A New Modeling Framework for Continuous, Sequential Domains

Hailiang Dong, James Amato, Vibhav Gogate, Nicholas Ruozzi
The University of Texas at Dallas, Richardson, TX, 75080, USA
{ *hailiang.dong, james.amato, vibhav.gogate, nicholas.ruozzi* }*@utdallas.edu*

Abstract

Temporal models such as Dynamic Bayesian Networks (DBNs) and Hidden Markov Models (HMMs) have been widely used to model time-dependent sequential data. Typically, these approaches limit focus to discrete domains, employ first-order Markov and stationary assumptions, and limit representational power so that efficient (approximate) inference procedures can be applied. We propose a novel temporal model for continuous domains, where the transition distribution is *conditionally tractable*: it is modelled as a tractable continuous density over the variables at the current time slice only, while the parameters are controlled using a Recurrent Neural Network (RNN) that takes *all* previous observations as input. We show that, in this model, various inference tasks can be efficiently implemented using forward filtering with simple gradient ascent. Our experimental results on two different tasks over several real-world sequential datasets demonstrate the superior performance of our model against existing competitors.

1 INTRODUCTION

Temporal/sequential data modeling problems arise naturally in a variety of applications from natural language processing to time series prediction tasks in science and engineering. In these problem domains, a natural task is to predict the future, i.e., predict $\boldsymbol{X}^{T+1}$ given a sequence of observations up to the present time, i.e., $\{\boldsymbol{X}^t\}_{t=1}^T$.

A wide variety of modeling approaches have been proposed for this task ranging from classical approaches such as Markov models, e.g., Markov chains, HMMs (Rabiner and Juang, 1986), dynamic Bayesian networks (DBNs) (Murphy, 2002), and variants thereof (Touloupou et al., 2020; Asghari et al., 2020; Li et al., 2019; Grzegorczyk, 2010; Nasfi et al., 2020), to more modern approaches based on recurrent neural networks (RNNs) (Gregor et al., 2015), e.g., long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRUs) (Bahdanau et al., 2014).

In Markov models, in order to make inference and learning tractable, the next observation is often assumed to be dependent on only a small preceding window of time, e.g., $\boldsymbol{X}^{T+1}$ is independent of $\{\boldsymbol{X}^t\}_{t=1}^{T-1}$ given $\boldsymbol{X}^T$. This limits the ability of these models to capture long-range dependencies. Further, much of the work on Markov models focus on the case of discrete random variables. When modeling continuous variables is required, the resulting models are often either limited to simple continuous exponential families for which inference can be done in closed form, e.g., multivariate Gaussians, or variational methods (Ranganath et al., 2014) are used to approximate intractable integrals, e.g., expectation propagation (Minka, 2013) in DBNs.

On the other hand, RNNs Hochreiter and Schmidhuber (1997); Bahdanau et al. (2014) attempt to overcome the finite time window assumption common in Markov models by using internal state, potentially allowing them to handle both long-term and short-term data dependencies. However, RNNs are purely discriminative, i.e., they do not yield a model of the joint distribution over the entire sequence. Therefore, they cannot jointly consider predictions over multiple time-steps. In particular, RNNs have limited ability to handle more sophisticated prediction tasks in which the data in each time slice may only be partially given, e.g., predict the future given only partial observations of the present.

Our aim in this work is to build a general modeling framework for continuous, temporal domains that combines RNNs and Markov models in an effort to overcome their complementary limitations. Specifically, we make the following contributions.

1. We propose a novel probabilistic representation in continuous, temporal domains that overcomes the finite time window assumption. The transition distribution we produce is conditionally tractable with parameters

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206. Copyright 2023 by the author(s).

controlled using RNNs.

2. We explain how maximum a posteriori (MAP) inference can be performed efficiently in our model, which allows our model to solve MAP prediction tasks for queries involving variables across potentially all of the time slices at the same time. This overcomes a limitation of pure RNN solutions.

3. We evaluate the the performance of our model against two competitive baseline models on a diverse collection of real-world sequential datasets under four different metrics through two different prediction tasks: trajectory prediction and sequence completion. Our experimental results demonstrate the superior discriminative performance of our model.

2 RELATED WORK

The field of tractable probabilistic models (TPMs) and probabilistic circuits (PCs) (Rahman et al., 2014; Liang et al., 2017; Lowd and Domingos, 2008; Choi et al., 2020) seeks to learn probabilistic models that admit polynomial time exact inference algorithms for various reasoning tasks such as computing the posterior marginal distribution over a subset of variables given observations and finding the most likely assignment to non-evidence variables. Recently, there has been a growing interest in designing tractable models more generally, and tractable temporal models specifically. Murphy (2002) presents a technique called the *interface* method for converting discrete dynamic Bayesian networks (DBNs) to junction trees, and showed that when the number of nodes in the interface between two time adjacent slices is bounded by a constant, inference in DBNs is tractable. Melibari et al. (2016) extended sum-product networks to temporal domains using a template network that is unrolled just like DBNs. Roy et al. (2021) proposed dynamic cutset networks that use AND/OR conditional cutset networks to model the transition distribution such that the joint model stays tractable. Most of these works focus on discrete domains. In addition, as discussed above, first-order Markov and stationary assumptions are made in these models, which limits their representational power and reduces the model's expressivity.

Several studies have explored the use of neural networks to summarize historical information or generate parameters for other models. For example, Shao et al. (2020), Thoma et al. (2021), and Dong et al. (2022) use neural networks to generate parameters. However, when applied to the temporal domain, these models still require the use of the first-order Markov assumption, which limits their ability to express complex relationships. An alternative approach is to use recurrent neural networks as an aggregation function in Graph Neural Networks Zhou et al. (2020) to summarize historical information, and then use the latter to model sequential data. However, these models are discriminative and have the same limitations as RNNs. Liu et al. (2020) show that Gaussian processes (GPs) provide another option for modeling sequential data. However, GPs only model the uncertainty over the output, and most existing works focus on a single or a small number of response variables.

Our work differs from these previous works in several ways. First, we propose a new probabilistic framework for representing and reasoning about uncertainty in continuous sequential domains. Second, we explore different tractable densities and neural network architectures for generating the parameters of the underlying probabilistic model. Finally, our model is generative and is designed to handle long multivariate sequences with a large number of variables. In particular, it can capture both long-term and short-term dependencies by harnessing the power of RNNs, without requiring the first-order Markov and stationary assumptions. We believe that generative models are more promising for complex sequential prediction tasks since they can infer the missing information jointly using both historical and future observations.

3 OUR MODEL

In this section, we introduce our proposed dynamic model and its architecture. In what follows, we will use bold uppercase letters to denote a set of random variables, e.g., $\boldsymbol{X}$, while a single random variable is denoted using uppercase letters, e.g., A. Instantiations (configurations) of random variables are denoted by lowercase letters. For example, $\boldsymbol{x}$ is a possible configuration of all variables in $\boldsymbol{X}$ and a is a possible value that the random variable A can take. For random variables that evolve over time, we use $\{\boldsymbol{X}^t\}_{t=1}^T$ to denote the sequence of (sets of) random variables generated by evolving $\boldsymbol{X}$ from time step 1 to time step T. All random variables considered in this paper are assumed to be defined over the real domain $\mathbb{R}$ unless otherwise noted.

