Abstract

The tensor network formalism has enjoyed over two decades of success in modeling the behavior of complex quantum-mechanical systems, but has only recently and sporadically been leveraged in machine learning. Here we introduce a uniform matrix product state (u-MPS) model for probabilistic modeling of sequence data. We identify several distinctive features of this recurrent generative model, notably the ability to condition or marginalize sampling on characters at arbitrary locations within a sequence, with no need for approximate sampling methods. Despite the sequential architecture of u-MPS, we show that a recursive evaluation algorithm can be used to parallelize its inference and training, with a string of length $n$ only requiring parallel time $O(\log n)$ to evaluate. Experiments on a context-free language demonstrate a strong capacity to learn grammatical structure from limited data, pointing towards the potential of tensor networks for language modeling applications.

1. Introduction

Learning and reproducing the statistics of natural language is a problem of significant interest in machine learning, which has seen major advances in the last few years coming from the rapid improvement of deep neural network models (LeCun et al., 2015). While much of this progress has arisen from models based on recurrent neural networks (RNNs) (Sutskever et al., 2014; Merity et al., 2018), the inherently sequential nature of RNNs has proved a significant obstacle for training models on large datasets. This was a key motivation for the development of the transformer model (Vaswani et al., 2017), whose ability to parallelize evaluation of input sequences allows for a significant speedup on GPU hardware. Furthermore, the autoregressive nature of standard language models restricts the contextual information they can use in sampling, limiting their usefulness for general sentence completion tasks. For example, sampling a sequence conditioned on a given length from an RNN language model requires approximate sampling methods (e.g., naively using rejection sampling) or specifically training a conditioned RNN model for the task (Ficler & Goldberg, 2017).

In this work, we propose a uniform matrix product state (u-MPS) model specifically suited for probabilistic sequence modeling that offers an efficient parallel evaluation procedure and a flexible sampling algorithm. This model belongs to the family of tensor networks, which have enabled major practical and theoretical advances in quantum many-body physics and have recently been successfully applied in machine learning (Novikov et al., 2015; Cohen et al., 2016; Stoudenmire & Schwab, 2016; Novikov et al., 2017; Han et al., 2018; Stoudenmire, 2018; Cheng et al., 2019). In contrast to previous models built on the fixed-size MPS tensor network, where a model is parameterized by a fixed number of core tensors (Stoudenmire & Schwab, 2016; Han et al., 2018), u-MPS are parameterized by a single core, making them recurrent in nature and able to evaluate and sample strings of arbitrary length.

One of the main appeals of using tensor network models in machine learning resides in the large collection of efficient and tractable optimization and evaluation algorithms developed in the quantum physics and numerical analysis community (see e.g., (Oseledets, 2011; Orús, 2014)). In particular for u-MPS, we leverage such algorithms to design a parallel training algorithm and a flexible sampling algorithm allowing one to condition sequence generation on arbitrary surrounding context. Sampling is carried out exactly, without any recourse to approximate sampling techniques, using an algorithm akin to the forward-backward algorithm for hidden Markov models which permits arbitrary characters in a string to be conditioned on or marginalized over. Because the uniform MPS’s outputs correspond to tensor network diagrams satisfying a general notion of associativity, its evaluation can be parallelized at the level of individual sequences, so that inputs of length $n$ can be evaluated in only $O(\log n)$ computational depth.

To highlight the usefulness and versatility of u-MPS language modeling, we conduct several experiments with syn-
thetic text data. Using strings chosen from context free grammars, we demonstrate the ability of u-MPS to generalize across various length scales and to accurately generate sequences from the grammar conditioned on unseen sequence lengths and arbitrary contexts (e.g., generating well balanced sequences of parenthesis of a given length \( n \) where the sub-sequence “ () (” appears at a given location in the middle of the string). We show that u-MPS significantly outperforms unidirectional and bidirectional LSTM baselines on a subset of tasks for which RNN language models can be used.\(^1\)

1.1. Summary of Contributions

We introduce a uniform MPS model for sequence modeling (Section 3), and show that its convenient mathematical layout permits a highly parallel evaluation method (Section 3.1) and a flexible conditional sampling algorithm (Section 3.2). We explore the performance of the u-MPS at different language modeling tasks using experiments on synthetic text data (Section 4), and show that it is able to make significant generalizations of formal grammatical structure, a first step towards applying the model more broadly for language modeling.

1.2. Related Work

Tensor networks have a long history in quantum many-body physics (Fannes et al., 1992; Orús, 2019), but have only recently been applied to machine learning tasks. These applications include parameter-efficient representations of large neural network weight matrices (Novikov et al., 2015), provable separations in expressivity between deep and shallow networks (Cohen et al., 2016), as well as standalone models for supervised (Stoudenmire & Schwab, 2016; Novikov et al., 2017) and unsupervised (Han et al., 2018; Stoudenmire, 2018; Cheng et al., 2019) learning. Of particular relevance to us are the works of (Han et al., 2018) and (Stokes & Terilla, 2019), where fixed-length MPS are used to learn distributions of fixed-size image and text data. Our sampling algorithm is similar to that described in (Han et al., 2018) (which in turn is based on the perfect sampling algorithm developed in (Ferris & Vidal, 2012)), but by virtue of the recurrent layout of u-MPS, is able to sample sequence data of arbitrary length. The use of positive matrices and completely positive maps in our sampling algorithm is similar to their use in the evaluation of hidden quantum Markov models (Monras et al., 2010; Srinivasan et al., 2018), and likewise admits a convenient interpretation in terms of concepts from quantum mechanics.

u-MPS are closely related to weighted finite automata (WFA), sequential models with close ties to formal language theory which encompass hidden Markov models. In particular, the quadratic weighted automata model proposed in (Bailly, 2011) is equivalent to u-MPS. The potential for flexible sampling and efficient training and parallel evaluation were not investigated in (Bailly, 2011), where a variant of the spectral learning algorithm is used in contrast with the gradient based training algorithm we propose. Our focus is on more practical aspects of u-MPS, with the aim of exploring how the unique properties of tensor network models can complement the landscape of existing deep learning methods for sequence modelling.

u-MPS can also be seen as a particular instance of linear RNNs: u-MPS are second-order RNNs with a linear recurrent activation function and a quadratic output activation function. Linear RNNs have been studied previously for parallelization and interpretability (Martin & Cundy, 2018; Foerster et al., 2017). Connections between second-order RNNs and weighted automata have also been recently studied in (Rabusseau et al., 2019). A key difference with our work is that the flexibility of such linear RNN models for complex sampling tasks was not investigated. To the best of our knowledge, we are the first to show the potential of u-MPS and linear recurrent networks as generative models.

Finally, there have been a number of works from diverse perspectives presenting theoretical applications of tensor networks for language modeling and understanding. This includes (Pestun & Vlassopoulos, 2017), (Pestun et al., 2017), (Coecke et al., 2010), (Gallego & Orús, 2017), and (DeGiuli, 2019) among others. Collectively, these works draw many interesting parallels between the mathematical structure of tensor networks and the grammatical structure of natural language, with (Pestun et al., 2017) in particular proposing an MPS model similar to the u-MPS, although with a different evaluation rule than (7). Our work demonstrate that these fundamental connections can be leveraged to design efficient and flexible new models for language modeling, and we believe that the algebraic nature of tensor networks will prove to be a powerful tool to encode complex syntactic and semantic information in language modeling tasks.

