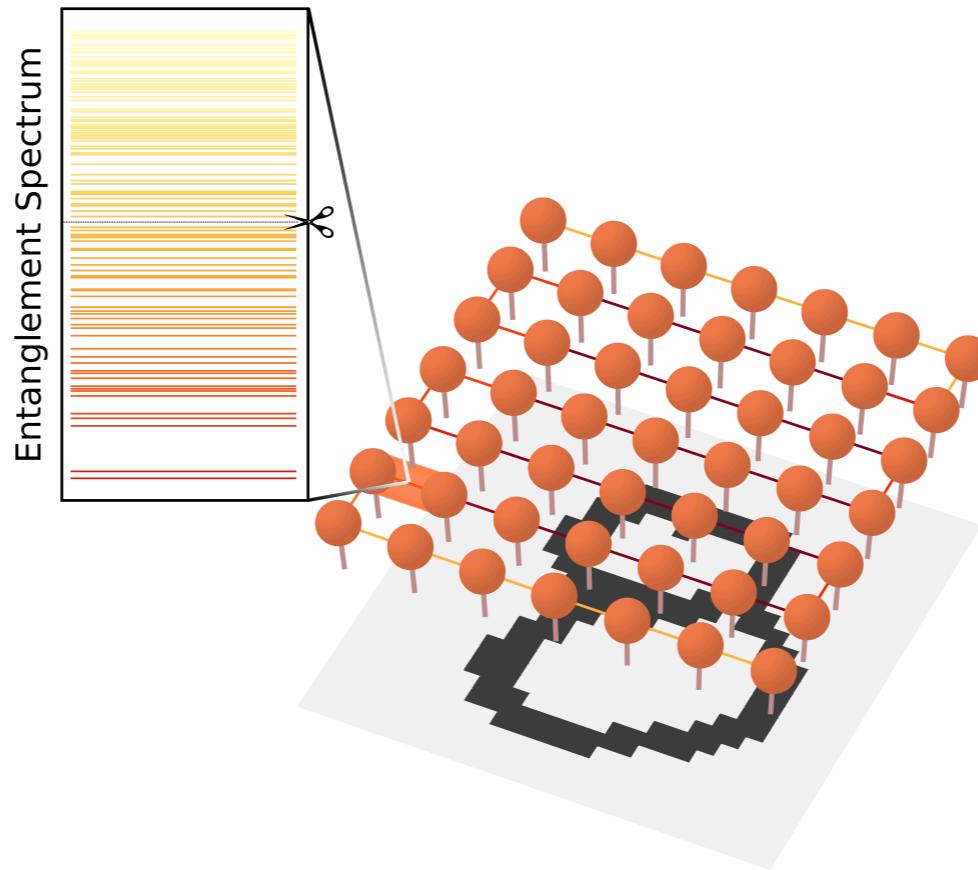


# Tensor networks

## for representation and learning



Pan Zhang  
ITP,CAS

SSH summer school  
May.06 — 10, 2019



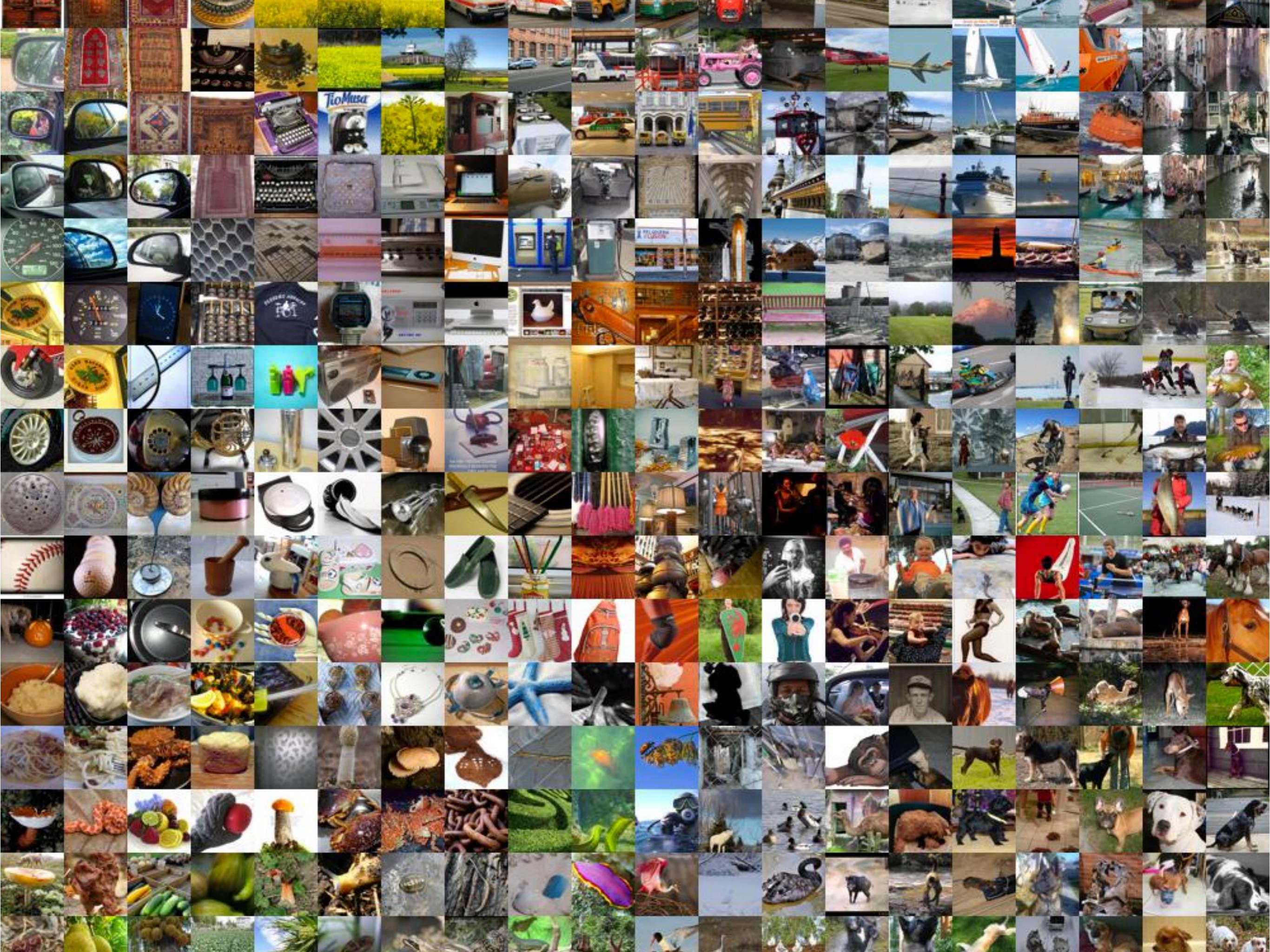
# Tensor Networks

- In physics: **wave functions**
- Out of physics:
  - Revealing internal low-rank structures (CP rank, TT rank)
  - Large scale representation: compression
  - Large scale optimization: ALS, eigenvalues...
  - Supervised learning: (kerneled) classification/regression...
  - Unsupervised learning: kernel PCA, generative modeling...

# Outline

## → Tensor networks for representations *(in a small space)*

- CP, Tucker, MPS, Tree Tensor networks
- Example: Compressing neural networks
- Contraction of tensor networks
- Tensor networks for learning  
*(in a large space)*
  - Hilbert space as feature space: classification and PCA using tensor networks
  - Tensor networks for generative modeling

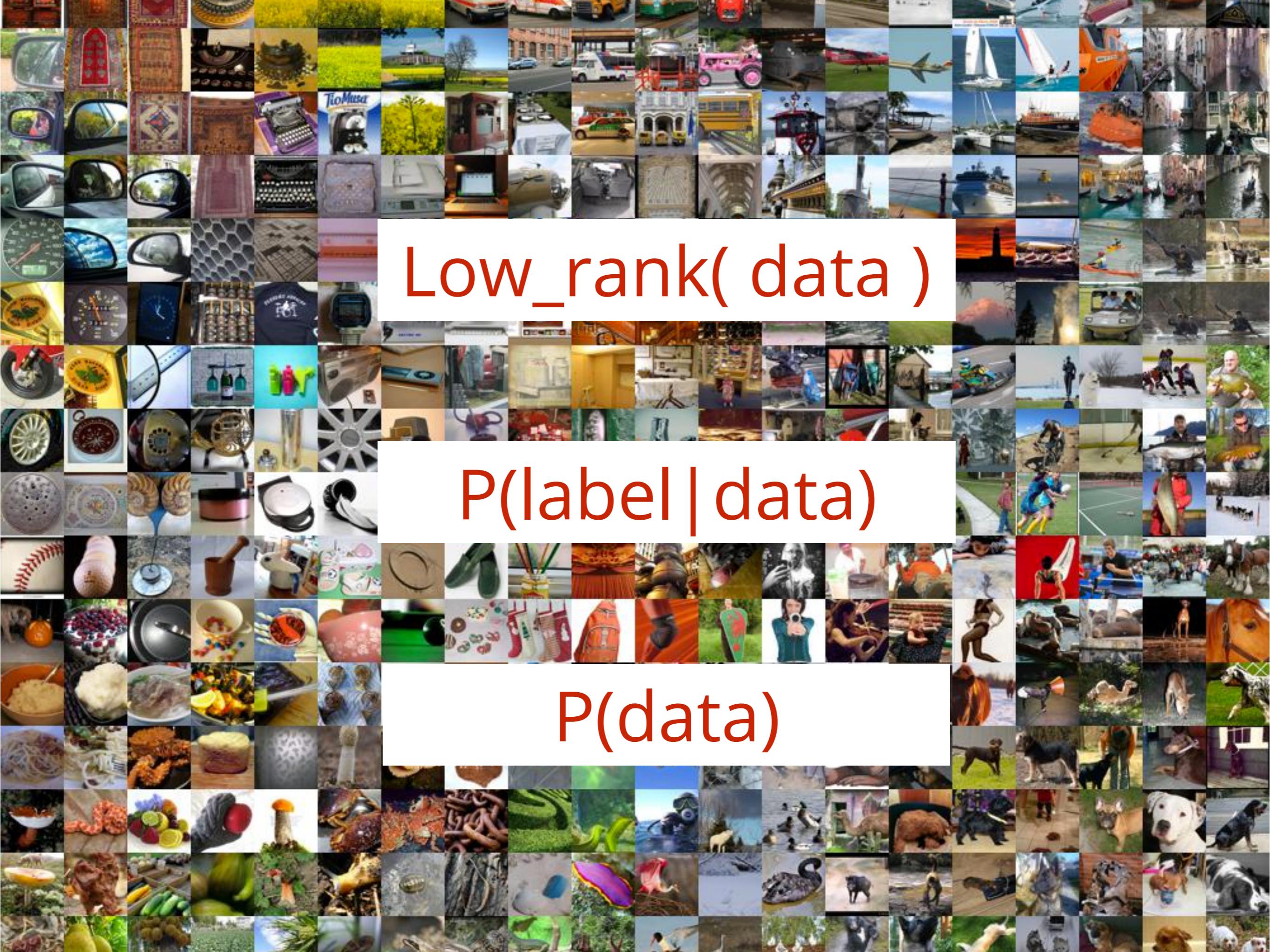


# Low\_rank( data )



Low\_rank( data )

P(label | data)



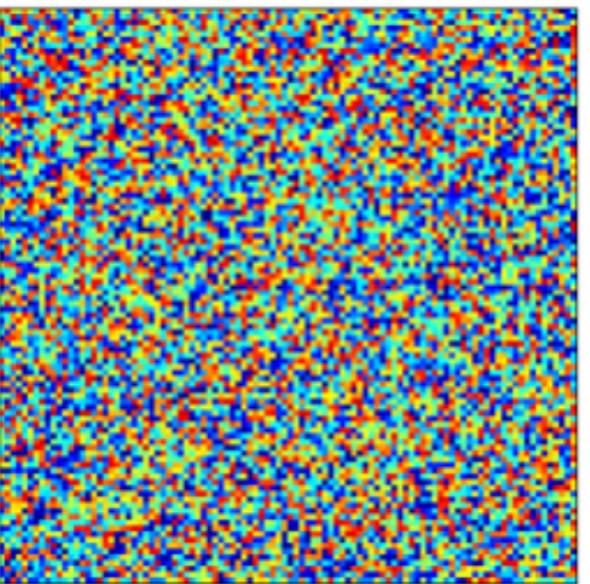
Low\_rank( data )

P(label | data)

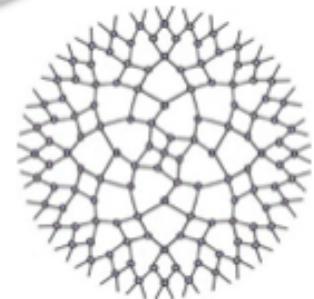
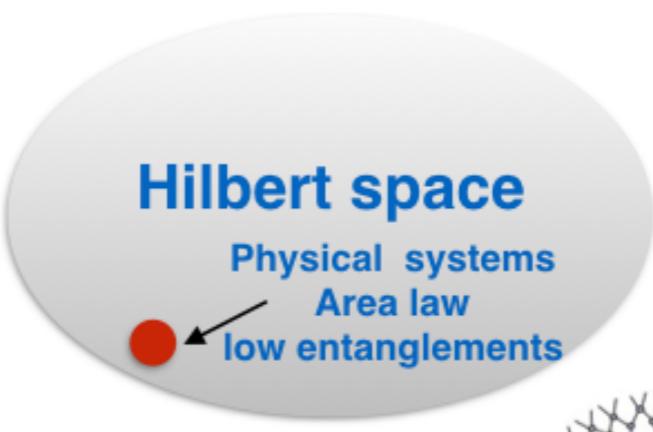
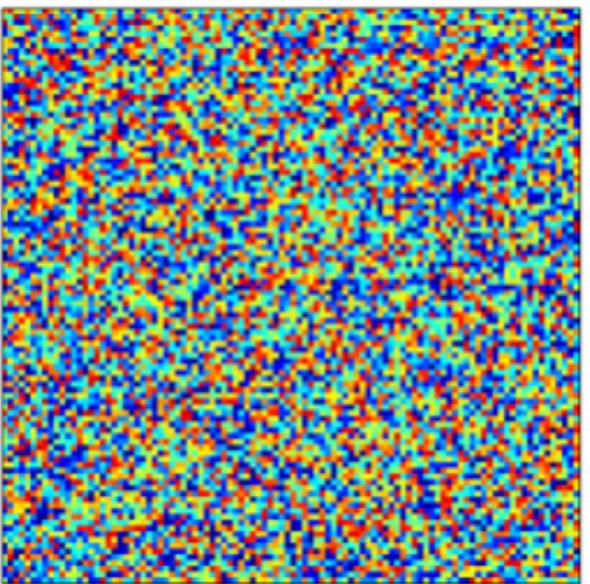
P(data)

# Analogous to quantum physics

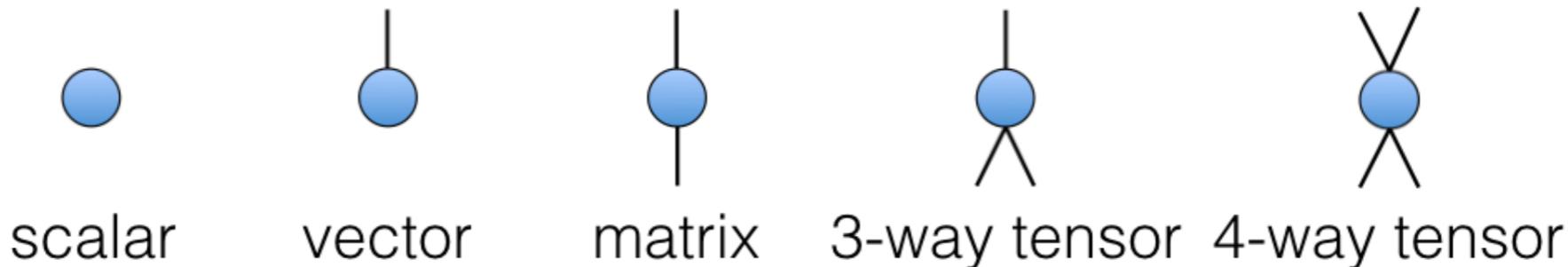
# Analogous to quantum physics



# Analogous to quantum physics



# Diagram notations



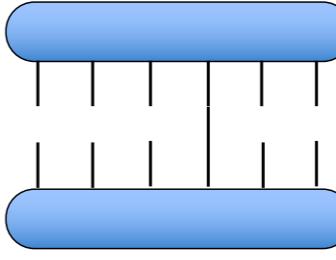
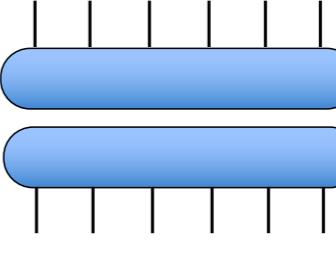
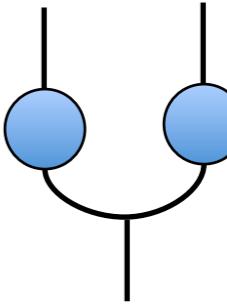
tensor product

Trace

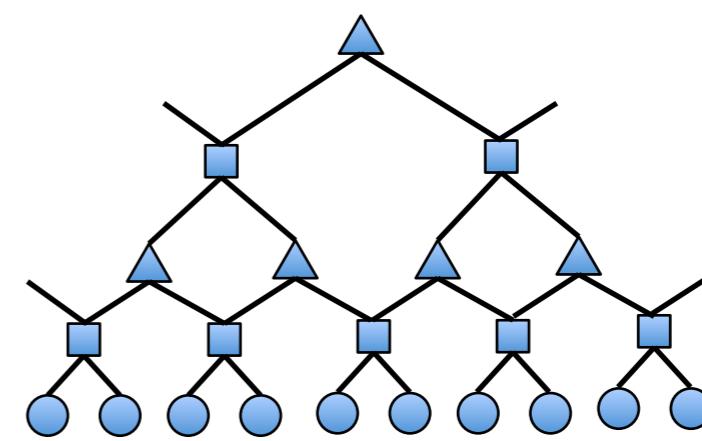
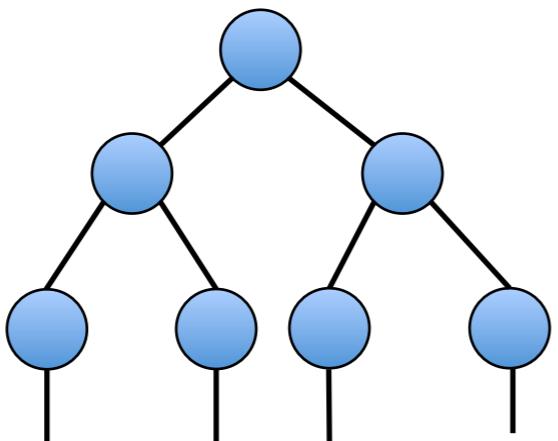
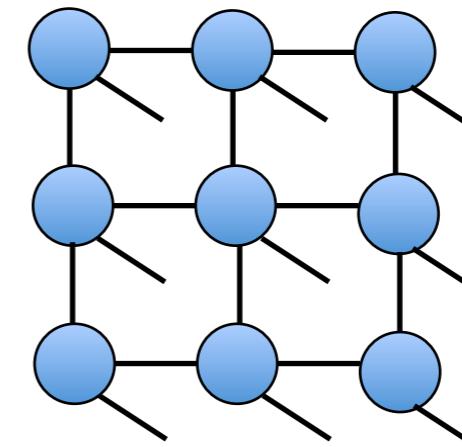
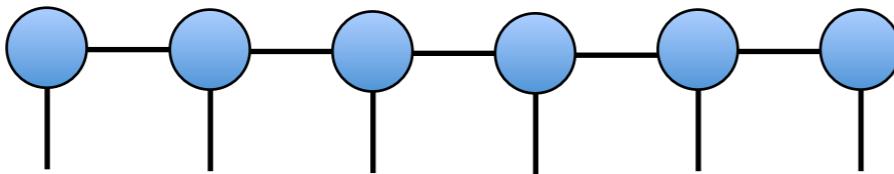
tensor contraction

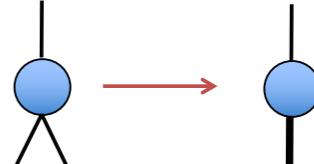
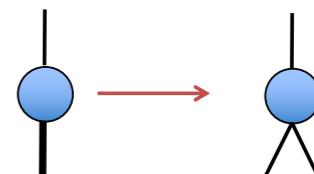
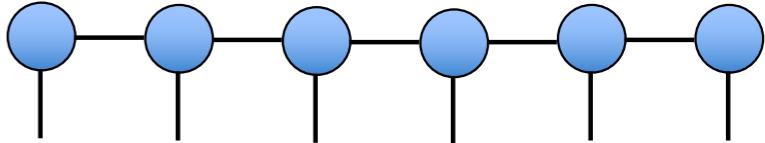
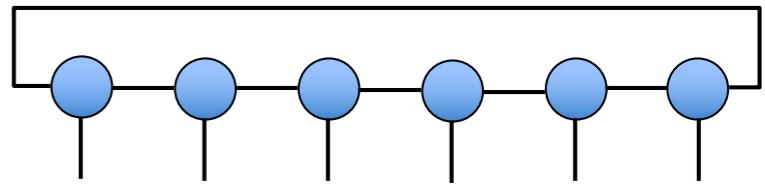
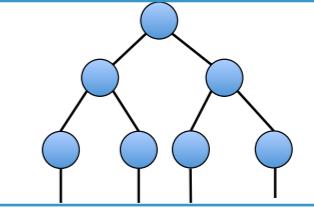
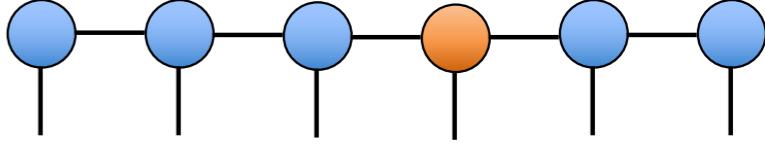
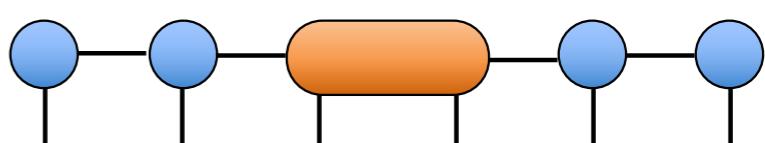
tensor contraction

# Diagram notations

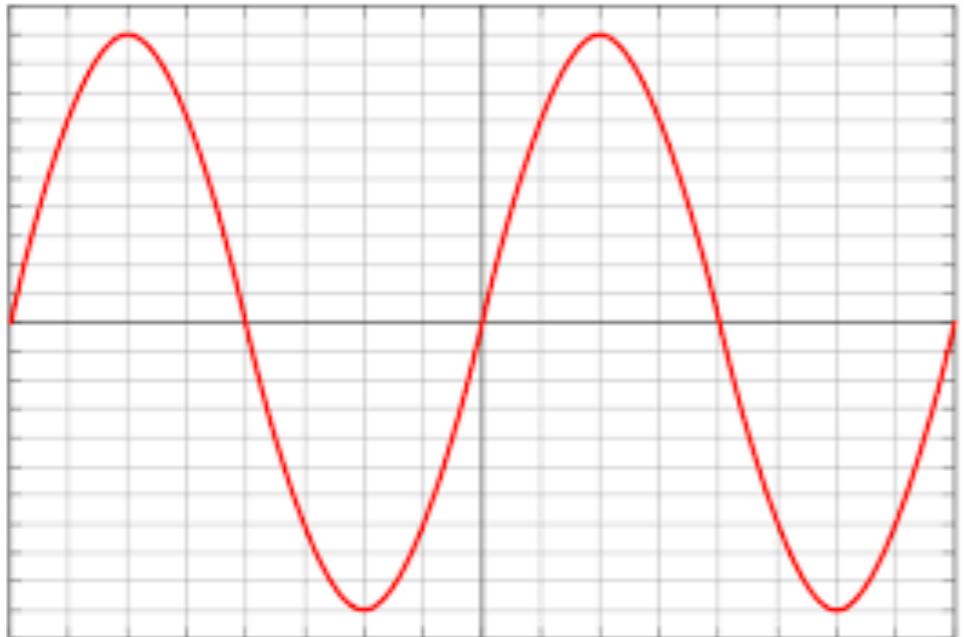
Operations	Formula	Diagram
n-mode product of two tensors contraction	$\mathcal{A} \times_n \mathcal{B}$	
Kronecker product of two tensors	$\mathcal{A} \otimes \mathcal{B}$	
Khatri-Rao product of two matrices	$\mathcal{A} \odot \mathcal{B}$	
Hadamard (component) product	$\mathcal{A} * \mathcal{B}$	

# Tensor networks in physics: imposing prior of physical wave function



In Physics	Out of Physics	Diagram
grouping of indices	unfolding, matricization	
splitting of indices	tensorizing	
matrix product states	tensor train decomposition	
periodic boundary MPS	tensor chain decomposition	
tree tensor networks	hierarchical Tucker decompostion	
single-site DMRG	alternating least square	
two-site DMRG	modified alternating least square	

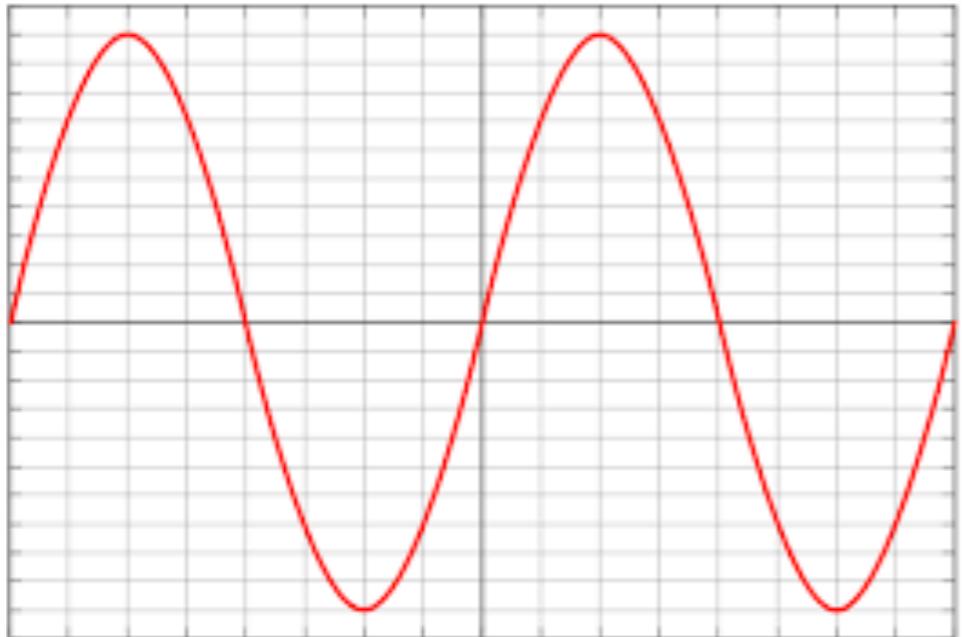
# Exploring internal structures in the data



```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  .....
  .....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

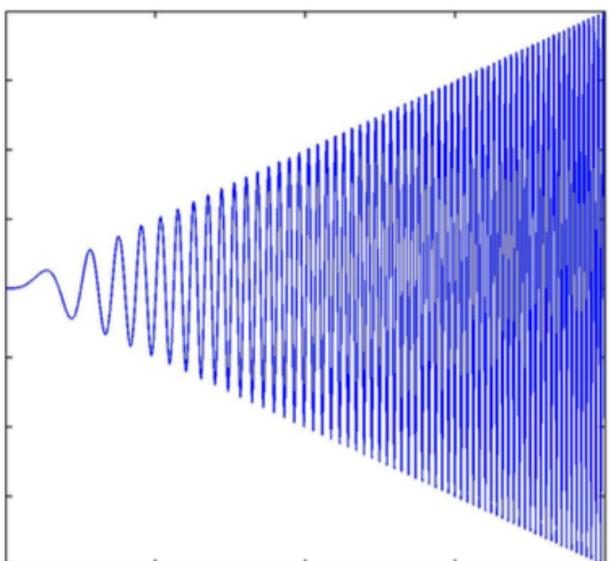
$10^6$  data points in a vector

# Exploring internal structures in the data

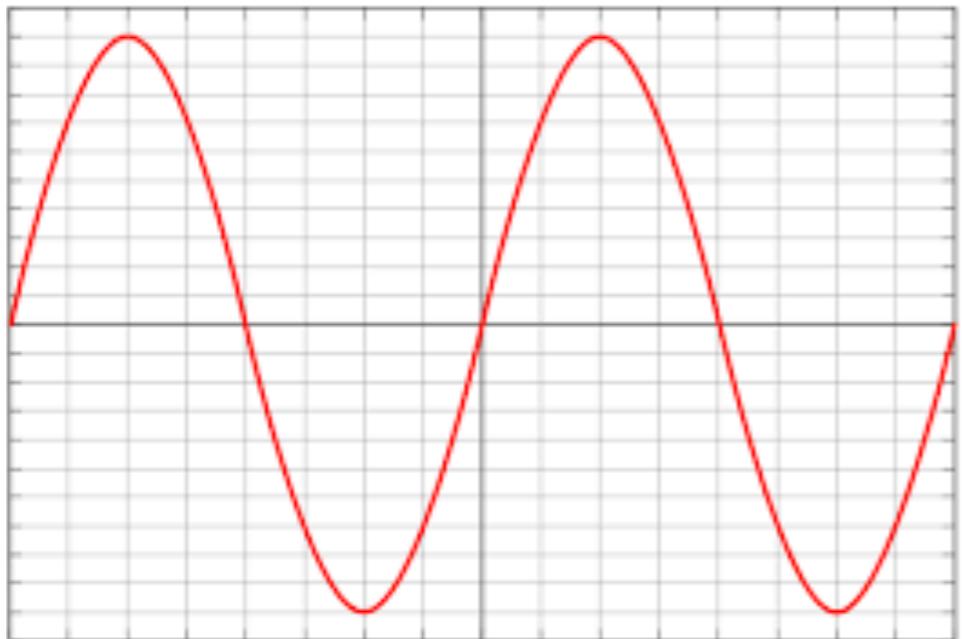


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector

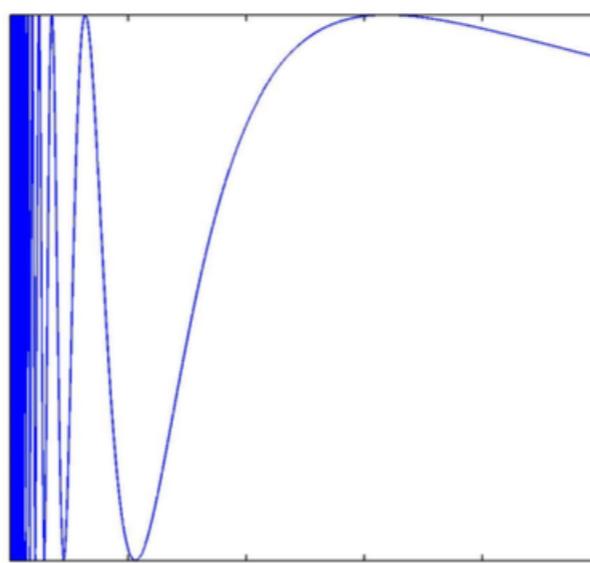
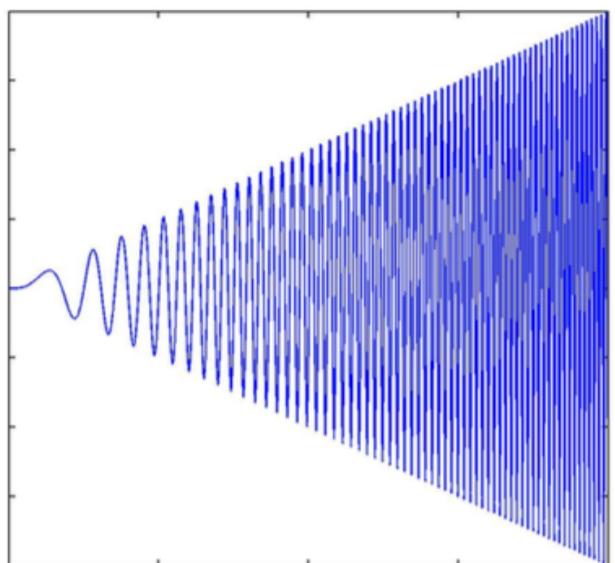


# Exploring internal structures in the data

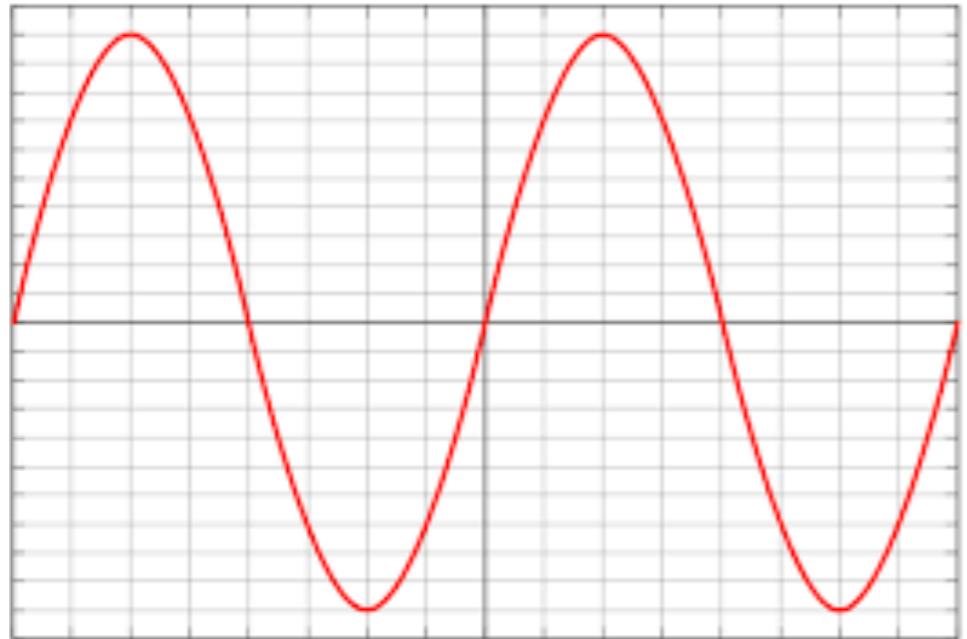


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector

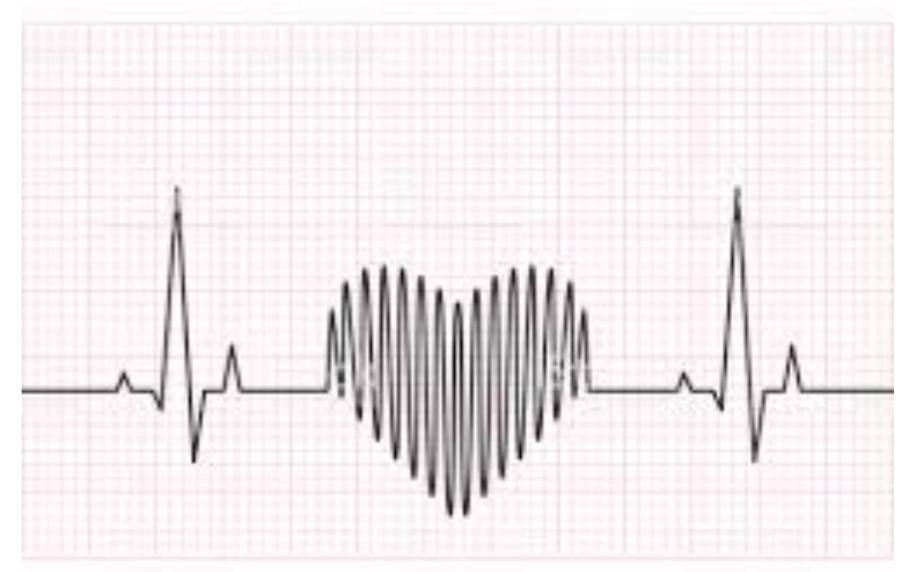
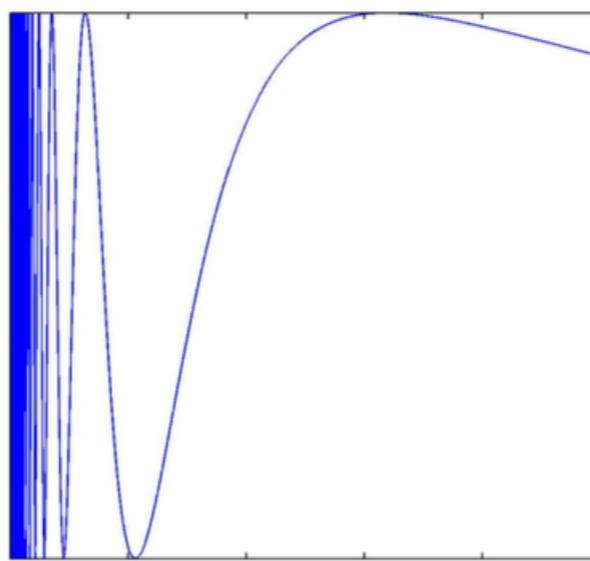
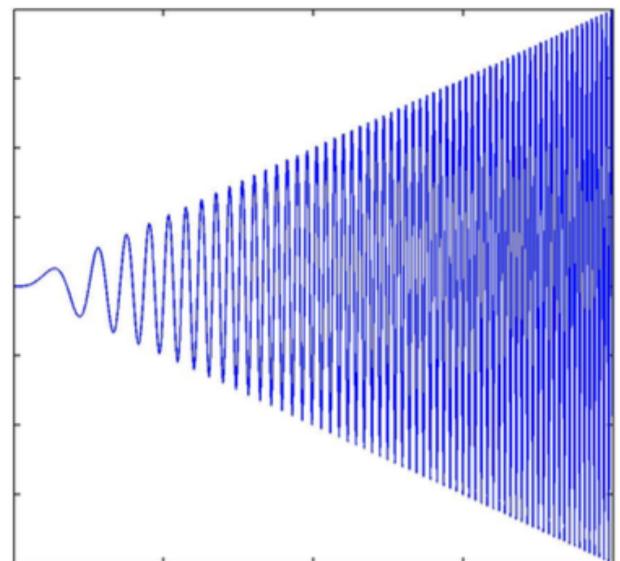


# Exploring internal structures in the data

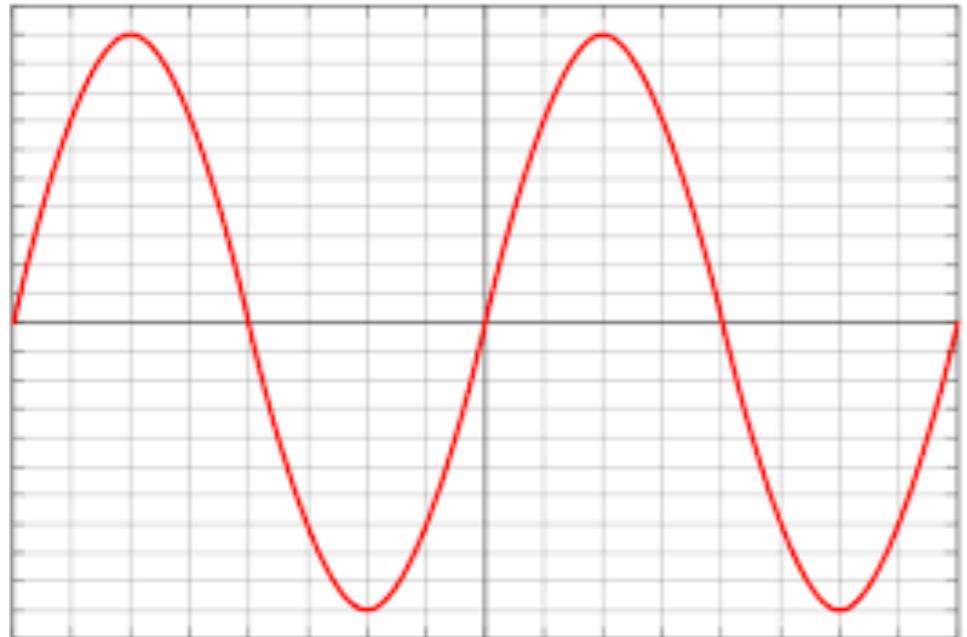


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector

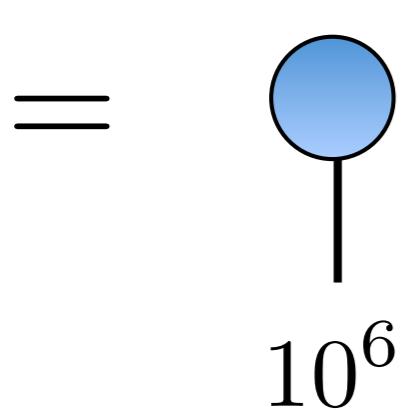


# Exploring internal structures in the data

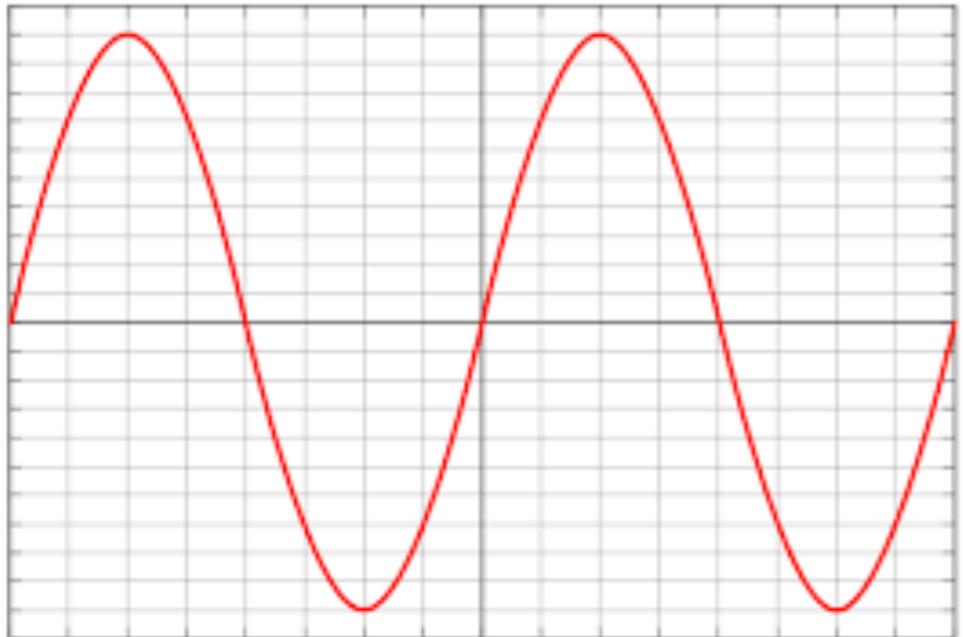


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  .....
  .....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector

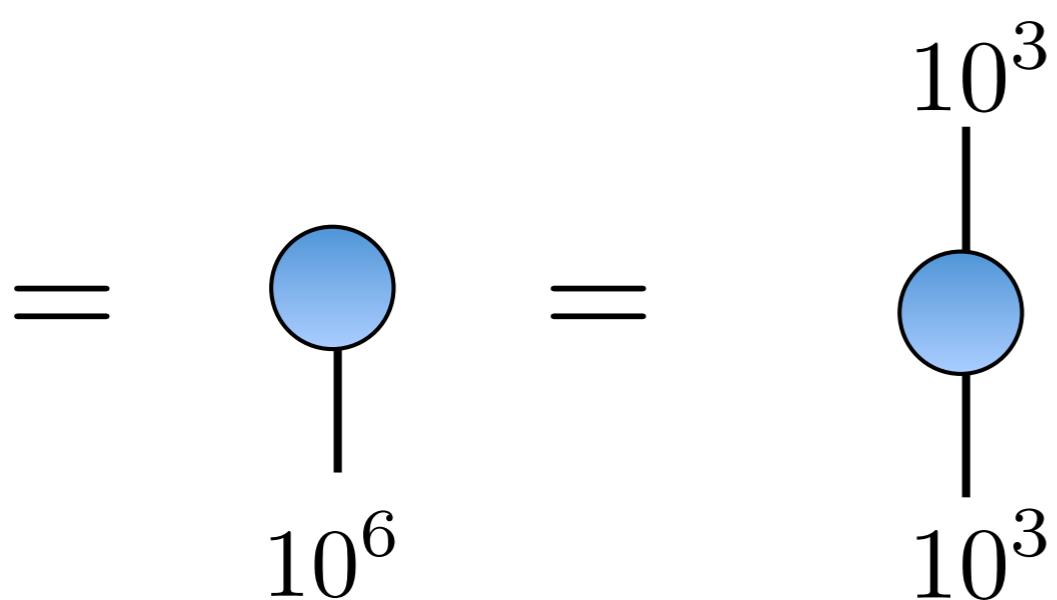


# Exploring internal structures in the data

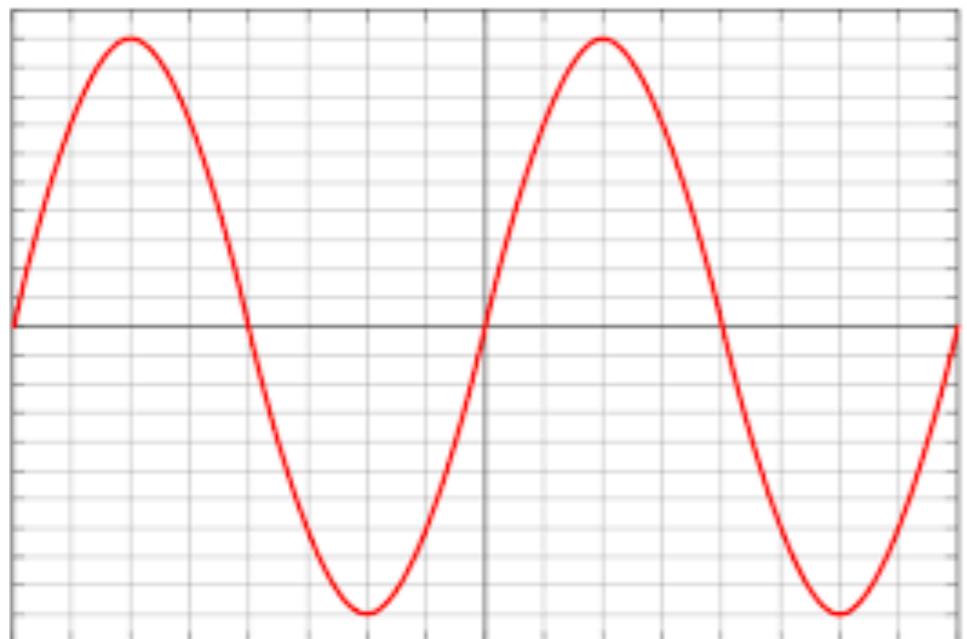


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector

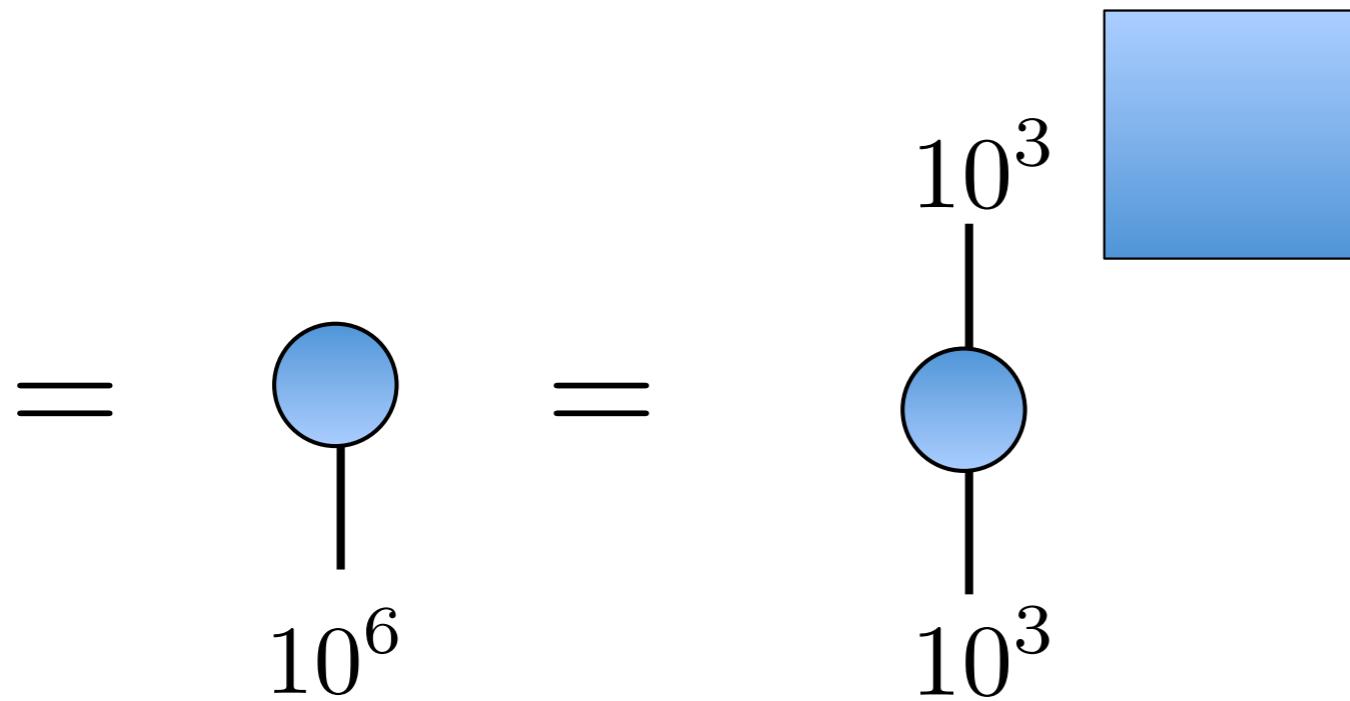


# Exploring internal structures in the data

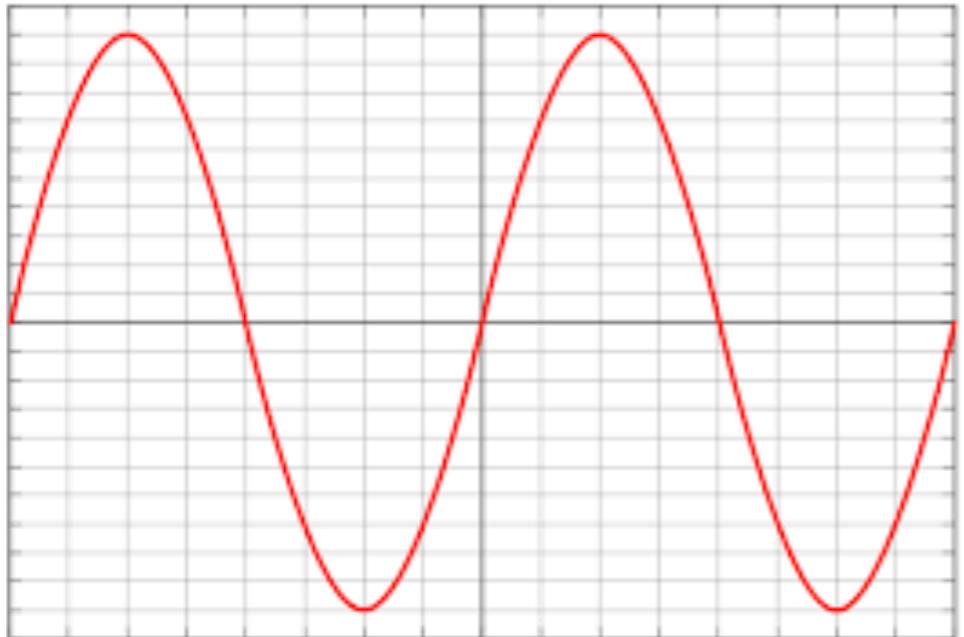


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector

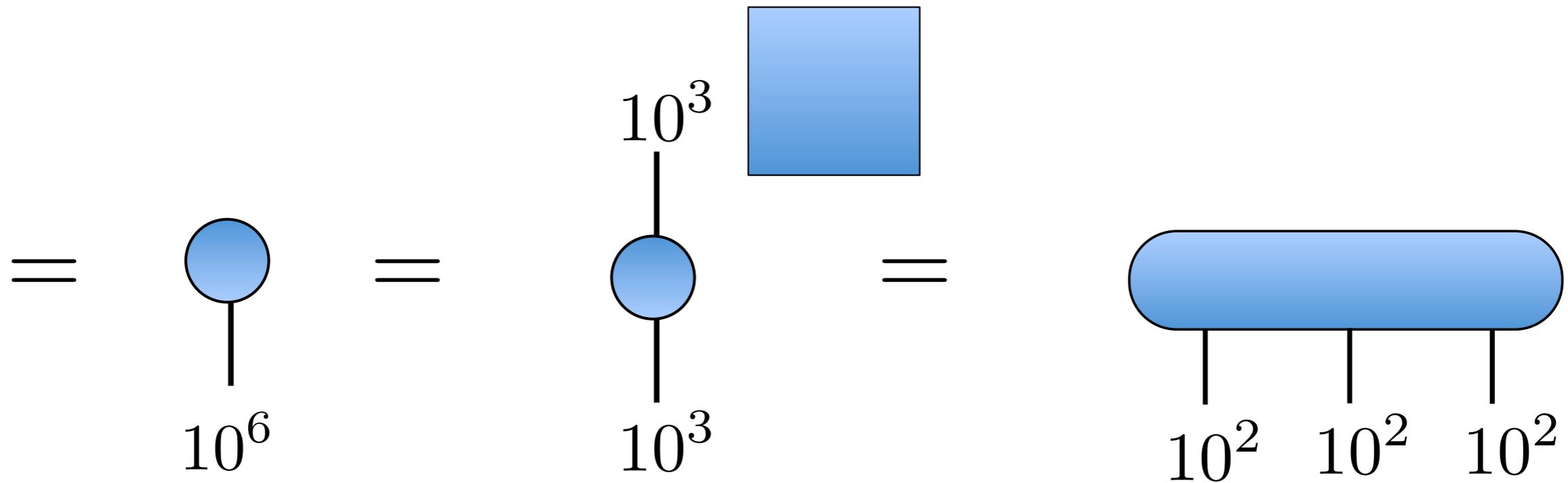


# Exploring internal structures in the data

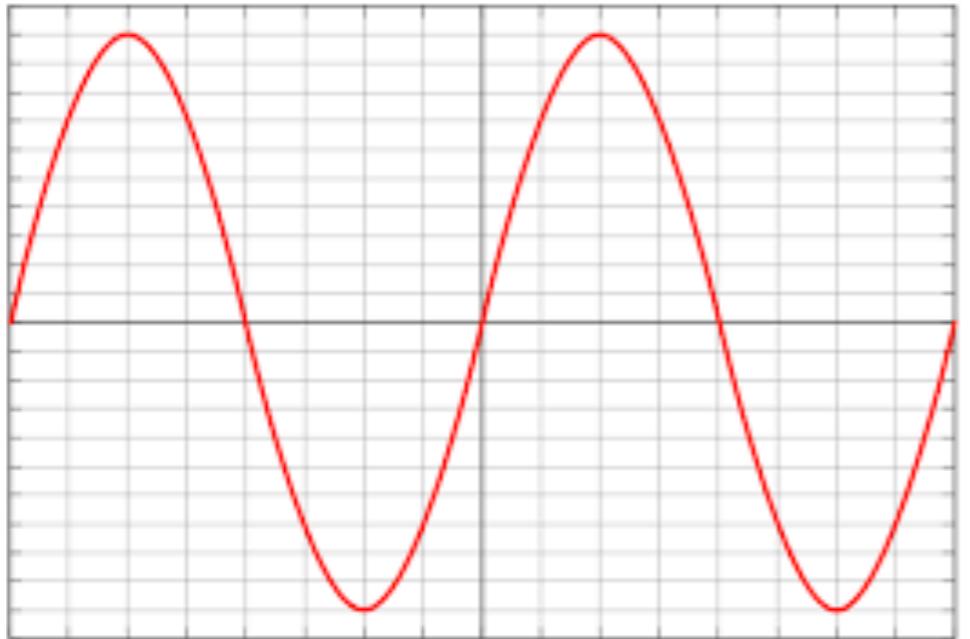


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,  
 0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,  
 0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,  
 0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,  
 .....  
 .....  
 -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,  
 -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,  
 -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,  
 0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector

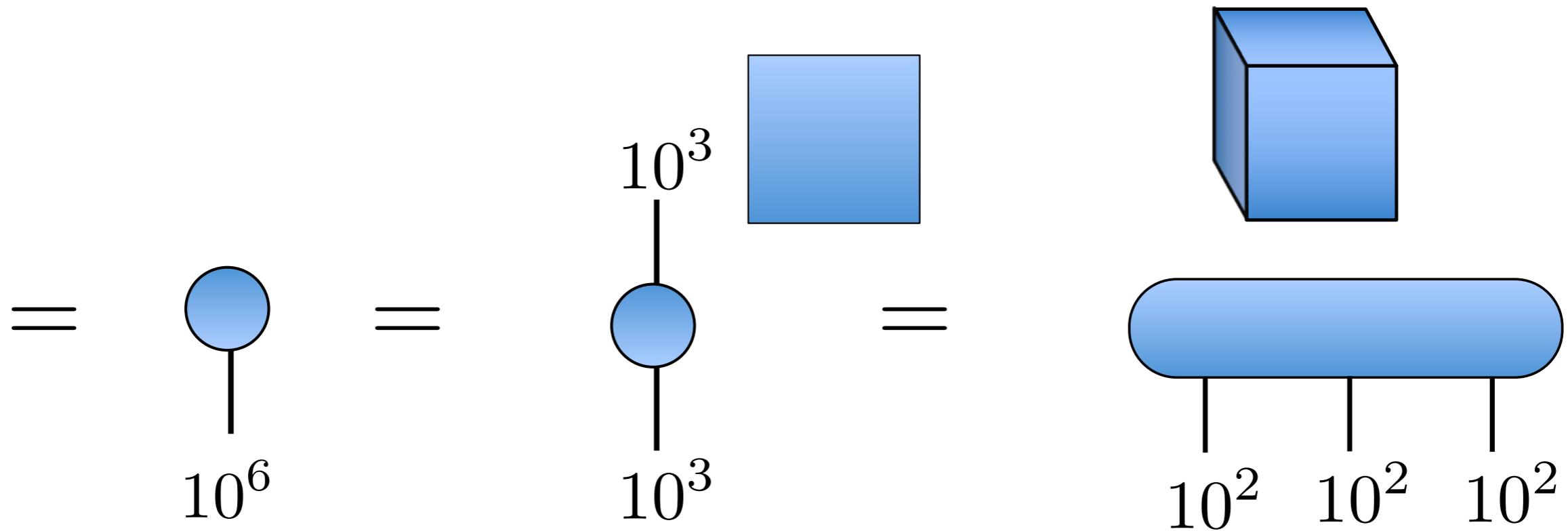


# Exploring internal structures in the data

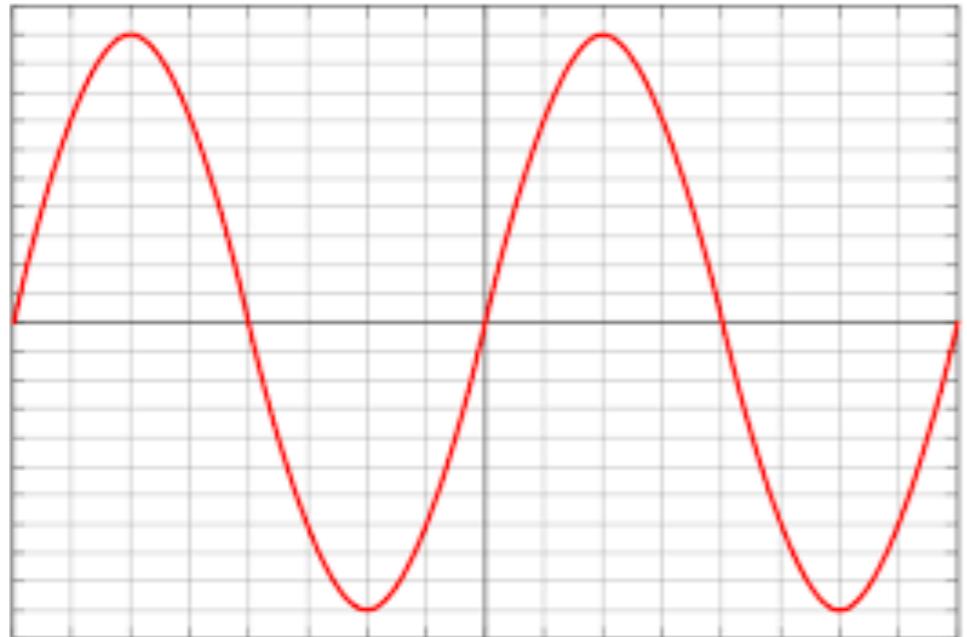


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector

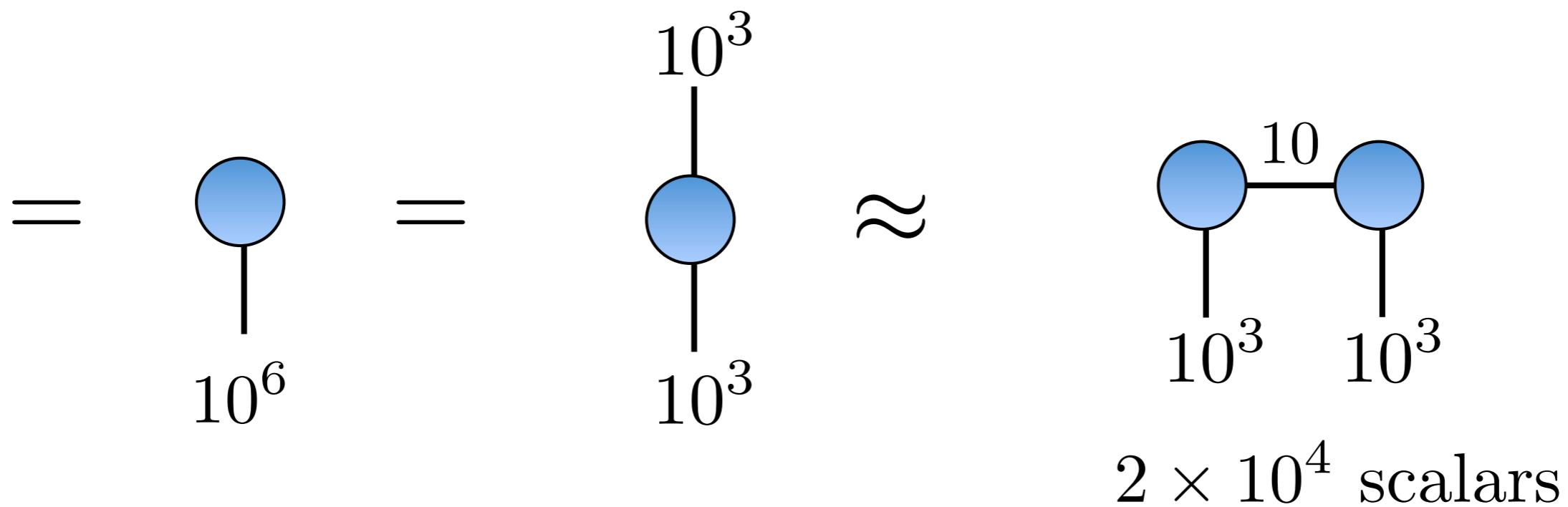


# Exploring internal structures in the data

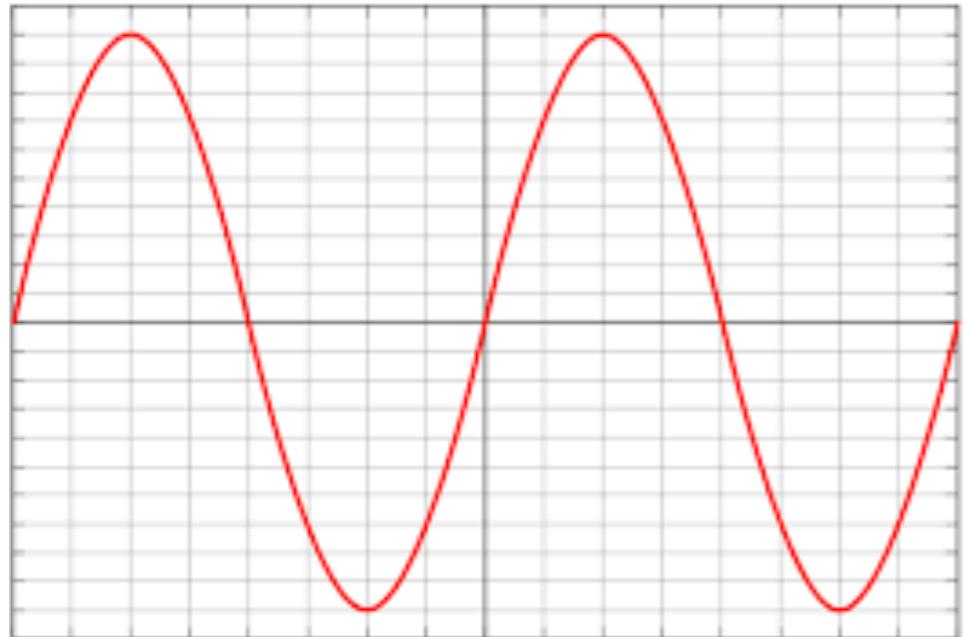


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector



# Exploring internal structures in the data