3.1 Model Architecture

Given a sequence of random variables $\{\boldsymbol{X}^t\}_{t=1}^T$, their joint distribution $p(\{\boldsymbol{X}^t\}_{t=1}^T)$ can be factorized using the chain rule as follows.

$$p(\{\boldsymbol{X}^t\}_{t=1}^T) = p(\boldsymbol{X}^1)\prod_{t=2}^{T} p(\boldsymbol{X}^t|\boldsymbol{X}^1,\ldots,\boldsymbol{X}^{t-1})$$

As the transition distribution $p(\boldsymbol{X}^t|\boldsymbol{X}^1,\ldots,\boldsymbol{X}^{t-1})$ becomes increasingly difficult to model as the time $t \to \infty$, existing works often simplify the transition distribution as $p(\boldsymbol{X}^t|\boldsymbol{X}^{t-1})$ using first-order Markov and stationary assumptions. However, such assumptions can hurt the model's representational power especially when the sequential data has long-term dependencies. In this work, we directly model the full transition distribution $p(\boldsymbol{X}^t|\boldsymbol{X}^1,\ldots,\boldsymbol{X}^{t-1})$

by combining Recurrent Neural Networks (RNNs) with Probabilistic Graphical Models (PGMs).

Modern RNNs such as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (GRUs) (Bahdanau et al., 2014) maintain a hidden state $\boldsymbol{h}^t$ that preserves all important information up to time t. Specifically, $\boldsymbol{h}^t$ is computed based on the previous hidden state $\boldsymbol{h}^{t-1}$ and input $\boldsymbol{X}^{t-1}$, i.e., $\boldsymbol{h}^t = RNN(\boldsymbol{h}^{t-1}, \boldsymbol{X}^{t-1})$. As a result, $\boldsymbol{h}^t$ depends on all $\{\boldsymbol{X}^i\}_{i=1}^{t-1}$ as well as a special input $\boldsymbol{h}^1$, which is the initial hidden state [1].

Based on this observation, we model the full transition distribution as a continuous density over just the current time slice $\boldsymbol{X}^t$ with parameters controlled by a generating function g taking $\boldsymbol{h}^t$ as its input, i.e.,

$$p(\boldsymbol{X}^t|\boldsymbol{X}^1, \ldots, \boldsymbol{X}^{t-1}) = p(\boldsymbol{X}^t|\boldsymbol{\theta}^t = g(\boldsymbol{h}^t)).$$

Note that the choice of distribution p should be (nearly) tractable such that maximum-a-posteriori (MAP) and filtering inference can be performed in an efficient way. The generating function, g, can be learned from data using any regression method. In this work, we will use a neural network (NN) to learn g for the following reasons: (1) NNs are very expressive and (2) NNs can be easily stacked on top of RNNs, which allows the whole model to be trained in an end-to-end fashion.

Figure 1 shows the detailed architecture for generating the parameters $\boldsymbol{\theta}^t$. It is mainly composed of an RNN and an output NN block used to learn the generating function g. The RNN block takes all previously observed evidence $\{\boldsymbol{X}^i\}_{i=1}^{t-1}$ as input and repeatedly updates its hidden state, $\boldsymbol{h}^t$. As a result, $\boldsymbol{h}^t$ contains all information about previous time slices, and we can treat this as a feature vector that summarizes the sequence $\{\boldsymbol{X}^i\}_{i=1}^{t-1}$. Note that any existing RNN structure can be used here, e.g., LSTMs and GRUs. It is also possible to stack multiple layers of RNNs and to take advantage of future RNN models in our framework.

The output block in Figure 1 takes in the hidden state $\boldsymbol{h}^t$ from the RNN and treats it as a feature vector representing $\{\boldsymbol{X}^i\}_{i=1}^{t-1}$. Following Dong et al. (2022), we use multiple small headers and each of them is responsible for predicting only a subset (group) of parameters. Compared to using a single giant header (best efficiency) or one head for each parameter (most expressive), such a configuration strikes a better balance between representational power and efficiency. Finally, we simply concatenate all subsets of parameters and output the predicted $\boldsymbol{\theta}^t$. Note that the detailed configuration of those headers depends on the continuous density we pick as well as how we divide parameters into groups.

In this paper, we will consider two possible choices of continuous densities, Gaussian Bayesian networks and mixtures of independent Gaussians, but similar ideas can be

[1]The initialization is typically a zero vector; though it may vary depending on the deep learning framework used.

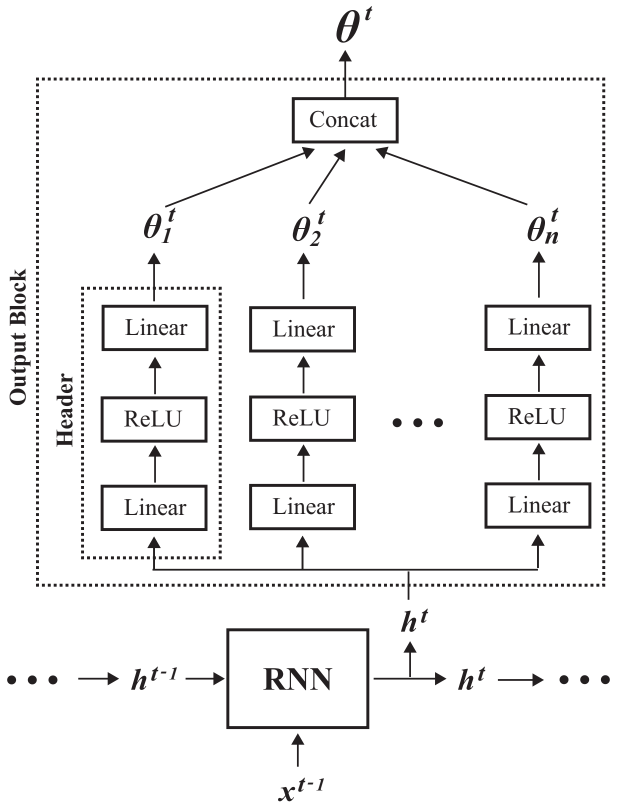

Figure 1: The neural architecture for generating the parameters $\boldsymbol{\theta}^t$ of the transition distribution $p(\boldsymbol{X}^t|\boldsymbol{\theta}^t)$.

applied into other densities as well. In particular, we consider Gaussian Bayesian Networks (GBNs) (Grzegorczyk, 2010) where the conditional distributions are modeled using conditional linear Gaussians (CLGs). This model is fully tractable even if the underlying graph structure is not a tree (Koller and Friedman, 2009). Each node (variable) x_i in a GBN has three types of parameters: (1) a standard deviation $\sigma_i > 0$, (2) a bias b_i, and (3) a weight vector $\boldsymbol{w}_i$. We first create two headers to predict all standard deviations $\{\sigma_i\}$ and biases $\{b_i\}$, respectively. Then, we create one header for each weight vector $\boldsymbol{w}_i$. Note that the total number of headers is linear in the number of variables $n = |\boldsymbol{X}|$, but in the most general case, the total number of parameters can grow quadratically in n. In order to avoid this potential quadratic dependence on n, we restrict the number of parents of each node to a constant k. In addition, for the header that predicts $\{\sigma_i\}$, we clip the output with a predefined threshold $t > 0$ to ensure all standard deviations are strictly positive (and thus we get a valid Gaussian density).

The second continuous density we consider is mixtures of independent multivariate Gaussians. This distribution is not fully tractable in general, but it admits efficient and almost exact inference in practice. Specifically, MAP inference can be done by conducting several iterations of gradient ascent starting from the mean of each component. In the case of marginal MAP inference, we just need to marginalize each component first and then conduct MAP inference on the mixtures of marginals. For a mixture with k components, the parameters we need to estimate are the mean vectors $\{\boldsymbol{u}_i\}$, a standard deviation vector $\{\boldsymbol{\sigma}_i\}$ for each of the components, and a vector of mixture weights $\boldsymbol{w}$ of size k, which controls the relative importance of each component. The configuration of headers in this case is quite straightforward. We create one header for each of the parameter vectors $\boldsymbol{\sigma}_i, \boldsymbol{\mu}_i, \boldsymbol{w}$. So, the total number of headers is $2k+1$. Similarly, we will use a predefined threshold to guarantee that $\boldsymbol{\sigma}$ and $\boldsymbol{w}$ are greater than zero. In addition, we normalize the mixture weight vector $\boldsymbol{w}$ such that its components sum to one.

