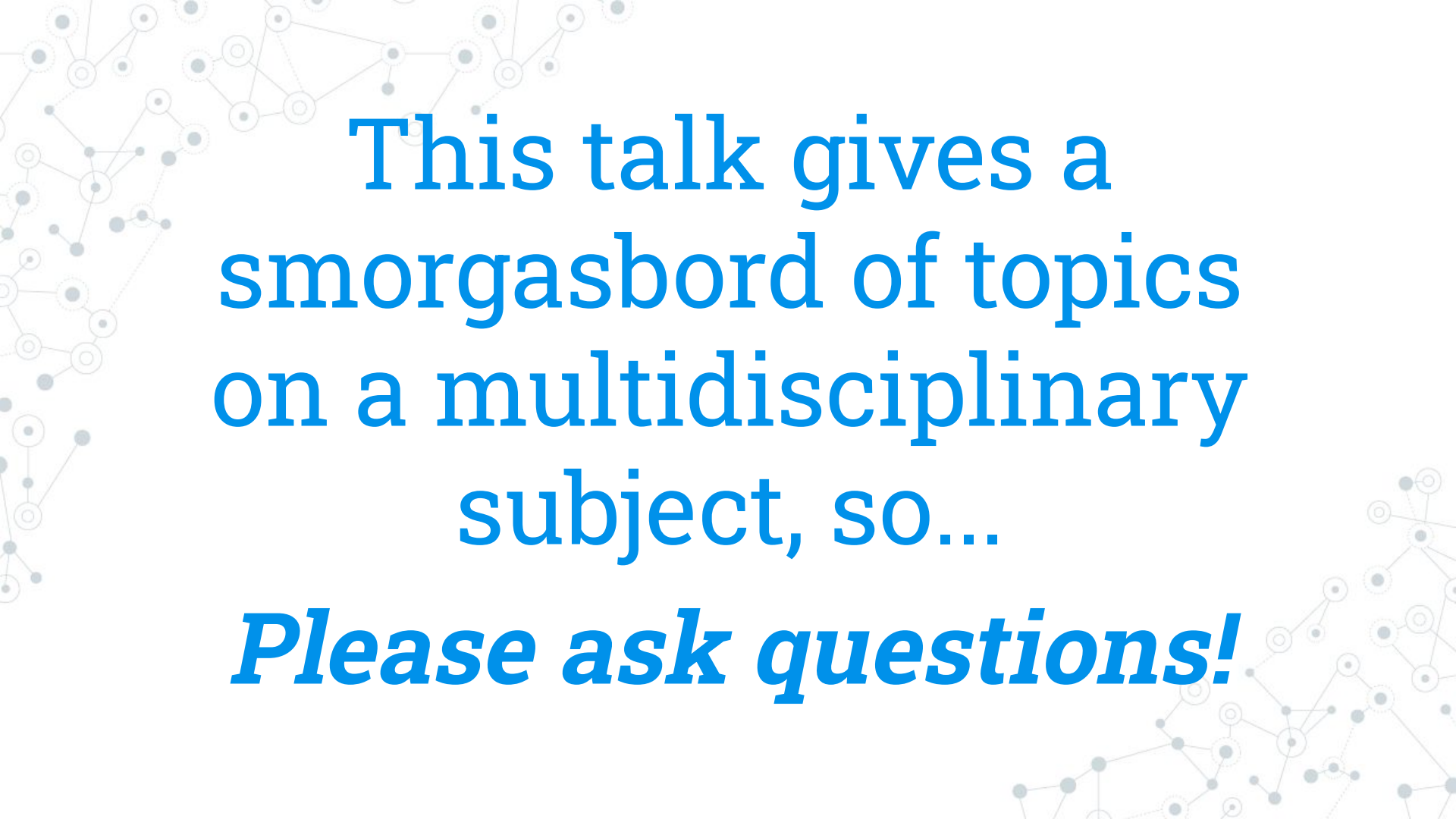


# Tensor Networks, Probabilistic Modeling, and Formal Grammar

*Jacob Miller, Mila (UdeM)*  
*April 1, 2021*

*Largely based on JM, G. Rabusseau, and J. Terilla,  
"Tensor Networks for Probabilistic Sequence Modeling",  
AISTATS 2021 ([arXiv:2003.01039](https://arxiv.org/abs/2003.01039))*