```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

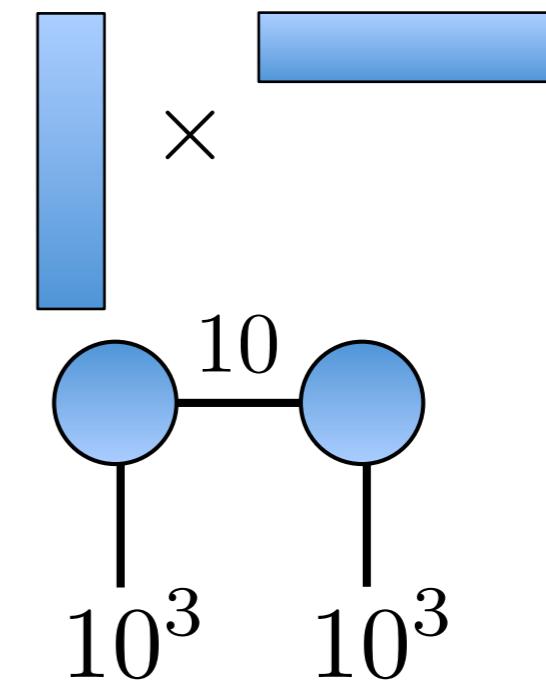
$10^6$  data points in a vector

$$= \text{ } \begin{matrix} \text{ } \\ \text{ } \end{matrix} \text{ } = \text{ } \begin{matrix} \text{ } \\ \text{ } \end{matrix} \text{ } = \text{ } \begin{matrix} \text{ } \\ \text{ } \end{matrix}$$

$10^3$   
 $10^6$

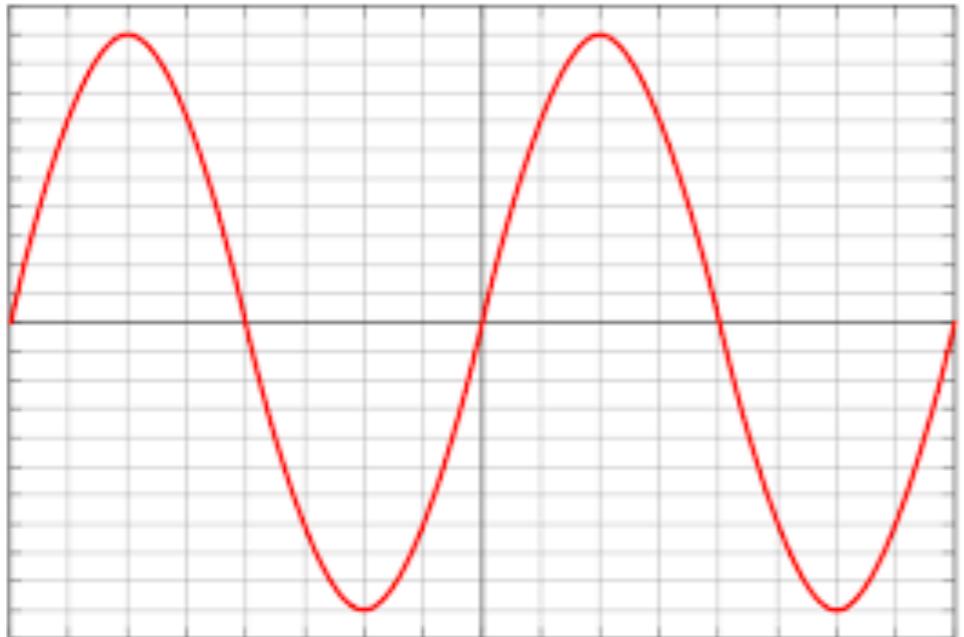
$$\approx \begin{matrix} \text{ } \\ \text{ } \end{matrix}$$

$10^3$   
 $10^3$



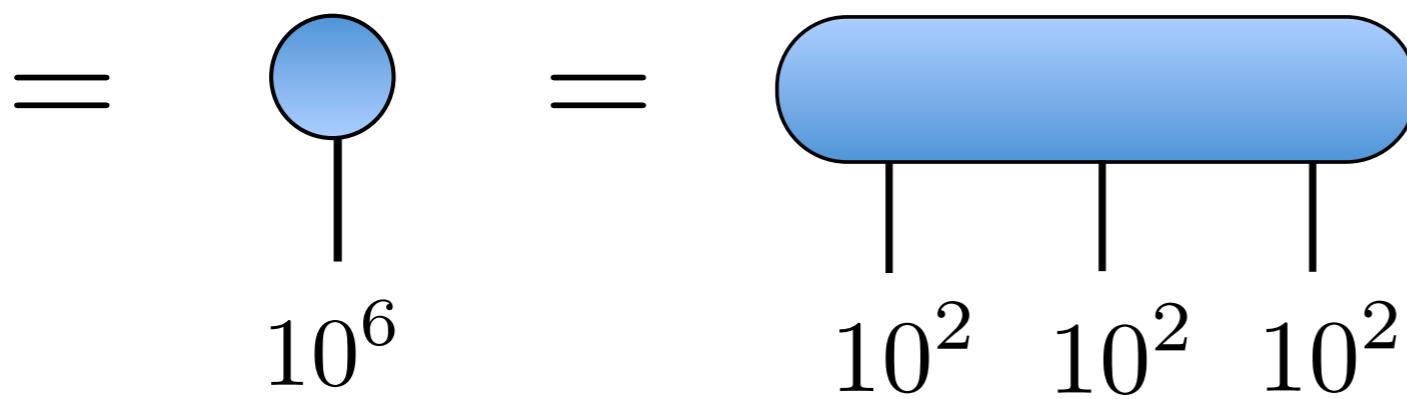
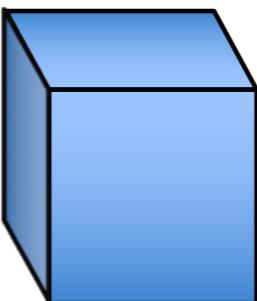
$2 \times 10^4$  scalars

# Exploring internal structures in the data

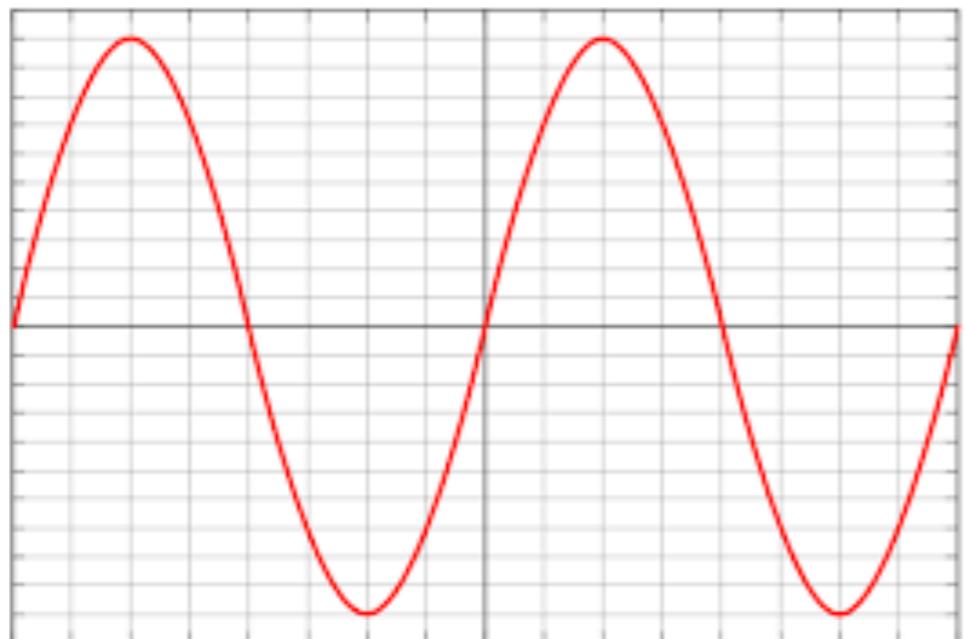


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

$10^6$  data points in a vector

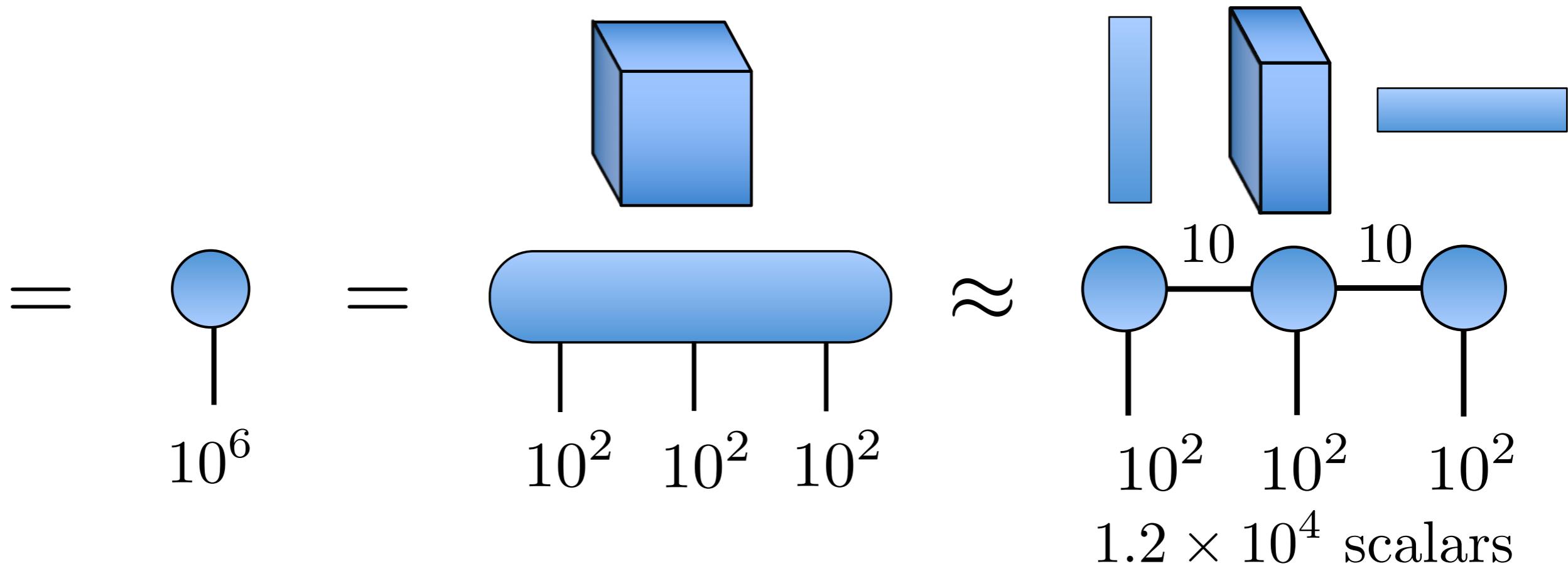


# Exploring internal structures in the data

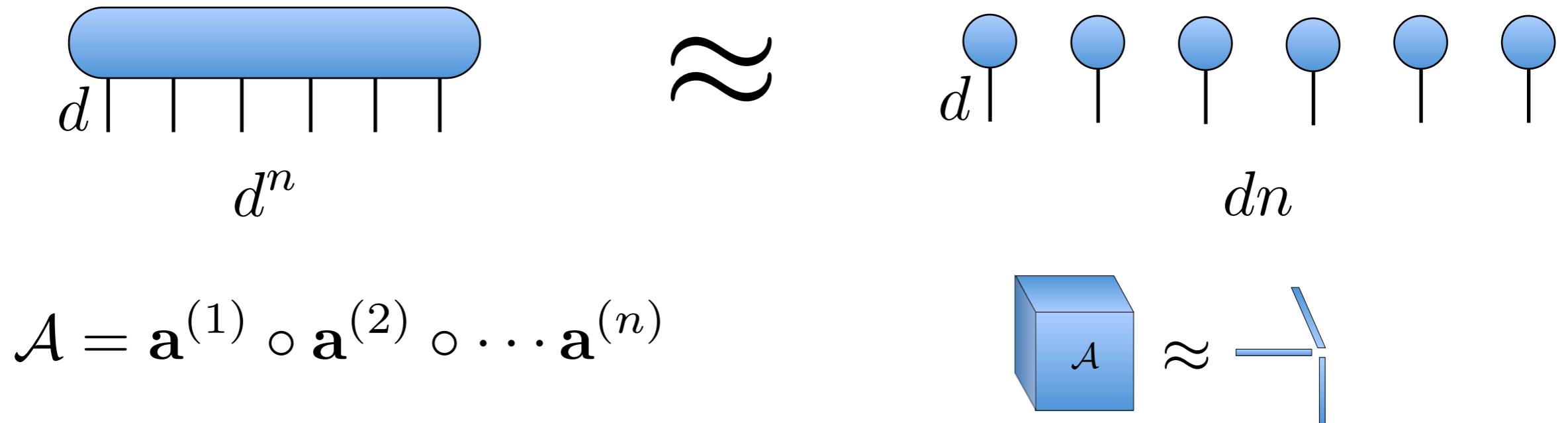


```
[ 0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
  0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
  0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
  0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
  ....
  ....
  -0.688, -0.757, -0.818, -0.872, -0.916, -0.952, -0.978, -0.994, -1.000,
  -0.996, -0.982, -0.959, -0.926, -0.883, -0.832, -0.773, -0.706, -0.631,
  -0.551, -0.465, -0.374, -0.279, -0.182, -0.083, 0.017, 0.117, 0.215,
  0.312, 0.405, 0.494, 0.578, 0.657, 0.729, 0.794, 0.850, 0.899, 0.938 ]
```

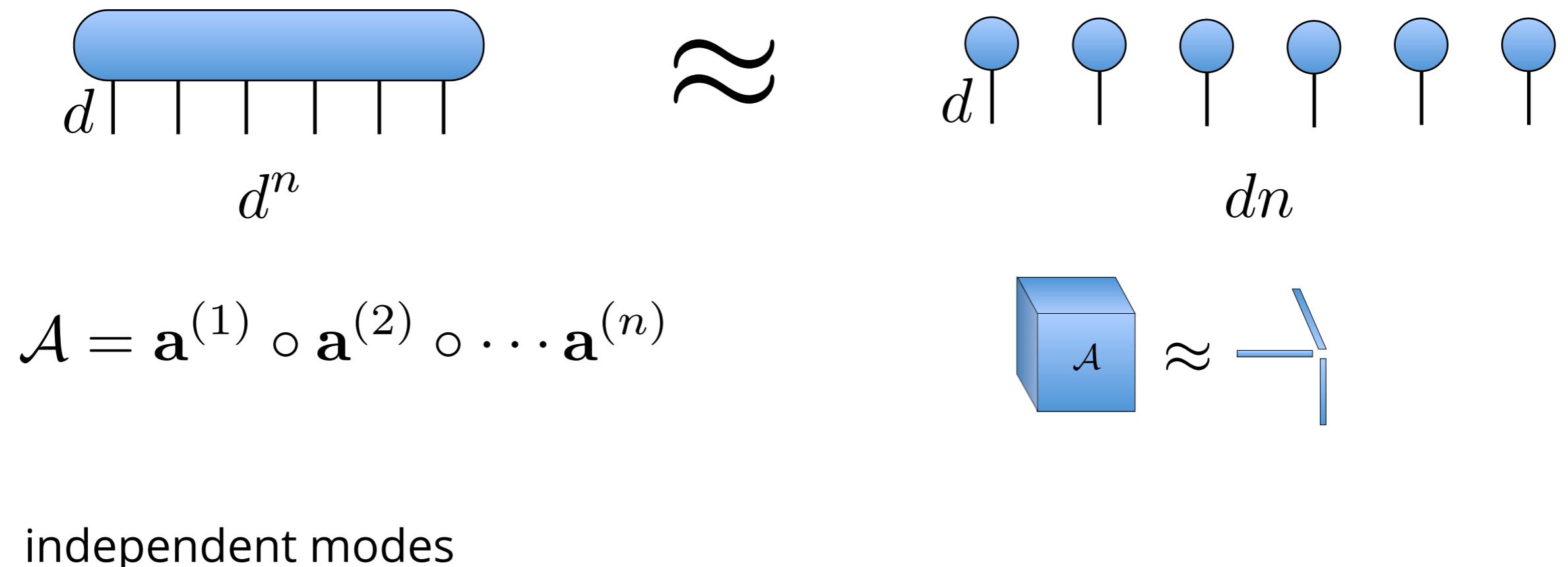
$10^6$  data points in a vector



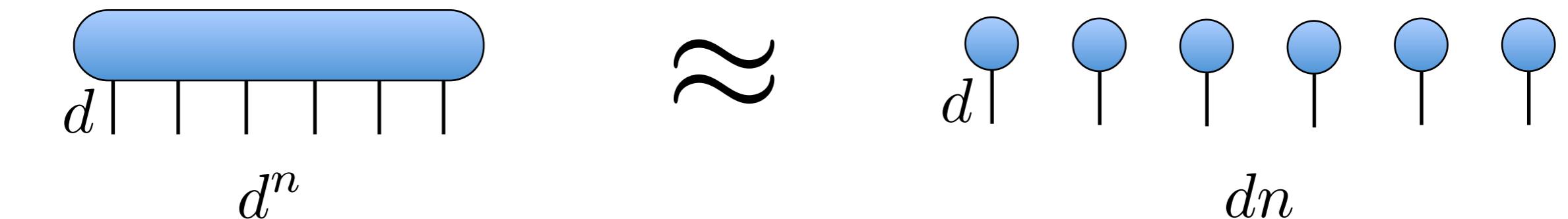
# Representation of a large tensor: rank-one



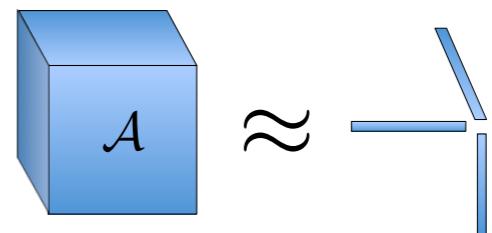
# Representation of a large tensor: rank-one



# Representation of a large tensor: rank-one



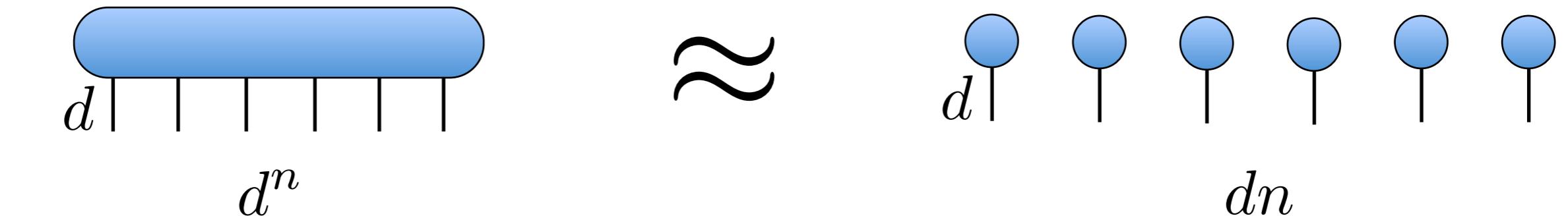
$$\mathcal{A} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \dots \mathbf{a}^{(n)}$$



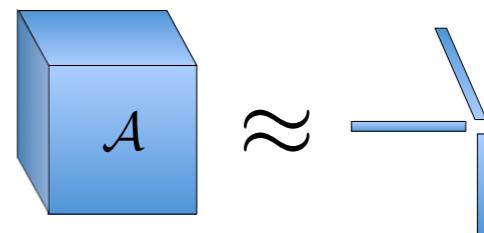
independent modes

factorized pure state

# Representation of a large tensor: rank-one



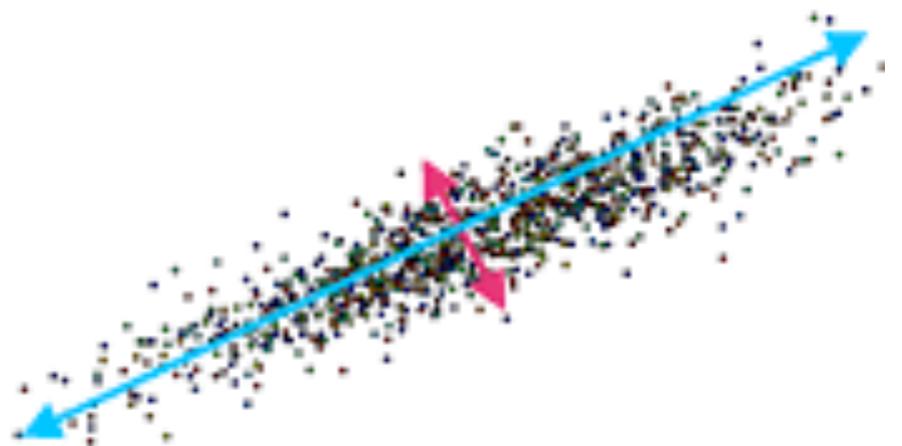
$$\mathcal{A} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \dots \mathbf{a}^{(n)}$$



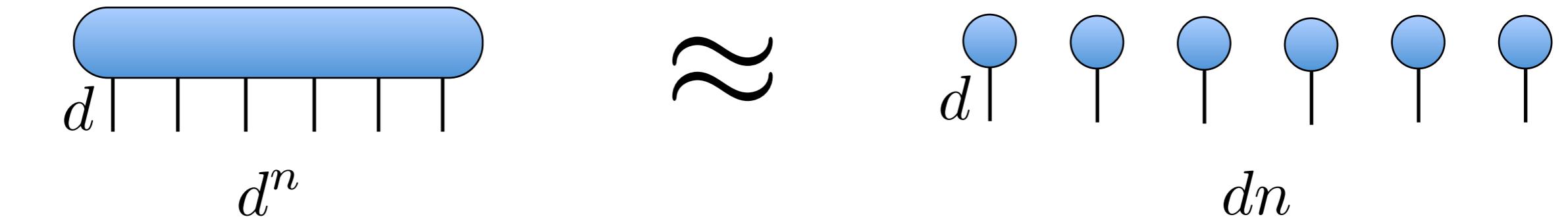
independent modes

factorized pure state

principled component



# Representation of a large tensor: rank-one



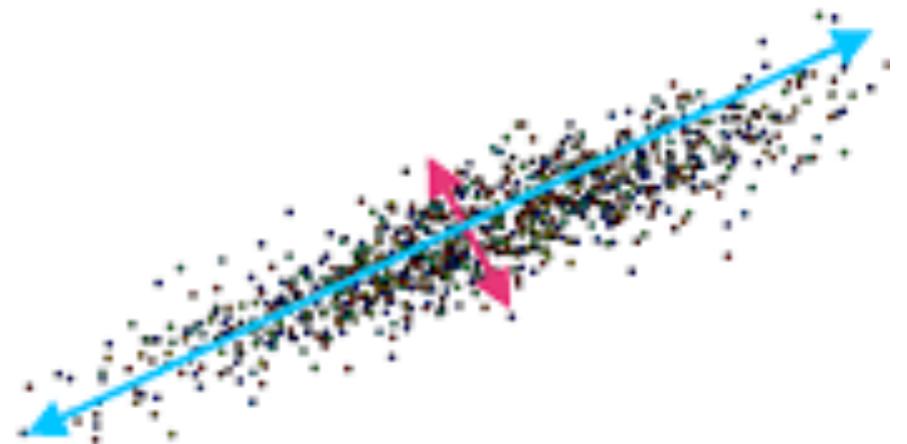
independent modes

factorized pure state

principled component

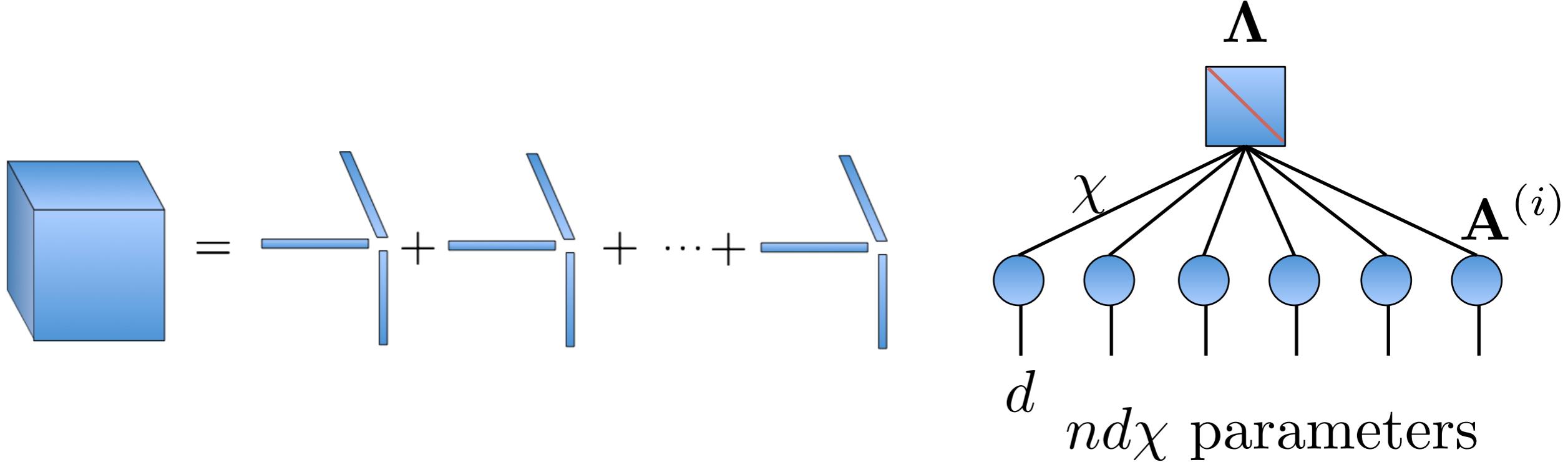
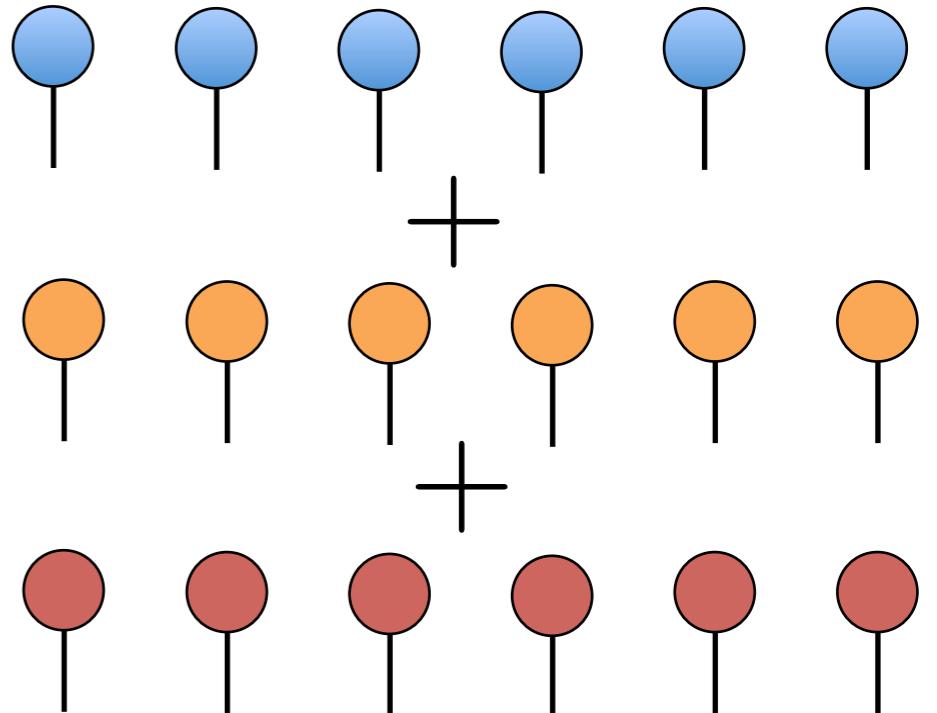
reminiscent of the *mean-field* approximation

$$p(\{x_1, x_2, \dots, x_n\}) = p(x_1)p(x_2)\dots p(x_n)$$



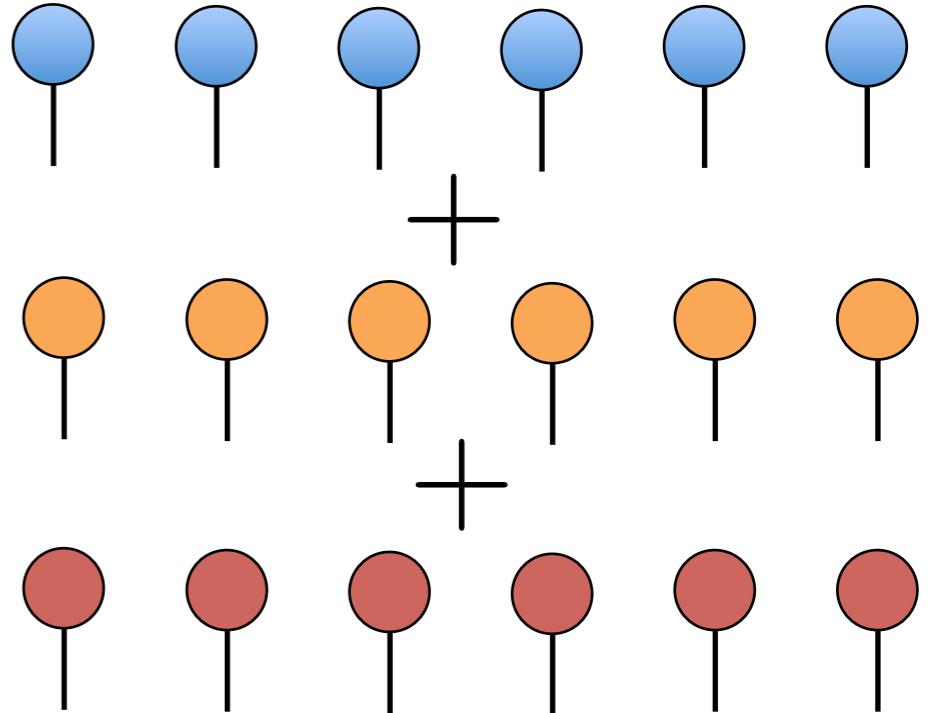
# Canonical Polyadic (CP) decomposition

$$\begin{aligned}
 \mathcal{A} &= \sum_{r=1}^{\chi} \lambda_r \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(n)} \\
 &= \boldsymbol{\Lambda} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_n \mathbf{A}^{(n)} \\
 &= [\boldsymbol{\Lambda}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(n)}]
 \end{aligned}$$



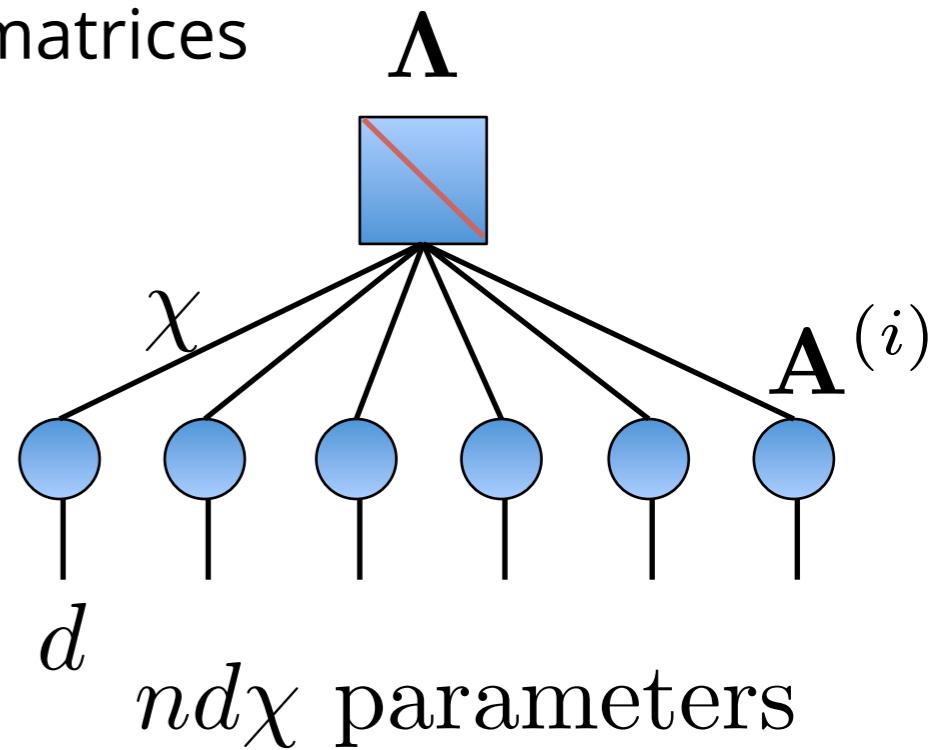
# Canonical Polyadic (CP) decomposition

$$\begin{aligned}
 \mathcal{A} &= \sum_{r=1}^{\chi} \lambda_r \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(n)} \\
 &= \boldsymbol{\Lambda} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_n \mathbf{A}^{(n)} \\
 &= [\boldsymbol{\Lambda}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(n)}]
 \end{aligned}$$

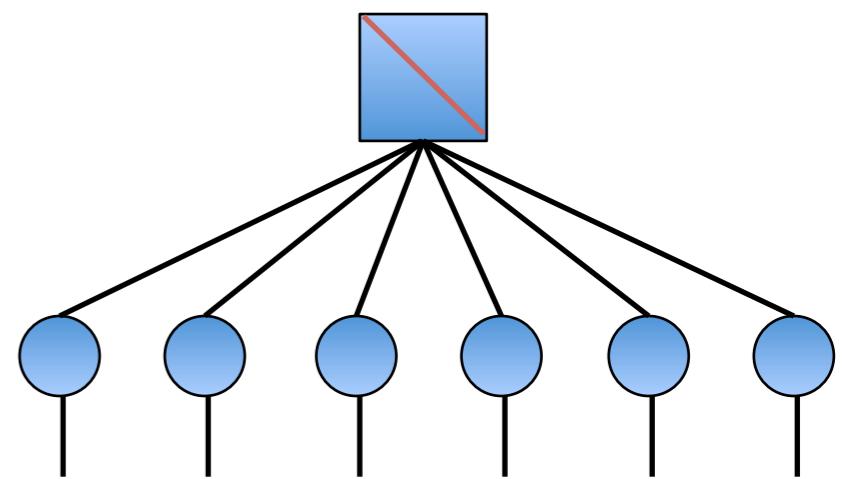
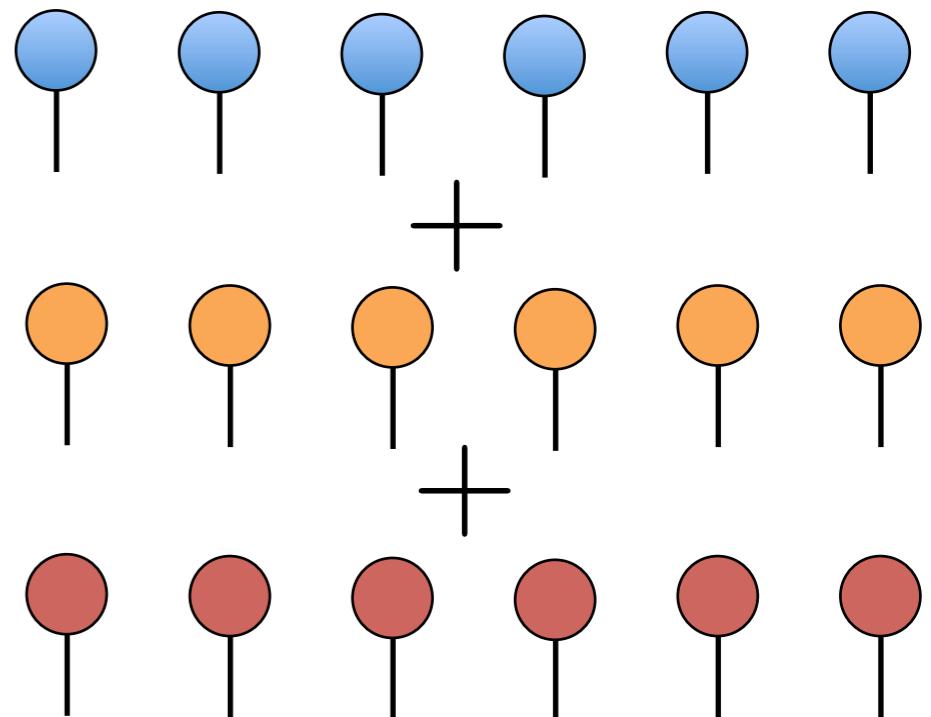


Convenient representations of CP using unfolded matrices

$$\mathbf{A}_{(i)} = \mathbf{A}^{(i)} \boldsymbol{\Lambda} \left( \mathbf{A}^{(1)} \odot \mathbf{A}^{(2)} \odot \cdots \odot \mathbf{A}^{(n)} \right)$$

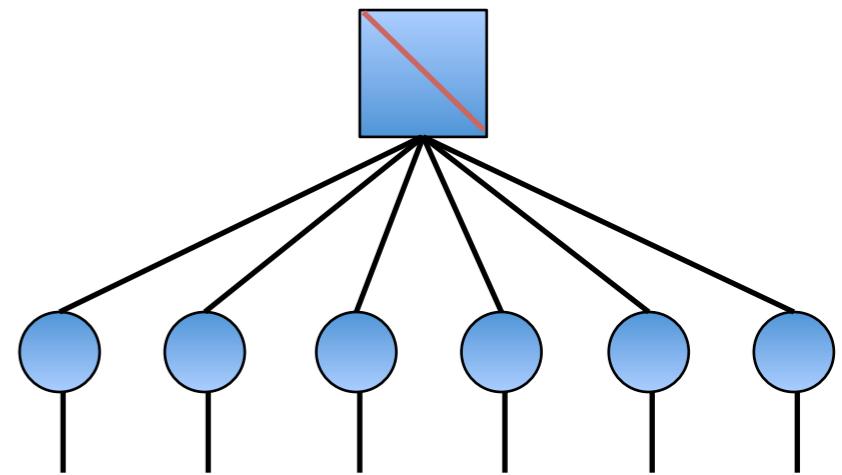
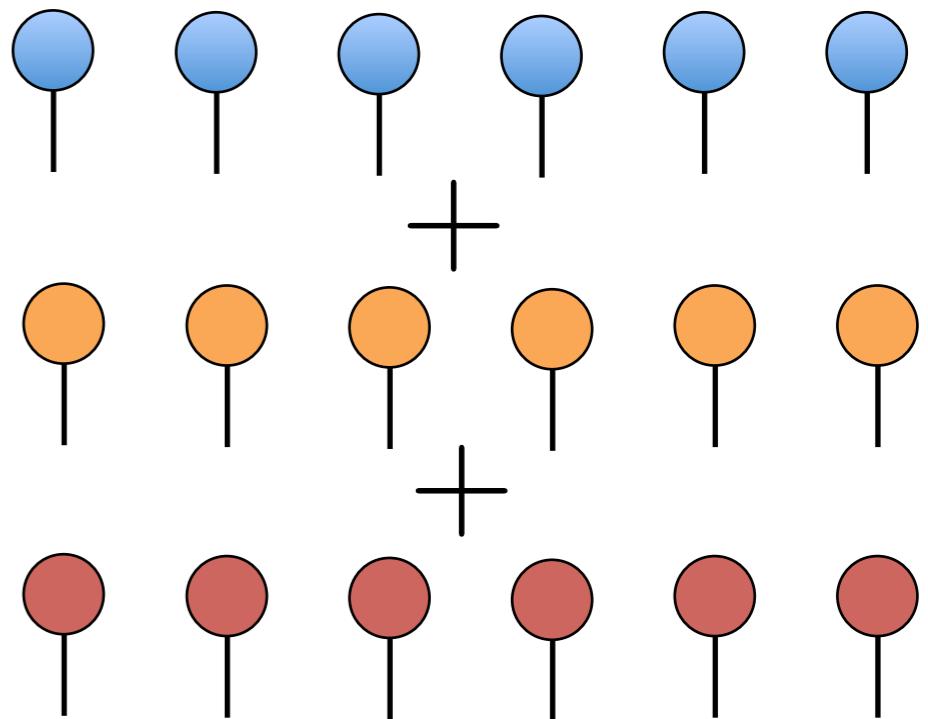


# Canonical Polyadic (CP) decomposition



# Canonical Polyadic (CP) decomposition

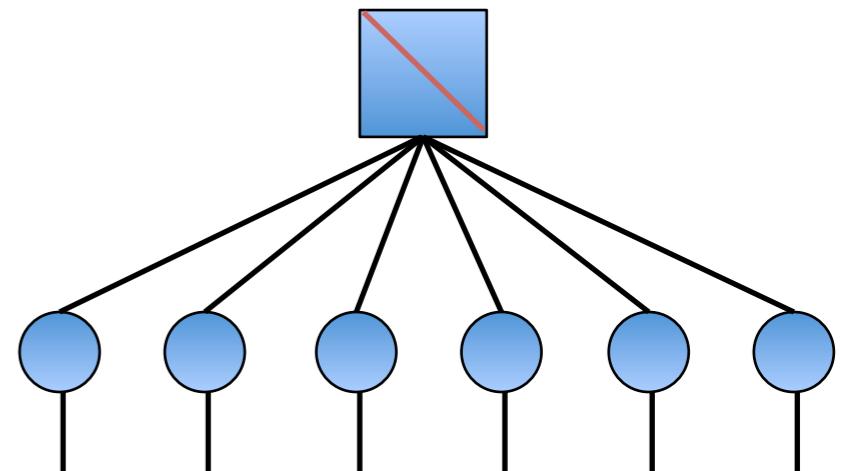
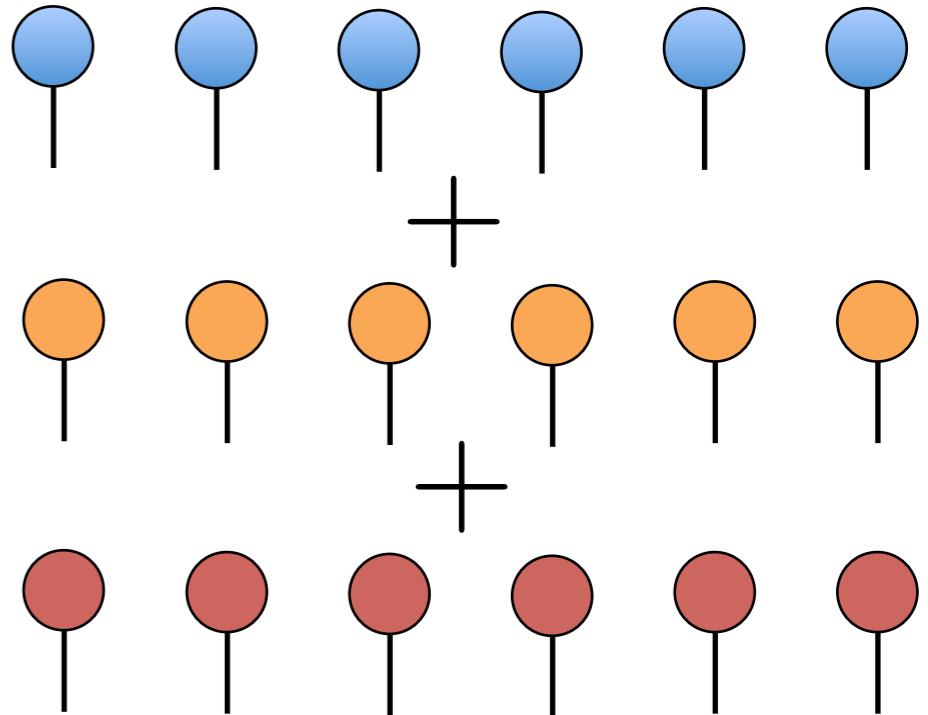
Also known as *Polyadic*, *PARAFAC*, *CANDECOMP*,  
and *Topographic*.



# Canonical Polyadic (CP) decomposition

Also known as *Polyadic*, *PARAFAC*, *CANDECOMP*, and *Topographic*.

First proposed in 1927 by *Hitchcock*, then became popular in 1970's in psychometric and in 1980's in chemometrics.

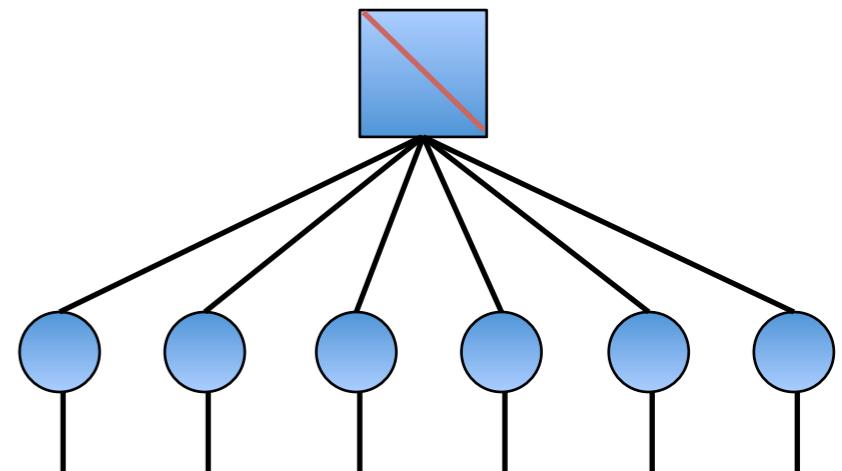
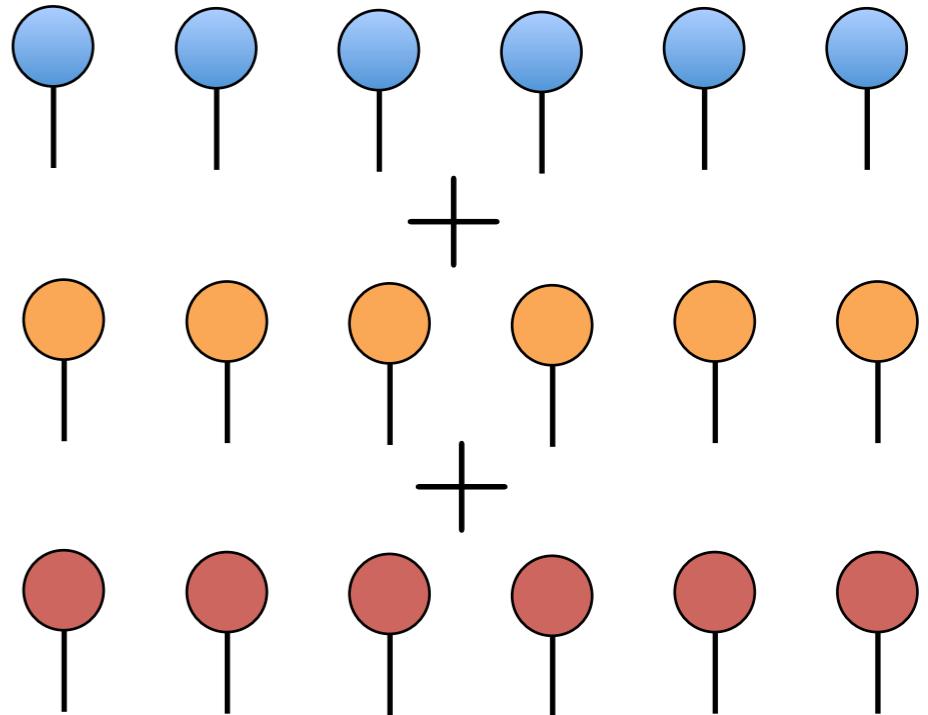


# Canonical Polyadic (CP) decomposition

Also known as *Polyadic*, *PARAFAC*, *CANDECOMP*, and *Topographic*.

First proposed in 1927 by *Hitchcock*, then became popular in 1970's in psychometric and in 1980's in chemometrics.

Found applications in signal processing, neural sciences, signal processing in 1990's, and image compression and classification from 2005.



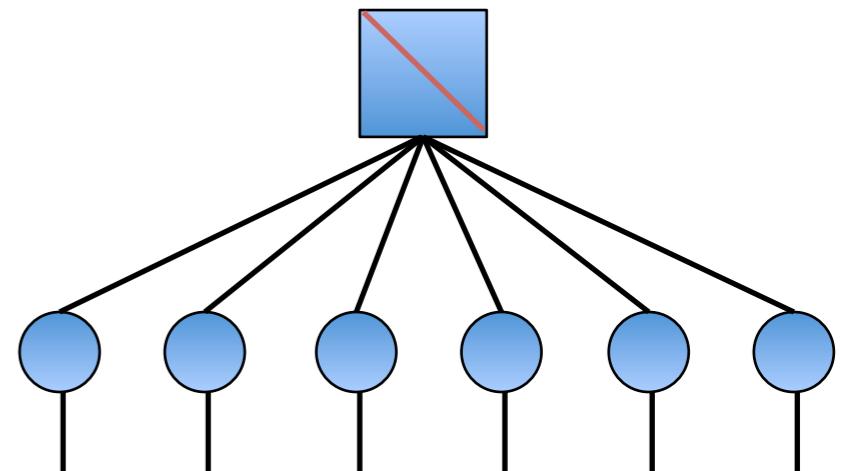
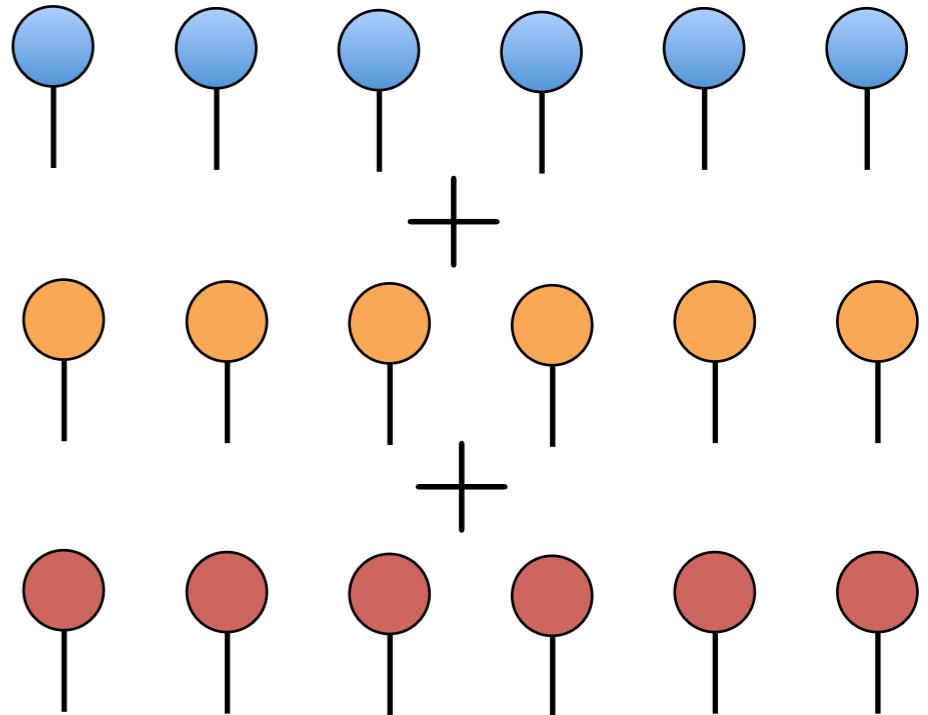
# Canonical Polyadic (CP) decomposition

Also known as *Polyadic*, *PARAFAC*, *CANDECOMP*, and *Topographic*.

First proposed in 1927 by *Hitchcock*, then became popular in 1970's in psychometric and in 1980's in chemometrics.

Found applications in signal processing, neural sciences, signal processing in 1990's, and image compression and classification from 2005.

Any tensor can be expressed as *finite* sum of rank-1 tensors.