3.2 Learning

Since the parameters of our model are controlled by the neural architecture shown in Figure 1, our objective is to learn such a neural network given N sequences $\{\{\boldsymbol{X}^t = \boldsymbol{x}^t\}_{t=1}^{T_i} | i = 1, 2.., N\}$ with different lengths, the i^{th} sequence has length T_i. Note that the prior distribution $p(\boldsymbol{X}^1)$ is represented as a special transition distribution $p(\boldsymbol{X}^1|\boldsymbol{X}^0)$, where $\boldsymbol{X}^0$ is the zero vector. In other words, the prior distribution is also learned from our neural architecture and is represented in the same way as the transition distributions. As we will see in later sections, this simplifies the inference procedure and makes the optimization process elegant and efficient.

We use the Adam optimizer from PyTorch Paszke et al. (2019) to train the whole neural network (the detailed configuration can be found in Section 4). For a given sequence $\{\boldsymbol{x}^t\}_{t=1}^{T}$, we feed $\{\boldsymbol{x}^t\}_{t=1}^{T-1}$ along with $\boldsymbol{x}^0$ to the neural network and use the output parameters $\{\boldsymbol{\theta}^t\}_{t=1}^{T}$ to evaluate the negative sequence loglikelihood of the sequence, i.e.,

$$L(\{\boldsymbol{x}^t\}_{t=1}^{T}) = -\sum_{t=1}^{T} \log p(\boldsymbol{x}^t|\boldsymbol{\theta}^t = g(\boldsymbol{h}^t)) \qquad (1)$$

It should be noted that the neural network is trained using mini-batch. We use the above equation to calculate the loss of each sequence in the batch and sum it together. After that, the parameters are updated by conducting back propagation through the entire network.

3.3 Inference

In this work, we evaluate the learned model's discriminative performance via a general MAP prediction task in sequential domains. Formally, we define a partially observed sequence as $\{\boldsymbol{X}^t\}_{t=1}^{T} = \{\boldsymbol{Y}^t, \boldsymbol{E}^t = \boldsymbol{e}^t\}_{t=1}^{T}$, where $\boldsymbol{E}^t$ are the observed/evidence variables with observed values $\boldsymbol{e}^t$. The $\boldsymbol{Y}^t$ are unknown and the inference task is to predict them based on the evidence, i.e., by maximizing the sequence likelihood. Note that the observed variables $\boldsymbol{E}^t$ may be different at each time slice, with possibly no observations for a given slice t, in which all variables $\boldsymbol{X}^t = \boldsymbol{Y}^t$ must be inferred.

Algorithm 1: Inference algorithm for MAP task

```
Input: (1) a partially observed sequence with $T$ time
  slices $\{X^t\}_{t=1}^T = \{Y^t, E^t = e^t\}_{t=1}^T$ ; (2) a trained
  RNN and generating function $g$; (3) max number of
  iteration allowed for gradient ascent $nIter$; (4)
  learning rate $lr$.
Output: the original sequence with $Y^t$ predicted, i.e.,
  $\{Y^t = y^t, E^t = e^t\}_{t=1}^T$.
$x^0 \leftarrow 0$ ;
$h^0 \leftarrow 0$ ;
$t \leftarrow 1$ ;
while $t \leq T$ do
    $h^t \leftarrow RNN(h^{t-1}, x^{t-1})$ ;
    $\theta^t \leftarrow g(h^t)$ ;
    Conduct MAP inference to get the best estimation
      $y^t \in \mathrm{argmax}_y\, p(y, e^t|\theta^t)$ ;
    $x^t \leftarrow (y^t, e^t)$ ;
    $t \leftarrow t + 1$ ;
end
Fix all parameters inside RNN and function $g$ ;
Set $\{y^t\}_{t=1}^T$ as the parameter to optimize ;
$i \leftarrow 1$ ;
while $i \leq nIter$ do
    Evaluate the negative loglikelihood loss $L$ using
      equation (1), respect to sequence $\{y^t, e^t\}_{t=1}^T$ ;
    Back propagate the loss $L$ and calculate the
      gradient $y_g^t$ respect to all $y^t$ ;
    Update $y^t \leftarrow y^t - lr \cdot y_g^t$ ;
    $i \leftarrow i + 1$ ;
end
Return the predicted sequence $\{y^t, e^t\}_{t=1}^T$ ;
```

This task contains the typical one-step future prediction as a special case, e.g., predict all of slice t given all of the preceding time slices, as well as longer time horizon predictions, e.g., predict the next k time steps of a sequence $\{\boldsymbol{x}^t\}_{t=1}^{T_i}$ (also known as the trajectory prediction problem). This is done by creating a partially observed sequence of length $T+k$ where all variables up to current time T are observed and all variables after T are unknown, i.e.,

$$\begin{cases} \boldsymbol{Y}^t = \varnothing, \boldsymbol{E}^t = \boldsymbol{x}^t, & \forall t \leq T \\ \boldsymbol{Y}^t = \boldsymbol{X}^t, \boldsymbol{E}^t = \varnothing, & \forall t > T \end{cases}.$$

For this task, we propose the two phase inference algorithm shown in Algorithm.1. In the first phase, we conduct local MAP inference step by step from the beginning to the final time slice. In each step, we predict the best estimation of $\boldsymbol{Y}^t = \boldsymbol{y}^t$ for the current time slice, and the predicted values $\boldsymbol{y}^t$ are used for subsequent MAP inference. All predictions generated in this fashion are locally optimal – only the

Table 1: The train and test size, number of variables, average length and type of all datasets (after pre-processing).

Name	#Train	#Test	#Var.	Len.	Type
japanvowels	270	370	12	16	Audio
arabicdigit	6599	2200	13	40	Speech
finger-movement	316	100	28	50	EEG
lsst	2384	2387	6	36	Other
face-detection	2933	996	144	62	EEG
natops	180	180	14	51	HAR
heartbeat	176	183	61	405	Audio
word-recognition	275	300	9	144	Motion
wing-beat	8122	511	199	8	Sensor
brown	6004	1502	100	22	NLP
conll2000	1598	400	100	16	NLP

evidence from previous time slices are used for each new prediction.

In the second phase, we conduct gradient ascent over the sequence loglikelihood to jointly optimize the predictions of all variables of interest $\{\boldsymbol{Y}^t\}_{t=1}^T$. This optimization process can be conducted in a very efficient and elegant way. Recall that the loss function for training the neural architecture was the negative sequence loglikelihood, which means we can treat the gradient ascent process as “training” the network using just one sequence. The only difference is that the parameters of the neural networks are now fixed, and we treat the input $\{\boldsymbol{Y}^t\}_{t=1}^T$ as the parameters in the optimization process. The whole inference procedure is efficient since (1) the continuous density is (nearly) tractable and (2) the gradient ascent process is equivalent to training the model using only *one* sequence.

4 EXPERIMENTS

In this section, we demonstrate the practical performance of our model against other existing competitors through three sets of experiments: trajectory prediction, sequence completion and comparison of generative performance.

We used nine real-word sequential datasets from the Time Series Classification Repository (William, 2021). We selected different types of datasets in order to ensure that we have diversity in terms of sequence length, number of sequences and number of features. In addition, we also considered two natural language processing (NLP) datasets: Brown Corpus (Francis and Kucera, 1979) and conll2000 (Sang and Buchholz, 2000). For these datasets, we treated each sentence in the corpus as a time series, and used a pre-trained Glove word embedding model (Pennington et al., 2014) to convert each word into a 100-dimensional vector.

We pre-processed all 11 datasets by first removing sequences whose lengths were less than seven. After that, following Uria et al. (2016), we removed one of the features from every pair of features whose Pearson correlation coefficient is greater than 0.98. All datasets were normalized by subtracting the mean and then dividing by the standard deviation. The detailed information of each sequential dataset after pre-processing is shown in Table. 1. Note that for the last two NLP datasets, the train/test split is not defined from the data source, and we randomly chose 80% of the sequences for the training split and the remaining sequences were used to form the test split. We additionaly set aside 20% of the training instances for validation purposes.

