CAMBRIDGE UNIVERSITY

MLSALT4: ADVANCED MACHINE LEARNING

Paper Replication: *Sequential Neural Models with Stochastic Layers*

Áine Cahill,
Paula Wesselmann,
Yeziwei Wang,
Gerry Che

April 4, 2018

Abstract

Uncertainty in latent space is a natural tool to model high level abstractions and variations in natural images, speech and natural languages. Although it is well-known that autoregressive models are able to capture randomness in the data, the uncertainty is modelled in the low level, i.e. softmax sampling. In this work, recurrent neural network based models that are able to capture high level uncertainty are explored. In particular, Stochastic Recurrent Neural Network (SRNN) architecture is implemented and discussed, which separates clearly the deterministic and stochastic layer, as proposed by the paper *Sequential Neural Models with Stochastic Layers*. It is compared with other recently proposed architectures as well as variations of the architecture.

Contents

1 Introduction

Recurrent neural networks (RNNs) are able to model long-term dependencies and randomness in sequential data by adapting a deterministic hidden state and Softmax-level random sampling. It has been recently proved that when complex sequences, such as speech and music, are modelled, the performances of RNNs can be dramatically improved when uncertainty is introduced in the hidden states. One common belief in the deep learning community is that to deal with "AI-data", namely high dimensional data with complex meaning such as natural images, speech or natural language, one has to be able to model the variations in the data at a high level [3]. Variations in the data at high level can be abstractions, such as semantics or natural objects, compared to low level choices such as characters or pixels.

This report studies a new way of embedding randomness into latent space of RNNs as proposed by [8] in the paper *Sequential Neural Models with Stochastic Layers*, and compares with various different previous and recent works. SRNN model[8] is replicated and modified for music data, using the code [1]. Pros and cons of these models are discussed and potential future directions are pointed out to explore along this line of research.

2 Project Motivation

There are many current architectures that model sequential data, such as Recurrent Neural Networks, State Space Models and Kalman Filter, especially for speech modelling. Most recent works have been proposing architectures integrating stochastic models with deterministic models to represent dependencies as well as uncertainty in the sequential data. This paper is among these work proposing a novel architecture, inserting stochastic variables into recurrent neural networks, to model speech. The model is also tested on music data. However, the investigation on music data is briefly touched upon. Due to the different nature of music and speech, especially midi music, it is thought to be important to further explore the architecture on music prediction.

3 Theory

In this section, theories of State Space Models (SSMs), Recurrent Neural Networks (RNNs) and several other different stochastic RNN models are reviewed, such as STORN and VRNN.

3.1 SSM

State Space Models (SSMs) are used to model sequential data of variable length. A SSM (Figure 1) is a generative model, comprising a transition model g, an observation model h, stochastic, continuous latent states $\mathbf{z}_t$, observations $\mathbf{x}_t$ and control inputs $\mathbf{u}_t$,

$$\mathbf{z}_t = g(\mathbf{u}_t, \mathbf{z}_{t-1}, \epsilon_t)$$
$$\mathbf{x}_t = h(\mathbf{z}_t, \mathbf{u}_t, \delta_t)$$

Where ϵ_t and δ_t correspond to the system noise and the observation noise at time t respectively [12]. The latent variables $\mathbf{z}_t$ are sufficient statistics for the observations $\mathbf{y}_t$. $\mathbf{z}_t$ evolve dynamically with time, and it is sought to model the uncertainty in $\mathbf{z}_t$. The transition dynamics can be either linear or nonlinear. Nonlinear SSMs can be parameterised by a neural network, as is performed in the SRNN in this report.

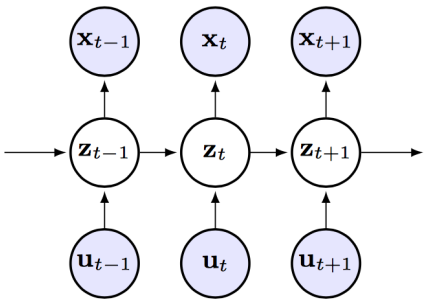


Figure 1: SSM

Modelling uncertainty in the latent states $\mathbf{z}_t$ requires computation of $p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)$ and $p(\mathbf{x}_t|\mathbf{z}_t)$. However, marginalization of $\mathbf{z}_t$ is only tractable when the probability distributions belong to the exponential family. For

[1] github.com/marcofraccaro/srnn

this reason, only simple SSMs are used in practice, such as Hidden Markov Models (HMMs) and Linear Gaussian Models (also called Kalman Filters), often with applications in tracking, control and planning.

The main advantage of SSMs is their ability to model the uncertainty in the latent states. However, their use is limited by the difficulty of obtaining tractable inference models and the difficulty of scaling to higher dimensional models.

3.2 RNN

Recurrent neural networks (RNNs) are networks with recurrent hidden layers that allow to represent long-term dependencies. They can be thought of as a chain of normal neural networks each passing a message to the next time instance. The chain-like nature is why they work well to model and generate sequential data such as music and speech. For example RNNs (Figure 2) model temporal sequences of vectors $x_{1:T} = (x_1, x_2, \ldots, x_T)$ that depend on inputs $u_{1:T} = (u_1, u_2, \ldots, u_T)$ with the assumption that the sequence of observations up to time t can be summarized by a latent state d_t that is deterministically calculated. RNNs recursively compute $d_t = f(d_{t-1}, u_t)$ using a parameterized nonlinear function f, like a LSTM cell or a GRU. The observation probabilities $p(x_t, d_t)$ are also modelled using nonlinear functions.

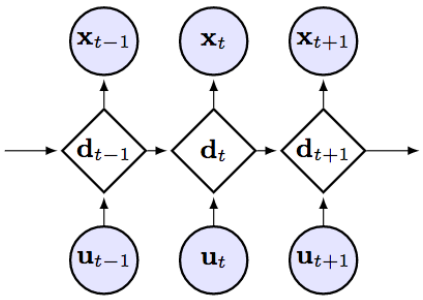


Figure 2: RNN

3.3 SRNN

The Stochastic Recurrent Neural Network (SRNN) proposed in the paper stacks a stochastic layer from SMM onto the deterministic layer in RNN. The particular architecture is constructed into two models: a generative model and a inference model.

