UPPSALA
UNIVERSITET

IT Licentiate theses
2019-007

Deep Learning Applied to System Identification: A Probabilistic Approach

Carl Andersson

UPPSALA UNIVERSITY
Department of Information Technology

UPPSALA
UNIVERSITET

Deep Learning Applied to System Identification: A Probabilistic Approach

Carl Andersson
Carl.Andersson@it.uu.se

December 2019

Division of Systems And Control
Department of Information Technology
Uppsala University
Box 337
SE-751 05 Uppsala
Sweden

http://www.it.uu.se/

Dissertation for the degree of Licentiate of Philosophy in Electrical Engineering with specialization in Signal Processing

© Carl Andersson 2019
ISSN 1404-5117
Printed by the Department of Information Technology, Uppsala University, Sweden

Abstract

Machine learning has been applied to sequential data for a long time in the field of system identification. As deep learning grew under the late 00's machine learning was again applied to sequential data but from a new angle, not utilizing much of the knowledge from system identification. Likewise, the field of system identification has yet to adopt many of the recent advancements in deep learning. This thesis is a response to that. It introduces the field of deep learning in a probabilistic machine learning setting for problems known from system identification.

Our goal for sequential modeling within the scope of this thesis is to obtain a model with good predictive and/or generative capabilities. The motivation behind this is that such a model can then be used in other areas, such as control or reinforcement learning. The model could also be used as a stepping stone for machine learning problems or for pure recreational purposes.

Paper I and Paper II focus on how to apply deep learning to common system identification problems. Paper I introduces a novel way of regularizing the impulse response estimator for a system. In contrast to previous methods using Gaussian processes for this regularization we propose to parameterize the regularization with a neural network and train this using a large dataset. Paper II introduces deep learning and many of its core concepts for a system identification audience. In the paper we also evaluate several contemporary deep learning models on standard system identification benchmarks. Paper III is the odd fish in the collection in that it focuses on the mathematical formulation and evaluation of calibration in classification especially for deep neural network. The paper proposes a new formalized notation for calibration and some novel ideas for evaluation of calibration. It also provides some experimental results on calibration evaluation.

Acknowledgments

First of all I want to thank my two supervisors Thomas Schön and Niklas Wahlström for support and encouragement during these past three years. Further on I want to thank Anna Wigren and Daniel Gedon for proofreading and useful comments on the thesis and David Widmann for the idea of a neater version for the proof in the Appendix. Finally I want to thank all coauthors on the papers included in this thesis: Antônio Riberio, Koen Tiels, David Widmann and Juozas Vaicenavicius.

This research was financially supported by the project *Learning flexible models for nonlinear dynamics* (contract number: 2017-03807), funded by the Swedish Research Council.

List of Papers

This thesis is based on the following papers

I C. Andersson, N. Wahlström, and T. B. Schön. "Data-Driven Impulse Response Regularization via Deep Learning". In: *18th IFAC Symposium on System Identification (SYSID).* Stockholm, Sweden, 2018

II C. Andersson, A. L. Ribeiro, K. Tiels, N. Wahlström, and T. B. Schön. "Deep convolutional networks in system identification". In: *Proceedings of the 58th IEEE Conference on Decision and Control (CDC).* Nice, France, 2019

III J. Vaicenavicius, D. Widmann, C. Andersson, F. Lindsten, J. Roll, and T. B. Schön. "Evaluating model calibration in classification". In: *Proceedings of AISTATS.* PMLR, 2019

Contents

Chapter 1

Introduction

Machine learning is booming, both in research and in industry. There are self-driving cars, computers beating world champions in strategic games such Go [51] and Starcraft 2 [59] and artificial videos of Barack Obama[1], synthesizing the ex-president's speech and appearance. This introductory chapter will introduce the main concepts behind this boom with an application focus on sequential data and simultaneously introduce the mathematical notation used. The reader that is familiar with machine learning can safely skip this chapter and start directly on Chapter 2.

1.1 What is Machine Learning?

Machine Learning, as a concept, is roughly the intersection between Artificial Intelligence (AI) and the field of statistics. Thus, to explain machine learning, we first need to define these two areas.

Artificial intelligence is the entire field of research to get machines, i.e. computers, to perform a chosen task that requires some level of reasoning. Many think that AI is still in the future but it is actually already all around us, for example

- when you ask your navigator for directions, the AI finds the shortest path between two points on a map (i.e. A*-algorithm [23])
- when you type a search string into google, the AI filters out a list of websites given a query
- when you visit a website and is presented with ads, the AI uses your browser history to maximize the chance of a click

[1]`https://www.youtube.com/watch?v=cQ54GDm1eL0`

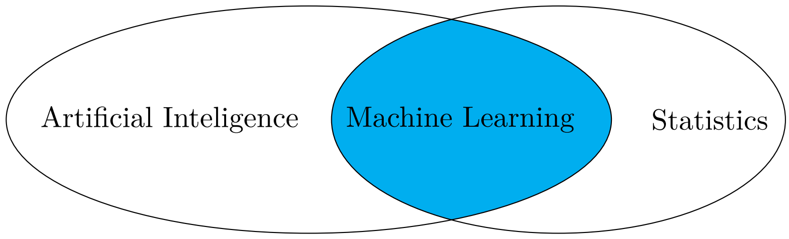


Figure 1.1: Machine learning is the intersection between Artificial Intelligence and Statistics.

Statistics, on the other hand, is centered around extracting information from data, to draw conclusions and to compute accurate predictions given a set of examples. Additionally, given even more examples (more data), the conclusion can be refined and the predictions should be more accurate.

Machine learning is at the intersection between statistics and artificial intelligence (Figure 1.1), where the goal of the machine requires reasoning and the solution is not an explicitly programmed behavior. Instead the behavior is learned by mimicking examples and by generalizing from the examples using a statistical framework.

Looking back at the previously mentioned tasks. All of them require some form of reasoning to be completed, thus they are all AI. However, only the two last will benefit from more examples. The navigator task will not improve by being presented with more trips, it will always propose the shortest trip.

Mitchell [43] phrased this in a slightly different way which is nowadays considered to be the definition of machine learning as

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

This learning process of the program or the model is often referred to as *training* and the data set (the experience) used is likewise called *training data*.

1.1.1 Supervised learning

A common setup in machine learning is to find a prediction of an output variable $y \in \mathcal{Y}$ (often called label) given observed input variables (often called input features) $x \in \mathcal{X}$. This is done by training a model with data from a set of n pairs $\{x^{(i)}, y^{(i)}\}_{i=1}^{n} = \mathcal{D}_S : x^{(i)} \in \mathcal{X}, y^{(i)} \in \mathcal{Y}$ where $x^{(i)}$ and $y^{(i)}$ are independently sampled from the true joint distribution $\pi(x, y)$. Both the labels and the input features could be in any arbitrary dimensional space. This problem setup is known as *supervised learning*, since every input have a matching output.

In this thesis we will consider models that corresponds to distributions, i.e. that are *probabilistic*. In the supervised learning case, this is often corresponds to the *predictive distribution*, $p(y \,|\, x)$. The goal is thus to approximate the true predictive distribution, $\pi(y \,|\, x)$, with $p(y \,|\, x)$ using the data set $\mathcal{D}_S$. Depending on if y is quantitative (i.e. real valued) or qualitative (i.e. classes) the supervised learning task is of either *regression* or *classification* type, respectively.

1.1.2 Unsupervised and semi-supervised learning

On the other hand, if the observed input features do not have a corresponding output variable y (i.e. the features are unlabeled) it is not possible to formulate a problem as a supervised learning problem. Instead, one is usually interested in some property of the features for a data set $\{x_i\}_{i=1}^n = \mathcal{D}_U : x_i \sim \pi(x)$, known as *unsupervised learning*.

In the probabilistic framework we can express this as finding an approximation with a *generative* model, $p(x)$, known as *density estimation*. A common practice when modeling such a distribution is to introduce a latent variable z that can explain some of the variability in the data although it is not observed. The model is thus altered to $p(x) = \mathbb{E}\, p(x \,|\, z)$ where the expectation is over z. This model can then be used for organize the data into groups where data samples that end up in the same group resembles each other in some way. This process is known as *clustering*. The model can also be used to artificially produce new examples.

If we have a large unlabeled data set $\mathcal{D}_U$ and a smaller labeled data set labeled $\mathcal{D}_S$ we can combine these two learning methodologies into *semi-supervised learning*. The large unsupervised data set can then be used to enhance the supervised learning by first clustering all data then use the small labeled dataset to label the clusters.

1.1.3 Reinforcement learning

Another type of problem is a setting where the goal is to perform a series of actions a_t that maximizes a cumulative reward $\sum r_t$ given an initial state x_0. A process known as *reinforcement learning*. After the first action is performed the state is propagated to a new state, x_1, (depending on the action) and a reward, r_1, (that also depends on the action) is received. Given this new state, a new action is chosen and the process continues.

In the most general case, both the state propagation, $p(x_{t+1} \,|\, x_t, a_t)$, and the function that describes that reward for an action and a state, the *reward function* $p(r_t \,|\, x_t, a_t)$, are unknown. Note that the performed action not only affect the reward but also the state propagation and thus also future rewards. Therefore, an action that seems good now can be suboptimal for the cumulative reward. As if this was not hard enough, in the common reinforcement setup, the algorithm, that is used for the problem, is also responsible for collecting the training data through repeated experiments and recording the reward received, while, at the same time, maximizing the cumulative reward during the data collection. This makes reinforcement learning to arguably one of the toughest problems in machine learning. For more information about reinforcement learning see [56] or [8].

1.2 Deep learning

This thesis will focus on parametric models, although many kinds of models can be used to fit a distribution, e.g. Gaussian Processes [46]. The class of parametric models is still very large and how one chooses to parameterize a distribution may affect properties of the training and can also exploit structure in the data that makes the model perform better with fewer examples. For example, using smaller handcrafted building blocks to extract more general features from the input features have been shown to vastly increase the performance [20]. However, designing such features can be both time consuming and requires a lot of domain knowledge. Deep Learning (DL) or Deep Neural Networks (DNN) is a specific way to parameterize a function that have shown to be very effective. In principal, a deep neural network parameterization enforces a structure that recursively extracts features from the input data. A key principle in deep learning is that the feature extractors are found by the algorithm by itself, without human intervention, which enables application to new fields with little domain knowledge.

Compared to other methods of machine learning, Deep Learning have shown to benefit more from larger data sets. This, in conjunction with increasing computational power, is the foundation for many of the recent advancements. Chapter 4 will cover deep learning at a much *deeper* level.

1.3 Sequence modeling

In short, sequence modeling is machine learning applied to sequential data. The sequential data can be any data that has a natural sequential order, for example signals, speech, text or music. It can be of either supervised or unsupervised type where supervised sequential modeling can be further divided into three different variants: one-to-sequence, sequence-to-one, and sequence-to-sequence. One-to-sequence takes some (non-sequential) data as input and a sequence as output (see Section 1.3.1). Sequence-to-one takes a sequential input and produces an (non-sequential) output, for example classification of sequences (see Section 1.3.2). Sequence-to-sequence takes a sequence and produces a sequence. This type can further be divided into two groups, either the whole input affect the whole output, for example translation of a sentence from English to French. Alternatively, the input and output have a causal relationship to uphold, for example a control signal as input and a position of a robot as output where position of the robot at some time point can not affected by a control signal that is received after this time point.

Sequential data can be divided into two groups: data with long memory and data with short memory. Long memory means that correlations between sequential data points that are far apart (in the sequential order) are substantial and can not be neglected. Examples of this is text, speech and music where a word in the beginning of a text can be strongly correlated with a word in the end or the chorus of a song that is repeated through the whole song. Short memory on the other hand do not have this property of far apart correlations, linear dynamical systems and close to linear systems often, but not always, exhibit short memory.

System identification [40] is field very related to sequence modeling that is substantially older. It also applies machine learning to sequential data but traditionally the data and the models employed have had short memory. Even though the fields are very similar the cross communication between them has been limited. This goes in both directions, where a lot of older research done in system identification is ignored by the much younger sequence modeling community, while the current trends of machine learning still has a lot of potential impact on the field of system identification.

1.3.1 Example: Word level language model

One of the biggest differences between sequence modeling before and after the introduction of deep learning is which kind of data the model is successfully applicable to. Before deep learning, applications towards natural language processing was limited to bag-of-words models [42] and various kinds of hidden Markov models [44]. These models suffered from having very short temporal consistency, meaning that the model can not capture the long memory that the data exhibit. Models using deep learning such as Recurrent Neural Networks (RNN) [15, 31] or more specifically Long Short Term Memory (LSTM) [28] (see Section 5.2) have proven to have significantly longer temporal consistency. An example of this, a (one-to-sequence) supervised

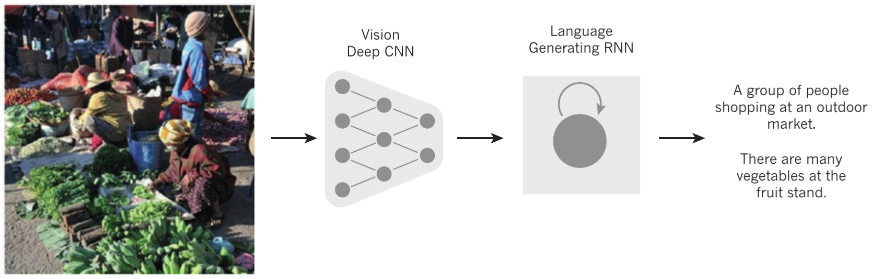

Vision
Deep CNN
Language
Generating RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

Figure 1.2: An image is processed by a deep Convolutional Neural Network (CNN) to produce features. These features are used as input to a Recurrent Neural Network (RNN) model, a sequential model, that produces a sequence of words. The model is probabilistic and the output needs to be sampled to be interpretable, two different sampled captions are shown. Reprinted by permission from Springer Nature: Deep learning, LeCun et al. [38] Copyright (2015).

learning model, is a model proposed by Xu et al. [61]. The trained model can take an arbitrary image and generate a coherent caption for it. The model does this by using a deep Convolutional Neural Network (CNN) [20] to extract features that represent the content of the image. The CNN takes the raw RGB pixel values as input and compresses it down to a, for the model, useful representation of the image. An LSTM model then takes these features as input to generate a sequence of words, i.e. the caption for the image.

The LSTM works by taking the features as a initial input and then produce a probability distribution over the first word of the caption. A word is sampled from this distribution and this sample is used as input to the next iteration of the LSTM together with the state from the previous iteration of the LSTM. This is used to produce a new distribution for the second word and the second word samples form that. This process continues until a special word, known as the stop token (usually a period), is sampled from the LSTM. The resulting word sequence is a sample caption. Note that this is a sample from the distribution of captions that the model represents and rerunning the process will possibly give another result. Figure 1.2 gives an overview of the flow in the model, additionally one can see two different sampled captions that both describe what is happening in the image.

1.3.2 Example: Electrocardiogram Classification

Diagnosing patients with heart rate abnormalities is a task typically reserved for medical doctors. The potential of proposing an initial classification to

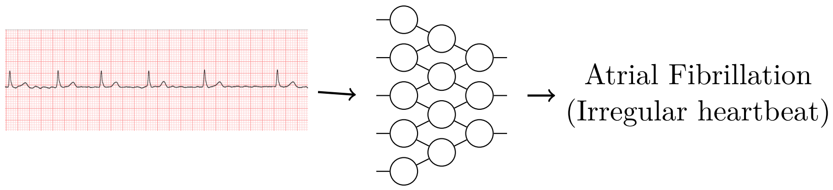


Figure 1.3: For each record that is classified, measurements from 12 different electrodes (each electrode records a signal like the one on the left) are feed to a deep neural network classifier. The network then classifies the record as being in either of 7 different classes (6 abnormalities or normal). Credits to and permission to use from the Telehealth Network of Minas Gerais.

relief the doctors is huge since the number of electrocardiograms (ECG) taken every day increases and doctors time is limited. With the recent advancement in machine learning and growing amount of medical data that is collected every day it is possible to automate some of this process.

In Ribeiro et al. [48] an automatic way of classifying the raw ECG tracings is proposed. The training data consisting of roughly 2 million 7 to 10 seconds long ECG measurements, each classified as being in one out of 6 abnormalities or as being healthy. The model uses a convolutional network of an architecture inspired by ResNet [25], an architecture mostly employed for images but here altered for one dimensional signals. ResNet is a deep architecture that create features through convolutions of the signal. The output features from the convolutions are then used to classify the ECG. Figure 1.3 depicts an overview of the process. The final prediction has an accuracy and specificity comparable or better than the prediction of a medical doctor.

1.3.3 Example: Midi generation

Being able to generate music and use deep learning as a creative tool is more and more becoming reality. A way to achieve a generative music model is to use unsupervised learning to mimic the examples in the training data. The trained model can then be used to generate new songs in the same genre as the music in the training data.

Boulanger-Lewandowski et al. [10] proposes a model for music in midi format (a music format the represents each pressed note at each time instance in a piano piece, like a piano sheet). The model combines an RNN with a restricted Boltzmann machine (RBM) and is depicted in Figure 1.4. Here the RBM is used to create a distribution over the keys of a piano pressed at every time step. This is modeled through an interaction between a hidden state, h_i and a visible state, visualized as the actual piano. The RBM will not be

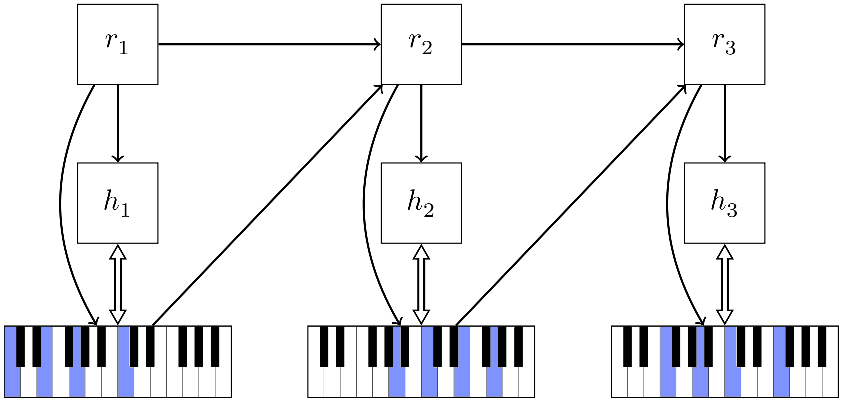


Figure 1.4: A recurrent neural network and a restricted Boltzmann machine used in conjuction to model piano music. The restricted Boltzmann machine is here depicted as the interaction between the hidden state h_i and the visible piano keys. The recurrent neural network takes r_{i-1} and the previous pressed piano keys as input to the next state, r_i.

explained in detail in this thesis, but the interested reader can read about it in Goodfellow et al. [20]. The temporal model is autoregressive i.e. it takes previously visible state as input to produce a predictive distribution for the next timestep. Whilst the RBM is used to model this predictive distribution at the current time step, the RNN is used to create a recurrent state, r_i, that condition the predictive distribution on all the previous outputs.

1.4 Outline

The rest of the thesis will give a deeper introduction to all the concepts given in the introduction. Chapter 2 will introduce the probabilistic notation and some of the basic concepts of machine learning. Chapter 3 will give a more mathematical introduction to sequence modeling. Chapter 4 deepens the introduction and brings up more concepts regarding deep learning. Finally, Chapter 5 aims at combining deep learning and sequence modeling. The end goal of the thesis are two. Firstly the thesis aims at giving an introduction to deep learning in general and secondly it aims at introducing deep learning models used for sequential data.

1.5 Included papers

Paper I: Data-driven impulse response regularization via deep learning

> C. Andersson, N. Wahlström, and T. B. Schön. "Data-Driven Impulse Response Regularization via Deep Learning". In: *18th IFAC Symposium on System Identification (SYSID)*. Stockholm, Sweden, 2018

Summary: In this paper we presents a novel idea on how to construct a prior for the finite impulse response of a system through deep learning. This prior is then be used to regularize an estimator of the finite impulse response. The main idea has inspiration from the impulse response estimations regularized with Gaussian Processes. In the previous work the Gaussian Process as a prior for the parameters in the impulse response estimation. In this paper however instead of using a Gaussian Process we learn a prior that we model with deep learning.

Contribution: The idea originated from Niklas Wahlström but the majority of the work, implementation and writing is made by me.

Paper II: Deep convolutional networks in classification

> C. Andersson, A. L. Ribeiro, K. Tiels, N. Wahlström, and T. B. Schön. "Deep convolutional networks in system identification". In: *Proceedings of the 58th IEEE Conference on Decision and Control (CDC)*. Nice, France, 2019

Summary: Many results from deep learning are yet to impact system identification. This paper tries to bridge this and experiments with known good models from deep learning and apply them to typical system identification problems. Additionally the paper investigates the relationship between the models from deep learning and the models known in system identification.

Contribution: The general idea for the paper originated from Thomas Schön while the idea for the connection to system identification via Volterra series was Koen Tiels. He is also the author of that part of the paper. The rest of writing is jointly made by me, Antônio Riberio, Niklas Wahlström. The implementation is done by me and Antônio.

Paper III: Evaluating model calibration in classification

> J. Vaicenavicius, D. Widmann, C. Andersson, F. Lindsten, J. Roll, and T. B. Schön. "Evaluating model calibration in classification". In: *Proceedings of AISTATS*. PMLR, 2019

Summary: A calibrated model has nothing to do with the accuracy of the model but how accurately it predicts the probability it makes the actuate prediction. This paper covers calibration for classification models, how to formalize the concept in a rigorous way and how to evaluate calibration or rather how the current standard of evaluation calibration is insufficient.

Contribution: The idea of this paper grew from a discussion between me, David Widmann and Juozas Vaicenavicius. The formalized notion is a product of Juozas and David while the theorems are results from discussions in between the three of us. The implementation is done by me and David.

Chapter 2

Probabilistic models

This chapter will introduce the probabilistic framework in a supervised learning setting which also translates to unsupervised learning. It will introduce how we find the model and give an example of two common problem setups. Further on we will describe how we evaluate the model and how to choose model complexity.

2.1 Bayesian or frequentist

Probabilistic models have two major opposing branches, *Bayesian* and *frequentist*, which corresponds to two different philosophical world interpretations. Discussions whether the Bayesian or the frequentist world view is the correct way of viewing the world (known as the Bayesian versus frequentist debate) has been raging for the last century and is still open.

In the supervised case (and analogously in the unsupervised case) both the Bayesian and the frequentist has the aim to find an approximate distribution $p(y \mid x)$ (the model) to a true distribution by $\pi(y \mid x)$ given only the data set $\mathcal{D}_S$. This model, in both cases, includes a parametric distribution called the likelihood function, denoted $p(y \mid x, \theta)$, parameterized with a set of parameters $\theta \in \Theta$. The frameworks differ in how they interpret the parameters. The frequentist assumes that the parameters are deterministic, thus the best parameter value are those that maximize the likelihood of the data. This maximum likelihood (ML) (or equivalently maximum log-likelihood) estimate of the parameters can be formalized as,

$$\hat{\theta} = \arg\max_{\theta} p(\mathcal{D}_S \mid \theta) = \arg\max_{\theta} \log p(\mathcal{D}_S \mid \theta) \tag{2.1}$$

where $p(\mathcal{D}_S \mid \theta)$ denotes $\prod_i p(y = y^{(i)} \mid x = x^{(i)}, \theta)$. From here on we will sloppily abuse the notation $p(y^{(i)} \mid x^{(i)}, \theta)$ to mean $p(y = y^{(i)} \mid x = x^{(i)}, \theta)$

when it is clear from the context what is meant. This estimate can then be used to form the approximate distribution $p(y \mid x, \hat{\theta})$. We call this object that we are optimizing, the *objective*, denoted $\mathcal{L}$ i.e. in this case $\mathcal{L}(\mathcal{D}_S, \theta) = \log p(\mathcal{D}_S \mid \theta)$.

The Bayesian instead assumes that the parameters are random variables that needs to be integrated out,

$$p(y \mid x) = \int_{\Theta} p(y \mid x, \theta) p(\theta \mid \mathcal{D}_S) d\theta \tag{2.2}$$

The posterior distribution, here denoted $p(\theta \mid \mathcal{D}_S) = \prod_i p(\theta \mid y^{(i)}, x^{(i)})$ and with that, the prior $p(\theta)$, is central in the Bayesian view and the two are related through Bayes rule,

$$p(\theta \mid \mathcal{D}_S) = \frac{p(\mathcal{D}_S \mid \theta) p(\theta)}{p(\mathcal{D}_S)} \tag{2.3}$$

The prior is free to choose and should correspond to some prior belief of what the parameters could be. The normalizing constant is calculated by integrating out the parameters,

$$p(\mathcal{D}_S) = \int_{\Theta} p(\mathcal{D}_S \mid \theta) p(\theta) d\theta \tag{2.4}$$

This integral and the integral in Equation (2.2) are generally very challenging tasks since the integral in many cases does not have any analytical solution. In practice we have to resort to approximations where a common approximation is Monte Carlo sampling (or Bayesian Variational Inference to mention an alternative). Alternatively, one can consider the Maximum A Posteriori (MAP) estimate,

$$\hat{\theta} = \arg\max_{\theta} \, p(\theta \mid \mathcal{D}_S) = \arg\max_{\theta} \, \log p(\mathcal{D}_S \mid \theta) + \log p(\theta). \tag{2.5}$$

Under some circumstances the ML estimate and the MAP estimate are equivalent. More on this in Section 2.4.

2.2 Regression and classification

As an example of the Frequentist approach we consider two common problems in supervised learning. Supervised learning problems can be divided into two groups, regression and classification. A regression problem is a problem where the goal is to predict a real valued or quantitative variable given some features, e.g. predicting the optimal radiation dosage to treat a cancer patient given the patient age and gender and the cancer type. A classification problem, on the other hand, is a problem where the goal is to predict a class or qualitative variable given some features, e.g. classifying the type of skin cancer given a photograph of a skin lesion (for example see [16]). In the probabilistic framework both regression and classification be expressed with the predictive distribution, $p(y \mid x)$, i.e. the density for output/label variable given the feature variables. Below we provide two examples where two common methods, the least square method and logistic regression, are motivated through the probabilistic frequentist view. In both examples, the predictive distribution is approximated with a parametric distribution, $p(y \mid x, \theta)$.

Example 2.2.1 (Regression)**.** A typical assumption in a probabilistic regression model is that the likelihood is approximated with a normal distribution where the mean is modeled with a parametric function and the variance is set to a constant diagonal matrix. In the radiation dosage example we assume that the optimal dosage is normally distributed with some fixed variance σ^2 and let the mean be a parametric function of the patient age, gender and the cancer type. Using the same notation for the data set as in the previous section (i.e. $\mathcal{D}_S$) and maximizing the likelihood (or equivalently the log-likelihood) under these conditions we get,

$$\arg\max_{\theta} \prod_i \mathcal{N}(y^{(i)} \mid f(x^{(i)}; \theta), \sigma^2) \tag{2.6}$$

For a more concise notation, let us denote the output of the parametric function $f(x^{(i)}; \theta)$ as $\hat{\mu}^{(i)}(\theta)$. Using this notation we can rewrite the ML formulation as,

$$\begin{aligned}
\arg\max_{\theta} \prod_i \mathcal{N}(y^{(i)} \mid \hat{\mu}^{(i)}(\theta), \sigma^2) &= \arg\max_{\theta} \sum_i \log \mathcal{N}(y^{(i)} \mid \hat{\mu}^{(i)}(\theta), \sigma^2) \\
&= \arg\max_{\theta} -\sum_i \frac{1}{2\sigma^2}(y^{(i)} - \hat{\mu}^{(i)}(\theta))^2 \\
&- \frac{1}{2} \log 2\pi\sigma^2 \\
&= \arg\min_{\theta} \sum_i (y^{(i)} - \hat{\mu}^{(i)}(\theta))^2
\end{aligned} \tag{2.7}$$

which can be recognize this as least squares model. Given the optimal parameters $\widehat{\theta}$ we can also fit the variance to the data. □

Example 2.2.2 (Classification)**.** A typical assumption for the classification model is to model the classes with a probabilistic model using a categorical distribution. A categorical distribution is used to model disjoint classes and assigns a probability for each individual class. In the skin cancer example we could consider classification of the cancer as benign or as one of $K-1$ different malign types of cancer summing up to K different classes in total. The model assumes that the cancer is one of these types and the probabilities must thus add up to one. Instead of modeling the probabilities directly it is common to model the distribution with the non-normalized log odds as parametric functions,

$$p(y \mid x; \theta) = \mathrm{Cat}(f(x; \theta)) \tag{2.8}$$

where the class y denotes the type of cancer and x denotes the pixel values from the image. To transform the non-normalized log odds to probabilities we make use of the function,

$$\hat{y}_j = \frac{\exp(f_j(x; \theta))}{\sum_j \exp(f_j(x; \theta))} \tag{2.9}$$

where y_j is the probability of class j. This function is commonly known as the softmax function and maximizing the log likelihood for this model will yield the classical logistic regression setting. $\square$

As seen above, probabilistic models can be used to motivate some common models and costs (least squares and logistic regression). However, the probabilistic formulation can also be extended to other models. For example, let us once more consider the regression problem but let both the mean and the standard deviation be modeled with the parametric functions so that $[\hat{\mu}^{(i)}(\theta), \hat{\sigma}^{(i)}(\theta)] = f(x^{(i)}; \theta)$, which yields the following cost function,

$$\arg\max_\theta \sum_i \log \mathcal{N}(y^{(i)} \mid \hat{\mu}^{(i)}(\theta), (\hat{\sigma}^{(i)}(\theta))^2 \mathrm{I}) =$$
$$\arg\min_\theta \sum_i \frac{1}{2(\hat{\sigma}^{(i)}(\theta))^2}(y^{(i)} - \hat{\mu}^{(i)}(\theta))^2 + \frac{1}{2}\log 2\pi(\hat{\sigma}^{(i)}(\theta))^2. \tag{2.10}$$

Other distributional assumptions will of course yield other cost functions but this framework gives a nice motivation for them.

The Bayesian approach to the same problems is set up in the same way with the same approximation of the likelihood functions. However, instead of solving the optimization problem in Equation (2.1), the parameters are integrated out as in Equation (2.2). Interestingly, these methods do not correspond to any of the commonly known methods, but they are also quite computationally heavy depending on the approximation method and accuracy required for the approximation.

2.3 Overfitting and the bias-variance trade-off

Consider a trained model for a supervised learning problem, the prediction error this model expresses for a new unseen test data point can be divided

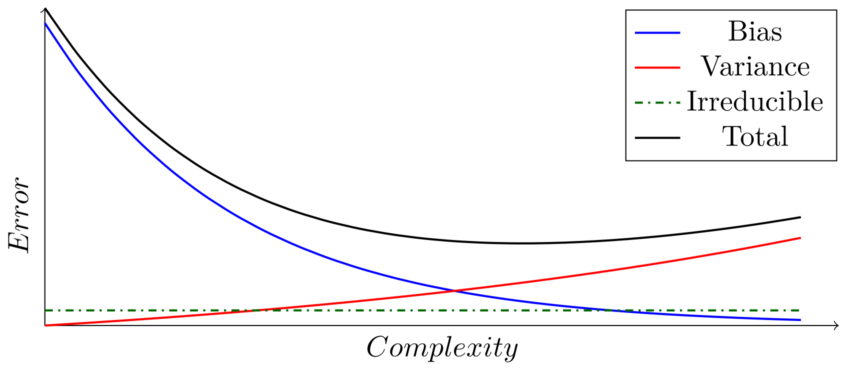


Figure 2.1: A schematic picture of the decomposition of the prediction error into its three components; bias error, variance error, and irreducible error. The decrease of the bias error and the increase of the variance error with increasing complxity is the foundation of the bias-variance tradeoff.

into three parts: a bias error, a variance error, and an irreducible error. The bias error exists due to a too simplistic model or biased predictions due to model assumptions. The variance error is related to the variance in the prediction due the randomness inherent in the training data, e.g. the measurement noise and the limited number of examples. Lastly, the irreducible error is the error that is intrinsic to the problem, i.e. the error that the true predictive distribution gets when predicting.

The prediction error has a number of different causes. First, and perhaps most intuitively, the prediction error can be decreased if the amount of training data is increased. This is because the randomness in the empirical distribution (i.e. the training data) decreases and thus the variance error decreases. Secondly, it is possible to vary the complexity of the model. The complexity of the model is usually related to the number of parameters in it. Increasing the complexity, and thus the number of parameters, decreases the bias error. However, increasing the complexity also increases the variance error as more information is needed to estimate the model. Figure 2.1 depicts the different parts of the prediction error and how it varies with model complexity. Since the bias error decreases with increasing complexity while the variance error increases there exists a minima of the bias-variance trade-off. Increasing the amount of available data will push the minima of the bias-variance trade-off towards more complex models.

This error decomposition is closely related to the concepts of *overfitting* and *underfitting*. An overfitted model follows the data too closely and is not able to generalize to new unseen data. In other words, the model is too complex for the dataset. The opposite is true for an underfitting model,

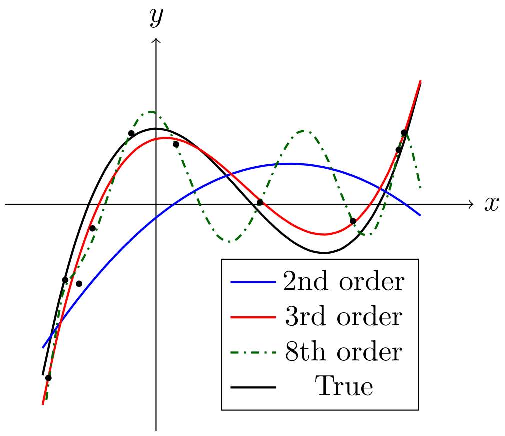


Figure 2.2: An illustration of overfitting and underfitting. The black points are sampled (with noise) from the true 3:rd degree polynomial (black). An estimated 2:nd degree polyniomal (blue) and a 3:rd degree polynomial (red) and an 8:th degree polynomial (green dashed) are also plotted.

where the model is not complex enough and we have a large bias error. An intuitive example of this is curve fitting with a high dimensional polynomial. Figure 2.2 shows an example of overfitting and underfitting. Ten black data points sampled from the true function (black) with some measurement noise. The true function is of degree 3 and is is plotted in black. We try to fit the data points with a 2:nd degree polynomial (blue), a 3:rd degree polynomial (red) and an 8:th degree polynomial (green dashed). We see that the data is best represented with a polynomial of degree 3 while the model of degree 2 is underfitting and the model of degree 8 is overfitting.

Although we see that the 3:rd degree polynomial fits the true curve the bests it is actually the 8:th degree polynomial that fit the data best, i.e. have lower training error. Thus minimizing the training error is not always advantageous as it is not a measure of how well the model will fit new previously unseen data points. Therefore a part of the data set is set aside and not used for training and instead only used for estimating the model performance on new unseen data to measure how well the model generalize. This data is called the *test* data. However, for model selection it might be useful to compare the generalization performance for multiple different models. Since this comparison also can lead to overfitting we choose to set aside yet another part of the training data for model *validation*. We thus have 3 different data sets. Training data used for model optimization, validation data used for model selection and test data used for evaluating the final model performance.

