Title: LECTURE: Tensor networks and quantum simulations

Speakers: Martin Ganahl

Collection: Quantum and AI Career Trajectories Mini-Course: Computational Methods and their Applications

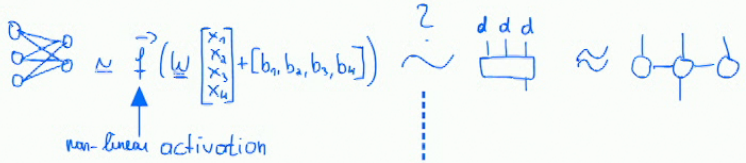
Date: May 12, 2023 - 9:30 AM

URL: https://pirsa.org/23050102

Abstract: TBC

Machine learning, data science

Novel research direction. Question: can TN be useful for ML tasks?
Motivation: can we map a NN to a TN?

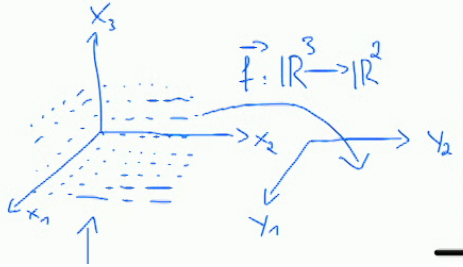


non linear map from $\mathbb{R}^n \to \mathbb{R}^m$

non linearities essential for universality

assumptions: bounded domain of $\vec{f}$:

$x_i \in [x_i^{min}, x_i^{max}]$



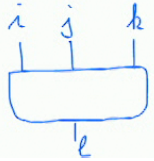
linear map from $\mathbb{R}^{d^n} \to \mathbb{R}^m$ (high-dimensional)

a sufficiently well behaved non-linear map $\mathbb{R}^n \to \mathbb{R}^m$ can in principle be approximated by a linear map $\mathbb{R}^{d^n} \to \mathbb{R}^m$:

encode each meshpoint $\vec{x}_r$ into a tuple of integers $(i^{(r)}, j^{(r)}, k^{(r)}) \in \mathbb{N}_+^3$

define

$T_{i^{(r)} j^{(r)} k^{(r)} \ell} = f_\ell(\vec{x}_r) \equiv$



TN for machine learning, high dimensional integration & PDEs

ML: - supervised learning
- generative modelling

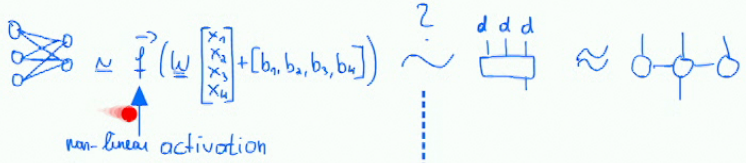
TT-cross approximation

Function encoding & PDE

Machine learning, data science

Novel research direction. Question: can TN be useful for ML tasks?

Motivation: can we map a NN to a TN?

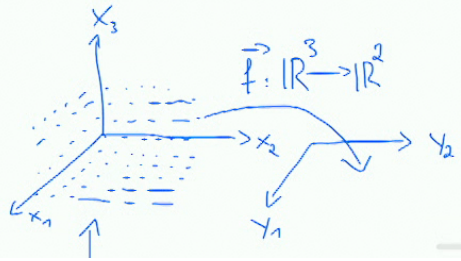


non linear map from $\mathbb{R}^n \to \mathbb{R}^m$

non linearities essential for universality

assumptions: bounded domain of $\vec{f}$:

$x_i \in [x_i^{min}, x_i^{max}]$



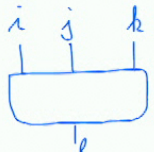
linear map from $\mathbb{R}^{d^n} \to \mathbb{R}^m$ (high-dimensional)

a sufficiently well behaved non-linear map $\mathbb{R}^n \to \mathbb{R}^m$ can in principle be approximated by a linear map $\mathbb{R}^{d^n} \to \mathbb{R}^m$:

encode each meshpoint $\vec{x}_r$ into a tuple of integers $(i^{(r)}, j^{(r)}, k^{(r)}) \in \mathbb{N}_+^3$

define

$T_{i^{(r)} j^{(r)} k^{(r)} e} = f_e(\vec{x}_r) \hat{=}$



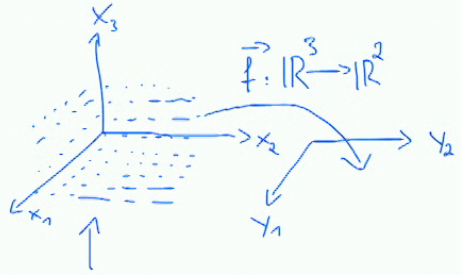
(f_1, f_2, f_3) non-linear activation bias

non linear map from $\mathbb{R}^n \to \mathbb{R}^m$

non linearities essential for universality

assumptions: bounded domain of $\vec{f}$:

$x_i \in [x_i^{min}, x_i^{max}]$



$d \times d \times d$ meshpoints

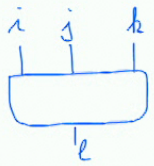
linear map from $\mathbb{R}^d \to \mathbb{R}^m$ (high-dimensional)

a sufficiently well behaved non-linear map $\mathbb{R}^n \to \mathbb{R}^m$ can in principle be approximated by a linear map $\mathbb{R}^{d^n} \to \mathbb{R}^m$:

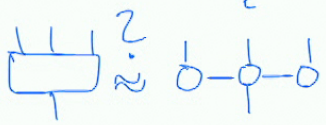
encode each meshpoint $\vec{x}_r$ into a tuple of integers $(i^{(r)}, j^{(r)}, k^{(r)}) \in \mathbb{N}_+^3$

define

$T_{i^{(r)} j^{(r)} k^{(r)} \ell} = f_\ell(\vec{x}_r) \in$



decompose $\approx$



09:49 Freitag 12. Mai

92 %

sent image as a vector in Hilbert space

al strategy: embed image using "feature maps"

$[0,127] \longrightarrow |\phi(x_i)\rangle =$ integer encoding

$$\begin{cases} [1, x_i, x_i^2, \dots, x_i^p] = |0\rangle + x_i|1\rangle + \dots + x_i^p|p\rangle & \text{polynomial} \\ \cos\left(\frac{\pi}{2}x_i\right)|0\rangle + \sin\left(\frac{\pi}{2}x_i\right)|1\rangle & \text{trigonometric} \\ \vdots \\ \text{(other choices possible)} \quad \text{(Novikov et al.)} \end{cases}$$

xel

h

lute

$$x_1 x_2 \cdots x_{784} \Rightarrow |\phi(x_1)\rangle \otimes |\phi(x_2)\rangle \otimes \cdots \otimes |\phi(x_N)\rangle.$$

image is mapped to a product state!

linear classifier in tensor network format.

$\vec{x}$ → ℓ → $\begin{bmatrix} z_0 \\ z_1 \end{bmatrix}$ ← one "score" for each digit

75 / 94

09:50 Freitag 12. Mai

92 %

sent image as a vector in Hilbert space

al strategy: embed image using "feature maps"

$[0,127] \longrightarrow |\phi(x_i)\rangle = \begin{cases} [1, x_i, x_i^2, \dots, x_i^p] = |0\rangle + x_i|1\rangle + \dots + x_i^p|p\rangle & \text{polynomial} \\ \cos\left(\frac{\pi}{2}x_i\right)|0\rangle + \sin\left(\frac{\pi}{2}x_i\right)|1\rangle & \text{trigonometric} \\ \vdots \\ \text{(other choices possible)} & \text{(Novikov et al.)} \end{cases}$

integer encoding

xel

h

lute

$x_1 x_2 \cdots x_{784} \Rightarrow |\phi(x_1)\rangle \otimes |\phi(x_2)\rangle \otimes \cdots \otimes |\phi(x_N)\rangle$

image is mapped to a product state!

linear classifier in tensor network format.

