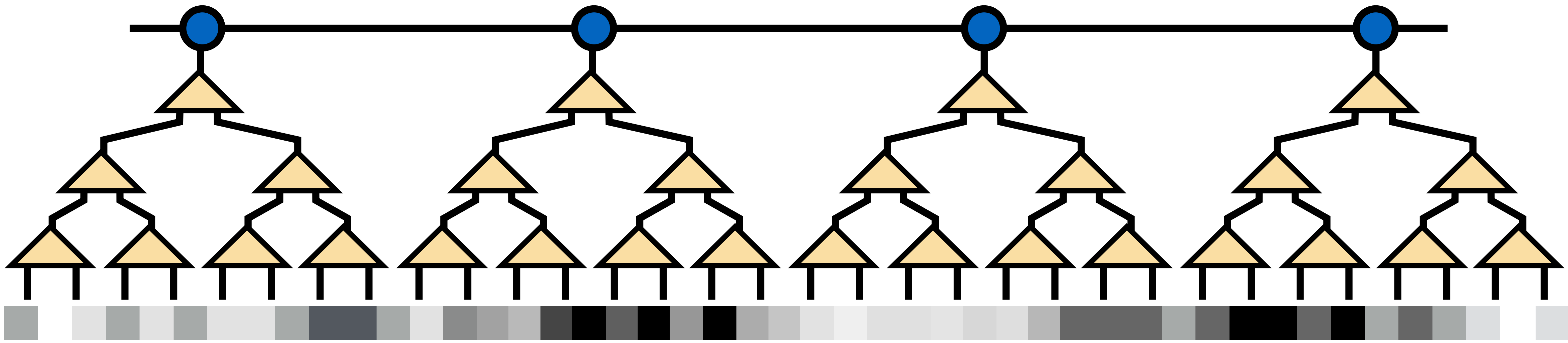
Learning Relevant Features of Data Using Multi-Scale Tensor Networks



E.M. Stoudenmire

Jan 23, 2018 - Santa Fe

FLATIRON INSTITUTE

SIMONS FOUNDATION

Flatiron Institute

FLATIRON INSTITUTE

SIMONS FOUNDATION

The mission of the Flatiron Institute is to advance scientific research through computational methods, including data analysis, modeling and simulation.

CCA: Center for Computational Astrophysics

CCB: Center for Computational Biology

CCQ: Center for Computational Quantum Physics

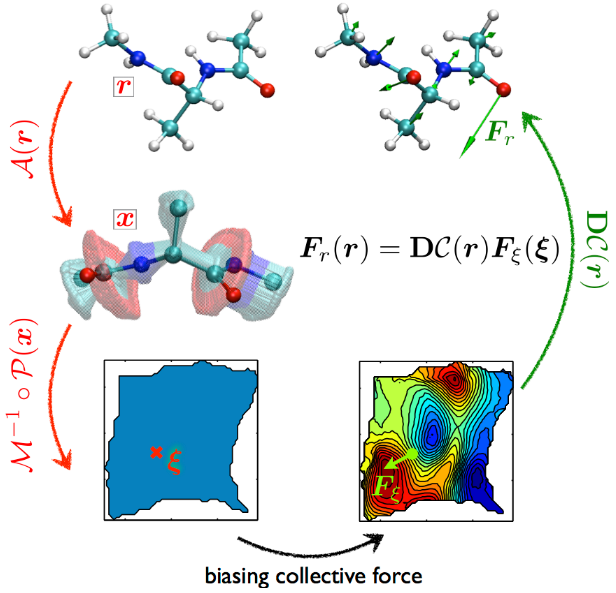
Plus fourth center to be decided

Exciting time for machine learning

Language Processing

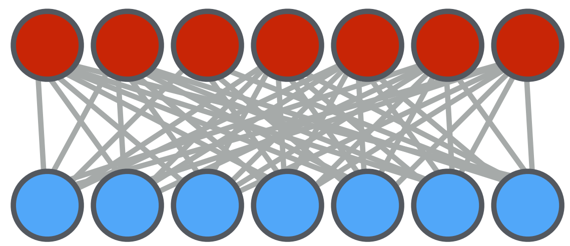
Self-driving cars

Medicine



Materials Science / Chemistry

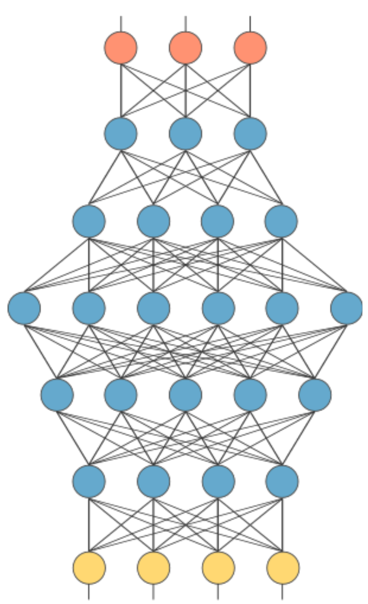
Machine learning has physics in its DNA



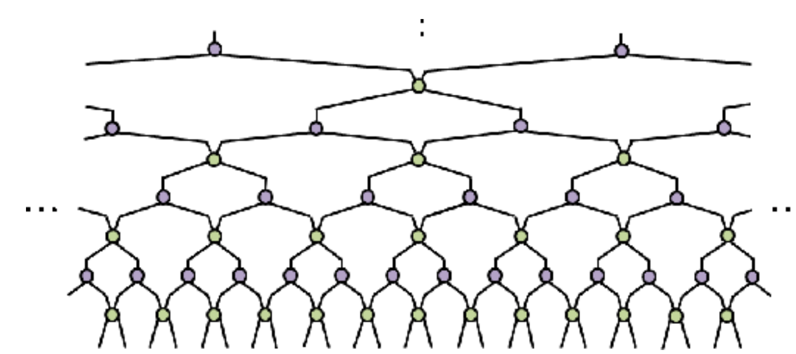
Boltzmann Machines

$\downarrow \uparrow \downarrow \downarrow \uparrow \uparrow \downarrow \uparrow$

Disordered Ising Model



Deep Belief Networks

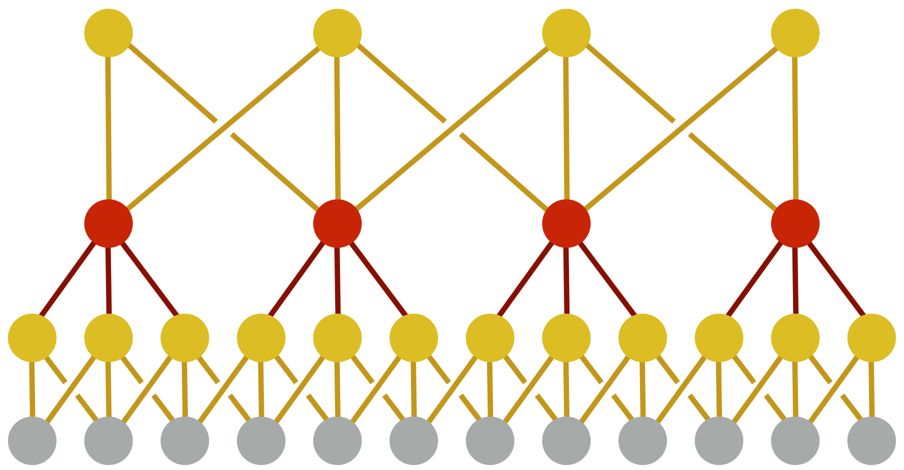


The "Renormalization Group"

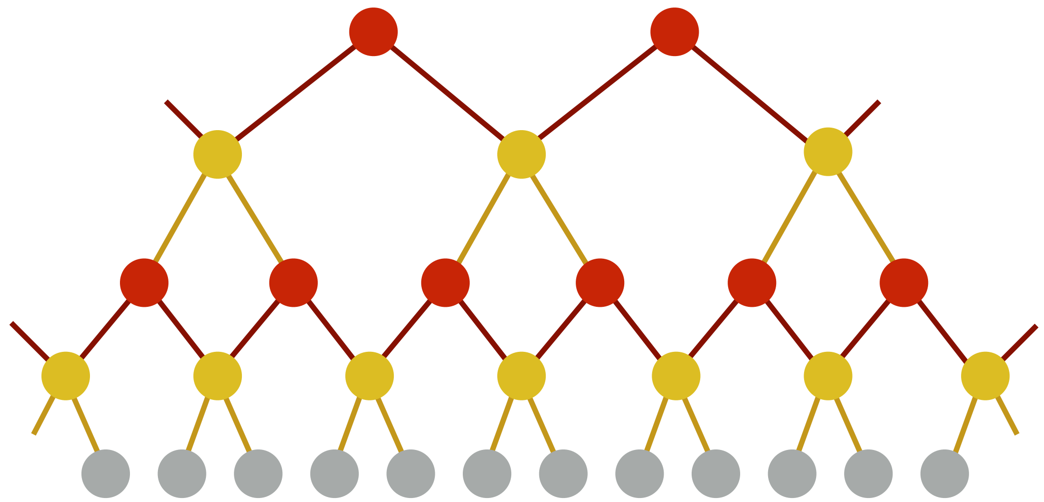
P. Mehta and D.J. Schwab, arxiv:1410.3831

S. Bradde and W. Bialek, arxiv:1610.09733

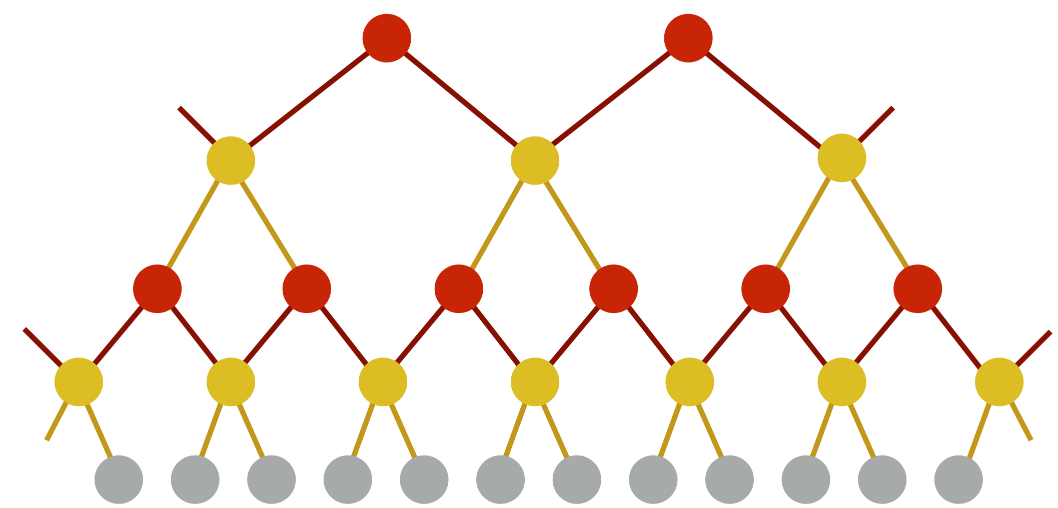
Convolutional neural network



"MERA" tensor network



Are tensor networks useful for
machine learning?



This Talk

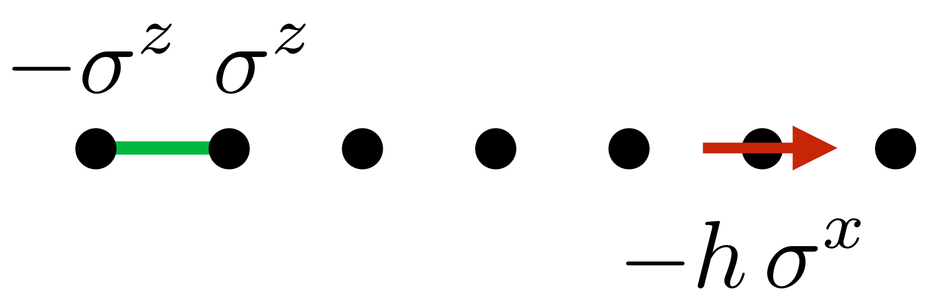
Tensor networks can represent weights of
useful and interesting machine learning models

Flexibility of tensor network algorithms leads
to creativity in devising new approaches

What are Tensor Networks?

Original setting is quantum mechanics

Spin model (transverse field Ising model):



$$\hat{H} = \sum_j (-\sigma_j^z \sigma_{j+1}^z - h\,\sigma_j^x)$$

$\uparrow \ \uparrow \ \uparrow \ \uparrow \ \uparrow \ \uparrow \ \uparrow$ $\quad h \ll 1$

$\rightarrow \ \rightarrow \ \rightarrow \ \rightarrow \ \rightarrow \ \rightarrow \ \rightarrow$ $\quad h \gg 1$

Wavefunction just a rule to
map spin configurations to numbers $\Psi^{s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8}$

$$\uparrow\downarrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow \longrightarrow \Psi^{\uparrow\downarrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow}$$

$$\uparrow\downarrow\uparrow\downarrow\uparrow\uparrow\uparrow\uparrow \longrightarrow \Psi^{\uparrow\downarrow\uparrow\downarrow\uparrow\uparrow\uparrow\uparrow}$$

$$\uparrow\uparrow\uparrow\uparrow\downarrow\uparrow\downarrow\uparrow \longrightarrow \Psi^{\uparrow\uparrow\uparrow\uparrow\downarrow\uparrow\downarrow\uparrow}$$

$$\uparrow\downarrow\downarrow\downarrow\downarrow\uparrow\uparrow\uparrow \longrightarrow \Psi^{\uparrow\downarrow\downarrow\downarrow\downarrow\uparrow\uparrow\uparrow}$$

Simplest rule: store every amplitude separately

Let's make a different rule

Introduce matrices, one for each spin

$$\uparrow \quad \longrightarrow \quad M^{\uparrow} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \end{bmatrix}$$

$$\downarrow \quad \longrightarrow \quad M^{\downarrow} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \end{bmatrix}$$

Östlund, Rommer, Phys. Rev. Lett. **75**, 3537 (1995)

Compute amplitude by multiplying matrices together (with boundary vectors v_L and v_R)

$$\Psi^{\uparrow\downarrow\uparrow\uparrow\downarrow} \approx v_L^\dagger \, M^{\uparrow} M^{\downarrow} M^{\uparrow} M^{\uparrow} M^{\downarrow} v_R$$

$$\Psi^{\uparrow\uparrow\downarrow\downarrow\downarrow} \approx v_L^\dagger \, M^{\uparrow} M^{\uparrow} M^{\downarrow} M^{\downarrow} M^{\downarrow} v_R$$

$$\Psi^{\uparrow\downarrow\downarrow\uparrow\uparrow} \approx v_L^\dagger \, M^{\uparrow} M^{\downarrow} M^{\downarrow} M^{\uparrow} M^{\uparrow} v_R$$

Östlund, Rommer, Phys. Rev. Lett. **75**, 3537 (1995)

This rule is called a *matrix product state* (MPS)

$$\Psi^{s_1 s_2 s_3 s_4} = v_L^\dagger M^{s_1} M^{s_2} M^{s_3} M^{s_4} v_R$$

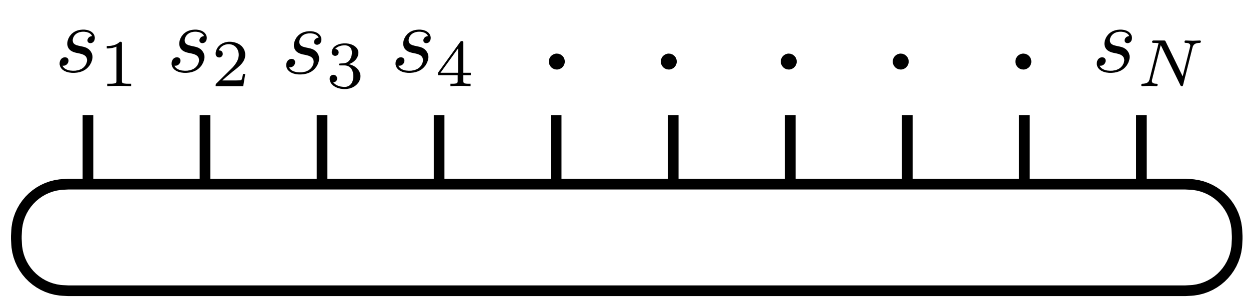
- Size of matrices called m (the "bond dimension")
- For m = $2^{N/2}$ can represent any state of N spins
- Really just a way of compressing a big tensor

Represents 2^N amplitudes using only
(2 N m^2) parameters

Tensor Diagrams

Helpful to draw N-index tensor as blob with
N lines

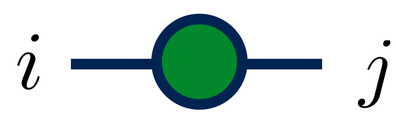
$$\Psi^{s_1 s_2 s_3 \cdots s_N} =$$



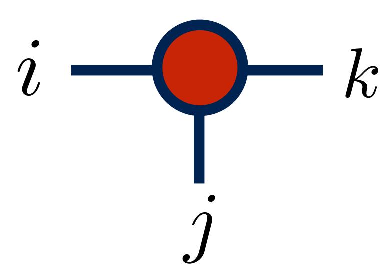
No symmetries, transformation properties assumed

Diagrams for simple tensors

v_j

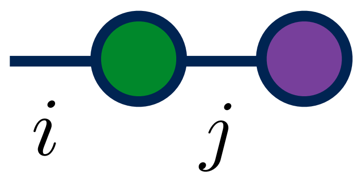


M_{ij}

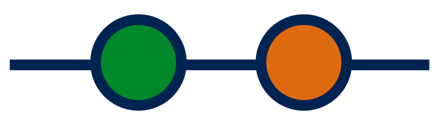


T_{ijk}

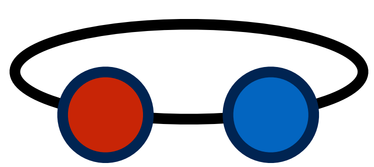
Joining lines implies contraction, can omit names



$$\longleftrightarrow \qquad \sum_j M_{ij} v_j$$



$$\longleftrightarrow \qquad A_{ij} B_{jk} = AB$$

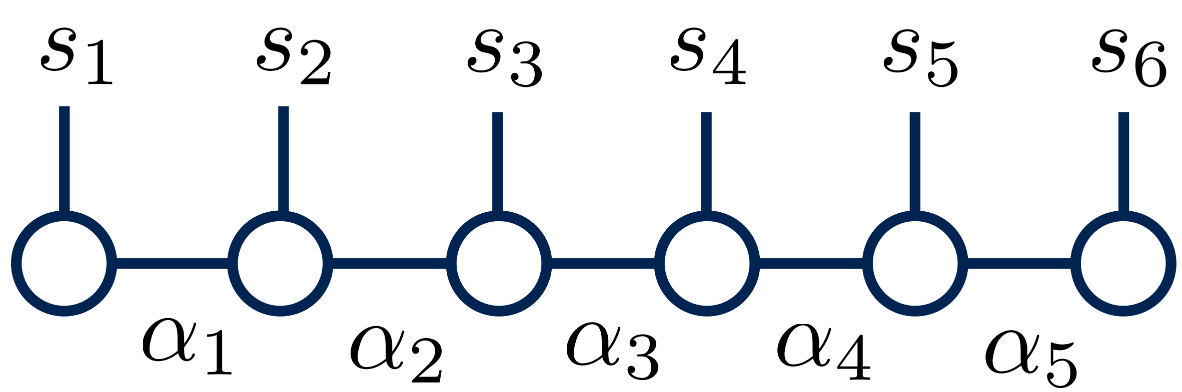


$$\longleftrightarrow \qquad A_{ij} B_{ji} = \mathrm{Tr}[AB]$$

Matrix product state in diagram notation

$$\Psi^{s_1 s_2 s_3 s_4 s_5 s_6} = \sum_{\alpha} M^{s_1}_{\alpha_1} M^{s_2}_{\alpha_1 \alpha_2} M^{s_3}_{\alpha_2 \alpha_3} M^{s_4}_{\alpha_3 \alpha_4} M^{s_5}_{\alpha_4 \alpha_5} M^{s_6}_{\alpha_5}$$

$$=$$

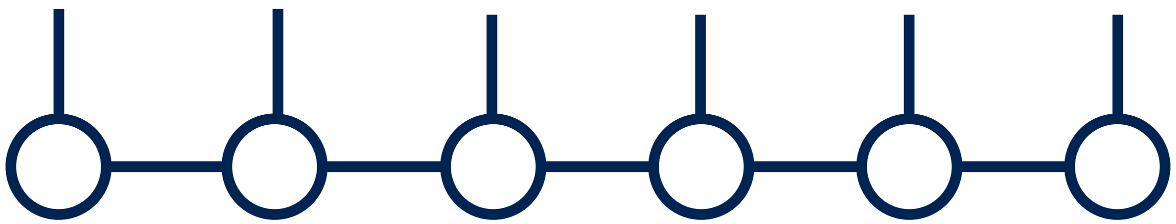


Can suppress index names, very convenient

Matrix product state in diagram notation

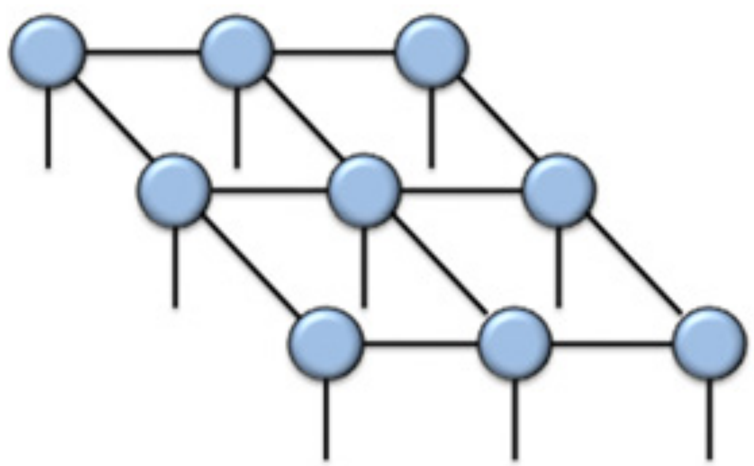
$$\Psi^{s_1 s_2 s_3 s_4 s_5 s_6} = \sum_{\alpha} M^{s_1}_{\alpha_1} M^{s_2}_{\alpha_1 \alpha_2} M^{s_3}_{\alpha_2 \alpha_3} M^{s_4}_{\alpha_3 \alpha_4} M^{s_5}_{\alpha_4 \alpha_5} M^{s_6}_{\alpha_5}$$