2.4 Regularization

A way to reduce a models tendency to overfit is to *regularize* it. This is done by modifying the cost function, often by introducing an additional term, e.g.

$$\mathcal{L}(\mathcal{D}_S, \theta) + g(\theta) \tag{2.11}$$

where $g(\theta)$ is the regularization term. This term only modifies the objective slightly but gives a model that is less prone to overfit. For the reader who is unfamiliar to the concept regularization we refer to the books of Bishop [9] and Hastie et al. [24]. Section 4.6 in Chapter 4 will also introduce some regularization techniques that are specific for deep learning.

The additional prior term that appear when you compare ML and MAP (Equations (2.1) and (2.5)) can also be viewed as a kind of regularization. As the prior gets less informative, the effect of the regularization term decreases and is reduced to zero for the non informative prior yielding equivalence between ML and MAP.

Example 2.4.1 (Bayesian regularization)**.** Consider the MAP estimate in Equation (2.5) and assume that the prior for the parameters are Gaussian centered around zero. Note that we can introduce a logarithm without affecting the argument for the maximum,

$$\begin{aligned}\hat{\theta} = \arg\max_{\theta}\, p(\mathcal{D}_U \,|\, \theta)p(\theta) &= \arg\max_{\theta}\, \log p(\mathcal{D}_U \,|\, \theta) + \log p(\theta) &(2.12)\\ &= \arg\max_{\theta}\, \log p(\mathcal{D}_U \,|\, \theta) + \frac{1}{2\nu^2}\theta^2 &(2.13)\end{aligned}$$

where ν^2 denotes the variance of the prior. This is equivalent to ML with L^2 regularization with $\frac{1}{2\nu^2}$ as regularization constant. As the variance of this prior increase the prior gets less informative and disappears completely as the variance approaches infinity, which corresponds to the regularization constant approaching zero. □

We can also consider the case where the prior is dependent on an additional set of parameters, λ, so that model looks like,

$$p(y; \lambda) = \int p(y \,|\, \theta)p(\theta; \lambda). \tag{2.14}$$

We can thus form an additional optimization problem to find an optimal value for λ using some set aside data set. This method is called *empirical Bayes.* In Paper I we, instead of optimizing the value of λ, let it be a function of the data, which we model with a neural network.

2.5 Calibration

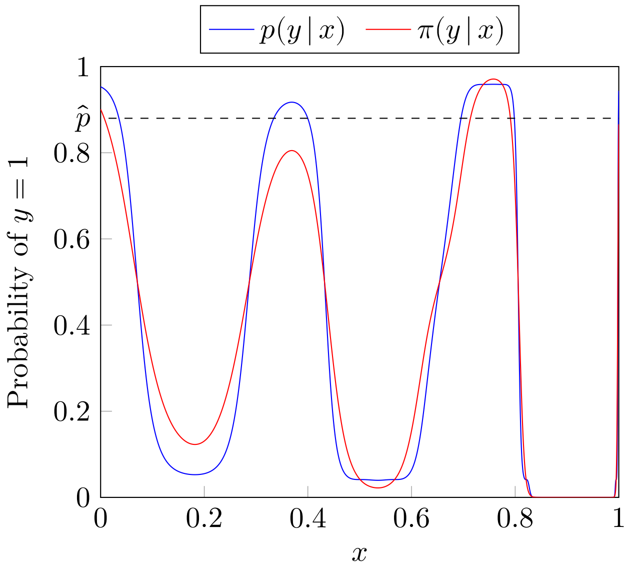
For the probabilistic framework to work it is important that the predictive distribution is *calibrated* [13], i.e. that one can have confidence in the predictive distribution. A calibrated classification model is a model that given that it predicts a class to be true with 70% probability, it should be correct 70% of the time. Or more generally, with $\hat{p}$ being the predicted class density, we can write

$$\pi(y \,|\, p(y \,|\, x, \theta) = \hat{p}) = \hat{p} \tag{2.15}$$

i.e. the (true) class density conditioned on the predicted class density $\hat{p}$ should be equal to the predicted class density. Another way to put this is that the model should never be over- or under-confident, e.g. if the risk of a tumor to be malign is 70%, the model should not predict malign with 90% confidence or 50% for that matter. It is common to visualize calibration with a reliability diagram. Figure 2.3 depicts an example of a model with an associated reliability diagram.

Maximizing the accuracy of a model is not sufficient to achieve a calibrated model. Consider for example a model that systematically predicts benign with 99% confidence even though the probability only is 70%. This model would have the same average accuracy as the calibrated model that accurately predicts 70%. Since the loss chosen in a later stage actually could depend on the actual predicted probability it might be useful to sacrifice some accuracy for a better calibrated model. Alternatively there exist some techniques that can be used to calibrate an uncalibrated model without affecting accuracy, see e.g. [22].

It has been shown that deep learning models in particular are overconfident when it comes to the predictions that they make [22]. This was the main motivation that initiated our work on Paper III which gives a deeper introduction to calibration, why it is necessary and how to evaluate it.



(a)

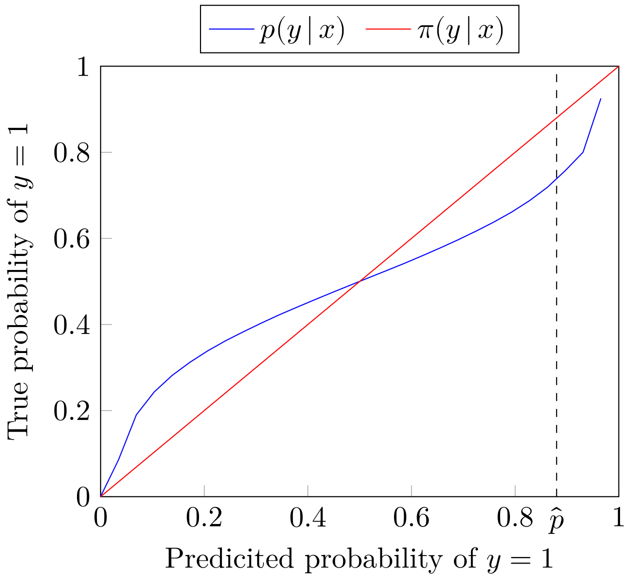


Figure 2.3: The prediction for the model $p(y \mid x)$ and for the true model (a) and the associated reliability diagram (b). When the model $p(y \mid x)$ predicts $y = 1$ with a probability in the region $50 - 100\%$ it is overconfident, see $\hat{p}$. The reliability diagram answers the question, what is the true probability of $y = 1$ when the model predicts $y = 1$ with probability $\hat{p}$.

Chapter 3

System identification in sequential modeling

This chapter will focus on introducing the concepts and models that are common in system identification, but in terms of sequential modeling. It will also serve as a foundation for Chapter 5 where the concepts will be combined with deep learning. This chapter will consider sequential data, $x_{1:T}$, i.e. a sequence of observations at discrete time instances 1 through T. We also want to consider systems with a sequential extraneous input, $u_{1:T}$. This differs from the sequence-to-sequence problem setup in the introduction in that the input also have a causal relationship with the output. The input u_t can thus only affect states and output starting from t and onwards. For a more in depth discussion of models used in system identification see for example Ljung [40].

3.1 State space models

The State Space Model (SSM) builds on the assumption that the observed data $x_{1:T}$ is generated by a sequence of latent variables, $z_{1:T}$, often called states, see Figure 3.1. The states will evolve over time according to some process and given a state, the observed variable corresponding to that state will be independent of all other observed variables.

Consider, for example, a GPS system that tracks the position of an object. Additionally, we consider we have a SSM where the state is the position and velocity of the object and the observed variable be a noisy observation of the position. Given that we actually know the exact state at a time instance, t, it is reasonable to assume that the measured position at a future time instance is independent of all measurements prior to t. On the other hand if we have many observations of the position we can combine these with the help of the

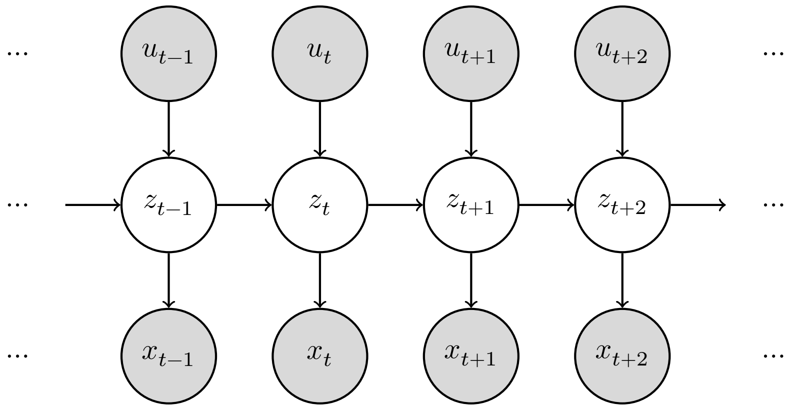


Figure 3.1: A description of the state space model with $z_{1:T}$ as latent variables and $x_{1:T}$ as observed variables with exogenous input $u_{1:T}$.

model to refine the positions. It is also possible to get an estimate of the velocity even though it is never directly observed. The Kalman filter [32] or sequential Monte Carlo[53] are two methods among others to achieve this.

Putting this in a mathematical notion the observed variable x_t is said to be independent of all states $z_i : i \neq t$ and all other observations $x_i : i \neq t$. Additionally the state space model assumes that the state z_{t+1} is independent of all previous states $z_i : i < t$ given z_t known as the *Markov property.* The propagation of the state z_t to z_{t+1} i.e. the distribution $p(z_{t+1} \mid z_t)$ is called *transition* distribution and the distribution of the observation given the state at that time instance, $p(x_t \mid z_t)$, is known as the *emission* or *observation* distribution. This gives rise to the factorization

$$p(x_{1:T}) = \int \prod_{t=1}^{T} p(x_t \mid z_t) p(z_t \mid z_{t-1}) dz_{1:T} \tag{3.1}$$

Consider now also the case with an exogenous input signal. Using the notion of supervised learning this can be expressed according to the factorization

$$p(x_{1:T} \mid u_{1:T}) = \int \prod_{t=1}^{T} p(x_t \mid z_t) p(z_t \mid z_{t-1}, u_t) dz_{1:T} \tag{3.2}$$

where $p(z_t \mid z_{t-1}, u_t)$ is the transition distribution.

Two common state space models are the Linear Gaussian State Space Model (LGSSM) and Hidden Markov Model (HMM). The advantage of these models is that they are analytically solvable which also makes them popular.

Example 3.1.1 (Linear Gaussian state space model)**.** The Linear Gaussian State Space Model uses a transition distribution and an observation distribution that, as the name suggest, are Gaussians. The model can be used for both supervised and unsupervised data. In the unsupervised case, the transition distribution is defined as

$$p(z_{t+1} \mid z_t) = \mathcal{N}(z_{t+1} \mid Az_t, \Sigma_T) \tag{3.3}$$

and the observation distribution is defined as

$$p(x_t \mid z_t) = \mathcal{N}(x_t \mid Cz_t, \Sigma_O) \tag{3.4}$$

where A and C is the transition and the observation matrix respectively and Σ_T and Σ_O are the covariance matrices for the transition and the observation respectively.

With an exogenous input signal we instead have

$$p(z_{t+1} \mid z_t, u_t) = \mathcal{N}(z_{t+1} \mid Az_t + Bu_t, \Sigma_T) \tag{3.5}$$

while the observation distribution is defined as

$$p(x_t \mid z_t) = \mathcal{N}(x_t \mid Cz_t, \Sigma_O) \tag{3.6}$$

which equivalently can be written on state space representation as

$$\begin{aligned} z_t &= Az_{t-1} + Bu_t + \epsilon_t \\ x_t &= Cz_t + \nu_t \end{aligned} \tag{3.7}$$

where ϵ_t and ν_t are i.i.d. and distributed as $\mathcal{N}(0, \Sigma_T)$ and $\mathcal{N}(0, \Sigma_O)$ respectively. □

3.2 Autoregressive models

Another way to model a sequence of data is with an autoregressive model. In its simplest form it can be expressed as

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{1:t-1}) \tag{3.8}$$

A very common approximation to this is to limit the memory of the model by only conditioning only on the k latest outputs,

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{t-k:t-1}) \tag{3.9}$$

A word of caution is that the definition of an autoregressive model differs between the sequence modeling community and the system identification community. In the system identification community an autoregressive model assumes linear dependence on the previous outputs and the corresponding name for this model is a nonlinear autoregressive model[40].

Similarly to the state space model we can also formulate an autoregressive model with an exogenous input signal. Since the input can not affect the future, only inputs up to time point t will affect output y_t like,

$$p(x_{1:T} \,|\, y_{1:T}) = \prod_{t=1}^{T} p(x_t \,|\, x_{1:t-1}, u_{1:t}) \tag{3.10}$$

which similarly can be approximated with a finite memory by only considering the k latest inputs and outputs

$$p(x_{1:T} \,|\, u_{1:T}) = \prod_{t=1}^{T} p(x_t \,|\, x_{t-k:t-1}, u_{t-k+1:t}) \tag{3.11}$$

Example 3.2.1 (Finite impulse response)**.** A special case of the autoregressive model with an exogenous input signal is to additionally assume

$$p(x_{1:T} \,|\, u_{1:T}) = \prod_{t=1}^{T} p(x_t \,|\, u_{t-k+1:t}) \tag{3.12}$$

i.e that the output is independent on past output. This can be a good assumption for a system if there is no process noise and the measurement noise is white. If we model $p(x_t \,|\, u_{t-k+1:t})$ with a Gaussian distribution and let the mean be a linear function of $u_{t-k+1:t}$ we arrive at the finite impulse response model. This model and the estimation of such a model is what is considered in Paper I. $\square$

Chapter 4

Deep learning

Contrary to what many believe, the definition of deep learning does not involve neural networks. A reason for this confusion is probably due to the current hype with successes and breakthroughs of deep learning using just neural networks (e.g [36]). Instead, deep learning is more general and can be described as an hierarchical feature transformation model. The input features to such a model are transformed to new features in a recursive fashion that facilitates both training and generalization of the model. The depth of the model is thus how many such recursive feature transformations are included in the model.

The most common way to construct a deep learning model today however, is to stack multiple layers of neural networks, although some alternatives do exist e.g. deep belief network [26] or deep forest [63]. This has, as mentioned, almost lead to an equivalence between deep learning and neural networks. Neural networks is also what this thesis use to achieve deep learning and thus we have to introduce neural networks.

4.1 Neural networks

In essence (artificial) neural networks are generic function approximators, i.e. a way to parameterize a nonlinear function, which is also how we will use them in this thesis. Section 2.2 introduced the probabilistic machine learning framework which relied on parameterized functions and is where neural networks enters the picture.

Consider the parameterization of, for example, the predictive mean $\widehat{\mu}$ with a input features x, as input (cf. Example 2.2.1). To begin with, consider solving this using a linear regression model as

$$\mu = Wx + b \tag{4.1}$$

where W is a matrix of weights, b is a vector of offsets and μ is the output the model produces. Neural networks are at their core a generalization of this, created by stacking two or more such affine transformations with some kind of nonlinear function, called activation function, in between. One such affine transformation together with the non linear function is one feature transformation building up the depth for deep learning. An example of a depth 3 neural network can thus be

$$\mu = W_3\sigma\left(W_2\sigma\left(W_1x + b_1\right) + b_2\right)v + b_3 \tag{4.2}$$

where $\sigma(\cdot)$ is any scalar activation function operating elementwise. The weights (W_1, W_2 and W_3) and the offsets (b_1, b_2 and b_3) are the parameters of the neural network and will jointly be denoted with θ. Two common choices fro activation function is the sigmoid function or the rectified linear unit (ReLU) function. See [20] for more discussion regarding activation functions.

An affine transformation together with an activation function σ, i.e. $h = \sigma(Wx + b)$, in a deep neural network is called a layer, more specifically this is a *fully connected* layer. The outputs, h, of a layer are called output features or hidden units and the input x to a layer is simply called the input features. Note that the dimensionality of W_1 and W_2, i.e. the number of hidden units in the first and second layer in the example above, can be chosen arbitrarily. This is something we will return to in Section 4.9.

The following sections will try to summarize the advancements made in recent years and the methods proposed earlier to give a complete picture of the tools and features of deep neural networks.

4.2 Training

Optimizing the neural network for the data is done by maximizing the log likelihood or find the posterior distribution as in Section 2.1. However the high dimensional parameter space limits the tractable solutions to either maximum likelihood or maximum a posteriori. We can represent both of these objectives as losses by just flipping the sign making it a minimization problem instead,

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\mathcal{D}, \theta) \tag{4.3}$$

where $\mathcal{L}$ corresponds to the losses in Equation (2.1) or Equation (2.5). The high dimensional parameter space also limits the possible solvers. One way of handling this is gradient decent based solvers. Gradient decent works by updating the parameters iteratively trying to minimize the loss function.

The parameters are updated by taking small steps in the negative direction of the gradient of the loss,

$$\theta_{i+1} = \theta_i - \eta \frac{\partial}{\partial \theta} \mathcal{L}(\mathcal{D}, \theta) \tag{4.4}$$

where η is the step size parameter for the gradient decent algorithm.

However, as neural networks also commonly are used for large data sets, ordinary gradient decent can also be too computationally heavy. It is much more common to randomly partition the full data set into M smaller chunks, called *mini-batches*, here denoted $\widetilde{\mathcal{D}}^{(i)}$ for $1 \leq i \leq M$. The gradient is then updated with one of the mini-batches at a time,

$$\theta_{i+1} = \theta_i - \eta \frac{\partial}{\partial \theta} \mathcal{L}(\widetilde{\mathcal{D}}^{(i)}, \theta) \tag{4.5}$$

After M iterations, called an *epoch*, the whole data set is again split into new random mini-batches. This method, called Stochastic Gradient Decent (SGD) [49] is by far the most common way of training deep neural networks. The stochastic nature of this solver also helps on avoiding local minima, that otherwise might pose a problem for these high dimensional problems. Several different alterations to SGD exist that takes the gradient and preform some smoothing of it with help of momentum that further can improve the performance, for example ADAM [33] and RMSprop [27].

It is worth mentioning that for the training of a deep neural network to be efficient or even successful, the initialization of the parameters and the standardization (or normalization) of data is very important[20]. This is one reason why previous attempts of applying deep neural networks failed and this insight was one of the enabling factors for the initial boom of deep neural networks in late 00's [19].

4.3 Convolutional layer

Similar to the fully connected layer, a convolutional layer is an affine transformation. A convolutional layer, as the name suggests, builds on the convolution operation to transform a multichannel image or image-like object x (with dimensions: width $\times$ height $\times$ # image channels) with a kernel or filter f (with dimensions: # new channels $\times$ kernel width (w_f) $\times$ kernel height (h_f) $\times$ # image channels) into a new image-like object of features h (with dimensions, width $\times$ height $\times$ # new channels) as

$$h_{i,j,m} = \sum_{l=-\lfloor w_f/2 \rfloor}^{-\lfloor w_f/2 \rfloor + w_f} \; \sum_{k=-\lfloor w_h/2 \rfloor}^{-\lfloor h_f/2 \rfloor + h_f} \sum_{c} f_{m,l,k,c} x_{i+l,j+k,c} \tag{4.6}$$

where $\lfloor \cdot \rfloor$ is a shorthand notation for the floor function. Further, $f_{m,l,k,c}$ is an element of the kernel f, $x_{i,j,c}$ is an element of x and $h_{i,j,m}$ is an element of h. This equation is depicted in Figure 4.1. Each pixel in the feature image, h, is thus a function of a small number of pixels in the input image, these input pixels are usually called the *receptive field* for that feature pixel. The convolutional layer might also include a elementwise scalar activation function similar to the fully connected layer. In contrast to the fully connected layer, a convolutional layer has much fewer parameters, since an output feature will only depend on a subset of the input and the weight for the different features are shared.

Depending on the width of the kernel, the index accesses in Equation (4.6) might request an element of the image that does not exist, i.e. has negative index or larger than the image width/height. To alleviate this problem there are two major principles. Either one can limit the output of the convolution to only compute the features with valid input. This will then produce an feature image with smaller dimensions than the input image (height - kernel height +1 $\times$ width - kernel width +1 $\times$ # new features). This is known as only taking the *valid* components. Alternatively one can just extend the input image by padding it with zeros so that any illegal invalid index access will return zero, known as *zero padding*. Figure 4.1 depicts an example of the convolution operation for a 3×3 kernel on a 6×6 image with zero padding.

4.4 Pooling layer

The pooling operation is often used in conjunction with a convolution, to reduce the dimensionality of the feature image, i.e. reduce the resolution. A lower dimensional representation is useful to avoid overfitting of the model. Similar to the convolution, the pooling operation is a convolution over the input image but the kernel is replaced with an function with a limited receptive field. Instead this function is a channelwise simple arithmetic function, such as, the max or the average value of its input.

The function is convolved with the input image with strides. The strides corresponds to that two neighboring output feature pixels have receptive fields that are shifted by more than the usual one step. If striding equals to two for example, the receptive field shifts by two when comparing to neighboring output feature pixels. For the pooling operation, the strides are often equal to the dimensions of the receptive field of the kernel. Figure 4.2 depicts an example of a pooling operation on a 6×6 image to reduce its dimension to 3×3 with a 2×2 pooling operation (strides are also 2).

It is also possible to incorporate the strides directly into the convolutional layer. However, utilizing a pooling layer can be seen as a way to force the

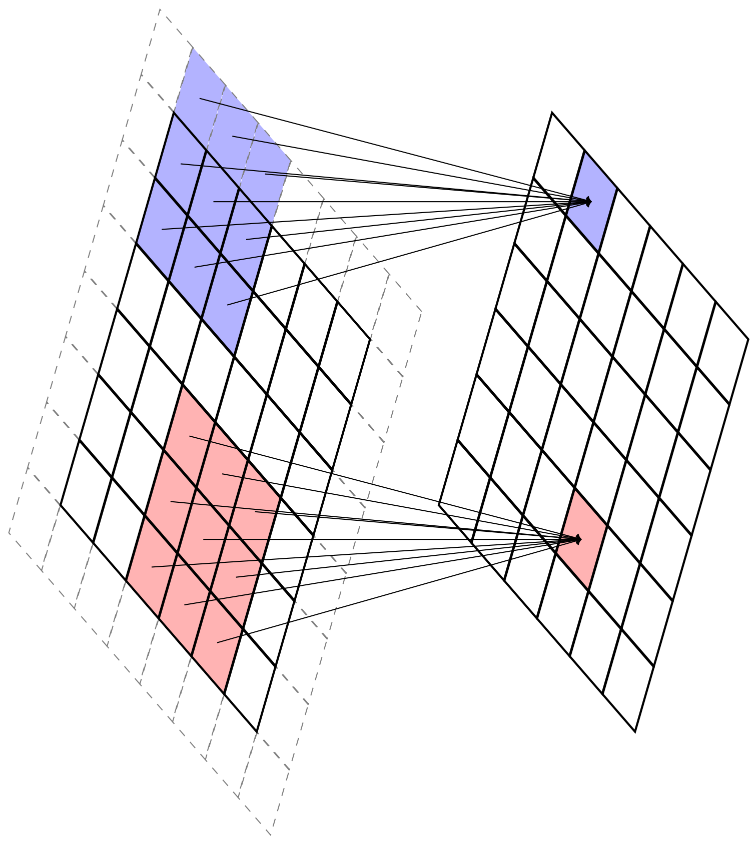


Figure 4.1: Schematic illustration of a convolution operation on a 6×6 pixels image. The left image is input and the right is the output where each pixel consists of an array of channels. The red pixel on the right is calculated as an affine transformation of the red pixels on the left, similar to a fully connected layer. The number of marked pixels on the left depends on the kernel size which in this case is 3×3. Each pixel on the right is computed in the same way and uses the same parameters for the affine transformation. To avoid accessing indices out of bounds, e.g. some of the blue pixels, the input image needs to be zero padded (the dashed squares) to 8×8 pixels.

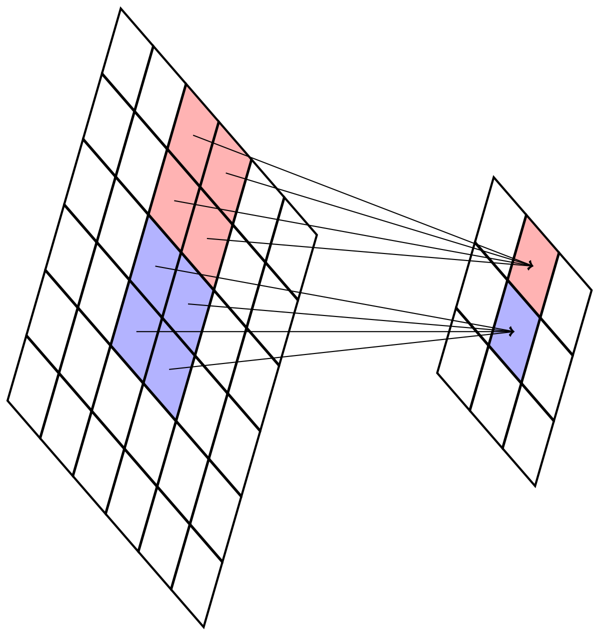


Figure 4.2: Pooling of a 6×6 pixels image with a 2×2 pooling operation. The stride is also 2 and hence the resulting output feature image is thus 3×3 pixels. A shift of one step in the output image corresponds to a shift of two steps in the input.

network to be invariant to small translations, which is the main motivation behind the structure of a pooling layer [20]. Stacking multiple convolutional layers with a pooling layers (and non linear activation functions) in between is the backbone of a convolutional neural network. The structure ensures that the output feature pixels depends on the whole image even though we only use small kernels and thus a relative few amount of parameters. The size of the receptive field also grows exponentially due to the pooling.

The reader should note that the name convolutional layer has different meanings depending on the literature. In some literature it corresponds only to the affine transformation in other to the affine transformation, a nonlinear activation function and the pooling layer altogether.

4.5 Diluted convolutional layer

In some problems we need to do predictions for each input pixel, for example, in segmentation problems. As mentioned, pooling layers reduce the dimension of the output features compared to the input features. This is not a desired feature for the one output per input kind of problems as in that case we also need to include an upscaling network. Instead we use diluted convolutions, sometimes called convolutions with holes. They are, just like ordinary convolutions, a linear operation, but the kernel is much larger and sparse. A feature pixel of a diluted convolution layer depends on input pixels that are evenly spaced and centered around the feature pixel's coordinate. The space between the dependent pixels is proportional to the dilation rate, d_w and d_h for the dilation in width and height respectively. We can write the operation as,

$$h_{i,j,m} = \sum_{l=-\lfloor w_f/2 \rfloor}^{-\lfloor w_f/2 \rfloor + w_f} \sum_{k=-\lfloor w_h/2 \rfloor}^{-\lfloor h_f/2 \rfloor + h_f} \sum_c f_{m,k,c} x_{i+d_w k, j+d_h l, c}. \tag{4.7}$$

For many applications the dilation rates for the different dimensions are equal, i.e. $d_h = d_w = d$. Note that the only difference compared to (4.6) is the multiplication of the dilatation rates, d_w and d_h on the index. Thus a dilation rate of $d = 1$ is identical to ordinary convolution. A dilation rate of d roughly corresponds to making the kernel d times larger inserting zeros to enlarge it. Figure 4.3 depicts a diluted convolution for the same problem as in Figure 4.1 but with a dilution rate of 2. Note also that the input is zero padded so that the output has the same dimensions as the input.

The main advantage of this type of network is that by stacking several diluted convolutional layers and exponentially increasing the dilation rate we may achieve both an exponentially increasing receptive field for each output feature and still have one prediction for each input pixel.

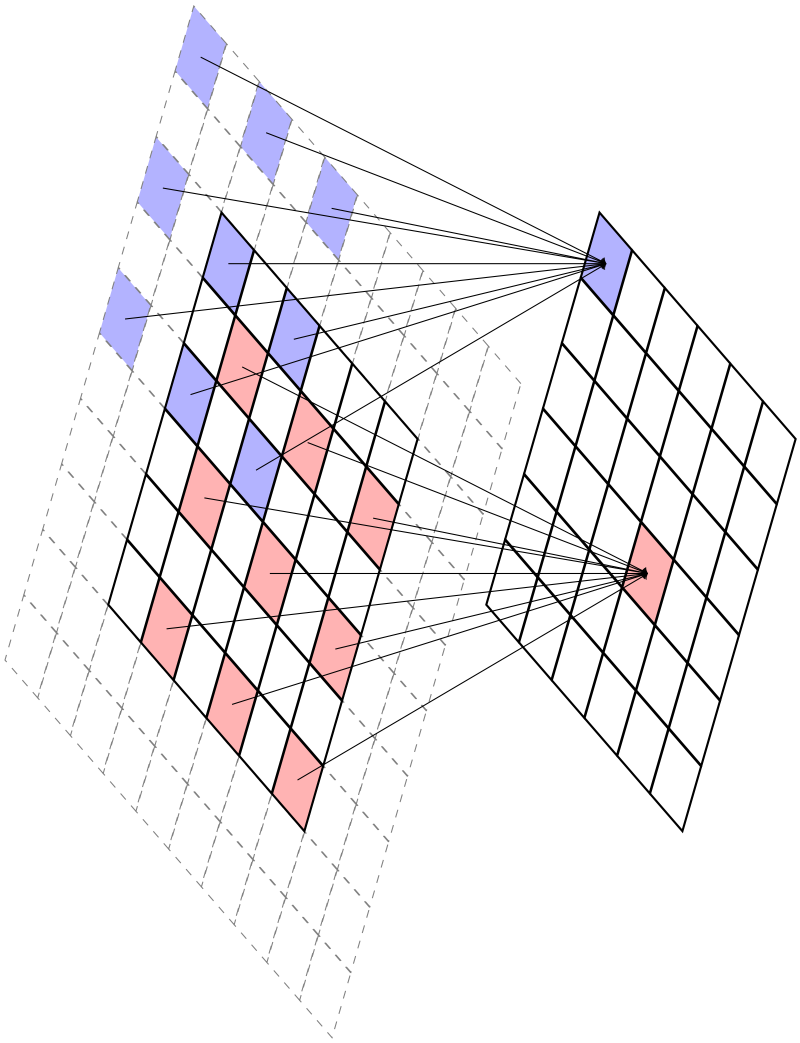


Figure 4.3: A similar set up as in Figure 4.1 but with dilation rate set to 2. The input only consist of the center 6×6 pixel. The dashed squares corresponds to the zero padding.

4.6 Regularization

To avoid overfitting a model to training data several different techniques exist to regularize the deep neural network model that also exists in other areas of machine learning, such as ridge regression and LASSO [9]. However, over the years, a couple of regularization techniques that are specific for deep neural networks and stochastic gradient decent have been proposed, out of which a few will be presented here.

Data augmentation [20] is a method to artificially increase the size of the training data. This is done by utilizing known invariances that are present in the data. For image classification it could be to slightly shift, scale and rotate the image slightly. For speech classification it could be to slightly change the amplitude or the frequency of the sequence. The augmented data is used to increase the size of the training data and thus reduce overfitting.

Early stopping is a technique specific for gradient decent algorithms which, as the name suggests, stops before the gradient decent algorithm has fully converged. Instead it stops when the validation error reaches a minimum as this avoids overfitting to the training data points. This effectively acts as a regularization[52].

Dropout is a technique unique to neural networks originally proposed by Srivastava et al. [55]. The idea is to add noise to each layer in the network by forcing a random subset of the hidden units to be zero. For each optimization iteration a new random subset is chosen. This enforces the network to learn useful features redundantly, since if a feature is only encoded by a single hidden unit it will eventually be set to zero in some iterations leading to subpar classification in that iteration. Encoding a feature redundantly decreases this probability which gives a more robust feature and in the end, less overfitting and a better generalization performance. During evaluation of the model the dropout rate is set to zero and the full network is used.

4.7 Going deeper

Deeper networks have been a catchphrase in deep learning, but training deeper neural networks has its complications. These difficulties are usually caused by vanishing and exploding gradients[7]. I.e. the gradient for the optimization algorithm either is very small, so that the optimization iterations becomes irrelevant even though the algorithm has not converged, or that the gradient is so large that a single optimization iteration destroys the progress made so far. To remedy this a number of different strategies have been proposed out of a few will be presented in the following subsections below.

4.7.1 Batch normalization

Batch normalization[30] is one strategy that is designed to circumvent vanishing and/or exploding gradients. Batch normalization works in conjunction with the ordinary layers of the neural network that normalize the activation of the hidden units just before applying the nonlinear function. Consider the fully connected layer $h = f(Wx + b)$. Batch normalization works by whitening Wx (denoted $\hat{z}$) with batch statistics, i.e.

$$\hat{z}_j^{(i)} = \frac{(Wx^{(i)})_j - \text{mean}(Wx)_j}{\text{std}(Wx)_j} \tag{4.8}$$

where the mean and standard deviation is calculated as the sample mean and variance of Wx for a batch. The index j here is an index for the hidden unit, while i is the index for an example in the batch. This whitening makes sure that the activation is normalized and by that also makes sure that the gradient does not explode or vanish. Since this operation will remove all scale information an additional scale parameter, γ, for each hidden unit is introduced. The new batch normalized fully connected layer thus looks like,

$$h = f(\gamma \odot \hat{z} + b) \tag{4.9}$$

where $\odot$ is elementwise multiplication. Note that batch normalization works just as well in a convolutional layer. Here we instead of only calculating the statistics over the batch we calculate it over the batch, width and height. Several alternatives have, since the introduction of batch normalization, been proposed, weight normalization [50] and layer normalization [5] to name a few.

4.7.2 Residual network

Concurrently with batch normalization, another design to achieve deeper neural networks was proposed. The idea is simple – avoid vanishing gradients by creating "shortcuts" for the gradient, called *residual* connections. Each such shortcut is called a residual block and can be described as,

$$h = f(x) + x \tag{4.10}$$

where f is some neural network with associated parameters. The network f is said to encode the residual between x and h, hence the name residual network[25]. Note that in this formulation f must keep the dimensions and number of hidden units the same, i.e. the output of f must have the same dimensions as x. A more general formulation instead lets f be a multi-layered network and introduce a simple layer g just to fix the dimension mismatch,

$$h = f(x) + g(x). \tag{4.11}$$

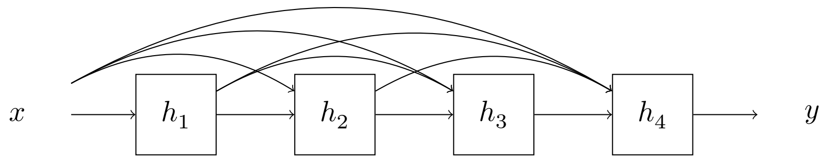


Figure 4.4: Schematic picture of the densenet structure. The input to a layer is a concatenation of the output from all the previous layers in the densenet block.

This way the gradient can still flow relatively freely through g while f can be a relatively deep network.