# Canonical Polyadic (CP) decomposition

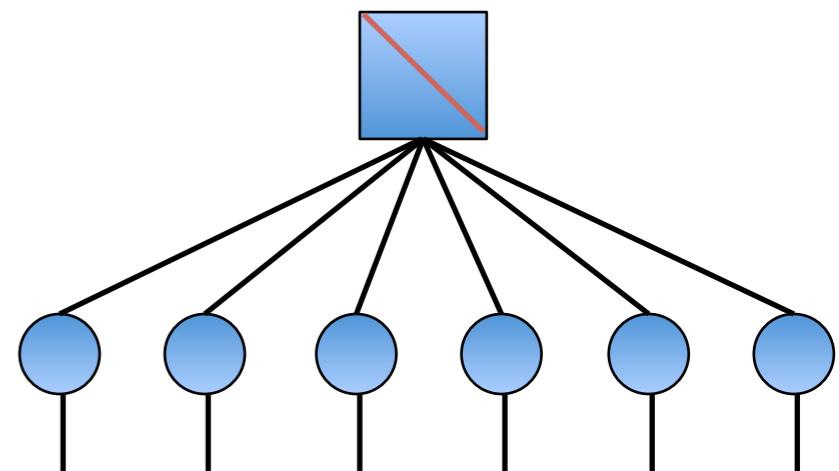
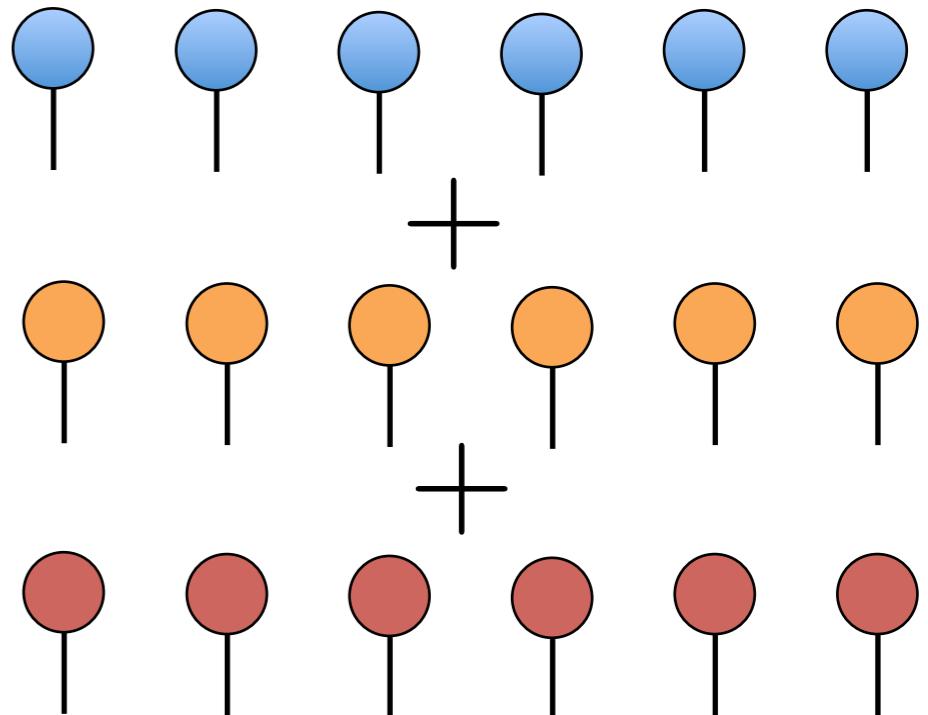
Also known as *Polyadic*, *PARAFAC*, *CANDECOMP*, and *Topographic*.

First proposed in 1927 by *Hitchcock*, then became popular in 1970's in psychometric and in 1980's in chemometrics.

Found applications in signal processing, neural sciences, signal processing in 1990's, and image compression and classification from 2005.

Any tensor can be expressed as *finite* sum of rank-1 tensors.

Tensor rank, or CP-rank, is defined as a *minimum number of rank-1 tensors* in the exact CP decomposition.



# Canonical Polyadic (CP) decomposition

Also known as *Polyadic*, *PARAFAC*, *CANDECOMP*, and *Topographic*.

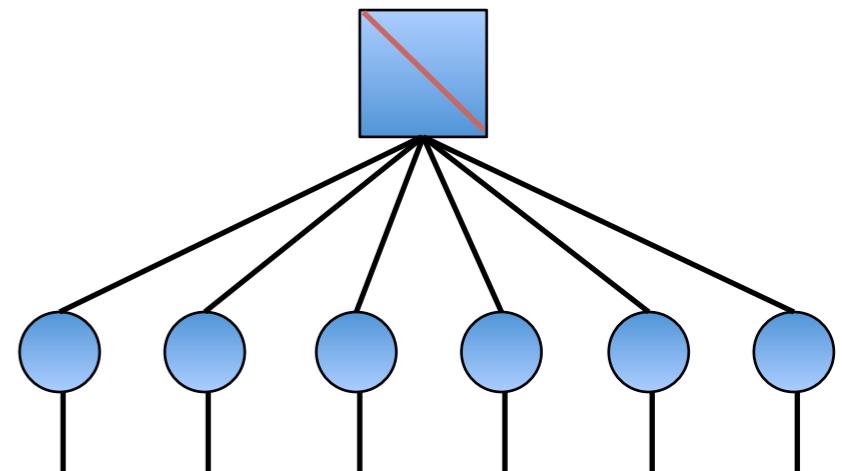
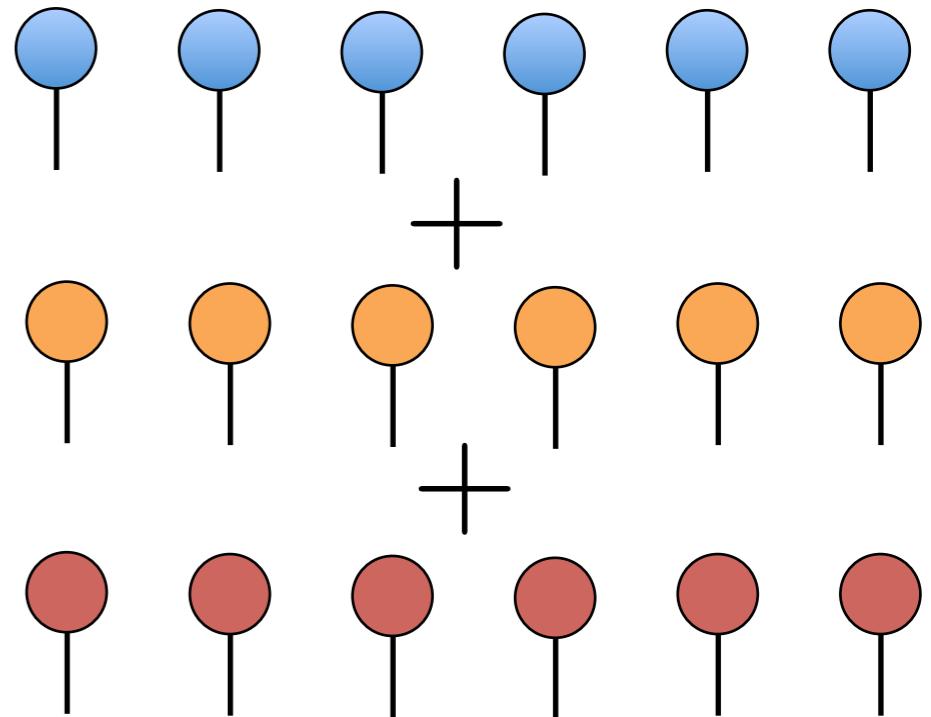
First proposed in 1927 by *Hitchcock*, then became popular in 1970's in psychometric and in 1980's in chemometrics.

Found applications in signal processing, neural sciences, signal processing in 1990's, and image compression and classification from 2005.

Any tensor can be expressed as *finite* sum of rank-1 tensors.

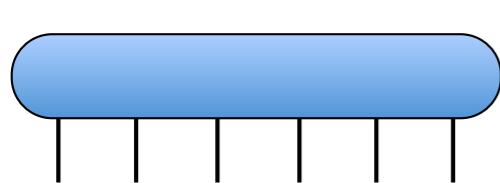
Tensor rank, or CP-rank, is defined as a *minimum number of rank-1 tensors* in the exact CP decomposition.

However, determining the CP-rank is *NP-hard* problem, finding the best CP decomposition is *hard*, yielding many local minimums.

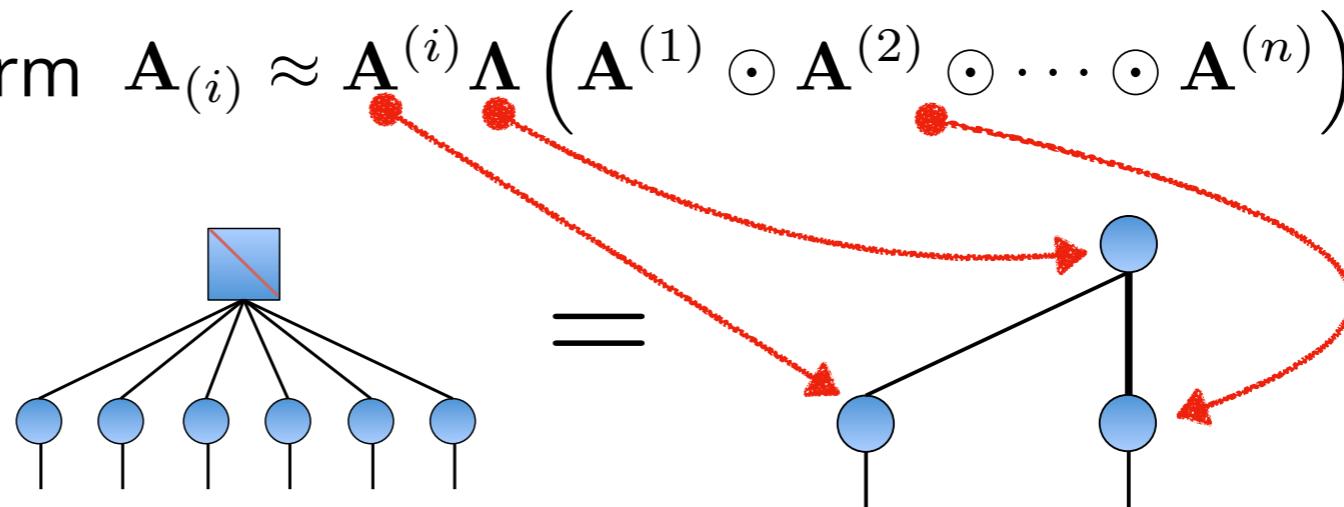


# The workhorse algorithm: Alternating Least Square

In the un-folded matrix form  $\mathbf{A}_{(i)} \approx \mathbf{A}^{(i)} \Lambda \left( \mathbf{A}^{(1)} \odot \mathbf{A}^{(2)} \odot \dots \odot \mathbf{A}^{(n)} \right)$

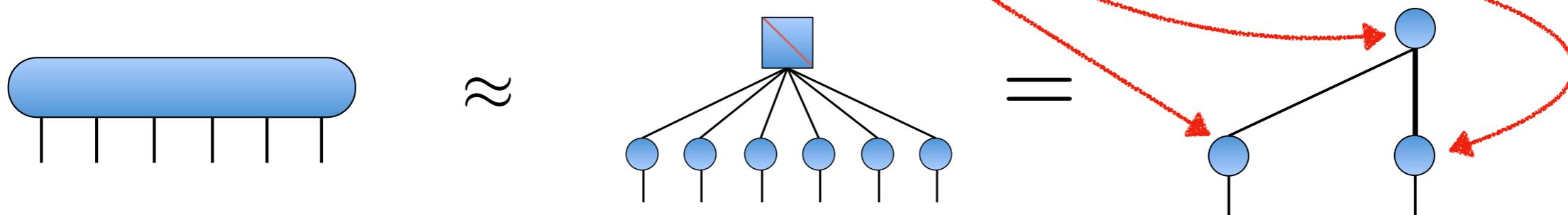


$\approx$

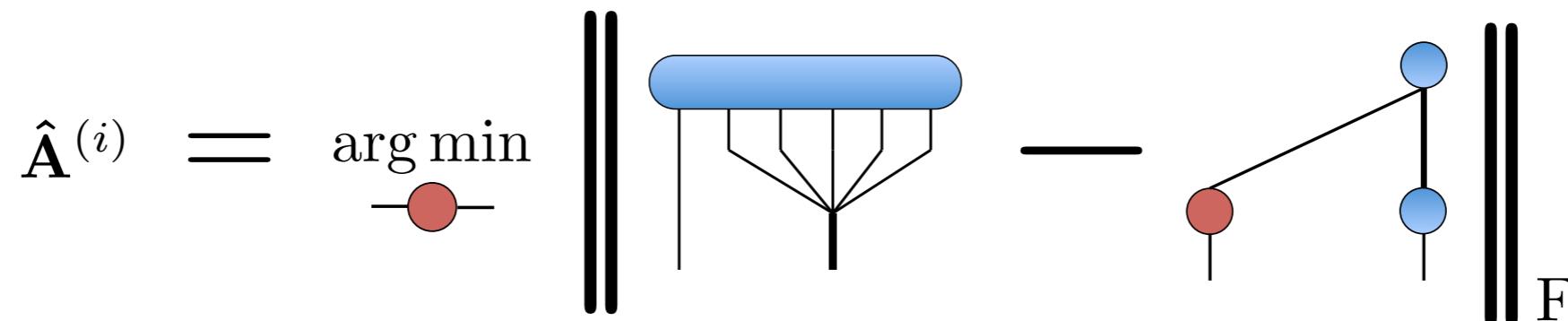


# The workhorse algorithm: Alternating Least Square

In the un-folded matrix form  $\mathbf{A}_{(i)} \approx \mathbf{A}^{(i)} \Lambda \left( \mathbf{A}^{(1)} \odot \mathbf{A}^{(2)} \odot \dots \odot \mathbf{A}^{(n)} \right)$

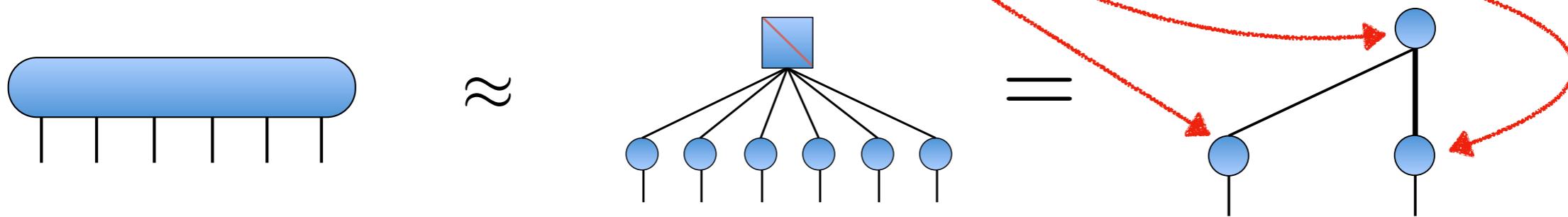


The ALS fixes all except one matrix for optimizing, the sub-problems is a least-square problem

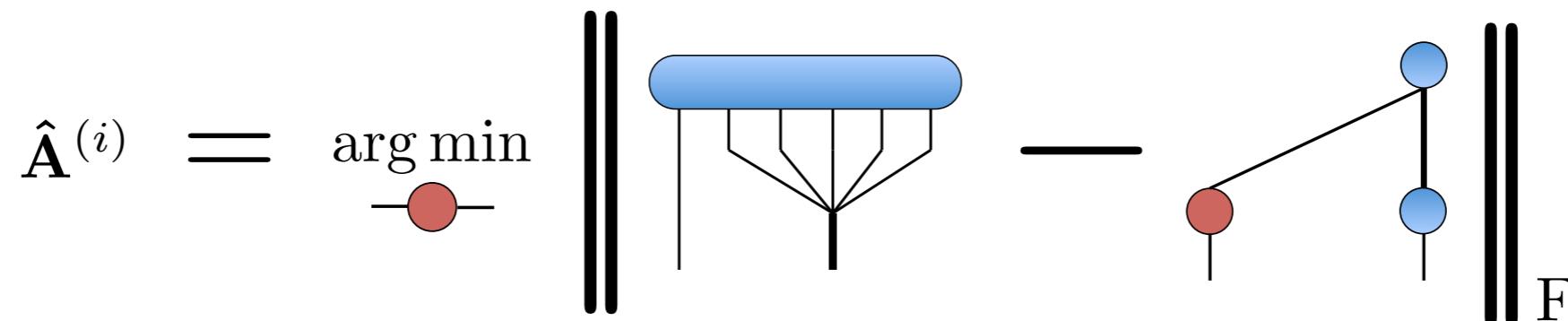


# The workhorse algorithm: Alternating Least Square

In the un-folded matrix form  $\mathbf{A}_{(i)} \approx \mathbf{A}^{(i)} \Lambda \left( \mathbf{A}^{(1)} \odot \mathbf{A}^{(2)} \odot \dots \odot \mathbf{A}^{(n)} \right)$



The ALS fixes all except one matrix for optimizing, the sub-problems is a least-square problem

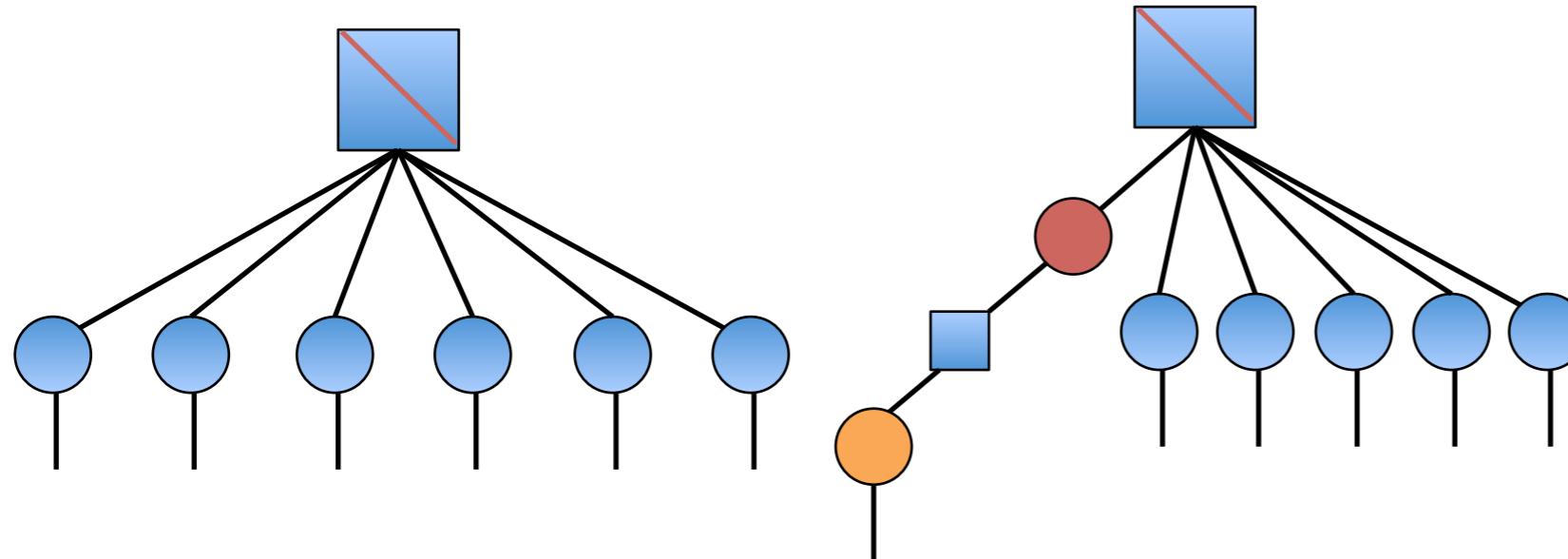


Solved simply using pseudo-inverse

$$\hat{\mathbf{A}}^{(i)} \leftarrow \mathbf{A}^{(i)} \left( \Lambda \mathbf{A}^{(1)} \odot \mathbf{A}^{(2)} \odot \dots \odot \mathbf{A}^{(i-1)} \odot \mathbf{A}^{(i+1)} \dots \odot \mathbf{A}^{(n)} \right)^+$$

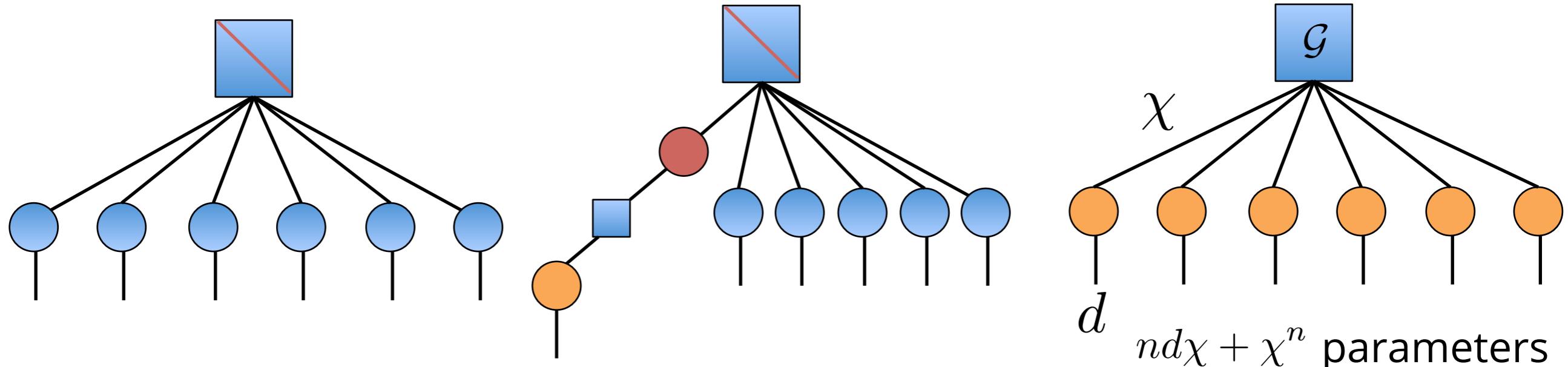
# From CP to Tucker

The CP formats is lack of canonical forms and orthogonalization  
by imposing orthogonalization, the core tensor is no more diagonal



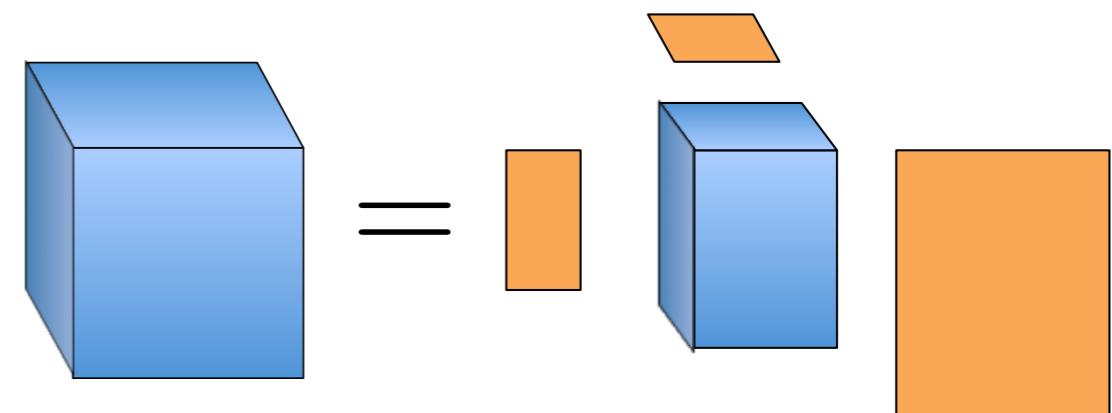
# From CP to Tucker

The CP formats is lack of canonical forms and orthogonalization  
by imposing orthogonalization, the core tensor is no more diagonal



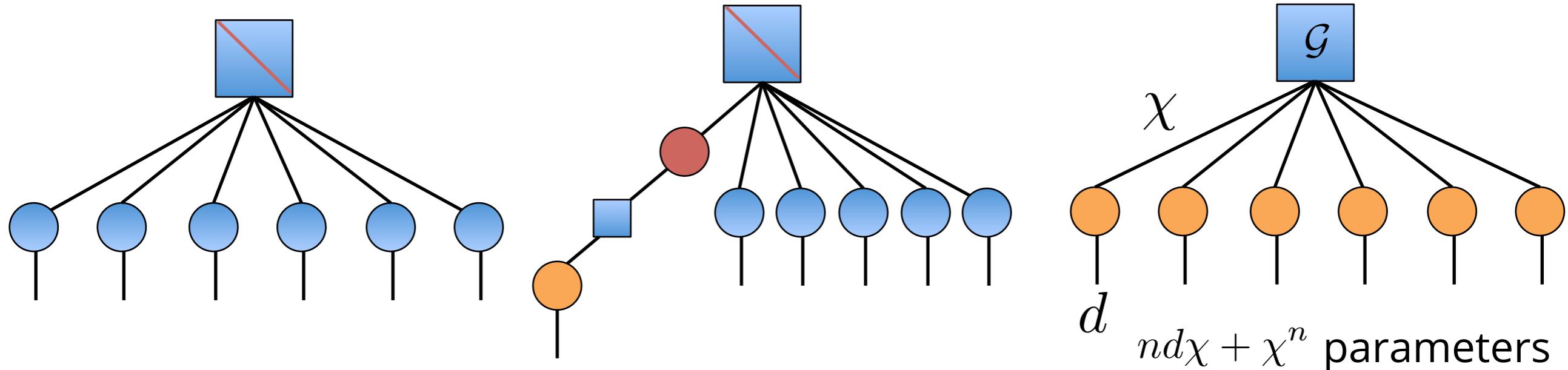
this leads to the *Tucker Decomposition*

$$\begin{aligned}\mathcal{A} &= [\mathcal{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(n)}] \\ &= \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_n \mathbf{A}^{(n)}\end{aligned}$$



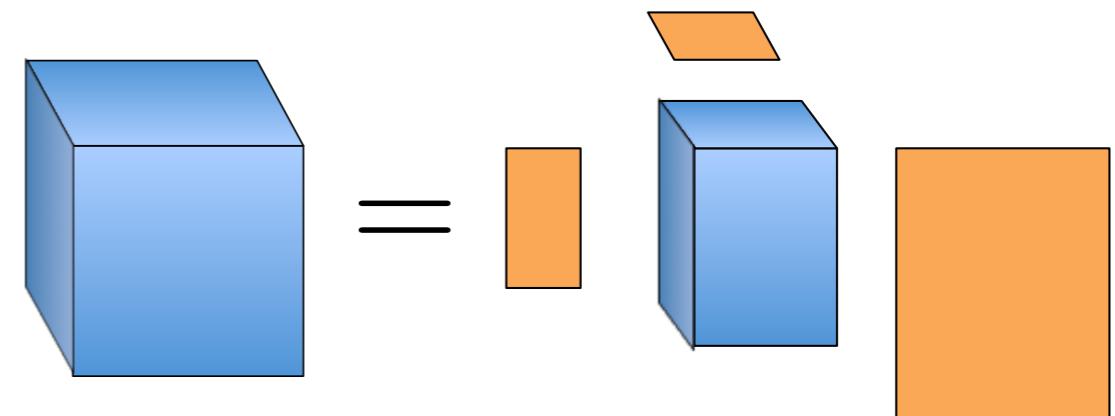
# From CP to Tucker

The CP formats is lack of canonical forms and orthogonalization by imposing orthogonalization, the core tensor is no more diagonal



this leads to the *Tucker Decomposition*

$$\begin{aligned}\mathcal{A} &= [\mathcal{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(n)}] \\ &= \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_n \mathbf{A}^{(n)}\end{aligned}$$



First introduced by Tucker in 1963, known as *N-mode PCA/SVD/Factor analysis*. Usually treated as a multilinear extension of PCA

# Applications of the Tucker decomposition

Data compression

K-means clustering

anomaly detection

matrix completion

computer Vision: TensorFace, hand-written digits classification...

high-order compressed sensing (using compositional dictionary)

# Applications of the Tucker decomposition

Data compression

K-means clustering

anomaly detection

matrix completion

computer Vision: TensorFace, hand-written digits classification...

high-order compressed sensing (using compositional dictionary)

$$y = F \times x$$

A diagram illustrating the Tucker decomposition. On the left, a vertical vector  $y$  is shown as a stack of four colored blocks: red, green, blue, and purple. In the center, a matrix  $F$  is shown as a 4x9 grid of colored blocks. To the right of  $F$  is a multiplication sign ( $\times$ ). On the far right, a vertical vector  $x$  is shown as a stack of six colored blocks: white, blue, red, black, orange, and white.

# Applications of the Tucker decomposition

Data compression

K-means clustering

anomaly detection

matrix completion

computer Vision: TensorFace, hand-written digits classification...

high-order compressed sensing (using compositional dictionary)

$$y = F \times x$$
$$\mathbf{F} = \mathbf{F}^{(1)} \otimes \mathbf{F}^{(2)} \otimes \dots \otimes \mathbf{F}^{(n)}$$

# Applications of the Tucker decomposition

Data compression

K-means clustering

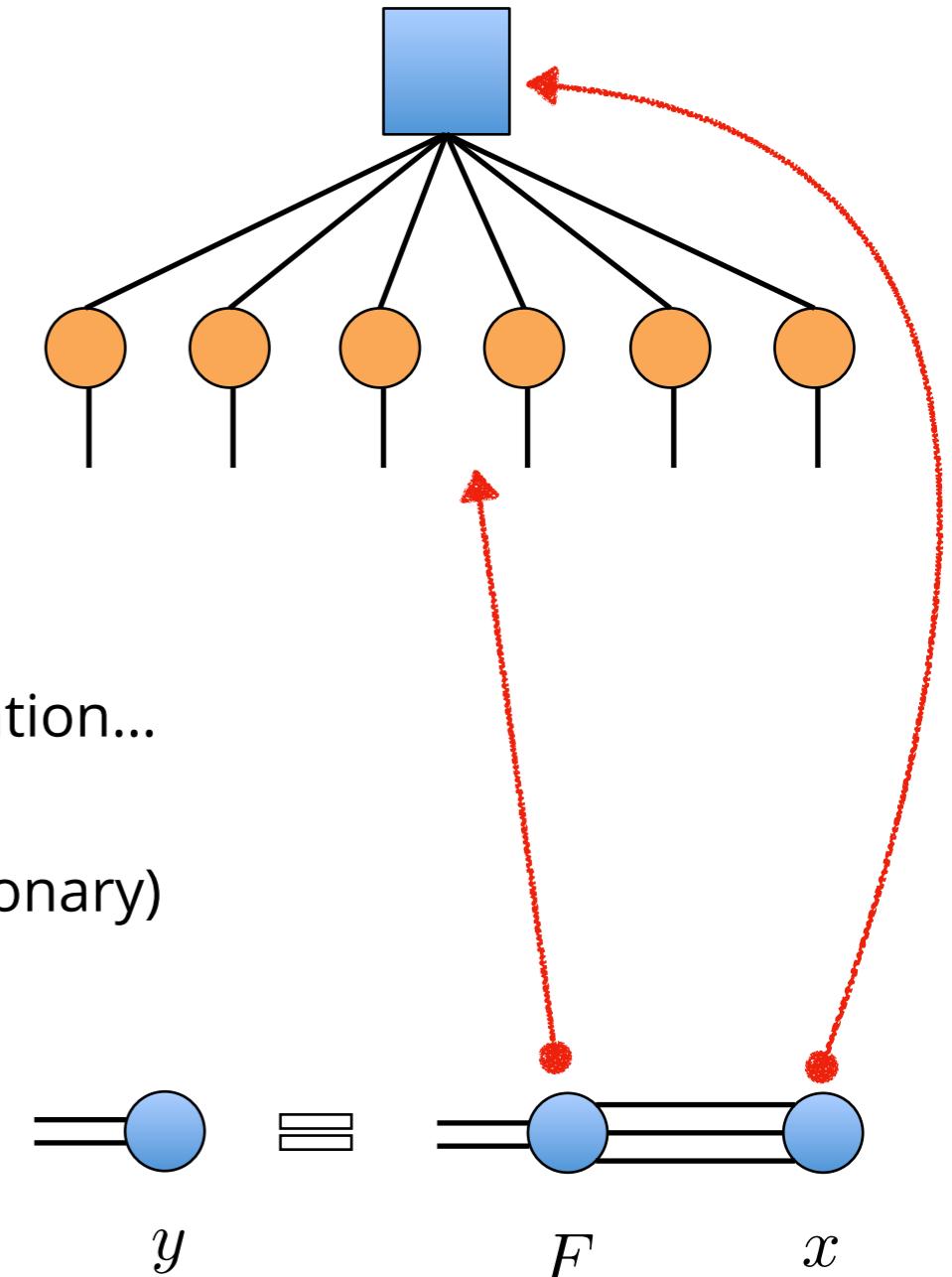
anomaly detection

matrix completion

computer Vision: TensorFace, hand-written digits classification...

high-order compressed sensing (using compositional dictionary)

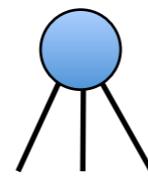
$$y = F \times x$$
$$\mathbf{F} = \mathbf{F}^{(1)} \otimes \mathbf{F}^{(2)} \otimes \dots \otimes \mathbf{F}^{(n)}$$



# High order singular value decomposition

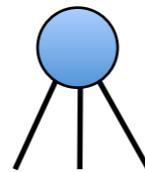
# High order singular value decomposition

given a raw tensor  $\mathcal{A}$

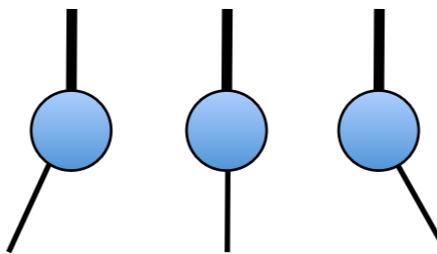


# High order singular value decomposition

given a raw tensor  $\mathcal{A}$

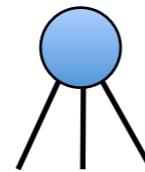


unfold to matrix at each mode

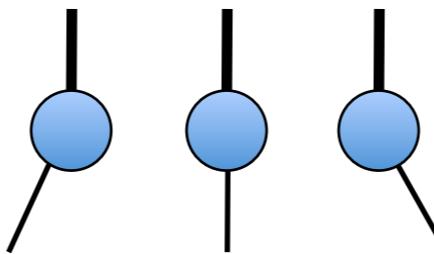


# High order singular value decomposition

given a raw tensor  $\mathcal{A}$

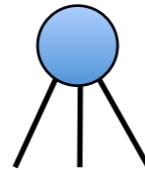


unfold to matrix at each mode

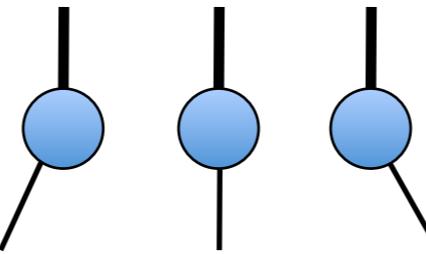


# High order singular value decomposition

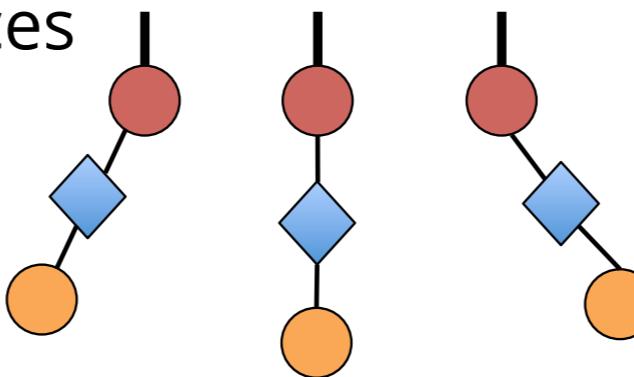
given a raw tensor  $\mathcal{A}$



unfold to matrix at each mode

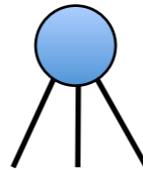


do SVD on obtained matrices

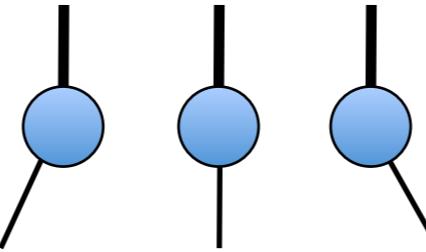


# High order singular value decomposition

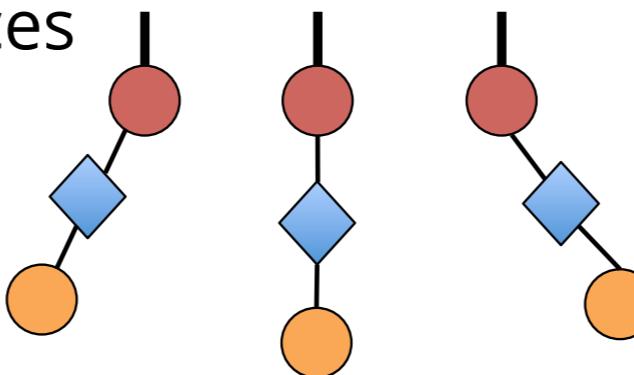
given a raw tensor  $\mathcal{A}$



unfold to matrix at each mode



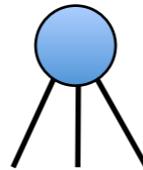
do SVD on obtained matrices



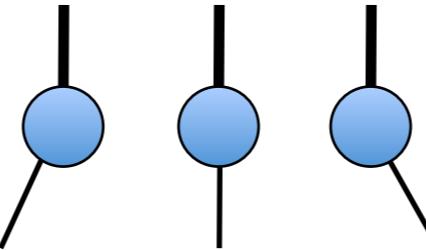
contract A to the inverse of the obtained column orthogonal matrices,  
to obtain the core  $\mathcal{G}$ .

# High order singular value decomposition

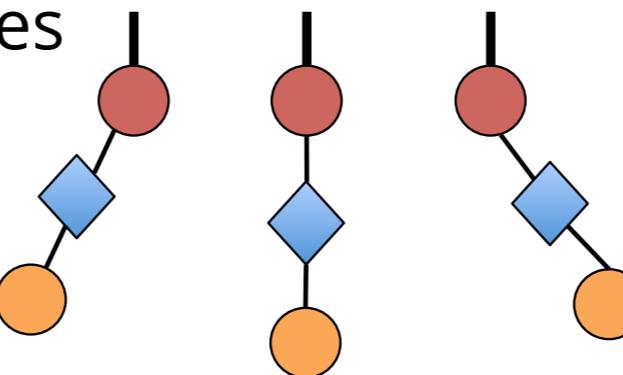
given a raw tensor  $\mathcal{A}$



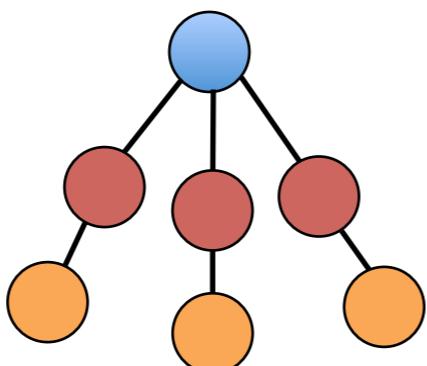
unfold to matrix at each mode



do SVD on obtained matrices

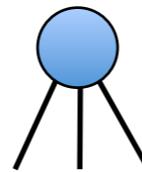


contract A to the inverse of the obtained column orthogonal matrices,  
to obtain the core  $\mathcal{G}$ .

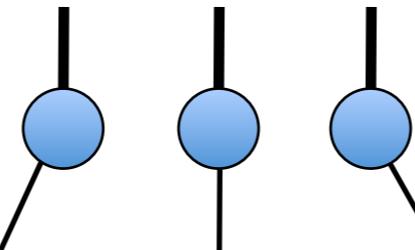


# High order singular value decomposition

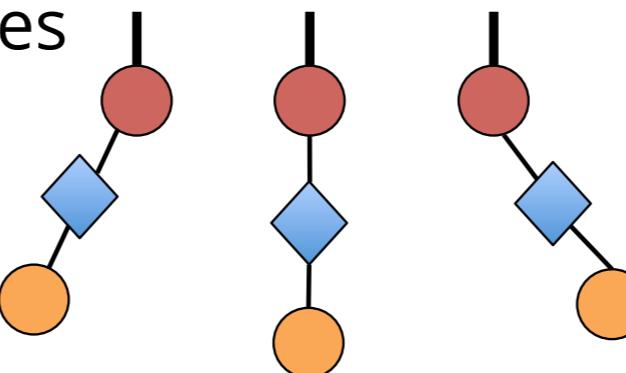
given a raw tensor  $\mathcal{A}$



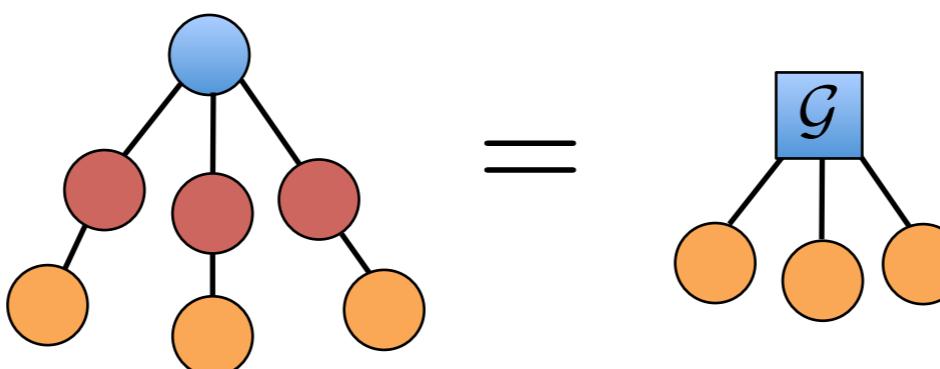
unfold to matrix at each mode



do SVD on obtained matrices



contract A to the inverse of the obtained column orthogonal matrices,  
to obtain the core  $\mathcal{G}$ .



# Improving HOSVD

HOSVD only gives a quasi-best approximation

$$\left\| \mathcal{A} - [\mathcal{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(n)}] \right\|_F \leq \sqrt{n} \|\mathcal{A} - \mathcal{A}_{\text{best}}\|_F$$

High Order Orthogonal Iteration (HOOI) interactively refines factor matrices using SVDs.

Elden and Savas's method contains factor matrices to a Grassmann manifold which defines an equivalent class of orthogonal matrices, then optimize them using Newton's method.

# Improving HOSVD

HOSVD only gives a quasi-best approximation

$$\left\| \mathcal{A} - [\mathcal{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(n)}] \right\|_F \leq \sqrt{n} \|\mathcal{A} - \mathcal{A}_{\text{best}}\|_F$$

High Order Orthogonal Iteration (HOOI) interactively refines factor matrices using SVDs.

Elden and Savas's method contains factor matrices to a Grassmann manifold which defines an equivalent class of orthogonal matrices, then optimize them using Newton's method.

limitation:

$nd\chi + \chi^n$  parameters, *not parameter efficient for  $n$  large*

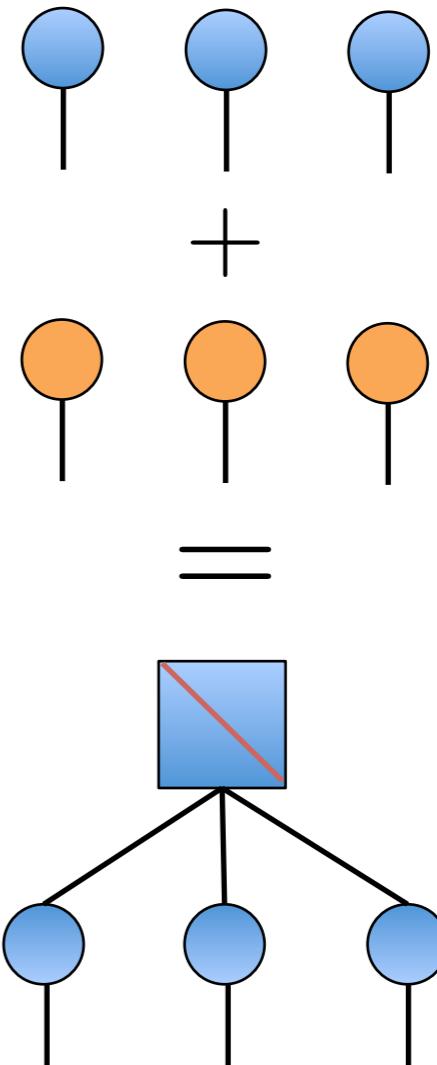
# From CP to MPS

Summing two rank-1 tensors

$$\mathcal{A} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \mathbf{a}^{(3)}$$

$$\mathcal{B} = \mathbf{b}^{(1)} \circ \mathbf{b}^{(2)} \circ \mathbf{b}^{(3)}$$

$$\mathcal{C} = \mathcal{A} + \mathcal{B}$$



# From CP to MPS

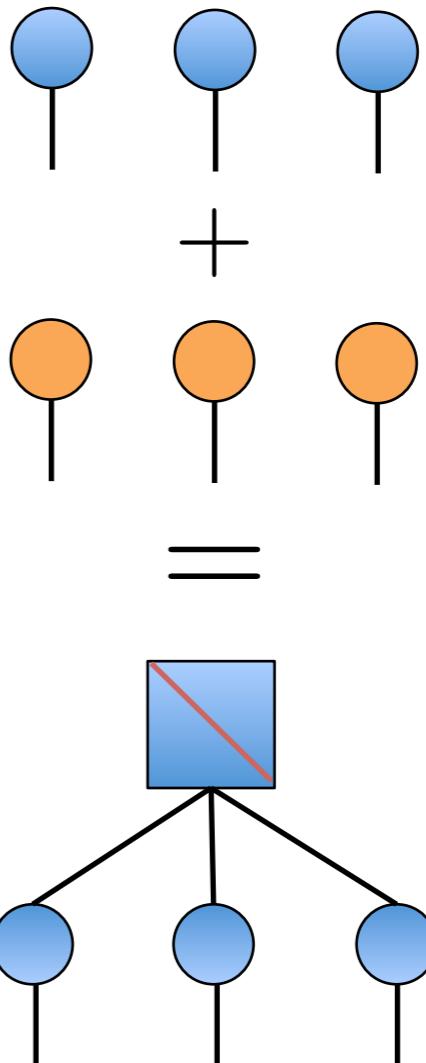
Summing two rank-1 tensors

$$\mathcal{A} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \mathbf{a}^{(3)}$$

$$\mathcal{B} = \mathbf{b}^{(1)} \circ \mathbf{b}^{(2)} \circ \mathbf{b}^{(3)}$$

$$\mathcal{C} = \mathcal{A} + \mathcal{B}$$

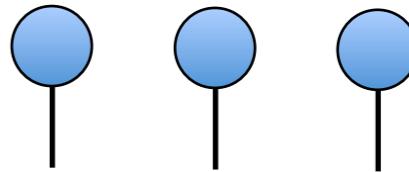
$$\begin{aligned} c_{s_1, s_2, s_3} &= a_{s_1}^{(1)} a_{s_2}^{(2)} a_{s_3}^{(3)} + b_{s_1}^{(1)} b_{s_2}^{(2)} b_{s_3}^{(3)} \\ &= \begin{pmatrix} a_{s_1}^{(1)} & b_{s_1}^{(1)} \end{pmatrix} \begin{pmatrix} a_{s_2}^{(2)} & 0 \\ 0 & b_{s_2}^{(2)} \end{pmatrix} \begin{pmatrix} a_{s_3}^{(3)} \\ b_{s_3}^{(3)} \end{pmatrix} \end{aligned}$$



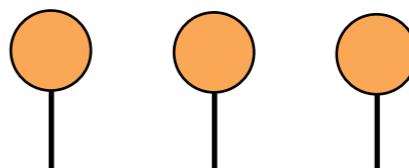
# From CP to MPS

Summing two rank-1 tensors

$$\mathcal{A} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \mathbf{a}^{(3)}$$



+

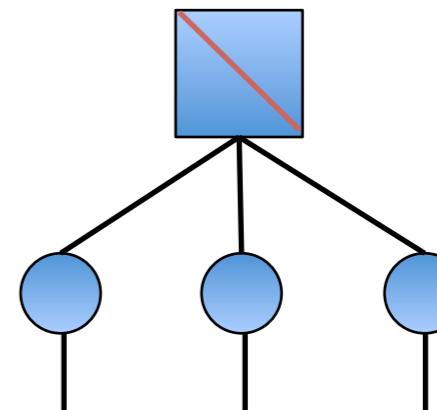


=

$$\mathcal{C} = \mathcal{A} + \mathcal{B}$$

$$c_{s_1, s_2, s_3} = a_{s_1}^{(1)} a_{s_2}^{(2)} a_{s_3}^{(3)} + b_{s_1}^{(1)} b_{s_2}^{(2)} b_{s_3}^{(3)}$$

$$= \begin{pmatrix} a_{s_1}^{(1)} & b_{s_1}^{(1)} \end{pmatrix} \begin{pmatrix} a_{s_2}^{(2)} & 0 \\ 0 & b_{s_2}^{(2)} \end{pmatrix} \begin{pmatrix} a_{s_3}^{(3)} \\ b_{s_3}^{(3)} \end{pmatrix}$$

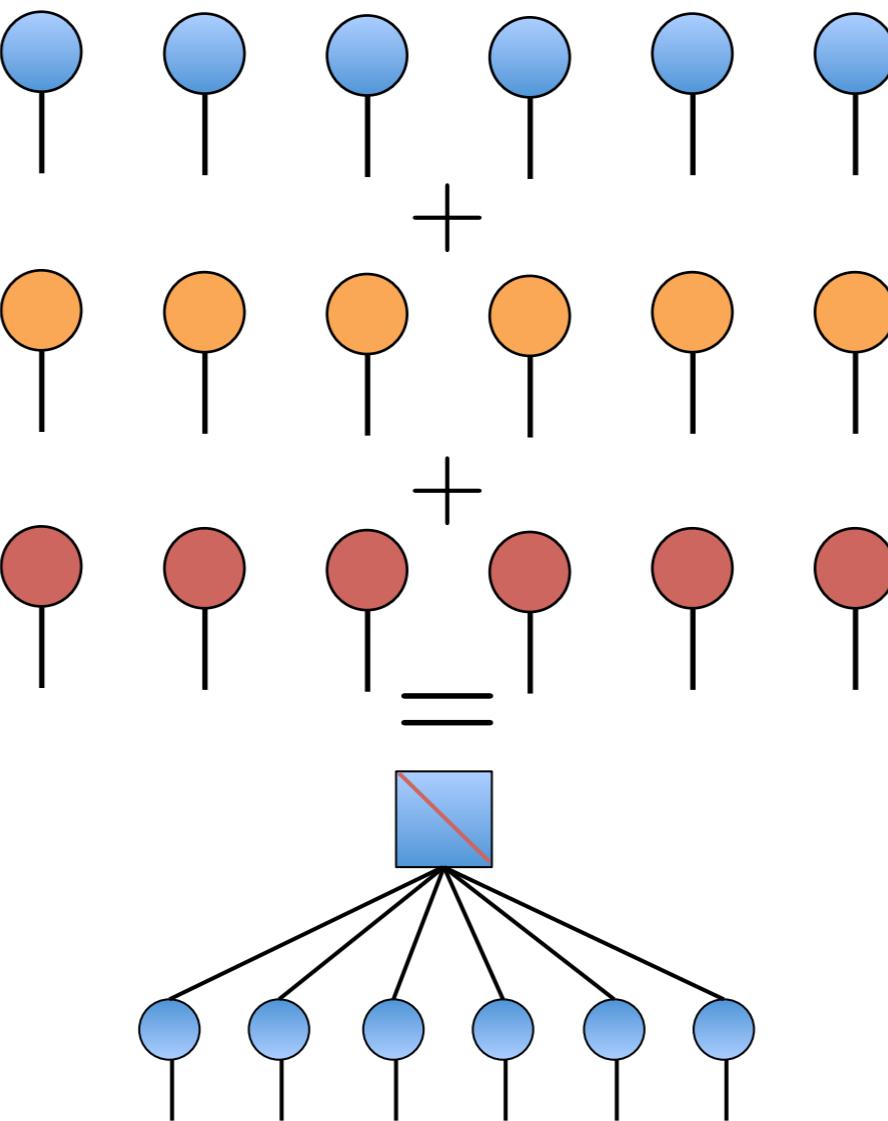


$$\mathbf{C}^{(1)}(s_1, :) \quad \mathcal{C}^{(2)}(:, s_2, :) \quad \mathbf{C}^{(3)}(:, s_3)$$

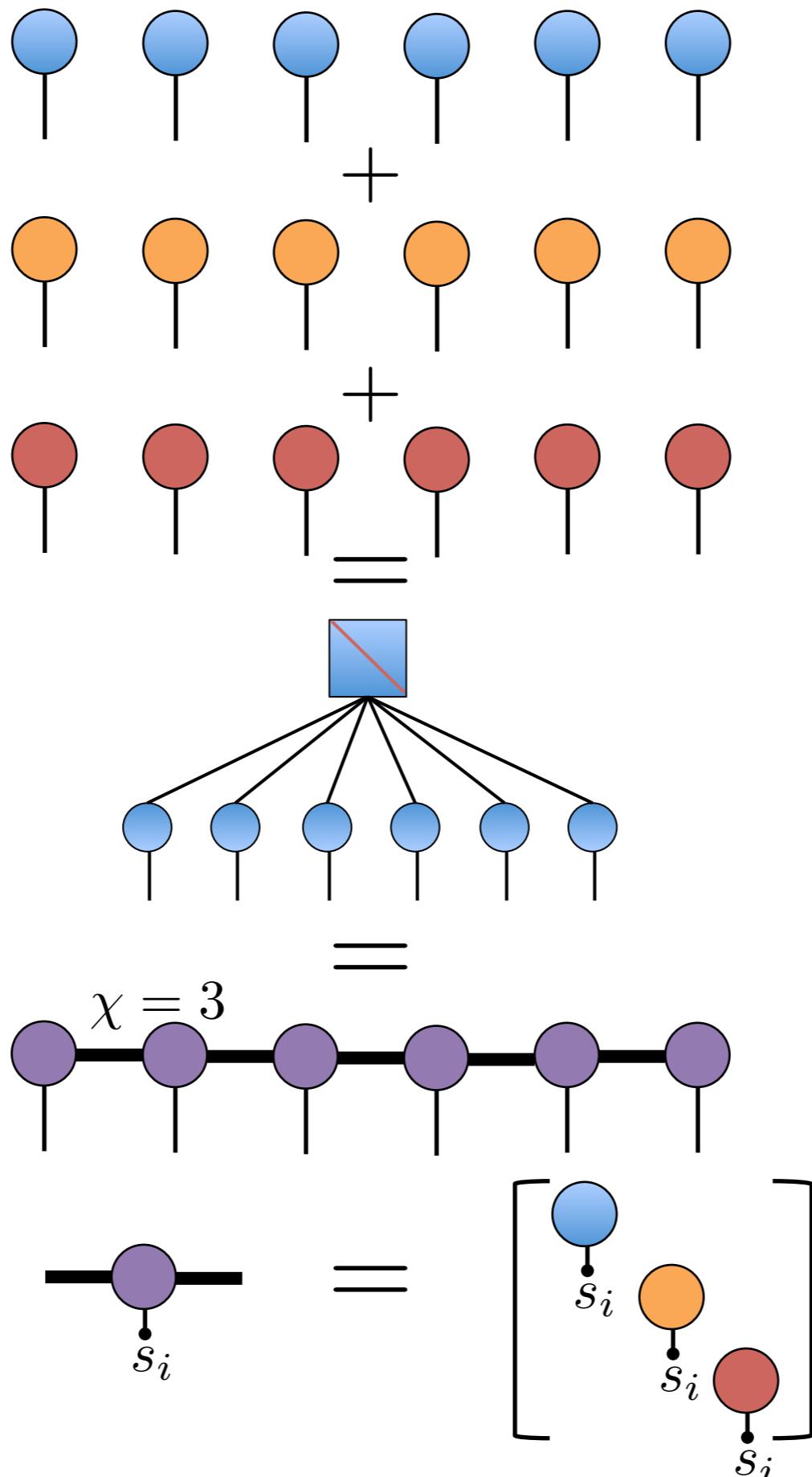
=



# From CP to MPS



# From CP to MPS

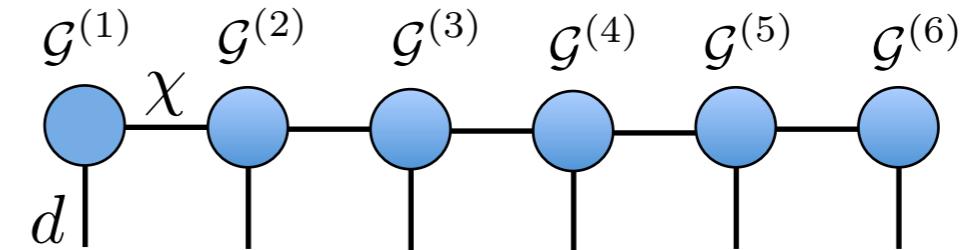


# Matrix Product States and Matrix Product Operator

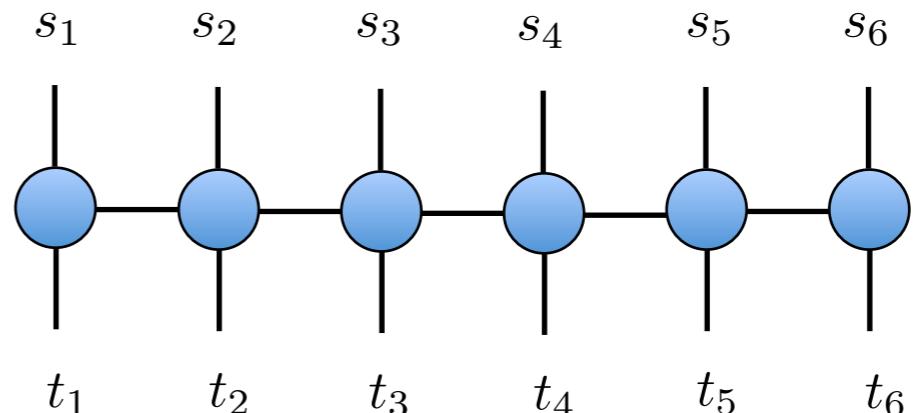
$$\begin{aligned}\mathcal{A} &= \langle \mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(n)} \rangle \\ &= \mathcal{G}^{(1)} \times_1 \mathcal{G}^{(2)} \times_1 \cdots \times_1 \mathcal{G}^{(n)}\end{aligned}$$

$$a_{s_1, s_2, \dots, s_n} = \mathbf{G}_{s_1}^{(1)} \mathbf{G}_{s_2}^{(2)} \cdots \mathbf{G}_{s_n}^{(n)}$$

$$a_{s_1, s_2, \dots, s_n} = \mathbf{G}_{s_1, t_1}^{(1)} \mathbf{G}_{s_2, t_2}^{(2)} \cdots \mathbf{G}_{s_n, t_n}^{(n)}$$



