Supplemental Material to Tensor Networks for Probabilistic Sequence Modeling

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

Guillaume Rabusseau
Mila and DIRO
Université de Montréal
Montréal QC, Canada
grabus@iro.umontreal.ca

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

Preprint. Under review.

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

A Completely Positive Maps and Generalized Transfer Operators

In this section, we give definitions and known results concerning completely positive (CP) maps and regular expressions (regex), as well as further details regarding the assignment of generalized transfer operators to regex[1]. We conclude with a proof that the recursive definition of generalized transfer operators given in Table 1 (repeated here for convenience) has an equivalent representation in terms of a weighted sum over all strings in the regex, which for unambiguous regex gives precisely the simple form of Equation 5.

We say that a matrix $Q \in \mathbb{R}^{D \times D}$ is positive semidefinite (PSD) if it is (a) Symmetric ($Q^T = Q$), and (b) satisfies $v^T Q v \geq 0$ for every $v \in \mathbb{R}^D$. If Q further satisfies the property that $v^T Q v = 0$ only when $v = 0$, then we call it positive definite. Given a PSD matrix Q, the diagonal elements of Q will necessarily be non-negative. For any vector v, the rank-1 matrix vv^T is necessarily a PSD matrix, and all PSD matrices $Q \in \mathbb{R}^{D \times D}$ can be expressed as the weighted sum of (at most) D such rank-1 matrices. This can be used to show that for any PSD matrices Q and Q', $\mathrm{Tr}(QQ') \geq 0$.

It is common in quantum mechanics to regard PSD matrices as a generalized form of probabilistic states [5], a viewpoint which allows us to consider the matrices Q_ℓ, Q_r as probabilistic latent states of a u-MPS. To this end, the family of completely positive (CP) maps is the natural generalization of stochastic maps, which act on these PSD matrices. A map $\mathcal{E}$ sending PSD $Q \in \mathbb{R}^{D \times D}$ to $Q' = \mathcal{E}(Q) \in \mathbb{R}^{D' \times D'}$ is said to be CP if it admits a Kraus representation, consisting of $r \geq 1$ matrices $A_i \in \mathbb{R}^{D' \times D}$ such that $\mathcal{E}$ can be expressed as:

$$\mathcal{E}(Q) = \sum_{i=1}^{r} A_i Q A_i^T \tag{1}$$

The condition (1) implies in particular that $\mathcal{E}(Q)$ is PSD if Q is. The matrices in (1) are called the Kraus operators of the map, and the same CP map $\mathcal{E}$ can be given multiple Kraus representations with inequivalent values of r. The minimum value of r such that $\mathcal{E}$ can be represented as (1) is called the rank of $\mathcal{E}$, and is always bounded as $r \leq D^2$. Nonetheless, Kraus representations with a greater number of Kraus operators can be useful for understanding the action of the map, as we will see below.

By taking the transpose of all the Kraus operators appearing in (1), we obtain a new CP map $\mathcal{E}^T$, which is the adjoint of $\mathcal{E}$. Mathematically, this means that for any CP map $\mathcal{E}$ and positive matrices Q_ℓ, Q_r, the equality $\mathrm{Tr}(Q_\ell \mathcal{E}(Q_r)) = \mathrm{Tr}(\mathcal{E}^T(Q_\ell) Q_r)$. For greater clarity, in the context of sequence modeling with u-MPS we frequently refer to a CP map and its adjoint as "right" and "left" maps $\mathcal{E}^r$ and $\mathcal{E}^\ell$, rather than $\mathcal{E}$ and $\mathcal{E}^T$.

The term "transfer operator" is common in many-body physics, and in our setting refers to a CP map $\mathcal{E}$ obtained from the Kraus representation with $A_i = \mathcal{A}(\varphi^{-1}(i))$, for $\mathcal{A}$ a u-MPS core tensor and $\varphi : \Sigma \to [d]$ a bijection mapping characters c in the size-d alphabet Σ to the numbers $\{1, 2, \ldots, d\}$ (see Figure 1f). In Section 4 we introduced a generalization of this standard notion of transfer operator to include a number of other CP maps $\mathcal{E}_R$ associated with an arbitrary regular expression R, whose recursive definition is given in Table 1.Using Σ to also denote the regex matching any single character in our alphabet, the standard transfer operator emerges as the special case $R = \Sigma$.

We sometimes assume that our regex R is unambiguous, in the sense that any string s matching R matches in exactly one way. This assumption can be made without loss of generality, since any ambiguous regex R can be converted into an unambiguous regex R' accepting the same set of strings [2]. For example, the ambiguous regex $R = a^* a^*$ matches the string $s = a$ in two ways, but can be replaced with the equivalent unambiguous regex $R' = a^*$.

[1] We present all definitions and results in this section in terms of real-valued matrices, but the corresponding statements for complex-valued matrices and CP maps is obtained by replacing matrix transposes Q^T by Hermitian adjoints $Q^\dagger$, representing the transposed and complex-conjugated counterpart of Q.

