



UPPSALA  
UNIVERSITET

*Digital Comprehensive Summaries of Uppsala Dissertations  
from the Faculty of Science and Technology 2139*

# Deep probabilistic models for sequential and hierarchical data

CARL ANDERSSON



ACTA  
UNIVERSITATIS  
UPSALIENSIS  
UPPSALA  
2022

ISSN 1651-6214  
ISBN 978-91-513-1478-5  
URN urn:nbn:se:uu:diva-470433

Dissertation presented at Uppsala University to be publicly examined in Sonja Lyttkens, 101121, Ångström, Lägerhyddsvägen 1, Uppsala, Tuesday, 24 May 2022 at 09:00 for the degree of Doctor of Philosophy. The examination will be conducted in English. Faculty examiner: Professor Roy Smith (Swiss Federal Institute of Technology).

### **Abstract**

Andersson, C. 2022. Deep probabilistic models for sequential and hierarchical data. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 2139. 87 pp. Uppsala: Acta Universitatis Upsaliensis. ISBN 978-91-513-1478-5.

Consider the problem where we want a computer program capable of recognizing a pedestrian on the road. This could be employed in a car to automatically apply the brakes to avoid an accident. Writing such a program is immensely difficult but what if we could instead use examples and let the program learn what characterizes a pedestrian from the examples. Machine learning can be described as the process of teaching a model (computer program) to predict something (the presence of a pedestrian) with help of data (examples) instead of through explicit programming.

This thesis focuses on a specific method in machine learning, called deep learning. This method can arguably be seen as sole responsible for the recent upswing of machine learning in academia as well as in society at large. However, deep learning requires, in human standards, a huge amount of data to perform well which can be a limiting factor. In this thesis we describe different approaches to reduce the amount of data that is needed by encoding some of our prior knowledge about the problem into the model. To this end we focus on sequential and hierarchical data, such as speech and written language.

Representing sequential output is in general difficult due to the complexity of the output space. Here, we make use of a probabilistic approach focusing on sequential models in combination with a deep learning structure called the variational autoencoder. This is applied to a range of different problem settings, from system identification to speech modeling.

The results come in three parts. The first contribution focus on applications of deep learning to typical system identification problems, the intersection between the two areas and how they can benefit from each other. The second contribution is on hierarchical data where we promote a multiscale variational autoencoder inspired by image modeling. The final contribution is on verification of probabilistic models, in particular how to evaluate the validity of a probabilistic output, also known as calibration.

*Keywords:* Machine learning, Deep learning, Sequential modelling

*Carl Andersson, Department of Information Technology, Division of Systems and Control, Box 337, Uppsala University, SE-75105 Uppsala, Sweden.*

© Carl Andersson 2022

ISSN 1651-6214

ISBN 978-91-513-1478-5

URN urn:nbn:se:uu:diva-470433 (<http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-470433>)

*Till morfar*



# Populärvetenskapling sammanfattning

*“När förändringens vind blåser, bygger en del vindskydd  
medan andra bygger väderkvarnar.”*

– Kinesiskt ordspråk

Människor är nyfikna och när det kommer till att utforska, tolka och förstå vår omgivning finns det inget som klår oss. Vi kan skaffa oss nya färdigheter och insikter enbart genom att observera och interagera med världen omkring oss. Maskininlärning handlar om att ge maskiner eller datorprogram en del av den förmågan; maskinen skall kunna lära sig utifrån exempel. I maskininlärning pratar man dock sällan om maskiner eller datorprogram utan istället modeller; vilka kan ses som komponenter i datorprogram. I grunden är dessa modeller egentligen matematiska funktioner vilka autonomt kan anpassas till de exempel (vanligtvis kallat data) som de lär sig av; en bättre term kanske således hade varit autonom modellanpassning.

Sedan den industriella revolutionen har maskiner hjälpt människan att genomföra monotona och repetitiva uppgifter. Detta har också fortsatt in i den digitala världen med datoriseringen. Vissa uppgifter har dock varit alltför komplicerade för att kunna programmeras explicit. Ett exempel på det är att identifiera ett föremål i en okänd miljö från en bild, se till exempel Figur 1. Maskininlärning möjliggör detta och löser problem som tidigare uppfattats som omöjliga.

Maskininlärning är i grunden ett förhållandevis gammalt koncept och formulerades redan i början av datoriseringen på 1960-talet. Det är dock under det senaste decenniet som maskininlärning tagit steget ut ur den akademiska sfären och börjat påverka samhället i allt större utsträckning; framförallt inom röst- och bildigenkänning med exempel så som robotdammsugare, självkörande bilar och personliga assistenter som Google now och iPhones Siri. Det är svårt att säga exakt vilka faktorer som har varit viktigast för att det steget skulle tas just nu, men en viktig faktor är rent beräkningsmässig i kombination med mängden data; det gick helt enkelt inte att göra de beräkningarna som behövdes tillräckligt snabbt tidigare. Detta har i sin tur lett till flera viktiga tekniska framsteg som förstärkt effekten av ökad beräkningskapacitet.

Denna avhandling behandlar framförallt en specifik maskininlärningsteknik kallad *deep learning*; en relativt ung teknik som kräver väldigt mycket data, men som också ger väldigt bra resultat. Från ett mänskligt perspektiv är dock



 Recognition results: Power Strip

Confidence: 85%

**Figur 1.** Från en robotdammsugares perspektiv har en maskininlärningsmodell i robotdammsugaren identifierat en grenkontakt under bokhyllan. Den blå rutan representerar var i bilden som modellen lokaliserat objektet; grenkontakten i detta fall. Programmet som styr dammsugaren är programmerat att undvika grenkontakter för att slippa trassel med sladdarna och kommer styra undan från den. Att manuellt (d.v.s. utan hjälp av maskininläring) programmera en modell som identifierar grenkontakter är nära på omöjligt.

mängden data ofta uppseendeväckande stor och överstiger med råge den mängden vi människor behöver för att lära oss en ny förmåga; ofta är den flera storleksordningar större. I denna avhandling presenterar vi olika alternativ för hur man kan utnyttja på förhand känd information och inkludera det i designen av modellen. Med en sådan modell kan man reducera mängden data men med bibehållen prestanda, alternativt få bättre prestanda med samma mängd data.

En begränsade faktor när det kommer till maskininläring är typiskt sett mängden tillgänglig data; framförallt den mängden data som har bra kvalitet. Föreställ dig en organisation som vill automatisera en process med hjälp av maskininläring. I en sådan situation är en faktor som lätt förbises kostnaden för att behandla data till en bra kvalitet. Den data som redan finns tillgänglig är ofta oorganiserad och ostandardiserad, något som har negativ inverkan på modellen. Att reducera mängden data som behöver förbehandlas genom att utnyttja redan kända egenskaper i datan kan således vara avgörande för om ett projekt kommer att lyckas.

I den här avhandlingen fokuserar vi på hur man kan utnyttja att datan har sekventiell natur och hur man kan kombinera det med deep learning, samt hur klassiska modeller för sekventiell data förhåller sig till modeller som bygger på deep learning. Vi behandlar också flera olika metoder som berör generering av sekvenser, till exempel att generera tal och text, och hur modeller som gör detta kan konstrueras effektivt.

# Acknowledgment

*“Try not. Do or do not. There is no try.”*

– Yoda, Star Wars

What do you say that summarizes five years of research? ”Eureka!”? ”Reach for the stars, end up in Storsylen<sup>1</sup>”? The least I can say is thank you – thank you for this opportunity and thank you for your support.

On a more personal level I would like to thank my supervisors Thomas Schön and Niklas Wahlström. Thank you for your support, feedback, encouragement and for believing in me, even in situations when I started to doubt myself. I am also grateful that you can take time for non-academic discussions, such as tips on the stock market or about hiking in Sarek.

I would also like to thank the projects that financially supported this work: the Swedish Research Council (VR) via the project *NewLEADS - New Directions in Learning Dynamical Systems* (contract number: 621-2016-06079); the Swedish Foundation for Strategic Research (SSF) via the project *ASSEMBLE* (contract number: RIT15-0012); *Learning flexible models for nonlinear dynamics* (contract number: 2017-03807) by the Swedish Research Council; *AI4Research* at Uppsala University; and *Kjell och Märta Beijer Foundation*. This work have also been supported with computational resources provided by the Swedish National Infrastructure for Computing (SNIC) at C3SE partially funded by the Swedish Research Council through grant agreement no. 2018-05973, as well as the generous sharing of computational resources provided by Joakim Jaldén at Kungliga Tekniska Högskolan.

This thesis would not have been possible without my co-authors and collaborators: Antônio, Koen, Juozas and David W, as well as, Össur and Henrik. I wish you all the best in your future endeavors. A big thanks to Calle J and Daniel for proof reading and your feedback on this thesis, and to Malin for the awesome cover art.

When I started my PhD studies I searched for a stimulating environment. Syscon and Uppsala University have stepped up and really delivered on that front, not only academically but also socially. Thank you, all former and current colleagues for making every moment more enjoyable, be it pub nights, running sessions or ice cream walks. Thank you, Aleksandar, Calle J, David W and Fredrik O for memorable boardgame nights. Thank you Jelena, Jonathan,

---

<sup>1</sup>The highest mountain climbed by the author during these years

Kristina and Simon for introducing me to one of the best sports in the world, Ultimate Frisbee; and to everyone else I have thrown a Frisbee with since then. Thank you, Fredrik W and everyone else in Fruitvendor Entertainment™ for disconnecting me from IRL from time to time. Thanks also to Grabbarna Grus for letting me be a part of your lives.

Those of you who know me, might notice that one person is missing above – one that could fit in almost every context. A special thanks to Anna W for always being there, no matter what.

Sist men inte minst, tack till mamma och pappa, och mina syskon, Johan och Malin med familjer, för all support och kärlek ni ger. Ni är den bästa familj man kan be om.



# Contents

1	Introduction .....	11
1.1	The sequential problem .....	11
1.2	Generative models .....	13
1.3	Deep learning .....	14
1.4	Hierarchically structured data .....	16
1.5	Running examples .....	17
1.5.1	Example: Cough classification .....	17
1.5.2	Example: Generative models .....	18
1.5.3	Example: Text-to-speech .....	19
1.5.4	Example: Circuit modeling .....	20
1.6	Contribution .....	21
1.7	Related but not included work .....	22
1.8	Thesis outline .....	23
2	Learning in a sequential setting .....	25
2.1	Training a model .....	26
2.2	Bias-variance trade-off .....	26
2.3	Training hyperparameters .....	28
2.4	Sequential models .....	28
2.5	State-space models .....	30
2.6	Autoregressive models .....	31
2.7	Example: Circuit modeling .....	33
3	Deep probabilistic models .....	35
3.1	Neural networks .....	35
3.2	Deep generative models .....	37
3.3	Variational autoencoders .....	39
3.3.1	REINFORCE and the reparameterization trick .....	41
3.3.2	Posterior collapse .....	42
3.4	Hierarchical variational autoencoders .....	43
3.5	Transfer Learning .....	45
3.6	Example: Cough classification .....	47
3.7	Calibration .....	47
4	Sequential deep learning models .....	53

4.1	Recurrent neural networks .....	53
4.2	Example: Circuit modeling continued .....	56
4.3	Long short-term memory model .....	56
4.4	Multiscale recurrent neural networks .....	58
4.5	Stochastic recurrent neural networks .....	60
4.6	Temporal convolutional networks .....	62
4.7	Stochastic temporal convolutional network .....	65
4.8	Multiscale autoregressive models .....	67
4.9	Example: Generative models .....	69
4.10	Transformers .....	70
4.11	Sequential classification .....	72
4.12	Sequential translation .....	73
4.13	Example: Text-to-speech .....	73
5	Conclusion and future work .....	77
5.1	Conclusion .....	77
5.2	Future work .....	78
	References .....	79

# 1

## Introduction

*“Look deep into nature, and then you understand everything better.”*

– Albert Einstein

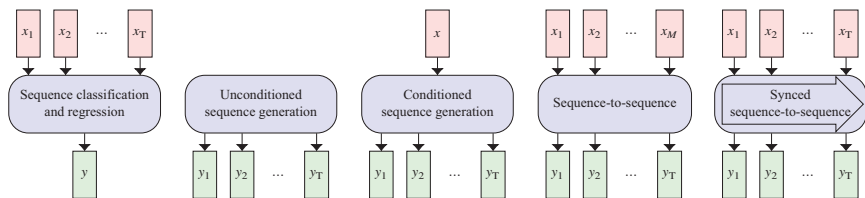
Machine learning sounds more abstract than it really is and can, in a nutshell, be described as finding patterns in data. In the machine learning lingo, this is called fitting a model to the data or training a model. Typically, machine learning problems are divided into three groups: supervised learning, unsupervised learning and reinforcement learning, which are distinguished from each other by how the data is used to train the model. A supervised model predicts a set of output features given a set of input features and needs matching examples of input and output data to be trained. An unsupervised model finds patterns in (or models) the data to discover properties or relations that explain the data without relying on any predefined labels, e.g. as in clustering. A reinforcement learning model learns how to act/predict given only a feedback signal (reward), while at the same time being responsible for gathering more data by interacting with its environment.

The vast majority of machine learning algorithms represents the model as a set of functions – machine learning and function approximation can thus be used almost interchangeably. An important aspect of this is that we can encode any prior knowledge that we have about the data into this function and the optimization of the function, to significantly increase the model’s performance.

This introductory chapter gives a brief background to the core concepts used in this thesis and puts them into a broader context. Furthermore, it provides the foundation for the chapters to come and how they relate to each other.

### 1.1 The sequential problem

The model structures used for the machine learning problem highly depend on the data they are applied to. Problems defined by data with a natural sequential ordering, i.e. sequential problems, can roughly be divided into five cases: sequence classification and regression; unconditioned sequence generation; conditioned sequence generation; sequence-to-sequence; and synced



**Figure 1.1:** The five different sequential model structures. The subscripts of the **inputs** and the **outputs** indicate the sequential order. Inputs and outputs without indices are considered nonsequential. The arrow in the case of synced sequence-to-sequence models indicates that information inside this box can not travel in the opposite direction, e.g.,  $y_1$  can not depend on  $x_2$ . Note that the input to the sequence-to-sequence model does not need to have the same length as the output.

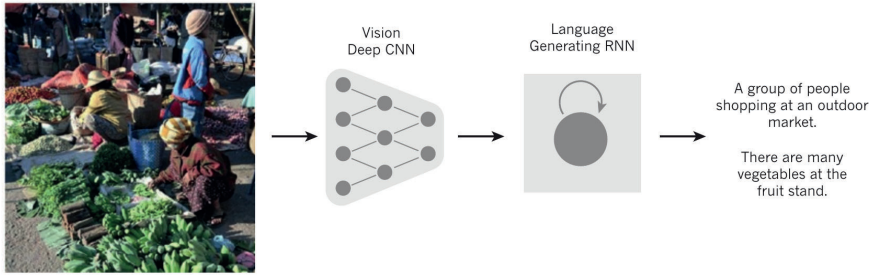
sequence-to-sequence. This division is visualized in Figure 1.1 and to exemplify four of these cases, we introduce running examples in Section 1.5.

**Sequence classification and regression** problems are defined by a sequential input and an output that we wish to predict. As an example of classification, we consider a problem where we want to binary classify whether a recorded sound is the sound of a person coughing or not. We introduce this problem further in Section 1.5.1.

All other problem cases have sequential output. For such problems without a dedicated input, i.e. **unconditioned sequence generation**, the goal is to accurately model and analyze the data and/or the generating system. The use case can be to continuously predict the future, i.e. forecasting, or to be able to generate new sequences. System identification without exogenous input [1, 2] is another example of this. This thesis use modeling of handwritten text and speech as an example of a sequence generation problem, where the intent of the modeling is to analyze the data and generate new examples. This is formally introduced in Section 1.5.2. A natural extension to this is to condition on some nonsequential data, i.e. **conditioned sequence generation**. An example of this is to generate handwritten text according to some input. We illustrate this case in Figure 1.2, but do not cover it with a dedicated example.

We can also consider the case where we condition on a sequential input. We call this case a **sequence-to-sequence** (also referred to as seq2seq [3]) or a translation problem due to its structural similarity with the problem of translating a sentence from one language to another language. In this thesis we exemplify the sequence-to-sequence problem in Section 1.5.3, where the goal is to generate speech from text.

Finally, we consider the case of **synced sequence-to-sequence** problems. Problems on this form are often similar to the unconditioned case, however, the use case is slightly broader as it can also include filtering, reinforcement learning and control problems. The fundamental difference between synced sequence-to-sequence and translation is that information can only travel for-



**Figure 1.2:** The output of a sequential model is often probabilistic and the output needs to be sampled to be interpretable. Here the model has produced two different sampled captions when conditioned on the image on the left. Reprinted by permission from Springer Nature: Deep learning, LeCun et al. [4] Copyright (2015).

ward inside the sequential model – something we call a causal constraint. This constraint limits the model so that the output can never depend on any inputs from the future. Examples of this case are control, model-based reinforcement learning and system identification with exogenous input [1, 2]. The last of these is used as a running example in Section 1.5.4.

A very common assumption for sequential models is that they are time-invariant. In machine learning there are three large classes of models that adhere to this property: autoregressive models with finite history; state-space models with a finite state-space; and transformers, a type of kernel-based models that can be considered to belong to both of the already mentioned classes [5]. Chapter 2 gives a more thorough background on autoregressive and state-space models, and transformers are further discussed in Section 4.10.

Traditionally, system identification has provided the go-to methods for sequential problems. The feasible problems were limited to problems that required either a relatively short memory or a very deep understanding of the data by the model engineer. With deep learning (see Section 1.3) the set of feasible problems changed heavily in favor of problems that require longer memory, broadening the range of solvable problems. This has generated an explosion of new models and methods, e.g. electrocardiogram classification [6], speech and text language modeling [7, 8], human motion prediction [9] and human level Starcraft II gameplay [10]. However, models based on deep learning tend to be a lot more data hungry, i.e. require more data, than system identification models.

## 1.2 Generative models

Classically, a generative model is viewed in contrast to a discriminative (supervised) model [11]. While both models are probabilistic, the generative model considers the joint distribution, i.e. the inputs and the outputs, and the

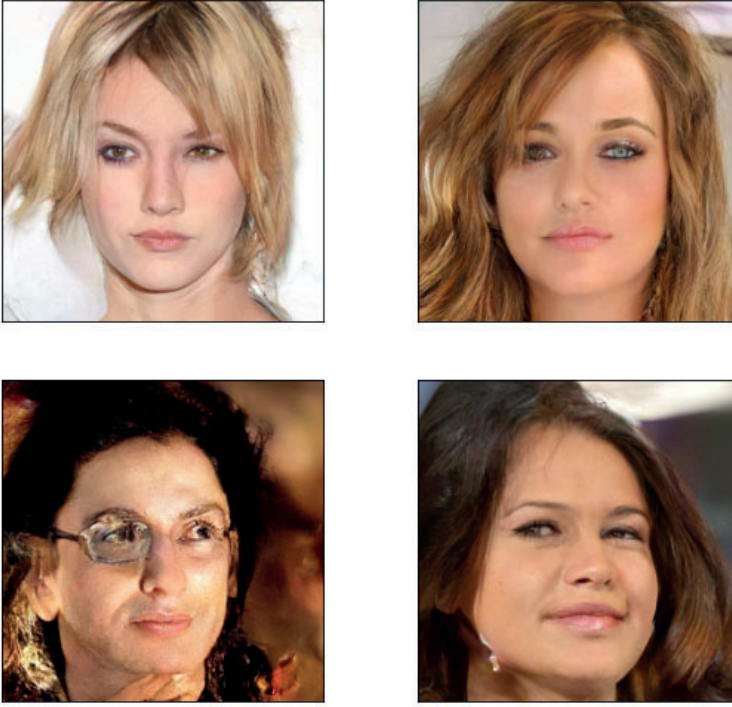
discriminative model considers the conditional distribution, i.e. the output conditioned on the input. However, in this thesis we slightly deviate from this terminology and use the term generative model in a wider sense. In generative models we also include problems that have a dedicated input if the output space is complicated and a substantial part of the complexity in the model appears in modeling the output space rather than the conditioning input space. This is the case for many models with sequential output, where the output distribution often is very complex. See for example Figure 1.2, where the same input image can generate several quite distinct plausible captions.

A generative model can also be seen in the light of the slightly wider concept of unsupervised modeling, which is not limited by the probabilistic notation that accompanies the generative terminology. The purpose of unsupervised modeling is to get insights about the data, or as a tool to enable a supervised machine learning algorithm to perform better when the amount of labeled data is scarce (i.e. semi-supervised learning and transfer learning, see Section 3.5). A generative model can fulfill this role, while also being able to generate artificial data and possibly detect out-of-distribution data.

One purpose for the generative models that are used today is to push the advancement of new architectures. Generative models enable us to investigate what properties of the data the model can capture, e.g. by visually inspecting samples from the model. In this way, we can guide the model development towards models that are able to represent features that humans identify as important in the data. The learned model structures can then be translated to other problem setups, e.g. a supervised problem, which can benefit from these features and also increases the interpretability of the model. One example of generative image modeling is the NVAE model [12] trained to generate images of humans, see Figure 1.3. Even though these generated images have some artifacts, we can see that the model generates many of the essential features of a face.

