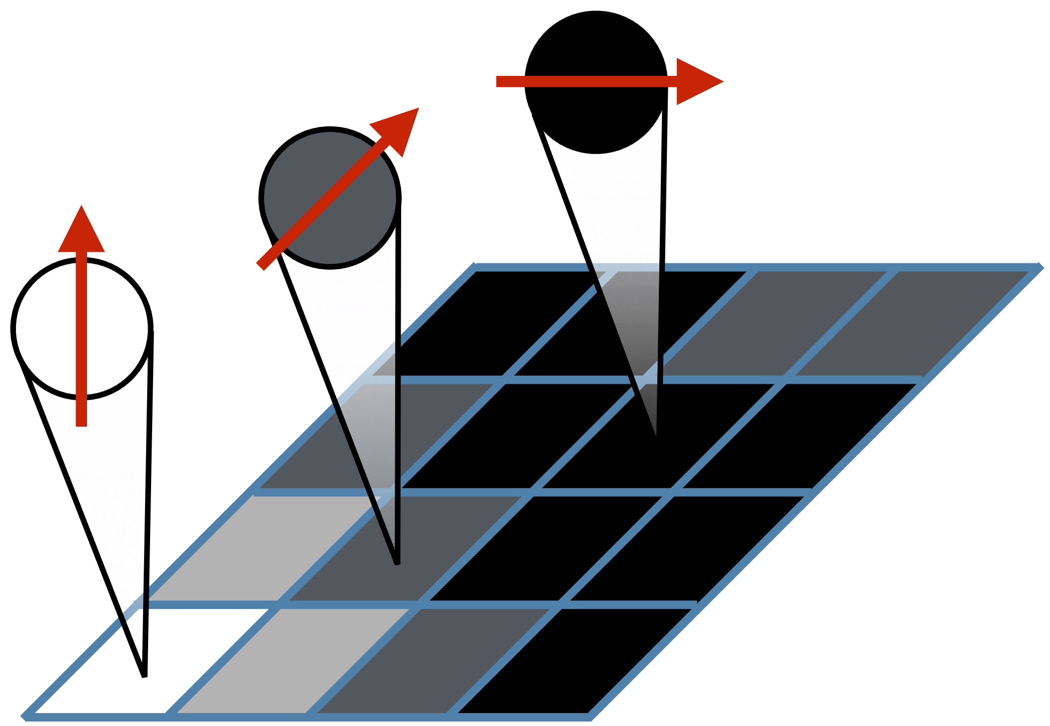
Machine Learning with Tensor Networks



E.M. Stoudenmire and David J. Schwab

Advances in Neural Information Processing **29**

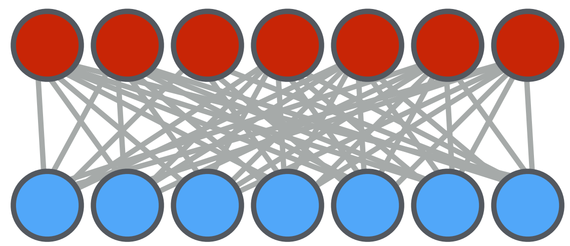
arxiv:1605.05775

Beijing – Jun 2017

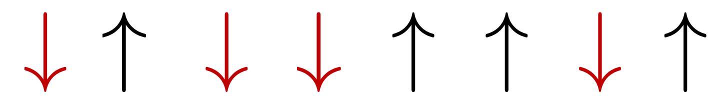
UCIrvine

SIMONS FOUNDATION

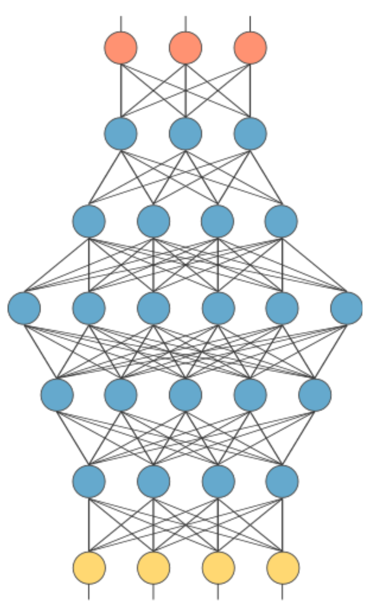
Machine learning has physics in its DNA



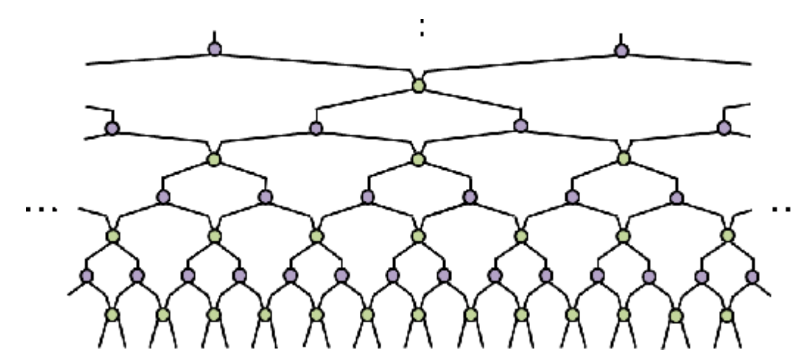
Boltzmann Machines



Disordered Ising Model



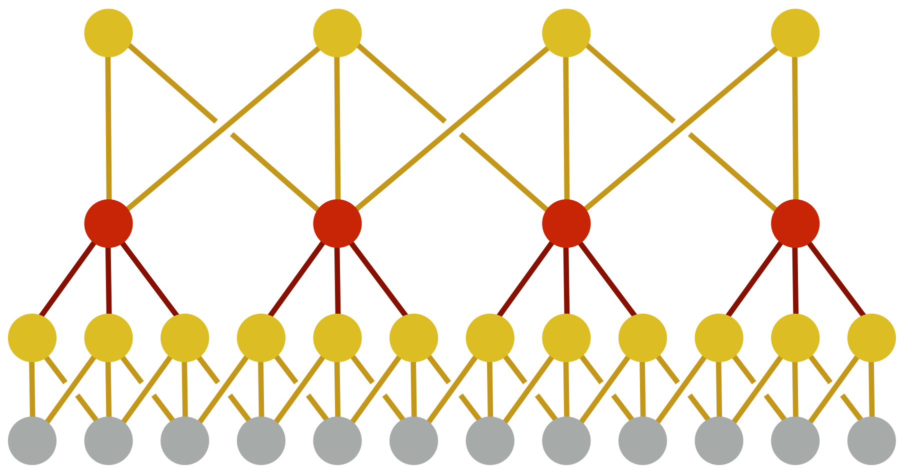
Deep Belief Networks



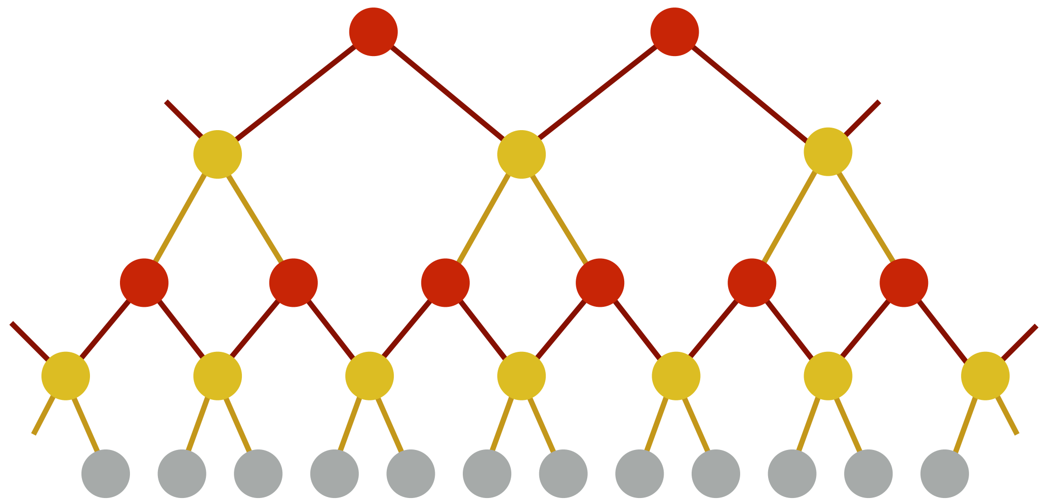
The "Renormalization Group"