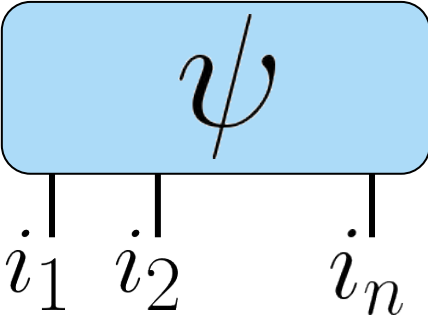
A decorative background featuring a network diagram of interconnected nodes and lines, primarily located on the left and right sides of the slide. The nodes are represented by small circles, some solid and some hollow, connected by thin lines. The overall aesthetic is clean and modern, suggesting a technical or scientific theme.

This talk gives a  
smorgasbord of topics  
on a multidisciplinary  
subject, so...

***Please ask questions!***

# Curse of Dimensionality: Quantum Physics Edition

- Quantum states of  $n$  particles described by  $n$ th order tensor  $\psi$ , generally requires  $\mathbf{O}(d^n)$  params
- Tensor networks** developed to give quasi-local description of  $\psi$  (i.e.  $\mathbf{O}(n)$  params), comes with **intuitive graphical notation**

$$\psi(i_1, i_2, \dots, i_n) =$$


The diagram shows a light blue rounded rectangle containing the symbol  $\psi$ . Three vertical lines extend downwards from the bottom edge of the rectangle, each ending in a dot. Below these dots are the labels  $i_1$ ,  $i_2$ , and  $i_n$  respectively.

*First proposed in Roger Penrose, "Applications of negative dimensional tensors", Combinatorial Mathematics and its Applications (1971)*

# Tensor Diagrams I: Nodes as Tensor Cores

This diagram indicates a tensor with  $n$  modes, equivalent to NumPy:

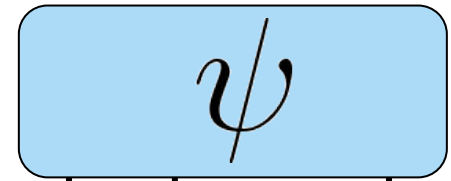
`psi.shape == (d1, d2, ..., dn)`

Values of  $i$ 's on edges give indexing:

`psi[i1, i2, ..., in]`

$$\psi(i_1, i_2, \dots, i_n)$$

=



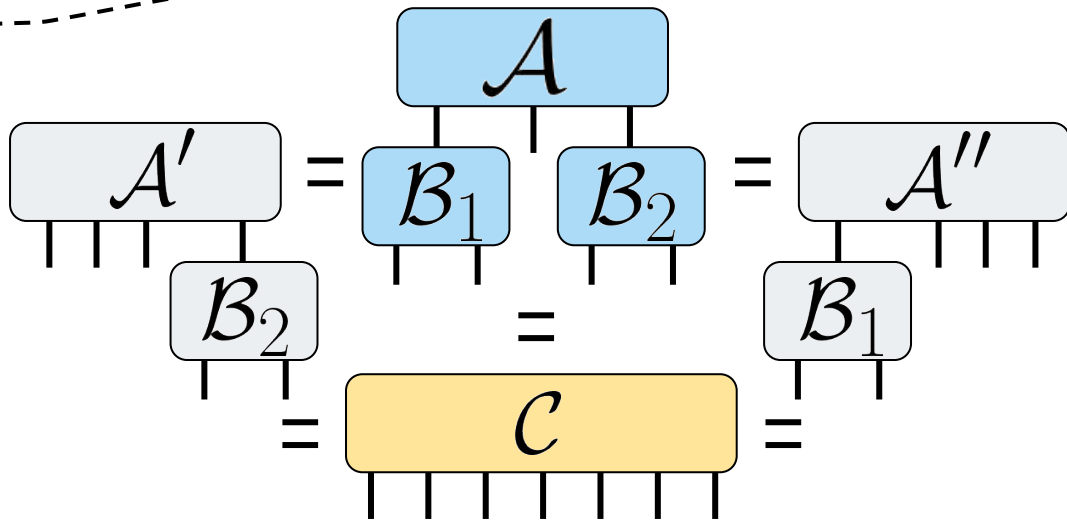
$i_1$   $i_2$   $i_n$

# Tensor Diagrams II: Tensor Contraction

- Tensor contraction indicated by linked edges, which is associative generalization of *vector inner product* and *matrix multiplication*

