FLATIRON INSTITUTE
Center for Computational Quantum Physics

MATRICES, VECTORS, TENSORS, OH MY

KARL PIERCE

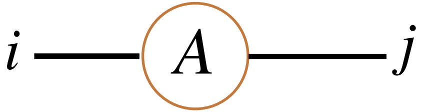
Itinerary

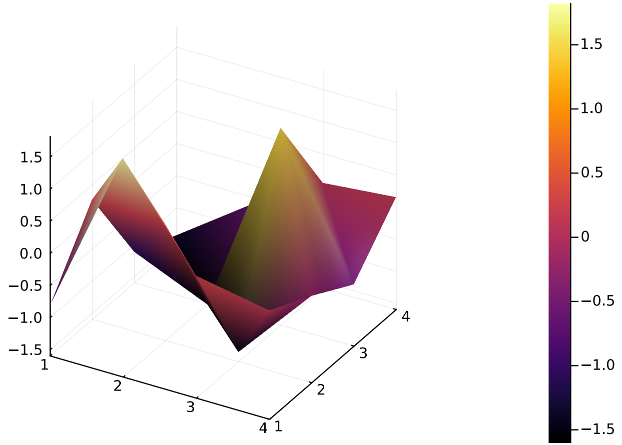
- Introduction of tensors
- Tensor correlation and sparsity
- Tensor decomposition, algorithms and optimization strategies
- Tensor networks and examples

What are tensors?

