STCN: STOCHASTIC TEMPORAL CONVOLUTIONAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Convolutional architectures have recently been shown to be competitive on many sequence modelling tasks when compared to the de-facto standard for sequence modeling of recurrent neural networks (RNNs), while providing significant computational advantages due to inherent parallelism. However, there currently remains a performance gap to more expressive stochastic RNN variants, especially those with several layers of dependent random variables. In this work, we propose stochastic temporal convolutional networks (STCNs) a novel architecture that combines the computational advantages of temporal convolutional networks (TCN) with the representational power and robustness of stochastic latent spaces. In particular, we propose a hierarchy of latent variables, tightly integrated with blocks of dilated convolutions, that captures temporal dependencies at different time-scales. We show that the proposed architecture achieves state of the art log-likelihoods across several tasks. Finally, we show that the model is capable of predicting high quality synthetic samples over a long-range temporal horizon in a variety of tasks including modeling of hand written individual digits and text.

1 INTRODUCTION

Generative modeling of sequence data requires capturing long-term dependencies and learning of correlations between output variables at the same time-step. Recurrent neural networks (RNNs) and its variants have been very successful in a vast number of problem domains which rely on sequential data. However, recent work in in audio synthesis, language modeling and machine translation tasks (Dauphin et al., 2016; Van Den Oord et al., 2016; Dieleman et al., 2018; Gehring et al., 2017) has demonstrated that temporal convolutional networks (TCNs) can achieve at least competitive performance without relying on recurrence and hence improving computational aspects of sequence modeling tasks. However, while TCN architectures have been shown to perform similar or better than standard recurrent architectures on selected tasks (Van Den Oord et al., 2016; Bai et al., 2018), there currently remains a performance gap to more recent stochastic RNN variants (Bayer & Osendorfer, 2014; Chung et al., 2015; Fabius & van Amersfoort, 2014; Fraccaro et al., 2016; Goyal et al., 2017). In this work we propose a new architecture that combines the computational advantages of TCNs with the representational power and robustness of stochastic latent spaces.

Both RNNs and TCNs model the joint probability distribution over sequences by decomposing the distribution over discrete time-steps. In other words, such models are trained to predict the next step, given all previous time-steps. RNNs are able to model long-term dependencies by propagating information through their deterministic hidden state, acting as an internal memory. However, this hidden state requires sequential computation of time-steps. In contrast, TCNs leverage large receptive fields by stacking many dilated convolutions, allowing them to model even longer time scales up to the entire sequence length. It is noteworthy that there is no explicit temporal dependency between the model outputs. Because filters are shared between layers computations can be performed in parallel. The TCN architecture also introduces a temporal hierarchy: the upper layers have access to longer input sub-sequences and learn representations at a larger time scale. The local information from the lower layers is propagated through the hierarchy by means of residual and skip connections (Van Den Oord et al., 2016; Bai et al., 2018) (see figure 1, inset on the right).

The introduction of uncertainty to RNNs can improve the performance of modeling complex natural sequences and stochastic RNNs (Bayer & Osendorfer, 2014; Chung et al., 2015; Fabius & van

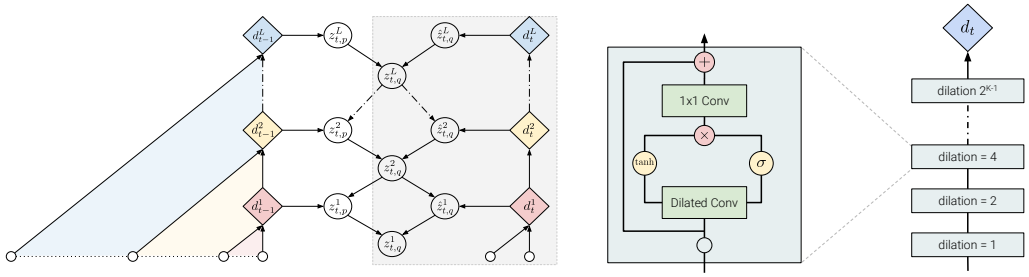

Figure 1: (*Left*): The computational graph of STCNs. The area shaded in gray denotes the inference model. The approximate posterior q is conditioned on $\mathbf{d}_t$ and is updated by the prior p which is conditioned on the TCN representations of the previous time-step $\mathbf{d}_{t-1}$. The random latent variables at the upper layers have access to a longer history while the lower layers receive more recent input steps. (*Right*): a deterministic block $\mathbf{d}_t$, consisting of K Wavenet blocks (Van Den Oord et al., 2016) (shown in inset). The dilation exponentially increases within the block.