$$\boxed{A} - \boxed{B} = \boxed{C}$$
$$C = \langle A, B \rangle$$

$$\boxed{A} - \boxed{B} = \boxed{C}$$
$$\boxed{A} - \boxed{B} = \boxed{C}$$
$$C = AB$$



# Tensor Diagrams and Proof Theory

Tensor diagrams are more than just intuitive pictures, they also admit diagram rewriting rules

Different flavors of tensor diagrams come with different ingredients and rewriting rules, which are each sound and complete for some associated (monoidal) category

$$\begin{array}{c} A & C \\ \circlearrowleft f & \circlearrowleft g \\ | & | \\ B & D \\ \circlearrowright h & \\ | & \\ E & E \end{array} \stackrel{\text{id}_B \rightarrow \{}}{=} \begin{array}{c} A & C \\ | & \circlearrowleft g \\ \circlearrowleft f & | \\ B & D \\ | & \circlearrowright h \\ E & E \end{array} = \begin{array}{c} A & C \\ | & \circlearrowleft g \\ \text{id}_A \rightarrow \{ & | \\ \circlearrowleft f & | \\ B & D \\ | & \circlearrowright h \\ E & E \end{array}$$

$(f \cdot \text{id}_B) \otimes (g \cdot h)$        $f \otimes (g \cdot h)$        $(\text{id}_A \cdot f) \otimes (g \cdot h)$

$$\begin{array}{c} C & C & D \\ \uparrow & \uparrow & \uparrow \\ \boxed{g} & & \\ \uparrow & & \\ B & & \\ \uparrow & & \\ \circlearrowleft k & & \\ \uparrow & & \\ \circlearrowleft f & & \\ \uparrow & & \\ A & & \\ \uparrow & & \\ A & & \end{array} \quad = \quad \begin{array}{c} C & C & D \\ \uparrow & \uparrow & \uparrow \\ \boxed{g} & & \boxed{h} \\ \uparrow & & \uparrow \\ \circlearrowleft k & & \circlearrowleft f \\ \uparrow & & \uparrow \\ A & & A \end{array}$$

$(g \circ f) \otimes k \otimes h$        $(g \otimes C \otimes h) \circ ((\zeta \circ (k \otimes f)) \otimes A)$

See Peter Selinger, "A survey of graphical languages for monoidal categories", [arxiv:0908.3347](https://arxiv.org/abs/0908.3347)



*Only the topology of a  
tensor network  
diagram matters,  
please disregard any  
variations in colors,  
drawing styles, etc.*

# Tensor Networks (TNs)

Complex higher-order tensors are parameterized as contraction of smaller **tensor cores**, arranged in architecture-dependent graph

Different TN architectures better suited for different tensors, deep ties to **entanglement** and geometry within quantum many-body physics

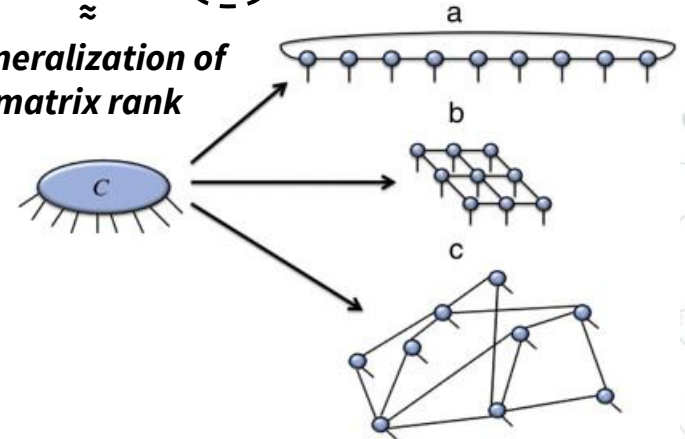
*Simplest Example of TN: Low-rank matrix*

