Tensor Networks for Probabilistic Sequence Modeling

Jacob Miller
Mila and DIRO
Université de Montréal
jmjacobmiller@gmail.com

Guillaume Rabusseau
CCAI chair - Mila and DIRO
Université de Montréal
grabus@iro.umontreal.ca

John Terilla
CUNY and Tunnel
City University of New York
jterilla@gc.cuny.edu

Abstract

Tensor networks are a powerful modeling framework developed for computational many-body physics, which have only recently been applied within machine learning. In this work we utilize a uniform matrix product state (u-MPS) model for probabilistic modeling of sequence data. We first show that u-MPS enable sequence-level parallelism, with length-n sequences able to be evaluated in depth $O(\log n)$. We then introduce a novel generative algorithm giving trained u-MPS the ability to efficiently sample from a wide variety of conditional distributions, each one defined by a regular expression. Special cases of this algorithm correspond to autoregressive and fill-in-the-blank sampling, but more complex regular expressions permit the generation of richly structured data in a manner that has no direct analogue in neural generative models. Experiments on sequence modeling with synthetic and real text data show u-MPS outperforming a variety of baselines and effectively generalizing their predictions in the presence of limited data.

1 Introduction

Tensor network models have long represented the state of the art in modeling complex quantum systems (White, 1992; Fannes et al., 1992; Orús, 2019), but have only recently been utilized as models for machine learning (Novikov et al., 2015; Cohen et al., 2016; Stoudenmire and Schwab, 2016; Novikov et al., 2017; Han et al., 2018; Stoudenmire, 2018; Cheng et al., 2019). In contrast to neural networks, tensor networks forgo the use of nonlinear activation functions, relying instead on multiplicative interactions to capture complex correlations within data. This gives tensor networks a convenient mathematical structure suitable for proving general theoretical results, such as the separation in expressivity between almost all deep tensor networks and their shallow counterparts (Cohen et al., 2016). However, these distinctive mathematical properties have yet to be leveraged for the development of new *operational* abilities, which would give more practical reasons for the wider adoption of tensor network models in real-world machine learning tasks.

In this work we apply a recurrent tensor network, the *uniform matrix product state* (u-MPS), to the task of probabilistic sequence modeling, and identify several novel abilities of u-MPS regarding their evaluation and generative capabilities. Despite its recurrent nature, we show that sequential inputs to u-MPS can be processed in a highly parallel manner, with sequences of length n being evaluated in parallel time $\mathcal{O}(\log n)$. While the difficulty of parallelizing deep recurrent neural networks (RNNs) has previously motivated the development of non-recurrent architectures for sequence processing tasks (e.g. (Gehring et al., 2017; Vaswani et al., 2017)), our finding shows that recurrent tensor networks represent another means of achieving greater parallelism.

We further show that u-MPS models are endowed with surprising generative capabilities closely tied to the structure of regular expressions (regex). While standard autoregressive models are constrained to generate sequences in a stream-like fashion conditioned on some prompt, we find that u-MPS can sample from a wide variety of distributions defined by conditioning regular expressions R. Our sampling algorithm efficiently produces unbiased samples from the probability distribution learned by the u-MPS, conditioned on the output sequence matching a given regular expression R. Standard autoregressive sampling follows from the choice $R = p\Sigma^*$ (for p a prefix string and Σ^* the regex

Proceedings of the 24[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130. Copyright 2021 by the author(s).

matching all sequences), but other special cases include fill-in-the-blank sampling ($R = p\Sigma^* s$, for suffix s), as well as the generation of samples constrained to contain some target phrase t ($R = \Sigma^* t \Sigma^*$).

Besides permitting the generation of sequences with rich internal structure, these techniques enable novel forms of regularization, where a u-MPS model can be penalized or incentivized during training to generate strings matching some target pattern. Such regularization can be applied in a variety of challenging tasks, including automatic code generation and mitigating gender bias in language models. Experiments on synthetic and real structured text datasets confirm these novel parallelism, sampling, and regularization benefits, and show u-MPS able to successfully generalize nonlocal correlations present in small strings to sequences of significantly greater length.

Summary of Contributions We give the first implementation of a u-MPS for probabilistic sequence modeling, and uncover several remarkable capabilities of this model[1]. The absence of nonlinear activation functions in the u-MPS allows us to utilize a parallel evaluation method during training and inference. We develop new techniques linking the structure of u-MPS "transfer operators" to that of regular expressions, which in turn enables a flexible recursive sampling algorithm and novel forms of regularization for the u-MPS. We expect these techniques to open up significant new research directions in the design of sequential generative models, with language modeling being a particularly promising domain.

Related Work Notable previous applications of tensor networks in machine learning include compressing large neural network weights (Novikov et al., 2015), proving separations in the expressivity of deep vs shallow networks (Cohen et al., 2016), and for supervised (Stoudenmire and Schwab, 2016; Novikov et al., 2017; Glasser et al., 2018) and unsupervised (Han et al., 2018; Stoudenmire, 2018; Cheng et al., 2019) learning tasks. Of particular relevance is (Stokes and Terilla, 2019), where (non-uniform) MPS were trained as generative models for fixed-length binary sequences using the density matrix renormalization group (DMRG) algorithm. A diverse range of tensor network architectures have also been proposed as theoretical tools for modeling and understanding natural language, such as (Pestun and Vlassopoulos, 2017; Coecke et al., 2010; Gallego and Orús, 2017; DeGiuli, 2019). The completely positive maps employed in our sampling algorithm are similar to those used in hidden quantum Markov models (HQMM) (Monras et al., 2010; Srinivasan et al., 2018), and can likewise be interpreted using concepts from quantum information theory.