Amersfoort, 2014; Fraccaro et al., 2016; Goyal et al., 2017) are now the state of the art in many sequence modeling tasks. Probabilistic inference is achieved by augmenting the deterministic internal state with stochastic latent variables within the variational auto-encoding framework (Kingma & Welling, 2013). Existing architectures mostly vary in the way they employ the latent variable and parametrize the approximate posterior for variational inference. While the models proposed by Goyal et al. (2017) and Fraccaro et al. (2016) learn a predictive latent variable to capture the future states of the sequence. Chung et al. (2015) and Bayer & Osendorfer (2014) use the latent random variable to capture high-level information causing the variability observed in sequential data.

Motivated by the simplicity and computational advantages of TCNs and the robustness and performance of stochastic RNNs, we introduce stochastic temporal convolutional networks (STCN) by incorporating a hierarchy of stochastic latent variables into TCNs which enables learning of representations at many timescales. However, due to the absence of an internal state in TCNs, introducing latent random variables analogously to stochastic RNNs is not feasible. Furthermore, defining conditional random variables across time-steps would result in breaking the parallelism of TCNs and is hence undesirable. We propose to augment stacked convolutional layers with stochastic latent variables via the variational auto-encoding framework (Kingma & Welling, 2013; Rezende et al., 2014; Sønderby et al., 2016). In contrast to stochastic RNNs, the random latent variables are not auxiliary but capture all relevant information to enable the prediction for the next step.

Crucially, in STCN the latent random variables are arranged in correspondence to the temporal hierarchy of the TCN blocks which effectively distributes the random variables over the various timescales (see figure 1, left). In contrast to related architectures (e.g., (Gulrajani et al., 2016; Sønderby et al., 2016)), the latent variables at the upper layers capture information at long-range time scales, whereas latent variables in lower layers capture more local, recent details (see figure 1, left). We propose two different inference networks. In the canonical configuration, samples from each latent variable are passed down from layer to layer and only one sample from the lowest layer is used to condition the prediction of the output. In the second configuration, called STCN-dense, we take inspiration from recent CNN architectures (Huang et al., 2017) and utilize samples from all latent random variables via concatenation before computing the final prediction.

2 Background

In this section we discuss related work and introduce important building blocks used in our model.

Auto-regressive models such as RNNs and TCNs factorize the joint probability of a variable-length sequence $\mathbf{x} = \{x_1, \ldots, x_T\}$ as a product of conditionals as follows:

$$p_\theta(\mathbf{x}) = \prod_{t=1}^{T} p_\theta(x_t | x_{1:t-1}) \quad , \tag{1}$$

where the joint distribution is parametrized by θ. The prediction at each time-step is conditioned on all previous observations. The observation model is frequently chosen to be a Gaussian or Gaussian mixture model (GMM) for real-valued data, and a categorical distribution for discrete-valued data.

2.1 RECURRENT NEURAL NETWORKS

An RNN captures temporal dependencies by recursively processing each input, while updating an internal state h_t at each time-step via its state-transition function:

$$h_t = f^{(h)}(x_t, h_{t-1}) \quad , \tag{2}$$

where $f^{(h)}$ is a deterministic transition function. To enable robust optimization $f^{(h)}$ is often implemented with LSTM (Hochreiter & Schmidhuber, 1997) or GRU (Cho et al., 2014) cells. Since the computation of h_t depends on h_{t-1}, the computation has to be performed sequentially. The final output probability distribution is then computed via:

$$p(x_t|x_{1:t-1}) = f^{(o)}(h_t) \quad , \tag{3}$$

where $f^{(o)}$ is the observation model mapping the hidden state h_t to a distribution of possible outputs.

2.2 TEMPORAL CONVOLUTIONAL NETWORKS

In TCNs the joint probabilities in Eq. (1) are parametrized by a stack of convolutional layers. *Causal convolutions* are the central building block of such models and are designed to be asymmetric such that the model has no access to future information. In order to produce outputs of the same size as the input, zero-padding is applied at every layer. In the absence of a state transition function, a large receptive field is crucial in capturing long-range dependencies. To avoid the need for vast numbers of causal convolution layers, typically *dilated* convolutions are used. Exponentially increasing the dilation factor results in an exponential growth of the receptive field size with depth (Yu & Koltun, 2015; Van Den Oord et al., 2016; Bai et al., 2018).

