Learning deep autoregressive models for hierarchical data

Carl R. Andersson * Niklas Wahlström * Thomas B. Schön *

* Department of information technology, Uppsala University, Sweden, (email: carl.andersson;niklas.wahlstrom;thomas.schon@it.uu.se)

Abstract: We propose a model for hierarchical structured data as an extension to the stochastic temporal convolutional network. The proposed model combines an autoregressive model with a hierarchical variational autoencoder and downsampling to achieve superior computational complexity. We evaluate the proposed model on two different types of sequential data: speech and handwritten text. The results are promising with the proposed model achieving state-of-the-art performance.

Copyright © 2021 The Authors. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0)

Keywords: Deep learning, variational autoencoders, nonlinear systems

1. INTRODUCTION

System identification and sequence modeling with deep learning are two different research areas that essentially solve the same problem, to model a sequence, \( y_{1:T} \), for predictive purposes. Even though the methods are very similar there are differences when it comes to what kind of data the method typically is applied to. Whereas system identification traditionally has focused on systems with relatively short memory and small datasets (e.g. systems on the nonlinear benchmark website (Schoukens, 2020)), deep learning has focused on the opposite, i.e. systems with long memory and larger datasets (e.g. text modeling). This dichotomy is not a product of any fundamental difference in methodology, but rather an effect of the rapid advancement in computational power that has accompanied the advancements in deep learning.

Neural networks are by no means new to the system identification community, they have in fact a rather rich history (Sjöberg et al., 1995). What deep learning brought with it was instead a new paradigm of model design and implicit regularization. Whereas the system identification community typically view neural networks as black box function approximators, the deep learning community instead augment how these black box models are built up and adapt them to the data, e.g. long-short term memory (LSTM) models (Hochreiter and Urgen Schmidhuber, 1997), convolutional neural networks (CNN) (Krizhevsky et al., 2012) and temporal convolutional networks (TCN). A very prominent example of the connection between model and data structure is indeed deep CNNs applied to images, where the model mirrors the hierarchical composition of natural images and locality of low-level features (LeCun et al., 2015) while high-level features are represented in a downsampled image. Such hierarchical composition can also be seen in many sequential datasets, e.g. language modeling, handwritten text modeling (see Figure 1a), human motion tracking or other systems of (hierarchical) switching nature where the high-level features evolve at a slower pace than low-level features (Chung et al., 2016; Koutník et al., 2014). In the light of this we propose a sequential hierarchical latent variable model, represented in Figure 2a, that focus on modeling multiscale hierarchically structured sequential data. This can seen as an extension the Stochastic TCN (Aksan and Hilliges, 2019) which also uses a sequential hierarchical latent variable model although without the multiscale property. The latent variables \( z = \{ z^{(l)}_{1:T} \} \) are structured sequentially with hierarchical layers, where layer \( l \) is downsampled with a factor \( S^{(l)} \). To facilitate training we propose to relax the state space formulation similarly to Aksan and Hilliges (2019), see Figure 2b. With this we get a deep sequential model that exploits the hierarchical nature of the data similarly to how a deep convolutional network exploits the nature of images. We realize the model using a combination of a Wavenet model (Van Den Oord et al., 2016) for the sequential component with a hierarchical variational autoencoder (Rezende et al., 2014; Kingma and Welling, 2014) for the latent variables. We employ the proposed model on two different datasets and show that by...
The temporal Convolutional Network (TCN) is a type of non-linear autoregressive (NARX) model that makes use of a deep autoregressive neural network model. Bai et al. (2018) observe that these autoregressive models are easier to train, less sensitive to hyperparameter selection and in many circumstances achieves superior results, compared to deterministic state space models, i.e. LSTMs and RNNs. This can partly be explained by the efficient implementation, both computationally and parameterwise, of the autoregressive neural networks here realized with convolutions.

The temporal Convolutional Network (TCN) is a type of non-linear autoregressive (NARX) model that makes use of a deep autoregressive neural network model. Bai et al. (2018) observe that these autoregressive models are easier to train, less sensitive to hyperparameter selection and in many circumstances achieves superior results, compared to deterministic state space models, i.e. LSTMs and RNNs. This can partly be explained by the efficient implementation, both computationally and parameterwise, of the autoregressive neural networks here realized with convolutions.