## 1.3 Deep learning

Efficiently approximating functions is fundamental in essentially all machine learning. Be it predicting the electricity price in 24 hours [13] or classifying the bird species producing that chirp you heard outside your window [14]. The great conundrum here is if and how we can get a model to learn as efficiently as a human, child or adult. In other words, how can we constitute the, for us, inherent human prior that seems rudimentary for efficient learning, into a model. Although humans as a species are far from unraveling the true nature of this problem, the recent breakthrough of machine learning using neural networks have made significant progress on this matter – solving problems that only 15 years ago would have been classified as impossible.



**Figure 1.3:** Images generated from the NVAE model [12].

Neural networks have, from being a niche area of machine learning, grown into one of the core pillars of machine learning today. The specific technical contributions behind this growth can be hard to pinpoint. Not only have increased computational power made it possible to do more and larger experiments, but, as more experiments are done, more data is also accumulated into ever-growing datasets supporting these complex models.

The rapid development of neural networks, which started in the early 2010s, led to networks that grew deeper and deeper, and researchers started to adopt the term deep learning. Originally, deep learning was introduced to denote models (not necessarily neural network-based) that were hierarchically divided into several layers, largely inspired by the functional behavior in popular biological models of the human brain. However, the domination of neural networks combined with the deep learning structure have ultimately led to the two becoming more or less synonymous with each other – both names now imply a heavily parameterized and layered function.

At present day, deep learning (with neural networks) have approached a more stable regime of its hype curve, where much of the research is focused on application or adoption to new fields. A large portion of the more architectural research have progressed to more niche domains, with datasets or com-

putational resources infeasible for most academic-funded research (GPT-3 [7], ViT [15], etc.). However, this does not mean the field is not moving, quite the opposite. Development of both practical and theoretical aspects are made continuously [16, 17], and applications and adoptions to new fields seems everlasting. It is now also possible to see a small shift in how deep learning models are applied. Rather than training a model from scratch, one can start off from models trained unsupervised on similar data and fine-tune them for the problem at hand (see Section 3.5). This enables much larger models to be used without overfitting. Recently, these large unsupervised models were dubbed foundation models [18] for their use as a starting point to build many different models.

## 1.4 Hierarchically structured data

Some areas where deep learning has had its greatest breakthroughs are problems involving images or spoken/written language. Before the introduction of deep learning, these were problem areas where humans outshone machine learning algorithms significantly. This has since then changed dramatically and today a sufficiently trained image classification algorithm can surpass even expert human classifiers, e.g. skin cancer classification models that surpass dermatologists [19]. Another example is text generated with the GPT-3 model [7] that approaches a level where it can be hard to distinguish between text written by a model and by a human. This has opened up a whole new avenue of possibilities, e.g. a role-playing game/story told interactively with a text-generating model<sup>1</sup>.

Both language and image data have in common that they are what we call *hierarchically structured*. This structure manifests itself in that features that efficiently describe the data have three properties. The first is a *compositional* property, i.e., the features can be described hierarchically and advanced features can be expressed using more basic features. The second is a *locality* property, i.e., features on the same hierarchical layer are more correlated the closer they are. The third is a *consolidation* property, i.e., more advanced features tend to have lower frequency components than more basic features, i.e., advanced features correspond to a larger portion of the image or sequence.

To exemplify this we can consider features that represent an image of a house. The simplest features are local and describe a local neighborhood well. It could be the color of a patch in the image, the presence and the orientation of an edge or the texture that make up a brick wall. On a second level, these features can be combined in their local neighborhood to create larger features, e.g. windows, walls and doors. Finally, these features can make up the whole concept of a house and whether it has a postmodernism style or a traditional

---

<sup>1</sup>aidungeon.io



Swedish red cottage style. With a similar argument one can decompose a text, with letters forming up words and words that in turn make up sentences.

The hierarchical structure of deep learning models often matches the hierarchical structure of the data. It has been shown that this model structure imposes a beneficial inductive bias on the model [20, 21]. In other words, it incentivizes the model to learn inductive rules rather than memorizing the data. Since the inductive bias affects the output of the model in a similar way as a prior, we say that the hierarchical structure imposes a prior on the model. For the purpose of this thesis we call this a *deep hierarchical prior*. For many problems, such priors can have a big impact – especially when data is more scarce.

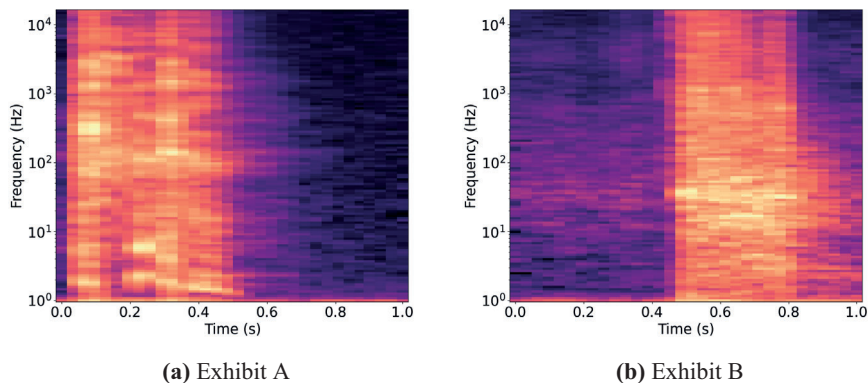
## 1.5 Running examples

To illustrate the different kinds of sequential problems we use four running examples that we revisit throughout the thesis. These examples are introduced in the following sections.

### 1.5.1 Example: Cough classification

Quantitative as opposed to qualitative measures can guide us to make objective decisions on what kind of treatment a patient in healthcare needs. It can also enable us to compare different drugs and evaluate their effectiveness in a more objective manner. One area that can benefit from quantitative measures is chronic cough. Chronic cough is estimated to affect 5-15 % of the adult population and is defined as coughing that lasts longer than 8 weeks [22, 23]. On a personal level, chronic cough is correlated with decreased life quality, social isolation and depression. Furthermore, patients with chronic cough have 50% higher chance of having seven or more days of sick leave per year [24]. Chronic cough also burdens the primary care, as cough is the third most common cause to seek healthcare at the primary care in Stockholm county, Sweden [25]. Measuring the severity of chronic cough for a patient, i.e. how often they cough, enables a quantitative measure on the effect on the patients' life quality and/or the effectiveness of a certain drug.

A simple quantitative measure on coughing is to count how many times per day the patient coughs. This can be done by giving a small collar mic and recording device to the patient for wear under a period of time (~24 hours). However, manually counting the incidence of coughs in the recording heavily limits the usage of this measure as it is very labor intensive and thus very expensive. The obvious alternative is to train a machine learning model to separate coughing sounds from other sounds and use this to count the number of coughs in the recorded audio. Since coughing sounds typically are around one second long, one possible approach is to simply split the recorded audio



**Figure 1.4:** Which of the spectrograms depicts a cough and which depicts dropped keys? Time is on the x-axis, frequency is on the y-axis and the power is shown with color, lighter color corresponds to higher power.<sup>2</sup>

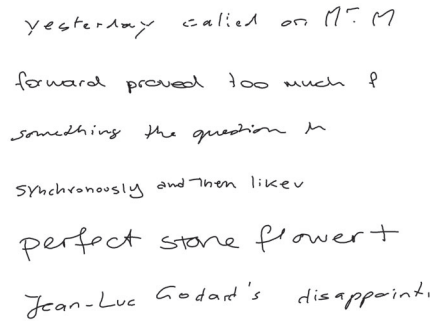
into one second chunks and classify each chunk as either cough or no cough, effectively turning the problem into a classification problem.

However, distinguishing between different sounds with machine learning is very difficult and traditionally mel-spectrogram features have been used to reduce the dimensionality of the input space [26] (see Figure 1.4). A spectrogram converts a signal into its frequency components and depicts how the power of each frequency component varies with time. Even so, the problem is still problematic and the traditional methods had to resort to manual supervision [27].

### 1.5.2 Example: Generative models

To model the generative process underlying some data can for multiple reasons be beneficial for the machine learning engineer, see Section 1.2. For the specific case of handwritten text, a trained generative model can be used to improve the decoding of the written text in a semi-supervised fashion. Similarly, it can also be used to detect out-of-distribution samples and frauds, e.g. fake signatures. Finally, it is also possible to use the trained model as a pretrained model for generating handwritten text conditioned on some input [28, 29].

In this example we consider two datasets. The first is a dataset of handwritten text on a digital whiteboard [30] from more than 200 different writers. The handwritten text consists of a series of pen positions coupled with an event of when the pen is lifted from the screen. The text that is written can be visualized by connecting all the pen positions for which the pen is not lifted. Some samples from the dataset can be seen in Figure 1.5. The second dataset is a speech dataset called Blizzard dataset [31] which consists of around 40 hours of 16-kHz speech from audio books read by a single reader.



yesterday called on M.T.M  
 forward proved too much &  
 something the question is  
 synchronously and then like  
 perfect stone flower +  
 Jean-Luc Godard's disappoint

**Figure 1.5:** Samples of handwritten text from five different writers in the IamOnDB [30] dataset.

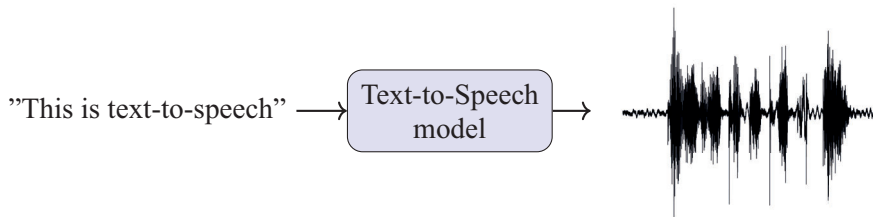
Both handwritten text and speech are complex and multimodal, and are difficult to model. The complexity of the data means that the model needs to be nonlinear, or have carefully designed features, to be able to accurately model the data. Furthermore, for downstream modeling capabilities, e.g. using the trained model to decode the text or speech, it is almost required to downsample the sequences. Examples when this is required are for problems with handwritten text, where the sampling rate of the whiteboard is approximately 25 samples per letter, and speech, where the typical phoneme length is tens or hundreds of milliseconds long, corresponding to 100-1000s samples. Thus, as a further constraint we concentrate on models that used downsampled intermediate features to model the sequences.

### 1.5.3 Example: Text-to-speech

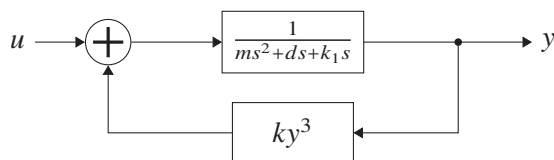
Text-to-speech, or speech synthesis [8, 32], is what we in this thesis call a sequence-to-sequence problem, where the goal is to generate speech conditioned on some text, see Figure 1.6. Speech synthesis is not only another interface for digital communication, but also a tool for speech impaired humans to speak or visually impaired humans to read.

An inherent problem with all sequence-to-sequence problems is that the input sequence and the output sequence have different lengths and the causal relationship between the two sequences could be arbitrary. This is very apparent when translating between Japanese to English, where the order of subject, predicate and object changes completely. In text-to-speech this mainly becomes a problem of alignment as different parts of the text input are pronounced at different rates, c.f. the length of a t sound with the pause due to a period. A common model structure for sequence-to-sequence problems, especially when

<sup>2</sup> Exhibit A is a coughing sound. The cough sounds starts in the beginning of the sequence and finish after 0.5 seconds. The key-drop sound starts at 0.5 seconds and finishes at 0.8.



**Figure 1.6:** The basic idea of a text-to-speech model. The model takes a sentence or a statement and generates the corresponding sentence in raw audio format.



**Figure 1.7:** A schematic of the circuit board used to generate the silverbox dataset [34]. The inputs signal  $u$  and the output signal  $y$  are sampled to produce the input and output sequences,  $x_{1:T}$  and  $y_{1:T}$ , respectively. Models trained on this data are typically unaware of this structure.

using deep learning, is the encoder-decoder structure [33]. In this structure the encoder operates on the input and the decoder generates the output of the full model conditioned on the output of the encoder. For this example, we consider the input as a string of one-hot encoded characters, and the output is raw audio.

#### 1.5.4 Example: Circuit modeling

Our final example is a problem from system identification. The data we want to model is recorded output from a system excited with a known input signal. The problem is what we call a synced sequence-to-sequence problem as inputs in the future cannot affect the output at an earlier time step. The trained model captures the behavior of the physical system and can be used for prediction of the output given an input signal, to control the output to a specific range or for insights about the physical system.

In the silverbox dataset [34] the input and the output are sampled from the system at  $\sim 600$  Hz over approximately 60 s with Gaussian white noise as input. The system is a circuit board created to simulate a nonlinear harmonic oscillator, see Figure 1.7. This dataset is significantly smaller compared to the other problems mentioned here, which in turn means that the model structures need to have stronger priors and stronger regularization to avoid overfitting.

## 1.6 Contribution

This thesis is located in the gap between the traditional field of system identification and the newer field of deep learning. The first papers are trying to reduce the distance between the fields by using deep learning on typical system identification problems (Paper I and Paper II). In addition to this it contributes to understanding how we can translate the success of deep learning for image modeling to the field of sequential modeling using similar constructions (Paper III). This is done while keeping the notation and interpretation probabilistic, which leads to the final contribution that is on the topic of assessing the validity of probabilistic models (Paper IV).

### Paper I

C. Andersson, N. Wahlström, and T. B. Schön. “Data-Driven Impulse Response Regularization via Deep Learning.” In: *Proceedings of 18th IFAC Symposium on System Identification (SYSID)*. Stockholm, Sweden, 2018, pp. 1–6

**Summary:** In this paper we present a novel idea on how to construct a prior for the finite impulse response of a system through deep learning. This prior is then used to regularize an estimator of the finite impulse response. The main idea draws inspiration from impulse response estimations regularized with Gaussian processes. In previous works, the Gaussian process is used as a prior for the parameters in the impulse response estimation. In this paper, we learn a prior that we model with deep learning instead of using a Gaussian process.

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

### Paper II

C. Andersson, A. L. Ribeiro, K. Tiels, N. Wahlström, and T. B. Schön. “Deep convolutional networks in system identification.” In: *Proceedings of the IEEE 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 connect deep learning and system identification and experiments with known good models from deep learning by applying 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 by Koen Tiels. He is also the author of that part of the paper. The rest of the writing was jointly made by me, Antônio Riberio, Niklas Wahlström. The implementation was done by me and Antônio.

## Paper III

C. R. Andersson, N. Wahlström, and T. B. Schön. “Learning deep autoregressive models for hierarchical data.” In: *Proceedings of 19th IFAC Symposium on System Identification (SYSID)*. Padova, Italy (online), 2021

**Summary:** Deep learning has shown great results on image data with convolutional neural networks, which utilize invariances and other prior beliefs we have of the data. This work aims to translate these ideas to sequential data and at the same time combine it with hierarchical variational autoencoders, which has also shown great results on modeling image data.

**Contribution:** The general idea for the paper originated from me and was refined with the help of Niklas Wahlström and Thomas Schön. The majority of the implementation and writing was made by me.

## Paper IV

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

**Summary:** A calibrated model has nothing to do with the accuracy of the model – instead it is a measure on how accurately it predicts the probability that it makes the correct prediction. This paper covers calibration for classification models, how to formalize the concept and how to evaluate calibration, or rather, how the current standard of evaluation calibration is insufficient.

**Contribution:** The idea behind this paper grew from a discussion between me, David Widmann and Jozas Vaicenavicius. The formalized notion and the formulation thereof are products of Jozas and David while the theorems are results from discussions in between the three of us. The implementation was done by me and David.

## 1.7 Related but not included work

[A] A. H. Ribeiro et al. “Automatic diagnosis of the 12-lead ECG using a deep neural network.” In: *Nature Communications* 11.1 (2020), p. 1760

- [B] L. Ljung, C. Andersson, K. Tiels, and T. B. Schön. “Deep Learning and System Identification.” In: *IFAC-PapersOnLine* 53.2 (2020), pp. 1175–1181
- [C] C. Andersson. *Deep learning applied to system identification : A probabilistic approach*. Licentiate thesis, Uppsala University, Sweden. 2019

## 1.8 Thesis outline

The four concepts (sequential models, deep learning, generative models and hierarchical structured data) presented in this introduction outlines to a large degree this thesis. Chapter 2 dives deeper into sequential problems and models and formalizes the learning problem in this setting. Chapter 3 gives a brief background on deep learning from a probabilistic view point as well as an introduction to different generative models. Chapter 4 crowns the creation by combining Chapter 2 and Chapter 3. It introduces deep learning for sequential problems and deep generative sequential models. The final chapter, Chapter 5, summarizes the thesis with concluding remarks and potential future work.





# 2

## Learning in a sequential setting

*“Arithmancy is predicting the future using numbers.”*

– J. K. Rowling

A machine learning problem can be characterized by a *model*, some data,  $\mathcal{D}$ , and a goal defined by a performance metric. The model is in turn parameterized with some model parameters,  $\theta$ , that are tuned to the data. Throughout this thesis we use a mix of frequentistic and Bayesian methodology. We only consider maximum likelihood/a posteriori estimates and we use  $p_\theta(\mathcal{D})$  to denote the model. However, we argue in terms of the prior, and aleatoric and epistemic uncertainty. See for example Wilson et al. [21] or Lakshminarayanan et al. [41] for the basic ideas behind this viewpoint. One unconventional notion we use is that we do not differentiate between the prior that acts on the parameters of a model and the (dirac) prior that is implied by a specific model choice.

The data is a set of  $n$  samples,  $\mathcal{D} = \{y^{(i)}\}_{i=1}^n$ , from what is called the data generating distribution,  $\pi(y)$ , in the unsupervised case. If the problem has some dedicated input each sample is instead a pair,  $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^n$ , from the joint distribution  $\pi(x, y)$ . The goal in the machine learning problem is to perform well – according to some metric – on some unseen data by fitting the model to  $\mathcal{D}$ . In addition to the model parameters there are hyperparameters, i.e., aspects of the parameterization of the model or fitting of the model that are not easily tuned. This typically gives rise to two linked optimization problems, one for the model parameters and one for the hyperparameters.

This chapter covers both how to find hyperparameters and how to solve the model parameter optimization problem in a general probabilistic machine learning case. Building on this we introduce the basic model assumptions that follow in the domain of sequential data. The coming chapters further define these concepts in the specific case of deep learning.

## 2.1 Training a model

To train or optimize a model means to fit the parameters in a way that improves the performance according to some training objective or loss,  $\mathcal{L}$ , given the data. Following the frequentist methodology, this corresponds to finding the point estimate that minimizes the loss function,

$$\hat{\theta} = \arg \min_{\theta} \mathcal{L}(\mathcal{D}, \theta). \quad (2.1)$$

There exists a multitude of different such losses, e.g. cross-entropy and squared loss, which can be motivated from a probabilistic parameterization via maximum likelihood or more specifically negative log-likelihood loss,

$$\mathcal{L}(\mathcal{D}, \theta) = -\log p_{\theta}(\mathcal{D}). \quad (2.2)$$

Under the assumption that the datapoints are sampled independently, this can further be simplified to

$$\mathcal{L}(\mathcal{D}, \theta) = - \sum_{(x,y) \in \mathcal{D}} \log p_{\theta}(x, y). \quad (2.3)$$

There are also losses that are not directly motivated from a probabilistic point of view, e.g. the loss used in generative adversarial networks [42].

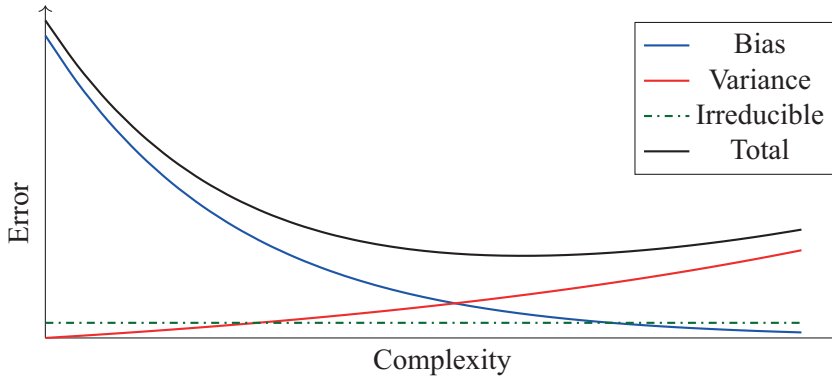
An important note here is that the loss does not necessarily have to correspond to the evaluation metric that we want to optimize in the end, but rather just acts as a proxy for the metric. A prominent example of this is a typical classification problem, which is trained by minimizing the cross-entropy, but evaluated with prediction accuracy.

The optimization technique to solve the minimization problem depends to a large degree on the problem at hand. For deep learning problems, which this thesis covers, the by far most common optimization techniques are based on first order gradient descent.

## 2.2 Bias-variance trade-off

The trained machine learning model is evaluated on a part of the dataset, which is set aside during the training. The performance on this (test) dataset should reflect how the model performs in the "wild" when deployed. This is known as the test error and can be decomposed into three parts: the irreducible error, which is inherent to the problem due to measurement noise; the bias error, which stems from a too simplistic model; and the variance error, which originates from the fact that the training data is stochastic by nature, i.e. sampled from the data generating distribution.

The variance error can be reduced in two ways, either by modifying the model, i.e. decreasing the complexity of the model, or by increasing the size



**Figure 2.1:** A schematic picture of the decomposition of the performance error into its three components; bias error, variance error, and irreducible error. The best possible model is where the total error is minimized.

of the training data. However, decreasing the model complexity increases the bias, since it makes the model less flexible. For a fixed dataset size there is thus a sweet spot, where the total error is minimized, see Figure 2.1.

