ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Generative Sequence Models of developing Birdsong with Deep Neural Networks

Master Thesis in Information Technology and Electrical Engineering

Sandro Giacomuzzi

April 2, 2019

Advisor: Dr. G. Narula,

Supervisor: Prof. Dr. R. Hahnloser,
Institute of Neuroinformatics, ETH Zürich

Abstract

Like humans, the vocalizations of songbirds exhibit a strong dependency on hearing the adults they will imitate. Both have an early phase of learning that is primarily perceptual, which then serves to guide vocal production. Our aim was to build a generative model consisting of a sequence generator and an autoencoder that can be used to analyse different learning stages of the bird's song development. A hidden Markov model (HMM) that modeled the brain motor control areas was trained on encoded sequences of birdsong spectrogram segments that represented motor activations. Sampled sequences of motor activations by the HMM were then mapped to birdsong by a decoder function that consisted of deep neural networks and served as a model for the syrinx, the vocal organ of the bird. We found that it is crucial to match the latent (motor) representation of the autoencoder with the HMM. Therefore, we investigated two autoencoders related to variational autoencoders (VAE) that were able to encode spectrogram segments of variable length. The first had a continuous latent space and matched the HMM by choosing the approximative posterior from the same distribution family than the emissions of the HMM. The second had a discrete latent space and could be used to learn discrete HMMs and additionally served for applications on clustering. We expect that the well suited latent space will significantly support the HMM to learn and hence, will improve the quality of the generated birdsong spectrogram.

Contents

Chapter 1

Introduction

In this chapter we will motivate why it is useful to study birdsong development through a computational lens, provide a brief overview of what has been done in that field and explain the contribution of this thesis.

1.1 Learning and development of birdsongs

There are a lot of animal species that produce vocalization in order to communicate. Most of them do not need exposure to the communicative signals of their species in order to vocalize in a typical fashion for that species. Only a few groups like humans, whales, dolphins and parrots are capable of learning vocalizations during their lifetime. However, the most well-established vocal learners besides humans are songbirds.
Young songbirds learn their songs from adult birds, their tutors [3]. In isolation from adult songs they are unable to learn normal songs [6]. As described in [6], the song learning procedure during the first few months is divided into two phases. During the initial sensory phase, the bird only listens to the songs of its tutor and memorizes their acoustic patterns. This passive exposure to the sequential structure of tutor song goes on for about one month (depending on the species). In a second, possibly overlapping sensorimotor learning phase, the young bird starts vocalizing. The initial vocalizations are similar to human infant "babbling". After about two additional months of practice and thousands of attempts later, the bird begins to produce a song that somewhat resembles the tutor song. This process of learning is similar to "trial-and-error" reinforcement learning because the bird compares its own produced song to its memory of the tutor song. The generated song during this phase is called plastic song. After about two to three months the sensorimotor phase ends and the pupil produces a stable, stereotypical adult song, which in some species, such as the zebra finch (Taeniopygia guttata), remains unchanged throughout the bird's life. In this

thesis we restrict our analysis to zebra finch songs.

1.2 Similarities to human vocal learning

There are several parallels in how humans and songbirds learn to produce specific vocalizations in the early development of their vocal capabilities. For example, there are stage-wise transitions between phonemes for humans and syllables for songbirds [5]. It has been observed [6] that the way prelingual human infants at the age of one to two years learn diverse vocalizations is very stereotypical. Instead of a single developmental shift from repetitive to diverse sequences, there are multiple asynchronous shifts of new syllable types or permutations of old syllables. In experiments with zebra finches, a well-studied species of songbird, a similar learning strategy has been observed. The authors of [5] first trained the birds to perform one song. Afterward, the training target was altered and the birds adapted their song in a series of steps by swapping the syllable order or inserting new syllables into the sequence [5].
Other parallels are described in [6], such as the fact that humans and songbirds both exhibit a strong dependency on hearing the adults they will imitate. They both have critical periods for vocal learning, with a much greater ability to learn early in life. Furthermore, they both have an early phase of learning that is primarily perceptual, which then serves to guide vocal production.

1.3 Previous work

A lot of work has been done in modeling the interactions of the brain regions that are crucial for the birdsong generation. Fiete et al. [7] propose a model for song learning focusing on the interaction of the three brain areas HVC, RA (involved in song production) and LMAN (important for song variability). The model consists of gradient-based reinforcement learning driven by an internal critic that compares the bird's own song with a memorized template of an adult tutor song. In [8], Weber and Hahnloser investigate neural spike correlations in the HVC and RA. Assuming that neural population activity evolves along a finite set of states during singing they show that a Markov model is able to reproduce observed firing statistics and spike correlations in different neuron types and behavioral states. Blaettler et al. [11] demonstrate how a pair hidden Markov model (HMM) can be used to align the song of a pupil bird to that of its tutor.
Thus, there are several lines of support for HMMs in the computational neuroscience literature. This forms our basic motivation to use sequence models with a hidden Markov structure as well.
There are also biophysical models of birdsong learning. However, they have

not been able to capture the dynamical process of song learning accurately (for example [10]).
The bird's syrinx (vocal organ) as a highly non-linear function from low dimensional motor activations to complex birdsong is difficult to optimize [3]. Therefore, we were motivated to use deep neural networks to model the syrinx as they are powerful function approximators. Furthermore, we believe that using interpretable Bayesian algorithms with latent variables for learning birdsong dynamics can offer insight into the interaction between songbird motor control brain areas such as HVC and RA, [4] and the syrinx.

1.3.1 Deep Model of the bird HVC, RA and syrinx

This thesis builds up on two previous theses of Pálsson [1] and Giacomuzzi [2] in which a deep, latent variable generative model for birdsongs of zebra finches was trained and analyzed. It serves as a model for the syrinx where the controllable degrees of freedom of its input are represented by a low dimensional latent space. Any point in the latent space can then be considered an encoding of the output of the RA. In the proposed model, the forward and backward map between the latent and the space of spectrograms has been learned by an encoder-decoder neural network together with a Generative Adversarial Network (MD-GAN [21]). On top of the trained model for the syrinx, HMMs (section 2.10) were used to model song sequence generation. In figure 1.1, there is an illustration of the two model parts.
It was found, that HMMs of first order can learn to approximate birdsong dynamics well [9]. This was supported by the results of [2]. However, the artificially generated bird songs in [2] suffer from the problem of strong discontinuities between the concatenated spectrogram snippets as shown in figure 3.6.

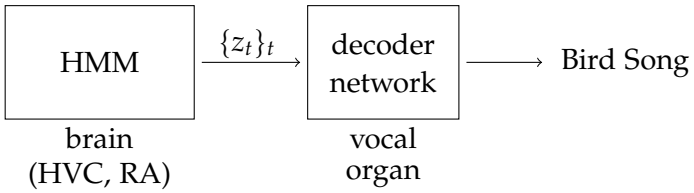


Figure 1.1: The two stage model. A HMM models the brain state of a particular learning stage. In a first step we generate a sequence of motor activations (or latent variables) $\{z_t\}_t$ from the HMM. Then the decoder network serves as a model for the vocal organ by converting the latent sequence to a birdsong.

1.4 Contribution of this thesis

In this project we investigated the GAN-HMM approach as described above in further detail. First, we trained HMMs with more hidden states and

with different types of Gaussian emissions for a more rigorous model selection. We divided our data set into equal-sized learning stages which we call "merges". For each merge, we analyzed how the model performance depends on the number of hidden states, the Gaussian covariance type and generally on the number of free parameters. We show that the latent space of the MD-GAN scrambled to such a degree that all the covariance types are comparable in performance and that using more than about 100 hidden states only helps to capture the few outliers not relevant for the quality of the generated songs. Second, we present how to get smoother samples without spectral discontinuities.
Furthermore, we devised two encoding schemes suitable for our problem, each of which has its own advantages over the MD-GAN. The first is called GS-VAE and infers a discrete vocabulary for fixed length spectrogram snippets. For any given spectrogram snippet, the encoder maps it to a discrete latent variable (the vocabulary). From there a decoder reconstructs the spectrogram snippet. This approach can be used as an unsupervised clustering method for fixed size spectrogram snippets. We used it also to train discrete HMMs in order to generate artificial birdsongs.
The second scheme is a recurrent Gaussian VAE-GAN. It encodes sequences of arbitrary length to a (fixed size) continuous latent space. The decoder network has a hidden dynamic structure that can generate for a given latent variable a spectrogram snippet of arbitrary length. Again, we can use this autoencoder to train HMMs on sequences of latent variables.
Finally, we married the two approaches to produce an autoencoder that maps sequences of arbitrary length to a discrete latent space. We call it the recurrent GS-VAE. It can be used for example as an unsupervised clustering method for sequences of arbitrary length.

1.5 Related work

1.5.1 Music generation

In principle, birdsong generation is quite similar to the challenge of artificial music generation as both are audio signals with a lot of harmonic content and rhythmic patterns [12]. Pure speech generation is closely related to that, although the topic of "text-to-speech", which produces audio from input text data, has more widely been researched. [13]
In the early days of music generation, the problem was approached with several preprocessing steps, such as segmentation and clustering, by criteria like rhythm and harmony. This kind of preprocessing based on human intuition is called feature engineering. However, as computers became more powerful, one could afford to employ deep differentiable architectures running through massive stocks of audio data and extract "good features" from scratch. This approach is called feature learning and is the kind of philoso-

phy that lies at the heart of deep learning.
It has long been believed that working with raw audio data, that is, working directly in the time domain of the audio samples, is more difficult than to transform the signal first to the frequency domain before feeding it to the learning algorithm. This is due to the fact that for harmonic sound (localized in frequency domain), samples in the time domain are highly non-local (uncertainty relation). However, in the frequency domain two consecutive short time Fourier transformations (spectrogram snippets) typically look very similar. The frequency domain was therefore assumed to be a better representation than the time domain, despite the fact that the signal becomes complex valued. Furthermore, it is the most useful domain of analysis for birdsong researchers. While in the earlier days, hidden Markov models were an important model for music and speech analysis [19], deep recurrent neural networks with long short-term memory (LSTM) [34] are now dominating the field of sequence learning and generation.
In 2016, DeepMind (Google) set a new benchmark on general audio generation with its deep generative neural network architecture called WaveNet [33]. Besides its great performance, it is remarkable that the network was trained on raw audio samples in the time domain.
However, deep autoregressive models like the WaveNet are not very useful for our purpose. They don't exhibit interpretable components like the hidden states of a HMM that are neurologically motivated. Futhermore, it is difficult to obtain samples from purely autoregressive models, because during the so called "forecasting" prediction errors accumulate. This typically causes the samples to become more and more dissimilar to the training data which in turn causes the model to make stronger predictive errors. We therefore choose to go for latent variable models instead. Nevertheless, we think it might be interesting for us to work with a WaveNet that is guided by a HMM.

1.5.2 Birdsong analysis with machine learning

We want to highlight three main challenges when it comes to the analysis of birdsong with machine learning.
The first is the problem of segmentation. The literature strongly suggests that a typical birdsong consists of a sequence of syllables. However, neither the boundaries nor the minimal length of a syllable are well-defined. A common assumption is that a syllable is characterized by a high value of autocorrelation and consists of harmonic patterns. The task of syllable segmentation can either be done manually by human, where the criterions are based on experience and intuition or one can use machine learning tools as in [14]. However, even in the latter case, the task of segmentation is not purely automatic and based on the choice of selected features and similarity measure by human. A system that is "free" of human choices would have

to infer both, the important features (feature learning) and the measure of similarity, from data.
The second challenge is sequence learning, that is, to train a sequence model able to produce birdsong similar to the original one. For example, Shiba [16] used waveNets in order to generate accurate birdsong and Sainburg [17] used diverse clustering and generation models like GAN based autoencoders.
The third challenge, which to the best of our knowledge has not been researched at all, is how the bird evaluates its song. Can we simulate how the bird compares its produced song relative to a tutor memory and using that evaluation to improve? One approach that can be taken is inverse reinforcement learning, which theoretically allows to infer the reward function of a learning pupil and hence explain its learning behavior [15].

Chapter 2

Methods

In this section we describe in more detail the methods we used to process the audio data and to build the generative models for the songs of the learning pupil.

2.1 Recording

The dataset we are working with consists of recordings of birdsong from one juvenile male zebra finch and its adult tutor. For the latter, song data was obtained from one day of singing undirected (isolated) song. For juveniles, the data was recorded over a period of about 80 days and during the initial 20 days the bird was exposed to its tutor for 2 hours every day in the morning from 9 to 11 am. The experiment started 45 days after birth. Songs were recorded at 32 KHz at 16 bit quantization and saved as LabView data files using in-house data acquisition code written in LabView (National Instruments). The data from one day of vocal behavior typically consisted of a mixture of songs, calls, and noise sources such as background noise and wing flaps. We cleaned the data by removing noise sources using a simple clustering procedure that discriminates noise from vocalization in a spectrogram. In figure 2.1, one finds spectrograms of a pupil in the early, middle and late learning stage.

2.2 Components

In this section we describe the data types and the data processing steps in more detail. A quick overview of the main components of the processing steps is given in figure 2.3.

1. **Preprocessing**
   It is not clear what a good representation of bird song audio data is

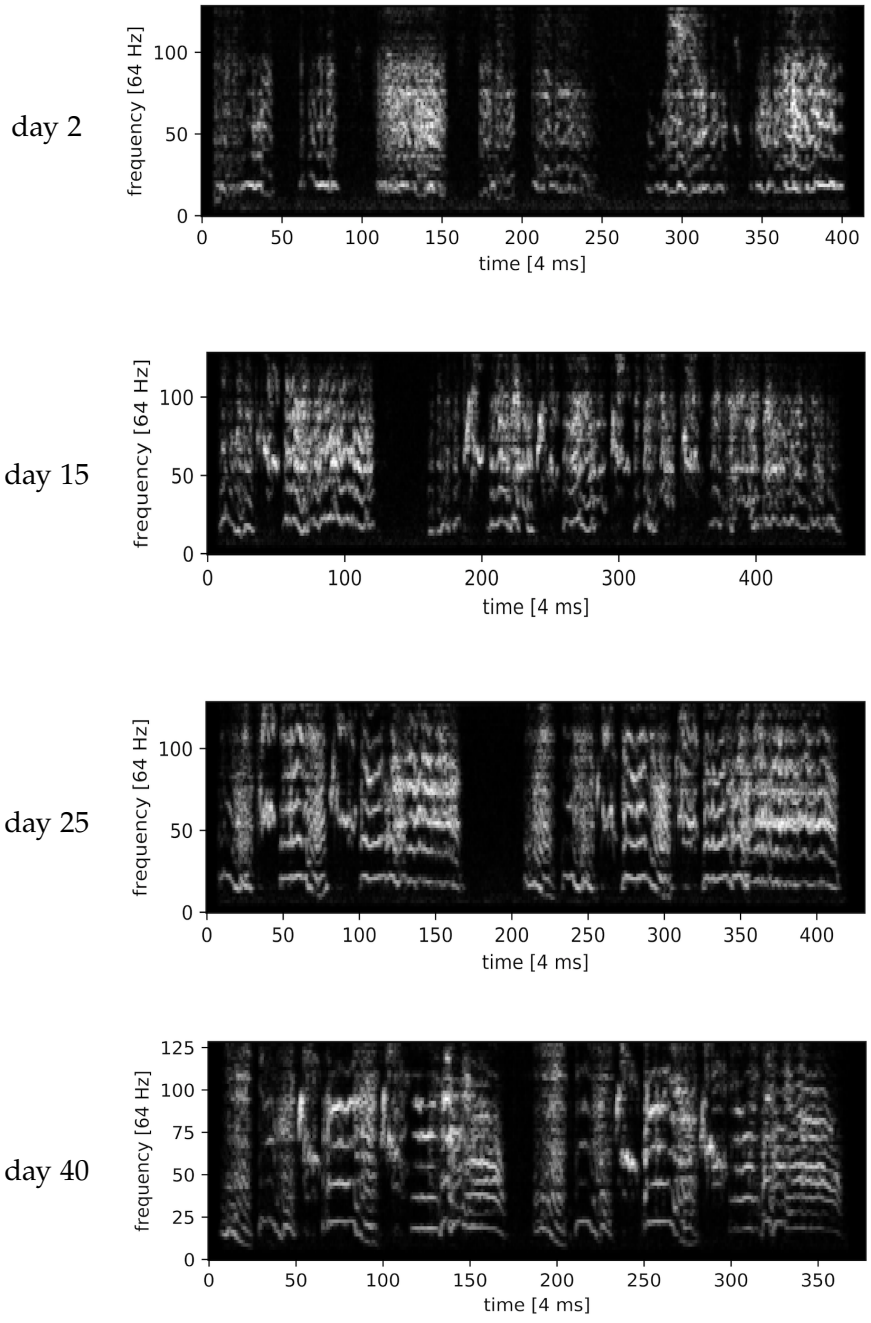


Figure 2.1: A comparison of birdsong spectrograms from different learning stages.

for our purpose. For example, one could work directly in the time space or its complex Fourier domain using short time Fourier transformations (STFT). Such representations are loss free, that is, we can

always perfectly recover the original data. In [1] we found that amplitude spectrograms are better suited for our task than complex STFT, although taking only the amplitude is a lossy representation because the phase information is lost.
The first preprocessing step was to compute the log-power magnitude of 75% overlapping STFT. We used a sample rate of 16 kHz and 256 samples for each STFT window.
For the input of the MD-GAN and the GS-VAE we cut the resulting spectrogram into small time chunks of 24 ms corresponding to 6 spectrogram columns. The resulting set of discrete spectrogram snippets was used for the training and test data of the MD-GAN.
The preprocessing step of the recurrent VAE-GAN and the recurrent GS-VAE was sligthly different. We segmented each song into silent and non-silent spectrogram snippets of variable length. We set a threshold that a silent snippet is at least 12 ms long and a non-silent snippet is at least 40 ms long. The preprocessed data then consisted of a set of n tuples $\{(x^{T_i}, \hat{T}_i)\}_i^n$, where x^{T_i} is a non-silent sequence of length T_i and $\hat{T}_i$ is the length of the following silent sequence.
The two cropping schemes are visualized in figure 2.2.

2. **Encoding**
   The autoencoder mapped spectrogram snippets $\{x_t\}_t$ of fixed/variable length to a low dimensional sequence of latent variables $\{z_t\}_t$. The MD-GAN and the recurrent VAE-GAN encoded to a continuous latent space and the GS-VAE to a discrete one. On this data representation we trained the HMMs.

3. **Sequence model**
   Depending on the type of latent variables we used a HMM with a Gaussian emission for a continuous latent space and categorical emission for a discrete latent space as described in section 2.10. The HMM's could then be used to generate an arbitrary long sequence in the latent space by sampling.

4. **Decoding**
   The decoder reconstructed the spectrogram snippet given a sequence of latent variables. We then concatenated the spectrogram snippets to a complete spectrogram image.

5. **Postprocessing**
   In order to reconstruct an audio time series from the magnitude spectrogram we applied Griffin-Lim's algorithm [18], a spectrogram inversion that iteratively restores the phase.

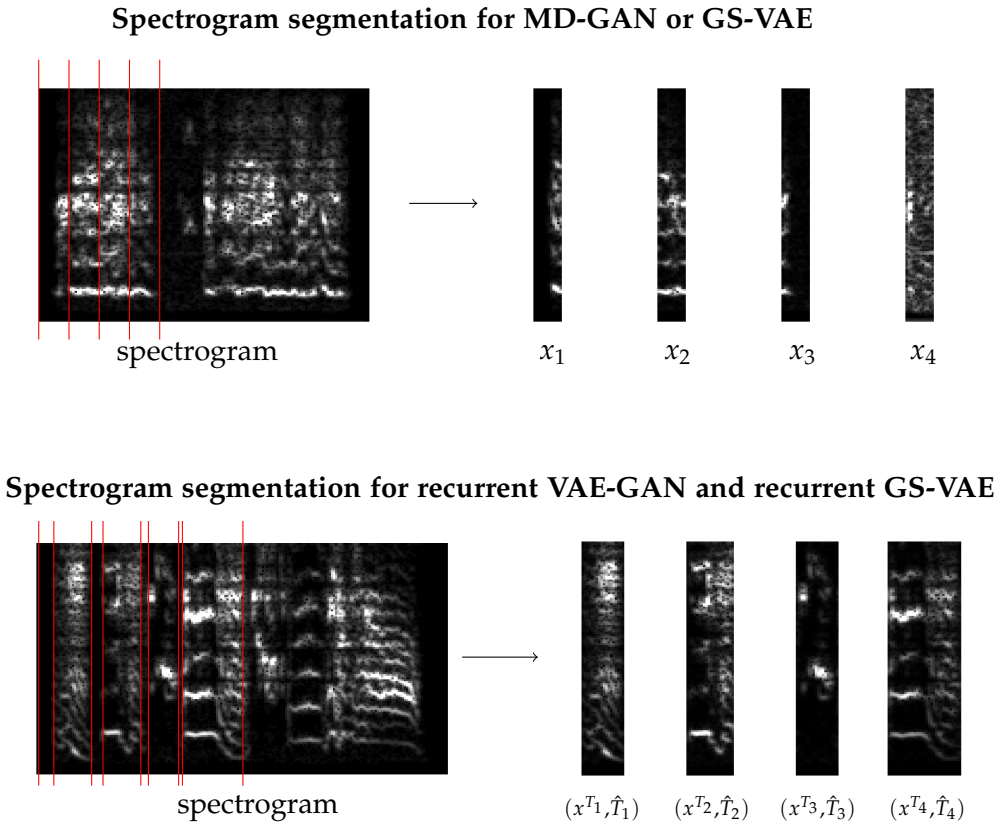


Figure 2.2: The two spectrogram segmentation schemes illustrate the preprocessing step for the discussed autoencoder as described in section 2.2. The upper scheme shows a equal length cropping and the lower scheme shows a segmentation into silent and non-silent spectrogram snippets (where only the non-silent segments are used to train the autoencoder).

