} \,|\, s_t, a_t]$, Ω is an observation model with probability distribution $\mathbb{P}[o_t \,|\, s_t, a_{t-1}]$, $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a reward function where, reward $r_t = R(s_t, a_{t-1})$ is gained by a transition from s_{t-1} to s_t by taking action a_{t-1}, $h \in \mathbb{N} \cup \{\infty\}$ is a planning horizon and $\gamma \in [0, 1]$ is a discount factor (if $h = \infty$ then $\gamma \neq 1$). The transition model T must, as in MDP, have the Markov property (def. 1.2).

Policy. Similarly, as in MPD, the goal is to find a policy π which maximizes its value function V^π. However, unlike in the case of MDP, the policy π in POMDP requires a full history of observations $\{o_i\}_{i=0}^{t}$ and actions $\{a_i\}_{i=0}^{t-1}$ to select an optimal action a_t.

$$V^\pi(\{o_i\}_{i=0}^{t}, \{a_i\}_{i=0}^{t-1}) = \mathbb{E}[\sum_{k=0}^{h-1} \gamma^k r_{t+k+1} \,|\, a_{t+k} = \pi(\{o_i\}_{i=0}^{t+k}, \{a_i\}_{i=0}^{t+k-1})] \quad (1.7)$$

POMDPs are a formalization of the parameter identification and parameter drift problems, the parameters of the model are the unobservable hidden states of the POMDP. To put this definition into an example of the drone, state s is

the drone's velocity and weight, the observation o is only its velocity, the action a is the rotors control inputs and our goal is to put the drone to a standstill. If the policy only uses the current observation of velocity, the control input might be too high for a lighter drone or too low for a heavier drone. However, if we use the history of the observations and actions, the policy can estimate the hidden state, the weight, and adjust the control accordingly.

Chapter 2

Literature review

2.1 Development of adaptive control

One of the first articles that are trying to optimally control stochastic system using algorithms and not analytical solutions is *Dynamic programming and stochastic control processes* by Richard Bellman [13]. The use of a neural network to control or identify dynamic systems has been proposed in *Identification and control of dynamical systems using neural networks* [14] and *Stable adaptive neural control scheme for nonlinear systems* [15]. a multiple model approach was proposed in *Adaptive control using multiple models* [16]. a robust controller for a multiple-input multiple-output system based on a neural network approximator was proposed in *Robust Adaptive Control of Feedback Linearizable MIMO Nonlinear Systems With Prescribed Performance* [17].

2.2 Online adaptive control and identification

Similarly to this thesis, online adaptive control and identification of dynamic systems using neural networks is discussed in *Preparing for the Unknown: Learning a Universal Policy with Online System Identification* [18], where information from an estimator of the hidden parameters is used in the control policy. Recurrent neural network architecture for this task is proposed

in *Adaptive Guidance and Integrated Navigation with Reinforcement Meta-Learning* [19].

2.3 Comparison of neural networks with classic control methods

One of the main goals of this thesis is the comparison of deep reinforcement learning with classic controllers. Similar subjects are discussed in *Adaptive control for non-linear systems using artificial neural network and its application applied on inverted pendulum* [20] and *Artificial Neural Networks, Adaptive and Classical Control for FTC of Linear Parameters Varying Systems* [21].

Chapter 3

Neural networks and reinforcement learning

3.1 Neural networks

The use of neural networks (artificial neural networks) and deep learning has been a prominent approach in the field of artificial intelligence and with the increase in computation power and availability of data, it has found success in a large variety of problems. Most of the information in this chapter comes from [2]. The main reason neural networks are used is that they are powerful function approximators, certain neural network architectures even have the property of universal function approximator [22], further discussed in section 3.1.1. Recurrent neural network architectures, described in section 3.1.3, besides their use on sequences, are also Turing complete [23].

3.1.1 Feed-forward network

As described in [2, ch. 6], feed-forward neural network is a function $\mathbf{y} = f(\mathbf{x}) = f(\mathbf{x}|\mathbf{w})$ with input $\mathbf{x} \in \mathbb{R}^n$, output $\mathbf{y} \in \mathbb{R}^m$ and learn-able parameters $\mathbf{w} \in \mathbb{R}^k$. The phrases *deep* learning and neural *network* are used, because in most cases, the function f is composed of many functions (layers) $f(\mathbf{x}) = f_n(f_{n-1}(\dots f_2(f_1(\mathbf{x}))\dots))$.

The most common layer is and affine transformation with activation function $f_i(\mathbf{x}) = f_i(\mathbf{x} \mid \mathbf{W}, \mathbf{b}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$, where weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$ are learn-able and ϕ is an element-wise activation function. The activation is necessary, because otherwise, multi-layer network would only be a composition of affine transformations, which is also an affine transformation. The most commonly used activation functions are hyperbolic tangent $\phi(x) = \tanh(x)$, sigmoid function $\phi(x) = 1/(1 + \mathrm{e}^{-x})$ and rectified linear unit (ReLU) $\phi(x) = \max(0, x)$ [24], graphs in figure 3.1.

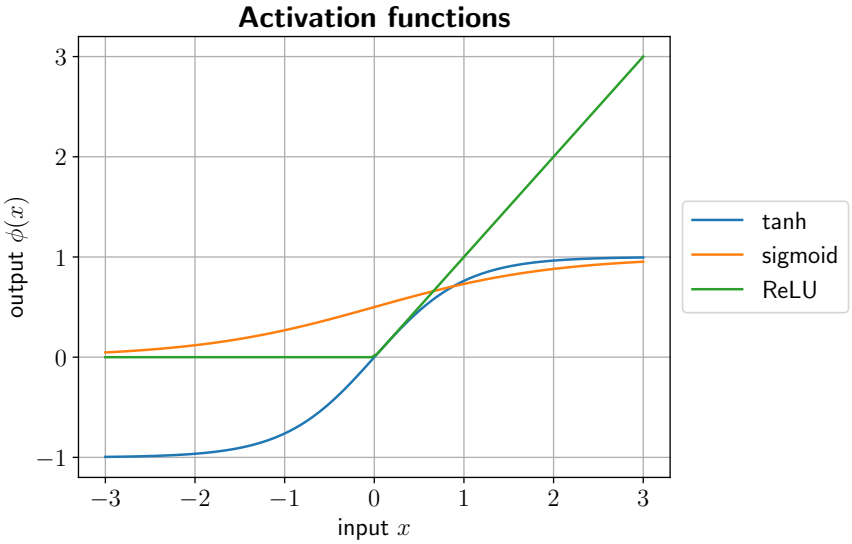


Figure 3.1: Activation functions

Proof. Composition of affine transformations $f_1 : \mathbb{R}^n \mapsto \mathbb{R}^k$ and $f_2 : \mathbb{R}^k \mapsto \mathbb{R}^m$, is an affine transformation $f : \mathbb{R}^n \mapsto \mathbb{R}^m$. Since $f_1(\mathbf{x}) = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$ and $f_2(\mathbf{y}) = \mathbf{W}_2\mathbf{y} + \mathbf{b}_2$, then composition $f_2(f_1(\mathbf{x})) = \mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W} = \mathbf{W}_2\mathbf{W}_1$ and $\mathbf{b} = \mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2$. For a composition of more then two transformations, induction can be used to prove this property. □

Universal function approximator. Such network can be used as a *universal function approximator* [22], which means that the networks parameters $\mathbf{w}$ can be learned so that a sufficiently large network $f : \mathcal{X} \mapsto \mathcal{Y}$ can approximate any function $f^* : \mathcal{X} \mapsto \mathcal{Y}$, where $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{Y} \subseteq \mathbb{R}^m$.

3.1.2 Neural network training

The parameters $\mathbf{w}$ of the network $f(\mathbf{x}|\mathbf{w})$ can be learned using the back-propagation algorithm, where the gradient, with respect to $\mathbf{w}$, of the loss function L is calculated and an optimization algorithm is used to minimize the loss function. The loss function is designed specifically for the task (problem) which the neural network is solving.

Back-propagation. Back-propagation is a way of propagating the gradient from the loss function to the weights of the network. to visualize the computation a computation graph is introduced [25]. For example, the computation graph for 3 layer neural network $f(\mathbf{x}|\mathbf{w}) = f_3(f_2(f_1(\mathbf{x}|\mathbf{w}_1)|\mathbf{w}_2)|\mathbf{w}_3)$ is in figure 3.2. First, the forward pass is calculated, then the backward pass, propagating

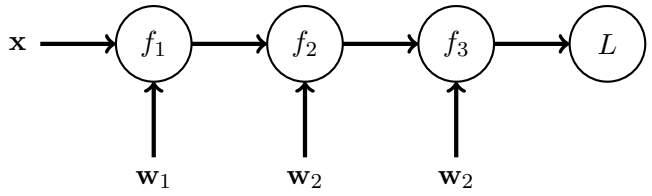


Figure 3.2: Computation graph for 3 layer network

the gradient, is calculated using the chain rule.

$$\frac{\partial L}{\partial \mathbf{w}_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}\frac{\partial f_1}{\partial \mathbf{w}_1} \tag{3.1}$$

$$\frac{\partial L}{\partial \mathbf{w}_2} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial \mathbf{w}_2} \tag{3.2}$$

$$\frac{\partial L}{\partial \mathbf{w}_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial \mathbf{w}_3} \tag{3.3}$$

Optimization algorithms. With the ability to calculate the gradient $\nabla_{\mathbf{w}} L = (\frac{\partial L}{\partial \mathbf{w}})^T$ a gradient descend algorithms are used to minimize the loss. The most simple one is classic gradient descend $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_t)$, with step size $\alpha > 0$. However, to speed up the learning process more sophisticated algorithms such as SGD or Adam optimizer are used.

Stochastic gradient descent. SGD or stochastic gradient descent [26] uses the idea of momentum $\mathbf{m}$, starting at $\mathbf{m}_0 = \mathbf{0}$, which speeds up the convergence

to the minimum. It has 2 hyper-parameters, step size (learning rate) $\alpha > 0$, and momentum coefficient $\mu \in [0, 1]$.

$$\begin{aligned} \mathbf{m}_t &\leftarrow \mu \mathbf{m}_{t-1} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_{t-1}) \\ \mathbf{w}_t &\leftarrow \mathbf{w}_{t-1} + \mathbf{v}_t \end{aligned} \tag{3.4}$$

The momentum prevents the "oscillation" of the weights in each update, meaning the weight go back and forth with very small changes.