$=$



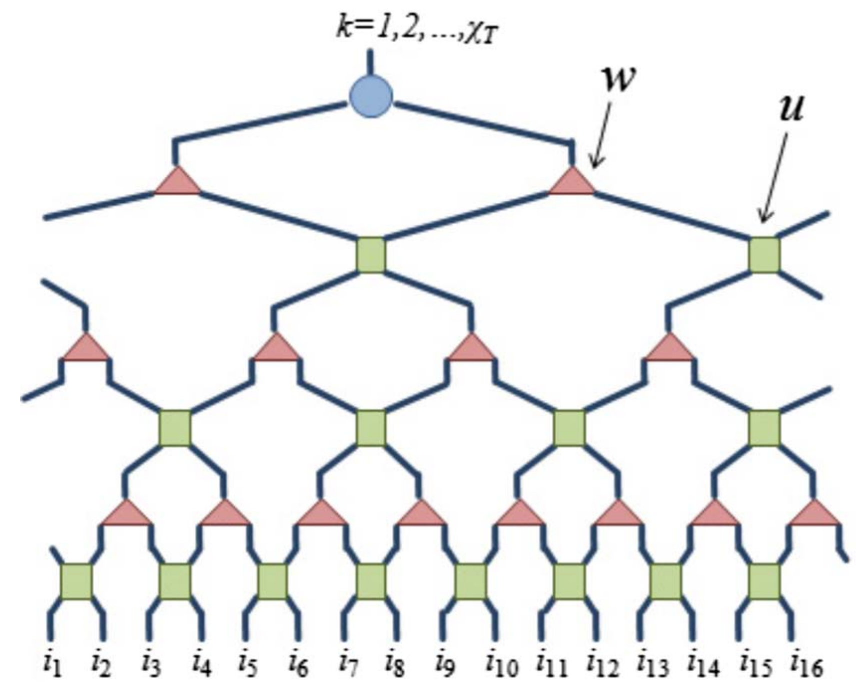
Can suppress index names, very convenient

Besides MPS, other successful tensor are PEPS and MERA



PEPS

(2D systems)



MERA

(critical systems)

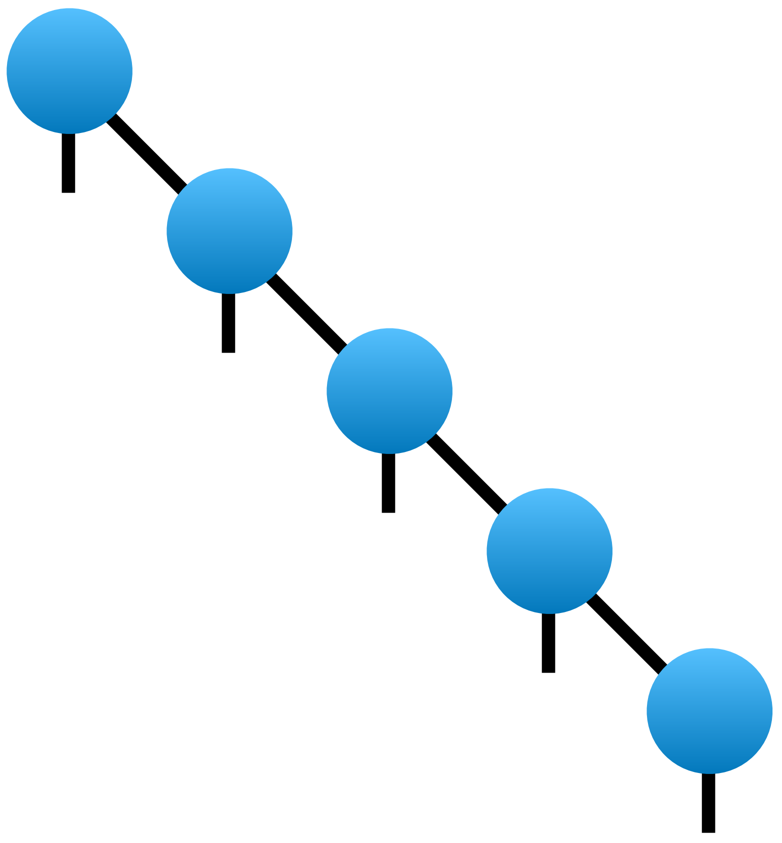
Evenbly, Vidal, PRB **79**, 144108 (2009)

Verstraete, Cirac, cond-mat/0407066 (2004)

Orus, Ann. Phys. **349**, 117 (2014)

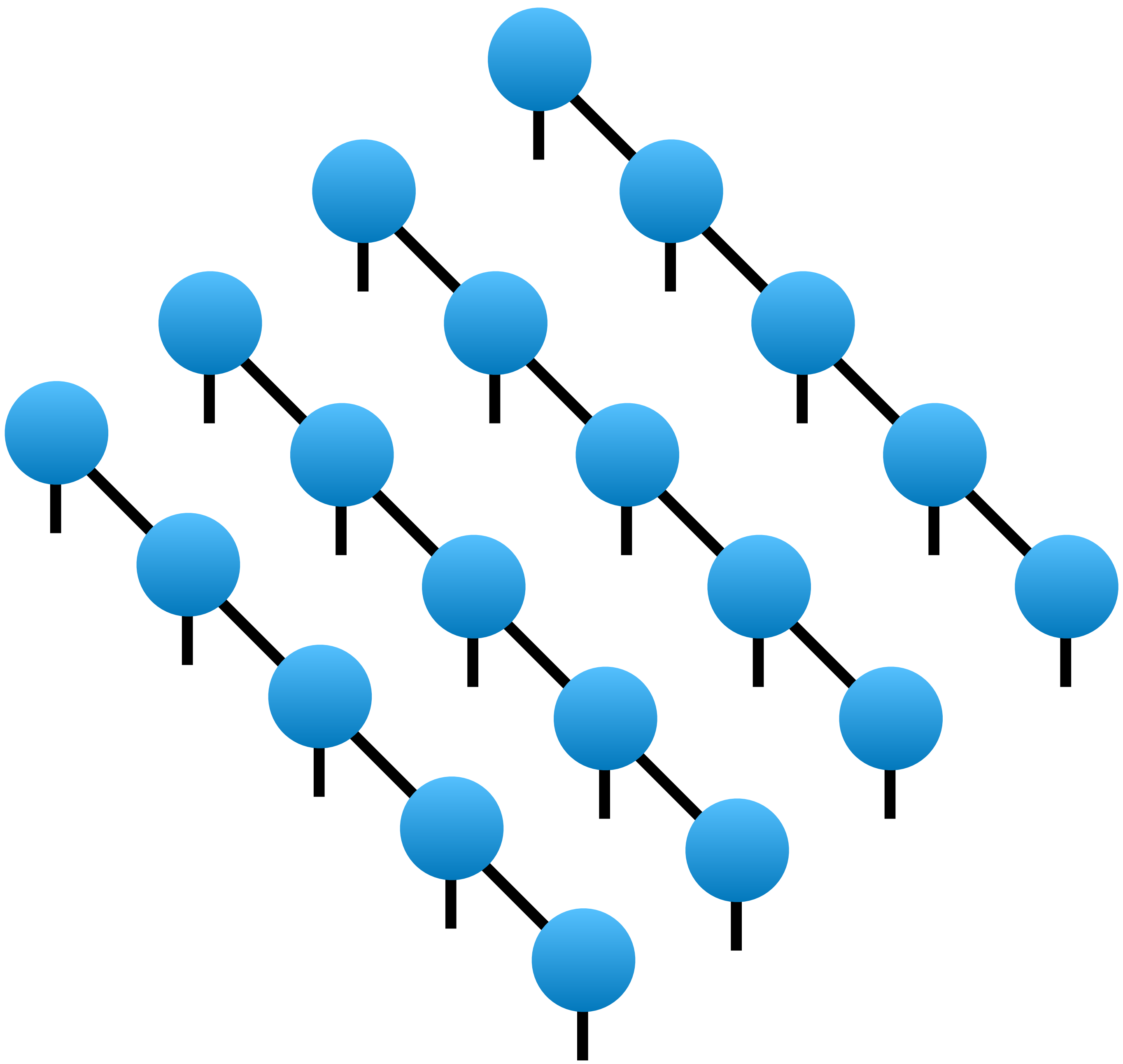
PEPS Tensor Network

Most straightforward extension of matrix product states to two-dimensional lattices



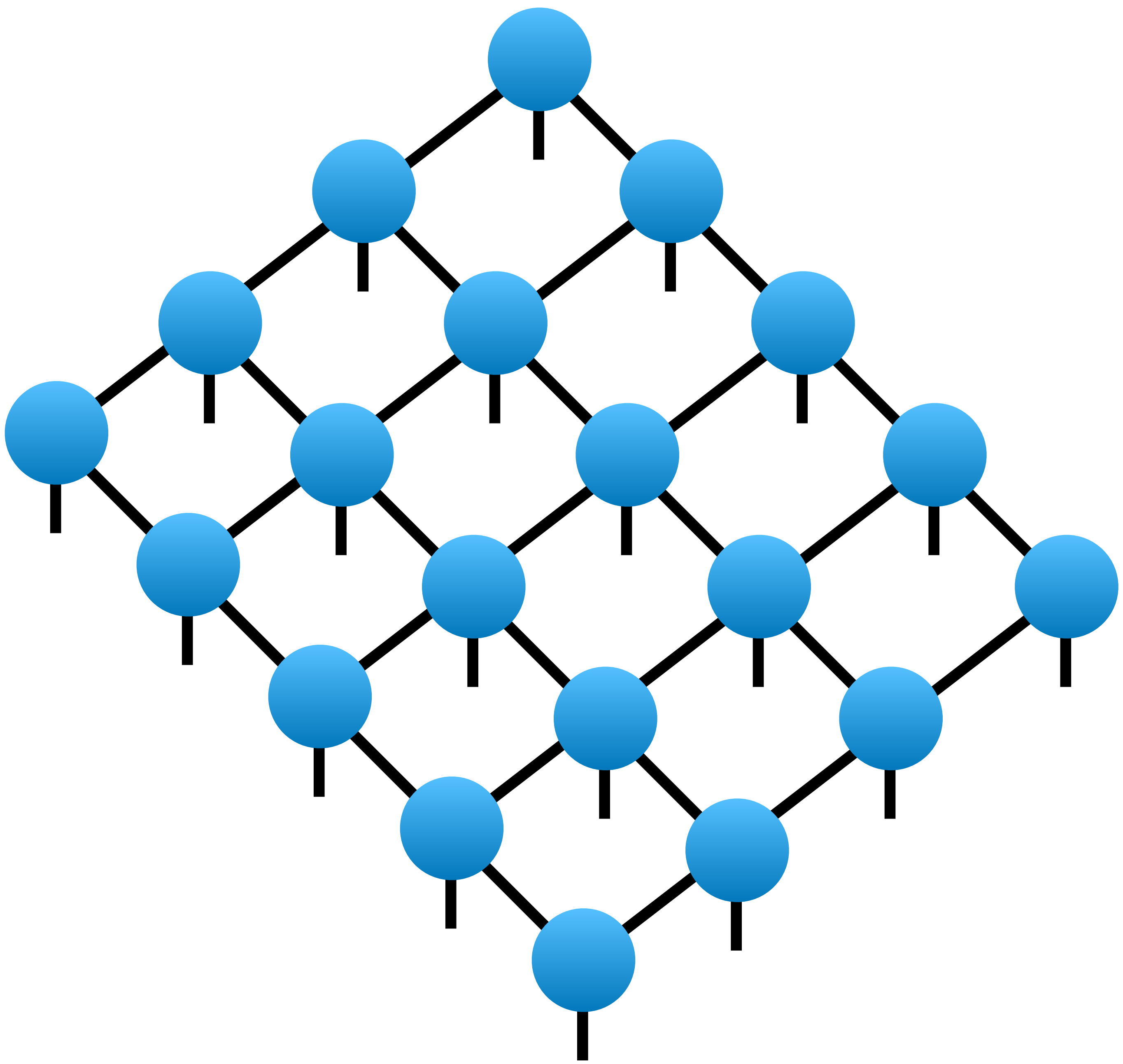
PEPS Tensor Network

Most straightforward extension of matrix product states to two-dimensional lattices



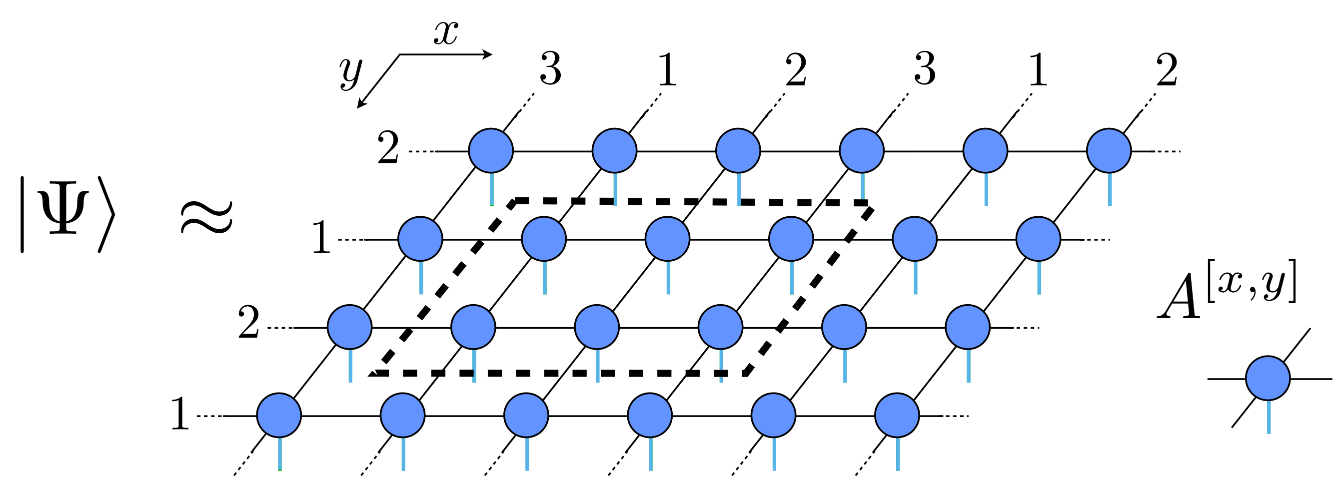
PEPS Tensor Network

Most straightforward extension of matrix product states to two-dimensional lattices



PEPS Tensor Network

Powerful algorithms to address
infinite 2D systems



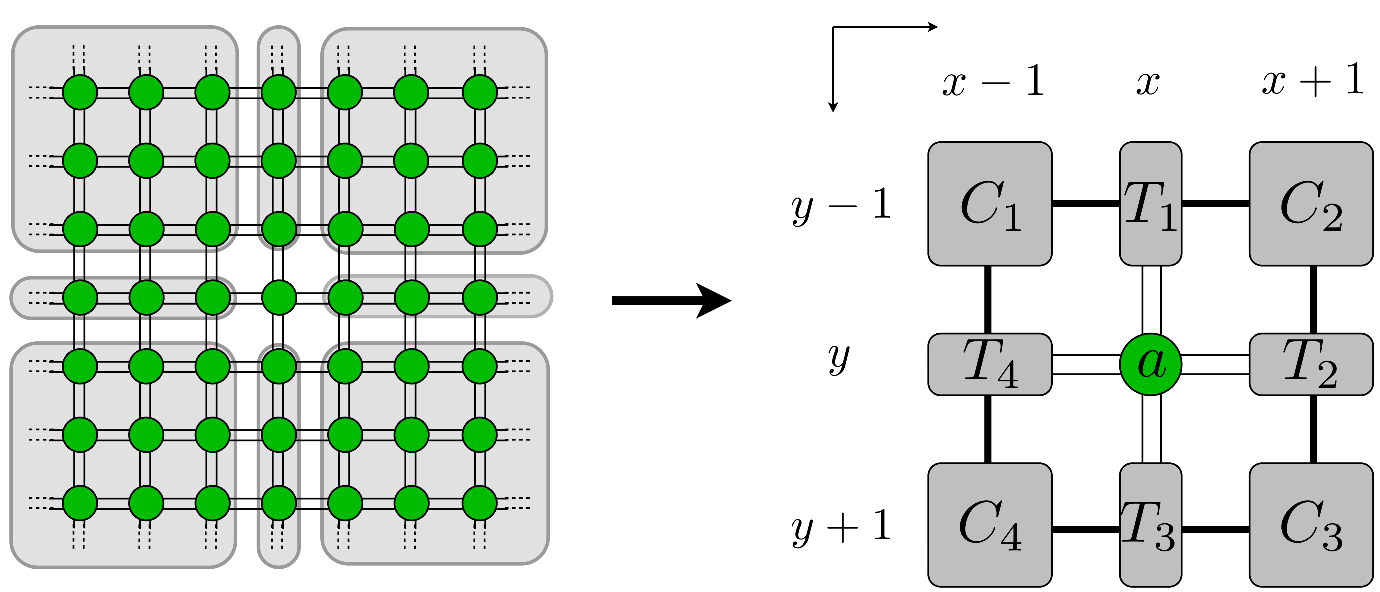
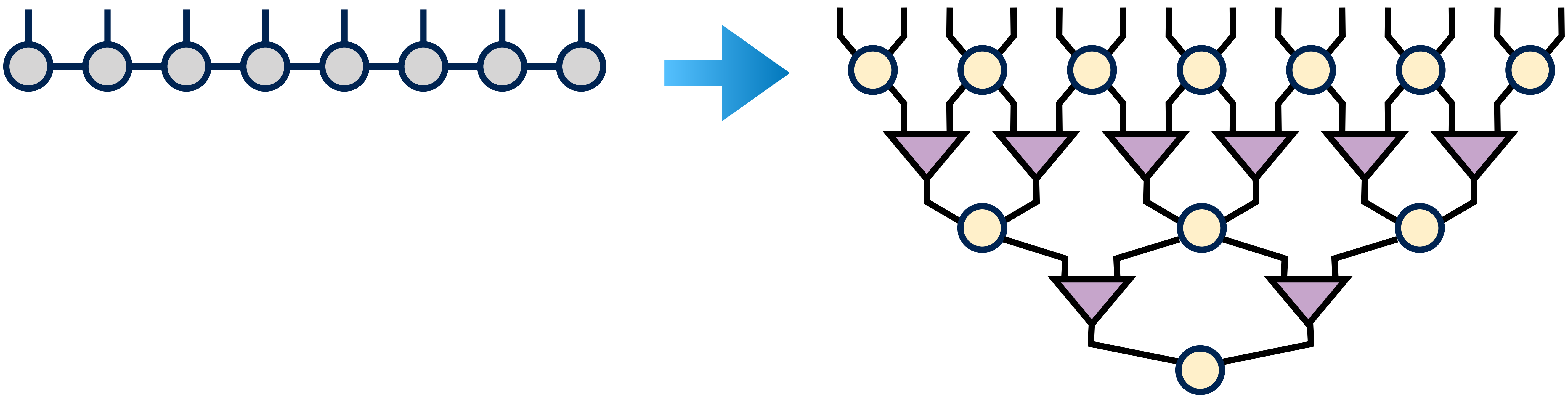