Fig. 2. Two versions of a hierarchical generative model for sequential data, \(y\), on state space form (top) and autoregressive form (bottom), with latent variables, \(z\). \(z^{(2)}\) updates only every other timestep and \(z^{(3)}\) updates only every forth.

including the prior information of the hierarchical nature of the data we can improve the model in terms of both parameter effectiveness and computational complexity.

2. BACKGROUND

In this section we will introduce and give some background on the components needed to realize the proposed model.

### 2.1 Temporal convolutional network

The temporal Convolutional Network (TCN) is a type of non-linear autoregressive (NARX) model that makes use of a deep autoregressive neural network model. Bai et al. (2018) observe that these autoregressive models are easier to train, less sensitive to hyperparameter selection and in many circumstances achieves superior results, compared to deterministic state space models, i.e. LSTMs and RNNs. This can partly be explained by the efficient implementation, both computationally and parameterwise, of the autoregressive neural networks here realized with convolutions.

The depth of the TCN is achieved by stacking multiple affine transformations intertwined with activation functions. Each affine transformation/activation function pair can be expressed with

\[ h_t = \phi(x_{t-k:t-1}), \tag{1} \]

where \(\phi\) combines an affine transformation and a nonlinear activation function of the regression vector, \(x_{t-k:t-1}\). The notation \(x_{i:j}\) denotes the sequence of \(x\) from \(i\) to \(j\). An alternative formulation of this is to view the entire \(h_{1:T}\) as a convolution between \(\phi\) and the input \(x_{1:T}\). The width of this convolution filter, \(k\), corresponds to the so called the receptive field of the convolution. We use the notation

\[ h_{1:T} = f(\text{Conv}(x_{1:T})), \tag{2} \]

where \(\text{Conv}\) stands for the affine transformation and \(f\) is an activation function.

The dilated convolution (Van Den Oord et al., 2016) is an option to the convolution that sets some of the parameters in the affine transformation to zero. With a dilation rate \(\mu\) only every \(\mu\)th element in the affine transformation is nonzero. By convention \(k\) denotes the number of effective parameters in this transformation. Thus, this increase the receptive field of the filter to \(k\mu\). By stacking several dilated convolutions with exponentially increasing dilation rate it is possible to produce a model with exponentially long memory (exponentially large respective field) with a linear increase in the number of parameters (Yu and Koltun, 2016). The parameters of the affine transformation are the parameters of the convolutional layer, while both \(k\) and \(\mu\) are hyperparameters.

### 2.2 Wavenet

One of the most prominent TCNs is Wavenet (Van Den Oord et al., 2016). Wavenet is built up of stacked so-called Wavenet blocks, where each block consists of a residual connection and an identity connection (inspired by ResNet (He et al., 2016)). Each residual connection in turn consists of a sequence of: a dilated convolution, a gate-like activation function, and a final activation function of the regression vector, \(\phi\). An \(1\times1\) convolution is a type of convolution that only operates locally (the filter has size 1). The Wavenet block can be summarized as

\[
\text{WavenetBlock}(x_{1:T}) = x_{1:T} + \text{Conv}_{1\times1}(\tanh(\text{Conv}(x_{1:T})) \odot \sigma(\text{Conv}(x_{1:T}))), \tag{3}
\]

where \(\odot\) denotes element-wise multiplication and \(\sigma\) denotes the sigmoid activation function.

When merging two data sequences with the Wavenet model Van Den Oord et al. (2016) proposed to consider one of the sequences as the main sequence \((x_{1:T})\) and the other as a conditional sequence \((c_{1:T})\). The two sequences are concatenated (in feature space) forming a new stream and this is used as input to the residual network. The identity connection only passes \(x_{1:T}\) forward. Thus, a conditional Wavenet block can be written as,

\[
\text{CondWavenetBlock}(x_{1:T}, c_{1:T}) = x_{1:T} + \text{Conv}_{1\times1}(\tanh(\text{Conv}([x_{1:T}, c_{1:T}])) \odot \sigma(\text{Conv}([x_{1:T}, c_{1:T}]))), \tag{4}
\]

where \([\cdot]\) denotes concatenation in the feature dimension. When stacking several such blocks together, all blocks share the same conditioning sequence. Lai et al. (2018) with Stochastic Wavenet and later Aksan and Hilliges (2019) extended these models to also include latent variables similar to the proposed model.