Adam optimizer. Adam optimizer [27] uses momentum, like SGD, and adaptability, which means normalizing the update element-wise for each dimension with is estimation from history. It has 4 hyper-parameters, step size (learning rate) $\alpha > 0$, and momentum decay rates β_1, $\beta_2 \in [0, 1)$, both starting at $\mathbf{0}$, and numeric stability constant ϵ, positive but close to 0. It calculates first and second-order moment estimates $\mathbf{m}$ and $\mathbf{v}$, calculates their unbiased versions $\hat{\mathbf{m}}$ and $\hat{\mathbf{v}}$, and uses the unbiased first-order momentum $\hat{\mathbf{m}}$, element-wise normalized by the square root of the unbiased second-order momentum $\hat{\mathbf{v}}$, as an update. All operations on vectors are element-wise.

$$\begin{aligned} \mathbf{g}_t &\leftarrow \nabla_{\mathbf{w}} L(\mathbf{w}_{t-1}) \\ \mathbf{m}_t &\leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t \\ \mathbf{v}_t &\leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t^2 \\ \hat{\mathbf{m}}_t &\leftarrow \mathbf{m}_t / (1 - \beta_1^t) \\ \hat{\mathbf{v}}_t &\leftarrow \mathbf{m}_t / (1 - \beta_2^t) \\ \mathbf{w}_t &\leftarrow \mathbf{w}_{t-1} - \alpha\, \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon) \end{aligned} \tag{3.5}$$

The adaptability improves the speed of the convergence, because when the loss function has a small gradient, it speeds up, and when loss function has a big gradient it slows down, preventing overshoot.

Loss function. As said, the loss function is designed specifically for each task. For simple regression $f^*(\mathbf{x}) \simeq \mathbf{y}$, dataset $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, the mean square error is used $L = \frac{1}{n} \sum_{i=1}^n ||f(\mathbf{x}) - \mathbf{y}||^2$. For classification, dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i$ is feature vector and y_i is label of the class, the network returns $\mathbf{p} \in [0, 1]^m$ (m is the number of classes) the probability of to which class the feature vector $\mathbf{x}$ belongs, the cross entropy loss is used $L = -\frac{1}{n} \sum_{i=1}^n \log(p_{y_i})$ [24].

3.1.3 Recurrent network

Recurrent network (RNN) is no longer only a function, but it has some memory (state) which is saved in each pass of the network [2, ch. 10]. This

is useful when the input and/or output is a sequence with varying length. It can be described as a state-space model, with the initial state $\mathbf{s}_0$. For example if we extend 3 layer network such that layers f_1 and f_2 are recurrent, the computation graph of this network is in figure 3.3.

In training RNNs the issue of exploding or vanishing gradient occurs [28].

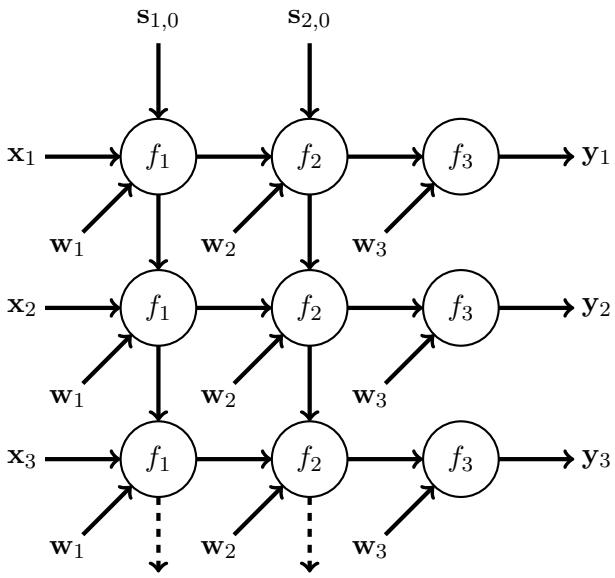


Figure 3.3: Computation graph for 3 layer network with recurrent architecture

Since the calculation of the state is based also on itself and the weights, if the weights are small the gradient vanishes and the update of the weights is almost non-existent, but if the weights are too big, the gradient explodes and the update is too large. To solve this issue a long short-term memory (LSTM) architecture is introduced.

Long short-term memory. LSTM architecture [3] [29], uses memory cells and gate units to keep information over long periods of time. LSTM can be described using equations 3.6, where σ is the sigmoid function, $\circ$ is element-wise product, $\oplus$ is vector concatenation, $\mathbf{a} \oplus \mathbf{b} = [\mathbf{a}^T\ \mathbf{b}^T]^T$, $\mathbf{x}_t \in \mathbb{R}^n$ is input, $\mathbf{f}_t \in \mathbb{R}^m$ is forget gate vector, $\mathbf{i}_t \in \mathbb{R}^m$ is input gate vector, $\mathbf{o}_t \in \mathbb{R}^m$ output gate vector, $\bar{\mathbf{c}}_t \in \mathbb{R}^m$ is long-term input, $\mathbf{c}_t \in \mathbb{R}^m$ is the long-term state and $\mathbf{h}_t \in \mathbb{R}^m$ is the short-term state and also the output vector. Weight matrices $\mathbf{W}_f$, $\mathbf{W}_i$, $\mathbf{W}_o$, $\mathbf{W}_c \in \mathbb{R}^{m \times n+m}$ and bias vectors $\mathbf{b}_f$, $\mathbf{b}_i$, $\mathbf{b}_o$, $\mathbf{b}_c \in \mathbb{R}^m$ are learnable. Diagram of the LSTM cell in figure 3.4.

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f(\mathbf{x}_t \oplus \mathbf{h}_{t-1}) + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i(\mathbf{x}_t \oplus \mathbf{h}_{t-1}) + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o(\mathbf{x}_t \oplus \mathbf{h}_{t-1}) + \mathbf{b}_o) \\
\bar{\mathbf{c}}_t &= \tanh(\mathbf{W}_c(\mathbf{x}_t \oplus \mathbf{h}_{t-1}) + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \bar{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{c}_t)
\end{aligned}
\tag{3.6}
$$

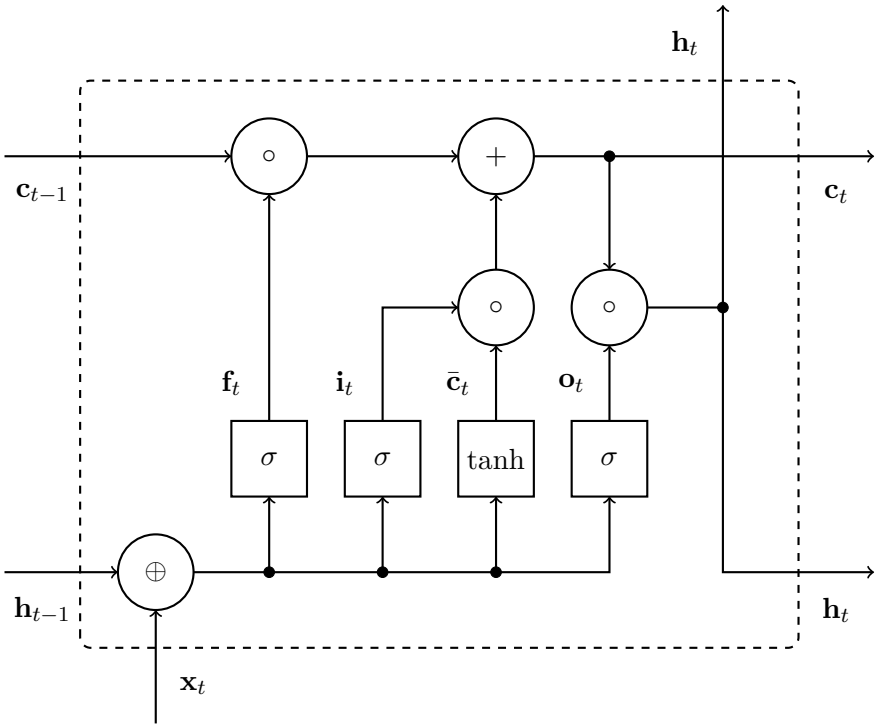


Figure 3.4: LSTM cell diagram

3.2 Reinforcement learning

Reinforcement learning is a framework of methods used approximately solve MDPs (def. 1.2 and POMDPs 1.3, as their exact solutions are computationally infeasible. Most information regarding this section was acquired from [1].

3.2.1 Basics

The basics of reinforcement learning systems [1, p. 6] are *environment*, *reward*, *policy* (agent), *episode* and *value function*.

Environment. The concept of environment is defining the whole problem, which the reinforcement algorithm is trying to solve. It can vary from a chess or computer game to a stock market or a flying plane. The environment follow its rules, described by its *states* in its possible state space $s \in \mathcal{S}$, *actions* in possible action space $a \in \mathcal{S}$ and optionally *observations* in observation space $o \in \mathcal{O}$. For example, for a chess game the state is the current position of pieces and action is moving a piece, for a stock market, the current state are current prices, amounts owned by all brokers and the capital available and the action is buying or selling a certain amount of stocks and for a flying plane the state is the position, velocity, weight, amount of fuel, etc. and the action is the control inputs to the motors, ailerons, rudder and elevators.

Reward. Reward $r \in \mathbb{R}$ is a numerical value which measures how good the taken action is and the current state is, or possibly how good a transition from a previous state to the current one is, usually described by a reward function $r_t = R(s_t, a_{t-1})$ or $r_t = R(s_t, a_{t-1}, s_{t-1})$. It is sent by the environment and its design is part of the problem definition, as different rewards designs result in different optimal behaviours.

Policy. Policy π (sometimes called agent) defines the behaviour, the actions a taken in the environment. It may be a function based on current observation, history of observations and actions, a fixed sequence of actions, a model with memory or completely random variable. Often, and specifically in the deep reinforcement learning section of this thesis (3.2.4), *stochastic* policy $\pi(a \mid s) = \pi(s, a)$ is used, which is a function returning probability of action a for discrete $\mathcal{A}$ or probability density of action a for continuous $\mathcal{A}$ while in state s. Practically, for continuous action space $\mathcal{A}$, the policy returns μ and possibly σ (sometimes σ is fixed) and the actions in training are sampled from normal distribution with mean μ and variance σ^2, $a \sim \mathcal{N}(\mu, \sigma^2)$ in training and $a = \mu$ in testing.

Episode. Episode is a sequence of states, actions and rewards, it is one run of the environment, for example, one game, one flight. The environment should

have a defined starting state s_0 distribution and usually a terminal state or states s_T (T is the terminal time) after which the episode ends.

$$E = \{S_0,\ A_0, R_1,\ S_1,\ A_1, R_2\ \ldots,\ S_{T-1},\ A_{T-1}, R_T\ S_T\} \tag{3.7}$$

All variables S_t, A_t, R_t are random and are generated by the environment and policy (defined by the MDP or POMDP formalization of the environment). Also expected discounted gain G_t for each time step can be defined, where discount factor $\gamma \in [0, 1]$ is defined, which is used because immediate reward should be preferred over the future reward. Naturally expected gain in the terminal state is zero, $G_T = 0$.

$$\begin{aligned} G_t &= \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \\ g_t &= \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \end{aligned} \tag{3.8}$$

Value function. Value function $V^\pi(s)$ [1, p. 58] measures how good a policy π is long term. Its definition is the same as in MDP (def. 1.2) or POMDP (def. 1.3), $V^\pi(s_t)$ is the expected discounted gain when taking actions according to the policy π given current state s_t.

$$V^\pi(s_t) = \mathbb{E}[G_t \mid S_t = s_t] \tag{3.9}$$

Action-value function. Action-value function $Q^\pi(s, a)$ [1, p. 58] is similar to a value function $V^\pi(s)$, but the next action a is also taken as an argument.

$$Q^\pi(s_t, a_t) = \mathbb{E}[G_t \mid S_t = s_t, A_t = a_t] \tag{3.10}$$

Advantage function. Advantage function $D^\pi(s, a)$ measures how good a next action a is given s.

$$D^\pi(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{3.11}$$

The advantage function makes sense only for stochastic policies, since for deterministic policies $\forall s\, D^\pi(s, a) = 0$, because action a is directly given by state s and $Q(s, a) = V(s)$.

