Time-varying Normalizing Flow for Generative Modeling of Dynamical Signals

Anubhab Ghosh*, Aleix Espuña Fontcuberta*, Mohamed R.-H. Abdalmoaty†, Saikat Chatterjee*
* Digital Futures, and School of Electrical Eng. and Comp. Science, KTH Royal Institute of Technology
{anubhabg, aleixef, sach}@kth.se
†Department of Information Technology, Division of Systems and Control, Uppsala University
mohamed.abdalmoaty@it.uu.se

***Abstract*—We develop a time-varying normalizing flow (TVNF) for explicit generative modeling of dynamical signals. Being explicit, it can generate samples of dynamical signals, and compute the likelihood of a (given) dynamical signal sample. In the proposed model, signal flow in the layers of the normalizing flow is a function of time, which is realized using an encoded representation that is the output of a recurrent neural network (RNN). Given a set of dynamical signals, the parameters of TVNF are learned according to maximum-likelihood approach in conjunction with gradient descent (backpropagation). Use of the proposed model is illustrated for a toy application scenario - maximum-likelihood based speech-phone classification task.**

***Index Terms*—Generative learning, recurrent neural networks, neural networks, normalizing flows**

I. Introduction

Generative models and their deep variants can be categorized in two main classes: implicit and explicit [1]. An explicit generative model has an analytical functional form, enables tractable computation of likelihood of a sample and can generate samples from the underlying probability distribution (also called target distribution). On the other hand, an implicit distribution allows sample generation, but computation of likelihood is not easy. A prominent example of implicit generative model is generative adversarial networks (GANs) [2]. GAN is based on deep neural networks (DNN). On the other hand, examples of explicit distributions are Gaussian models, Gaussian mixture models (GMM), DNN based normalizing flows (NF) [3], [4], etc. Due to use of DNN, NF and its variants are capable of modeling complicated probability distributions [5]. The above mentioned examples of explicit and implicit models are used to learn distributions of static signals.

In this article, we consider explicit generative modeling of a dynamical signal - a time-series signal with time-varying statistics. The probability distribution of the dynamical signal is assumed to be continuous, unknown and need to be learned from real data. Recent works considered modeling of a dynamical signal using a time-varying GMM whose parameters are time-varying; the parameters are provided by a recurrent neural network (RNN) [6], [7]. Our ansatz is that use of Gaussians in GMM has a limited capability to model an arbitrary distribution. This motivates us to explore the use of NF in lieu of GMM, and develop a powerful explicit generative model in conjunction with RNN. The development leads to time-varying NF (TVNF). Due to the use of NF, the proposed TVNF is expected to model arbitrarily complicated distributions better than time-varying GMM (TVGMM).

Our main contribution in this article is the development of TVNF, where we engineer a time-varying signal flow inside the layers of NF. The time-varying signal flow is realized using an encoding obtained from past samples of the dynamical signal. This encoding serves as a side-information and is computed recursively using an RNN. We train TVNF using the principle of maximum-likelihood (ML) approach, where the parameters are learned using gradient descent (backpropagation). We demostrate the performance of TVNF using a toy example - a speech phone classification task.

Finally we mention that there is a key difference between the TVGMM of [6], [7] and the proposed TVNF. Unlike those works where GMM parameters are time-varying, we do not vary NF parameters over time. Instead, we incorporate time-varying transformations in the signal flow of an NF. These transformations consist of scaling and translation operations in NF that are parameterized using an RNN.

A. Relevant literature

The authors in [8] use an NF with continuous-time transformations for modeling dynamical signals. They use a dynamical model for continuous latent variables. We do not use such a scheme in TVNF. For the case of discrete latent variables, NFs and their mixture models have been used together with hidden Markov models (HMMs) in [9], [10]. In case of a setup where observations are indirectly measured, traditional methods utilising Kalman filters have been used for modeling tasks involving continuous latent variables [11]. RNNs have been used quite extensively in modeling sequential data for several classification and regression tasks [12]–[14]. They have been also used in conjunction with Kalman filters to yield deep Kalman filters, shown in [15].

