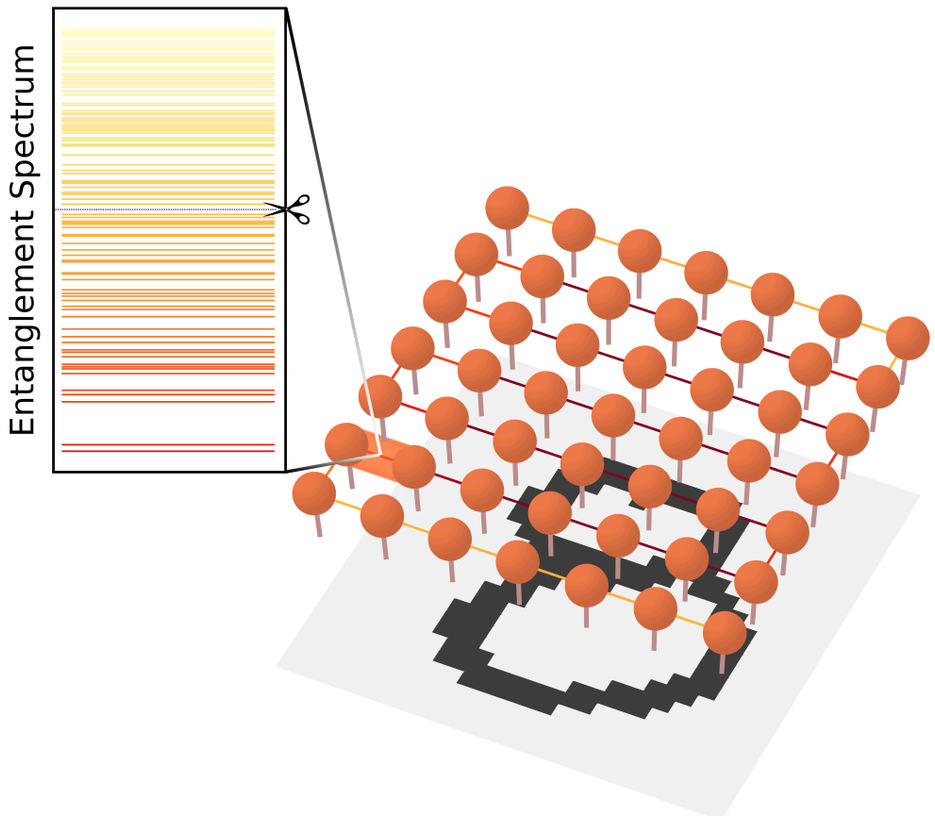
Tensor networks

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

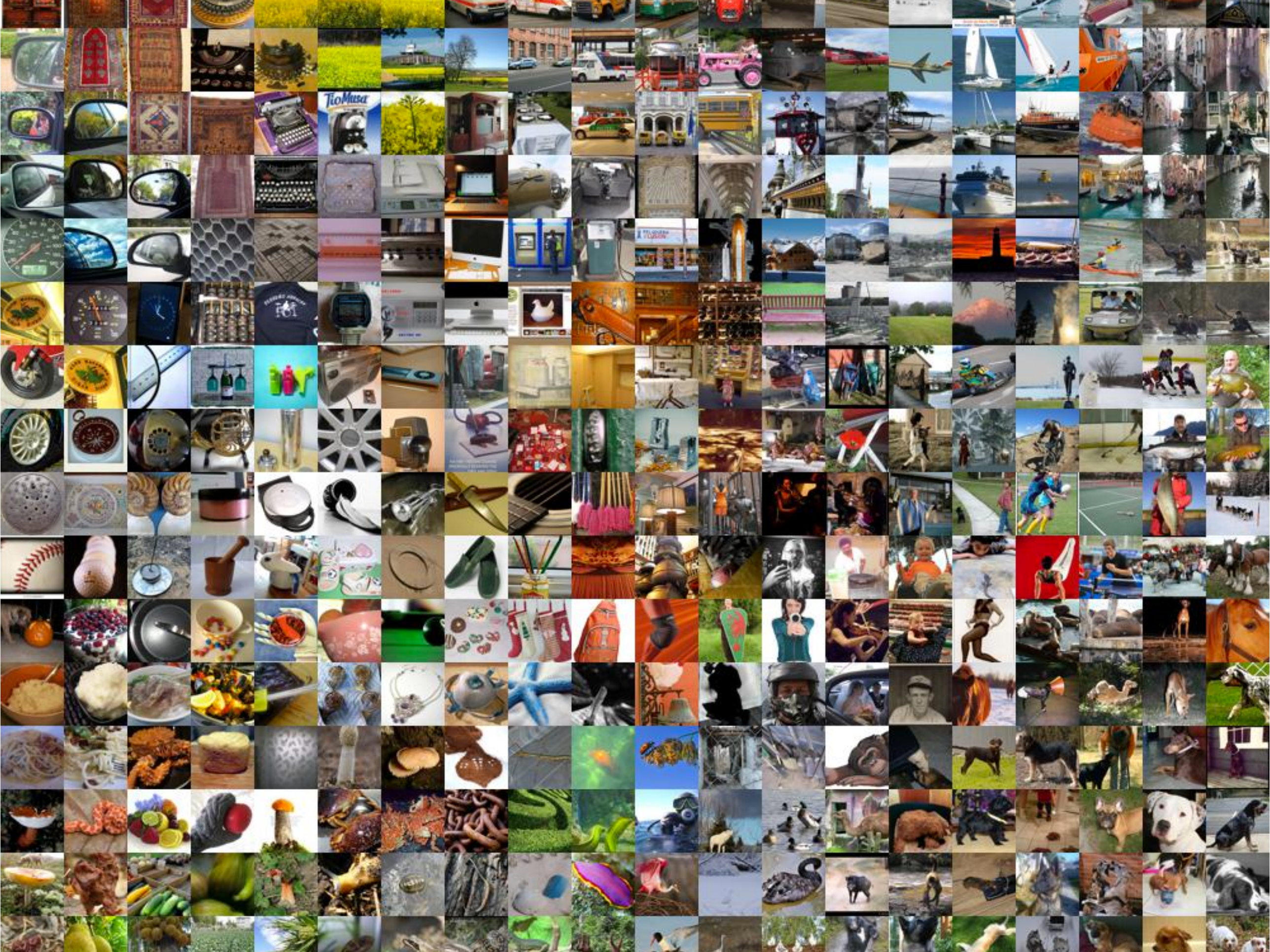
➡ Tensor networks for representations
(in a small space)

- 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)

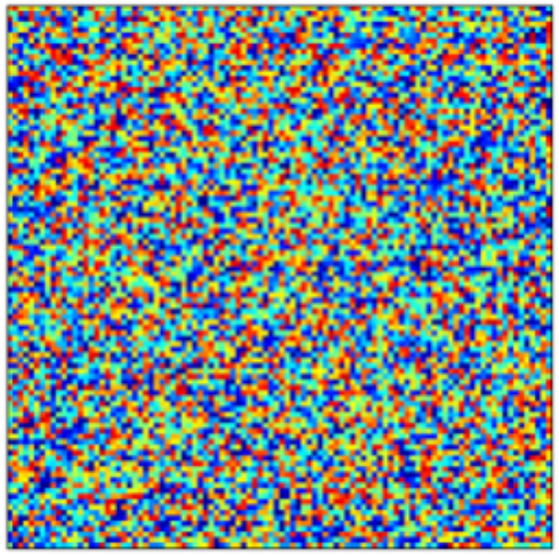
Low_rank(data)

P(label|data)

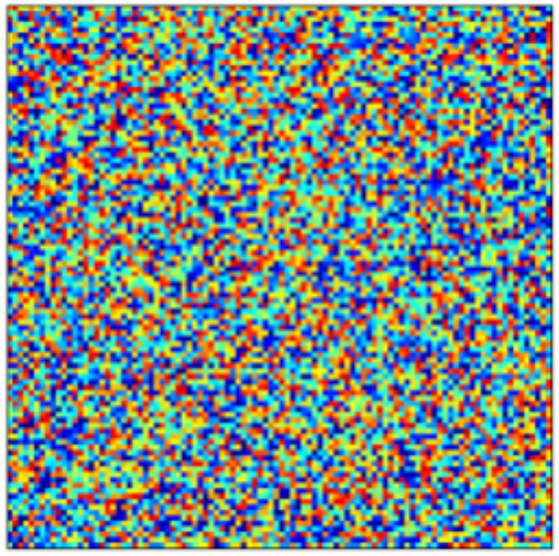
P(data)

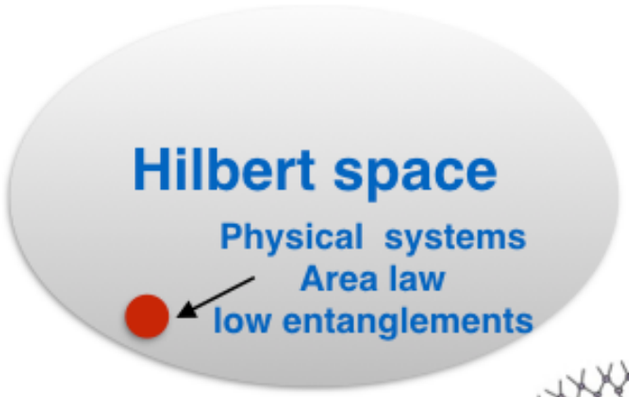
Analogous to quantum physics

Analogous to quantum physics



Analogous to quantum physics





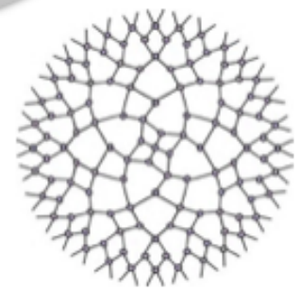
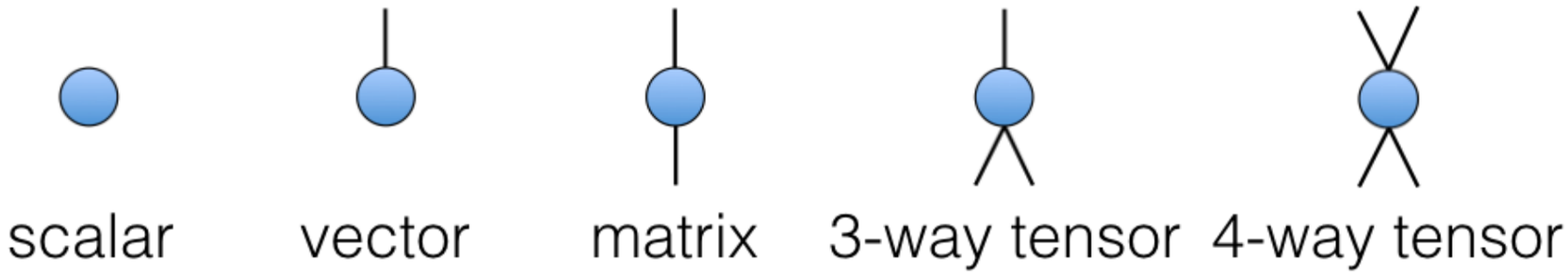
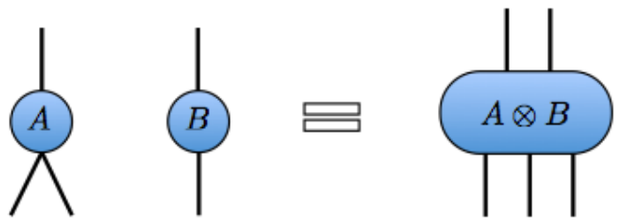
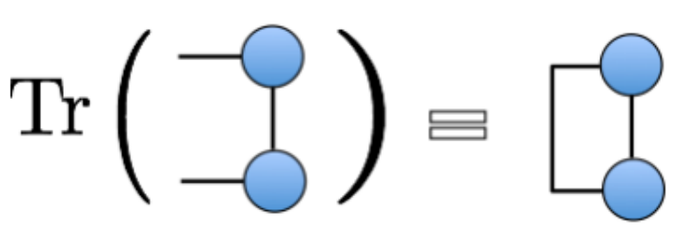


Diagram notations

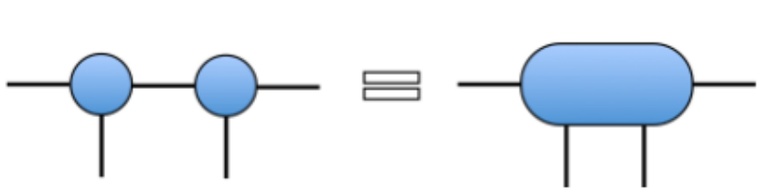




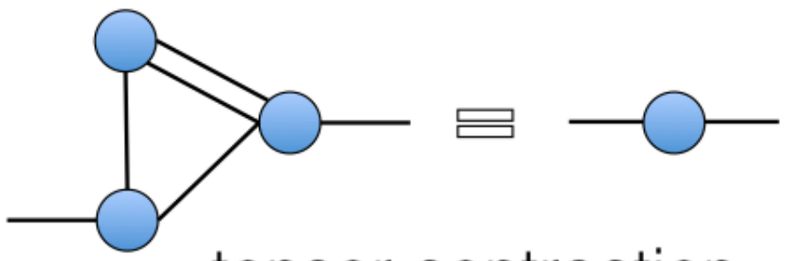
tensor product



Trace

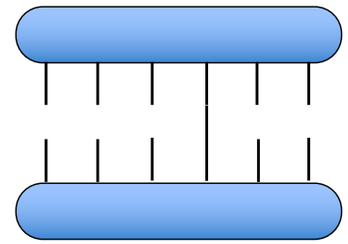
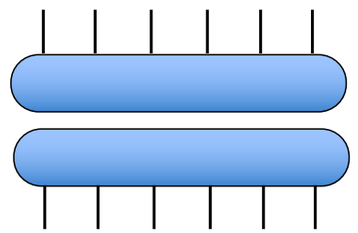
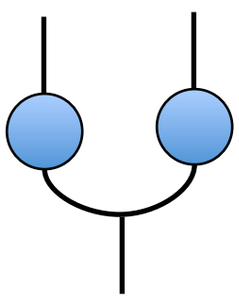


tensor contraction

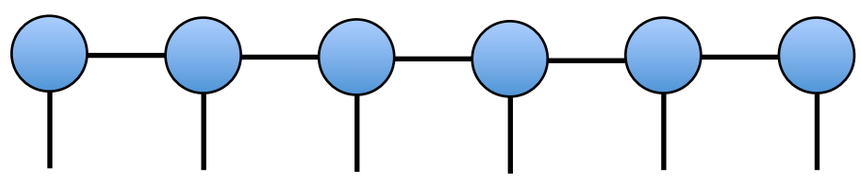


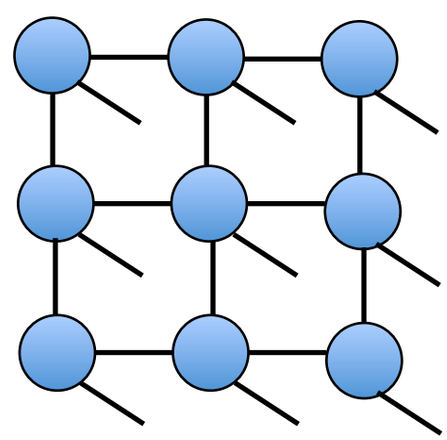
tensor contraction

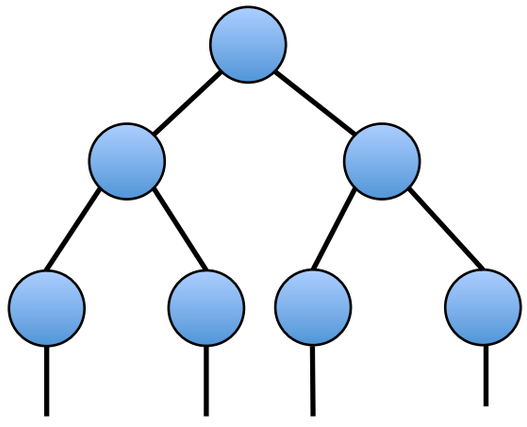
Diagram notations

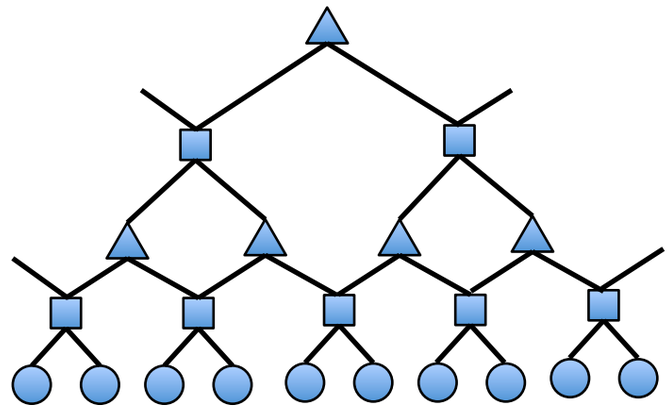
Operations	Formula	Diagram
n-mode product of two tensors contration	$\mathcal{A} \times_n \mathcal{B}$	
Kronecer product of two tensors	$\mathcal{A} \otimes \mathcal{B}$	
Khatri-Rao product of two matrices	$\mathcal{A} \odot \mathcal{B}$	
Hadamard (component) product	$\mathcal{A} * \mathcal{B}$	

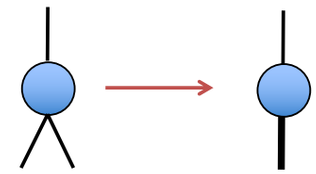
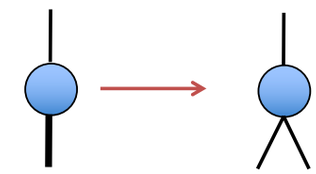
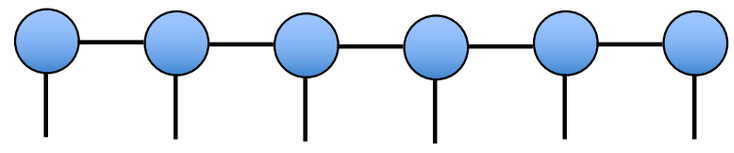
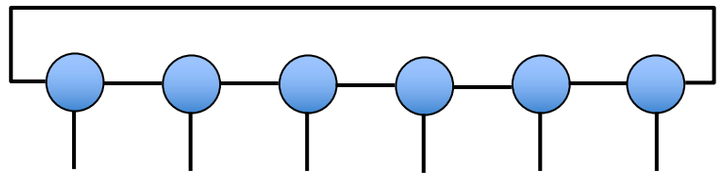
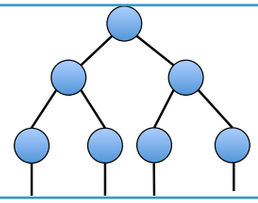
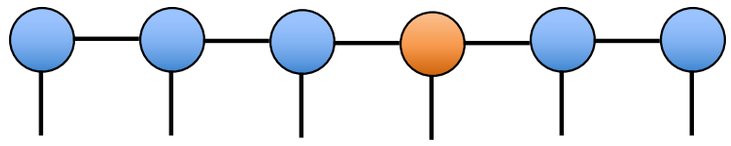
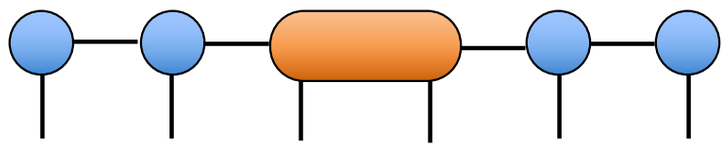
Tensor networks in physics:
imposing prior of physical wave function



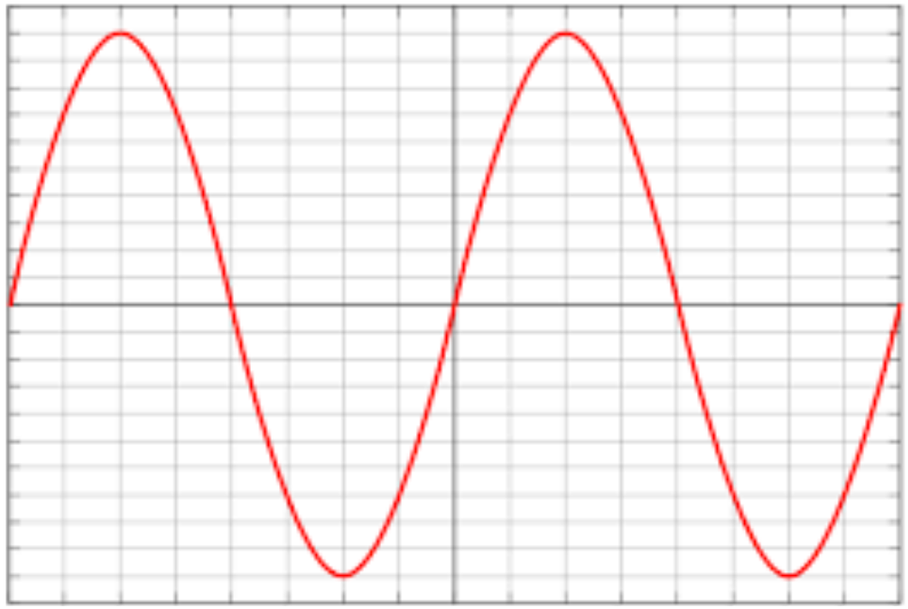






In Physics	Out of Physics	Diagram
grouping of indices	unfolding, matricization	
splitting of indices	tensorizing	
matrix product states	tensor train decomposition	
periodic boundary MPS	tensor chain decomposition	
tree tensor networks	hierarchical Tucker decompostion	
single-site DMRG	alternating least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

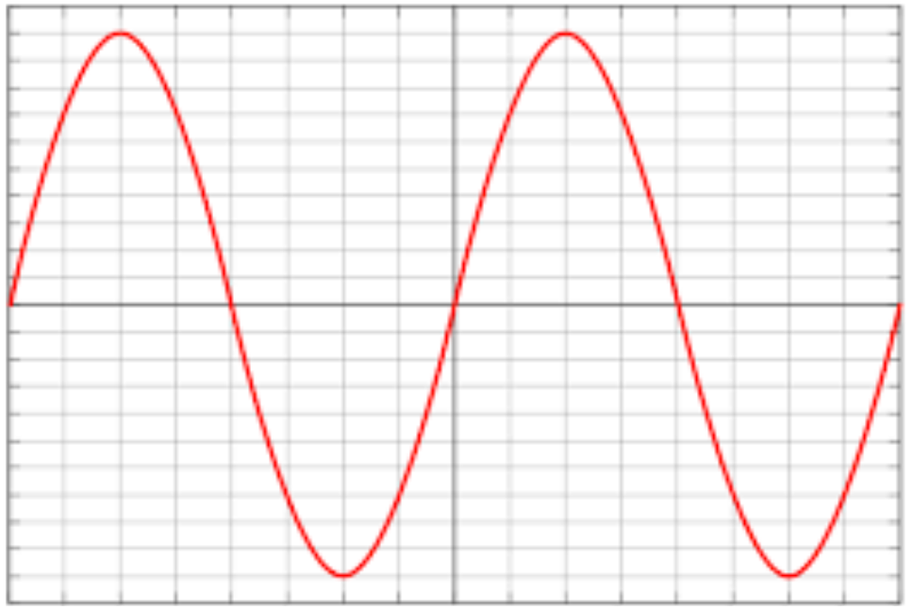
Exploring internal structures in the data



[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]

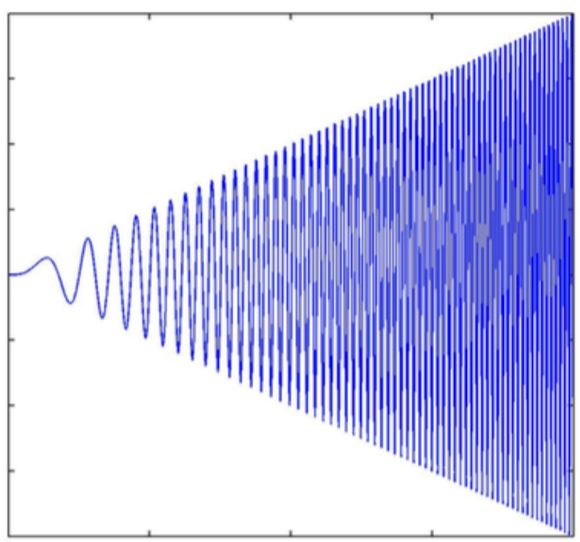
10^6 data points in a vector

Exploring internal structures in the data

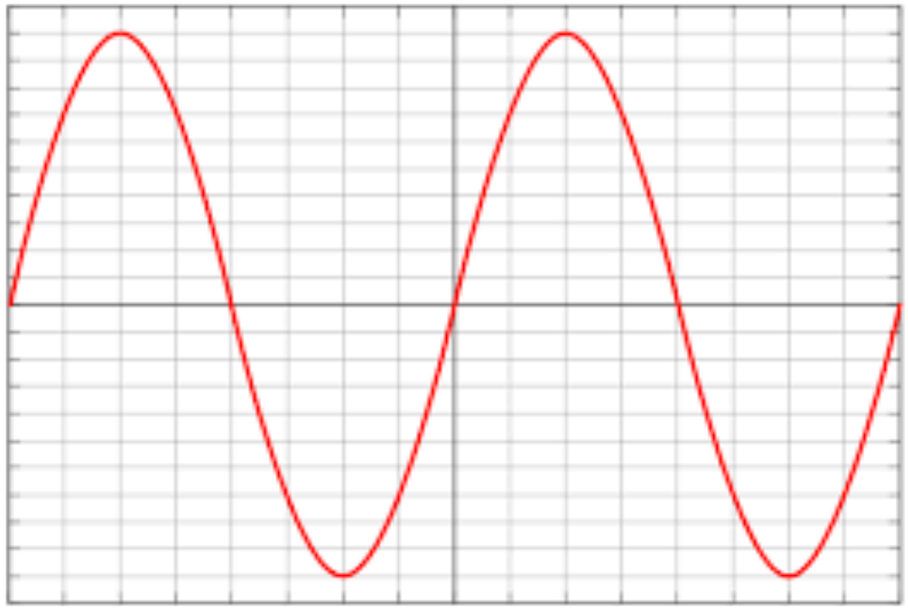


[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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

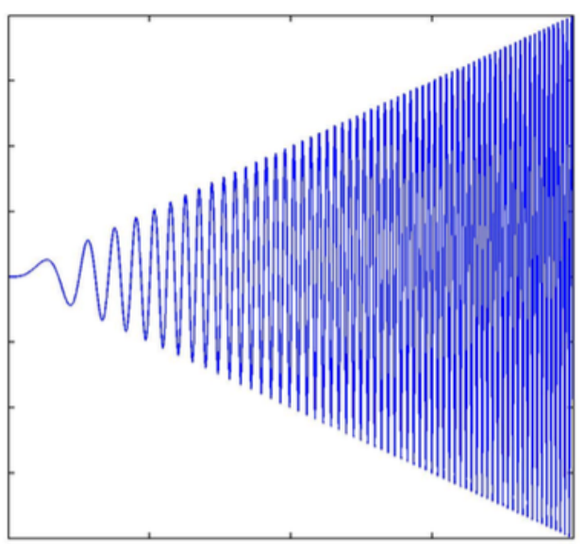


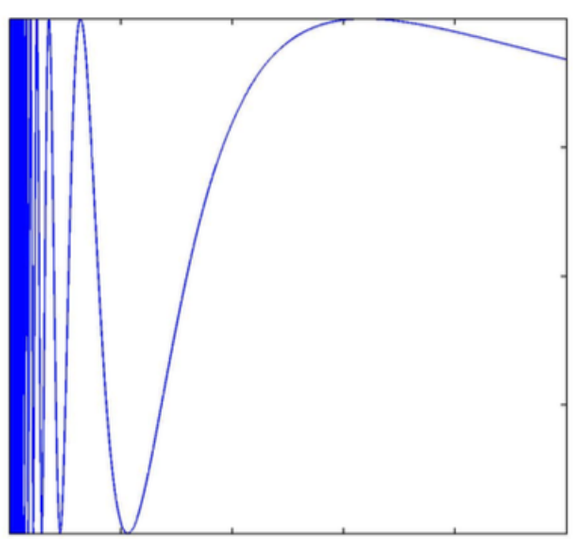
Exploring internal structures in the data



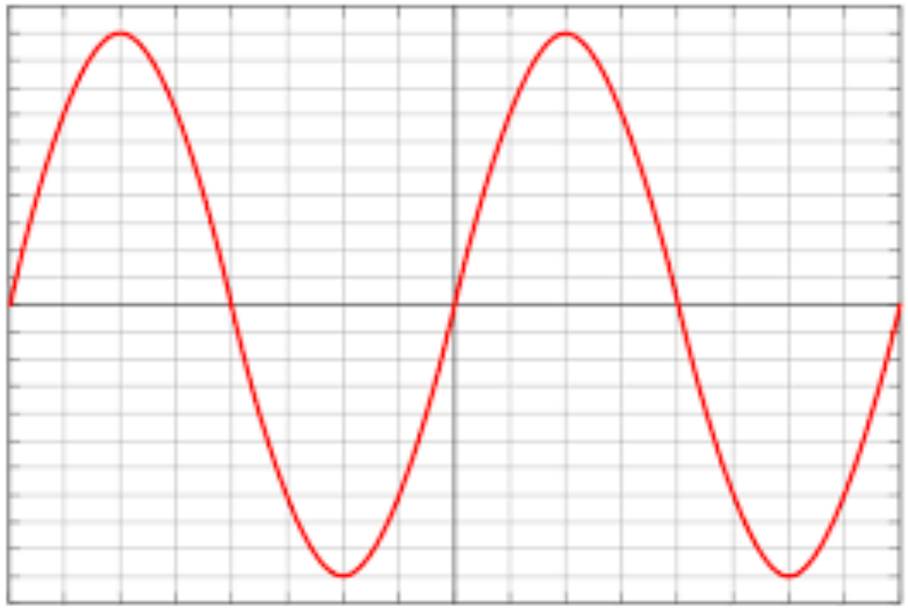
[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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



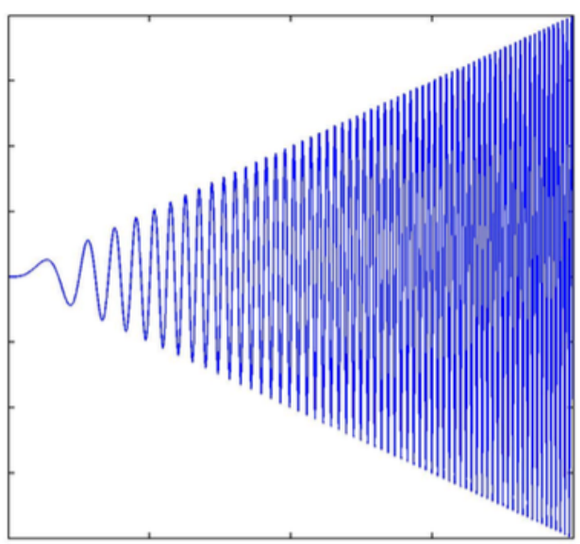


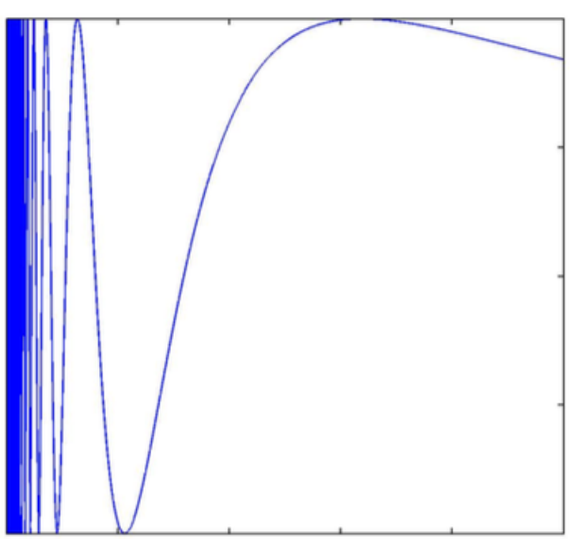
Exploring internal structures in the data

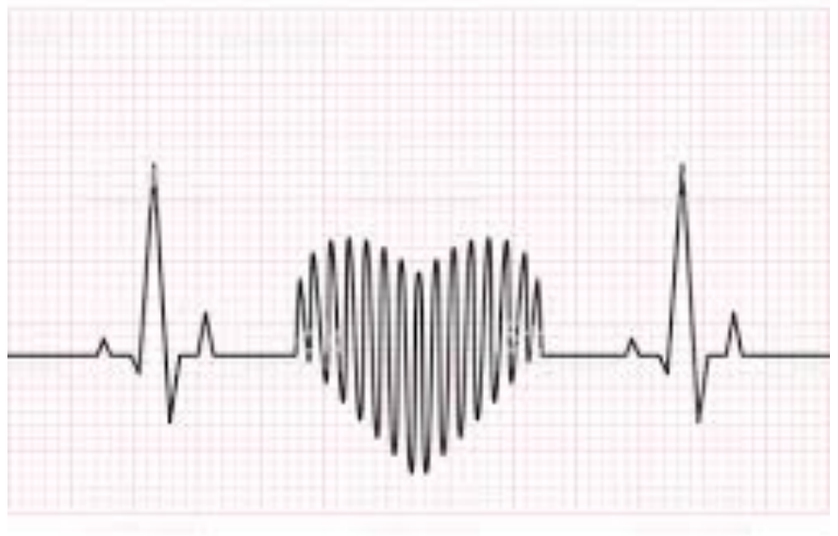


[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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]

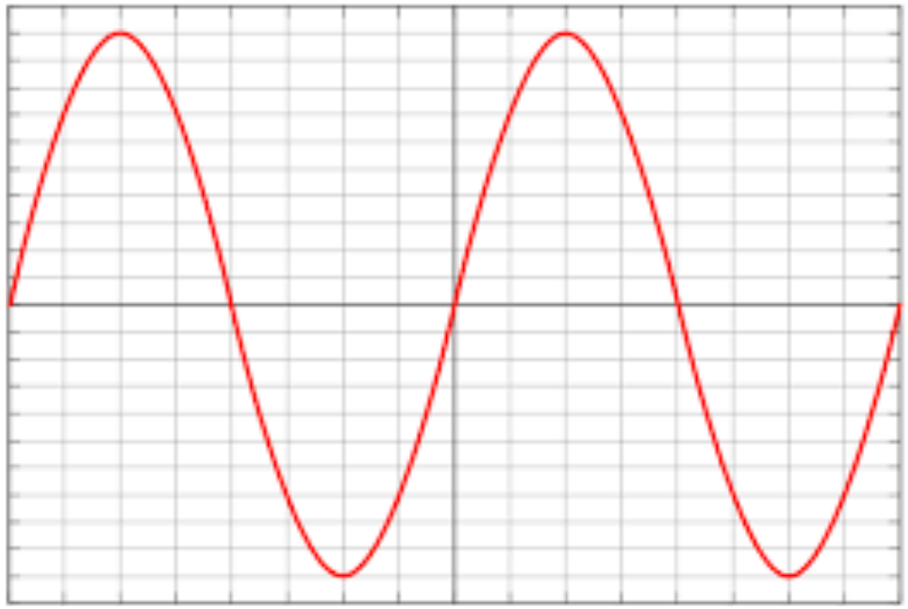
10^6 data points in a vector





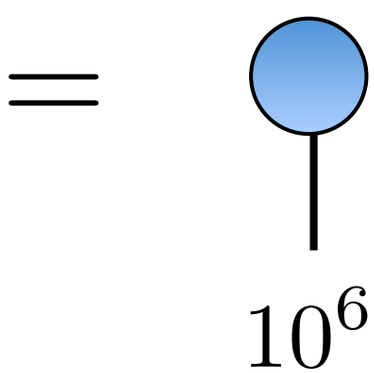


Exploring internal structures in the data

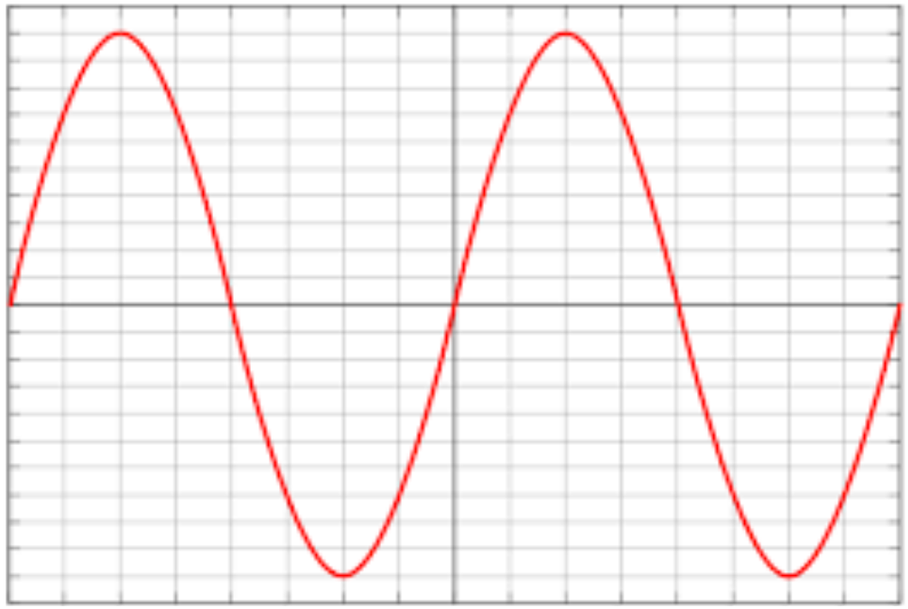


[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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

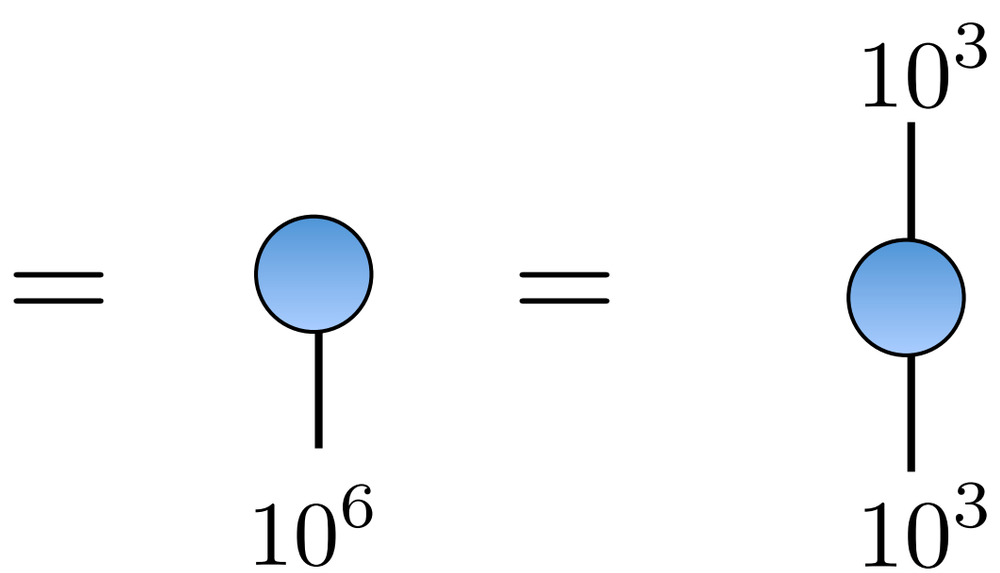


Exploring internal structures in the data

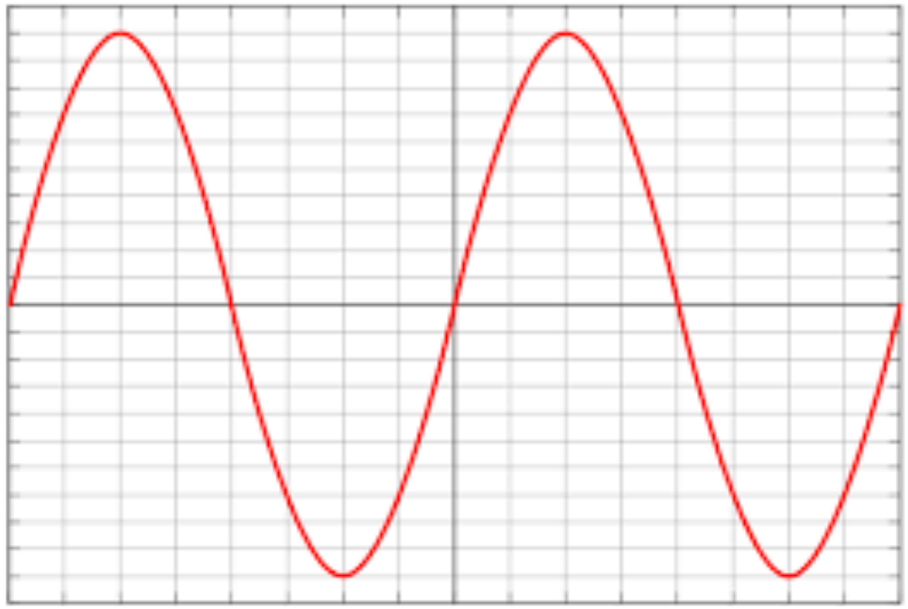


[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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

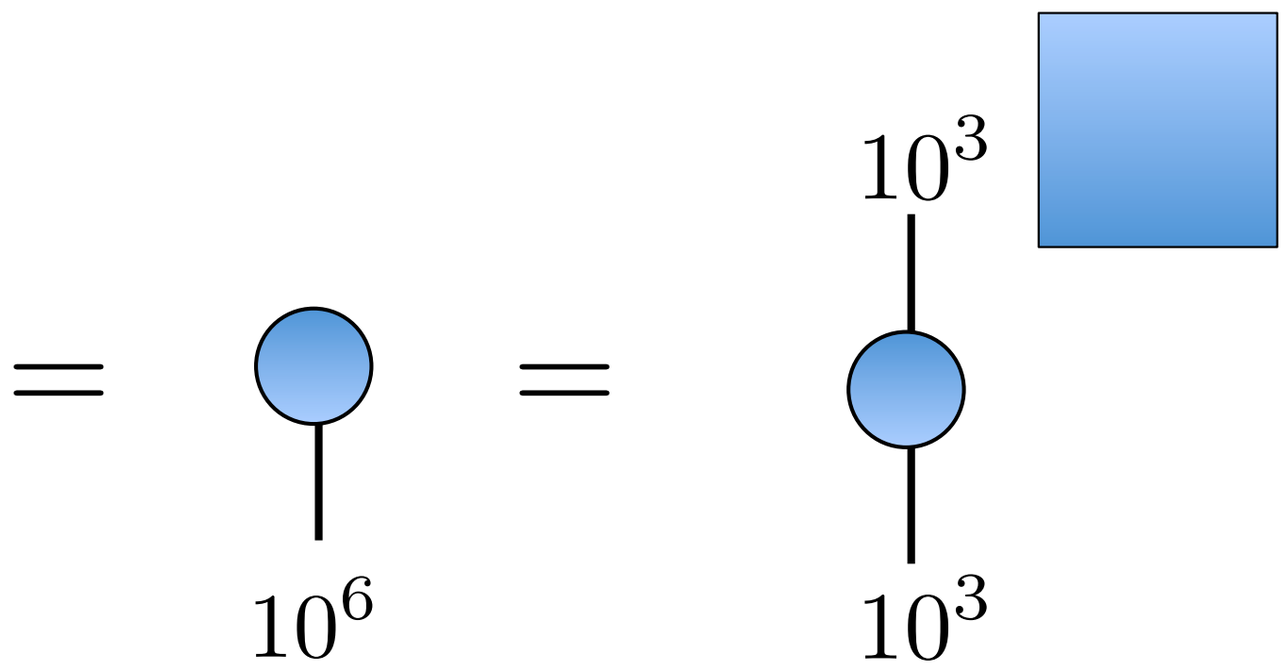


Exploring internal structures in the data

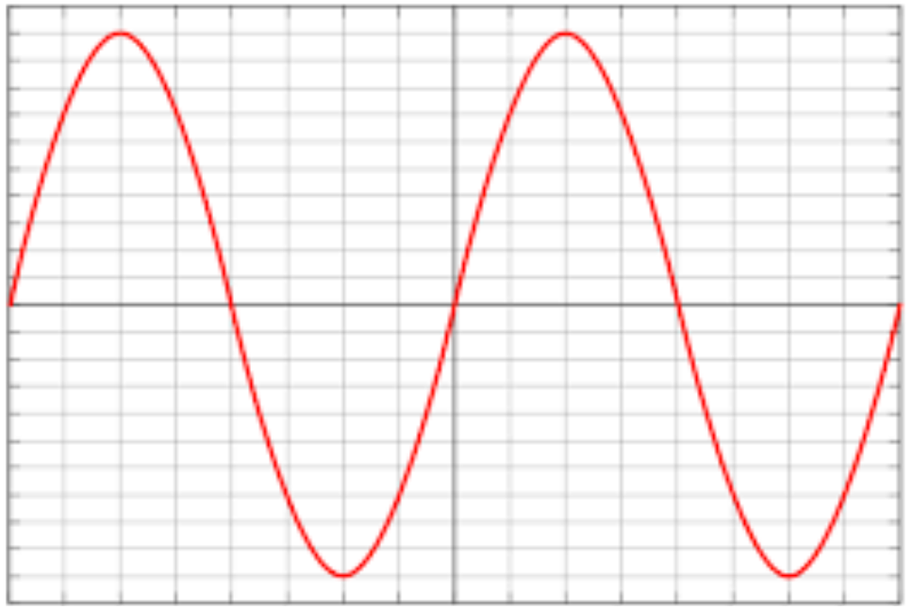


[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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

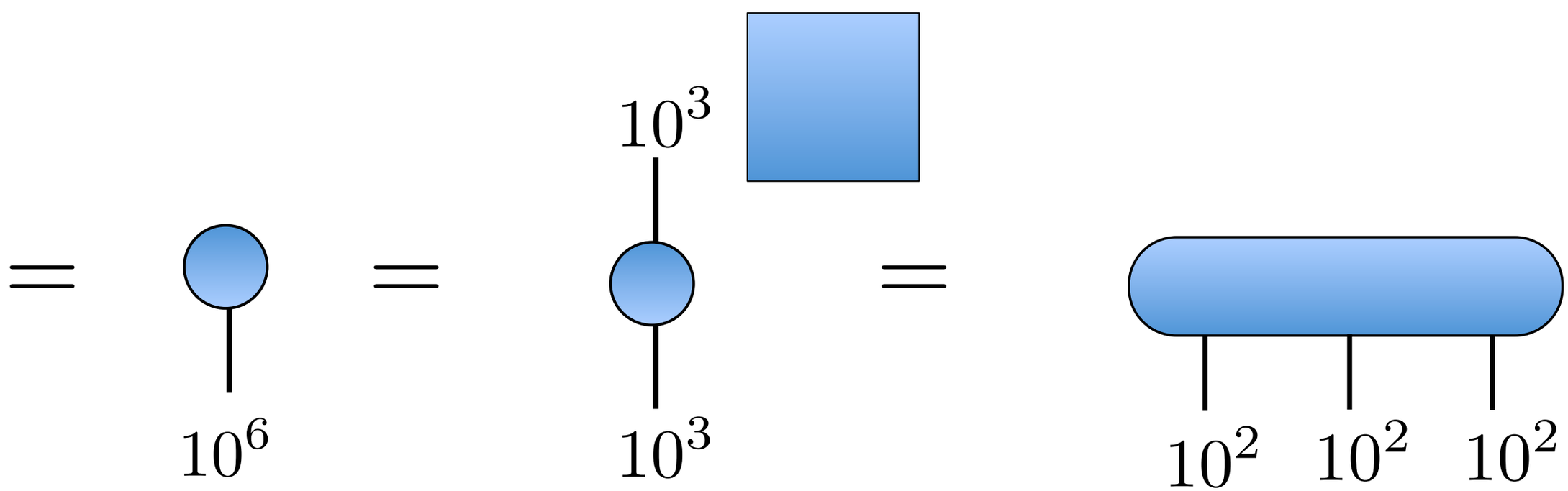


Exploring internal structures in the data

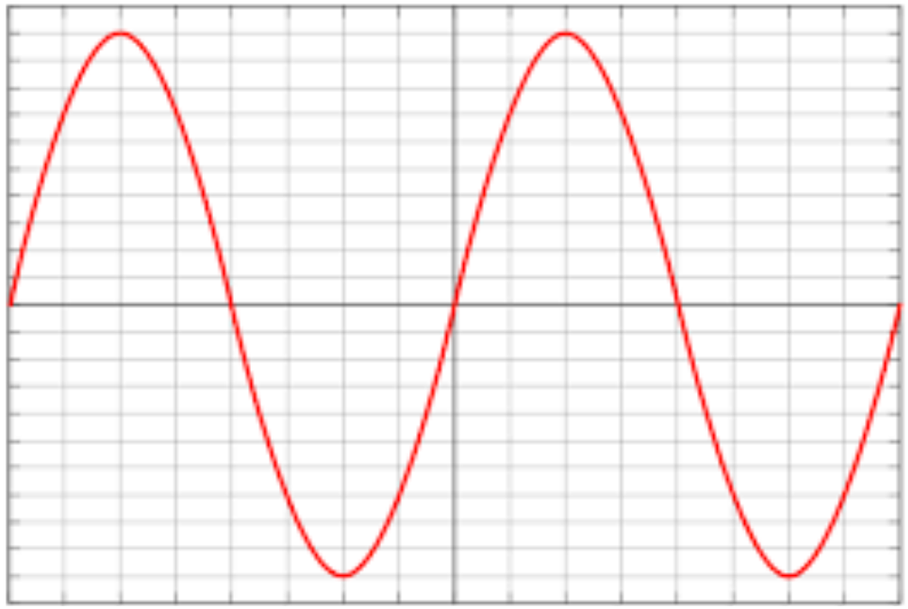


[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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

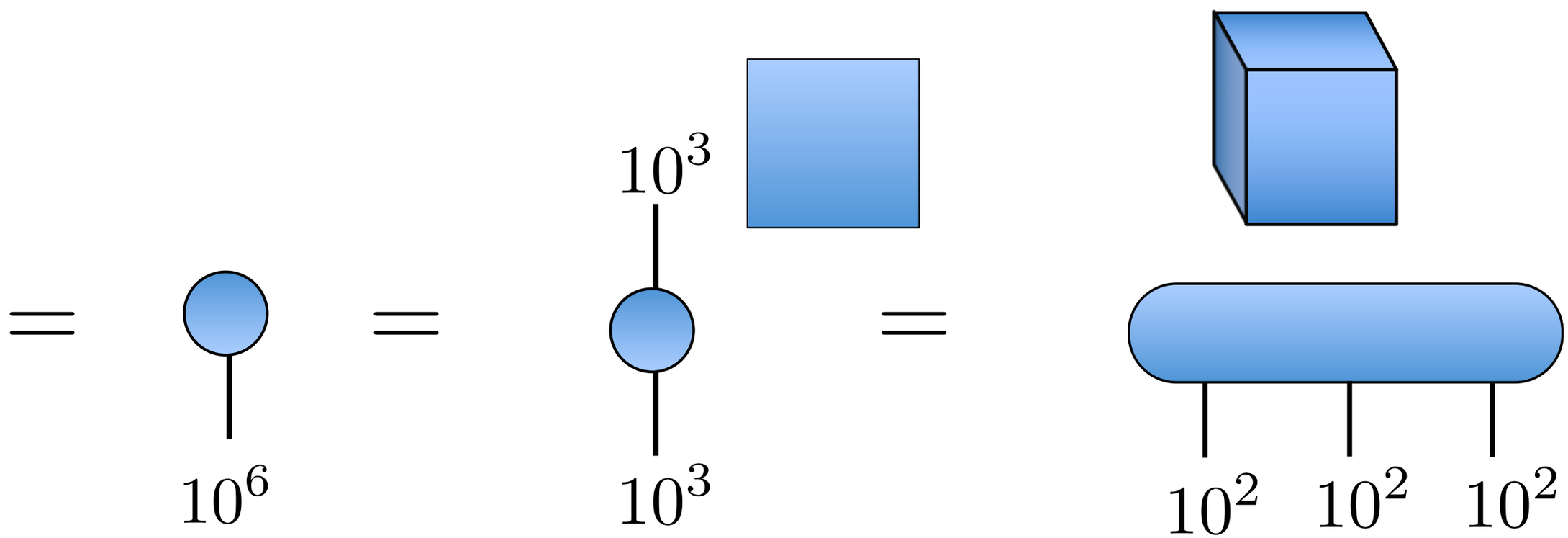


Exploring internal structures in the data

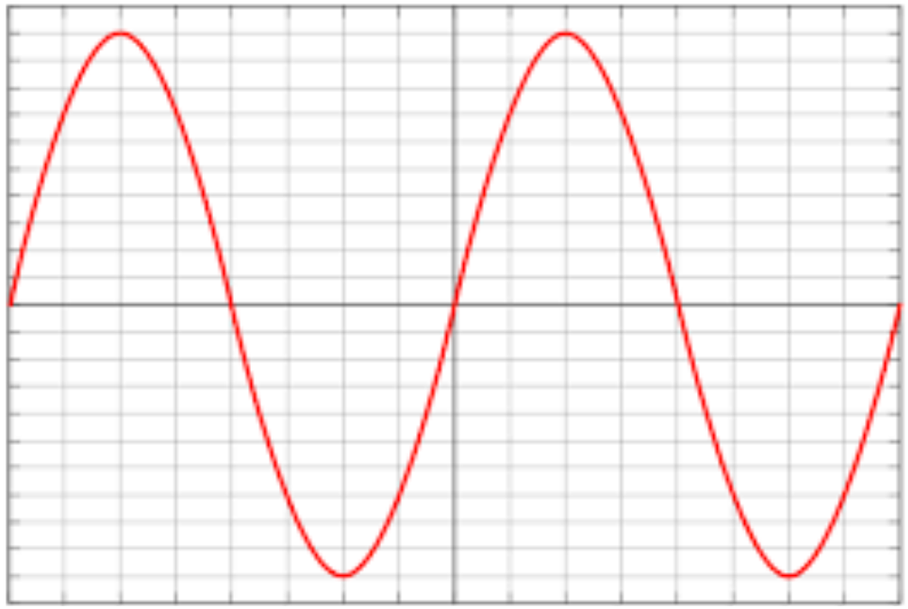


[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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

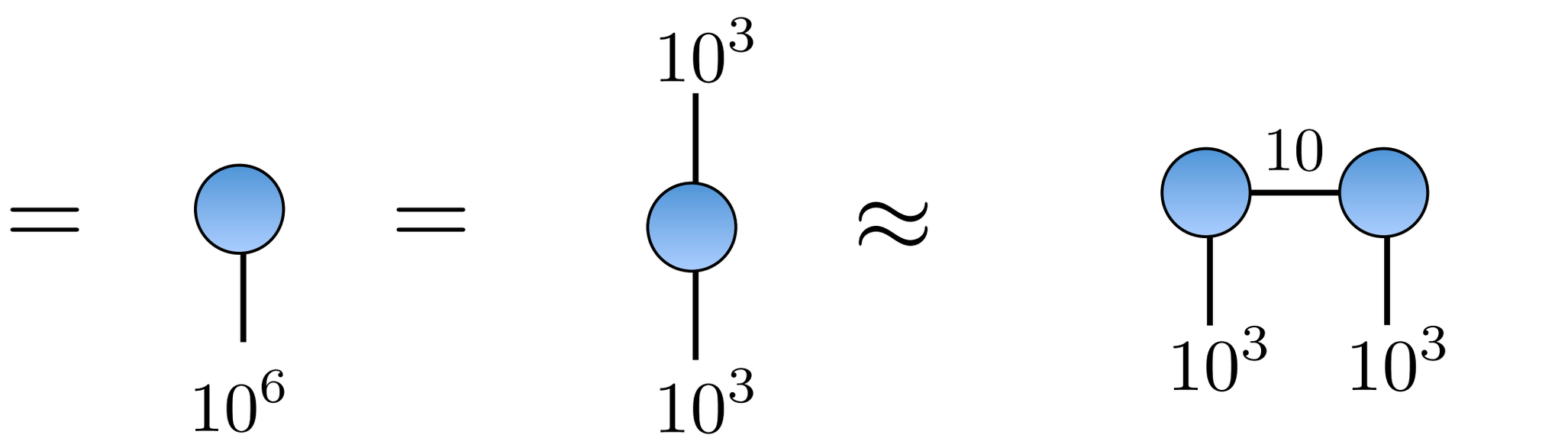


Exploring internal structures in the data



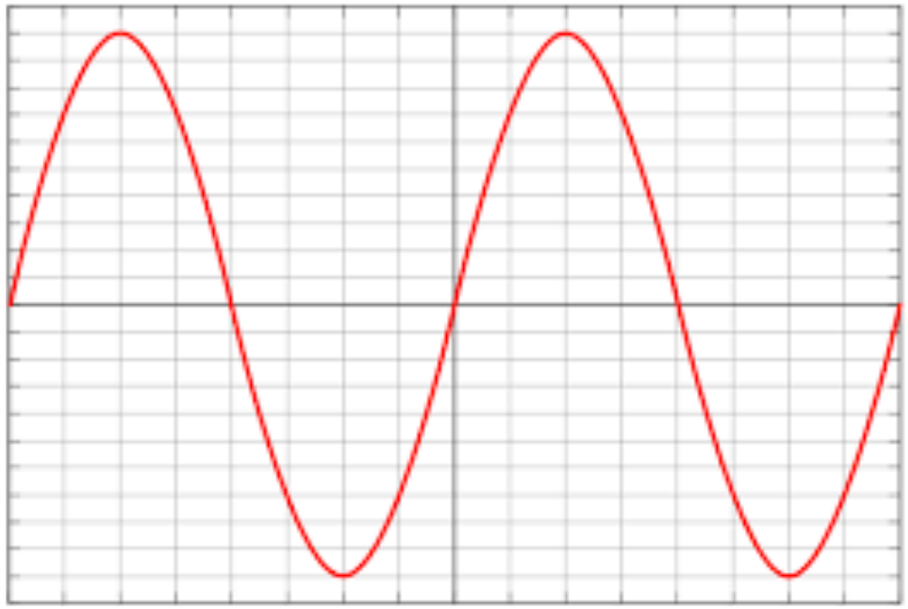
[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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



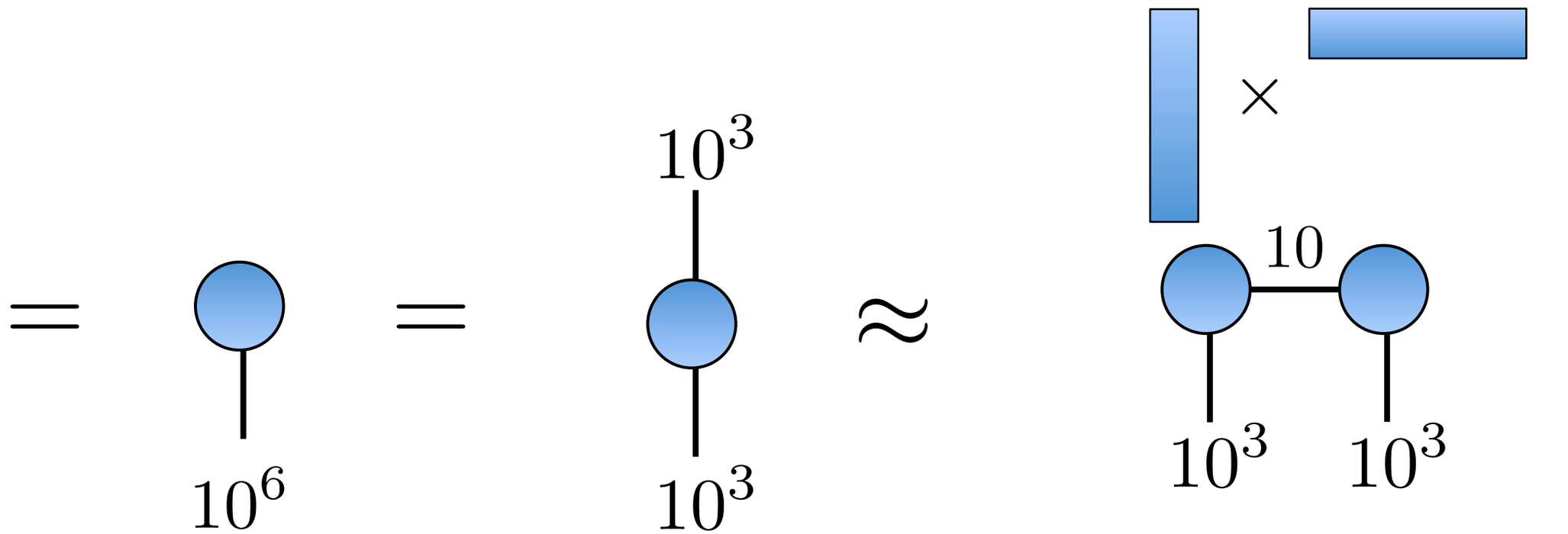
2×10^4 scalars

Exploring internal structures in the data



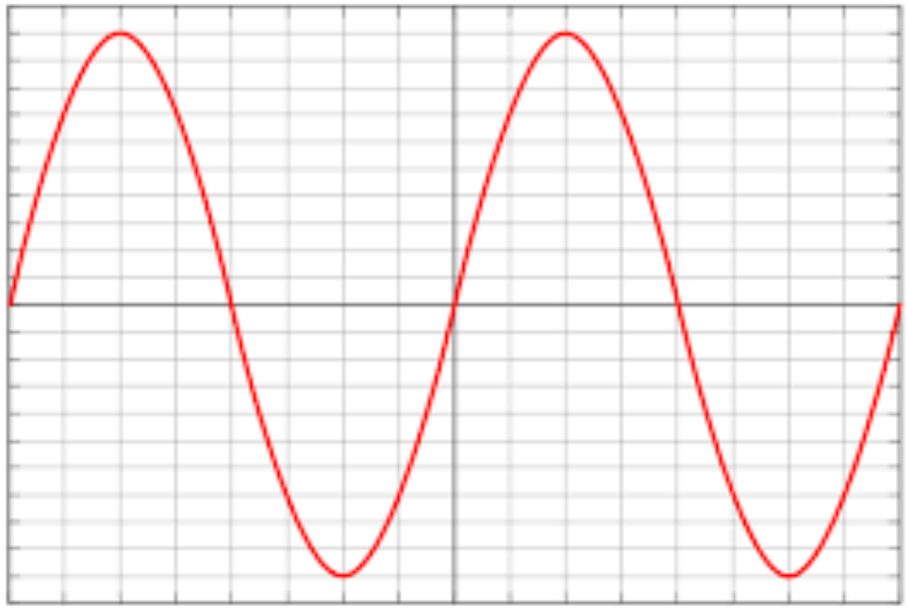
[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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



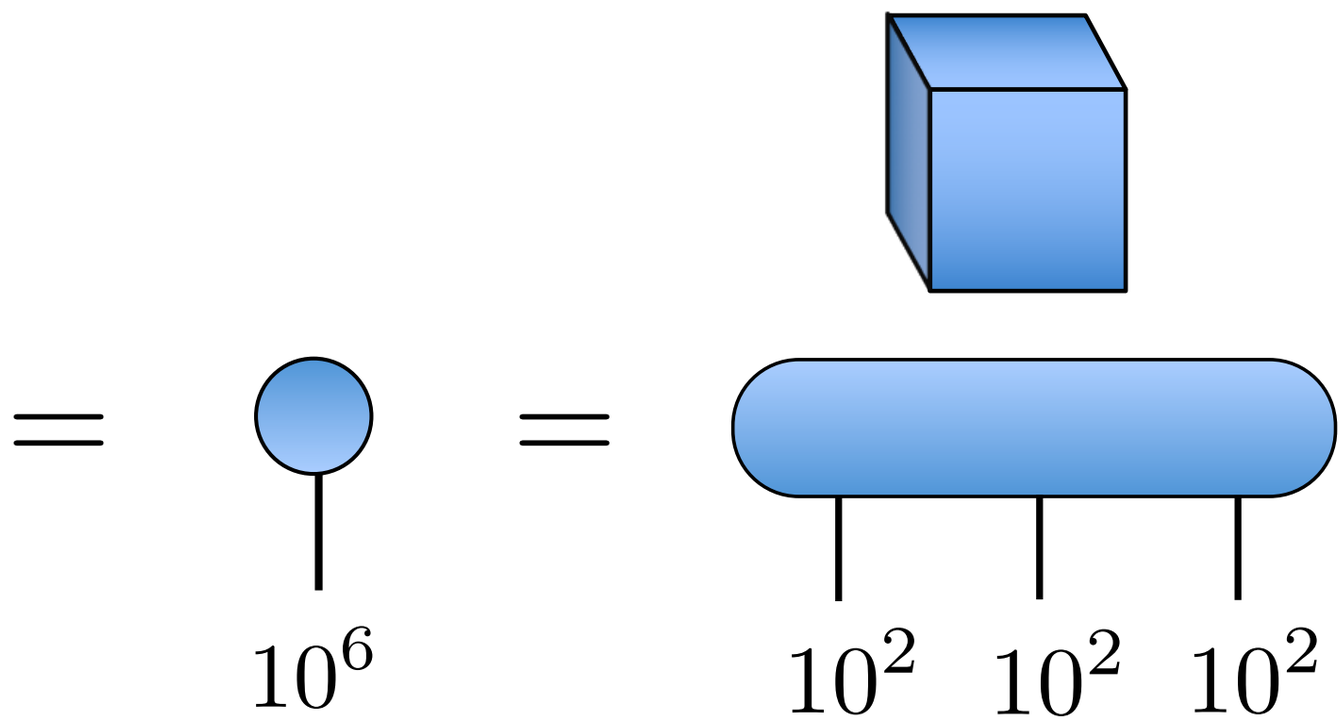
2×10^4 scalars

Exploring internal structures in the data

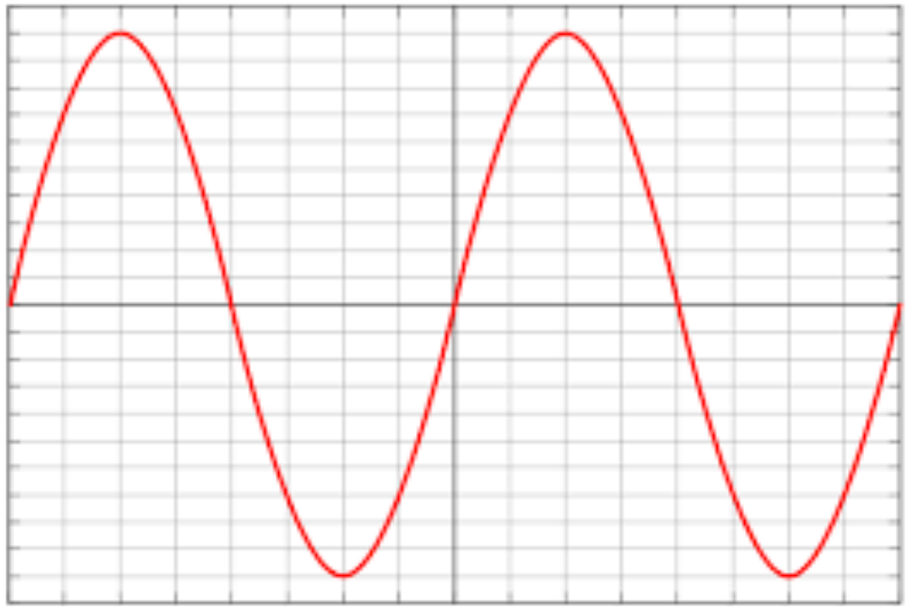


[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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

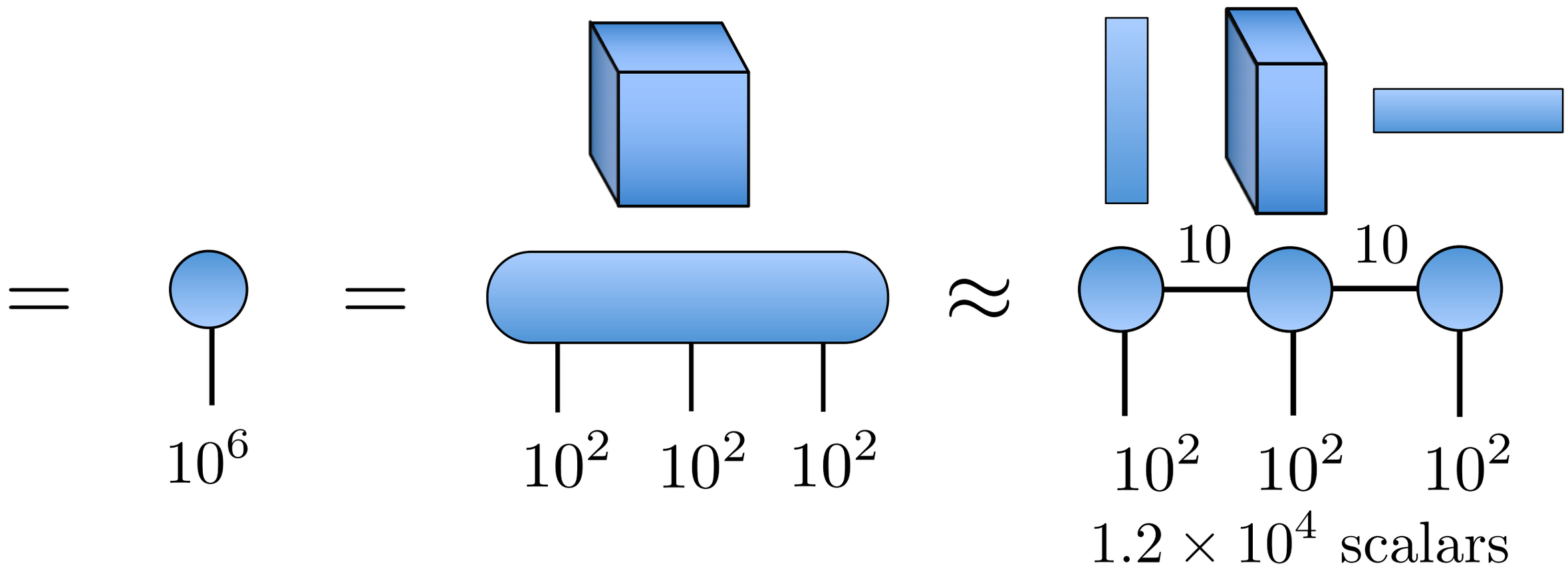


Exploring internal structures in the data



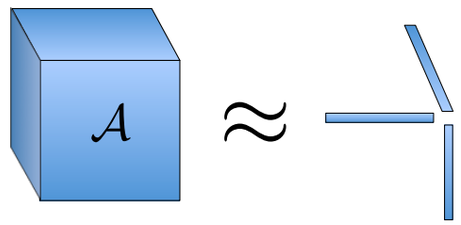
[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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



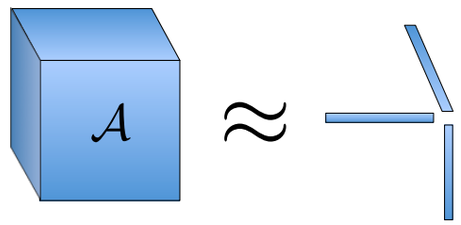
Representation of a large tensor: rank-one

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



Representation of a large tensor: rank-one

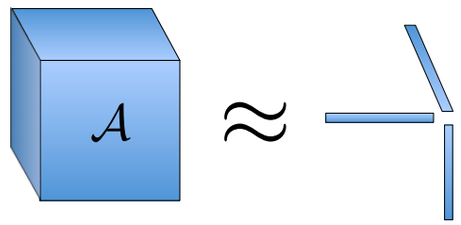
$$\mathcal{A} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \mathbf{a}^{(n)}$$



independent modes

Representation of a large tensor: rank-one

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

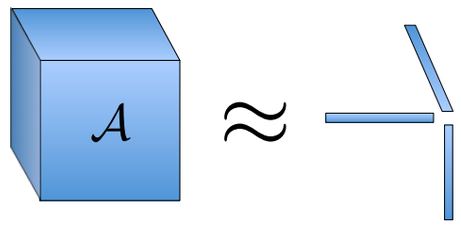


independent modes

factorized pure state

Representation of a large tensor: rank-one

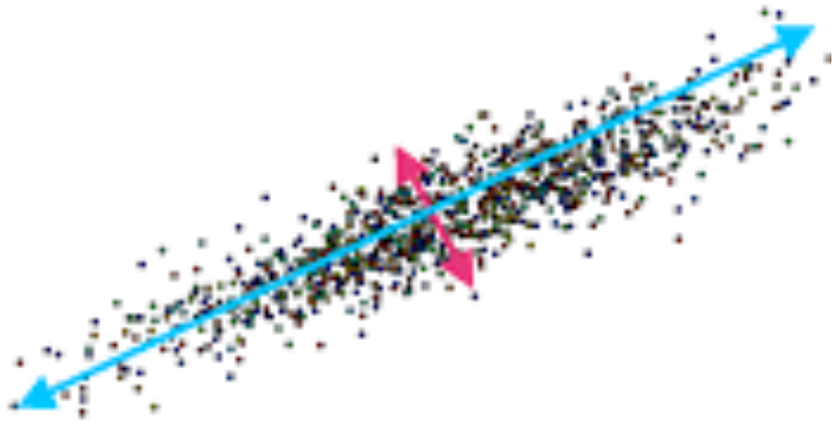
$$\mathcal{A} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \mathbf{a}^{(n)}$$



independent modes

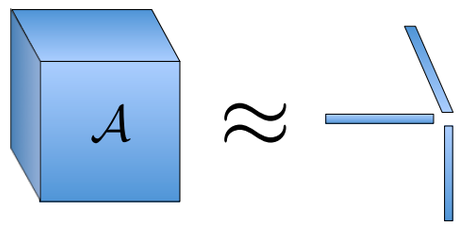
factorized pure state

principled component



Representation of a large tensor: rank-one

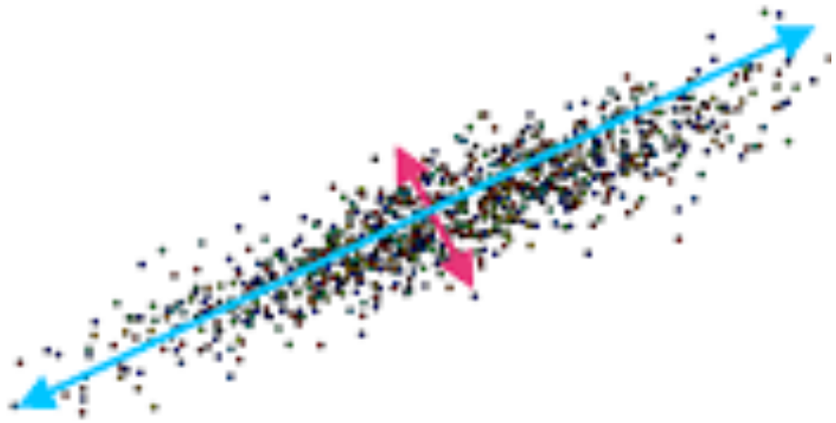
$$\mathcal{A} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \mathbf{a}^{(n)}$$



independent modes

factorized pure state

principled component

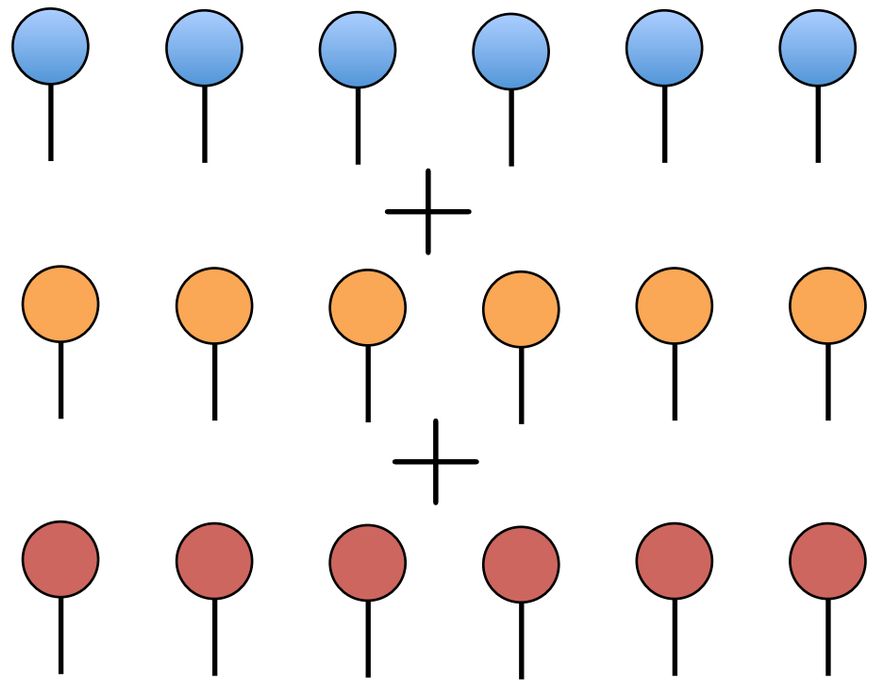


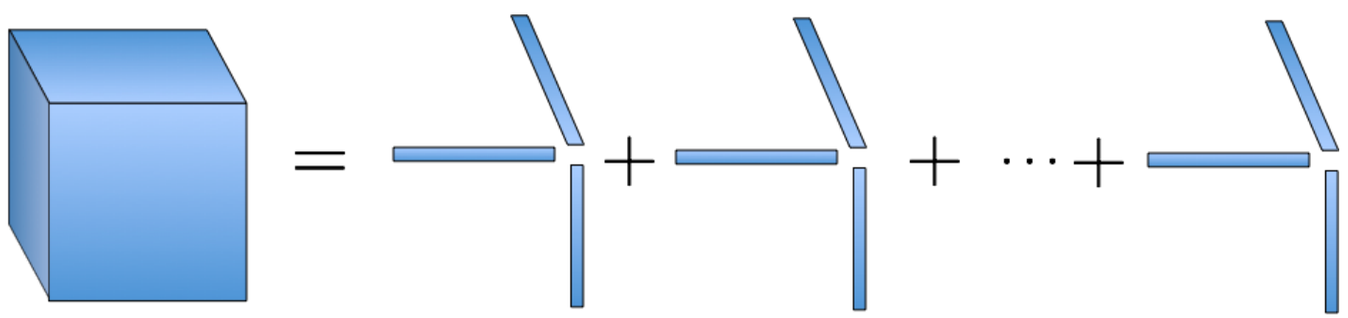
reminiscent of the *mean-field* approximation