2. Background

We consider sequences over a finite alphabet \( \Sigma \), and use \( \lambda \) to denote the empty string. \( \Sigma^\ast \) refers to the set of all length-\( n \) strings, and \( \Sigma^\ast \) to the set of all strings of finite length. In the context of sampling from a distribution of strings, we will call the locations where characters can be placed as “sites”.

In the following, \( \mathbb{K} \) will denote a field which can be chosen as either \( \mathbb{R} \) or \( \mathbb{C} \), with properties which require a particular choice stated explicitly. \( \|\nu\| \) denotes the 2-norm of a
vector (or higher-order tensor) \( v \), while \( v^\dagger \) is its Hermitian adjoint (conjugate transpose, where \( v^\dagger = v^T \) for \( K = \mathbb{R} \)). We say a matrix \( M \) is Hermitian when \( M^\dagger = M \), a condition which implies that \( M \) can be expressed in some basis as a diagonal matrix of real values. Finally, we use \( \text{Tr}(M) = \sum_{k=1}^D M_{kk} \) to represent the trace of a square matrix \( M \in \mathbb{K}^{D \times D} \).

2.1. Tensors

A tensor \( T \) of shape \((d_1, d_2, \ldots, d_n)\) over a field \( \mathbb{K} \) can be seen as an indexed collection of elements \( T_{i_1, i_2, \ldots, i_n} \in \mathbb{K} \), where each index \( i_k \) ranges over the set \( \{d_k\} := \{1, 2, \ldots, d_k\} \). Tensors with \( n \) indices are said to be \( n \)th order, and the set of \( n \)th order tensors naturally form a vector space of dimension \( d := \prod_{k=1}^n d_k \).

The most well-known families of tensors are vectors (first-order tensors) and matrices (second-order tensors), with the basic operations available on these objects (such as matrix multiplication and inner products) having rich generalizations in the tensor setting. Tensor contraction is one such operation, acting on tensors \( T \) and \( T' \) of shapes \((d_1, \ldots, d_k, \ldots, d_n)\) and \((d'_1, \ldots, d'_k, \ldots, d'_n)\), provided \( d_k = d'_k \) for some \( k, k' \). \( T \) and \( T' \) can then be contracted along their \( k \) and \( k' \) indices to give a product tensor \( T'' \), with elements

\[
T''_{i_1,\ldots,i_{k-1},i_{k+1},\ldots,i_n,i'_{k-1},i'_{k+1},\ldots,i'_{n'}} = \sum_{i_k=1}^{d_k} T_{i_1,\ldots,i_{k-1},i_k,i_{k+1},\ldots,i_n} T'_{i'_1,\ldots,i'_{k-1},i_k,i'_{k+1},\ldots,i'_{n'}}
\]

Although appearing complicated in algebraic form, (1) is in fact a straightforward generalization of matrix multiplication, and can be reasoned about with a simple graphical notation (see Figure 1). In this tensor network notation, individual tensors correspond to nodes, and tensor indices correspond to edges. The contraction of two tensors is indicated by joining their corresponding edges, and (1) gives a computational evaluation procedure yielding a new node containing all the uncontracted edges of \( T \) and \( T' \) (see Figure 1d).

Of great importance is the fact that tensor contraction is associative, meaning that the order of pairwise tensor contractions used to evaluate a tensor network has no impact on its final reduced value. In practice, this property often allows seemingly intractable quantities to be computed efficiently and exactly, by rearranging the order of tensor contractions used in computation.

Although one might think that tensors of high order rarely appear in real-world settings, an important class of examples are joint distributions over \( n \) discrete variables, which can naturally be interpreted as \( n \)th order tensors. In particular, the probability distribution \( P_n \) of a language model conditioned on a fixed length \( n \) is an \( n \)th order tensor whose indices are chosen from a vocabulary \( \Sigma \) of size \( d \). One can thus think of a language model as an infinite family of tensors \( \{P_n\}_{n \in \mathbb{N}} \).

Although language models can be associated with an infinite family of high-order tensors, storing these tensors directly (as in \( n \)-gram models) is infeasible, owing to the exponentially growing number of independent tensor elements. Nonetheless, the tensor network formalism permits an efficient way to parameterize high-order tensors in terms of simpler low-order factors, with the matrix product state being a good example of this method which has proven useful in quantum physics and machine learning (see e.g., (Novikov et al., 2015)).

2.2. Matrix Product States

We first discuss the (fixed-size) matrix product state (MPS) (Perez-Garcia et al., 2007), also known as the tensor train format (Oseledets, 2011), before introducing its recurrent counterpart, the uniform MPS. A fixed-size MPS representation parameterizes an \( n \)th order tensor \( T \) of shape \((d_1, d_2, \ldots, d_n)\) in terms of a collection of \( n \) tensor “cores” \( \{A^{(j)}\}_{j=1}^n \), where \( A^{(1)} \) has shape...
A closely related model is the uniform MPS. This direct interpretation of \( T \) is not without issue though, as determining if a given u-MPS generates \( T \) with non-negative elements for all \( n \) is in fact undecidable (Denis & Esposito, 2008). One way to circumvent this issue is to restrict all entries of \( A, \alpha, \alpha, \omega \) to be non-negative real numbers, an approach which produces models equivalent to hidden Markov models. We instead take a different approach inspired by quantum mechanics, which permits deeper ties with tensor network theory.

2.3. Born Machines

An appealing means of employing tensor networks as generative models was given in (Han et al., 2018), under the name of the Born machine architecture. This construction involves mapping an \( n \)th order tensor \( T_n \) to a probability distribution over length-\( n \) sequences via the rule:

\[
P(s_1, s_2, \ldots, s_n) = |T_n(s_1, s_2, \ldots, s_n)|^2 / Z_n, \tag{3}
\]

where the normalization factor \( Z_n = \|T_n\|^2 \) is the 2-norm of \( T_n \), ensuring that \( P(s_1, s_2, \ldots, s_n) \) sums to 1 over all sequences of length \( n \). The name “Born machine” comes from the fact that (3) is equivalent to the Born rule of quantum mechanics (Born, 1926), giving the tensor \( T_n \) a formal interpretation as the state of an \( n \)-body quantum system. Although computing the normalization factor \( Z_n \) is intractable for arbitrary tensors \( T_n \), we can exactly and efficiently compute it whenever \( T_n \) is parameterized by an MPS. This well-known procedure is described in Section 3, in the context of uniform MPS.

2.4. Positive Matrices and Completely Positive Maps

We say that a matrix \( Q \in \mathbb{K}^{D \times D} \) is positive if it is (a) Hermitian, and (b) satisfies \( v^\dagger Q v \geq 0 \) for every \( v \in \mathbb{K}^D \). For a positive \( Q \), we call \( Q \) positive definite if \( v^\dagger Q v = 0 \) only when \( v = 0 \), and positive semidefinite otherwise. Given a positive matrix \( Q \), the diagonal elements of \( Q \) will necessarily be non-negative. For any vector \( v \), the rank-1 matrix \( vv^\dagger \) is necessarily a positive matrix, and all positive matrices \( Q \in \mathbb{K}^{D \times D} \) can be expressed as the weighted sum of \( (at most) \ D \) such rank-1 matrices. This can be used to show that for any positive matrices \( Q \) and \( Q' \), \( \text{Tr}(QQ') \geq 0 \).

