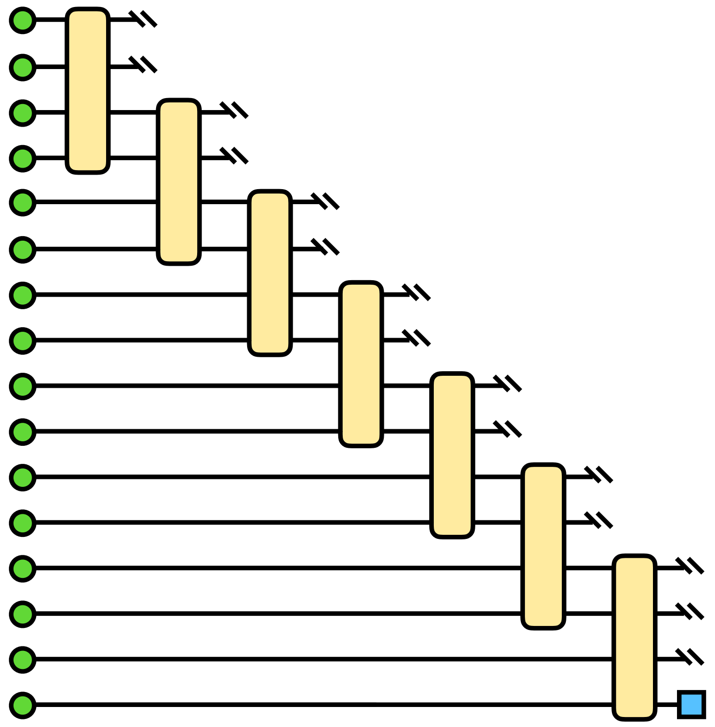
Applications of Tensor Networks: Machine Learning & Quantum Computing



E.M. Stoudenmire

Aug 2018 - ICTP Trieste

FLATIRON INSTITUTE

SIMONS FOUNDATION

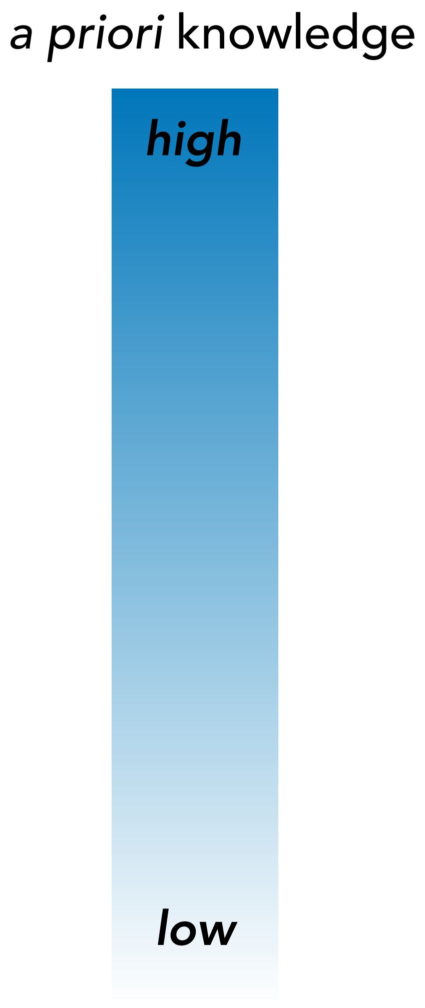
Outline:

- **Yesterday:**
  - intro to <u>tensor networks</u>, mainly matrix product states (MPS)
  - computations with MPS
  - intro to machine learning & tensor-network M.L.
- **Today:**
  - tensor network machine learning
  - quantum computing with tensor networks

Basics of Machine Learning, Continued…

Types of learning tasks:

- Supervised learning (labeled data)
- Unsupervised learning (unlabeled data)
- Reinforcement learning ('reward' data)



Supervised Learning

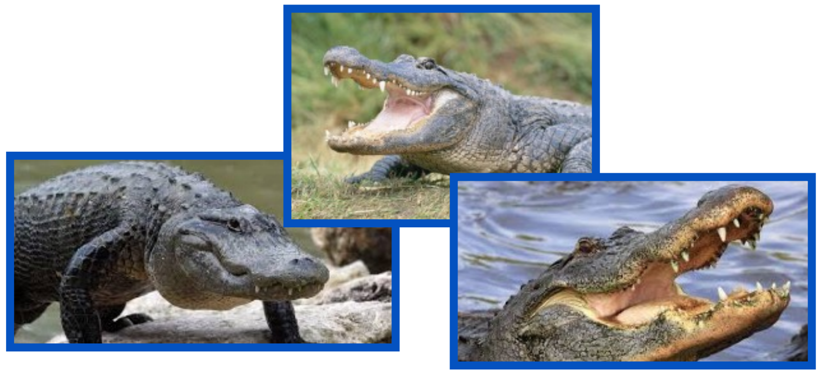
Given labeled training data (labels A and B)

Find *decision function* $f(\mathbf{x})$

$$f(\mathbf{x}) > 0 \qquad \mathbf{x} \in A$$

$$f(\mathbf{x}) < 0 \qquad \mathbf{x} \in B$$

Example: identify photos of alligators and bears



Unsupervised Learning

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

- Find function   such that
- Find function   such that
- Find data clusters and which data belongs to each cluster
- Discover reduced representations of data for other learning tasks (e.g. supervised)

General Philosophy of Machine Learning

- Solution to problem just some function $y(\mathbf{x})$
- Parameterize very flexible functions $f(\mathbf{x})$ (prefer convenient over "correct")
- Of all f that come closest to y for training data, prefer the simplest f

Model Architectures

Let's discuss the 3 most used types of models (increasing complexity)

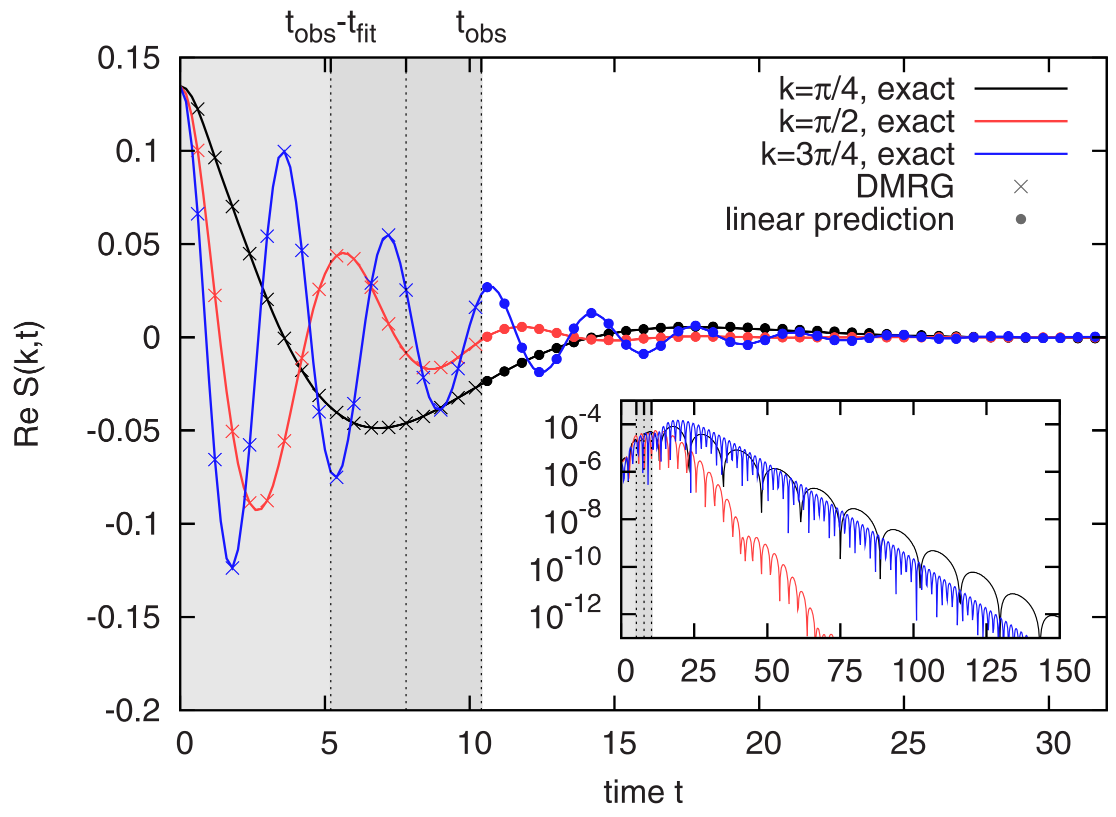
- The linear model
- Kernel learning / support vector machines
- Neural networks

The linear model

$$f(\mathbf{x}) = W \cdot \mathbf{x} + W_0$$

Where W and W_0 are the weights to be learned

Can be surprisingly powerful, and a useful starting point



Barthel, Schollwöck, White, PRB 79, 245101

Example: Linear Supervised Learning

Recall strategy:

given training set $\{\mathbf{x}_j, y_j\}$, minimize cost function

$$C = \frac{1}{N_T} \sum_j (f(\mathbf{x}_j) - y_j)^2 \qquad y_j = \begin{cases} +1 & \mathbf{x}_j \in A \\ -1 & \mathbf{x}_j \in B \end{cases}$$

by varying adjustable params of f

Cost function measures distance of trial function $f(\mathbf{x}_j)$ from idealized "indicator" function y_j

Example: Linear Supervised Learning

Cost function for linear model:

$$C = \frac{1}{2N_T} \sum_j (W \cdot \mathbf{x}^{(j)} - y^{(j)})^2$$

Gradient with respect to nth weight component

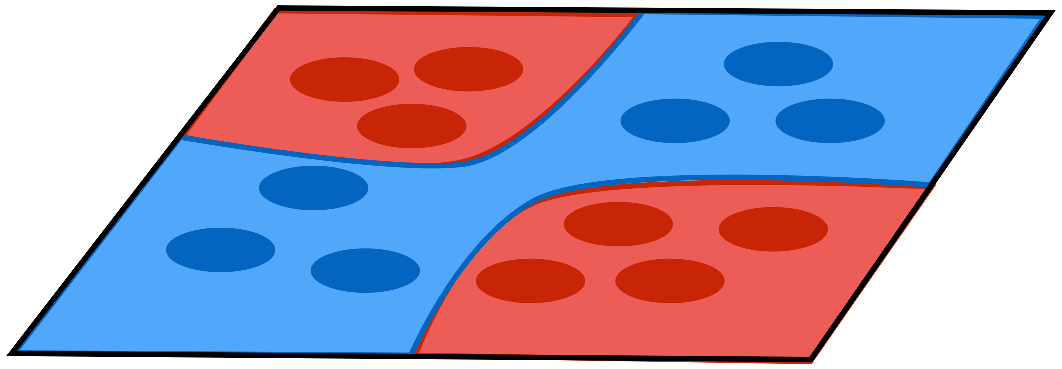
$$\frac{\partial C}{\partial W_n} = \frac{1}{N_T} \sum_j (W \cdot \mathbf{x}^{(j)} - y^{(j)}) \;\; x_n^{(j)}$$

Update Wn with negative gradient times small step α

$$W_n \leftarrow W_n - \alpha \frac{\partial C}{\partial W_n}$$

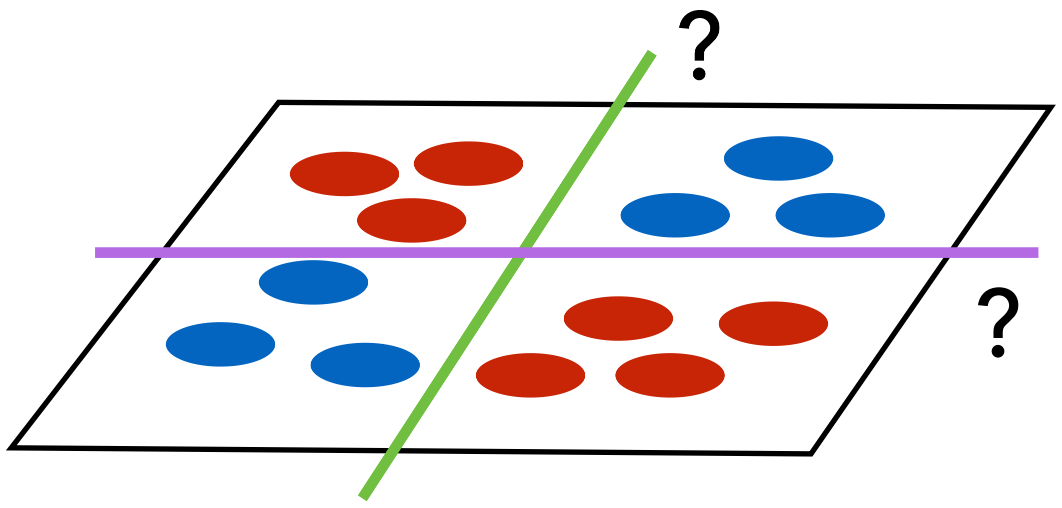
Kernel learning

Want $f(\mathbf{x})$ to separate classes, say



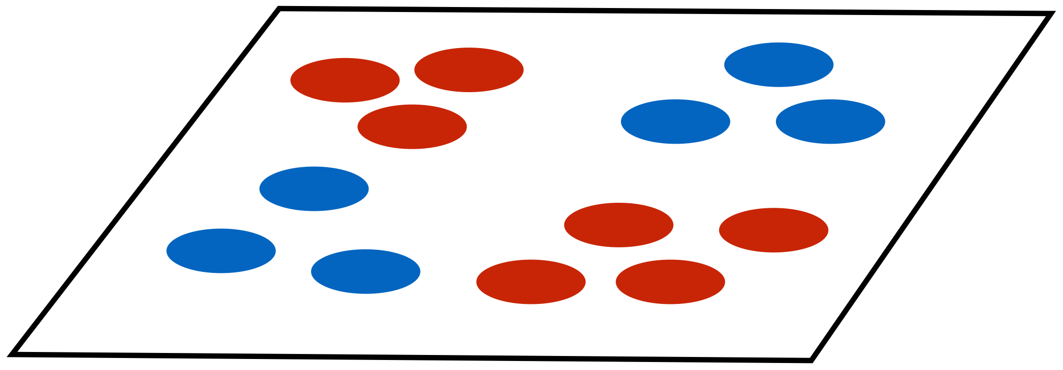
Linear classifier
may be insufficient

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



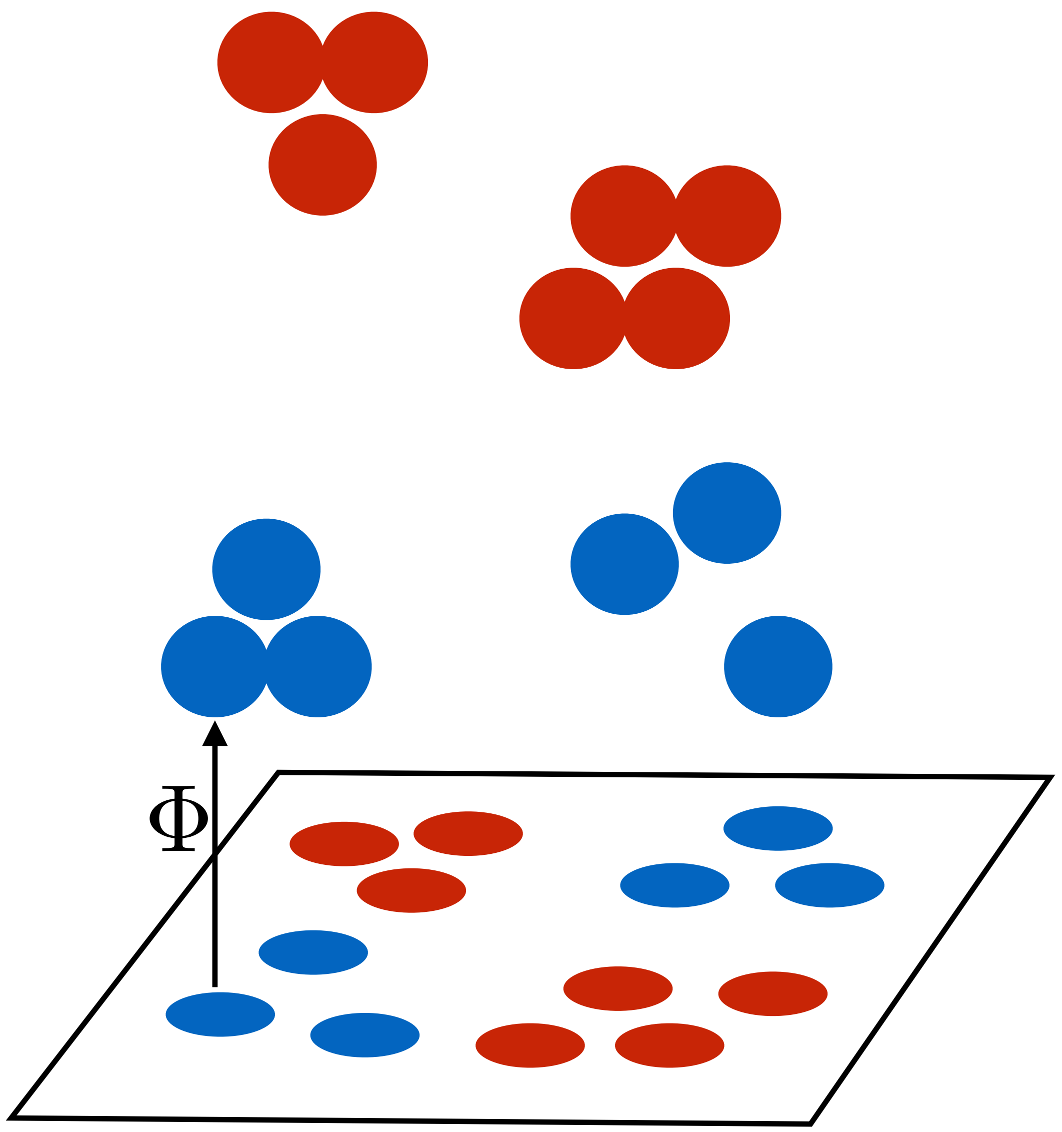
Kernel learning

Apply non-linear "feature map" $\mathbf{x} \rightarrow \Phi(\mathbf{x})$



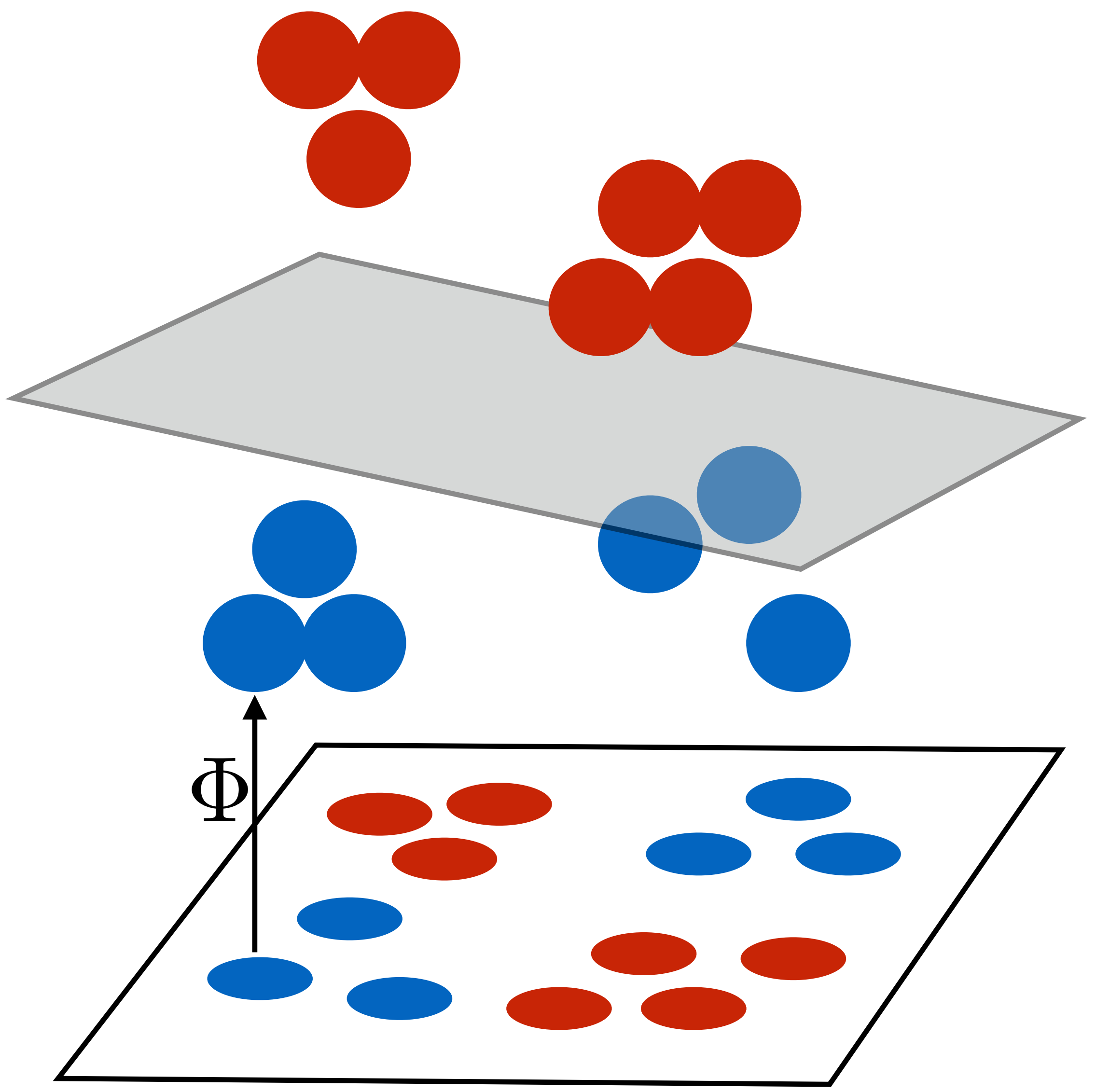
Kernel learning

Apply non-linear "feature map" $\mathbf{x} \to \Phi(\mathbf{x})$



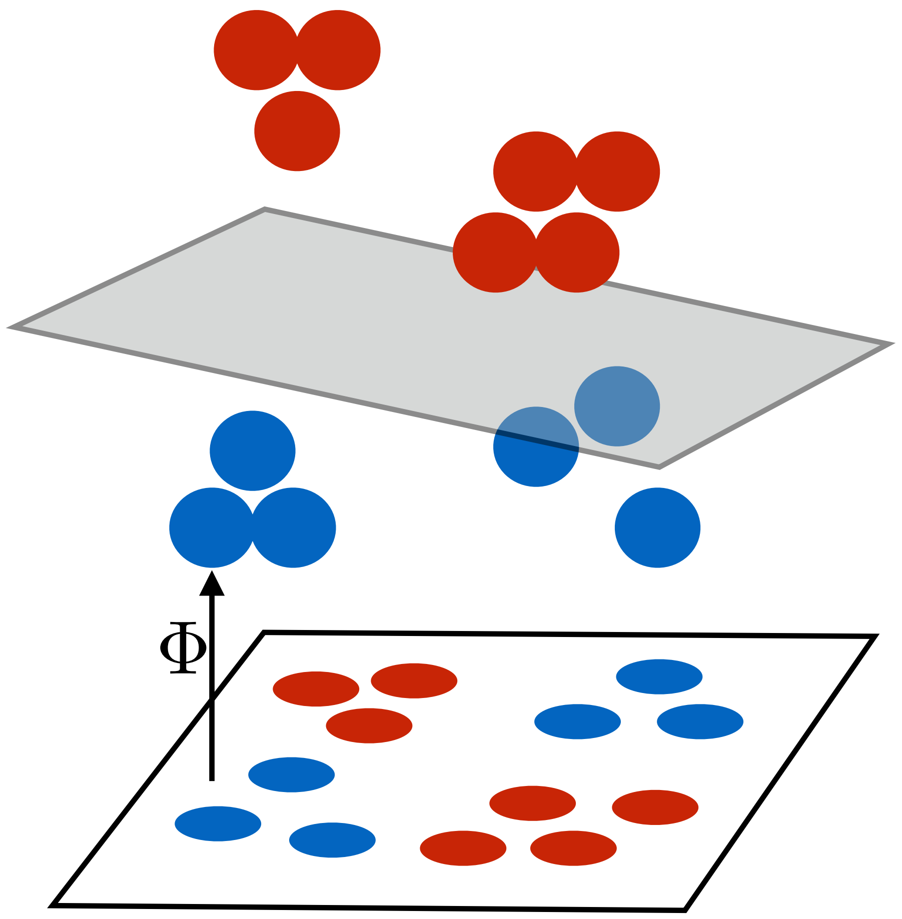
Kernel learning

Apply non-linear "feature map" $\mathbf{x} \to \Phi(\mathbf{x})$



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

Kernel learning

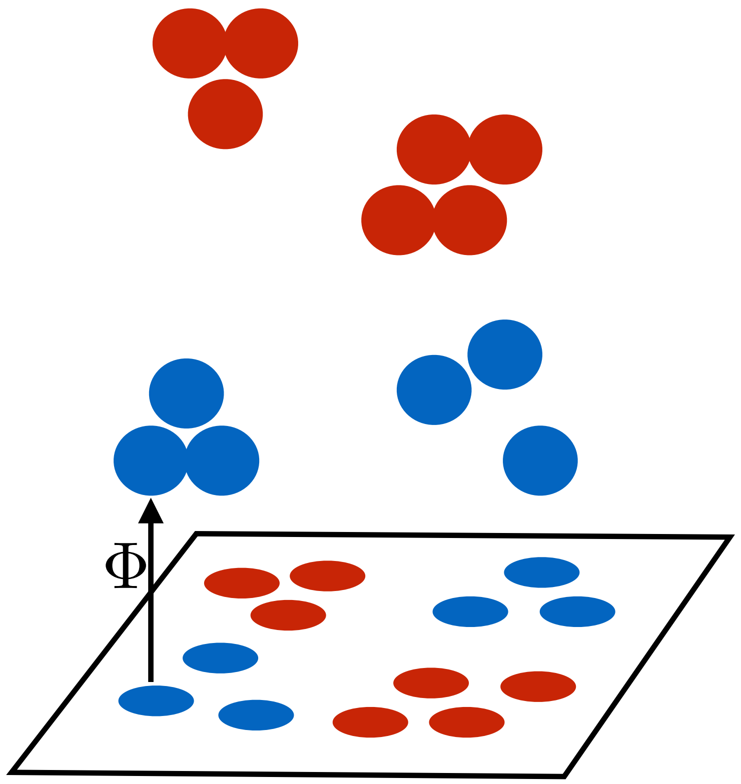


Decision function $$\boxed{f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})}$$

Linear classifier in *feature space*

Kernel learning

Example of *feature map*



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

$$\Phi(\mathbf{x}) = (1,\, x_1,\, x_2,\, x_3,\, x_1x_2,\, x_1x_3,\, x_2x_3)$$

$\mathbf{x}$ is "lifted" to feature space

Kernel learning

Technical notes:

- Also called "support vector machine" when using a particular choice of cost function

- Name "kernel learning" comes from idea that $\Phi(\mathbf{x})$ may be too high dimensional, yet $K_{ij} = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ may be efficiently computable, enough to optimize

- Very generally, optimal weights have the form

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

a result known as the "representer theorem"

Kernel learning

Kernel learning still popular among academics & for certain applications (e.g. life sciences)

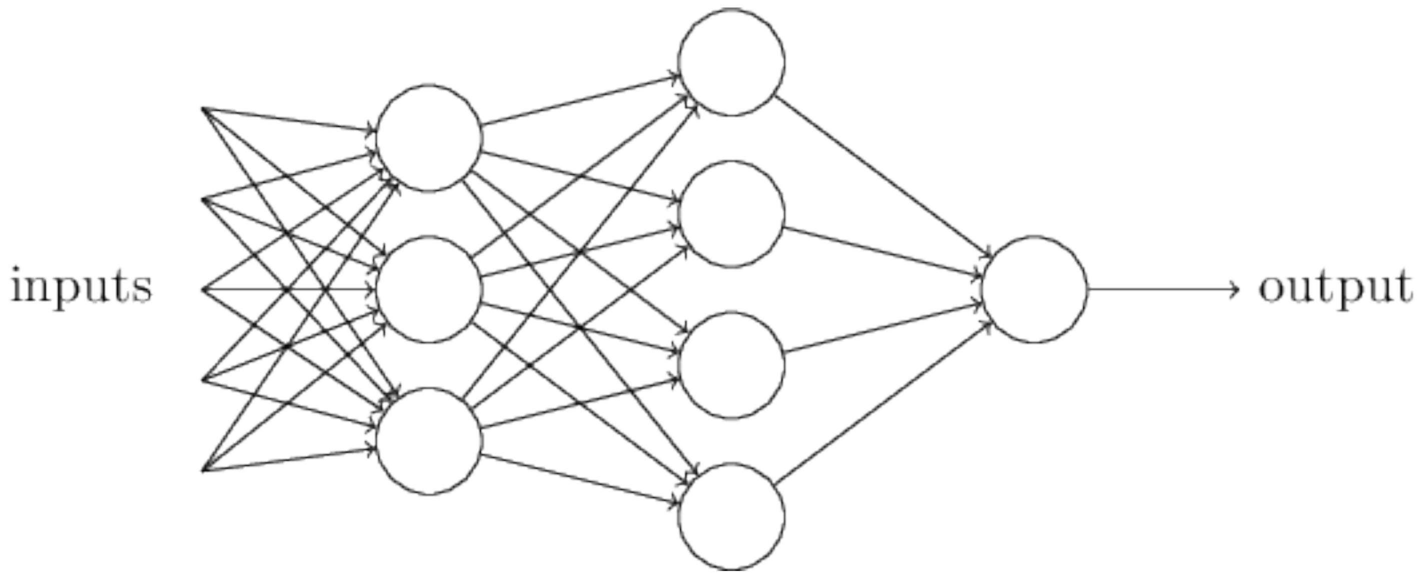
But "kernelization" approach scales as N^3 where N is size of training set – very costly!

Thus kernel methods not popular with engineers

Tomorrow: learning kernel models with tensor network weights

Neural networks

Current favorite of M.L. engineers

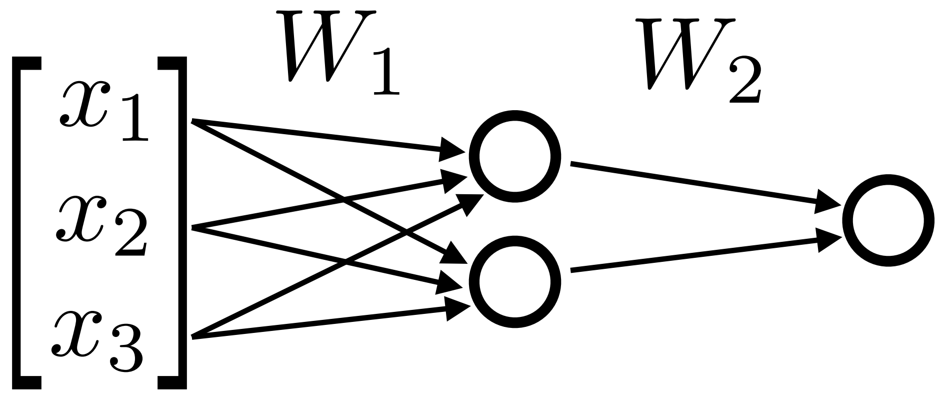


Often notated diagrammatically
(not a tensor diagram!)

Neural networks

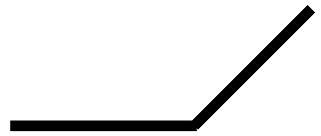
Actually very simple: compute a function $f(\mathbf{x})$ as

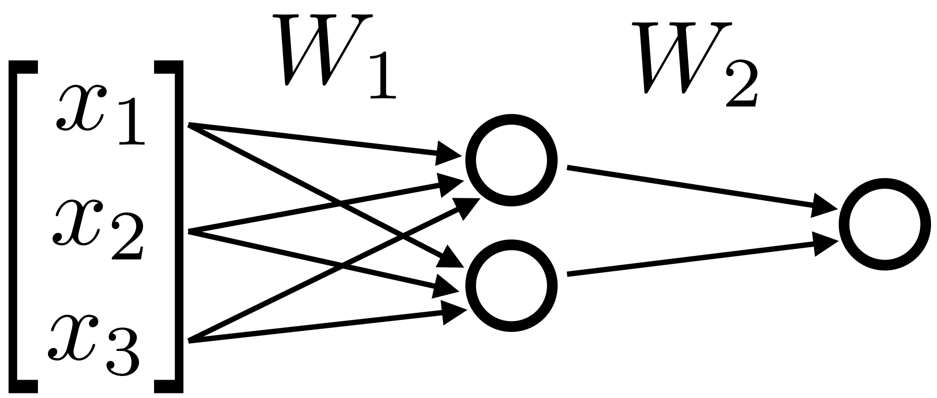
- Multiply input $\mathbf{x}$ by rectangular "weight" matrix W_1
- Point-wise evaluate components of $\mathbf{x}' = W_1\mathbf{x}$ by some non-linear function [e.g. $\sigma(x'_j) = 1/(1 - e^{x'_j - b})$]
- Multiply result by second weight matrix W_2
- Plug new components into non-linearities, etc.



Neural networks

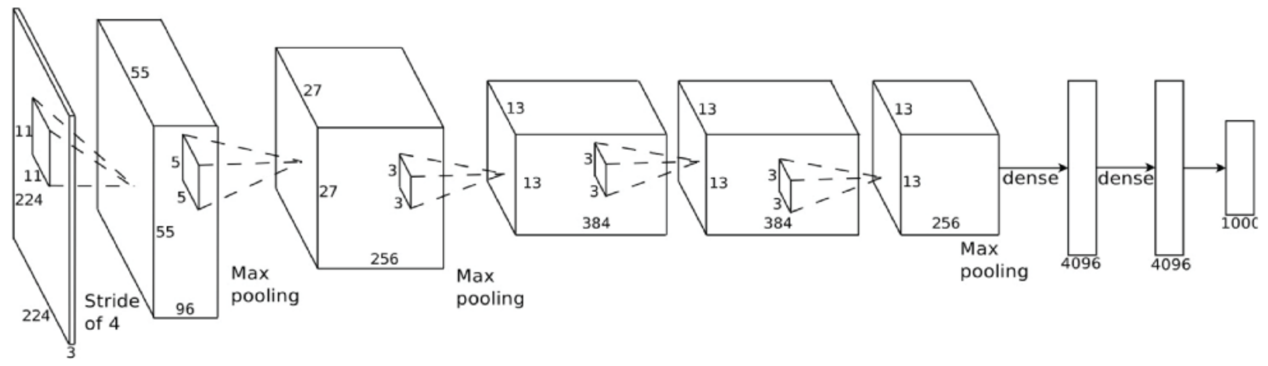
Additional facts:

- Non-linearities $\sigma(x)$ called "neurons"
- Other neurons include tanh and ReLU 
- Neural net with more than one weight matrix is "deep"
- Number of neurons is arbitrary, but with enough can represent any function



Neural networks

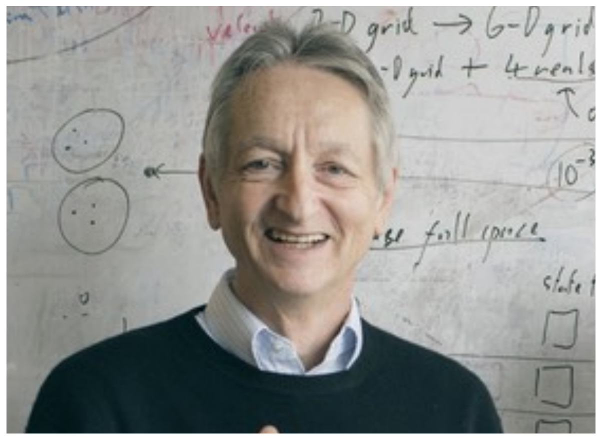
Many successful neural nets include "convolutional layers"
These have sparser weight layers with few parameters.