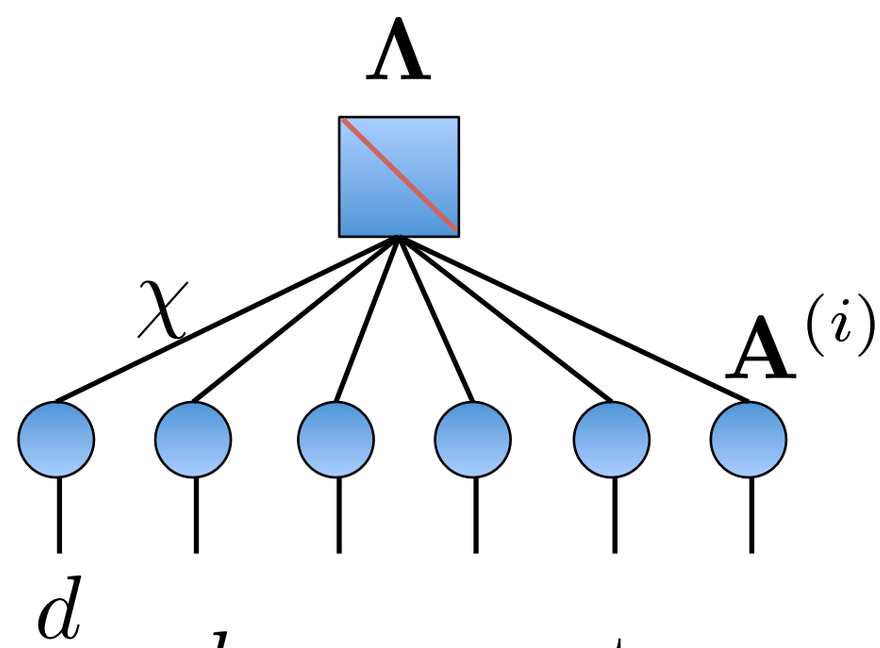
$$p(\{x_1, x_2, ..., x_n\}) = p(x_1)p(x_2)...p(x_n)$$

Canonical Polyadic (CP) decomposition

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

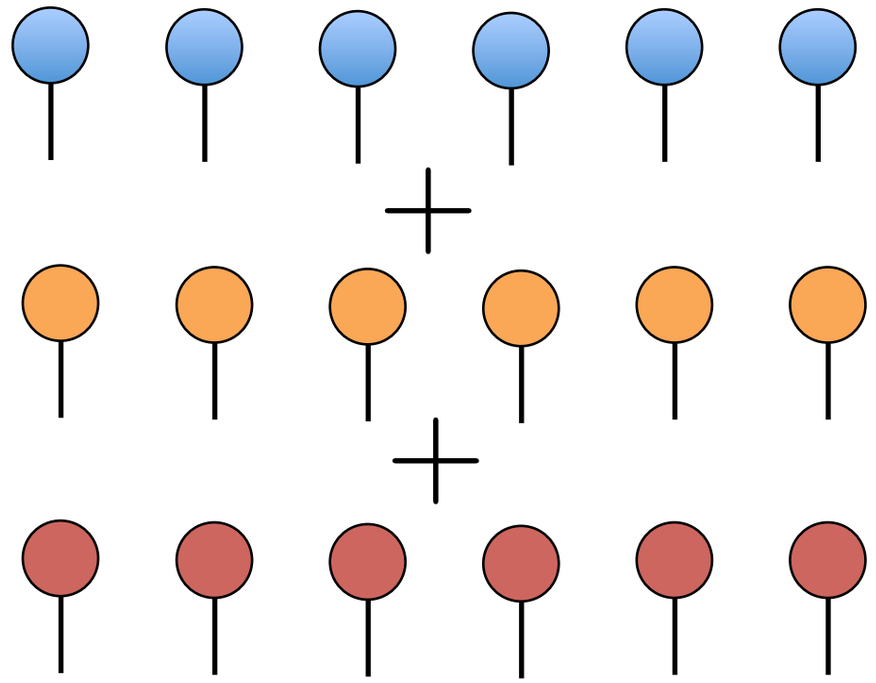






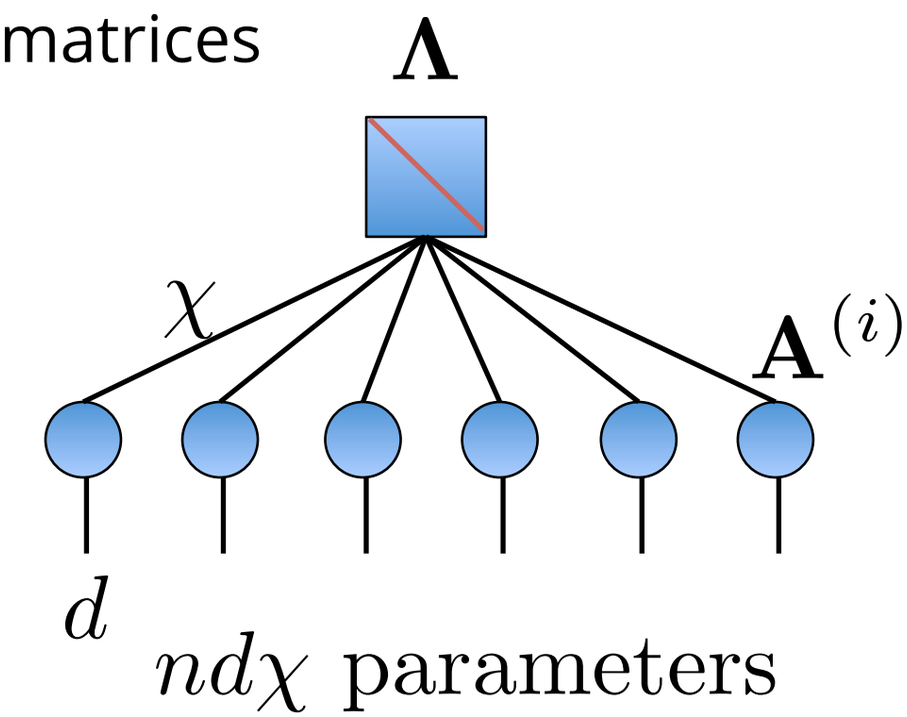
Canonical Polyadic (CP) decomposition

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

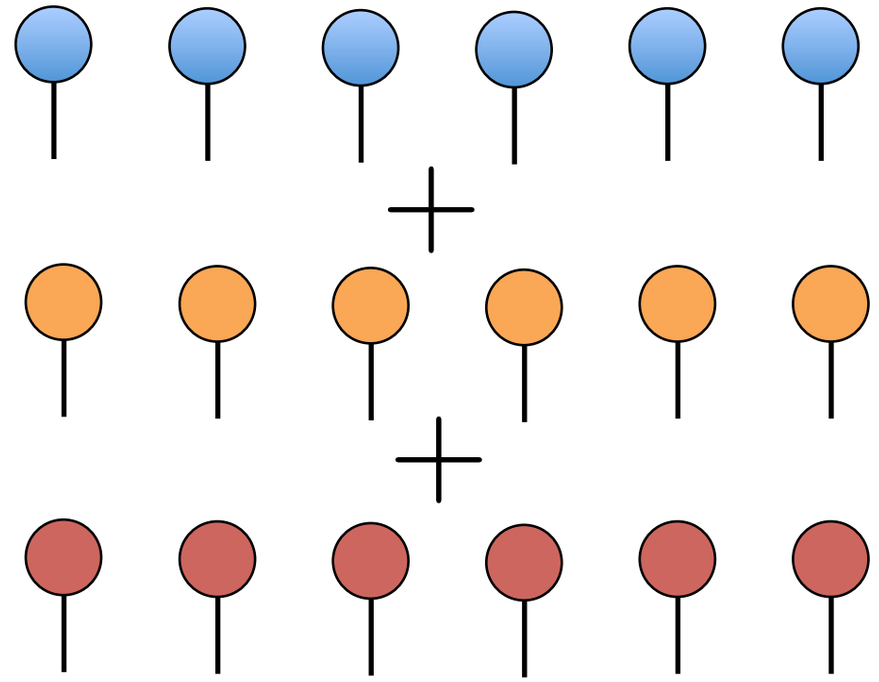


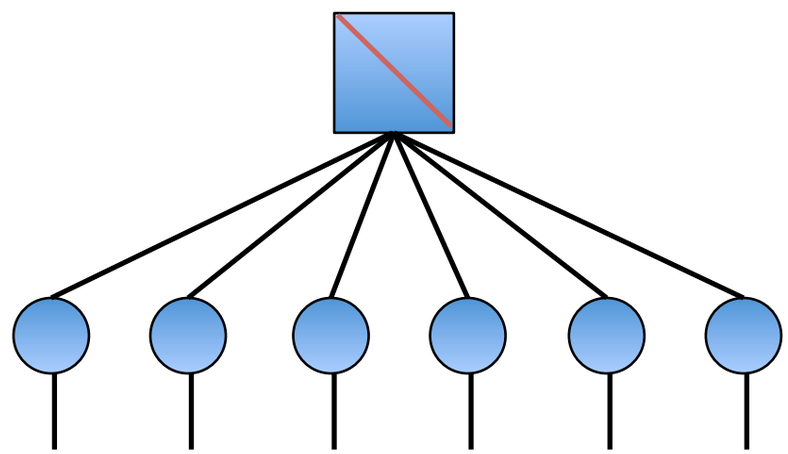
Convenient representations of CP using unfolded matrices

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



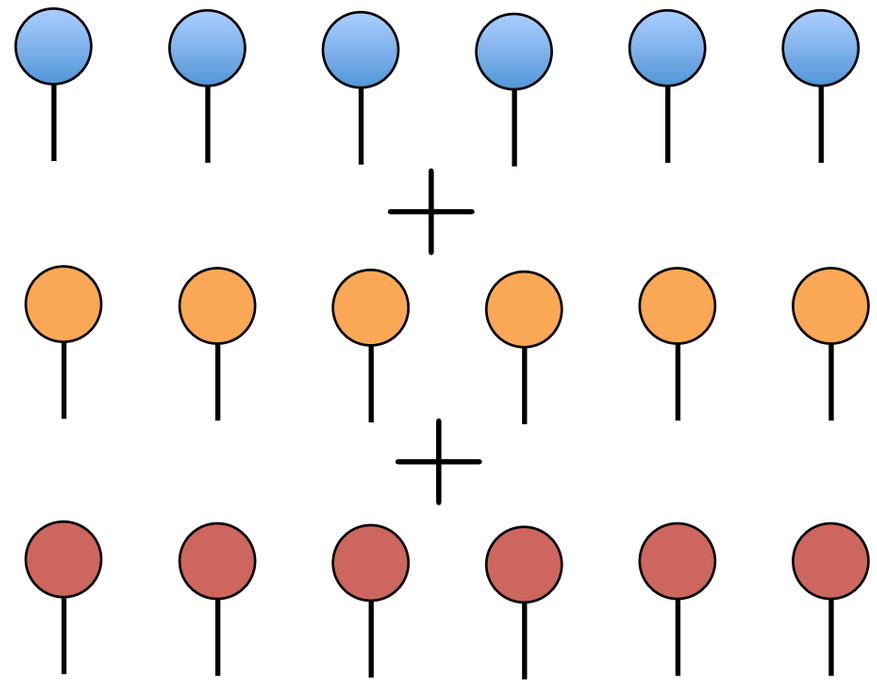
Canonical Polyadic (CP) decomposition

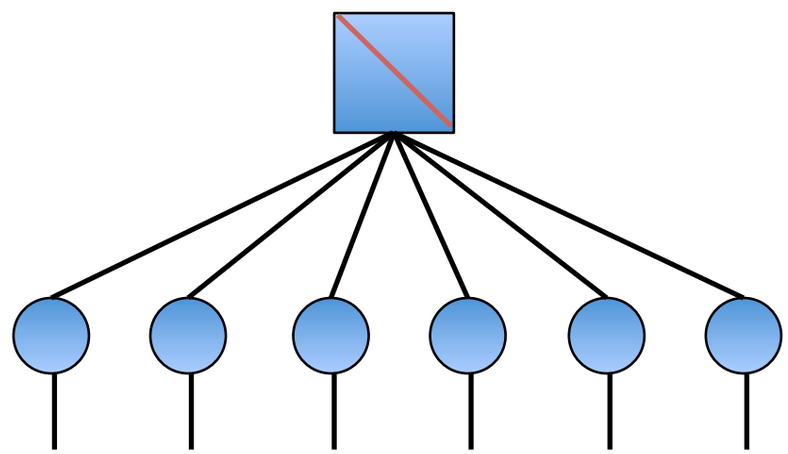




Canonical Polyadic (CP) decomposition

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

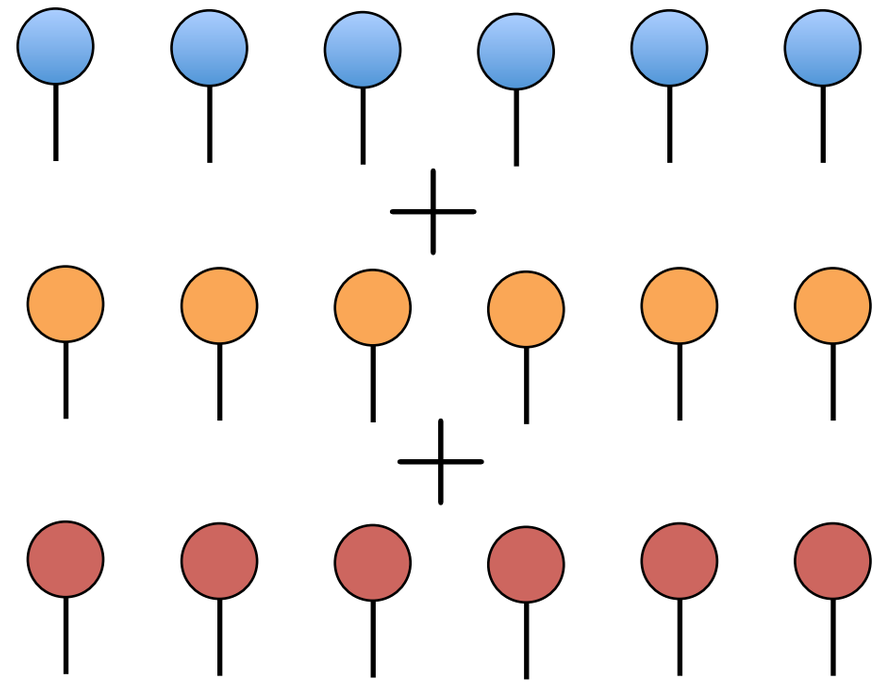


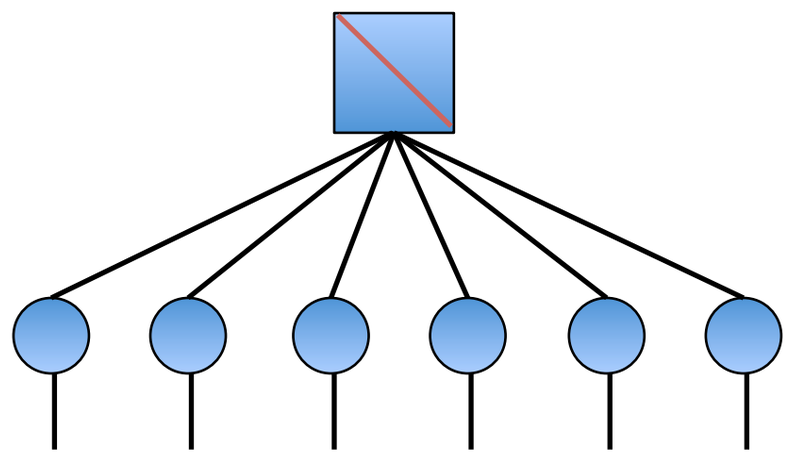


Canonical Polyadic (CP) decomposition

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.



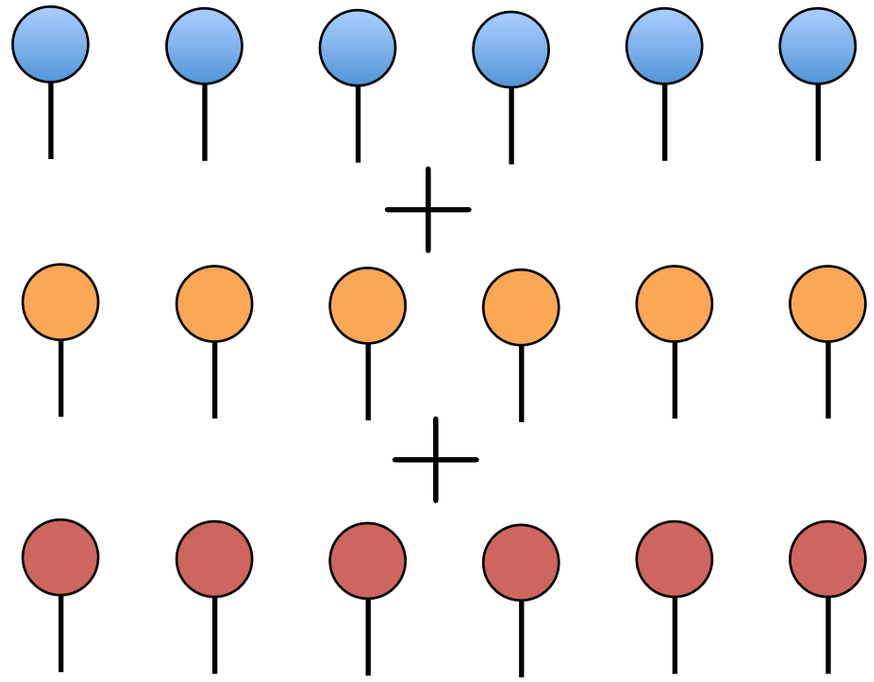


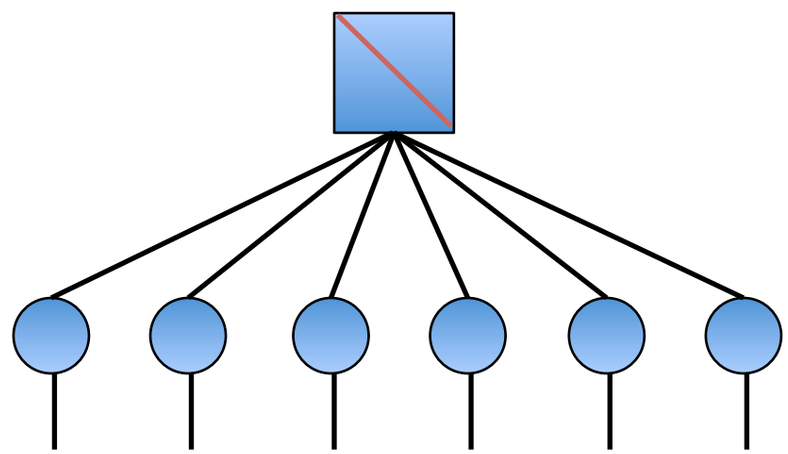
Canonical Polyadic (CP) decomposition

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.



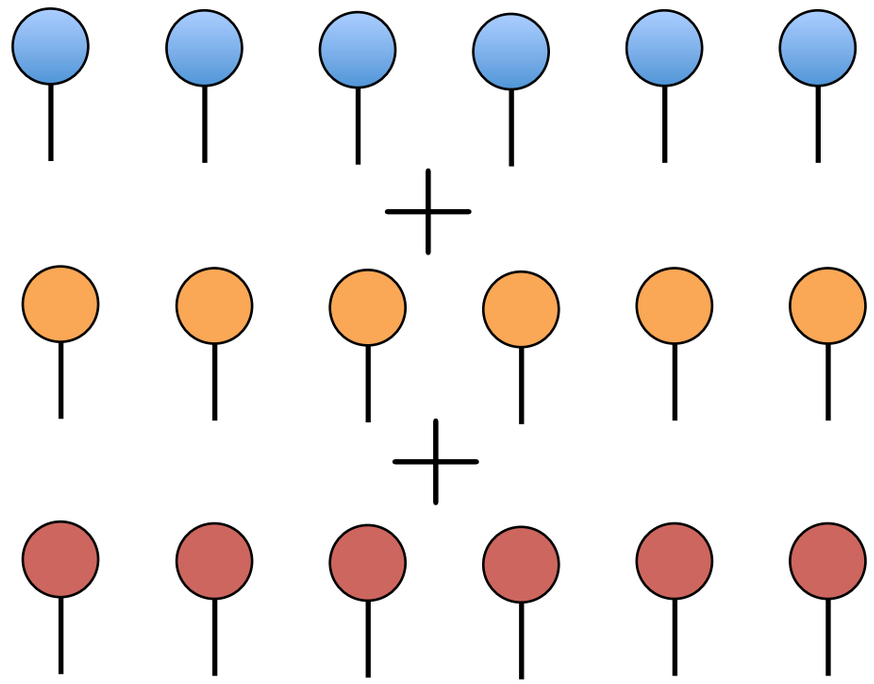


Canonical Polyadic (CP) decomposition

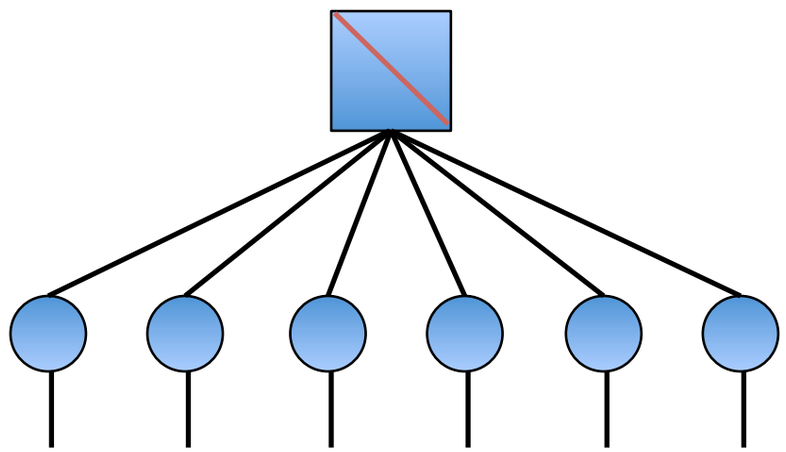
Also known as *Polyadic*, *PARAFAC*, *CANDECOMP*, and *Topographic*.

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.

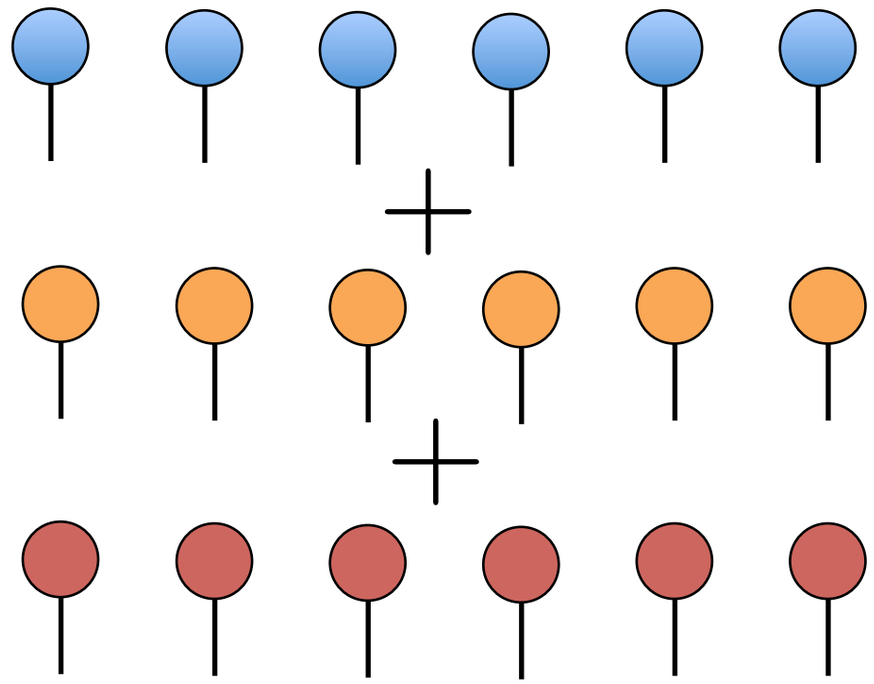


Canonical Polyadic (CP) decomposition

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

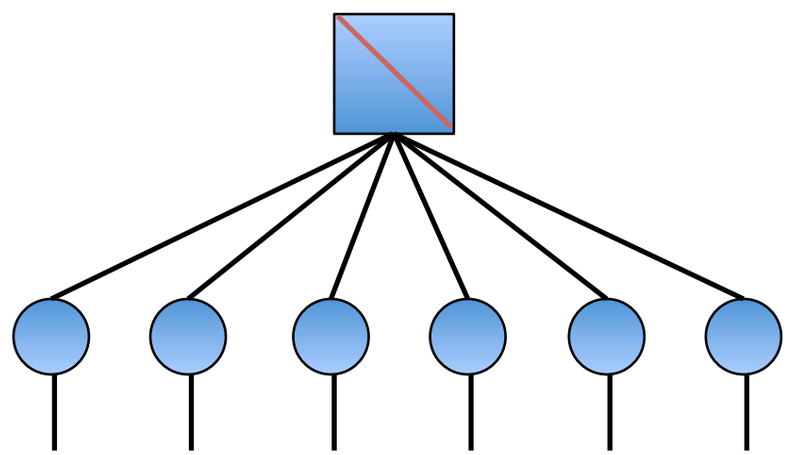
First proposed in 1927 by *Hitchcock,* then became popular in 1970's in psychometric and in 1980's in chemometrics.

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.

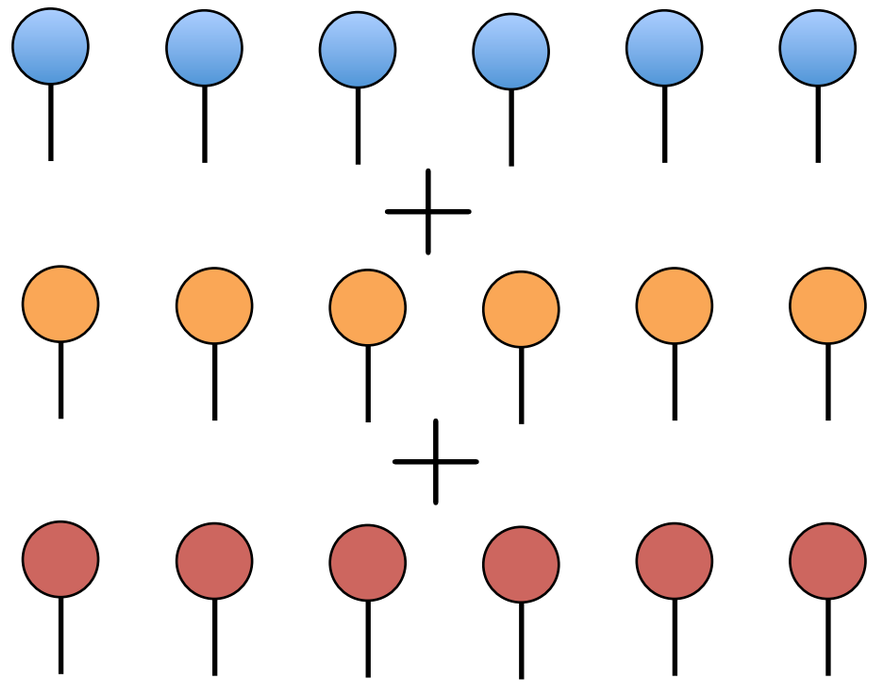


Canonical Polyadic (CP) decomposition

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.

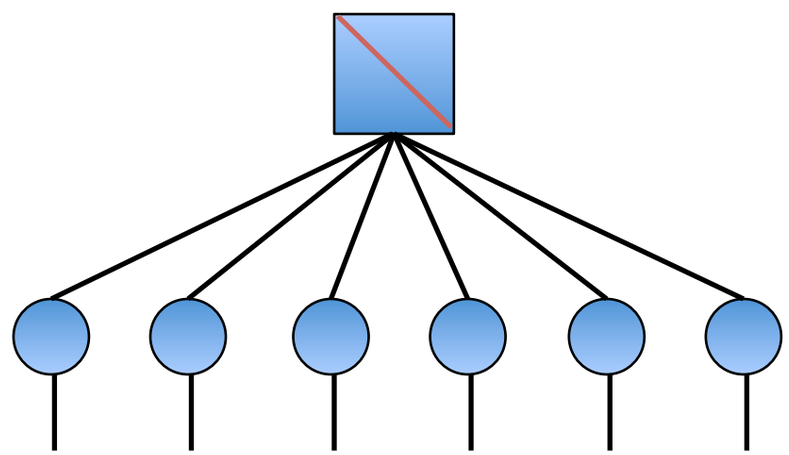
Found applications in signal processing, neural sciences, signal processing in 1990's, and image compression and classification from 2005.



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

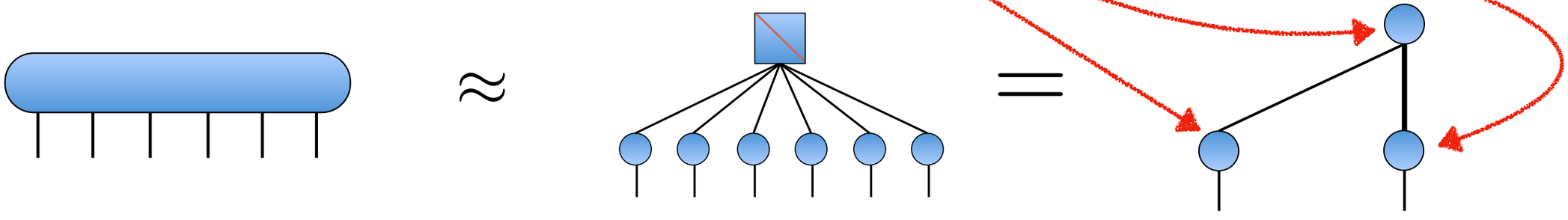
Tensor rank, or CP-rank, is defined as a *minimum number of rank-1 tensors* in the exact CP decomposition.

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



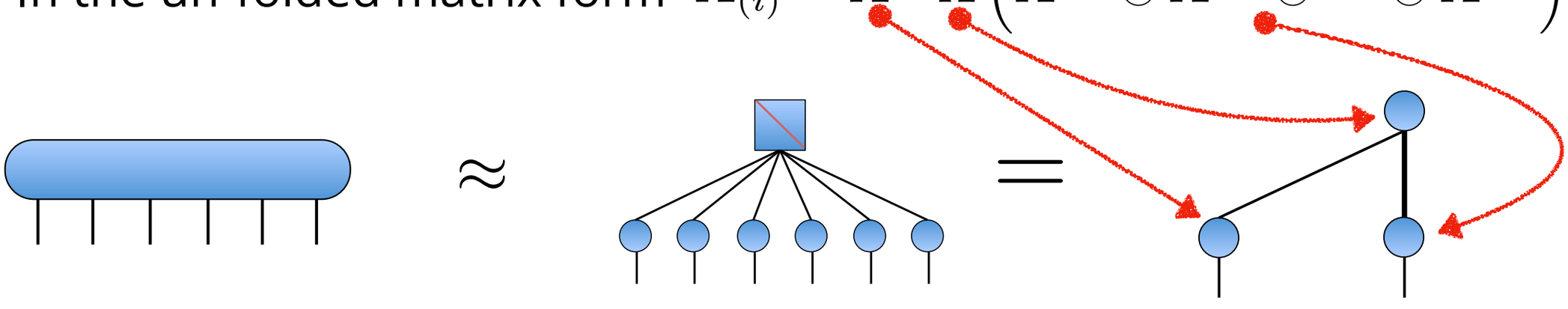
The workhorse algorithm: Alternating Least Square

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

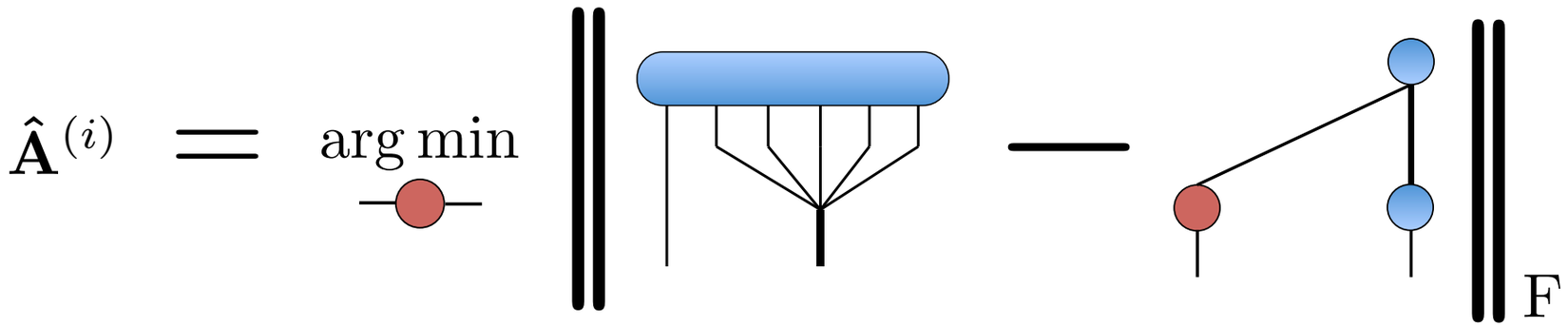


The workhorse algorithm: Alternating Least Square

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

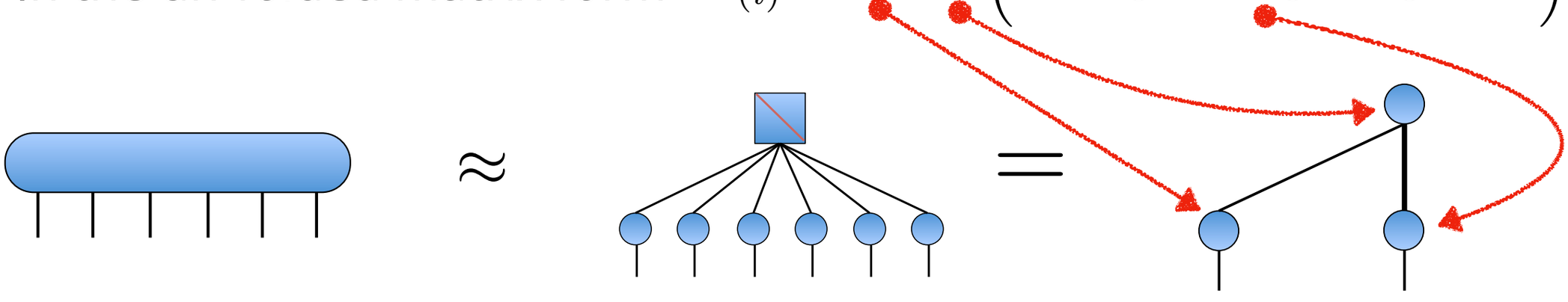


The ALS fixes all except one matrix for optimizing, the sub-problems is a least-square problem



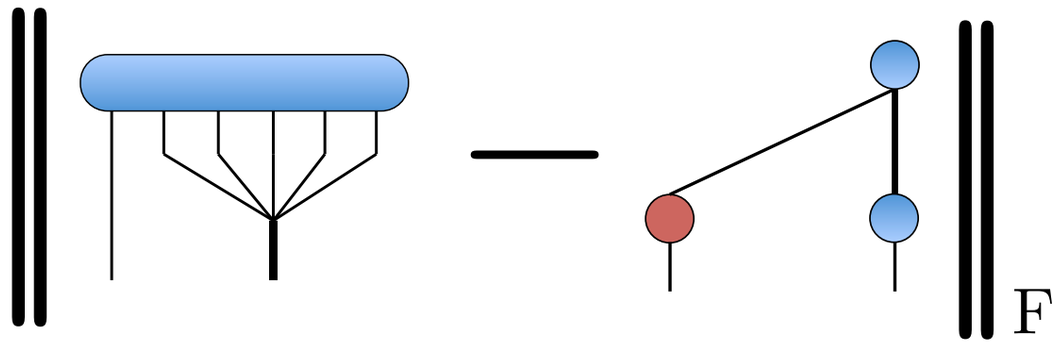
The workhorse algorithm: Alternating Least Square

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



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

$$\hat{\mathbf{A}}^{(i)} = \arg\min$$

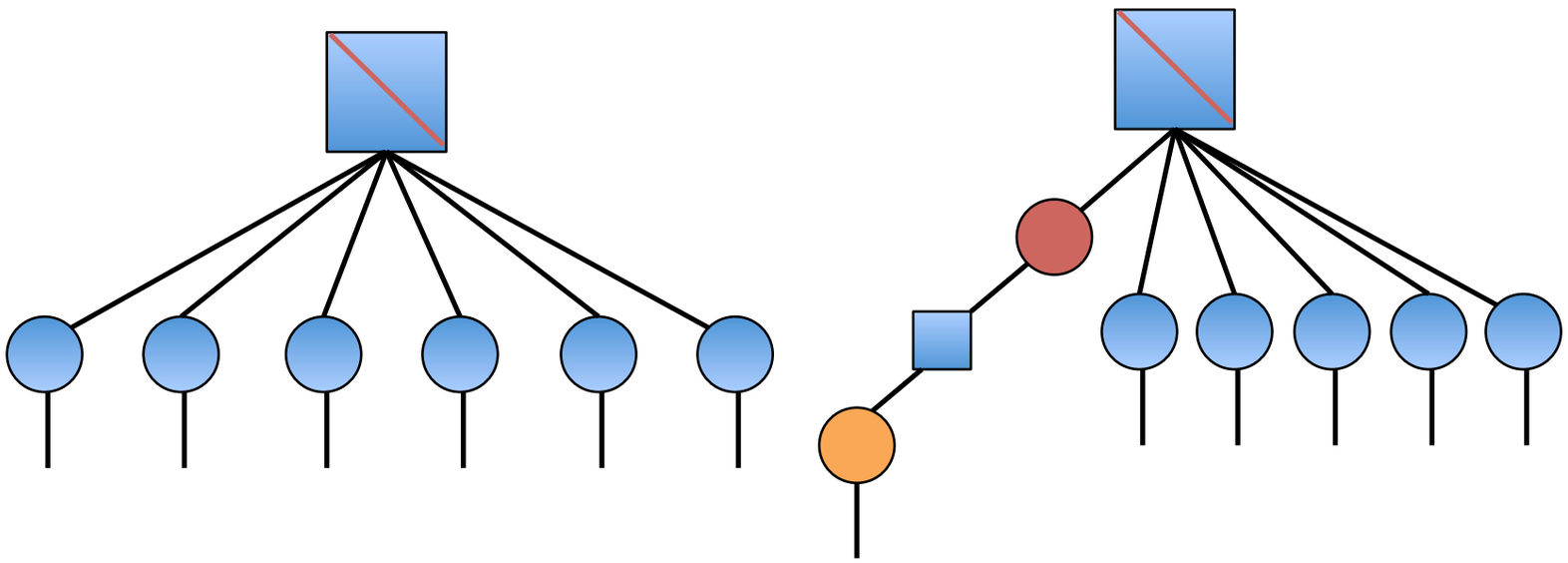


Solved simply using pseudo-inverse

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

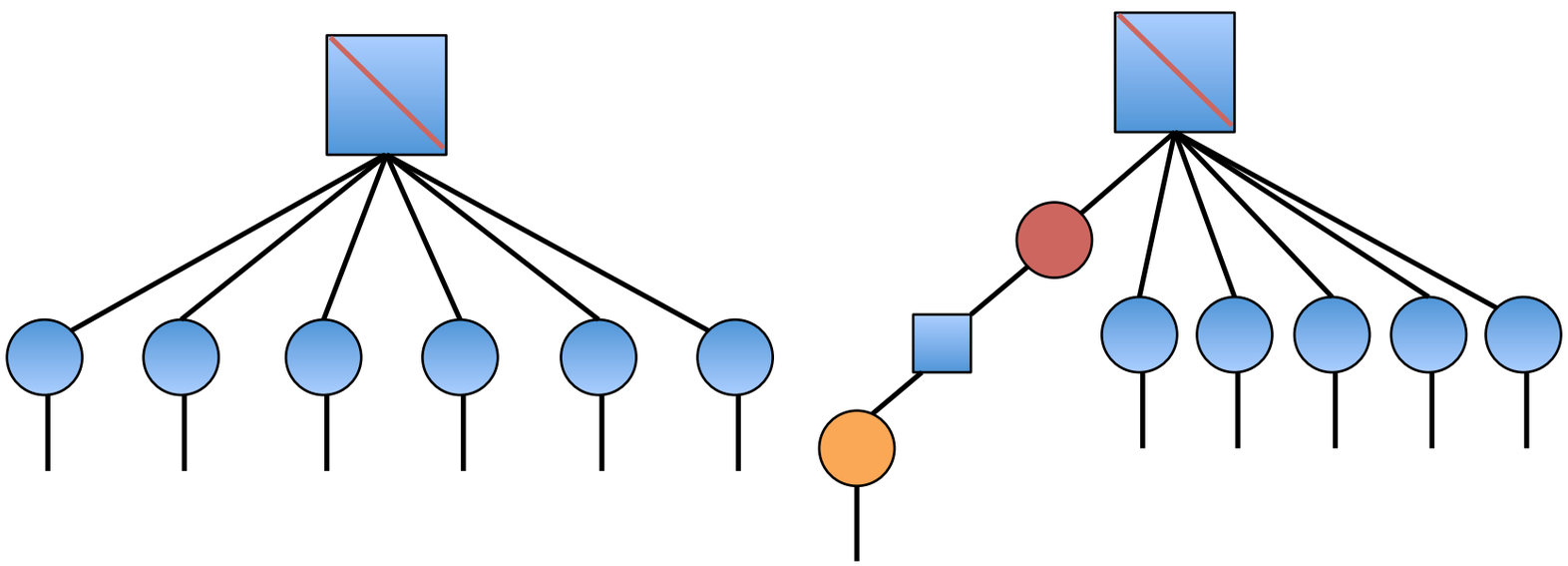
From CP to Tucker

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



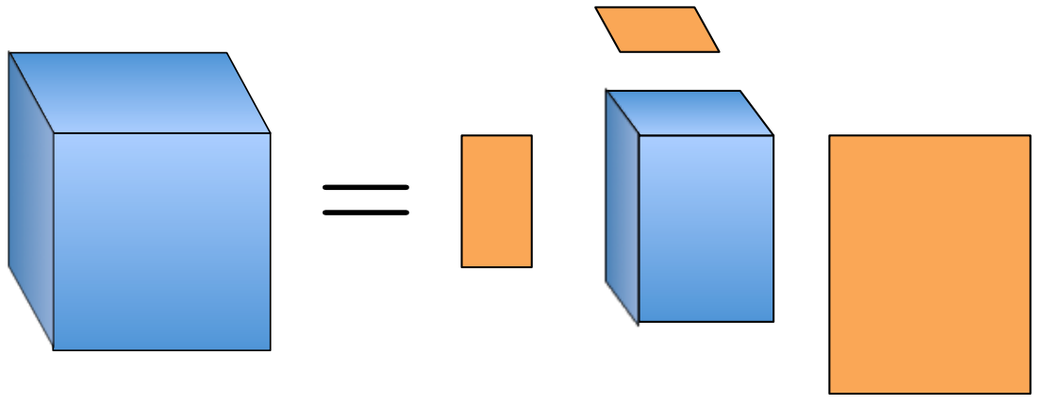
From CP to Tucker

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



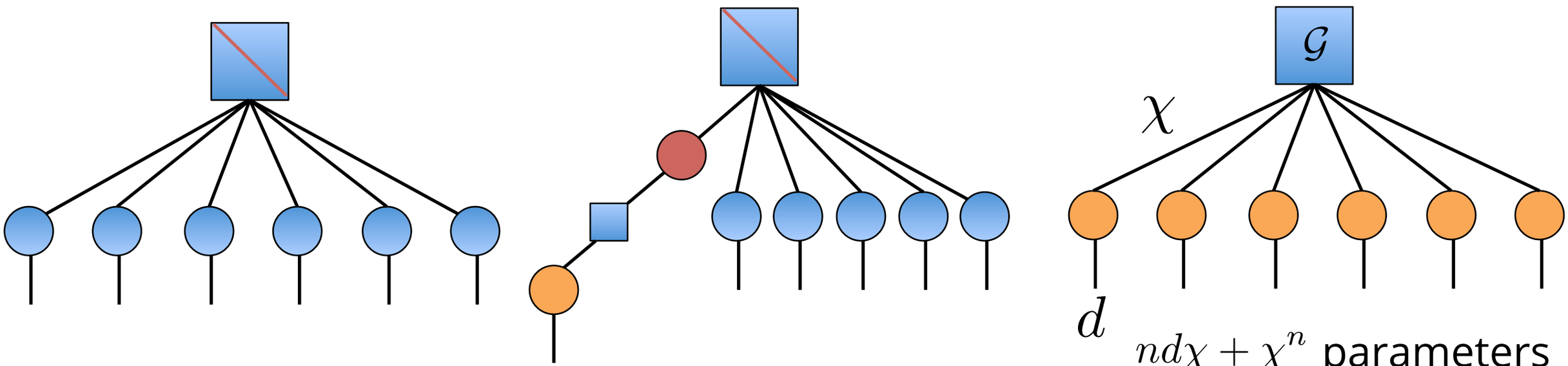
this leads to the *Tucker Decomposition*

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



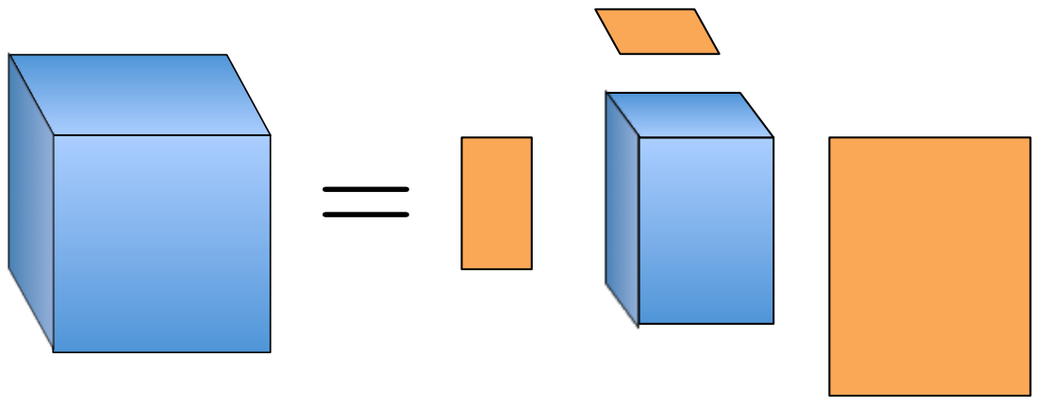
From CP to Tucker

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*

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



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

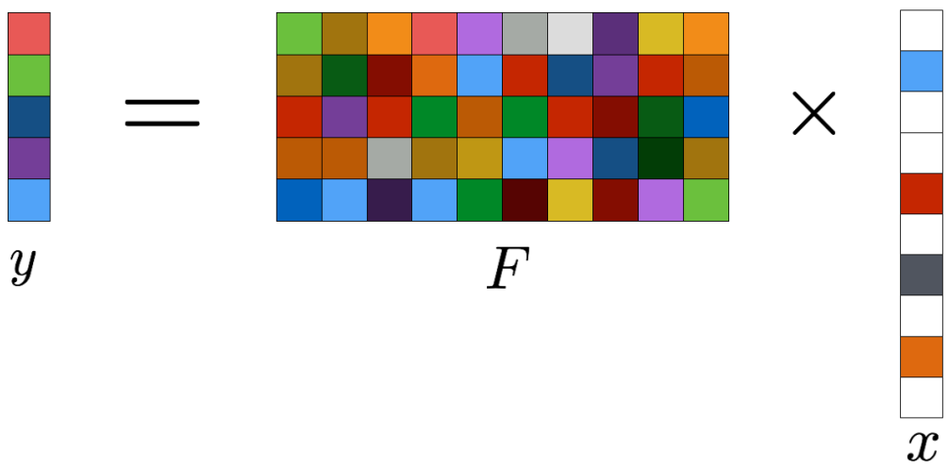
K-means clustering

anomaly detection

matrix completion

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

high-order compressed sensing (using compositional dictionary)



Applications of the Tucker decomposition

Data compression

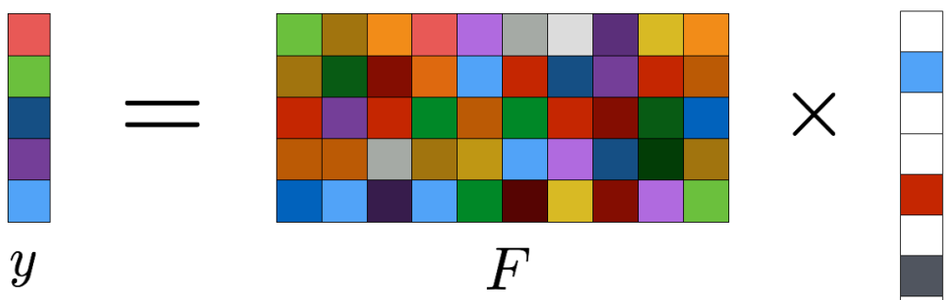
K-means clustering

anomaly detection

matrix completion

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

high-order compressed sensing (using compositional dictionary)



$$\mathbf{F} = \mathbf{F}^{(1)} \otimes \mathbf{F}^{(2)} \otimes \cdots \otimes \mathbf{F}^{(n)}$$

Applications of the Tucker decomposition

Data compression

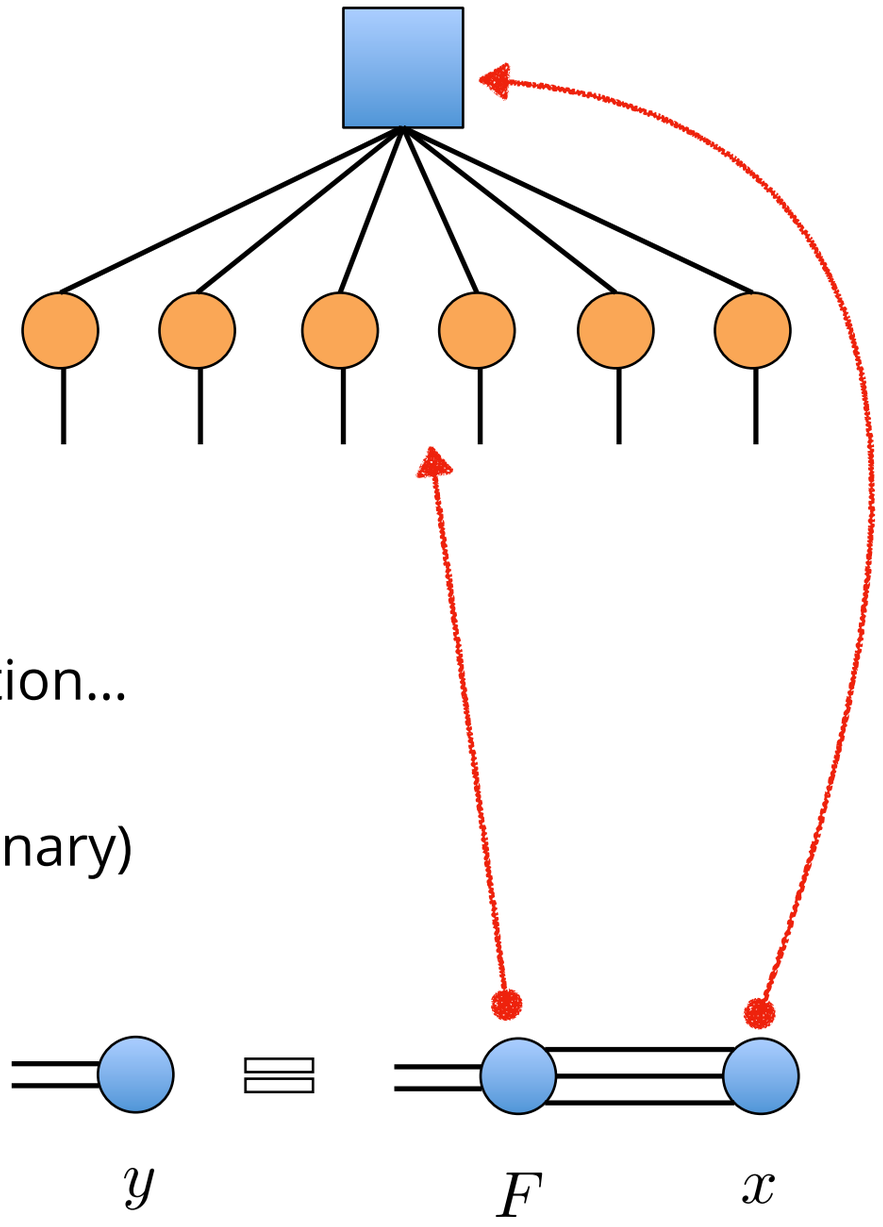
K-means clustering

anomaly detection

matrix completion

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

high-order compressed sensing (using compositional dictionary)

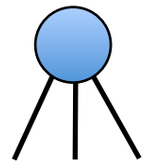


$$\mathbf{F} = \mathbf{F}^{(1)} \otimes \mathbf{F}^{(2)} \otimes \cdots \otimes \mathbf{F}^{(n)}$$

High order singular value decomposition

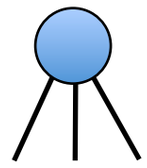
High order singular value decomposition

given a raw tensor $\mathcal{A}$

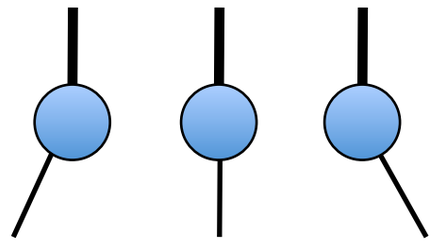


High order singular value decomposition

given a raw tensor $\mathcal{A}$

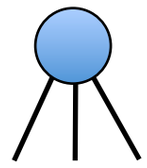


unfold to matrix at each mode

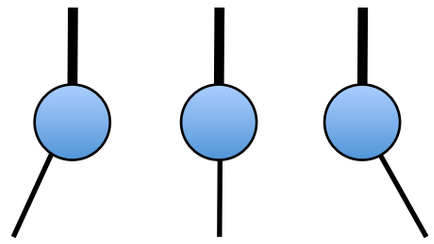


High order singular value decomposition

given a raw tensor $\mathcal{A}$

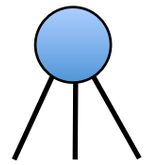


unfold to matrix at each mode

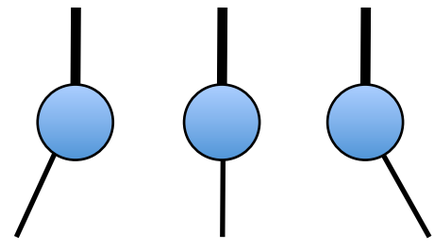


High order singular value decomposition

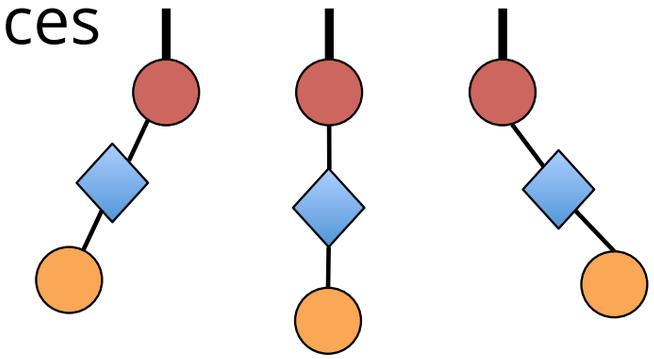
given a raw tensor $\mathcal{A}$



unfold to matrix at each mode

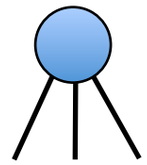


do SVD on obtained matrices

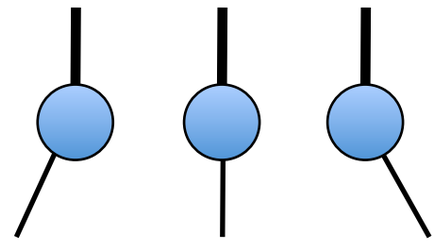


High order singular value decomposition

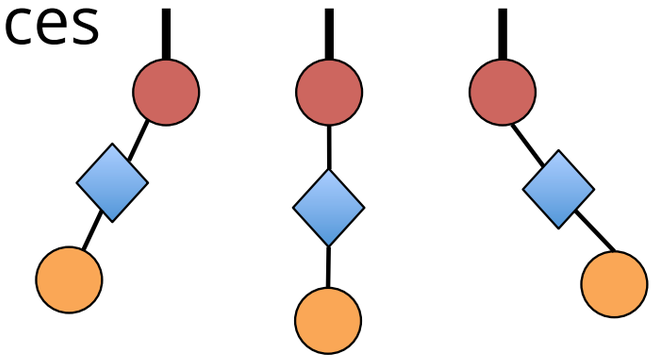
given a raw tensor $\mathcal{A}$



unfold to matrix at each mode



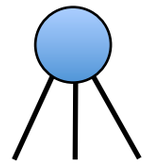
do SVD on obtained matrices



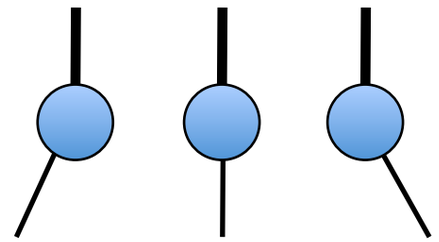
contract A to the inverse of the obtained column orthogonal matrices, to obtain the core $\mathcal{G}$.

High order singular value decomposition

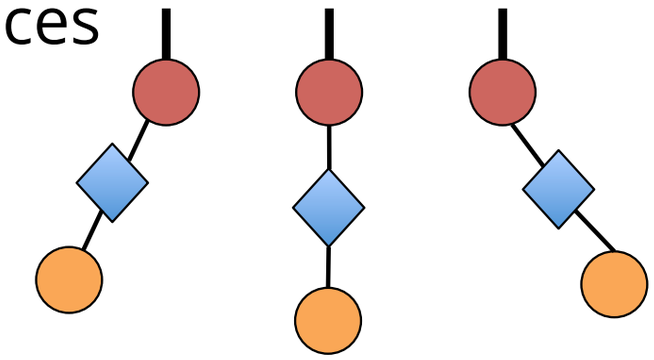
given a raw tensor $\mathcal{A}$



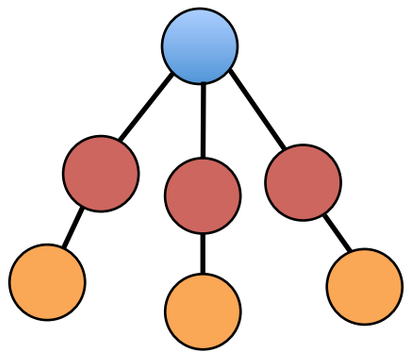
unfold to matrix at each mode



do SVD on obtained matrices

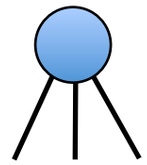


contract A to the inverse of the obtained column orthogonal matrices, to obtain the core $\mathcal{G}$.

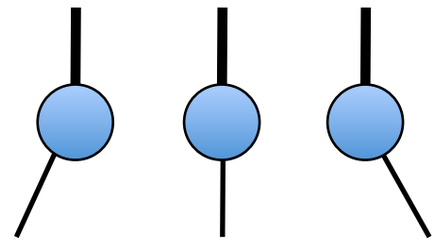


High order singular value decomposition

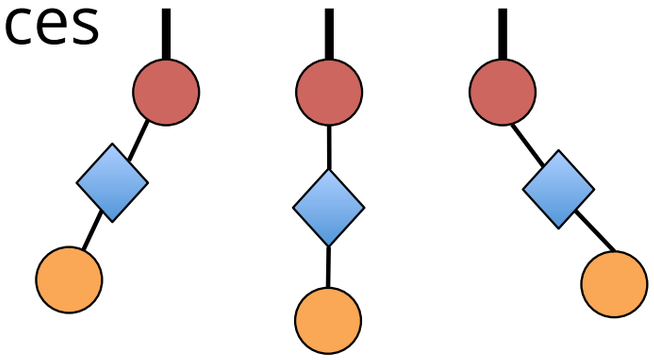
given a raw tensor $\mathcal{A}$



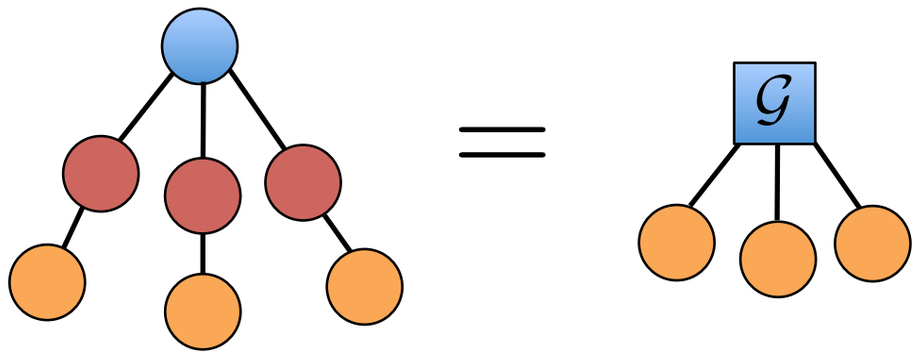
unfold to matrix at each mode



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

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

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.

Lathauwer, Moor, Vandewalle. SIAM J. Matrix Anal. Appl. 21, 1253 (2000)
Elden, Savas, SIAM J. Matrix Anal. Appl. 31, 248 (2009)

Improving HOSVD

HOSVD only gives a quasi-best approximation

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

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*

Lathauwer, Moor, Vandewalle. SIAM J. Matrix Anal. Appl. 21, 1253 (2000)
Elden, Savas, SIAM J. Matrix Anal. Appl. 31, 248 (2009)

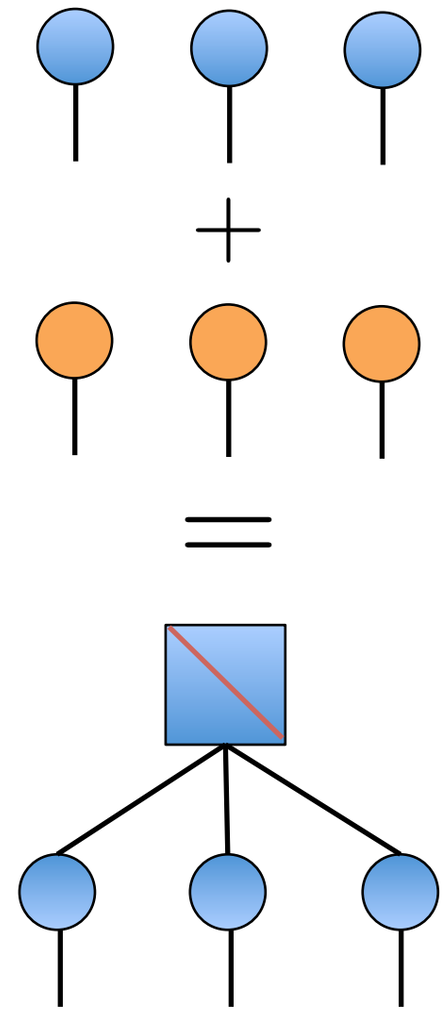
From CP to MPS

Summing two rank-1 tensors

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

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

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



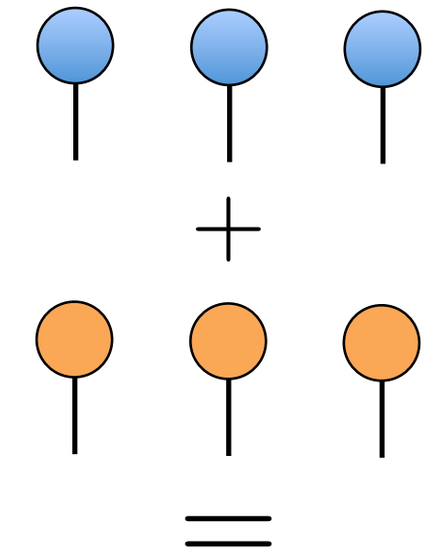
From CP to MPS

Summing two rank-1 tensors

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

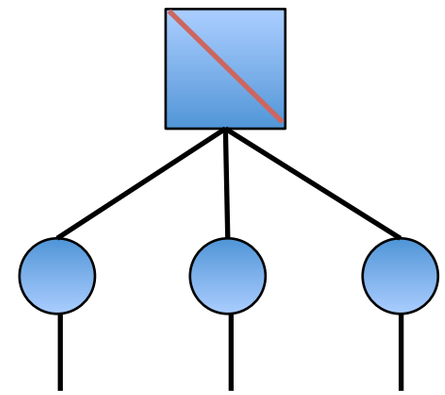
$$\mathcal{B} = \mathbf{b}^{(1)} \circ \mathbf{b}^{(2)} \circ \mathbf{b}^{(3)}$$

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



$$c_{s_1,s_2,s_3} = a^{(1)}_{s_1} a^{(2)}_{s_2} a^{(3)}_{s_3} + b^{(1)}_{s_1} b^{(2)}_{s_2} b^{(3)}_{s_3}$$

$$= \begin{pmatrix} a^{(1)}_{s_1} & b^{(1)}_{s_1} \end{pmatrix} \begin{pmatrix} a^{(2)}_{s_2} & 0 \\ 0 & b^{(2)}_{s_2} \end{pmatrix} \begin{pmatrix} a^{(3)}_{s_3} \\ b^{(3)}_{s_3} \end{pmatrix}$$



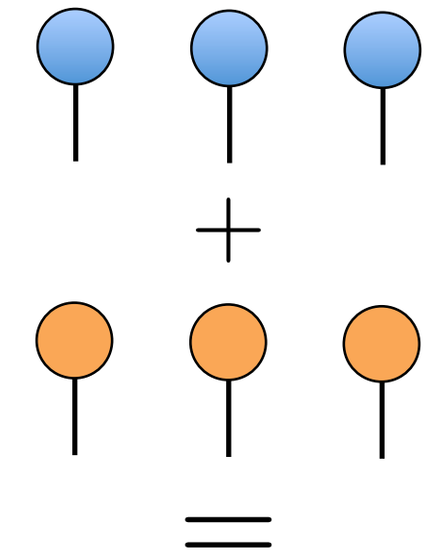
From CP to MPS

Summing two rank-1 tensors

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

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

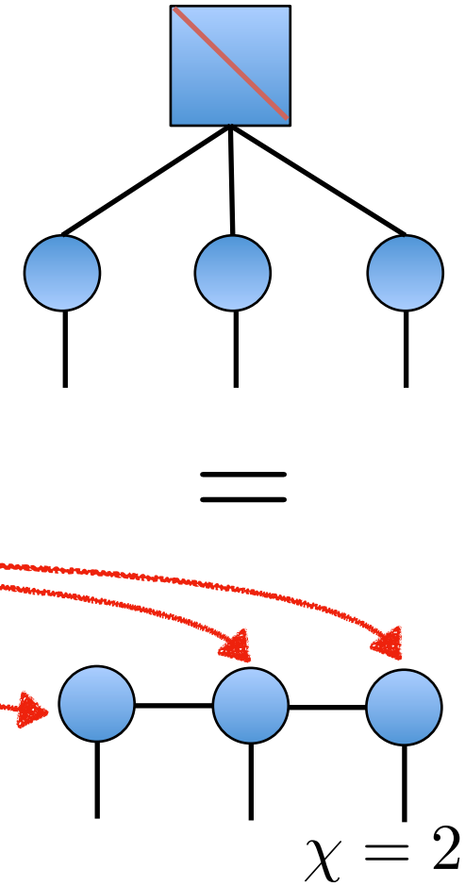
$$\mathcal{C} = \mathcal{A} + \mathcal{B}$$



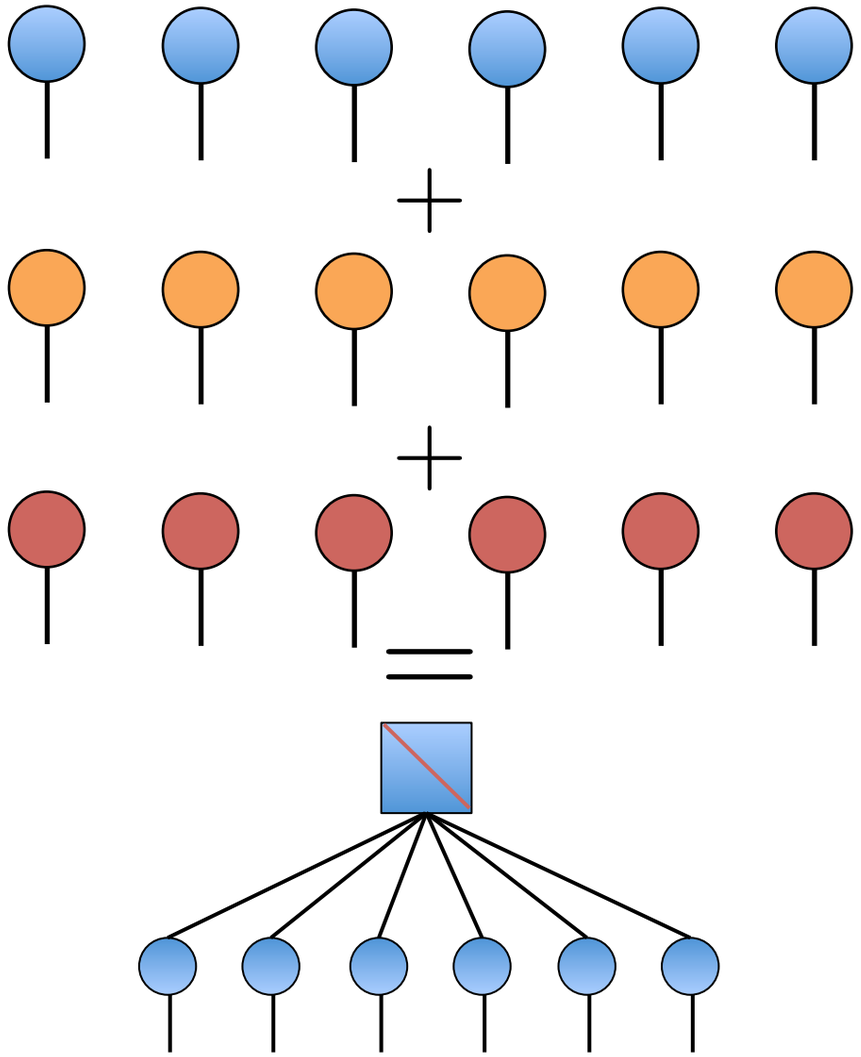
$$c_{s_1,s_2,s_3} = a^{(1)}_{s_1} a^{(2)}_{s_2} a^{(3)}_{s_3} + b^{(1)}_{s_1} b^{(2)}_{s_2} b^{(3)}_{s_3}$$

$$= \begin{pmatrix} a^{(1)}_{s_1} & b^{(1)}_{s_1} \end{pmatrix} \begin{pmatrix} a^{(2)}_{s_2} & 0 \\ 0 & b^{(2)}_{s_2} \end{pmatrix} \begin{pmatrix} a^{(3)}_{s_3} \\ b^{(3)}_{s_3} \end{pmatrix}$$

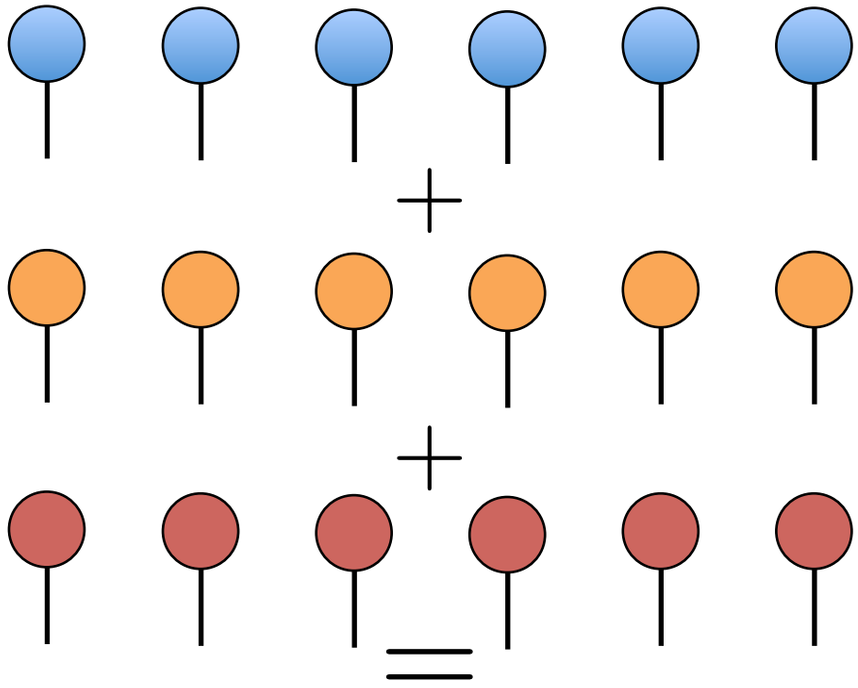
$$\mathbf{C}^{(1)}(s_1,:) \quad \mathcal{C}^{(2)}(:,s_2,:) \quad \mathbf{C}^{(3)}(:,s_3)$$

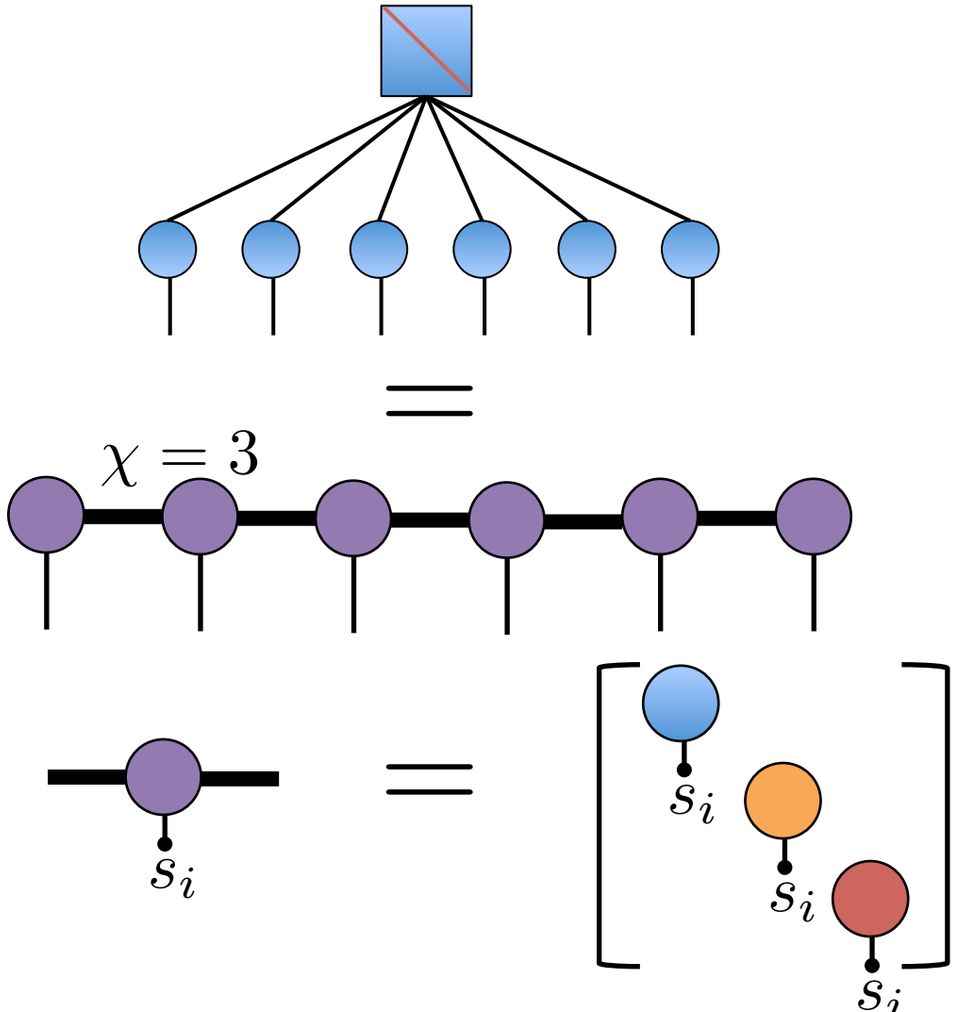


From CP to MPS



From CP to MPS

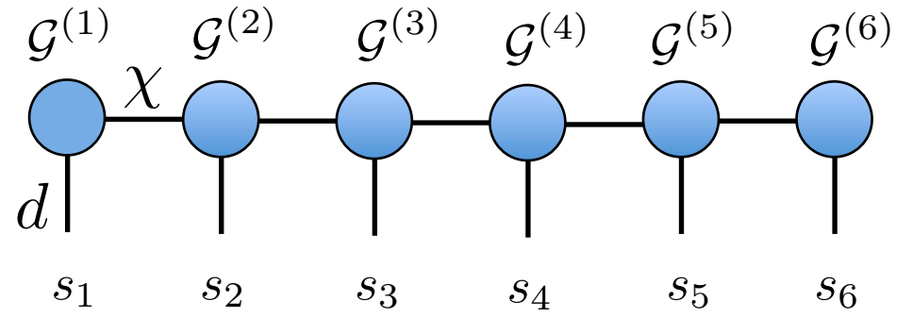




Matrix Product States and Matrix Product Operator

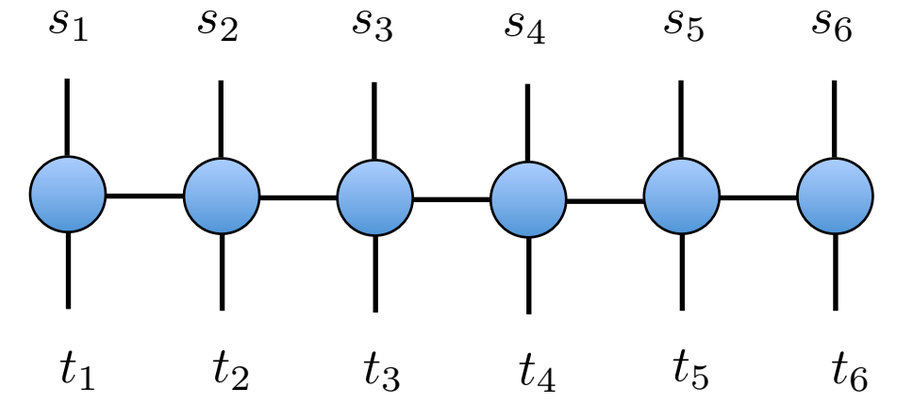
$$\mathcal{A} = \langle \mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \cdots, \mathcal{G}^{(n)} \rangle$$
$$= \mathcal{G}^{(1)} \times_1 \mathcal{G}^{(2)} \times_1 \cdots \times_1 \mathcal{G}^{(n)}$$

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



$$\approx nd\chi^2 \text{ parameters}$$

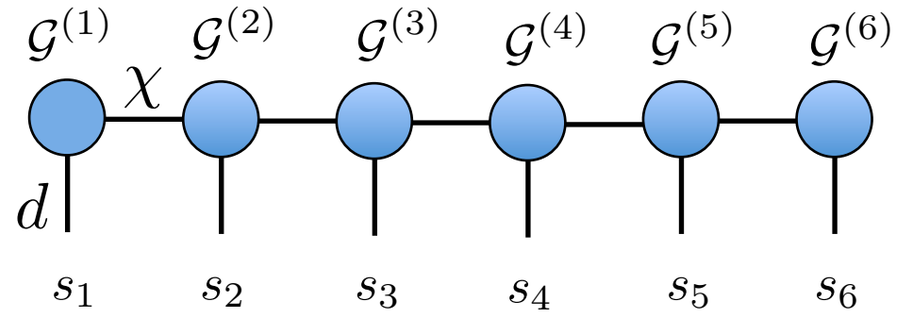
$$a_{s_1,s_2,\cdots,s_n} = \mathbf{G}^{(1)}_{s_1,t_1} \mathbf{G}^{(2)}_{s_2,t_2} \cdots \mathbf{G}^{(n)}_{s_n,t_n}$$



Matrix Product States and Matrix Product Operator

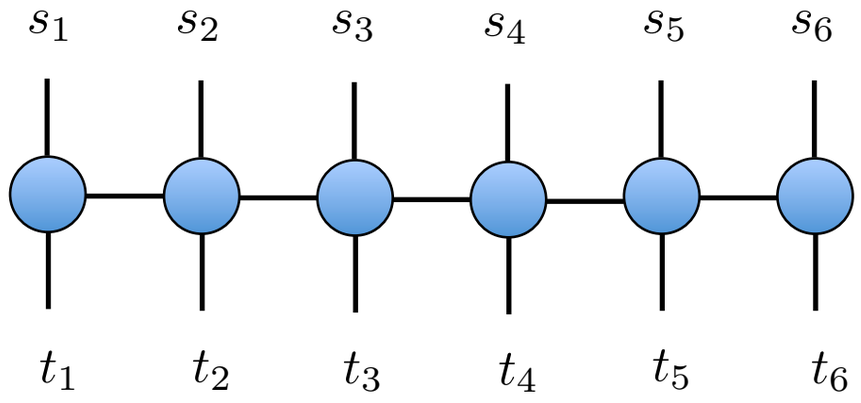
$$\mathcal{A} = \langle \mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \cdots, \mathcal{G}^{(n)} \rangle$$
$$= \mathcal{G}^{(1)} \times_1 \mathcal{G}^{(2)} \times_1 \cdots \times_1 \mathcal{G}^{(n)}$$

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



$$\approx nd\chi^2 \text{ parameters}$$

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

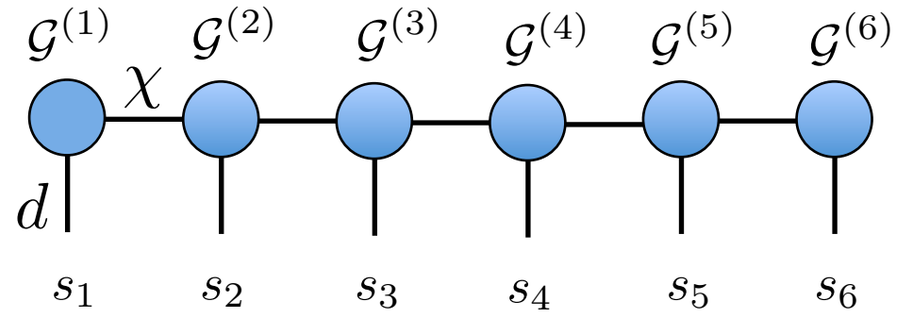


with χ=1, MPS is a rank-one tensor

Matrix Product States and Matrix Product Operator

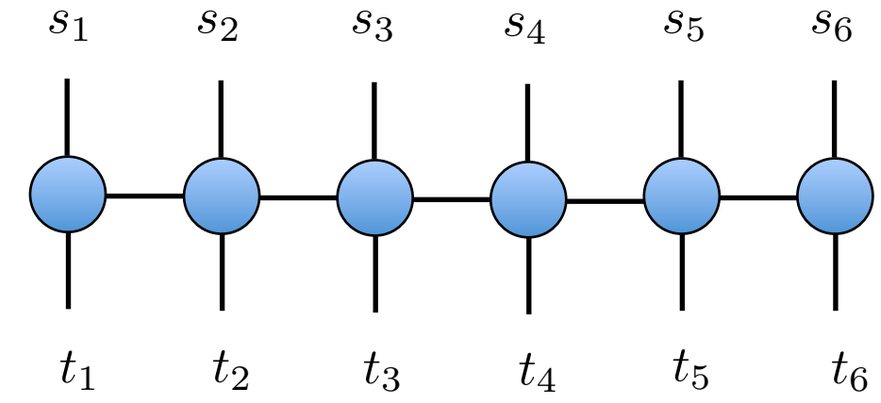
$$\mathcal{A} = \langle \mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \cdots, \mathcal{G}^{(n)} \rangle$$
$$= \mathcal{G}^{(1)} \times_1 \mathcal{G}^{(2)} \times_1 \cdots \times_1 \mathcal{G}^{(n)}$$

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



$$\approx nd\chi^2 \text{ parameters}$$

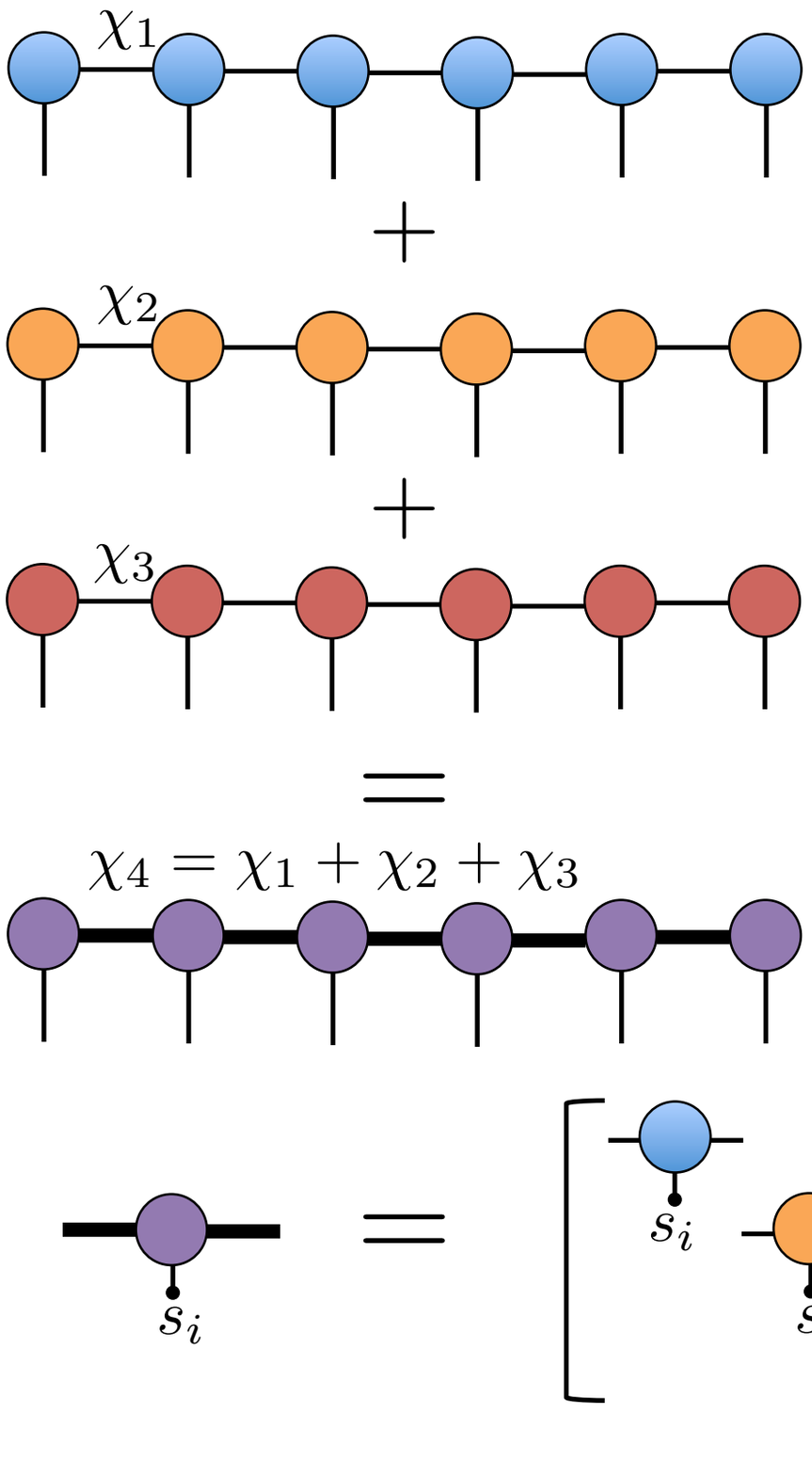
$$a_{s_1,s_2,\cdots,s_n} = \mathbf{G}^{(1)}_{s_1,t_1} \mathbf{G}^{(2)}_{s_2,t_2} \cdots \mathbf{G}^{(n)}_{s_n,t_n}$$



with χ=1, MPS is a rank-one tensor

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

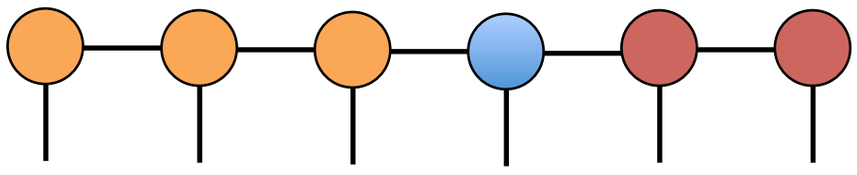
Summing MPSes

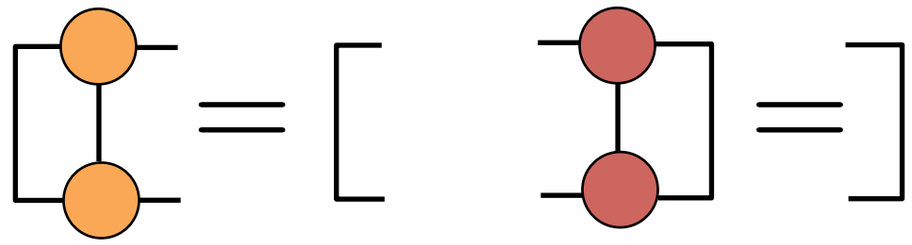


Canonical forms of MPS

Canonical forms of MPS

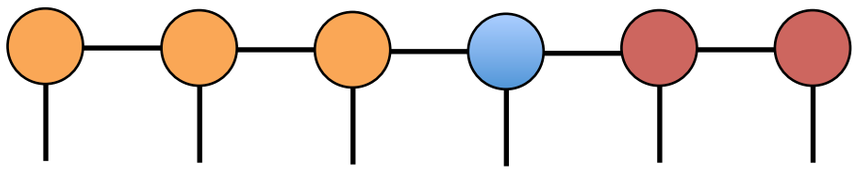
Analogous to the Tucker decomposition and HOSVDs, MPS has the benefits of orthogonality.

