Undergraduate/Master's Thesis

Your Title Goes Here

Robin Uhrich

Examiner: Prof. Dr. Joschka Boedecker
Advisers: Jasper Hoffman

University of Freiburg
Faculty of Engineering
Department of Computer Science
Neurobotics Lab

September 9, 2023

Writing Period

05. 07. 2016 – 05. 10. 2016

Examiner

Prof. Dr. Joschka Boedecker

Second Examiner

Prof. Dr. Wile E. Coyote

Advisers

Jasper Hoffman

Bachelor Thesis

Your Title Goes Here

Robin Uhrich

Gutachter: Prof. Dr. Joschka Boedecker
Betreuer: Jasper Hoffman

Albert-Ludwigs-Universität Freiburg
Technische Fakultät
Institut für Informatik
Lehrstuhl für Neurorobotik

September 9, 2023

Bearbeitungszeit

05. 07. 2016 – 05. 10. 2016

Gutachter

Prof. Dr. Joschka Boedecker

Zweitgutachter

Prof. Dr. Wile E. Coyote

Betreuer

Jasper Hoffman

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

_____ _____

Place, Date Signature

Abstract

foo bar

Zusammenfassung

German version is only needed for an undergraduate thesis.

Contents

1 Introduction **1**

1.1 Template Structure 1

1.2 setup.tex 2

1.3 Advice 4

2 Related Work **7**

3 Background **9**

3.1 RL Framework 9

3.2 Generalized Policy Iteration 13

3.3 Soft Actor Critic 15

3.3.1 Actor-Critic Algorithms 15

3.3.2 Entropy Regularization 17

3.3.3 Algorithm Architecture 19

3.3.4 Advantages 22

3.4 Neural Networks 24

3.4.1 The Rise of Deep Learning 25

3.4.2 Neural Network Architecture 25

3.4.3 Train Neural Networks 28

3.4.4 Applications of Neural Networks 29

3.5 Variational Autoencoder 30

3.5.1 Autoencoders 30

3.5.2 Variational Autoencoders . . . 31
3.5.3 Conditional Variational Autoencoders . . . 33
3.5.4 Evidence Lower Bound . . . 34
3.5.5 VAEs and CVAEs in Practice . . . 36
3.6 Kinematics . . . 37
3.6.1 Forward kinematics . . . 38
3.6.2 Inverse kinematics . . . 38

4 Methodology **43**
4.1 Research Idea . . . 43
4.2 RL Environment . . . 44
4.2.1 State Space . . . 45
4.2.2 Action Space . . . 46
4.2.3 Reward Function . . . 47
4.3 Dataset Creation . . . 47
4.3.1 Uniform Sampling . . . 48
4.3.2 Expert Guidance . . . 50
4.4 Latent Criterion . . . 52
4.4.1 Kulback Leiber Divergence . . . 52
4.4.2 Reconstruction Loss . . . 53
4.5 Learning the Latent Model . . . 54
4.6 Software . . . 56
4.6.1 Inverse kinematics Environment . . . 56
4.6.2 Latent Module . . . 56
4.6.3 Soft actor critc . . . 56

5 Experiments **57**
5.1 Baseline SAC . . . 58
5.2 VAE . . . 59
5.2.1 Distance Loss . . . 62

5.2.2 Distance plus Imitation Loss 64
5.3 SAC with VAE 65
5.4 Supervised 70
5.4.1 Distance Loss 71
5.4.2 Distance + Imitation Loss 71
5.5 SAC with Supervised 72

6 Discussion **75**
6.1 Baseline SAC 75
6.2 Latent Model 78
6.3 SAC with Latent Model 80

7 Conclusion **83**

8 Acknowledgments **85**

Bibliography **89**

9 Appendix **91**
9.1 Hyperparameters 91
9.2 Additional Experiment Plots 91

List of Figures

3.1 interaction between agent and the environment 10
3.2 policy iteration concept 14
3.3 Policy iteration funnel 14
3.4 neural network and neuron schema 26
3.5 common activation functions in a neural network 27
3.6 advanced neural network architectures 27
3.7 Autoencoder schematics 31
3.8 Variational Autoencoder schematics 32
3.9 Conditional Variational Autoencoder schematics 33
3.10 CCD geometry 41
3.11 ccd trajectory 41
3.12 CCD iteration heatmap 42

4.1 Research idea 44
4.2 Plane Robot Environment 45
4.3 Reward function 48
4.4 Uniform dataset properties 49
4.5 action correlation CCD 51
4.6 Runtime complexity CCD 52
4.7 Target Gaussian Schematics 55

5.1 SAC baseline experiment results 58

5.2 SAC baseline inference 60
5.3 SAC baseline inference 61
5.4 VAE validation results, only distance loss and latent = 4 62
5.5 VAE latent dimension comparison on reconstruction loss 63
5.6 VAE validation results with imitation loss 65
5.7 SAC + VAE on latent dim = 4 67
5.8 SAC + VAE latent dimension comparison 67
5.9 SAC + VAE latent dimension comparison 68
5.10 SAC + VAE on latent dim = 4 69
5.11 action correlation comparison 70
5.12 Supervised Distance Loss 71
5.13 Supervised Distance and Imitation Loss 72
5.14 Supervised Distance and Imitation Loss 73
5.15 Supervised Distance and Imitation Loss 74

6.1 CCD iteration heatmap 76
6.2 action correlation 76
6.3 SAC min distance heatmap 78
6.4 CCD iteration heatmap 79

9.1 alpha loss with $N = 15$ 92
9.2 RL Agent iteration evaluation baseline 95
9.3 RL Agent iteration evaluation on VAE with latent = 2 96
9.4 RL Agent iteration evaluation on VAE with latent = 4 97
9.5 RL Agent iteration evaluation on VAE with latent = 8 98

List of Tables

4.1 Latent workflow parameter 55

5.1 Used VAE checkpoints for SAC 66
5.2 policy log probabilities 72

6.1 SAC Solved ratio 79

9.1 Environment Hyperparameter 91
9.2 VAE Hyperparameter 93
9.3 SAC Hyperparameter 94

List of Algorithms

1 Soft Actor Critic 23

2 Stochastic gradient descent 28

3 Forward Kinematics 38

4 Cyclic Coordinate Descent Pseudo Code 40

5 Expert Guided Dataset Creation 50

1 Introduction

This is a template for an undergraduate or master's thesis. The first sections are concerned with the template itself. If this is your first thesis, consider reading Section 1.3.

The structure of this thesis is only an example. Discuss with your adviser what structure fits best for your thesis.

1.1 Template Structure

- To compile the document either run the makefile or run your compiler on the file 'thesis_main.tex'. The included makefile requires latexmk which automatically runs bibtex and recompiles your thesis as often as needed. Also it automatically places all output files (aux, bbl, ...) in the folder 'out'. As the pdf also goes in there, the makefile copies the pdf file to the parent folder. There is also a makefile in the chapters folder, to ensure you can also compile from this directory.

- The file 'setup.tex' includes the packages and defines commands. For more details see Section 1.2.

- Each chapter goes into a separate document, the files can be found in the folder chapters.

- The bib folder contains the .bib files, I'd suggest to create multiple bib files for different topics. If you add some or rename the existing ones, don't forget to also change this in thesis_main.tex. You can then cite as usual [1, 2, 3].

- The template is written in a way that eases the switch from scrbook to book class. So if you're not a fan of KOMA you can just replace the documentclass in the main file. The only thing that needs to be changed in setup.tex is the caption styling, see the comments there.

1.2 setup.tex

Edit setup.tex according to your needs. The file contains two sections, one for package includes, and one for defining commands. At the end of the includes and commands there is a section that can safely be removed if you don't need algorithms or tikz. Also don't forget to adapt the pdf hypersetup!!
setup.tex defines:

- some new commands for remembering to do stuff:
  - `\todo{Do this!}`: **(TODO: Do this!)**
  - `\extend{Write more when new results are out!}`:
    (EXTEND: Write more when new results are out!)
  - `\draft{Hacky text!}`: **(DRAFT: Hacky text!)**

- some commands for referencing, 'in `\chapref{chap:introduction}`' produces 'in Chapter 1'
  - `\chapref{}`

  - `\secref{sec:XY}`
  - `\eqref{}`
  - `\figref{}`
  - `\tabref{}`

- the colors of the Uni's corporate design, accessible with
  `{\color{UniX} Colored Text}`
  - UniBlue
  - UniRed
  - UniGrey

- a command for naming matrices `\mat{G}`, $\mathbf{G}$, and naming vectors `\vec{a}`, $\mathbf{a}$. This overwrites the default behavior of having an arrow over vectors, sticking to the naming conventions normal font for scalars, bold-lowercase for vectors, and bold-uppercase for matrices.

- named equations:

```
\begin{align}
    d(a,b) &= d(b,a)\\ \eqname{symmetry}
\end{align}
```

$$d(a,b) = d(b,a) \tag{1.1}$$

symmetry

1.3 Advice

This section gives some advice how to write a thesis ranging from writing style to formatting. To be sure, ask your advisor about his/her preferences.
For a more complete list we recommend to read Donald Knuth's paper on mathematical writing. (At least the first paragraph). `http://jmlr.csail.mit.edu/reviewing-papers/knuth_mathematical_writing.pdf`

- If you use formulae pay close attention to be consistent throughout the thesis!

- In a thesis you don't write 'In [24] the data is..'. You have more space than in a paper, so write 'AuthorXY et al. prepare the data... [24]'. Also pay attention to the placement: The citation is at the end of the sentence before the full stop with a no-break space. `... last word~\cite{XY}.`

- Pay attention to comma usage, there is a big difference between English and German. '...the fact that bla...' etc.

- Do not write 'don't ', 'can't' etc. Write 'do not', 'can not'.

- If an equation is at the end of a sentence, add a full stop. If it's not the end, add a comma: $a = b + c \quad (1)$,

- Avoid footnotes if possible.

- Use ``'' for citing, not "".

- It's important to look for spelling mistakes in your thesis. There are also tools like aspell that can help you find such mistakes. This is never an excuse not to properly read your thesis again, but it can help. You can find an introduction under `https://git.fachschaft.tf/fachschaft/aspell`.

- If have things like a graph or any other drawings consider using tikz, if you need function graphs or diagrams consider using pgfplots. This has the advantage that the style will be more consistent (same font, formatting options etc.) than when you use some external program.

- Discuss with your advisor whether to use passive voice or not. In most computer science papers passive voice is avoided. It's harder to read, more likely to produce errors, and most of the times less precise. Of course there are situations where the passive voice fits but in scientific papers they are rare. Compare the sentence: 'We created the wheel to solve this.' to 'The wheel was created to solve this', you don't know who did it, making it harder to understand what is your contribution and what is not.

- In tables avoid vertical lines, keep them clean and neat. See **??** for an example. More details can be found in the 'Small Guide to Making Nice Tables' `https://www.inf.ethz.ch/personal/markusp/teaching/guides/guide-tables.pdf`

2 Related Work

LASER: Learning Latent Action Space for Efficient Reinforcement Learning

3 Background

The following part is to provide a short introduction into the theory of reinforcement learning, the Soft Actor Critic Algorithm (SAC) as well to Variational Autoencoders (VAE) and Conditional Variational Autoencoders (CVAE). **(TODO: update!!!)**

3.1 RL Framework

Learning is a topic that is present in all our lives since their beginning. We all learn to move our arms, like an infant that is wiggling around, learn to walk, to speak and more or less any other skill that we find in our personal repository until now.

Reinforcement Learning (RL) is about to have a computational approach to this kind of learning. In the class of RL-Problems the environment provides a challenge like, riding a bicycle. But it is not told how this goal can be achieved. Therefore it has be learned how to derive actions from situations on or off the bike in order to complete this task or - speaking of the numerical approach - maximize a reward function. In most cases the problem solving methods that are classified as RL-methods are applied on tasks that require not only a sequence of actions to interact with an environment but also some kind of feedback. Therefore these setups are designed to provide this kind of feedback. Either if it is an immediate feedback like an additional time step the agent has not crashed the bike or if it is more like a long shot feedback, for example

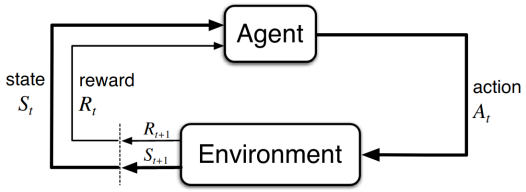


Figure 3.1: interaction between agent and the environment

a bonus that the agent has earned after winning board game. These principals can be boiled down to three main characteristics of reinforcement learning problems:

- There is a closed loop mechanism between actions and feedback that is provided by the environment where the agent is operating in.
- The agent is not given any direct instructions how to solve the given problem.
- The actions an agent is performing have consequences over time. Either for him or the environment he is working in.

Before we can have an exemplary look on the closed loop mechanics, we would like to introduce a couple of key components of RL including the Markov Decision Process.

(TODO: cite figure)

Every agent exists in a specified environment and also interacts within a sequence of discrete time steps $t \in \mathbb{N}$. In order to do so the agent has to perceive the environment through a **state** representation $S_t \in \mathcal{S}$ which is an element of all possible states, the state space $\mathcal{S}$. To really interact with the environment the agent has to perform an **action** $A_t \in \mathcal{A}(S_t)$ where $\mathcal{A}(S_t)$ is the set of all possible actions in given state S_t. To chose an action the agent calculates for each possible action A_t a probability $\pi_{\theta,t}(A_t|S_t)$ for a given state S_t and chooses the maximum with respect to the given actions. To achieve a more compact notation we will refer to the action $A_t = a$ and the state $S_t = s$. To avoid confusions with the number $\pi \approx 3.141$ we refer to the

policy always as a parameterized policy $\pi_\theta = \pi_{\theta,t}(a|s)$. This probability distribution is called the **policy**. After the agent has chosen and executed an action, which also means the environment will move one time step further $t \to t+1$, the environment will return the next state S_{t+1} and the agent will perceive a **reward** $R_{t+1} \in \mathbb{R}$ with $\mathbb{R}$ as the set of all possible rewards.

The long term goal of the agent is to adapt its policy to maximize the total amount reward the agent receives from the environment. These key terminology can also be found in the notation of a Markov Decision Process which is classified as a 4-Tuple of: $(\mathcal{S}, \mathcal{A}, P_a(s,s'), R_a(s,s'))$ with $P_a(s,s')$ as a probability distribution that action a in state state s will lead to the next state $s' = s_{t+1}$ and $R_a(s,s')$ as reward function for transitioning from s to s' with a.

(TODO: Explain the concept of reinforcement learning and its applications Describe the Markov decision process (MDP) and the Bellman equation Discuss the challenges of using reinforcement learning in practice)

(TODO: Es wird gern gesehen: Optimal value function optimal policy / Tradeoff, Sutton Barto. Generalized Policy iteration. Es geht darum einaml ne policy und ne value function zu fnden, Bellman equations)

The overall goal of an agent is to maximize its received cumulative return G_t for one run until time step T. This is metric is denoted in Equation (3.1). In Equation (3.1) γ is the discount rate, $0 \leq \gamma \leq 1$, a parameter to regularize the *importance* of individual received rewards over time.

(TODO: cite sutton barto)

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{3.1}$$

$$= R_{t+1} + \gamma G_{t+1} \tag{3.2}$$

Unfortunately the individual rewards R_t are highly dependent on state action pairs in the future, so they are not accessible at the current time step. To solve this problem reinforcement algorithms are trying to maximize the expected return $v_{\pi_\theta}(s)$ as in Equation (3.3), or value function, for a given state. For Markov decision process we can define the value function as in Equation (3.4).

(TODO: cite sutton barto)

$$v_{\pi_\theta}(s) \doteq \mathbb{E}_{\pi_\theta}[G_t | S_t = s] \tag{3.3}$$

$$\begin{aligned}
&\overset{(3.2)}{=} \mathbb{E}_{\pi_\theta}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \sum_a \pi_\theta(a|s) \sum_{s'} \sum_r P(s', r|s, a) \left[r + \gamma \mathbb{E}_{\pi_\theta}[G_{t+1} | S_{t+1} = s'] \right] \\
&= \sum_a \pi_\theta(a|s) \sum_{s', r} P(s', r|s, a) \left[r + \gamma v_{\pi_\theta}(s' m) \right]
\end{aligned} \tag{3.4}$$

For a finite Markov decision process we can also find a the optimal value function $v_*(s)$ with respect to the optimal policy in Equation (3.7).

$$q_{\pi_\theta}(s, a) = \sum_{s', r} P(s', r|s, a) \left[r + \gamma \sum_{a'} \pi_\theta(a'|s') q_{\pi_\theta}(s', a') \right] \tag{3.5}$$

$$v_*(s) \doteq \max_{\pi_\theta} v_{\pi_\theta}(s) \tag{3.6}$$

$$= \max_a \sum_{s',r} P(s', r|s, a)[r + \gamma v_*(s')] \tag{3.7}$$

Because Equation (3.6) holds this implies the reinforcement learning problem in Equation (3.8) for the optimal policy π^*_θ and a performance measure $J(\pi_\theta) = \mathbb{E}_{\pi_\theta}[G_t|S_t = s]$ as following:

$$\pi^*_\theta = \arg\max_{\pi_\theta} J(\pi_\theta) \tag{3.8}$$

$$= \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}[G_t|S_t = s] \tag{3.9}$$

$$\overset{(3.3)}{=} \arg\max_{\pi_\theta} v_{\pi_\theta}(s) \tag{3.10}$$

$$q_*(s, a) = \sum_{s',r} P(s', r|s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right] \tag{3.11}$$

3.2 Generalized Policy Iteration

This section is about to provide an introduction to the idea of policy iteration. For a more detailed look into the topic it is recommended refer to the Reinforcement Learning Textbook from Richard Sutton and Andrew Barto.

