A recurrent Markov state-space generative model for sequences

Anand Ramachandran[1], Steven S. Lumetta[1], Eric Klee[2], Deming Chen[1]
[1]University of Illinois at Urbana-Champaign, [2]Mayo Clinic

Abstract

While the Hidden Markov Model (HMM) is a versatile generative model of sequences capable of performing many exact inferences efficiently, it is not suited for capturing complex long-term structure in the data. Advanced state-space models based on Deep Neural Networks (DNN) overcome this limitation but cannot perform exact inferences. In this article, we present a new generative model for sequences that combines both aspects, the ability to perform exact inferences and the ability to model long-term structure, by augmenting the HMM with a deterministic, continuous state variable modeled through a Recurrent Neural Network. We empirically study the performance of the model on (i) synthetic data comparing it to the HMM, (ii) a supervised learning task in bioinformatics where it outperforms two DNN-based regressors and (iii) in the generative modeling of music where it outperforms many prominent DNN-based generative models.

1 Introduction

The Hidden Markov Model (HMM) [25] has been applied to a wide range of applications from speech recognition [10] to read alignment [8]. While it is versatile, the HMM is unable to represent complex long-term patterns due to a parsimonious state-space [30]. Specifically, at any time-step, the future of a sequence generated by an HMM is conditionally independent of its past, given the HMM's state at that time-step. The HMM's state, being a discrete random variable, is not descriptive enough to capture sufficient information about the history of its operation. Hence, the past can only exert a limited influence on the future, contributing to an inability to model long-term patterns with complex structure.

One approach to modeling long-term patterns is to use the internal state of a Recurrent Neural Network (RNN) to parameterize an output distribution [13] or to set the hidden state and parameters of a generative model such as a Restricted Boltzmann Machine [31] or a Neural Autoregressive Distribution Estimator [5]. However, the only information about history here is the RNN state, which due to its deterministic nature, does not suit applications where it is necessary to determine higher order features or "regimes" under which the data is generated. Models with highly descriptive stochastic state-spaces have also been constructed. For example, discrete state-spaces which can encode exponentially higher amount of information than the HMM [11][30], and models where stochastic state variables are associated with an RNN's deterministic state [7][9][19][12] have been proposed. These methods cannot perform exact inference, instead an approximate posterior distribution parameterized by a neural network is provided.

The advanced state-space models do not have some functional aspects of the HMM that make it successful in applications where domain knowledge about a stochastic, discrete latent space is used to design a solution in a principled, and well-reasoned manner. E.g.,

- Cases where the type of distribution under different regimes is known (e.g., [2], or HMMs used in speech). The HMM allows constructing a sequential model of these regimes with training and inference posed as optimization problems. Alternatives using advanced regressors such as RNNs may depend on sub-optimal algorithms for inference (e.g., Connectionist Temporal Classification [14] in speech) that may require heuristics or manual tuning.
- Cases where *temporal logic* is important, where the regimes need to conform to rules. For example, in sequence alignment [8], the alignments produced by the model should have logical consistency (one character should not be aligned to two places etc). In other cases, certain state-transitions may be disallowed. Such logic may be coded into the HMM's architecture easily. It is not readily clear how such logic may be enforced on many advanced generative models with complex state spaces.
- Cases where specialized inferences are used. The

Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89. Copyright 2019 by the author(s).

HMM offers the popular Viterbi, forward, and backward inferences which are exact and efficient. In addition, the HMM allows drawing of inferences where arbitrary regimes are included or excluded. The *likelihood ratio test* popular in bioinformatics uses such inferences to compare hypotheses.

Hence, while the HMM cannot capture long-term patterns, it provides an *interface* to encode domain-specific knowledge, with the ability to draw exact inferences. Complex state-space models that can represent long-term patterns lack such an interface, or cannot perform exact inference. Hence, there is currently a need for a model that can represent complex long-term patterns well, and provide a framework in which domain knowledge may be incorporated efficiently with the ability to perform exact inference.

In this paper, we present such a model. It augments the HMM by adding a continuous, deterministic state variable that is modeled using an RNN. This augmentation removes the conditional independence of state transition and emission on history, boosting the ability of the model to represent long-term patterns. At the same time, the model preserves the powerful interface of the HMM, as well as its ability to perform exact inferences. We examine the model's performance empirically in three cases, spanning supervised and unsupervised learning applications, where the ability to represent complex long-term relationships can be helpful. The model performs at par or better than existing methods in these cases. Specifically, the contributions of this work are as follows.

- A generative model for sequences that can represent long-term structure in the data and perform exact inferences.
- An emprical comparison of the model with the HMM using two synthetic datasets that contain non-linear long-range patterns.
- Application of the model to the bioinformatic task of determining the binding-specificity of Transcription Factors, where it outperforms two DNN-based regressors. This is a supervised learning application.
- Application of the model to the generative modeling of piano music sequences, where the model outperforms competing methods in two out of four cases. This is an unsupervied learning application.

2 Methods

2.1 Model description

We term the sequence generative model we introduce in this paper, the Long Short-term Graphical Model (LSGM). The LSGM consists of a finite set of discrete states (say Π), and a continuous state, $Q \in R^n$, where n is a positive integer indicating the dimensionality of the state. Consider the generation of a sequence of length T from the LSGM, namely $X_{1:T} = \{x_1, x_2, \cdots x_T\}$. The generative process may be described as follows. At the start of the time-step, t, the LSGM is said to be in one of the discrete states $\pi_t \in \Pi$, and a continuous state Q_{t-1}. During the time-step, the following operations are performed (Figure 2a).