2.3 Gumbel-Softmax relaxation

In this section we summarize the results from [25] and [26], which we will use in order to describe the Gumbel-Softmax VAE (GS-VAE).
Stochastic networks with categorical latent variables suffer from the problem, that it is in principle not possible to backpropagate through samples. The Concrete (or Gumbel-Softmax) distribution is a continuous relaxation of discrete random variables and is parameterized by a probability mass function and a temperature hyperparameter. The temperature controls how peaked the Concrete distribution is. We can use the concrete distribution for encoding to a (approximately) discrete latent space.
Let $q \in \Delta^{K-1}$ be a categorical probability distribution from the $(K-1)$-simplex. q is parameterized by $\pi_i \equiv q(i)$, such that

$$\sum_i \pi_i = 1, \quad \pi_i \geqslant 0 \, \forall i$$

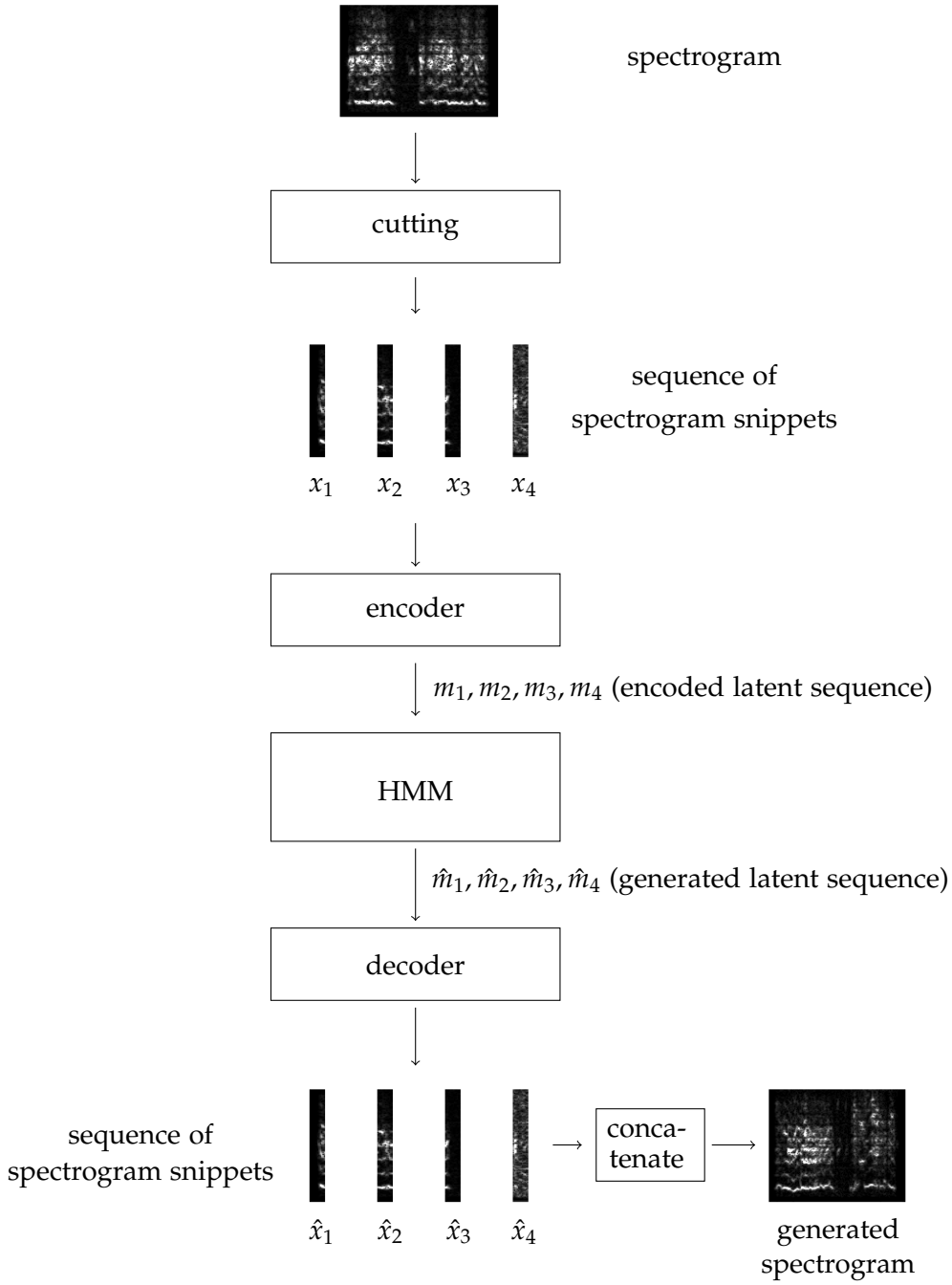


Figure 2.3: This figure illustrates the general data flow. An encoder maps spectrogram snippets to latent variables m. Next, a HMM is trained on sequences of latent variables. The trained HMM can then generate itself a sequence $\hat{m}_1, \hat{m}_2, ...$ Finally, a decoder maps latent variables back to spectrogram snippets.

and $\alpha_i = \lambda q(i) \equiv \lambda \pi_i$, $\lambda > 0$, i.e. $\alpha \in \mathbb{R}_+^K$, be a positively scaled version of it. Furthermore, let $oneHot(\cdot)$ be the operator that maps an index to the corresponding *oneHot*-vector, i.e.

$$oneHot(i) = (0, ..., 0, 1, ..., 0),$$

where the non-zero entry is at position i. We can now sample one-hot vectors from the PMF q as follows:

$$z = oneHot\left(\underset{i}{\text{argmax}}\{g_i + \log(\alpha_i)\}\right),$$

where g_i are i.i.d. $Gumbel(0, 1)$. We obtain

$$Pr[z = oneHot(i)] = \pi_i,$$

by the Gumbel arg-max property (appendix B). This is a reparameterization of the PMF q as described in appendix D. To see this, let's define

$$g_\pi(\varepsilon) = oneHot\left(\underset{i}{\text{argmax}}\{F^{-1}(\varepsilon_i) + \log(\alpha_i)\}\right),\ \varepsilon_i \text{ i.i.d. } Uni(0, 1),$$

where the function F^{-1} is the inverse cumulative distribution (CDF^{-1}) of the Gumbel normal distribution. Furthermore, let $q_\pi(z)$ be the PMF over the one-hot vectors z. We then indeed obtain $g_\pi(\varepsilon) \sim q_\pi(z)$. However, we can not back-propagate through this function, since $\nabla_\pi g_\pi(\varepsilon)$ is zero almost everywhere. The way out is a relaxation of the *argmax*-function to a softmax. The softmax function is defined by

$$softmax:\ \mathbb{R}^K \to \Delta^{K-1},\ u \to softmax(u)_i = \frac{\exp(u_i)}{\sum_{j=1}^K \exp(u_j)}.$$

For convenience let's abbreviate $\xi_i \equiv g_i + log(\alpha_i)$, $\xi = (\xi_1, ..., \xi_K)$. We then approximate

$$oneHot\left(\underset{i}{\text{argmax}}\{\xi_i\}\right) \equiv g_\pi(\varepsilon) = z \approx y = softmax\left(\frac{\xi}{\tau}\right),\ \tau > 0,$$

where the approximation can be made arbitrary good for a small enough temperature τ. Indeed, this approximation of the reparameterized PMF allows back-propagation and is called the Concrete relaxation. The corresponding PDF over the continuous y-variables is called the Concrete distribution $p_\tau(y|\pi)$.

Properties:

1. Rounding:
$$Pr[y_k < y_i, \forall i \neq k] = \pi_k$$
Assigning a sample y to its nearest one-hot vector is an unbiased operation.

2. Zero temperature:
$$\lim_{\tau \to 0} softmax\left(\frac{\xi}{\tau}\right)_k = \pi_k$$

3. Concrete PDF:
$$q_\tau(y|\pi) = c(\tau)\left(\sum_{i=1}^{K}\frac{\lambda\pi_i}{y_i^\tau}\right)^{-K}\prod_{j=1}^{K}\left(\frac{\lambda\pi_j}{y_j^{\tau+1}}\right),$$
with the prefactor $c(\tau) = (K-1)!\,\tau^{K-1}$. Note, that the PDF is actually independent of λ, since it cancels out in the fraction. We will use this property of invariance under scaling.

4. Convexity of the Concrete PDF:
If $\tau \leqslant (K-1)^{-1}$, then $q_\tau(y|\pi)$ is log-convex in y, i.e. all modes are at the corners of the probability simplex as desired.

Calculating log-probabilities of Concrete random variables can suffer from underflow. One approach to overcome this issue is to work directly in the log-space, as proposed by [25] and summarized in appendix C. Note that this issue is mentioned for the sake of completeness, since we will not need to calculate log-probabilites of the Concrete distribution in this work. However, a KL-divergence regularization term $D_{KL}\left(q_\tau(y|\alpha)||p_{\tau_p}(y|\alpha_p)\right)$ like it is usually used in VAEs (section 2.5) would indeed involve the computation of log-probabilities.

2.4 Generative adversarial network (GAN)

The GAN [20] is a crucial ingredient in two of our autoencoders, namely in the MD-GAN and the VAE-GAN. Therefore, we summarize its working principles in this section.
The goal of a GAN is to train a generative network that can approximate the data generating distribution $p_{data}(x)$. For that purpose, two deep neural networks are trained in parallel, namely a generator G and a discriminator D. The generator takes as input samples from a noise source $z \sim p(z)$ and transforms them into data points x. Hence, it transforms the noise distribution $p(z)$ into a "fake" data distribution $G(z)$.

The discriminator takes data points x and guesses whether they come from the real data generating distribution $p_{data}(x)$ or from the fake data distribution $G(z)$. On the other hand, the generator is trying to fool the discriminator. Hence, the two networks are trapped in a two player game, see figure 2.4. While the generator produces data that looks more and more realistic, the discriminators job gets harder and harder. This kind of variational inference between the two distributions $G(z)$ and $p_{data}(x)$ will eventually force the generated samples to look almost indistinguishable from real ones.

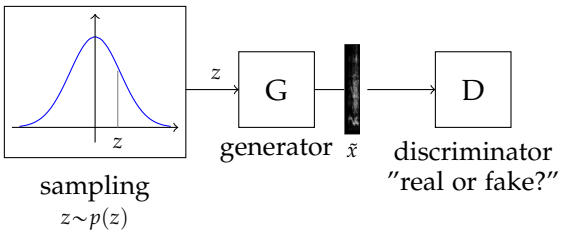


Figure 2.4: Illustration of the components of a GAN. The generator G transforms samples from a noise source into fake data points and the discriminator tries to distinguish them from real ones.

Loss function of the GAN

Let $D : x \to [0, 1]$ be the discriminator's probability that the data point x is "real". Then the loss function of the generator for a given generated sample $\hat{x} \sim G(z)$ is

$$loss_G(\hat{x}) = -\log\big(D(\hat{x})\big).$$

The discriminators loss for a real sample $x \sim p_{data}(x)$ and a fake sample $\hat{x} \sim G(z)$ is

$$loss_D(x, \hat{x}) = -\log\big(D(x)\big) - \log\big(1 - D(\hat{x})\big).$$

2.5 Variational autoencoder (VAE)

Our models are strongly related to variational autoencoders [36]. Therefore, we want to summarize the features of the VAE in this section.
Let $\{x_i\}_{i=1}^n$ be i.i.d. drawings from some data generating distribution $p_{data}(x)$. Furthermore, assume there are continuous unobserved latent variables z, s.t.

$$p_{data}(x) = \int_{\mathcal{Z}} p(x, z)dz = \int_{\mathcal{Z}} p(x|z)p(z)dz,$$

for some prior distribution $p(z)$ over the latent variables. Usually the integral expression is analytically intractable as well as the (real) posterior $p(z|x)$.

The VAE approach is to approximate the data likelihood $p_{data}(x)$ by first fixing a prior $p(z)$ and then maximize a lower bound of the data likelihood by learning the parameterized distributions $q(z|x)$ (approximate posterior) and $p(x|z)$ from data. We will call $q(z|x)$ the encoder and $p(x|z)$ the decoder.

Variational lower bound

In the following we derive a lower bound of the log-likelihood.

$$\begin{aligned}
\log(p_{data}(x)) &= \log(p_{data}(x)) \int q(z|x)dz \\
&= \int q(z|x) \log\left(\frac{p(x,z)}{p(z|x)}\right) dz \\
&= \mathbb{E}_q\left[\log\left(\frac{p(x,z)}{p(z|x)}\frac{q(z|x)}{q(z|x)}\right)\right] \\
&= D_{KL}\left(q(z|x))||p(z|x)\right) + \mathbb{E}_q\left[\log\left(\frac{p(x,z)}{q(z|x)}\right)\right] \\
&\geqslant \mathbb{E}_q\left[\log\left(\frac{p(x,z)}{q(z|x)}\right)\right] \\
&= \mathbb{E}_{q(z|x)}\left[\log(p(x|z))\right] - D_{KL}\left(q(z|x)||p(z)\right) \\
&= \mathcal{L}(x).
\end{aligned}$$

where we used the fact, that the Kullback-Leibler divergence is non-negative. Increasing this lower bound by optimizing the encoder and decoder also increases the data likelihood. Having a well trained model, we can sample data points by first sampling from the prior $z \sim p(z)$ and then sample from the decoder $x \sim p(x|\hat{z})$.

Optimization

Since we use deep neural networks for the encoder $q_\varphi(z|x)$ and the decoder $p_\theta(z|x)$ (φ and θ represent the network parameters) we need to calculate the gradients of the variational lower bound in order to update the network parameters. Hence, we need to calculate the term

$$\nabla_{\theta,\varphi}\mathcal{L}(\theta,\varphi;x) = \nabla_{\theta,\varphi}\mathbb{E}_{q_\varphi}(z|x)\left[\log(p_\theta(x|z)\right] - \nabla_\varphi D_{KL}\left(q_\varphi(z|x)||p(z)\right).$$

If the integrals are not solvable we need to approximate the gradients by sampling.
We will choose the prior to be D-variate Normal distributed $p(z) = \mathcal{N}(z|0,1)$ and the posterior to be a D-variate Gaussian with diagonal covariance matrix $q(z|x) = \mathcal{N}(z|\mu(x),\sigma^2(x))$. In that case the KL-divergence term is explicitly

solvable and we get [36]:

$$\begin{aligned} D_{KL}\left(q(z|x)||p(z)\right) &= D_{KL}\left(\mathcal{N}(z|\mu(x),\sigma^2(x))||\mathcal{N}(z|0,1)\right) \\ &= \int \mathcal{N}(z|\mu(x),\sigma^2(x))\left[\log\left(\mathcal{N}(z|0,1)\right) - \log\left(\mathcal{N}(z|\mu(x),\sigma^2(x))\right)\right] dz \\ &= ... \\ &= \frac{1}{2}\sum_{d=1}^{D}(1+\log(\sigma_d^2(x))) - \mu_d^2(x) - \sigma_d^2(x), \end{aligned}$$

where the last step involves to solve the standard integral and we used the notation $\mu = (\mu_1, \mu_2, ..., \mu_D)$ as well as $\sigma = diag(\sigma_1, \sigma_2, ..., \sigma_D)$.
For the expectation term we can use the reparameterization trick, described in appendix D, in order to approximate the gradient by sampling. A valid reparameterization of the Gaussian posterior by the auxiliary noise variable $\varepsilon \sim \mathcal{N}(0,1)$ is given by

$$z \sim \mathcal{N}(\mu, \sigma^2) \Rightarrow z = \mu + \sigma\varepsilon.$$

Therefore,

$$\begin{aligned} \mathbb{E}_{q_\varphi}(z|x)\left[\log\left(p_\theta(x|z)\right)\right] &= \mathbb{E}_{\mathcal{N}(z|\mu(x),\sigma^2(x))}\left[\log\left(p_\theta(x|z)\right)\right] \\ &= \mathbb{E}_{\mathcal{N}(\varepsilon|0,1)}\left[\log\left(p_\theta(x|\mu(x)+\sigma(x)\varepsilon)\right)\right] \\ &\approx \frac{1}{L}\sum_{l=1}^{L}\log\left(p_\theta(x|\mu(x)+\sigma(x)\varepsilon^{(l)})\right), \end{aligned}$$

where $\varepsilon^{(l)} \sim \mathcal{N}(0,1)$. For large enough minibatches we can choose $L = 1$. The overall loss function of this Gaussian VAE becomes

$$\begin{aligned} loss_{VAE}(x) &= -\mathcal{L}(x) \\ &\approx \frac{1}{2}\sum_{d=1}^{D}\left(1+\log(\sigma_d^2(x))\right) - \mu_d^2(x) - \sigma_d^2(x) - \log\left(p_\theta(x|\mu(x)+\sigma(x)\varepsilon)\right). \end{aligned}$$

In figure 2.5 we illustrate the described components of the VAE.

2.6 Models

In this thesis we mainly work with four different models, three of them are autoencoders that learn a low dimensional latent data representation and the fourth is a sequence generating network that learns the sequential data structure of the latent representation of the encoders.
For the endmost model we decided to work with hidden Markov models (HMM) described in section 2.10, because they are powerful and their architecture is transparent and easy to interpret.

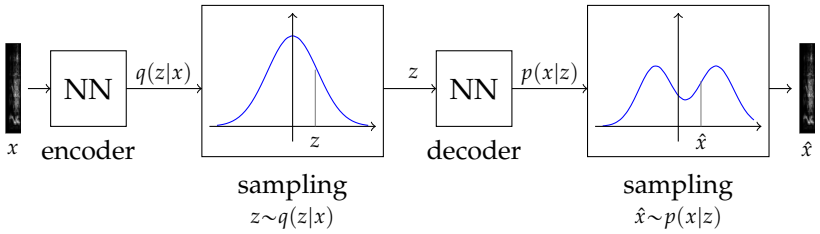


Figure 2.5: An illustration of the components of the VAE. Both the encoder and the decoder are probabilistic. They consist of deep neural networks and parameterize a probability distribution.

The three autoencoders are the MD-GAN, VAE-GAN and the GS-VAE. While the two former models have a continuous latent space, the latter has a discrete latent space.
In general, it is not clear how to evaluate an autoencoder, that is, how to determine which of two autoencoders is the better one. On the one hand, one could take a standard tool of measurement like pixel wise squared distance between real and reconstructed spectrograms, on the other hand it is unclear whether this is a useful metric for high dimensional images such as spectrograms. The best metric would be the one that the pupils use when they try to match their own vocalization with the one of the tutors. However, since we do not have such a "bird-metric" available, we need to rely on human evaluation by manually inspecting the quality of spectrograms and reconstructed audio signals.

2.6.1 MD-GAN

Standard autoencoders consisting of an encoder and a decoder neural network trained with a typical loss function like squared Euclidean distance are well suited to compress high dimensional data into a low dimensional latent space but they tend to suffer from blurry reconstructed images [40]. However, the generative adversarial network (GAN) - see section 2.4 - has shown to be incredibly successful in generating very sharp images that are hard to distinguish from real data. The mode regularized GAN, MD-GAN [21], combines both of the discribed properties as it consists of an autoencoder that benefits from the "sharpness property " of the GAN. MD-GAN is an extended model of the classical GAN. The original contribution of the MD-GAN extension was to train an autoencoder jointly with a GAN objective with the purpose of mode regularization. While the GAN is a strong model for data generation that captures its most important features (more precisely modes, that is, local concentration in the data generating probability distribution) it usually misses a lot of minor features, that is, less frequent but data typical characters will not be generated at all by the GAN. The reconstruction error of the autoencoder in the MD-GAN makes it very unlikely that a minor mode gets "overlooked" by the generator network.