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

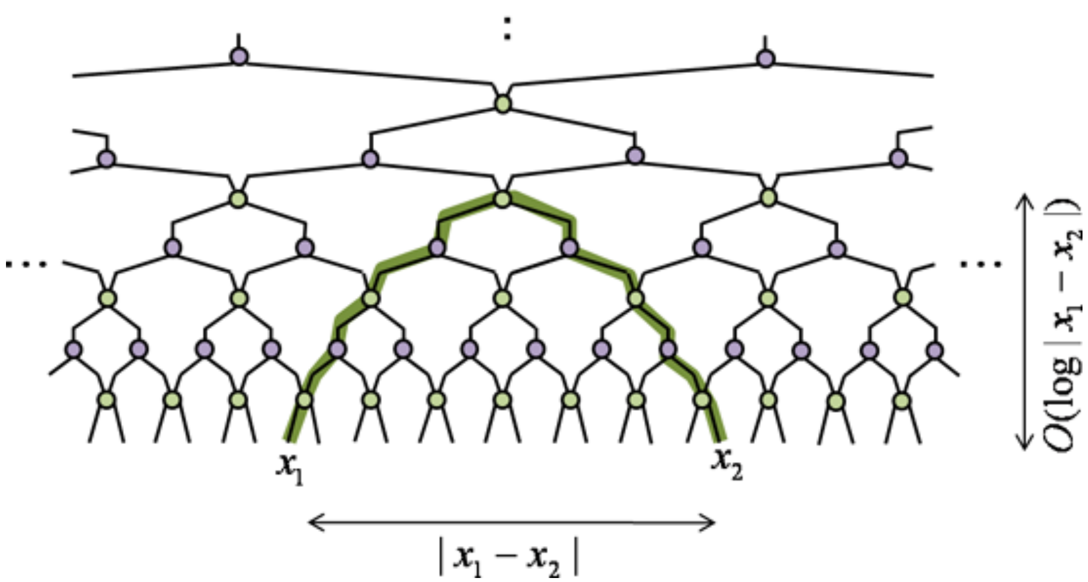
Convolutional neural network



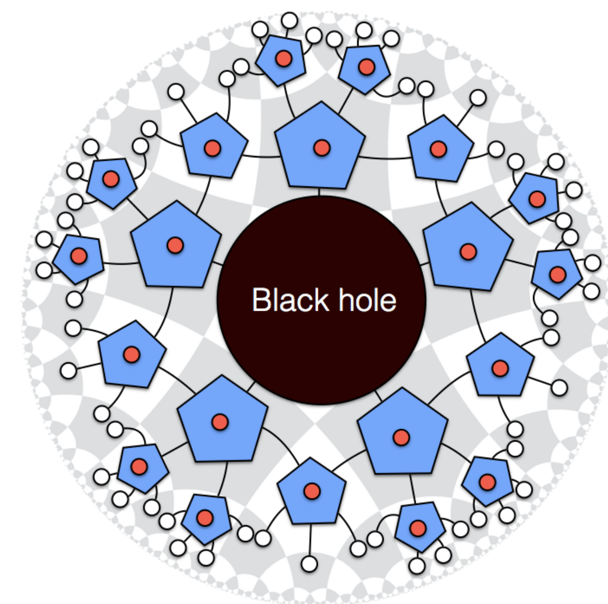
"MERA" tensor network



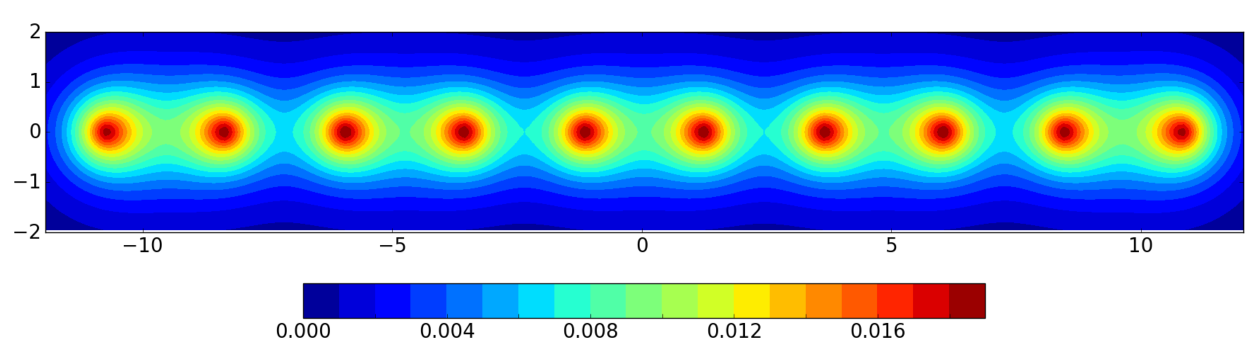
Impact of tensor networks in physics



Critical Phenomena

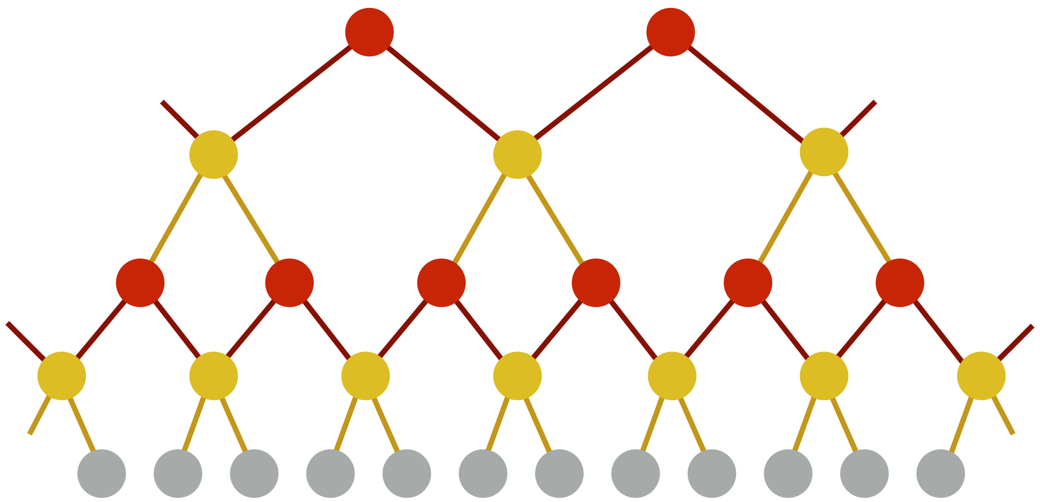


Quantum Gravity



High Precision Quantum Chemistry

Are tensor networks useful for machine learning?



This Talk

Tensor networks can compress weights of powerful machine learning models

Benefits include

- Linear scaling
- Adaptive optimization
- Feature sharing

Prior tensor networks + machine learning

Markov random field models

Novikov et al., Proceedings of 31st ICML (2014)

Large scale PCA

Lee, Cichocki, arxiv: 1410.6895 (2014)

Feature extraction of tensor data

Bengua et al., IEEE Congress on Big Data (2015)

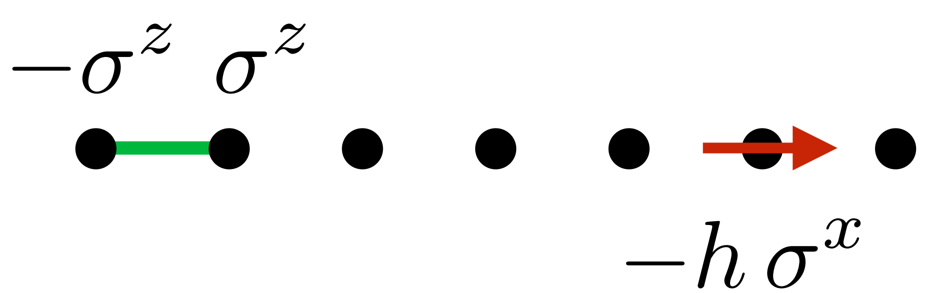
Compressing weights of neural nets

Novikov et al., Advances in Neural Information Processing (2015)

What are Tensor Networks?

Original setting is quantum mechanics

Spin model (Transverse field Ising model):



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

$\uparrow \ \uparrow \ \uparrow \ \uparrow \ \uparrow \ \uparrow \ \uparrow$ $\qquad h \ll 1$

$\rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow$ $\qquad h \gg 1$

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

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

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

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

$$\Psi^{\uparrow\downarrow\uparrow\uparrow\downarrow} \approx v_L^\dagger$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- Matrices can vary from site to site
- 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

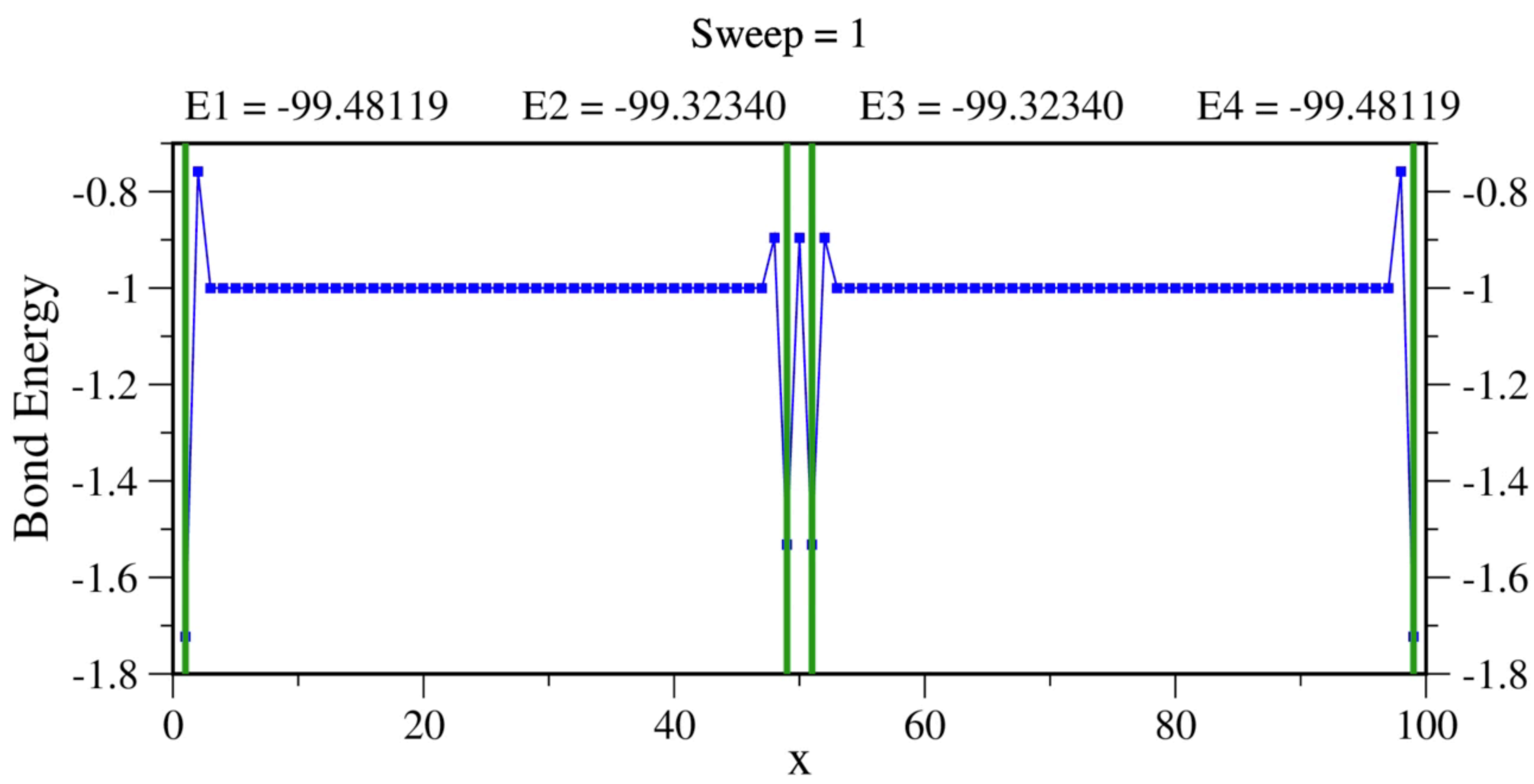
What have we gained?

