DEGREE PROJECT IN ENGINEERING PHYSICS 300 CREDITS, SECOND CYCLE

STOCKHOLM, SWEDEN 2015

Detecting anomalies in robot time series data using stochastic recurrent networks

MAXIMILIAN SÖLCH

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCHOOL OF ENGINEERING SCIENCES

DETECTING ANOMALIES IN ROBOT TIME SERIES DATA USING STOCHASTIC RECURRENT NETWORKS

MAXIMILIAN SÖLCH

MASTER'S THESIS

DEPARTMENT OF MATHEMATICS
TECHNISCHE UNIVERSITÄT MÜNCHEN

DIVISION OF OPTIMIZATION AND SYSTEMS THEORY
DEPARTMENT OF MATHEMATICS
SCHOOL OF ENGINEERING SCIENCES
ROYAL INSTITUTE OF TECHNOLOGY KTH STOCKHOLM

TUM SUPERVISOR:
Prof. Dr. Daniel Cremers

KTH EXAMINER:
Prof. Dr. Xiaoming Hu

TUM ADVISORS:
Prof. Dr. Patrick van der Smagt
Dipl.-Inf. Justin Bayer

KTH SUPERVISOR:
Asst. Prof. Dr. Johan Karlsson

December 1, 2015

Maximilian Sölch: *Detecting Anomalies in Robot Time Series Data using Stochastic Recurrent Networks*

TUM SUPERVISOR:
Prof. Dr. Daniel Cremers

KTH EXAMINER:
Prof. Dr. Xiaoming Hu

TUM ADVISORS:
Prof. Dr. Patrick van der Smagt
Dipl.-Inf. Justin Bayer

KTH SUPERVISOR:
Asst. Prof. Dr. Johan Karlsson

LOCATION:
München

TIME FRAME:
June 1, 2015 through December 1, 2015

DECLARATION

I hereby declare that this thesis is my own work and that no other sources have been used except those clearly indicated and referenced.

München, December 1, 2015

Maximilian Sölch

DETECTING ANOMALIES IN ROBOT TIME SERIES DATA USING STOCHASTIC RECURRENT NETWORKS

MAXIMILIAN SÖLCH

M.SOELCH@TUM.DE
SOLCH@KTH.SE

ABSTRACT

This thesis proposes a novel anomaly detection algorithm for detecting anomalies in high-dimensional, multimodal, real-valued time series data. The approach, requiring no domain knowledge, is based on Stochastic Recurrent Networks (STORNs), a universal distribution approximator for sequential data leveraging the power of Recurrent Neural Networks (RNNs) and Variational Auto-Encoders (VAEs).

The detection algorithm is evaluated on real robot time series data in order to prove that the method robustly detects anomalies off- and on-line.

ANOMALIEDETEKTION IN ROBOTERZEITREIHEN MITTELS STOCHASTISCHER REKURRENTER NETZWERKE

MAXIMILIAN SÖLCH

M.SOELCH@TUM.DE
SOLCH@KTH.SE

ZUSAMMENFASSUNG

In dieser Arbeit wird ein neuartiger Algorithmus entwickelt, um in hochdimensionalen, multimodalen, reellwertigen Zeitreihen Anomalien zu detektieren. Der Ansatz benötigt keine domänenspezifisches Fachwissen und basiert auf Stochastischen Rekurrenten Netzwerken (STORN), einem universellen Wahrscheinlichkeitsverteilungsapproximator für sequenzielle Daten, der die Stärken von Rekurrenten Neuronalen Netzwerken (RNN) und dem Variational Auto-Encoder (VAE) vereinigt.

Der Detektionsalgorithmus wird auf realen Robotertrajektorien evaluiert. Es wird gezeigt, dass Anomalien robust online und offline gefunden werden können.

ANOMALIDETEKTION I ROBOT-TIDSSERIER MED HJÄLP AV STOKASTISKA ÅTERKOMMANDE NÄTVERK

MAXIMILIAN SÖLCH

M.SOELCH@TUM.DE
SOLCH@KTH.SE

SAMMANFATTNING

Detta arbete förslår en ny detektionsalgoritm för anomalier i högdimensionell multimodal reellvärd tidsseriedata. Metoden kräver ingen domänkunskap och baseras på Stochastic Recurrent Networks (STORNs), en teknik för oövervakad och universell fördelningssapproximation för sekventiell data som bygger på Recurrent Neural Networks (RNNs) och Variational Auto-Encoders (VAEs).

Algoritmen utvärderades på robotgenererade tidsserier och slutsatsen är att metoden på ett robust sätt upptäcker anomalier både offline och online.

CONTENTS

ACRONYMS

NN Neural Network

RNN Recurrent Neural Network

VAE Variational Auto-Encoder

STORN Stochastic Recurrent Network

pdf probability density function

cdf cumulative density function

KL Kullback-Leibler divergence

LSTM Long Short-Term Memory

ROC Receiver Operating Characteristic

tpr true positive rate

fpr false positive rate

AUC Area under Curve

PPV Positive Predictive Value

NOTATION

t	time index, $t \in \{1, \ldots, T\}$
d	dimension index, $d \in \{1, \ldots, D\}$
n	sample index, $n \in \{1, \ldots, N\}$
x	scalar quantity (normal font, lower case)
$\mathbf{x}$	vector-valued quantity, $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T \in \mathbb{R}^{d_x}$ (bold font, lower case)
$\mathbf{W}$	matrix-valued quantity (bold font upper case)
$\mathbf{x}_{a:b}$	with $b \geqslant a$: $\mathbf{x}_{a:b} = (\mathbf{x}_a, \mathbf{x}_a + 1, \ldots, \mathbf{x}_b)$, a sequence of $b - a + 1$ steps
$\mathbf{x}_{a:b}$	with $b < a$: short-hand notation for a random variable taking one value with probability 1 (i. e., 'impossible' rather than reversed sequence); usually used when conditioning on previous time steps in the initial state

PREFACE

Anomaly detection is a task that has attracted many researchers from a plethora of different domains. It has been of great interest for both practitioners and theoreticians as a challenging real-world task.

Still, with all progress that has been made, a convincing generic approach to anomaly detection still is to be found. Our intuitive notion of normality turns out to be extremely difficult to define in mathematical terms.

This thesis aims at presenting a framework with the potential for generic anomaly detection for time series data. The framework is based on recent advances in approximate variational inference with the help of Neural Networks (NNs) and Recurrent Neural Networks (RNNs).

Specifically, the goal of this thesis will be to evaluate the potential of Stochastic Recurrent Networks (STORNs), [2], to detect anomalies in robot time series data

To this end, the thesis has been split into three chapters. Chapter 1 will give an introduction to anomaly detection, inspect related work and determine the task at hand. Chapter 2 will then introduce the novel theoretical framework for detecting anomalies. Chapter 3 evaluates experiments conducted to verify and justify the approach.

1 INTRODUCTION

This first chapter focuses on the intricacies of anomalies in the domain of robots, Section 1.1, examine related work, Section 1.2, in order to then specify the task in more detail in Section 1.3.

1.1 WHAT IS ANOMALY DETECTION?

Anomaly detection[1] is a special instance of classification problems. We would like to learn a model that is able to discriminate between 'normal' and 'anomalous' behavior of the robot system. The major difference compared to common (and fairly well-understood) two-class classification problems is an imbalanced data set.

While data of a regularly running system can be recorded with decent effort, data of the second class are much harder and more costly to record: Firstly, one would have to destroy (or at least risk destruction of) the robot system in order to achieve reasonable data. Secondly, the nature of anomalies can be very diverse (or multimodal). While the normal behavior follows distinct, albeit complicated patterns, it is hard to generalize all types of anomalies that can occur.

A situation where the robot is hit accidentally will differ significantly from an anomaly caused by broken or worn-out parts, which again will differ from an anomaly where the robot fails to grasp an item. Moreover, anomalies that were previously unknown may occur. Thus, not only will it be hard to sample all possible classes of anomalies, but collecting sufficiently many examples for learning will be almost impossible.

1.2 RELATED WORK

The problems of anomaly detection outlined above have been examined by many researchers over the past decades. To justify a novel anomaly detection algorithm, we will have a closer look at previous approaches.

A very thorough 2014 review paper [20] investigates the available literature (with over 300 references) and, moreover, extracts five high-level approaches to anomaly detection.

1 Anomaly detection is also often referred to as either novelty detection or outlier detection (cf. [20]). While these can almost be used as synonyms, anomaly detection is most suitable in the context of robots, as we want to detect mostly faulty behavior, which is best coined as an anomaly.

PROBABILISTIC APPROACHES: The probability density of the data assumed to be normal is estimated. Regions of low probability mass are assumed to be anomalies.

While this yields a compact representation of the normal data set (which may be discarded after training), it requires a reasonably high amount of data, in particular when concerned with multivariate data. In this case, the *curse of dimensionality* kicks in: The amount of possible patterns grows exponentially in the number of dimensions. A lot of approaches thus apply extensive preprocessing or assume independence of the components.

Furthermore, a lot of the probabilistic approaches come with a specific assumption on the nature of the data, like fitting a Gaussian Mixture Model to the data.

Time series data is mostly covered by so-called state-space models such as Hidden Markov Models (HMM) or Kalman filters (both being subclasses of Dynamic Bayesian Networks (DBNs)). However, as it is put in [7], "DBNs have typically been limited to either relatively simple state transition structures (e. g., linear models in the case of the Kalman filter) or to relatively simple internal state structure (e. g., the HMM state space consists of a single set of mutually exclusive states)."

DISTANCE-BASED APPROACHES: Methods in this class harvest ideas similar to k-Nearest-Neighbors or other clustering methods, assuming that normal data clusters in space. This is convenient because no knowledge about the underlying data distribution is required.

However, a well-defined distance measure for two data points is required, which is particularly tricky for time series of differing lengths. In essence, the distance measure captures a lot of our understanding of what an anomaly is (and, through the back door, assumptions on the underlying distribution enter the game). This is not always suitable.

If, e. g., an approach assumes anomalies to be sparse in space, it will have a hard time detecting consistent wrong behavior caused by worn-out parts of the robot.

Moreover, distance-based approaches do not scale well with large multivariate data sets, as a lot of comparisons (or good heuristics) are necessary to determine the normality of a new sample.

DOMAIN-BASED APPROACHES: Domain-based solutions take a different approach than probabilistic and distance-based ones. Instead of trying to capture the structure of the data itself, they try to determine the domain of the data, i. e., a boundary around normal data. Anything beyond this boundary is marked an

anomaly. Most prominent within these approaches are Support Vector Machines. The rationale behind these approaches is to try and solve a simpler problem than finding the structure of data, namely discriminating between normal and anomalous data.

The authors of [20] conclude, however, that the domain-based approaches struggle with high-dimensional data. Furthermore, the choice of appropriate parameters (in particular the kernel functions involved in Support Vector Machines) is very hard.

RECONSTRUCTION-BASED APPROACHES: This class of approaches is closely related to nonlinear regression and Neural Networks (NNs). By techniques such as auto-encoding (cf. Section 2.1.3), the network learns the structure of the data autonomously and uses the reconstruction of the data as a measure of normality.

While these auto-encoding networks autonomously extract a lower-dimensional representation, other approaches explicitly map data to subspaces, e. g., by variants of Principal Component Analysis.

Major drawbacks of these approaches are, on the one hand, their sensitivity to the particular choice of (hyper-)parameters, and on the other hand, their inapplicability to time series data.

INFORMATION THEORETIC APPROACHES: Approaches based on Information Theory introduce a different type of measure of normality. The basic assumption is that novel data significantly changes the information content of the normal data. To this end, entropies are calculated either globally or locally and their changes are evaluated in order to find anomalous data.

This already points to the main drawback: The choice of the specific metric is arbitrary and, thus, will not capture all types of anomalies, in particular short term anomalies. From an information theory point of view it is hard to assign any kind of score to single points.

None of the approaches outlined in the review paper seems to tackle robot time series data appropriately.

Thus, we need to come up with a novel algorithm that suits our requirements.

A notable approach is taken in [17]. The authors worked on the same raw data set as this thesis. Their method is based on the assumption that anomalies are sparse and the trajectories are piecewise polynomial. It combines Robust Principal Component Analysis (RPCA) with Group Fused LASSO (Least Absolute Shrinkage and Selection Operator). The idea is that an RPCA-like error term reconstructs the trajectory while Group Fused LASSO inspired regularization terms impose a (grouped) sparsity among the components used

to reconstruct, so that particular parts of a movement can only be reconstructed by certain groups.

While their anomaly detection routine yields good results, the approach has a couple of drawbacks: First and foremost, it is bound to work off-line, as it needs to reconstruct the whole time series to work properly. Secondly, the assumptions on the type of anomaly (sparse) and trajectory (polynomial) are reasonable, yet fairly restrictive.

Comparison of the two methods, though using the same data set, will be hard, as the paper only states results such as true and false positive rates for segmentation imposed by the algorithm itself.

Consequently, a novel approach seems justified.

1.3 PROBLEM SPECIFICATION

In the previous section, we have seen that all approaches to anomaly detection suffer from at least one of the following drawbacks. The approaches are either

- only suitable for and also dependent on specific models (e. g., Kalman filters or distribution assumptions like Gaussian Mixture Models)
- tailored to very specific domain knowledge or anomalies,
- very specific as to how an anomaly manifests itself in the data (e. g., in a sparse region or in a changed information content),
- not suitable for data with high temporal dependencies (as opposed to data streams of very little structure),
- not suitable for multivariate data,
- or computationally infeasible for large data-sets.

With these preconditions in mind, we can specify requirements for our approach.

inherently multivariate data

We will deal with robot data, for a full specification see Section 3.1. Most importantly, the data is multivariate. We will have to deal with inherent dependencies between the sensors. A simple example shows the consequences: For detecting certain anomalies of a robotic arm, one might simply come up with hard-coded thresholds on individual joint torques. This approach, however, neglects the aforementioned dependencies: While in a certain arm configuration, a joint torque value can be perfectly normal, in a different configuration it might be anomalous. In this case, the anomaly may only be detected by considering the torque value in the context of all variates.

real-valued time series data

Moreover, we deal with (real-valued) time series data, as opposed to streams of random sensor data. The major difference is that the temporal structure is important for detecting anomalies. Suppose,

e. g., the robot is fulfilling a task in a sequence of steps. While a movement may be suitable for one of the steps, it can be anomalous in another step. Thus, the temporal structure has to be considered.

complex, multimodal data distribution

Furthermore, a sequence of tasks or steps implies that data points will have different modes. Our approach is thus required to be able to deal with multimodal distributions.

no explicit movement or task model

Such complex distributions are particularly difficult: We would like to find an approach that is able to deal with these distributions without explicitly implementing an underlying model. In fact, we would like to find a sufficiently generic approach that can solve robot time series anomaly detection on a variety of tasks and movements, not just tailored to each and every individual task.

little anomalous data

As mentioned before, data will be imbalanced. In fact, our training data will consist of normal behavior data only. That is to say, our approach has to be unsupervised (or semi-supervised).

on-line detection

Lastly, in the setting of a robot fulfilling a task, any anomaly should be detected as soon as possible. To prevent hazards, we might even require instantaneous detection of some anomalies. In particular, we cannot wait for the whole trial to be finished before we can check our data for anomalies. This means that we require a real-time and on-line capable approach. This limits our methodology, since we can only check data prior to the anomalies. It also limits us to fairly simple, real-time capable calculations (the limitations are given by hardware).

1.4 A NOVEL APPROACH

All examined approaches fall short of at least one, usually even several of the aforementioned essential requirements. Thus, a novel approach to anomaly detection is justified. We will introduce it in Chapter 2 and try to classify it according to the classes in the review paper by Pimentel et al. Chapter 3 will then evaluate our approach against the requirements.

2

BACKGROUND AND THEORY

The present chapter is dedicated to introducing Stochastic Recurrent Networks (STORNs). We will first develop the theory of Variational Auto-Encoders (VAEs) in Section 2.1, which we will use as the basis for STORNs in Section 2.2. Once we have developed theoretical properties, a corresponding computational architecture and a suitable learning algorithm, Section 2.3 will justify STORNs against the requirements specified in Section 1.3.

For this chapter, we assume basic knowledge about properties of Neural Networks (NNs) and Recurrent Neural Networks (RNNs). For a short recap, the reader is referred to Appendix A.1.

2.1 VARIATIONAL AUTO-ENCODER

Our goal is to learn complex nonlinear distributions that capture the structure of our data at hand and efficiently determine the likelihood of new data points. Moreover, we would like to be able to efficiently generate new samples according to the complex distribution. In a first attempt to achieve this, we will neglect any temporal structure of the data.

A recent approach named Variational Auto-Encoder (VAE), developed independently by [21, 14], aims at solving this problem by first sampling from simpler distributions (e.g., Gaussian distributions) and then applying a differentiable, parametrized and thus trainable nonlinear transform. After introducing the mathematical basis, we will examine and explain this approach further.

2.1.1 *Sampling by the Inverse Transform Method*

In theory, sampling can easily be done by the so-called inverse transform method. Assume that you can generate samples from a uniform random variable u on the unit interval $[0, 1]$. Furthermore, assume that you have a random variable z with associated cumulative density function (cdf) F_z. Then, the random variable $F_z^{-1}(u)$ follows the same distribution as z.[1]

In addition, we observe that for any *continuous* distribution of z, the cdf and its inverse cancel and we can conclude that the random variable $F_z(z) = F_z(F_z^{-1}(u)) = u$ follows a uniform distribution on

1 Note that, depending on the type of the random variable z, e.g., in the case of a discrete variable, the cdf F_z may not be invertible in closed form. In this case, we define $F_z^{-1}(u) := \inf\{y \mid F_z(y) \geqslant u\}$.