In a previous project [1] we have found that indeed MD-GANs are capable of encoding spectrogram snippets to a low dimensional latent space while still having very sharp and accurate looking reconstructed spectrograms, see figure 3.1.

2.6.2 GS-VAE

A discrete encoding scheme allows us to train HMMs with discrete emissions. This has the advantage over HMMs with Gaussian emissions that we don't have to care about scattered latent spaces. Also, such an encoding scheme can be used as an unsupervised clustering method for example to find similar syllables. In a previous project [2] we discretized the latent space of the MD-GAN with k-means and analyzed the learning and generation of HMMs. Although the dynamic aspect of the generated songs looked plausible the discrete k-means encoding scheme seemed far from accurate. We therefore believe that k-means is not suitable in this case, since it basically assumes an Euclidean metric in the latent space, which is an arbitrary assumption given that the MD-GAN consists of deep neural networks that can transform a space in a highly non-linear manner.
However, we don't want to give up a discrete encoding scheme. Therefore, we are testing the relatively recently published Gumbel-Softmax variational autoencoder [26]. It provides a workaround for the problem of backpropagation through discrete nodes and hence gives us the possibility to directly learn a discrete encoding from data using deep neural networks.
During the thesis we upgraded the GS-VAE to a recurrent GS-VAE that can process spectrogram segments of different lengths using the same technique as for the recurrent VAE-GAN discribed in section 2.6.3.

2.6.3 Recurrent VAE-GAN

The recurrent VAE-GAN has a continuous latent space and is capable to encode spectrogram snippets of arbitrary length. This allows us to replace the rather arbitrary fixed size spectrogram cropping by more natural cropping schemes, like syllable of different length or silent/non-silent segmentation, see figure 2.2.
In order to process and generate spectrogram segments of different length we used a bidirectional RNN on the encoder side and a dynamical system (DS) on the decoder side.
Although a VAE consists of a probabilistic encoder and decoder we first implemented all components in a deterministic manner in order to compare the difference to a probabilistic encoder. If we specifically want to refer to the recurrent VAE-GAN with a probabilistic encoder we will call it recurrent Gaussian VAE-GAN. The decoder was always deterministic in this work.
On top of the VAE we used a GAN to regularize the decoder "towards

sharper images", that is, similar to the MD-GAN we added a discriminator that tries to distinguish real spectrogram segments from fake ones.

2.7 Encoder component functions

Basically, an encoder is a map from a high dimensional space (where the original data lives) to a low dimensional space (latent space) such that reconstruction is possible and, in a way that as little information as possible is lost. For example, we want that similar data points are represented close to each other in the latent space (or even represented by the same state in case of a discrete latent space), while dissimilar data points should be well distinguishable in the latent space. Hence, it becomes immediately clear that the choice of similarity measure in the image/data space, e.g. squared Euclidean distance, must be well suited to the kind of information we want to preserve. In the following, we present three different kinds of encoders.

2.7.1 MD-GAN encoder network

The encoder component of the MD-GAN is in our case a 5-layer convolutional neural network that maps spectrogram snippets from the image space $\mathcal{X}$ to a 16 dimensional latent space $\mathcal{Z}$.

$$\varphi_e : x \to z$$

More details can be found in [1], as the implementation and fine tuning of the MD-GAN is not part of this thesis.

2.7.2 GS-VAE encoder network

We use the Gumbel-Softmax relaxation to build an encoder with an approximately discrete latent space. In the following, we let $\{x_i\}_i^n$ be the data set of spectrogram snippets with fixed length. The decoding consists of two main steps.
First, a neural network NN maps spectrogram snippets to a latent vector $\log(\alpha) \in \mathbb{R}^K$, where $\alpha \in \mathbb{R}_+^K$ as in section 2.3, that is,

$$NN : x \to \log(\alpha).$$

Second, we sample from the Concrete distribution by using the reparameterization trick as described in section 2.3. Formally we have

$$y = softmax\left(\frac{g_i + \log(\alpha_i)}{\tau}\right), \quad g_i \sim Gumbel(0,1).$$

Note that y is a sample from the Concrete distribution $q_\tau(y|\alpha)$ as described in section 2.3. Hence, the complete encoder is the probabilistic map

$$\varphi_e : x \to q(y|x) = q_\tau\big(y|\alpha = \exp(NN(x))\big)$$

During the learning process we therefore should average over many samples (more details in appendix D), which works well for large enough minibatches. We found that for 400 latent states and a minibatch of size 1000 this works well. For fewer latent states one can choose much smaller minibatches. Figure 2.6 illustrates the two encoding steps.
One open issue is the adjustment of the temperature parameter τ during the training. If τ is choosen too small, then the gradient suffers from high variance while for a too high temperature the gradient is biased towards the interior of the probability simplex Δ^{K-1}. Although there is given an upper bound $\tau \leqslant (K-1)^{-1}$ in which the Concrete PDF is guaranteed to be convex, for large numbers of latent states K we can use much higher temperatures, since in [25] it was empirically found that the Concret distribution becomes significantly more peaked towards larger K. In our experiments with $K = 400$ we found that to start with, a temperature $\tau = 1$ worked well. During the training we decreased the temperature down to 0.5. We did not go lower, because the samples looked already good enough, see figure 3.7.
For a trained network with a small enough τ the vector y are samples from a very peaked Concrete distribution and y should therefore appear very close to a one-hot vector. Assigning y to its nearest one-hot vector is ensured to be an unbiased operation by the rounding property (section 2.3).
However, if we don't want to sample but assign the most likely latent state $i_{ML}(x)$ to a given spectrogram snippet x, we can simply select the maximum alpha-value, i.e.

$$i_{ML}(x) = \underset{i}{\operatorname{argmax}}\{\alpha_i\} = \underset{i}{\operatorname{argmax}}\{\exp\left(\varphi_e(x)\right)\} = \underset{i}{\operatorname{argmax}}\{\varphi_e(x)\}.$$

We did this whenever we used the GS-VAE (or recurrent GS-VAE) to cluster spectrogram segments.

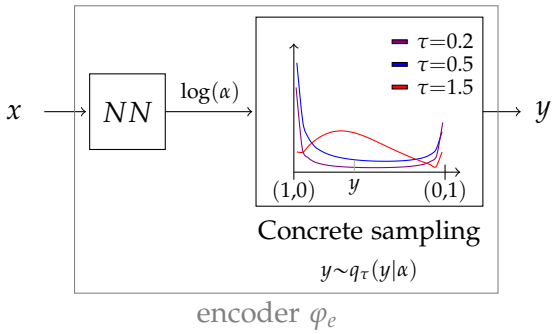


Figure 2.6: A visualization of the GS-VAE encoder network. In the box "Concrete sampling" a plot of the Concrete distribution is shown for the binary special case for different temperatures τ and with $\alpha_1/\alpha_2 = 2$, i.e. the first state is twice as probable as the second. The two states are marked in terms of the one-hot vectors $(1,0)$ and $(0,1)$. Note also that for example $\tau = 1.5$ leads to a inner mode.

2.7.3 Recurrent VAE-GAN encoder network

Since our data set consisted of sequences that have different lengths $\{x^{T_i}\}_i^n$, where $x^T \equiv (x_1, ..., x_T)$, we wanted to obtain a whole single latent representation of that sequence in a continuous space. As a encoder that can handle sequences of arbitrary length we used a bidirectional RNN that runs through the sequences. More specifically, we use a gated recurrent unit (GRU) [27]. For a sequence of length T we took the last two (fixed sized) state vectors $h_f^{(T)}$ and $h_b^{(1)}$ of the GRU. To be clear, $h_f^{(T)}$ was the last state of the forward direction, which corresponded to the time stamp T and $h_b^{(1)}$ is the last state of the backward direction, which corresponds to the time stamp 1.

$$\text{RNN} : x^T \to (h_f^{(T)}, h_b^{(1)}) = RNN(x^T)$$

A second neural network NN mapped to a low dimensional latent space $\mathcal{M}$. Its output m was used as an encoding of the sequence.

$$NN : (h_f^{(T)}, h_b^{(1)}) \to m$$

Hence, the complete encoder was

$$\varphi_e = NN \circ \text{RNN} : x^T \to m.$$

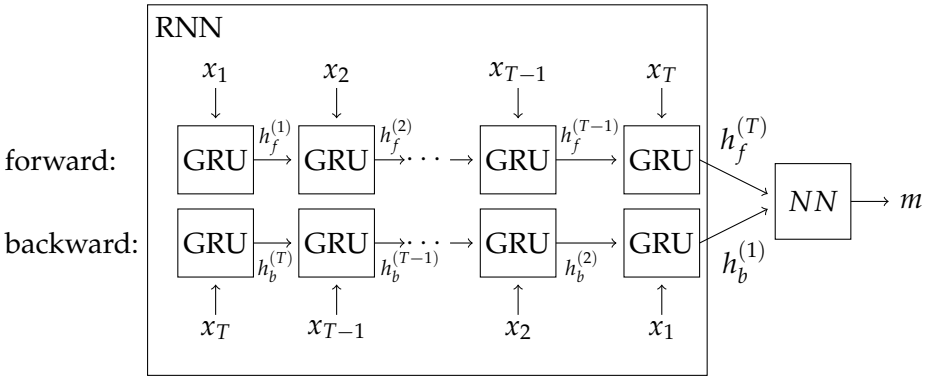


Figure 2.7: The bidirectional RNN encoder unrolled in time. A sequence $(x_1, ..., x_T)$ of arbitrary length T is encoded by the latent variable m.

2.8 Decoder component functions

The decoder's task was to find a good approximative inverse of the encoder. As such, it is a map from the latent space to the original data space while trying to make the reconstruction as similar as possible to the original data. The measure of similarity is induced by a loss function discussed in section

2.9. In more geometrical terms, a decoder defines a submanifold (that has the dimension of the latent space) in the original space that tries to fit the data points in a regressive manner.

2.8.1 MD-GAN decoder network

The decoder of the MD-GAN is identical to the generator of the deep convolutional GAN with 5 layers and ReLU activations. Essentially, it is a deep neural network that maps from the latent space back to the original data space

$$\varphi_d : z \to x$$

2.8.2 GS-VAE decoder network

The decoder network of the GS-VAE was a three-layer fully connected neural network φ_d with ReLU activations mapping back to the space of spectrogram snippets:

$$\varphi_d : y \to x$$

For our purpose we did not only want to encode and decode spectrogram snippets but we needed them to look as realistically as possible. Since we used single pixel squared error the reconstructed images looked very blurry, see figure 3.7. Luckily in this case there was a simple way out. We could assign a representative snippet to every state, that is, we went through the data and stored for every state the one snippet that was assigned to it with the largest likelihood. Formally, to every latent state i we defined the corresponding representative spectrogram snippet $x_i^{(r)}$ by

$$x_i^{(r)} = \arg\max_{x \in \text{data}} \{p(i|x)\},$$

where $p(i|x) = {}^{\alpha_i(x)}/_{\sum_j \alpha_j(x)}$ is the assignment probability of the spectrogram snippet x to the latent state i.

2.8.3 Recurrent VAE-GAN decoder network

In order to map back to the space of spectrogram snippets, we need a dynamic decoder that can map a given latent variable to a sequence of arbitrary length. For the sake of model interpretability we do this in two steps. First, a dynamical system generates a hidden sequence of arbitrary length (this idea was influenced by [28]). It is constructed via a dynamical space together with a vector field that defines for any given initial hidden state a unique trajectory along the flow (figure 2.8). We came up with two different kinds of such vector fields that are described below.
In a second step, a fully connected 4-layer neural network maps single sequence points of the dynamical trajectory back to single spectrogram columns of the spectrogram snippet.

Global non-linear dynamical system

The Global non-linear dynamical system is a sequence independent vector field $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$, $z \rightarrow \Delta z$, that is learned by a neural network. Together with an initial condition z_1, this uniquely defines a dynamical trajectory. The map from m to z_1 for each sequence is learned by a NN_1

$$NN_1 : m \rightarrow z_1.$$

Another neural network learns the global dynamics via the vector field $f : \mathbb{R}^d \rightarrow [0,1]^d$, $z \rightarrow \Delta z$. We used a sigmoid output at the last layer of the vector field in order to keep the step size reasonably small. The initial state together with the dynamical system determines uniquely the trajectory via the recursion

$$z_{t+1} = z_t + f(z_t), \; z_1 = NN_1(m).$$

This can be seen as an Euler discretization of a continuous time non-linear system with $\frac{dz(t)}{dt} \approx f(z(t))$ and step size one.

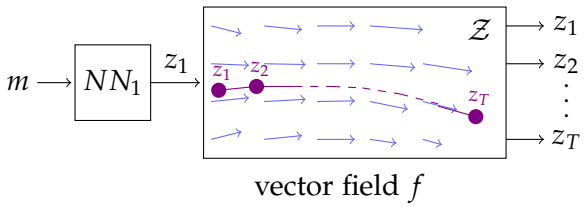


Figure 2.8: The figure illustrates the global non-linear dynamical system. The vector field f, indicated by the bluish arrows, drives the initial state z_1 along a trajectory (marked in violet). This scheme can straightforwardly be adapted to the linear dynamical system with the only difference that the output of NN_1 has additionally the linear map A and the stable point b.

Sequence dependent linear dynamical system

In contrast to the global non-linear dynamical system, we wanted to learn linear dynamics about a fixed point. Thus, not only the initial state z_1 needed to be learned, but also a linear map $A : \mathbb{R}^d \rightarrow \mathbb{R}^d$ and a fixed point b. Hence, in case of the linear dynamical system the first neural network of the decoder is the map

$$NN_1 : m \rightarrow (dz, A, b).$$

Note that the initial state of the dynamical system is the sum of a fixed point b and the relative position dz to it such that $z_1 = b + dz$. The dynamic trajectory is then given by the recursion

$$z_{t+1} = A^t dz + b, \; \forall t \geqslant 0.$$

The system is stable (that is, it converges to b) if for all eigenvalues λ_i of A it holds that $|\lambda_i| < 1$. The system is then converging towards its fixed point

b. To impose stability, we restricted the search space for A to matrices with elements satisfying

$$|a_{ij}| \cdot d < 1, \; \forall \, i, j.$$

We achieved this with a hyperbolic tangent output for each matrix element divided by d. Stability was then guaranteed by the following theorem:

Theorem 1. *For any matrix A and any matrix norm $\| \cdot \|$ it holds that*

$$|\lambda| \leqslant \|A\|, \textit{ for all eigenvalues } \lambda \textit{ of } A.$$

A proof can be found in [24]. For the Frobenius norm it indeed follows that

$$a_{ij} \in \tanh(\mathbb{R})/d \implies |a_{ij}| \cdot d < 1 \implies |\lambda|^2 \leqslant \|A\|_F^2 = \sum_{ij} |a_{ij}|^2 < \sum_{ij} \frac{1}{d^2} = 1.$$

Final map back to the image space

Finally, a second neural network NN_2 mapped back to the original data space. For a given latent sequence $z_1, ..., z_T$ the neural network performed a pointwise reconstruction

$$NN_2 : z_t \to \hat{x}_t, \; \forall t.$$

Hence, the complete decoder map was

$$\varphi_d = NN_2 \circ f \circ NN_1 : m \to \hat{x}^T.$$

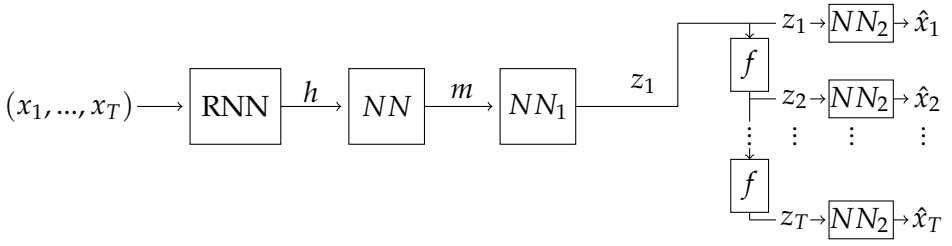


Figure 2.9: Schema of the complete recurrent VAE-GAN architecture. The dynamic function f is either a symbol for the non-linear global dynamics or the specific linear dynamical system as described in section 2.8.3. Note that in the former case the dynamics is sequence independent, i.e. independent of m.

2.9 Loss Function

In all three autoencoder types we used single pixel squared distance as the distance measure between the original and the reconstructed spectrogram snippet, that is

$$loss(x) = \|x - \hat{x}\|^2,$$

where $\hat{x} = \varphi_d(\varphi_e(x))$ is the reconstructed spectrogram snippet. However, depending on the model, other regularization terms have been added to the overall loss function.

2.9.1 MD-GAN, objective and algorithm

The MD-GAN is regularized by a loss term depending on the decision of a discriminator $D : \mathcal{X} \to [0,1]$, a neural network that tries to distinguish between real images and reconstructed images. Hence, the autoencoder is not only trying to map the reconstructed image close to the real one (in Euclidean distance) but also to fool the discriminator. The autoencoder and the discriminator are trained in a alternating manner and will eventually get closer and closer to their Nash equilibrium. In figure 2.10 we discribe the full training procedure called the manifold diffusion algorithm, from which it got its name.

Manifold Step:
1. Sample $\{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_m\}$ from data generating distribution $p_{data}(x)$.
2. Update discriminator D_1 using SGD with gradient ascent:

$$\nabla_{\theta_d^1} \frac{1}{m} \sum_{i=1}^{m} [\log D_1(\mathbf{x}_i) + \log(1 - D_1(G(E(\mathbf{x}_i))))]$$

3. Update generator G using SGD with gradient ascent:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [\lambda \log D_1(G(E(\mathbf{x}_i))) - ||\mathbf{x}_i - G(E(\mathbf{x}_i))||^2]$$

Diffusion Step:
4. Sample $\{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_m\}$ from data generating distribution $p_{data}(x)$.
5. Sample $\{\mathbf{z}_1, \mathbf{z}_2, \cdots \mathbf{z}_m\}$ from prior distribution $p_\sigma(z)$.
6. Update discriminator D_2 using SGD with gradient ascent:

$$\nabla_{\theta_d^2} \frac{1}{m} \sum_{i=1}^{m} [\log D_2(G(E(\mathbf{x}_i))) + \log(1 - D_2(\mathbf{z}_i))]$$

7. Update generator G using SGD with gradient ascent:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [\log D_2(G(\mathbf{z}_i))]$$

Figure 2.10: The training algorithm for MD-GAN cited from [21]. Note that $E : \mathcal{X} \to \mathcal{Z}$ denotes the encoder and $G : \mathcal{Z} \to \mathcal{X}$ the decoder (also called generator in this context).

2.9.2 GS-VAE loss

Although, squared pixel distance leads to blurry images we decided not to use a GAN regularization to make the image sharper. As described in section 2.8.2 we could instead chose a representative spectrogram snippet for each latent state, which was trivially a sharp and realistic image, as it came directly from real data.
In our experiments we did not match the posterior distribution $q(y|x)$ to any prior distribution as it is usually done in VAE (so there is no KL-divergence term in the loss function). This is, because the latent space is discrete and therefore we do not need to care about a scattered latent space.
However, we regularized the scale of the autoencoders output $\log(\alpha)$ as we observed that $\alpha = \exp(\log(\alpha))$ became large quickly and sometimes even went to infinity such that the training failed. Recall, that the Concrete distribution does not depend on the scale of α but only on its relative values $\pi = \alpha/\sum_j \alpha_j$. It should therefore be no hard constraint for the model to keep the $\log(\alpha)$ values reasonably small. We obtain the following loss function:

$$loss(x) = \|x - \hat{x}\|^2 + \lambda\|\alpha\|^2,$$

where $\lambda > 0$ is a hyperparameter that determines the strength of the regularization. Our experiments showed that even a small value like $\lambda = 0.01$ was enough. However, we observed that there was still a very small probability that, especially in the beginning of the training, for some data points the $\log(\alpha)$-values became too large. Hence, we applied a clipping function: $\log(\alpha) \leftarrow \min\{\log(\alpha), 10\}$.
Note that during the training we needed only the $\log(\alpha)$-values, so one might ask why we even cared about large α-values? The answer is, that we needed to calculate the assignment probabilities $p(i|x) \equiv \pi_i(x) = \alpha_i(x)/\sum_j \alpha_j(x)$ in order to evaluate the best representations for each state and this indeed involved to work with directly the α-values.