3.3.1 Generative Model

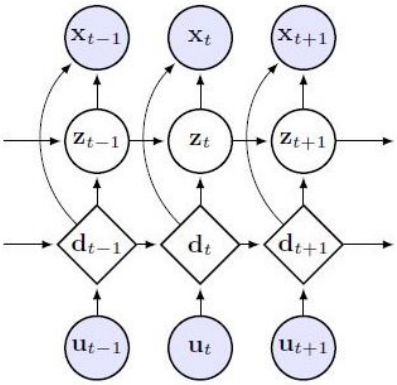


Figure 3: Generative model p_θ

The generative model (Figure 3) gives a probability distribution over a sequence, which factorises as

$$\begin{aligned} p_\theta(x_{1:T}, z_{1:T}, d_{1:T} | u_{1:T}, z_0, d_0) &= p_{\theta_x}(x_{1:T}|z_{1:T}, d_{1:T}) p_{\theta_z}(z_{1:T}|d_{1:T}, z_0) p_{\theta_d}(d_{1:T}|u_{1:T}, d_0) \\ &= \prod_{t=1}^{T} p_{\theta_x}(x_t|z_t, d_t) p_{\theta_z}(z_t|z_{t-1}, d_t) p_{\theta_d}(d_t|d_{t-1}, u_t) \end{aligned}$$

The deterministic hidden layer is d, which is determined by the deterministic non-linear function

$$d_t = f_{\theta_d}(d_{t-1}, u_t)$$

The non-linear function is chosen as GRU nodes with parameters θ_d in this paper. The prior probability of hidden layer d is therefore a delta distribution

$$p_{\theta_d}(d_t|d_{t-1}, u_t) = \delta(d_t - \tilde{d}_t)$$

where $\tilde{d}_t$ is the value computed by the non-linear function. The probability of stochastic hidden layer z is determined by

$$p_{\theta_z}(z_t|z_{t-1}, d_t) = N(z_t; \mu_t^{(P)}, v_t^{(p)})$$

a Gaussian distribution with diagonal covariance matrix, where

$$\mu_t^{(p)} = NN_1^{(p)}(z_{t-1}, d_t), \quad logv_t^{(p)} = NN_2^{(p)}(z_{t-1}, d_t)$$

The paper implements the parameterization with a two layer feed-forward neural network. Similarly, the output distribution $p_{\theta_x}(x_t|z_t, d_t)$ has parameters that are generated by a similar neural network parameterized by θ_x.

3.3.2 Inference Network

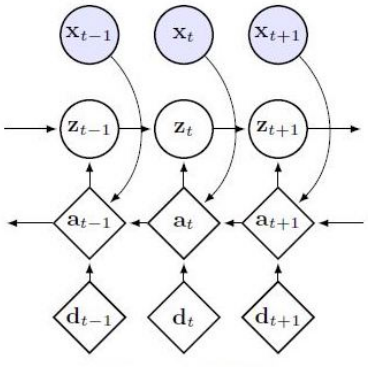


Figure 4: Inference network q_ϕ

An inference network (Figure 4) is used to represent the probability distribution of hidden states, which is factorised as:

$$q_\phi(z_{1:T}, d_{1:T}|x_{1:T}, u_{1:T}, z_0, d_0) = q_\phi(z_{1:T}|d_{1:T}, x_{1:T}, z_0)q(d_{1:T}|x_{1:T}, u_{1:T}, d_0)$$

where $q(d_{1:T}|x_{1:T}, u_{1:T}, d_0) = p_{\theta_d}(d_{1:T}|u_{1:T}, d_0)$ is a delta function, which is the prior of $d_{1:T}$.
The true posterior distribution of stochastic states $z_{1:T}$ is

$$p_\theta(z_{1:T}|d_{1:T}, x_{1:T}, u_{1:T}, z_0) = \prod_t p_\theta(z_t|z_{t-1}, d_{t:T}, x_{t:T})$$

which is approximated by variational approximation

$$\begin{aligned} q_\phi(z_{1:T}|d_{1:T}, x_{1:T}, z_0) &= \prod_t q_\phi(z_t|z_{t-1}, d_{t:T}, x_{t:T}) \\ &= \prod_t q_\phi(z_t|z_{t-1}, a_t = g_{\phi_a}(a_{t+1}, [d_t, x_t])) \end{aligned}$$

Let $q_\phi(z_t|z_{t-1}, a_t)$ be a Gaussian distribution with diagonal covariance matrix, then the parameters can be represented from neural networks like in the generative model

$$\mu_t^{(q)} = NN_1^{(q)}(z_{t-1}, a_t), \quad logv_t^{(q)} = NN_2^{(q)}(z_{t-1}, a_t)$$

This variational distribution approximates the dependence of the true posterior. The paper proposes an improvement in parameterization to offer better performance. Instead of learning the distribution of priori, neural networks are used to learn the residual between $\mu_t^{(p)}$ and $\mu_t^{(q)}$

$$\hat{\boldsymbol{\mu}}_t^{(p)} = \int NN_1^{(p)}(\mathbf{z}_{t-1}, \mathbf{d}_t)p(\mathbf{z}_{t-1}|\mathbf{x}_{1:T})d\mathbf{z}_{t-1} \approx \int NN_1^{(p)}(\mathbf{z}_{t-1}, \mathbf{d}_t)q_\phi^*(\mathbf{z}_{t-1})d\mathbf{z}_{t-1} \tag{1}$$

$$\boldsymbol{\mu}_t^{(q)} = \hat{\boldsymbol{\mu}}_t^{(p)} + NN_1^{(q)}(\mathbf{z}_{t-1}, \mathbf{a}_t) \tag{2}$$

This makes it easier for the inference network to track the changes in the generative model and also avoids the inference network from being stuck by trying to optimize the ELBO through sampling from the prior of the model.

3.3.3 Model Training

Observed data sequences are described by marginal log-likelihood, i.e evidence:

$$\begin{aligned}\mathcal{L}(\theta) &= \log p_\theta(\mathbf{x}^i_{1:T}|\mathbf{u}^i_{1:T}, \mathbf{z}^i_0, \mathbf{d}^{i\,N}_{0\,i=1}) \\ &= \sum_i \log p_\theta(\mathbf{x}^i_{1:T}|\mathbf{u}^i_{1:T}, \mathbf{z}^i_0, \mathbf{d}^i_0) \\ &= \sum_i \mathcal{L}_i(\theta)\end{aligned}$$

As maximizing the log-likelihood is intractable, the paper proposed to maximize the variational evidence lower bound (ELBO), described by Equation 3, of the log-likelihood using stochastic gradient ascent. The maximization is performed over ϕ and θ iteratively, similar to the EM algorithm term.

$$\mathcal{F}(\theta, \phi) = \sum_i \mathcal{F}_i(\theta, \phi) \leq \mathcal{L}(\theta) \tag{3}$$

ELBO is computed as the difference between the expectation of log-likelihood of the observation sequence and the Kullback–Leibler (KL) divergence between true posterior and its approximation. As both generative model and inference network factorize over time steps, the ELBO is written as a sum over time steps.