The building blocks of Wavenet (Van Den Oord et al., 2016) and TCNs (Bai et al., 2018) are the two candidates to build our model. The major difference lies in the activation functions. In audio synthesis tasks, gated activation units (van den Oord et al., 2016) have been reported to perform better. Bai et al. (2018) use the ReLU activation function in several classification and language modeling tasks. The STCN model with Wavenet blocks performed better in our experiments, and hence we use the building blocks of Wavenet (Van Den Oord et al., 2016) in this paper.

The stochastic variables $\mathbf{z}$ are conditioned on the deterministic TCN representations $\mathbf{d}$ (see figure 1, right). We compute $d^l, l = 1 \dots L$ by stacking K Wavenet blocks over the previous d^{l-1}:

$$d_t^l = \text{Conv}^{(l)}(d_t^{l-1}) \tag{4}$$

2.3 NON-SEQUENTIAL LATENT VARIABLE MODELS

VAEs (Kingma & Welling, 2013; Rezende et al., 2014) introduce a latent random variable $\mathbf{z}$ to learn the variations in the observed non-sequential data where the generation of the sample $\mathbf{x}$ is conditioned on the latent variable $\mathbf{z}$. The joint probability distribution is defined as:

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z}) \quad , \tag{5}$$

and parametrized by θ. Optimizing the marginal likelihood is intractable due to the non-linear mappings between $\mathbf{z}$ and $\mathbf{x}$ and the integration over $\mathbf{z}$. Instead the VAE framework introduces an approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and optimizes a lower-bound on the marginal likelihood:

$$\log p_\theta(\mathbf{x}) \geq -KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \quad , \tag{6}$$

where KL denotes the Kullback-Leibler divergence.

Typically the prior $p_\theta(\mathbf{z})$ and the approximate $q_\phi(\mathbf{z}|\mathbf{x})$ are chosen to be in simple parametric form, such as a Gaussian distribution with diagonal covariance, which allows for an analytical calculation of the KL-term in Eq. (6) and training of the inference network q_ϕ via standard backpropagation.

Rezende et al. (2014) propose Deep Latent Gaussian Models (DLGM) and Sønderby et al. (2016) propose Ladder Variational Autoencoder (LVAE). In both frameworks the latent variables $\mathbf{z}$ are split into layers and conditioned on the preceding stochastic layer. LVAEs improve upon DLGMs via implementation of a top-down hierarchy both in the generative and inference model. The approximate posterior is computed via a precision-weighted update of the approximate likelihood (i.e., the

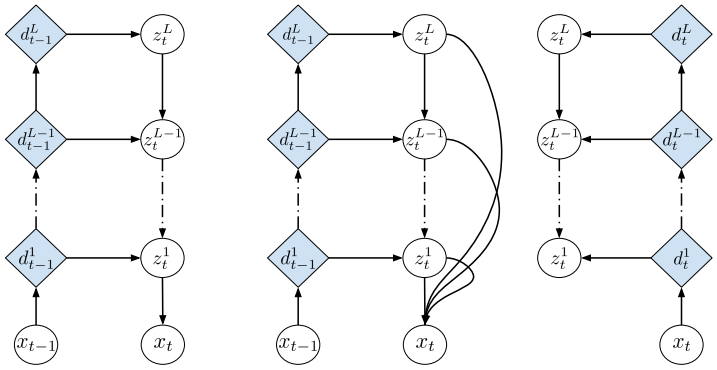

Figure 2: Generative models of STCN (*left*) and STCN-dense (*middle*), and the inference model (*right*). Both variants have the same inference model. Diamonds represents outputs of deterministic dilated convolution layers, x_t and z_t are observable inputs and latent random variables. The generative task is to predict the next step in the sequence, given all past steps. Note that in the STCN-dense variant the next step is conditioned on all latent variables z_t^l for $l = 1 \dots L$.

inference model) and prior (i.e., the generative model). Similarly, PixelVAE (Gulrajani et al., 2016) incorporates a hierarchical latent space decomposition and uses an autoregressive decoder. Our approach differs in the crucial aspects that STCNs use dynamic priors at every level of the hierarchy. Furthermore, our latent space is designed such that each latent random variable captures information over a different time-scale and hence contributes complementary information to the final prediction.