2.9.3 Recurrent Gaussian VAE-GAN loss

As already mentioned, we compared a deterministic encoder to a probabilistic one. For the former we decided to simulate the Gaussian prior distribution in the latent space by adding an Euclidean regularization $\lambda\|m\|^2$, where $m \equiv \varphi_e(x)$ and $\lambda > 0$ is a weighting factor. This makes it more likely, that the cloud of encoded data points have no big holes in it.
On the other hand, for the probabilistic Gaussian encoder we used the KL-divergence $D_{KL}(q(m|x)||p(m))$ to force the posterior $q(m|x)$ to be close to the Gaussian prior $p(m)$.
In order to achieve sharper images we applied the same trick we use in normal GANs. We trained a discriminator $D : x \to [0, 1]$ (sigmoid activation) in parallel to the autoencoder that tries to distinguish reconstructed images

from original ones. The autoencoder and the discriminator behaved like two agents that compete in a game. The former tries to sell his reconstructed images to the discriminator as real ones while the discriminator get's punished if he accepts a reconstructed image (and of course also if he refuses an original image). $D(x)$ can be interpreted as the probability that the discriminator assigns to the image x that it is an original spectrogram. Hence, if $D(x)$ is close to one it is very likely to be an original spectrogram image. The loss function of the recurrent VAE-GAN then gets an additional term that punishes the autoencoder if the discriminator can easily detect a reconstructed image. Formally, the additional term is the negative log-likelihood

$$loss_{Ad}(x) = -\lambda_{Ad} \log\big(D(\hat{x})\big),$$

where $\lambda_{Ad} > 0$ is a hyperparameter that determines how much we want to weight the discriminator.
For the discriminator we have the following negative log-likelihood loss function:

$$loss_D(x) = -\log\big(D(x)\big) - \log\big(1 - D(\hat{x})\big).$$

Note that since we are dealing with spectrogram segments of variable length, the discriminator is also equipped with a bidirectional RNN.
The overall loss function of the recurrent VAE-GAN in case of the deterministic encoder is

$$loss(x) = \|x - \hat{x})\|^2 + \lambda \|m\|^2 - \lambda_{Ad} \log\big(D(\hat{x})\big)$$

and in case of the probabilistic encoder it is

$$loss(x) = \|x - \hat{x}\|^2 + D_{KL}(q(m|x)||p(m)) - \lambda_{Ad} \log\big(D(\hat{x})\big).$$

Finally, we want to mention that the loss function of the probabilistic encoder would be the same if we choose a Gaussian probabilistic decoder. Indeed if we let $p(x|m) = \mathcal{N}(x|\varphi_d(m), 1)$, where $\varphi_d(m) \equiv \hat{x}$, we obtain for the reconstruction loss term:

$$-\mathbb{E}_{q(m|x)}\Big[\log(p(x|m))\Big] \approx -\frac{1}{L}\sum_{l=1}^{L} \log\big(\mathcal{N}(x|\varphi_d(m), 1)\big) \propto \sum_{l=1}^{L} \|x - \hat{x}\|^2 + const.$$

Therefore, we argue that if we would upgrade our decoder to a Gaussian probabilistic decoder we can expect similar results. Hence, if we would do that in a future work it might be more interesting to learn for example a mixture of Gaussian instead.

2.10 Hidden Markov Models

In order to model the brain motor control area we need a sequence generator that can learn to produce sequences of motor activations (latent variables).

We decided to use a HMM as it is a powerful sequence model but simple enough in order to interprete its parameter. Furthermore, there is good evidence that a HMM is well suited to approximate birdsong dynamics [9].
The HMM is a very simple generative Bayesian sequence model. In comparison to more powerful "black box models" like RNNs, HMMs allow us to interpret the model parameters (like the state transition matrix) themselves. There is a quick refresher on HMMs in appendix A.

2.10.1 Continuous Gaussian observations

The observations z_t (our latent variables) live in the continuous space $\mathcal{Z} = \mathbb{R}^d$. Every discrete hidden state $h \in \mathcal{H}$, where $|\mathcal{H}|$ is the total amount of hidden states in the HMM, is responsible for a multivariate Gaussian distribution on $\mathbb{R}^d$ with the corresponding mean vector and covariance matrix from the set $\{\mu_h, \Sigma_h\}_h$ that has to be inferred from the data. The likelihood of a sequence of observations $(z_1, z_2, ..., z_T)$ given a sequence of hidden states $(h_1, h_2, ..., h_T)$ factorizes like

$$p(z_1, ..., z_T | h_1, ..., h_T) = \prod_{t=1}^{T} p(z_t|h_t) = \prod_{t=1}^{T} \mathcal{N}(z_t|\mu_{h_t}, \Sigma_{h_t}).$$

In our work, the continuous observation space $\mathcal{Z}$ is the latent space of the MD-GAN or the recurrent Gaussian VAE-GAN.

2.10.2 Discrete observations

When the observation space $\mathcal{Z}$ is discrete the emission $p(z|h)$ is a PMF over $\mathcal{Z}$. Using the one-hot vector notation, as introduced in appendix A, we can write

$$p(z_t|h_t, B) = z_t^\top B h_t, \;\; 0 \leqslant B_{ij} \leqslant 1, \;\; \sum_i B_{ij} = 1,$$

where B is a $|\mathcal{Z}|$ by $|\mathcal{H}|$ matrix. The joint distribution over the observables given the hidden states factorizes in the following manner:

$$p(z_1, ..., z_T | h_1, ..., h_T) = \prod_{t=1}^{T} p(z_t|h_t) = \prod_{t=1}^{T} z_t^\top B h_t.$$

The amount of hidden states $|\mathcal{H}|$ is a hyperparameter of the model that influences its complexity and power.
We used HMMs with discrete emissions when we trained it on discrete latent sequences from the discrete latent space of the GS-VAE.

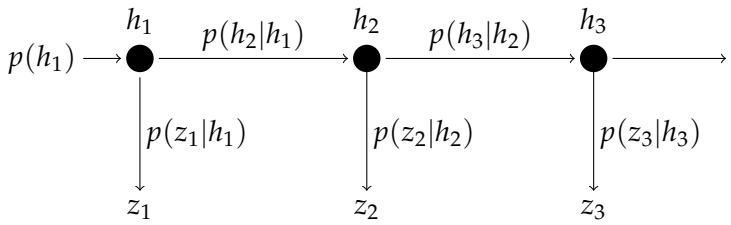


Figure 2.11: Illustration of the HMM, that is, a Markov chaintogether with state dependent emission probabilities.

2.10.3 Training

We optimized the HMMs using the Baum-Welch algorithm [41], an iterative updating scheme, where it is guaranteed that the data likelihood becomes larger (or equal) in every iteration step. The Baum-Welch algorithm belongs to the large class of expectation–maximization (EM) algorithms. A precise description and explanation of the Baum-Welch algorithm can be found in the book "pattern recognition and machine learning" by Christopher M. Bishop [22]. While convergence of the algorithm is guaranteed, the converged model can be far from optimal. One way to overcome this problem is to run the algorithm many times with different initial parameters and then pick the model with the larges likelihood. In our experiments where we use HMMs with a lot hidden states trained on a quite large data set this was too time consuming. We therefore relied on the assumption that the more degrees of freedom a network has, it typically becomes less likely to end up at a bad local optimum as most of them are sufficiently good.

Chapter 3

Results

3.1 MD-GAN

In [1] we studied the MD-GAN in detail and found that it indeed generates realistic reconstructions, see figure 3.1. Our analysis has shown that a latent space of 16 dimensions is good enough for our purpose. Furthermore, we worked with spectrogram snippets corresponding to 64 milliseconds (16 columns, 1 column = 4 ms). Since the training of the MD-GAN is not part of this thesis, we will not go into further details here.

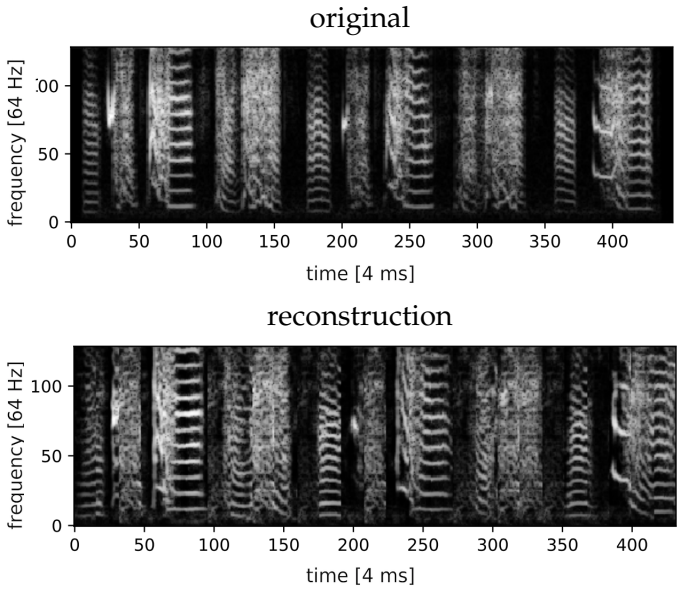


Figure 3.1: Comparison of an original spectrogram and its MD-GAN reconstruction. The fixed length of the spectrogram snippets is 16 columns (64 ms).

3.1.1 Sequence learning with HMM-MDGAN

We used the MD-GAN to encode bird songs into sequences of 16 dimensional latent variables. We treated the encoded z variables $(z_1, ..., z_T)$ as observations of a stochastic process. We then trained HMM models on collections of such sequences.
In this experiment we mainly wanted to estimate the number of hidden states needed to explain the observations for different learning stages. Hence, we divided the data set into subsets of approximately equal age. Since the number of sequences collected on each day of the experiment varied heavily (especially for the very early days we could not record so many sequences) we decided to merge days up to a fixed number of 6000 sequences. Such a large number was necessary in order to make sure that the HMMs did not overfit the training data (as they had a lot of hidden states). Finally, this resulted in seven merged data sets:

- merge 0: from day 0 to day 9
- merge 1: from day 10 to day 12
- merge 2: from day 13 to day 16
- merge 3: from day 17 to day 19
- merge 4: from day 20 to day 22
- merge 5: from day 23 to day 25
- merge 6: from day 26 to day 28

Note that day 0 does not mean that the pupil is 0 days old but that it is the first recording day (day 45 after birth) as described in section 2.1. For every merged data set we learned HMM parameters for several different numbers of hidden states from 40 up to 200.
Since the computational complexity scales quadratically in the number of hidden states, we performed an initial experiment, where we trained HMMs with only up to 100 hidden states. For every choice of number of hidden states, we trained models with different Gaussian covariance types. These were: diagonal, spherical (proportional to identity covariance) and tied (one full covariance matrix shared among all states). In these preliminary experiments, we found that the data log-likelihood of the tied type soon saturated as the number of hidden states increased and the likelihood of the spherical type was much lower than the other two. Therefore, we decided to focus our efforts on the diagonal type for the subsequent experiments, where we trained models with more than 100 hidden states.
The more hidden states a model has the more parameters (degrees of freedom) it has to fit the data. In order to have a trained model that not only performs well on the training data but also on unseen held-out data we

needed to make sure that the model did not overfit the data. Hence, we used the common hyperparameter tuning approach called cross-validation, that is, comparing the likelihood of the training data against the likelihood of the hold-out test data. We normalized the log-likelihood, that is, for a given sequence we divided its log-likelihood by its length. This compensates for the fact that a data set consisting of larger sequences on average is biased towards smaller (unnormalized) log-likelihood. We found that we used enough training data, becuase the models generally did not overfit the data, for every choice of numbers of hidden states, see figure 3.2.

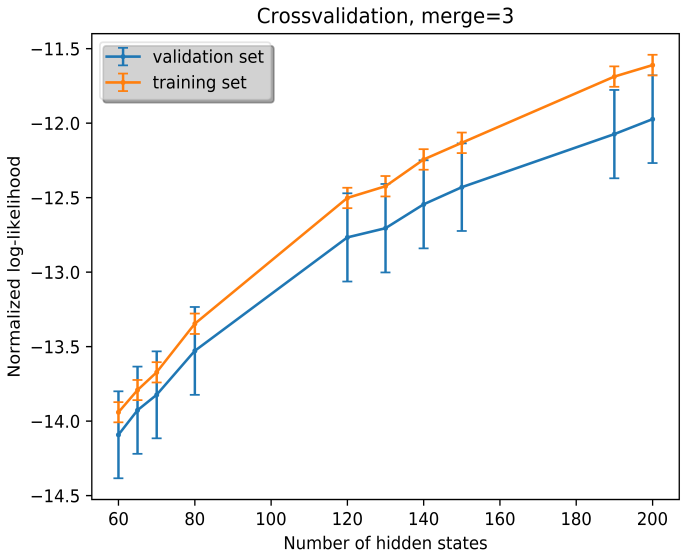


Figure 3.2: The figure shows crossvalidation between training and test likelihood of an HMM using the merge 3 (day 17-19). We see that the model does absolutely not overfit as the likelihood of the unseen test data is always increasing.

Next we wanted to figure out how many hidden states were necessary for a good model. Of course, one could argue that as long as the model was not overfitting the data we should allow the model to have as many degrees of freedom as possible. But we should keep in mind that adding many more hidden states than effectively needed could produce a lot of states that have very low transition probabilities such that they never occur in practice and therefore, are not usable for further analysis. Also, we wanted to avoid learning models that need a prohibitively long time to train. Therefore, we checked how many states are actually "active" for a given number of hidden states in the model. We did that by sampling very long sequences several times and counting the number of states that were visited at least once (we then call this the number of active states). In figure 3.3, we see that for the

different merged data sets, the number of states that have been activated at least once during the sampling experiment. With the age of the animal the number of active states decreases. It comes as little surprise that older days have much less variance in the state space than younger ones because zebra finch adults produce very stereotyped songs. This confirms that it is not helpful to use up to 200 hidden states for our models and that it might be beneficial to use more hidden states for younger pupils than for older ones. There is evidence that there is a correspondence for this observation. During song development, the produced code of the neural motor activations in the song nucleus HVC becomes sparser [30].

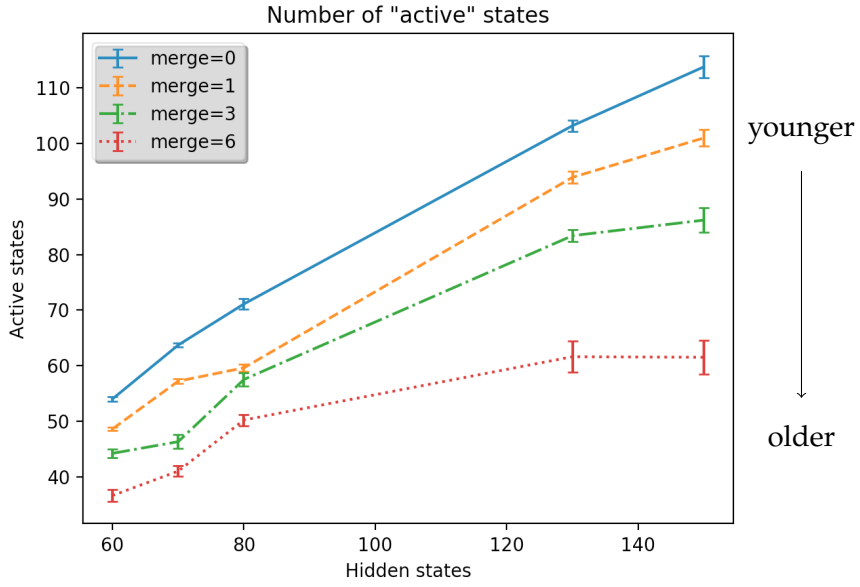


Figure 3.3: The curves show the number of states that are effectively used given the number of hidden states available. We see for example for merge 6 that having more than about 100 hidden states does not really help. The additional hidden states have an ultra small probability of occurrence. Hence they seem to help only to capture the small likelihood of outliers.

Besides crossvalidation there are also other more sophisticated criterions for model selection. One of them is the Bayesian information criterion (BIC) [31]. It adds a penalty term to the negative log-likelihood which accounts for the number of parameters in the model and is formally defined by

$$BIC = k\log(n) - 2\log(\hat{L}),$$

where $\hat{L}$ is the data likelihood, n the number of data points the model is trained on and k the number of parameters of the model. In our case, we counted every point in a sequence as a data point. The BIC recomends to

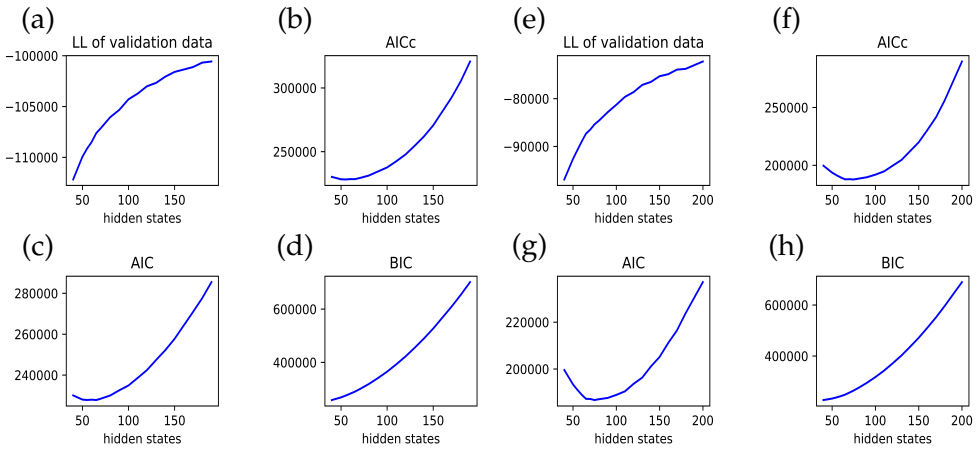
select the model with the smallest BIC value.
A very related criterion is the Akaike information criterion (AIC) [32] defined by

$$AIC = 2k - 2\log(\hat{L}).$$

In comparison to BIC it is independent of the number of data points and punishes model parameters less heavily. There is also a regularized version called AICc that should be preferred over AIC when the number of data points is small [29]. It makes the following correction

$$AICc = AIC + \frac{2k^2 + 2k}{n - k - 1}.$$

In figure 3.4 all the discussed criterions are pictured for the early merge 0 and the late merge 6. While the validation log-likelihood curve (LL) tells us that we could use even more hidden states, the AIC and AICc vote for about 60 and the BIC for even less than 50 hidden states. We therefore have an indication that more than 100 hidden states are not necessary.



(a)-(d): merge 0 (day 0-9) (e)-(h): merge 6 (day 25-28)

Figure 3.4: Model selection: A comparison of different criterions.

The remaining question was: Why did the addition of hidden states to the HMM strictly increase the data likelihood although the number of active states did not significantly increase?
We suspected that similar spectrogram snippets did not cluster together well enough in their latent space coordinates. Note that when we fit a Gaussian to the data, we restrict the space of all possible distributions massively. Essentially, we assume that the distribution is unimodal and has several symmetry properties. The trained encoder network of the MD-GAN is biased

to map into a region close to zero due to the diffusion step (figure 2.10). Indeed, when we look at the data it is well clustered about zero with no big "holes" apart from a few outliers, figure 3.5. However, the encoder is not constrained to map similar points in the image space to similar points in the latent space, that is, the metric in the image space might be significantly altered (as this is actually occuring because encoders must compress high dimensional data into a low dimensional space). Hence, we might find that the latent space is heavily scattered such that we would need more flexible emission distributions like a mixture of Gaussian in order to represent similar points with the same hidden state. In order to get more insight into this issue, we plotted several projections (i.e. randomly chosen 2-D subspaces of the full 16-D latent space) of the diagonal Gaussian emissions of the trained models together with some randomly drawn latent points, see figure 3.5. We see that in models with only a few hidden states there were a lot of outliers not captured by any Gaussian. For models with a lot of hidden states, the additional Gaussians seemed to be used to fit a few of the outliers where their corresponding hidden state had only a very small transition probability. Of course, this does not help us generate better spectrograms.
A scattered latent space means that neighboring points do not necessarily look similar. Hence, when we sample from a Gaussian, we can not expect to produce similar spectrogram snippets. This could explain discontinuities when we concatenate these spectrogram snippets.

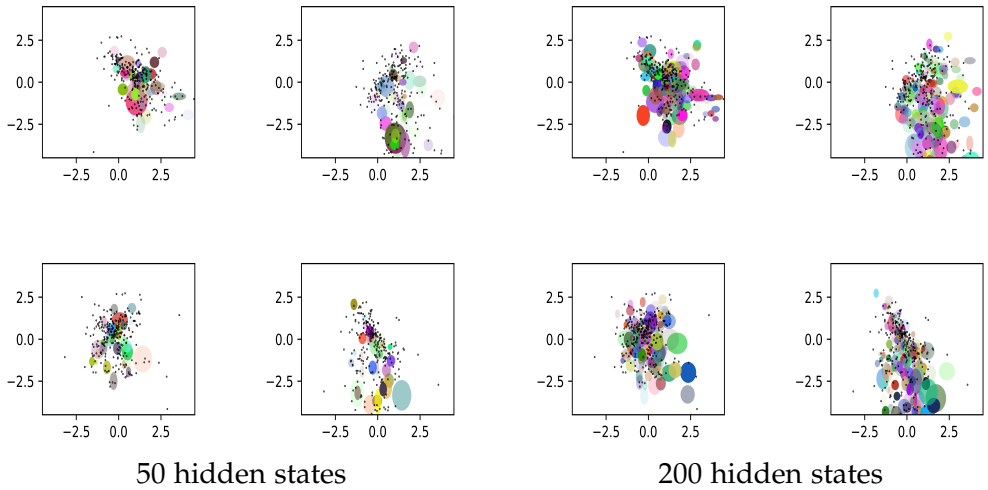


Figure 3.5: Learned Gaussian emissions of a model with 50 hidden states (left) and 200 hidden states (right). Both are trained on the data of an older pupil (merge 6). The ellipses show the interior of $1/2$ standard deviation given by the diagonal elements of the diagonal covariance matrices. The projection planes and the coloring are chosen randomly.