Remark. Usually, an advantage function D^π is denoted by A^π or simply A, but this notation is in conflict with the notation of random variable of action A_t.

3.2.2 Optimal policies and value functions

Optimal policy π^* [1, p. 62] is a policy which maximizes its value function $V^{\pi^*} = V^*$ for all states $s \in \mathcal{S}$ and its action-value function $Q^{\pi^*} = Q^*$ for all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$. Multiple policies might be optimal, but they all share an optimal value function V^* and optimal action-value function Q^*.

$$\begin{aligned} V^*(s) &= \max_\pi V^\pi(s) \\ Q^*(s,a) &= \max_\pi Q^\pi(s,a) \end{aligned} \tag{3.12}$$

There are certain interesting equations regarding these functions.

$$Q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1}) \mid S_t = s, A_t = a] \tag{3.13}$$

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} Q^*(s,a) = \\ &= \max_{a \in \mathcal{A}} \mathbb{E}[G_t \mid S_t = s, A_t = a] = \\ &= \max_{a \in \mathcal{A}} \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] = \\ &= \max_{a \in \mathcal{A}} \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1}) \mid S_t = s, A_t = a] \end{aligned} \tag{3.14}$$

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}[G_t \mid S_t = s, A_t = a] = \\ &= \mathbb{E}[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q^*(S_{t+1}, a') \mid S_t = s, A_t = a] \end{aligned} \tag{3.15}$$

These equations (3.14 and 3.15) are called *Bellman optimality equations* [1, p. 63]. If the probability distributions of the transitions were known, it would be possible to find the policy π^* directly using these equations, unfortunately, in most cases, the distribution is not known, or the direct computation of the policy is infeasible.

3.2.3 Methods for finite state and action space

There exists a number of methods for solving problems with finite state space $\mathcal{S}$ and finite action space $\mathcal{A}$. They are based on the fact that policy π, value function V^π and action-value function Q^π can be represented as a look-up table and iteratively improve the values until they converge to π^*, V^* and Q^*. Naive methods based on value function estimation, policy and value iteration, are described in chapter 4 of [1], Monte-Carlo search based methods in chapter 5 of [1] and temporal difference methods, such as Q-learning, in chapter 6 of [1]. All of these methods, even in their improved versions, fail when the state space $\mathcal{S}$ and/or action space $\mathcal{A}$ is not only infinite but just simply too big to search fully.

3.2.4 Deep reinforcement learning

For infinite or even continuous state spaces $\mathcal{S}$ and action spaces $\mathcal{A}$ approximate solution methods are necessary [1, p. 195]. They transform the problem of finding π^*, V^* and Q^* to finding the parameters $\mathbf{w}$ of a function which approximates them, so $\pi^*(s) \approx \pi(s \mid \mathbf{w}) = \pi_{\mathbf{w}}(s)$, $V^*(s) \approx V(s \mid \mathbf{w}) = V_{\mathbf{w}}(s)$ and $Q^*(s,a) \approx Q(s,a \mid \mathbf{w}) = Q_{\mathbf{w}}(s,a)$. This is where NNs come into the picture, a parametrized function used as an approximator is their key use. Since the goal of this thesis is to control environments with continuous state and action spaces, policy gradient methods are used [1, p. 321].

Remark. By convention, in reinforcement learning, the goal is to maximize the objective $J = J(\mathbf{w})$, to use the gradient descend methods the loss $L = -J$. Also, all gradients are calculated with respect to parameters $\mathbf{w}$ so $\nabla = \nabla_{\mathbf{w}}$.

REINFORCE. As introduced in [30], the REINFORCE algorithm updates the using objective

$$J(\mathbf{w}) = \hat{\mathbb{E}}_t[G_t \ln(\pi_{\mathbf{w}}(A_t \mid S_t))] \tag{3.16}$$

where $\hat{\mathbb{E}}_t$ denotes the empirical average over a finite batch of samples [31]. In practice, the policy is updated after each episode, or possibly multiple episodes N, the objective gradient is calculated using equation 3.17, where index n denotes to which episode the terminal time T_n, gain $g_{t,n}$, state $s_{t,n}$ and action $a_{t,n}$ belongs.

$$\nabla J(\mathbf{w}) = \frac{1}{n} \sum_{n=1}^{N} \sum_{t=0}^{T_n - 1} g_{t,n} \nabla \ln(\pi_{\mathbf{w}}(a_{t,n} | s_{t,n})) \tag{3.17}$$

Advantage function estimation. Since the values of gain G_t might vary greatly, and is usually biased, its is better to calculate an advantage estimation $\hat{D}_t$, with realization $\hat{d}_t$, and use it in the calculation of the policy gradient. to calculate the estimation, a value function estimator $V_{\mathbf{w}}(s)$ is necessary. This is called the actor-critic method, where the policy $\pi_{\mathbf{w}}$ is the actor and value function $V_{\mathbf{w}}$ is the critic, and both are learned simultaneously,. The parameters $\mathbf{w}$ might be partially shared because the information in some of the layers might be useful for both policy and value function. The value function $V_{\mathbf{w}}$ updated by using a square error loss L_V (eq. 3.18) scaled by $\beta > 0$, which is included in the objective. The updated policy objective J_π is in equation 3.20 and the policy gradient ∇J_π calculation is in equation 3.21. The policy objective L_π and the value loss L_V are then combined to create the whole objective J and its gradient calculation ∇J

$$L_V(\mathbf{w}) = \hat{\mathbb{E}}_t[(V_{\mathbf{w}}(S_t) - G_t)^2] \tag{3.18}$$

$$\nabla L_V(\mathbf{w}) = \frac{2}{N}\sum_{n=1}^{N}\sum_{t=0}^{T_n-1}\hat{d}_{t,n}(V_{\mathbf{w}}(s_t) - g_t)\nabla V_{\mathbf{w}}(s_t) \tag{3.19}$$

$$J_\pi(\mathbf{w}) = \hat{\mathbb{E}}_t[\hat{D}_t \ln(\pi_{\mathbf{w}}(A_t \mid S_t))] \tag{3.20}$$

$$\nabla J_\pi(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T_n-1}\hat{d}_{t,n}\nabla \ln(\pi_{\mathbf{w}}(a_{t,n}|s_{t,n})) \tag{3.21}$$

$$J(\mathbf{w}) = J_\pi(\mathbf{w}) - \beta L_V(\mathbf{w}) \tag{3.22}$$

$$\nabla J(\mathbf{w}) = \nabla J_\pi(\mathbf{w}) - \beta\nabla L_V(\mathbf{w}) \tag{3.23}$$

To there are many possible ways to calculate the advantage estimate $\hat{D}_t$ [32]. The first is the gain with baseline $\hat{D}_t^{BL}$.

$$\hat{D}_t^{BL} = G_t - V_{\mathbf{w}}(S_t) \tag{3.24}$$

The second is temporal difference.

$$\hat{D}_t^{TD} = \delta_t = R_{t+1} + \gamma V_{\mathbf{w}}(S_{t+1}) - V_{\mathbf{w}}(S_t) \tag{3.25}$$

The third option is generalized advantage estimator (GAE) $\hat{D}_t^{GAE(\lambda)}$, which is a combination of the first two estimations, parametrized by factor $\lambda \in [0, 1]$ [32].

$$\hat{D}_t^{GAE(\lambda)} = \sum_{k=0}^{T-t-1}(\lambda\gamma)^k\delta_{t+k} \tag{3.26}$$

Interesting fact for GAE is that $\hat{D}_t^{GAE(0)} = \hat{D}_t^{TD} = \delta_t$ and $\hat{D}_t^{GAE(1)} = \hat{D}_t^{BL}$ [32].

Policy entropy. To assure that the policy $\pi_{\mathbf{w}}$ will remain exploring, the entropy term J_H is added to the objective, scaled by $\beta_H > 0$, to promote higher entropy policies, further discussed in [33]. Entropy for state s of stochastic policy π is denoted $\mathrm{H}(\pi(\cdot \mid s))$.

$$J_H(\mathbf{w}) = \hat{\mathbb{E}}_t[\mathrm{H}(\pi_{\mathbf{w}}(\cdot \mid S_t))] \tag{3.27}$$

$$J(\mathbf{w}) = J_\pi(\mathbf{w}) - \beta L_V(\mathbf{w}) + \beta_H J_H(\mathbf{w}) \tag{3.28}$$

Proximal policy optimization. Another problem in actor-critic methods is that using one set of episodes for multiple optimization steps might lead to destructively large policy updates [31]. To be able to reuse the set of episodes for multiple optimization steps in one update a proximal policy optimization (PPO) [31], which is an improvement of trust region policy optimization (TRPO) [34], is introduced. Both PPO and TRPO put a limit on how much a policy $\pi_{\mathbf{w}}$ can change compared to the policy before the update $\pi_{\mathbf{w}_{old}}$, the main advantage of PPO, compared to TRPO, is that the optimization task is unbounded. At the beginning of each update, the parameters $\mathbf{w}$ are saved

so $\mathbf{w}_{old} \leftarrow \mathbf{w}$, and the ratio of change of policy $\rho_t(\mathbf{w})$ (in the original paper [31] it is denoted $r(\theta)$, where θ are parameters).

$$\rho_t(\mathbf{w}) = \frac{\pi_{\mathbf{w}}(A_t \mid S_t)}{\pi_{\mathbf{w}_{old}}(A_t \mid S_t)} \tag{3.29}$$

The policy objective of the PPO, without the value function update, is then in equation

$$J_\pi(\mathbf{w}) = \hat{\mathbb{E}}_t[\min(\hat{D}_t \rho_t(\mathbf{w}), \hat{D}_t \operatorname{clip}(\rho_t(\mathbf{w}), 1-\epsilon, 1+\epsilon))] \tag{3.30}$$

The clipping of the policy change ratio assures that update of the policy will not be too big. For a positive $\hat{D}_t$, it assures that action probability will not rise too much, however, it can be lowered without limits if an even better action is found, on the other hand for a negative $\hat{D}_t$, it assures that the action probability will not drop too much, as it would remove it from exploration completely, but it can be raised without limits if an even worse action is found.

Chapter 4

Classical control

This chapter about the basics in control theory. The first part includes the definition of the state-space model and discretization. The second focus on model predictive control design and moving horizon estimation.