- π_t emits x_t according to a probability distribution, $P[x_t|\pi_t, Q_{t-1}]$. This is called the emission distribution.
- Q_{t-1} is updated to Q_t in a deterministic fashion following $Q_t = f(Q_{t-1}, x_t)$. f is an RNN.
- LSGM transitions to a state $\pi_{t+1} \in \Pi$ sampled randomly from a second probability distribution, $P[\pi_{t+1}|\pi_t, Q_t]$, called the transition distribution.

The first state of the LSGM, π_1, may be determined by sampling from a distribution $P[\pi_1|Q_0]$. This may be considered as a separate initializing distribution. Alternatively, we may assume that the LSGM rests in a start state prior to any symbols being emitted. In this case, $P[\pi_1|Q_0]$ is considered the transition distribution at time $t = 0$ from the start state. Q_0 is some fixed parameter of the model.

Dependence on Q_t may be rewritten as dependence on $X_{1:t}$ and Φ, the LSGM's parameter set. Hence, we may rewrite the emission and transition distributions as follows.

$$P[x_t|\pi_t, Q_{t-1}] = P[x_t|\pi_t, X_{1:t-1}, \Phi] \triangleq e_t(\pi_t) \qquad (1)$$

$$P[\pi_{t+1}|\pi_t, Q_t] = P[\pi_{t+1}|\pi_t, X_{1:t}, \Phi] \triangleq \gamma_t(\pi_t, \pi_{t+1}) \qquad (2)$$

There are different ways to implement e_t. In one implementation (an example of which is given in Figure 1a), we associate a feed-forward fully-connected (FC) neural network with each state. The input to each neural network is Q_t and the k^{th} neural network models the emission distribution associated with state k (or $e_t(k)$). In another implementation (an example of which is given in Figure 1b), there is a single feed-forward fully-connected neural network for the entire LSGM. In this case, to model the emission distribution associated with state k, the neural network accepts Q_t and k as inputs. Instead of using a single scalar value, k, as the input identifying the state, we could input an *embedding* of the state k. The embedding of a state is simply a real-valued vector associated with the state. The LSGM carries a dictionary mapping states to their embeddings. The second implementation scales well for large state-spaces as it shares a single fully-connected network among the states. Other parameter sharing schemes are possible - e.g., states may share all parameters of the Emission Network except the biases of the output layer, which may be initialized to reflect prior assumptions. The FC (interchangeably called "dense") network(s) implementing the emission distributions are collectively termed, the Emission network.

γ_t consists of $|\Pi|$ categorical distributions. Each distribution allows sampling from the LSGM state set, Π. This is implemented as a fully-connected feed-forward neural network which takes Q_t as input and provides $|\Pi|^2$ outputs which are arranged in a $|\Pi| \times |\Pi|$ matrix where each row is normalized using the softmax function. The k^{th} row represents the transition distribution associated with state k. Modeling of the transition distribution is illustrated in Figure 1c. The FC network implementing the transition distribution is termed the Transition network.

Implicit in the description of the generative process of the LSGM are the following conditional independence assumptions.

$$x_t \perp \pi_{t'} | \big(\pi_t, X_{1:t-1}\big),\ for\ t' < t \tag{3}$$

$$\pi_t \perp \pi_{t'} | \big(\pi_{t-1}, X_{1:t-1}\big),\ for\ t' < t-1 \tag{4}$$

Under these assumptions, the following recursions are applicable to the LSGM. The recursions are derived in the supplementary document.

$$P\big[X_{1:t+1}, \pi_{t+1} = j|\Phi\big] = \alpha_{t+1}(j) = e_{t+1}(j) \sum_k \alpha_t(k)\gamma_t(k,j) \tag{5}$$

$$P\big[X_{t+1:T}|\pi_t = k, X_{1:t}, \Phi\big] = \beta_t(k) = \sum_j \beta_{t+1}(j)\gamma_t(k,j)e_{t+1}(j) \tag{6}$$

The recurrences in Equations 5 and 6 are similar to the forward and backward recurrences in the HMM. Given the parameters γ_t and e_t, they have the same complexity, $O(N^2T)$, as the HMM versions for N states and T time-steps ($O(N)$ operations required per state per time-step). Similar to the forward and backward recursions, the Viterbi recursion is also applicable to the LSGM. At the same time, it may be noted that two of the independence assumptions used in the HMM have been removed in the LSGM. The assumptions not used in the LSGM, but used in the HMM are

$$x_t \perp x_{t'} | \pi_t,\ for\ t' < t \tag{7}$$

$$\pi_{t+1} \perp x_{t'} | \pi_t,\ for\ t' \leq t \tag{8}$$

Figure 2a summarizes the generative process in the LSGM. The generative process in an HMM is shown in Figure 2b. Compared to the HMM, the LSGM emissions and transitions are stochastically influenced by Q_t, which is a function of unlimited past history of model outputs. This function has the capability to represent rich information about the past because it is continuous and multidimensional. Figure 2c represents the relationships among the states in a three-state LSGM at time step t. Figure 3a represents the overall LSGM architecutre. It assumes the emission modeling method in Figure 1b.

2.2 Maximum Likelihood estimation

The LSGM may be trained through Maximum Likelihood (ML) estimation performed through gradient descent. During Maximum Likelihood (ML) estimation, a model's parameters are adjusted to improve the likelihood with which it generates a training set. For this, first, it must be possible to determine the likelihood of an observed sequence under the model. For the LSGM, this may be done by marginalizing the probability term in Equation 5 as follows.

$$P\big[X_{1:t}|\Phi\big] = \sum_{\pi_t} P\big[X_{1:t}, \pi_t|\Phi\big] = \sum_k \alpha_t(k) \tag{9}$$

Second, it must be possible to determine the derivative of an observed sequence with respect to the parameters of the model. The parameters of the LSGM (Φ) are the parameters of the emission (Φ_e) and transition networks (Φ_γ), and the parameters of the RNN implementing the Q_t updates (Φ_Q), as shown in Figure 3a. Hence, we should be able to compute $\frac{\partial P[X_{1:T}]}{\partial \Phi_e}$ etc. This may be achieved as follows (also, Figure 3b).