A tensor is a representation of a function of N variables

$$y = A(i, j)$$





We could therefore plot this function in 3D

Tensors and Contractions

Vector

A_i

Order-1

1 mode: i

i has dimension I

Matrix

A_{ij}

Order-2

modes: i, j

i has dimension I

j has dimension J

Higher-order tensor

A_{ijk}

Order-3

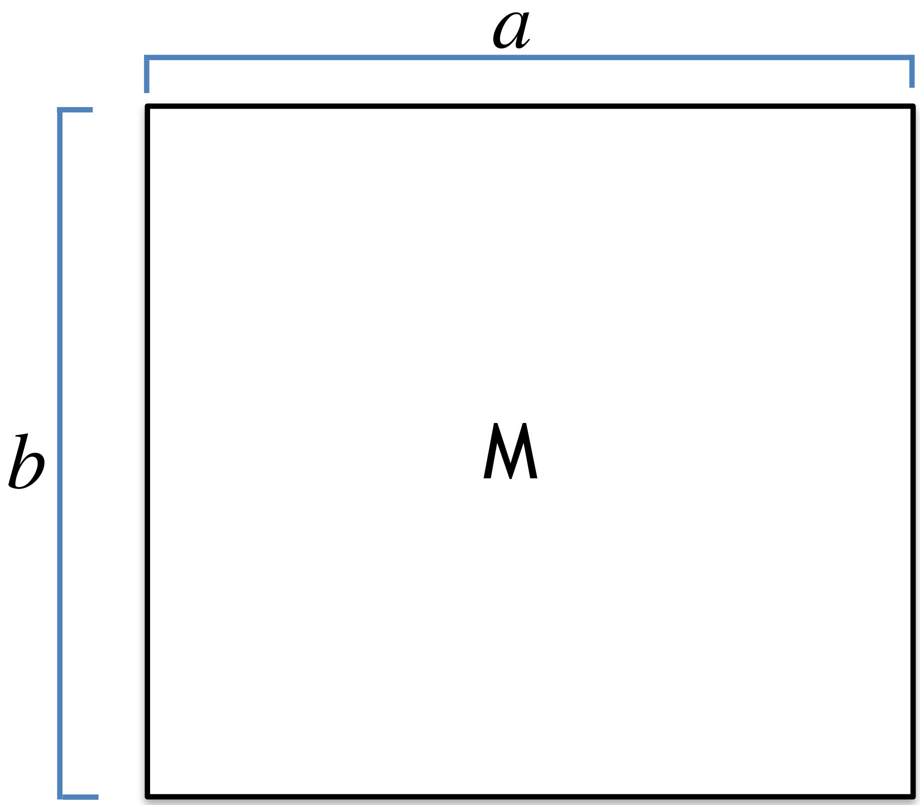
modes: i, j, k

i has dimension I

j has dimension J

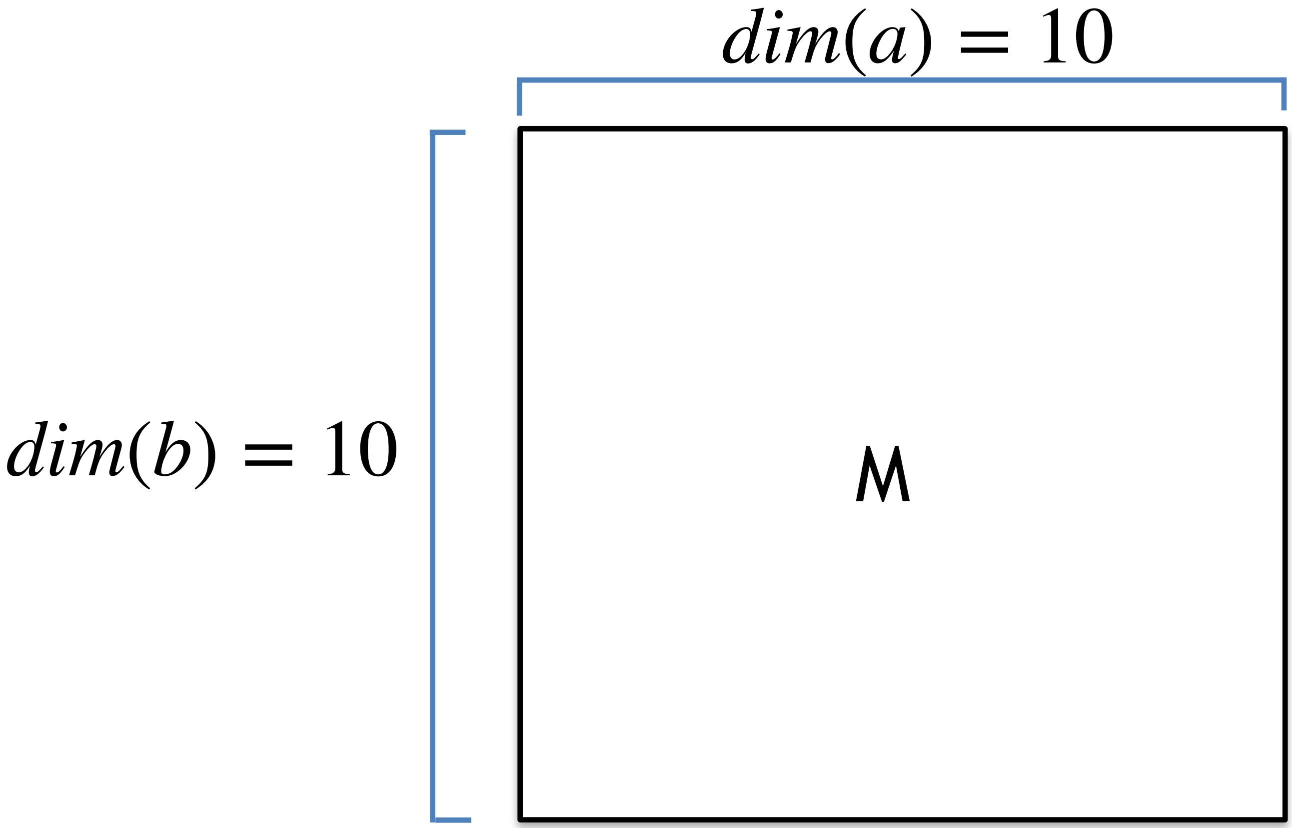
k has dimension K

Tensor indices can store metadata



FLATIRON INSTITUTE
Center for Computational Quantum Physics

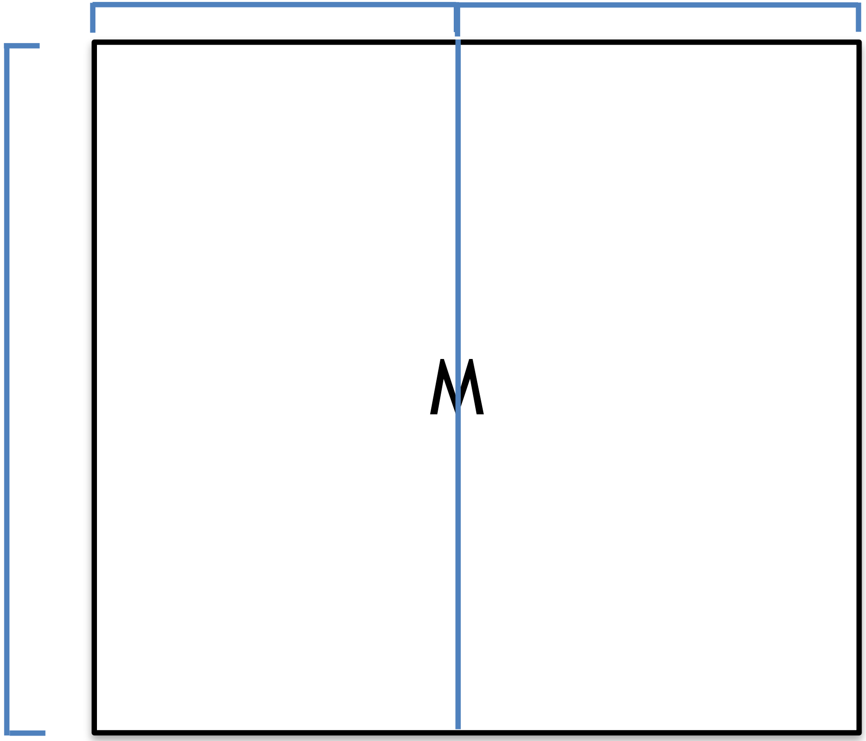
Tensor indices can store metadata



Tensor indices can store metadata

$$dim(a) = [5,5]$$

$$dim(b) = [10,]$$

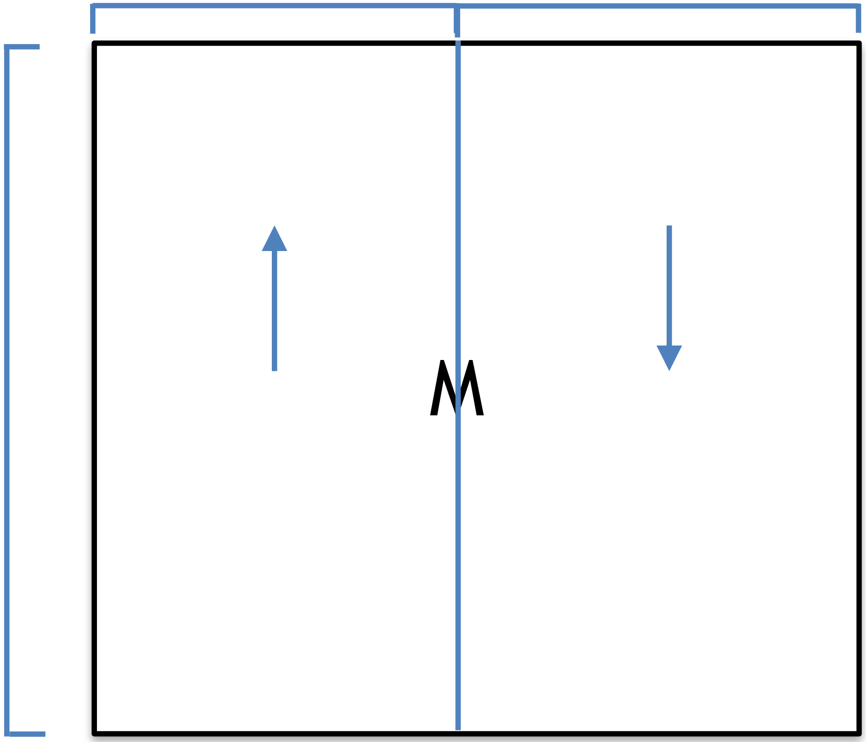


The tensor M has two blocks

Tensor indices can store metadata

$$dim(a) = [(5, \uparrow), (5, \downarrow)]$$

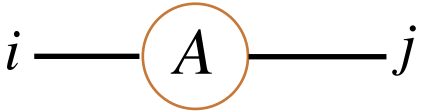
$$dim(b) = [10,]$$



Tensor Correlation

The canonical form of a tensor defines instantaneous interaction between orthogonal directions

$$y = A(i, j)$$



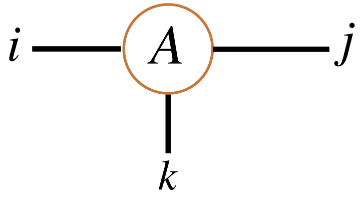
To modify the i direction we must loop over all other variables in the tensor

```
for i in 1:N
    for j in 1:N
        A(i,j) = L(i) * A(i,j)
    end
end
```

Tensor Correlation

The canonical form of a tensor defines instantaneous interaction between orthogonal directions

$$y = A(i, j, k)$$



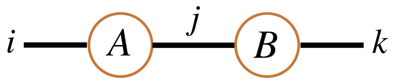
Adding more modes to the tensor increases the number of for loops

```
for i in 1:N
    for j in 1:N
      for k in 1:N
        A(i,j,k) = L(i) * A(i,j,k)
    end
end
```

Curse of Dimensionality

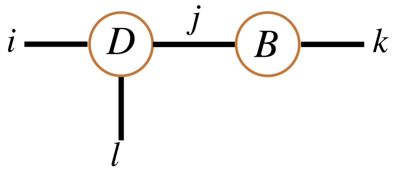
The cost of storing and accessing elements of a tensor grows exponentially with the number of dimensions in the tensor

$$C_{ik} = \sum_j A_{ij} B_{jk}$$



$$\mathcal{O}(N^3)$$

$$F_{ikl} = \sum_j D_{ijl} B_{jk}$$



$$\mathcal{O}(N^4)$$

Breaking the Curse: Do we really need all of this storage?

Physical data often has structure which leads to sparsity.
There are two forms of sparsity