Any regex R can be built inductively from (a) Single characters $c \in \Sigma$, (b) Concatenations of regex $R = R_1 R_2$, (c) Unions of regex $R = R_1|R_2$, and (d) Kleene closures of regex $R = S^*$. We prove by induction over the structure of R that any generalized right transfer operator $\mathcal{E}_R^r$ defined by the recursive procedure in Table 1acts according to a generalization of Equation 5,as stated in

Theorem 1. *Consider the generalized transfer operators $\mathcal{E}_R^r$ and $\mathcal{E}_R^\ell$ associated with an arbitrary regex R and a u-MPS with core tensor $\mathcal{A}$, which are defined by the recursive rules in Table 1.Then $\mathcal{E}_R^r$ converges if and only if $\mathcal{E}_L^r$ converges, and in this case the transfer operators are described by the Kraus representations,*

$$\mathcal{E}_R^r(Q_r) = \sum_{s\in\Sigma^*} |s|_R \mathcal{A}(s) Q_r \mathcal{A}(s)^T, \qquad \mathcal{E}_R^\ell(Q_\ell) = \sum_{s\in\Sigma^*} |s|_R \mathcal{A}(s)^T Q_\ell \mathcal{A}(s), \tag{2}$$

where $|s|_R$ denotes the number of times the string s matches the regex R. For unambiguous regex, this Kraus representation is identical to that of Equation 5.

Proof. For each of the four types of regex R, we make the inductive assumption that the subexpressions of R (if any) satisfy (2), and use this to prove that the transfer operator $\mathcal{E}_R^r$ satisfies (2). This allows us to immediately prove the corresponding statement for the left transfer operator $\mathcal{E}_R^\ell$.

$\mathbf{R = c}$: Apparent from Table 1and the single string which matches R, $s = c$.

$\mathbf{R = R_1 R_2}$: Assume R_1 and R_2 both satisfy (2). Table 1gives:

$$\begin{aligned}\mathcal{E}_{R_1R_2}^r(Q_r) = \mathcal{E}_{R_1}^r(\mathcal{E}_{R_2}^r(Q_r)) &= \sum_{s_1\in\Sigma^*}\sum_{s_2\in\Sigma^*} |s_1|_{R_1}|s_2|_{R_2}\mathcal{A}(s_1)\mathcal{A}(s_2) Q_r \mathcal{A}(s_2)^T\mathcal{A}(s_1)^T \\ &= \sum_{s_1\in\Sigma^*}\sum_{s_2\in\Sigma^*} |s_1|_{R_1}|s_2|_{R_2}\mathcal{A}(s_1s_2) Q_r \mathcal{A}(s_1s_2)^T = \sum_{s\in\Sigma^*} |s|_{R_1R_2}\mathcal{A}(s) Q_r \mathcal{A}(s)^T.\end{aligned}$$

In the second-to-last equality we used the compositional property $\mathcal{A}(s_1)\mathcal{A}(s_2) = \mathcal{A}(s)$, while in the last equality we used the identity $|s|_{R_1R_2} = \sum_{s_1 s_2 = s} |s_1|_{R_1}|s_2|_{R_2}$, where the sum over s_1, s_2 represents all possible partitions of s into a prefix and suffix.

$\mathbf{R = R_1|R_2}$: Assume R_1 and R_2 both satisfy (2). Table 1gives:

$$\begin{aligned}\mathcal{E}_{R_1|R_2}^r(Q_r) = \mathcal{E}_{R_1}^r(Q_r) + \mathcal{E}_{R_2}^r(Q_r) &= \left(\sum_{s\in\Sigma^*} |s|_{R_1}\mathcal{A}(s) Q_r \mathcal{A}(s)^T\right) + \left(\sum_{s\in\Sigma^*} |s|_{R_2}\mathcal{A}(s) Q_r \mathcal{A}(s)^T\right) \\ &= \sum_{s\in\Sigma^*} |s|_{R_1|R_2}\mathcal{A}(s) Q_r \mathcal{A}(s)^T.\end{aligned}$$

In the final equality we have used the identity $|s|_{R_1|R_2} = |s|_{R_1} + |s|_{R_2}$.

$\mathbf{R = S^*}$: Assume S satisfies (2). The operator $\mathcal{E}_{S^*}^r$ will converge only when the spectral norm of $\mathcal{E}_S^r$ satisfies $\rho(\mathcal{E}_S^r) < 1$, in which case Table 1gives:

$$\begin{aligned}\mathcal{E}_{S^*}^r(Q_r) = \sum_{n=0}^{\infty} (\mathcal{E}_S^r)^{\circ n}(Q_r) &= \sum_{n=0}^{\infty}\sum_{s_1\cdots s_n\in\Sigma^*} |s_1|_S\cdots|s_n|_S\, \mathcal{A}(s_1\cdots s_n) Q_r \mathcal{A}(s_1\cdots s_n)^T \\ &= \sum_{s\in\Sigma^*} |s|_{S^*}\, \mathcal{A}(s) Q_r \mathcal{A}(s)^T.\end{aligned}$$

In the final equality we have used the identity $|s|_{S^*} = \sum_{n=0}^{\infty}\sum_{s_1\cdots s_n = s} |s_1|_S\cdots|s_n|_S$, where the sum over $s_1, \ldots, s_n$ represents all possible partitions of s into n contiguous pieces.