This work can be seen as a continuation of (Pestun et al., 2017), where u-MPS were introduced from a theoretical perspective as a language model, but without the parallelization, sampling, or experimental results given here. Our sampling algorithm is a significant generalization of the fixed-length Born machine algorithm introduced in (Han et al., 2018) (which in turn follows that of (Ferris and Vidal, 2012)), and by virtue of the recurrent nature of u-MPS, permits the generation of discrete sequences of arbitrary length. The u-MPS model is equivalent to quadratic weighted finite automata (Bailly, 2011) and (to a lesser extent) the norm-observable operator model (NOOM) (Zhao and Jaeger, 2010), and is also an example of a linear (second-order) RNN (Rabusseau et al., 2019). Benefits of linear (first-order) RNNs for parallelization and interpretability were described in (Martin and Cundy, 2018; Foerster et al., 2017).

A key difference from previous works is the general techniques developed for evaluating and sampling from regex-structured probability distributions, which to the best of our knowledge are completely new. These techniques apply not only to u-MPS, but also to a broad family of models with similar internal structure, such as weighted finite automata (WFA) (Droste et al., 2009), hidden Markov models (HMM) (Rabiner and Juang, 1986), and predictive state representations (Littman and Sutton, 2002). We consequently expect the algorithms developed here for regex sampling, regularization, and parallel evaluation to immediately generalize to any of these models which parameterize valid probability distributions.

2 Background

We consider sequences over a finite alphabet Σ, with Σ^n denoting the set of all length-n strings, Σ^* the set of all strings, and ε the empty string. We use $\|v\|$ to denote the 2-norm of a vector, matrix, or higher-order tensor v, and $\mathrm{Tr}(M) = \sum_{i=1}^{D} M_{ii}$ to denote the trace of a square matrix $M \in \mathbb{R}^{D \times D}$.

A real-valued[2] tensor $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_n}$ is said to have shape $(d_1, d_2, \ldots, d_n)$, and can be specified by an indexed collection of elements $\mathcal{T}_{i_1, i_2, \ldots, i_n} \in \mathbb{R}$, where each index $i_k \in [d_k] := \{1, 2, \ldots, d_k\}$. Tensors with n indices are said to be nth order, and the set of nth order tensors form a vector space of dimension $\Pi_{k=1}^{n} d_k$.

[1] The code used to produce our results can be found at `https://github.com/jemisjoky/umps_code`.

[2] The restriction to real-valued tensors is natural for machine learning, but differs from the standard in quantum physics of using complex parameters. The results given here carry over to the complex setting, and only require the replacement of some tensors by their complex conjugate.

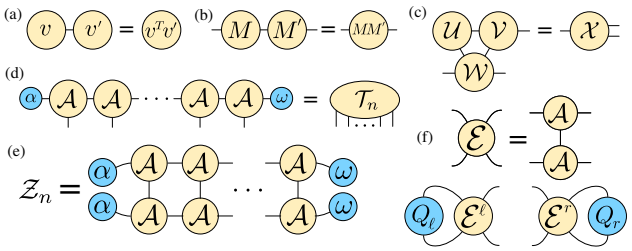

Figure 1: (a-b) Two well-known cases of tensor contractions, inner products of vectors and matrix multiplication. (c) A simple tensor network, where 2nd, 3rd, and 4th order tensors are contracted to form a 3rd order tensor. In numerical libraries, small tensor contractions can be computed with the `einsum` function, and the output $\mathcal{X}$ is independent of contraction order. (d) The u-MPS model, which uses a core tensor $\mathcal{A}$ of shape (D, d, D) and D-dimensional vectors α and ω to define tensors of arbitrary order. (e) The length-n normalization factor $\mathcal{Z}_n$ defined by Equation 3, expressed as a network of tensor contractions. (f) The 4th order tensor $\mathcal{E}$ defined by two copies of the u-MPS core tensor $\mathcal{A}$. The contraction of $\mathcal{E}$ with a matrix on the left or right gives the left and right *transfer operators* of the u-MPS, linear maps which allow the efficient computation of $\mathcal{Z}_n$ via Equation 4.

Matrices, vectors, and scalars are the simplest examples of tensors, of 2nd, 1st, and 0th order, respectively. Tensor contraction is a generalization of both matrix multiplication and vector inner product, and multiplies two tensors along a pair of indices with equal dimension. If the tensors $\mathcal{T}$ and $\mathcal{T}'$ have respective shapes $(d_1, \ldots, d_k, \ldots, d_n)$ and $(d'_1, \ldots, d'_{k'}, \ldots, d'_{n'})$, for $d_k = d'_{k'}$, then the contraction of the k and k' indices gives a product tensor $\mathcal{T}''$, described by elements

$$T''_{i_1,\ldots,i_{k-1},i_{k+1},\ldots,i_n,i'_1,\ldots,i'_{k'-1},i'_{k'+1},\ldots,i'_{n'}} = \sum_{i_k=1}^{d_k} \mathcal{T}_{i_1,\ldots,i_k,\ldots,i_n} \mathcal{T}'_{i'_1,\ldots,i_k,\ldots,i'_{n'}}. \quad (1)$$

The contraction operation Equation 1 is more easily understood with a convenient graphical notation (see Figure 1), where individual tensors correspond to nodes in an undirected graph, and edges describe contractions to be performed. Contracting along an index corresponds to merging two connected nodes, to produce a new node whose outgoing edges are the union of those in the tensors being contracted. An important property of tensor contraction is its generalized associativity, so that a network of tensors can be contracted in any order, with the final product tensor being the same in every case.

A natural example of an nth order tensor is a probability distribution over length-n sequences Σ^n, where the probabilities associated with all possible sequences form the $|\Sigma|^n$ separate tensor elements. This exponential growth in the number of elements makes dense representations of higher order tensors infeasible, but convenient tensor decompositions frequently permit the efficient manipulation of tensors with high order, even into the thousands.

The fixed-size matrix product state (Perez-García et al., 2007) (MPS, also known as tensor train (Oseledets, 2011)) model parameterizes an nth order tensor $\mathcal{T}$ with shape $(d_1, d_2, \ldots, d_n)$ as a sequential contraction of n independent tensor "cores" $\{\mathcal{A}^{(j)}\}_{j=1}^n$, which form the parameters of the model. Each $\mathcal{A}^{(j)}$ has shape (D_{j-1}, d_j, D_j), where $D_0 = D_n = 1$. The dimensions D_j are referred to as bond dimensions (or ranks) of the MPS, and by choosing the D_j to be sufficiently large, it is possible to exactly represent any nth order tensor.