One fundamental concept for reinforcement learning algorithms is policy iteration. At Its core it is described as the alternating interaction between a policy improvement step and a policy evaluation step as in Figure 3.2. This interaction goes back and forth during the execution of most reinforcement learning algorithms until a state of convergence in the value function and policy has arrived. That means both value

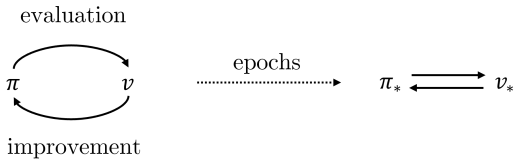


Figure 3.2: policy iteration concept

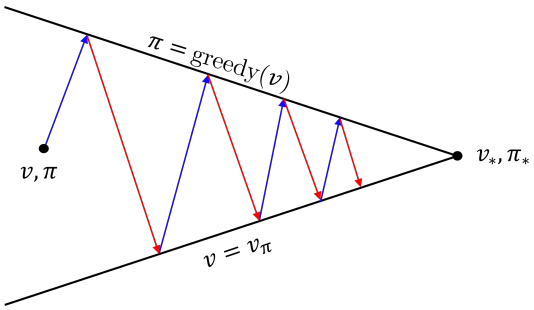


Figure 3.3: Policy iteration funnel. The red arrows are a symbolic policy improvement step which is moving away from a precise value function. The blue arrows are a symbolic policy evaluation step which is moving the current policy further away from a greedy and thus stable policy. Both criterions are moving over time closer together until convergence with the optimal policy π_* and the optimal value function v_*

function and policy are optimal for the given problem. This state can be reached if the "value function is consistent with the current policy, and the policy stabilizes only when it is greedy with respect to the current value function". This can be also seen in the Bellman optimally equation as in Equation (3.7).

We can also think about this process as a tradeoff between improving the policy towards an optimal greedy behavior which makes the value function incorrect (blue arrows indicate a policy improvement step) and adapting the value function with respect to the current policy which makes the policy automatically less optimal (red arrows indicate a policy evaluation step). **(TODO: Why?)** Despite this tradeoff the algorithm tends to find an optimal solution for the policy and the value function in the long run. **(TODO: Where is the proof)**

3.3 Soft Actor Critic

Soft Actor-Critic was introduced by **(TODO: cite paper)** in **(TODO: year)**. It is an algorithm belonging to the family of model free, off-policy, actor-critic reinforcement learning algorithms. The family of actor-critic algorithms addresses challenges in exploration and sample efficiency. Similar to other actor-critic algorithms it also employs the alternating concept of a policy improvement step and a policy evaluation step. In contrast to other actor-critic algorithms like XXXX **(TODO: fill)** it uses the concept of entropy regularization and uses a stochastic policy for promoting exploration in the reinforcement learning environment. By jointly optimizing the policy and value functions using the maximum entropy objective enabled by entropy regularized, SAC effectively explores the environment while seeking to maximize rewards. This approach results in robust and adaptive policies that can efficiently handle various RL tasks. **(TODO: examples by paper)**

In this section will introduce the concept of actor-critic algorithms, provide a detailed description of the SAC algorithm architecture, including entropy regularization and explain how to train the algorithm under the maximum entropy objective.

3.3.1 Actor-Critic Algorithms

The basic actor-critic architecture is a type of reinforcement learning algorithm that consists of two components: an actor and a critic. The actor also referred to as the policy π_θ is responsible for selecting actions based on the current state, while the critic also referred to as a value function v_{π_θ} or a state value function q_{π_θ}, evaluates the quality of the actor's actions by estimating the expected return. Like as introduced in Section 3.2 the actor uses the feedback from the critic to adjust its policy and improve its performance.

(TODO: Maybe in RL Formulation or a separated section?) One additional factor to distinguish between reinforcement learning algorithms and also actor-critic algorithms is by either on-policy learning or off-policy learning. On-policy learning or off-policy learning refer to different methods for updating the policy in reinforcement learning. In on-policy learning, the agent learns from the data generated by its current policy, while in off-policy learning, the agent learns from data generated by a different policy. On-policy learning can be more stable, but it may require more data to converge. Off-policy learning can be more efficient, but it can be more sensitive to the quality of the data. **(TODO: cite sutton barto)**

The advantage of using a critic in the actor-critic algorithm is: it provides a more stable feedback signal than using only rewards. The critic estimates the expected return from a state, for the value function, or a state action pair for the action-value function, which takes into account the long-term consequences of actions. This allows the actor to learn from the critic's feedback and improve its performance more efficiently than if it only received reward signals. Additionally, the critic can help to generalize across different states and actions, improving the overall performance of the algorithm.

One limitation of the basic actor-critic architecture is that it can suffer from high variance and slow convergence due to the interaction between the actor and critic. This can be addressed through the use of techniques such as baseline subtraction and eligibility traces. **(TODO: look for paper) (TODO: cite sutton barto)**

Popular examples for actor critic algorithms are:

- Deep Deterministic Policy Gradient
- Soft Actor-Critic
- Proximal Policy optimization

3.3.2 Entropy Regularization

Entropy regularization is a policy regularization technique by incorporating the policy entropy, used to encourage the policy to explore a diverse range of actions during training. In SAC, entropy regularization is achieved by adding the entropy $H(\pi_\theta)$ as in Equation (3.12) to the policy objective function in Equation (3.27).

$$\begin{aligned} H(\pi_\theta) &= \mathop{\mathbb{E}}_{s\sim\mathcal{S},a\sim\mathcal{A}}[-\log(\pi_\theta(a|s))] \\ &= -\log(\pi_\theta(a|s)) \end{aligned} \tag{3.12}$$

(TODO: is this correct?) (TODO: something has to be cleared up with $H(\pi_\theta)! = H(\pi_\theta(\cdot|s))$)

Including entropy into the objective function encourages the policy to generate actions with higher entropy (higher randomness), leading to more exploration, possibly accelerate training and preventing converging to a poor local optimum. **(TODO: cite: spinning up open ai)**

The randomness or uncertainty of a policy for a given state action pair $\pi_\theta(a|s)$ can be computed by seeing $\pi_\theta(a|s)$ as a density function and taking $\pi_\theta(\cdot|s) = \mathcal{N}\left(\mu(\pi_\theta(\cdot|s)), \sigma(\pi_\theta(\cdot|s))\right)$ as a parameterized normal distribution from which the action $a \sim \pi_\theta(\dot{|}s)$ is sampled. Therefor it possible to acquire the probability $\pi_\theta(a|s)$ of an action a with as in Equation (3.13).

$$\pi_\theta(a|s) = \frac{1}{\sigma(\pi_\theta(\cdot|s))\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(a-\mu(\pi_\theta(\cdot|s)))^2}{\sigma(\pi_\theta(\cdot|s))^2}\right) \tag{3.13}$$

As a result of entropy regularization in each time step both value function v_{π_θ} and action-valuefunction q_{π_θ} become affected and turn into their counterpart $v_{\pi_\theta,H}$ and $q_{\pi_\theta,H}$ in Equation (3.14) and Equation (3.15) below. Note that Equation (3.16) is the recursive version of Equation (3.15).

$$v_{\pi_\theta,H}(s) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t\left(R_t + \alpha H(\pi_\theta(\cdot|s_t))\right) \mid s_0 = s\right] \tag{3.14}$$

$$q_{\pi_\theta,H}(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t R_t + \alpha\sum_{t=1}^{T}\gamma^t H(\pi_\theta(\cdot|s_t)) \mid s_0 = s, a_0 = a\right] \tag{3.15}$$

$$= \mathop{\mathbb{E}}_{s'\sim P, a'\sim\pi_\theta}\left[R_t + \gamma(q_{\pi_\theta,H}(s',a') + \alpha H(\pi_\theta(\cdot|s')))\right] \tag{3.16}$$

(TODO: include entropy in the first time step -> look at the code and search for the appropriate paper)

This changes subsequently also the original relation between v_{π_θ} and q_{π_θ}. $v_{\pi_\theta,H}$ and $q_{\pi_\theta,H}$ are connected vai Equation (3.17).

$$v_{\pi_\theta,H}(s) = \mathbb{E}_{\pi_\theta}\left[q_{\pi_\theta}(s,a)\right] + \alpha H(\pi_\theta) \tag{3.17}$$

(TODO: proof in appendix)

Note that we introduce a new parameter α into $v_{\pi_\theta,H}$ and $q_{\pi_\theta,H}$. This parameter controls the tradeoff between exploration and exploitation. Higher values of α promote more exploration, whereas lower values of α encourage more exploitation in the Soft Actor-Critic algorithm. For more details how to this parameter is used in Soft-Actor-Critic please have look into Section 3.3.3

For simplification we will now refer to the entropy regularized value function $v_{\pi_\theta,H}$ as v_{π_θ} and action-value-function $q_{\pi_\theta,H}$ as q_{π_θ}.

This change also influences the original reinforcement learning problem as stated in Equation (3.10) and converts it into Equation (3.18). Due to the stated relation between regularized value-function and regularized action-value-function as in Equation (3.17) we can make the optimization problem independent from the value-function and get Equation (3.19).

$$
\begin{aligned}
\pi_\theta^* &= \arg\max_{\pi_\theta} v_{\pi_\theta,H} \\
&\overset{(3.10)}{=} \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t \left(R_t + \alpha H(\pi_\theta)\right) \mid S_t = s\right] \qquad (3.18) \\
&\overset{(3.17)}{=} \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}\left[q_{\pi_\theta}(s,a)\right] + \alpha H(\pi_\theta) \\
&\overset{(3.12)}{=} \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}\left[q_{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s)\right] \qquad (3.19)
\end{aligned}
$$

(TODO: was ist allgemeiner .... bis inf oder bis T)

3.3.3 Algorithm Architecture

In contrast to other actor-critic algorithms like: XXX, or XXX, **(TODO: add examples)** SAC learns additional to a parameterized policy π_θ, two critics in the form of two parametrized q functions Q_{ϕ_0} and Q_{ϕ_1} as well as their corresponding target functions $Q_{\phi_{\text{target},1}}$ and $Q_{\phi_{\text{target},1}}$. The usage of target action-value functions enhances the accuracy of the estimates and contributes to a more stable learning process. **(TODO: proof)**

Further SAC utilizes a stochastic policy, which means that the policy outputs a parameterized probability distribution over actions instead of directly selecting deterministic actions and optionally adding noise on top. This stochastic nature allows SAC to

capture the exploration-exploitation trade-off directly within the policy. The training process in SAC is based on the maximum entropy objective. Its goal is to find a policy that maximizes the cumulative reward Equation (3.1) and the maximum entropy Equation (3.12). Fortunately we have derived an entropy regularized value-function and action-value-function in Section 3.3.2 and are able to use those definition to derive loss functions for actor and critic based on the maximum entropy objective.

Because we will have a closer look into actual implementation, we now refer to the action-value-function as their parameterized approximations Q_{π_i}.

To **train the parameterized action-value-functions** the objective function (3.20) is defined as the expected squared difference between the action-state-value $Q_{\phi_i}(s, a)$ and its temporal difference target $y(r, s', d)$. Equation (3.20) leads directly to the update equation (3.26) as its derived empirical counterpart.

$$L_Q(\phi_i, \mathcal{D}) = \mathbb{E}_{\mathcal{D}} \left[\left(Q_{\phi_i}(s, a) - y(r, s', d) \right)^2 \right] \tag{3.20}$$

The temporal difference target target y is defined as in Equation (3.21) with $\hat{a}' \sim \pi\theta(\cdot|s')$. **(TODO: drop the buzz word: td target)**. To stabilize the target signal SAC applies the clipped double-Q trick, where it selects the minimum q-value between the two target q-functions $Q_{\phi_{\text{target},i}}$.

$$y(r, s', d) = r + \gamma(1 - d) \left(\min_{j=0,1} Q_{\phi_{\text{target},i}}(s', \hat{a}') - \alpha \log \pi\theta(\hat{a}'|s') \right) \tag{3.21}$$

(TODO: where exactly is the target coming from?)

This function is employed in the SAC algorithm in Equation (3.25) but with the small difference that all arguments are sampled from a minibatch $\mathcal{B}$.

To **train the policy** we are required to use the reparameterization trick to be able to differenciate the parameterized policy π_θ. **(TODO: cite: https://sassafras13.github.io/ReparamT**

This process transforms the action $\hat{a}$ sampled from the policy into the function as specified in Equation (3.22).

$$\hat{a}_\theta(s,\epsilon) = \tanh(\mu(\pi_\theta(\cdot|s)) + \sigma(\pi_\theta(\cdot|s)) \bigodot \epsilon),\ \epsilon \sim \mathcal{N}(0,I) \tag{3.22}$$

Within the reparameterization of an action we also bound it into a finite range of $(-1,1)$ this has the advantage , ... **(TODO: fill)**. Since we squash the actions with tanh we also have to adapt the log-likelihood $\log \pi_\theta(a|s)$ of an action into:

$$\log \pi_\theta(a|s) = \log \pi_\theta(a|s) - \sum_{i=1}^{|\mathcal{A}|} \log(1 - \tanh(\hat{a}_\theta(s,\epsilon))),\ \epsilon \sim \mathcal{N}(0,I)$$

For more details please refer to **(TODO: cite: https://arxiv.org/pdf/1812.05905.pdf)**.

To compute the policy loss, as in training the action-value-function approximation, the crucial step involves replacing q_{π_θ} with one of our function approximators Q_{ϕ_i}. SAC utilizes the minimum of the two q_{π_θ} approximators $\min_{i=0,1} Q_{\phi_i}$. Consequently, the policy is optimized based on this minimum action-state-value approximation and therefor make only more conservative estimates.

$$L_\pi(\theta,\mathcal{D}) = -\mathbb{E}_{\mathcal{D},\mathcal{N}(0,I)}\left[\min_{i=0,1} Q_{\phi_i}(s,\hat{a}_\theta(s,\epsilon) - \alpha \log \pi_\theta(\hat{a}_\theta(s,\epsilon)|s))\right] \tag{3.23}$$

Note that we are taking the negative expectation because we want to maximize the expected return and the entropy by using SGD as in Equation (3.27) in Algorithm 2. The different notation can be explained a $\{s,a,r,s',d\}_{\mathcal{B},k}$ stand for the kth element from the minibatch $\mathcal{B}$..

Now lets turn to the entropy regularization parameter α. In general there are two types of Soft-Actor-Critic implementations. On proposed by XXXX **(TODO: fill)** treads α as constant parameter. The optimal α parameter, leading to the most

stable and rewarding learning, may vary across different environments, necessitating thoughtful tuning for optimal performance. The second approach how to treat α, proposed by XXX **(TODO: fill)**, adjusts α constantly during the training process. The optimization criterion is stated as:

$$L_\alpha = \mathop{\mathbb{E}}_{a_t \sim \pi_\theta} \left[-\alpha \log \pi_(a_t|s_t) - \alpha \bar{H} \right] \tag{3.24}$$

Equation (3.24) resembles an objective for dual gradient descent because we are trying to minimize α but also the expected difference between the policy entropy and a target entropy $\bar{H}$ as a hyperparameter. In practice this can be translated into a positive gradient if the expected difference is positive so the agent is less exploring as should be and into a negative gradient if the difference is negative so the agent is to focused on exploring. Selecting a target entropy is not as delicate as selecting a fixed α because you are able to read from training results if your environment requires a more or less greedy policy. **(TODO: dual gradient descent chapter.)**

(TODO: rework equations from pseudo code)

3.3.4 Advantages

SAC offers several advantages over other actor-critic algorithms. Firstly, the entropy regularization leads to improved exploration, enabling the agent to efficiently explore its environment and discover optimal or near-optimal solutions **(TODO: paper)**. Secondly, by encouraging stochastic policies, SAC provides more robust and adaptive policies that can handle uncertainties and variations in the environment.