### 2.3 Variational autoencoder

A common approach for latent variables in deep learning is to use variational autoencoders (VAE) (Rezende et al.,...
2014; Kingma and Welling, 2014). This section aims to introduce the notation we use in this paper and for a more in depth and pedagogical description of the VAE see Kingma and Welling (2019). The idea of a VAE is to model the distribution of $y$ with a generative model as

$$p(y) = E\left[p(y | z)p(z)\right], \quad (5)$$

with

$$p(y | z) = \mathcal{N}(y | \mu_y(z), \Sigma_y(z)),$$

$$p(z) = \mathcal{N}(z | 0, I),$$

where $\mu_y(z)$ and $\Sigma_y(z)$ are modeled as neural networks.

In the VAE setting the expectation in (5) is estimated with Monte Carlo samples and (amortized) variational inference. The variational inference is done by introducing a parameterized approximate posterior distribution, $q$, 

$$q(z) = \mathcal{N}(z | \mu_q(y), \Sigma_q(y)),$$

where $\mu_q(y)$ and $\Sigma_q(y)$ are also modeled as neural networks. It is also possible to condition the VAE on some arbitrary data, $c$, in which case the prior is similarly expressed with neural networks,

$$p(z | c) = \mathcal{N}(z | \mu_p(c), \Sigma_p(c)). \quad (6)$$

The approximate posterior $q(z)$ and the generative distribuion will then also depend on $c$.

Using this approximate posterior and Jensen’s inequality, we can bound the log-likelihood from below. This gives us the evidence lower bound (ELBO)

$$\mathcal{L}_{\text{ELBO}}(y; \theta) = E_q[\log p(y | z)] - D_{KL}(q(z) || p(z)), \quad (7)$$

where $D_{KL}$ denotes the Kullback-Liebler divergence and $\theta$ are the parameters of both the approximative posterior and the generative model. The expectation here is evaluated with respect to $q$ and it is most commonly approximated with a single Monte Carlo sample. The ELBO objective is used to update both the parameters of the generative model and the approximate posterior.

The variational RNN (VRNN) (Chung et al., 2015), stochastic RNN (SRNN) (Fraccaro et al., 2016) and STORN (Bayer and Osendorfer, 2014) are all examples of VAEs that have been adapted to sequential data. The core of these models is an RNN which is extended with a latent variable. These models require a backward flow for inference which is implemented by an RNN running backwards in time.

### 2.4 Hierarchical variational autoencoders

A hierarchical variational autoencoder extends the VAE by introducing hierarchically stacked latent variables, $\{z^{(l)}\}^{L}_{l=1}$, in $L$ layers. The use of stacked latent variables was proposed along with the original variational autoencoder, though with small improvements compared to the single layer VAE. The hierarchical models were later improved by Sonderby et al. (2016) with the so called ladder VAE, where the approximate posterior is linked to the generative distribution (see Figure 3a). This idea has since then proved to be fertile ground for numerous other architectures, e.g. ResNet VAE (Kingma et al., 2016), BIVA (Maaløe et al., 2019) and NVAE (Vahdat and Kautz, 2020).

The approximate posterior’s dependence on $y$ is implemented by extracting a bottom-up hierarchy of features, $d^{(l)}$, where each successive layer of features depends on the previous layer, $d^{(l-1)}$. The link between the generative and the approximate posterior distribution has proven to be essential for efficient training of these models. Although, the implementation details for this link varies it can represented as Figure 3a in that the posterior also depends on the extracted features $h^{(l)}$ from the generative model. The parameters of the approximate posterior are optimized jointly with the parameters of the generative model.

### 3. MODEL

In our model we combine temporal convolutional networks and striding as building blocks to instantiate Figure 2b. We combine this with a hierarchical VAE where we condition the latent states on the previously observed data in an autoregressive fashion, as visualized in Figure 4a. Analogously to the hierarchical VAE, the model can roughly be split into a bottom-up (from data to more abstract features) and a top-down (from abstract features to predictions) network (Maaløe et al., 2019). The autoregressive features will also be used as features for the approximate posterior, inspired by the work of Aksan and Hilliges (2019) which this work extends.