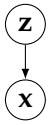


Figure 1: Probabilistic graphical model of a Variational Auto-Encoder (VAE). It expresses the conditional dependence between the observed or desired variable $\mathbf{x}$, and a latent variable $\mathbf{z}$. The graphical model shows how to factorize the joint distribution: $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})$.

the standard interval. Merging these two observations, we can generate samples of a random variable x with associated cdf F_x by first drawing a sample from an arbitrary *continuous* random variable z and then transforming $x := F_x^{-1}(F_z(z))$. Usually, z follows a simpler distribution than x in order to justify the computational overhead. All these results can also be established for vector-valued variables $\mathbf{x}$ and $\mathbf{z}$.

However, in practice we run into a plethora of problems. The most prominent problem is that for most complicated distributions the cdf is not available in closed form. The same applies for its inverse. In fact, even the inverse cdf of the Gaussian distribution can only be approximated numerically.

2.1.2 *Graphical Model Perspective*

Sampling by the inversion method can also be viewed from a different perspective: as a special case of graphical models.

Graphical models use graph theory in order to depict dependency structures among random variables. The core assumption is that, given a directed graph $\mathcal{G}$ where each node corresponds to a random variable or vector $\mathbf{x}_i, i = 1, \ldots, N$, one can efficiently write the joint distribution as

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{i=1}^{N} p(\mathbf{x}_i \mid \mathbf{PA}_i). \tag{1}$$

Here, $\mathbf{PA}_i$ denotes the set of all variables that correspond to parent nodes of the node $\mathbf{x}_i$ in $\mathcal{G}$. [2]

The core idea is to represent the joint probability with the simplest possible conditional distributions.

For a more thorough introduction to graphical models, cf. [18]. In the context of this thesis, we will explicitly model the dependencies according to graphical models. In the spirit of the inverse transform, some of our conditional distributions will be deterministic: We

2 Informally, one can think of the parent variables influencing their children, although, in principle, the probabilistic dependency works both ways. In general, extracting *causal* structures is a field of ongoing research, cf. [19].

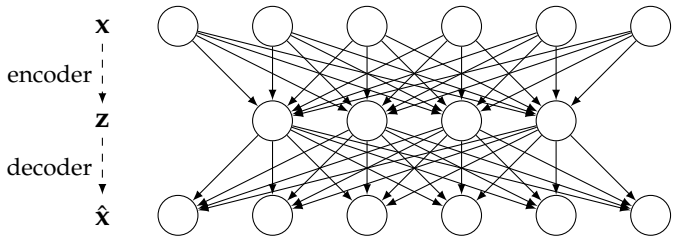


Figure 2: Example of a Deterministic Auto-Encoder with six input units and consequently also six output units, as well as four latent units. One can see the typical bottleneck to prevent the auto-encoder from simply passing the input from top to bottom.

start of from a simple distribution $p(\mathbf{z})$ and model $p(\mathbf{x} \mid \mathbf{z})$ explicitly through an approximation of the inverse transform. Then, we can use the fact that

$$p(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) \, d\mathbf{z} = \int_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \, d\mathbf{z}. \tag{2}$$

By integrating out $\mathbf{z}$, the marginal density $p(\mathbf{x})$ becomes stochastic, even if our model $p(\mathbf{x} \mid \mathbf{z})$ was deterministic.

The admittedly simple graph corresponding to the inverse transform is depicted in Fig. (1). The graphical model view will be particularly useful when introducing time to our model in Section 2.2.

2.1.3 *Auto-Encoding*

Besides the inverse transform view and the graphical model view, one can interpret the relation between the simpler $\mathbf{z}$ and the complex $\mathbf{x}$ in a third way: $\mathbf{z}$ encodes a latent structure found in the observed variables $\mathbf{x}$.

This has been studied in the deterministic case: So-called auto-encoders (cf. [18, p.1000f]) learn to find reasonable representations $\mathbf{z}$ by simultaneously learning two Neural Networks (NNs). The first network encodes a specific $\mathbf{x}$ in a corresponding $\mathbf{z}$, while the second network takes this $\mathbf{z}$ as input and reconstructs (i. e., decodes) an approximation $\hat{\mathbf{x}}$, thus the name auto-encoder. The parameters of the two networks can, e. g., be trained simultaneously by minimizing the squared error between $\mathbf{x}$ and $\hat{\mathbf{x}}$.

The dimension of $\mathbf{z}$ is smaller than that of $\mathbf{x}$ in order to create a bottleneck and to prevent the auto-encoder from learning the identity transform as both encoder and decoder. If the reconstruction error is minimized, one assumes to have found a reasonable code $\mathbf{z}$.

Fig. (2) depicts an example of a deterministic auto-encoder with the typical bottleneck in the latent space.

According to [18], linear auto-encoders, i. e., those using NNs without nonlinear transfer functions, are provably equivalent to Principal

Component Analysis (PCA), an orthogonal transformation onto the most prominent eigenvectors. Nonlinear auto-encoders can find nonlinear structures in the data, but suffer from vanishing gradients and poor local minima, cf. [6].

2.1.4 *Merging the Ideas: Variational Auto-Encoder*

The Variational Auto-Encoder aims at combining the deterministic auto-encoder mechanism with the stochastic inversion method.

The deterministic bottleneck layer $\mathbf{z}$ of the auto-encoder is replaced by stochastic units with. By doing so, we do not simply get a specific code $\mathbf{z}$ per sample of $\mathbf{x}$, but a probability distribution of codes. From this distribution, we then compute a distribution of the complex data $\mathbf{x}$ as a nonlinear transform (or decoding) of $\mathbf{z}$. This nonlinear transform is a differentiable approximation of the mapping $F_{\mathbf{x}}^{-1} \circ F_{\mathbf{z}}$ from the inversion method. In the multivariate case, we can extend this approach: The inversion method requires vectors of the same dimension all along. Here, we explicitly allow a different dimensionality—we expect that our high-dimensional data is tightly coupled and generated by a latent structure of different dimension. We will refer to the decoding, approximate inverse transform part as the *generative model*.

The upper, encoding part will be denoted as the *recognition model*. It is necessary for two reasons. Firstly, we have no ground truth about latent states $\mathbf{z}$, just observed data. Hence we need the trainable recognition model to provide the generative model with input. Secondly, the final model will not only be used for generating new samples (for which the generative model would be sufficient), but also for evaluating the likelihood. In this case, we will get an observation $\mathbf{x}$ and need the corresponding code distribution over $\mathbf{z}$. This will be provided by the recognition model.

The approximation of the nonlinear transforms involved can be done via Neural Networks (NNs). They are particularly suited because of their universal approximation capability (as proven by [15, 12, 9], cf. Appendix A.1), which is of particular interest since the wanted transform is usually not available in closed form (even the inverse transform of Gaussian variables cannot be done in closed form). This has a very interesting theoretical consequence: VAEs are *universal distribution approximators*. The inverse transform method is universal for all distributions, and because the universal approximation capability of NNs does not impose restrictions on the non-linear transform applied, the particular distribution transform $F_{\mathbf{x}}^{-1} \circ F_{\mathbf{z}}$ can be approximated arbitrarily well with a sufficiently large NN.

Nevertheless, it should be noted that the approach is not exclusively bound to NNs. Any function approximation method, e. g., Gaussian Processes, can replace them.

It is worth noting that, other than the deterministic auto-encoder, the Variational Auto-Encoder does not necessarily require a dimensionality reduction in the latent layer: A given $\mathbf{x}$ is mapped to a *distribution* $p(\mathbf{z} \mid \mathbf{x})$; it does not suffice that these latent distributions differ numerically. In the decoding process of the auto-encoder, we will draw a single sample $\mathbf{z}$ from $p(\mathbf{z} \mid \mathbf{x})$, which imposes noise on the latent representations. This means that the latent distribution $p(\mathbf{z} \mid \mathbf{x}_1)$ and $p(\mathbf{z} \mid \mathbf{x}_2)$ of two points $\mathbf{x}_1$ and $\mathbf{x}_2$ may differ, but still produce the same sample $\mathbf{z}$. For the two points to be distinguishable, their latent distribution must share little or no probability mass.

This observation can be called a 'soft'or 'differential'discretization of the latent space. The discretization is not hard-coded (by, e. g., only allowing integers), but it limits the ability to encode information even in high-dimensional latent spaces.

2.1.5 *Energy Function: Variational Lower Bound*

In order to apply Neural Networks (NNs), we will have to find an energy function that captures how well the data is explained by the distribution that the NN approximates. A good, interpretable energy function circumvents negative properties of the deterministic auto-encoder.

To this end, let now $\mathbf{x}$ represent the vector-valued input, e.g., our data. We are interested in the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$, an optimal code distribution given the input. However, it is not easily assessable. Hence, we approximate it by a recognition model, which is parametrized by the parameter set ϕ and represented by the distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$ [3]. Note that this means that we will not get a deterministic code $\mathbf{z}$ for a given $\mathbf{x}$, but a probability distribution. Our goal will be to find q_ϕ such that it closely resembles the true posterior.

We can analyze the unknown data likelihood $p(\mathbf{x})$. We are interested in finding a model that maximizes the likelihood of our:

$$\ln p(\mathbf{x}) = \ln p(\mathbf{x}) \underbrace{\int q_\phi(\mathbf{z} \mid \mathbf{x}) \, d\mathbf{z}}_{=1} = \int q_\phi(\mathbf{z} \mid \mathbf{x}) \ln p(\mathbf{x}) \, d\mathbf{z} \tag{3}$$

$$= \int q_\phi(\mathbf{z} \mid \mathbf{x}) \ln \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})} \, d\mathbf{z} \tag{4}$$

$$= \int q_\phi(\mathbf{z} \mid \mathbf{x}) \ln \frac{p(\mathbf{x}, \mathbf{z}) q_\phi(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z} \mid \mathbf{x}) q_\phi(\mathbf{z} \mid \mathbf{x})} \, d\mathbf{z} \tag{5}$$

$$= \underbrace{\int q_\phi(\mathbf{z} \mid \mathbf{x}) \ln \frac{p(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} \, d\mathbf{z}}_{=: \mathcal{L}_{\text{VAE}}(q(\mathbf{z}))} + \underbrace{\int q_\phi(\mathbf{z} \mid \mathbf{x}) \ln \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z} \mid \mathbf{x})} \, d\mathbf{z}}_{= \mathrm{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}|\mathbf{x})\right)} \tag{6}$$

3 If it is clear from the context, we will use ϕ or q_ϕ to identify the whole recognition model, not just the parameters or the distribution

In the first and the third line we conveniently multiplied by 1, in the second line we used that $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z} \mid \mathbf{x})p(\mathbf{x})$.

Kullback-Leibler divergence

The term $\mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})\big)$ denotes the Kullback-Leibler divergence (KL) between our learned model q_ϕ and the wanted but unknown posterior distribution $p(\mathbf{z} \mid \mathbf{x})$ on the codes. For given densities f and g with shared support, it is defined as

$$\mathrm{KL}(f \parallel g) = \int f(\mathbf{x}) \ln \frac{f(\mathbf{x})}{g(\mathbf{x})} \, d\mathbf{x}. \tag{7}$$

Though it fails to be symmetric, the KL is a measure of dissimilarity between the densities f and g (or, more precisely, the difference in information content of g from f). If they are very close to each other, the argument of the logarithm is close to 1 and it evaluates to values around 0. Conversely, if f has high mass where g has mass close to 0, the logarithm's argument evaluates to extremely high values, leading to an increased KL.

The KL, also of great importance in information theory, is closely related to concepts like Shannon entropy. Intuitively speaking, it measures the (expected) amount of extra bits needed to encode $\mathbf{x}$ sampled from g with a code optimized for f.[4]

With this in mind, remember that our goal was approximating the posterior with the recognition model distribution q_ϕ. For example, we might like to choose

$$q^*(\mathbf{z}) = \arg\min_{q_\phi} \mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})\big). \tag{8}$$

Yet again, we run into the problem that we have no explicit knowledge about the posterior. Nevertheless, we can now exploit two facts.

On the one hand, the left-hand side of Eq. (6), the data log-likelihood, is constant (though unknown) given ϕ.

On the other hand, the KL is nonnegative:

$$\begin{aligned} \mathrm{KL}(f \parallel g) = \int f(\mathbf{x}) \ln \frac{f(\mathbf{x})}{g(\mathbf{x})} \, d\mathbf{x} &= -\mathbb{E}_f \left[\ln \frac{g(\mathbf{x})}{f(\mathbf{x})} \right] && (9) \\ &\geqslant -\ln \mathbb{E}_f \left[\frac{g(\mathbf{x})}{f(\mathbf{x})} \right] = -\ln \int f(\mathbf{x}) \frac{g(\mathbf{x})}{f(\mathbf{x})} \, d\mathbf{x} && (10) \\ &= -\ln \int g(\mathbf{x}) \, d\mathbf{x} = 0 && (11) \end{aligned}$$

From first to second line, we used Jensen's inequality, which can be applied from right to left because the negative logarithm is convex and g/f is integrable on the support of f and g (which we assume to

4 Consider the Morse code for the English language. If words of a different language are encoded in this code, they will be longer than they would be in a Morse code optimized for this particular language. The expected amount of extra bits (corresponding to one signal in Morse) is measured by the KL between the letter frequencies in the two languages.

be equal) as a ratio of densities. In the last step, we used the fact that g is a probability density and thus integrates to 1.

Given these two facts, let us reconsider Eq. (6). Because the left-hand side is constant w.r.t. ϕ, we see that the two ultimate summands are coupled and thus

$$\arg\max_{q_\phi} \mathcal{L}_{\text{VAE}}\big(q_\phi(\mathbf{z} \mid \mathbf{x})\big) = \arg\min_{q_\phi} \text{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})\big). \tag{12}$$

On the right-hand side, we recognize our desired, yet intractable minimization task from Eq. (8). Fortunately, it can now be replaced with a maximization task, which will turn out to be comparably simple.

Moreover, since the KL is nonnegative, it follows that

$$\mathcal{L}_{\text{VAE}}\big(q_\phi(\mathbf{z} \mid \mathbf{x})\big) \leqslant \ln p(\mathbf{x}). \tag{13}$$

The term $\mathcal{L}_{\text{VAE}}\big(q_\phi(\mathbf{z} \mid \mathbf{x})\big)$ is thus called the *variational lower bound*.

Since we would like to have an optimal, yet intractable $p(\mathbf{x})$, we can instead maximize the lower bound and indirectly improve $p(\mathbf{x})$. Simultaneously, due to our finding in Eq. (12), we achieve a good approximation of the posterior on the latent code $p(\mathbf{z} \mid \mathbf{x})$ on the fly. [5]

Consequently, we should have a closer look at the lower bound. To this end, we will need to introduce a prior $p(\mathbf{z})$ on the codes, which is independent of the input. Now:

$$\begin{aligned}
\mathcal{L}_{\text{VAE}}(q_\phi) &= \int q_\phi(\mathbf{z} \mid \mathbf{x}) \ln \frac{p(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} \, d\mathbf{z} && (14)\\
&= \int q_\phi(\mathbf{z} \mid \mathbf{x}) \ln \frac{p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} \, d\mathbf{z} && (15)\\
&= \int q_\phi(\mathbf{z} \mid \mathbf{x}) \ln p(\mathbf{x} \mid \mathbf{z}) \, d\mathbf{z} - \int q_\phi(\mathbf{z} \mid \mathbf{x}) \ln \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})} \, d\mathbf{z} && (16)\\
&= \mathbb{E}_{q_\phi}[\ln p(\mathbf{x} \mid \mathbf{z})] - \text{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})\big) && (17)
\end{aligned}$$

Again, we used the multiplication rule, as well as a rephrasing according to logarithm rules.

This form turns out to be very useful. If we interpret $p(\mathbf{x} \mid \mathbf{z})$ from the first term as a parametrized generative model $p_\theta(\mathbf{x} \mid \mathbf{z})$, we can install and train any parametrized model. Given this, we can interpret this expected value as a probabilistic analogue of the reconstruction error that would be used in the optimization of a normal auto-encoder. Good reconstruction is reflected by a high expected likelihood. Since $\mathbf{z}$ is distributed according to the recognition model distribution q_ϕ, we can use Monte Carlo and stochastic gradient methods to determine this part of the loss.

5 In Appendix A.2, we will derive a Monte Carlo estimate for the true data likelihood $p(\mathbf{x})$, which is useful for evaluating the tightness of the lower bound.

The second term is also tractable: The recognition model distribution q_ϕ is now compared to the fixed prior $p(\mathbf{z})$. This KL and its derivative can be evaluated in closed form if the distributions are of a simple form, such as Gaussian or Laplace distributions.[6]

The essential trick is that we can still calculate the derivative with respect to the parameters ϕ. For Gaussian distributions, this is possible, because if $q_\phi \sim \mathcal{N}(\mu, \sigma^2)$, then we can draw $\epsilon \sim \mathcal{N}(0, 1)$ and reparametrize $z = \mu + \sigma\epsilon$. The white noise ϵ can be viewed as an additional input to our model and we can take the derivative w.r.t. ϕ without having to propagate through a stochastic variable. This key trick to VAEs is thus called *reparametrization trick*.

It is also important to notice that the two summands of the lower bound interact. The reconstruction term would like to optimize q_ϕ such that the data likelihood is maximized. This is most easily done by fitting perfect code distributions with very little variance to the data. This leads to very good reconstructions. However, all information extracted from data during the learning process is encoded in the recognition model and the generative model bears very little information. In particular, sampling results will be poor.

In contrast, the KL divergence regularizes q_ϕ such that it does not differ too much from our prior. In other words: The loss function implicitly tries to find a trade-off between a good reconstruction and a simple latent structure. The simplicity in the latent structure forces the learning algorithm to encode information in the generative model, since it cannot manipulate the latent code distributions arbitrarily.