When we used HMMs to generate sequences of spectrogram snippets the image showed discontinuities at the positions where two spectrogram snippets were concatenated, see figure 3.6. This can be explained by a chaotically

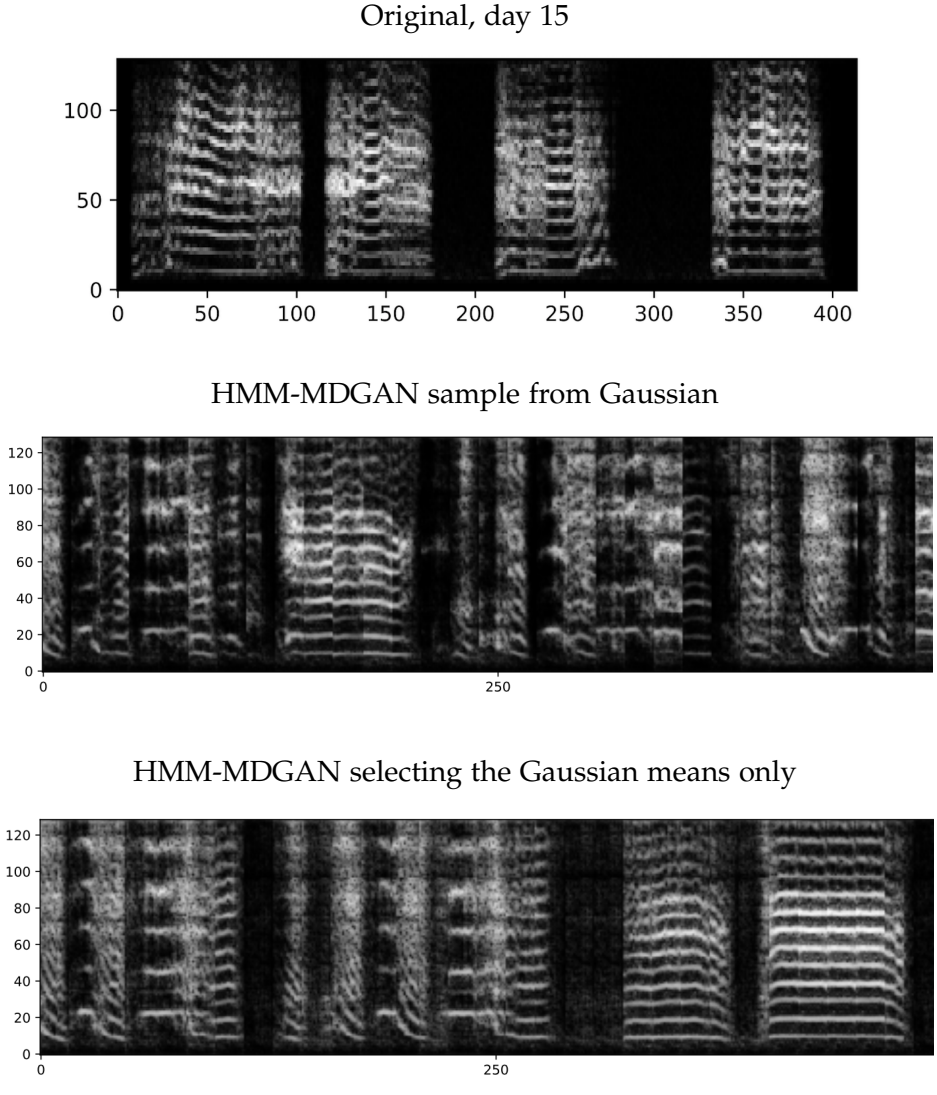


Figure 3.6: A comparison of an original spectrogram and two sampled spectrograms from a HMM-MDGAN. The spectrogram in the middle is a sampling from the Gaussian emissions and the bottom spectrogram a version where we selected only the means of the Gaussian. The MD-GAN had been trained on data of day 15. We see that simply selecting the means leads to much smoother generated spectrograms.

scattered latent space. However, if we do not sample from the Gaussian but take their means only we achieve much smoother spectrogram images, see figure 3.6 (bottom). This is indeed what we would observe when neigh-

boring points in the latent space (in Euclidean distance) do not necessarily correspond to similar spectrogram snippets in the image space.
Although this finding was somewhat disappointing, by choosing from the means of the Gaussians we have still successfully trained a generative model with realistic looking samples. One might argue that we do not sample from the complete space of all songs a pupil of the corresponding age produces but this is really not so tragic as it first sounds. Our data processing step is lossy anyway and we would be happy even with a good discrete latent representation. The important criterion for a good generator are twofold. First, it must have a high likelihood for any sequence from the data set and should also be able to generate very similar spectrogram sequences. Second, generated spectrograms and corresponding audio songs should look and sound realistic to us as human observers (recall the problem of model evaluation in the introductory part of section 2.6). While the second criterion is satisfied by choosing the means the first criterion is not necessarily. Selecting the means is a massive restriction on the space of all possible reconstructions and therefore the diversity of the generated spectrograms might not be as rich as the original spectrograms.

3.2 GS-VAE

We trained a GS-VAE on data of day 25 corresponding to 60'000 fixed length spectrogram snippets of 24 ms (6 columns). The encoder and decoder neural network each consisted of three hidden layers with 1000, 1000 and 300 units. All layers were fully connected. We used 400 discrete nodes for the latent space $(K = 400)$. We trained the autoencoder in two rounds, the first with temperature $\tau = 1$ and the second with temperature $\tau = 0.5$. From the quality of the figures, we found it not necessary to train on smaller temperatures. In figure 3.7 we compare the relaxed-reconstruction to the argmax-reconstruction. We refer to the relaxed-reconstruction if it is a decoded sampling from the Concrete distribution (exactly how it is sampled during the training), i.e.

$$\hat{x}_{relaxed} = \varphi_d(y), \;\; y \sim q_\tau(y|\alpha), \;\; \alpha = \varphi_e(x),$$

where $\hat{x}_{relaxed}$ is the reconstructed spectrogram and y is the inferred Concrete random variable. We refer to the argmax-reconstruction if it is a decoding from the point in the latent space with the largest probability in the Concrete distribution (the maximum likelihood point corresponds to a corner of the probability simplex corresponding to the largest α)

$$\hat{x}_{argmax} = \varphi_d \left(oneHot(\underset{i}{\text{argmax}}\{\alpha_i\}) \right).$$

Furthermore, we calculated for every latent state the closest snippet as its

representation, that is, the snippet that is assigned to a specific state with the largest probability (as described in section 2.8.2). In figure 3.7, we see that the reconstructed spectrogram snippets are similar to the original ones and also look very realistic (up to minor discontinuities mainly due to discretization).

3.2.1 Sequence learning with HMM-GSVAE

Using the discrete encoding of the GS-VAE, we trained a discrete HMM with 50 hidden states on 600 birdsong sequences (corresponding to 40000 spectrogram snippets) from day 25. Our HMM implementation can be found on GitHub [35].
In order to check that the model did not overfit the data we crossvalidated the model on 100 left out sequences, see figure 3.8.

In figure 3.9 we see two sampling from the HMM. The upper spectrogram was directly sampled from the HMM (using the learned initial probability distribution of the first hidden state $p(h_1)$) and for the bottom spectrogram we used an initial seed $(x_1, x_2, ..., x_t)$ of 480 milliseconds from an original birdsong and sampled the first hidden state at time $t+1$ from the conditional distribution $p(h_{t+1}|x_1, x_2, ..., x_t)$. The fixed length of a single spectrogram snippet was 6 columns corresponding to 24 ms. We see that while the dynamic of the spectrogram looks good there are discontinuities between the concatenated snippets.

We used viterbi decoding to visualize the model accuracy. In this method, the most likely sequence of hidden states for a given observation is calculated and then observations are generated by sampling from the corresponding Gaussians. The similarity between the observed spectrogram and the viterbi decoding indicates how well a model can reflect a given sequence, see figure 3.10.

3.3 Recurrent VAE-GAN

Recall that the recurrent VAE-GAN can encode spectrogram segments of arbitrary length to a continuous latent space.
We trained the recurrent VAE-GAN with a determenistic and also with a Gaussian probabilistic encoder on 4000 non-silent segments from day 25. The segments were between 24 and 240 milliseconds long. The RNN recognition network of the encoder was trained with one layer and 100 hidden units (for both directions). The last hidden states (forward and backward direction concatenated) were mapped to a 15-D latent space $\mathcal{M}$ by a neural network with two hidden layers with 1000 and 500 units. The first decoder neural network that mapped the m-variables to the 5-D $\mathcal{Z}$-space had also

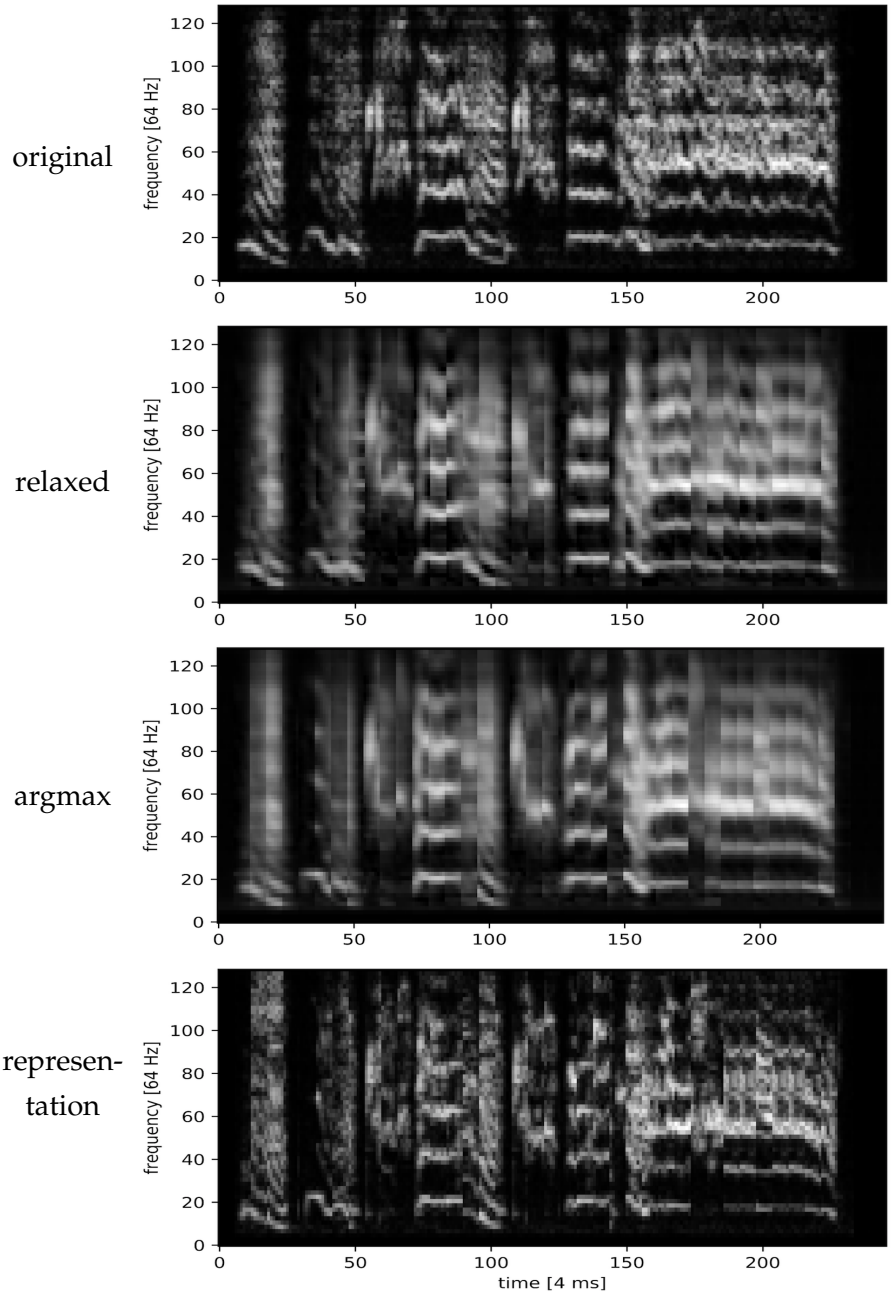


Figure 3.7: Comparison of different reconstruction schemes of the GS-VAE. The second spectrogram from the top shows a relaxed-reconstruction, the third shows an argmax-reconstruction and in the bottom spectrogram we used the representation technique (all methods are described in section 3.2). The original spectrogram is from recording day 25.

two hidden layers with 500 and 500 units. The second decoder neural net-

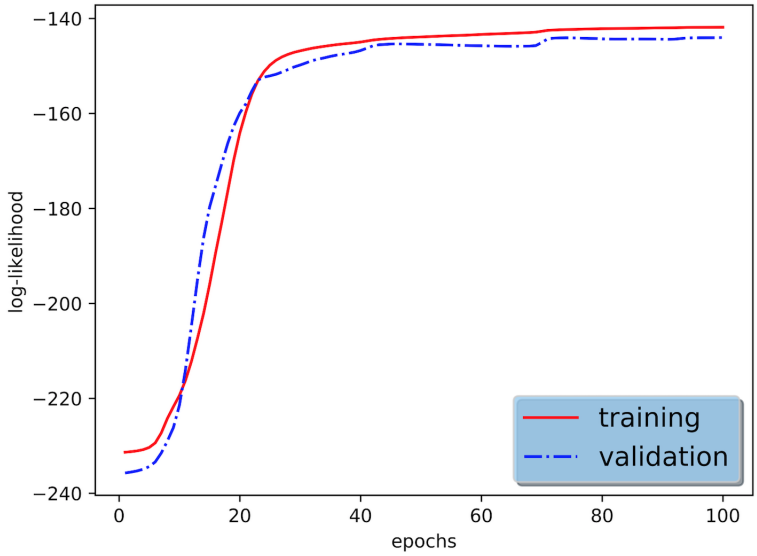


Figure 3.8: Crossvalidation of the discrete HMM-GSVAE. The validation curve suggests that the HMM-GSVAE did not overfit the data. The evaluation of the log-likelihood was after every epoch.

work that mapped the dynamic variables back to the space of spectrogram images consisted of three hidden layers with 500, 1000 and 1000 units. All layers were fully connected.
In figure 3.11 we compared original spectrogram segments to reconstructions without adversarial regularization. We see that while the autoencoder learned the dynamics in the segments quite well, the reconstructed images are very blurry.

Recall from section 2.9.3 that in order to achieve sharper images we trained a discriminator $D : x \to [0,1]$ in parallel to the autoencoder that tried to distinguish reconstructed images from original ones. The discriminator had a similar RNN recognition network as the recurrent autoencoder, namely a one layer GRU with 100 hidden units. Recall that the loss function of the VAE-GAN got an additional term that punished the network depending on how easily the discriminator would distinguish a reconstructed image from an original one. The adversarial regularization term of the recurrent VAE-GAN was

$$loss_{Ad}(x) = -\lambda_{Ad} \log\big(D(\hat{x})\big),$$

where $\lambda_{Ad} > 0$ is a hyperparameter that determines how much we want to weight the discriminator and $\hat{x}$ is the reconstructed spectrogram segment.
It is a delicate task to train a discriminator in parallel to an autoencoder. Usually, what is done when training a GAN is solving in every epoch a min-

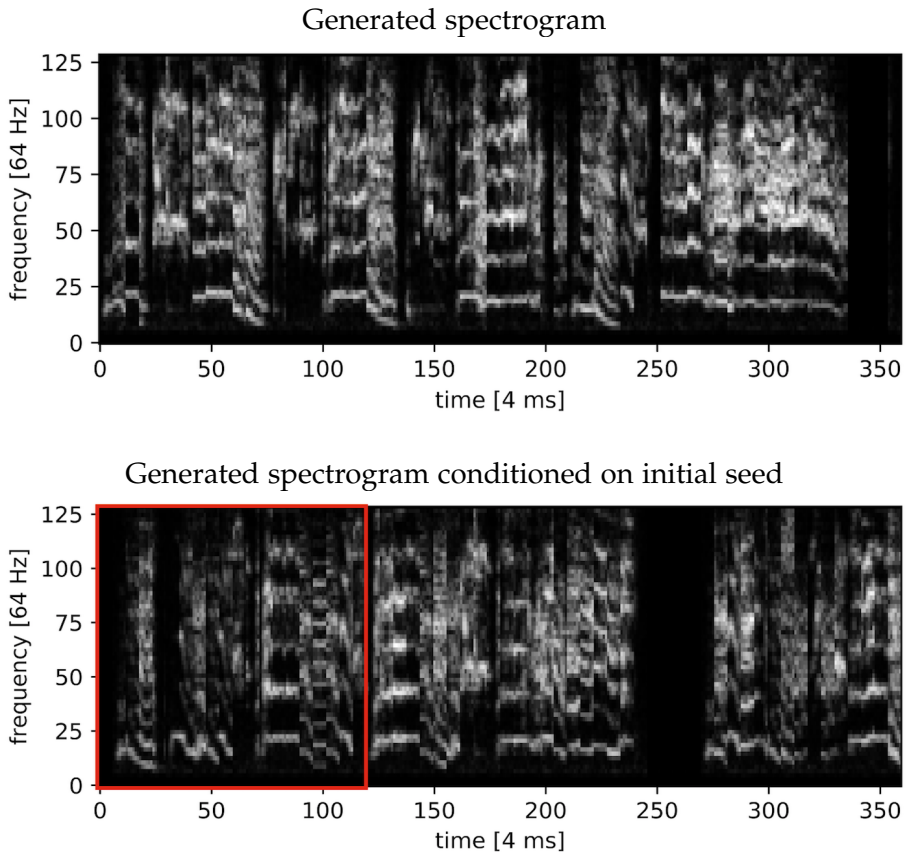


Figure 3.9: Sampled spectrograms from a HMM-GSVAE with 60 hidden states. In the lower spectrogram the generation was conditioned on an initial seed of observations $(x_1, x_2, ..., x_t)$ marked by a red window in the figure. While in the upper spectrogram we sampled the first hidden state using the initial distribution over the hidden states $p(h_1)$ in the lower spectrogram we used the conditional distribution over the hidden states given the initial observations $p(h_{t+1}|x_1, ..., x_t)$.

max problem, where one first trains the discriminator for a few iterations before training the generator once. However, this seemed to be inefficient in our case, since this had the effect that at the beginning of every epoch the discriminator loss term $loss_{Ad}(x)$ of the autoencoder was much larger than the reconstruction loss such that the autoencoder gave up its learned similarities in squared pixel distance in favour of fooling the discriminator. Hence, there is a huge computational waste in unlearning and relearning the closeness in Euclidean distance. We therefore trained the discriminator in the same minibatch-loop as the autoencoder, that is, whenever the parameters of the autoencoder got updated, the parameters of the discriminator where updated as well. Furthermore, we found it reasonable to pre-train the autoencoder without the discriminator loss ($\lambda_{Ad} = 0$), because the sharpening of the image can be seen as a fine-tuning step.