$\vec{x} \rightarrow$ [ℓ] $\rightarrow \begin{bmatrix} z_0 \\ z_1 \end{bmatrix}$ ← one "score" for each digit

75 / 94

takes a value between 0,1
black, white

$x_i \in [0,1]$

How can we use an MPS here?

1) represent image as a vector in Hilbert space

General strategy: embed image using "feature maps"

$x_i \in [0,1] \longrightarrow |\phi(x_i)\rangle =$

- $[1, x_i, x_i^2, \dots, x_i^p] = |0\rangle + x_i|1\rangle + \dots + x_i^p|p\rangle$ polynomial
- $\cos(\frac{\pi}{2}x_i)|0\rangle + \sin(\frac{\pi}{2}x_i)|1\rangle$ trigonometric
- ⋮
- (other choices possible) (Novikov et al.)

x_i ∈ [0,1] — integer encoding

i-th pixel

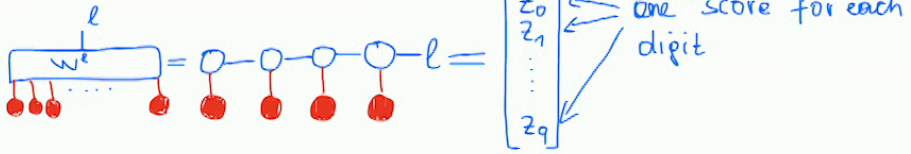
$x_i = 0$ = black

$x_i = 127$ is white

image $x_1 x_2 \cdots x_{784} \Rightarrow |\phi(x_1)\rangle \otimes |\phi(x_2)\rangle \otimes \cdots \otimes |\phi(x_N)\rangle$

each image is mapped to a product state!

2) Use a linear classifier in tensor network format.

$W_\ell \vec{\phi}(\vec{x}) =$ 

- integer encoding
th pixel

$x_i = 0 \to 1$

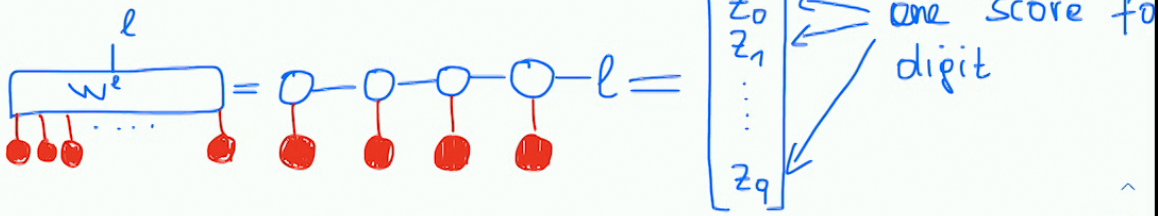
(other choices possible) (Novikov

image $x_1 x_2 \cdots x_{784} \Rightarrow |\phi(x_1)\rangle \otimes |\phi(x_2)\rangle \otimes \cdots \otimes |\phi(x_N)\rangle$.

pixel 1 ... pixel 75

each image is mapped to a product state!

use a linear classifier in tensor network format.

$W_\ell \vec{\phi}(\vec{x}) =$ 

classification of an image $x_1 \cdots x_N$: $\underset{\ell}{\text{argmax}} \left[W^\ell \vec{\phi}(\vec{x}) \right]$

- integer encoding

th pixel

$x_i = 0 \to 1$

(other choices possible) (Novikov

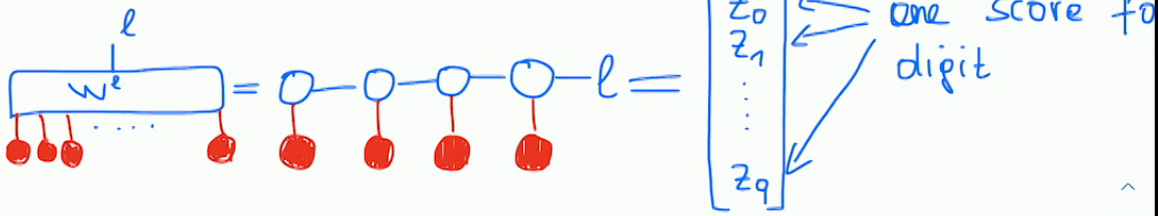
$x_i \in [0,1]$

image $x_1 x_2 \cdots x_{784} \Rightarrow |\phi(x_1)\rangle \otimes |\phi(x_2)\rangle \otimes \cdots \otimes |\phi(x_N)\rangle$

pixel 1 ... pixel 784

each image is mapped to a product state!

use a linear classifier in tensor network format.

$W_\ell \vec{\phi}(\vec{x}) =$ 

classification of an image $x_1 \cdots x_N$: $\underset{\ell}{\text{argmax}} \left[W^\ell \vec{\phi}(\vec{x}) \right]$

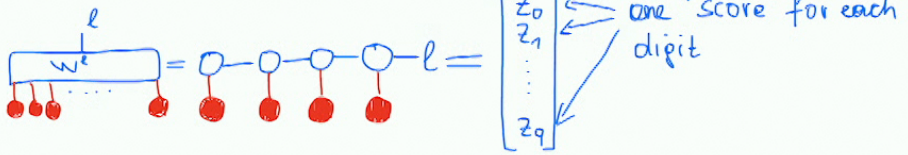
image $x_1 x_2 \cdots x_{784} \Rightarrow |\phi(x_1)\rangle$

pixel 1, pixel 784

implicitly assuming ordering

each image is mapped to a product state!

2) use a linear classifier in tensor network format.

$W_\ell \vec{\phi}(\vec{x}) =$ 

classification of an image $x_1 \cdots x_N$: $\underset{\ell}{\text{argmax}} \left[W^\ell \vec{\phi}(\vec{x}) \right]$

3) optimize the MPS parameters on the training set

possible cost functions:

$$C = \sum_i^{N_T} \| W^\ell \vec{\phi}(\vec{x}_i) - \vec{y}_i \|^2$$

image $x_1 x_2 \cdots x_{784} \Rightarrow |\phi(x_1)\rangle \otimes |\phi(x_2)\rangle \otimes \cdots \otimes |\phi(x_{784})\rangle$

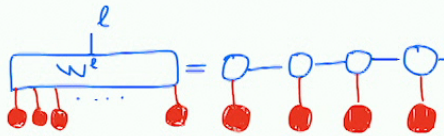
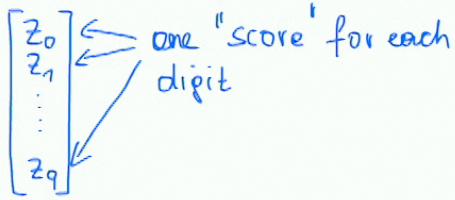
pixel 1, pixel 784

implicitly assuming ordering

each image is mapped to a product state!

2) use a linear classifier in tensor network format.

$|\phi(k_1)\rangle$ $|\phi(k_N)\rangle$

$W_\ell \vec{\phi}(\vec{x}) =$  $=$ $\ell =$ 

$\begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_9 \end{bmatrix}$ ⇐ one "score" for each digit

classification of an image $x_1 \cdots x_N$: $\underset{\ell}{\text{argmax}} \left[W^\ell \vec{\phi}(\vec{x}) \right]$

3) optimize the MPS parameters on the training set

possible cost functions:

image $x_1 x_2 \cdots x_{784} \Rightarrow |\phi(x_1)\rangle \otimes |\phi(x_2)\rangle \otimes \cdots \otimes |\phi(x_{784})\rangle$

pixel 1 … pixel 784

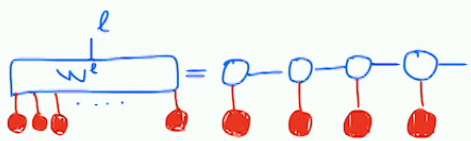
implicitly assuming ordering

each image is mapped to a product state!