Recent upsurge of neural nets since 2012 (ImageNet paper)

"Deep learning" often associated with 3 researchers:

Yann LeCun (Facebook)


Geoff Hinton (Vector/Google)

Yoshua Bengio (Montreal)

Other model types

Graphical models

very similar to tensor networks, except

- always interpreted as probability
- non-negative parameters only

Boltzmann machines

identical to random-bond classical Ising (T=1)

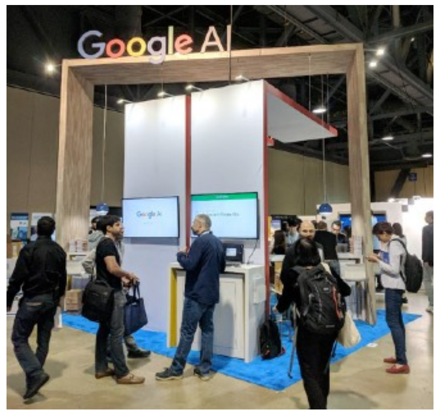
J_{ij} values learnable parameters

generate data by sampling subset of spins

Decision trees

make decisions about input by taking forking paths

Machine Learning Research Culture



One sub-community is academic: papers often involve theorems

Another community is engineering-oriented: papers focus on results, developments are intuitive/faddish

Conference talks/posters valued above journal articles

Strong industry ties: Google, Microsoft, etc. have booths at conferences, grad students poached often

Recommended Resources

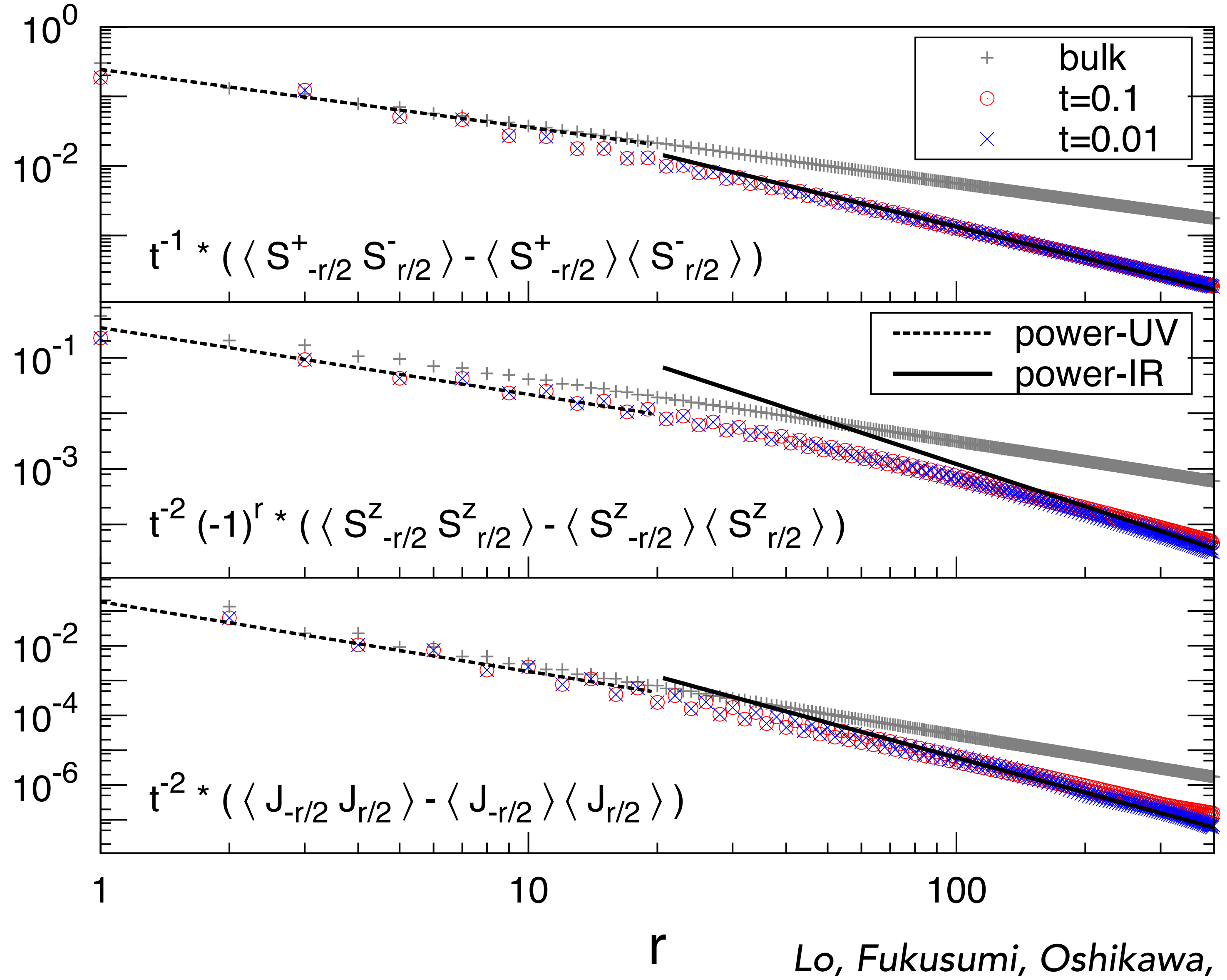
- Online book by Michael Nielsen (quant. computing author)
  http://neuralnetworksanddeeplearning.com
- Caltech Lectures by Yaser Abu-Mostafa CS 156
  Available on YouTube. Companion book "Learning from Data"
- M.L. review article by Pankaj Mehta, David Schwab
  aimed at physicists
- TensorFlow examples (MNIST demo)
- Blogs of Chris Olah and Andrej Karpathy

Tensor Network Machine Learning

Tensor Network Machine Learning

Stoudenmire, Schwab, *Advanced in Neural Information Processing Systems (NIPS)*, **29**, 4799 [arxiv:1605.05775]

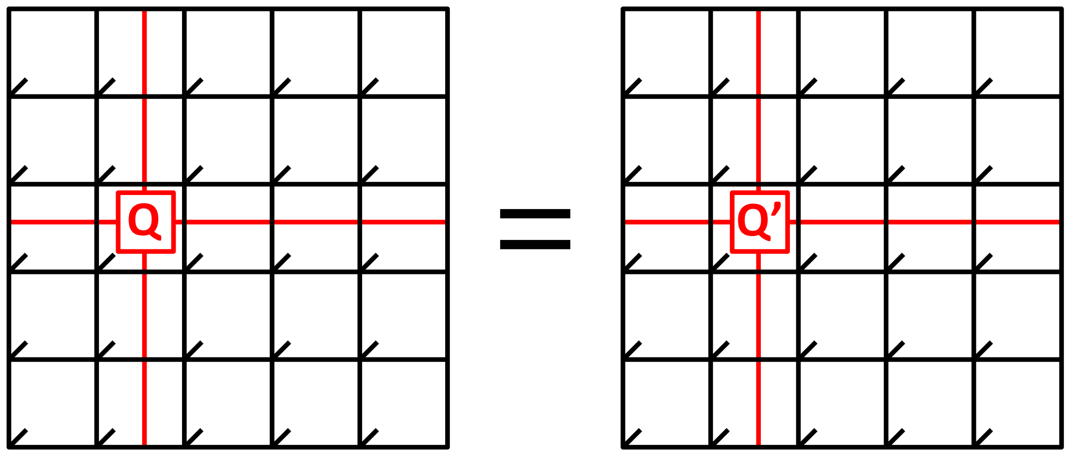
Tensor network methods admit powerful optimization techniques, giving high precision results



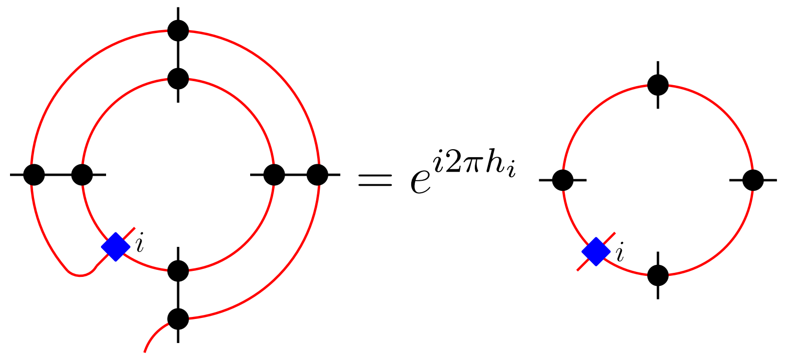
Lo, Fukusumi, Oshikawa, Kao, Chen, arxiv:1805.05006

Long-distance properties due to impurities in Luttinger liquids

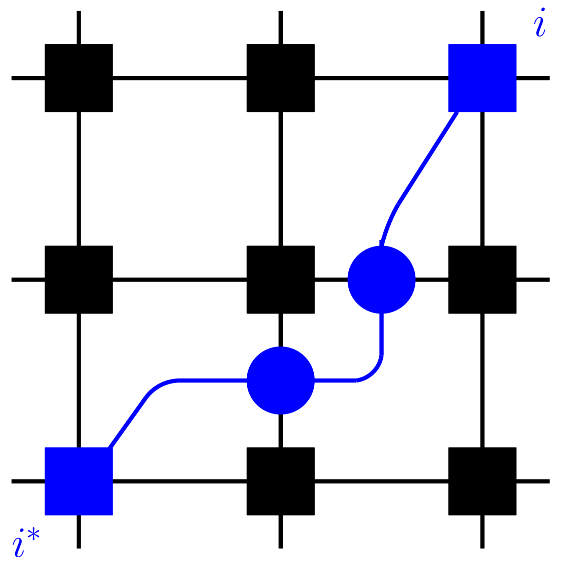
Tensor networks are highly interpretable, due to linear structure



Ground state degeneracy



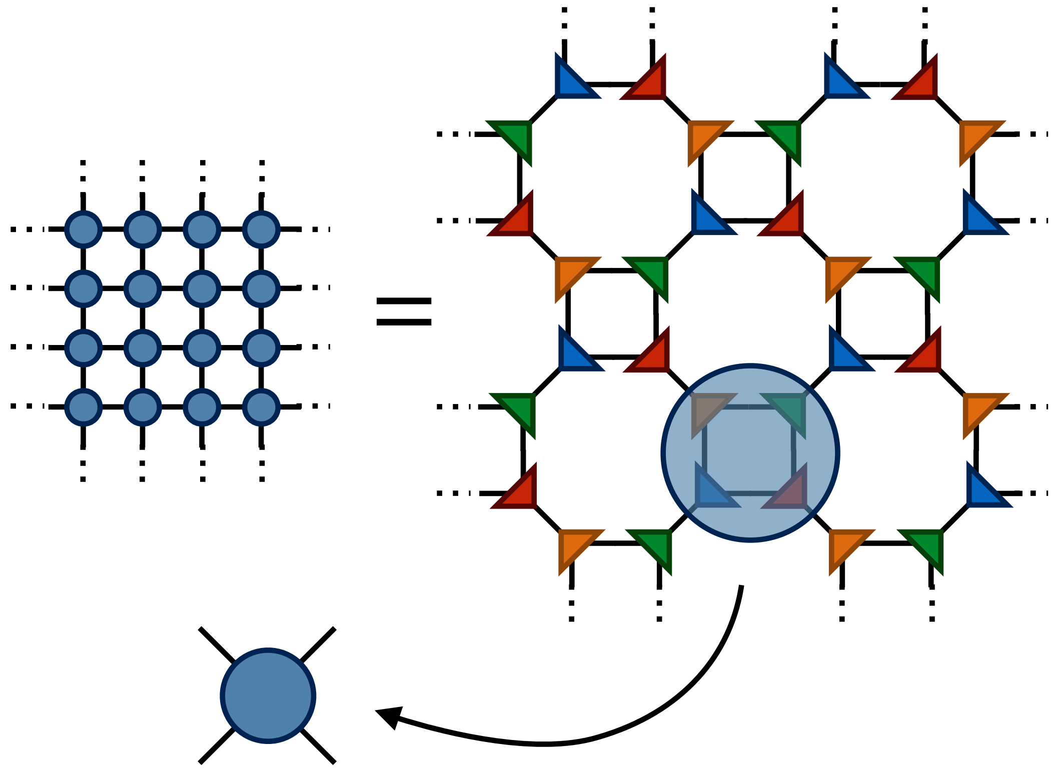
Topological spin of anyons

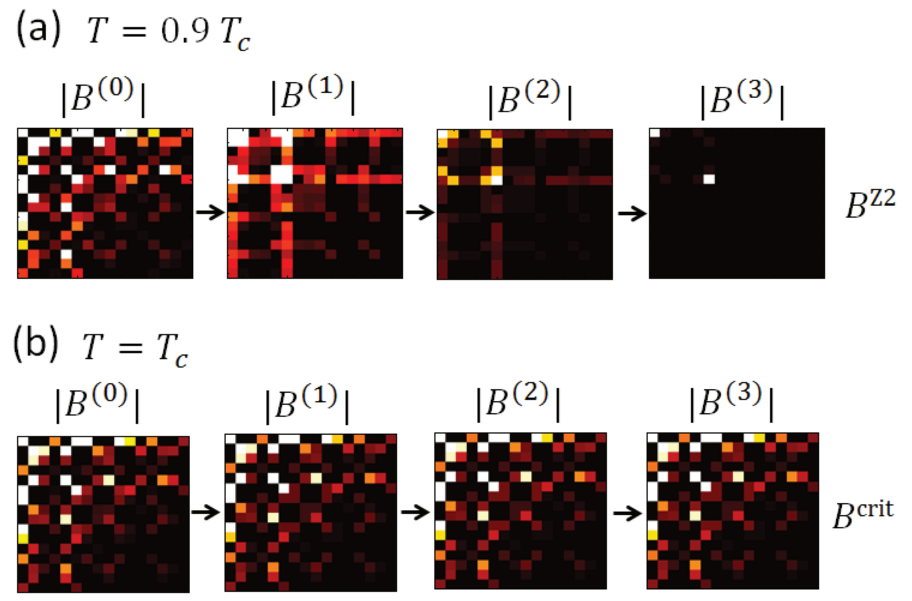


MPO "pulling through" condition

Şahinoğlu et al., arxiv:1409.2150
Williamson et al., arxiv:1412.5604
Bultinck et al., arxiv:1511.08090

Applicable to classical systems too – tensor RG family of methods





	exact	TRG(64)	TRG+env(64)	TEFR(64)	TNR(24)
c	0.5	0.49982	0.49988	0.49942	0.50001
σ	0.125	0.12498	0.12498	0.12504	0.1250004
ϵ	1	1.00055	1.00040	0.99996	1.00009
	1.125	1.12615	1.12659	1.12256	1.12492
	1.125	1.12635	1.12659	1.12403	1.12510
	2	2.00243	2.00549	-	1.99922
	2	2.00579	2.00557	-	1.99986
	2	2.00750	2.00566	-	2.00006
	2	2.01061	2.00567	-	2.00168

*Levin, Nave, PRL **99**, 120601 (2007)*

*Evenbly, Vidal, PRL **115**, 200401 (2015)*

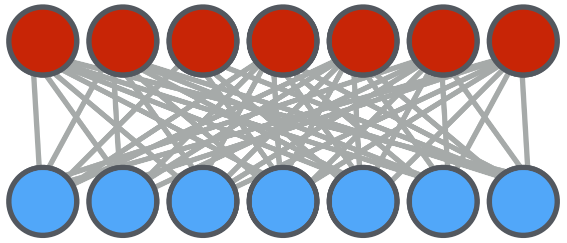
🤔

Wavefunction, transfer matrix just large tensors

Tensor network just a math technique

Useful for more than physics?

Machine learning has many connections to physics

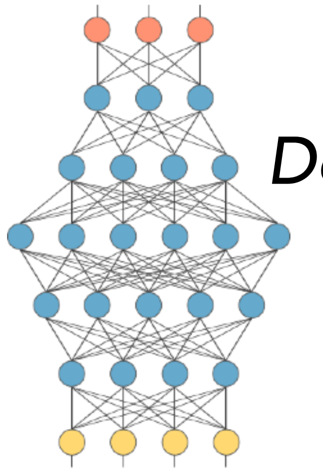


Boltzmann Machines

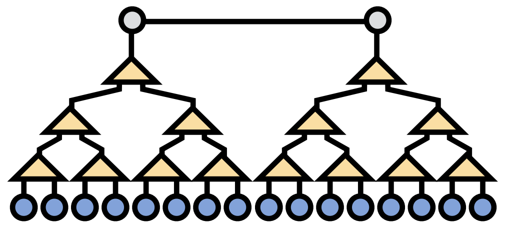
↓ ↑ ↓ ↓ ↑ ↑ ↓ ↑

1920s

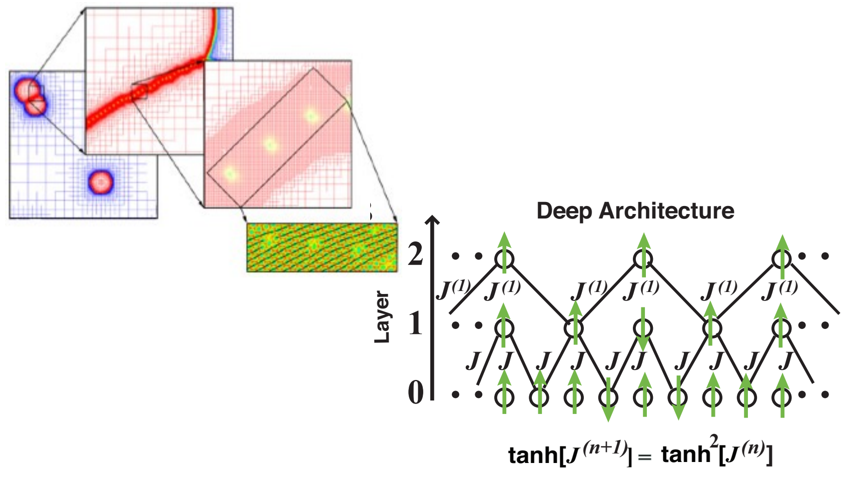
Disordered Ising Model



Deep Networks



Heirarchical PCA Methods



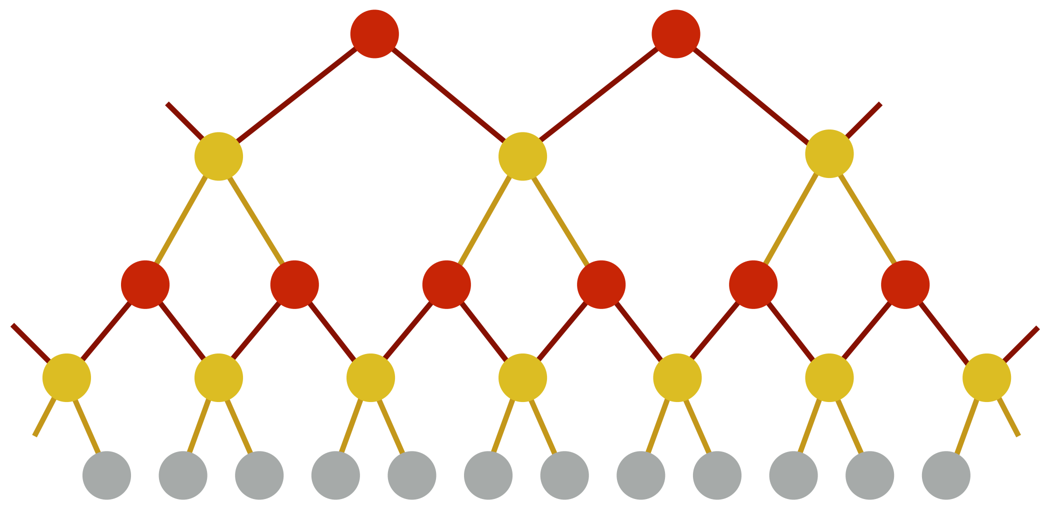
1970s

The "Renormalization Group"

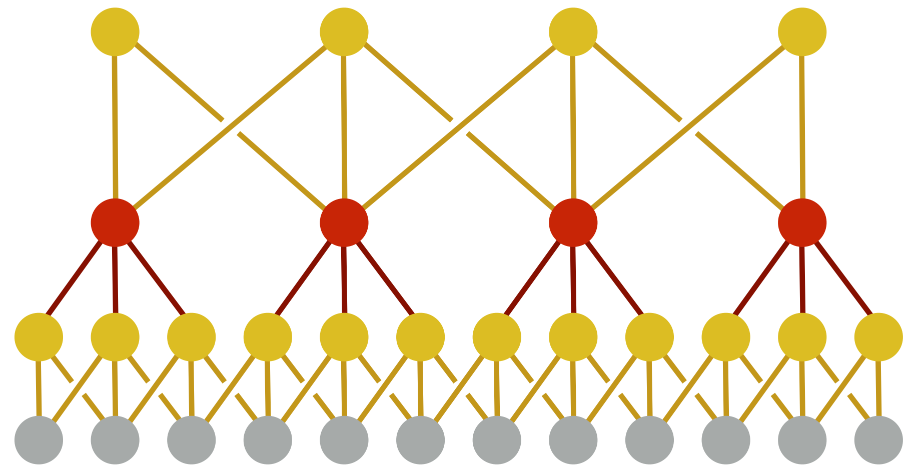
P. Mehta and D.J. Schwab, arxiv:1410.3831
S. Bradde and W. Bialek, arxiv:1610.09733
E.M. Stoudenmire, arxiv:1801.00315

More recent ideas from physics useful for machine learning?

"MERA" tensor network

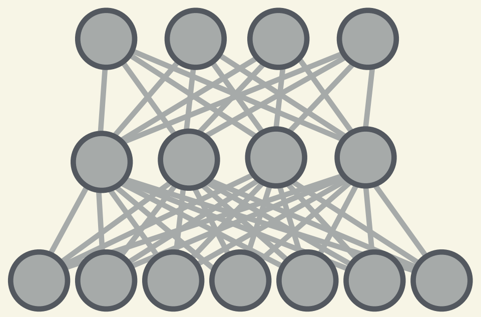


Convolutional neural network



Analogy between wavefunctions & M.L. models

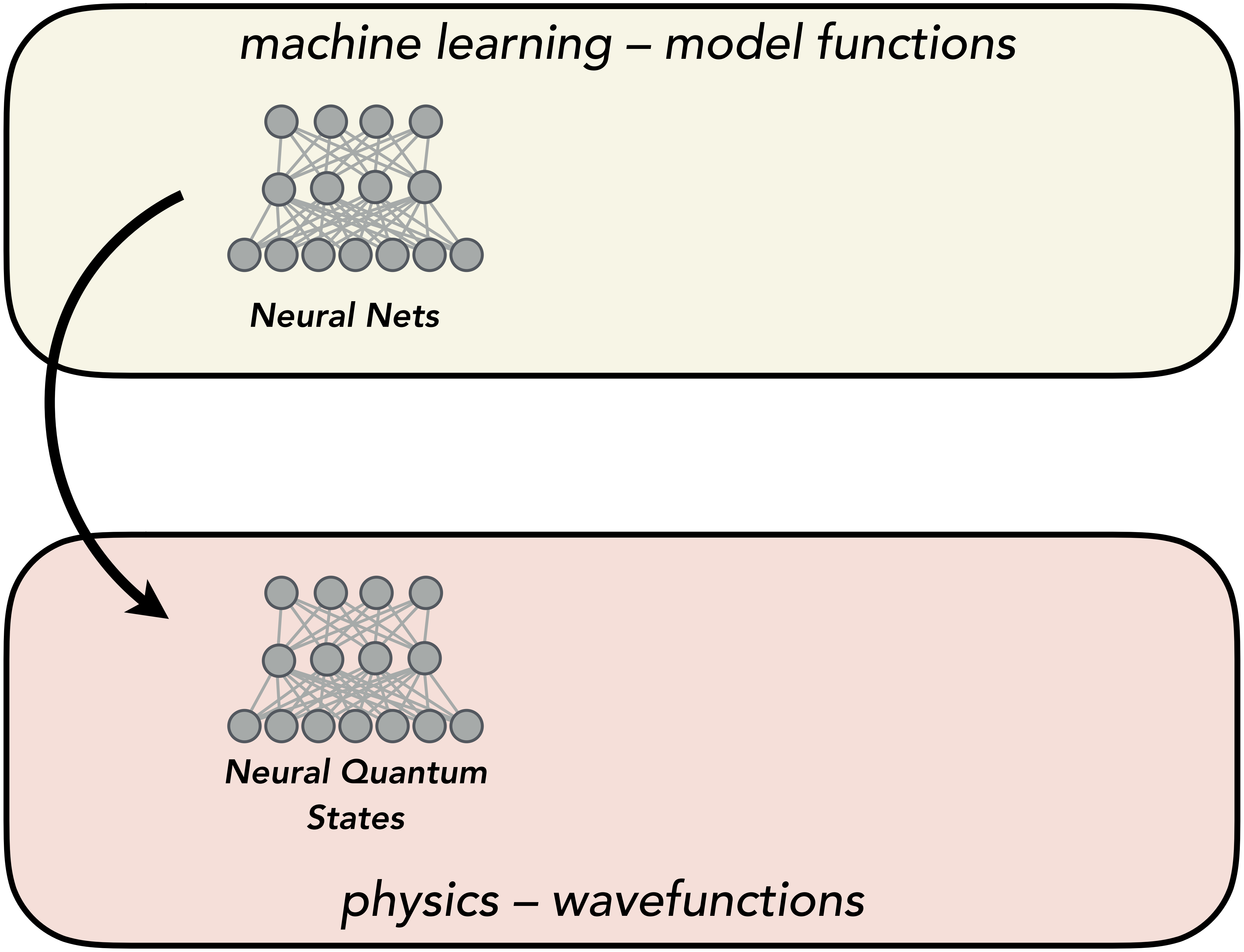
machine learning – model functions



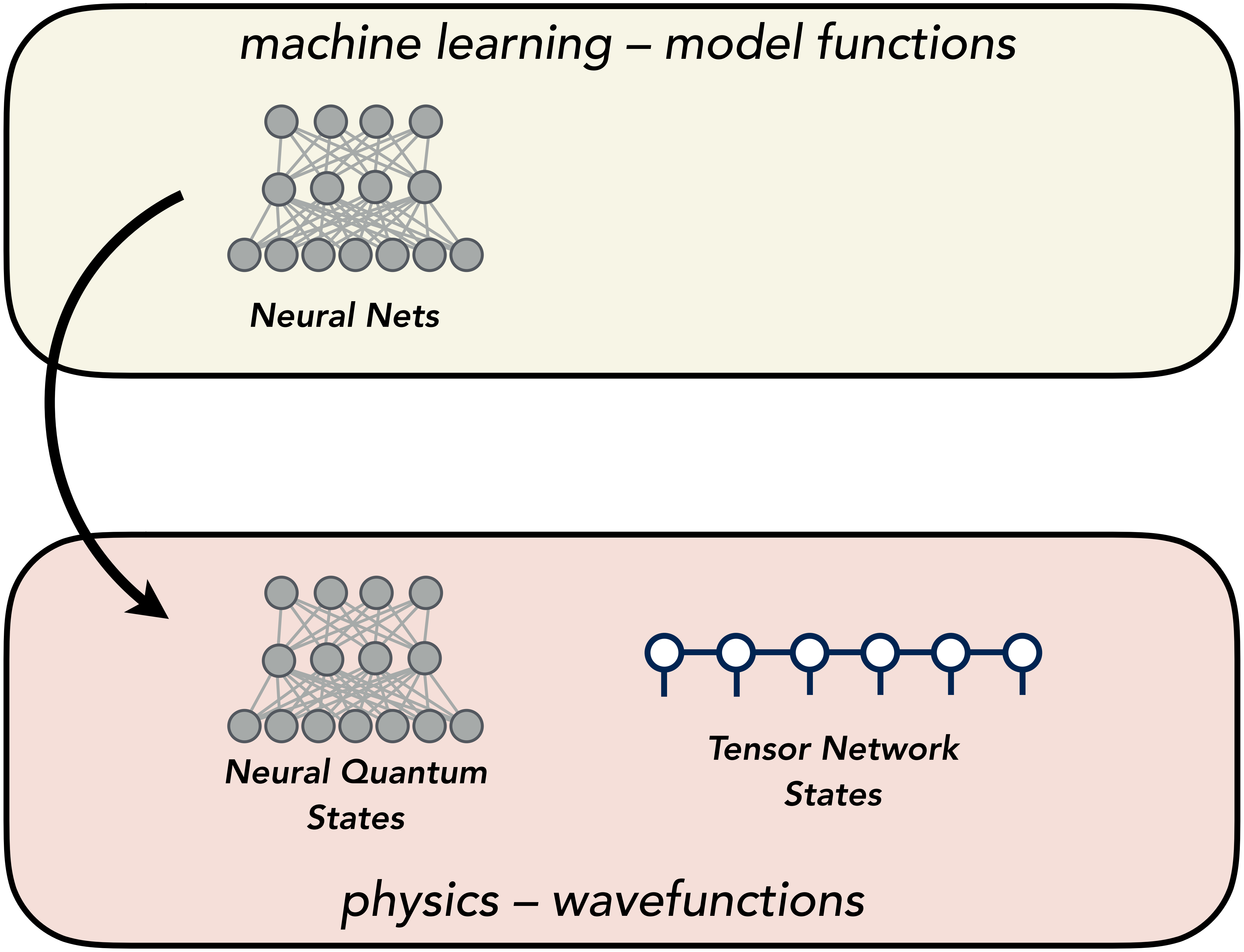
Neural Nets

physics – wavefunctions

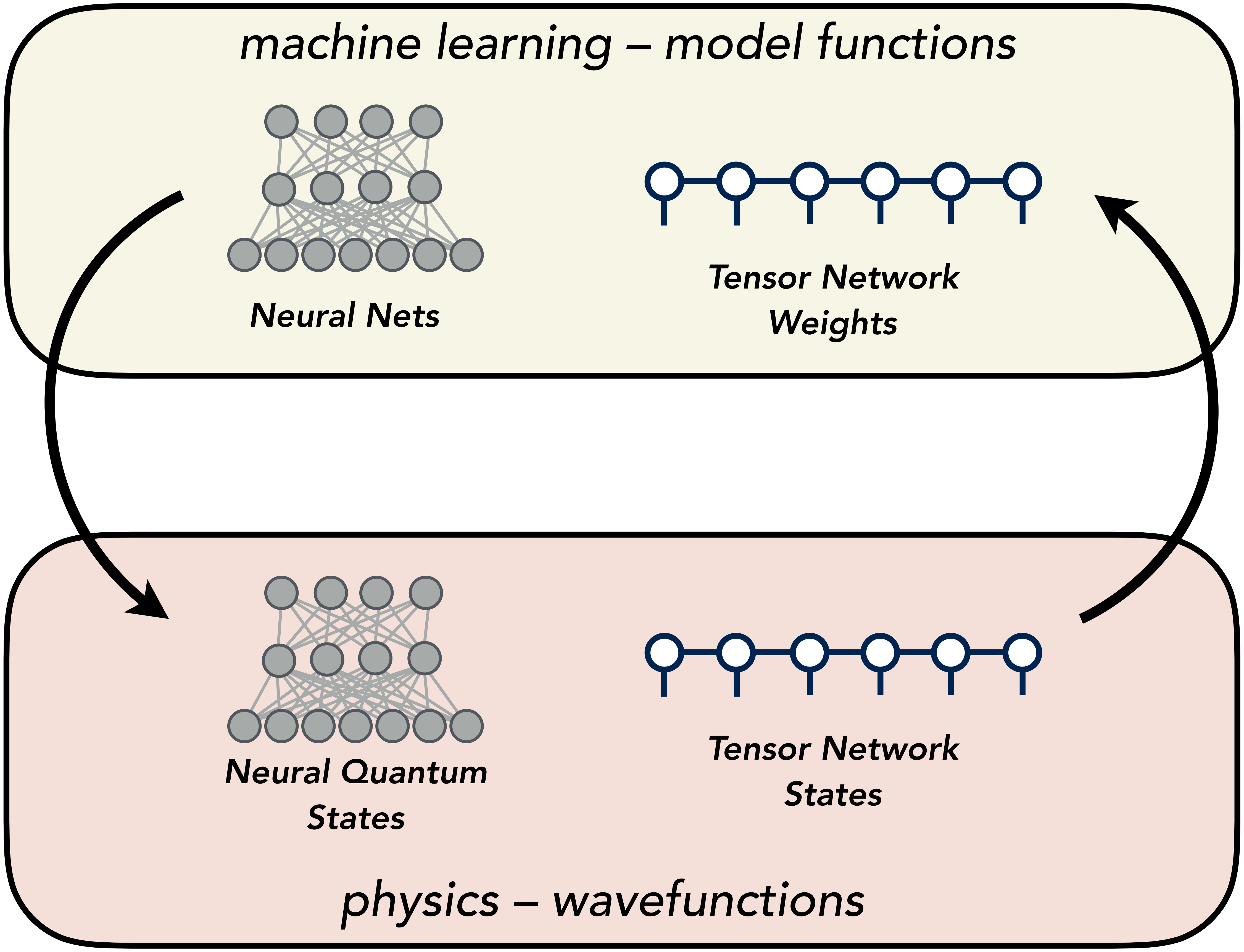
Analogy between wavefunctions & M.L. models



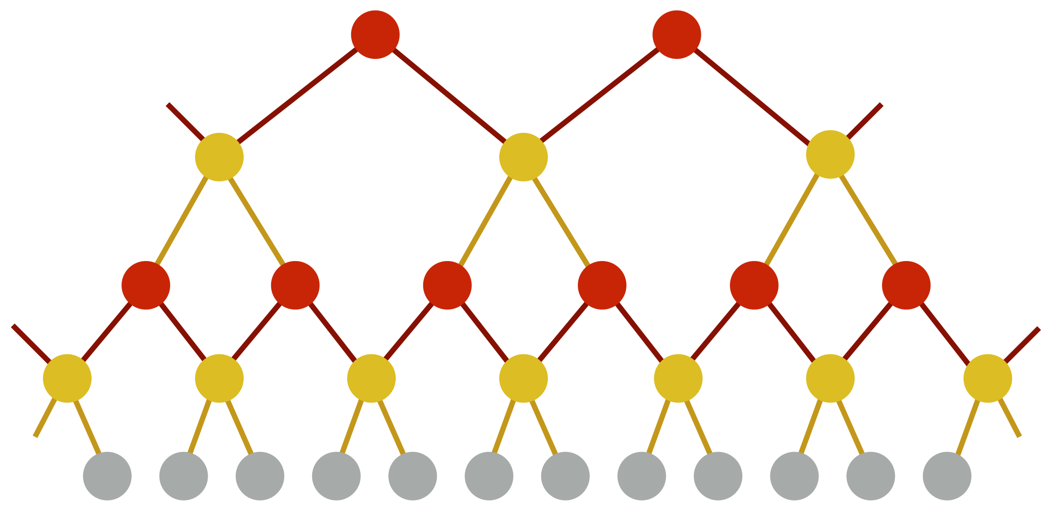
Analogy between wavefunctions & M.L. models


machine learning – model functions
Neural Nets
Neural Quantum States
Tensor Network States
physics – wavefunctions

Analogy between wavefunctions & M.L. models



Are tensor networks useful for machine learning?


"MERA" tensor network

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

Realized benefits:

- Linear scaling
- Adaptive weights
- Learning data "features"

Future benefits?