3 Uniform MPS

In this work we utilize the *uniform MPS* (u-MPS) model, a recurrent tensor network obtained by choosing all cores of an MPS to be identical tensors $\mathcal{A}^{(j)} = \mathcal{A}$ with shape (D, d, D). To obtain scalar tensor elements, D-dimensional vectors α and ω are used as "boundary conditions" to terminate the initial and final bond dimensions of the network. In contrast to fixed-length MPS, the recurrent nature of u-MPS allows the generation of nth order tensors $\mathcal{T}_n \in \mathbb{R}^{d^n}$ for any $n \in \mathbb{N}$, which in turn allows u-MPS to be applied in problems involving sequential data.

For discrete sequences over an alphabet Σ of size d, a u-MPS (paired with a bijection $\varphi : \Sigma \to [d]$) can be used to map a sequence of arbitrary length-n to the index of an nth order tensor $\mathcal{T}_n$, defining a scalar-valued function $f_{\mathcal{A}}$ over sequences. Using $\mathcal{A}(c) = \mathcal{A}_{:,\varphi(c),:} \in \mathbb{R}^{D \times D}$ to denote the matrix associated with the character $c \in \Sigma$, a u-MPS acts on a sequence $s = s_1 s_2 \cdots s_n \in \Sigma^n$ as

$$f_{\mathcal{A}}(s) = \alpha^T \mathcal{A}(s_1)\mathcal{A}(s_2)\cdots\mathcal{A}(s_n)\omega = \alpha^T \mathcal{A}(s)\omega, \quad (2)$$

where we use $\mathcal{A}(s) := \mathcal{A}(s_1)\mathcal{A}(s_2)\cdots\mathcal{A}(s_n)$ to denote the matrix product appearing in Equation 2. The function $\mathcal{A}(s)$ can be seen as a matrix-valued representation of arbitrary sequences $s \in \Sigma^*$, and is *compositional* in the sense that st is represented by the product of representations $\mathcal{A}(s)$ and $\mathcal{A}(t)$.

While u-MPS are clearly laid out as a sequential model, the evaluation of $f_{\mathcal{A}}(s)$ for $|s| = n$ can be parallelized by evaluating Equation 2 using $\lceil \log_2(n) \rceil$ batched matrix-

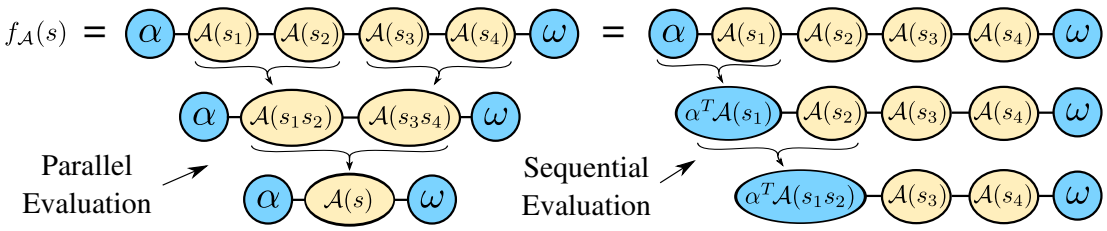

Figure 2: Illustration of parallel and sequential evaluation of $f_{\mathcal{A}}(s)$ when $|s| = 4$, where $f_{\mathcal{A}}(s) = (\mathcal{T}_4)_{i_1,i_2,i_3,i_4}$, an element of the 4th order tensor defined by a u-MPS. After obtaining the matrix representations $\mathcal{A}(s_1), \ldots, \mathcal{A}(s_n)$ from s, parallel evaluation involves repeated batch multiplications of nearest-neighbor pairs of matrices, with the boundary vectors α and ω only incorporated after the matrix product $\mathcal{A}(s)$ has been obtained. Sequential evaluation instead uses iterated matrix-vector multiplications starting with a boundary vector to contract this product. Parallel and sequential evaluation have respective costs of $\mathcal{O}(nD^3)$ and $\mathcal{O}(nD^2)$, but the former can be carried out in $\mathcal{O}(\log n)$ parallel time. The mathematical equivalence of these evaluation strategies is a basic example of the associativity of tensor contractions, allowing an appropriate method to be chosen based on the size of the model, the problem at hand, and the availability of hardware acceleration.

matrix multiplications on all nearest-neighbor pairs of matrices, as shown in Figure 2. This form of parallelization requires the absence of nonlinear activation functions in the evaluation, and can also be carried out in linear RNNs (Martin and Cundy, 2018).

3.1 Born Machines

While Equation 2 is identical to the evaluation rule for WFA, and well-suited for regression tasks, we are interested in using u-MPS as probabilistic models. This requires the interpretation of $f_{\mathcal{A}}(s)$ as a non-negative probability $P(s)$, and deciding if a general WFA outputs negative values is undecidable (Denis and Esposito, 2008). This issue can be circumvented by requiring all entries of $\mathcal{A}$, α, and ω to be non-negative real numbers, but such models can be seen as largely equivalent to hidden Markov models (Denis and Esposito, 2008).

We instead follow the approach introduced in (Pestun et al., 2017) (see also (Han et al., 2018)), which is inspired by the typical usage of MPS in quantum mechanics. For the case of u-MPS, this *Born machine* approach converts a scalar value $f_{\mathcal{A}}(s)$ to an unnormalized probability $\tilde{P}(s) := |f_{\mathcal{A}}(s)|^2$. This can be converted into a properly normalized distribution over sequence of fixed length n by choosing $P_n(s) = \tilde{P}(s)/\mathcal{Z}_n$, where the normalization function $\mathcal{Z}_n$ is given by

$$\begin{aligned} \mathcal{Z}_n &= \sum_{s\in\Sigma_n} \tilde{P}(s) = \sum_{i_1\in[d]} \sum_{i_2\in[d]} \cdots \sum_{i_n\in[d]} |(T_n)_{i_1,i_2,\ldots,i_d}|^2 \\ &= \|\mathcal{T}_n\|^2 , \end{aligned} \tag{3}$$

and with $\mathcal{T}_n$ the nth order tensor defined by the u-MPS. This quadratic evaluation rule is equivalent to the Born rule of quantum mechanics (Born, 1926), which gives a formal interpretation of such models as wavefunctions over n quantum spins. However this probabilistic correspondence is richer in the case of u-MPS, since distributions over sequences of different lengths can be easily defined. The distribution $P(s) = \tilde{P}(s)/\mathcal{Z}_*$ in particular gives a probability distribution over strings of arbitrary length, where the normalization factor $\mathcal{Z}_*$ is identical to that given in Equation 3, but with the sum over Σ^n replaced by one over Σ^*. We show in Section 4 how normalization functions of this form can be generalized further to incorporate sums over all strings matching an arbitrary regular expression R.