$$\begin{aligned}\mathcal{F}_i(\theta, \phi) &= \mathbb{E}_{q_\phi}\left[\log p_\theta(x_{1:T}|z_{1:T}, \tilde{d}_{1:T})\right] - KL\Big(q_\phi(z_{1:T}|\tilde{d}_{1:T}, x_{1:T}, z_0)||p_\theta(z_{1:T}|\tilde{d}_{1:T}, z_0)\Big) \\ &= \sum_t \mathbb{E}_{q^*_\phi(z_{t-2})}\Big[\mathbb{E}_{q_\phi(z_t|z_{t-1}, \tilde{d}_{t:T}, x_{t:T})}\left[\log p_\theta(x_t|z_t, \tilde{d}_t)\right] - KL\Big(q_\phi(z_t|z_{t-1}, \tilde{d}_{t:T}, x_{t:T}, z_0)||p_\theta(z_t|z_{t-1}, \tilde{d}_t)\Big)\Big]\end{aligned}$$

where $q^*_\phi(z_{t-1})$ is the marginal distribution of z_{t-1} in the variational approximation to the posterior

$$q_\phi(z_{1:t-1}|\tilde{d}_{1:T}, x_{1:T}, z_0)$$

3.4 Latent Variable RNNs

Latent variables can be employed in RNNs to model uncertainty between the hidden states of an RNN. A large number of different configurations of latent variable RNN models have been shown in the literature to yield performance improvements on regular RNNs for sequential data modelling, particularly for speech and music data.

3.4.1 RNN-RBM

The RNN-RBM is proposed in [5]. This model (Figure 5) comprises a sequence of conditional RBMS, whose parameters are the outputs of a full RNN.

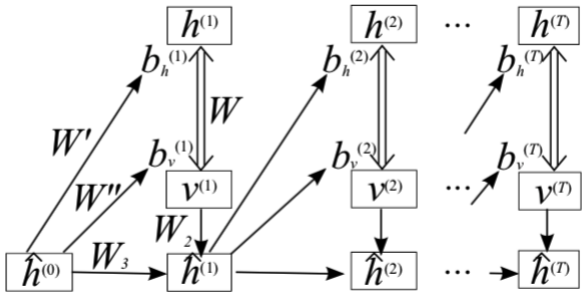


Figure 5: Single layer RNN-RBM; single arrows represent a deterministic function, double arrows represent the stochastic hidden-visible connections of an RBM. The upper half of the RNN-RBM is the RBM stage while the lower half is a RNN with hidden units $\hat{h}^{(t)}$. The RBM biases $b_h^{(t)}$, $b_v^{(t)}$ are a linear function of $\hat{h}^{(t-1)}$.

3.4.2 STORN

The STORN model [1] is the first RNN model using stochastic latent variables. It uses a series of independent distributed latent variables $\mathbf{z}_{1:T}$ with unit Gaussian prior, and model the sequence distribution as:

$$p(\mathbf{x}_{1:T}) = \int_{\mathbf{z}} p(\mathbf{z}_{1:T}) \prod_{t=0}^{T} p(x_{t+1}|h_t(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}))$$

The inference network can be represented as $q(\mathbf{z}_t|\mathbf{x}_{\leq t})$.

3.4.3 VRNN

The variational RNN [7] is the second earliest variational RNN model. The generative model of the architecture is roughly:

$$p(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}) = \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{x}_{<t}, \mathbf{z}_{\leq t}) p(\mathbf{z}_t|\mathbf{x}_{\leq t}, \mathbf{z}_{<t})$$

It uses an RNN to encode dependency, specifically $\mathbf{h}_t = \phi_r(\mathbf{h}_{t-1}, \mathbf{z}_t, \mathbf{x}_t)$. The conditional distribution in the above equation is represented as $p(\mathbf{x}_t|\mathbf{x}_{<t}, \mathbf{z}_{\leq t}) = p(\mathbf{x}_t|\mathbf{h}_t, \mathbf{z}_t)$. The inference model is represented as $q(\mathbf{z}_t|\mathbf{x}_{\leq t}, \mathbf{z}_{<t})$. The computational graph of VRNN is summarized in Figure 6.

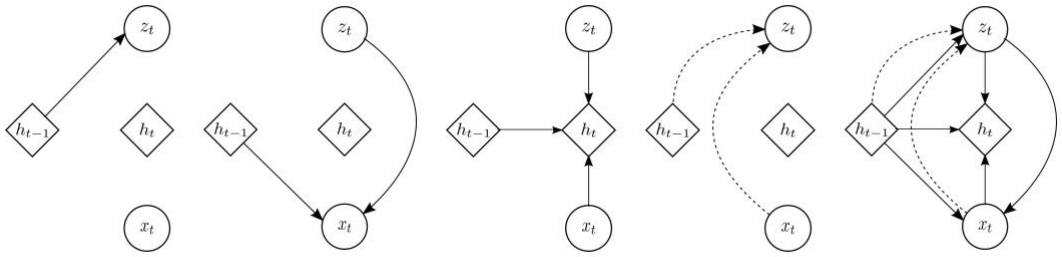


Figure 6: Graphical illustrations of each operation of the VRNN: (a) computing the conditional prior; (b) generating function; (c) updating the RNN hidden state; (d) inference of the approximate posterior; (e) overall computational paths of VRNN

3.5 KL-Vanishing Problem and Z-forcing

It has been observed in different cases that in Variational Auto-encoder (VAE) like models with expressive decoders, such as RNNs, KL vanishing problem is a major challenge[14, 6]. Namely the KL divergence regularizer between the prior and posterior collapse to zero, so the learning algorithm completely ignores the effect of the latent variables. The architecture would in fact collapse to a deterministic RNN. In current deep learning community, some authors argue that the this is because the decoder is so powerful that it can model the entire distribution itself ignoring the stochastic latent space.

Recently, [10] proposed a way of addressing the KL-vanishing problem in the setting of SRNNS. The algorithm simply adds two reconstruction losses to force the latent variables to be useful for the future contents. Difference architectures of latent variable RNNs are shown in Figure 7.