Moreover, the recognition model ϕ and the generative model θ are trained simultaneously through one stochastic gradient descent step.

Starting from an intractable data log-likelihood, we have gained at least three desirable outcomes:

1. We have an approximate, implicit representation of $p(\mathbf{x})$, which has provable universal approximation capabilities. It will allow for probabilistic anomaly detection on this distribution.

2. Not only can we access $p(\mathbf{x})$ indirectly, but we can create new and unknown samples following the approximate distribution.

3. We have found a differentiable and well interpretable compression technique.

Note that our derivation used no assumptions other than our ability to sample from either $p(\mathbf{z})$ or $q_\phi(\mathbf{z} \mid \mathbf{x})$. The only restriction so far is that we optimize a lower bound. Nevertheless, the theory allows that a good model choice and good optimization techniques in Eq. (12) can close the gap arbitrarily well.

6 Otherwise, if necessary, it can be sampled and estimated efficiently by Monte Carlo methods: The KL is an expected value. We need to draw samples from q_ϕ, which is easily possible by construction, and evaluate both q_ϕ and the prior $p(\mathbf{z})$, which is also easily possible because we can control both by our model choice.

2.1.6 *Architecture and Computation*

To understand the model better, we will now have a look at the explicit architecture to implement the theory derived above.

In Fig. (3), we show one possible architecture of the VAE, as suggested by [14, 21]. It consists of two main parts. On the left-hand side, the teal colored nodes represent a fully-connected, top-down Feed-Forward Neural Network (NN). This implements our recognition model ϕ. Its input is data, depicted by the green box and inputting to the blue input nodes, and the output produced are the sufficient statistics of the distribution q_ϕ, depicted by cyan nodes. Note that in theory, we could choose any q_ϕ. In practice, we need to evaluate the KL with the prior, so that we will restrict ourselves to, e. g., Gaussian distributions. In this case, the output would be, e. g., a set of tuples, each of which holds mean and covariance for a scalar component of $\mathbf{z}$, which would mean that we assume a latent diagonal Gaussian distribution structure. Similarly, one can imagine different structures, such as correlated Gaussian variables or completely different distributions. It is important to notice that decoupled latents $\mathbf{z}_{1:T}$ do *not* imply that the observed $\mathbf{x}_{1:T}$ are decoupled as well.

Throughout this thesis, we will consider diagonal Gaussian distributions only, with the exception of Appendix A.3, where we tested uniform distributions, but found them to contribute no improvement.

During the training procedure, we aim at reconstructing our data. To this end, we sample from the distribution fixed by the output of the first neural network. The sample (yellow nodes; the rectangular shape indicates stochasticity) is fed as input to the second neural network, which is depicted on the right-hand side as a bottom-up, fully-connected feed-forward neural network (orange nodes). The output of this are sufficient statistics of the generative output distribution $p_\theta(\mathbf{x} \mid \mathbf{z})$ (violet nodes). Note that the reconstruction sampled from the output distribution is not deterministic. Instead, this probability distribution leaves room for modeling of noise that cannot be explained by the nonlinear generative model, e.g., sensor noise.

The lower bound function defined in Eq. (17) will serve as our loss function. Its two components are depicted as red diamonds.

Due to the simultaneous training of recognition and generative model, after learning is completed, the generative model θ can be used independently for sampling artificial data that follows the same distribution as our ground-truth data. Instead of using samples from the recognition model distribution q_ϕ, we now use samples from the prior $p(\mathbf{z})$ (which is why both are depicted in cyan). Notice that the error function punishes divergence between these two sampling distributions, so that in a well-trained model, this should ideally not make much of a difference. This generative sampling process corre-

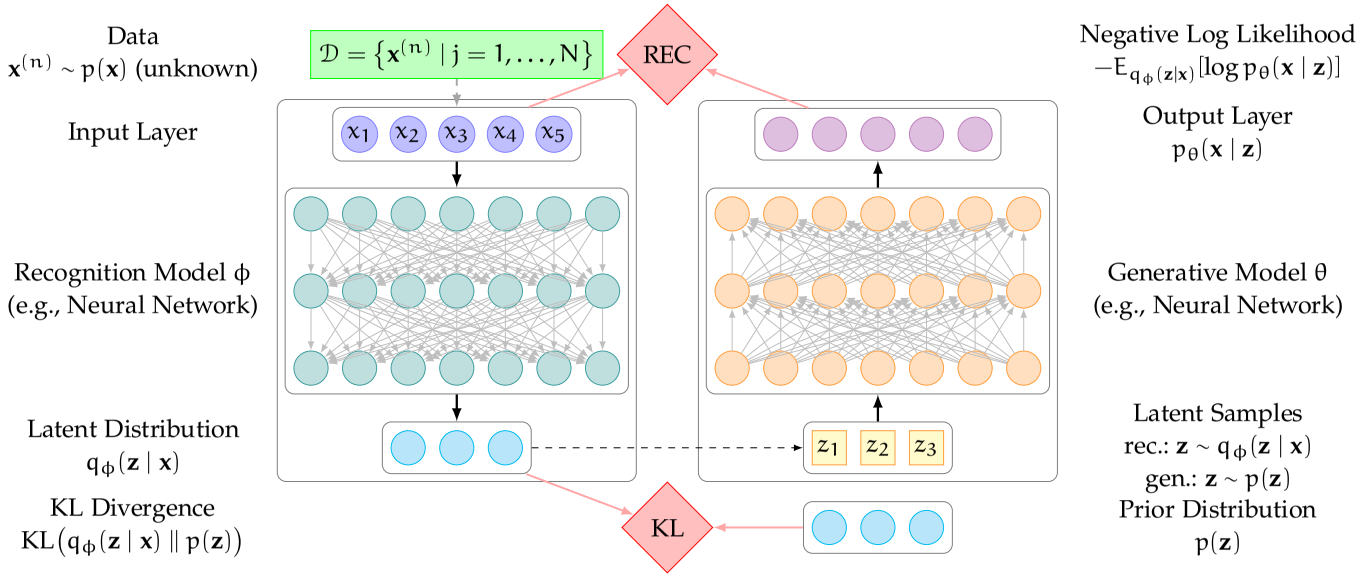


Figure 3: Computational architecture of a Variational Auto-Encoder (VAE). The left part shows the recognition model or encoder. It is parametrized by ϕ, in this case the weights of a fully connected Feed-Forward Neural Network (NN). Its output fully characterizes the distribution q_ϕ (e. g., mean and covariance of a Gaussian distribution). The right part shows the generative model θ, implemented by a fully-connected Feed-Forward NN. It is fed with samples (indicated by square nodes and dashed lines) from q_ϕ (in case of training/reconstruction) or a prior $p(\mathbf{z})$ (in case of sampling). The red diamonds correspond to the two summands of the energy term. Computational flow is in direction of the arrows, backpropagation of the gradient against the direction of the arrows. Input, output or latent dimension as well as the NN structures may deviate from this figure and need not be symmetric.
The figure only depicts computational flow during training or reconstruction. For purely generative sampling, only the generative model on the right side is necessary and fed with samples from the prior.

sponds to using only the full right-hand side in Fig. (3), including the cyan-printed prior distribution.

The difference between q_ϕ and $p(\mathbf{z})$ can mostly be seen in regions where the prior has significantly more mass than the recognition model distribution. Samples from these regions correspond to latent codes $\mathbf{z}$ that, after the generative transformation through θ, have potentially no resemblance to any point in our data set. It remains to evaluate whether this effect becomes negligible after sufficiently thorough training.

2.1.7 *Conclusion*

In this first part of Chapter 2, we have introduced the so-called Variational Auto-Encoder (VAE). With its help, we have derived a tractable and readily interpretable lower bound $\mathcal{L}_{\text{VAE}}(q_\phi)$ of the data log-likelihood. Upon maximizing, we get a probabilistic encoder and decoder that maximizes the likelihood of our data. We have thus found a probabilistic compression algorithm that tries to find a trade-off between simplicity of the latent structure and precise reconstruction, which should be more robust to common phenomena such as overfitting.

Moreover, we have introduced the computational architecture to exploit these theoretical results.

One drawback of the VAE so far is that it is bound to fixed-size input. This can be overcome by replacing the Feed-Forward NNs by RNNs, which yields so-called STORNs, as first introduced by [2]. The details will be described in Section 2.2.

2.2 STOCHASTIC RECURRENT NETWORKS

In Section 2.1 we have introduced the Variational Auto-Encoder (VAE), a powerful model for approximating nonstandard, high-dimensional probability distributions. In order to apply this model to robot data, we would like to generalize the concept of VAEs to time series data.

In [2], this is achieved by replacing the two NNs of the VAE, recognition and generative model respectively, by RNNs. This model is called a Stochastic Recurrent Network (STORN).

This section is dedicated to formally introducing and justifying STORNs, much like we previously did with the VAE. Section 2.3 will be comparing STORN against the model requirements of our task as lined out in Section 1.3

2.2.1 *Modeling distributions with Recurrent Neural Networks*

Previously, we have seen that NNs are a powerful tool: Due to the universal approximation capability, we are able to model any trans-

form of distributions via an approximation of the inverse transform $F_x^{-1} \circ F_z$ (plus potential additional nonlinear transforms).

Recurrent Neural Networks (RNNs) can be seen as extensions of Neural Networks (NNs). The approximation capability is inherited by RNNs: They were shown to be Turing-complete, cf. [25], i. e., informally, any computation a computer can do can be expressed through a RNN. Moreover, they are able to approximate any measurable mapping of time series onto one another under very mild conditions on input and output, cf. [8].

Using RNNs for distribution representation is a common task. By applying the multiplication rule, one may write[7]

$$p(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}), \tag{18}$$

which fits the architecture of a common RNN: Time step t is processed from the information extracted from all previous time steps 1 through $t-1$. The network now has to model the conditional distributions $p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1})$. This can be done in numerous ways.

In a simple setting, one may apply naïve Bayes, i. e., assuming independence of the components of the random vector $\mathbf{x}_t$ given all previous time steps $\mathbf{x}_{1:t-1}$:

$$p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}) = \prod_{d=1}^{D} p(x_{t,d} \mid \mathbf{x}_{1:t-1}) \tag{19}$$

While this only implies conditional independence of the components, it is still a strong assumption, in particular with our high-dimensional robot data.

As an example, consider a moving robotic arm: Given the previous trajectory $\mathbf{x}_{1:t-1}$, one can, e. g., extrapolate the next points linearly. This would be fully captured by naïve Bayes: The d-th component $x_{t,d}$ is an extrapolation of the d-th component of the previous time steps.

Naïve Bayes is suboptimal, though, in the following scenario: Consider a turning point in the trajectory with multiple options to continue. Rather than choosing one direction to continue and applying it to all components, naïve Bayes would pick one direction for each component, leading to samples that are actually implausible or impossible.

Coupling the components at time step t has been a field of research for the past years. As described in [2], all of these approaches suffer from one or several of the following drawbacks:

7 $\mathbf{x}_{1:0} = \Omega$, the underlying probability space, which is not of further interest for our needs. This convention will be used throughout this thesis without further mention to facilitate notation.

- The model scales at least linearly, which is inconvenient for high-dimensional data-sets.
- The restrictions on the distributions $p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1})$ are strong.
- Costly computations such as Markov Chain Monte Carlo steps are required.

VAEs suffer from none of these drawbacks. Thus, we seek to enhance their power on static data to time series data.

2.2.2 *STORN: Recurrent Nets and the Variational Auto-Encoder*

The VAE introduced a latent structure $\mathbf{z}$ into our model $p(\mathbf{x})$ of the observable variables $\mathbf{x}$. We will proceed analogously in the case of time series data, though we need some additional arguments to end up with a similar generative model as for the VAE.

Starting from Eq. (18), the law of total probability allows us to rewrite the likelihood $p(\mathbf{x}_{1:T})$ of our time series as

$$p(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}) \tag{20}$$

$$= \int_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}) \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}, \mathbf{z}_{t+1:T}) \, d\mathbf{z}_{1:T}. \tag{21}$$

Much like with the VAE, we have introduced a latent structure $\mathbf{z}_{1:T}$ that will allow us to leverage the power of RNNs. For now, we assume that we can handle $p(\mathbf{z}_{1:T})$ and focus on the temporal transition terms of type $p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}, \mathbf{z}_{t+1:T})$. It is this term that has to be modeled (or, more precisely, variationally approximated) by an RNN.

transition term

This step, however, is not as straightforward as before: In the VAE case, $\mathbf{x}$ was only conditioned on one $\mathbf{z}$, so that we could model $p(\mathbf{x} \mid \mathbf{z})$ through a NN that learns the random variable transformation from $\mathbf{z}$ onto $\mathbf{x}$. With the new temporal structure, not only is $\mathbf{x}_t$ conditioned on multiple latent variables $\mathbf{z}_{1:T}$, but furthermore also on $\mathbf{x}_{1:t-1}$.

At this point, further statistical analysis depends on the choice of our model to approximate the transition terms (or vice versa, our choice of model depends on statistical choices). RNNs exist in a plethora of different architectures. For simplicity, we will stick to the following recursive architecture with nonlinearities $f_{\mathbf{h}}$ and $f_{\mathbf{y}}$ and for $t = 1, \ldots, T$:

$$\mathbf{h}_0 = \mathbf{b}_{\text{init}} \tag{22}$$

$$\mathbf{h}_t = f_{\mathbf{h}}\Big(\mathbf{x}_{t-1}\mathbf{W}^{(g)}_{\text{in}} + \mathbf{z}_t\bar{\mathbf{W}}^{(g)}_{\text{in}} + \mathbf{h}_{t-1}\mathbf{W}_{\text{recurr}} + \mathbf{b}_{\text{hidden}}\Big), \tag{23}$$

$$\mathbf{y}_t = f_{\mathbf{y}}(\mathbf{h}_t\mathbf{W}_{\text{out}} + \mathbf{b}_{\text{out}}) \tag{24}$$

If we interpret the latent variable $\mathbf{z}_t$ as an additional input to the transfer function, these equations form a simple RNN with one hidden

layer. The outputs $\mathbf{y}_t$ represent sufficient statistics of the temporal transition distribution:

$$p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}, \mathbf{z}_{t+1:T}) = p(\mathbf{x}_t \mid \mathbf{y}_t) \tag{25}$$

The weight matrices and the biases form the set of parameters[8]

$$\theta = \left\{ \mathbf{W}_{\text{in}}^{(g)}, \bar{\mathbf{W}}_{\text{in}}^{(g)}, \mathbf{W}_{\text{recurr}}, \mathbf{W}_{\text{out}}, \mathbf{b}_{\text{init}}, \mathbf{b}_{\text{hidden}}, \mathbf{b}_{\text{out}} \right\}. \tag{26}$$

It should be noted, though, that the extension to richer RNN architectures (e. g., Long Short-Term Memory (LSTM) RNN, cf. Appendix A.1.2) can be done with reasonable effort as long as the probabilistic dependency structure of the temporal transition in Eq. (21) is preserved. This prohibits, e. g., bidirectional RNNs, which can go back and forth in time.

Of course, we can also apply deeper architecture in terms of layers. Since it would mostly lead to notational overhead rather than theoretical insight, it will be left out in this analysis (and applied in the experiments).

Statistically, the model assumptions (23) and (24) imply two effects. Firstly, we neglect any effect of future latent on present observable variables due to the recursive structure of the equations. For our transition distribution, this means

$$p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}, \cancel{\mathbf{z}_{t+1:T}}) = p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}). \tag{27}$$

This seems a reasonable choice, in accordance with our common perception of 'no present effects from future causes'. Nevertheless, it is only an assumption of statistical independence—in this case a very subtle difference: Reconsider the robot trajectory with a turning point and multiple potential trajectories to continue. Knowledge about the future trajectory (or its latent cause) will affect our prediction for the transition at the turning point.

(Right now, we are only considering the *generative* model. We will later use the observation about useful knowledge of the future to install different *recognition* model architectures. Depending on the concrete application of STORN, e. g., bidirectional RNNs are allowed or disallowed.)

Secondly, we assume some Markovian property: Given the hidden state $\mathbf{h}_t$, $\mathbf{x}_t$ (or rather its distribution, cf. Eq. (27)) is fully determined and independent of $\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}$, and also $\mathbf{h}_{0:t-1}$. This assumption is mostly justified by the simplicity of our model. If necessary, the assumption can be weakened by using a richer model. A notable example are LSTM networks.

8 The choice of θ as notation is motivated by the notation in Section 2.1.

For further analysis, we will again switch back to the purely statistical point of view. Starting from Eq. (27), we introduce the hidden states $\mathbf{h}_{0:t}$ by applying the (conditional) law of total probability:

$$p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}) \tag{28}$$

$$= \int_{\mathbf{h}_{0:t}} p(\mathbf{x}_t \mid \underbrace{\cancel{\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}, \mathbf{h}_{0:t}}}_{\mathbf{h}_t}) p(\mathbf{h}_{0:t} \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}) \, d\mathbf{h}_{0:t} \tag{29}$$

The introduction of $\mathbf{h}_{0:t}$ allows us to use the Markov property introduced by Eq. (23) and Eq. (24) at the cost of introducing a new distribution. Consider this new conditional distribution of the hidden states $\mathbf{h}_{0:t}$ given the latent variables $\mathbf{z}_{1:t}$ and data $\mathbf{x}_{1:t-1}$:

$$p(\mathbf{h}_{0:t} \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}) = \prod_{s=0}^{t} p(\mathbf{h}_s \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}, \mathbf{h}_{0:s-1}) \tag{30}$$

$$= p(\mathbf{h}_0) \prod_{s=1}^{t} p(\mathbf{h}_s \mid \mathbf{x}_{s-1}, \mathbf{z}_s, \mathbf{h}_{s-1}) \tag{31}$$

The first line is once again the multiplication law. The second line stems from the structural equation Eq. (23): $\mathbf{h}_s$ solely depends on $\mathbf{x}_{s-1}$, $\mathbf{z}_s$ and $\mathbf{h}_{s-1}$, with the special case $\mathbf{h}_0$ as in Eq. (22). In fact, the dependency is deterministic: given all three conditional variables, $\mathbf{h}_s$ is uniquely determined.