Besides varying the model complexity, it is also possible to alter the cost function in the purpose of reducing the variance of the trained model. The most common alteration is to add a *regularization* cost. This extra cost, which (typically) only depends on the parameters, is simply added to the cost in Equation (2.1). We do not go into more detail on regularization, but for the reader who is interested we refer to Hastie et al. [43] and Bishop [11].

We purposely left out the effect of a prior in the above argument. Although a prior has the same general effect as decreasing the complexity of the model, the relationship is a bit more complex. First and foremost, the increase of the bias error from the prior will dissipate with more training data. Thus, introducing a prior is typically superior to decreasing the complexity as it scales better with more data. Secondly, not all priors are equal, but can be more or less adapted for the specific problem. Thus, if a prior more accurately encodes prior information it can reduce the variance error more without affecting the bias as much. An example of this is the use of an inductive bias in the model [21].

Other terms that often come up in conjunction with the bias-variance trade-off are underfitting and overfitting. Underfitting corresponds to the case when the model is too restrictive and the bias error dominates the variance error. On the other hand, if a model is overfitting it fits the data too well, the training error is typically a lot smaller than the test error and the model does not generalize to unseen data. In this regime the variance error dominates the bias error.

In recent years there have been some developments subjugating these results with some doubt by observing something that is denoted *double decent* for overparameterized models, i.e. models with more parameters than data points

in the training dataset. Belkin et al. [44] observed that the test error does not have the typical U-shape. Instead, it starts to decrease again in the overparameterized region. The implications on model design from these results is still a subject for research, but it can potentially explain why some overparameterized models (e.g. deep learning-based models) perform very well.

## 2.3 Training hyperparameters

Training the hyperparameters,  $\nu$ , of a model typically requires two steps: an inner optimization for the model parameters,  $\theta$ , and an outer optimization for the hyperparameters. There are primarily two reasons for this: the hyperparameters are often not differentiable, e.g. the number of model parameters; and the hyperparameters are used to optimize the bias-variance trade-off. To estimate the test error we use two disjoint datasets to fit hyperparameters: a training dataset to fit model parameters, and a validation dataset,  $\mathcal{D}_V$ , that acts as test dataset. The complete optimization can thus be described as

$$\hat{\nu} = \arg \min_{\nu} \mathcal{L}_V(\mathcal{D}_V, \hat{\theta}(\nu)), \quad (2.4)$$

where  $\hat{\theta}(\nu)$  are the parameters from Equation (2.1). Note that  $\mathcal{L}_V$  does not need to be the same as the training loss,  $\mathcal{L}$ , and can be nondifferentiable.

To solve this optimization problem there are a couple of different options. For problems where solving the model parameter optimization is fast, one can typically use simple gridding or sampling to find a good estimate. As the model parameter optimization becomes more expensive, these brute force approaches can turn out to be too expensive for an exhaustive search and more advanced sampling schemes, e.g. Bayesian optimization [45, 46], could be a solution. One should note though that these more advanced schemes are no silver bullet and manual tuning, i.e. trial and error, can be at least as good. As a matter of fact, to a certain degree it is the intuition behind the hyperparameter tuning that characterizes a good machine learning engineer, and is by some considered more art than science.

## 2.4 Sequential models

This and the following sections continue the description of the sequential problems introduced in Chapter 1, and how the sequential nature of the data affect the models we use.

In the case of sequential classification and regression problems the input is sequential, i.e.,  $x = x_{1:T}$ , where  $T$  is the length of the sequence, and the output is nonsequential output,  $y$ . Thus, we use  $p_{\theta}(y|x_{1:T})$  to denote the model for these problems. For problems where the output is sequential, i.e.  $y = y_{1:T}$ ,

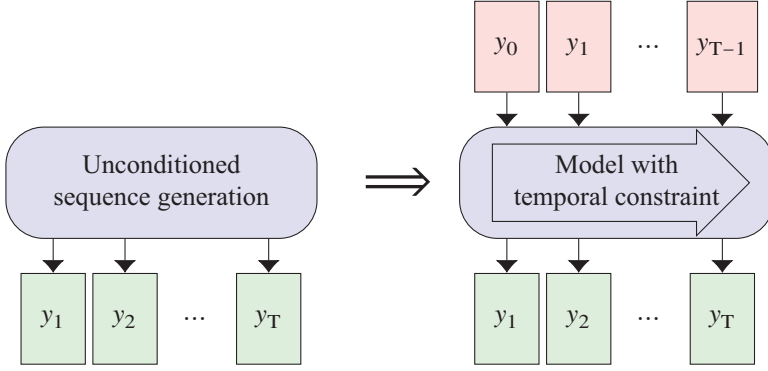
**Table 2.1:** A partition of sequential problems and how they can be factorized.

Problem	Factorization
Sequence classification and regression	$p(y   x_{1:T})$
Unconditioned sequence generation	$\prod_t p(y_t   y_{1:t-1})$
Conditioned sequence generation	$\prod_t p(y_t   y_{1:t-1}, x)$
Sequence-to-sequence	$\prod_t p(y_t   y_{1:t-1}, x_{1:M})$
Synced sequence-to-sequence	$\prod_t p(y_t   y_{1:t-1}, x_{1:t})$

we have four different scenarios. In the first two cases, where the input is not sequential or simply nonexistent, we have an unsupervised type of problem. We denote the model used as  $p_\theta(y_{1:T} | x)$  or  $p_\theta(y_{1:T})$ . Secondly, if the input is sequential, but given separately from the output, we have a sequence-to-sequence problem. In this case the input does not need to have the same length as the output sequence and we denote these models as  $p_\theta(y_{1:T} | x_{1:M})$  where  $M$  is the length of the input sequence. Finally there is the synced sequence-to-sequence problem, where the input and the output sequence are in sync. The input here is typically of the same length as the output, i.e.  $x = x_{1:T}$ . However, it can also have some fraction of the output length as they can have different sampling rates. With a slight misuse of notation we use  $p_\theta(y_{1:T} | x_{1:T})$  to denote the model, noting that the output cannot depend on future inputs, i.e. causally constrained.

A very common way to model problems with sequential output is to further factorize the models sequentially. One should note though that this is not necessarily required for good performance, and we give a brief background on models that do not use this factorization in Section 3.2. The sequential factorization can be expressed as,  $p_\theta(y_{1:T} | x_{1:M}) = \prod_t p_\theta(y_t | y_{1:t-1}, x_{1:M})$  for sequence-to-sequence problems and  $p_\theta(y_{1:T} | x_{1:T}) = \prod_t p_\theta(y_t | y_{1:t-1}, x_{1:t})$  for synced sequence-to-sequence problems. The other sequential output problems can be derived from these, see Table 2.1 for an overview of these factorizations. An effect of this factorization is that we can let the previous output be an explicit input to the model as long as the model uphold the causal constraint. Thus, it is possible to model an unconditioned modeling problem with a similar model as for a synced sequence-to-sequence problem, see Figure 2.2.

The sequential factorization gives rise to autoregressive models and state-space models, two popular models in machine learning as well as physical modeling. Both of the models approximate the parameters,  $\theta$ , in  $p_\theta(y_t | y_{1:t-1}, x_{1:t})$  to either be the same at each time step, i.e. time invariant models, or have a known dependence on time, i.e. time varying models.



**Figure 2.2:** It is possible to realize an unconditioned sequence generation problem with a causally constrained model that takes the previous output as the input. If we also have exogenous inputs,  $x_{1:T}$ , these can simply be concatenated to the previous output.

## 2.5 State-space models

A state-space model (SSM) introduces a Markovian latent state,  $z_t$ , to capture the history of the past observations and inputs up to a time point  $t$ , see Figure 2.3. We can formulate this in the synced sequence-to-sequence problem formulation as

$$p_\theta(y_t | y_{1:t-1}, x_{1:t}) = \int p_\theta(y_t | z_t) p_\theta(z_t | z_{t-1}, x_t) p_\theta(z_{t-1} | y_{1:t-1}, x_{1:t-1}) dz_{t-1:t}, \quad (2.5)$$

which also translates naturally to the other sequential output problems. The two probabilistic densities, the *transition* density  $p(z_t | z_{t-1}, x_t)$  and the *emission* or *observation* density  $p_\theta(y_t | z_t)$  are sufficient to fully describe a state-space model. The full state-space model is given by

$$p_\theta(y_{1:T} | x_{1:T}) = \int \prod_{t=1}^T p_\theta(y_t | z_t) p_\theta(z_t | z_{t-1}, x_t) dz_{1:T}. \quad (2.6)$$

Coupled with the state-space formulation is the filtering problem, i.e. finding the posterior distribution  $p_\theta(z_{t-1} | y_{1:t-1}, x_{1:t-1})$ , which is needed to infer the state given the previous observations. This inference problem can, depending on the model structure, be difficult to solve. A key property of the state-space model is that the inference can be done recursively,

$$p_\theta(z_t | y_{1:t}, x_{1:t}) \propto \int p_\theta(y_t | z_t) p_\theta(z_t | z_{t-1}, x_t) p_\theta(z_{t-1} | y_{1:t-1}, x_{1:t-1}) dz_{t-1}, \quad (2.7)$$

which is the working stone in filtering algorithms. A related distribution is the smoothing distribution which finds the posterior of the state given all observations, i.e.  $p_\theta(z_t | y_{1:T}, x_{1:T})$ .

The linear Gaussian state-space model (LGSSM) and the hidden Markov model (HMM) are traditionally two of the most prominent state-space mod-

els [11]. Their linear structure makes them advantageous when solving the filtering problem, which in these cases can be done analytically via the Kalman filter [47] and dynamic programming [48] for the LGSSM and the HMM, respectively. In the more general case both the transition density as well as the observation density can be arbitrary complex and even multimodal. Such models, called nonlinear state-space models, are in contrast more difficult to do inference on and one usually has to resort to sampling methods such as sequential Monte Carlo [49].

The state-space formulation can be natural for some physical problems due to known physical properties of the data generating process. However, it should be pointed out that for most problems the introduction of a latent state to capture the history of the observations is an approximation, at least as long as the dimension of the state stays finite. Even so, it can still be a useful approximation that introduces an effective prior for many sequential problems.

**Example 2.1** (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 Gaussian. With an exogenous input signal we have,*

$$p(z_t | z_{t-1}, x_t) = \mathcal{N}(z_t | Az_{t-1} + Bx_t, \Sigma_T), \quad (2.8)$$

while the observation distribution is defined as,

$$p(y_t | z_t) = \mathcal{N}(y_t | Cz_t, \Sigma_O), \quad (2.9)$$

which equivalently can be written on state-space form as,

$$\begin{aligned} z_t &= Az_{t-1} + Bx_t + \epsilon_t, \\ y_t &= Cz_t + v_t, \end{aligned} \quad (2.10)$$

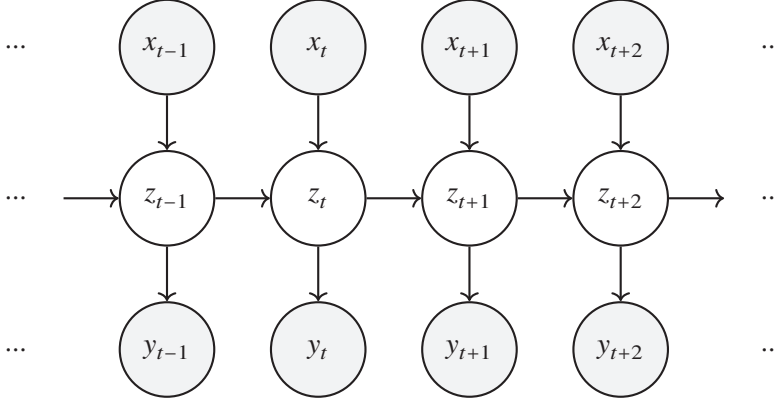
where  $\epsilon_t$  and  $v_t$  are i.i.d. and distributed as  $\mathcal{N}(0, \Sigma_T)$  and  $\mathcal{N}(0, \Sigma_O)$ , respectively.

## 2.6 Autoregressive models

An alternative way to approximate the factors in the sequentially factorized model is to truncate the history as

$$p_\theta(y_t | y_{1:t-1}, x_{1:t}) = p_\theta(y_t | y_{t-k:t-1}, x_{t-k:t}), \quad (2.11)$$

where  $k$  is the truncation horizon – also known as the *receptive field*. Similar to the state-space model, this model is an approximation to the one-step ahead prediction density, as long as this horizon stays finite. In this case, data with negative time indices is usually replaced with zeros, referred to as zero



**Figure 2.3:** A typical description of the state-space model with  $z_{1:T}$  as latent variables and  $y_{1:T}$  as observed variables with exogenous input  $x_{1:T}$  in the synced sequence-to-sequence problem formulation. Observed (or observable) variables are gray ( $x_{1:T}$  and  $y_{1:T}$ ) and unobserved variables are white ( $z_{1:T}$ ).

padding. In this thesis, we call this an autoregressive model to keep the notation concise with other work, even though a more proper terminology is a nonlinear autoregressive model with exogenous input (NARX).

**Example 2.2** (Finite impulse response). *A special case of the autoregressive model with an exogenous input signal is to additionally assume that*

$$p(y_{1:T} | x_{1:T}) = \prod_{t=1}^T p(y_t | x_{t-k:t}), \quad (2.12)$$

*i.e., the current output is conditionally independent of past outputs. This can be a good assumption for a system if there is no or little process noise and the measurement noise is close to white. If we model  $p(y_t | x_{t-k:t})$  with a Gaussian distribution and let the mean be a linear function of  $x_{t-k:t}$  we arrive at the so-called finite impulse response model.*

Training an autoregressive model can be difficult, especially in the case where the truncation horizon is large since the number of parameters grows linearly with  $k$ . In this thesis we present two alternative ways of handling this. In Paper I we regularize a linear model with a long horizon to prevent overfitting and in Paper III we use a model where the number of parameters only grows logarithmically with the horizon.



## 2.7 Example: Circuit modeling

Let us return to the circuit modeling example introduced in Section 1.5.4. This example is typical for system identification and the time invariant factorization we introduced in this chapter, i.e.  $\prod_t p(y_t | y_{1:t-1}, x_{1:t})$ . We fit this model to the data by using the negative log-likelihood as the loss function and get the optimization problem

$$\hat{\theta} = \arg \min_{\theta} \mathcal{L}(\mathcal{D}, \theta) = \arg \min_{\theta} \sum_{(x,y) \in \mathcal{D}} \sum_{t=1}^T -\log p_{\theta}(y_t | y_{1:t-1}, x_{1:t}). \quad (2.13)$$

This problem formulation is also known as the *one-step-ahead prediction* problem since what is optimized is the one-step-ahead prediction density.

With this we can substitute our modeling choice, e.g. the linear autoregressive model

$$p_{\theta}(y_t | y_{1:t-1}, x_{1:t}) \approx p_{\theta}(y_t | x_{t-k:t}, y_{t-k:t-1}) = \mathcal{N}\left(y_t | \theta \begin{pmatrix} x_{t-k:t} \\ y_{t-k:t-1} \end{pmatrix}, \sigma^2\right), \quad (2.14)$$

into Equation (2.13) and solve the corresponding optimization problem. In the specific case of a linear autoregressive model this optimization problem corresponds to a linear regression/least squares problem.



# 3

## Deep probabilistic models

*“You take the red pill – you stay in wonderland, and I show you how deep the rabbit hole goes.”*

– Morpheus, The Matrix

This chapter introduces three concepts that are essential throughout this thesis, namely deep learning, neural networks and probabilistic models. In the interest of keeping this thesis to the point, we assume that the reader has a basic understanding of neural networks. This includes fully connected neural networks, convolutional neural networks and core concepts around training and evaluating the network, such as back-propagating gradient, minibatch gradient decent and early stopping. Section 3.1 gives a short background on the subject and introduces the notation used in this thesis. For a more detailed background on neural networks we refer to Goodfellow et al. [46].

A probabilistic model in this context corresponds to models where we model the output with a probability distribution. Additionally, we can introduce some latent variables,  $z$ , that can help modeling complex distributions. Through the probabilistic framework it is possible to derive many of the standard losses, e.g. the least square and cross entropy loss. However, the general framework makes it possible to also use other assumptions that do not translate to any common loss, e.g. mixture of Gaussians. In addition to this, the framework also increases the interpretability of the model and the output as more information is conveyed through a distribution than a single predicted value (see Section 3.7).

Deep probabilistic models are orthogonal to and should not be confused with what is known as Bayesian neural networks [50], where all the parameters of the neural network are interpreted in a Bayesian way. Neither should it be confused with probabilistic neural networks [51], which are an alternative to neural networks altogether, building on nonparametric function estimators.

### 3.1 Neural networks

A neural network can be explained as a function approximator that is optimized with stochastic gradient descent. However, compared to other function

parameterizations, neural networks can be made very flexible while still generalizing well, even when overparameterized [44]. There is still a lot of theory missing related to how and why they perform as good as they do, even if some progress have been made in recent years [16, 17]. An alternative way of framing the success of neural networks is to say that the network structure provides a good inductive bias that acts as a prior, guiding the neural network to solutions that generalize well.

A neural network consists of several layers of linear parametric functions alternated with nonlinear activation functions. The most basic linear function is

$$h_i = \sum_c W_{i,c} x_c + b_i, \quad (3.1)$$

where  $W$  and  $b$  are the parameters,  $x$  is the input to the layer and  $h$  is the transformed input. When this is combined with an activation function we get a fully connected (FC) layer.

The second most common linear function is the convolutional layer. This parameterization provides a useful inductive bias when working on data that is translational invariant, e.g. time-invariant signals and images. The translation invariance can possibly range over several dimensions. For example, a time-invariant signal has a one-dimensional invariance, an image has a two-dimensional invariance and a video has a three-dimensional invariance. In addition to the invariant dimensions there is usually one extra dimension for features. For example, image data is typically represented with three dimensions: two for the invariant spatial dimensions plus one dimension for features. Thus, the data has dimension, Width  $\times$  Height  $\times$  Feature channels, for which we denote a convolution as,

$$h_{i,j,m} = \sum_{l=-\lfloor w_w/2 \rfloor}^{-\lfloor w_w/2 \rfloor + w_w} \sum_{k=-\lfloor w_h/2 \rfloor}^{-\lfloor w_h/2 \rfloor + w_h} \sum_c W_{m,l,k,c} x_{i+l,j+k,c} + b_m, \quad (3.2)$$

where  $\lfloor \cdot \rfloor$  is a shorthand notation for the floor function. Furthermore,  $W$  is called a kernel or filter and  $w_w$  and  $w_h$  denotes the range of this kernel in the Width and Height dimension, respectively. The kernel and the offset,  $b$ , are the parameters of this function.

Following the probabilistic notation of this thesis the output of a neural network is a distribution, or more accurately, the parameters of a parametric distribution. Thus, for a simple supervised model we use the notation

$$p_\theta(y|x) = \mathcal{P}(y|NN_\theta(x)), \quad (3.3)$$

where  $NN_\theta(x)$  is a neural network with parameters  $\theta$  that output features for the parametric distribution,  $\mathcal{P}$ . These features can for example correspond to: the mean and covariance of a Gaussian distribution; the means, covariances

and weights of a Gaussian mixture distribution; or the logits of a categorical distribution.

An important aspect of neural networks is to properly initialize the parameters and normalize/standardize the input and output data. Today, the most popular deep learning frameworks, Tensorflow [52] and PyTorch [53], handles the initialization of the parameters behind the scenes. However, the data normalization is left for the user to implement.

## 3.2 Deep generative models

With the introduction of deep learning, generative models have conquered new domains with unprecedented results. Some of the most notable results are in the domains of natural images [12] and text [7], but also in music [54] and speech [8], to name a few. This section encompasses some of the most influential models and the techniques behind them.

A core property of generative models is the ability to generate different results for the same input and express correlations in the output. This requires stochasticity to be included into the model, either by recursively feeding observation noise back into the model or by introducing latent variables in the model. The recursive method can be motivated by using the sequential factorization introduced in Chapter 2,

$$p(y) = \prod_{k=1}^K p(y_k | y_{1:k-1}), \quad (3.4)$$

where  $K$  is the dimension of the output. However, this method is not limited to sequential data, but can be used for any kind of data. Some examples of this method are, PixelRNN [55] and PixelCNN++ [56] that generates images by interpreting an image as a sequence of pixels, GPT-3 [7] that generates text, and Wavenet [8] that generates speech. Generating new samples with this model structure is typically computationally heavy, since each generated output needs to be propagated through a large portion of the network for each new prediction. The advantage is that the training can be done in parallel and, since there is no explicit noise, the models are relatively straightforward to train and implement using likelihood-based losses. A possible extension is to also include latent variables as done in e.g. STORN [57], SRNN [58] and VRNN [59], which is covered further in Section 4.5.

For models with latent variables there are two major directions: likelihood-based methods such as variational autoencoders (VAE), normalizing flows, and diffusion models; and nonlikelihood-based models such as generative adversarial networks (GANs). For the likelihood-based methods a major problem to overcome is how to efficiently compute the likelihood,

$$p_{\theta}(y) = \int p_{\theta}(y, z) dz. \quad (3.5)$$

The VAE solves this problem by introducing an approximate posterior distribution  $q_\theta(z; y)$ , which through importance sampling estimates the likelihood in a tractable way. In Section 1.2 we introduced the NVAE [12], which sampled images of high fidelity, but VAEs can also be used as a component in other models to create flexible and interpretable distributions, e.g. STORN, SRNN, and VRNN mentioned in the previous paragraph. In Section 3.3 VAEs are explained in more detail with more examples.