2.4 Stochastic RNNs

Random latent variables increase the representational power of deep learning models and can yield better generative performance. The VAE framework itself has been extended for sequential data, where a latent variable z_t exists for each sequence step x_t. The joint distribution $p_\theta(\mathbf{x}, \mathbf{z})$ is modeled via an auto-regressive model which results in the following factorization:

$$p_\theta(\mathbf{x}, \mathbf{z}) = \prod_{t=1}^{T} p_\theta(x_t | z_{1:t}, x_{1:t-1}) p(z_t | x_{1:t-1}, z_{1:t-1}) \quad . \tag{7}$$

VAEs define a fixed prior with $\mathcal{N}(\mathbf{0}, \mathbf{I})$. In contrast, sequential variants define prior distributions conditioned on the RNN hidden state h_t and the input sequence $\mathbf{x}$ (Chung et al., 2015).

Bayer & Osendorfer (2014) define the hidden state of the RNN h_t as a function of the latent variable z_t with a fixed Gaussian prior where the latent variables z_t between time-steps are independent from each other. Chung et al. (2015) share the parameters of the RNN in the generative and inference networks. They define an interaction between the prior and the RNN state, and both are estimated recursively. The dynamic prior is conditioned on the previous RNN state h_{t-1} and in turn, the new RNN state uses the latent variable of the previous step z_{t-1}. In (Fraccaro et al., 2016) and (Goyal et al., 2017), the predictive latent variable $\mathbf{z}$ is interacting with the *backward* RNN states.

The mechanics of STCNs are related to those of VRNNs and LVAEs. Intuitively, the RNN state h_t is replaced by temporally independent TCN layers d_t. In the absence of an internal state, we define hierarchical latent variables z_t that are conditioned *vertically*, i.e., in the same time-step, but independent *horizontally*, i.e., across time-steps. We follow a similar approach to LVAEs (Sønderby et al., 2016) in defining the hierarchy in a *top-down* fashion and in how we estimate the approximate posterior. The inference network first computes the approximate likelihood, and then this estimate is corrected by the prior, resulting in the approximate posterior. The TCN layers d^l are shared between the inference and generator networks, analogous to VRNNs (Chung et al., 2015).

3 Stochastic Temporal Convolutional Networks

Figure 2 depicts the proposed STCN as a graphical model. STCNs consist of two main modules: the deterministic temporal convolutional network and the stochastic latent variable hierarchy. For

a given input sequence $\mathbf{x} = \{x_t\}, t = 1 \dots T$ we first apply dilated convolutions over the entire sequence to compute a set of deterministic representations $d_t^l, l = 1 \dots L$. Here, d_t^l corresponds to the output of a block of dilated convolutions at layer l and time-step t. The output d_t^l is then used to update a set of random latent variables z_t^l arranged to correspond with different time-scales.

To preserve the *parallelism* of TCNs, we do not introduce an explicit dependency between different time-steps. However, we suggest that conditioning a latent variable z_t^{l-1} on the preceding variable z_t^l implicitly introduces temporal dependencies. Importantly the random latent variables in the upper layer have access to a larger receptive field due to its deterministic input d_{t-1}^l, whereas latent random variables in lower layers are updated with different, more local information. However, the latent variable z_t^{l-1} may receive longer-range information from z_t^l both during training and inference. As the update process progresses in a top-down order the context of the latent variables changes from long-range *global* patterns to short-range *local* variation.

The generative and inference models are jointly trained by optimizing a step-wise variational lower bound on the log-likelihood (Kingma & Welling, 2013; Rezende et al., 2014). In the following sections we describe these components and build up the lower-bound for a single time-step t.

3.1 Generative Model

Each sequence step x_t is generated from a set of latent variables z_t, split into layers as follows:

$$\begin{aligned} p_\theta(z_t|d_{t-1}) &= p_\theta(z_t^L|d_{t-1}^L) \prod_{l=1}^{L-1} p_\theta(z_t^l|z_t^{l+1}, d_{t-1}^l) \quad , \\ p_\theta(z_t^l|\cdot) &= \mathcal{N}(\mu_{t,p}^l, \sigma_{t,p}^l) \\ [\mu_{t,p}^l, \sigma_{t,p}^l] &= f^{(l)}(z_t^{l+1}, d_{t-1}^l) \quad . \end{aligned} \tag{8}$$

Here the prior is modeled by a Gaussian distribution with diagonal covariance, as is commonly done in the VAE framework. The subscript p denotes that μ and σ belong to the generative distribution. For the inference distribution we use the subscript q. The distributions are parameterized by a neural network $f^{(l)}$ and conditioned on (1) samples z_t^{l+1} from the previous level at the same time-step and (2) the representation d_{t-1}^l computed by the dilated convolutions from the previous time-step.