While we only characterized the action of the right transfer operators $\mathcal{E}_R^r$, substituting all matrices $\mathcal{A}(s)$ with their transposed counterparts $\mathcal{A}(s)^T$ immediately yields the corresponding characterization for the action of $\mathcal{E}_R^\ell$. In this latter case, the direction-reversing identity $\mathcal{A}(s_1s_2)^T = \mathcal{A}(s_2)^T\mathcal{A}(s_1)^T$ is accounted for by the transfer operator correspondence $\mathcal{E}_{R_1R_2}^\ell = \mathcal{E}_{R_2}^\ell \mathcal{E}_{R_1}^\ell$.

Because $\mathcal{E}_R^r$ and $\mathcal{E}_R^\ell$ are adjoints of each other, their eigenvalue spectra are identical, and therefore $\mathcal{E}_R^r$ converges if and only if $\mathcal{E}_R^\ell$ converges. Finally for unambiguous regex R, the quantity $|s|_R \in \{0, 1\}$, giving the equality $\sum_{s\in\Sigma^*} |s|_R \mathcal{A}(s) Q_r \mathcal{A}(s)^T = \sum_{s\in R} \mathcal{A}(s) Q_r \mathcal{A}(s)^T$ which proves Equation 5.

□

Although it is not obvious a priori when a transfer operator $\mathcal{E}_R^r$ will converge for a given regex R and core tensor $\mathcal{A}$, it is clear that the Kleene closure is the only operation permitting divergence. Consequently, a regex R will converge only when all of its subexpressions of the form S_i^* have spectral norm $\rho(\mathcal{E}_{S_i}^r) < 1$. Note that any S^* for which S accepts the empty string is guaranteed to produce a divergent transfer operator $\mathcal{E}_{S^*}^r$, so that in particular any transfer operator of the form $\mathcal{E}_{(S^*)^*}^r$ is divergent.

B Proof of Theorem 1

In order to prove Theorem 1, we first prove a more general Lemma 1, which characterizes the probability distribution $P_R(s, Q_\ell, Q_r)$ of strings output by $\text{SAMPLE}(R, Q_\ell, Q_r)$ for arbitrary R.

Lemma 1. *Consider a u-MPS model with core tensor $\mathcal{A}$ and a regex R for which the generalized right transfer operator $\mathcal{E}_R^r$ defined recursively by Table 1converges. Then for any PSD matrices Q_ℓ, Q_r, the probability distribution of strings output by $\text{SAMPLE}(R, Q_\ell, Q_r)$ is $P_R(s, Q_\ell, Q_r) = |s|_R \tilde{P}(s, Q_\ell, Q_r)/\mathcal{Z}_R(Q_\ell, Q_r)$, where $\tilde{P}(s, Q_\ell, Q_r) = \text{Tr}(Q_\ell \mathcal{E}_s^r(Q_r))$, $\mathcal{Z}_R(Q_\ell, Q_r) = \text{Tr}(Q_\ell \mathcal{E}_R^r(Q_r))$, and $|s|_R$ counts the number of times the string s matches the regex R.*

Proof. We prove Lemma 1 by induction over the structure of R, where we assume that sampling from a regex subexpression R' of R with any boundary matrices Q'_ℓ and Q'_r produces strings from the distribution $P_{R'}(s, Q'_\ell, Q'_r)$. For each of the four cases of regex formation, we use this inductive assumption to show that sampling from R produces strings from the distribution $P_R(s, Q_\ell, Q_r)$.

$\mathbf{R = c}$: From Algorithm 1,$\text{SAMPLE}(c, Q_\ell, Q_r)$ will always output the string $s = c$. Because the quantity $|s|_c$ is 1 when $s = c$ and 0 otherwise, the sampling distribution can be written as

$$P_c(s, Q_\ell, Q_r) = |s|_c = |s|_c \frac{\text{Tr}(Q_\ell \mathcal{E}_s^r(Q_r))}{\text{Tr}(Q_\ell \mathcal{E}_c^r(Q_r))} = |s|_c \frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_c(Q_\ell, Q_r)}.$$