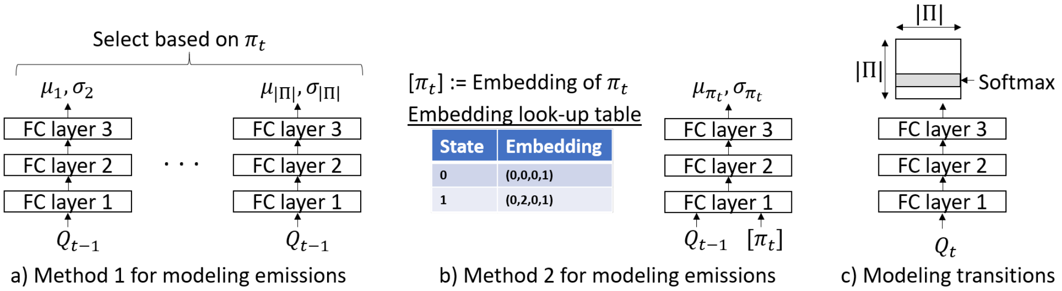
1. Determine $\frac{\partial P[X_{1:T}]}{\partial e_t}$ and $\frac{\partial P[X_{1:T}]}{\partial \gamma_t}$, for all t
2. Use the derivatives in step 1 to determine $\frac{\partial P[X_{1:T}]}{\partial \Phi_e}$, $\frac{\partial P[X_{1:T}]}{\partial \Phi_\gamma}$, and $\frac{\partial P[X_{1:T}]}{\partial Q_t}$ through the backpropagation algorithm
3. Invoke the backpropagation-through-time algorithm to determine $\frac{\partial P[X_{1:T}]}{\partial \Phi_Q}$ from $\frac{\partial P[X_{1:T}]}{\partial Q_t}$

Among these steps, the backpropagation (step 2) and the backpropagation through time (step 3) algorithms are well-known. Hence, we will treat step 1 in this article. Let Y be the likelihood of an observed sequence, or a function of its likelihood. Then, the following recursions hold, and summarize the steps needed to achieve step 1.

$$\frac{\partial Y}{\partial \alpha_t(j)} = \sum_k \frac{\partial Y}{\partial \alpha_{t+1}(k)} \frac{\partial \alpha_{t+1}(k)}{\partial \alpha_t(j)} = \sum_k e_{t+1}(k)\pi_t(j,k)\frac{\partial Y}{\partial \alpha_{t+1}(k)} \tag{10}$$

$$\frac{\partial Y}{\partial e_t(j)} = \frac{\partial Y}{\partial \alpha_t(j)} \frac{\partial \alpha_t(j)}{\partial e_t(j)} = \frac{\partial Y}{\partial \alpha_t(j)} \sum_k \alpha_{t-1}(k)\gamma_{t-1}(k,j) \tag{11}$$

$$\frac{\partial Y}{\partial \gamma_t(j,k)} = \frac{\partial Y}{\partial \alpha_{t+1}(k)} \frac{\partial \alpha_{t+1}(k)}{\partial \gamma_t(j,k)} = e_{t+1}(k)\alpha_t(j)\frac{\partial Y}{\partial \alpha_{t+1}(k)} \tag{12}$$



State	Embedding
0	(0,0,0,1)
1	(0,2,0,1)

a) Method 1 for modeling emissions

b) Method 2 for modeling emissions

c) Modeling transitions

Figure 1: (a) Emissions: $|\Pi|$ FC networks model $|\Pi|$ pairs, (μ_i, σ_i); the correct pair is picked based on the value of π_t and it parameterizes a Gaussian distribution that emits x_t (b) Emissions: an FC network accepts Q_{t-1} and $[\pi_t]$ as inputs and provides the mean and standard deviation of a Gaussian distribution that emits x_t (c) Transitions: An FC network outputs $|\Pi|^2$ outputs, which are arranged as a $|\Pi| \times |\Pi|$ matrix; each row is a probability vector thanks to the application of the softmax function

Equation 10 tells us how to propagate the derivative of the learning objective, Y, backward through the time-steps with respect to the forward variable. Equations 11 and 12 tell us, given the back-propagated objective with respect to the forward variable at different time-steps, $\partial Y / \partial \alpha_t(k)$, how to compute the derivatives with respect to e_t and γ_t. Once these recursions are finished, we have $\partial Y / \partial \gamma_t$ and $\partial Y / \partial e_t$, and steps 2-3 may be executed as described above. The specific chain rules shown in Equations 10–12 arise from the following dependencies (i) $\alpha_t(j)$ is used in the computation of $\alpha_{t+1}(k), \forall k$ (ii) $e_t(j)$ is used only in the computation of $\alpha_t(j)$ (iii) $\gamma_t(j,k)$ is used only in the computation of $\alpha_{t+1}(k)$. Equations 10-12 have a complexity of $O(N^2 T)$ to be completed, for T time-steps.

2.3 LSGM variants

We term the LSGM model presented so far a *first-order full LSGM*. A *first-order half LSGM* would additionally use one of the two independence assumptions in Equations 7, 8. In this case, one of (not both) transition or emission parameters will be decoupled from the RNN. In a *zeroth-order* LSGM model (full or half), the next-state transition is independent of the current state, π_t, but dependent on the history of observations Q_t. In these cases, the RNN state variable, Q is part of the model, and hence the model does not fall under the standard HMM paradigm. Half- and zeroth-order LSGMs may be used in cases where domain knowledge provides evidence for a simpler temporal structure in the data than that implied by the full LSGM formulation. The forward, backward, and Viterbi recurrences and their derivations given in the Supplementary document, and the ML estimation method for training given in Section 2.2 are valid in these cases.