- Interpretability / theory
- Better algorithms
- Quantum computing

Many proposals already to use tensor networks for machine learning

Compressing weights of neural nets (& other models)

Yu et al., Advances in Neural Information Processing (2017), arxiv:1711.00073

Izmailov et al., arxiv:1710.07324 (2017)

Yang et al., arxiv:1707.01786 (2017)

Garipov et al., arxiv:1611.03214 (2016)

Novikov et al., Advances in Neural Information Processing (2015) (arxiv:1509.06569)

Large scale PCA

Lee, Cichocki, arxiv: 1410.6895 (2014)

Gaussian Processes

Izmailov, Novikov, Kropotov, arxiv:1710.07324 (2017)

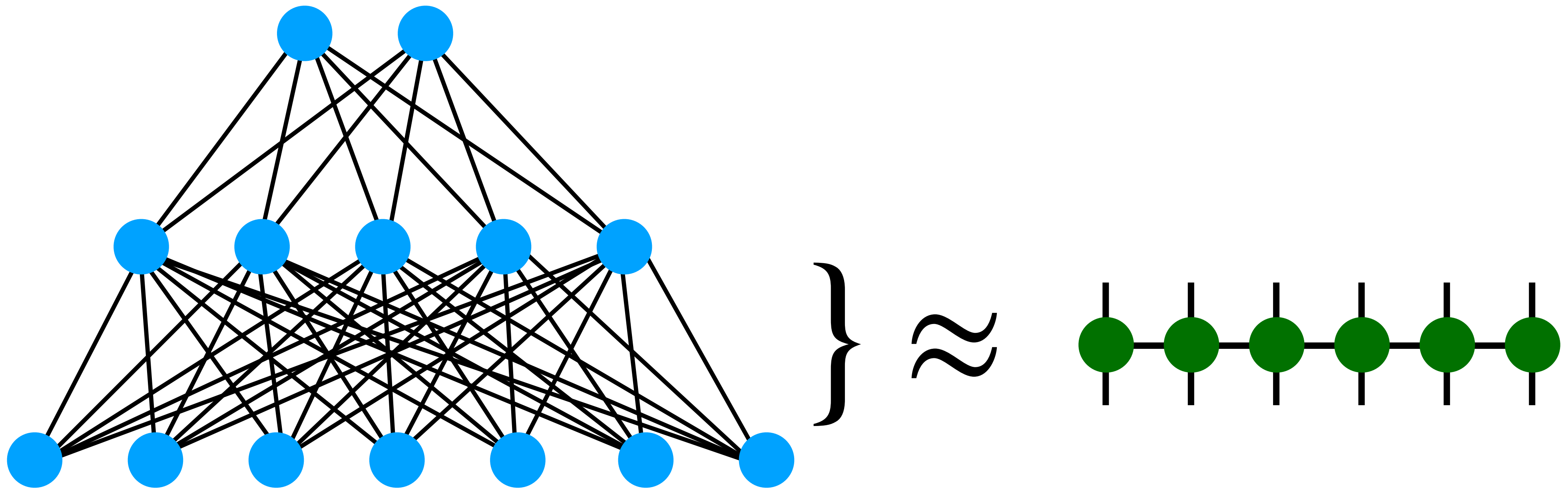
Feature extraction & tensor completion

Bengua et al., arxiv:1606.01500, arxiv:1607.03967, arxiv:1609.04541 (2016)

Phien et al., arxiv:1601.01083 (2016)

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

Example: compressing neural network weight layers

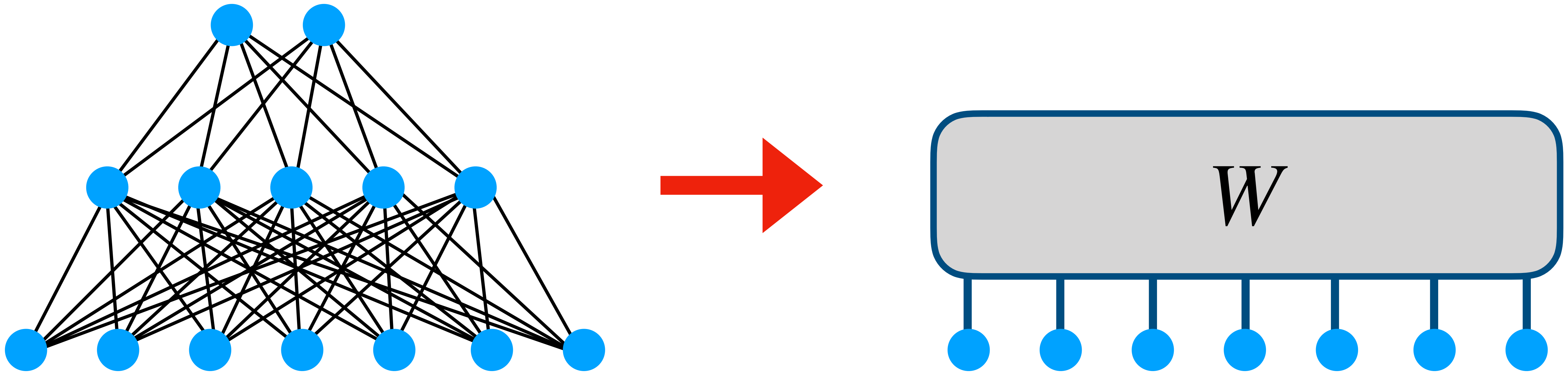


Novikov et al., Advances in Neural Information Processing (2015) (arxiv:1509.06569)

Garipov, Podoprikhin, Novikov, arxiv:1611.03214

- Train very "wide" model: 262,144 hidden units
- Achieve 80x compression, only 1% accuracy loss

Framework where tensor network plays central role?



Motivation:

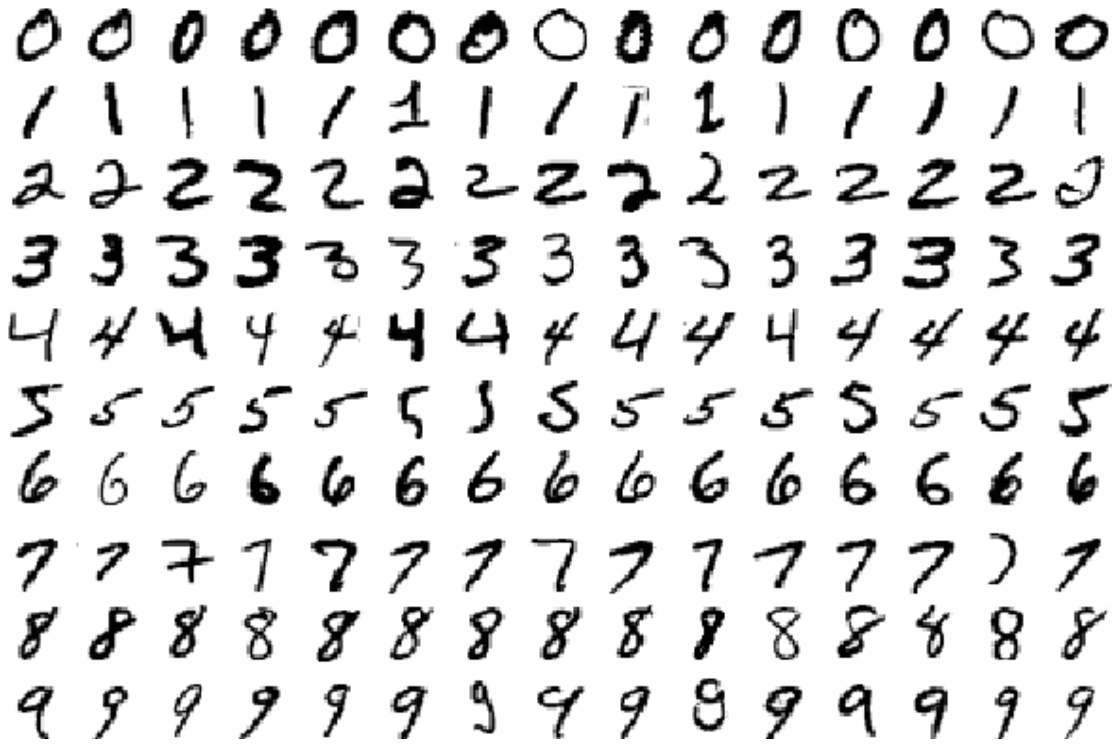
- Can natural images be more complex than wavefunctions?
- Import many ideas, algorithms from physics
- Improve tensor network methods

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

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

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

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, plus other "feature maps"

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

$$
\begin{aligned}
f(\mathbf{x}) &= W \cdot \Phi(\mathbf{x}) \\
&= \sum_{\mathbf{s}} W_{s_1 s_2 s_3 \cdots s_N} \phi^{s_1}(x_1)\phi^{s_2}(x_2)\phi^{s_3}(x_3) \cdots \phi^{s_N}(x_N)
\end{aligned}
$$

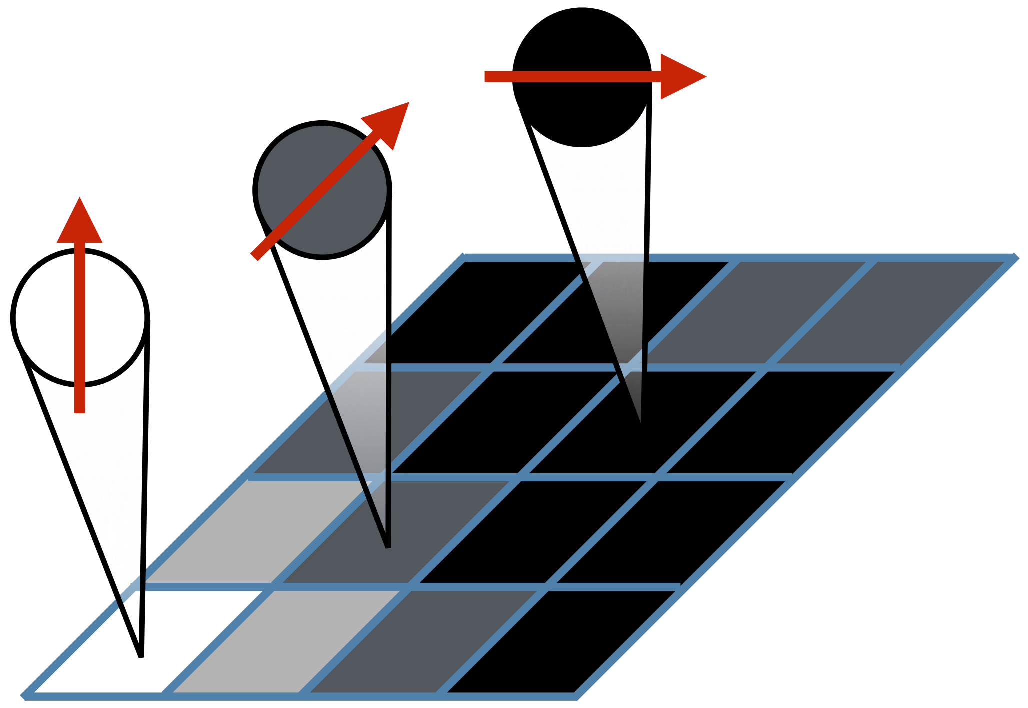
Highly expressive!

$\mathbf{x} =$ input

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"



$\mathbf{x} =$ input

$\phi =$ local feature map

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

$$\Phi^{s_1 s_2 \cdots s_N}(\mathbf{x}) = \phi^{s_1}(x_1) \otimes \phi^{s_2}(x_2) \otimes \cdots \otimes \phi^{s_N}(x_N)$$

- Tensor product of local feature maps / vectors
- Just like product state wavefunction of spins
- Vector in 2^N dimensional space

$\mathbf{x} =$ input

$\phi =$ local feature map

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

More detailed notation

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

raw inputs

$$\Downarrow$$

$$\Phi(\mathbf{x}) = \begin{bmatrix} \phi_1(x_1) \\ \phi_2(x_1) \end{bmatrix} \otimes \begin{bmatrix} \phi_1(x_2) \\ \phi_2(x_2) \end{bmatrix} \otimes \begin{bmatrix} \phi_1(x_3) \\ \phi_2(x_3) \end{bmatrix} \otimes \cdots \otimes \begin{bmatrix} \phi_1(x_N) \\ \phi_2(x_N) \end{bmatrix}$$

feature vector

$\mathbf{x} =$ input

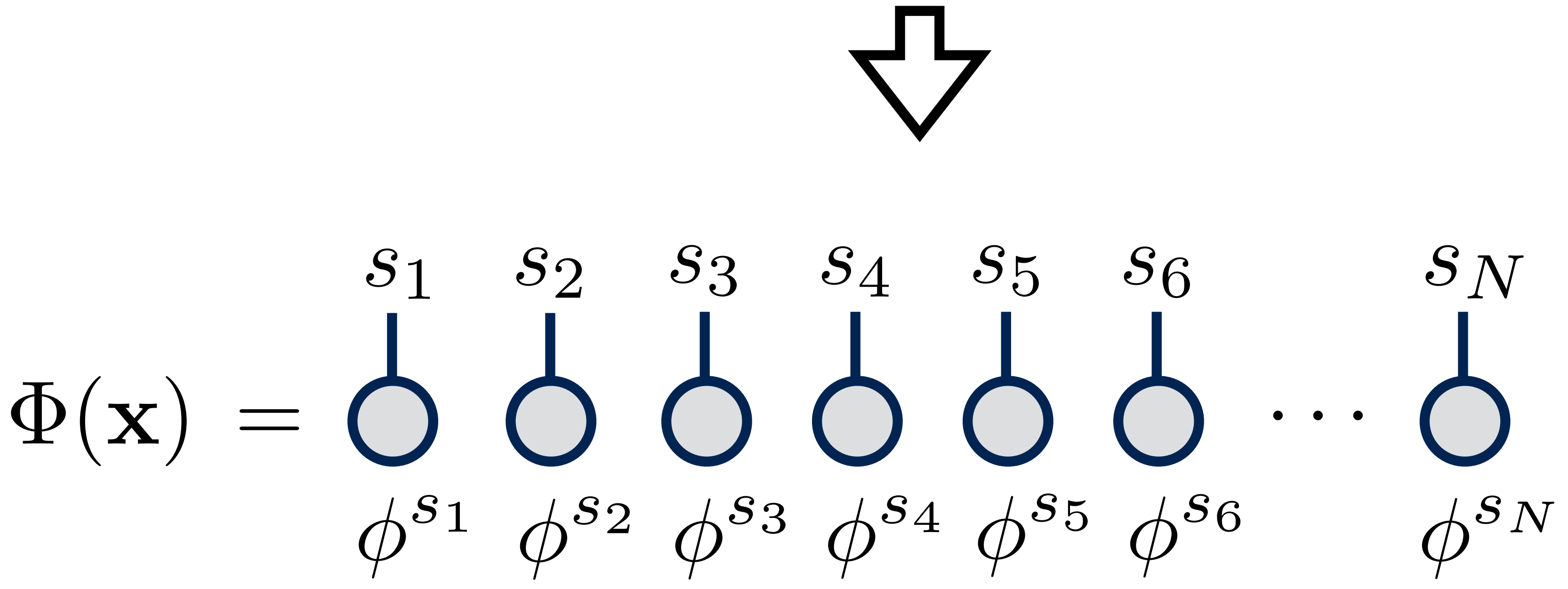
$\phi =$ local feature map

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

Tensor diagram notation

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

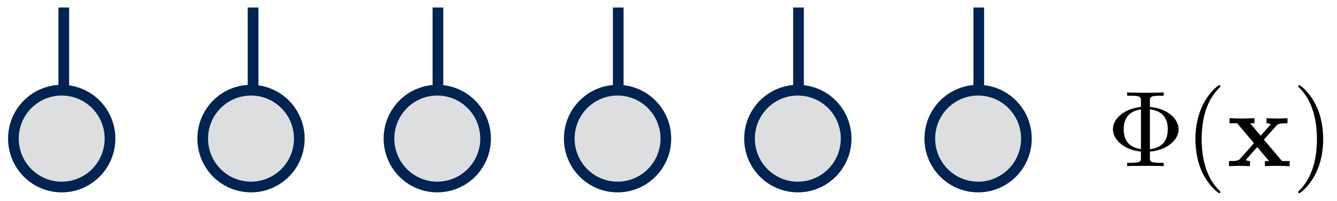
raw inputs



feature vector

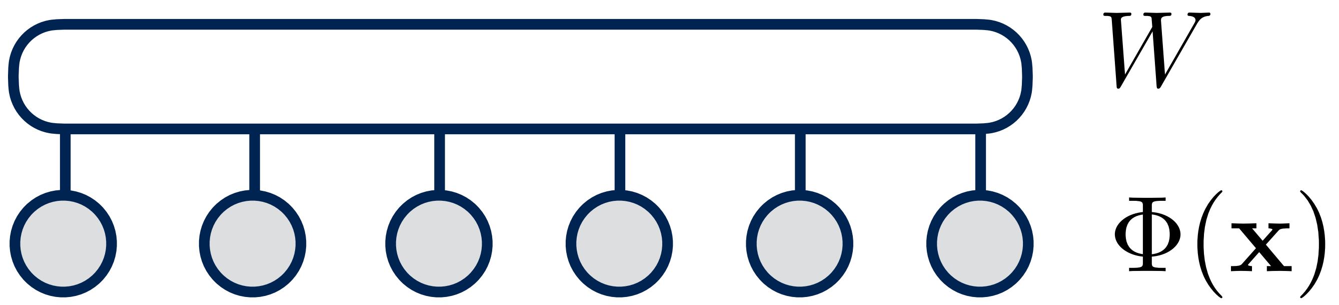
Construct decision function

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



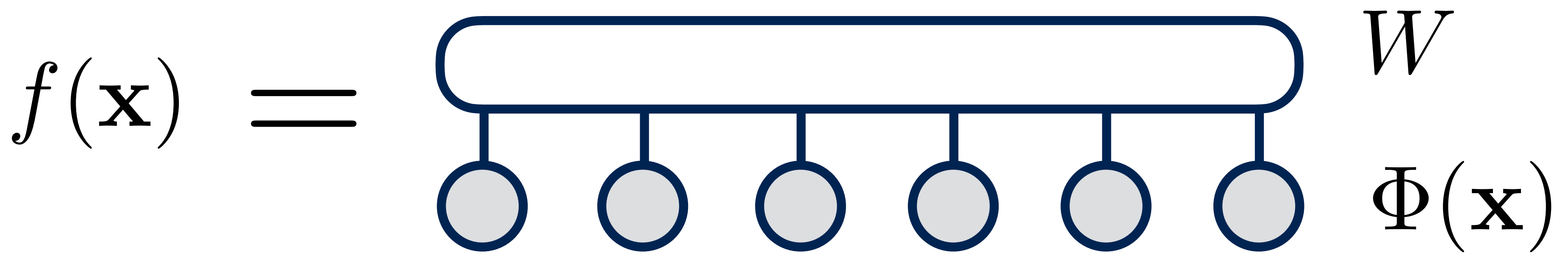
Construct decision function

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



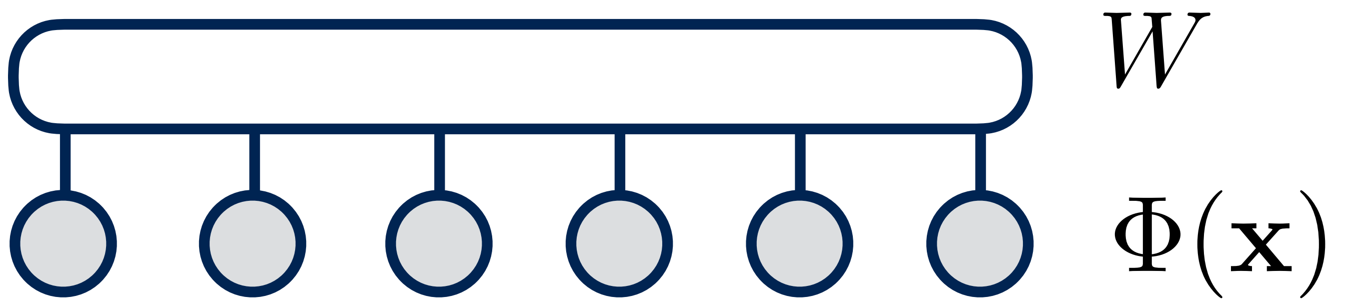
Construct decision function

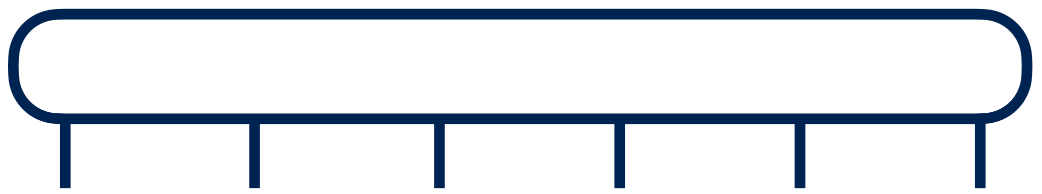
$$\boxed{f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})}$$



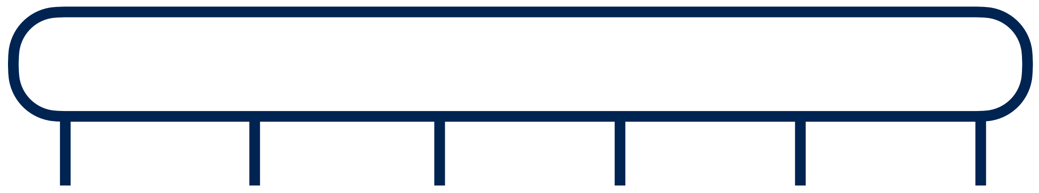
Construct decision function

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

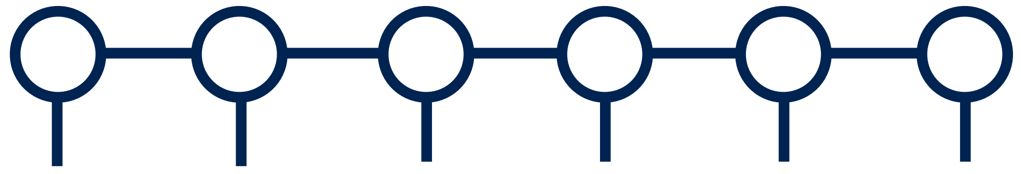
$f(\mathbf{x}) =$ 

$W =$ 

Main approximation

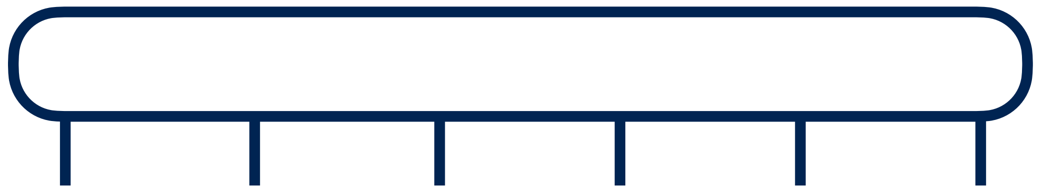
$W =$ 

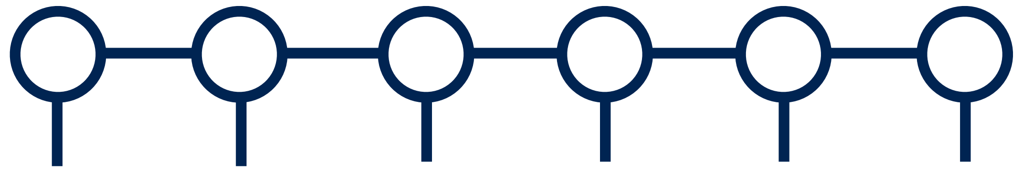
order-N tensor

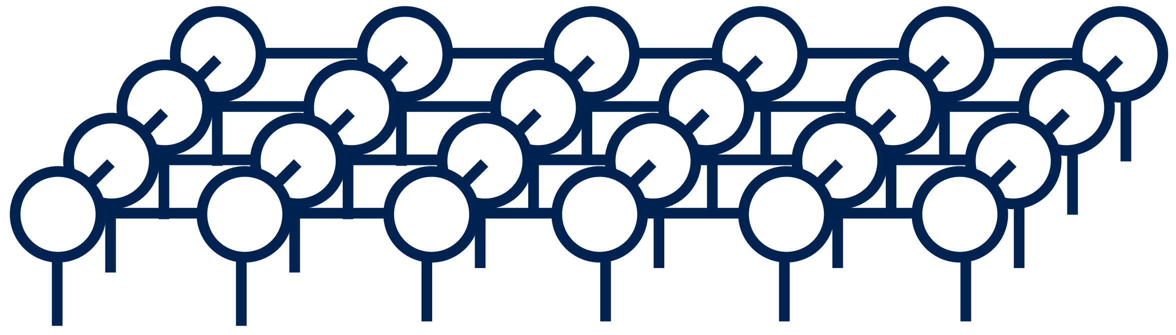
$\approx$ 

matrix
product
state (MPS)

Main approximation

$W =$  *order-N tensor*

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

$\Big(\approx$  *PEPS* $\Big)$

Linear scaling

Can use algorithm similar to DMRG to optimize

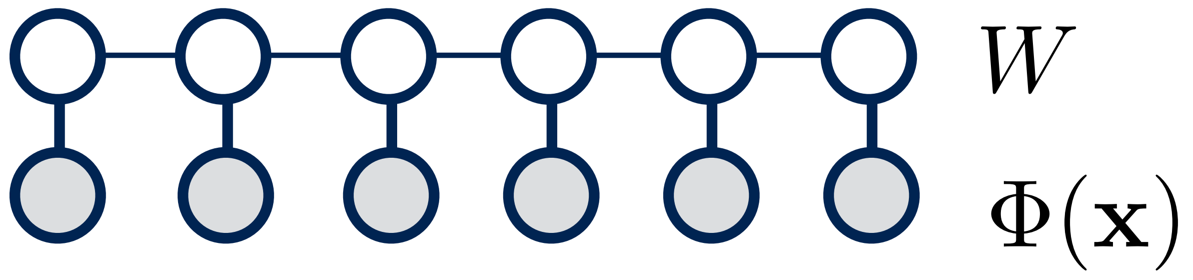
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$



Linear scaling

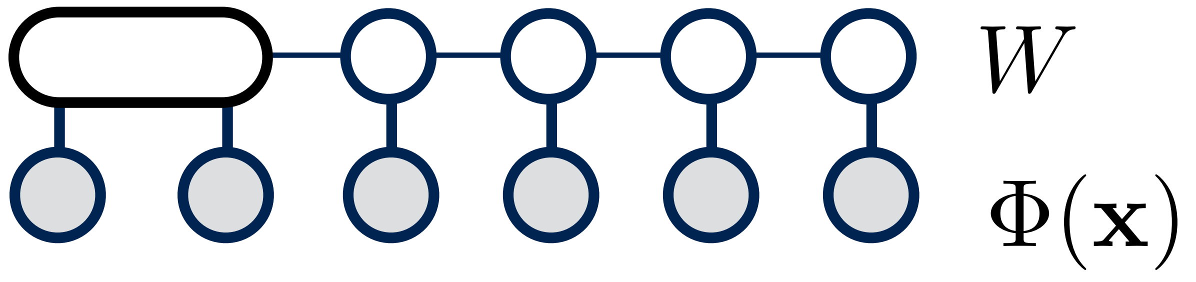
Can use algorithm similar to DMRG to optimize

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

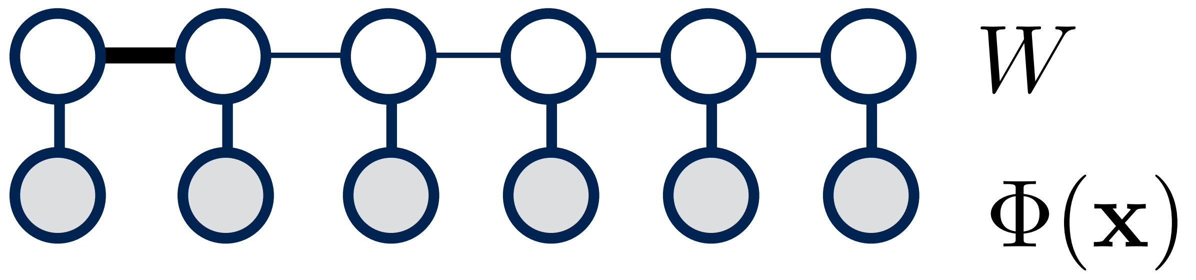
Can use algorithm similar to DMRG to optimize

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

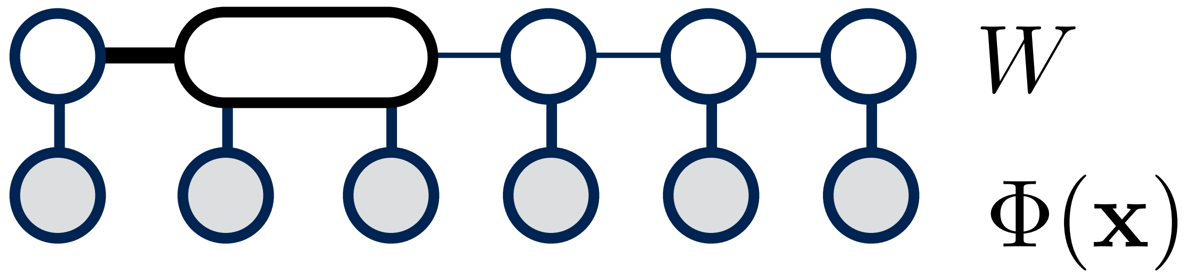
Can use algorithm similar to DMRG to optimize

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

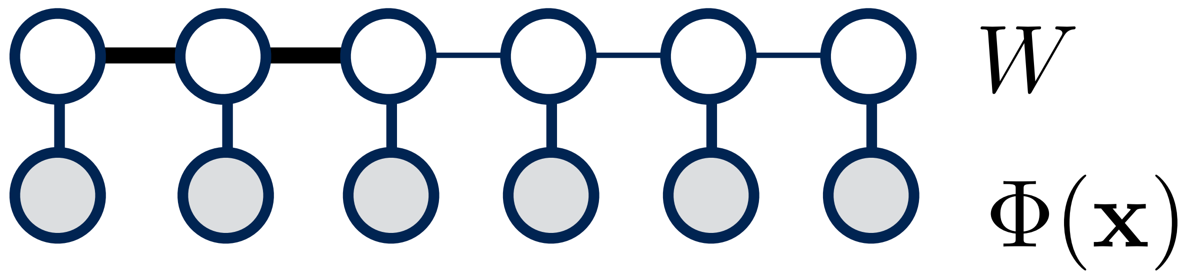
Can use algorithm similar to DMRG to optimize

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

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

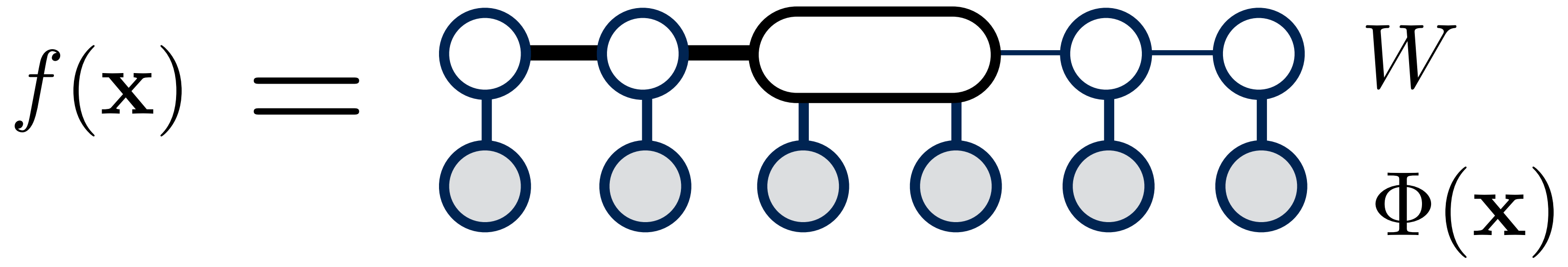
$f(\mathbf{x}) =$ 

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



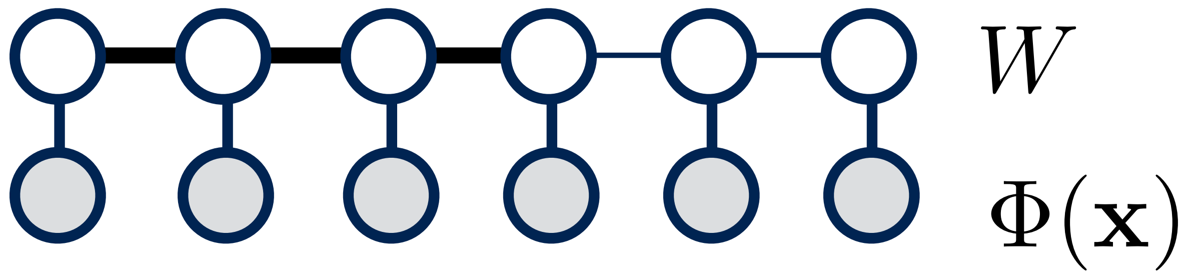
Linear scaling

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

$f(\mathbf{x}) =$



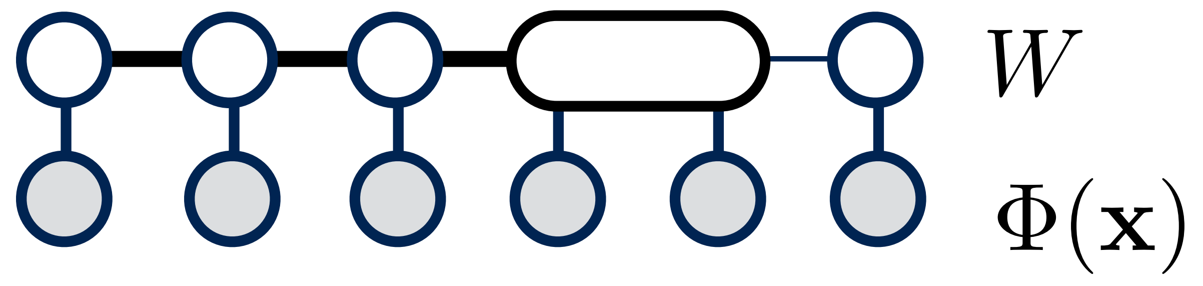
Linear scaling

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

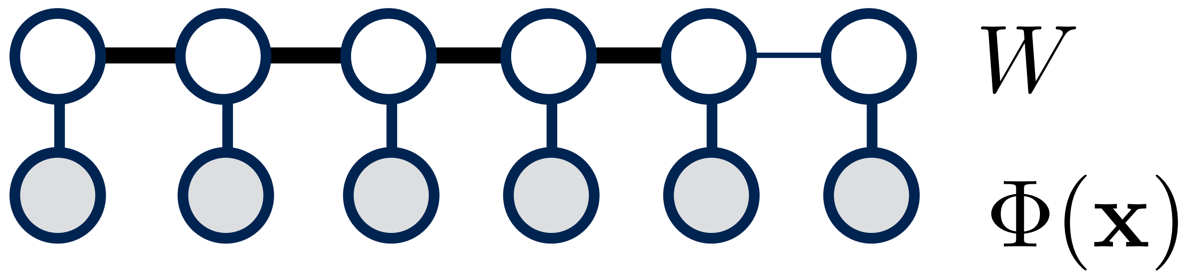
Can use algorithm similar to DMRG to optimize

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

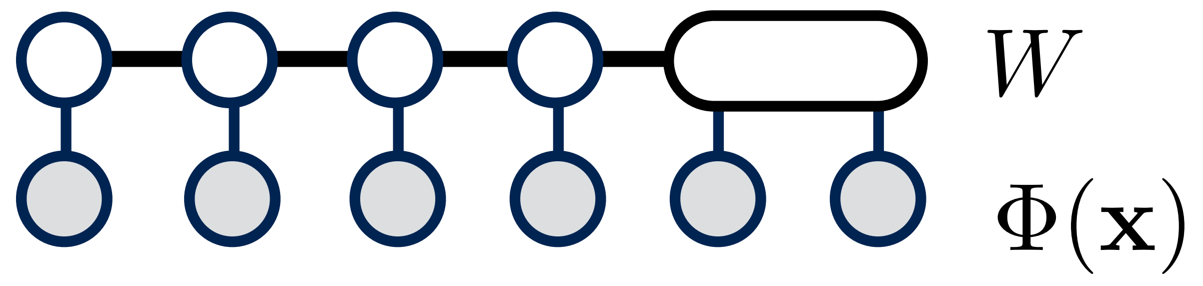
Can use algorithm similar to DMRG to optimize

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

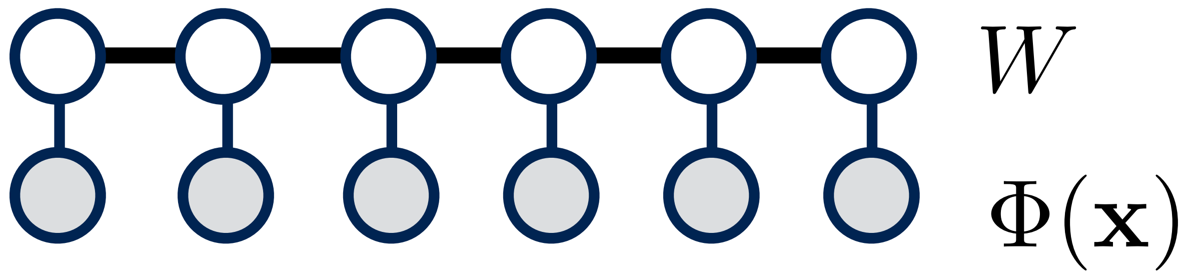
Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

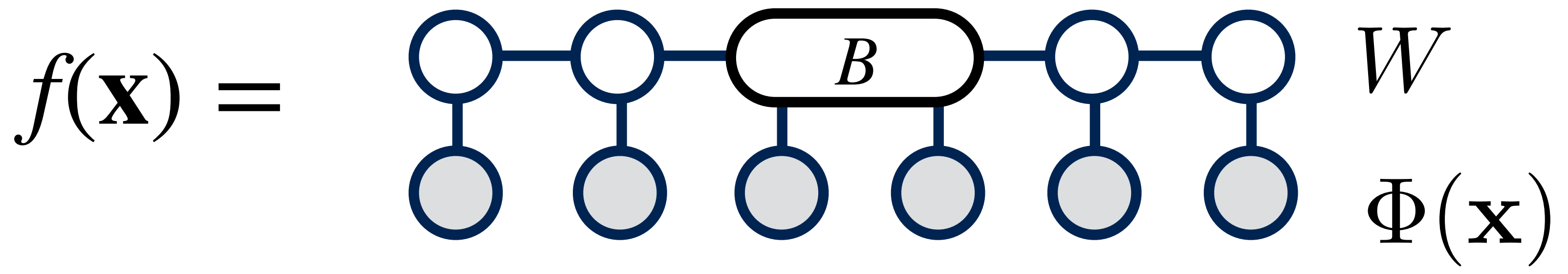
$m =$ MPS bond dimension

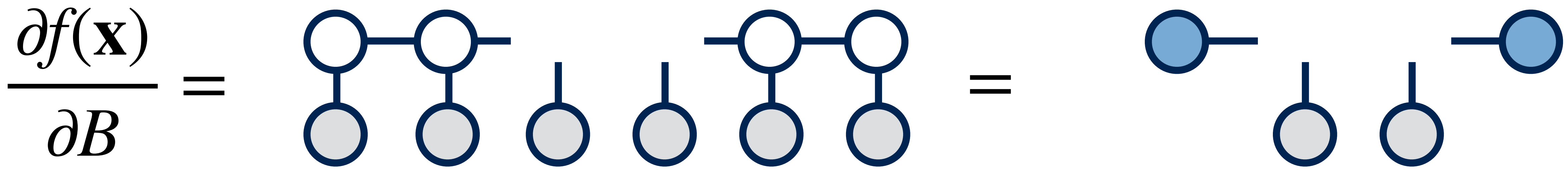
$f(\mathbf{x}) =$

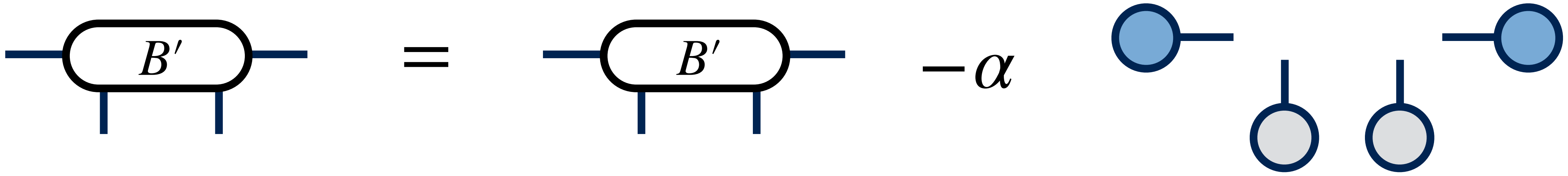


Gradient step:

At each bond, update "bond tensor" by computing and applying the gradient







Why should this work at all?