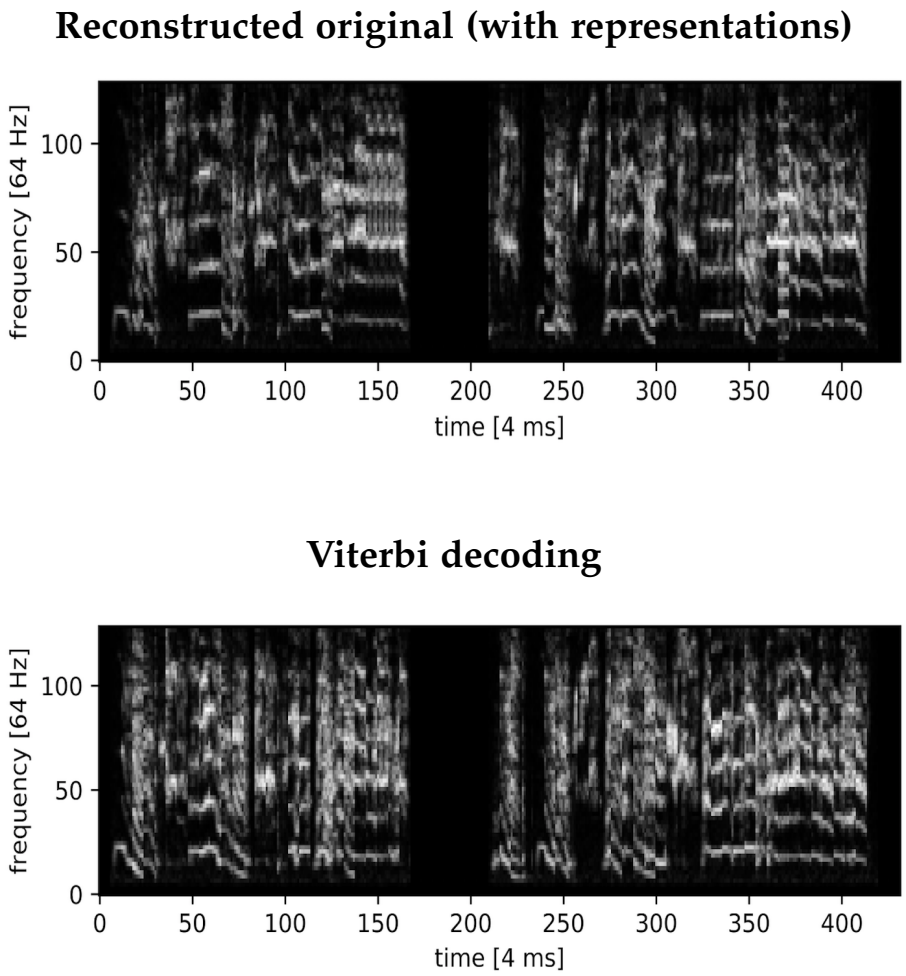


Figure 3.10: Viterbi decoding from HMM-GSVAE. For a given observation, i.e. the encoding of a real data spectrogram (upper) we calculated the most likely sequence of hidden states and sampled from those to produce the bottom spectrogram.

It is not obvious what the order of magnitude of the weight λ_{Ad} should be. If it is too small, the reconstruction stays blurry and if it is too large the reconstruction becomes sharp but not necessarily similar to the original image.
We found that a value of $\lambda_{Ad} = 1$ was a good trade-off between similarity to the original image (in Euclidean distance) and sharpness. Note that such a value depends on the scale of the image data. To be precise, let's assume that the spectrogram snippets were arbitrarily rescaled during the preprocessing by some factor $\beta > 0$ such that $\|x\|^2 \leftarrow \|\beta x\|^2 = \beta^2 \|x\|^2$. From that it becomes clear that we are actually looking for a good ratio between the weights of the two loss terms, that is, λ_{Ad}/β^2.
By simultaneously training the autoencoder and the discriminator, the prob-

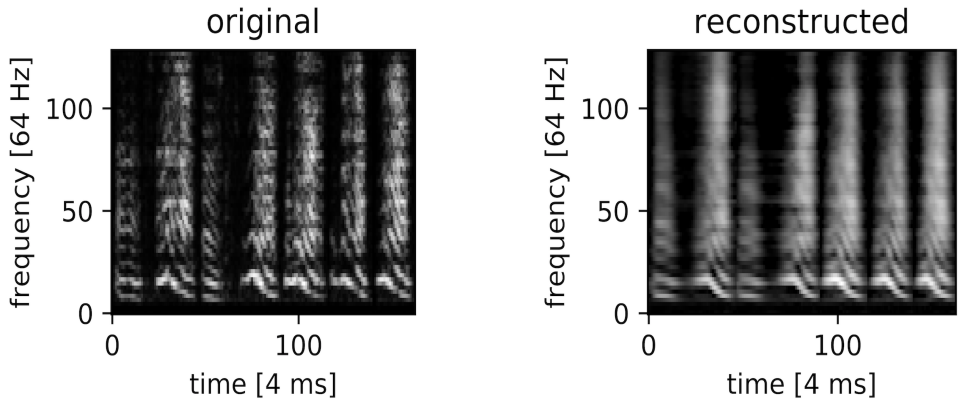


Figure 3.11: Comparison of original and reconstructed spectrogram segments of the recurrent VAE (no adversarial loss) with a global non-linear hidden dynamic. Unfortunately, the squared distance loss alone leads to blurry images.

ability that the discriminator correctly classifies reconstructed images (as fake or real) was quite constant over the epochs. Depending on λ_{Ad} this probability was in favour of the discriminator or of the autoencoder. We also observed that when the learning rate was not sufficiently small, a kind of oscillation occurred between the winning probability of the two agents. In figure 3.12 we see how the classification probability behaved over training epochs for different λ_{Ad}. Furthermore, one curve is shown that corresponds to training with a large learning rate which resulted in oscillations.

Global non-linear vs. specific linear

What is better, a specific linear dynamical system or a global non-linear dynamical system?
To answer this question, we fixed the architecture of two recurrent VAE-GANs up to the dynamical system. While the dimensionality of the dynamic space $\mathcal{Z}$ was fixed to 5 for both the first had a global non-linear system and the second a linear dinamic system, as described in section 2.8.3. We then trained the two autoencoders with the same data (day 25) and the same adversarial regularization weight ($\lambda_{Ad} = 1$). The result favoured clearly the global non-linear dynamical system, see figure 3.14. However, the fact that the adversarial training is unstable we should ideally perform many more repetitions of the experiment with different initializations and different hyperparameters in order to make a more significant statement.

Recurrent Gaussian VAE-GAN: probabilistic encoder

Up to this point, all experiments where made with a deterministic encoder. In figure 3.15 we compare the results of a probabilistic encoder to the one

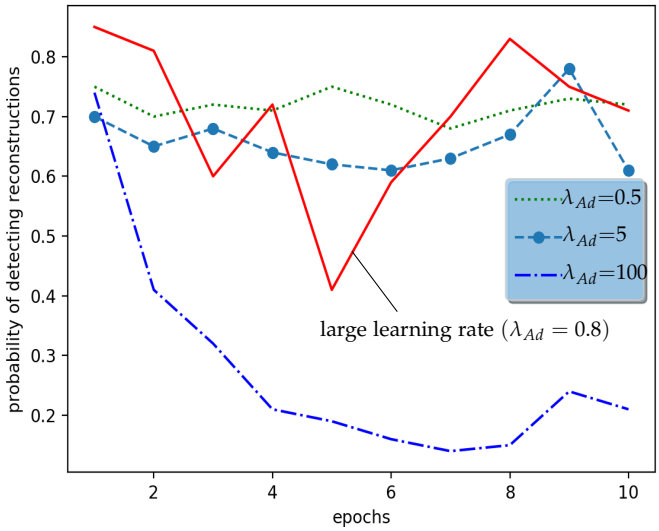


Figure 3.12: Comparison of the adversarial loss term with different weights during the training. For a small learning rate (10^{-5}) the probability that the discriminator detects reconstructed images as reals becomes approximately stable over the epochs. The larger the regularization weight λ_{Ad} the more is the agent equilibrium shifted in favour of the autoencoder. For example the blue dashed-dotted line with $\lambda_{Ad} = 100$ has a much smaller detection probability than the green dotted line with $\lambda_{Ad} = 0.5$. The red continuous line shows the described oscillation phenomenon if the learning rate is too large.

with a deterministic encoder. All other components of the recurrent VAE-GAN are left equal (same architecture as ro the deterministic encoder).

3.4 Recurrent GS-VAE

Finally, we married the GS-VAE and the recurrent VAE-GAN by taking the recurrent architecture of the recurrent VAE-GAN and the discrete latent space of the GS-VAE. The resulting recurrent GS-VAE is capable of clustering and also reconstruct spectrogram segments of arbitrary length. It was trained with the exact same architecture as the recurrent VAE-GAN apart from the map to the latent space $\mathcal{M}$ which was replaced by a 100-dimensional continuous space corresponding to the parameters of the Concrete distribution over a 99-dimensional probability simplex (namely the $\log(\alpha)$-values). Hence, the recurrent GS-VAE had 100 discrete states. It was trained on a data set of 4000 non-silent segments from day 25. The lengths of the segments were between 24 and 240 milliseconds.
Similar to the non-recurrent GS-VAE, we assigned to every latent state the most likely segment of the training data serving as a representation of that

state. However, the big difference to the non-recurrent GS-VAE was that the segments could have different lengths. Therefore the 100 representative segments for each latent states additionally needed to represent a certain segment length of that state. Figure 3.16 shows a comparison between original segments and the corresponding representative segments.

We can use the recurrent GS-VAE to cluster spectrogram segments of different length in a probabilistic manner by simply mapping the spectrogram to the discrete latent space.
In figure 3.17 we use this clustering method to investigate the latent space of the recurrent VAE-GAN. More precisely, we compared the latent space of the deterministic encoder to the one of the probabilistic encoder by looking at random projections of the latent space. We observe that due to the Gaussian prior of the probabilistic encoder the latent points are nicely distributed about zero while for the deterministic encoder this is not true. Although the latent points are close to zero due to L2-regularization of the latent space there are a lot of ”holes” and outliers. We used the recurrent GS-VAE to cluster and color the latent points.

In figure 3.18 we compare the trajectories of the dynamic space of the specific linear dynamical system to the global non-linear dynamical system, using the recurrent VAE-GAN. Again, we used the recurrent GS-VAE to color the trajectories. We see that for the global non-linear dynamical system the points on the trajectories are uniformly far away from each other due to a sigmoid output at the last layer of the neural network corresponding to the vector field. On the other hand, the trajectories of the linear dynamical system suffer from the problem that they have large ”jumps” initially and then the distance between two consecutive points become smaller and smaller towards the fixed point, which is expected because of the stability constraint. Indeed, if the sequence of the linear system converges too fast to zero the corresponding spectrogram segment becomes constant and does not show complex dynamics, indicated in figure 3.18.

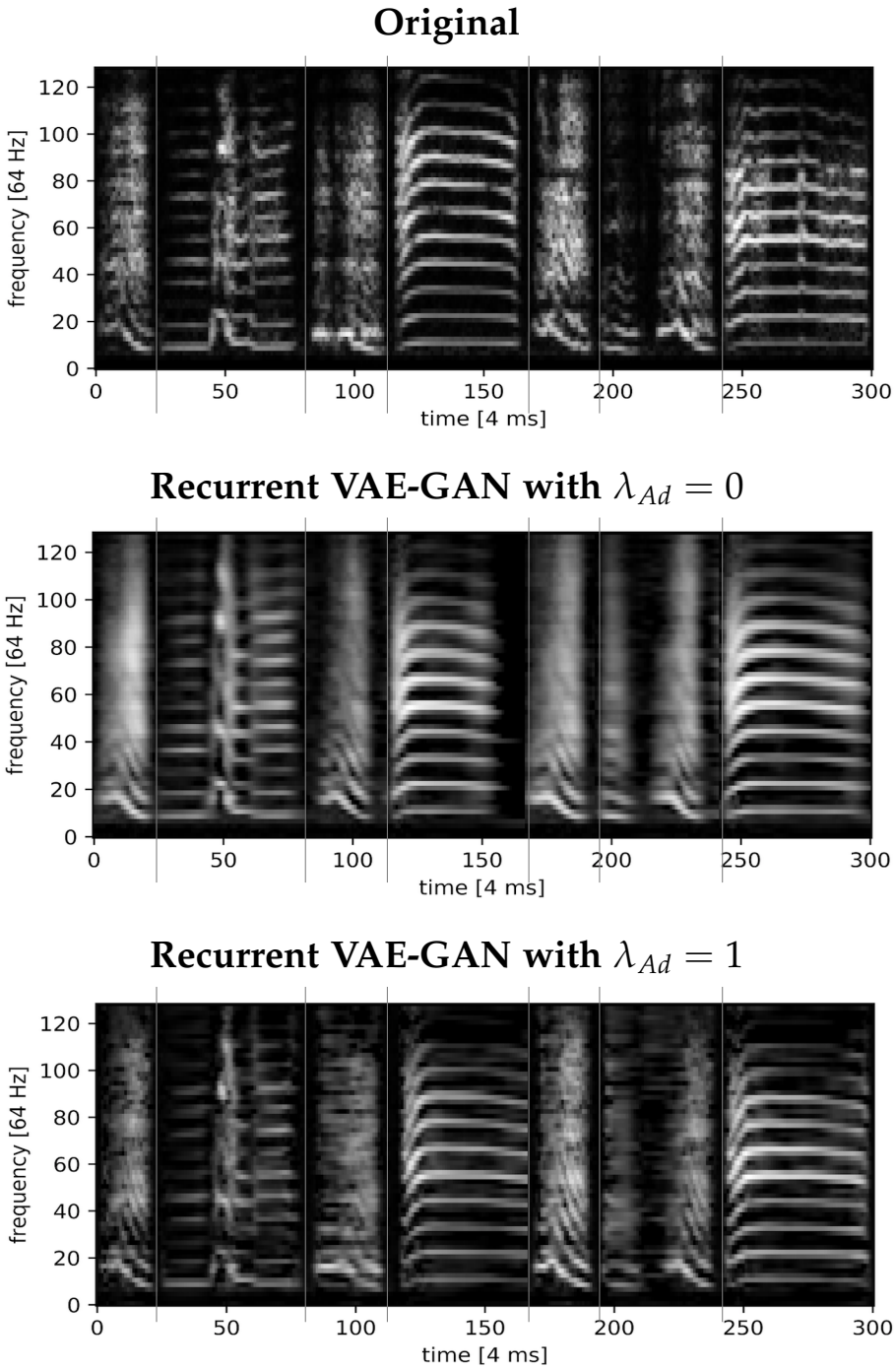


Figure 3.13: Comparison between randomly selected original spectrogram segments and reconstructions of two recurrent VAE-GANs with a global non-linear dynamical system. The middle shows a reconstruction with no adversarial regularization $\lambda_{Ad} = 0$ and the bottom spectrogram is a reconstruction with $\lambda_{Ad} = 1$. We see that the choice of $\lambda_{Ad} = 1$ indeed generates a sharper birdsong spectrogram, similar to the original (top row). The gray vertical lines indicate the boundaries between two segments. Note that theses segments are not contiguous in time, i.e. they are not making up one song and are just randomly selected.

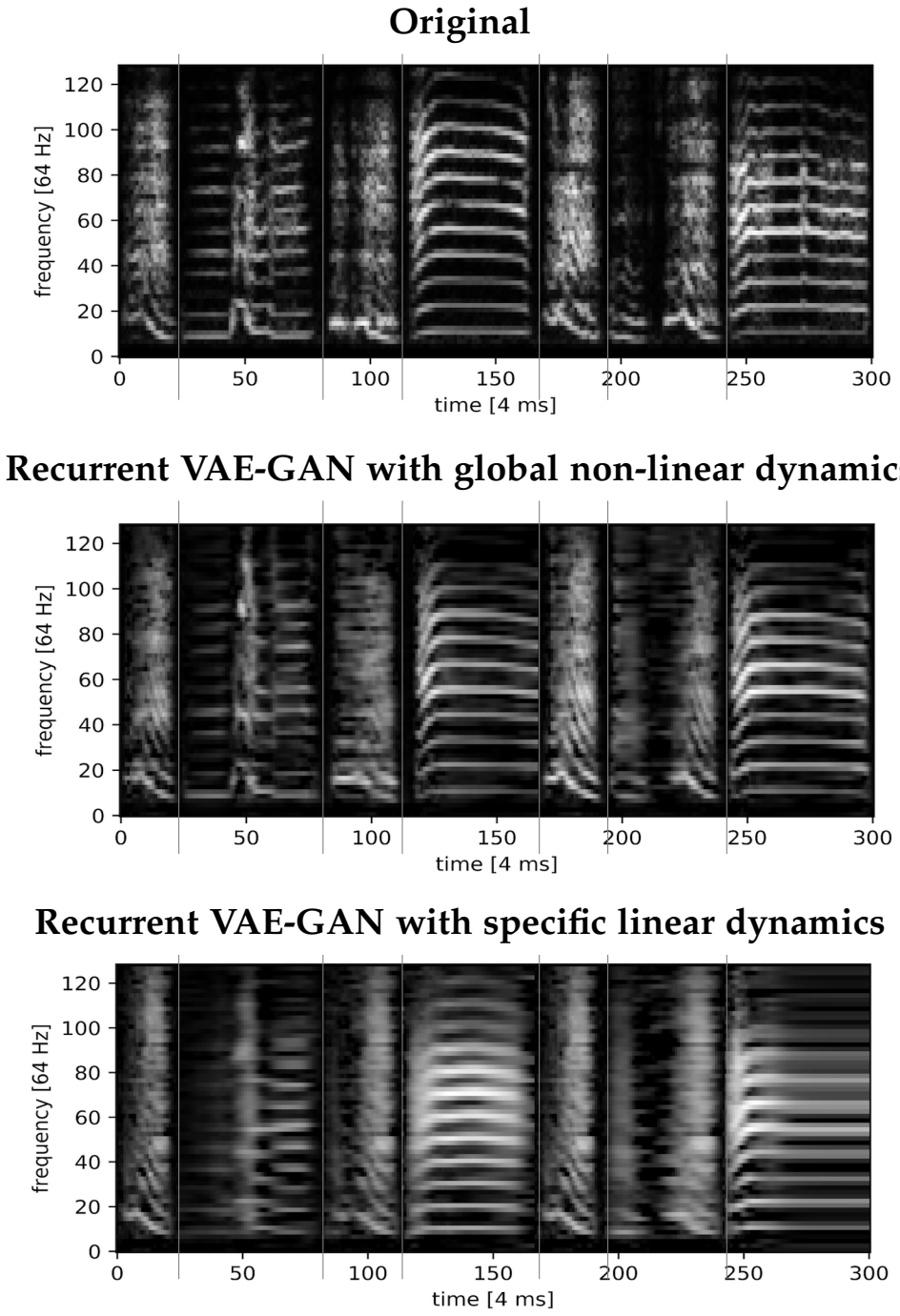


Figure 3.14: Comparison between randomly selected (not actually contiguous) original spectrogram segments and reconstructions of two recurrent VAE-GANs. The first (middle) had a global non-linear dynamical system and the second (bottom) had a segment specific linear dynamical system. We see that the recurrent VAE-GAN with the global non-linear dynamical system produces much sharper images with a better dynamic than the one with the linear dynamical system. The gray vertical lines indicate the boundary between two segments.

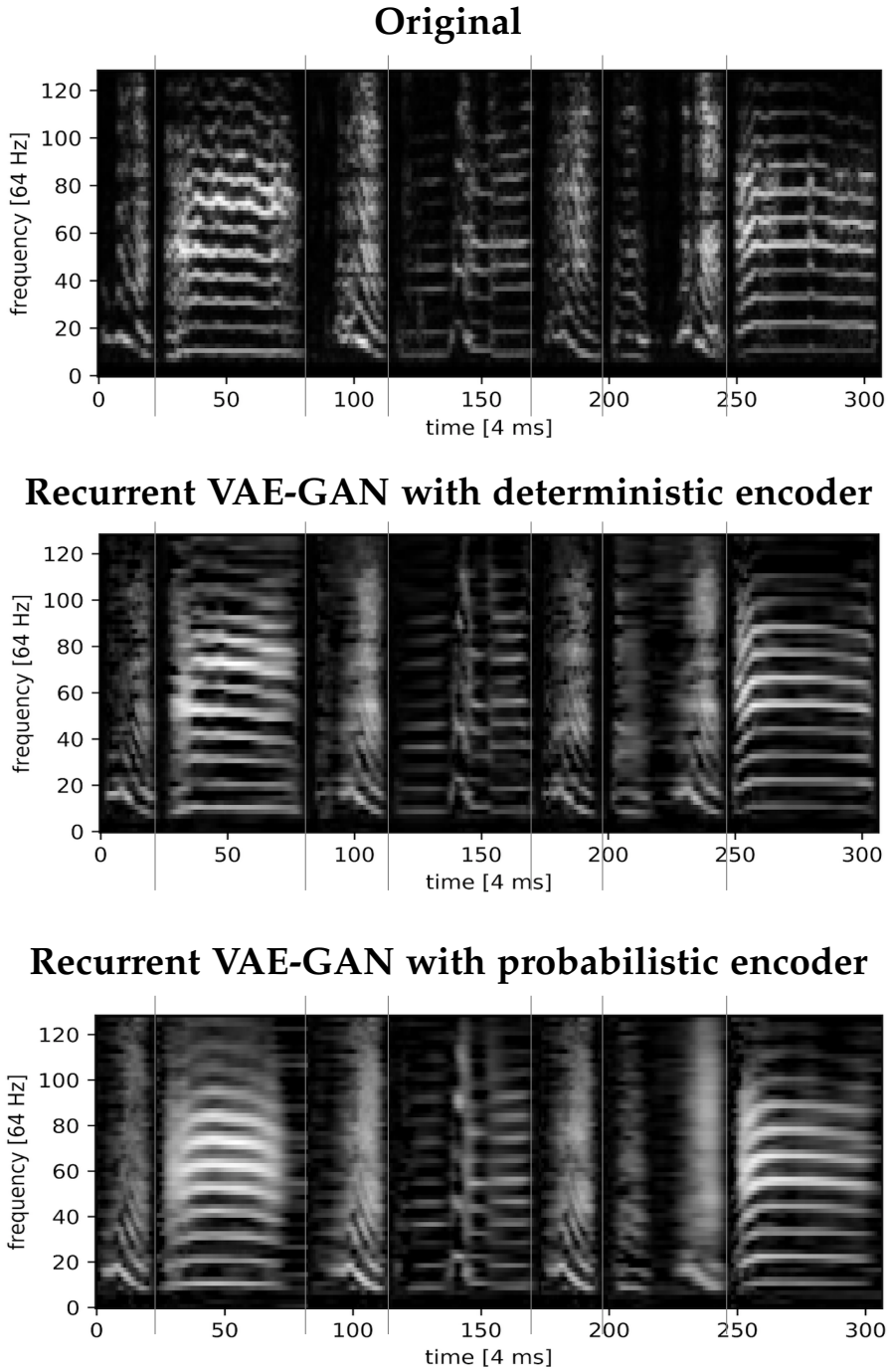


Figure 3.15: Comparison of deterministic and probabilistic encoder in the recurrent VAE-GAN. (Top) Randomly selected original spectrogram segments. (Middle) Reconstructions using a deterministic encoder and (bottom) reconstructions from a probabilistic encoder.