2.4 Synthetic data and model overview

To empirically compare the LSGM to the HMM, we construct two sets of synthetic data. The first is a set of sequences of multidimensional data into which long-term non-linear relationships are introduced in increasing degrees. We compare two toy models, one an HMM and the other an LSGM, with matched number of parameters and see how their performance varies as the data becomes more complex. The second dataset consists of a single set of scalar sequences, also with long-term structure. We use the Bayesian Information Criterion (BIC) [27] to select an HMM instance that fits to the data. The BIC uses probabilistic arguments to determine the model instance that explains the data well, but without using too many parameters. We then find a smaller LSGM instance that can model the data at par or better than the HMM, demonstrating the ability of the LSGM to scale more efficiently for such datasets.

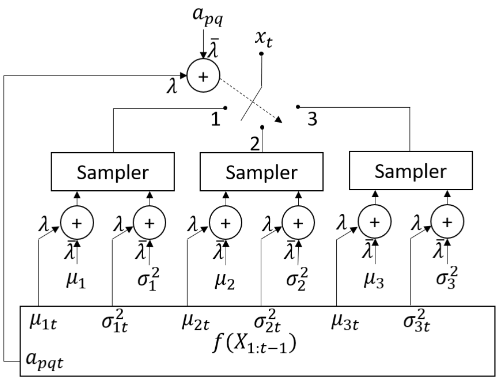


Figure 4: Architecture of the multi-dimensional synthetic data generation system

Figure 4 illustrates the setup for creating the multidimensional synthetic dataset. There are three regimes in the system, each of which produces samples from a Gaussian distribution. A switch chooses one of the three samples at each time-step and outputs it. The goal is to learn the distribution of the outputs without information regarding which regime is chosen at each time-step. Some details are as follows:

- $f(X_{1:t-1})$ is a function of the history of the sequence that can modify the behaviour of the switch and the regimes. We implemented f us-

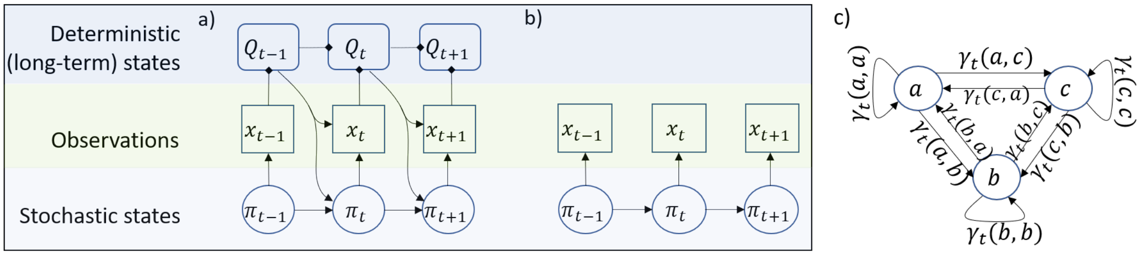


Figure 2: Diamond arrowheads indicate deterministic dependencies, and regular arrowheads indicate stochastic dependencies. (a,b) Generative process in the LSGM, HMM respectively. (c) Graphical model describing temporal relationships in the latent space of a 3-state LSGM. $\gamma_t(p,q)$ are functions of $X_{1:t}$,Φ,p, q.

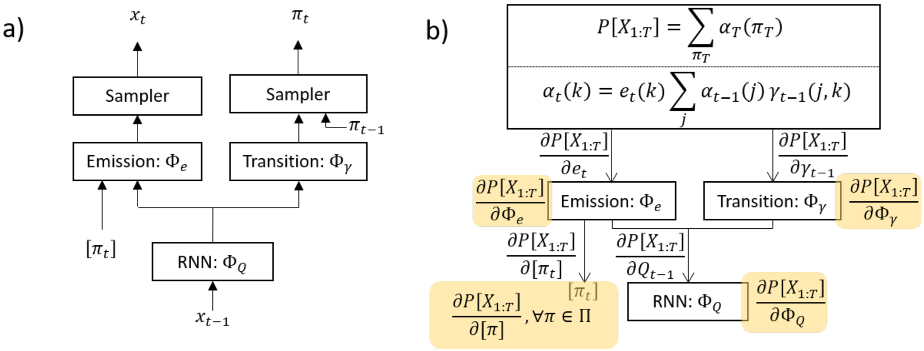

a)
x_t
π_t
Sampler
Sampler
π_{t-1}
Emission: Φ_e
Transition: Φ_γ
$[\pi_t]$
RNN: Φ_Q
x_{t-1}
b)
$P[X_{1:T}] = \sum_{\pi_T} \alpha_T(\pi_T)$
$\alpha_t(k) = e_t(k) \sum_j \alpha_{t-1}(j)\, \gamma_{t-1}(j,k)$
$\frac{\partial P[X_{1:T}]}{\partial e_t}$
$\frac{\partial P[X_{1:T}]}{\partial \gamma_{t-1}}$
$\frac{\partial P[X_{1:T}]}{\partial \Phi_e}$
Emission: Φ_e
Transition: Φ_γ
$\frac{\partial P[X_{1:T}]}{\partial \Phi_\gamma}$
$\frac{\partial P[X_{1:T}]}{\partial [\pi_t]}$
$\frac{\partial P[X_{1:T}]}{\partial Q_{t-1}}$
$[\pi_t]$
$\frac{\partial P[X_{1:T}]}{\partial [\pi]}, \forall \pi \in \Pi$
RNN: Φ_Q
$\frac{\partial P[X_{1:T}]}{\partial \Phi_Q}$