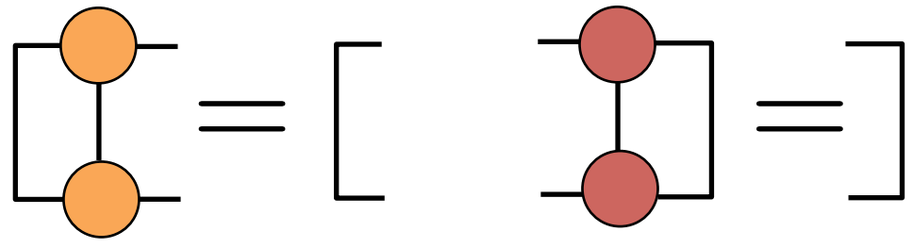




Canonical forms of MPS

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

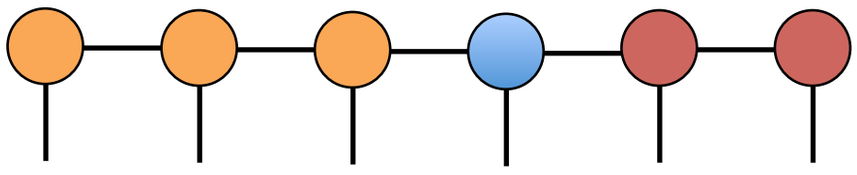


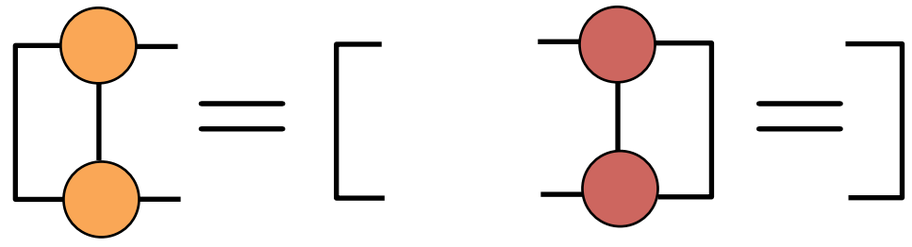


Benefits

Canonical forms of MPS

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



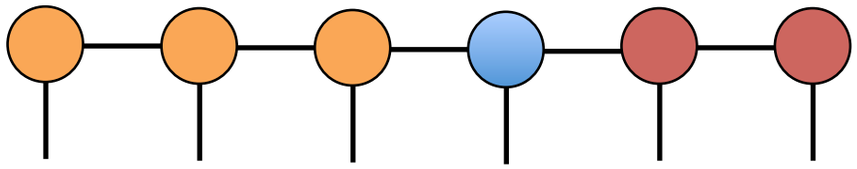


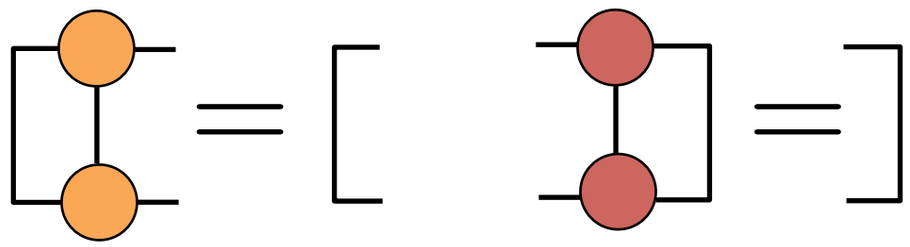
Benefits

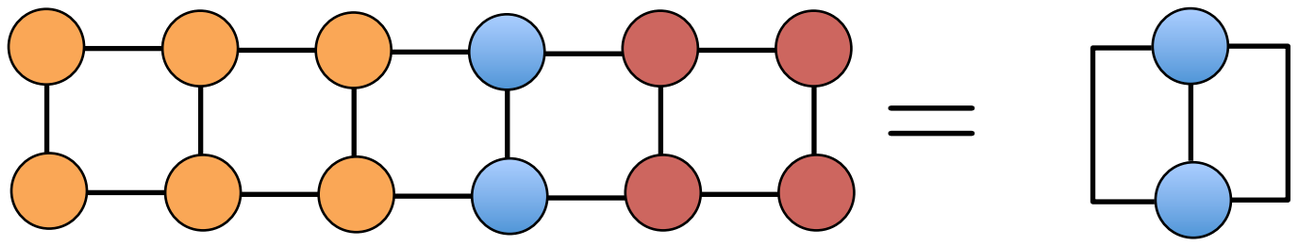
- Fixed gauge, no ambiguity

Canonical forms of MPS

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





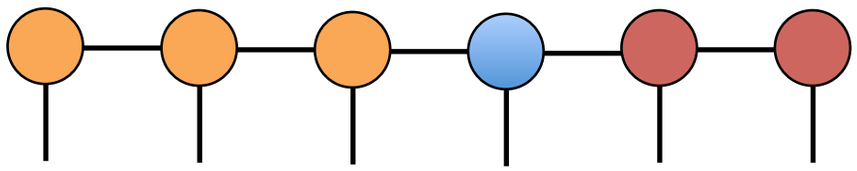


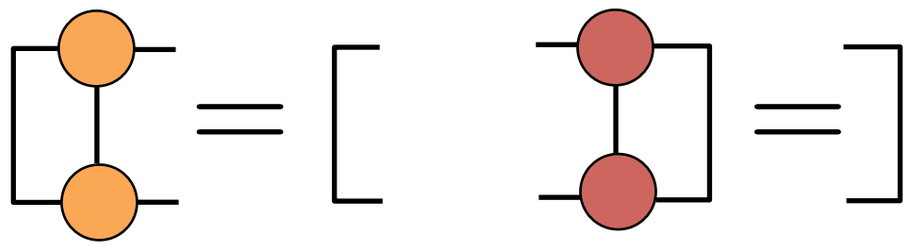
Benefits

- Fixed gauge, no ambiguity
- Easy norm computation

Canonical forms of MPS

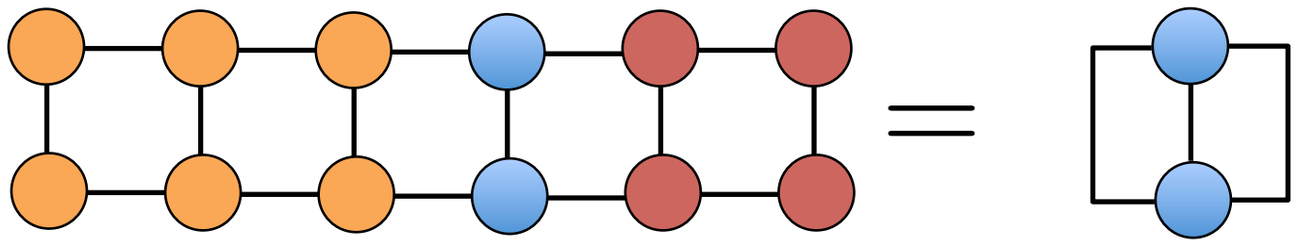
Analogous to the Tucker decomposition and HOSVDs, MPS has the benefits of orthogonality.

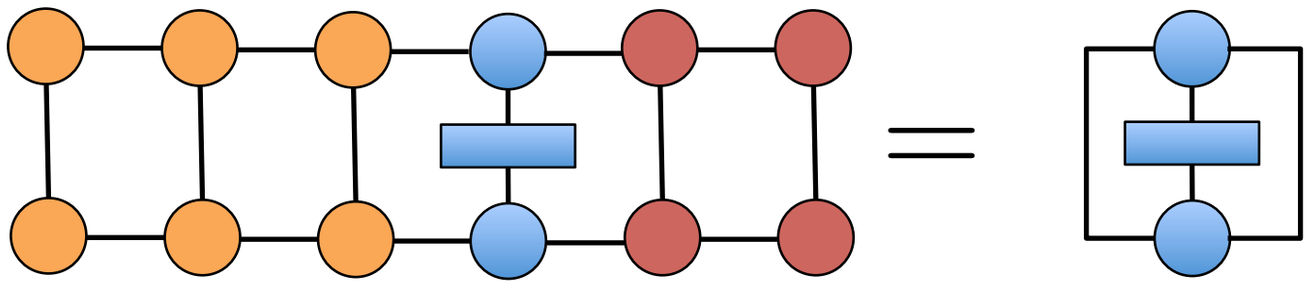


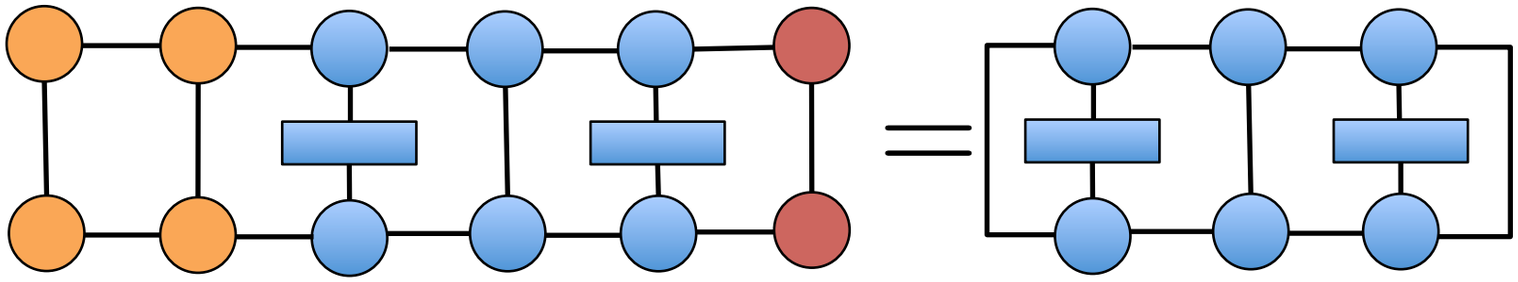


Benefits

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

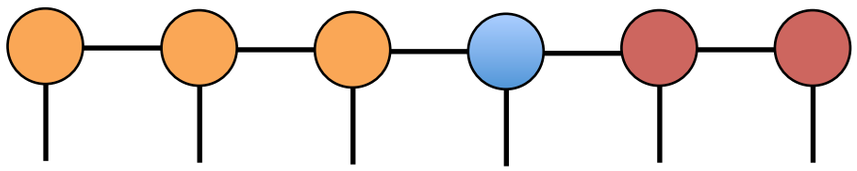


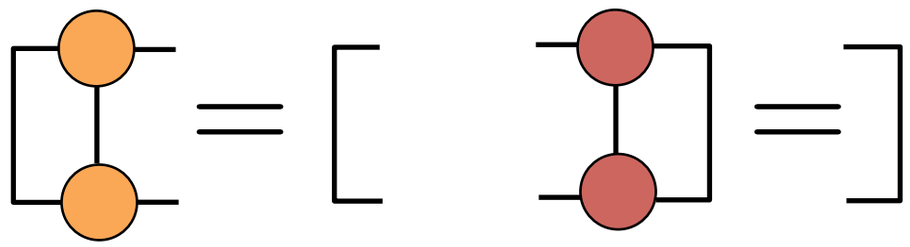




Canonical forms of MPS

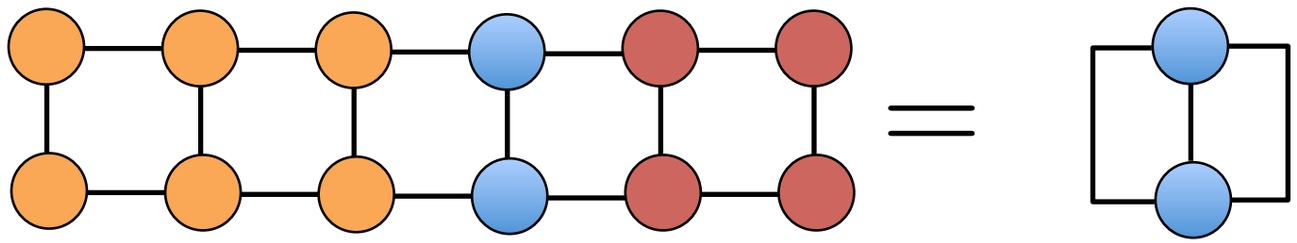
Analogous to the Tucker decomposition and HOSVDs, MPS has the benefits of orthogonality.

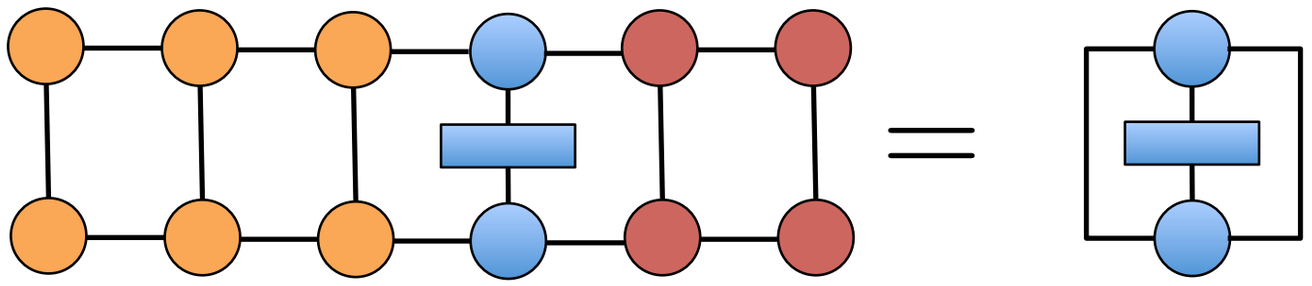


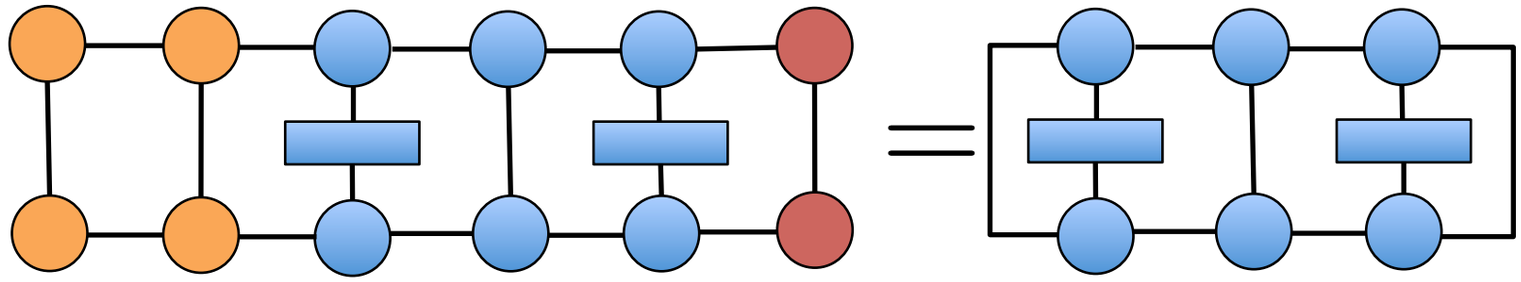


Benefits

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



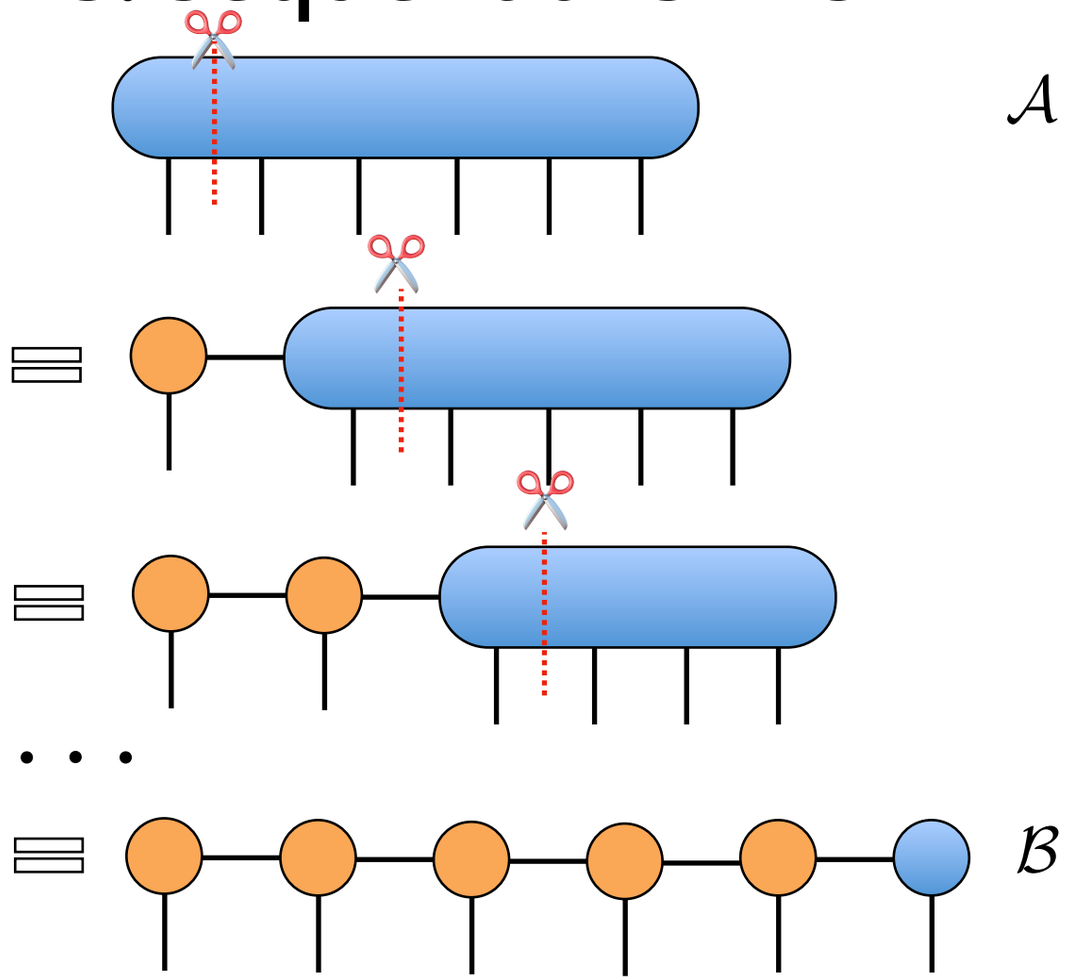




Convert a raw tensor to a MPS: sequential SVDs

Convert a raw tensor to a MPS: sequential SVDs

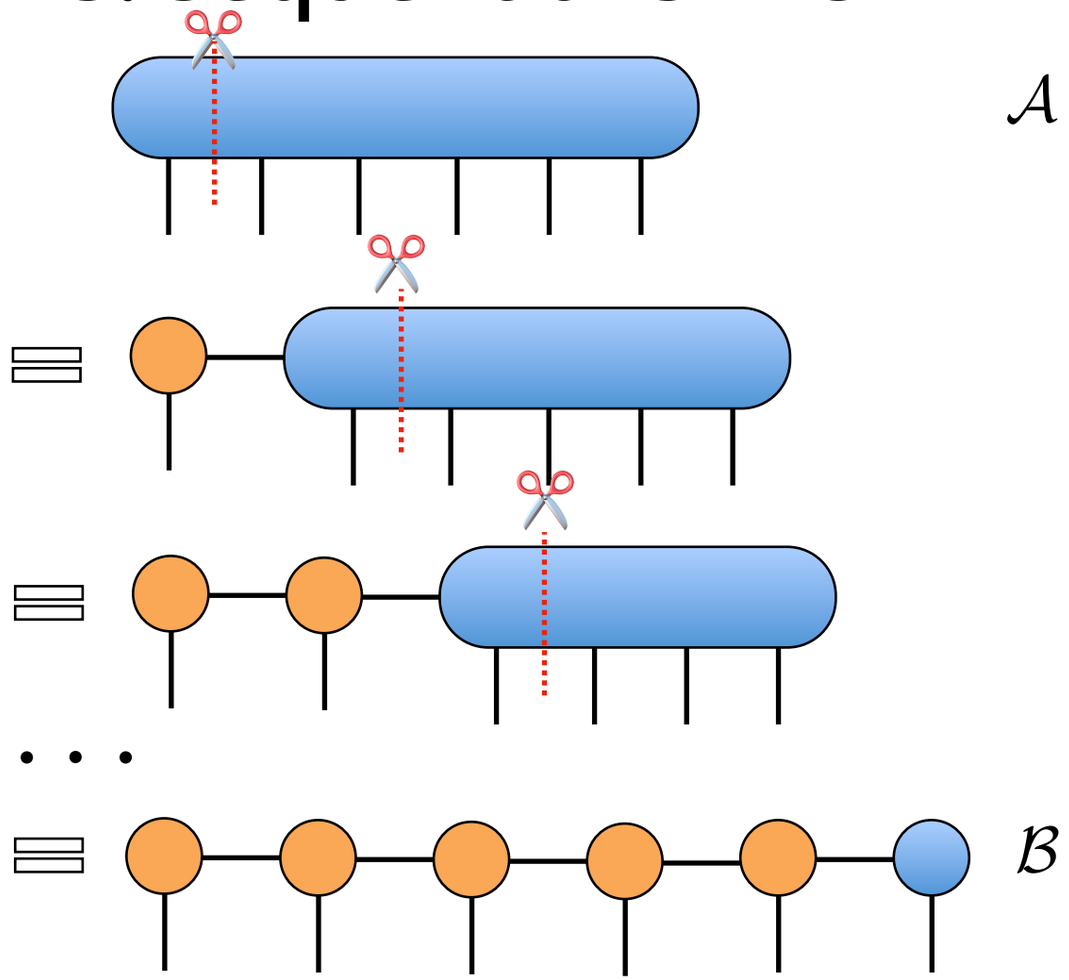
analogous to HOSVD, but processes every mode *one by one*



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



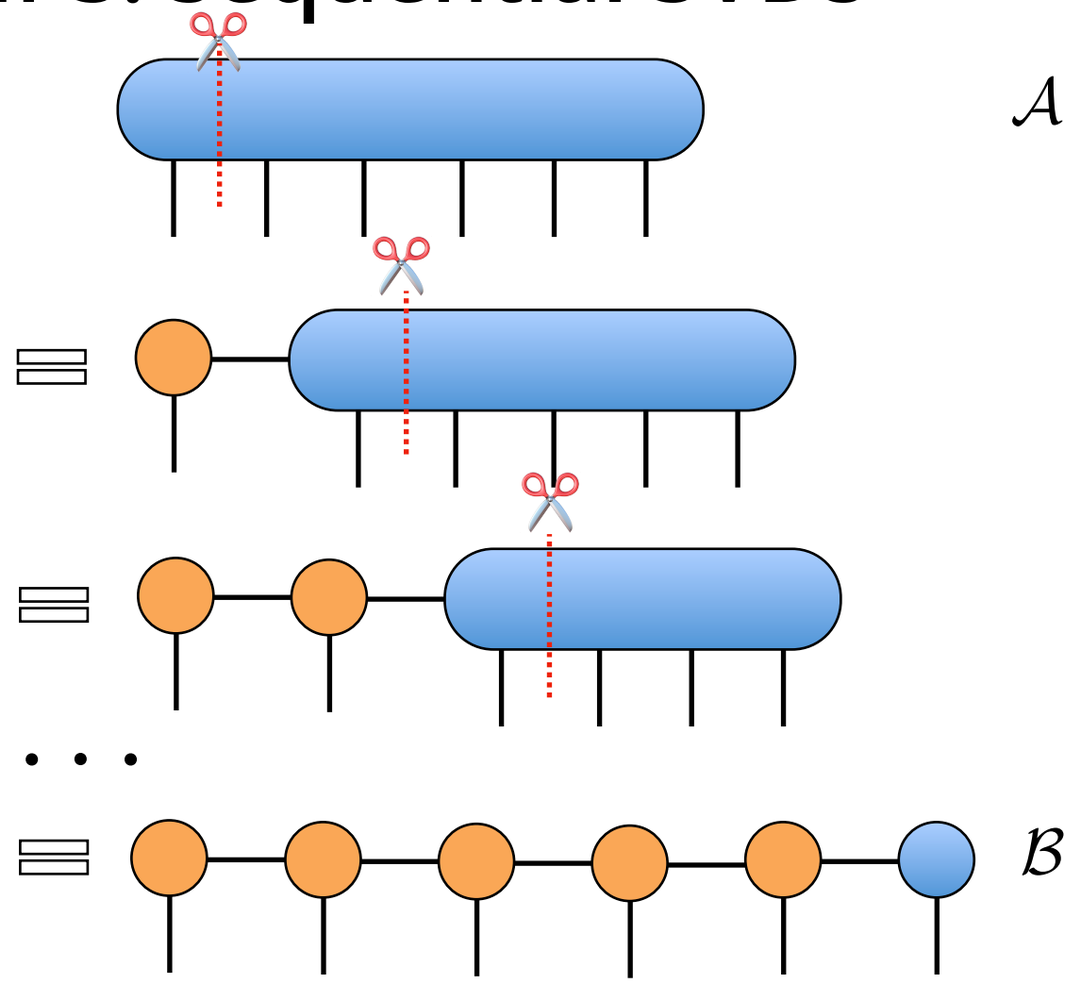
Convert a raw tensor to a MPS: sequential SVDs

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

matrices encountered during SVDs are larger than HOSVD

error made in compression

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



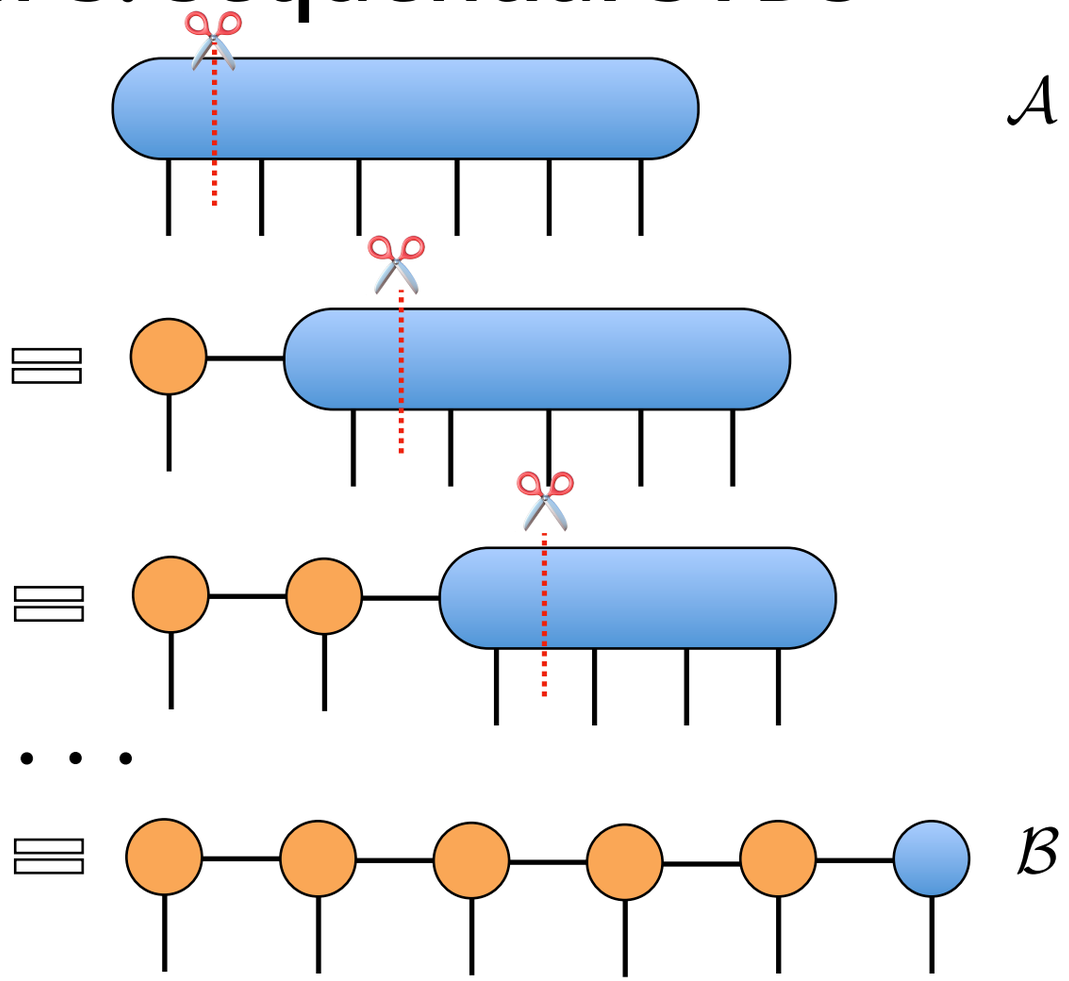
Convert a raw tensor to a MPS: sequential SVDs

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

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

Oseledets, SIAM J. Sci. Comput. 33, 2295 (2011)

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

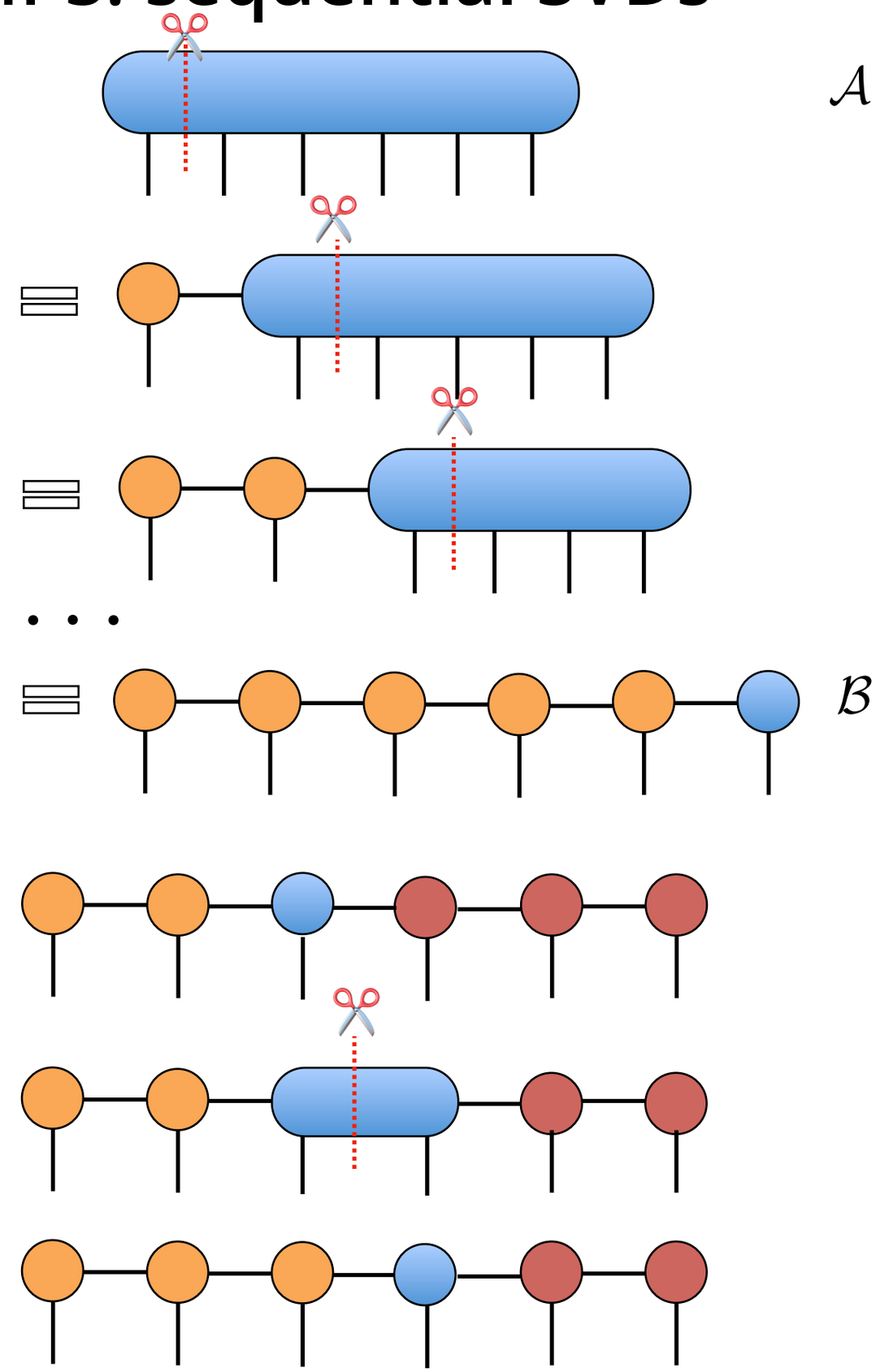
$$\|\mathcal{A} - \mathcal{B}\|_F \leq \sqrt{\sum_{i=1}^{n} \epsilon_i^2}$$

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)

Oseledets, SIAM J. Sci. Comput. 33, 2295 (2011)



Application: compressing neural networks

Application: compressing neural networks

Deep neural networks are most popular machine learning models.

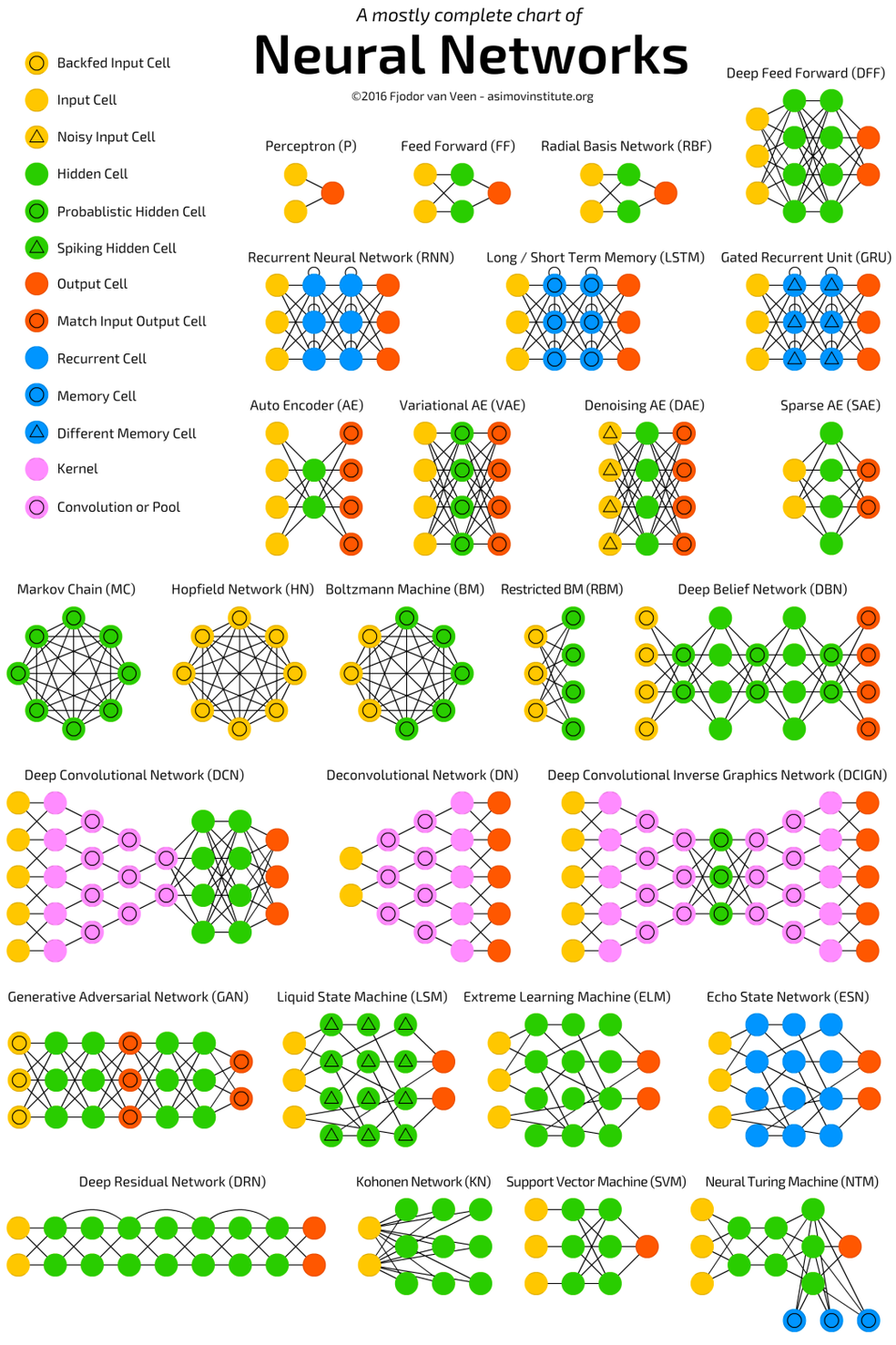
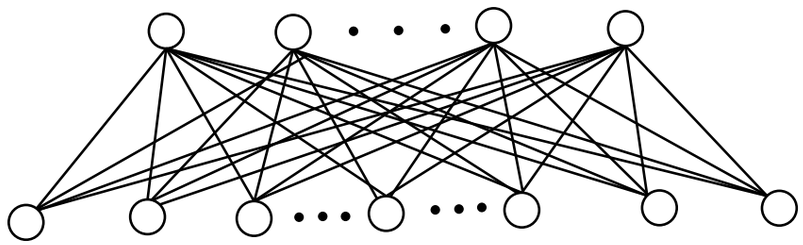


Image courtesy: Fjodor van Veen

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*.



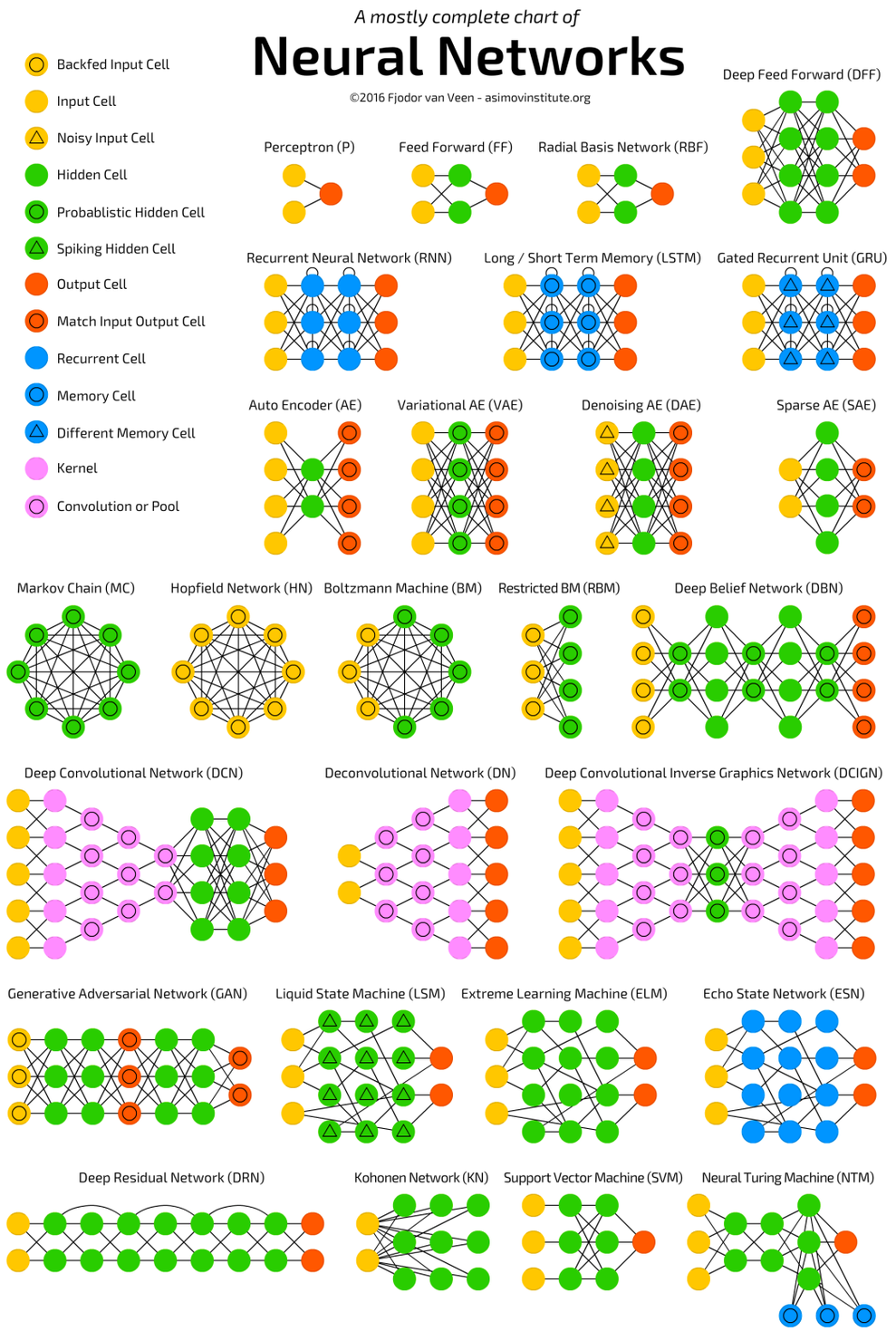
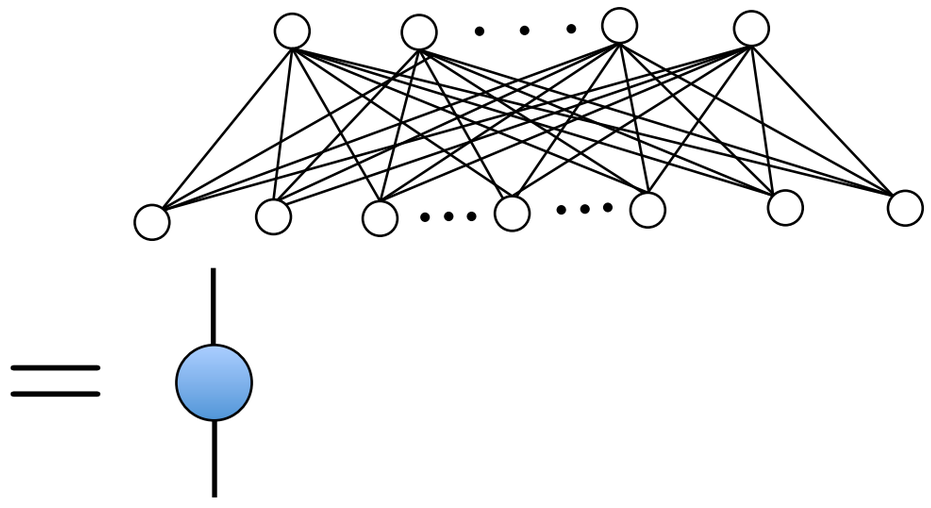


Image courtesy: Fjodor van Veen

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*.



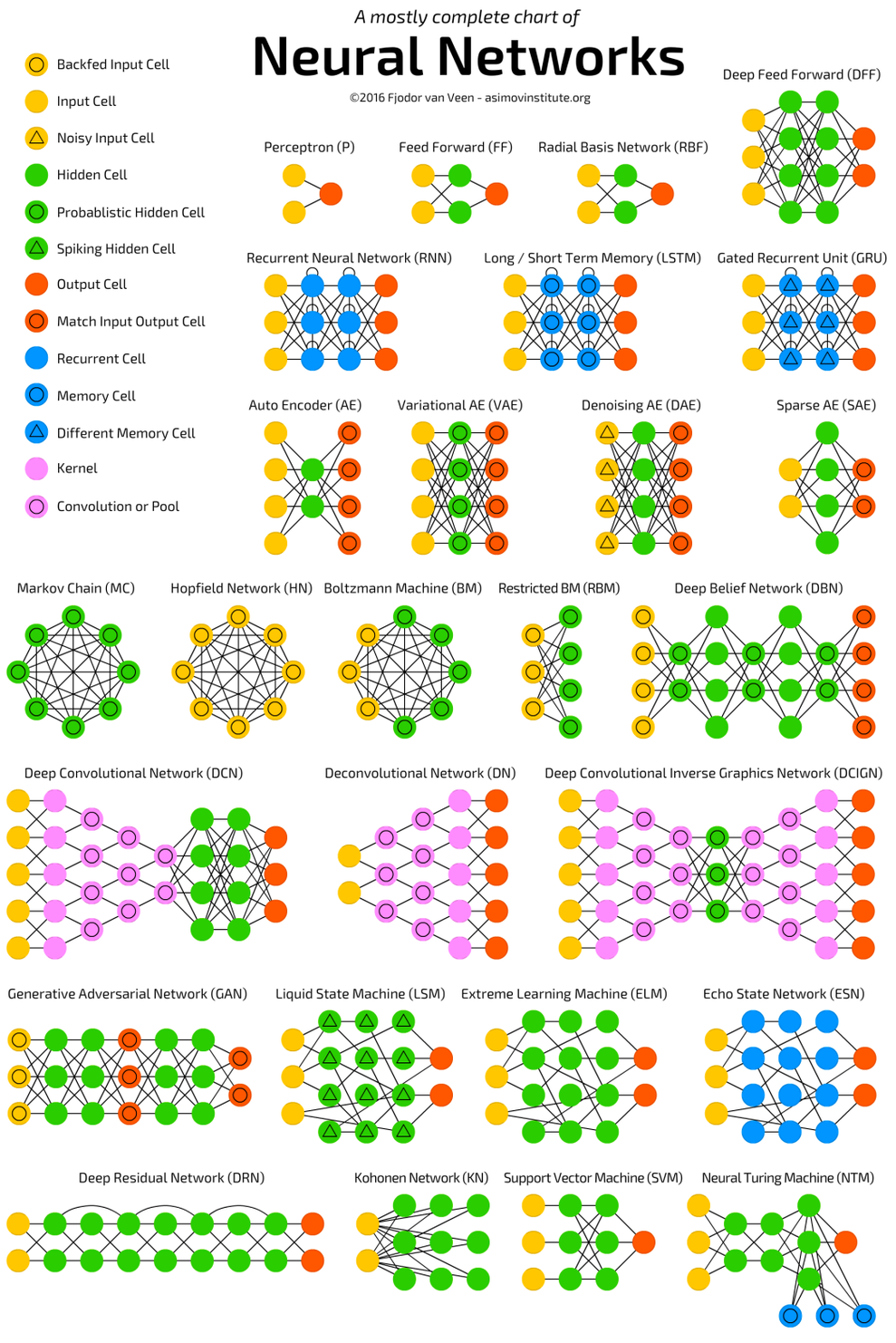
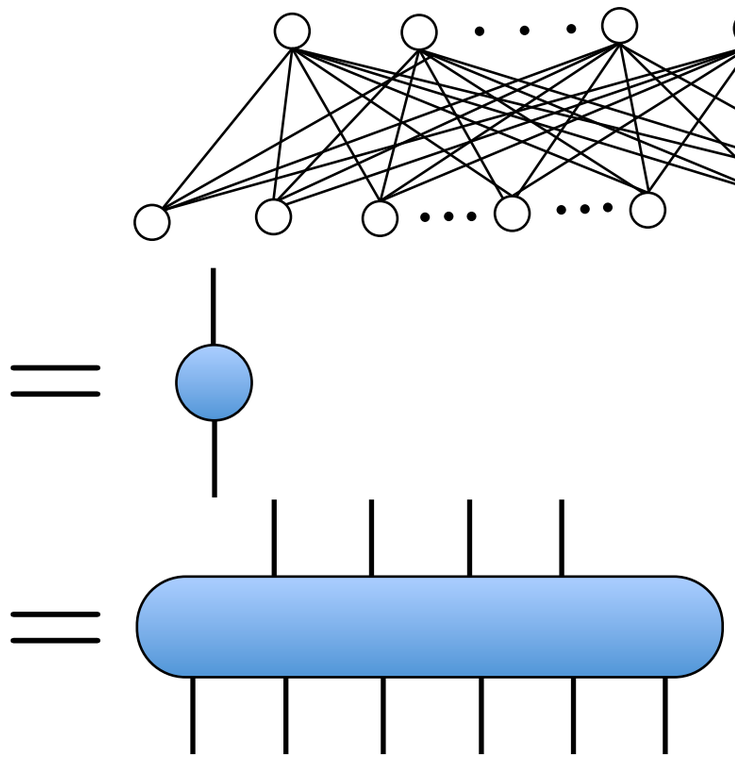


Image courtesy: Fjodor van Veen

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*.



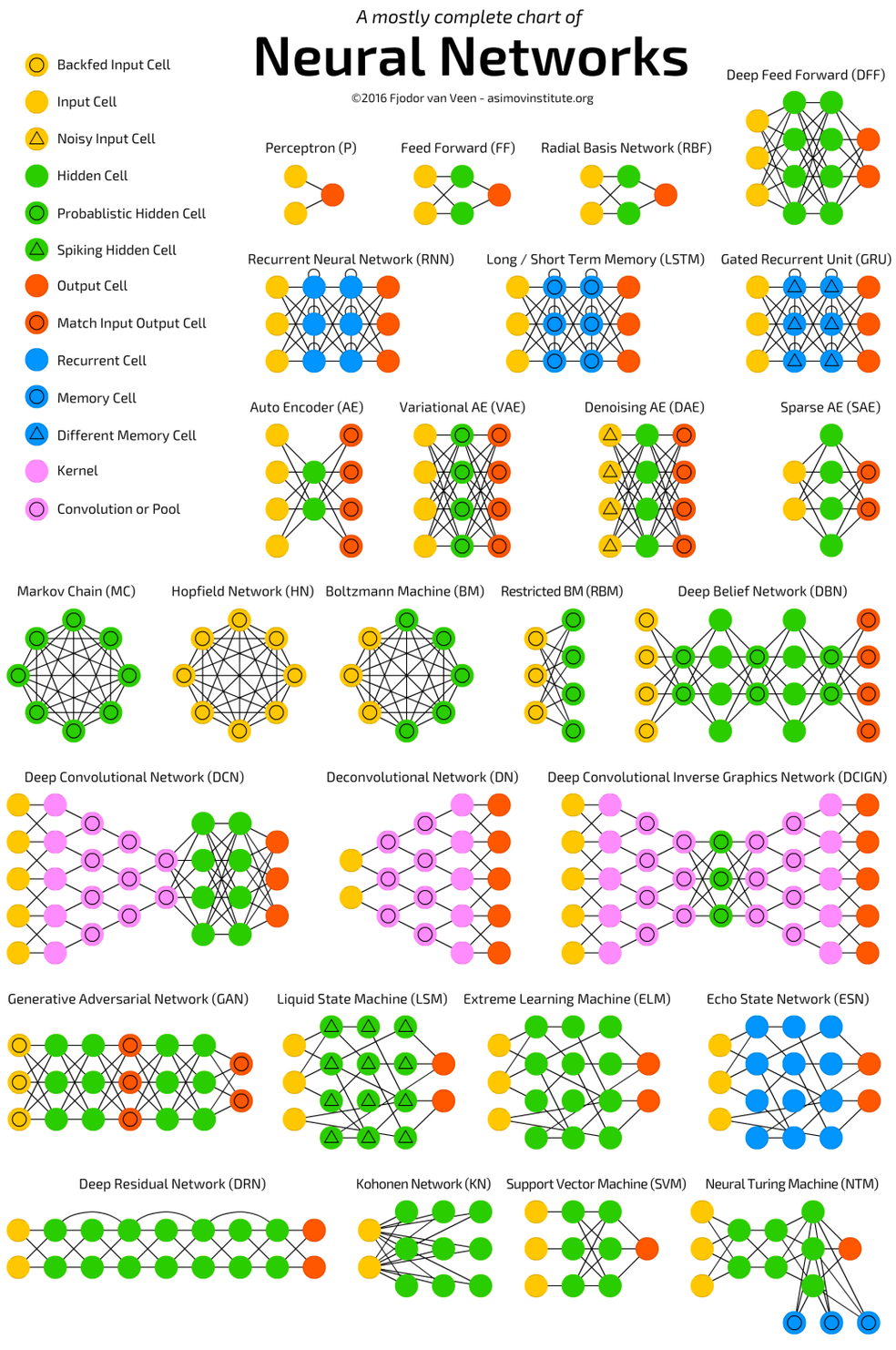
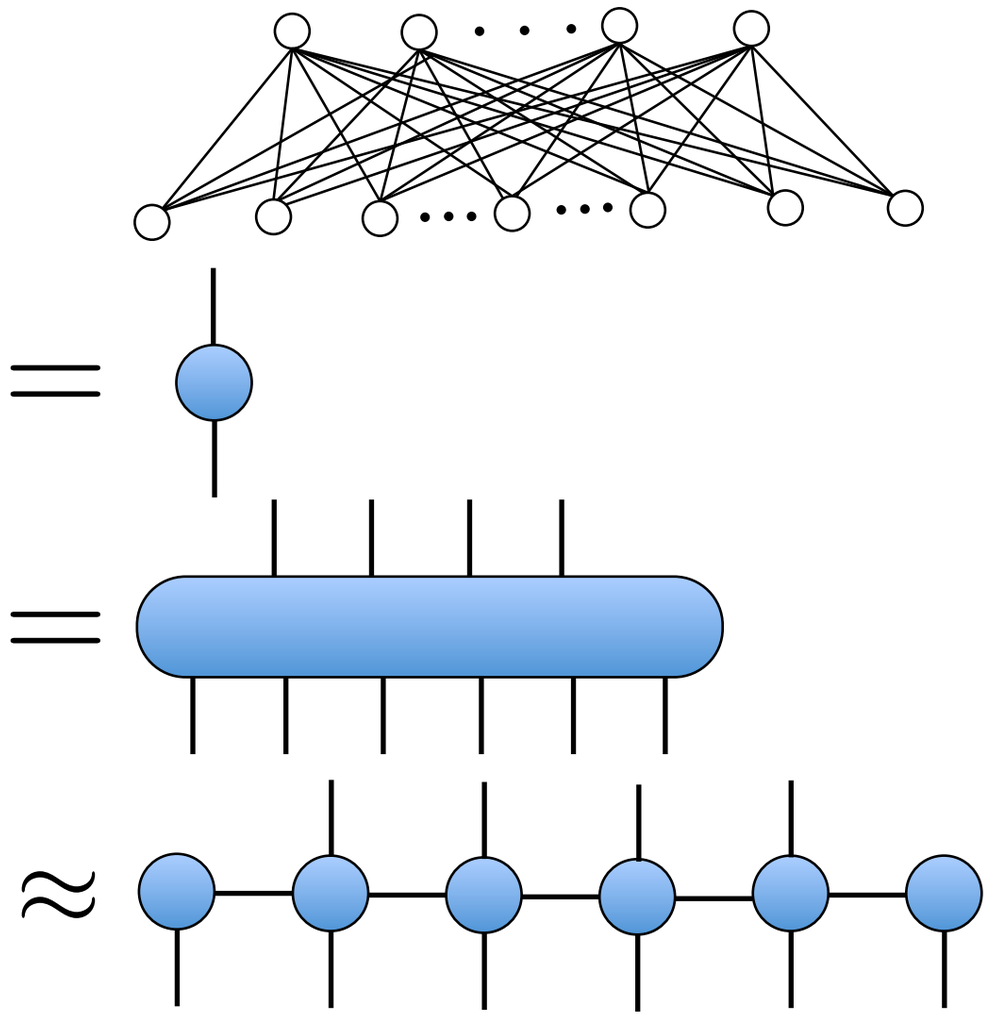


Image courtesy: Fjodor van Veen

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*.



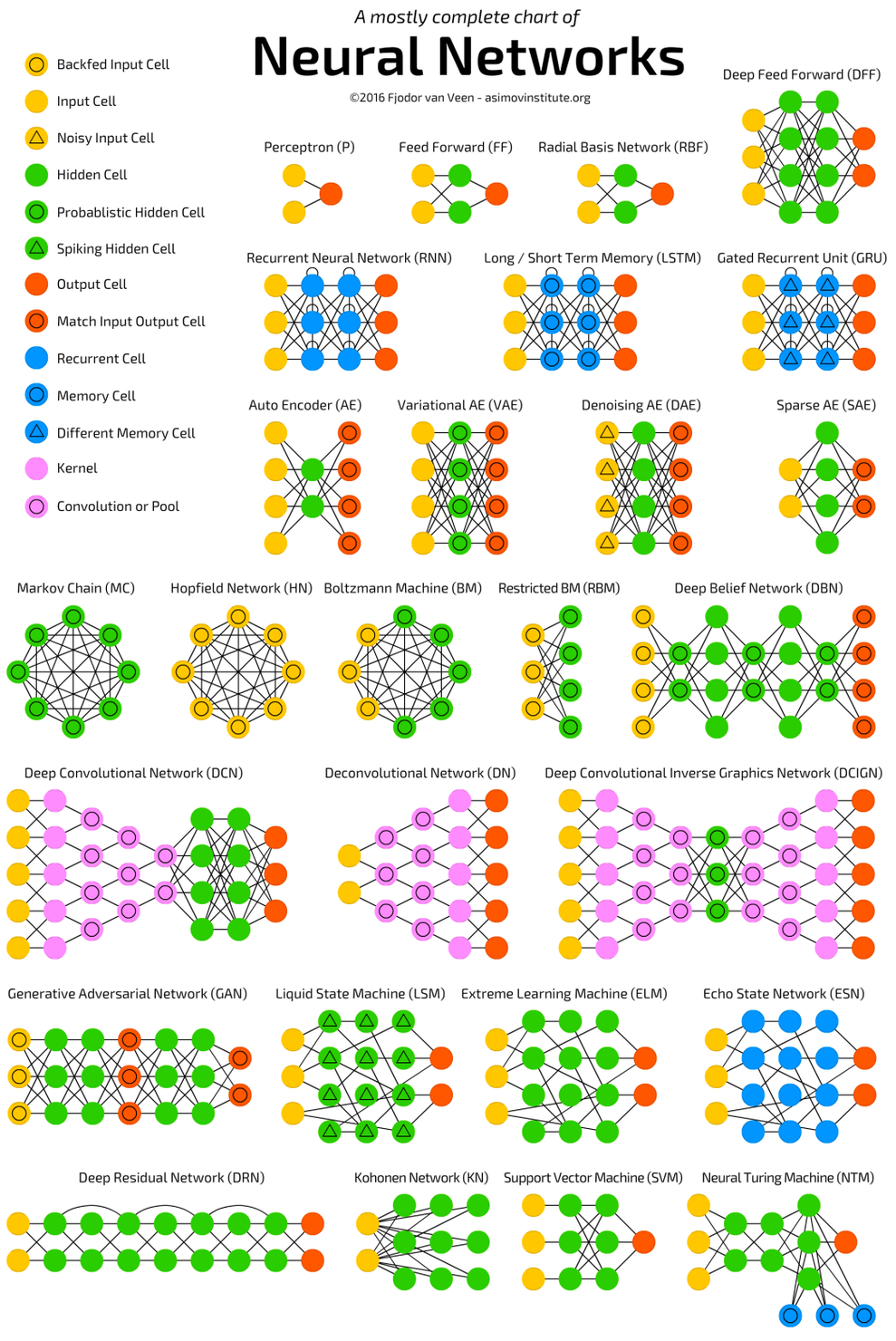
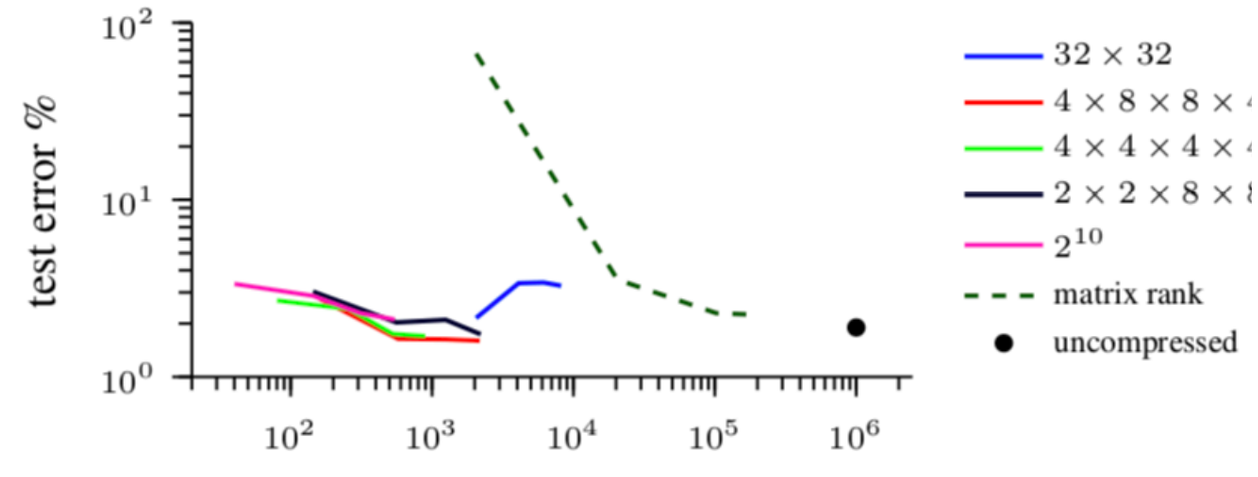


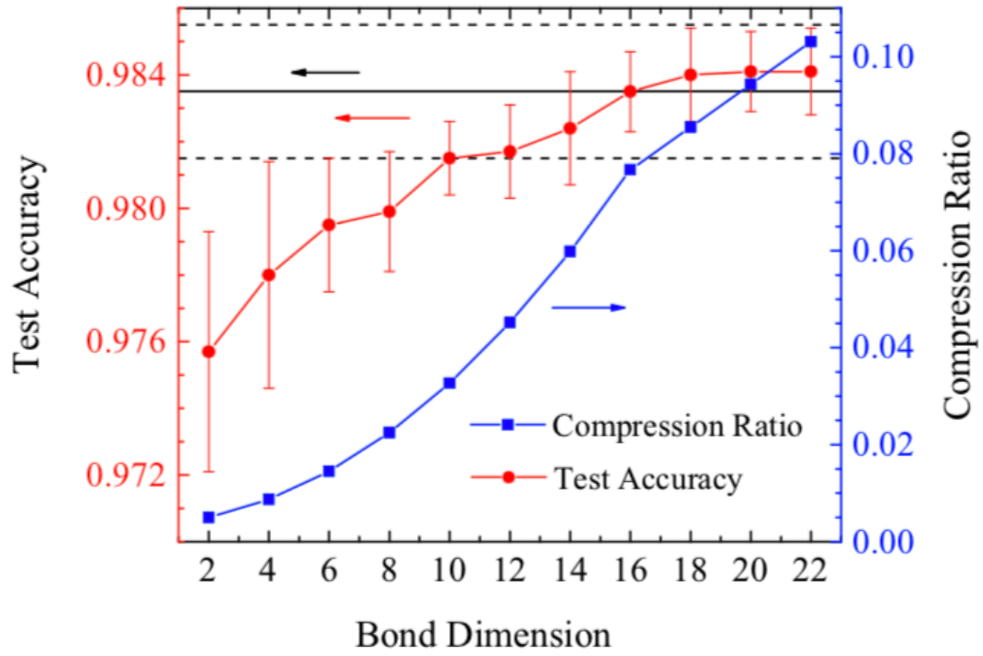
Image courtesy: Fjodor van Veen

Application: compressing neural networks

On 2-layer perceptron, MNIST



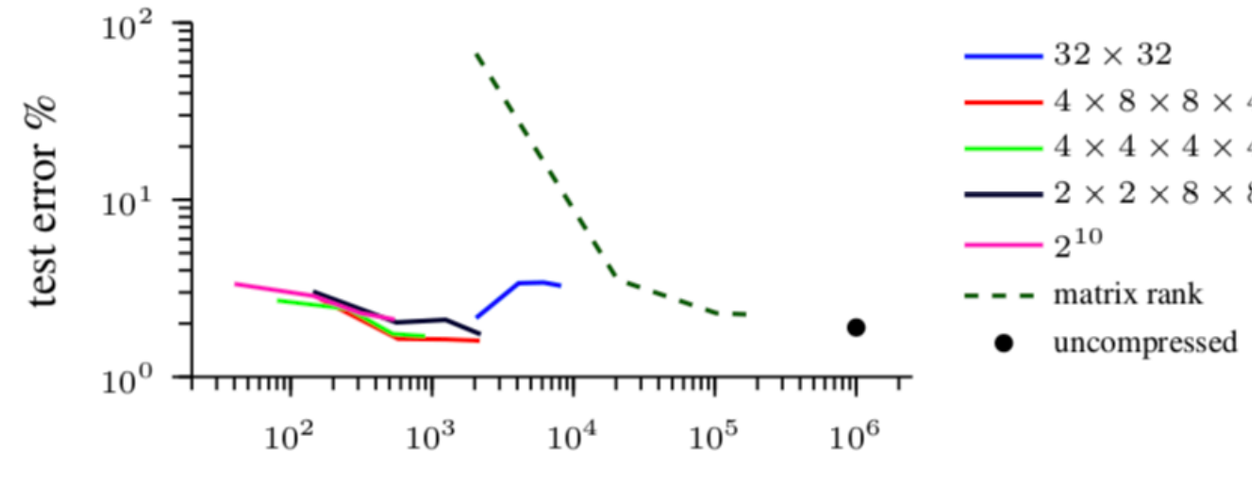
Novikov,Podoprikhin, Osokin, Vetrov, NIPS 2015



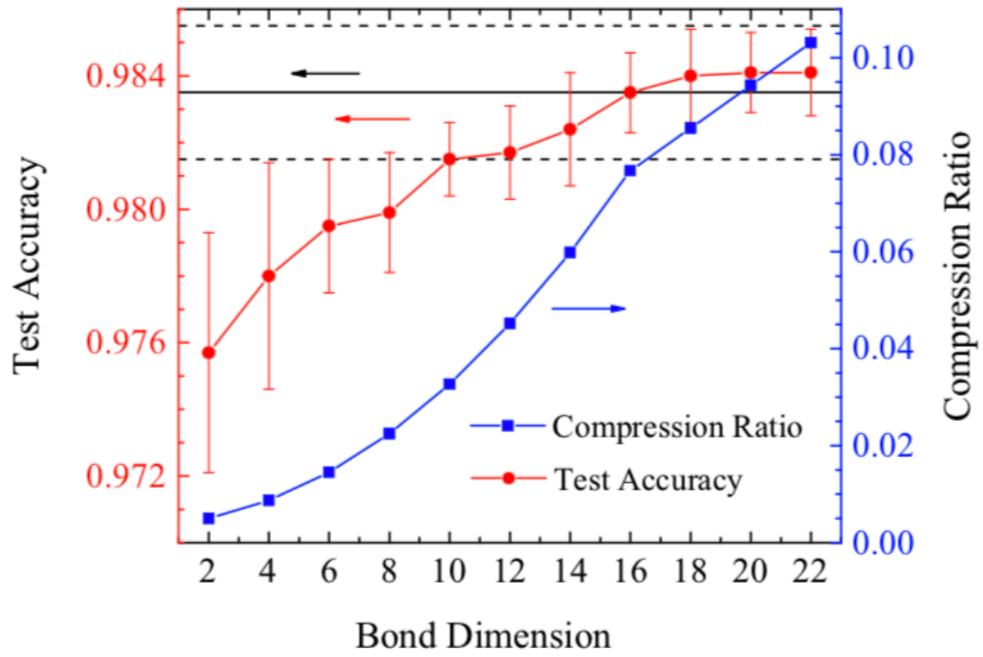
Gao et al, arXiv:1904.06194

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

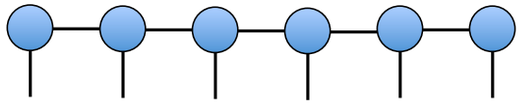
- Tensor networks for representations
  (in a small space)
  - CP, Tucker, MPS, Tree Tensor networks
  - Example: Compressing neural networks

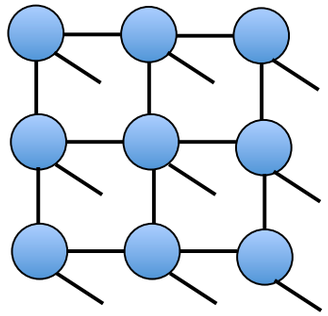
➡ Contraction of tensor networks

- 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

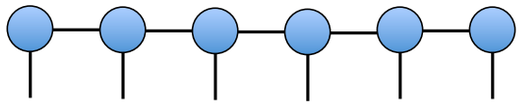
- Evaluating a wave function

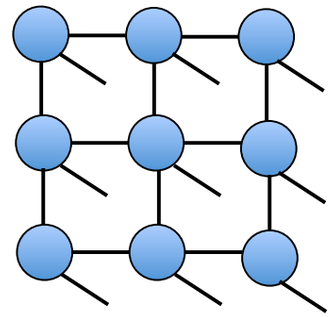




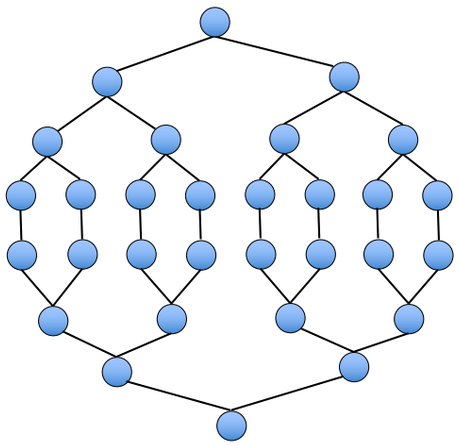
Contracting tensor networks corresponds to

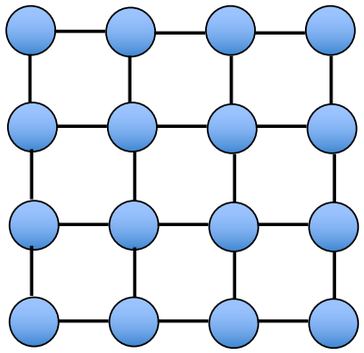
- Evaluating a wave function





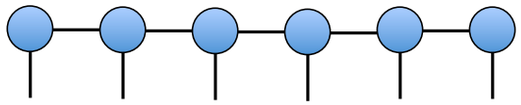
- Marginalizing a graphical model

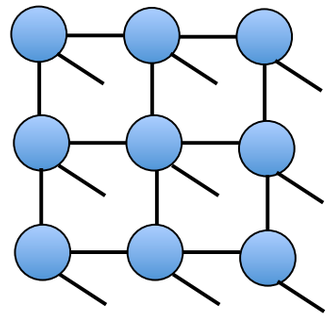




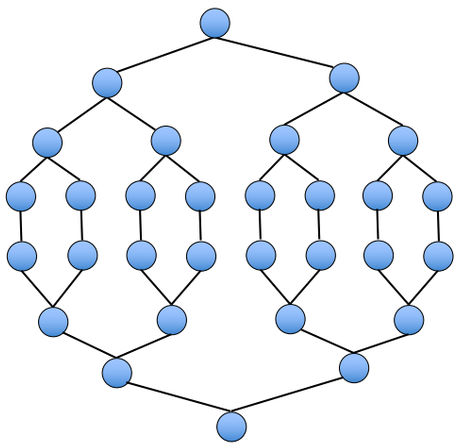
Contracting tensor networks corresponds to

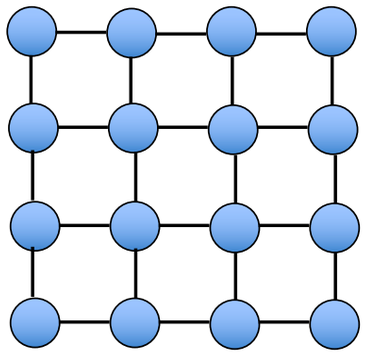
- Evaluating a wave function



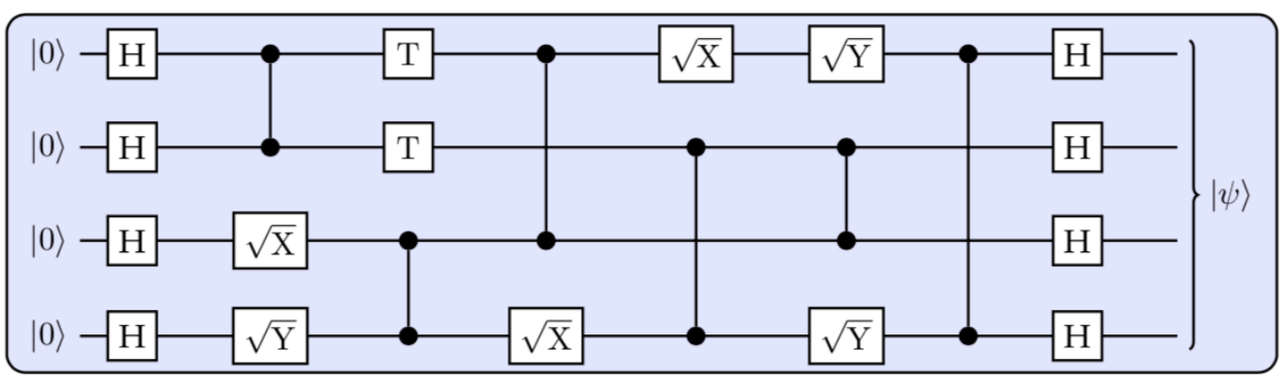


- Marginalizing a graphical model





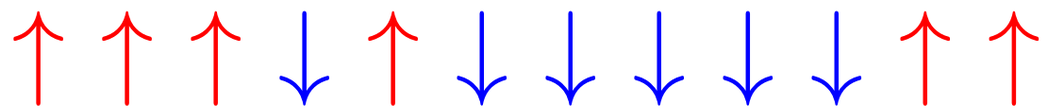
- Simulating a quantum circuit



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

Example: Ising model

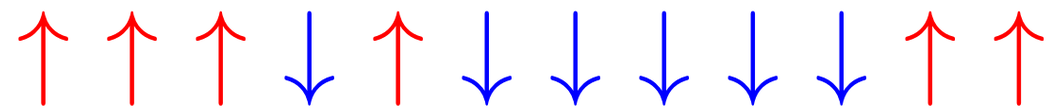
$$\mathbf{S} = \{+1, -1\}^n$$

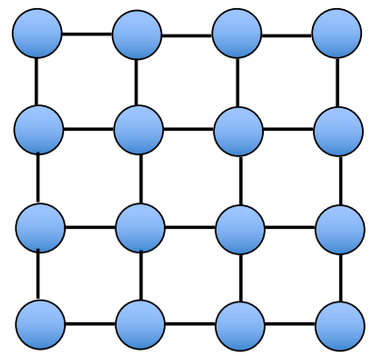


$$P(\mathbf{S}) = \frac{1}{Z} \mathrm{e}^{-\beta E(\mathbf{S})} \qquad Z = \sum_{\mathbf{s}} \mathrm{e}^{-\beta E(\mathbf{S})}$$

Example: Ising model

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



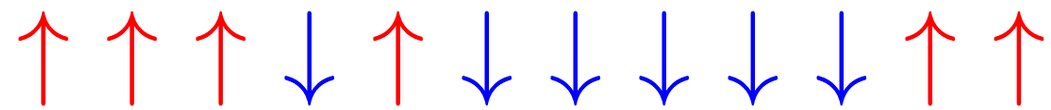


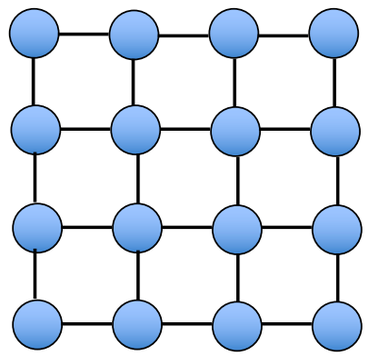
$$P(\mathbf{S}) = \frac{1}{Z} \mathrm{e}^{-\beta E(\mathbf{S})} \qquad Z = \sum_{\mathbf{s}} \mathrm{e}^{-\beta E(\mathbf{S})}$$

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

Example: Ising model

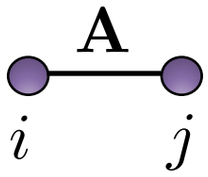
$$\mathbf{S} = \{+1, -1\}^n$$

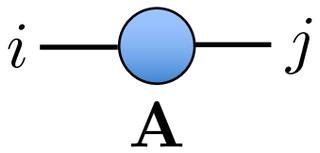




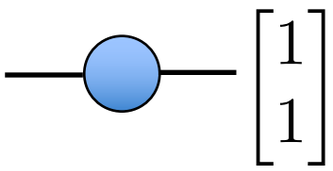
$$P(\mathbf{S}) = \frac{1}{Z} \mathrm{e}^{-\beta E(\mathbf{S})} \qquad Z = \sum_{\mathbf{s}} \mathrm{e}^{-\beta E(\mathbf{S})}$$

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



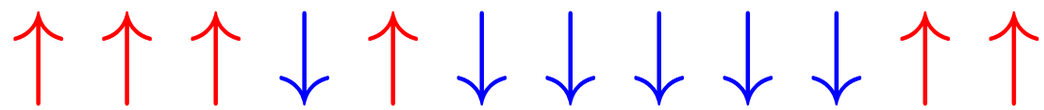


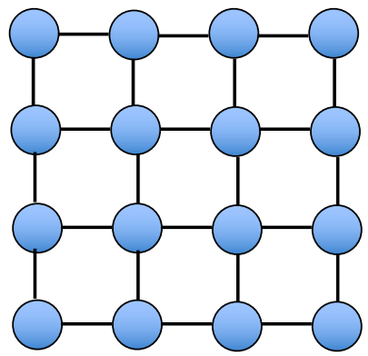
$$Z = \sum_{i,j} A_{ij} = \mathbf{1}^T A \mathbf{1} =$$



Example: Ising model

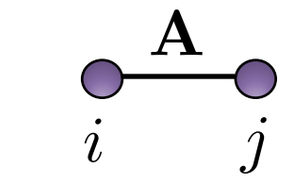
$$\mathbf{S} = \{+1, -1\}^n$$

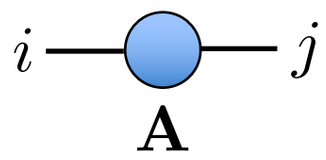




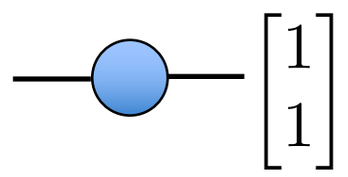
$$P(\mathbf{S}) = \frac{1}{Z} \mathrm{e}^{-\beta E(\mathbf{S})} \qquad Z = \sum_{\mathbf{s}} \mathrm{e}^{-\beta E(\mathbf{S})}$$

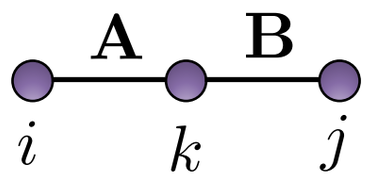
Probability distribution is a tensor, hence can be represented by a tensor network in general.