$$\boxed{A} - \boxed{B} = \boxed{C}$$

*Bond dimension*

$\approx$

*Generalization of matrix rank*



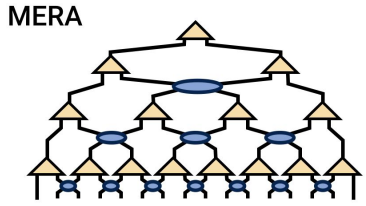
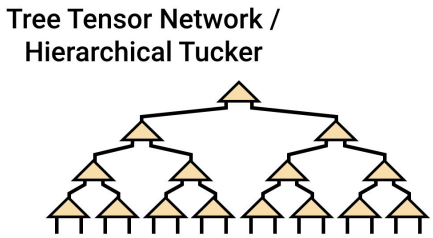
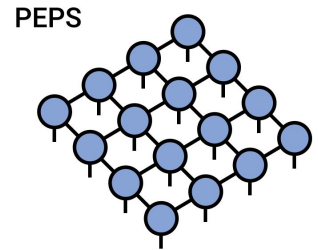
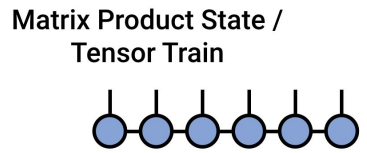
*Orús, Practical Intro to Tensor Networks, Annals of Physics 2014.*



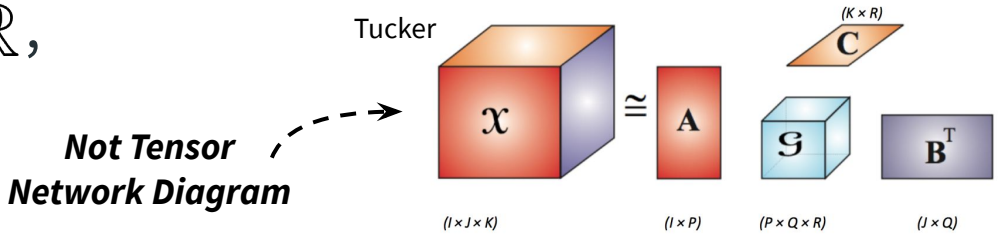
# Different Names, Same TNs

Many TN architectures independently discovered in applied mathematics, with different names

Applied math tends to work with tensors over  $\mathbb{R}$ , and physics over  $\mathbb{C}$



(Miles Stoudenmire, <https://tensornetwork.org/>)



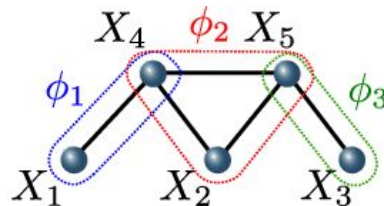
# Another TN Example: PGMs

Probabilistic graphical models (PGMs) are also tensor networks

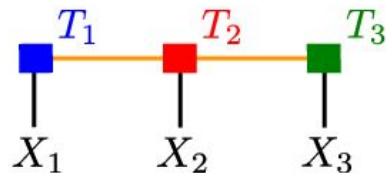
Graphical notation for PGMs is **dual** to that of TNs

This won't be discussed further here, but it's an interesting subject

**PGM-style Notation**  
(variables as nodes)



**TN-style Notation**  
(correlations as nodes)



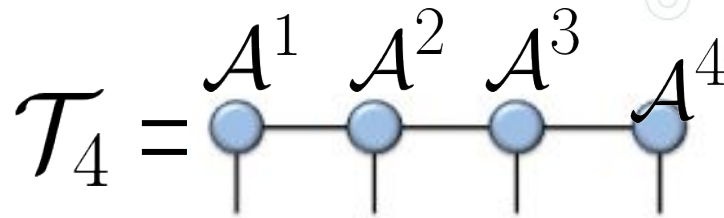
*Ivan Glasser, Nicola Pancotti, and J. Ignacio Cirac, "From probabilistic graphical models to generalized tensor networks for supervised learning", 2020*