2) use a linear classifier in tensor network format.

$|\phi(x_1)\rangle$ … $|\phi(x_N)\rangle$

$W_\ell \quad \vec{\phi}(\vec{x}) \in \mathbb{R}^{2^N} = |\phi(x_1)\cdots\phi(x_N)\rangle$



$= \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_9 \end{bmatrix}$ ← one "score" for each digit

classification of an image $x_1 \cdots x_N$: $\underset{\ell}{\text{argmax}} \left[W^\ell \vec{\phi}(\vec{x}) \right]$

3) optimize the MPS parameters on the training set

possible cost functions:

image $x_1 x_2 \cdots x_{784} \Rightarrow |\phi(x_1)\rangle \otimes \cdots$ (pixel 1 … pixel 784)

implicitly assuming ordering

each image is mapped to a product state!

2) use a linear classifier in tensor network format

$W_\ell \, \vec{\phi}(\vec{x}) = \ell$ (label for $\ell = 7$)

$\vec{\phi}(\vec{x}) \in \mathbb{R}^{2^n}$

$|\phi(x_1)\rangle \cdots |\phi(x_N)\rangle$

$|\phi(x_1)\rangle$ $|\phi(x_N)\rangle$

W^ℓ — 2^n — n

ℓ = output 10

image [7]

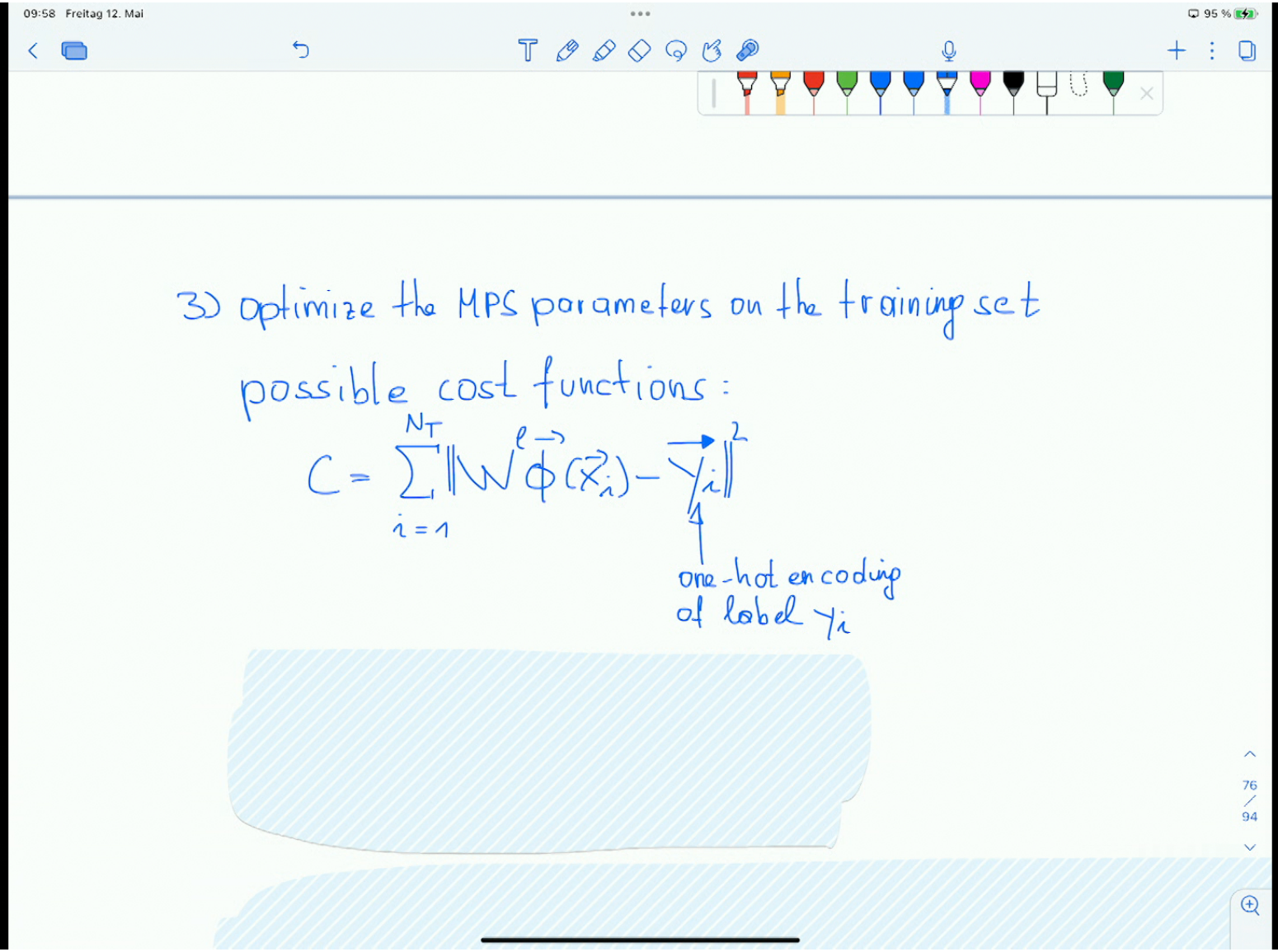
$\ell = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_9 \end{bmatrix}$ ← one "score" for each digit

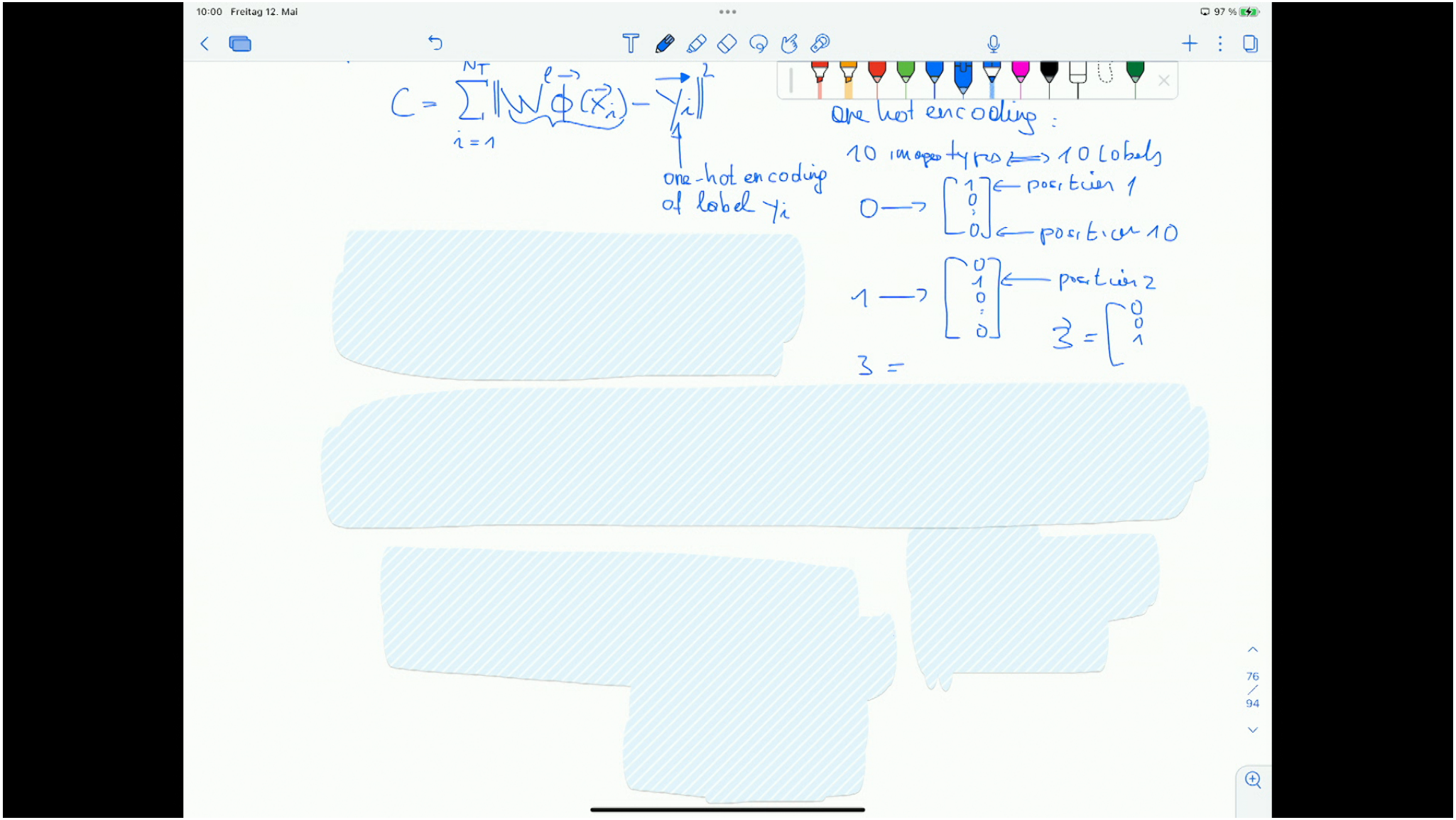
classification of an image $x_1 \cdots x_N$: $\underset{\ell}{\text{argmax}} \left[W^\ell \vec{\phi}(\vec{x}) \right]$