By representing a wavefunction as an MPS with small matrices (small bond dimension m)

Then we've represented 2^N amplitudes using only ($2\ N\ m^2$) parameters

Efficient to compute properties of an MPS, or to optimize an MPS (DMRG algorithm)

MPS come with powerful optimization techniques (*DMRG algorithm*)

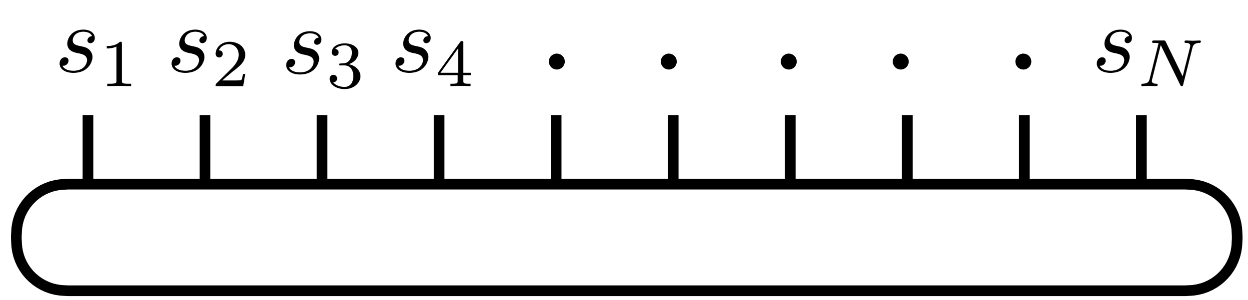


White, PRL **69**, 2863 (1992)

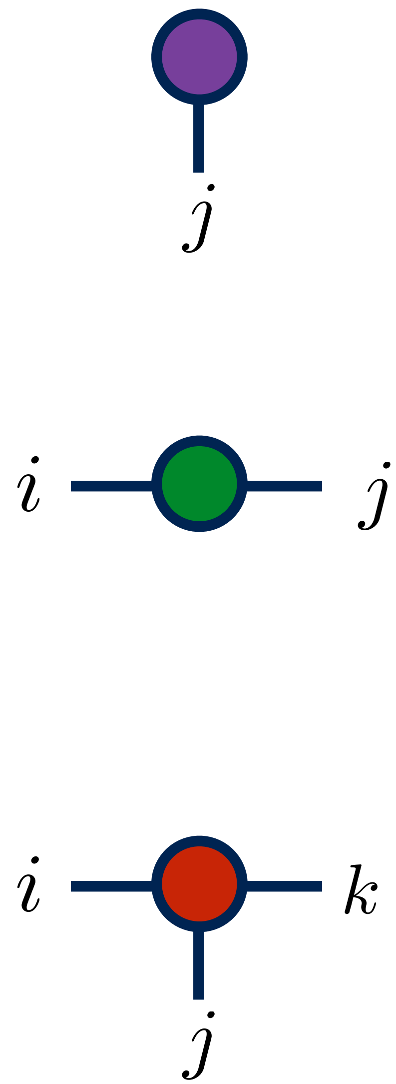
Stoudenmire, White, PRB **87**, 155137 (2013)

Tensor Diagrams (Briefly)

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

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

Diagrams for simple tensors

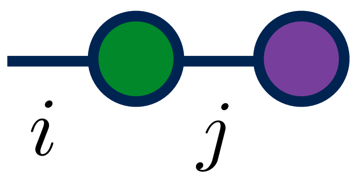


v_j

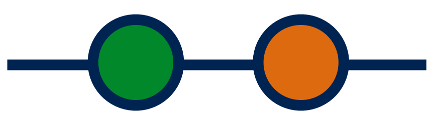
M_{ij}

T_{ijk}

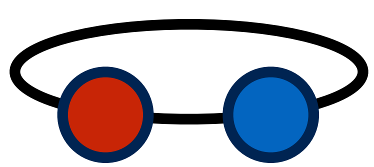
Joining lines implies contraction, can omit names



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



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

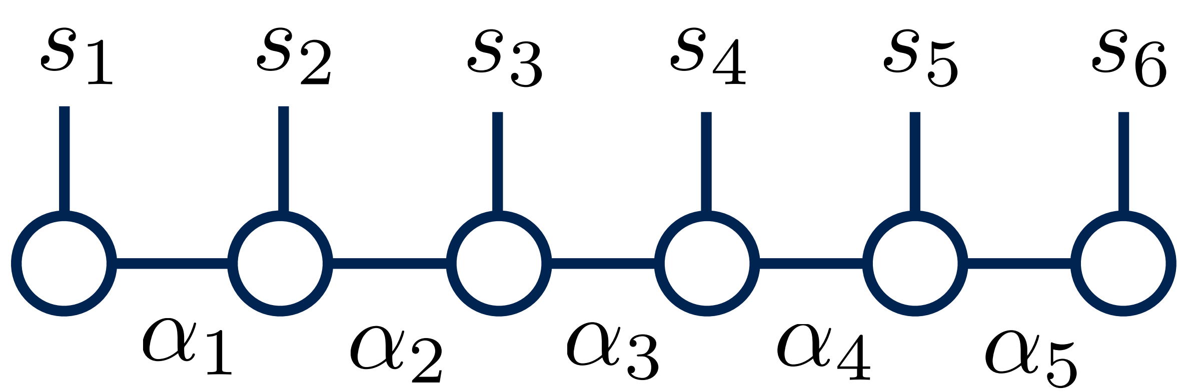


$$\longleftrightarrow \quad A_{ij} B_{ji} \quad = \mathrm{Tr}[AB]$$

Matrix product state in diagram notation

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

$=$

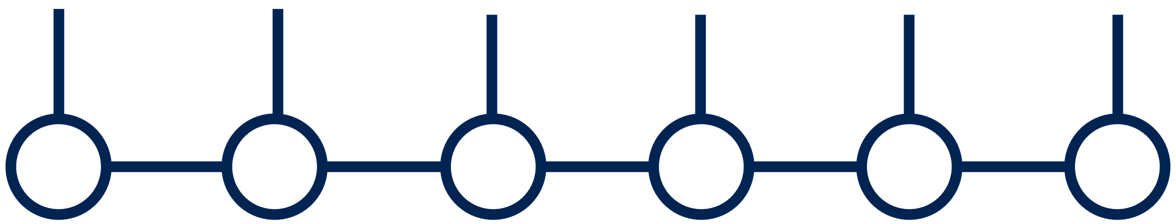


Can suppress index names, very convenient

Matrix product state in diagram notation

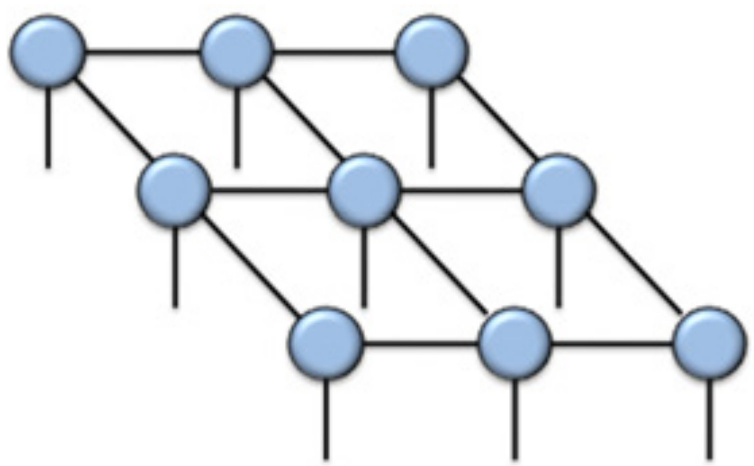
$$\Psi^{s_1 s_2 s_3 s_4 s_5 s_6} = \sum_{\alpha} M^{s_1}_{\alpha_1} M^{s_2}_{\alpha_1 \alpha_2} M^{s_3}_{\alpha_2 \alpha_3} M^{s_4}_{\alpha_3 \alpha_4} M^{s_5}_{\alpha_4 \alpha_5} M^{s_6}_{\alpha_5}$$

$=$



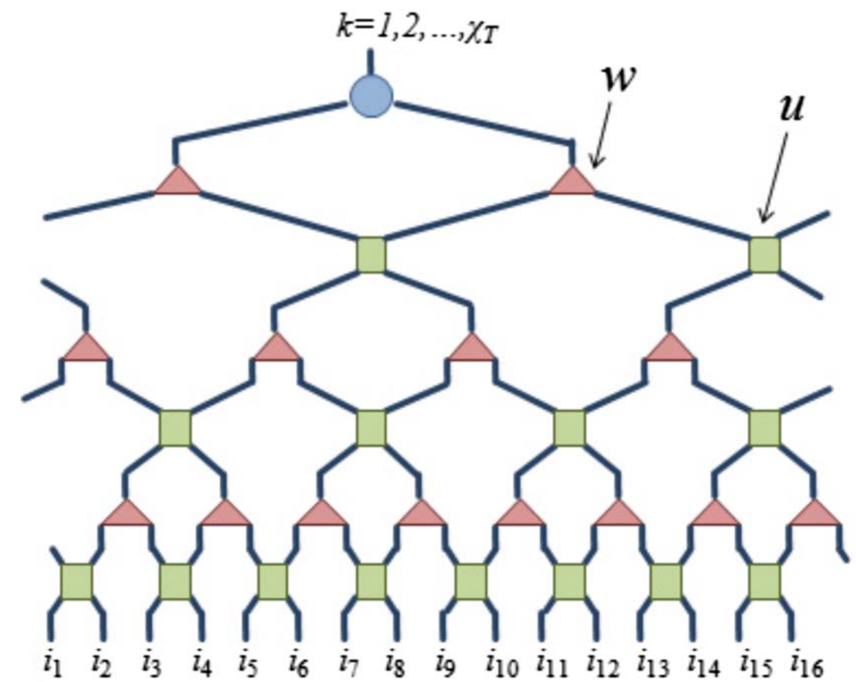
Can suppress index names, very convenient

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)

Learning with Tensor Networks

Proposal:

1. Lift data to exponentially higher space (feature space = Hilbert space)

2. Apply linear classifier in feature space

3. Compress weights using a tensor network

Following slides use feature map of Novikov et al.