4.7.3 Densenet

An alternative way of creating shortcuts is Densenet [29]. Instead of adding the previous hidden units as a residual layer would, a Densenet layer concatenates all preceding hidden units and uses this as input for a layer. A dependency graph for a Densenet of depth 4 would thus look like Figure 4.4. By concatenating the previous outputs the gradient can skip large parts of the network and thus avoid vanishing or exploding.

Densenet can (as well as residual networks) be applied to either convolutional network or fully connected networks. When the densenet structure is applied to convolutional networks the concatenation is done in the channel dimension. Therefore, all convolutional layers in a Densenet block needs to have the same dimensions (except number of channels).

4.8 Unsupervised learning with deep learning

Chapter 1 gave an introduction to unsupervised learning. A common end goal with unsupervised learning is to learn some kind of representation of the data, also known as *representation learning*. In this introduction we will mostly have a probabilistic formulation, but we start with considering a general problem formulation.

Consider n data points, $\{x_i\}_{i=1}^n$ where $x_i \in \Re^m$ are generated from an unknown distribution. The goal is to find an encoder, $f : \Re^m \to \Re^p$, that maps this data into a lower dimensional space $z \in \Re^p$ (i.e. $p \ll m$), and a decoder $g : \Re^p \to \Re^m$ that maps the low dimensional representation back to the original space again to a reconstruction, $\tilde{x}$. The encoder and the decoder should minimize the a loss between x and this reconstruction, i.e.

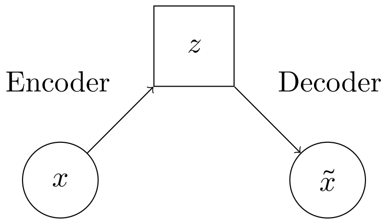


Figure 4.5: An illustration of the autoencoder.

the reconstruction loss,

$$\sum_i L(x_i, g(f(x_i))) \tag{4.12}$$

where L is an arbitrary loss function. The output of f is said to be a code for the data, i.e. it is a more compressed, although perhaps lossy, representation of the data. Figure 4.5 illustrates the overall structure of the autoencoder.

The reconstruction loss can, if the decoder and encoder are chosen freely, be arbitrarily small and all information is kept in the latent space. By limiting the capacity of the decoder and the encoder we can force the model to only keep the most relevant information. However, if we limit the decoder and encoder too much we will end up with a model which is not able to achieve the desired quality of the reconstruction. Especially, if f and g are taken as linear functions and L is the squared loss, one can show that the problem setup is equivalent to Principal Component Analysis (PCA) [20].

An Autoencoder (AE) (e.g. [11]) is a model for unsupervised learning where the encoder and the decoder are parameterized with neural networks (e.g. a convolutional network) instead of linear functions. The nonlinear functions makes it possible to use an autoencoder to model more complex distributions and possibly multimodal distributions, which is impossible to do with linear functions. However, there is no mechanism, that enforces the representation to be a compressed version of the data except for the number of parameters. The AE must thus be regularized by varying the network size and search for an useful model. The low dimensional representation can after training be used for classification or visualization, similarly to how PCA is used.

4.8.1 Log-likelihood autoencoders

Similarly to the connection between the least square and the probabilistic models in Section 2.2, one can formulate a probabilistic version of the autoencoder. This autoencoder is centered around the unsupervised learning problem with the log-likelihood objective,

$$\log p(x) = \log \int p(x \mid z) p(z) dz \tag{4.13}$$

where z is a random latent variable with a prior distribution $p(z)$. Relating back to the non-probabilistic autoencoder, $p(x \mid z)$ corresponds to the decoder. So far though there is no correspondence to the encoder.

Calculating the integral in Equation (4.13) is in many cases intractable, especially if we parameterize $p(x \mid z)$ with a neural network. The integral could be approximated using e.g. Monte Carlo estimation, but may require many samples for the integral to converge. To alleviate this problem, we introduce an alternate objective, that has lower variance than the naive Monte Carlo estimation and is a lower bound to the true objective. By maximizing this lower bound the hope is that the true objective is also maximized. To express this bound we introduce an approximate posterior distribution, $q(z \mid x)$. As long as the approximate posterior has support everywhere where $p(z)$ has support we can rewrite the initial objective as,

$$\log \int p(x \mid z) p(z) dz = \log \int p(x \mid z) p(z) \frac{q(z \mid x)}{q(z \mid x)} dz = \log \mathop{\mathbb{E}}_{q(z \mid x)} p(x \mid z) \frac{p(z)}{q(z \mid x)}. \tag{4.14}$$

Estimating this expectation with N Monte Carlo samples, given that the approximate posterior is good, has lower variance than without the approximate posterior [45] and is referred to as the *Importance Weighted* (IW) estimation or *Importance Sampling* (IS). We write this as

$$\log \mathop{\mathbb{E}}_{q(z \mid x)} p(x \mid z) \frac{p(z)}{q(z \mid x)} \approx \log \frac{1}{N} \sum_{i}^{N} p(x \mid z_i) \frac{p(z_i)}{q(z_i \mid x)} = \mathcal{L}_{\mathrm{IW}}^{N}(x) \tag{4.15}$$

where $z_i \sim q(z \mid x)$. On the other hand by utilizing Jensen's inequality we can get another with possibly less variance (if we can calculate the KL divergence analytically),

$$\log \mathop{\mathbb{E}}_{q(z \mid x)} p(x \mid z) \frac{p(z)}{q(z \mid x)} \geq \mathop{\mathbb{E}}_{q(z \mid x)} \log p(x \mid z) - \mathrm{KL}(q(z \mid x) || p(z)) = \mathcal{L}_{\mathrm{ELBO}}(x) \tag{4.16}$$

where KL is the Kullback–Leibler divergence. This lower bound to the true objective is called the Evidence Lower Bound (ELBO) or variational lower bound. The IW objective, the ELBO objective and the true objective are also related to each other as,

$$\log p(x) \geq \mathbb{E}\, \mathcal{L}_{\mathrm{IW}}^{N+1}(x) \geq \mathbb{E}\, \mathcal{L}_{\mathrm{IW}}^{N}(x) \geq \mathbb{E}\, \mathcal{L}_{\mathrm{IW}}^{1}(x) = \mathcal{L}_{\mathrm{ELBO}}(x) \tag{4.17}$$

where the expectations are over the random samples from the approximate posterior, z_i. In other words, this means that the importance weighted objectives give a better bound than ELBO but only in expectation. A proof of this relation is given in Appendix A.1.

We thus have three distributions to specify, an observation distribution $p(x \mid z)$, the latent prior distribution $p(z)$ and the approximate posterior $q(z \mid x)$. This can also be interpreted in the framework of the autoencoder with the observation distribution as the decoder and the approximate posterior as the encoder. If we parameterize the observation distribution and the approximate posterior as neural networks we get either a Variational Autoencoder (VAE) [34, 47] or an Importance Weighted Autoencoder (IWAE) [14] depending on the objective used. As an umbrella term for these autoencoders and following autoencoders that maximize the log likelihood under different approximations I propose the term *log-likelihood autoencoders* (LLAE).

One should note that even though the approximate posterior distribution can be interpreted as a variational distribution, it is strictly speaking not a variational distribution, but an amortized variational distribution. The parameters of an amortized variational distribution are not optimized to fit a specific data point but a compromise for the whole data set. A proper variational distribution would instead find the optimal parameter value for each specific datapoint. This variational distribution should rather be viewed as a function of the datapoint (see [62] for further explanation).

One of the most common parameterizations of an LLAE is to let the prior over the latent variable be a multivariate Gaussian distribution with zero mean and the identity matrix as covariance matrix,

$$p(z) = \mathcal{N}(z \mid 0, I). \tag{4.18}$$

The approximate posterior is assumed to be a Gaussian distribution with diagonal covariance matrix where the mean, $\mu_q(x, \theta)$, and covariance, $\operatorname{diag}(\sigma_q(x, \theta))$, is parameterized as neural networks,

$$q(z \mid x) = \mathcal{N}(z \mid \mu_q(x, \theta), \operatorname{diag}(\sigma_q(x, \theta))). \tag{4.19}$$

The observation distribution can vary depending on the data (see Section 2.2) but this could, for example, also be a Gausssian distribution,

$$p(x \mid z) = \mathcal{N}(z \mid \mu_p(z, \theta), \operatorname{diag}(\sigma_p(z, \theta))). \tag{4.20}$$

Examining the ELBO objective from Equation (4.16) we see that it is divided into two parts. The first part is related to the reconstruction cost (cf. Equation (4.12)) while the second part, the KL divergence term, can be interpreted as a regularization term. This fact and the probabilistic interpretability of the LLAE are perhaps the two most common selling points for the LLAE.

4.8.2 Hierarchical autoencoder

An apparent extension to the model used for the log-likelihood autoencoders is to use a hierarchical latent space to mirror the use of deep neural networks. The generative distribution for such a hierarchical model can be described as

$$p(x) = \int p(x \mid z^{(1)}, z^{(2)}, z^{(3)}) p(z^{(1)} \mid z^{(2)}, z^{(3)}) p(z^{(2)} \mid z^{(3)}) p(z^{(3)}) dz^{(1)} dz^{(2)} dz^{(3)} \tag{4.21}$$

here with 3 latent levels. However, naively introducing an approximate posterior, as for the VAE and IWAE, will only partially increase the modeling capability of the network [14]. The reason the mediocre improvement is related to the fact that the generative distribution can drift away from the approximate posterior. I.e the since $p(z^{(2)}) = \int p(z^{(2)} \mid z^{(3)}) p(z^{(3)}) dz^{(3)}$ is approximated by sampling this distribution can end up not matching the proposed approximate posterior $q(z^2)$ depending on the sample. The approximate posterior is thus not updated with a useful gradient. By tying the approximate posterior to the generative distribution this problem can partially be alleviated and training significantly deeper VAE is made possible. This tying of the two distributions is realized by sharing some parameters in the neural networks. Tying the approximate posterior to the generative distribution for VAEs can partly be contributed to Sønderby et al. [54] although it has successfully been employed for other models too (e.g. Inverse Autoregressive flow [35]). The connection between the two distributions is perhaps best explained in Figure 4.6.

Another key feature to efficiently train these hierarchical models is to make sure the KL units are used to convey information. A problem with all VAEs is that a KL term close to zero is partially stable with very small derivatives (similar to vanishing gradient). To enforce that all the KL terms are used efficiently a number of tricks have been proposed. This thesis will cover two such tricks.

KL annealing [12, 54] is an idea where one multiplies the KL term of the ELBO with a discount factor $0 \leq \gamma \leq 1$. This factor is initialized to a small number and then progressively annealed linearly towards 1 during the initial phase of training. During this phase the gradient affecting the KL divergence term is significantly larger which leads to nonzero KL divergence term.

Free bits [35] is an alternative strategy to force more information into the latent state by essentially setting the discount factor to zero if the KL term is below a certain threshold. In other words the KL term is only reduced to a certain level but not further. The amount of free bits, i.e. the value of the threshold, can also be annealed during training when the posterior is more stable. Free bits also has the advantage that it does not grow indefinitely even without annealing.

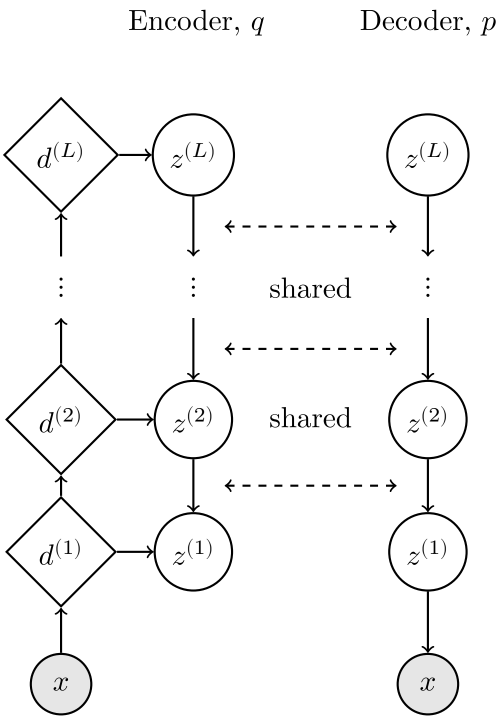


Figure 4.6: Inference model and generative model for ladder variational autoencoder. $d^{(1)}, \ldots, d^{(L)}$ are deterministic nodes.

4.8.3 Reinforce and the reparameterization trick

Both the importance weighted autoencoder and the variational autoencoder have an expectation over the approximate posterior which is approximated with Monte Carlo as,

$$\mathop{\mathbb{E}}_{q(z;\theta)} f(z) = \sum_i f(z_i) \quad \text{where} \quad z_i \sim q(z;\theta) \tag{4.22}$$

where $q_\theta(z)$ also depends on the optimization parameters, θ and $f(z)$ depends on the particular problem. The optimization (stochastic gradient decent) also requires the gradient of this expectation with respect to the parameters of the approximate posterior, but it is impossible to calculate the derivative of random samples. Instead it is possible to rewrite this expectation in two different ways, called REINFORCE [60] and the reparameterization trick [34] for which the gradient can be estimated.

The idea behind REINFORCE is to rewrite the expectation as,

$$\frac{\partial}{\partial\theta} \mathop{\mathbb{E}}_{q_\theta(z)} f(z)\bigg|_{\theta=\theta'} = \mathop{\mathbb{E}}_{q_{\theta'}(z)} f(z) \frac{\partial}{\partial\theta} \frac{q_\theta(z)}{q_{\theta'}(z)} \tag{4.23}$$

where θ' is the point at which the derivative is evaluated. However, this can be further simplified by noting that

$$\frac{\partial}{\partial\theta} \frac{q_\theta(z)}{q_{\theta'}(z)}\bigg|_{\theta=\theta'} = \frac{\partial}{\partial\theta} \log q_\theta(z)\bigg|_{\theta=\theta'} \tag{4.24}$$

and thus

$$\frac{\partial}{\partial\theta} \mathop{\mathbb{E}}_{q(z)} f(z)\bigg|_{\theta=\theta'} = \mathop{\mathbb{E}}_{q_{\theta'}(z)} f(z) \, \frac{\partial}{\partial\theta} \log q_\theta(z)\bigg|_{\theta=\theta'} \tag{4.25}$$

The other option is the reparameterization trick. It is not applicable to all distributions, only those where samples from the distribution can be rewritten as an arithmetic function of samples from a base distribution, independent of the parameters. A big family of such distributions is the location-scale family including, among others, the Gaussian distribution. The reparameterization trick for a distribution in the location-scale family can be written as

$$\frac{\partial}{\partial\theta} \mathop{\mathbb{E}}_{q_\theta(z)} f(z) = \mathop{\mathbb{E}}_{p(\epsilon)} \frac{\partial}{\partial\theta} f(\mu_\theta + \sigma_\theta \epsilon) \tag{4.26}$$

where $p(\epsilon)$ is independent of θ. For the Gaussian family it corresponds to the unit Gaussian.

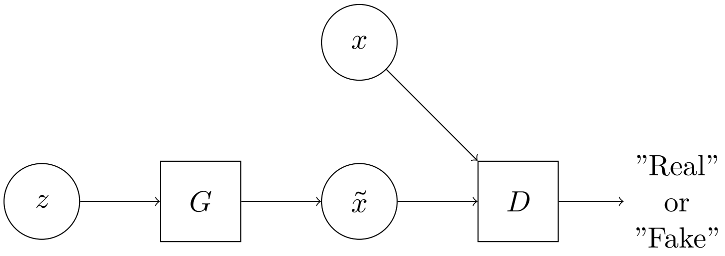


Figure 4.7: An illustration of the generative adversarial network. The network D is randomly fed either x or $\tilde{x}$ and trained to predict from which distribution the input was from.

Both the expectations arrived at with REINFORCE and reparametrization trick can then be estimated with Monte Carlo after the manipulations are done, but the estimated derivative will have different properties. Most prominent is that the gradient that the reparameterization trick produces a gradient estimate that has lower variance than that of REINFORCE. This is not universally true but for most problems in practice [18]. REINFORCE on the other hand have the advantage that it is applicable to all kinds of distributions, including discrete distributions for which the reparameterization trick can not be applied (atleast not without additional approximations).

4.8.4 Generative adversarial networks

Another very popular modeling method is the Generative Adversarial Network (GAN) which will here only be explained in relation to the log-likelihood autoencoders. As with autoencoders and variational autoencoders they were developed in two phases. First through an engineering approach [21] and then multiple followups [4, 39] which solidifies the model in a more mathematical and probabilistic framework.

Similarly to autoencoders, GANs have a decoder and an encoder, here called generator, G, and discriminator, D, respectively, both parameterized with neural networks. The difference between autoencoders and GANs is how the networks are trained. Instead of minimizing the reconstruction cost, the generator is trying to generate an example, $\tilde{x}$, that looks like being from the training data given a sample from a latent distribution, $p(z)$. The discriminator on the other hand is trying to predict whether an example presented to it is generated by the generator or taken from the training data. The goal of the generator is thus to fool the discriminator and the goal of the discriminator is to predict whether an example is fake (output should be 0) or real (output should be 1). Figure 4.7 illustrates the network. This can be viewed as a two player minimax game with the objective

$$V(D, G) = \mathop{\mathbb{E}}_{x \sim \pi(x)} [\log D(x)] + \mathop{\mathbb{E}}_{z \sim p(z)} \log(1 - D(G(z))) \tag{4.27}$$

where $\pi(x)$ is approximated using the training data. This value function is alternatingly being optimized as

$$D^{(i+1)} = \arg\max_{D} \mathop{\mathbb{E}}_{x \sim \pi(x)} [\log D(x)] + \mathop{\mathbb{E}}_{z \sim p(z)} \log(1 - D(G^{(i)}(z))) \tag{4.28}$$

and

$$G^{(i+1)} = \arg\min_{G} \mathop{\mathbb{E}}_{z \sim p(z)} \log(1 - D^{(i)}(G(z))). \tag{4.29}$$

In practice though neither of these optimizations are continued until convergence. Instead both the discriminator and the generator are trained in alternating succession. The optimization starts training one of the networks, for example the generator until it is good enough to fool its counterpart sufficiently well. At this point the optimization starts optimizing the decoder until it is not fooled anymore. This minimax optimization continues like this until the process converges.

Generative adversarial networks have received a lot of attention since their introduction by Goodfellow et al. [21]. When trained on image data, the samples from the generative distribution are sharp with many details compared to models based on LLAE. However, training the models can be very cumbersome and verifying that it actually converges can be very hard. It is also complicated to compare different generative adversarial networks as the end results are generators (generating samples) which often are limited to subjective comparison. Log-likelihood autoencoders do not have this limitation as it is easy to compare the estimated log-likelihood, but have on the other hand problems generating sharp samples when trained on images. However, more recent research on log-likelihood autoencoders, especially models expanding on the hierarchical autoencoders [35, 41], have reduced the gap regarding the sample quality.

4.9 How to choose hyperparameters?

A very common question in deep learning with neural network is how to choose the hyperparameters as the number of such parameters can grow quite big. For example, how many layers to use, how many hidden units in each layer, which constants to use for regularization, how big batches should one choose and which learning rate. There is, however, no easy answer to this question except using model validation and hyperparameter search. To some degree choosing hyperparameters is also a craftsmanship and which can be seen as implicit priors of the user. See [20] as a start for more information on the subject.

Chapter 5

Deep sequence modeling

This chapter aims at combining the sequential models from Chapter 3 with the deep learning models from Chapter 4. The models presented here are thus examples of nonlinear dynamical models.

5.1 Temporal convolutional network

As mentioned in Section 3.2, autoregressive models with exogenous inputs are used to approximate the joint distribution for a series of observations as,

$$p(x_{1:T} \mid u_{1:T}) = \prod_t p(x_t \mid x_{1:t-1}, u_{1:t}) \approx \prod_t p(x_t \mid x_{t-k:t-1}, u_{t-k+1:t}) \qquad (5.1)$$

where k denotes how many timesteps are used in the approximation. A way to parameterize this, that have proved to be efficient, is with the temporal convolutional network (TCN)[37], which, as the name suggests, use a neural network heavily inspired by convolutional neural networks (Section 4.3). As presented in the introduction convolutional neural networks are commonly used for 2-D data, like images, here on the other hand they are used on a multi-channel 1-D signal.

The distribution we want to model is thus $p(x_t \mid x_{t-k:t-1}, u_{t-k+1:t})$. For example consider parameterizing the mean of this distribution with a convolutional neural network. If this is done independently for all time steps it becomes computationally very heavy. Fortunately, as with standard CNN:s it is possible to run these computations in parallel and also reuse some of the computations when training such a model by using a diluted convolutional neural network. This network takes $x_{1:T}$ and $u_{1:T}$ as input and produce a prediction of the mean for all $x_{1:T}$ at the same time. However, when using this network we have to make sure that we still model $p(x_t \mid x_{t-k:t-1}, u_{t-k+1:t})$ by being careful that x_t does not depend on any future data or itself in the

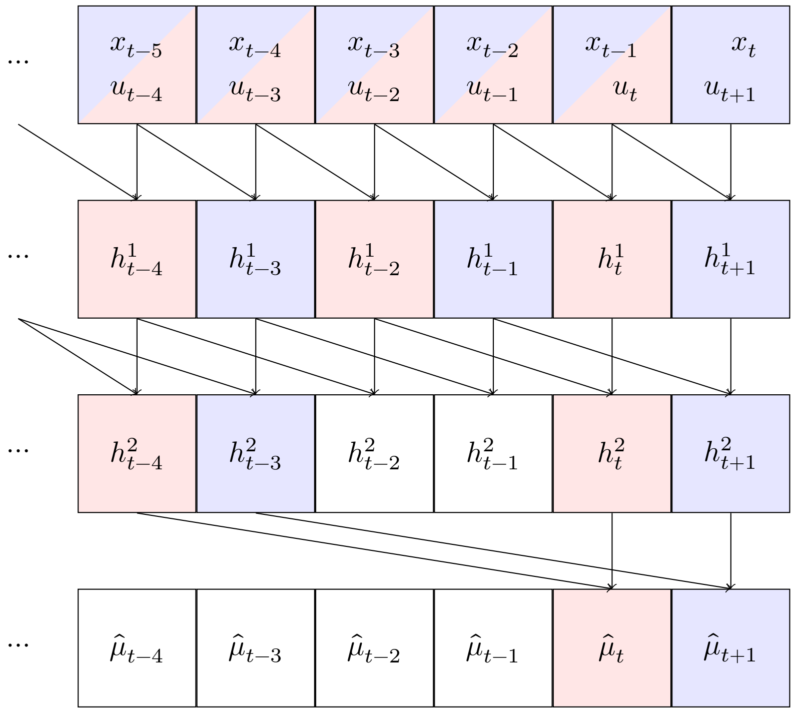

...
x_{t-5} u_{t-4}
x_{t-4} u_{t-3}
x_{t-3} u_{t-2}
x_{t-2} u_{t-1}
x_{t-1} u_{t}
x_{t} u_{t+1}
h^1_{t-4} h^1_{t-3} h^1_{t-2} h^1_{t-1} h^1_{t} h^1_{t+1}
h^2_{t-4} h^2_{t-3} h^2_{t-2} h^2_{t-1} h^2_{t} h^2_{t+1}
$\hat{\mu}_{t-4}$ $\hat{\mu}_{t-3}$ $\hat{\mu}_{t-2}$ $\hat{\mu}_{t-1}$ $\hat{\mu}_{t}$ $\hat{\mu}_{t+1}$

Figure 5.1: A Temporal Convolutional Network (TCN) with 3 layers. The layers have dilation rate 1, 2 and 4 and a kernel size of 2. Note that none of the h^*_t nor $\hat{\mu}_t$ depend on any inputs greater than or equal to t. The elements colored red and blue corresponds to the receptive field for $\hat{\mu}_t$ and $\hat{\mu}_{t+1}$ respectively.

network. The TCN does this by shifting the input, $x_{1:T}$, to the network by one time step (removing the last element and inserting a zero in the beginning) and zero pad all diluted convolutions only from the left. An example of a 3-layered TCN with dilations rates 1, 2 and 4 is depicted in Figure 5.1 where $\hat{\mu}_t$ is the predicted mean for x_t. One can of course also consider the model where only $u_1 : T$ is used as input. This would the correspond to a nonlinear finite impulse response model.

Th most prominent application of the TCN is probably Wavenet [58] which, through modeling each sample in a raw audio signal independently achieves state of the art speech quality. Since then, TCN, have been applied and compared to recurrent models with great promise[6].

5.2 Recurrent neural network

An alternative path to rewrite the factorized joint distribution is to summarize $x_{1:t-1}$ and $u_{1:t}$ into a statistic, s_t, as,

$$p(x_{1:T} \mid u_{1:T}) = \prod_t p(x_t \mid x_{1:t-1}, u_{1:t}) \approx \prod_t p(x_t \mid s_t) \tag{5.2}$$

where the statistic is updated recursively like $s_t = g(s_{t-1}, x_{t-1}, u_t)$. Note that this does not have to be an approximation, for example, the Kalman filter[32] for a linear Gaussian state space model is of this form, where the mean and the covariance of the state is the statistic. However, for more complex models this recursive update is, in most cases, intractable and needs to be approximated. One way of approximating such a function is with neural networks, which are called Recurrent Neural Networks (RNN).

Tha vanilla RNN comes in many different, but similar, flavors, e.g. the Elman RNN[15] or the Jordan RNN[31]. Here we only cover the Elman RNN. This model does not include an exogenous input and use a recursive function g parameterized as a neural network according to,

$$s_t = \sigma(W_x x_t + W_s s_{t-1} + b_s) \tag{5.3}$$

where W_* and b_s are the parameters of the model. How we choose to model $p(x_t \mid s_{t-1})$ depends on the data, e.g. for real valued data it can be modeled as a Gaussian distribution where the mean is a neural network with the statistic as input. This way of parameterizing the recurrent function suffers, similarly to deep neural networks, from vanishing and exploding gradient. Multiple models have been proposed to handle this problem and the most famous is the Long Short Term Memory (LSTM) model[28], which proposes a gating mechanism to allow the gradient to flow better. For the LSTM, the next state is calculated as,

$$\begin{aligned}
f_t &= \sigma(W_{fu} u_t + W_{fh} h_{t-1} + b_f) \\
i_t &= \sigma(W_{iu} u_t + W_{ih} h_{t-1} + b_i) \\
o_t &= \sigma(W_{ou} u_t + W_{oh} h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{cu} u_t + W_{ch} h_{t-1} + b_c) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}$$

where the statistic, $s_t = [c_t, h_t]$. Here, h_t is the output of the model i.e. the prediction of x_t in this case and u_t is the exogenous input. W_{**} and b_* are the parameters of the model. $\odot$ denotes elementwise multiplication and we thus interpret r_t, i_t and o_t as gates and will help in training the model. The observant reader probably noted that g for the LSTM does not include any dependence on x_{t-1}. The most common strategy include this is to simply concatenate x_{t-1} to u_t.

The LSTM and the TCN corresponds to nonlinear state space models and a nonlinear autoregressive models, respectively. However, these two particular models are not well used in the typical system identification community. Paper II evaluates these models on typical benchmark problems from system identification.

5.3 Stochastic RNN

An idea by Fraccaro et al. [17] called a Stochastic RNN (SRNN) is to combine the state space model with a variational model. The generative model of a SRNN combines a state space model with an autoregressive model using statistics. The full generative model looks like,

$$p(x_{1:T}) = \prod_t \mathbb{E}\, p(x_t \,|\, z_t, d_t) p(z_t \,|\, z_{t-1}) \tag{5.4}$$

where $d_t = f(s_{t-1}, x_{t-1})$.

As we did for the variational autoencoder we can introduce the variational/proposal distribution, $q(z_{1:T} \,|\, x_{1:T})$, which we can factorize as,

$$q(z_{1:T} \,|\, x_{1:T}) = \prod_t q(z_t \,|\, z_{t-1}, d_t, x_{t:T}) \tag{5.5}$$

using the same Markovian assumption as for the model. $x_{t:T}$ can be approximated with a recursive statistic as done in Section 5.2 but running backwards in time, as

$$q(z_t \,|\, z_{t-1}, x_{t:T}) \approx q(z_t \,|\, z_{t-1}, a_t) \tag{5.6}$$

where $a_t = f(a_{t+1}, d_t, x_t)$, note that the dependence on d_t was combined into a_t. Inserting this into the factorized version of the likelihood we get,

$$p(x_{1:T}) = \prod_t \mathop{\mathbb{E}}_{q(z_t \,|\, z_{t-1}, a_t)} p(x_t \,|\, z_t) \frac{p(z_t \,|\, z_{t-1})}{q(z_t \,|\, z_{t-1}, a_t)}. \tag{5.7}$$

More specifically it has both a deterministic statistic, d_t and a latent state, z_t, as state variables and the predictive distribution is dependent on both of these. The generative model thus looks like,

$$p(x_{1:T}) = \prod_t \mathbb{E}\, p(x_t \,|\, z_t, d_t) p(z_t \,|\, z_{t-1}) \tag{5.8}$$

where $d_t = \mathrm{RNN}(x_{t-1}, d_t)$. The entire model is described in Figure 5.2. Both the deterministic statistics (d_t and a_t) are modeled with LSTM:s while the latent state is modeled as,

$$p(z_{t+1} \mid z_t, d_t) = \mathcal{N}(z_{t-1} \mid \mu = f_{\mu,p}(z_t, d_t), \sigma^2 = f_{\sigma,p}(z_t, d_t))$$
$$q(z_{t+1} \mid z_t, a_{t+1}) = \mathcal{N}(z_{t-1} \mid \mu = f_{\mu,q}(z_t, a_t), \sigma^2 = f_{\sigma,q}(z_t, a_t))$$

where the functions $f_{\mu,p}$, $f_{\sigma,p}$, $f_{\mu,q}$ and $f_{\sigma,q}$ are all modeled as 2-layered neural networks where some of the weights in $f_{\mu,*}$ and $f_{\Sigma,*}$ are shared for p and q respectively.

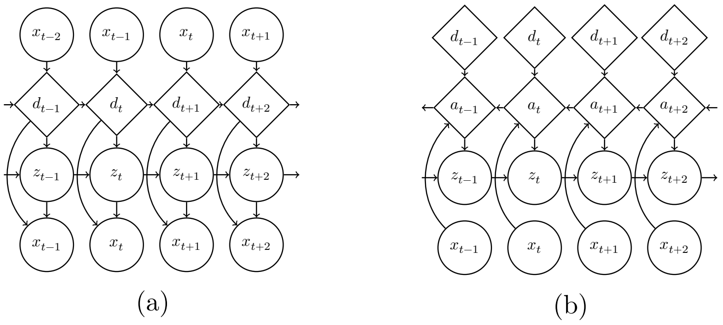


Figure 5.2: The stochastic RNN model. a The generative network p and b the inference network q.

5.4 Stochastic TCN

The Stochastic TCN (STCN) model [1] is a model that combines the depth of a hierarchical variational autoencoder with the sequential component from the TCN. The STCN network uses a TCN to create a hierarchical representation ($h_t^1, h_t^2, \ldots$ in Figure 5.1) for both the inference network and the generative network. These different levels ($h_t^1, h_t^2, \ldots$) are used as additional input to the deterministic nodes, ($d^1, d^2, \ldots$), in the inference network, c.f. Figure 4.6. Additionally the levels are also used as additional input to the respective layer in the generative network in Figure 4.6. However, the generative network only uses information up to timestep $t-1$ when doing a prediction of timestep t while the inference network uses information upto timestep t. Hence the networks uses two consecutive hierarchical representations for the prior and the approximate posterior. Figure 5.3 depicts an overview of the STCN.

The hierarchical representation enforced by the TCN is reflected in the hierarchical representation of the hierarchical variational autoencoder forcing long temporal dependencies to be encoded in the highest latent states while latent states further down only encode more local information.

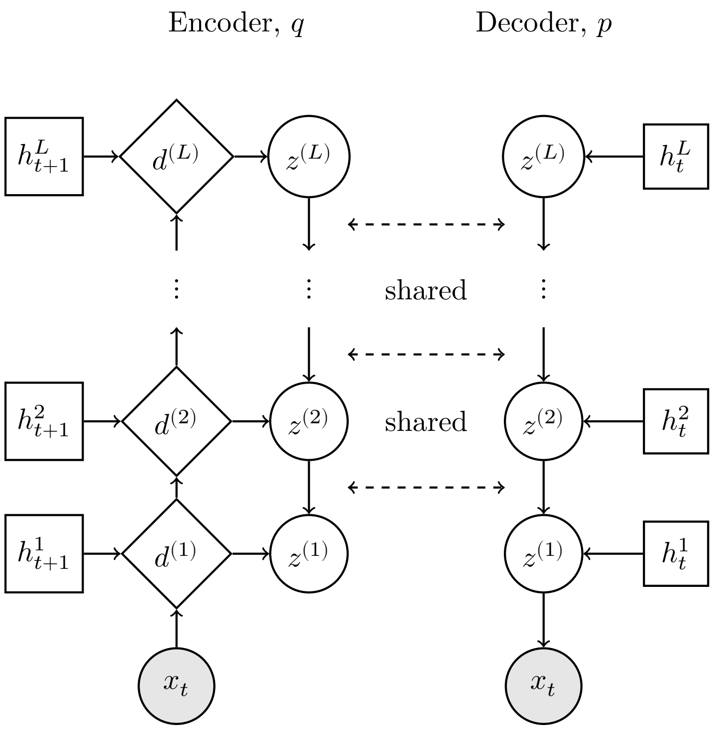


Figure 5.3: Inference model and generative model for conditioned on the output of a TCN

Chapter 6

Concluding remarks

6.1 Conclusion

The area of deep learning applied to sequence modeling is huge. This thesis only covers a small fraction of the techniques and models that are available. It is exiting times for everyone and especially when doing research, with many new submission every day. It is also a taunting task of keeping oneself updated and to navigate through the vast space of papers.

In this thesis we have covered how to apply deep learning to system identification in both traditional and novel ways. We noted that LSTM and TCN can be used as deep variants of the nonlinear state space model and the nonlinear autoregressive model, respectively. We have also seen that we can use deep learning to parameterize a regularization matrix to improve estimation of a finite impulse response model.

6.2 Future work

For me the next step will be to apply the sequence models to other types of problems. The problem that we have had in mind since the beginning of these 3 years is to combine it with reinforcement learning and/or optimal control. The immediate future however will focus on extensions on the papers mentioned in the thesis. Below follows some ideas:

- We have plans on an extension to Paper II to also consider the colored process/measurement noise and for systems where the measurement noise not is symmetric. One idea to handle this is to use the variational autoencoder to build more complex distributions.
- We also have plans to extend paper III in several different directions. First some ideas on improving the usefulness of some of the theorems

so that a more practical lower bound to the expected miscalibration can be found.

- Even though we so far have not submitted anything on this matter, a strong research area for me is to marry the hierarchical variational autoencoder with a sequence model, much like what Aksan et al. [1] have done.