Putting the pieces together, we can rewrite the transition probability as

$$p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}) \tag{32}$$

$$= \int_{\mathbf{h}_{0:t}} p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}, \mathbf{h}_{0:t}) p(\mathbf{h}_0) \prod_{s=1}^{t} p(\mathbf{h}_s \mid \mathbf{x}_{s-1}, \mathbf{z}_s, \mathbf{h}_{s-1}) \, d\mathbf{h}_{0:t}. \tag{33}$$

Despite the cross-dependencies over time, we can solve integral (33) starting from $\mathbf{h}_0$. Since $p(\mathbf{h}_0)$ is, from Eq. (22), a Dirac measure, integrating out $\mathbf{h}_0$ will turn every occurrence into a deterministic value, which is inserted into $p(\mathbf{h}_1 \mid \mathbf{z}_1, \mathbf{h}_0)$, the only remaining term depending on $\mathbf{h}_0$. This way, we entirely eliminated the integral w.r.t. $\mathbf{h}_0$. But with the random variable $\mathbf{h}_0$ being replaced by a deterministic value, we can now apply the same arguments to $p(\mathbf{h}_1 \mid \mathbf{z}_1, \mathbf{h}_0)$, and subsequently to the whole product.

In the end, we are left with

$$p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}) = p(\mathbf{x}_t \mid \mathbf{h}_t(\mathbf{x}_{t-1}, \mathbf{z}_t, \mathbf{h}_{t-1})). \tag{34}$$

Note that this is in fact a recursive notation. The term $\mathbf{h}_{t-1}$, e. g., is itself a function evaluation $\mathbf{h}_{t-1}(\mathbf{x}_{t-2}, \mathbf{z}_{t-1}, \mathbf{h}_{t-2})$ at the previous time step. Furthermore, $\mathbf{h}_t$ in Eq. (29) refers to a random variable, as opposed to a deterministic value in Eq. (34).

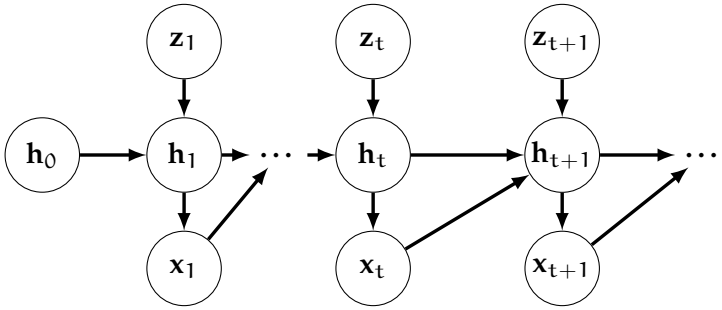


Figure 4: Probabilistic graphical model of a Stochastic Recurrent Network (STORN).

In total, we end up with the following model:

$$p(\mathbf{x}_{1:T}) = \int_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}) \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:T}) \, d\mathbf{z}_{1:T} \tag{35}$$

$$= \int_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}) \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{h}_t(\mathbf{x}_{t-1}, \mathbf{z}_t, \mathbf{h}_{t-1})) \, d\mathbf{z}_{1:T} \tag{36}$$

So far, we have ignored the prior distribution of the latent variables $\mathbf{z}_{1:T}$ introduced in Eq. (21).

independent prior

The original paper [2] assumes that the prior factorizes, i. e., the latent variables are independent over time:

$$p(\mathbf{z}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{z}_t) \tag{37}$$

This assumption greatly simplifies sampling: All prior samples can be drawn simultaneously because we can draw standard normal noise and then reparametrize with the sufficient statistics from the recognition model output, the reparametrization trick of [14, 21] that also allows us to calculate the gradient w.r.t. the sufficient statistics without having to propagate error through stochastic inputs.

Under this assumption, we get

$$p(\mathbf{x}_{1:T}) = \int_{\mathbf{z}_{1:T}} \prod_{t=1}^{T} p(\mathbf{z}_t) p(\mathbf{x}_t \mid \mathbf{h}_t(\mathbf{x}_{t-1}, \mathbf{z}_t, \mathbf{h}_{t-1})) \, d\mathbf{z}_{1:T}, \tag{38}$$

which is an implementation of the graphical model depicted in Fig. (4).

2.2.3 *The Variational Lower Bound for STORN*

In Section 2.2.2, we have derived a model that combines the VAE with RNNs. Much like for the VAE in Section 2.1.5, we will find a suitable energy function to train this model.

Again, we seek to maximize the data log-likelihood (for which we found a representation in Section 2.2.2). As it turns out, all arguments of the derivation of the variational lower bound for the VAE still apply in the case of STORN, so that we end up with a very similar variational lower bound:

$$\begin{aligned}
&\ln p(\mathbf{x}_{1:T}) && (39)\\
&= \int_{\mathbf{z}_{1:T}} q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) \cdot \ln \frac{p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T})}{p(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})} \frac{q_\phi}{q_\phi} \, d\mathbf{z}_{1:T} && (40)\\
&= \int q_\phi \ln \frac{p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T})}{q_\phi} \, d\mathbf{z}_{1:T} + \underbrace{\int q_\phi \ln \frac{q_\phi}{p(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})} \, d\mathbf{z}_{1:T}}_{\geqslant 0, \text{ as a KL}} && (41)\\
&\geqslant \int_{\mathbf{z}_{1:T}} q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) \ln \frac{p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T})}{q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})} \, d\mathbf{z}_{1:T} && (42)\\
&= \underbrace{\mathbb{E}_{q_\phi}\left[\sum_{t=1}^{T} \ln p(\mathbf{x}_t \mid \mathbf{h}_t)\right] - \mathrm{KL}\big(q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) \parallel p(\mathbf{z}_{1:T})\big)}_{=: \mathcal{L}_{\text{STORN}}(q_\phi)} && (43)
\end{aligned}$$

Like before, q_ϕ is a variational approximation of the useful, yet again intractable $p(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})$, which we will also call the recognition model (since it recognizes latent $\mathbf{z}_{1:T}$ in observable $\mathbf{x}_{1:T}$). This variational approximation will be performed by a second RNN (which is distinct from the generative RNN encountered in Section 2.2.2).

2.2.4 *Alternative Model: STORN with RNN prior*

With the previous derivation, there is no chance to get an adaptive prior in the sense that it can incorporate trends from the previous samples. This is a criticism of STORN given in [7]. Their solution called Variational Recurrent Neural Network (VRNN), however, takes a different approach, incorrectly classifying STORN as a special case: While the probabilistic derivation is indeed identical, the computational architecture is very different. VRNN uses one RNN only, while STORN inherently uses two.

We will now derive a new approach which adds the idea of [7] to STORN. The derivation is more driven by the graphical model, which is depicted in Fig. (5). Previously, in Eq. (21), we used the most straightforward factorization of the joint distribution $p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T})$, which was motivated by the derivation of the VAE.

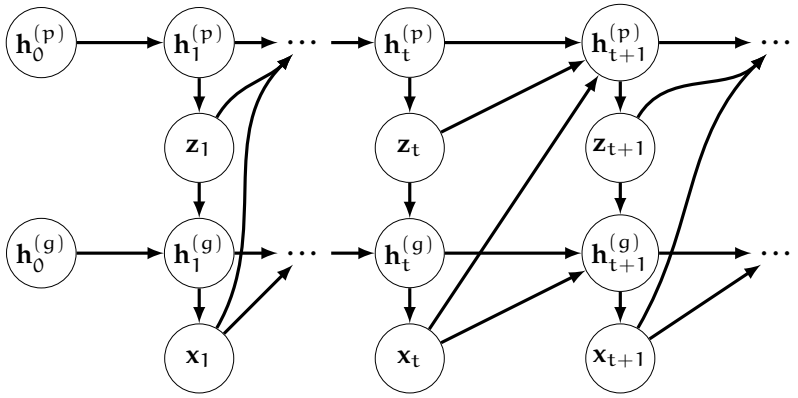


Figure 5: Probabilistic graphical model of a Stochastic Recurrent Network (STORN) with a Recurrent Neural Network (RNN) prior.

However, since we have a sequence of vectors rather than single vectors, we can employ a more sophisticated factorization:

$$p(\mathbf{x}_{1:T}, \mathbf{z}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{x}_t, \mathbf{z}_{1:T} \mid \mathbf{x}_{1:t-1}) \tag{44}$$

$$= \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:T}) p(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:t-1}) \tag{45}$$

$$= \prod_{t=1}^{T} \left[p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:T}) \prod_{s=1}^{T} p(\mathbf{z}_s \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:s-1}) \right] \tag{46}$$

$$= \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1}) p(\mathbf{z}_t \mid \mathbf{x}_{1:t-1}, \mathbf{z}_{1:t-1}) \tag{47}$$

$$= \prod_{t=1}^{T} p\left(\mathbf{x}_t \mid \mathbf{h}_t^{(g)}\right) p\left(\mathbf{z}_t \mid \mathbf{h}_t^{(p)}\right) \tag{48}$$

The first three steps are the by now well-known manipulations. The second last step is a restricting assumption that so that in the last step, like applied previously, we can incorporates an RNN structure as suggested by the graphical model in Fig. (5). In fact, we recognize that (47) and (48) are variants of (35) and (36).

2.2.4.1 *The Variational Lower Bound for STORN with RNN prior*

Interestingly, the lower bound does not change much compared to basic STORN. The reason for this is that up until Eq. (42), we do not

use the particular factorization of the joint distribution of latents $\mathbf{z}_{1:T}$ and observables $\mathbf{x}_{1:T}$. Hence, only the last line changes:

$$\mathcal{L}_{\text{RNN prior}}(q_\phi) := \mathbb{E}_{q_\phi}\left[\sum_{t=1}^{T} \ln p\left(\mathbf{x}_t \mid \mathbf{h}_t^{(g)}\right)\right] - \mathrm{KL}\left(q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) \parallel p\left(\mathbf{z}_{1:T} \mid \mathbf{h}_{0:T}^{(p)}\right)\right) \quad (49)$$

The prior is now also conditioned on the recurrent information $\mathbf{h}_{0:T}^{(p)}$ from previous time steps.

2.2.5 *Architecture and Computation*

In the previous sections, we have derived two variants of a probabilistic model for STORN. This section will show how to implement such a network with the help of RNNs.

The graph in Fig. (6) shows the generic computational flow architecture of the original STORN from [2]. The color coding of the respective VAE Fig. (3) is preserved to highlight the connections. At each time step, the top-down computations can be unfolded according to the scheme in Fig. (3), plus the recurrent inputs

In fact, the remarks from Section 2.1.6 still hold, which is why we will mostly stress the differences here.

We see two RNNs. The first, bidirectional RNN consists of the blue, teal and cyan nodes. It forms the recognition model ϕ. Notice that this is a schematic depiction. One can add hidden layers, and each hidden layer may be vector-valued with differing dimension; for the sake of simplicity, this is reduced as much as possible. Moreover, rather than a bidirectional, one could decide to apply a normal RNN. This largely depends on the application. While the bidirectional version is arguably more powerful, if we want to apply the network for on-line detection we are bound to restrict ourselves.

The recognition model takes observed data (green) as input and outputs sufficient statistics of q_ϕ, the variational approximation of the posterior distribution $p(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})$.

The yellow, orange and violet nodes form the second RNN, the generative model with parameters θ. Due to the Markovian decomposition in our derivation, it is bound to be unidirectional. The rectangular, yellow nodes are samples from the latent distribution. In the case of reconstructive sampling (as during training or analysis of data), the sample is drawn from the output distribution of the recognition model—as depicted in this figure—, whereas in the case of purely generative sampling, it is drawn from the prior, thus the dashed edges.

The red nodes depict the two summands of the lower bound energy function.

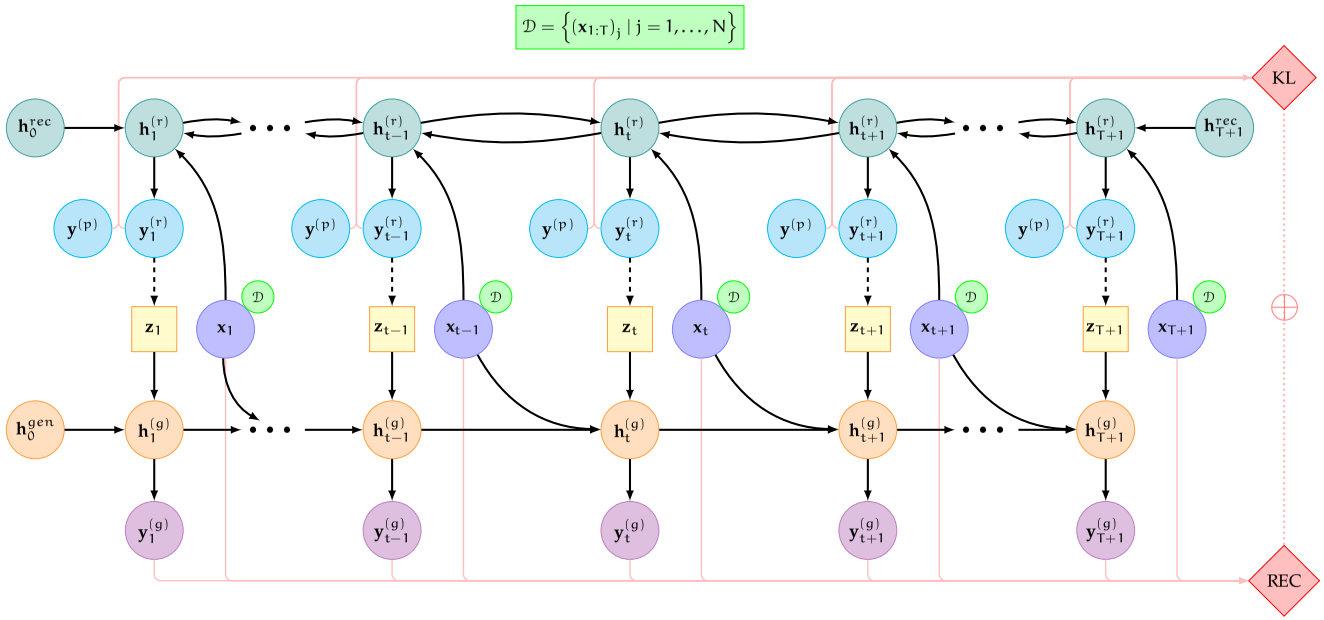


Figure 6: Computational architecture of a Stochastic Recurrent Network (STORN). The graph schematically shows the generic computational flow architecture. The color coding of Fig. (3) is preserved. The blue, teal and cyan form a bidirectional RNN implementing the recognition model ϕ. The horizontal edges depict recurrent connections. The recognition model takes observed data (green) as input and outputs sufficient statistics of q_ϕ, the variational approximation of the posterior distribution $p(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})$.
The yellow, orange and violet nodes form the second, generative model RNN parametrized ϕ. The rectangular yellow nodes are samples from the recognition distribution or the prior distribution. The red nodes depict the two summands of the lower bound loss function.

The three outmost nodes, two on the left, one on the right, depict the RNN initializations. These initial states are also adjustable parameters to be learnt during the training procedure.

Fig. (7) extends Fig. (6). For simplicity, initializations were left out. Previously present edges are shown lightly, while the new RNN that introduces the timely dependence for the prior is depicted by bold edges and newly introduced time indices for the prior nodes.

2.3 APPLICABILITY TO ANOMALY DETECTION

In the previous sections, we introduced Stochastic Recurrent Networks (STORNs) as a means to modeling time series data. With the deepened understanding of STORN in mind, we will now try to characterize the model in the context of anomaly detection. On the one hand, we will compare it to previous approaches as outlined in Section 1.2 and in [20]. On the other hand, we will justify a novel approach to anomaly detection by juxtaposition of our new model against the requirement specification in Section 1.3.

2.3.1 *Categorization*

At first sight, STORN is a generative probabilistic model, i. e., it does not simply discriminate between normal and anomalous time series data, but instead aims at learning the underlying latent structure that generates the data. The network explicitly encodes a probability distribution over time series. In fact, STORN allows us to distinguish between learning the complex data distribution and separates this task from actually detecting anomalies. The detection algorithm can then be based on multiple outputs of STORN, e. g., the likelihood or the distribution of the latent statistics etc.

At second sight, the new approach is not exclusively a probabilistic model. While it is not explicitly reconstructive, the derivation of the variational lower bound in Section 2.2.3 showed that our energy function automatically rewards good reconstruction of our data.

Choosing STORN for detecting anomalies tries to combine the strengths of both approaches to overcome each other's weaknesses. For example, we found that reconstruction-based approaches suffer from sensitivity to hyperparameters. While this is not entirely overcome by shifting to STORN, the loss function derived in Section 2.2.3 as well as the stochasticity of the approach make it more robust to this deficiency. Moreover, we found that probabilistic approaches have been limited to a restricted class of models. In contrast to this, the RNN structure is, in principle, capable of modeling any measurable state transition.