Normalization functions like $\mathcal{Z}_n$ occur frequently in many-body physics, and can be efficiently computed via a simple reordering of tensor contractions. By Equation 3, $\mathcal{Z}_n$ equals the 2-norm of $\mathcal{T}_n$, which is represented diagrammatically as Figure 1e. The naive method of evaluating $\mathcal{Z}_n$ involves first generating all elements of $\mathcal{T}_n$ via contraction along the horizontal D-dimensional indices of the u-MPS, but the generalized associativity of tensor contraction lets us evaluate this expression more efficiently.

By first contracting two copies of $\mathcal{A}$ along a vertical d-dimensional index (see Equation 1f) we obtain a 4th order tensor $\mathcal{E}$, which can be interpreted as a linear map on a space of matrices in two main ways, by contracting either its left or its right indices with an input. These linear maps, known as *transfer operators*, are examples of completely positive (CP) maps, a generalization of stochastic matrices which find frequent application in the context of quantum information theory (see the supplementary material for more details). These maps admit the Kraus representations $\mathcal{E}^r(Q_r) = \sum_{c\in\Sigma} \mathcal{A}(c) Q_r \mathcal{A}(c)^T$ and $\mathcal{E}^\ell(Q_\ell) = \sum_{c\in\Sigma} \mathcal{A}(c)^T Q_\ell \mathcal{A}(c)$, which are connected by the adjoint identity $\mathrm{Tr}(Q_\ell \mathcal{E}^r(Q_r)) = \mathrm{Tr}(\mathcal{E}^\ell(Q_\ell) Q_r)$.[3]

[3] In general, CP maps are linear operators $\mathcal{F}$ acting on square matrices by the rule $\mathcal{F}(Q) = \sum_{i=1}^K A_i Q A_i^T$. CP maps are guaranteed to send positive semidefinite (PSD) to other PSD matrices, allowing us to assume in the following

Table 1: Dictionary giving the correspondence between regular expressions (regex) and generalized transfer operators associated with a u-MPS (note the reversal of order in $\mathcal{E}^{\ell}_{R_1R_2}$). The infinite sum giving $\mathcal{E}^{r}_{S^*}(Q_r)$ can be efficiently approximated by partial sums, or else computed as the solution Q_r^* to the linear equation $(I - \mathcal{E}^r_S)Q_r^* = Q_r$ (similarly for $\mathcal{E}^{\ell}_{S^*}(Q_\ell)$).

Regex R	$\boldsymbol{\mathcal{E}^r_R(Q_r)}$	$\boldsymbol{\mathcal{E}^\ell_R(Q_\ell)}$
s	$\mathcal{A}_s Q_r \mathcal{A}_s^T$	$\mathcal{A}_s^T Q_\ell \mathcal{A}_s$
$\mathbf{R_1R_2}$	$\mathcal{E}^r_{R_1}(\mathcal{E}^r_{R_2}(Q_r))$	$\mathcal{E}^\ell_{R_2}(\mathcal{E}^\ell_{R_1}(Q_\ell))$
$\mathbf{R_1 \vert R_2}$	$\mathcal{E}^r_{R_1}(Q_r) + \mathcal{E}^r_{R_2}(Q_r)$	$\mathcal{E}^\ell_{R_1}(Q_\ell) + \mathcal{E}^\ell_{R_2}(Q_\ell)$
$\mathbf{S}^*$	$\sum_{n=0}^{\infty}(\mathcal{E}^r_S)^{\circ n}(Q_r)$	$\sum_{n=0}^{\infty}(\mathcal{E}^\ell_S)^{\circ n}(Q_\ell)$

The normalization $\mathcal{Z}_n$ can be equivalently computed in terms of left or right transfer operators, with the latter option yielding

$$\begin{aligned}\mathcal{Z}_n &= \alpha^T \mathcal{E}^r(\mathcal{E}^r(\cdots \mathcal{E}^r(\omega\omega^T))\cdots)\alpha \\ &= \mathrm{Tr}\left(Q_\ell^\alpha \ (\mathcal{E}^r)^{\circ n}(Q_r^\omega)\right), \end{aligned} \tag{4}$$

where $Q_\ell^\alpha = \alpha\alpha^T$ and $Q_r^\omega = \omega\omega^T$ are rank-1 matrices constituting boundary conditions for the normalization term. We use $(\mathcal{E}^r)^{\circ n}$ to denote the composition of $\mathcal{E}^r$ with itself n times, and define $(\mathcal{E}^r)^{\circ 0}$ to be the identity map acting on square matrices. For an MPS of bond dimension D over an alphabet of size d, a single transfer operator application requires time $\mathcal{O}(dD^3)$, giving a sequential runtime of $\mathcal{O}(ndD^3)$ for computing $\mathcal{Z}_n$. By representing transfer operators as $D^2 \times D^2$ matrices, this computation can be parallelized in a similar manner as described in Section 3, but at the price of increasing the total computational cost to $\mathcal{O}(nD^6)$.