It is common in quantum mechanics to regard positive matrices as probabilistic states (Nielsen & Chuang, 2002), a viewpoint we adopt here. To this end, the family of completely positive (CP) maps is the natural class of operations used to manipulate positive matrices. A map \( \mathcal{E} \) sending positive \( Q \in \mathbb{K}^{D' \times D} \) to \( Q' = \mathcal{E}(Q) \in \mathbb{K}^{D' \times D'} \) is said to be CP if there exist \( r \) matrices \( A_i \in \mathbb{K}^{D' \times D} \) such that \( \mathcal{E} \) takes the form:

\[
\mathcal{E}(Q) = \sum_{i=1}^r A_i Q A_i^\dagger \tag{4}
\]

The condition (4) implies in particular that \( \mathcal{E}(Q) \) is necessarily positive. The matrices in (4) are called the Kraus operators of the map, with \( r \) referred to as its rank. By taking the adjoint of all the Kraus operators appearing in (4),
we obtain a new CP map $\mathcal{E}'$, which is the Hermitian adjoint of $\mathcal{E}$.

For each uniform MPS, there is an associated CP map $\mathcal{E}$ called the transfer operator, which is defined using the $d$ Kraus operators $\{A(i)\}_{i=1}^d$ making up the core tensor $\mathcal{A}$:

$$\mathcal{E}(Q) = \sum_{c=1}^d A(c)QA(c)^\dagger. \quad (5)$$

Each character $c \in \Sigma$ produces a rank-1 CP map $\mathcal{E}_c$, defined by the single Kraus operator $A(c)$, called the conditional transfer operator of $\mathcal{A}$ (relative to $c$):

$$\mathcal{E}_c(Q) = A(c)QA(c)^\dagger. \quad (6)$$

Given that each $\mathcal{E}_c$ is associated with exactly one Kraus operator of $\mathcal{E}$, it is immediate to check that $\sum_{c \in \Sigma} \mathcal{E}_c = \mathcal{E}$, an important fact for the proof of the correctness of the sampling procedure.

### 3. Uniform MPS Born Machines

Our model is obtained as an immediate generalization of the Born machine model from (Han et al., 2018), in which a uniform MPS is used in place of a finite-size MPS. Thus, we modify (3) to give the probability of a sentence $s = s_1 s_2 \ldots s_n$ with the recurrent tensor given in (2), yielding:

$$P_n(s) = \tilde{P}_n(s)/Z_n = |\alpha^\dagger A(s)\omega|^2/Z_n = |\alpha^\dagger A(s_1)A(s_2)\cdots A(s_n)\omega|^2/Z_n. \quad (7)$$

The normalization factor $Z_n = \sum_{s \in \Sigma^n} \tilde{P}_n(s)$ can be computed exactly, by expressing the sum as a ladder-shaped tensor network and rearranging the order of contraction as shown in Figure 2e. By first contracting the vertical indices (i.e., summing over outcomes $s_j$) before contracting the horizontal indices, this expression can be evaluated in terms of the $n$th power of the CP map $\mathcal{E}$ associated with the MPS core, as:

$$Z_n = \alpha^\dagger \mathcal{E}^n(\omega^\dagger)\alpha = \text{Tr}(\alpha^\dagger \mathcal{E}^n(\omega^\dagger)). \quad (8)$$

Applying the sequence of CP maps in (8) takes $\mathcal{O}(ndD^3)$ time, and produces the exact normalization factor $Z_n$ needed to properly normalize the length-$n$ sampling distribution. In the limit of large $n$, the CP map $\mathcal{E}^n$ can be approximated arbitrarily well in terms of spectral properties of $\mathcal{E}$, via the use of so-called “infinite” methods (e.g., (McCulloch, 2008)). Although we do not utilize them here, we mention that infinite methods can be used within generative modeling for (a) sampling (bi-)infinite streams of text with a u-MPS (described in Appendix B), and (b) circumventing the sequential bottleneck associated with computing $Z_n$ for large values of $n$.
3.1. Evaluation

To obtain the unnormalized probability \( \hat{P}(s) \) of a sentence \( s \) within the distribution \( P_n \) defined by the u-MPS, we must evaluate the length-\( n \) product of matrices and vectors in the right hand side of (7) using the characters in \( s \). This can be carried out in one of several different ways.

One choice for evaluating this product is with iterated vector-matrix multiplications, producing a sequence of row vectors \( v_0, v_1, \ldots, v_n \), where \( v_{j+1} = v_j A(j) \) and \( v_0 := \alpha^1 \). This evaluation method is similar to that used in vanilla RNNs, and requires \( O(nD^2) \) time to carry out. However, the dependence of each hidden state \( v_j \) on the previous \( v_{j-1} \) makes this method inherently sequential, meaning that even with unlimited parallel computing resources, the evaluation time is still lower-bounded by \( \Omega(n) \).

By contrast, an alternate means of evaluation uses hierarchical matrix-matrix multiplications to reduce all the \( \mathcal{A}(s_j) \) into a single transition matrix \( \mathcal{A}(s) \), at which point the unnormalized probability is given by \( |\alpha^1 \mathcal{A}(s) \omega|^2 \). This proceeds in a recursive fashion on the \( \{\mathcal{A}(s_j)\}_{j=1}^n \), with all pairs of neighboring matrices \( (\mathcal{A}_{2k}, \mathcal{A}_{2k+1}) \) multiplied together to yield \( [\gamma/2] \) product matrices, followed by all \( [\gamma/4] \) pairs of those products, and so forth. After \( \lceil \log_2 n \rceil \) repetitions of this iterated multiplication, the resultant product matrix \( \mathcal{A}(s) \) is obtained (see Figure 3).

This recursive evaluation strategy has a total computational cost of \( \mathcal{O}(nD^3) \), a factor of \( D \) increase compared to sequential evaluation. In return, the ability to parallelize the matrix-matrix multiplications means that the parallel evaluation time of this method is only \( \mathcal{O}(\log(n)D^3) \). The ideal choice of evaluation method thus depends on the size of the model and length of the strings being trained on, with parallel evaluation becoming increasingly attractive with models of smaller size acting on longer strings.

The fact that probabilities computed by u-MPS can be expressed as a tensor network and contracted in arbitrary order is the primary factor enabling our parallel evaluation method. It is worth mentioning that if we had used nonlinear activation functions within the hidden state of u-MPS (as is typical in RNNs), this convenient parallelization property would be lost. For a discussion of these issues for first-order RNNs, see (Martin & Cundy, 2018).

3.2. Sampling

Sampling with the u-MPS is straightforward, and following a flexible procedure permitting arbitrary sites in the random output string to be either sampled (producing a random symbol \( s_j \)), marginalized (left unsampled), or conditioned on taking a given value \( c_j \). In general, the cost of this sampling procedure when generating a string of \( n \) sites using a u-MPS with vocabulary size \( d \) and bond dimension \( D \) is \( \mathcal{O}(ndD^3) \).