4.1 Basics

4.2 State-space model

The state-space model definition in the introduction has a very general set of states $\mathcal{S}$, actions $\mathcal{A}$ and observations $\mathcal{O}$. In this chapter, all of these sets a vector space, $\mathcal{S} = \mathbb{R}^n$, $\mathcal{A} = \mathbb{R}^m$ and $\mathcal{O} = \mathbb{R}^k$, with possible lower or upper bounds. The introduction definition is only for discrete-time system, however, control theory is applicable also on continuous-time system. It also differs in notation, as states are denoted $\mathbf{x}(t)$, actions $\mathbf{u}(t)$ and outputs (observations) $\mathbf{y}(t)$, where t is time (discrete or continuous). Time derivations are denoted $\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \dot{\mathbf{x}}(t)$.

Definition 4.1. Continuous-time state-space model is defined by dynamics function $f : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ and output function $g : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^k$ [35].

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) &= g(\mathbf{x}(t), \mathbf{u}(t)) \end{aligned} \tag{4.1}$$

Definition 4.2. Discrete-time state-space model is defined by dynamics function $f : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{Z} \mapsto \mathbb{R}^n$ and output function $g : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^k$ [35].

$$\begin{aligned} \mathbf{x}(t+1) &= f(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) &= g(\mathbf{x}(t), \mathbf{u}(t)) \end{aligned} \tag{4.2}$$

Remark. Definitions 4.1 and 4.2 assume that the systems are time invariant. In some definitions of the state-space models, functions f and g are time dependent, but time can be represented as an uncontrollable and unobservable state.

4.2.1 Discretization

Contiguous-time system can be discretized by using the Euler method [36], which is a first-order approximation. It is causal, which means that the next state is only affected by the previous states, it also introduces a minimal delay caused by discretization. The principle is $\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} \approx \frac{\Delta\mathbf{x}(t)}{\Delta t}$, where Δt is the discretization step.

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = f(\mathbf{x}(t), \mathbf{u}(t)) \approx \frac{\Delta\mathbf{x}(t)}{\Delta t} = \frac{\mathbf{x}(t+1) - \mathbf{x}(t)}{\Delta t} = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{4.3}$$

Then the equation is rearranged into a discrete-time state-space model

$$\mathbf{x}(t+1) = \hat{f}(\mathbf{x}(t), \mathbf{u}(t), t) = \mathbf{x}(t) + \Delta f(\mathbf{x}(t), \mathbf{u}(t)) \tag{4.4}$$

There are a lot more discretization methods, described in chapter 8 Digital control in [35], all of them trading off between accuracy (higher-order methods), computational complexity and introduced delay.

4.3 Controller and estimator design

4.3.1 Model predictive control

Model predictive control (MPC) [37] is a feed-back control method used to control discrete or discretized systems. It is based on an optimization algorithm which looks h steps ahead to "predict" where the system will be. In practice it requires a solution of a constrained optimization problem, using the dynamic function f, assumed time-invariant, given initial state $\mathbf{x}_0$, (assuming state is fully observable, $\mathbf{y} = \mathbf{x}$), and a loss function L, the optimization

variables are the sequence of states $\{\mathbf{x}_i\}_{i=1}^{h}$ and actions $\{\mathbf{u}_i\}_{i=0}^{h-1}$. Optionally upper $\mathbf{b}_{u,s}$, $\mathbf{b}_{u,a}$ and/or lower bounds $\mathbf{b}_{l,s}$, $\mathbf{b}_{l,a}$ are added on states and actions, $\mathbf{x}_t = \mathbf{x}(t)$, $\mathbf{u}_t = \mathbf{u}(t)$.

$$\begin{aligned} &\min_{\{\mathbf{x}_i\}_{i=1}^{h},\{\mathbf{u}_i\}_{i=0}^{h-1}} L(\{\mathbf{x}_i\}_{i=0}^{h},\{\mathbf{u}_i\}_{i=0}^{h-1}) \\ &\text{subject to} \\ &\qquad \mathbf{x}_{t+1} = f(\mathbf{x}_t,\mathbf{u}_t), \quad t \in \{1,\ldots,h\} \\ &\qquad \mathbf{b}_{u,s} \geq \mathbf{x}_t \geq \mathbf{b}_{l,s}, \quad t \in \{1,\ldots,h\} \\ &\qquad \mathbf{b}_{u,a} \geq \mathbf{u}_t \geq \mathbf{b}_{l,a}, \quad t \in \{0,\ldots,h-1\} \end{aligned} \tag{4.5}$$

The loss function is usually composed of 3 terms. Lagrange term L_1 for all time steps except last, Meyer term for the final state $\mathbf{x}_h$ and action difference penalization defined by positive semi-definite matrix $\mathbf{R} \in \mathbb{R}^{m\times m}$ [38]. The first action $\mathbf{u}_0$ is then used as the output of the controller.

$$L = \sum_{t=0}^{h-1} L_1(\mathbf{x}_t,\mathbf{u}_t) + L_2(\mathbf{x}_h) + \sum_{t=0}^{h-2} \mathbf{u}_t^T \mathbf{R}\,\mathbf{u}_t \tag{4.6}$$

4.3.2 Moving horizon estimation

Moving horizon estimation (MHE) is a method for estimating the unobserved or noisy states of the system [39] when the dynamics function f and observation function g, dependent only on the state $\mathbf{x}$, are known and time-invariant. Given the history of observations $\{\mathbf{y}_i\}_{i=0}^{h}$ and actions $\{\mathbf{u}_i\}_{i=0}^{h-1}$. The estimations of the states $\{\hat{\mathbf{x}}_i\}_{i=1}^{h}$ are gained by optimizing the following task, equation 4.7, $\mathbf{P} \in \mathbb{R}^{n\times n}$, $\mathbf{Q} \in \mathbb{R}^{n\times n}$ are positive semi-definite matrices.

$$\begin{aligned} \min_{\{\hat{\mathbf{x}}_i\}_{i=0}^{h}} &\sum_{t=0}^{h} \Delta\hat{\mathbf{y}}_t^T \mathbf{P} \Delta\hat{\mathbf{y}}_t + \sum_{t=0}^{h-1} \Delta\hat{\mathbf{x}}_t^T \mathbf{Q} \Delta\hat{\mathbf{x}}_t \\ \Delta\hat{\mathbf{y}}_t &= \mathbf{y}_t - g(\hat{\mathbf{x}}_t) \\ \Delta\hat{\mathbf{x}}_t &= \hat{\mathbf{x}}_{t+1} - f(\hat{\mathbf{x}}_t,\mathbf{u}_t) \end{aligned} \tag{4.7}$$

The performance of the estimator greatly depends on the choice of $\mathbf{P}$ and $\mathbf{Q}$. If the states $\mathbf{x}$ have big disturbances and the observation $\mathbf{y}$ is exact, then $\mathbf{P}$ should be relatively bigger, however, if the states $\mathbf{x}$ suffer from low disturbance and the observation $\mathbf{y}$ is noisy, then $\mathbf{Q}$ should be relatively bigger.

In practice observations $\{\mathbf{y}_i\}_{i=0}^{h}$ and actions $\{\mathbf{u}_i\}_{i=0}^{h-1}$ are only a temporal history of the system with fixed length h, as further history is unnecessary and would only slow down the computation.

Chapter 5

Environment

For the environment, on which all policies (controllers) have been tested and evaluated, the cart-pole swing-up task has been chosen. It is non-linear, low dimensional, its dynamics change greatly depending on the current state and parameters of the system. These proprieties were crucial, as the low dimensionality allowed us to create a good mathematical model and thus a good MPC controller and dynamics variability were necessary to demonstrate the adaptability of the controllers.

5.1 Formulation

The formulation of the task is this, in normal gravitational field $g = 9.81\,\mathrm{m/s^2}$, a moving cart on a rail with weight $m_c = 1\,\mathrm{kg}$ has attached a pendulum with weight $m_p = 0.2\,\mathrm{kg}$ and length $l \in [0.05\,\mathrm{m}, 1.95\,\mathrm{m}]$. The observation of the environment consists of the position $x \in [-5\,\mathrm{m}, 5\,\mathrm{m}]$ and velocity $\dot{x}$ in $\mathrm{m/s}$ of the cart, and the angle θ in rad and angular velocity $\dot{\theta}$ in $\mathrm{rad/s}$ of the pendulum, the upward position is $\theta = 0\,\mathrm{rad}$. The goal is to move the system from the initial position $x \in [-4\,\mathrm{m}, 4\,\mathrm{m}]$, $\dot{x} = 0\,\mathrm{m/s}$, $\theta = \pi\,\mathrm{rad}$, $\dot{\theta} = 0\,\mathrm{rad/s}$ to the final position $x \in 0\,\mathrm{m}$, $\dot{x} = 0\,\mathrm{m/s}$, $\theta = 0\,\mathrm{rad}$, $\dot{\theta} = 0\,\mathrm{rad/s}$ using input control $a \in [-1, 1]$, where force $F_a = 50\,\mathrm{N}\,a$ is applied on the cart. Also dampening force $F_d = -b\,\dot{x}$ (dampening coefficient $b = 0.1\,\mathrm{kg/s}$) is applied on the card and input noise $a_n \sim N(0, 0.02^2)$ is added to input control ($N(\mu, \sigma^2)$ is a normal distribution with mean μ and variance σ^2).

Remark. Further in this thesis, the units are omitted and assumed to be

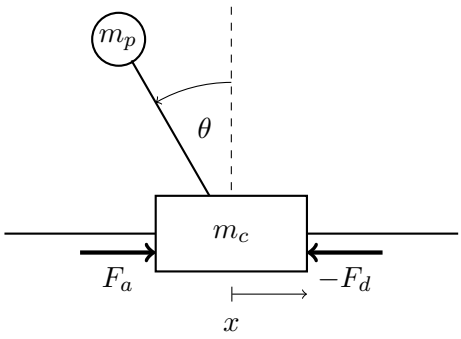


Figure 5.1: Cartpole model

the ones in the formulation of the system.

5.2 Mathematical model

To create the mathematical model, the approach of Lagrange equations has been used [40]. Firstly, the equations for potential and kinetic energies were formulated. Secondly. the Euler-Lagrange equations were derived. Lastly, the state-space model has been created.

Remark. During the whole creation of the mathematical model, even thought the parameter l varies, it was assumed to be fixed in time.

Potential and kinetic energies. To calculate these energies, the expression of the absolute positions x_p, y_p and velocities $\dot{x}_p$, $\dot{y}_p$ of the pendulum were necessary (y is the height in which the pendulum is relative to the rail).

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix} + \begin{bmatrix} -l\sin(\theta) \\ l\cos(\theta) \end{bmatrix} = \begin{bmatrix} x - l\sin(\theta) \\ l\cos(\theta) \end{bmatrix} \tag{5.1}$$

To calculate the velocities, equation 5.1 was simply differentiated by time, x and θ are time dependant.