An alternative to introducing an approximate posterior is to restrict the function that connects the latent variables and the observations to be bijective. Normalizing flows [60] use a neural network,  $y = f_\theta(z)$ , as the connecting function with the restriction that every layer needs to be bijective. Under this constraint the likelihood can be evaluated exactly without any approximations as,

$$p_\theta(y) = p(z = f_\theta^{-1}(y)) \det \left| \frac{\partial f_\theta^{-1}}{\partial y} \right|. \quad (3.6)$$

This method proposes a very flexible parameterization that still allows for exact evaluation. However, this bijective constraint also limits the model. It is uncertain how much of the generalizing performance of neural networks that is preserved with this constraint and it is also difficult to train the model with high dimensional data. Yet, Glow [61] showed that it is indeed possible to generate images of faces with high fidelity. It is also possible to combine the normalizing flow with an autoregressive factorization to form so-called autoregressive flows [62, 63]. In this case,  $y$  and  $z$  are represented as sequences instead, i.e.  $y = y_{1:T}$  and  $z = z_{1:T}$ . The advantage of this factorization is that the function only needs to be bijective for the functions that connect  $z_t$  and  $y_t$ , enabling a larger class of functions to be used. Masked autoregressive flows [62] and inverse autoregressive flows [63] are two prominent examples of this structure. For a more thorough background on normalizing flows and applications of it, see e.g. Papamakarios et al. [64].

Even though diffusion models [65] are likelihood-based, the data likelihood is not used to train these models. Instead, they are trained to reverse one step of a predefined noising process. This noising process progressively corrupts the original sample,  $y_0$ , with Gaussian noise over  $K$  steps,  $y_1, \dots, y_K$  – after which all information on the original data is gone. The distribution of the final corrupted data is then unit Gaussian, i.e.  $p(y_K) = \mathcal{N}(y_K | 0, 1)$ . The goal of the model is to reverse the corruption, i.e. denoise the data, by approximating  $p(y_{k-1} | y_k)$  with a neural network. A sample from the model is produced by recursively applying  $p_\theta(y_{k-1} | y_k)$ , starting from a unit Gaussian sample for  $y_K$ . Diffusion models are among the strongest deep generative models and can produce high fidelity samples of both images [65] and speech [66]. However, due to the recursive structure of the generative process they tend to be computationally heavy and thus slow at generating samples.

Generative adversarial networks [42] (GANs) avoid the problem with evaluating the likelihood altogether by using a completely different loss function.

**Table 3.1:** A selection of generative models and some of their properties. All models, except normalizing flows, are free to use any parameterization of the neural networks.

Model	Likelihood-based	Latent variables	Fast sampling	Unconstrained neural networks
Autoregressive	✓	-	-	✓
VAE	✓	✓	✓	✓
Normalizing flow	✓	✓	✓	-
Diffusion	✓	✓	-	✓
GAN	-	✓	✓	✓

In addition to a generative model,  $G_\theta(z)$ , a discriminative model,  $D_\phi(y)$  is proposed for the sole purpose of distinguishing generated data from true data, where  $\phi$  are the parameters of this network. The two networks are trained in conjunction in a game theory setup, where both networks try to outperform the other. Thus, the two networks are trained in alternation, keeping the other one fix, with the following losses

$$\mathcal{L}_D(\phi) = -\mathbb{E}_{y \in \mathcal{D}} D_\phi(y) + \mathbb{E}_{z \sim p(z)} D_\phi(G_\theta(z)), \quad (3.7)$$

$$\mathcal{L}_G(\theta) = -\mathbb{E}_{z \sim p(z)} D_\phi(G_\theta(z)). \quad (3.8)$$

Note that Equation (3.7) and Equation (3.8) are not optimized until convergence – they are optimized until the other loss starts to deteriorate. A problem with this formulation is that there is no objective measure on how well the generative model is performing, which makes it very difficult to compare different GANs. The training also tends to be very unstable and can easily get stuck in local minima. Even with these constraints, GANs (together with diffusion models) are one of the qualitatively best performing generative models to this date, e.g. StyleGAN [67, 68]. Some theoretical progress has been made to make the training more stable. One example of that is Wasserstein GAN [69], where the GAN is reformulated using a Wasserstein distance to improve the stability. In Table 3.1 we try to summarize all the different generative models and some of the distinguishing properties.

### 3.3 Variational autoencoders

The variational autoencoder [70, 71] is briefly introduced in Section 3.2 as a probabilistic model with latent variables. These latent variables benefit the model in primarily two ways. Firstly, they enable more complex and possibly multimodal distributions, which is often required for good performance on high dimensional and correlated outputs. Secondly, the learned latent variables can help us analyze the data by categorizing and clustering it.

However, as discussed in Section 3.2, latent variable models entail a problem with estimating the marginal likelihood,

$$p_\theta(y) = \int p_\theta(y|z)p(z)dz = \mathbb{E}_{z \sim p(z)} p_\theta(y|z), \quad (3.9)$$

where  $p(z)$  is the prior of the latent variables. This integral is in general intractable, especially in the case where  $p_\theta(y|z)$  is modeled with an arbitrary neural network.

One way to estimate this integral is to use a Monte Carlo method. However, estimating Equation (3.9) naively requires a significant number of samples (to reduce the variance of the estimate), making it infeasible in practice. To reduce the variance of the estimate and thus effectively reducing the number of Monte Carlo samples required, importance sampling with a proposal,  $q(z)$ , can be used. The best such proposal, i.e. the minimal variance proposal, is the true posterior,  $p_\theta(z|y)$ , [72]. However, it is as intractable as computing the sought likelihood since it requires the marginal data distribution. Variational inference [11] offers an approximation to this problem by introducing an approximate posterior found by minimizing the KL-divergence to the true posterior under some constraints. For example, with a Gaussian constraint this posterior is expressed as

$$\hat{q}(z; y) = \arg \min_{\tilde{q} \in \mathcal{N}} \text{KL}(\tilde{q}(z) \| p_\theta(z|y)), \quad (3.10)$$

where  $\tilde{q} \in \mathcal{N}$  denotes that  $\tilde{q}$  is from the class of Gaussian distributions. Here we have emphasized the fact that this approximate posterior,  $\hat{q}(z; y)$ , will depend on  $y$ . This KL-divergence can be rewritten in terms of the likelihood and the so-called evidence lower bound (ELBO), sometimes also known as the variational lower bound, as,

$$\text{KL}(q(z) \| p_\theta(z|y)) = \underbrace{-\mathbb{E}_{z \sim q(z)} \log p_\theta(y|z) + \text{KL}(q(z) \| p(z))}_{-\text{ELBO}(y)} + \log p_\theta(y). \quad (3.11)$$

Using this, we note that the optimization in Equation (3.10) can be done by maximizing the ELBO instead (since  $\log p_\theta(y)$  is independent of  $q(z)$ ). The advantage of the ELBO is that it is typically is more tractable to compute than the original KL-divergence and depending on the constraints used for  $q(z)$ , one or both of the terms in the ELBO can be analytically tractable. For example, in the case where the prior is a Gaussian, the posterior can be constrained to a Gaussian and yields a KL-term that is analytically tractable. The ELBO can also be used to derive a lower bound on the likelihood, hence its name, as

$$\begin{aligned} \log p_\theta(y) &= \mathbb{E}_{z \sim q(z)} \log p_\theta(y|z) - \text{KL}(q(z) \| p(z)) + \text{KL}(q(z) \| p_\theta(z|y)) \\ &\geq \mathbb{E}_{z \sim q(z)} \log p_\theta(y|z) - \text{KL}(q(z) \| p(z)), \end{aligned} \quad (3.12)$$



since the KL-divergence is always positive. Thus, depending on the accuracy of the approximative posterior, the ELBO can also be used to approximate, or at least give a lower bound on, the likelihood.

The variational autoencoder does one additional approximation. The optimization problem in Equation (3.10) is generally too computationally heavy to compute during the training of a neural network model (even with the ELBO). Instead the variational autoencoder approximates the solution directly with a neural network, that outputs the parameters of a parameterized distribution, i.e.  $\hat{q}(z; y) \approx q_\theta(z; y)$ . Finally, since both the approximate posterior and (a lower bound on) the log-likelihood can be optimized by maximizing the ELBO, it is possible to simultaneously optimize them by maximizing

$$\mathcal{L}(y, \theta) = \mathbb{E}_{z \sim q_\theta(z; y)} \log p_\theta(y|z) - \text{KL}(q_\theta(z; y) \| p(z)). \quad (3.13)$$

This parameterization of the optimization problem is known as amortized variational inference [73]. It distinguishes itself from ordinary variational inference in that the approximate posterior is not necessarily the best posterior under the given constraint. Thus, if a good approximate posterior is what is sought, further tuning of the approximate posterior is advised. The most common assumption for variational autoencoders is to choose a Gaussian distribution for the prior and the approximative posterior even though other assumptions exists, e.g. Bernoulli distributed [74].

### 3.3.1 REINFORCE and the reparameterization trick

Optimizing Equation (3.13) still requires us to estimate the expectation taken over the approximate posterior, even if the KL-term can be calculated analytically. This is done with Monte Carlo sampling, usually only with a single sample. However, since the optimization is done with gradient descent, we need to take some extra measures to be able to propagate the gradient through these samples. There are at least two different approaches to rewrite the gradient of this expectation, called REINFORCE [75] and the reparameterization trick [71].

REINFORCE was originally developed for the reinforcement learning problem, hence the name. The idea behind it is to change the distribution over which we take the derivative and consider a fix point instead as,

$$\begin{aligned} \frac{\partial}{\partial \theta} \mathbb{E}_{z \sim q_\theta(z; y)} \log p_\theta(y|z) \Big|_{\theta=\theta'} &= \mathbb{E}_{z \sim q_{\theta'}(z; y)} \frac{\partial}{\partial \theta} \log p_\theta(y|z) \frac{q_\theta(z; y)}{q_{\theta'}(z; y)} \\ &= \mathbb{E}_{z \sim q_{\theta'}(z; y)} \frac{\partial}{\partial \theta} \log p_\theta(y|z) \Big|_{\theta=\theta'} \\ &\quad + \mathbb{E}_{z \sim q_{\theta'}(z; y)} \log p_{\theta'}(y|z) \frac{\partial}{\partial \theta} \log q_\theta(z; y) \Big|_{\theta=\theta'}, \end{aligned} \quad (3.14)$$

where  $\theta'$  is the point where the derivative is evaluated. This has the advantage that it can be used to differentiate samples from any distribution.

The reparameterization trick, on the other hand, is limited in the distributions it can be applied to. It can only be used on distributions that can be expressed as a differential function of a sample from a fixed base distribution,  $p(\epsilon)$ . One large class of distributions with this property is the location-scale family, including the Gaussian distribution. For this case the expectation can be rewritten as,

$$\frac{\partial}{\partial \theta} \mathbb{E}_{q_{\theta}(z;y)} \log p_{\theta}(y|z) = \mathbb{E}_{p(\epsilon)} \frac{\partial}{\partial \theta} \log p_{\theta}(y | \mu_{\theta}(y) + \sigma_{\theta}(y)\epsilon), \quad (3.15)$$

where  $p(\epsilon) = \mathcal{N}(0, 1)$  is independent of  $\theta$ , and  $\mu_{\theta}(y)$  and  $\sigma_{\theta}(y)$  are some arbitrary functions. Both methods can be used to estimate this expectation and it is possible to prove that none of the methods have any fundamental advantage over the other in general [76]. However, the reparameterization trick produces lower variance estimates in the case of variational autoencoders empirically.

### 3.3.2 Posterior collapse

One problem that can arise when training variational autoencoders is posterior collapse. This corresponds to a state where the approximate posterior closely matches the prior and almost no information is encoded in the latent space, i.e., the KL-divergence term in Equation (3.13) is close to zero and dominated by the likelihood term. This state is in practice difficult to escape, effectively making it a local minimum. To enforce that all the modeling capabilities are used, a couple of different methods have been proposed. Here we present two of these methods.

*KL annealing* [77, 78] is one such method, where the core idea is to discount the KL-term in the loss. This is done by multiplying the KL-term with a discount factor,  $0 \leq \gamma \leq 1$ , which allows more information to be encoded for the same amount of loss. This is then annealed linearly from 0 towards 1 during the initial phase of training. After this phase the KL-term has a strong enough gradient to avoid the pitfall.

*Free bits* [63] is an alternative method with the same purpose. However, instead of setting a scaling discount factor, this method gives each KL-unit an amount of "free bits" and the gradient is only passed through the KL-units if it is greater than the set amount of free bits. This ensures that each KL-unit (the KL divergence in each dimension of the latent space) encodes a minimum amount of information. The threshold of free bits can also be annealed during training if the threshold was unnecessary high.

### 3.4 Hierarchical variational autoencoders

We previously discussed how deep learning can use the hierarchical structure of the data by extracting hierarchical features (see Section 3.1). In a similar fashion, such a hierarchical mapping can also benefit a variational autoencoder realized by stacking several layers of latent variables. For example, a model with three such layers yields a *generative distribution* as

$$p(x) = \int p(x | z^{(1)}, z^{(2)}, z^{(3)}) p(z^{(1)} | z^{(2)}, z^{(3)}) p(z^{(2)} | z^{(3)}) p(z^{(3)}) dz^{(1)} dz^{(2)} dz^{(3)}. \quad (3.16)$$

This makes it possible to express very complex distributions using only simple components.

Even though stacked latent variables were proposed already in the original submission on variational autoencoders [71], it only had a minor effect on the performance of the model. The ladder variational autoencoder, proposed by Sønderby et al. [78], was the first model that showed some real performance gain from this structure, where the key improvement was how to model the approximate posterior. The idea can be summarized as factorizing the posterior similarly as for the generative distribution,

$$q(z^{(1)}, z^{(2)}, z^{(3)}; y) = p(z^{(1)} | z^{(2)}, z^{(3)}; y) p(z^{(2)} | z^{(3)}; y) p(z^{(3)}; y). \quad (3.17)$$

We call this model structure a *hierarchical variational autoencoder* and it has since its introduction been used as a backbone in a number of follow up models, e.g. ResNet VAE [63], BIVA [79] and NVAE [12].

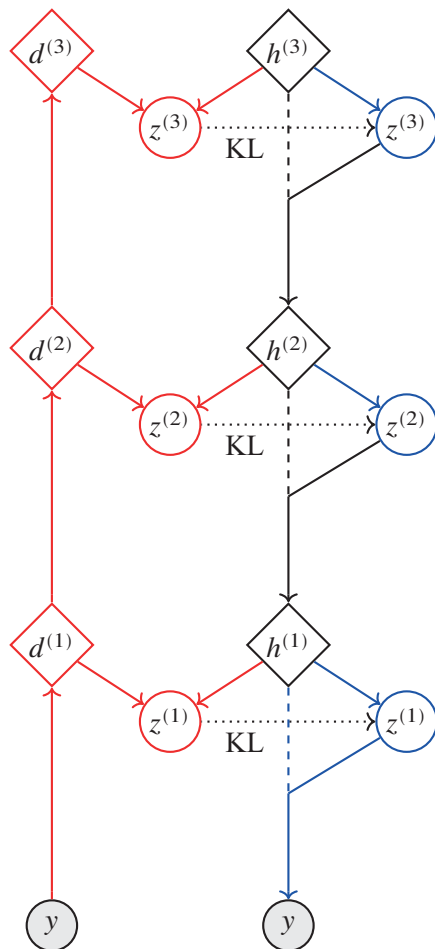
The hierarchical variational autoencoder can be described with a bottom-up and a top-down hierarchy of features (see Figure 3.1). The bottom-up hierarchy of features,  $d^{(l)}$ , are extracted from the observations  $y$ , and each successive layer of features depends on the previous layer,  $d^{(l-1)}$ . The top-down features,  $h^{(l)}$ , are structured in a similar fashion, but in the opposite direction. The posterior distribution uses both the bottom-up and the top-down features, while the generative distribution only uses the top-down features. Using these features we can rewrite Equations (3.16) and (3.17) as

$$p(x) = \int p(x | h^{(0)}) \prod_{l=1}^2 p(z^{(l)} | h^{(l)}) p(z^{(3)}) dz^{(1)} dz^{(2)} dz^{(3)} \quad (3.18)$$

$$q(z^{(1)}, z^{(2)}, z^{(3)}; y) = \prod_{l=1}^2 q(z^{(l)} | h^{(l)}, d^{(l)}) q(z^{(3)} | d^{(3)}) \quad (3.19)$$

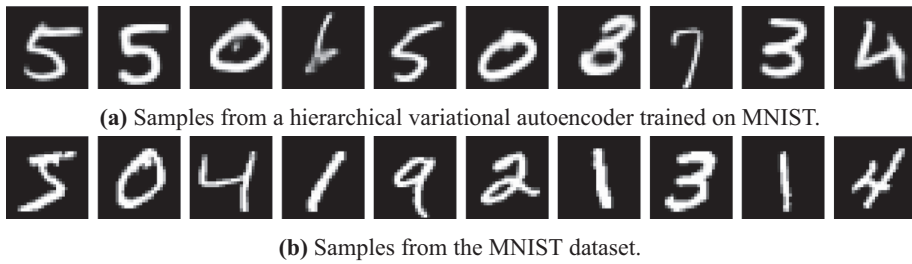
where  $h^{(l)} = \text{NN}(z^{(l+1)}, h^{(l+1)})$ ,  $d^{(l)} = \text{NN}(d^{(l-1)})$ ,  $h^{(4)} = \{\}$  and  $d^{(0)} = y$ .

The hierarchical variational autoencoder works especially well for high dimensional observations. One explanation for this is that the model structure encourages the model to explain away some of the lower-level features, so that modeling of the higher-level features can be done more efficiently. An alternative interpretation of this is that a model with latent variables at different



**Figure 3.1:** A picture of a hierarchical VAE, with the bottom-up network extracting features,  $d^{(*)}$ , on the left, and the top-down network with intermediate features,  $h^{(*)}$ , for generating an observation on the right. The bottom-up is exclusively used for the **approximate posterior** while the top-down network is partially **shared** between the approximate posterior and the **generative distribution**. When estimating the likelihood of an observation the latent variables,  $z^{(*)}$ , are sampled from the approximate posterior and when sampling from the model they are sampled from the generative distribution. The vertical dashed line acts as a residual connection and was not present in the original ladder VAE [78], but it was used in the ResNet VAE [63]. In this thesis we use circles to denote stochastic variables/features and diamonds or squares to denote deterministic variables/features.

hierarchical levels makes it possible to capture both complex concepts that involve many observations and simpler concepts that only model dependencies between a few observations. In Paper III and in Section 4.7 we introduce a hierarchical variational autoencoder adopted to sequential data.



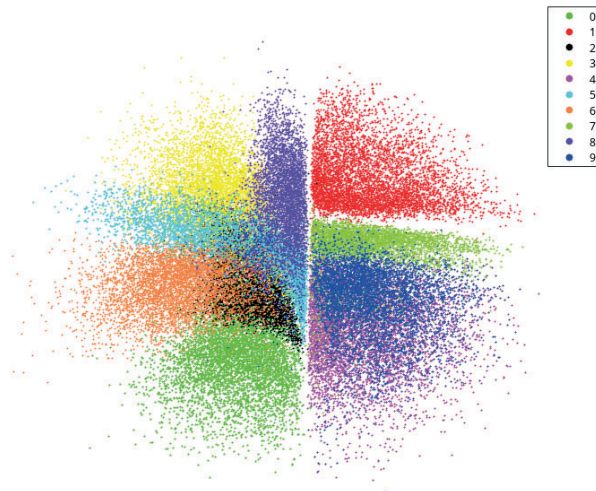
**Figure 3.2:** A comparison of images from the MNIST dataset and images generated with a simple hierarchical variational autoencoder trained on the same images.

One use case of a hierarchical variational autoencoder is to model images. Figure 3.2 depicts samples from a model trained on the popular handwritten number dataset MNIST [80]. Besides generating numbers similar to those in the dataset, the latent structure can help us further analyze the data and what the model have captured from the data. One example of this is to use the approximate posterior to investigate the latent space of the generative distribution. Figure 3.3 visualizes the mean of this approximate posterior for the topmost latent variables, color coded with the label that corresponds to each image. Even though no information on the classes is given to the model, it clusters the data somewhat and even separates out the number one completely. An alternative it to use the trained model to distinguish between in-distribution data, i.e. data from the MNIST dataset, and out-of-distribution data, i.e. nonnumeric symbols [81].

### 3.5 Transfer Learning

Neural networks are, relative to other machine learning methods, data hungry, i.e., they require a larger amount of collected and annotated data before they reach a comparable performance level. An idea to jump-start a neural network is to use transfer learning [82]. That is, take a neural network trained on a "similar" problem with a larger dataset (source), adapt the pretrained model to the specific problem, and finetune it with the available annotated data (target). A model trained through this approach can use an annotated dataset that is substantially smaller while still having a performance that is comparable to that of a model trained solely on a large annotated dataset.

A surprising observation with transfer learning is that the source problem and dataset used for the pretrained network can differ significantly from the target problem. A telling example of this is the work by Esteva et al. [19], which utilized a model trained on an image classification task on ImageNet [83], a large dataset containing natural images of primarily animals and also some inanimate objects such as building and clothes, with a total of 1000 classes.



**Figure 3.3:** A visualization of latent space of a hierarchical variational autoencoder trained on MNIST. The visualized space is the mean of the approximate posterior for the topmost latent variables (2 dimensions) for the test data, color coded with the label that each example corresponds to. Even if the model structure does not make any clustering assumptions, the model clusters the data in groups that partly matches the labels.