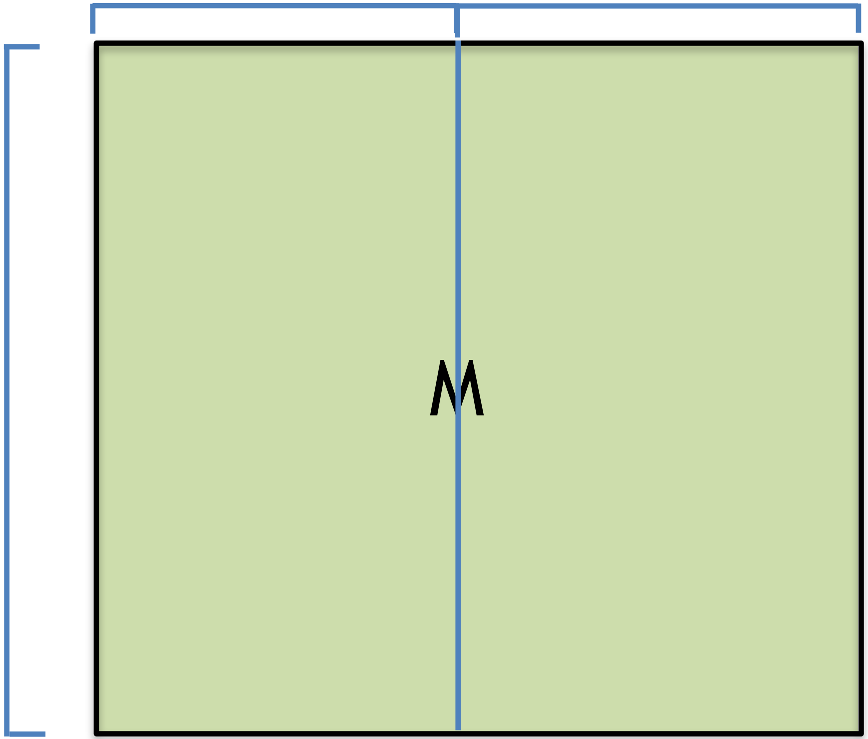
Element-wise

FLATIRON INSTITUTE
Center for Computational Quantum Physics

Tensor data can have structure

$$dim(a) = [5,5]$$

$$dim(b) = [10,]$$

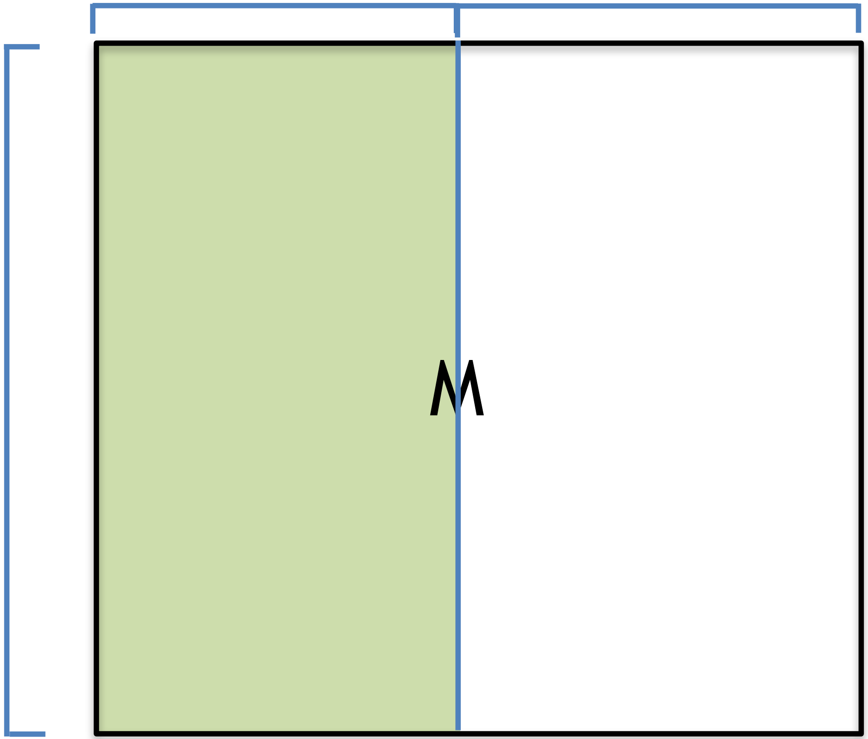


M is block-wise dense

Tensor data can have structure

$$dim(a) = [5{,}5]$$

$$dim(b) = [10{,}]$$

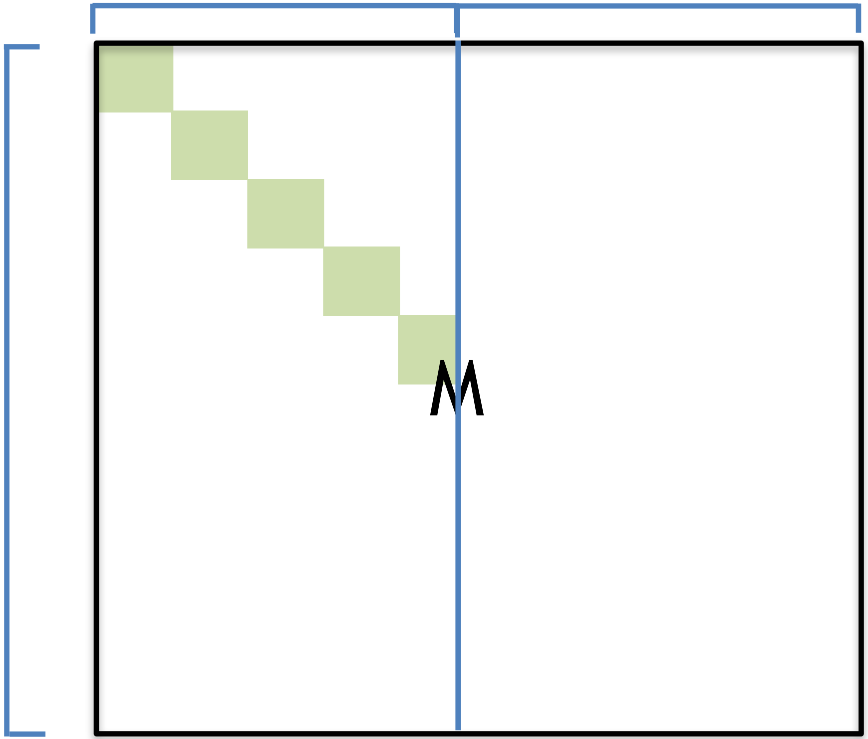


M is block-wise sparse

Tensor data can have structure

$$dim(a) = [5,5]$$

$$dim(b) = [10,]$$



M is block-wise and element-wise sparse

Do we really need all of this storage?

Physical data often has structure which leads to sparsity.
There are two forms of sparsity

Rank-Wise

Flatiron Institute
Center for Computational Quantum Physics

Tensor compression techniques

Matrix Decomposition

Singular Value Decomposition

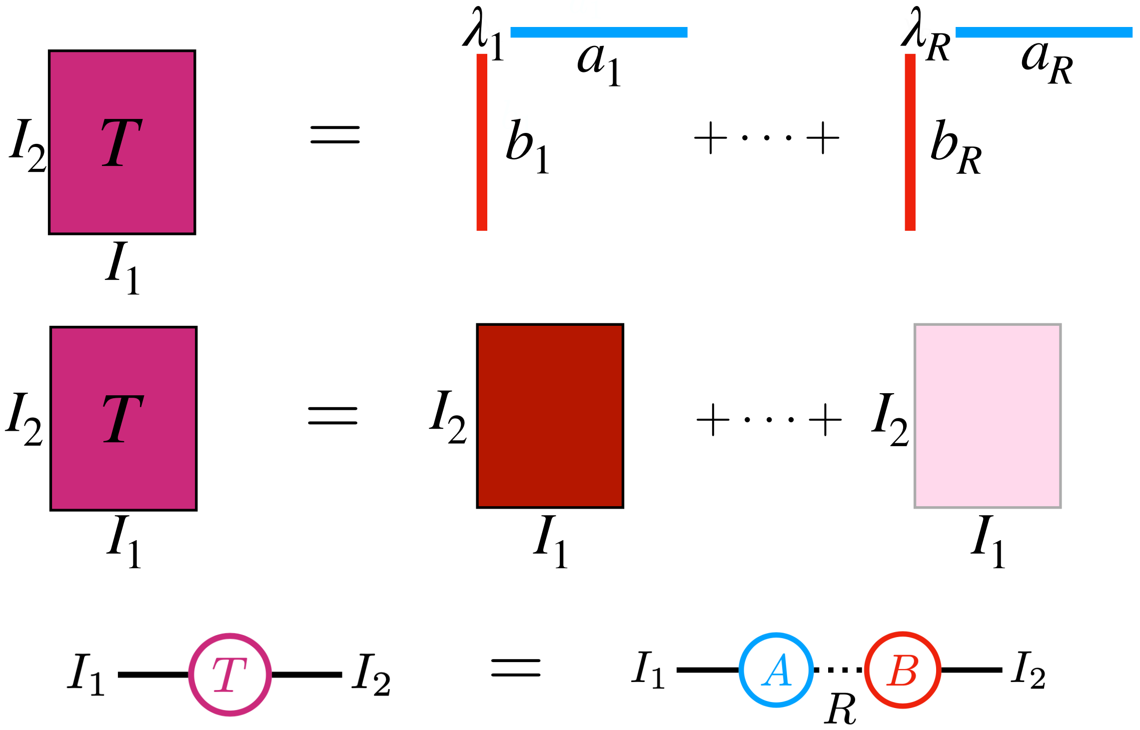
Cholesky Decomposition

Tensor Decomposition

Canonical Polyadic Decomposition

Tucker Decomposition

Singular value decomposition (SVD)