Figure from: Corboz, PRB 94, 035133 (2016)

MERA Tensor Network

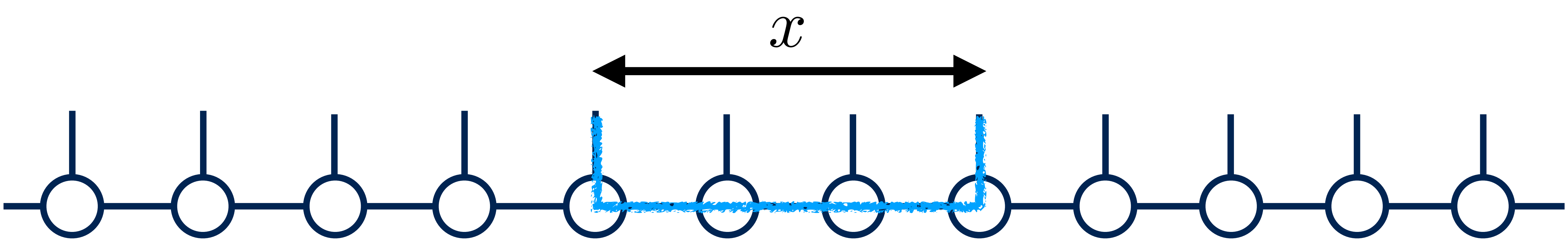
The MERA tensor network generalizes matrix product state to a layered structure



Similar to dilated conv net in machine learning

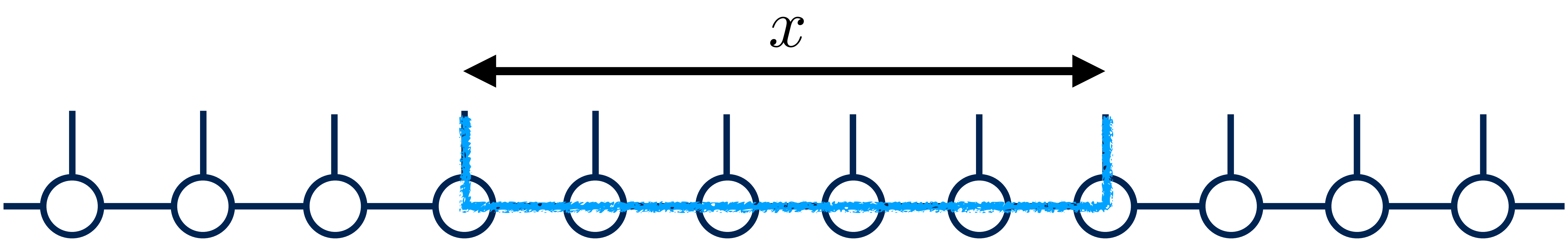
MERA Tensor Network

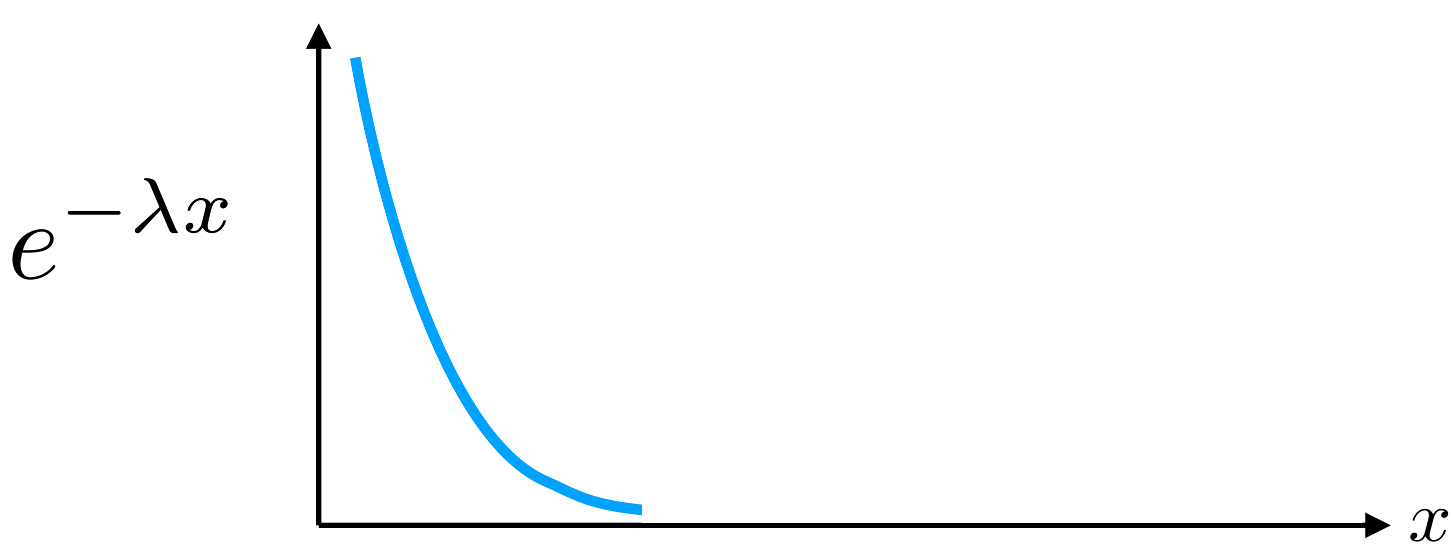
Matrix product state captures only exponential correlations



MERA Tensor Network

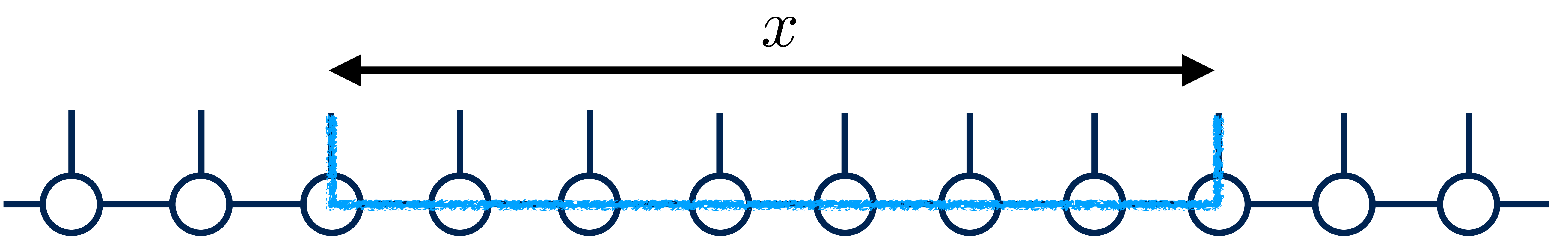
Matrix product state captures only exponential correlations

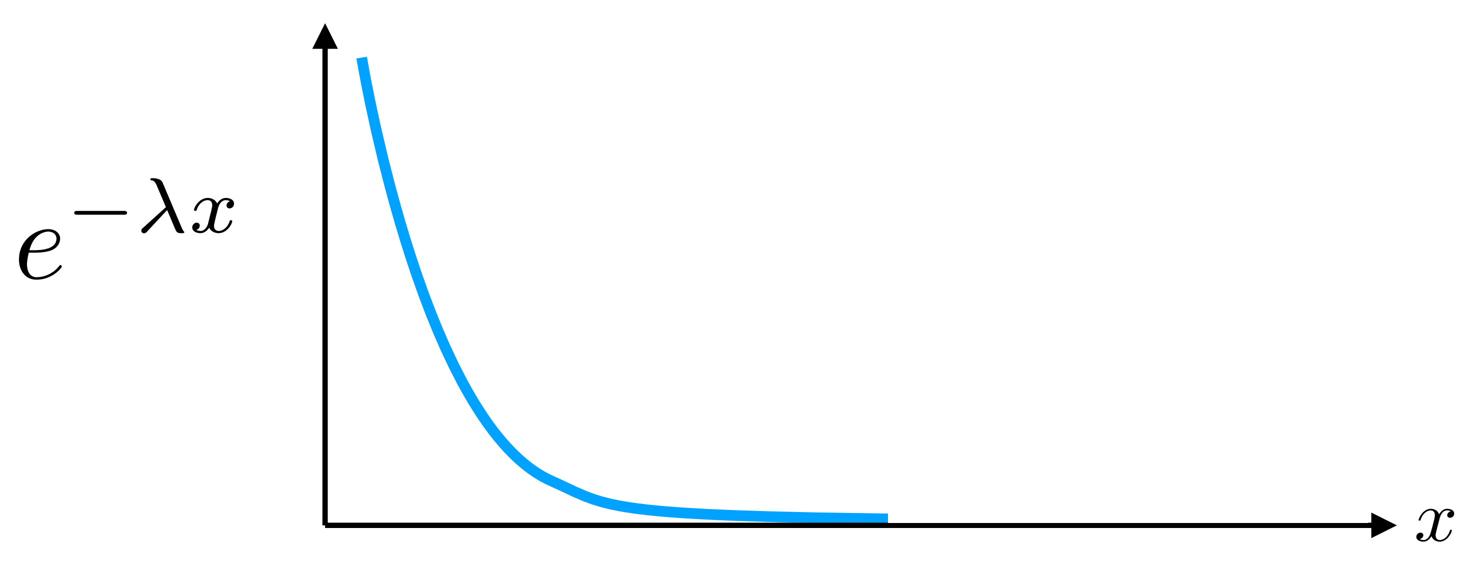




MERA Tensor Network

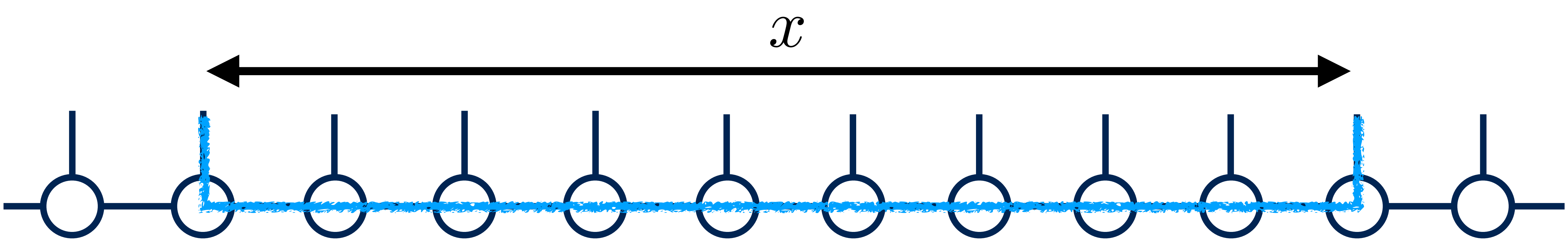
Matrix product state captures only exponential correlations

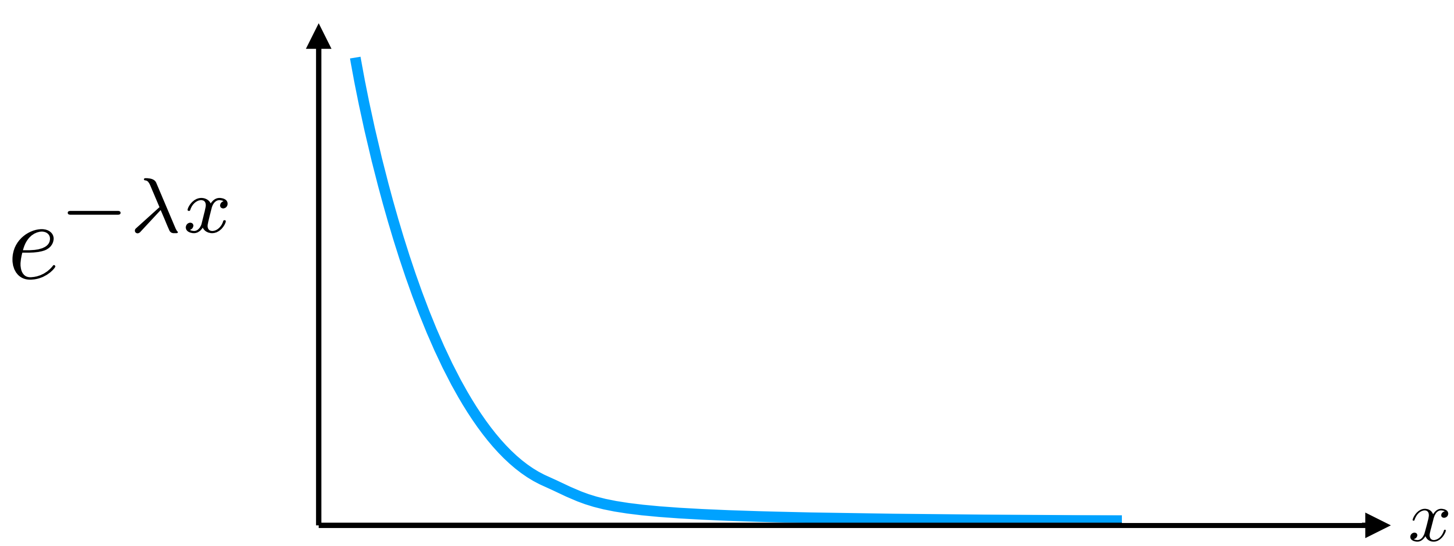




MERA Tensor Network

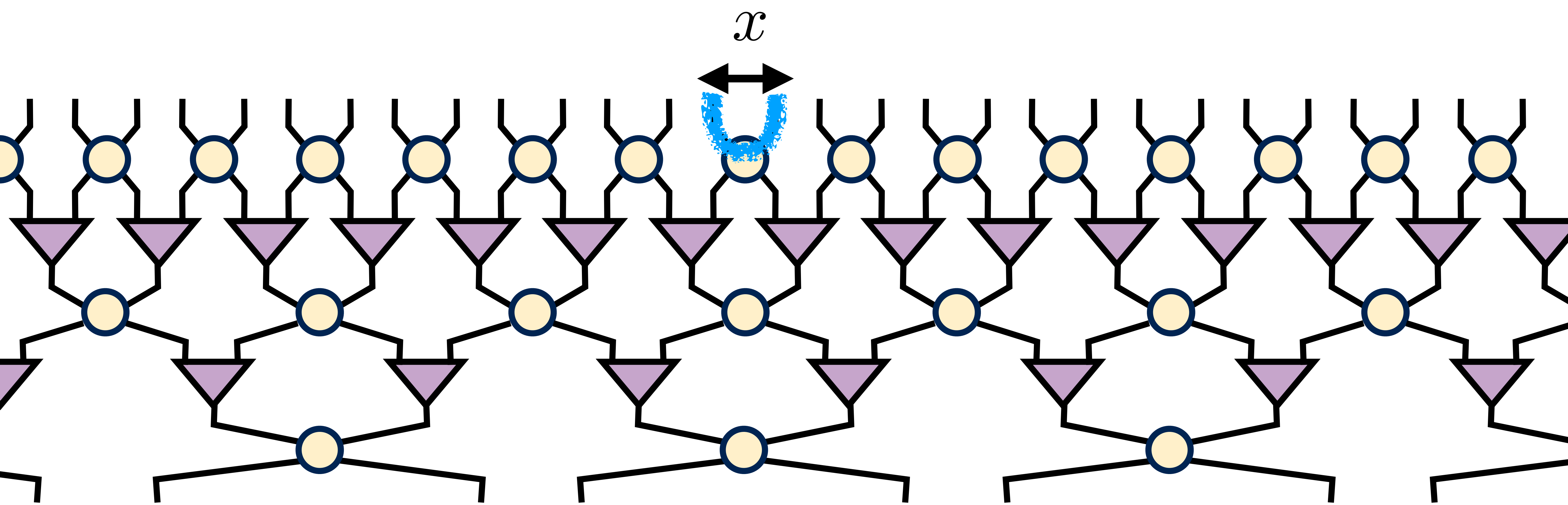
Matrix product state captures only exponential correlations

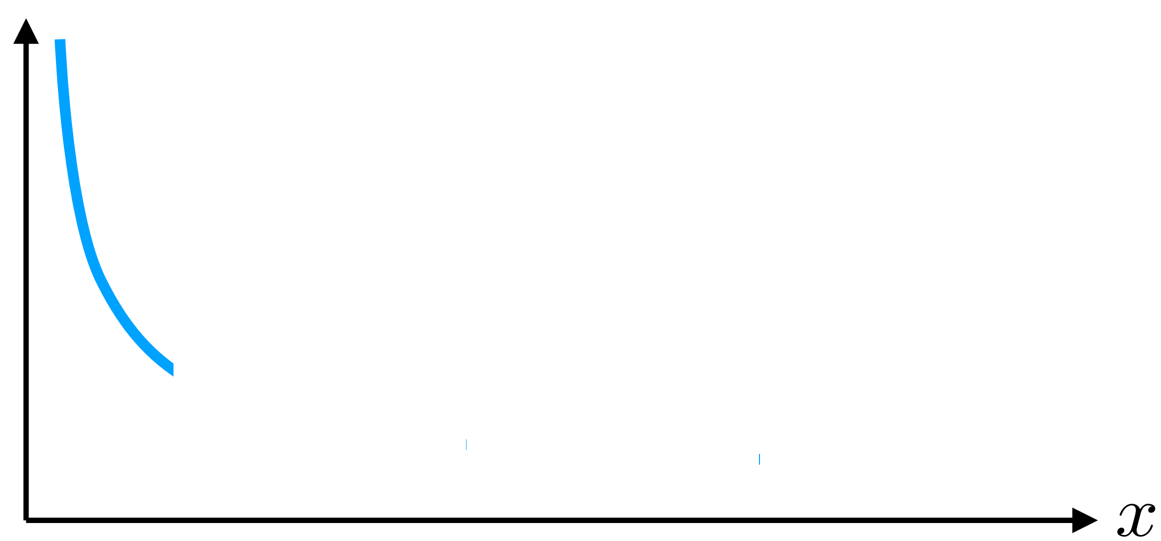




MERA Tensor Network

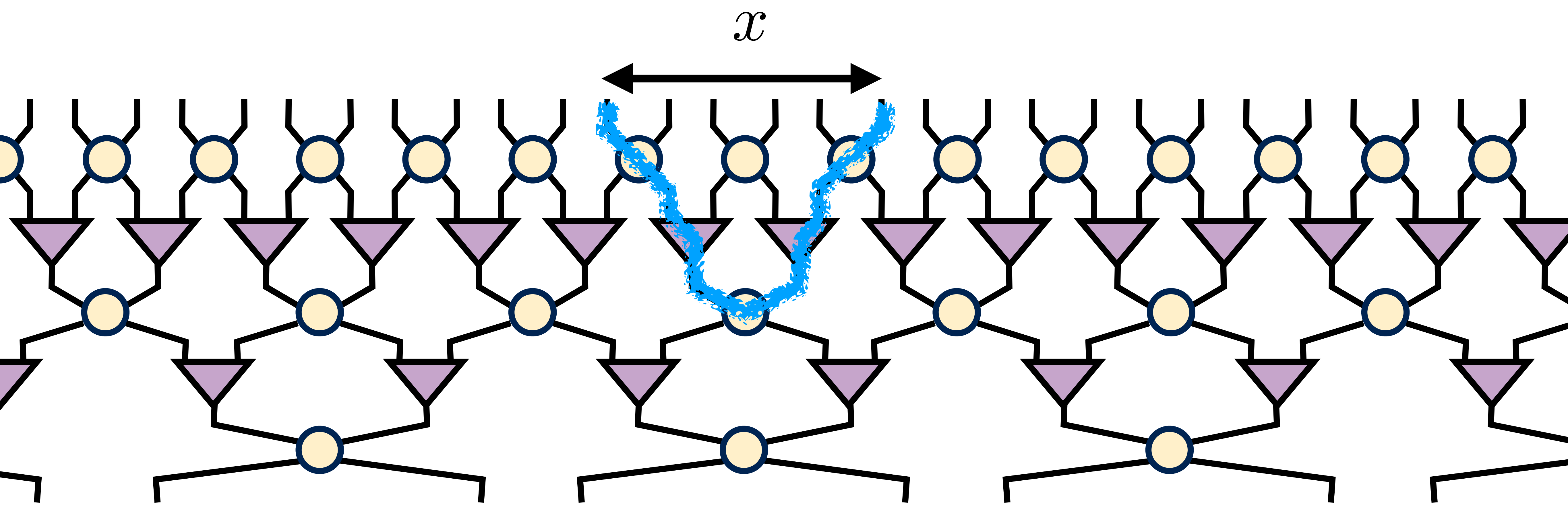
MERA layered architecture captures
power-law correlations