For simplicity, we will describe the sampling procedure for a distribution of strings with fixed length \( n \), in a manner which broadly parallels that of (Ferris & Vidal, 2012; Han et al., 2018). However, slight modifications of this procedure make it possible to sample strings of random (finite) length, as well as (bi-)infinite character streams from a given u-MPS. These variations of the fixed-length sampling algorithm are discussed in Appendix B.

The sampling procedure relies on an atomic operation for drawing a single character from a u-MPS, given a specification of two positive “context” matrices \( Q_L \) and \( Q_R \). The probability assigned to a character \( c \) by this atomic operation is given by

\[
P(c) = \frac{\text{Tr}(Q_L E_c (Q_R))}{Z},
\]

where \( Z = \text{Tr}(Q_L E(Q_R)) \) is a normalization factor.

We can think of the context matrices \( Q_L \) and \( Q_R \) as hidden states, which contain the model’s entire collection of knowledge about sites to the left and right of the sampled site. The following theorem (whose proof is given in Appendix A) shows how the context matrices are constructed from the parameters of a u-MPS to account for any combinations of marginalized and conditioned sites to the left and right of the sampling site.

**Theorem 1.** Let \( C \subset [n] \) be a set of conditioned positions, let \( (c_j)_{j \in C} \in \Sigma^{|C|} \) be the corresponding set of conditioning symbols and let \( i \) be a sampling position not in \( C \).

Then, given a u-MPS \( (A, \alpha, \omega) \), the probability of the \( i \)th symbol being \( s_i \), conditioned on the symbols \( (c_j)_{j \in C} \) appearing in the corresponding positions in a sequence of length \( n \) is given by

\[
P_n(s_i | (c_j)_{j \in C}) = \frac{\text{Tr}(Q_L E_{s_i}(Q_R))}{Z},
\]

where \( Z = \text{Tr}(Q_L E(Q_R)) \) and the left and right context matrices are defined by

\[
Q_L = \mathcal{E}_{i-1}^t \cdots \mathcal{E}_{i-2}^t \mathcal{E}_i^t (\alpha \alpha^\dagger), \quad Q_R = \mathcal{E}_{i+1} \cdots \mathcal{E}_n (\omega \omega^\dagger)
\]

with

\[
\mathcal{E}_j = \begin{cases} 
\mathcal{E}_{c_j} & \text{if } j \in C \\
\mathcal{E} & \text{otherwise}
\end{cases}
\]

for all \( j \), where \( \mathcal{E} \) and \( \mathcal{E}_c \) are the transfer operator and conditional transfer operator of \( A \) defined in Eq. (5) and (6).

By generating a collection of left and right context matrices \( \{Q_L^{(j)}, Q_R^{(j)}\}_{j=1}^n \) for the sites we wish to sample, we can use the single-character sampling operation defined by (10) (referred to as \texttt{drawSamp}(A, Q_L^{(j)}, Q_R^{(j)})), to obtain all the samples needed.
The left and right sampling contexts are generated in a two-process.

In the first sweep, a sequence of CP maps is applied to the context matrices to draw a random sample when conditioning on the occurrence of a character or when it is conditioned on. When a sampling site is sampled characters start with random characters \( s \) and a left-to-right sweep then sampling characters \( s \), and a left-to-right sweep then sampling characters \( s \) while conditionally evolving the left context to \( Q_L^{(j+1)} \).

In the first sweep, a sequence of CP maps is applied to the leftward-propagating context matrix \( Q_R^{(j)} \), starting with \( Q_R^{(n)} = \omega \alpha^1 \). Different CP maps are used for different sites, with the choice of map depending on whether the corresponding site is being conditioned on or marginalized over. When conditioning on the occurrence of a character \( c_j \), the CP map \( E_{c_j} \) is applied to \( Q_R^{(j)} \), otherwise \( E \) is applied. In both cases, the resultant sequence of contexts \( Q_R^{(n)}, Q_R^{(n-1)}, \ldots, Q_R^{(1)} \) is saved for the second sweep.

In the second sweep, we similarly proceed one site at a time, but now with random characters \( s_j \) being drawn at each sampling site. Starting with \( Q_L^{(1)} = \alpha \alpha^1 \), we move \( Q_L^{(j)} \) rightward via the map \( E \) when site \( j \) is marginalized over, or \( E_{c_j} \), when it is conditioned on. When a sampling site is reached, the function \( \text{drawSamp} \) (relying on Eq. (10)) is used with the context matrices to draw a random sample \( s_j \), after which \( Q_L^{(j)} \) is conditionally evolved using \( E_{c_j} \). Finishing the second left-to-right sweep completes the sampling process.

The overall sampling procedure is summarized in Algorithm 1. The proof that this process indeed samples from the appropriately conditioned and marginalized distribution (relative to the induced distribution given in Eq. (7)) is a simple corollary of Theorem 1 and the following factorization of the joint distribution over sampled sites \( S \) conditioned on conditioning sites \( C \):

\[
P_n((s_i)_{i \in S} | (c_j)_{j \in C}) = \prod_{i \in S} P_n (s_i | (c_j)_{j \in C}, (s_j)_{j \in S, j < i})
\]

4. Experiments

To assess the capacity of u-MPS for learning distributions of structured text, we carry out experiments on synthetic strings from a context-free language. These experiments involve training on sequences of a fixed length, then assessing the performance at (a) predicting single missing characters in unseen sequences, and (b) generating new sequences with no prior context. In both cases, we vary the length of sequences used for assessment, and measure the fraction of sampled outputs which produce valid strings in the language. We use a bidirectional LSTM as a baseline for (a) and a unidirectional LSTM for (b), where the number of hidden units equals the bond dimension of the u-MPS.

While MPS are traditionally trained using the density matrix renormalization group (DMRG) procedure, we train the MPS model using stochastic gradient descent (SGD) relative to a negative log likelihood (NLL) loss on the probability \( P_n(s_1, s_2, \ldots, s_n) \) from (7), with gradients computed via automatic differentiation. In our experiments we take the bond dimension to be \( D = 50 \), and use early stopping relative to the NLL on a separate validation set to determine the end of training.

The context-free language we use is defined over a three symbol vocabulary \( \Sigma_M = \{ (, *, ) \} \), and consists of all strings in \( \Sigma_M^* \) whose parentheses are properly balanced, with no constraints made on the placement of the * characters. For example, "** ( (*) *) (*)" and "** ( *) * ( *" are contained in the language, while "* ( *) * ( *)") and "* ( *) * ( *" are not. We will call strings in this language “Motzkin strings”, owing to their correspondence with Motzkin walks (e.g., see (Alexander et al., 2018)).