$\mathbf{R = R_1 R_2}$: From Algorithm 1, $\text{SAMPLE}(R_1 R_2, Q_\ell, Q_r)$ will first output a string s_1 from $\text{SAMPLE}(R_1, Q_\ell, \mathcal{E}_{R_2}^r(Q_r))$, then use s_1 to output a string s_2 from $\text{SAMPLE}(R_2, \mathcal{E}_{s_1}^\ell(Q_\ell), Q_r)$. Using our inductive assumption for R_1 and R_2, the probability assigned to the output string s from all possible partitions into a prefix and suffix as $s_1 s_2 = s$ is

$$\begin{aligned}
P_{R_1R_2}(s, Q_\ell, Q_r) &= \sum_{s_1 s_2 = s} P_{R_1}(s_1, Q_\ell, \mathcal{E}_{R_2}^r(Q_r)) \cdot P_{R_2}(s_2, \mathcal{E}_{s_1}^\ell(Q_\ell), Q_r) \\
&= \sum_{s_1 s_2 = s} \left(|s_1|_{R_1} \frac{\text{Tr}(Q_\ell \mathcal{E}_{s_1}^r(\mathcal{E}_{R_2}^r(Q_r)))}{\text{Tr}(Q_\ell \mathcal{E}_{R_1}^r(\mathcal{E}_{R_2}^r(Q_r)))} \right) \left(|s_2|_{R_2} \frac{\text{Tr}(\mathcal{E}_{s_1}^\ell(Q_\ell) \mathcal{E}_{s_2}^r(Q_r))}{\text{Tr}(\mathcal{E}_{s_1}^\ell(Q_\ell) \mathcal{E}_{R_2}^r(Q_r))} \right) \\
&= \sum_{s_1 s_2 = s} |s_1|_{R_1} |s_2|_{R_2} \left(\frac{\text{Tr}(Q_\ell \mathcal{E}_{s_1 R_2}^r(Q_r))}{\text{Tr}(Q_\ell \mathcal{E}_{R_1 R_2}^r(Q_r))} \right) \left(\frac{\text{Tr}(Q_\ell \mathcal{E}_{s_1 s_2}^r(Q_r))}{\text{Tr}(Q_\ell \mathcal{E}_{s_1 R_2}^r(Q_r))} \right) \\
&= |s|_{R_1 R_2} \frac{\text{Tr}(Q_\ell \mathcal{E}_s^r(Q_r))}{\text{Tr}(Q_\ell \mathcal{E}_{R_1 R_2}^r(Q_r))} = |s|_{R_1 R_2} \frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{R_1 R_2}(Q_\ell, Q_r)}.
\end{aligned}$$

In the third equality above, we use the composition rule $\mathcal{E}_{s'}^r \mathcal{E}_{R'}^r = \mathcal{E}_{s'R'}^r$ and the adjunction rule $\text{Tr}(\mathcal{E}_{s'}^\ell(Q'_\ell) Q'_r) = \text{Tr}(Q'_\ell \mathcal{E}_{s'}^r(Q'_r))$, while in the fourth equality, we use the identity $|s|_{R_1 R_2} = \sum_{s_1 s_2 = s} |s_1|_{R_1} |s_2|_{R_2}$.

$\mathbf{R} = \mathbf{R_1}|\mathbf{R_2}$: From Algorithm 1, SAMPLE$(R_1|R_2, Q_\ell, Q_r)$ will first pick a random index $i \in 1, 2$ with probability $p(i) = \mathcal{Z}_{R_i}(Q_\ell, Q_r) \;/\; \mathcal{Z}_{R_1|R_2}(Q_\ell, Q_r)$, and then use this to output a string s from SAMPLE(R_i, Q_ℓ, Q_r). Using our inductive assumption, the probability assigned to the output string s is

$$\begin{aligned} P_{R_1|R_2}(s, Q_\ell, Q_r) &= \sum_{i\in\{1,2\}} p(i) \cdot P_{R_i}(s_i, Q_\ell, Q_r) \\ &= \sum_{i\in\{1,2\}} \left(\frac{\mathcal{Z}_{R_i}(Q_\ell, Q_r)}{\mathcal{Z}_{R_1|R_2}(Q_\ell, Q_r)}\right)\left(|s|_{R_i}\frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{R_i}(Q_\ell, Q_r)}\right) \\ &= |s|_{R_1|R_2}\frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{R_1|R_2}(Q_\ell, Q_r)}. \end{aligned}$$

In the final equality, we have used the identity $|s|_{R_1|R_2} = |s|_{R_1} + |s|_{R_2}$.

$\mathbf{R} = \mathbf{S}^*$: To sample from the regex R we must have the infinite sum defining $\mathcal{E}^r_R$ in Table 1converge, which is guaranteed by the assumptions of Lemma 1. Given this convergence, calling SAMPLE(S^*, Q_ℓ, Q_r) will either output the empty string $s = \varepsilon$, or else call SAMPLE(SS^*, Q_ℓ, Q_r). In the latter case, the concatenation rule will then sample some $s' \in S$ before calling SAMPLE$(S^*, \mathcal{E}^\ell_{s'}(Q_\ell), Q_r)$ to sample some random number $n \geq 0$ occurrences of S.

We denote the unnormalized collection of probabilities associated with strings produced from exactly n occurrences of S as $P^{(n)}_{S^*}$, and we use an inductive proof to show that $P^{(n)}_{S^*} = p(n)P_{S^n}$, for $p(n, Q_\ell, Q_r) = \mathcal{Z}_{S^n}(Q_\ell, Q_r)/\mathcal{Z}_{S^*}(Q_\ell, Q_r)$. In other words, our recursive sampling procedure for S^* is equivalent to first sampling a random length using $p(n)$, then calling the corresponding SAMPLE(S^n, Q_ℓ, Q_r).