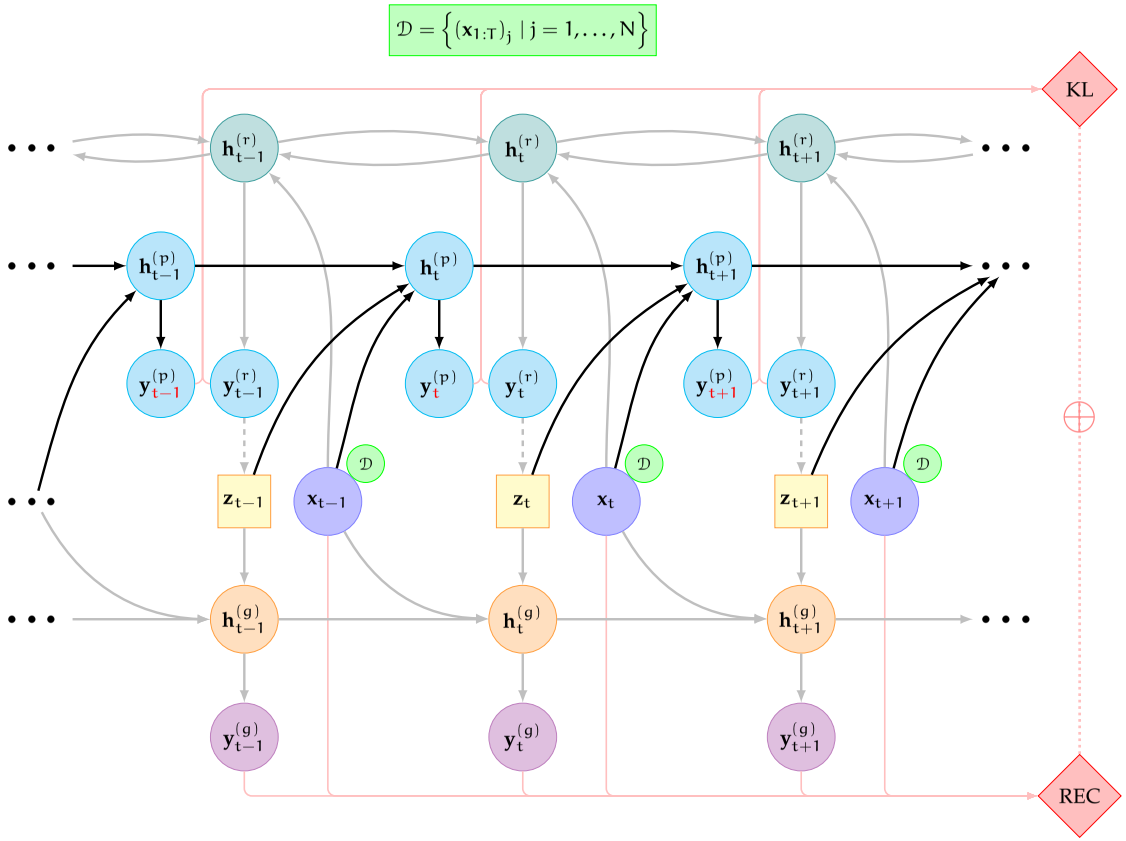


Figure 7: Computational Architecture of a Stochastic Recurrent Network (STORN) with an RNN prior. The light edges are identical to the computational Architecture of the regular STORN, cf. Fig. (6). The bold edges represent the newly introduced RNN that introduces a Markovian dependence structure on the prior variables, notice the added time indices. Hidden statesof the RNN are left out for the sake of clarity.

2.3.2 *Justification against Requirements*

However, this alone does not justify a novel approach. We identified several criteria such an approach needs to fulfill. These were

- handling of multivariate data,
- real-valued time series data,
- multimodality in the underlying distribution,
- no imposed domain knowledge,
- little anomalous data available for training,
- and on-line detection.

As it turns out, STORN can handle all of these requirements.

The auto-encoding mechanism learns a latent structure in the data distribution. If there exists a dependency between two variables (consider the robotic arm examples given in Section 2.2.1), the learning algorithm is designed to learn this structure (e. g., a common cause).

RNNs are very naturally applied to real-valued time series data. Universal approximation capabilities are guaranteed when working on such domains.

The supreme idea behind the VAE is that it is much easier to sample from comparably simple distributions and then map them to more complicated spaces based on the inverse sampling method. RNNs then allow us to have rich distributions that are not limited such as Gaussian Mixture Models. The latter would allow for multimodality but only in a restricted fashion that relies on handcrafted hyperparameters, such as the number of mixture components.

Furthermore, the auto-encoding mechanism allows us to train our model in an unsupervised fashion. This helps us in two ways: First of all, STORN will model normal data only and we will, in an independent step, define means to use it to discriminate anomalous data. Thus, in order to train STORN, not only can we leave out anomalous data for training, we are even encouraged to abandon it for training in order to get a precise model of normality.

Secondly, the learning algorithm is theoretically designed such that we do not have to impose prior knowledge. We could do so by choosing an appropriate prior distribution—which might yield better results in practice—but we are not required to do so by the derivation.

One core assumption with all the previous arguments is that our training data covers the observable space sufficiently. If it does not, our model will learn reasonably well on the training data, but it will not generalize well to previously unobserved new data. While recording normal data is much less costly than anomalous data, resources for recording are still limited. It will, thus, be of interest whether a

limited amount of data suffices to leverage all theoretical advantages outlined in this section.

The last remaining item on the list of requirements was on-line detection. Since all error terms are decomposed into individual summands due to the usage of the log-space as well as Markovian and independence properties in the derivation, the likelihood (or the lower bound, our energy function) can easily be calculated on-line. STORN is capable of detecting unusual, i. e., anomalous, developments of the log-likelihood or any derived normality criterion on-line without the need to look at the whole time series. Moreover, the computations are comparably cheap, a very important component of real-time capability. The hard effort is to be made during the training phase long before application.

Summary

We can conclude that STORN theoretically fulfills all our requirements for a novel approach in anomaly detection for robot time series data. While experiments will need to prove whether the given amount of data is sufficient to leverage the theoretical advantages, to the best of our knowledge, there has been no such attempt of anomaly detection. This justifies further consideration of STORN for the problem at hand.

3

EXPERIMENTS & EVALUATION

This third and last chapter reports the experiments conducted to verify the applicability of STORNs to anomaly detection. To this end, Section 3.1 will analyze the data set. Section 3.2 will then describe the training procedure. In Section 3.3, the experiments are evaluated thoroughly. Lastly, Section 3.4 summarizes the findings of this thesis.

3.1 DATA

The crucial difficulty with anomalies is that they cannot be described in closed form. If they could, anomaly detection would be easy, since we could fit a model to this description. Hence, it is hard to simulate anomalies, and consequently our goal is to evaluate our method using real-world data. To the best of our knowledge, there exists no labeled data set of high-dimensional, interdependent anomalies. Hence, we recorded our own anomaly data set with the application on robots in mind.

3.1.1 *Baxter Robot*

We recorded data on the research robot Baxter[1]. Baxter has, among other features, two arms with seven degrees of freedom, namely two joints in its shoulder, two more in its elbow and three in its wrist. One such arm is shown in Fig. (8)

The seven degrees of freedom include a shoulder roll and pitch, an elbow roll and pitch, and two wrist rolls and one wrist pitch.

For the experiments in this thesis, we restricted ourselves to one arm of Baxter.

Baxter can record commanded and measured joint torques for each joint, its commanded and measured configuration in configuration space, i. e., relative angles of the joints rather than Cartesian coordinates, as well as the measured acceleration of the wrist and camera image data from the hand.

3.1.2 *Anomaly Scenario and Data Set*

As an anomaly scenario, we designed an experiment where the robot fulfills a task and bumps into an obstacle (like a human co-worker or any other dynamic entity within the robot's environment).

1 http://www.rethinkrobotics.com/baxter-research-robot/

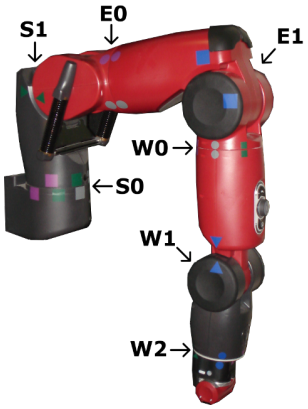


Figure 8: One seven-degree-of-freedom arm of a Baxter research robot with respective joint labeling for shoulder (S0: shoulder roll, S1: shoulder pitch), elbow (E0: Elbow Roll, E1: Elbow Pitch) and wrist (W0: First Wrist Roll, W1: Wrist Pitch, W2: Second Wrist Roll).

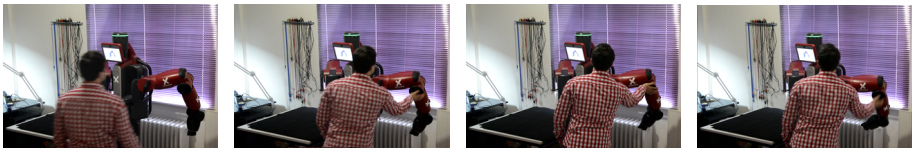


Figure 9: Human operator hitting the arm on command to create anomalies in the data sets.

To simulate a task, Baxter starts from a predefined configuration and then moves between ten selected waypoints in Cartesian space. This imitates a working process where Baxter has to fulfill a certain grab-and-drop or sorting task.

To make the trajectories more diverse (and harder to learn), the sequence of waypoints is sampled randomly from the ten given waypoints with each trial. Baxter moves for at least 30 seconds and is afterwards commanded to return to its original position.

The anomaly is simulated as follows: For each segment between two waypoints, Baxter internally makes a random Bernoulli decision whether an anomaly occurs or not. If an anomaly is supposed to occur, Baxter displays a 'hit' command to a (human) operator, which then hits a randomly selected part of the arm. The process is depicted in Fig. (9).

Time stamps for the commands are available. Due to reaction time of the operator, there is a delay between the time stamp and the actual occurrence of the anomaly.

Given this anomaly scenario, we recorded 1308 trials of varying length around 35–40 seconds. Out of these, 1008 trials are guaranteed to be normal. 640 of these will serve as training samples, 160 as validation sample and the remaining 208 as parts of the test samples. The

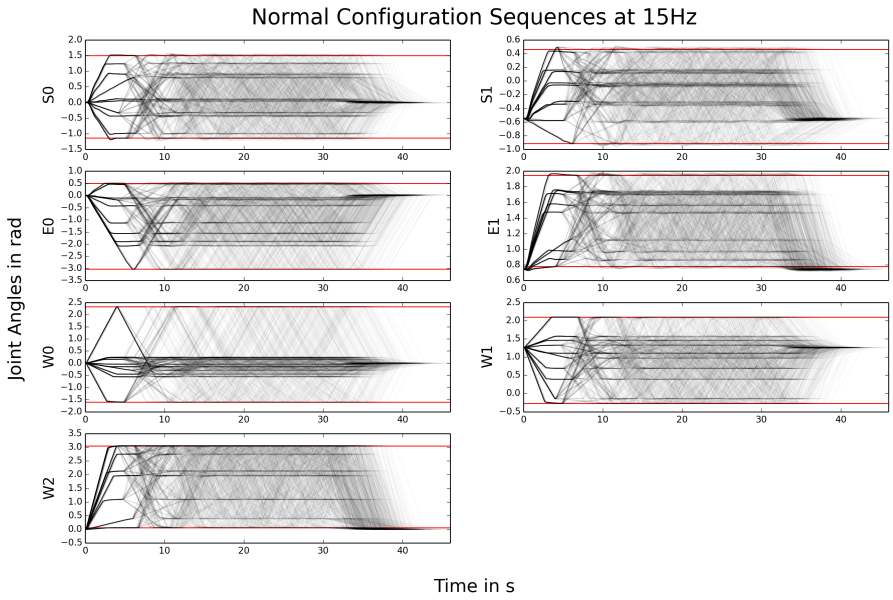


Figure 10: All normal trajectories superimposed on top of each other. The red lines show the maximum value of the ten given waypoints for each joint. Baxter not obeying these is due to tolerances. The plot roughly shows the marginal data distributions to learn by our model.

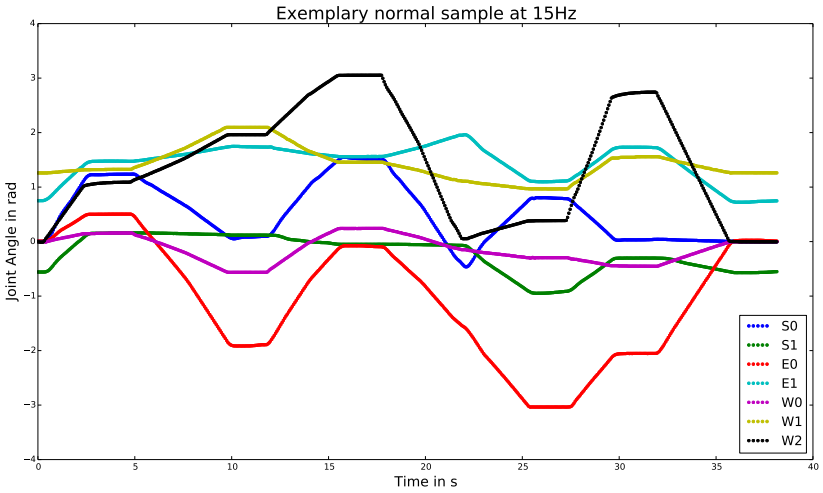
other 300 are potentially anomalous trials (that is, some may have no anomaly since the Bernoulli experiments for commanding a hit all failed). These will also be used for testing only, so that we end up with 508 test samples.

Finding a hit is easy if one can use either joint torques (since they get unusually high) or any commanded information (since a simple comparison with the actual trajectory easily shows the anomaly). Thus, in our data sets, we restrict ourselves to the seven-dimensional data of measured configuration angles. In these seven dimensions, hit anomalies are much more subtle to discover. Still, the data is seven-dimensional and highly dependent (in particular over time).

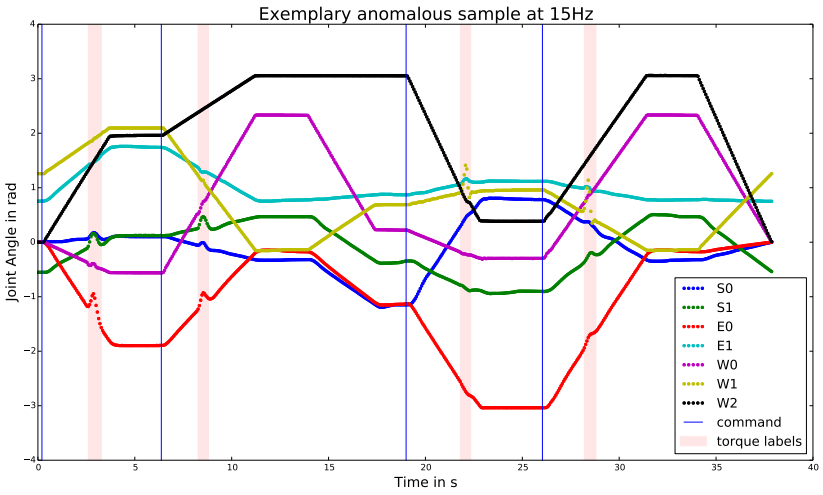
The measured configuration can be sampled at up to 800 Hz. Since Baxter is no real-time system in the sense that it can guarantee equidistant time steps, we have created a data set interpolated with linear splines to get equidistant time stamps at 15Hz.

The superposition of the entire 1008 normal samples is depicted in Fig. (10). This shows the seven marginal distributions of the joint distribution $p(\mathbf{x})$ our model is supposed to learn.

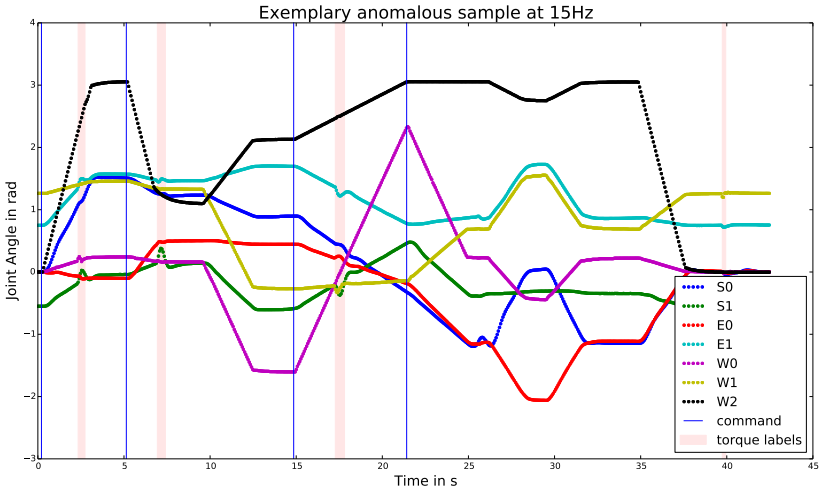
Fig. (11) shows three exemplary samples from the data set, one normal and two anomalous ones. The first anomalous sample shows some anomalies that are easy to detect visually and some that are hard to distinguish from some noise and tolerance based artifacts in the normal sample. Depicted with blue vertical lines, one can see



(a) Normal sample. Each line corresponds to the configurations of one joint over time. It can be seen that due to imprecisions of Baxter, some of the linear parts are nonlinear.



(b) Anomalous sample. The blue vertical lines show where an anomaly was commanded. The red areas depict areas time steps where the derivative of the torque was above a threshold so that we deemed it an anomaly.



(c) As in (11b). One can see that the manual labeling of anomalies sometimes fails.

Figure 11: Exemplary samples.