Novikov, Trofimov, Oseledets, "Exponential Machines", arxiv:1605.03795

Stoudenmire, Schwab, "Supervised Learning with Tensor Networks", arxiv:1605.05775

Original / raw data vectors

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

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

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

1. Lift data to exponentially higher space (feature space = Hilbert space)

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

↓ *lift*

$$\Phi(\mathbf{x}) = (1, x_1, x_2, x_3, \ldots$$ *singles*

$$x_1 x_2,\ x_1 x_3,\ x_2 x_3, \ldots$$ *pairs*

$$x_1 x_2 x_3,\ x_1 x_2 x_4, \ldots$$ *triples*

$$\ldots$$ ...

$$x_1 x_2 x_3 \cdots x_N)$$ *N-tuple*

2. Apply linear classifier in feature space

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

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

Weights are an N-index tensor
Just like an N-site wavefunction

N=3 example:

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

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

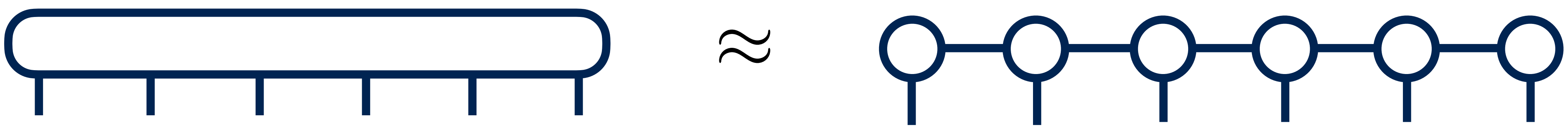
$$+ W_{110}\, x_1 x_2 + W_{101}\, x_1 x_3 + W_{011}\, x_2 x_3$$

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

Contains linear classifier, and various poly. kernels

3. Compress weights as a tensor network

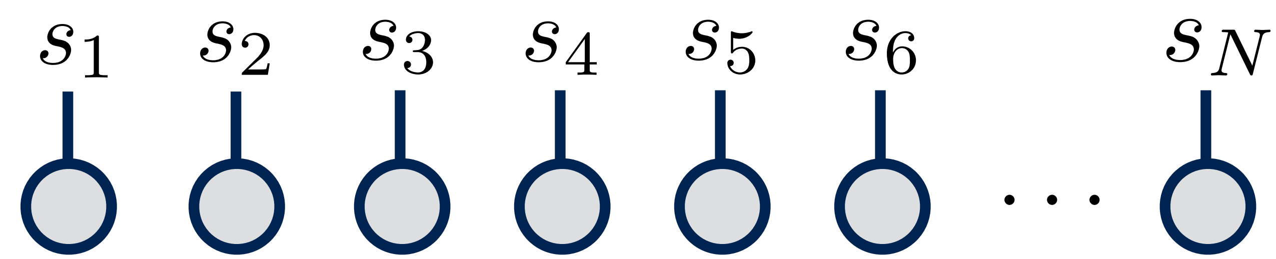
$$W_{s_1 s_2 s_3 \cdots s_N} \approx M_{s_1} M_{s_2} M_{s_3} \cdots M_{s_N}$$

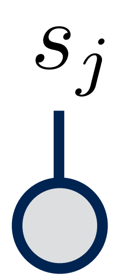


Could also use MERA or PEPS instead of MPS

Tensor diagrams of the approach

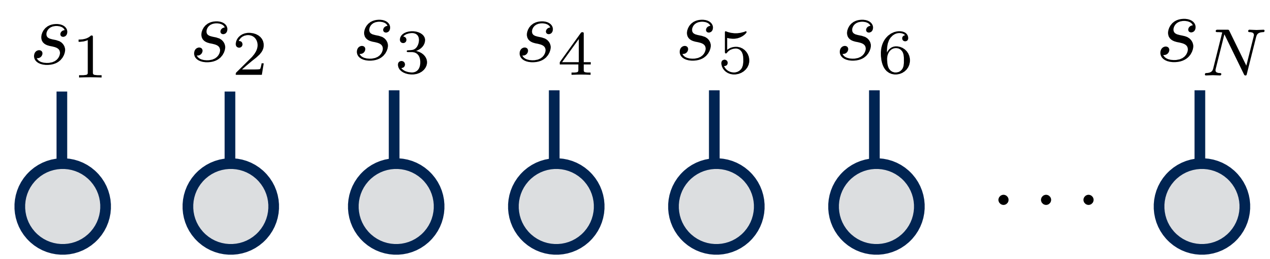
$$\mathbf{x} \longrightarrow \Phi(\mathbf{x}) =$$

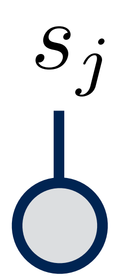




$$= \begin{bmatrix} 1 \\ x_j \end{bmatrix}$$

Tensor diagrams of the approach

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



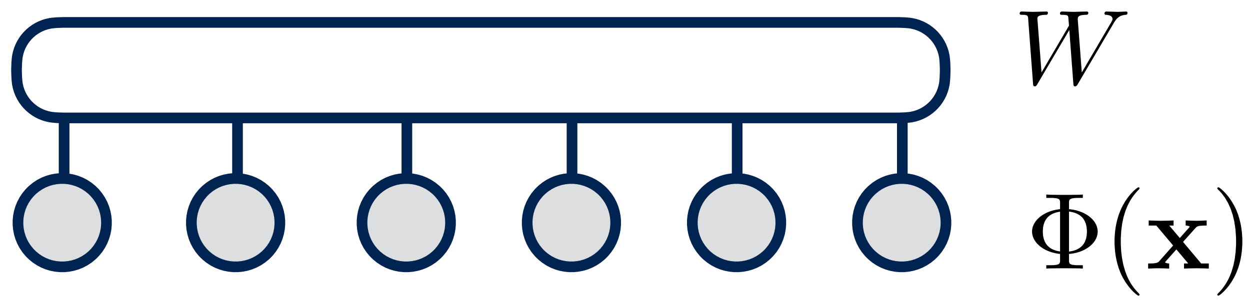
$$= \begin{bmatrix} 1 \\ x_j \end{bmatrix}$$

Other choices include:

$$\begin{bmatrix} 1 \\ x_j \\ x_j^2 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \cos\left(\frac{\pi}{2} x_j\right) \\ \sin\left(\frac{\pi}{2} x_j\right) \end{bmatrix}$$

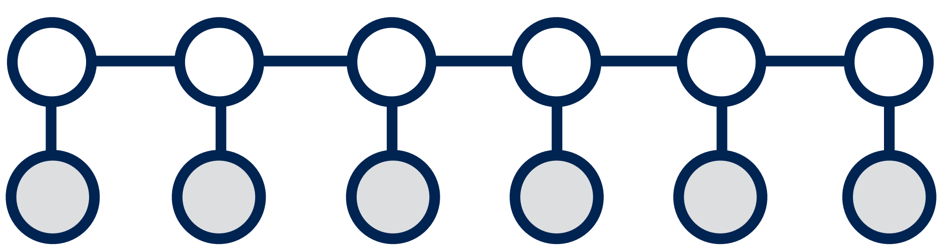
Tensor diagrams of the approach

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



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

$$\approx$$



Linear scaling

Can use algorithm similar to DMRG to optimize

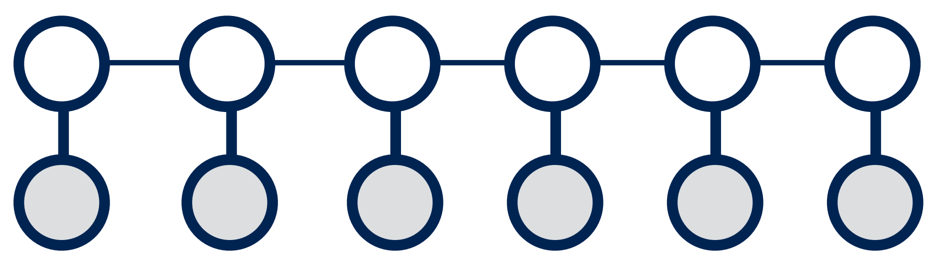
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$



W

$\Phi(\mathbf{x})$

Linear scaling

Can use algorithm similar to DMRG to optimize

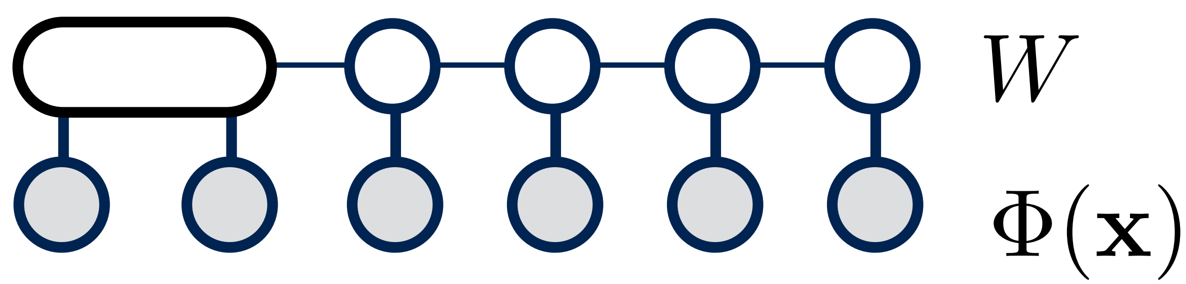
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$



Linear scaling

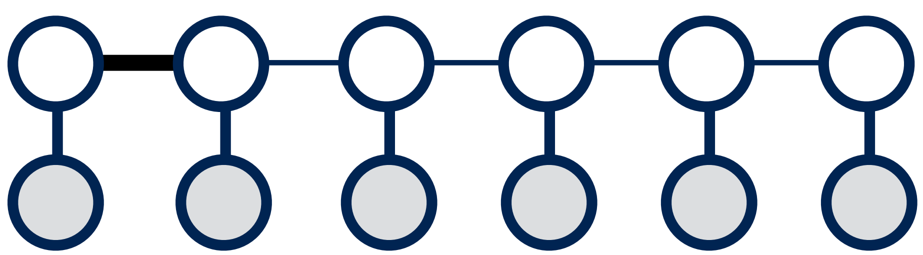
Can use algorithm similar to DMRG to optimize

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

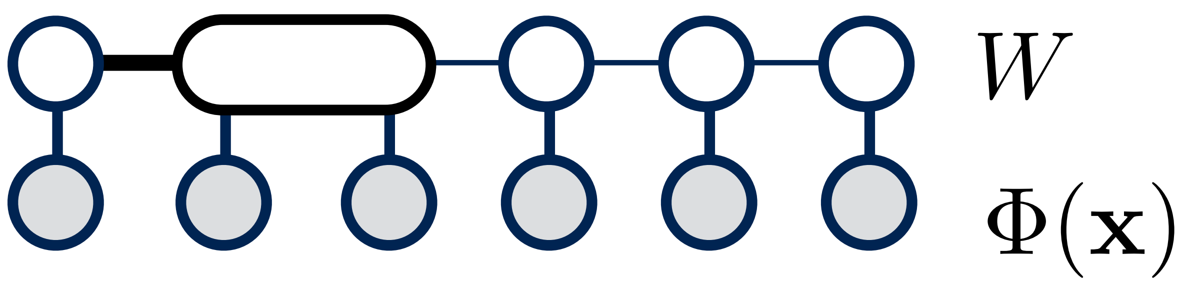
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$