We considered two strong NN-based competitors along with two variants of our model in the experiments.

1. **NN-GBN**: a strong generative baseline model in which the transition distribution is modeled as a Gaussian Bayesian Network (GBN) and Parameter Generation Neural Networks (PGNNs) (Dong et al., 2022) are used to control the parameters of the GBN. Note this model makes two assumptions that are standard in temporal domains: the first-order Markov and stationary transition distribution assumptions.
2. **GRU**: a strong discriminative baseline model that uses Gated Recurrent Units (GRUs) (Bahdanau et al., 2014) to predict the next state given all previously observed states. Compared to Long-Short Term Memories (LSTMs) (Hochreiter and Schmidhuber, 1997), GRU is more efficient to train and often achieves comparable or better performance (Chung et al., 2014).
3. **GRU-GBN**: a variant of our model where we use GRU as the RNN block (see Figure 1) and GBNs as the underlying template continuous density.
4. **GRU-IndMGx3**: a variant of our model where the underlying continuous density is replaced with a three-component mixture of multivariate Gaussians with diagonal covariance matrices.

Note that for the NN-GBN and GRU-GBN models, each node (random variable) inside the GBN is restricted to have up to 5 parents to ensure that these two models scale linearly in the dimension of the datasets, which allows them to handle large datasets like *wingbeat* efficiently.

We evaluated the discriminative regression performance of the above models using three standard regression metrics implemented in scikit-learn (Pedregosa et al., 2011): Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE) and Median Absolute Error (MdAE).

In addition, we are also interested in how well the predicted values can be used to make decisions (Bi and Bennett, 2003). In such cases, we only care about whether the predicted values are within some error threshold of the true value and the absolute difference is ignored. For example, consider

Table 2: Trajectory prediction performance of our models (GRU-GBN, GRU-IndMGx3) against NN-GBN, GRU on eleven real-world sequential datasets under MAPE, MAE, MdAE and MREC metrics (Best results and results within 5% deviation to the best are marked in bold).

Metric	Model	japanvowels	natops	fingermov.	lsst	wordrec.	conll2000	arabicdigit	heartbeat	facedetect	wingbeat	brown
MAPE	NN-GBN	**1.540**	3.170	3.860	2.747	1.148	**1.490**	3.493	3.729	**4.597**	17.917	2.129
	GRU	2.331	3.009	1.571	8.939	0.870	1.753	**3.322**	3.536	5.979	14.130	**1.674**
	GRU-GBN	**1.520**	**1.242**	**1.399**	3.238	**0.771**	2.178	4.058	3.535	4.882	17.505	2.540
	GRU-IndMGx3	**1.493**	1.730	1.841	**2.296**	0.899	4.367	**3.473**	**3.222**	6.659	**2.323**	4.522
MAE	NN-GBN	0.262	0.156	0.316	**0.413**	0.286	**0.803**	0.528	0.356	0.548	0.274	**0.820**
	GRU	**0.224**	0.109	0.153	0.472	0.226	**0.797**	**0.464**	**0.298**	**0.512**	**0.207**	**0.794**
	GRU-GBN	**0.232**	**0.100**	**0.130**	**0.425**	**0.212**	0.838	0.499	0.358	**0.518**	0.251	0.835
	GRU-IndMGx3	**0.226**	**0.097**	0.207	**0.416**	**0.219**	0.976	**0.467**	0.341	0.542	**0.205**	0.939
MdAE	NN-GBN	0.215	0.124	0.261	0.274	0.189	**0.665**	0.430	0.228	0.415	0.103	**0.682**
	GRU	**0.179**	0.080	0.123	0.347	0.145	**0.663**	**0.379**	**0.199**	**0.386**	0.071	**0.655**
	GRU-GBN	**0.185**	**0.077**	**0.099**	0.282	0.128	0.705	0.409	0.236	**0.389**	0.091	0.694
	GRU-IndMGx3	**0.182**	**0.075**	0.113	**0.261**	**0.119**	0.800	**0.377**	0.214	0.413	**0.066**	0.747
MRAR	NN-GBN	8.714	16.497	6.742	21.048	11.094	2.626	4.365	12.851	**4.835**	20.902	2.497
	GRU	**10.532**	21.651	13.544	11.147	14.160	2.615	**5.129**	12.518	**4.798**	**26.576**	2.684
	GRU-GBN	**10.410**	**23.398**	**16.762**	**22.501**	16.129	2.452	4.649	11.366	**4.738**	23.177	2.446
	GRU-IndMGx3	**10.342**	**24.022**	14.862	**23.466**	**17.197**	**4.212**	**5.191**	**14.055**	4.337	**27.959**	**6.006**

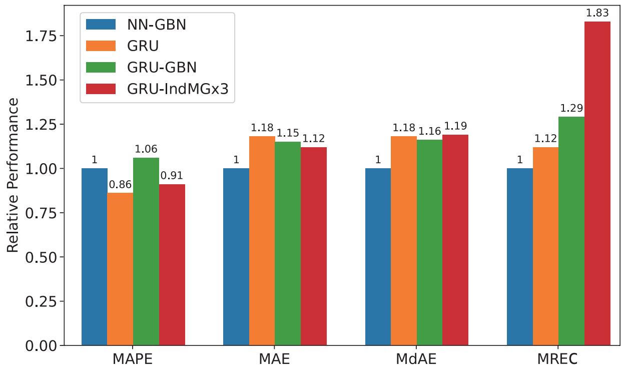

Figure 2: Average relative performance (the higher, the better) of each model against NN-GBN over all datasets for each metric, in the trajectory prediction task.

a temperature prediction scenario where we use a model to predict the temperature of the next day given previous days' temperatures. Suppose that a person will go out for fun only if the temperature is below 90°F. Assume that the true temperature is $100°$F, and therefore, any predicted temperature that is within a 10% error threshold of the true value will not affect the decision to go out.

None of the above standard error metrics reflect the correctness of the model when used as part of a decision-making processes based on thresholds. To assess this, we added one more evaluation metric based on Regression Error Characteristic (REC) curves proposed by Bi and Bennett (2003). These curves are similar to Receiver Operating Characteristic curves Hanley and McNeil (1982) that are widely used for assessing the performance of classification algorithms. Specifically, the REC curve is calculated as follows. For a given error threshold p, it first calculates r as the ratio of predicted values that fall into an interval centered at true value t with a width of $2p$. By varying the threshold p and calculating the corresponding ratio r, we can obtain an REC curve formed by points $\{(p_i, r_i)\}$. This curve is strictly increasing and the area under the REC curve can serve as an indicator of the model's decision performance. In our evaluation, we will refer to this area as the Mean Regression Error Characteristic (MREC) score. In the following experiments, as we are more concerned about relative errors, we will consider error thresholds from the set $\{5\%t, 10\%t, ..., 50\%t\}$ where t is the true value of the variable.

All models are implemented using Pytorch (Paszke et al., 2019) and trained using the Adam optimizer for a fixed number of epochs (120). As for hyperparameter tuning, we employ Optuna (Akiba et al., 2019) to conduct efficient and intelligent search over the hyperparameter space for 50 trials where the objective is defined to minimize the loss on the validation set. The hyperparameters we consider and their respective proposal distributions are: (1) drop out probability from a uniform distribution over the interval $[0, 0.4]$; (2) maximum learning rate from a log-uniform distribution over the interval $[10^{-4}, 5 \times 10^{-2}]$; (3) hidden size for GRU (feature size for NN-GBN) from a uniform distribution over the interval $[2D, \min(8D, 10^3)]$, where D is the dimension of the dataset, and (4) weight decay from a log-uniform distribution over the interval $[10^{-4}, 1]$. The best hyperparameters found for each model under each dataset can be found in the Appendix B.