$\approx nd\chi^2$  parameters

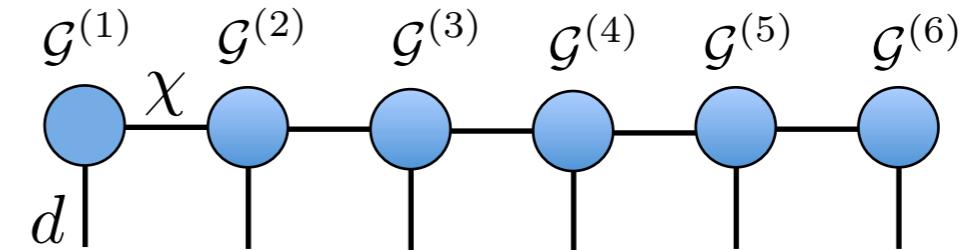


# Matrix Product States and Matrix Product Operator

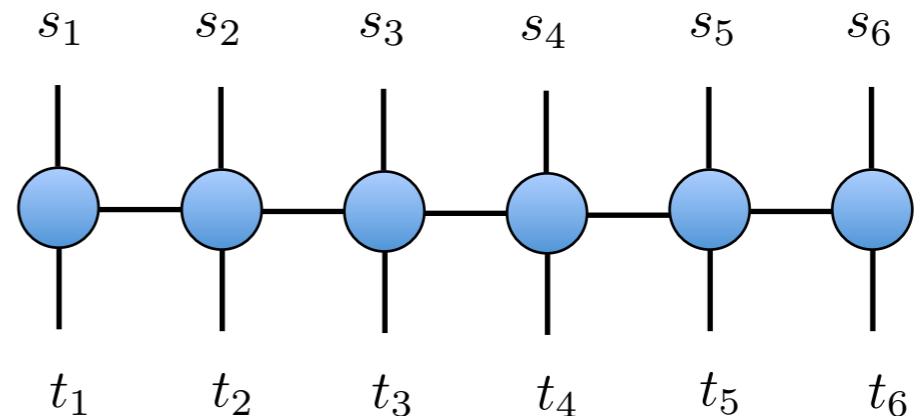
$$\begin{aligned}\mathcal{A} &= \langle \mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(n)} \rangle \\ &= \mathcal{G}^{(1)} \times_1 \mathcal{G}^{(2)} \times_1 \cdots \times_1 \mathcal{G}^{(n)}\end{aligned}$$

$$a_{s_1, s_2, \dots, s_n} = \mathbf{G}_{s_1}^{(1)} \mathbf{G}_{s_2}^{(2)} \cdots \mathbf{G}_{s_n}^{(n)}$$

$$a_{s_1, s_2, \dots, s_n} = \mathbf{G}_{s_1, t_1}^{(1)} \mathbf{G}_{s_2, t_2}^{(2)} \cdots \mathbf{G}_{s_n, t_n}^{(n)}$$



$\approx nd\chi^2$  parameters



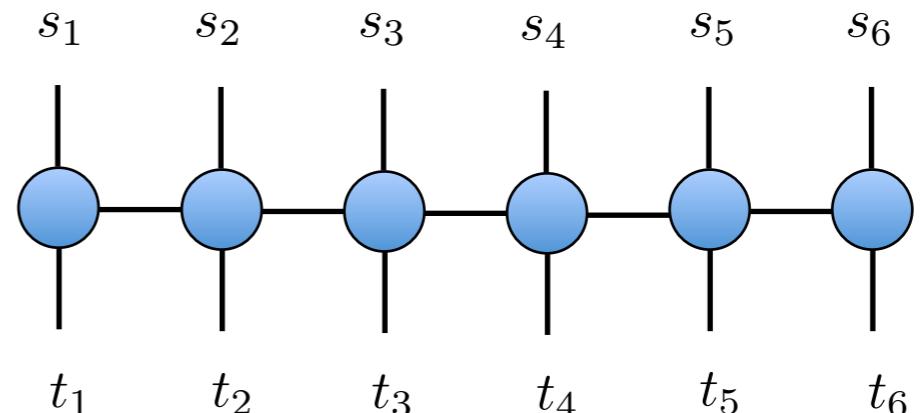
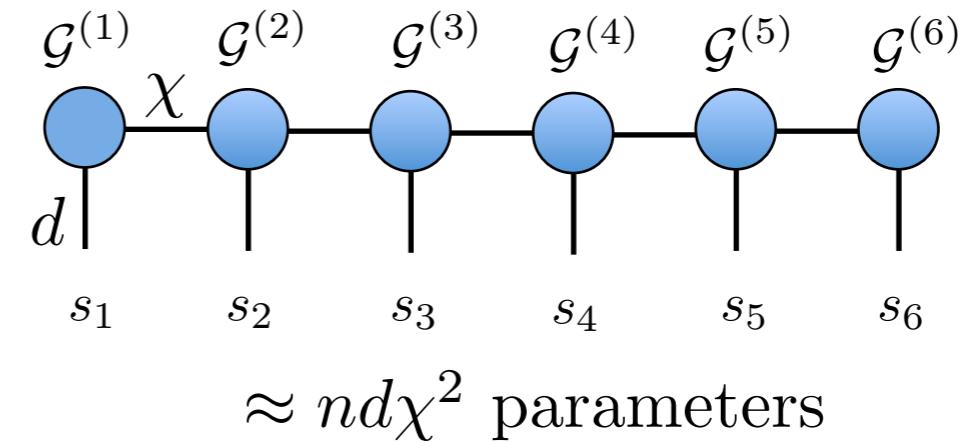
with  $\chi=1$ , MPS is a rank-one tensor

# Matrix Product States and Matrix Product Operator

$$\begin{aligned}\mathcal{A} &= \langle \mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(n)} \rangle \\ &= \mathcal{G}^{(1)} \times_1 \mathcal{G}^{(2)} \times_1 \cdots \times_1 \mathcal{G}^{(n)}\end{aligned}$$

$$a_{s_1, s_2, \dots, s_n} = \mathbf{G}_{s_1}^{(1)} \mathbf{G}_{s_2}^{(2)} \cdots \mathbf{G}_{s_n}^{(n)}$$

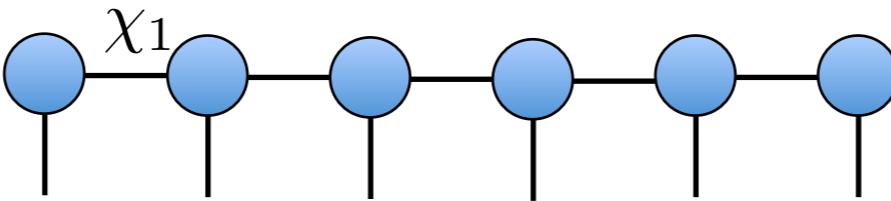
$$a_{s_1, s_2, \dots, s_n} = \mathbf{G}_{s_1, t_1}^{(1)} \mathbf{G}_{s_2, t_2}^{(2)} \cdots \mathbf{G}_{s_n, t_n}^{(n)}$$



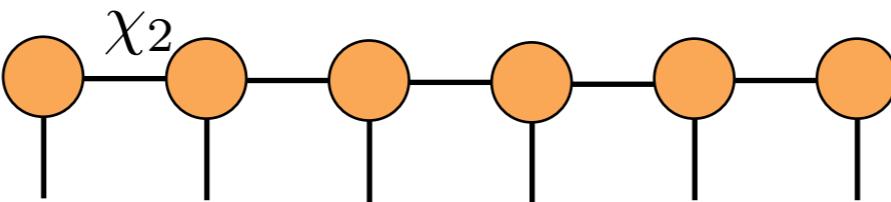
with  $\chi=1$ , MPS is a rank-one tensor

besides CP, MPS is another generalization of rank-one tensors

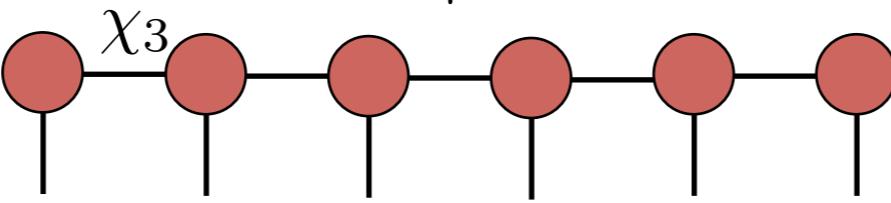
# Summing MPSes



+

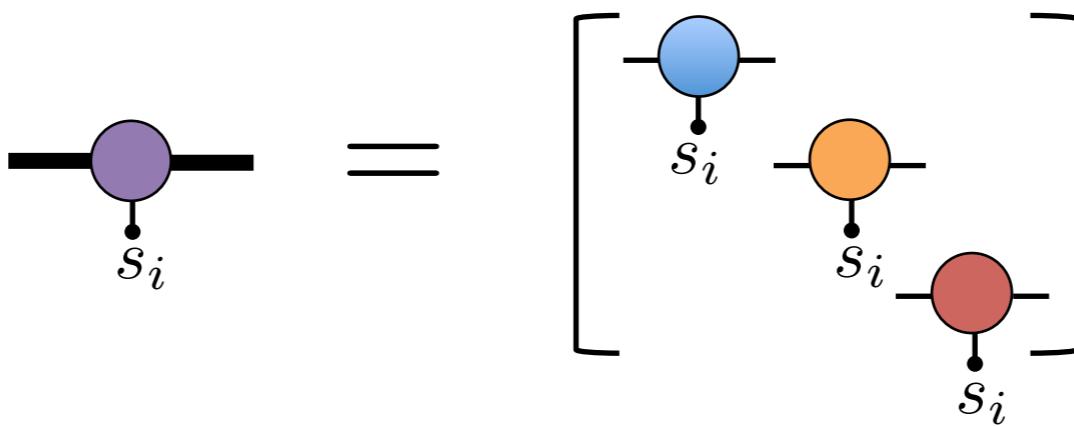
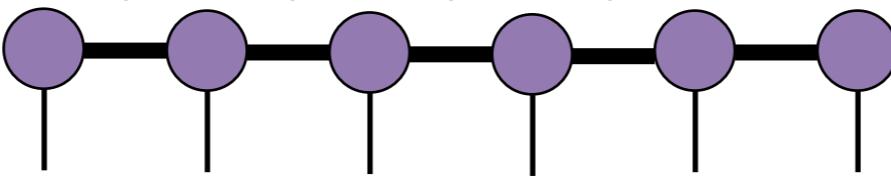


+



=

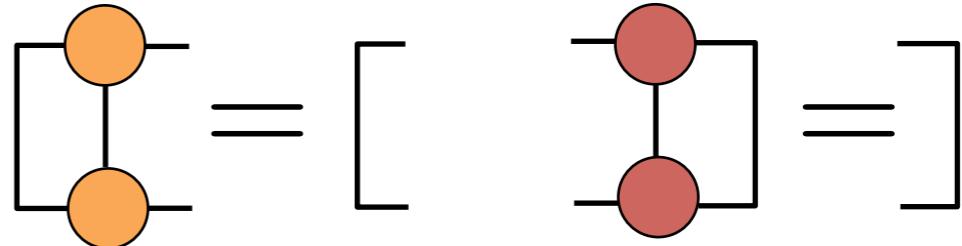
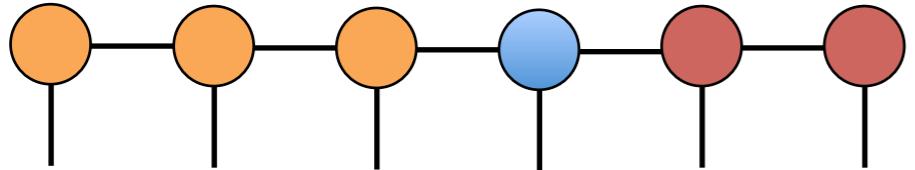
$$\chi_4 = \chi_1 + \chi_2 + \chi_3$$



# Canonical forms of MPS

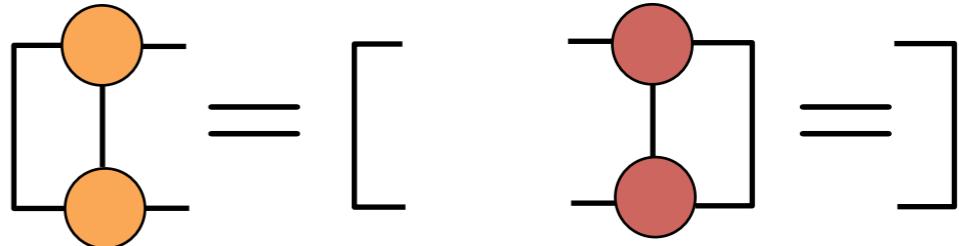
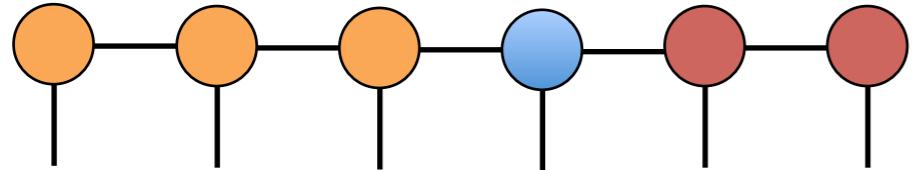
# Canonical forms of MPS

Analogous to the Tucker decomposition and HOSVDs, MPS has the benefits of orthogonality.



# Canonical forms of MPS

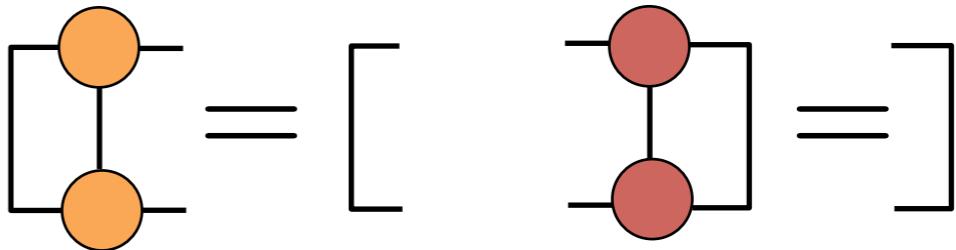
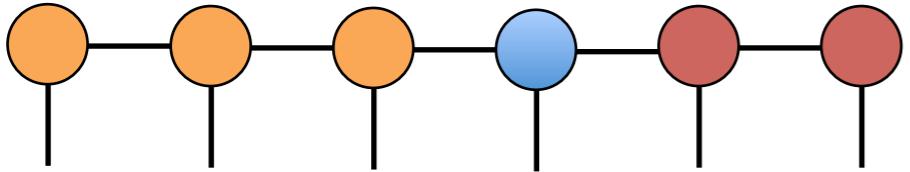
Analogous to the Tucker decomposition and HOSVDs, MPS has the benefits of orthogonality.



Benefits

# Canonical forms of MPS

Analogous to the Tucker decomposition and HOSVDs, MPS has the benefits of orthogonality.

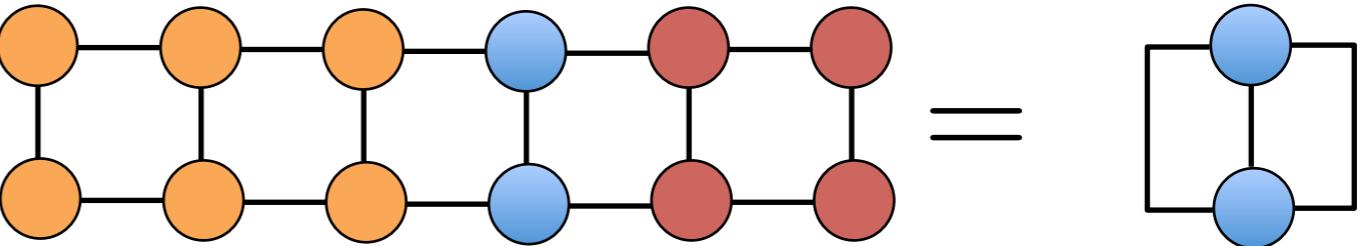
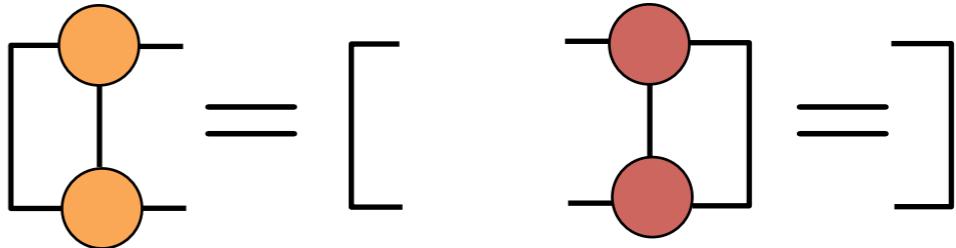
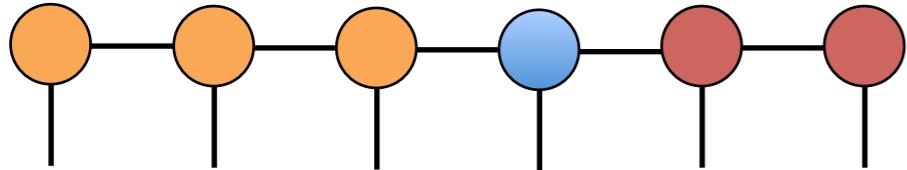


## Benefits

- Fixed gauge, no ambiguity

# Canonical forms of MPS

Analogous to the Tucker decomposition and HOSVDs, MPS has the benefits of orthogonality.

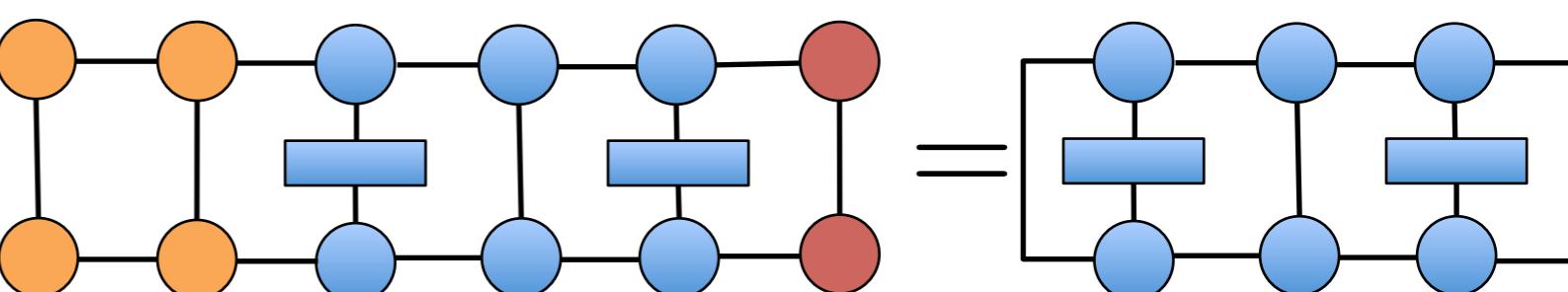
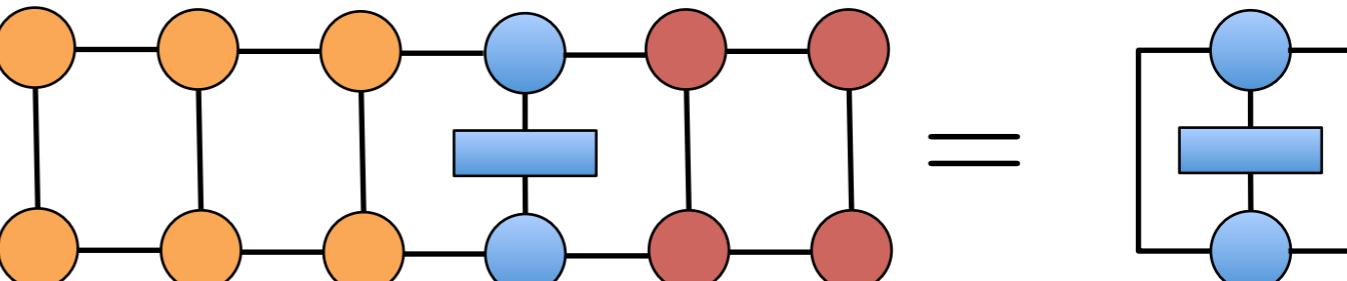
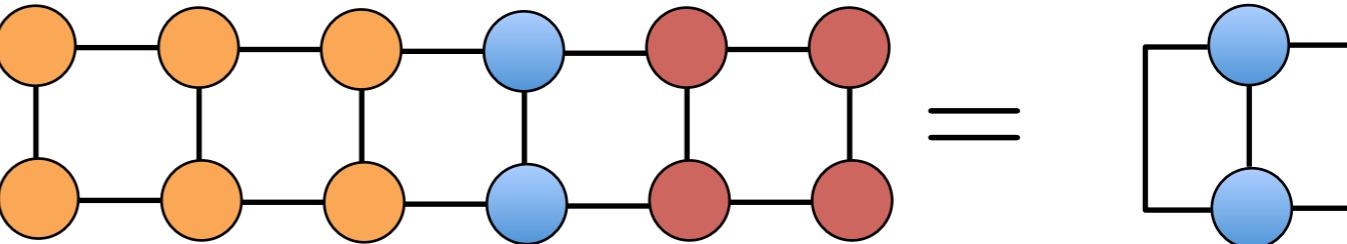
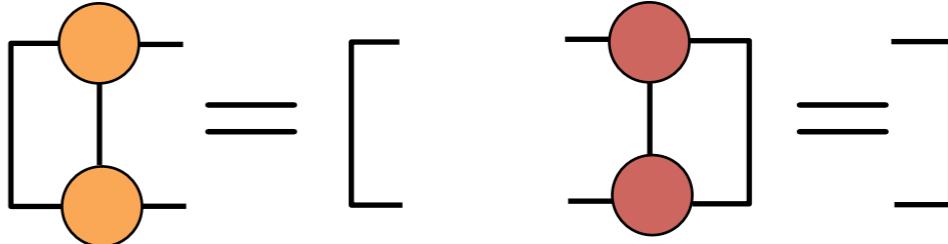
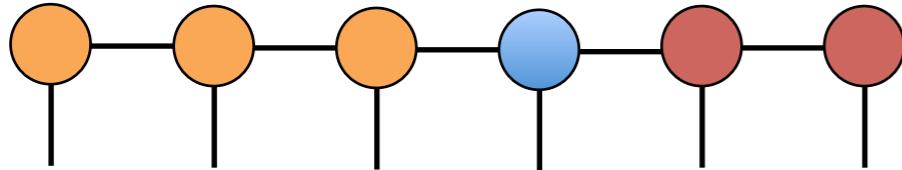


Benefits

- Fixed gauge, no ambiguity
- Easy norm computation

# Canonical forms of MPS

Analogous to the Tucker decomposition and HOSVDs, MPS has the benefits of orthogonality.

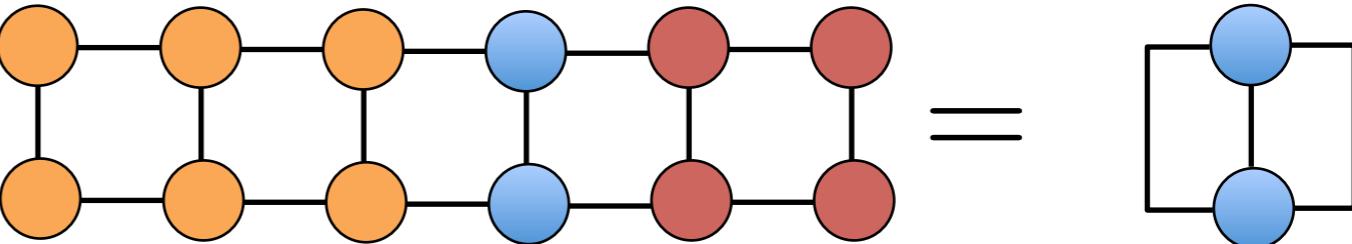
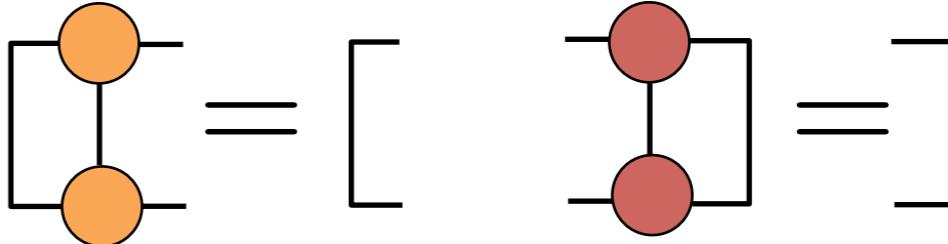
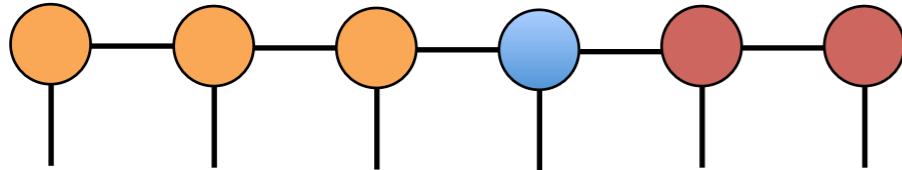


## Benefits

- Fixed gauge, no ambiguity
- Easy norm computation
- Easy expectation/  
correlation computation

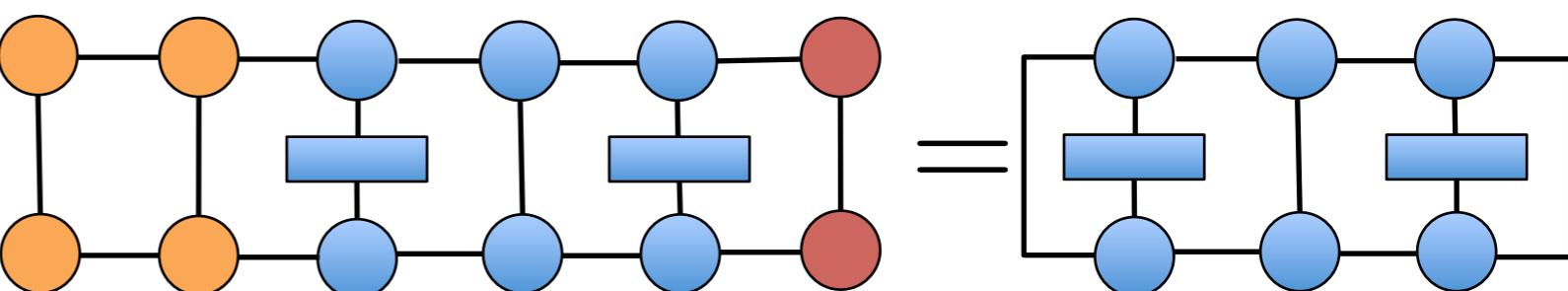
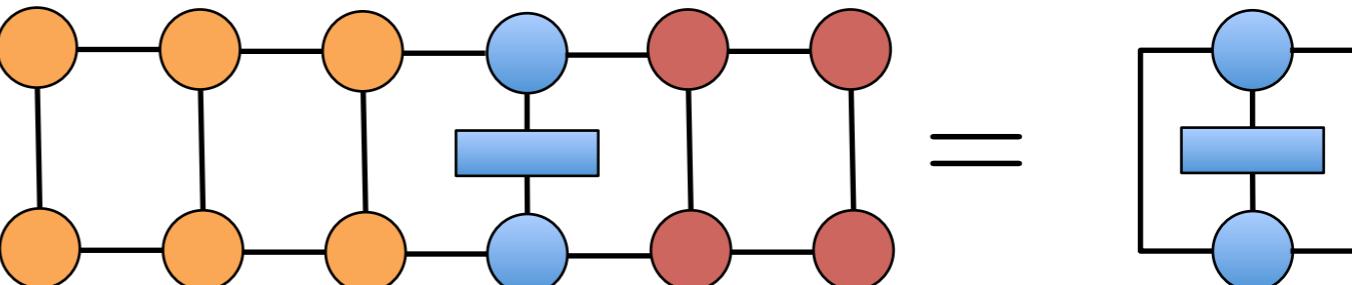
# Canonical forms of MPS

Analogous to the Tucker decomposition and HOSVDs, MPS has the benefits of orthogonality.



## Benefits

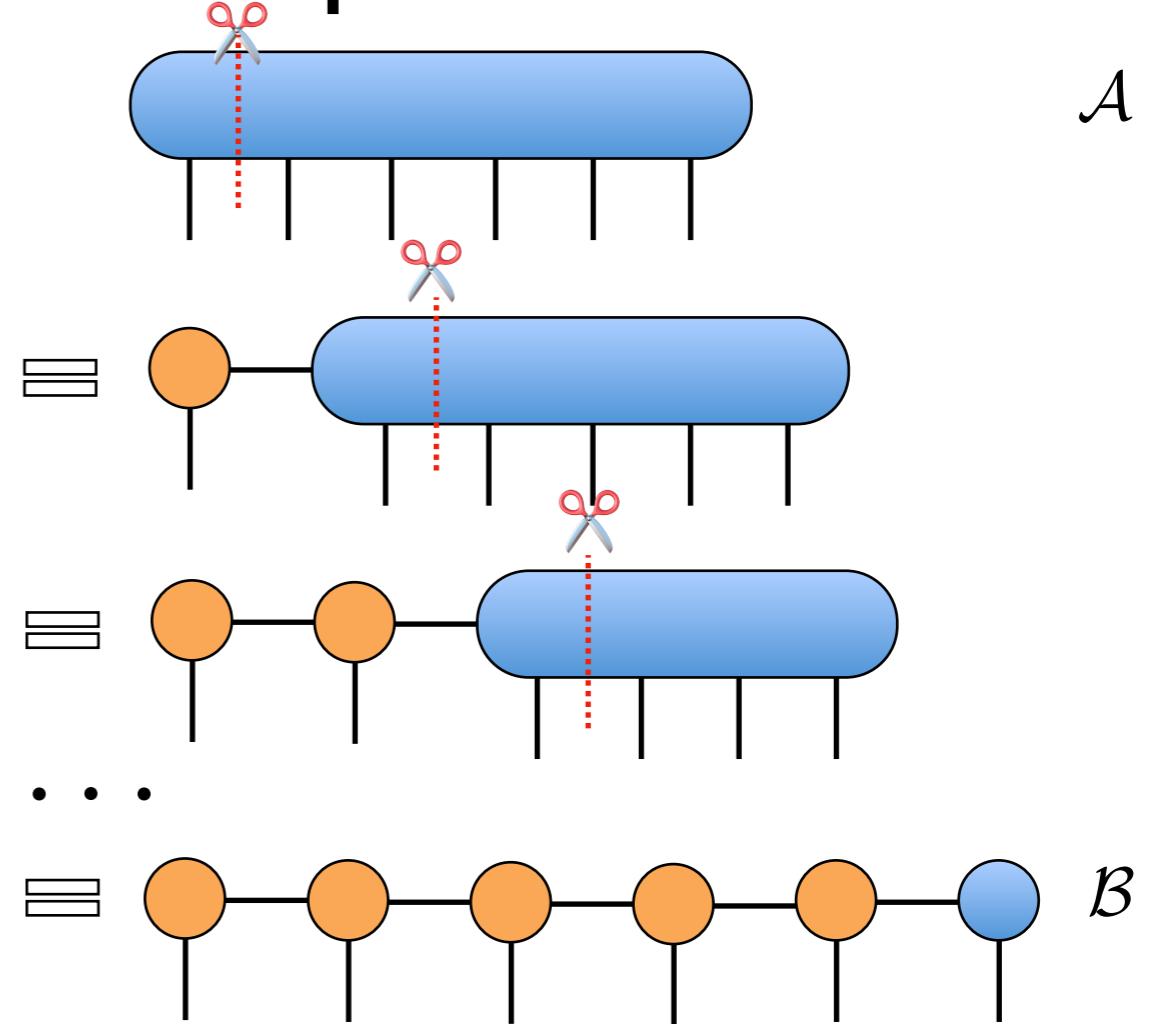
- Fixed gauge, no ambiguity
- Easy norm computation
- Easy expectation/  
correlation computation
- Always good conditioned



# Convert a raw tensor to a MPS: sequential SVDs

# Convert a raw tensor to a MPS: sequential SVDs

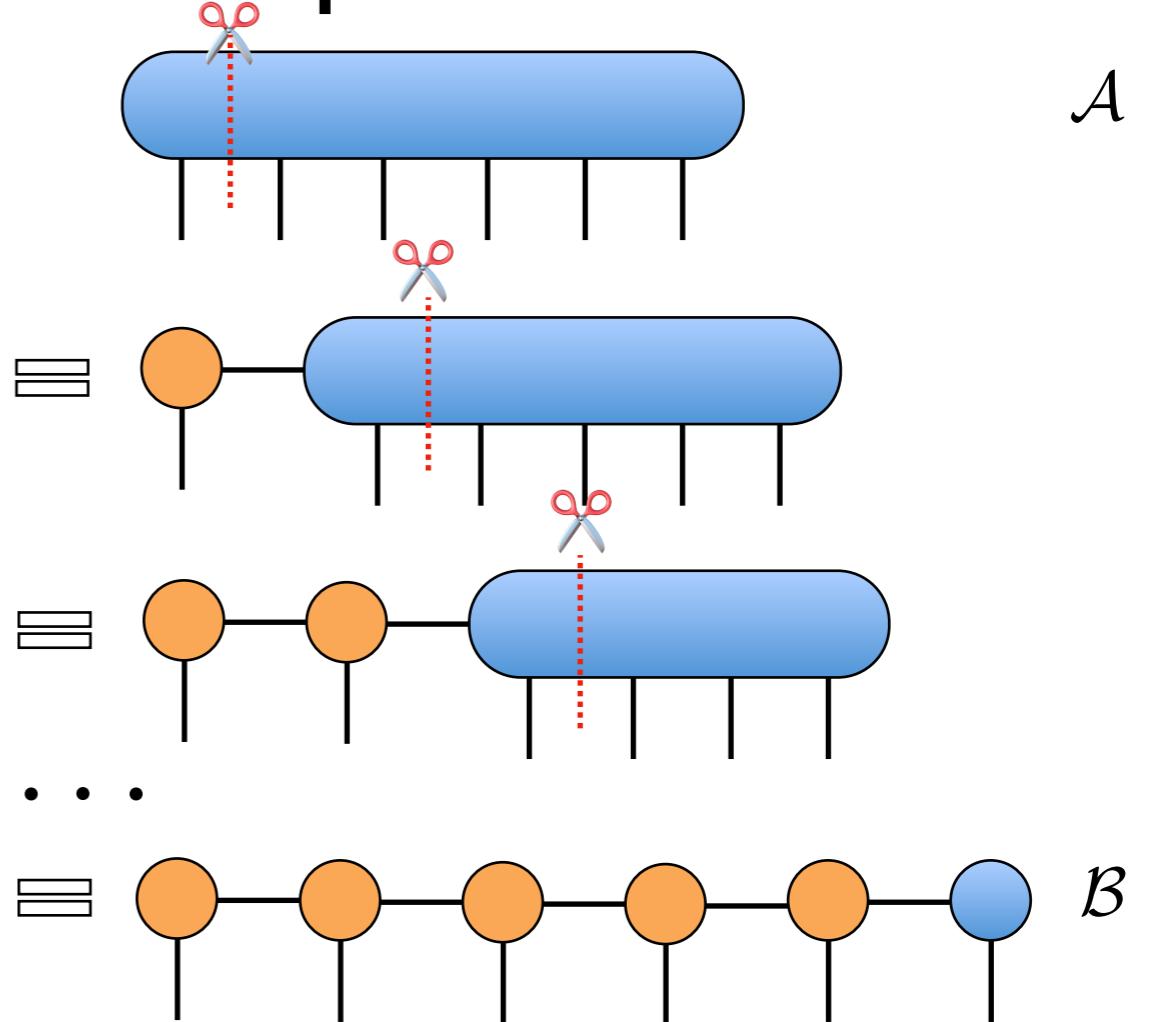
analogous to HOSVD, but processes every mode *one by one*



# Convert a raw tensor to a MPS: sequential SVDs

analogous to HOSVD, but processes every mode *one by one*

matrices encountered during SVDs are larger than HOSVD



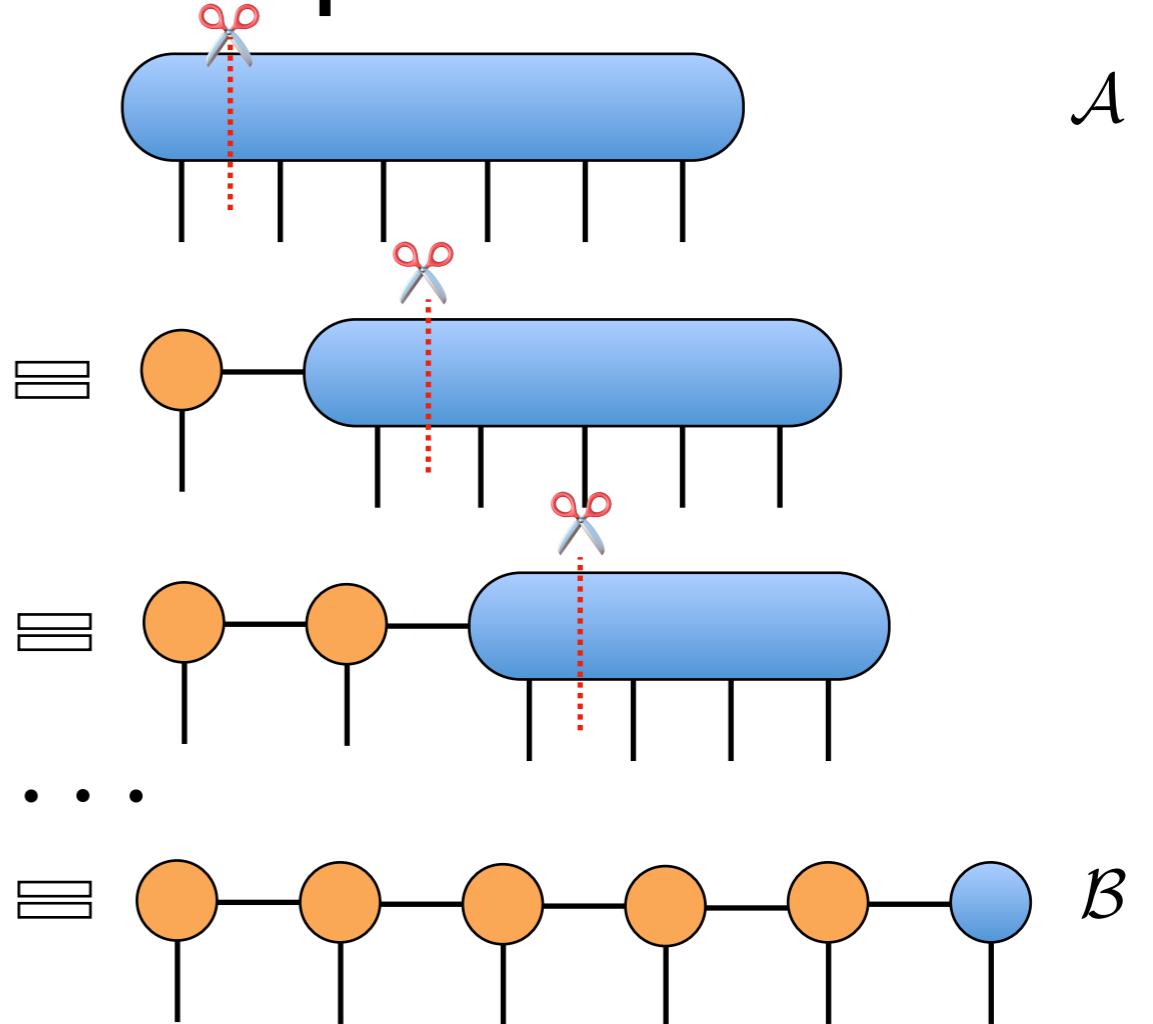
# Convert a raw tensor to a MPS: sequential SVDs

analogous to HOSVD, but processes every mode *one by one*

matrices encountered during SVDs are larger than HOSVD

error made in compression

$$\|\mathcal{A} - \mathcal{B}\|_F \leq \sqrt{\sum_{i=1}^n \epsilon_i^2}$$



# Convert a raw tensor to a MPS: sequential SVDs

analogous to HOSVD, but processes every mode *one by one*

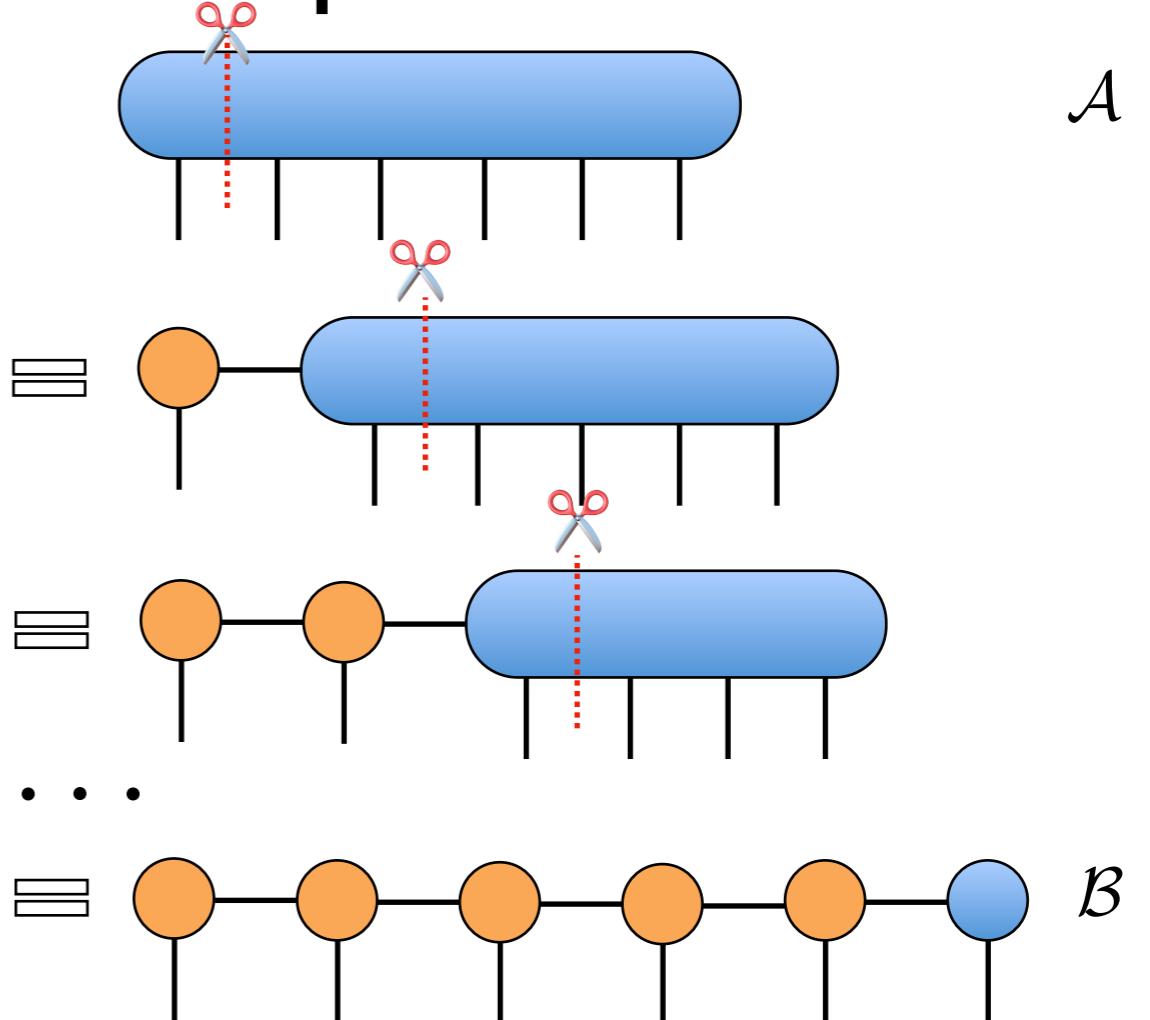
matrices encountered during SVDs are larger than HOSVD

error made in compression

$$\|\mathcal{A} - \mathcal{B}\|_F \leq \sqrt{\sum_{i=1}^n \epsilon_i^2}$$

sequential SVD is quasi-optimal

$$\|\mathcal{A} - \mathcal{B}\|_F \leq \sqrt{n} \|\mathcal{A} - \mathcal{B}_{\text{best}}\|_F$$



# Convert a raw tensor to a MPS: sequential SVDs

analogous to HOSVD, but processes every mode *one by one*

matrices encountered during SVDs are larger than HOSVD

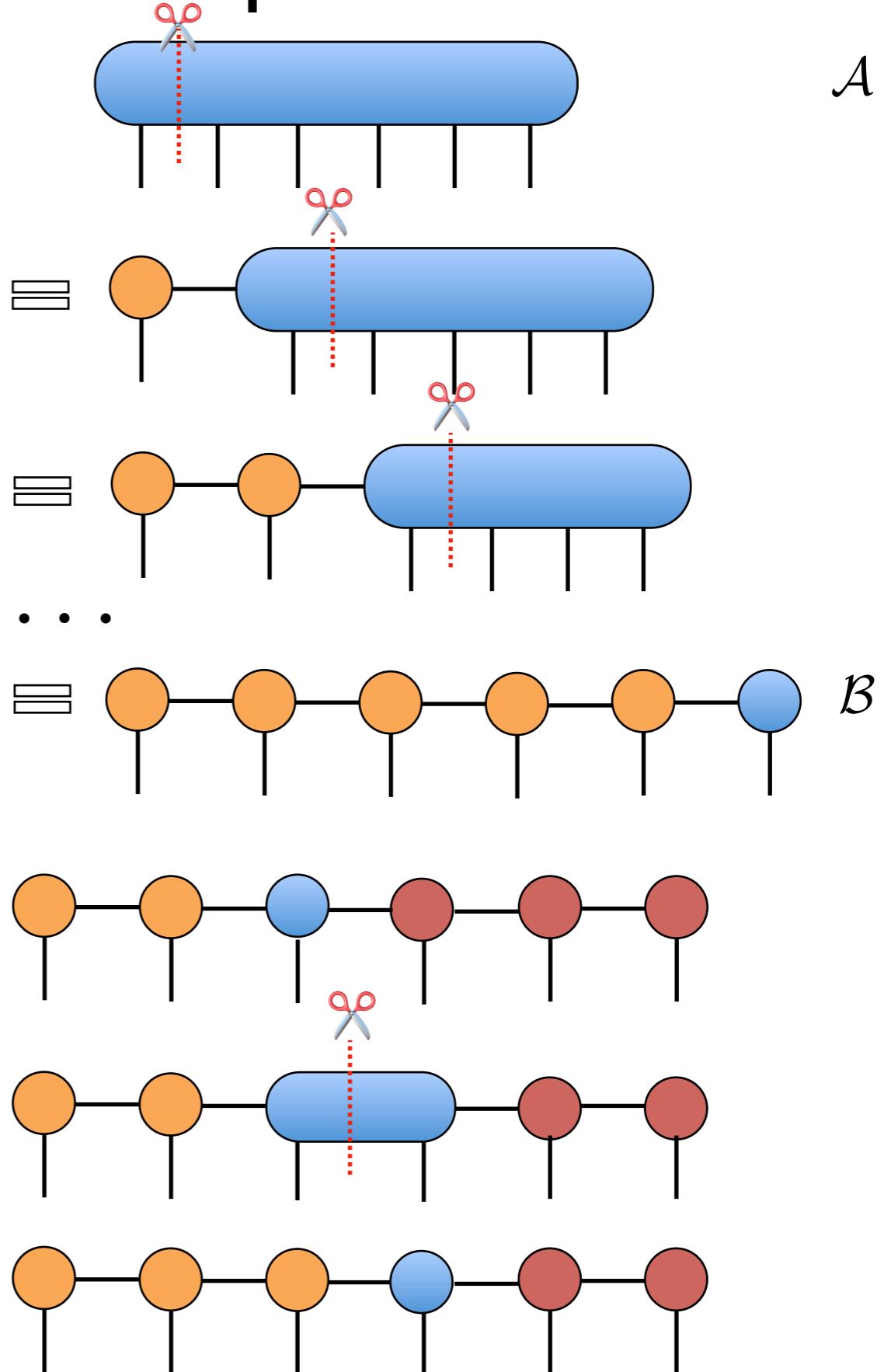
error made in compression

$$\|\mathcal{A} - \mathcal{B}\|_F \leq \sqrt{\sum_{i=1}^n \epsilon_i^2}$$

sequential SVD is quasi-optimal

$$\|\mathcal{A} - \mathcal{B}\|_F \leq \sqrt{n} \|\mathcal{A} - \mathcal{B}_{\text{best}}\|_F$$

can be refined iteratively (known as recompression, rounding)



# Application: compressing neural networks

# Application: compressing neural networks

Deep neural networks are most popular machine learning models.

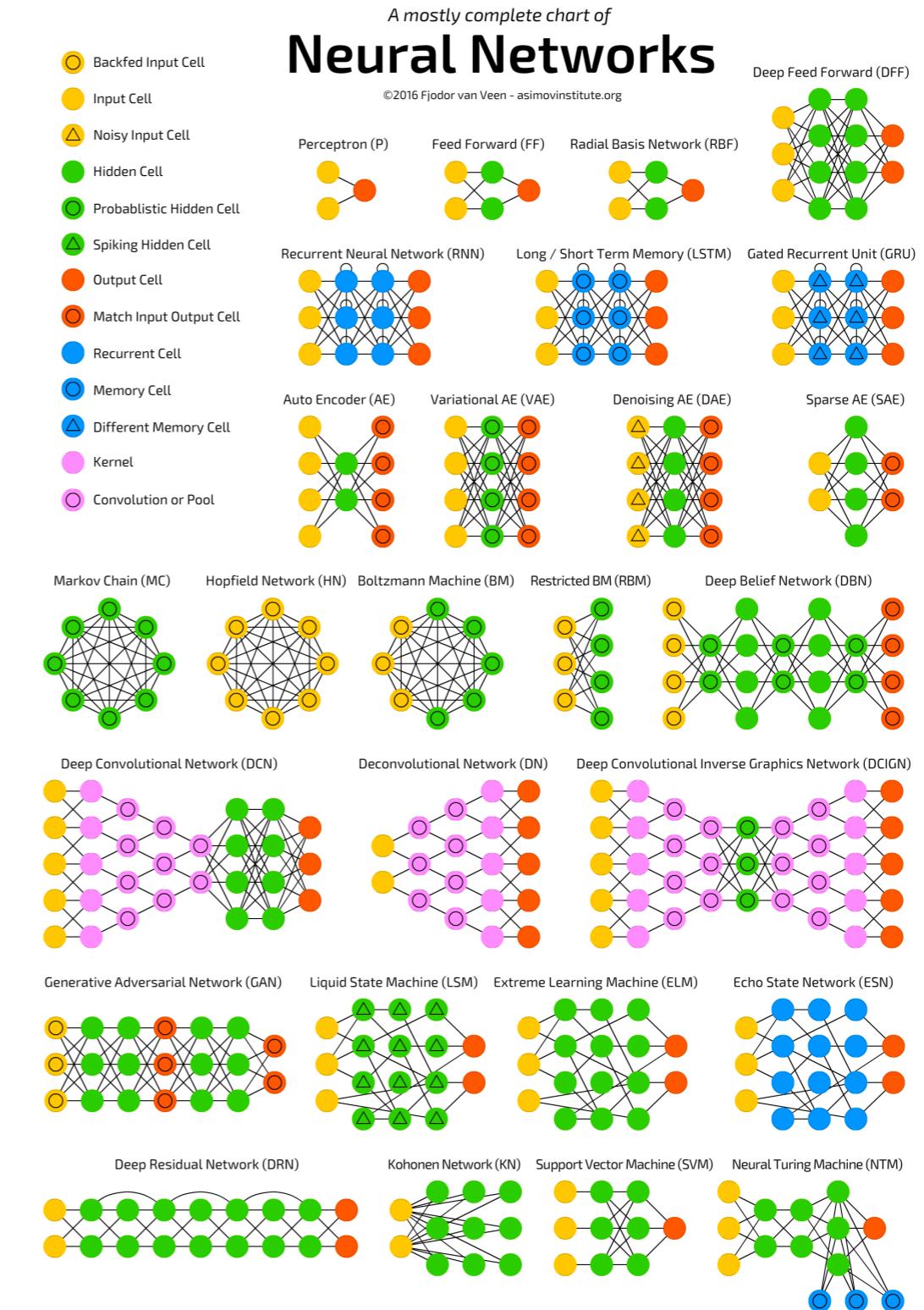


Image courtesy: Fjodor van Veen

# Application: compressing neural networks

Deep neural networks are most popular machine learning models.

Most of the parameters are contained in fully-connected layers, which are *dense matrices*.

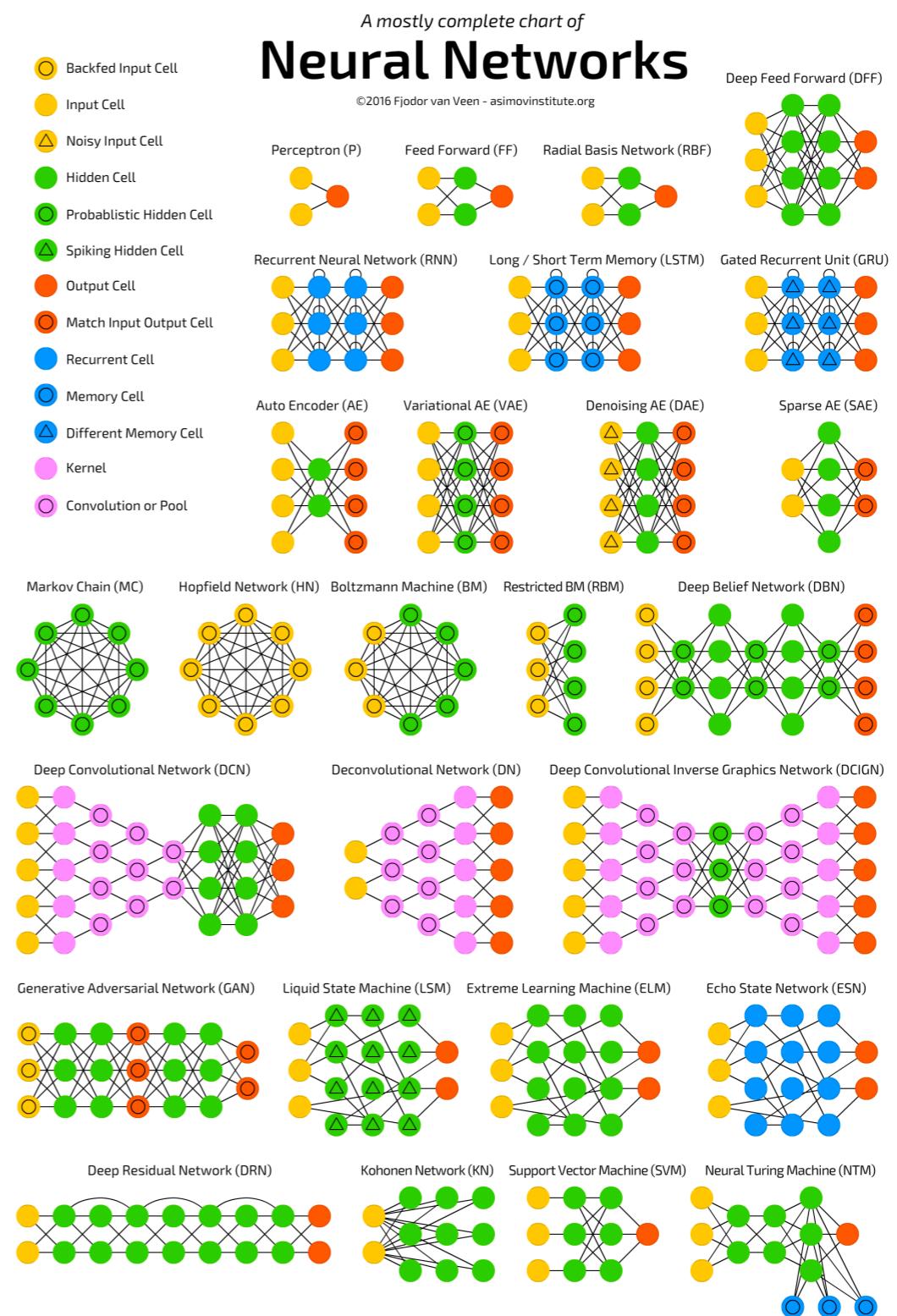
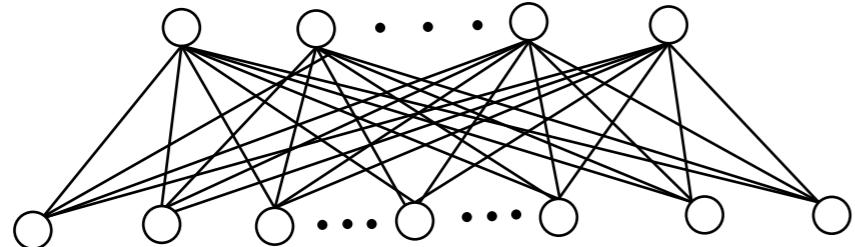


Image courtesy: Fjodor van Veen

# Application: compressing neural networks

Deep neural networks are most popular machine learning models.

Most of the parameters are contained in fully-connected layers, which are *dense matrices*.

