Tensor Network Machine Learning

An Introduction

Bogdan Pirvu

NOVOMATIC

About Me

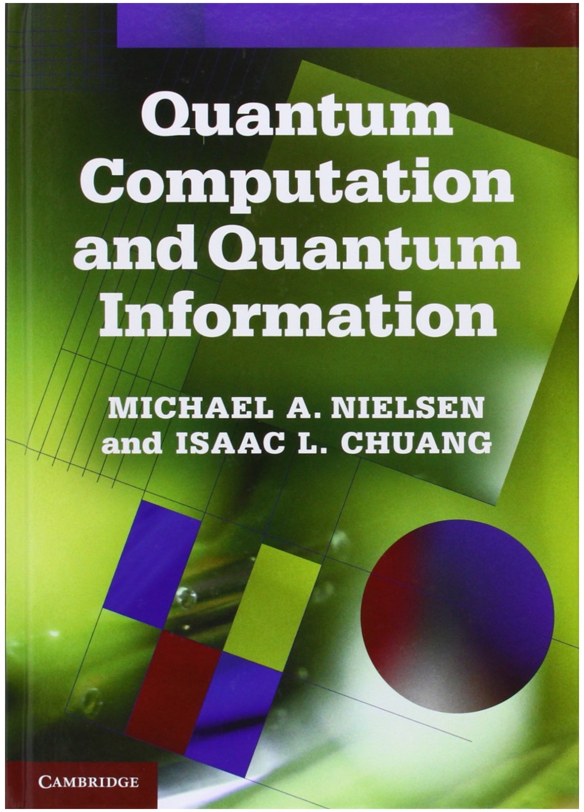
- Head of Data Science at Novomatic AG
- Lecturer at University of Applied Sciences Wiener Neustadt
- Previously
  - Researcher in Quantum Physics
  - Full Stack Software Engineer

Outline

- Quantum States
- Tensors
- Tensor Networks States
- Tensor Network Machine Learning

Quantum States

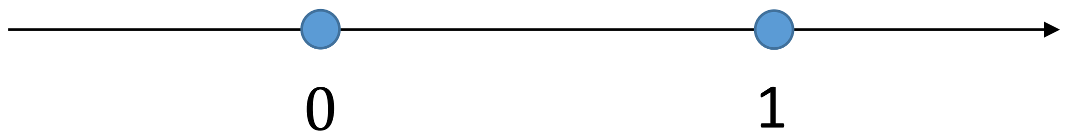
Nielsen & Chuang



https://michaelnielsen.org

Classical Bit

$x \in \{0,1\}$

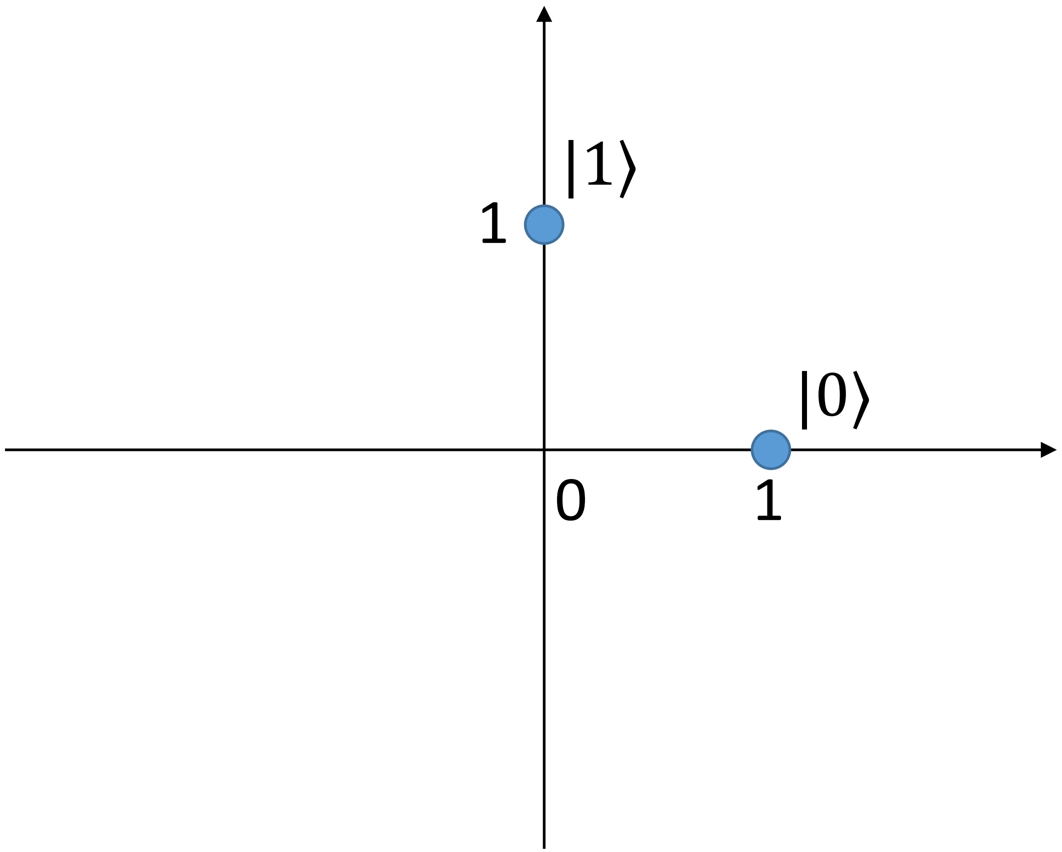


Vector Space Representation

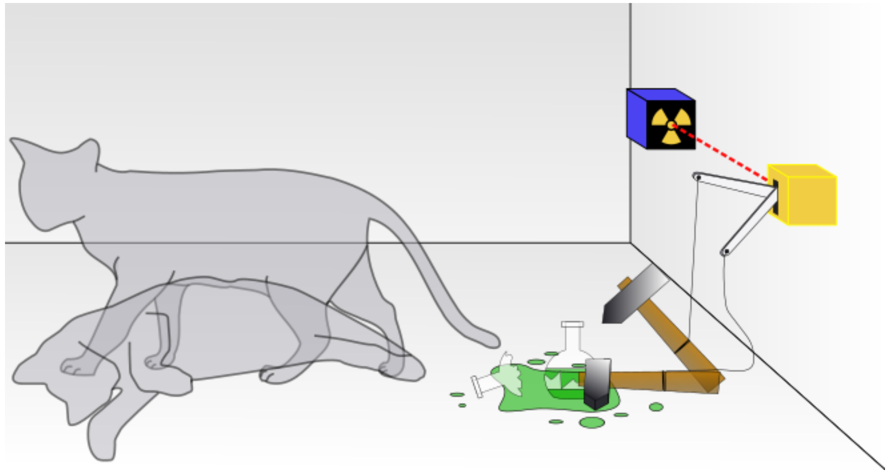
$$0 \rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \vec{e}_0 = |0\rangle$$

$$1 \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \vec{e}_1 = |1\rangle$$

$$\vec{x} \in \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\} = \{|0\rangle, |1\rangle\}$$