Appendix A

Appendix

A.1 Relation between IWAE and VAE

Here follows a sketch for a proof of the following relation stated in Section 4.8.1,

$$\log p(x) \geq \mathbb{E}\,\mathcal{L}_{\mathrm{IW}}^{N+1}(x) \geq \mathbb{E}\,\mathcal{L}_{\mathrm{IW}}^{N}(x) \geq \mathbb{E}\,\mathcal{L}_{\mathrm{IW}}^{1}(x) = \mathcal{L}_{\mathrm{ELBO}}(x) \tag{A.1}$$

The final relation between $\mathbb{E}\,\mathcal{L}_{\mathrm{IW}}^{1}(x)$ and $\mathcal{L}_{\mathrm{ELBO}}(x)$ can be seen by just inserting the expression for the losses. The other relations are a bit more complicated. We start by rewriting $\mathcal{L}_{\mathrm{IW}}^{N}(x)$ as

$$\begin{aligned}\mathcal{L}_{\mathrm{IW}}^{N}(x) &= \log \frac{1}{N}\sum_{i}^{N} p(x \mid z_i)\frac{p(z_i)}{q(z_i \mid x)} \\ &= \log \frac{1}{N}\sum_{i=1}^{N} f(x, z_i)\end{aligned} \tag{A.2}$$

where we use $f(x, z_i)$ as a shorthand notation for the quantity $p(x \mid z_i)\frac{p(z_i)}{q(z_i \mid x)}$. We will assume that $f(x, z_i)$ is positive for all z_i. We split $f(x, z_i)$ in to its mean $\bar{f}(x)$ and the deviation from the mean $\tilde{f}(x, z_i)$ like

$$\bar{f}(x) = \mathbb{E}\, f(x, z_i) = p(x) \tag{A.3}$$

$$\tilde{f}(x, z_i) = f(x, z_i) - \mathbb{E}\, f(x, z_i) \tag{A.4}$$

Note that $\log \bar{f}(x)$ actually equals what we wanted to estimate from the beginning, $\log p(x)$. We further rewrite this as,

$$\begin{aligned}\mathcal{L}_{\text{IW}}^{N}(x) &= \log \frac{1}{N}\sum_{i=1}^{N} f(x,z_i) \\ &= \log \left[p(x) + \frac{1}{N}\sum_{i}^{N} \tilde{f}(x,z_i) \right] \\ &= \log p(x) + \log \left[1 + \frac{1}{N}\sum_{i}^{N} \frac{\tilde{f}(x,z_i)}{p(x)} \right] \end{aligned} \tag{A.5}$$

Finally we denote $\frac{\tilde{f}(x,z_i)}{p(x)}$ with ϵ_i. We can now see that ϵ_i due to the constraints on f is a zero mean i.i.d. random variable greater than -1. We assume that ϵ_i follows some regularity assumptions so that the law of large numbers apply. Thus if N is sufficiently large we have that $\nu(N) = \frac{1}{N}\sum_{i}^{N} \epsilon_i \sim \mathcal{N}(0, \frac{1}{N}\text{Var}[\epsilon_i])$. Taylor series expanding the log in Equation (A.5) we finally get,

$$\begin{aligned}\mathbb{E}\,\mathcal{L}_{\text{IW}}^{N}(x) &= \log p(x) + \mathbb{E}\left[\nu(N) - \frac{1}{2}\nu(N)^2 + \frac{1}{3}\nu(N)^3 - \frac{1}{4}\nu(N)^4 + O(\nu(N)^5) \right] \\ &= \log p(x) - \frac{1}{2N}\text{Var}(\epsilon_i) - O(1/N^2). \end{aligned} \tag{A.6}$$

Thus the first relation in Equation (A.1) does hold and this gives an intuitive reasoning for the other relations to hold also. However, we also need to show that this hold also for small N where the law of large numbers does not apply.

To show this more general statement we start form Equation (A.2), which we can rewrite as follows,

$$\begin{aligned}\mathcal{L}_{\text{IW}}^{N}(x) &= \log \frac{1}{N}\sum_{i=1}^{N} f(x,z_i) \\ &= \log \frac{1}{N}\sum_{i=1}^{N-1} \left(f(x,z_i) + f(x,z_N) \right) \\ &= \log \frac{N-1}{N}\frac{1}{N-1}\left(\sum_{i=1}^{N-1} f(x,z_i) + f(x,z_N) \right) \\ &= \log \frac{N-1}{N}\frac{1}{N-1}\sum_{i=1}^{N-1} f(x,z_i) \left(1 + \frac{f(x,z_N)}{\sum_{i=1}^{N-1} f(x,z_i)} \right) \\ &= \underbrace{\log \frac{1}{N-1}\sum_{i=1}^{N-1} f(x,z_i)}_{\mathcal{L}_{\text{IW}}^{N-1}(x)} + \log \frac{N-1}{N}\left(1 + \frac{f(x,z_N)}{\sum_{i=1}^{N-1} f(x,z_i)} \right). \end{aligned} \tag{A.7}$$

Taking the expectation of this yields

$$\begin{aligned}\mathbb{E}\,\mathcal{L}_{\mathrm{IW}}^{N}(x) &= \mathbb{E}\,\mathcal{L}_{\mathrm{IW}}^{N-1}(x) + \mathbb{E}\log\frac{N-1}{N}\left(1+\frac{f(x,z_N)}{\sum_{i=1}^{N-1} f(x,z_i)}\right)\\ &\geq \mathbb{E}\,\mathcal{L}_{\mathrm{IW}}^{N-1}(x) + \log\frac{N-1}{N}\,\mathbb{E}\left(1+\frac{f(x,z_N)}{\sum_{i=1}^{N-1} f(x,z_i)}\right)\end{aligned} \tag{A.8}$$

by Jensen's inequality. We can bound the second term from above as,

$$\begin{aligned}\frac{N-1}{N}\,\mathbb{E}\left(1+\frac{f(x,z_N)}{\sum_{i=1}^{N-1} f(x,z_i)}\right) &= \frac{N-1}{N}\left(1+\mathbb{E}\,f(x,z_N)\,\mathbb{E}\,\frac{1}{\sum_{i=1}^{N-1} f(x,z_i)}\right)\\ &\leq \frac{N-1}{N}\left(1+\mathbb{E}\,f(x,z_N)\frac{1}{\mathbb{E}\sum_{i=1}^{N-1} f(x,z_i)}\right)\\ &= \frac{N-1}{N}\left(1+\frac{1}{N-1}\right) = 1\end{aligned} \tag{A.9}$$

again using Jensen's inequality. This gives

$$\log\frac{N-1}{N}\,\mathbb{E}\left(1+\frac{f(x,z_N)}{\sum_{i=1}^{N-1} f(x,z_i)}\right) \leq 0 \tag{A.10}$$

and thus

$$\mathbb{E}\,\mathcal{L}_{\mathrm{IW}}^{N}(x) \geq \mathbb{E}\,\mathcal{L}_{\mathrm{IW}}^{N-1}(x) \tag{A.11}$$

$\square$

References

[1] E. Aksan and O. Hilliges. *STCN: Stochastic Temporal Convolutional Networks*. Tech. rep. arXiv: `1902.06568v1`.

[2] C. Andersson, A. L. Ribeiro, K. Tiels, N. Wahlström, and T. B. Schön. “Deep convolutional networks in system identification”. In: *Proceedings of the 58th IEEE Conference on Decision and Control (CDC)*. 2019.

[3] C. Andersson, N. Wahlström, and T. B. Schön. “Data-Driven Impulse Response Regularization via Deep Learning”. In: *18th IFAC Symposium on System Identification (SYSID)*. 2018.

[4] M. Arjovsky, S. Chintala, and L. Bottou. “Wasserstein Generative Adversarial Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 214–223.

[5] J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer Normalization*. Tech. rep. 2016. arXiv: `1607.06450`.

[6] S. Bai, J. Zico Kolter, and V. Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. Tech. rep. 2018. arXiv: `1803.01271v2`.

[7] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994).

[8] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. 1st. Athena Scientific, 1996.

[9] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[10] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. “Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription”. In: *International Conference on Machine Learning 29*. 2012.

[11] H. Bourlard and Y. Kamp. “Auto-association by multilayer perceptrons and singular value decomposition.” In: *Biological cybernetics* 59 (1988).

[12] S. R. Bowman, L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio. “Generating Sentences from a Continuous Space”. In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. 2016.

[13] J. Bröcker. “Some Remarks on the Reliability of Categorical Probability Forecasts”. In: *Monthly Weather Review* 136.11 (2008).

[14] Y. Burda, R. Grosse, and R. Salakhutdinov. *Importance weighted autoencoders*. Tech. rep. 2015. arXiv: `1509.00519v4`.

[15] J. L. Elman. “Finding Structure in Time”. In: *Cognitive Science* 14.2 (1990).

[16] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. “Dermatologist-level classification of skin cancer with deep neural networks”. In: *Nature* 542.7639 (2017).

[17] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. “Sequential Neural Models with Stochastic Layers”. In: *Advances in Neural Information Processing Systems 29*. 2016.

[18] Y. Gal. “Uncertainty in Deep Learning”. PhD thesis. University of Cambridge, 2016.

[19] X. Glorot and Y. Bengio. “Understanding the difficulty of training deep feedforward neural”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010.

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. 2014.

[22] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. “On Calibration of Modern Neural Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. 2017.

[23] P. E. Hart, N. J. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968).

[24] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning : data mining, inference, and prediction*. Springer, 2009.

[25] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[26] G. E. Hinton, S. Osindero, and Y.-W. Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18.7 (2006).

[27] G. Hinton, N. Srivastava, and K. Swersky. *Neural Networks for Machine Learning Lecture 6a Overview of mini---batch gradient descent*. 2012.

[28] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997). arXiv: `1206.2944`.

[29] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[30] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. 2015.

[31] M. I. Jordan. "Serial Order: A Parallel Distributed Processing Approach". In: *Advances in Psychology* 121 (1997).

[32] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* (1960).

[33] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2015.

[34] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. 2014.

[35] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. "Improved Variational Inference with Inverse Autoregressive Flow". In: *Advances in Neural Information Processing Systems 29*. 2016.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. 2012.

[37] C. Lea, R. Vidal, A. Reiter, and G. Hager. "Temporal convolutional networks: A unified approach to action segmentation". In: *Computer Vision – ECCV 2016 Workshops, Proceedings*. 2016.

[38] Y. LeCun, Y. Bengio, G. Hinton, L. Y., B. Y., and H. G. "Deep learning". In: *Nature* 521.7553 (2015). arXiv: `arXiv:1312.6184v5`.

[39] C.-L. Li, W.-C. Chang, Y. Cheng, Y. Yang, and B. Poczos. "MMD GAN: Towards Deeper Understanding of Moment Matching Network". In: *Advances in Neural Information Processing Systems 30*. 2017.

[40] L. Ljung, ed. *System Identification (2nd Ed.): Theory for the User.* Prentice Hall PTR, 1999.

[41] L. Maaløe, M. Fraccaro, V. Liévin, and O. Winther. *BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling.* Tech. rep. 2019. arXiv: `1902.02102v1`.

[42] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing.* MIT Press, 1999.

[43] T. M. Mitchell. *Machine Learning.* 1st ed. McGraw-Hill, Inc., 1997.

[44] L. R. Rabiner and B.-H. Juang. *Fundamentals of speech recognition.* PTR Prentice Hall, 1993.

[45] T. Rainforth, A. R. Kosiorek, T. A. Le, C. J. Maddison, M. Igl, F. Wood, and Y. W. Teh. “Tighter Variational Bounds are Not Necessarily Better”. In: *Proceedings of the 35th International Conference on Machine Learning.* 2018.

[46] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2005.

[47] D. J. Rezende, S. Mohamed, and D. Wierstra. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”. In: *Proceedings of the 31st International Conference on Machine Learning.* Vol. 32. Proceedings of Machine Learning Research 2. 2014.

[48] A. H. Ribeiro et al. *Automatic Diagnosis of the Short-Duration 12-Lead ECG using a Deep Neural Network: the CODE Study.* Tech. rep. 2019. arXiv: `1904.01949v1`.

[49] H. Robbins and S. Monro. “A Stochastic Approximation Method”. In: *Ann. Math. Statist.* 22.3 (1951).

[50] T. Salimans and D. P. Kingma. “Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks”. In: *Advances in Neural Information Processing Systems 29.* 2016.

[51] D. Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016).

[52] J. Sjöberg and L. Ljung. “Overtraining, regularization and searching for a minimum, with application to neural networks”. In: *International Journal of Control* 62 (1995).

[53] A. F. M. Smith. “Novel approach to nonlinear/non-Gaussian Bayesian state estimation”. In: *IEE Proceedings F (Radar and Signal Processing)* 140.2 (1993).

[54] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. “Ladder Variational Autoencoders”. In: *Advances in Neural Information Processing Systems 29*. 2016.

[55] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014). arXiv: `1102.4807`.

[56] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. 1st. MIT Press, 1998.

[57] J. Vaicenavicius, D. Widmann, C. Andersson, F. Lindsten, J. Roll, and T. B. Schön. “Evaluating model calibration in classification”. In: *Proceedings of AISTATS*. 2019.

[58] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. *WaveNet: A Generative Model for Raw Audio*. Tech. rep. arXiv: `1609.03499v2`.

[59] O. Vinyals et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`. 2019.

[60] R. J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Mach. Learn.* 8.3-4 (1992).

[61] K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015.

[62] C. Zhang, J. Bütepage, H. Kjellström, and S. Mandt. *Advances in Variational Inference*. Tech. rep. 2017. arXiv: `1711.05597v3`.

[63] Z.-H. Zhou and J. Feng. “Deep forest”. In: *National Science Review* 6.1 (2019).

Paper I

Title

Data-Driven Impulse Response Regularization via Deep Learning

Authors

Carl Andersson, Niklas Wahlström, and Thomas B. Schön

Edited version of

C. Andersson, N. Wahlström, and T. B. Schön. "Data-Driven Impulse Response Regularization via Deep Learning". In: *18th IFAC Symposium on System Identification (SYSID)*. Stockholm, Sweden, 2018

Data-Driven Impulse Response Regularization via Deep Learning

Abstract

We consider the problem of impulse response estimation of stable linear single-input single-output systems. It is a well-studied problem where flexible non-parametric models recently offered a leap in performance compared to the classical finite-dimensional model structures. Inspired by this development and the success of deep learning we propose a new flexible data-driven model. Our experiments indicate that the new model is capable of exploiting even more of the hidden patterns that are present in the input-output data as compared to the non-parametric models.

1 Introduction

Impulse response estimation has for a long time been at the core of system identification. Up until some five to seven years ago, the generally held belief in the field was indeed that we knew all there was to know about this topic. However, the enlightening work by Pillonetto et al. [15] changed this by showing that the estimate can in fact be improved significantly by placing a Gaussian Process (GP) prior on the impulse response, which acts as a regularizer. This model-driven approach has since then been further refined [4, 13, 14], where the prior in this case could be interpreted to encode not only smoothness information, but also information about the exponential decay of the impulse response. In this paper we employ deep leaning (DL) to find a suitable regularizer via a method that is driven by data. Fig. 1 depicts the general idea and the similarity of our method compared to the method based on Gaussian processes.

Deep learning is a fairly new area of research that continues the work on

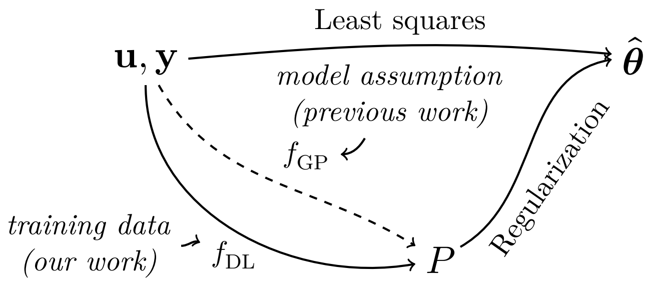


Figure 1: Schematic figure over the proposed method for impulse response estimation using deep learning in relation to the previous work using Gaussian processes. The functions f_{GP} and f_{DL} maps the input sequence $\mathbf{u}$ and the output sequence $\mathbf{y}$ of a system to an inverse regularization matrix P for the Gaussian process approach and the deep learning approach, respectively.

neural networks from the 1990's. To get a brief, but informative, overview of the field of deep learning we recommend the paper by LeCun et al. [11] and for a more complete snapshot of the field we refer to the monograph by Goodfellow et al. [8]. Deep learning has recently revolutionized several fields, including image recognition (e.g. Cirean et al. [6]) and speech recognition (e.g. Hinton et al. [9]). In both fields, deep learning has surpassed domain specific methods and hand-crafted feature design, by making use of large quantities of data in order to learn data-driven neural network models as general function approximators.

The idea of using neural networks within system identification is certainly not new and they have been a standard tool for a long time, see e.g. Sjöberg et al. [17]. However, the current development in deep learning is different from the past due to advances in computational software and hardware. As a consequence, contemporary neural networks are more reproducible than before which has increased the credibility of the area. Finally, the amount of available data has sky-rocketed which has made it possible to train larger models with better results. We believe that system identification still has lots to gain from using deep learning and this paper is just one concrete example of what can be done.

2 Background and problem formulation

Consider a stable single-input single-output time-invariant linear system G^0 relating an input sequence $u(t)$ to an output sequence $y(t)$ according to

$$y(t) = G^0(q)u(t) + v(t), \tag{I.1}$$

where q denotes the shift operator $qu(t) = u(t+1)$ and $v(t)$ denotes additive noise. The noise is assumed to be Gaussian white noise with zero mean and variance σ^2. The system $G^0(q)$ is represented using a transfer function

$$G^0(q) = \sum_{k=1}^{\infty} g_k^0 q^{-k}, \tag{I.2}$$

where $g_1^0, g_2^0, \ldots$ denote the impulse response coefficients of the system. We make use of superscript 0 to denote the true system.

Based on data, i.e. an input sequence $\mathbf{u} = (u(1), \ldots, u(N))^T$ and an output sequence $\mathbf{y} = (y(1), \ldots, y(N))^T$ both of length N, the task is to compute an estimate that represents the true system $G^0(q)$ as well as possible.

Traditionally, the transfer function is encoded using a finite number of parameters $\boldsymbol{\theta}$ ($\dim(\boldsymbol{\theta}) = n \ll N$), for example, via a finite impulse response (FIR) model,

$$G(q; \boldsymbol{\theta}) = B(q) = b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}, \tag{I.3}$$

or an output-error (OE) model,

$$G(q; \boldsymbol{\theta}) = \frac{B(q)}{F(q)} = \frac{b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}}{f_1 q^{-1} + \cdots + f_{n_f} q^{-n_f}}, \tag{I.4}$$

where $\boldsymbol{\theta} = (b_1, \ldots, b_{n_b})^{\mathrm{T}}$ or $\boldsymbol{\theta} = (b_1, \ldots, b_{n_b}, f_1, \ldots, f_{n_f})^{\mathrm{T}}$, respectively. See Ljung [12] for a comprehensive list of model structures. With these model structures, the prediction error method can be used to compute a point estimate $\hat{\boldsymbol{\theta}}$ of the unknown parameters by solving the following optimization problem,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{t=1}^{N} (y(t) - G(q, \boldsymbol{\theta}) u(t))^2 + \boldsymbol{\theta}^{\mathrm{T}} D \boldsymbol{\theta}, \tag{I.5}$$

where $\boldsymbol{\theta}^{\mathrm{T}} D \boldsymbol{\theta}$ describes the added regularization term, governed by the regularization matrix D. It has been shown by Chen et al. [4] and Pillonetto et al. [14, 15] that the use of an effective regularization is more important than the specific choice of model-order, n. Intuitively the use of a GP model and the regularization term $\boldsymbol{\theta}^{\mathrm{T}} D \boldsymbol{\theta}$ opens up for model selection at a "finer" scale compared to what is possible with the classical finitely parameterized model structures. Adding this kind of regularization is especially important when the number of samples, N, is low compared to the parameters we wish to estimate, n.

In the case of the FIR model, the optimization problem reduces to a linear least squares problem to find the optimal parameters $\hat{\boldsymbol{\theta}}$. For numerical reasons the inverse regularization matrix $P = D^{-1}$ is often used in place of D. This gives rise to the following analytic solution,

$$\hat{\boldsymbol{\theta}} = \left(PR + I_n\right)^{-1} PF_N, \tag{I.6a}$$

where

$$R = \sum_{t=n+1}^{N} \varphi(t)\varphi(t)^{\mathrm{T}}, \tag{I.6b}$$

$$\varphi(t) = \{u(t-1) \dots u(t-n)\}^{\mathrm{T}}, \tag{I.6c}$$

$$F_N = \sum_{t=n+1}^{N} \varphi(t)y(t). \tag{I.6d}$$

We will throughout this paper denote then estimate (I.6) by $\hat{\boldsymbol{\theta}}(P)$ to stress its dependence on the inverse regularization matrix P. As a special case, with $P = 0$, we have the least squares solution, which we denote $\hat{\boldsymbol{\theta}}_{\mathrm{LS}}$.

One question still remains though; how do we find the inverse regularization matrix P? One of the most general ideas is to let it depend on $\mathbf{u}$ and $\mathbf{y}$. This was done in Chen et al. [4] and Pillonetto et al. [14, 15] by modelling the impulse response as a Gaussian process. A Gaussian process is known to be a very flexible prior, even so, since the model only depends on a low number of hyper-parameters, typically one for a lengthscale and one for some noise, it heavily depends on the specific model we choose for the covariance function. These hyper-parameters are replaced by a point estimate obtained by maximizing the marginal likelihood of the observed data, a procedure known as Empirical Bayes [2]. The regularization matrix will thus implicitly depend on $\mathbf{u}$ and $\mathbf{y}$ via the hyper-parameters. We explicitly denote this dependence by $P = f_{\mathrm{GP}}(\mathbf{u}, \mathbf{y})$. This method is also explained in more detail in Section 3.

We instead propose an arguably even more flexible model by parametrizing the inverse regularization matrix P with a neural network $P = f_{\mathrm{DL}}(\mathbf{u}, \mathbf{y})$. In contrast to the Gaussian process model these parameters are computed by training the model with lots of training data consisting of an input sequence, an output sequence and the true impulse response for either real or simulated systems. Compared to the GP model this is a more data-driven approach to the problem which also makes it possible to use existing techniques from deep learning when building and training the model.

3 Regularization using Gaussian Process

The Gaussian prior offers a natural way of encoding the smoothness and decay characteristics that we find in the impulse response from a stable linear system. The specific details of these characteristics are tuned via the hyper-parameters λ. The resulting GP prior can be written as

$$\boldsymbol{\theta} \sim p_{\lambda}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \,|\, 0, P^{\lambda}), \tag{I.7}$$

where P^{λ} in this Bayesian setting is equal to the inverse regularization matrix in (I.6). The matrix P^{λ} is related to the covariance function of the Gaussian Process as $k_{\lambda}(i,j) = P^{\lambda}_{ij}$, where P^{λ}_{ij} denotes then entry on row i and column j of P^{λ}. With the aid of measured data we can select a point estimate of the hyper-parameters $\hat{\lambda}$ by maximizing the marginal log-likelihood,

$$\hat{\lambda} = \arg\max_{\lambda} \log \int p(\mathbf{y} \,|\, \boldsymbol{\theta}; \mathbf{u}) p_{\lambda}(\boldsymbol{\theta}) d\boldsymbol{\theta}, \tag{I.8}$$

where

$$p(\mathbf{y} \,|\, \boldsymbol{\theta}; \mathbf{u}) = \prod_{t=1}^{N} p(y(t) \,|\, \boldsymbol{\theta}; \mathbf{u}) = \prod_{t=1}^{N} \mathcal{N}(y(t) \,|\, \boldsymbol{\theta}^{\mathrm{T}} \varphi(t), \sigma^2). \tag{I.9}$$

As a direct consequence of this, the optimal inverse regularization matrix $P^{\hat{\lambda}}$ implicitly depends on the input sequence $\mathbf{u}$ and the output sequence $\mathbf{y}$. The equation (I.8) then describe the function $P = f_{\mathrm{GP}}(\mathbf{u}, \mathbf{y})$. Using this inverse regularization matrix together with (I.6), we obtain an estimate of the FIR parameters that has better accuracy and is more robust than the non-regularized approach [4].

4 Regularizing using Deep Learning

In contrast to previous work where the regularization matrix only depends on a few hyper-parameters we instead model the regularization matrix directly with a neural network that depends on a large number of parameters η according to

$$P = f_{\mathrm{DL}}(\mathbf{u}, \mathbf{y}; \eta). \tag{I.10}$$

To select specific values for all these parameters we start by formulating the mean squared error (MSE) of the estimate $\hat{\boldsymbol{\theta}}(P)$ in (I.6), the so-called estimation error,

$$\mathrm{MSE}\left(\hat{\boldsymbol{\theta}}\right) = \left\| \hat{\boldsymbol{\theta}}(f_{\mathrm{DL}}(\mathbf{u}, \mathbf{y}; \eta)) - \boldsymbol{\theta}_0 \right\|^2, \tag{I.11}$$

where $\|\cdot\|^2$, denotes the 2-norm and $\boldsymbol{\theta}_0$ denotes the true FIR parameters which corresponds to the truncated true impulse response. Here we make use of the neural network model (I.10) of P when forming the MSE. The parameters $\hat{\eta}$ to use in (I.10) are found by simply minimizing this estimation error,

$$\hat{\eta} = \arg\min_{\eta} \frac{1}{M} \sum_{i=1}^{M} \left\| \hat{\boldsymbol{\theta}} \left(f_{\mathrm{DL}}(\mathbf{u}^{(i)}, \mathbf{y}^{(i)}; \eta) \right) - \boldsymbol{\theta}_0^{(i)} \right\|^2, \tag{I.12}$$

where M is the number of training examples. To set the terminology straight we use the term *training* for the minimization of the estimation error w.r.t. η whereas *estimation* refers to the computation of a point estimate of the FIR parameters by minimizing the one step prediction error w.r.t. $\boldsymbol{\theta}$.

The following sections describe our method and an overview is provided in Algorithm 1.

4.1 Regularization model

We know that P must be positive semi-definite. Inspired by this fact we choose P as a weighted sum of rank-one positive definite matrices. The idea behind this choice is to have a representation that is flexible enough to represent all possible regularization matrices and at the same time encode the knowledge that the optimal regularization matrix is a rank-one matrix (see Appendix 8).

Our rank-one matrices are constructed as outer products of a vector $\mathbf{s}_i$ with itself where the elements of the vectors are free parameters. The weights, w_i, that weight our rank-one matrices, are modelled as the output of a softmax layer from a neural network. The input to this neural network is $\mathbf{u}$ and $\mathbf{y}$ as well as the nonregularized least squares solution, which only depends on $\mathbf{u}$ and $\mathbf{y}$. Hence, we have,

$$P = \sum_{i=1}^{n_m} w_i(\mathbf{u}, \mathbf{y}, \hat{\boldsymbol{\theta}}_{\mathrm{LS}}; \eta') \mathbf{s}_i \mathbf{s}_i^{\mathrm{T}}, \tag{I.13}$$

where n_m is the number of rank-one matrices used to represent the regularization matrix and η' is the parameters of the neural network. We use $\eta = \{\eta', \mathbf{s}_1, \dots, \mathbf{s}_{n_m}\}$ to collect all the parameters in the regularization matrix model.

4.2 Neural network model

We have made use of four fully connected layers in our neural network, where the final layer is a softmax layer producing weights between 0 and 1. The other activation functions for the fully connected layers are Rectified Linear Units (ReLU) which are defined as $\mathrm{ReLU}(x) = \max(0, x)$. A typical layer of the network is thus written as $\mathbf{h}^{(i+1)} = \mathrm{ReLU}(W\mathbf{h}^{(i)} + \mathbf{b})$, where $\mathbf{h}^{(i)}$

denotes the input to the layer, $\mathbf{h}^{(i+1)}$ denotes the output from the layer, W denotes a so-called weight matrix of dimensions $\dim(\mathbf{h}^{(i+1)}) \times \dim(\mathbf{h}^{(i)})$ and $\mathbf{b}$ denotes a so-called bias term of dimension $\dim(\mathbf{h}^{(i)})$. Both the weight matrices and the bias terms are part of the parameters η' describing the network, i.e. $W^{(i)} \in \eta'$ and $\mathbf{b}^{(i)} \in \eta'$. To regularize the training procedure we add a dropout layer after the softmax layer which with 30% probability sets a weight to zero during training. This is a standard technique to avoid overfitting in NN [18]. The input to the network is the input $\mathbf{u}$ and output $\mathbf{y}$ sequence of the system we want to estimate and the corresponding non-regularized least squares solution $\hat{\boldsymbol{\theta}}_{\text{LS}}$. The resulting network is schematically illustrated in Fig. 2.

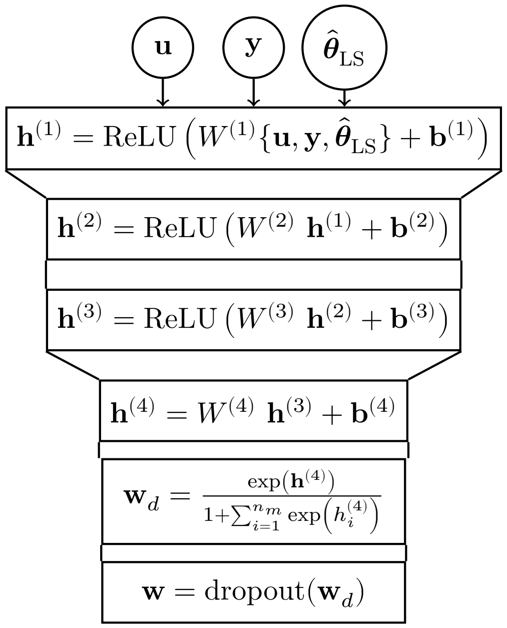

$\mathbf{u}$
$\mathbf{y}$
$\hat{\boldsymbol{\theta}}_{\text{LS}}$
$\mathbf{h}^{(1)} = \text{ReLU}\left(W^{(1)}\{\mathbf{u},\mathbf{y},\hat{\boldsymbol{\theta}}_{\text{LS}}\} + \mathbf{b}^{(1)}\right)$
$\mathbf{h}^{(2)} = \text{ReLU}\left(W^{(2)}\ \mathbf{h}^{(1)} + \mathbf{b}^{(2)}\right)$
$\mathbf{h}^{(3)} = \text{ReLU}\left(W^{(3)}\ \mathbf{h}^{(2)} + \mathbf{b}^{(3)}\right)$
$\mathbf{h}^{(4)} = W^{(4)}\ \mathbf{h}^{(3)} + \mathbf{b}^{(4)}$
$\mathbf{w}_d = \frac{\exp(\mathbf{h}^{(4)})}{1+\sum_{i=1}^{n_m}\exp\left(h_i^{(4)}\right)}$
$\mathbf{w} = \text{dropout}(\mathbf{w}_d)$

Figure 2: Schematic description of our neural network. The weights from (I.13) are denoted by $\mathbf{w} = \{w_1, \dots, w_{n_m}\}$. The dimensions of $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}$ and $\mathbf{h}^{(4)}$ are $600, 300, 200$ and 500 respectively. Note that the dimension of the final layer is equal to the number of matrices used.

4.3 Normalizing the data

A key aspect to successfully train neural networks is the normalization of the input and output of the network by subtracting the mean and dividing with the standard deviation. We notice that we can, without loss of generality, normalize each data example of $\mathbf{y}$ and $\mathbf{u}$ if we at the same time do the corresponding scaling of the impulse response $\boldsymbol{\theta}$ to keep the analytic relationship intact.

The non-regularized least square solution, $\hat{\boldsymbol{\theta}}_{\text{LS}}$, that we use as an input to the network is also straightforward normalize with the statistics form the true impulse response calculated from the training data. Finally, we want to normalize the networks dependence of the vectors $\mathbf{s}_i$. The outer product of these vectors should correspond to the optimal regularization (see Section 4.1 and Appendix 8). To enforce that they follow the same statistics, we initialize the vectors $\mathbf{s}_i$ with unit Gaussian and then multiply we the standard deviation and add the mean of the true impulse response.

Algorithm 1 Training procedure

```
1: Collect evaluation data.
2: Collect training data with similar behaviour as evaluation data (e.g.
   through simulations).
3: Normalize $\mathbf{u}$ and $\mathbf{y}$ for each example (Section 4.3).
4: Normalize $\hat{\boldsymbol{\theta}}_{\text{LS}}$ with statistics from the whole training dataset (Section
   4.3).
5: Find $\hat{\eta}$ using training data by minimizing Equation (I.12).
6: Use $\hat{\eta}$ to predict $\hat{\boldsymbol{\theta}}$ on evaluation data.
```

5 Experiment

The model explained in Section 4 is implemented using Tensorflow [1] and our implementation of the model is available on GitHub[1].

5.1 Simulate data using `rss`

To train the model, we use an artificial distribution over real systems to produce input and output sequences along with their true impulse responses. The artificial distribution over systems we use is MATLAB's `rss` function. This function is not ideal as was recently pointed out by for example Rojas et al. [16]. Hence, there is potential to further improve the results by making use of better data.

To generate data we use the same method as Chen et al. [4] with some minor alterations concerning the signal to noise ratio (SNR). The full procedure follows as:

1. Sample a system of order 30 using a slightly modified version of MATLAB's `rss` where the probability of having an integrating pole is zero.

[1]https://github.com/carl-andersson/impest

2. Sample $N = 125$ timesteps from the system at a sampling rate of 3 times the bandwidth, i.e.,

```
bw = bandwidth(m);
f = 3*(bw*2*pi);
md = c2d(m,1/f,'zoh');
```

3. Calculate the true FIR parameters as the truncated impulse response to length $n = 50$.
4. Sample white noise as an input sequence $\mathbf{u}$ and get the corresponding output sequence $\mathbf{y}^*$. Add noise to the output with the SNR drawn randomly from a uniform distribution between 1 and 10, i.e., the noise added has variance that is between 1 and 1/10 of the variance in the output sequence.
5. Repeat all steps until we have M examples from M different systems.

These M training examples are then used in Equation (I.12) and are called the training data. Using the same method we generate a validation set which we also split in two sets depending on the SNR, one with SNR larger than 5.5 and one set with SNR less than 5.5 with roughly $M_v \approx 5\,000$ examples in each.

5.2 Evaluation metrics

To evaluate the performance of the model we use a metric that is calculated as the mean squared error of the estimate normalized with the mean squared error of the least squares solution without any regularization. We denote this metric by S, i.e.,

$$S = \frac{1}{M_v} \sum_{i=1}^{M_v} \left[\frac{\left\| \hat{\boldsymbol{\theta}}\left(f_{\mathrm{DL}}(\mathbf{u}^{(i)}, \mathbf{y}^{(i)}; \eta)\right) - \boldsymbol{\theta}_0^{(i)} \right\|^2}{\left\| \hat{\boldsymbol{\theta}}_{\mathrm{LS}}^{(i)} - \boldsymbol{\theta}_0^{(i)} \right\|^2} \right], \tag{I.14}$$

where M_v denotes the number of examples in the validation set. This metric makes sure that each impulse response gets equal weighting when computing the performance of the algorithm and measures the average effect of the regularization. A perfect match for this measure corresponds to a measure of 0.