Moreover, the SAC algorithm exhibits enhanced sample efficiency, meaning that it requires fewer interactions with the environment to learn effective policies. The utilization of two critics Q_{ϕ_0} and Q_{ϕ_1} further contributes to a more stable learning

Algorithm 1 Soft Actor Critic

Input: initial policy parameters θ, Q-function parameters ϕ_0, ϕ_1, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\phi_{\text{target},0} \leftarrow \phi_0$, $\phi_{\text{target},1} \leftarrow \phi_1$

for i in number of epochs **do**

$s \leftarrow$ reset environment

for t number of timesteps **do**

$a \sim \pi_\theta(\cdot|s)$

s', r, $d \leftarrow$ execute a in environment

Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

if d is true **then**

break and reset environment

end if

$s \leftarrow s'$

end for

if $|\mathcal{D}| >$ minimal buffer size **then**

for number of train iterations **do**

sample minibatch: $(s_\mathcal{B}, a_\mathcal{B}, r_\mathcal{B}, s'_\mathcal{B}, d_\mathcal{B}) = \mathcal{B} \leftarrow \mathcal{D}$

compute td target y with $\tilde{a}'_\mathcal{B} \sim \pi_\theta(\cdot|s'_\mathcal{B})$

$$y(r_\mathcal{B}, s'_\mathcal{B}, d_\mathcal{B}) = r + \gamma \cdot d \cdot \left(\min_{i \in \{0,1\}} \left(Q_{\phi_{\text{target},i}}(s'_\mathcal{B}, \tilde{a}'_\mathcal{B}) \right) - \alpha \cdot \log \left(\pi_\theta(\tilde{a}'_\mathcal{B}|s'_\mathcal{B}) \right) \right) \tag{3.25}$$

Update Q-functions for parameters ϕ_i $i \in \{0, 1\}$ using:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{k \in |\mathcal{B}|} \mathcal{L}_\beta \left(y(r_{\mathcal{B},k}, s'_{\mathcal{B},k}, d_{\mathcal{B},k}), Q_{\phi_i}(s_{\mathcal{B},k}, a_{\mathcal{B},k}) \right) \tag{3.26}$$

Update policy with $\tilde{a}_\mathcal{B} \sim \pi_\theta(\cdot|s_\mathcal{B})$ using:

$$\nabla_\theta \frac{1}{|\mathcal{B}|} \sum_{k \in |\mathcal{B}|} \min_{i \in \{0,1\}} Q_{\phi_i}(s_{\mathcal{B},k}, \tilde{a}_{\mathcal{B},k}) - \alpha \cdot \log \left(\pi_\theta(\tilde{a}_{\mathcal{B},k}|s_{\mathcal{B},k}) \right) \tag{3.27}$$

Update α with target entropy H_{target} and $\tilde{a}_\mathcal{B} \sim \pi_\theta(\cdot|s_\mathcal{B})$ using:

$$\nabla_\alpha - \frac{\alpha}{|\mathcal{B}|} \sum_{k \in |\mathcal{B}|} \left(\log \left(\pi_\theta(a'_{\mathcal{B},k}|s_{\mathcal{B},k}) \right) \ - H_{\text{target}} \right)$$

Update target networks $\phi_{\text{target},i}$ $i \in \{0, 1\}$ with ϕ_i $i \in \{0, 1\}$ using:

$$\phi_{\text{target},i} \leftarrow \rho \phi_{\text{target},i} + (1 - \rho) \phi_i$$

end for

end if

end for

process and can mitigate the issues of overestimation bias, leading to more accurate value estimates. **(TODO: paper)**

Applications

In robotics, the SAC algorithm has been used to control robots in tasks such as grasping objects, locomotion, and manipulation. The SAC algorithm's ability to handle continuous action spaces makes it a suitable choice for robotic control, where fine-grained control is often required. **(TODO: look for papers to support this claim)**

In game playing, the SAC algorithm has been used to train agents to play video games such as Atari and Super Mario Bros. **(TODO: paper)** The SAC algorithm's ability to balance exploration and exploitation makes it effective in game playing, where agents must learn to navigate complex environments and respond to changing conditions.

3.4 Neural Networks

Neural networks are a class of artificial intelligence algorithms inspired by the structure and functioning of the human brain. They consist of interconnected layers of artificial neurons that process and transform data. Neural networks are crucial in modern machine learning and deep learning applications due to their ability to learn complex patterns and representations from data, enabling them to solve a wide range of problems effectively.

In this chapter, we will delve into neural networks, looking at their evolution, architecture, training process and diverse applications.

3.4.1 The Rise of Deep Learning

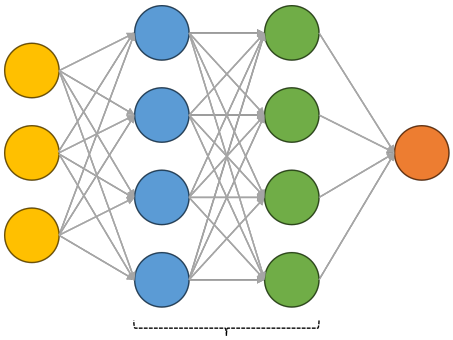
Neural networks were first invented in the 1940s and 1950s as a computational model inspired by the interconnected structure and functioning of the human brain, but their practical development was hindered by limitations in computing power and the lack of large datasets. The resurgence of interest in neural networks in the 2000s can be attributed to two major factors: the development of new algorithms and the availability of large datasets. Before the 2000s, neural networks faced limitations in training and optimization, hindering their effectiveness. However, new algorithms, such as the backpropagation algorithm and variants like stochastic gradient descent, emerged, making it feasible to train deeper networks efficiently. Additionally, advancements in computing power and the availability of vast datasets facilitated by the internet like image net **(TODO: cite)** allowed neural networks to leverage big data for improved learning and performance.

Nowadays neural networks are employed over a vast variety of tasks like image recognition, speech recognition, natural language processing or robotics.

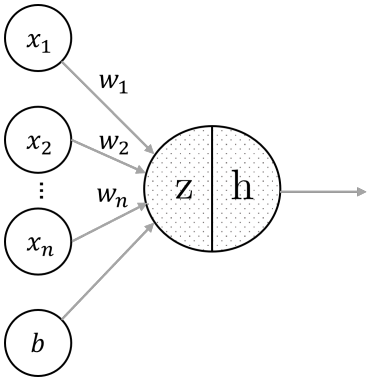
3.4.2 Neural Network Architecture

Neural networks are a type of machine learning model inspired by the structure and function by the cell type of neurons. It consists like their biological model, of interconnected layers of individual units, called neurons.

The basic architecture of a neural network as in Figure 3.4a, includes an input layer (yellow), one or more hidden layers (blue and green), and an output layer (red). The input layer receives the input data, which is then passed through the hidden layers before producing the output. Because the flow of information is only in the forward direction we call this basic architecture a feed forward neural network. The



(a) schematic drawing of a feed forward neural network



(b) schematic drawing of a single neuron

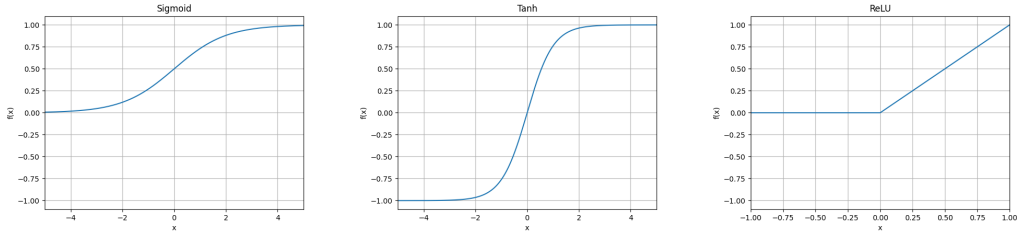
Figure 3.4: The figure shows the schematic drawing of a feed forward neural network and a neuron

number of nodes in each layer and the connections between them are configured by the architecture of a network.

If we look at each individual neuron as in Figure 3.4b, we can observe that each neuron is designed in the same way. First it calculates a weighted sum z of its inputs x, the corresponding weights w and bias b, before passing it into an activation function h and finally passing the information to the next neuron. The activation function is designed to introduce nonlinearity into the model, allowing it to capture complex relationships between variables. Without a nonlinear activation function the whole network would be a single linear combination of its inputs and weights. Common activation functions include the sigmoid function, the hyperbolic tangent function, and the rectified linear unit (ReLU) function as in Figure 3.5

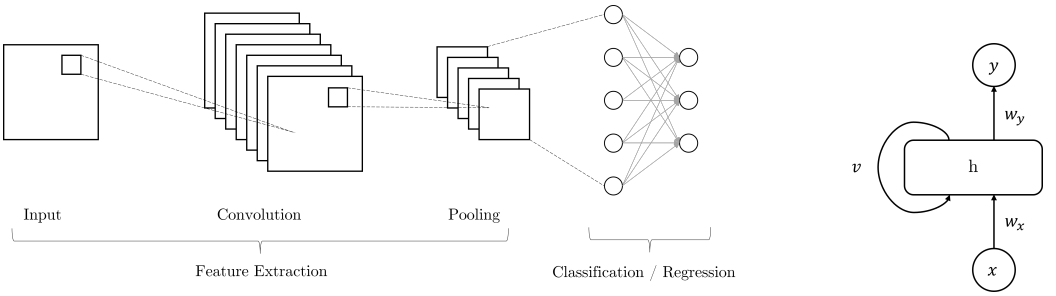
There are several types of neural network architectures but most of them are based on following types:

- **Feedforward networks** as in Figure 3.4a are the simplest type of neural network, consisting of a series of layers that process information in a single direction.



(a) Sigmoid **(b)** hyperbolic tangent function **(c)** rectified linear unit

Figure 3.5: common activation function for neural networks.



(a) schematic drawing of convolutional neural network

(b) schematic drawing of a recurrent neural network

Figure 3.6: architecture of more advanced neural networks like the convolutional and the recurrent neural network

- **Convolutional networks** as in Figure 3.6a are designed for image processing tasks and use convolutional layers to identify patterns and features within images.

- **Recurrent networks** as in Figure 3.6ballow information to be passed between nodes in a cyclical manner, making them suitable for processing sequential data. The development of Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures further improved RNNs' ability to model long-range dependencies.

3.4.3 Train Neural Networks

Neural networks are trained using techniques such as backpropagation and stochastic gradient descent as presented in Algorithm 2. Backpropagation is an algorithm for calculating the gradient $\hat{g}$ of an optimization criterion with respect to the weights of the network. The gradient can then be used to update the weights and therefor improve the performance of the model with respect to the opimization function. A we can see in Algorithm 2 stochastic gradient descent minimizes the error presented by the opimization criterion by iteratively adjusting the weights based on randomly selected subsets of the training data.

Algorithm 2 Stochastic gradient descent

Input: Learning rate schedule: $\epsilon_1, \epsilon_2, \ldots$. Initial parameter θ
$k \leftarrow 1$
while stopping criterion not met **do**
Sample a minibatch of m examples from training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Compute gradient estimate:
$\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
Apply update:
$\theta \leftarrow \theta - \epsilon_k \hat{g}$
$k \leftarrow k + 1$
end while

Selecting a random subset of training data has a couple of advantages.

- **Computational Efficiency**: In one epoch we are computing the gradient only on a small subset. We do not have to pass the complete dataset through the network to get a sufficient gradient.

- **Improved Generalization**: Because each minibatch contains a different composition of data points from the dataset the gradient signal is also varying. These variations are improving the generalization of the model with respect to unseen data.

- **Avoiding Local Minima**: One of the risks during the training-process of a neural network are local minima. By introducing randomness through mini-batch sampling, SGD can escape local minima more easily and continue to explore the parameter space, increasing the chances of finding a global minimum.

- **Parallel Processing**: Using mini-batches allows parallel processing during training.

(TODO: find sources)

3.4.4 Applications of Neural Networks

Neural networks are widely used in real-world applications like in computer vision, marketing or medical applications. Particularly in combination with reinforcement learning they are successfully used in robotics for object recognition and manipulation, in gaming for complex decision-making, and in finance for predicting stock prices.

3.5 Variational Autoencoder

Variational autoencoders (VAEs) and conditional variational autoencoders (CVAEs) are types of deep generative models that are used for unsupervised learning. Unsupervised learning is a type of machine learning where the model is not given labeled data for training. Instead, the model is tasked with finding patterns or structure in the data on its own. The goal of unsupervised learning is often to find hidden relationships or groupings within the data that can be used for further analysis or decision-making VAEs and CVAEs are important because they can learn to generate realistic and diverse samples from complex high-dimensional data distributions, such as images or audio.

3.5.1 Autoencoders

Lets start with a Autoencoder. An Autoencoder $f = d \circ e$ consists of two parameterized function approximators in series, one encoder e_ϕ and one decoder d_ψ. The encoder maps the high dimensional input $x \in \mathbb{X}$ into a lower dimensional latent space $z \in \mathbb{Z}$, $z = e(x)$. The latent vector z is typically a lower dimensional representation of the input x. In the second stage we map the latent vector z back into the high dimensional features space X, $d : \mathbb{Z} \to \mathbb{X}$. This should optimally reconstruct the given input $\hat{x} = d(z) = d(e(x))$.

Because Autoenders are designed to learn a compact determin representation in the latent space it follows that similar input values x should correspond to similar latent vectors z and similar latent vectors z correspond to similar outputs $\hat{x}$.

Traditional Autoenders have prooven to be successful on a variaty of task like: ..., but their simple and effective design comes along with some limitations:

- Overfitting: Because of the deterministic mapping into the latent space and from the latent space back to the feature space traditional Autoencoders are

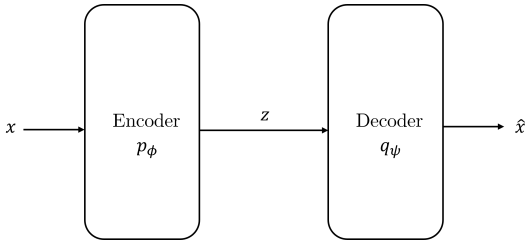


Figure 3.7: schematic drawing of a Autoencoder

prone to suffer from overfitting and can lack of regularization which could lead to poor performance on unseen data.

- Incomplete or noisy data: Due to the focus on only reconstructing the input, Autoencoders are not well suited for incomplete or noisy data. One reason could be the lack of disentanglement in the latent space, which means that different dimensions correspond to different underlying features in the input data distribution.

- Probabilistic Interpretation: Another drawback of the deterministic mapping approach is the lack of probabilistic interpretation capabilities for uncertainty in the learned representation and generated outputs.

3.5.2 Variational Autoencoders

(TODO: ELBO: https://probml.github.io/pml-book/book2.html 779)

A Variational Autoencoder is similar to an Autoencoder but also with some key changes. Similar are the basic setup as a generative unsupervised learning model and the underlying encoder decoder structure. The key difference lays in the in the latent space between the encoder and decoder. Instead of having a fixed deterministic latent space VAEs operate on a probabilistic latent space.

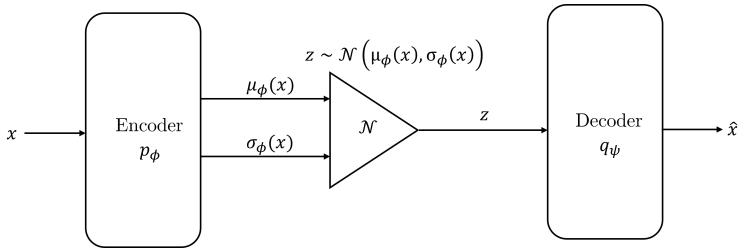


Figure 3.8: schematic drawing of a Variational Autoencoder

The encoder network denoted as a approximated parameterized posterior distribution $q_\phi(z|x)$, is a conditional probability distribution in Bayesian inference with the probability of the latent variable $p(z)$ (the prior) and the evidence $p(x)$:

$$q(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{3.28}$$

$$p(x) = \int p(x|z)p(z)dz \tag{3.29}$$

In the context of a Variational Autoencoder, calculating $q(z|x)$ is not possible. As we can see in Equation (3.29) accessing the evidence as the probability of the observed data is not possible because we have to integrate over all possible values of the latent variable z. Therefor we approximate the posterior distribution with a parameterized conditional distribution $q_\phi(z|x)$. $q_\phi(z|x)$ as known as the encoder network, parameterized with ϕ, hereby maps input data x to latent parameters like mean and variance for a gaussian distribution, of the latent distribution with random variable z.

On the other hand, the decoder network takes samples from the latent space with latent variables z and reconstructs the data from these samples, generating a parameterized probabilistic distribution over the data given the latent variables $p_\psi(x|z)$.

As mentioned before the objective in training a Variational Autoencoder is to find

parameters ϕ and ψ and matches best the true posterior distribution. In order to do so we can use the Evidence Lower Bound as the objective function. We go a bit deeper into the fundamentals of the Evidence Lower bound in Section 3.5.4.
During training, the VAE aims to maximize the ELBO by iteratively adjusting its parameters ϕ andψ using backpropagation. The encoder network maps the input data to a distribution over latent variables, while the decoder network reconstructs the data from the sampled latent variables. The reparameterization trick similar to SAC in Section 3.3 is employed to ensure differentiability during backpropagation through the stochastic sampling process. **(TODO: reparaphrase)**

3.5.3 Conditional Variational Autoencoders

A Conditional Variational Autoencoder is an extension of the VAE architecture that takes into account conditional information c. As we can see in Figure 3.9In a Conditional Variational Autoencoder, both the encoder and decoder are modified to accept an additional input that represents the conditional information.