Linear classifier $f(\mathbf{x}) = V \cdot \mathbf{x}$ exactly m=2 MPS

$$W = \begin{bmatrix} V_0 & 1 \end{bmatrix} \begin{bmatrix} \hat{1} & 0 \\ \hat{V}_1 & \hat{1} \end{bmatrix} \begin{bmatrix} \hat{1} & 0 \\ \hat{V}_2 & \hat{1} \end{bmatrix} \begin{bmatrix} \hat{1} & 0 \\ \hat{V}_3 & \hat{1} \end{bmatrix} \cdots$$

$$\hat{1} = [1 \;\; 0]$$

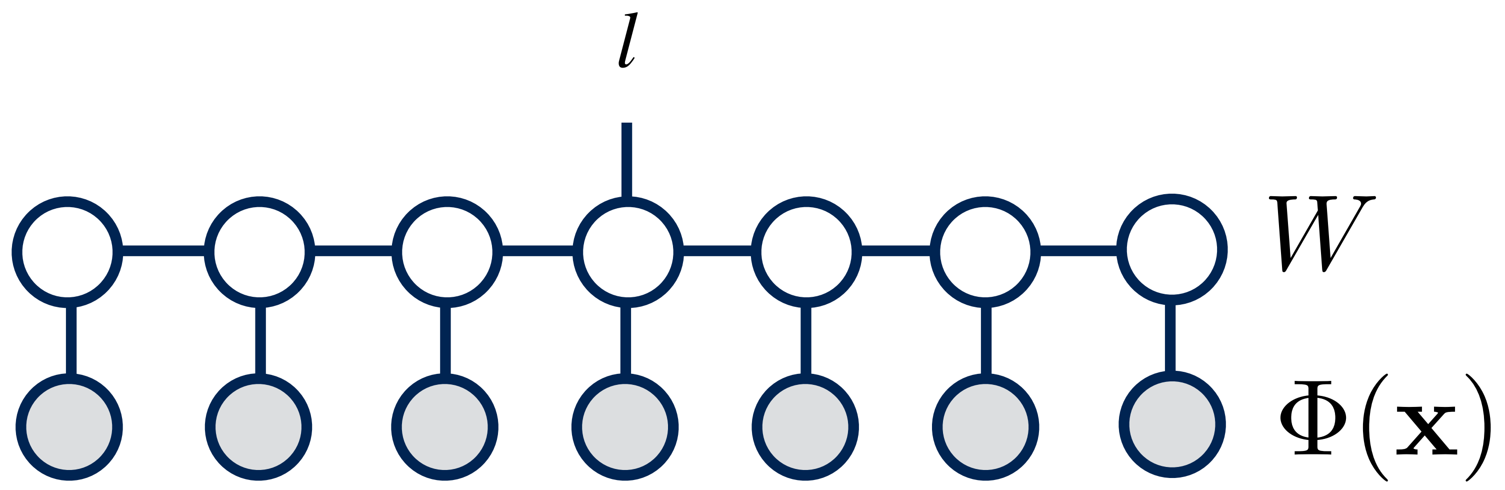
$$\hat{V}_j = [0 \;\; V_j]$$

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

$$\phi^{s_j}(x_j) = [1\,,\; x_j]$$

Novikov, Trofimov, Oseledets, arxiv:1605.03795

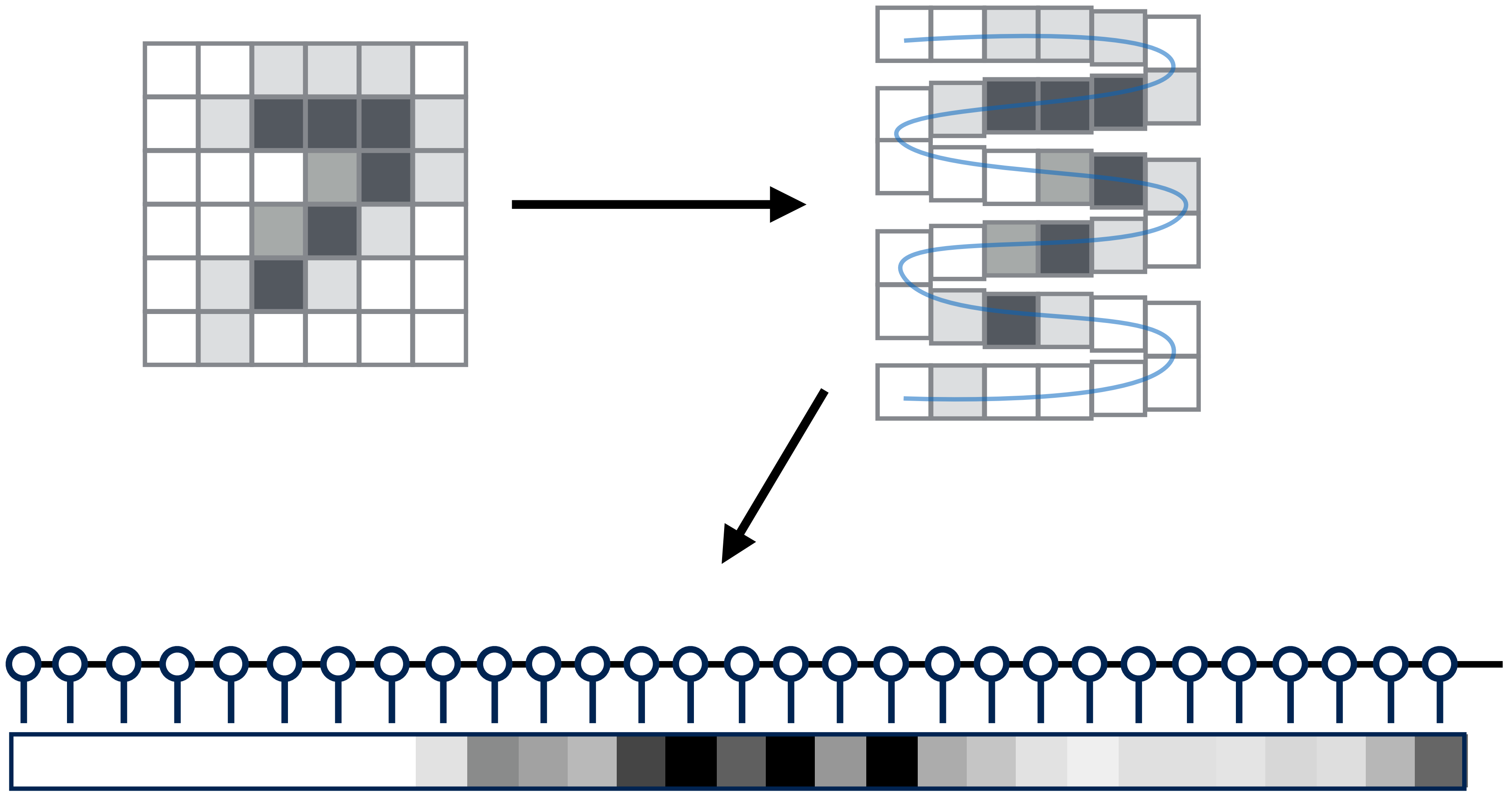
Extendable to multiple outputs

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

Output is a vector over the index l

Models exhibit "feature sharing" – only differ in center tensor

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)

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*

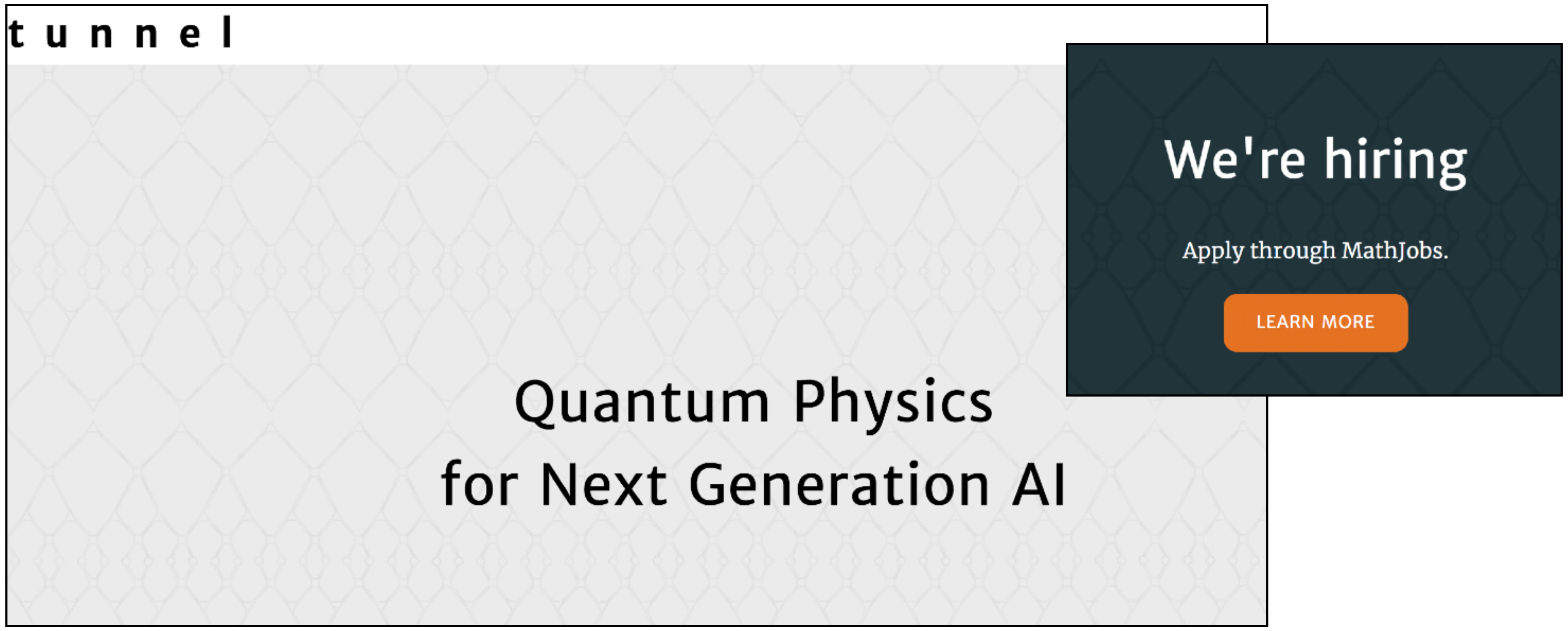
Supervised Learning

- *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*

Even startups getting into the game!

Tunnel Tech, New York City

John Terilla


t u n n e l

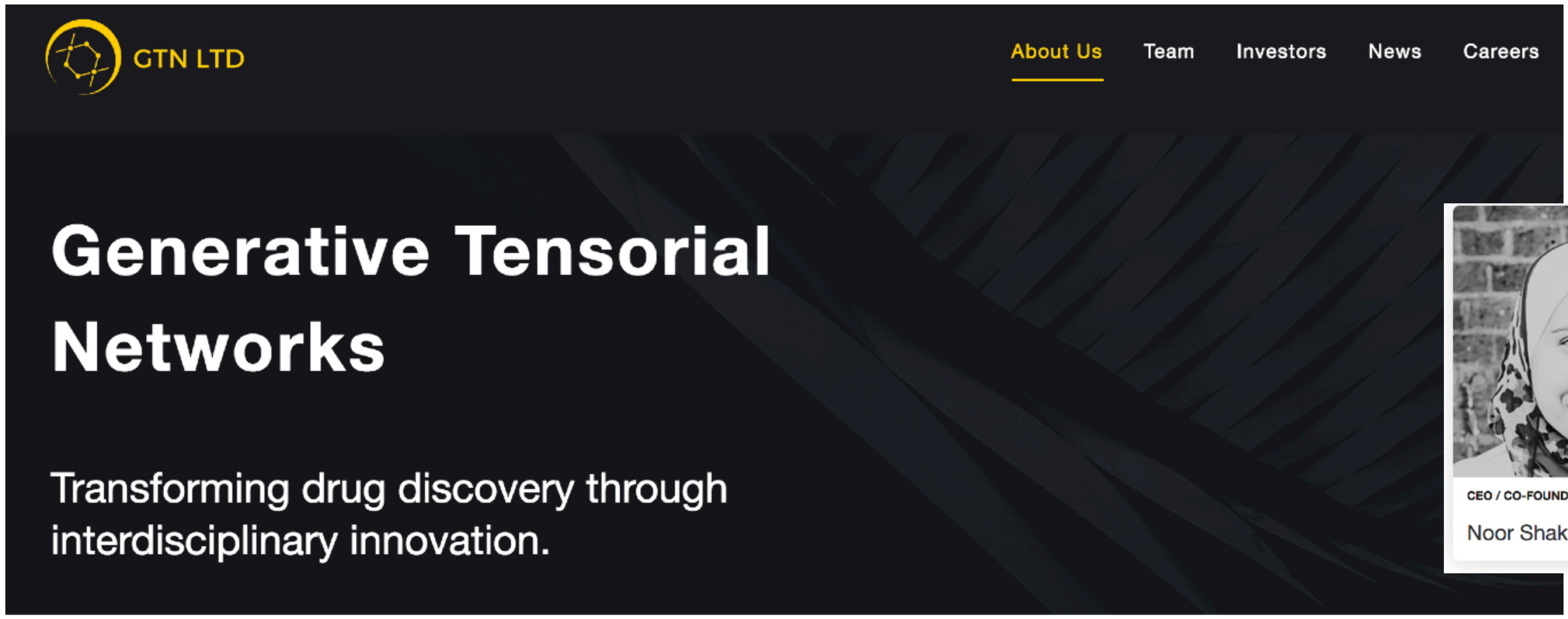
Quantum Physics for Next Generation AI

We're hiring

Apply through MathJobs.

LEARN MORE

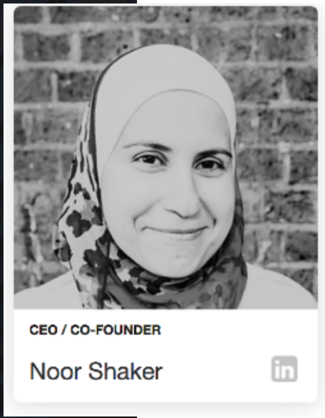
Generative Tensorial Networks (GTN), London


GTN LTD

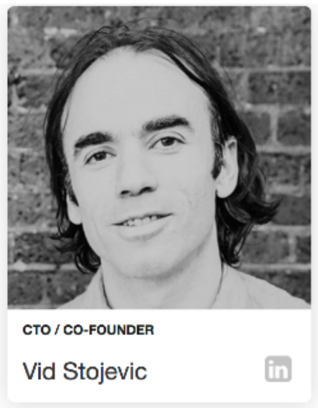
About Us Team Investors News Careers

Generative Tensorial Networks

Transforming drug discovery through interdisciplinary innovation.


CEO / CO-FOUNDER

Noor Shaker


CTO / CO-FOUNDER

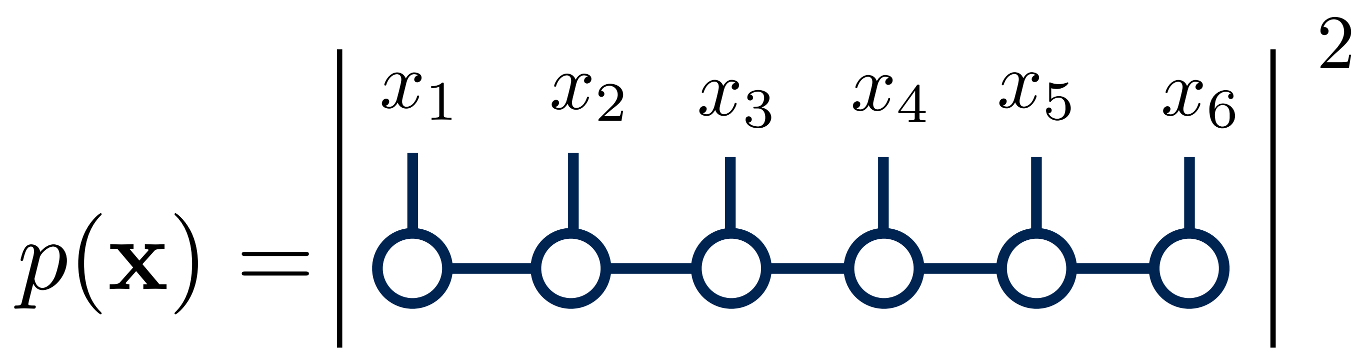
Vid Stojevic

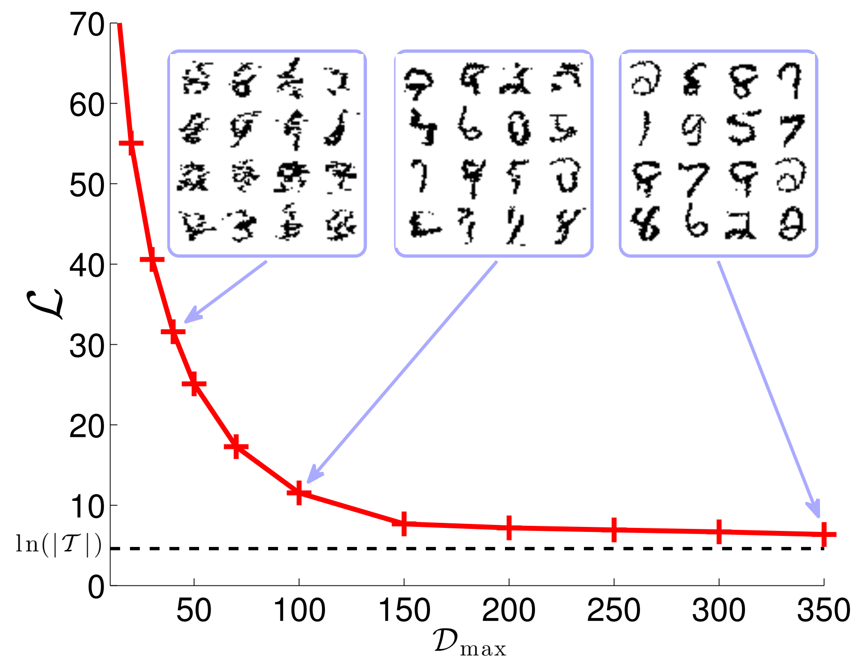
Tensor Network Machine Learning Studies

Unsupervised Generative Modeling Using MPS

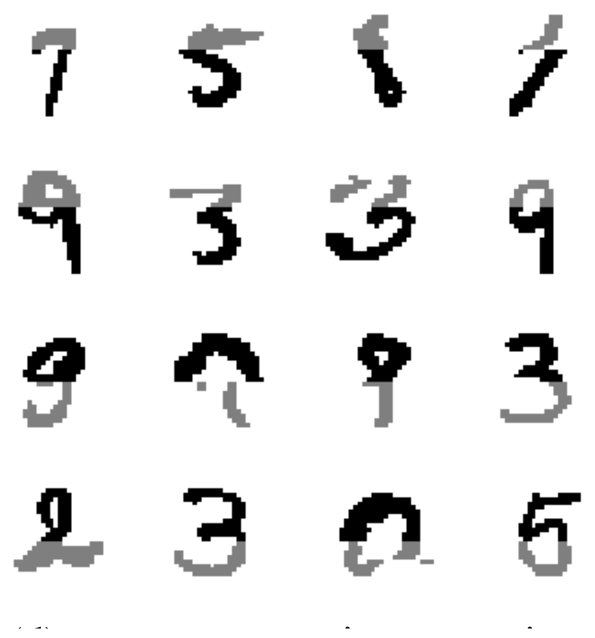
Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, Pan Zhang

- Map data to product state, tensor network weights
- <u>Squared</u> output is probability – "Born machine"
- "Perfect" sampling (no autocorrelation)





Negative Log-Likelihood

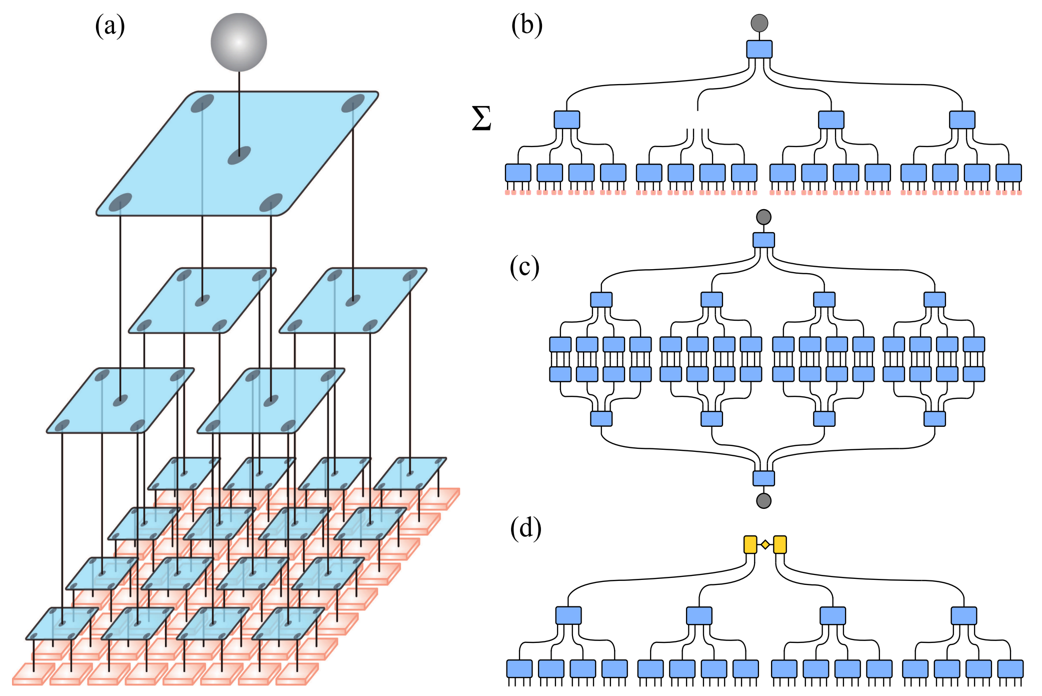


Reconstructing Testing Images

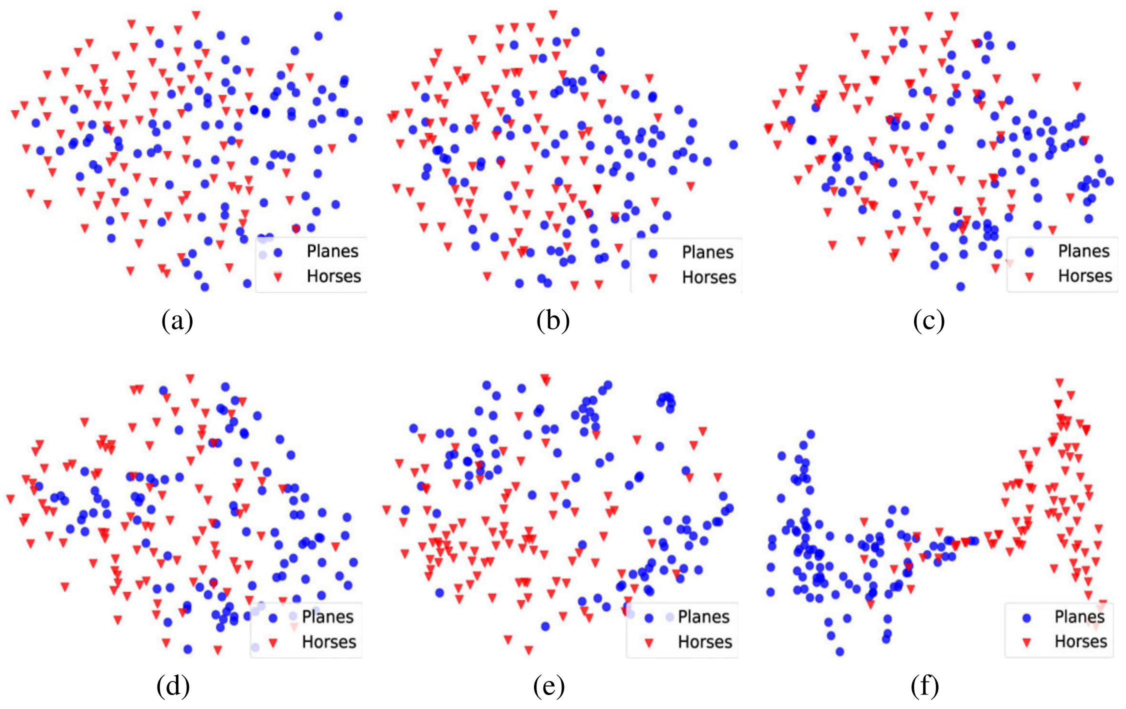
Machine Learning By Hierarchical Tensor Networks…

Ding Liu, Shi-Ju Ran, Peter Wittek, Cheng Peng, Raul Blazquez Garcia, Gang Su, Maciej Lewenstein

- Supervised learning with tree tensor networks
- Tests on MNIST, CIFAR-10
- Studied properties of the trained model (feature representations, entanglement)



Model Architecture

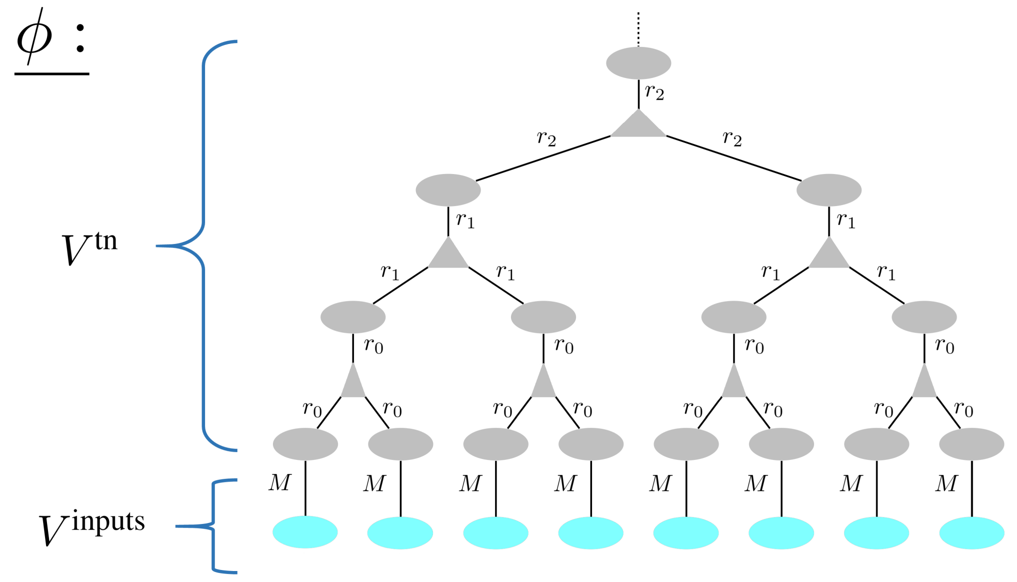


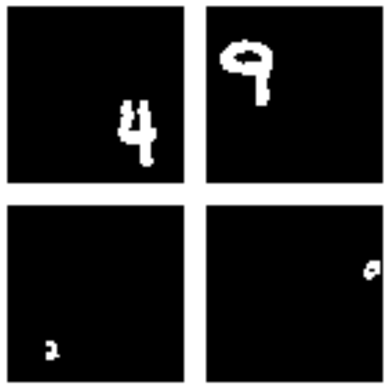
Data Representation at Different Scales

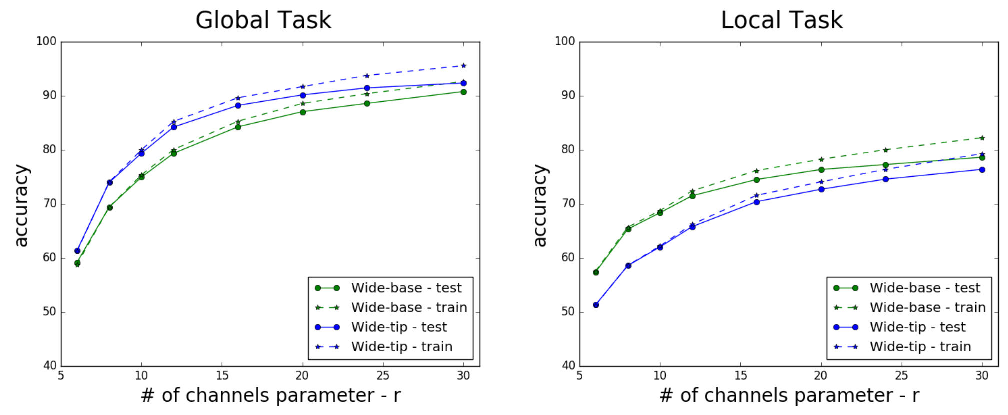
Deep Learning and Quantum Entanglement...

Yoav Levine, David Yakira, Nadav Cohen, Amnon Shashua

- "ConvAC" deep neural net = tree tensor network
- Tensor network rank as capacity of model
- Experiment on "inductive bias" of model architecture







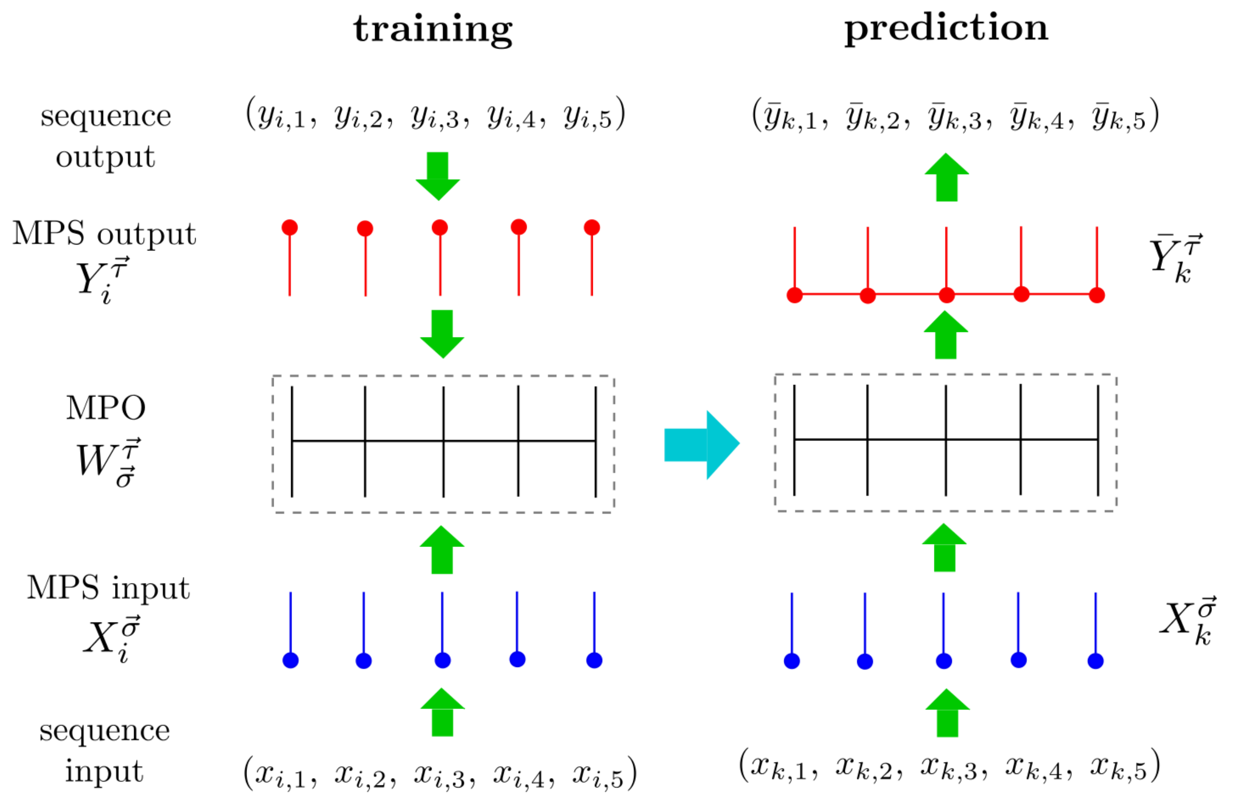
Tree Network as a Deep Neural Net

Inductive Bias Experiment

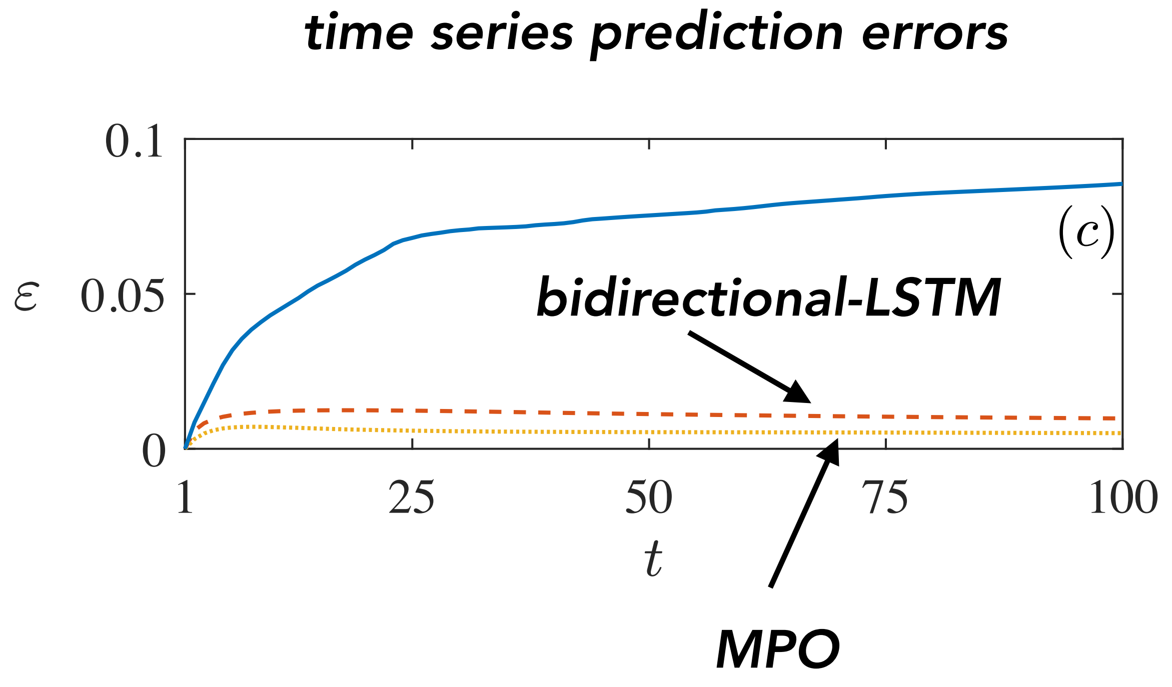
Matrix Product Operators for Sequence to Sequence…

Guo, Jie, Lu, Poletti, arxiv:1803.10908

- Product-state input processed by an MPO model
- Output is an MPS, approximated as another product state
- Capabilities like recurrent neural nets; better results than LSTM!