Hyperparameter	Range of Choices
Batch Size	$5, 10, 20$
Recognition Model	
No. Layers	$1, 2, 3$
No. Hidden Units per Hidden Layer	$8, 16, \ldots, 128$
Generative Model	
No. Layers	$1, 2, 3$
No. Hidden Units per Hidden Layer	$8, 16, \ldots, 128$
No. Stochastic Units	$8, 16, \ldots, 128$
Output Standard Deviation	$0.00001, \ldots, 0.1, 1$
Fast-Dropout Rates	
Input-to-Hidden, Hidden-to-Hidden,	
Hidden-to-Output, Shortcut	Interval $(0, 1)$
Weight Initialization (Standard Deviations)	
Input Layer	0.000001
Hidden Layer	$1/64, 1/32, \ldots, 2$
Transfer Functions	sigmoid, tanh, LSTM

Table 1: Hyperparameter grid on which random search was performed.

the commands to the operator to cause anomaly. The red areas show where the absolute derivative of the torque showed unusually high values. This will serve as a ground truth labeling for on-line detection. The second anomalous sample, however, shows that these labels should be handled with care. On some occasions, the joint torque labeling fails to detect obvious anomalies and takes artifacts as anomalies.

3.2 TRAINING

In Section 2.2, we derived two variants of STORN. For the original variant without RNN prior, we ran experiments with a unidirectional and a bidirectional RNN in the recognition model. For each of the three models, 100 experiments were conducted to find good hyperparameters, with early stopping based on the validation performance. Hyperparameters of the models were optimized by random search on the grid depicted in Table (1). We used fast dropout, [27, 3], and potentially LSTM transfer functions.

The models were optimized by stochastic gradient descent with mini-batches, i. e., each step was not based on the gradient w.r.t. the

	Training	Validation
unidirectional	23.0405	23.0368
bidirectional	22.7066	22.6996
RNN prior	24.3020	24.2908

Table 2: Training results for the best model in each of the three categories. The best lower bound value is shown.

full data set, but on a significantly smaller subset. We applied *adadelta* for the learning rate (step size), cf. [29].

Training was unsupervised on normal samples. No preprocessing, except for random shuffling of the samples per experiment and normalization to zero mean and unit standard deviation for the whole data set, was performed.

Training and validation was based on the lower bound performance only. Anomaly detection and learning on labels was introduced only after completion of training.

The models were implemented in *theano*, [4, 1], a python-based software package that supports automatic differentiation, which greatly simplifies training of neural networks, and *climin*, [13], a python-based optimization toolbox.

The optimal training results are shown in Table (2). Interestingly, there is no over-fitting effect on the training data visible.

We observed that the usage of LSTM transfer functions lead to no improvement, particularly when contrasted to the increased model size.

More surprisingly, unidirectional model needed fewer epochs (i. e., runs through the whole training set) to yield a lower training and validation error.

Fig. (12) shows reconstruction errors and one-step prediction errors (the difference is that the latter has no information about the current time stamp while the former tries to reconstruct it from data).[2] Ground truth and reconstruction/prediction are almost indistinguishable. Notice the small scale of the squared residuals.

For normal samples, the highest errors for both reconstruction and prediction occur at change points in the trajectory. This was to be expected. On the one hand, the most plausible prediction is linear continuation, not a change of direction. On the other hand, even if the system notices that it has reached one of the ten waypoints, the change points suffer most from Baxter's tolerance as to when it thinks it has reached a goal. Moreover, the change in direction causes slight repercussions.

2 Predictions are missing for RNN priors due to an incomplete implementation of the purely generative sampling—the generative model is still fed with the unconditioned prior.

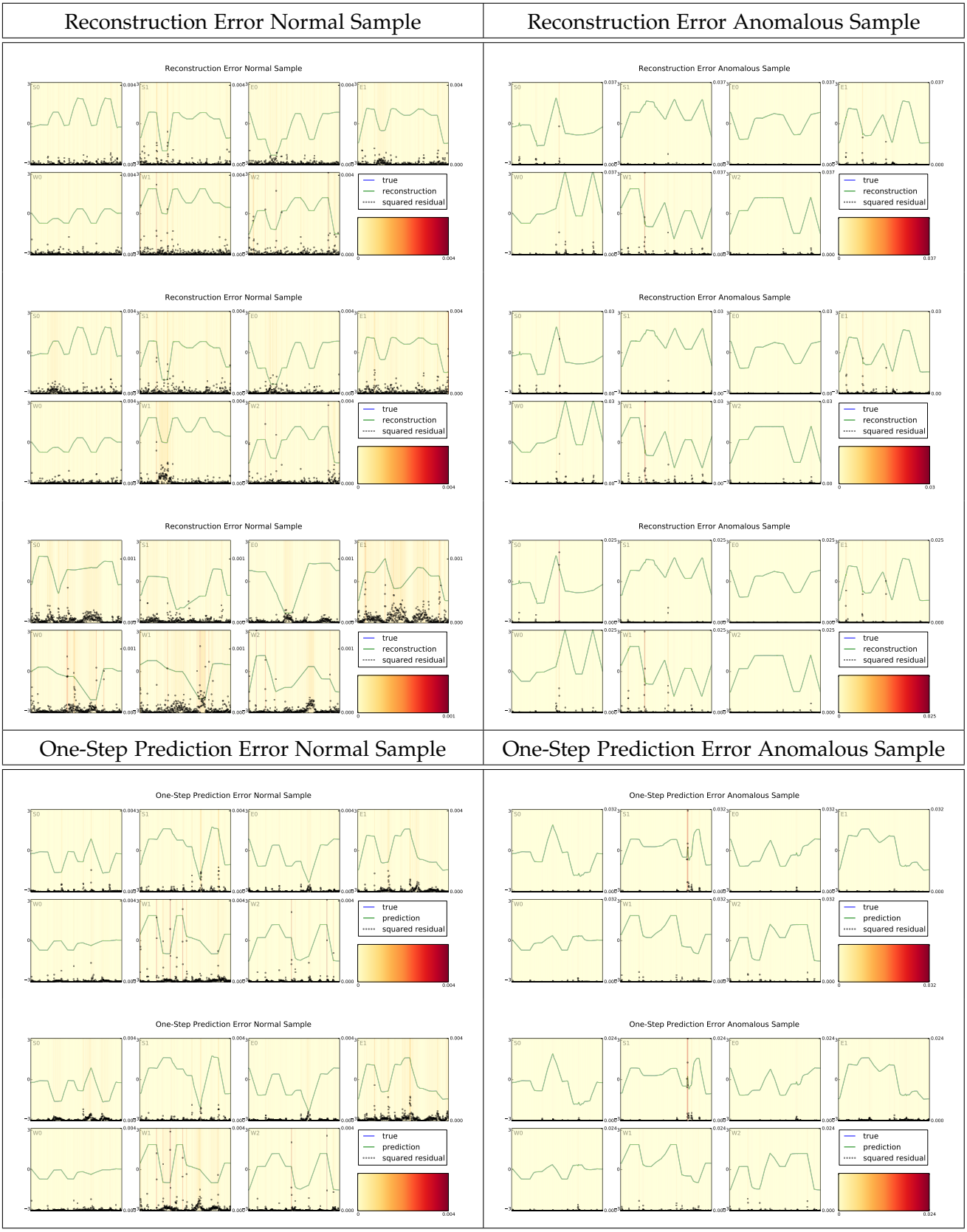


Figure 12: Reconstruction Error (top) and One-Step Prediction (bottom) for normal (left) and one anomalous (right) sample for unidirectional, bidirectional and RNN prior recognition model, respectively (no prediction for RNN prior). Notice the different scale of the error axis and color coding.

We see that the residuals for anomalous samples are at least one order of magnitude higher than for normal samples. We will exploit this later for anomaly detection.

3.3 ANOMALY DETECTION EVALUATION

Based on the optimal models from the training process, we would like to build an anomaly detector. We will tackle two kinds of detection mechanisms.

Firstly, we will classify entire sequences into whether they hold an anomaly or not. We will denote this as *off-line detection*.

Secondly, we will classify individual time steps into whether an anomaly is occurring in this specific time-step, which we will denote as *on-line detection*. Here, on-line refers both to the fact that we are trying to detect anomalies without knowledge about future time steps as well as the fact that we classify individual time steps.

The boundaries between these two classes are not strict. It is clear that a good on-line algorithm can also be used for off-line detection.

3.3.1 *Off-line Detection*

The labels for off-line detection are extracted from the commands. An entire sequence is considered anomalous if there was at least one hit command throughout the sequence. On average, an anomalous sequence has roughly three hit commands. Thus, an anomaly label is weak in the sense that the detector only needs to detect one in three anomalies.

We could try to detect the number of anomalies off-line. However, not every hit command corresponds to a true anomaly, cf. Section 3.1.

Our off-line detection algorithms all employ the following principle: A test sample is mapped to a scalar called the *anomaly score*. The anomaly label in combination with the label can then be used to establish a threshold for anomalies.

We tested the following four criteria:

LOWER BOUND After the derivations of Chapter 2, the lower bound as the output of STORN is the most natural quantity to choose. Given the fully trained model, it is easy and fast to calculate and is very close to the data likelihood.

LOG-LIKELIHOOD ESTIMATE As shown in Appendix A.2, we can get an estimate of the true data likelihood. In practice, this method is not favorable because it is computationally much more costly than, e. g., the lower bound. However, for comparison with the lower bound criterion, it is interesting to check the estimate as well.

GLOBAL ONE-STEP PREDICTION ERROR THRESHOLD As indicated in Section 3.2, the discrepancy in scale between prediction error for normal and anomalous samples strongly suggests using it as a measure of anomaly. In order to map the entire sequence to an anomaly score, we could take the maximum of the values, but this is error-prone due to outliers and boundary effects. Thus, we choose a .995 percentile of the residuals.

GLOBAL STEPWISE LOWER BOUND THRESHOLD The lower bound calculated by STORN is actually a mean of stepwise lower bounds. In an attempt to combine the idea of the lower bound with the detection mechanism of the prediction error threshold, we define the following anomaly score: the .005 percentile of all stepwise lower bounds for the entire sequence. A low stepwise bound corresponds to an unexpected data point, which we assume to be anomalous.

For analyzing the binary classification of a continuous scalar quantity, the so called Receiver Operating Characteristic (ROC) is a useful tool. The ROC plots the true positive rate (tpr) (i. e., the fraction of correctly classified anomalies among the samples labeled anomalous) against the false positive rate (fpr) (i. e., the fraction of normal samples incorrectly classified as anomalies) for an increasing classification threshold. A perfect classifier has a tpr of 100% and an fpr of 0%.

Usually, there is a trade-off between tpr and fpr and the optimal trade-off depends largely on the application, e. g., a trade-off between the cost of false alarms like system shutdowns and the costs of anomalies not detected soon enough.

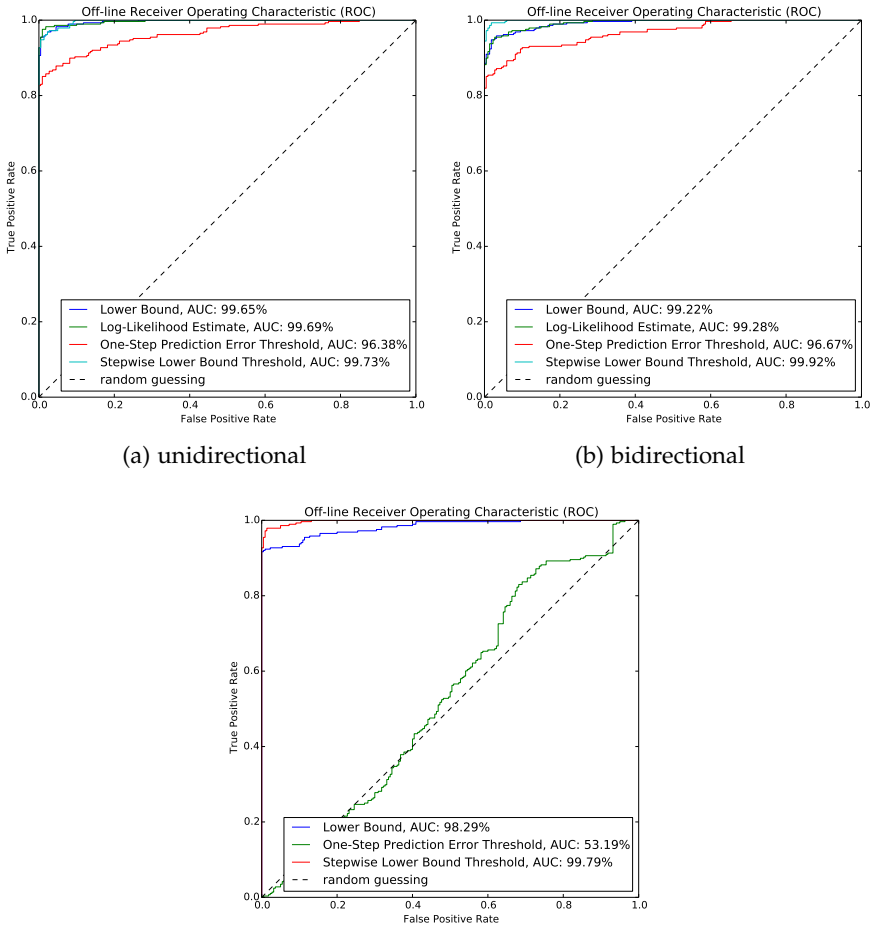
A general measure for the quality of an anomaly measure is the so-called Area under Curve (AUC). Intuitively speaking, it 'integrates out' the trade-off considerations by considering the area under the ROC curve, i. e., integrating up the quality of all possible thresholds.

Fig. (13) shows the ROCs and respective AUCs for the three best models[3].

The results are remarkably good. With a bidirectional recognition model, we achieve an AUC as high as 99.92%. The Stepwise Lower Bound Percentile Criterion turns out to consistently outperform the other approaches. Just as notable are the high entry points on the left, i. e., very high tprs with an fpr of 0%.

Keeping in mind that the label is very weak, we can still conclude that off-line detection works extremely well.

3 Again, the bad result for prediction and the missing log-likelihood estimate go back to the erroneous implementation of the generative sampling.



(a) unidirectional (b) bidirectional

(c) RNN prior

Figure 13: ROC for the four criteria with respective AUC. The top left corner corresponds to a perfect classifier. High AUC indicates a good generalization of the anomaly score.

3.3.2 *On-line Detection*

With off-line detection solved, we can turn towards on-line detection. As before, we will derive anomaly scores. The difference is that the score will be assigned to individual time steps rather than the entire sequence as with off-line detection.

For on-line detection, we have developed several different scoring methods:

ON-LINE LOWER BOUND Like before, this is a natural choice given the output of STORN. Rather than the total lower bound, we consider it per time step.

SMOOTHED ON-LINE LOWER BOUND Since the lower bound is a sample from a stochastic quantity, it suffers from certain fluctuations. By convolving with a small 1D Gaussian Kernel, i. e., a moving weighted average, we can smooth the empirical lower bound and reduce unwanted artifacts.

ON-LINE LOWER BOUND FORWARD DIFFERENCES The two previous methods assume that a change in the lower bound corresponds to an anomaly. A natural extension of this approach is to consider the *rate* of change in the lower bound, i. e., the first derivative. The first derivative is approximated by forward differences, since we have no closed form solution for the lower bound.

INPUT GRADIENT MAGNITUDE The fourth approach has a very different view of anomalies. This method calculates the gradient of the lower bound w.r.t. the input. In other words, because the gradient indicates the local direction of steepest ascent, we calculate where to change the current input most effectively to improve the lower bound. By calculating the magnitude of the gradient time series per time step, we get an indicator as to which time step profits most from a change. This can be used as an anomaly score.

Remember that, as discussed before, we are limited to the models where the recognition model is unidirectional. Technically, we could also apply bidirectional recognition models, but then we would have to evaluate the entire sequence up until time step t to get the anomaly score for the respective time step.

After having discussed on-line anomaly scores, we have to focus on second weak spot of on-line anomaly detection: Other than with off-line detection, we want to avoid missing any anomaly.

Moreover, it is much harder to find meaningful labels. Even if one had perfect labels of when the anomaly was triggered, its effects can still be detectable long after the cause has been dissolved. It depends very much on the system at hand whether a detection algorithm that

cannot detect the triggering moment, but only the aftermath of an anomaly, is still valid.

This difficulty can be pushed even further: Suppose that we were bound to detect the anomaly in a very short period after it was triggered and suppose the cause was present for ten time steps (i. e., such that these time steps could be labeled anomalous while all others are deemed normal). Do we consider the anomaly as detected if one of the anomalous time steps is correctly classified? From an engineering point of view, we will in many cases agree. This, however, implies that we are not interested in good classification performance in terms of labels.

In the latter case, the so-called Positive Predictive Value (PPV), i. e., the ratio of truly anomalous time steps among all anomalously labeled time steps, might be a better criterion. The PPV again has disadvantages of its own: It rewards being overly cautious. In fact, the PPV is optimal if we detect one and only one anomalous time step correctly.

With all these considerations in mind, we conclude that the evaluation of an on-line anomaly detection approach depends much more on the system to which it is applied. The choice of good thresholds is more of an engineering task than a statistics task.