Properties of the SVD

$$A_{i,j} = \sum_{r,r'} U_{i,r} \Sigma_{r,r'} V_{r'j}$$

$$\sum_{j} V_{rj} V_{j,r'} = \mathbb{1}_{r,r'} \qquad \sum_{j} U_{r,i} U_{i,r'} = \mathbb{1}_{r,r'}$$

$$A^{\dagger}_{i,j} = U_{i,r} \frac{1}{\Sigma_{r,r'}} V_{r'j}$$

Properties of the SVD

$$A_{i,j} = \sum_{r,r'} U_{i,r} \Sigma_{r,r'} V_{r'j}$$

$$\sum_{j} A_{i,j} (A_{i',j})^T = U_{i,r} (\sum_{r',l'} \Sigma_{r,r'} (\sum_{j} V_{r'j} V_{l'j}) \Sigma_{l',l}) U_{i',l}$$

$$\sum_{j} A_{i,j} (A_{i',j})^T = \sum_{r,,l,} U_{i,r} \Sigma^2_{r,rl} U_{i',l}$$

U is the eigenvalue of the $AA*$

Properties of the SVD

$$A_{i,j} = \sum_{r,r'} U_{i,r} \Sigma_{r,r'} V_{r'j}$$

The singular vectors in U and V are unique and best in the L_2 sense

Ekart-Young theorem

$$\tilde{A}_{i,j} \approx \sum_{k,k'} U_{i,r} \Sigma_{k,k'} V_{k'j}$$

This is determined using the the k largest singular values.

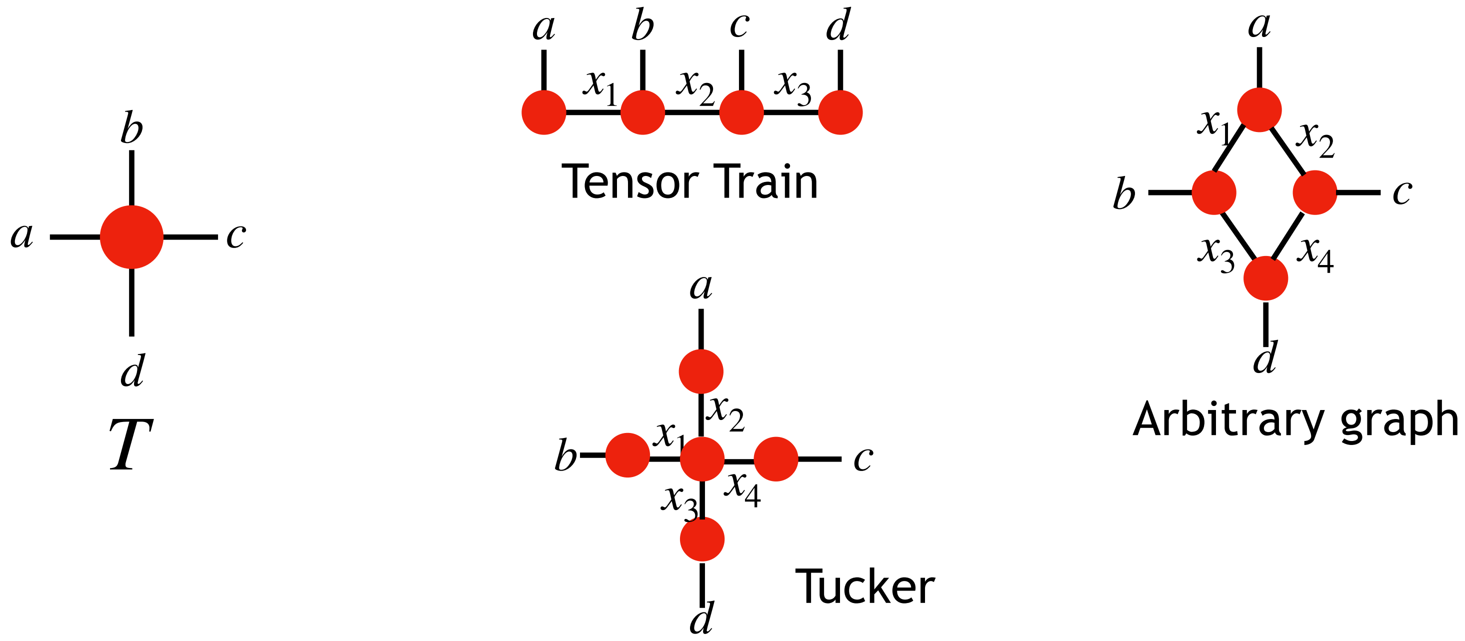
Error in $\tilde{A}_{i,j} \propto \sigma_{k+1}$

Other matrix decompositions

Eigenvalue	$A = Q\Lambda Q^{-1}$	Determines the roots of a linear system. Only works for square matrices
QR	$A = QR$	Works for rectangular matrices can be used to solve linear systems. Q is orthogonal and R is upper triangular.
LU	$A = LU$	Views matrix as Gaussian elimination problem. Used in solve linear problems. Can be faster than QR
Cholesky Decomposition	$A = LL*$	Positive definite matrices, Used in numerically efficient inversion algorithms. Faster than the LU

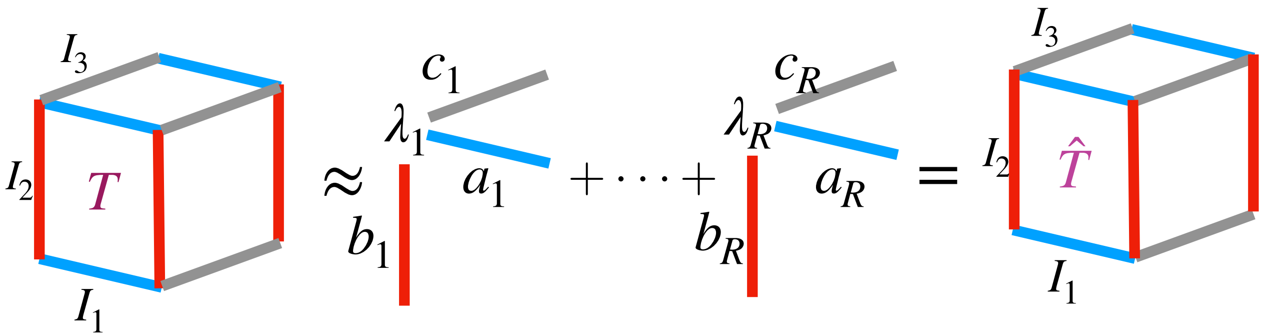
Hands-On Break

Tensor Decompositions

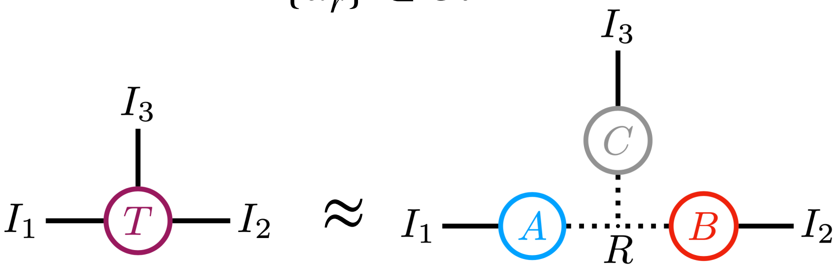


We choose a topology that represents the true correlation of the tensor

Canonical Polyadic Decomposition (CPD)



$$A = \{a_r\} \in \mathscr{R}^{I_1 \times R}$$



$$\mathcal{O}(I^3) \rightarrow \mathcal{O}(3IR)$$

Harshman, R. UCLA Work. Pap. Phonetics 1970, 16 (10), 1–84.
Carroll, J. D.; Chang, J. J. Psychometrika 1970, 35 (3), 283–319.