For more details, see *Elina Robeva and Anna Seigal, "Duality of Graphical Models and Tensor Networks", 2017.*

# Matrix Product States (MPS)

⊙ MPS defined as contraction of  $n$  third-order cores  $\mathcal{A}^i$  on the line graph, yields  $n$ 'th order tensor  $\mathcal{T}_n$

⊙ Bond dimensions  $D_i$  determine expressiveness of MPS. Arbitrary tensors can be described by MPS with (exponentially) large  $D_i$



⊙ Same model also called **tensor train** (TT) in applied math and ML

# Uniform MPS (uMPS)

- ⊙ MPS w/ identical cores ( $\mathcal{A}^i = \mathcal{A}$ ) are **uniform MPS (uMPS)**, equivalent to **weighted finite automata (WFA)**
- ⊙ Pair of "boundary" vectors  $\alpha, \omega \in \mathbb{C}^D$  needed at edges
- ⊙ Standard MPS defines single fixed-order tensor  $\mathcal{T}_n$ , while uMPS defines  $\mathcal{T}_*$ , the direct sum of  $\mathcal{T}_n$  for all  $n \geq 0$ .  $\mathcal{T}_*$  is equivalent to vector over  $\Sigma^*$  (all strings over alphabet  $\Sigma$ )

Element of the **free tensor algebra** over  $\Sigma$

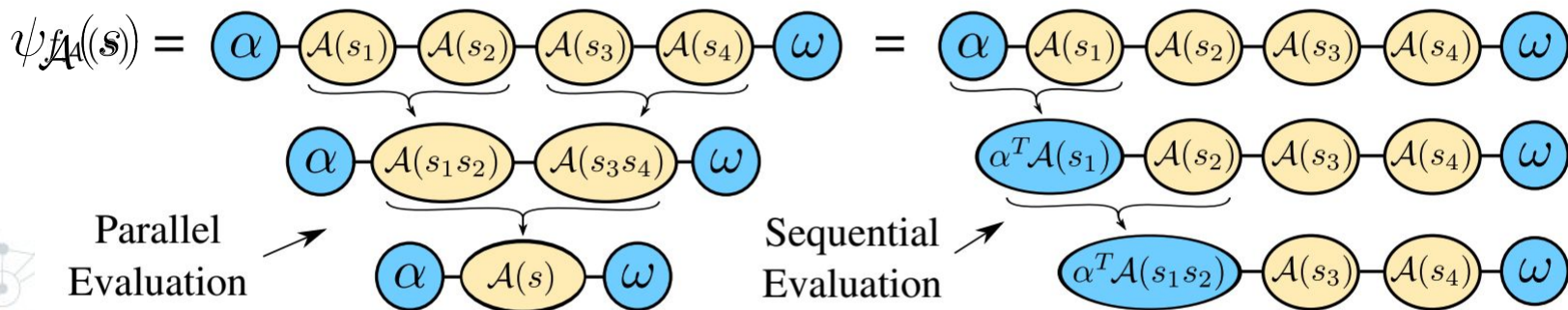
$$\mathcal{T}_* = \bigoplus_{n=0}^{\infty} \mathcal{T}_n = \bigoplus_{n=0}^{\infty} \left( \left\langle \alpha \mid \underbrace{A \cdots A}_n \mid \omega \right\rangle \right)$$

$n$  MPS cores

# uMPS and Parallel Eval

- Although uMPS is sequential model, evaluation of sequences of length  $n$  can be done in parallel time  $\mathcal{O}(\log n)$ , with total computational cost of  $\mathcal{O}(nD^3)$

(vs. parallel. time  $\mathcal{O}(n)$  and cost  $\mathcal{O}(nD^2)$  for sequential eval)