For the character completion task, we train on Motzkin strings of length \( n_{\text{train}} = 15 \), before feeding the MPS strings of varying lengths to use for conditional sampling. For each string, Algorithm 1 is used to sample a character \( s_j \) conditioned on knowledge of all other characters. \(^3\)Note that while u-MPS are able to perform significantly more complex sampling tasks (i.e., conditioning on an any number of arbitrary locations), these two simple tasks were chosen because they allow for direct comparison with RNN models.
Table 1. Character completion experiment with synthetic Motzkin strings. After training on strings of length $n_{\text{train}} = 15$, we use Algorithm 1 to conditionally sample at each site of a set of Motzkin strings of length $n_{\text{eval}}$, given knowledge of all other characters in the string. We use a single-layer bidirectional LSTM as a baseline, with the number of hidden states in its forward and backwards LSTMs each equal to the bond dimension of the u-MPS, $D = 50$. We conduct the experiments with $N_{\text{train}} = 1000$ and $N_{\text{train}} = 10000$ Motzkin strings in the training data.

| $n_{\text{eval}}$ | $N_{\text{train}}$ | u-MPS % | BiLSTM % |
|------------------|----------------------|--------|----------|
| 10               | 10k                  | 99.9   | 98.3     |
| 15               | 10k                  | 99.6   | 99.8     |
| 30               | 10k                  | 93.0   | 49.9     |
| 50               | 10k                  | 84.0   | 39.5     |

| $n_{\text{eval}}$ | $N_{\text{train}}$ | u-MPS % | BiLSTM % |
|------------------|----------------------|--------|----------|
| 10               | 1k                   | 95.8   | 89.1     |
| 15               | 1k                   | 87.4   | 92.7     |
| 30               | 1k                   | 75.5   | 55.0     |
| 50               | 1k                   | 63.0   | 42.1     |

Table 2. Unconditional sampling experiment with synthetic Motzkin strings. After training on strings of length $n_{\text{train}} = 15$, we use Algorithm 1 to unconditionally sample new strings of length $n_{\text{samp}}$. For each sample length, we report the percentage of output samples which are valid Motzkin strings, and compare the performance of the u-MPS to a baseline unidirectional LSTM model. While the MPS generalizes well to different lengths, samples from the LSTM tend to consist of Motzkin strings of length $n_{\text{train}}$, followed by a stream of ungrammatical text.

| $n_{\text{samp}}$ | $N_{\text{train}}$ | u-MPS % | LSTM % |
|-------------------|---------------------|--------|--------|
| 1                 | 10k                 | 87.2   | 38.5   |
| 10                | 10k                 | 99.6   | 13.1   |
| 15                | 10k                 | 99.1   | 80.9   |
| 50                | 10k                 | 67.6   | 0.0    |

$c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_n$. This sampling process is carried out for all values of $j \in [n]$, with the overall percentage of correctly\(^4\) sampled characters given in Table 1.

While both models manage to easily predict the missing character in strings of equal length to those used for training, we find the u-MPS model better able to correctly extend its predictions to strings of different lengths. This is especially true in the presence of additional training data, which causes the u-MPS’s accuracy to increase at all tested string lengths, whereas the BiLSTM ends up generalizing worse in this setting.

For the unconditional sampling task, we again train a u-MPS on fixed-length Motzkin strings, but now use the trained model for unconditional sequence generation at different lengths. Given the difficulty of sampling streams of text from a bidirectional model, we use a single-layer unidirectional LSTM as a baseline, where strings are produced in an autoregressive fashion. The results are reported in Table 2. We find the MPS capable of correctly generalizing the rules of the context-free grammar given only fixed-length training data. Even at lengths over 3 times that used for training, we find that the model produces valid strings the majority of the time. Equally surprising is the model’s ability to infer the correct distribution of Motzkin strings of length $n_{\text{samp}} = 1$, in which the only allowed string is the character $\ast$.

In contrast, the sequences output by the LSTM quickly degenerates into a sequence of $)$ and $\ast$ characters, as seen in the randomly chosen sample $\ast((*)(())***)***)***)$ of length

\[^{4}\text{Given knowledge of all other characters in a Motzkin string, each site has exactly one character } s_j \in \Sigma_M \text{ which can make it a valid Motzkin string.}\]

\[^{5}\text{While the initial substring of length } n_{\text{train}} = 15 \text{ (in bold) does indeed belong to the language, the sampled characters after this point appear to be an incorrect generalization based on characters frequently appearing at the end of training sequences. This is in contrast to output from the MPS, such as the randomly chosen sample } \ast((*)(*)(())((*)())())(\ast)\ast), \text{ which do not exhibit such differences.}\]

5. Discussion

We introduced a u-MPS model for probabilistic modeling of sequences of arbitrary length. We show that this model permits a flexible sampling procedure, in which arbitrary regions of a sequence can be sampled in the presence of arbitrary conditional context in other regions of the sequence. Furthermore, we present a parallel evaluation procedure which enables the model to parallelize its computation within individual inputs, allowing it to overcome the sequential evaluation bottleneck of standard RNNs when training on long sequences. We carry out experiments on synthetic text data, and find that u-MPS achieve remarkable accuracy in generalizing to text of significantly different length than was present in the training data.

While we used standard results from the tensor network formalism to prove our results, there is still much that remains to explore. Of particular interest is the use of tensor networks to “tensorize” high-dimensional linear algebraic operations, which was shown in (Novikov et al., 2015) to yield significant improvements in the parameter efficiency and runtime of existing neural network models. When applying this tensorization process to a u-MPS model (which is itself a tensor network), these techniques are even more
powerful, and would enable the efficient manipulation of hidden states of exponentially large dimension\(^5\). We expect this technique to be crucial in scaling up the model’s capacity and performance, giving additional promise to the use of these models for competitive language modeling applications.

**Acknowledgements**

Jacob would like to thank Shawn Tan for helpful discussions regarding language modeling. This research is supported by the Canadian Institute for Advanced Research (CIFAR AI chair program) and the MITACS Accelerate International program.

**References**

Alexander, R. N., Evenbly, G., and Klich, I. Exact holographic tensor networks for the motzkin spin chain. *arXiv:1806.09626*, 2018.

Bailly, R. Quadratic weighted automata: Spectral algorithm and likelihood maximization. In *Asian Conference on Machine Learning*, pp. 147–163, 2011.

Bulle, B., Panangaden, P., and Precup, D. Singular value automata and approximate minimization. *Mathematical Structures in Computer Science*, 86(1):1–35, 2019.

Berglund, M., Raiko, T., Honkala, M., Kärkkäinen, L., Vetek, A., and Karhunen, J. T. Bidirectional recurrent neural networks as generative models. In *Advances in Neural Information Processing Systems*, pp. 856–864, 2015.

Born, M. Quantenmechanik der stoßvorgänge. *Zeitschrift für Physik*, 38(11-12):803–827, 1926.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., and Wanderman-Milne, S. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Cheng, S., Wang, L., Xiang, T., and Zhang, P. Tree tensor networks for generative modeling. *Physical Review B*, 99(15):155131, 2019.

Coecke, B., Sadrzadeh, M., and Clark, S. Mathematical foundations for a compositional distributional model of meaning. *arXiv:1003.4394*, 2010.

Cohen, N., Sharir, O., and Shashua, A. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory (CoLT)*, pp. 698–728, 2016.

De las Cuevas, G., Cirac, J. I., Schuch, N., and Perez-García, D. Irreducible forms of matrix product states: Theory and applications. *Journal of Mathematical Physics*, 58(12):121901, 2017.

DeGiuli, E. Random language model. *Physical Review Letters*, 122:128301, 2019.