The target dataset on the other hand is approximately 130 000 closeup photographic and dermoscopic (microscopic) images of skin lesions labeled with 757 distinct classes. Even though ImageNet does not contain any images on humans, let alone images on skin, this pretraining boosts the finetuned model to achieve accuracy on par with expert dermatologists. One should note that a key aspect in this example, observed in follow up work by Raghu et al. [84], is that the model is overparameterized compared to the target dataset and the benefit of transfer learning is reduced with smaller models.

In the previous example both the source and the target problem were of a classification nature. However, transfer learning can be used also when the structure of the source problem and the target problem do not match. An example of this is BERT [85], which is trained to predict one or several words that have been masked in a sentence. The trained model can then be finetuned on a range of different classification tasks in natural language processing with good generalizing capabilities [85]. A key point is that the source problem does not require any labeled data and is thus unsupervised. The abundance of unsupervised data that is available today have resulted in very large models that are trained solely for the purpose of transfer learning. Recently, such models were dubbed foundational models [18] for the unique role that they can be shared across a large range of downstream tasks and the potential that brings to

the future of deep learning. The example on cough classification demonstrates transfer learning with one such model called wav2vec 2.0 [86].

## 3.6 Example: Cough classification

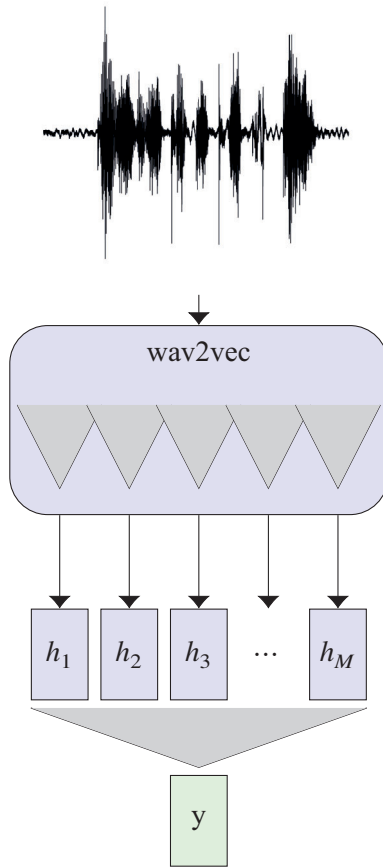
In this section we continue on the cough classification example introduced in Section 1.5.1. Speech recognition is a very data intense division of machine learning requiring thousands of hours of annotated data to reach an acceptable performance [86]. Detecting speech and cough in audio are conceptually similar tasks – even if speech is expressed with a significantly wider range of different phonemes, the fundamental features are still to a large degree common.

One idea to reduce the amount of training data required to achieve an acceptable performance in cough classification is to use a network pretrained on speech data. One such model is wav2vec 2.0 [86], which is trained unsupervised on a huge dataset of audiobooks. This model is primarily intended for speech analysis, but the features extracted can still prove useful for detecting coughs (c.f. the example in Section 3.5). More specifically wav2vec transforms the one second recorded snippets into a sequence of features, where each feature corresponds to 20 ms of sound. This sequence of features is then aggregated by calculating the mean of the sequence (see Section 4.11 for other ideas on how to do this aggregation). The aggregate is then fed to a simple classification network (see Figure 3.4). With the pretrained wav2vec model and a single layer classification network it is possible to train a model that detects 22 seconds of cough (only missing a single cough) and 11 one second sequences false positives in a one-hour recording (precision 95.6 % and specificity 99.7 % treating all one-second sequences independently). This is achieved with less than 3 hours of annotated data, where only approximately 10 minutes is used for training<sup>1</sup>.

## 3.7 Calibration

The performance of a probabilistic model cannot reliably be expressed by a single quantity, e.g. accuracy, but is in reality multifaceted [87]. Consider a binary classification problem and two different models that both have a 90% accuracy. However, the models are probabilistic, meaning that they output a distribution for the prediction. In this example, the first model always predicts its selected class with 99% confidence, while the other model predicts with 90% confidence. Which of the models should you choose? The first model predicts that it will be correct 99% of the time. However, comparing this to the measured accuracy we see that it does not match. In reality it was only correct

<sup>1</sup>This is ongoing work, which is yet to be published.



**Figure 3.4:** A model structure to distinguish if the input, here represented by the waveform in top bottom of the figure, corresponds to a coughing sound or not, represented by  $y$  in the bottom of the figure. `wav2vec` [86], a pretrained feature extractor, consolidates the raw audio (depicted with the gray triangles in `wav2vec`) into a sequence of features,  $h_{1:M}$ , which are aggregated and used for the prediction of the label,  $y$ .



90% of the time, so the model was overconfident. The other model however correctly predicted its accuracy, i.e., its confidence matches the accuracy. This makes the second model more *reliable*.

Why should we care if a model is reliable and not only about its accuracy? Firstly, a model that knows what it knows and, perhaps more importantly, knows what it does not know is more trustworthy for someone that wants to interpret the model output. It makes it easier to identify problematic input and reason about the cost associated with potential misclassifications [88]. It also enables further probabilistic reasoning by combining the predicted distribution with other probabilistic objects. For example, we can reason about the cost of gathering more data, as well as, which data to gather to make the model more confident and more accurate [89].

The example we used so far is a bit unrealistic. A model does typically not output only one specific confidence – instead it depends on the available evidence in the sample that it is trying to predict for. Evidence can be the presence of certain features useful for the classification, or the quality of these features. For example, when classifying the race of a dog from an image, some features like the color of the nose might be occluded if the image is taken from behind, or if the image has too low resolution it might not be possible to figure out the texture of the fur. Thus, for samples with less available evidence, the model might produce less confident predictions, while other samples with more available evidence will produce more confident predictions. However, since the ground truth for each sample is only a single observation it is difficult to know if the prediction is reliable or not. The idea in calibration is to condition on a prediction distribution instead of conditioning on the specific input. In other words, what is the average prediction when the model predicts a distribution,  $\hat{p}$ . In a mathematical notation we denote this with a *calibration function*,

$$r(\hat{p}) = \mathbb{P}(y | p_\theta(y|x) = \hat{p}), \quad (3.20)$$

where  $\mathbb{P}$  is the average distribution of  $y$  under  $x, y \sim \pi(x, y)$ .

For a *calibrated* [90, 91] model this calibration function should be equal to the identity function. It is also possible to condition on other aspects of the model or the data, e.g., the model should be calibrated only for the most probable class. By doing so we can construct other calibration functions that corresponds to different aspects of calibration. In fact, the most common calibration function conditions only on the maximum class, i.e.

$$r_{\max}(\hat{p}) = \mathbb{P}\left(y = \hat{c} \mid \max_c p_\theta(y = c|x) = \hat{p}\right), \quad (3.21)$$

where  $\hat{c} = \arg \max_c p_\theta(y = c|x)$ . This is the calibration function used in the well-known expected calibration error [92] (ECE).

The performance of a model in the aspect of calibration is typically presented either as a reliability diagram, see Figure 3.5, or as a quantitative measure, such as expected calibration error. The expected calibration error is a

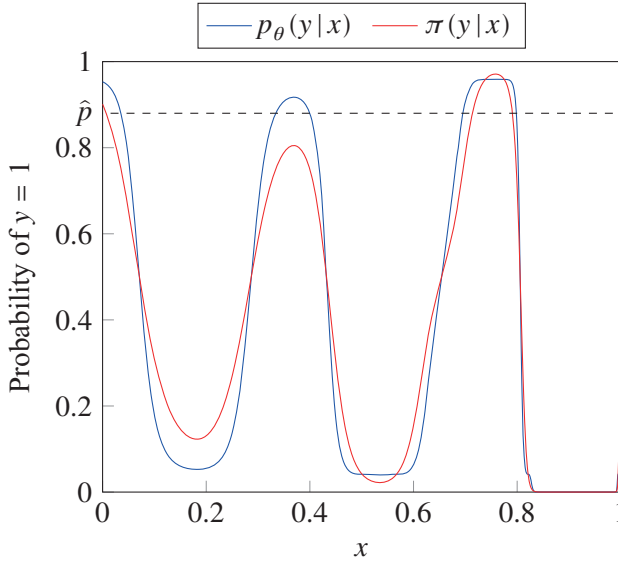
measure on the distance between the maximum class calibration function and the identity function defined as,

$$\int |r_{\max}(\hat{p}) - \hat{p}| \mathbb{P}(\hat{p}) d\hat{p}. \quad (3.22)$$

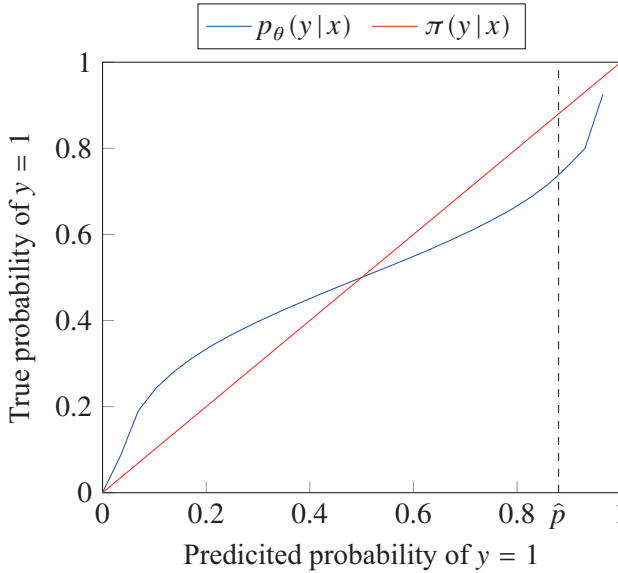
This integral and its factors,  $\mathbb{P}(\hat{p})$  and  $r_{\max}(\hat{p})$ , are in this case estimated by binning over the predicted probability.

It is important to note that accuracy and calibration are two orthogonal measures. A model that only predicts the marginal class distribution for every input is perfectly calibrated, but has quite poor accuracy. On the other hand, a model that always predicts correctly with only 51% confidence is underconfident and thus uncalibrated. There also exist some techniques that can be used to make an uncalibrated model more calibrated without affecting accuracy, e.g. temperature scaling [92].

Calibration of models is often overlooked and most users consider more accurate models as more desirable than perfectly calibrated models. However, it has been shown that deep learning models in particular are overconfident when it comes to the predictions that they make [92], which makes calibration something that should not be ignored – especially in safety critical applications such as skin cancer diagnosis and autonomous driving. In Paper IV different aspects and pitfalls of calibration are explained in more detail.



(a) The prediction of the model (blue) and the true probability (red) for different values of the input,  $x$ .



(b) A reliability diagram for the model in (a).

**Figure 3.5:** In (a), the prediction for a model  $p_\theta(y|x)$  on a binary classification problem is visualized, where the model is trained on samples from the true model  $\pi(y|x)$ . In (b), the associated reliability diagram is visualized for both the true model and the trained model. For a perfectly calibrated model the blue line in should be equal to the red line (b). Note that it is not required for them to be equal in (a) for this to hold.  $\hat{p}$  corresponds to the same confidence level in both subfigures and are here simply used for visualization purposes.



# 4

## Sequential deep learning models

*“A wizard is never late, nor is he early, he arrives precisely when he means to.”*

– Gandalf the Grey

In Chapter 2 we introduced sequential learning in a very general framework. In this chapter we focus on how we can utilize deep learning in combination with this framework. We introduce the analogues of autoregressive models and state-space models in deep learning. Furthermore, we discuss these models in relation to a deep hierarchical prior and how they can be combined with latent variables for improved generative properties. The beginning of this chapter focuses on the case of sequential output, that is, either the synced sequence-to-sequence problem (Section 1.5.4) or sequence generation problem (Section 1.5.2). Sequential input problems are then covered in Section 4.11 and Section 4.12.

### 4.1 Recurrent neural networks

One of the core models in sequential deep learning is the recurrent neural network [93] (RNN), originally proposed in 1986. The structure was derived from the earlier Hopfield networks [94] and the Ising model [95], which are popular biologically inspired models for memory and recall. In essence, the recurrent neural network is a state-space model (see Section 2.5), but the sequential state is, for the most part, considered to be deterministic. To explicitly specify that the state is purely deterministic we denote it with,  $h_{1:T}$ , whereas a stochastic state,  $z_{1:T}$ , can include both a deterministic and a stochastic part. In addition to this we also define an input,  $\tilde{x}_{1:T}$ , and an output,  $\tilde{y}_{1:T}$ , to the recurrent neural network. It is important to note that these inputs and outputs do not have to coincide with the input and output of the problem structure – it can be slightly altered for the task and model at hand, hence the  $\sim$ -notation. In this setting, a recurrent neural network is defined by two functions, a transition function,  $f_\theta$ , that recursively updates a state subjugated by the input, and an output function,  $g_\theta$ , that defines the output given a state. This is expressed

as

$$h_t = f_\theta(h_{t-1}, \tilde{x}_t), \quad (4.1a)$$

$$\tilde{y}_t = g_\theta(h_t), \quad (4.1b)$$

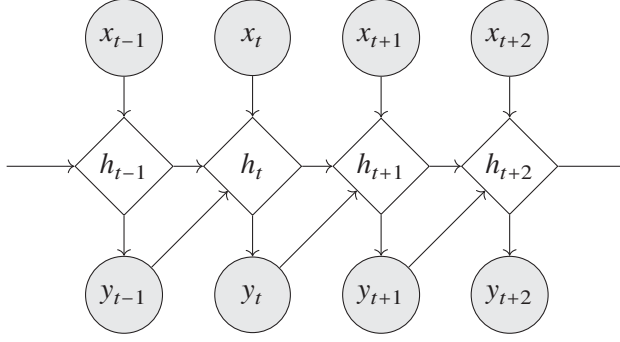
where  $f_\theta$  and  $g_\theta$  are parameterized with neural networks and  $h_{t-1}$  is calculated recursively starting with some initial state,  $h_0$ . One of the most basic models on this form is the Elman network [96], which implements  $f_\theta$  and  $g_\theta$  as single layered neural networks.

With this basic notation of a recurrent neural network we can compare it with the state-space model from Section 2.5 in the synced sequence-to-sequence problem setting. For the state-space model, the one-step ahead prediction function is defined via the latent state as,

$$p_\theta(y_t | y_{1:t-1}, x_{1:t}) = \int p_\theta(y_t | z_t) p_\theta(z_t | z_{t-1}, x_t) p_\theta(z_{t-1} | y_{1:t-1}, x_{1:t-1}) dz_{t-1:t}. \quad (4.2)$$

Since we in Equation (4.1) consider the state as deterministic, i.e.  $z_t = h_t$  and  $p_\theta(h_t | h_{t-1}, \tilde{x}_t) = \delta(h_t = f_\theta(h_{t-1}, \tilde{x}_t))$ , the integral in Equation (4.2) disappears. Similarly, the integral in Equation (2.7) disappears and with it the need for solving the (intractable) filtering problem. However, this also poses an issue. Without the filtering, the state is deterministically given by the input to the model alone. If we in this case let the input to the recurrent neural network be equal to the input in problem structure, i.e.  $\tilde{x}_{1:T} = x_{1:T}$ , the model predictions are independent of earlier observations and the model effectively turns into  $\prod_t p_\theta(y_t | x_{1:t})$ . For systems with a substantial amount of process noise or with a close to chaotic behavior, this assumption becomes very strict and the predictive performance suffers. To avoid this issue the input to a recurrent neural network often also includes the most recent output, i.e.  $\tilde{x}_t = (x_t, y_{t-1})$  in the synced sequence-to-sequence problem setting and  $\tilde{x}_t = y_{t-1}$  in the sequence generation problem setting, see Figure 4.1.

One alternative approach to motivate the additional input is to change how we interpret the states,  $h_t$ , of the recurrent neural network. Instead of interpreting the states as deterministic variants of some latent variables,  $z_t$ , we interpret them as statistics of the predictive distribution of the latent states,  $p_\theta(z_t | x_{1:t}, y_{1:t-1})$ . These statistics can then be viewed as deterministic states which are updated recursively. For example, these statistics could be the mean and variance, which are sufficient statistics for the Gaussian distribution, and be updated recursively, as in the Kalman filter [47]. However,  $h_t$  may encode other arbitrary abstract statistics of the predictive distribution of the latent variables. Following this approach, instead of parameterizing the transition density,  $p_\theta(z_t | z_{t-1})$  directly, it is parameterized as a function that takes the prior of the state,  $p_\theta(z_t | x_{1:t}, y_{1:t-1})$ , solves the inference problem associated with observing an output and transitions it to the prior of the next state,



**Figure 4.1:** A schematic illustration of the recurrent neural network. The output is fed back into the model by letting  $h_t$  depend on  $y_{t-1}$ . This allows the state to be deterministic, while still being able to model complex sequences.

that is

$$\underbrace{p_\theta(z_t | x_{1:t}, y_{1:t-1})}_{h_t} = \int p_\theta(y_{t-1} | z_{t-1}) p_\theta(z_t | z_{t-1}) \underbrace{p_\theta(z_{t-1} | x_{1:t-1}, y_{1:t-2})}_{h_{t-1}} dz_{t-1} \approx f_\theta(h_{t-1}, x_t, y_{t-1}), \quad (4.3)$$

c.f. with the one-step ahead prediction with a Kalman filter. Finally, the observation density can similarly be expressed as

$$p_\theta(y_t | x_{1:t}, y_{1:t-1}) = \int p_\theta(y_t | z_t) \underbrace{p_\theta(z_t | x_{1:t}, y_{1:t-1})}_{h_t} dz_t \approx \mathcal{P}(y_t | g_\theta(h_t)), \quad (4.4)$$

where the output of  $g_\theta(h_t)$  are some features parameterizing the output distribution.

This feedback of the output into the model is not without drawbacks. During training the true observation is most commonly used for this feedback, also known as *teacher forcing* [46]. However, when generating (i.e. simulating) from the model, these true observations are (of course) not known and the model-generated output is used as input for the next step instead. This implies a mismatch between training and generation (a distributional shift between training time and testing time), that may impose a problem [97]. An alternative idea is to instead sample an output from the model also when training. This, however, tends to be a lot harder to optimize, since this loss is significantly more stochastic. It is also possible to do a mix of the two ideas called *scheduled sampling*, as discussed by Bengio et al. [97], where each time index is either sampled or teacher forced randomly. With this approach a more robust model can be achieved.

## 4.2 Example: Circuit modeling continued

Let us again return to the circuit modeling example from Section 1.5.4 and use a recurrent neural network as the model. We assume the observation density to be

$$p_{\theta}(y_t | x_{1:t}, y_{1:t-1}) = \mathcal{N}(y_t | g_{\theta}(h_t), \exp(\sigma)^2), \quad (4.5)$$

and the dynamics are given by

$$h_t = f_{\theta}(h_{t-1}, x_t, y_{t-1}), \quad (4.6)$$

where  $\sigma$  is an independent parameter that is optimized jointly with the parameters of the recurrent neural network. In Code 1, we show how an Elman network using PyTorch [53] can be implemented and how to calculate the training loss. Note that the standard recurrent neural network in PyTorch only defines the transition function and not the observation function. The observation function is instead implemented with a linear layer that operates on each state independently. Finally, both the initial output,  $y_0$ , and the initial state,  $h_0$ , are set to zero.

In Paper II, we compare a recurrent neural network (specifically a long-short term memory model), a temporal convolutional network and a nonlinear autoregressive model with state-of-the-art methods from system identification on the SilverBox dataset.

## 4.3 Long short-term memory model

A recurrent neural network is trained using gradient descent similar to any other neural networks, also known as backpropagation through time as it similarly to the forward pass updates the gradients recursively. In other words, the gradient must be propagated in the opposite direction of all the arrows in Figure 4.1. A problem with this is that the gradient information has a tendency to increase or decay exponentially fast due to the recursive gradient calculation. In turn, this may lead to either exploding gradients that destroy any training progress made, or vanishing gradients that simply never converge [98].

The long short-term memory (LSTM) model [98] was developed to mitigate the vanishing/exploding gradient problem by allowing information to pass more freely in both the forward pass and the backpropagation. This is done by using a state update function that more closely resembles a memory model, where the state updates are governed by so-called gates. This allows information to be stored over longer periods of time without being diluted by noise or unimportant inputs, which in turn allows the gradient information to be backpropagated with less susceptibility to the vanishing/exploding gradient problem.