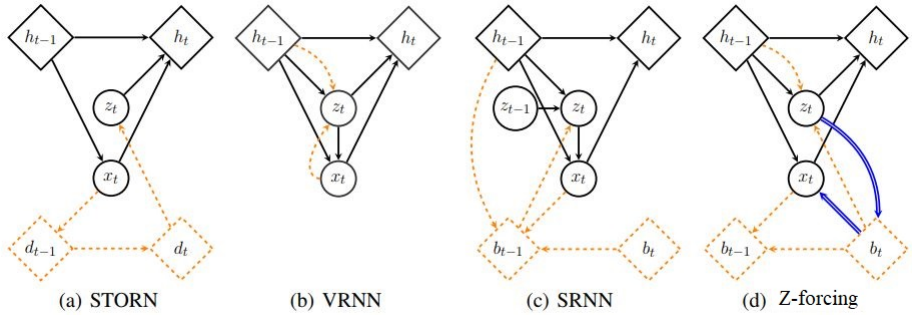


Figure 7: Computation graph for generative models of sequences that use latent variables: STORN[1], VRNN[7], SRNN[8] and Z-forcing[10]

Note that in the graph, Z-forcing forces z_t to predict the backward RNN state b_t. It also forces b_t to predict previous input x_t. These two additional losses make sure that z_t contains useful information about the future.

The SRNN in [8] addresses the KL-vanishing problem and improves posterior approximation by using posterior approximations over previous latent states,

$$q_\phi(\mathbf{z}_{1:t-1}|\tilde{\mathbf{d}}_{1:T}, x_{1:T}, \mathbf{z}_0) \approx q^*_\phi(\mathbf{z}_{t-1})$$

This ensures that the inference network g_ϕ does not simply imitate the behaviour of the prior using information from hidden state $\tilde{\mathbf{d}}_t$. As the variational distribution is constructed sequentially, the predictive prior mean on the prior dynamics of $\mathbf{z}_t$ can be approximated as Equation 1. This predictive prior mean then is used in the parameterization of the mean of the variational approximation $q_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_t)$ as in Equation 2.

3.6 Midi Music Data

MIDI (Musical Instrument Digital Interface) is a standard music technology protocol that enables music to be communicated digitally between electronic instruments and other digital music tools using MIDI messages. For example, MIDI messages are used to communicate music generated on an electronic keyboard and a synthesizer to an amplifier for amplification and to a computer for post-processing. MIDI messages specify notation, pitch, velocity (music dynamics,e.g. piano, forte), instruments, control signals for volume, audio panning, theatre cues, pedals and clock signals for synchronizing tempo between devices [13].

A 'note on' MIDI message, including *note* and *velocity* values, is generated when a key is pressed: $note \in [0, 127]$, describes the pitch of a pressed key. $velocity \in [0, 127]$ describes the volume of a MIDI note. When a note is released, a 'note off' MIDI message is generated. Additional MIDI messages include $aftertouch \in [0, 127]$, which describes key pressure changes, $pitchbend \in [0, 127]$, which denotes a shift in the pitch of the note played, up or down by less than a semitone [13].

MIDI has a very small file size compared to audio data, which has a very high sample rate. MIDI messages can be easily modified, e.g. a note can be changed, or dynamics adjusted without the need to re-record music. MIDI is compatible for use with many electronic instruments and electronic music devices and tools. Furthermore, MIDI music can be played on any music device without need for adjustment to the music data.

The music data is in MIDI format and includes four datasets; 'jsb.midi' (JSB), 344kB, 'piano.midi' (Piano), 894kB, 'nottingham.midi' (Nottingham), 1.6MB, and 'muse.midi' (Muse), 3.9MB. Each dataset is internally split into training, validation and test sets. *JSB Chorales* is very simple, comprising multiple notes playing simultaneoulsy at a consistent tempo. *Piano* music contains varied and complex piano music, with multiple voices, ornamentation and a widely changing tempo. *Muse* is choral music, including voice and a background music. *Nottingham* music is very simple piano music with mostly a single note playing at any one time.

4 Experiments and Results

The SRNN model was implemented in Python using the Lasagne, Parmesan and Theano packages.

4.1 Original Architecture

The default SRNN implementation uses multi-layer perceptrons (MLPs) hidden layers of following dimensions $|d| = 300$, $|z| = 100$, $|a| = 300$ and sequence length = 100, where d is the deterministic layer, z is the stochastic layer and a is the auxiliary states in the inference network. The distribution of the stochastic layer is parameterized with a 1-layer neural networks with 500 units per layer. The number of units per layer for the music SRNN model is much smaller than that for the speech SRNN model as speech tends to be more noisy and carries more uncertainty, hence, a network of larger dimensions is required. The output layer has 88 units representing 88 notes on a piano roll. In the original paper, the authors used the same model architecture (not the same latent variable dimensions) for modeling speech and MIDI music, except the output for the speech SRNN is modeled using a fully factorized Gaussian distribution, whereas the music SRNN architecture outputs Bernoulli variables to model the active notes. The default dropout probability during training for the latent variables $\mathbf{z}$ and $\mathbf{d}$ and for the inputs $\mathbf{u}$ and $\mathbf{x}$ is 0.0.

The 'adam' optimiser is used. Mini-batch gradient descent is implemented with batch sizes of 16. Nesterov's accelerated gradient descent is used, with a momentum constant of 0.9. Initially, the learning rate is $10^{-2.5} = 0.00316$. The learning rate is decayed exponentially with a decay scale factor of 1.3, starting after 25 training epochs. A KL temperature coefficient, β, is used to gradually introduce the KL term to the ELBO during training. β is varied from 0.2 to 1.0 over 20 epochs, such that a linearly increasing fraction of the KL term is introduced to the ELBO over 20 epochs. Gradients greater than ± 10.0 are clipped to prevent exploding gradients. To prevent vanishing gradients leaky and clipped ReLU activation functions are used. The activation functions clip values to ± 2.0 and are leaky; defined with a value of 0.9, where 1.0 is a linear activation function, 0.0 is a standard rectifier and any value in the range $(0.0, 1.0)$ yields a leaky rectifier. In the original SRNN, training is performed over 50 epochs and evaluated every 2nd epoch using the validation data. The batch size for validation and testing is 32.

4.2 Replication of Original Experiment

The results from the paper were replicated using the experimental set-up described in the previous section, and using the code provided by the authors on GitHub (see footnote 1). The replication experiment using polyphonic music and the original SRNN set-up from the paper yields comparable, but not the exact same results (see line 2 in Table 1) for the JSB Chorales, MuseData and Nottingham datasets and numerical improvements for the Piano dataset. In all instances the replication results for the ELBO values are slightly below the results from the original paper (line 1 of Table 1). This could be due to better tuned settings used in the original experiment, since only the main settings are shared in the paper.