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} \dot{x} \\ 0 \end{bmatrix} + \begin{bmatrix} -l\cos(\theta)\dot{\theta} \\ -l\sin(\theta)\dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} - l\cos(\theta)\dot{\theta} \\ -l\sin(\theta)\dot{\theta} \end{bmatrix} \tag{5.2}$$

The potential energy of the system V has only one element, the gravitation potential of the pendulum V_p, since the cart's height is fixed by the rail.

$$V = V_p = gm_p y_p = gm_p l\cos(\theta). \tag{5.3}$$

The kinetic energy of the system T has two components, the kinetic energy of the cart T_c, and the kinetic energy of the pendulum T_p.

$$T_c = \frac{1}{2} m_c \dot{x}^2 \tag{5.4}$$

$$\begin{aligned} T_p &= \frac{1}{2} m_p (\dot{x}_p^2 + \dot{y}_p^2) = \frac{1}{2} m_p ((\dot{x} - l\cos(\theta)\dot{\theta})^2 + (-l\sin(\theta)\dot{\theta})^2) = \\ &= \frac{1}{2} m_p (\dot{x}^2 - 2l\dot{x}\cos(\theta)\dot{\theta} + l^2\dot{\theta}^2) \end{aligned} \tag{5.5}$$

$$T = T_c + T_p = \frac{1}{2}(m_c + m_p)\dot{x}^2 + \frac{1}{2} m_p l^2 \dot{\theta}^2 - m_p l \dot{x}\cos(\theta)\dot{\theta} \tag{5.6}$$

Remark 5.1. Since the pendulum was assumed to be a point of mass, the rotational kinetic energy was omitted.

Euler-Lagrange equations. Using the potential (eq. 5.3) and kinetic energies (eq. 5.6), the Lagrangian L is defined.

$$L = T - V = \frac{1}{2}(m_c + m_p)\dot{x}^2 + \frac{1}{2} m_p l^2 \dot{\theta}^2 - m_p l \dot{x}\cos(\theta)\dot{\theta} - g m_p l \cos(\theta) \tag{5.7}$$

The Euler-Lagrange equation for x and θ are then

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = F_a + F_d \tag{5.8}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = 0 \tag{5.9}$$

First, the derivations for x were calculated

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{x}} &= \frac{\mathrm{d}}{\mathrm{d}t}\left((m_c + m_p)\dot{x} - m_p l\cos(\theta)\dot{\theta}\right) = \\ &= (m_c + m_p)\ddot{x} + m_p l\sin(\theta)\dot{\theta}^2 - m_p l\cos(\theta)\ddot{\theta} \end{aligned} \tag{5.10}$$

$$\frac{\partial L}{\partial x} = 0 \tag{5.11}$$

then for θ

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{\theta}} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(m_p l^2\dot{\theta} - m_p l\dot{x}\cos(\theta)\right) = \\ &= m_p l^2\ddot{\theta} - m_p l\ddot{x}\cos(\theta) + m_p l\dot{x}\sin(\theta)\dot{\theta} \end{aligned} \tag{5.12}$$

$$\frac{\partial L}{\partial \theta} = m_p l\dot{x}\sin(\theta)\dot{\theta} + g m_p l\sin(\theta) \tag{5.13}$$

and then combined into the Euler-Lagrange equations

$$(m_c + m_p)\ddot{x} + m_p l\sin(\theta)\dot{\theta}^2 - m_p l\cos(\theta)\ddot{\theta} = -b\dot{x} + F_a \tag{5.14}$$

$$m_p l^2\ddot{\theta} - m_p l\ddot{x}\cos(\theta) - g m_p l\sin(\theta) = 0 \tag{5.15}$$

State-space model. Equations 5.14 and 5.15 are then combined to create a continuous-time state-space model.

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{-m_p l \sin(\theta)\dot{\theta}^2 + g m_p \cos(\theta)\sin(\theta) - b\dot{x} + F_a}{m_c + m_p - m_p \cos(\theta)^2} \\ \dot{\theta} \\ \frac{-m_p l \cos(\theta)\sin(\theta)\dot{\theta}^2 - b\dot{x}\cos(\theta) + g m_p \sin(\theta) + g m_c \sin(\theta) + \cos(\theta) F_a}{l(m_c + m_p - m_p \cos(\theta)^2)} \end{bmatrix} \tag{5.16}$$

5.3 Simulation

For the simulation of this physical system, the library PyBullet was used [41]. It was encapsulated into a Python [42] class, which has the properties of an Open AI Gym [43] environment, which has become a standard in the field of reinforcement learning. Besides constructor, this class has two main methods, *reset*, which resets the environment to the initial condition and returns the initial observation, and *step*, which takes action as an argument, moves the simulation one step forward and returns the new observation, reward and information whether the episode has ended.

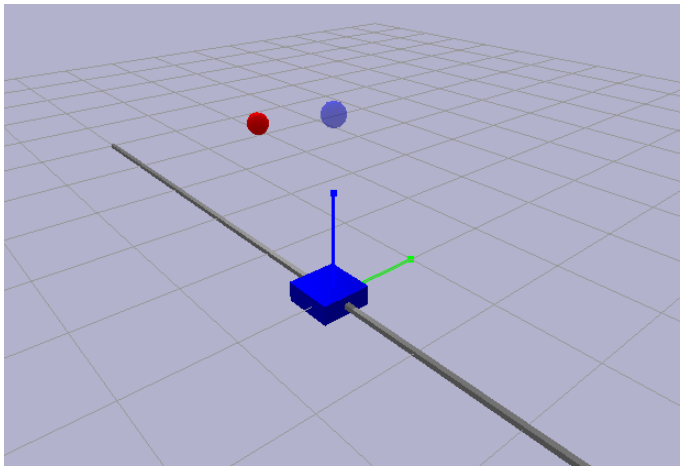


Figure 5.2: Screenshot of the environment simulation

Parameters. The simulation is discretized with the time step $\Delta t = 20\,\mathrm{ms}$, which is sufficient to get a good approximation of the real system. The maximum number of steps in an episode (one run of the simulation) is 300, therefore the maximum time of the simulation is 6 s, when running in real-time. The episode also ends when $|x_p| < 0.05$, $|\dot{x}_p| < 0.05$, $|\theta| < 0.05$ and

$|\dot{\theta}| < 0.05$. This termination condition was chosen to speed up the simulation, as the system, whose state is within these parameters, is sufficiently close to the final position described in the formulation.

5.4 Reward function

The reward function for each step was chosen as following

$$\begin{aligned} r_t =& R(s = s_t, a = a_{t-1}) = \frac{1}{2}(\cos(\theta) - 1) - 0.5\frac{|x_p|}{5} - \\ & -0.01\dot{\theta}^2 - 0.01\dot{x}_p^2 - 0.1a^2 - 10(|x| > 4.95) \end{aligned} \tag{5.17}$$

This reward function was constructed to promote fast moving of the pendulum to the upward position (term $\frac{1}{2}(\cos(\theta) - 1)$) and moving the pendulum above the center of the rail (term $-0.5\frac{|x_p|}{5}$), while conserving energy used on the action (term $-0.1a^2$) and punishing undesired behaviours such as moving too fast (term $-0.01\dot{\theta}^2 - 0.01\dot{x}_p^2$) or abusing the end of the rail (term $-10(|x| > 4.95)$). This reward function is never positive to promote policies which reach the final position as fast as possible.

Chapter 6

Experiments

6.1 MPC controller

Using a Python [42] library *Do-MPC* [38] an MPC controller has been created. This library is suitable for this task, as it allows the creation of fully non-linear system model with changing parameters and custom optimization objective. It also saves the previous solution in each time step, so the optimizations procedure has a "hot start", a good initial guess of the optimal values, which speeds up the process. The MPC had a prediction horizon of $h = 30$, and its objective C is defined in equation 6.1.

$$\begin{aligned} C_{pos}(t) &= \frac{1}{2}(1 - \cos(\theta(t))) + 0.5\left(\frac{x_p(t)}{5}\right)^2 + 0.01\dot{\theta}(t)^2 + 0.01\dot{x}_p(t)^2 \\ C &= h\, C_{pos}(h) + \sum_{t=1}^{h-1} C_{pos}(t) + 0.01\sum_{t=0}^{h-1} a(t)^2 + 0.01\sum_{t=0}^{h-2}(a(t) - a(t+1))^2 \end{aligned} \tag{6.1}$$

This objective is consistent with the environment reward function (section 5.4), as to get optimal behaviour, only replacing $|x_p|$ with x_p^2, to improve the optimization stability. The term $(a(t) - a(t+1))^2$ was added to smooth out the input a, preventing rapid changes. Two constraints were added, first one was $|x| < 4.95$, to prevent the end of the rail penalization and second $|\theta| < 2.5\pi$ to stabilize the numerical optimization algorithm, as it iterated over all $\theta + 2k\pi, k \in \mathbb{Z}$, which sometimes resulted in a fail of convergence.

Remark. Because of a bug in the *Do-MPC* library [38], the objective could not depend on the varying parameter l, so the pendulum position $x_p(t)$ and

velocity $\dot{x}_p(t)$ is calculated under the assumption $l = 1$. To mitigate the error, an additional term $0.01x(h)^2$ was added to the objective C.

Three MPC controllers were evaluated, first one was MPC with fixed length, *mpc-fixed*, which had no information about the length and assumed $l = 1$. The second was MPC with length oracle, *mpc-oracle*, which had real current length information. The third one was MPC with length estimation, *mpc-estim*, with information from the length estimator, section 6.3.

6.2 Neural network policies

6.2.1 Feed-forward network policy

Feed-forward network with 3 different options as inputs has been created and trained using *PPO2* algorithm [31] with the deep reinforcement library *Stable Baselines* [44].

Architecture. The network is a fully connected network with ReLU activation function, diagram is in figure 6.1, the numbers indicate the numbers of neurons in the layer. All model had the first layer of 64 neurons, except temporal history model, which had the first layer of 256 neurons.

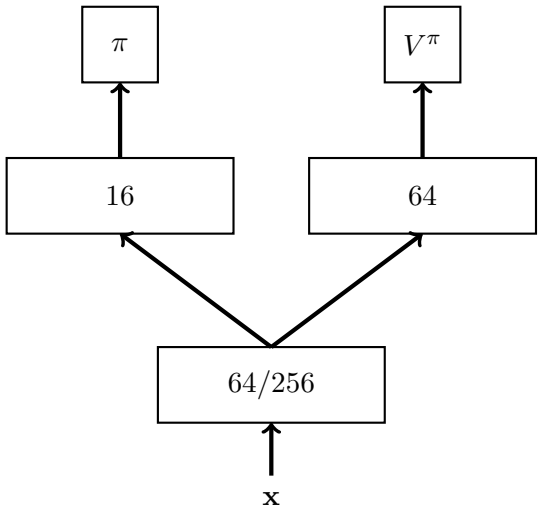


Figure 6.1: Feed-forward network policy diagram

Input transformation. The inputs into the neural network should be within the range $[-1, 1]$, so a bijective transformation of coordinates has been created, in according to this requirement, for the NN input $\mathbf{x} \in \mathbb{R}^5$, equation 6.2. The scaling factors for $\dot{x}$ and $\dot{\theta}$ have been chosen empirically. The tanh transformation of velocities allowed more precise control around $\dot{x} \approx 0$ and $\dot{\theta} \approx 0$.

$$\mathbf{x} = \begin{bmatrix} x/5 \\ \tanh(\dot{x}/2) \\ \sin(\theta) \\ \cos(\theta) \\ \tanh(\dot{\theta}/4) \end{bmatrix} \tag{6.2}$$

Models. Five models were created using this architecture, they differed in input and in training parameters of the environment. The first two models, feed-forward with fixed length, *ff-fixed*, and feed-forward with random length, *ff-random*, had $\mathbf{x}$ (equation 6.2) as an input. They differed in the training procedure, as ff-fixed was trained only on setups where $l = 1$, all other models have been trained on the whole range $l \in [0.05, 1.95]$.