The distribution $P(s) = \tilde{P}(s)/\mathcal{Z}_*$ over all strings $s \in \Sigma^*$ plays an important role in the following, but to employ this we must ensure the infinite summation $\mathcal{Z}_* = \sum_{s\in\Sigma^*}\tilde{P}(s)$ does in fact converge. This convergence can be guaranteed by rescaling the core tensor to a new $\mathcal{A}' = \gamma\mathcal{A}$, for any scalar γ satisfying $0 < |\gamma| < \sqrt{\rho(\mathcal{E}^r)}$, where $\rho(\mathcal{E}^r)$ is the spectral radius of $\mathcal{E}^r$. Such rescaling leaves all fixed-length distributions $P_n(s)$ invariant, while introducing a bias towards shorter ($|\gamma| < 1$) or longer ($|\gamma| > 1$) strings in $P(s)$.

4 Regular Expressions and u-MPS

While transfer operators as defined above are standard in quantum many-body physics, we now show how this transfer operator calculus can be richly generalized in the setting of sequential data. We work with regular expressions (regex) R over an alphabet Σ of size d, which can be recursively defined in terms of: (a) String literals $s \in \Sigma^*$, (b) Concatenations of regex $R = R_1R_2$, (c) Unions of regex $R = R_1|R_2$, and (d) Kleene closures of regex $R = S^*$. We use Σ to denote the union regex of all single characters $c \in \Sigma$, and Σ^n to denote the concatenation of Σ with itself n times.

Any regex R defines a set $\mathrm{Lang}(R) \subset \Sigma^*$, the language of strings matching the pattern specified by R. While $\mathrm{Lang}(R)$ is uniquely determined by R, it is typically possible to choose multiple regex which define the same language. We assume in the following that we have chosen an unambiguous regex R, so that each string $s \in \mathrm{Lang}(R)$ matches R exactly once. This involves no loss of generality, since any ambiguous regex can be replaced by an unambiguous regex defining the same language (Book et al., 1971). In such cases, we will use R to also represent the subset $\mathrm{Lang}(R)$.

To each regex R, we associate a pair of generalized transfer operators $\mathcal{E}^r_R$ and $\mathcal{E}^\ell_R$, formed by summing over all strings in the language R, which act on matrices as

$$\begin{aligned}\mathcal{E}^r_R(Q_r) &= \sum_{s\in R}\mathcal{A}(s)Q_r\mathcal{A}(s)^T, \\ \mathcal{E}^\ell_R(Q_\ell) &= \sum_{s\in R}\mathcal{A}(s)^T Q_\ell \mathcal{A}(s). \end{aligned} \tag{5}$$

While the naive sum appearing in Equation 5 can have infinitely many terms, the action of such CP maps can still be efficiently and exactly computed in terms of the recursive definition of the regex itself. Table 1 gives the correspondence between the four primitive regex operations introduced above and the corresponding operations on CP maps. Proof of the consistency between these recursive operations and Equation 5 for unambiguous regex, as well as a generalized correspondence holding for arbitrary regex, is given in the supplementary material.

While most regex operations in Table 1 are straightforward, the Kleene closure $\mathcal{E}^r_S$ involves an infinite summation which is guaranteed to converge whenever the spectral norm of $\mathcal{E}^r_S$ is bounded as $\rho(\mathcal{E}^r_S) < 1$. We denote the value of this convergent sum by Q_r^*, which can be approximated using a finite number of summands or alternately computed as the solution to the linear equation $(I - \mathcal{E}^r_S)Q_r^* = Q_r$ (see (Balle et al., 2019)).

A fruitful way of interpreting the transfer operators $\mathcal{E}^r_R$ and $\mathcal{E}^\ell_R$ is as normalization functions for u-MPS sampling distributions. We define the quantity $\mathcal{Z}_R(Q_\ell, Q_r) = \mathrm{Tr}(Q_\ell\mathcal{E}^r_R(Q_r))$ to be the (unnormalized) probability associated to a regex R in the presence of boundary matrices Q_ℓ, Q_r, and the quantity $\mathcal{Z}_R = \mathcal{Z}_R(\alpha\alpha^T, \omega\omega^T)$ as utilizing the boundary matrices of the u-MPS. It follows from Equation 5 that $\mathcal{Z}_R = \sum_{s\in R}\tilde{P}(s)$ does indeed give the unnormalized

that all Q_ℓ and Q_r are PSD.

Algorithm 1 Regex sampling algorithm for u-MPS

function REGSAMP(R, Q_ℓ, Q_r)
  if $R = s$ **then**
    # Sample a string literal $s \in \Sigma^*$
    return s
  else if $R = R_1 R_2$ **then**
    # Sample a sequence of expressions
    $s_1 = \text{REGSAMP}(R_1, Q_\ell, \mathcal{E}^r_{R_2}(Q_r))$
    $s_2 = \text{REGSAMP}(R_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)$
    return $s_1 s_2$
  else if $R = R_1 | R_2$ **then**
    # Sample a union of expressions
    Sample random $i \in \{1, 2\}$, with probs
      $p(i) = \mathcal{Z}_{R_i}(Q_\ell, Q_r) \;/\; \mathcal{Z}_{R_1|R_2}(Q_\ell, Q_r)$
    $s_i = \text{REGSAMP}(e_i, Q_\ell, Q_r)$
    return s_i
  else if $R = S^*$ **then**
    # Sample regex S *zero or more times*
    Sample random $i \in \{\text{HALT}, \text{GO}\}$, with probs
      $p(\text{HALT}) = \text{Tr}(Q_\ell Q_r)/\mathcal{Z}_{S^*}(Q_\ell, Q_r)$,
      $p(\text{GO}) = 1 - p(\text{HALT})$
    if $i = \text{HALT}$ **then**
      # Return empty string
      return ε
    else
      # Sample one or more chars
      return REGSAMP(SS^*, Q_ℓ, Q_r)

probability associated to all strings s matching R. We recover as special cases of this the quantities $\mathcal{Z}_n = \mathcal{Z}_{\Sigma^n}$ and $\mathcal{Z}_* = \mathcal{Z}_{\Sigma^*}$ defined above.

5 Regex Sampling and Regularization

The correspondence developed above between syntactic operations on regex and linear-algebraic operations on transfer operators endows u-MPS models with surprising capabilities unavailable to probabilistic models based on neural networks. We discuss the application of these techniques for sampling from conditional distributions of strings matching a target regex, as well as for utilizing a novel form of task-specific regularization during training.