	Models	Nottingham	JSB chorales	MuseData	Piano-midi.de
1	SRNN original	$\geq$ -2.94	$\geq$ -4.74	$\geq$ -6.28	$\geq$ - 8.20
2	SRNN replication	$\geq$ -3.18	$\geq$ -5.37	$\geq$ -7.43	$\geq$ -7.57
3	SRNN (small dim)	$\geq$ -3.31	$\geq$ -5.04	$\geq$ -7.56	$\geq$ -7.46
4	SRNN (merged data)	$\geq$ -3.27	$\geq$ -5.64	$\geq$ -7.64	$\geq$ -8.03
5	TSBN	$\geq$ -3.67	$\geq$ -7.48	$\geq$ -6.81	$\geq$ -6.81
6	NASMC	$\approx$ -2.72	$\approx$ **-3.99**	$\approx$ -6.89	$\approx$ -7.61
7	STORN	$\approx$ -2.85	$\approx$ -6.91	$\approx$ -6.16	$\approx$ -7.13
8	RNN-NADE	$\approx$ **-2.31**	$\approx$ -5.19	$\approx$ **-5.60**	$\approx$ **-7.05**
9	RNN	$\approx$ -4.46	$\approx$ -8.71	$\approx$ -8.13	$\approx$ -8.37

Table 1: Average log-likelihood on the test sets, including replication results and results from papers. TSBN results are from [9], NASMC results are from [11], STORN results are from [2] and RNN-NADE and RNN results are from [4].

The training curves over 50 epochs for the four datasets are plotted in figure 8. The plot shows that the learning curve of train, test and validation dataset agree well with each other for all four different datasets, indicating that the training is not overfiting. Also, based on observation convergence occurs after 20 epochs. Therefore, 20 training epochs are used for the following extension experiments.

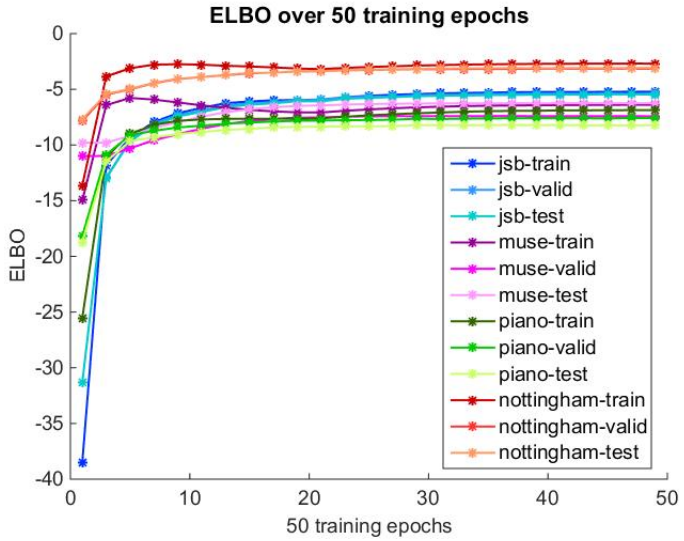


Figure 8: Training ELBO on Valid Dataset over 50 Epochs

4.3 Extension Experiments

4.3.1 Varying Dimensions

To further investigate the effect of different dimensions in the stochastic layer ($\mathbf{z}$) and deterministic layer ($\mathbf{d}$), models were trained with different dimensions. The units in the $\mathbf{z}$ layer vary from 20 to 180 in intervals of 20 units, and the $\mathbf{d}$ layer vary from 200 to 400 units in intervals of 50 units.

The results are plotted in Figure 9. In general, when the dimensionality of the $\mathbf{z}$ layer is increased, the ELBO decreases. This may be because larger dimensions introduces more randomness into the model, and too much randomness can make the ELBO bound less tight. Considering the dimension of the $\mathbf{d}$ layer, different

dimensions do not have a significant effect on the ELBO. These results indicate that an SRNN architecture of smaller dimensions may yield superior performance in music sequence prediction.

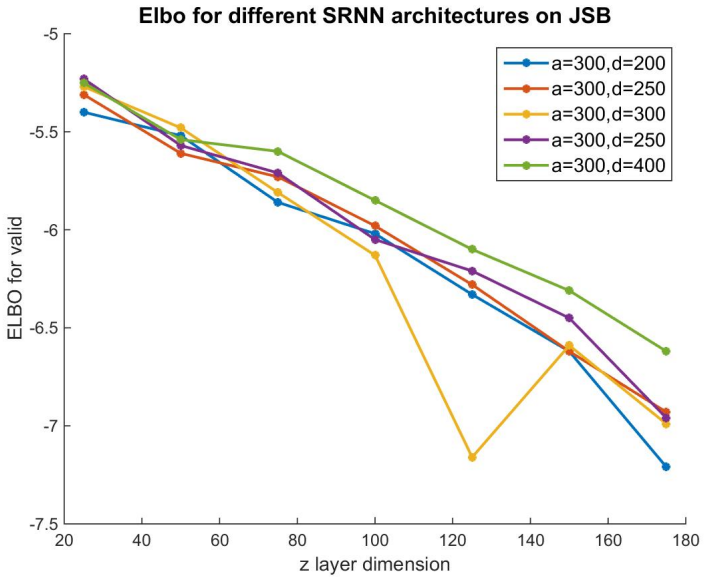


Figure 9: ELBO for varying dimensions of SRNN

4.3.2 Small Model Architectures

By experimenting with varying dimensions it was discovered that the ELBO improves with lower dimensions of the stochastic layer $\mathbf{z}$. Hence, experiments were conducted for all datasets using very small dimensions compared to the original settings of the replication task. Dimensions are set to $|a| = 30$, $|d| = 30$ and $|z| = 10$, each being a tenth of the original setting whilst the sequence length is kept at 100. Results for the small dimension experiment are shown in line 3 of Table 1. They are very similar to the results of the original SRNN replication experiment in line 2, with slight improvements for JSB and piano-midi.de and a slight decrease in ELBO values for Nottingham and Muse. A more complex model, therefore, is not needed nor beneficial for music modelling using the SRNN architecture and the small dimensions seem to be sufficient for this task.