II. Time-varying Normalizing Flow

In this section, we describe our proposed TVNF model, which is built on two main ideas: first, modeling the unknown, conditional distribution $p(\mathbf{x}_t|\mathbf{x}_1 \dots \mathbf{x}_{t-1})$ at time instant t using an explicit generative model $\mathcal{F}(\cdot)$. In this work, we use a normalizing flow as an explicit generative model. We denote the flow as a function $\mathcal{NF}(\cdot;\mathbf{\Psi})$ having a set of learnable parameters denoted by $\mathbf{\Psi}$. Next, using the information from

$t-1$ past data points, viz. $\mathbf{x}_1 \dots \mathbf{x}_{t-1}$ to incorporate time-varying transformations into $\mathcal{NF}$. This side information is encoded using an RNN $\boldsymbol{\Phi}(\cdot;\boldsymbol{\theta_\phi})$, with a set of learnable parameters denoted by $\boldsymbol{\theta_\phi}$. For a given sample sequence $\mathbf{x}_{1:T} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T]$, we wish to compute the joint distribution in terms of conditionals as follows:

$$p(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{x}_t | \mathbf{x}_1 \dots \mathbf{x}_{t-1}). \tag{1}$$

Here we model $p(\mathbf{x}_t|\mathbf{x}_1 \dots \mathbf{x}_{t-1})$ using a combination of $\mathcal{NF}(\cdot)$ and RNN $\boldsymbol{\Phi}(\cdot;\boldsymbol{\theta_\phi})$. If $\tilde{\mathbf{x}}_t \sim p(\mathbf{x}_t|\mathbf{x}_1 \dots \mathbf{x}_{t-1})$, and

$$\mathbf{x}_t = \mathcal{NF}\left(\mathbf{z}_t; \boldsymbol{\Phi}(\mathbf{x}_{1:t-1};\boldsymbol{\theta_\phi}), \boldsymbol{\Psi}\right), \text{s.t. } \mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{2}$$

then $\mathbf{x}_t \overset{d}{=} \tilde{\mathbf{x}}_t$, i.e. the two variables $\mathbf{x}_t$ and $\tilde{\mathbf{x}}_t$ are similarly distributed. The function $\mathcal{NF}(\cdot)$ uses the value at the present time instant $\mathbf{x}_t$ and the output of the encoding RNN from the past time points $\mathbf{x}_{1:t-1}$ to model the conditional distribution. Note that $\mathbf{z}_t$ represents the input Gaussian noise to generate $\mathbf{x}_t$ using $\mathcal{NF}(\cdot)$. A schematic representation of our proposed architecture is shown in Fig. 1. Before delving into describing TVNFs in further detail, we provide a brief overview as a background of normalizing flows (NF) and some details regarding the choice of our encoding RNNs.

A. Brief overview of normalizing flows

A normalizing flow is a DNN-based model and defined by a bijective mapping $\mathbf{f}: \mathcal{X} \to \mathcal{Z}$ where $\mathcal{X}, \mathcal{Z} \in \mathbb{R}^D$ and denote the data space and latent space respectively. As a result of bijectivity, there also exists an inverse mapping $\mathbf{g}: \mathcal{Z} \to \mathcal{X}$, i.e. $\mathbf{g} = \mathbf{f}^{-1}$. For a data point $\mathbf{x} \in \mathcal{X}$, the value of $p(\mathbf{x})$ can be computed exactly using the change of variable formula:

$$\begin{aligned} p(\mathbf{x}) &= p(\mathbf{z}) \left| \det \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right| \\ &= p(\mathbf{f}(\mathbf{x})) \left| \det \left(\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|. \end{aligned} \tag{3}$$

In most cases, the base density (denoted by $p(\mathbf{z})$) is a known, tractable probability density function such as a multivariate Gaussian. We call a single transformation from $\mathcal{X}$ to $\mathcal{Z}$ (or vice-versa) a *flow layer*. It is a common practice to combine several such flow layers for modeling richer and more complex probability distributions [5]. This can be done as follows:

$$\mathbf{z} = \mathbf{h}^{[\mathbf{0}]} \underset{\mathbf{f}^{[1]}}{\overset{\mathbf{g}^{[1]}}{\rightleftharpoons}} \mathbf{h}^{[\mathbf{1}]} \underset{\mathbf{f}^{[2]}}{\overset{\mathbf{g}^{[2]}}{\rightleftharpoons}} \mathbf{h}^{[\mathbf{2}]} \underset{\mathbf{f}^{[3]}}{\overset{\mathbf{g}^{[3]}}{\rightleftharpoons}} \mathbf{h}^{[\mathbf{3}]} \dots \underset{\mathbf{f}^{[L]}}{\overset{\mathbf{g}^{[L]}}{\rightleftharpoons}} \mathbf{h}^{[\mathbf{L}]} = \mathbf{x}, \tag{4}$$