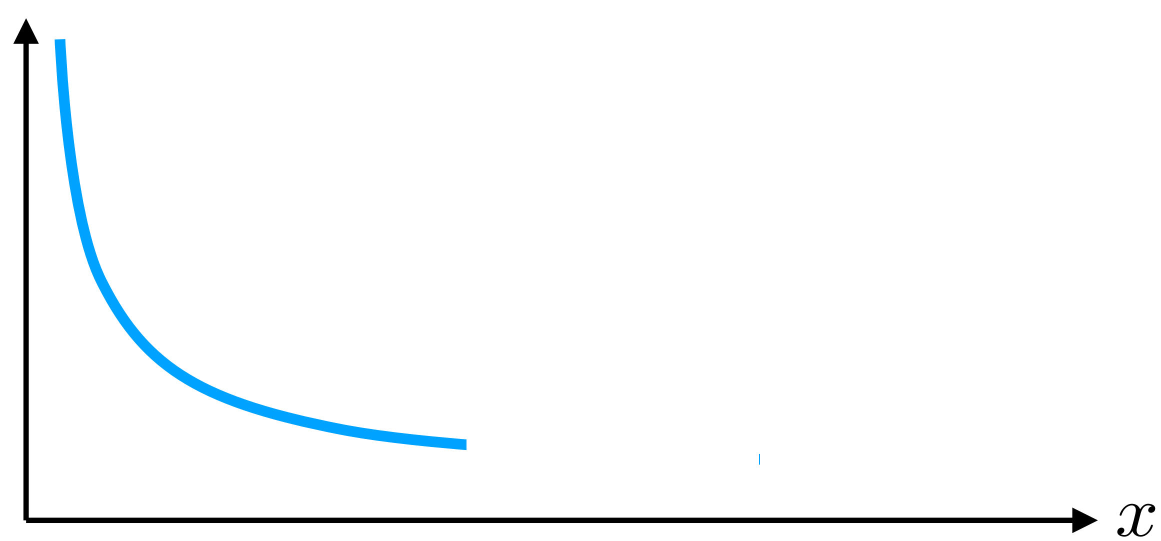




MERA Tensor Network

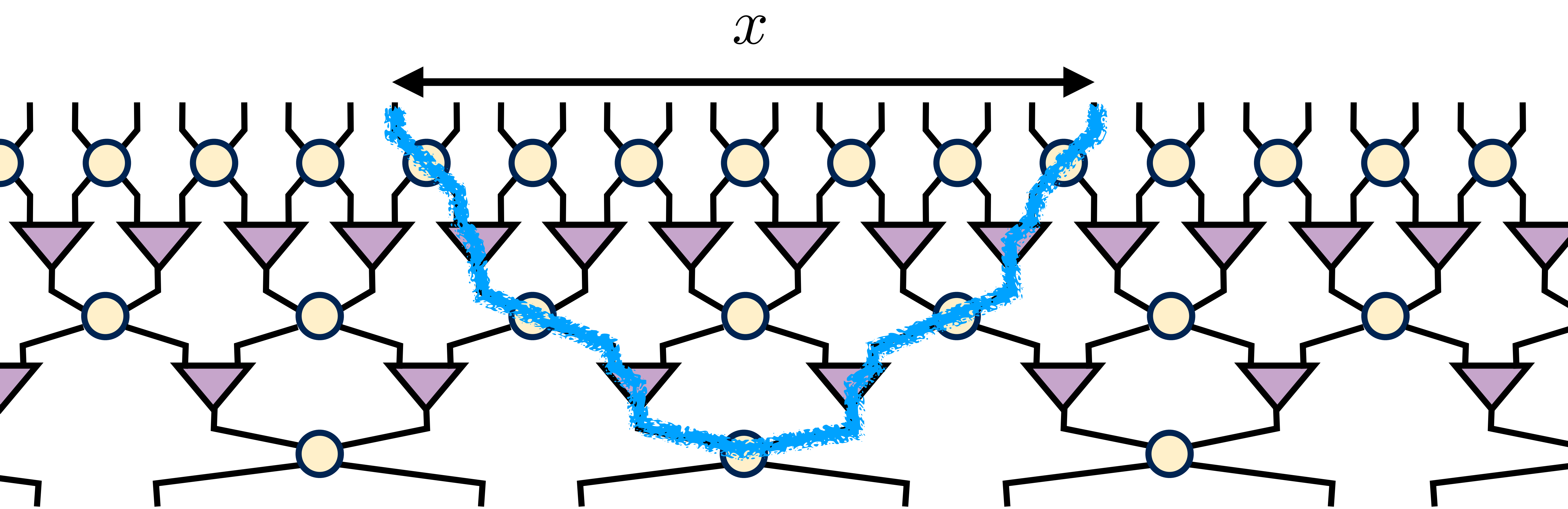
MERA layered architecture captures
power-law correlations

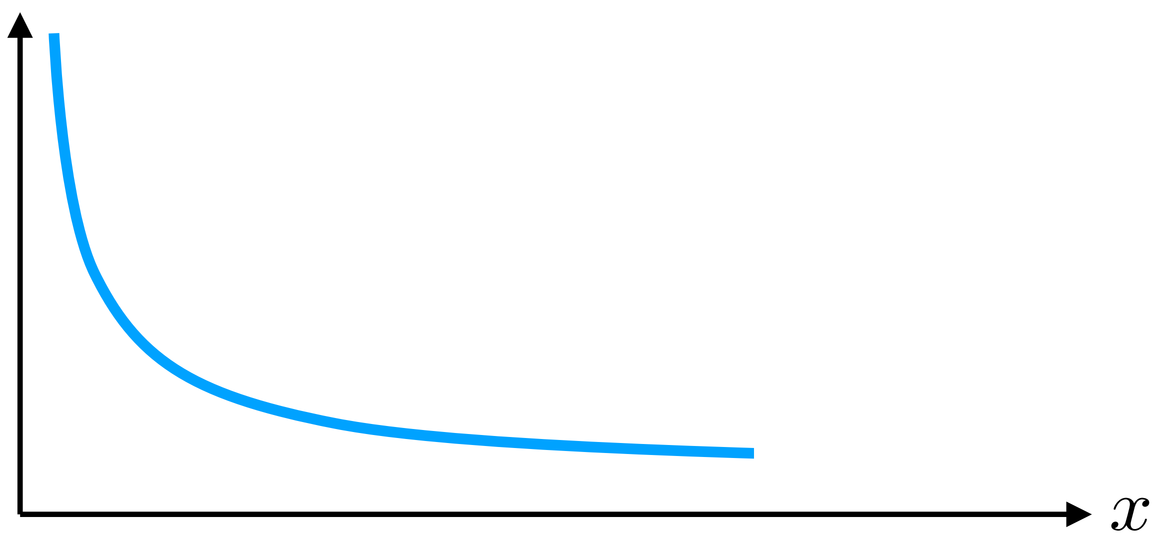




MERA Tensor Network

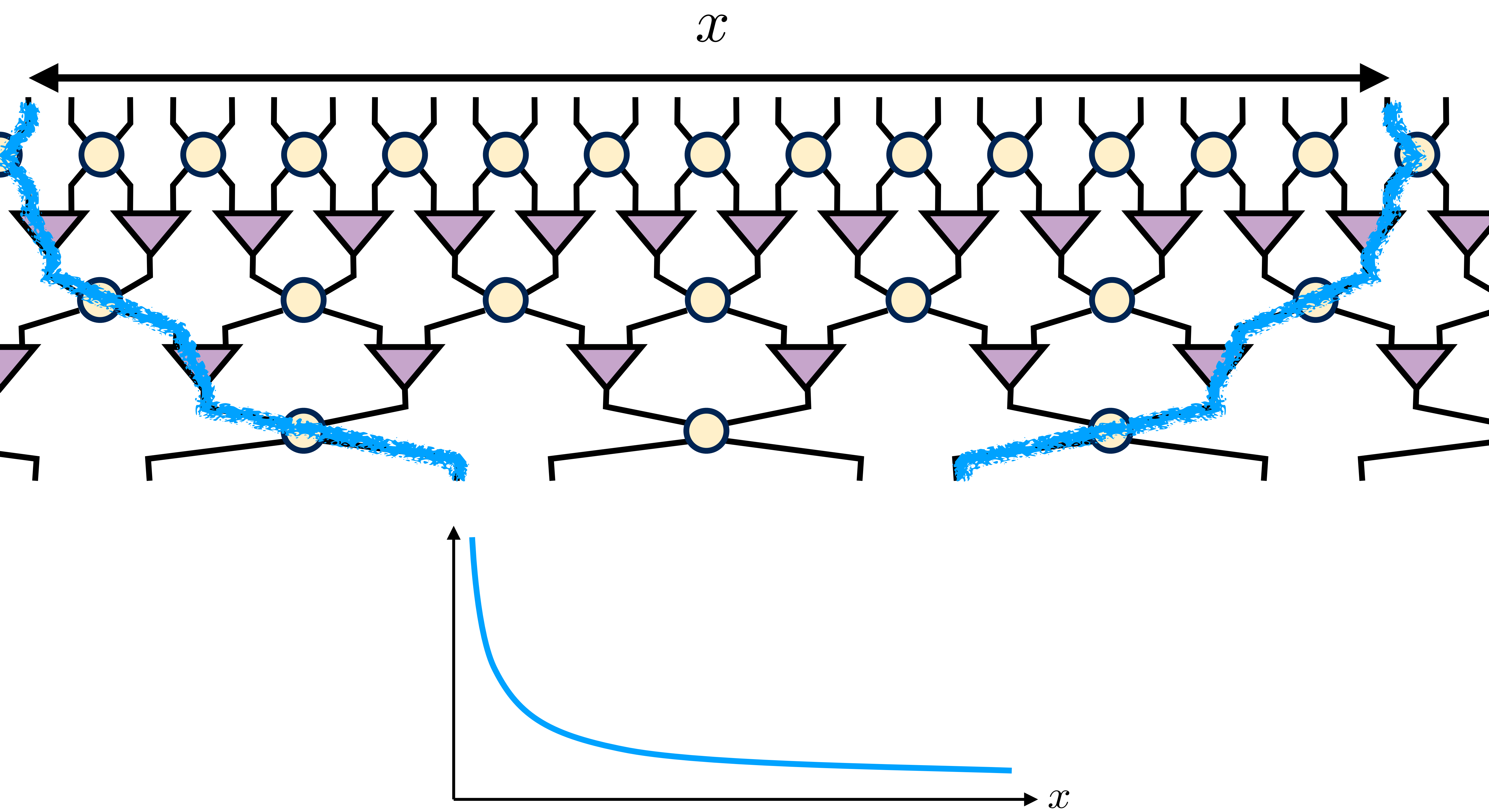
MERA layered architecture captures
power-law correlations





MERA Tensor Network

MERA layered architecture captures
power-law correlations



Tensor Network Machine Learning

Raw data vectors

$$\mathbf{x} = (x_1, x_2, x_3, \ldots, x_N)$$

Example: grayscale images, components of $\mathbf{x}$ are pixels

$$x_j \in [0, 1]$$

Propose following model

$$f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})$$

$$= \sum_{\mathbf{s}} W_{s_1 s_2 s_3 \cdots s_N} \; x_1^{s_1} x_2^{s_2} x_3^{s_3} \cdots x_N^{s_N} \qquad\qquad s_j = 0, 1$$

Weights are N-index tensor
Like N-site wavefunction

Cohen et al. arxiv:1509.05009
Novikov, Trofimov, Oseledets, arxiv:1605.03795
Stoudenmire, Schwab, arxiv:1605.05775

N=3 example:

$$f(\mathbf{x}) = W \cdot \Phi(\mathbf{x}) = \sum_{\mathbf{s}} W_{s_1 s_2 s_3}\ x_1^{s_1} x_2^{s_2} x_3^{s_3}$$

$$= W_{000} + W_{100}\, x_1 + W_{010}\, x_2 + W_{001}\, x_3$$

$$+ W_{110}\, x_1 x_2 + W_{101}\, x_1 x_3 + W_{011}\, x_2 x_3$$

$$+ W_{111}\, x_1 x_2 x_3$$

Contains linear classifier, and various poly. kernels

More generally, apply local "feature maps" $\phi^{s_j}(x_j)$

$$f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})$$

$$= \sum_{\mathbf{s}} W_{s_1 s_2 s_3 \cdots s_N} \phi^{s_1}(x_1) \phi^{s_2}(x_2) \phi^{s_3}(x_3) \cdots \phi^{s_N}(x_N)$$

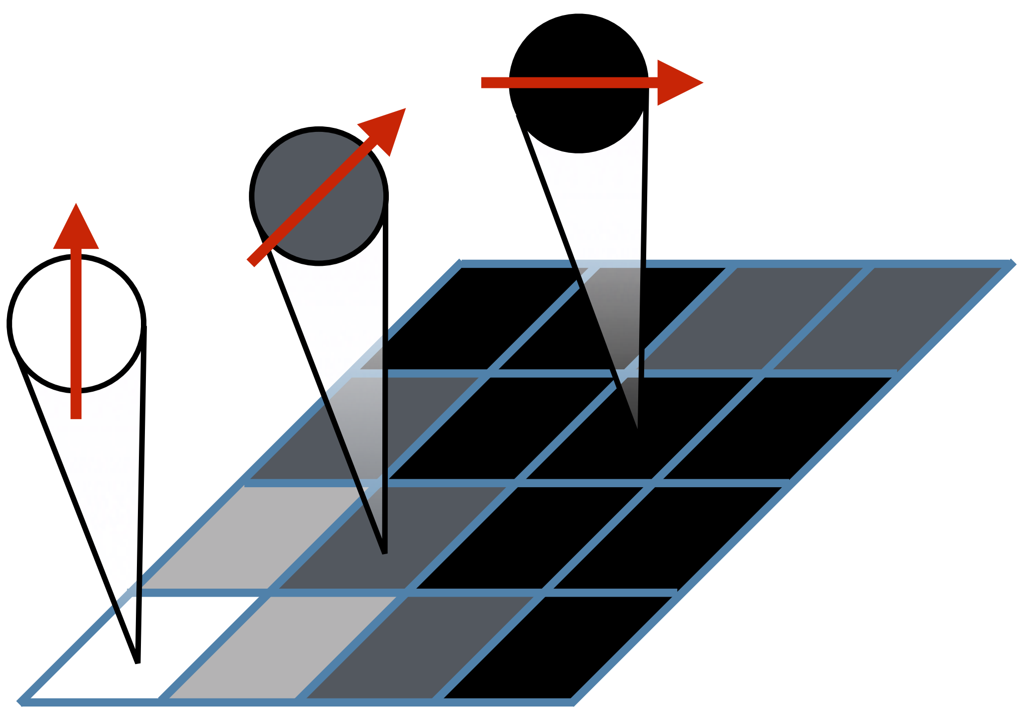
Highly expressive

Could put additional parameters into maps ϕ

For example, following local feature map

$$\phi(x_j) = \left[\cos\left(\frac{\pi}{2}x_j\right), \sin\left(\frac{\pi}{2}x_j\right)\right] \qquad x_j \in [0,1]$$

Picturesque idea of pixels as "spins"



Stoudenmire, Schwab, arxiv:1605.05775

$\mathbf{x} =$ input

$\phi =$ local feature map

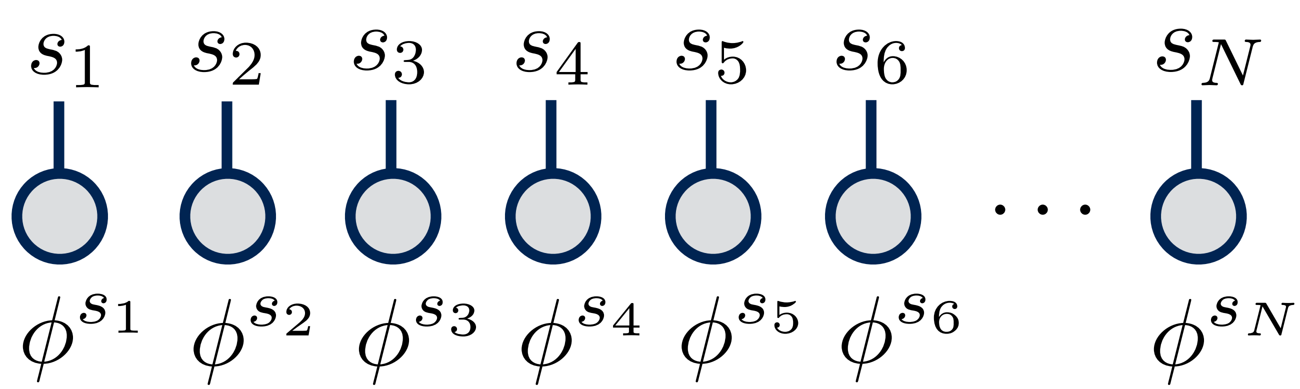
Total feature map $\Phi(\mathbf{x})$

Tensor diagram notation

$$\mathbf{x} = [x_1, \quad x_2, \quad x_3, \quad \ldots \quad , \quad x_N]$$

raw inputs

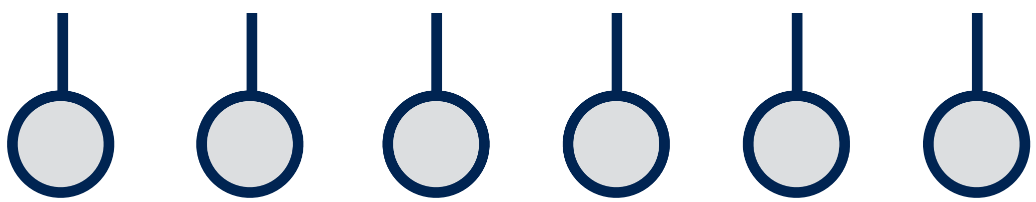
$\Phi(\mathbf{x}) =$



feature vector

Construct decision function

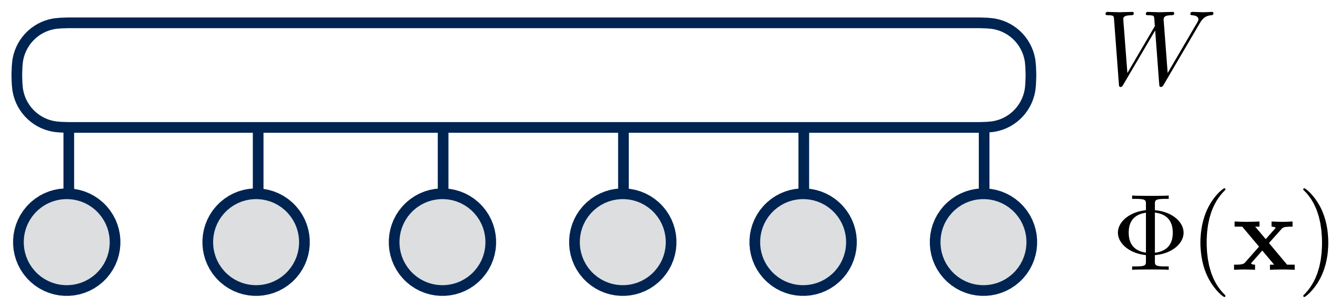
$$\boxed{f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})}$$



$\Phi(\mathbf{x})$

Construct decision function

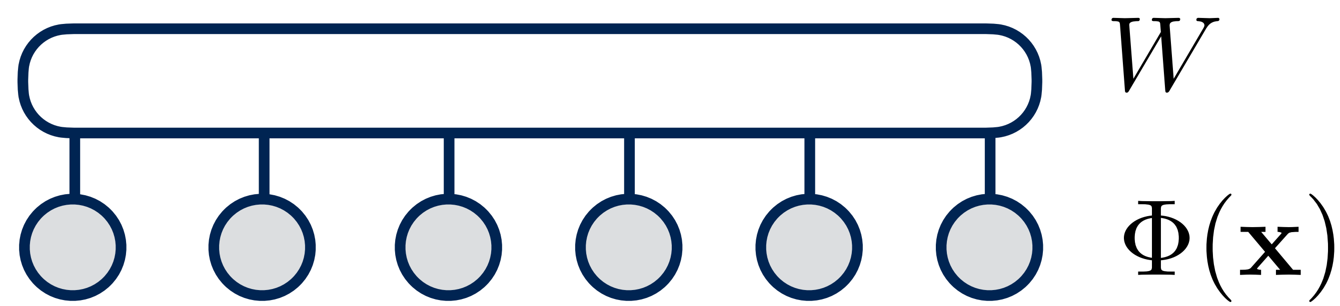
$$\boxed{f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})}$$