The bottom-up network produces a set of features that is divided into hierarchical layers. This network also downsamples the signal in between every layer to capture the multiscale property of the model. To avoid excessive cluttering we use the notation $d^{(l)} = d^{(l)}_{1:S:0:T}$ to denote the bottom-up features which have been downsampled by a total factor of $S^{(l)}$. The downsample is implemented inside the first WavenetBlock function where the residual connection simply uses strided convolutions and the identity connection use the average. We use superscript D to denote Wavenet with downsampling.

$$d^{(1)} = \text{Wavenet}(y), \quad \text{(8a)}$$

$$d^{(1)} = \text{Wavenet}^{D}(d^{(1-1)}), \quad l \neq 1. \quad \text{(8b)}$$

The features of the top-down network, $h^{(l)} = h^{(l)}_{1:S:0:T}$, follows the hierarchical VAE with an additional conditioning and upsampling. The conditioning is performed by concatenating the output from the previous hierarchical layer, which we denote $g^{(l)} = g^{(l)}_{1:S:0:T}$, with the delayed features of the bottom-up network, followed by a $1 \times 1$ convolution

$$h^{(l)} = \text{Wavenet}_{1 \times 1}([g^{(l)}, \text{Delay}(d^{(l)})]), \quad l \neq L, \quad \text{(9a)}$$

$$h^{(L)} = \text{Wavenet}_{1 \times 1}([\text{Delay}(d^{(L)})]), \quad \text{(9b)}$$

where $[\cdot]$ denotes concatenation in feature domain. The Delay function delays $d^{(l)}$ one step at the current downsample level and prepends zeros, i.e.

$$\text{Delay}(d_{1:S:T}) = [0, d_{1:S:T} - s], \quad \text{(10)}$$

here the concatenation is done in time domain. Finally $\text{Wavenet}_{1 \times 1}$ denotes a Wavenet where all the dilated convolutions are replaced with $1 \times 1$ convolutions.

The prior distribution for the latent variable is given by

$$p(z^{(l)} | h^{(l)}) = \mathcal{N}\left(z^{(l)} | \mu_{1 \times 1}(h^{(l)}), \Sigma = \text{diag}(\text{softplus}(\text{Conv}_{1 \times 1}(h^{(l)}))^2)\right), \quad \text{(11)}$$
Finally, the output of the top-down layer is calculated as,

\[ g^{(0)} = \text{CondWavenet}_{1 \times 1}(h^{(1)}, z^{(1)}) \]  
\[ g^{(l-1)} = \text{CondWavenet}_{1 \times 1}^{U}(h^{(l)}, z^{(l)}), \quad l \neq 1, \quad (12a) \]

\[ g^{(l)} = \text{Conv}_{1 \times 1}(h^{(l)}, z^{(l)}), \quad l \neq 1. \quad (12b) \]

Thus, the latent variables are used as a conditioning sequence in the Wavenet. The Upsampling here, CondWavenet\(^U\), is similar to the downsample made as a part of the Wavenet. The identity connection is hear upsampled with nearest neighbor and the residual with strided transpose convolutions. The predictive distribution is chosen as \( p(y \mid \text{Conv}_{1 \times 1}(g^{(0)})) \). The shape of the predictive distribution is chosen with respect to the problem – Gaussian for real valued data and Bernoulli for binary data or mixtures thereof.

The approximate posterior is expressed using the previously defined features \( d^{(l)} \) and \( h^{(l)} \) as

\[ q(z^{(l)}) = \mathcal{N}(z^{(l)} \mid \mu = \text{Conv}_{1 \times 1}([h^{(l)}, d^{(l)}]), \Sigma = \text{diag}(\text{softplus}(\text{Conv}_{1 \times 1}([h^{(l)}, d^{(l)}])^2). \quad (13) \]

We do not calculate \( q \) as a combination of \( p \) and another normal distribution, as advocated by Sønderby et al. (2016) and Aksan and Hilliges (2019). Instead \( q \) is parameterized independently of the common features, similar to Kingma et al. (2016).

Figures 4a and 4b visualize the model in an alternative fashion compared to Figure 3b. Figure 4a resolves the temporal dimension and Figure 4b visualizes a hierarchical layer locally in time. The parameters of this model (both the generative model and the approximate posterior) are the parameters for the Wavenets, the Convolutional layers, the Upsample layers and the Downsample layers.