where each $\mathbf{h}^{[\mathbf{i}]} \in \mathbb{R}^D, i = \{0, 1, 2, \dots, L\}$, L denotes the number of flow-layers. Thus from (4), the function $\mathbf{g}$ defining the normalizing flow is given as:

$$\mathbf{g}(\cdot) = \mathbf{g}^{[L]} \circ \mathbf{g}^{[L-1]} \circ \mathbf{g}^{[L-2]} \circ \dots \circ \mathbf{g}^{[1]}. \tag{5}$$

If each individual transformation $\mathbf{g}^{[i]}$ is bijective, then the composition of bijective transformations results in $\mathbf{g}$ being bijective as well. Similar arguments hold for the function $\mathbf{f}$, since $\mathbf{f}^{[i]} = (\mathbf{g}^{[i]})^{-1}$. The Jacobian in (3) can also be computed inexpensively by appropriate choice of the functions $\mathbf{f}$ (or equivalently $\mathbf{g}$).

B. Implementation of encoding RNNs using GRUs

A crucial aspect of our TVNF is the side-information provided at each flow layer by the RNN $\boldsymbol{\Phi}(\cdot;\boldsymbol{\theta_\phi})$ as shown in Fig. 1. The encoded vector $\mathbf{e}_t$ at time instant t is obtained recursively by utilising the previous $\mathbf{e}_{t-1}$ and the current $\mathbf{x}_t$ as

$$\mathbf{e}_t = \boldsymbol{\Phi}(\mathbf{e}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta_\phi}). \tag{6}$$

A variety of RNNs known as gated recurrent units (GRUs) are designed to control information flow through the use of gates [16]. In subsection II-E, we show that the learning problem involves learning the parameters of the GRU as well as the normalizing flow.

C. Details of the proposed TVNF

We now proceed to describe our TVNF model in further detail. We use a variety of coupling flows called RealNVP [3] as our choice of $\mathcal{NF}$. We assume that our input here $\mathbf{x}_t$ is multidimensional. In case of the RealNVP flow, the input $\mathbf{x}_t$ is split into two halves $\mathbf{x}_{t,A}, \mathbf{x}_{t,B}$, i.e. if $\mathbf{x}_t = \begin{bmatrix} \mathbf{x}_{t,A} & \mathbf{x}_{t,B} \end{bmatrix}^\top \in \mathbb{R}^D$, then $\mathbf{x}_{t,A} \in \mathbb{R}^d, \mathbf{x}_{t,B} \in \mathbb{R}^{D-d}$, where $d > 0$ and usually $d = D/2$. In our proposed architecture, we wish to model the function $\mathbf{g}_t: \mathcal{Z} \to \mathcal{X}$ for a single flow-layer as

$$\begin{aligned} \mathbf{x}_{t,A} &= \mathbf{z}_{t,A} \\ \mathbf{x}_{t,B} &= \mathbf{z}_{t,B} \odot \exp(\mathbf{s}_t) + \boldsymbol{\mu}_t, \end{aligned} \tag{7}$$

where $\odot$ represents element-wise multiplication, the slope $\mathbf{s}_t$, and the intercept $\boldsymbol{\mu}_t$ are considered to be functions of $\mathbf{z}_{t,A}$ and an encoding of $\mathbf{x}_{1:t-1}$ denoted by $\mathbf{e}_t$ (6). Introducing the encoding ensures that the scaling and translation operations in (7) become time-varying. We use a neural network $NN(\cdot)$ to compute $\mathbf{s}_t, \boldsymbol{\mu}_t$ as

$$\mathbf{s}_t, \boldsymbol{\mu}_t = \text{NN}(\mathbf{z}_{t,A}, \mathbf{e}_t; \boldsymbol{\Psi}). \tag{8}$$

$\text{NN}(\cdot)$ can be a shallow feed forward neural network having parameters $\boldsymbol{\Psi}$. Similarly we can define the function $\mathbf{f}_t: \mathcal{X} \to \mathcal{Z}$ as:

$$\begin{aligned} \mathbf{z}_{t,A} &= \mathbf{x}_{t,A} \\ \mathbf{z}_{t,B} &= (\mathbf{x}_{t,B} - \boldsymbol{\mu}_t) \odot \exp(-\mathbf{s}_t). \end{aligned} \tag{9}$$