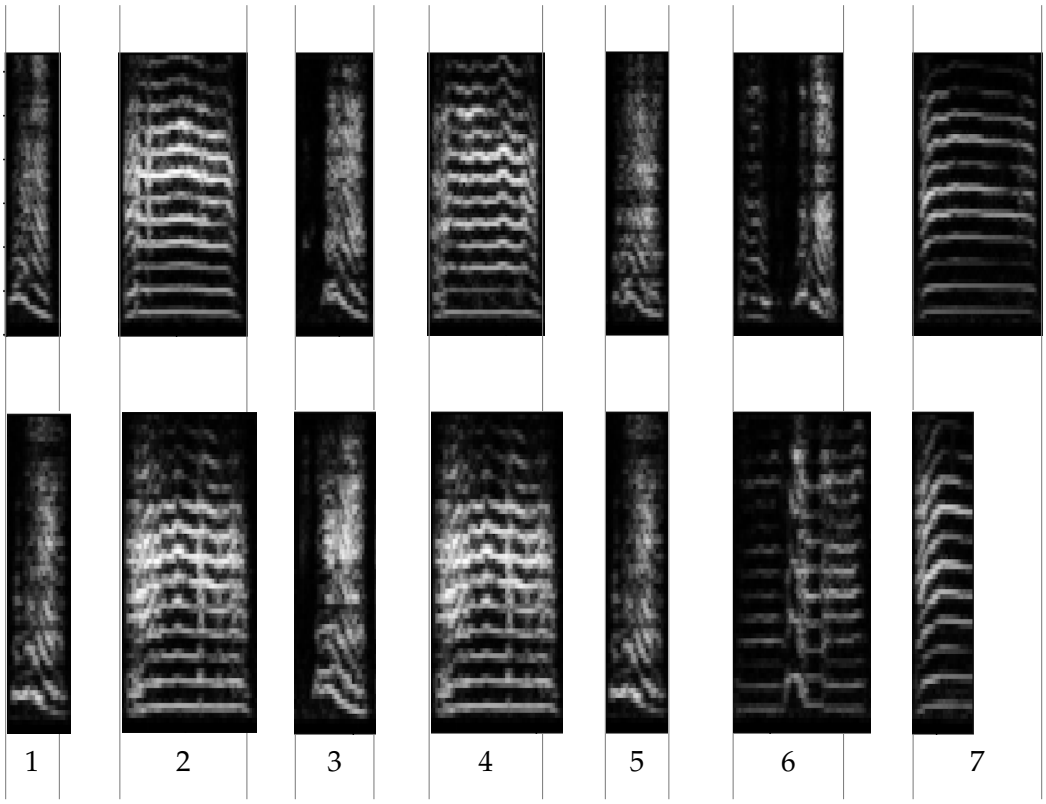


Figure 3.16: Comparison of randomly selected original (upper) and the corresponding representations (lower) assigned by the recurrent GS-VAE. The gray lines indicate that the original and the reconstructed segments are not necessarily equally long (because they are the corresponding cluster representations). Note that the representation of segment 1 and 5 is identical, because the encoder mapped them to the same discrete latent variable (they belong to same cluster).

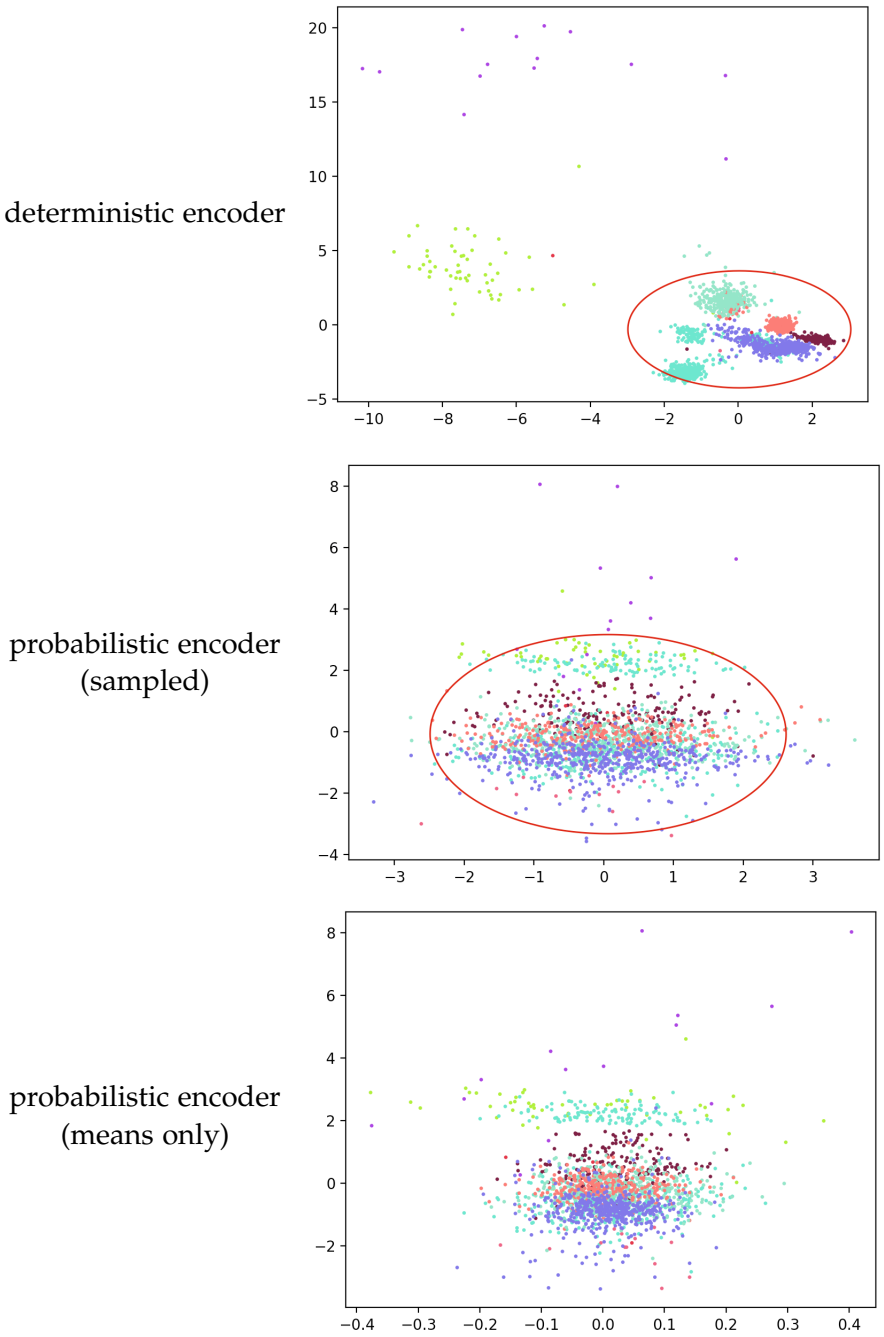


Figure 3.17: Projection of the 15-D latent space $\mathcal{M}$ on to two randomly selected axis. We compare the distribution of latent points in case of the deterministic encoder (upper) and the probabilistic Gaussian encoder (middle and bottom) of the recurrent VAE-GAN. The recurrent GS-VAE was used to cluster the segments. The red ellipses (distortion due to different scales of the axis) about zero should help to notice how well the data is distributed about zero. By comparing the middle and the bottom figure, one get's an idea of how much variance comes from the sampling in case of the Gaussian probabilistic encoder as we plotted only the means of the Gaussian posterior $q(m|x)$ in the bottom figure.

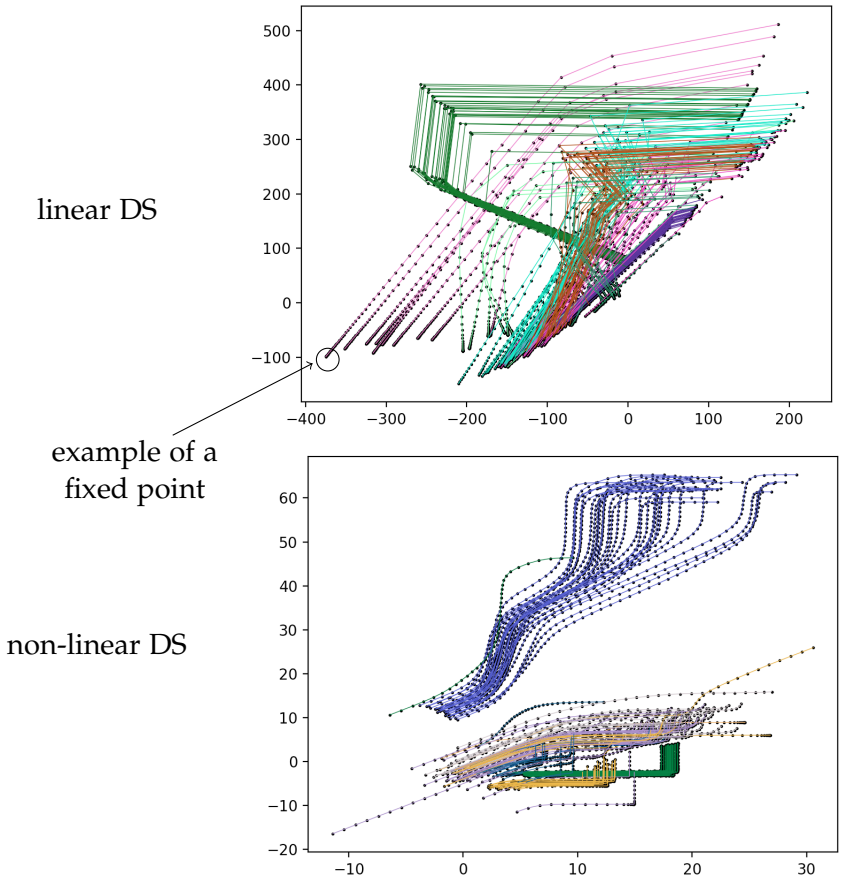


Figure 3.18: Projection of the 5-D dynamic space on to two randomly selected axis. We compare the trajectories in the dynamic space for a linear dynamical system (top) and a non-linear dynamical system (bottom). The recurrent GS-VAE was used for the coloring. We see that in case of the non-linear DS the distances between two consecutive points are evenly distributed. On the other hand, the trajectories of the linear dynamical system converge to a fixed point (marked in the upper figure).

Chapter 4

Discussion

The main goal of this thesis was to develop a generative two stage model for birdsong generation. The idea was to train an autoencoder on the complete data set, that is, over all recording days. Hence, the decoder of such an autoencoder would be a highly non-linear map from the low dimensional space of motor activations to the high dimensional space of spectrogram snippets. Therefore, the decoder would essentially model the vocal organ, syrinx. We encoded birdsong spectrogram segments to get approximations of motor activations. Finally, our aim was to learn a Bayesian sequence model, namely a HMM that could use the encodings for learning and thus generate arbitrarily long birdsong-like spectrogram sequences using the learned decoder. We experimented with various types of autoencoder models, which could represent both fixed length and arbitrary length spectrogram segments, were deterministic or probabilistic in nature and were trained using different types of loss functions, each with its own advantages and disadvantages.

4.1 MD-GAN

Using a MD-GAN, trained in a previous project, we have shown that first order HMMs are capable of learning good birdsong dynamics. Our analysis has shown that about 100 hidden states are enough to reflect the dynamics of a birdsong from the very first recording days when songs exhibit a lot of spectro-temporal variability. Our experiments have shown that towards the later learning stages, the required number of hidden states gradually drops. One qualitative issue with the output was that the generated spectrograms suffered from strong discontinuities between two concatenated spectrogram snippets. We identified the problem as stemming from a highly scattered latent space that not suited to train HMMs with Gaussian emissions. The many outliers and the dissimilarity between neighboring latent points re-

sulted in a significant variance in the spectrogram snippets when sampled from a Gaussian of a givenhidden state.
An additional drawback of the MD-GAN was that it could only encode spectrogram snippets of a fixed length. This forced us to crop the spectrogram at (arbitrary) positions which did not necessarily reflect "real state transitions" of the birds neural motor control nuclei (such as HVC). We believe, this fixed size cropping scheme resulted in additional difficulty for the HMM-MDGAN combination to learn smooth birdsong dynamics and hence, we motivated the use of recurrent networks and continuous dynamical systems based autoencoders.

4.2 GS-VAE

An alternative solution to overcome the problem of a scattered latent space was to train an autoencoder with a discrete latent space because it would circumvent the problem of continuity and neighborhoods in the latent space. We successfully trained a GS-VAE that can encode spectrogram segments of fixed length to a discrete latent space. Trained on the data of recording day 25, we demonstrated how we can use the GS-VAE to train discrete HMMs and also how to use it as a probabilistic clustering method. Finally, we added a RNN and a dynamic system to the GS-VAE and achieved that the resulting recurrent GS-VAE could process spectrogram segments of arbitrary length.

During our experiments, we found that the GS-VAE used only about 70% of the available latent states (the others could be "noisy states"). It would therefore be beneficial to think of a technique that forces the model to use all the available latent states. One way to do that is to add a KL-divergence regularization to the loss function:

$$REG(x) = D_{KL}(q_\tau(y|\alpha(x))||p_\tau(y|\alpha_p)),$$

where α_p is some unnormalized prior distribution. However, this regularization, motivated by the loss function of the VAE, is criticized [39], because it does not do exactly what we want. By minimizing the aforementioned KL-divergence over N samples drawn from $x \sim p_{data}(x)$ we eventually force the expectation of the posteriors to be close to the prior, that is,

$$REG_{tot} = \frac{1}{N}\sum_{n=1}^{N} REG(x_n) \approx \mathbb{E}_{p_{data}(x)}\left[D_{KL}(q_\tau(y|\alpha(x))||p_\tau(y|\alpha_p))\right].$$

What we actually want is that the continuous mixture

$$q_\tau(y) = \mathbb{E}_{p_{data}(x)}\left[q_\tau(y|\alpha(x))\right]$$

is close to the prior $p_\tau(y|\alpha_p)$. We will dive more into this issue in section 4.3 when we discuss an alternative regularization to the KL-divergence term of

the VAE-GAN loss function.
For the GS-VAE, we propose to achieve this by estimating the marginal probability distribution over the latent states in every batch and then regularize it via a KL-divergence term. Let N be the amount of data points in a given batch. We then define the empirical marginal probability of the i-th latent state

$$Q_i = \frac{1}{N}\sum_{n=1}^{N} q(i|x_n) \approx \mathbb{E}_{p_{data}(x)}\left[q(i|x)\right],$$

where $q(i|x) = {}^{\alpha_i(x)}/_{\sum_{j=1}^{K}\alpha_j(x)}$ is the probability that x is assigned to cluster i. Hence, $Q = (Q_1, Q_2, ..., Q_K)$ is indeed a probability distribution over the latent states. We can then define the following regularization term

$$REG_{batch} = \lambda D_{KL}(Q||P),$$

where $P = (P_1, P_2, ..., P_K)$ is any prior distribution over the latent states, and $\lambda > 0$ is a hyperparameter to tradeoff this term with the other terms in the overall loss function. In order to force the GS-VAE to use all its latent states we could for example choose a uniform prior $P_i = {}^{1}/_{K}$, $\forall i = 1, 2, ..., K$.

There is an additional tool that is provided by the GS-VAE if we use it as a clustering method. The probabilistic cluster assignments of the GS-VAE allows us to define a distance measure between clusters. Let $q(i|x)$ be the probability that the segment x is assigned to the cluster i and let $\{x_i^{(r)}\}_{i=1}^{K}$ be the representative segments of the K clusters. Recall that these representations were defined by

$$x_i^{(r)} = \arg\max_{x \in \text{data}} \{q(i|x)\}.$$

Therefore it holds that

$$q(i|x_i^{(r)}) \geqslant q(i|x_j^{(r)}), \quad \forall i, j.$$

We can use this to define the asymmetric distance measure via the log-ratio of the probability that a representative segment is assigned to its own cluster to the probability that it is assigned to an other cluster

$$d : \{1, ..., K\}^2 \to [0, \infty], \quad d(i,j) = \log\left(\frac{q(i|x_i^{(r)})}{q(j|x_i^{(r)})}\right),$$

which we can make symmetric by taking the averages of the asymmetric pairs

$$d_s(i,j) = \frac{1}{2}\big(d(i,j) + d(j,i)\big).$$

We might use this measure for future analysis.

4.3 Recurrent VAE-GAN

Using a bidirectional RNN, we managed to encode spectrogram segments of variable length. In addition to that, a dynamical system in the decoder has allowed us to generate smooth spectrogram segments that show realistic dynamics. The global non-linear dynamic system performed significantly better than the specific linear dynamic system. We think that this is because in the latter case, for every dynamic trajectory in the $\mathcal{Z}$-space there are big "jumps" at the beginning making it hard to generate smooth reconstructions. On the other hand, the very small jumps close to the fixed point makes the reconstruction overly smooth.
Similar to the MD-GAN, we successfully used an adversarial loss in order to generate sharper spectrogram images.
In our experiments, the deterministic encoder had a slightly better performance than the probabilistic one. However, we believe that the performance of our current deterministic encoder can be matched or even exceeded by further tuning of the hyperparameters of the probabilistic encoder. Simple projective images of the latent space have confirmed that using a probabilistic Gaussian encoder leads to a much less scattered latent space.
At this point, we must emphasize that we trained the recurrent VAE-GAN on data of day 25 only, and even in that case we needed to train the network for quite a long time (about 2-5 times longer than without the adversarial regularization, 3-5 days on one CPU core). This is due to the adversarial component which makes the training more unstable than in the case of a normal VAE. Training on a larger data set over many days might therefore be a significant challenge, especially because the spectrograms of recorded songs of the early days are very variable which could make it prohibitively difficult for the dynamical system to learn robustly.

In the following paragraph we want to propose an improved version of the recurrent VAE-GAN.
As briefly mentioned in section 4.2 the KL-divergence regularization of the VAE loss forces the per data point approximate posterior $q(z|x)$ to be close to the prior $p(z)$. Therefore, the posteriors are biased to overlap at the modes of the prior, which leads to problems with reconstructions. In contrast, if we force the continuous mixture $q(z) = \mathbb{E}_{p_{data}(x)}[q(z|x)]$ to be close to the prior $p(z)$, the latent codes of different data points have a chance to stay away from each other, promoting a better reconstruction and a more disentangled representation [39]. One way to remedy the issue is to use a GAN objective to penalize the discrepancy between $q(z)$ and $p(z)$. In [38], an adversarial autoencoder (AAE) was presented, whose only difference to the VAE was that the KL-divergence regularization was replaced by such a GAN objective. Indeed, they demonstrated that the latent space of their AAE could fit the prior better than vanilla VAE, see figure 4.1. Further, [39] have shown

that an AAE is equivalent to a Wasserstein autoencoder.