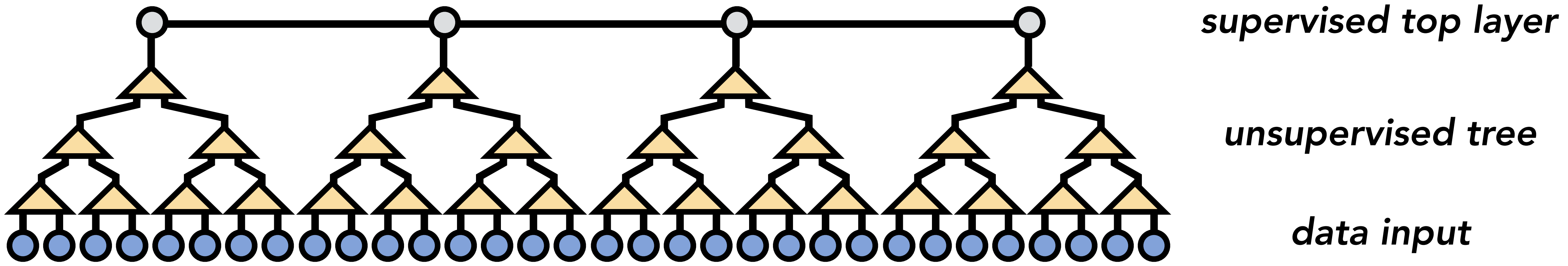
training
prediction
sequence output
$(y_{i,1},\ y_{i,2},\ y_{i,3},\ y_{i,4},\ y_{i,5})$
$(\bar{y}_{k,1},\ \bar{y}_{k,2},\ \bar{y}_{k,3},\ \bar{y}_{k,4},\ \bar{y}_{k,5})$
MPS output $Y_i^{\vec{\tau}}$
$\bar{Y}_k^{\vec{\tau}}$
MPO $W_{\vec{\sigma}}^{\vec{\tau}}$
MPS input $X_i^{\vec{\sigma}}$
$X_k^{\vec{\sigma}}$
sequence input
$(x_{i,1},\ x_{i,2},\ x_{i,3},\ x_{i,4},\ x_{i,5})$
$(x_{k,1},\ x_{k,2},\ x_{k,3},\ x_{k,4},\ x_{k,5})$



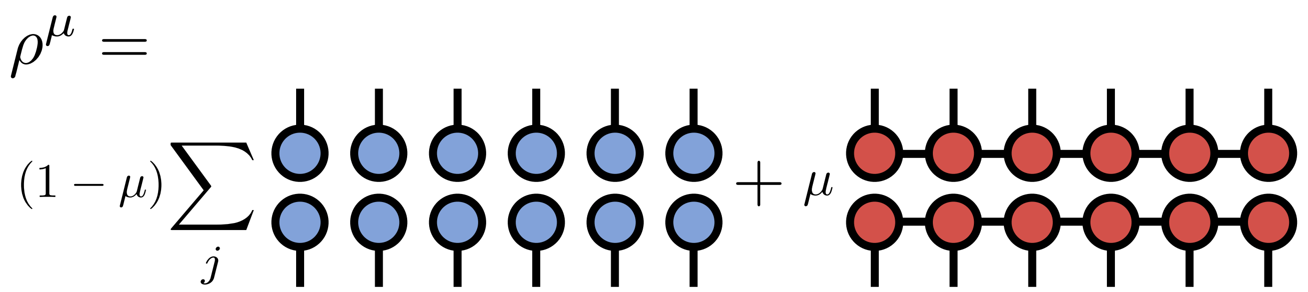
Learning Relevant Features of Data...

E.M. Stoudenmire

- Unsupervised determination of tree tensor network (compress data)
- Supervised training of top layer
- Excellent performance with "features" determined by tree tensors



89% accuracy on
Fashion MNIST data set

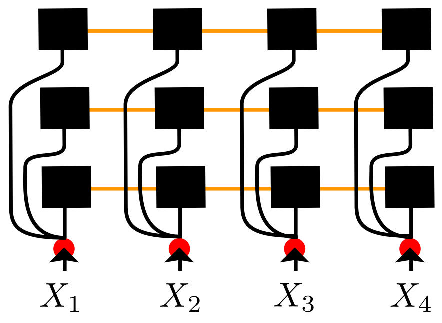


mixed training
supervised / unsupervised

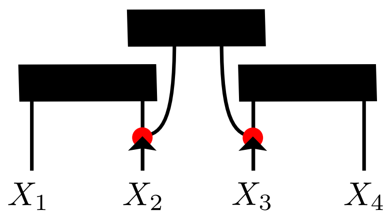
Supervised Learning with Generalized Tensor Networks

I. Glasser, N. Pancotti, J.I. Cirac, arxiv:1806.05964

- Models where inputs are copied, then processed by multiple tensor networks
- Hybrid CNN / string bond architecture gives **92.3%** on fashion MNIST test set!



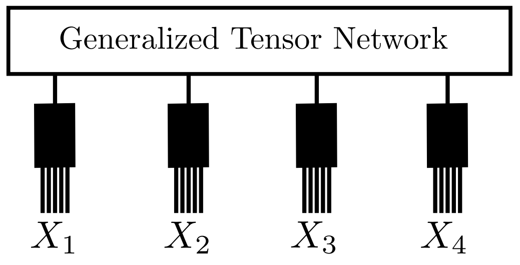
string-bond states

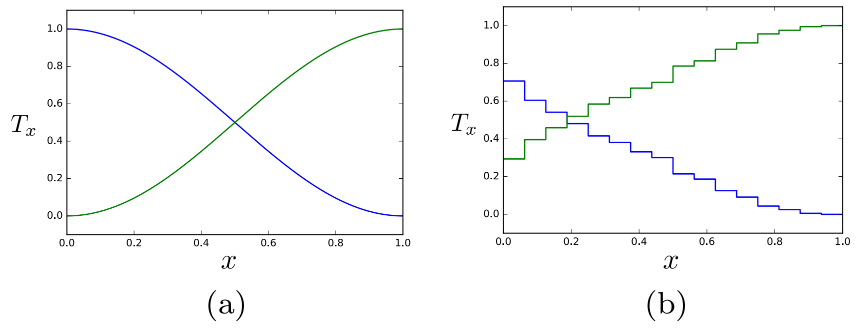


entangled plaquette states

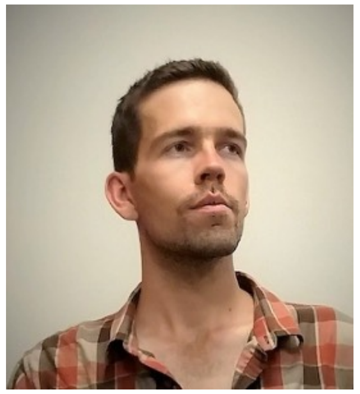
fashion MNIST

learning local features:





Quantum Machine Learning with Tensor Networks



Bill Huggins

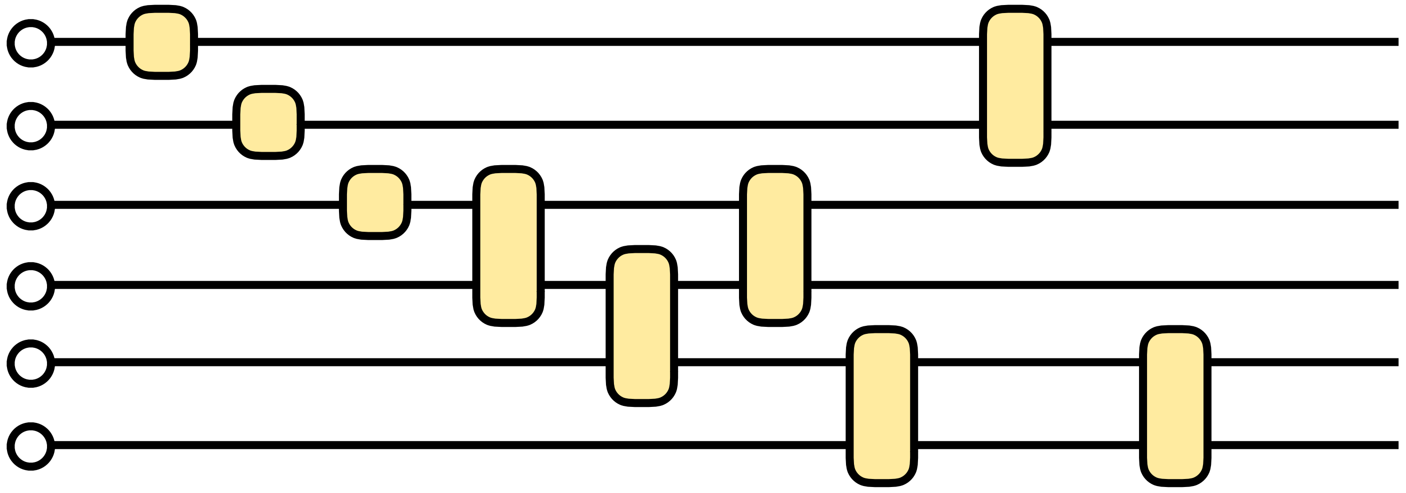
Huggins, Patil, Whaley, Stoudenmire, arxiv:1803.11537

Grant, Benedetti, et al., arxiv:1804.03680

What is a quantum computer?

A set of coherent qubits for which one can:

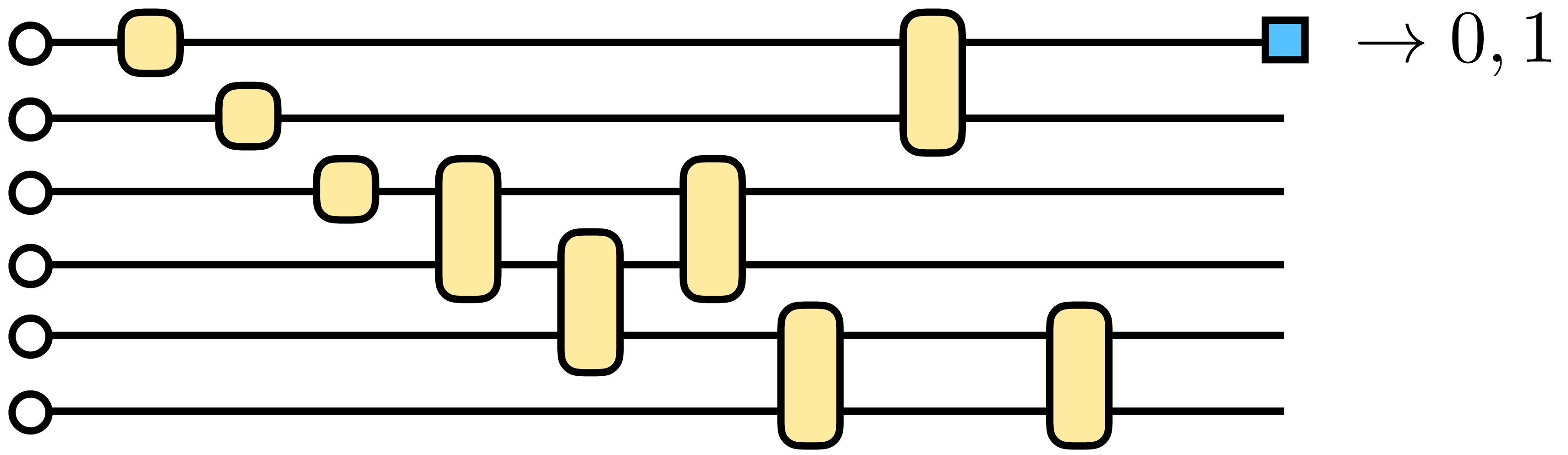
- efficiently prepare certain initial states
- apply unitary operations (usually 1- and 2-qubit)
- perform measurements



What is a quantum computer?

A set of coherent qubits for which one can:

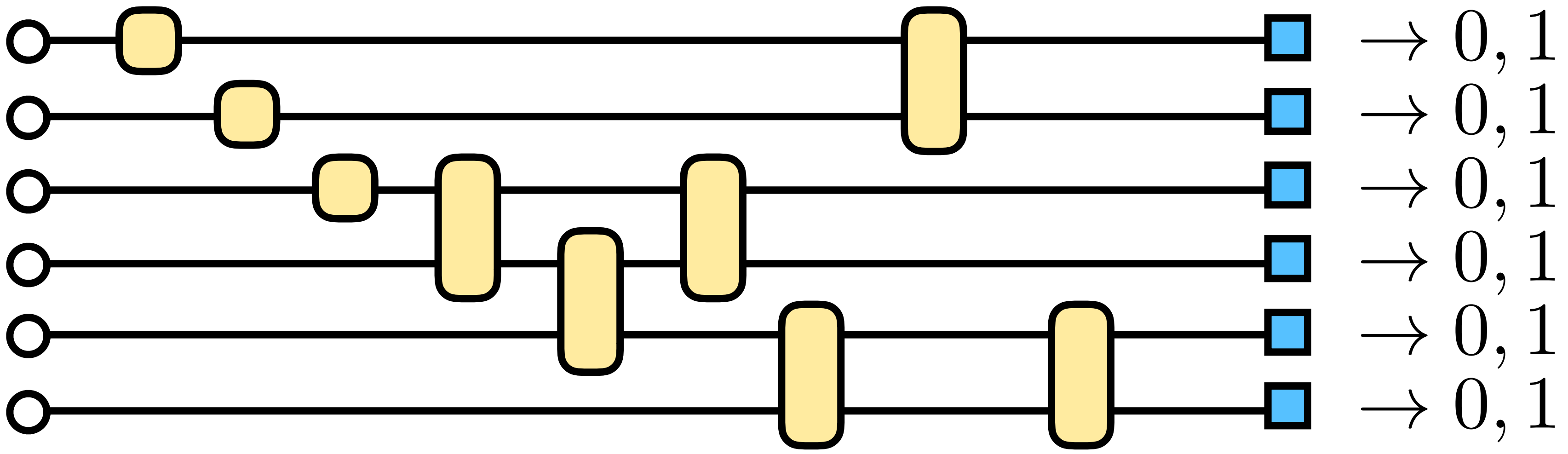
- efficiently prepare certain initial states
- apply unitary operations (usually 1- and 2-qubit)
- perform measurements



What is a quantum computer?

A set of coherent qubits for which one can:

- efficiently prepare certain initial states
- apply unitary operations (usually 1- and 2-qubit)
- perform measurements



Two recent ideas for machine learning with a quantum computer

1. supervised / discriminative learning

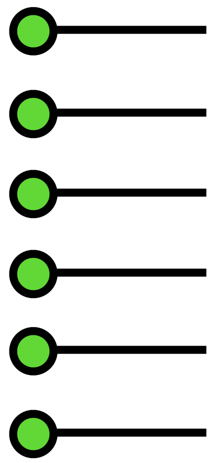
Farhi, Neven, arxiv:1802.0600

Schuld, Killoran, arxiv:1803.07128

Two recent ideas for machine learning with a quantum computer

1. supervised / discriminative learning

$\mathbf{x} \rightarrow$



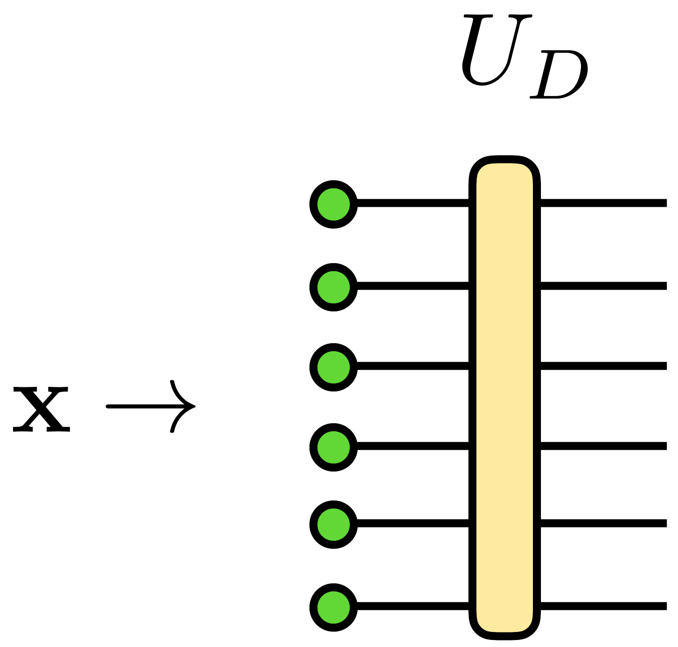
- prepare data as product state

Farhi, Neven, arxiv:1802.0600

Schuld, Killoran, arxiv:1803.07128

Two recent ideas for machine learning with a quantum computer

1. supervised / discriminative learning



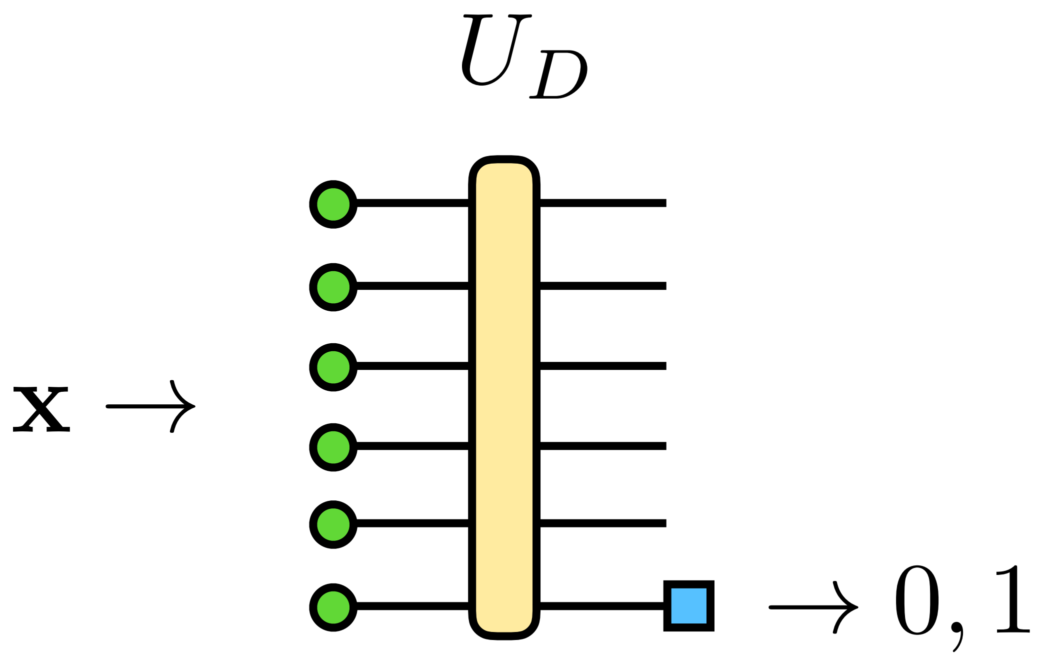
- prepare data as product state
- apply gates to prepared state

Farhi, Neven, arxiv:1802.0600

Schuld, Killoran, arxiv:1803.07128

Two recent ideas for machine learning with a quantum computer

1. supervised / discriminative learning



- prepare data as product state
- apply gates to prepared state
- measure output qubit

Farhi, Neven, arxiv:1802.0600

Schuld, Killoran, arxiv:1803.07128

Two recent ideas for machine learning with a quantum computer

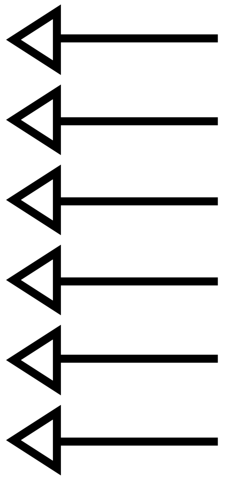
2. generative modeling

Gao, Zhang, Duan, arxiv:1711.02038

Benedetti, Garcia-Pintos, Nam, Perdomo-Ortiz, arxiv:1801.07686

Two recent ideas for machine learning with a quantum computer

2. generative modeling



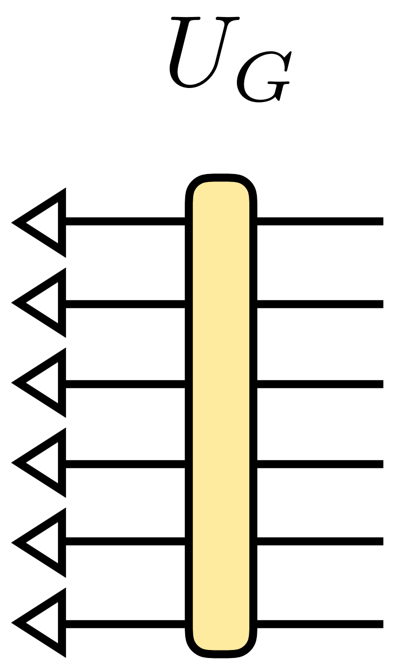
- prepare reference state

Gao, Zhang, Duan, arxiv:1711.02038

Benedetti, Garcia-Pintos, Nam, Perdomo-Ortiz, arxiv:1801.07686

Two recent ideas for machine learning with a quantum computer

2. generative modeling



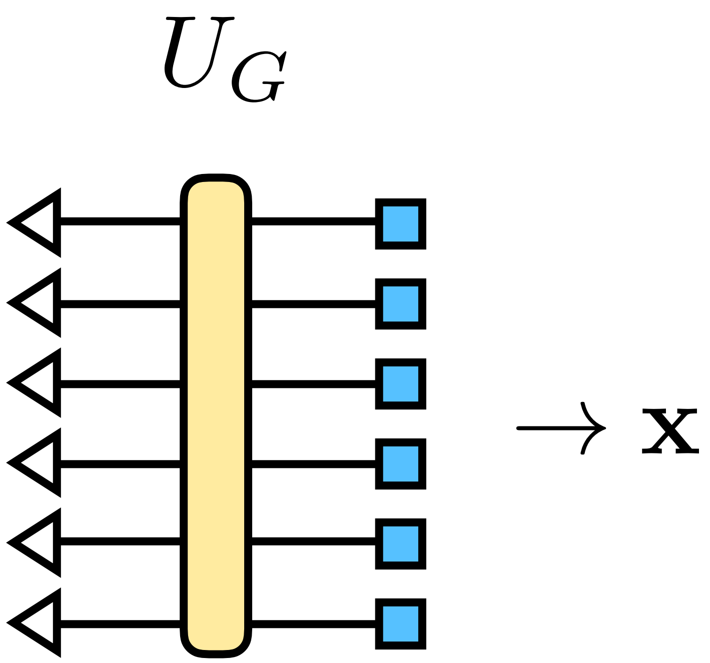
- prepare reference state
- apply gates to qubit

Gao, Zhang, Duan, arxiv:1711.02038

Benedetti, Garcia-Pintos, Nam, Perdomo-Ortiz, arxiv:1801.07686

Two recent ideas for machine learning with a quantum computer

2. generative modeling

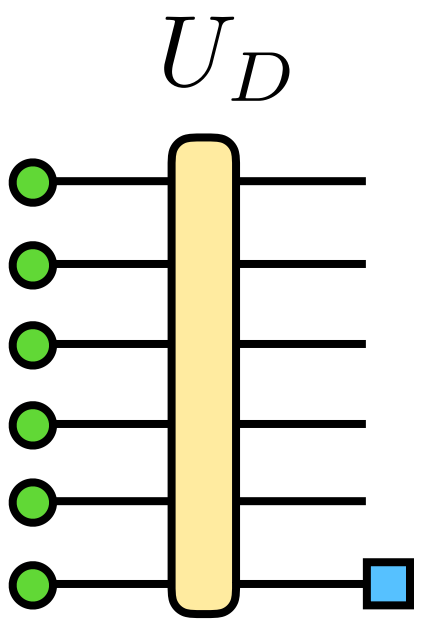


- prepare reference state
- apply gates to qubit
- measure all qubits

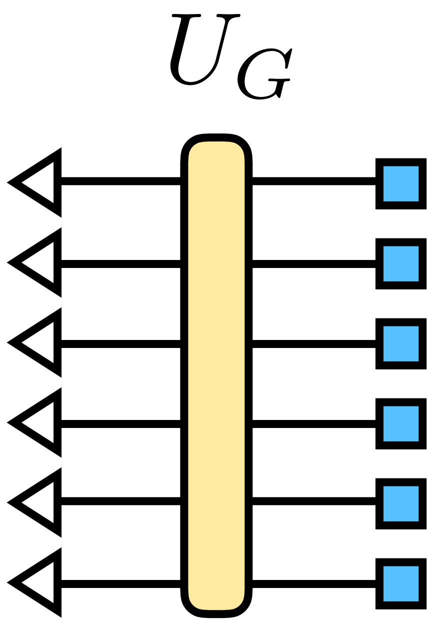
Gao, Zhang, Duan, arxiv:1711.02038

Benedetti, Garcia-Pintos, Nam, Perdomo-Ortiz, arxiv:1801.07686

Two issues with these proposals

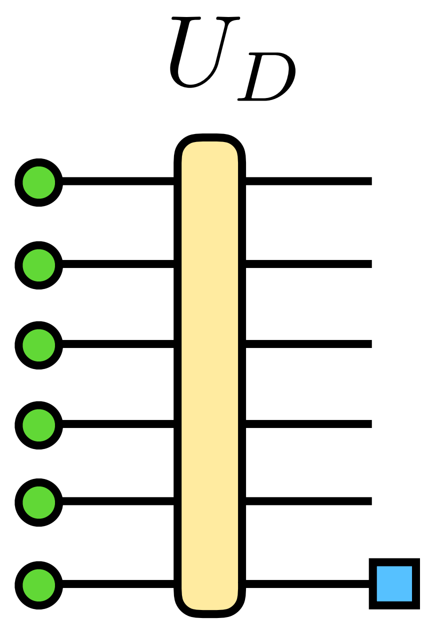


discriminative

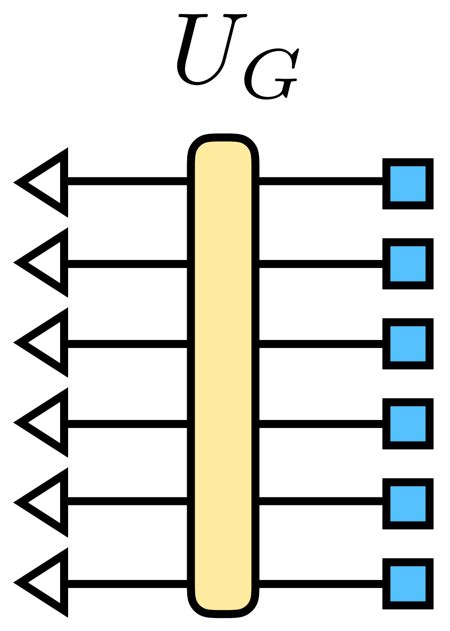


generative

Two issues with these proposals



discriminative

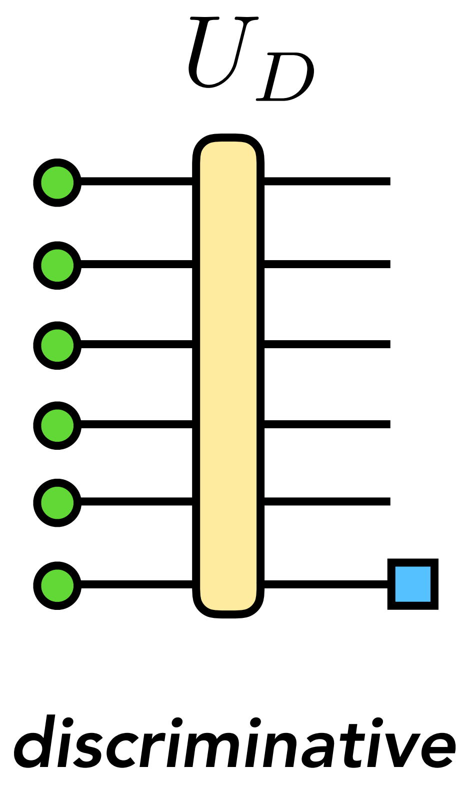


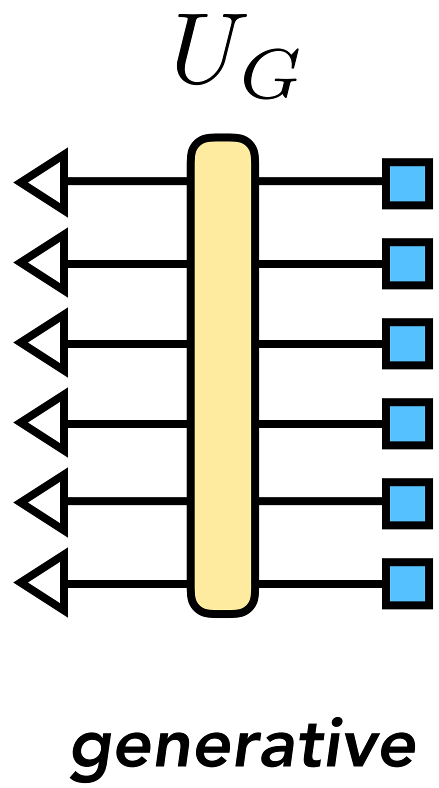
generative

1. efficient parameterization of N-qubit circuit?
   (vanishing gradient?*)

* McClean, Boixo, et al., arxiv:1803.11173

Two issues with these proposals



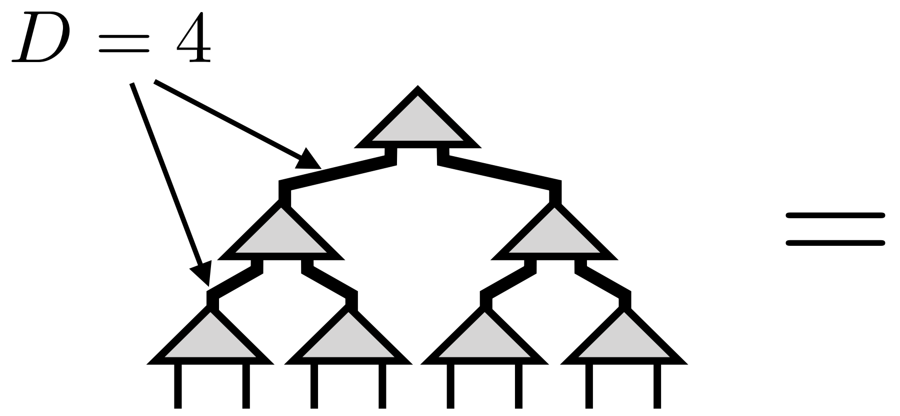


1. efficient parameterization of N-qubit circuit? (vanishing gradient?*)

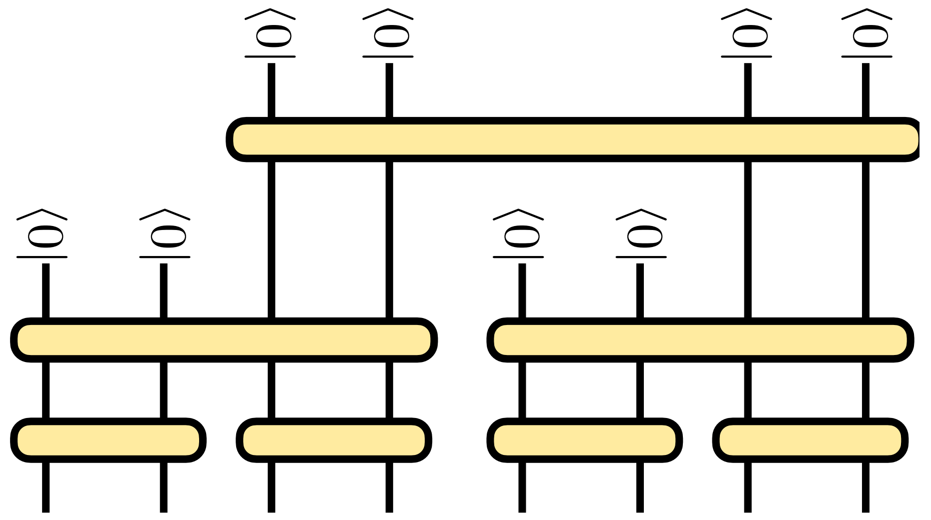
2. require too many qubits for realistic data sizes

* McClean, Boixo, et al., arxiv:1803.11173

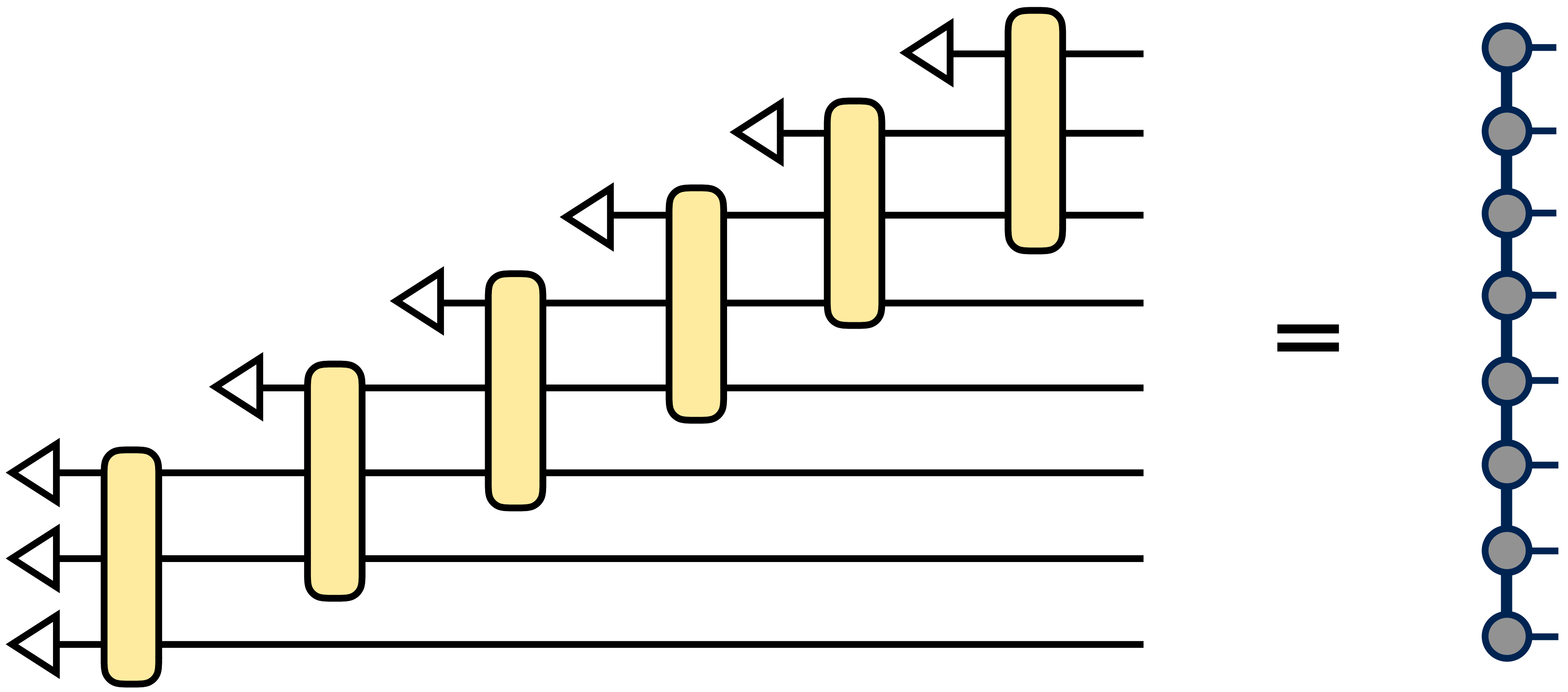
Tensor networks are equivalent to quantum circuits