Chen et al. [4] make use of a slightly different metric defined as

$$\tilde{S} = \frac{1}{M_v} \sum_{i=1}^{M_v} 100 \left(1 - \left[\frac{\left\| \hat{\boldsymbol{\theta}}\left(f_{\mathrm{DL}}(\mathbf{u}^{(i)}, \mathbf{y}^{(i)}; \eta)\right) - \boldsymbol{\theta}_0^{(i)} \right\|^2}{\left\| \boldsymbol{\theta}_0^{(i)} - \bar{\boldsymbol{\theta}}_0^{(i)} \right\|^2} \right] \right)$$

where $\bar{\boldsymbol{\theta}}_0^{(i)}$ is the mean of $\boldsymbol{\theta}_0^{(i)}$. This metric averages over a so-called 'model fit', i.e. how well the estimated parameters fit the true impulse response. Besides the shifting and scaling in $\tilde{S}$, the only difference between the two metrics is the normalization factor used, where S is normalized with the least squares estimation error and $\tilde{S}$ is normalized with the variance of the true impulse response. We have empirically observed that the terms in $\tilde{S}$ might vary a lot between different examples and the average might thus be dominated by a few examples leading to measures that are hard to interpret. Our slightly modified metric S will on the other hand measure the average effect of using a regularization method compared to not using a regularization method, which seem to result in a more stable performance indicator.

5.3 Simulation results

The model is trained using $M = 1\,000\,000$ training examples for roughly 2.5 hours using a desktop computer with an Nvidia Titan Xp GPU. The chosen hyperparameters of the model is described in Figure 2. Note that while training require a GPU with large memory, the evaluation can easily be done in CPU on an ordinary computer. We are using early stopping as a stopping criteria even though the model essentially does not seem to overfit with this amount of training data. The model is not very dependent on the number of examples in the training data either. Even with $M = 10\,000$ it managed to achieve comparable results to previous methods.

Fig. 3 shows a subset of the matrices $\mathbf{s}_i\mathbf{s}_i^T$ from (I.13) after training. Note that the matrices have an oscillating pattern with different periods and a decay towards zero for parameters with high index (lower right corner). Fig. 4 shows three different regularization matrices for an example in the validation dataset. We can see that the deep learning regularization seems to capture the behaviour of the optimal regularization matrix fairly well. In Fig. 5 we can compare the estimates from the different regularization approaches.

The trained model does not produce a useful regularization matrix for all examples. In cases where it fails the neural network seems to fail in the same way for all examples by producing a similar regularization matrix for each example with a bad performance for S as consequence. Despite this problem, the model manages to produce average results which are comparable or better than previous methods which is reflected in Table 1. We can see that the performance decreases when the SNR gets larger. This is due to the improved effectiveness of the least squares estimate and we are thus less dependent on the regularization. For comparison we also present the result for the optimal regularization which of course is unachievable since it depends on the true impulse response, but it is still an useful lower bound.

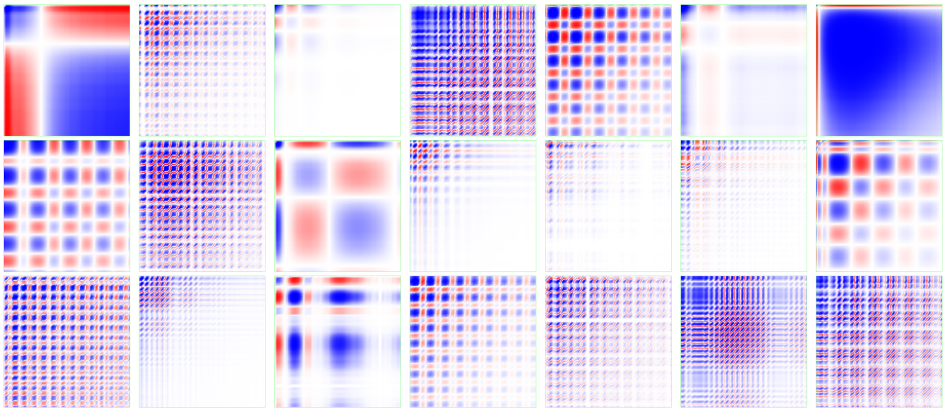


Figure 3: Illustration of 21 matrices from $\mathbf{s}_i {\mathbf{s}_i}^T$ after training. The matrices are rescaled to the interval $[+1, -1]$, where blue indicates a positive value, white around zero and red indicates a negative value. The upper left corner corresponds to the lowest index of the estimate $\boldsymbol{\theta}$.

5.4 Real data results

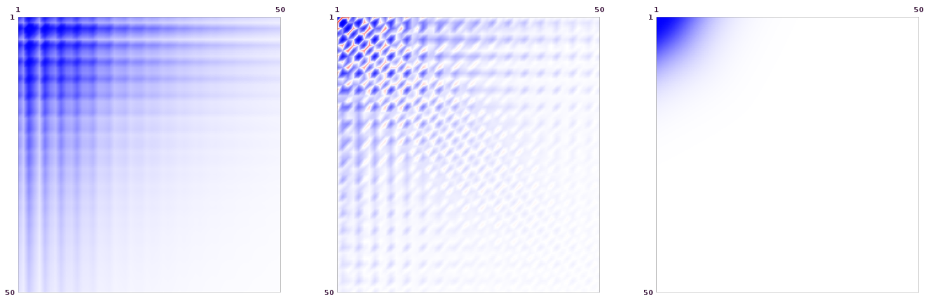
To show that our method at least does not give unreasonable estimates for real systems and data we test our method on data measured at a processing plant. See e.g. [3] for an introduction to this problem area. We do not know the true parameter values for these systems, implying that we cannot evaluate the performance of the estimates. We use the same trained network as we evaluated in the previous section. In Fig. 6 we simply present an input sequence and the corresponding output sequence from a real system together with the estimates produced by our method compared to using least squares without regularization. We note also that the results seems reasonable and that the regularization removes a lot of noise in the estimation.

6 Conclusion

In this paper we present a method to regularize an impulse response estimation problem. We train a model with simulated data in a data-driven fashion to encode this regularization with parameters in a neural network. A trained model can then be used to improve the mean squared error of the new estimations. The results of our method seems promising and there is plenty of scope for future work along this line of research, both when it comes to impulse response estimation, but also for other problems. We find it especially interesting that this model can mimic the optimal regularization matrix to higher degree than previous methods which we believe is the reason

Table 1: Comparing the different models using the metric from (I.14) evaluated using the validation set. LS stands for Least Squares, OR stands for Optimal regularization (see Appendix 8), GP stands for Gaussian process regularization and DL stands for deep learning regularization. LS, OR and GP are not data driven approaches and they are thus not dependent on any training data.

Model	SNR < 5.5	SNR > 5.5
LS	1	1
OR	0.04	0.05
GP	0.31	0.40
DL	0.20	0.23



(a) Optimal regularization (b) Deep learning regularization (c) Gaussian process regularization

Figure 4: Comparison between rescaled inverse regularization matrices for a validation example where our method captures the behaviour of the optimal regularization matrix.

for why it sometimes produce better estimates.

Although training our model is quite time consuming, estimating the impulse response using our method is very fast since it only involves a couple of matrix multiplications to compute the regularization matrix, whereas the method of Chen et al. [4] needs to solve an optimization problem for each example.

7 Future work

We are planning to further investigate how one can make use of real data from e.g. the process industry. This would make it possible to use large amounts of collected data to improve the estimated parameters in a data-driven manner.

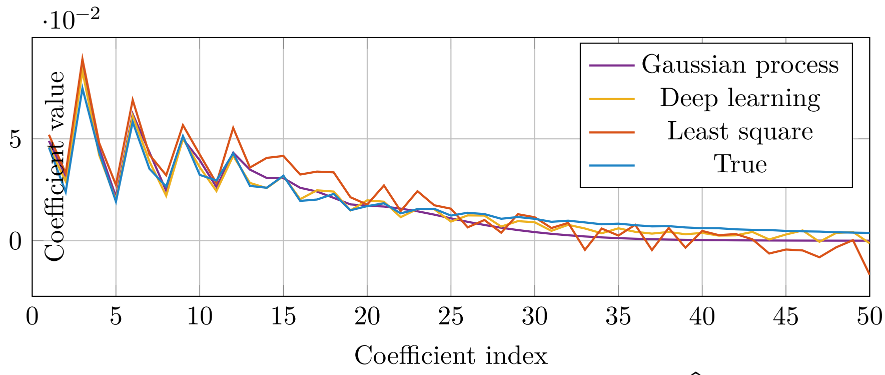
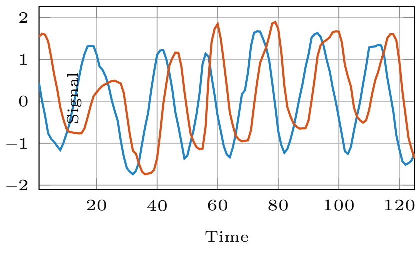
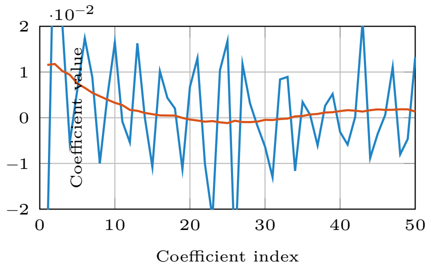


Figure 5: Estimates of the impulse response coefficients, $\hat{\boldsymbol{\theta}}$, using the inverse regularization matrices from Fig. 4 and the same input and output sequence.



(a) An input sequence (blue) and the corresponding output (red) for a real system.



(b) Impulse response estimates using the data in Fig. 6a for regular least squares (blue) and our method (red).

Figure 6: Our method applied to an example from a real system and real measured data.

The process industry has a lot of data available and makes extensive use of linear models.

The idea of learning a prior by representing it with a regularization matrix in the form of a neural network is not unique to the problem of estimating the impulse response. It could easily be generalized to other situations where the least squares solution is available but the prior of the solution is either unknown or intractable. If one can simulate many such systems at low cost, or have data from true systems available, formulating a regularization matrix as a neural network might be a tractable way of regularizing the estimate.

The presented approach can easily be extended to multi-input multi-output systems where the only difference is that the dimension of the input and output sequences and the parameters $\boldsymbol{\theta}$ increases. The deep structure of the model automatically induces any relevant connection between the different system input and output components present in the training data.

Finally we want to stress that this is just one example of what one might do with deep learning in system identification. There might and should be other areas where it is possible to make use of either simulated or real data to improve standard methods, or invent new methods for system identification. For example it might be worth looking into Recurrent Neural Networks (RNN) such as Long Short Term Memory (LSTM)[10], Gated Recurrent Units (GRU) [5] or Stocastic Recurrent Neural Network (SRNN)[7] and apply it in a system identification setting or even to bring some of the system identification knowledge of dynamical systems to the field of deep learning.

Acknowledgements

We would like to thank Professor Krister Forsman at Perstorp and the Norwegian University of Science and Technology for providing us with the real world data. We would also like to thank Dr Jack Umenberger for useful feedback on an early draft of this work.

References

[1] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2016. arXiv: `1603.04467`.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] A. C. Bittencourt, A. Isaksson, D. Peretzki, and K. Forsman. “An algorithm for finding process identification intervals from normal operating data”. In: *Processes* 3.2 (2015), pp. 357–383.

[4] T. Chen, H. Ohlsson, and L. Ljung. “On the estimation of transfer functions, regularizations and Gaussian processes - Revisited”. In: *Automatica* 48.8 (2012), pp. 1525–1535.

[5] K. Cho, B. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014.

[6] D. C. Cirean, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidthuber. “Flexible, high performance convolutional neural networks for image classification”. In: *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence (IJCAI)*. 2011.

[7] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. “Sequential Neural Models with Stochastic Layers”. In: *Advances in Neural Information Processing Systems 29*. 2016, pp. 2199–2207. arXiv: 1605.07571.

[8] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016.

[9] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

[10] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997). arXiv: 1206.2944.

[11] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[12] L. Ljung, ed. *System Identification (2nd Ed.): Theory for the User*. Prentice Hall PTR, 1999.

[13] G. Pillonetto, F. Dinuzzo, T. Chen, G. De Nicolao, and L. Ljung. “Kernel methods in system identification, machine learning and function estimation: a survey”. In: *Automatica* 50.3 (2014), pp. 657–682.

[14] G. Pillonetto, A. Chiuso, and G. De Nicolao. “Prediction error identification of linear systems: a nonparametric Gaussian regression approach”. In: *Automatica* 47.2 (2011), pp. 291–305.

[15] G. Pillonetto and G. De Nicolao. “A new kernel-based approach for linear system identification”. In: *Automatica* 46.1 (2010), pp. 81–93.

[16] C. R. Rojas, P. E. Valenzuela, and R. A. Rojas. “A Critical View on Benchmarks based on Randomly Generated Systems”. In: *IFAC-PapersOnLine* 48.28 (2015), pp. 1471–1476.

[17] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky. “Nonlinear black-box modeling in system identification: a unified overview”. In: *Automatica* 31.12 (1995), pp. 1691–1724.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

8 Appendix: Optimal regularization

The optimal regularization matrix is a term coined by Chen et al. [4]. It corresponds to the regularization matrix that is optimal in the sense that it minimizes the mean squared error. The optimal regularization matrix can be written as, $P = \sigma^{-2}\boldsymbol{\theta}_0\boldsymbol{\theta}_0^{\mathrm{T}}$ [4], where σ^2 is the variance of the additive noise $v(t)$ in (I.1), $\boldsymbol{\theta}_0$ is the true impulse response of the system without noise.

Paper II

Title

Deep Convolutional Networks in System Identification

Authors

Carl Andersson*, Antônio H. Ribeiro*, Koen Tiels, Niklas Wahlström, and Thomas B. Schön
*Equal contribution.

Edited version of

C. Andersson, A. L. Ribeiro, K. Tiels, N. Wahlström, and T. B. Schön. "Deep convolutional networks in system identification". In: *Proceedings of the 58th IEEE Conference on Decision and Control (CDC).* Nice, France, 2019

Deep Convolutional Networks are Useful in System Identification

Abstract

Recent developments within deep learning are relevant for nonlinear system identification problems. In this paper we establish connections between the deep learning and the system identification communities. It has recently been shown that convolutional architectures are at least as capable as recurrent architectures when it comes to sequence modelling tasks. Inspired by these results we explore the explicit relationships between the recently proposed temporal convolutional network (TCN) and classic system identification model structures. We end the paper with an experimental study where we provide results on two real-world problems, the well-known Silverbox dataset and a newer dataset originating from ground vibration experiments on an F-16 fighter aircraft.

1 Introduction

Deep learning has, over the past decade, had a massive impact on several branches of science and engineering, including for example computer vision [24], speech recognition [17] and natural language processing [33]. While

[0]This work has been supported by the Brazilian agencies CAPES, CNPq and FAPEMIG, by the Swedish Research Council (VR) via the project *NewLEADS – New Directions in Learning Dynamical Systems* (contract number: 621-2016-06079), and by the Swedish Foundation for Strategic Research (SSF) via the project *ASSEMBLE* (contract number: RIT15-0012). C. Andersson, A. H. Ribeiro, K. Tiels, N. Wahlström and T. B. Schön are with the Dept. of Information Technology, Uppsala University, 751 05 Uppsala, Sweden. A. Ribeiro is also with the Graduate Program in Electrical Engineering at Universidade Federal de Minas Gerais (UFMG) - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil

the basic model class—neural networks—has been around for more than 70 years [32], there has been quite a few interesting and highly relevant technical developments within the deep learning community that has, to the best of our knowledge, not yet been fully exploited within the system identification community. Just to mention a few of these developments we have; new regularization methods [13, 53, 62], new architectures [15, 50, 54], improved optimization algorithms [3, 21, 23], new insights w.r.t. activation functions [16, 29, 30]. Furthermore, the capability to significantly increase the depth [15, 50, 54] in the models has further improved the performance. Most of the existing model architectures have been made easily available through high quality open source frameworks [1, 39], allowing deep learning to be easily implemented, trained, and deployed.

The deep learning developments that are most relevant for system identification are probably the ones that can be found under the name of *sequence learning*. The classical architecture for sequence learning tasks has been the recurrent models, such as the recurrent neural networks (RNN) and its extensions which include the long short-term memory (LSTM) [19] and the gated recurrent units (GRU) [14]. Within the neighbouring area of computer vision, the use of the so-called convolutional neural networks (CNNs) [67] has had a very strong impact on tasks such as image classification [25], segmentation [28] and object detection [43]. Interestingly, it has recently [2] been shown that the CNN architecture is highly useful also for sequence learning tasks. More specifically, the so-called temporal CNN (TCN) can match or even outperform the older recurrent architectures in language and music modelling [2, 7, 35], text-to-speech conversion [35], machine translation [11, 22] and other sequential tasks [2]. We will, for this reason, focus this paper on making use of TCNs for nonlinear system identification.

Neural networks have enjoyed a long and fruitful history [5, 34, 51] also within the system identification community, where they remain a popular choice when it comes to modeling of nonlinear dynamical systems [6, 8, 26, 59, 66].

The reason we are writing this paper is to reinforce the bridge between the system identification and the deep learning communities. We do believe that there is a lot to be gained from doing so and more specifically we will describe the new TCN model from a system identification point of view (Section 2). We will also show that there are indeed interesting connections between the deep TCN structure and model structures used within system identification (Section 3). Perhaps most importantly we will in Section 4 provide experimental results on two real-world problems (the Silverbox [63] and the F-16 [47] datasets) and on a toy problem.

2 Neural networks for temporal modeling

The neural network is a universal function approximator [20] with a sequential model structure of the form:

$$\hat{y} = g^{(L)}(z^{(L-1)}), \tag{II.1a}$$

$$z^{(l)} = g^{(l)}(z^{(l-1)}), \quad l = 1, \dots, L-1, \tag{II.1b}$$

$$z^{(0)} = x, \tag{II.1c}$$

where $x, z^{(l)}, \hat{y}$ denotes the input, the hidden variables and the output, respectively. The transformation within each layer is of the form $g^{(l)}(z) = \sigma(W^{(l)}z + b^{(l)})$ consisting of a linear transformation $W^{(l)}z + b^{(l)}$ followed by a scalar nonlinear mapping that acts element-wise. In the final (output) layer the nonlinearity is usually omitted, i.e. $g^{(L)}(z) = W^{(L)}z + b^{(L)}$.

The neural network parameters $\{W^{(l)}, b^{(l)}\}_{l=1}^{L}$ are usually referred to as the weights $W^{(l)}$ and the bias terms $b^{(l)}$ and they are estimated by minimizing the prediction error $\frac{1}{N}\sum_{k=1}^{N} \|\hat{y}[k] - y[k]\|^2$ for some training dataset $\{x[k], y[k]\}_{k=1}^{N}$.

To train deep neural networks with many hidden layers (large L) have been proved to be a notoriously hard optimization problem, where we face challenges such as getting stuck in bad local minimas, exploding and/or vanishing gradients, and dealing with large-scale datasets. It is only over the past decade that these challenges have been addressed with improved hardware and algorithms to the extent that training truly deep neural networks has become feasible. We will very briefly review some of these developments below.

2.1 Temporal convolutional network

As the name suggests, the temporal convolutional network (TCN) is based on convolutions [2]. The use of TCNs within a system identification setting can in fact be interpreted as using the nonlinear ARX model as the basic model component:

$$\hat{y}[k+1] = g(x[k], \dots x[k-(n-1)]), \tag{II.2}$$

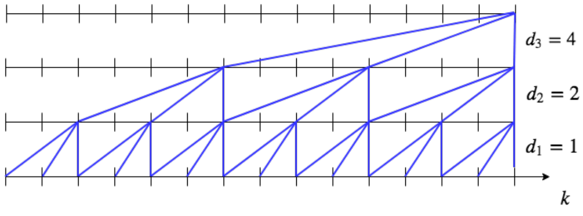
with $x[k] = (u[k],\ y[k])$. We will proceed with this interpretation, where (II.2) would correspond to a one-layer TCN model.

A full TCN can be understood as a sequential construction of several nonlinear ARX models stacked on top of each other:

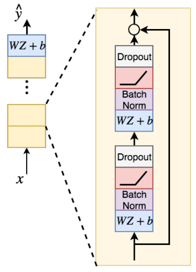
$$\hat{y}[k+1] = g^{(L)}(Z^{(L-1)}[k]), \tag{II.3a}$$

$$z^{(l)}[k] = g^{(l)}(Z^{(l-1)}[k]), \quad l = 1, \dots, L-1, \tag{II.3b}$$

$$z^{(0)}[k] = x[k], \tag{II.3c}$$



(a)



(b)

Figure 1: Illustration of the temporal convolution network (TCN) with residual blocks. (a) Temporal convolutional network with dilated causal convolutions with dilation factor $d_1 = 1$, $d_2 = 2$ and $d_3 = 4$ and kernel size $n = 3$. (b) A TCN residual block. Each block consists of two dense layers and an identity (or linear) map on the skip connection.

where:

$$Z^{(l-1)}[k] = \left(z^{(l-1)}[k], z^{(l-1)}[k-d_l], \dots, z^{(l-1)}[k-(n-1)d_l]\right).$$

The number of layers L, the size of each intermediate layer $z^{(l)}[k]$, and the model order n, are all design choices determined by the user. This will also determine the number of parameters included in the model.

For each layer, we optionally introduce a *dilation factor* d_l. With $d_l = 1$ we recover the standard nonlinear ARX model in each layer. With $d_l > 1$ the corresponding output of that layer can represent a wider range of past time instances. The effective memory of that layer will be $(n-1)d_l$. Typically the dilation factor is chosen to increase exponentially with the number of layers, for example $d_l = 2^{(l-1)}$, see Fig. 1a. If we assume that we have the same number of parameters in each layer, the memory will increase exponentially, not only with the number of layers, but also with the number of parameters in the model. This is a very attractive but yet uncommon property for system identification models present in the literature.

Each layer in a TCN can also be seen as dilated causal convolution where n would be the kernel size and $\dim(z^{(l)}[k])$ the number of channels in layer l. These convolutions can be efficiently implemented in a vectorized manner where many computations are reused across the different time steps k. Analogously to what is done in convolutional neural networks we use zero-padding for $z^{(l)}[k]$ where $k < 1$. We refer to [2] for a presentation of TCN based on convolutions.

2.2 Regularization

Similar to other approaches within system identification, L2- and L1- regularization are commonly used to reduce the flexibility of a model and hence avoid overfitting. A number of other regularization techniques have also appeared more specialized to neural networks. One of them is the dropout [53] which is a technique where a random subset of the hidden units in each layer is set to zero during training. New random subsets are drawn and set to zero in each optimization step which effectively means that a random subnetwork is trained during each iteration.

Data augmentation is very common in classification problems and it can also be interpreted as a regularization technique. It is used to artificially increase the training dataset by utilizing the fact that the class is invariant under some transformation of the input (e.g. translation for image) or in the presence of some low intensity noise (e.g. salt pepper noise for images).

Finally, early stopping is a pragmatic approach in which, as the name suggests, the optimization algorithm is interrupted before convergence. The stopping point is chosen as the point where a validation error is minimized. Hence, it avoids overfitting and can as such also be interpreted as a regularization technique.

2.3 Batch normalization

Before training a neural network, the inputs are commonly normalized by subtracting the mean and dividing by the variance. The purpose of this is to avoid early saturation of the activation function and assuring that values in the proceeding layers are within the same dynamic range. In deep networks it is beneficial to not only normalize the input layer, but also the intermediate hidden layers. This idea is exploited in batch normalization [21] which, in addition to this normalization, introduces scaling parameters γ and a shift β to be learned during training. The output of the layer is then:

$$\tilde{z}^{(l)}[k] = \gamma \bar{z}^{(l)}[k] + \beta. \tag{II.4}$$

where $\bar{z}^{(l)}[k]$ is normalized version of layer l output. The parameters γ and β will be trained jointly with all other parameters of the network. Batch normalization has become very popular in deep learning models.

An alternative to batch normalization is weight normalization which is, essentially, a reparametrization of the weight matrix, decoupling the magnitude and direction of the weights [45].

2.4 Residual blocks

A residual block is a combination of possibly several layers together with a skip connection

$$z^{(l+p)} = \mathcal{F}(z^{(l)}) + z^{(l)}, \tag{II.5}$$

where the skip connection adds the value from the input of the block to its output. The purpose of the residual block is to let the layers learn deviations from the identity map rather than the whole transformation. This property is beneficial, especially in deep networks where we want the information to gradually change from the input to the output as we proceed through the layers. There is also some evidence that this makes it easier to train deeper neural networks [15].

We employ residual blocks in our models by following the model structure in [2]. Each block consist of one skip connection and two linear mappings, each of them followed by batch normalization, activation function and dropout regularization, see Fig. 1b. For both mappings a common dilation factor is used and hence the whole block can be seen as one of the layers $g^{(l)}(z)$ in the TCN model (II.3). Note that, with our formulation, the skip connection only passes $z^{(l-1)}[k]$ to the next layer and not the whole $Z^{(l-1)}[k]$ for each time instance k. In cases where $z^{(l-1)}[k]$ and $z^{(l)}[k]$ are of different dimensions, a linear mapping is used between them. The coefficients of this linear mapping are also learned during training.

2.5 Optimization algorithms

Neural networks are trained using gradient-based optimization methods. At each iteration only a random subset of the training data is used to compute the gradient and update the parameters. This is called mini-batch gradient descent and is a crucial component for efficient training of a neural network when the dataset is large.

Multiple extensions to mini-batch gradient descent have been proposed to make the learning more efficient. Momentum [42] applies a first order low-pass filter to the stochastic gradients to compensate for the noise introduced by the random sub-sampling. RMSprop [55] uses a low-passed version of the squared gradients to scale the learning rate in the different dimensions. One of the most popular optimization method today is referred to as Adam [23] which basically amounts to using RMSprop with momentum.

3 Connections to system identification

This section describes equivalences between the basic TCN architecture (i.e. without dilated convolutions and skip connections) and models in the system identification community, namely Volterra series and block-oriented models. The discussion is limited to the nonlinear FIR case (where $x = u$) instead of the more general NARX case ($x = (u, y)$) considered in (II.2), and to single input single output systems.

3.1 Connection with Volterra series

A Volterra series [46] can be considered as a Taylor series with memory. It is essentially a polynomial in (delayed) inputs $u[k], u[k-1], \ldots$. Alternatively, a Volterra series can be considered as a nonlinear generalization of the impulse response $h_1[\tau]$. The output of a Volterra series is obtained using higher-order convolutions of the input with the Volterra kernels $h_d[\tau_1, \ldots, \tau_d]$ for $d = 0, 1, \ldots, D$. These kernels are the polynomial coefficients in the Taylor series.

The basic TCN architecture is essentially the same as the time delay neural network (TDNN) in [61], except for the zero padding [2] and the use of ReLU activations instead of sigmoids. The TDNN has been shown to be equivalent to an infinite-degree ($D \to \infty$) Volterra series in [65]. This connection is made explicit in [65] by showing how to compute the Volterra kernels from the estimated network weights W. The key ingredient is to use a Taylor series expansion of the activation functions σ either around zero (the bias term b is then considered part of the activation function) or alternatively around the bias values if the Taylor series around zero does not converge for example.

3.2 Connection with block-oriented models

Block-oriented models [12, 49] combine linear time-invariant (LTI) subsystems (or blocks) and nonlinear static (i.e. memoryless) blocks. For example, a Wiener model consists of the cascade of an LTI block and a nonlinear static block. For a Hammerstein model, the order is reversed: it is a nonlinear block followed by an LTI block. Generalizations of these simple structures are obtained by putting more blocks in series (as in [64] for Hammerstein systems) or in parallel branches (as in [48] for Wiener-Hammerstein systems) and/or to consider multivariate nonlinear blocks.

A basic two-layer TCN can be considered as a parallel Wiener-Hammerstein model where the linear blocks are FIR filters and the static nonlinear blocks are the activation functions (including the bias). A multi-layer basic TCN can be considered as cascading parallel Wiener models, one for each hidden layer, but this time with multivariate nonlinear blocks. The linear output layer corresponds to adding FIR filters at the end of each parallel branch. The layers can be squeezed together to form less but larger layers (cf. squeezing together the sandwich model discussed in [38]). This is due to the fact that the dynamics consist of time delays and time delays can be placed before or after a static nonlinear function without changing the resulting output $(q^{-1}\sigma(z[k]) = \sigma(q^{-1}z[k]) = \sigma(z[k-1]))$. The TCN model could be squeezed down to a parallel Wiener model.

3.3 Conclusion

The basic TCN architecture is equivalent to Volterra series and parallel Wiener models. They are thus all universal approximators for time-invariant systems with fading memory [4]. This equivalence does *not* mean that all these model structures can be trained with equal ease and will perform equally well in all identification tasks. For example, a Volterra series uses polynomial basis functions, whereas TDNNs use sigmoids and TCNs use ReLU activation functions. Depending on the system at hand, one basis function might be better suited than another to avoid bad local minima and/or to obtain both an accurate and sparse representation.

4 Numerical results

We now present the performance of the TCN model on three system identification problems. We compare this model with the classical NARX Multilayer Perceptron (MLP) network with two layers and with the Long Short-Term Memory (LSTM) network. When available, results from other papers on the same problem are also presented.

We make a distinction between *training*, *validation* and *test* datasets. The *training* dataset is used for estimating the parameters. The performance in the *validation* data is used as the early stopping criteria for the optimization algorithm and for choosing the best hyper-parameters (i.e. neural network number of layers, number of hidden nodes, optimization parameter, and so on). The *test* data allows us to assess the model performance on unseen data. Since the major goal of the first example is to compare different hyper-parameter choices we do not use a *test* set.

In all the cases, the neural network parameters are estimated by minimizing the mean square error using the Adam optimizer [23] with default parameters and an initial learning rate of lr $= 0.001$. The learning rate is reduced whenever the validation loss does not improve for 10 consecutive epochs.

We use the Root Mean Square Error ($\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\|\hat{y}[k] - y[k]\|^2}$) as metric for comparing the different methods in the validation and test data. Throughout the text we will make clear when the predicted output $\hat{y}$ is computed through the free-run simulation of the model and when it is computed through the one-step-ahead prediction.

4.1 Example 1: Nonlinear toy problem

The nonlinear system [5]:

$$\begin{aligned} y^*[k] &= (0.8 - 0.5e^{-y^*[k-1]^2})y^*[k-1] - \\ &\quad (0.3 + 0.9e^{-y^*[k-1]^2})y^*[k-2] + u[k-1] + \\ &\quad 0.2u[k-2] + 0.1u[k-1]u[k-2] + v[k], \\ y[k] &= y^*[k] + w[k], \end{aligned} \tag{II.6}$$

was simulated and the generated dataset was used to build neural network models. Fig. 2 shows the validation results for a model obtained for a training and validation set generated with white Gaussian process noise v and measurement noise w. In this section, we repeat this same experiment for different neural network architectures, with different noise levels and different training set sizes N.

In each possible training configuration, we have trained the TCN for all possible combinations of: number of hidden layers in {16, 32, 64, 128}; dropout rate in {0.0, 0.3, 0.5, 0.8}; number of residual blocks in {1, 2, 4, 8}; for the kernel size in {2, 4, 8, 16}; for the presence or absence of dilations; and, for the use of *batch norm*, *weight norm* or nothing after each convolutional layer. We have trained the MLP for all combination of: number of hidden layers in {16, 32, 64, 128, 256}; model order n in {2, 4, 8, 16, 32, 64, 128}; activation function in {ReLU, sigmoid}. Finally, we trained the LSTM for all combinations of: number of hidden layers in {16, 32, 64, 128}; number of stacked number of LSTM layers in {1, 2}; dropout rate in {0.0, 0.3, 0.5, 0.8}.

The best results for each neural network architecture on the validation set are compared in Table 1. It is interesting to see that when few samples ($N = 500$) are available for training, the TCN performs the best among the different architectures. On the other hand, when there is more data ($N = 8\,000$) the other architectures gives the best performance.

Fig. 3 shows how different hyper-parameter choices impact the performance of the TCN. We note that standard deep learning techniques such as dropout, batch normalization and weight normalization did not improve performance. The use of dropout actually hurts the model performance on the validation set. Furthermore, increasing the depth of the neural network does not actually improve its performance and the TCN yields better results

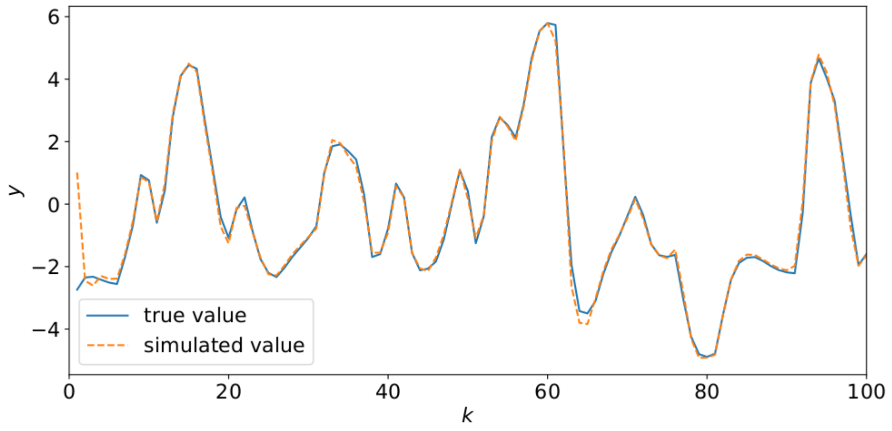


Figure 2: **(Example 1)** Displays 100 samples of the free-run simulation TCN model vs the simulation of the true system. The kernel size for the causal convolutions is 2, the dropout rate is 0, it has 5 convolutional layers and a dilation rate of 1. The training set has 20 batches of 100 samples and was generated with (II.6) for v and w white Gaussian noise with standard deviations $\sigma_v = 0.3$ and $\sigma_w = 0.3$. The validation set has 2 batches of 100 samples. For both, the input u is randomly generated with a standard Gaussian distribution, each randomly generated value held for 5 samples.

in the training set without the use of dilations, which makes sense considering that this model does not require a long memory since the data were generated by a system of order 2.

4.2 Example 2: Silverbox

The Silverbox is an electronic circuit that mimics the input/output behavior of a mass-spring-damper with a cubic hardening spring. A benchmark dataset is available through [63].[1]

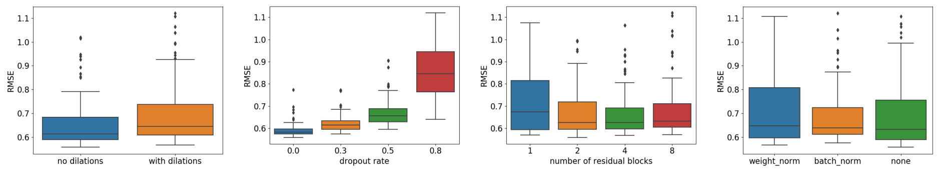
The training and validation input consists of 10 realizations of a random-phase multisine. Since the process noise and measurement noise is almost nonexistent in this system, we use all the multisine realizations for training data, simply training until convergence.