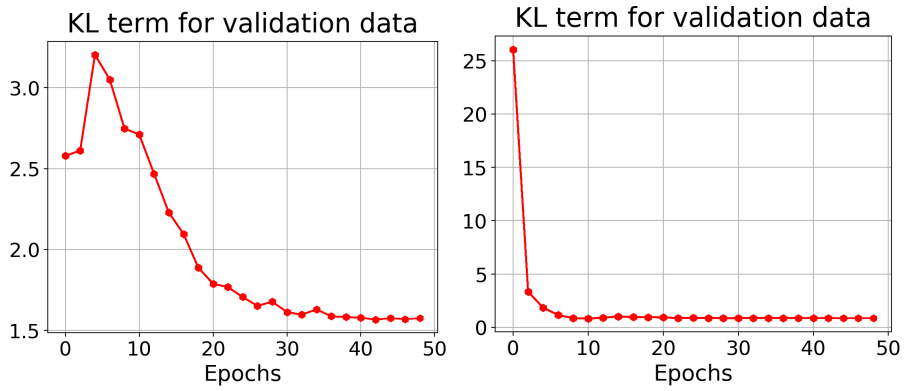
4.3.3 Combined Datasets

All four MIDI music training datasets were merged to investigate how using a larger data set will influence testing results. Testing and validation is still performed using the test/validation data for each individual data set. All dimensions are set as in the original replication task. Results are shown in line 4 of Table 1. Using the merged datasets for training, i.e. increasing and diversifying the amount of data for training, does not change the results compared to the results for the original replication task. Merging the training datasets only decreases the ELBO values slightly. The slight decrease in ELBO (i.e. worse performance) could be explained by the training data being more diverse, imbalanced and less informative for each individual music dataset, thereby cancelling out the benefit of an increased amount of training data.

4.3.4 Evolution of KL-divergence during Training

The KL term in the expression for ELBO is plotted for the validation sets. The results are shown in Figure 10. As training progresses and the ELBO term increases, the KL term decreases, as expected. This implies the variational posterior distribution is getting closer to the true posterior distribution.

Figure 10.(a) shows the evolution of the KL term for the replication experiment using the original SRNN architecture and model dimensions for the Nottingham dataset. The KL term appears to converge to approximately 1.5. Figure 10.(b) shows the convergence of the KL term to approximately 0.8 for the original SRNN architecture and model dimensions for the JSB Chorales dataset. Convergence of the KL term to approximately 1.0 is representative for similar experiments conducted on other datasets and using different SRNN model dimensions.



(a) (b)

Figure 10: (a): Evolution of the KL-term during training with the Nottingham dataset. (b): Evolution of KL-term during training with the JSB data set of settings $|a| = 300$, $|d| = 300$ and $|z| = 125$.

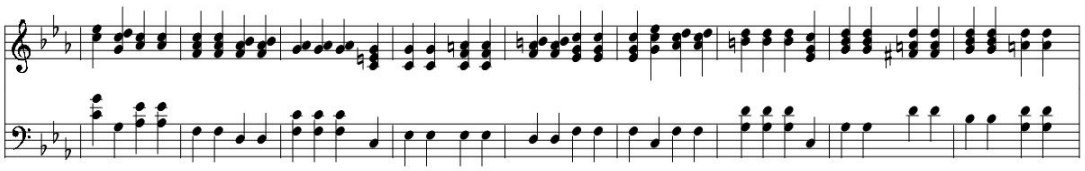
In Figure 9, the ELBO for the SRNN model with latent variable dimensions $|a| = 300$, $|d| = 300$ and $|z| = 125$ is an aberration from the observed trend for varying dimensions of the SRNN and the ELBO value is lower than expected. To investigate this the KL-term is observed during training and shown in plot (a) in figure 10. Vanishing KL does not appear to be the problem (KL-term approaching zero), however, the KL-term is lower than for other settings of the latent variable dimensions, where the terms converge similar as for the Nottingham data shown at around 1.5.

5 Discussions

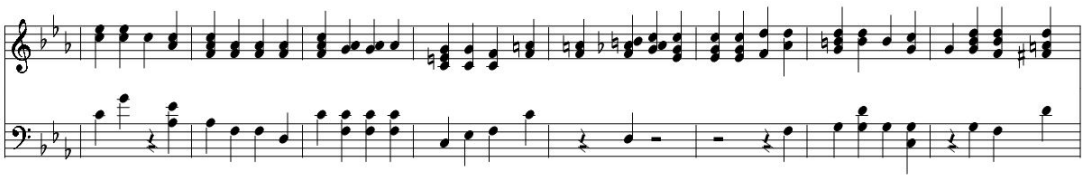
The paper converts the midi music into piano roll representation where for each time instance a vector of 88 dimensions represents the notes that are active. All these midi files can be found in https://drive.google.com/open?id=1iNUso_FjiJPMP9I-awChWsC1MMMOITKD.

Figure 12 shows that the overall active notes follow the same pattern in the input and output. However, in the output music some notes are shorter compared to the input. The same results can be observed after printing the Midi into sheet music. Some of the notes are replaced with breaks making the notes less coherent. JSB Choral music has simple tempo and less variation, which is close to the piano roll representation.

Similar results are observed in the Piano dataset, where most notes stay active for shorter length. This is obvious in the piano roll in Figure 14. The Piano dataset contains piano music that has complex tempo variation and note combinations. Piano roll representation may not be a good way to fully represent the music. This can also explain the poor ELBO performance.



(a) JSB Input.



(b) JSB Output.

Figure 11: JSB example music in music sheet

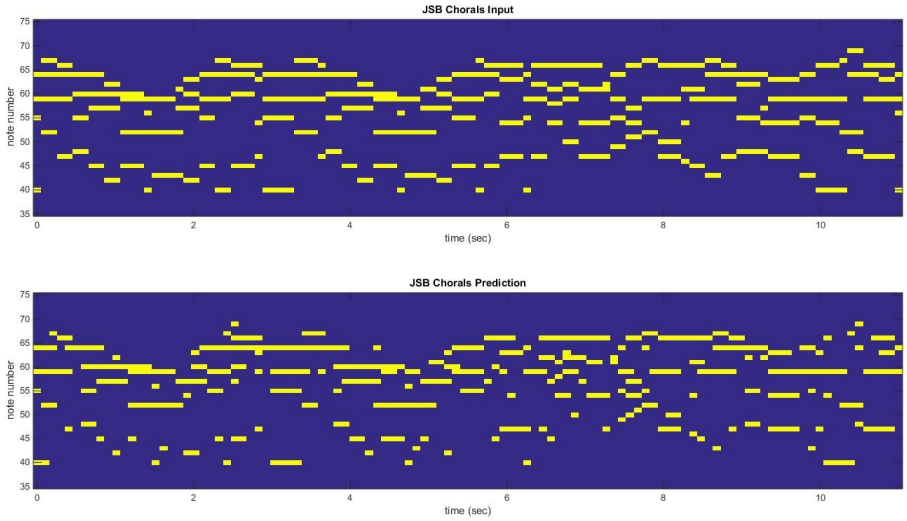
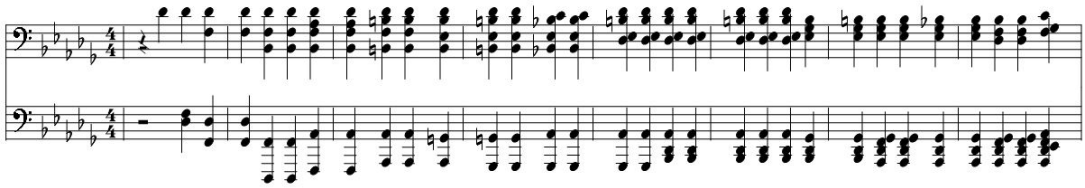
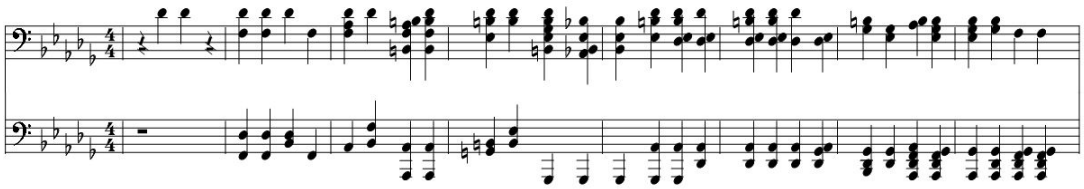