Quantum World – Superposition Principle



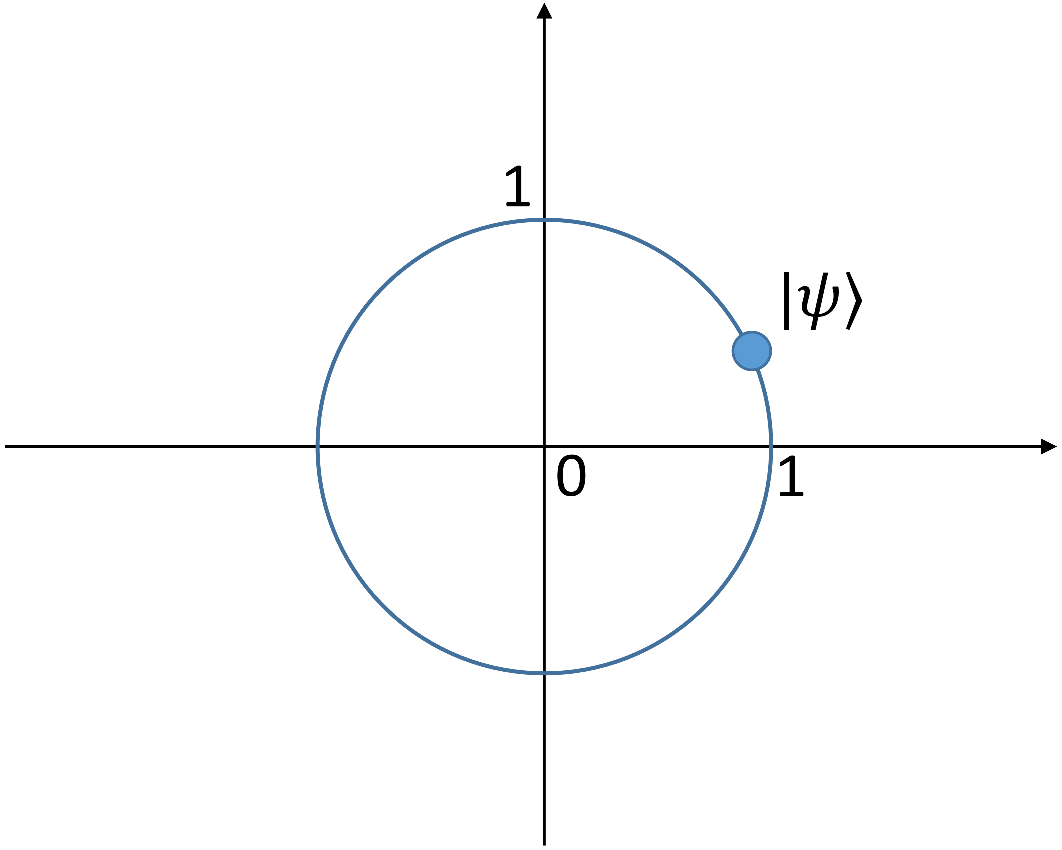
Quantum Bit - Qubit

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\psi\rangle = \alpha\begin{pmatrix}1\\0\end{pmatrix} + \beta\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}\alpha\\\beta\end{pmatrix}$$

$$\|\psi\|^2 = \langle\psi|\psi\rangle = 1$$

$$|\psi\rangle \in \left\{\sum_{i=0}^{1}\psi_i|i\rangle \;\middle|\; \|\psi\|^2 = 1\right\}$$



Two-Bit System

$\overrightarrow{x_1} \in \{|0\rangle, |1\rangle\}$

$\overrightarrow{x_2} \in \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} = \vec{e}_{00} \qquad |01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\1\\0\\0\end{pmatrix} = \vec{e}_{01}$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\end{pmatrix} = \vec{e}_{10} \qquad |11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\1\end{pmatrix} = \vec{e}_{11}$$

Two-Qubit System

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

$$|\psi\rangle = \psi_{00}|00\rangle + \psi_{01}|01\rangle + \psi_{10}|10\rangle + \psi_{11}|11\rangle = \begin{pmatrix} \psi_{00} \\ \psi_{01} \\ \psi_{10} \\ \psi_{11} \end{pmatrix}$$

$$|\psi\rangle \in \left\{ \sum_{i_1, i_2 = 0}^{1} \psi_{i_1 i_2} |i_1 i_2\rangle \;\middle|\; \|\psi\|^2 = 1 \right\}$$

N-Qubit System

$$|\psi\rangle = \sum_{i_1, i_2 \dots i_N = 0}^{1} \psi_{i_1 i_2 \dots i_N} |i_1 i_2 \dots i_N\rangle$$

Computational problem: deal with huge dimension $= 2^N$!!!

For $N \approx 130$ the number of amplitudes is greater than the number of atoms in the known universe!