3) optimize the MPS parameters on the training set

possible cost functions:

$$C = \sum_{i=1}^{N_T} \left\| W^\ell \vec{\phi}(\vec{x}_i) - \vec{y}_i \right\|^2$$

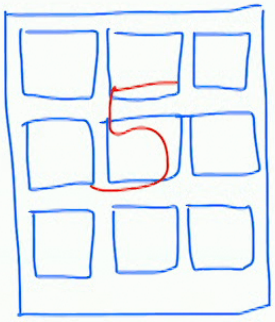

09:58 Freitag 12. Mai
95 %
3) optimize the MPS parameters on the training set
possible cost functions:
$C = \sum_{i=1}^{N_T} \| W^{\ell}\vec{\phi}(\vec{x}_i) - \vec{y}_i \|^2$
one-hot encoding of label y_i
76 / 94


10:00 Freitag 12. Mai
97 %
$C = \sum_{i=1}^{N_T} \| W \vec{\phi}(\vec{x}_i) - \vec{y}_i \|^2$
one-hot encoding of label y_i
one hot encoding:
10 image types ⟺ 10 Labels
0 → [1 0 ⋮ 0] ← position 1
position 10
1 → [0 1 0 ⋮ 0] ← position 2
3 = [0 0 1]
3 =
76 / 94

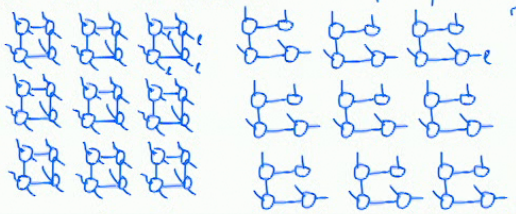
- optimization carried out based on gradient descent methods (ADAM, ...)
- gradients can be computed through autodiff or manually

Performance: Can reach ~ 98.5% test accuracy on MNIST
~ 87% test accuracy on Fashion-MNIST

extensions:



decompose image into plaquette product state instead of "snake".
Use PEPS or MPS on each plaquette



- more expressive but overfits more easily
- performance still below MPS method

76/94

Outlier detection

Motivation: detection of anomalous ~~data among~~ normal datapoints

Semi-supervised tensor network approach

Data: vectors in $\mathbb{R}^n$, where n is the number of features (eg. pixels of image)

Basic idea:

(1) embed datapoint $\vec{x}$ into high dimensional Hilbert space

$\vec{x} \to \vec{\phi}(\vec{x}) = |\phi(x_1)\rangle |\phi(x_2)\rangle \cdots |\phi(x_n)\rangle = |\phi(x_1) \cdots \phi(x_n)\rangle$

↑ feature map:

feature map maps all input onto hypersphere surface

(2) use rank deficient map to collapse anomalous instances

$P: \mathbb{R}^{2^n} \to \mathbb{R}^{2^m}$, $m < n$



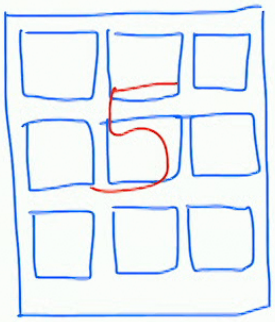
optimize P s.t. $\| P |\phi(x_1) \cdots \phi(x_n)\rangle \| = \begin{cases} \approx 1 & \text{for normal instances} \\ \approx 0 & \text{for anomalous instances} \end{cases}$

$\sum_e \| W^e \Phi^e(x_n) \|$

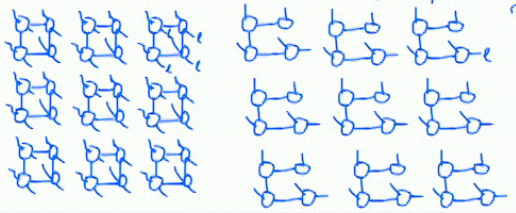
- optimization carried out based on gradient descent methods (ADAM, ...)
- gradients can be computed through autodiff or manually

Performance: Can reach ~ 98.5% test accuracy on MNIST
~ 87% test accuracy on Fashion-MNIST

extensions:



decompose image into plaquette product state instead of "snake".
Use PEPS or MPS on each plaquette



- more expressive but overfits more easily
- performance still below MPS method

76 / 94

Outlier detection

possible cost functions:

$$C = \sum_{i=1}^{N_T} \| W^{\ell} \vec{\phi}(\vec{x}_i) - \vec{y}_i \|^2$$

tensorflow — one-hot encoding of label y_i

N_T = # training images

one hot encoding:

10 image types $\Longleftrightarrow$ 10 Labels

$0 \longrightarrow \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ ← position 1, ← position 10

$1 \longrightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ ← position 2

$2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$

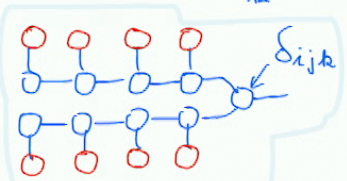
$$C = \sum_{i=1}^{N_T} \left\| \frac{W^{\ell} \vec{\phi}(\vec{x}_i)}{\sqrt{\| W^{\ell} \vec{\phi}(\vec{x}_i) \|}} - \vec{y}_i \right\|^2$$

$$C = -\sum_{i=1}^{N_T} \log \operatorname{softmax}\left[W^{\ell} \vec{\phi}(\vec{x}_i) \right], \quad \operatorname{softmax} \vec{y} = \frac{e^{\vec{y}}}{\sum_k e^{y_k}}$$

↑ log

$$C = -\sum_i \sum_{\ell} \delta_{\ell y_i} \log p(y_i = \ell)$$

$$\frac{\| W^{\ell} \vec{\phi}(\vec{x}_i) \|^2}{\sum_{\ell} \| W^{\ell} \vec{\phi}(\vec{x}_i) \|^2}$$