Figure 12: Music Roll of an Input Test Music and its Predictive Output for the JSB set.



(a) Piano Input.



(b) Piano Output.

Figure 13: Piano example music in music sheet

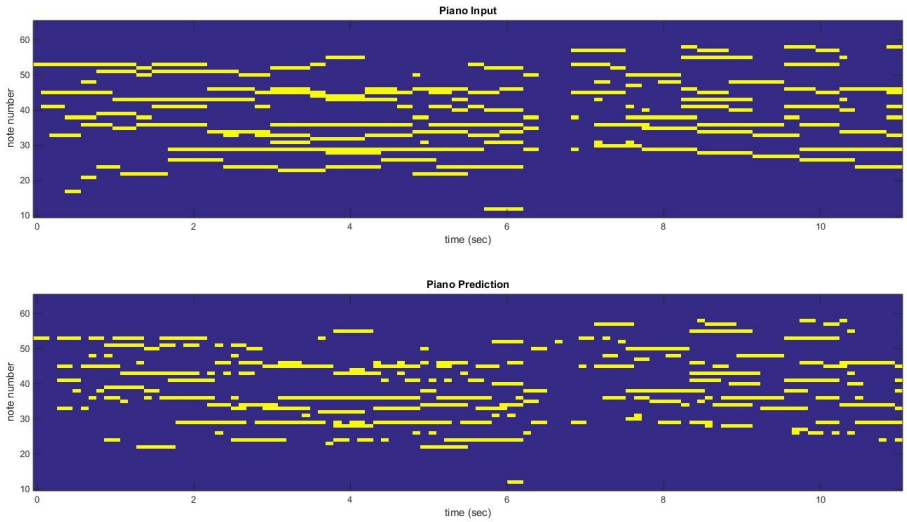


Figure 14: Music Roll of an Input Test Music and its Predictive Output for the piano-midi dataset.

Nottingham music example is shown in Figure 15, where the background notes stay constant most of the time

along with a varying single note. The background notes in the output are almost identical due to its simplicity. The single note, however, is different from the input. In Table 1, Nottingham dataset has the best ELBO value. This may be due to the stability of the background notes in the dataset.

Figure 17 shows the original music from Nottingham provided by http://www-etud.iro.umontreal.ca/~boulanni/icml2012. The input piano represents the notes at a constant pace, which removes the variation information in the note length and tempo changes. This results in the music sounding unnatural. It is noticeable that there is a random low note at the beginning of the output music, which is because the model has no information prior the first note.

Although Midi data itself contains rich information about the music, when the music is converted into training data (piano roll), most of these information are lost. To generate music that are more natural, it is important to predict not only notes but also other information that are contained the in the midi file. This is very different from working with speech. It is assumed that if features can be extracted from music waveforms the same way that speech is processed, similar architecture may produce good results for both speech and music generation.

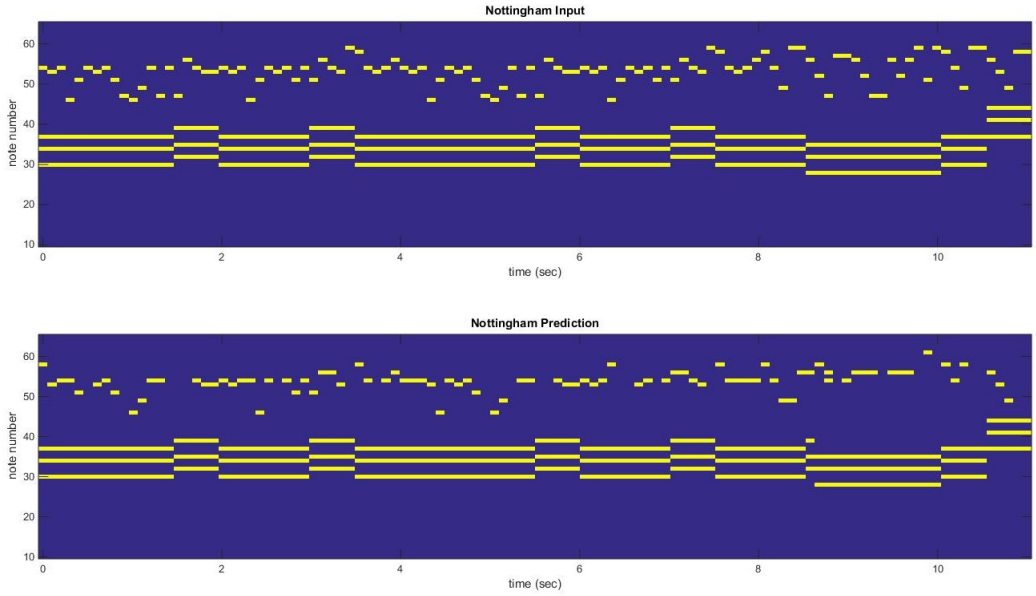
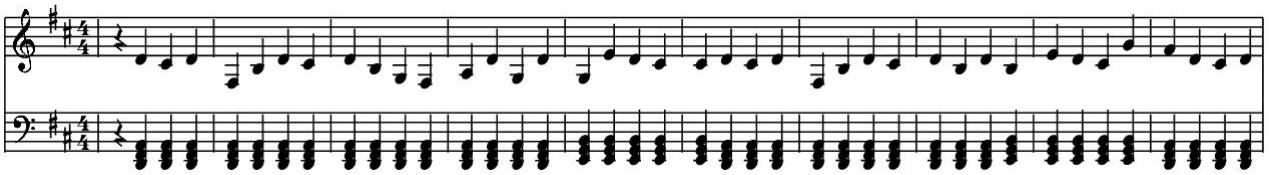
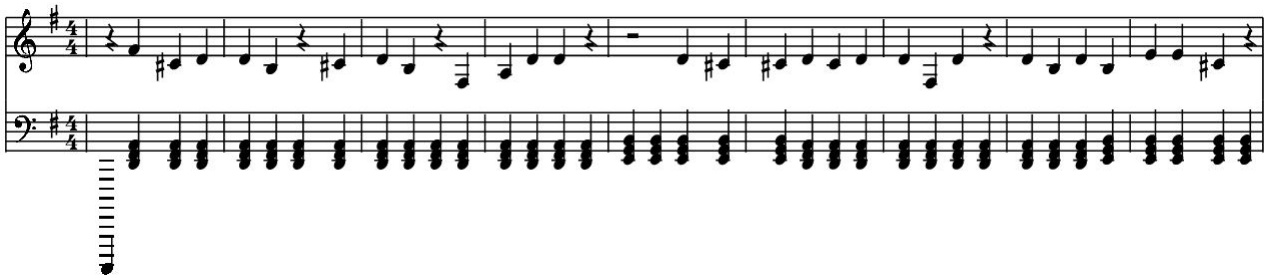