Figure 3: (a) The complete architecture of the LSGM (b) Invoking backpropagation algorithms for Maximum Likelihood estimation through gradient descent; shaded portions indicate derivatives of the sequence likelihood with respect to the parameters of the LSGM

ing randomly initialized GRU

- λ is a mixing factor that determines the impact of long-term history on the system's parameters. $\lambda \in [0,1], \bar{\lambda} = 1 - \lambda$
- The regimes may be described as $\mathcal{N}(\bar{\lambda}\mu_i + \lambda\mu_{it}, \bar{\lambda}\sigma_i^2 + \lambda\sigma_{it}^2)$ for $i \in \{1,2,3\}$. All covariance used is diagonal for multidimensional data.
- $\bar{\lambda}a_{pq} + \lambda a_{pqt}$ is the probability of switching from regime p to regime q at time t

As λ is increased, long-term structure starts manifesting in the data through the function $f(X_{1:t-1})$. The HMM will attempt to model each state's emissions using a static distribution that assumes that emissions at a time-step are conditionally independent of the past, given the state. However, with a full covariance matrix, the HMM does not assume independence among the different dimensions of the data. Temporal structure through f may manifest in the form of local correlations that the HMM can use to improve its predictions. The LSGM does not assume that emissions are conditionally independent of the past. The LSGM implemented for this experiment uses a diagonal covariance matrix, which cannot model local correlations. However, the behavior of the synthetic data arises from temporal structure which is modeled by the LSGM. Hence, we expect the LSGM to be able to capture these dynamics more faithfully.

We generated four-dimensional x_t; f is a single RNN layer with 33 GRU units initialized randomly. We constructed a three-state first-order HMM and a three-state first-order full LSGM. The HMM emissions use non-diagonal covariance matrices. The LSGM uses the emission architecture shown in Figure 1a. Every state gets a separate dense network with eight outputs - four for mean, and four for a diagonal covariance matrix.

For the second dataset consisting of sequences of scalars, we built a second system with three regimes, $0, 1,$ and 2. Regime 0 emits exponentially increasing values, 1 maintains the previous value, and 2 emits exponentially decreasing values. Regime switching is conditional on the sum of a fixed number of past emissions from the system. Details are in the Supplementary document. The HMM and LSGM models used are similar to the ones described for the first dataset.

2.5 Transcription Factor Binding Specificity

It is well known that sequences in the Deoxyribonucleic Acid (DNA) called genes code for proteins. As the first step in protein synthesis, a gene is trascribed into an intermediate molecule called the Ribonucleic Acid (RNA). Transcription is regulated by the presence of other proteins called Transcription Factors (TF) that bind to locations on the DNA called binding sites. TFs have been observed to bind preferentially to sites with certain patterns or motifs. A motif may be visualized through a Position Weight Matrix (PWM), which is a tabulation of the relative frequency of the DNA bases observed at different positions in the binding site. The columns of a PWM may be normalized and interpreted as probabilities (e.g., Figure 5a).

Traditional methods that study TF binding preference build probabilistic models using PWMs [34][33][6][28] that try to reason about the binding process. For instance, HMM-based methods try to determine a binding score for a TF for a given DNA sequence by considering all consistent alignments of a set of known PWMs associated with the TF to the DNA sequence. Recent DNN-based methods such as DeepBind [1] and DeeperBind [16] recognize that 1-D convolutional kernels can represent PWMs and include such layers in the DNN architecture, but do not explicitly reason about the binding process; instead a binding score is derived from a black-box DNN. DeepBind uses convolutional kernels, while DeeperBind uses both convolutional and recurrent layers based on LSTM.

Protein Binding Microarray [20] (PBM) is a sequencing technology that provides measurements reflecting TF binding preference. For a given TF, PBM provides a list of DNA sequences and a score for each sequence, called intensity, reflecting the binding preference of the TF to that sequence (higher intensity implies higher preference). Localization of the specific binding site within each DNA sequence is not provided. Computational methods such as those mentioned above are used to predict PBM intensity from DNA sequence, and are usually trained from available PBM measurements.

We build a probabilistic model for predicting PBM intensity, following traditional methods which explicitly design methods to determine binding score. However, we use the LSGM instead of a traditional model such as the HMM, which allows us to operate without the strong conditional independence assumptions. We believe that the combination of the ability to reason about the binding process, as well as the ability to represent contextual distributions proves to be a strong point for our model in this application.

The LSGM instance in our model has "motif" and "non-motif" states. If a DNA sequence has strong evidence for one or more motif states, then the chances of a TF binding to that sequence are high and the PBM intensity measure will also be high. If a DNA sequence contains no motifs, then it's PBM intensity value will be low. To capture this behaviour, we compare the likelihood of the DNA sequence allowing all states, to the likelihood when motifs are disallowed, and convert the result to a binding score through a simple nonlinear function. Details follow.

LSGM emissions. This is a half-LSGM (Section 2.3) meaning Q_t doesn't influence emissions. There are multiple motif states which use PWMs for emissions, and a single background state that uses a multinomial distribution. A modified forward recursion accounting for motif states that emit symbols of length greater than 1 is given in the Supplementary document.

LSGM transitions. Transition distributions are modeled using an LSTM-based RNN. Transitions are zeroth-order. An alternative view is possible; motif states that emit symbols of length longer than one may be considered a string of states, each emitting length=1 symbols, connected in a linear fashion; in this view, the transitions are first order for these states.