All experiments were conducted on a workstation with a 16-core Intel Xeon Gold 6130 CPU and two Quadro P5000 GPUs. The datasets and code used in the experiments is publicly available on GitHub [2].

[2] *https://github.com/LeonDong1993/Probabilistic_RNN*

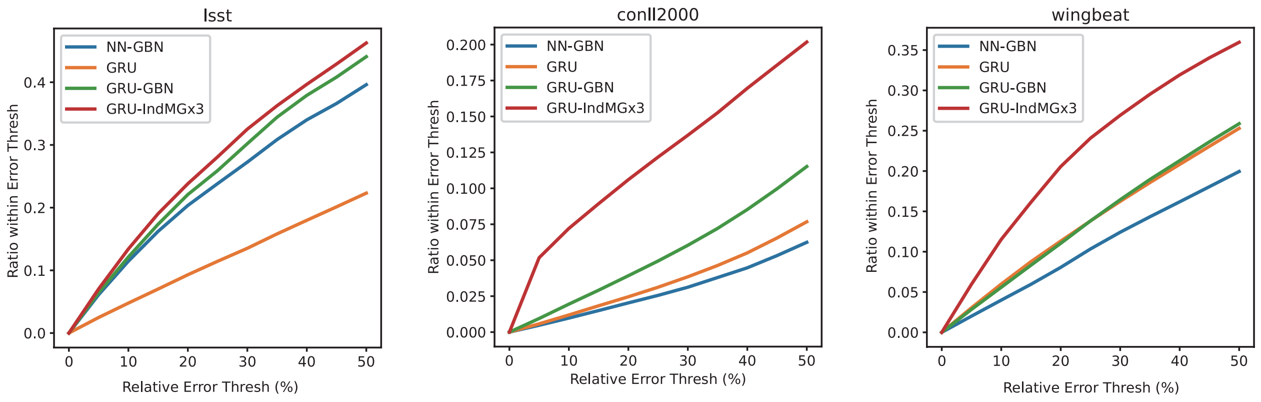

Figure 3: Regression Error Characteristic (REC) curve of four models on *lsst*, *conll2000* and *wingbeat* datasets for the trajectory prediction task.

4.1 Trajectory Prediction

In this set of experiments, we consider the trajectory prediction problem where we aim to predict the next few states based on what we have observed so far. Specifically, for each sequence in the test set, we manually mask the last three observed states and ask the above models to predict those states given all previous states as input.

We evaluated the performance of each model under each metric for all of the datasets. We present the full results in Table 2 and summarize these results by calculating the average relative performance of four models against NN-GBN over all datasets, under each evaluation metric. Specifically, the relative performance of a model M under metric C is calculated as the performance of M on metric C divided by the performance of NN-GBN on metric C. If smaller values are better for a given metric C, we invert the result so that higher relative performances are always better. Figure 2 shows the summarized results.

Observe, first, that our models can achieve comparable or better performance than both the NN-GBN and GRU baselines. On the one hand, when compared to NN-GBN, it is not surprising that our GRU-GBN model is always better since it models the full transition distribution (in fact, NN-GBN can be treated as a special case of our model). As for the GRU-IndMGx3 model, it achieves significant performance boost to NN-MG except for MAPE metric and we found this is caused by some overfitting on two NLP datasets. On the other hand, compared to GRU, we can achieve comparable performance in terms of MAE and MdAE metric and significant better results for both MAPE and MREC metric. This is not trivial since GRU is a discriminative model that is tailored for this task while ours are generative models designed for general inference tasks. Nevertheless, one of the advantages of being generative is that we can consider all three future states together and optimize them jointly over the whole distribution, which explains the competitive results of our models.

Second, if we consider the performance of each model over all metrics, NN-GBN underperforms the other models in most cases. This is likely because the first-order Markov and stationary assumptions used in the model are not satisfied on average across the data sets – given the diversity of our datasets, the average performance of NN-GBN is heavily hurt by those sequential datasets that have long-range dependencies.

Last, the GRU model has inferior performance on the relative measures (i.e., MREC, MAPE). On the contrary, our models tend to exhibit superior performance on relative measures, especially MREC. These observations reveal an important difference between our models and GRU: the ratio of predictions that are within a fixed error threshold of their true value always appears to be higher under our models. Heuristically, as the MAE of the different methods tends to be quite close, this means that our models tend to produce predictions that are often quite close to the true value but in rare instances our model can yield very poor estimates. In contrast, the predictions generated by GRU are neither particularly close nor particularly far from the true value. To illustrate this, we show the REC curve of all models for *lsst*, *conll2000* and *wingbeat* datasets in Figure 3 (visualizations for other datasets can be found in the Appendix A). As we can see, our models have much higher chance to generate accurate predictions compared to the NN-GBN and GRU models, especially when the error threshold is small.

In short, although GRU is a discriminative model tailored for this task, our generative models can still achieve comparable or better performance on standard error metrics such as MAE while yielding a higher chance to give predictions within a fixed relative error compared to the two competitors.

4.2 Sequence Completion

In this experiment, we consider a scenario in which the sequence can have missing entries at each time step. In prac-

Table 3: Sequence completion performance of our models (GRU-GBN, GRU-IndMGx3) against NN-GBN, GRU on eleven real-world sequential datasets under MAPE, MAE, MdAE and MREC metrics (Best results and results within 5% deviation to the best are marked in bold).

Metric	Model	japanvowels	natops	fingermov.	lsst	wordrec.	conll2000	arabicdigit	heartbeat	facedetect	wingbeat	brown
MAPE	NN-GBN	1.162	2.210	1.812	**2.651**	1.051	3.780	8.118	12.746	**3.989**	29.255	2.545
	GRU	1.238	1.628	1.210	5.219	1.195	**2.097**	8.097	**7.390**	4.911	7.472	**1.682**
	GRU-GBN	**0.999**	**1.086**	**0.822**	3.044	0.902	3.451	5.189	15.682	4.875	21.482	3.078
	GRU-IndMGx3	**1.038**	1.312	1.183	6.739	**0.840**	3.346	**4.430**	12.925	5.514	**5.823**	2.570
MAE	NN-GBN	0.189	0.159	0.224	**0.430**	0.195	0.773	0.384	**0.246**	0.514	0.397	0.767
	GRU	0.224	0.124	0.162	0.504	0.209	0.776	0.436	0.265	**0.499**	**0.322**	0.777
	GRU-GBN	**0.158**	**0.089**	**0.122**	**0.445**	**0.153**	0.770	**0.347**	**0.241**	**0.480**	0.356	0.755
	GRU-IndMGx3	0.196	0.101	0.166	0.480	0.196	**0.726**	**0.360**	**0.235**	**0.479**	**0.324**	**0.704**
MdAE	NN-GBN	0.146	0.114	0.167	**0.271**	0.104	0.625	0.291	0.144	0.386	0.157	0.614
	GRU	0.173	0.087	0.127	0.353	0.119	0.637	0.346	0.167	0.377	**0.066**	0.640
	GRU-GBN	**0.116**	**0.059**	**0.089**	**0.267**	**0.080**	0.620	**0.261**	**0.140**	**0.353**	0.124	0.603
	GRU-IndMGx3	0.144	0.067	0.106	0.283	**0.078**	**0.569**	0.276	**0.136**	**0.353**	0.074	**0.541**
MREC	NN-GBN	12.560	17.873	10.452	**19.890**	17.907	2.809	6.445	**15.278**	**5.005**	15.051	2.929
	GRU	10.956	20.975	13.269	11.175	16.040	2.723	5.367	13.212	4.882	**24.147**	2.736
	GRU-GBN	**15.836**	**27.283**	**18.131**	**20.196**	**21.312**	2.839	**7.444**	**14.897**	**5.196**	17.435	2.987
	GRU-IndMGx3	14.604	**27.022**	16.334	18.385	**21.941**	**6.130**	7.054	**15.456**	**5.050**	**23.198**	**8.487**