Construct decision function

$$\boxed{f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})}$$

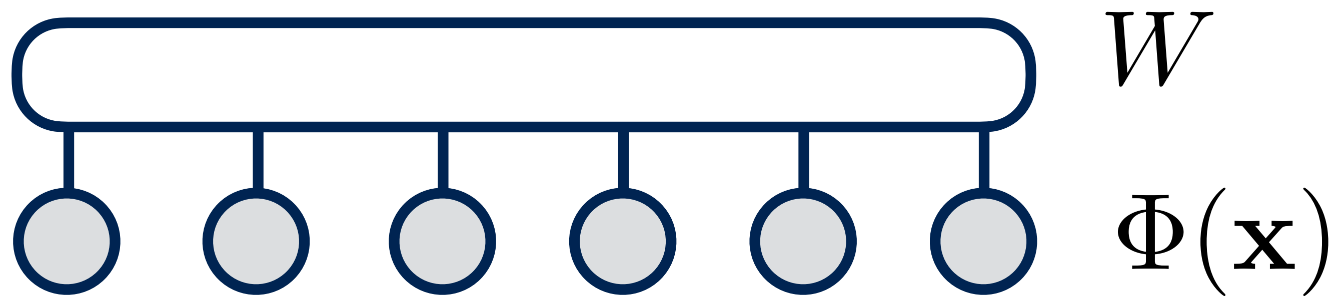
$$f(\mathbf{x}) =$$



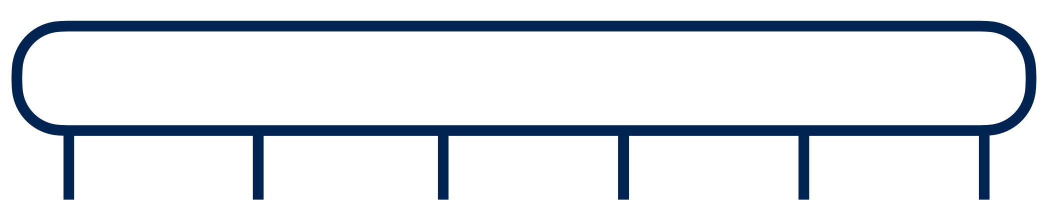
Construct decision function

$$\boxed{f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})}$$

$f(\mathbf{x}) =$

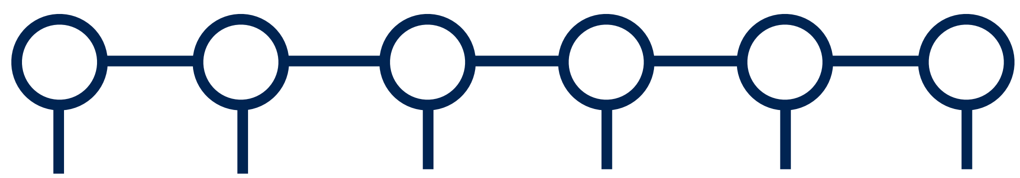


$W =$



Main approximation

$W =$  *order-N tensor*

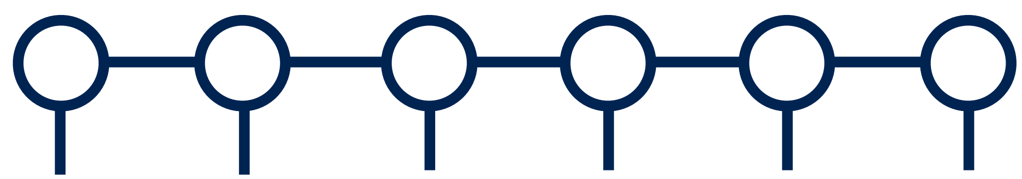
$\approx$  *matrix product state (MPS)*

Main approximation

$W =$

order-N tensor

$\approx$



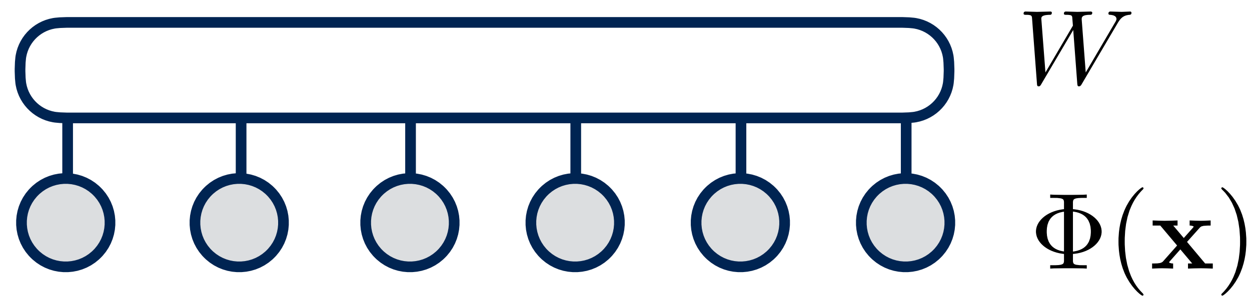
matrix product state (MPS)

($\approx$

PEPS)

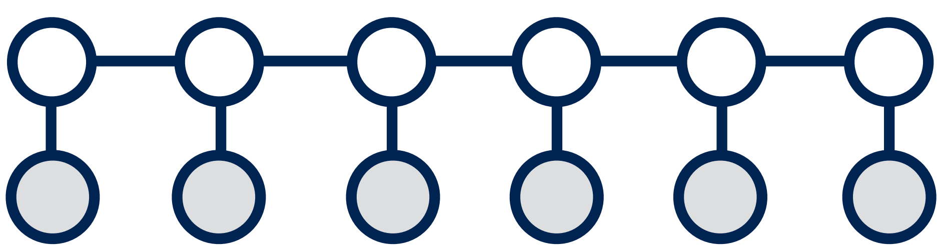
Tensor diagrams of the approach

$$f(\mathbf{x}) = W \cdot \Phi(\mathbf{x}) =$$



$$\approx (M_{s_1} M_{s_2} \cdots M_{s_N}) \Phi^{s_1 s_2 \cdots s_N}(\mathbf{x})$$

$$\approx$$



Linear scaling

Can use algorithm similar to DMRG to optimize

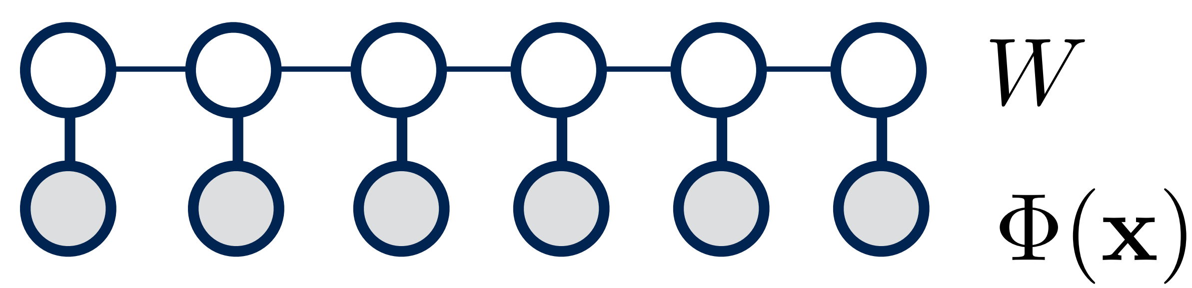
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$



Linear scaling

Can use algorithm similar to DMRG to optimize

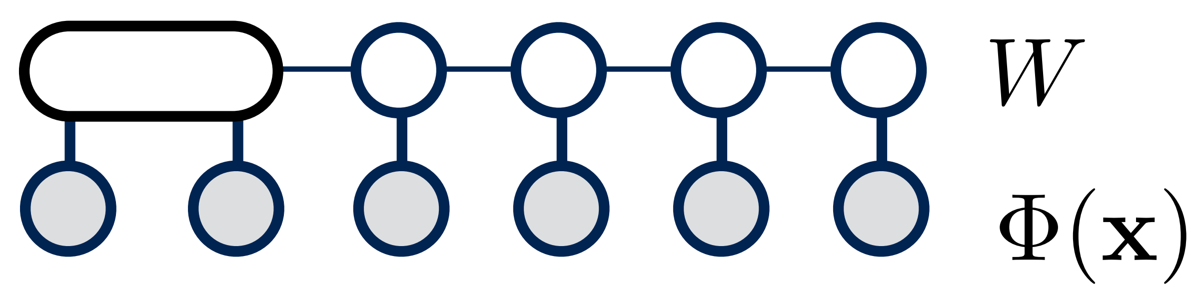
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$



Linear scaling

Can use algorithm similar to DMRG to optimize

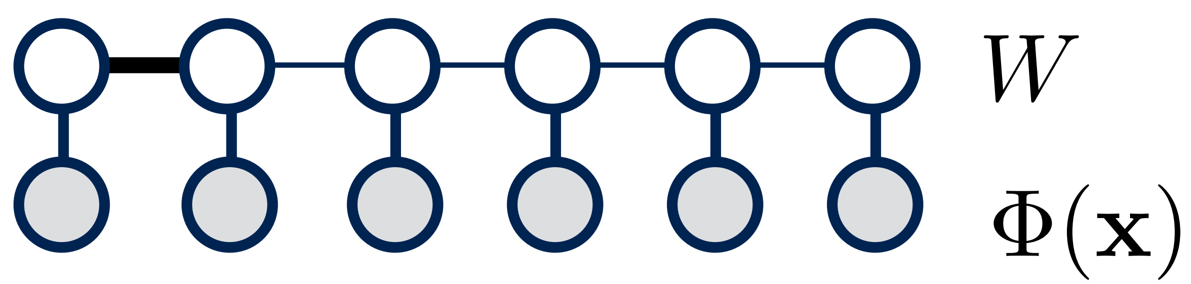
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$



Linear scaling

Can use algorithm similar to DMRG to optimize

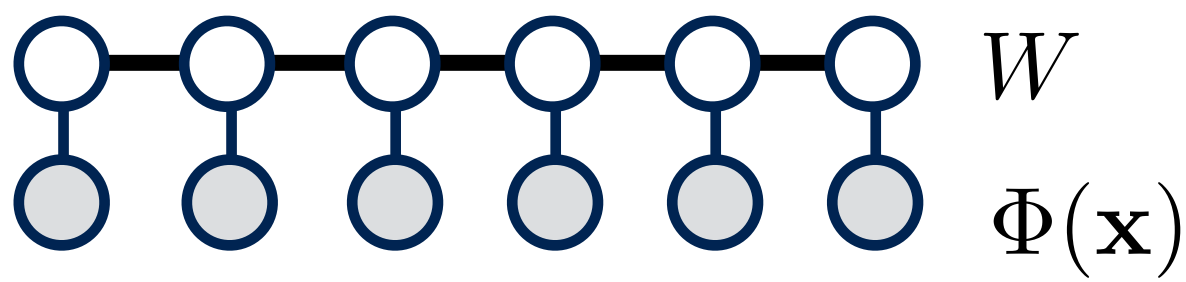
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

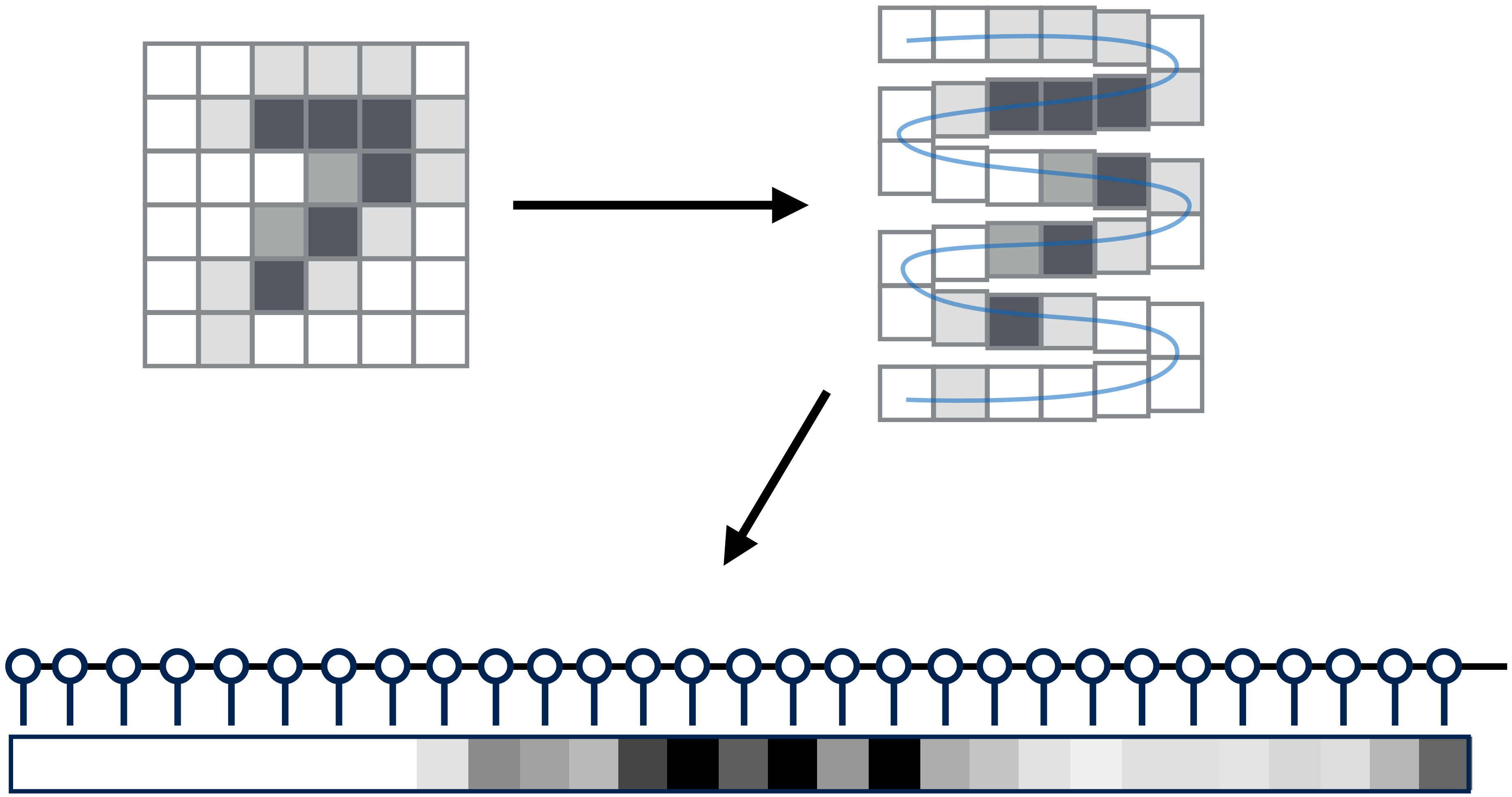
$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$



Experiment: handwriting classification (MNIST)



Train to 99.95% accuracy on 60,000 training images

Obtain 99.03% accuracy on 10,000 test images
(only 97 incorrect)

Stoudenmire, Schwab, arxiv:1605.05775

Papers using tensor network machine learning

Expressivity & priors of TN based models

- *Levine et al., "**Deep Learning and Quantum Entanglement: Fundamental Connections with Implications to Network Design**" arxiv:1704.01552*
- *Cohen, Shashua, "**Inductive Bias of Deep Convolutional Networks through Pooling Geometry**" arxiv:1605.06743*
- *Cohen et al., "**On the Expressive Power of Deep Learning: A Tensor Analysis**" arxiv:1509.05009*

Generative Models

- *Han et al., "**Unsupervised Generative Modeling Using Matrix Product States**" arxiv:1709.01662*
- *Sharir et al., "**Tractable Generative Convolutional Arithmetic Circuits**" arxiv:1610.04167*

Supervised Learning

- *Novikov et al., "**Expressive power of recurrent neural networks**", arxiv:1711.00811*
- *Liu et al., "**Machine Learning by Two-Dimensional Hierarchical Tensor Networks: A Quantum Information Theoretic Perspective on Deep Architectures**", arxiv:1710.04833*
- *Stoudenmire, Schwab, "**Supervised Learning with Quantum-Inspired Tensor Networks**", arxiv:1605.05775*
- *Novikov et al., "**Exponential Machines**", arxiv: 1605.03795*

Learning Relevant Features of Data

For a model $f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})$

Given training data $\{\mathbf{x}_j\}$

Can show optimal W is of the form

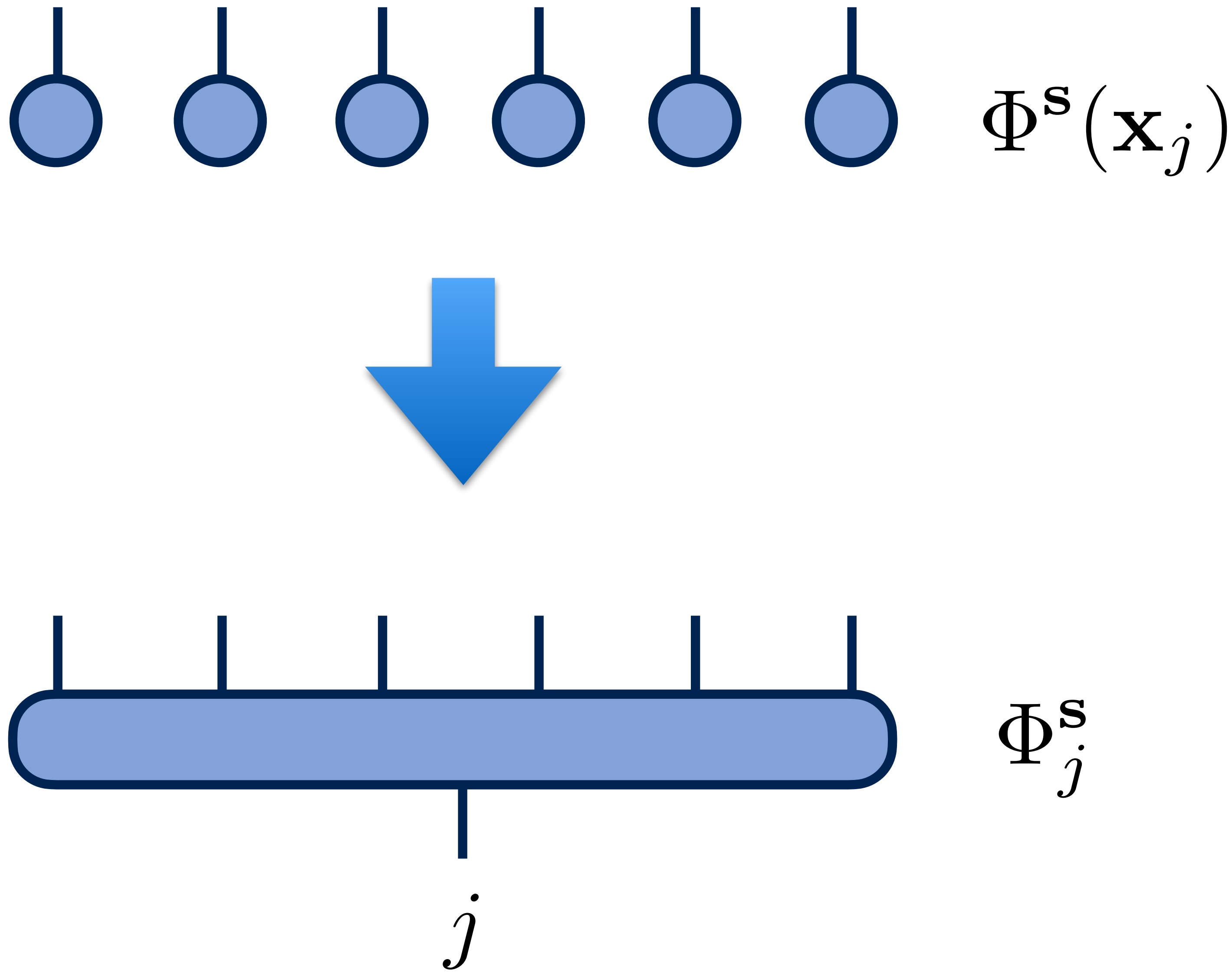
$$W = \sum_j \alpha_j \, \Phi(\mathbf{x}_j)$$