Base case $n = 0$**:** The regex S^0 matches only the empty string $s = \varepsilon$, and from Algorithm 1, this occurs with probability

$$P^{(0)}_{S^*}(s, Q_\ell, Q_r) = \frac{\mathrm{Tr}(Q_\ell Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} = \frac{\mathcal{Z}_{S^0}(Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} = p(0)P_{S^0}(s, Q_\ell, Q_r),$$

where we have used the identity $\mathcal{E}^r_{S^0} = \mathcal{E}^r_\varepsilon = I$, and the fact that $P_{S^0}(s)$ is 1 for $s = \varepsilon$ and 0 otherwise.

Step case $n+1$**:** From Algorithm 1, the probability of sampling a string $s = s_1 s_2$ with s_1 matching S and s_2 matching S^n is

$$\begin{aligned} P^{(n+1)}_{S^*}(s, Q_\ell, Q_r) &= \sum_{s_1 s_2 = s}\left(1 - \frac{\mathrm{Tr}(Q_\ell Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}\right) P_S(s_1, Q_\ell, \mathcal{E}^r_{S^*}(Q_r))P^{(n)}_{S^*}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r) \\ &= \sum_{s_1 s_2 = s}\left(\frac{\mathcal{Z}_{SS^*}(Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}\right)\left(|s_1|_S\frac{\tilde{P}(s_1, Q_\ell, \mathcal{E}^r_{S^*}(Q_r))}{\mathcal{Z}_S(Q_\ell, \mathcal{E}^r_{S^*}(Q_r))}\right) P^{(n)}_{S^*}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r) \\ &= \sum_{s_1 s_2 = s}|s_1|_S\frac{\tilde{P}(s_1, Q_\ell, \mathcal{E}^r_{S^*}(Q_r))}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}\frac{\mathcal{Z}_{S^n}(\mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)}{\mathcal{Z}_{S^*}(\mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)}|s_2|_{S^n}\frac{\tilde{P}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)}{\mathcal{Z}_{S^n}(\mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)} \\ &= \sum_{s_1 s_2 = s}|s_1|_S|s_2|_{S^n}\frac{\tilde{P}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} = \sum_{s_1 s_2 = s}|s_1|_S|s_2|_{S^n}\frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} \\ &= \frac{\mathcal{Z}_{S^{n+1}}(Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}|s|_{S^{n+1}}\frac{\tilde{P}(s_1 s_2, Q_\ell, Q_r)}{\mathcal{Z}_{S^{n+1}}(Q_\ell, Q_r)} = p(n+1, Q_\ell, Q_r)P_{S^{n+1}}(s, Q_\ell, Q_r). \end{aligned}$$

In the above we used the following identities: $\mathcal{Z}_{S^*}(Q_\ell, Q_r) = \mathrm{Tr}(Q_\ell Q_r) + \mathcal{Z}_{SS^*}(Q_\ell, Q_r)$ (second equality), $\mathcal{Z}_S(Q_\ell, \mathcal{E}^r_{S^*}(Q_r)) = \mathcal{Z}_{SS^*}(Q_\ell, Q_r)$ (third equality), $\tilde{P}(s_1, Q_\ell, \mathcal{E}^r_{S^*}(Q_r)) = \mathcal{Z}_{S^*}(\mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)$ (fourth equality), $\tilde{P}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r) = \tilde{P}(s_1 s_2, Q_\ell, Q_r)$ (fifth equality), and $|s|_{S^{n+1}} = \sum_{s_1 s_2 = s}|s_1|_S|s_2|_{S^n}$ (sixth equality).

With this inductive characterization of the unnormalized distributions $P^{(n)}_{S^*}$, we can finally show

$$\begin{aligned} P_{S^*}(s, Q_\ell, Q_r) &= \sum_{n=0}^{\infty} P^{(n)}_{S^*}(s, Q_\ell, Q_r) = \sum_{n=0}^{\infty} \frac{\mathcal{Z}_{S^n}(Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} |s|_{S^n} \frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{S^n}(Q_\ell, Q_r)} \\ &= |s|_{S^*} \frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}, \end{aligned}$$

where we have used the identity $\sum_{n=0}^{\infty} |s|_{S^n} = |s|_{S^*}$ in the last equality.

□

Having proved Lemma 1, we can now prove Theorem 1 as a simple corollary, which is restated here for ease of reference.