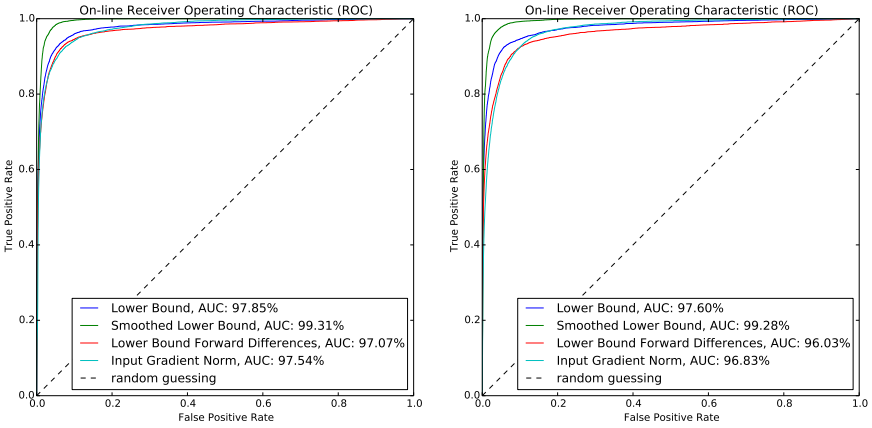
While this may at first sound like reintroducing domain knowledge through the back door, the contrary is true: The major advantage over a domain-specific anomaly detection approach is that the 'engineering' takes place in the space of sensitivity, specificity, tprs and fprs—STORN transfers any domain to a common ground where generic methods can be applied and combined out of the box.

In the case of the data set described in Section 3.1, we have two different labels for each time step.

The first label is torque-based: If the derivative (in terms of forward differences) of the torque is above a manually set threshold, we consider the time step to be anomalous. As discussed before, this misses some of the subtle anomalies while it detects anomalies where there are only artifacts in the data.

The second label is based on the hit commands: In a certain Δt after a hit command, we expect there to be an anomaly, hence we label the whole span as anomalous. This obviously has drawbacks of its own: Not all hit commands correspond to an anomaly and some anomalies may have a higher latency than Δt w.r.t. their hit command. Moreover, the labeling is very broad. While we chose $\Delta t = 4$, none of the anomalies in the data set is visible for longer than a second. This labeling is inspired by the above consideration that we are actually more satisfied if we can, during a real anomaly, spot only one anomalous time step.

In the light of these observations, we will apply three different methods for extracting a threshold on the on-line anomaly scores.



(a) unidirectional (b) RNN prior

Figure 14: On-line ROC based on labels derived from torque with respective AUCs.

The first two only consider tprs and fprs w.r.t. the torque-based labels. To this end, we calculate the ROC. It can be seen in Fig. (14). While at first glance, these ROCs look just as promising as in the off-line case, they should be considered with care. The labels are imbalanced towards normal labels, because even for anomalous trajectories, most time steps are perfectly normal. Due to this fact, even a small fpr leads to many misclassifications—remember the standard statistics text book example on sensitivity and specificity of disease tests.

Nevertheless, for evaluation purposes the first of the three thresholds will be the one corresponding to the point on the ROC closest to the 'perfect' spot in the top-left corner. Let κ denote an arbitrary threshold, then this first evaluation threshold κ_1 can be computed as:

$$\kappa_1 = \arg\min_{\kappa} \left(\mathrm{fpr}(\kappa)^2 + (1 - \mathrm{tpr}(\kappa))^2\right) \tag{50}$$

The second threshold incorporates the notion of 'being more cautious' with anomaly labels. It regularizes the cost function in Eq. (50) with the PPV:

$$\kappa_2 = \arg\min_{\kappa} \left(\mathrm{fpr}(\kappa)^2 + (1 - \mathrm{tpr}(\kappa))^2 - \underbrace{\frac{\mathrm{tpr}(\kappa)}{\mathrm{tpr}(\kappa) + \mathrm{fpr}(\kappa)}}_{=:\,\mathrm{PPV}}\right) \tag{51}$$

The third threshold is more complicated. It uses both types of labels, torque-based and command-based.

First, we compute the ROC for any on-line score w.r.t. the torque-based labels, i. e., we get a set of thresholds with associated fpr and tpr. For every such threshold (there are only finitely many because we have finitely many data points and thus finitely many gaps to place thresholds), we then compute the fpr and tpr, but this time w.r.t. the command-based labels.

The third threshold is finally computed as:

$$\kappa_3 = \arg\min_{\kappa} \Bigg(f_t(\kappa)^2 + (1 - t_t(\kappa))^2 - \frac{t_t(\kappa)}{t_t(\kappa) + f_t(\kappa)} - \lambda \frac{t_c(\kappa)}{t_c(\kappa) + f_c(\kappa)} \Bigg) \tag{52}$$

For brevity, we have shortened the torque- and command-based fpr and tpr to f_t, t_t, f_c and t_c, respectively.

The factor λ is a regularization weight that balances the trade-off between the two different labels. The higher λ, the more important the command-based label and vice versa.

By combining the four anomaly scores with the three thresholds, we yield twelve different thresholds[4].

Exemplary on-line detection is depicted in Fig. (15) and Fig. (16). We can observe multiple things.

Firstly, κ_3 is a cautious, but not overly cautious threshold. It detects nothing on normal samples and robustly detects torque-labeled anomalies. Secondly, it is not restricted to any of the labels. It also detects spurious, anomalous artifacts in the data far outside any region around a hit command. We can conclude that the trade-off between the two labels boosts the robustness of the threshold significantly.

As stated previously, the evaluation of an on-line algorithm is hard to do without a context like reduced costs. However, we find a proxy to measure the quality of our newly gained thresholds: We use our on-line detectors for off-line detection. If this improves the off-line performance, we have a hint that the new threshold extracted more information from the model.

The respective ROCs can be seen in Fig. (17). The committee mentioned in the figures is an average score out of the other four on-line anomaly scores after normalization. Applying weights to the members of the committee also showed no effect.

We see almost perfect classification. A less optimal model w.r.t. validation loss even scored perfectly, finding a threshold that discriminates the off-line anomaly labels perfectly.

We can conclude that clever statistical engineering boosts the performance of on-line anomaly detection. A quantitative analysis would require additional metrics, such has expected running and repair costs.

3.4 CONCLUSION

This thesis has successfully introduced Stochastic Recurrent Networks (STORNs), a combination of Recurrent Neural Networks (RNNs) with

4 We also tried smoothed forward differences and smoothed input gradients, none of which had any effect on the results. Thus, these scores are discarded in this analysis.

Figure 15: On-line detection for the unidirectional recognition model. The trajectories show one normal and two anomalous samples of joint configurations over time. The twelve bars on top show the four anomaly scores combined with the three thresholds κ_1, κ_2 and κ_3. Green means that no anomaly is detected, red the opposite. Yellow colors indicate an anomaly score around the threshold.

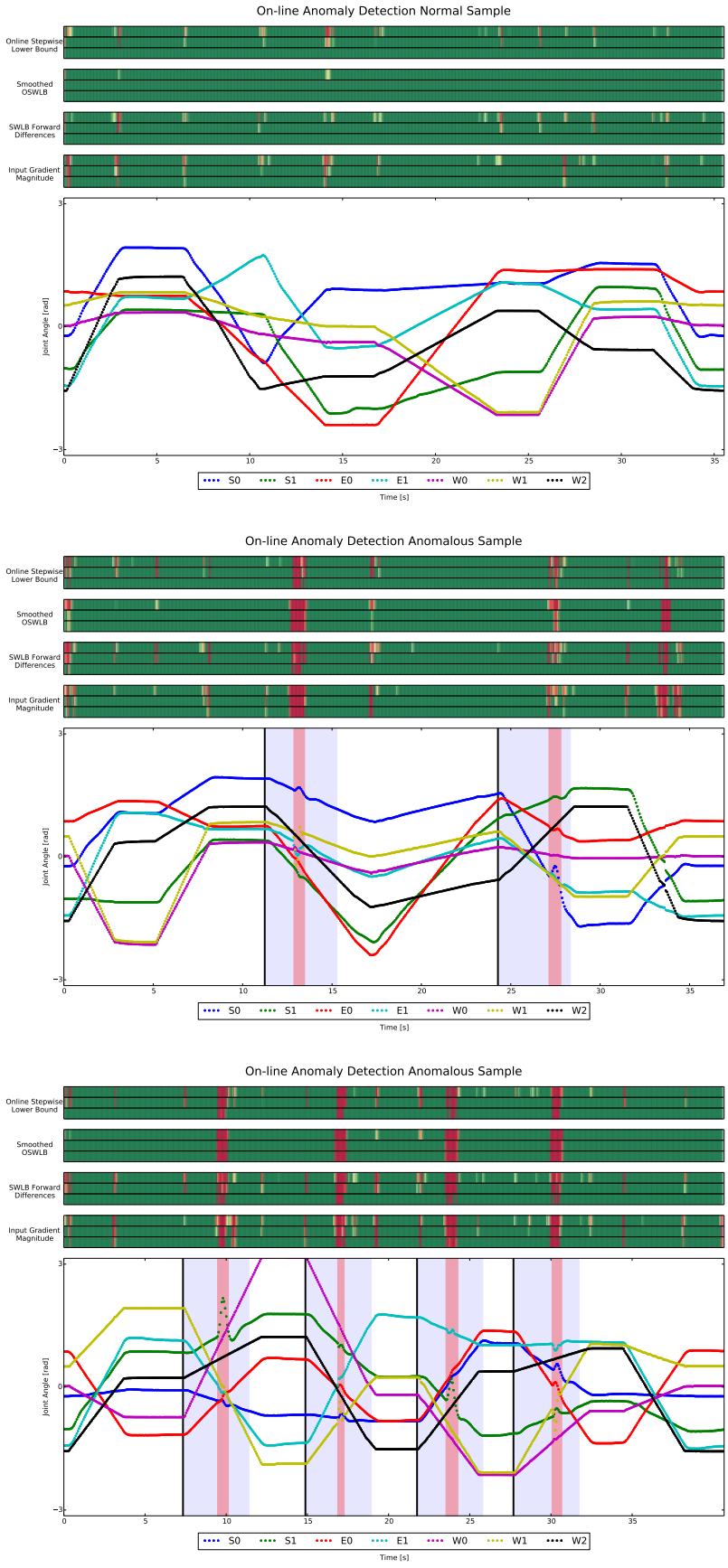
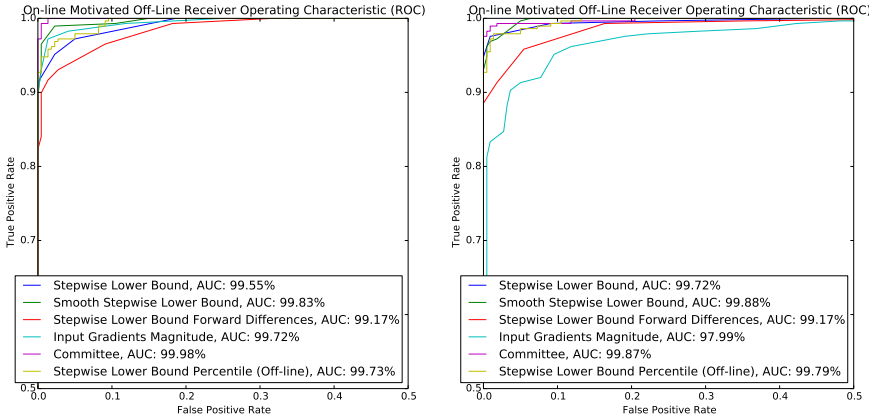


Figure 16: On-line detection for the RNN prior recognition model. The trajectories show one normal and two anomalous samples of joint configurations over time. The twelve bars on top show the four anomaly scores combined with the three thresholds κ_1, κ_2 and κ_3. Green means that no anomaly is detected, red the opposite. Yellow colors indicate an anomaly score around the threshold.



(a) unidirectional (b) RNN prior

Figure 17: Off-line detection ROC based on on-line thresholds with respective AUCs. Only the top-left corner of the ROC is shown. For comparison, the previously best off-line method is shown as well.

Variational Auto-Encoders (VAEs) as a framework for detecting anomalies in time series data. Real-world robot data served as a proof of concept with remarkably robust results both on- and off-line, with perfect classification off-line. The detection algorithm was solely based on metrics derived from the outputs of STORN, which makes it easy to transfer the approach to different data sets. No domain knowledge was required.

Future investigations will include different data sets from different robots and/or different tasks and anomalies. Furthermore, it is worth examining whether the latent dynamics of the learned model in any way resemble the true dynamics. Long-term sampling from STORN indicates room for improvement, though, with [26] in mind, this might be a rash conclusion. One possibility is the explicit integration of control inputs, which should stabilize the internal dynamics of STORN.

A

APPENDIX

A.1 NEURAL NETWORKS

We will recall the theory and techniques for applying Neural Networks (NNs) in the context. For a more detailed review, the reader is referred to [5, Ch.5] as an introduction, or to [23] for a more research-driven, historical outline.

Of the many ways NNs can be interpreted (e. g., as the name suggests, the biological), it is recommended to view them as black boxes for universal function approximation throughout this thesis—a powerful means to an end.

We will revise two of the most common architectures: Feed-Forward Neural Networks (NNs) and Recurrent Neural Networks (RNNs).

A.1.1 *Feed-Forward Neural Networks*

Feed-Forward Neural Networks (NNs) aim at approximating a function $g : \mathcal{X} \to \mathcal{Y}$, $\mathcal{X} \subset \mathbb{R}^{d_x}, \mathcal{Y} \subset \mathbb{R}^{d_y}$. To this end, we apply a fixed sequence of parametrized functions, each of which consists of an affine mapping followed by a non-linear transform. This can be cast into the following recursive definition for $l = 1, \dots, L$:

$$\mathbf{h}^{(l)} = f^{(l)}\left(\mathbf{h}^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}\right) = f^{(l)}\left((\mathbf{h}^{(l-1)}, 1)\tilde{\mathbf{W}}^{(l)}\right) \tag{53}$$

$$\mathbf{h}^{(0)} = \mathbf{x} \in \mathcal{X} \tag{54}$$

The entire calculation is called *forward-propagation*. Each of the L mappings (53) is usually referred to as a *layer*, with the intermediate layers called *hidden*, hence the notation. The *transfer functions* $f^{(l)}$ are fixed non-linear functions. Common examples are sigmoidal functions like the tangens hyperbolicus tanh or the logistic function $(1 + \exp(-x))^{-1}$, which are applied component-wise if the output of the affine mapping is vector valued.

forward-propagation
layers
transfer functions

The parameters $\tilde{\mathbf{W}}^{(l)} = (\mathbf{W}^{(l)}, \mathbf{b}^{(l)})^T$ are adjustable through training and are called *weights* (hence the notation). Often, the weights of a NN are gathered in a parameter set

weights

$$\theta = \left\{\tilde{\mathbf{W}}^{(l)} \mid l \in \{1, \dots, L\}\right\}.$$

The output of the last layer is assumed to be an approximation of the true function value: $\mathbf{h}^{(L)} \approx g(\mathbf{x})$. It is not obvious that NNs are a reasonable tool for approximating the function g. However, it has been

proven [15, 12, 9] that feed-forward NNs are universal function approximation tools. This and the state-of-the-art performance in many machine learning and pattern recognition tasks and competitions render the popularity of NNs.

dropout and fast dropout

A state-of-the art technique to prevent over-fitting of the NN is dropout, cf. [10]. During the learning procedure, in every application of a weight, it is randomly set to 0 (i. e., a Bernoulli experiment for switching it off is conducted). This prevents over-fitting because information has to be extracted from data by the NN under the assumption that parts of the network are not present. In essence, all 2^W architectures, with W being the total number of weights of the network, are trained simultaneously. The final network is a weighted average with the weights according to the probability of the particular architecture.

In the spirit of the Central Limit Theorem, a computationally more feasible generalization of dropout, called fast dropout, has been developed. Here, each weight is associated with a variance of a Gaussian distribution (the limit of multiple Bernoulli experiments). For details, cf. [27].

A.1.1.1 *Learning and Back-Propagation*

The proofs of the universal approximation capability are non-constructive. Thus, the central task in applying NNs is finding 'appropriate' weights. This task consists of two parts.

Firstly, the appropriateness of a set of weights is defined through an energy function $\mathcal{L}(\theta, \mathbf{x})$, which implicitly uses the NN, and we would like to maximize $\mathbb{E}_{p(\mathbf{x})}[\mathcal{L}(\theta, \mathbf{x})]$.[1]

Usually, it is impossible to evaluate the quality of the function approximation for the entire (and mostly inherently unknown) data distribution $p(\mathbf{x})$. Thus, the loss function is evaluated on some training data set $\mathcal{D}$. In the setting where we want to approximate a function, this could be a (finite) number of tuples $(\mathbf{x}_n, g(\mathbf{x}_n))$ and we aim at maximizing

$$\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\theta, \mathbf{x}_n) \approx \mathbb{E}_{p(\mathbf{x})}[\mathcal{L}(\theta, \mathbf{x})] \tag{55}$$

w.r.t. θ. We seek to find θ^* such that

$$\theta^* = \arg\max_{\theta} \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\theta, \mathbf{x}_n) \tag{56}$$

This is done via non-linear optimization techniques and usually involves first-order gradient descent methods (assuming, of course, differentiability of the energy function). The optimization is the second

1 The derivation of new algorithms often involves finding an energy function that captures the structure of the problem to solve, as is done in this thesis in Section 2.1.5 and Section 2.2.3. The details of these procedures are beyond the scope of this appendix.

important part of finding appropriate weights. The efficient computation of the derivative is crucial for training neural networks.

The state-of-the-art algorithm for doing so is called *back-propagation* (it propagates errors against the direction of forward-propagation). The term was coined by [22], but the techniques involved have been discovered and applied much earlier, as [23, Ch.5.5] discusses.

By applying the chain rule to the gradient w.r.t. some $\tilde{\mathbf{W}}^{(l)}$, we get

$$\frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{W}}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(L)}} \frac{\partial \mathbf{h}^{(L)}}{\partial \tilde{\mathbf{W}}^{(l)}}. \tag{57}$$

While the first factor is merely the derivative of the energy function w.r.t. the network's output, which is problem-specific, the second term, the gradient w.r.t. our model, is completely independent of the energy function.