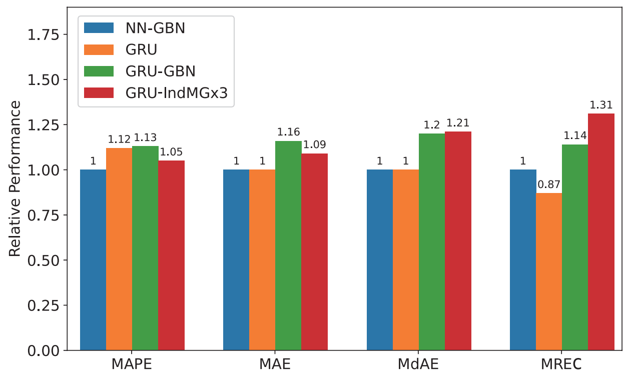

Figure 4: Average relative performance (the higher, the better) of each model against NN-GBN over all datasets for each metric, in the sequence completion task.

tice, these types of queries can arise for a variety of reasons such as equipment malfunctions, measurement failures, data corruption, human errors, etc. Our task is to complete those missing entries using their most probable values: in this task, we are trying to predict the values using evidence from both previous and future time slices. Recall that GRU models are not able to make predictions if part of the variables are not observed in the previous time slices. Therefore, in order to fill out all missing values, we use GRU to predict the missing variables starting from the first time slice and moving forward. Note that, as future information is not used in the prediction, estimates produced by GRU in this way are likely to be suboptimal.

We conducted a simulation experiment to evaluate the models' sequence completion performance by randomly removing 50% of the variables in the last ten time slices of each sequence in the test set. As before, we present the full results in Table 3 and summarize the average relative performance of all models against NN-GBN over all 11 datasets in Figure. 4. We also visualized the REC curve of sequence completion task for all datasets in Appendix A.

In this task, our models are able to achieve significantly better results compared to both NN-GBN and GRU models, in almost every metric. There is one exception to this general observation: the GRU-IndMGx3 model has slightly worse MAPE compared to GRU, and this, again, appears to be the result of the overfitting issue discussed in the previous section. Furthermore, we observed that the performance of GRU has no improvement over the NN-GBN model if we jointly consider all four metrics, despite the fact that NN-GBN only models the one-step simple transition distribution. The poor performance of GRU is not surprising because one of the strengths of the statistical models is that they admit a joint inference procedure over all of the query variables while GRU does not have such a feature. Specifically, as the GRU model is trained for the discriminative task, it can only use past evidence to predict the missing values in current time slice and use these estimated results to produce further predictions.

4.3 Comparison of Generative Performance

We compared the generative performance of our models with the NN-GBN baseline model using the average log-likelihood of the test-set sequence. The detailed results on each dataset are presented in Table 4.

We observed that both of our models achieved higher test-set log-likelihood compared to the baseline, which is not surprising. Specifically, the RNN-IndMGx3 model outperformed all others or achieved similar results in 9 out of the 11 cases, while the RNN-MG model performed the best or similarly in 4 out of 11 cases. This indicates that our models are capable of handling long-term dependencies and capture more complex dynamics inside the sequence.

Moreover, we found that the RNN-IndMGx3 model had significantly higher log-likelihood compared to the other two models. This is likely because the RNN-IndMGx3 model is more expressive than the RNN-MG model and can easily handle multi-modal distributions.

Table 4: Average test-set loglikelihood achieved on each dataset for our models against NN-MG (best and results within 5% deviation to the best are marked in bold).

Dataset	NN-MG	RNN-IndMGx3	RNN-MG
japanvowels	-1.1016	0.6254	**1.2994**
natops	-2.6976	**7.2336**	5.6073
fingermovement	-3.2159	9.1962	**12.3232**
lsst	-2.9415	**1.3694**	-0.4
wordrecognition	1.5606	**3.6882**	2.6556
conll2000	-137.9127	**-114.2535**	-139.2675
arabicdigit	-9.2131	**-8.1424**	**-8.1398**
heartbeat	2.2209	**11.4525**	0.2584
facedetect	-174.5454	**-155.2994**	**-158.5556**
wingbeat	-235.3982	**-57.2524**	-203.3685
brown	-137.1666	**-103.0166**	-136.5463
Average	-63.67	**-36.76**	-56.74

5 DISCUSSION AND CONCLUSION

In this work, we developed a flexible modeling framework for continuous temporal domains that combines PGMs and RNNs in a way that overcomes the primary weaknesses inherent in each approach. We demonstrated the superior discriminative performance of our framework through two different prediction tasks on a variety of datasets. In particular, we found that (1) although our models are generative, we can achieve comparable or better performance than discriminative models such as GRU; (2) the first order Markov assumption can hurt a model's generative as well as discriminative performance, especially on sequential data with long term dependencies, and (3) our models tend to give highly accurate predictions in most cases, which makes our model ideal for scenarios in which decisions are made based on thresholding the predicted values.

One of the main limitations of our model is that, it is not tractable when evaluating the likelihood of a partially observed sequence as inference in this case involves a high dimensional integral over the neural network functions. One possible approximation method is to use particle filtering to generate a set of samples for the missing variables and use those samples to estimate the integral. Another minor issue is that our model contains several hyperparameters that require some tuning effort. However, compared to modern deep neural networks, our architecture is typically small and the longest training time for all datasets we discussed in this paper is less than 25 minutes on a low-end GPU. More detailed training and inference time of all models under each dataset is included in the Appendix C.

In future work, we hope to explore applications of this approach to spatio-temporal data as well as to investigate the performance of a larger variety of RNN architectures and tractable continuous densities. In addition, we are also interested in adapting this framework for temporal models with hidden variables/missing data at training time.

Acknowledgements

This work was supported in part by the DARPA Perceptually enabled Task Guidance (PTG) Program under contract number HR00112220005 and by the National Science Foundation CAREER award IIS-1652835.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

Parviz Asghari, Elnaz Soleimani, and Ehsan Nazerfard. Online human activity recognition employing hierarchical hidden markov models. *Journal of Ambient Intelligence and Humanized Computing*, 11(3):1141–1152, 2020.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Jinbo Bi and Kristin P Bennett. Regression error characteristic curves. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 43–50, 2003.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Lecture notes: Probabilistic circuits: Representation and inference. February 2020. URL `http://starai.cs.ucla.edu/papers/LecNoAAAI20.pdf`.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Hailiang Dong, Chiradeep Roy, Tahrima Rahman, Vibhav Gogate, and Nicholas Ruozzi. Conditionally tractable density estimation using neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 6933–6946. PMLR, 2022.

W Nelson Francis and Henry Kucera. Brown corpus manual. *Letters to the Editor*, 5(2):7, 1979.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *International conference on machine learning*, pages 1462–1471. PMLR, 2015.

Marco Grzegorczyk. An introduction to gaussian bayesian networks. In *Systems biology in drug discovery and development*, pages 121–147. Springer, 2010.

James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

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

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Tao Li, Minsoo Choi, Kaiming Fu, and Lei Lin. Music sequence prediction with mixture hidden markov models. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 6128–6132. IEEE, 2019.

Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31(11):4405–4423, 2020.

Daniel Lowd and Pedro M. Domingos. Learning arithmetic circuits. In *Uncertainty in Artificial Intelligence*, pages 383–392, 2008.

Mazen Melibari, Pascal Poupart, Prashant Doshi, and George Trimponias. Dynamic sum product networks for tractable inference on sequence data. In *Conference on Probabilistic Graphical Models*, pages 345–355. PMLR, 2016.

Thomas P Minka. Expectation propagation for approximate bayesian inference. *arXiv preprint arXiv:1301.2294*, 2013.

Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. University of California, Berkeley, 2002.

Rim Nasfi, Manar Amayri, and Nizar Bouguila. A novel approach for modeling positive vectors with inverted dirichlet-based hidden markov models. *Knowledge-Based Systems*, 192:105335, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset Networks: A Simple, Tractable, and Scalable Approach for Improving the Accuracy of Chow-Liu Trees. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II*, pages 630–645, 2014.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial intelligence and statistics*, pages 814–822. PMLR, 2014.