Denis, F. and Esposito, Y. On rational stochastic languages. *Fundamenta Informaticae*, 86(1):41–47, 2008.

Evans, D. E. and Høegh-Krohn, R. Spectral properties of positive maps on C*-algebras. *Journal of the London Mathematical Society*, 2(2):345–355, 1978.

Fannes, M., Nachtergaele, B., and Werner, R. F. Finite correlation states on quantum spin chains. *Communications in mathematical physics*, 144(3):443–490, 1992.

Ferris, A. J. and Vidal, G. Perfect sampling with unitary tensor networks. *Physical Review B*, 85(16):165146, 2012.

Ficler, J. and Goldberg, Y. Controlling linguistic style aspects in neural language generation. In *Proceedings of the Workshop on Stylistic Variation*, pp. 94–104, 2017.

Foerster, J. N., Gilmer, J., Sohl-Dickstein, J., Chorowski, J., and Sussillo, D. Input switched affine networks: an rnn architecture designed for interpretability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1136–1145, 2017.

Gallego, A. and Orús, R. Language design as information renormalization. *arXiv:1708.01525*, 2017.

Glasser, I., Pancotti, N., and Cirac, J. I. Supervised learning with generalized tensor networks. *arXiv:1806.05964*, 2018.

Han, Z.-Y., Wang, J., Fan, H., Wang, L., and Zhang, P. Unsupervised generative modeling using matrix product states. *Physical Review X*, 8(3):031012, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, 2015.

Martin, E. and Cundy, C. Parallelizing linear recurrent neural nets over sequence length. In *Conference on Learning Theory (CoLT)*, 2018.

McCulloch, I. P. Infinite size density matrix renormalization group, revisited. *arXiv:0804.2509*, 2008.

Merit, S., Shirish Keskar, N., and Socher, R. Regularizing and optimizing lstm language models. In *International Conference on Learning Representations (ICLR)*, 2018.

---

\(^5\)In tensor network language, this amounts to using a projected entangled pair state (PEPS) (Verstraete et al., 2006) in place of a uniform MPS.
Monras, A., Beige, A., and Wiesner, K. Hidden quantum markov models and non-adaptive read-out of many-body states. *arXiv:1002.2337*, 2010.

Nielsen, M. A. and Chuang, I. L. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

Novikov, A., Trofimov, M., and Oseledets, I. Exponential machines. In *International Conference on Learning Representations (ICLR)*, 2017.

Orús, R. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

Orús, R. Tensor networks for complex quantum systems. *Nature Reviews Physics*, 1(9):538–550, 2019.

Oseledets, I. V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Perez-García, D., Verstraete, F., Wolf, M. M., and Cirac, J. I. Matrix product state representations. *Quantum Information and Computation*, 7(5-6):401–430, 2007.

Pestun, V. and Vlassopoulos, Y. Tensor network language model. *arXiv:1710.10248*, 2017.

Pestun, V., Terilla, J., and Vlassopoulos, Y. Language as a matrix product state. *arXiv:1711.01416*, 2017.

Rabusseau, G., Li, T., and Precup, D. Connecting weighted automata and recurrent neural networks through spectral learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

Srinivasan, S., Gordon, G., and Boots, B. Learning hidden quantum markov models. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

Stokes, J. and Terilla, J. Probabilistic modeling with matrix product states. *Entropy*, 21(12), 2019.

Stoudenmire, E. and Schwab, D. J. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, pp. 4799–4807, 2016.

Stoudenmire, E. M. Learning relevant features of data with multi-scale tensor networks. *Quantum Science and Technology*, 3(3):034003, 2018.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Verstraete, F., Wolf, M. M., Perez-Garcia, D., and Cirac, J. I. Criticality, the area law, and the computational power of projected entangled pair states. *Physical review letters*, 96(22):220601, 2006.
A. Proof of Theorem 1

Theorem. Let $C \subset [n]$ be a set of conditioned positions, let $(c_j)_{j \in C} \in \Sigma^{|C|}$ be the corresponding set of conditioning symbols and let $i$ be a sampling position not in $C$.

Then, given a u-MPS $(\mathcal{A}, \alpha, \omega)$, the probability of the $i$th symbol being $s_i$ conditioned on the symbols $(c_j)_{j \in C}$ appearing in the corresponding positions in a sequence of length $n$ is given by

$$P_n(s_i \mid (c_j)_{j \in C}) = \frac{\text{Tr}(Q_L \mathcal{E}_{s_i} (Q_R))}{Z},$$

(12)

where $Z = \text{Tr}(Q_L \mathcal{E}(Q_R))$ and the left and right context matrices are defined by

$$Q_L = \mathcal{E}_{i-1}^\dagger \mathcal{E}_{i-2}^\dagger \cdots \mathcal{E}_1^\dagger (\alpha \alpha^\dagger), \quad Q_R = \mathcal{E}_{i+1} \cdots \mathcal{E}_n (\omega \omega^\dagger)$$

with

$$\mathcal{E}_j = \begin{cases} \mathcal{E}_{c_j} & \text{if } j \in C \\ \mathcal{E} & \text{otherwise} \end{cases}$$

for all $j$, where $\mathcal{E}$ and $\mathcal{E}_c$ are the transfer operator and conditional transfer operator of $\mathcal{A}$ defined in Eq. (5) and (6).

Proof. Let $M = [n] \setminus (C \cup \{i\}) := \{k_1, \cdots, k_d\}$ be the set of positions that are marginalized. We have

$$P_n(s_i, (c_j)_{j \in C}) = \sum_{c_{k_1} \in \Sigma} \cdots \sum_{c_{k_d} \in \Sigma} P_n(c_1, \cdots, c_{i-1}, s_i, c_{i+1}, \cdots, c_n)$$

$$= \sum_{c_{k_1} \in \Sigma} \cdots \sum_{c_{k_d} \in \Sigma} \text{Tr}(\alpha \alpha^\dagger \mathcal{E}_{c_1} \cdots \mathcal{E}_{c_{i-1}} \mathcal{E}_{s_i} \mathcal{E}_{c_{i+1}} \cdots \mathcal{E}_n (\omega \omega^\dagger))/Z_n$$

$$= \text{Tr}(\alpha \alpha^\dagger \mathcal{E}_1 \cdots \mathcal{E}_{i-1} \mathcal{E}_{s_i} \mathcal{E}_{i+1} \cdots \mathcal{E}_n (\omega \omega^\dagger))/Z_n$$

where the linearity of the trace and the fact that $\sum_{c_{k_j}} \mathcal{E}_{c_{k_j}} = \mathcal{E}$ for any $j \in M$ is used in the last equality. Using the identity $\text{Tr}(Q \mathcal{E}' (Q')) = \text{Tr}(\mathcal{E} (Q) \mathcal{E}' (Q'))$, which holds for arbitrary matrices $Q, Q'$ and CP maps $\mathcal{E}, \mathcal{E}'$, it follows that

$$P_n(s_i, (c_j)_{j \in C}) = \frac{\text{Tr}(Q_L \mathcal{E}_{s_i} (Q_R))}{Z_n}$$

and, consequently,

$$P_n(s_i \mid (c_j)_{j \in C}) = \frac{P_n(s_i, (c_j)_{j \in C})}{P_n((c_j)_{j \in C})} \propto P_n(s_i, (c_j)_{j \in C}) \propto \text{Tr}(Q_L \mathcal{E}_{s_i} (Q_R)).$$

\[ \square \]

B. Alternate Sampling Methods for the u-MPS

In Section 3.2, Algorithm 1 was given as a means of sampling strings with pre-determined length $n$ from a u-MPS model. The restriction to fixed-length samples is not fundamental however, as we show here using standard constructions employed in the study of MPS and WFA. In particular, we describe how a u-MPS can be used to sample either (a) infinite streams of text, with new characters conditioned on previously sampled characters, or (b) strings of a random (finite) length. These sampling procedures somewhat resemble those used in autoregressive RNN language models (with case (b) employing the equivalent of beginning/end of sequence tokens), but have the novel feature that sampling can also proceed backwards, with a random prefix being sampled based on all characters following it.