Linear scaling

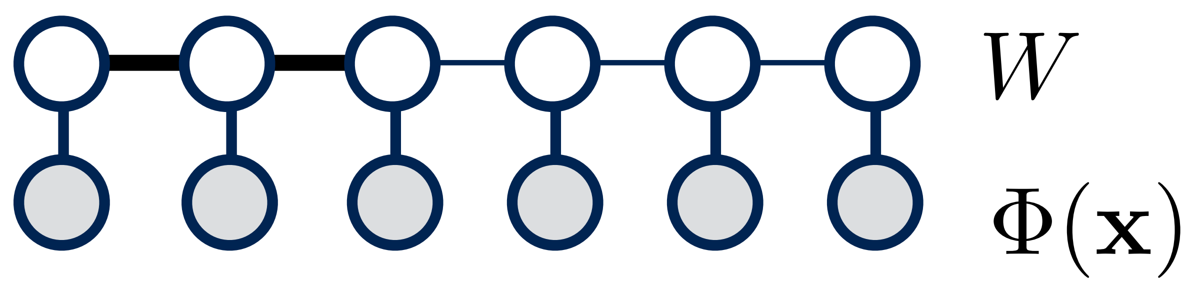
Can use algorithm similar to DMRG to optimize

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

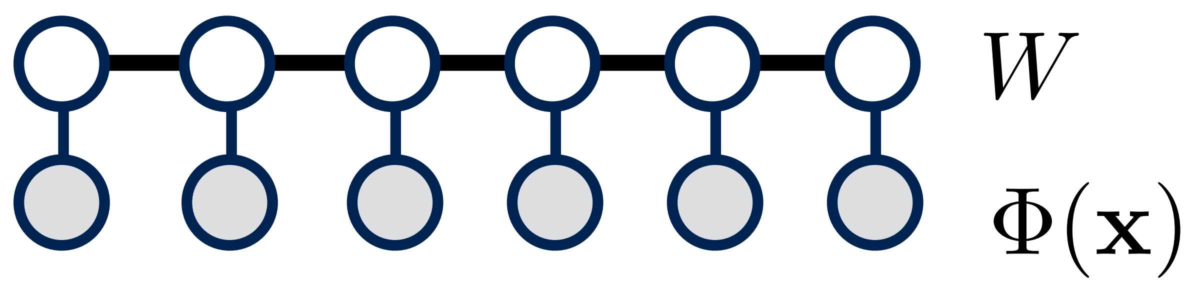
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$



Could improve with stochastic gradient

Linear scaling

Model similar to kernel learning

Can train without "kernel trick", avoiding its N_T^2 scaling problem

Does the weight tensor obey an "area law"?

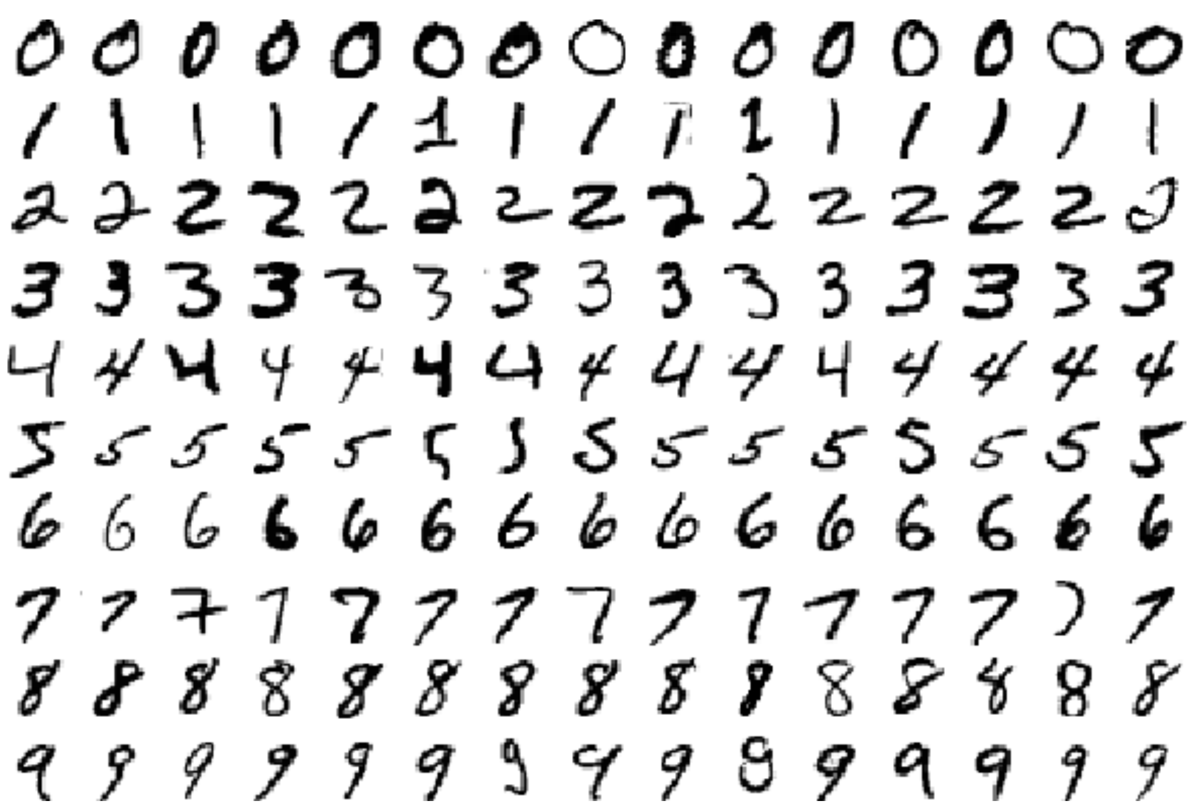
More entangled than a ground-state wavefunction?
Less entangled?

Do an experiment to find out...

MNIST Experiment

MNIST is a benchmark data set of grayscale handwritten digits (labels $\ell = 0,1,2,\ldots,9$)

60,000 labeled training images
10,000 labeled test images



MNIST Experiment

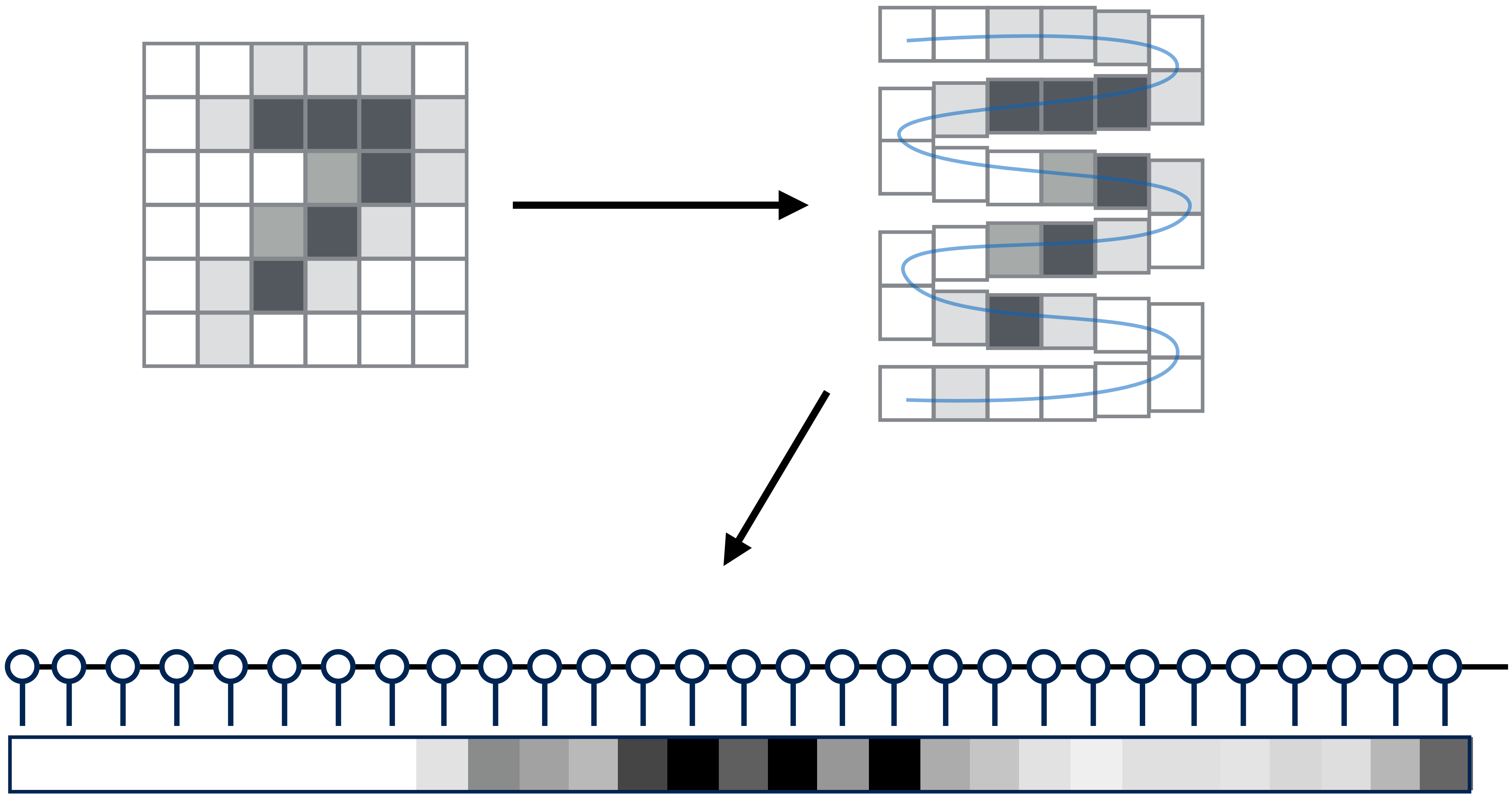
Details:

- Shrink images from 28x28 $\longrightarrow$ 14x14
- Trained 10 models* to distinguish each digit, largest output is prediction
- Minimize quadratic cost function

$$C = \frac{1}{N_T} \sum_{n=1}^{N_T} (W^\ell \Phi(\mathbf{x}_n) - y_n^\ell)^2 + \lambda |W|^2$$