# Probability Distributions are Tensors Too

- ◎ N-gram probability distributions  $P_n(w_1, w_2, \dots, w_n)$  over words  $w_i \in \Sigma$  in finite lexicon are  $n$ 'th order tensors
- ◎ More generally, probability distributions  $P(s)$  over arbitrary strings  $s \in \Sigma^*$  are equiv. to elements of free tensor algebra w/  
(Non-negativity)  $P(s) \geq 0$ , (Normalization)  $\sum_{s \in \Sigma^*} P(s) = 1$

# Tensor Networks for Language Modeling

- ◎ **Idea:** Since probability distributions  $P(s)$  over strings define tensors with same "shape" as those generated by uMPS, so why not use trainable **uMPS as language models?**
- ◎ **Issue:** How can we ensure that our uMPS will generate a valid probability distribution? In general, for uMPS with cores over this problem is *undecidable*



# Classical Solution: Probabilistic Automata

- Choose all cores to take non-negative values in  $\mathbb{R}^{\geq 0}$
- Equivalent to hidden Markov models (HMMs), which have simple (classical) probabilistic semantics

$$\tilde{P}(s) = \left\langle \alpha \mid A \mid A \mid \dots \mid A \mid \omega \right\rangle$$

$$P(s) = \tilde{P}(s) / \mathcal{Z}_n$$

$\mathcal{Z}_n = \sum_{s \in \Sigma^n} \tilde{P}(s)$  is normalization factor giving unnormalized prob. of all length- $n$  strings  $s = s_1 s_2 \dots s_n$



# Quantum Solution: Born Machines (BMs)

- Use TNs to learn synthetic quantum states, probs given by Born rule,  $P(s) = |\psi(s)|^2 / |\psi|^2$

**Problem:** How do we compute the normalization factor  $\mathcal{Z}_n$ ?

$$\tilde{P}(s) = \begin{array}{ccccccc} \alpha^* & \boxed{A^*} & \boxed{A^*} & \cdots & \boxed{A^*} & \omega^* \\ & | & | & & | \\ & s_1 & s_2 & & s_n \\ & | & | & & | \\ \alpha & \boxed{A} & \boxed{A} & \cdots & \boxed{A} & \omega \end{array}$$

$$P(s) = \tilde{P}(s) / \mathcal{Z}_n$$

$$\mathcal{Z}_n = \sum_{s \in \Sigma^n} \tilde{P}(s)$$

\* ZY Han, J Wang, H Fan, L Wang, and P Zhang. Unsupervised gen. modeling using MPS. PRX 2018.

\* AJ Ferris and G Vidal. Perfect sampling with unitary tensor networks. PRB 2012.

\* MJ Zhao and H Jaeger. Norm-observable operator models. Neur Comp 2010.

\* R. Bailly. Quadratic weighted automata. Asian Conf on ML 2011.

\* V. Pestun, J. Terilla, & Y. Vlassopoulos. Language as an MPS. arXiv:1711.01416.

\* S Srinivasan, S Adhikary, JM, G Rabusseau, and B Boots. Quantum TNs, Stochastic Processes, and WFA. arXiv:2010.10653.

# Efficient Normalization

- Normalization factors  $Z_n$  efficiently computable w/ **transfer operators**  $\mathcal{E}$ , maps derived from MPS core tensor  $\mathcal{A}$

$$Z_n = \sum_{s=s_1 \dots s_n \in \Sigma^n} \tilde{P}(s) = \sum_{s \in \Sigma^n} |\alpha^\dagger \mathcal{A}(s) \omega|^2$$

$$= \begin{array}{c} \alpha^* \mathcal{A}^* \mathcal{A}^* \dots \mathcal{A}^* \omega^* \\ \alpha \mathcal{A} \mathcal{A} \dots \mathcal{A} \omega \end{array}$$

$$= \rho_l \mathcal{E} \mathcal{E} \dots \mathcal{E} \rho_r$$

$$= \rho_l \mathcal{E}_{\Sigma^n} \rho_r = (\rho_l | \mathcal{E}_{\Sigma^n} | \rho_r)$$

- To compute normalization, convert sum of products into product of sums using associativity of contraction