Chiradeep Roy, Tahrima Rahman, Hailiang Dong, Nicholas Ruozzi, and Vibhav Gogate. Dynamic cutset networks. In *International Conference on Artificial Intelligence and Statistics*, pages 3106–3114. PMLR, 2021.

Erik F Sang and Sabine Buchholz. Introduction to the conll-2000 shared task: Chunking. *arXiv preprint cs/0009008*, 2000.

Xiaoting Shao, Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Thomas Liebig, and Kristian Kersting. Conditional sum-product networks: Imposing structure on deep probabilistic architectures. In *International Conference on Probabilistic Graphical Models*, pages 401–412. PMLR, 2020.

Nils Thoma, Zhongjie Yu, Fabrizio Ventola, and Kristian Kersting. Recowns: Probabilistic circuits for trustworthy time series forecasting. *arXiv preprint arXiv:2106.04148*, 2021.

Panayiota Touloupou, Bärbel Finkenstädt, and Simon EF Spencer. Scalable bayesian inference for coupled hidden markov and semi-markov models. *Journal of Computational and Graphical Statistics*, 29(2):238–249, 2020.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

Vickers William. Time series classification repository. `https://timeseriesclassification.com/dataset.php`, 2021.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

A Extra Visualizations of Regression Error Characteristic (REC) Curve

As discussed in the experiment section, our models tend to produce predictions that are often quite close to the true value but in rare instances our model can yield very poor estimates. In contrast, the predictions generated by GRU are neither particularly close nor particularly far from the true value. We illustrate the REC curve of our models against GRU and NN-GBN on all of datasets for both trajectory prediction and sequence completion task in Figure.5 and Figure.6, respectively. As we can see, our models have much higher chance to generate accurate predictions compared to the NN-GBN and GRU models, especially when the error threshold is small.

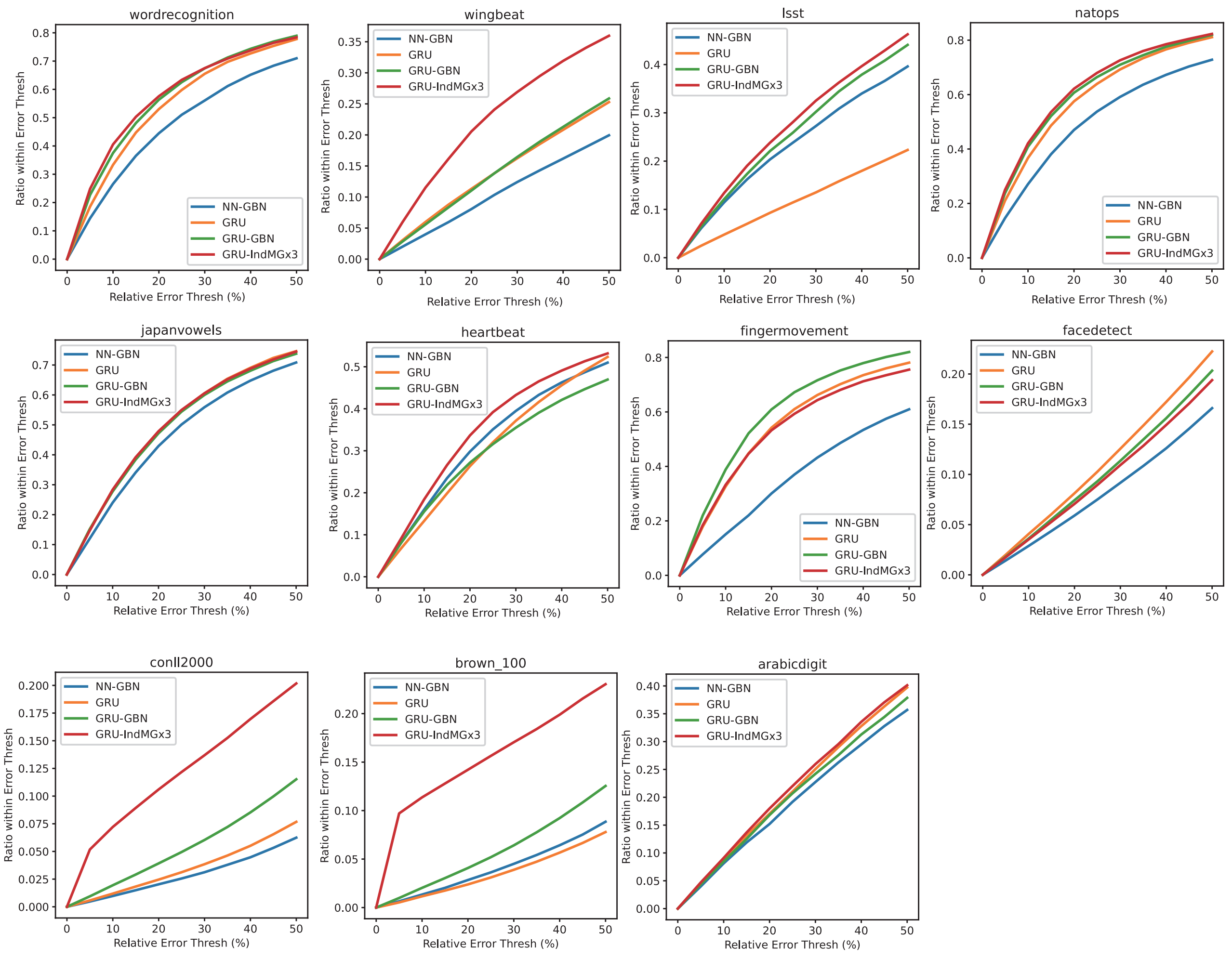

Figure 5: Regression Error Characteristic (REC) curve of four models on 11 real-world sequential datasets for trajectory prediction task.