MNIST Experiment

One-dimensional mapping

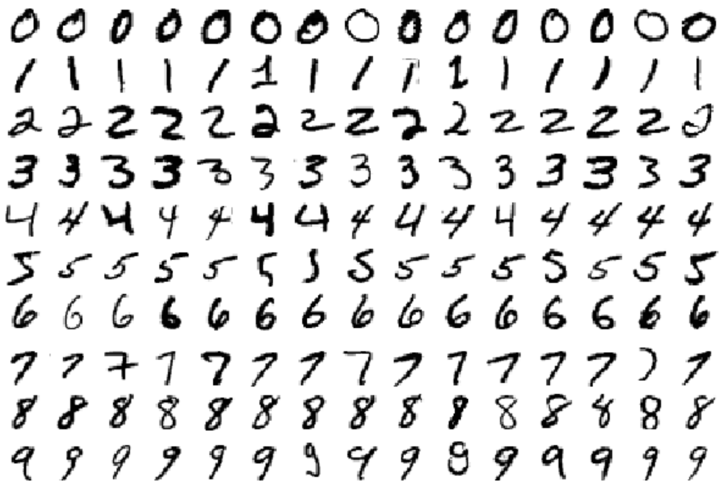


MNIST Experiment

Results:

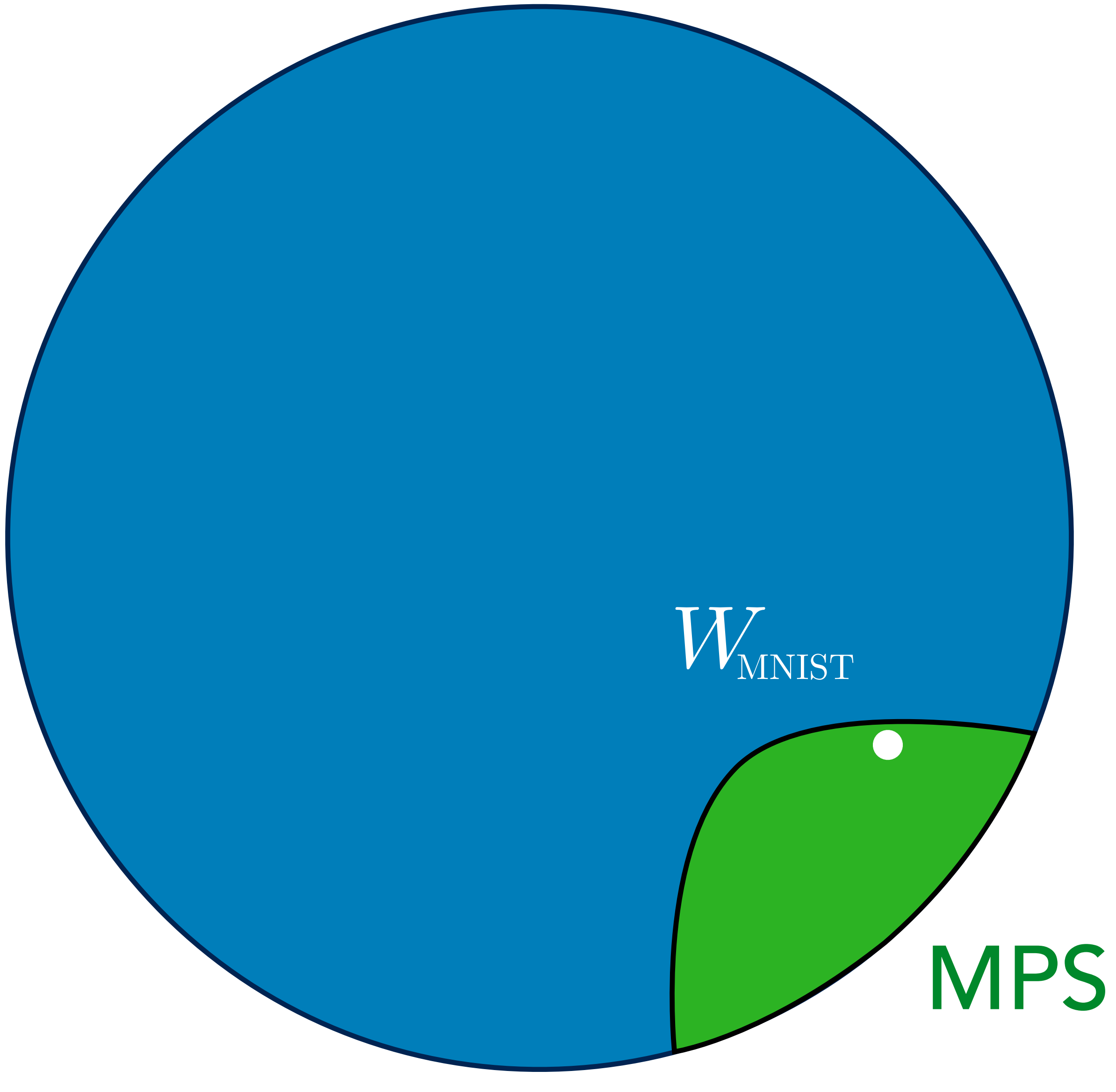
Bond dimension	Test Set Error
m = 10	~5% (500/10,000 incorrect)
m = 20	~2% (200/10,000 incorrect)
m = 120	0.97% (97/10,000 incorrect)

MNIST Experiment

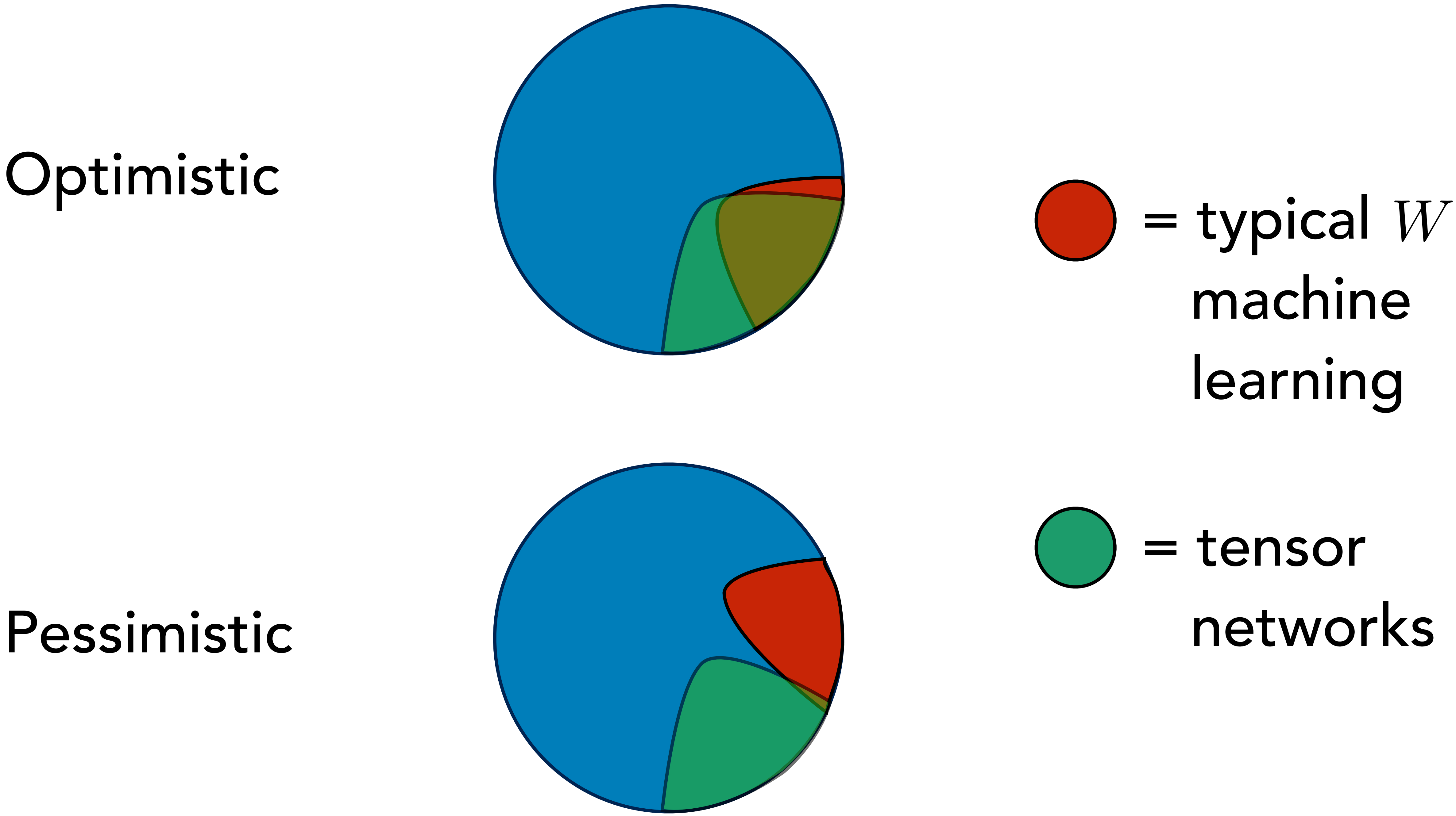


→ *Demo*

MNIST in friendly neighborhood of Hilbert space (feature space)

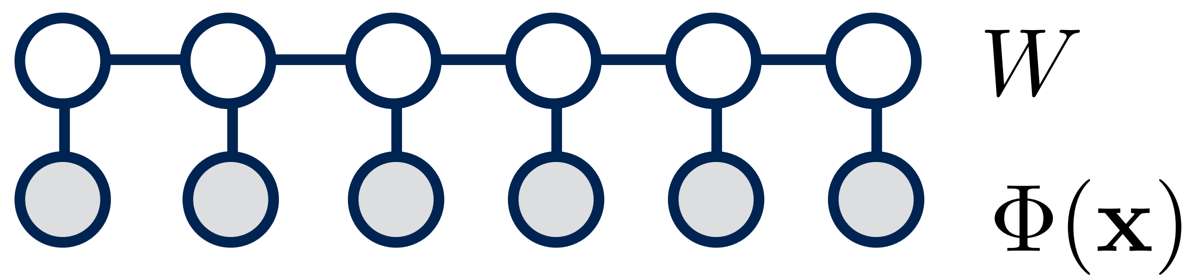


Situation for other data sets?