D. Defining the neural network $NN(\cdot)$

The backbone behind the functions defined in (7), (9) is the neural network $\text{NN}(\cdot)$ shown in Fig. 2. It is a function whose parameters $\boldsymbol{\Psi}$ constitute the set $\{\mathbf{W}_q, \mathbf{L}_q, \mathbf{V}_s, \mathbf{V}_\mu, \mathbf{c}_q, \mathbf{c}_s, \mathbf{c}_\mu\}$, with $\mathbf{W}_q \in \mathbb{R}^{l \times N}$, $\mathbf{L}_q \in \mathbb{R}^{l \times D}$, $\mathbf{V}_s, \mathbf{V}_\mu \in \mathbb{R}^{D \times l}$, $\mathbf{c}_q \in \mathbb{R}^l, \mathbf{c}_s \in \mathbb{R}^D, \mathbf{c}_\mu \in \mathbb{R}^D$. Here N, l denote the size of the encoding vector and number of hidden units respectively. The variable $\mathbf{q}_t \in \mathbb{R}^l$ is a non-linear transform of the sum of $\mathbf{W}_q \mathbf{e}_t$ and $\mathbf{L}_q(\mathbf{b} \odot \mathbf{x}_t)$, where $\mathbf{b}$ is an element-wise mask. Masking is done to ensure alternate dimensions get subjected to equal amount of transformations [3]. The slope $\mathbf{s}_t$ and the intercept $\boldsymbol{\mu}_t$ are calculated as:

$$\begin{aligned} \mathbf{q}_t &= \text{ReLU}(\mathbf{W}_q \mathbf{e}_t + \mathbf{L}_q(\mathbf{b} \odot \mathbf{x}_t) + \mathbf{c}_q), \\ \mathbf{s}_t &= \tanh(\mathbf{V}_s \mathbf{q}_t + \mathbf{c}_s), \\ \boldsymbol{\mu}_t &= \mathbf{V}_\mu \mathbf{q}_t + \mathbf{c}_\mu. \end{aligned} \tag{10}$$

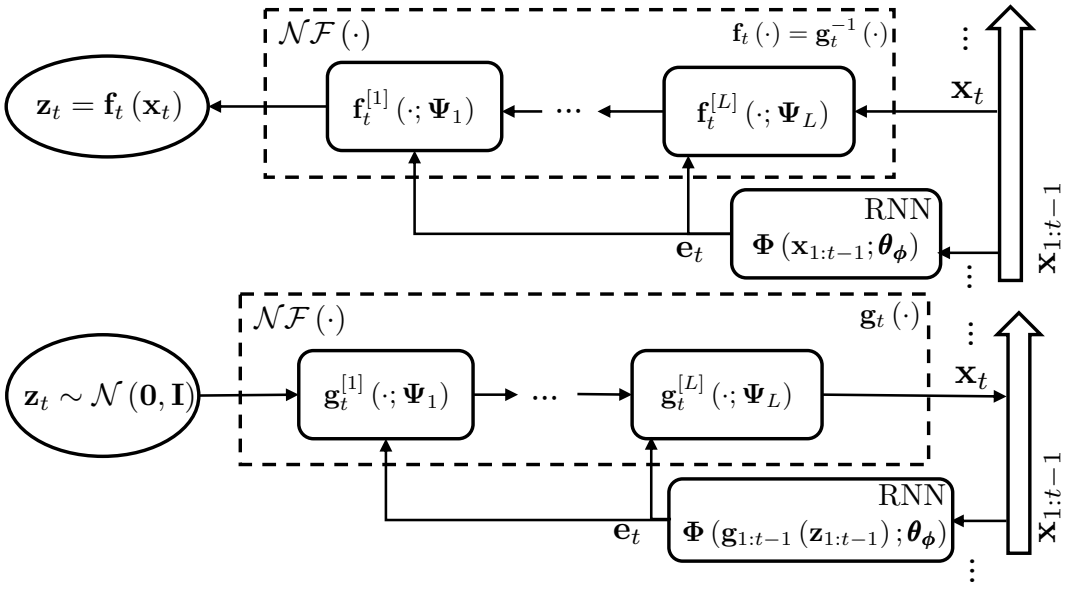

Fig. 1. Schematic representation of a TVNF. The figure shows the transformations for likelihood computation, denoted by $\mathbf{f}_t(\cdot)$ (top) and for generating samples, denoted by $\mathbf{g}_t(\cdot)$ (bottom). $\mathcal{NF}$ is composed of L individual transformations / flow layers. Every l^{th} transformation is parameterized by $\boldsymbol{\Psi}_l$ and receives the encoding of past samples $\mathbf{e}_t$ as a side-information from the RNN $\boldsymbol{\Phi}$.