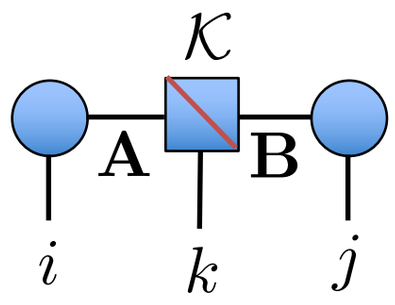




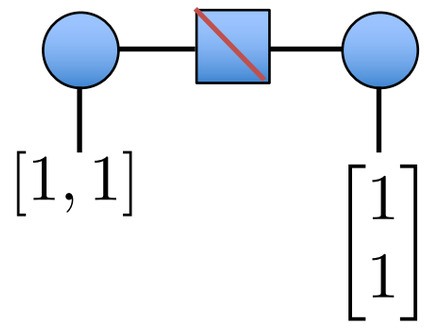
$$Z = \sum_{i,j} A_{ij} = \mathbf{1}^T A \mathbf{1} =$$





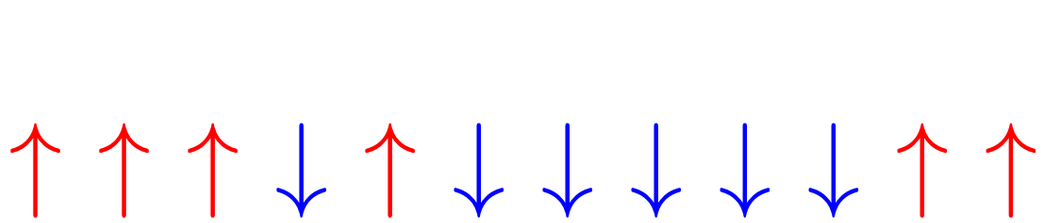


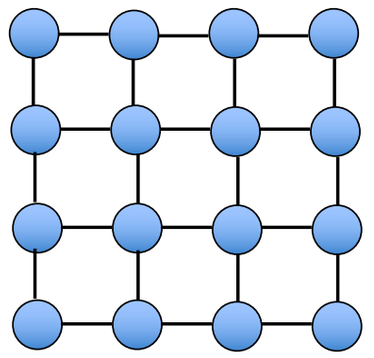
$$Z = \mathrm{Tr}\mathbf{AB} =$$



Example: Ising model

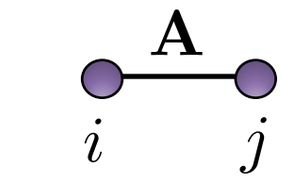
$$\mathbf{S} = \{+1, -1\}^n$$

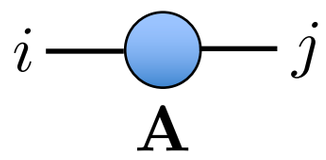




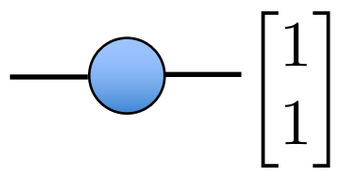
$$P(\mathbf{S}) = \frac{1}{Z} \mathrm{e}^{-\beta E(\mathbf{S})} \qquad Z = \sum_{\mathbf{s}} \mathrm{e}^{-\beta E(\mathbf{S})}$$

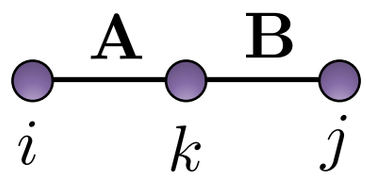
Probability distribution is a tensor, hence can be represented by a tensor network in general.

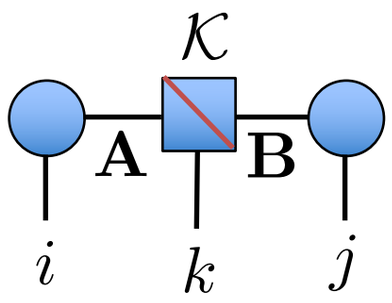




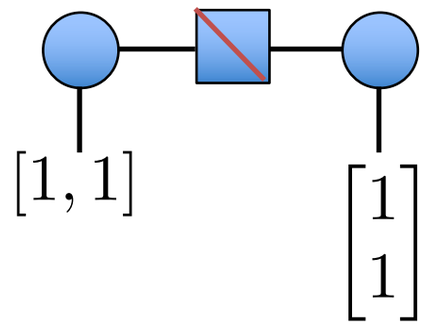
$$Z = \sum_{i,j} A_{ij} = \mathbf{1}^T A \mathbf{1} =$$





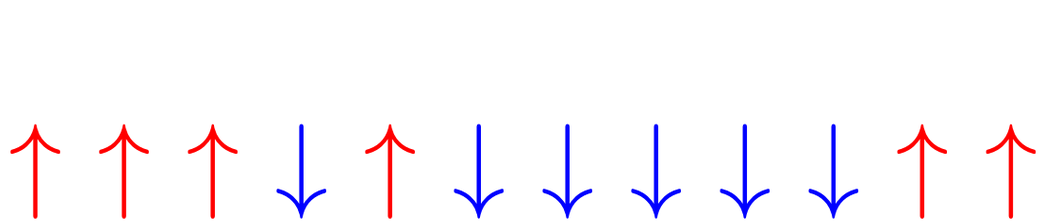


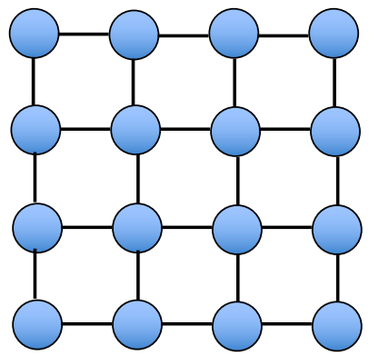
$$Z = \mathrm{Tr}\mathbf{AB} =$$



Example: Ising model

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





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

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

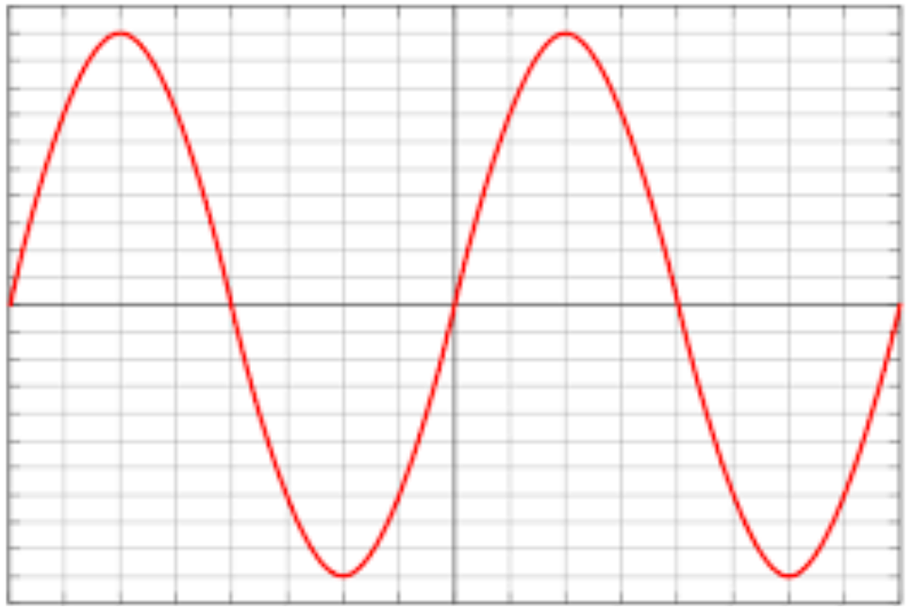
Outline

- 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) Generalization !!!

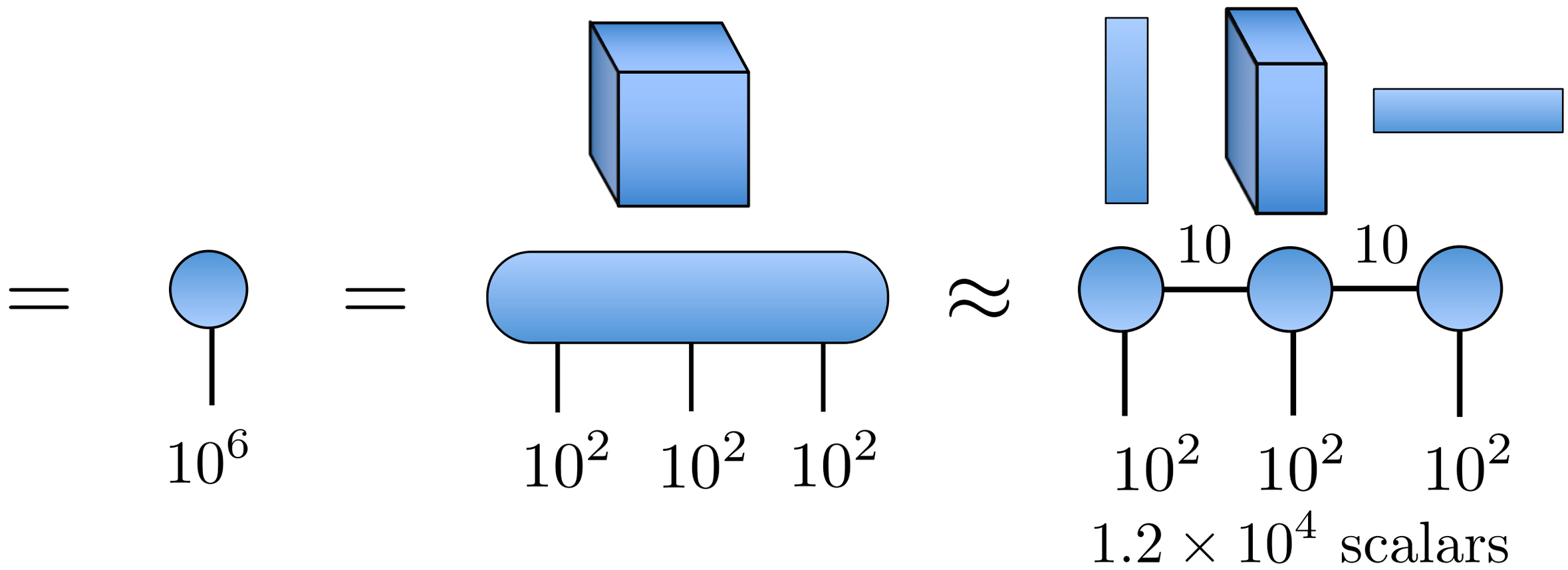
  - Hilbert space as feature space: classification and PCA using tensor networks
  - Tensor networks for generative modeling

Overview: large-scale representation



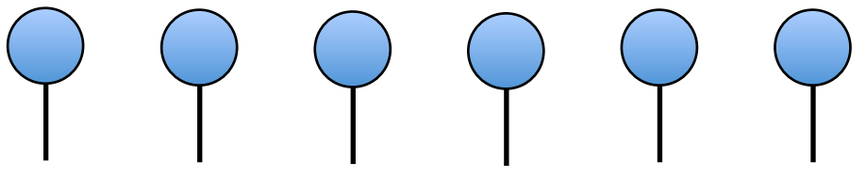
[0.000, 0.100, 0.199, 0.296, 0.389, 0.479, 0.565, 0.644, 0.717, 0.783,
0.841, 0.891, 0.932, 0.964, 0.985, 0.997, 1.000, 0.992, 0.974, 0.946,
0.909, 0.863, 0.808, 0.746, 0.675, 0.598, 0.516, 0.427, 0.335, 0.239,
0.141, 0.042, -0.058, -0.158, -0.256, -0.351, -0.443, -0.530, -0.612,
.....
.....
-0.688, -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

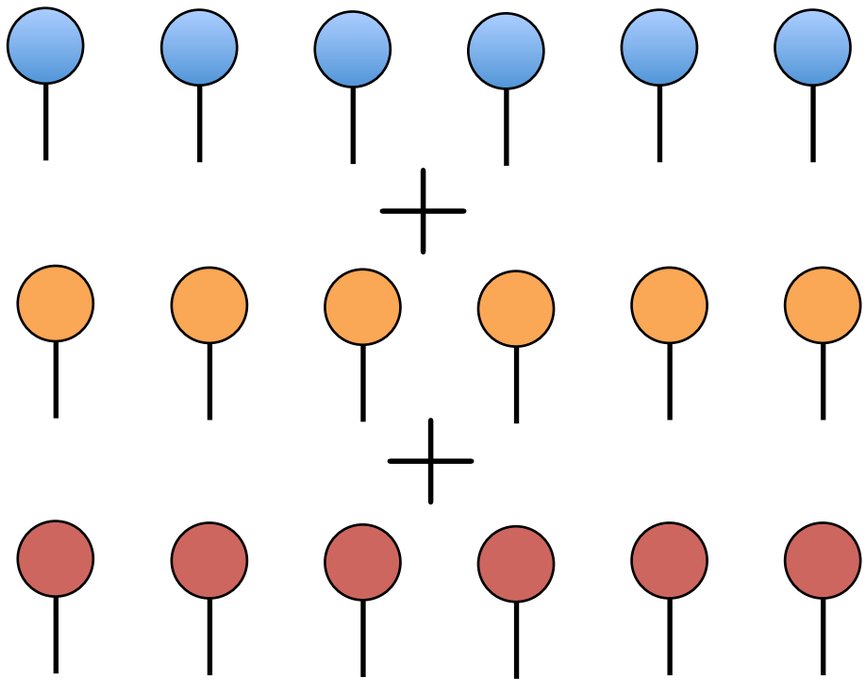


Overview: tensor decompositions

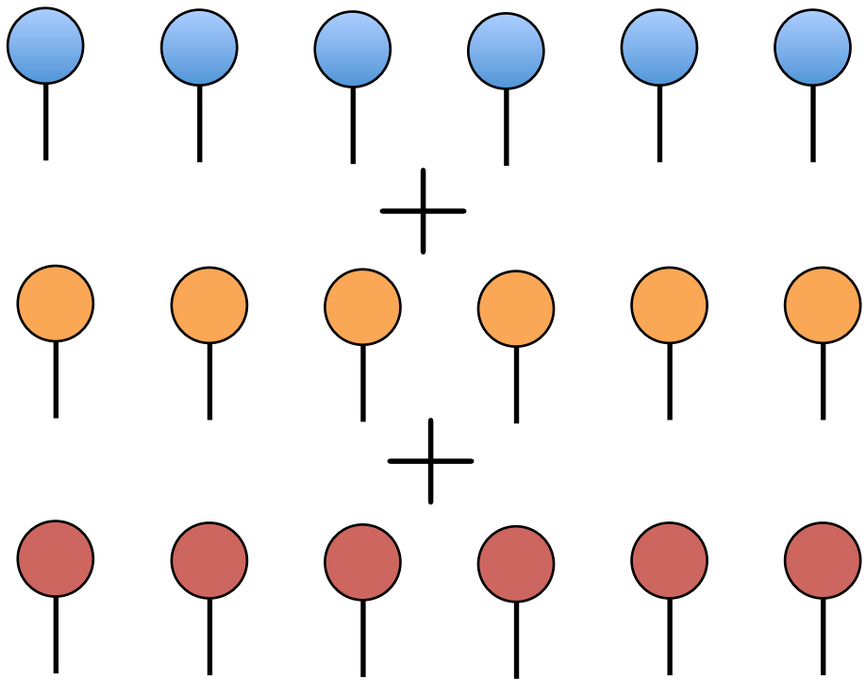
Overview: tensor decompositions

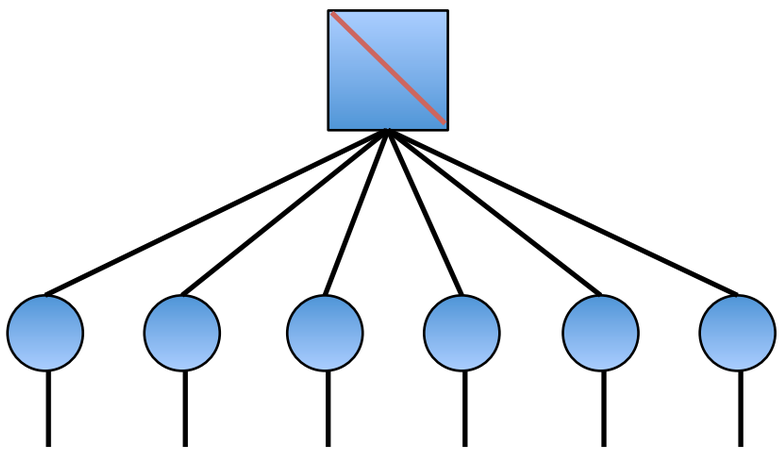


Overview: tensor decompositions

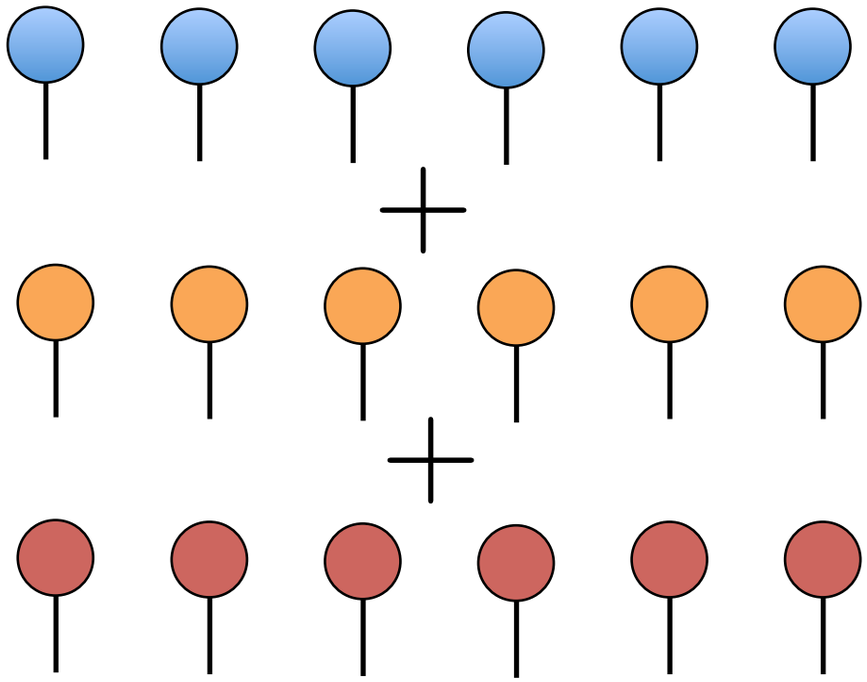


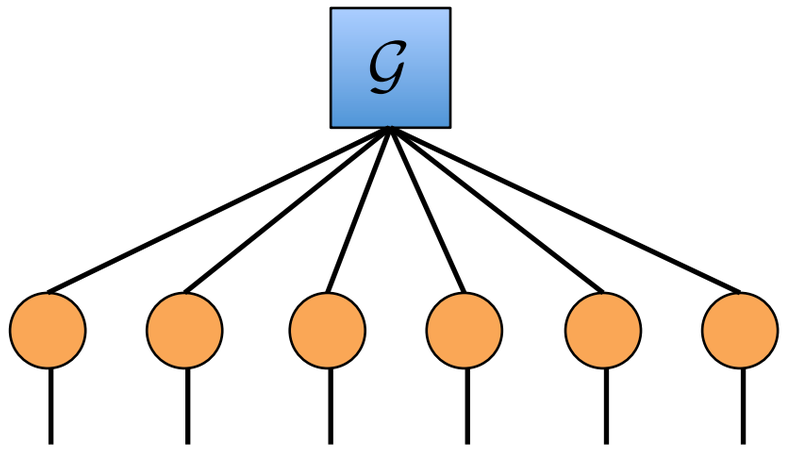
Overview: tensor decompositions

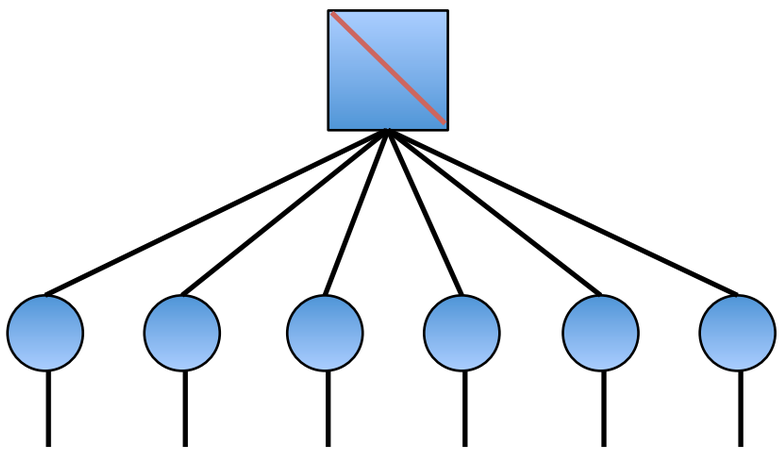




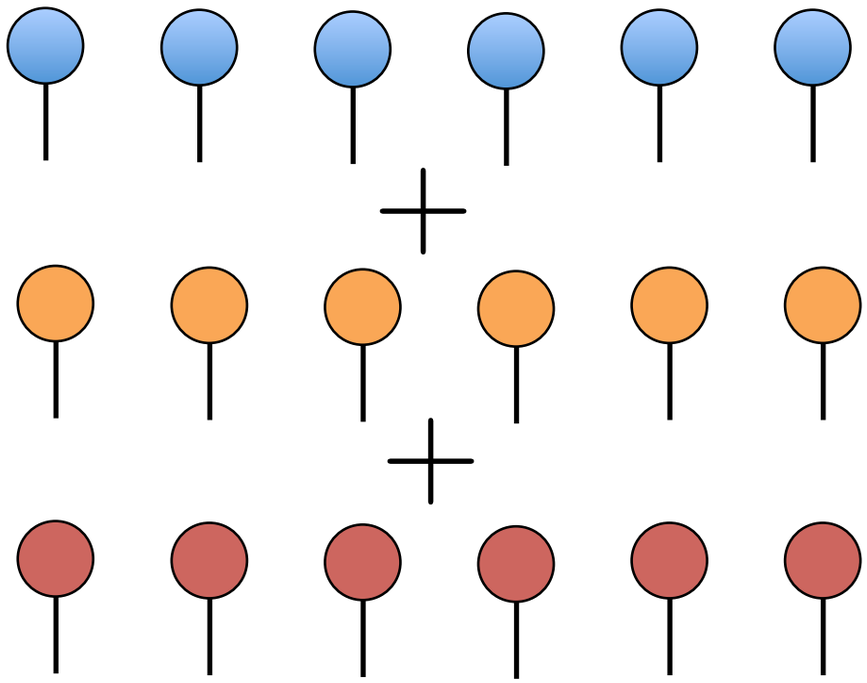
Overview: tensor decompositions

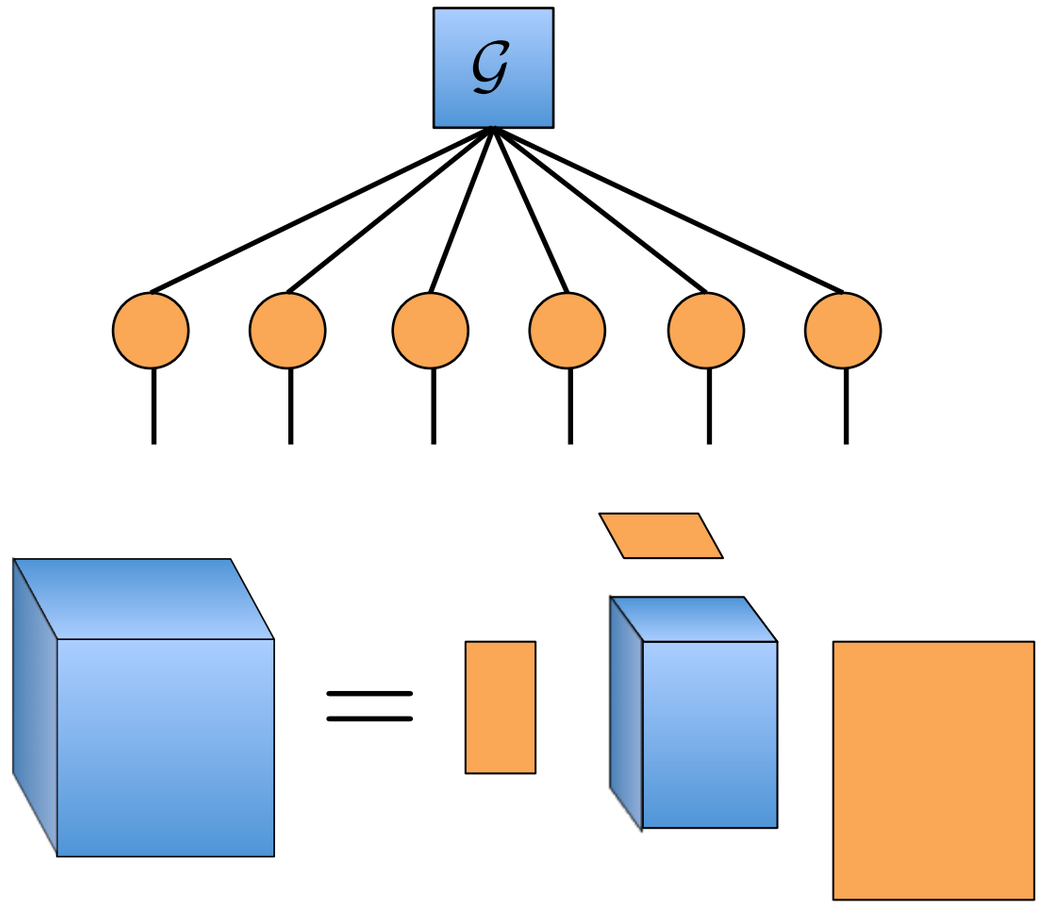


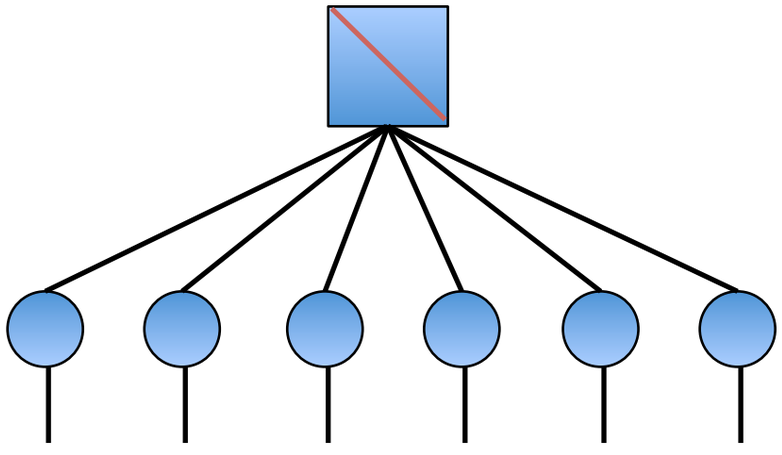




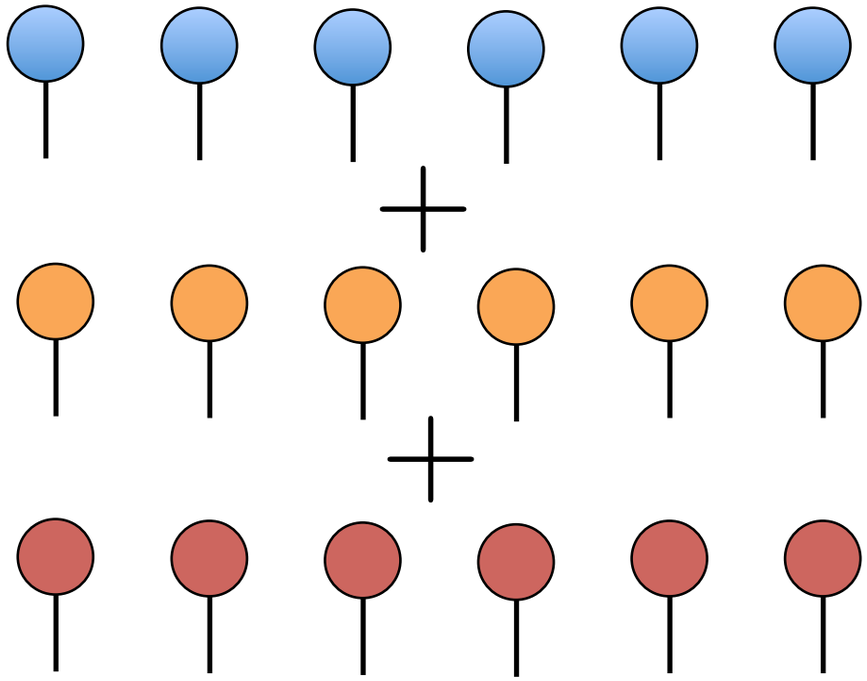
Overview: tensor decompositions

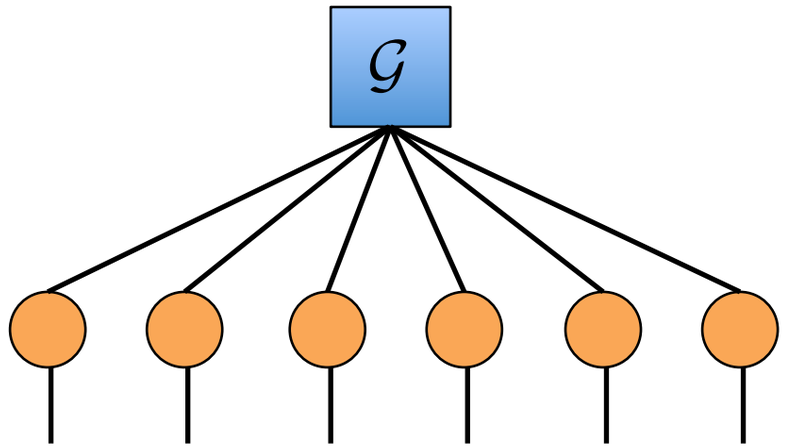


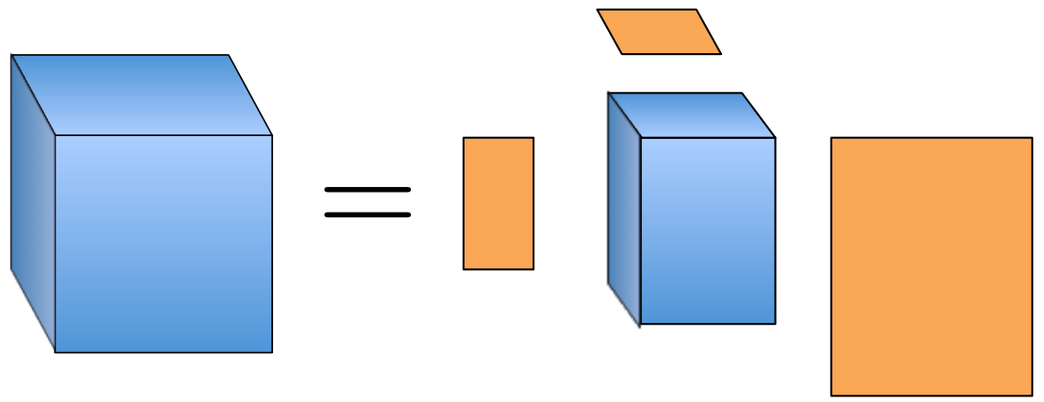


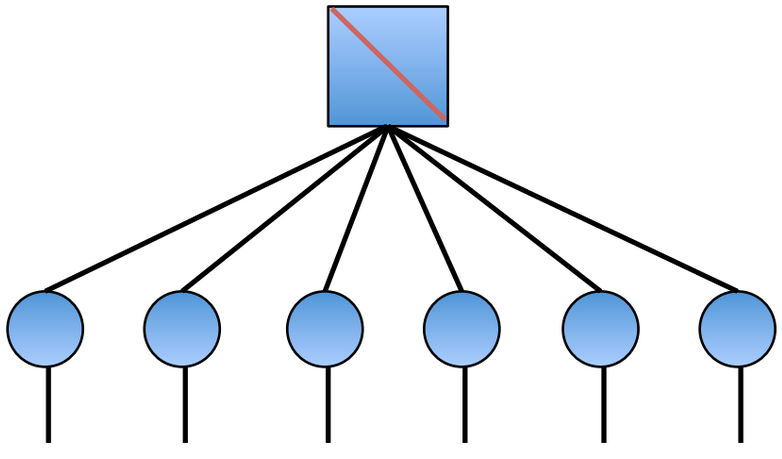


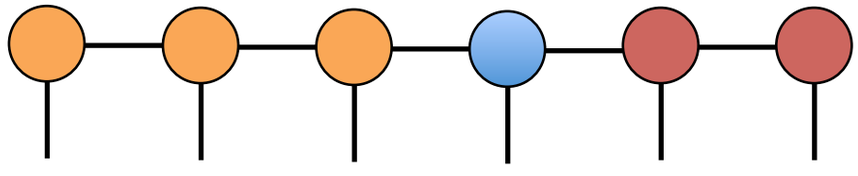
Overview: tensor decompositions



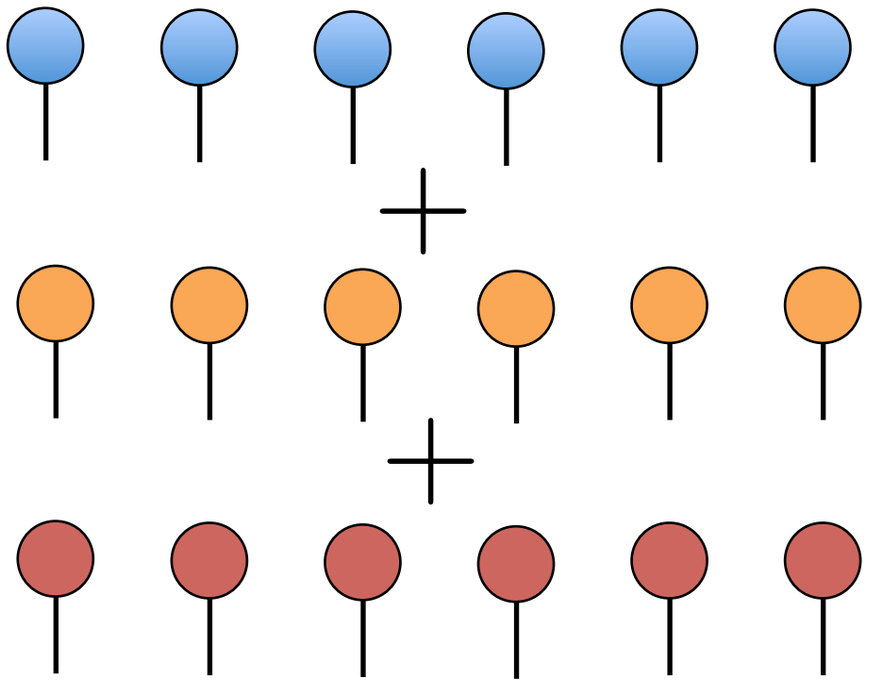


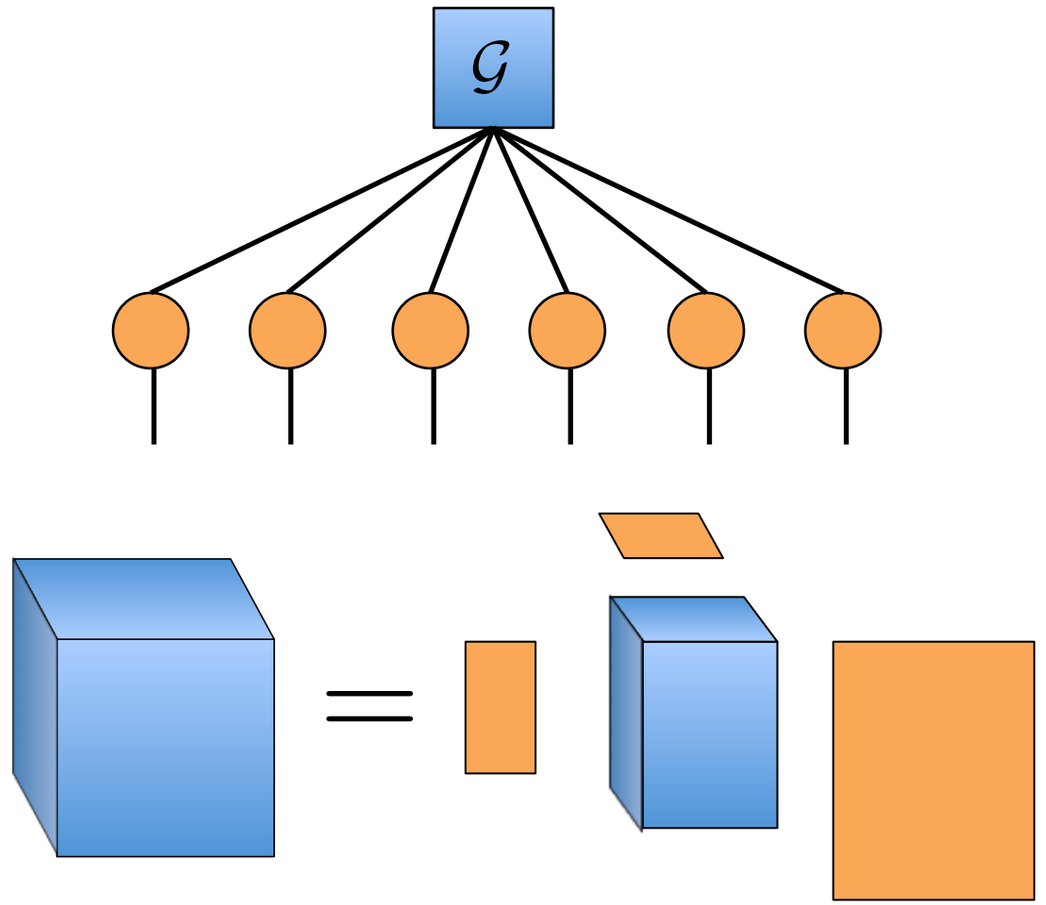


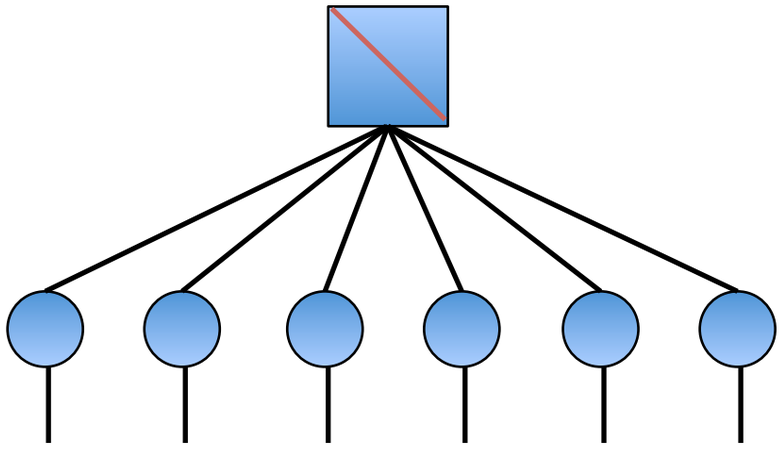


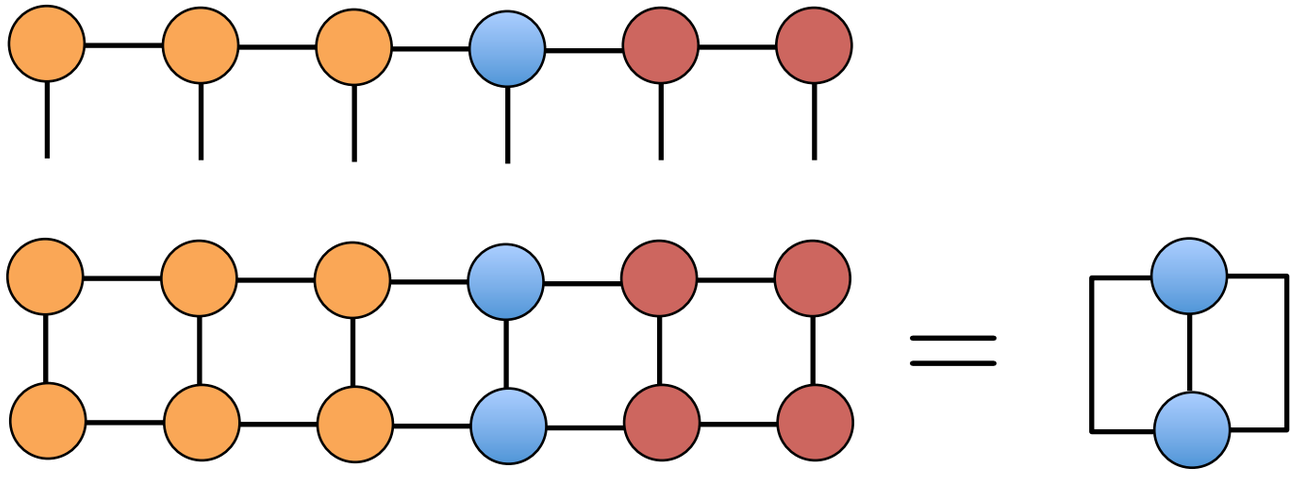


Overview: tensor decompositions



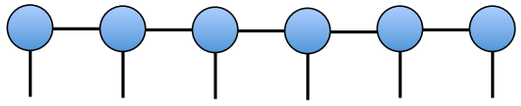


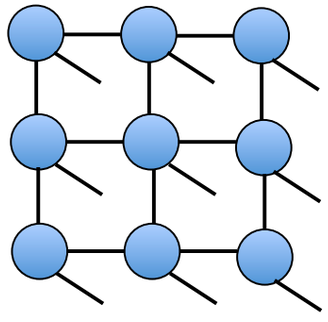




Overview: tensor contractions

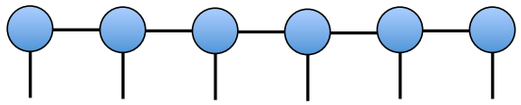
- Evaluating a wave function

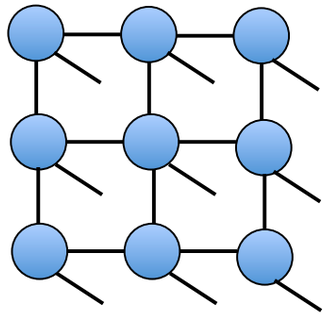




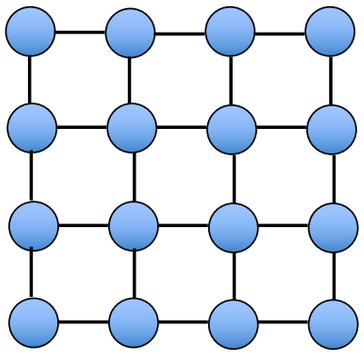
Overview: tensor contractions

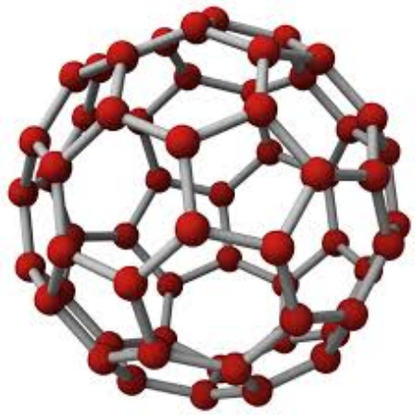
- Evaluating a wave function





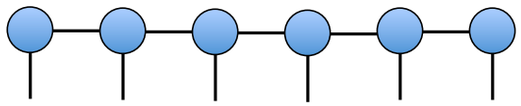
- Marginalizing (tracing off) a graphical model

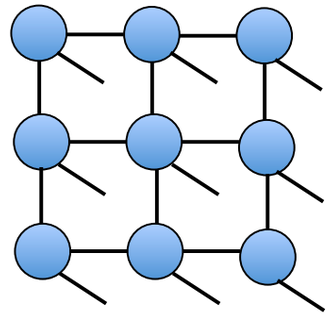




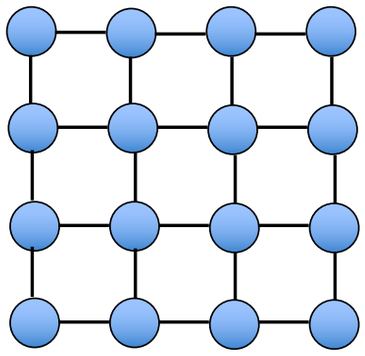
Overview: tensor contractions

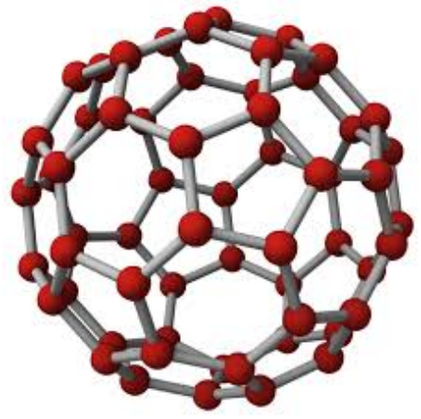
- Evaluating a wave function



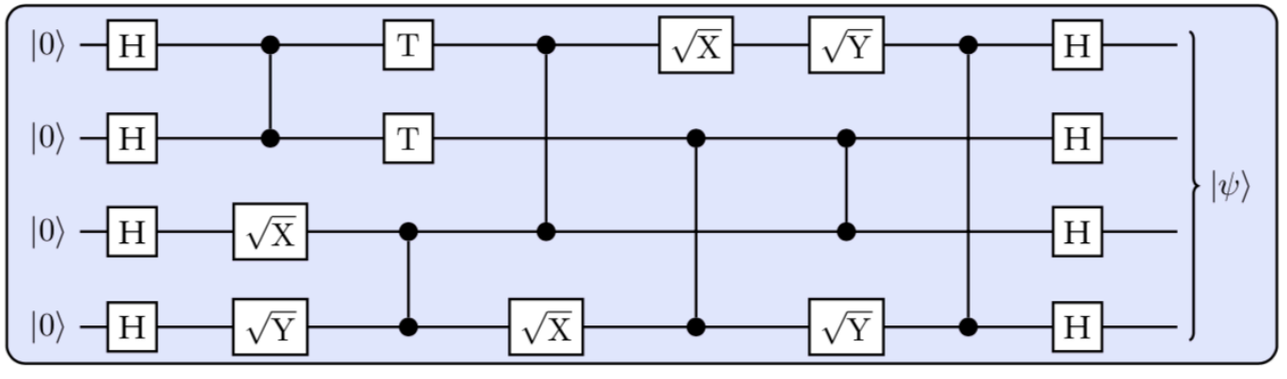


- Marginalizing (tracing off) a graphical model





- Simulating a quantum circuit

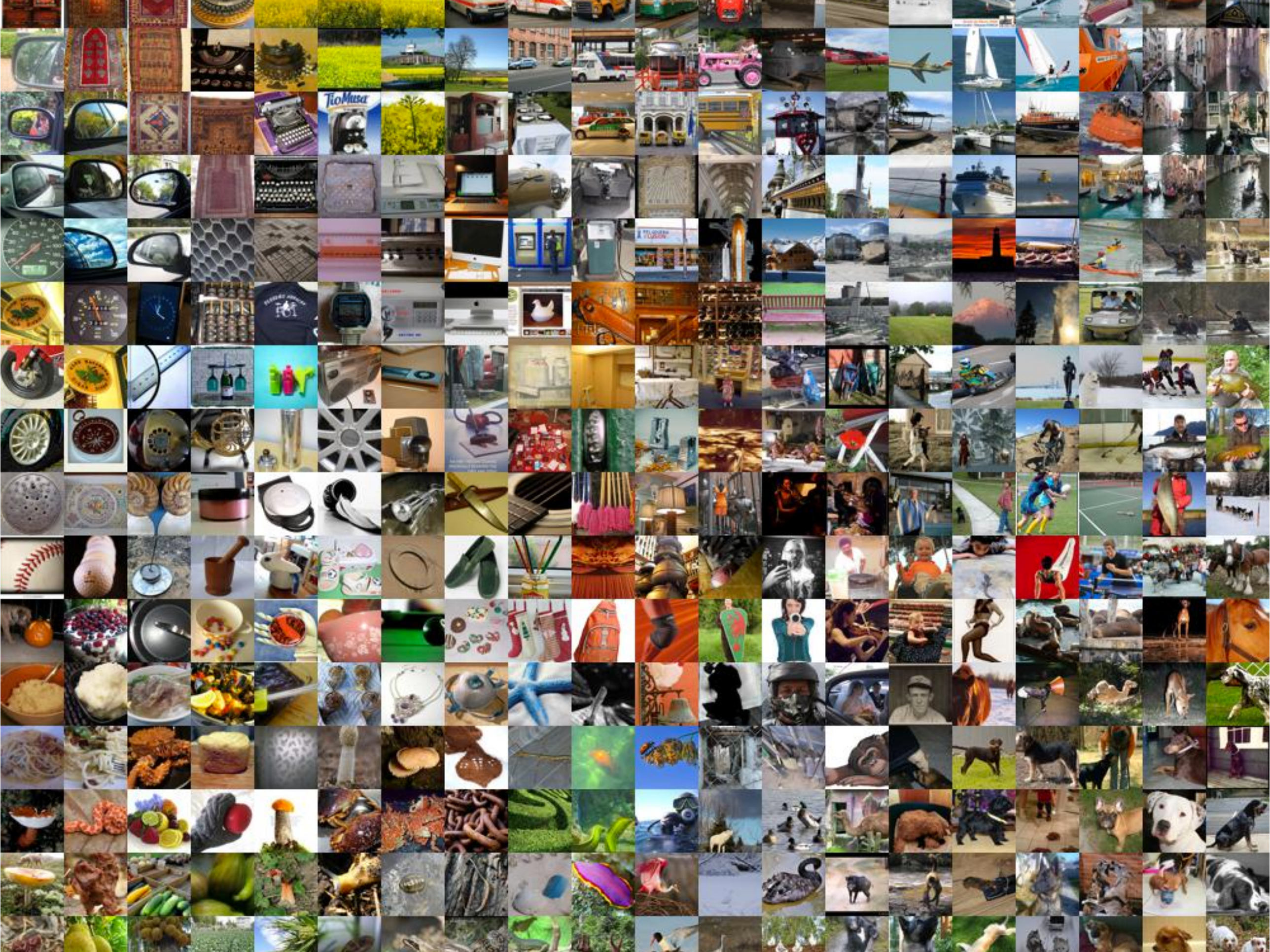


Outline

- 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) Generalization !!!

  - Hilbert space as feature space: classification and PCA using tensor networks
  - Tensor networks for generative modeling

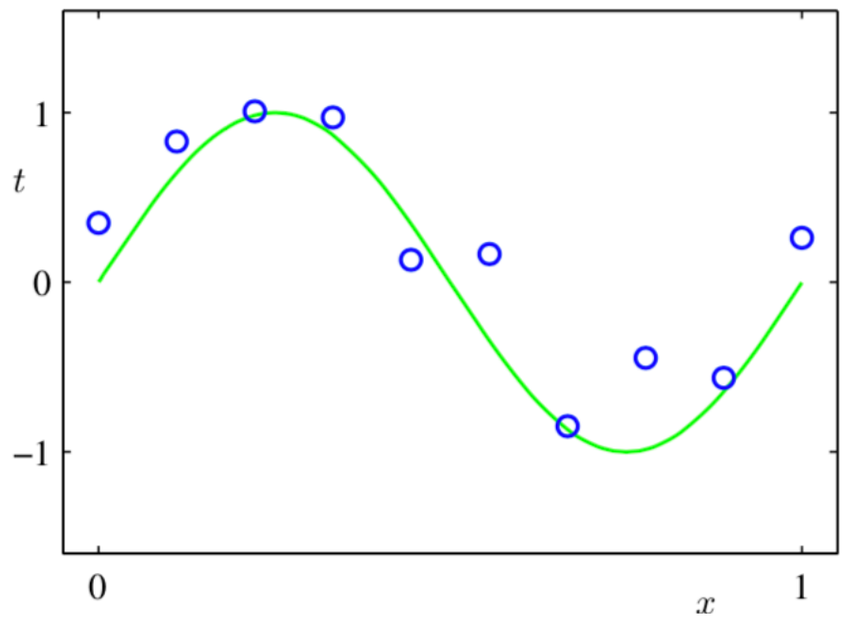



Prob.(label | data)

Prob.(*label* | *data*)

Prob.(*data*)

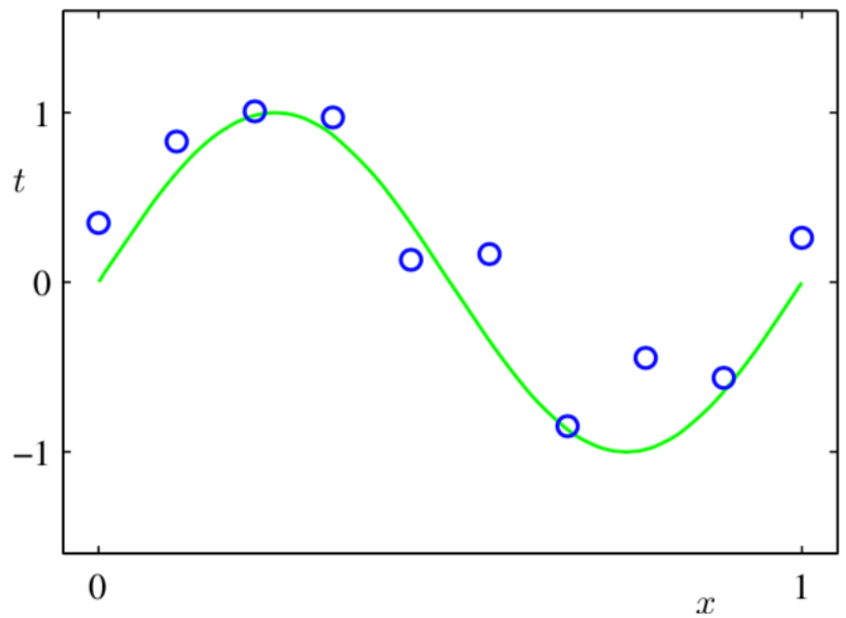
Example of learning: curve fitting



Curve fitting: Implicit discovering of the underline function $t_{\text{true}}(x) = \sin(2\pi x)$

*Image courtesy: Bishop **PRML** 2006*

Example of learning: curve fitting



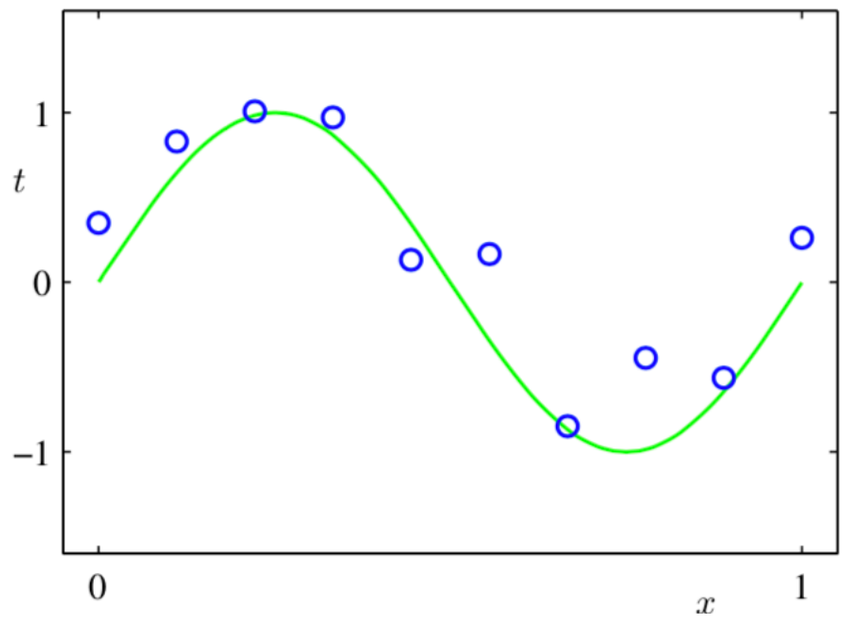
Curve fitting: Implicit discovering of the underline function $t_{\text{true}}(x) = \sin(2\pi x)$

Our model: polynomial functions of order *M*

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{i=1}^{M} w_i x^j$$

*Image courtesy: Bishop **PRML** 2006*

Example of learning: curve fitting



Curve fitting: Implicit discovering of the underline function $t_{\text{true}}(x) = \sin(2\pi x)$

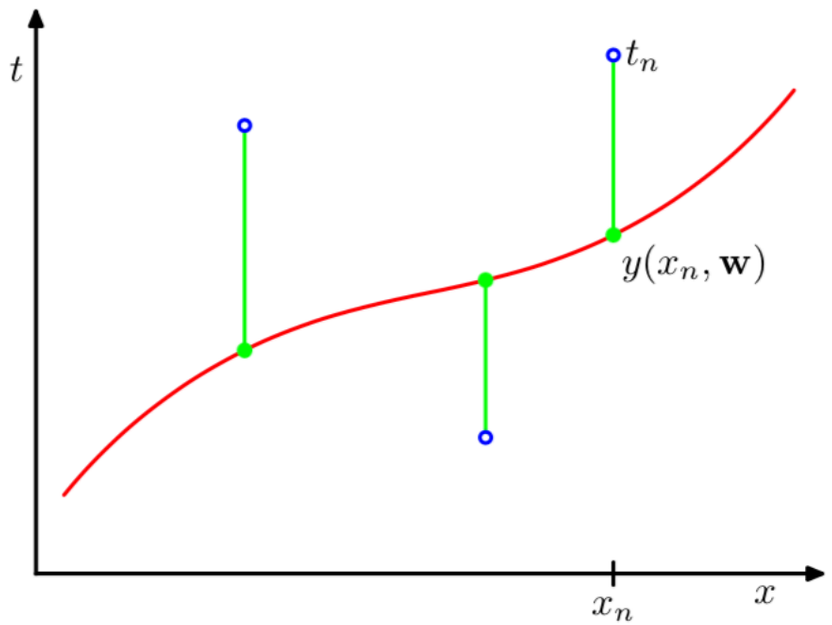
Our model: polynomial functions of order *M*

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{i=1}^{M} w_i x^j$$

Linear model in the sense of parameter ***w***

*Image courtesy: Bishop **PRML** 2006*

Example of learning: polynomial curve fitting



Commonly chose

cost (error/loss) function: sum of squares

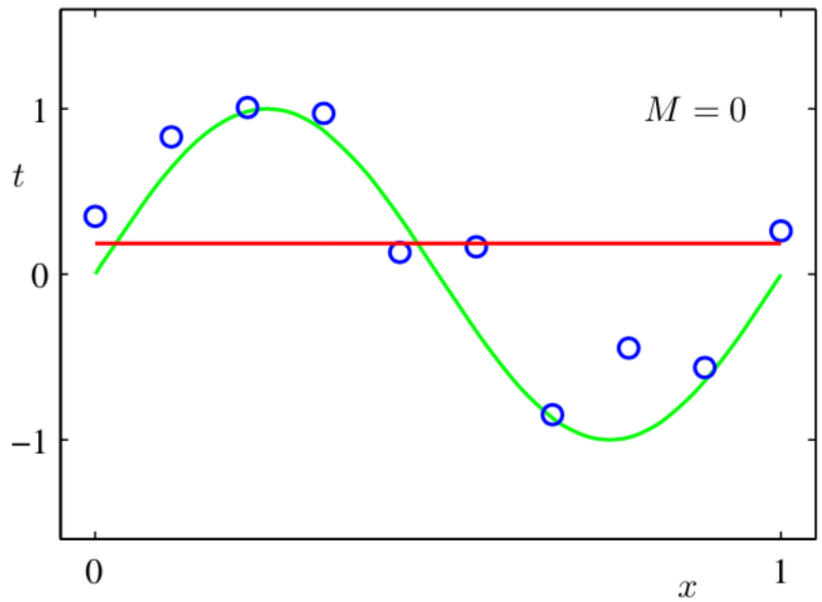
$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \mathbf{w}) - t_n)^2$$

The cost function is a quadratic function of parameters **w**

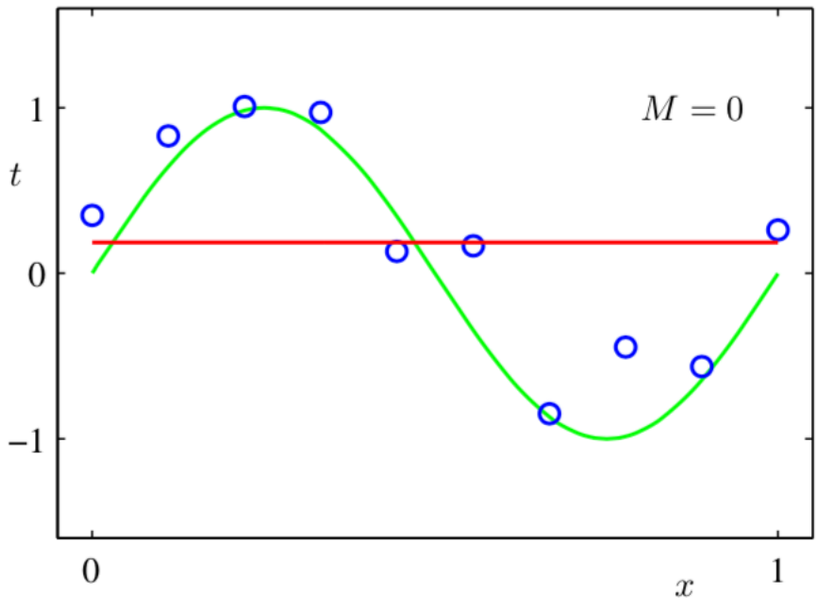
Its derivative with respect to w is linear, yielding a unique solution **w***

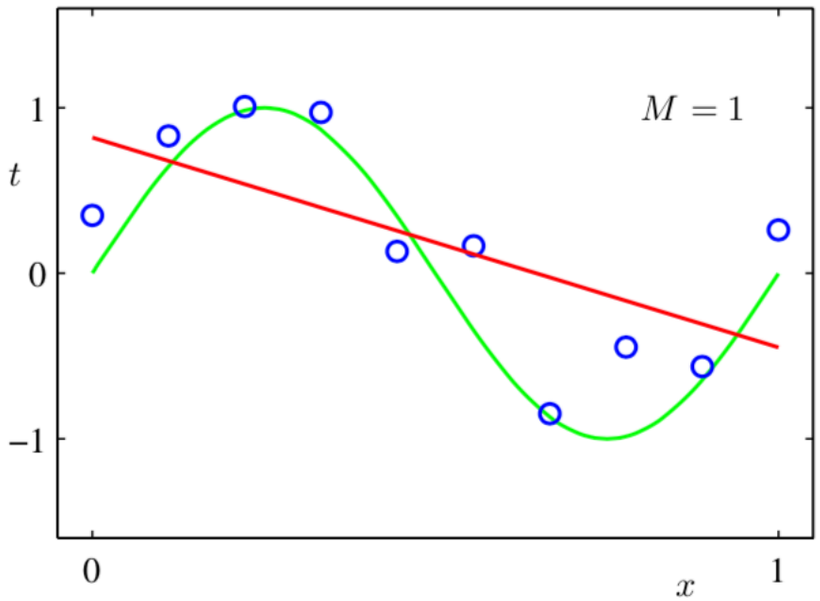
The resulting polynomial can be found in a closed form *y(x,* **w***)*

Example of learning: polynomial curve fitting

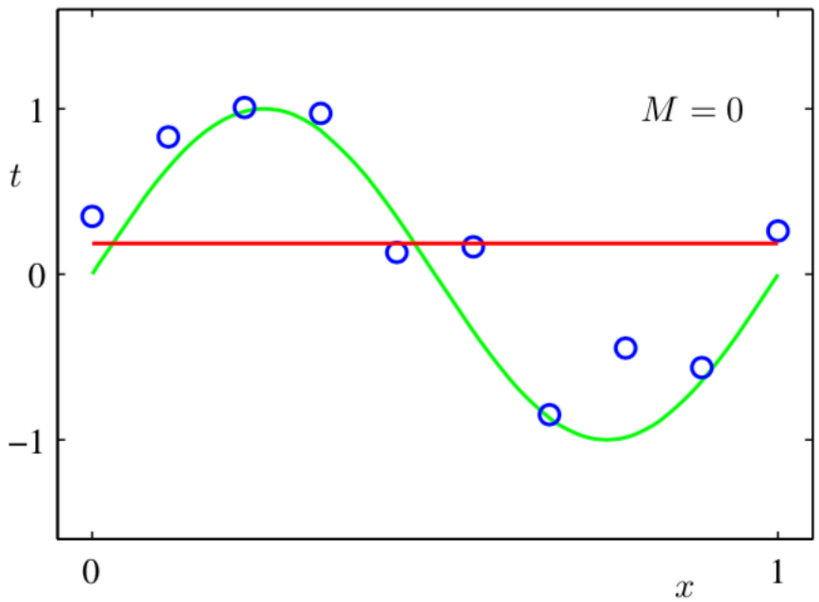


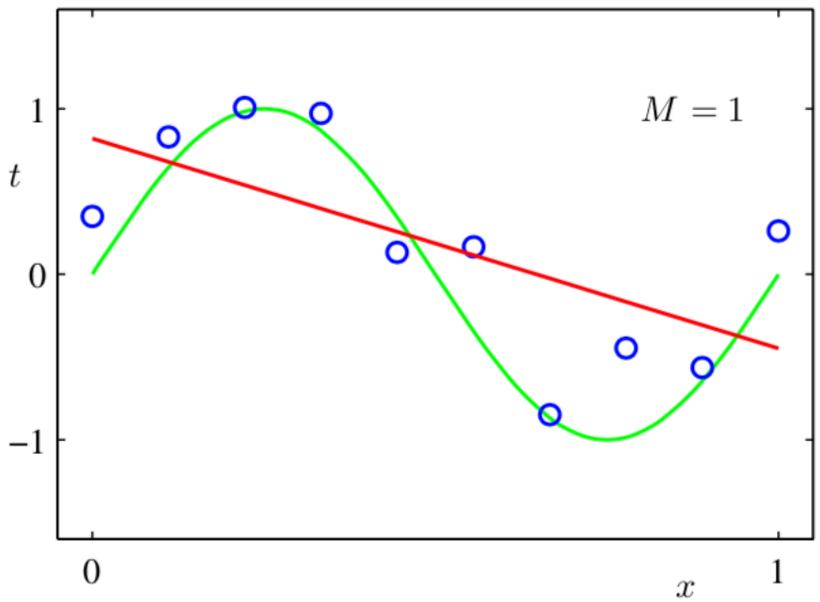
Example of learning: polynomial curve fitting

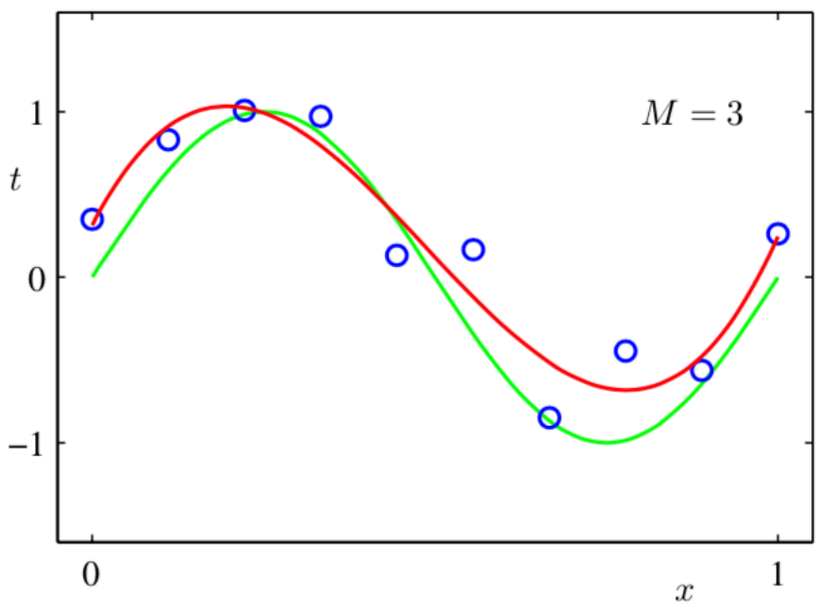




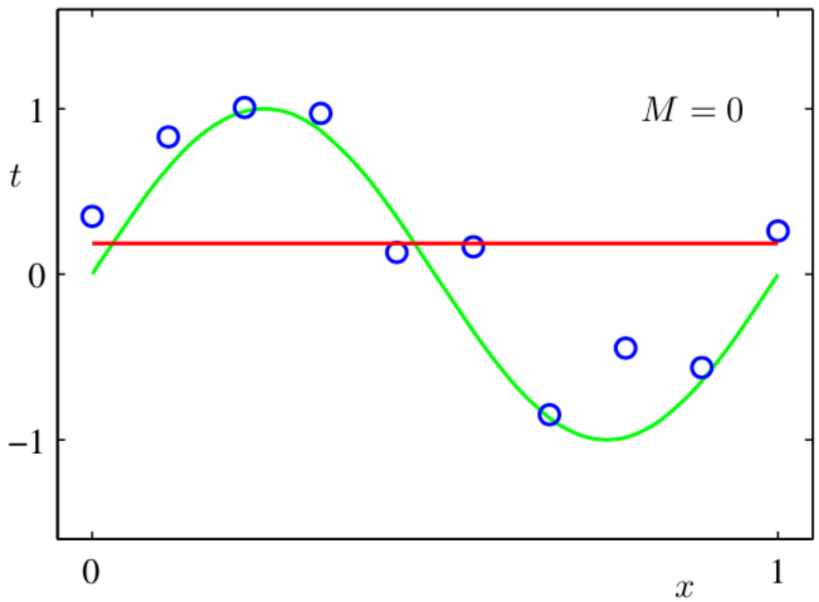
Example of learning: polynomial curve fitting