Theorem. *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* $\text{SAMPLE}(R, \alpha\alpha^T, \omega\omega^T)$ *generates a random string $s \in \Sigma^*$ from the conditional u-MPS distribution $P_*(s|s \in R) = P_*(s)/P_*(R)$, where $P_*(R) := \sum_{s'\in R} P_*(s')$.*

Proof. From Lemma 1, we know the probability distribution of strings output by $\text{SAMPLE}(R, \alpha\alpha^T, \omega\omega^T)$, as well as $P_*(s) = \text{SAMPLE}(\Sigma^*, \alpha\alpha^T, \omega\omega^T)$. This characterization lets us show

$$\begin{aligned} P_R(s, \alpha\alpha^T, \omega\omega^T) &= |s|_R \frac{\tilde{P}(s, \alpha\alpha^T, \omega\omega^T)}{\mathcal{Z}_R(\alpha\alpha^T, \omega\omega^T)} = |s|_R \frac{\tilde{P}(s, \alpha\alpha^T, \omega\omega^T)}{\text{Tr}(\alpha\alpha^T \mathcal{E}^r_R(\omega\omega^T))} \\ &= |s|_R \left(\frac{\tilde{P}(s, \alpha\alpha^T, \omega\omega^T)}{\mathcal{Z}_{\Sigma^*}(\alpha\alpha^T, \omega\omega^T)} \right) \left(\frac{\mathcal{Z}_{\Sigma^*}(\alpha\alpha^T, \omega\omega^T)}{\sum_{s'\in R} \tilde{P}(s', \alpha\alpha^T, \omega\omega^T)} \right) \\ &= |s|_R \frac{P_{\Sigma^*}(s, \alpha\alpha^T, \omega\omega^T)}{\sum_{s'\in R} P_{\Sigma^*}(s', \alpha\alpha^T, \omega\omega^T)} = \begin{cases} P_*(s)/P_*(R), & \text{if } s \in R \\ 0, & \text{otherwise} \end{cases} \\ &= P_*(s|s \in R). \end{aligned}$$

In the third equality we used (2) in Theorem 1 (which reduces to Equation 5 for an unambiguous R) to express $\mathcal{E}^r_R$ as $\mathcal{E}^r_R = \sum_{s\in R} \mathcal{E}^r_s$, while also introducing a normalization factor associated with Σ^* to the numerator and denominator. In the last equality we have used the definition of the conditional probability distribution associated with s matching the regex R, and have also utilized the fact that $|s|_R$ is either 1 or 0 for unambiguous regex.

□

C Runtime Analysis

Algorithm 1 is written as a recursive procedure, which makes its runtime analysis nontrivial. We show here that this sampling procedure can in most cases be carried out using compute and storage costs which scale linearly with the length L_R of the defining regex R. As a technical assumption, we require the star height of R to be bounded, where the star height $h_*(R)$ is defined recursively as $h_*(c) = 0$, $h_*(R_1R_2) = h_*(R_1|R_2) = \max(h_*(R_1), h_*(R_2))$, and $h_*(S^*) = 1 + h_*(S)$. In practice this assumption is very mild.

Theorem 2. *Consider a core tensor $\mathcal{A}$ and regex R of length L_R with bounded star height, for which the associated right transfer operator $\mathcal{E}^r_R$ converges. Then calling* $\text{SAMPLE}(R, Q_\ell, Q_r)$ *will return a random string of mean length $\langle n \rangle = \mathcal{O}(L_R)$, with average-case compute cost of $C_R = \mathcal{O}(L_R d D^3)$ and worst-case memory cost of $M_R = \mathcal{O}(L_R D^2)$.*

Proof. We again utilize a proof by induction, with the additional assumption that C_R is also an upper bound on the expected cost of applying the transfer operator $\mathcal{E}^r_R$. For the regex length L_R, we consider the characters $|$, $($, $)$, *, and c (for $c \in \Sigma$) as having length 1, along with the single-character

regex Σ. This definition of length is closer to that of real-world regex, where the metacharacter "." corresponds to our Σ.

In order to utilize caching in Algorithm 1, we replace the simple recursive case of binary regex concatenation $R = R_1 R_2$ with a maximal concatenation of smaller regex $R = R_1 R_2 \cdots R_K$, where each R_i is either a single character, a union of regex, or a Kleene closure. Such concatenations are the only place where caching is utilized, allowing us to bound the memory usage in terms of the regex length L_R, rather than the random string length n_R. We don't include the non-cache memory usage required to hold our u-MPS parameters and intermediate variables, which is in every case $\mathcal{O}(dD^2)$.

Given that the length n_R of an output sample is typically a random variable, we first show that the mean length is bounded as $\langle n_R \rangle = \mathcal{O}(L_R)$. It is apparent that for any regex R constructed without Kleene closures, we have the stronger bound $n_R \leq L_R$. We therefore start our inductive proof with this last remaining case of Kleene closures, showing that $\langle n_R \rangle = \mathcal{O}(L_R)$ for all regex R with bounded star height. We then use this result to characterize the compute and memory requirements of Algorithm 1.