The second two models had additional information about the length, so their input $\hat{\mathbf{x}} \in \mathbb{R}^6$ is $\hat{\mathbf{x}} = [\mathbf{x}^T \ \hat{l}]^T$, where $\hat{l}$ is the real current length, normalized to range $[-1, 1]$, for the benchmark model feed-forward with length oracle, *ff-oracle*, and $\hat{l}$ is the output of the length estimator (section 6.3) for the model feed-forward with length estimation, *ff-estim*.

The last model is feed-forward with temporal history, *ff-hist*, which is an adaptive model whose input is the recent episode history of states and actions. It is the only adaptive model which had no information about the length during training or testing. The input of this model is a matrix $\bar{\mathbf{x}} \in \mathbb{R}^{7\times 128}$, as the history length is 128, and the inputs are $\mathbf{x}$, the last action and history tag, which denotes if the state is part of the episode history.

$$\bar{\mathbf{x}}_t = \begin{bmatrix} \mathbf{0} & \dots & \mathbf{0} & \mathbf{x}_0 & \mathbf{x}_1 & \dots & \mathbf{x}_{t-1} & \mathbf{x}_t \\ 0 & \dots & 0 & 0 & a_0 & \dots & a_{t-2} & a_{t-1} \\ 0 & \dots & 0 & 1 & 1 & \dots & 1 & 1 \end{bmatrix}, t < 128 \tag{6.3}$$

$$\bar{\mathbf{x}}_t = \begin{bmatrix} \mathbf{x}_{t-127} & \mathbf{x}_{t-126} & \dots & \mathbf{x}_{t-1} & \mathbf{x}_t \\ a_{t-128} & a_{t-127} & \dots & a_{t-2} & a_{t-1} \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix}, t \geq 128 \tag{6.4}$$

6.2.2 Recurrent neural network policy

One recurrent neural network policy, *rnn*, has been created and trained using *PPO2* [31] in *Stable Baselines* library [44]. Because of the problem of exploding gradient, policy and value function share no weights, as big spikes in value function, destroyed previous prototype policies. Both policy and value function were composed of one LSTM cell with 64-dimensional output. The input to this policy $\tilde{\mathbf{x}}$ is a state transformed for neural networks (eq.6.2) $\mathbf{x}_t$, the last action a_{t-1} and current reward r_t, $\tilde{\mathbf{x}}_t = [\mathbf{x}^T\, a_{t-1}, r_t]^T$, $a_{-1} = 0$, as inspired by [9].

6.3 Length estimator

The length estimator is an LSTM model in the deep learning library *PyTorch* [45]. The input $\tilde{\mathbf{x}}$ is a state transformed for neural networks (eq.6.2) $\mathbf{x}_t$ and the last action a_{t-1}, $\tilde{\mathbf{x}}_t = [\mathbf{x}^T\, a_{t-1}]^T$, $a_{-1} = 0$. The architecture is simple, it is one LSTM cell, with output dimension 64, into a fully connected layer outputting the length estimate y normalized to $[-1, 1]$. There are separate models for the MPC and feed-forward network, although the weights of the MPC model were initialized by the feed-forward model to speed up training. An example of estimation values during 4 episodes with different lengths is in figure 6.2.

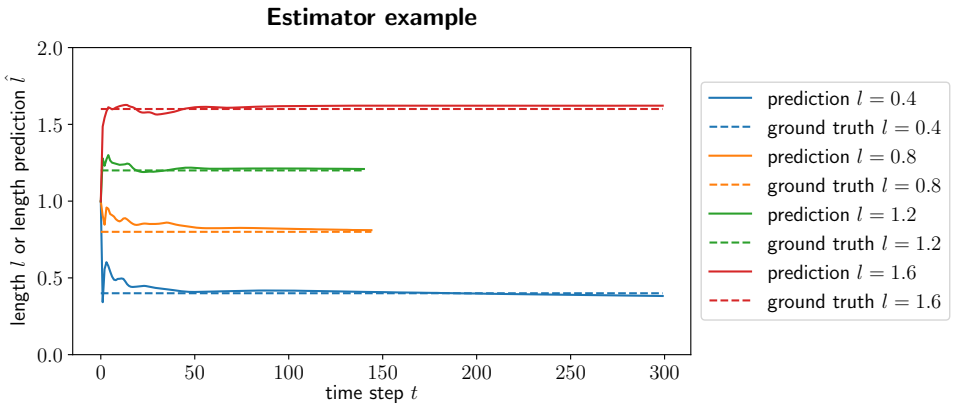


Figure 6.2: Estimator example

Chapter 7

Results

7.1 Policy performance

7.1.1 Evaluation method

Each policy (controller) was evaluated on a fixed set of scenarios. For every combination of $l \in L_{eval} = \{0.1, 0.2, \ldots, 1.8, 1.9\}$ and starting positions $x_0 \in X_{eval} = \{-4, -3.6, \ldots, 3.6, 4\}$, the simulation was run 10 times and mean of cumulative rewards $cr = \sum_{t=1}^{T} r_t$ (T is the number of steps taken in the episode) for each episode has been taken as the metric of performance. In addition, during the evaluation, if the cumulative reward for the episode reached -500, the episode was ended and considered failed, but the cumulative reward of -500 was still used in the metric. The global metric $M(\pi)$ of the policy is the expected cumulative reward over the distribution of parameters and starting states.

$$\mathbb{E}_{l,s_0}[\sum_{t=1}^{T} R_t|\pi] \approx M(\pi) = \frac{1}{|L_{eval}|}\frac{1}{|X_{eval}|}\frac{1}{10}\sum_{l\in L_{eval}}\sum_{x\in X_{eval}}\sum_{e=1}^{10} cr_{\pi,l,x_0,e} \quad (7.1)$$

7.1.2 Comparison

All policies were compared on the cart-pole swing-up environment and the metric $M(\pi)$ was calculated. Fail rate $fr(\pi)$ (episode is failed when cumulative reward ≤ -500) and the average time for time step calculation (step of the environment and calculation of policy) $\bar{\delta t}(\pi)$ were also calculated.

π	$M(\pi)$ [-]	$fr(\pi)$ [-]	$\bar{\delta t}(\pi)$ [ms]
ff-fixed	-87.9	0.067	0.31
ff-random	-110.3	0.032	0.30
ff-oracle	-41.1	0.000	0.30
ff-estim	-40.5	0.001	0.56
ff-hist	-46.1	0.000	0.43
rnn	-293.5	0.319	0.44
mpc-fixed	-195.4	0.094	22.04
mpc-oracle	-37.4	0.000	15.43
mpc-estim	-37.4	0.000	15.62

Table 7.1: Policies comparison

Very important for policy evaluation, and mainly its evaluation of adaptability, is its performance on a certain length. For this a graph of performance on length $M(\pi \mid l)$ is shown. Comparison of neural network policy performances based on length l is in figure 7.1. Since *ff-random* and *rnn* policy are far worse than other policies, they were omitted in further performance comparisons. Performance of MPC compared with NNs, is in figure 7.2.

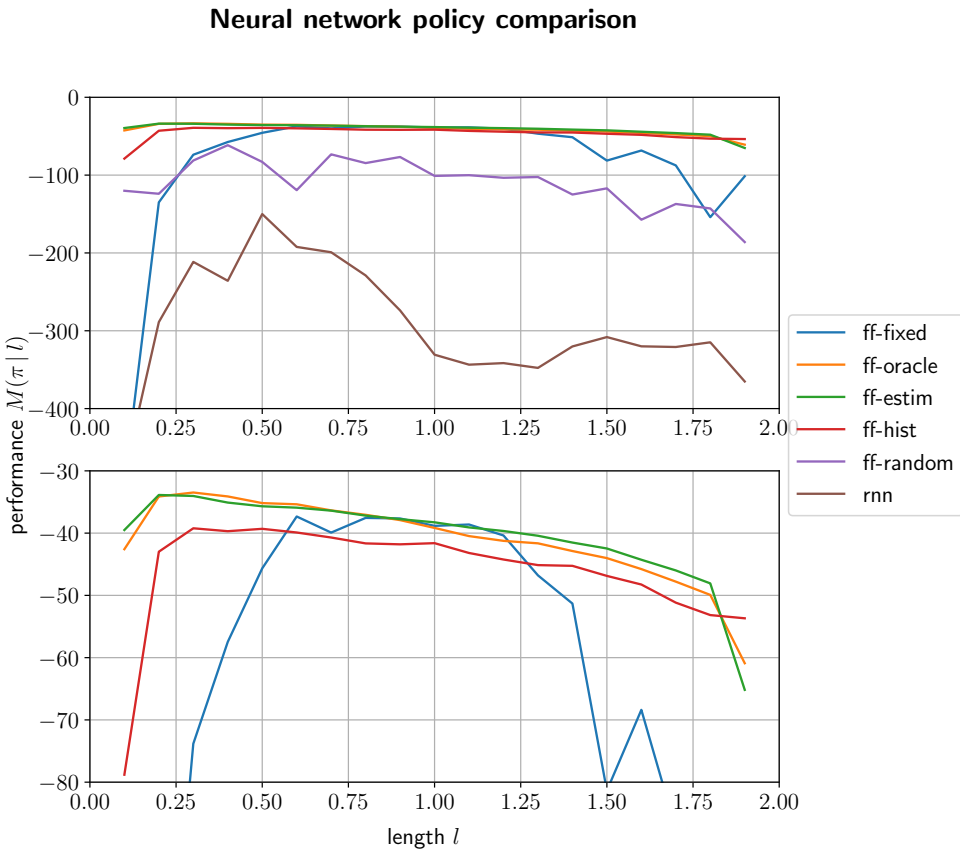


Figure 7.1: Neural network policy performance comparison

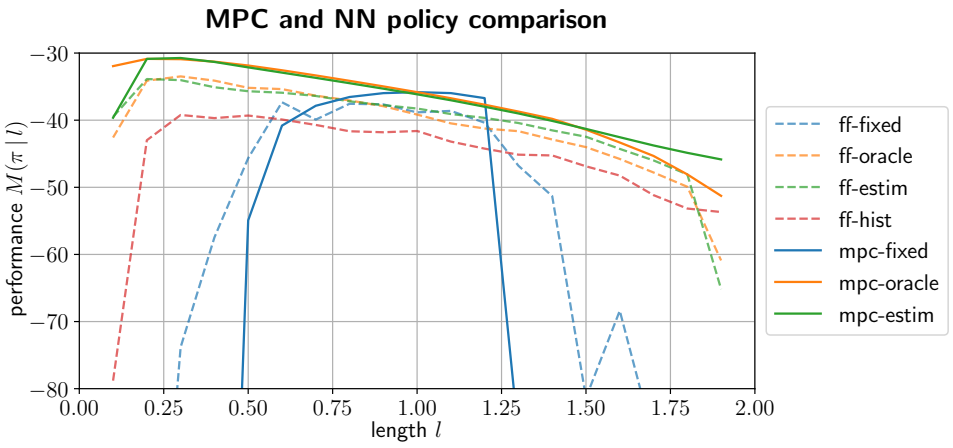


Figure 7.2: MPC and NN policy performance comparison

7.2 Estimation

7.2.1 Evaluation metrics

Four metrics were used for the evaluation of the estimator. First one is the absolute error ϵ, second is the relative error η and third and fourth are threshold steps τ_ϵ and τ_η, after which all predictions in the episode have absolute or relative error lower than threshold $E_\epsilon = 0.05$, $E_\eta = 0.1$. Ground truth length is denoted l and predicted length at step t is $\hat{l}_t$. All these metrics are measured on the same setting as policy performances, 10 times for each length $l \in L_{eval}$ and starting position $x_0 \in X_{eval}$.