Issues with the CPD

There is no finite algorithm to determine the CP rank

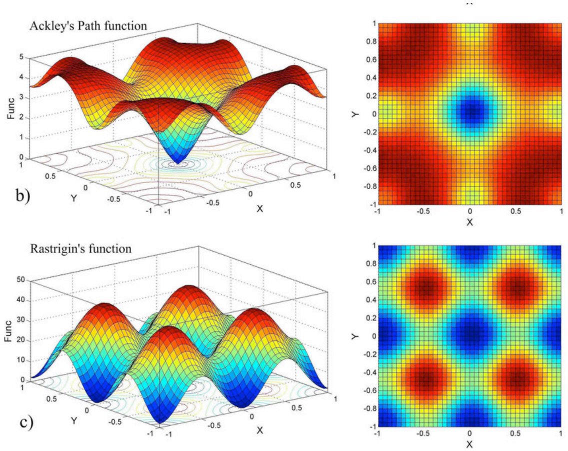
Optimization is an issue

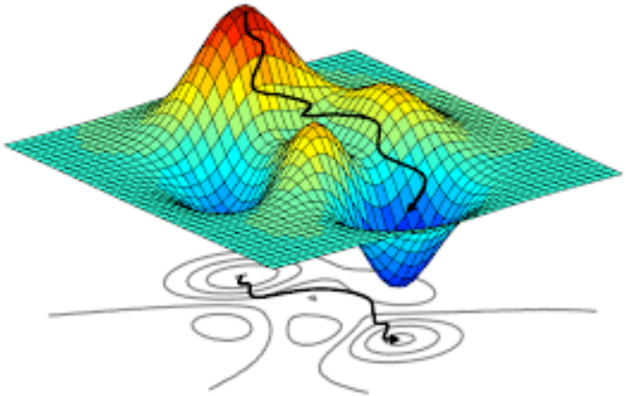
Best rank k decomposition cannot be determined from rank r approximation

Uniqueness of solutions is likely but unknown for tensors greater than order-3

Optimizing Tensor Decompositions

Multidimensional optimizations are difficult!





Finding a Global extrema is not possible (in general)

Optimizing Tensor Decompositions

$$f(x, y, z) \qquad \begin{pmatrix} \frac{\partial f(x,y,z)}{\partial x} \\ \frac{\partial f(x,y,z)}{\partial y} \\ \frac{\partial f(x,y,z)}{\partial z} \end{pmatrix} \qquad \begin{pmatrix} \frac{\partial^2 f(x,y,z)}{\partial x \partial x} & \frac{\partial^2 f(x,y,z)}{\partial x \partial y} & \frac{\partial^2 f(x,y,z)}{\partial x \partial z} \\ \frac{\partial^2 f(x,y,z)}{\partial y \partial x} & \frac{\partial^2 f(x,y,z)}{\partial y \partial y} & \frac{\partial^2 f(x,y,z)}{\partial y \partial z} \\ \frac{\partial^2 f(x,y,z)}{\partial z \partial x} & \frac{\partial^2 f(x,y,z)}{\partial z \partial y} & \frac{\partial^2 f(x,y,z)}{\partial z \partial z} \end{pmatrix}$$

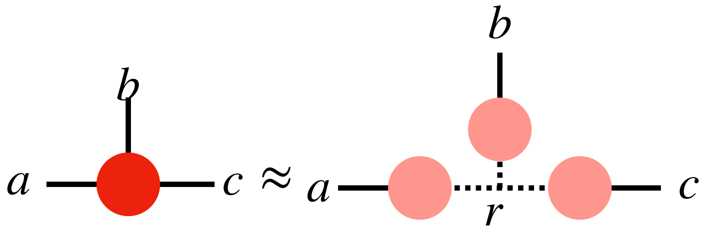
Computing the gradient and Hessian are time consuming and memory intensive

The derivative of a:

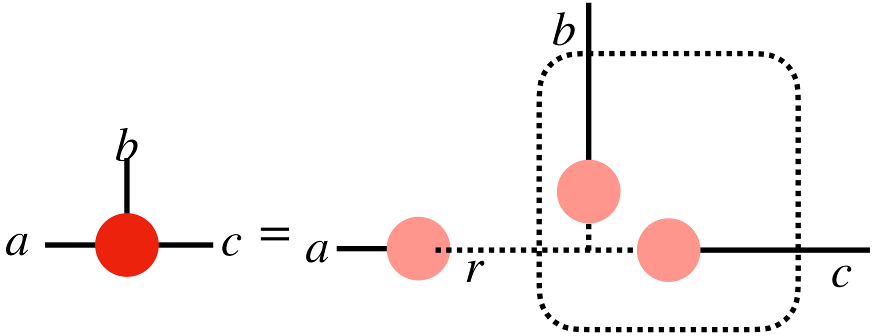
- tensor is a tensor
- Tensor network is a tensor network (often of higher order)

Algorithms to compute the CPD

$$T_{abc} \approx \sum_r A_{ar} B_{br} C_{cr}$$



Algorithms to compute the CPD



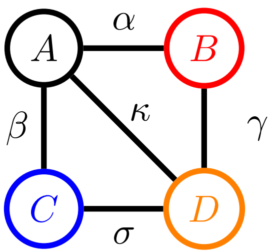
$$T_{abc} = A^*_{ar}[B \odot C]_{r,bc} \qquad T_{abc}[B \odot C]^{-1}_{r,bc} = A^*_{ar}$$

The Khatri-Rao Product (KRP)

ITensors makes Complex operations made simple

```
julia> using ITensors
julia> α = Index(2)   # id = 203
julia> β = Index(2)   # id = 38
julia> γ = Index(2)   # id = 724
julia> σ = Index(2)   # id = 160
julia> κ = Index(2)   # id = 74

julia> A = ITensor(α, β, κ)
julia> B = ITensor(α, γ)
julia> C = ITensor(β, σ)
julia> D = ITensor(γ, σ, κ)
```

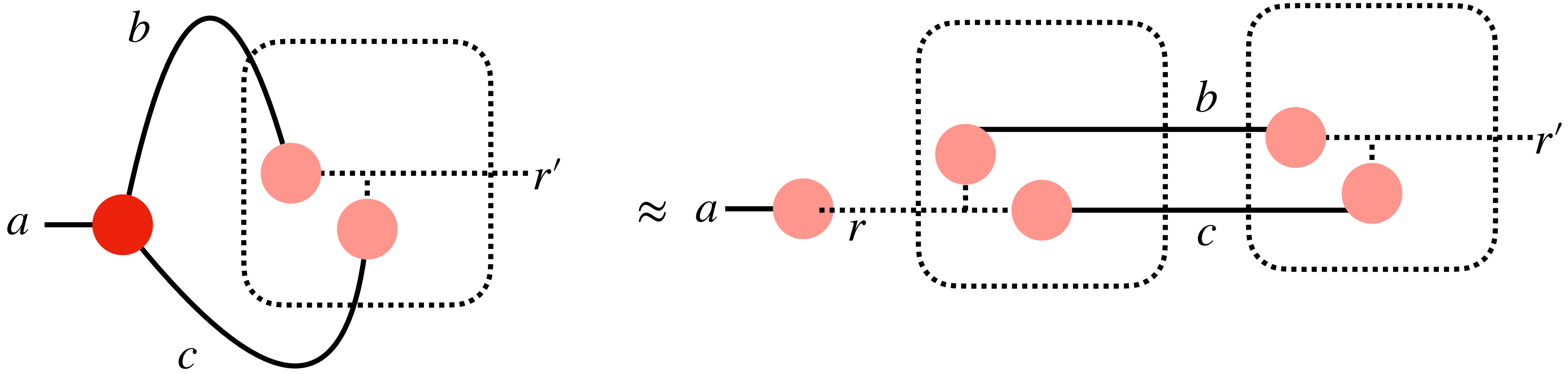
$$f(A,B,C,D) =$$




```
julia> f(A,B,C,D) = (A * B * C * D)[]
```