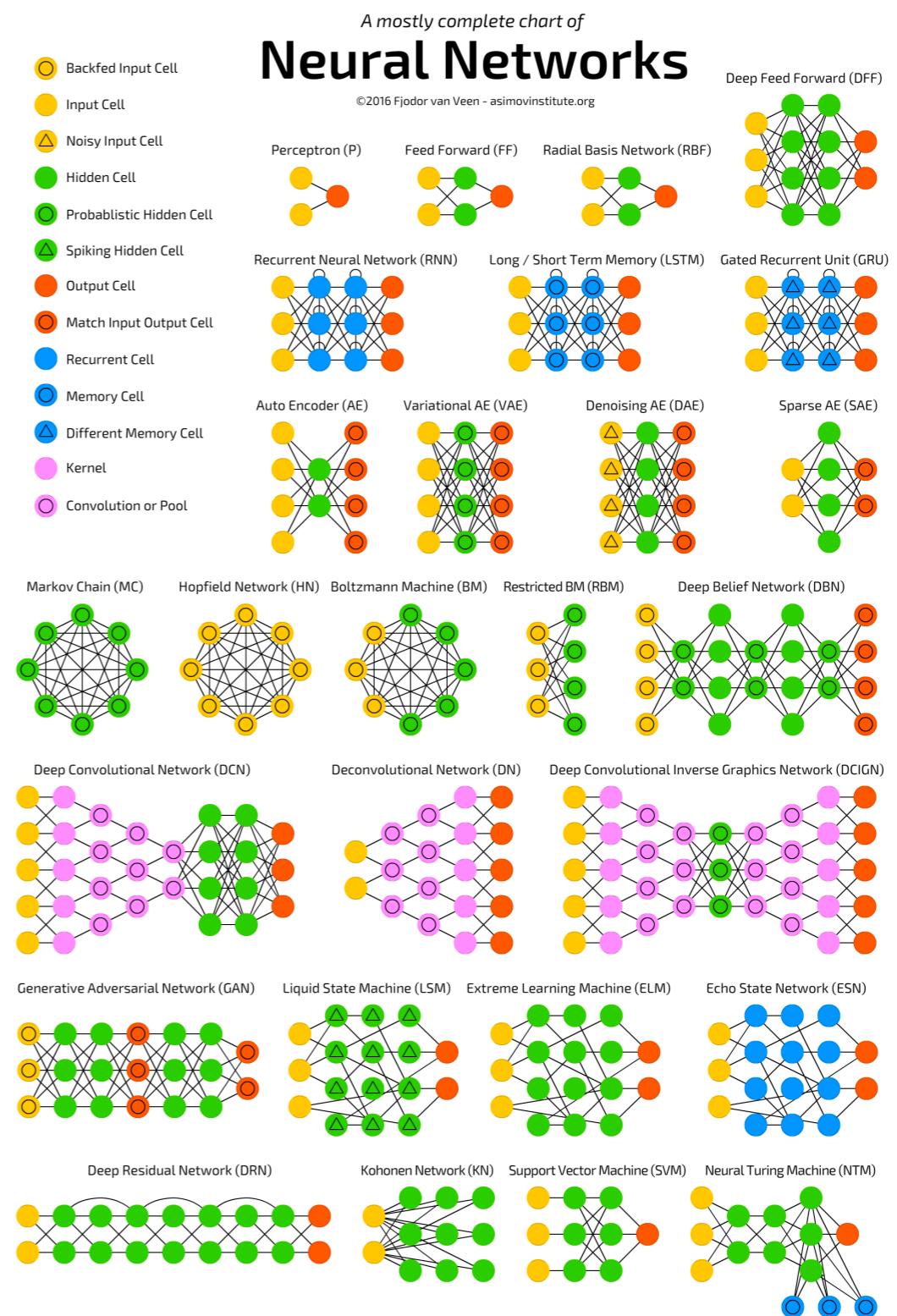
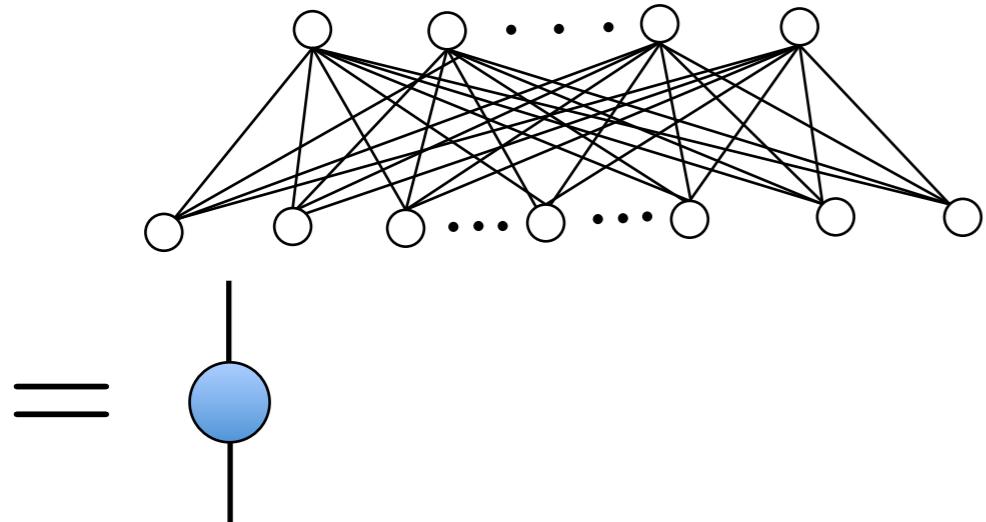


Image courtesy: Fjodor van Veen

# Application: compressing neural networks

Deep neural networks are most popular machine learning models.

Most of the parameters are contained in fully-connected layers, which are *dense matrices*.

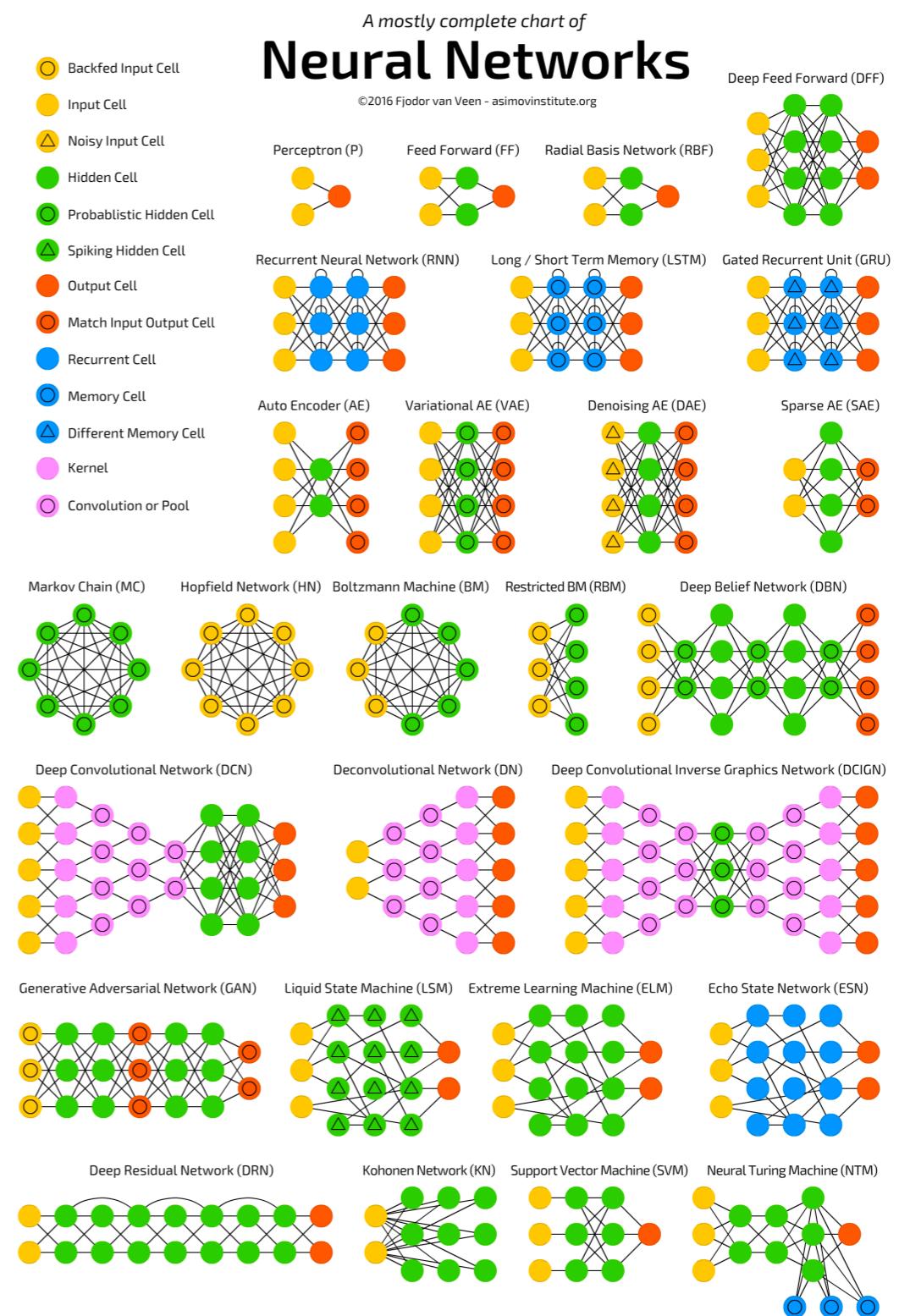
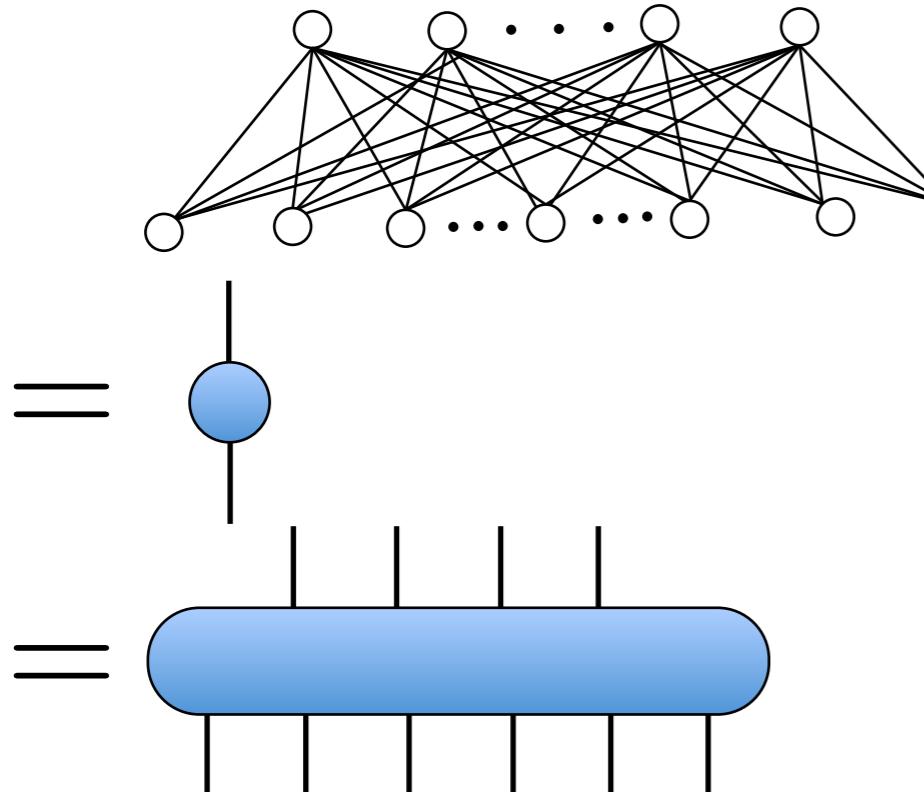


Image courtesy: Fjodor van Veen

# Application: compressing neural networks

Deep neural networks are most popular machine learning models.

Most of the parameters are contained in fully-connected layers, which are *dense matrices*.

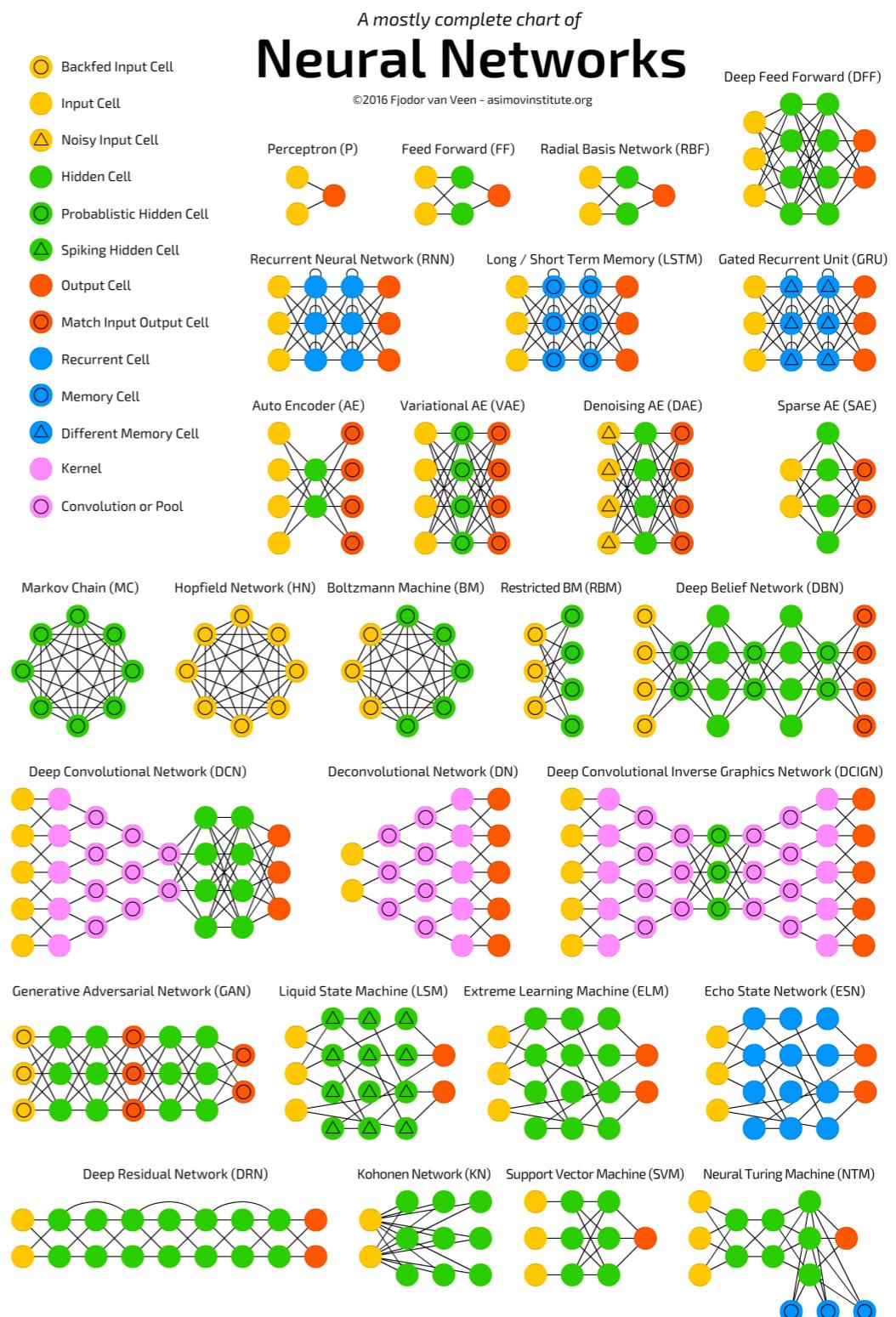
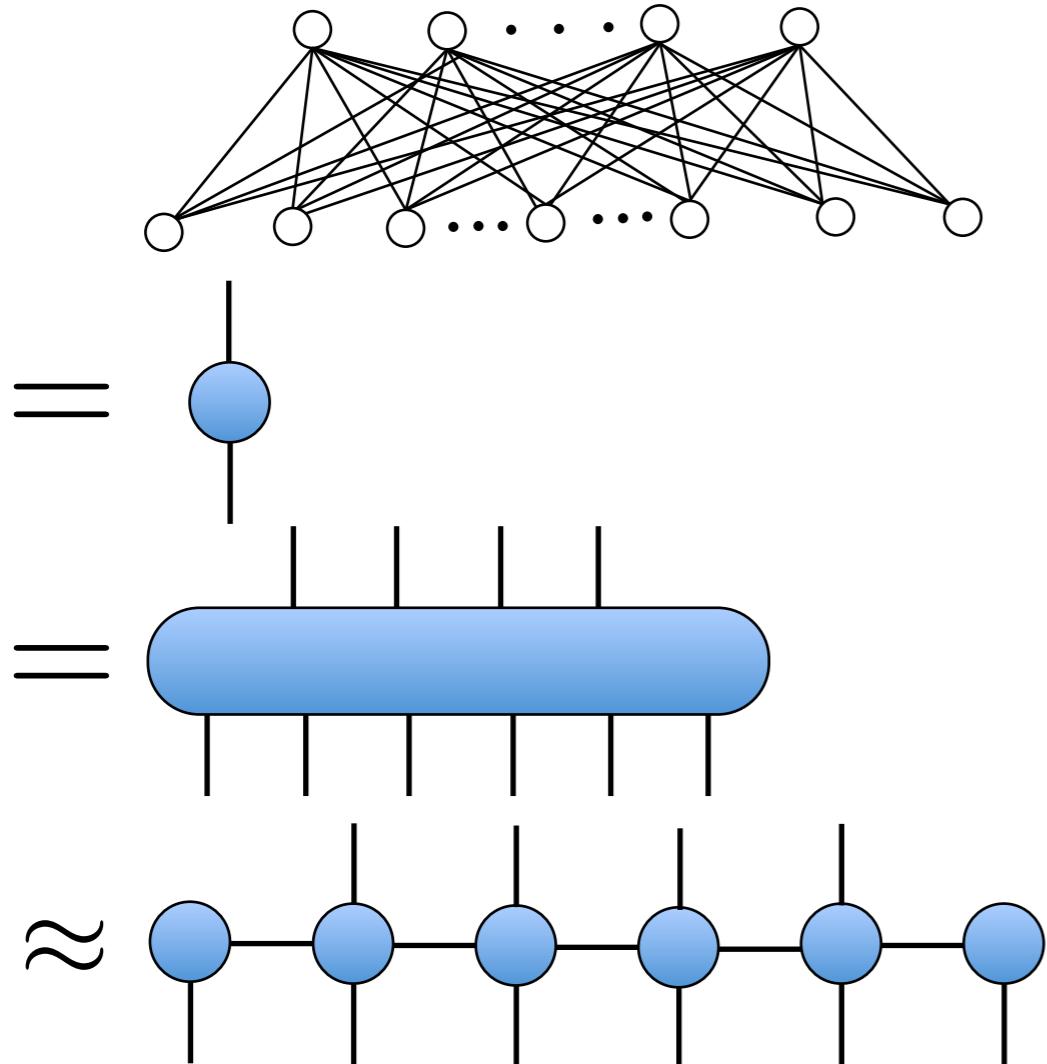
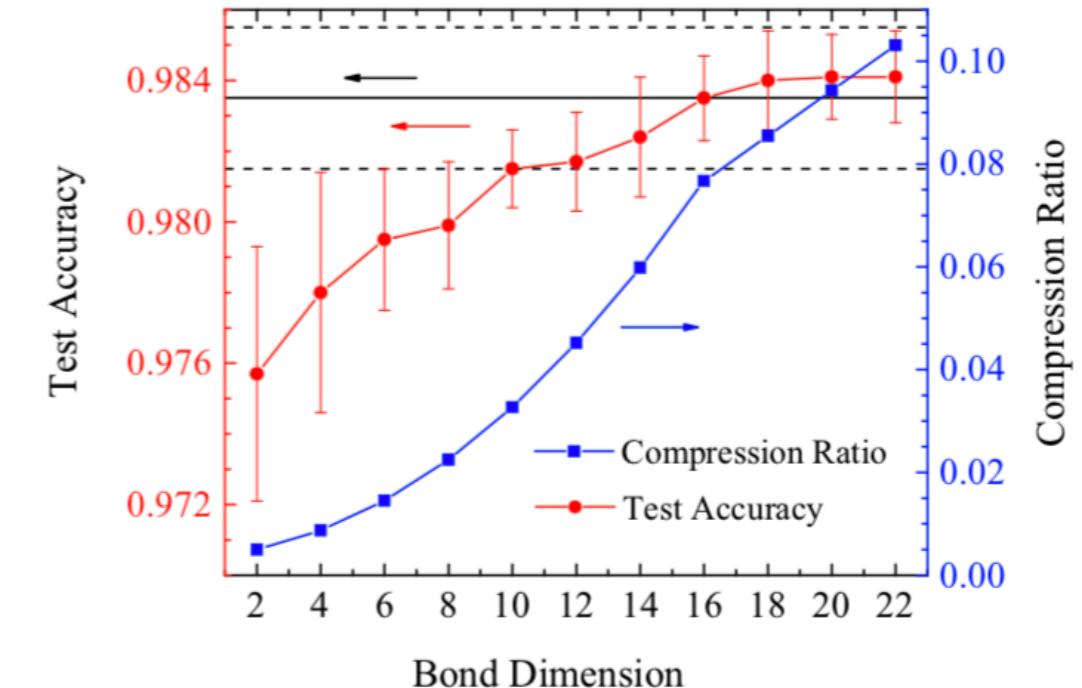
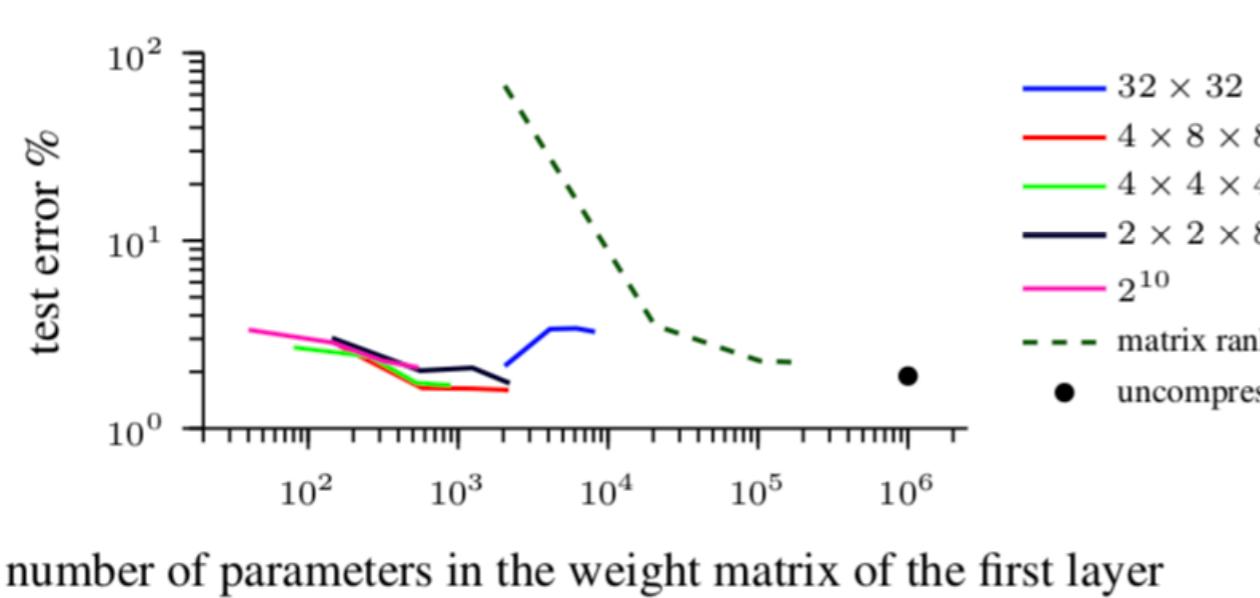


Image courtesy: Fjodor van Veen

# Application: compressing neural networks

On 2-layer perceptron, MNIST

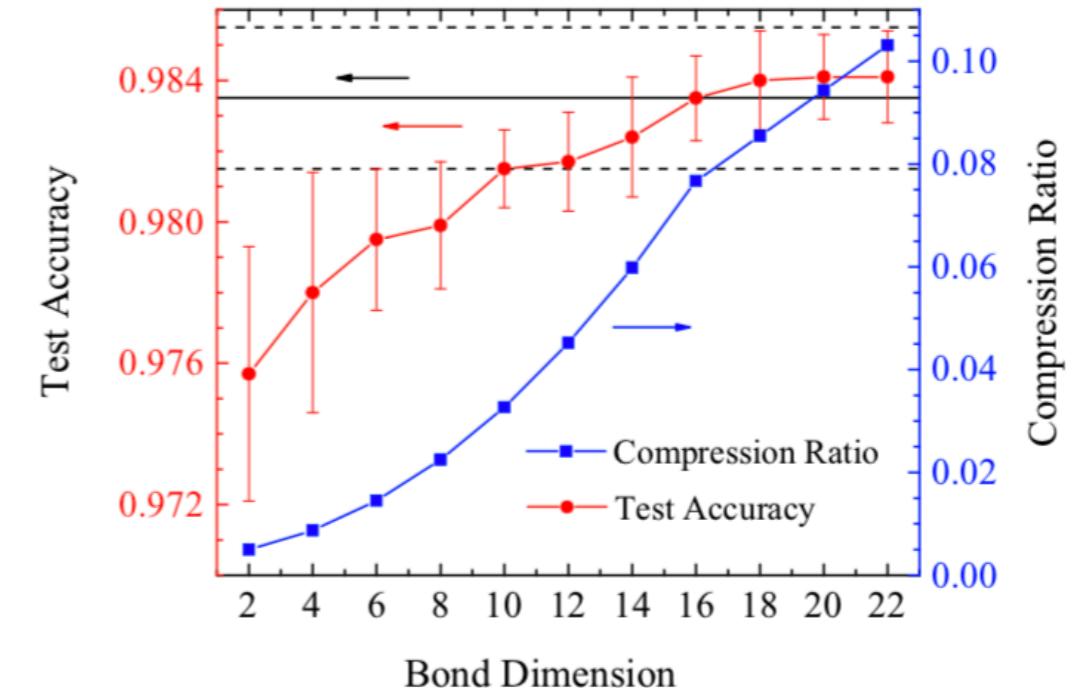
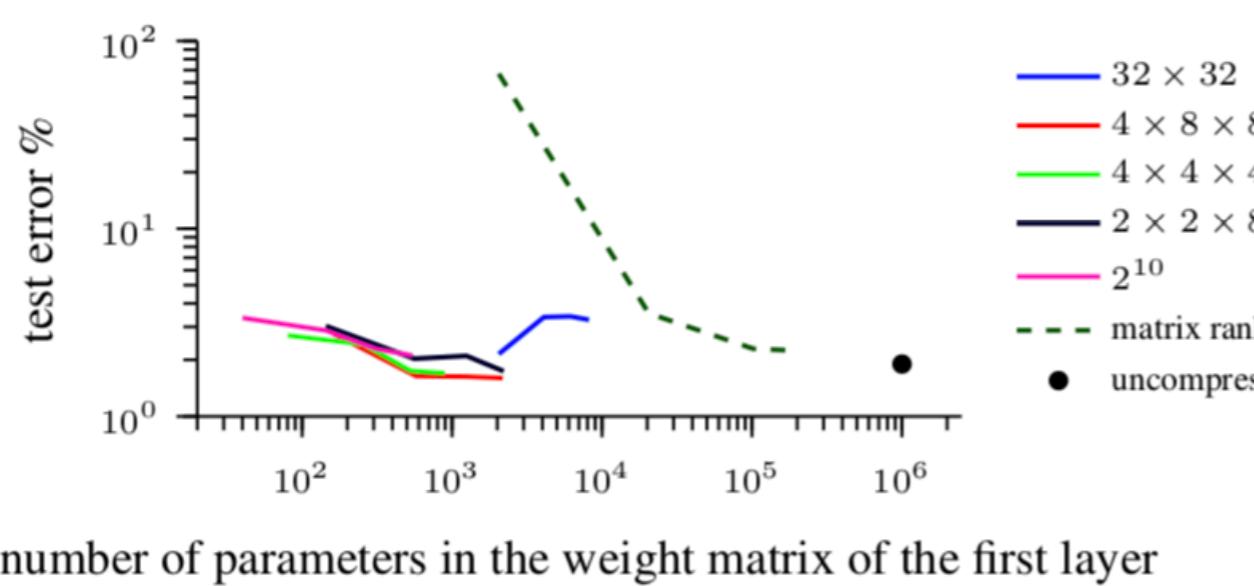


Novikov, Podoprikhin, Osokin, Vetrov, NIPS 2015

Gao et al, arXiv:1904.06194

# Application: compressing neural networks

On 2-layer perceptron, MNIST



Novikov, Podoprikhin, Osokin, Vetrov, NIPS 2015

Gao et al, arXiv:1904.06194

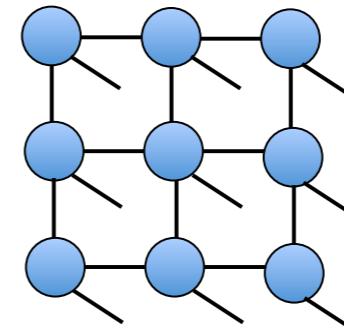
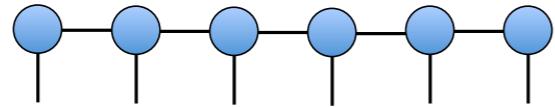
Similar ideas have been applied to RBM as  
“Matrix product operator restricted Boltzmann Machine” in  
Chen, Batselier, Ko, Wong arXiv:1811.04608

# Outline

- Tensor networks for representations  
*(in a small space)*
  - CP, Tucker, MPS, Tree Tensor networks
  - Example: Compressing neural networks
- Contraction of tensor networks
- Tensor networks for learning  
*(in a large space)*
  - Hilbert space as feature space: classification and PCA using tensor networks
  - Tensor networks for generative modeling

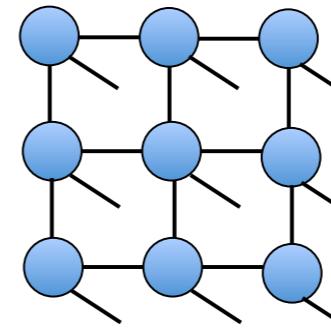
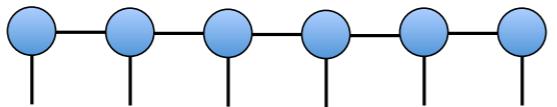
# Contracting tensor networks corresponds to

- Evaluating a wave function

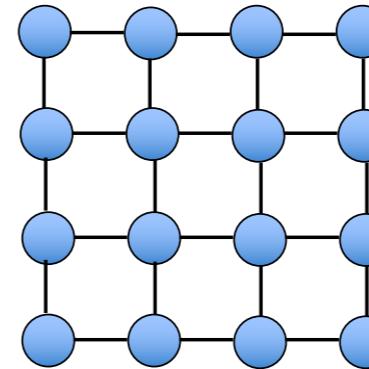
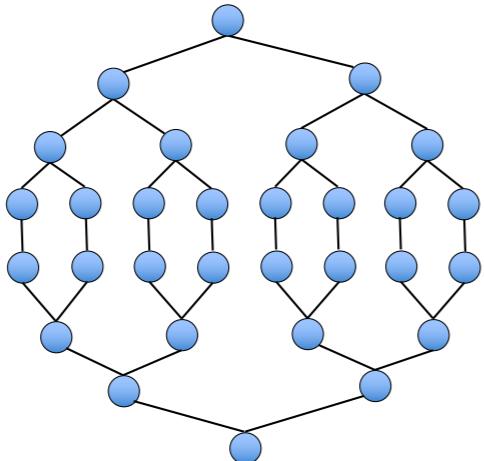


# Contracting tensor networks corresponds to

- Evaluating a wave function

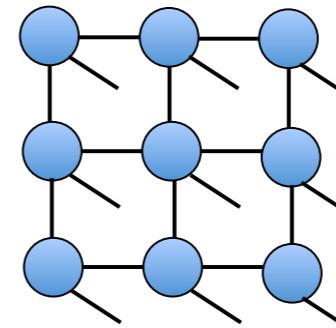
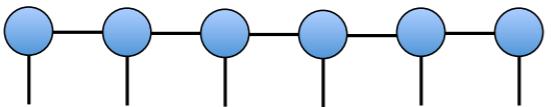


- Marginalizing a graphical model

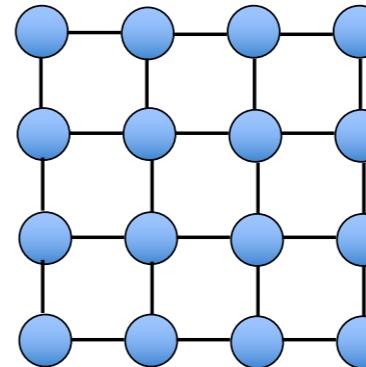
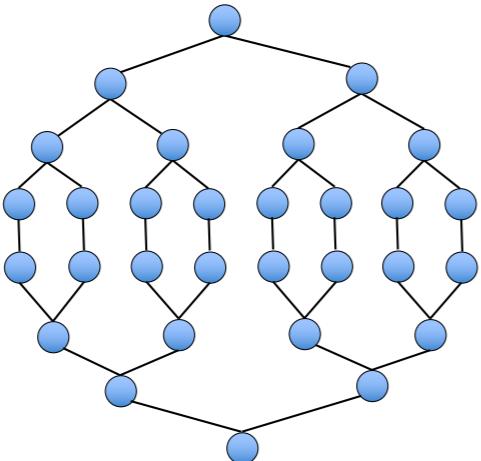


# Contracting tensor networks corresponds to

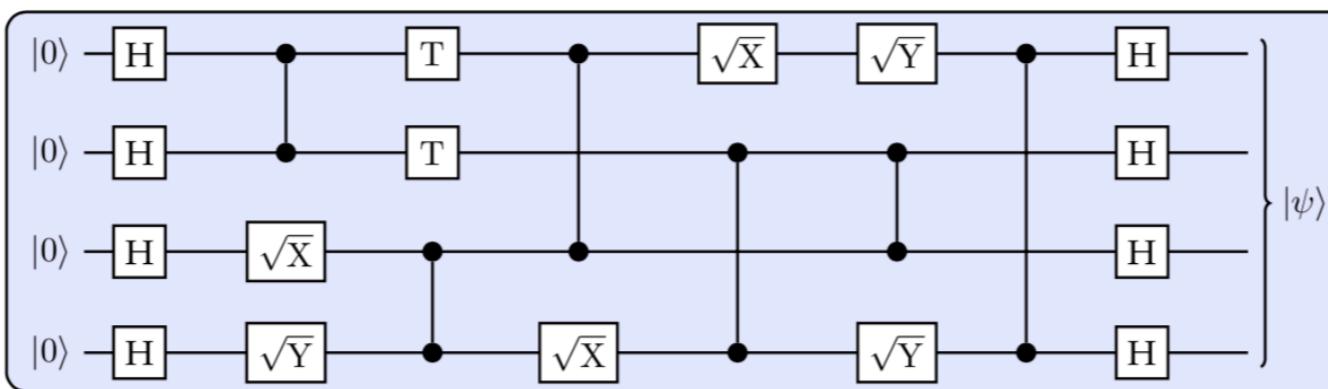
- Evaluating a wave function



- Marginalizing a graphical model



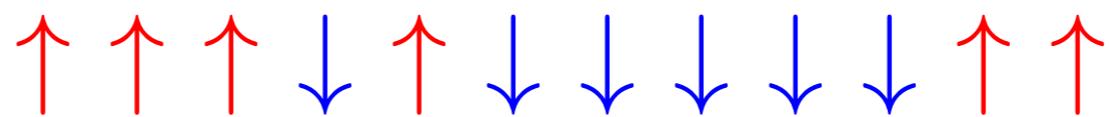
- Simulating a quantum circuit



Robeva et al, arXiv:1710.01437  
Boixo et al, arXiv:1712.05384  
Chen et al, arXiv:1805.01450

## Example: Ising model

$$\mathbf{S} = \{+1, -1\}^n$$



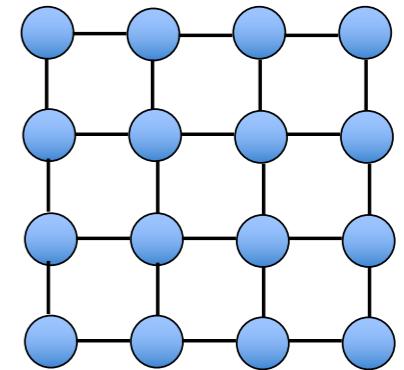
$$P(\mathbf{S}) = \frac{1}{Z} e^{-\beta E(\mathbf{S})}$$

$$Z = \sum_{\mathbf{S}} e^{-\beta E(\mathbf{S})}$$

## Example: Ising model

$$\mathbf{S} = \{+1, -1\}^n$$

↑ ↑ ↑ ↓ ↑ ↓ ↓ ↓ ↑ ↑



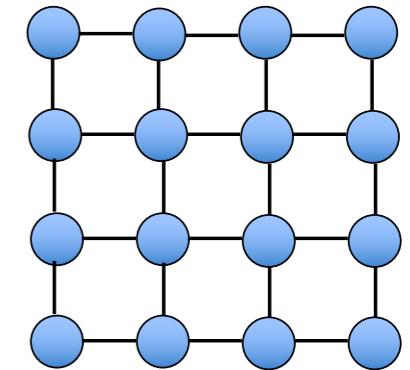
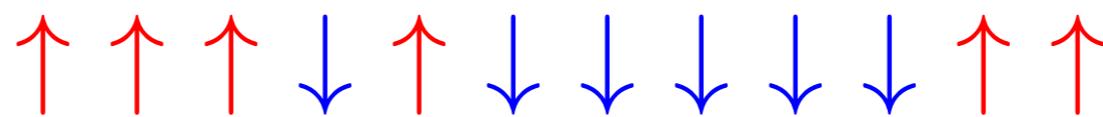
$$P(\mathbf{S}) = \frac{1}{Z} e^{-\beta E(\mathbf{S})}$$

$$Z = \sum_{\mathbf{S}} e^{-\beta E(\mathbf{S})}$$

Probability distribution is a tensor, hence can be represented by a tensor network in general.

### Example: Ising model

$$\mathbf{S} = \{+1, -1\}^n$$



$$P(\mathbf{S}) = \frac{1}{Z} e^{-\beta E(\mathbf{S})}$$

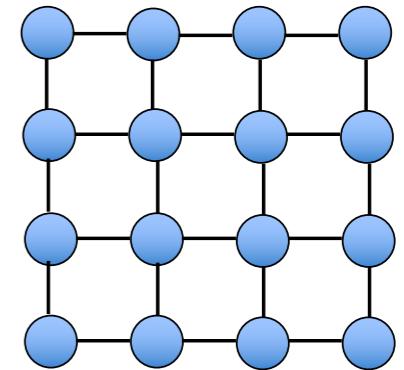
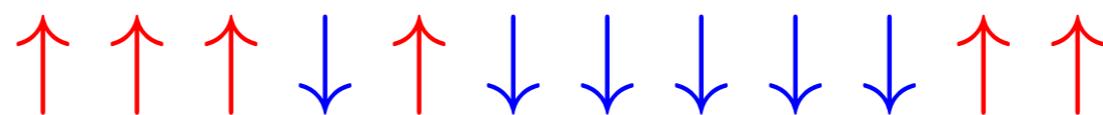
$$Z = \sum_{\mathbf{S}} e^{-\beta E(\mathbf{S})}$$

Probability distribution is a tensor, hence can be represented by a tensor network in general.

$$\begin{array}{c} \text{A} \\ \text{i} \quad \text{j} \end{array} \quad i - \text{A} - j \quad Z = \sum_{i,j} A_{ij} = \mathbf{1}^T A \mathbf{1} = [1, 1] - \text{A} - \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

### Example: Ising model

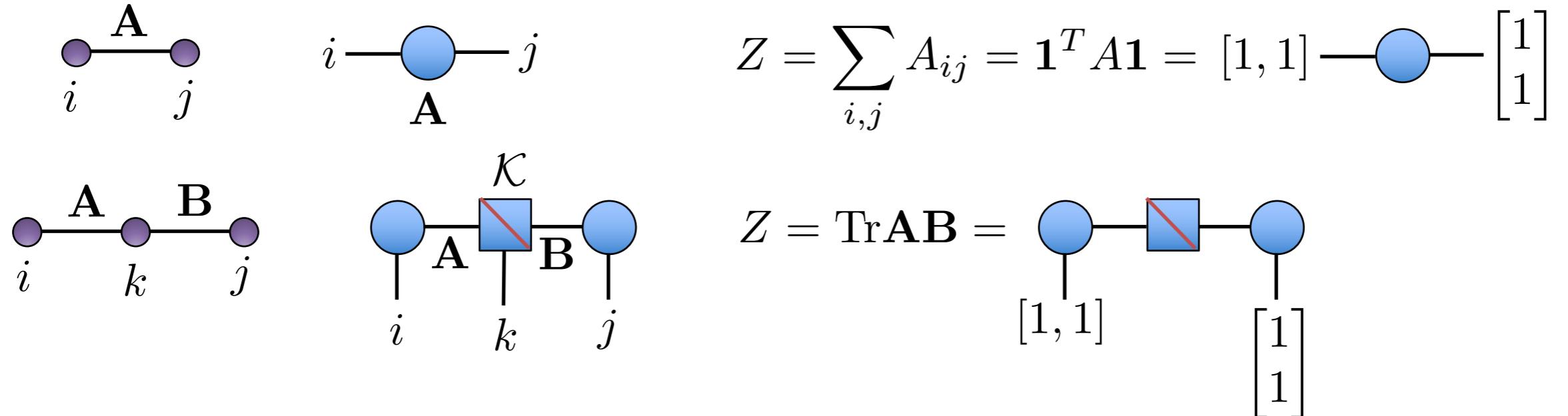
$$\mathbf{S} = \{+1, -1\}^n$$



$$P(\mathbf{S}) = \frac{1}{Z} e^{-\beta E(\mathbf{S})}$$

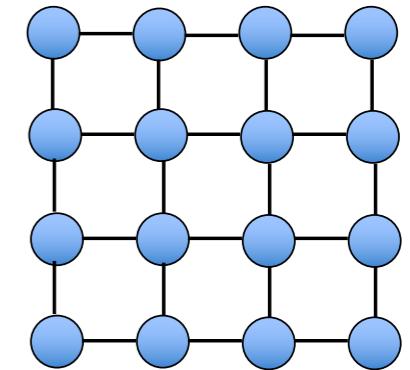
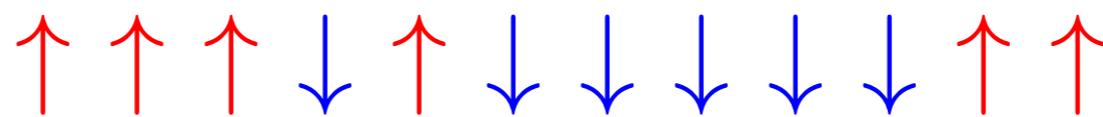
$$Z = \sum_{\mathbf{S}} e^{-\beta E(\mathbf{S})}$$

Probability distribution is a tensor, hence can be represented by a tensor network in general.



## Example: Ising model

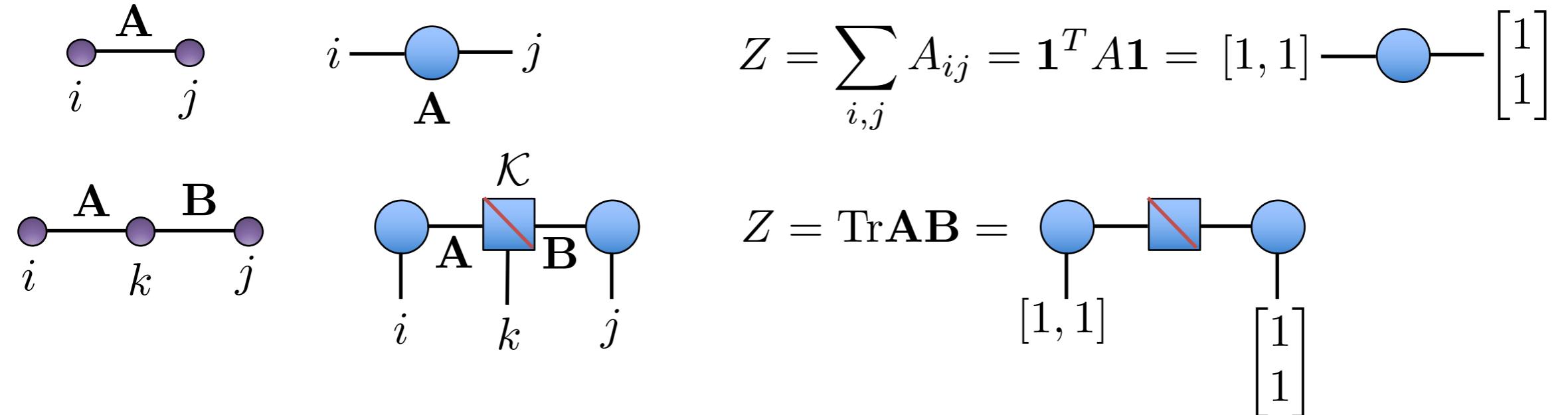
$$\mathbf{S} = \{+1, -1\}^n$$



$$P(\mathbf{S}) = \frac{1}{Z} e^{-\beta E(\mathbf{S})}$$

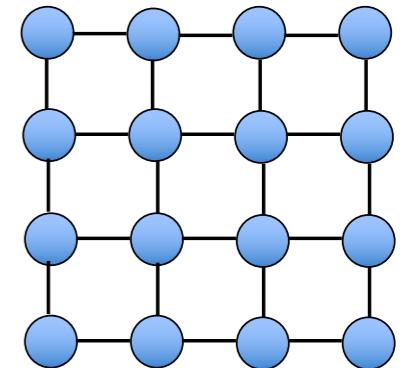
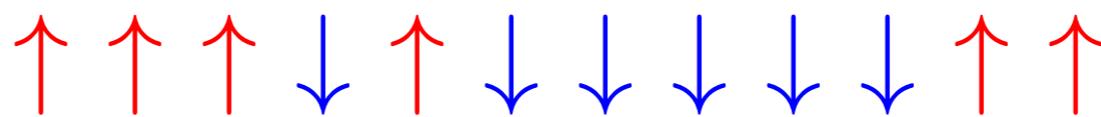
$$Z = \sum_{\mathbf{S}} e^{-\beta E(\mathbf{S})}$$

Probability distribution is a tensor, hence can be represented by a tensor network in general.



### Example: Ising model

$$\mathbf{S} = \{+1, -1\}^n$$



→ Tutorial for computing partition function of the 2-D Ising model

$$P(\mathbf{S}) = \frac{1}{Z} e^{-\beta E(\mathbf{S})}$$

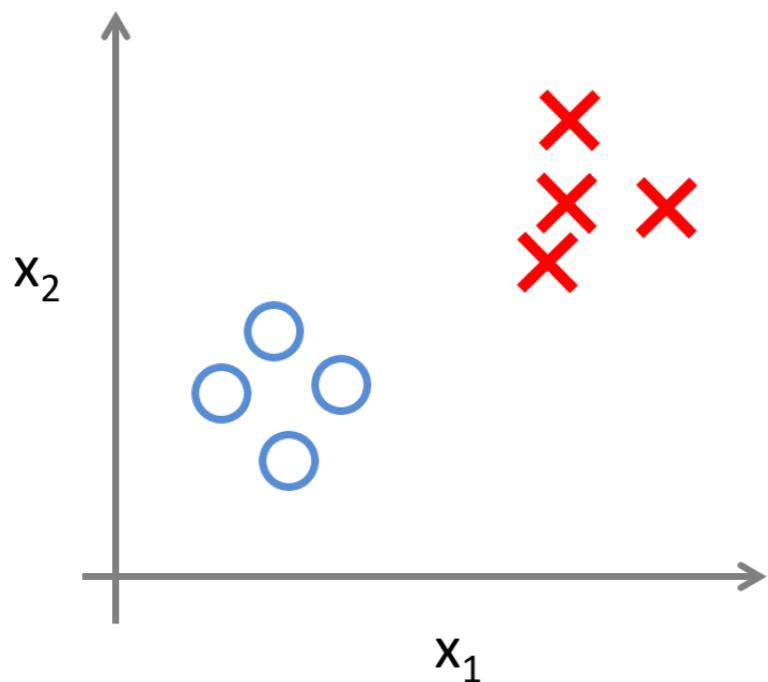
$$Z = \sum_{\mathbf{S}} e^{-\beta E(\mathbf{S})}$$

# Outline

- Tensor networks for representations  
*(in a small space)*
    - CP, Tucker, MPS, Tree Tensor networks
    - Example: Compressing neural networks
  - Contraction of tensor networks
- Tensor networks for learning  
*(in a large space)*
- Hilbert space as feature space: classification and PCA using tensor networks
  - Tensor networks for generative modeling

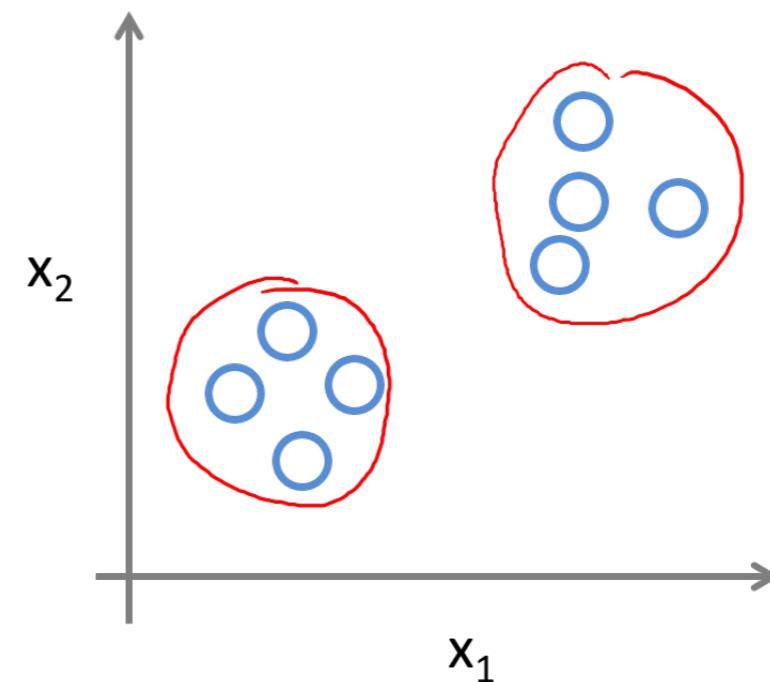
# Learning problems

Supervised Learning



Predicting Labels

Unsupervised Learning



Finding structures in the data

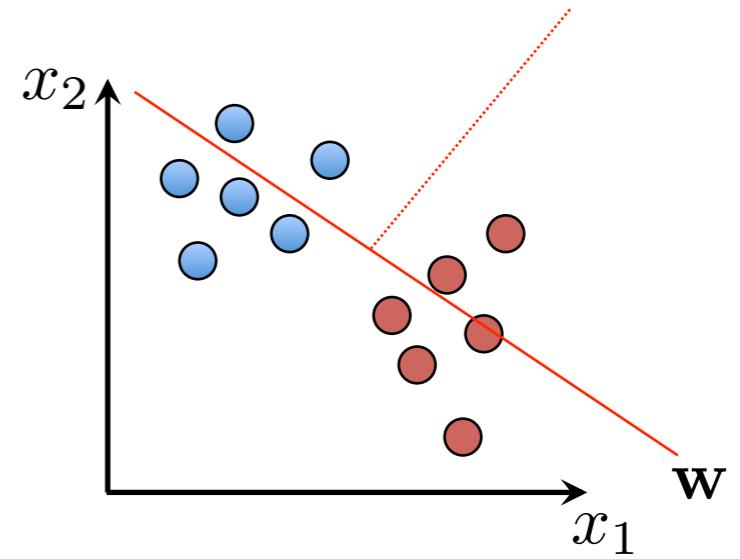
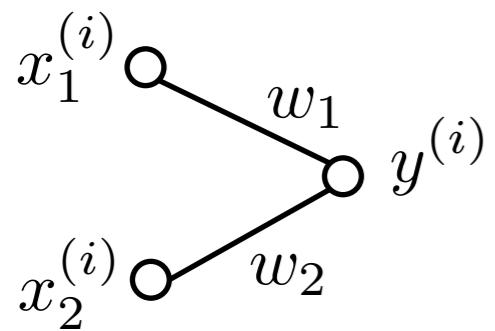
# Linear classifier

$$\mathbf{y} = \Theta(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

# Linear classifier

$$\mathbf{y} = \Theta(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

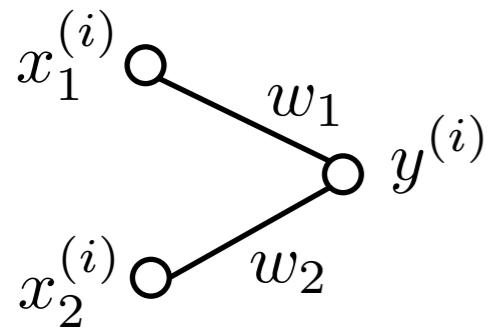
$$y^{(i)} = \Theta(w_1 x_1^{(i)} + w_2 x_2^{(i)} + b_i)$$



# Linear classifier

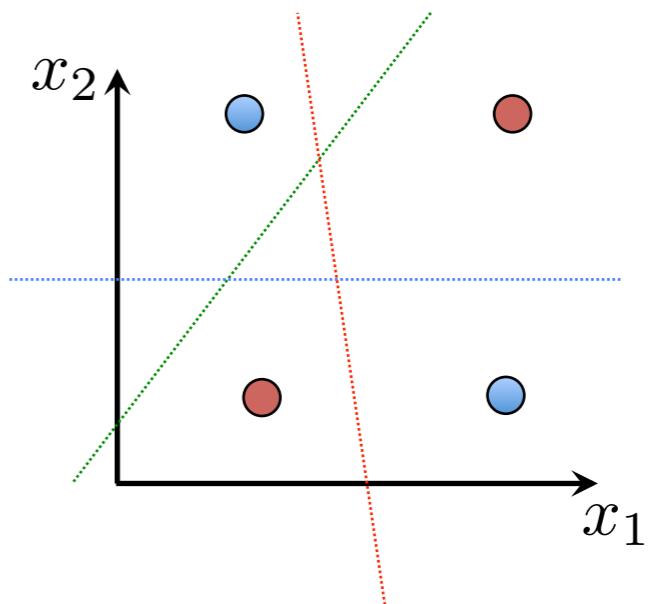
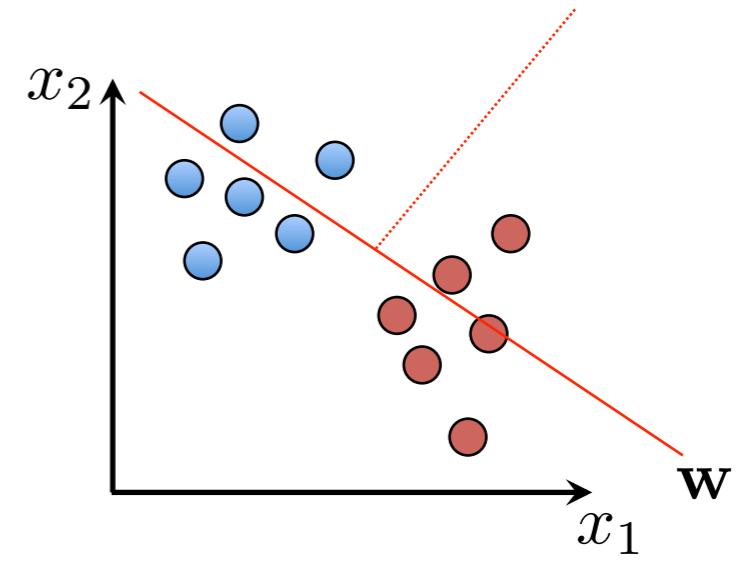
$$\mathbf{y} = \Theta(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

$$y^{(i)} = \Theta(w_1 x_1^{(i)} + w_2 x_2^{(i)} + b_i)$$



Sometimes not possible : not linear separable in 2-D.

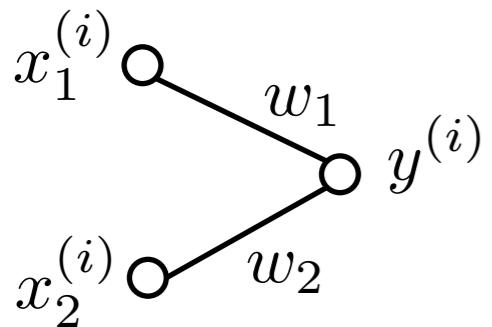
[Minsky, Paper 1969]



# Linear classifier

$$\mathbf{y} = \Theta(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

$$y^{(i)} = \Theta(w_1 x_1^{(i)} + w_2 x_2^{(i)} + b_i)$$

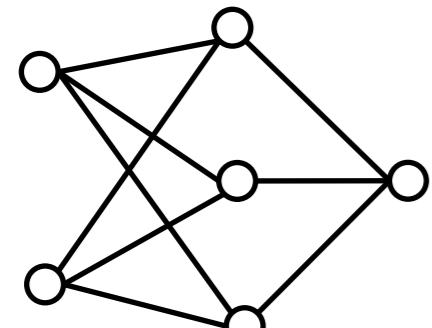


Sometimes not possible : not linear separable in 2-D.

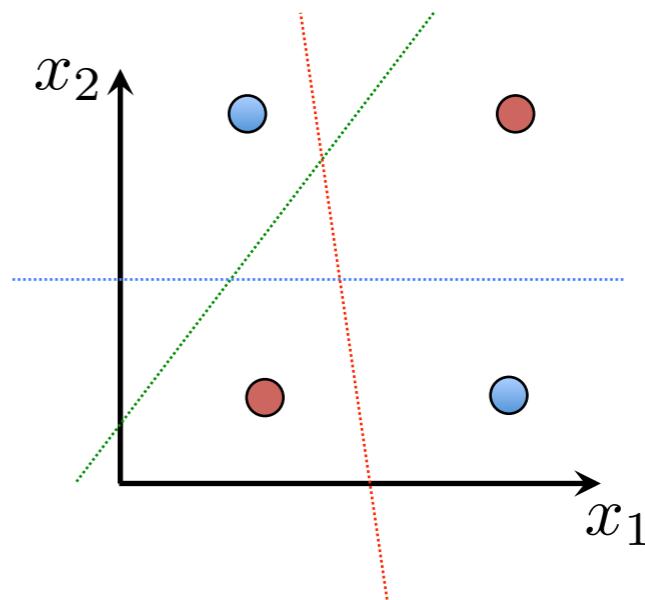
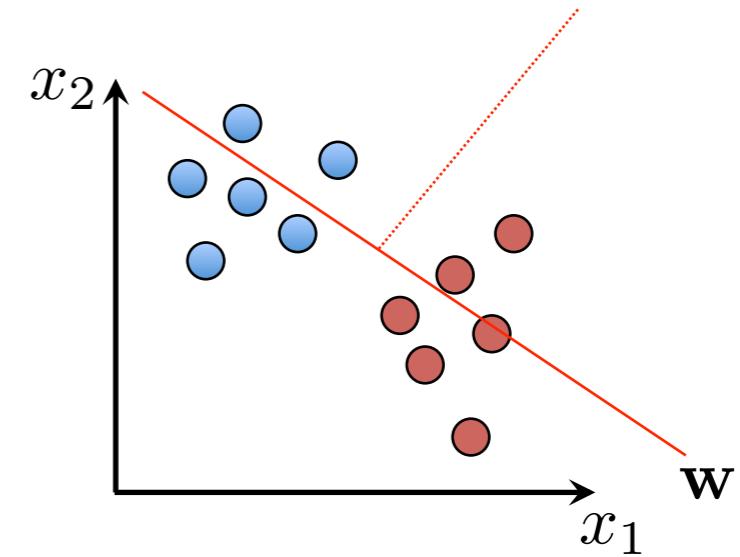
[Minsky, Paper 1969]

But could be linearly separable in a larger space

$$y^{(i)} = \Theta(\mathbf{W}\sigma(\mathbf{F}\mathbf{x}^{(i)}) + b_i)$$



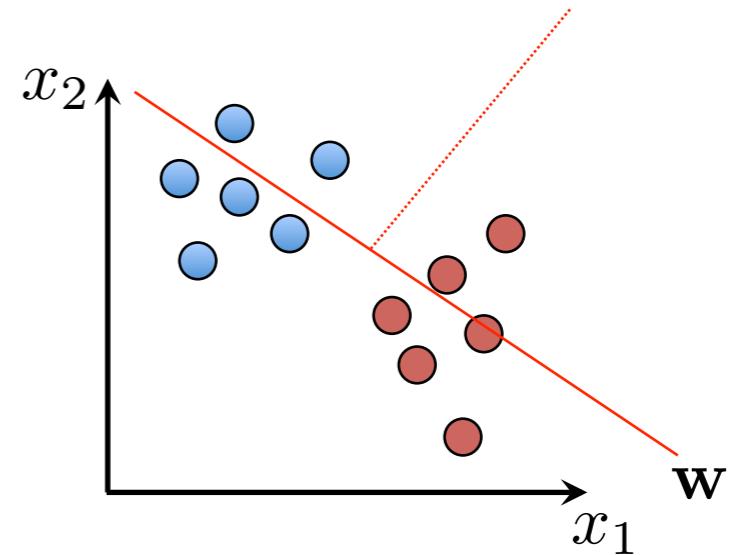
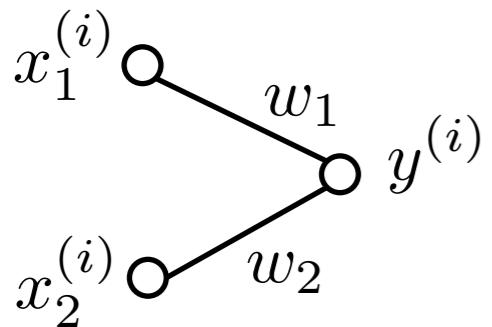
$$\mathcal{R}^2 \rightarrow \mathcal{R}^3$$



# Linear classifier

$$\mathbf{y} = \Theta(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

$$y^{(i)} = \Theta(w_1 x_1^{(i)} + w_2 x_2^{(i)} + b_i)$$

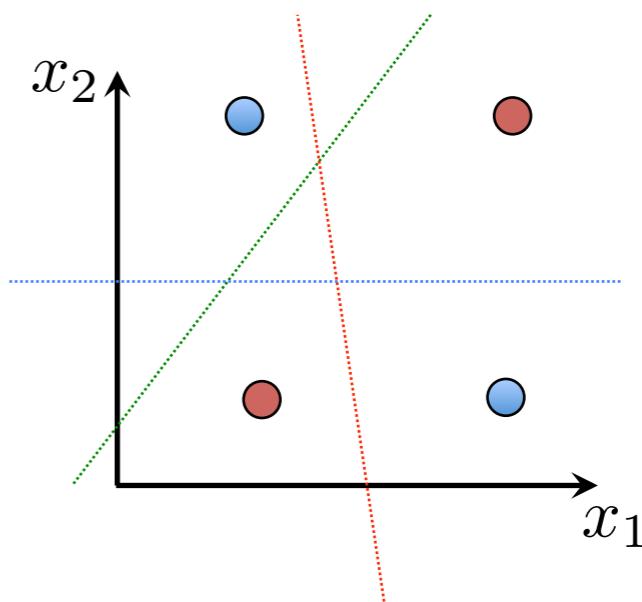
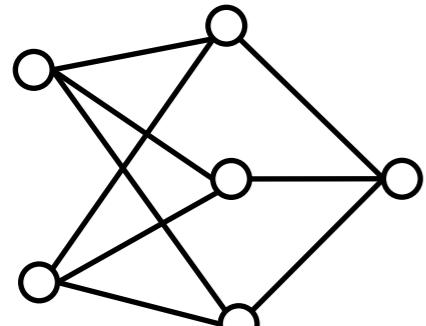


Sometimes not possible : not linear separable in 2-D.