The test input consists of 40 400 samples of a Gaussian noise with a linearly increasing amplitude. This leads to the variance of the test output being larger than the variance seen in the training and validation dataset in the last third of the test data, hence the model needs to extrapolate

[1]Data available for download at:
http://www.nonlinearbenchmark.org/#Silverbox

Table 1: **(Example 1)** One-step-ahead RMSE on the validation set for the models (MLP, LSTM and TCN) trained on datasets generated with: different noise levels (σ) and lengths (N). The standard deviation of both the process noise v and the measurement noise w is denoted by σ. We report only the best results among all hyper-parameters and architecture choices we have tried out for each entry.

	N=500			N=2 000			N=8 000		
σ	LSTM	MLP	TCN	LSTM	MLP	TCN	LSTM	MLP	TCN
0.0	0.362	0.270	**0.254**	0.245	0.204	**0.196**	0.165	**0.154**	0.159
0.3	0.712	0.645	**0.607**	0.602	0.586	**0.558**	**0.549**	0.561	0.551
0.6	1.183	1.160	**1.094**	1.105	1.070	**1.066**	**1.038**	1.052	1.043



(a) (b) (c) (d)

Figure 3: **(Example 1)** Box plots showing how different design choices affect the performance of the TCN for noise standard deviation $\sigma = 0.3$ and training data length $N = 2\,000$. On the y-axis the one-step-ahead RMSE on the validation set is displayed, and on the x-axis we have: in (a) the presence or absence of dilations; in (b) the dropout rate {0.0, 0.3, 0.5, 0.8}; in (c) the number of residual blocks {1, 2, 4, 8}; and, in (d) if *batch norm*, *weight norm* or nothing is used for normalizing the output of each convolutional layer. The variation in performance for the box plot quartiles is achieved through the variation for all the other hyper-parameters not fixed by the hyper-parameter choice indicated on the x-axis.

in this region. Fig. 4 visualizes this extrapolation problem and Table 2 shows the RMSE only in the region where no extrapolation is needed. The corresponding RMSE for the full dataset is presented in Table 3. Similarly to Section 4.1 we found that the TCN did no benefit of the standard deep learning techniques such as dropout and batch normalization. We also see that the LSTM outperforms the MLP and the TCN suggesting the silverbox data is large enough to benefit of the increased complexity of the LSTM.

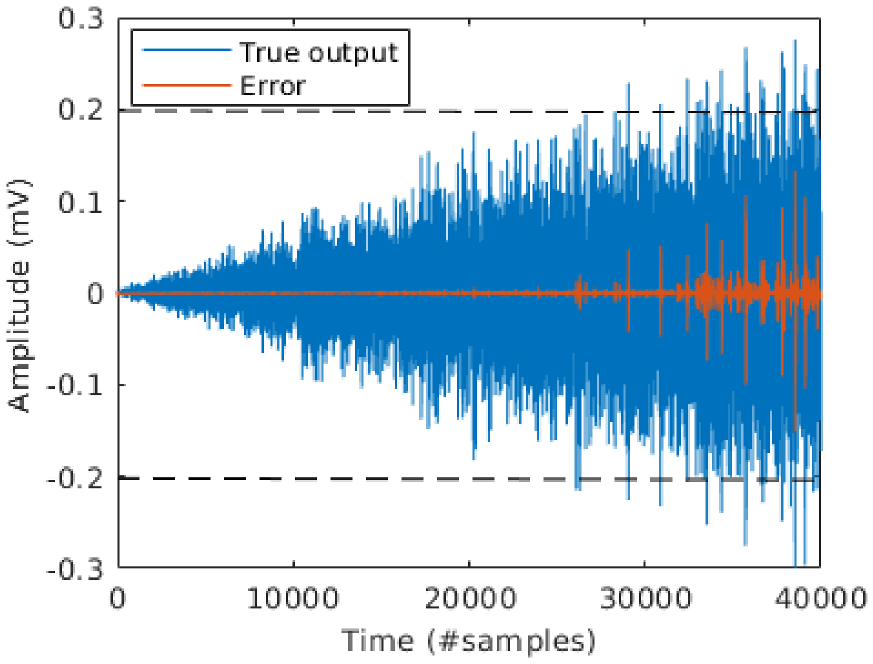


Figure 4: **(Example 2)** The true output and the prediction error of the TCN model in free-run simulation for the Silverbox data. The model needs to extrapolate approximately outside the region ± 0.2 marked by the dashed lines.

4.3 Example 3: F-16 ground vibration test

The F-16 vibration test was conducted on a real F-16 fighter equipped with dummy ordnances and accelerometers to measure the structural dynamics of the interface between the aircraft and the ordnance. A shaker mounted on the wing was used to generate multisine inputs to measure this dynamics. We used the multisine realizations with random frequency grid with 49.0 N

Table 2: **(Example 2)** Free-run simulation results for the Silverbox example on part of the test data (avoiding extrapolation).

RMSE (mV)	Which samples	Approach	Reference
0.7	first 25 000	Local Linear State Space	[60]
0.24	first 30 000	NLSS with sigmoids	[31]
1.9	400 to 30 000	Wiener-Schetzen	[57]
0.31	first 25 000	LSTM	this paper
0.58	first 30 000	LSTM	this paper
0.75	first 25 000	MLP	this paper
0.95	first 30 000	MLP	this paper
0.75	first 25 000	TCN	this paper
1.16	first 30 000	TCN	this paper

Table 3: **(Example 2)** Free-run simulation results for the Silverbox example on the full test data. (*Computed from FIT=92.2886%).

RMSE (mV)	Approach	Reference
0.96	Physical block-oriented	[18]
0.38	Physical block-oriented	[36]
0.30	Nonlinear ARX	[27]
0.32	LSSVM with NARX	[9]
1.3	Local Linear State Space	[60]
0.26	PNLSS	[37]
13.7	Best Linear Approximation	[37]
0.35	Poly-LFR	[58]
0.34	NLSS with sigmoids	[31]
0.27	PWL-LSSVM with PWL-NARX	[10]
7.8	MLP-ANN	[52]
4.08*	Piece-wise affine LFR	[40]
9.1	Extended fuzzy logic	[44]
9.2	Wiener-Schetzen	[57]
3.98	LSTM	this paper
4.08	MLP	this paper
4.88	TCN	this paper

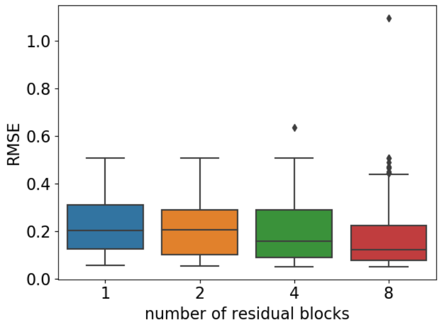


Figure 5: **(Example 3)** Box plot showing how different depths of the neural network affects the performance of the TCN. Should be interpreted in the same way as Fig. 3.

RMS amplitude [47] for training, validating and testing the model.[2]

We trained the TCN, MLP and LSTM networks for all the same configurations used in Example 1. The analysis of the different architecture choices for the TCN in the validation set again reveals that common deep learning techniques such as dropout, batch normalization, weight normalization or the use of dilations do not improve performance. The major difference here is that the use of a deeper neural network actually outperforms shallow neural networks (Fig. 5).

The best results for each neural network architecture are compared in Table 4 for free-run simulation and one-step-ahead prediction. The results are averaged over the 3 outputs. The TCN performs similar to the LSTM and the MLP.

An earlier attempt on this dataset with a polynomial nonlinear state-space (PNLSS) model is reported in [56]. Due to the large amount of data and the large model order, the complexity of the PNLSS model had to be reduced and the optimization had to be focused in a limited frequency band (4.7 to 11 Hz). That PNLSS model only slightly improved on a linear model. Compared to that, the LSTM, MLP, and TCN perform better, also in the frequency band 4.7 to 11 Hz. This can be observed in Fig. 6 and 7, which compare the errors of these models with the noise floor and the total distortion level (= noise + nonlinear distortions), computed using the robust method [41]. Around the main resonance at 7.3 Hz (the wing-torsion mode [47]), the errors of the neural networks are significantly smaller than the total distortion level, indicating that the models do capture significant nonlinear behavior. In contrast to the PNLSS models, the neural networks did not

[2]Data available for download at:
http://www.nonlinearbenchmark.org/#F16

Table 4: **(Example 3)** RMSE for free-run simulation and one-step-ahead prediction for the F16 example averaged over the 3 outputs. The average RMS value of the 3 outputs is 1.0046.

Mode	LSTM	MLP	TCN
Free-run simulation	0.74	0.48	0.63
One-step-ahead prediction	0.023	0.045	0.034

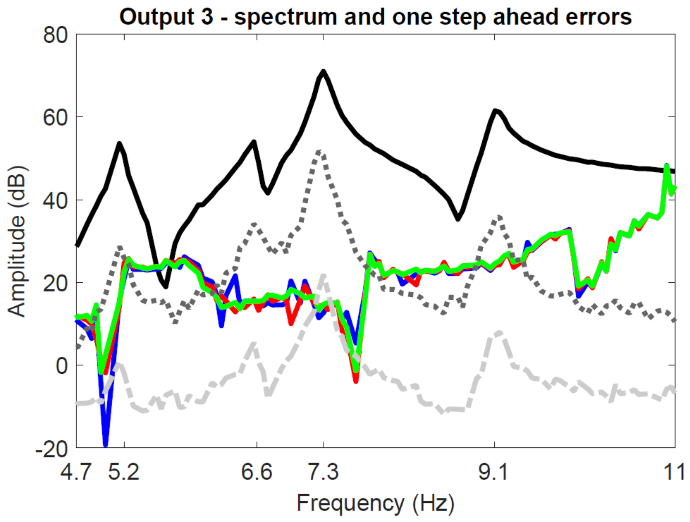


Figure 6: **(Example 3)** In one-step-ahead prediction mode, all tested model structures perform similar. The error is close to the noise floor around the main resonance at 7.3 Hz. (plot only at excited frequencies in $[4.7, 11]$ Hz; true output spectrum in black, noise distortion in grey dash-dotted line, total distortion (= noise + nonlinear distortions) in grey dotted line, error LSTM in green, error MLP in blue, and error TCN in red)

have to be reduced in complexity. Due to the mini-batch gradient descent, it is possible to train complex models on large amounts of data.

5 Conclusion and Future Work

In this paper we applied recent deep learning methods to standard system identification benchmarks. Our initial results indicate that these models have potential to provide good results in system identification problems, even if this requires us to rethink how to train and regularize these models. Indeed, methods which are used in traditional deep learning settings do not always improve the performance. For example, dropout did not yield better

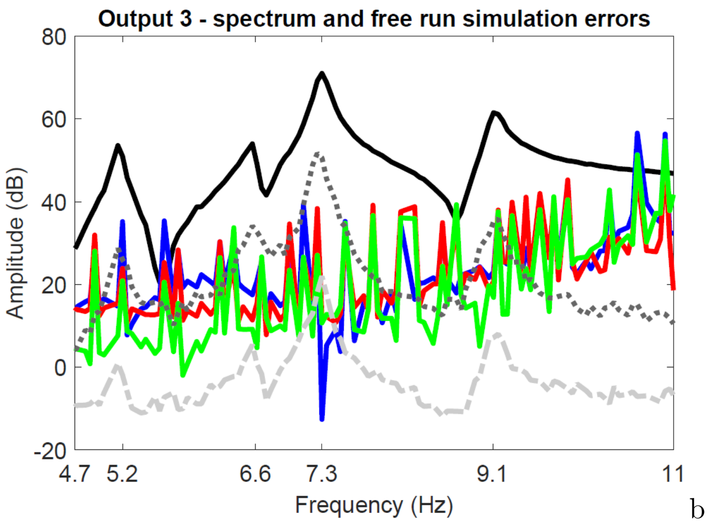


Figure 7: **(Example 3)** In free-run simulation mode, all tested model structures perform similar overall, but the LSTM is better around the wing-torsion mode at 7.3 Hz. (plot only at excited frequencies in $[4.7, 11]$ Hz; true output spectrum in black, noise distortion in grey dash-dotted line, total distortion (= noise + nonlinear distortions) in grey dotted line, error LSTM in green, error MLP in blue, and error TCN in red)

results in any of the problems. Also the long memory offered by the dilation factor in TCNs did not offer any improvement, which is most likely due to the fact that these problems have a relatively short and exponentially decaying memory, as most dynamical systems do. Other findings are that TCNs work well also for small datasets and that LSTMs did show a good overall performance despite being very rarely applied to system identification problems.

Causal convolutions are effectively similar to NARX models and share statistical properties with this class of models. Hence, they are also expected to be biased for settings where the noise is not white. This could justify the limitations of TCNs observed in our experiments. Extending TCNs to handle situations where the data is contaminated with non-white noise seems to be a promising direction in improving the performance of these models. Furthermore, both LSTMs and the dilated TCNs are designed to work well for data with long memory dependencies. Therefore it would be interesting to apply these models to system identification problems where such long term memory is actually needed, e.g. switched systems, or to study if the long-term memory can be translated into accurate long-term predictions, which could have interesting applications in a model predictive control setting.

References

[1] M. Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems". In: (2015). Software available from tensorflow.org.

[2] S. Bai, J. Z. Kolter, and V. Koltun. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". In: (2018).

[3] L. Bottou, F. E. Curtis, and J. Nocedal. "Optimization Methods for Large-Scale Machine Learning". en. In: *SIAM Review* 60.2 (2018), pp. 223–311.

[4] S. Boyd and L. O. Chua. "Fading memory and the problem of approximating nonlinear operators with Volterra series". In: *IEEE Transactions on Circuits and Systems* CAS-32.11 (1985), pp. 1150–1161.

[5] S. Chen, S. A. Billings, and P. M. Grant. "Non-Linear System Identification Using Neural Networks". In: *International Journal of Control* 51.6 (1990), pp. 1191–1214.

[6] D. Masti and A. Bemporad. "Learning Nonlinear State-Space Models Using Deep Autoencoders". In: *2018 IEEE Conf. Decision and Control (CDC).* 17, pp. 3862–3867.

[7] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. “Language Modeling with Gated Convolutional Networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 2017, pp. 933–941.

[8] J. de Jesús Rubio. “Stable Kalman Filter and Neural Network for the Chaotic Systems Identification”. In: *Journal of the Franklin Institute* 354.16 (2017), pp. 7444–7462.

[9] M. Espinoza, K. Pelckmans, L. Hoegaerts, J. A. Suykens, and B. De Moor. “A comparative study of LS-SVM’s applied to the Silver Box identification problem”. In: *IFAC Proceedings Volumes* 37.13 (2004), pp. 369–374.

[10] M. Espinoza, J. A. Suykens, and B. De Moor. “Kernel based partially linear models and nonlinear identification”. In: *IEEE Trans. Autom. Control* 50.10 (2005), pp. 1602–1606.

[11] J. Gehring, M. Auli, D. Grangier, and Y. Dauphin. “A Convolutional Encoder Model for Neural Machine Translation”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2017, pp. 123–135.

[12] F. Giri and E.-W. Bai, eds. *Block-oriented Nonlinear System Identification*. Springer London, 2010.

[13] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. “Maxout Networks”. In: *Proceedings of the 30th International Conference on Machine Learning*. 2013, pp. 1319–1327.

[14] C. Gulcehre, K. Cho, R. Pascanu, and Y. Bengio. “Learned-Norm Pooling for Deep Feedforward and Recurrent Neural Networks”. In: 2014.

[15] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. arXiv: `1512.03385`.

[16] K. He, X. Zhang, S. Ren, and J. Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification”. In: *Proc. IEEE Int. Conf. Computer Vision*. 2015, pp. 1026–1034.

[17] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups”. In: *IEEE Signal Process. Mag.* 29.6 (2012), pp. 82–97.

[18] H. Hjalmarsson and J. Schoukens. “On Direct Identification of Physical Parameters in Non-Linear Models”. In: *IFAC Proceedings Volumes* 37.13 (2004), pp. 375–380.

[19] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* (1997).

[20] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366.

[21] S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015, pp. 448–456.

[22] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu. “Neural Machine Translation in Linear Time”. en. In: *arXiv:1610.10099 [cs]* (2016). arXiv: `1610.10099 [cs]`.

[23] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *Proceedings of the 3rd International Conference for Learning Representations (ICLR)*. 2014. arXiv: `1412.6980`.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems NIPS*. 2012.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[26] R. Kumar, S. Srivastava, J. R. P. Gupta, and A. Mohindru. “Comparative Study of Neural Networks for Dynamic Nonlinear Systems Identification”. In: *Soft Computing* 23.1 (2019), pp. 101–114.

[27] L. Ljung, Q. Zhang, P. Lindskog, and A. Juditski. “Estimation of grey box and black box models for non-linear circuit data”. In: *IFAC Proceedings Volumes* 37.13 (2004), pp. 399–404.

[28] J. Long, E. Shelhamer, and T. Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition*. 2015, pp. 3431–3440.

[29] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton. “On Rectified Linear Units for Speech Processing”. In: *2013 IEEE Int. Conf. Acoustics, Speech and Signal Processing*. 2013, pp. 3517–3521.

[30] A. L. Maas, A. Y. Hannun, and A. Y. Ng. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: *In ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[31] A. Marconato, J. Sjöberg, J. Suykens, and J. Schoukens. “Identification of the Silverbox Benchmark Using Nonlinear State-Space Models”. In: *IFAC Proceedings Volumes* 45.16 (2012), pp. 632–637.

[32] W. S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[33] T. Mikolov, K. Chen, G. Corrado, and J. Dean. *Efficient estimation of word representations in vector space.* Tech. rep. arXiv:1301.3781, 2013.

[34] K. S. Narendra and K. Parthasarathy. “Identification and Control of Dynamical Systems Using Neural Networks”. In: *IEEE Trans. Neural Netw.* 1.1 (1990), pp. 4–27.

[35] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. “WaveNet: A Generative Model for Raw Audio”. In: *arXiv:1609.03499 [cs]* (2016). arXiv: `1609.03499 [cs]`.

[36] J. Paduart, G. Horváth, and J. Schoukens. “Fast identification of systems with nonlinear feedback”. In: *IFAC Proceedings Volumes* 37.13 (2004), pp. 381–385.

[37] J. Paduart. “Identification of Nonlinear Systems using Polynomial Nonlinear State Space Models”. PhD thesis. Vrije Universiteit Brussel, 2008.

[38] G. Palm. “On representation and approximation of nonlinear systems”. In: *Biological Cybernetics* 34.1 (1979), pp. 49–52.

[39] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. “Automatic Differentiation in PyTorch”. In: (2017).

[40] E. Pepona, S. Paoletti, A. Garulli, and P. Date. “Identification of Piecewise Affine LFR Models of Interconnected Systems”. In: *IEEE Trans. Control Syst. Technol.* 19.1 (2011), pp. 148–155.

[41] R. Pintelon and J. Schoukens. *System identification: A frequency domain approach.* 2nd ed. Wiley-IEEE Press, 2012.

[42] N. Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1 (1999), pp. 145–151.

[43] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition.* 2016, pp. 779–788.

[44] F. Sabahi and M. R. Akbarzadeh-T. “Extended Fuzzy Logic: Sets and Systems”. In: *IEEE Trans. Fuzzy Syst.* 24.3 (2016), pp. 530–543.

[45] T. Salimans and D. P. Kingma. *Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks.* 2016.

[46] M. Schetzen. *The Volterra & Wiener Theories of Nonlinear Systems.* Krieger Publishing Company, 2006.

[47] M. Schoukens and J.-P. Noël. “F-16 aircraft benchmark based on ground vibration test data”. In: *Workshop on Nonlinear System Identification Benchmarks.* 2017.

[48] M. Schoukens, A. Marconato, R. Pintelon, G. Vandersteen, and Y. Rolain. “Parametric Identification of Parallel Wiener-Hammerstein Systems”. In: *Automatica* 51 (2015), pp. 111–122. arXiv: `1708.06543`.

[49] M. Schoukens and K. Tiels. “Identification of block-oriented nonlinear systems starting from linear approximations: A survey”. In: *Automatica* 85 (2017), pp. 272–292.

[50] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv:1409.1556 [cs]* (2014). arXiv: `1409.1556 [cs]`.

[51] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky. “Nonlinear black-box modeling in system identification: a unified overview”. In: *Automatica* (1995).

[52] L. Sragner, J. Schoukens, and G. Horváth. “Modelling of a slightly nonlinear system: a neural network approach”. In: *IFAC Proceedings Volumes* 37.13 (2004), pp. 387–392.

[53] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[54] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going Deeper with Convolutions”. In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition.* 2015, pp. 1–9.

[55] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude.* COURSERA: Neural Networks for Machine Learning. 2012.

[56] K. Tiels. “Polynomial nonlinear state-space modeling of the F-16 aircraft benchmark”. In: *Workshop on Nonlinear System Identification Benchmarks.* 2017.

[57] K. Tiels. “Wiener system identification with generalized orthonormal basis functions”. PhD thesis. Vrije Universiteit Brussel, 2015.

[58] A. Van Mulders, J. Schoukens, and L. Vanbeylen. “Identification of systems with localised nonlinearity: From state-space to block-structured models”. In: *Automatica* 49.5 (2013), pp. 1392–1396.

[59] J. A. Vargas, W. Pedrycz, and E. M. Hemerly. “Improved Learning Algorithm for Two-Layer Neural Networks for Identification of Nonlinear Systems”. In: *Neurocomputing* 329 (2019), pp. 86–96.

[60] V. Verdult. “Identification of Local Linear State-Space Models: The Silver-Box Case Study”. In: *IFAC Proceedings Volumes* 37.13 (2004), pp. 393–398.

[61] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. “Phoneme recognition using time-delay neural networks”. In: *IEEE Trans. Acoust., Speech, Signal Process.* 37.3 (1989), pp. 328–339.

[62] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. “Regularization of Neural Networks Using DropConnect”. In: *Proceedings of the 30th International Conference on Machine Learning.* 2013, pp. 1058–1066.

[63] T. Wigren and J. Schoukens. “Three free data sets for development and benchmarking in nonlinear system identification”. In: *2013 European Control Conference (ECC).* 2013.

[64] A. Wills and B. Ninness. “Generalised Hammerstein–Wiener system estimation and a benchmark application”. In: *Control Engineering Practice* 20.11 (2012), pp. 1097–1108.

[65] J. Wray and G. G. R. Green. “Calculation of the Volterra kernels of non-linear dynamic systems using an artificial neural network”. In: *Biological Cybernetics* 71.3 (1994), pp. 187–195.

[66] X. Qian, H. Huang, X. Chen, and T. Huang. “Generalized Hybrid Constructive Learning Algorithm for Multioutput RBF Networks”. In: *IEEE Trans. Cybern.* 47.11 (2017), pp. 3634–3648.

[67] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-Based Learning Applied to Document Recognition”. In: *Proc. IEEE* 86.11 (1998), pp. 2278–2324.

Paper III

Title

Evaluating model calibration in classification

Authors

Juozas Vaicenavicius, David Widmann, Carl Andersson, Fredrik Lindsten, Jacob Roll, and Thomas B. Schön

Edited version of

J. Vaicenavicius, D. Widmann, C. Andersson, F. Lindsten, J. Roll, and T. B. Schön. “Evaluating model calibration in classification”. In: *Proceedings of AISTATS*. PMLR, 2019

Evaluating model calibration in classification

Abstract

Probabilistic classifiers output a probability distribution on target classes rather than just a class prediction. Besides providing a clear separation of prediction and decision making, the main advantage of probabilistic models is their ability to represent uncertainty about predictions. In safety-critical applications, it is pivotal for a model to possess an adequate sense of uncertainty, which for probabilistic classifiers translates into outputting probability distributions that are consistent with the empirical frequencies observed from realized outcomes. A classifier with such property is called *calibrated.* In this work, we develop a general theoretical calibration evaluation framework grounded in probability theory, and point out subtleties present in model calibration evaluation that lead to refined interpretations of existing evaluation techniques. Lastly, we propose new ways to quantify and visualize miscalibration in probabilistic classification, including novel multidimensional reliability diagrams.

1 Introduction

Understanding whether the predictions of a classification model are trustworthy is of crucial importance in machine learning applications. Many popular classifiers such as neural networks output a real-valued vector whose values are typically used to choose the most likely class or even rank the likeliness of different classes. Hence, the key question in the evaluation of such classifiers is whether the classifier output can be interpreted as real-world probabilities, which are what matters for decision making in reality. As much of machine learning research and applications concern building models with good predictive performance, the question of how well the confidence score (expressed as a number between 0 and 1) of the predicted class is calibrated has been dominating the model evaluation literature [6, 11, 18]. Put differently, it is whether the confidence score of the predicted class can be interpreted as the probability of the classifier getting the class right—a natural question in many applications.

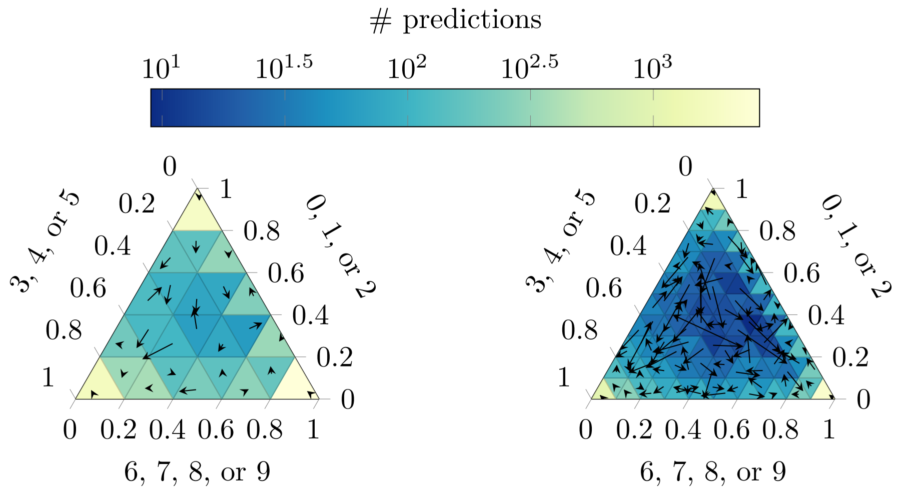


Figure 1: Two-dimensional reliability diagrams for LeNet on the CIFAR-10 test set with 25 and 100 bins of equal size. The predictions are grouped into three groups $\{0, 1, 2\}$, $\{3, 4, 5\}$, and $\{6, 7, 8, 9\}$ of the original classes. Arrows represent the deviation of the estimated calibration function value (arrow head) from the group prediction average (arrow tail) in a bin. The empirical distribution of predictions is visualized by color-coding the bins.

However, in a number of new, especially safety-critical, applications of machine learning it is of increasing importance to know whether the entire classifier output can be interpreted probabilistically, not just the confidence of the predicted class; this is the main question to be addressed in this paper. We illustrate the need for a probabilistic interpretation of the entire classifier output via a simplistic yet illustrative example. Suppose we have a classification problem in which given an image containing either a single or no living entity a classifier outputs a probability vector expressing the likeliness of three different classes "`no creature`", "`person`", and "`animal`". Suppose that for the same image the outputs of two different classifiers are ($\texttt{no creature} = 0.9, \texttt{person} = 0.1, \texttt{animal} = 0$) and ($\texttt{no creature} = 0.9, \texttt{person} = 0, \texttt{animal} = 0.1$). Since all widely adopted calibration evaluation techniques take into account only the score of the predicted class, the two mentioned classifier outputs are the same, as far as evaluation is concerned. However, if these outputs actually corresponded to real-world probabilities, the control actions based on these vectors might be very different, e.g., for an autonomous vehicle, due to behavioral differences between pedestrians and animals or differing driving norms in proximity to objects of the distinct classes. This simple example motivates the significance of our quest for calibration evaluation of probabilistic multiclass classifiers beyond just the confidence score of the predicted class.

In this paper, we develop and motivate a general mathematical framework grounded in probability theory for evaluating probabilistic multiclass classifiers. Within it, the existing calibration evaluation techniques can be seen as special cases. The theoretical construct allows for rigorous in-depth analysis of chosen aspects of model calibration and provides clear statistical interpretation of results. Moreover, it unlocks new and sheds light on existing calibration evaluation methods. In particular, we build up theoretical understanding of how popular calibration evaluation techniques should be reinterpreted to avoid jumping to attractive though unjustified conclusions with possibly undesirable practical consequences. Furthermore, drawing inspiration from the statistics literature we revisit classical reliability diagrams used for calibration evaluation and put forward our variant that conveniently summarizes the relevant calibration information in an interpretable way. Addressing the multidimensional nature of calibration evaluation in a multiclass setting we propose novel multidimensional reliability diagrams that are capable of representing the full calibration evaluation information for classification problems with three and four classes. The proposed calibration evaluation methods are illustrated on standard neural network classifiers.[1]

In the NeurIPS 2017 spotlight paper by Lakshminarayanan et al. [12], the authors conclude with “We hope that our work will encourage the community to consider non-Bayesian approaches (such as ensembles) and other interesting evaluation metrics for predictive uncertainty”. Sharing the same interest in predictive uncertainty evaluation, we respond to the call with this article.

2 Problem formulation

In this paper, we study a classical supervised classification problem. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space on which we have independent and identically distributed random variable pairs (X, Y), (X_1, Y_1), ..., $(X_n, Y_n) \in \mathcal{X} \times \mathcal{Y}$ having the same joint distribution $p(X, Y)$. Here $\mathcal{X}$ denotes the input space while $\mathcal{Y}$ denotes the finite set of m available class labels. Given a training data set of realizations $\{(x_i, y_i)\}_{i=1}^n$ of $\{(X_i, Y_i)\}_{i=1}^n$, ideally we would like to find the optimal probabilistic classifier $g^{\mathrm{opt}} := p(Y| \, X = \cdot)$; here the popular

[1]Accompanying code is provided at `https://github.com/uu-sml/calibration`.

"·" notation acts as a placeholder for the argument of a function.

Unfortunately, it is generally impossible to recover the optimal classifier given only a data sample of finite size. Acknowledging this inexorable limitation we can only hope to find a measurable model $g\colon \mathcal{X} \to \mathcal{P}(\mathcal{Y})$ with $g \approx g^{\mathrm{opt}}$ that best suits our purpose. Here $\mathcal{P}(\mathcal{Y})$ denotes the space of probability measures on $\mathcal{Y}$. Since $\mathcal{Y}$ is a finite discrete set, we can identify the space $\mathcal{P}(\mathcal{Y})$ with the $(m-1)$-dimensional probability simplex $\Delta_{m-1} := \{x \in [0,1]^m \colon \|x\|_1 = 1\}$. As it is often more convenient to work with vectors in Δ_{m-1} than with elements in $\mathcal{P}(\mathcal{Y})$, we will use both spaces interchangeably.

In safety-critical applications, it is of crucial importance that the predictive distribution of a classification model honestly expresses its true predictive uncertainty. In statistical terminology, such honest classifiers are called *reliable* or *calibrated* [see, e.g., 1, 24]. Mathematically the reliability condition reads as

$$\mathbb{P}[Y \in \cdot \,|\, g(X)] = g(X), \tag{III.1}$$

i.e., the distribution of the target class conditional on any given prediction of our model is exactly equal to that prediction. For instance, the prediction $(0.95, 0.05)$ of a reliable binary classifier implies that the probability of the target being in the first class is 0.95 and being in the second class is 0.05, when considering all inputs resulting in the classifier output $(0.95, 0.05)$. Interestingly, within the machine learning community [6] a probabilistic model is called perfectly calibrated even if the much weaker condition

$$\mathbb{P}[Y = \arg\max\, g(X) \,|\, \max g(X)] = \max g(X) \tag{III.2}$$

is satisfied, possibly giving rise to some confusion. As discussed by Zadrozny et al. [24], in practice it is also desired that

$$\mathbb{P}[Y = y \,|\, g(X)(\{y\})] = g(X)(\{y\}) \tag{III.3}$$

for all $y \in \mathcal{Y}$, i.e., that the marginal predictions of a model are calibrated. The conditions in Equations (III.1) to (III.3) are equivalent for binary classification problems but *not* in a general classification setting as the following example shows.

Example 1 (A motivating toy example)**.** Let us take a look at a simple example in which perfect calibration according to Guo et al. [6] and calibrated marginal predictions do not imply reliability. Suppose $\mathcal{Y} = \{1, 2, 3\}$. Let g be a probabilistic classifier that predicts the six probability distributions in the first column of Table 1 with equal probability and assume that the true conditional distribution $\mathbb{P}[Y \in \cdot \,|\, g(X)]$ is given by the second column. Figure 2 depicts these quantities on the probability simplex. The model is perfectly calibrated according to Guo et al. [6] and additionally all marginal predictions are calibrated. However, g is not reliable since Equation (III.1) is not satisfied. Taking a safety-critical viewpoint and viewing model reliability as being of crucial importance, in this paper we set out to develop a comprehensive calibration evaluation methodology, a goal that will also require more future research to reach. $\square$

$g(X)$	$\mathbb{P}[Y \in \cdot \mid g(X)]$
(0.1, 0.3, 0.6)	(0.2, 0.2, 0.6)
(0.1, 0.6, 0.3)	(0.0, 0.7, 0.3)
(0.3, 0.1, 0.6)	(0.2, 0.2, 0.6)
(0.3, 0.6, 0.1)	(0.4, 0.5, 0.1)
(0.6, 0.1, 0.3)	(0.7, 0.0, 0.3)
(0.6, 0.3, 0.1)	(0.5, 0.4, 0.1)

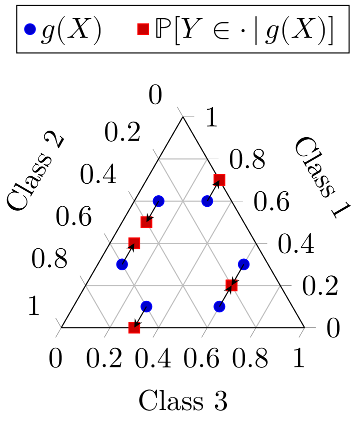


Figure 2: Probabilistic classifier g for $\mathcal{Y} = \{1, 2, 3\}$ with six uniformly distributed predictions.

Structure of the paper In Appendix 3, we present a general theoretical calibration evaluation framework allowing for investigation of arbitrary aspects of model calibration. Appendix 4 is dedicated to empirical evaluation of model calibration. It includes a study of estimators of key quantities, explanations of inherent subtleties of practical importance, refined interpretations of existing calibration evaluation techniques, and newly proposed calibration evaluation tools such as hypotheses tests and multidimensional reliability diagrams shown in Figure 1. The proposed evaluation methods are illustrated on standard neural network classifiers as well as on a tractable Gaussian mixture model in Appendix 5. The main text is supported by closely related appendices, available in the supplementary material, in which proofs, additional examples, explanations, and numerical results are included.

2.1 Related work

Practical application of calibrated classification models requires the knowledge of at least two crucial things: 1) how to evaluate the calibration of a given model, and 2) how to build a candidate model.

Predictive uncertainty evaluation Calibration evaluation of neural networks in classification tasks has been a research topic for over a decade and has witnessed a resurgence of interest over the last two years. Niculescu-Mizil et al. [18] used classical reliability diagrams [15, 16] to investigate calibration of shallow neural networks in a binary classification setting and concluded that these networks are well-calibrated. Recently Guo et al. [6] showed empirically that modern neural networks are ill-calibrated and ignited a strand of research [9, 11, 21, 25] proposing methods for obtaining calibrated classifiers. In all these works, calibration of the classifier is judged based on classical reliability diagrams and the so-called expected calibration error (ECE) [6] whose real-valued estimate is calculated from a reliability diagram and its associated confidence histogram. As we explain in this paper, calibration evaluation using this popular methodology suffers from certain shortcomings that we address in this work.