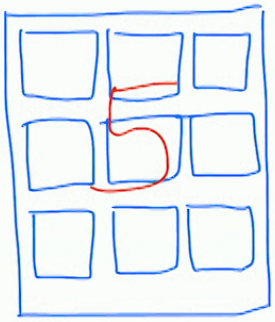


$$\frac{\| W^{\ell} \vec{\phi}(\vec{x}_i) \|}{\sum_{\ell} \| W^{\ell} \vec{\phi}(\vec{x}_i) \|^2}$$

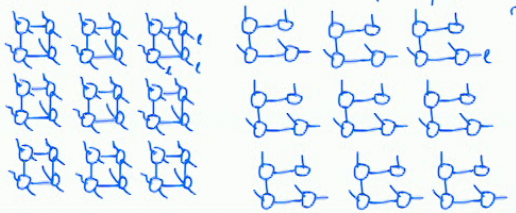
- optimization carried out based on gradient descent methods (ADAM, ...)
- gradients can be computed through autodiff or manually

Performance: Can reach ~ 98.5% test accuracy on MNIST
~ 87% test accuracy on Fashion-MNIST

extensions:



decompose image into plaquette product state instead of "snake".
Use PEPS or MPS on each plaquette



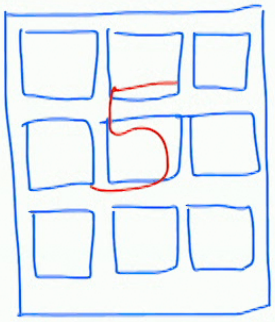
- more expressive but overfits more easily
- performance still below MPS method

methods (ADAM, ...)

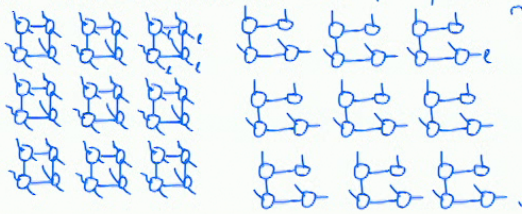
- gradients can be computed through autodiff or manually

Performance: Can reach ~ 98.5% test accuracy on MNIST
~ 87% test accuracy on Fashion-MNIST

extensions:



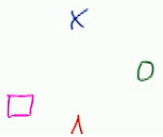
decompose image into plaquette product state instead of "snake".
Use PEPS or MPS on each plaquette



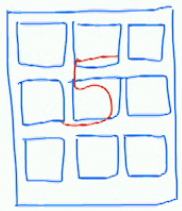
- more expressive but overfits more easily
- performance still below MPS method

Outlier detection

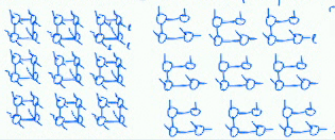
Motivation: detection of anomalous data among normal datapoints



extensions:



decompose image into plaqu
instead of "snake".
Use PEPS or MPS on each plaquette



- more expressive
  but overfits more easily
- performance still below
  MPS method

Outlier detection

Motivation: detection of anomalous data among normal datapoints



Semi-supervised tensor network approach – two class classification

Data: vectors in $\mathbb{R}^n$, where n is the number of features
(eg. pixels of image)

Basic idea:

(1) embed datapoint $\vec{x}$ into high dimensional Hilbert space

$$\vec{x} \to \vec{\phi}(\vec{x}) = |\phi(x_1)\rangle |\phi(x_2)\rangle \cdots |\phi(x_n)\rangle = |\phi(x_1) \cdots \phi(x_n)\rangle$$

↑ feature map:

feature map maps all input onto hypersphere surface

(2) use rank deficient map to collapse anomalous instances



77 / 94

Generative modelling

model the distribution of a trainingset & generate samples from this learned distribution

example: MNIST again 

each image represented by 28×28 array of integers $x_1 x_2 \cdots x_{784}$

each x_i is treated as a random variable;

each image $x_1 \cdots x_{784}$ has a joint probability $p(x_1, \cdots, x_{784})$

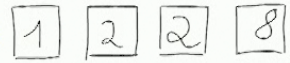
given a large but finite amount of images, can we learn a representation of $p(x_1, \cdots, x_{784})$?

$$\underbrace{\frac{1}{N_T} \sum_{i=1}^{N_T} \sum_{x_1 \cdots x_{784}} \delta_{x_1 x_1^i} \delta_{x_2 x_2^i} \cdots \delta_{x_N x_N^i}}_{\text{empirical distribution}} \approx \underbrace{p_{\vec{\theta}}(x_1, \cdots, x_N)}_{\text{model}}$$

Requirement: efficiently train & sample from $p_{\vec{\theta}}(\vec{x})$

Generative modelling

Model the distribution of a trainingset & generate samples from this learned distribution

example: MNIST again 

each image represented by 28×28 array of integers $x_1, x_2 \dots x_{784}$

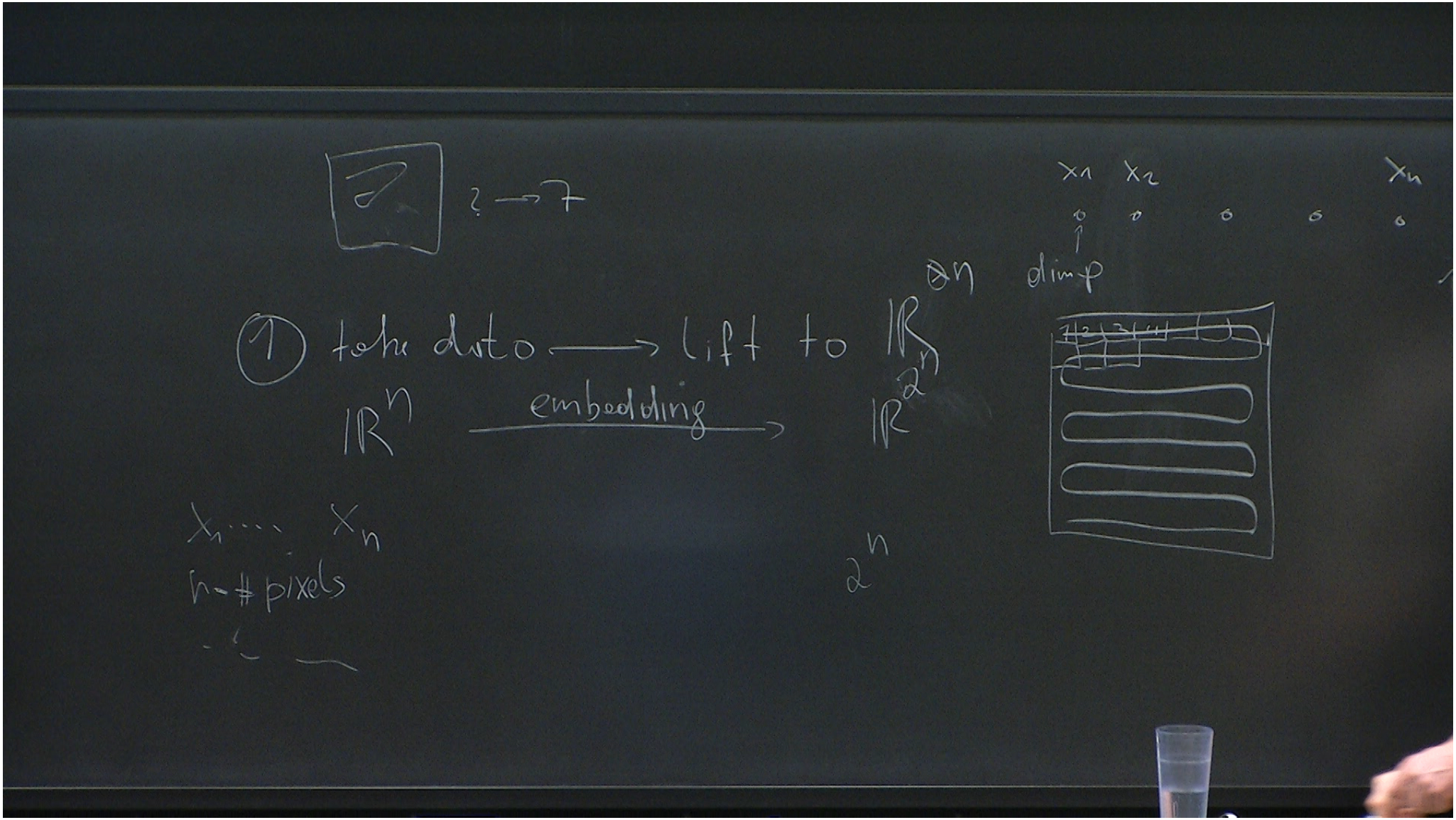
each x_i is treated as a random variable;

each image $x_1 \dots x_{784}$ has a joint probability $p(x_1, \dots, x_{784})$

given a large but finite amount of images, can we learn a representation of $p(x_1, \dots, x_{784})$?