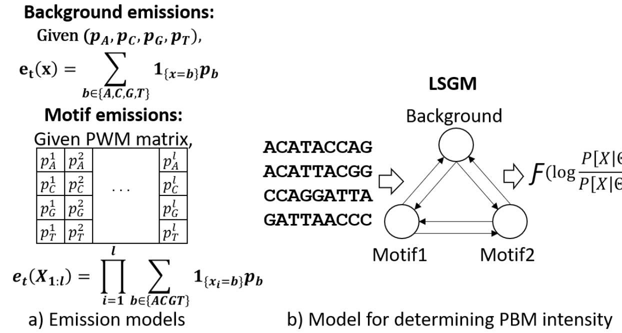
Computing intensity. The likelihood of the sequence allowing all states, $\log P[X|\Theta]$, is computed from the forward recursion. The likelihood allowing only non-motif states, $\log P[X|\Theta']$, is obtained by removing motif states from the same recursion. Θ, Θ' are the hypotheses under which the likelihoods are computed. Then, we compute intensity as,

$$Intensity = \mathcal{F}\big(\log \frac{P(X|\Phi)}{P(X|\Phi')}\big) = \sigma(\eta \log \frac{P(X|\Phi)}{P(X|\Phi')})w + b \tag{13}$$

η, w, b are scalar parameters, and σ is a nonlinear function.

Training. The Mean Square Error (MSE) between the predicted intensity and actual intensity in the training set is minimized through gradient descent.

HMM baseline. We implement an HMM baseline for comparison. The expression in Equation 13 is used to compute intensity and the MSE of the prediction is minimized through gradient descent.


Background emissions:
Given (p_A, p_C, p_G, p_T),
$\mathbf{e}_t(\mathbf{x}) = \sum_{b\in\{A,C,G,T\}} \mathbf{1}_{\{x=b\}} p_b$
Motif emissions:
Given PWM matrix,
p_A^1 p_A^2 p_A^l
p_C^1 p_C^2 p_C^l
p_G^1 p_G^2 p_G^l
p_T^1 p_T^2 p_T^l
...
$e_t(X_{1:l}) = \prod_{i=1}^{l} \sum_{b\in\{ACGT\}} \mathbf{1}_{\{x_i=b\}} p_b$
LSGM
Background
ACATACCAG
ACATTACGG
CCAGGATTA
GATTAACCC
$\mathcal{F}(\log \frac{P[X|\Theta]}{P[X|\Theta']})$
Motif1
Motif2

a) Emission models

b) Model for determining PBM intensity

Figure 5: (a) Emission distributions (b) a DNA sequence is scored using an LSGM with 2 motif states and a background state; the likelihoods are converted to predicted intensity values using the function $\mathcal{F}$

2.6 Generative modeling of music

Polyphonic piano music belonging to four different categories with predefined test, train, validation splits has been previously introduced [5]. The four categories represent different types of music, from folk tunes to a classical music collection. Piano music data is in the form of sequences of frames, where a frame is a list of piano keys pressed at a time step in the sequence. We encode each frame into an 88-dimensional binary vector corresponding to the 88 keys in the piano. Each bit in the vector indicates whether the corresponding piano key is pressed. The goal of this learning task is to train an LSGM model to generate music sequences of these types.

We construct a first-order full LSGM following the architecture in Figure 3a, except for a small modification - a dense network is inserted at the output of the RNN feeding both the Emission and Transition networks. State emissions follow the embedding architecture presented in Figure 1b where the emission layer parameters are shared by all the states of the LSGM. State identities are encoded into multidimensional embeddings. The emission layer provides 88 outputs which are terminated using sigmoid activation, modeling the parameters of 88 independent Bernoulli distributions (independent given the state, and the history).

Training is through Maximum Likelihood estimation. In addition to dropout regularization, we modified the learning objective in the first few epochs of training by adding an additional cost function. The goal of this regularizing cost function is to prevent the LSGM from heavily favouring any state during the first few epochs. Specifically, the regularizing cost is $-\sum_t \sum_{\pi_t} P[\pi_t|X_{1:t}] \log P[\pi_t|X_{1:t}]$. This is the sum of the entropy of the probability distribution of the LSGM state at time-step t given the sequence $X_{1:t}$, over the length of the sequence. Maximizing this quantity pushes the conditional distribution of states towards a uniform distribution, thus preventing the LSGM from favoring any state. This regularizer is easily computed as a by-product of the forward algorithm, which is used to compute the sequence likelihood for training.

3 Experiments

3.1 Synthetic Data

Multidimensional sequences. The HMM and the LSGM instances have 68 and 65 parameters respectively. The means and covariances of the HMM states are initialized from k-means clustering and global covariance (similar to [23]). The means and variances of the LSGM are initialized similarly by setting the bias values in the Emission Network. Additional details are provided in the Supplementary document. The training set has 90000 sequences and validation and test sets have 5000 sequences each. Sequences have length 25. The HMM implementation is from the Pomegranate package [26]. The average log-likelihood per sample (one item in a sequence) in the test set is presented in Table 1. The HMM and LSGM perform identically for the case where long-term structure is not introduced ($\lambda = 0$ case) into the data. When long-term structure is introduced, the LSGM performs better.

Scalar sequences. We generated 1000 sequences of length 25. HMMs with 3 to 60 states were trained on the data. The BIC was computed for each model using $-2\log(L) + k\log(n)$ where L is the maximum likelihood estimate, k the number of free parameters in the model, and n the number of data points. An HMM with 52 states (2807 parameters) came out to be the best, with an average log likelihood of 1.416 per data point. We tried a single LSGM configuration with 10 states and 2 recurrent neurons (284 parameters), and it achieved an average log likelihood of 2.259 per data point. The LSGM instance hence seems to be significantly more efficient on average (approx. 10× fewer parameters). Additional details are in the Supplementary document.