5.1 Sampling

We introduce a regex-parameterized sampling function REGSAMP in Algorithm 1. REGSAMP gives a recursive means of converting any regex R into an efficient sampling procedure, whose random outputs are (for unambiguous R) unbiased samples from the conditional u-MPS distribution associated with the subset $R \subset \Sigma^*$. This is formalized in

Theorem 1. *Consider a u-MPS model with core tensor $\mathcal{A}$ and boundary vectors α and ω, along with an unambiguous regex R whose right transfer operator $\mathcal{E}^r_R$ converges. Let P indicate the probability distribution over arbitrary strings defined by the u-MPS, so that $\Sigma_{s\in\Sigma^*} P(s) = 1$. Then calling* REGSAMP($R, \alpha\alpha^T, \omega\omega^T$) *samples a string $s \in \Sigma^*$ from the conditional u-MPS distribution $P(s|s \in R) = P(s)/P(R)$, with $s \in R$ and where $P(R) := \sum_{s'\in R} P(s')$.*

We prove Theorem 1 in the supplementary material, which also discusses the use of ambiguous regex R. For this latter case, Algorithm 1 works identically, but weights strings s by the number of times s matches R.

Although Algorithm 1 is written in a recursive manner, it is useful to consider the simple example $R = \Sigma^n$, a concatenation of the single-character regex Σ with itself n times, to understand the overall control flow. In this case, Algorithm 1 first attempts to sample the initial character in the string via a recursive call to REGSAMP($\Sigma, \alpha\alpha^T, \mathcal{E}^r_{\Sigma^{n-1}}(\omega\omega^T)$). This requires $n-1$ applications of the transfer operator $\mathcal{E}^r$ to the initial right boundary matrix, and yields one new character before continuing to the right and repeating this process again.

As is common with recursive algorithms, caching intermediate information permits the naive cost of $(n-1) + (n-2) + \cdots + 1 = \mathcal{O}(n^2)$ transfer operator applications to be reduced to $\mathcal{O}(n)$. This cached version is equivalent to a simple iterative algorithm, where a sequence of right boundary matrices is first generated and saved during a right-to-left sweep, before a left-to-right sweep is used to sample text and propagate conditional information using the left boundary matrices. Using this idea, we show in the supplementary material that for typical regex R, Algorithm 1 can be run with average-case runtime $\mathcal{O}(LdD^3)$ and worst-case memory usage $\mathcal{O}(LD^2)$, for L the number of characters in R, d the size of Σ, and D the bond dimension of the u-MPS.

5.2 Regularization

The normalization function $\mathcal{Z}_R$ defined by a regex R gives the unnormalized probability assigned to all strings matching R. We first show that for the case of unambiguous R, this probability can be properly normalized.

Theorem 2. *Consider a u-MPS model and an unambiguous regex R satisfying identical conditions as in Theorem 1. Then the probability $P(R) = \sum_{s\in R} P(s)$ assigned by the u-MPS to the set of strings matching R can be exactly calculated as $P(R) = \mathcal{Z}_R/\mathcal{Z}_*$.*

The practical importance of Theorem 2 lies in the

ability to compute any $\mathcal{Z}_R = \mathrm{Tr}(\alpha\alpha^T \mathcal{E}_R^r(\omega\omega^T))$ inside of an automatic differentiation library, possibly with the aid of techniques described in (Liao et al., 2019). By making $P(R)$ directly computable as a function of the u-MPS parameters $\mathcal{A}$, α, and ω, this probability can be incorporated as a regularizer (i.e. a differentiable loss term) during training.

Although it is not immediately clear how to think about such "regex regularizers", we provide three examples which can be used during gradient-based training of a u-MPS model. First, $P(R)$ can be directly added to the loss, encouraging gradient updates of the model to minimize the probability of strings belonging to R. In the context of language models, this could be used to avoid learning offensive phrases seen in training data, for example by choosing $R = \Sigma^* S \Sigma^*$ with S a union of strings extracted from a dataset of abusive language.

A related loss is $\mathcal{L} = |P(R_1) - P(R_2)|$, which penalizes differences in the probabilities assigned to regex R_1 and R_2. Such regularization would be most effective when the regex R_1 and R_2 are similarly constructed, with the loss enforcing an indifference between these two options. This could be applied for the mitigation of gender bias in language models, for example by choosing each R_i to be $R_i = \Sigma^* s_i \Sigma^*$, for s_1 and s_2 a pair of identical but oppositely gendered phrases (e.g. "his career" vs. "her career" or "he cooks" vs. "she cooks").

Finally, using a loss $\mathcal{L} = -\log(P(R))$ encourages maximizing the probability of strings belonging to R. This type of regularization is natural when all strings produced by the model should belong to some regular language, for example when choosing a syntactically valid variable name in a code completion task. The extension of Theorem 1 and 2 to *context-free* languages would greatly broaden the range of applications for these methods in language modeling, given the fundamental role played by context-free grammars in structuring natural language.

6 Experiments

We use experiments on synthetic and real structured text datasets to assess the performance of u-MPS in probabilistic sequence modeling and grammatical inference, as well as to verify the benefits of u-MPS for parallelization, regex sampling, and regularization.

6.1 Synthetic Experiments

We first carry out experiments on several synthetic text datasets consisting of five Tomita grammars of binary strings and a context-free "Motzkin" grammar over the trinary alphabet $\Sigma_M = \{$ (, 0 ,) $\}$ (Tomita, 1982; Alexander et al., 2018). The latter consists of all strings whose parentheses are properly balanced, with no constraints placed on the 0 characters.

Table 2: Experiments on Tomita grammars 3-7 (see the supplementary material for the definitions of these grammars), where the training data is randomly chosen from strings of lengths between 1 and 15 belonging to the grammar. The trained models are used to sample strings of lengths 16 and 30, with the percentage of grammatically correct samples reported. The u-MPS consistently gives better generalization across different lengths (quite substantially for Tomita 5), except for Tomita 6 which neither model is able to learn. Most of the Tomita grammars are too small to train with more than 1,000 strings, but Tomita 5 and 6 permit experiments with larger datasets.