$$\underbrace{\frac{1}{N_T} \sum_{i=1}^{N_T} \sum_{x_1 \dots x_{784}} \delta_{x_1 x_1^i} \delta_{x_2 x_2^i} \dots \delta_{x_N x_N^i}}_{\text{empirical distribution}} \approx \underbrace{p_{\vec{\lambda}}(x_1, \dots, x_N)}_{\text{model}}$$

Requirement: efficiently train & sample from $p_{\vec{\lambda}}(\vec{x})$


2 → 7
x1 x2 xn
dim p
① take data → lift to $\mathbb{R}^{2^n}$
$\mathbb{R}^n$ embedding → $\mathbb{R}^{2^n}$
$x_1 \dots x_n$
n = # pixels
2^n

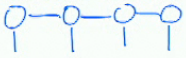
Tensor network approaches for generative modelling

Basic strategy: lift representational power of TN from many-body wave functions to multivariate distributions

$\psi(i_1, \ldots, i_N) \xrightarrow{2} p(i_1, \ldots, i_N)$

$\psi(i_1, \ldots, i_N) \in \mathbb{C}$, $p(i_1, \ldots, i_N) \in \mathbb{R}_+$

Approaches:

- represent p as "standard" TN, e.g. $p(i_1, \cdots, i_N) =$ 

  Problem: how to enforce positivity?

  has been explored by a few researchers, but not extensively used

- "Born rule": $p(i_1, i_2, \ldots, i_N) \equiv |\psi(i_1, \ldots, i_N)|^2$, i.e. diagonal of $\hat{\rho}(i_1, \ldots, i_N, i_1', \ldots, i_N')$

  → $\sum_{i_1 \ldots i_N} p(i_1, \ldots, i_N) = 1$ (normalization)

  → $p(i_1, \ldots, i_N) \geq 0$ (positivity)

  Example: modelling $p(i_1, \ldots, i_N)$ as an MPS

Tensor network approaches for generative modelling

Basic strategy: lift representational power of TN from many-body wave functions to multivariate distributions

$\psi(i_1, \dots, i_N) \xrightarrow{2} p(i_1, \dots, i_N)$, $i_n \in [-1, 1]$

$\psi(i_1, \dots, i_N) \in \mathbb{C}$, $p(i_1, \dots, i_N) \in \mathbb{R}_+$

Approaches:

- represent p as "standard" TN, e.g. $p(i_1, \dots, i_N) =$ 

  Problem: how to enforce positivity?

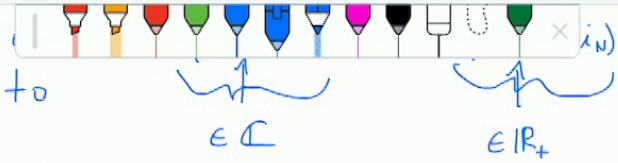
  has been explored by a few researchers, but not extensively used

- "Born rule": $p(i_1, i_2, \dots, i_N) \equiv |\psi(i_1, \dots, i_N)|^2$, i.e. diagonal of $\hat{\rho}(i_1, \dots, i_N, i'_1, \dots, i'_N)$

  → $\sum_{i_1 \dots i_N} p(i_1, \dots, i_N) = 1$ (normalization)

  → $p(i_1, \dots, i_N) \geq 0$ (positivity)

Basic strategy: lift representational power from many body wave functions to multivariate distributions



Approaches:

- represent p as "standard" TN, e.g. $p(i_1,\dots,i_N) =$ 

  Problem: how to enforce pocitivity?

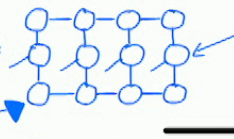
  has been explored by a few researchers, but not extensively used

- "Born rule": $p(i_1, i_2, \dots, i_N) \equiv |\psi(i_1,\dots,i_N)|^2$, i.e. diagonal of $\hat{\rho}(i_1,\dots,i_N, i'_1,\dots,i'_N)$

  → $\sum_{i_1 i_2 \dots} p(i_1,\dots,i_N) = 1$ (normalization)

  → $p(i_1,\dots,i_N) \geq 0$ (positivity)

  Example: modelling $p(i_1,\dots,i_N)$ as an MPS

  $p(i_1,\dots,i_N|W) =$  copy tensor δ_{ijk} (has no parameters)

Basic strategy: Lift representational power from many body wave functions to multivariate distributions



Approaches:

- represent p as "standard" TN, e.g. $p(i_1,\dots,i_N) =$ 

  Problem: how to enforce pocitivity?

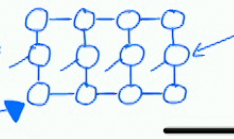
  has been explored by a few researchers, but not extensively used

- "Born rule": $p(i_1, i_2, \dots, i_N) \equiv |\psi(i_1,\dots,i_N)|^2$, i.e. diagonal of $\hat{\rho}(i_1,\dots,i_N, i'_1,\dots,i'_N)$

  → $\sum_{i_1 i_2 \dots} p(i_1,\dots,i_N) = 1$ (normalization)

  → $p(i_1,\dots,i_N) \geq 0$ (positivity)

  Example: modelling $p(i_1,\dots,i_N)$ as an MPS

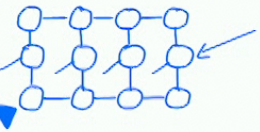
  $p(i_1,\dots,i_N|W) =$  copy tensor δ_{ijk} (has no parameters)

- "Born rule": $p(i_1, i_2, \dots, i_N) \equiv |\psi(i_1, \dots, i_N)|$, i.e. diagonal of $\hat{\rho}(i_1, \dots, i_N, i'_1, \dots, i'_N)$

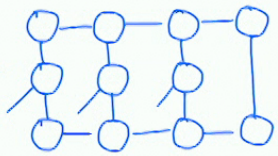
  → $\sum_{i_1 \dots i_N} p(i_1, \dots, i_N) = 1$ (normalization) $|\psi\rangle\langle\psi| = \hat{\rho}$

  → $p(i_1, \dots, i_N) \geq 0$ (positivity)

  Example: modelling $p(i_1, \dots, i_N)$ as an MPS

  $p(i_1, \dots, i_N | W) =$  ← copy tensor δ_{ijk} (has no parameters)