Solution: represent Quantum States as Tensor Networks

Tensors

Rank 1 Tensor

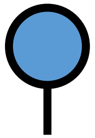
Example: 1-qubit state. Can be represented as vector:

$$|\psi\rangle = \sum_{i=0}^{1} a_i |i\rangle$$

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$a_0 = \frac{1}{\sqrt{2}}, a_1 = \frac{1}{\sqrt{2}}$$

$$a_i \equiv \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$$

$a_i \equiv$ 

i

Rank 2 Tensor

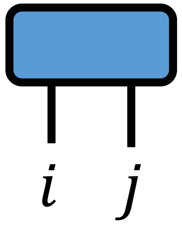
Example: 2-qubit state. Can be represented as matrix:

$$|\psi\rangle = \sum_{i,j=0}^{1} a_{ij}|ij\rangle$$

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle = \begin{pmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 1/\sqrt{2} \end{pmatrix}$$

$$a_{00} = \frac{1}{\sqrt{2}}, a_{11} = \frac{1}{\sqrt{2}}$$

$$a_{ij} \equiv \begin{pmatrix} 1/\sqrt{2} & 0 \\ 0 & 1/\sqrt{2} \end{pmatrix}$$

$a_{ij} \equiv$ 

Rank 3 Tensor

Example: 3-qubit state

$$|\psi\rangle = \sum_{i,j,k=0}^{1} a_{ijk}|ijk\rangle$$

$$|W\rangle = \frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle) = \frac{1}{\sqrt{3}}\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$a_{001} = \frac{1}{\sqrt{3}}$$

$$a_{010} = \frac{1}{\sqrt{3}}$$

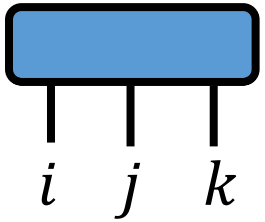
$$a_{100} = \frac{1}{\sqrt{3}}$$

Rank 3 Tensor

Can be represented as an array of matrices:

$$a_{ijk} \equiv \left[\begin{pmatrix} a_{000} & a_{001} \\ a_{010} & a_{011} \end{pmatrix}, \begin{pmatrix} a_{100} & a_{101} \\ a_{110} & a_{111} \end{pmatrix}\right] = \frac{1}{\sqrt{3}}\left[\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\right]$$

```
In [3]: np.array([0,1,1,0,1,0,0,0]).reshape((2, 2, 2))

Out[3]: array([[[0, 1],
                [1, 0]],

               [[1, 0],
                [0, 0]]])
```

$a_{ijk} \equiv$ 

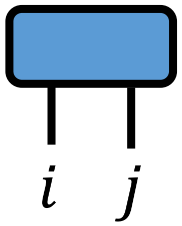
Tensor Contraction

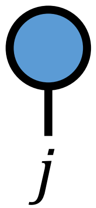
Rank-2-Tensor-Vector Contraction

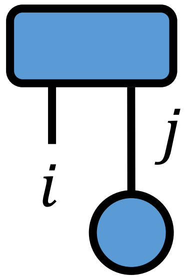
Is just a Matrix-Vector dot product:

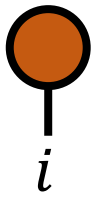
$$\vec{c} = A\vec{b} \qquad A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \qquad \vec{c} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

$$c_i = \sum_{j=0}^{1} a_{ij} b_j \qquad \vec{c} = \begin{pmatrix} a_{11}b_1 + a_{12}b_2 \\ a_{21}b_1 + a_{22}b_2 \end{pmatrix}$$

$a_{ij} \equiv$ 

$b_j \equiv$ 

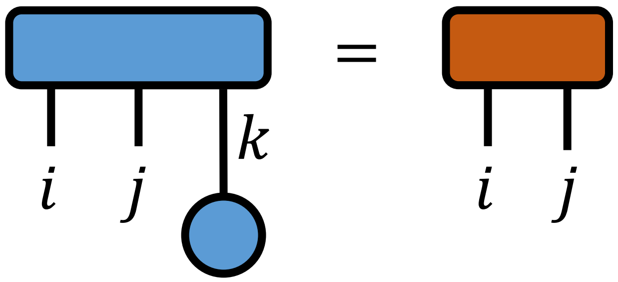
$c_i \equiv$ 

$=$ 

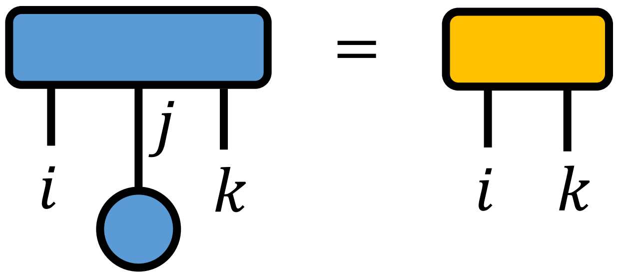
Rank-3-Tensor-Vector Contraction

Examples:

$$c_{ij} = \sum_{k=0}^{1} a_{ijk} b_k$$



$$d_{ij} = \sum_{j=0}^{1} a_{ijk} b_j$$

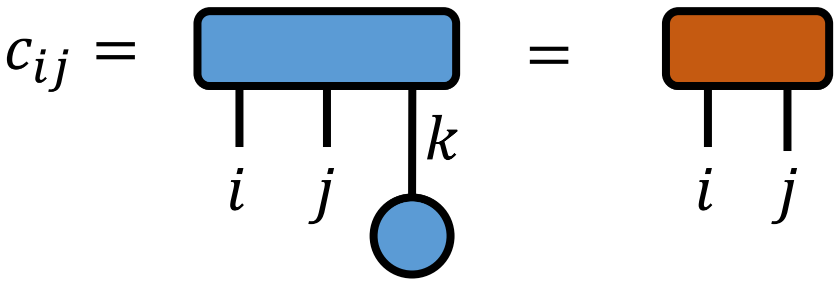


Example

$$A = \left(\begin{pmatrix} a_{111} & a_{112} \\ a_{121} & a_{122} \end{pmatrix}, \begin{pmatrix} a_{211} & a_{212} \\ a_{221} & a_{222} \end{pmatrix} \right) = \left(\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \right) \qquad \vec{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

```
In [113]: A=np.array(range(1,9)).reshape((2, 2, 2)) # i,j,k
          print(A)

[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]

In [115]: b=np.array([1,2]).reshape((2,1))          # k
          print(b)

[[1]
 [2]]
```