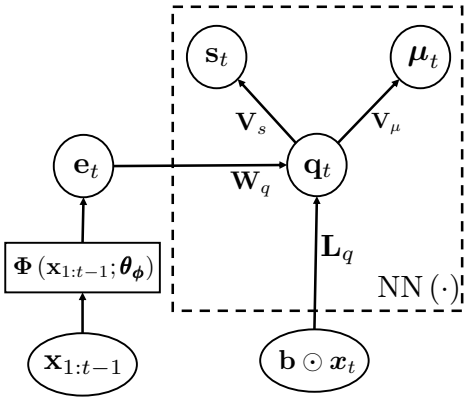

Fig. 2. Computational graph of the neural network NN$(\cdot)$ (shown in dotted lines) within the mapping $\mathbf{f}_t : \mathcal{X} \to \mathcal{Z}$.

We use a $\tanh$ activation for the slope as motivated in [3]. There is also a possibility to add hidden layers on top of $\mathbf{q}_t$ and we show results concerning the same in subsection III-C.

E. Training of TVNF

Let us suppose we have N_b number of i.i.d. samples in a given batch of training data, where each sample is a sequence of data points. This can be shown as $\mathbf{X}^{(N_b)} = \left[\mathbf{x}_{1:T_1}^{(1)}, \ldots, \mathbf{x}_{1:T_{N_b}}^{(N_b)}\right]$, where $T_i, (i = 1, \ldots, N_b)$ can be different. The parameters of the proposed TVNF include $\{\boldsymbol{\Psi}, \boldsymbol{\theta_\phi}\}$, and are learned by maximizing the averaged, joint log-likelihood computed using all the samples in the batch. This is shown in (11), and the optimization is practically achieved through mini-batch gradient descent and backpropagation.

$$
\begin{aligned}
\boldsymbol{\Psi}^\star, \boldsymbol{\theta_\phi}^\star &= \arg\max_{\boldsymbol{\Psi},\boldsymbol{\theta_\phi}} \frac{1}{N_b} \log\left(p\left(\mathbf{X}^{(N_b)}; \boldsymbol{\Psi}, \boldsymbol{\theta_\phi}\right)\right) \\
&= \arg\max_{\boldsymbol{\Psi},\boldsymbol{\theta_\phi}} \frac{1}{N_b} \log\left(\prod_{m=1}^{N_b} p\left(\mathbf{x}_{1:T_m}^{(m)}; \boldsymbol{\Psi}, \boldsymbol{\theta_\phi}\right)\right) \\
&= \arg\max_{\boldsymbol{\Psi},\boldsymbol{\theta_\phi}} \frac{1}{N_b} \sum_{m=1}^{N_b} \log\left(p\left(\mathbf{x}_{1:T_m}^{(m)}; \boldsymbol{\Psi}, \boldsymbol{\theta_\phi}\right)\right) \\
&= \arg\max_{\boldsymbol{\Psi},\boldsymbol{\theta_\phi}} \frac{1}{N_b} \sum_{m=1}^{N_b} \sum_{t=1}^{T_m} \log\left(p\left(\mathbf{x}_t^{(m)} | \mathbf{e}_t^{(m)}; \boldsymbol{\Psi}\right)\right) \\
&= \arg\max_{\boldsymbol{\Psi},\boldsymbol{\theta_\phi}} \frac{1}{N_b} \sum_{m=1}^{N_b} \sum_{t=1}^{T_m} \log p\left(\mathbf{f}_t\left(\mathbf{x}_t^{(m)}; \mathbf{e}_t^{(m)}, \boldsymbol{\Psi}\right)\right) \\
&\qquad + \log\det\left|\left(\frac{\partial \mathbf{f}_t\left(\mathbf{x}_t^{(m)}; \mathbf{e}_t^{(m)}, \boldsymbol{\Psi}\right)}{\partial \mathbf{x}_t^{(m)}}\right)\right|,
\end{aligned}
\tag{11}
$$

where $\mathbf{e}_t$ is the encoding given by the RNN using (6) and the last step in (11) is obtained by applying the change of variable formula described in (3). The pseudocode for training TVNFs is shown in Algorithm 1.