Although the stochastic TCN (STCN) (Aksan and Hilliges, 2019) is very related to our model, there are some distinctions. We generalize the STCN with the hierarchical VAE in favor of the Ladder VAE and also extend it by incorporating multiple timescales. Aksan and Hilliges (2019) argue that their model expresses multiple timescales but we disagree since it is missing the downsampling and upsampling steps. Indeed, the deeper features of the STCN gathers information from a larger receptive field. However, these features do only affect the current time step and are not shared between different timesteps.

4. EXPERIMENTS

We conduct experiments on two hierarchically structured sequential datasets, raw waveform speech (Blizzard) and handwritten text (IAM-OnDB). For all experiments we use linear free bits (see Appendix A) and the Adam optimizer with an exponentially decaying learning rate of \( 5 \cdot 10^{-4} \). For free bits we used a scheme to smoothly decay the threshold of free bits exponentially over the course of training at a rate of halving the threshold every 300000 iterations. We train for 100 epochs and do not see any tendencies of overfitting on the training data. Finally, each bottom-up Wavenet consists of four Wavenet blocks and each top-down consists of two Wavenet blocks. The number of Wavenet blocks are thus the same in the bottom-up and the top-down networks. For exact hyperparameter settings and additional implementation details we refer to the appendix and code.

2. https://github.com/carl-andersson/MS-STCN
also report the result with our model without the stochastic latent variables (i.e. only deterministic autoregressive). However, this control showed instabilities and eventually diverged so we report the best fit we got before it diverged. This divergence might be an indication that the latent variables are somehow regularizing the model. In Table 2 we also report the amount of information that is used in the KL-term for the different layers. Results show that KL units on all levels are being used.

**IAM-OnDB** (Liwicki and Bunke, 2005) is a handwritten text dataset where every sample consists of the pen position (R²) and a binary variable indicating whether the pen was lifted after this position or not. We use the same preprocessing as Chung et al. (2015) and Aksan and Hilliges (2019). However, we noted that the discretization of the pen position in some cases led to models that greatly overestimated the likelihood fit to the data. To cope with these discretization artifacts we added artificial uniform measurement noise to the pen position.

Samples from the model trained on IAM-OnDB (Figure 1) shows complex sequences were it is even possible to recognize a few words. However, the added noise means that the likelihood estimates of the model are not easily comparable with previous results on this dataset.

### 5. CONCLUSION AND FUTURE WORK

In this paper we argue that a deep learning model is not that different from a structured autoregressive model. However, the philosophy behind the model and the problems it is applied to, are significantly different from a typical system identification model. The model we proposed has shown to model hierarchically structured data very well while at the same time being more parameter efficient compared to existing models. We hypothesize that the proposed model exploit the hierarchical nature of the sequential data similar to how the CNN exploits the hierarchical structure in image data.

The proposed model is in principle not limited to one-dimensional temporal data. Similar to PixelVAE (Gujrathi et al., 2017), our model can be applied to images or even video. Investigating whether the promising results of this model transfers into other regimes of hierarchically
structured data is an interesting avenue for future work. Another area for further experiments is to apply the model to other hierarchically structured data discussed in the introduction, one such example is pose and orientation estimation from inertial sensors (Kok et al., 2017).

Table 2. Average KL-divergence per layer on the Blizzard dataset where KL1 corresponds to the layer closest from the data.

<table>
<thead>
<tr>
<th>Model</th>
<th>KL1</th>
<th>KL2</th>
<th>KL3</th>
<th>KL4</th>
<th>KL5</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW=200</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>75</td>
</tr>
<tr>
<td>FW=200 w/o multiscale</td>
<td>49</td>
<td>34</td>
<td>48</td>
<td>146</td>
<td>143</td>
</tr>
<tr>
<td>FW=25</td>
<td>235</td>
<td>44</td>
<td>44</td>
<td>67</td>
<td>102</td>
</tr>
<tr>
<td>FW=25 w/o multiscale</td>
<td>321</td>
<td>127</td>
<td>488</td>
<td>543</td>
<td>180</td>
</tr>
<tr>
<td>FW=2</td>
<td>629</td>
<td>49</td>
<td>45</td>
<td>35</td>
<td>58</td>
</tr>
<tr>
<td>FW=2 w/o multiscale</td>
<td>204</td>
<td>210</td>
<td>158</td>
<td>209</td>
<td>50</td>
</tr>
</tbody>
</table>

ACKNOWLEDGMENT

We thank Joakim Jaldén’s group at KTH, Royal Institute of Technology, Stockholm, for lending computational power.

REFERENCES