Tomita (N_{train})	Samp. Len.	u-MPS	HMM	LSTM	TR
3 (1K)	16	**100.0**	91.6	90.2	28.8
3 (1K)	30	**100.0**	82.0	85.6	9.4
4 (1K)	16	**99.9**	99.4	85.4	50.7
4 (1K)	30	99.5	**99.6**	64.7	32.5
5 (10K)	16	**100.0**	52.0	49.9	51.1
5 (10K)	30	**99.9**	49.8	52.8	50.5
6 (10K)	16	**35.9**	34.4	33.1	32.9
6 (10K)	30	33.1	33.1	**34.4**	32.7
7 (1K)	16	**99.3**	98.1	89.2	51.3
7 (1K)	30	**89.4**	79.3	29.1	10.0

In each case we train the u-MPS on strings of a restricted length from the grammar and then sample new strings of unseen lengths from the trained u-MPS, with the model assessed on the percentage of sampled strings which match the grammar. The sampling comes in two forms, either fixed length-n sampling (corresponding to $R = \Sigma^n$), or character completion sampling, where a single character in a reference string is masked and the prefix and suffix p and s are used to guess it (corresponding to $R = p\Sigma s$). While more general sampling experiments can easily be imagined, we have chosen these tasks because they allow for direct comparisons with a variety of baselines, including (unidirectional and bidirectional) LSTMs, HMMs, and Transformers.

While unbiased fixed-length sampling is easy for u-MPS via Algorithm 1, we found that the unidirectional LSTM baseline required an additional positional encoding in its inputs to avoid rapid degeneration in the output text when sampling past the longest length seen in training. At sampling time, we vary the length scale associated with this encoding based on the desired sampling length, so that the final step of sampling is always associated with the same positional encoding vector.

We train the u-MPS, LSTM, HMM, and small Transformer models using gradient descent on a negative log

Table 3: Experiments on the context-free Motzkin grammar, where the training set is fixed to contain only strings of length 15. We explore both fixed-length sampling (Samp) and character completion (Comp) tasks, where the model either samples a string from scratch, or predicts a missing character in a reference string given access to the character's prefix and suffix. In each case, the same trained u-MPS, HMM, and Transformer are used to generate both sampling and character completion data. The bidirectional LSTM performs best on shorter strings in the character completion task, but quickly degrades in accuracy as the length of the reference strings are increased.

Task (N_{train})	Str. Len.	u-MPS	HMM	LSTM	TR
Samp (1K)	1	**89.4**	37.4	41.7	39.1
Samp (1K)	16	**74.4**	30.3	41.2	2.3
Samp (1K)	50	**32.5**	12.6	0.0	0.6
Samp (10K)	1	**99.3**	36.0	35.7	36.2
Samp (10K)	16	**99.8**	34.3	60.4	0.5
Samp (10K)	50	**91.6**	12.4	5.4	0.2
Comp (1K)	1	89.4	39.2	**99.9**	32.1
Comp (1K)	16	69.6	29.1	**99.5**	30.7
Comp (1K)	50	58.8	13.1	**61.3**	30.2
Comp (10K)	1	99.3	36.3	**100.0**	33.5
Comp (10K)	16	99.8	31.7	**100.0**	34.5
Comp (10K)	50	**92.4**	14.8	69.1	33.7

likelihood (NLL) loss with the Adam (Kingma and Ba, 2015) optimizer. For each experiment we use models of $D = 20$ and $D = 50$ hidden units, with LSTMs chosen to have one layer and Transformers with two layers and 4 heads. Five independent trials are used for each value of D, with the final validation loss used to select the best model for generating samples. We use a piecewise constant learning rate between 10^{-2} and 10^{-4}, and early stopping to choose the end of training.

In the Tomita experiments (Table 2) u-MPS give impressive performance, in many cases achieving perfect accuracy in sampling strings of unseen sizes within the language. This is true not only in the simpler grammars Tomita 3 and 4, but also in the more difficult Tomita 5, where valid strings satisfy the nonlocal constraint of containing an even number of `0`'s and of `1`'s. The HMM and LSTM also attain reasonably high sample accuracy, although in a manner that degrades faster with sequence length than the u-MPS, while the Transformer performs worst.

Similar results are seen with the context-free Motzkin language (Table 3), where a fixed-length sampling task similar to the Tomita experiment is paired with a character completion task. A separate bidirectional LSTM must be used for this latter task, since unidirectional LSTMs cannot make use of future context information. By contrast, a trained u-MPS model can be employed in both of these settings without any task-specific adaptation, as well as in more general sentence completion tasks involving connected or disjoint regions of missing text (tasks which cannot be easily handled by standard RNN models). The u-MPS does substantially better in generalizing and reproducing the structure of Motzkin strings than the unidirectional LSTM, HMM, and Transformer, being able to sample strings of over 3 times the length seen in training with over 90% accuracy. The u-MPS is outperformed only by the bidirectional LSTM in character completion experiments on smaller training sets.

Table 4: Runtimes for computing the loss and gradient with respect to model parameters using a u-MPS with bond dimension 50, for a batch of 100 strings of length 500 evaluated on a CPU or GPU. While computation on a CPU favors sequential evaluation, owing to its lower overall cost, the reduced parallel depth inherent to parallel evaluation leads to a reduced runtime in the presence of GPU hardware acceleration.

	Sequential Eval. Runtime (ms)	Parallel Eval. Runtime (ms)
CPU comp.	**73.0**	232.7
GPU comp.	49.9	**45.2**

Given the ability of HMMs to exactly reproduce the distributions associated with Tomita languages and (bounded length) Motzkin languages, it is surprising that the u-MPS still manages to more easily learn such distributions in practice. Somewhat surprising also is the poor performance of the Transformer models, which is likely a result of the small sizes of the string datasets used.