$$\epsilon_t = |\hat{l}_t - l| \tag{7.2}$$

$$\eta_t = \frac{|\hat{l}_t - l|}{l} = \frac{\epsilon_t}{l} \tag{7.3}$$

$$\tau_\epsilon = \min_{t \in \{0,\dots,T\}} \{t \,|\, \epsilon_{t'} \leq E_\epsilon,\, t' \in \{t+1,\dots,T\}\} \tag{7.4}$$

$$\tau_\eta = \min_{t \in \{0,\dots,T\}} \{t \,|\, \eta_{t'} \leq E_\eta,\, t' \in \{t+1,\dots,T\}\} \tag{7.5}$$

7.2.2 Performance

The table for mean absolute error $\bar{\epsilon}(\pi)$, mean relative error $\bar{\eta}(\pi)$ and mean threshold steps $\bar{\tau}_\epsilon(\pi)$, $\bar{\tau}_\eta(\pi)$ is table 7.2. To see the estimation improvement

π	$\bar{\epsilon}(\pi)$	$\bar{\eta}(\pi)$	$\bar{\tau}_\epsilon(\pi)$	$\bar{\tau}_\eta(\pi)$
ff	0.028	0.049	39.5	30.5
mpc	0.025	0.060	26.8	30.6

Table 7.2: Estimator performance

over time, both absolute and relative errors are averaged for a fixed time step, this relation is shown in figure 7.4. All 4 metrics are also averaged for a fixed length, to see whether the estimator works over the whole range of possible lengths, these relations are shown in figures 7.4 and 7.5.

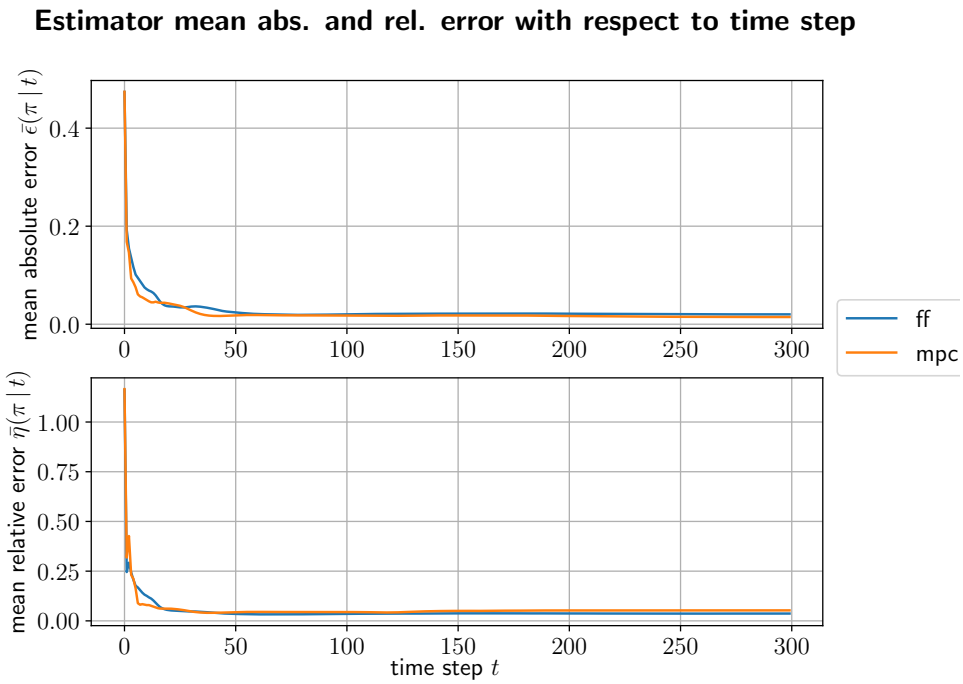


Figure 7.3: Mean absolute and relative error of the estimator with respect to time step

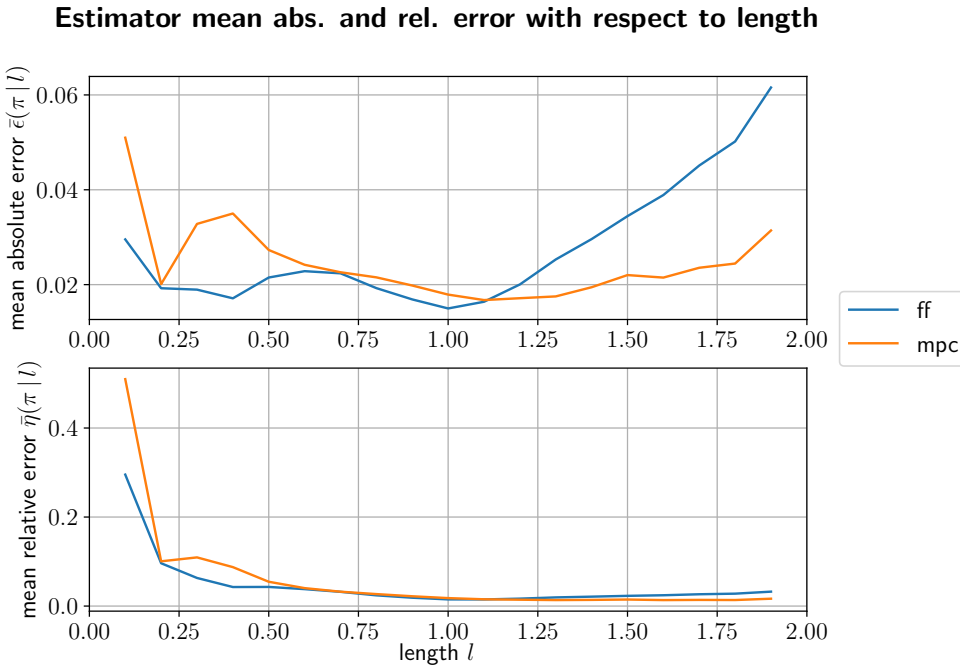


Figure 7.4: Mean absolute and relative error of the estimator with respect to length

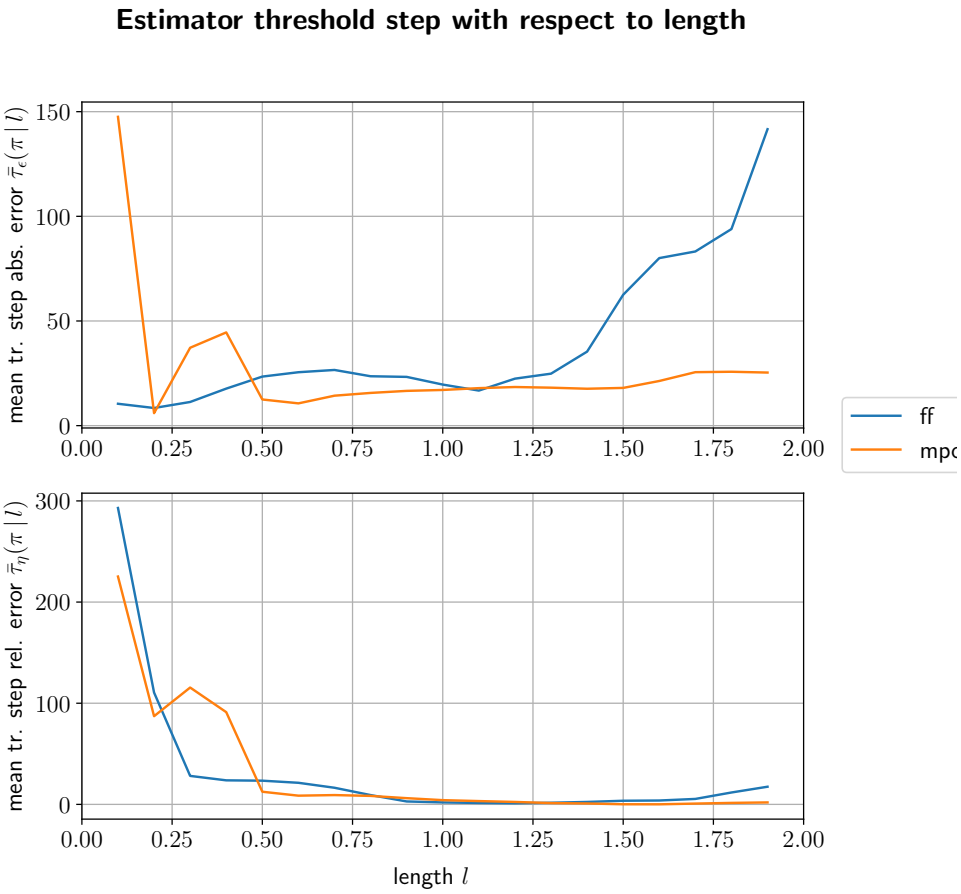


Figure 7.5: Mean threshold step for absolute and relative error of the estimator with respect to length

Chapter 8

Discussion

8.1 Neural network policies

Issues. Unfortunately, the *rnn* model was not successful, mainly because RNNs are very sensitive to the choice of hyper-parameters and their training takes a lot of computation time. The model *ff-random* also was unsuccessful, because the unexplained variance caused big gradient spikes in learning and thus the policy often diverged. Besides training, it was also key to find a good transformation of the observations, suitable for NNs, and tune the action force multiplier of the environment to make it possible for the neural network to act on the environment and still be precise in the control.