Holds for wide variety of cost functions / tasks

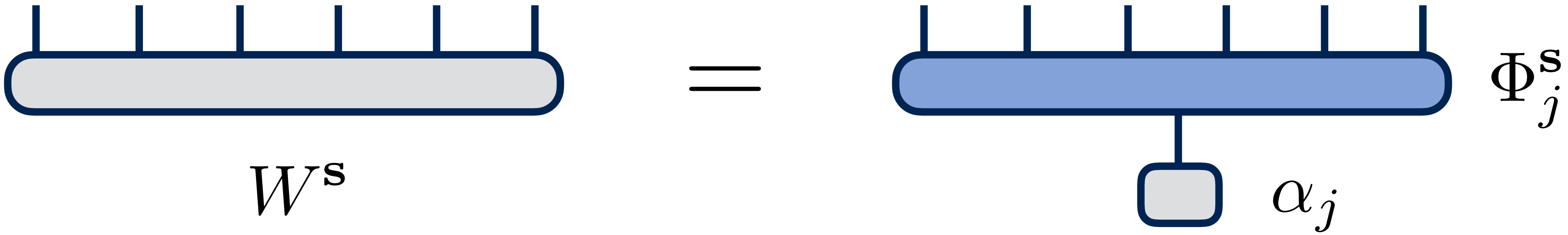
"representer theorem"

Schölkopf, Smola, Müller, Neural Comp. 10, 1299 (1998)

View $\Phi^{\mathbf{s}}(\mathbf{x}_j) = \Phi^{\mathbf{s}}_j$ as a tensor

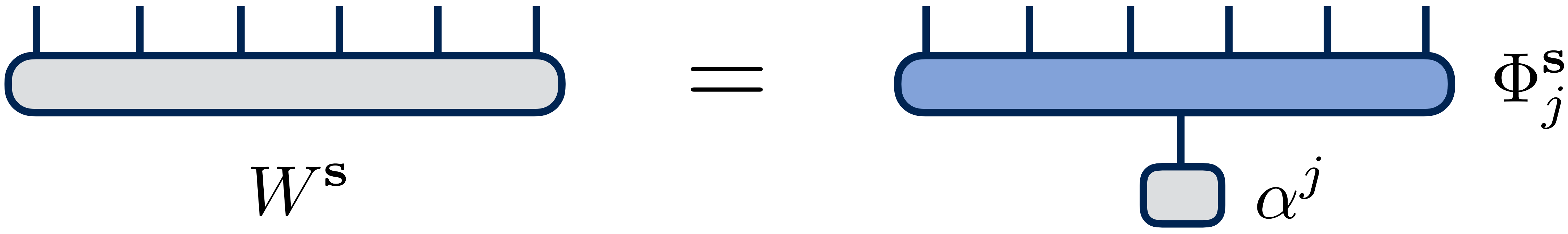


Representer theorem says

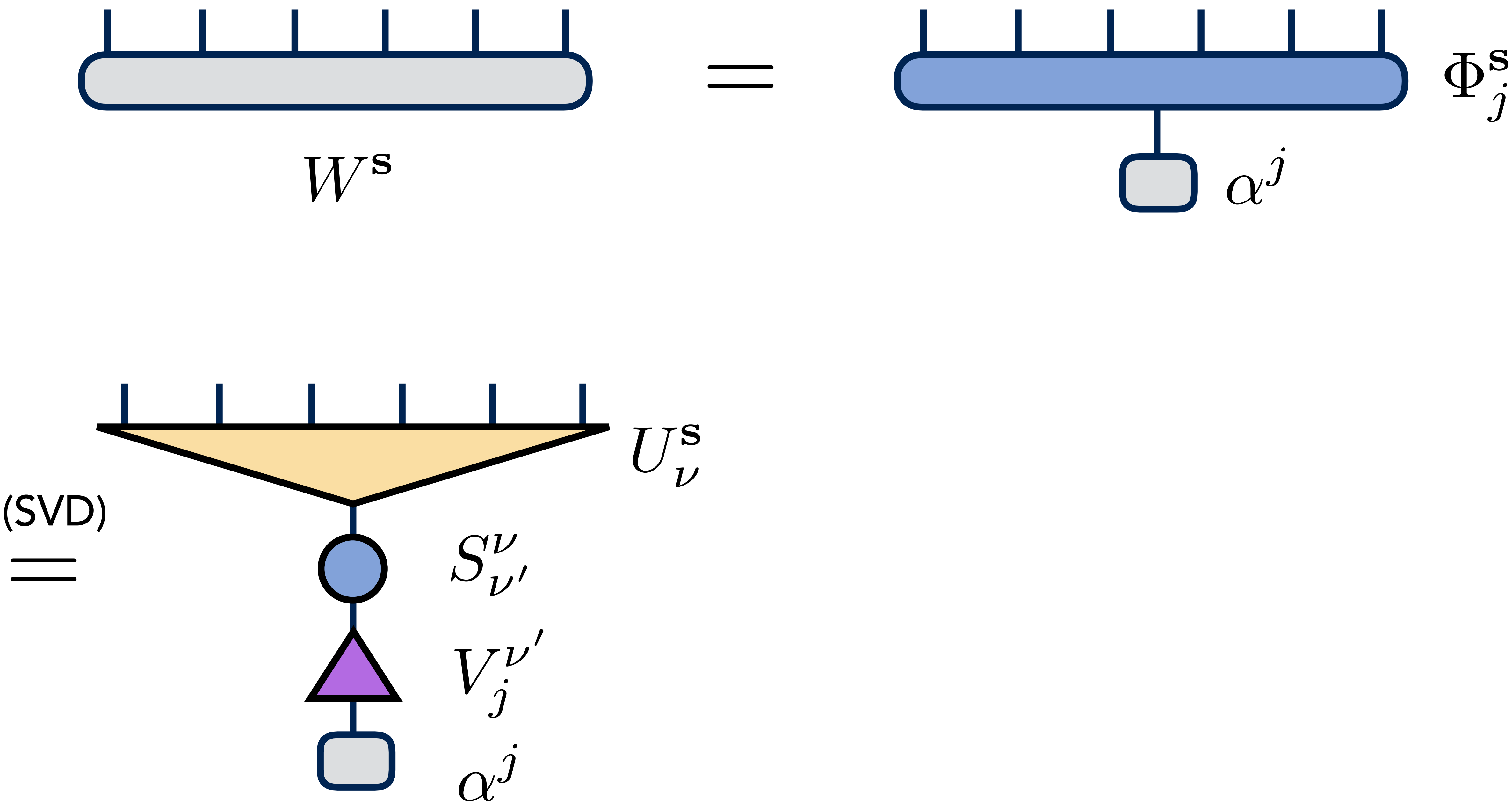


Really just says weights in the span of $\{\Phi^{\mathbf{s}}_j\}$

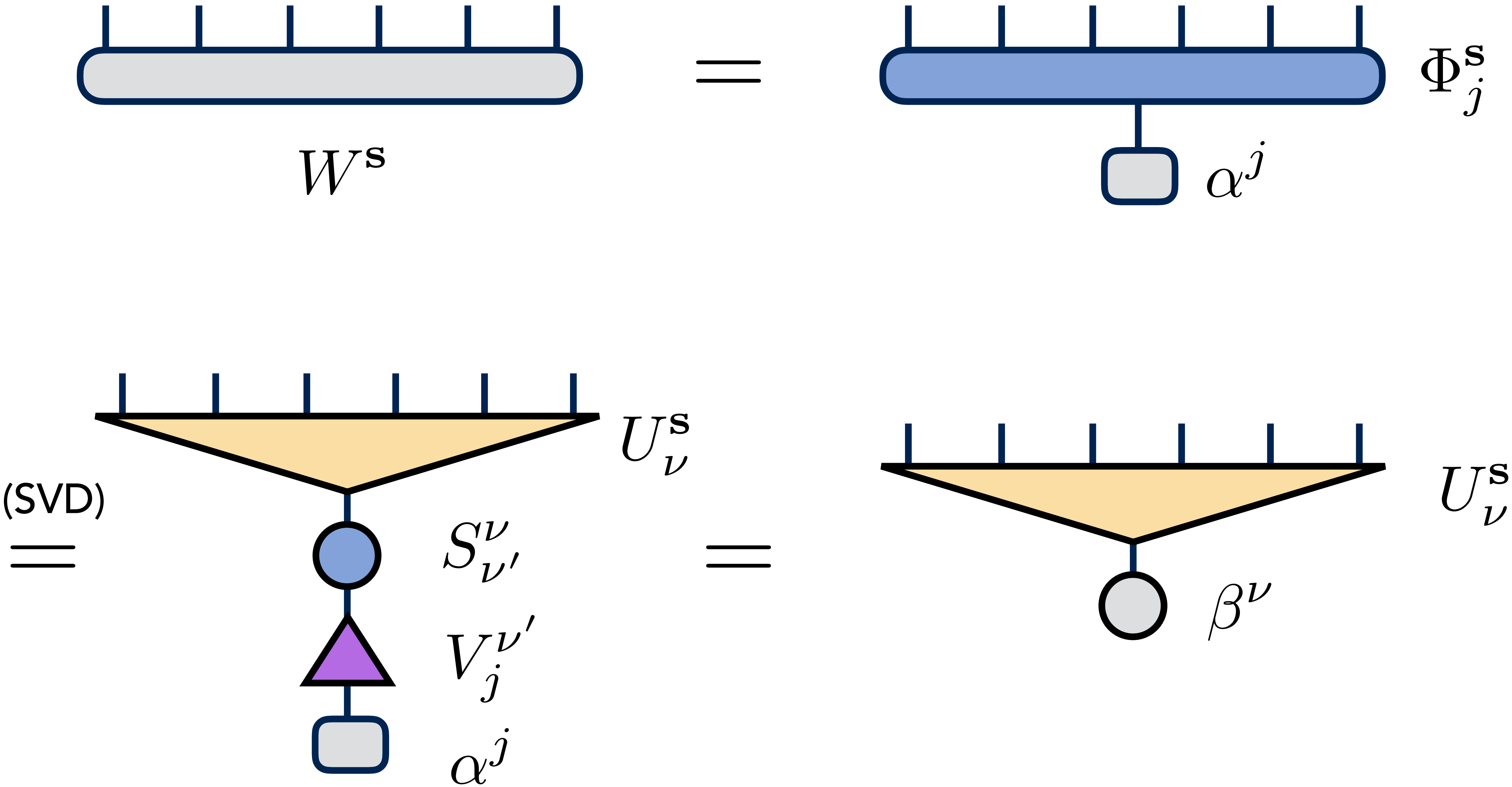
Can choose any basis for span of $\{\Phi_j^{\mathbf{s}}\}$



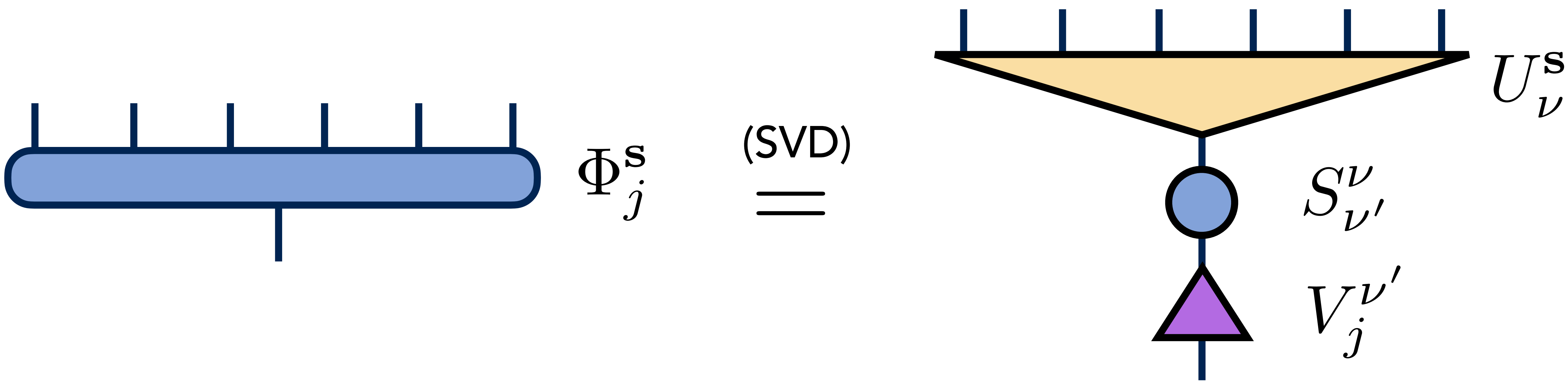
Can choose any basis for span of $\{\Phi_j^{\mathbf{s}}\}$



Can choose any basis for span of $\{\Phi_j^{\mathbf{s}}\}$



Why switch to $U_\nu^{\mathbf{s}}$ basis?



Orthonormal basis

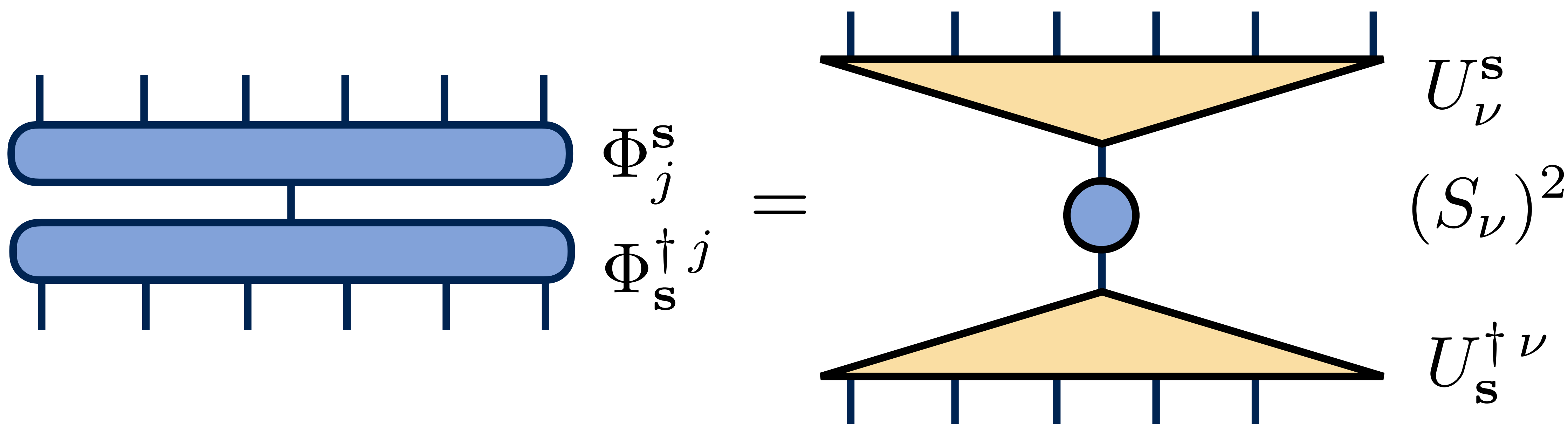
Can discard basis vectors corresponding to small s. vals.

Can compute $U_\nu^{\mathbf{s}}$ fully or partially using *tensor networks*

Computing $U_\nu^{\mathbf{s}}$ efficiently

Define *feature space covariance matrix* (similar to density matrix)

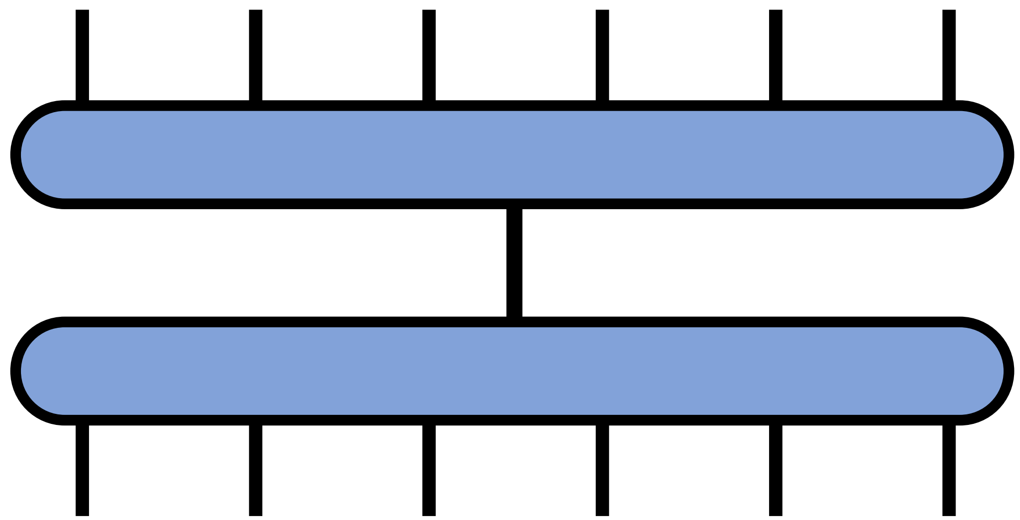
$$\rho = \frac{1}{N_T}$$



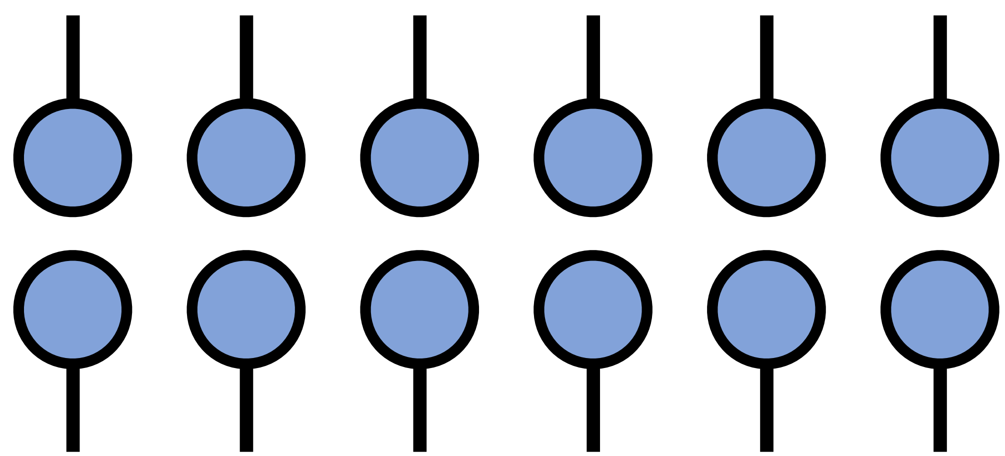
Strategy: compute $U_\nu^{\mathbf{s}}$ iteratively as a layered (tree) tensor network