Hands-On: Write a CPD algorithm

Algorithms to compute the CPD



$$T_{abc}[B \odot C]_{r',bc} = A^*_{ar}[BB \odot CC]_{r,r'}$$

Computing $T_{abc}[B \odot C]_{r',bc}$ scales exponentially with the order of T

We can use tensor decomposition to reduce the cost of tensor contractions

Given a string of tensors

$$[X, Y, Z, \ldots]$$

Compute the result of contracting the string

$$T = XYZ\ldots$$

Contracting Tensor Networks

Given a string of tensors

$$[X, Y, Z, \ldots]$$

Compute the result of contracting the string

$$\color{red}{T} = XYZ\ldots$$

There are a number of potential issues:

- The resulting tensor requires large amount of storage

Contracting Tensor Networks

Given a string of tensors

$$[X, Y, Z, \ldots]$$

Compute the result of contracting the string

$$T = XYZ\ldots$$

There are a number of potential issues:

- The resulting tensor requires large amount of storage
- Intermediates require large amounts of storage

Approximating Tensor Network components

Given the tensor contraction

$$T = XYZ\ldots$$

Approximate some of the tensors in the set

$$X \approx \tilde{X} \qquad\qquad Z \approx \tilde{Z}$$

These approximations can make computing T easier

$$T \approx \tilde{X}Y\tilde{Z}\ldots$$

Error in Approximated Tensor Networks

The error introduced by approximating parts of a network

$$T = \tilde{X} Y \tilde{Z} \ldots + \mathbf{\Delta T}$$

Can be written in terms of error of T's parts

$$X = \tilde{X} + \Delta X \qquad Z = \tilde{Z} + \Delta Z$$

And therefore $\mathbf{\Delta T}$ grows nonlinearly

$$\mathbf{\Delta T} = [(\Delta X) Y \tilde{Z} \ldots] + [\tilde{X} Y (\Delta Z) \ldots] + [(\Delta X) Y (\Delta Z) \ldots] + \ldots$$

Directly decomposing tensor networks

We attempt to approximate T directly

$$f(\hat{T}) = \min_{\tilde{T}} \frac{1}{2} \|T - \tilde{T}\|^2$$

We can replace T with a matrix-free representation

$$f(\hat{T}) = \min_{\tilde{T}} \frac{1}{2} \|XYZ\ldots - \tilde{T}\|^2$$

Error in the approximation of T is directly controllable

Routes to improve CPD optimizations

1

Approximate canonically expensive loss function

$$T \approx XY$$

$$f(\hat{T}; T) \approx f(\hat{T}; XY)$$

2

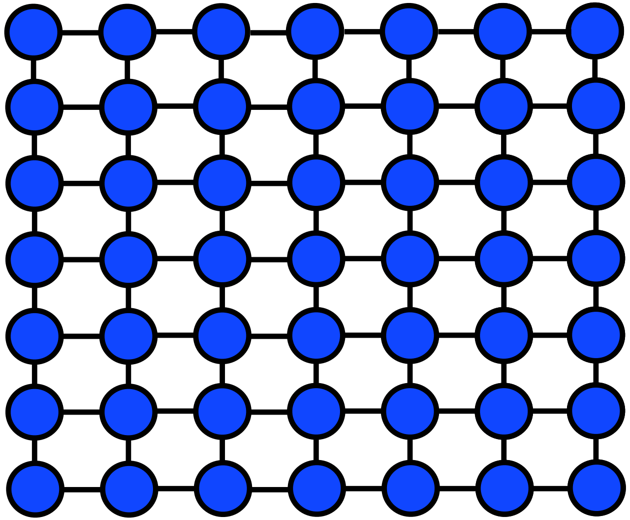
Approximate canonically expensive tensor network contractions

$$T = XY$$

$$f(\hat{T}; T) = f(\hat{T}; XY)$$

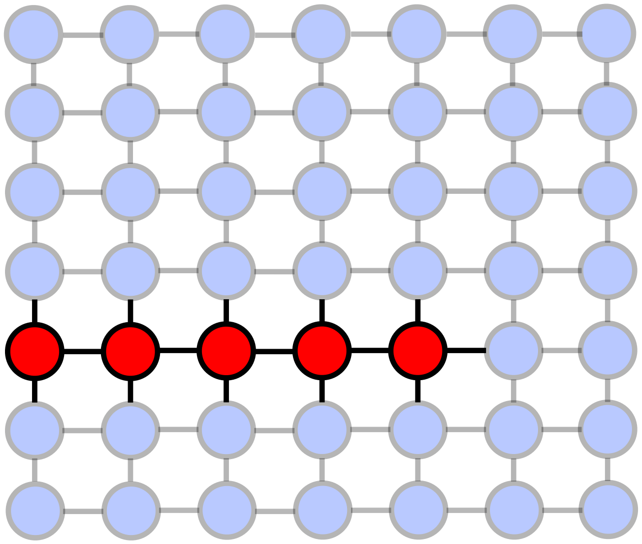
ITensorCPD.jl makes decomposing networks easy

```
### Set up A 7 x 7 Grid ####
nx=7
ny=7
grid =  named_grid((nx, ny))
indices = IndsNetwork(grid; link_space = 2)

### Fill the 7 x 7 grid with the Ising partition function ###
network = ising_network(Float64, indices, 0.4)
```



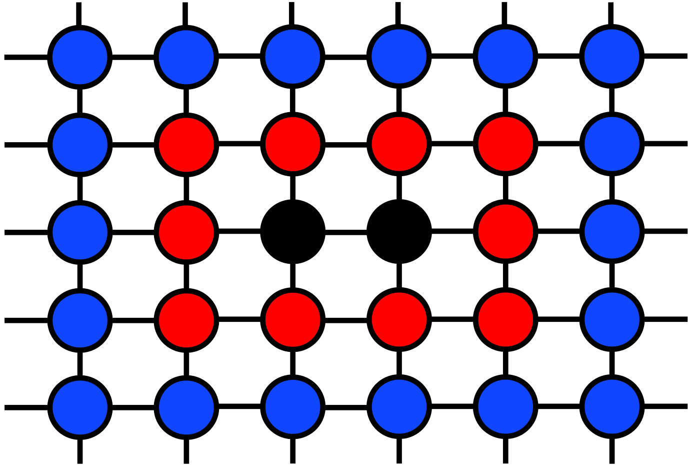
ITensorCPD.jl

```
### Grab a subgraph of the network ###
line_subtn = subgraph(network, ((5,1), (5,2), (5,3),
                                (5,4), (5,5)))

### Set up the CP problem ####
r = Index(20, "CP_rank")
n_linesubtn = norm_of_network(line_subtn)
check = FitCheck(1e-3, 100, n_linesubtn)

### Create a CP initial guess and optimize ###
initial_guess = random_CPD(line_subtn, r);
cp_opt = als_optimize(line_subtn, initial_guess;
                          check);
```