The role of the conditional input in the encoder is to help the model capture the conditional dependencies between the input data and the class label. So the conditional information could be for instance a label encoding or a set of labels that describe the class or category to which the input belongs. The encoder must learn to map both

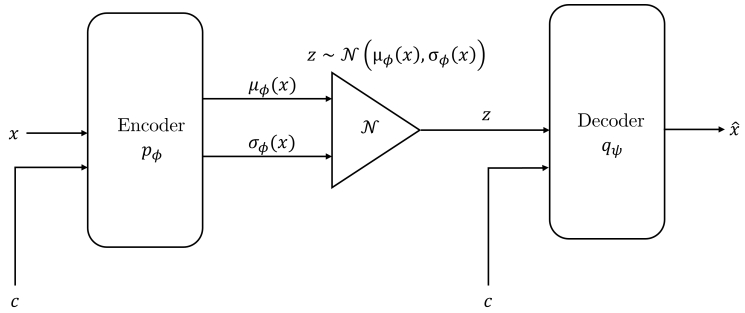


Figure 3.9: schematic drawing of a conditional Variational Autoencoder

the input data and the conditional input to a meaningful latent space representation that captures the relevant information for the generation process. In the decoder, the conditional input is used to guide the generation process and ensure that the generated samples are representative of the specified class.

Similar to Variational Autoencoders the objective function to train the encoder and decoder is the Evidence Lower Bound. But there have to be a couple of changes to make the individual ELBO components suitable to take the conditional information c. The final objective can be observed in Equation (3.33).

3.5.4 Evidence Lower Bound

The Evidence Lower Bound (ELBO) is a fundamental concept in the training of Variational Autoencoders (VAEs) It is derived from the principle of variational inference and represents a lower bound on the log-likelihood of the data given the model's parameters. Maximizing the ELBO is equivalent to minimizing the Kullback-Leibler (KL) divergence between the true posterior distribution $q(z|x)$ over latent variables z and the approximated posterior distribution $q_\phi(z|x)$ used in the VAE.

$$L_{\text{ELBO}}(x, z) = \mathbb{E}_{z\sim q_\phi(\cdot|x)} \left[\log p\psi(x|z)\right] - \text{KL}\left(q_\phi(z|x) \| p(z)\right) \tag{3.30}$$

Further I will discuss the individual components of the Evidence lower bound

- **Posterior Distribution**: In general the posterior distribution $q(z|x)$ is considered as the probability of the latent variable z given the observed data x. Specific for Variational Autoencoder it can be computed from Bayesian Inference as in Equation (3.28) which is as mentioned in Section 3.5.2 not feasible is therefor approximated.

- **Likelihood**: The likelihood function $p(x|z)$ or conditional likelihood in the context of probabilistic models is the probability distribution of the observed data x given the latent variables z. Dependent on the observed data it is possible to chose from a set of different probability distributions. Examples are a multivariate gaussian distribution for continuous data, a Bernoulli distribution for binary data or categorical distribution for discrete data with multiple categories.

- **Prior Distribution**: The prior distribution $p(z)$ is a probabilistic distribution that represents the initial belief or assumption about the latent variables z in Bayesian inference. Additional $p(z)$ serves in Variational Autoencoder as a regularizer during training as it encourages the VAE to learn meaningful and smooth latent representations in the latent space. Most popular choice for the prior distribution is the standard normal distribution $(N)(0, I)$.

- **Evidence**: p(x) represents the evidence, also known as the marginal likelihood or model evidence, in the context of Bayesian statistics. It is the probability of the observed data x given a particular statistical model. The evidence serves in Equation (3.29) as a normalization constant, ensuring that the posterior distribution $q(z|x)$ over model parameters integrates to 1. It quantifies how well the model, with its specific set of parameters, explains or fits the observed data.

- **Reconstrion Loss**:

$$\mathbb{E}_{z\sim q(\cdot|x)}\left[\log p(x|z)\right] \approx \mathbb{E}_{z\sim q_\phi(\cdot|x)}\left[\log p_\psi(x|z)\right] \tag{3.31}$$

  Therm (3.31) measures the similarity between the reconstructed data and the original input. Described in words it is the expected log-likelihood of the data given the latent variables, which is computed by sampling from the

approximating distribution $q_\phi(z|x)$ and evaluating the likelihood $p_\psi(x|z)$ using the decoder network.

- **KL Divergence Loss**:

$$\mathrm{KL}\left(q(z|x)\|p(z)\right) \approx \mathrm{KL}\left(q_\phi(z|x)\|p(z)\right) \tag{3.32}$$

This term quantifies the difference between the approximated parameterized posterior distribution $q_\psi(z|x)$ over latent variables z and a chosen prior distribution $p(z)$. As mentioned before it is a popular choice to use a standard normal distribution $p(z) = (N)(0, I)$ as the prior distribution. The KL divergence encourages the latent variables to be close to the prior distribution, promoting regularization and preventing overfitting.

(TODO: Explain prior and posterior distributions)

The basic notation of the Evidence Lower Bound for Variational Autoencoders as in Equation (3.30) can be easily extented for an Conditional Variational Autoencoder:

$$L_{\mathrm{ELBO}}(x, z, c) = \mathbb{E}_{z\sim q_\phi(\cdot|x,c)}\left[\log p\psi(x|z, c)\right] - \mathrm{KL}\left(q_\phi(z|x, c)\|p(z|c)\right) \tag{3.33}$$

(TODO: Describe also the components of the CVAE ELBO)

By optimizing the ELBO, VAEs effectively learn to approximate the true posterior distribution and generate meaningful latent representations of the data, enabling various tasks such as data generation, interpolation, and denoising within a Bayesian framework.

3.5.5 VAEs and CVAEs in Practice

VAEs and CVAEs have been successfully applied to a wide range of real-world applications. In image generation, VAEs have been used to generate novel images of

faces, objects, and scenes. Similarly, CVAEs have been used for conditional image generation, allowing for the generation of images based on specific attributes or classes. In text generation, VAEs have been used to generate natural language sentences and paragraphs.

VAEs and CVAEs can also be used for data compression and denoising. By learning a compressed representation of the input data, VAEs and CVAEs can reduce the dimensionality of the input space while preserving important features. Similarly, by learning to reconstruct the original input from noisy or corrupted data, VAEs and CVAEs can be used for denoising and data restoration.

One of the advantages of VAEs and CVAEs is their ability to learn a continuous latent representation of the input data. This allows for easy manipulation and exploration of the latent space, enabling applications such as image editing and style transfer **(TODO: cite paper)**.

However, VAEs and CVAEs have some limitations. The generated samples may not be as sharp or detailed as those produced by other generative models such as GANs **(TODO: cite)**. Additionally, the trade-off between the reconstruction loss and the KL divergence term can be difficult to balance, potentially leading to overfitting or underfitting **(TODO: paper)**. Nonetheless, VAEs and CVAEs remain a popular and powerful tool for generative modeling and data compression.

3.6 Kinematics

Kinematics is the study of motion, specifically the description and analysis of the position, velocity, and acceleration of objects or systems without considering the forces causing the motion. It focuses on understanding the spatial relationships and geometrical aspects of moving objects. The following sections provide an insight into

forward and inverse kinematics for kinematic chains. Those two concepts are often used in robotics, animation, virtual reality or even protein folding.

3.6.1 Forward kinematics

Forward kinematics is a concept from robotics that involves determining the position and orientation of an end-effector, like a gripper of a robot arm, based on the joint angles and geometric parameters, like segment length, of the system. It provides a mathematical model for mapping the joint angles to the end-effector position in order to understand the overall configuration and motion of the robot. Equation (3) describes a forward kinematics implementation.

Algorithm 3 Forward Kinematics

Input: Current joint angles q, Segment Length l.
Define origin position in 2D space: $p \leftarrow (0, 0)$
for each i in $[0, \ldots, N-1]$ **do**
    Update position
    $p_0 \leftarrow p_0 + \cos(q_i) * l_i$
    $p_1 \leftarrow p_1 + \sin(q_i) * l_i$
end for

Throughout this thesis this function is used with a constant segment length $l = \{1\}^N$ therefor it is referred to as in Equation (3.34) which maps an angle configuration q into the end-effector position in 2D space.

$$fk : \mathbb{R}^N \rightarrow \mathbb{R}^2 \tag{3.34}$$

3.6.2 Inverse kinematics

Inverse kinematics (IK) is a fundamental problem in robotics, animation or virtual reality that involves finding a required joint angle configuration or positions to reach a desired end-effector position and optionally a desired orientation. It plays a crucial

role in controlling the motion and manipulation of robotic systems, enabling them to interact with the environment and perform complex tasks. In this section you will find a brief summary of existing approaches, a deeper explanation of the Cyclic Coordinate Descent algorithm and the description of the used inverse kinematics RL environment.

Existing Approaches to Solve Inverse Kinematics

Numerous approaches have been proposed to solve the inverse kinematics problem. These approaches can be broadly categorized into:

- analytical methods: rely on geometric methods to derive closed form solutions.
- numerical methods: iteratively approximate the joint angles that satisfy the desired end-effector position and orientation.
- heuristic methods: iteratively propagate positions along the kinematic chain to converge on the desired end-effector position.
- sampling-based methods: randomized search strategy to explore the joint space and find feasible solutions to the inverse kinematics problem

Cyclic Coordinate Descent

Cyclic Coordinate Descent (CCD) is a popular numerical method for solving inverse kinematics. It is an iterative algorithm that adjusts the joint angles of a robotic system one at a time, from a base joint to the end-effector, in order to align the end-effector with the desired target position.

The algorithm works by iteratively updating the joint angles based on the discrepancy between the current and desired end-effector positions (p_{target}). At each iteration,

Algorithm 4 Cyclic Coordinate Descent Pseudo Code

Input: Current joint angles q, Desired end-effector position p_{target}.
while unitl convergence **do**
  for each i in $[N-1, \ldots, 0]$ **do**
    Calculate vector from current joint position to end-effector position:
    $v_{\text{current}} \leftarrow p_{N-1} - p_i$
    Calculate vector from current joint position to target position:
    $v_{\text{target}} \leftarrow p_{\text{target}} - p_i$
    Calculate the necessary rotation to align v_{current} with v_{target}:
    $\delta q_i \leftarrow \text{angle\_between}(v_{\text{current}}, v_{\text{target}})$
    Update joint angle:
    $q_i \leftarrow q_i + \delta q_i$
  end for
end while

CCD focuses on a single joint and adjusts its angle to minimize the positional error. By sequentially updating the joint angles in a cyclic manner, CCD aims to converge towards a solution that satisfies the desired end-effector position.

As illustrated in Algorithm 4 and Figure 3.10 in each iteration, CCD calculates the vector from the current joint position p_i with p_i as the position of the ith joint with $i \in \{0, \ldots, N-1\}$, to the end-effector position (v_{current}) and the vector from the current joint position to the target position (v_{target}). By finding the rotation necessary to align v_{current} with v_{target}, represented as $\delta\phi$, the algorithm updates the joint angle accordingly. This process is repeated for each joint in the kinematic chain until convergence is achieved.

This afore mentioned strategy can be reviewed in Figure 3.11. Here we can see that the strategy appears to move closer and closer to the origin before heading towards the target position in blue.

That CCD needs different amounts of while iterations depending on start and end position can be observed in Figure 3.12. The heatmap plots the different amounts of while iterations needed to get from a constant start position at $p = [N, 0]$ with all angles equal 0, to a desired target position. Interesting to see are the patterns

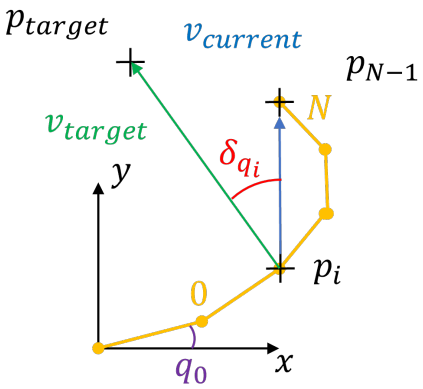
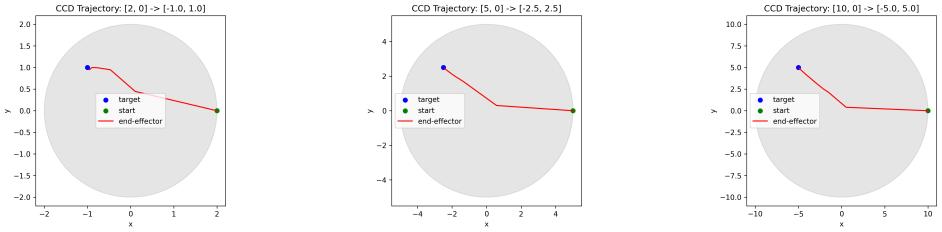
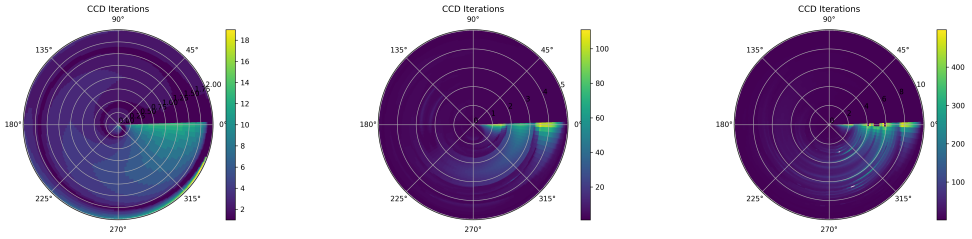


Figure 3.10: CCD geometry: Shown is the geometry of one step in Algorithm 4. p_{target} is the target position. p_i is the 2D position of each joint. Not that we start with the first joint outside the origin since the base position is fixed to the origin.



(a) $N = 2$ with 5 iterations **(b)** $N = 5$ with 11 iterations **(c)** $N = 10$ with 64 iterations

Figure 3.11: Computed trajectory of CCD. The two scattered lines resemble the start end effector positon (green) and the target position (blue). The red line is the trajectory of the end effector after each while iterations as in Algorithm 4.



(a) $N = 2$ **(b)** $N = 5$ **(c)** $N = 10$

Figure 3.12: Heatmaps for while iterations needed to reach from a start position at $[N, 0]$ to target positions at $p_{(i,j)} = [\cos(\omega_i), \sin(\omega_i)] * \mathfrak{R}_j, \omega = [0, \ldots, 2\pi], \mathfrak{R} = [0, \ldots, N]$

emerging in those heatmaps where target positions counter clockwise of the start positions are faster to solve than target positions clockwise to the start positions.

4 Methodology

covers: research idea rl-environment experiments on VAE, Supervised and merged pipeline

4.1 Research Idea

The primary research idea of my thesis is to pursue a transformation from the RL environment action space $\mathcal{A}$ into a lower-dimensional latent action space $\mathcal{A}_L$. This latent action space will align with the latent space between the encoder and decoder of a VAE model or the feature space of a feed-forward-neural network. By doing so, we aim to enable RL agents to effectively explore and learn within a more compact and smoothened action representation. An schematic drawing can be revisited at Figure 4.1

We propose two potential approaches for achieving this reduction in dimensionality: VAEs and supervised models.

In the VAE approach, a conditional or unconditional generative model is employed to learn a latent representation of the action space. By training the VAE on actions from an expert or on state target combinations to emphasis a solution which is independent from an expert, we aim to capture the underlying structure and patterns within the action space from the RL environment. This latent representation can potentially

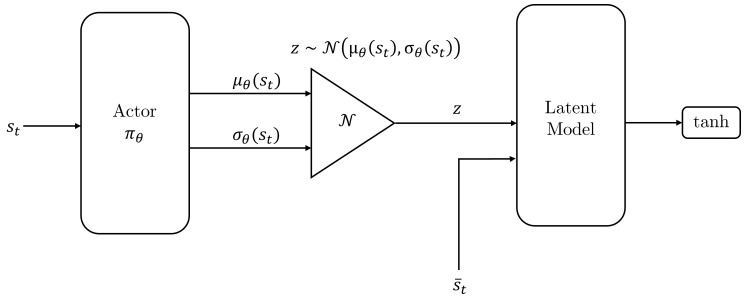


Figure 4.1: Schematic drawing of how the models would be chained together. The laten model will be replaced by either the decoder of a VAE, a CVAE or a supervised feed forward network. Note that we adapt the additional information $\bar{s_t}$ by the needs of the individual models.

offer a more concise and informative representation of the actions, encourage more efficient learning and exploration for RL agents.

Alternatively, the supervised model approach involves training a supervised learning model, such as a neural network, to directly transform the a defined lower dimensional latent action into the high-dimensional action space. This transformation is learned based on labeled examples of actions and their corresponding latent representations. In the conducted experiments the lower dimensional-action space is just the desired action outcome. By leveraging supervised learning techniques, we aim to find a mapping between state information and desired action outcome to the RL environment action space, allowing RL agent to operate effectively on a higher level within the reduced-dimensional latent action space.

4.2 RL Environment

To apply RL in general you need an environment the agent can send actions to and receives feedback as discussed in Section 3.1. As previously introduced the key problem we are targeting is inverse kinematics of a robot arm. This problem turn out to be suitable because:

- the action space is scalable by simply adding additional joints to the robot arm
- it can be simplified into a 2D space with a fast and reliable implementation
- it is expandable. You are always able to extend the environment with additional constrains like objects the robot has to navigate around or joint angle constrains.

In this section I'm going to present the a novel RL-Environment for bench-marking the performance of different algorithms to solve inverse kinematics for a robot arm with N many joints and subsequently N many segments with lengths $l \in \mathbb{R}^N_{>0}$ in 2D space. For simplicity reasons l is constant with $l = \{1\}^N$.