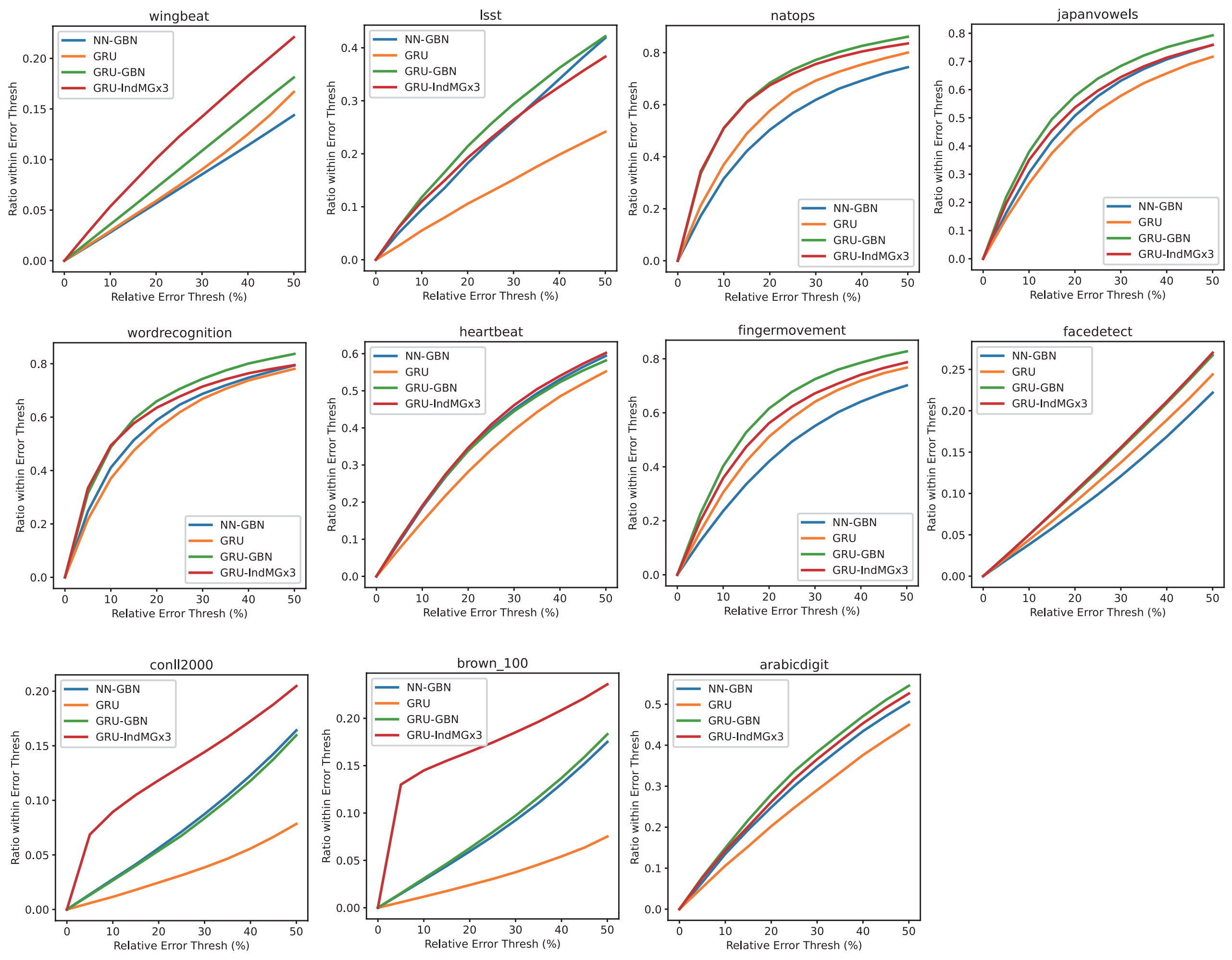

Figure 6: Regression Error Characteristic (REC) curve of four models on 11 real-world sequential datasets for sequence completion task.

B Best Hyper-Parameters Found for All Models and Datasets

In our study, we used Optuna Akiba et al. (2019) to fine-tune the hyperparameters of our models. Optuna is an automated hyperparameter optimization framework that uses a sequential model-based optimization (SMBO) algorithm to explore the hyperparameter space efficiently. One of the main advantages of using Optuna is that it can find good hyperparameters quickly and efficiently, even for complex models with many hyperparameters.

The best hyperparameters discovered for each model under all datasets are presented in Table 5. It is worth noting that, in our case, the values for the drop rate, learning rate, and weight decay can be any arbitrary continuous number. This is because Optuna automatically searches for the best value within a given interval, which is different from traditional grid search where the set of values are predetermined. By leveraging Optuna's efficient and effective search algorithm, we were able to identify the optimal hyperparameters for our models in a timely and automated manner, leading to improved performance on the test set.

Table 5: Best hyper parameters found for all models and datasets.

Dataset	Model	Hidden/Feature Size	Drop Rate	Learning Rate	Weight Decay
japanvowels	NN-MG	72	0.015719	0.009861	0.000200
	RNN	72	0.070386	0.002603	0.000136
	RNN-IndMGx3	72	0.056716	0.002332	0.002720
	RNN-MG	96	0.036241	0.004912	0.021372
natops	NN-MG	112	0.035675	0.009490	0.000383
	RNN	112	0.137134	0.006485	0.000125
	RNN-IndMGx3	84	0.027134	0.010418	0.001549
	RNN-MG	112	0.024404	0.002890	0.002322
fingermovement	NN-MG	168	0.244687	0.012035	0.002450
	RNN	168	0.031809	0.007291	0.000122
	RNN-IndMGx3	224	0.048307	0.004367	0.000796
	RNN-MG	168	0.018321	0.003374	0.001363
lsst	NN-MG	48	0.051760	0.011565	0.000208
	RNN	48	0.069321	0.006024	0.000211
	RNN-IndMGx3	48	0.020662	0.015836	0.000301
	RNN-MG	36	0.047935	0.021818	0.000215
wordrecognition	NN-MG	72	0.018344	0.013527	0.000156
	RNN	18	0.006405	0.028050	0.000189
	RNN-IndMGx3	72	0.023400	0.014779	0.000181
	RNN-MG	54	0.002958	0.020209	0.001340
conll2000	NN-MG	400	0.116993	0.000261	0.329768
	RNN	200	0.060404	0.001736	0.000165
	RNN-IndMGx3	400	0.330103	0.006267	0.004380
	RNN-MG	200	0.280500	0.007497	0.018540
arabicdigit	NN-MG	104	0.032135	0.016248	0.000233
	RNN	104	0.000349	0.001478	0.000172
	RNN-IndMGx3	104	0.108671	0.003683	0.000315
	RNN-MG	104	0.035049	0.002076	0.001295
heartbeat	NN-MG	244	0.236493	0.009757	0.000498
	RNN	366	0.276080	0.001106	0.000162
	RNN-IndMGx3	122	0.100064	0.009874	0.004329
	RNN-MG	488	0.016831	0.002614	0.043031
facedetect	NN-MG	864	0.132928	0.020272	0.002454
	RNN	576	0.190621	0.003568	0.000104
	RNN-IndMGx3	576	0.346137	0.004688	0.007418
	RNN-MG	576	0.235544	0.002761	0.008861
brown	NN-MG	800	0.175577	0.000207	0.104149
	RNN	800	0.010860	0.001171	0.000114
	RNN-IndMGx3	200	0.163857	0.005269	0.001261
	RNN-MG	400	0.223694	0.047403	0.007399
wingbeat	NN-MG	398	0.353549	0.007781	0.029734
	RNN	398	0.154231	0.000418	0.000101
	RNN-IndMGx3	398	0.297048	0.010332	0.000299
	RNN-MG	796	0.274634	0.003165	0.005794

C Training and Inference Time for All Models and Datasets

We present the training time (including hyper parameter tuning) and inference/prediction time spent for each model under all datasets in Table. 6. Specifically, the training time is measured in minutes, representing the time spent after 50 trials using Optuna. The inference time, measured in seconds, is the total prediction time on 300 test samples. It should be noted that the training and inference time heavily depend on the hyper-parameters, particularly the hidden/feature size of the model. Thus, tuning time can be highly biased if Optuna tries more hyper-parameters with large hidden/feature size. The same applies to inference time if the best parameter found has high hidden/feature size.

We have the following observations. First, RNN is the fastest model for both training and prediction. This is because it is a discriminative model and it doesn't need 'inference' to generate predictions. Second, our model has a faster training time in general compared to the NN-MG model. However, when conducting inference, as our model needs to jointly optimize all missing information using all observations, it is slightly slower than NN-MG. However, we notice that the difference is quite small and sometimes we are faster as well.

Table 6: The training and the inference/prediction time for all models, datasets.

Phase	Model	japanvowels	natops	fingermovement	lsst	wordrecognition	conll2000	arabicdigit	heartbeat	facedetect	brown	wingbeat
Tuning (mins)	NN-MG	164.35	161.68	201.57	101.32	142.47	187.77	173.97	223.88	692.78	218.10	253.72
	RNN-STD	14.35	13.12	21.63	39.53	35.95	12.57	84.53	133.22	389.78	125.07	36.98
	RNN-IndMGx3	64.28	43.32	77.67	172.52	92.62	114.92	274.52	77.87	324.37	142.65	134.25
	RNN-MG	64.23	47.67	104.73	132.42	87.68	150.42	286.13	133.07	727.55	308.73	323.03
Inference (secs)	NN-MG	40	29	85	30	45	192	45	255	8002	356	354
	RNN-STD	1	1	1	1	6	1	2	32	15	10	1
	RNN-IndMGx3	249	260	134	407	895	321	461	1283	1987	245	193
	RNN-MG	105	147	94	162	612	251	183	3411	2026	434	638