F. In comparison to time-varying GMM

In order to compare the performance of TVNF, we consider two additional time-varying, explicit generative models, viz. time-varying GMM (TVGMM) and time-varying Gaussian (i.e. TVGMM with a single mixture component), which have been modeled in a similar manner as in [6], [7]. An equation for a multivariate GMM that models a conditional distribution

Algorithm 1: Learning algorithm for training TVNF

Input: Dataset $\mathbf{X} = \left[\mathbf{x}_{1:T_1}^{(1)}, \ldots, \mathbf{x}_{1:T_{N_s}}^{(N_s)}\right]$, initial model parameters $\boldsymbol{\Psi}, \boldsymbol{\theta_\phi}$, initial RNN encoding $\mathbf{e}_0 = \mathbf{0}$
Result: Optimized model parameters: $\boldsymbol{\Psi}^\star, \boldsymbol{\theta_\phi}^\star$
Set training parameters: learning rate (η), no. of samples in a mini-batch (N_b), max. number of training epochs (N_{max})
Initialization epoch counter $(n) \leftarrow 0$
while $(n \leq N_{max})$ **do**
  Sample a batch of training data $\mathbf{X}^{(N_b)}$
  Compute the averaged log-likelihood $\Lambda := \frac{1}{N_b} \log\left(p\left(\mathbf{X}^{(N_b)}; \boldsymbol{\Psi}, \boldsymbol{\theta_\phi}\right)\right)$ using (11)
  Compute gradients $\partial\boldsymbol{\Psi}, \partial\boldsymbol{\theta_\phi}$ by optimizing Λ (Mini-batch gradient descent using Adam [17])
  Update $\boldsymbol{\Psi} \leftarrow \boldsymbol{\Psi} + \eta \cdot \partial\boldsymbol{\Psi}$
  Update $\boldsymbol{\theta_\phi} \leftarrow \boldsymbol{\theta_\phi} + \eta \cdot \partial\boldsymbol{\theta_\phi}$
  $n \leftarrow n + 1$
end

of $\mathbf{x}_t \in \mathbb{R}^D$ given the previous observed frames / time-points $\mathbf{x}_{1:t-1}$ $(t = 2, 3, \ldots, T)$ would be:

$$p\left(\mathbf{x}_t | \mathbf{x}_{1:t-1}\right) = \sum_{m=1}^{M} w_{m,t} \mathcal{N}\left(\mathbf{x}_t; \boldsymbol{\mu}_{m,t}, \mathbf{C}_{m,t}\right), \tag{12}$$

where the parameters $\boldsymbol{\theta}_{\text{GMM}} = \{\mathbf{C}_{m,t}, \boldsymbol{\mu}_{m,t}, w_{m,t}\}_{m=1}^{M}$ are functions of the sequence encoding $\mathbf{e}_t \in \mathbb{R}^N$ given by an RNN $\boldsymbol{\Phi}\left(\mathbf{x}_{1:t-1}; \boldsymbol{\theta_\phi}\right)$. It is important to note that there are certain constraints on some of the parameters in $\boldsymbol{\theta}_{\text{GMM}}$, i.e. the weights should sum to one and covariance matrices should be symmetric and positive definite. In our case, we consider diagonal covariance matrices, where we first map $\mathbf{e}_t$ to $\mathbf{z}_t$ using a shallow, feed-forward network, and then we compute the model parameters as follows:

$$\begin{aligned} \{\boldsymbol{\mu}_{m,t}\}_{m=1}^{M} &= \mathbf{W}_{\mu z}\mathbf{z}_t + \mathbf{b}_\mu, \\ \{\text{diag } \mathbf{C}_{m,t}\}_{m=1}^{M} &= \text{sigmoid}\left(\mathbf{W}_{Cz}\mathbf{z}_t + \mathbf{b}_C\right), \\ \{w_{m,t}\}_{m=1}^{M} &= \text{softmax } \left(\mathbf{W}_{wz}\mathbf{z}_t + \mathbf{b}_w\right), \end{aligned} \tag{13}$$

where $\text{diag}(\mathbf{A})$ represents a vector of the diagonal elements of a matrix $\mathbf{A}$, $\text{sigmoid}(x) = 1/(1+e^{-x})$ and $\text{softmax}(x)_i = e^{-x_i}/(\sum_j e^{-x_j})$ are element-wise activation functions.