We propose two variants of the observation model. In the non-sequential scenario, the observations are defined to be conditioned on only the last latent variable in the hierarchy, i.e., $p_\theta(x_t|z_t^1)$, following Sønderby et al. (2016); Gulrajani et al. (2016) and Rezende et al. (2014) our STCN variant uses the same observation model, allowing for an efficient optimization. However, latent units are likely to become inactive during training in this configuration (Burda et al., 2015; Bowman et al., 2015) resulting in a loss of representational power.

The latent variables at different layers are conditioned on different contexts due to the inputs d_t^l. Hence, the latent variables are expected to capture complementary aspects of the temporal context. To propagate the information all the way to the final prediction and to ensure that gradients flow through all layers, we take inspiration from Huang et al. (2017) and directly condition the output probability on samples from *all* latent variables. We call this variant of our architecture *STCN-dense*.

The final predictions are then computed by the respective observation functions:

$$p_\theta(x_t|z_t) = f^{(o)}(z_t^1) \quad \text{and} \quad p_\theta^{dense}(x_t|z_t) = f^{(o)}(z_t^1 \dots z_t^L) \quad . \tag{9}$$

3.2 Inference Model

In the original VAE framework the inference model is defined as a bottom-up process, where the latent variables are conditioned on the stochastic layer below. Furthermore, the parameterization of the prior and approximate posterior distributions are computed separately (Burda et al., 2015; Rezende et al., 2014). In contrast, Sønderby et al. (2016) propose a top-down dependency structure shared across the generative and inference models. From a probabilistic point of view, the approximate Gaussian likelihood, computed bottom-up by the inference model, is combined with the Gaussian prior, computed top-down from the generative model. We follow a similar procedure in computing the approximate posterior.

First, the parameters of the approximate likelihood are computed for each stochastic layer l:

$$\hat{\mu}^l_{t,q}, \hat{\sigma}^l_{t,q} = f^{(l)}(z^l_t, d^l_t) \quad , \tag{10}$$

followed by the downward pass, recursively computing the prior and approximate posterior by precision-weighted addition:

$$\sigma^l_{t,q} = \frac{1}{(\hat{\sigma}^l_{t,q})^{-2} + (\sigma^l_{t,p})^{-2}} \quad , \tag{11}$$

$$\mu^l_{t,q} = \sigma^l_{t,q}(\hat{\mu}^l_{t,q}(\hat{\sigma}^l_{t,q})^{-2} + \mu^l_{t,p}(\sigma^l_{t,p})^{-2}) \quad . \tag{12}$$

Finally, the approximate posterior has the same decomposition as the prior (see Eq. (8)):

$$\begin{aligned} q_\phi(z_t|x_t) &= q_\phi(z^L_t|d^L_t) \prod_{l=1}^{L-1} q_\phi(z^l_t|z^{l+1}_t, d^l_t) \quad , \\ q_\phi(z^l_t|\cdot) &= \mathcal{N}(\mu^l_{t,q}, \sigma^l_{t,q}) \quad , \\ d^1_t &= \text{Conv}^{(1)}(x_{1:t}) \quad , \end{aligned} \tag{13}$$

where $\text{Conv}^{(1)}$ denotes the causal convolution operations in Eq. (4) applied to the input sequence $\mathbf{x}$. Note that both the inference and generative network share the same parameters including the dilated convolution layers d and non-linear transformation functions $f^{(l)}$, hence $\theta = \phi$. The inference and generative models only differ in the type of information they are conditioned on, namely the dilated convolution outputs d from the current step t and the previous step $t-1$, respectively.

3.3 Learning

The variational lower-bound on the log-likelihood at time-step t can be defined as follows:

$$\begin{aligned} \log p(x_t) &\geq \mathbb{E}_{q_\phi(z_t|x_t)}[\log p_\theta(x_t|z_t)] - D_{KL}(q_\phi(z_t|x_t)||p_\theta(z_t)) \\ &= \mathbb{E}_{q_\phi(z^1_t \dots z^L_t|x_t)}[\log p_\theta(x_t|z^1_t \dots z^L_t] - D_{KL}(q_\phi(z^1_t \dots z^L_t|x_t)||p_\theta(z^1_t \dots z^L_t)) \\ \mathcal{L}_t(\theta, \phi; x_t) &= \mathcal{L}^{Recon}_t + \mathcal{L}^{KL}_t. \end{aligned} \tag{14}$$