For efficiency, exploit product structure of Φ

$$\rho = \Phi\Phi^\dagger = \frac{1}{N_T}$$



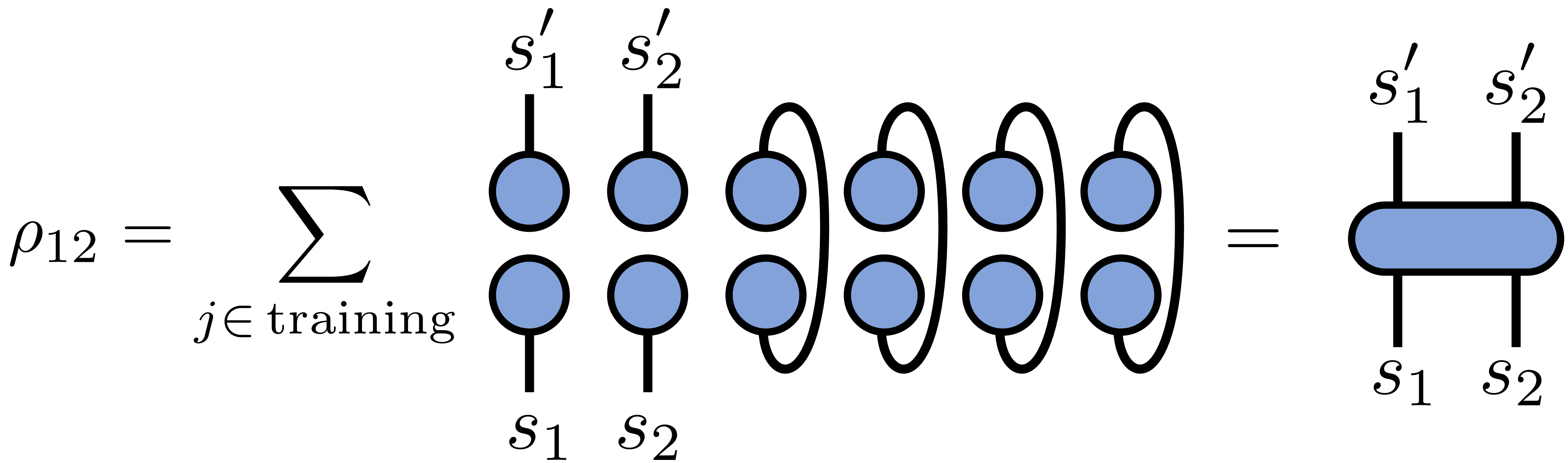
$$= \frac{1}{N_T}\sum_{j=1}^{N_T}$$

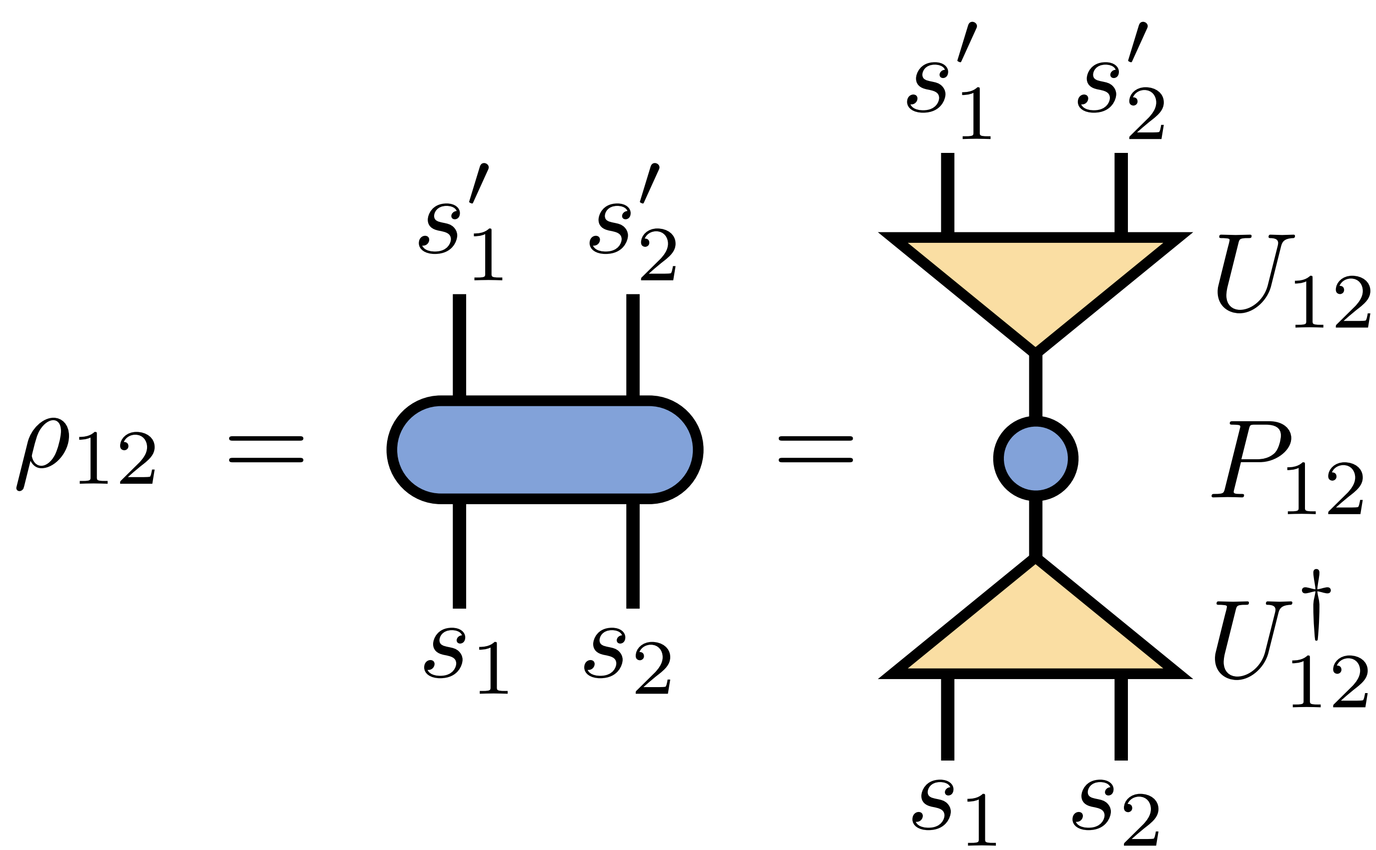


$\Phi(\mathbf{x}_j)$

$\Phi^\dagger(\mathbf{x}_j)$

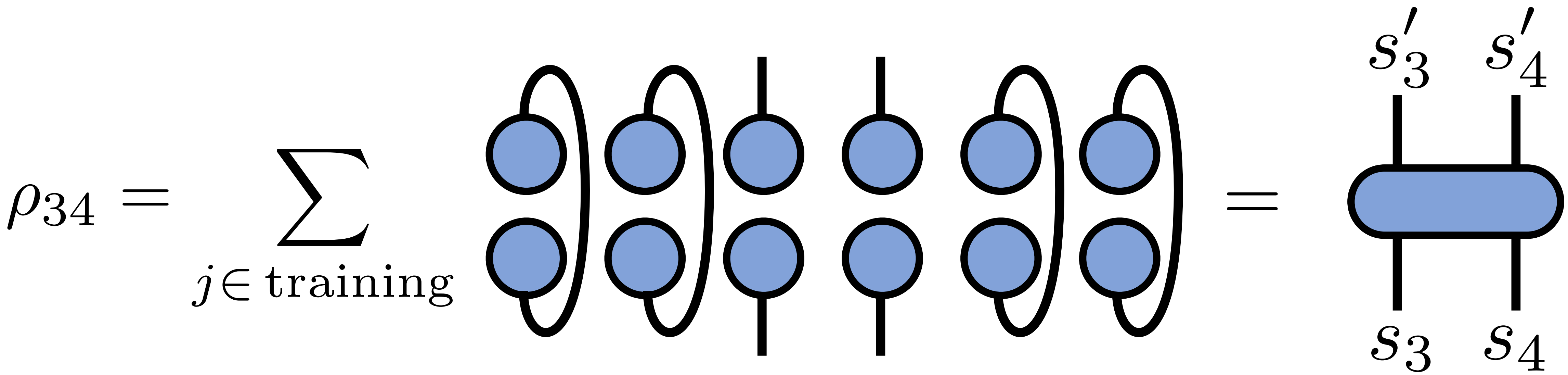
Compute tree tensors from reduced matrices

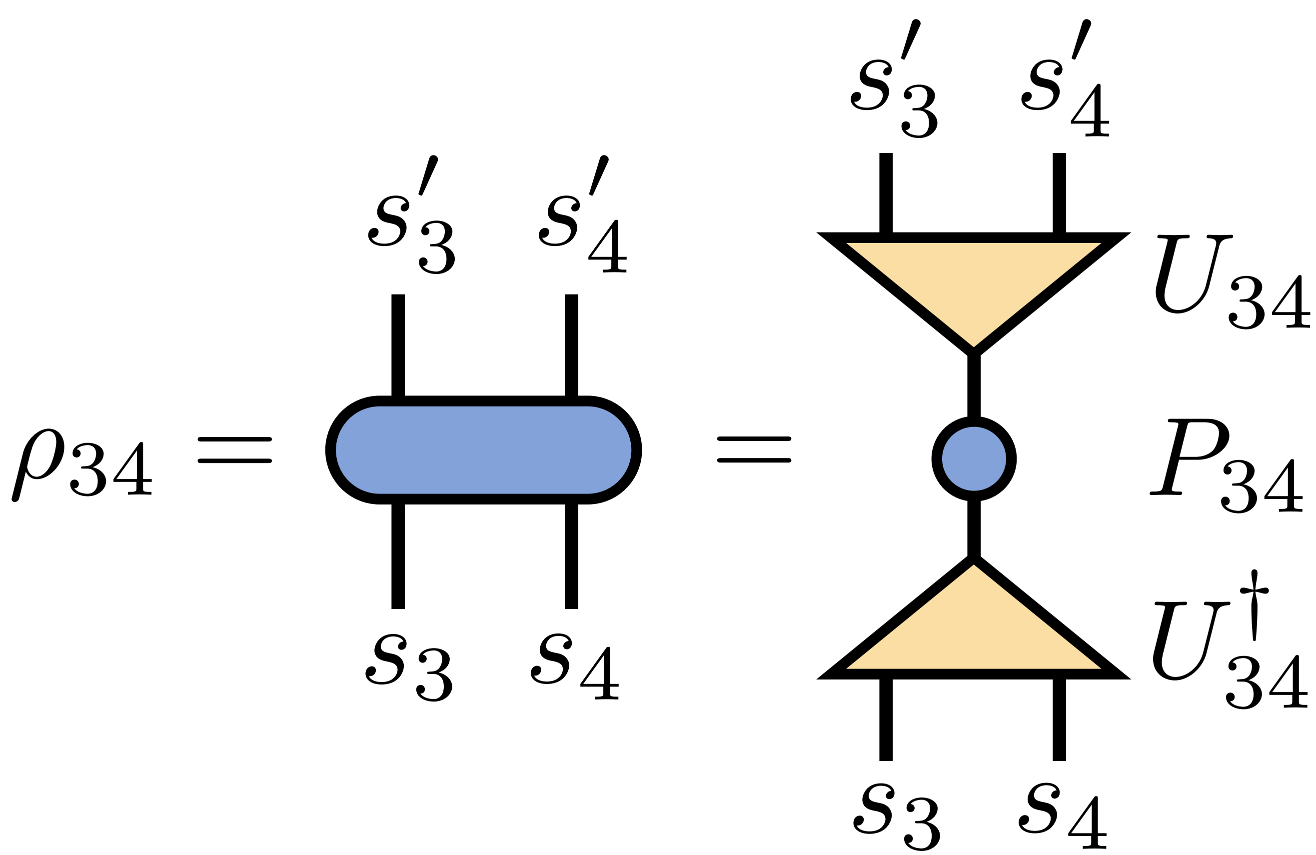




Truncate small eigenvalues

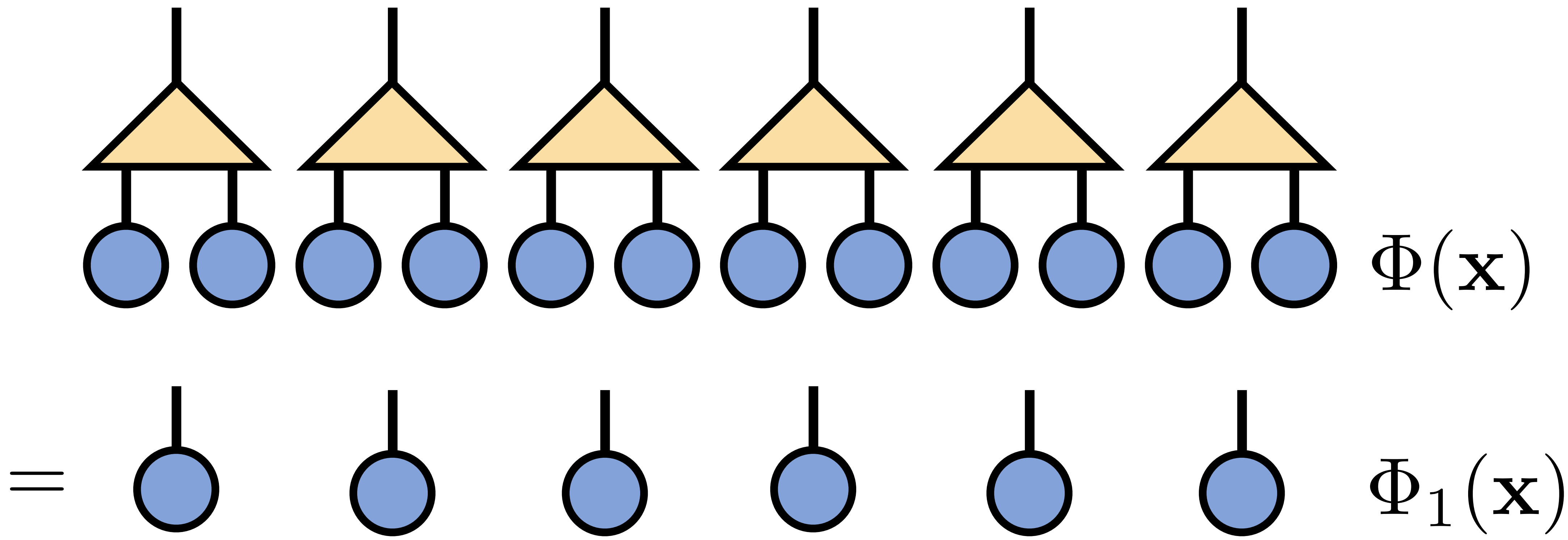
Compute tree tensors from reduced matrices



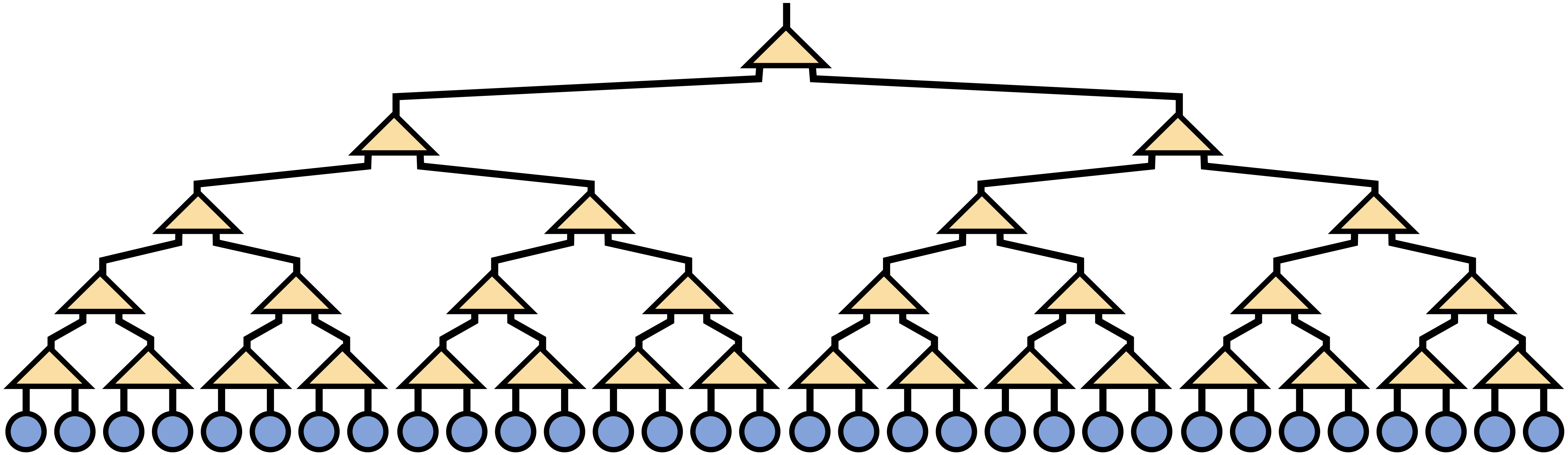


Truncate small eigenvalues

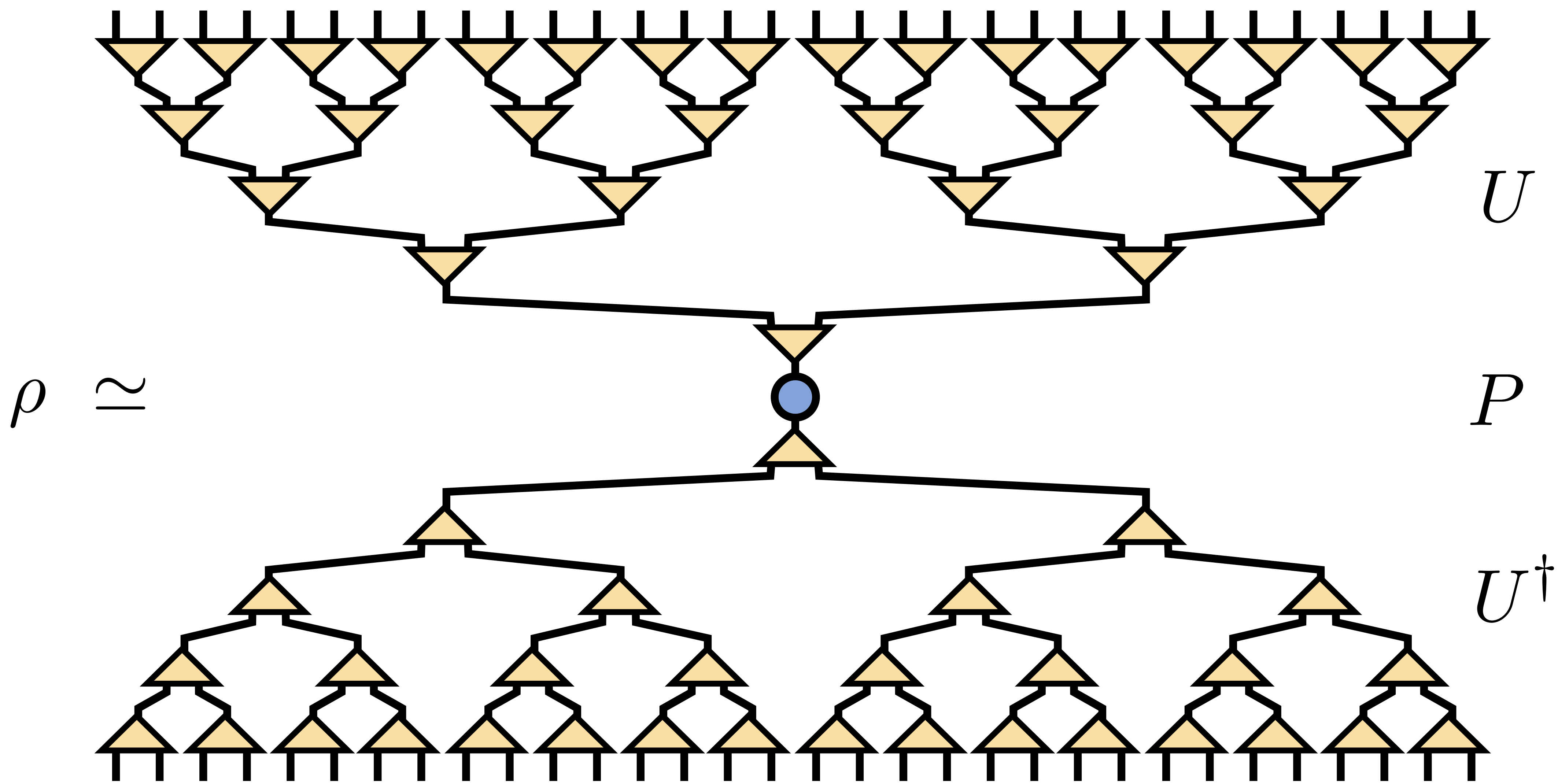
Having computed a tree layer, rescale data