For simplicity, we will only discuss the case of unconditional sampling, where no sites are reserved for conditioning or marginalization. This restriction is not fundamental however, and the same techniques used in Algorithm 1 to handle conditioning and marginalization can be adapted to both forms of sampling given below.
B.1. Stream Sampling

Sampling long streams of text in a left-to-right manner can be seen as the application of Algorithm 1 in the limit \( n \to \infty \), where the end of string boundary has moved infinitely far to the right. Algorithm 1 can’t be used directly in this limit, as we would have to produce an infinite number of right context matrices in the right-to-left sweep before sampling, but a simple analysis of the update rule used in this sweep shows a way of proceeding.

We show in the following how the right context matrices \( Q_R^{(j)} \) generated in the initial sweep of Algorithm 1 generically end up converging to a fixed point \( \hat{Q}_R^{(*)} \), so that for any fixed \( k \), the equality \( Q_R^{(1)} = Q_R^{(2)} = \ldots = Q_R^{(k)} = Q_R^{(*)} \) holds to arbitrary precision in the \( n \to \infty \) limit. This gives a means of performing stream sampling with a u-MPS, by first identifying a right fixed point \( Q_R^{(*)} \), and then iterating over the left-to-right sweep in Algorithm 1 with the choice of right context matrix \( Q_R^{(j)} = Q_R^{(*)} \). This idea gives a general method to perform stream sampling, with the only remaining ingredients being a proof of existence and means of calculating such fixed points.

The basic rule used in Algorithm 1 to obtain new right context matrices from old ones is:\(^6\)

\[
Q_R \mapsto \hat{E}(Q_R) := E(Q_R)/\text{Tr}(E(Q_R)).
\] (13)

The properties of \( \hat{E} \) are closely related to the eigenvalues and eigenmatrices\(^7\) of \( E \), and the CP nature of the latter is quite useful. We say that an orthogonal projection matrix \( P \) is an invariant subspace of a CP map \( E \) when \( E(PMP) = P E(M) P \) for all matrices \( M \). CP maps are called irreducible when their only invariant subspaces are the trivial projectors \( P = 0, I_D \). Irreducible CP maps admit a convenient characterization in terms of a generalized Perron-Frobenius theorem (Evans & Høegh-Krohn, 1978):

**Theorem 2.** For any irreducible CP map \( E \), there exists a positive matrix \( Q_* \) associated with a positive eigenvalue \( \lambda_* \) such that:

- \( Q_* \) is the unique eigenmatrix with eigenvalue \( \lambda_* \).
- \( \lambda_* = \rho(\hat{E}) \), the spectral radius of \( E \) (i.e. it is an eigenvalue of maximum norm).
- \( Q_* \) is positive definite.

Although general CP maps \( E \) won’t be irreducible, by identifying their invariant subspaces \( \{ P^{(*)i} \}_{i=1}^r \), we can recursively express any \( \hat{E} \) as a direct sum of \( r \) irreducible CP maps \( \{ E^{(*)i} \}_{i=1}^r \), for some \( r \) with \( 1 \leq r \leq D \). Each irreducible map is associated with a positive eigenvalue \( \lambda^{(*)i} \), and the eigenmatrices whose eigenvalues equal the spectral radius of \( E \) (\( \rho(\hat{E}) = \max \rho(\hat{E}^{(*)i}) \)), are called the dominant eigenmatrices of \( E \). It is clear that under the repeated action of \( \hat{E} \), any weighted sum of the eigenmatrices \( Q^{(*)i}_R \) associated to the irreducible maps composing \( \hat{E} \) will converge to a corresponding sum containing only the maximal eigenmatrices. In other words, the space of maximal eigenmatrices are precisely the generic fixed points of \( \hat{E} \), and at least one such eigenmatrix exists for any CP map \( E \).

Given this proof of existence, fixed points \( Q_R^{(*)} \) of \( \hat{E} \) can generally be obtained by simply repeating the action \( Q_R \mapsto \hat{E}(Q_R) \) until convergence, starting from the initial positive matrix \( \omega \). In theory, this simple process of repeating \( \hat{E} \) can end up converging to an oscillatory sequence of positive matrices with period \( m > 1 \), a phenomenon arising from the existence of irreducible CP maps whose spectrum contains all eigenvalues of the form \( \{ \lambda_* \exp \left( \frac{2\pi ik}{m} \right) \}_{k=1}^m \). More information about such “non-primitive” CP maps can be found in (Evans & Høegh-Krohn, 1978; De las Cuevas et al., 2017).

In practice however, such oscillatory behavior isn’t an issue. Simply averaging many hidden states \( Q_R \) occurring at the end of the above iteration is sufficient to produce a genuine fixed-point, as is the use of a sparse eigensolver to directly compute the positive matrix \( Q_R^{(*)} \) associated with the largest positive eigenvalue of \( E \). More interestingly, periodic sequences of right context matrices can be used for stream sampling, in place of the constant sequence \( Q_R^{(*)} = Q_R^{(*)} \), a situation which can permit the generation of long streams of text exhibiting periodic behavior.

B.2. Random-Length Sampling

In contrast to both fixed-length sampling and infinite stream sampling, which sample strings of known (potentially infinite) length, random-length sampling produces samples from a probability distribution \( P(s) \) over all strings \( s \in \Sigma^* \). In order to

\(^6\)We restrict the inputs of \( \hat{E} \) to be positive matrices with unit trace, a restriction which still allows for a linearly independent basis of matrices to be chosen.

\(^7\)Given that CP maps are linear mappings between spaces of matrices, we refer to their eigenvectors as eigenmatrices.
normalize the distribution $P(s)$, we must have the sum $Z := \sum_{s \in \Sigma^*} P(s)$ converge to a finite value, a condition which holds generically when the u-MPS’s transfer operator $\mathcal{E}$ has a spectral radius $\rho(\mathcal{E}) < 1$. Note that for fixed-length (as well as infinite stream) sampling, the spectral radius has no impact on the sampled distribution, with any rescaling of the u-MPS core tensor being exactly cancelled out by the normalization factor $Z_n$, appearing in (7). By performing such a rescaling, we can therefore assume without loss of generality that $\rho(\mathcal{E}) < 1$, with no impact on the u-MPS’s other modes of sampling.