But now we can apply the chain rule multiple times:

$$\begin{aligned} \frac{\partial \mathbf{h}^{(L)}}{\partial \tilde{\mathbf{W}}^{(l)}} &= \frac{\partial \mathbf{h}^{(L)}}{\partial \mathbf{h}^{(L-1)}} \frac{\partial \mathbf{h}^{(L-1)}}{\partial \tilde{\mathbf{W}}^{(l)}} && (58)\\ &= \dots && (59)\\ &= \left(\prod_{k=l}^{L-1} \frac{\partial \mathbf{h}^{(k+1)}}{\partial \mathbf{h}^{(k)}} \right) \frac{\partial \mathbf{h}^{(l)}}{\partial \tilde{\mathbf{W}}^{(l)}} && (60) \end{aligned}$$

It becomes apparent that, when taking the derivate w.r.t. $\tilde{\mathbf{W}}^{(l-1)}$, we can reuse many of the components from Eq. (60). In general, we apply the Dynamic Programming principle and start from $l = L$ to save a lot of redundant computations.

A.1.2 *Recurrent Neural Networks*

As we have seen in the previous section, Feed-Forward Neural Networks (NNs) are a powerful tool for function approximation on some fixed input space. When concerned with robot trajectories, we usually find time series of differing length on some fixed space. We need a model that can (i) deal with input of differing length, and (ii) exploit temporal dependencies, preferably with similar provable properties as NNs. Recurrent Neural Networks (RNNs) turn out to be such a tool. To understand them, we will revisit Eq. (53) and Eq. (54). We add a temporal component:

$$\begin{aligned} \mathbf{h}_t^{(l)} &= f^{(l)}\left(\mathbf{h}_t^{(l-1)} \mathbf{W}^{(l)} + \mathbf{h}_{t-1}^{(l)} \mathbf{W}_{\text{rec}}^{(l)} + \mathbf{b}^{(l)} \right) && (61)\\ \mathbf{h}_t^{(0)} &= \mathbf{x}_t && (62)\\ \mathbf{h}_0^{(l)} &= \mathbf{b}_{\text{init}}^{(l)} && (63)\\ \mathbf{h}_t^{(L)} &= f^{(l)}\left(\mathbf{h}_t^{(L-1)} \mathbf{W}^{(L)} + \mathbf{b}^{(L)} \right) && (64) \end{aligned}$$

With the exception of the last layer, the output of layer l at time t depends not only on the output of the previous layer $l-1$ as with

non-recurrent NNs, but also on the output of the previous time step $t-1$ at that layer. Each RNN time step is a normal NN with 'sideways' temporal input.

The universal approximation capability is, thus, leveraged to sequences. Indeed, RNNs are Turing-complete [25] and, under mild conditions, universally approximate measurable mappings of sequences to sequences [8].

The Back-Propagation principle encountered in the previous section can also be applied to RNNs, it is then called *Back-Propagation Through Time*, [28].

The same applies for fast dropout, which has been generalized to RNNs in [3].

A.1.2.1 *Variants: Bidirectional and Long Short-Term Memory Networks*

The research on Recurrent Neural Networks (RNNs) has given rise to a multitude of variants. We will briefly discuss two important variants that we will use throughout this thesis.

BIDIRECTIONAL RNNS As seen in Eq. (61), standard RNNs only look at the past input samples. However, in some settings (one of which is mentioned in Section 2.2.1) it can make sense to look at future samples as well. Here, bidirectional RNNs come into play. Proposed by [24], they consist of two individual RNNs of identical size, one of which is working on the reversed input. The output of a hidden layer of the bidirectional RNN is then a combination (usually the sum) of the outputs of the two individual networks at that time step and layer.

LONG SHORT-TERM MEMORY One flaw discovered by the research community is that RNNs struggle to model long-time dependencies. This has to do with the multiplicative error propagation. One successful attempt to overcome this deficiency are so called Long Short-Term Memory (LSTM) networks, [11]. LSTM is a transfer function which is parametrized with a data-dependent state. Depending on the input, it can 'decide' whether to apply or change the state. The decision mechanism is a differentiable version of a boolean decision, implemented by a sigmoid function that saturates at 0 and 1, corresponding to `True` and `False`. Since LSTM can be viewed as a specific, powerful transfer function rather than a stand-alone network architecture, it can easily be combined with, e. g., bidirectional NNs.

A.2 ESTIMATING THE MARGINAL DATA LIKELIHOOD

In Section 2.1.5, we have found a method to approximate the true, yet unknown data likelihood through a variational approximation that yields a lower bound.

As both papers [14, 21] show, one can derive a Monte Carlo estimator of the true data likelihood. Though this is not sufficiently efficient from a computational point of view, it helps evaluating the goodness of fit of the variational approximation.

The derivation goes as follows:

$$p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \, d\mathbf{z} \qquad \left[= \mathbb{E}_{p(\mathbf{z})}[p(\mathbf{x} \mid \mathbf{z})]\right] \tag{65}$$

$$= \int \frac{p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} q_\phi(\mathbf{z} \mid \mathbf{x}) \, d\mathbf{z} \tag{66}$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})}\right] \tag{67}$$

$$\approx \frac{1}{S} \sum_{s=1}^{S} \frac{p(\mathbf{x} \mid \mathbf{z}_s) p(\mathbf{z}_s)}{q_\phi(\mathbf{z}_s \mid \mathbf{x})}, \qquad \mathbf{z}_s \sim q_\phi(\mathbf{z} \mid \mathbf{x}) \tag{68}$$

The notation is as previously, i. e., $p(\mathbf{z})$ is the latent prior, $p(\mathbf{x} \mid \mathbf{z})$ is the output distribution and $q_\phi(\mathbf{z}_s \mid \mathbf{x})$ denotes the recognition model, the variational approximation of the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$.

As indicated in the first line, drawing from the prior would yield a valid Monte Carlo estimate. However, drawing from the recognition model in the spirit of importance sampling gives a lower variance fit—the samples drawn fit the data better.

This result can easily be extended to time series and was omitted for simplicity of notation.

From a computational point of view, we are actually interested in estimating the log-likelihood

$$\ln p(\mathbf{x}) \approx \ln\left(\frac{1}{S} \sum_{s=1}^{S} \frac{p(\mathbf{x} \mid \mathbf{z}_s) p(\mathbf{z}_s)}{q_\phi(\mathbf{z}_s \mid \mathbf{x})}\right) \tag{69}$$

$$= \ln\left(\sum_{s=1}^{S} \frac{p(\mathbf{x} \mid \mathbf{z}_s) p(\mathbf{z}_s)}{q_\phi(\mathbf{z}_s \mid \mathbf{x})}\right) - \ln S \tag{70}$$

With the logarithm outside the sum in Eq. (70), we recognize the setting of the so-called LogSumExp trick:

$$\ln \sum_{s=1}^{S} \exp(\xi_s) = a + \ln \sum_{s=1}^{S} \exp(\xi_s - a), \qquad a := \max_s \xi_s \tag{71}$$

The latter is numerically favorable because $0 \leqslant \exp(\xi_s - a) \leqslant 1$, which prevents overflow because the exponentiation does not explode, and underflow with negative effects of the logarithm, because at least one of the exponentiations (the one corresponding to the maximum) yields 1, which leads to higher precision in the calculation of the logarithm.

The LogSumExp is straightforwardly applicable by identifying

$$\xi_s = \ln p(\mathbf{x} \mid \mathbf{z}_s) + \ln p(\mathbf{z}_s) - \ln q_\phi(\mathbf{z}_s \mid \mathbf{x}), \tag{72}$$

where all summands are core quantities of VAEs and STORNs.

A.3 EXPERIMENTS: ALTERNATIVE LATENT DISTRIBUTIONS

The inverse transform method is fed by uniform random variables. VAEs, however, start with Gaussian variables. We would like to skip implicitly transforming the Gaussian variables into uniform variables, i. e., choose a family of parametrized distributions that generalize the uniform distribution.

The Beta distribution with parameters a and b does so—in the case of $a, b = 1$ it coincides with the standard uniform distribution. Furthermore, for any pair of parameters, it is defined on the standard unit interval. In addition to these features, for parameters $a, b < 1$, the probability density function (pdf) of the corresponding Beta distribution is U-shaped and could thus serve to approximate binarized/discretized latent variables.

In order to learn distributions through NNs, we need to derive gradients of all parameters involved. In particular, we need derivatives of the generation of the random latent variables w.r.t. the parameters of the distribution.

We will now derive the backpropagation equations for beta distributions, closely following the suggestions of [21] in notation and procedure. We are interested in the gradient

$$\nabla_\theta \mathbb{E}_{p(x|\theta)}[f(x)], \tag{73}$$

which can be rewritten as

$$-\mathbb{E}_{p(x|\theta)}[\nabla_x[B(x)f(x)]] \tag{74}$$

with a suitable function $B(x)$. In the paper, a general formula for deriving B for an exponential family distribution of the form

$$p(x \mid \theta) = h(x)\exp\big(\eta(\theta)^T \phi(x) - A(\theta)\big) \tag{75}$$

is given by

$$B(x) = \frac{\nabla_\theta \eta(\theta)\phi(x) - \nabla_\theta A(\theta)}{\nabla_x \log h(x) + \eta(\theta)^T \nabla_x \phi(x)}. \tag{76}$$

This formula stems from the observation that

$$\nabla_\theta p(x \mid \theta) = p(x \mid \theta) \underbrace{[\nabla_\theta \eta(\theta)\phi(x) - \nabla_\theta A(\theta)]}_{=: C(x)}, \tag{77}$$

$$\nabla_x p(x \mid \theta) = p(x \mid \theta) \underbrace{\big[\nabla_x \log h(x) + \eta(\theta)^T \nabla_x \phi(x)\big]}_{=: D(x)}, \tag{78}$$

so that

$$\begin{aligned} \nabla_\theta p(x \mid \theta) &= p(x \mid \theta) C(x) \\ &= p(x \mid \theta) D(x) B(x) = \nabla_x [p(x \mid \theta)] B(x). \end{aligned}$$

This is the reparametrization trick: instead of taking the derivative w.r.t. the random variable x, we have reparametrized such that we can take the derivative w.r.t. the parameters θ.

The beta distribution with parameters $a, b > 0$ belongs to the exponential family and has the pdf

$$p(x \mid a, b) = \frac{1}{B(a, b)} x^{a-1}(1-x)^{b-1}, \tag{79}$$

where $x \in [0, 1]$ and $B(a, b)$ is the Beta function.

In order to derive the suitable B for gradients w.r.t. to the first parameter a, we can match patterns between Eq. (75) and Eq. (79) and arrive at

$$h(x) = (1-x)^{b-1}, \tag{80}$$
$$\eta(\theta) = a - 1, \tag{81}$$
$$\phi(x) = \log(x), \tag{82}$$
$$A(\theta) = \log B(a, b). \tag{83}$$

Inserting into Eq. (76), we arrive at

$$B_{a,b}(x) = \frac{\log(x) - \Psi(a) + \Psi(a+b)}{\frac{a-1}{x} - \frac{b-1}{1-x}} \tag{84}$$
$$= \frac{x(1-x)(\log(x) - \Psi(a) + \Psi(a+b))}{(a-1)(1-x) - (b-1)x}. \tag{85}$$

Here, Ψ is the digamma function, the first derivative of the Γ-function.

The denominator is a linear function in x. Unless $a + b = 2$, its root is $x = (a-1)/(a+b-2)$ (this corresponds to the mode or, in case of U-shaped pdfs, the anti-mode of the Beta distribution). For most combinations of positive parameters a and b, this root lies within the unit interval, the domain of the Beta distribution. Since we need to evaluate $B_{a,b}$ for Beta samples in order to calculate the (stochastic) gradient, we yield a singularity in the domain of our latent variables.

Numerical experiments show that the evaluation of $B_{a,b}$ suffers from an intolerably high variance. The empirical mean, even for large samples, varies on a scale as large as 10^0.

The previous result is a provable, theoretical drawback of the Beta distribution in the framework of VAEs. Moreover, in practice, Beta distributions turn out to be hard to sample from. Since there is no closed-form expression for the cdf of Beta distributions, one cannot apply the inverse transform method, which is efficiently implementable. Instead, one has to use rejection sampling methods, which are hard to transfer into a parallel computing framework or GPUs.

While the idea of using a Beta distribution seemed promising in the first place, the two drawbacks outlined above render the application impractical. Initially, our goal was to find a suitable extension of the uniform distribution. While the Beta distribution as a member of

the exponential family seemed a somewhat natural choice, there are reasonable alternatives, such as the Kumaraswamy distribution.

The Kumaraswamy (KW) distribution is a distribution on the standard unit interval with two parameters $a, b > 0$. Though not exactly related, it resembles the Beta distribution, while it has some advantages such as efficient sampling and simpler, closed-form expressions for pdf and cdf.

In order to use the KW distribution, we need a couple of quantities associated with this distribution. For $x \in [0, 1]$ and $a, b > 0$, the pdf of a $KW(a, b)$ distribution is

$$p(x \mid a, b) = abx^{a-1}(1-x)^{b-1}. \tag{86}$$

For $x \in [0, 1]$ and $a, b > 0$, the pdf of a $KW(a, b)$ distribution is

$$F(x|a, b) = 1 - (1 - x^a)^b. \tag{87}$$

In order to optimize the shape of the posterior distribution in the variational learning process, we need to optimize the log-likelihood of the KW distribution. Taking the negative log of the pdf above, we yield

$$LL(x \mid a, b) = \ln a + \ln b + (a-1)\ln x + (b-1)\ln(1-x) \tag{88}$$

In the learning process of VAEs, we use the KL between the posterior (which is to be approximated by a KW distribution) and some prior distribution on the latent variables. A suitable, simple prior for the KW distribution is the uniform prior. In this case, we get that the KL boils down to the (negative) entropy of the respective KW distribution. Hence[2]:

$$\begin{aligned} &KL(KW(x \mid a, b) \parallel U(0, 1)) \\ &= \ln(ab) + \left(\frac{a-1}{a}\right)\left(\gamma - \Psi(b) + \frac{1}{b}\right) - \frac{b-1}{b} \end{aligned}$$

Here, γ is the Euler-Mascheroni constant and Ψ is the digamma function, the derivative of the Γ-Function.

An important aspect of using the KW distribution in the variational framework is sampling. Since the cdf is invertible with

$$F^{-1}(x|a, b) = \left(1 - (1-x)^{1/b}\right)^{1/a},$$

we can produce samples $KW(a, b)$ by taking a uniform sample u and applying the inverse. $F^{-1}(u \mid a, b)$ then has the desired distribution.

In early experiments with static data, we found that the KW distribution trains more slowly (both in terms of computation time due to less optimized functions involved than with the very common Gaussian distributions and in terms of iterations until convergence) and yields worse results in terms of the lower bound. We henceforth discarded it and sticked to Gaussian latent distributions.

2 For details on the derivation, cf. [16, 4.11]

BIBLIOGRAPHY

[1] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012. (Cited on page 38.)

[2] Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014. (Cited on pages 1, 19, 20, 24, and 27.)

[3] Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. *arXiv preprint arXiv:1311.0701*, 2013. (Cited on pages 37 and 54.)

[4] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation. (Cited on page 38.)

[5] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. (Cited on page 51.)

[6] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988. (Cited on page 12.)

[7] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C. Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *CoRR*, abs/1506.02216, 2015. URL `http://arxiv.org/abs/1506.02216`. (Cited on pages 4 and 25.)

[8] Barbara Hammer. On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1):107–123, 2000. (Cited on pages 20 and 54.)

[9] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on Neural Networks*, pages 593–605. IEEE, 1989. (Cited on pages 12 and 52.)

[10] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. (Cited on page 52.)

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. (Cited on page 54.)

[12] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. (Cited on pages 12 and 52.)

[13] J.Bayer, C.Osendorfer, S. Diot-Girard, T. RÃŒckstiess, and Sebastian Urban. climin - a pythonic framework for gradient-based function optimization. `https://github.com/BRML/climin`, 2015. (Cited on page 38.)

[14] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. (Cited on pages 9, 17, 24, and 55.)

[15] Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Amer. Math. Soc. Transl*, 28:55–59, 1963. (Cited on pages 12 and 52.)

[16] J.V. Michalowicz, J.M. Nichols, and F. Bucholtz. *Handbook of Differential Entropy*. CRC Press, 2013. ISBN 9781466583177. URL `https://books.google.de/books?id=k2fOBQAAQBAJ`. (Cited on page 58.)

[17] Zoltán Á Milacski, Marvin Ludersdorfer, András Lorincz, and Patrick van der Smagt. Robust detection of anomalies via sparse methods. In *Proc. 22nd Int. Conf. on Neural Information Processing (ICONIP 2015)*, 2015. (Cited on page 5.)

[18] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029. (Cited on pages 10 and 11.)

[19] Judea Pearl. *Causality*. Cambridge university press, 2009. (Cited on page 10.)

[20] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99: 215–249, 2014. (Cited on pages 3, 5, 7, and 29.)

[21] Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep

generative models. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1278–1286. JMLR Workshop and Conference Proceedings, 2014. URL `http://jmlr.org/proceedings/papers/v32/rezende14.pdf`. (Cited on pages 9, 17, 24, 55, and 56.)

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3, 1988. (Cited on page 53.)

[23] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. (Cited on pages 51 and 53.)

[24] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11): 2673–2681, 1997. (Cited on page 54.)

[25] Hava T Siegelmann and Eduardo D Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991. (Cited on pages 20 and 54.)

[26] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015. (Cited on page 49.)

[27] Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 118–126, 2013. (Cited on pages 37 and 52.)

[28] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. (Cited on page 54.)

[29] Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. (Cited on page 38.)

TRITA -MAT-E 2015:90
ISRN -KTH/MAT/E--15/90--SE