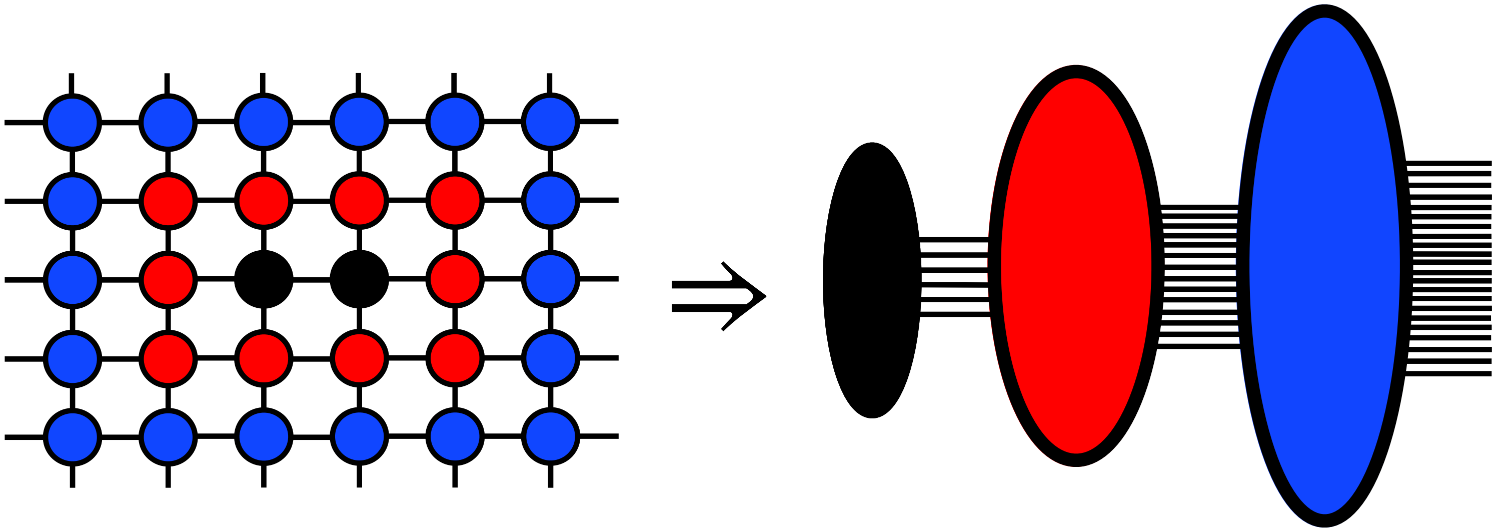
Physics example

CP approximation of Lattice networks



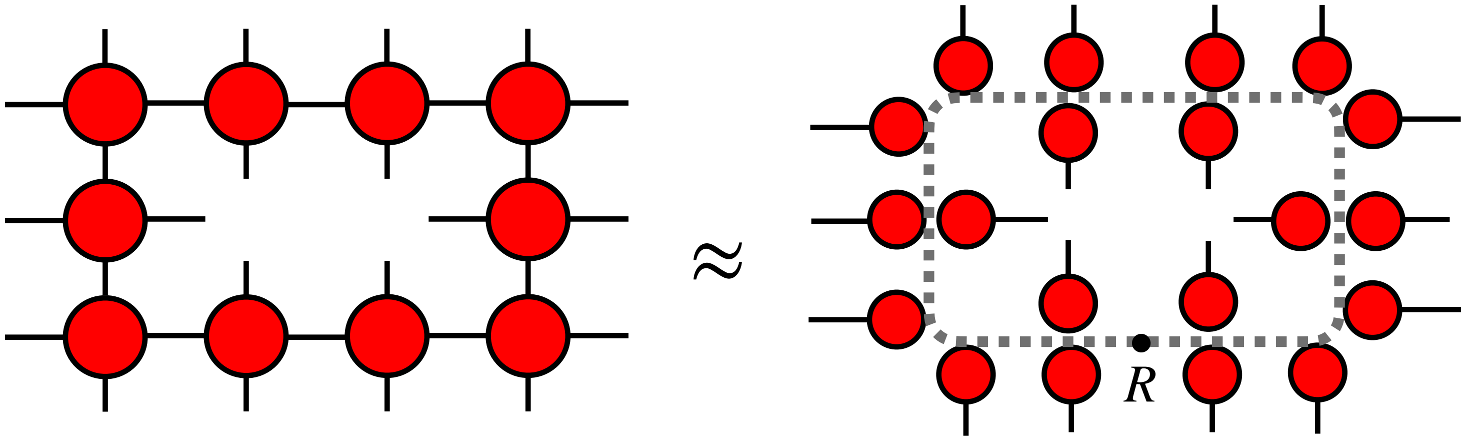
Two-dimensional tensor-networks can represent Ising partition functions, quantum circuits and beyond