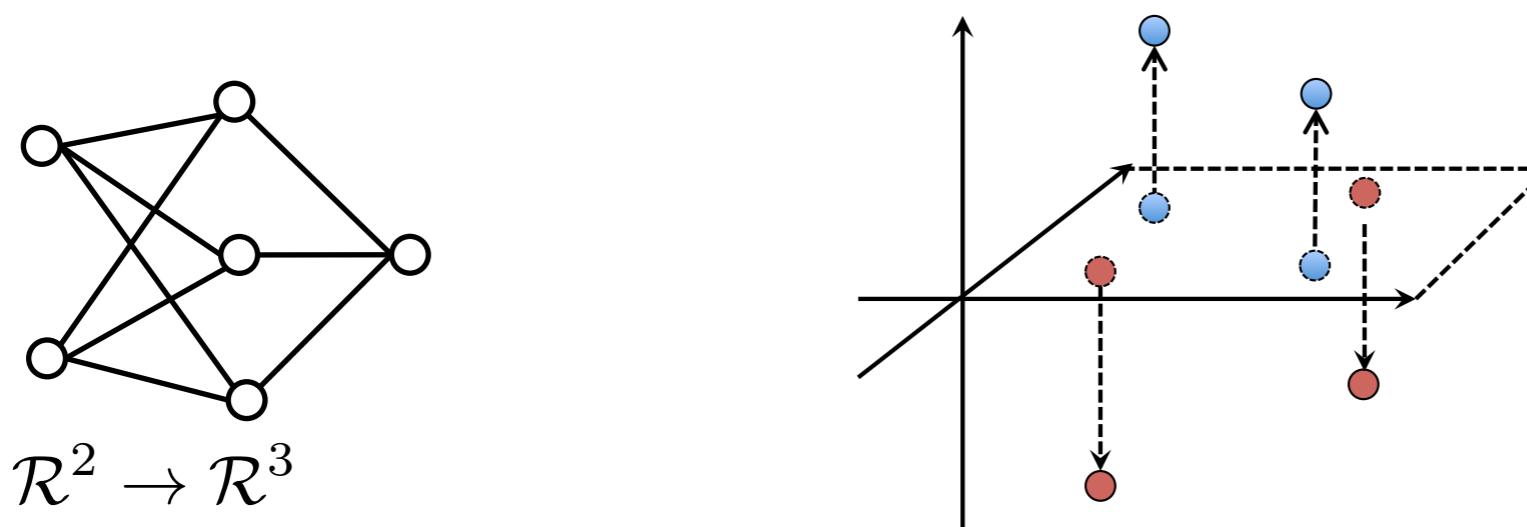
[Minsky, Paper 1969]

But could be linearly separable in a larger space

$$y^{(i)} = \Theta(\mathbf{W}\sigma(\mathbf{F}\mathbf{x}^{(i)}) + b_i)$$



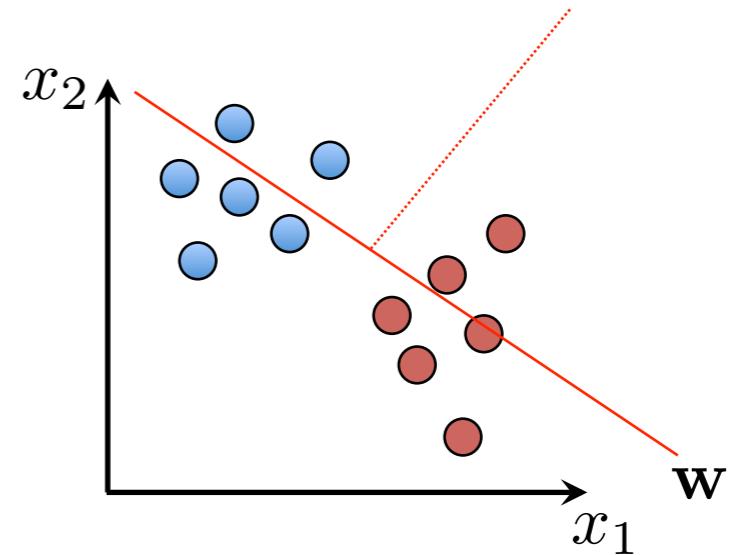
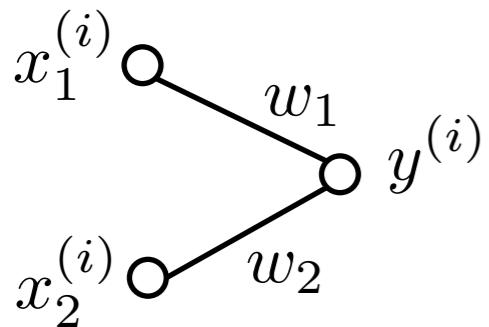
*Feature Space*



# Linear classifier

$$\mathbf{y} = \Theta(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

$$y^{(i)} = \Theta(w_1 x_1^{(i)} + w_2 x_2^{(i)} + b_i)$$

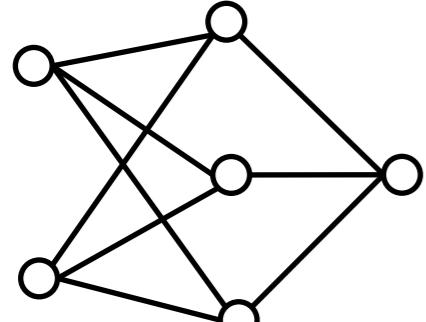


Sometimes not possible : not linear separable in 2-D.

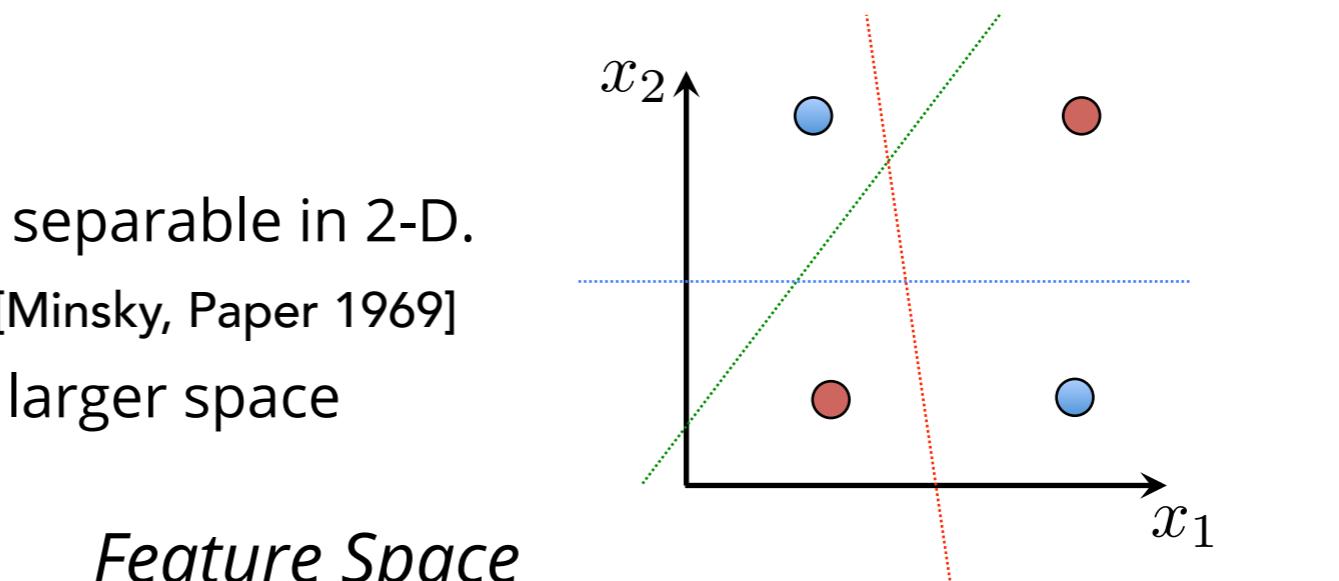
[Minsky, Paper 1969]

But could be linearly separable in a larger space

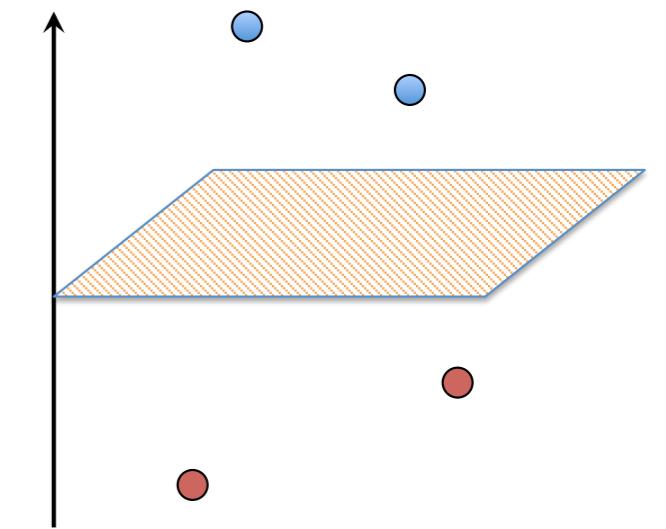
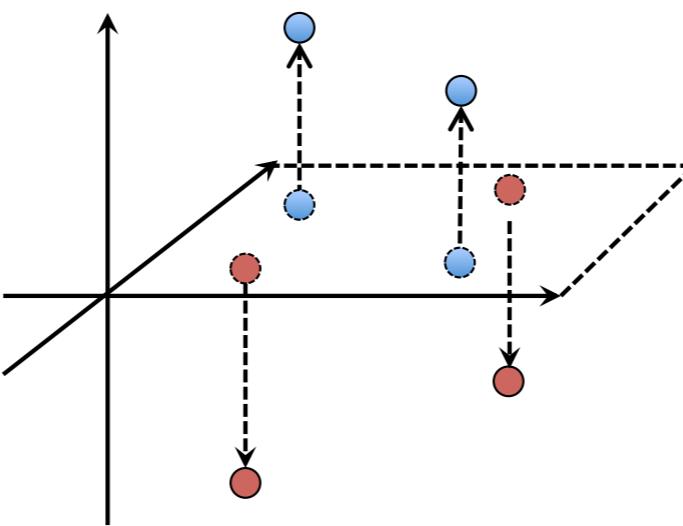
$$y^{(i)} = \Theta(\mathbf{W}\sigma(\mathbf{F}\mathbf{x}^{(i)}) + b_i)$$



$$\mathcal{R}^2 \rightarrow \mathcal{R}^3$$

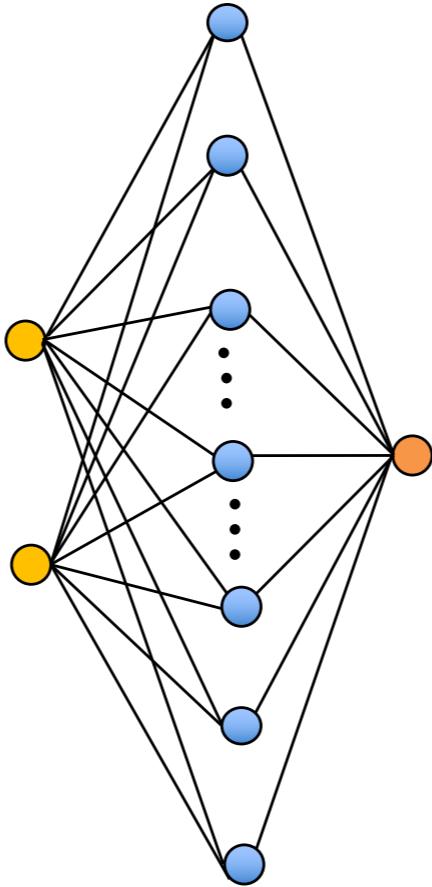
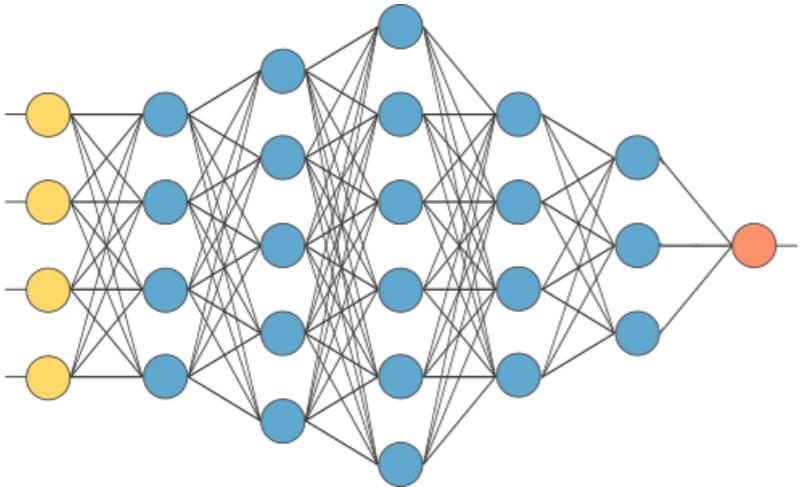


*Feature Space*



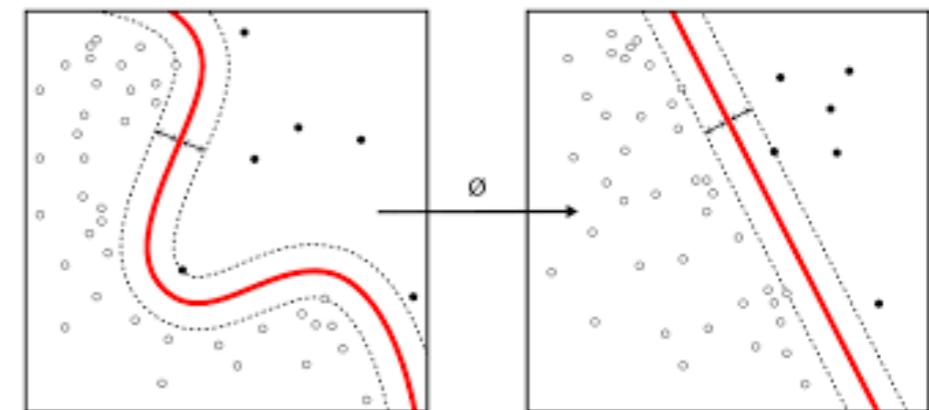
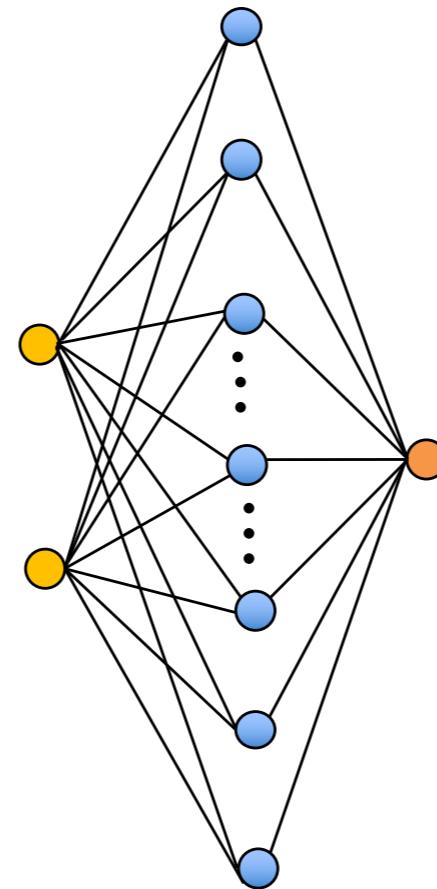
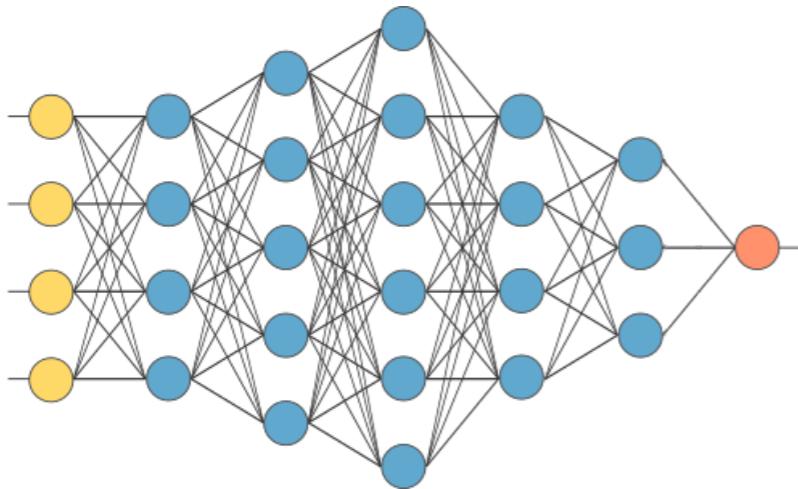
# Kernel learning

# Kernel learning



Neural network with an infinite-large hidden layer can represent any function.

# Kernel learning

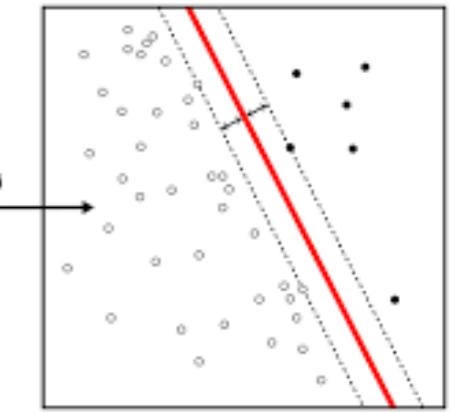
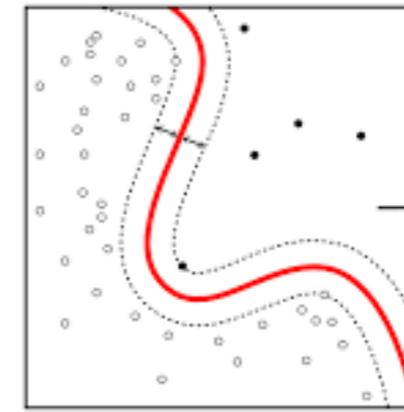
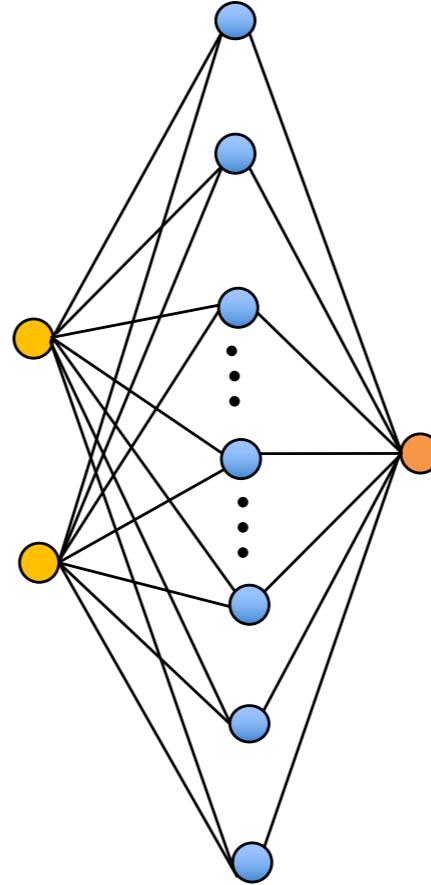
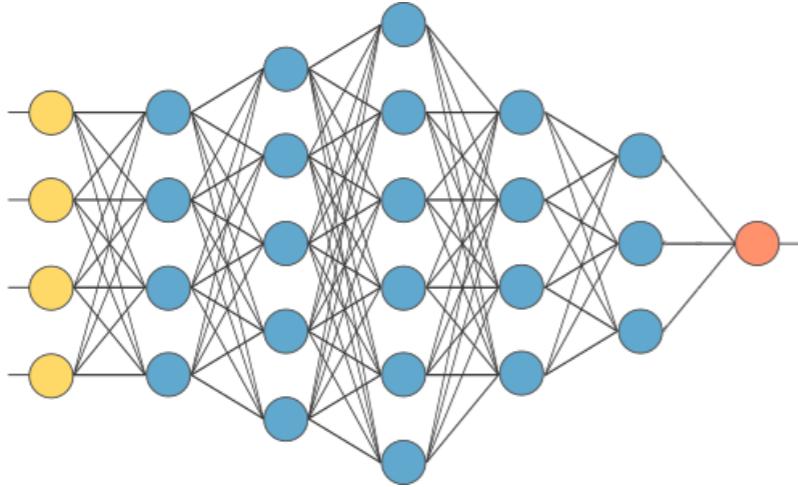


*Image courtesy: Wikipedia*

Neural network with an infinite-large hidden layer can represent any function.

Kernel support vector machine is a classic example.

# Kernel learning



*Image courtesy: Wikipedia*

Neural network with an infinite-large hidden layer can represent any function.

Kernel support vector machine is a classic example.

Conventional kernel methods does not construct explicitly the feature space. Instead, it evaluates inner product of vectors in the feature space, or a *kernel function*.

# Feature mapping to Hilbert space

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9

MNIST handwritten digits with labels 0,1,...,9

60,000 training images

10,000 test images

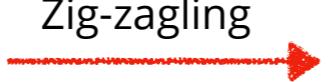
# Feature mapping to Hilbert space

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9

MNIST handwritten digits with labels 0,1,...,9

60,000 training images

10,000 test images

$\mathbf{z} \in \{1, 0\}^{28 \times 28}$   ● ○ ● ● ○ ... ● ● ○  $\mathbf{x}$

# Feature mapping to Hilbert space

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9

MNIST handwritten digits with labels 0,1,...,9

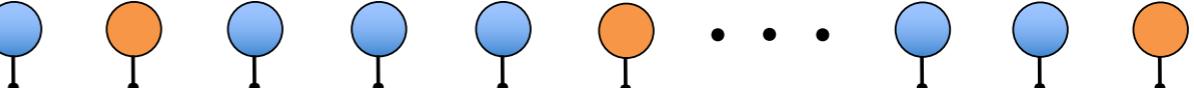
60,000 training images

10,000 test images

$\mathbf{z} \in \{1, 0\}^{28 \times 28}$     $\mathbf{x}$

●  $0 \rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}$  

○  $1 \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix}$  

  $\Phi(\mathbf{x})$

# Feature mapping to Hilbert space

9 3 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9

MNIST handwritten digits with labels 0,1,...,9

60,000 training images

10,000 test images

$\mathbf{z} \in \{1, 0\}^{28 \times 28}$    $\bullet \quad \circ \quad \bullet \quad \bullet \quad \bullet \quad \circ \quad \dots \quad \bullet \quad \bullet \quad \circ \quad \mathbf{x}$

$\bullet \quad 0 \rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \bullet$   $\circ \quad 1 \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \circ$   $\Phi(\mathbf{x})$

$\left( \begin{matrix} 0 \\ 1 \end{matrix} \right) \otimes \left( \begin{matrix} 1 \\ 0 \end{matrix} \right) \otimes \left( \begin{matrix} 0 \\ 1 \end{matrix} \right) \otimes \left( \begin{matrix} 0 \\ 1 \end{matrix} \right) \otimes \left( \begin{matrix} 0 \\ 1 \end{matrix} \right) \otimes \left( \begin{matrix} 1 \\ 0 \end{matrix} \right) \otimes \dots \otimes \left( \begin{matrix} 0 \\ 1 \end{matrix} \right) \otimes \left( \begin{matrix} 0 \\ 1 \end{matrix} \right) \otimes \left( \begin{matrix} 1 \\ 0 \end{matrix} \right)$

Dimension:  $2^{28 \times 28} = 2^{784}$

Rebentrost, Mohseni, Lloyd, Phys. Rev. Lett. 113, 130503 (2014)

Havlíček, Córcoles, Temme, Harrow, Kandala, Chow, Gambetta, Nature 567, 209 (2019)

Stoudenmire, Schwab, NIPS (2016 )

# MPS as a classifier in the Hilbert feature space

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9

MNIST handwritten digits with labels 0,1,...,9

Tensor-product feature map:

$$\bullet = \sin\left(\frac{\pi}{2}x_i\right) \bullet + \cos\left(\frac{\pi}{2}x_i\right) \bullet$$

$$\mathbf{F}(\text{6}) = \mathbf{F}(\mathbf{x}_i) = \mathbf{F}(\bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet) = \begin{array}{cccccc} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{array}$$

$$\text{Loss}(\bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet) = \sum_{i \in \text{data}} \|\mathbf{F}(\mathbf{x}_i) - \mathbf{y}_i\|_2^2$$

# Learning algorithm analogous to DMRG

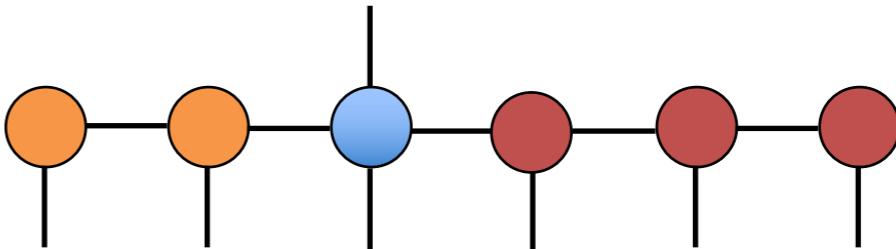
$$\text{Diagram} = \arg \min_{\text{Diagram}} \sum_{i \in \text{data}} \left\| \text{Diagram} - \mathbf{y}_i \right\|^2$$

The diagram illustrates a learning algorithm. On the left, a horizontal sequence of six orange circles is shown, with vertical lines connecting each circle to a single vertical line at the right end. This represents the target or desired state. An equals sign follows this diagram. To the right of the equals sign is a mathematical expression:  $\arg \min_{\text{Diagram}} \sum_{i \in \text{data}} \left\| \text{Diagram} - \mathbf{y}_i \right\|^2$ . Below the equals sign, there is a small diagram showing a horizontal sequence of blue circles connected by horizontal lines, with vertical lines connecting each circle to a single vertical line at the right end, representing a learned state.

# Learning algorithm analogous to DMRG

$$\text{Diagram of a linear chain of orange nodes} = \arg \min_{\text{Diagram of blue nodes}} \sum_{i \in \text{data}} \left\| \text{Diagram of blue nodes} - \mathbf{y}_i \right\|^2$$

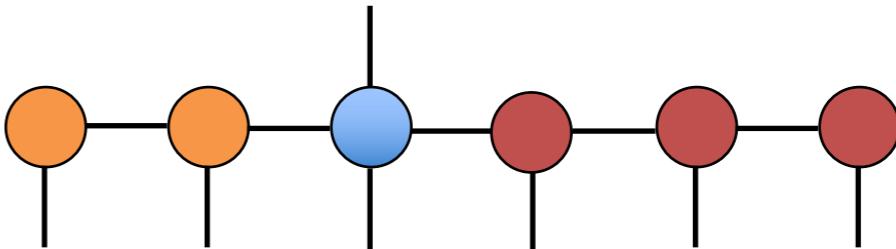
Mixed canonical form



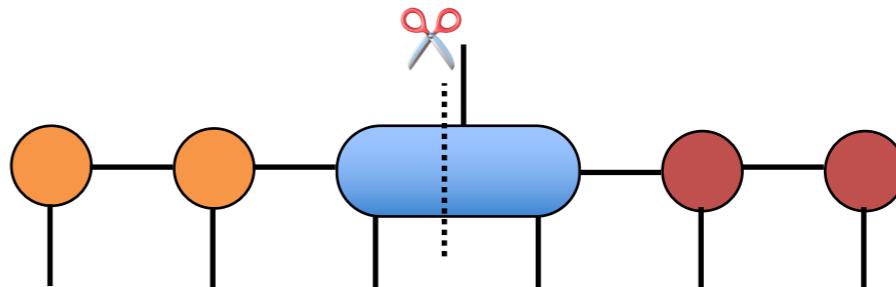
# Learning algorithm analogous to DMRG

$$\text{Diagram of a linear chain of orange nodes} = \arg \min_{\text{Diagram of blue nodes}} \sum_{i \in \text{data}} \left\| \text{Diagram of blue nodes} - \mathbf{y}_i \right\|^2$$

Mixed canonical form



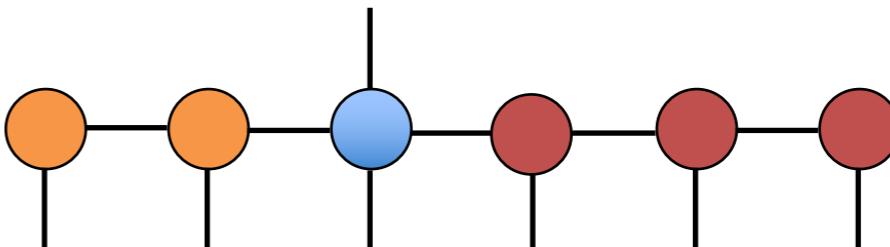
Merging + gradient descent



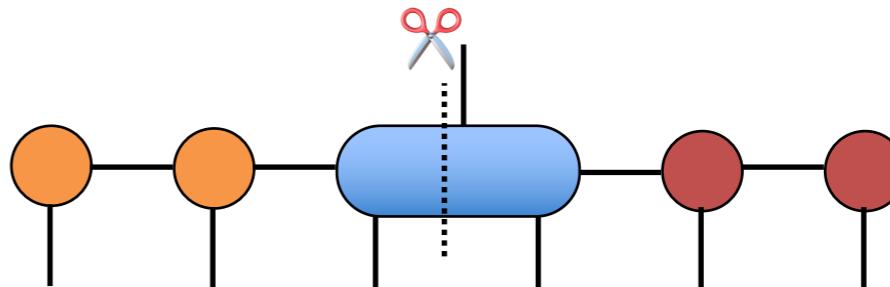
# Learning algorithm analogous to DMRG

$$\text{Diagram of a neural network layer with orange nodes} = \arg \min_{\text{Diagram of a neural network layer with blue nodes}} \sum_{i \in \text{data}} \left\| \text{Diagram of a neural network layer with blue nodes} - \mathbf{y}_i \right\|^2$$

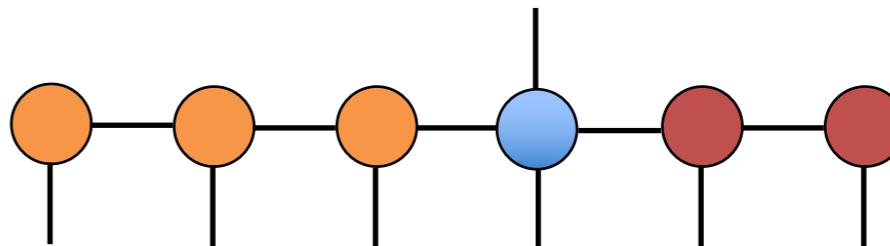
Mixed canonical form



Merging + gradient descent



Applying SVD

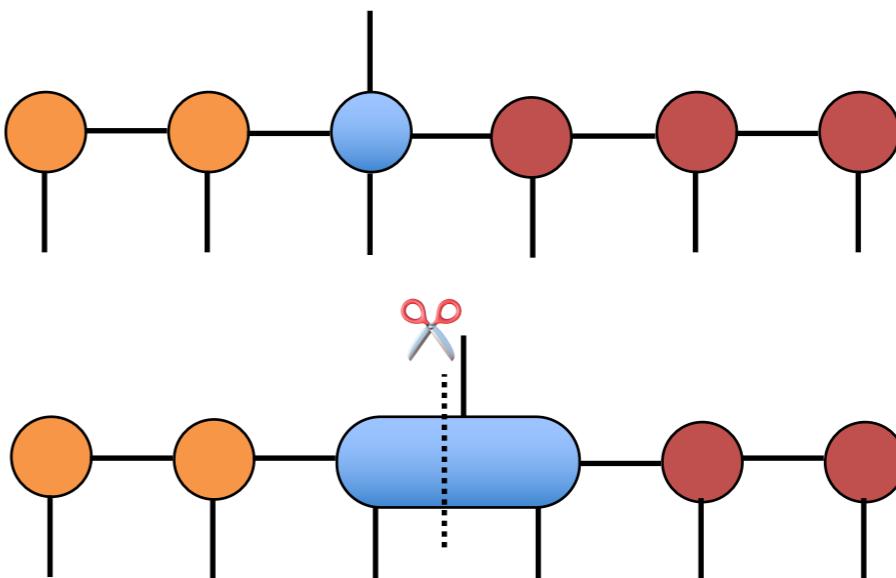


# Learning algorithm analogous to DMRG

$$\text{Diagram} = \arg \min_{\text{Diagram}} \sum_{i \in \text{data}} \left\| \text{Diagram} - \mathbf{y}_i \right\|^2$$

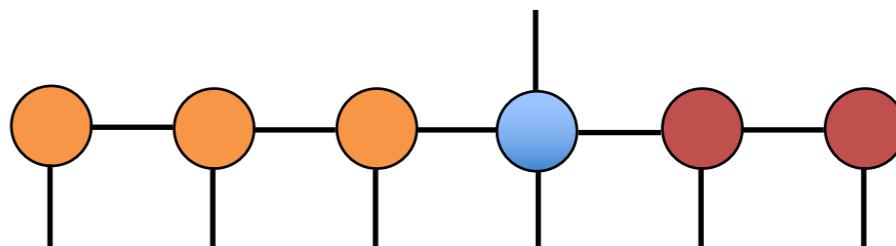
Strengths:

- explicit feature space
- adaptive bond-dimension
- canonical form, well-conditioned

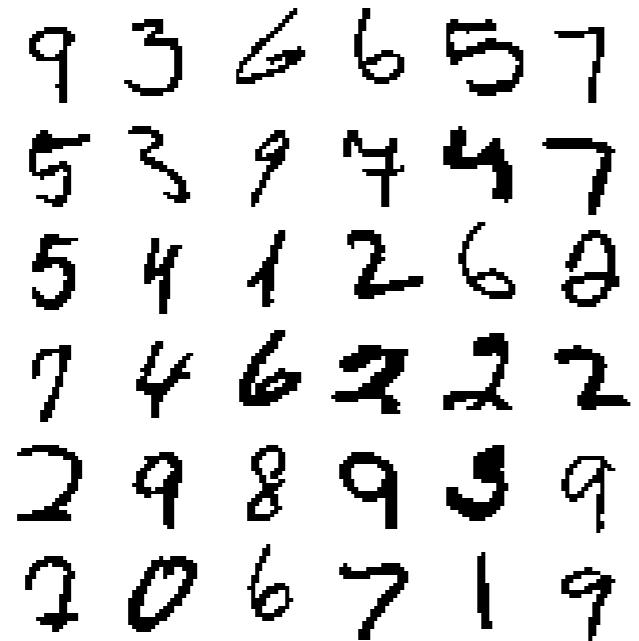


Limitations:

- sequential update, non parallel
- no gradient history, momentum, Adam...
- exponential decay of correlations



# Results on MNIST



MNIST handwritten digits with labels 0,1,...,9

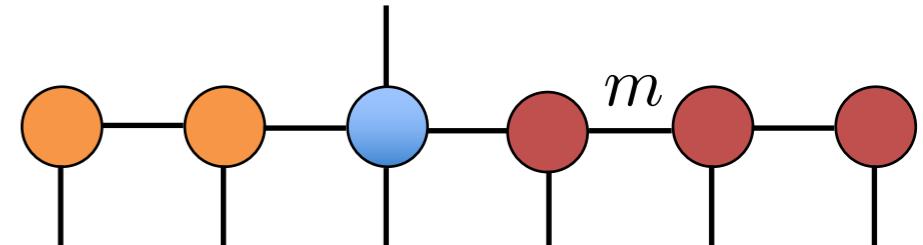
Tensor-product feature map:

$$\bullet = \sin\left(\frac{\pi}{2}x_i\right) \bullet + \cos\left(\frac{\pi}{2}x_i\right) \bullet$$

Croase grained to 14x14

Only takes several sweeps to converge !

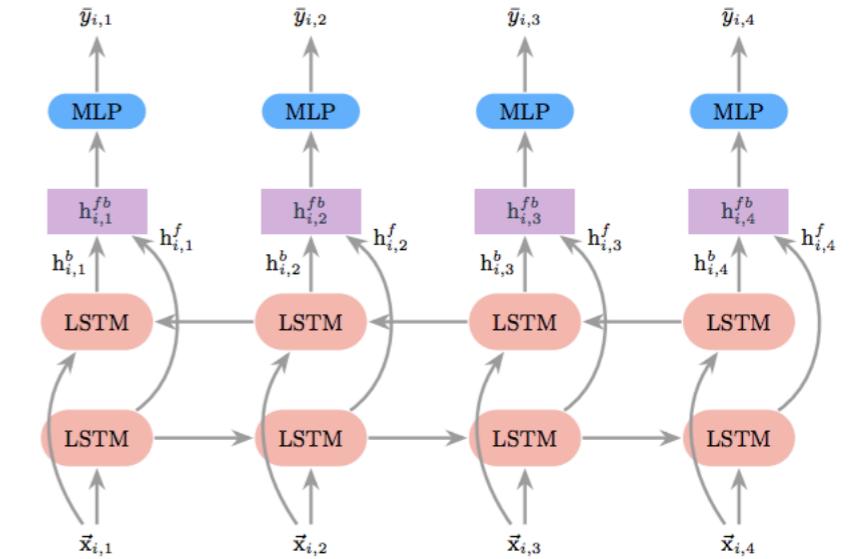
Bond dimension	Test Set Error	
$m = 10$	~5%	(500/10,000 incorrect)
$m = 20$	~2%	(200/10,000 incorrect)
$m = 120$	0.97%	(97/10,000 incorrect)



# Sequence-to-sequence learning using MPO

Sequence to sequence learning has been widely used in natural language processing and machine translation etc.

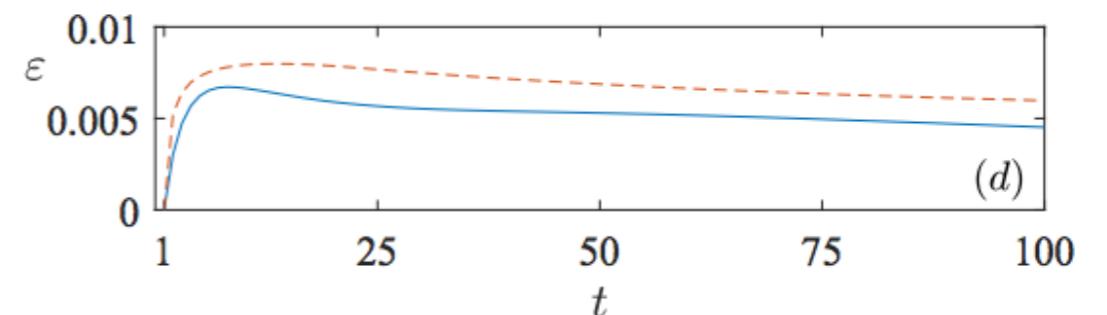
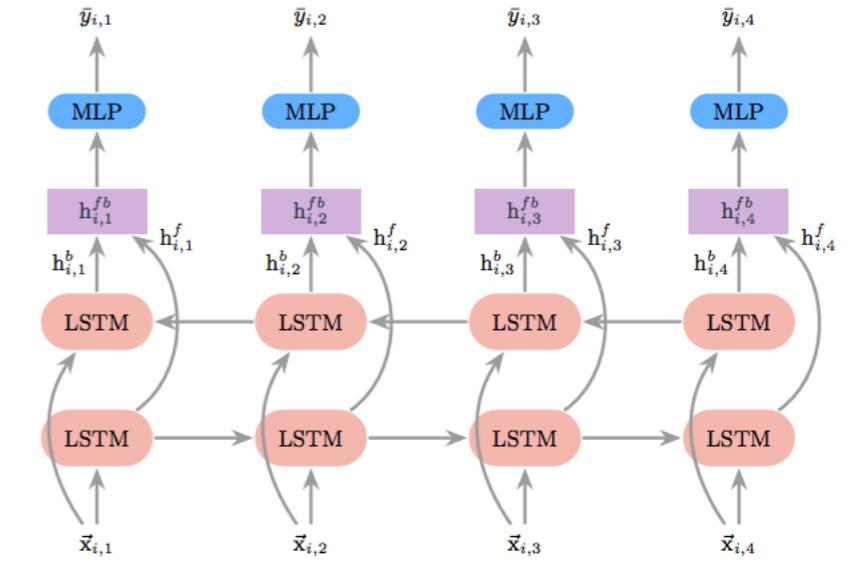
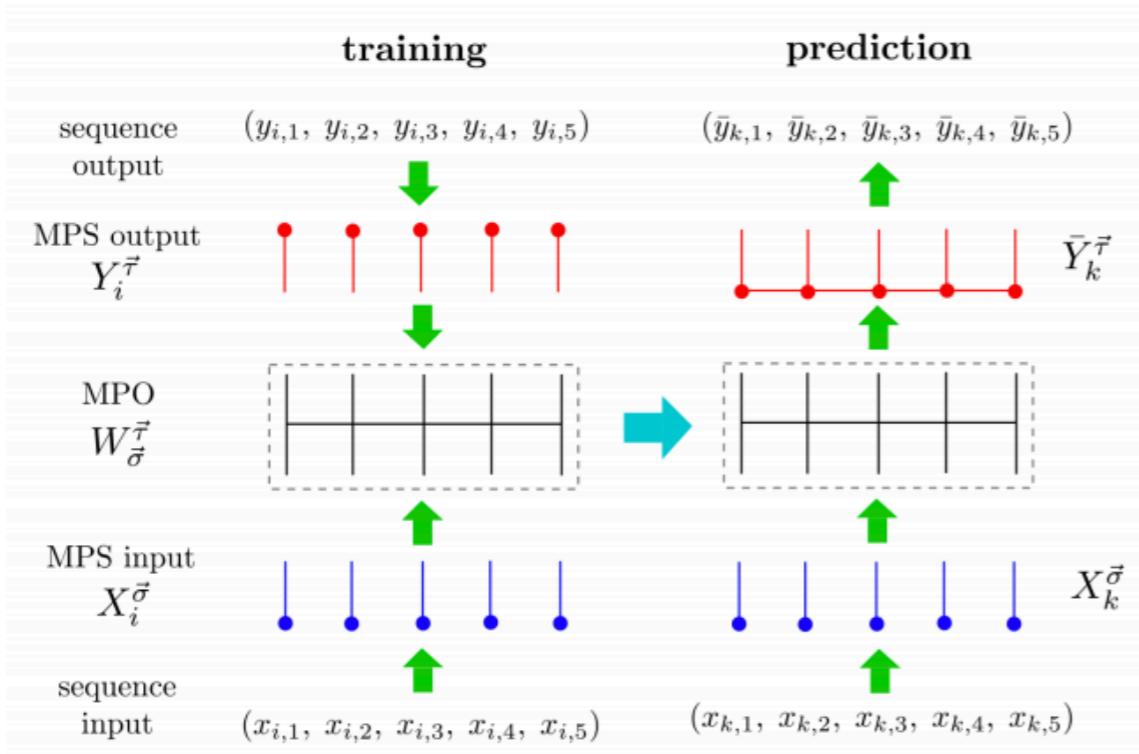
State-of-the-arts methods are based on RNN, especially the LSTM.



# Sequence-to-sequence learning using MPO

Sequence to sequence learning has been widely used in natural language processing and machine translation etc.

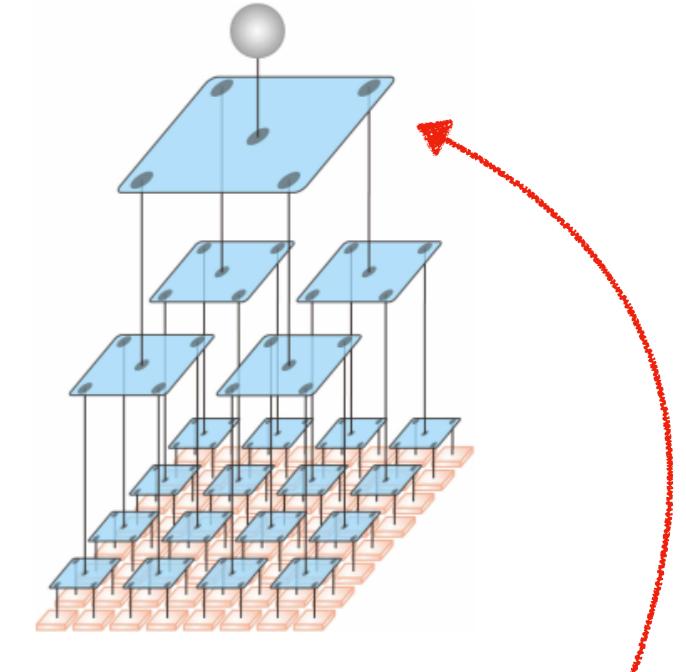
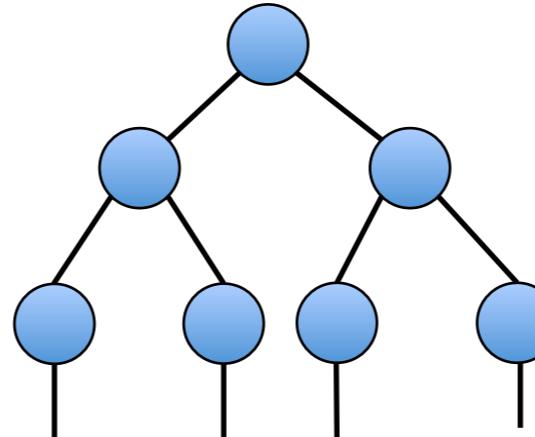
State-of-the-arts methods are based on RNN, especially the LSTM.



In some tasks MPO outperforms LSTM!

# Tree tensor network to MNIST classifications

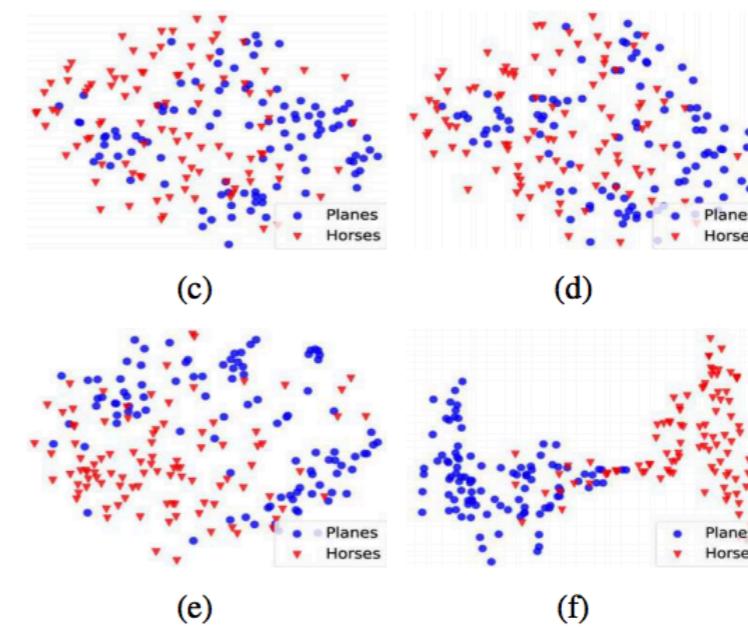
9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9



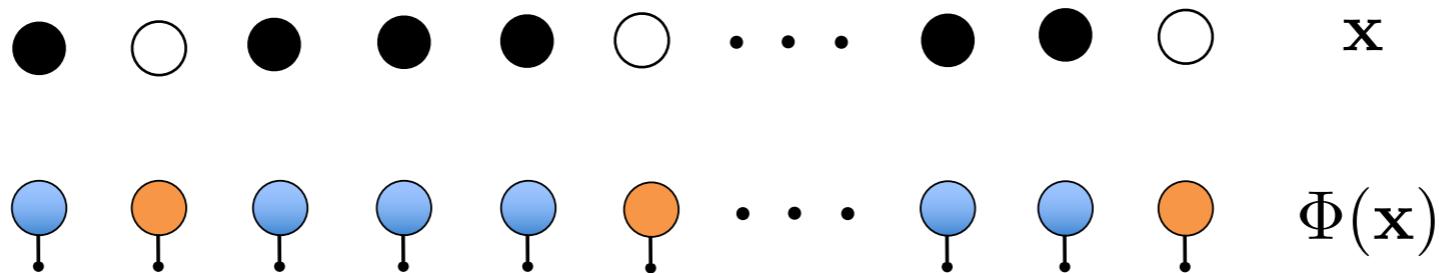
Using 2-D structures

Gradient-free: alternating least square

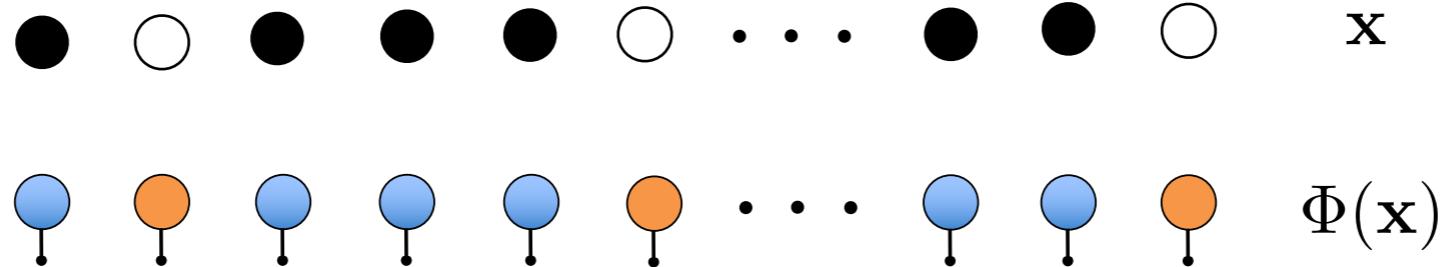
Dimensionality reduction at top layers



# Dimensionality reduction using kernel PCA



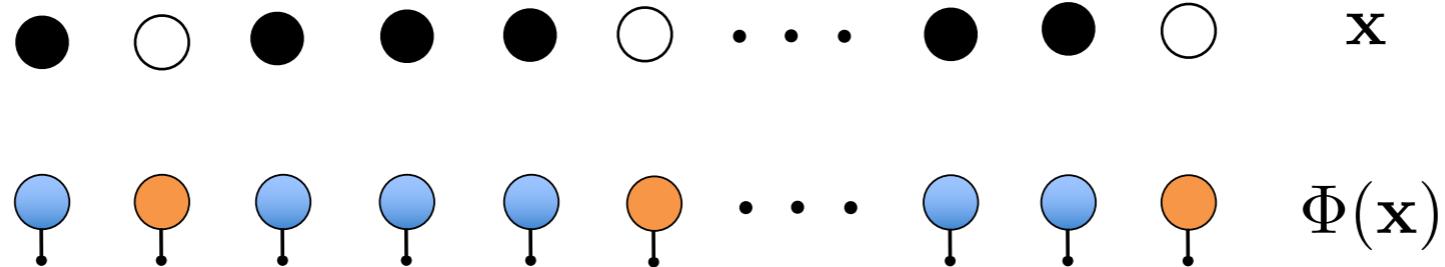
# Dimensionality reduction using kernel PCA



the optimal weights is expressed by span of the feature space.

— *representer theorem*

# Dimensionality reduction using kernel PCA

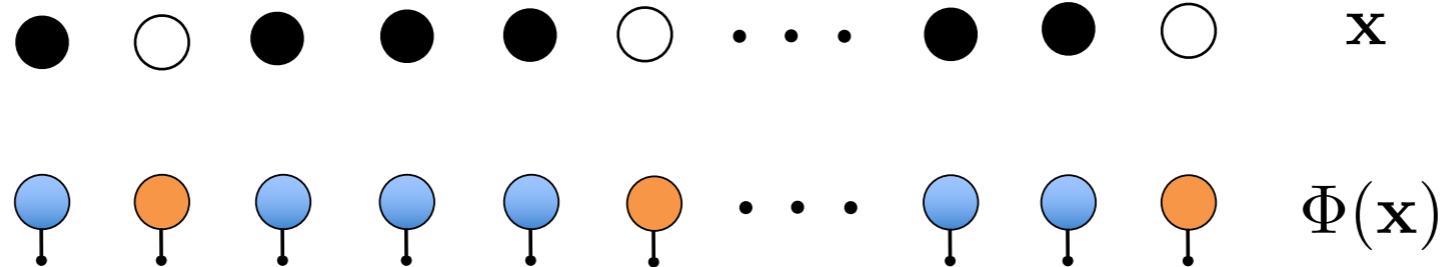


the optimal weights is expressed by span of the feature space.

— *representer theorem*

the feature space is very large, it could be of low-rank.

# Dimensionality reduction using kernel PCA



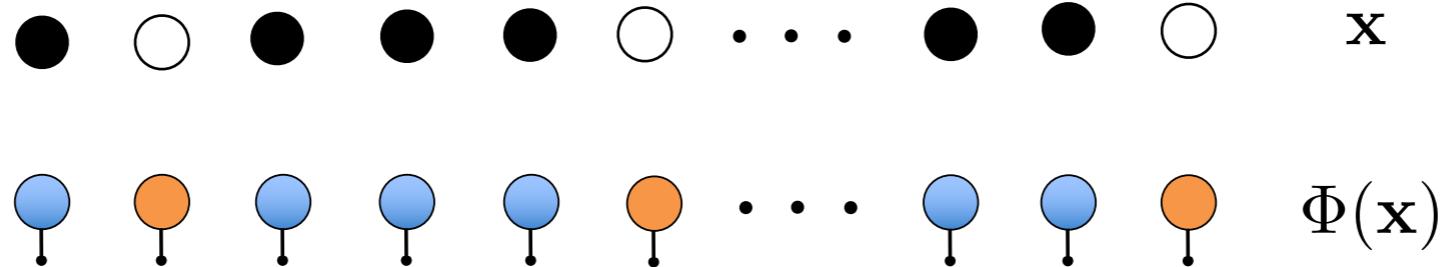
the optimal weights is expressed by span of the feature space.

— *representer theorem*

the feature space is very large, it could be of low-rank.

finding a set of good basis of the feature space amounts to finding relevant features of data.

# Dimensionality reduction using kernel PCA



the optimal weights is expressed by span of the feature space.

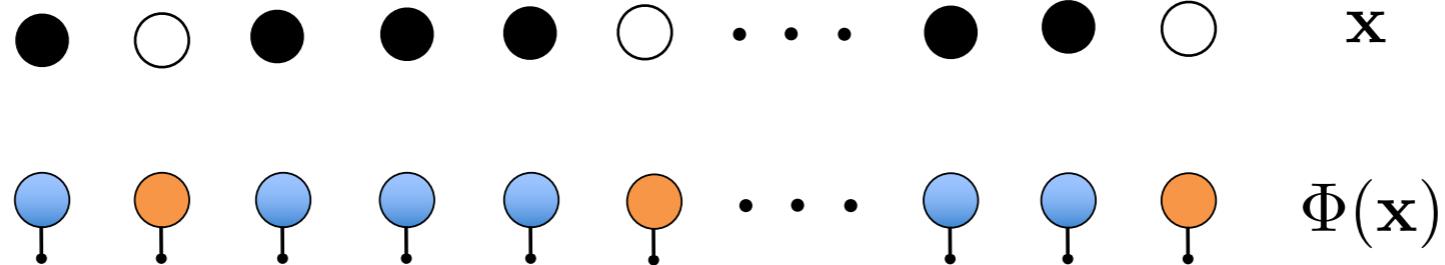
— *representer theorem*

the feature space is very large, it could be of low-rank.

finding a set of good basis of the feature space amounts to finding relevant features of data.

this is archived using PCA, which diagonalize the covariance matrix.

# Dimensionality reduction using kernel PCA



the optimal weights is expressed by span of the feature space.

— *representer theorem*

the feature space is very large, it could be of low-rank.

finding a set of good basis of the feature space amounts to finding relevant features of data.

this is archived using PCA, which diagonalize the covariance matrix.