  MPS parameters

  marginals: $p(i_1, \dots, i_{N-1} | W) = \sum_{i_N} p(i_1, \dots, i_N | W) =$ 

- "learning" by minimizing suitable cost function

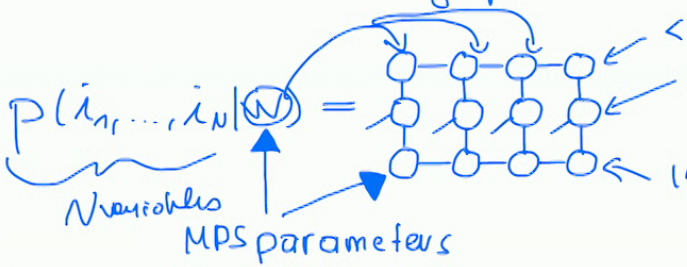
  e.g. $C(W) = -\frac{1}{N_T} \sum_{n=1}^{N_T} \log p(\vec{i}_n | W)$ → $W^* = \underset{W}{\operatorname{argmin}}\, C(W)$

  likelihood of parameters

$p(i_1, \dots, i_N) \geq 0$ (positivity)

Example: modelling $p(i_1, \dots, i_N)$ as an MPS

$\delta_{ijk} = 1$ if $i =$

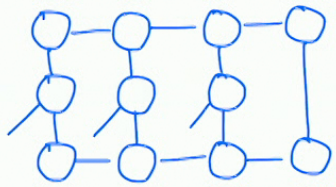


copy tensor δ_{ijk} (has no parameters)

$|\psi\rangle\langle\psi| =$



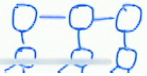
marginals: $p(i_1, \dots, i_{N-1} | W) = \sum_{i_N} p(i_1, \dots, i_N | W) =$



"learning" by minimizing suitable cost function

e.g. $C(W) = -\frac{1}{N_T} \sum_{n=1}^{N_T} \log p(\vec{i}_n | W) \longrightarrow W^* = \underset{W}{\operatorname{argmin}}\, C(W)$

likelihood of parameters

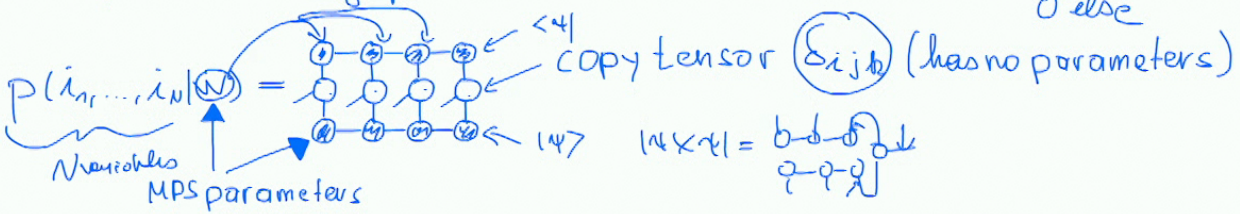


→ $\sum_{i_1 \dots i_N} p(i_1, \dots, i_N) = 1$ (normalizat

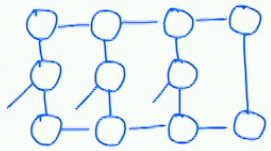
→ $p(i_1, \dots, i_N) \geq 0$ (positivity)

Example: modelling $p(i_1, \dots, i_N)$ as an MPS

$\delta_{ijk} = 1$ if $i+j=k$, 0 else



marginals: $p(i_1, \dots, i_{N-1} | W) = \sum_{i_N} p(i_1, \dots, i_N | W) =$

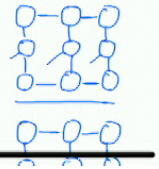


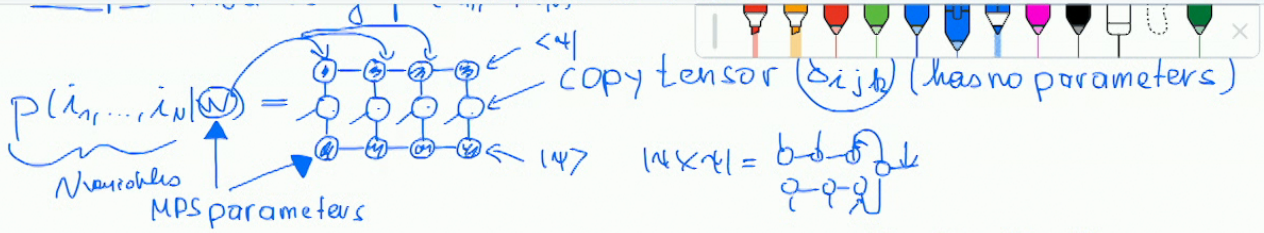
- "learning" by minimizing suitable cost function

e.g. $C(W) = -\frac{1}{N_T} \sum_{n=1}^{N_T} \log p(\vec{i}_n | W)$ → $W^* = \operatorname{argmin}_W C(W)$

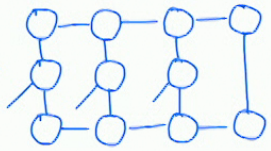
likelihood of parameters

$p(i_1, \dots, i_N | W) = \frac{|\psi|^2}{Z} =$





marginals: $p(i_1,\dots,i_{N-1}|W) = \sum_{i_N} p(i_1,\dots,i_N|W) =$

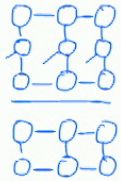


- "learning" by minimizing suitable cost function

e.g. $C(W) = -\frac{1}{N_T}\sum_{n=1}^{N_T} \log p(\vec{i}_n|W) \longrightarrow W^* = \underset{W}{\operatorname{argmin}}\, C(W)$

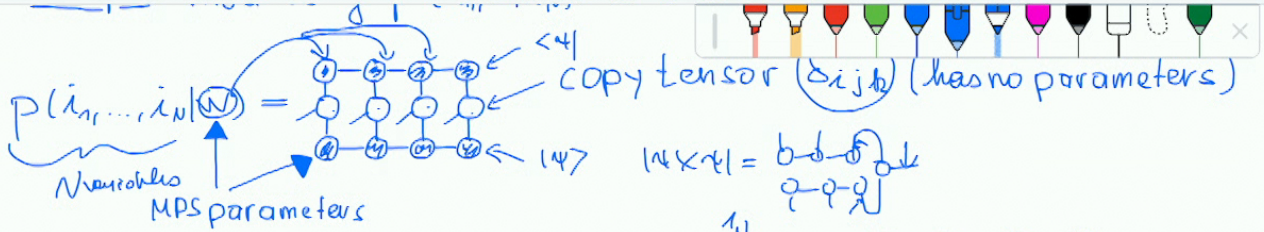
likelihood of parameters

$p(i_1,\dots,i_N|W) = \frac{|\psi|^2}{Z} =$

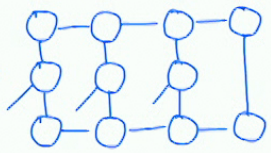


normalization factor

perform iterative minimization:



marginals: $p(i_1,\dots,i_{N-1}|W) = \sum_{i_N} p(i_1,\dots,i_N|W) =$



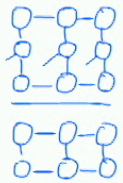
N-1 variables — N variables

- "learning" by minimizing suitable cost function

e.g. $C(W) = -\frac{1}{N_T}\sum_{n=1}^{N_T} \log p(\vec{i}_n|W) \longrightarrow W^* = \underset{W}{\operatorname{argmin}}\, C(W)$

likelihood of parameters

$p(i_1,\dots,i_N|W) = \frac{|\psi|^2}{Z} =$



normalization factor

perform iterative minimization:

given a large but finite amount of images, can we learn a representation of $p(x_1, \dots, x_{784})$?