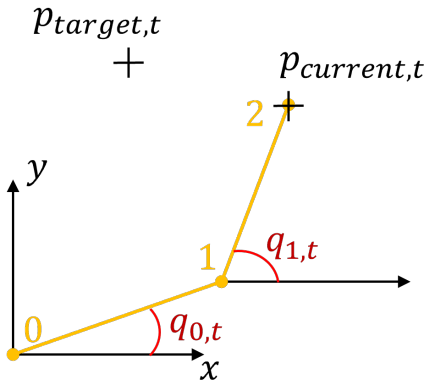


Figure 4.2: Schematic drawing of the individual state space components.

4.2.1 State Space

The state space $\mathcal{S} \subset \mathbb{R}^{4+N}$ for this environment consists of three main building blocks.

- **goal information** $p_{\text{target},t} \in \mathbb{R}^2$: This vector provides information where to move the end-effector. This position is lies always in for the robot arm, reachable distance. Mathematical speaking: $||p_{\text{target}}||_2 \leq \sum_{i=1}^{N} l_i$

- **state position** $p_{\text{current},t} \in \mathbb{R}^2$: This vector contains the current end-effector position in 2D space and should help the agent to understand where the end-effector is placed and how an action has influenced the current end-effector position. Because the current position is attached to the robot arm: $||p_{\text{current}}||_2 \leq \sum l$.

- **joint angles** $q_t \in [0, 2\pi^N)$: This vector contains information about the joint angle configuration at time t. Note that each joint angles describes the delta angle between the joint direction and a horizontal line from origin in positive x direction.

A state $s_t \in \mathcal{S}$ at time t has for all t the same composition

$$s_t = (p_{\text{target},t}, p_{\text{current},t}, q_t) \tag{4.1}$$

To refer to individual parts from index i to j of a state with: $s_{t,(i,j)}$. Therefor we can extract the individual parts with:

- $p_{\text{target},t} = s_{t,(0,1)}$

- $p_{\text{current},t} = s_{t,(2,3)}$

- $q_t = s_{t,(4,N+4)}$

4.2.2 Action Space

The action space $\mathcal{A} \subseteq \mathbb{R}^N$ for this particular environment is continuous and contains all possible joint angle configurations for a robot arm with N joints.

A generated action $\hat{a}$ from the agent is sent to the environment. Inside the environment the incoming action is added on top of the current state angles q_t. To ensure the

constrains of $q_{t+1} \in [0, 2\pi)^N$ we take the signed remainder of a division by 2π to write into q_{t+1}:

$$q_{t+1} = (q_t + \hat{a}) \ \% \ 2\pi$$

Subsequently after updating the state angles the current end-effector position get updated by a forward kinematics call on q_{t+1}:

$$p_{\text{current},t} = \text{FK}(q_{t+1})$$

4.2.3 Reward Function

The reward function $R : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ as in Equation (4.2) for the conducted experiments and this environment aims to minimize the distance between the current end-effector position $p_{\text{current},t}$ and the current target position $p_{\text{target},t}$. The current end-effector position is calculated by the forward-kinematics function on state-angles q_t plus action a_t. The target position is sampled at the beginning of an episode and stays constant throughout the episode until completion or time limit is reached.

$$R(s_t, a_t) = ||FK(s_{t,(4,\ldots,N+4)} + a_t), s_{t,(0,1)}||_2 \tag{4.2}$$

If you want to normalize the reward function you have to divide R by N.

In Figure 4.2 we can see the linear gradient for any current position towards the target position at $[-0.5, 0.5]$.

4.3 Dataset Creation

To realize the idea of joining a latent model to the actor network of SAC we do need a dataset to train such latent model. A dataset should contain environment state

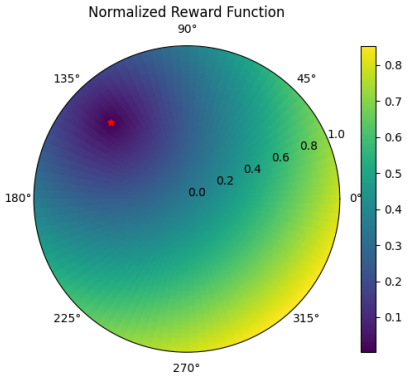


Figure 4.3: Normalized Reward function with $q \in [0, 2\pi]^N$ as element of the whole space and $p_{\text{target}} = [-0.5, 0.5]$. The target position p_{target} is marked with a red start.

information as well as possible actions. Consistent over all experiments with the VAE and Supervised Model sizes of the datasets are consistent:

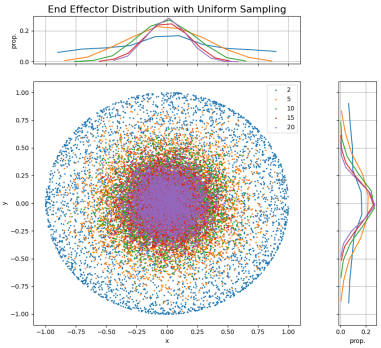
- train: 10.000 data points
- validation: 2000 data points
- test: 1000 data points

4.3.1 Uniform Sampling

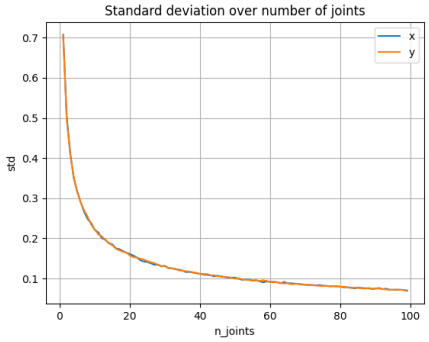
The vanilla sampling algorithm for sampling a dataset is to sample state angles q_t and actions a_t from a uniform distribution

$$q_t, a_t \sim \mathcal{U}^N_{[0,2\pi)} \tag{4.3}$$

and the target position with



(a) Scatter plot of different end effector positions generated by applying forward kinematics on uniformly sampled angles. The end effector positions are normalized by their corresponding number of joints. This leads to the constrain that all positions have a maximum distance to the origin of 1. For each number of joints we sample 5e3 angles. The upper and right plot are describing the desnsity function of an end-effector position.



(b) Standard deviation of end effector positions in x and y direction with repsect to the number of joints for a robot arm. For each number of joints we sample 1e4 angles.

Figure 4.4: Properties of dataset with actions sampled from a uniform distribution.

$$p = [\cos(\beta), \sin(\beta)] \cdot r \tag{4.4}$$
$$\beta \sim \mathcal{U}_{[0,2\pi)} \quad r \sim \mathcal{U}_{[0,N]}$$

While observing $p_{\text{current},t}$ in Figure 4.4a we can observe that with an increasing number of joints the scattered dots are forming a two dimensional gaussian distribution around the origin. As we can see in Figure 4.4b the standard deviation is exponentially decreasing.

In our application this makes the vanilla algorithm not suitable because we are not covering the observation space close to the maximum reach of the robot arm with more than 2 joints.

4.3.2 Expert Guidance

The problems encountered with the vanilla sampling algorithm in Section 4.3.1 lead to the development of an sampling algorithm guided by an expert.

Algorithm 5 Expert Guided Dataset Creation

Input: number of joints: N, number of samples: K.
$\mathcal{D} \leftarrow \{\}$ Dataset
for $i \in [0, \ldots, N-1]$ **do**
  $p_{\text{current}} \leftarrow$ (4.4) sample a position in 2D space with Equation (4.4)
  $q_t \sim \mathcal{U}$ sample initial arm angles from a uniform distribution as in Equation (4.3)
  $q_t \leftarrow IK(q_t, p_{\text{current}})$ solve IK for p_{current} and with q_t as start
  $p_{\text{target}} \leftarrow$ (4.4) sample a position in 2D space with Equation (4.4)
  $a_t \leftarrow IK(q_t, p_{\text{target}}) - q_t$ IK for the target position
  $\mathcal{D}_i \leftarrow ((p_{\text{target}}, p_{\text{current}}, q_t), a_t)$
end for

The clear advantage from this algorithm is that we ensure now that p_{target} and p_{current} are uniform with respect to direction and distance to origin.

On the other hand we are now bound to the actions the IK-solver, in our application CCD as in Algorithm 4, is providing. This could lead to unpredicted behavior of the RL agent if we leaf the covered part of q_t as shown in Figure 4.5. Another disadvantage is that this algorithm is now much slower compared to the vanilla sampling algorithm.

To have a more detailed look into the runtime complexity we focus in the runtime complexity of CCD as this is the slowest part of Algorithm 5. We can observe that the runtime of each CCD run is different since CCD belongs to the family of iterative inverse kinematics solver and the runtime is either limited by a given maximum

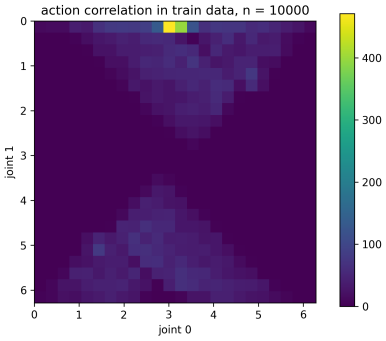
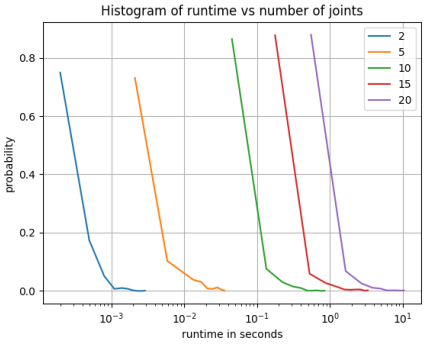


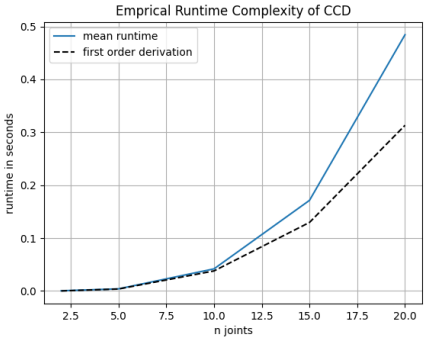
Figure 4.5: Plotted is the state angle correlation of two joints as a 2D histogram. Each bucket can be translated to the amount of angles in one 3D bucket b with limits $b_{0,0}, b_{0,1}, b_{1,0}, b_{1,1}$. Mathematical speaking: $|b| = |\{q_t | q_t \in [0, 2\pi)^2 \wedge q_{0,t} \in [b_{0,0}, b_{0,1}] \wedge q_{1,t} \in [b_{1,0}, b_{1,1}]\}|$

number of iterations or a break condition depending on how close we are to the goal. In Figure 4.6a we can see that all continuous probability density functions have a similar shape but with different standard deviations and means. Since we are interested in the runtime complexity of CCD we plotted the mean runtime of each density function in Figure 4.6b. Here we can observe that the runtime complexity of CCD is at least $\mathcal{O}(N^k)$ with $k \geq 3$ since the first order empirical derivation is still at least quadratic. Further derivations would be not plausible because with each derivation we would decrease the number of datapoints by one and therefor having no points left anymore.

Another interesting point of view is the correlation between resulting angles from ccd. Since we are limited to maximal two dimensional plot we can only plot a heatmap to show the correlations between state joint actions as in Figure 4.5. Important to note is that the actions are already in environment format and therefor are absolute angles with respect to a horizontal line in positive x direction. We can clearly see that CCD does not use the whole space but only focuses on a rhombus shaped area.



(a) continuous desnsity function of collected runtimes of ccd over different amount of joints. For each joint we sampled 1000 differnt start conditions, (target, angles) for ccd to solve inverse kinematics.



(b) plotted is the mean runtime for 1000 runs of ccd over the number of joints

Figure 4.6: Runtime complexity CCD

4.4 Latent Criterion

In this section I will cover the employed loss functions to train the latent models. As discussed in Section 3.5.2 we employ the Evidence Lower Bound as the objective function to train Variational Autoencoder. To fit our problem of inverse kinematics we tried out 3 different reconstruction loss function.

In this section we will explain the tangible KL divergence as well as the individual reconstruction loss functions.

4.4.1 Kulback Leiber Divergence

The Kulllback Leiber divergence first pulbished by Solomon Kullback and Richard Leiber in 1951 **(TODO: cite: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-1/On-Information-and-Sufficiency/10.1214/aom** is mathematical measure of how a probability distribution P differs from a second

distribution P' and is denoted as $D_{\mathrm{KL}}(P||P')$. Inside the ELBO it functions as a regularization element between the latent distribution $p_\phi(z|x)$ and a target distribution $p(z)$. Throughout this thesis we take the standard normal distribution as the target distribution, $p(z) = \mathcal{N}(0, I)$. Since we implement a parameterized gaussian as the latent distribution due to the nature of continuous input data, $p_\phi(z|x) = \mathcal{N}_\phi(\mu_\phi(x)|\ \Sigma_\phi(x))$. Therefor we can calculate the Kullback Leiber Divergence as:

$$\begin{aligned} D_{\mathrm{KL}}(\mathcal{N}_\phi(\mu_\phi(x)|\Sigma_\phi(x))||\mathcal{N}(0,I)) &= \frac{1}{2}\left(\mu_\phi(x)^T\mu_\phi(x) + tr\left(\Sigma_\phi(x)\right) - K - \log|\Sigma_\phi(x)|\right) \\ &= \frac{1}{2}\left(\mu_\phi(x)^T\mu_\phi(x) + \sum_{i=1}^{K}\Sigma_\phi(x)_{ii} - K - \log\left(\sum_{i=1}^{K}\Sigma_\phi(x)_{ii}\right)\right) \end{aligned}$$

(TODO: employ in code and test diffferent KL divergence) (TODO: proof in appendix from https://mr-easy.github.io/2020-04-16-kl-divergence-between-2-gaussian-distributions/) (TODO: KL div)

4.4.2 Reconstruction Loss

Inside the Evidence Lower Bound objective, the reconstruction loss minimizes the distance between the input data x and the model output $\hat{x}$. In our experiments we employed three different approaches for the reconstruction loss:

Imitation Loss. The imitation loss is a criterion to minimize the mean squared error between a given label y, in this case an action from an expert which results in $y \in \mathbb{R}^N$ and the predicted action $\hat{x} \in \mathbb{R}^N$. It is defined as in Equation (4.5).

$$\begin{aligned} \mathcal{L}_{\mathrm{Imi}} : \mathbb{R}^N \times \mathbb{R}^N &\to \mathbb{R} \\ y, \hat{x} &\mapsto \frac{1}{N}\sum_{i=0}^{N-1}(y_i - \hat{x}_i)^2 \end{aligned} \tag{4.5}$$

Distance Loss. This loss function minimizes the distance between a desired position in 2D space as label p_{target} and the action outcome of a state action combination. The state information needed for this criterion are only the current robot arm angles q. Like before the action is referred to as $\hat{x}$. **(TODO: wirte forward kinematics algorithm)**

$$\begin{aligned} \mathcal{L}_{\text{Dist}} : \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^N \to \mathbb{R} \\ p_{\text{target}}, \hat{x}, q \mapsto ||p_{\text{target}} - \text{FK}(q + \hat{x})||_2 \end{aligned} \tag{4.6}$$

This criterion is also used in a standard supervised learning approach where the model predicts actions based on state information.

Inverse kinematics Loss. This criterion as in Equation (4.7) is the result of merging distance loss and imitation loss in one loss function as a weighted sum of those two components with weights $w_{\text{Imi}}, w_{\text{Dist}} \in \mathbb{R}$. Inside the code, the loss function can be configured to work also a pure distance loss function without providing any expert action and make it therefor also applicable to the afore mentioned supervised learning approach.

$$\begin{aligned} \mathcal{L}_{\text{IK}} : \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^N, \mathbb{R}^N \to \mathbb{R} \\ p_{\text{target}}, \hat{x}, q, y \mapsto w_{\text{Imi}} \cdot \mathcal{L}_{\text{Imi}}(y, \hat{x}) + w_{\text{Dist}} \cdot \mathcal{L}_{\text{Dist}}(p_{\text{target}}, \hat{x}, q) \end{aligned} \tag{4.7}$$

4.5 Learning the Latent Model

For the upcoming experiments we settled one setting with one kind of dataset, the TargetGaussian.
The idea behind the gaussian target dataset is to learn an action which moves the

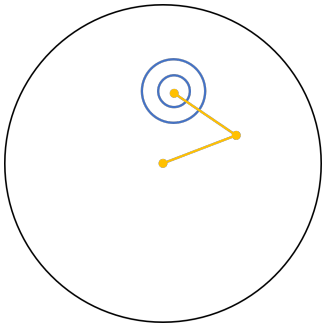


Figure 4.7: Schematic drawing how the target gaussian dataset works. First we are going to take the state as is comes form Algorithm 5 (in yellow). Then we are going to sample a target position from inside a the two dimensional truncated or non truncated gaussian distribution (in blue) and define it as new target position. Optionally we can also compute the corresponding CCD action.

Name	Value
VAE x	$p_{\text{target},t}$
Supervised x	s_t
c	$(p_{\text{current},t}, q_t)$
$\hat{x}$	q_{t+1}

Table 4.1: Latent workflow parameter with their correlation

end - effector towards a target sampled from a two dimensional gaussian distribution around the end - effector as demonstrated in Figure 4.7.

While training the Variational Autoencoder we settled on hyperparameter as in Table **??** and input vectors as in Table **??**. It is important to note that x for VAE, concatenated with c equal to s_t is which is the same input as for the Supervised model.