In the quantum word, the covariance matrix is analogous to the density matrix

# Canonical PCA

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & | & \mathbf{x}_2 & | & \cdots & | & \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}$$

## Canonical PCA

$$\mathbf{X} = \begin{bmatrix} | & | & \dots & | & | \\ \mathbf{x}_i & & & & \end{bmatrix} = \begin{bmatrix} \mathbf{x}_i \\ \vdots \\ \mathbf{x}_i \end{bmatrix}$$

data matrix

$$\rho = \mathbf{X}^\top \mathbf{X} = \sum_{i=1}^m \mathbf{x}_i \circ \mathbf{x}_i = \sum_{i=1}^m \begin{bmatrix} \mathbf{x}_i \\ \vdots \\ \mathbf{x}_i \end{bmatrix} = \begin{bmatrix} \mathbf{x}_i \\ \vdots \\ \mathbf{x}_i \end{bmatrix}$$

rank of the covariance matrix could be lower than  $m$

## Canonical PCA

$$\mathbf{X} = \begin{bmatrix} & & \cdots & & \\ | & | & & | & | \\ \mathbf{x}_i & & & & \end{bmatrix} = \begin{bmatrix} & & \cdots & & \\ \text{---} & \text{---} & & \text{---} & \text{---} \\ \mathbf{x}_i & \mathbf{x}_i & \cdots & \mathbf{x}_i & \mathbf{x}_i \end{bmatrix}$$

data matrix

$$\rho = \mathbf{X}^\top \mathbf{X} = \sum_{i=1}^m \mathbf{x}_i \circ \mathbf{x}_i = \sum_{i=1}^m \begin{array}{c} \text{---} \\ \text{---} \\ \mathbf{x}_i \\ \text{---} \\ \text{---} \end{array} = \begin{array}{c} \text{---} \\ \text{---} \\ \mathbf{x}_i \\ \text{---} \\ \text{---} \end{array}$$

rank of the covariance  
matrix could be lower  
than  $m$

## Hilbert feature space PCA

$$\mathbf{x} \rightarrow \Phi(\mathbf{x}) = \begin{array}{cccccccccc} \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \cdots & \text{---} & \text{---} & \text{---} \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & \bullet \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & & \text{---} & \text{---} & \text{---} \end{array}$$

## Canonical PCA

$$\mathbf{X} = \begin{bmatrix} & & \cdots & & \\ | & | & & | & | \\ \mathbf{x}_i & & & & \end{bmatrix} = \begin{bmatrix} & & & & \\ \textcolor{blue}{\bullet} & \textcolor{orange}{\bullet} & & \textcolor{red}{\bullet} & \textcolor{purple}{\bullet} \\ | & | & & | & | \\ \mathbf{x}_i & & & & \end{bmatrix}$$

data matrix

$$\rho = \mathbf{X}^\top \mathbf{X} = \sum_{i=1}^m \mathbf{x}_i \circ \mathbf{x}_i = \sum_{i=1}^m \begin{array}{c} \textcolor{blue}{\bullet} \\ \textcolor{blue}{\bullet} \\ | \end{array} = \begin{array}{c} \textcolor{blue}{\bullet} \\ | \end{array}$$

rank of the covariance matrix could be lower than  $m$

## Hilbert feature space PCA

$$\mathbf{x} \rightarrow \Phi(\mathbf{x}) = \begin{array}{cccccccccc} \textcolor{blue}{\bullet} & \textcolor{orange}{\bullet} & \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} & \cdots & \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} & \textcolor{orange}{\bullet} \\ | & | & | & | & | & | & & | & | & | \end{array}$$

$$\begin{aligned} \rho &= \Phi(\mathbf{X})^\top \Phi(\mathbf{X}) = \sum_{i=1}^m \Phi(\mathbf{x}_i) \circ \Phi(\mathbf{x}_i) \\ &= \sum_{i=1}^m \begin{array}{cc} \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} \\ \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} \\ | & | \end{array} \cdots \begin{array}{cc} \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} \\ \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} \\ | & | \end{array} = \begin{array}{c} | & | & \cdots & | & | \\ \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} & \cdots & \textcolor{blue}{\bullet} & \textcolor{blue}{\bullet} \\ | & | & \cdots & | & | \end{array} \end{aligned}$$

# Dimensionality reduction

Hilbert feature covariance matrix is analogous to the density matrix

It is possibly of *low-rank*

# Dimensionality reduction

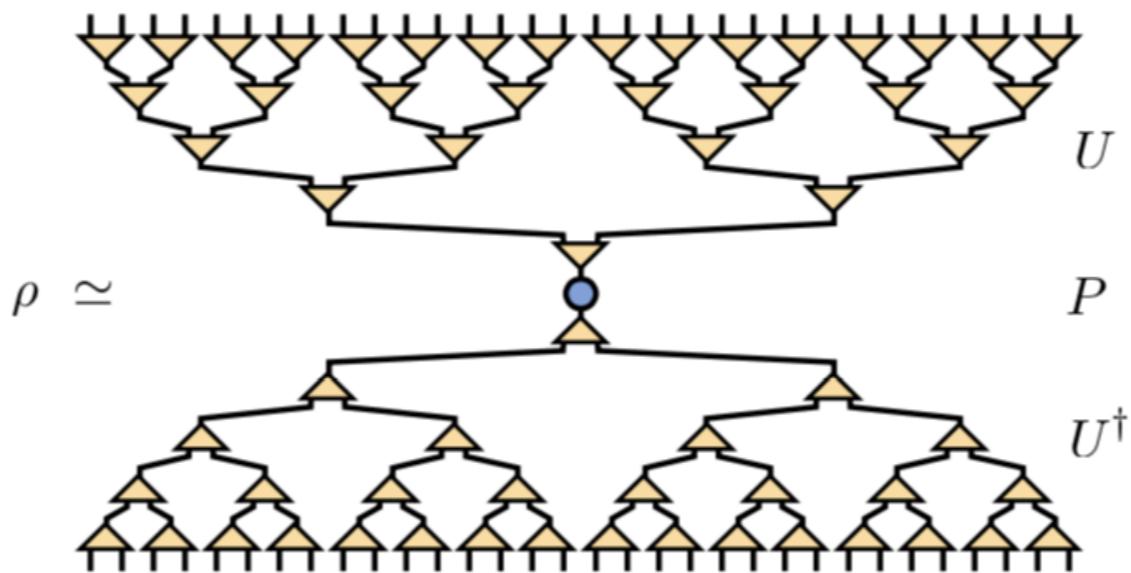
Hilbert feature covariance matrix is analogous to the density matrix

It is possible of *low-rank*

# Dimensionality reduction

Hilbert feature covariance matrix is analogous to the density matrix

It is possible of *low-rank*



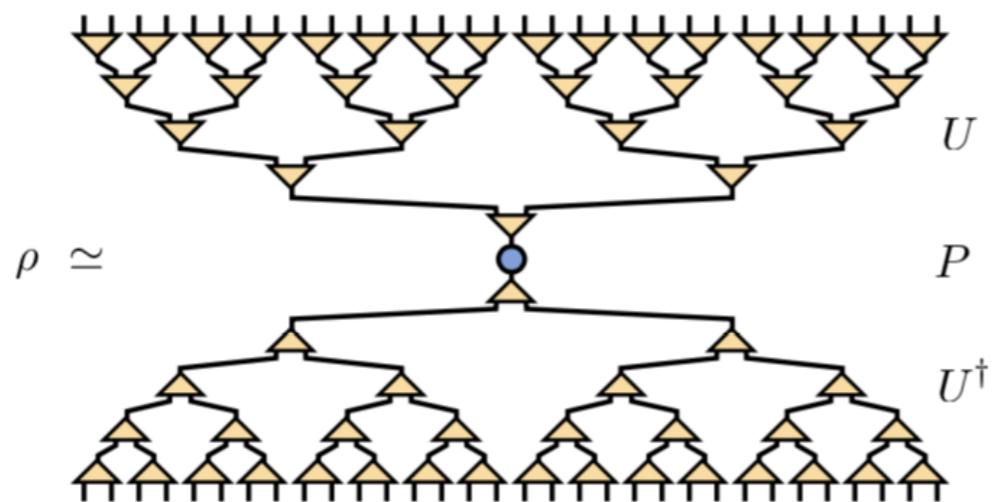
# Apply to the more challenge dataset fashion MNIST



28x28 grayscale images with 10 labels.

60,000 training images

10,000 test images



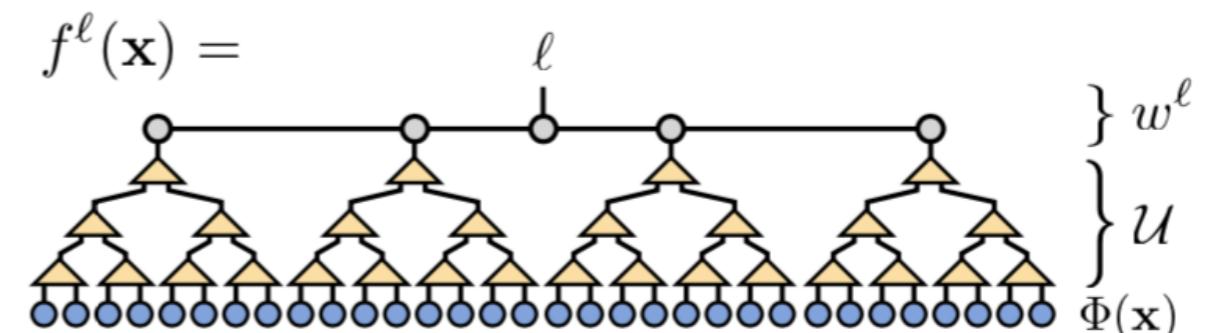
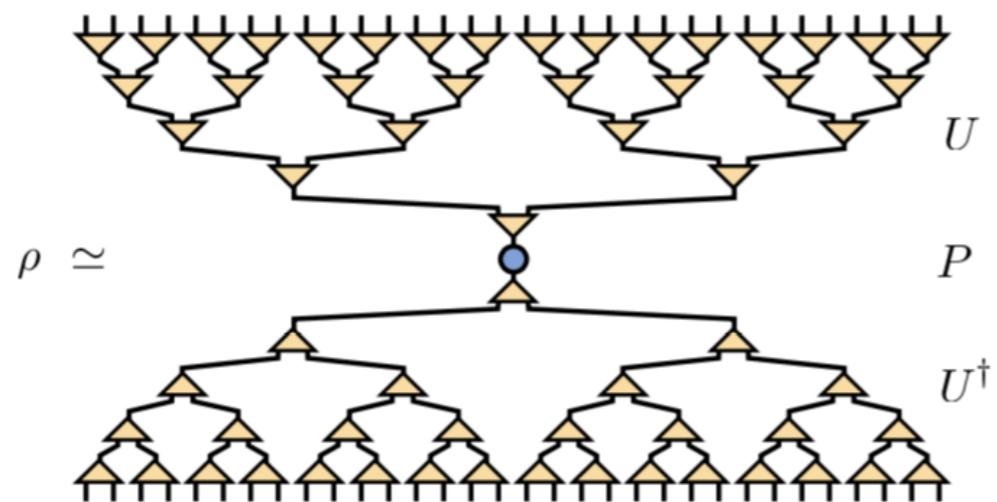
# Apply to the more challenge dataset fashion MNIST



28x28 grayscale images with 10 labels.

60,000 training images

10,000 test images



# Apply to the more challenge dataset fashion MNIST

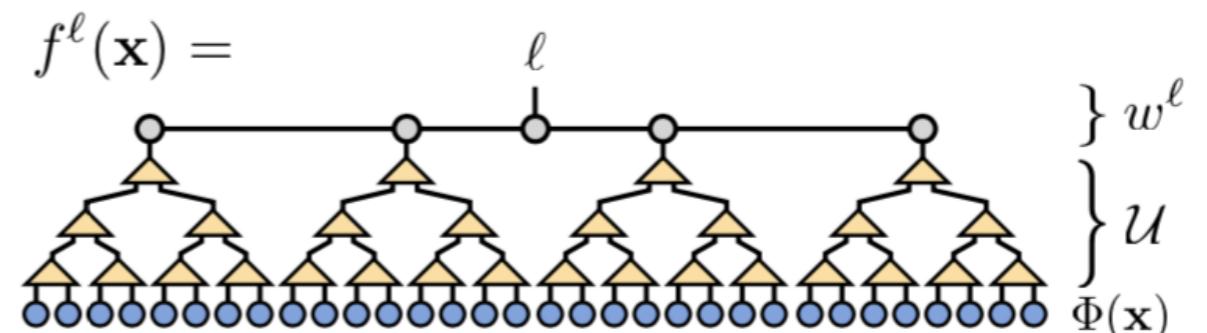


28x28 grayscale images with 10 labels.

60,000 training images

10,000 test images

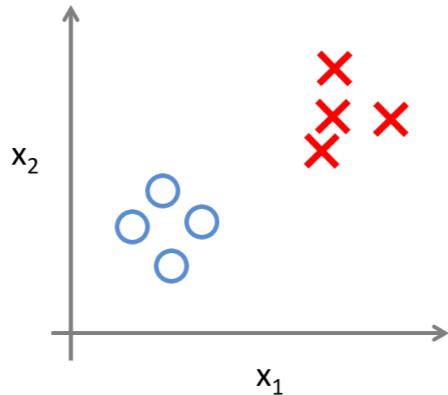
Method	Test Accuracy
AlexNet	88.90%
Tree tensor net. <sup>1</sup>	88.97%
String bond state <sup>2</sup>	89.2%
CNN-String bond state <sup>2</sup>	92.3%
GoogLeNet	93.7%



1. Stoudenmire, Quant. Sci. Tech., **3**, 034003 (2018)
2. Glasser, Pancotti, Cirac, arxiv:1806.05964

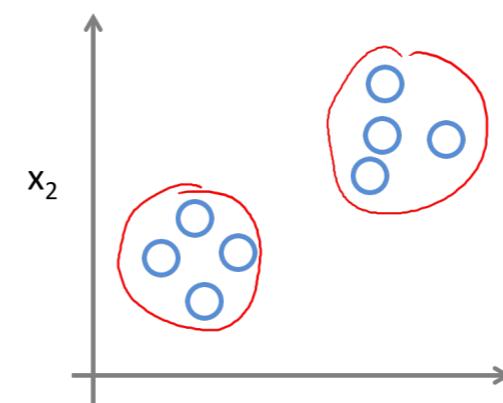
# From supervised to unsupervised learning

Supervised Learning



Predicting Labels

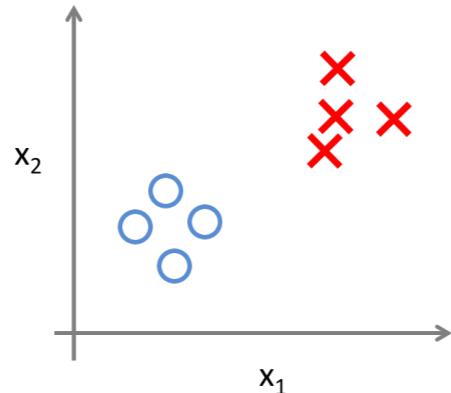
Unsupervised Learning



Finding structures in the data

# From supervised to unsupervised learning

Supervised Learning



Predicting Labels



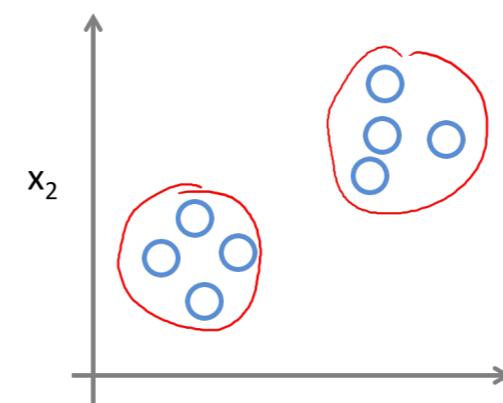
Discriminative

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

$$\text{or } p(y | \mathbf{x})$$

Classification  
Sequence prediction

Unsupervised Learning



Finding structures in the data



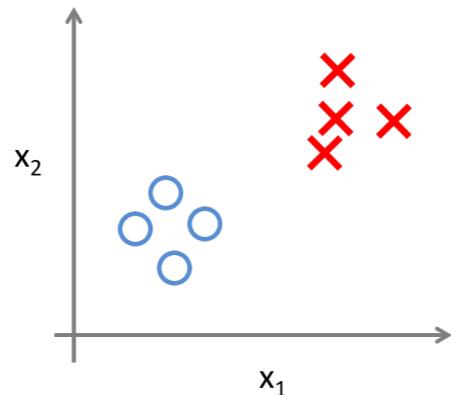
Generative

$$p(\mathbf{x}, y)$$

"fake image generation"

# From supervised to unsupervised learning

Supervised Learning



Predicting Labels



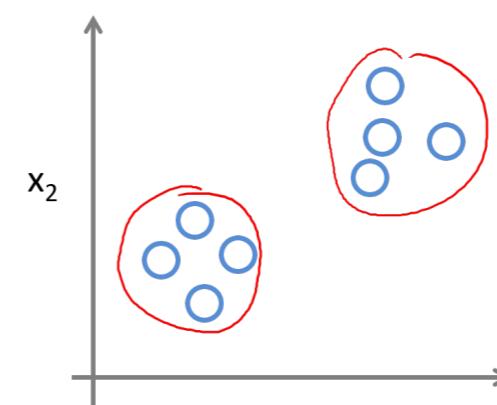
Discriminative

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

$$\text{or } p(y | \mathbf{x})$$

Classification  
Sequence prediction

Unsupervised Learning



Finding structures in the data



$$p(\mathbf{x}, y)$$

“fake image generation”

A machine learning generated print

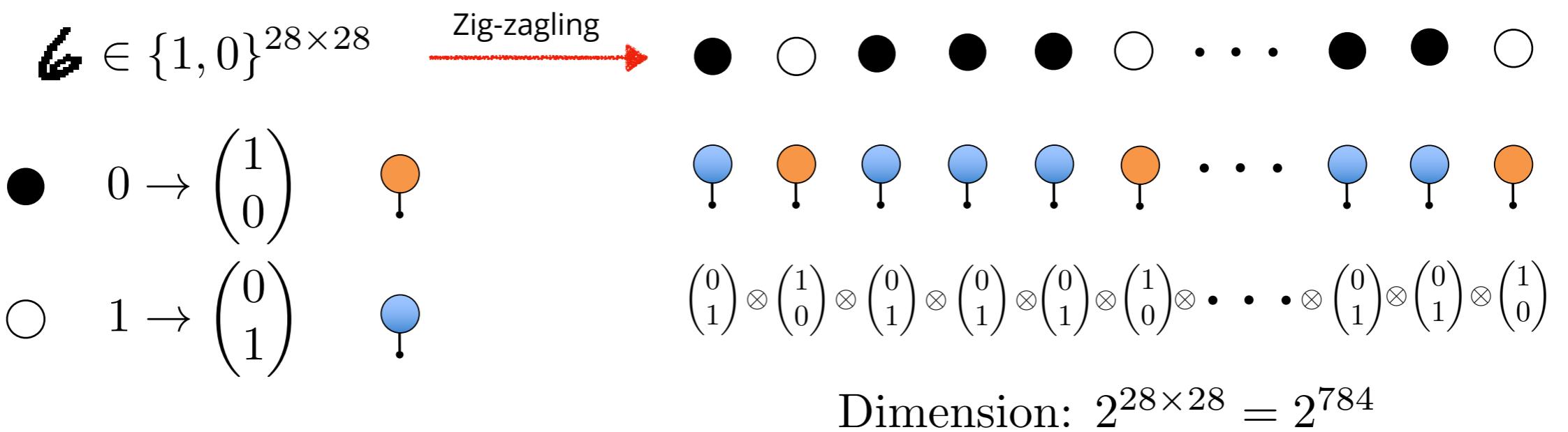
*sold for \$432,500*

# Feature mapping to Hilbert space

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9

The largest possible space for binary data

The right space for computing the “partition function”



# Parametrizing the joint probability distribution

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9

The largest possible space for binary data

The right space for computing the “partition function”

$$P(\text{6}) = P(\mathbf{x}) = P(\text{blue circle} \quad \text{orange circle} \quad \text{orange circle} \quad \text{blue circle} \quad \text{blue circle} \quad \text{orange circle}) = \frac{1}{Z} \left\| \text{graph} \right\|^2$$

# Parametrizing the joint probability distribution

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9

The largest possible space for binary data

The right space for computing the “partition function”

$$P(\text{6}) = P(\mathbf{x}) = P(\text{blue circle} \quad \text{orange circle} \quad \text{orange circle} \quad \text{blue circle} \quad \text{blue circle} \quad \text{orange circle}) = \frac{1}{Z} \left\| \begin{array}{cccccc} \text{blue circle} & \text{blue circle} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{orange circle} & \text{orange circle} & \text{orange circle} & \text{blue circle} & \text{blue circle} & \text{orange circle} \end{array} \right\|^2$$

*Born's rule*

# Parametrizing the joint probability distribution

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9

The largest possible space for binary data

The right space for computing the “partition function”

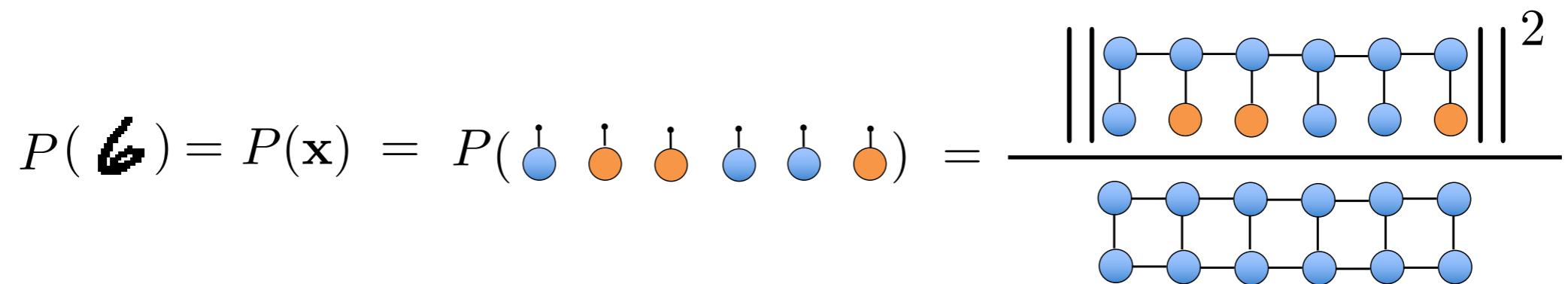
$$P(\text{6}) = P(\mathbf{x}) = P(\text{blue dot} \quad \text{orange dot} \quad \text{orange dot} \quad \text{blue dot} \quad \text{blue dot} \quad \text{orange dot}) = \frac{1}{Z} \left\| \text{blue dot} - \text{orange dot} - \text{blue dot} - \text{blue dot} - \text{blue dot} - \text{orange dot} \right\|^2$$

Z can be computed exactly !

*Born's rule*

$$Z = \text{blue dot} - \text{orange dot} - \text{orange dot} - \text{blue dot} - \text{red dot} - \text{red dot} = \boxed{\text{blue dot}}$$

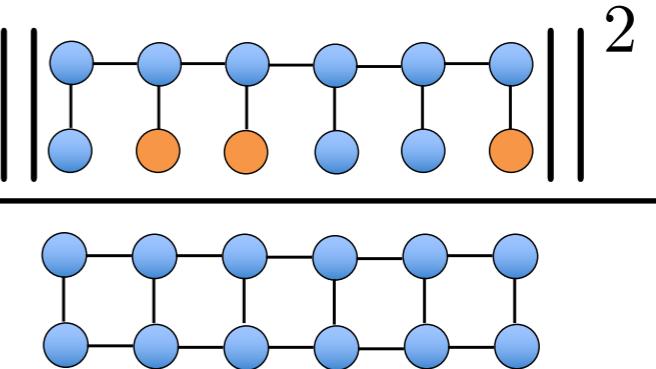
# Make the MPS distribution close to data distribution

$$P(\text{6}) = P(\mathbf{x}) = P(\text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \text{█ }) = \frac{\| \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \|_2^2}{\| \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \|_2^2}$$


A variational distribution has been parametrized using MPS

We need to make the distribution as close as possible to the *data distribution*

# Make the MPS distribution close to data distribution

$$P(\text{6}) = P(\mathbf{x}) = P(\text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \text{█ }) = \frac{\| \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \|_2^2}{\| \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \|_2^2}$$


A variational distribution has been parametrized using MPS

We need to make the distribution as close as possible to the *data distribution*

$$P_{\text{data}}(\mathbf{x}) \propto \sum_{\mathbf{x}^{(i)} \in \text{data}} \delta(\mathbf{x} - \mathbf{x}^{(i)})$$

# Make the MPS distribution close to data distribution

$$P(\text{6}) = P(\mathbf{x}) = P(\text{ } \uparrow \text{ } \uparrow \text{ } \uparrow \text{ } \uparrow \text{ } \uparrow) = \frac{\| \text{ } \uparrow \text{ } \|_2^2}{\| \text{ } \uparrow \text{ } \|_2^2}$$

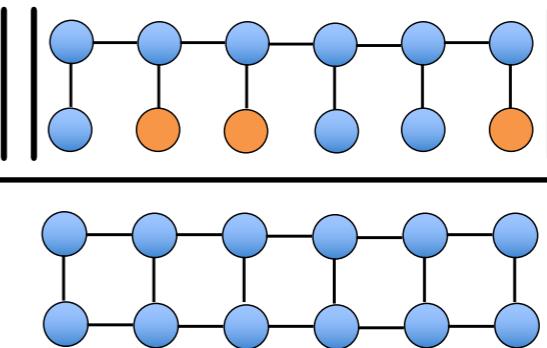
A variational distribution has been parametrized using MPS

We need to make the distribution as close as possible to the *data distribution*

$$P_{\text{data}}(\mathbf{x}) \propto \sum_{\mathbf{x}^{(i)} \in \text{data}} \delta(\mathbf{x} - \mathbf{x}^{(i)})$$

The closeness can be measured by Kullback-Leibler divergence between two distributions

# Make the MPS distribution close to data distribution

$$P(\text{6}) = P(\mathbf{x}) = P(\text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \text{█ }) = \frac{\parallel \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \parallel^2}{\parallel \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \text{█ } \parallel^2}$$


A variational distribution has been parametrized using MPS

We need to make the distribution as close as possible to the *data distribution*

$$P_{\text{data}}(\mathbf{x}) \propto \sum_{\mathbf{x}^{(i)} \in \text{data}} \delta(\mathbf{x} - \mathbf{x}^{(i)})$$

The closeness can be measured by Kullback-Leibler divergence between two distributions

$$D_{\text{KL}}(P_{\text{data}} \| P_{\text{mps}}) = \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{P_{\text{data}}(\mathbf{x})}{P_{\text{mps}}(\mathbf{x})}$$

# Learning algorithm

$$\text{Diagram} = \underset{\text{Diagram}}{\arg \min} D_{\text{KL}}(P_{\text{data}} \| P_{\text{mps}})$$

# Learning algorithm

$$\begin{aligned} \text{Diagram of 6 orange nodes connected sequentially with vertical inputs/outputs} &= \arg \min_{\text{Diagram of 6 blue nodes connected sequentially with vertical inputs/outputs}} D_{\text{KL}}(P_{\text{data}} \| P_{\text{mps}}) \\ &= \arg \min_{\text{Diagram of 6 blue nodes connected sequentially with vertical inputs/outputs}} \mathcal{L}(\text{Diagram of 6 blue nodes}) \quad \text{Negative Log-Likelihood} \end{aligned}$$

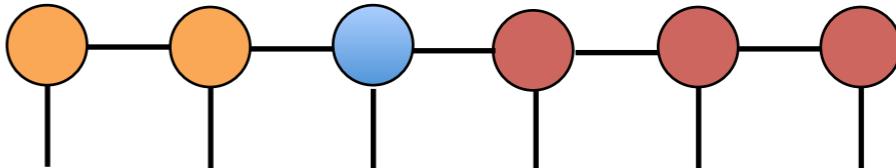
# Learning algorithm

$$\begin{aligned} \text{Diagram of a 1D chain of orange nodes} &= \arg \min_{\text{Diagram of a 1D chain of blue nodes}} D_{\text{KL}}(P_{\text{data}} \| P_{\text{mps}}) \\ &= \arg \min_{\text{Diagram of a 1D chain of blue nodes}} \mathcal{L}(\text{Diagram of a 1D chain of blue nodes}) \quad \text{Negative Log-Likelihood} \\ &= \arg \min_{\text{Diagram of a 1D chain of blue nodes}} \sum_{\substack{\text{Diagram of a 1D chain of blue nodes} \\ \in \text{data}}} \log \frac{\left\| \text{Diagram of a 1D chain of blue nodes} \right\|^2}{\left\| \text{Diagram of a 1D chain of blue nodes} \right\|^2} \end{aligned}$$

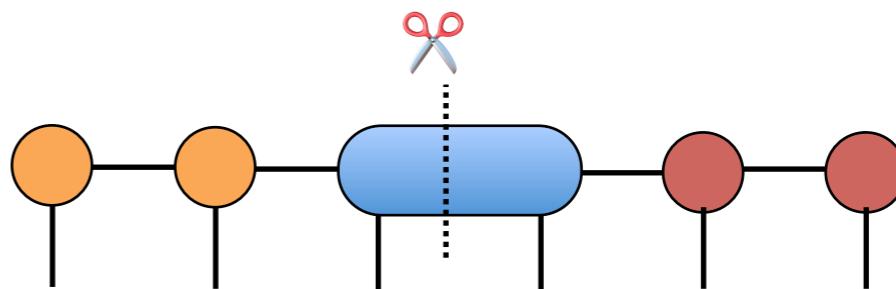
# Learning algorithm

$$\begin{aligned} \text{Diagram of a 1D chain of orange nodes} &= \arg \min_{\text{Diagram of a 1D chain of blue nodes}} D_{\text{KL}}(P_{\text{data}} \| P_{\text{mps}}) \\ &= \arg \min_{\text{Diagram of a 1D chain of blue nodes}} \mathcal{L}(\text{Diagram of a 1D chain of blue nodes}) \quad \text{Negative Log-Likelihood} \\ &= \arg \min_{\text{Diagram of a 1D chain of blue nodes}} \sum_{\substack{\text{Diagram of a 1D chain of blue nodes} \\ \in \text{data}}} \log \frac{\left\| \text{Diagram of a 1D chain of blue nodes} \right\|^2}{\left\| \text{Diagram of a 1D chain of blue nodes} \right\|^2} \end{aligned}$$

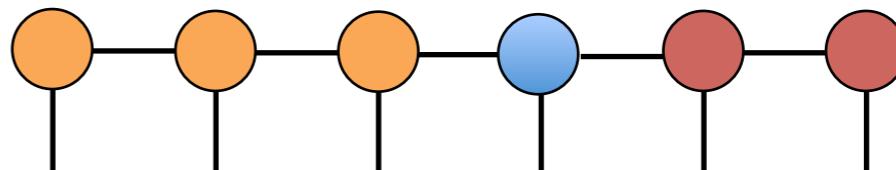
Mixed canonical form



Merging  
+ gradient descent for decreasing NLL



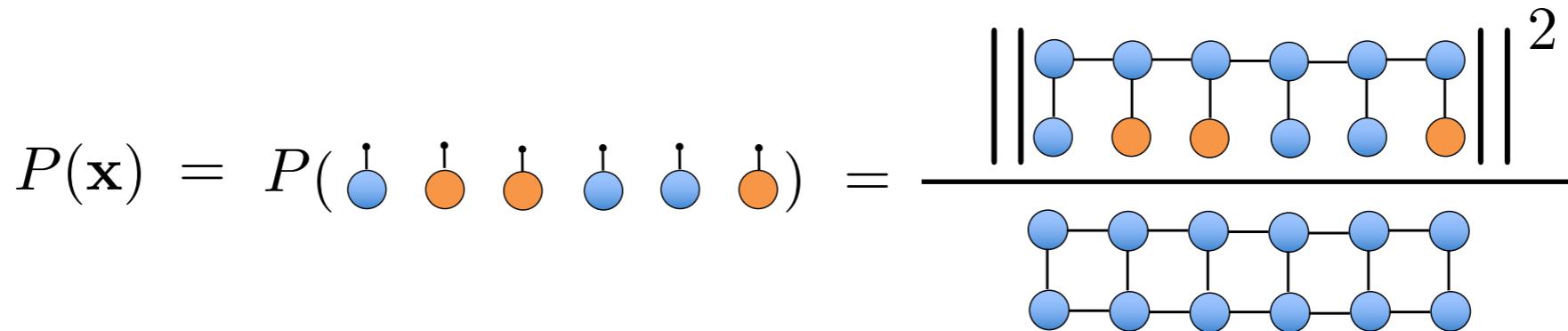
Applying SVD



# Direct sampling

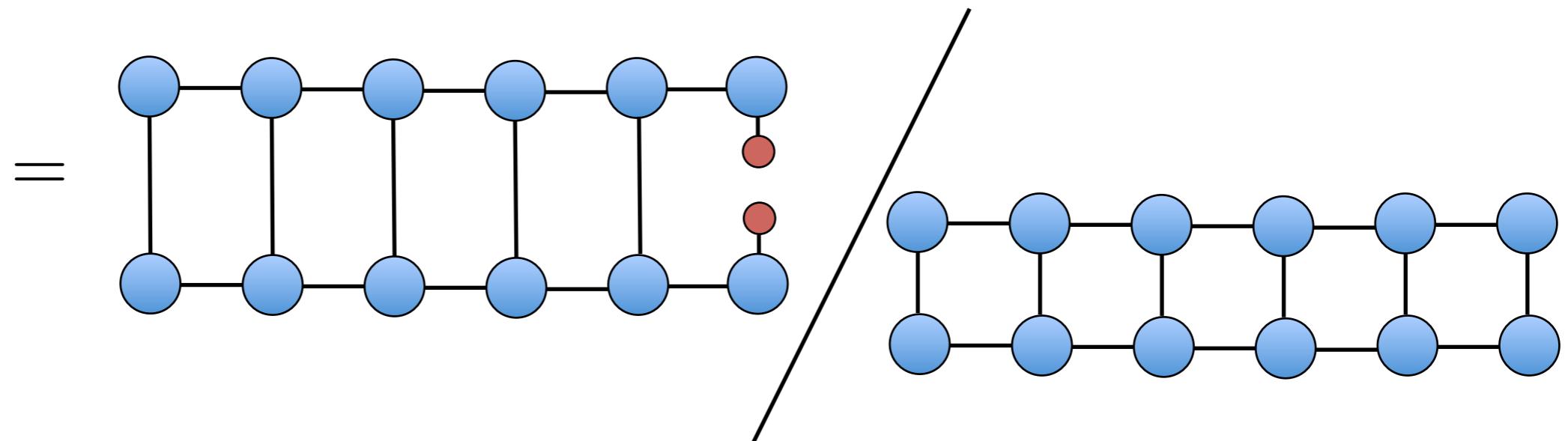
Computing exactly the partition function leads to the ability of computing arbitrary marginals and conditionals.

# Direct sampling

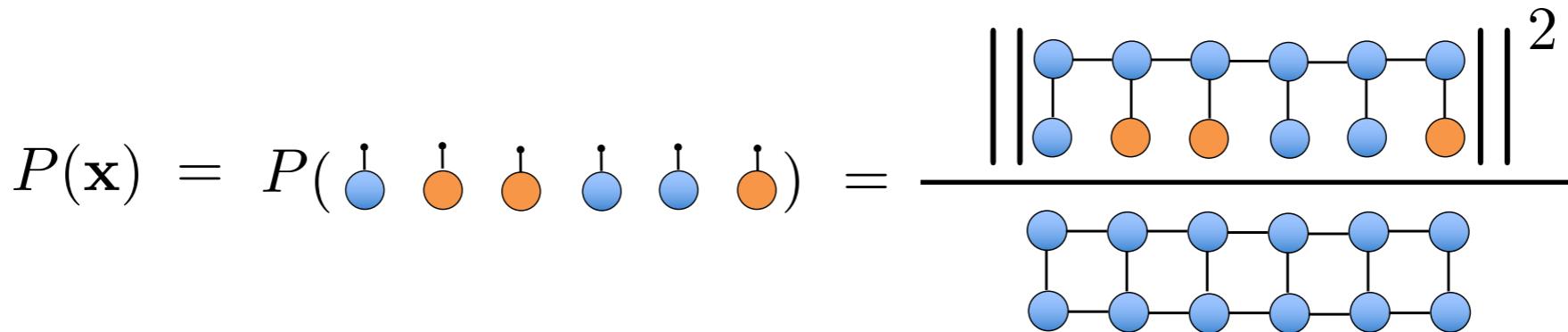
$$P(\mathbf{x}) = P(\text{---}) = \frac{\text{---}}{\text{---}}^2$$


Computing exactly the partition function leads to the ability of computing *arbitrary marginals and conditionals.*

$$P(x_1) = \sum_{x_2} \sum_{x_3} \cdots \sum_{x_n} P(\mathbf{x})$$



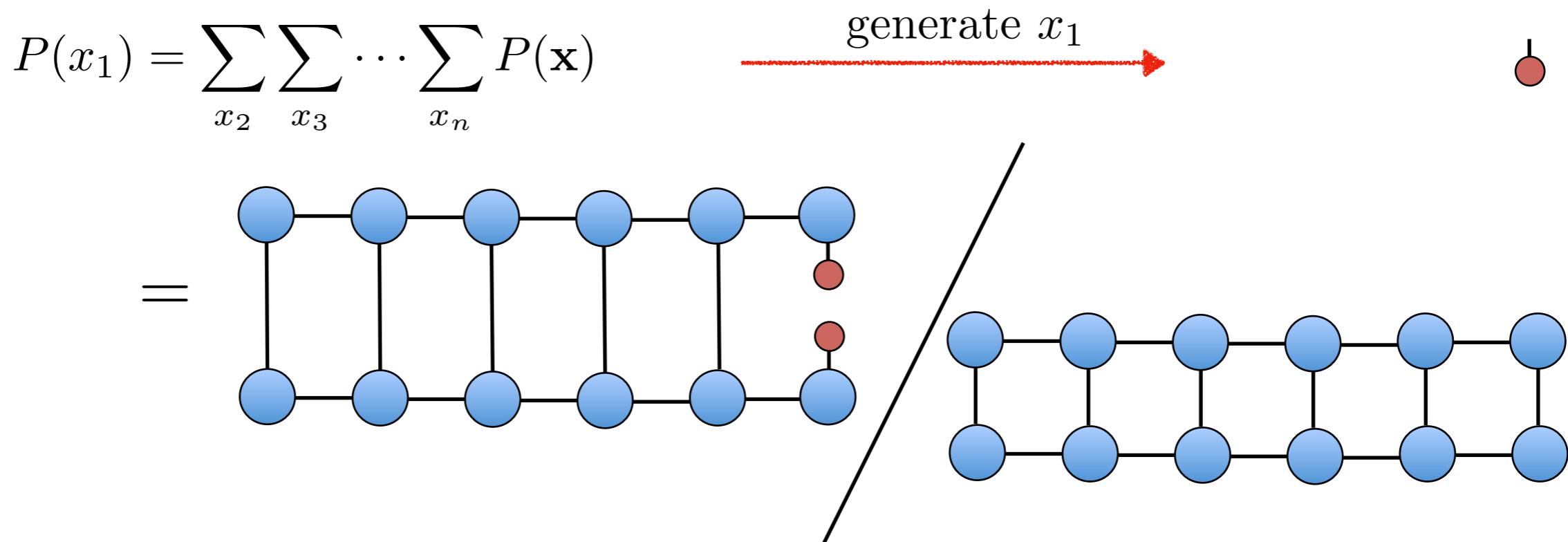
# Direct sampling

$$P(\mathbf{x}) = P(\text{---}) = \frac{\text{---}}{\text{---}}^2$$


Computing exactly the partition function leads to the ability of computing *arbitrary marginals and conditionals.*

$$P(x_1) = \sum_{x_2} \sum_{x_3} \cdots \sum_{x_n} P(\mathbf{x})$$

generate  $x_1$  →

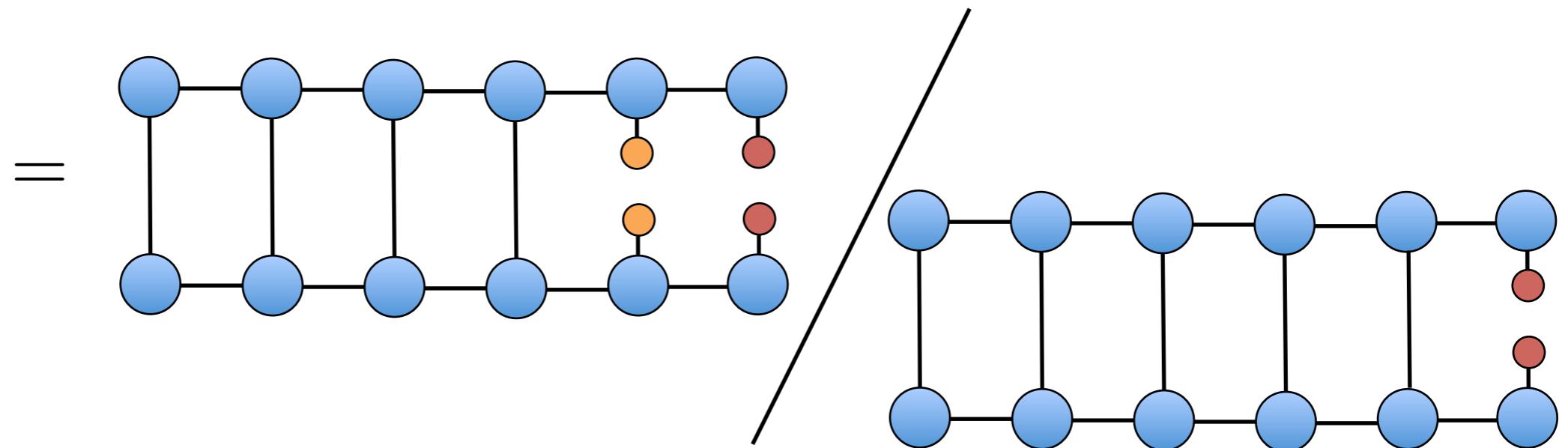


# Direct sampling

$$P(\mathbf{x}) = P(\text{graph}) = \frac{\text{graph}_1^2 - \text{graph}_2^2}{\text{graph}_1 + \text{graph}_2}$$

Computing exactly the partition function leads to the ability of compute *arbitrary marginals and conditionals*.

$$P(x_2|x_1) = P(x_1, x_2)/p(x_1)$$



# Direct sampling

Computing exactly the partition function leads to the ability of compute arbitrary marginals and conditionals.

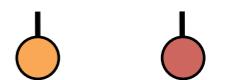
The diagram illustrates the generation of  $x_2$  from a sequence of hidden states. On the left, the formula  $P(x_2|x_1) = P(x_1, x_2)/p(x_1)$  is shown. A red arrow labeled "generate  $x_2$ " points from the formula to a sequence of hidden states. The sequence consists of two parallel rows of blue circles connected by vertical lines. The top row has six blue circles, and the bottom row has five blue circles. Between the two rows, there are two additional circles: an orange circle above the fifth blue circle and a red circle below the fifth blue circle. To the right of the sequence, a diagonal line connects the bottom row to a final row of seven blue circles, representing the generated sequence.

# Direct sampling

$$P(\mathbf{x}) = P(\text{graph}) = \frac{\text{graph}_1^2}{\text{graph}_2^2}$$

Computing exactly the partition function leads to the ability of compute arbitrary marginals and conditionals.

$$P(x_3|x_1, x_2) = P(x_1, x_2, x_3)/p(x_1, x_2)$$



# Direct sampling

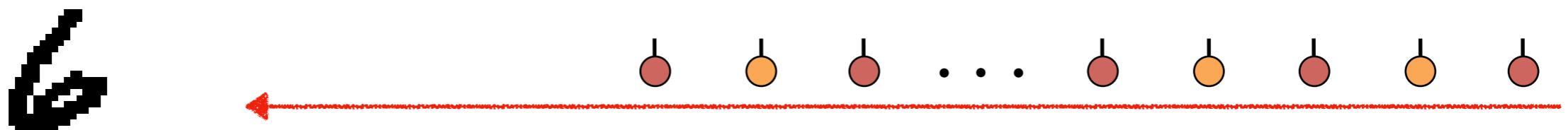
Computing exactly the partition function leads to the ability of compute arbitrary marginals and conditionals.

The diagram illustrates the generation of  $x_3$  from a sequence of hidden states. It shows two rows of nodes connected by horizontal edges. The top row consists of six blue nodes. The bottom row consists of five blue nodes. Three red nodes and one orange node are attached to the bottom row nodes at positions 3, 5, and 6. A red arrow labeled "generate  $x_3$ " points from the top row to the bottom row. Below the diagram, an equals sign indicates that the top row represents the hidden state sequence and the bottom row represents the observed sequence.

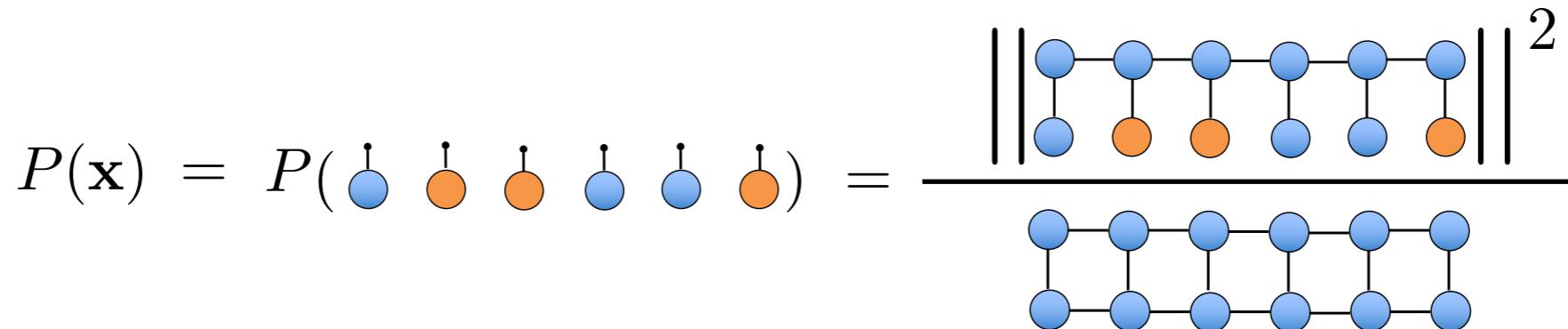
# Direct sampling

$$P(\mathbf{x}) = P(\text{graph}) = \frac{\text{graph}_1^2 - \text{graph}_2^2}{\text{graph}_1 + \text{graph}_2}$$

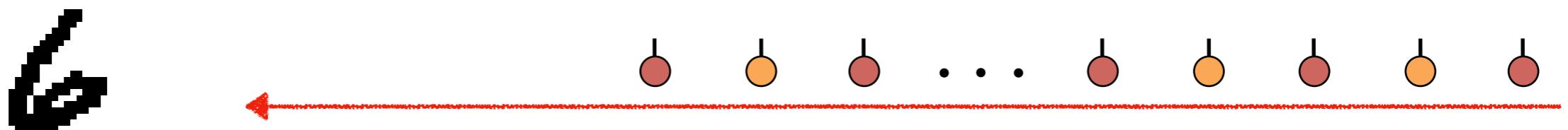
Computing exactly the partition function leads to the ability of compute arbitrary marginals and conditionals.



# Direct sampling

$$P(\mathbf{x}) = P(\text{---}) = \frac{\parallel \text{---} \parallel^2}{\parallel \text{---} \parallel^2}$$


Computing exactly the partition function leads to the ability of compute *arbitrary marginals and conditionals.*

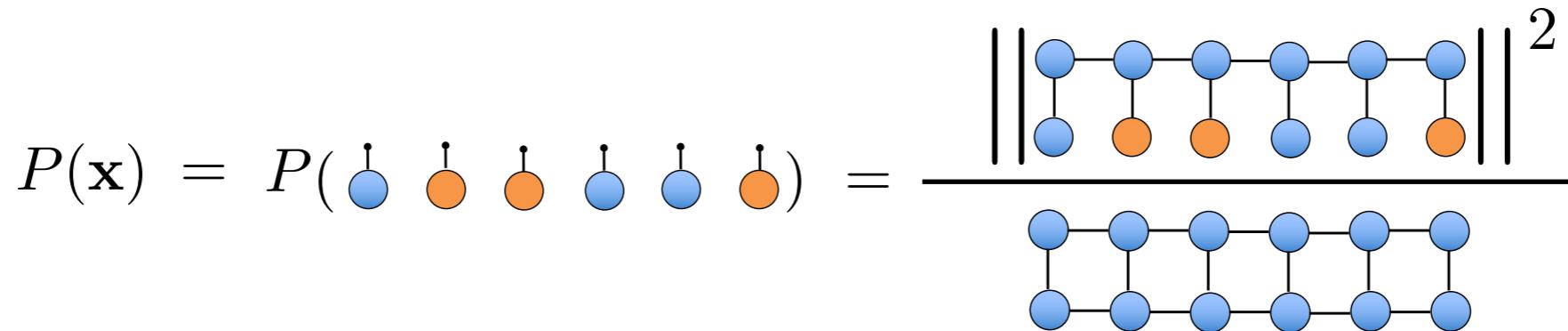


known as *ancestral sampling* in machine learning and  
known as “*perfect sampling with unitary tensor networks*” in physics.

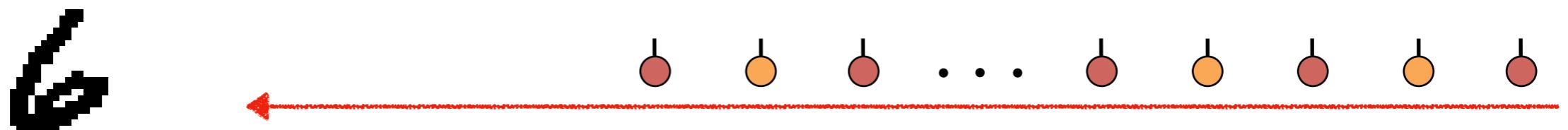
Bishop, “Machine Learning and Pattern Recognition” 2006

Ferris, Vidal, Phys. Rev. B 85, 165146 (2012)

# Direct sampling

$$P(\mathbf{x}) = P(\text{---}) = \frac{\parallel \text{---} \parallel^2}{\parallel \text{---} \parallel^2}$$


Computing exactly the partition function leads to the ability of compute *arbitrary marginals and conditionals.*



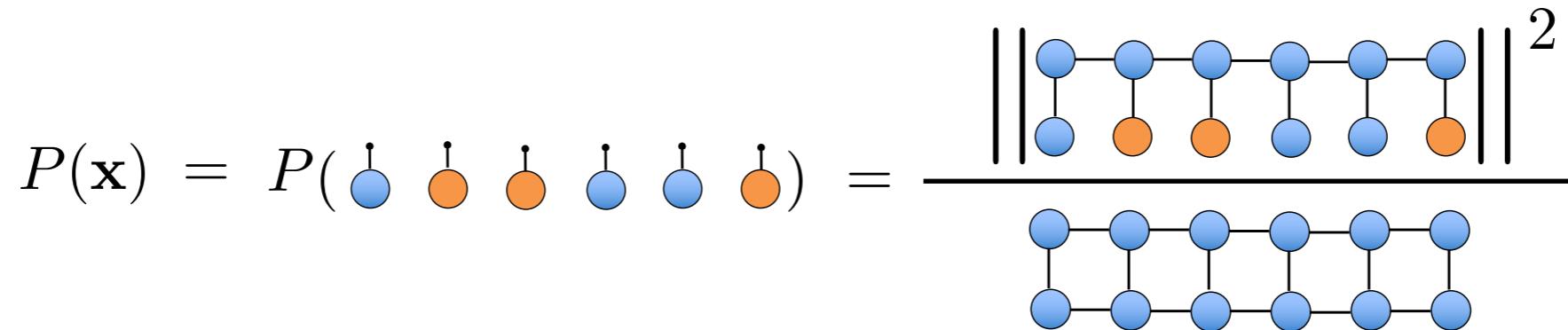
known as *ancestral sampling* in machine learning and  
known as “*perfect sampling with unitary tensor networks*” in physics.

It is a *unbiased/perfect* sampling, without introducing any autocorrelation

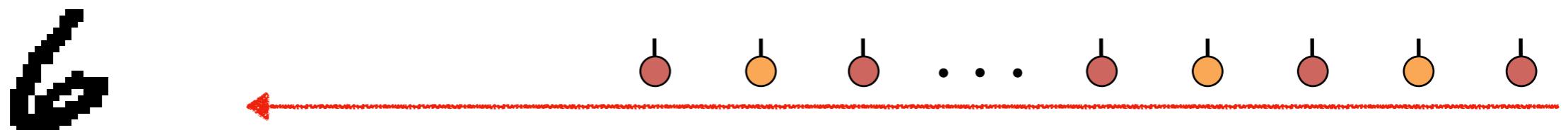
Bishop, “Machine Learning and Pattern Recognition” 2006

Ferris, Vidal, Phys. Rev. B 85, 165146 (2012)

# Direct sampling

$$P(\mathbf{x}) = P(\text{---}) = \frac{\| \text{---} \|_2^2}{\| \text{---} \|_2^2}$$


Computing exactly the partition function leads to the ability of compute *arbitrary marginals and conditionals.*



known as *ancestral sampling* in machine learning and known as “*perfect sampling with unitary tensor networks*” in physics.