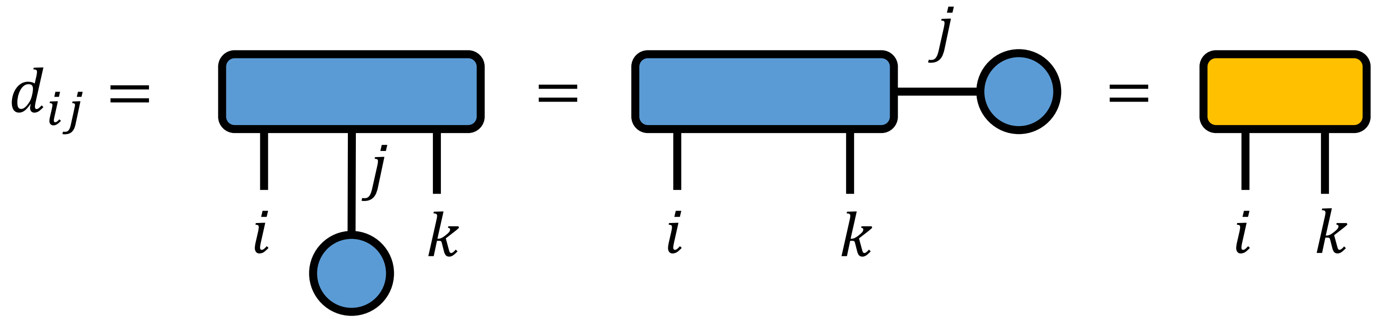
… continued




```
In [118]: C=np.tensordot(A,b, axes=([2],[0]))        # i,j
          print(C)

[[ 5 11]
 [17 23]]
```

```
In [122]: C_mat_mul = (A.reshape(4,2)@b).reshape(2,2)
          print(C_mat_mul)

[[ 5 11]
 [17 23]]
```

… continued




```
In [119]: D=np.tensordot(A,b, axes=([1],[0]))        # i,j
          print(D)

[[ 7 10]
 [19 22]]
```

```
In [123]: D_mat_mul = (A.transpose(0,2,1).reshape(4,2)@b).reshape(2,2)
          print(D_mat_mul)

[[ 7 10]
 [19 22]]
```

Tensor Network States

Many Particle Quantum States

$$|\psi\rangle = \sum_{i_1,i_2\dots i_N=0}^{d-1} a_{i_1 i_2 \dots i_N} |i_1 i_2 \dots i_N\rangle$$

Computational problem: deal with huge dimension $= 2^N$!!!

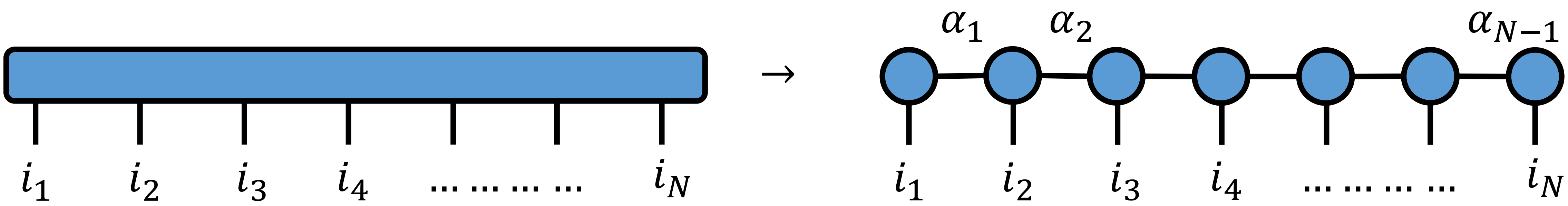
For $N \approx 130$ the number of amplitudes is greater than the number of atoms in the known universe!

Solution: represent Quantum States as a Matrix Product State

Matrix Product States (MPS)

Rewrite amplitude as a product of tensors with 3 indices:

$$a_{i_1 i_2 \ldots i_N} \rightarrow \sum_{\alpha_1,\alpha_2,\ldots,\alpha_{N-1}=1}^{D} A^{[1]}_{i_1,\alpha_1} A^{[2]}_{i_2,\alpha_1\alpha_2} A^{[3]}_{i_3,\alpha_2\alpha_3} \cdots A^{[N]}_{i_N\,\alpha_{N-1}} = A^{[1]}_{i_1} \cdot A^{[2]}_{i_2} \cdot A^{[3]}_{i_3} \cdots A^{[N]}_{i_N}$$



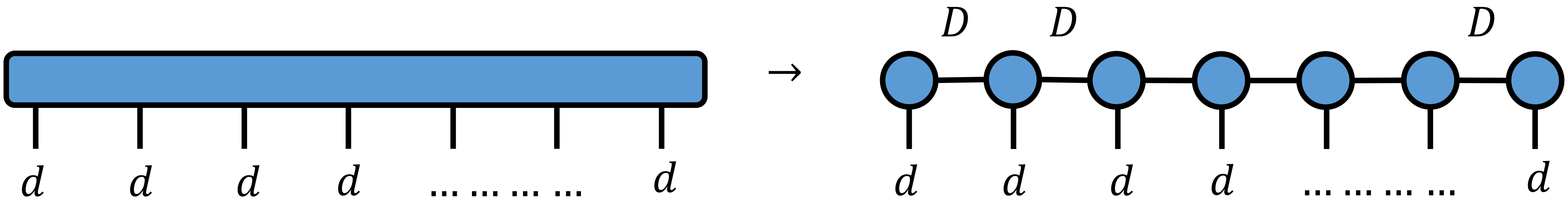
Why?

The MPS representation can be significantly more efficient:

$$\underbrace{a_{i_1 i_2 \dots i_N}}_{= d^N} \to \underbrace{A^{[1]}_{i_1} \cdot A^{[2]}_{i_2} \cdot A^{[3]}_{i_3} \cdots A^{[N]}_{i_N}}_{\approx NdD^2}$$

Exponential scaling

Linear scaling

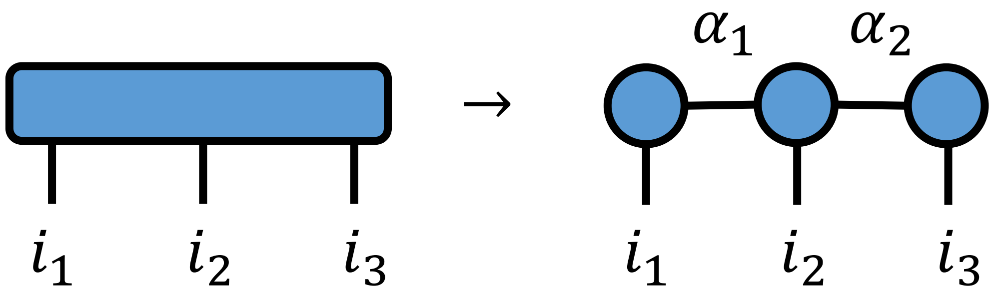


Computational complexity: $\mathcal{O}(NdD^3)$

The W state as MPS

$$|W\rangle = \frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle) = \frac{1}{\sqrt{3}} \sum_{i_1,i_2,i_3=0}^{1} a_{i_1 i_2 i_3} |i_1 i_2 i_3\rangle$$

$$a_{i_1 i_2 i_3} \rightarrow \sum_{\alpha_1,\alpha_2,\alpha_3=1}^{D} A^{[1]}_{i_1,\alpha_1} A^{[2]}_{i_2,\alpha_1 \alpha_2} A^{[3]}_{i_3,\alpha_2}$$