Performance. As shown in figure 7.1, models *ff-oracle* and *ff-estim* had the best performance, because of the additional information about length. Surprisingly *ff-estim* was slightly better, however, this could be caused by additional training, as *ff-estim* required more training steps to account for the uncertainty of the estimator. The performance of *ff-hist* was slightly worse than that of *ff-oracle* and *ff-estim*, consistent with expectations, as it had to use some time and energy for inferring the hidden state, before maximizing the reward. Model *ff-fixed*, as expected, had good performance near the assumed condition $l = 1$, but failed when it left the neighbourhood of the assumption, and since *ff-estim* and *ff-hist* performed well even outside the neighbourhood, it demonstrated the *adaptability* of neural network policies.

8.2 Comparison with MPC

As shown in figure 7.2, MPC outperformed the neural network policies, however only slightly, and for the cost of much higher computation time (table 7.1). It can also be seen that *ff-fixed* outperformed *mpc-fixed* in the further parts of the length range from $l = 1$, as the feature of neural networks is their adaptability to unseen cases. Interestingly *mpc-estim* outperformed *mpc-oracle* when $l > 1.5$, possibly because for longer l the dynamics of the model are slower and thus the prediction horizon is too small to get the optimal behaviour and the estimation of shorter l could result in better performance.

8.3 Estimation

As seen in table 7.2 and figures 7.3, 7.4 and 7.5, both models have comparable results, the feed-forward model performs slightly better on bigger lengths, MPC on smaller lengths. Roughly after 30 time steps, the estimator has relative error less than 10 %. The estimation of the length is sufficiently precise, as the performance of *ff-estim* and *mpc-estim* are similar and sometimes even better than their counterparts with ground truth information *ff-oracle* and *mpc-oracle* (figure 7.1 and 7.2).

8.4 Future work

The most important extension of this work is to test the used methods on other systems, possibly with more varying parameters, and to use them on real models and not only simulations. With more computation power, it would also be possible to make a better hyper-parameter testing pipeline to speed-up the training process. Additionally, it would be interesting to use and compare other deep reinforcement learning algorithms, such as Hindsight Experience Replay [46].

Chapter 9

Conclusion

Environment. An environment simulating a non-linear parameter varying system has been created. Additionally, unified method of benchmarking policies was created to compare both classic methods and deep reinforcement learning policies.

Adaptability of neural networks policies. This thesis completed its specification of demonstrating the adaptability of neural networks policies in the control of parameter varying systems, on its simulation of a non-linear system. Unfortunately, the training of recurrent policy was unsuccessful, due to limited resources.

Comparison with classic control. All neural network policies were compared with 3 fully non-linear MPC controllers, which achieve almost optimal behaviour in this setting.

Estimation. As an extension of the specification, an LSTM network estimator of the hidden parameter was trained and its information was used in the control policy.

Subjective evaluation. As the author of this thesis, the specification motivated me to learn a lot about two key fields. First one was deep reinforcement learning and the second was dynamic systems control. Deep reinforcement

learning has a steep learning curve, as the algorithms are very complicated. Fortunately an working implementation exists in the library Stable-Baselines [44]. In the field of dynamic systems control, I expanded my knowledge with information from optimal control theory, especially about the design of very powerful MPC controllers.

Appendix A

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning; an Introduction*, 2nd ed. MIT Press, 2018. [Online]. Available: http://www.incompleteideas.net/book/the-book.html

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[4] C. Greatwood and A. G. Richards, "Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control," *Autonomous Robots*, vol. 43, pp. 1681 – 1693, 2019.

[5] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 2016.

[6] O. Kilinc, Y. Hu, and G. Montana, "Reinforcement learning for robotic manipulation using simulated locomotion demonstrations," 2019.

[7] M. M. Noel and B. J. Pandian, "Control of a nonlinear liquid level system using a new artificial neural network based reinforcement learning approach," *Applied Soft Computing*, vol. 23, pp. 444 – 451, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494614003111

[8] O. Ogunmolu, X. Gu, S. Jiang, and N. Gans, "Nonlinear systems identification using deep dynamic neural networks," 2016.

[9] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl^2: Fast reinforcement learning via slow reinforcement learning," 2016.

[10] Z. Chen and E. N. Brown, "State space model," *Scholarpedia*, vol. 8, no. 3, p. 30868, 2013, revision #189565.

[11] Z. Guang, Z. Heming, and B. Liang, "Attitude dynamics of spacecraft with time-varying inertia during on-orbit refueling," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 8, pp. 1744–1754, 2018. [Online]. Available: https://doi.org/10.2514/1.G003474

[12] P. Poupart, *Partially Observable Markov Decision Processes*. Boston, MA: Springer US, 2010, pp. 754–760. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_629

[13] R. Bellman, "Dynamic programming and stochastic control processes," *Information and Control*, vol. 1, no. 3, pp. 228 – 239, 1958. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019995858800030

[14] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

[15] M. M. Polycarpou, "Stable adaptive neural control scheme for nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 41, no. 3, pp. 447–451, 1996.

[16] K. S. Narendra and J. Balakrishnan, "Adaptive control using multiple models," *IEEE Transactions on Automatic Control*, vol. 42, no. 2, pp. 171–187, 1997.

[17] C. P. Bechlioulis and G. A. Rovithakis, "Robust adaptive control of feedback linearizable mimo nonlinear systems with prescribed performance," *IEEE Transactions on Automatic Control*, vol. 53, no. 9, pp. 2090–2099, 2008.

[18] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," 2017.

[19] B. Gaudet, R. Linares, and R. Furfaro, "Adaptive guidance and integrated navigation with reinforcement meta-learning," 04 2019.

[20] A. K. Singh and P. Gaur, "Adaptive control for non-linear systems using artificial neural network and its application applied on inverted pendulum," in *India International Conference on Power Electronics 2010 (IICPE2010)*, 2011, pp. 1–8.

[21] L. E. Garza-Castañón, R. Morales-Menendez, A. Favela-Contreras, A. Raimondi, A. Vargas-Martínez, and V. Puig, "Artificial neural networks, adaptive and classical control for ftc of linear parameters varying systems," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 13 540 – 13 545, 2011, 18th IFAC World Congress. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S147466701645799X

[22] A. Kratsios, "The universal approximation property: Characterizations, existence, and a canonical topology for deep-learning," 2019.

[23] H. Siegelmann and E. Sontag, "On the computational power of neural nets," *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 132 – 150, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000085710136

[24] K. Zimmermann, "Learning for vision V - Training and layers," 2019, accessed 4.8.2020. [Online]. Available: https://cw.fel.cvut.cz/b191/_media/courses/b3b33vir/learning_for_vision_v_layers.pdf

[25] ——, "Learning for vision II - Neural networks," 2019, accessed 4.8.2020. [Online]. Available: https://cw.fel.cvut.cz/b191/_media/courses/b3b33vir/learning_for_vision_ii_neural_nets.pdf

[26] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *30th International Conference on Machine Learning, ICML 2013*, pp. 1139–1147, 01 2013.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[28] R. Grosse, "Intro to neural networks and machine learning, lecture 15: Exploding and vanishing gradients," 2017, accessed 4.8.2020.

[29] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000. [Online]. Available: https://doi.org/10.1162/089976600300015015

[30] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[32] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015.

[33] Z. Ahmed, N. L. Roux, M. Norouzi, and D. Schuurmans, "Understanding the impact of entropy on policy optimization," 2018.

[34] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2015.

[35] G. F. Franklin, J. D. Powell, A. Emami-Naeini, and H. S. Sanjay, *Feedback control of dynamic systems*, seventh, global ed. Boston: Pearson, 2015.

[36] Z. Hurák, "Introduction to numerical simulation," 2018, accessed on 30. 7. 2020. [Online]. Available: https://moodle.fel.cvut.cz/pluginfile.php/210391/mod_resource/content/4/msd_11_intro_to_numerical_simulation_of_ODE.pdf

[37] Grune, J. P. Pannek, Jurgen, and Lars, *Nonlinear Model Predictive Control: Theory and Algorithms*, 2nd ed. Cham: Springer, 2016;2017;.

[38] S. Lucia, A. Tatulea-Codrean, C. Schoppmeyer, and S. Engell, "Rapid development of modular and sustainable nonlinear model predictive control solutions," *Control Engineering Practice*, vol. 60, p. 51–62, 03 2017. [Online]. Available: https://www.do-mpc.com/

[39] C. V. Rao, J. B. Rawlings, and D. Q. Mayne, "Constrained state estimation for nonlinear discrete-time systems: stability and moving horizon approximations," *IEEE Transactions on Automatic Control*, vol. 48, no. 2, pp. 246–258, 2003.

[40] Z. Hurák, "Lagrange's equations; intro to an energy-based analytical modeling methodology," 2018, accessed on 30. 7. 2020. [Online]. Available: https://moodle.fel.cvut.cz/pluginfile.php/210341/mod_resource/content/3/msd_6_intro_to_lagrange_technique.pdf

[41] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016. [Online]. Available: https://pybullet.org/

[42] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[43] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. [Online]. Available: https://gym.openai.com/

[44] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner,

L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[46] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," 2017.

Appendix B

Notation table

symbol	explanation
$\mathbf{a} \in \mathbf{R}^n$	column vector of length n
$\mathbf{A} \in \mathbf{R}^{n \times m}$	matrix with n rows and m columns n
$\mathcal{S}$	state space
$\mathcal{A}$	action (input) space
$\mathcal{O}$	observation (output) space
s_t, $s(t)$, $\mathbf{x}_t$, $\mathbf{x}(t)$	state in time t
a_t, $a(t)$, $\mathbf{u}_t$, $\mathbf{u}(t)$	action (input) in time t
o_t, $o(t)$, $\mathbf{y}_t$, $\mathbf{y}(t)$	observation (output) in time t
r_t	reward in time t
g_t	discounted gain in time t
$\mathbb{P}[A\|B]$	probability of A with prior B
$\mathbb{E}[A\|B]$	expected value of A with prior B
$\hat{\mathbb{E}}[A\|B]$	empirical average of A with prior B
π	policy
$R(s,a)$	reward function
$V^\pi(s)$	value function for policy π
$Q^\pi(s,a)$	action-value function for policy π
$D^\pi(s,a)$	advantage function for policy π
S_t	random variable of state in time t
A_t	random variable of action in time t
O_t	random variable of observation in time t
R_t	random variable of reward in time t
G_t	random variable of discounted gain in time t
$\hat{D}_t$	estimation of advantage in time t

Table B.1: Notation table - part 1

$\mathbf{w}$	vector of parameters
$f(\mathbf{x} \mid \mathbf{w})$, $f_{\mathbf{w}}(\mathbf{x})$	function f parametrized by $\mathbf{w}$
$\frac{\partial f}{\partial \mathbf{w}}$	row vector (matrix) of partial derivatives of f wrt. $\mathbf{w}$
$\nabla_{\mathbf{w}} f$	column vector of partial derivatives of f wrt. $\mathbf{w}$
L	loss (minimized in optimization)
J	objective (maximized in optimization)
$\dot{\mathbf{x}}$	time derivation of $\mathbf{x}$
$M(\pi)$	performance metric of policy π
$M(\pi \mid l)$	performance metric of policy π on length l
ϵ	absolute error
η	relative error
τ	threshold step

Table B.2: Notation table - part 2