# Generalized Transfer Operators

- Similar techniques used w/ WFA, but for strings of all finite lengths

$$\begin{aligned} \text{Length } n: \quad \mathcal{Z}_n &= \sum_{s \in \Sigma^n} \tilde{P}(s) = (\rho_\ell | \mathcal{E}^n | \rho_r) \\ &= (\rho_\ell | \mathcal{E}_{\Sigma^n} | \rho_r) \end{aligned}$$

- Problem:** General subsets of strings  $R$  don't permit efficient summation. For which  $R$  is this generally possible?

$$\begin{aligned} \mathcal{Z} &= \sum_{s \in \Sigma^*} \tilde{P}(s) = (\rho_\ell | \sum_{n=0}^{\infty} \mathcal{E}^n | \rho_r) \\ &= (\rho_\ell | \mathcal{E}_{\Sigma^*} | \rho_r) \end{aligned}$$

$$\begin{aligned} \mathcal{Z}_R &= \sum_{s \in R} \tilde{P}(s) = \sum_{s \in R} |\alpha^\dagger \mathcal{A}(s) \omega|^2 \\ &= (\rho_\ell | \mathcal{E}_R | \rho_r) \end{aligned}$$

# Summing over Regular Subsets

- Normalization  $\mathcal{Z}_R$  of **regular subsets**  $R \subseteq \Sigma^*$  of strings are efficiently computable, w/ *recursive correspondence* between regex  $R$  and generalized transfer operators  $\mathcal{E}_R$

$$\mathcal{Z}_R = \sum_{s \in R} \tilde{P}(s) = (\rho_\ell | \mathcal{E}_R | \rho_r), \text{ where}$$

Regex $R$	$\mathcal{E}_R(Q_r)$
$s$	$\mathcal{A}_s Q_r \mathcal{A}_s^T$
$R_1 R_2$	$\mathcal{E}_{R_1}^r(\mathcal{E}_{R_2}^r(Q_r))$
$R_1   R_2$	$\mathcal{E}_{R_1}^r(Q_r) + \mathcal{E}_{R_2}^r(Q_r)$
$S^*$	$\sum_{n=0}^{\infty} (\mathcal{E}_S^r)^{on}(Q_r)$

# Regular Expression (Regex) Sampling

**Theorem 1:** Consider a uMPS model and (unambiguous) regex  $R$ . Then there exists an *efficient recursive sampling algorithm* returning unbiased samples from  $R$ , with probs  $P(s|s \in R) = P(s)/P(R)$

JM, Guillaume Rabusseau, and John Terilla, "Tensor Networks for Probabilistic Sequence Modeling", AISTATS 2021

## Special Cases of Regex Sampling

Auto-regressive

$$R = p\Sigma^*$$

Sample conditioned on prefix  $p$

Fixed-len

$$R = \Sigma^n$$

Sample string of length exactly  $n$

Providing context

$$R = \Sigma^* s \Sigma^*$$

Sample string of length exactly  $n$

Fill in the blank

$$R = p\Sigma^* s_1 \Sigma^* s_2 \Sigma^* \cdots \Sigma^* s_k$$

Complete string with  $k$  disjoint blanks

# Sequential TNs and Formal Grammars

This is ongoing work, with ties to formal languages, NLP, and maybe programming language theory. If you're curious and/or have ideas for cool applications of such methods, I'd love to talk sometime!

- ◎ **Surprising fact:** The recursive correspondence between transfer operators and regex extends to non-terminal symbols of a CFG, allows for efficient sampling from CF languages

A decorative background featuring a network diagram of interconnected nodes and lines, primarily in shades of gray and blue, located in the corners of the slide.

***Any Questions?***

# Regex Regularization

Also lets us compute *exact* prob of any regular subset  $R$

$P(R) = \mathcal{Z}_R / \mathcal{Z}$  is a differentiable function of uMPS model parameters, can be used in training as **novel regularization loss**

## Examples of regularization losses

$R$  = Strings containing an offensive term

$$\mathcal{L} = P(R)$$

$R$  = Valid variable names in programming language

$$\mathcal{L} = -\log P(R)$$

$R_i$  = Similar but differently gendered sentences (gender bias)

$$\mathcal{L} = |P(R_1) - P(R_2)|$$

Avoided Structure

Desired structure

Equality of Probs