Another recent paper by Lakshminarayanan et al. [12] argues that predictive uncertainty calibration should be evaluated in terms of empirical approximations of expected scores based on strictly proper scoring rules [5]. However, expected scores do not quantify model calibration directly; in fact, a model that is *not* close to the true classifier with respect to the expected score can nevertheless be calibrated. Also, different scoring rules can rank the same models differently [14, 22].

Building calibrated probabilistic classifiers There exist two general approaches to obtain calibrated classifiers. The first one encourages model calibration already during training [9, 11, 12, 21, 25, see, e.g.,]. An alternative approach is to recalibrate an existing ill-calibrated model; for binary classification this post-processing was applied by Platt [20] and Zadrozny et al. [23], and for multiclass classification by Guo et al. [6] and Zadrozny et al. [24]. A quality comparison of different approaches calls for a sound calibration evaluation methodology. Moreover, reliability diagrams and expected miscalibration estimators can also be utilized as recalibration devices, similar to the work of Platt [20] and Zadrozny et al. [23], and hence the developments in the present paper are also of relevance for recalibration of classifiers.

3 Theoretical framework

Before delving into calibration evaluation questions let us point out that for a given classification problem there are typically infinitely many calibrated models. Hence, there is hope to find a calibrated classifier (or at least one close to being calibrated) even though the ideal model g^{opt} is almost impossible to find.

Proposition 1 (Many calibrated classifiers). *For any measurable function $h\colon \mathcal{X} \to \mathcal{Z}$, where $\mathcal{Z}$ is some measurable space, the map g defined by*

$$g(X) := \mathbb{P}[Y \in \cdot \,|\, h(X)]$$

is a calibrated probabilistic classifier.

A proof of this statement is provided in Appendix 7.2. In particular, if h is a bijection, then g is the ideal model g^{opt}. At the other extreme, if h is a constant function, then g is a calibrated but quite uninformative classifier that always predicts the marginal distribution of Y regardless of the input.

3.1 Calibration functions

All information about the calibration of a probabilistic classifier g is contained in its *(canonical) calibration function* r given by

$$r(\mu) := \mathbb{P}[Y \in \cdot \,|\, g(X) = \mu] \tag{III.4}$$

for every $\mu \in \mathcal{P}(\mathcal{Y})$ for which it is well-defined. By the definition of reliability in Equation (III.1), a model is reliable if and only if its calibration function is the identity function. Furthermore, deviations from the identity function capture the degree of miscalibration present at different predictions. In many practical situations, especially when the number of target classes is large, we will typically not have enough data to accurately estimate the calibration function. Nonetheless, with the data available we might still be able to investigate certain aspects of model calibration; we next turn to formalize this idea of partial calibration evaluation.

The main principle underlying partial calibration evaluation is that a calibrated classifier induces calibrated classifiers for induced problems. Let $\psi\colon \mathcal{Y} \times \mathcal{P}(\mathcal{Y}) \to \mathcal{Z}$ be a measurable function that maps a pair of an observation in $\mathcal{Y}$ and a probabilistic prediction on $\mathcal{Y}$ to an element in a measurable space $\mathcal{Z}$ which is possibly different from $\mathcal{Y}$. Then ψ induces a function $\pi_\psi\colon \mathcal{P}(\mathcal{Y}) \to \mathcal{P}(\mathcal{Z})$ given by

$$\pi_\psi(\mu) := \mathbb{P}_{Y' \sim \mu}[\psi(Y', \mu) \in \cdot], \quad \mu \in \mathcal{P}(\mathcal{Y}),$$

which maps a probabilistic prediction on $\mathcal{Y}$ to a prediction on $\mathcal{Z}$. Thus the model g and the function ψ yield an *induced predictive model* $g_\psi := \pi_\psi \circ g$ for an *induced problem*. In this induced problem, the random variable pairs (X_1, Z_1), ..., $(X_n, Z_n) \in \mathcal{X} \times \mathcal{Z}$, where $Z_i := \psi(Y_i, g(X_i))$, correspond to the observable data, and finding a probabilistic prediction of $Z = \psi(Y, g(X))$ given X is of interest. Instead of the calibration function of g in Equation (III.4), we can consider the calibration function of g_ψ, which we also call the *calibration function* $r_\psi \colon \mathcal{P}(\mathcal{Z}) \to \mathcal{P}(\mathcal{Z})$ *induced by* ψ. It is given by

$$\begin{aligned} r_\psi(\nu) &:= \mathbb{P}[Z \in \cdot \mid g_\psi(X) = \nu] \\ &= \mathbb{P}[\psi(Y, g(X)) \in \cdot \mid \pi_\psi(g(X)) = \nu] \end{aligned}$$

for every $\nu \in \mathcal{P}(\mathcal{Z})$ for which it is well-defined. Informally, each function ψ provides a lens through which we can inspect a particular aspect of model calibration. If a model is calibrated, then every induced calibration function is equal to the identity function. However, even if an induced calibration function is equal to the identity function, a model does not have to be calibrated, as can be seen from Example 1 together with Example 1 below. We now look at a few natural choices of the *calibration lens* ψ. Specifically, any popular calibration evaluation performed in machine learning literature can be phrased in this general setup.

Example 1 (Calibration functions). (i) *The canonical calibration function.* The calibration function induced by $\psi \colon (y, \mu) \mapsto y$ is just the canonical calibration function.

(ii) *The fixed partition.* Fix a partition $\{A_i\}_{i=1}^k$ of $\mathcal{Y}$. Then $\psi \colon (y, \mu) \mapsto \sum_{i=1}^k i \mathbb{1}_{A_i}(y)$ induces a probabilistic classifier for a classification problem with target classes $A_1, \ldots, A_k$ and a calibration function that captures its reliability.

(iii) *The k largest predictions.* Define $\alpha \colon \{1, \ldots, m\} \times \mathcal{P}(\mathcal{Y}) \to \mathcal{Y}$ such that $\alpha(i, \mu)$ is the class of the ith largest prediction according to μ. Let $1 \leq k < m$ and define $\psi_k(y, \mu) := \sum_{i=1}^k i \mathbb{1}_{\{\alpha(i,\mu)\}}(y)$, which induces a calibration function capturing how well the k largest predictions are calibrated. The perfect calibration definition by Guo et al. [6] corresponds to the special case $k = 1$.

□

Since $\mathcal{Y}$ is a finite discrete space in classification problems, it is sufficient to consider only finite spaces $\mathcal{Z}$ with $|\mathcal{Z}| \leq m$. Consequently, every result about canonical calibration functions generalizes immediately to induced calibration functions by applying the results to the induced models. Thus without loss of generality we only provide results for canonical calibration functions.

3.2 Measures of miscalibration

Often it is desirable to summarize the overall calibration performance in a single number. This can be achieved with the help of a distance function

$d \colon \Delta_{m-1} \times \Delta_{m-1} \to [0, \infty)$ gauging the closeness between the output of a probabilistic classifier and the corresponding value of the calibration function.

One natural quantity of interest is the *expected miscalibration on $A \subseteq \Delta_{m-1}$ with respect to d* defined by

$$\eta_d := \mathbb{E}[d(r(g(X)), g(X)) \,|\, g(X) \in A]. \tag{III.5}$$

Another interesting metric is the worst case miscalibration $\max_{\mu \in A} d(r(\mu), \mu)$. Choosing $A = \Delta_{m-1}$ allows us to quantify miscalibration on the entire probability simplex whereas by selecting $A \subsetneq \Delta_{m-1}$ we can focus on miscalibration in a particular region of interest. For the sake of brevity we will omit the dependency on the subset A in our notation when no confusion arises.

In the language of our framework, Guo et al. [6] used the popular total variation (TV) distance $d(x, y) = \frac{1}{2}\|x - y\|_1$ to measure miscalibration of an induced binary probabilistic classifier. The squared Euclidean distance $d(x, y) = \|x - y\|_2^2$ is another easily interpretable distance, though any distance function can be chosen to suit the application.

4 Empirical calibration evaluation

Having introduced the theoretical framework, we next turn to empirical evaluation questions.

4.1 Estimators and their properties

Calibration functions Let $\Phi = \{\Phi^{(i)}\}_{i=1}^{L}$ be a random partition of $A \subseteq \Delta_{m-1}$ with the randomness arising solely from the dependence on the observable data $\{(g(X_i), Y_i)\}_{i=1}^{n}$. We denote the unique set in this partition containing a vector $w \in A$ by $\Phi[w]$.

The histogram regression estimator $\hat{r} \colon A \to \mathcal{P}(\mathcal{Y}) \cup \{0\}$ of the calibration function on A is defined by

$$\hat{r}(w)(\{y\}) := \frac{|\{i \colon g(X_i) \in \Phi[w] \wedge Y_i = y\}|}{|\{i \colon g(X_i) \in \Phi[w]\}|}$$

for all $w \in A$ and $y \in \mathcal{Y}$, with the convention that $\hat{r}(w) = 0$ if the denominator happens to be zero, i.e., if there are no data points in $\Phi[w]$. The estimate $\hat{r}$ is constant on every set in the partition Φ, and hence we can define $\hat{r}^{(i)}$ as its unique value on $\Phi^{(i)}$.

For sensible partitioning schemes that result in finer and finer partitions as the data set grows the estimate $\hat{r}$ converges to the calibration function in an appropriate sense as shown by Nobel [19]. Splitting the probability simplex into bins of equal size (as often done in the machine learning literature for binary problems) is the simplest option. However, choosing a data-dependent binning scheme is known to provide much faster convergence in practice [19].

Expected miscalibration Let us denote the proportion and the average of all predictions in a subset $\Phi^{(i)}$ by

$$\hat{p}^{(i)} := \frac{|\{j \colon g(X_j) \in \Phi^{(i)}\}|}{|\{j \colon g(X_j) \in A\}|} \text{ and } \hat{g}^{(i)} := \frac{\sum_{j \colon g(X_j) \in \Phi^{(i)}} g(X_j)}{|\{j \colon g(X_j) \in \Phi^{(i)}\}|},$$

respectively. Similarly as before, we adopt the convention that $\hat{p}^{(i)} = 0$ and $\hat{g}^{(i)} = 0$ if the denominators happen to be zero.

The next result tells us that

$$\hat{\eta}_d := \sum_{i=1}^{L} \hat{p}^{(i)} d(\hat{r}^{(i)}, \hat{g}^{(i)}) \tag{III.6}$$

can be interpreted as an estimator of the expected miscalibration η_d. Guo et al. [6] used a special case of this estimator to estimate the expected miscalibration of an induced binary classifier with respect to the total variation distance.

Theorem 1. *Suppose that the calibration function r is Lipschitz continuous and $d \colon \Delta_{m-1} \times \Delta_{m-1} \to [0, \infty)$ is continuous and uniformly continuous in the first argument. Let $\{\Phi_N\}_{N \in \mathbb{N}}$ be a sequence of finite data-independent partitions of $A \subseteq \Delta_{m-1}$ such that $\lim_{N \to \infty} \max_{S \in \Phi_N} \operatorname{diam} S = 0$, where $\operatorname{diam} S := \sup_{x,y \in S} \|x - y\|_2$. Then*

$$\lim_{N \to \infty} \lim_{n \to \infty} \hat{\eta}_{d,N} = \eta_d,$$

with limits in the almost sure sense, where estimator $\hat{\eta}_{d,N}$ is defined according to Equation (III.6) *for each $N \in \mathbb{N}$.*

Interestingly, the estimator in Equation (III.6) yields a lower bound for the expected miscalibration if the partition is kept fixed as the size of the data set grows to infinity.

Theorem 2. *Let $d \colon \Delta_{m-1} \times \Delta_{m-1} \to [0, \infty)$ be a continuous convex function and let Φ be a finite data-independent partition of $A \subseteq \Delta_{m-1}$. Then*

$$\lim_{n \to \infty} \hat{\eta}_d \leq \eta_d, \tag{III.7}$$

with limit in the almost sure sense. Moreover, if d can be written as $d(p, p') = f(p - p')$ for a convex function f, then equality holds if and only if for every $\Phi^{(i)}$ there exists a set $S \subseteq \mathbb{R}^m$ such that $\mathbb{P}[r(g(X)) - g(X) \in S \,|\, g(X) \in \Phi^{(i)}] = 1$ and f coincides almost surely with an affine function on the convex hull of S.

Proofs of Theorems 1 and 2 are given in Appendix 7.2. Note that the popular distances arising from $\|\cdot\|_1$ and $\|\cdot\|_2^2$ satisfy the convexity condition in Theorem 2.

4.2 Quantifying the estimator variability

In the previous section, we discussed estimation of the calibration function and the expected miscalibration. Abstractly, both can be viewed as a realization of a random variable of the form $h(D)$, where $D := \{(g(X_i), Y_i)\}_{i=1}^n$ is random and h is a deterministic function. Thus to avoid being fooled by randomness, the distribution of the calculated statistic must be taken into account when drawing any conclusions.

Consistency resampling The randomness of the estimator $h(D)$ can be attributed to the randomness of $g(X_1)$, ..., $g(X_n)$ and the randomness of $Y_1 \,|\, g(X_1)$, ..., $Y_n \,|\, g(X_n)$. The variability of $h(D)$ due to the randomness of $g(X_i)$ can be estimated by bootstrapping new data sets s^1, s^2, ..., where $s^j = \{g_i^j\}_{i=1}^n$, from the predictions $\{g(x_i)\}_{i=1}^n$ [see 4]. Moreover, assuming that the true calibration function r equals some known function ρ (in the following we indicate this assumption with a superscript ρ), we can approximate the variability of the estimator $h(D^\rho)$ due to the randomness arising from $Y_i^\rho \,|\, g(X_i)$ by sampling values $y_1^{\rho,j}$, ..., $y_n^{\rho,j}$ from the distributions $\rho(g_1^j)$, ..., $\rho(g_n^j)$, respectively. The described resampling procedure is referred to as *consistency resampling* [3]. Given a big enough data set, we can also perform the usual bootstrapping from the complete data set $\underline{D} := \{(g(x_i), y_i)\}_{i=1}^n$ to estimate the variability of $h(D)$.

4.3 Interpreting empirical calibration evaluation

Testing a hypothesis of perfect calibration The approximate distribution of $h(D^\rho)$ obtained via consistency resampling can be used to perform null hypothesis tests. A natural choice is to check whether the calibration function equals the identity function, i.e., $\rho = \mathrm{id}$, as is the case for calibrated models. If h is a real-valued function we can estimate $\mathbb{P}[h(D^{\mathrm{id}}) \geq h(\underline{D})]$ and use this value as a p-value to reject the hypothesis of perfect calibration.

Comparing miscalibration estimates Suppose we want to compare two probabilistic classifiers g and g' in terms of their reliability. Let $\widehat{\eta}_d$ and $\widehat{\eta}_d'$ be the estimators of η_d and η_d', respectively. It is currently common practice [see, e.g., 6, 11, 17] to compare the reliability of g and g' in terms of just the realized values $\underline{\widehat{\eta}}_d$ and $\underline{\widehat{\eta}}_d'$ of $\widehat{\eta}_d$ and $\widehat{\eta}_d'$, respectively. Alas, we do not know anything about how much the biases $\mathbb{E}[\widehat{\eta}_d] - \eta_d$ and $\mathbb{E}[\widehat{\eta}_d'] - \eta_d'$ differ (it is easy to come up with examples showing that the two biases can differ significantly, see Appendix 7.1). Moreover, the statistics $\widehat{\eta}_d$ and $\widehat{\eta}_d'$ are random variables typically having different distributions. As a result, comparing calibration of two models just in terms of the realizations of these estimators is unjustified and not necessarily meaningful. This reasoning indicates that the current widespread practice does not reveal the full story and leads to certain doubts about the conclusions reached.

Let us see how the direct comparison in terms of the estimates contrasts with our preferred p-value approach. Using the ideas of Appendix 4.2, we can approximate the distributions of $\hat{\eta}_d^{\mathrm{id}}$ and $\hat{\eta}_d'^{\mathrm{id}}$ in order to calculate $\mathbb{P}[\hat{\eta}_d^{\mathrm{id}} \geq \underline{\hat{\eta}}_d]$ and $\mathbb{P}[\hat{\eta}_d'^{\mathrm{id}} \geq \underline{\hat{\eta}}_d']$. It might be that $\underline{\hat{\eta}}_d > \underline{\hat{\eta}}_d'$ but $\mathbb{P}[\hat{\eta}_d^{\mathrm{id}} \geq \underline{\hat{\eta}}_d] > \alpha > \mathbb{P}[\hat{\eta}_d'^{\mathrm{id}} \geq \underline{\hat{\eta}}_d']$, where α is a chosen significance level. Hence, our p-value test would reject the hypothesis of perfect calibration for model g' based on the small estimate $\mathbb{P}[\hat{\eta}_d'^{\mathrm{id}} \geq \underline{\hat{\eta}}_d'] < \alpha$ but not for model g, regardless of the larger estimate $\underline{\hat{\eta}}_d > \underline{\hat{\eta}}_d'$.

Lower bound interpretation The result in Equation (III.7) tells us that $\hat{\eta}_d$ calculated using a fixed partition is prone to underestimating the true expected miscalibration for uncalibrated models as the size of the data set grows to infinity. The condition for strict inequality in Theorem 2 holds for distances arising from strictly convex functions (unless $r(g(X)) - g(X)$ is constant within every bin) as well as for the total variation distance used by Guo et al. [6] (unless $r(g(X)) - g(X)$ belongs to a single orthant within every bin). Hence, the common practice of estimating expected miscalibration with a fixed small number of bins and many data points can lead to reporting better than actual model miscalibration measures and thus might even put the safety of machine learning applications at risk. Rephrased in statistical terminology, the popular estimator in Equation (III.6) for the expected miscalibration is asymptotically inconsistent in many situations with a negative asymptotic bias.

We emphasize, however, that this lower bound interpretation relies on asymptotics in the number of data points with a fixed binning. Establishing rates of convergence for the bias and variance of various binning schemes is an interesting topic for future work, which would enable us to draw more informative conclusions about the risk of underestimating miscalibration.

4.4 Visualizing model miscalibration

Classical reliability diagrams A classical reliability diagram is a method to investigate the miscalibration of a binary probabilistic classifier [see 2, 16, for an early and a more recent account] by plotting for one class the average prediction versus the regression estimate of the calibration function in each bin. In a binary classification setting the reliability diagram contains all information about $\hat{g}^{(i)}$ and $\hat{r}^{(i)}$ for every i and hence can be viewed as depicting estimates for $\mathbb{E}[g(X) \,|\, g(X) \in \Phi^{(i)}]$ and $\mathbb{E}[r(g(X)) \,|\, g(X) \in \Phi^{(i)}]$.

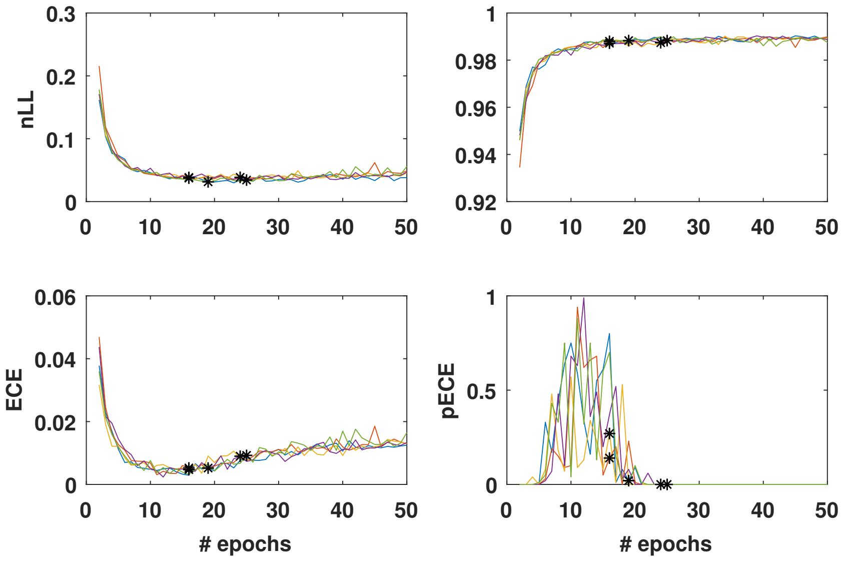


Figure 3: Negative log-likelihood (nLL), accuracy, expected miscalibration (ECE), and p-value under consistency assumption (pECE) of LeNet during training on MNIST (5 different initializations). Asteriks indicate models with minimum negative log-likelihood on the validation data set.

In Figure 4, we present our variant of the classical reliability diagram inspired by Bröcker et al. [3]. We plot the difference between the regression estimate and the average prediction (the so-called deviation) instead of just the regression estimate. In our opinion, this modification simplifies the visual comparison of both estimates while retaining the same information. Additionally consistency bars indicate quantiles of the estimates obtained under the consistency assumption, either by resampling or by making use of the binomial distribution of consistent outcomes as discussed by Bröcker et al. [3]. Thus consistency bars provide a visual indication of the range within which deviations are likely to appear even if the model is calibrated. Moreover, they can be interpreted as a hypothesis test of perfect calibration in each bin. Figure 4a shows equally-sized bins whereas Figure 4b presents the same data using a data-dependent binning scheme that is explained in Appendix 8.

Higher-dimensional reliability diagrams In multiclass classification, a classical one-dimensional reliability diagram contains information only about a specific partial aspect of model reliability as it only represents an induced binary classification problem (recall Example 1 and Appendix 3.1). A direct generalization of our one-dimensional reliability diagram to higher dimensions would be to plot the deviations of the regression estimates from the average predictions for all but one class as arrows pointing from the average predictions to the regression estimates. However, such a visualization would depend on the choice of the left-out class. In our opinion, a better alternative is to plot the deviations of the estimates from the average predictions of all classes as arrows directly on the probability simplex. An estimate of the proportion of predictions falling into a bin can be included in the plot by color-coding the corresponding arrows or bins.

An example of a higher-dimensional reliability diagram is presented in Figure 1. More details such as the variability of the estimates (similar to consistency bars in one-dimensional reliability diagrams) can be added to the diagrams but are not included here to avoid overloaded visualizations. For classification problems with up to four classes, a full multiclass classification reliability diagram can be plotted. If the number of classes is greater than four, we cannot visualize the full probability simplex, but we can still explore particular aspects of model calibration using one-, two-, and three-dimensional reliability diagrams of induced classifiers.

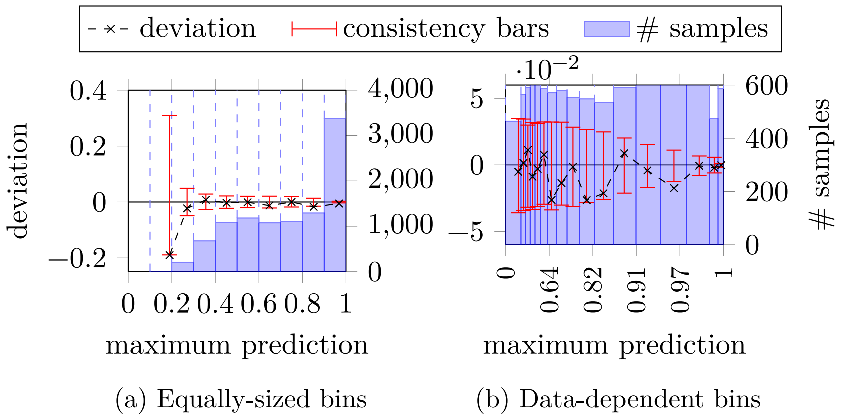


Figure 4: Reliability diagrams for the maximum predictions of LeNet on the CIFAR-10 test set w.r.t. the total variation distance. Crosses indicate the deviation of the outcome distribution from the predictions in each bin. Blue bars show the distribution of predictions. Red bars visualize the 5th and 95th percentiles of the deviation in 1000 consistency resamples.

5 Experimental results

Next, we illustrate the proposed calibration evaluation framework with reliability diagrams by applying it to an analytically tractable example and standard neural networks.

5.1 One-dimensional Gaussian mixture model

We consider a simple Gaussian mixture model consisting of two equally likely standard normally distributed classes with mean -1 and 1. More concretely, we take $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = \{-1, 1\}$, $p(Y = -1) = p(Y = 1) = 1/2$, and $p(X = x \,|\, Y = y) = \mathcal{N}(x; y, 1)$ for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$. This yields $p(Y = -1 \,|\, X = x) = 1/(1 + \exp(2x))$, and thus the perfect model is covered by the class of logistic regression models $g \colon \mathcal{X} \to \Delta_1$ of the form $g_1(x) = 1/(1 + \exp(-(\beta_0 + \beta_1 x)))$, $g_2(x) = 1 - g_1(x)$, with parameters $\beta_0, \beta_1 \in \mathbb{R}$. The only calibrated logistic regression models are the perfect model ($\beta_0 = 0$, $\beta_1 = -2$) and the constant model with parameters $\beta_0 = \beta_1 = 0$ and $g_1(x) = g_2(x) = 1/2$ for all x. In addition to those calibrated models, we consider the uncalibrated logistic regression model with $\beta_0 = \beta_1 = 1$.

The expected miscalibration η_{TV} with respect to the total variation distance is 0 for the calibrated models and approximately 0.56 for the uncalibrated model. For each of these models we repeatedly computed empirical estimates of miscalibration according to Equation (III.6) for varying number of samples and bins. The results are presented in Figures 6, 7, 9, 10, 12 and 13 in Appendix 9. As expected, for the calibrated models the empirical estimates are close to zero but almost always overestimate the expected miscalibration. For the uncalibrated model both over- and underestimation of the expected miscalibration can be observed. It seems that in this example with increasing number of samples and, maybe even more importantly, increasing number of samples per bin, underestimation becomes more likely.

Moreover, we randomly generated 10000 input-output pairs of the Gaussian mixture model and plotted one-dimensional reliability diagrams for the perfect model, the calibrated constant model, and the uncalibrated model in Figures 8, 11 and 14 in Appendix 9, respectively. In addition to the empirical deviation, the distribution of the predictions, and the consistency bars, these plots show the true analytical deviation, which is closely matched by the empirical estimates.

5.2 Neural networks

We trained LeNet [13], DenseNet [8], and ResNet [7] models on the CIFAR-10 data set [10] in the standard way described in the literature. For the LeNet model, one-dimensional reliability diagrams of the maximum predictions are shown in Figure 4, for equally-sized and data-dependent bins. Visually both diagrams cannot rule out the reliability hypothesis for the LeNet model, in accordance with previous publications [6, 18].

Various alternative inspections of reliability are possible with the proposed two-dimensional reliability diagrams such as the plots in Figure 1, for which the original 10 classes of the CIFAR-10 data set are combined into three groups. In these diagrams, the deviation between outcomes and average predictions is small, particularly in regions with frequent predictions, which is again consistent with a calibrated model hypothesis. The one- and two-dimensional reliability diagrams of the DenseNet and ResNet models in Figures 15 to 18 in Appendix 9, and in particular the one-dimensional reliability diagrams with data-dependent bins, do not seem to support the reliability hypothesis for these models to the same extent, in line with previous results by Guo et al. [6].

To observe the proposed p-value test in action, we trained LeNet on the MNIST data set [13] and visualized the p-value evolution as well as other relevant metrics during training in Figure 3. Interestingly, we see that models with the best predictive uncertainty as measured by the negative log-likelihood (a strictly proper scoring rule evaluation advocated by Lakshminarayanan et al. [12]) are quite consistently indicated as miscalibrated by the p-value test. Moreover, in Appendix 10, expected miscalibration estimates are calculated for different neural networks. Our results exhibit the importance of the binning scheme and show that ResNet and DenseNet can have lower expected miscalibration estimates than LeNet, contrasting with the model reliability story told by the reliability diagrams.

6 Conclusion

Evaluation of model calibration is about checking whether probabilities predicted by a model match the distribution of realized outcomes. In this article, we built on existing calibration evaluation approaches and proposed a general mathematical framework for evaluating model calibration, or a chosen aspect of it, in classification problems. We showed that empirical estimates of intuitive miscalibration measures should not be used in a naive way to compare probabilistic classifiers but instead can be employed in hypothesis tests for testing model reliability. We hope our developments and attempts in rigorous model calibration evaluation will encourage other researchers to study this essential topic further.

References

[1] J. Bröcker. “Reliability, Sufficiency, and the Decomposition of Proper Scores”. In: *Quarterly Journal of the Royal Meteorological Society* 135.643 (2009), pp. 1512–1519.

[2] J. Bröcker. “Some Remarks on the Reliability of Categorical Probability Forecasts”. In: *Monthly Weather Review* 136.11 (2008), pp. 4488–4502.

[3] J. Bröcker and L. A. Smith. “Increasing the Reliability of Reliability Diagrams”. In: *Weather and Forecasting* 22.3 (2007), pp. 651–661.

[4] B. Efron and T. Hastie. *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. 1st. Cambridge University Press, 2016.

[5] T. Gneiting and A. E. Raftery. “Strictly Proper Scoring Rules, Prediction, and Estimation”. In: *Journal of the American Statistical Association* 102.477 (2007), pp. 359–378.

[6] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. “On Calibration of Modern Neural Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. 2017, pp. 1321–1330.

[7] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[8] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. "Densely Connected Convolutional Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2261–2269.

[9] A. Kendall and Y. Gal. "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" In: *Advances in Neural Information Processing Systems 30*. 2017, pp. 5574–5584.

[10] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009.

[11] A. Kumar, S. Sarawagi, and U. Jain. "Trainable Calibration Measures for Neural Networks from Kernel Mean Embeddings". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. 2018, pp. 2805–2814.

[12] B. Lakshminarayanan, A. Pritzel, and C. Blundell. "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems 30*. 2017, pp. 6402–6413.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied To Document Recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[14] E. C. Merkle and M. Steyvers. "Choosing a Strictly Proper Scoring Rule". In: *Decision Analysis* 10.4 (2013), pp. 292–304.

[15] A. H. Murphy and R. L. Winkler. "A General Framework for Forecast Verification". In: *Monthly Weather Review* 115.7 (1987), pp. 1330–1338. eprint: `https://doi.org/10.1175/1520-0493(1987)115$<$1330:AGFFFV>2.0.CO;2`.

[16] A. H. Murphy and R. L. Winkler. "Reliability of Subjective Probability Forecasts of Precipitation and Temperature". In: *Applied Statistics* 26.1 (1977), pp. 41–47.

[17] M. P. Naeini, G. Cooper, and M. Hauskrecht. "Obtaining Well Calibrated Probabilities Using Bayesian Binning". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015, pp. 2901–2907.

[18] A. Niculescu-Mizil and R. Caruana. "Predicting Good Probabilities with Supervised Learning". In: *Proceedings of the 22nd International Conference on Machine Learning*. 2005, pp. 625–632.

[19] A. Nobel. "Histogram Regression Estimation Using Data-Dependent Partitions". In: *The Annals of Statistics* 24.3 (1996), pp. 1084–1105.

[20] J. C. Platt. "Probabilities for SV Machines". In: *Advances in Large Margin Classifiers*. Ed. by A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans. MIT Press, 2000, pp. 61–74.

[21] G.-L. Tran, E. V. Bonilla, J. P. Cunningham, P. Michiardi, and M. Filippone. *Calibrating Deep Convolutional Gaussian Processes.* 2018. arXiv: `1805.10522 [stat.ML]`. URL: `http://arxiv.org/abs/1805.10522v1`.

[22] R. L. Winkler and A. H. Murphy. "'Good' Probability Assessors". In: *Journal of Applied Meteorology* 7.5 (1968), pp. 751–758.

[23] B. Zadrozny and C. Elkan. "Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers". In: *Proceedings of the Eighteenth International Conference on Machine Learning.* 2001, pp. 609–616.

[24] B. Zadrozny and C. Elkan. "Transforming Classifier Scores into Accurate Multiclass Probability Estimates". In: *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 2002, pp. 694–699.

[25] G. Zhang, S. Sun, D. Duvenaud, and R. Grosse. "Noisy Natural Gradient as Variational Inference". In: *Proceedings of the 35th International Conference on Machine Learning.* Vol. 80. Proceedings of Machine Learning Research. 2018, pp. 5852–5861.

Supplementary material for 'Evaluating model calibration in classification'

7 Theoretical results on calibration evaluation

7.1 Additional examples

The following example shows that perfect calibration according to Guo et al. [6] does not imply calibrated marginal predictions.

Example 1 (No calibrated marginal predictions). Suppose $\mathcal{Y} = \{1, 2, 3\}$. Let g be a probabilistic classifier that predicts only the two probability distributions in the first column of Table 2 with equal probability and assume that the true conditional distribution $\mathbb{P}[Y \in \cdot \,|\, g(X)]$ is given by the second column. The model is perfectly calibrated according to Guo et al. [6]. However, all marginal predictions are uncalibrated and additionally g is not reliable since Equation (III.1) is not satisfied. Figure 5 provides an illustration of these observations. □

Using a modification of our framework, one can express the calibration evaluation by Kendall et al. [9] by an induced calibration function.

$g(X)$	$\mathbb{P}[Y \in \cdot \,\vert\, g(X)]$
(0.6, 0.1, 0.3)	(0.7, 0.2, 0.1)
(0.4, 0.6, 0.0)	(0.3, 0.5, 0.2)

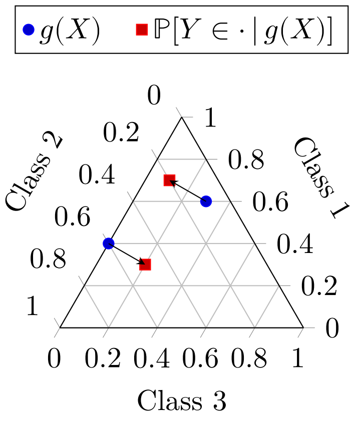


Figure 5: Probabilistic classifier g for $\mathcal{Y} = \{1, 2, 3\}$ with two uniformly distributed predictions.

Example 2 (Marginalized calibration)**.** Kendall et al. [9] evaluated model calibration by estimating the calibration function

$$r_\psi(u) = \mathbb{P}[\psi_u(Y, \mu) \in \cdot \,|\, \pi_{\psi_u}(g(X)) = u\delta_1 + (1-u)\delta_0]$$

induced by $\psi_u(y, \mu) = \mathbb{1}_{\{\mu(\{y\})=u\}}$, where $u \in [0, 1]$ and δ_a denotes the Dirac measure at a. □

The following example shows that the biases of the estimators of expected miscalibration can differ significantly even for calibrated models.

Example 3 (Different bias)**.** Consider a binary classification problem with $\mathcal{Y} = \{1, 2\}$ and $p(Y = 1) = p(Y = 2) = 1/2$ that is clearly separable, i.e., with $p(Y = 0 \,|\, X = x) \in \{0, 1\}$ for all $x \in \mathcal{X}$. Both the optimal model g^{opt} and the constant model $g \equiv (1/2, 1/2)$ are calibrated and hence the expected miscalibration η_{TV} is zero for both models.