Table 1: Log-likelihood per sample in the test set

Setting	HMM	LSGM
$\lambda = 0.0$	2.70	2.70
$\lambda = 0.25$	-0.58	-0.47
$\lambda = 0.50$	-2.47	-2.29
$\lambda = 1.00$	-4.43	-4.36

3.2 TF-binding specificity

We perform experiments on five TFs, PBM data for which have been published [4]. Experiments on two of these TFs were presented before [16] using two DNN-based methods, DeepBind [1] and DeeperBind[16]. To expand the dataset and also to subject all models to similar training flows, we reimplemented DeepBind (our DeepBind implementation uses both avg and max pooling) and DeeperBind architectures. Our experiments result in a slightly higher accuracy than those presented in [16] for the two TFs presented in that paper. We also built LSGM-based and HMM-based models as outlined in Section 2.5. Architecture search of the implementations is performed; the search space is summarized in the Supplementary document. The largest DeeperBind model has a larger parameter set than the largest LSGM. Following the experiments in [16], we included a configuration with 5 motifs of length 11 for all models. Additionally, for DeepBind we included configurations with 16 motifs and motifs with length 24 following the original publication of DeepBind.

For each TF, two sets of PBM measurements are available, designated array 1 and array 2. Each set contains over 40,000 sequences and associated intensity measurements. Training and validation are performed using array 1 for the different architectures of the four

Table 2: Testset Spearman rank correlation coefficient

TF	HMM	DeepBind	DeeperBind	LSGM
Oct-1	0.48	0.56	0.61	**0.67**
CEH-22	0.27	0.43	0.45	**0.54**
Zif268	0.38	0.45	0.45	**0.50**
Cbf1	0.16	0.14	0.16	**0.23**
Rap1	**0.29**	0.15	0.26	0.22

different models, and the best performing architectures are picked for each model for testing on array 2. Performance is measured in terms of the Spearman rank correlation coefficient between prediction and ground truth intensity. The results are in Table 2.

3.3 Music modeling

We built LSGM instances as described in Section 2.6 and trained them on the four music corpuses. All our models except for JSB use a single layer of 512 GRU units, and 16 states. For JSB, we used a single RNN layer of 128 GRU units and 32 states. Further details are in the Supplementary document.

During the training iteration, the log-likelihood of the training sequences is improved through gradient descent. We applied regularization cost described in Section 2.6 for ten epochs. Early stopping comes into force after the regularizing epochs; if validation accuracy does not improve in three consecutive epochs, training is terminated. The trained models are evaluated on the test set to measure the average log-likelihood per frame per sequence (the higher, the better). We compare our method with others in Table 3. LSGM performs best for Nottingham and Muse datasets, which are of different characteristics: Nottingham is a collection of folk tunes and Muse is a collection of classical music. Hence the LSGM may be a good alternative to other methods for music generation.

Previously, the RNN-NADE [5] had the highest performance in this task. The RNN-NADE makes the observation that notes occur in highly correlated simultaneities. NADEs used in that model are capable of modeling such simultaneities. Our model is a departure from this approach in that, given a state and the history of emissions, the note distributions are independent. The strong performance of our model in two cases provides evidence to suspect that the patterns of such note correlations may be clustered into a discrete set of regimes in some cases. In the two cases where the model does not perform as well, it is likely that such correlations need to be explicitly modeled in each regime just like in the RNN-NADE. However, this will need further study.

Table 3: Performance of models on music dataset

Method	Nottingham	JSB	Muse	Piano-midi
SRNN [9]	≥ -2.94	≥ -4.74	≥ -6.28	≥ -8.20
TSBN [11]	≥ -3.67	≥ -7.48	≥ -6.81	≥ -7.98
NASMC [15]	≈ -2.72	$\approx \mathbf{-3.99}$	≈ -6.89	≈ -7.61
STORN [3]	≈ -2.85	≈ -6.91	≈ -6.16	≈ -7.13
RNN-NADE [5]	≈ -2.31	≈ -5.56	≈ -5.60	$\approx \mathbf{-7.05}$
RNN [5]	≈ -4.46	≈ -8.71	≈ -8.13	≈ -8.37
DMM-Aug [19]	≈ -2.87	≈ -6.69	≈ -5.76	≈ -8.02
LSGM (this work)	$\mathbf{-2.09}$	-5.91	$\mathbf{-5.31}$	-7.96

4 Related Work

In a previous work [32] one of the conditional independence assumptions in the HMM (Equation 8) is removed by using an RNN to model transitions for a single application with discrete data. Compared to that work, we consider models that remove two conditional independent assumptions in the HMM (Equations 7 and 8) as both emissions and transitions are conditioned on the RNN state. Removing the conditional independence of emissions can be a powerful tool in modeling high-dimensional data with complex dependencies across time. We also present ways to implement LSGM instances for continuous, multidimensional data, and evaluate it across multiple applications in supervised and unsupervised learning. HMMs with autoregressive emissions [29] and time-varying transitions [21][17] have been presented before. The autoregression is linear, of fixed window, and the transitions vary according to some simple function, such as a logistic function or a generalized linear model (GLM), of exogenous covariates. In contrast, we implement an explicit internal state modeled using an RNN, which is not limited to looking at a fixed window in the past, and it can represent more complex structure than that implied by simpler functions such as the logistic function or GLMs. There is another work [18], where graphical models that use neural networks to parameterize observation models are adapted from the structure of Linear Dynamical Systems (LDS) and Switching LDS (SLDS). The SLDS includes a discrete latent space which manifests itself through a second continuous state space that also evolves stochastically. Compared to this, our state-space structure is simpler in that the continuous state-space is deterministic, and our model is closer to the HMM than LDS.