---

**Code 1** Training a recurrent neural network
 

---

```

import torch

# Dimensionality of the problem
channels_x = 1
channels_y = 1

# Size of the hidden state arbitrarily chosen to 3
hidden_size = 3

# The state evaluation
rnn = torch.nn.RNN(input_size=channels_x + channels_y, hidden_size=hidden_size)
# The observation function, (g)
obs_g = torch.nn.Linear(in_features=hidden_size, out_features=channels_y)
# The standard deviation in the observation density
sigma = torch.nn.Parameter(torch.zeros(1))

def train_loss(x, y):
    # x: a batch of input sequences (batch size x sequence length x channels_x)
    # y: a batch of output sequences (batch size x sequence length x channels_y)

    # add zeros first and remove last entry of y
    y_padded = torch.nn.functional.pad(y, (0, 0, 1, 0))[:, :-1, :]
    model_input = torch.cat((x, y_padded), -1)

    # This call recursively calculates the states for each sequence in the batch
    states, _ = rnn(model_input)

    # This call operates independently on each state to produce the mean prediction
    output_mean = obs_g(states)

    # Calculate the loss as the negative loss likelihood
    loss = (sigma + 0.5 * ((output_mean - y) / sigma.exp()).square()).sum()
    # or alternatively
    pred_dist = torch.distributions.normal.Normal(output_mean, sigma.exp().pow(2))
    loss = - pred_dist.log_prob(y)
    return loss.mean(0).sum()

def sample(x):
    # x: input sequences (sequence length x channels_x)
    y = []
    T = len(x)

    # The last observation is initialized to zero
    last_obs = torch.zeros(1)
    state = torch.zeros(1, 1, hidden_size)

    for t in range(T):
        input_rnn = torch.cat((x[t], last_obs), -1)
        # Fix the dimensions of the input
        input_rnn = input_rnn[None, None, :]
        _, state = rnn(input_rnn, state)
        # Normal is a normal distribution
        pred_dist = torch.distributions.normal.Normal(obs_g(state[0, 0]),
                                                    sigma.exp().pow(2))

        last_obs = pred_dist.sample()
        y.append(last_obs)
    return y
  
```

---

The LSTM has three different kinds of gates: a forget gate,  $f_t$ , that when active makes the memory cell reset; an input gate,  $i_t$ , that when active lets the network store information in the state; and an output gate,  $o_t$ , that limits what information is propagated out from the memory cells. The gates are implemented as soft binary factors using the sigmoid function, which allows the model to be fully differentiable. The state is divided into two parts: one part that is storing the actual information,  $c_t$ , and one part that corresponds to the output of the model,  $d_t$ . All in all, the network can be described with the following equations,

$$\begin{aligned} f_t &= \sigma(W_{fu}\tilde{x}_t + W_{fd}d_{t-1} + b_f), \\ i_t &= \sigma(W_{iu}\tilde{x}_t + W_{id}d_{t-1} + b_i), \\ o_t &= \sigma(W_{ou}\tilde{x}_t + W_{od}d_{t-1} + b_o), \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{cu}u_t + W_{cd}d_{t-1} + b_c), \\ d_t &= o_t \odot \tanh(c_t), \end{aligned}$$

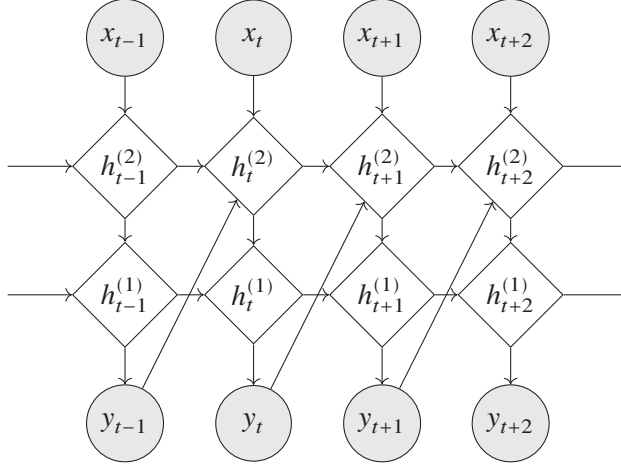
where  $W_*$  and  $b_*$  are parameters and  $\odot$  denotes elementwise multiplication.

One should note that this interpretation of the LSTM states should not be seen as an instrument to investigate how the LSTM works, but rather for how the design was motivated. In fact, since the introduction of the LSTM, several other architectures have been proposed that rely less on the interpretability of the states in favor of less complex model structures with preserved or even superior performance. One commonly used model structure in this category is the gated recurrent unit (GRU) [59], which simplifies the expressions to only have two gates and one state.

The initial area of usage of the LSTM was for language modeling, but it has in more recent years been replaced with transformer-based models in this domain (see Section 4.10). In the current deep learning arena, the LSTM and its alternatives, e.g. GRU, have stabilized as a baseline method with a wide variety of applications as new areas adopt models based on deep learning, e.g. price forecasting [99].

## 4.4 Multiscale recurrent neural networks

Despite the success of recurrent neural networks, they, by themselves, do not imply any of the deep structure that is characteristic for deep learning – most notably is the absence of hierarchical feature extraction, which made neural networks so powerful. One idea to approach such a deep structure is to stack several recurrent neural networks on top of each other, similarly to how fully connected or convolutional networks are structured. The idea is thus to let the



**Figure 4.2:** A stacked recurrent neural network with two layers in the synced sequence-to-sequence problem formulation.

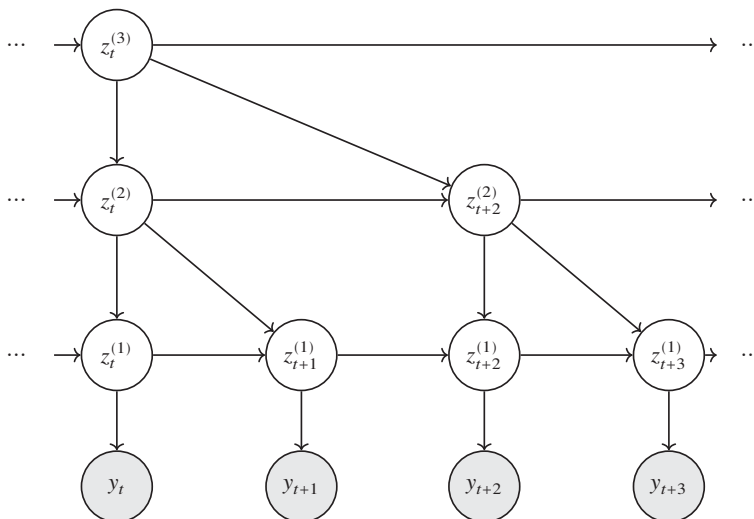
output from one recurrent network be used as an input to the next as,

$$h_{1:T}^{(1)} = \text{RNN}^{(1)}(\tilde{x}_{1:T}), \quad (4.7a)$$

$$h_{1:T}^{(2)} = \text{RNN}^{(2)}(h_{1:T}^{(1)}). \quad (4.7b)$$

This structure, called stacked recurrent neural networks (see Figure 4.2), has proven to be beneficial for modeling complex distributions, in particular for natural language problems [3].

Another idea to further adopt a deep hierarchical prior (see Section 1.4) is to similarly stack the recurrent neural networks, but let them run with different update rates, i.e., how often they update their internal state. For example, a recurrent neural network with an update rate of two skips every other state update and simply outputs the last state instead. The idea can easily be portrayed by considering a model that generates a sentence with two stacked recurrent neural networks operating with different update rates and thus at different timescales. The slower network captures features that evolve slowly in the sentence, e.g., what words that follow certain words, while the faster network captures how the words are actually spelled, i.e., what letters follow others, conditioned on the word that the faster network outputs, see e.g. hierarchical multiscale recurrent neural networks [100] and clockwork recurrent neural network [101]. In the literature, multiple different structures have been proposed to achieve this. One rough division between the structures are models that use adaptive update rates and those that use fixed update rates. Models with adaptive update rates try to adapt the interval at which the state is updated depending on the data it generates, i.e., the slower network updates only when a new word is required. In this case we also need some method to detect when an update should be made [100, 102]. Fixed update rates use, as the name suggests, a



**Figure 4.3:** A multiscale state-space model with three hierarchical levels. A model with this structure encodes the deep hierarchical prior introduced in Section 1.4.

fixed update scheme, meaning that the features in the slower network do not necessarily match words and are thus less interpretable [103, 101, 104]. The advantage is that the network is simpler and easier to train as it avoids the complexity related to discovering delimiters. Furthermore, it is straightforward to generalize this structure to even more hierarchical levels, which simplifies the scaling of the model.

A generalization of the multiscale recurrent neural networks, with fixed or adaptive interval, is a sequential hierarchical latent variable model, see Figure 4.3. We choose to visualize this model in favor of any of deterministic versions mentioned in the previous paragraph as the stochastic states avoids explicit feedback connections in the visualization. This allows us to focus on the essential properties of a multiscale state-space model without going into too much detail. One such property is that the multiscale state-space model embodies the hierarchical prior we associate with structured sequential data. An important factor to note here is that the sequential submodel on each hierarchical level does not need to be able to capture long dependencies as these relations can be expressed in the slower evolving submodels. For example, in Figure 4.3 the relationship between  $y_t$  and  $y_{t+3}$  may just as likely be expressed through their common feature  $z_t^{(3)}$ .

## 4.5 Stochastic recurrent neural networks

Even though we saw that a state-space model effectively can be represented using a deterministic recurrent neural network, it still can be beneficial to keep

some of the latent variables as stochastic. The latent states might provide an effective inductive bias, improve the interpretability, and increase the flexibility of the model. However, the inclusion of these latent states still gives rise to the filtering problem related to finding the posterior distribution over them. One idea to make this more feasible is to use a variational autoencoder as a means of solving the inference problem. This approach is the core idea of most recurrent neural networks involving latent variables.

The use of VAEs has generated a number of different stochastic adaptations of recurrent neural networks, each with slight differences in how the latent variables interact with the states and observations as well as how to parameterize the approximative posterior. The stochastic recurrent network (STORN) [57], the variational recurrent neural network, (VRNN) [59], the stochastic recurrent neural network (SRNN) [58], and Z-Forcing [105] are some examples that implement this idea. In all of these models the state is represented with one stochastic part,  $z_t$ , and one deterministic part,  $h_t$ .

In this section we only describe one of these architectures in detail, the SRNN [58], which combines a recurrent neural network with a VAE. The generative distribution,  $p$ , can be explained as having two coupled recurrent networks, one deterministic and one stochastic, see Figure 4.4a. The main takeaway is that the deterministic states are independent of the latent states, which implies that they can be computed independently of the latent variables, simplifying the inference problem significantly. In addition, it does not affect the factorization, which can be expressed as,

$$p_\theta(y_t | y_{1:t-1}, \tilde{x}_{1:t}, h_{1:t}) = \int p_\theta(y_t | z_t, h_t) p_\theta(z_t | z_{t-1}, h_t, \tilde{x}_t) p_\theta(z_{t-1} | y_{1:t-1}, \tilde{x}_{1:t-1}, h_{1:t-1}) dz_{t-1} dz_t, \quad (4.8)$$

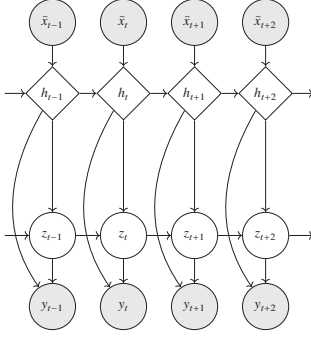
where  $h_{1:T}$  is given by the deterministic recurrent neural network. Note that we once again denote the input with  $\tilde{x}_{1:T}$  as we most often consider the past output to be concatenated with the exogenous input  $x_{1:T}$ , c.f. Section 4.1. The reason for this is that the model is not flexible enough to capture the long-term relations in the output using only the stochastic state.

The approximative posterior,  $q$ , of the SRNN approximates the smoothing distribution. This is implemented with the help of another recurrent neural network that runs backwards in time to aggregate the future information into a separate state,  $a_t$ . This gives the approximative posterior as,

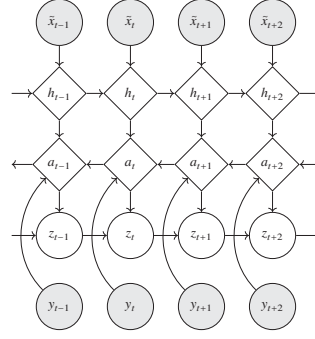
$$q_\theta(z_{1:T}; x_{1:T}, y_{1:T}) = \prod_{t=1}^T q_\theta(z_t | z_{t-1}; a_t(a_{t+1}, h_t)), \quad (4.9)$$

which is depicted in Figure 4.4b.

All of different variants on RNNs in combination with VAEs suffer from a problem similar to posterior collapse, i.e., no information is encoded in the latent space. For these networks, the problem is slightly more disguised as the model still can perform well with only the deterministic states. Free bits



(a) The generative distribution of a stochastic recurrent neural network.



(b) The approximate posterior of a stochastic recurrent neural network.

**Figure 4.4:** The stochastic recurrent neural network combines the variational autoencoder with a recurrent neural network. The approximative posterior reuses the deterministic states of the generative distribution and a recurrent neural network backwards in time to approximate the smoothing distribution.

and KL-annealing is often used also here to circumvent this. Z-forcing [105] proposed a slightly altered loss function to force the model to use the latent space for the same purpose.

## 4.6 Temporal convolutional networks

Similarly to how a recurrent neural network is the neural network version of a state-space model, there are neural network-based versions of autoregressive models. These autoregressive networks, developed in the early 90's, have been frequently used in system identification [106]. The recent deep learning development has also benefited these models, where the perhaps most prominent examples are temporal convolutional networks [107] and the Wavenet [8].

The most basic version of the models from the early era straightforwardly parameterizes the one-step ahead distribution function with a neural network. In the case that the observations are real-valued this could be implemented with a simple normal distribution, i.e.

$$p_{\theta}(y_t | y_{t-k:t-1}, x_{t-k:t}) = \mathcal{N}(y_t | f_{\theta}([y_{t-k:t-1}, x_{t-k:t}]), \sigma^2), \quad (4.10)$$

where  $f$  is a fully connected neural network and  $k$  is the truncation horizon. However, the performance of this model often falls short compared to the recurrent neural network-based ideas, especially in the case where long memory is needed as this model scales poorly with increased truncation horizon.

The temporal convolutional network (TCN) is a generalization of Equation (4.10) to a model that uses convolutions instead of fully connected layers. One interpretation of this is as stacked convolutions that extract sequen-

tial features in a hierarchical manner, very similar to how convolutional neural networks act on an image (although without any downsampling/pooling operation). However, stacking such convolutional networks must be done with precaution as the prediction at every time step is never allowed to depend on the future (the convolution must be off-center, see Figure 4.5). A convolutional neural network that fulfills this constraint is called a *temporal convolutional network* and got a lot of attention with the work of Bai et al. [107], which studied its performance in comparison with recurrent neural networks. The convolutions in this network allow for a larger degree of parameter and computational efficiency when training compared to the more basic neural network-based autoregressive model.

A temporal convolutional network has another advantage compared to Equation (4.10). The idea is to increase the receptive field, i.e. the history used for each prediction, without increasing the number of parameters as quickly. This is done by parameterizing the convolutional network with kernels, where only every  $\mu$ -th element is nonzero, called dilated convolutions, and stacking several such convolutions. With a kernel size  $k$ , i.e. the number of parameters in the kernel, a dilated convolution has a receptive field of  $k$  times  $\mu$ . Each temporal convolution layer is mathematically defined as

$$h_t = \sum_{i=0}^{k-1} \sum_c W_{i,c} \tilde{x}_{t-i\cdot\mu,c}, \quad (4.11)$$

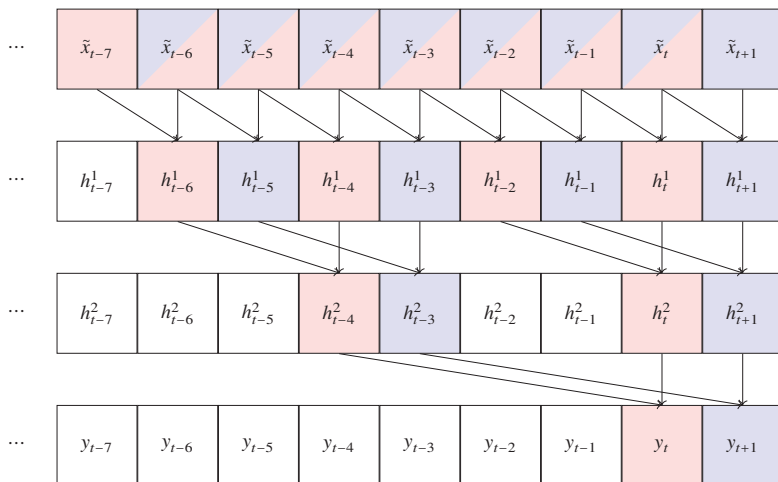
where  $W \in \mathbb{R}^{k \times F_{\text{in}} \times F_{\text{out}}}$  is the kernel, and  $F_{\text{in}}$  and  $F_{\text{out}}$  are the number of features of the input and output of the convolution, respectively. By stacking several dilated convolutions with an exponentially increasing dilation rate it is possible to create a model with exponentially long memory with only a linear increase in the number of parameters, see Figure 4.5.

One large advantage of using convolutions is that it makes it possible to use many of the tools and tricks originally developed for convolutions applied to images even in a sequential setting. For example, layernorm [108], dropout [109] and parameter initialization schemes can almost be directly transferred from image models to the temporal convolutional network. Furthermore, Bai et al. [107] showed that convolution tends to be much easier to optimize (more stable gradients) and tune than recurrent neural networks on the same problem. In addition, the convolutions are faster during training since many of the features are shared and the model is highly parallelizable.

Another well-established structure is the residual connection [110], which also easily translates to the temporal convolutional network. This residual connection can be expressed as,

$$h_{1:T} = \tilde{x}_{1:T} + \text{DilConv}(\tilde{x}_{1:T}), \quad (4.12)$$

where DilConv stands for a dilated convolution and could in principle be a series of convolutions. Wavenet [8] is one prominent example of a network



**Figure 4.5:** An illustration of a temporal convolutional network where the input,  $\tilde{x}_{1:T}$ , is operated on with 3 dilated convolutions with kernel size 2 and dilation rates of 1, 2 and 4, respectively. The full model has a receptive field of 8 – but only 6 parameters, that is 2 for each layer. The blue and red markings depict the functional dependence the output has on the input and the intermediate features.

that uses residual connections. This network models 16 kHz raw audio with a dilation rate of up to 1 024 samples and thus a receptive field in the order of 1 000's of samples.

The structure of a TCN closely resembles a convolutional neural network without down/up-sampling. This implies that it also embodies some properties of a deep hierarchical prior in that advanced features are built up by more basic features and the basic features focus on a short range correlations. However, it is not consolidating the advanced features meaning that they may contain high frequency components.

In some cases, a temporal convolution is not necessary and it is enough to operate on each time step independently, i.e.,  $h_t = f_\theta(\tilde{x}_t)$ , where  $f_\theta$  is a neural network. Such an operation can still be implemented with a convolution as long as the kernel size is set to 1. Such convolutions are also very common in image modeling, where they operate on each pixel independently with 1x1 kernels. Since many of the ideas in temporal convolutional networks are inspired from image modeling, this has led to the somewhat confusing notation that 1x1 convolutions denote time independent feature transformations also in the sequential case. For example, a feature transformation that uses the same activation functions and residual connections as Wavenet is referred to as a 1x1 Wavenet.



## 4.7 Stochastic temporal convolutional network

Similarly to how latent variables can be included in the recurrent neural network it is also possible to introduce latent variables for the temporal convolutional networks. A prominent example of this is the stochastic temporal convolutional network (STCN) [111], which uses a hierarchical variational autoencoder for each observation independently. The model can be described by starting from the one-step ahead prediction factorization, which is approximated by an autoregressive model. This is in turn modeled by introducing a stochastic latent variable,  $z_t$ , as

$$p(y_t | y_{1:t-1}) \approx \int p_\theta(y_t | y_{t-k:t-1}, z_t) p_\theta(z_t | y_{t-k:t-1}) dz. \quad (4.13)$$

This can be formalized as a hierarchical variational autoencoder by assuming a factorization of  $p_\theta(z_t | y_{t-k:t-1})$  as

$$\begin{aligned} p_\theta(z_t | y_{t-k:t-1}) &= p_\theta(z_t^{(1)} | z_t^{(2)}, y_{t-k:t-1}) p_\theta(z_t^{(2)} | z_t^{(3)}, y_{t-k:t-1}) \\ &\quad p_\theta(z_t^{(3)} | y_{t-k:t-1}), \end{aligned} \quad (4.14)$$

where  $z_t = (z_t^{(1)}, z_t^{(2)}, z_t^{(3)})$ . The number of hierarchical layers here is arbitrarily assumed to be three, but could be any number of layers.