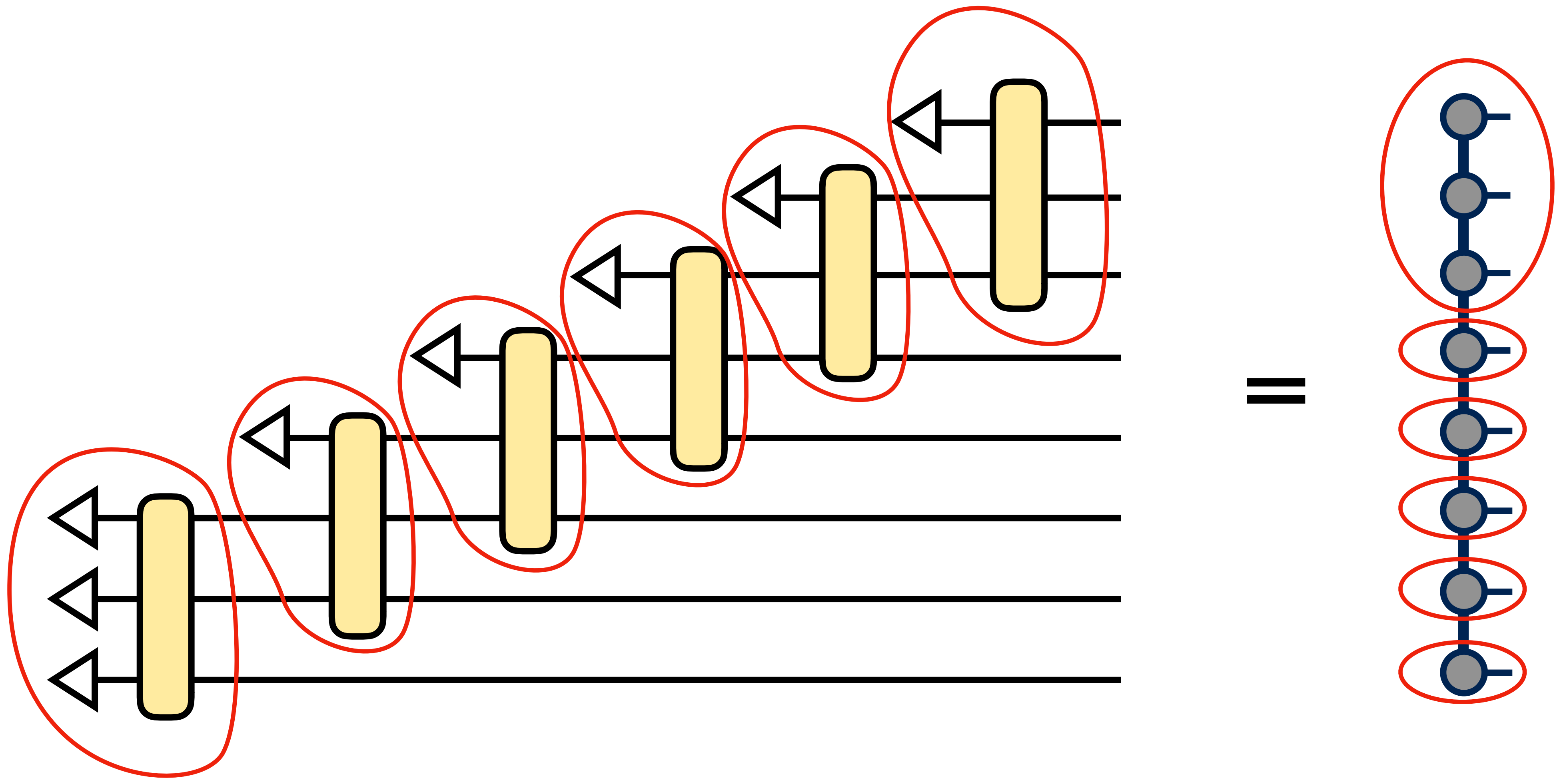
$$=$$



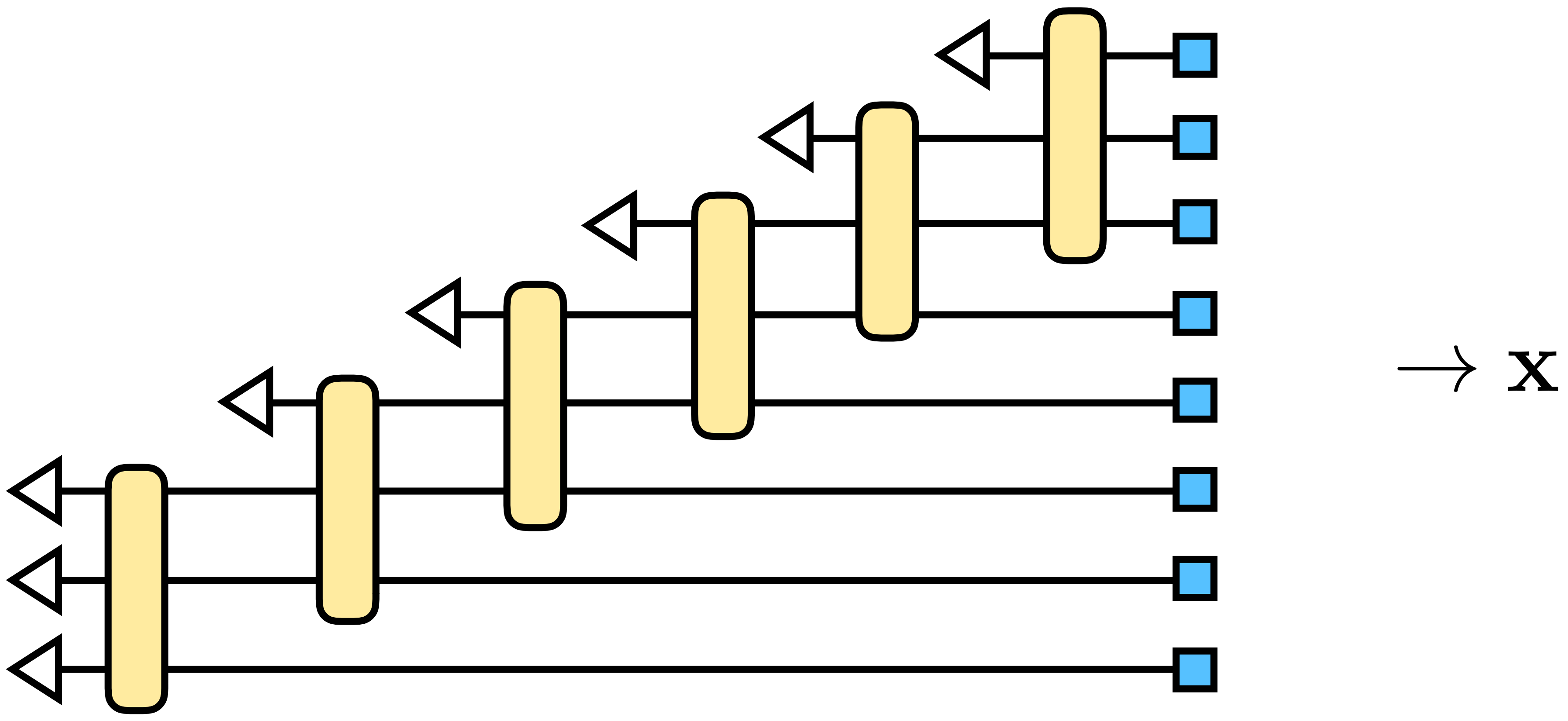
Quantum circuit for matrix product state (m = 4)



Quantum circuit for matrix product state (**m** = 4)

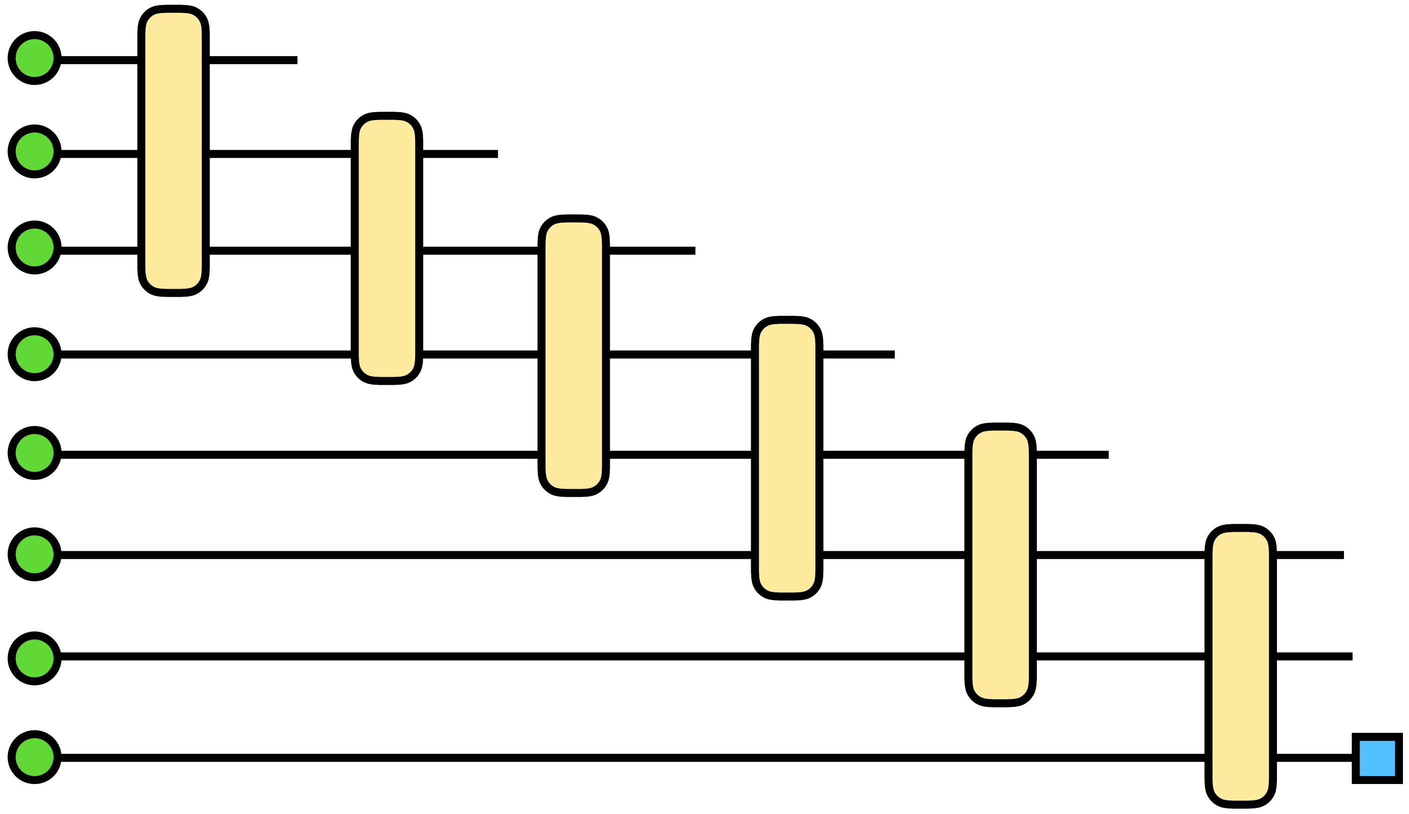


Suggests MPS parameterization of generative quantum model

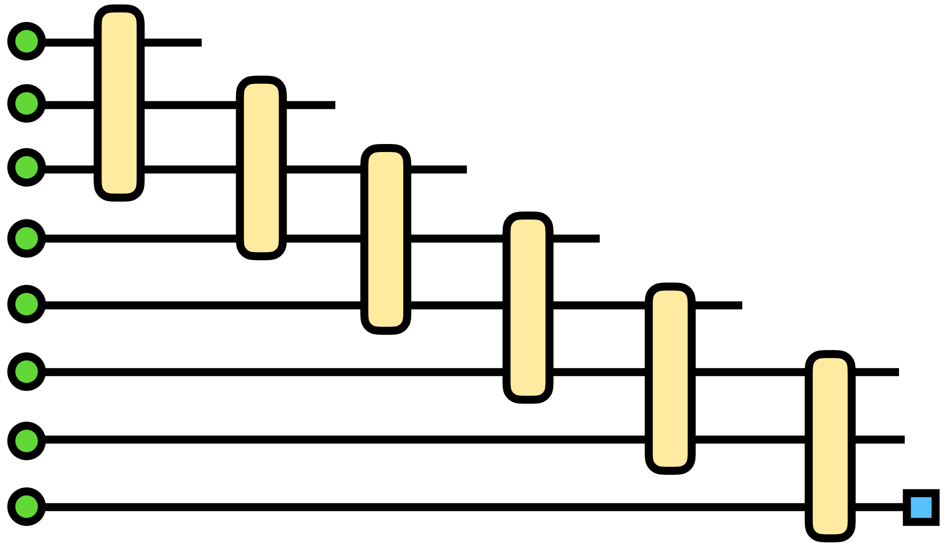


- much fewer parameters than arbitrary circuit
- can initialize with classically optimized MPS

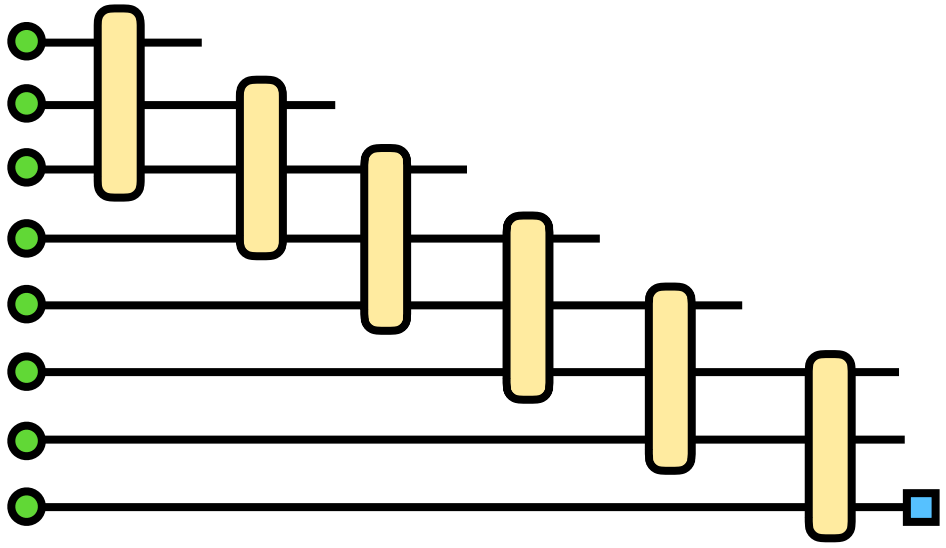
Discriminative model basically the reverse



Training a quantum program:

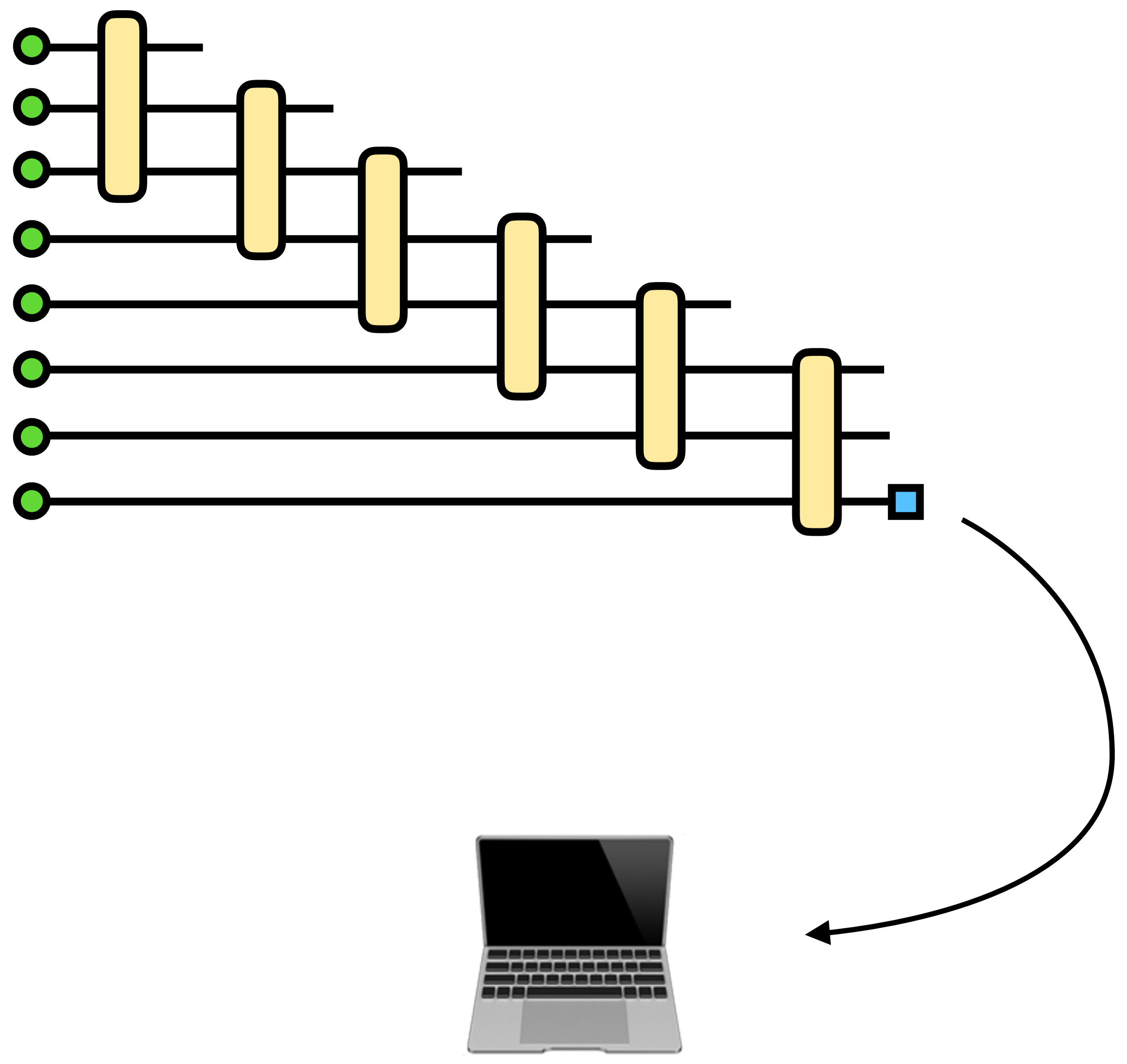


Training a quantum program:



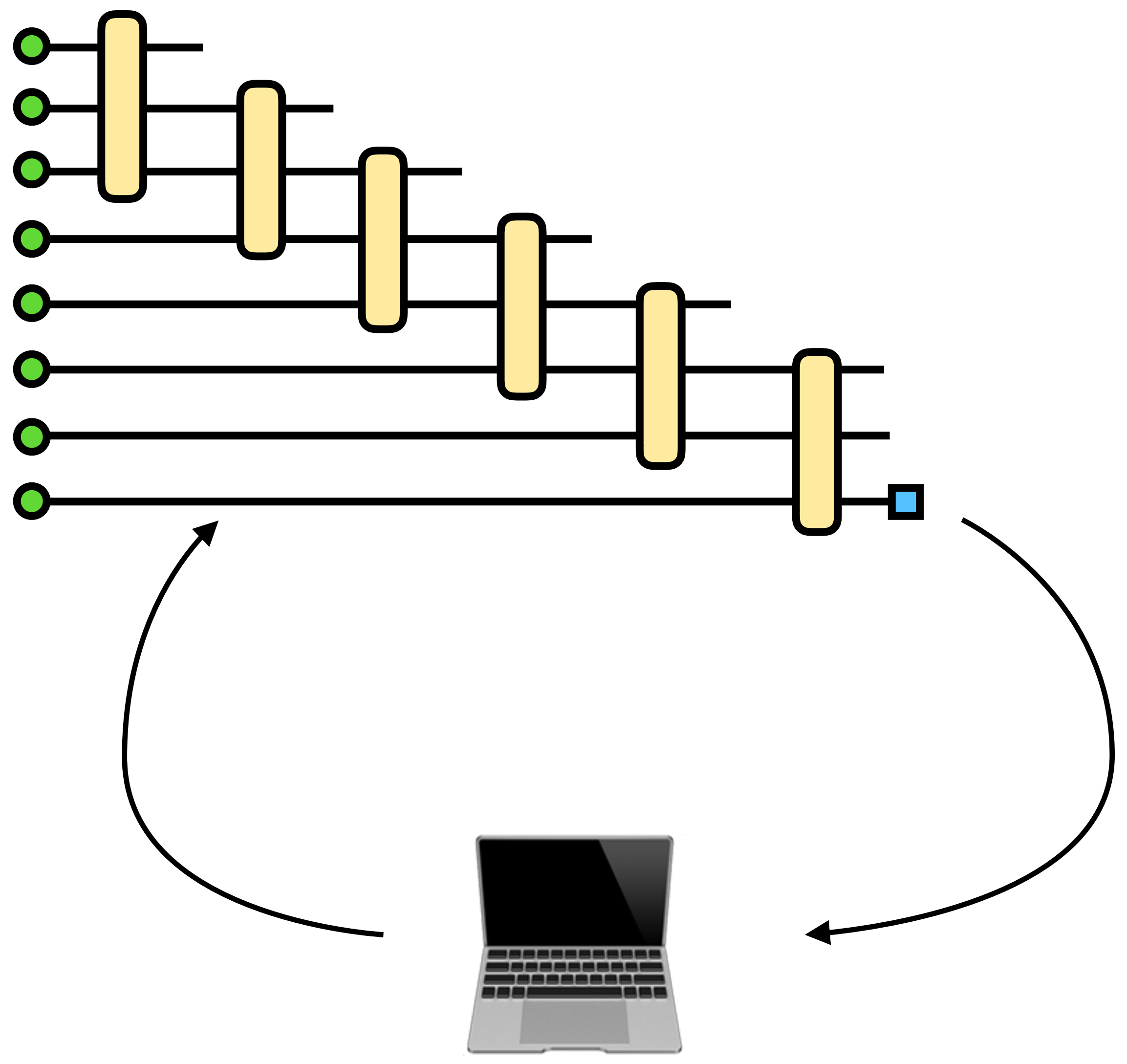
- run the program (multiple times to estimate output)

Training a quantum program:



- run the program (multiple times to estimate output)
- feed results to classical algorithm

Training a quantum program:

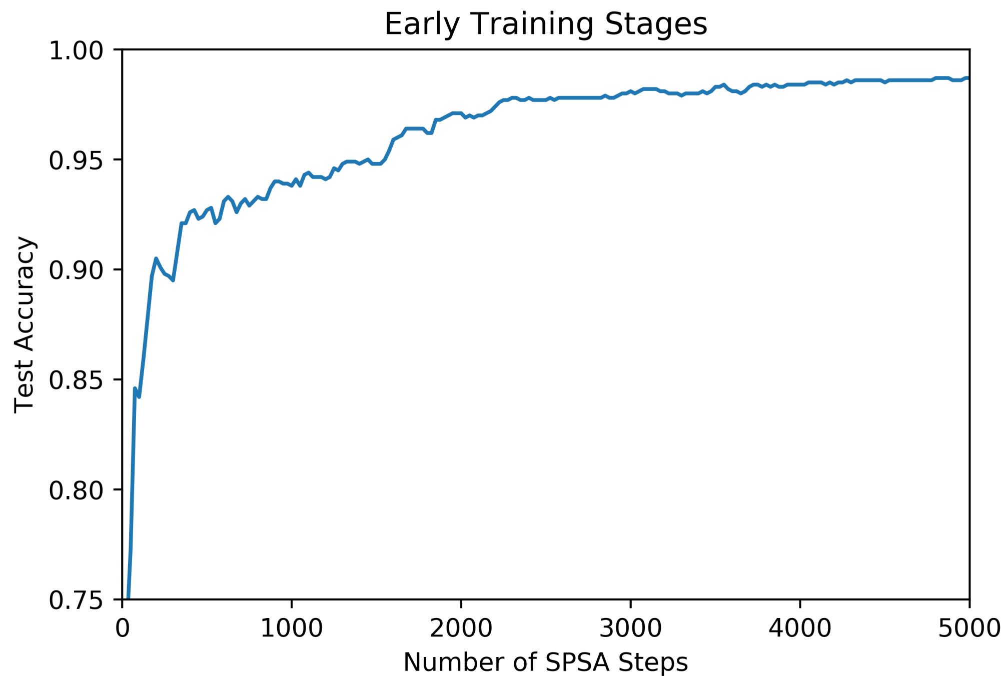


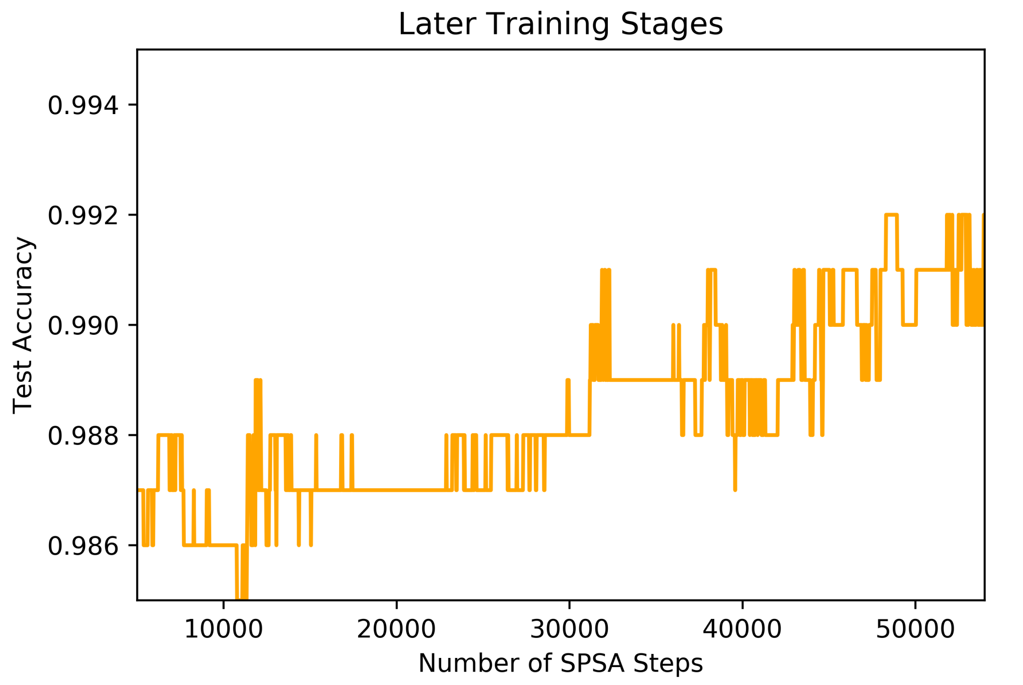
- run the program (multiple times to estimate output)
- feed results to classical algorithm
- algorithm proposes new parameters

Also possible to estimate gradient using modified circuit

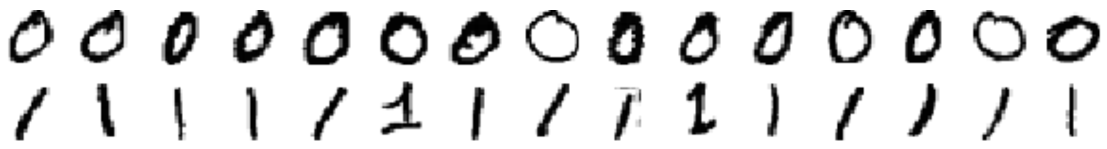
Test discriminative idea, using only operations available to quantum hardware:

Bill Huggins





8x8 images (MNIST)
distinguish 0's from 1's



Obtain 99% accuracy
training & test

Test discriminative idea, using only operations available to quantum hardware

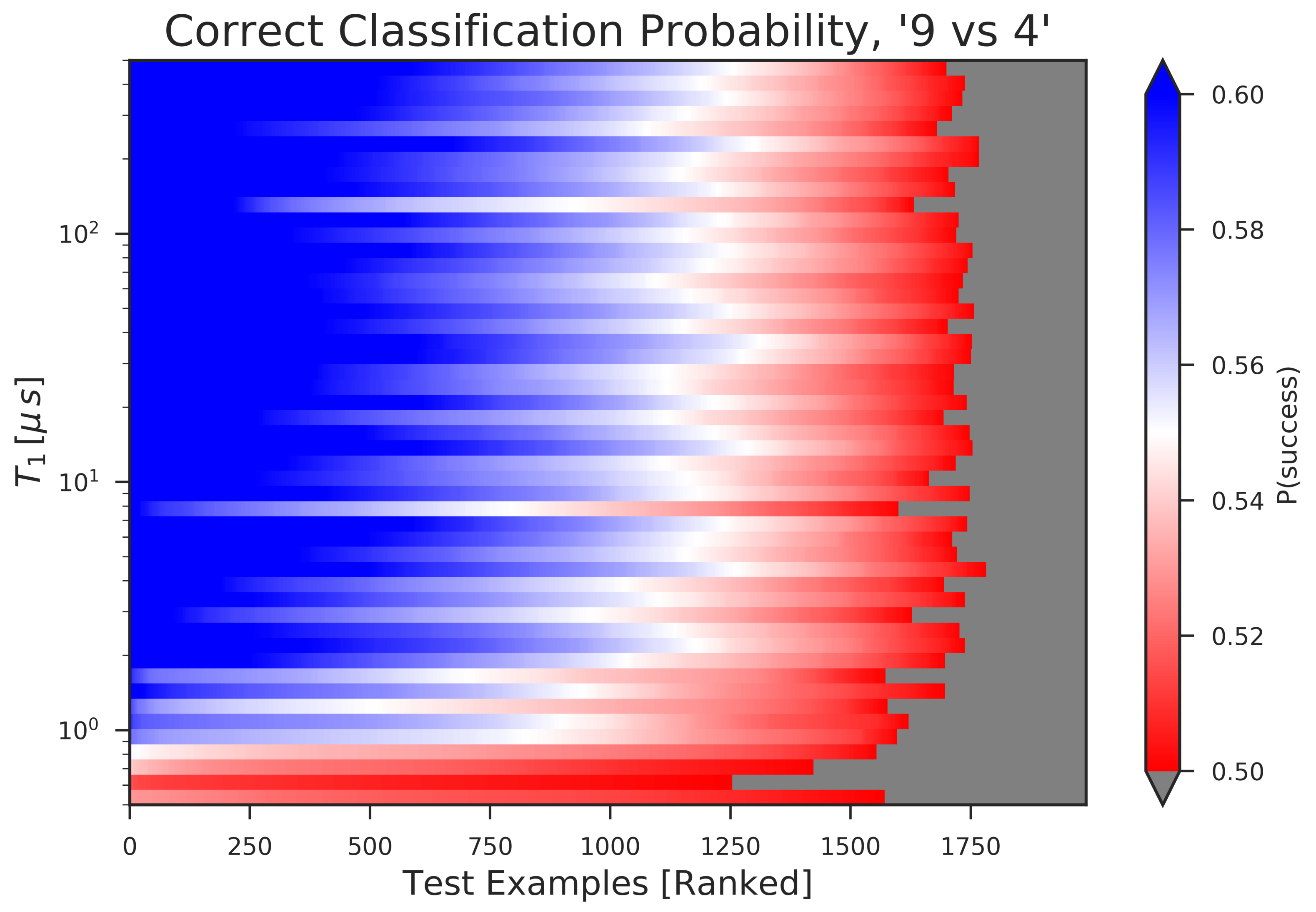
Bill Huggins

Steps to train ("SPSA" algorithm):

- pick one of the angles of the unitaries
- make two new circuits:
    - slight increase of the angle
    - slight decrease of the angle
- evaluate both & accept the better one

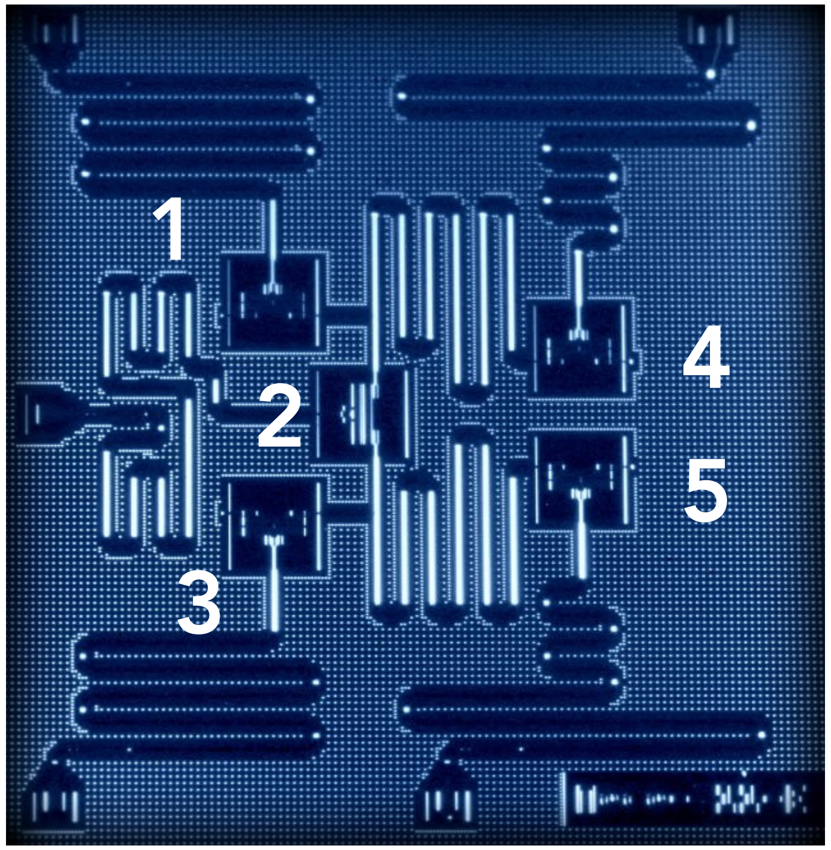
Evidence of robustness to noise

Increasing noise



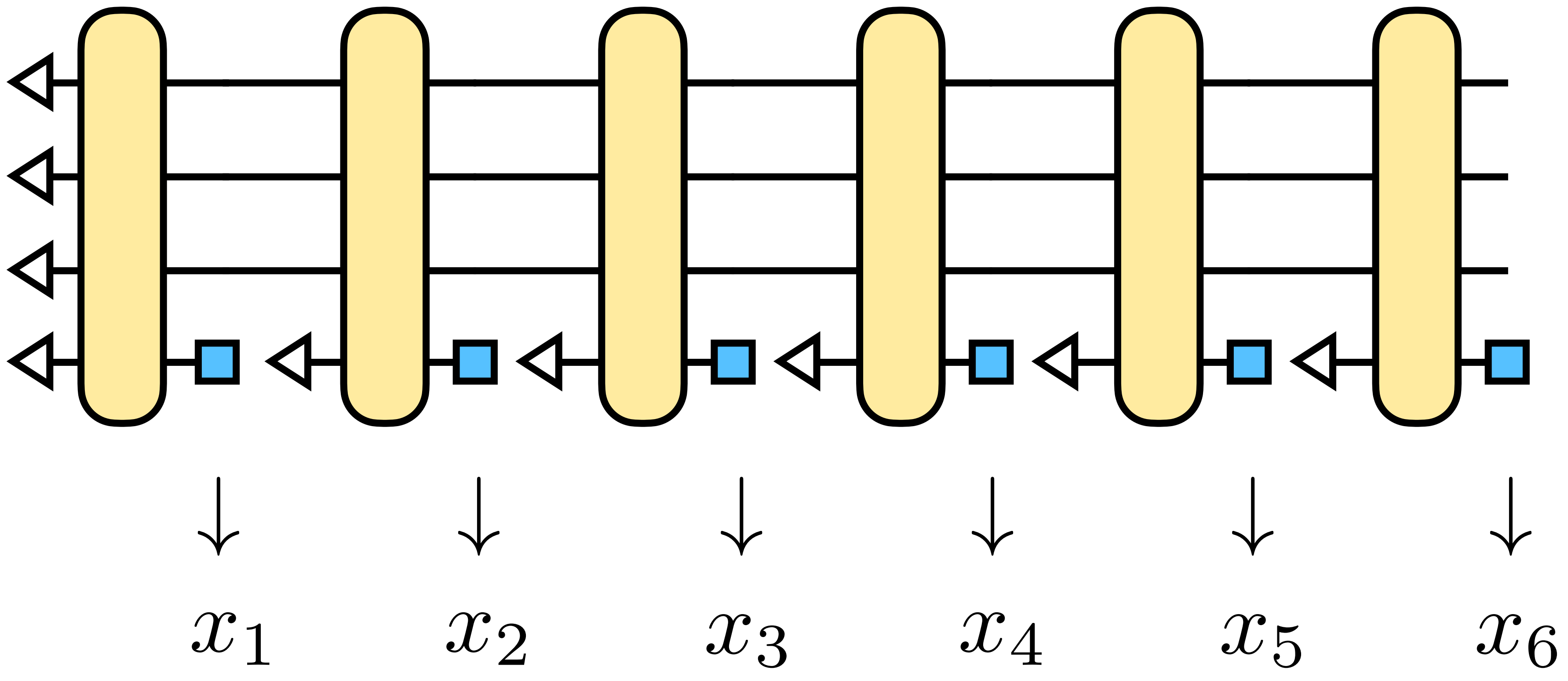
As long as correct output > 50% likely, can sample to get correct answer

Near-term quantum computers (of high quality) will have a **limited number** of qubits