While training VAE and Supervised model you have to account the additional `tanh` stage in the back of Figure 4.1. This additional stage in introduced with an instance of `PosProcessor`. The `PosProcessor` is enabled during training the latent model but deactivated during training SAC, because SAC applies a `tanh` call in the `PolicyNet`.

4.6 Software

In the following section we are going to present the developed software stack to train the latent models as well as the reinforcement learning agent.

4.6.1 Inverse kinematics Environment

As presented previously in Section 4.2 we tackle the problem of inverse kinematics of a robot arm with N many joints in a two dimensional setting. The implemented environment

4.6.2 Latent Module

The latent module contains every functionality regarding either a Variational Autoencoder or a simple feed forward supervised regression model. As a deep learning library we build on pytorch 1.**(TODO: find out version)**.

You can find out about the functionality either with

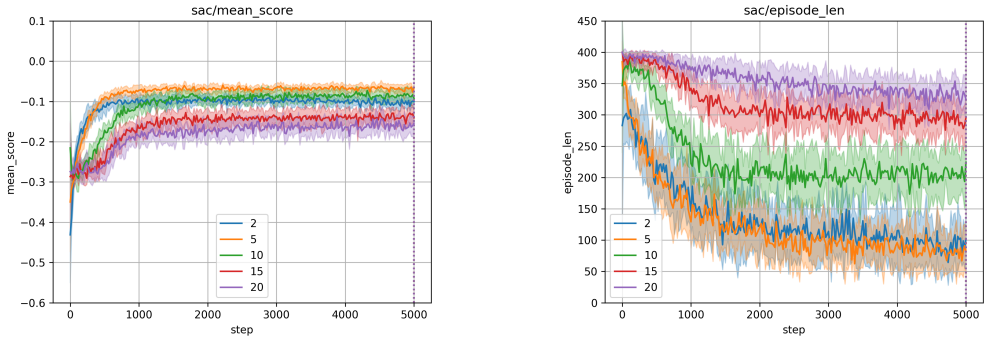
4.6.3 Soft actor critc

5 Experiments

In this chapter we want to present the experimental results. We focus hereby on five main sections:

- **Baseline SAC**: To see how SAC solves the environment without any further help.
- **VAE**: Where we look into the results how the Variational Autoencoder.
- **SAC with VAE**: How SAC performs together with the VAE decoder.
- **Supervised**: We are looking into results how the supervised approach is handling the low level inverse kinematics task.
- **SAC with Supervised**: How SAC is performing using the supervised model as a low level controller.

In this thesis we are focusing on merging the Soft Actor Critic with a decoder from a pretrained Variational Autoencoder. Results on merging SAC with a pretrained supervised model are only for comparison and to leverage the importance why someone should use a VAE instead of a supervised neural network.



(a) Mean reward per step over the last 20 episodes.

(b) Mean episode length over the last 20 episodes

Figure 5.1: SAC baseline experiment results with an increasing number of joints. We can clearly see that the algorithm does not scale well with an increasing number of joints and therefor drops in performance. Each experiment was conducted 10 times and the shaded areas resemble the standard deviation around the mean.

5.1 Baseline SAC

First we would like to have a look how a the plain SAC algorithm, where the actor network directly returns actions for the environment, will perform on the inverse kinematics environment as described in Section 4.2. Further we will describe this kind of experiments as SAC baseline experiments. All Hyperparameters used for the SAC baseline experiments can be found in Chapter 9.

In Figure 5.1 we plotted the mean reward per step and the average episode length over the last 20 epochs for different $N \in [2, 5, 10, 15, 20]$. Overall we can observe a decrease of collected reward per step in Figure 5.1a and an increasing amount of steps in one episode in Figure 5.1b while increasing the number of joints N. In Figure 5.1b the training curves are approaching the maximum step limit of one episode while increasing the number of joints. Additionally we can observe that the standard deviation over multiple experiments is decreasing while increasing the number of joints.

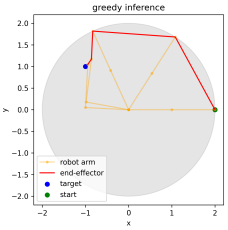
To show the actual behavior of an agent in a standardized setting we also plot an inference episode in Figure 5.2 and the distance remaining to target in Figure 5.3. The standardization in Figure 5.2 for one selected experiment for a relative consistent target position at $[-0.5, 0.5] \cdot N$.

As expected from the performance statistics in Figure 5.1 agents with two and five joints perform much better than agents with ten or more joints.

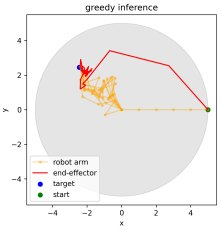
Interesting to observe is that all agents are minimizing the distance to the target quite fast in their first couple of steps. But after missing the target closely the agent behavior becomes kind of desperate and the end effector moves towards the origin. After increasing the distance by moving towards the origin the distance starts to oscillate as we can see in Figure 5.2d and Figure 5.2e and nevers goes below the threshold of 0.1. Another interesting detail to observe is the strategy an agent is following to reach the target position. Instead of moving first closely to a straight line towards the target position the agent is moving its end-effector closely to the outer limit of its reach. As we have seen in section Section 4.3.1 randomly having an end-effector position far away from the origin is less probable compared to a position close to the origin.

5.2 VAE

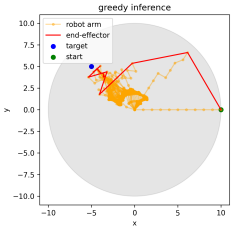
In the following section I will present the results of our experiments with the Variational-Autoencoder to encode and decode actions from the environment but also learn to solve inverse kinematics by minimizing the presented distance loss form Equation (4.6). Within the section we are covering two loss function setups. The first setup we are going to concentrate on only uses the distance loss while the second setup incorporates also an imitation loss.



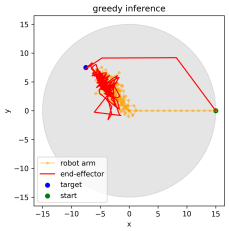
(a) From experiment 2_1691621262 at checkpoint 5000



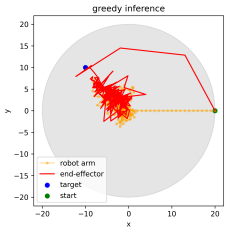
(b) From experiment 5_1691624939 at checkpoint 5000. To make things more clear only every 5th robot arm is drawn



(c) From experiment 10_1691622498 at checkpoint 5000.To make things more clear only every 5th robot arm is drawn

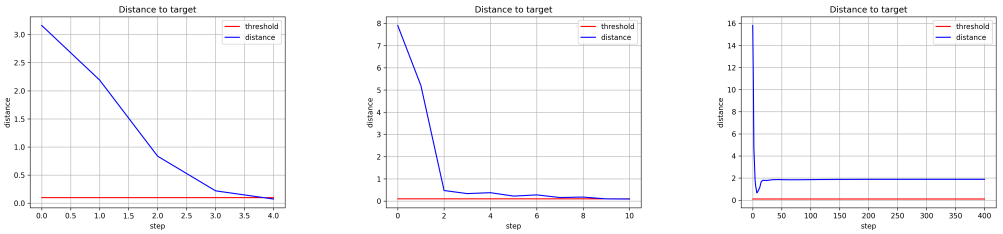


(d) From experiment 15_1691619106 at checkpoint 5000. To make things more clear only every 40th robot arm is drawn



(e) From experiment 20_1691619159 at checkpoint 5000. To make things more clear only every 40th robot arm is drawn

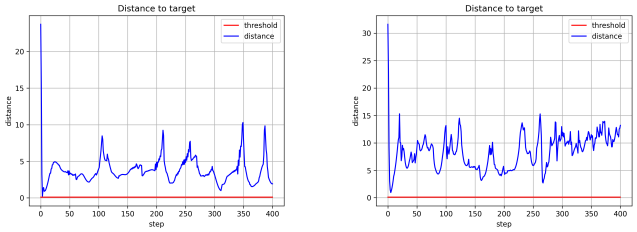
Figure 5.2: SAC inference. The trajectory is plotted in red. robot arms are drawn in yellow with the little dots as positions of each joint. Target position and start position are scattered in blue and green. The space an end-effector is able to reach is plotted in grey.



(a) Distance to target policy from experiment 2_1691621262 at checkpoint 5000

(b) Distance to target policy from experiment 5_1691624939 at checkpoint 5000

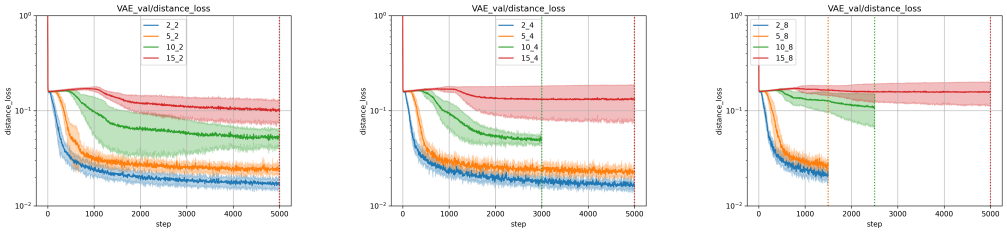
(c) Distance to target policy from experiment 10_1691622498 at checkpoint 5000



(d) Distance to target policy from experiment 15_1691619106 at checkpoint 5000

(e) Distance to target policy from experiment 20_1691619159 at checkpoint 5000

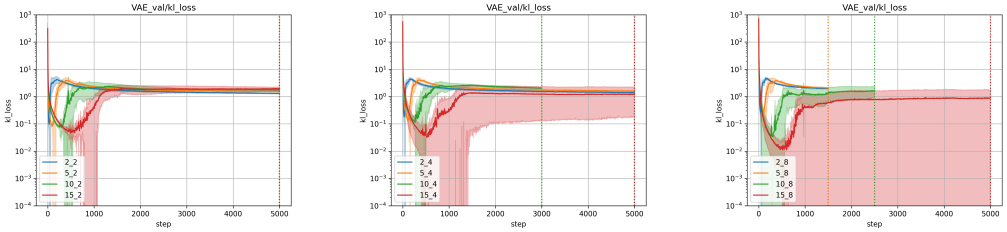
Figure 5.3: Distance to target for each action outcome. The target is consistent at $[-0.5, 0.5] \cdot N$. The threshold of 0.1 at which an episode is concluded successfully is drawn in red.



(a) mean reconstruction loss over the las 20 epochs for **latent dimension = 2**

(b) mean reconstruction loss over the las 20 epochs for **latent dimension = 4**

(c) mean reconstruction loss over the las 20 epochs for **latent dimension = 8**



(d) mean KL-loss over the las 20 epochs for **latent dimension = 2**

(e) mean KL-loss over the las 20 epochs for **latent dimension = 4**

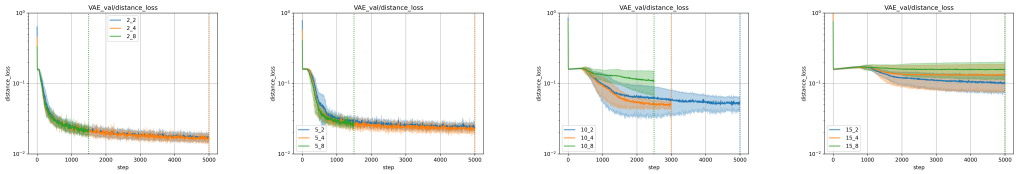
(f) mean KL-loss over the las 20 epochs for **latent dimension = 8**

Figure 5.4: VAE validation results over different amounts of joints and with a latent dimension of 4. Each experiment was conducted 10 times with different random seeds. The solid curve is the average over those 10 experiments and the color shaded area resembles the standard deviation. Notice that the y axis in bot plots is in log scale.

5.2.1 Distance Loss

Coming up are the training results for different latent dimensions but only with the distance loss enabled which means `dataset target mode = POSITION`.
Because we only leverage the usage of the distance loss we do not need to encode and decode an action from an inverse kinematics solver, we just need to encode target information where we want to go plus state information as conditional information.

In Figure 5.4 we have a closer look into the reconstruction loss and kl loss. While increasing the number of joints $N \in [2, 5, 10, 15]$ we are also increasing the level of



(a) $N = 2$ **(b)** $N = 5$ **(c)** $N = 10$ **(d)** $N = 15$

Figure 5.5: Comparison on the VAE latent dimension based on validation results over different amounts of joints. Each experiment was conducted 10 times with different random seeds. The solid curve is the average over those 10 experiments and the color shaded area resembles the standard deviation. Notice that the y axis in bot plots is in log scale. In the legend each label is structured as `<N>_<latent dimension>`

complexity the network has to master to come up with suitable action which leads to a low distance loss. The increasing level of complexity is quite noticeable in both the kl loss and reconstruction loss.

All KL-Loss curves in Figure 5.4 (d) to (e) are shaped in the same way. First a drop before rising with a little overshoot and finally approaching the terminal value asymptotically. While increasing N we can see that the first local minimum shifts to the right, the upwards slope becomes slacker and the overshoot diminishes. This behavior confirms the the theory of increasing complexity while increasing the number of joints.

We can find another indicator by analyzing the reconstruction loss in Figure 5.4 (a) to (c). For all joints it is quite clear that the learning curves are shaped similar but approaching different final performances. For those experiments of $N \in [2, 5]$ we get a reconstruction loss < 0.02 independent of the latent dimension. But if we turn towards $N = 10$ we can clearly see that the latent dimension does make a significant difference between finding a solution, with a latent dimension of four, or failing with a latent dimension of eight. A further increase of N does not lead to better reconstruction losses across all latent space dimensions.

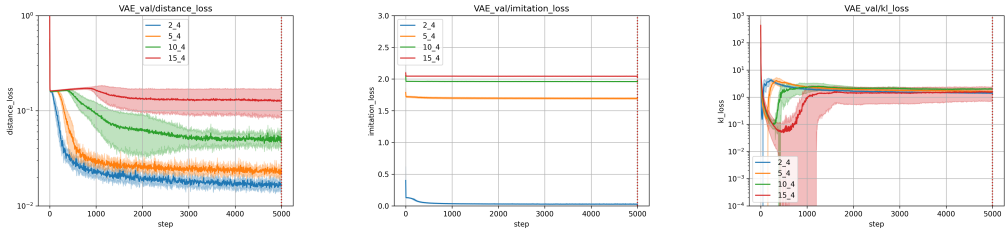
By observing the individual number of joints in each plot of Figure 5.5 form (a) to

(d) we can observe that the latent dimension does make a difference in the final reconstruction loss performances. Starting with two joints there is no significant difference between the latent dimensions of two to eight. Continuing with 5 joints we can observe no significant difference in the finial performance but a slight difference with respect of how fast the models are converging. The experiments on a latent dimension of four and eight are very similar while the standard deviation for a latent dimension equal to two increases around 600 epochs indicating that some experiments have slight troubles to converge.
A significant difference in final performance can be observed with $N = 10$. Here the best performance is returned by experiments with a latent dimension of four closely followed by two. Comparing two and four in Figure 5.5c shows that four is much more stable and faster in learning the problem as two. Last ranked are the experiments for a latent dimension of eight. Those experiments are by far not on the same level as with a latent dimension of two and four. A deeper look into the inference shows that the model fails to come up with an action to reach the given target position.
In Figure 5.5d a performance difference is noticeable with a latent dimension of two as best and eight as worst. But since the best performance of the experiments with a latent dimension of two is on the same level as eight in Figure 5.5c we can conclude that the the VAE fails at $N = 15$ with the present hyperparameter as in Table **??**.

5.2.2 Distance plus Imitation Loss

In this section we are going to present the results of training the VAE with the imitation loss to pre-calculated solutions from Algorithm 4. The important change in the config file is setting `dataset target mode = ACTION` to tell the dataset to also calculate action from ccd.

Because of time limitation we applied this loss function only on a setting with a latent dimension of four since this was the best latent dimension fitted only with the distance loss.



(a) Distance loss

(b) Imitation loss

(c) KL diveregence from standard normal distribution

Figure 5.6: Results on the validation dataset while training a VAE with a latent dimension of 4 but incorporating an imitation loss weighted with 0.01. Results are collected from ten different runs and with a solid mean curve and a shaded standard deviation area. $N \in [2, 5, 10, 15]$

All in all the distance loss functions in Figure 5.6a yield almost the same examples as the distance loss functions in Figure 5.4b.

Additional to the distance loss we have got also an imitation loss between the computed action from CCD and the action yielded from form the neural Network in Figure 5.6b. Here it is interesting to note that the imitation loss is increasing alongside N but not in the same scale. Where the differences between the imitation losses are becoming smaller for bigger N. Another thing that is particular interesting is the convergence behavior of each curve. They all hav a huge drop in the beginning but stay on their level for the rest of the training as if the have reached their optimal value.

Finally looking at the KL-divergence we do can note an almost indistinguishable plot form Figure 5.4e.