Using the decompositions from Eq. (8), (13), the Kullback-Leibler divergence term takes the form:

$$\mathcal{L}^{KL}_t = -\sum_{l=1}^{L} \mathbb{E}_{q_\phi(z^{l+1}_t|\cdot)}[D_{KL}(q_\phi(z^l_t|z^{l+1}_t, d^l_t)||p_\theta(z^l_t|z^{l+1}_t, d^l_{t-1}))] \quad , \tag{15}$$

where $p_\theta(z^{L+1}_t|\cdot) = q_\phi(z^{L+1}_t|\cdot) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. The Kullback-Leibler divergence loss term is the same for the STCN and STCN-dense variants. The reconstruction term $\mathcal{L}^{Recon}_t$ however differs between the variants. In STCN we only use samples from the lowest layer of the hierarchy, whereas in STCN-dense we use all latent samples in the observation model:

$$\mathcal{L}^{Recon-STCN}_t = \mathbb{E}_{q_\phi(z^1_t \dots z^L_t|x_t)}[\log p_\theta(x_t|z^1_t)] \quad , \tag{16}$$

$$\mathcal{L}^{Recon-dense}_t = \mathbb{E}_{q_\phi(z^1_t \dots z^L_t|x_t)}[\log p_\theta(x_t|z^1_t \dots z^L_t] \quad . \tag{17}$$

In the dense variant, samples drawn from the latent variables z^l_t are carried over the dense connections. Similar to Maaløe et al. (2016), the expectation over z^l_t variables are computed by Monte Carlo sampling using the reparameterization trick (Kingma & Welling, 2013; Rezende et al., 2014).

Please note that the computation of $\mathcal{L}^{Recon-dense}_t$ does not introduce any additional computational cost. In STCN, all latent variables have to be visited in terms of ancestral sampling in order to draw the latent sample z^1_t for the observation x_t. Similarly in STCN-dense, the same intermediate samples z^l_t are used in the prediction of x_t.

One alternative option to use the latent samples could be to sum individual samples before feeding them into the observation model, i.e., $sum([z^1_t \dots z^L_t])$, (Maaløe et al., 2016). We empirically found that it does not work well in STCN-dense. Instead, we concatenate all samples $[z^1_t \circ \dots \circ z^L_t]$ analogously to DenseNet (Huang et al., 2017) and (Kaiser et al., 2018).

Models	Iamondb		Deepwriting		TIMIT	
	Gaussian	GMM	Gaussian	GMM	Gaussian	GMM
Wavenet *	1053	1381	1.21	2.34	-7443	30188
Wavenet-dense *	1030	1380	1.19	2.47	-8579	30636
RNN Chung et al. (2015) (*)	1016	1358	1.34	1.98	-1900	26643
VRNN Chung et al. (2015) (*)	$\geq$ 1337	$\geq$ 1384	$\geq$ 1.84	$\geq$ 2.57	$\geq$ 28805	$\geq$ 28982
STCN	$\geq$ 1308	$\geq$ 1376	$\geq$ 1.79	$\geq$ 2.38	$\geq$ 36553	$\geq$ 32588
STCN-dense	$\geq$ 1685	$\geq$ **1743**	$\geq$ 2.44	$\geq$ **2.66**	$\geq$ 38450	$\geq$ 34567
STCN+ kla	$\geq$ 1362	$\geq$ 1374	$\geq$ 1.71	$\geq$ 2.34	$\geq$ **48066**	$\geq$ 45063
STCN-dense+ kla	$\geq$ 1714	$\geq$ 1719	$\geq$ 2.27	$\geq$ 2.42	$\geq$ 43540	$\geq$ 42591
Z-forcing Goyal et al. (2017)	n/a	n/a	n/a	n/a	$\geq$ **70469**	n/a
SRNN Fraccaro et al. (2016)	n/a	n/a	n/a	n/a	60550	n/a

Table 1: Average log-likelihood per sequence on TIMIT and Iamondb datasets. On the Deepwriting dataset we report average log-likelihood per time-step. Asterisks * denote our own implementation of the model. Asterisks in brackets (*) indicate that we only used our own re-implementation for the Deepwriting dataset. Note that (Goyal et al., 2017; Fraccaro et al., 2016) use future information.