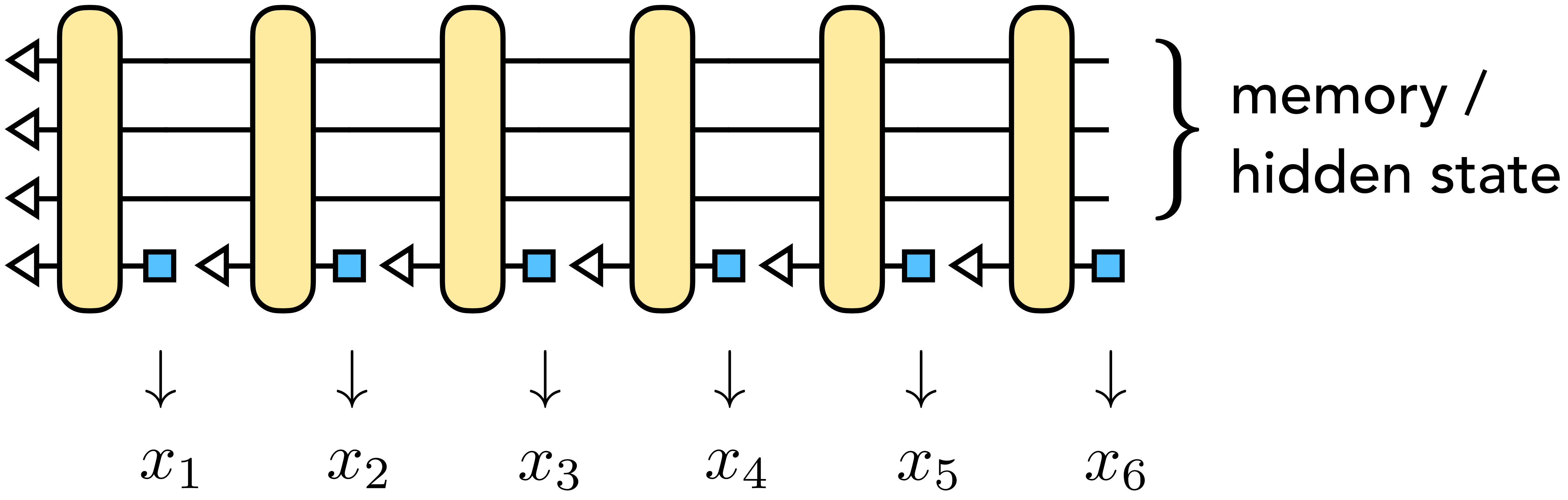


IBM quantum computer

Sampling higher-dimensional output than number of qubits

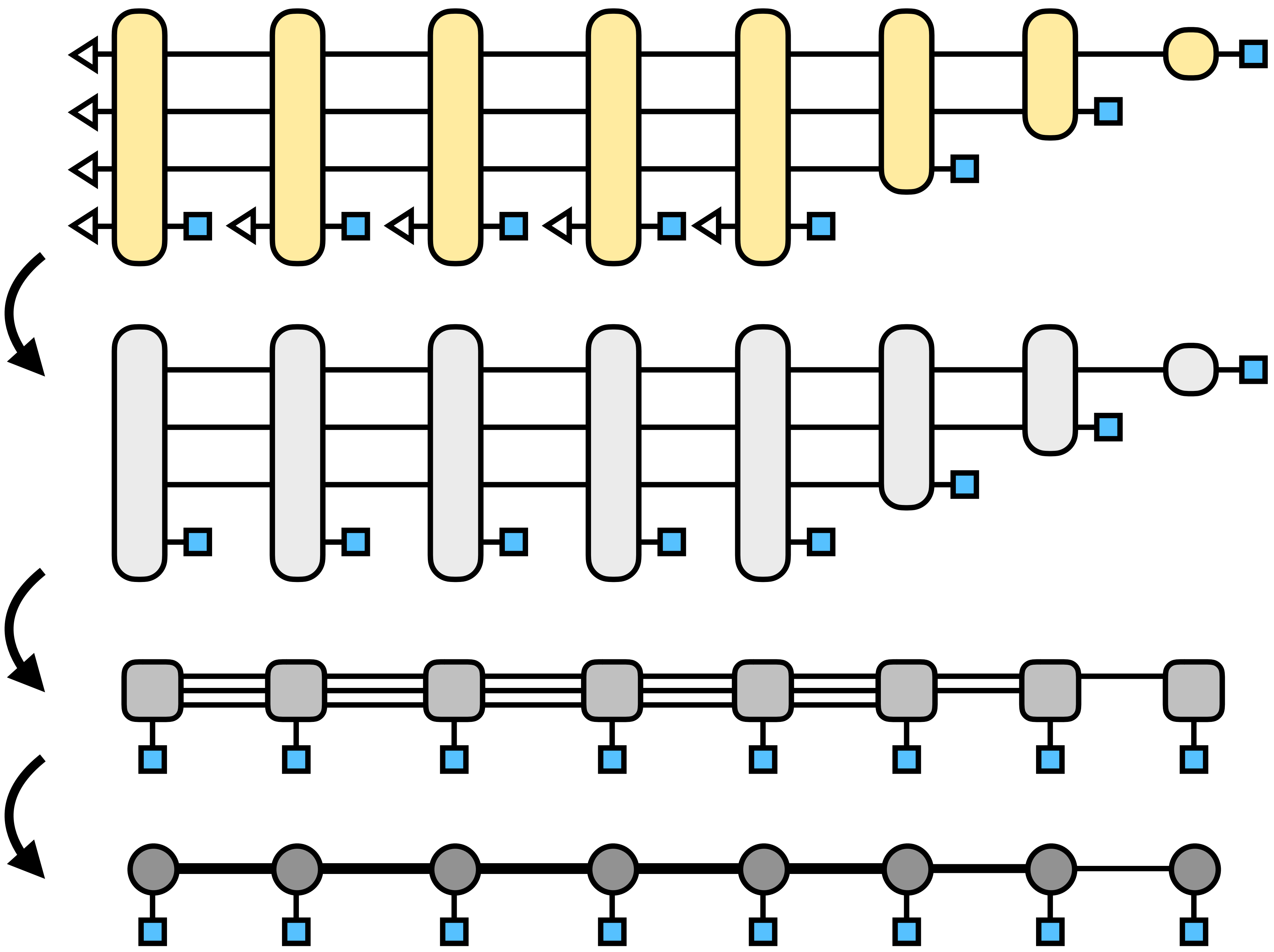


Sampling higher-dimensional output than number of qubits



memory / hidden state size
exponential in number of qubits

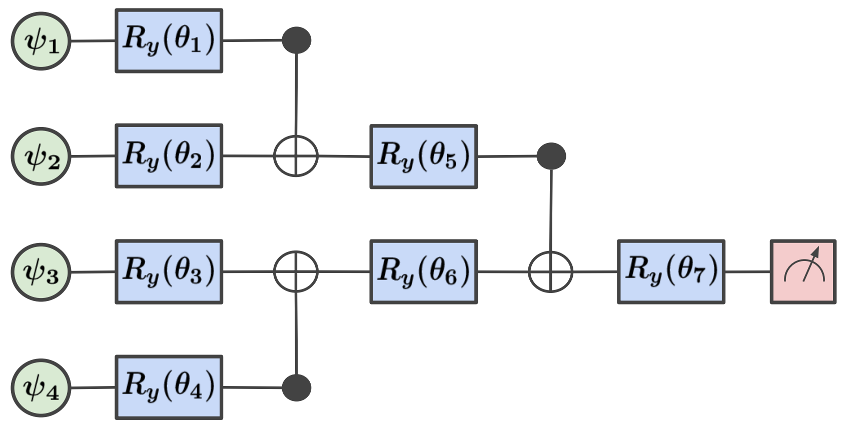
Equivalent to sampling from a matrix product state



Test of tensor network model on actual quantum device!

4.4 Deployment on a quantum computer

In this experiment we deployed the Iris classifier for classes 1 and 2 (see Sec. 2) on the ibmqx4 quantum computer available in the IBM Quantum Experience. As shown in Fig. 6, this TTN classifier has three CNOT gates and seven rotations in the Y direction. A test set of 34 unseen examples was used to determine accuracy. For each example, the circuit was run 400 times, and the samples were used to compute the most likely class. The circuit correctly classified 100% of the test set, and achieved a cost function value of 0.0811 (Eq. (3)).



θ_1	3.64129925
θ_2	1.04422998
θ_3	2.99327683
θ_4	4.18602991
θ_5	1.3215133
θ_6	2.78138208
θ_7	4.68090534

Figure 6: *Iris TTN classifier circuit schematic and parameters.*

Grant, Benedetti, Cao, Hallam, Lockhart, Stojevic, Green, Severini, arxiv:1804.03680

Learning Relevant Features of Data With Tensor Networks

EM Stoudenmire, Quant. Sci. Tech. 3, 034003 (2018)

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

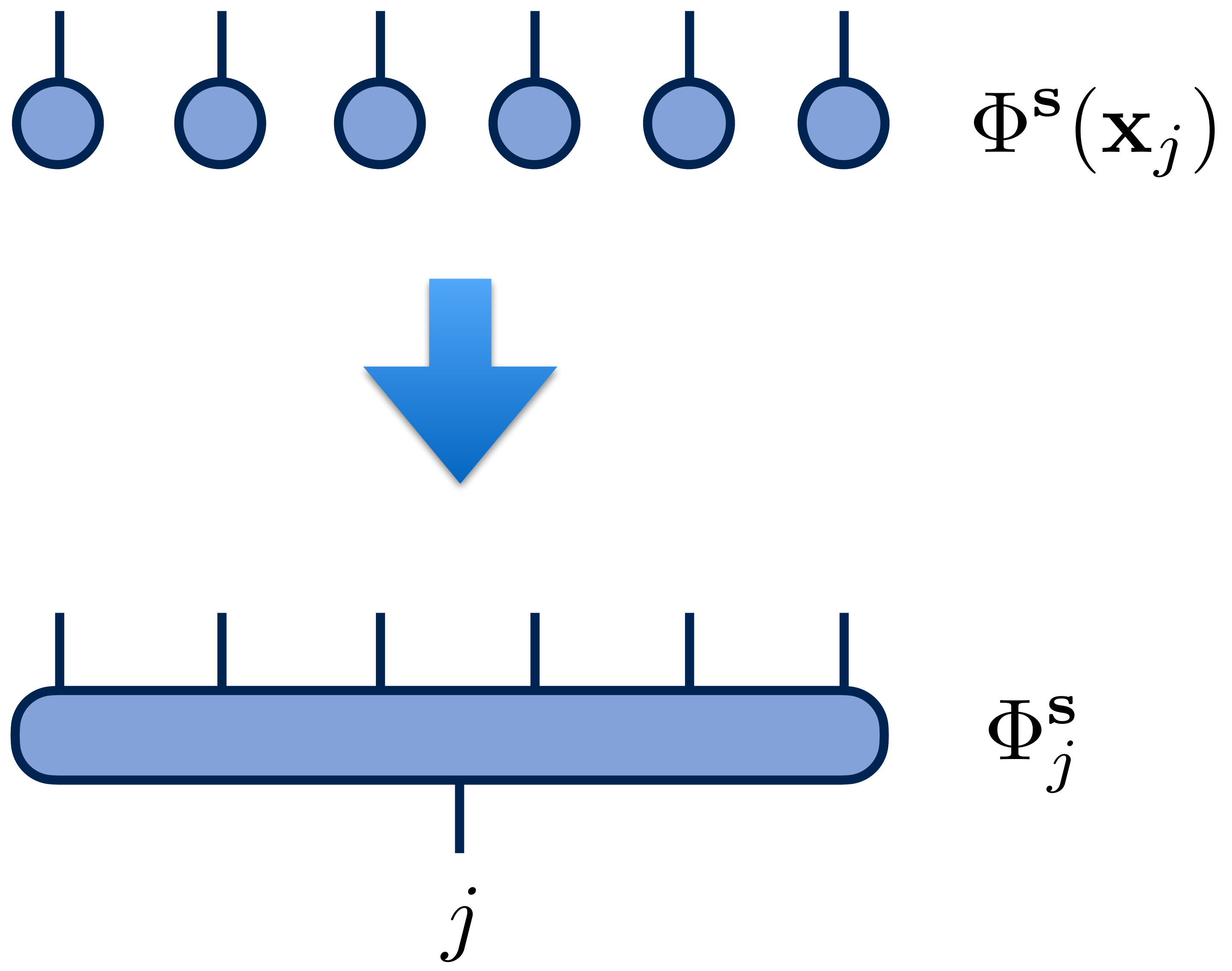
$$W = \sum_j \alpha_j \, \Phi(\mathbf{x}_j)$$

Holds for wide variety of cost functions / tasks

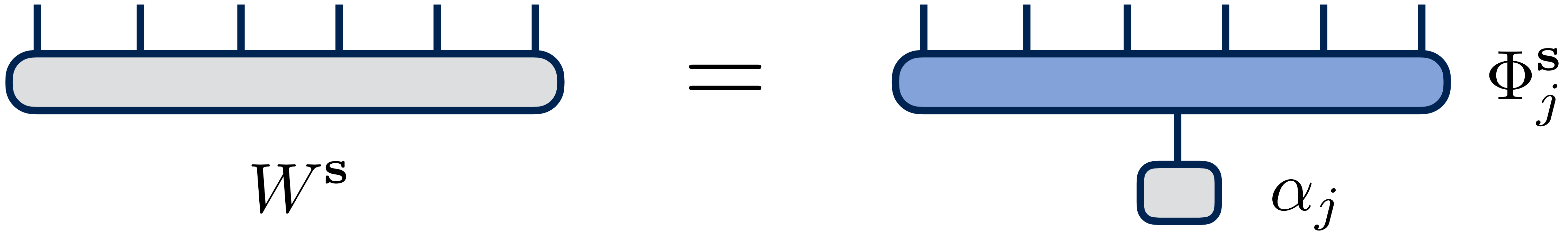
"representer theorem"

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

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

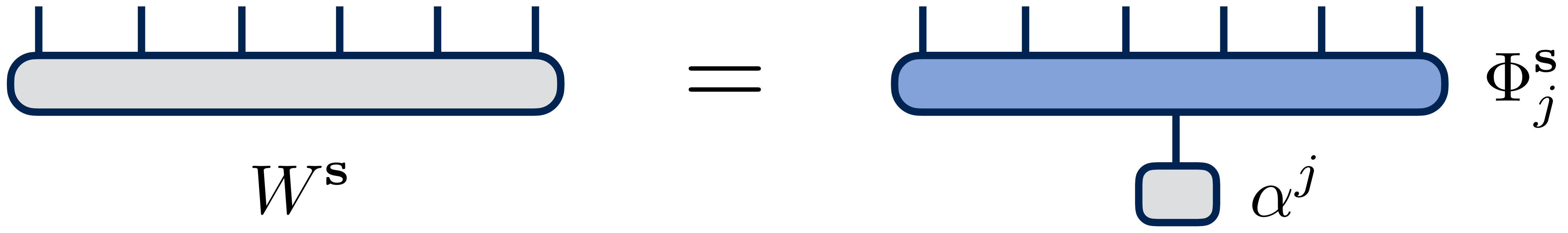


Representer theorem says

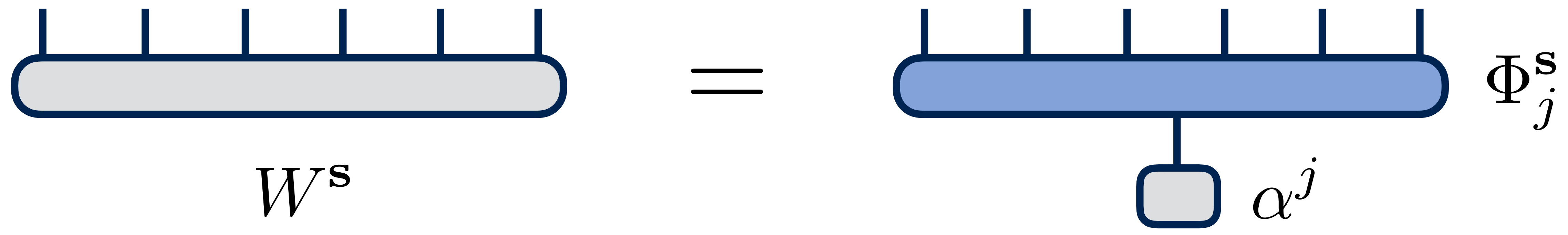


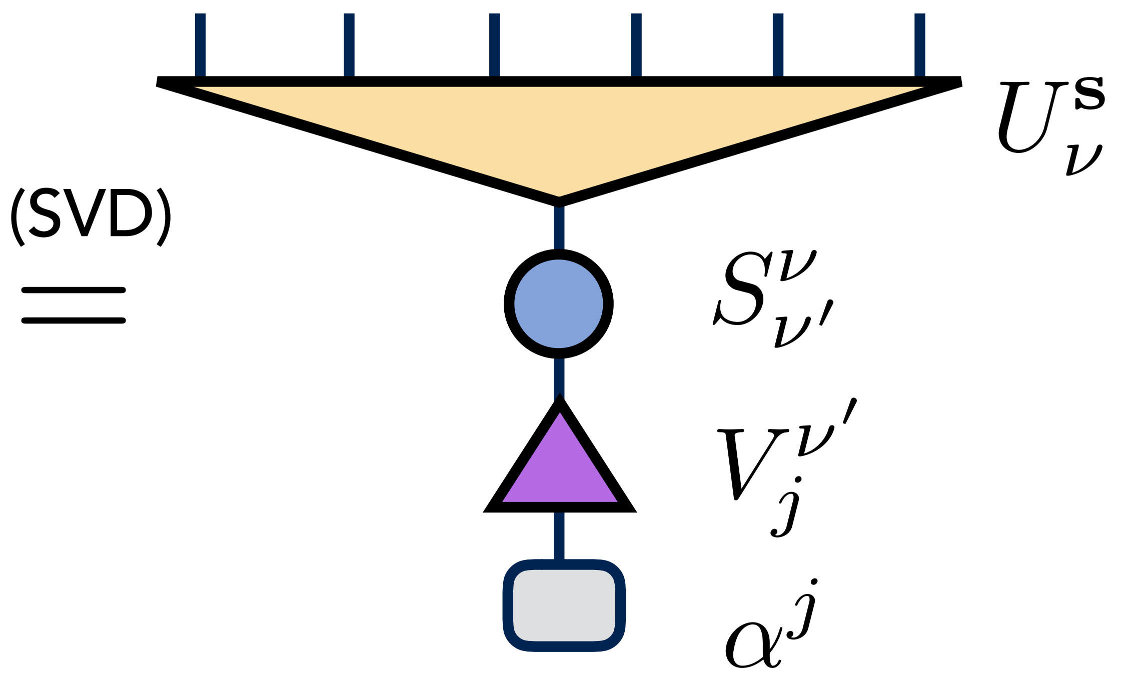
Really just says weights in the span of $\{\Phi_j^{\mathbf{s}}\}$

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

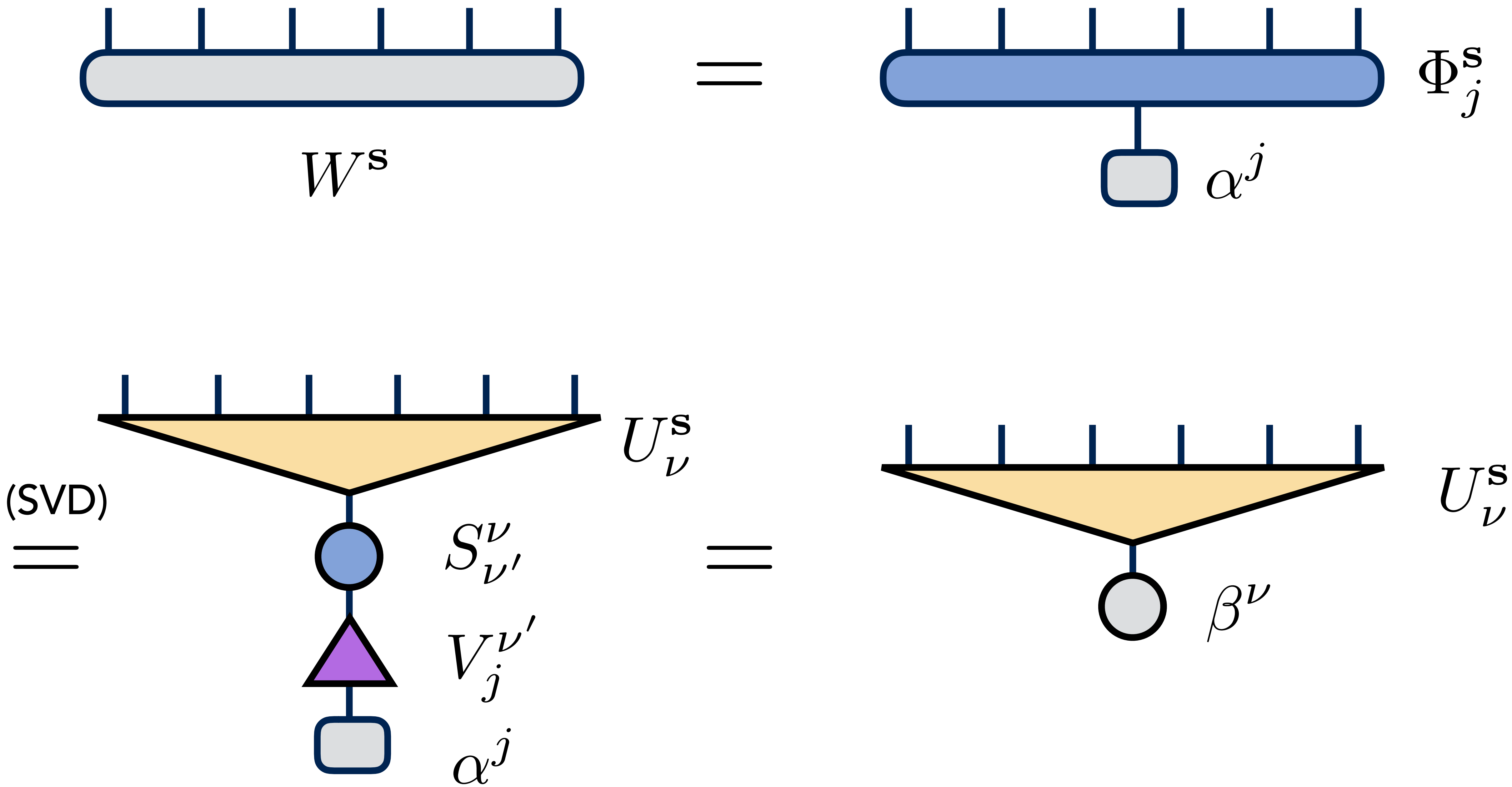


Can choose any basis for span of $\{\Phi^{\mathbf{s}}_j\}$

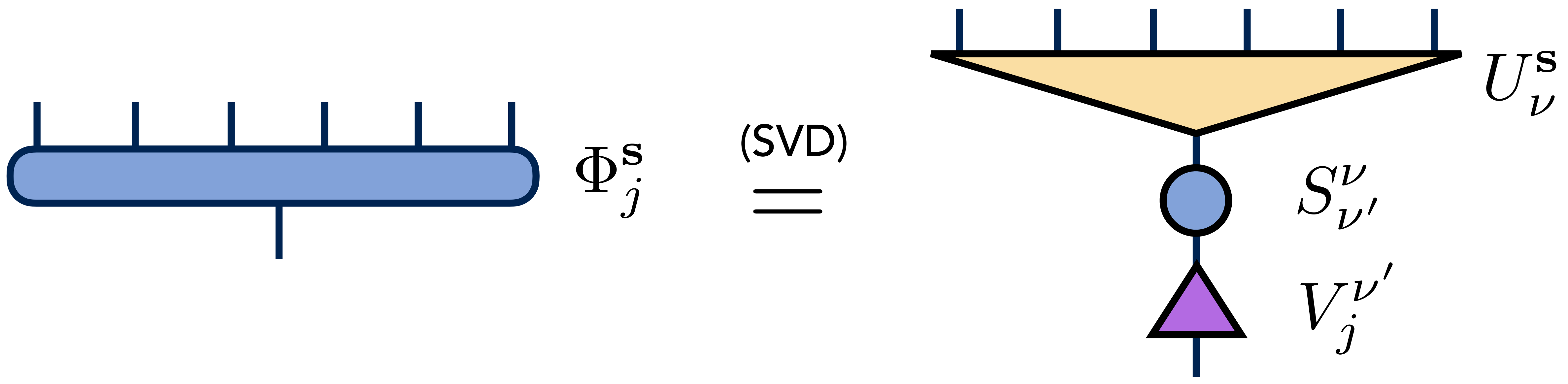




Can choose any basis for span of $\{\Phi^{\mathbf{s}}_j\}$



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



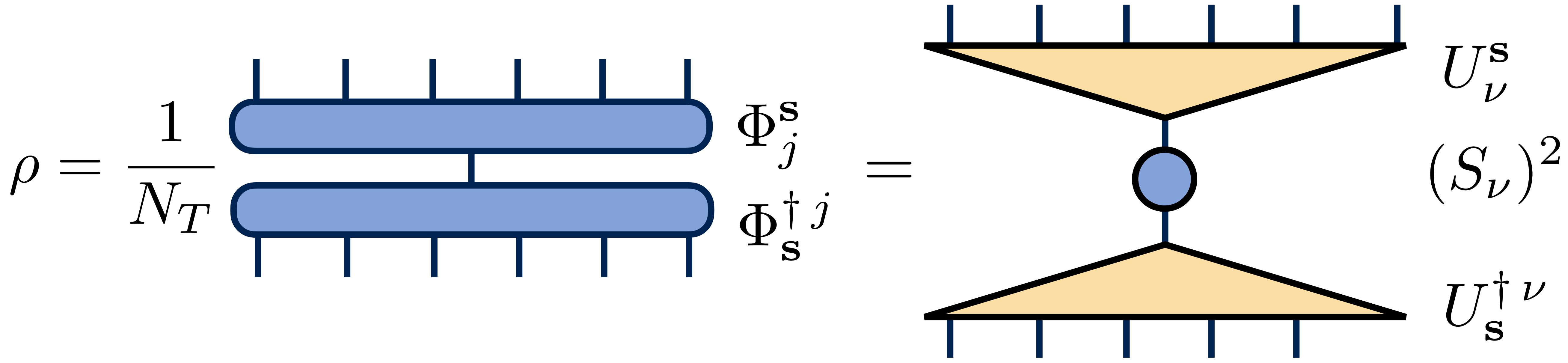
Orthonormal basis

Can discard basis vectors corresponding to small s. vals.

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

Computing $U^{\mathbf{s}}_{\nu}$ efficiently

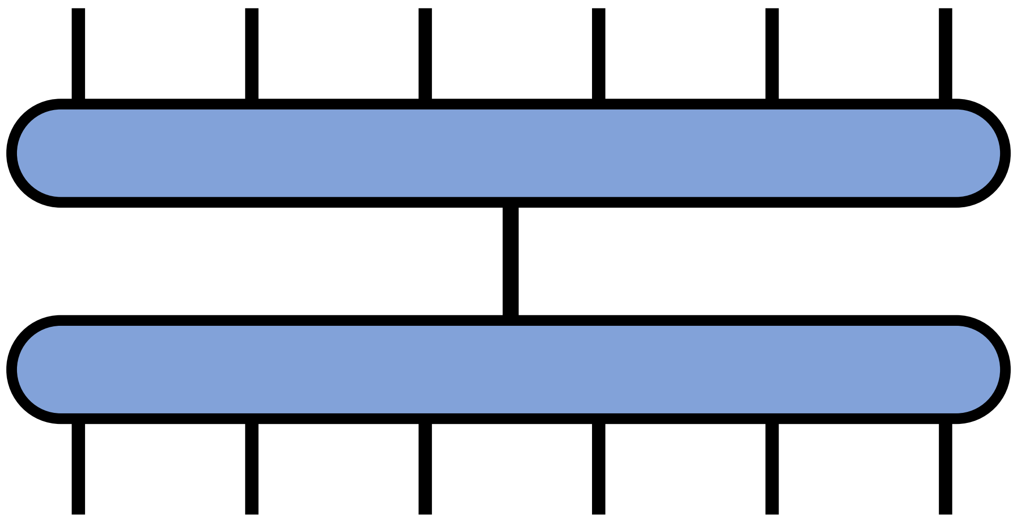
Define *feature space covariance matrix* (similar to density matrix)



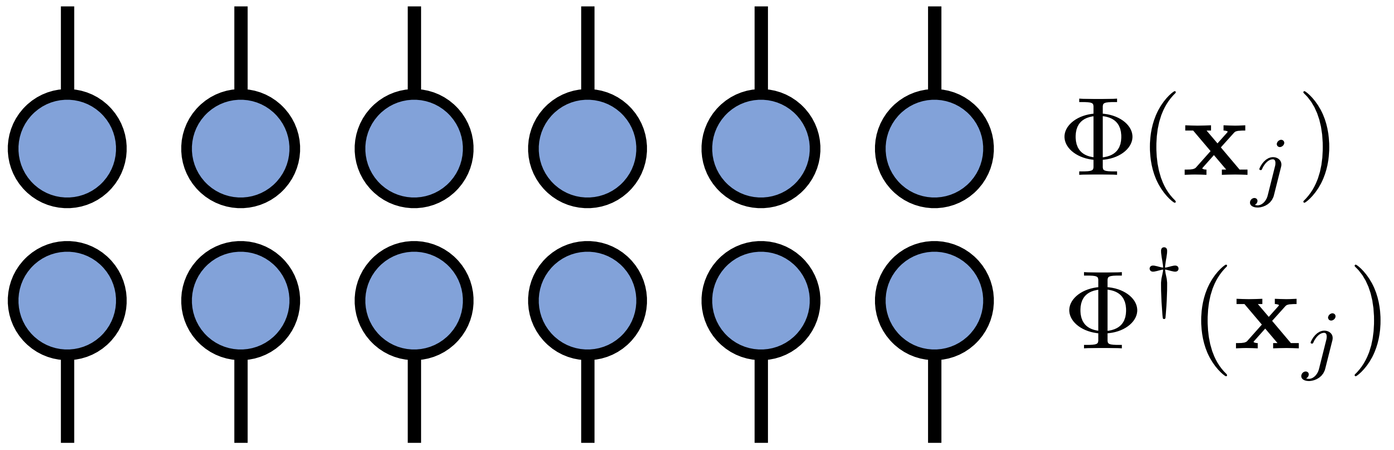
Strategy: compute $U^{\mathbf{s}}_{\nu}$ iteratively as a layered (tree) tensor network