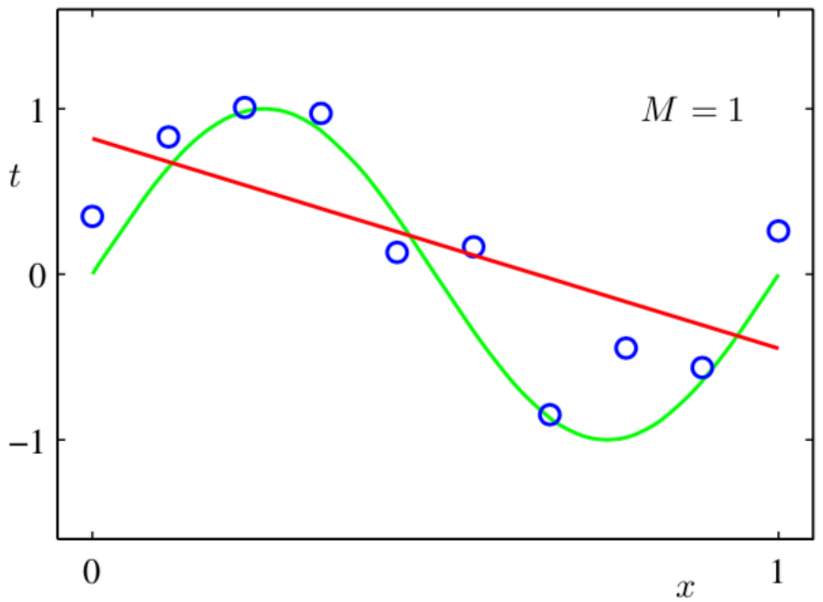


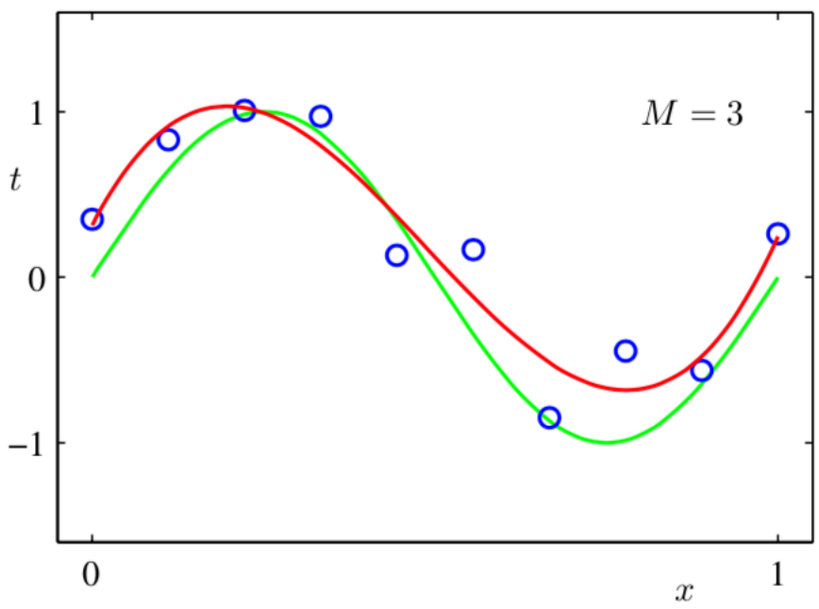


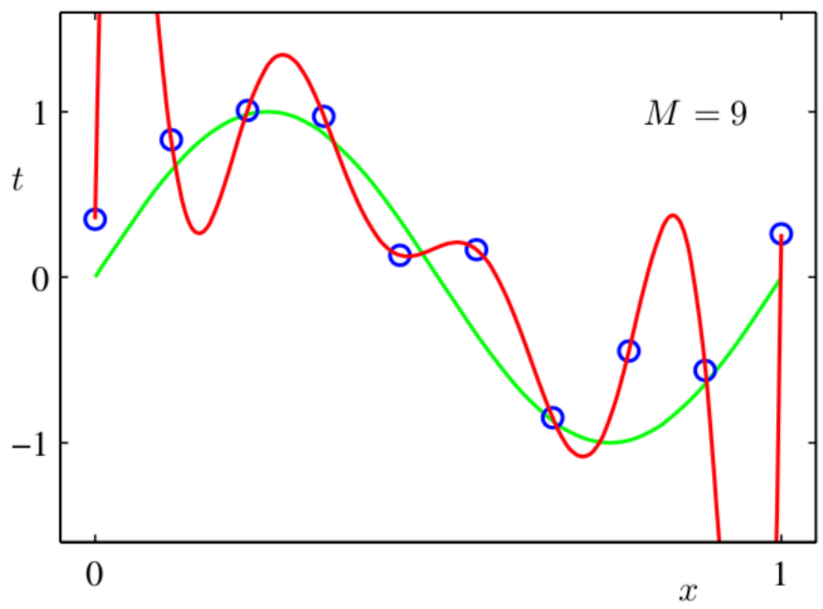


Example of learning: polynomial curve fitting



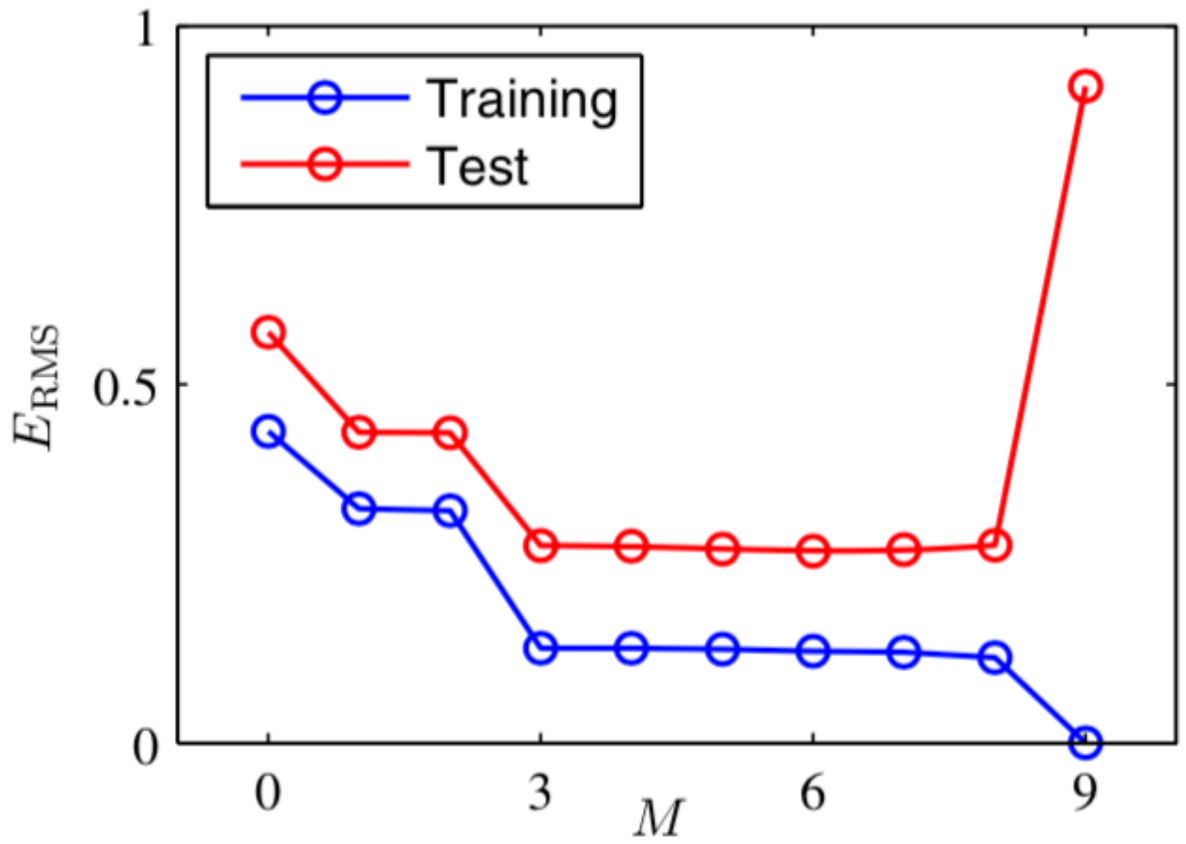






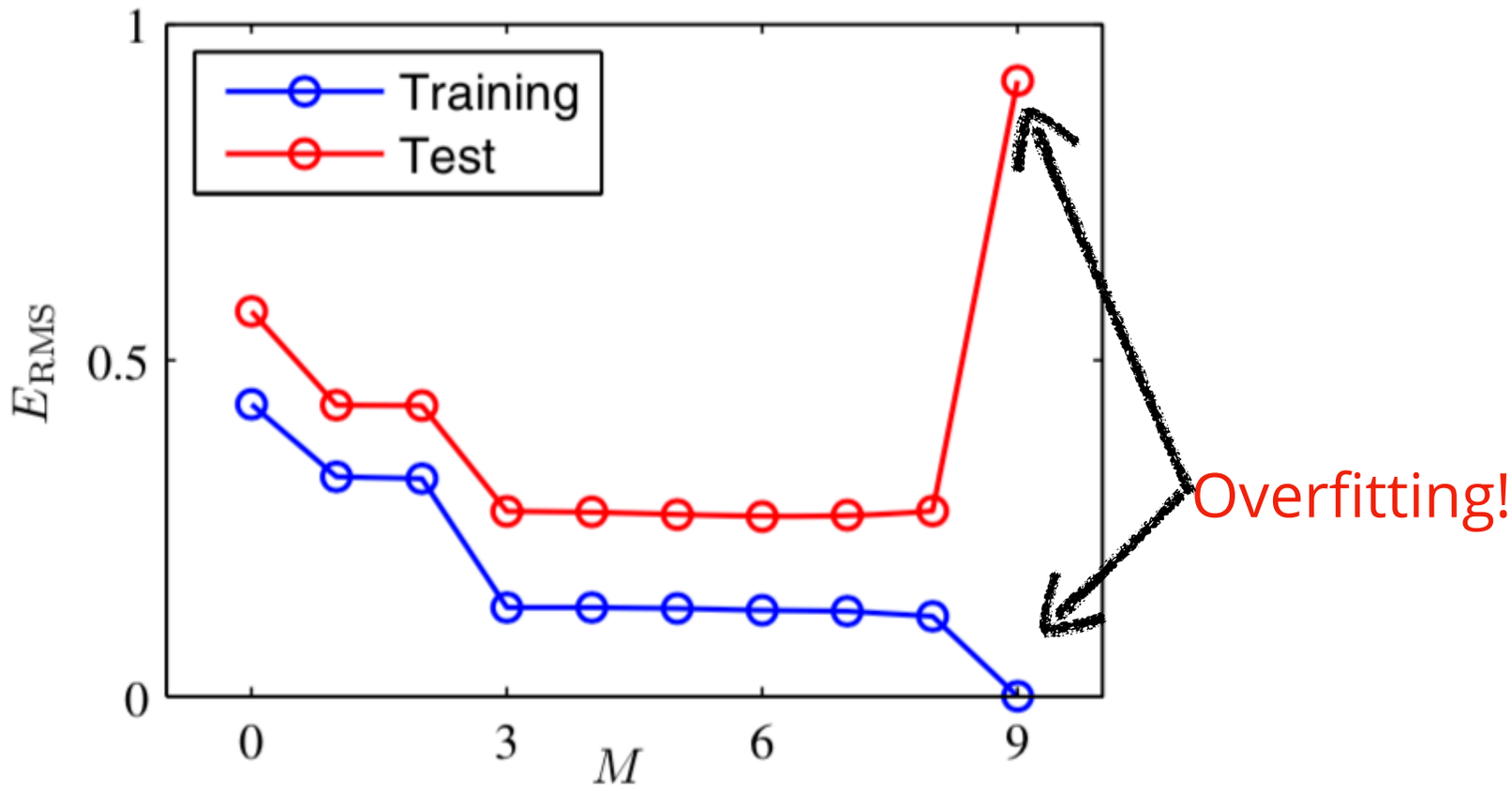
Overfitting!

Example of learning: polynomial curve fitting



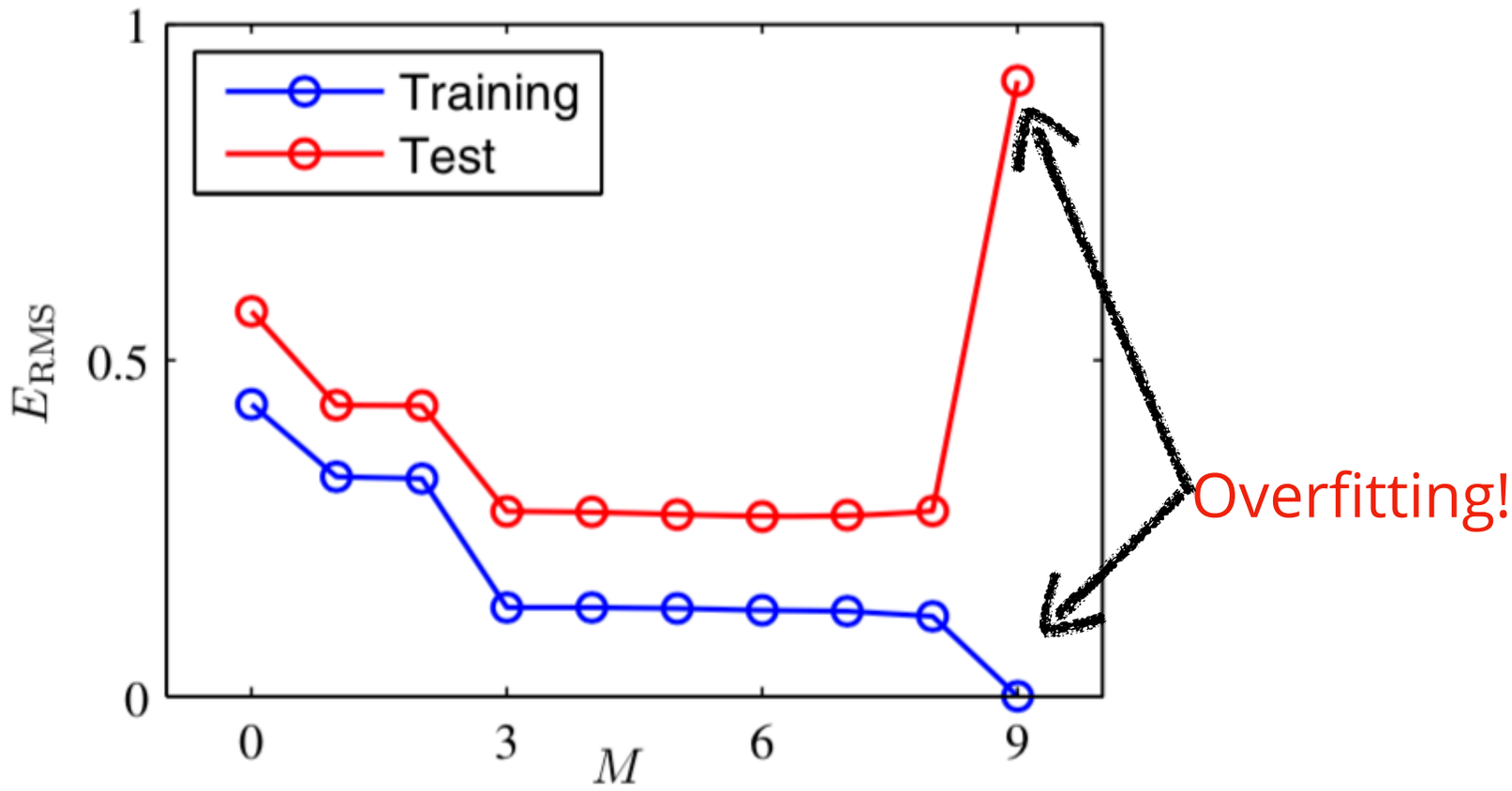
Root Mean Square error $R_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

Example of learning: polynomial curve fitting



Root Mean Square error $R_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

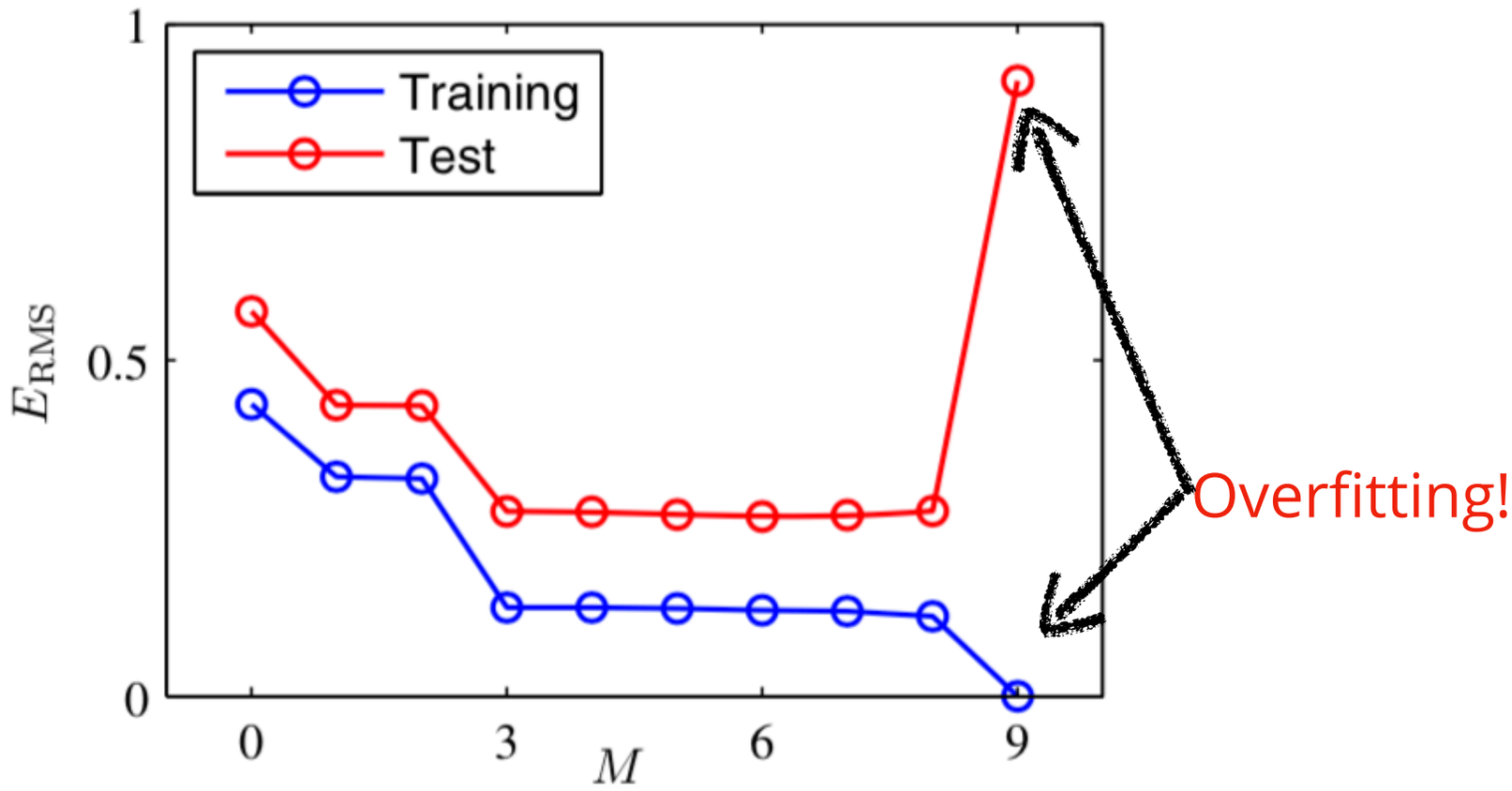
Example of learning: polynomial curve fitting



Root Mean Square error $R_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

Why overfitting?

Example of learning: polynomial curve fitting

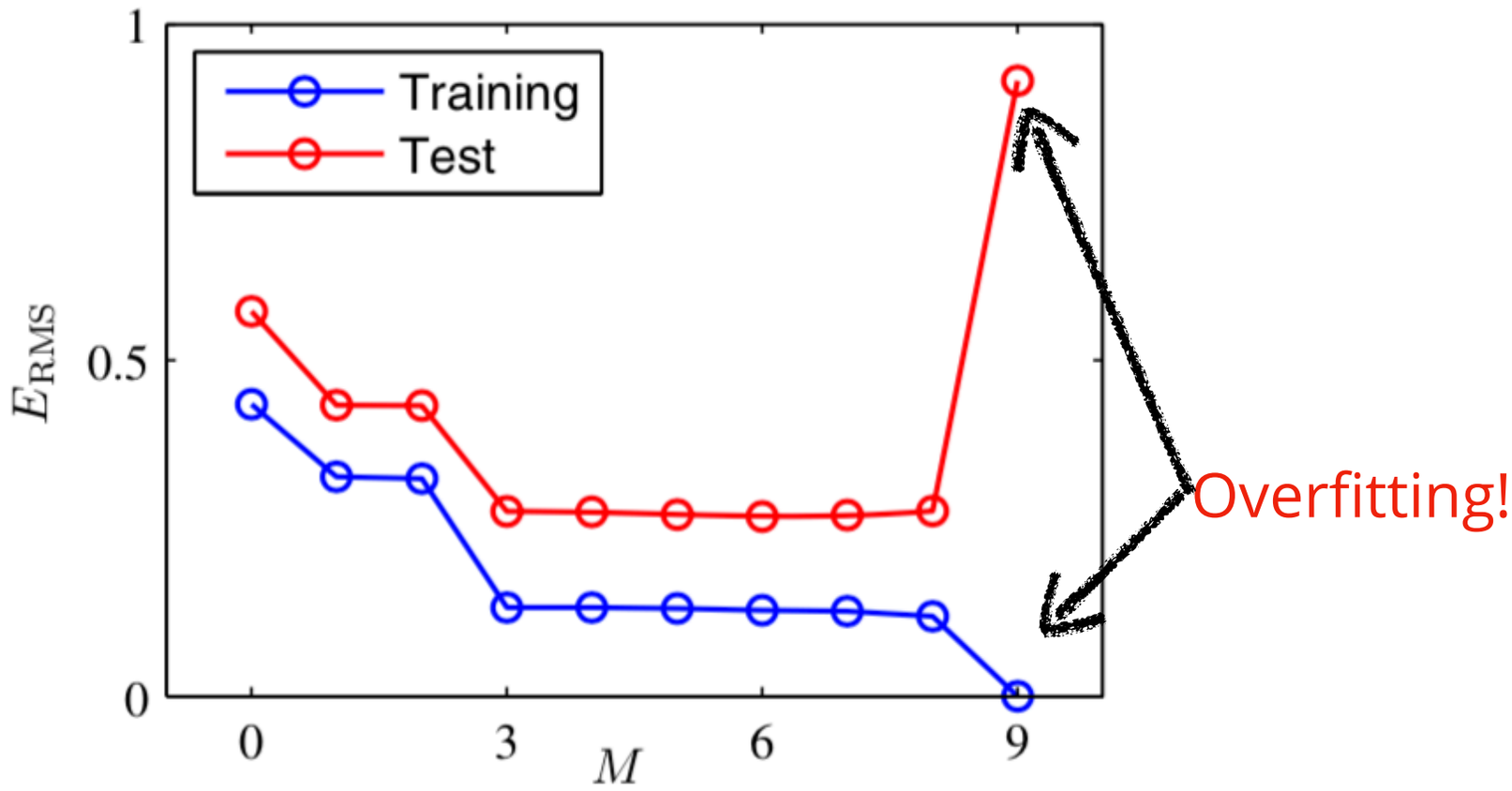


Root Mean Square error $R_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

Why overfitting?

- (effectively) over-parametrized

Example of learning: polynomial curve fitting

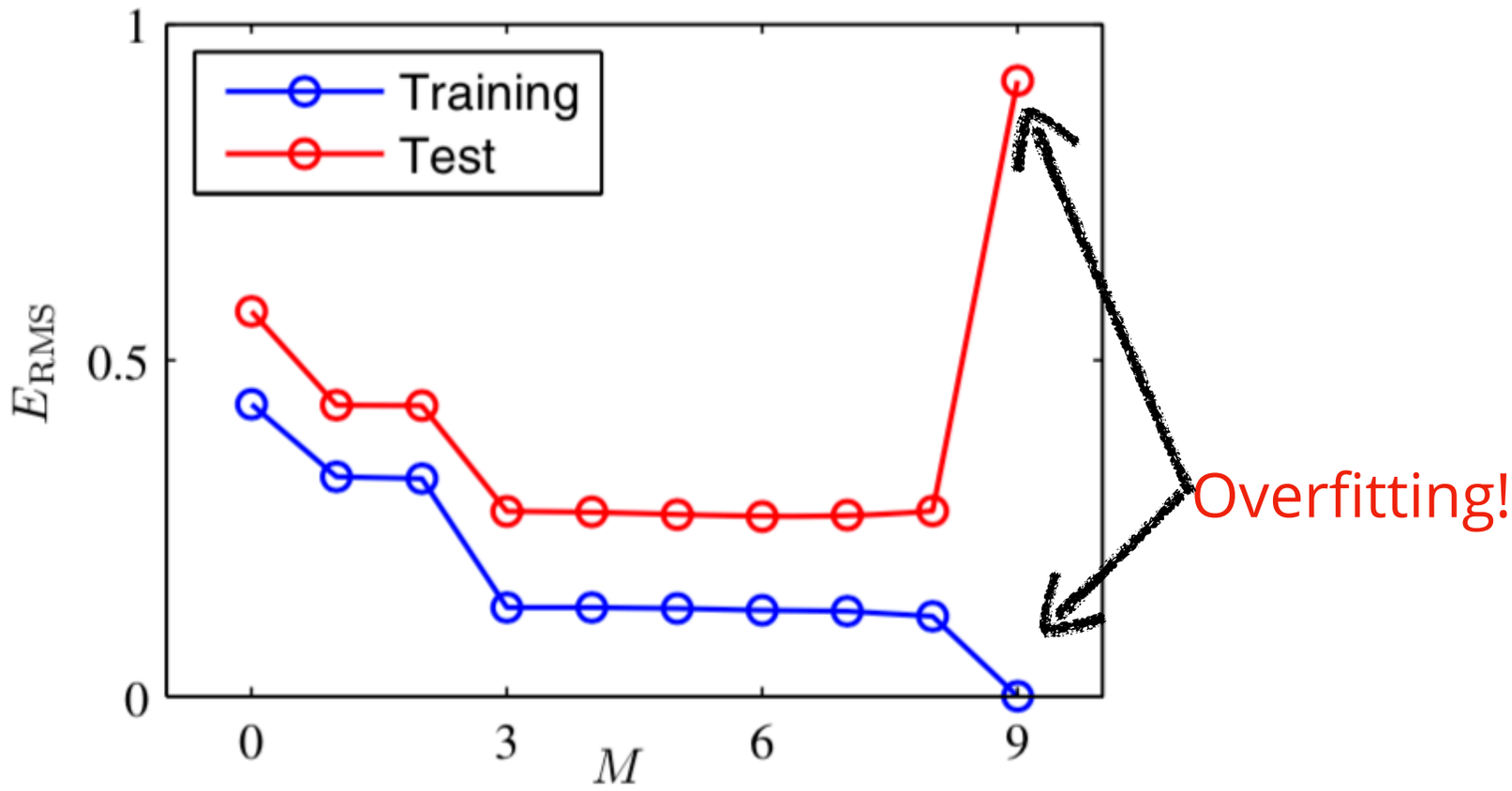


Root Mean Square error $R_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

Why overfitting?

- (effectively) over-parametrized
- Maximum-likelihood

Example of learning: polynomial curve fitting



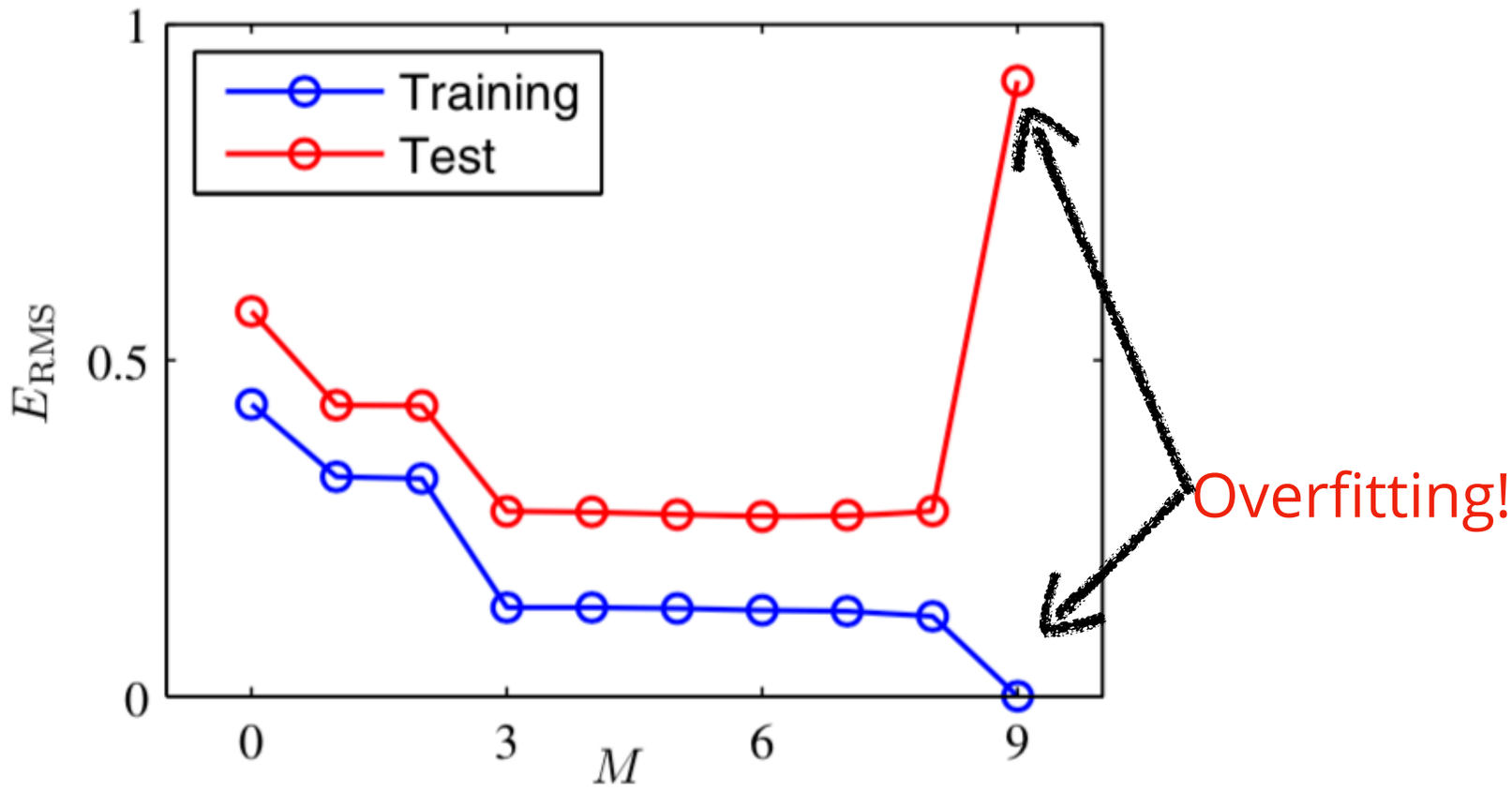
Root Mean Square error $R_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

Why overfitting?

- (effectively) over-parametrized
- Maximum-likelihood

How to alleviate it?

Example of learning: polynomial curve fitting



Root Mean Square error $R_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

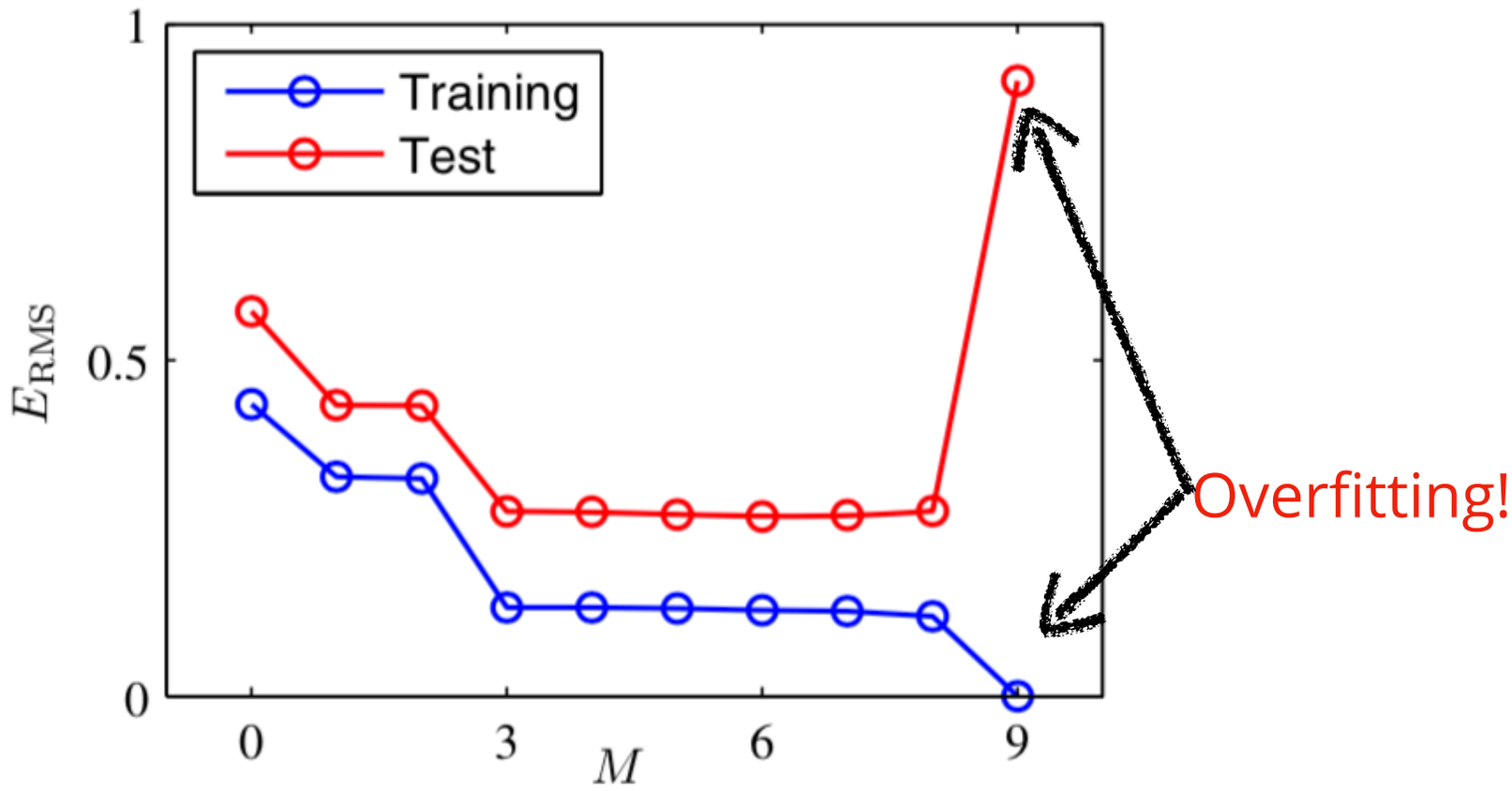
Why overfitting?

- (effectively) over-parametrized
- Maximum-likelihood

How to alleviate it?

- adding regularizations

Example of learning: polynomial curve fitting



Root Mean Square error $R_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

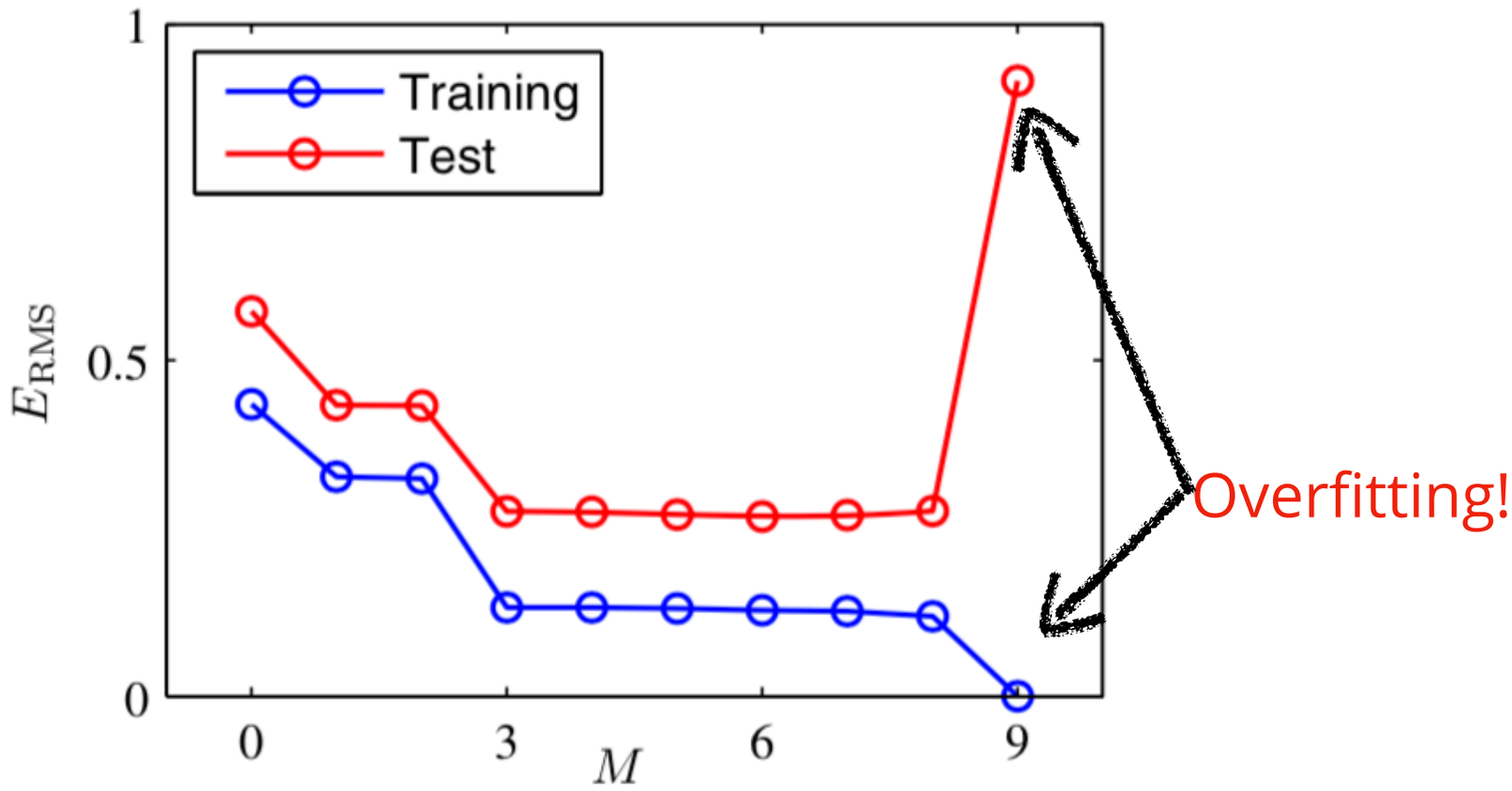
Why overfitting?

- (effectively) over-parametrized
- Maximum-likelihood

How to alleviate it?

- adding regularizations
- using more data

Example of learning: polynomial curve fitting



Root Mean Square error $R_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

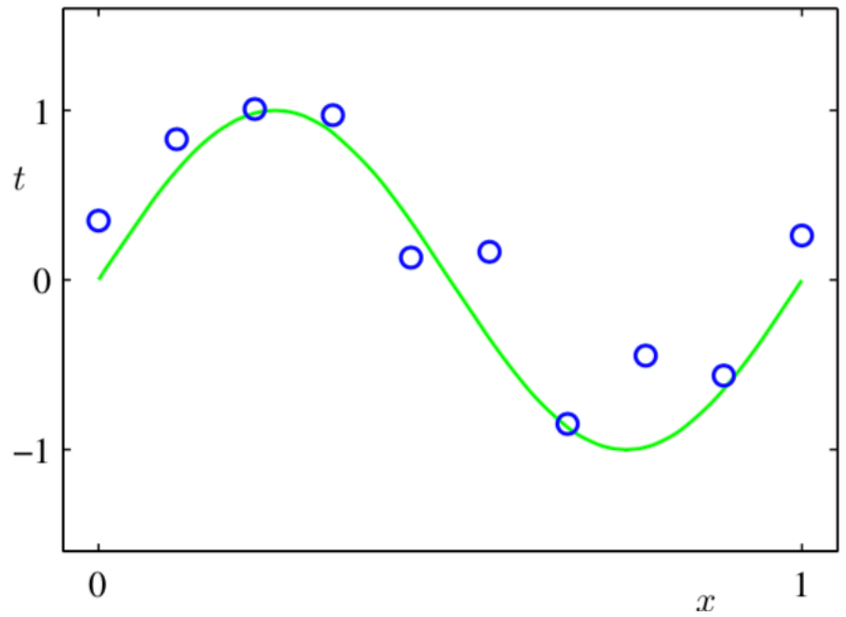
Why overfitting?

- (effectively) over-parametrized
- Maximum-likelihood

How to alleviate it?

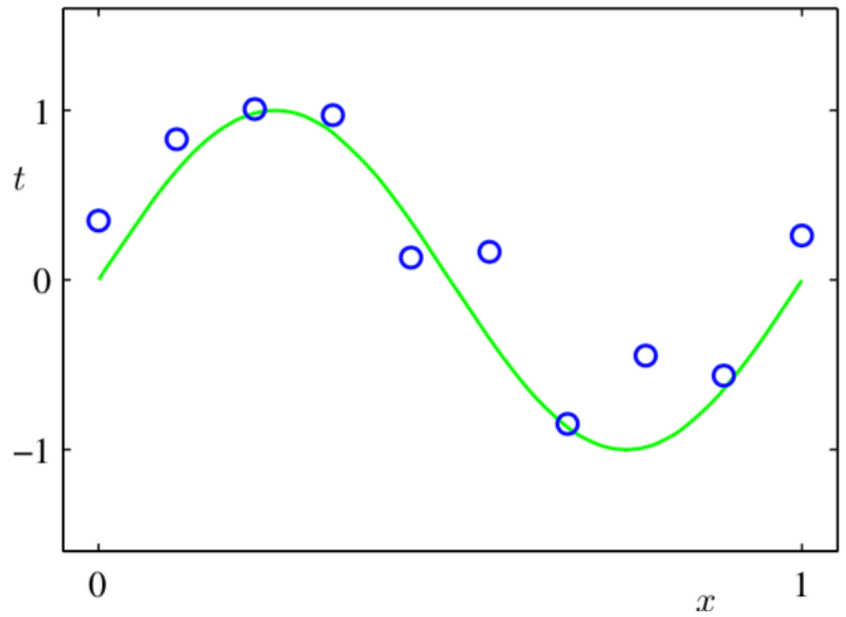
- adding regularizations
- using more data
- Bayesian

Probabilistic point of view



to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

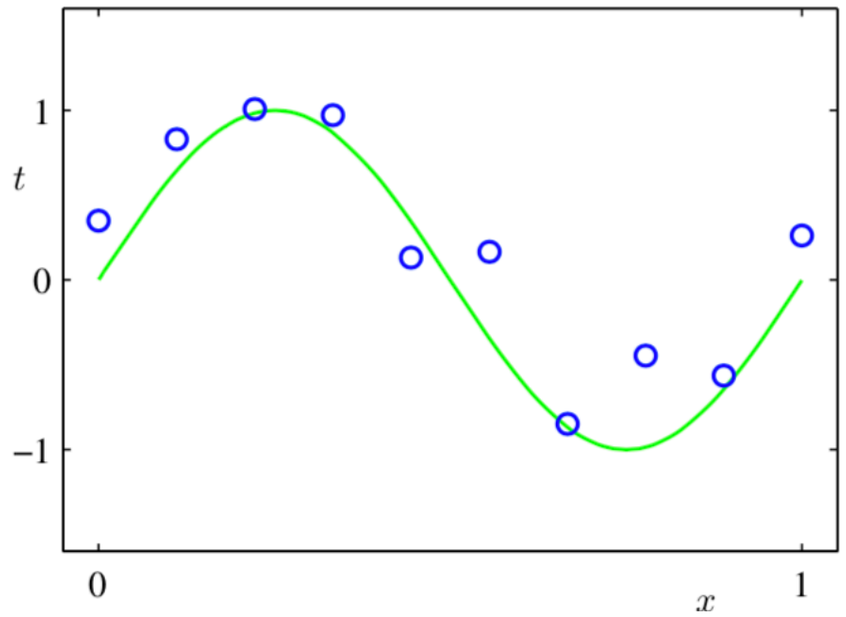
Probabilistic point of view



to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

$$P(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1})$$

Probabilistic point of view

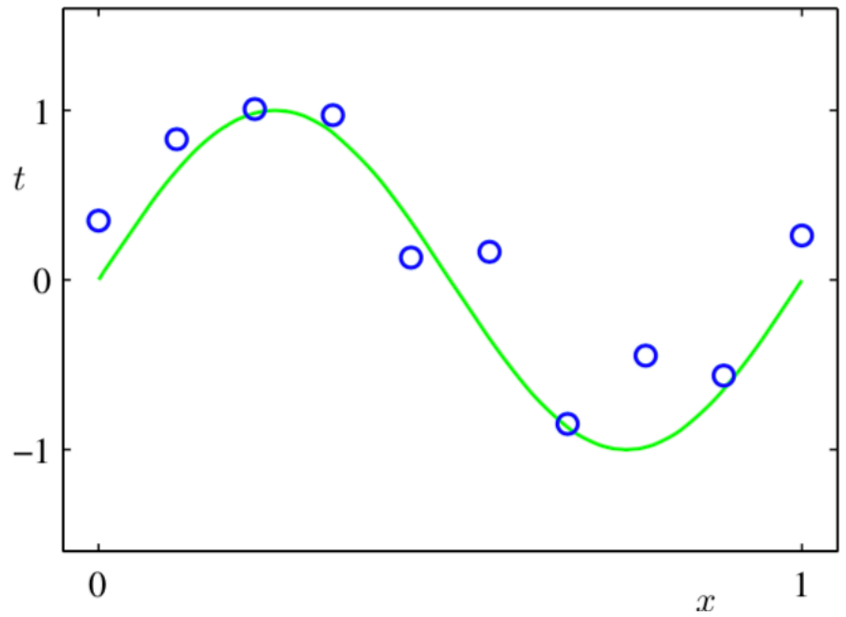


to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

$$P(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1})$$

$$P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(x_i, \mathbf{w}), \beta^{-1})$$

Probabilistic point of view



to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

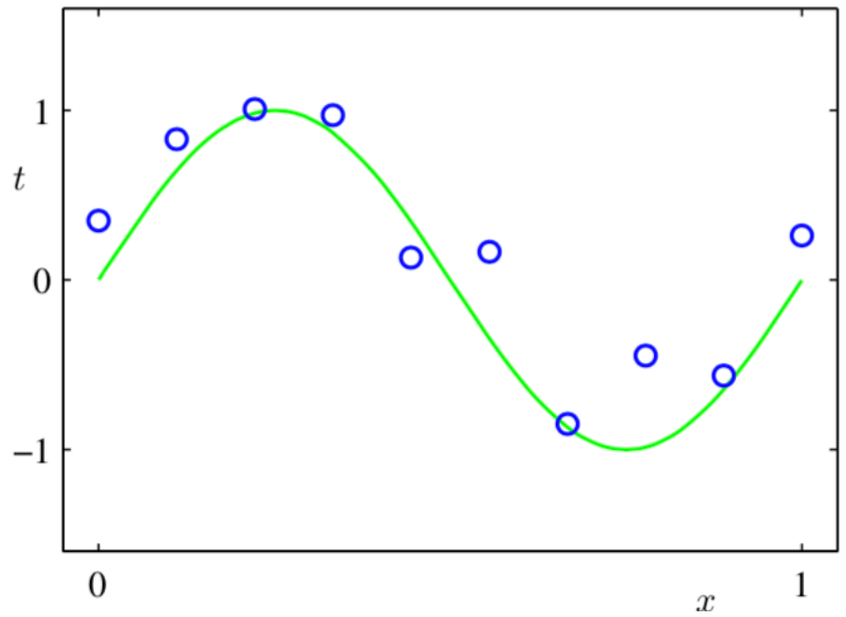
$$P(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1})$$

$$P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(x_i, \mathbf{w}), \beta^{-1})$$

Bayesian estimate

$$P(\mathbf{w}|\mathbf{x}, \mathbf{t}, \beta) = \frac{P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})}{\int d\mathbf{w} P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})}$$

Probabilistic point of view



to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

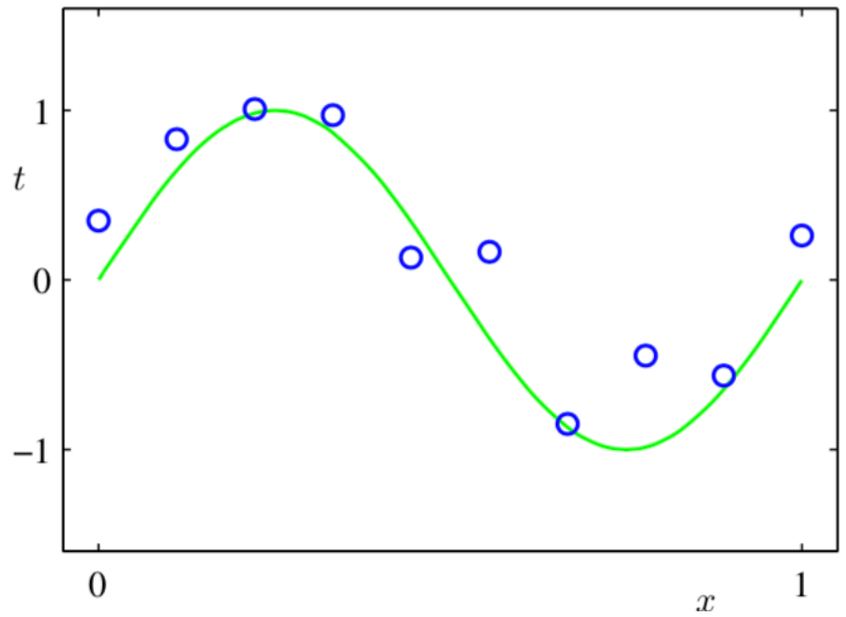
$$P(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1})$$

$$P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(x_i, \mathbf{w}), \beta^{-1})$$

Bayesian estimate $$P(\mathbf{w}|\mathbf{x}, \mathbf{t}, \beta) = \frac{P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})}{\int d\mathbf{w} P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})}$$

Maximum A Posterior $$\mathbf{w}_{M.A.P.} = \arg\max_{\mathbf{w}} \log\left[P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})\right]$$

Probabilistic point of view



to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

$$P(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1})$$

$$P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(x_i, \mathbf{w}), \beta^{-1})$$

Bayesian estimate

$$P(\mathbf{w}|\mathbf{x}, \mathbf{t}, \beta) = \frac{P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})}{\int d\mathbf{w} P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})}$$

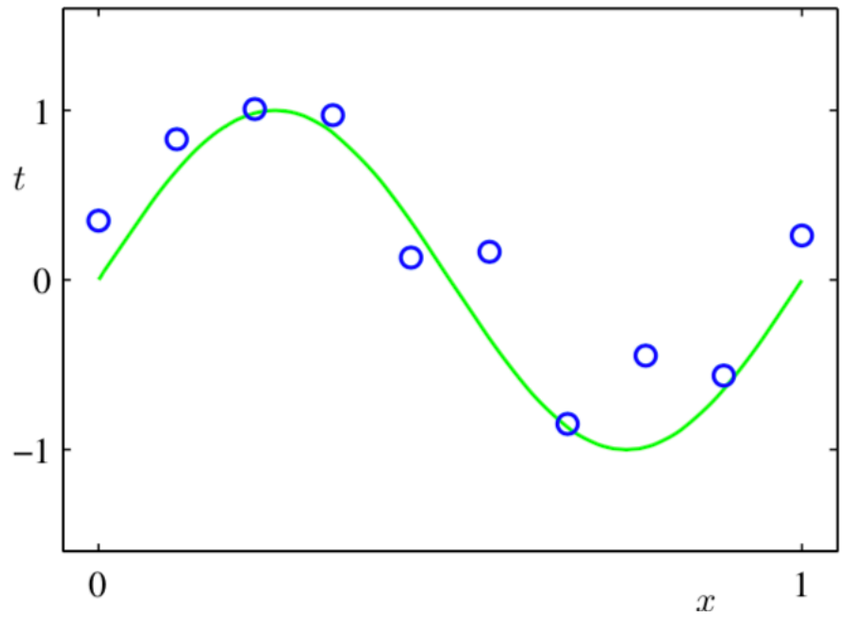
Maximum A Posterior

$$\mathbf{w}_{M.A.P.} = \arg\max_{\mathbf{w}} \log \left[P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})\right]$$

Maximum Likelihood

$$\mathbf{w}_{ML} = \arg\max_{\mathbf{w}} \log P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)$$

Maximum likelihood and least squares



to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

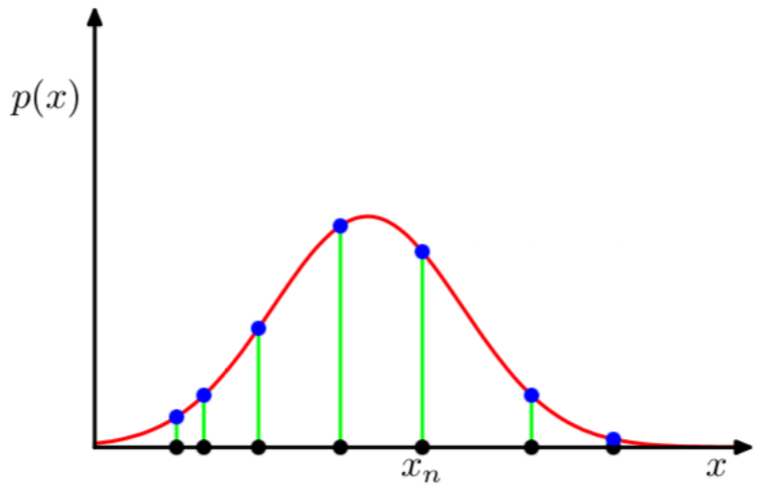
$$\log P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(x_i, \mathbf{w}), \beta^{-1})$$

$$= -\frac{\beta}{2}\sum_{i=1}^{N}(y(x_i, \mathbf{w}) - t_i)^2 + \frac{N}{2}\log\beta - \frac{N}{2}\log(2\pi)$$

$$\mathbf{w}_{ML} = \arg\min_{\mathbf{w}} \log P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)$$

$$= \arg\min_{\mathbf{w}} \frac{\beta}{2}\sum_{i=1}^{N}(y(x_i, \mathbf{w}) - t_i)^2$$

Maximum likelihood and minimizing KL divergence



Data points are generated i.i.d.
Its empirical distribution is assumed to be

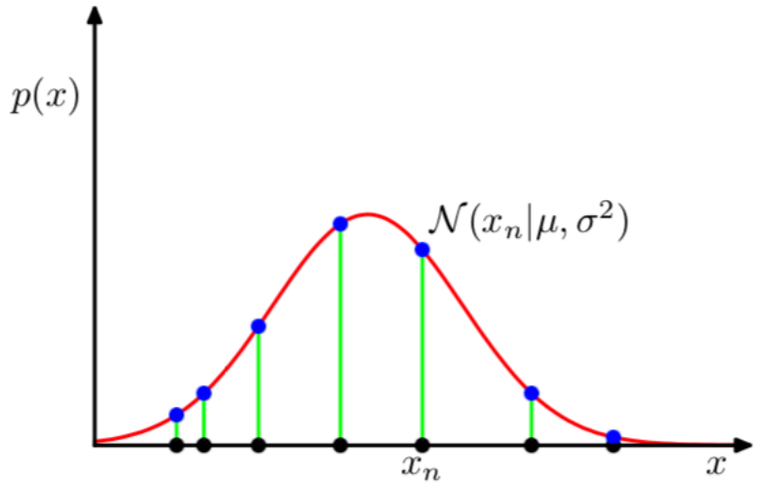
$$P_{\text{data}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x} - \mathbf{x}_{\text{data}}^{(\mathbf{i})})$$

Density estimation fits it using $q_\theta(\mathbf{x})$ by minimizing

$$\begin{aligned} D_{\text{KL}}(P_{\text{data}}\|q_\theta) &= \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{P_{\text{data}}\mathbf{x}}{\log q_\theta(\mathbf{x})} \\ &= -\sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log q_\theta(\mathbf{x}) - S_P \\ &= -\sum_{x\in\text{data}} \log q_\theta(\mathbf{x}) - S_P \end{aligned}$$

Observe that $q_\theta(\mathbf{x})$ has to be narrower than the underlying distribution.

Limitation of the maximum likelihood estimation



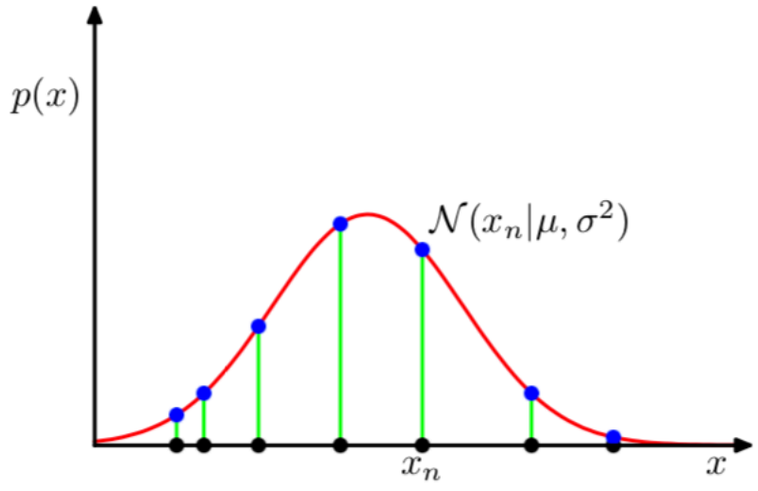
Data points are generated i.i.d.

Its empirical distribution is assumed to be

$$P_{\text{data}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x} - \mathbf{x}^{(\mathbf{i})}_{\text{data}})$$

*Image courtesy: Bishop **PRML** 2006*

Limitation of the maximum likelihood estimation



Data points are generated i.i.d.
Its empirical distribution is assumed to be

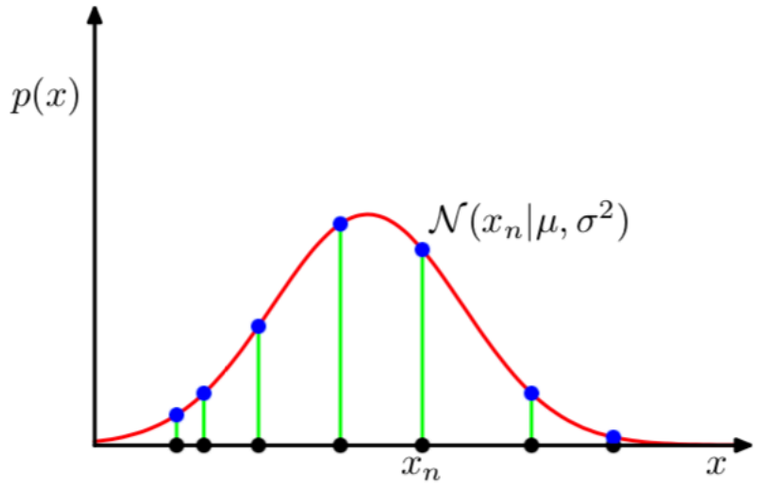
$$P_{\text{data}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x} - \mathbf{x}^{(\mathbf{i})}_{\text{data}})$$

When Gaussian distribution is used

$$q_\theta(\mathbf{x}, \mu, \sigma^2) = \prod_{n=1}^{N}\mathcal{N}(x_n|\mu, \sigma^2)$$

*Image courtesy: Bishop **PRML** 2006*

Limitation of the maximum likelihood estimation



Data points are generated i.i.d.
Its empirical distribution is assumed to be

$$P_{\text{data}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x} - \mathbf{x}_{\text{data}}^{(\mathbf{i})})$$

When Gaussian distribution is used

$$q_\theta(\mathbf{x}, \mu, \sigma^2) = \prod_{n=1}^{N}\mathcal{N}(x_n|\mu, \sigma^2)$$

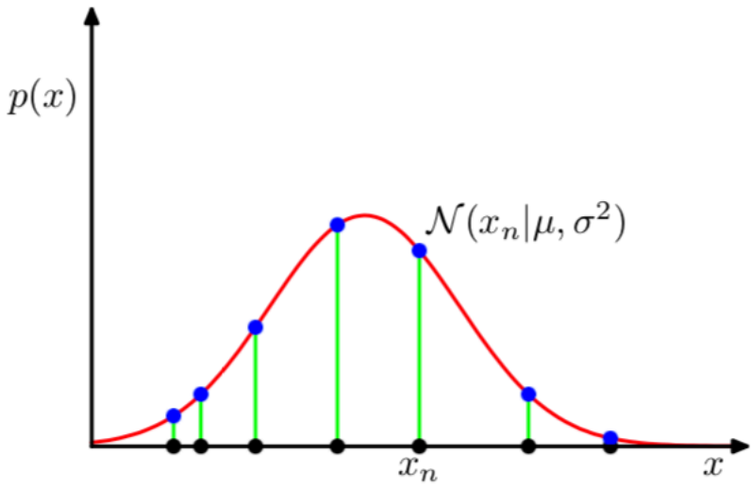
By maximizing the likelihood:

$$\mathbb{E}[\mu_{\text{ML}}] = \mu_{\text{true}}$$

$$\mathbb{E}[\sigma^2_{\text{ML}}] = \left(\frac{N-1}{N}\right)\sigma^2_{\text{true}}$$

*Image courtesy: Bishop **PRML** 2006*

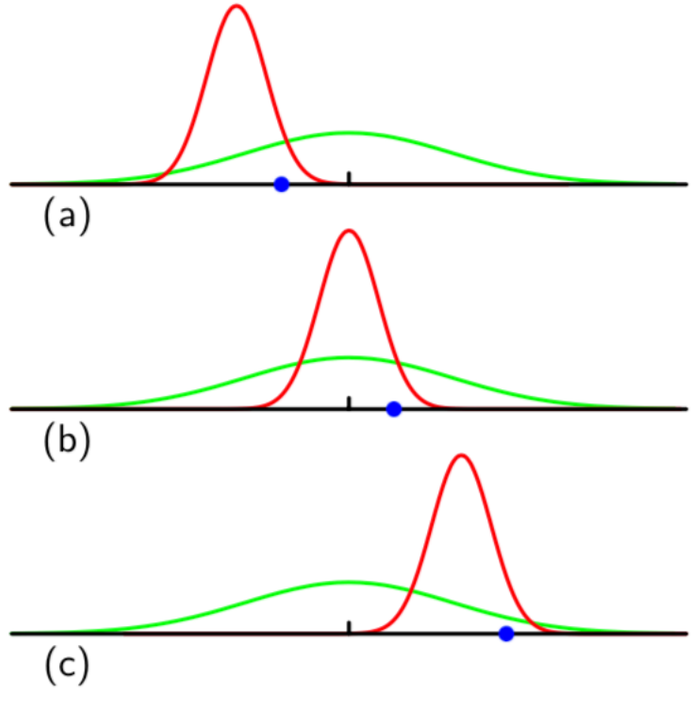
Limitation of the maximum likelihood estimation



Data points are generated i.i.d.

Its empirical distribution is assumed to be

$$P_{\text{data}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x} - \mathbf{x}_{\text{data}}^{(\mathbf{i})})$$



When Gaussian distribution is used

$$q_\theta(\mathbf{x}, \mu, \sigma^2) = \prod_{n=1}^{N}\mathcal{N}(x_n|\mu, \sigma^2)$$

By maximizing the likelihood:

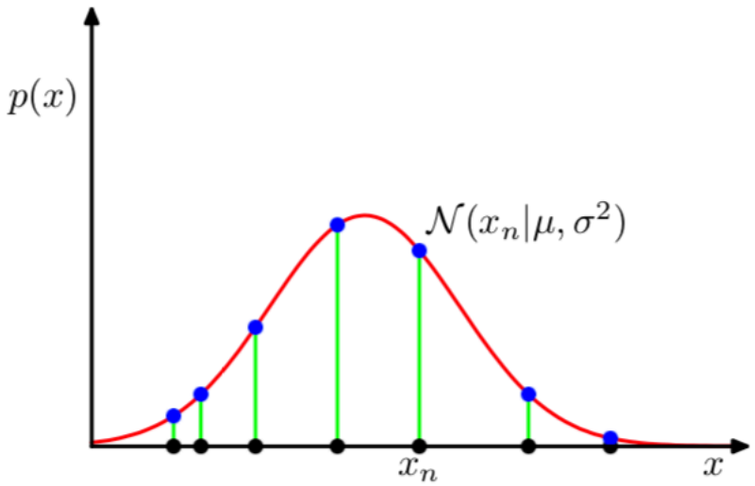
$$\mathbb{E}[\mu_{\text{ML}}] = \mu_{\text{true}}$$

$$\mathbb{E}[\sigma^2_{\text{ML}}] = \left(\frac{N-1}{N}\right)\sigma^2_{\text{true}}$$

bias: systematically underestimate the variance

*Image courtesy: Bishop **PRML** 2006*

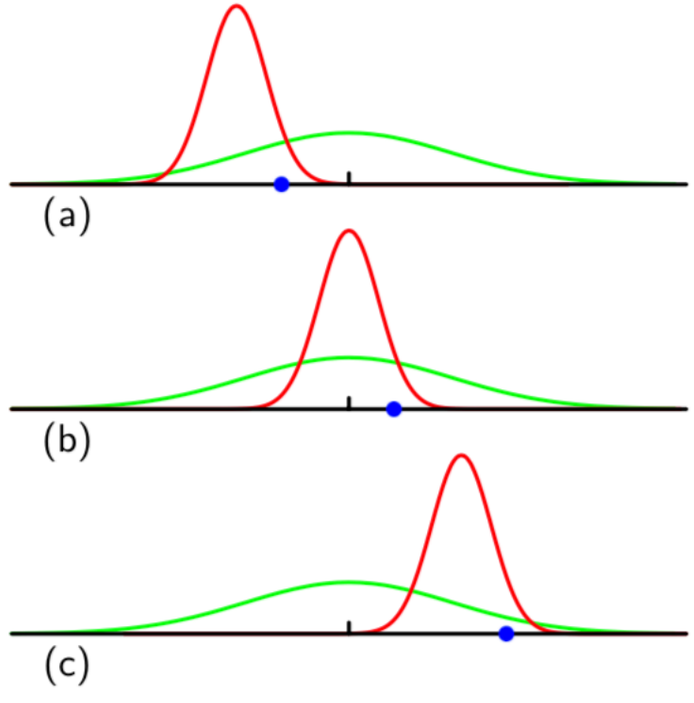
Limitation of the maximum likelihood estimation



Data points are generated i.i.d.

Its empirical distribution is assumed to be

$$P_{\text{data}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x} - \mathbf{x}_{\text{data}}^{(\mathbf{i})})$$



When Gaussian distribution is used

$$q_\theta(\mathbf{x}, \mu, \sigma^2) = \prod_{n=1}^{N}\mathcal{N}(x_n|\mu, \sigma^2)$$

By maximizing the likelihood:

$$\mathbb{E}[\mu_{\text{ML}}] = \mu_{\text{true}}$$

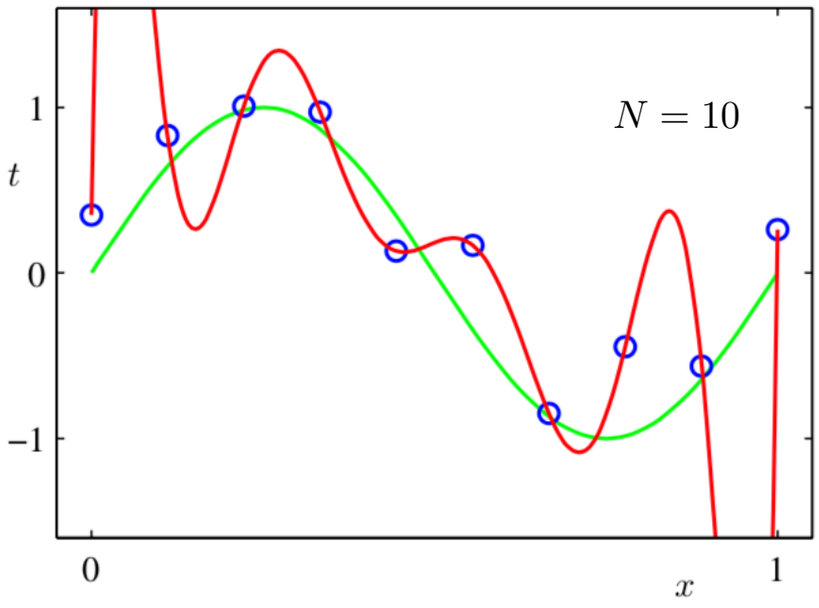
$$\mathbb{E}[\sigma^2_{\text{ML}}] = \left(\frac{N-1}{N}\right)\sigma^2_{\text{true}}$$

bias: systematically underestimate the variance

*Image courtesy: Bishop **PRML** 2006*

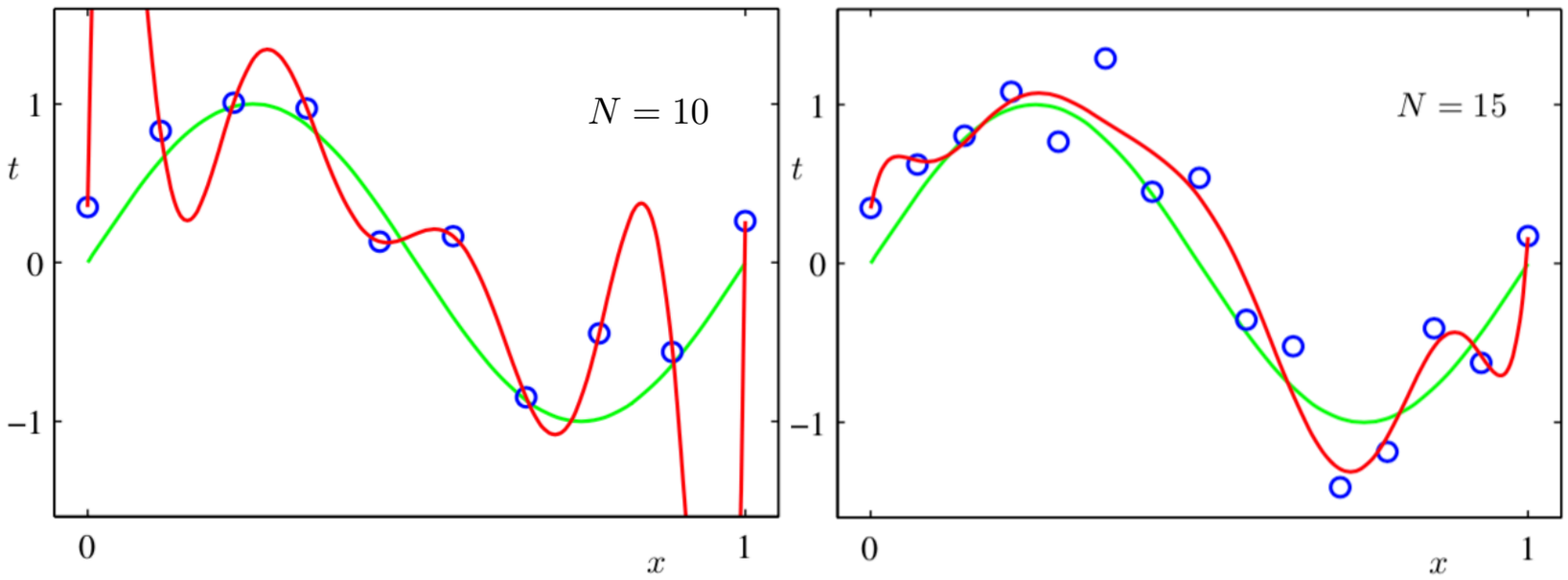
How to connect this property with KL divergence?

One way to alleviate: adding more data !



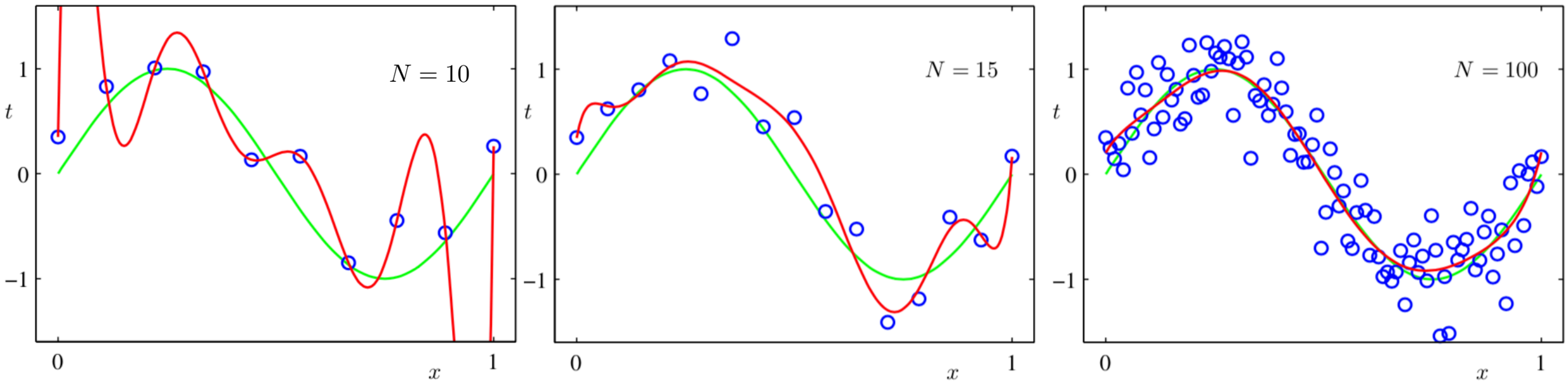
Fitting more data points with M=9 order polynomials

One way to alleviate: adding more data !



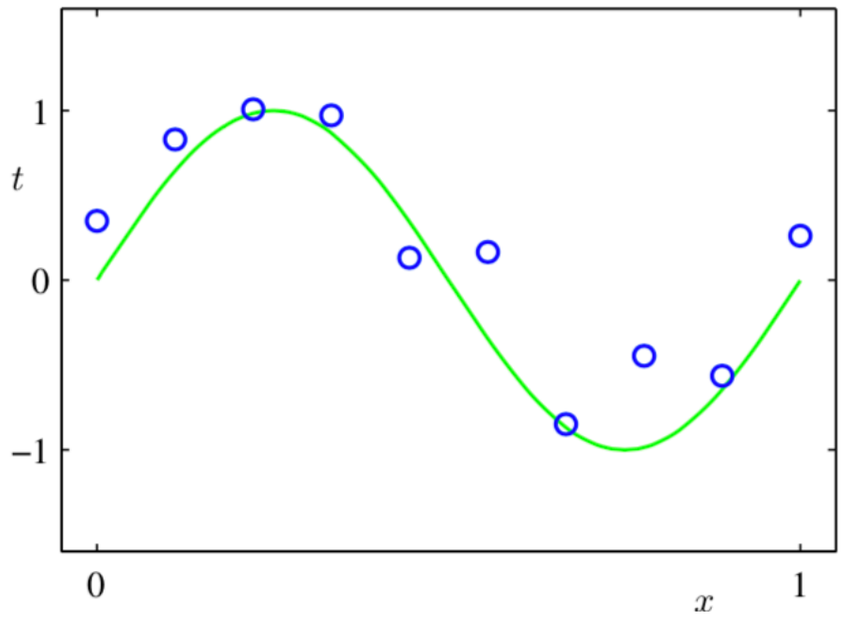
Fitting more data points with M=9 order polynomials

One way to alleviate: adding more data !



Fitting more data points with M=9 order polynomials

Another way to alleviate: regularization and M.A.P.

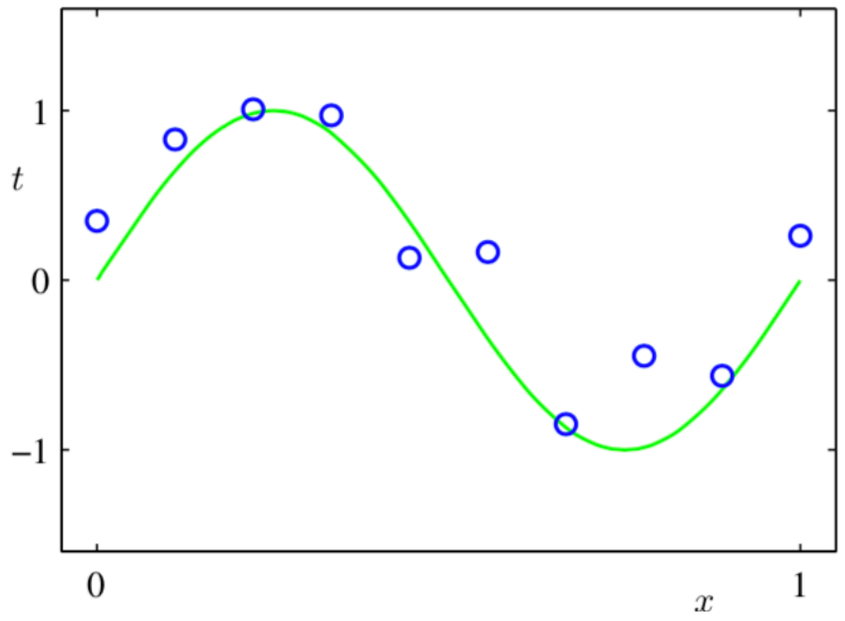


to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

$$\log P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(x_i, \mathbf{w}), \beta^{-1})$$

$$= -\frac{\beta}{2}\sum_{i=1}^{N}(y(x_i, \mathbf{w}) - t_i)^2 + \frac{N}{2}\log\beta - \frac{N}{2}\log(2\pi)$$

Another way to alleviate: regularization and M.A.P.