4 EXPERIMENTS

We evaluate the proposed variants STCN and STCN-dense both quantitatively and qualitatively on three sequence modelling tasks: 1) modeling of digital handwritten text, 2) speech modeling and, 3) modeling of images of handwritten digits (sequential MNIST). We compare with vanilla TCNs, RNNs, VRNNs and state-of-the art models on the corresponding tasks.

In our experiments we use two variants of the Wavenet model: (1) the original model proposed in (Van Den Oord et al., 2016) and (2) a variant that we augment with skip connections analogously to STCN-dense. This additional baseline evaluates the benefit of learning *multi-scale* representations in the deterministic setting.

Furthermore, we compare performance using both unimodal Gaussian or multi-modal Gaussian Mixture Model (GMM) as the observation model for real-valued data (Graves, 2013; Chung et al., 2015), demonstrating that the main improvement in performance stems from our proposed latent-space structure.

Handwritten text: Iamondb and Deepwriting datasets consist of digital handwriting sequences where each time-step contains real-valued (x, y) pen coordinates and a binary *pen-up* event. The Iamondb data is split and pre-processed as done in (Chung et al., 2015). Aksan et al. (2018) extend this dataset with additional samples and better pre-processing. We leverage this dataset mostly for qualitative analysis. Figure 3 shows samples from several models, note how ours maintains visual style, while Wavenet diverges and VRNN produces lower quality samples.

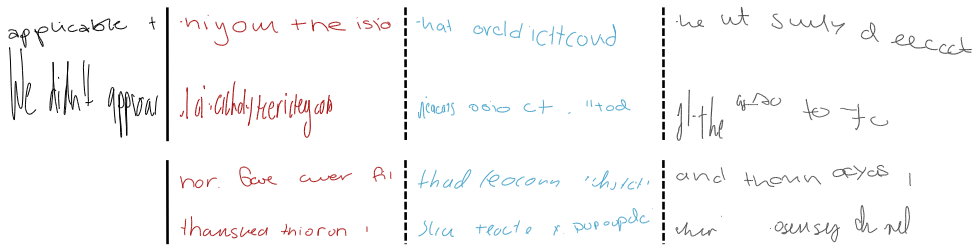

Figure 3: Biased (*top*) and purely synthetic (*bottom*) samples from STCN-dense (*red*), Wavenet-dense (*blue*), VRNN (*gray*).

Table 1 reveals that again both our variants outperform the vanilla variants of TCNs and RNNs on Iamondb. While the stochastic VRNN is competitive wrt to the STCN variant, it is outperformed by the STCN-dense version. The same relative ordering is maintained on the Deepwriting dataset, indicating that the proposed architecture is robust across tasks and datasets.

Models	MNIST Bernoulli
Wavenet *	80.42
Wavenet-dense *	80.35
RNN *	80.39
VRNN *	$\geq$ 80.22
STCN	$\geq$ 81.08
STCN-dense	$\geq$ **60.25**

Models	MNIST Bernoulli
Pixel VAE Gulrajani et al. (2016)	$\approx$ 79.02
P-Forcing$_{\text{(3-layer)}}$ Goyal et al. (2016)	79.58
PixelRNN$_{\text{(1-layer)}}$ Van Den Oord et al. (2016)	80.75
PixelRNN$_{\text{(7-layer)}}$ Van Den Oord et al. (2016)	79.20
MatNets Bachman (2016)	78.50
Z-forcing Goyal et al. (2017)	$\leq$ 80.09

Table 2: Performance on MNIST dataset. A star * denotes own implementation of the model.

Speech modeling: TIMIT is a standard benchmark dataset in speech modeling. The dataset consists of samples from 630 speakers. We apply the same pre-processing as Chung et al. (2015). The models are trained and tested on 200 dimensional real-valued amplitudes.

Our STCN model outperforms the deterministic TCNs and RNN as well as the stochastic VRNN (Table 1). The deterministic networks suffer from overfitting on the TIMIT dataset. Here we observe the benefit of annealing the weight of the KL-divergence loss term $\mathcal{L}^{KL}$. STCN and STCN-dense variants can make better use of the latent space with the help of a *warm-up* procedure as proposed by Sønderby et al. (2016).

Both Z-forcing (Goyal et al., 2017) and SRNN (Fraccaro et al., 2016) outperform our models. This can be explained by (1) the dynamics of these models and (2) the difference in length of the training and test splits. During training these models learn long-range future dependencies by using both forward and backward RNN cells, whereas our models have only access to information up to the current time-step. Furthermore, the training samples only consist of 40 time-steps while the test sample length varies between 86 and 604. This significantly disadvantages the TCN based architectures which rely on increasing receptive field sizes (also at training time).