Let us define an estimator $\hat{\eta}_{\text{TV}}$ of expected miscalibration with only one bin on the whole probability simplex and a single data point (X, Y). Then the bias of the estimator is $\mathbb{E}[\hat{\eta}_{\text{TV}}] - \eta_{\text{TV}} = 0$ for the perfect model since the expected miscalibration estimate is always zero. However, the constant model yields a bias of $\mathbb{E}[\hat{\eta}_{\text{TV}}] - \eta_{\text{TV}} = 1/2$ since in that case the expected miscalibration estimate is always 1/2. □

7.2 Proofs

Proposition 1 (Many calibrated classifiers). *For any measurable function $h \colon \mathcal{X} \to \mathcal{Z}$, where $\mathcal{Z}$ is some measurable space, the map g defined* *by*

$$g(X) := \mathbb{P}[Y \in \cdot \,|\, h(X)]$$

is a calibrated probabilistic classifier.

Proof. Let $y \in \mathcal{Y}$. We have

$$g(X)(\{y\}) = \mathbb{E}[g(X)(\{y\}) \,|\, g(X)] = \mathbb{E}[\mathbb{E}[\mathbb{1}_{\{y\}}(Y) \,|\, h(X)] \,|\, g(X)].$$

Since by definition $g(X)$ is a function of $h(X)$ it follows from the tower property that

$$g(X)(\{y\}) = \mathbb{E}[\mathbb{1}_{\{y\}}(Y) \,|\, g(X)] = \mathbb{P}[Y = y \,|\, g(X)].$$

Hence $g(X) = \mathbb{P}[Y \in \cdot \,|\, g(X)]$, and therefore the probabilistic classifier g is calibrated. □

Theorem 1. *Suppose that the calibration function r is Lipschitz continuous and $d\colon \Delta_{m-1} \times \Delta_{m-1} \to [0,\infty)$ is continuous and uniformly continuous in the first argument. Let $\{\Phi_N\}_{N\in\mathbb{N}}$ be a sequence of finite data-independent partitions of $A \subseteq \Delta_{m-1}$ such that $\lim_{N\to\infty} \max_{S\in\Phi_N} \operatorname{diam} S = 0$, where $\operatorname{diam} S := \sup_{x,y\in S} \|x-y\|_2$. Then*

$$\lim_{N\to\infty} \lim_{n\to\infty} \widehat{\eta}_{d,N} = \eta_d,$$

with limits in the almost sure sense, where estimator $\widehat{\eta}_{d,N}$ is defined according to Equation (III.6) *for each $N \in \mathbb{N}$.*

Proof. To keep the notation simple we provide a proof for $A = \Delta_{m-1}$. The case $A \subsetneq \Delta_{m-1}$ follows in the same way by conditioning on $g(X) \in A$.

For $N \in \mathbb{N}$ let $\Phi_N = \{\Phi_N^{(i)}\}_{i=1}^{l_N}$ be a finite data-independent partition of Δ_{m-1} such that $\lim_{N\to\infty} \max_i \operatorname{diam} \Phi_N^{(i)} = 0$. We define $\widehat{r}_N^{(i)}$, $\widehat{g}_N^{(i)}$, $\widehat{p}_N^{(i)}$, and $\widehat{\eta}_{d,N}$ analogously to the notation in Appendix 4.1. Similarly we denote the average output distribution, the average predicted distribution, and the proportion of predictions in subset $\Phi_N^{(i)}$ by $\overline{r}_N^{(i)} := \mathbb{E}[r(g(X)) \,|\, g(X) \in \Phi_N^{(i)}]$, $\overline{g}_N^{(i)} := \mathbb{E}[g(X) \,|\, g(X) \in \Phi_N^{(i)}]$, and $\overline{p}_N^{(i)} := \mathbb{P}[g(X) \in \Phi_N^{(i)}]$, respectively.

From the continuous mapping theorem it follows that for all $N \in \mathbb{N}$

$$\begin{aligned} \lim_{n\to\infty} \sum_{i=1}^{l_N} \left| \widehat{p}_N^{(i)} d\left(\widehat{r}_N^{(i)}, \widehat{g}_N^{(i)}\right) - \overline{p}_N^{(i)} d\left(r(\overline{g}_N^{(i)}), \overline{g}_N^{(i)}\right) \right| &= \\ \sum_{i=1}^{l_N} \overline{p}_N^{(i)} \left| d\left(\overline{r}_N^{(i)}, \overline{g}_N^{(i)}\right) - d\left(r(\overline{g}_N^{(i)}), \overline{g}_N^{(i)}\right) \right|&, \end{aligned} \tag{III.8}$$

with limit in the almost sure sense.

Let $K \geq 0$ be a Lipschitz constant for calibration function r. Hence for all $N \in \mathbb{N}$ and $i \in \{1, ..., l_N\}$ we have

$$\begin{aligned}\left\|\overline{r}_N^{(i)} - r(\overline{g}_N^{(i)})\right\|_2 &= \left\|\mathbb{E}\left[r(g(X)) - r(\overline{g}_N^{(i)}) \,|\, g(X) \in \Phi_N^{(i)}\right]\right\|_2 \\ &\leq \mathbb{E}\left[\|r(g(X)) - r(\overline{g}_N^{(i)})\|_2 \,|\, g(X) \in \Phi_N^{(i)}\right] \\ &\leq K\mathbb{E}\left[\|g(X) - \overline{g}_N^{(i)}\|_2 \,|\, g(X) \in \Phi_N^{(i)}\right] \\ &\leq K\mathbb{E}\left[\operatorname{diam}\Phi_N^{(i)} \,|\, g(X) \in \Phi_N^{(i)}\right] = K \operatorname{diam}\Phi_N^{(i)} \\ &\leq K \max_{S \in \Phi_N} \operatorname{diam} S.\end{aligned} \tag{III.9}$$

Let $\epsilon > 0$. Since by assumption distance function d is uniformly continuous in the first argument, there exists $\delta > 0$ such that for all $x, y, z \in \Delta_{m-1}$ with $\|x - y\|_2 < \delta$ inequality $|d(x, z) - d(y, z)| < \epsilon$ holds. From Equation (III.9) and the assumption $\lim_{N\to\infty} \max_{S\in\Phi_N} \operatorname{diam} S = 0$ we know that there exists $N_0 \in \mathbb{N}$ such that for all $N \geq N_0$ and all $i \in \{1, ..., l_N\}$ we have $\|\overline{r}_N^{(i)} - r(\overline{g}_N^{(i)})\|_2 < \delta$. Hence together with Equation (III.8) we obtain for all $N \geq N_0$

$$\lim_{n\to\infty} \sum_{i=1}^{l_N} \left|\hat{p}_N^{(i)} d\left(\hat{r}_N^{(i)}, \hat{g}_N^{(i)}\right) - \overline{p}_N^{(i)} d\left(r(\overline{g}_N^{(i)}), \overline{g}_N^{(i)}\right)\right| < \sum_{i=1}^{l_N} \overline{p}_N^{(i)} \epsilon = \epsilon,$$

with limit in the almost sure sense. Since ϵ was chosen arbitrarily this implies

$$\lim_{N\to\infty} \lim_{n\to\infty} \sum_{i=1}^{l_N} \left|\hat{p}_N^{(i)} d\left(\hat{r}_N^{(i)}, \hat{g}_N^{(i)}\right) - \overline{p}_N^{(i)} d\left(r(\overline{g}_N^{(i)}), \overline{g}_N^{(i)}\right)\right| = 0, \tag{III.10}$$

with limits in the almost sure sense.

For all $N \in \mathbb{N}$ the triangle inequality yields, with limits taken in the almost sure sense,

$$\begin{aligned}\left|\eta_d - \lim_{n\to\infty} \hat{\eta}_{d,N}\right| &= \left|\mathbb{E}\left[d\left(r(g(X)), g(X)\right)\right] - \lim_{n\to\infty} \sum_{i=1}^{l_N} \hat{p}_N^{(i)} d\left(\hat{r}_N^{(i)}, \hat{g}_N^{(i)}\right)\right| \\ &\leq \left|\mathbb{E}\left[d\left(r(g(X)), g(X)\right)\right] - \sum_{i=1}^{l_N} \overline{p}_N^{(i)} d\left(r(\overline{g}_N^{(i)}), \overline{g}_N^{(i)}\right)\right| \\ &\quad + \left|\lim_{n\to\infty} \sum_{i=1}^{l_N} \hat{p}_N^{(i)} d\left(\hat{r}_N^{(i)}, \hat{g}_N^{(i)}\right) - \sum_{i=1}^{l_N} \overline{p}_N^{(i)} d\left(r(\overline{g}_N^{(i)}), \overline{g}_N^{(i)}\right)\right|.\end{aligned} \tag{III.11}$$

From the definition of the Riemann-Stieltjes integral it follows that

$$\lim_{N\to\infty} \left|\mathbb{E}\left[d\left(r(g(X)), g(X)\right)\right] - \sum_{i=1}^{l_N} \overline{p}_N^{(i)} d\left(r(\overline{g}_N^{(i)}), \overline{g}_N^{(i)}\right)\right| = 0, \tag{III.12}$$

and Equation (III.10) implies that

$$\begin{aligned}\lim_{N\to\infty}\left|\lim_{n\to\infty}\sum_{i=1}^{l_N}\widehat{p}_N^{(i)}d\left(\widehat{r}_N^{(i)},\widehat{g}_N^{(i)}\right)-\sum_{i=1}^{l_N}\overline{p}_N^{(i)}d\left(r(\overline{g}_N^{(i)}),\overline{g}_N^{(i)}\right)\right|\\ \leq \lim_{N\to\infty}\lim_{n\to\infty}\sum_{i=1}^{l_N}\left|\widehat{p}_N^{(i)}d\left(\widehat{r}_N^{(i)},\widehat{g}_N^{(i)}\right)-\overline{p}_N^{(i)}d\left(r(\overline{g}_N^{(i)}),\overline{g}_N^{(i)}\right)\right|=0,\quad\text{(III.13)}\end{aligned}$$

with limits in the almost sure sense. Thus all in all, from Equations (III.11) to (III.13) we obtain

$$\left|\eta_d-\lim_{N\to\infty}\lim_{n\to\infty}\widehat{\eta}_{d,N}\right|=\lim_{N\to\infty}\left|\eta_d-\lim_{n\to\infty}\widehat{\eta}_{d,N}\right|\leq 0+0=0,$$

with limits in the almost sure sense. □

Theorem 2. *Let $d\colon \Delta_{m-1}\times\Delta_{m-1}\to[0,\infty)$ be a continuous convex function and let Φ be a finite data-independent partition of $A\subseteq\Delta_{m-1}$. Then*

$$\lim_{n\to\infty}\widehat{\eta}_d\leq\eta_d,\qquad\text{(III.7)}$$

with limit in the almost sure sense. Moreover, if d can be written as $d(p,p')=f(p-p')$ for a convex function f, then equality holds if and only if for every $\Phi^{(i)}$ there exists a set $S\subseteq\mathbb{R}^m$ such that $\mathbb{P}[r(g(X))-g(X)\in S\,|\,g(X)\in\Phi^{(i)}]=1$ and f coincides almost surely with an affine function on the convex hull of S.

Proof. To keep the notation simple we provide a proof for $A=\Delta_{m-1}$. The case $A\subsetneq\Delta_{m-1}$ follows in the same way by conditioning on $g(X)\in A$.

Let $\Phi=\{\Phi^{(i)}\}_{i=1}^{l}$ be a finite data-independent partition of Δ_{m-1}. Then we have

$$\begin{aligned}\eta_d&=\mathbb{E}[d(r(g(X)),g(X))]&\text{(III.14)}\\&=\sum_{i=1}^{l}\mathbb{P}[g(X)\in\Phi^{(i)}]\mathbb{E}[d(r(g(X)),g(X))\,|\,g(X)\in\Phi^{(i)}]&\text{(III.15)}\\&\geq\sum_{i=1}^{l}\mathbb{P}[g(X)\in\Phi^{(i)}]d\left(\mathbb{E}[r(g(X))\,|\,g(X)\in\Phi^{(i)}],\mathbb{E}[g(X)\,|\,g(X)\in\Phi^{(i)}]\right),\\&&\text{(III.16)}\end{aligned}$$

where Equation (III.15) follows from the law of total probability and Equation (III.16) from Jensen's inequality. Hence by the continuous mapping theorem we get

$$\eta_d\geq\lim_{n\to\infty}\sum_{i=1}^{l}\widehat{p}^{(i)}d(\widehat{r}^{(i)},\widehat{g}^{(i)})=\lim_{n\to\infty}\widehat{\eta}_d,$$

with limit in the almost sure sense. The exact equality condition is obtained by unwrapping the equality condition in Jensen's inequality. □

8 Binning schemes

With enough data available the calibration function can be approximated by partitioning the probability simplex into bins and calculating the observed empirical frequencies of realized outcomes associated with the bins, e.g., by using the histogram regression of Nobel [19]. In the commonly considered binary classification setting the unit interval $[0, 1]$ is typically split into a given number of intervals of equal width [2] (fixed-width binning). This approach can be extended to multiple dimensions by using symmetric equally-sized higher-dimensional bins but the number of bins grows exponentially with the number of classes. Additionally, predictions of neural networks after training are usually highly non-uniformly distributed, often making accurate estimation of the calibration function in multiclass classification with moderately sized amounts of data practically infeasible in large parts of the probability simplex.

Thus an attractive alternative is to partition the probability simplex into bins with approximately equal number of predictions. Bröcker [2] suggests this binning procedure even in the case of non-uniformly distributed predictions of binary outcomes. In our study we employed a simple recursive partitioning scheme and split predictions along the mean of the dimension with highest variance as long as the number of predictions per bin was above a given threshold value, which was typically set to 1000 in our experiments. As discussed by Nobel [19], different data-dependent binning schemes are possible and described in literature.

9 Additional visualizations

9.1 Gaussian mixture model

Perfect model

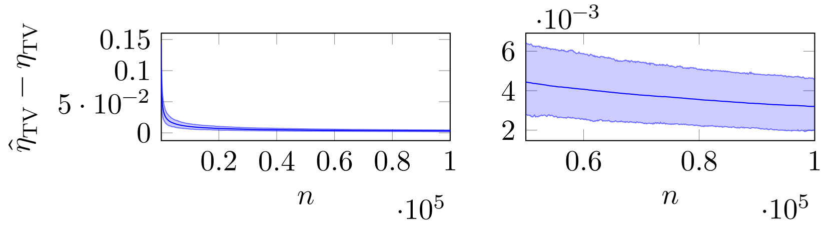


Figure 6: 5th percentile, mean, and 95th percentile of the difference of estimated and expected miscalibration of the perfect model ($\beta_0 = 0$, $\beta_1 = -2$) w.r.t. the total variation distance and 10 equally-sized bins (1000 series of random data).

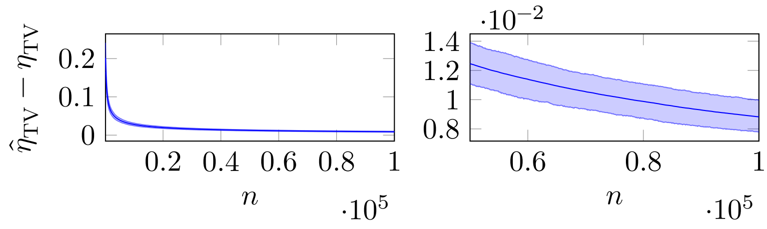
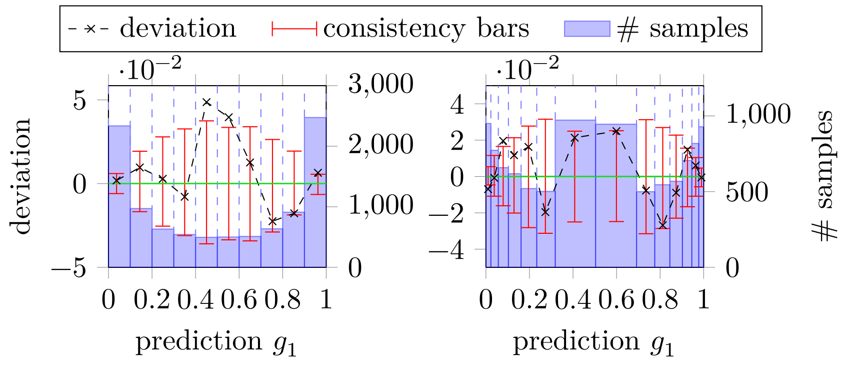


Figure 7: 5th percentile, mean, and 95th percentile of the difference of estimated and expected miscalibration of the perfect model ($\beta_0 = 0$, $\beta_1 = -2$) w.r.t. the total variation distance and 100 equally-sized bins (1000 series of random data).



(a) Equally-sized bins (b) Data-dependent bins

Figure 8: Reliability diagrams for the perfect model ($\beta_0 = 0$, $\beta_1 = -2$) w.r.t. the total variation distance on a randomly generated test set (10000 inputs). Crosses indicate the deviation of the outcome distribution from the predictions in each bin. Blue bars show the distribution of predictions. Red bars visualize the 5th and 95th percentiles of the deviation in 1000 consistency resamples. The green curve shows the true analytical deviation.

Calibrated constant model

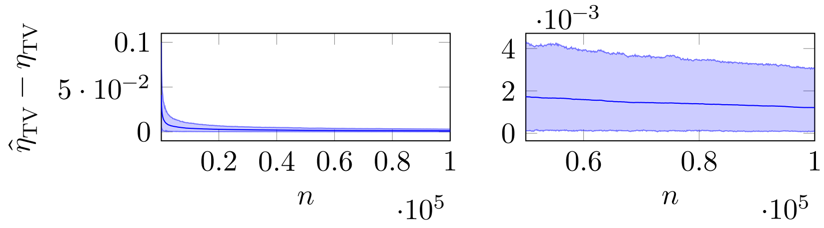


Figure 9: 5th percentile, mean, and 95th percentile of the difference of estimated and expected miscalibration of the calibrated constant model ($\beta_0 = \beta_1 = 0$) w.r.t. the total variation distance and 10 equally-sized bins (1000 series of random data).

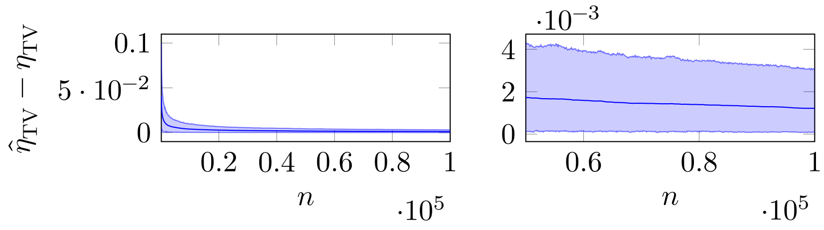
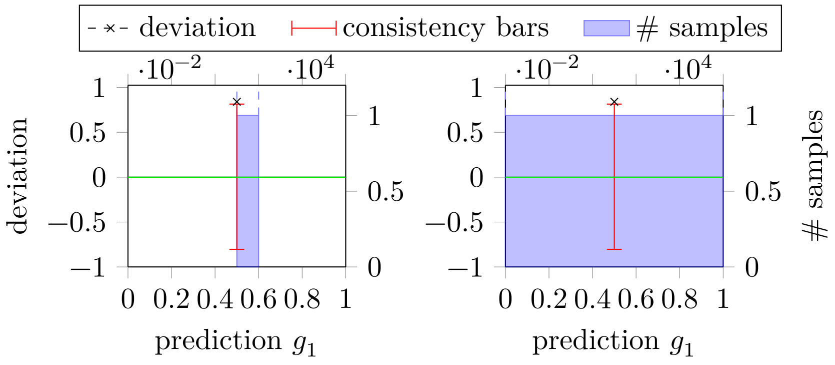


Figure 10: 5th percentile, mean, and 95th percentile of the difference of estimated and expected miscalibration of the calibrated constant model ($\beta_0 = \beta_1 = 0$) w.r.t. the total variation distance and 100 equally-sized bins (1000 series of random data).



(a) Equally-sized bins (b) Data-dependent bins

Figure 11: Reliability diagrams for the calibrated constant model ($\beta_0 = \beta_1 = 0$) w.r.t. the total variation distance on a randomly generated test set (10000 inputs). Crosses indicate the deviation of the outcome distribution from the predictions in each bin. Blue bars show the distribution of predictions. Red bars visualize the 5th and 95th percentiles of the deviation in 1000 consistency resamples. The green curve shows the true analytical deviation.

Uncalibrated model

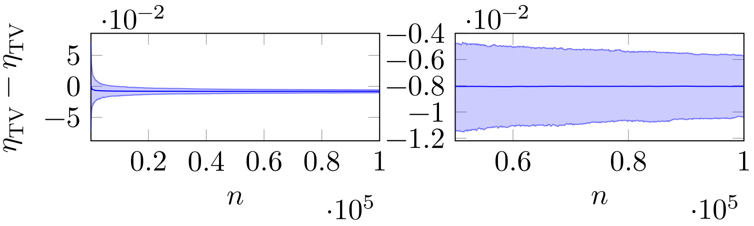


Figure 12: 5th percentile, mean, and 95th percentile of the difference of estimated and expected miscalibration of the uncalibrated model ($\beta_0 = \beta_1 = 1$) w.r.t. the total variation distance and 10 equally-sized bins (1000 series of random data).

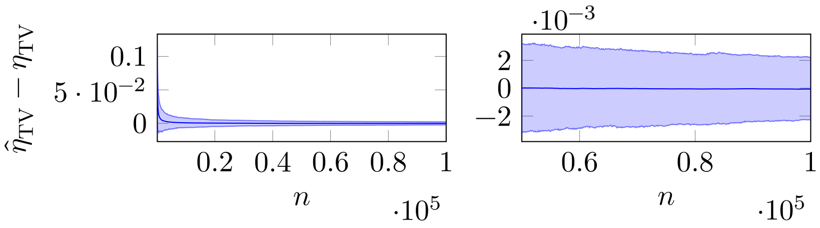
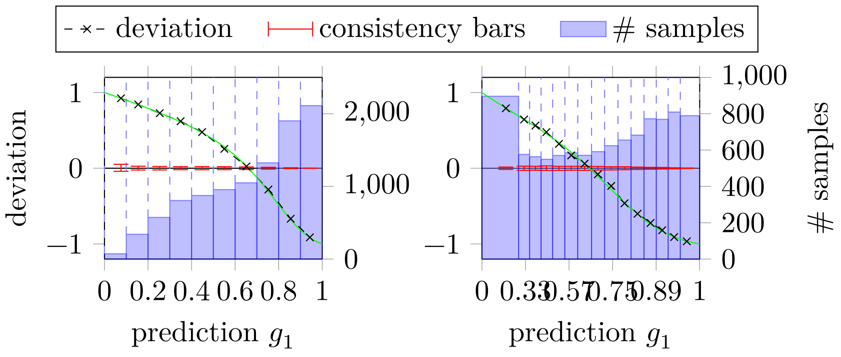


Figure 13: 5th percentile, mean, and 95th percentile of the difference of estimated and expected miscalibration of the uncalibrated model ($\beta_0 = \beta_1 = 1$) w.r.t. the total variation distance and 100 equally-sized bins (1000 series of random data).



(a) Equally-sized bins (b) Data-dependent bins

Figure 14: Reliability diagrams for the uncalibrated model ($\beta_0 = \beta_1 = 1$) w.r.t. the total variation distance on a randomly generated test set (10000 inputs). Dots indicate the deviation of the outcome distribution from the predictions in each bin. Blue bars show the distribution of predictions. Red bars visualize the 5th and 95th percentiles of the deviation in 1000 consistency resamples. The green curve shows the true analytical deviation.

9.2 Neural network models

DenseNet on CIFAR-10

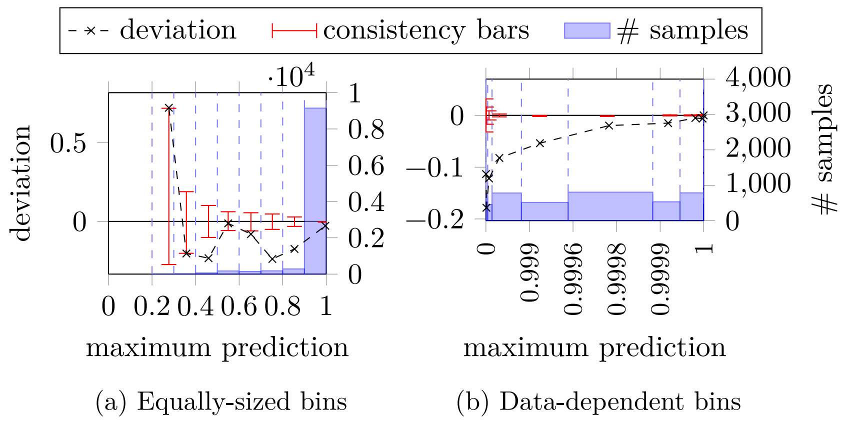


Figure 15: Reliability diagrams for the maximum predictions of DenseNet on the CIFAR-10 test set w.r.t. the total variation distance. Crosses indicate the deviation of the outcome distribution from the predictions in each bin. Blue bars show the distribution of predictions. Red bars visualize the 5th and 95th percentiles of the deviation in 1000 consistency resamples.

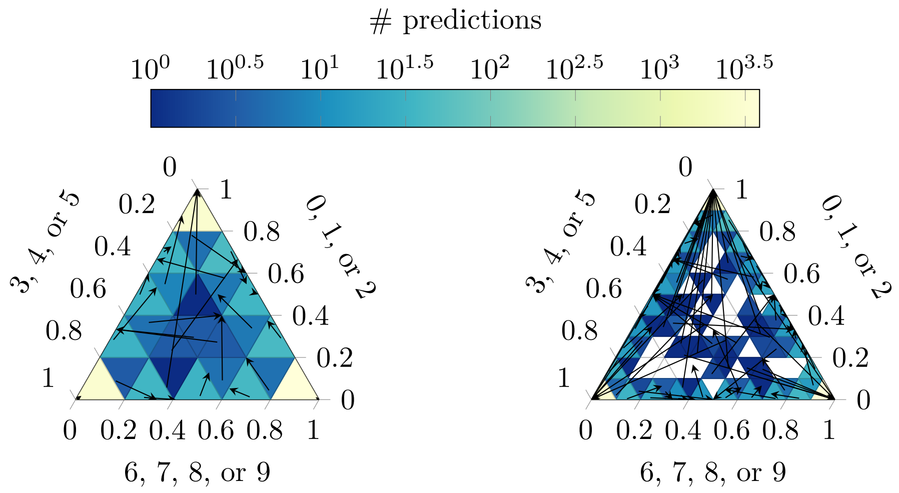
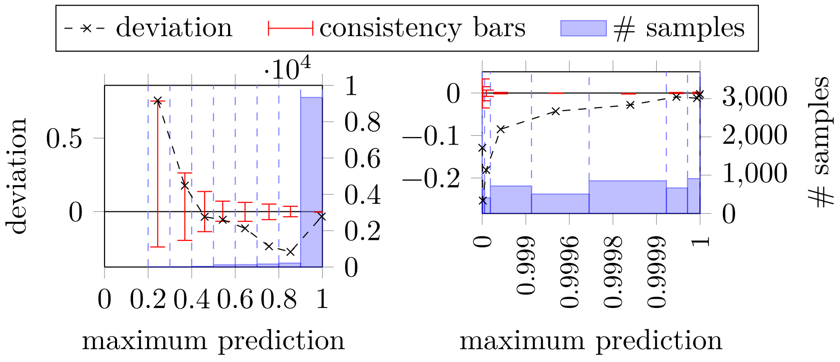


Figure 16: Two-dimensional reliability diagrams for DenseNet on the CIFAR-10 test set with 25 and 100 bins of equals. The predictions are grouped into three groups $\{0, 1, 2\}$, $\{3, 4, 5\}$, and $\{6, 7, 8, 9\}$ of the original classes. Arrows represent the deviation of the estimated calibration function value (arrow head) from the group prediction average (arrow tail) in a bin. The empirical distribution of predictions is visualized by color-coding the bins.

ResNet on CIFAR-10



(a) Equally-sized bins (b) Data-dependent bins

Figure 17: Reliability diagrams for the maximum predictions of ResNet on the CIFAR-10 test set w.r.t. the total variation distance. Crosses indicate the deviation of the outcome distribution from the predictions in each bin. Blue bars show the distribution of predictions. Red bars visualize the 5th and 95th percentiles of the deviation in 1000 consistency resamples.

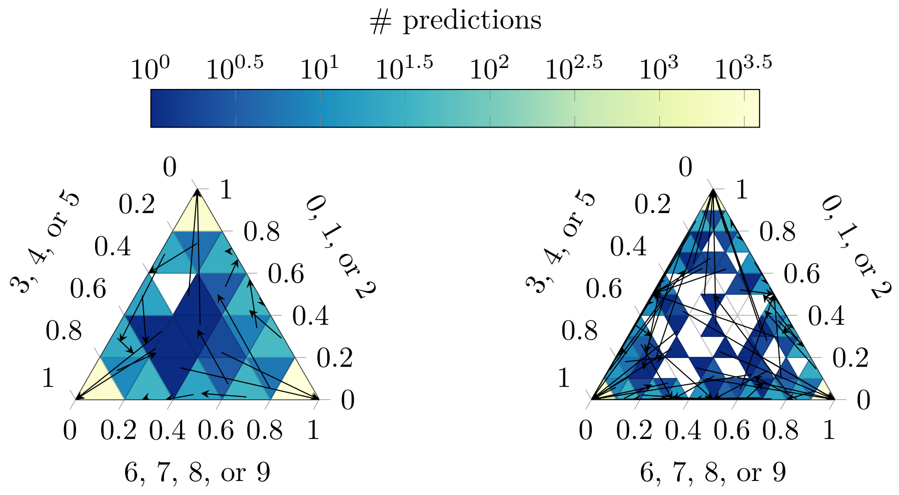


Figure 18: Two-dimensional reliability diagrams for ResNet on the CIFAR-10 test set with 25 and 100 bins of equal size. The predictions are grouped into three groups $\{0, 1, 2\}$, $\{3, 4, 5\}$, and $\{6, 7, 8, 9\}$ of the original classes. Arrows represent the deviation of the estimated calibration function value (arrow head) from the group prediction average (arrow tail) in a bin. The empirical distribution of predictions is visualized by color-coding the bins.

10 Expected miscalibration estimates for neural networks

The results presented in Tables 3 to 5 show estimates of the expected miscalibration η for DenseNet, ResNet, and LeNet models trained on CIFAR-10, using different binning schemes (see Appendix 8), calibration lenses (see Example 1), and distance functions (see Appendix 3.2). Estimates with respect to the total variation distance for maximum predictions and equally-sized bins correspond to the expected miscalibration error used by Guo et al. [6]. For comparison, we also approximate estimates of the expected miscalibration under the consistency assumption of a perfectly calibrated model.

Standard deviations are estimated using bootstrapping. The accuracy of the investigated DenseNet, ResNet, and LeNet models is 0.933 ± 0.002, 0.934 ± 0.002, and 0.727 ± 0.004, respectively.

Table 3: Estimates of the expected miscalibration for DenseNet trained on CIFAR-10.

Calibration lens	Distance d	Equally-sized bins		Data-dependent bins	
		$\widehat{\eta}_d$	$\widehat{\eta}_d^{\mathrm{id}}$	$\widehat{\eta}_d$	$\widehat{\eta}_d^{\mathrm{id}}$
Canonical	Total variation	0.059 ± 0.002	0.029 ± 0.001	0.041 ± 0.002	0.007 ± 0.001
	Squared Euclidean	0.072 ± 0.003	0.034 ± 0.001	0.046 ± 0.003	0.006 ± 0.001
Maximum	Total variation	0.038 ± 0.002	0.002 ± 0.001	0.038 ± 0.002	0.001 ± 0.001
	Squared Euclidean	0.054 ± 0.003	0.006 ± 0.001	0.053 ± 0.003	0.004 ± 0.001

Table 4: Estimates of the expected miscalibration for ResNet trained on CIFAR-10.

Calibration lens	Distance d	Equally-sized bins		Data-dependent bins	
		$\widehat{\eta}_d$	$\widehat{\eta}_d^{\mathrm{id}}$	$\widehat{\eta}_d$	$\widehat{\eta}_d^{\mathrm{id}}$
Canonical	Total variation	0.059 ± 0.002	0.022 ± 0.001	0.042 ± 0.002	0.007 ± 0.001
	Squared Euclidean	0.071 ± 0.003	0.028 ± 0.001	0.047 ± 0.003	0.005 ± 0.001
Maximum	Total variation	0.043 ± 0.002	0.002 ± 0.001	0.043 ± 0.002	0.001 ± 0.001
	Squared Euclidean	0.061 ± 0.003	0.004 ± 0.001	0.061 ± 0.003	0.004 ± 0.001

Table 5: Estimates of the expected miscalibration for LeNet trained on CIFAR-10.

Calibration lens	Distance d	Equally-sized bins		Data-dependent bins	
		$\widehat{\eta}_d$	$\widehat{\eta}_d^{\mathrm{id}}$	$\widehat{\eta}_d$	$\widehat{\eta}_d^{\mathrm{id}}$
Canonical	Total variation	0.219 ± 0.003	0.215 ± 0.003	0.027 ± 0.003	0.023 ± 0.002
	Squared Euclidean	0.243 ± 0.004	0.238 ± 0.003	0.024 ± 0.003	0.019 ± 0.002
Maximum	Total variation	0.007 ± 0.003	0.010 ± 0.002	0.009 ± 0.003	0.011 ± 0.002
	Squared Euclidean	0.010 ± 0.004	0.015 ± 0.003	0.013 ± 0.004	0.011 ± 0.003

Recent licentiate theses from the Department of Information Technology

2019-006	Kristiina Ausmees: Efficient Computational Methods for Applications in Genomics
2019-005	Carl Jidling: Tailoring Gaussian Processes for Tomographic Reconstruction
2019-004	Amin Kaveh: Local Measures for Probabilistic Networks
2019-003	Viktor Bro: Volterra Modeling of the Human Smooth Pursuit System in Health and Disease
2019-002	Anton G. Artemov: Inverse Factorization in Electronic Structure Theory: Analysis and Parallelization
2019-001	Diane Golay: An Invisible Burden: An Experience-Based Approach to Nurses' Daily Work Life with Healthcare Information Technology
2018-004	Charalampos Orfanidis: Robustness in Low Power Wide Area Networks
2018-003	Fredrik Olsson: Modeling and Assessment of Human Balance and Movement Disorders Using Inertial Sensors
2018-002	Tatiana Chistiakova: Ammonium Based Aeration Control in Wastewater Treatment Plants - Modelling and Controller Design
2018-001	Kim-Anh Tran: Static Instruction Scheduling for High Performance on Energy-Efficient Processors
2017-003	Oscar Samuelsson: Fault Detection in Water Resource Recovery Facilities
2017-002	Germán Ceballos: Modeling the Interactions Between Tasks and the Memory System

UPPSALA
UNIVERSITET

Department of Information Technology, Uppsala University, Sweden