III. Experiments and Results

In this section, we describe an illustrative example where we apply our TVNF models to a 5-class speech-phone classification problem. Our aim is mainly to motivate the TVNF-based approach for modeling dynamical signals. We compare the performances with the two baseline models using TVGMM. The code used for reproducing the results can be found at: https://github.com/AleixEF/DynamicFlows.

A. Experimental setup

Human speech is a dynamically generated signal and a speech phone is one of the fundamental units of speech. In the context of linguistics, a speech phone is also referred to as a phoneme. The dataset used for this purpose was the TIMIT dataset [18], [19], which has utterances labelled at the phoneme level.

The dataset consists of 6300 phoneme-level speech utterances that were split into two sets - a training set consisting of 4620 utterances and a testing set consisting of 1680 utterances. Originally the TIMIT dataset consists of utterances labelled for 61 phonemes, which can be mapped to a smaller set of 39 phonemes [20]. We partitioned both the training set and the testing set into 39 classes for generative learning, and set aside 20 % of the class-wise training data set randomly as a validation data set for hyperparameter tuning.

As a pre-processing step, we used 13-dimensional Mel-frequency cepstral coefficients (MFCCs) computed frame-by-frame with a window length of 25 ms and a shift of 10 ms. We also computed dynamic features - delta (Δ) and double-delta $(\Delta\Delta)$ coefficients from the MFCCs and concatenate them together with the MFCC features along with an additional DC component. So, in total, we used a 40-dimensional feature vector as an input.

B. Model training

We trained models for each class using mini-batch gradient descent with a batch size of 128 samples. Each TVNF consisted of 4 flow-layers. The models were implemented using Python and PyTorch, and trained in parallel using two GPUs [21]. We used Adam as our optimizer [17], with a learning rate (η) 10^{-4}, which was decayed in a step-wise fashion by a factor of 0.9 after every 30% of the total number of training epochs. We also used early-stopping as a regularization strategy by monitoring the relative change of the averaged log-likelihood computed on the validation data set.

C. Results

The five classes consisted of MFCC features for the following phonemes: 'n', 'iy', 'hh', 'oy' and 'p' respectively. The performance metric was the accuracy computed on the held-out validation set for each class, which are then weighted and averaged to yield a single accuracy value. Accuracy was calculated as a ratio of the number of phonemes predicted correctly to the total number of phonemes present. We used a GRU model for the RNN where we tuned the hyperparameters, viz. size of the encoding vector (N), the size of the hidden unit in the feed-forward network (l), no. of hidden layers in the feed-forward network (n_H) and the number of flow layers (L). For the GMM models, we also tuned the number of mixture components (M). The results for the different models are shown in Tables I, II and III.

Using the chosen configurations as per the performance on the validation set (shown in respective tables in 'bold'), we also report the performance of the three models on the corresponding test datasets in Table IV. We can see that on this problem, the proposed TVNF performed better than the other two time-varying models on the test set. This agrees with our expectations, since normalizing flows are more powerful

TABLE I
PERFORMANCE ON THE VALIDATION SET FOR DIFFERENT CONFIGURATIONS OF A TVGMM ($M = 1$) ON THE 5-CLASS PROBLEM. ACCURACY VALUES HAVE A $\pm 2\%$ DEVIATION FROM THOSE DENOTED IN THE TABLE.

S.No.	N	l	Val. Acc. (in %)
1	30	8	78.47
2	30	15	80.37
3	**64**	**8**	**88.39**
4	64	15	73.50
5	80	8	84.41
6	80	15	78.31
7	128	8	83.69
8	128	15	75.75

TABLE II
PERFORMANCE ON THE VALIDATION SET FOR DIFFERENT CONFIGURATIONS OF TVGMM ($M > 1$) ON THE 5-CLASS PROBLEM. ACCURACY VALUES HAVE A $\pm 2\%$ DEVIATION FROM THOSE DENOTED IN THE TABLE.

S.No.	N	l	M	Val. Acc. (in %)
1	**64**	**8**	**3**	**88.86**
2	64	8	5	87.96
3	64	8	10	88.33
4	80	8	3	81.32
5	80	8	5	79.09
6	80	8	10	81.82

in generative modeling than GMMs. Among the time-varying Gaussian model and time-varying GMM, the performance improvement for GMM was marginal.