This model uses temporal convolutional networks to extract hierarchical features,  $d_{1:T}^{(1)}, d_{1:T}^{(2)}, d_{1:T}^{(3)}$ , from the data matching the hierarchical structure of the latent variables,

$$\begin{aligned} d_{1:T}^{(1)} &= \text{TCN}(y_{1:T}), \\ d_{1:T}^{(2)} &= \text{TCN}(d_{1:T}^{(1)}), \\ d_{1:T}^{(3)} &= \text{TCN}(d_{1:T}^{(2)}). \end{aligned} \quad (4.15)$$

These features can then be used to represent,  $y_{t-k:t-1}$ , at each corresponding layer. Thus, the full generative distribution can be described as,

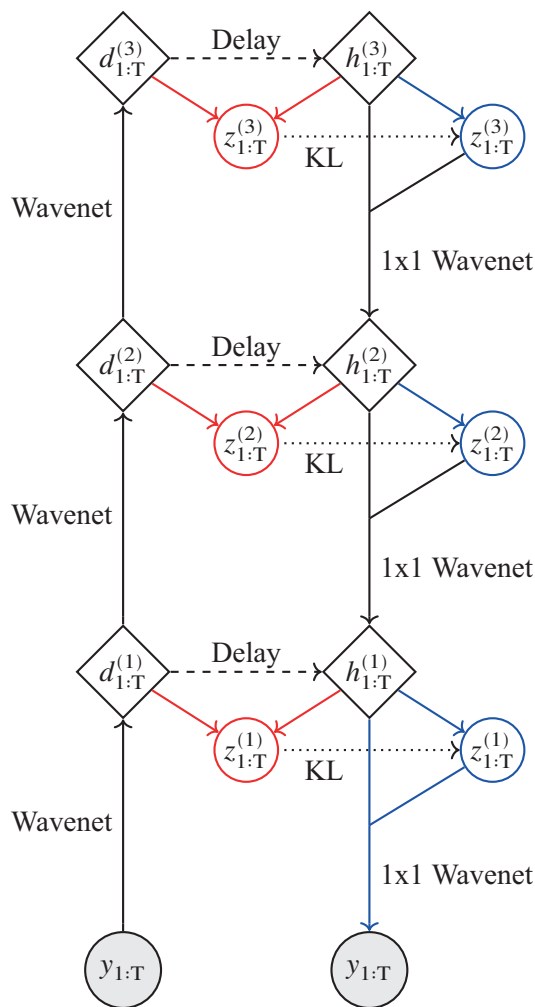
$$\begin{aligned} p(y_t | y_{1:t-1}) &\approx \int p_\theta(y_t | z_t^{(1)}) p_\theta(z_t^{(1)} | z_t^{(2)}, d_{t-1}^{(1)}) p_\theta(z_t^{(2)} | z_t^{(3)}, d_{t-1}^{(2)}) \\ &\quad p_\theta(z_t^{(3)} | d_{t-1}^{(3)}) dz_t^{(1)} dz_t^{(2)} dz_t^{(3)} \end{aligned} \quad (4.16)$$

A key assumption in the stochastic temporal convolutional network is that the approximate posterior of the variational autoencoder can also use these features. Thus, the approximate posterior is given as

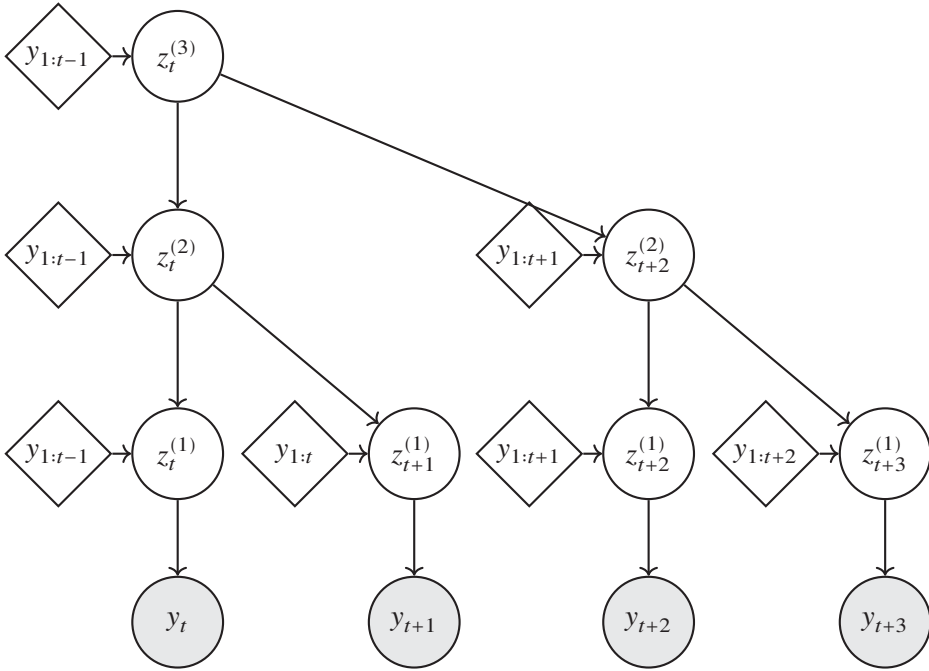
$$q_\theta(z_t; y_{1:T}) = q_\theta(z_t^{(1)} | z_t^{(2)}; d_t^{(1)}) q_\theta(z_t^{(2)} | z_t^{(3)}; d_t^{(2)}) q_\theta(z_t^{(3)}; d_t^{(3)}). \quad (4.17)$$

Note that this uses the features without the delay of one timestep compared to the generative distribution. The full model is summarized in Figure 4.6, where we introduced deterministic features,  $h_{1:T}$ , in the generative part for visual clarity.

The stochastic temporal convolutional network uses a Wavenet structure for the temporal convolutional network, i.e., it uses a residual connection. However, one should note that when this model is used on raw speech data, each observation is considered to be a frame of 200 samples instead of each sample independently.



**Figure 4.6:** A stochastic temporal convolutional network combines a hierarchical variational autoencoder with a temporal convolutional network. The **features extracted** are used for both the **approximate posterior** and the **generative distribution**.



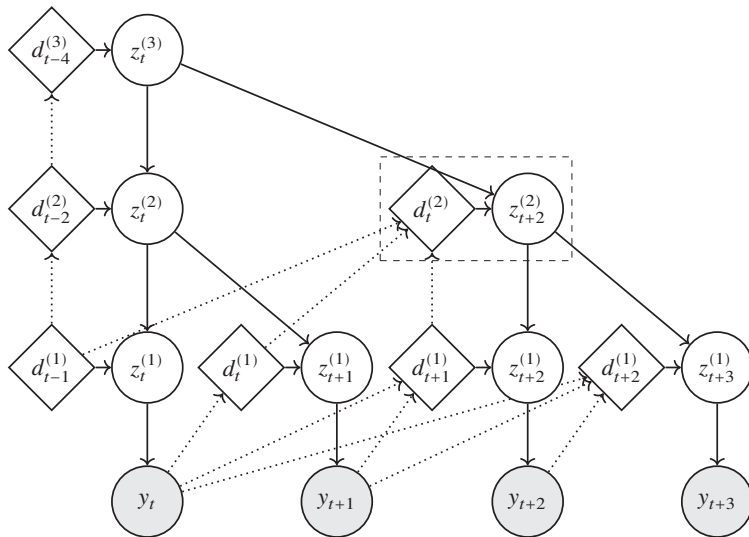
**Figure 4.7:** The overarching idea of a multiscale autoregressive model. Higher level features are shared among a larger set of observations while lower level features only capture the local neighborhood. This model structure encodes the same deep hierarchical prior as Figure 4.3.

## 4.8 Multiscale autoregressive models

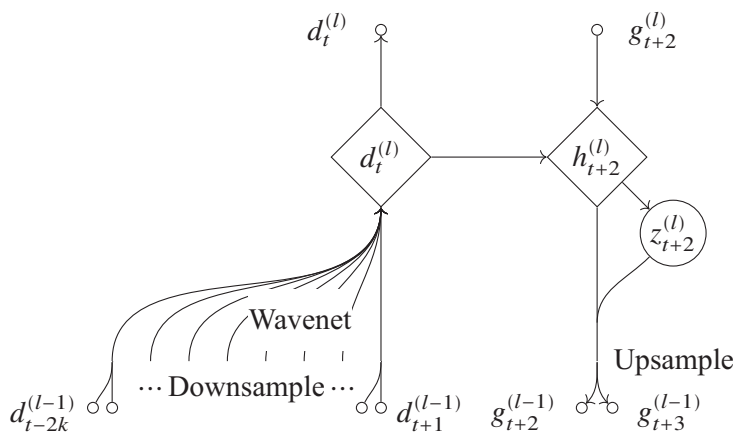
A multiscale autoregressive model is constructed with the deep hierarchical prior in mind. Compared to the temporal convolutional network, which does not progressively consolidate the features, it utilizes the same hierarchical structure as the multiscale recurrent neural network, c.f. Figure 4.7 with Figure 4.3, but without any recurrent connections.

The network consists of two parts, one part that extracts features from the previous observations, called the bottom-up network, and one part that uses these features to predict the next output, called the top-down network (c.f. Section 3.4). The features extracted from the history are, similar to the stochastic temporal convolutional network, produced progressively with a temporal convolutional network, but alternated with downsampling to consolidate the information. The top-down network is then responsible for upsampling these features again and produce the one-step ahead prediction. The top-down and bottom-up networks are also connected on each hierarchical level to create information shortcuts. The temporal convolutional network has a relatively short receptive field, meaning that the long-range dependencies are mainly

captured through the high-level features, c.f. U-net [112], which uses a similar idea for image segmentation. Figure 4.8 depicts both the bottom-up (dotted) and the top-down (solid) as well as the multiscale aspect with a downsampling/upsampling of two between each hierarchical layer.



(a) The bottom-up network (dotted) extracts features hierarchically and downsamples them. This implies a U-net [112] like structure that allows local information to be used in a local context.



(b) A close-up of the dashed square in Figure 4.8a for the generative distribution. Here  $k$  is the receptive field of the Wavenet model in this layer.

**Figure 4.8:** A more specific version of the multiscale autoregressive model (see Figure 4.7) using a U-net [112] structure and hierarchically extracted features. Depending on the complexity of the data, the latent variables could be ignored.

**Figure 4.9:** Samples generated from a multiscale autoregressive model trained on handwritten text from the IamOnDB [30] dataset.

## 4.9 Example: Generative models

Let us continue the example on generative sequential models introduced in Section 1.5.2 and apply a multiscale autoregressive model to the handwritten text. Figure 4.9 demonstrates samples generated by the resulting model. These samples can be used to investigate what aspects of the data that the model has been able to capture. For example, by visually inspecting the samples we can see that the generated characters physically resemble those of written text and potentially also a few words. On a higher level we can also see that the samples seem to follow different styles indicating that the model also captured those properties of the data.

The multiscale autoregressive model can be used also without latent variables. In this case it is possible to investigate the effect of the multiscale prior and compare it to a Wavenet [8] model, which does not impose that prior. We experiment with two different model architectures, one reimplementations of Wavenet [8] and one multiscale autoregressive model structurally identical to the models used in Paper III, but without latent variables and slightly smaller. Even though the multiscale model uses significantly more parameters ( $> 10\times$ ), both models use a comparable computational complexity. The Wavenet even tends to require more memory than the multiscale model during training ( $\approx 30\%$  more). Note that for this experiment we consider all the samples independently and not grouped into frames. The estimated log-likelihoods for these models on the Blizzard dataset [31] are presented in Table 4.1, where the multiscale structure shows a considerably better performance regardless of the model size. Paper III further investigates the multiscale autoregressive model also with latent variables.

The usefulness of the intermediate features from the multiscale autoregressive models on other downstream tasks, i.e. transfer learning, has not yet been investigated and is a subject for future work. However, wav2vec [86] uses a

**Table 4.1:** Average log-likelihood per sequence for speech with and without a multiscale prior on the Blizzard dataset [31]. The large model uses approximately twice the amount of computational complexity as the smaller model and is slightly smaller than the models used in Paper III.

Model	Log-likelihood
Wavenet, Small	15 553
Wavenet, Large	16 608
Multiscale, Small	17 457
Multiscale, Large	17 919

similar network to produce downsampled features, which are very beneficial for speech recognition, see Section 3.6 for an example on this.

## 4.10 Transformers

Transformers [33] are a class of models that for the past years have dominated many of the competitions in deep learning, especially for problems that require very long memory and/or involve language. Even though transformers are not explicitly used in this work, they are used in many of this thesis’ related works and their position within sequential modeling today makes it difficult to ignore them.

The core component of the transformer is the attention layer which has three variants, standard attention, self-attention and masked self-attention – each of these is explained below. A transformer is built up by stacking several, so-called, transformer blocks, each consisting of a sequentially independent feature transformation (a  $1 \times 1$  convolution) and an attention layer. The terminology regarding transformers is not standardized and can, depending on the application, involve any of the three attention variants, in any combination. However, the most common case is that the transformer only includes masked self-attention, which is the convention used in this thesis if nothing else is mentioned.

One approach to explain the standard attention layer is to give an intuitive feeling for how it functions. An attention layer has two inputs, here called the prober and the information bank. The high-level intuition in attention is that each element in the prober wants to extract some information from the information bank. For this purpose, the information bank is restructured as a ”dictionary” with a set of values accessible by a matching set of keys, both created by a transformation of the elements in the information bank. Similarly, the elements of the prober are transformed into queries. In a simplified way, each query is used to extract a value from the information bank, where the value is selected as the value from the best matching key-value pair for the

query. The match is calculated as a similarity metric between the key and the query. However, instead of choosing the single best value for each query, a weighted average of the values in the dictionary is used, where the weights are the similarity measures.

In a mathematical notation the standard attention function is defined as  $A(\tilde{x}, \tilde{c})$ , where the inputs, the prober  $\tilde{x}$  and the information bank  $\tilde{c}$ , are represented with tensors as  $\tilde{x} = (\tilde{x}_i)_{i \in [N]} \in \mathbb{R}^{N \times F}$  and  $\tilde{c} = (\tilde{c}_i)_{i \in [M]} \in \mathbb{R}^{M \times \tilde{F}}$ . Here,  $F$  and  $\tilde{F}$  are the number of features in the inputs, respectively, while  $N$  and  $M$  are the number of elements in the inputs, respectively. With this we get the queries, keys and values, as

$$\begin{aligned} Q &= \tilde{x} W_Q, \\ K &= \tilde{c} W_K, \\ V &= \tilde{c} W_V, \end{aligned}$$

where  $W_Q \in \mathbb{R}^{F \times G}$ ,  $W_K \in \mathbb{R}^{\tilde{F} \times G}$ ,  $W_V \in \mathbb{R}^{\tilde{F} \times H}$  are projection matrices, and  $G$  and  $H$  are hyperparameters. For each query,  $Q_i$ , the output,  $\tilde{V}_i = A(\tilde{x}, \tilde{c})_i$ , is defined as,

$$\tilde{V}_i = \text{softmax}(Q_i K^T) V \quad (4.18)$$

In the more specialized case of self-attention, the same set of data,  $x$ , is used as input for the queries and for the keys and values, as the name suggests. Note that the attention layer is invariant to the order of the elements in both  $\tilde{x}$  and  $\tilde{c}$ .

Attention layers excel at operating on sets (see e.g. [113]) in the sense that by design are size/length and order agnostic in terms of the input. However, if appropriately customized, it can also be applied to sequential/ordered data. The main idea to achieve this is to expand the input (to the full transformer model) with more features, a position encoding, that contains the temporal information. Thus, when the input set  $\tilde{x}$  is of sequential nature, i.e.  $\tilde{x} = \tilde{x}_{1:T} \in \mathbb{R}^{T \times F}$ , it is expanded with some additional temporal features,  $\mathcal{F}(t)$ , as  $([\tilde{x}_t, \mathcal{F}(t)])_{t \in [T]}$ . These temporal features are fixed (nonparameteric) functions of the time step,  $t$ , typically as a set of trigonometric functions with different predetermined frequencies.

A transformer,  $T(\tilde{x}_{1:T})$ , with self-attention operates on the entire sequence in parallel to produce a sequence of output features. Similar to the recurrent neural network and temporal convolutional network the input is expanded with the past observations when modeling sequential output, or simply replaced (i.e.  $\tilde{x}_{1:T} = y_{0:T-1}$ ) when the original input is nonexistent. Without any constraints the self-attention layer can gather information from both the past and the future. Masked self-attention avoids this problem by masking every entry from future data by setting the similarity score to negative infinity for these entries. This effectively turns Equation (4.18) into

$$\tilde{V}_t = \text{softmax}(Q_t K_{1:t}^T) V_{1:t}. \quad (4.19)$$

There are three dominant use cases for transformers: translation, which can be seen as the breakthrough of the transformer structure [33] (see Section 4.12); generative models, which use very large models of stacked masked self-attention layers, e.g. GPT-3 [7]; and (nongenerative) unsupervised models, which learn useful representations of the sequence that can be used for transfer learning, e.g. BERT [85]. Recently they have also been applied to other types of data, most notably on images, e.g. ViT [15].

## 4.11 Sequential classification

All the sequential models presented so far in this chapter have had sequential output with (optional) causally constrained input as focus. This section instead focuses on sequential input for classification, and in the next section sequence-to-sequence. Sequential classification with deep learning has no standardized solution, although there are several established approaches. One of these approaches is to use any of the basic models in this chapter as a method of feature extraction, i.e. recurrent neural networks, convolutional neural network or transformers, which we generally denote,

$$h_{1:T} = \text{SequenceModel}(\tilde{x}_{1:T}). \quad (4.20)$$

One significant difference of these sequence models compared to the models previously introduced, is that they do not necessarily have to uphold the causal constraints that limited the autoregressive and state-space models. Hence, in the case of convolutions it is possible to use ordinary 1D convolutions instead of temporal convolutional networks (see e.g. [6]). Similarly, in the case of recurrent neural network, it is possible to use bidirectional recurrent neural networks that aggregate information both forward and backward in time, see e.g. [114]. Finally, in the case of transformers, the feature extraction is not limited to masked self-attention see e.g. [85].

However, one obstacle with this approach of feature extraction is that the length of the input sequence and thus the length of the features might vary, which implies that it is necessary to aggregate the extracted features somehow. The perhaps most basic idea to do this aggregation is with global average pooling, i.e. averaging the extracted features. The averaged feature is then independent of the sequence length and can be used as input to a standard classification network,

$$\tilde{y} = \text{NN}(\text{Average}(h_{1:T})), \quad (4.21)$$

where  $\tilde{y}$  is used for the final prediction. Recurrent neural networks provide an alternate avenue to achieve this aggregation by simply using the final state or the output of the final state as a representation of the entire sequence,

$$\tilde{y} = \text{NN}(h_T). \quad (4.22)$$



It is also possible to use the final state of both the forward and the backward networks of a bidirectional recurrent neural networks [114]. Transformers have yet another unique solution to this aggregation problem. Recalling that the transformer essentially operates on a set it is possible to introduce a unique input token, often denoted [CLS], and appending it to the input vector. The prediction is then made using the output feature corresponding to that token, i.e.

$$h_{1:T+1} = \text{Transformer}([CLS], \tilde{x}_{1:T}), \quad (4.23)$$

$$\tilde{y} = \text{NN}(h_{T+1}). \quad (4.24)$$

See for example Devlin et al. [85] for a more detailed explanation of this idea.

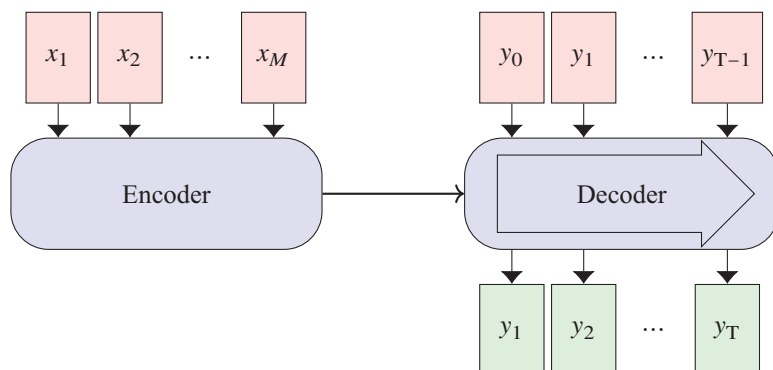
## 4.12 Sequential translation

The model used for a sequence-to-sequence or translation problem can be divided into two parts: an encoder that extracts features from the input sequence and a decoder that generates an output conditioned on these features. The encoder can, in principle, be chosen as any of the structures for feature extraction used for sequential classification (that is not necessarily including the aggregation part) presented in Section 4.11. The decoder can in turn be chosen as any of the sequential output methods discussed in this chapter or several of them in conjunction. See Figure 4.10 for a schematic overview of a translation model.

A problem with sequential translation is that the input sequence could be of arbitrary length and independent of the output sequence length. This gives rise to a similar aggregation problem as for sequential classification. There are primarily two solutions to this problem, either aggregate the input features in the same fashion as for sequential classification [115] or use attention [33] (see Section 4.10). An example of translation using attention for the conditioning between the encoder and the decoder is presented in Section 4.13 with the goal of generating speech from text.

## 4.13 Example: Text-to-speech

Let us continue the example on text-to-speech introduced in Section 1.5.3 by presenting some of the state-of-art text-to-speech models. Text-to-speech models typically consist of two parts: an encoder that operates on the text input and extracts features from it; and a decoder that generates the waveform. Up until recently, text-to-speech models have required a combination of different submodules and algorithms to achieve good performance. For example, the decoder in Tacotron [116] predicts a sequence of spectrograms, which then are converted into waveforms using the Griffin-Lim algorithm [117]. An



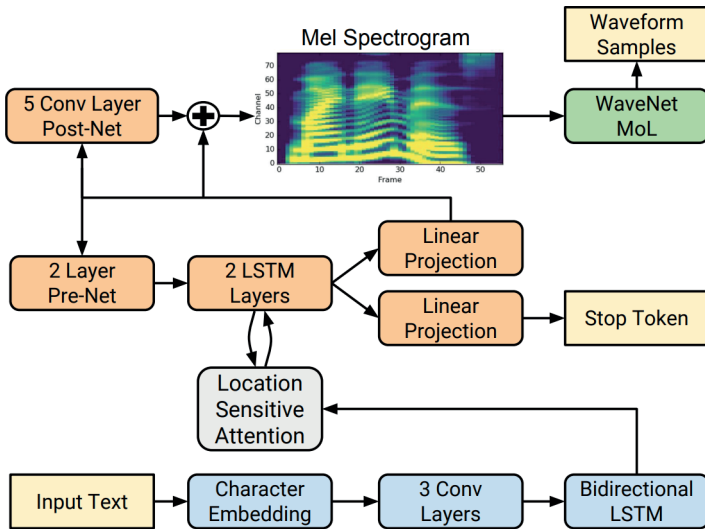
**Figure 4.10:** A schematic picture of a sequence-to-sequence model. The encoder produces features that are used by the decoder that generates the output. The encoder is structurally similar to a sequential classification model, i.e. no sequential constraints, while the decoder model is a generative model, here with the previous outputs explicitly included.

improved version of this is the Wave-Tacotron [32], that uses a normalizing flow to generate the waveform instead, which also implies that the intermediate representation with spectrograms is no longer needed. Apart from these two methods for the decoder there are also other models using for example Wavenet [118] (see Figure 4.11) and GANs [119].