Can view as *unsupervised learning* of representation of training data



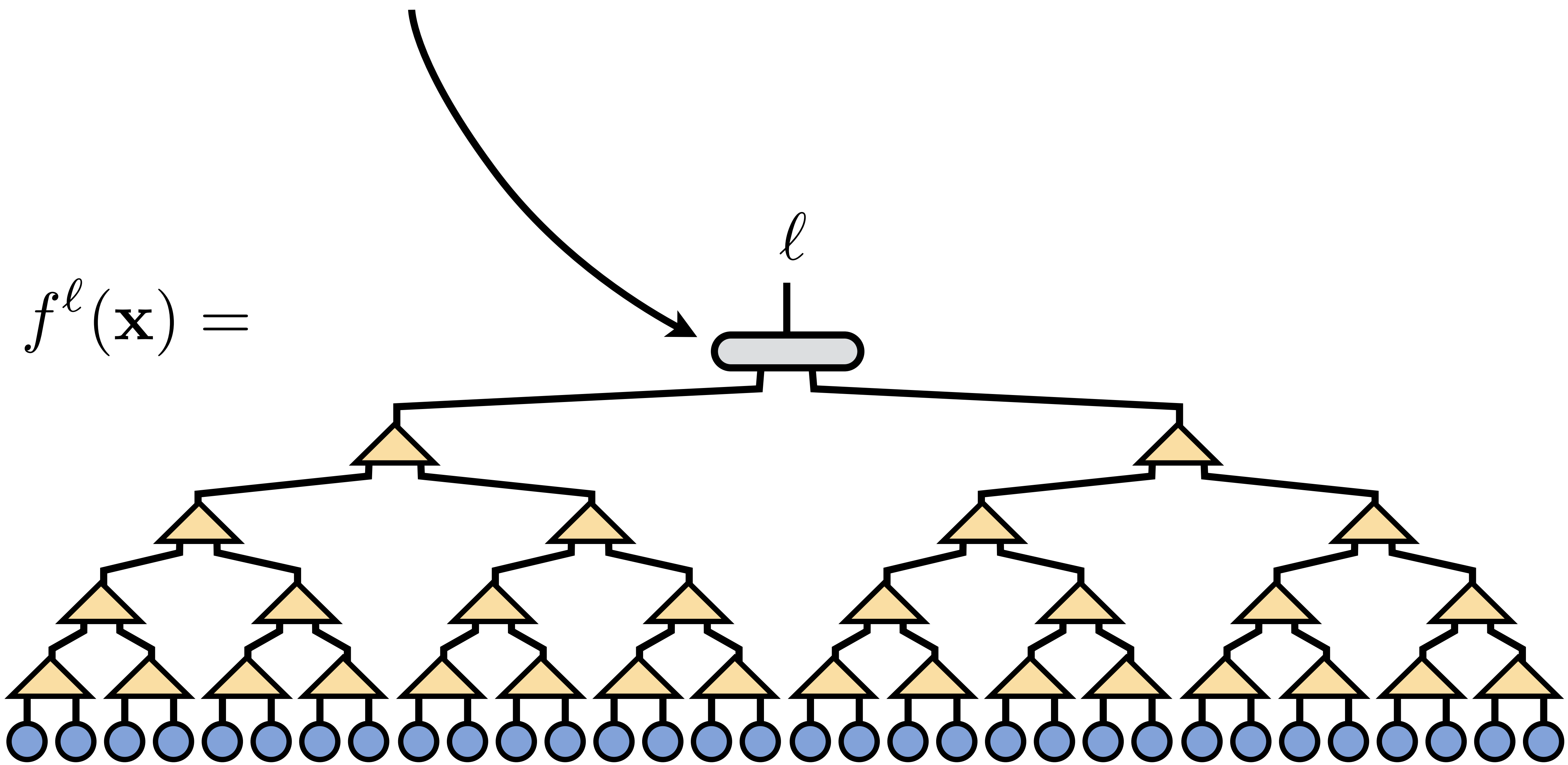
Computing all layers approximately diagonalizes covariance matrix



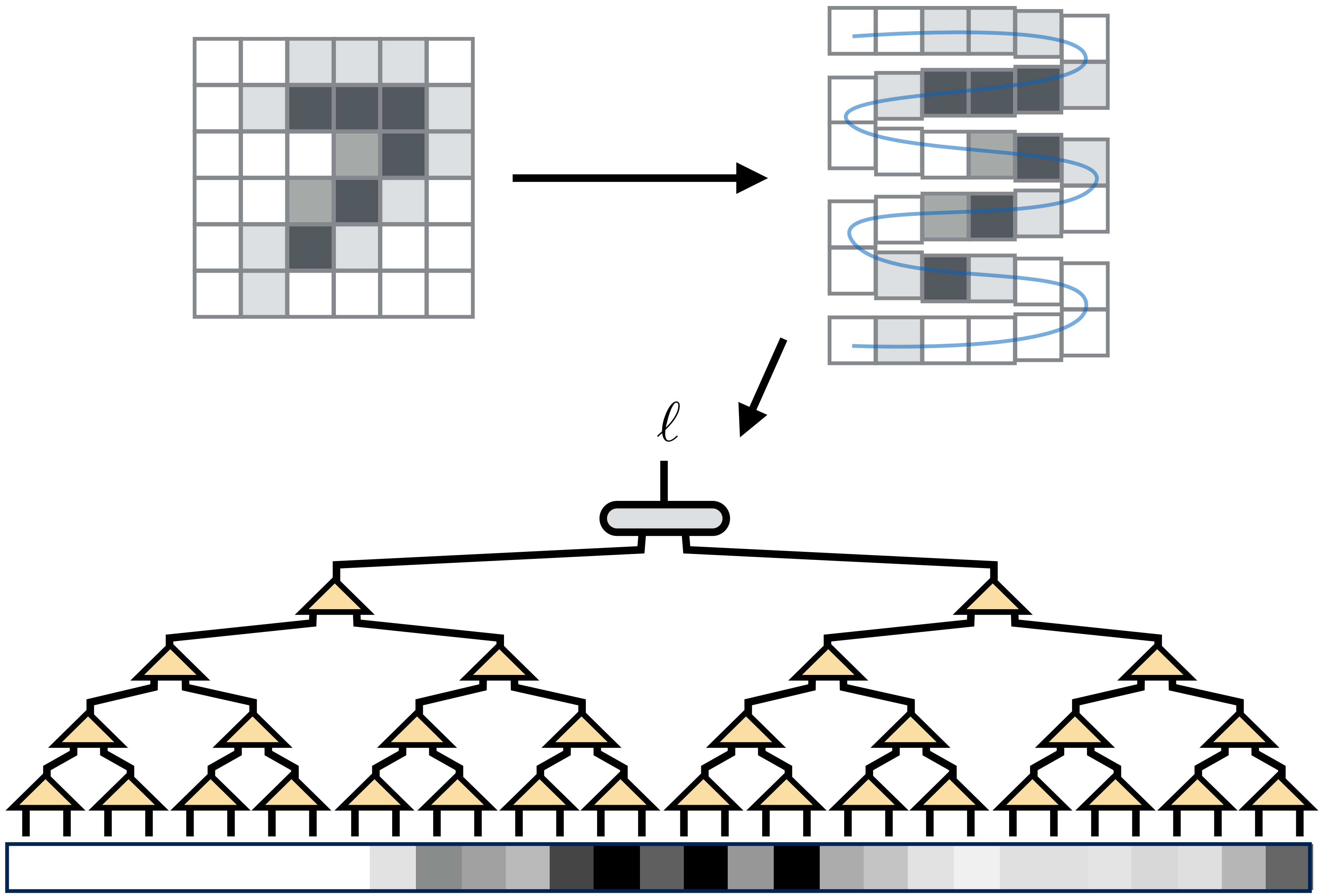
Use as starting point for supervised learning

Only train top tensor for supervised task

$f^{\ell}(\mathbf{x}) =$



Experiment: handwriting classification (MNIST)



Cutoff 6x10^{-4} gave top indices sizes 328 and 444

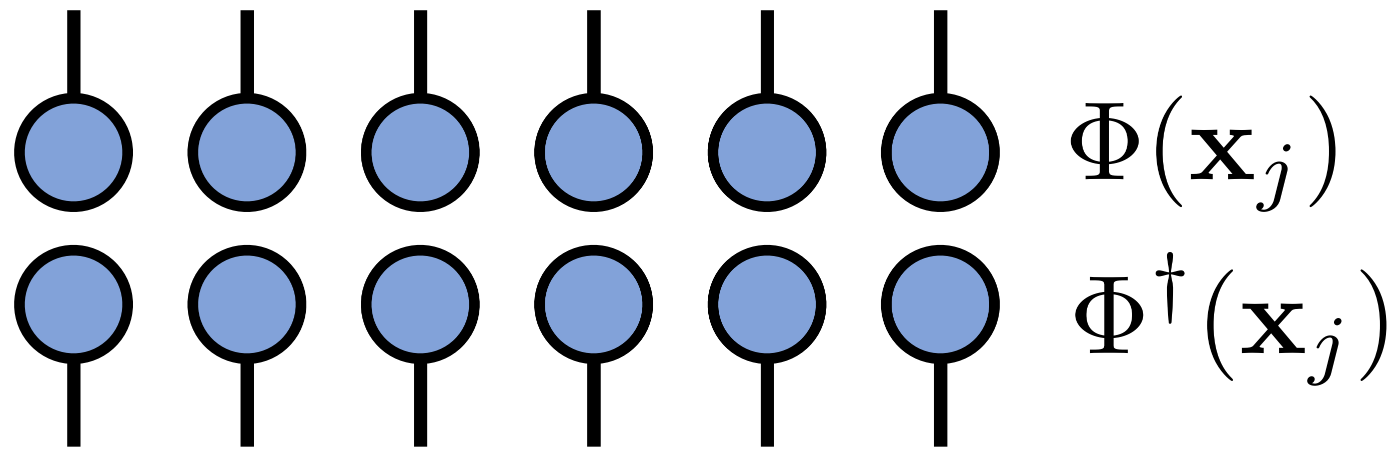
Training acc: 99.68% **Test acc:** 98.08%

Refinements and Extensions

No reason we must base tree around ρ

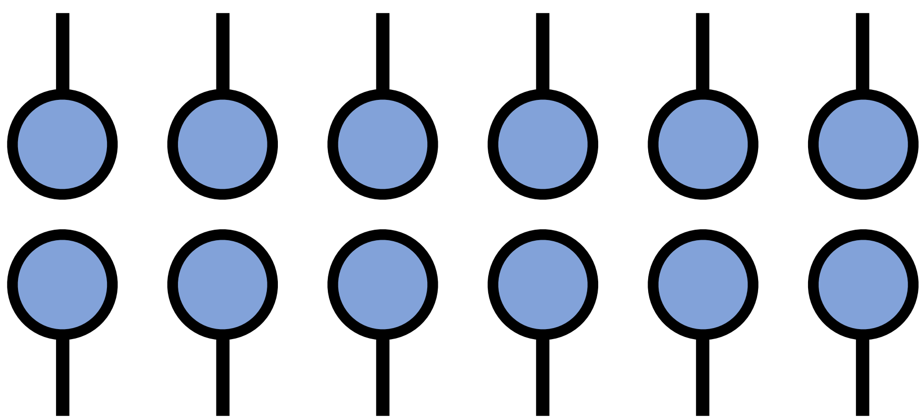
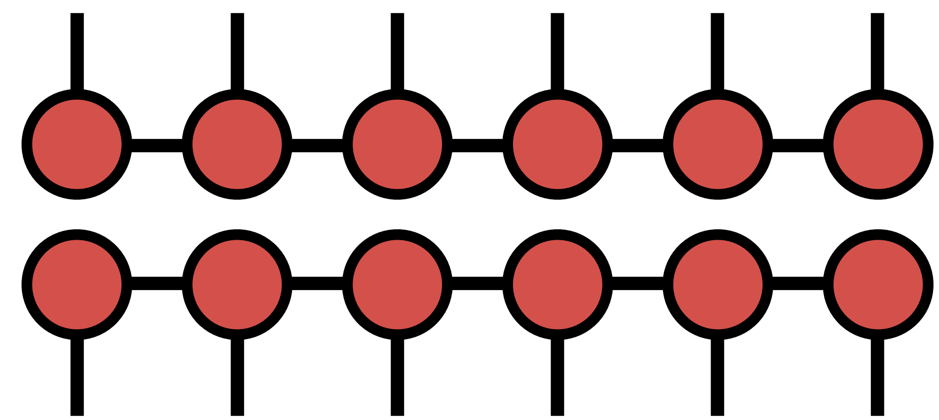
Could reweight based on importance of samples

$$\tilde{\rho} = \frac{1}{N_T} \sum_{j=1}^{N_T} w_j$$



Another idea is to mix in a "lower level" model trained on a given task (e.g. supervised learning)

$$\rho^{\mu} =$$

$$(1-\mu)\sum_j$$  $$+ \mu$$ 

If $\mu = 1$, tree provides basis for provided weights

If $0 < \mu < 1$, tree is "enriched" by data set

Experiment: mixed correlation matrix for MNIST

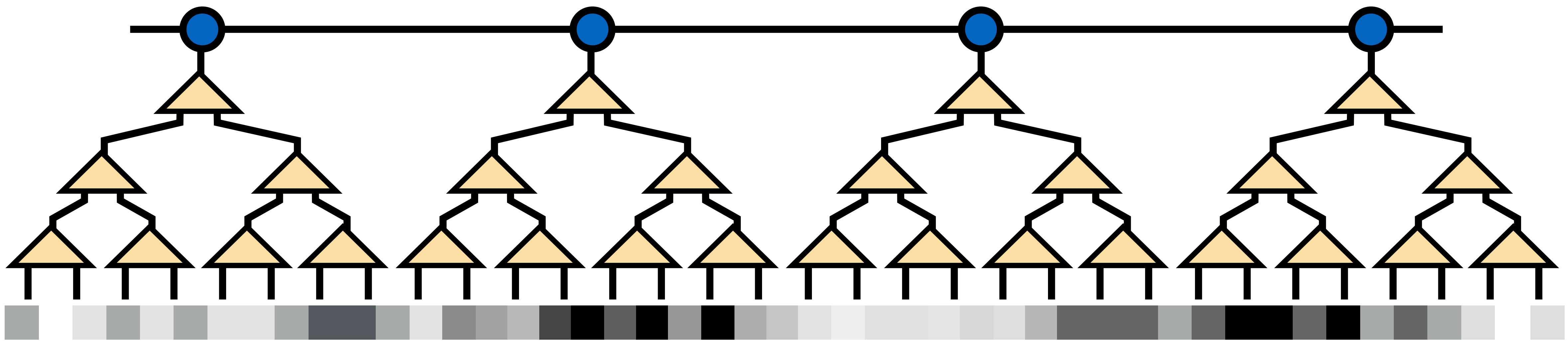
Using $\rho^{\mu} = (1-\mu)\rho + \mu \sum_{\ell} |W^{\ell}\rangle\langle W^{\ell}|$

with trial weights trained from a linear classifier
and $\mu = 0.5$

Train acc: 99.798% **Test acc:** 98.110%
Top indices of size 279 and 393.

Comparable performance to unmixed case with
top index sizes 328 and 444

Also no reason to build entire tree



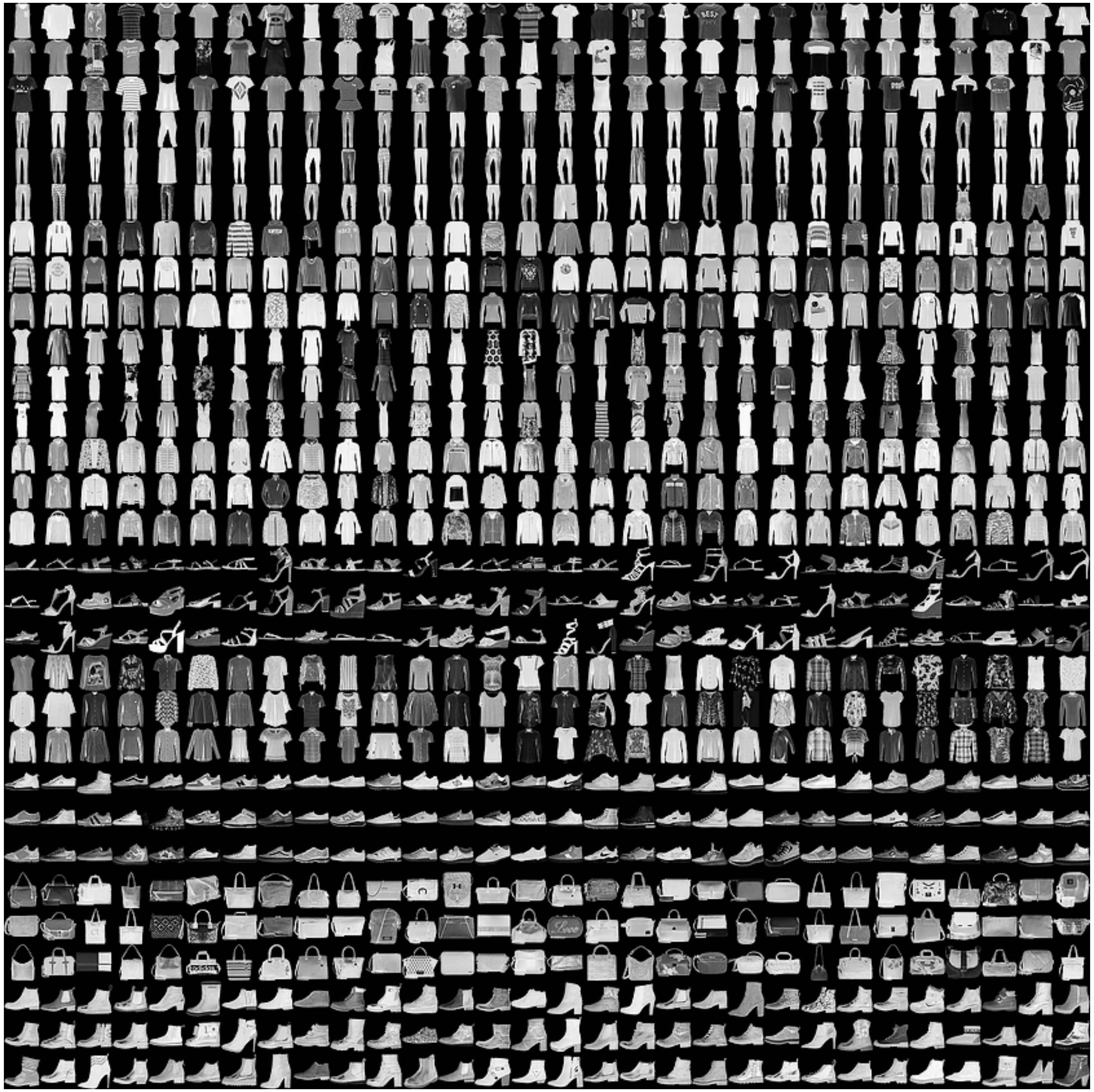
Approximate top tensor by MPS

Experiment: "fashion MNIST" dataset

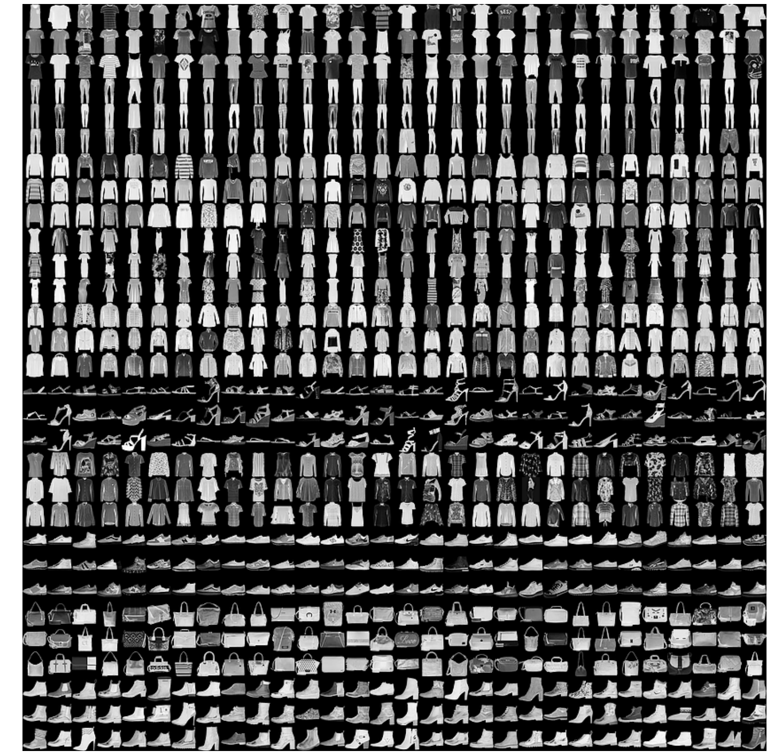
28x28 grayscale

60,000 training images

10,000 testing images



Experiment: "fashion MNIST" dataset



- Used 4 tree tensor layers
- Dimension of top "site" indices ranged from 11 to 30
- Top MPS bond dimension of 300 and 30 sweeps

Train acc: 95.38% **Test acc:** **88.97%**

Comparable to XGBoost (**89.8%**), AlexNet (**89.9%**), Keras Conv Net (**87.6%**)

Best (w/o preprocessing) is GoogLeNet at **93.7%**

Much Room for Improvement

- Use MERA instead of tree layers
- Optimize all layers, not just top, for specific task
- Iterate mixed approach: feed trained network into new covariance/density matrix
- Stochastic gradient based training

Recap & Future Directions

- Trained layered tensor network on real-world data in unsupervised fashion

- Specializing top layer gives very good results on challenging supervised image recognition tasks

- Linear tensor network approach gives enormous flexibility. Progress toward interpretability.