5.3 SAC with VAE

In this section we are going to talk about the experimental results of combining the decoder from a Variational Autoencoder with soft actor critic. As we know from Chapter 4 we only use the decoder of the trained Variational Autoencoder with

N	latent dimension					
	2		4		8	
	random seed	epoch	random seed	epoch	random seed	epoch
2	1693105015	4950	1692461245	4890	1691608881	1480
5	1693096269	4495	1692476369	4795	1691606505	1290
10	1693098415	3615	1691627455	2535	1691608513	2485
15	1693103204	4815	1691628295	3845	1691618374	4760

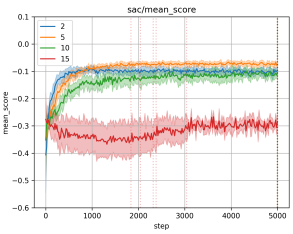
Table 5.1: Used checkpoints to train SAC with decoder from VAE. All experiments can be found in `results/vae/<`N`>_<latent dimension>_<random seed>/VAE_<epoch>*.pt`

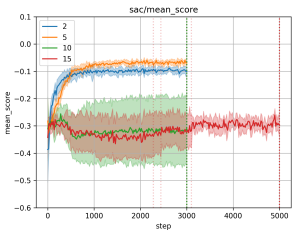
input from the corresponding state and a latent action directly sampled from the parameterized distribution from the actor network.

As we have seen in the previous section we carried out VAE experiments with three different latent space dimensions. Accordingly we trained ten SAC runs for each latent space size and number of joints with $N \in [2, 5, 10, 15]$. We used the best VAE checkpoint available according to the overall validation loss. The detailed listing which checkpoints where actually used can be found in Table 5.1.

In Figure 5.8 we can observe what difference the choice of latent dimension has on the performance of the Soft Actor Critic algorithm. Quite obviously to see is that the the choice of latent dimension really starts to matter after 5 joints. Where at 10 joints 2 and 4 succeed but 8 doesn't. This is clearly related to the performance of the VAE during its training. There we were also able to see that the latent dimension has an impact of the reconstruction loss performance of the VAE. Therefor it is no surprise to see that the best VAE checkpoints on 15 joints and the best VAE checkpoint on 10 joints with a latent dimension of 8 fails for the SAC + VAE setting.

By having a closer look into the difference between the collected mean score we can see that for two joints all mean curves, the solid ones, are packet closely together where at five joints we can start to see a slight decrease in performance at 1000 epochs continuing up to 5000 epochs, between the experiments done with a latent dimension

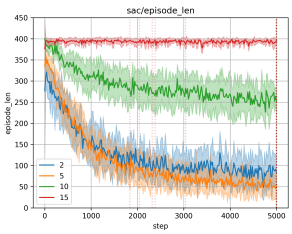


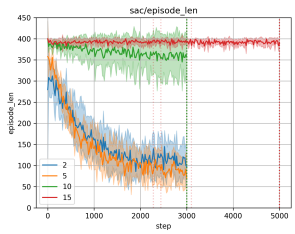


(a) mean episode reward per step averaged over the last 20 episodes. With a **latent dimension = 2**

(b) mean episode reward per step averaged over the last 20 episodes. With a **latent dimension = 4**

(c) mean episode reward per step averaged over the last 20 episodes. With a **latent dimension = 8**



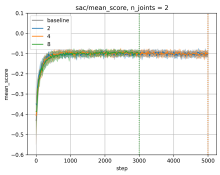


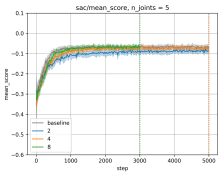
(d) mean episode length averaged over the last 20 episodes. With a **latent dimension = 2**

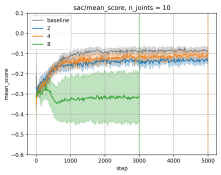
(e) mean episode length averaged over the last 20 episodes. With a **latent dimension = 4**

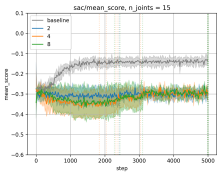
(f) mean episode length avergaed over the last 20 episodes. With a **latent dimension = 8**

Figure 5.7: SAC + decoder form VAE with different latent dimensions. Each experiment was conducted 10 times with different random seeds. The solid strong line is the mean over those 10 experiments. The color shaded area covering the mean is the standard deviation around the mean.



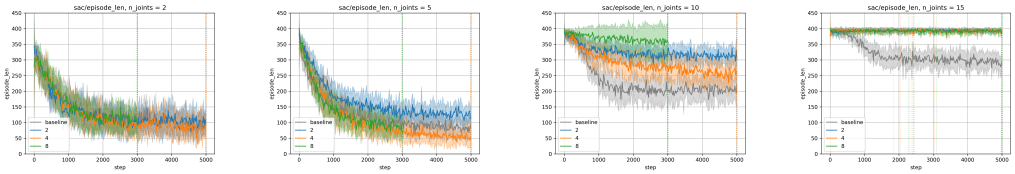






(a) $N = 2$ **(b)** $N = 5$ **(c)** $N = 10$ **(d)** $N = 15$

Figure 5.8: Shown are the collected mean scores over the the last 20 episodes per step, compared different latent dimensions and the baseline. Solid line is the mean over 10 experiments and the color shaded area is the corresponding standard deviation. Higher means better with 0 as the maximum. The colored dotted lines mark an end of an experiment.



(a) $N = 2$ **(b)** $N = 5$ **(c)** $N = 10$ **(d)** $N = 15$

Figure 5.9: Shown are the mean episode length over the the last 20 episodes, compared different latent dimensions and the baseline. Solid line is the mean over 10 experiments and the color shaded area is the corresponding standard deviation. Lower means better with 1 as the minimum and 400 as the maximum.

of two and the others.

This decrease is even more noticeable at ten joints where is now a gap between the experiments done with a latent dimension of four and the baseline experiments too.

(TODO: set in perspective of VAE performance)

In Figure 5.9 we can observe the mean episode length for different $N \in [2, 5, 10, 15]$. Starting at two joints we can see not much of a difference but at five joints we can observe slight differences between different latent dimensions. First up is the latent dimension of two At this value the experiments perform as in Figure 5.8b worst.

Moving on to a latent dimension of four, those experiments perform best in terms of mean episode length and even undercut the baseline experiments although they have not performed as good in terms of the mean score per step in Figure 5.8b. Finally those experiments with a latent dimension of eight are performing almost on the same level as the baseline experiments but not as good as the experiments with a latent dimension of four.

In Figure 5.8c we can see wide variety of best performances. The best performing setting are the baseline experiments followed by the experiments with a latent dimension of four. The next best performance is coming from a latent dimension equal to two and last ranked are the experiments with a latent dimension of eight. As we have mentioned earlier we suspect this drop in performance caused by the quality

(a) mean episode reward per step averaged over the last 20 episodes.

(b) mean episode length avergaed over the last 20 episodes

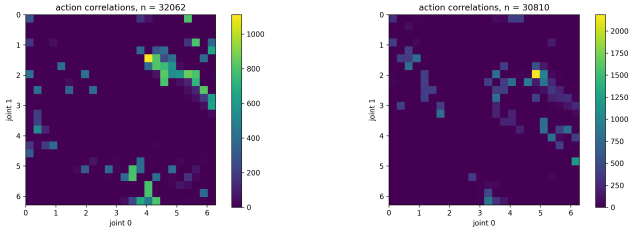
Figure 5.10: SAC + decoder form VAE with **latent dimension = 4** and trained with imitation loss enabled. Each experiment was conducted 10 times with different random seeds. The solid strong line is the mean over those 10 experiments. The color shaded area covering the mean is the standard deviation around the mean.

the decoder gets hand over to the Soft Actor Critic.

Note that although the maximum number of steps per episode was set to 400 the standard deviation of the experiment with a latent dimension of eight seam to surpass this threshold which is with a closer look into the actual curves clearly not the case. **(TODO: insert plot in appendix)**.

On the performed experiments with 15 joints we can see that none of the latent experiments is able to match the performance of the baseline experiments with even a couple of experiments with a latent dimension of four or eight aborting early. As it turns out this abort is caused by abnormally high values for alpha loss as shown in Figure 9.1 resulting in `nan` values in update steps. A positive alpha loss translates to a an increase of alpha translates to an increase in exploration for the policy, by encouraging a higher standard deviation for the parameterized distribution returned by the actor network.

Because we have seen a lack in performance between the baseline SAC experiments we additional trained VAEs with the imitation loss enabled to minimize the distance



(a) SAC baseline experiments

(b) SAC + VAE with latent dimension = 4 and only distance loss enabled

(c) SAC + VAE with latent dimension = 4, distance and imitation loss enabled

Figure 5.11: Action correlation for baseline experiments, latent dimension only trained on distance loss and SAC + VAE trained on distance and imitation loss.

between the state angle distribution the decoder is receiving as in Figure 5.11 versus the state angle distribution the decoder is trained as in Figure 4.5. But if we have a look into Figure 5.11c and compare to the plots on the left we do not see a significant change. This could be extrapolated to higher N and explain the missing improvement we would expect. But since this is not the main focus of this thesis we leaf these steps for future research.

5.4 Supervised

For the supervised actions we only relied on state information as the input to predict an action to reach from the current state the given target position. Since this approach is very similar to the Variational Autoencoder setting we are also able to apply the afore mentioned Inverse Kinematics Loss with distance and imitation loss as in Equation (4.7).

The results of the supervised experiments are divided into two parts. The first is only applying the distance loss while the second one is incorporating also an imitation loss with respect ot pre computed actions from the CCD solver.

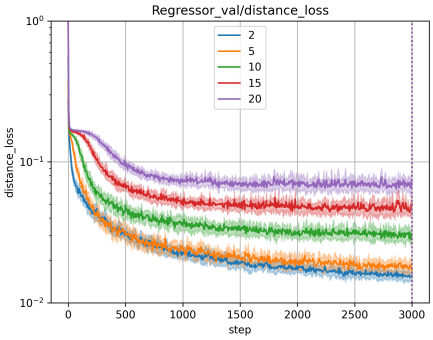


Figure 5.12: Distance loss for supervised experiments over different $N \in [2, 5, 10, 15, 20]$. For each N we conducted 10 experiments.

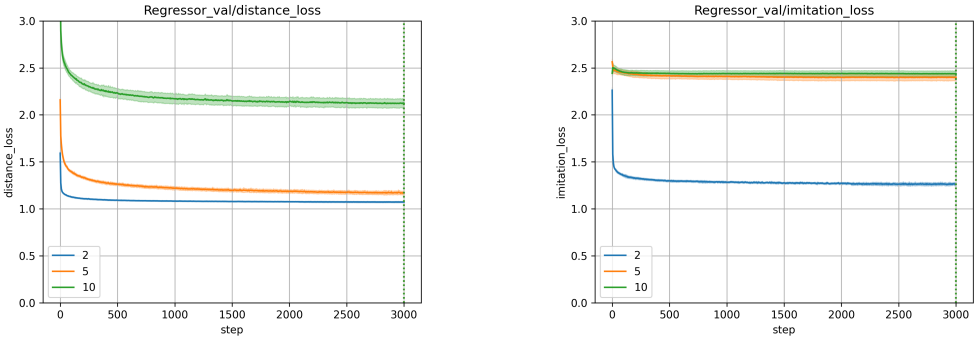
5.4.1 Distance Loss

The distance loss results over different amount of joints in Figure 5.12 are very similar compared to Figure 5.4 (a) to (c). Parallel to Figure 5.4 we can also observe an overall increase in loss while increasing N and finishing at different convergence levels. Different to the experiments on the VAE is the standard deviation between the experiments. The supervised experiments show a significant smaller standard deviation across the whole training period compared to the VAE experiments.

5.4.2 Distance + Imitation Loss

For this series of experiments we also want to emphasis solutions close to solutions computed by the CCD by setting the dataset mode to `ACTION` and setting the imitation loss weight to 0.01.

Unlike as in the previous section the distance loss differs a lot from Figure 5.12. We do not see a steady increase in loss over different numbers of joints instead we do see vastly different levels of final distance loss performances starting with two joints at around 1.1 and ending with 15 joints at XXX.
A bit more interesting are the results on the imitation loss. Similar to Figure 5.6b all



(a) Distance Loss **(b)** Imitation Loss

Figure 5.13: Validation results for supervised experiments on distance and imitation loss.

curves have a drop in the first couple of epochs but are constant for the rest of the training. Comparing those curves with respect to the number of joints we observed that all mean curves have vastly different levels of convergence which seems not to be linear correlated with N.

5.5 SAC with Supervised

Merging the supervised model with SAC yields performances as in Figure 5.14. We do can see very similar performances as with merging SAC with VAE.

experiment series	N					
	2		5		10	
	mean	std.	mean	std.	mean	std.
baseline	-5.16	± 0.95	-15.29	± 1.62	-32.51	± 2.64
latent actor 4	-13.45	± 2.93	-15.60	± 2.49	-15.45	± 4.42
supervised distance loss	10.96	± 2.07	41.76	± 2.75	39.71	± 2.89
supervised distance loss	-9.03	± 0.98	-10.54	± 0.41		

Table 5.2: Log probabilities of each policy series during training form the last 50 episodes.

(a) Mean score per step **(b)** episode length

Figure 5.14: Validation results for supervised experiments on distance and imitation loss.

A very

Experiments conducted with a supervised model trained with imitation loss enabled are shown in Figure 5.15.

(a) Mean score per step

(b) episode length

Figure 5.15: Validation results for supervised experiments on distance and imitation loss.

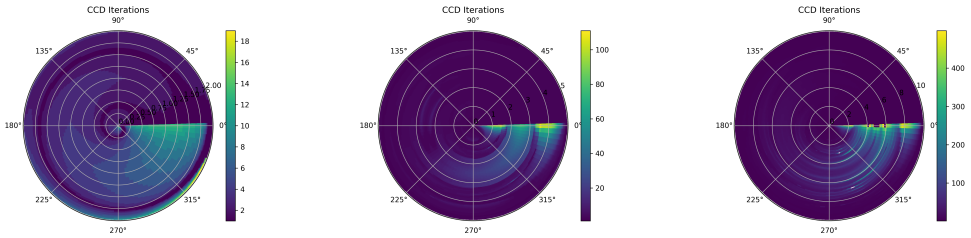
6 Discussion

6.1 Baseline SAC

Starting with the baseline results for SAC we have seen a clear correlation between the algorithms performance and the number of joints we train on. This is reasonable because the performance of other also other Inverse Kinematics solver mentioned earlier, including CCD are highly dependent on the number of joints required to solve for, as we have seen in the section about runtime complexity of CCD in Figure 4.6b.

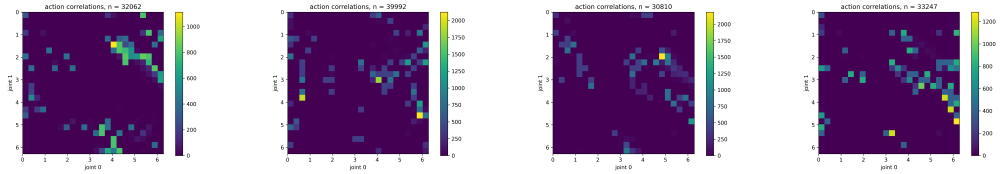
While comparing the CCD with baseline SAC we found a couple of differences those two solvers present.

- **Target Dependence**: By looking deeper into CCD it is clear that its granular performance in a single run is also highly dependent on the given start configuration and target as show in Figure 6.1. Note that we talk about the performance of CCD as the number of iterations needed to reach the desired target position. If we now look into the steps needed to reach the desired target position of SAC we can seed that while increasing the number of joints we are increasingly often capped by the maximum number of iterations than ending the episode by reaching the target. One possible reason is teased by Figure 5.2. Here we were able to see that the agents behavior sometimes collapses as it is about to reach the target but after failing very closely the end-effector moves



(a) $N = 2$ **(b)** $N = 5$ **(c)** $N = 10$

Figure 6.1: Heatmap of how many iterations CCD has needed to solve inverse kinematics starting from $[N, 0]$ targeting the middle of each polygon. The maximum budged are 20 times N with a target precision of 0.1.



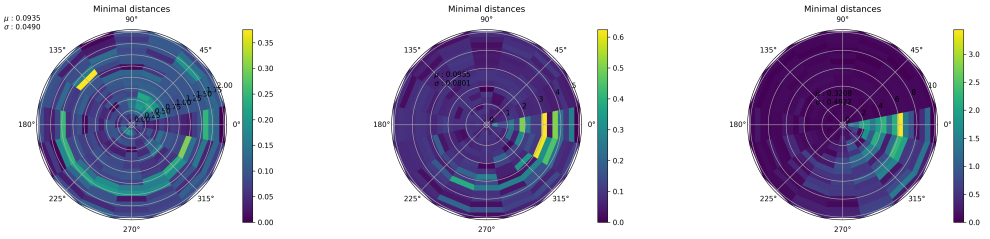
(a) baseline. number of samples = 32062 **(b)** laten dimension is 2. number of samples = 39992 **(c)** laten dimension is 4. number of samples = 30810 **(d)** laten dimension is 8. number of samples = 33247

Figure 6.2: Plotted is the state angle correlation of two joints as a 2D histogram. The state angles are sampled from multiple episodes to uniformly sampled target positions. Each bucket can be translated to the amount of angles in one 3D bucket b with limits $b_{0,0}, b_{0,1}, b_{1,0}, b_{1,1}$. Mathematical speaking: $|b| = |\{q_t | q_t \in [0, 2\pi)^2 \wedge q_{0,t} \in [b_{0,0}, b_{0,1}] \wedge q_{1,t} \in [b_{1,0}, b_{1,1}]\}|$

rapidly towards the origin and never coming as close as before to the desired target position. This observation is supported by Figure 6.4 where the minimal distance to the target can be found almost all of the time at around 50 or less steps in the episode. One reason why the agent struggles to reach the target with a increasing amount of joints could be the constant threshold to end an episode of 0.1 euclidean distance between end-effector and target position. As we increase the number of joints the area where the end-effector position ends an episode decreases due to $A = \pi N^2$ quadratically which makes it even harder to reach the position while increasing N. Against this argument stands that the robot arm is capable to perform more precise movements or even reducing the problem further down. One strategy could position joint 1 to $N - 2$ so the last two joints are able to reach the target and let do joint $N - 1$ to N do the precise work. But why is the agent failing with a high margin after it went so close towards the target? An explanation could be that the agent is reaching a target very often dependent on the actual target position and has no opportunity to learn from those experiences. On the other hand are the extended episode length and the increasing failure rate as in Table 6.1.