5 Conclusions

We present a generative model for sequences that combines a powerful interface suitable for application development using domain knowledge, with the ability to perform exact inferences and represent complex long-term relationships. The model performs strongly in supervised and unsupervised applications in different domains. In the future, we plan to apply the method to bioinformatic tasks at a higher level of abstraction than that presented in this work [24][35][22] where the type of function served by a DNA sequence is important, rather than simply the physical mechanics of its activity such as binding preference. We also plan to tackle more complex data augmenting our model with sophistcated distribution estimators such as NADEs.

Acknowledgements

This material is based upon work supported by the National Science Foundation (NSF) under Grant Nos. CNS 1624790, and CNS 1337732. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

We thank Tanmay Gangwani, Ashok Vardhan Makkuva, and Professors Mark Hasegawa-Johnson, Saurabh Sinha and Pramod Viswanath from the University of Illinois at Urbana-Champaign for helpful discussions.

References

[1] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey. Predicting the sequence specificities of dna- and rna-binding proteins by deep learning. *Nature Biotechnology*, 2015.

[2] P. Arnold, I. Erb, M. Pachkov, N. Molina, and E. van Nimwegen. MotEvo: integrated Bayesian probabilistic methods for inferring regulatory sites and motifs on multiple alignments of DNA sequences. *Bioinformatics*, 2012.

[3] J. Bayer and C. Osendorfer. Learning stochastic recurrent networks. *CoRR*, 2014.

[4] M. F. Berger, A. A. Philippakis, A. M. Qureshi, F. S. He, P. W. E. III, and M. L. Bulyk. Compact, universal DNA microarrays to comprehensively determine transcription-factor binding site specificities. *Nature Biotechnology*, 2006.

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

[6] X. Chen, T. R. Hughes, and Q. Morris. RankMotif++: a motif-search algorithm that accounts for relative ranks of K-mers in binding transcription factors. *Bioinformatics*, 2007.

[7] J. Chung, K. Kastner, L. Dinh, K. Goel, A. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, 2015.

[8] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 1998.

[9] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. Sequential Neural Models with Stochastic Layers. *CoRR*, 2016.

[10] M. Gales and S. Young. The application of hidden markov models in speech recognition. *Found. Trends Signal Process.*, 1(3):195–304, Jan. 2007.

[11] Z. Gan, C. Li, R. Henao, D. Carlson, and L. Carin. Deep Temporal Sigmoid Belief Networks for Sequence Modeling. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, 2015.

[12] A. Goyal, A. Sordoni, M. Côté, N. R. Ke, and Y. Bengio. Z-Forcing: Training Stochastic Recurrent Networks. *NIPS*, 2017.

[13] A. Graves. Generating sequences with recurrent neural networks. *CoRR*, 2013.

[14] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. ICML '06. ACM, 2006.

[15] S. Gu, Z. Ghahramani, and R. E. Turner. Neural adaptive sequential monte carlo. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2629–2637, Cambridge, MA, USA, 2015. MIT Press.

[16] H. R. Hassanzadeh and M. D. Wang. DeeperBind: Enhancing prediction of sequence specificities of DNA binding proteins. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2016.

[17] T. Holsclaw, A. M. Greene, A. W. Robertson, and P. Smyth. Bayesian non-homogeneous markov models via polya-gamma data augmentation with applications to rainfall modeling. *The Annals of Applied Statistics*, 2017.

[18] M. Johnson, D. K. Duvenaud, A. Wiltschko, R. P. Adams, and S. R. Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems 29*. 2016.

[19] R. G. Krishnan, U. Shalit, and D. Sontag. Structured inference networks for nonlinear state space models. *CoRR*, 2016.

[20] Martha L. Bulyk. Protein binding microarrays for the characterization of DNA-protein interactions. *Advances in biochemical engineering/biotechnology*, 2007.

[21] L. Meligkotsidou and P. Dellaportas. Forecasting with non-homogeneous hidden markov models. *Statistics and Computing*, 2011.

[22] X. Min, W. Zeng, S. Chen, N. Chen, T. Chen, and R. Jiang. Predicting enhancers with deep convolutional neural networks. *BMC Bioinformatics*, 2017.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] D. Quang and X. Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *Nucleic Acids Research*, 44(11):e107, 2016.

[25] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 1989.

[26] J. Schreiber. Pomegranate: fast and flexible probabilistic modeling in python. *Journal of Machine Learning Research*, 18(164):1–6, 2018.

[27] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[28] S. Sinha, E. van Nimwegen, and E. D. Siggia. A probabilistic method to detect regulatory modules. *Bioinformatics*, 2003.

[29] S. Srinivasan, T. Ma, D. May, G. Lazarou, and J. Picone. Nonlinear mixture autoregressive hidden markov models for speech recognition. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2008.

[30] I. Sutskever and G. Hinton. Learning multilevel distributed representations for high-dimensional sequences. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, 2007.

[31] I. Sutskever, G. E. Hinton, and G. W. Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems*. 2009.

[32] K. Tran, Y. Bisk, A. Vaswani, D. Marcu, and K. Knight. Unsupervised neural hidden markov models. *CoRR*, 2016.

[33] K.-C. Wong, T.-M. Chan, C. Peng, Y. Li, and Z. Zhang. Dna motif elucidation using belief propagation. *Nucleic Acids Research*, 2013.

[34] Y. Zhao and G. D. Stormo. Quantitative analysis demonstrates most transcription factors require only simple models of specificity. *Nature Biotechnology*, 2011.

[35] J. Zhou and O. G. Troyanskay. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature Methods*, 2015.