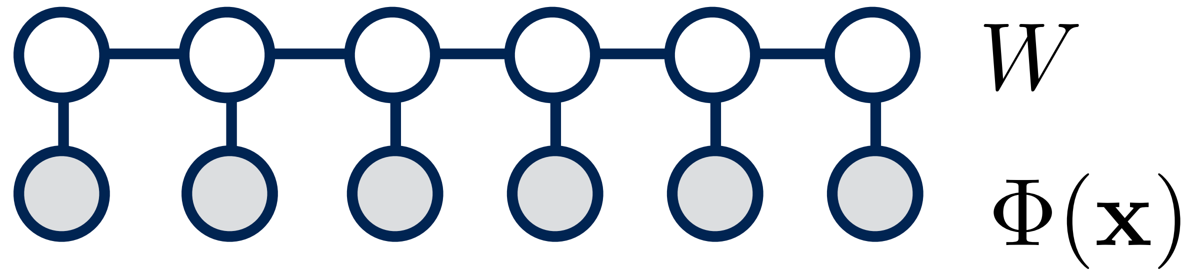


Benefits of Tensor Network Models

$f(\mathbf{x}) =$



$f(\mathbf{x}) =$

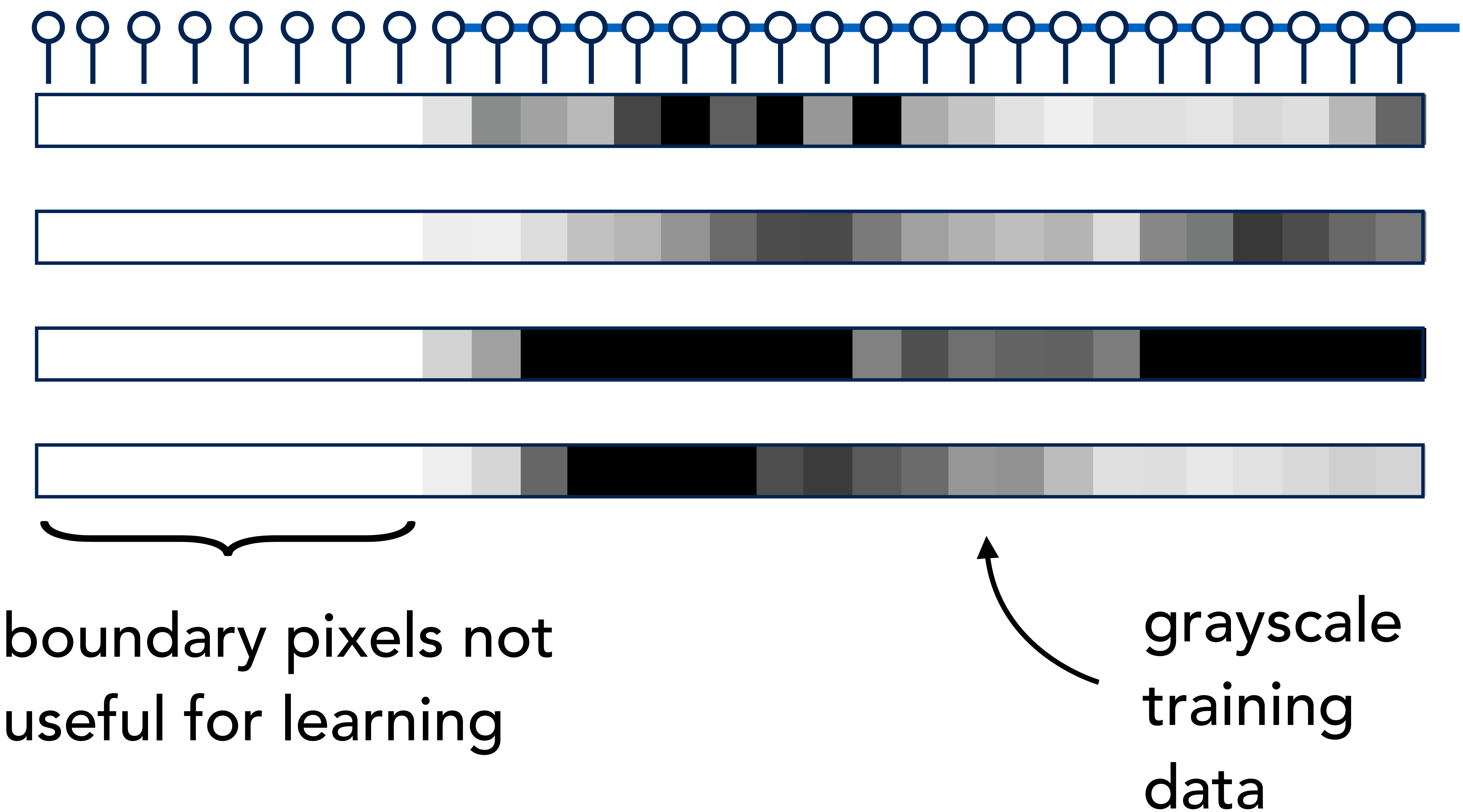


Many interesting benefits of using tensor network weights.

Two benefits:

1. Adaptive training
2. Feature sharing

1. Tensor networks are adaptive

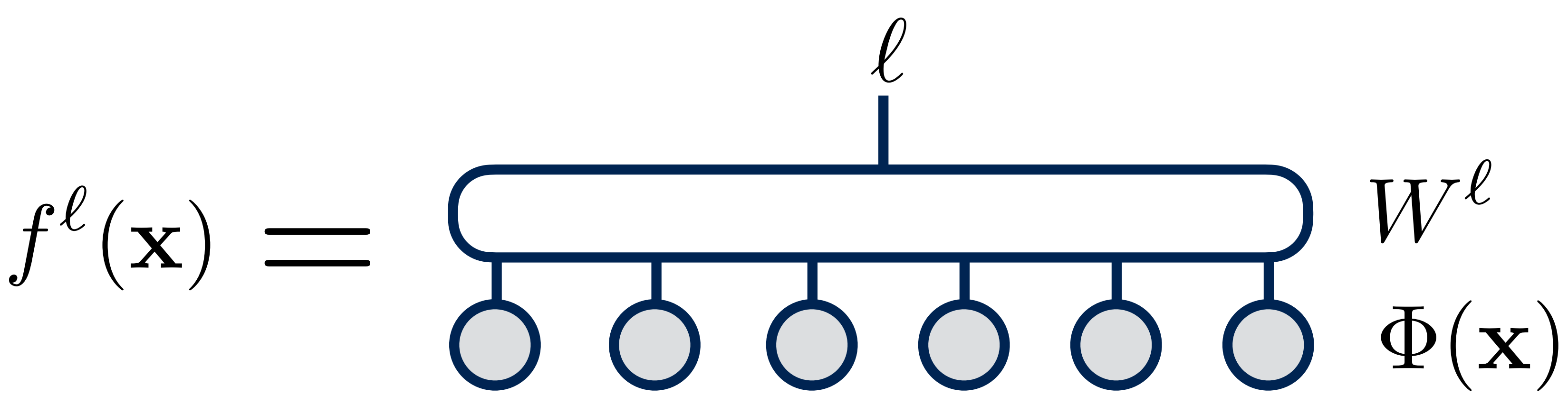


2. Feature sharing

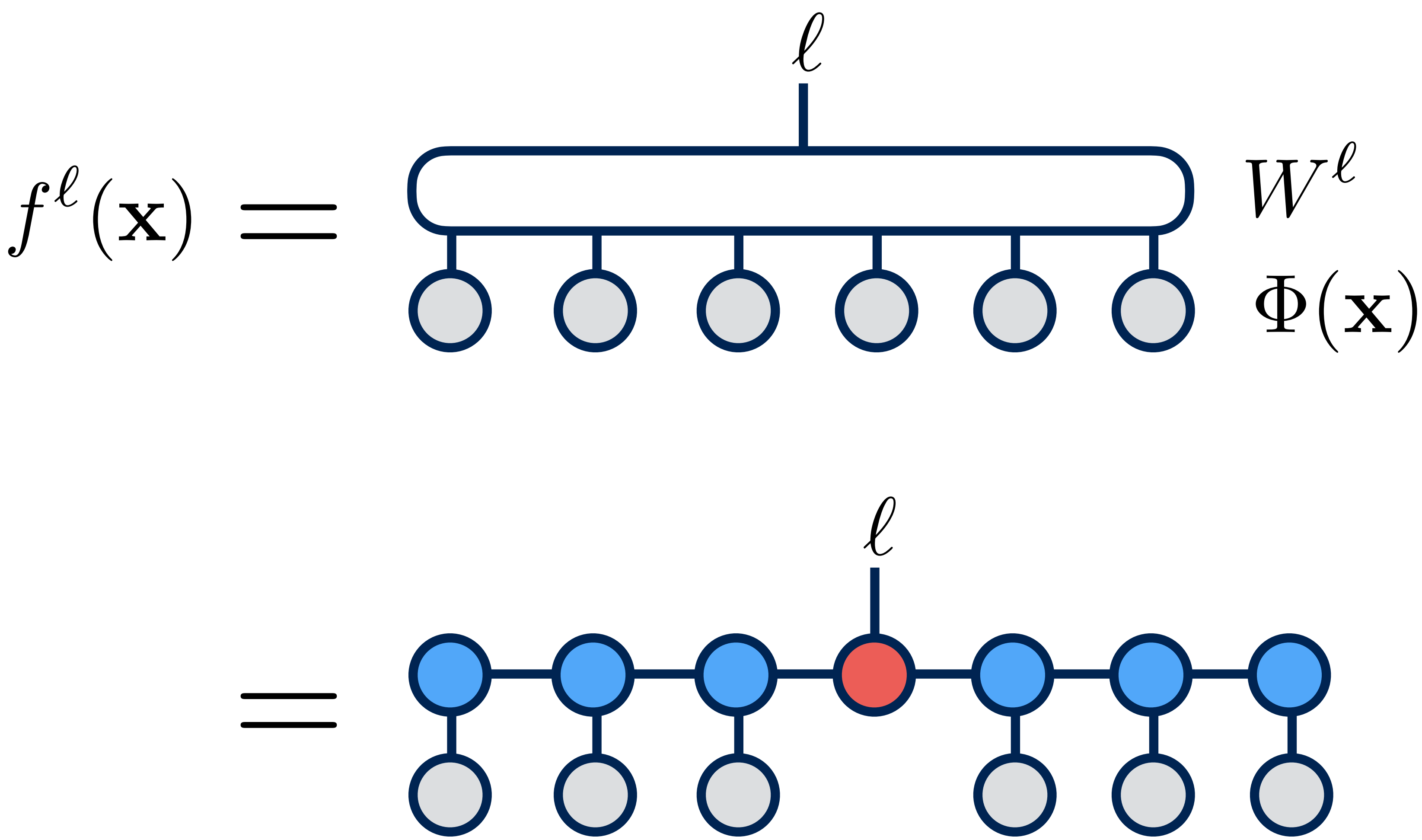
Multi-class decision function $f^{\ell}(\mathbf{x}) = W^{\ell} \cdot \Phi(\mathbf{x})$