$\mathbf{R} = \mathbf{S}^*$: In order for $\mathcal{E}^r_{S^*}$ to converge, we must have the spectral radius of $\mathcal{E}^r_S$ be $\lambda_S := \rho(\mathcal{E}^r_S) < 1$. Noting that this implies $\mathrm{Tr}(Q_\ell \mathcal{E}^r_S(Q_r)) \leq \lambda_S \,\mathrm{Tr}(Q_\ell Q_r)$ for any boundary matrices Q_ℓ, Q_r, we find that the probability of obtaining m occurrences of S is upper bounded as

$$p(m) = \mathrm{Tr}(Q_\ell (\mathcal{E}^r_S)^{\circ m}(Q_r))/\mathcal{Z}_R(Q_\ell, Q_r) \leq \lambda_S^m \,\mathrm{Tr}(Q_\ell Q_r)/\mathcal{Z}_R(Q_\ell, Q_r) = \lambda_S^m \, p(0).$$

Given this exponentially decaying upper bound, the output of SAMPLE(S^*) on average will consist of $\langle m \rangle = \mathcal{O}(\chi_S)$ calls to SAMPLE(S), where $\chi_S := \lambda_S^{-1}$. Assuming we can obtain a boundary-independent upper bound on the expected length of SAMPLE(S), then this proves that $\langle n_{S^*} \rangle = \mathcal{O}(\chi_S \langle n_S \rangle)$.

If S itself contains expressions with deeply nested Kleene closures then this task becomes difficult, with the above bound translating to $\langle n_R \rangle = \mathcal{O}(\chi^{h_*(R)} L_R)$, for $h_*(R)$ the star height of R and χ the maximum χ_{S_i} among all nested subexpressions $(S_i)^*$ within R. However, if we assume R has bounded star height, then this reduces to $\langle n_R \rangle = \mathcal{O}(\chi^{h_*(R)} L_R) = \mathcal{O}(L_R)$, our desired bound.

Moving on to a consideration of the resource costs of SAMPLE(S^*), we make the inductive assumption that a single call to SAMPLE(S) has average-case runtime of $\mathcal{O}(L_S d D^3)$ and worst-case memory usage of $\mathcal{O}(L_S D^2)$. Algorithm 1 in this case will m samples from S, using the same right boundary condition Q_r^* each time. This leads to regex length, runtime, and memory usage of

$$\begin{aligned} L_R &= L_S + 1 = \mathcal{O}(L_S), \qquad C_R = \mathcal{O}(\langle m \rangle C_S) = \mathcal{O}(\chi_S L_S d D^3) = \mathcal{O}(L_R d D^3), \\ M_R &= M_S = \mathcal{O}(L_S D^2) = \mathcal{O}(L_R D^2), \end{aligned}$$

where the last equality of C_R uses the bounded star height of R. We finally note that the above bound on C_R also applies to the transfer operator $\mathcal{E}^r_{S^*}$, whose action on Q_r can be approximated to arbitrary precision $\epsilon = \exp(\mathcal{O}(-m/\chi_S))$ using m applications of $\mathcal{E}^r_S$.

$\mathbf{R} = \sigma$, for $\sigma =$ c or $\mathbf{\Sigma}$: For the case of $R = c$, no resources are required for sampling. For $R = \Sigma$, the sampling procedure costs $C_\Sigma = \mathcal{O}(dD^3)$, which also gives an upper bound on the cost of applying the transfer operator $\mathcal{E}^r_\sigma$. The regex and output string lengths are both 1 here and no caching is involved, so

$$L_\sigma = 1, \qquad C_\sigma = \mathcal{O}(dD^3) = \mathcal{O}(L_\sigma d D^3), \qquad M_\sigma = 0 = \mathcal{O}(L_\sigma D^2).$$

$\mathbf{R} = \mathbf{R_1 R_2 \cdots R_K}$: When evaluating SAMPLE($R_1 \cdots R_K, Q_\ell, Q_r$), we first compute and cache the K right boundary matrices $Q_r^{(1)}, Q_r^{(2)}, \ldots, Q_r^{(K)}$ in a right-to-left sweep, via the rules $Q_r^{(K)} = Q_r$ and $Q_r^{(i-1)} = \mathcal{E}^r_{R_i}(Q_r^{(i)})$. This has a memory cost of $M_R = \mathcal{O}(KD^2)$. With these right boundary matrices in hand, we then use a left-to-right sweep to obtain strings $s_1, s_2, \ldots, s_K$ via repeated calls to SAMPLE($R_i, Q_\ell^{(i)}, Q_r^{(i)}$), where the left boundary matrices are defined as $Q_\ell^{(1)} = Q_\ell$ and $Q_\ell^{(i+1)} = \mathcal{E}^\ell_{s_i}(Q_\ell^{(i)})$. No caching of the $Q_\ell^{(i)}$ is required, and each call to SAMPLE($R_i, Q_\ell^{(i)}, Q_r^{(i)}$) generally involves some additional memory usage, which is freed immediately afterwards.