It is a *unbiased/perfect* sampling, without introducing any autocorrelation

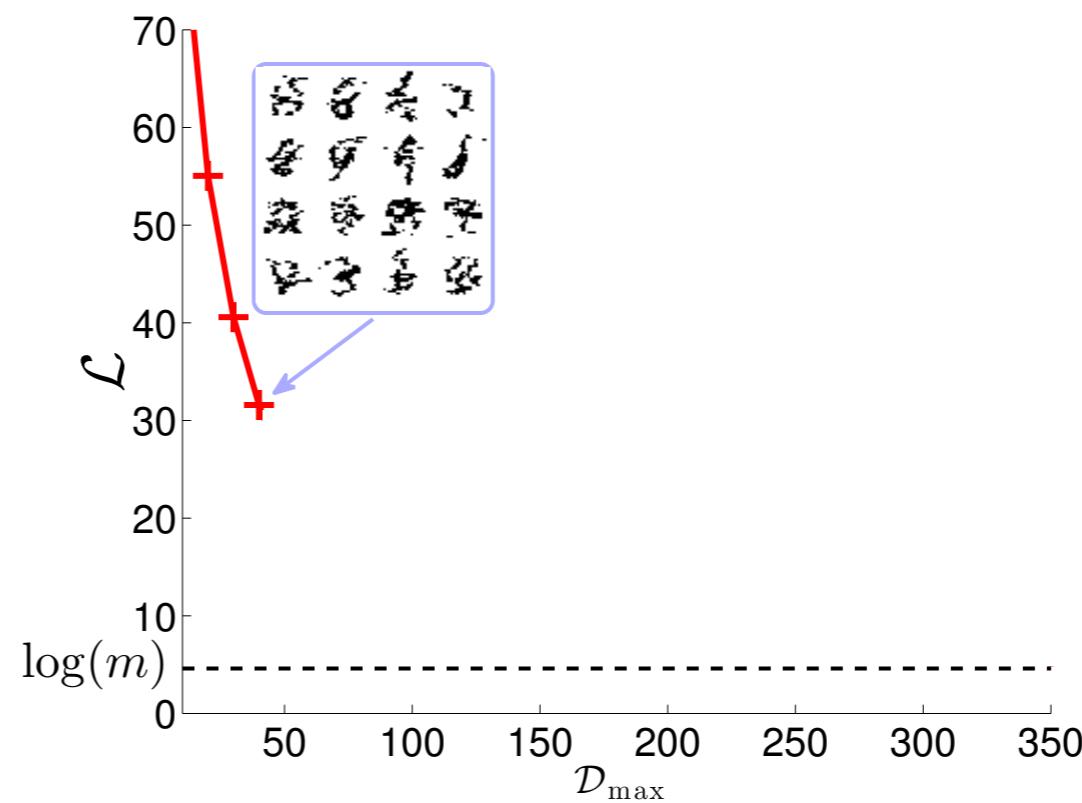
Readily parallelizable

Bishop, “Machine Learning and Pattern Recognition” 2006

Ferris, Vidal, Phys. Rev. B 85, 165146 (2012)

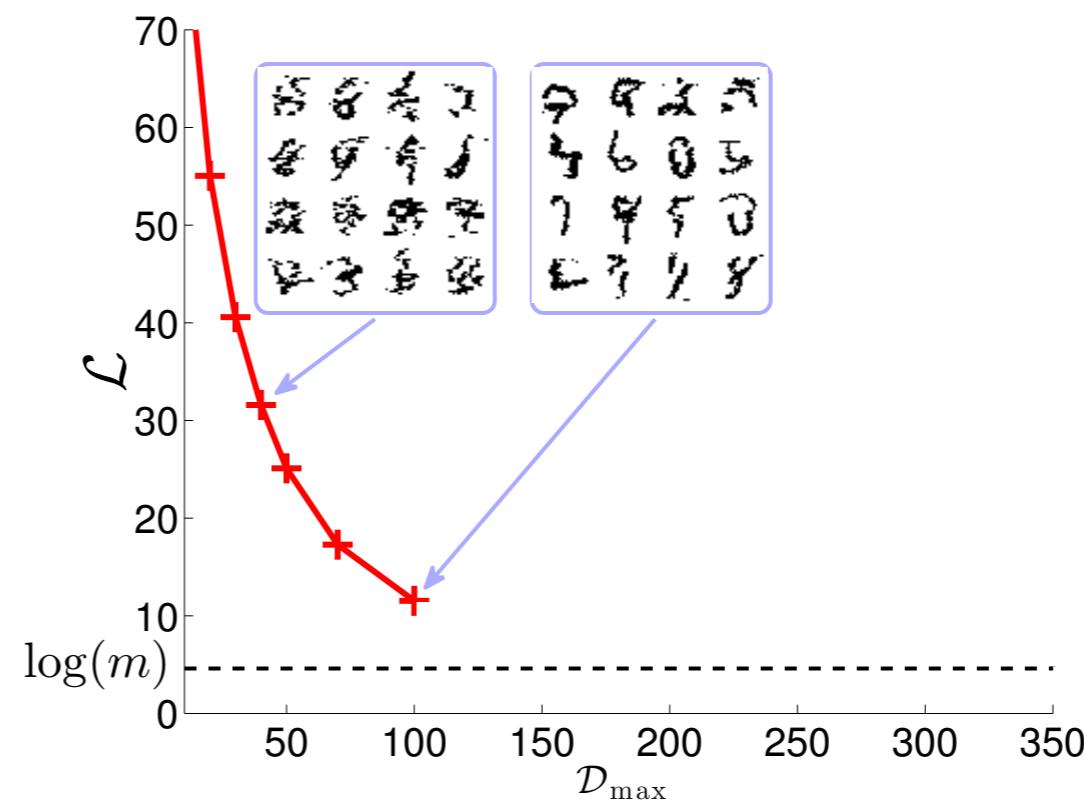
# On the MNIST dataset

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9



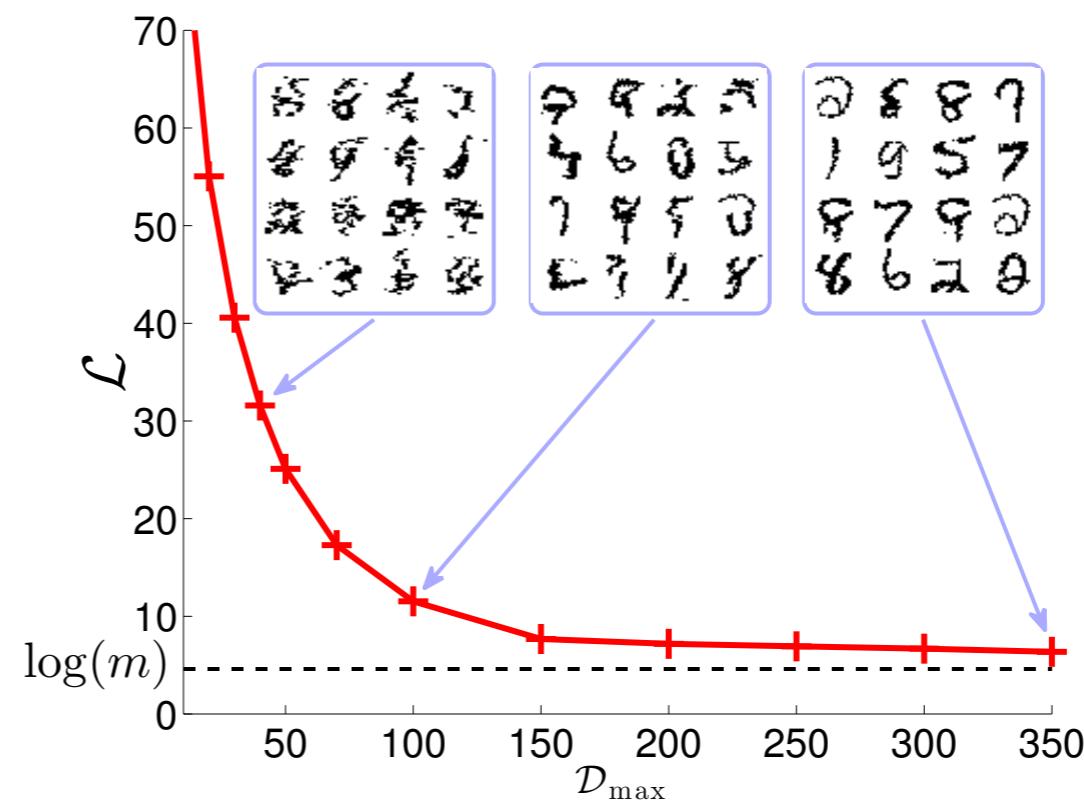
# On the MNIST dataset

9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9



# On the MNIST dataset

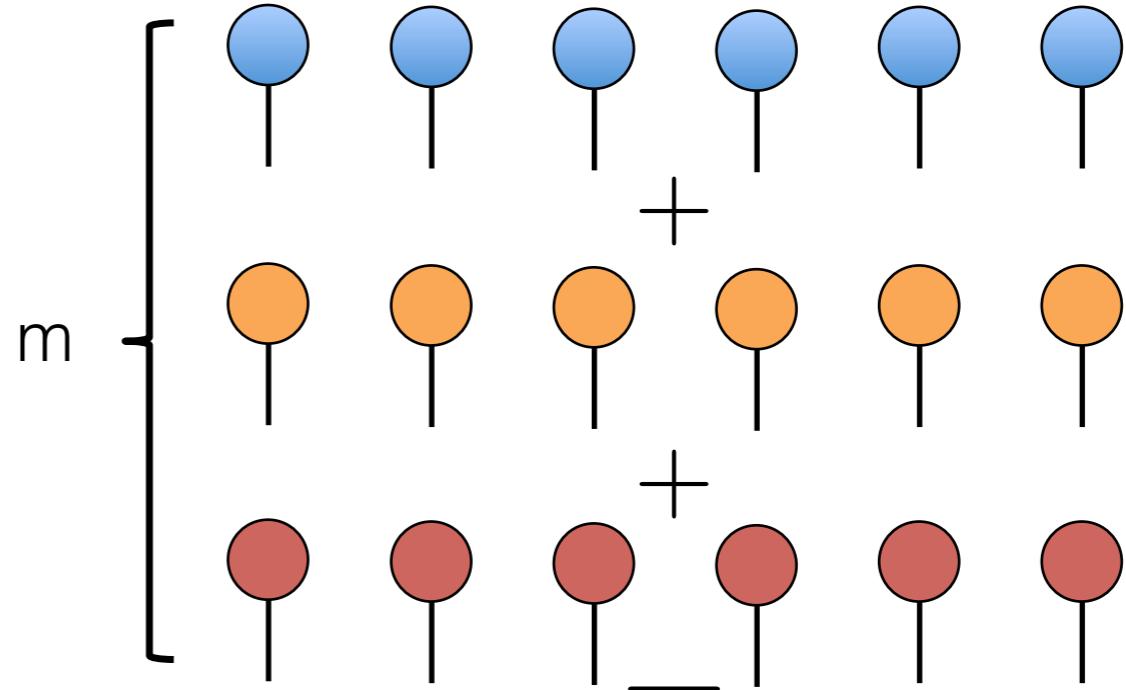
9 3 6 6 5 7  
5 3 9 4 5 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9



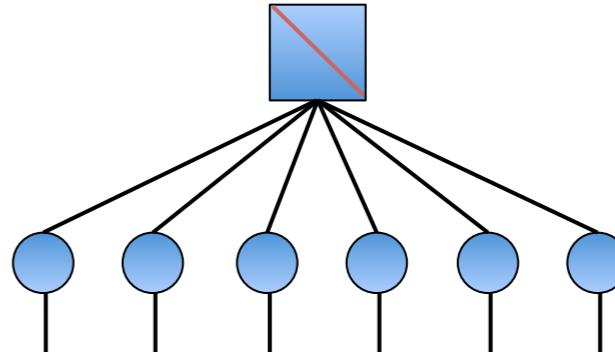
# Exact memorizing patterns

# Exact memorizing patterns

6  
4  
2  
  
m images

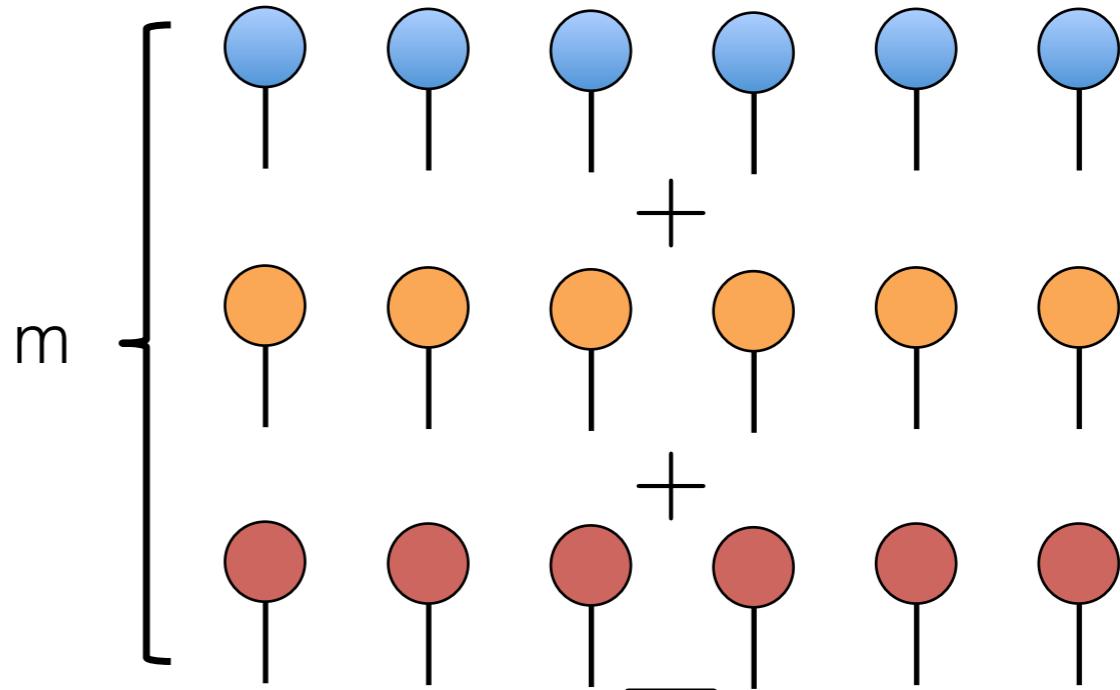


Exact learning of  $m$  images amounts to construct a CP tensor with rank  $m$ .



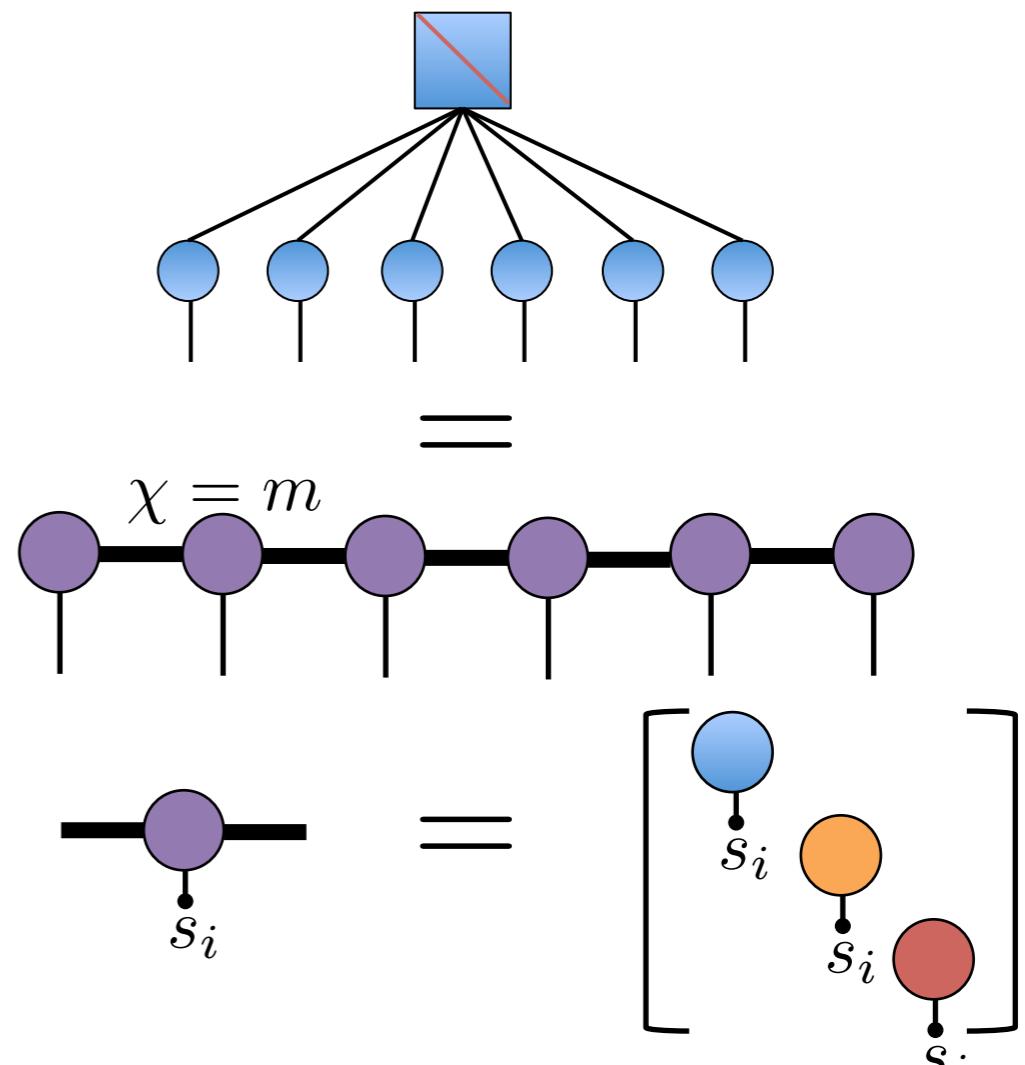
# Exact memorizing patterns

6  
4  
2  
  
m images



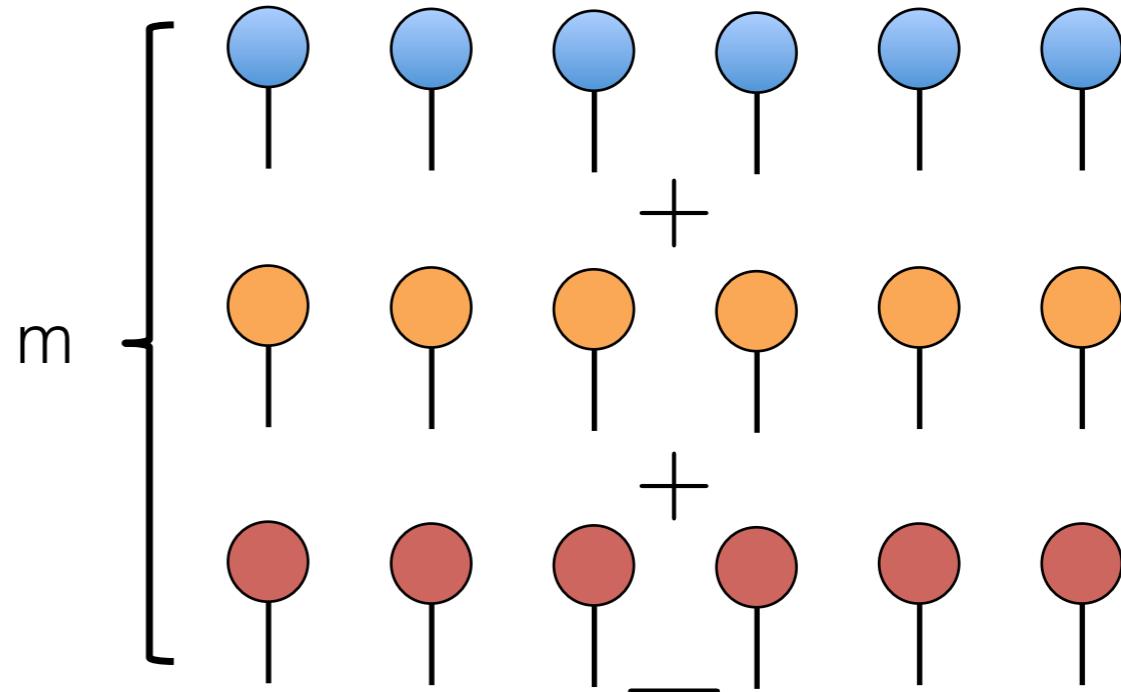
Exact learning of  $m$  images amounts to construct a CP tensor with rank  $m$ .

This can be converted to a MPS with bond dimension  $m$ .



# Exact memorizing patterns

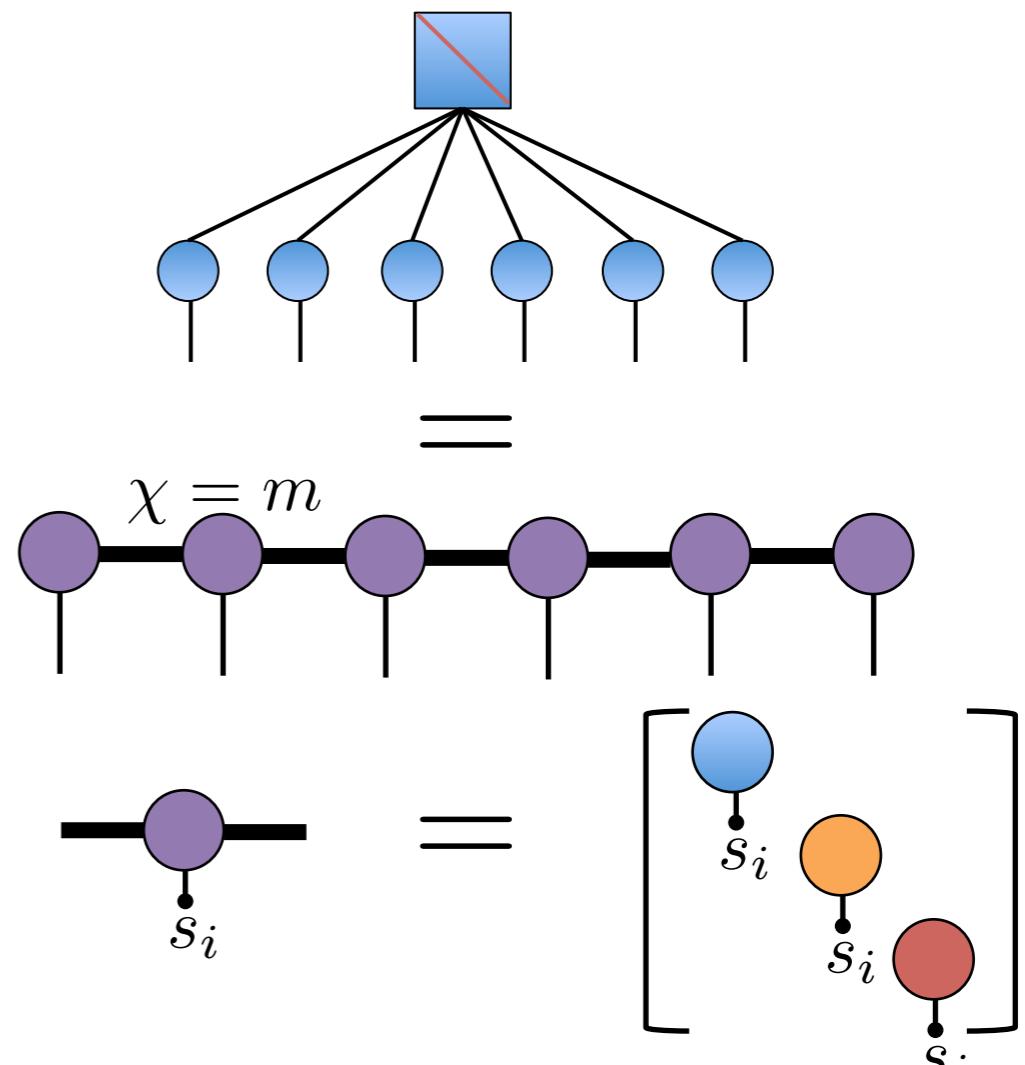
6  
4  
2  
  
m images



Exact learning of  $m$  images amounts to construct a CP tensor with rank  $m$ .

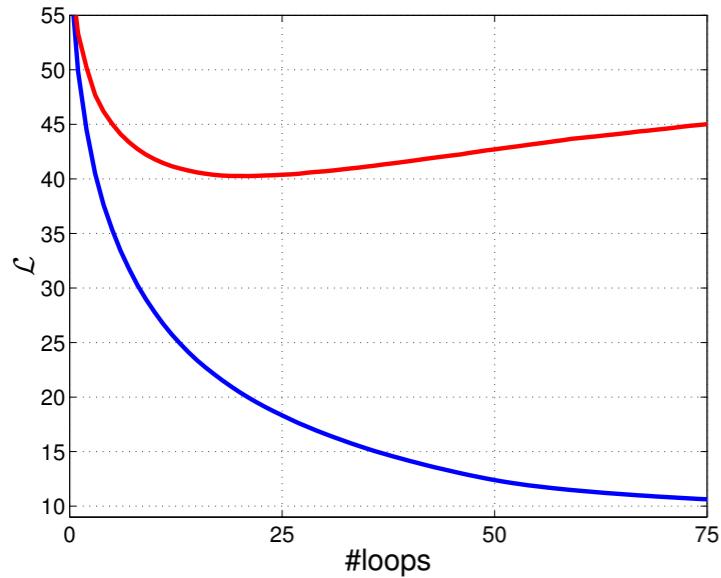
This can be converted to a MPS with bond dimension  $m$ .

With exact learning, NLL of the MPS equals to the lower bound  $\log(m)$ .



# Generalization to unseen images

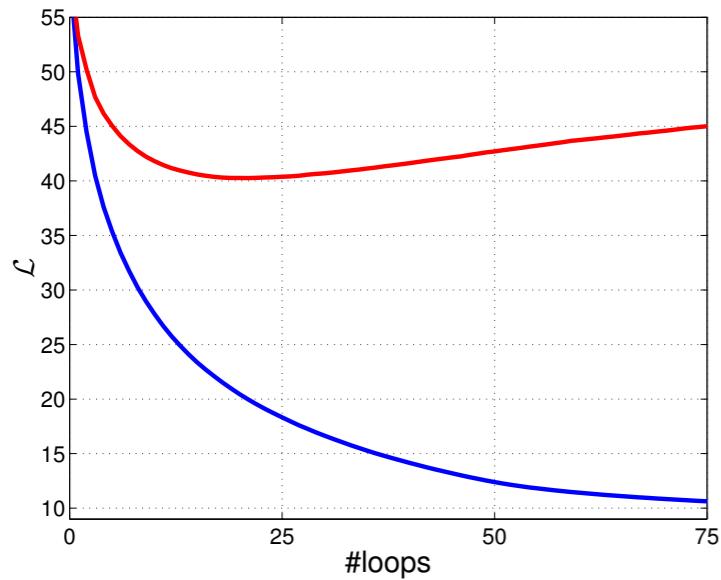
However, exact memorizing will give zero probability to all images except those in the training set —— *no generalization power!*



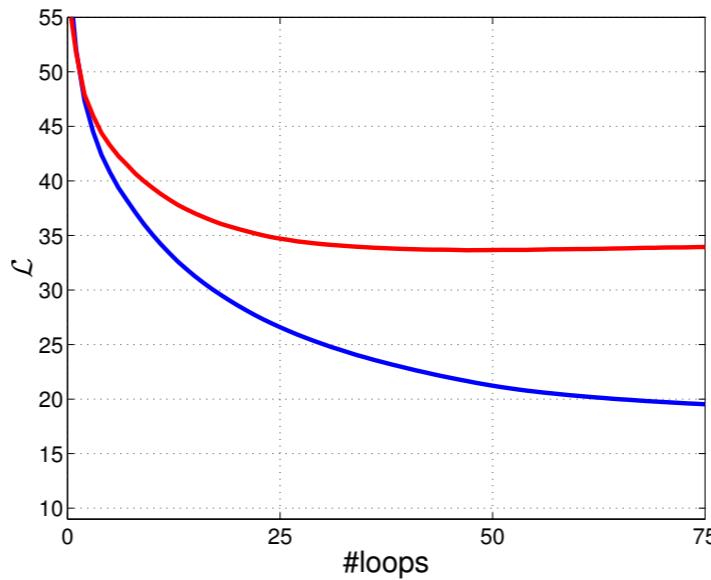
1000 training images

# Generalization to unseen images

However, exact memorizing will give zero probability to all images except those in the training set —— *no generalization power!*



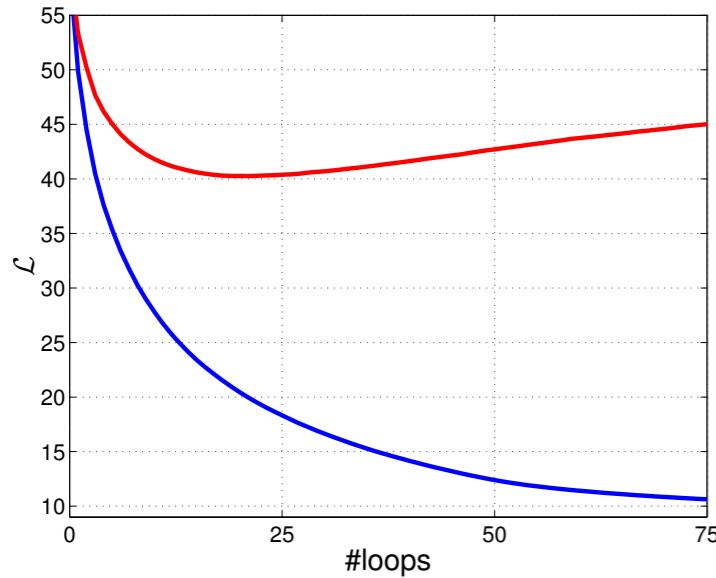
1000 training images



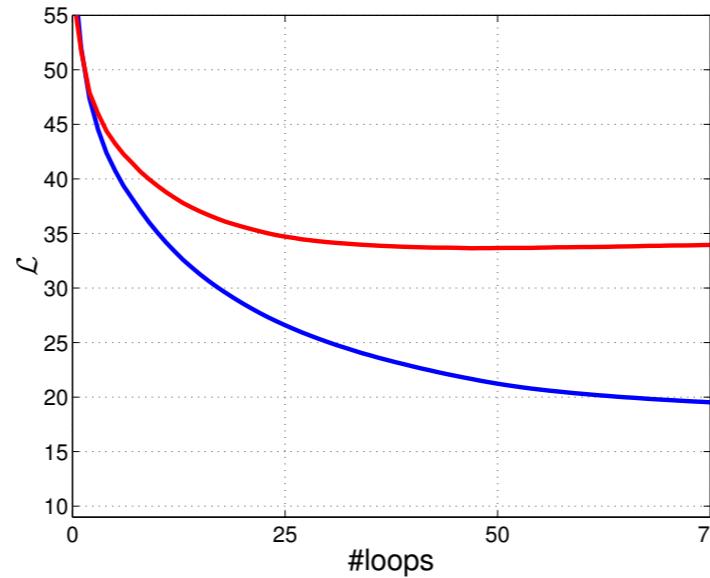
10000 training images

# Generalization to unseen images

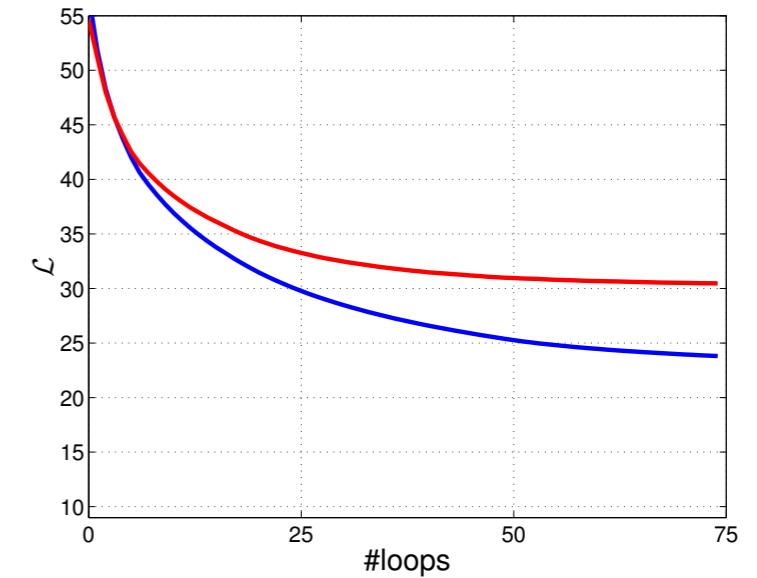
However, exact memorizing will give zero probability to all images except those in the training set —— *no generalization power!*



1000 training images



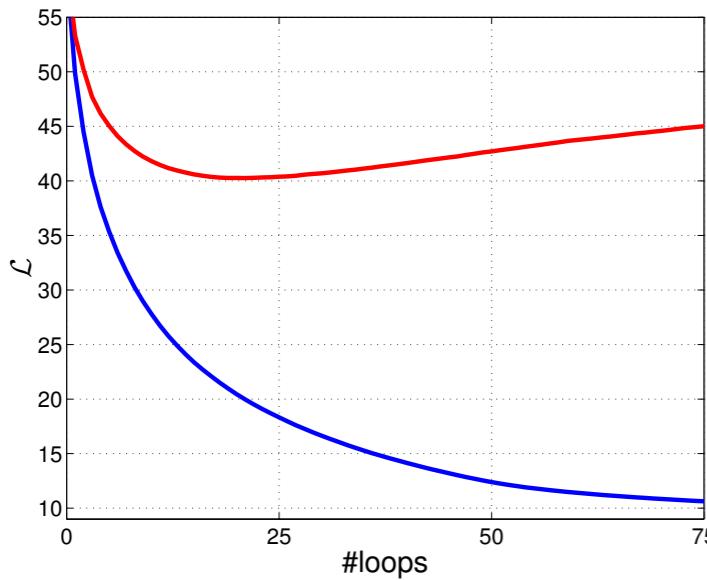
10000 training images



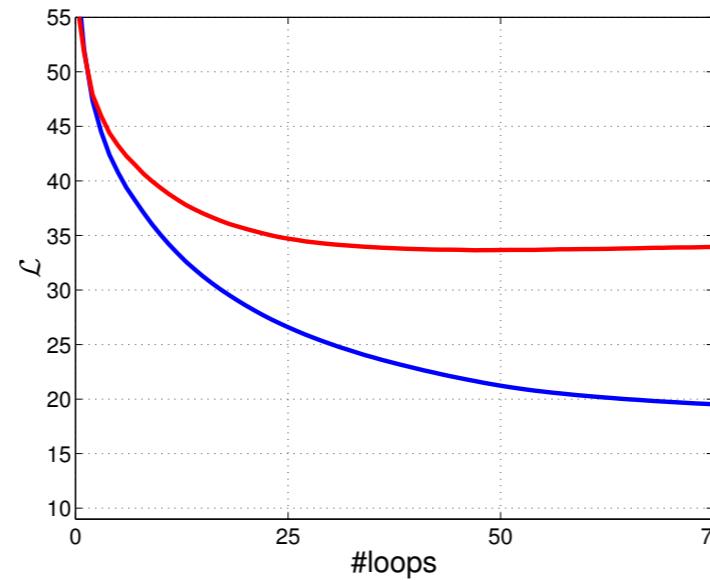
60000 training images

# Generalization to unseen images

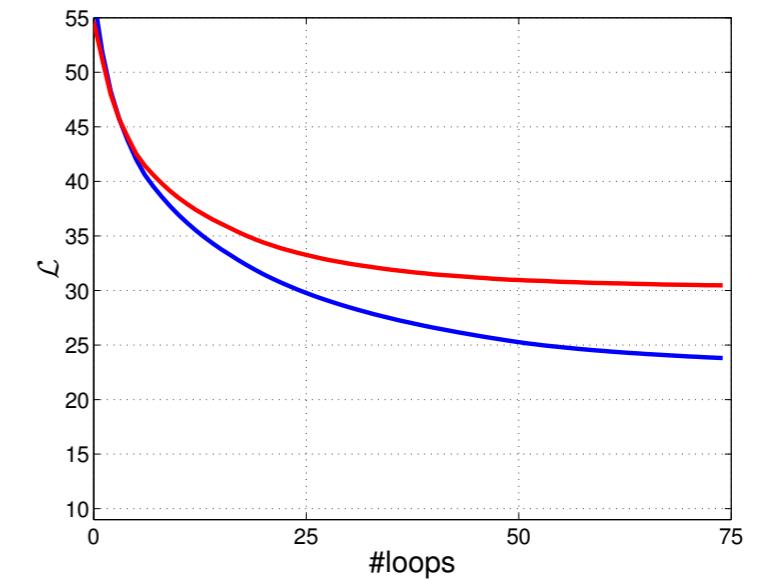
However, exact memorizing will give zero probability to all images except those in the training set —— *no generalization power!*



1000 training images



10000 training images



60000 training images

Generalization is the most important ability in machine learning.  
Increasing generalization ability requires more data, better data, and  
better *inductive bias*.

# Image reconstruction

6 5 9 ,  
0 7 2 3  
4 7 1 8  
2 3 6 ,

8 3 2 ,  
0 1 2 5  
6 1 2 7  
0 2 4 1

# Image reconstruction

8 3 9 7  
0 1 3 5  
6 1 2 0  
2 2 4 1

8 3 3 7  
0 1 3 5  
6 1 2 0  
2 2 4 1

# MPS Born machine

# MPS Born machine

- Features:

# MPS Born machine

- Features:
  - Trackable likelihood via efficient tensor contractions.

$$Z = \begin{array}{ccccccccc} \text{orange} & \text{orange} & \text{orange} & \text{blue} & \text{red} & \text{red} \\ \text{orange} & \text{orange} & \text{orange} & \text{blue} & \text{red} & \text{red} \end{array} = \boxed{\text{blue}}$$

# MPS Born machine

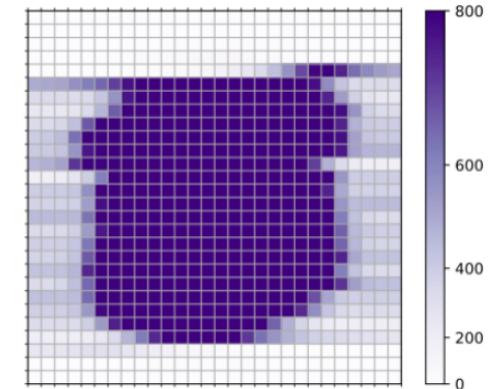
- Features:
  - Trackable likelihood via efficient tensor contractions.
  - Adaptively allocating bond dimension.

$$Z = \begin{array}{ccccccccc} \text{orange} & & \text{orange} & & \text{orange} & & \text{blue} & & \text{red} \\ | & & | & & | & & | & & | \\ \text{orange} & & \text{orange} & & \text{orange} & & \text{blue} & & \text{red} \end{array} = \boxed{\text{blue}}$$

Training images

4	1	9	2	1	3
3	5	3	6	1	7
6	9	4	0	9	1
4	3	2	7	3	8
0	5	6	0	7	6
1	9	3	9	8	5

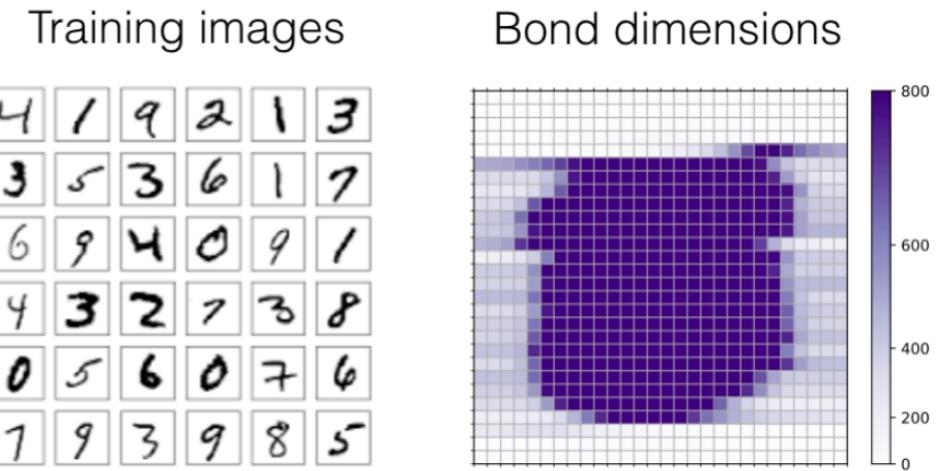
Bond dimensions



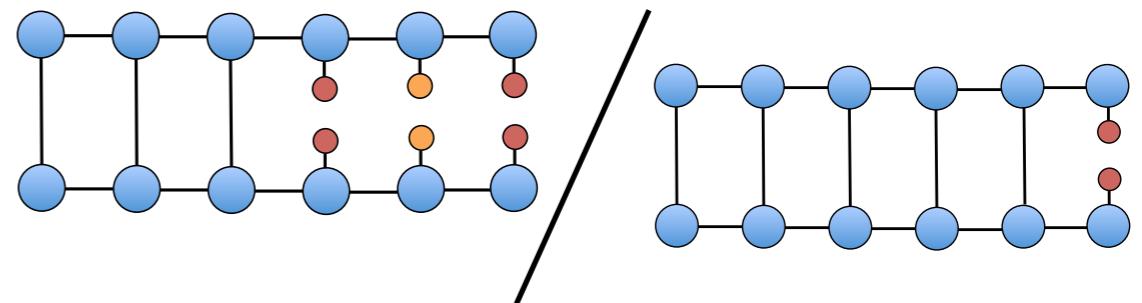
# MPS Born machine

- Features:
  - Trackable likelihood via efficient tensor contractions.
  - Adaptively allocating bond dimension.
  - Fast direct sampling without autocorrelation.

$$Z = \begin{array}{ccccccccc} \textcolor{orange}{\circ} & \textcolor{orange}{\circ} & \textcolor{orange}{\circ} & \textcolor{blue}{\circ} & \textcolor{red}{\circ} & \textcolor{red}{\circ} \\ | & | & | & | & | & | \\ \textcolor{orange}{\circ} & \textcolor{orange}{\circ} & \textcolor{orange}{\circ} & \textcolor{blue}{\circ} & \textcolor{red}{\circ} & \textcolor{red}{\circ} \end{array} = \boxed{\textcolor{blue}{\circ}} \quad \boxed{\textcolor{blue}{\circ}}$$



$$p(\mathbf{x}) = \prod_i \frac{p(\mathbf{x}_{<i+1})}{p(\mathbf{x}_{$$



# MPS Born machine

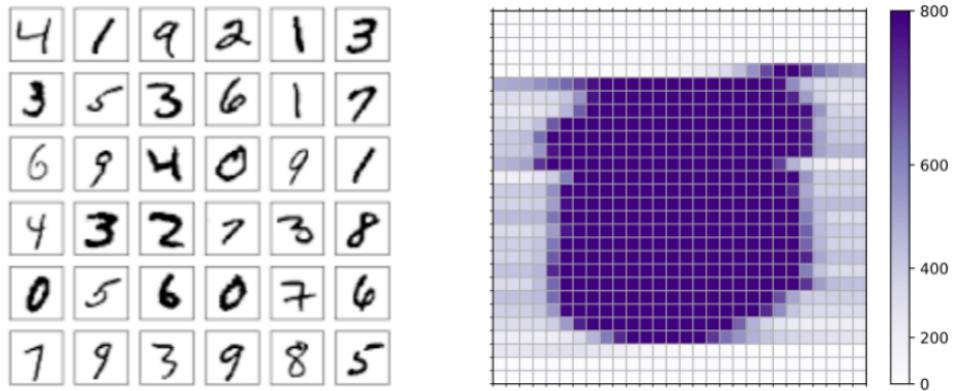
- Features:
  - Trackable likelihood via efficient tensor contractions.
  - Adaptively allocating bond dimension.
  - Fast direct sampling without autocorrelation.

$$Z = \begin{array}{ccccccccc} \textcolor{orange}{\circ} & \textcolor{orange}{\circ} & \textcolor{orange}{\circ} & \textcolor{blue}{\circ} & \textcolor{red}{\circ} & \textcolor{red}{\circ} \\ | & | & | & | & | & | \\ \textcolor{orange}{\circ} & \textcolor{orange}{\circ} & \textcolor{orange}{\circ} & \textcolor{blue}{\circ} & \textcolor{red}{\circ} & \textcolor{red}{\circ} \end{array} = \boxed{\textcolor{blue}{\circ}} \quad \boxed{\textcolor{blue}{\circ}}$$

Training images

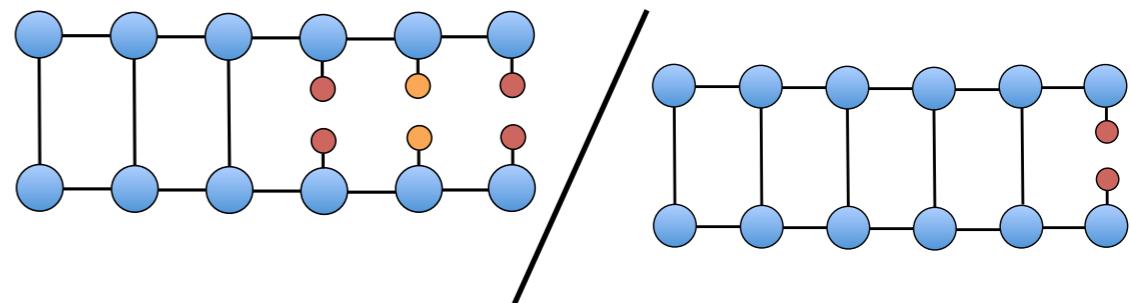
4	1	9	2	1	3
3	5	3	6	1	7
6	9	4	0	9	1
4	3	2	7	3	8
0	5	6	0	7	6
1	9	3	9	8	5

Bond dimensions



- Limitations:

$$p(\mathbf{x}) = \prod_i \frac{p(\mathbf{x}_{<i+1})}{p(\mathbf{x}_{$$



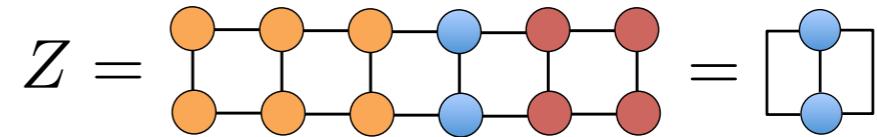
# MPS Born machine

- Features:

- Trackable likelihood via efficient tensor contractions.
- Adaptively allocating bond dimension.
- Fast direct sampling without autocorrelation.

- Limitations:

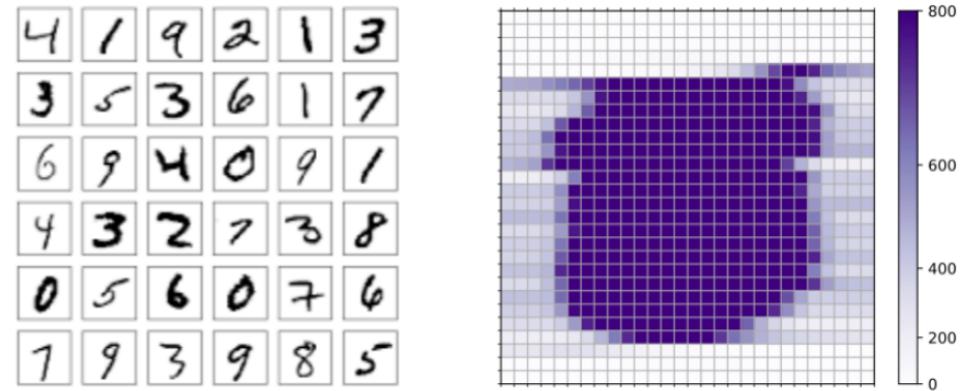
- Fast decay of correlations



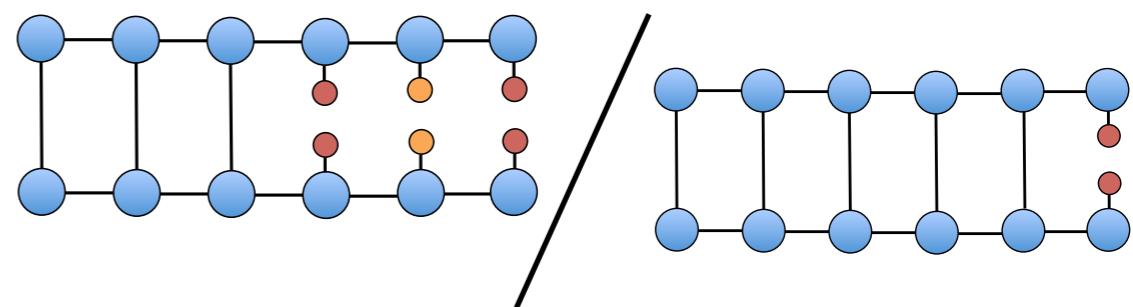
Training images

4	1	9	2	1	3
3	5	3	6	1	7
6	9	4	0	9	1
4	3	2	7	3	8
0	5	6	0	7	6
1	9	3	9	8	5

Bond dimensions



$$p(\mathbf{x}) = \prod_i \frac{p(\mathbf{x}_{<i+1})}{p(\mathbf{x}_{$$



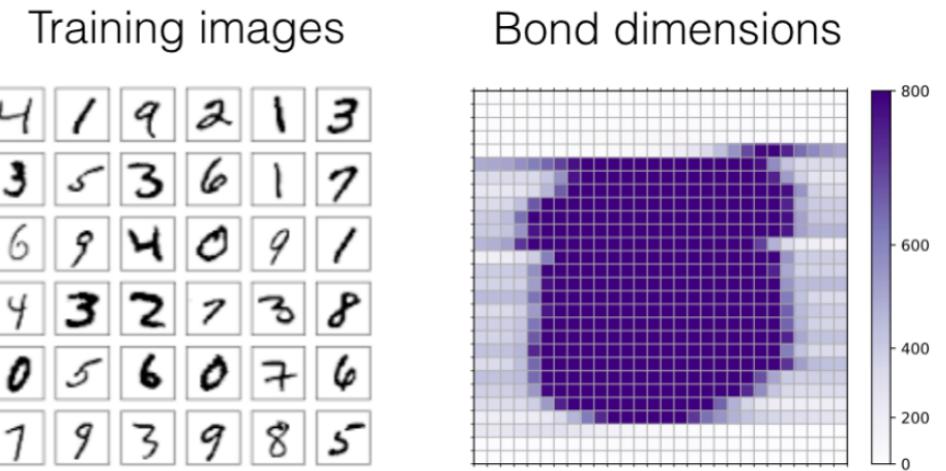
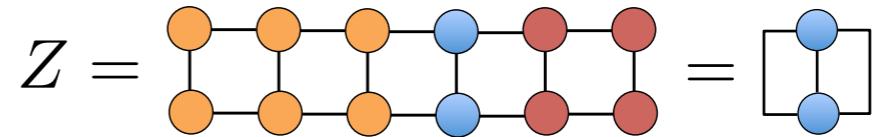
# MPS Born machine

- Features:

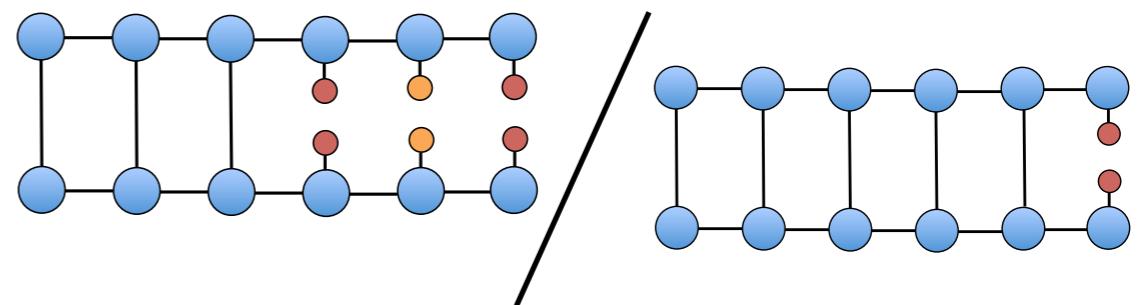
- Trackable likelihood via efficient tensor contractions.
- Adaptively allocating bond dimension.
- Fast direct sampling without autocorrelation.

- Limitations:

- Fast decay of correlations
- Bad prior for 2-D images

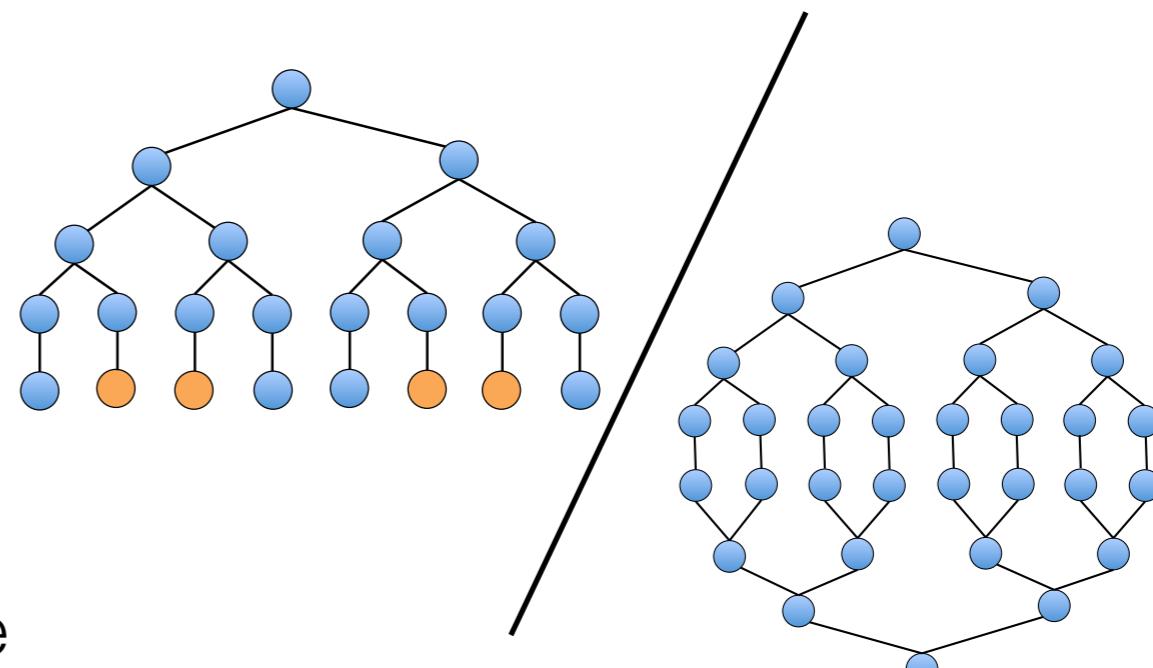


$$p(\mathbf{x}) = \prod_i \frac{p(\mathbf{x}_{<i+1})}{p(\mathbf{x}_{$$



# Tree Tensor Network Born machine

$$P(\mathbf{x}) = P(\text{blue circle}, \text{orange circle}, \text{orange circle}, \text{blue circle}, \text{blue circle}, \text{orange circle}) =$$

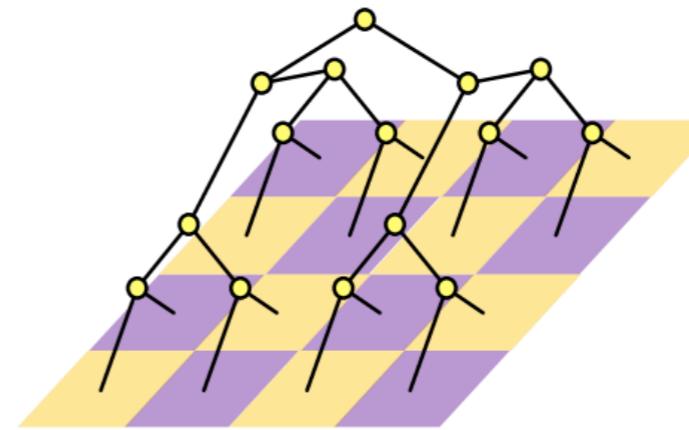


- Features:

- Analogous to MPS: trackable likelihood, direct sampling, and canonical forms.
  - Good prior for 2-D images
  - Better correlation length (in practice)

- Limitations:

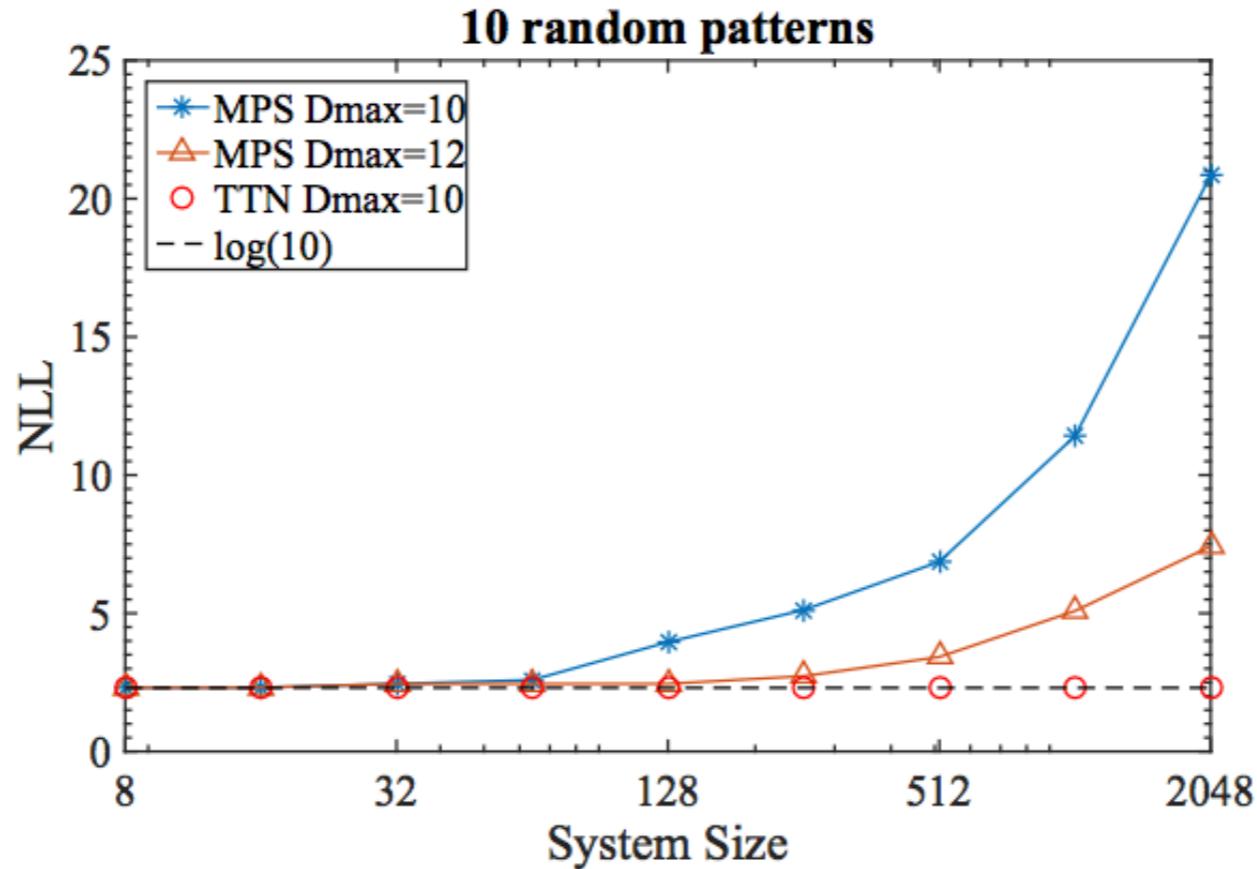
- Higher computational complexity than MPS Born machine



(a)

1	2	3	4	1	3	9	11
5	6	7	8	2	4	10	12
9	10	11	12	5	7	13	15
13	14	15	16	6	8	14	16

# Better correlation length in practice



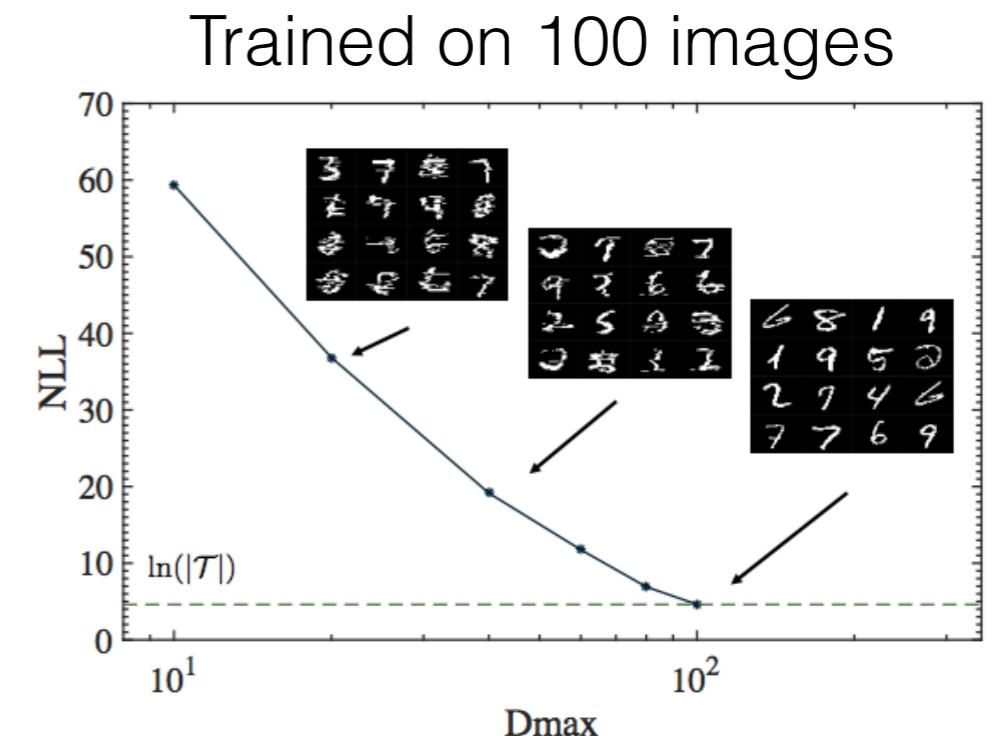
Random patterns are good benchmarks for testing the *capacity*

Theoretical limits for both MPS and TTN born machines are  $\ln(m)$

However, in practice it is much easier to train TTN to saturate the threshold.

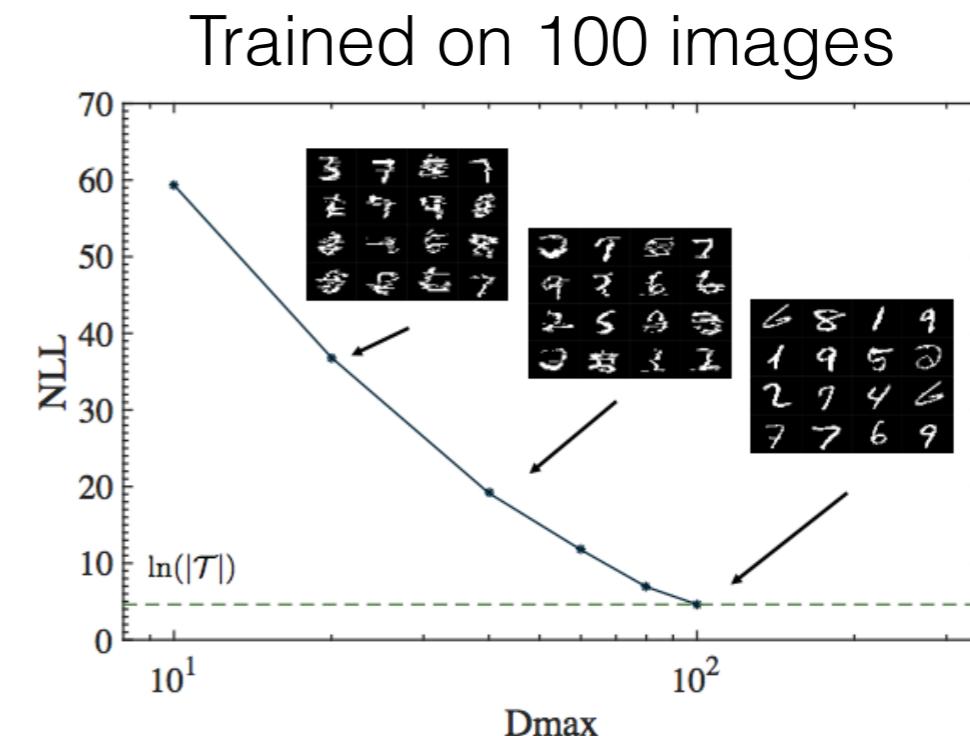
# On the MNIST dataset

9 3 6 6 5 7  
5 3 9 4 4 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9



# On the MNIST dataset

9 3 6 6 5 7  
5 3 9 4 4 7  
5 4 1 2 6 0  
7 4 6 2 2 2  
2 9 8 9 3 9  
2 0 6 7 1 9



Model	Test NLL
Tree factor graph	175.8
MPS	101.5
TTN-1d	96.9
TTN-2d	94.3
RBM	86.3* <sup>41</sup>
VAE	84.8* <sup>43</sup>
PixelCNN	81.3 <sup>10</sup>

\* stands for approximated NLL.

# Perspectives

Deeper exploration of benefits of the huge *Hilbert feature space*.

Imposing better internal structures (MERA, PEPS) for inductive bias in learning relevant features of data.

MPS/MPO is potentially very powerful for sequence data.

Relation between neural network models (e.g. convolution nets, autoregressive models) and TNs.

Understanding theoretical limits of learning, making use of linearity of TNs?

Neural network architectures inspired by tensor networks (e.g. MERA...)