Index ℓ runs over possible labels

Predicted label is $\mathrm{argmax}_{\ell} |f^{\ell}(\mathbf{x})|$



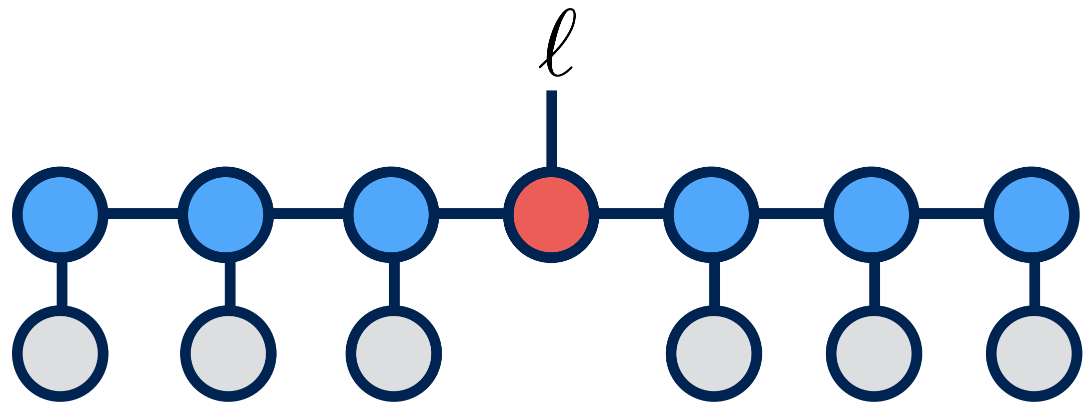
2. Feature sharing



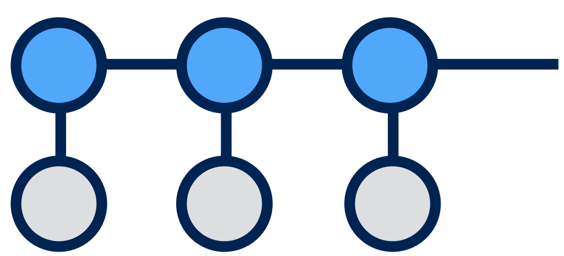
- Different central tensors
- "Wings" shared between models
- Regularizes models

2. Feature sharing

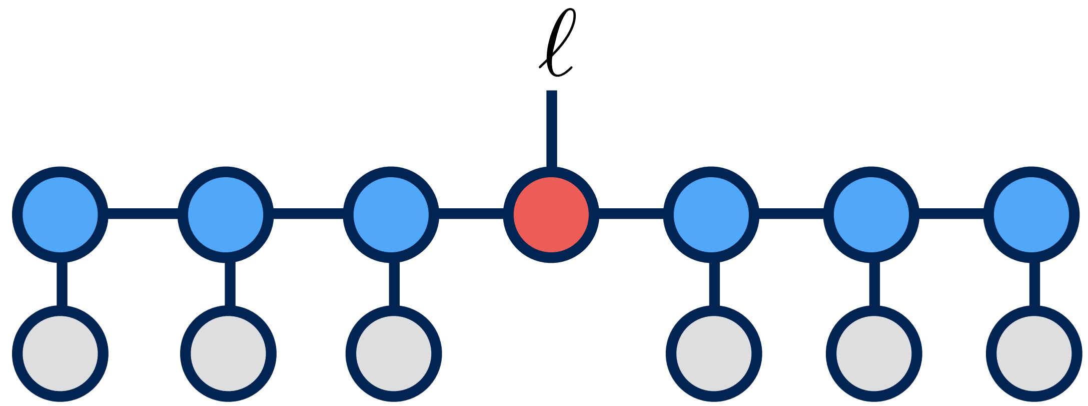
$$f^{\ell}(\mathbf{x}) =$$



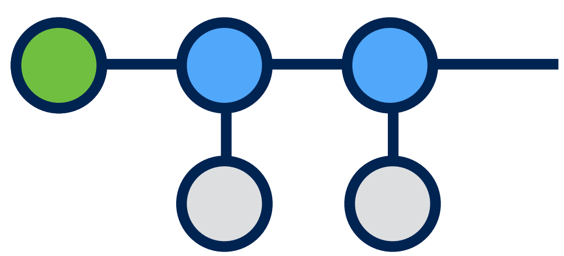
Progressively learn shared features



2. Feature sharing

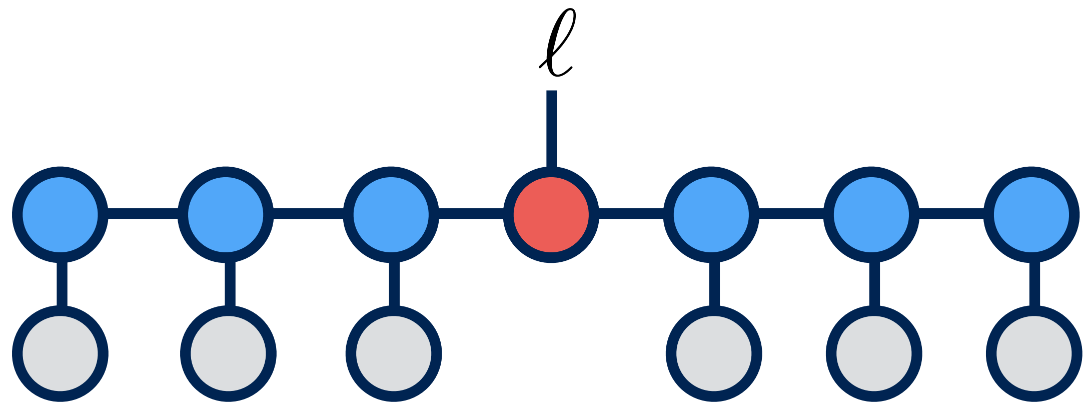
$f^{\ell}(\mathbf{x}) =$ 

Progressively learn shared features

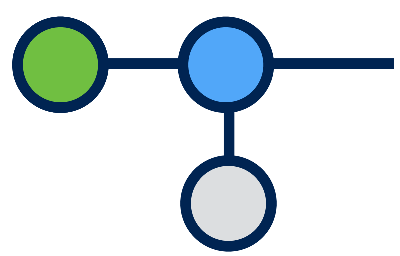


2. Feature sharing

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

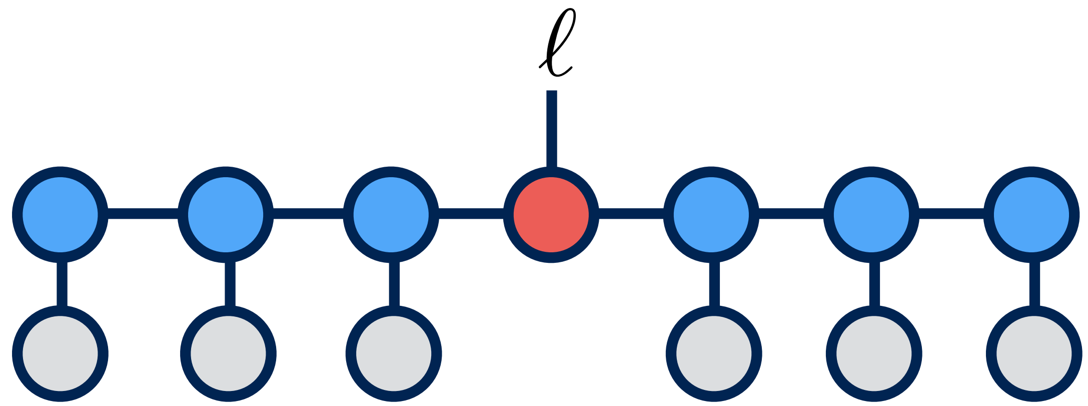


Progressively learn shared features

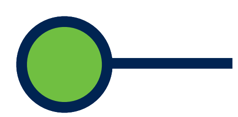


2. Feature sharing

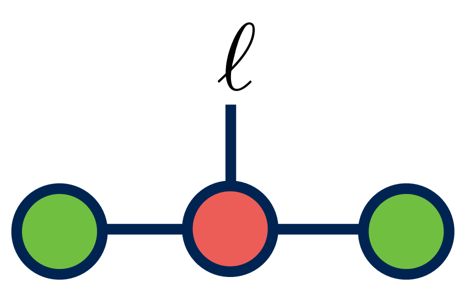
$$f^{\ell}(\mathbf{x}) =$$



Progressively learn shared features



Deliver to central tensor



Implications for quantum computing?

Weights formally inhabit same space as quantum Hilbert space (space of wavefunctions)

Negative Outlook 😈

Weights have low entanglement, quantum computer not needed

Positive Outlook 😎

Quantum computer could train extremely expressive models

Tensor networks equivalent to finite-depth quantum circuits...

Connections to Other Approaches

- **Graphical models**: like tensor networks but with positive weights (Rolfe, "Multifactor Expectation Maximization...")
- **Weighted Finite Automata**: like translation invariant MPS, trained with *spectral method* (Balle, "Spectral Learning..." Mach Learn (2014) 96:33-63)
- **Neural Networks**: "ConvAC" neural networks with linear activation and product pooling equivalent to tensor networks (Levine, "Deep Learning and Quantum Entanglement..." arxiv:1704.01552)

ITENSOR

ITensor Codes

Open-source codes based on ITensor for a variety projects and tasks. If you have a high-quality code you'd like listed here, please contact us. Codes extending core ITensor features may become candidates for inclusion in ITensor at a later date.

Name	**Contributors**	**Description**
Finite T MPS	Benedikt Bruognolo Miles Stoudenmire	Codes for finite temperature calculations with MPS techniques, including the minimally entangled typical thermal states (METTS) algorithm applied to 2D systems.
Tensor Network Machine Learning	Miles Stoudenmire	Handwriting recognition using matrix product states (MPS) to parameterize the weights of the model, and a DMRG-like algorithm to optimize.
Parallel DMRG	Miles Stoudenmire	Real-space parallel DMRG code. Works for both single MPO Hamiltonians and Hamiltonians that are a sum of separate MPOs. Uses MPI to communicate DMRG boundary tensors across nodes.