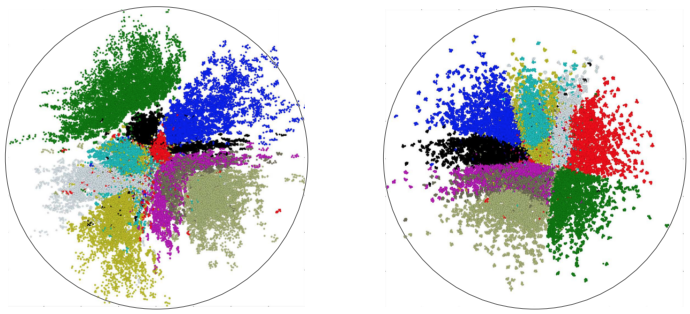


Figure 4.1: Comparison between the latent representation of a VAE and a AAE trained on MNIST, sourced from [38]. We see that in contrast to the VAE, the latent representation of the AAE does fit the Gaussian prior very well, has no "holes" and has sharp and (almost) non-overlapping transitions between different clusters. The gray circle indicates (a curve of constant probability of) the Gaussian prior.

Furthermore, the latent space can have much sharper transitions between different clusters with almost no overlap resulting in a more efficient encoding. This is because in contrast to the KL-divergence term of the VAE, the GAN objective in the AAE does not necessarily punish very peaked posteriors (very small variance in case of Gaussian posteriors). However, it is debatable whether this is desirable for our purpose, since we want to enforce that neighboring points in the latent space look similar. We think it is worthwhile to conduct experiments to replace the KL-divergence term by a GAN objective. In that case, we would have two GANs that regularize our reconstruction loss. The first is responsible for a nice ”shape” of the latent space and the second one for sharper images (see 4.2). To the best of our knowledge, this kind of generative autoencoder has not been proposed yet.

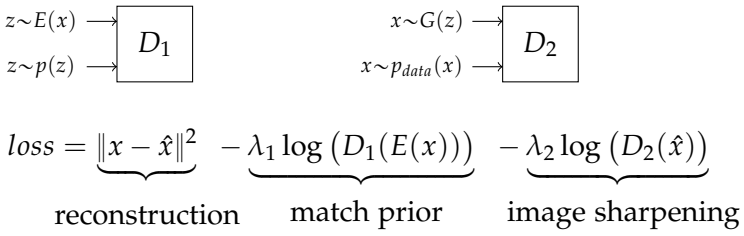


Figure 4.2: Replacing the KL-divergence term in the recurrent VAE-GAN by a GAN objective. Eventually, this leads to two different discriminators, one matches the posterior to the prior in the latent space and the other matches the "fake" data distribution to the real data distribution.

4.3.1 Learning the distribution of durations for different types of spectrogram segments

For the purpose of future work, let's assume we have a recurrent autoencoder with a continuous latent space that encodes non-silent spectrogram segments of variable length. Furthermore, let's assume we learned an HMM on encoded sequences. Then, when sampling a latent sequence from the HMM $(m_1, m_2, ...)$ we run into the problem that we don't know how long the corresponding spectrogram segment should be, that is, we do not know how many consecutive z-values we must produce in the dynamical system of the decoder. This is compounded by the second problem that, between any two non-silent segments, there is a silent segment of some arbitrary length.
Our solution is to train a regression model from the encoded latent points m to the length of the corresponding non-silent spectrogram segment and the subsequent length of the silent segment. Recall that the segmentation into silent and non-silent segments explicitly collected for every non-silent sequence x^T also the length of the following silent segment $\hat{T}$. Therefore, we can easily train a regression model on the data set $\{(m_i, T_i, \hat{T}_i)\}_i$, where $m_i = \varphi_e(x_i^{T_i})$ are the features and $(T_i, \hat{T}_i)$ the corresponding targets.

4.3.2 Conclusion

It was a critical step to identify the importance of matching the HMM to the autoencoder at the interface, namely the latent space. The contribution of the recurrent VAE-GAN is twofold. First, the encoding of spectrogram segments of arbitrary length gives us more design freedom in the cropping of the birdsong spectrograms. Second, we expect that the well suited latent space will significantly support the HMM with Gaussian emissions to learn and hence, will improve the quality of the generated birdsong spectrogram. We want also to highlight the strength of an adversarial regularization which was experimentally demonstrated in this work.
The investigation of the GS-VAE was very successfull and we can use its discrete representations of the spectrogram segments to learn HMMs with discrete emissions. However, the clusters obtained by assigning a spectrogram segment to its maximum likelihood latent state might be more relevant for applications on clustering.

Bibliography

[1] Sveinn Pálsson."Birdsong Generation with Generative Adversarial Networks", Dec. 2017.

[2] Sandro Giacomuzzi."Analysis of Dynamics in Birdsongs with Hidden Markov Models", Jul. 2018.

[3] C. K. Catchpole, P. J. B. Slater "Bird Song: Biological Themes and Variations", Cambridge [England]: Cambridge University Press.

[4] R. Hahnloser et al. "An ultra-sparse code underlies the generation of neural sequences in a songbird", Nature 2002.

[5] Lipkind, D., Marcus, G., Bemis, D. K., Sasahara, K., Jacoby, N., Takahasi, M., et al. (2013). "Stepwise acquisition of vocal combinatorial capacity in songbirds and human infants", Nature 498, 104–108. doi: 10.1038/nature12173.

[6] Doupe, A. J., & Kuhl, P. K. (2008), "Birdsongs and human speech: Common Themes and Mechanisms", Neuroscience of birdsong (pp. 5-31).

[7] Ila R. Fiete; Michale S. Fee; and H. Sebastian Seung, "Model of Birdsong Learning Based on Gradient Estimation by Dynamic Perturbation of Neural Conductances", J Neurophysiol 98: 2038–2057, 2007.

[8] Andrea P. Weber; Richard H. R. Hahnloser, "Spike Correlations in a Songbird Agree with a Simple Markov Population Model", PLoS Comput Biol 3(12): e249. doi:10.1371/journal, 2007.

[9] Robert C. Berwick; Kazuo Okanoya; Gabriel J.L. Beckers; Johan J. Bolhuis, "Songs to syntax: the linguistics of birdsong", Trends Cogn Sci. 2011 Mar; 15(3):113-21. doi: 10.1016/j.tics.2011.01.002.

[10] K. Doya; T.J. Sejnowski, "A Novel Reinforcement Model of Birdsong Vocalization Learning", MIT Press, Cambridge, MA (1995).

[11] Blaettler Florian; Kollmorgen Sepp; Herbst Joshua; Hahnloser Richard, "Hidden Markov Models in the Neurosciences", April 2011, DOI: 10.5772/14183.

[12] David Rothenberg; Tina C. Roeske; Henning U. Voss; Marc Naguib; and Ofer Tchernichovski "Investigation of musicality in birdsong", Article in Hearing research, September 2013.

[13] Sercan Ö. Arik et al., "Deep Voice: Real-time Neural Text-to-Speech", arXiv:1702.07825v2 [cs.CL] 7 Mar 2017.

[14] Louis Ranjarda; Howard A. Rossb , "Unsupervised bird song syllable classification using evolving neural networks", The Journal of the Acoustical Society of America · July 2008.

[15] Ng, A. Y.; Russell, S. J., "Algorithms for inverse reinforcement learning", In Icml (pp. 663-670), June 2000.

[16] Shintaro Shiba, Generating birdsong with WaveNet, GitHub, 2017.

[17] Tim Sainburg, Animal Vocalization Generative Network, GitHub, November 2018.

[18] Nathanaël Perraudin; Peter Balazs; Peter L. Søndergaard, "A FAST GRIFFIN-LIM ALGORITHM", 2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics.

[19] Rabiner, Lawrence, "First Hand: The Hidden Markov Model", IEEE Global History Network. Retrieved 2 October 2013.

[20] I. J. Goodfellow et al. "Generative Adversarial Networks", In: ArXiv e-prints (June 2014). arXiv: 1406.2661 [stat.ML].

[21] T. Che et al. "Mode Regularized Generative Adversarial Networks", In: ArXiv e-prints (Dec. 2016). arXiv: 1612.02136 [cs.LG].

[22] Christopher M. Bishop. "Pattern Recognition and Machine Learning", Publisher: Springer (2006).

[23] Diederik P. Kingma, Max Welling. "Auto-Encoding Variational Bayes", arXiv:1312.6114v10 [stat.ML] 1 May 2014.

[24] Kenneth R. Garren. "Bounds for the eigenvalues of a matrix", National Aeronautics and Space Administration 1968.

[25] Chris J. Maddison1, Andriy Mnih, Yee Whye Teh. "The concrete distribution: A continuous relaxation of discrete random variables", arXiv:1611.00712v3 [cs.LG] 5 Mar 2017.

[26] Eric Jang, Shixiang Gu, Ben Poole. "Categorical reparameterization with Gumbel-softmax", arXiv:1611.01144v5 [stat.ML] 5 Aug 2017.

[27] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", arXiv:1412.3555v1 [cs.NE] 11 Dec 2014.

[28] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud. "Neural Ordinary Differential Equations", arXiv:1806.07366v4 [cs.LG] 15 Jan 2019.

[29] J. E. Cavanaugh. "Unifying the derivations of the Akaike and corrected Akaike information criteria", Statistics and Probability Letters, 31: 201–208, doi:10.1016/s0167-7152(96)00128-9, 1997.

[30] Tatsuo S. Okubo, Emily L. Mackevicius, Hannah L. Payne, Galen F. Lynch, Michale S. Fee. "Growth and splitting of neural sequences in songbird vocal development", Nature volume 528, pages 352–357 (17 December 2015).

[31] Schwarz, Gideon E. (1978), "Estimating the dimension of a model", Annals of Statistics, 6 (2): 461–464.

[32] Aho, K.; Derryberry, D.; Peterson, T. (2014), "Model selection for ecologists: the worldviews of AIC and BIC", Ecology, 95: 631–636, doi:10.1890/13-1452.1.

[33] Oord, Aaron van den; Dieleman, Sander; Zen, Heiga; Simonyan, Karen; Vinyals, Oriol; Graves, Alex; Kalchbrenner, Nal; Senior, Andrew; Kavukcuoglu, Koray (2016-09-12), "WaveNet: A Generative Model for Raw Audio", arXiv:1609.03499v2 [cs.SD] 19 Sep 2016.

[34] Sepp Hochreiter; Jürgen Schmidhuber, "Long Short-term Memory", Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[35] Sandro Giacomuzzi, , GitHub, April 2019.

[36] Diederik P. Kingma; Max Welling, "Auto-Encoding Variational Bayes", arXiv:1312.6114v10 [stat.ML] 1 May 2014.

[37] Larsen A. B. L.; Sønderby S. K.; Larochelle H.; Winther O., "Autoencoding beyond pixels using a learned similarity metric", Proceedings of

the 33rd International Conference on Machine Learning, New York, NY, USA, 2016.

[38] Alireza Makhzani; Jonathon Shlens; Navdeep Jaitly; Ian Goodfellow; Brendan Frey, "Adversarial Autoencoders", arXiv:1511.05644v2 [cs.LG] 25 May 2016.

[39] Ilya Tolstikhin; Sylvain Gelly; Olivier Bousquet; Bernhard Schölkopf, "Wasserstein auto-encoders", Published as a conference paper at ICLR 2018.

[40] Anders Boesen Lindbo Larsen; Søren Kaae Sønderby; Hugo Larochelle; Ole Winther, "Autoencoding beyond pixels using a learned similarity metric", arXiv:1512.09300v2 [cs.LG] 10 Feb 2016.

[41] L. Baum, T. Petrie, G. Soules, and N. Weiss,, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains", The Annals of Mathematical Statistics, pp. 164–171, 1970.

Appendices

A Hidden Markov Model

A Hidden Markov Model (HMM) is a Bayesian network where the likelihood of observing a sequence $(z_1, ..., z_T)$ is a function of a sequence of hidden states $(h_1, ..., h_T)$ that have the structure of a Markov Chain.
Let H_t be a random variable representing the hidden state at the discrete time t. A possible realization $H_t = h_t$ can take values from a discrete set $\mathcal{H}$. Under the assumption that the process is stationary and has the Markov property $p(h_t|h_{t-1}, h_{t-2}, ...) = p(h_t|h_{t-1})$ the joint probability distribution over the hidden states factorizes as

$$p(h_1, ..., h_T) = p(h_1) \prod_{t=2}^{T} p(h_t|h_{t-1}).$$

The initial distribution $p(h_1)$ and the conditional distribution $p(h_t|h_{t-1})$ need to be parameterized and inferred from the data. Note that all distributions are time independent (stationarity). It is then convenient to interpret the hidden states h_t as probability column vectors of dimension $|\mathcal{H}|$, that is, all elements are positive and sum up to one. If the hidden state is in the j-th state this corresponds to a one-hot encoding, that is, the j-th element $h_{t,j} = 1$ is equal to one and all others are zero, $h_{t,i} = 0, \forall i \neq j$. The parameters of the conditional distribution $p(h_t|h_{t-1})$ can be collected in a $|\mathcal{H}|$ by $|\mathcal{H}|$ transition matrix A and we can use the usual matrix multiplication convention to write

$$p(h_t|h_{t-1}, A) = h_t^\top A h_{t-1}, \;\; 0 \leqslant A_{ij} \leqslant 1, \;\; \sum_i A_{ij} = 1.$$

The initial distribution $p(h_1)$ is parameterized by a probability vector π such that we get $p(h_1) = p(h_1|\pi) = h_1^\top \pi$. The likelihood of a hidden state sequence can then be written in the form

$$p(h_1, ..., h_T) = p(h_1) \prod_{t=2}^{T} p(h_t|h_{t-1}) = h_1^\top \pi \prod_{t=2}^{T} h_t^\top A h_{t-1}.$$

The observables, are again random variables Z_t with realizations $Z_t = z_t$ that take values from a discrete or continuous set $\mathcal{Z}$. In a HMM of first order an observable Z_t at time t depends only on the hidden state H_t at time t. Therefore, the likelihood of an observed sequence given a sequence of hidden states factorizes as

$$p(z_1, ..., z_T | h_1, ..., h_T) = \prod_{t=1}^{T} p(z_t | h_t).$$

Again, the conditional distribution $p(z_t|h_t)$ needs to be parameterized and inferred from the data. In order to find parameters of the HMM that have a high likelihood on training data we use the Baum–Welch optimization algorithm.

B Gumbel normal distribution

Probability density function (PDF):

$$p(x) = \exp(-\exp(-x))\exp(-x)$$

Cumulative distribution function (CDF):

$$F(x) = \int_{-\infty}^{x} p(x')dx' = \exp(-\exp(-x))$$

Inverse Cumulative distribution function (CDF^{-1}):

$$F^{-1}(y) = -\log(-\log(y))$$

Corollary, we can sample from the Gumbel distribution by first sampling from the normal uniform distribution $u \sim Uni(0,1)$ and then apply the CDF^{-1}, i.e.

$$g = F^{-1}(u),\ u \sim Uni(0,1) \quad \Rightarrow \quad g \sim Gumbel(0,1)$$

Argmax-property:

Theorem 2. *Let $q \in \Delta^{K-1}$ be a PMF with $q(i) = \pi_i$ and $\{g_i\}_{i=1}^{K}$ be K i.i.d drawings from the Gumbel normal distribution, then it holds:*

$$\pi_k = \Pr[k = \underset{i}{\operatorname{argmax}}\{g_i + \log(\pi_i)\}] \equiv \Pr[g_k + \log(\pi_k) > g_i + \log(\pi_i),\ \forall i \neq k]$$

Proof.

$$\begin{aligned}\Pr[g_k + \log(\pi_k) > g_i + \log(\pi_i),\ \forall i \neq k] &= \Pr[g_i < g_k + \log(\pi_k) - \log(\pi_i),\ \forall i \neq k] \\ &= \int p(g) \prod_{i \neq k} \Pr[g_i < g + \log(\pi_k) - \log(\pi_i)] dg \\ &= \int p(g) \prod_{i \neq k} F\big(g + \log(\pi_k) - \log(\pi_i)\big) dg \\ &= ... \\ &= \pi_k,\end{aligned}$$

where in the last step we simply need to insert the Gumbel PDF and CDF and solve the standard integral. □

A useful fact is that the property even holds for unnormalized PMF's, i.e. let's define the unnormalized probability distribution $\alpha \in \mathbb{R}^+ \cdot \Delta^{K-1}$, that is, $\alpha(i) \equiv \alpha_i = \lambda \pi_i$ for some $\lambda \in \mathbb{R}^+$. It still holds:

$$\Pr[k = \underset{i}{\operatorname{argmax}}\{g_i + \log(\alpha_i)\}] = \frac{\alpha_k}{\sum_i \alpha_i} = \pi_k$$

This becomes immediately clear from the proof of theorem 2, which depends on the term

$$\log(\pi_k) - \log(\pi_i) = \log\left(\frac{\pi_k}{\pi_i}\right) = \log\left(\frac{\lambda \pi_k}{\lambda \pi_i}\right) = \log(\alpha_k) - \log(\alpha_i)$$

C Numerical issues with log-Concrete

In practice, we need a work around in order to evaluate the log-Concrete distribution numerically stable. Recall the Concrete PDF:

$$p_\tau(y|\alpha) = c(\tau) \left(\sum_{i=1}^K \frac{\alpha_i}{y_i^\tau}\right)^{-K} \prod_{j=1}^K \left(\frac{\alpha_j}{y_j^{\tau+1}}\right).$$

One way to achieve that is [25] to consider a variable change to the log-domain. Let $Y \sim p_\tau(\cdot|\alpha)$. As long as $y_i > 0\ \forall i, \tau > 0$ the transformation $Y \to \widetilde{Y} = \log(Y)$ (where the function $\log(\cdot)$ is applied element wise) is invertible. We can write the transformation in terms of the gumbel samples g_i as

$$\widetilde{Y}_i = \frac{\xi_i}{\tau} - \log\left(\sum_{j=1}^K \exp\left(\frac{\xi_j}{\tau}\right)\right).$$

It can be shown, that the corresponding PDF is

$$\kappa_\tau(\widetilde{y}|\alpha) = c(\tau)\left(\sum_{i=1}^{K}\frac{\alpha_i}{\exp(\widetilde{y}_i\cdot\tau)}\right)^{-K}\prod_{j=1}^{K}\frac{\alpha_j}{\exp(\widetilde{y}_j^\tau)},$$

which is called the expconcrete distribution. The corresponding log-likelihood is

$$\log\left(\kappa_\tau(\widetilde{y}|\alpha)\right) = \log\left(c(\tau)\right) - K\cdot\log\left(\sum_{j=1}^{K}\exp\left(log(\alpha_i)-\tau\widetilde{y}_i\right)\right) + \sum_{j=1}^{K}\log(\alpha_i) - \tau\widetilde{y}_j.$$

Finally, we might keep in mind the property of "entropy-like" terms which are invariant under invertible transformations, such that we have for example:

$$D_{KL}\left(p_\tau(y|\alpha)||p_{\tau_p}(y|\alpha_p)\right) \equiv D_{KL}\left(\kappa_\tau(\widetilde{y}|\alpha)||\kappa_{\tau_p}(\widetilde{y}|\alpha_p)\right),$$

for some arbitrary prior distribution $p_{\tau_p}(y|\alpha_p)$. Therefore, for most of the loss functions used in practice we can use the expconcrete distribution instead of the concrete distribution without a lot of further ado.

D Reparameterization Trick

If we need to calculate gradients through an expectation it is often the case that the corresponding integral is not solvable. The straight forward Monte-Carlo estimation, which is based on the identity

$$\nabla_\varphi \mathbb{E}_{q_\varphi(z)}[f_\varphi(z)] \equiv \mathbb{E}_{q_\varphi(z)}\left[\nabla_\varphi f_\varphi(z) + f_\varphi(z)\nabla_\varphi \log\left(q_\varphi(z)\right)\right]$$

has been shown to suffer from high variance. A more stable method is [23], if possible, to find a reparameterization of the distribution $q_\varphi(z)$ in terms of a noise parameter $\varepsilon \sim p(\varepsilon)$ such that its transformation of a suitable (deterministic) transformation $g_\varphi(\varepsilon)$ is distributed according to $q_\varphi(z)$, i.e. $g_\varphi(\varepsilon) \sim q_\varphi(z)$. The reparameterized expectation is then

$$\mathbb{E}_{q_\varphi(z)}[f_\varphi(z)] = \mathbb{E}_{p(\varepsilon)}[f_\varphi(g_\varphi(\varepsilon))],$$

on which we can directly apply the following Monte-Carlo approximation

$$\mathbb{E}_{q_\varphi(z)}[f_\varphi(z)] \approx \frac{1}{L}\sum_{l=1}^{L} f_\varphi(g_\varphi(\varepsilon^{(l)})),$$

where $\{\varepsilon^{(l)}\}_{l=1}^{L}$ i.i.d. $\sim p(\varepsilon)$.
The overall optimization procedure can be done in minibatches in order to reduce the variance of the gradient. The larger the number of minibatches the more we can reduce the amount of Monte-Carlo samples L. In practice, the minibatches are usually large enough in order to set $L = 1$.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.