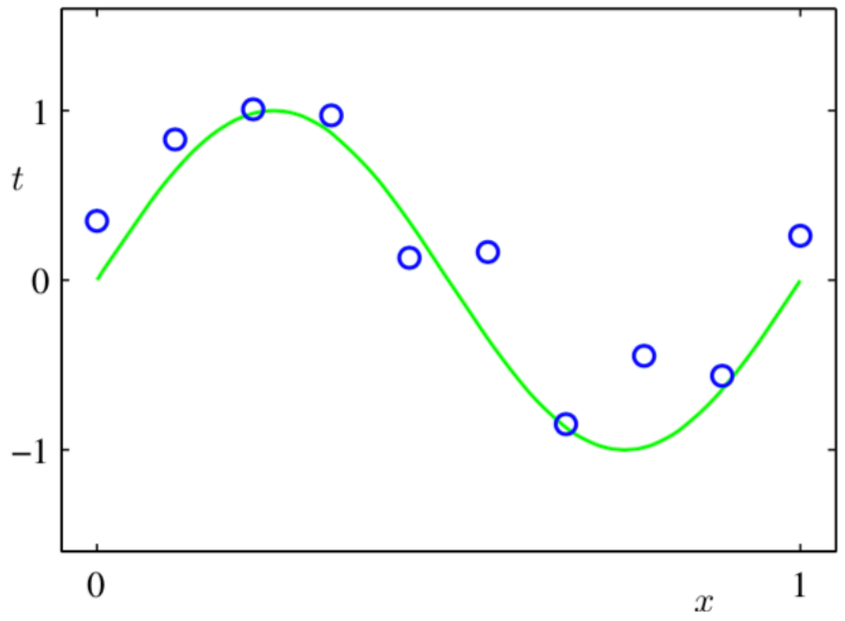
to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

$$\log P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(x_i, \mathbf{w}), \beta^{-1})$$

$$= -\frac{\beta}{2}\sum_{i=1}^{N}(y(x_i, \mathbf{w}) - t_i)^2 + \frac{N}{2}\log\beta - \frac{N}{2}\log(2\pi)$$

$$P(\mathbf{W}) = \prod_{i} \mathcal{N}(0, \lambda^{-1})$$

Another way to alleviate: regularization and M.A.P.



to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

$$\log P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(x_i, \mathbf{w}), \beta^{-1})$$

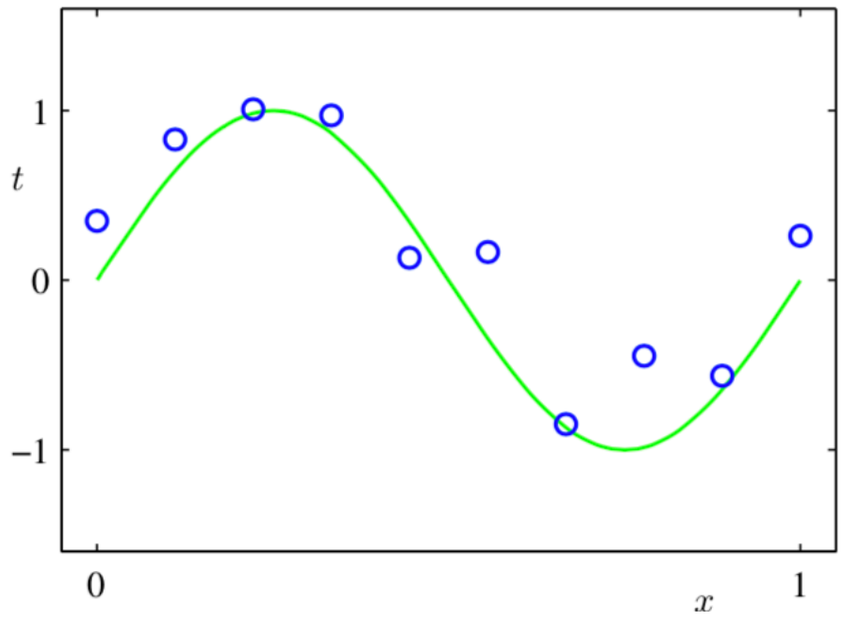
$$= -\frac{\beta}{2}\sum_{i=1}^{N}(y(x_i, \mathbf{w}) - t_i)^2 + \frac{N}{2}\log\beta - \frac{N}{2}\log(2\pi)$$

$$P(\mathbf{W}) = \prod_{i} \mathcal{N}(0, \lambda^{-1})$$

$$\mathbf{w}_{M.A.P.} = \arg\min_{\mathbf{w}} \log\left[P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})\right]$$

$$= \arg\min_{\mathbf{w}} \frac{\beta}{2}\sum_{i=1}^{N}(y(x_i, \mathbf{w}) - t_i)^2 - \frac{1}{2}\lambda\|\mathbf{w}\|_2^2$$

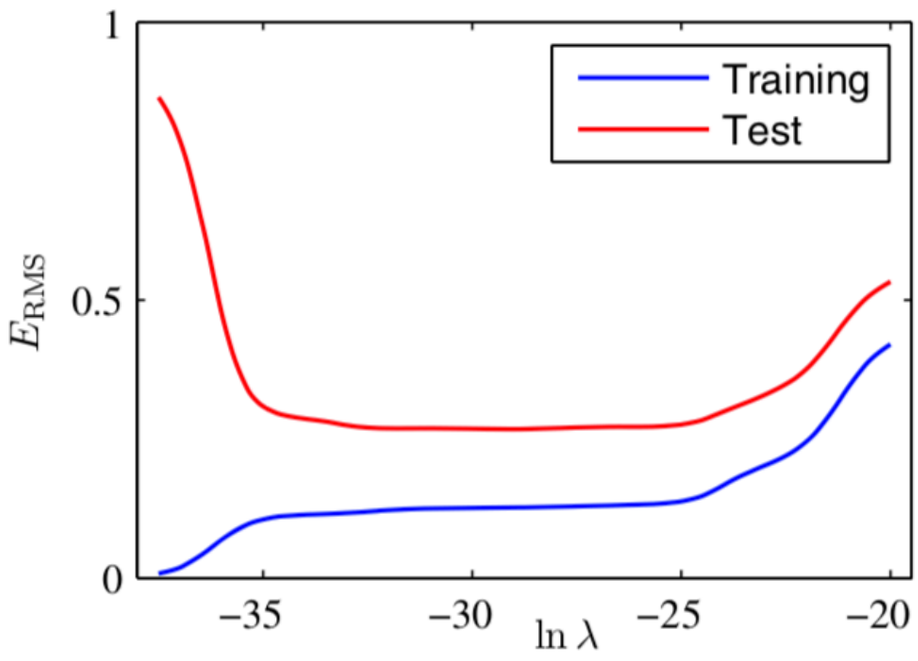
Another way to alleviate: regularization and M.A.P.



to predict $t(x)$ probabilistically, first choose a probabilistic (generative) model:

$$\log P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|y(x_i, \mathbf{w}), \beta^{-1})$$

$$= -\frac{\beta}{2}\sum_{i=1}^{N}(y(x_i, \mathbf{w}) - t_i)^2 + \frac{N}{2}\log\beta - \frac{N}{2}\log(2\pi)$$



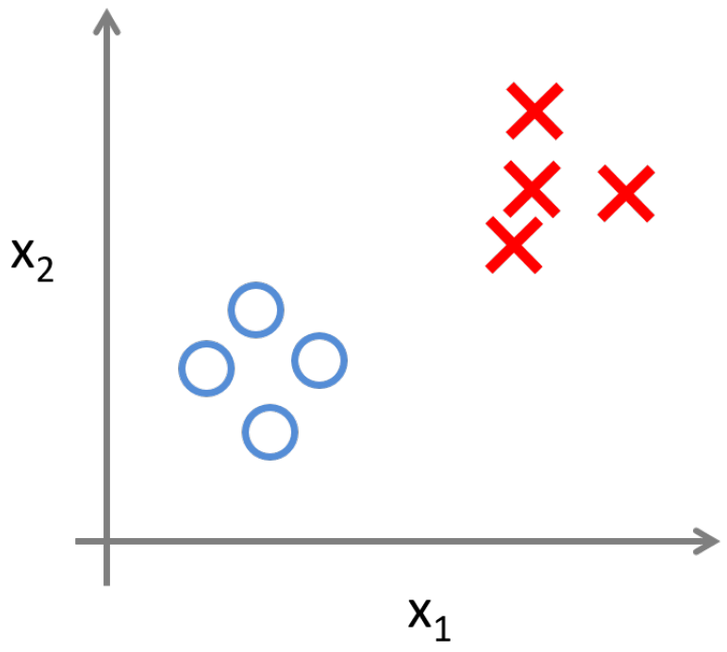
$$P(\mathbf{W}) = \prod_{i} \mathcal{N}(0, \lambda^{-1})$$

$$\mathbf{w}_{M.A.P.} = \arg\min_{\mathbf{w}} \log\left[P(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)P(\mathbf{w})\right]$$

$$= \arg\min_{\mathbf{w}} \frac{\beta}{2}\sum_{i=1}^{N}(y(x_i, \mathbf{w}) - t_i)^2 - \frac{1}{2}\lambda\|\mathbf{w}\|_2^2$$

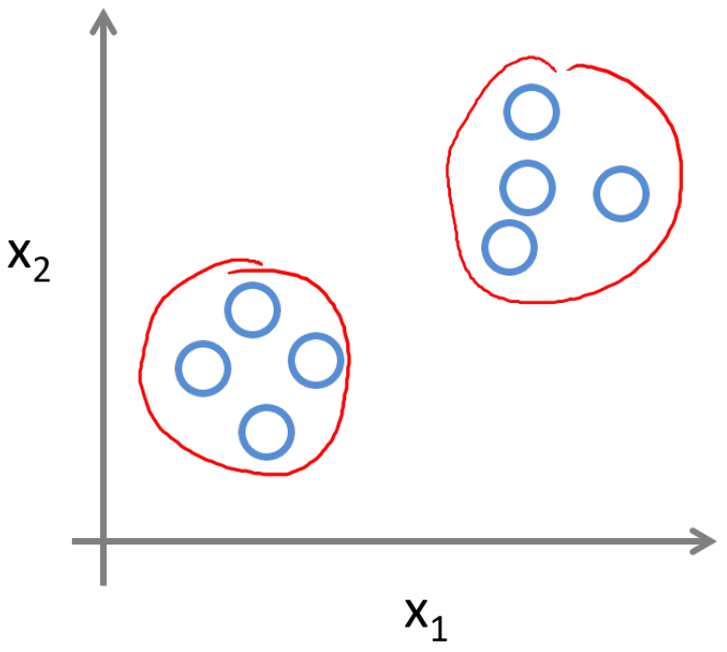
Learning problems

Supervised Learning



Predicting Labels

Unsupervised Learning



Finding structures in the data

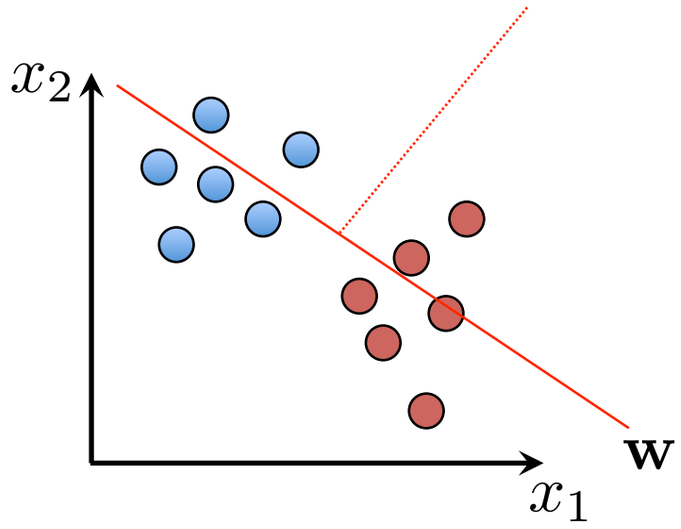
Linear classifier

Decision boundaries are linear functions of of input **x**, and hence are defined by *N-1* dimensional hyperplanes.

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

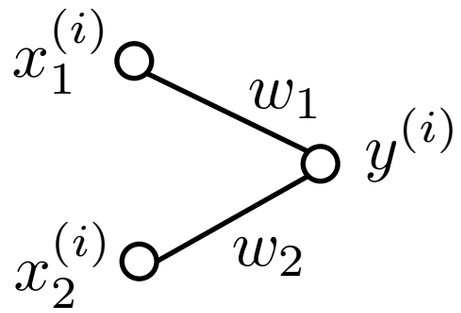
Linear classifier

Decision boundaries are linear functions of of input **x**, and hence are defined by *N-1* dimensional hyperplanes.



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

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

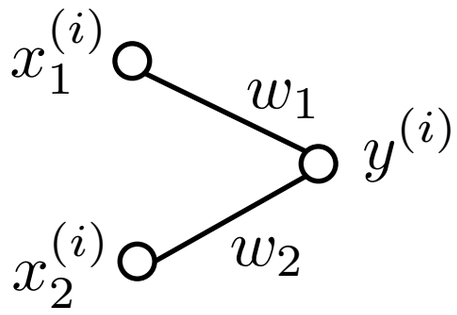


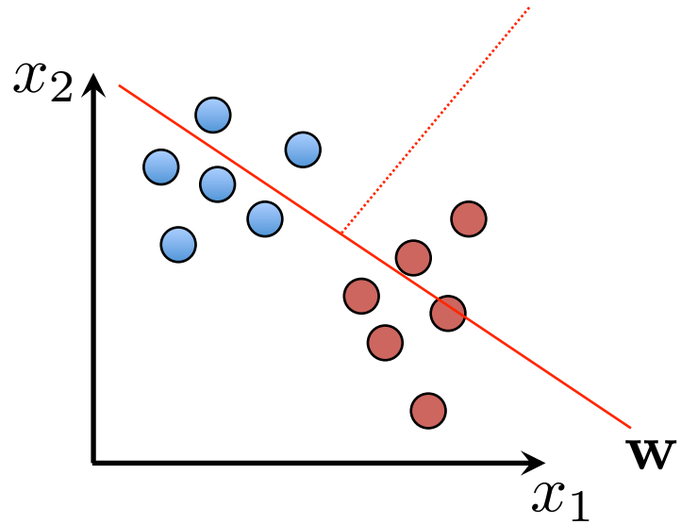
Linear classifier

Decision boundaries are linear functions of of input **x**, and hence are defined by *N-1* dimensional hyperplanes.

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

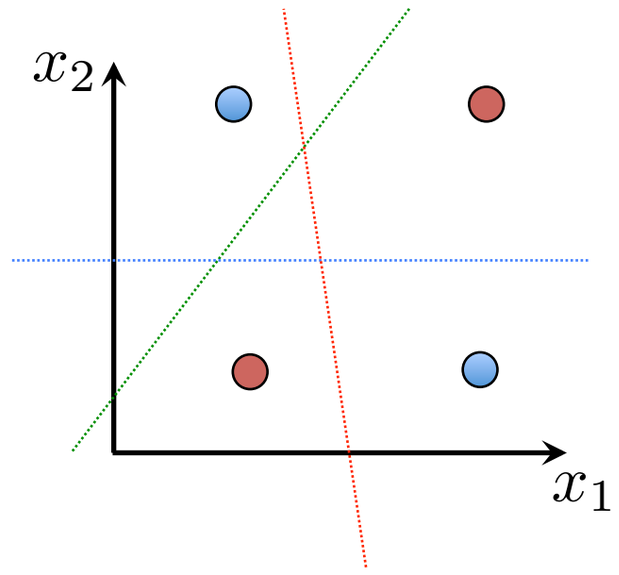
$$y^{(i)} = \Theta\left(w_1 x_1^{(i)} + w_2 x_2^{(i)} + b_i\right)$$





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

[Minsky, Paper 1969]

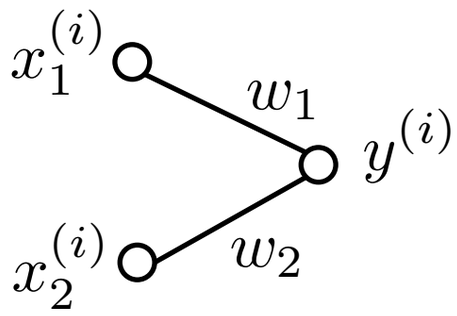


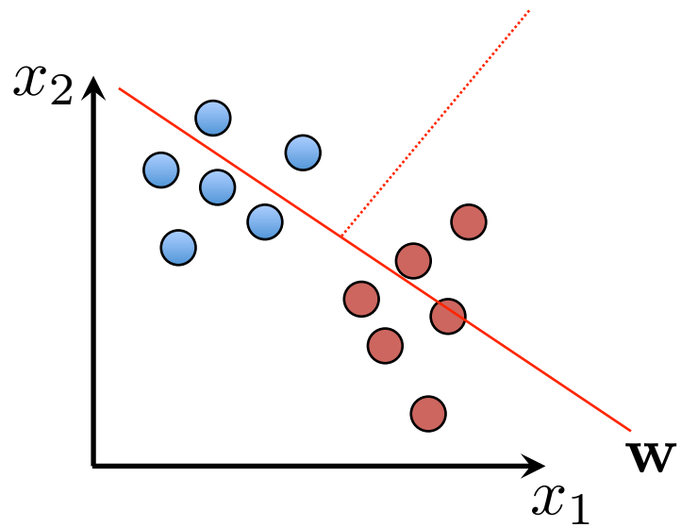
Linear classifier

Decision boundaries are linear functions of of input **x**, and hence are defined by *N-1* dimensional hyperplanes.

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

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



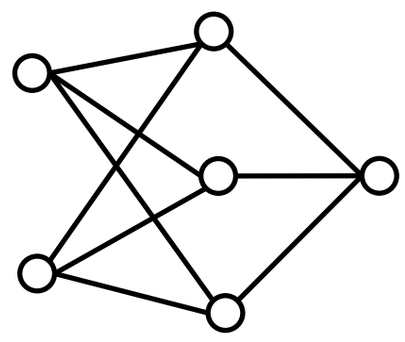


Sometimes not possible : not linear separable in 2-D.

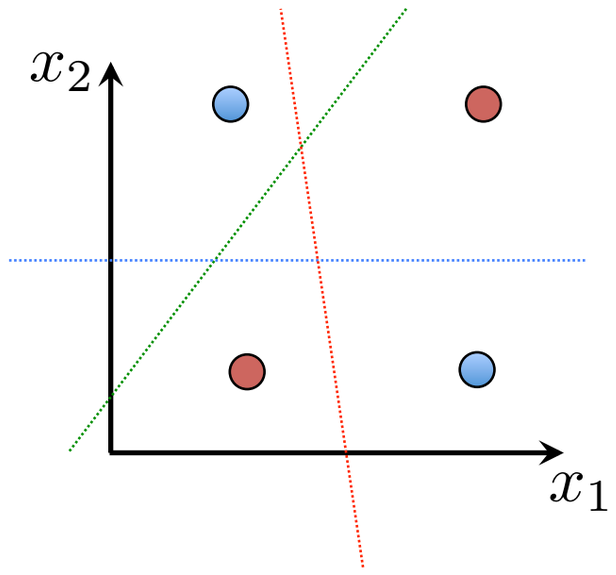
[Minsky, Paper 1969]

But could be linearly separable in a larger space

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

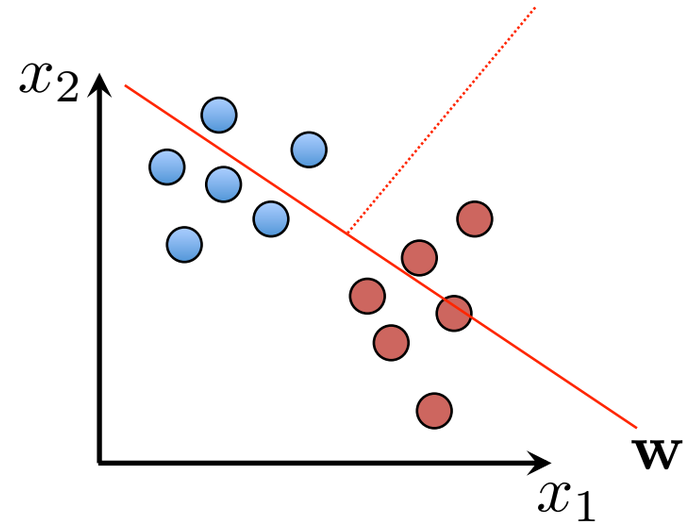


$\mathcal{R}^2 \rightarrow \mathcal{R}^3$



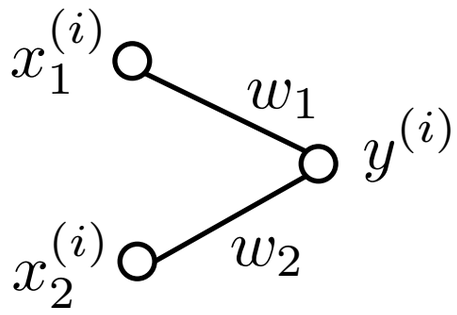
Linear classifier

Decision boundaries are linear functions of of input **x**, and hence are defined by *N-1* dimensional hyperplanes.



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

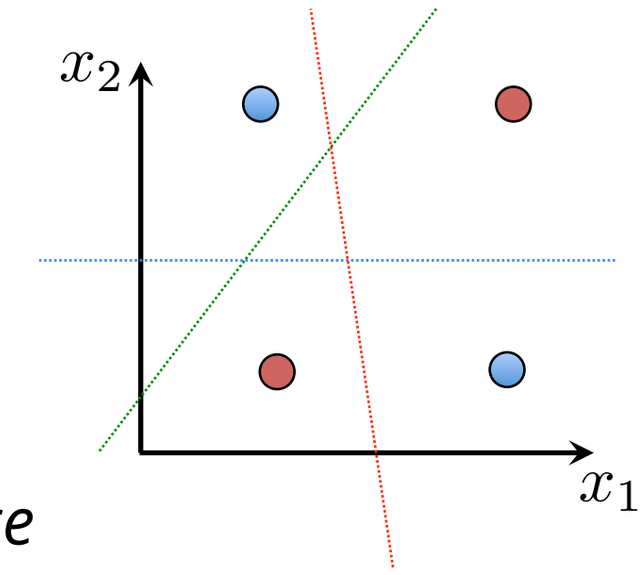
$$y^{(i)} = \Theta\left(w_1 x_1^{(i)} + w_2 x_2^{(i)} + b_i\right)$$



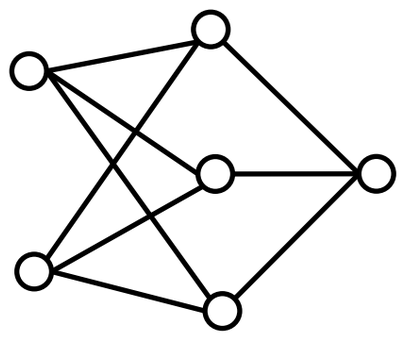
Sometimes not possible : not linear separable in 2-D.

[Minsky, Paper 1969]

But could be linearly separable in a larger space

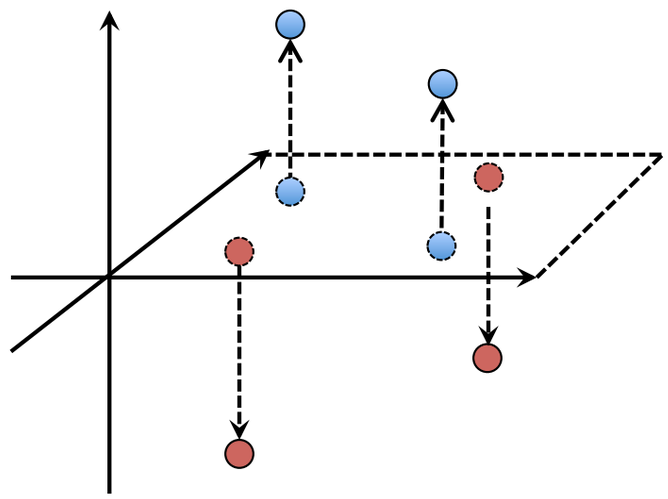


$$y^{(i)} = \Theta\left(\mathbf{W}\sigma(\mathbf{F}\mathbf{x}^{(i)}) + b_i\right)$$



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

Feature Space

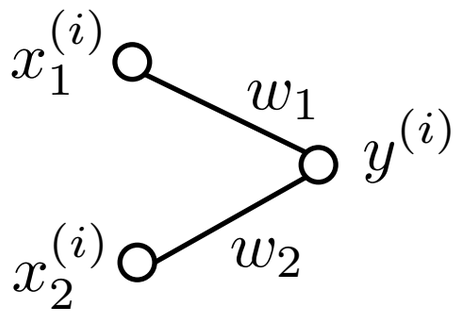


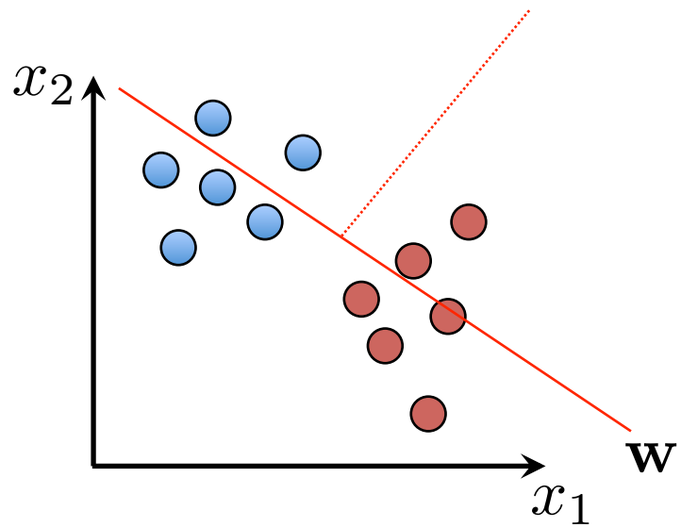
Linear classifier

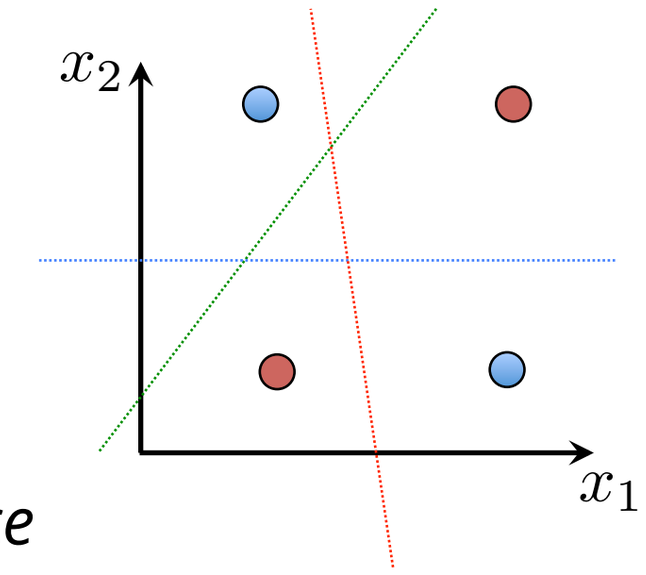
Decision boundaries are linear functions of of input **x**, and hence are defined by *N-1* dimensional hyperplanes.

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

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





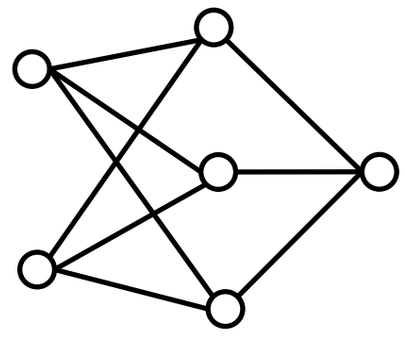


Sometimes not possible : not linear separable in 2-D.

[Minsky, Paper 1969]

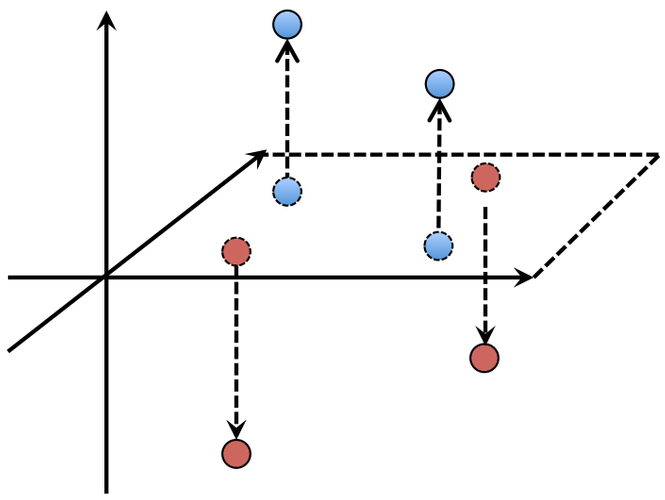
But could be linearly separable in a larger space

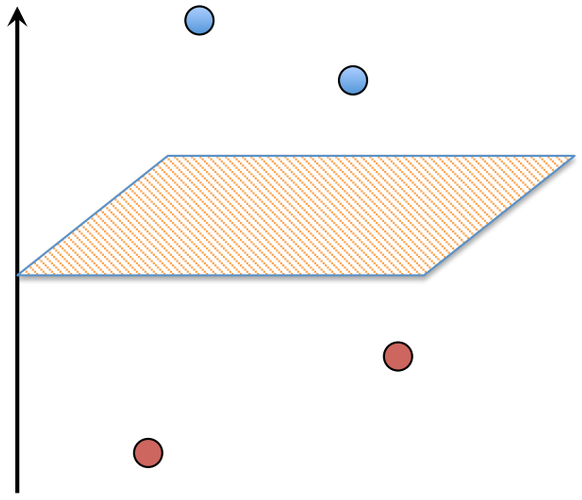
$$y^{(i)} = \Theta\left(\mathbf{W}\sigma(\mathbf{F}\mathbf{x}^{(i)}) + b_i\right)$$



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

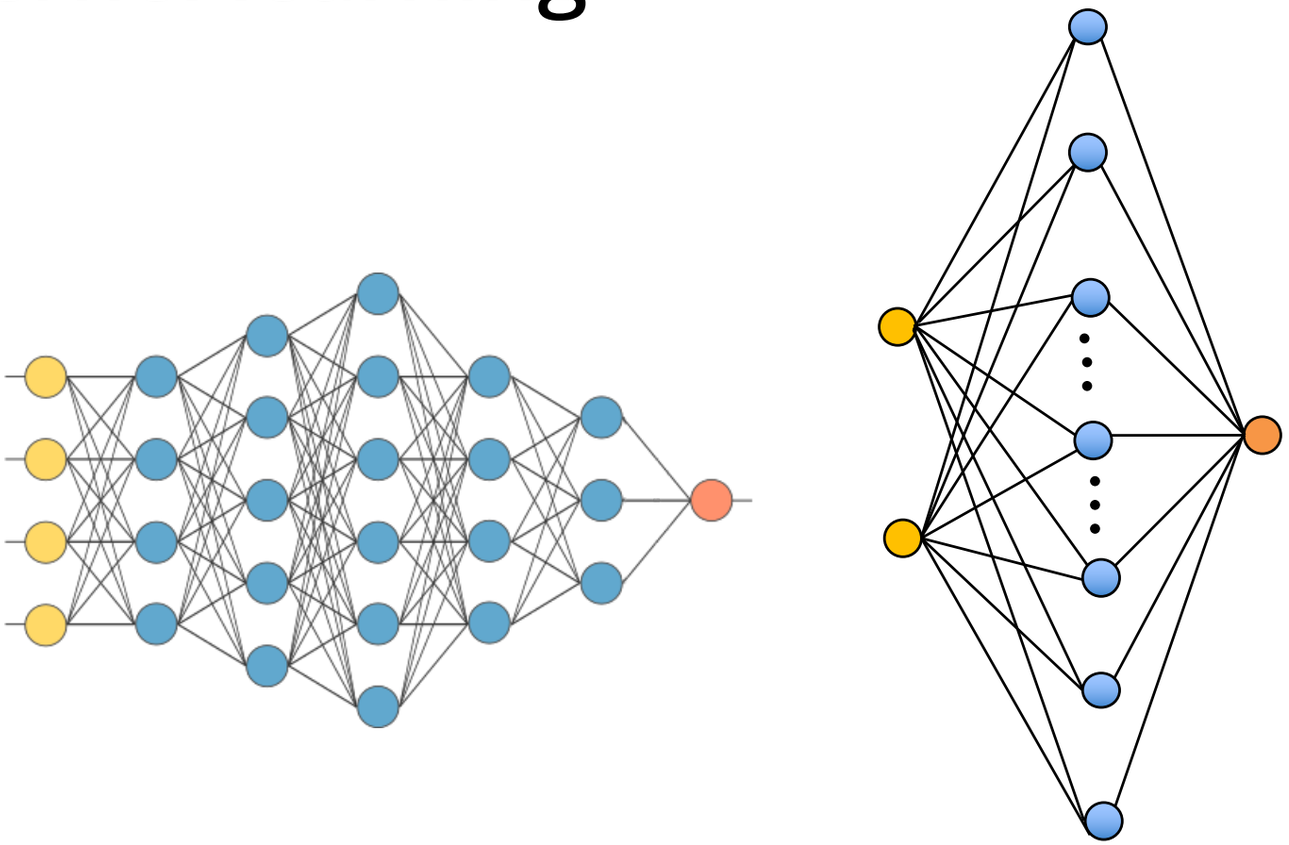
Feature Space





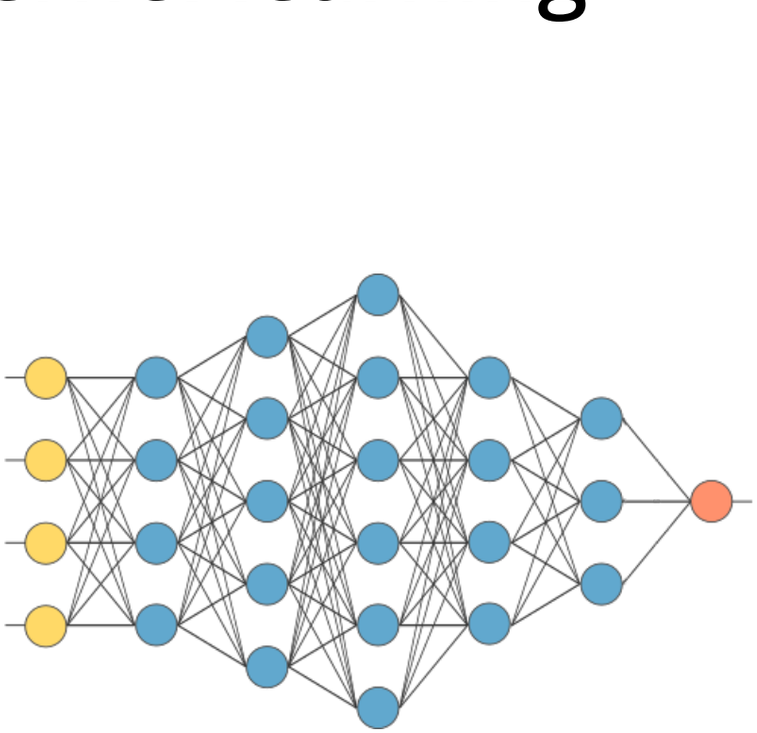
Kernel learning

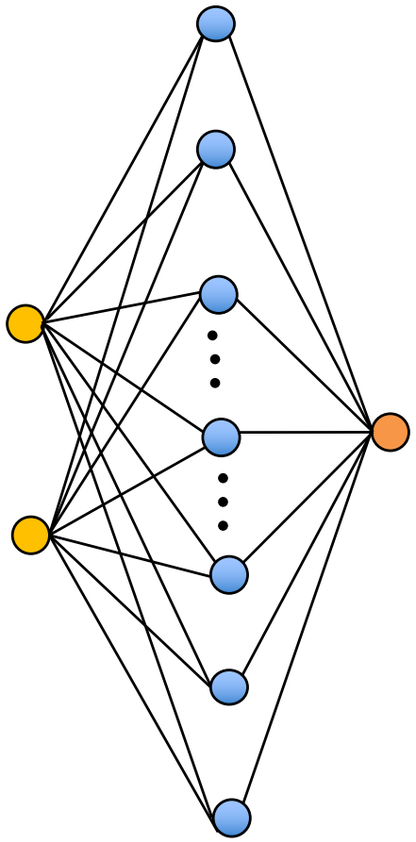
Kernel learning



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

Kernel learning





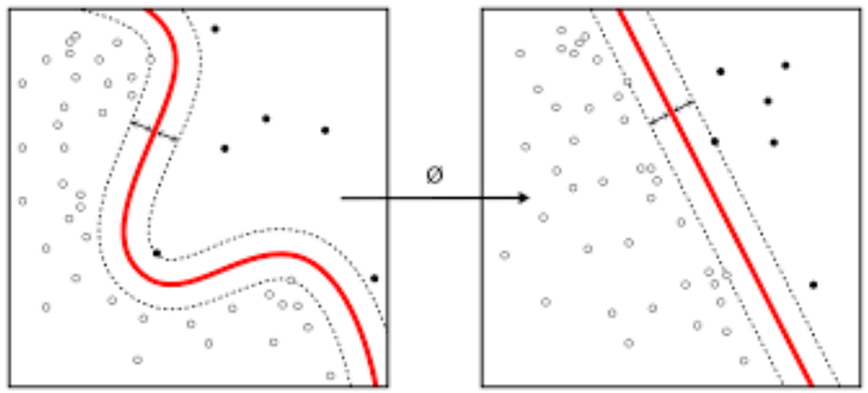
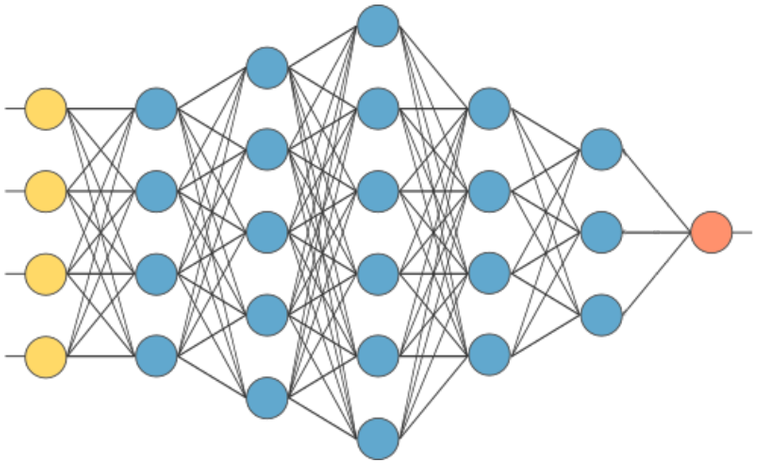


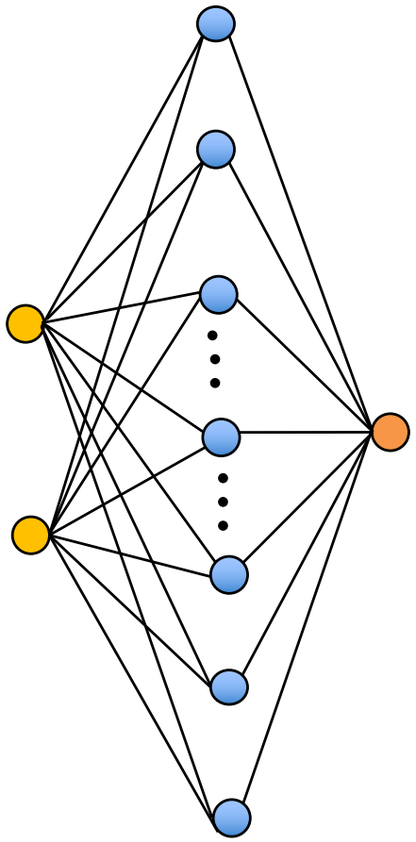
Image courtesy: Wikipedia

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

Kernel support vector machine is a classic example.

Kernel learning





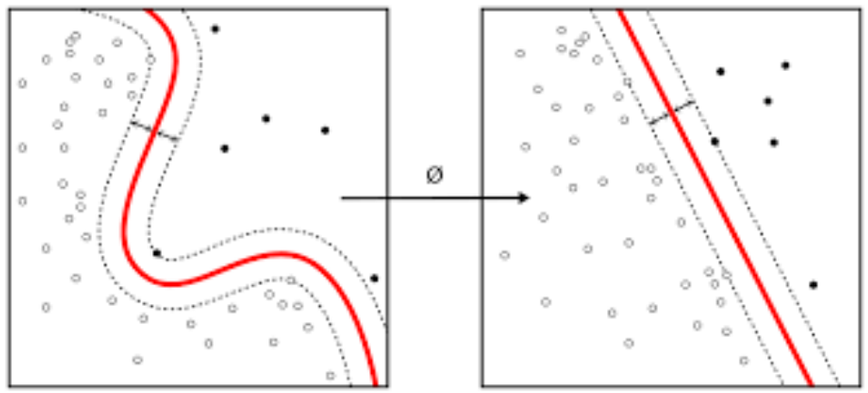


Image courtesy: Wikipedia

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*.

Kernel trick

Usually we do not need to explicitly construct the feature map $x \to \phi(x)$

Kernel trick

Usually we do not need to explicitly construct the feature map $x \to \phi(x)$

Instead, define a kernel function $k(x, x') = \phi(x)^T \phi(x')$

Kernel trick

Usually we do not need to explicitly construct the feature map $x \to \phi(x)$

Instead, define a kernel function $k(x, x') = \phi(x)^T \phi(x')$

Example: data in the two-dimensional input space, define

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$$

Kernel trick

Usually we do not need to explicitly construct the feature map $x \to \phi(x)$

Instead, define a kernel function $k(x, x') = \phi(x)^T \phi(x')$

Example: data in the two-dimensional input space, define

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 \\
&= (x_1 z_2 + x_2 z_2)^2 \\
&= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^T \\
&= \phi(\mathbf{x})^T \phi(\mathbf{z})
\end{aligned}
$$

Kernel trick

Usually we do not need to explicitly construct the feature map $x \to \phi(x)$

Instead, define a kernel function $k(x, x') = \phi(x)^T \phi(x')$

Example: data in the two-dimensional input space, define

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 \\
&= (x_1 z_2 + x_2 z_2)^2 \\
&= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^T \\
&= \phi(\mathbf{x})^T \phi(\mathbf{z})
\end{aligned}
$$

$$
\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longrightarrow \begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix}
$$

Kernel trick

Usually we do not need to explicitly construct the feature map $x \to \phi(x)$

Instead, define a kernel function $k(x, x') = \phi(x)^T \phi(x')$

Example: data in the two-dimensional input space, define

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 \\
&= (x_1 z_2 + x_2 z_2)^2 \\
&= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^T \\
&= \phi(\mathbf{x})^T \phi(\mathbf{z})
\end{aligned}
$$

$$
\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longrightarrow \begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix}
$$

Another example, Gaussian kernels

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\|\mathbf{x} - \mathbf{z}\|^2 / (2\sigma^2)\right)$$

Kernel trick

Usually we do not need to explicitly construct the feature map $x \to \phi(x)$

Instead, define a kernel function $k(x, x') = \phi(x)^T \phi(x')$

Example: data in the two-dimensional input space, define

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 \\
&= (x_1 z_2 + x_2 z_2)^2 \\
&= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^T \\
&= \phi(\mathbf{x})^T \phi(\mathbf{z})
\end{aligned}
$$

$$
\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longrightarrow \begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix}
$$

Another example, Gaussian kernels

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\|\mathbf{x} - \mathbf{z}\|^2 / (2\sigma^2)\right)$$

Question: what is the dimension of the corresponding feature vector ?

Explicit feature map to a large feature space

Explicit feature map to a large feature space

in many cases the kernel function is hard to construct.

Explicit feature map to a large feature space

in many cases the kernel function is hard to construct.

and we really want more properties in the feature space rather than only the inner products!

Explicit feature map to a large feature space

in many cases the kernel function is hard to construct.

and we really want more properties in the feature space rather than only the inner products!

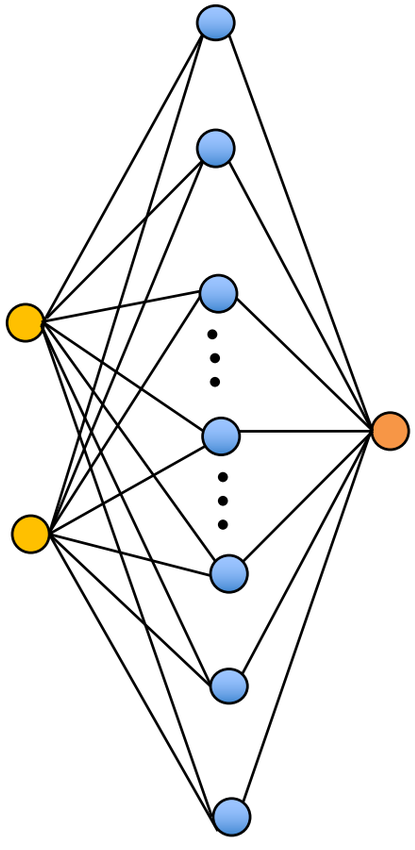
Is there any way to construct the feature space explicitly ?

Explicit feature map to a large feature space

in many cases the kernel function is hard to construct.

and we really want more properties in the feature space rather than only the inner products!

Is there any way to construct the feature space explicitly ?

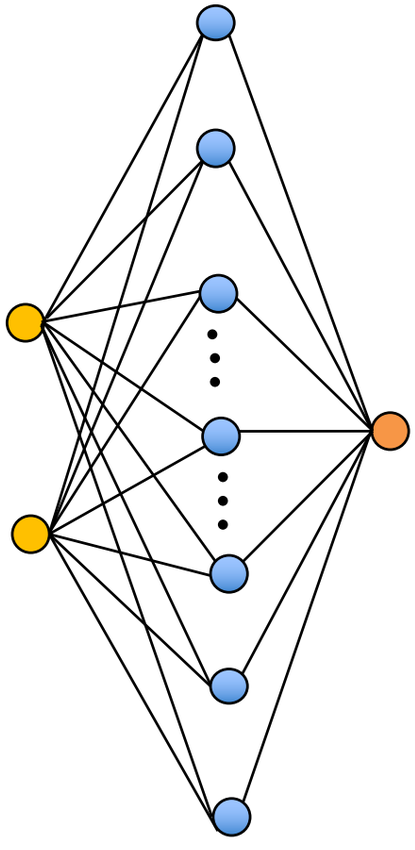


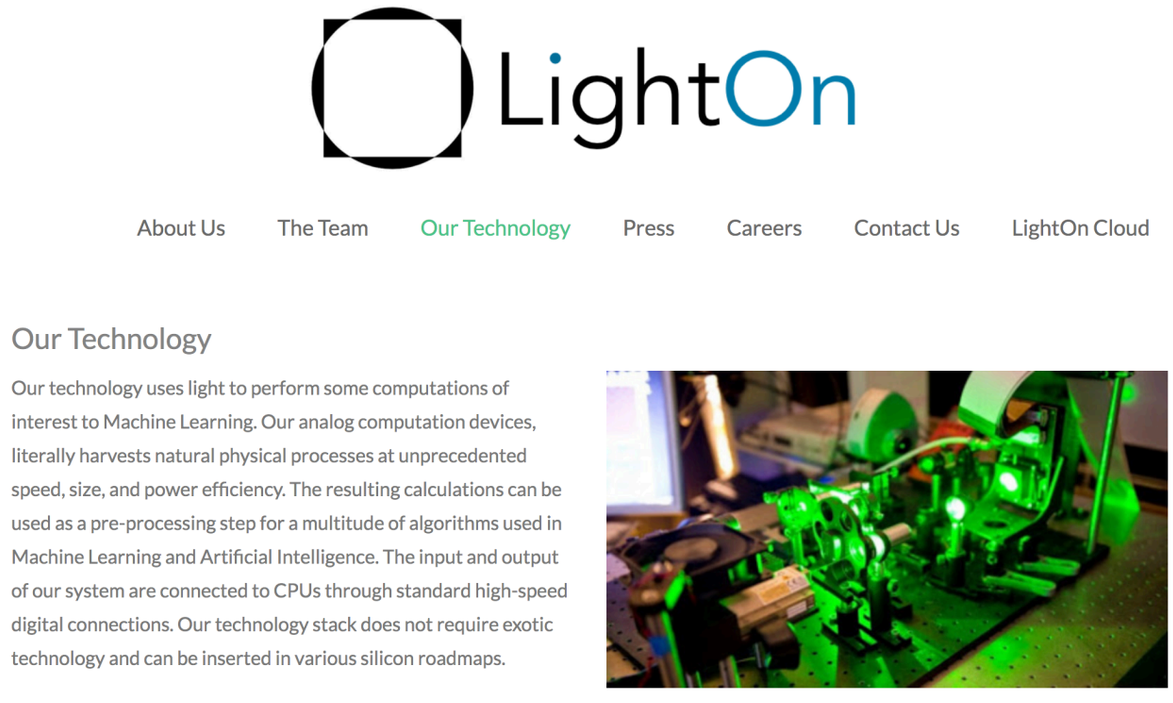
Explicit feature map to a large feature space

in many cases the kernel function is hard to construct.

and we really want more properties in the feature space rather than only the inner products!

Is there any way to construct the feature space explicitly ?



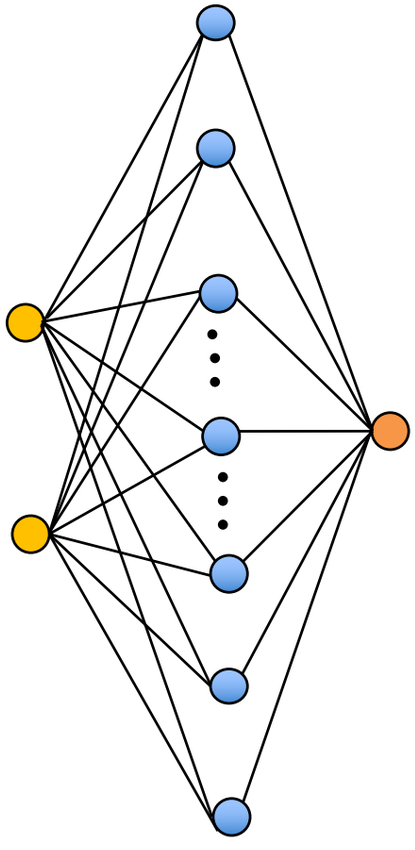


Explicit feature map to a large feature space

in many cases the kernel function is hard to construct.

and we really want more properties in the feature space rather than only the inner products!

Is there any way to construct the feature space explicitly ?

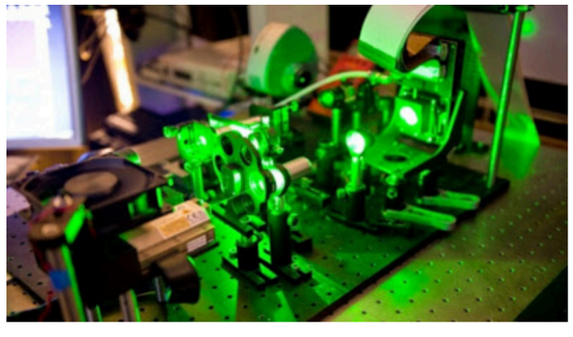


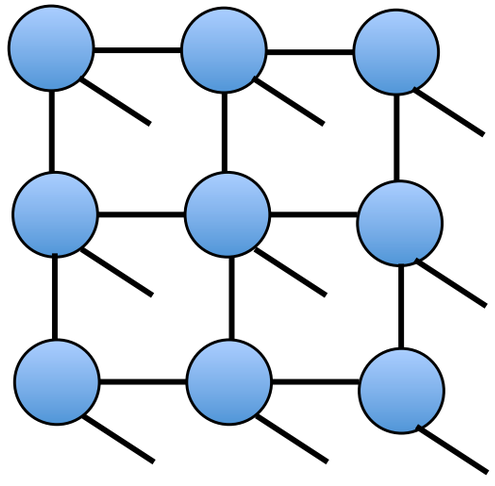
LightOn

About Us The Team Our Technology Press Careers Contact Us LightOn Cloud

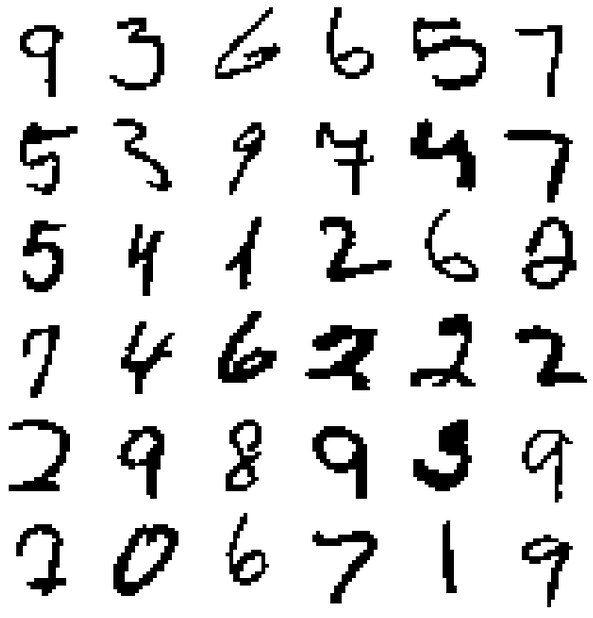
Our Technology

Our technology uses light to perform some computations of interest to Machine Learning. Our analog computation devices, literally harvests natural physical processes at unprecedented speed, size, and power efficiency. The resulting calculations can be used as a pre-processing step for a multitude of algorithms used in Machine Learning and Artificial Intelligence. The input and output of our system are connected to CPUs through standard high-speed digital connections. Our technology stack does not require exotic technology and can be inserted in various silicon roadmaps.





Feature mapping to Hilbert space

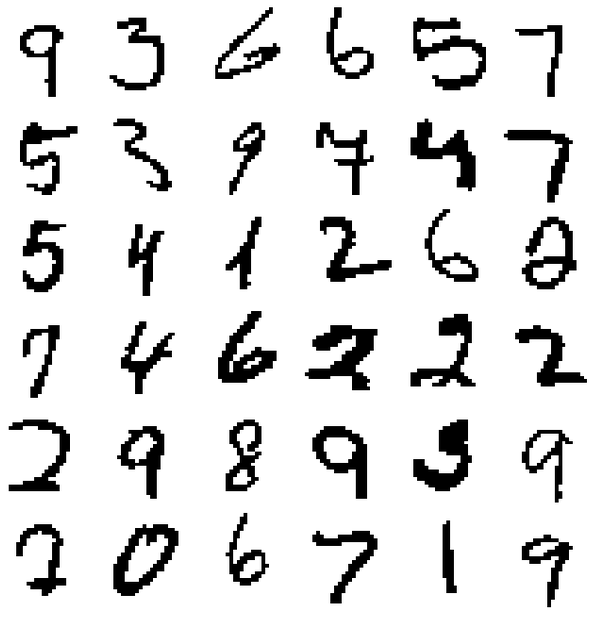


MNIST handwritten digits with labels 0,1,...,9

60,000 training images

10,000 test images

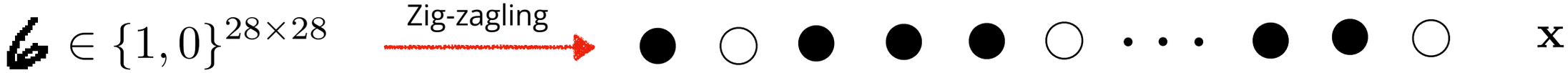
Feature mapping to Hilbert space



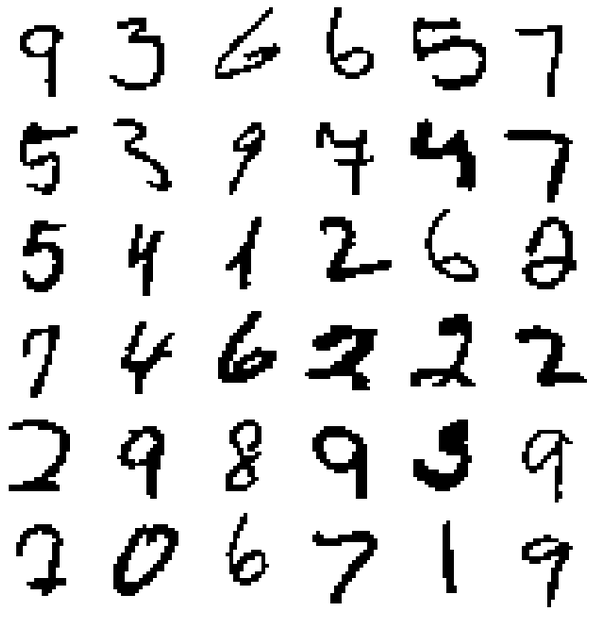
MNIST handwritten digits with labels 0,1,…,9

60,000 training images

10,000 test images



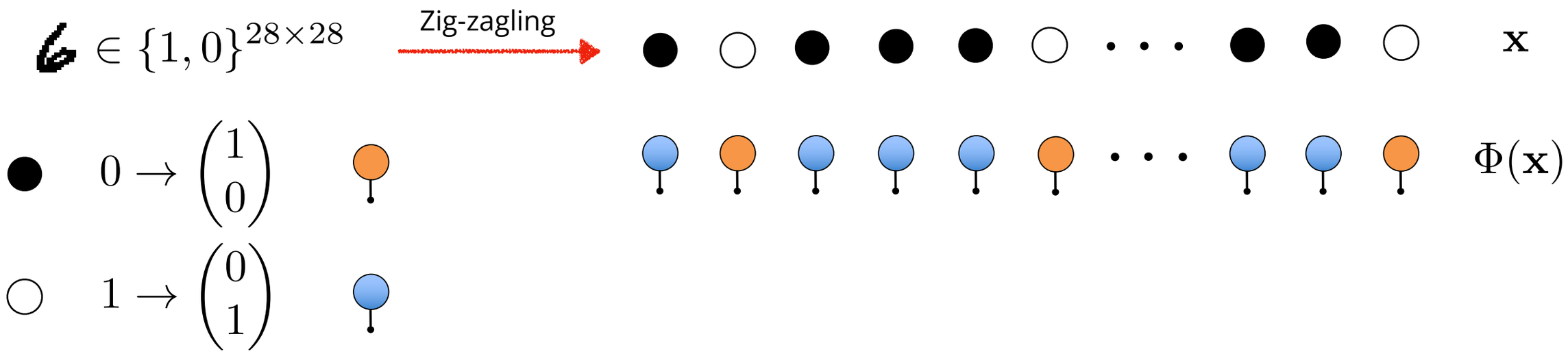
Feature mapping to Hilbert space



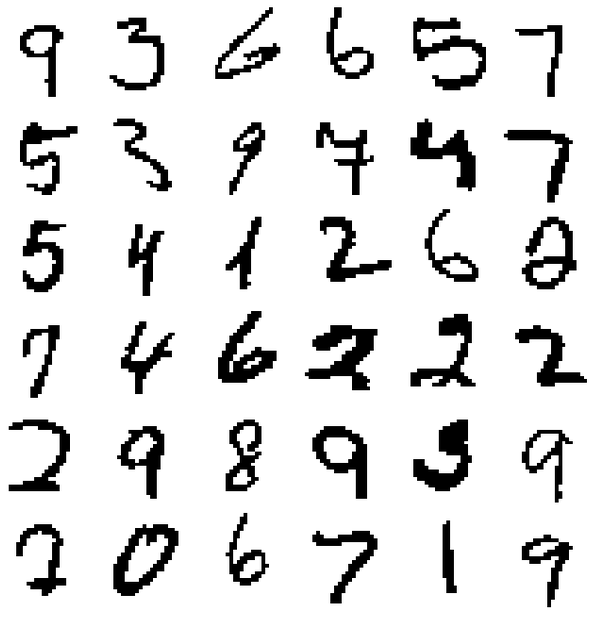
MNIST handwritten digits with labels 0,1,…,9

60,000 training images

10,000 test images



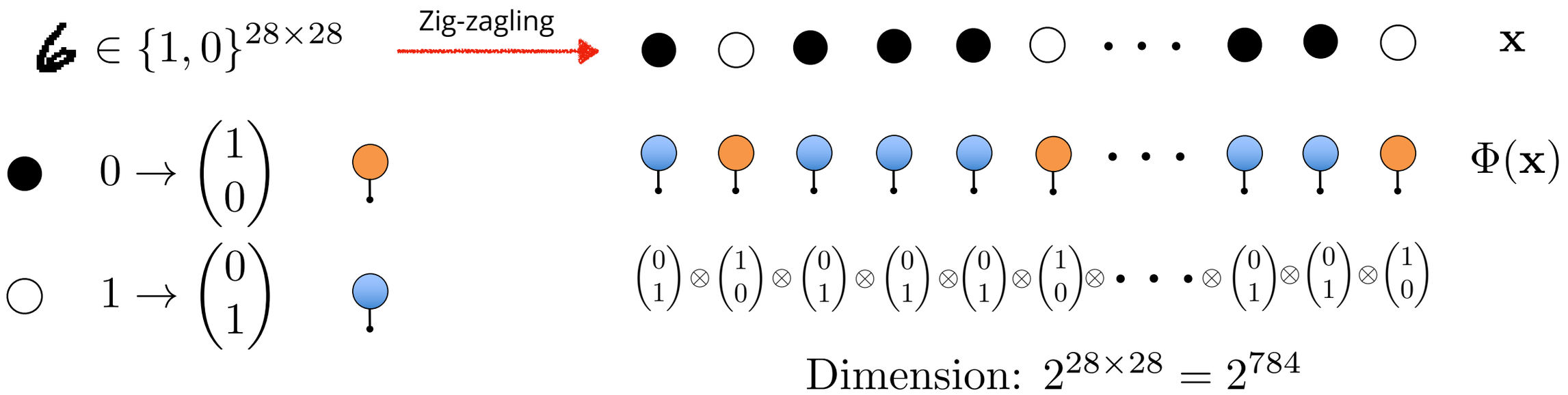
Feature mapping to Hilbert space



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

60,000 training images

10,000 test images

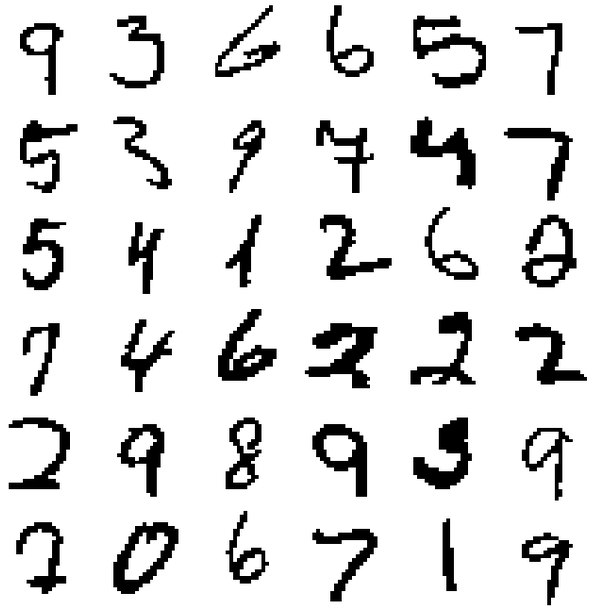


Rebentrost, Mohseni, Lloyd, Phys. Rev. Lett. 113, 130503 (2014)

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

Stoudenmire, Schwab, NIPS (2016)

MPS as a classifier in the Hilbert feature space

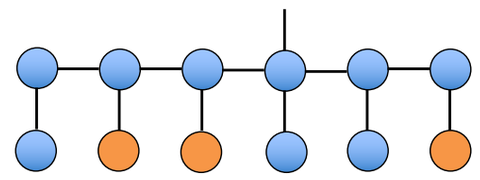


MNIST handwritten digits with labels 0,1,…,9

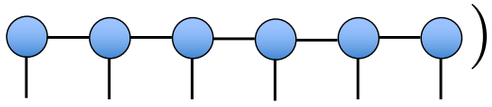
Tensor-product feature map:

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

$$\mathbf{F}(\ \ ) = \mathbf{F}(\mathbf{x_i}) = \mathbf{F}\ (\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ ) \ =$$



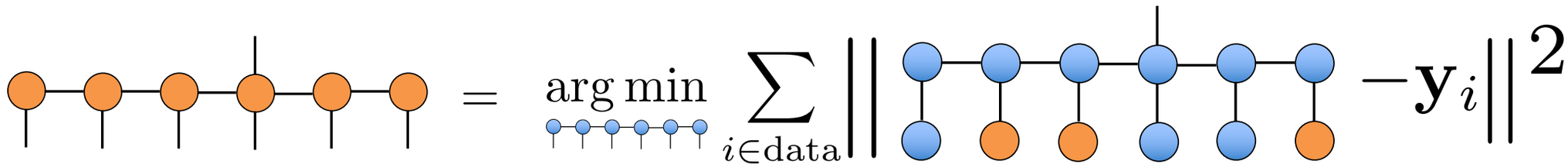
$$\mathrm{Loss}($$



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

Stoudenmire, Schwab, NIPS 2016

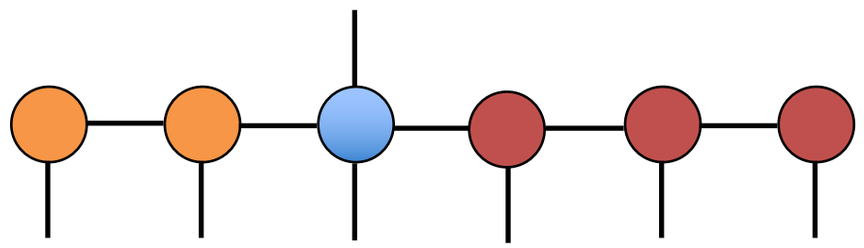
Learning algorithm analogous to DMRG



Stoudenmire, Schwab, NIPS 2016

Learning algorithm analogous to DMRG

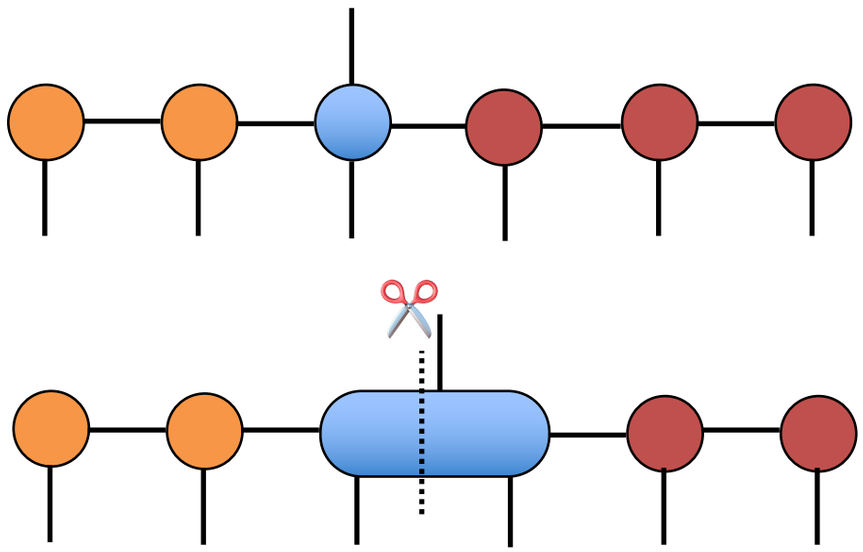
Mixed canonical form



Stoudenmire, Schwab, NIPS 2016

Learning algorithm analogous to DMRG

Mixed canonical form

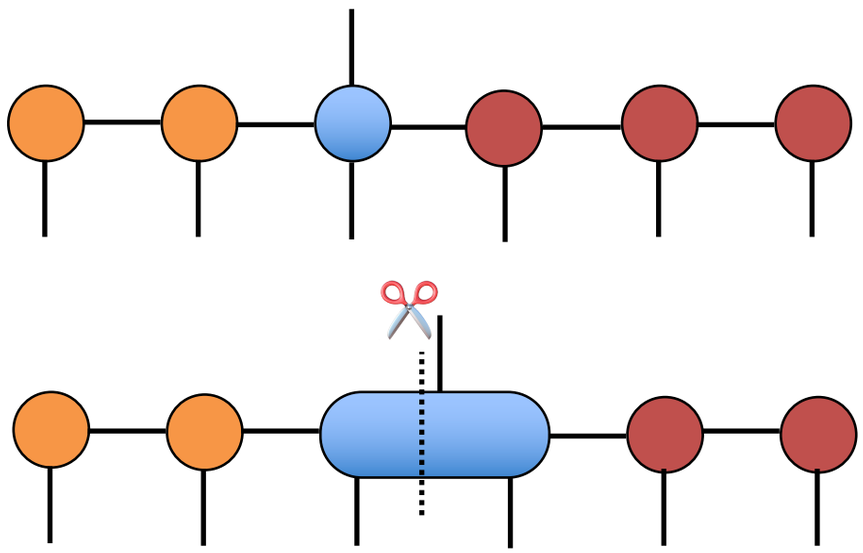


Merging + gradient descent

Stoudenmire, Schwab, NIPS 2016

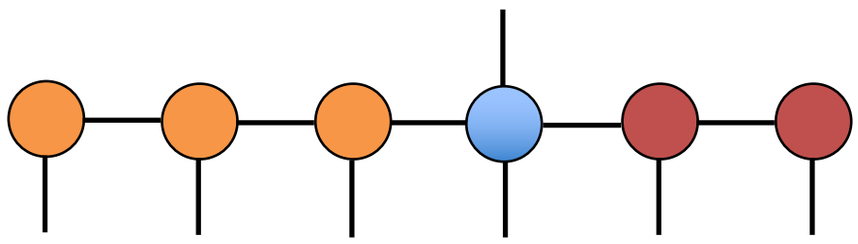
Learning algorithm analogous to DMRG

Mixed canonical form



Merging + gradient descent

Applying SVD

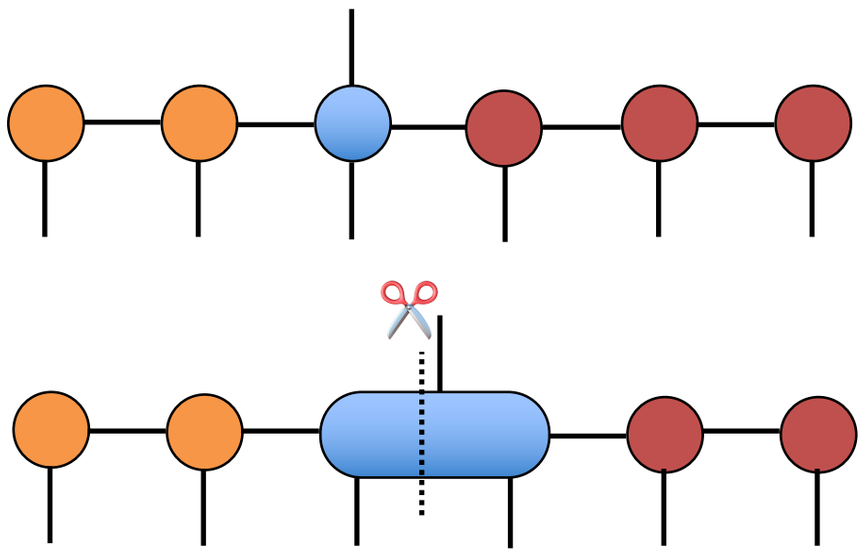


Stoudenmire, Schwab, NIPS 2016

Learning algorithm analogous to DMRG

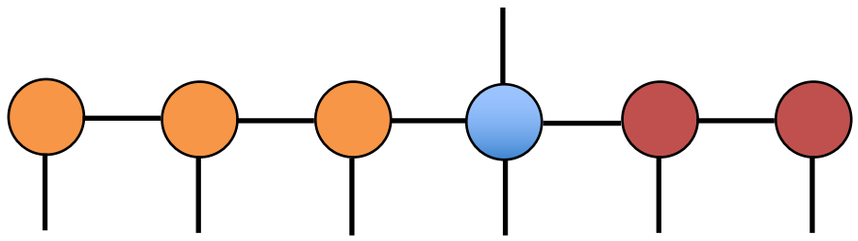
Strengths:

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



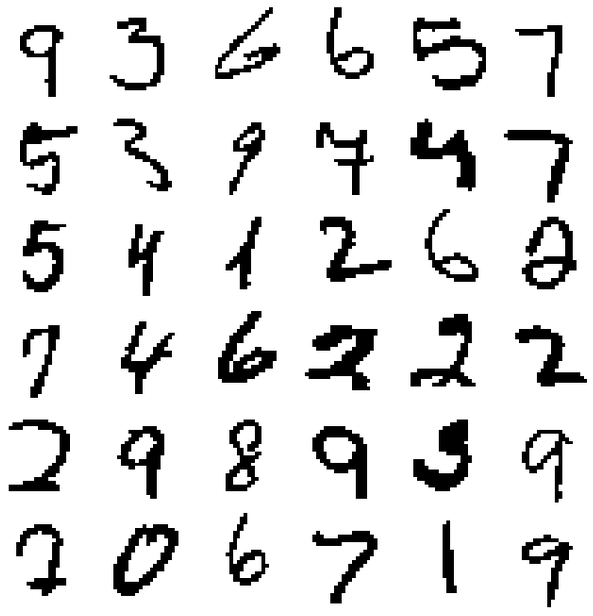
Limitations:

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



Stoudenmire, Schwab, NIPS 2016

Results on MNIST

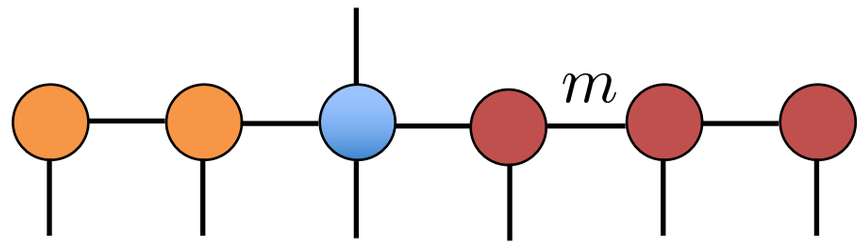


MNIST handwritten digits with labels 0,1,…,9

Tensor-product feature map:

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

Croase grained to 14x14



Only takes several sweeps to converge !

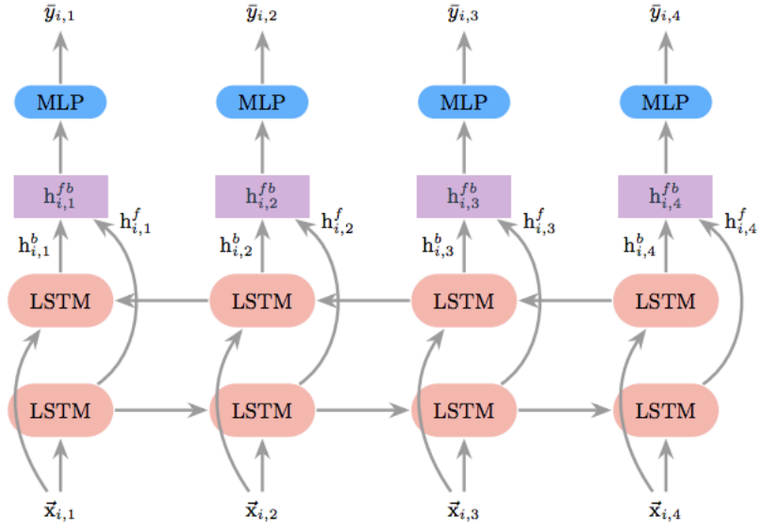
Bond dimension	Test Set Error
m = 10	~5% (500/10,000 incorrect)
m = 20	~2% (200/10,000 incorrect)
m = 120	0.97% (97/10,000 incorrect)

Stoudenmire, Schwab, NIPS 2016

Sequence-to-sequence learning using MPO

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

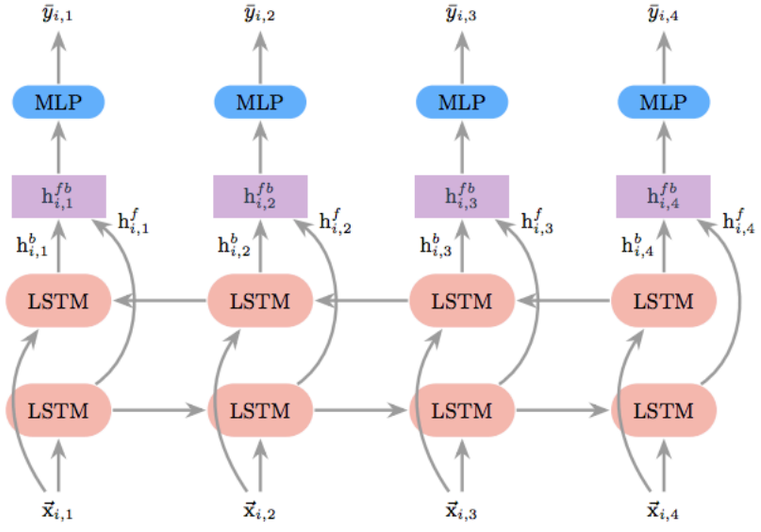
State-of-the-arts methods are based on RNN, especially the LSTM.