… continued

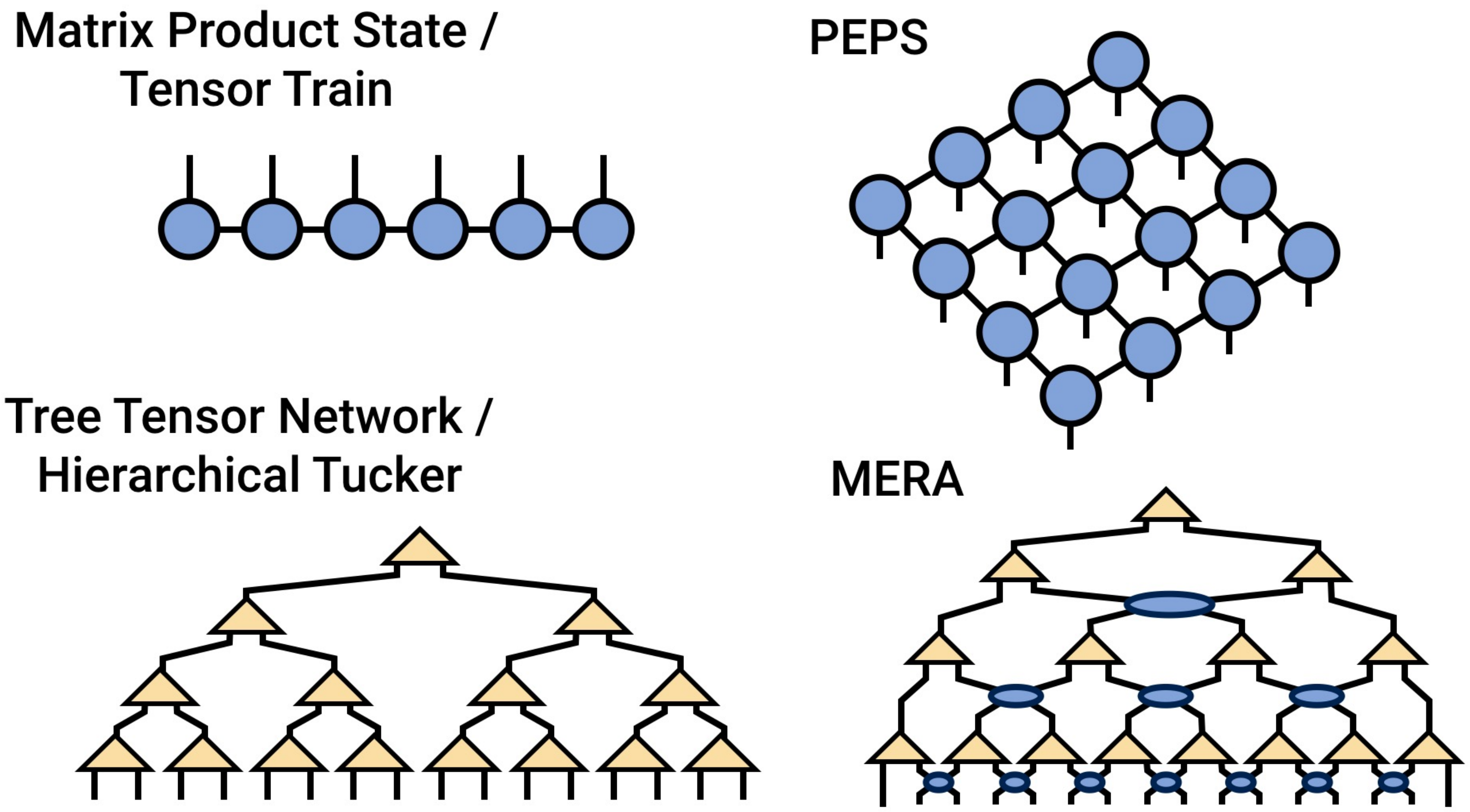
```
In [125]: A1=np.array([1,0,0,1]).reshape((2, 2)) # [d_1, D_1]
          print(A1)

[[1 0]
 [0 1]]
```

```
In [126]: A2=np.array([1,0,0,1,0,1,0,0]).reshape((2, 2, 2)) # [d_2, D_1, D_2]
          print(A2)

[[[1 0]
  [0 1]]

 [[0 1]
  [0 0]]]
```

```
In [127]: A3=np.array([0,1,1,0]).reshape((2, 2)) # [d_3, D_3]
          print(A3)

[[0 1]
 [1 0]]
```

```
In [133]: A12  = np.tensordot(A1,A2,  axes=([1],[1]))  # [d1, d2, D2]
          A123 = np.tensordot(A12,A3, axes=([2],[1]))  # [d1, d2, d3]
          print(A123.reshape(8))

[0 1 1 0 1 0 0 0]
```

Generalizations



Verstraete, F. and J. Cirac. “Renormalization algorithms for Quantum-Many Body Systems in two and higher dimensions.” *arXiv: Strongly Correlated Electrons* (2004)
Vidal, G. (2008). Class of quantum many-body states that can be efficiently simulated. *Physical review letters, 101 11*, 110501 .