TABLE III
PERFORMANCE ON THE VALIDATION SET FOR DIFFERENT CONFIGURATIONS OF TVNF ON THE 5-CLASS PROBLEM. ACCURACY VALUES HAVE A $\pm 1\%$ DEVIATION FROM THOSE DENOTED IN THE TABLE.

S.No.	N	l	n_H	L	Val. Acc. (in %)
1	64	8	2	4	91.03
2	**64**	**30**	**2**	**4**	**92.16**
3	80	8	2	4	91.1
4	80	30	1	4	85.05
5	80	30	2	4	92.12

TABLE IV
PERFORMANCE ON THE TEST SET FOR THE THREE MODELS FOR THE 5-CLASS CLASSIFICATION PROBLEM. ACCURACY VALUES HAVE A $\pm 1\%$ DEVIATION FROM THOSE DENOTED IN THE TABLE.

TVGMM (M=1)	TVGMM (M=3)	TVNF
89.17	89.72	91.89

IV. CONCLUSION

In this work, we demonstrated the use of time-varying normalizing flows for modeling dynamical signals. We showed that they can be trained using backpropagation and are applicable to a maximum-likelihood based classification scenario, where they provide better performance than time-varying GMMs. Future work may include further extensive experimentation of the TVNF models on other datasets or perhaps incorporating NFs with continuous-time transformations.

ACKNOWLEDGMENT

The authors would like to thank KTH Digital Futures Center for support amid a pandemic.

REFERENCES

[1] I. Goodfellow, "NeurIPS 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in NeurIPS*, vol. 27, 2014.

[3] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using RealNVP," in *5th International Conference on Learning Representations (ICLR)*, 2017.

[4] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," *Journal of Machine Learning Research*, vol. 22, no. 57, pp. 1–64, 2021.

[5] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[6] E. McDermott, "A deep generative acoustic model for compositional automatic speech recognition," in *Proc. of NeurIPS Workshop: Interpretability and Robustness in Audio, Speech, and Language*, 2018.

[7] X. Wang, S. Takaki, and J. Yamagishi, "An autoregressive recurrent mixture density network for parametric speech synthesis," in *Proc. of ICASSP*. IEEE, 2017, pp. 4895–4899.

[8] R. Deng, B. Chang, M.A. Brubaker, G. Mori, and A. Lehrmann, "Modeling continuous stochastic processes with dynamic normalizing flows," *Advances in NeurIPS*, vol. 32, pp. 5320–5330, 2019.

[9] A. Ghosh, A. Honoré, D. Liu, G. E. Henter, and S. Chatterjee, "Robust classification using hidden Markov models and mixtures of normalizing flows," in *Proc. of IEEE MLSP*. IEEE, 2020, pp. 1–6.

[10] A. Ghosh, A. Honoré, D. Liu, G.E. Henter, and S. Chatterjee, "Normalizing flow based hidden Markov models for classification of speech phones with explainability," *arXiv preprint arXiv:2107.00730*, 2021.

[11] S. Roweis and Z. Ghahramani, "Learning nonlinear dynamical systems using the expectation-maximization algorithm," *Kalman filtering and neural networks*, vol. 6, pp. 175–220, 2001.

[12] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. of ICASSP*. IEEE, 2013, pp. 6645–6649.

[13] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," *Advances in NeurIPS*, vol. 26, pp. 190–198, 2013.

[14] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," in *Proc. of ICASSP*. IEEE, 2015, pp. 4520–4524.

[15] R.G. Krishnan, U. Shalit, and D. Sontag, "Deep Kalman filters," in *NeurIPS 2016 Workshop: Advances in Approximate Bayesian Inference*, 2016.

[16] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proc. of* 8^{th} *workshop, SSST-8*, 2014, pp. 103–111.

[17] D.P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

[18] C. Lopes and F. Perdigao, "Phone recognition on the TIMIT database," *Speech Technologies/Book*, vol. 1, pp. 285–302, 2011.

[19] V. Zue, S. Seneff, and J. Glass, "Speech database development at MIT: TIMIT and beyond," *Speech communication*, vol. 9, no. 4, pp. 351–356, 1990.

[20] K-F Lee and H-W Hon, "Speaker-independent phone recognition using hidden Markov models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 11, pp. 1641–1648, 1989.

[21] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," *Advances in NeurIPS*, vol. 32, pp. 8026–8037, 2019.