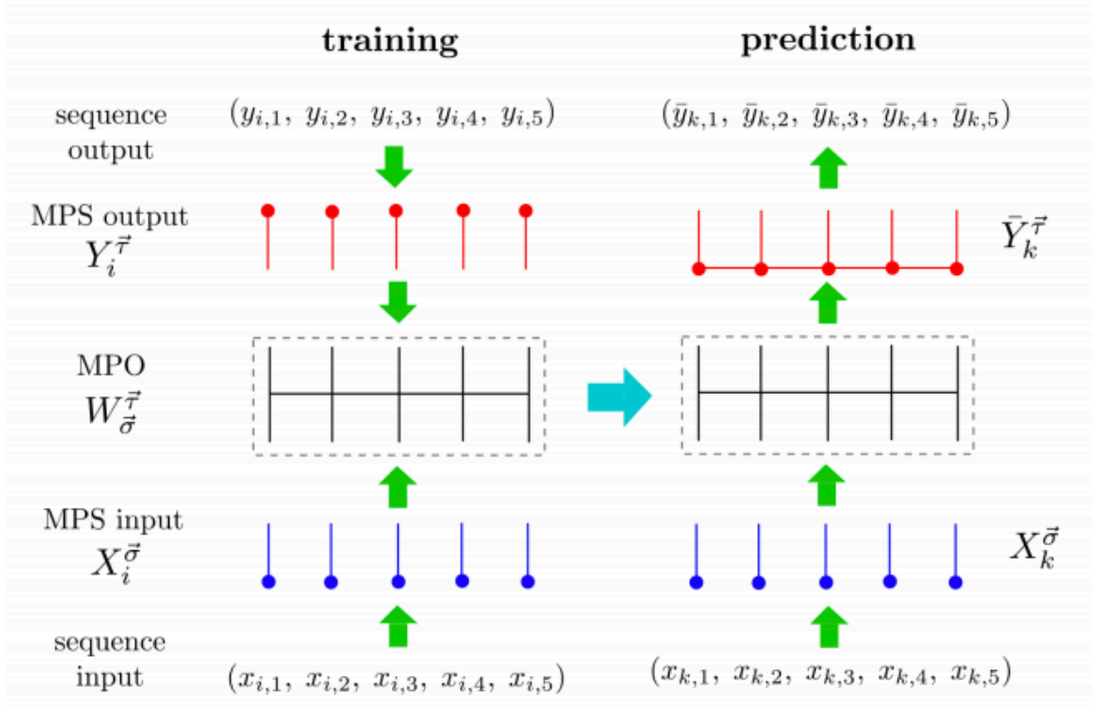


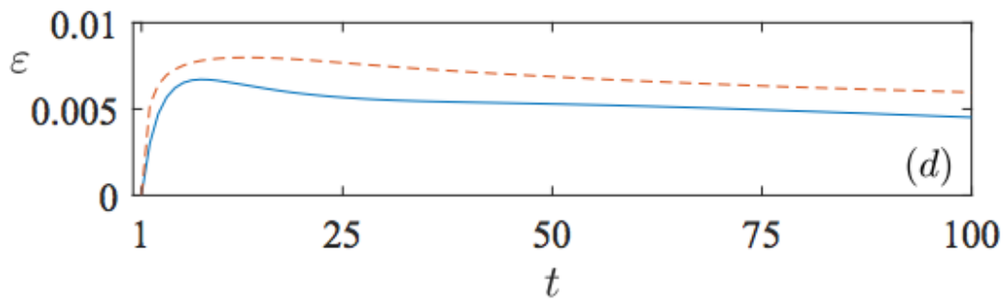
Sequence-to-sequence learning using MPO

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.



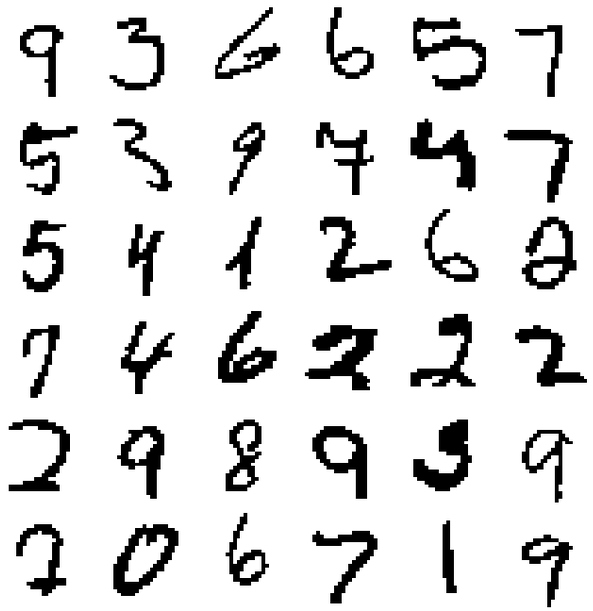


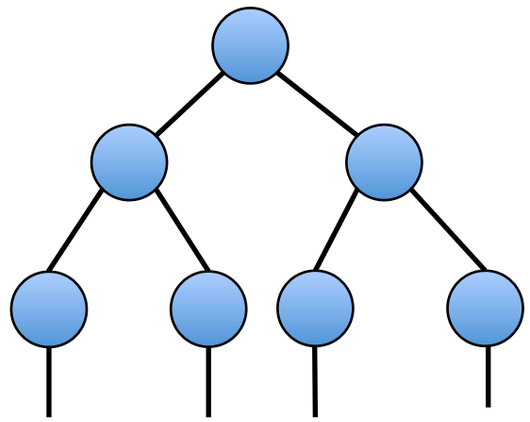


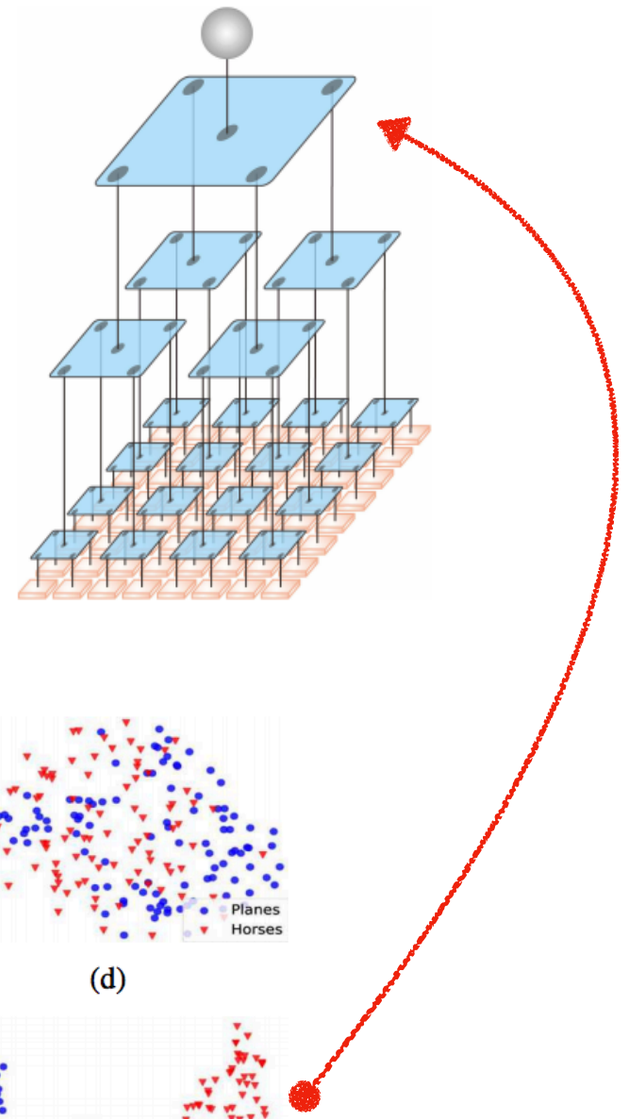
In some tasks MPO outperforms LSTM!

Guo, Jie, Lu, Poletti, arxiv:1803.10908

Tree tensor network to MNIST classifications



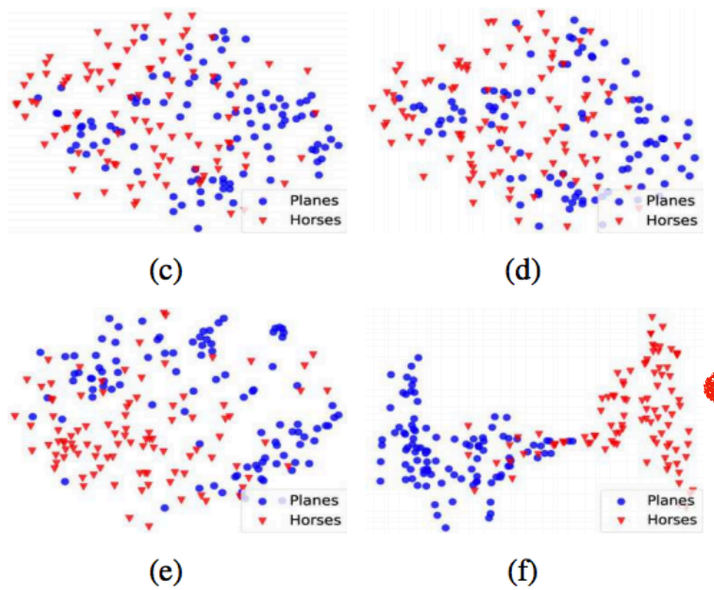




Using 2-D structures

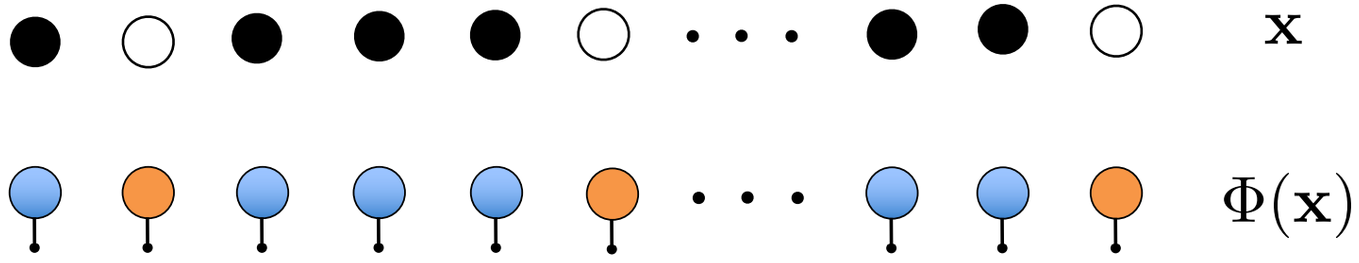
Gradient-free: alternating least square

Dimensionality reduction at top layers

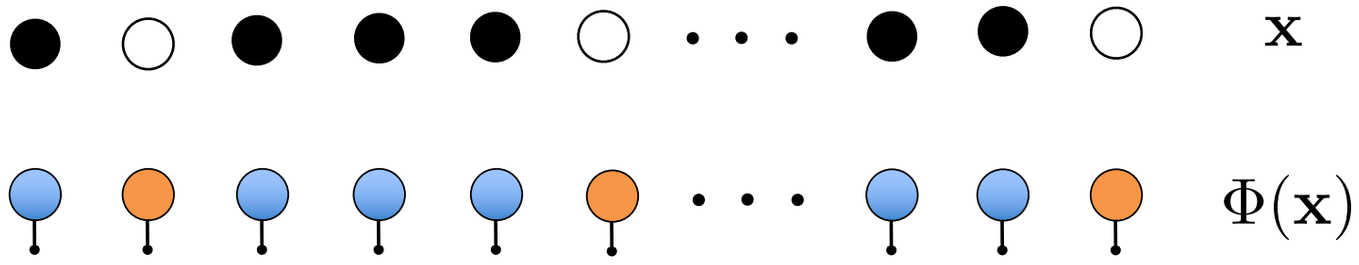


Liu, Ran, Wittek, Peng, Garcia, Su, Lewenstein, arxiv:1710.04833

Dimensionality reduction using kernel PCA



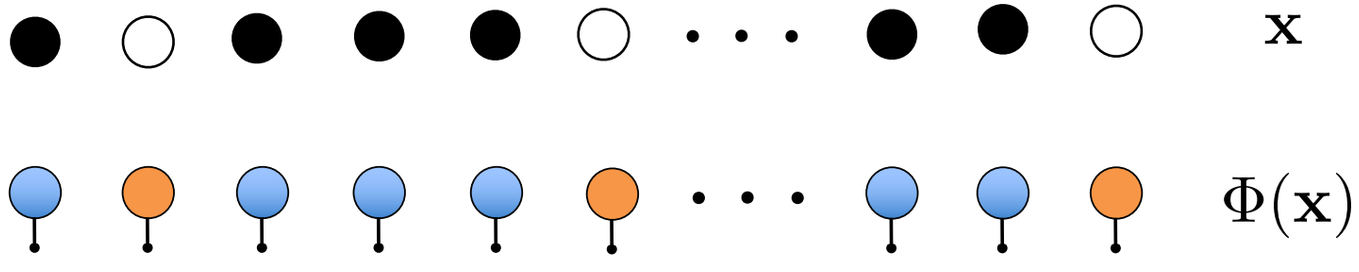
Dimensionality reduction using kernel PCA



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

— *representer theorem*

Dimensionality reduction using kernel PCA

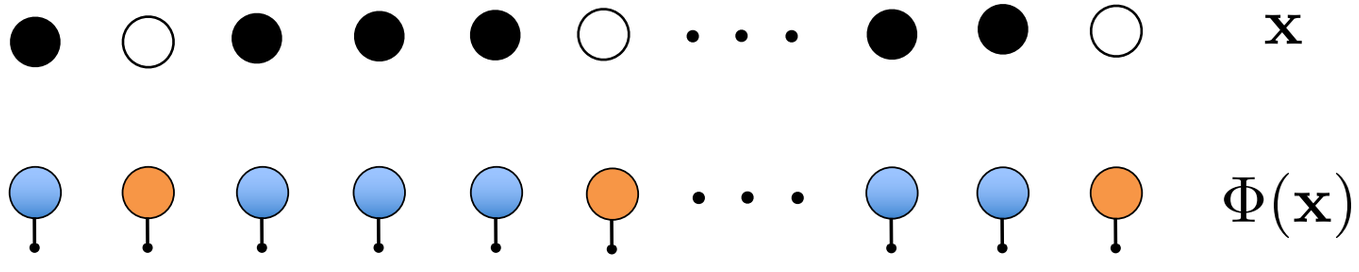


the optimal weights is expressed by span of the feature space.

— *representer theorem*

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

Dimensionality reduction using kernel PCA



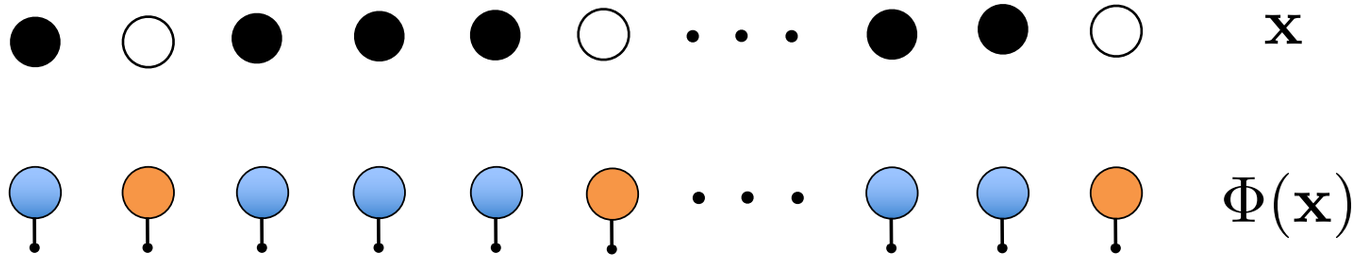
the optimal weights is expressed by span of the feature space.

— *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.

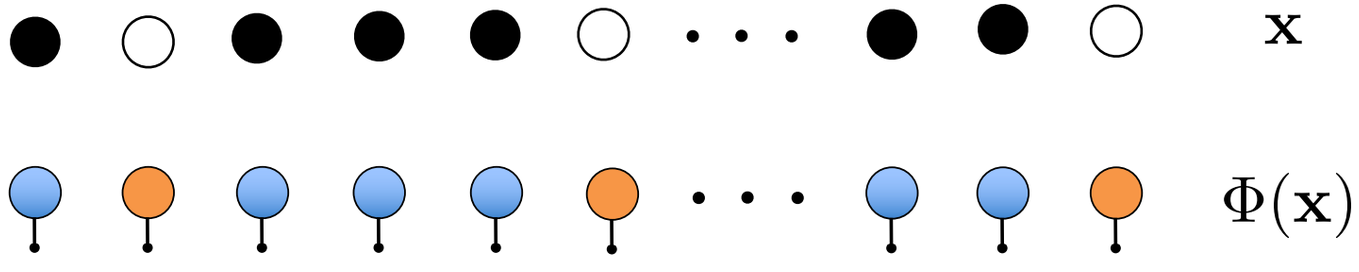
— *representer theorem*

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*

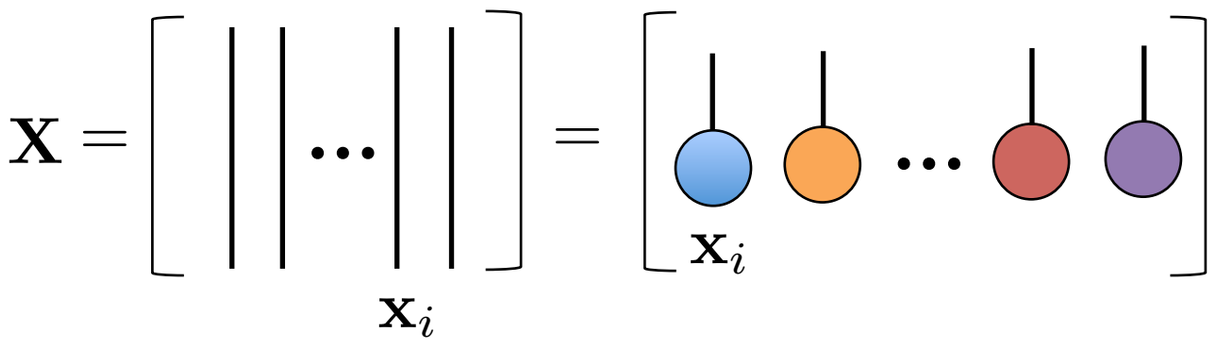
the feature space is very large, it could be of low-rank.

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.

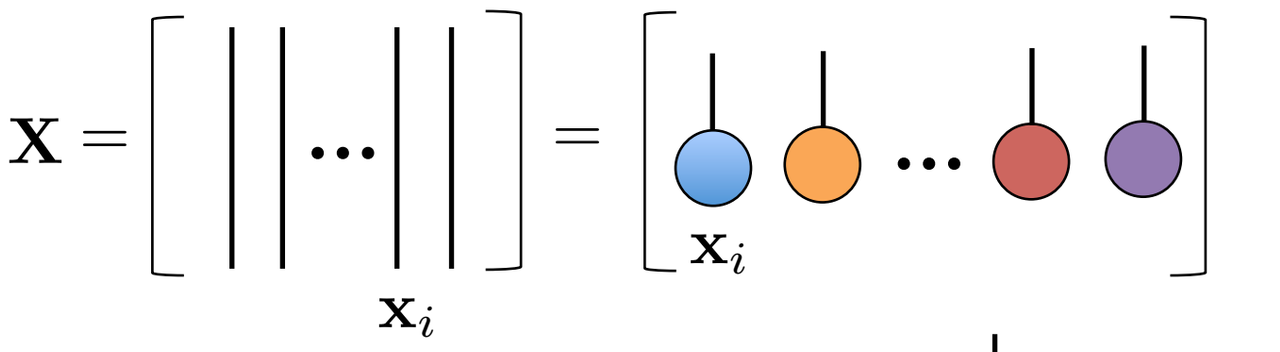
In the quantum word, the covariance matrix is analogous to the density matrix

Canonical PCA

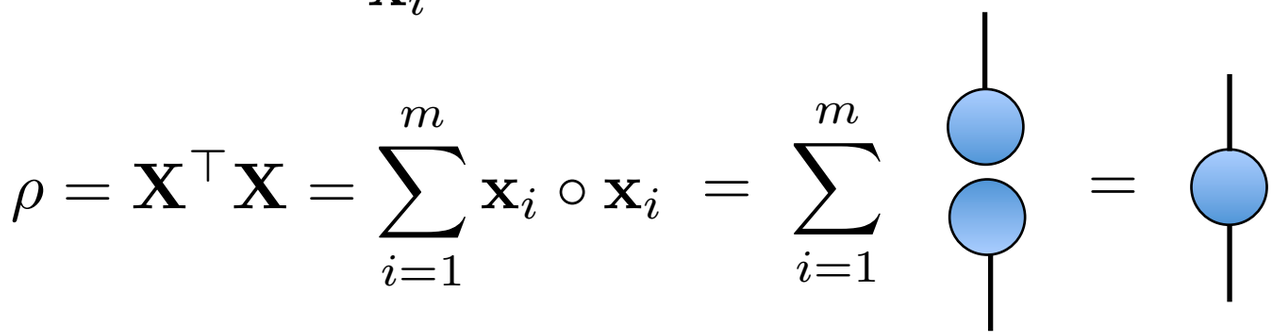


data matrix

Canonical PCA



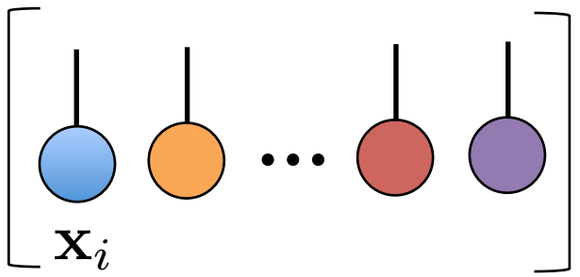
data matrix



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

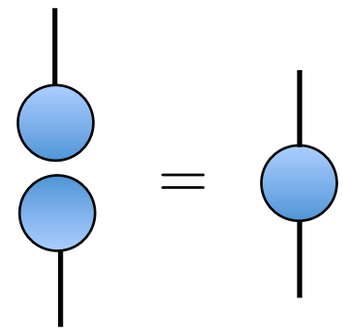
Canonical PCA

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



data matrix

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



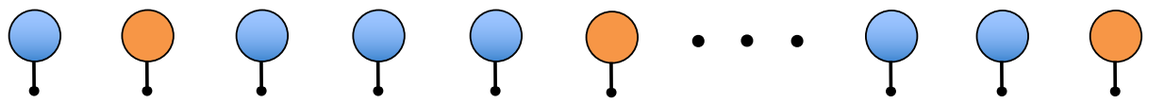
$$\sum_{i=1}^{m}$$

$$=$$

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

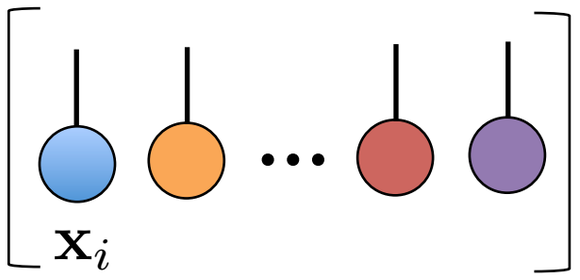
Hilbert feature space PCA

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



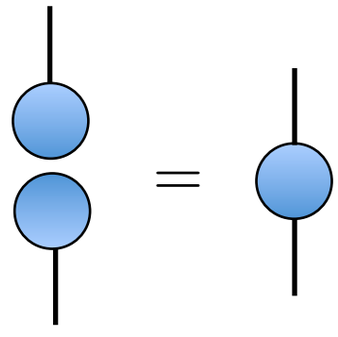
Canonical PCA

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



data matrix

$$\rho = \mathbf{X}^\top \mathbf{X} = \sum_{i=1}^{m} \mathbf{x}_i \circ \mathbf{x}_i \; = \sum_{i=1}^{m}$$

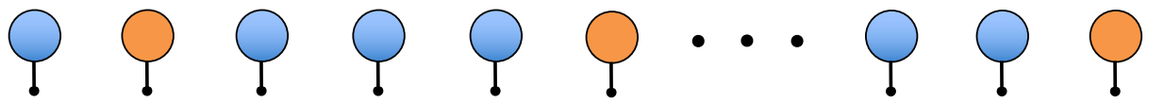


$=$

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

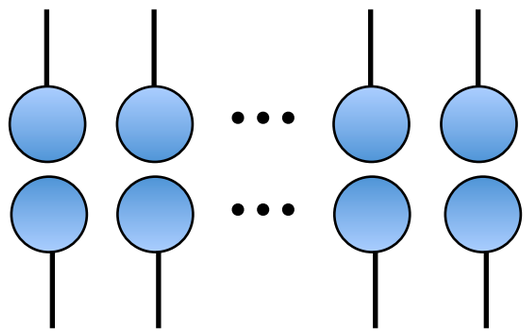
Hilbert feature space PCA

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

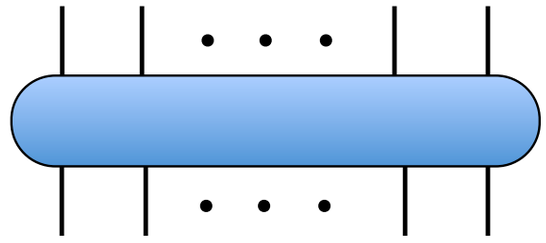


$$\rho = \Phi(\mathbf{X})^\top \Phi(\mathbf{X}) = \sum_{i=1}^{m} \Phi(\mathbf{x}_i) \circ \Phi(\mathbf{x}_i)$$

$$= \sum_{i=1}^{m}$$



$=$



Dimensionality reduction

Hilbert feature covariance matrix is analogous to the density matrix

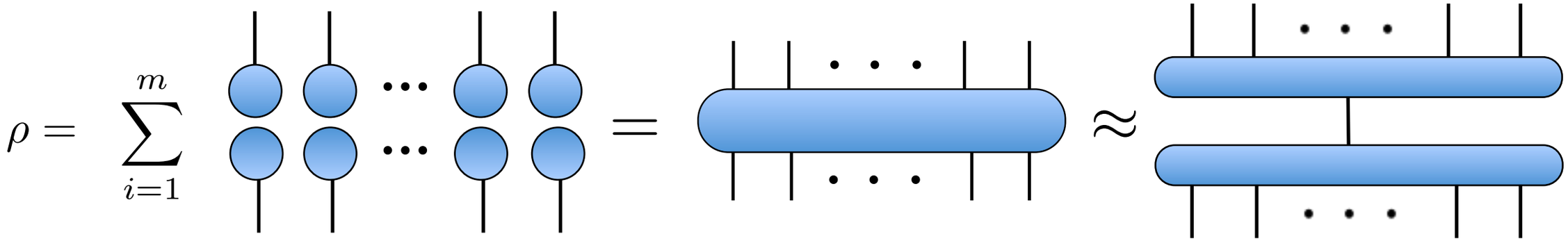
It is possibly of *low-rank*

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

Dimensionality reduction

Hilbert feature covariance matrix is analogous to the density matrix

It is possibly of *low-rank*

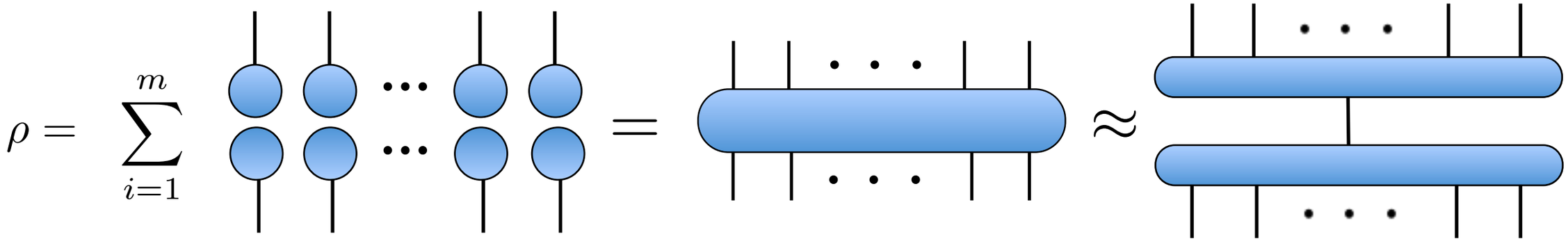


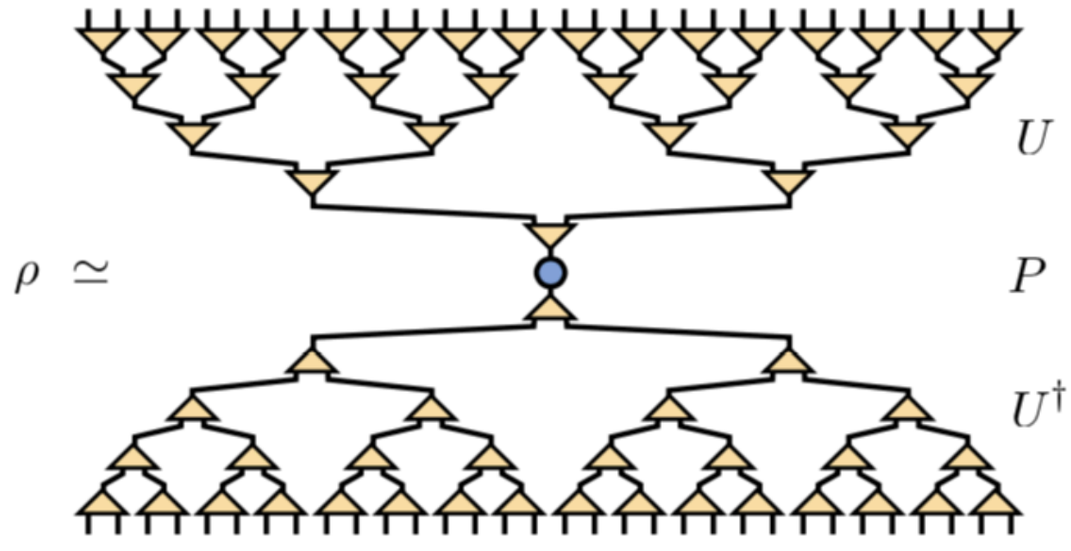
*Stoudenmire, Quant. Sci. Tech., **3**, 034003 (2018)*

Dimensionality reduction

Hilbert feature covariance matrix is analogous to the density matrix

It is possibly of *low-rank*





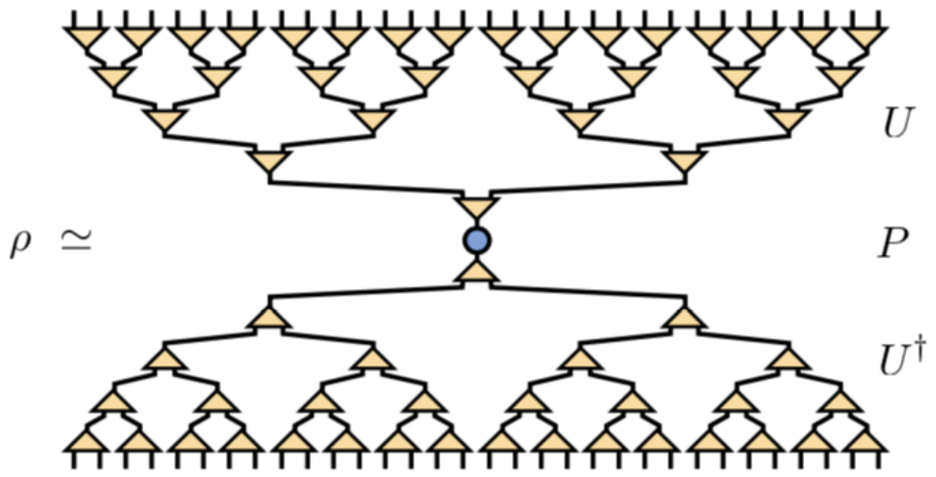
*Stoudenmire, Quant. Sci. Tech., **3**, 034003 (2018)*

Apply to the more challenge dataset fashion MNIST

28x28 grayscale images with 10 labels.

60,000 training images

10,000 test images



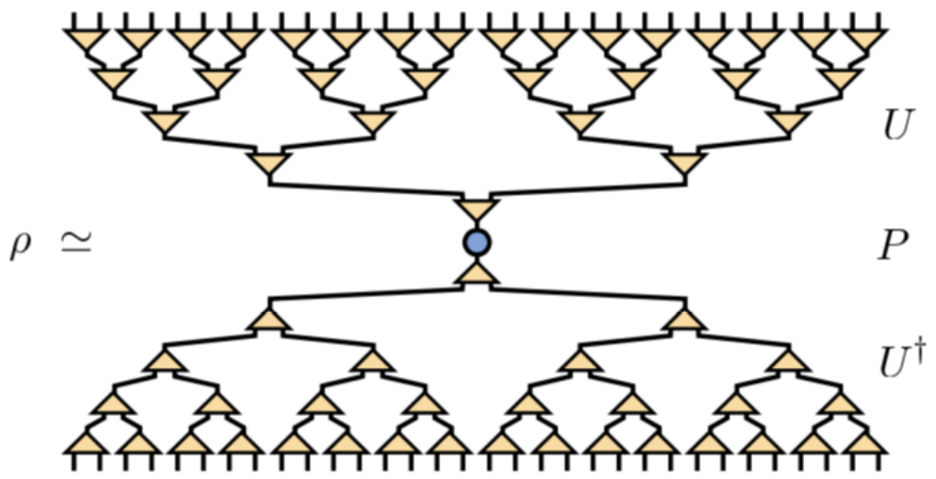
*Stoudenmire, Quant. Sci. Tech., **3**, 034003 (2018)*

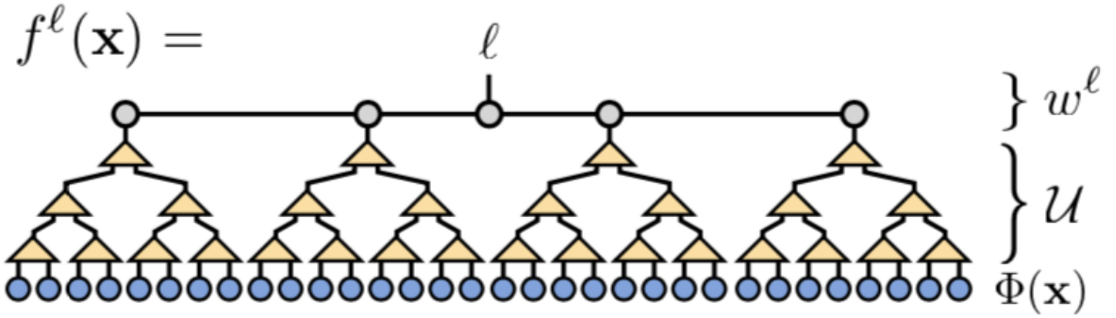
Apply to the more challenge dataset fashion MNIST

28x28 grayscale images with 10 labels.

60,000 training images

10,000 test images





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

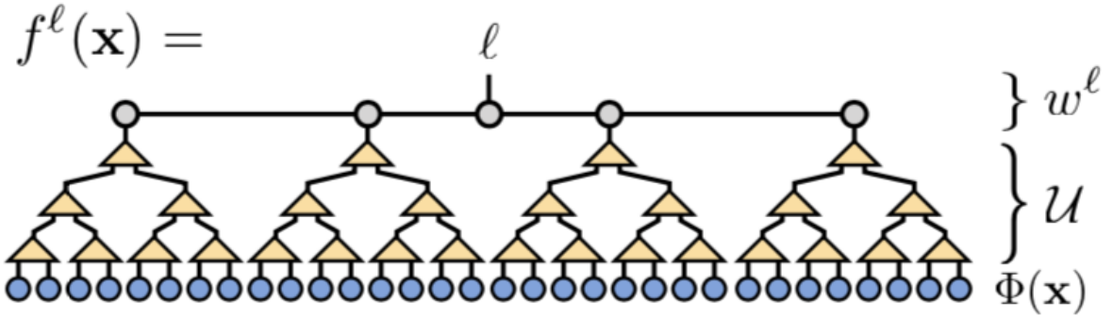
Apply to the more challenge dataset fashion MNIST

28x28 grayscale images with 10 labels.

60,000 training images

10,000 test images

Method	Test Accuracy
AlexNet	88.90%
Tree tensor net. [1]	88.97%
String bond state [2]	89.2%
CNN-String bond state [2]	92.3%
GoogLeNet	93.7%

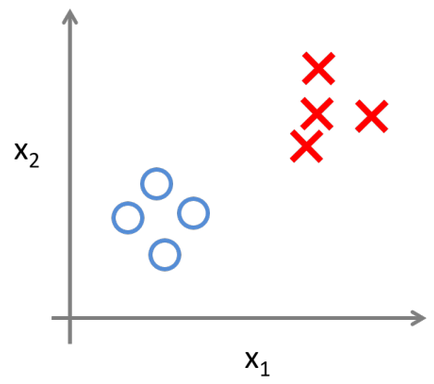


*1. Stoudenmire, Quant. Sci. Tech., **3**, 034003 (2018)*

2. Glasser, Pancotti, Cirac, arxiv:1806.05964

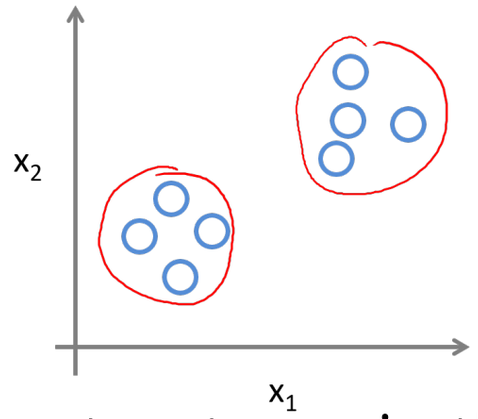
From supervised to unsupervised learning

Supervised Learning



Predicting Labels

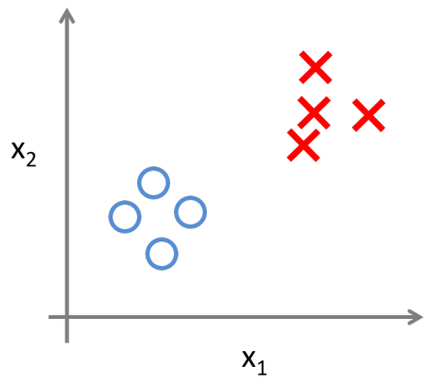
Unsupervised Learning



Finding structures in the data

From supervised to unsupervised learning

Supervised Learning



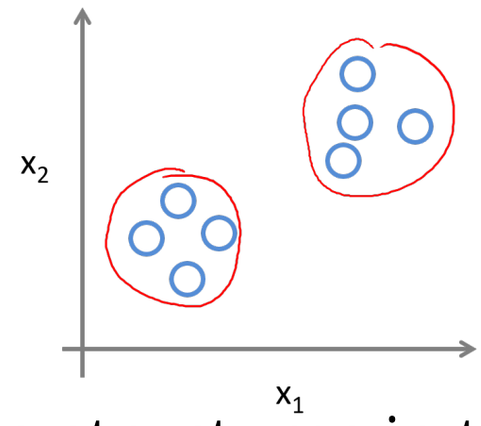
Predicting Labels

Discriminative

$$y = f(\boldsymbol{x})$$
$$\text{or } p(y \mid \boldsymbol{x})$$

Classification
Sequence prediction

Unsupervised Learning



Finding structures in the data

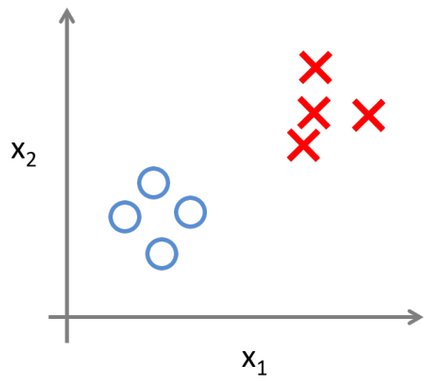
Generative

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

“fake image generation”

From supervised to unsupervised learning

Supervised Learning



Predicting Labels

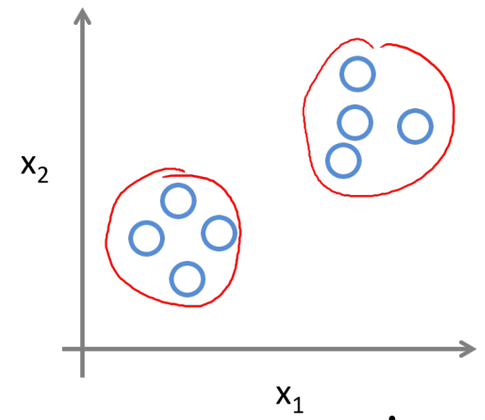
Discriminative

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

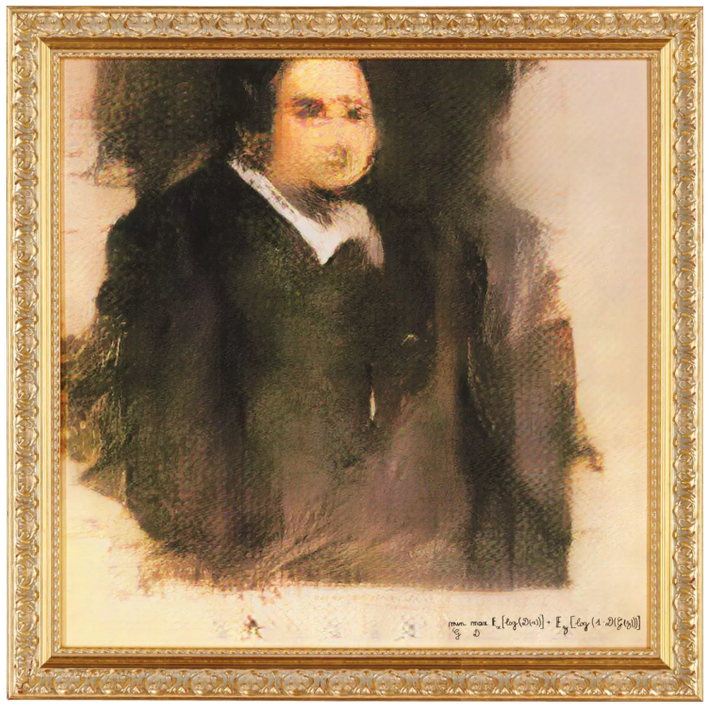
or $p(y \mid \boldsymbol{x})$

Classification
Sequence prediction

Unsupervised Learning



Finding structures in the data



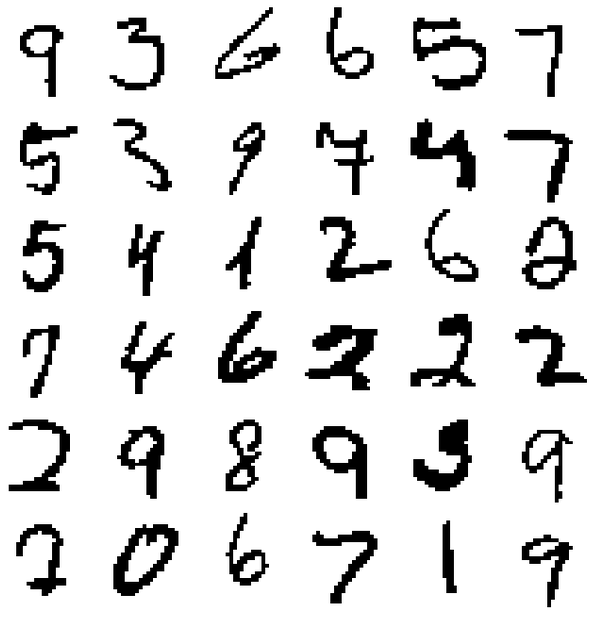
A machine learning generated print

sold for $432,500

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

“fake image generation”

Feature mapping to Hilbert space



The largest possible space for binary data

The right space for computing the “partition function”

$$\in \{1,0\}^{28\times 28}$$

Zig-zagling

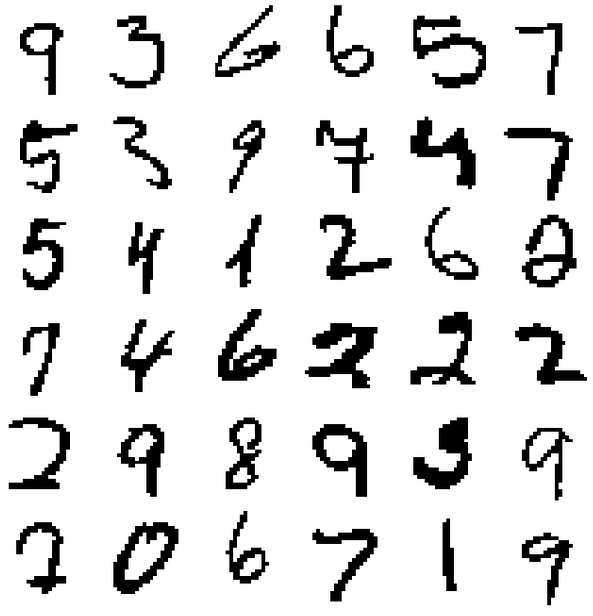
$$\bullet \quad 0 \to \begin{pmatrix}1\\0\end{pmatrix}$$

$$\circ \quad 1 \to \begin{pmatrix}0\\1\end{pmatrix}$$

$$\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix}\otimes\cdots\otimes\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix}$$

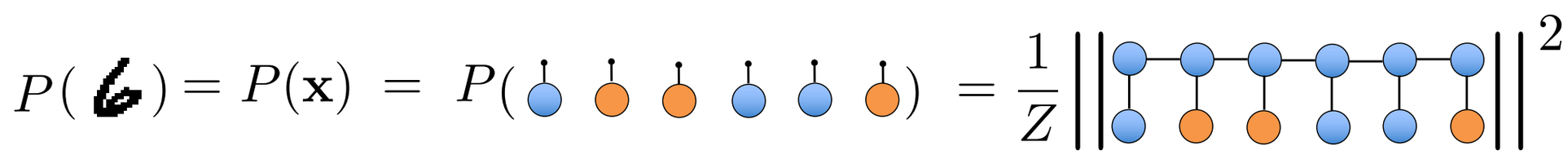
Dimension: $2^{28\times 28} = 2^{784}$

Parametrizing the joint probability distribution



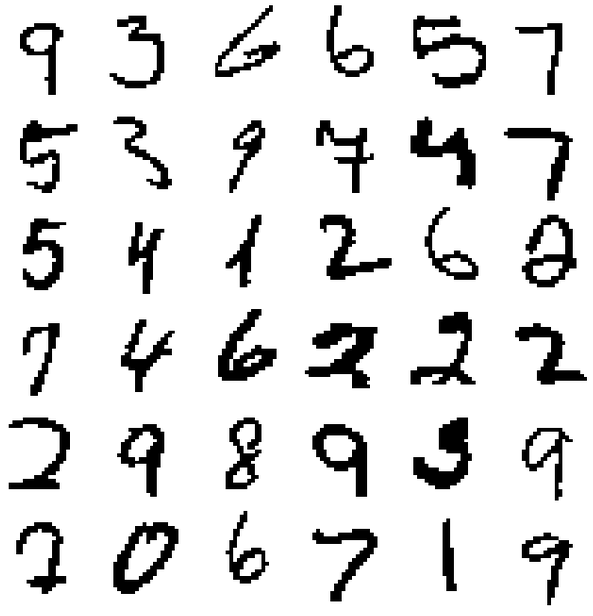
The largest possible space for binary data

The right space for computing the “partition function”



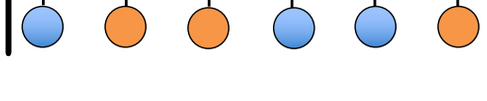
Han, Wang, Fan, Wang, Zhang, Phys. Rev. X 8, 031012 (2018)

Parametrizing the joint probability distribution



The largest possible space for binary data

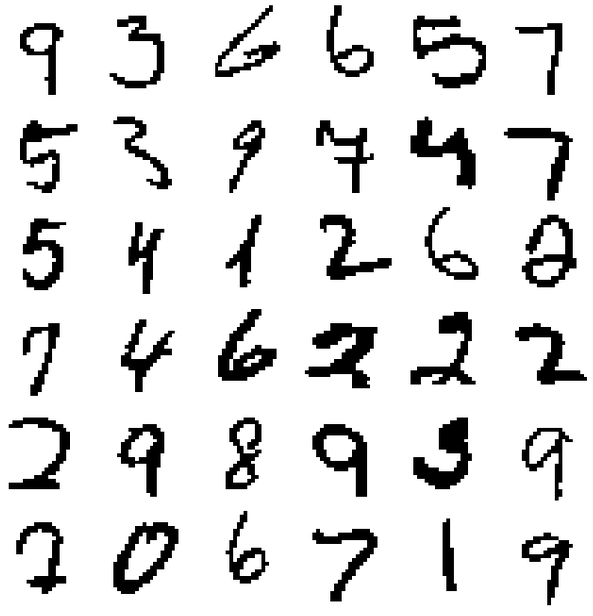
The right space for computing the “partition function”

$$P(\,\raisebox{-0.3em}{  }\,) = P(\mathbf{x}) = P(\,\raisebox{-0.3em}{  }\,) = \frac{1}{Z}\left\|\,\raisebox{-0.3em}{  }\,\right\|^2$$

Born’s rule

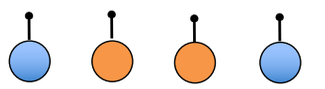
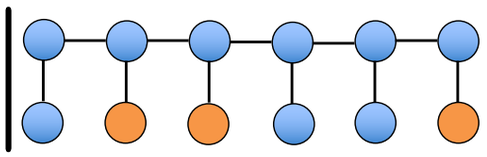
Han, Wang, Fan, Wang, Zhang, Phys. Rev. X 8, 031012 (2018)

Parametrizing the joint probability distribution



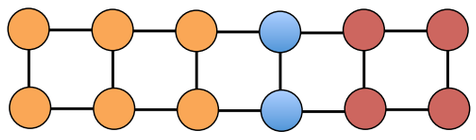
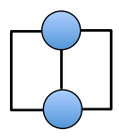
The largest possible space for binary data

The right space for computing the “partition function”

$$P(\,\raisebox{-0.2em}{  }\,) = P(\mathbf{x}) = P(\,\raisebox{-0.2em}{  }\,) = \frac{1}{Z}\Big\|\,\raisebox{-0.2em}{  }\,\Big\|^2$$

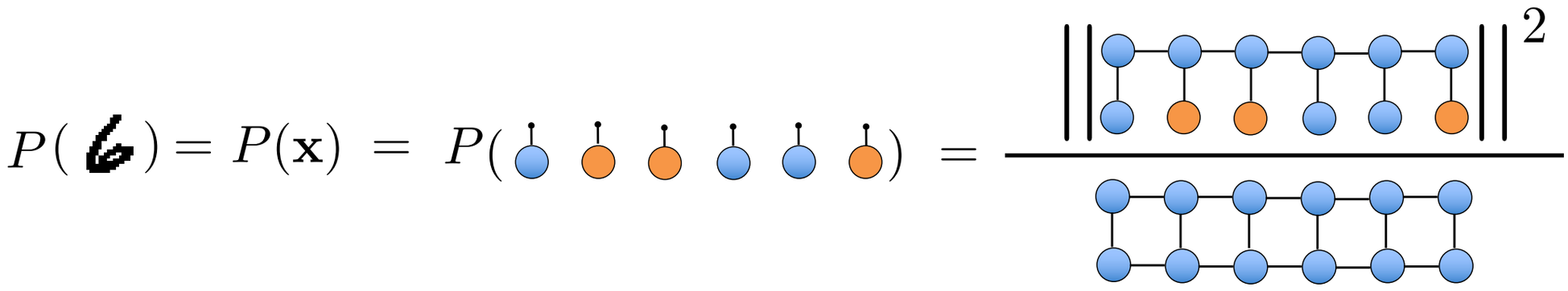
Born’s rule

Z can be computed exactly !

$$Z = \raisebox{-0.2em}{  } = \raisebox{-0.2em}{  }$$

Han, Wang, Fan, Wang, Zhang, Phys. Rev. X 8, 031012 (2018)

Make the MPS distribution close to data distribution

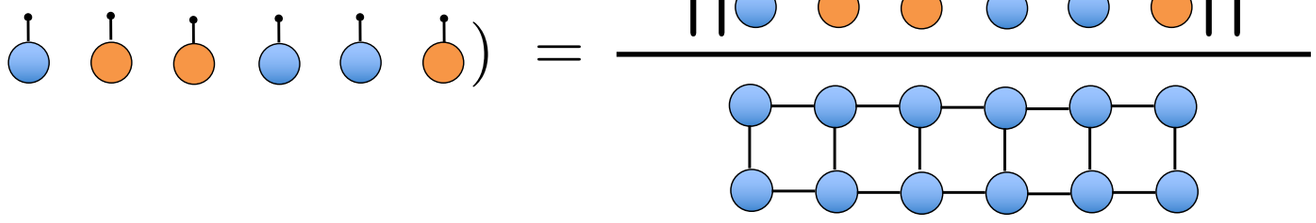


A variational distribution has been parametrized using MPS

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

Make the MPS distribution close to data distribution

$$P(\,\mathbf{x}\,) = P(\mathbf{x}) = P(\,\mathbf{x}\,) = \frac{\left\| \quad \right\|^2}{\quad}$$



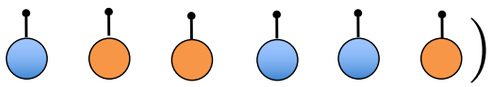
A variational distribution has been parametrized using MPS

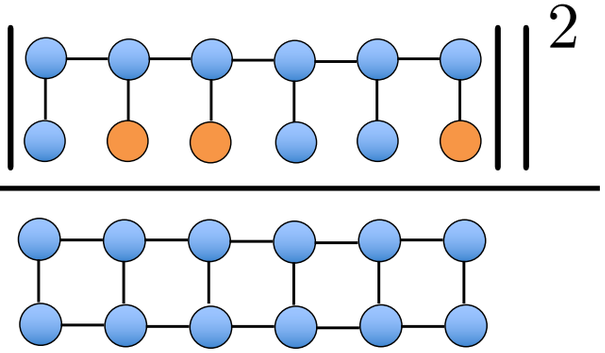
We need to make the distribution as close as possible to the *data distribution*

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

Make the MPS distribution close to data distribution

$$P(\,\mathbf{x}\,) = P(\mathbf{x}) = P(\,\mathbf{x}\,) = \frac{\left\| \quad \right\|^2}{\quad}$$





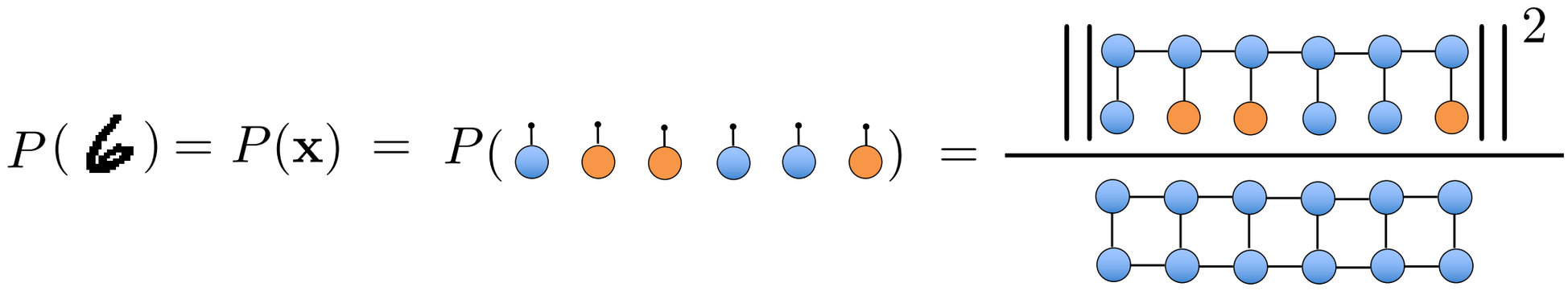
A variational distribution has been parametrized using MPS

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



A variational distribution has been parametrized using MPS

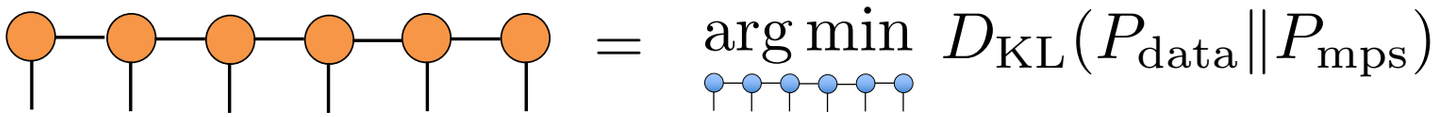
We need to make the distribution as close as possible to the *data distribution*

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

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

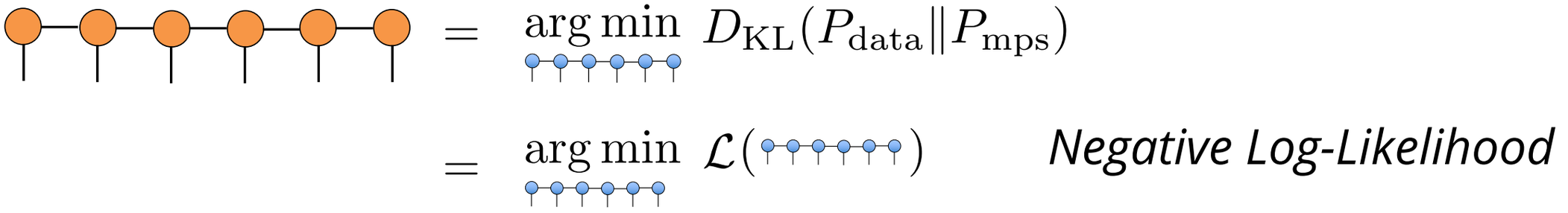
$$D_{\text{KL}}(P_{\text{data}} \| P_{\text{mps}}) = \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{P_{\text{data}}(\mathbf{x})}{P_{\text{mps}}(\mathbf{x})}$$

Learning algorithm

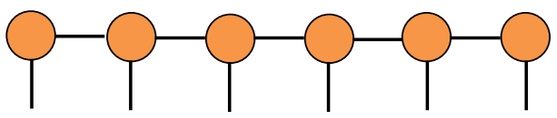


$$= \underset{}{\arg\min} \; D_{\mathrm{KL}}(P_{\mathrm{data}} \| P_{\mathrm{mps}})$$

Learning algorithm



Learning algorithm

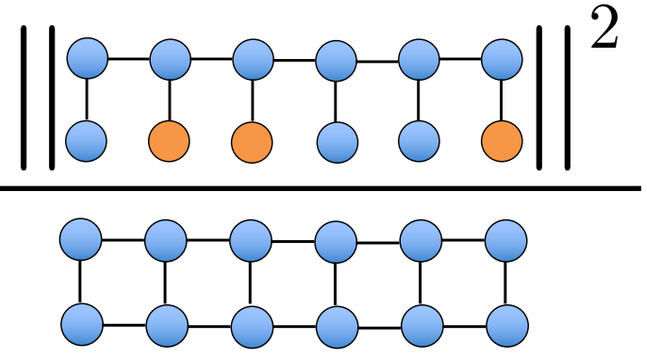


$$= \underset{}{\arg\min}\ D_{\mathrm{KL}}(P_{\mathrm{data}} \| P_{\mathrm{mps}})$$

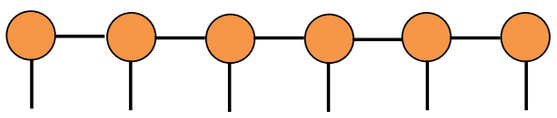
$$= \underset{}{\arg\min}\ \mathcal{L}(\ )$$

Negative Log-Likelihood

$$= \underset{}{\arg\min} \sum_{\in \mathbf{data}} \log$$



Learning algorithm



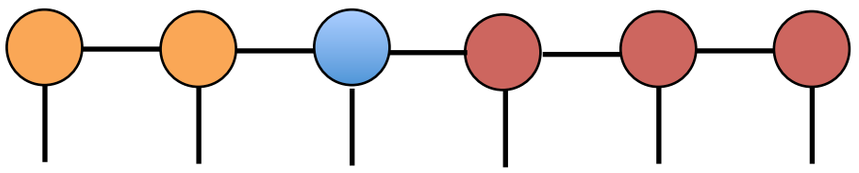
$$= \underset{\text{MPS}}{\arg\min}\ D_{\mathrm{KL}}(P_{\mathrm{data}} \| P_{\mathrm{mps}})$$

$$= \underset{\text{MPS}}{\arg\min}\ \mathcal{L}(\text{MPS})$$

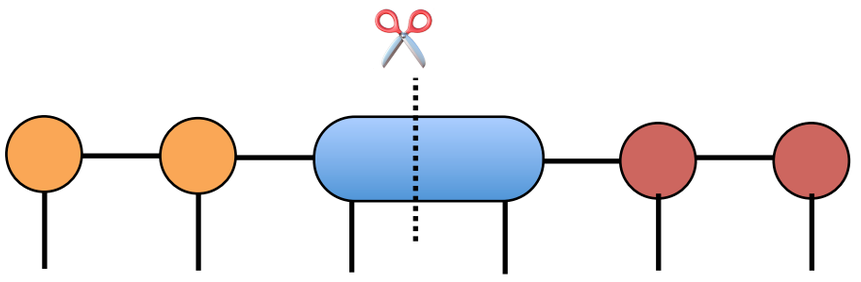
Negative Log-Likelihood

$$= \underset{\text{MPS}}{\arg\min} \sum_{\in \mathbf{data}} \log \frac{\left\| \text{MPS} \right\|^2}{\text{MPS}}$$

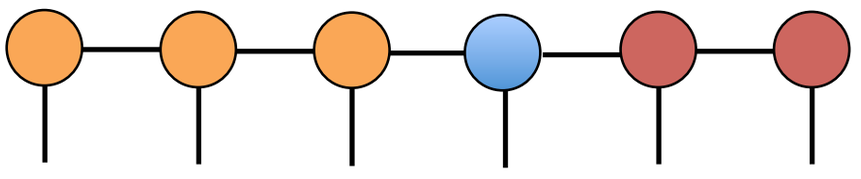
Mixed canonical form



Merging
+ gradient descent *for decreasing NLL*

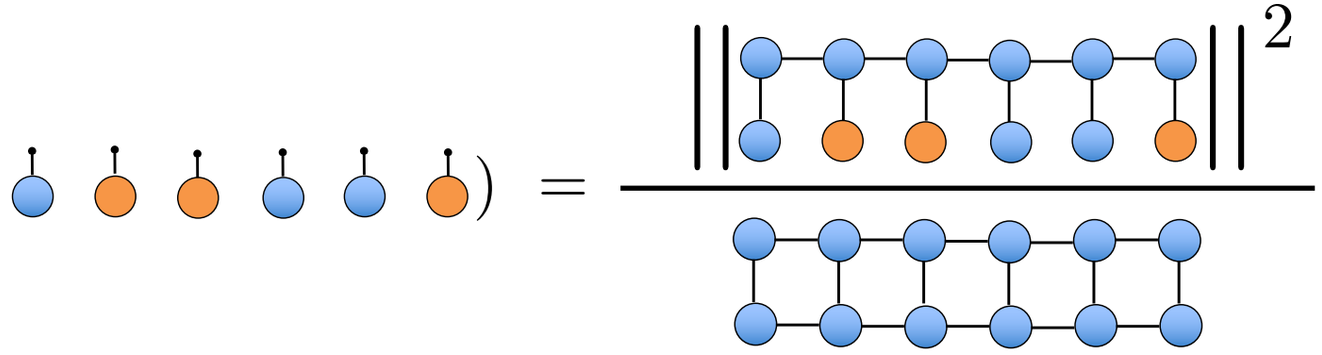


Applying SVD



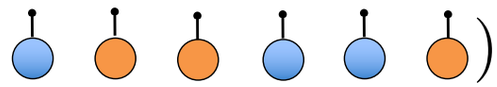
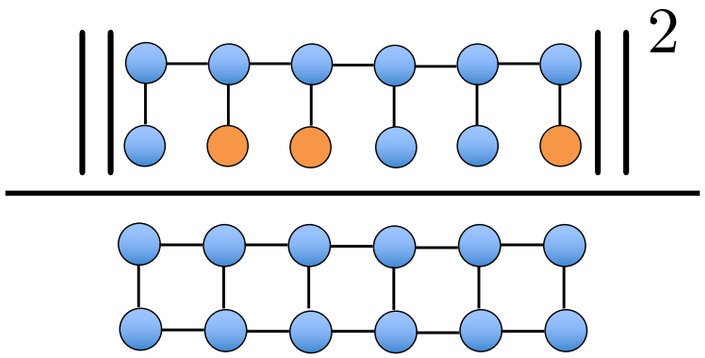
Direct sampling

$$P(\mathbf{x}) = P(\;\;) = \frac{\left\| \quad \right\|^2}{\quad}$$



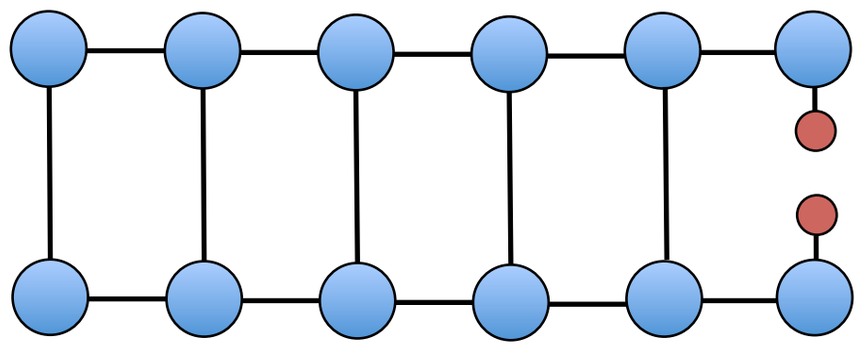
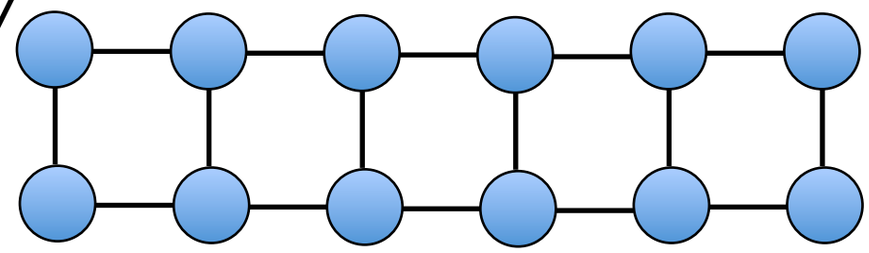
Computing exactly the partition function leads to the ability of computing *arbitrary marginals and conditionals*.

Direct sampling

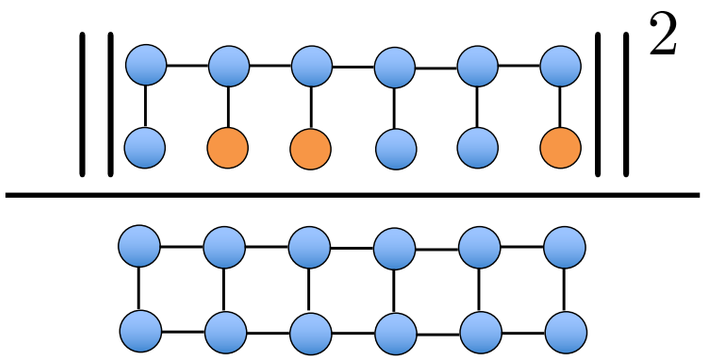
$$P(\mathbf{x}) = P($$  $$) =$$ 

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

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

$$=$$  $$/$$ 

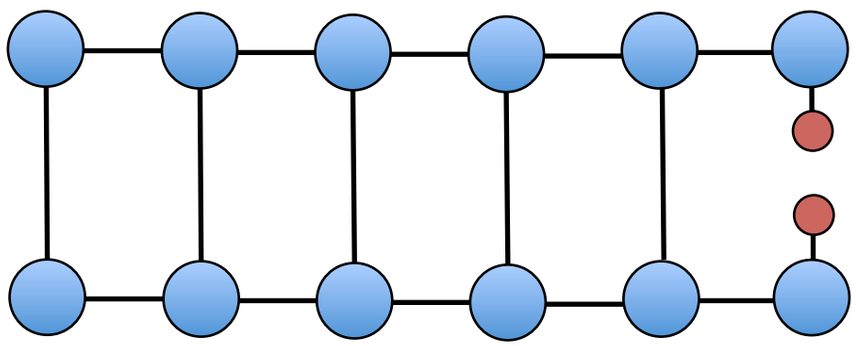
Direct sampling

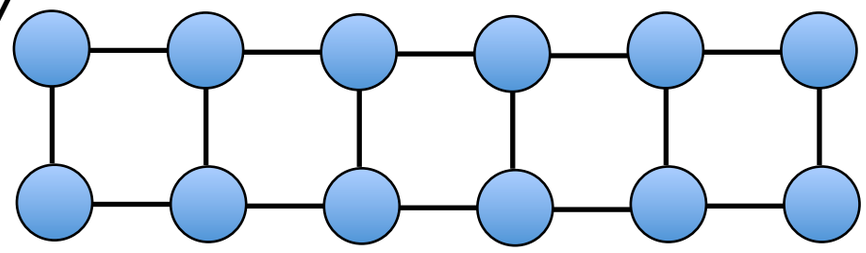
$$P(\mathbf{x}) = P($$  $$) =$$ 

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

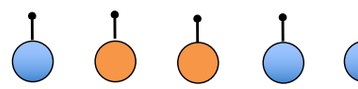
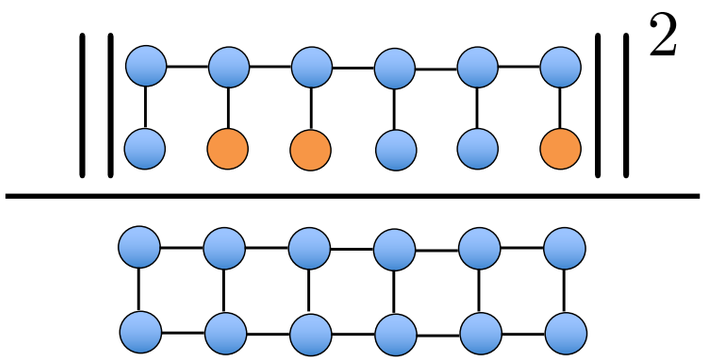
$$P(x_1) = \sum_{x_2} \sum_{x_3} \cdots \sum_{x_n} P(\mathbf{x})$$

generate x_1

$=$ 

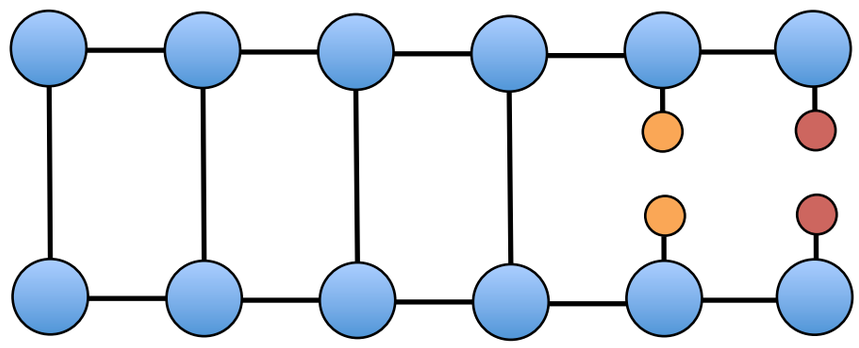


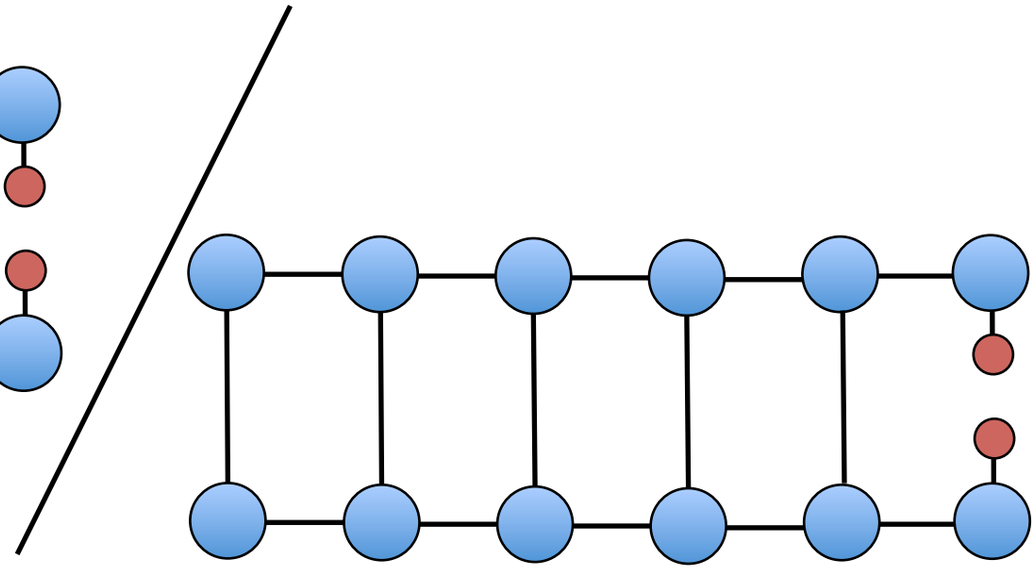
Direct sampling

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

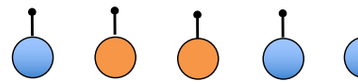
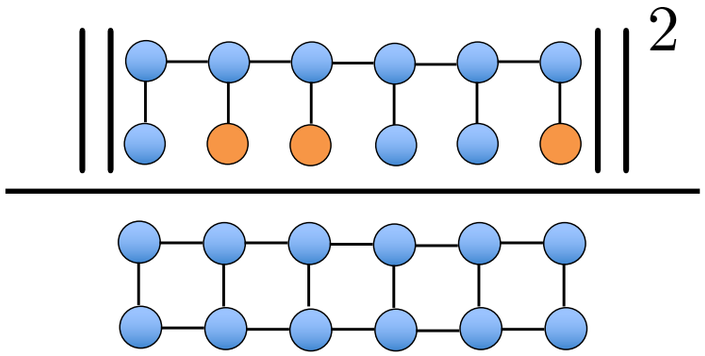
Computing exactly the partition function leads to the ability of compute *arbitrary marginals and conditionals*.

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

$=$ 



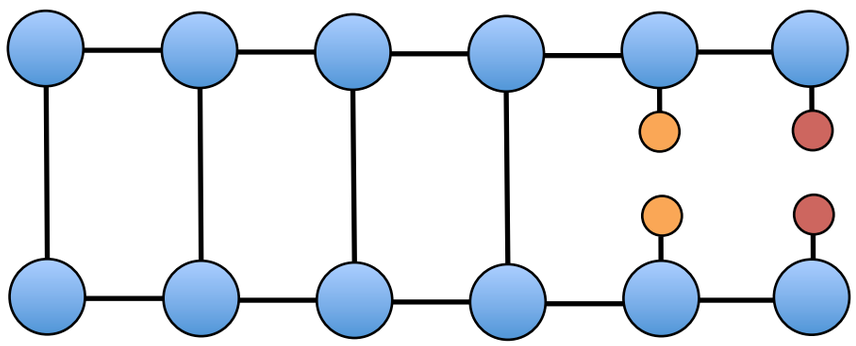
Direct sampling

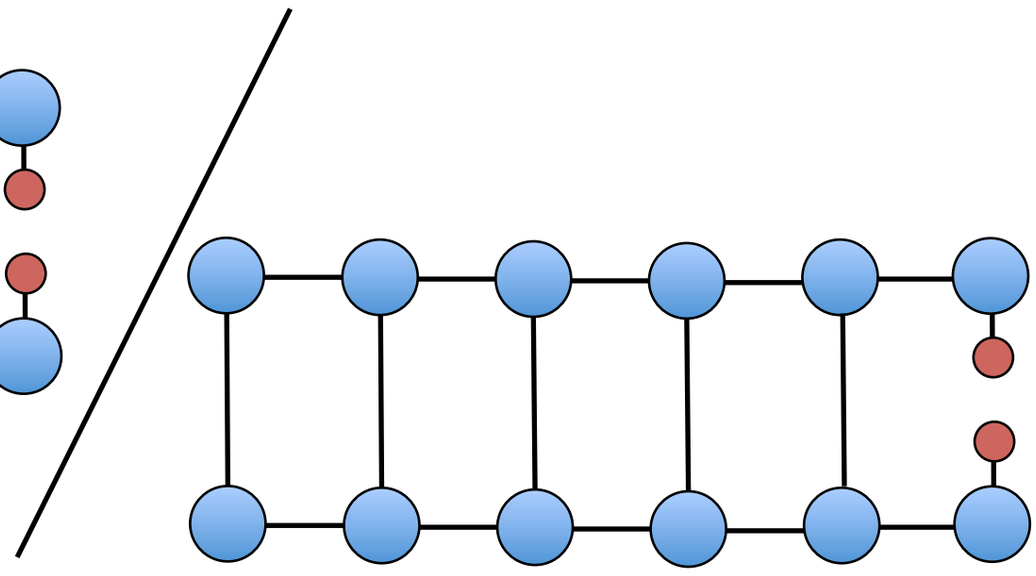
$$P(\mathbf{x}) = P($$  $$) =$$ 

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

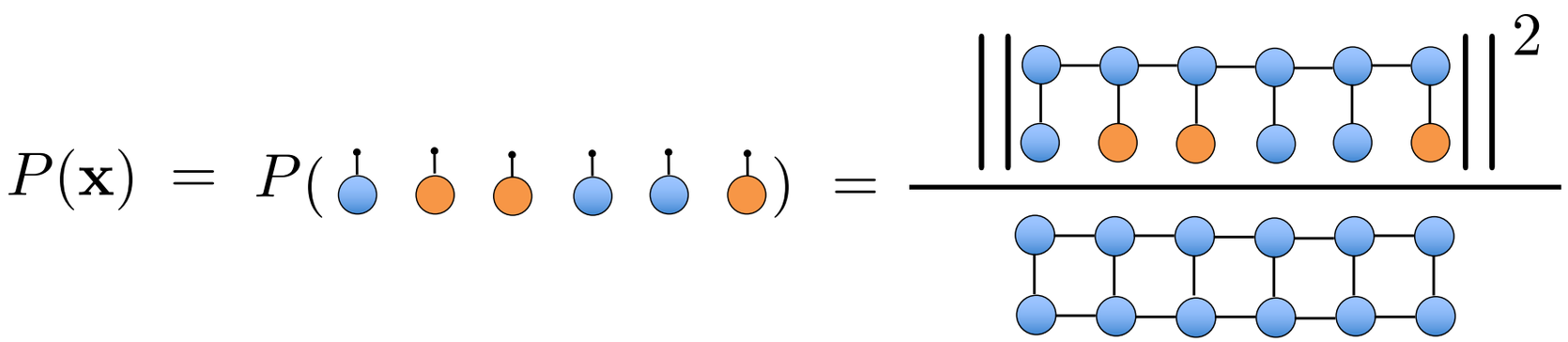
$$P(x_2|x_1) = P(x_1, x_2)/p(x_1)$$

generate x_2

$=$ 

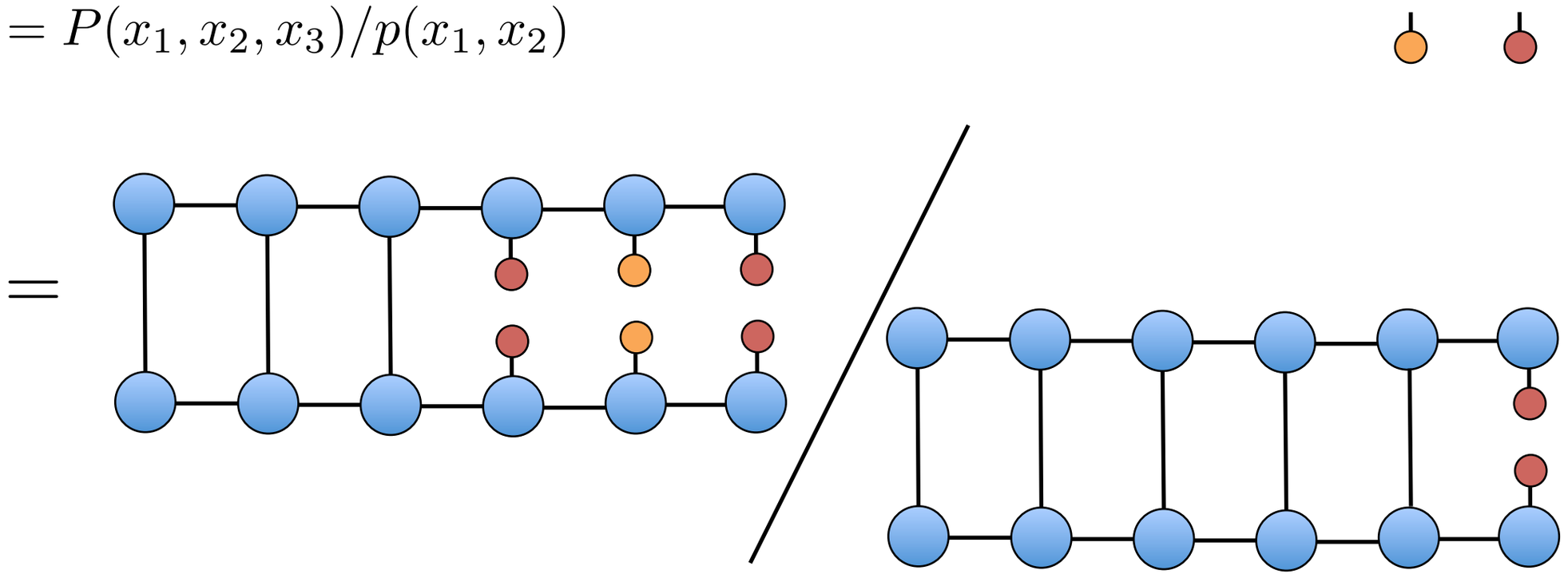


Direct sampling

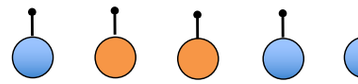
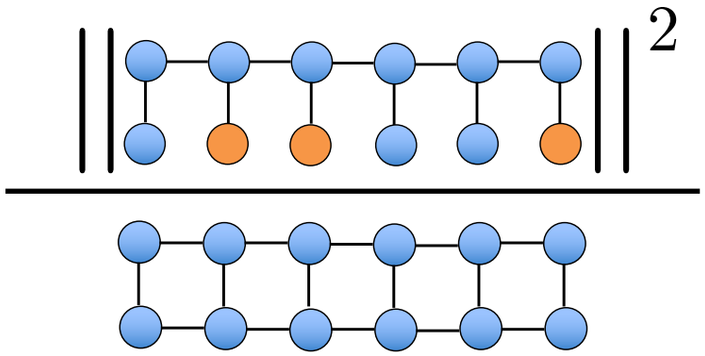


Computing exactly the partition function leads to the ability of compute *arbitrary marginals and conditionals*.