Tensor Network Machine Learning

Stoudenmire, E.M., & Schwab, D. (2016). Supervised Learning with Tensor Networks. *NIPS*.
Novikov, A., Trofimov, M., & Oseledets, I. (2016). Tensor Train polynomial models via Riemannian optimization. *ArXiv, abs/1605.03795*.
ICTP 2018 Lecture: https://www.youtube.com/watch?v=kzbVHT36aLE
ICTP 2018 Slides: http://indico.ictp.it/event/8331/session/41/contribution/159/material/slides/0.pdf

Linear Regression Problem

Training set:

$$\{(\vec{x}^{(1)}, y^{(1)}), (\vec{x}^{(2)}, y^{(2)}), \ldots (\vec{x}^{(m)}, y^{(m)})\} \qquad \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} \qquad y \in \mathbb{R}$$

Model: $f_W(\vec{x}^{(i)}) = \bar{y}^{(i)}$

Train by minimizing the cost function with respect to parameters W:

$$C(W) = \frac{1}{2m}\sum_{i=1}^{m}\left(\bar{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{2m}\sum_{i=1}^{m}\left(f_W(\vec{x}^{(i)}) - y^{(i)}\right)^2$$

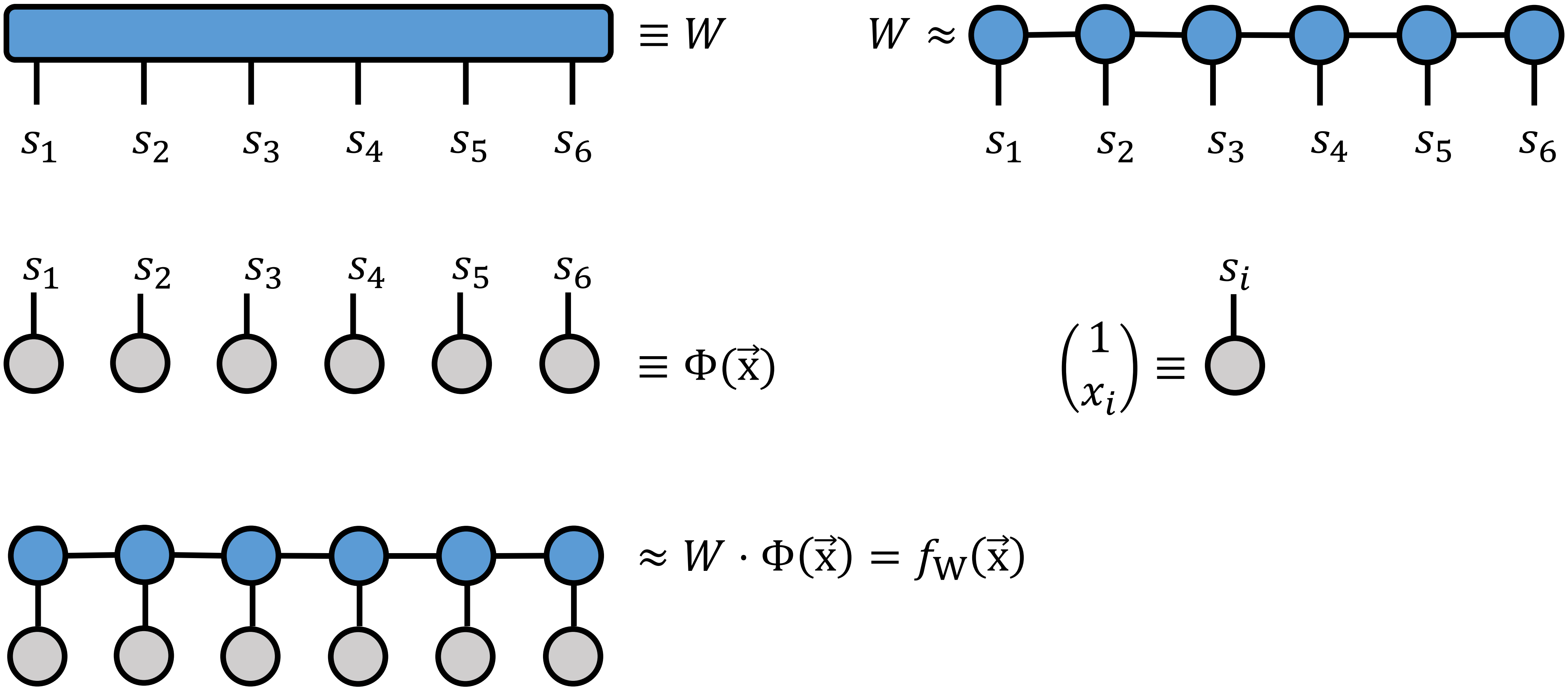
Linear Regression with TN

Choose: $$f_W(\vec{x}) = W \cdot \Phi(\vec{x}) = \sum_{s_i=0}^{1} W_{s_1 s_2 s_3 \dots s_N}\, x_1^{s_1} x_2^{s_2} x_3^{s_3} \dots x_N^{s_N}$$

Coefficients look the same as amplitudes of a Many-Body wave function. For instance in the case of a problem with 3 features:

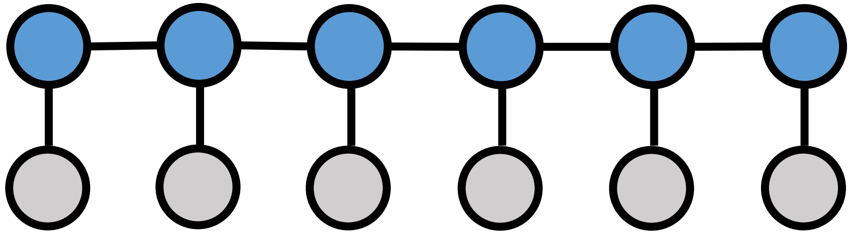
$$\begin{aligned} f_W(\vec{x}) = W \cdot \Phi(\vec{x}) &= \sum_{s_i=0}^{1} W_{s_1 s_2 s_3}\, x_1^{s_1} x_2^{s_2} x_3^{s_3} \\ &= W_{000} + W_{100} x_1 + W_{010} x_2 + W_{001} x_2 \\ &+ W_{110} x_1 x_2 + W_{101} x_1 x_3 + W_{011} x_2 x_3 \\ &+ W_{111} x_1 x_2 x_3 \end{aligned}$$

Tensor Networks Representation

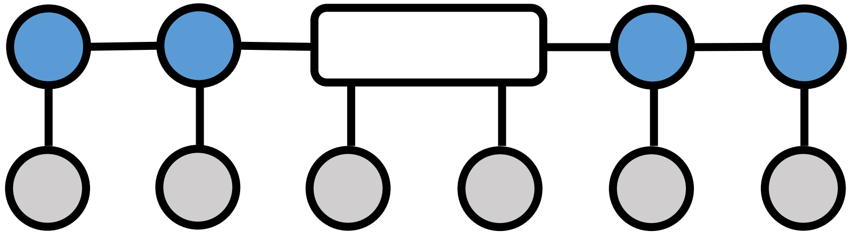


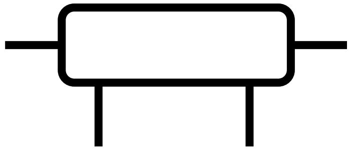
Algorithm

Start with a random MPS W and express the model scoring $f_W(\vec{x})$ as:



Combine 2 sites:

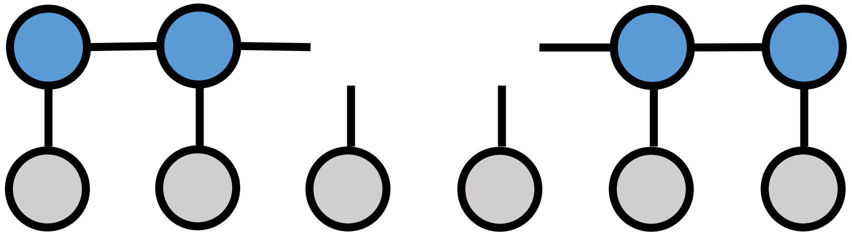


$B_{ijkl} :=$ 

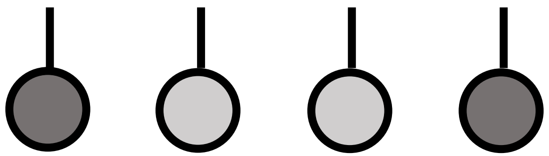
...continued

Fix other sites and optimize B, such that the cost function is minimized as a function B: $\min\limits_{B \in \mathbb{R}^{d^2 D^2}} C(B)$

$$C(W) \propto \sum_{i=1}^{m} \left(W\, \Phi\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)} \right)^2 \rightarrow C(B) \propto \sum_{i=1}^{m} \left(B\, \widetilde{\Phi}\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)} \right)^2$$

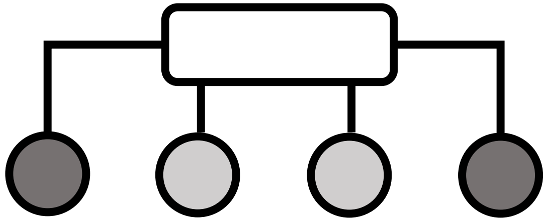


$\Phi\left(\vec{\mathrm{x}}^{(i)}\right)$ $=$



$\widetilde{\Phi}\left(\vec{\mathrm{x}}^{(i)}\right) =: \widetilde{\Phi}^{(i)}$

$B\widetilde{\Phi}^{(i)} =$



B

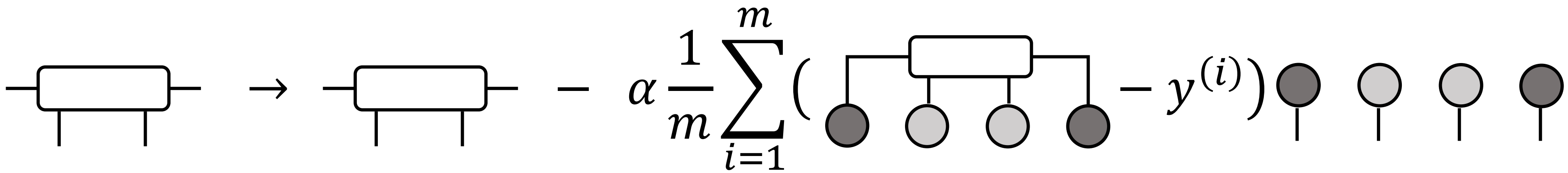
$\widetilde{\Phi}^{(i)}$

…continued

Use Gradient Descent to compute find $\min_{B} C(B)$:

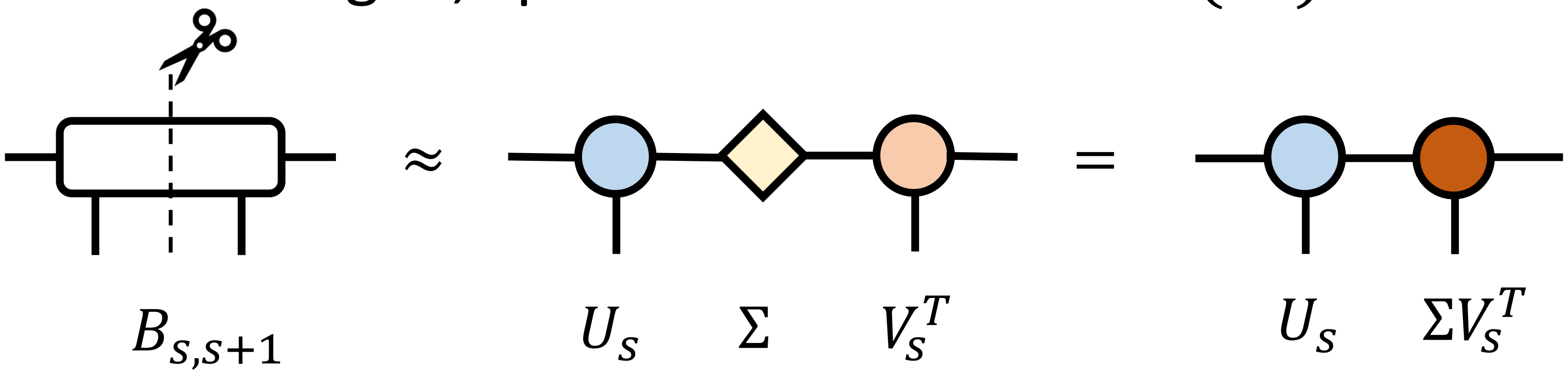
$$B \to B - \alpha\, \Delta B$$

$$\Delta B = \frac{1}{2m} \frac{\partial}{\partial B} \sum_{i=1}^{m} \left(B\, \tilde{\Phi}^{(i)} - y^{(i)}\right)^2 = \frac{1}{m} \sum_{i=1}^{m} \left(B\, \tilde{\Phi}^{(i)} - y^{(i)}\right) \tilde{\Phi}^{(i)}$$