We additionally benchmark the relative runtime of sequential and parallel evaluation for u-MPS running on a CPU or GPU (Table 4). For a u-MPS of D hidden states with strings of length n, RNN-style sequential evaluation has parallel depth $\mathcal{O}(n)$ and cost $\mathcal{O}(nD^2)$, while parallel evaluation trades this for a parallel depth of $\mathcal{O}(\log n)$ and cost $\mathcal{O}(nD^3)$. This would suggest parallel evaluation having benefits for the total runtime when hardware acceleration is present, which the runtimes in Table 4 confirm.

6.2 Email Experiments

To verify the correctness and relative benefits of regex sampling and regularization, we train on a dataset of approximately 4,000 email addresses taken from the CLAIR fraudulent emails dataset (Radev, 2008). Although the correctness of an email address gener-

Table 5: Training on emails and assessing the per-character perplexity (PPL) and accuracy of unconditional sampling using a u-MPS with bond dimension 50. Making use of a regularizer associated with a regex R_e, approximating the formatting of valid email addresses, leads to small gains in both the syntactic correctness of unconditionally sampled strings and overall perplexity. Note that the use of *conditional* sampling relative to R_e would guarantee the generation of syntactically valid strings (Theorem 1), a fact we verify experimentally.

	u-MPS w/ regex reg.	u-MPS w/o regex reg.
Correct %	**37.2**	35.4
Test PPL	**7.3**	7.8

ally depends on non-syntactic considerations (such as domain name resolution), we can approximate the format of valid email addresses using the regex R_e = `[\w-.]+@([\w-]+.)+[\w-][\w-]+` (a generalization of the pattern `name@site.tld`).

We first train a u-MPS using gradient descent on this email address dataset and look at the perplexity of a held-out validation set and correctness of (unconditionally) sampled strings, both with and without the use of regularization associated with R_e. We find the use of regex regularization to yield small improvements to perplexity and correctness of sampled text, as shown in Table 5.

Although the correctness of conditional regex sampling is already guaranteed by Theorem 1, we confirm this experimentally by using the regex R_e to periodically produce conditional samples during training. We find that conditional sampling relative to R_e does indeed always yield random strings matching the desired regex, but with the quality of generated text gradually improving during training. While conditional sampling of the randomly initialized u-MPS produces syntactically-valid but otherwise random strings, such as `k90@4riuh2600xz1.wz`, training the model on the email address dataset leads to the production of more realistic-looking email addresses, such as `sail203@yahoo.com`.

7 Conclusion

We develop a u-MPS model for probabilistic modeling of sequence data, which we show has distinctive capabilities regarding parallelism, sampling, and regularization, in a manner which mirrors the structure of regular languages. Although our results are derived in the specific context of u-MPS, the underlying techniques used to demonstrate these capabilities are applicable to other models with a similar mathematical layout, such as WFA, HMM, and PSR. As a result, our Theorems 1 and 2 should generalize to other such models. We expect the algorithms developed here to be associated with different runtimes when applied to other models, leading to different performance tradeoffs of parallelization, sampling, and regularization methods than what is reported here. For example, the parallel evaluation method used here requires $\mathcal{O}(nD^3)$ resources for a u-MPS, compared with $\mathcal{O}(nD^6)$ for a HQMM.

Beyond these immediate generalizations, a more interesting extension of our results is the generalization of Theorems 1 and 2 to the setting of context-free languages. While all regular languages are context-free, the latter has significantly greater expressive power and relevance for applications in natural language processing and automatic code generation. Surprisingly, we have found that such languages can indeed be sampled from and employed for regularizers within u-MPS models, although with a higher cost of $\mathcal{O}(D^6)$.

Given the ability we demonstrate to use grammars to constrain the probability distribution of u-MPS models, a natural question is whether the inverse process is possible: Namely, given a trained u-MPS model, do there exist automatic means of identifying grammatical rules which account for some portion of the correlations present within the learned probability distribution? Such techniques would represent a qualitatively new type of grammatical inference, where grammar rules and language models interact in a two-way manner.

Finally, a natural next step is scaling up u-MPS for real-world sequence modeling tasks, notably language modeling. Some current obstacles to this process are (a) the $\mathcal{O}(D^3)$ cost of certain u-MPS operations, and (b) the absence of well-established best-practices for training large tensor networks with gradient descent. We expect these obstacles to be overcome by dedicated engineering effort and the rapidly growing number of software libraries for manipulating tensor networks, along with the adoption of powerful computational methods developed by the many-body physics community into machine learning. Considering the unexpected benefits demonstrated here, we expect recurrent tensor network architectures to have a bright future.

Acknowledgements

This research is supported by the Canadian Institute for Advanced Research (CIFAR AI chair program).

References

Alexander, R. N., Evenbly, G., and Klich, I. (2018). Exact holographic tensor networks for the Motzkin

spin chain. *arXiv:1806.09626*.

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

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

Book, R., Even, S., Greibach, S., and Ott, G. (1971). Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, 100(2):149–153.

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

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

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

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

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

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

Droste, M., Kuich, W., and Vogler, H. (2009). *Handbook of weighted automata*. Springer.

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

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

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

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

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org.

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

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

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

Liao, H.-J., Liu, J.-G., Wang, L., and Xiang, T. (2019). Differentiable programming tensor networks. *arXiv:1903.09650*.

Littman, M. L. and Sutton, R. S. (2002). Predictive representations of state. In *Advances in neural information processing systems*, pages 1555–1561.

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

Monras, A., Beige, A., and Wiesner, K. (2010). Hidden quantum markov models and non-adaptive read-out of many-body states. *arXiv:1002.2337*.

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

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

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

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

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

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

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

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

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

Radev, D. (2008). Clair collection of fraud email, acl data and code repository. *ADCR2008T001*.

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

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

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

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

Tomita, M. (1982). Dynamic construction of finite-state automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pages 105–108.

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

White, S. R. (1992). Density matrix formulation for quantum renormalization groups. *Physical review letters*, 69(19):2863.

Zhao, M.-J. and Jaeger, H. (2010). Norm-observable operator models. *Neural computation*, 22(7):1927–1959.