Figure 15: Music Roll of an Input Test Music and its Predictive Output for the Nottingham dataset.



(a) Nottingham Input.



(b) Nottingham Output.

Figure 16: Nottingham example music in music sheet

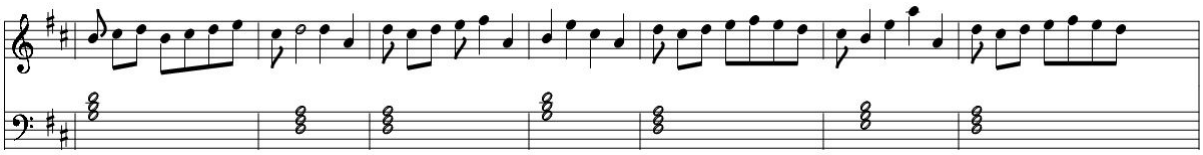


Figure 17: Nottingham Original music Representation

6 Conclusion and Future Work

Success in this kind of latent variable autoregressive models has been a proof that incorporating noise at a high level is helpful for modelling complex sequential data. Comparing to STORN and VRNN, a major novelty of this paper is the use of a backward RNN as inference model. Backward RNNs provides a summary of the entire future, which makes the posterior inference more accurate and powerful. Furthermore, this work provides a clear separation between stochastic and deterministic layers, which makes the inference network easier. Only do the stochastic variable need to be referred. The deterministic states are accurate throughout the sequence.

For future work the models architecture could be improved and parameters tuned to specifically fit the music data, since the SRNN model from the paper is build for speech and only modified to accommodate music data. Music data of different complexity as Nottingham and piano-midi.de for example also require different settings for optimal results.

References

[1] J. Bayer and C. Osendorfer. Learning Stochastic Recurrent Networks. *ArXiv e-prints*, November 2014.

[2] Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks.

[3] Yoshua Bengio, Gregoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 552–560, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[4] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription.

[5] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. 2012.

[6] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *CoRR*, abs/1611.02731, 2016.

[7] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C. Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *CoRR*, abs/1506.02216, 2015.

[8] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers.

[9] Zhe Gan, Chunyuan Li, Ricardo Henao, David Carlson, and Lawrence Carin. Deep temporal sigmoid belief networks for sequence modeling.

[10] A. Goyal, A. Sordoni, M.-A. Côté, N. R. Ke, and Y. Bengio. Z-Forcing: Training Stochastic Recurrent Networks. *ArXiv e-prints*, November 2017.

[11] Shixiang Gu, Zoubin Ghahramani, and Richard E. Turner. Neural adaptive sequential monte carlo.

[12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[13] Francis Ramsey and Tim McCormick. *Sound and Recording: An Introduction, Chapter 13: MIDI*. 2002.

[14] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. *CoRR*, abs/1605.06069, 2016.

Appendix

A Code: Save midi file into Text

```
## --- Save midi file into text --- ##
## Format of the text is binary matrix for sequence of 88 notes.

import numpy as np

JSB_output = np.load('output.jsb.npz')
JSB_output_test01 = JSB_output['p_out_test'][0]
JSB_input = np.load('data/jsb_raw_batchsize16_seqlen100.npz')
JSB_input_test01 = JSB_input['x_test'][0]

np.savetxt('JSB.input.test.01', JSB_input_test01.astype(int), fmt='%d')
np.savetxt('JSB.output.test.01', JSB_output_test01.astype(int), fmt='%d')
```

B Code: Write midi music

```
%% --- Read in the midi data --- %%
JSB_input = dlmread('JSB.input.test.01')';
JSB_output = dlmread('JSB.output.test.01')';
time = [0:0.1:0.1*110];
note_id = [35:1:75];

figure;
subplot(2,1,1)
imagesc(time, note_id, JSB_input(35:75,1:110))
axis xy;
xlabel('time (sec)')
ylabel('note number')
title 'JSB Chorals Input'

subplot(2,1,2)
imagesc(time, note_id, JSB_output(35:75,1:110))
axis xy;
xlabel('time (sec)')
ylabel('note number')
title 'JSB Chorals Prediction'

%% --- Write piano-roll into midi file --- %%
%% write input
N = 88; % number of notes
M = zeros(N*5,6);

M(:,1) = 1;  % all in track 1
M(:,2) = 1;  % all in channel
M(:,4) = 90; % velocity constant at 90

count = 0;
for t=1:110
   note = JSB_input(:,t);
   for i=1:length(note)
       if note(i)==1
           count = count + 1;
           M(count,3) = i+8;
           M(count,5) = 0.5*t;
       end
   end
end
M(:,6) = M(:,5)+.5;

JSB_input_midi = matrix2midi(M);
writemidi(JSB_input_midi, 'JSB_input_midi.mid');

%% write output
N = 88; % number of notes
M = zeros(N*5,6);

M(:,1) = 1;  % all in track 1
M(:,2) = 1;  % all in channel
```

```
M(:,4) = 90; % velocity constant at 90

count = 0;
for t=1:110
   note = JSB_output(:,t);
   for i=1:length(note)
       if note(i)==1
           count = count + 1;
           M(count,3) = i+8; % assume the first note is C3(48), add 8 to original note number
           M(count,5) = 0.5*t;
       end
   end
end
M(:,6) = M(:,5)+.5;

JSB_output_midi = matrix2midi(M);
writemidi(JSB_output_midi, 'JSB_output_midi.mid');
```