For efficiency, exploit product structure of Φ

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

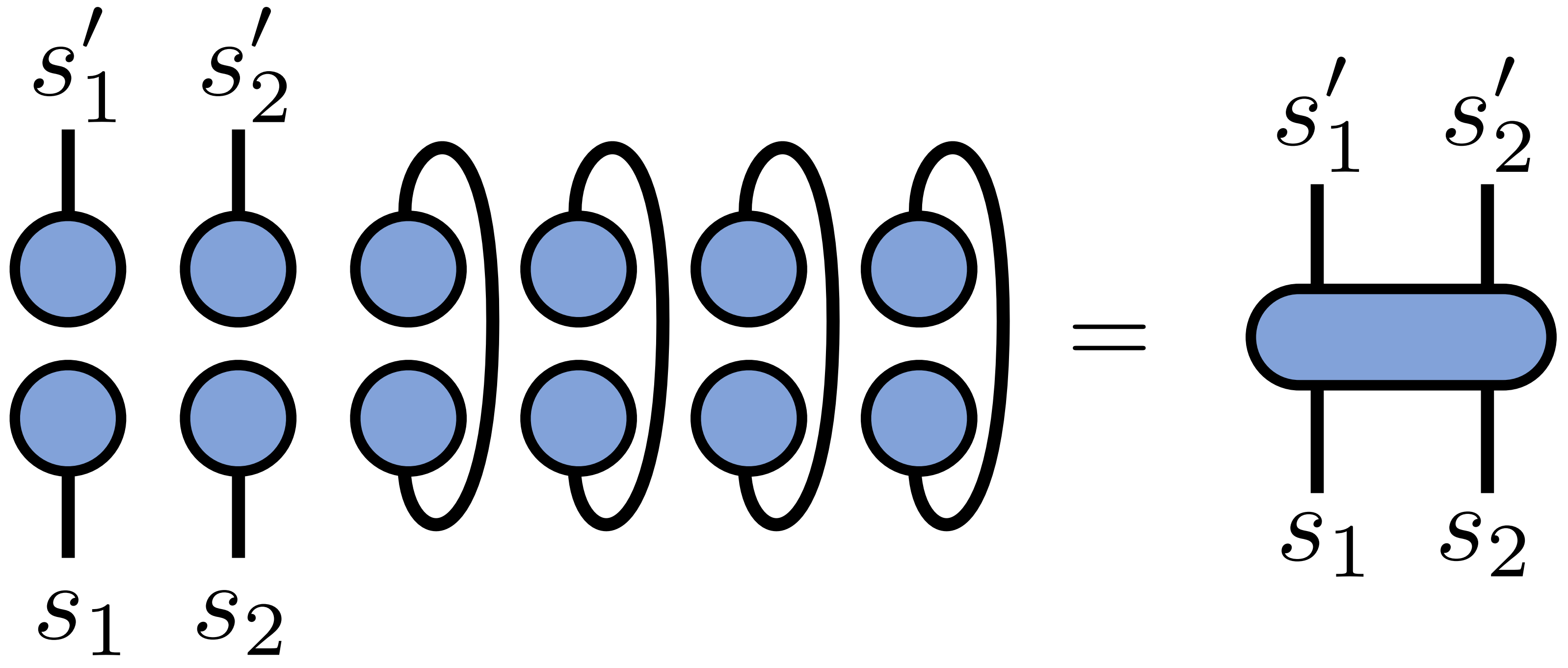


$$= \frac{1}{N_T}\sum_{j=1}^{N_T}$$

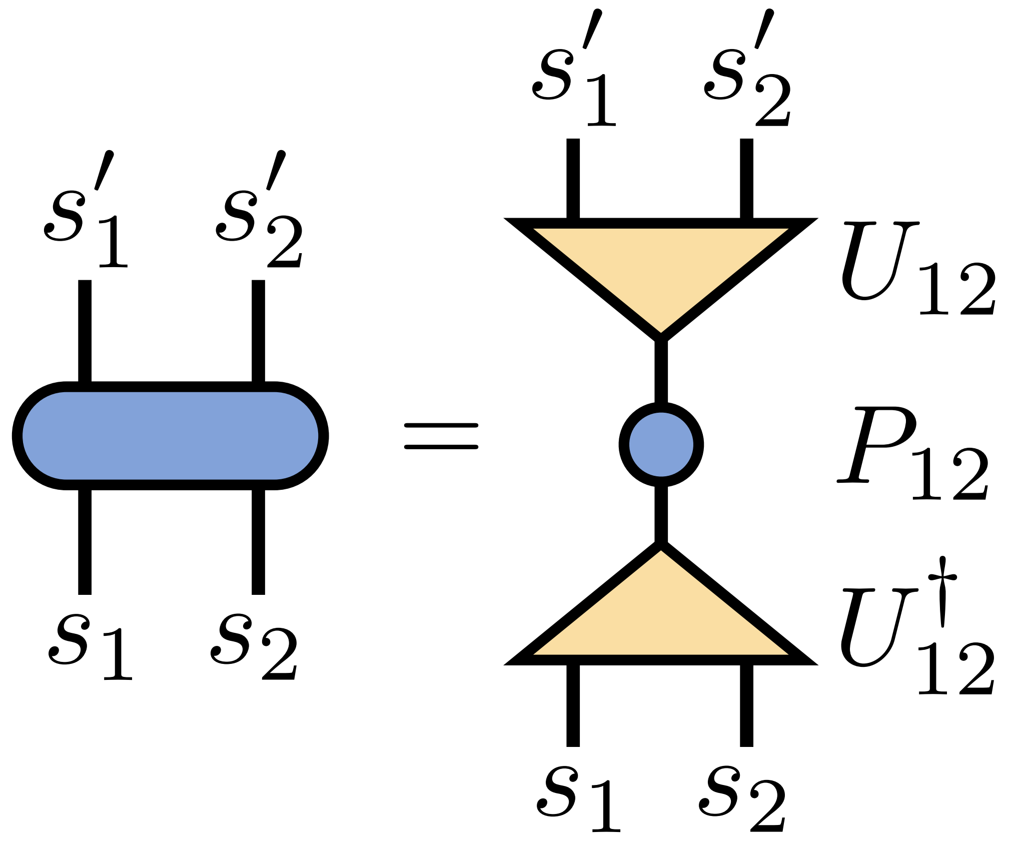


Compute tree tensors from reduced matrices

$\rho_{12} = \sum_{j \in \text{training}}$

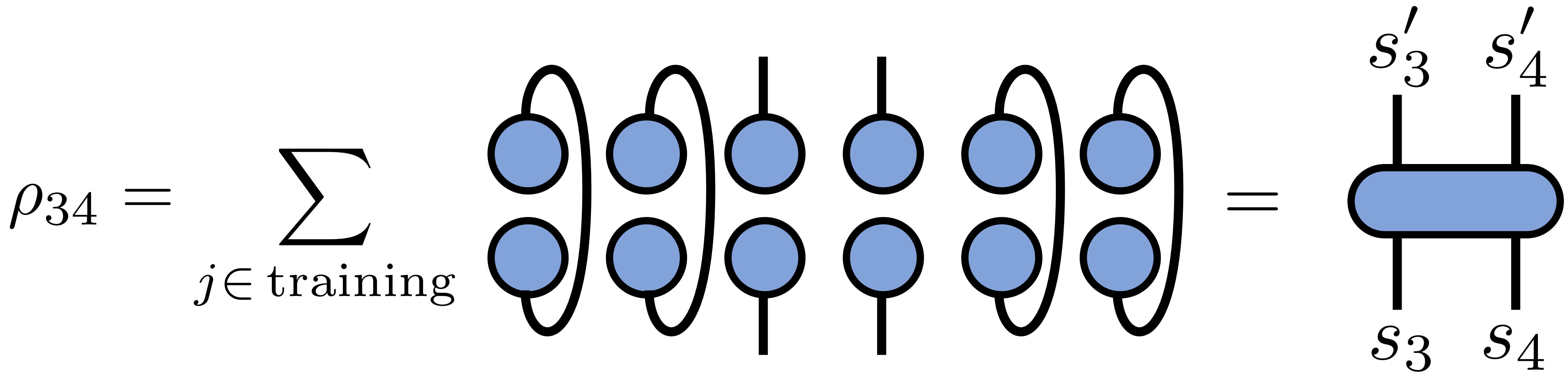


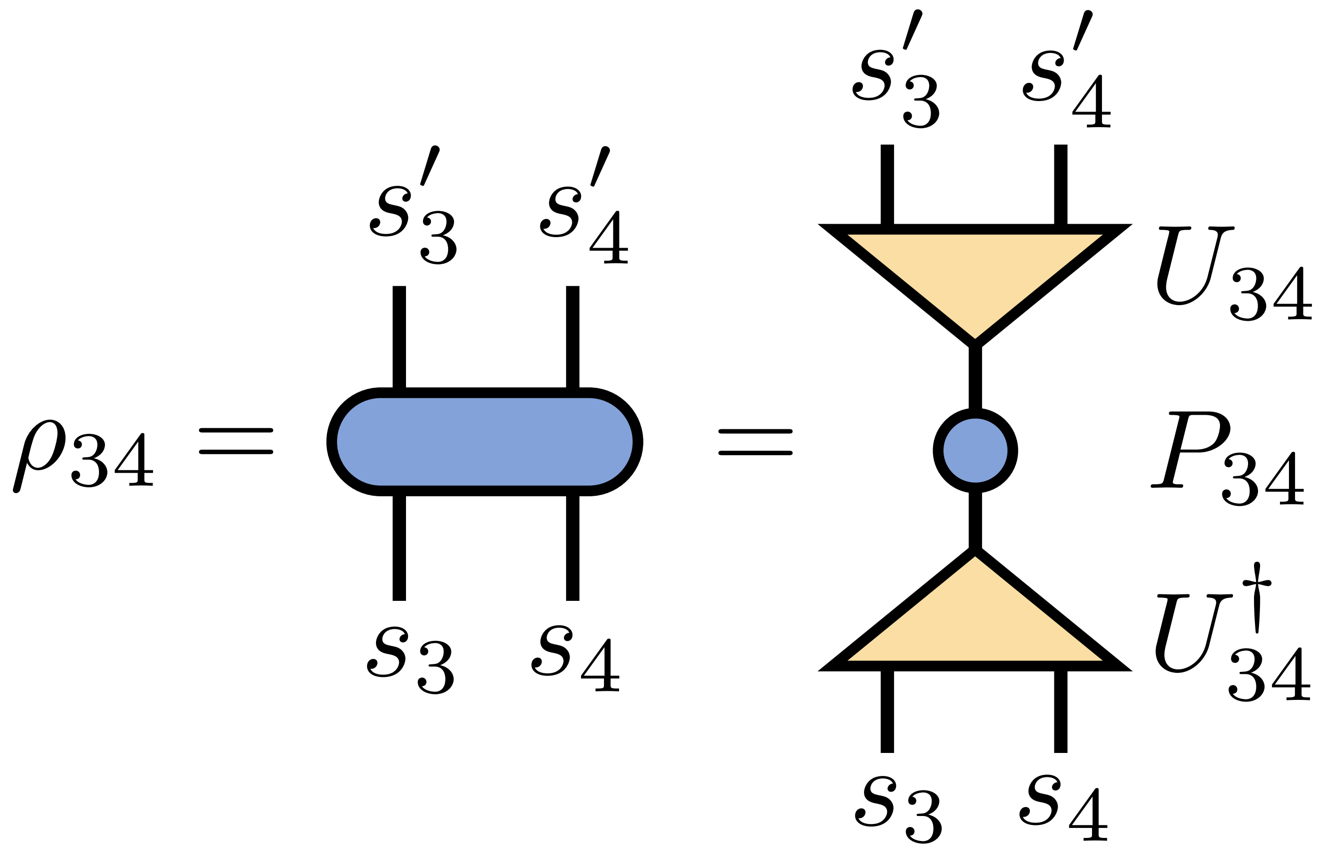
$\rho_{12} =$



Truncate small eigenvalues

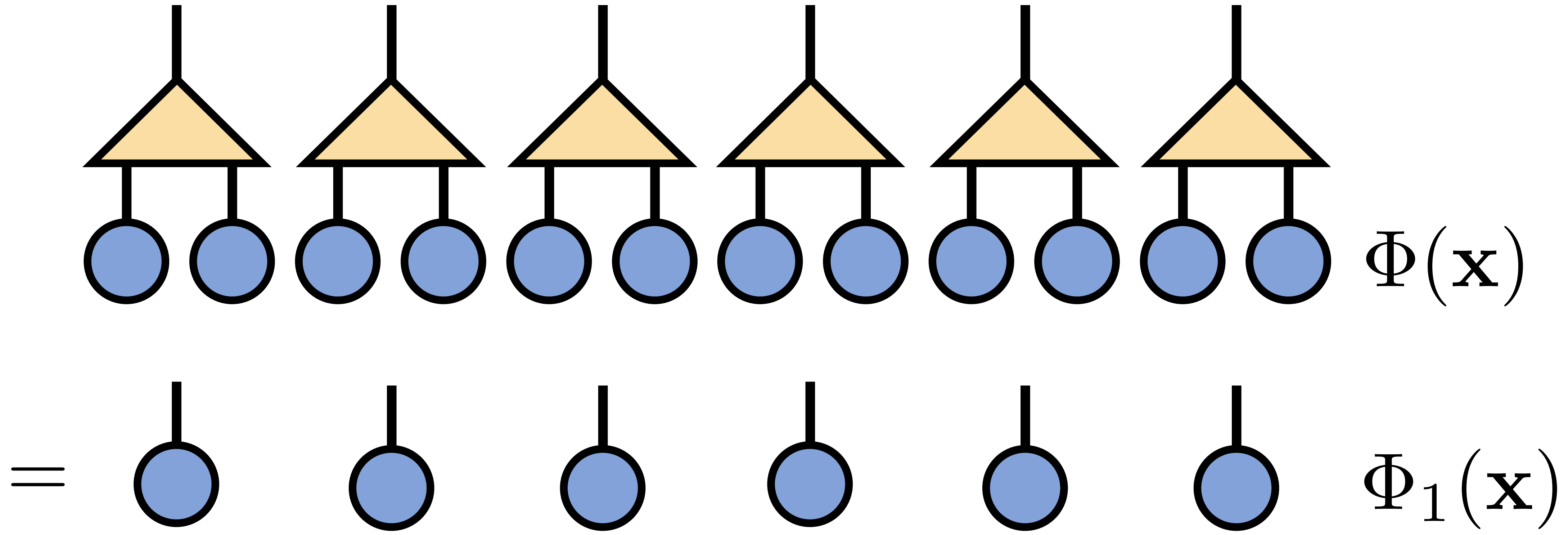
Compute tree tensors from reduced matrices



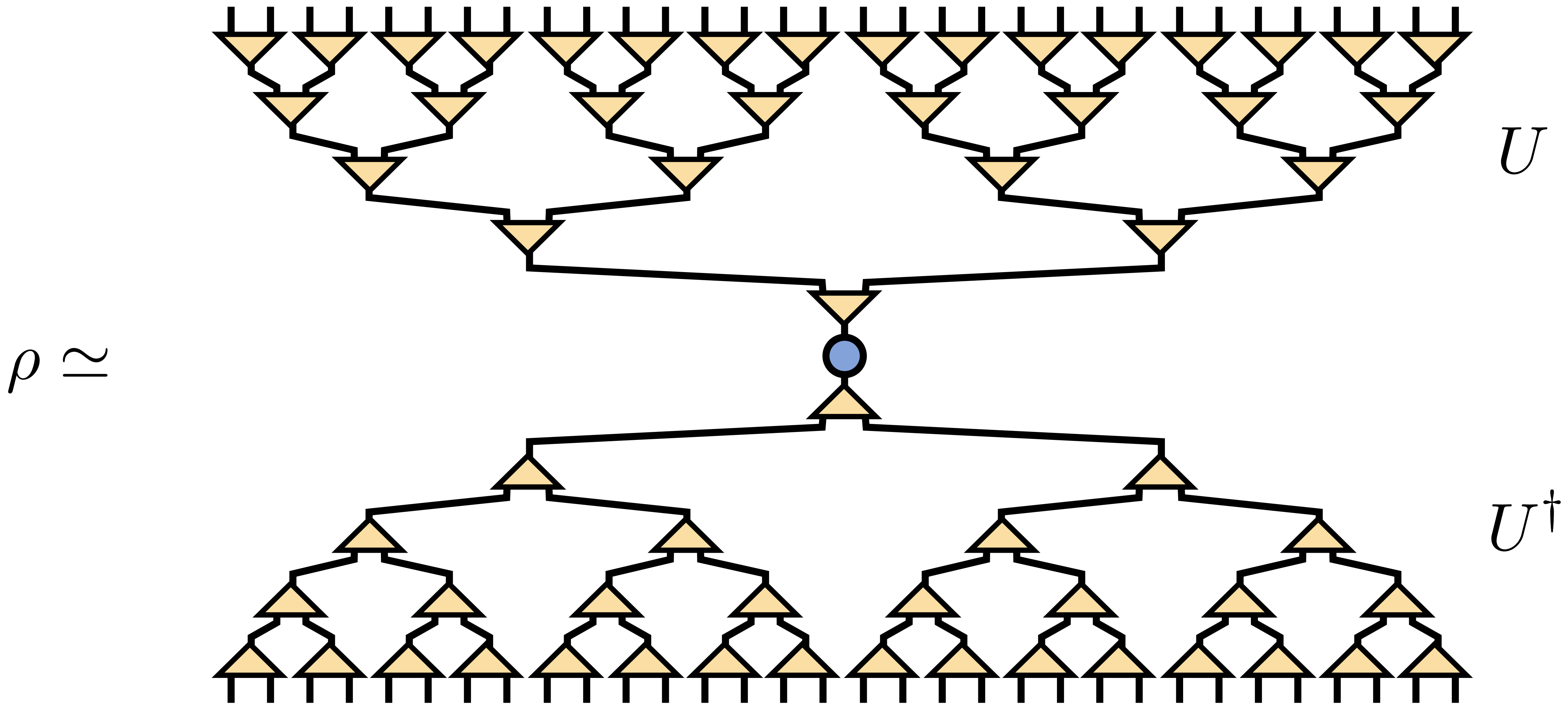


Truncate small eigenvalues

Having computed a tree layer, rescale data

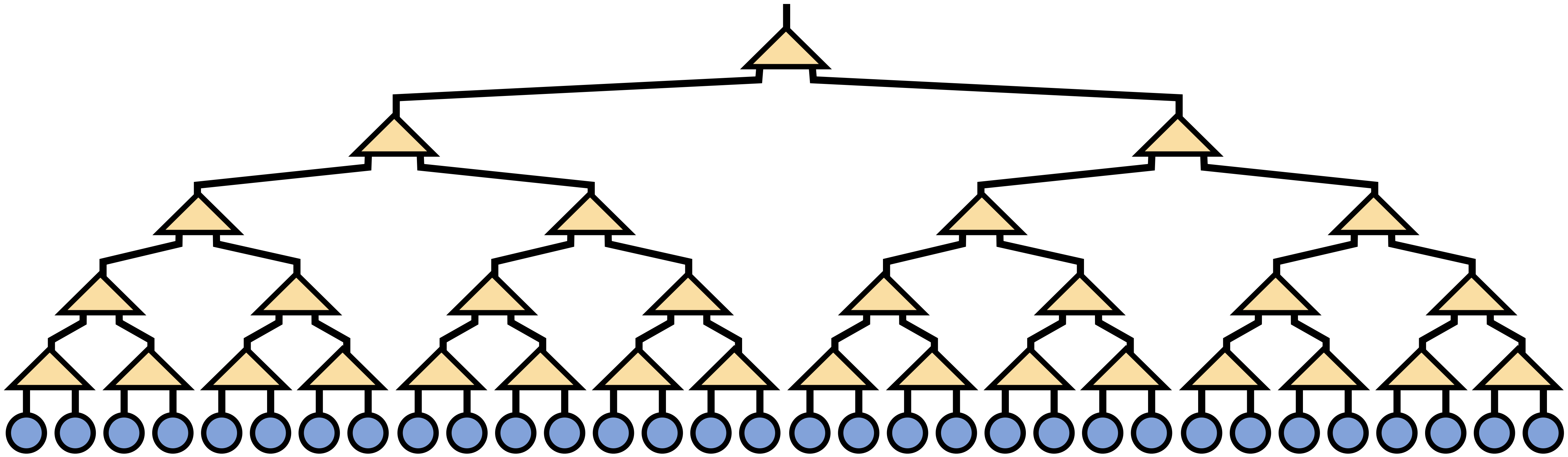


With all layers, have approximately diagonalized ρ



Equivalent to *kernel PCA*,
but linear scaling with size of data set

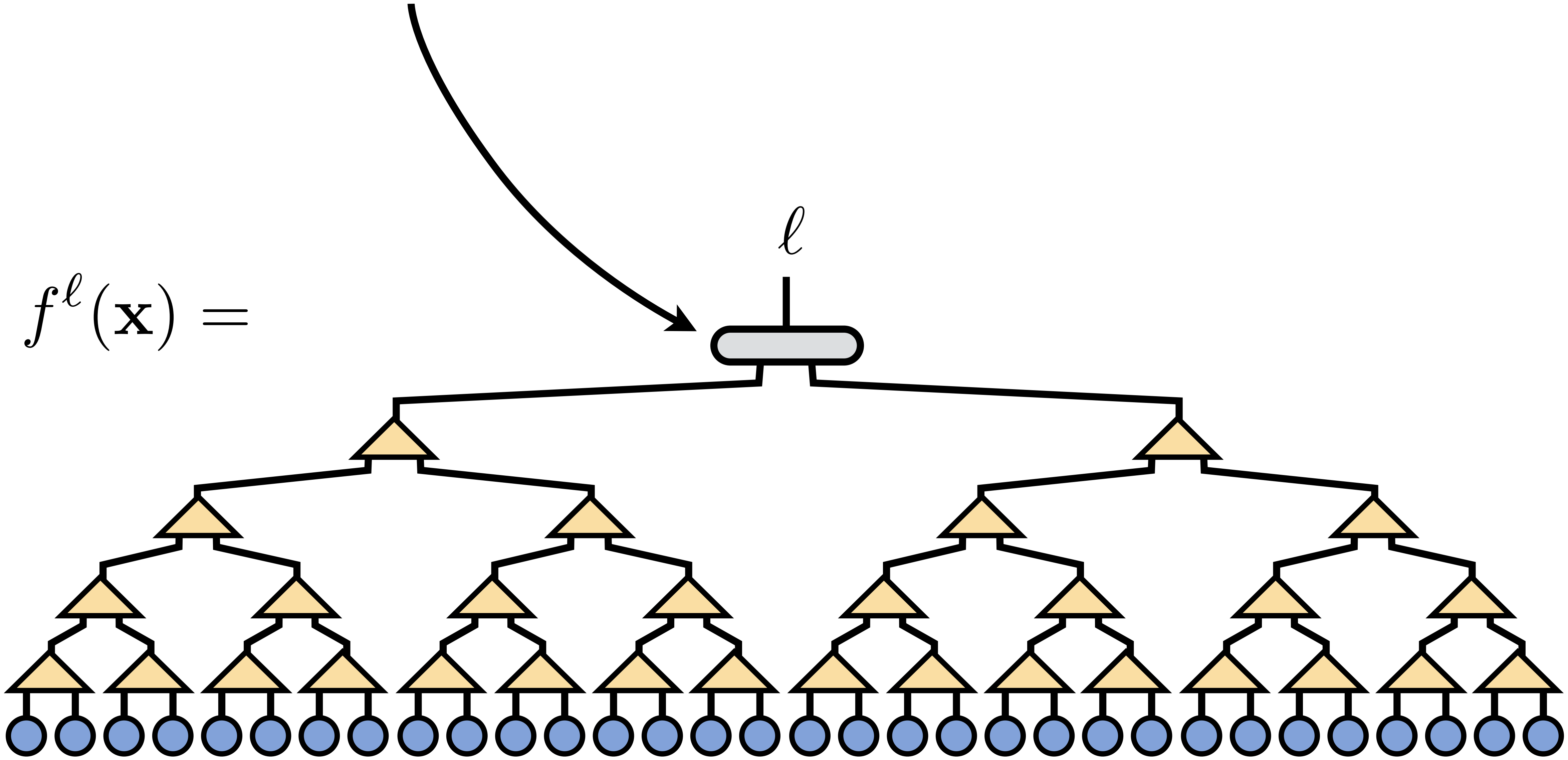
Can view as *unsupervised learning* of representation of training data



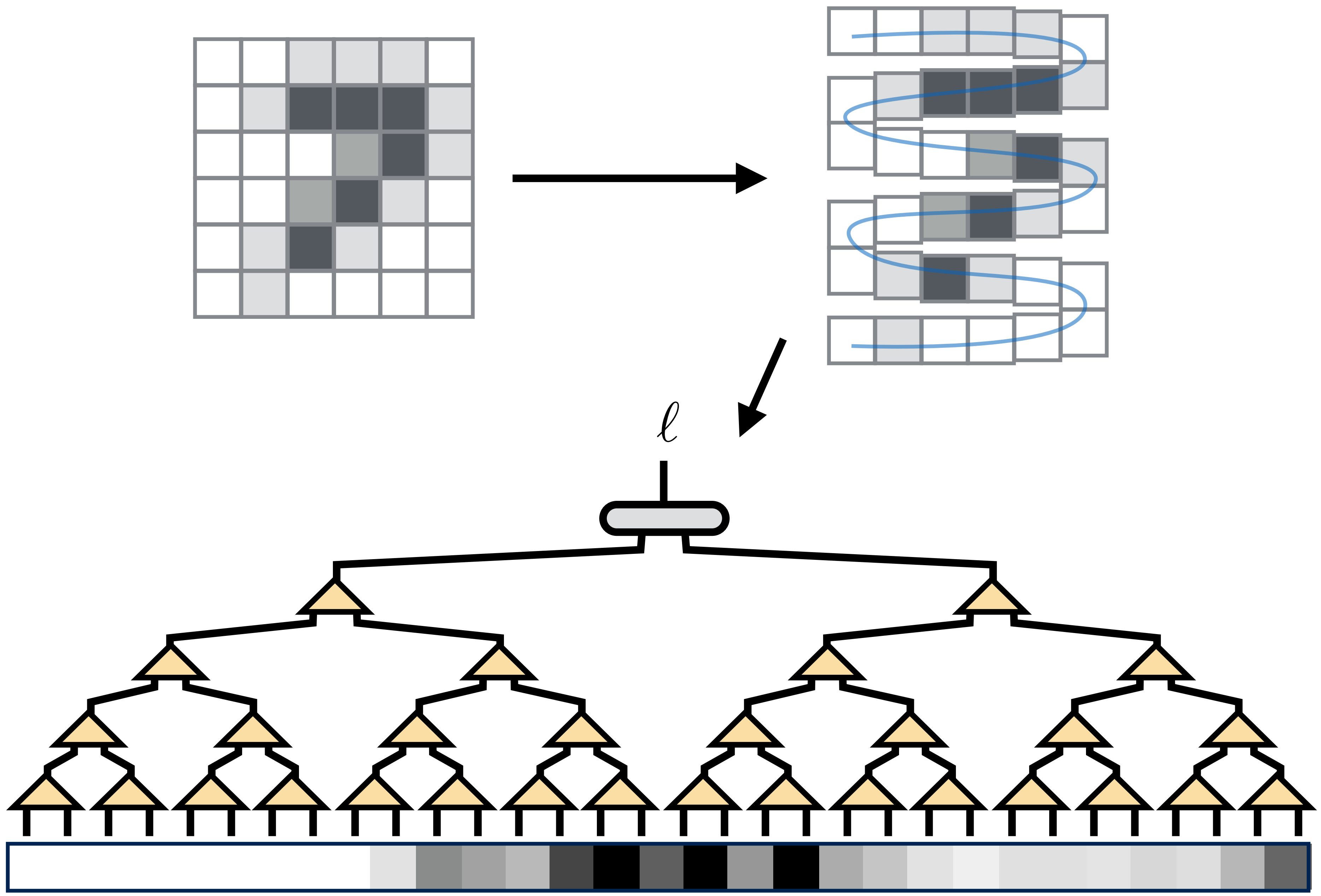
Use as starting point for supervised learning

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

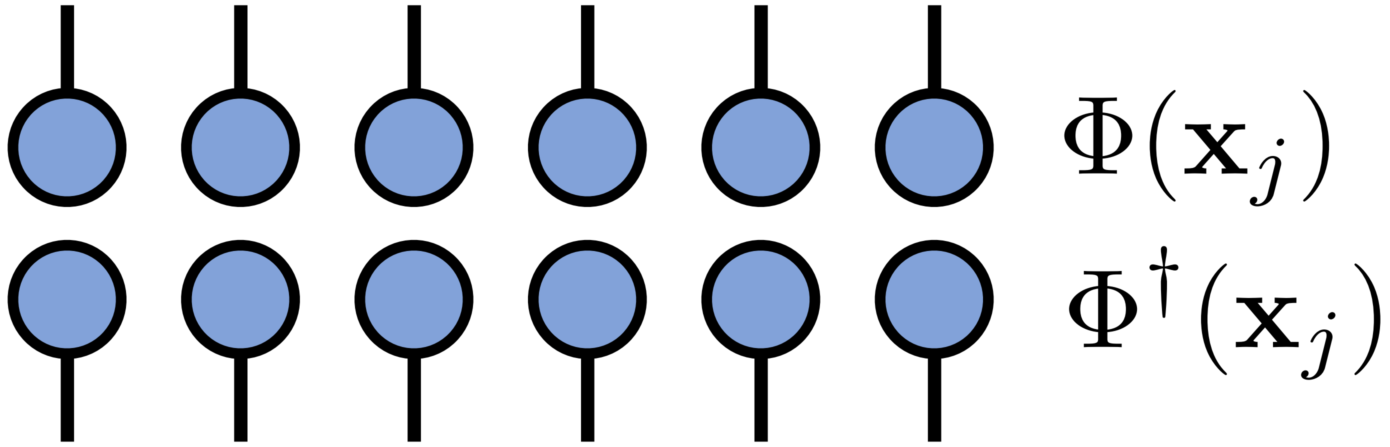
Training acc: 99.68% **Test acc:** 98.08%

Refinements and Extensions

No reason we must base tree around ρ

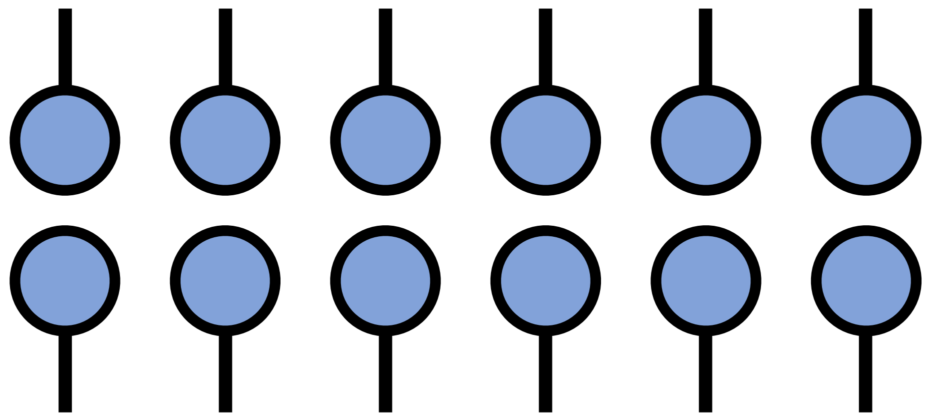
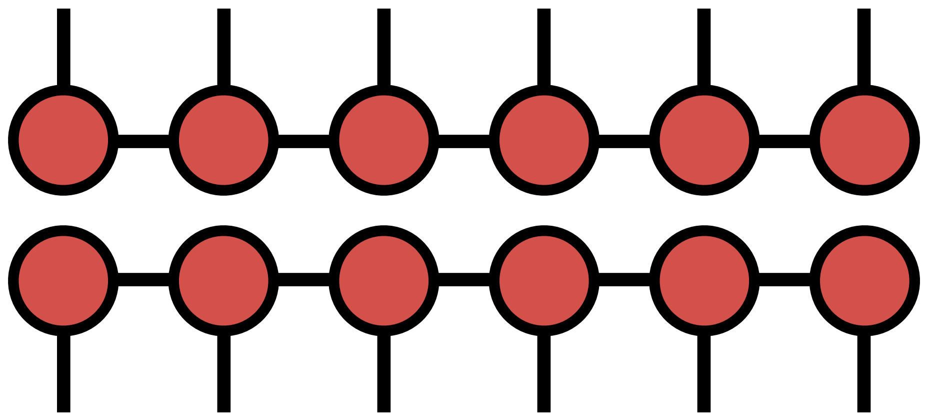
Could reweight based on importance of samples

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



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

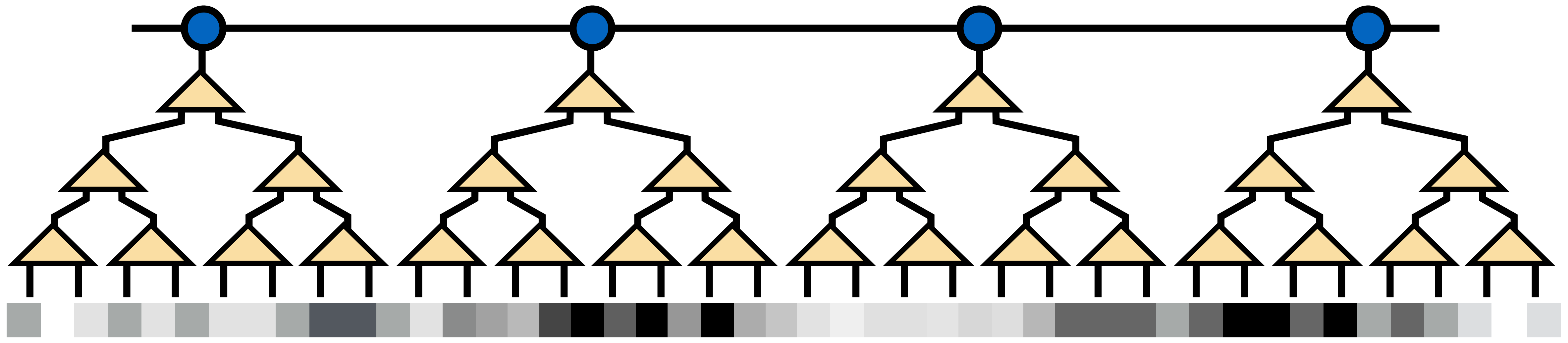
Using $\rho^\mu = (1-\mu)\rho + \mu \sum_\ell |W^\ell\rangle\langle W^\ell|$

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



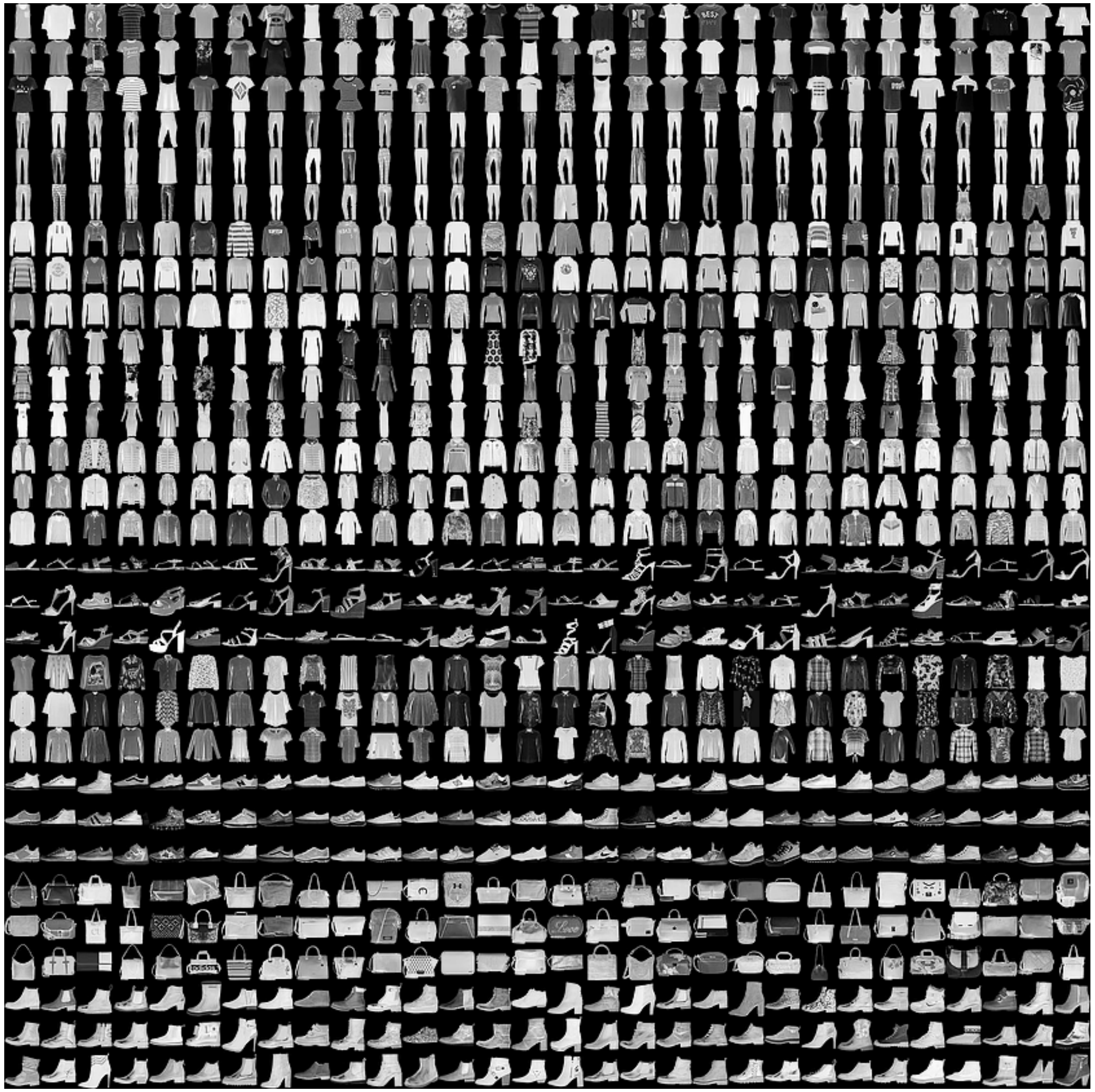
Approximate top tensor by MPS

Experiment: "fashion MNIST" dataset

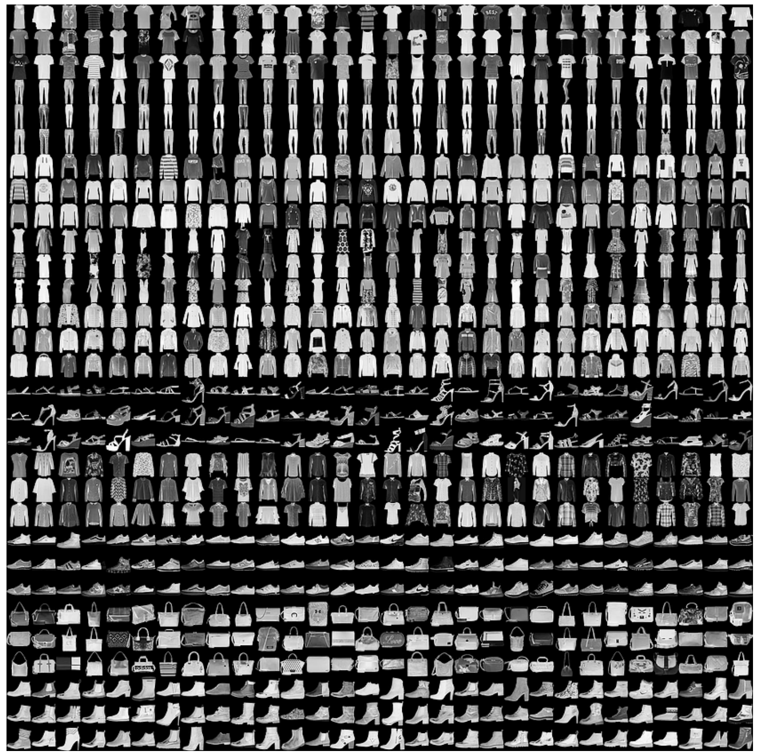
28x28 grayscale

60,000 training images

10,000 testing images

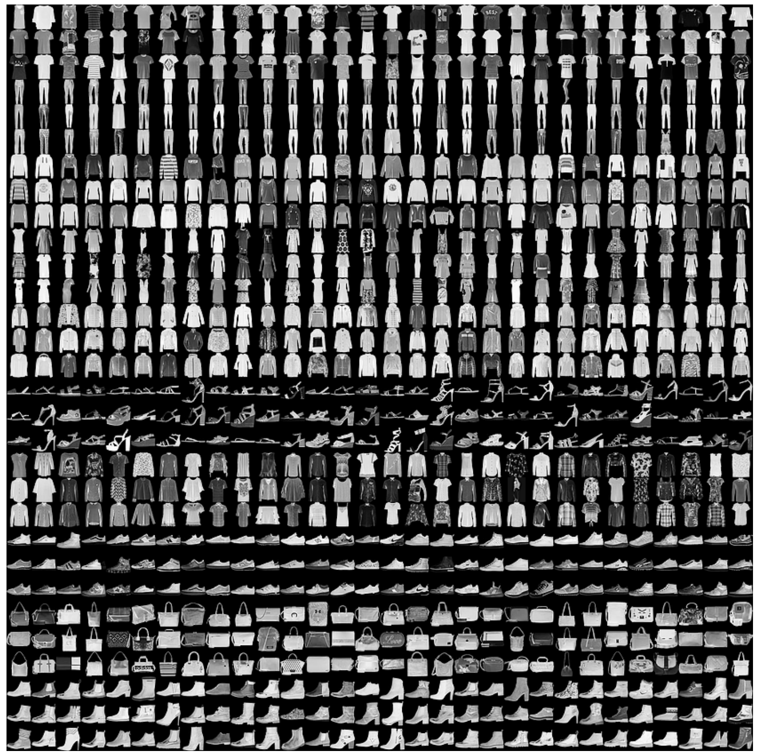


Experiment: "fashion MNIST" dataset



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

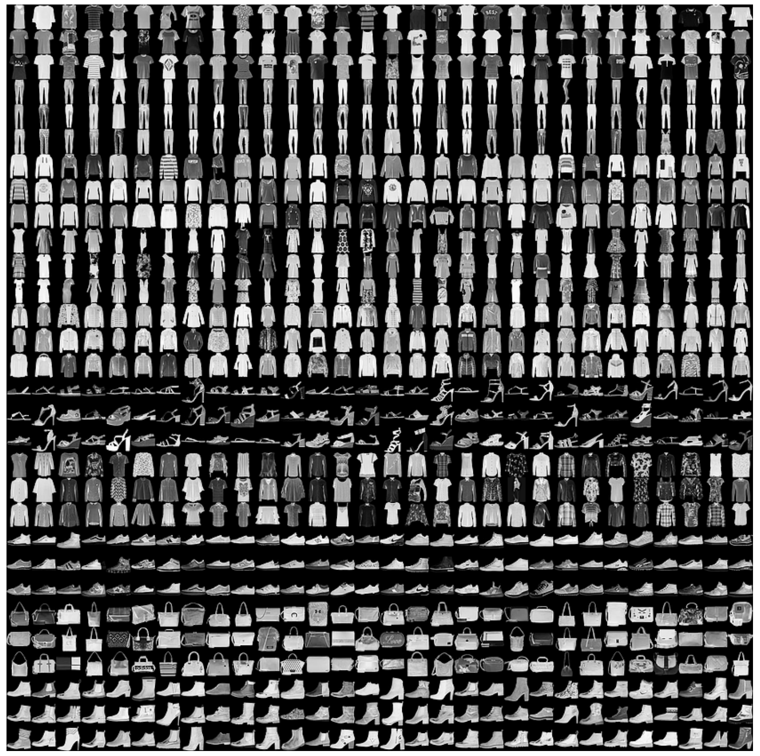
Experiment: "fashion MNIST" dataset



- 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%

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

- Models with tensor network weights have interesting capabilities

- Same models can be applied on classical or quantum hardware

- Tensor networks can be used for adaptive, unsupervised learning similar to renormalization group