Handwritten digits: sequential MNIST contains binary images of handwritten, individual digits and is commonly used in evaluations of sequence modeling approaches. The task typically consists of predicting the next pixel given all pixels up to the current observation, or to generate entirely novel digits, while maintaining the visual appearance.

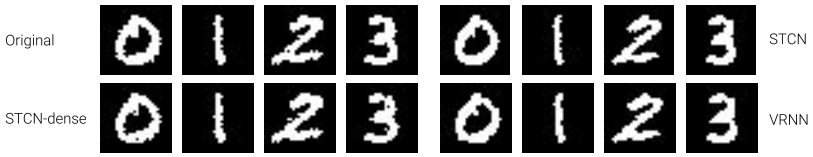

Figure 4: Original (*top-left*) and reconstructions of MNIST digits.

Table 2 summarizes the average log-likelihood per sequence of our approach and a number of state of the art baselines. On this dataset the, STCN performs similar to the VRNN and the STCN-dense model outperforms the other models by a large margin, most likely because it can leverage the increased model capacity more effectively. Figure 4 also shows that the lower log-likelihood score translates to preservation of high frequency details in the reconstructed samples, while the VRNN and STCN produce slightly more blurry images.

We furthermore note that across all our experiments the performance gain in the stochastic models with GMM output model is relatively small compared to the deterministic models. This suggests that indeed the addition of the latent space plays a larger role than the choice of output parametrization in modeling of natural sequences.

5 Conclusion

In this paper we proposed STCNs, a novel auto-regressive model, combining the computational benefits of convolutional architectures and expressiveness of hierarchical stochastic latent spaces. We have shown the effectivness of the approach across several sequence modelling tasks and datasets.

REFERENCES

Emre Aksan, Fabrizio Pece, and Otmar Hilliges. DeepWriting: Making Digital Ink Editable via Deep Generative Modeling. In *SIGCHI Conference on Human Factors in Computing Systems*, CHI '18, New York, NY, USA, 2018. ACM.

Philip Bachman. An architecture for deep, hierarchical generative models. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4826–4834. Curran Associates, Inc., 2016.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pp. 2980–2988, 2015.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. *arXiv preprint arXiv:1612.08083*, 2016.

Sander Dieleman, Aäron van den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. *arXiv preprint arXiv:1806.10474*, 2018.

Otto Fabius and Joost R van Amersfoort. Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*, 2014.

Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pp. 2199–2207, 2016.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

Anirudh Goyal ALIAS PARTH Goyal, Alex M Lamb, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4601–4609. Curran Associates, Inc., 2016.

Anirudh Goyal ALIAS PARTH Goyal, Alessandro Sordoni, Marc-Alexandre Côté, Nan Ke, and Yoshua Bengio. Z-forcing: Training stochastic recurrent networks. In *Advances in Neural Information Processing Systems*, pp. 6713–6723, 2017.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, pp. 3, 2017.

Łukasz Kaiser, Aurko Roy, Ashish Vaswani, Niki Pamar, Samy Bengio, Jakob Uszkoreit, and Noam Shazeer. Fast decoding in sequence models using discrete latent variables. *arXiv preprint arXiv:1803.03382*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pp. 3738–3746, 2016.

Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, pp. 125, 2016.

Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016.

Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1747–1756. JMLR.org, 2016.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

6 APPENDIX

6.1 TRAINING DETAILS

In our experiments we keep the filter size and number of filters fixed for every layer. For Iamondb, Deepwriting and TIMIT datasets, the filter size is set to 2 unless it is 1×1 convolution. The number of filters is 256 and similarly equal at every layer. The output layer consists of 2 1×1 convolution with $ReLU$ activation function after each of them. Finally, the parameters of output distribution are estimated by another 1×1 convolution.

The number of Wavenet blocks is 6 for Iamondb and Deepwriting. We stack 5 of them. In other words, there are 5 stochastic layers. In computations of stochastic layer parameters, we again use 2 1×1 convolutions with $ReLU$ activation function.

We use ADAM optimizer with and initial learning rate of 5^{-4}. The learning rate is annealed by using exponential decay with rate of 0.94 and 1000 decay steps. We apply early stopping on the validation splits.

The code will be made publicly available.