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

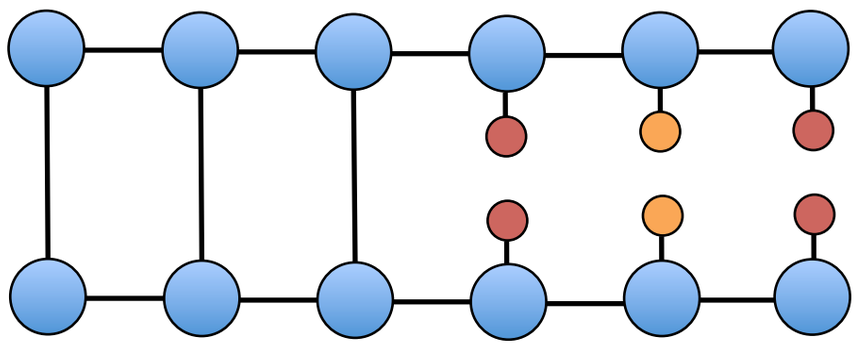
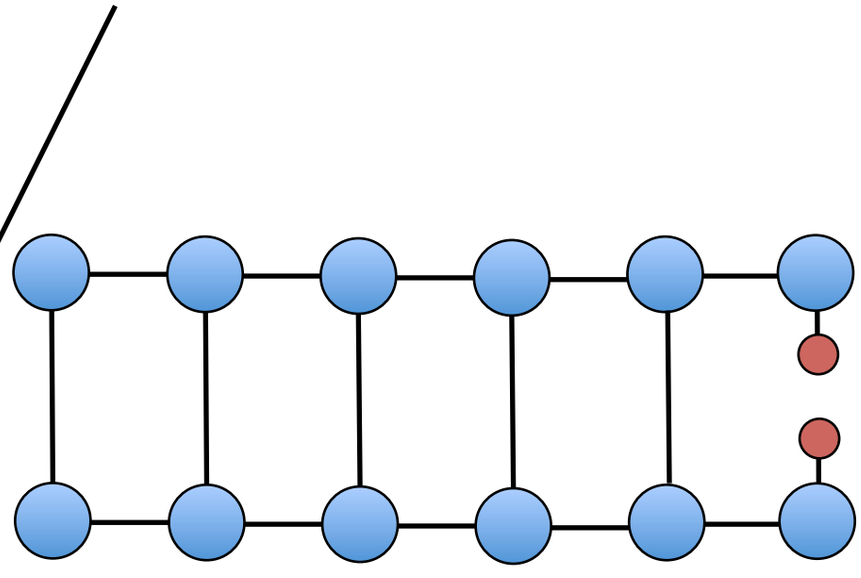


Direct sampling

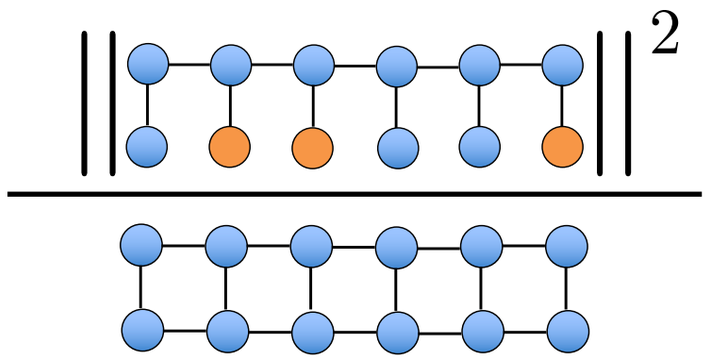
$$P(\mathbf{x}) = P($$  $$) =$$ 

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

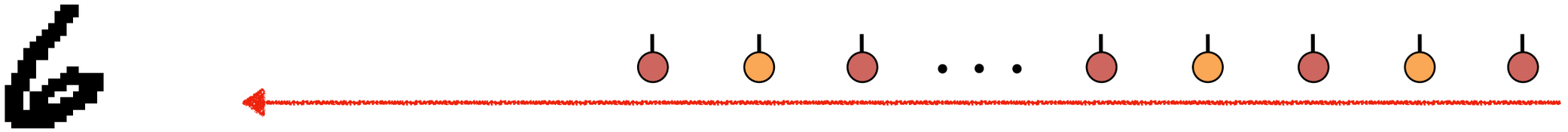
$P(x_3|x_1, x_2) = P(x_1, x_2, x_3)/p(x_1, x_2)$ $\xrightarrow{\text{generate } x_3}$ 

$=$  $/$ 

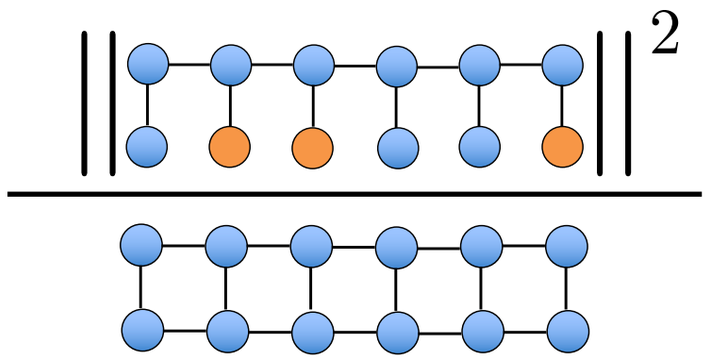
Direct sampling

$$P(\mathbf{x}) = P(\;\;) =$$ 

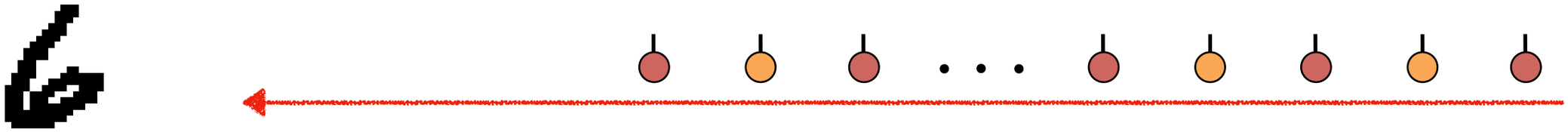
Computing exactly the partition function leads to the ability of compute *arbitrary marginals and conditionals*.



Direct sampling

$$P(\mathbf{x}) = P(\;\;) =$$ 

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

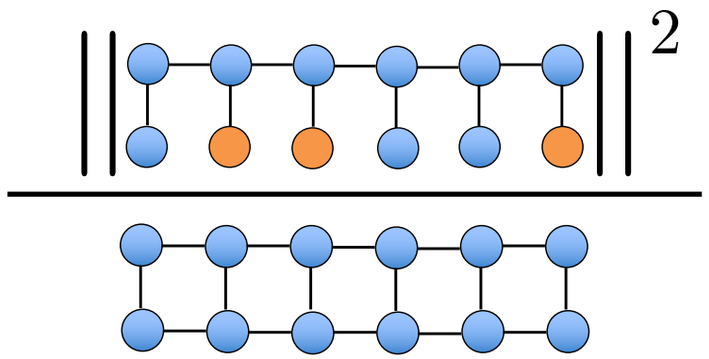


known as *ancestral sampling* in machine learning and
known as “*perfect sampling with unitary tensor networks”* in physics.

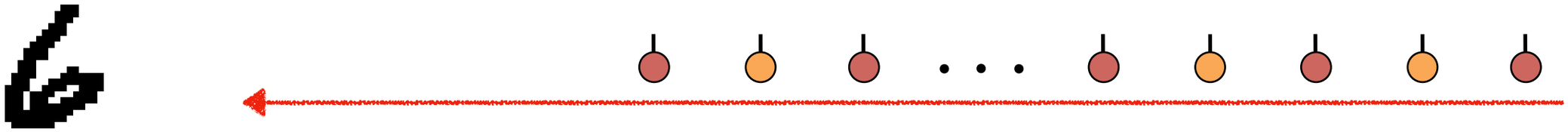
Bishop, “Machine Learning and Pattern Recognition” 2006

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

Direct sampling

$$P(\mathbf{x}) = P(\;\;) =$$ 

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.

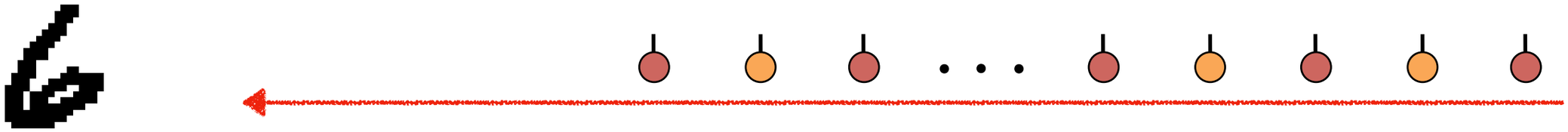
It is a *unbiased/perfect* sampling, without introducing any autocorrelation

Bishop, "Machine Learning and Pattern Recognition" 2006

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

Direct sampling

$$P(\mathbf{x}) = P(\;\;) = \frac{\left\| \quad \right\|^2}{\quad}$$



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.

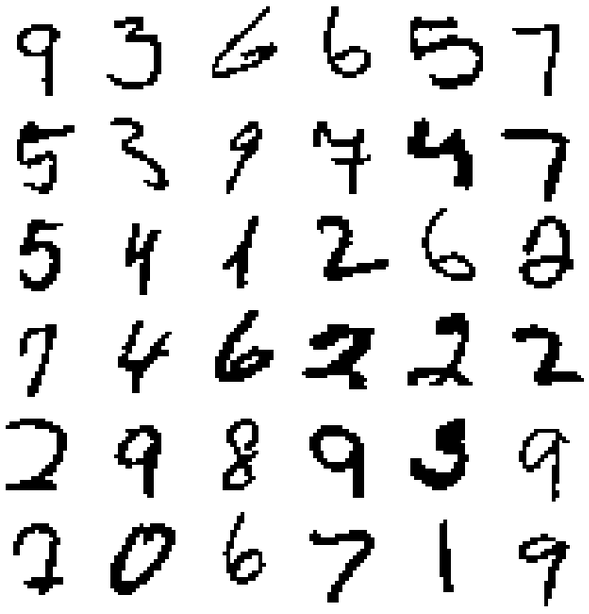
It is a *unbiased/perfect* sampling, without introducing any autocorrelation

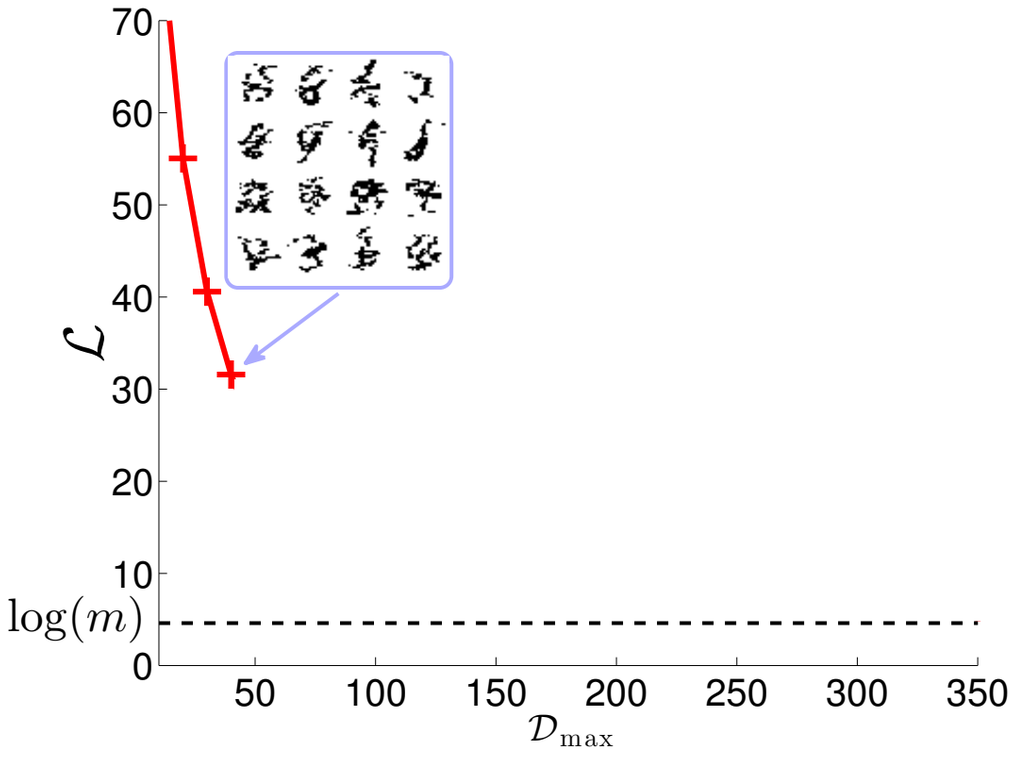
Readily parallelizable

Bishop, "Machine Learning and Pattern Recognition" 2006

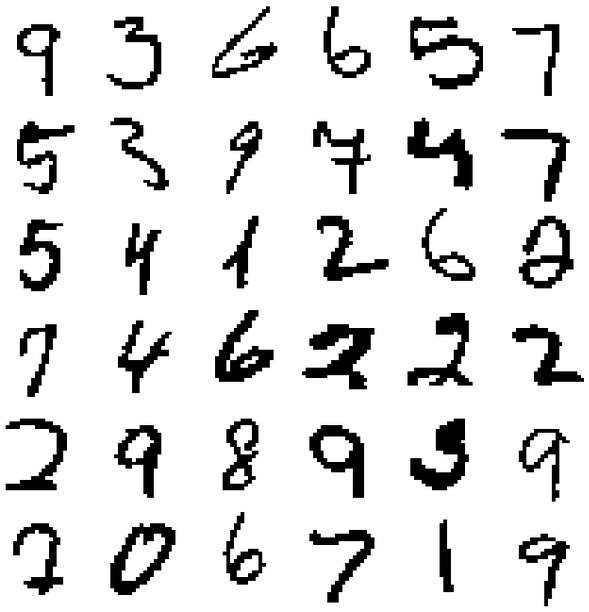
Ferris,Vidal, Phys. Rev. B 85, 165146 (2012)

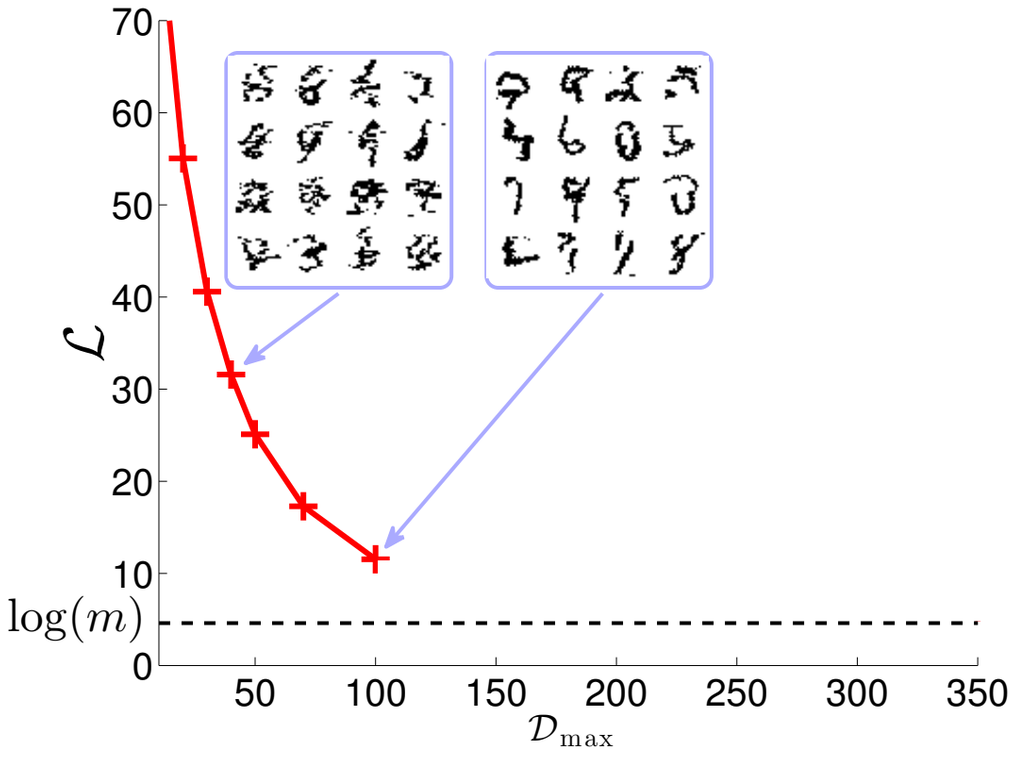
On the MNIST dataset



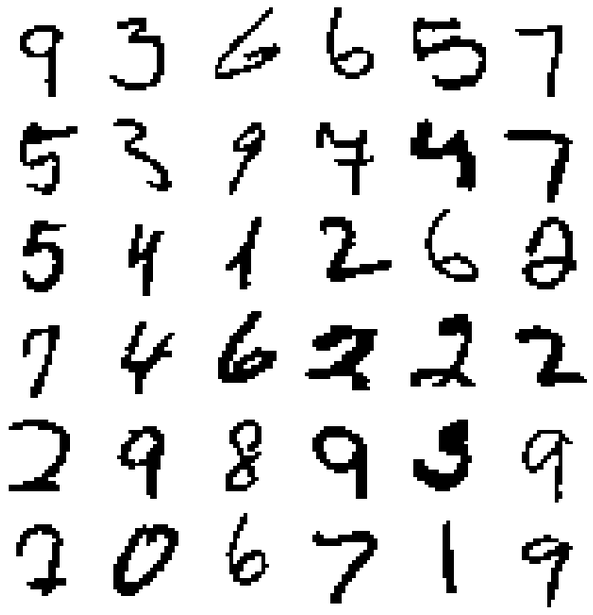


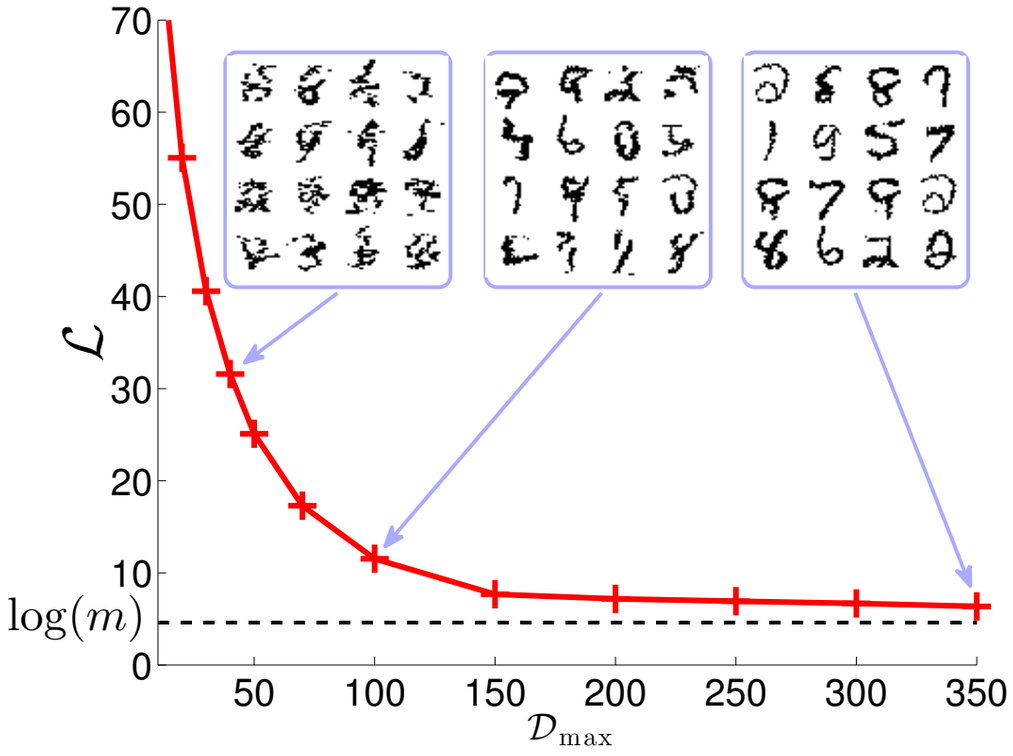
On the MNIST dataset





On the MNIST dataset

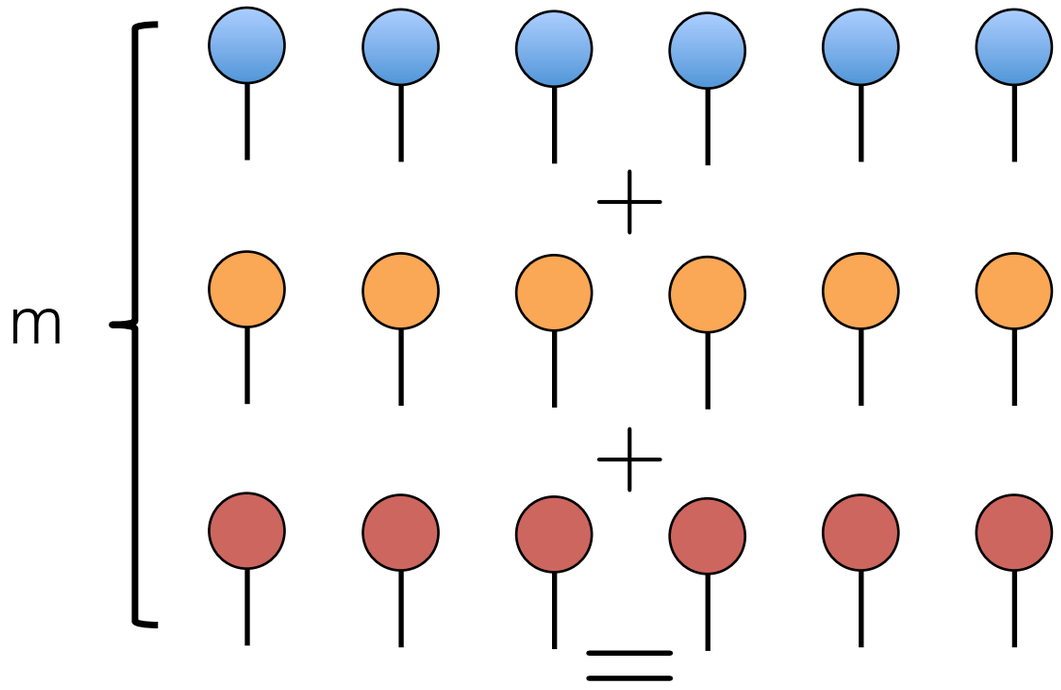




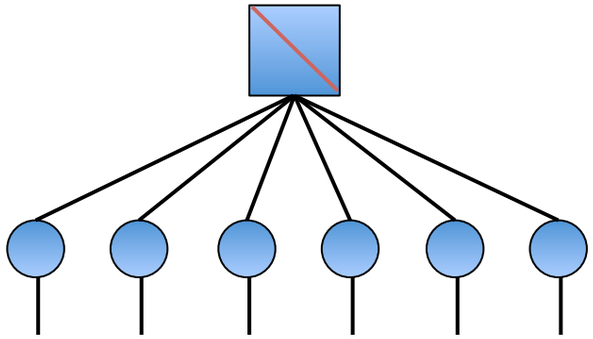
Exact memorizing patterns

Exact memorizing patterns

m images

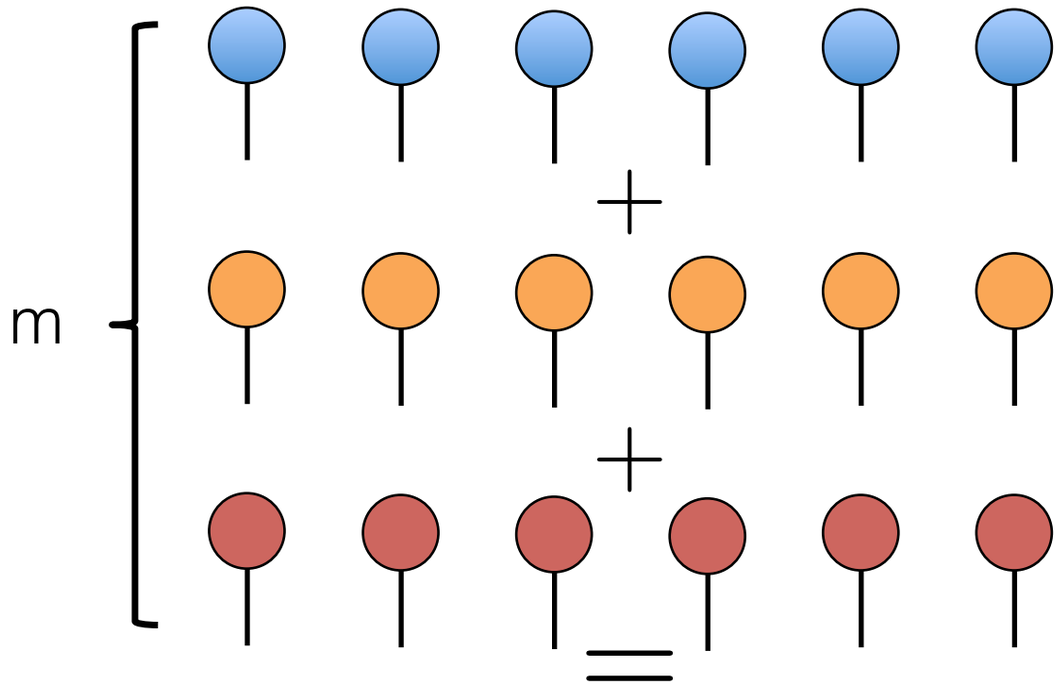


Exact learning of *m* images amounts to construct a CP tensor with rank *m.*



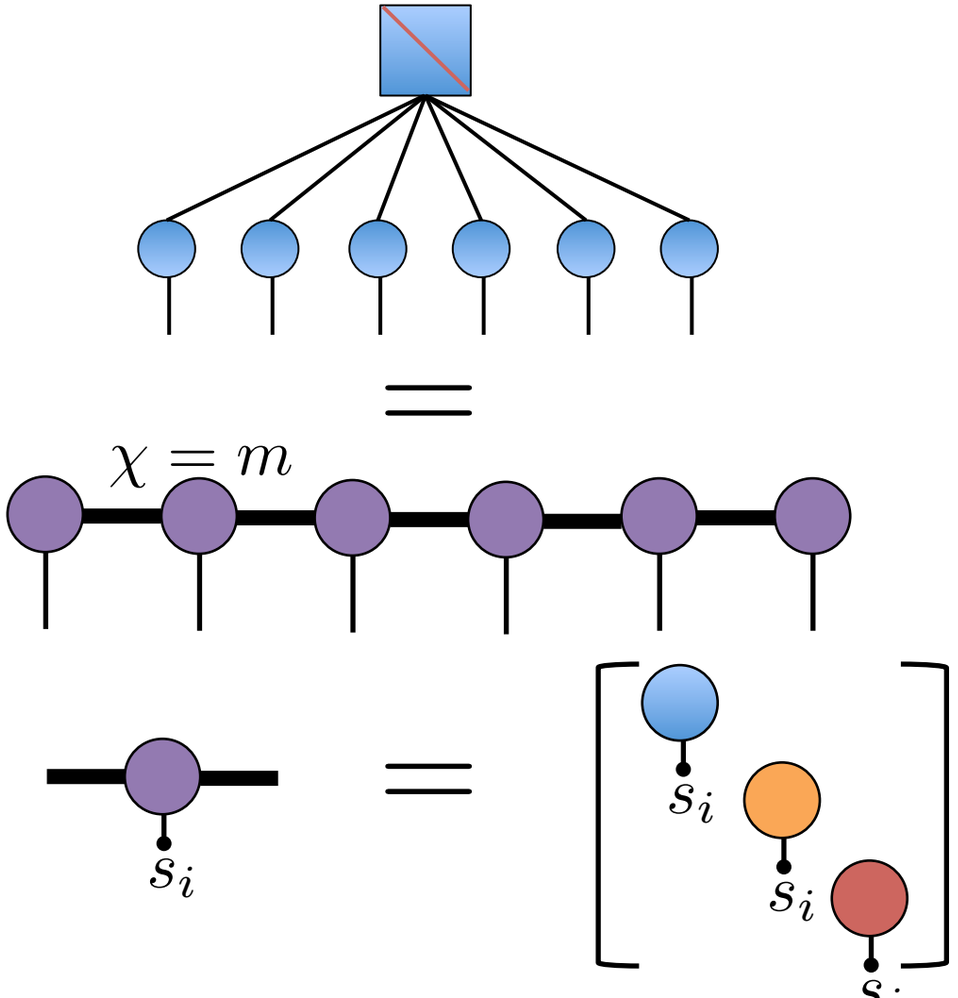
Exact memorizing patterns

m images



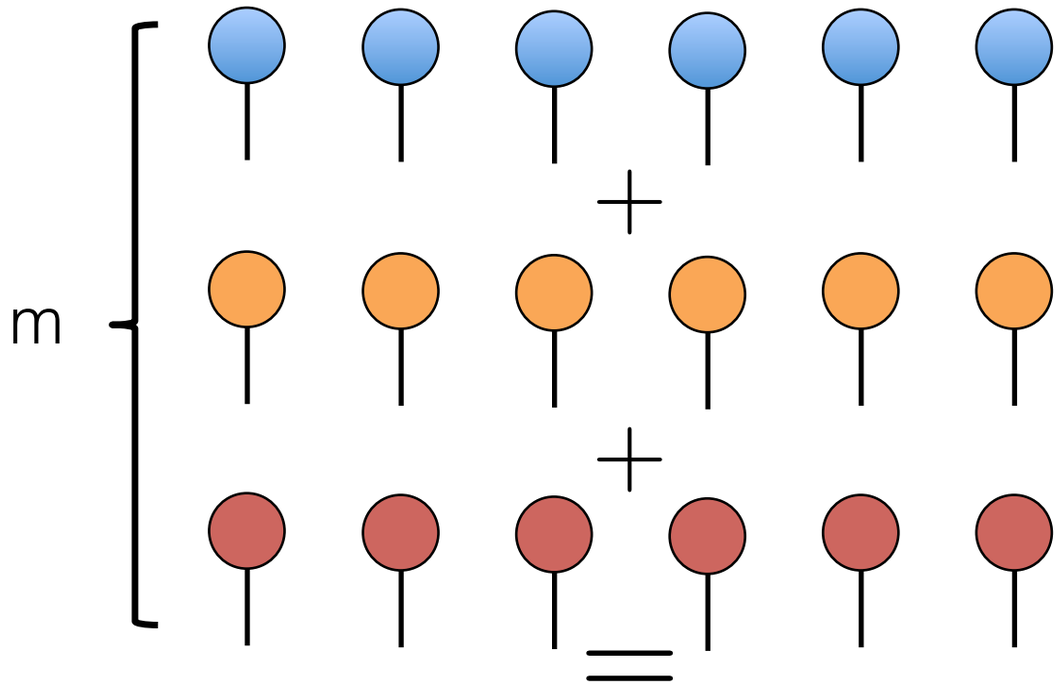
Exact learning of *m* images amounts to construct a CP tensor with rank *m.*

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



Exact memorizing patterns

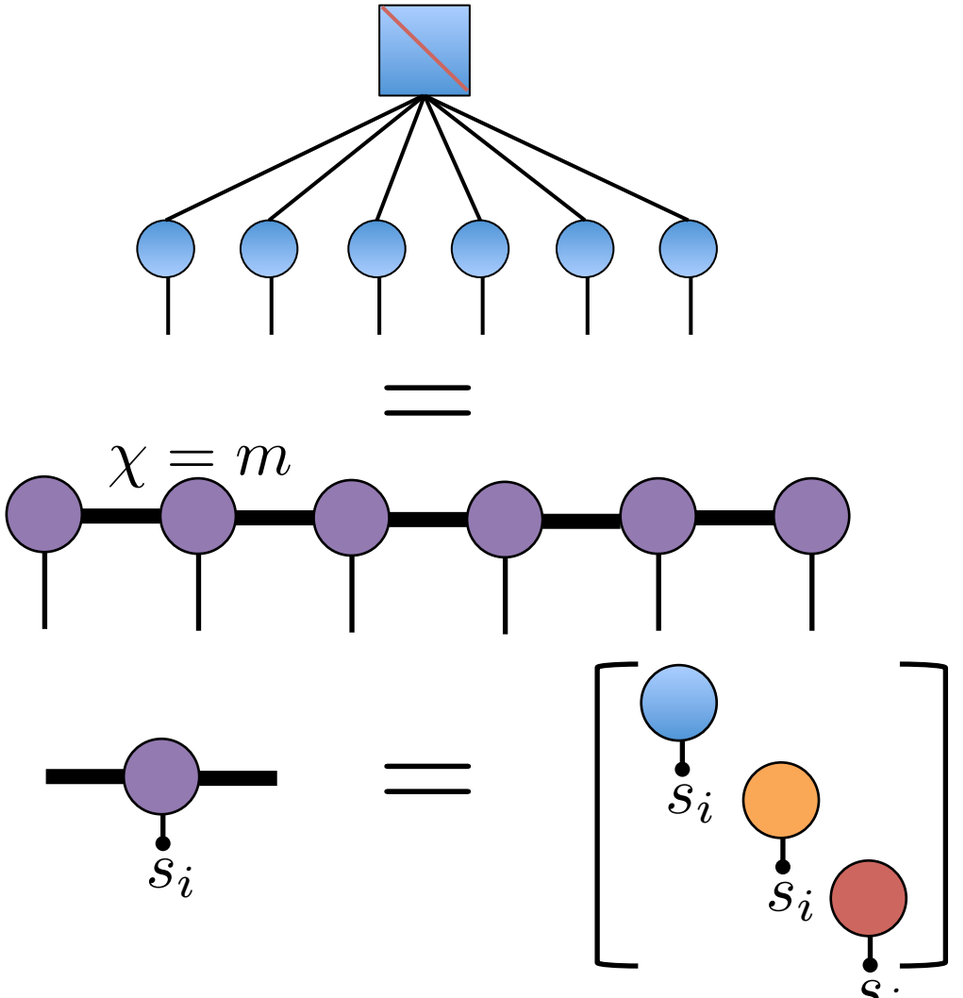
m images



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

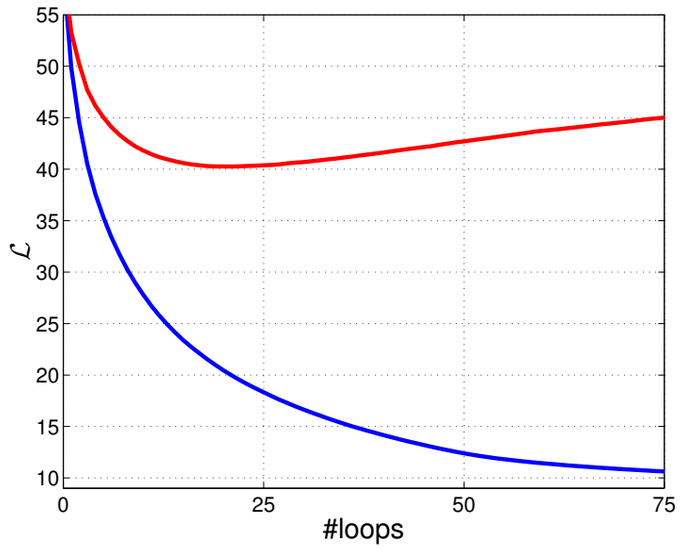
This can be converted to a MPS with bond dimension *m.*

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* !

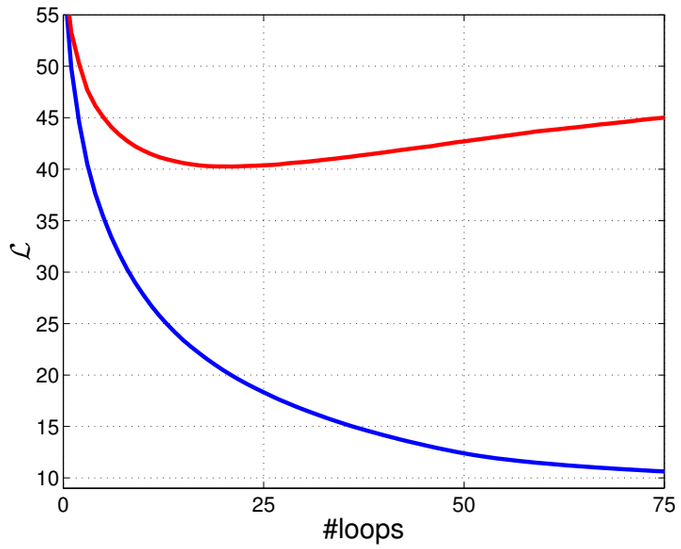


1000 training images

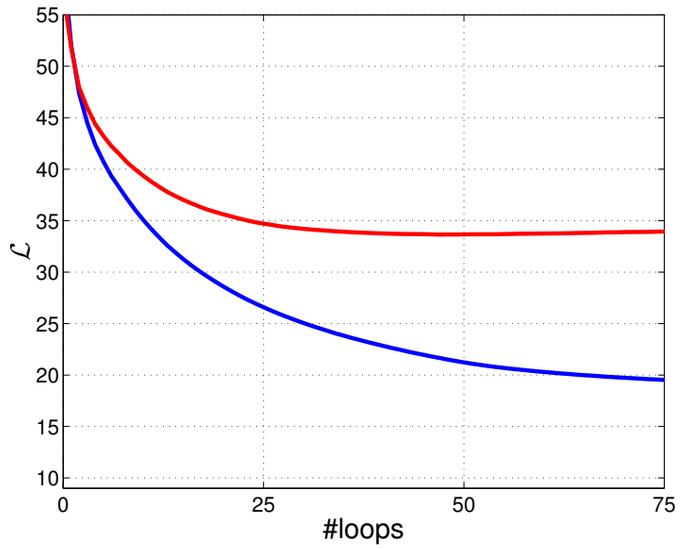
Han, Wang, Fan, Wang, Zhang, Phys. Rev. X 8, 031012 (2018)

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

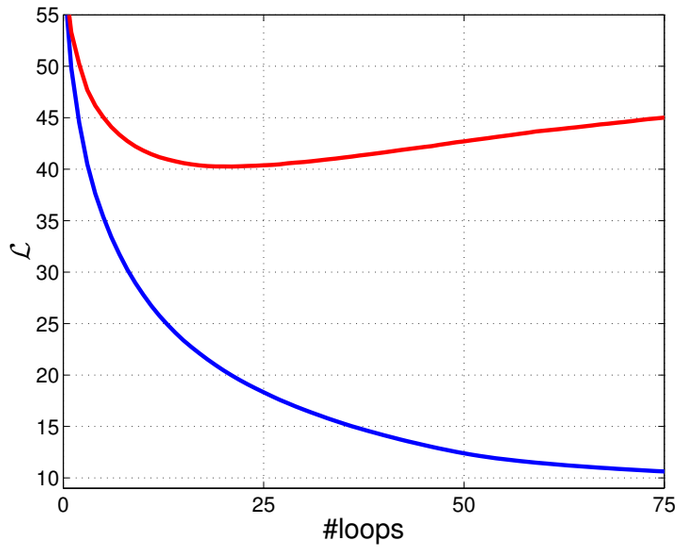


10000 training images

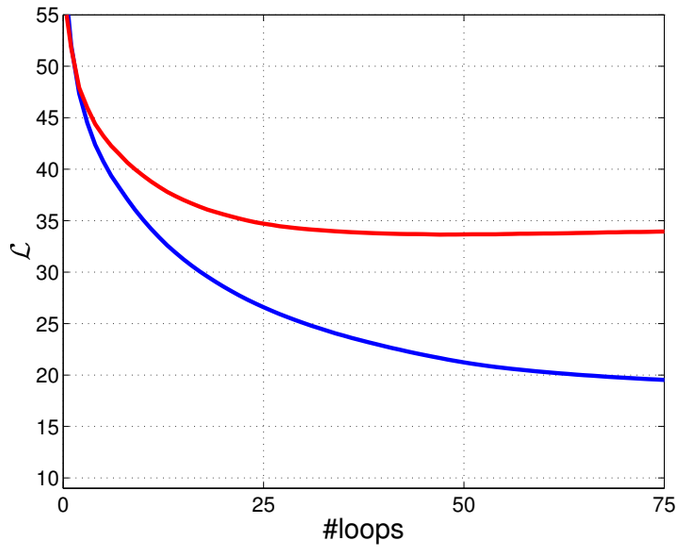
Han, Wang, Fan, Wang, Zhang, Phys. Rev. X 8, 031012 (2018)

Generalization to unseen images

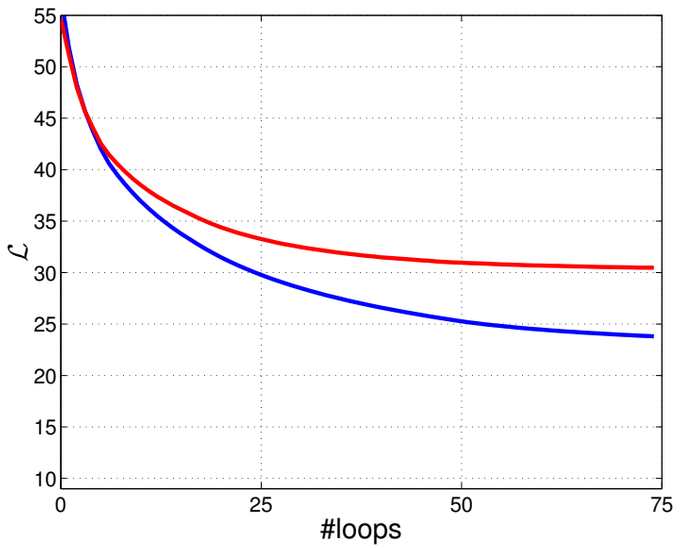
However, exact memorizing will give zero probability to all images except those in the training set —— *no generalization power* !



1000 training images



10000 training images

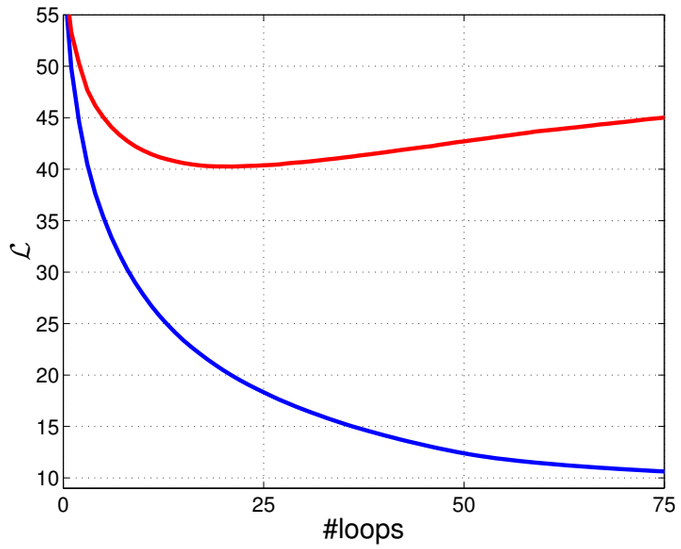


60000 training images

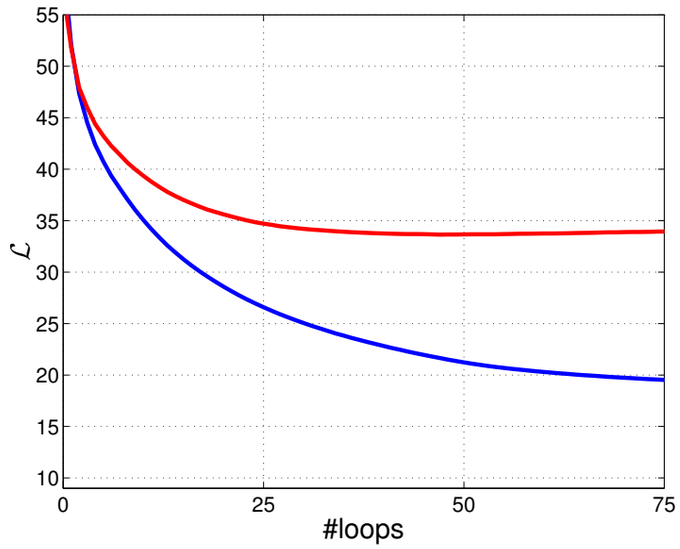
Han, Wang, Fan, Wang, Zhang, Phys. Rev. X 8, 031012 (2018)

Generalization to unseen images

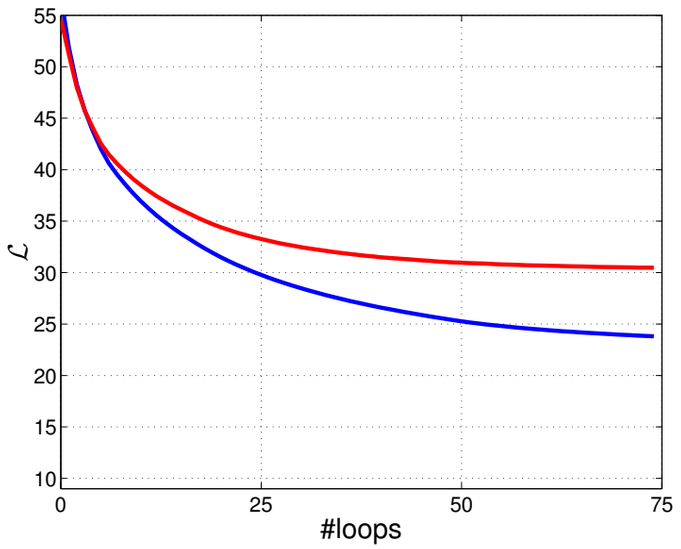
However, exact memorizing will give zero probability to all images except those in the training set —— *no generalization power* !



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.*

Han, Wang, Fan, Wang, Zhang, Phys. Rev. X 8, 031012 (2018)

Image reconstruction

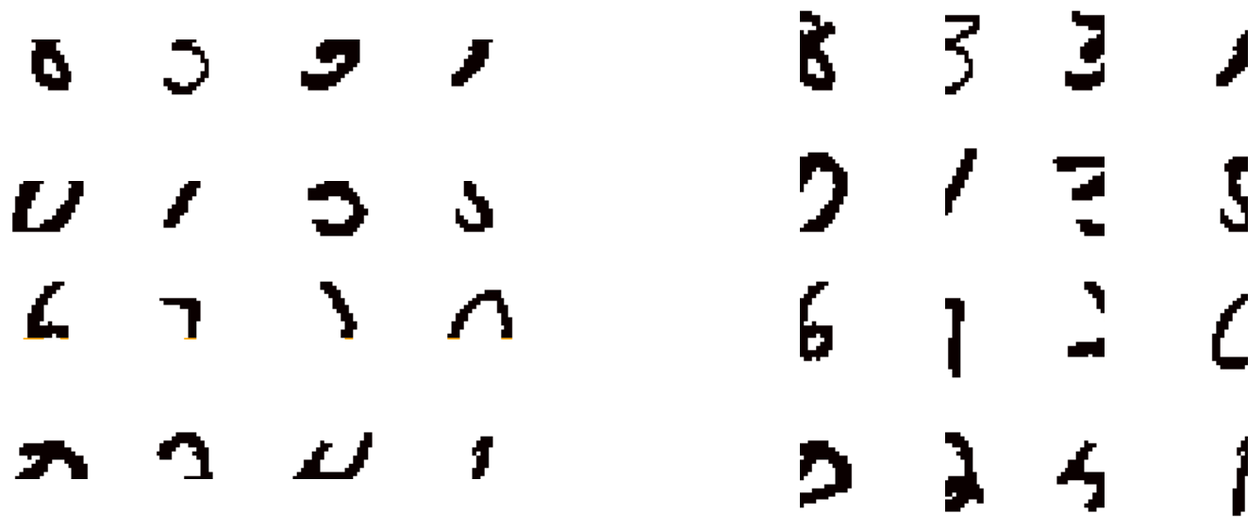
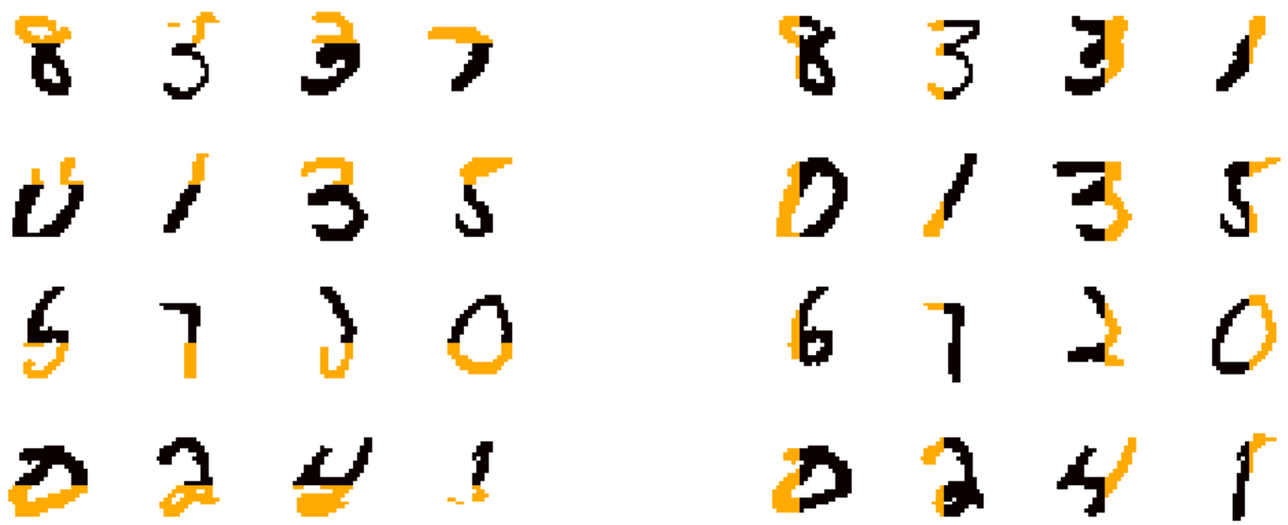


Image reconstruction



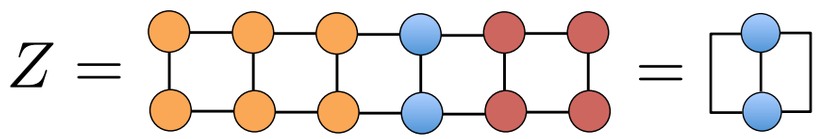
MPS Born machine

MPS Born machine

- Features:

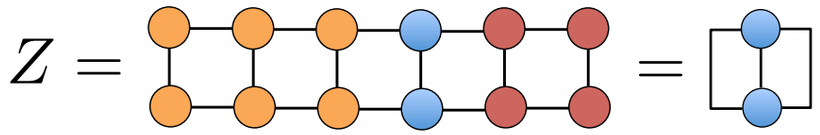
MPS Born machine

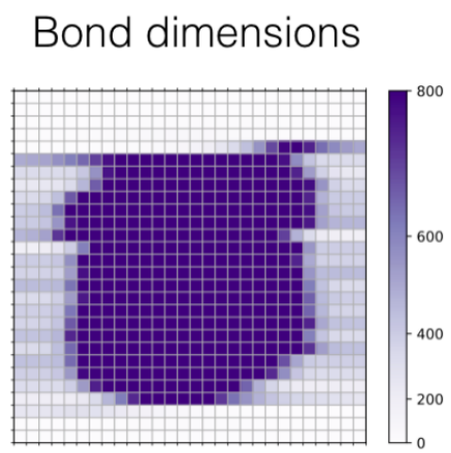
- Features:
  - Trackable likelihood via efficient tensor contractions.



MPS Born machine

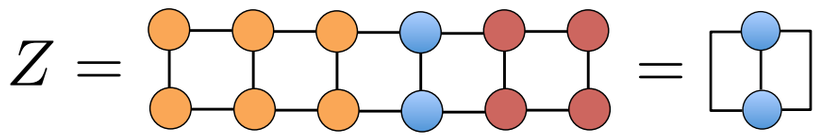
- Features:
  - Trackable likelihood via efficient tensor contractions.
  - Adaptively allocating bond dimension.

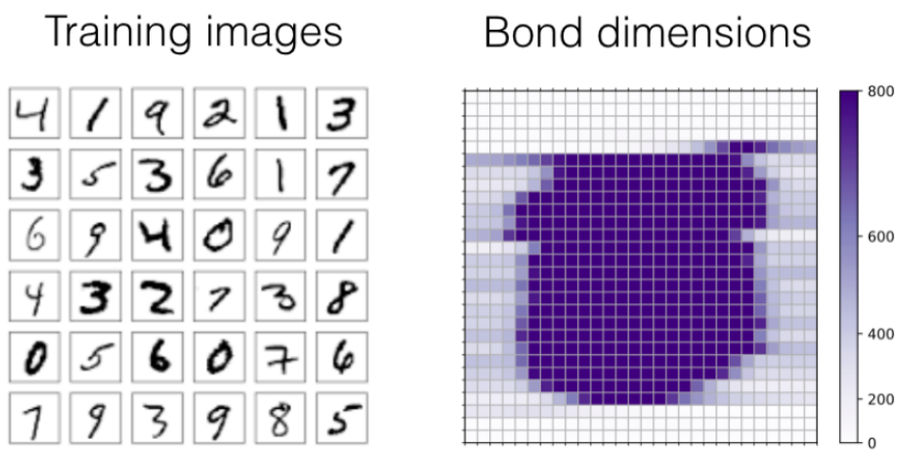




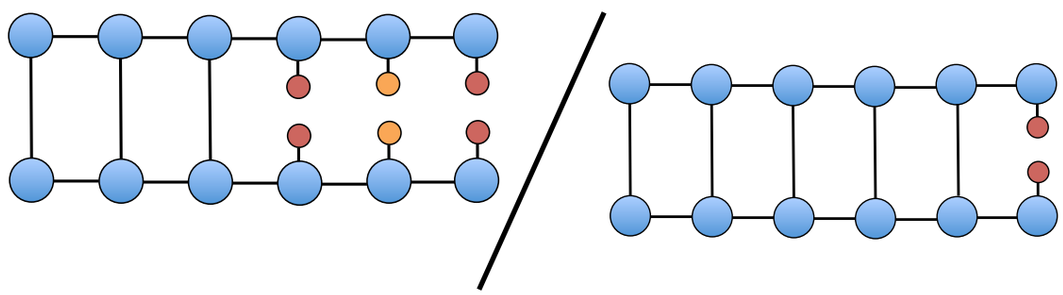
MPS Born machine

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



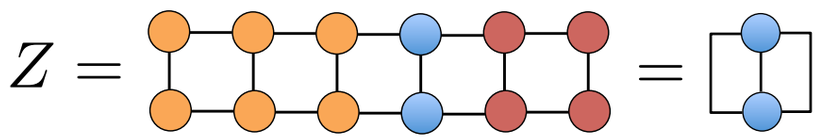


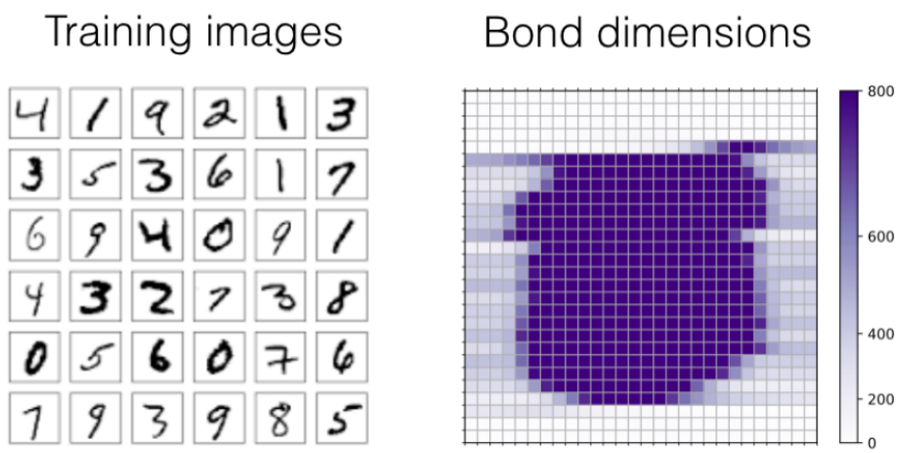
$$p(\boldsymbol{x}) = \prod_i \frac{p(\boldsymbol{x}_{<i+1})}{p(\boldsymbol{x}_{<i})} = \prod_i p(x_i|\boldsymbol{x}_{<i})$$



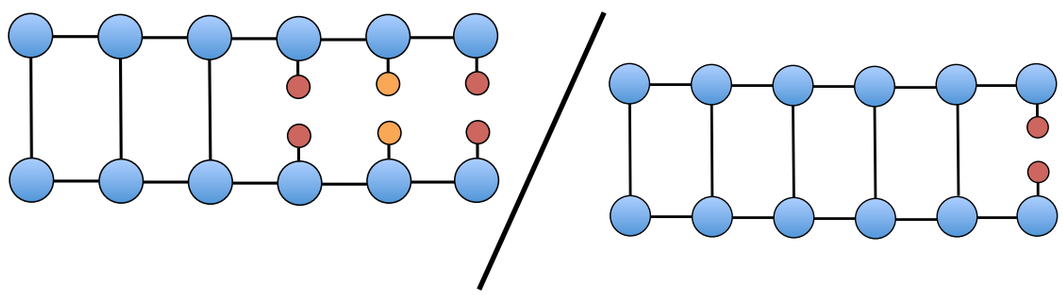
MPS Born machine

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



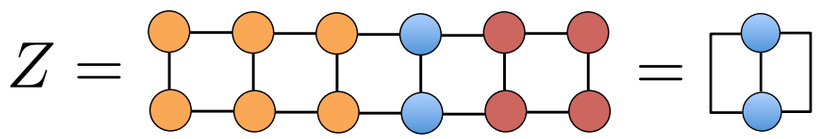


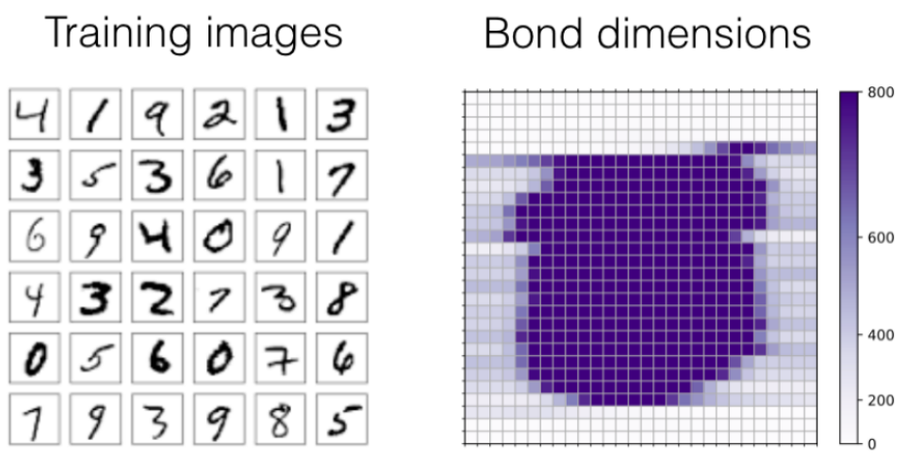
$$p(\boldsymbol{x}) = \prod_i \frac{p(\boldsymbol{x}_{<i+1})}{p(\boldsymbol{x}_{<i})} = \prod_i p(x_i|\boldsymbol{x}_{<i})$$



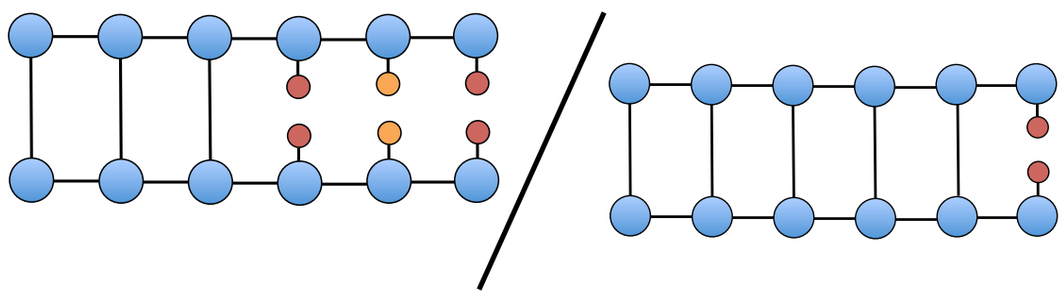
MPS Born machine

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



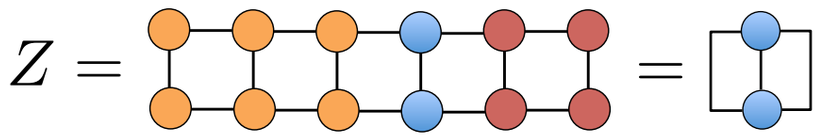


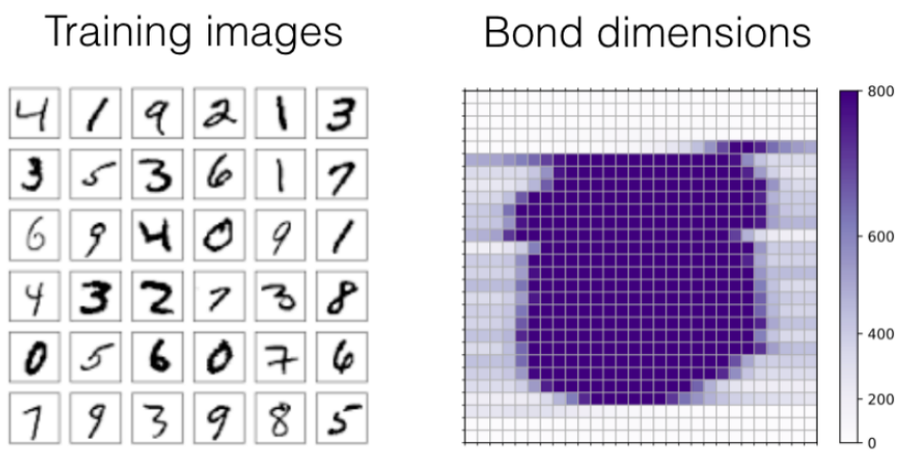
$$p(\boldsymbol{x}) = \prod_i \frac{p(\boldsymbol{x}_{<i+1})}{p(\boldsymbol{x}_{<i})} = \prod_i p(x_i|\boldsymbol{x}_{<i})$$



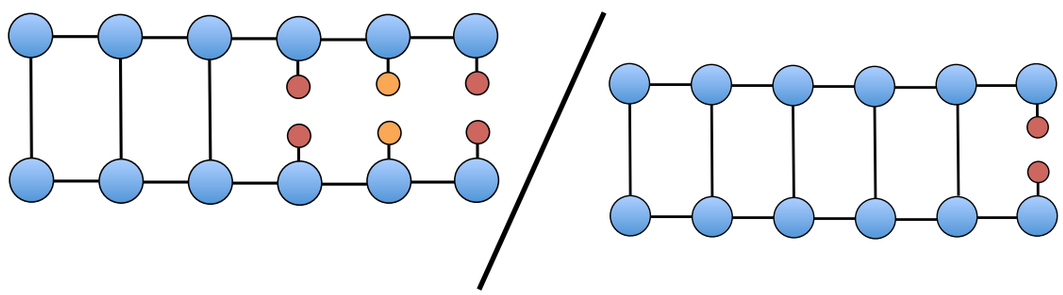
MPS Born machine

- 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



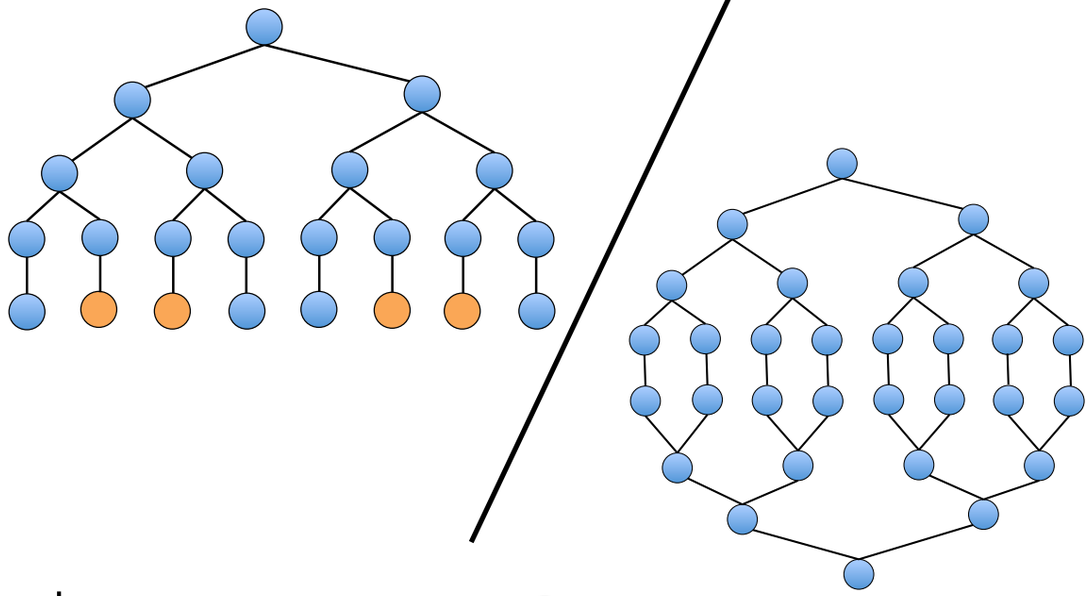


$$p(\boldsymbol{x}) = \prod_i \frac{p(\boldsymbol{x}_{<i+1})}{p(\boldsymbol{x}_{<i})} = \prod_i p(x_i|\boldsymbol{x}_{<i})$$

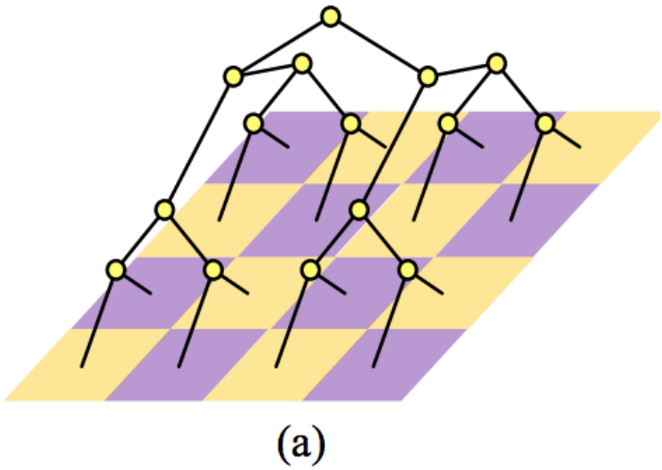


Tree Tensor Network Born machine

$$P(\mathbf{x}) = P(\;\;\;\;\;\;) =$$



- 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



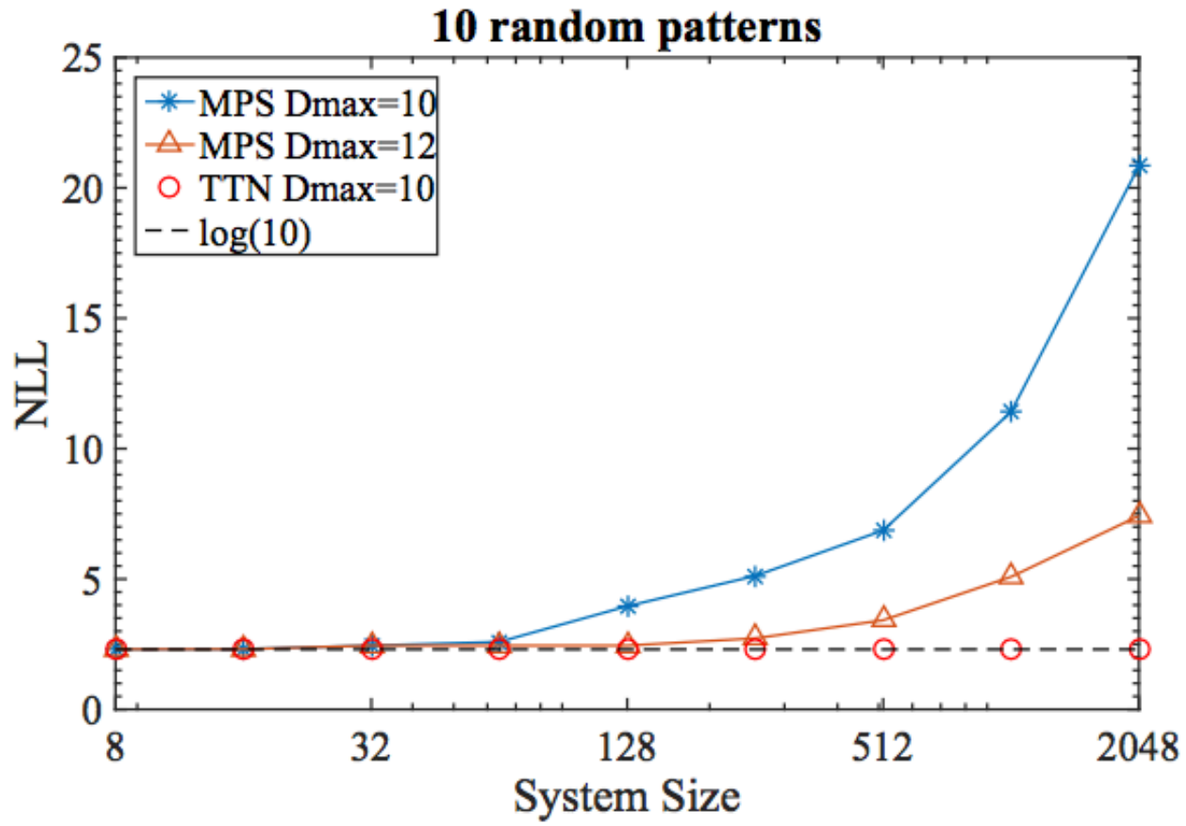
(a)

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

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

Cheng, Wang, Xiang, Zhang, arXiv:1901.02217

Better correlation length in practice

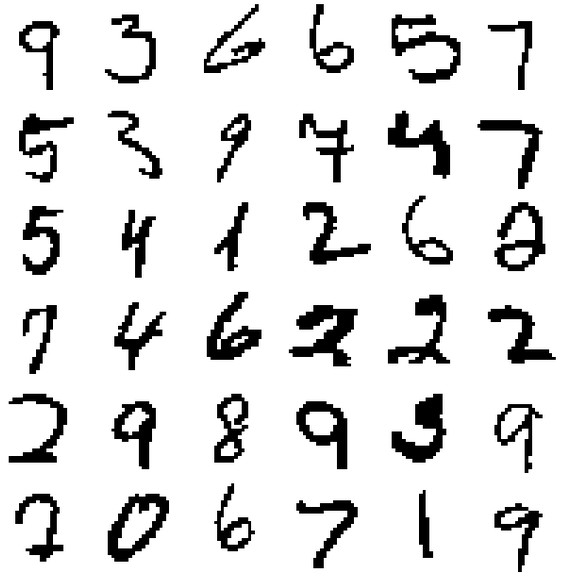


Random patterns are good benchmarks for testing the *capacity*

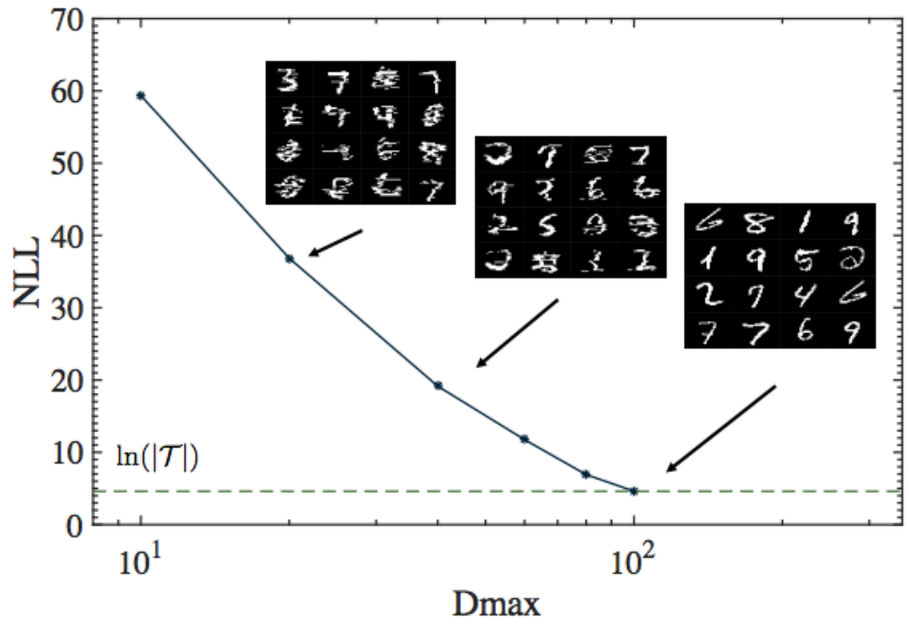
Theoretical limits for both MPS and TTN born machines are *ln(m)*

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

On the MNIST dataset

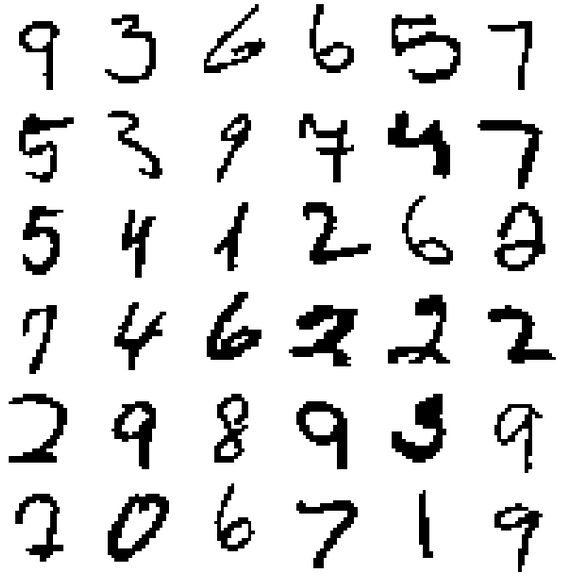


Trained on 100 images

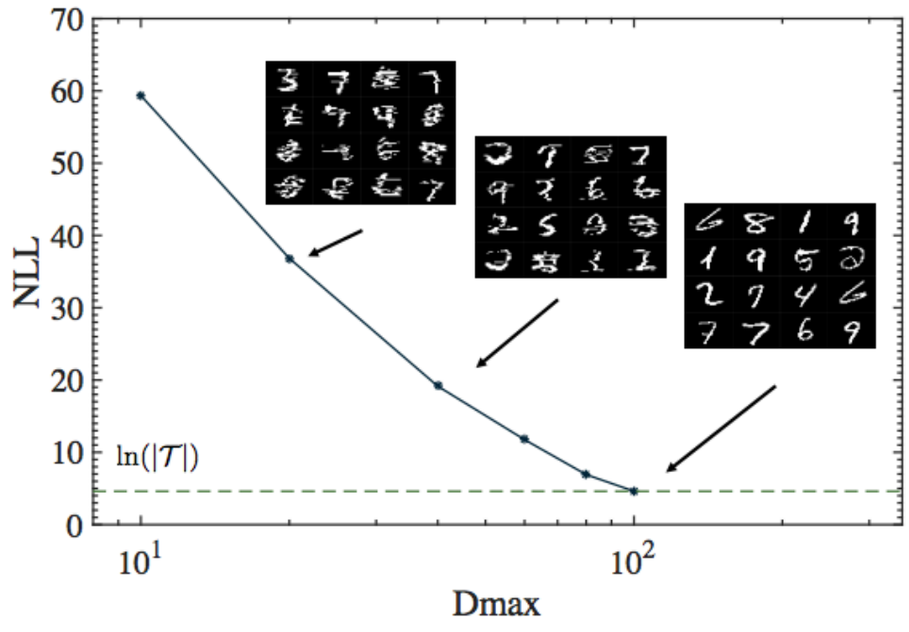


Cheng, Wang, Xiang, Zhang, arXiv:1901.02217

On the MNIST dataset



Trained on 100 images



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

* stands for approximated NLL.

Cheng, Wang, Xiang, Zhang, arXiv:1901.02217

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...)