- **Solutions**: While having a deeper look into the solutions how SAC and CCD solve Inverse Kinematics we can observe a vast difference in the state angle distribution Comparing Figure 4.5 to Figure 6.2a. While in Figure 4.5 the state angles to reach the target distribution are rhombus shaped the state angle plot in Figure 6.2a is way more scattered. This change is also observable while looking deeper into the chosen trajectory.

Note that this is only an qualitative comparison because CCD has the full action space on its disposal while SAC is limited to -1 to one because of the action normalizer at the back of SAC. Although scaling the tanh output up to a span of 2π or higher is not the perfect solution because SAC could find multiple actions for the same action outcome. It could also interfere with computing the probability of $\log(\pi_\theta(a|s))$.



(a) $N = 2$ **(b)** $N = 5$ **(c)** $N = 10$

Figure 6.3: Minimal distance to a target during inference starting from $[N, 0]$ and targeting the middle of each polygon.

The presented decrease in the SAC performance in Figure 5.1 could be also explained because of the linear increase of complexity of the observation space since $S \subset \mathbb{R}^{4+N}$. To counter the increase of complexity some could also increase the number of hidden layer or neurons per layer and have a closer look into neural architecture search. Another way to maybe proof this theory could be the application of Neural-Evolution-of-Augmented-Topologies short NEAT. This algorithm finds the smallest possible architecture to conquer a given task. Given the provided background information we assume the found neural architecture increases in complexity while increasing N.

Comparing RL with SAC we already described the end-effector movements in Chapter 5. With the current training settings it is clear that SAC is clearly not the better solutions than a more standard approach like SAC as we can see for the solved rar

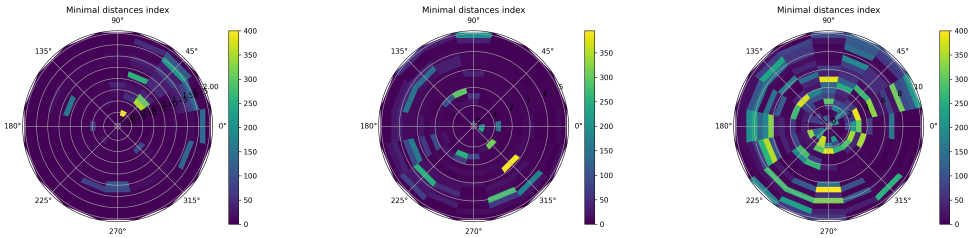
6.2 Latent Model

Moving on to the experiments to train a latent model. We train a latent model to augment SAC and let it control the latent model and therefor the actions from a higher level while having access to all state space information.

Since the standalone results of each latent model type are very similar I will discuss

N	baseline		latent 2		latent 4		latent 8	
	$\mathfrak{S}$	max R_t	$\mathfrak{S}$	max R_t	$\mathfrak{S}$	max R_t	$\mathfrak{S}$	max R_t
2	0.73	0.09	0.66	0.11	0.76	0.09	0.72	0.09
5	0.83	0.10	0.78	0.15	0.84	0.09	0.88	0.08
10	0.57	0.32	0.33	0.83	0.54	0.50	0.22	2.97
15	0.42	0.74	0.04	3.62				
20	0.30	0.82						

Table 6.1: Solved ratio $\mathfrak{S}$ and closest distance to target during one episode as known as the maximum reward during one episode, for all SAC experiments starting at $[N, 0]$ and targeting the same targets as in Figure 6.3 with a maximal budged of 400 steps. Those are examples from individual experiments all from epoch 5000. For more details which experiments please have a look into the figures directory. The suffix of each png describes where to find the experiment. For a closer inspection of how the individual models perform over the evaluation please refer to Chapter 9



(a) $N = 2$ **(b)** $N = 5$ **(c)** $N = 10$

Figure 6.4: Heatmap of how many iterations CCD has needed to solve inverse kinematics starting from $[N, 0]$ targeting the middle of each polygon. The maximum budged are 20 times N with a target precision of 0.1.

them together. Both ways to train a model with the TargetGaussian Dataset yielded almost the same performance in the distance loss. This is not very surprising as we have seen neural networks struggle with the inverse Kinematics task while increasing the number of joints N. Looking deeper into the model performance shows that the distance loss is dependent is also dependent on the state information we are providing in datasets. **(TODO: think about how to plot this)**

We also have similar results with the imitation loss **(TODO: big change in observation space distribution)**

6.3 SAC with Latent Model

Embedding the Latent Model into the workflow of SAC yielded performances as presented in Chapter 5. While understanding and seeing the vast difference in problem solving strategies between the trained latent models and the Inverse Kinematics solver CCD it is quite surprising this approach works in the first place. Usually such a significant change in the observation space distribution, leads to unpredictable and unfeasible behavior in the predicted actions in the Action space. We tried to master this issue by introducing the imitation loss to emphasis predicted actions leading to state angles which are much closer to the original distribution. **(TODO: look into results.)** Arguments that the approach actually works can be found by looking into a greedy inference with the latent models where we simulate a very light weight version of the Plane-Robot-Environment while assuming the optimal policy, following the direction from start to end.

Another interesting perspective are the different log probabilities yielded by the different experiments. While the actor network of SAC has to be very certain about what action to generate from the supervised latent model, the VAE as the latent model leaf a bit more

But why do we see differences for different latent dimensions? Looking first into those experiments with a latent dimension of 2. Since we are trying to encode the two dimensional target information which is sampled according to the idea of the TargetGaussianDataset, into a two dimensional gaussian manifold, the best information keeping strategy would be to pipe the mean values while returning a standard deviation close to zero. Because we do employ the ELBO with a gaussian distribution for a Variational Autoencoder we have to pick a parameterized latent distribution close the standard normal distribution. An additional difficulty is to encode the target position as independent random variables. Knowing these properties and constrains it could be quite challenging to compress. On the other hand choosing the latent dimension to high could add to much computational complexity, difficulty in sampling and a reduction in robustness.

As a result of the previous observations we can set the latent dimension of a VAE as an additional hyperparameter to optimize. Because setting it to high could lead the extremely high variance in the Soft Actor Critic task as shown in Figure **??** and setting it to low could lead to loss in encoded information.

7 Conclusion

8 Acknowledgments

First and foremost, I would like to thank Jasper Hoffman for his support.

I also thank Jan Ole von Harzt and Jens Rahnfeld for great discussions about Variational Autoencoders

- advisers
- examiner
- person1 for the dataset
- person2 for the great suggestion
- proofreaders

ToDo Counters

To Dos: 58; 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58

Parts to extend: 1; 1

Draft parts: 1; 1

Bibliography

[1] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[2] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 04, pp. 669–688, 1993.

[3] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.," *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

9 Appendix

9.1 Hyperparameters

Name	Value
num joints	-
segment length	1
n time steps	400
action mode	"strategic"
discrete mode	false
rescale actions enabled	False
rescale actions min	-1
rescale actions max	1
task type	"ReachGoalTask"
task epsilon	0.1
task bonus	0
task normalize	true
task order	2

Table 9.1: Hyperparameter for the inverse kinematics environment. Not that num joints is different from experiment to experiment. You can adjust those values by altering *config/base_ vae.yaml*

9.2 Additional Experiment Plots

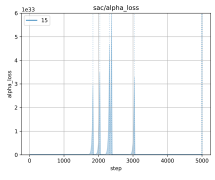


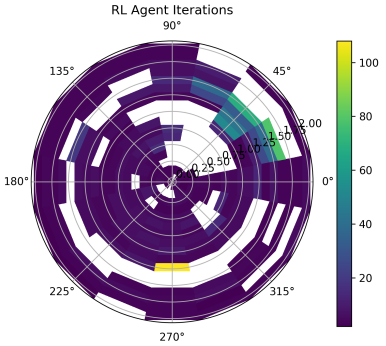
Figure 9.1: Plot to show the correlation between training abortion and spike in alpha loss.

Name	Value
num joints	-
num epochs	5001
learning rate	0.003
val interval	5
latent dim	-
encoder architecture	[128, 128]
encoder activation function	"ReLU"
decoder architecture	[256, 256]
decoder activation function	"ReLU"
kl loss weight	0.001
reconstruction loss weight	1
distance loss weight	1
imitation loss weight	0.001
dataset type	VAETargetGaussianDataset
dataset mode	"IK random start"
dataset batch size	2096
dataset shuffle	False
dataset action constrain radius	0.2
dataset std	0.2
dataset target mode	"POSITION"
dataset truncation	0
post processor enabled	True
post processor min action	-1
post processor max action	1

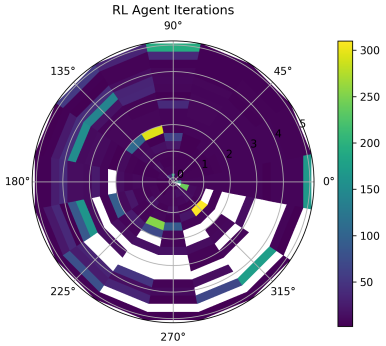
Table 9.2: Hyperparameter for Variational Autoencoder. Note that num joints and latent dim varies from between different types of experiments and is therefor not fixed. You can adjust those values by altering *config/base_ vae.yaml*. There are multiple datyset types available. But for the majority of our experiments we used the VAETargetGaussianDataset

Name	Value
lr q	0.001
init alpha	0.01
gamma	0.98
batch size	32
buffer limit	50000
start buffer size	1000
train iterations	20
tau	0.01
target entropy	-40.0
lr alpha	0.001
n epochs	5000
actor type	-
actor learning rate	0.0005
actor learning mode	0
actor latent dim	4
actor latent checkpoint dir	"results/vae/baseline"
actor architecture	[128, 128]
actor activation function	ReLU

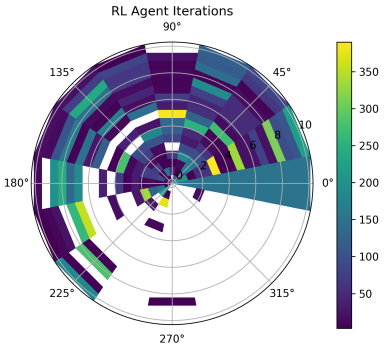
Table 9.3: Hyperparameter for SAC. Note that the actor type was left empty. For using a standard feed forward actor use "Actor", for using a VAE Decoder as latent model use: "LatentActor" and for using a separated pretrained supervised training model use "SuperActor".



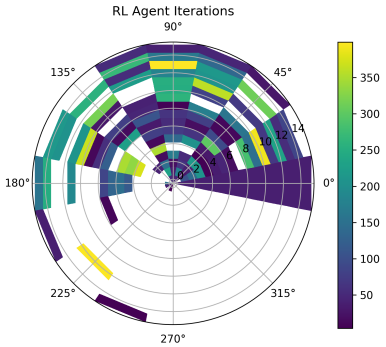
(a) $N = 2$



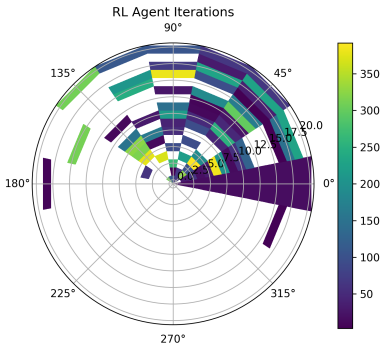
(b) $N = 5$



(c) $N = 10$

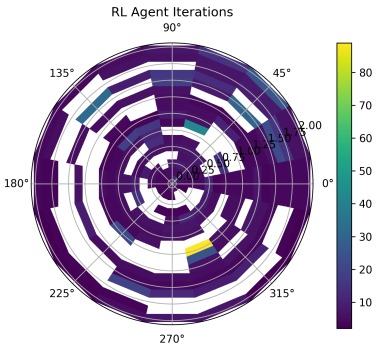


(d) $N = 15$

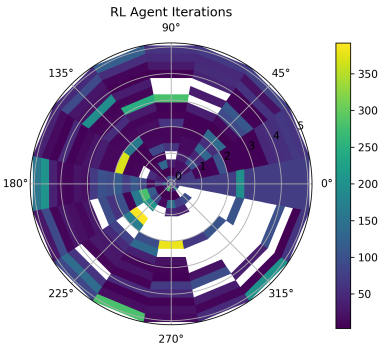


(e) $N = 20$

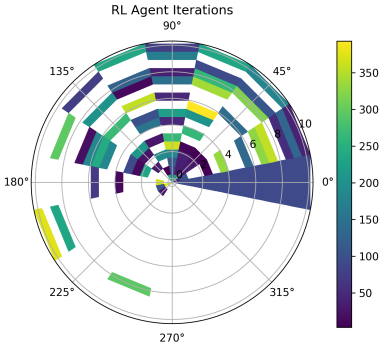
Figure 9.2



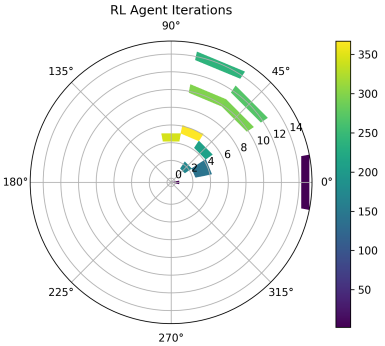
(a) $N = 2$



(b) $N = 5$

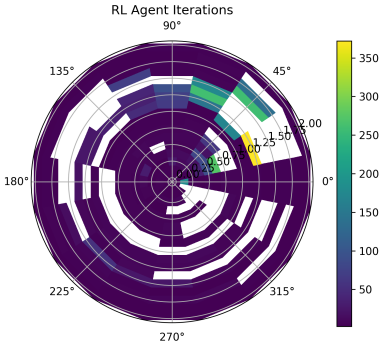


(c) $N = 10$

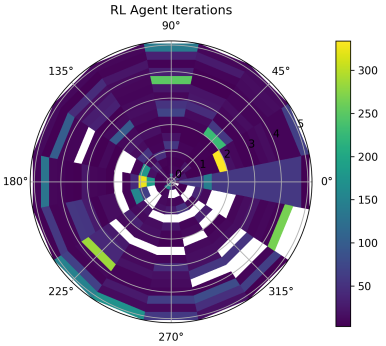


(d) $N = 15$

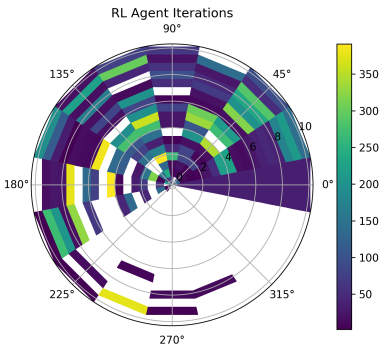
Figure 9.3



(a) $N = 2$



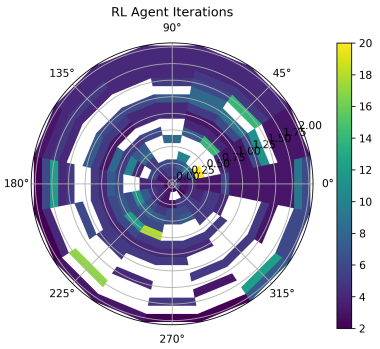
(b) $N = 5$



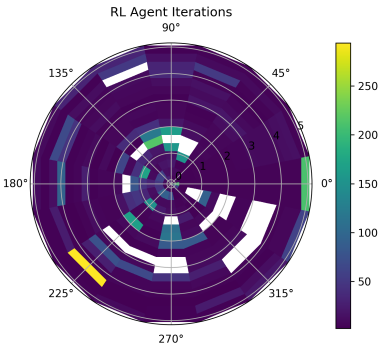
(c) $N = 10$

(d) $N = 15$

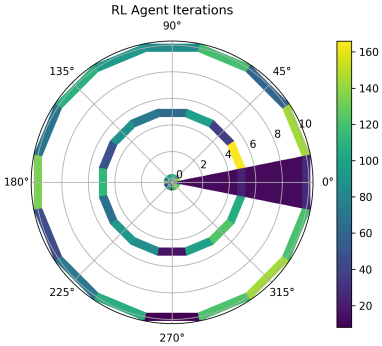
Figure 9.4



(a) $N = 2$



(b) $N = 5$



(c) $N = 10$

Figure 9.5