For random-length sampling, the probability $P(s)$ of obtaining a string $s$ is

$$P(s) = \frac{\tilde{P}_n(s)}{Z} = \frac{|\alpha^1 A(s)\omega|^2}{Z},$$  \hspace{1cm} (14)$$

where the normalization factor $Z$ can be expressed as

$$Z = \sum_{s \in \Sigma^*} \frac{\tilde{P}_n(s)}{Z} = \sum_{n=0}^{\infty} \sum_{s \in \Sigma^n} \tilde{P}_n(s) = \sum_{n=0}^{\infty} Z_n$$

$$= \sum_{n=0}^{\infty} \text{Tr}(\alpha \mathcal{E}^n(\omega^\dagger)) = \text{Tr} \left( \alpha \mathcal{E}^\dagger \sum_{n=0}^{\infty} \mathcal{E}^n(\omega^\dagger) \right)$$

$$= \lim_{n \to \infty} \text{Tr} \left( \alpha \mathcal{E}^\dagger \mathcal{E}^n(\omega^\dagger) \right) = \text{Tr} \left( \alpha \mathcal{E}^\dagger \tilde{Q}_R \right).$$  \hspace{1cm} (15)$$

We use $\hat{\mathcal{E}}$ to denote the affine map $\hat{\mathcal{E}} : Q_R \mapsto \mathcal{E}(Q_R) + \omega^\dagger$, and $\tilde{Q}_R$ to denote the positive matrix $\tilde{Q}_R = \lim_{n \to \infty} \hat{\mathcal{E}}^n(\omega^\dagger)$ which is a fixed point of $\hat{\mathcal{E}}$. The fact that $\rho(\mathcal{E}) < 1$ implies that the limit appearing in (15) exists and is unique (Balle et al., 2019), in which case simply iterating the action of $\hat{\mathcal{E}}$ starting from $\omega^\dagger$ is guaranteed to produce the fixed point $\tilde{Q}_R$.

With $\tilde{Q}_R$ in hand, sampling proceeds by a left-to-right sweep where left context matrices $Q^{(j)}_L$ are used to produce new samples, starting with $Q^{(0)}_L = \alpha^1$. In each step of this sweep we use $\tilde{Q}_R$ and $Q^{(j)}_L$ to sample a new character $s_{j+1} \in \Sigma \cup \{\sigma_{EOS}\}$, where $\sigma_{EOS}$ denotes a new end of sequence character. The probability of obtaining $s_{j+1}$ is then

$$P(s_{j+1}) = \begin{cases} \frac{\text{Tr}(Q^{(j)}_L \mathcal{E}_{s_{j+1}}(\tilde{Q}_R))/Z}{\text{Tr}(Q^{(j)}_L \omega^\dagger)/Z}, & \text{if } s_{j+1} \in \Sigma, \\ \text{if } s_{j+1} = \sigma_{EOS}, \end{cases}$$  \hspace{1cm} (16)$$

where the normalization factor $Z$ is $Z = \text{Tr}(Q^{(j)}_L \tilde{Q}_R) = \omega^\dagger + \sum_{i \in \Sigma} \mathcal{E}_i(\tilde{Q}_R) = \tilde{Q}_R$. If our outcome $s_{j+1}$ belongs to $\Sigma$, then we evolve the left context matrix as $Q^{(j+1)}_L = \mathcal{E}_{s_{j+1}}(Q^{(j)}_L)/\text{Tr}(\mathcal{E}_{s_{j+1}}(Q^{(j)}_L))$ and continue sampling as in the left-to-right sweep of Algorithm 1. If $s_{j+1} = \sigma_{EOS}$ then we terminate sampling and return $s = s_1 \cdots s_j$ as our randomly sampled string.

Note that random length sampling with a u-MPS allows the empty string to be sampled, an event which occurs with probability $P(\lambda) = |\alpha^1 \omega|^2/Z$. Additionally, the spectral norm of $\mathcal{E}$ can be adjusted by multiplying the core tensor $A$ with a scalar, allowing the expected length of output sentences to be tuned freely.

**B.3. Sampling Backwards**

Although Algorithm 1 describes an initial right-to-left sweep to produce right context matrices, followed by a left-to-right sweep to generate samples, this ordering is not necessary to sample from the distribution defined by a u-MPS. For example, we could have chosen the algorithm to consist of an initial left-to-right sweep to produce left context matrices, followed by a right-to-left sweep to sample new characters. The fact that both procedures produce samples from the same distribution can be seen by applying Theorem 1 to the reverse-ordered unfolding of the conditional distribution

$$P_n((s_i)_{i \in S} \mid (c_j)_{j \in C}) = \prod_{i \in S} P_n(s_i \mid (c_j)_{j \in C}, (s_j)_{j \in S, j > i}),$$  \hspace{1cm} (17)$$

which holds for any disjoint subsets $S, C \subset [n]$ (cf. (11)). Given that the distribution sampled by the operation $\text{giveSamp}$ is manifestly invariant under the interchange of $\alpha$ and $\omega$, along with all CP maps by their adjoints, the unfolding of (17) gives an algorithm which samples strings from right to left, but yields the same distribution as Algorithm 1.
This same left/right equivalence can be applied to the sampling procedures described in Sections B.1 and B.2. In the latter case, as with fixed-length sampling, such a reversal in direction results in samples which are drawn from the same sampling distribution. In the case of infinite stream sampling however, this permits infinite right-to-left streams of text, as well as bi-infinite streams, where new characters can be sampled at the beginning or end of the growing sampled string.

C. Experiment Details

For the experiments, the u-MPS model was written in JAX (Bradbury et al., 2018), while the LSTM module from PyTorch (Paszke et al., 2017) was used for the baselines. The LSTMs were single-layer models with 50 hidden units (50 in each direction for the bidirectional LSTM), and a linear decoder and softmax output layer used to obtain character probabilities. The bond dimension of the u-MPS was 50. In both experiments, models were trained for 100 epochs of gradient descent, with a negative log likelihood (NLL) loss and Adam optimizer (Kingma & Ba, 2015). A learning rate of $10^{-3}$ was used, with early stopping based on the loss on a validation set determining the time at which the models’ correct sampling fraction was evaluated.

The u-MPS was trained identically for both experiments, with a loss given by the NLL of training strings of length 15. The unidirectional LSTM was trained the same way, while the bidirectional LSTM was trained specifically for the conditional sampling task. In particular, the loss was taken as a sum of the NLL of the correct character at each site of the training strings, given knowledge of all characters on the other sites.

For the unconditional generation task, we first tried training the LSTM on strings with a final EOS token appended and using rejection sampling to obtain samples of arbitrary length. This was unable to generate significantly longer strings though, so we instead sampled streams of text from the LSTM and took the first $n_{samp}$ characters for each sample, with the initial hidden state of the LSTM treated as a trainable parameter.

The Motzkin strings were obtained by generating all strings in the context-free language of length 15, then choosing a random subset. The code for both experiments can be found in the supplementary material.

---

$^8$The decomposition of $\tilde{E}$ into irreducible CP maps utilized in Section B.1 proceeds exactly the same with the adjoint map $\tilde{E}^\dagger$, and the eigenvalue spectra (but not the eigenmatrix $Q_\perp$) of both maps are identical.