The encoder encodes the input text to a more suitable representation. This can also include a phonetic middle step by a phonemizer, which gives a better representation of how to pronounce the given text [119]. The overall structure of the encoder is similar between different text-to-speech models and typically uses a sequential feature transformation as those we discussed in this chapter, e.g. convolutions and bidirectional recurrent neural networks [32, 116, 119].

For proper conditioning on the text, it is typically not enough for the text-to-speech model to simply aggregate the information as in sequential classification (Section 4.11). Instead, an attention mechanism is used as this enables the model to condition on a sequence with varying length. This is not exactly the same attention as in Section 4.10, since this attention also feeds back what the model attended in the last time step. It is thus more similar to a recurrent neural network than the attention introduced there [120].

This section gives only a brief background on the ideas for text-to-speech modeling and is not at all meant as a comprehensive overview of the subject. However, the area poses as potential future work for the multiscale autoregressive model, which neatly fits into the sequence translation framework as a part of the decoder network.



**Figure 4.11:** A schematic overview of the text-to-speech model developed by Shen et al. [118]. The bottom three boxes correspond to the encoder while the top orange and green boxes correspond to the decoder. The attention model that is used to connect them is the gray box in the middle. ©2018 IEEE



## Conclusion and future work

*“So long and thanks for all the fish.”*

– Douglas Adams

This work came to be during a very influential time in machine learning. Artificial intelligence has moved into the public sphere, both in form of new applications, e.g. driver assistance and self-driving cars, but also in aspects on social discussions e.g. ethics associated with machine learning.

### 5.1 Conclusion

The contributions we cover in this thesis are in three quite distinct directions, although deep learning, probabilistic modeling and sequential data are a common thread among them. The first contribution is in the intersection of deep learning and system identification, both exploring new ways of applying deep learning in typical system identification settings as well as relating the deep learning framework to system identification. The second is on aspects of verification and evaluation of probabilistic models. Specifically, we discuss the aspect of calibration for classification models. The final contribution is related to sequential deep learning and what we call a deep hierarchical prior and its role as an inductive prior for sequential data.

This work is to no amount exhaustive in the respective areas and still have many potential avenues for further research. While probabilistic and sequential modeling has a strong theoretic background, deep learning and the many aspects around it are still to a large degree empirical. Even if some theoretical advancements have been made it should still be regarded as an engineering field. However, this does not imply that deep learning should be avoided. One can draw parallels to the early ages of aviation where the physics behind airplanes was not fully understood, yet they flew and have since changed the society drastically.

Even though it is difficult, if not impossible, to predict what the future has in store in terms of artificial intelligence and machine learning, I would still like to make a guess, or perhaps more of a hope, about the future. One structural flaw

with many state-of-the-art models of the current era is the computation and power consumption associated with them. While the performance of the models is approaching human level in many areas, the energy consumption required to achieve this level is orders of magnitude higher than that of a biological brain. When I visualize the successor of deep learning, I see analog/neuromorphic circuits instead of digital to reduce energy consumption. Furthermore, instead of optimizing the entire function globally, as with gradient descent, I believe in a decentralized optimization – where the training is done in local cliques. However, this vision is perhaps as ill-advised as airplanes with flapping wings. Only the future will tell.

## 5.2 Future work

As hinted on in the previous section there are still many open questions in the directions that this thesis visits.

- We believe that deep learning still has a lot to gain from system identification and vice versa. One concrete example of this is the HIPPO [121] parameterization of a state-space model, which has shown remarkable success on long-term memory problems in deep learning and has so far not been covered in system identification.
- The hierarchical multiscale autoregressive model is still quite unexplored. Here an interesting avenue is to evaluate the learned features usefulness in transfer learning as well as in a conditional generative setting, for example text-to-speech.
- Finally, we believe that foundational models are yet to be applied more broadly, especially in the area of audio, e.g. text-to-speech, but also other new domains.

## References

- [1] L. Ljung. *System Identification: Theory for the User*. Upper Saddle River, NJ, USA: Prentice Hall, 1986.
- [2] T. Söderström and P. Stoica. *System Identification*. New York: Prentice Hall, 1989.
- [3] I. Sutskever, O. Vinyals, and Q. V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv:1409.3215v3.
- [4] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning.” In: *Nature* (2015).
- [5] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. “Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention.” In: *Proceedings of the International Conference on Machine Learning*. 2020.
- [6] A. H. Ribeiro, M. H. Ribeiro, G. M. M. Paixão, D. M. Oliveira, P. R. Gomes, J. A. Canazart, M. P. S. Ferreira, C. R. Andersson, P. W. Macfarlane, W. Meira Jr., T. B. Schön, and A. L. P. Ribeiro. “Automatic diagnosis of the 12-lead ECG using a deep neural network.” In: *Nature Communications* (2020).
- [7] T. B. Brown et al. “Language Models are Few-Shot Learners.” In: *Advances in Neural Information Processing Systems*. 2020.
- [8] 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*. 2016. arXiv:1609.03499v2.
- [9] S. Aliakbarian, F. Saleh, L. Petersson, S. Gould, and M. Salzmann. *Contextually Plausible and Diverse 3D Human Motion Prediction*. 2019. arXiv:1912.08521v4.
- [10] 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.
- [11] C. M. Bishop. *Pattern Recognition And Machine Learning*. Springer, 2006.
- [12] A. Vahdat and J. Kautz. “NVAE: A Deep Hierarchical Variational Autoencoder.” In: *Neural Information Processing Systems*. 2020.

- [13] J. Lago, F. De Ridder, and B. De Schutter. “Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms.” In: *Applied Energy* (2018).
- [14] S. Kahl. “Identifying Birds by Sound: Large-scale Acoustic Event Recognition for Avian Activity Monitoring.” PhD thesis. Chemnitz University of Technology, 2019.
- [15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. arXiv: 2010.11929.
- [16] D. G. T. Barrett and B. Dherin. “Implicit Gradient Regularization.” In: *International Conference on Learning Representations*. 2021.
- [17] A. Jacotécole, J. Jacotécole, P. Fédérale De Lausanne, and F. Gabriel. “Neural Tangent Kernel: Convergence and Generalization in Neural Networks.” In: *International Conference on Neural Information Processing Systems*. 2018.
- [18] R. Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2021. arXiv: 2108.07258.
- [19] 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* (2017).
- [20] S. Abnar, M. Dehghani, and W. Zuidema. *Transferring Inductive Biases through Knowledge Distillation*. 2020. arXiv: 2006.00555.
- [21] A. G. Wilson and P. Izmailov. *Bayesian deep learning and a probabilistic perspective of generalization*. 2020. arXiv: 2002.08791.
- [22] W. J. Song, Y. S. Chang, S. Faruqi, J. Y. Kim, M. G. Kang, S. Kim, E. J. Jo, M. H. Kim, J. Plevkova, H. W. Park, S. H. Cho, and A. H. Morice. “The global epidemiology of chronic cough in adults: A systematic review and meta-analysis.” In: *European Respiratory Journal* (2015).
- [23] A. H. Morice et al. “ERS guidelines on the diagnosis and treatment of chronic cough in adults and children.” In: *The European respiratory journal* (2020).
- [24] H. Johansson, A. Johannessen, M. Holm, B. Forsberg, V. Schlünssen, R. Jögi, M. Clausen, E. Lindberg, A. Malinovschi, and Ö. I. Emilsson. “Prevalence, progression and impact of chronic cough on employment in Northern Europe.” In: *The European respiratory journal* (2021).
- [25] P. Wändell, A. C. Carlsson, B. Wettermark, G. Lord, T. Cars, and G. Ljunggren. “Most common diseases diagnosed in primary care in Stockholm, Sweden, in 2011.” In: *Family practice* (2013).



- [26] S. Matos, S. S. Birring, I. D. Pavord, and D. H. Evans. “Detection of cough signals in continuous audio recordings using hidden Markov models.” In: *IEEE Transactions on Biomedical Engineering* (2006).
- [27] S. S. Birring, T. Fleming, S. Matos, A. A. Raj, D. H. Evans, and I. D. Pavord. “The Leicester Cough Monitor: preliminary validation of an automated cough detection system in chronic cough.” In: *European Respiratory Journal* (2008).
- [28] L. Kang. “Robust Handwritten Text Recognition in Scarce Labeling Scenarios: Disentanglement, Adaptation and Generation.” PhD thesis. Autonomous University of Barcelona, 2020.
- [29] S. Fogel, H. Averbuch-Elor, S. Cohen, S. Mazor, R. Litman, and A. Rekognition. “ScrabbleGAN: Semi-Supervised Varying Length Handwritten Text Generation.” In: 2020.
- [30] M. Liwicki and H. Bunke. “IAM-OnDB - an On-Line English Sentence Database Acquired from Handwritten Text on a Whiteboard.” In: *Proceedings of the Eighth International Conference on Document Analysis and Recognition*. USA, 2005.
- [31] S. K. Karaiskos and Vasilis. “The Blizzard Challenge 2013.” In: *Blizzard Challenge workshop*. 2013.
- [32] R. J. Weiss, R. Skerry-Ryan, E. Battenberg, S. Mariooryad, and D. P. Kingma. “Wave-Tacotron: Spectrogram-free end-to-end text-to-speech synthesis.” In: *International Conference on Acoustics, Speech and Signal Processing*. 2020.
- [33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. “Attention Is All You Need.” In: *Advances in Neural Information Processing Systems*. 2017.
- [34] T. Wigren and J. Schoukens. “Three free data sets for development and benchmarking in nonlinear system identification.” In: *European Control Conference*. 2013.
- [35] C. Andersson, N. Wahlström, and T. B. Schön. “Data-Driven Impulse Response Regularization via Deep Learning.” In: *Proceedings of 18th IFAC Symposium on System Identification (SYSID)*. Stockholm, Sweden, 2018.
- [36] C. Andersson, A. L. Ribeiro, K. Tiels, N. Wahlström, and T. B. Schön. “Deep convolutional networks in system identification.” In: *Proceedings of the IEEE 58th IEEE Conference on Decision and Control (CDC)*. Nice, France, 2019.
- [37] C. R. Andersson, N. Wahlström, and T. B. Schön. “Learning deep autoregressive models for hierarchical data.” In: *Proceedings of 19th IFAC Symposium on System Identification (SYSID)*. Padova, Italy (online), 2021.

- [38] J. Vaicenavicius, D. Widmann, C. Andersson, F. Lindsten, J. Roll, and T. Schön. “Evaluating model calibration in classification.” In: *Proceedings of Machine Learning Research*. 2019.
- [39] L. Ljung, C. Andersson, K. Tiels, and T. B. Schön. “Deep Learning and System Identification.” In: *IFAC-PapersOnLine* (2020).
- [40] C. Andersson. *Deep learning applied to system identification : A probabilistic approach*. Licentiate thesis, Uppsala University, Sweden. 2019.
- [41] B. Lakshminarayanan, A. Pritzel, and C. B. Deepmind. “Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles.” In: *Advances in Neural Information Processing Systems*. 2017.
- [42] 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*. 2014.
- [43] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning : data mining, inference, and prediction*. 2009.
- [44] M. Belkin, D. Hsu, S. Ma, and S. Mandal. “Reconciling modern machine-learning practice and the classical bias–variance trade-off.” In: *Proceedings of the National Academy of Sciences of the United States of America* (2019).
- [45] J. Močkus. *On Bayesian Methods for Seeking the Extremum*. Springer Berlin Heidelberg, 1975.
- [46] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. [http : //www.deeplearningbook.org](http://www.deeplearningbook.org). MIT Press, 2016.
- [47] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems.” In: *Journal of Basic Engineering* (1960).
- [48] A. J. Viterbi. “Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm.” In: *IEEE Transactions on Information Theory* (1967).
- [49] A. Doucet, S. Godsill, and C. Andrieu. “On sequential Monte Carlo sampling methods for Bayesian filtering.” In: *Statistics and Computing* (2000).
- [50] C. Blundell, J. Cornebise, K. Kavukcuoglu, W. Com, and G. Deepmind. “Weight Uncertainty in Neural Networks Daan Wierstra.” In: *International Conference on Machine Learning*. 2015.
- [51] D. F. Specht. “Probabilistic Neural Networks.” In: *Neural Networks* (1990).
- [52] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2016. arXiv:1603.04467.

- [53] 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: *Advances in Neural Information Processing Systems, Workshop Autodiff*. 2017.
- [54] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck. “A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music.” In: *International Conference on Machine Learning*. 2018.
- [55] N. Kalchbrenner, K. Com, and G. Deepmind. “Pixel Recurrent Neural Networks Koray Kavukcuoglu.” In: *International Conference on Machine Learning*. 2016.
- [56] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. “PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications.” In: *International Conference on Learning Representations*. 2017.
- [57] J. Bayer and C. Osendorfer. *Learning Stochastic Recurrent Networks*. 2014. [arXiv: 1411.7610](https://arxiv.org/abs/1411.7610).
- [58] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. “Sequential Neural Models with Stochastic Layers.” In: *Advances in Neural Information Processing Systems*. 2016.
- [59] J. Chung, K. Kastner, L. Dinh, K. Goel, A. Courville, and Y. Bengio. “A Recurrent Latent Variable Model for Sequential Data.” In: *Advances in Neural Information Processing Systems*. 2015.
- [60] L. Dinh, J. Sohl-Dickstein, and S. Bengio. “Density estimation using Real NVP.” In: *International Conference on Learning Representations*. 2016.
- [61] D. P. Kingma and P. Dhariwal. “Glow: Generative Flow with Invertible  $1 \times 1$  Convolutions.” In: *Neural Information Processing Systems*. 2018.
- [62] G. Papamakarios, T. Pavlakou, and I. Murray. “Masked Autoregressive Flow for Density Estimation.” In: *Neural Information Processing Systems*. 2017.
- [63] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. *Improved Variational Inference with Inverse Autoregressive Flow*. 2016.
- [64] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. “Normalizing Flows for Probabilistic Modeling and Inference.” In: *Journal of Machine Learning Research* (2021).
- [65] J. Ho, A. Jain, and P. Abbeel. “Denoising Diffusion Probabilistic Models.” In: *Advances in Neural Information Processing Systems*. 2020.
- [66] Z. Kong, W. Ping, J. Huang, K. Zhao, B. Research, and B. Catanzaro. “Diffwave: A versatile diffusion model for audio synthesis.” In: *International Conference on Learning Representations*. 2021.

- [67] T. Karras, S. Laine, and T. Aila. “A style-based generator architecture for generative adversarial networks.” In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2019.
- [68] T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, and T. Aila. “Alias-Free Generative Adversarial Networks.” In: *Advances in Neural Information Processing Systems*. 2021.
- [69] M. Arjovsky, S. Chintala, and L. Bottou. “Wasserstein Generative Adversarial Networks.” In: *Proceedings of the 34th International Conference on Machine Learning*. 2017.
- [70] D. J. Rezende, S. Mohamed, and D. Wierstra. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models.” In: *International Conference on Machine Learning*. Beijing, China, 2014.
- [71] D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes.” In: *International Conference on Learning Representations*. 2014.
- [72] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method*. Wiley Publishing, 2016.
- [73] C. Zhang, J. Bütepage, H. Kjellström, and S. Mandt. *Advances in Variational Inference*. 2017. arXiv: 1711.05597v3.
- [74] J. T. Rolfe. “Discrete variational autoencoders.” In: *International Conference on Learning Representations*. 2017.
- [75] R. J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.” In: *Machine Learning* (1992).
- [76] Y. Gal. “Uncertainty in Deep Learning.” PhD thesis. University of Cambridge, 2016.
- [77] 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*. Stroudsburg, PA, USA, 2016.
- [78] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. “Ladder Variational Autoencoders.” In: *Advances in Neural Information Processing Systems*. 2016.
- [79] L. Maaløe, M. Fraccaro, V. Liévin, and O. Winther. *BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling*. 2019. arXiv: 1902.02102v1.
- [80] Y. LeCun, C. Cortes, and C. Burges. *MNIST handwritten digit database*. 2010. URL: <http://yann.lecun.com/exdb/mnist>.
- [81] J. D. D. Havtorn, J. Frellsen, S. Hauberg, and L. Maaløe. “Hierarchical VAEs Know What They Don’t Know.” In: *International Conference on Machine Learning*. 2021.

- [82] S. J. Pan and Q. Yang. “A survey on transfer learning.” In: *IEEE Transactions on Knowledge and Data Engineering* (2010).
- [83] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei. *ImageNet: A large-scale hierarchical image database*. 2010.
- [84] M. Raghu, C. Zhang, G. Brain, J. Kleinberg, and S. Bengio. “Transfusion: Understanding Transfer Learning for Medical Imaging.” In: *Conference on Neural Information Processing Systems*. 2019.
- [85] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: 1810.04805v2.
- [86] A. Baevski, H. Zhou, A. Mohamed, and M. Auli. “wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations.” In: *Advances in Neural Information Processing Systems*. 2020.
- [87] R. Caruana. “Predicting Good Probabilities With Supervised Learning Alexandru Niculescu-Mizil.” In: *International Conference on Machine Learning*. 2005.
- [88] B. Zadrozny and C. Elkan. “Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers.” In: *International Conference on Machine Learning*. 2001.
- [89] B. Settles. *Active Learning Literature Survey*. Tech. Rep., University of Wisconsin–Madison, USA. 2010.
- [90] J. Bröcker. “Some Remarks on the Reliability of Categorical Probability Forecasts.” In: *Monthly Weather Review* (2008).
- [91] B. Zadrozny and C. Elkan. “Transforming classifier scores into accurate multiclass probability estimates.” In: *International Conference on Knowledge Discovery and Data Mining*. 2002.
- [92] 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.
- [93] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors.” In: *Nature* (1986).
- [94] J. J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities.” In: *Proceedings of the National Academy of Sciences of the United States of America* (1982).
- [95] S. G. Brush. “History of the Lenz-Ising Model.” In: *Reviews of Modern Physics* (1967).
- [96] J. L. Elman. “Finding Structure in Time.” In: *Cognitive Science* (1990).
- [97] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks.” In: *Neural Information Processing Systems*. 2015.

- [98] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory.” In: *Neural Computation* (1997).
- [99] R. Ly, F. Traore, and K. Dia. “Forecasting Commodity Prices Using Long-Short-Term Memory Neural Networks.” In: *International Food Policy Research Institute Discussion Paper*. 2021.
- [100] J. Chung, S. Ahn, and Y. Bengio. *Hierarchical Multiscale Recurrent Neural Networks*. 2016. arXiv:1609.01704v7.
- [101] J. Koutník, K. Greff, F. Gomez, and J. Schmidhuber. “A clockwork RNN.” In: *International Conference on Machine Learning*. 2014.
- [102] Z. Ghahramani and G. E. Hinton. “Variational Learning for Switching State-Space Models.” In: *Neural Computation* (2000).
- [103] S. E. Hihi and Y. Bengio. “Hierarchical Recurrent Neural Networks for Long-Term Dependencies.” In: *Conference on Neural Information Processing Systems*. 1995.
- [104] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio. “SampleRNN: An Unconditional End-to-end Neural Audio Generation Model.” In: *International Conference on Learning Representations*. 2017.
- [105] A. Goyal, A. Sordoni, M. Maluuba, M.-A. Côté, N. Rosemary, K. Mila, P. Montréal, and Y. Bengio. *Z-Forcing: Training Stochastic Recurrent Networks*. 2017. arXiv:1711.05411v2.
- [106] 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).
- [107] S. Bai, J. Zico Kolter, and V. Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. 2018. arXiv:1803.01271v2.
- [108] J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer Normalization*. 2016. arXiv:1607.06450.
- [109] 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* (2014).
- [110] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition.” In: *Conference on Computer Vision and Pattern Recognition*. 2016.
- [111] E. Aksan and O. Hilliges. “STCN: Stochastic Temporal Convolutional Networks.” In: *International Conference on Learning Representations*. 2019.



- [112] O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation.” In: *Medical Image Computing and Computer-Assisted Intervention* (2015).
- [113] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh. “Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks.” In: *International Conference on Machine Learning* (2019).
- [114] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. *Neural Architectures for Named Entity Recognition*. 2016. arXiv:1603.01360v3.
- [115] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv:1406.1078.
- [116] Y. Wang et al. “Tacotron: Towards end-to-end speech synthesis.” In: *INTERSPEECH*. 2017.
- [117] D. W. Griffin and J. S. Lim. “Signal Estimation from Modified Short-Time Fourier Transform.” In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* (1984).
- [118] J. Shen et al. “Natural TTS synthesis by conditioning wavenet on mel spectrogram predictions.” In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. 2018.
- [119] J. Donahue, S. Dieleman, M. Bińkowski, E. Elsen, and K. Simonyan. “End-to-End Adversarial Text-to-Speech.” In: *International Conference on Learning Representations*. 2021.
- [120] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. “Attention-Based Models for Speech Recognition.” In: *Advances in Neural Information Processing Systems* (2015).
- [121] A. Gu, K. Goel, and C. Ré. *Efficiently Modeling Long Sequences with Structured State Spaces*. 2021. arXiv:2111.00396.

# Acta Universitatis Upsaliensis

*Digital Comprehensive Summaries of Uppsala Dissertations  
from the Faculty of Science and Technology 2139*

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

Distribution: [publications.uu.se](http://publications.uu.se)  
urn:nbn:se:uu:diva-470433



ACTA  
UNIVERSITATIS  
UPSALIENSIS  
UPPSALA  
2022