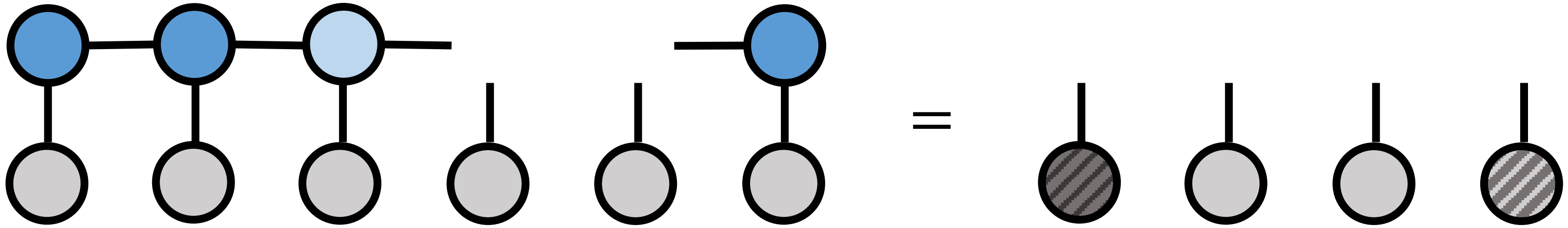


...continued

Once B has converged, split the 2-site tensor: $C(W)$

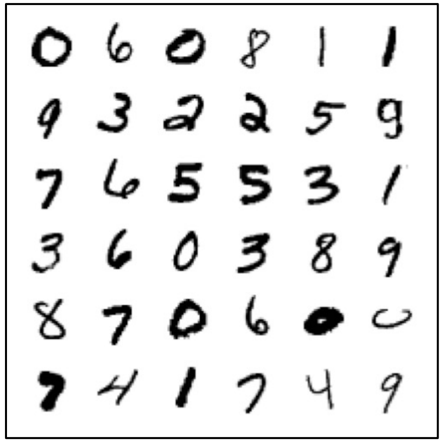


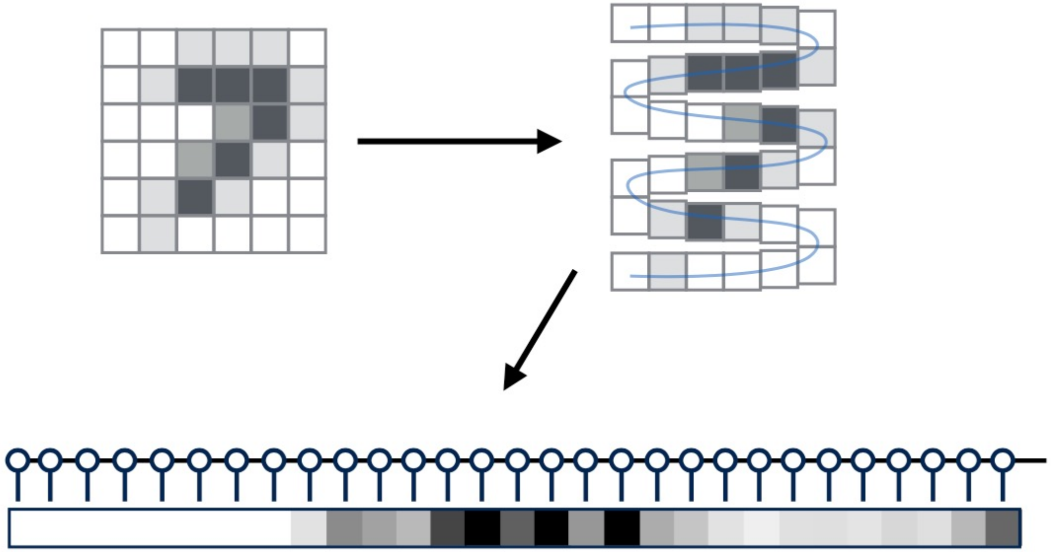
And move one site further in the chain:



Sweep back and forth until $C(W)$ has converged.

Results – MNIST



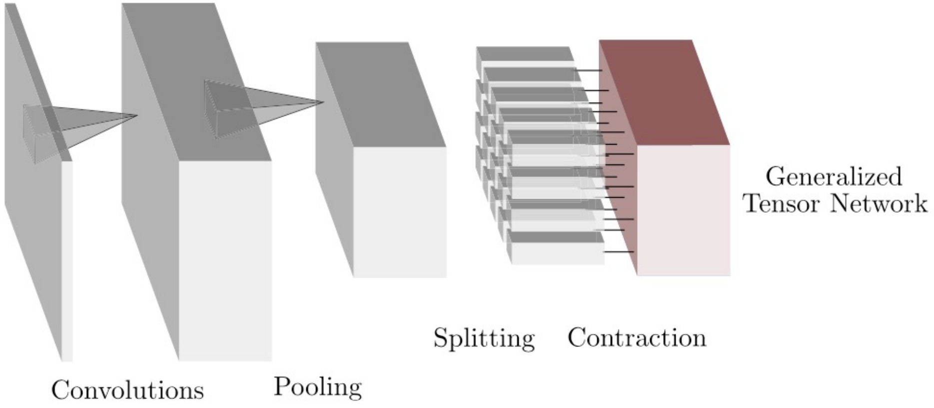


Train to 99.95% accuracy on 60,000 training images

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

Stoudenmire, E.M., & Schwab, D. (2016). Supervised Learning with Tensor Networks. *NIPS*.

Results – Fashion MNIST



Method	Accuracy
Support Vector Machine	84.1%
EPS + linear classifier	86.3%
Multilayer perceptron	87.7%
EPS-SBS	88.6%
Snake-SBS	89.2%
AlexNet	89.9%
CNN-snake-SBS	92.3%
GoogLeNet	93.7%

Glasser, I., Pancotti, N., & Cirac, J.I. (2020). From Probabilistic Graphical Models to Generalized Tensor Networks for Supervised Learning. *IEEE Access, 8*, 68169-68182.

Libraries

- TensorNetwork (Python)
- TorchMPS (Python)
- Itensor (C++, Julia)
- TensorLy (Python)

https://tensornetwork.org/software/

Benefits

- Realized Benefits
    - Linear scaling in nr of inputs
    - Adaptive weights, i.e. model complexity changes dynamically
    - Learning data “features”
- Future benefits?
    - Interpretability
    - Better algorithms
    - Quantum computing

Thank you!

bpirvu@novomatic.com

NOVOMATIC

NOVOMATIC