CP approximation of Lattice networks

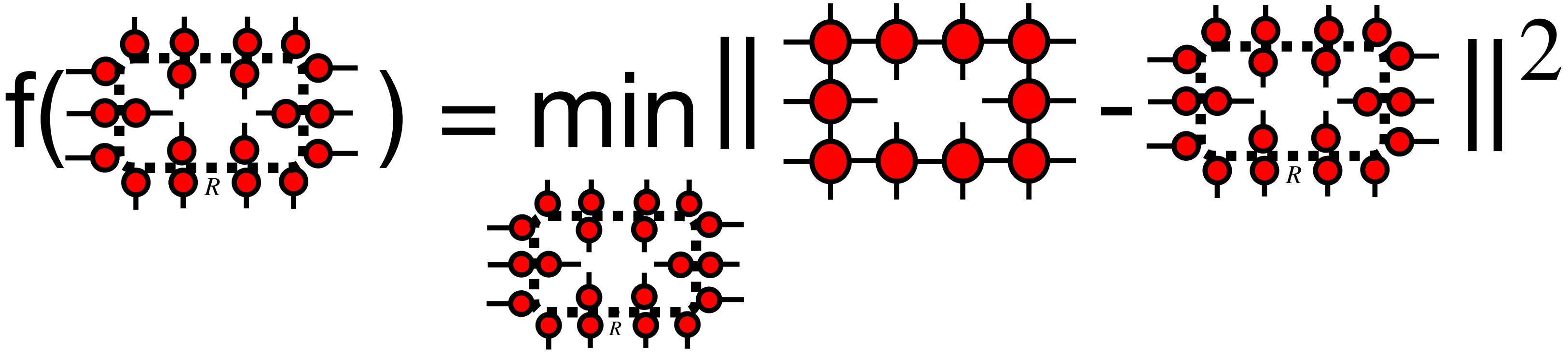


Contracting these models exactly becomes practically impossible very quickly

CP approximation of Lattice networks

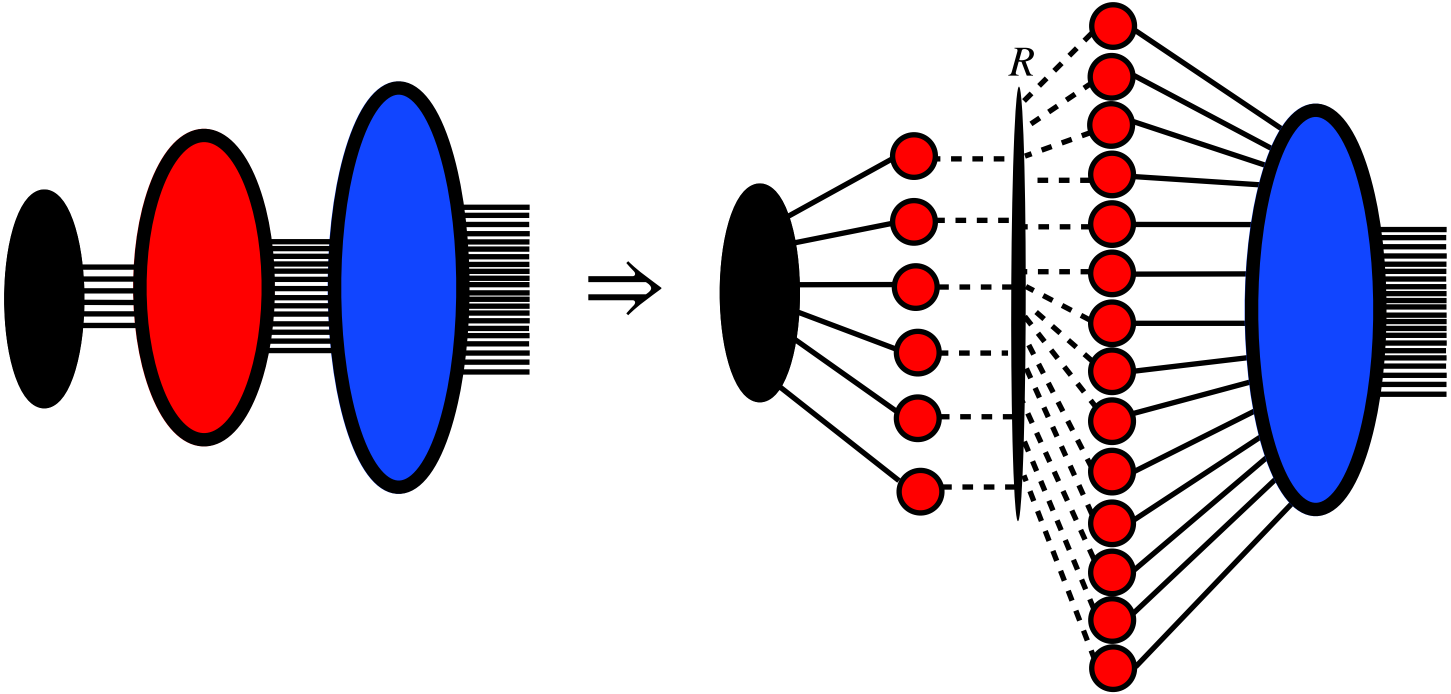


CP approximation of Lattice networks



Pierce, K; Tindall, J; Stoudenmire, EM; preprint 2025

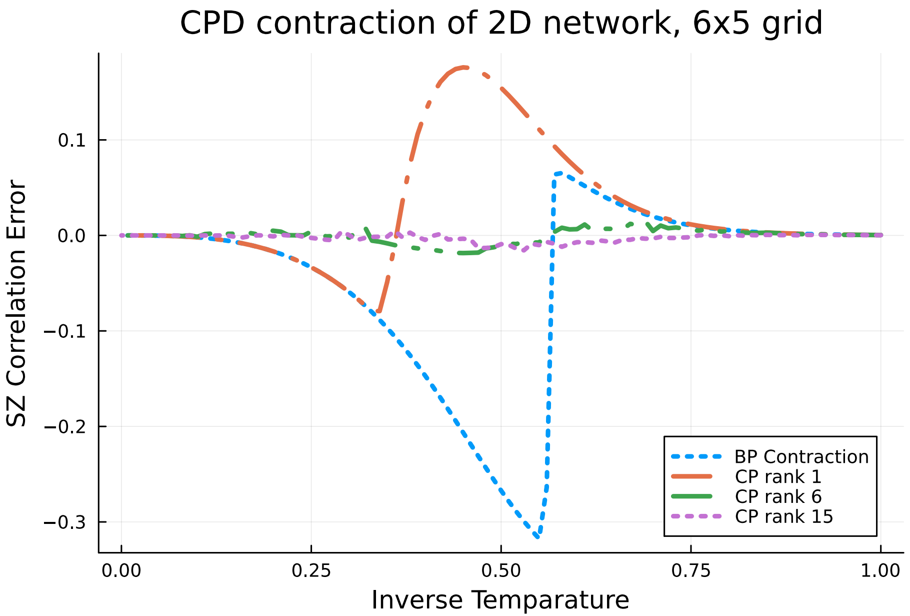
CP approximation of Lattice networks

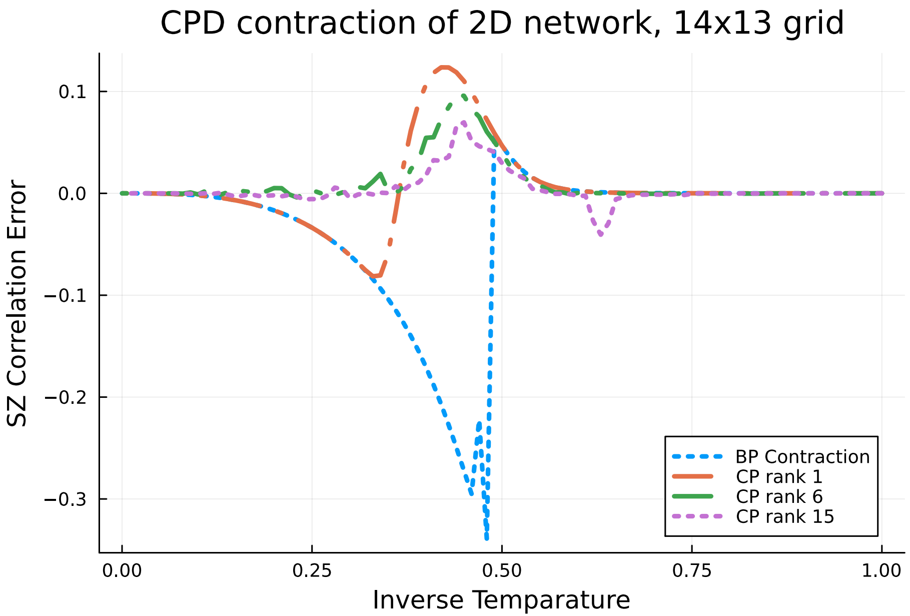


Decompose each matrix node simultaneously and independently

Pierce, K; Tindall, J; Stoudenmire, EM; preprint 2025

CP approximation of Lattice networks





Pierce, K; Tindall, J; Stoudenmire, EM; preprint 2025

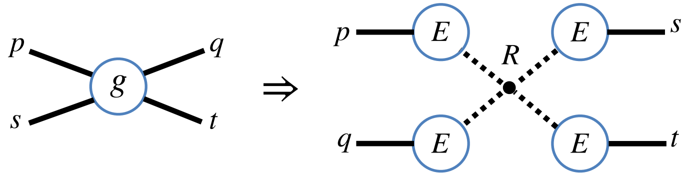
FLATIRON INSTITUTE

Center for Computational Quantum Physics

Approximately decompose the Coulomb integrals

Goal: decompose the two-particle (order-4) integral tensor

$$\hat{g}^{pq}_{st} = \sum_{r}^{R} P^p_r Q^q_r S^r_s T^r_t$$



$$f(\hat{g}) = \min_{\hat{g}} \frac{1}{2} \| g^{pq}_{st} - \sum_{r}^{R} P^p_r Q^q_r S^r_s T^r_t \|^2$$

Prohibitively large!!

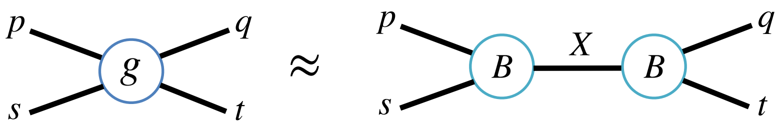
Storage: $\mathcal{O}(N^4)$

Scaling: $\mathcal{O}(N^5)$

Approximately decompose the Coulomb integrals

$$f(\hat{g}) = \min_{\hat{g}} \frac{1}{2} \| \boxed{g^{pq}_{st}} - \sum_{r}^{R} P^p_r Q^q_r S^r_s T^r_t \|^2$$

The tensor G can be quickly decomposed via the DF approximation



$$f(\hat{g}) \approx \min_{\hat{g}} \frac{1}{2} \| \sum_{X} B^{pX}_s B^{qX}_t - \sum_{r}^{R} P^p_r Q^q_r S^r_s T^r_t \|^2$$

Storage: $\mathcal{O}(N^3)$ Scaling: $\mathcal{O}(N^4)$