Applying our inductive assumption about the runtime and memory usage of each of the transfer operators and SAMPLE calls for the subexpressions $R_1, \ldots, R_K$, we get

$$L_R = \sum_{i=1}^{K} L_{R_i}, \qquad C_R = \mathcal{O}\left(\sum_{i=1}^{K} C_{R_i}\right) = \mathcal{O}\left(\sum_{i=1}^{K} L_{R_i} dD^3\right) = \mathcal{O}(L_R dD^3),$$
$$M_R = \mathcal{O}(KD^2) + \max_i(M_{R_i}) = \mathcal{O}(KD^2) + \mathcal{O}(\max_i(L_{R_i})D^2) = \mathcal{O}(L_R D^2).$$

$\mathbf{R} = \mathbf{R_1}|\mathbf{R_2}|\cdots|\mathbf{R_K}$: To evaluate SAMPLE$(R_1|\cdots|R_K, Q_\ell, Q_r)$, we must first sample a random i from the distribution $p(i) = \mathcal{Z}_{R_i}(Q_\ell, Q_r)/\mathcal{Z}_R(Q_\ell, Q_r)$, then call SAMPLE$(R_i, Q_\ell, Q_r)$. This gives the following characterization of the overall runtime and memory usage

$$L_R = \mathcal{O}\left(\sum_{i=1}^{K} L_{R_i}\right), \qquad C_R = \mathcal{O}\left(\sum_{i=1}^{K} C_{R_i}\right) = \mathcal{O}\left(\sum_{i=1}^{K} L_{R_i} dD^3\right) = \mathcal{O}(L_R dD^3),$$
$$M_R = \max_i(M_{R_i}) = \mathcal{O}(\max_i(L_{R_i})D^2) = \mathcal{O}(L_R D^2).$$

□

D Experimental Details

Table 1: Definition of Tomita grammars 3-7 given in [1], which states a necessary and sufficient condition for a string to belong to each grammar. Tomita grammars 1 and 2 correspond to the respective family of strings 1^n and $(01)^*$, and are unused because of their small size and simple structure.

Tomita #	3	4	5	6	7
Description	Doesn't contain $1^{2n+1}0^{2m+1}$ as a substring	Doesn't contain 000 as a substring	Contains an even number of 0's and 1's	Number of 0's minus number of 1's is a multiple of 3	Has the form $0^*1^*0^*1^*$

The u-MPS model we utilized was built from scratch in JAX [3], while the LSTM and Transformer modules from PyTorch [6] were used for baselines. The code for the experiments can be found at `https://github.com/jemisjoky/umps_code`. The LSTMs are single-layer models with 20 or 50 hidden units (20 or 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 similar chosen to be 20 or 50, and for both types of models, five independent trials were used for each number of hidden states and the model with the lowest validation error was used to produce the sampling statistics reported in Section 6.

For each grammar, the models were trained on either 1,000 or 10,000 randomly chosen strings, with 1,000 strings used as a held-out validation set. The sampling percentages for Table 2 and the sampling tasks of Table 3 were obtained from sampling 1,000 random strings from the respective models, while the completion tasks of Table 3 used 1,000 random strings from a held-out reference set, where the models were used to infer each character in each string when all other characters were used as bidirectional context.

In all experiments, models were trained with gradient descent relative to a negative log likelihood (NLL) loss and Adam optimizer [4]. An initial learning rate of 10^{-2} was used, which was decreased by a factor of 10 each time the validation loss failed to improve for 5 consecutive epochs. In this manner, piecewise constant learning rates of 10^{-2}, 10^{-3}, and 10^{-4} were used, with the next drop in learning rate signalling the end of training.

The u-MPS, HMMs, and Transformers were trained identically for all experiments, with the unidirectional LSTM trained in the same way. The bidirectional LSTM was trained specifically for the string completion task, with the loss 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 each pair of sampling and completion tasks in Table 3, the same trained u-MPS model was used to produce both statistics.

For the dataset of email addresses used in the real-text experiments, we extracted all sender and receiver addresses contained in the CLAIR fraudulent emails dataset [7], giving approximately 4,000 distinct addresses. Training was conducted in a similar manner to the synthetic data experiments, where the regular expression R_e = `[\w-.]+@([\w-]+.)+[\w-][\w-]+` was used to judge the correctness of unconditionally sampled strings.

Our regex sampler and regex regularizer are straightforward recursive implementations of Algorithm 1 and the correspondence in Table 1, respectively. Although these naive implementations would typically lead to a significant overhead compared to implementations specialized for R_e, the use of just-in-time (JIT) compilation within JAX gives a significant reduction in this overhead. Employing JIT in this setting is in fact historically well-motivated, considering that one of the earliest applications of JIT compilation was for identifying text matching regular expressions [8].

References

[1] Yoshua Bengio and Paolo Frasconi. An EM approach to learning sequential behavior. *Advances in Neural Information Processing Systems*, 7, 1994.

[2] Ronald Book, Shimon Even, Sheila Greibach, and Gene Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, 100(2):149–153, 1971.

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[5] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002.

[6] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *Advances in Neural Information Processing Systems*, 2017.

[7] D Radev. Clair collection of fraud email, acl data and code repository. *ADCR2008T001*, 2008.

[8] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.