$$\underbrace{\frac{1}{N_T} \sum_{i=1}^{N_T} \sum_{x_1 \dots x_{784}} \delta_{x_1 x_1^i} \delta_{x_2 x_2^i} \dots \delta_{x_N x_N^i}}_{\text{empirical distribution}} \approx \underbrace{p_{\vec{\lambda}}(x_1, \dots, x_N)}_{\text{model}}$$

pixel 1 … pixel N

Requirement: efficiently train & sample from $p_{\vec{\lambda}}(\vec{x})$

N-1 variables

i_N

N-variables

- "learning" by minimizing suitable cost function

e.g. $C(W) = -\frac{1}{N_T} \sum_{n=1}^{N_T} \log p(\vec{i}_n | W) \longrightarrow W^* = \underset{W}{\operatorname{argmin}}\, C(W)$

likelihood of parameters

$p(i_1, \dots, i_N | W) = \frac{|\psi|^2}{Z} =$

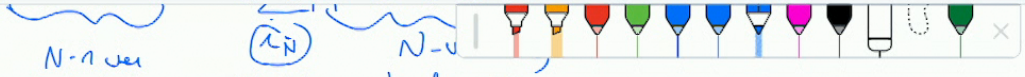
normalization factor

perform iterative minimization:

1) initialize MPS

2) for each site n:

i_n i_n

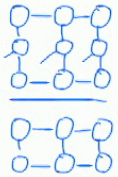


- "learning" by minimizing suitable cost function

e.g. $C(W) = -\frac{1}{N_T} \sum_{n=1}^{N_T} \log p(\vec{i}_n | W) \longrightarrow W^* = \underset{W}{\operatorname{argmin}}\, C(W)$

likelihood of parameters

$p(i_1, \dots, i_N | W) = \frac{|\psi|^2}{Z} =$



normalization factor

perform iterative minimization:

1) initialize MPS

2) for each site n:
   - compute $\frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}}$ and update $A^{i_n}_{\alpha_{n-1}\alpha_n} \longrightarrow A^{i_n}_{\alpha_{n-1}\alpha_n} - \eta \cdot \frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}}$

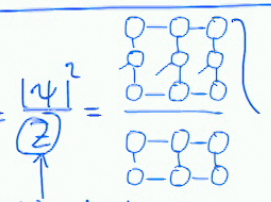
3) if converged stop, else go back to 2)

- "learning" by minimizing suitable cost function

e.g. $C(W) = -\frac{1}{N_T} \sum_{n=1}^{N_T} \log p(\vec{i}_n | W) \longrightarrow W^* = \underset{W}{\operatorname{argmin}}\, C(W)$

(W: MPS tensors; $p(\vec{i}_n|W)$: likelihood of parameters)

$$p(i_1, \dots, i_N | W) = \frac{|\psi|^2}{Z} =$$



(Z: normalization factor)

perform iterative minimization:

1) initialize MPS

2) for each site n:
   - compute $\frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}}$ and update $A^{i_n}_{\alpha_{n-1}\alpha_n} \longrightarrow A^{i_n}_{\alpha_{n-1}\alpha_n} - \eta \cdot \frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}}$

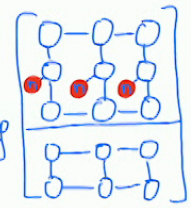
3) if converged stop, else go back to 2)

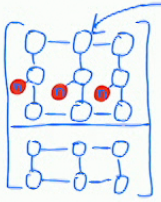
perform iterative minimization:

1) initialize MPS

2) for each site n:
   - compute $\frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}}$ and update $A^{i_n}_{\alpha_{n-1}\alpha_n} \longrightarrow A^{i_n}_{\alpha_{n-1}\alpha_n} - \eta \cdot \frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}}$

3) if converged stop, else go back to 2)

details of gradient computation: $C(w) = -\frac{1}{N_T}\sum_n \log$ 

$\frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}} = -\frac{1}{N_T}\sum_n \frac{\partial}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}} \log$ 

$A^{i_n}_{\alpha_{n-1}\alpha_n}$

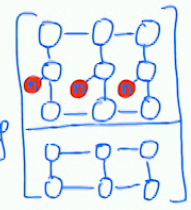
perform iterative minimization:

1) initialize MPS $\longrightarrow p(i_1 \dots i_N)$

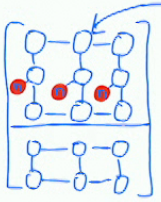
2) for each site n:

- compute $\frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}}$ and update $A^{i_n}_{\alpha_{n-1}\alpha_n} \longrightarrow A^{i_n}_{\alpha_{n-1}\alpha_n} - \eta \cdot \frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}}$

3) if converged stop, else go back to 2)

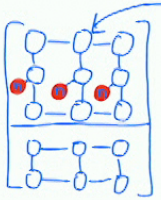
details of gradient computation: $C(w) = -\frac{1}{N_T} \sum_n \log$ 

$$\frac{\partial C}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}} = -\frac{1}{N_T} \sum_n \frac{\partial}{\partial A^{i_n}_{\alpha_{n-1}\alpha_n}} \log$$

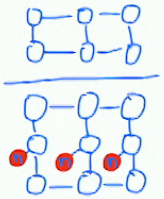
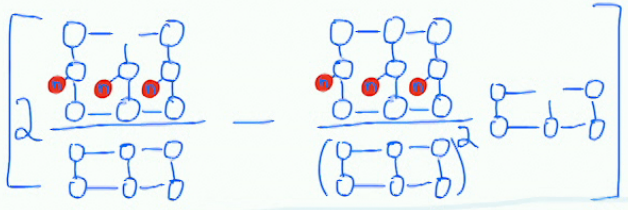


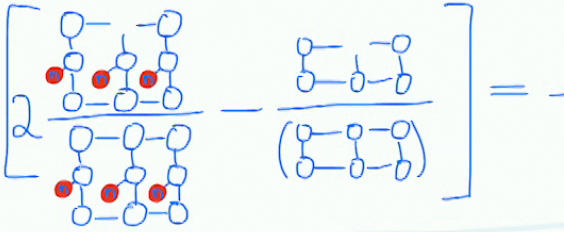
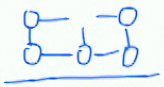
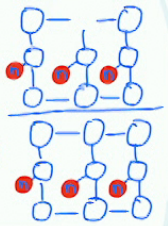
$A^{i_n}_{\alpha_{n-1}\alpha_n}$

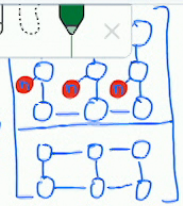
details of gradient computation: $C(\omega) = -\frac{1}{N_T}\sum_n \log$ 

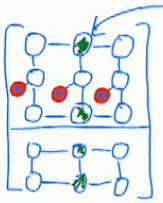
$$\frac{\partial C}{\partial A^{i_n}_{d_{n-1}d_n}} = -\frac{1}{N_T}\sum_n \frac{\partial}{\partial A^{i_n}_{d_{n-1}d_n}} \log$$ 

$A^{i_n}_{d_{n-1}d_n}$

$$= -\frac{1}{N_T}\sum_n$$  

$$= -\frac{1}{N_T}\sum_n$$  $= \frac{}{}$  $- \frac{2}{N_T}\sum_{n=1}^{N_T}$ 

details of gradient computation: $C(w) = -\frac{1}{N_T}\sum_{n=1}^{N_T} \log$ 

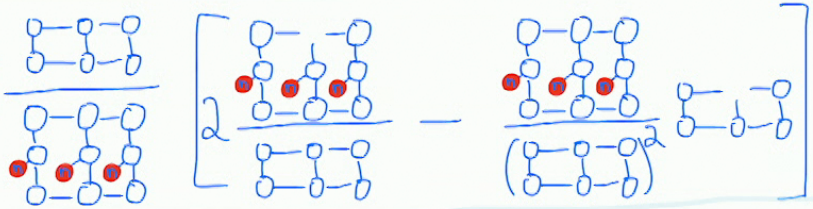
$$\frac{\partial C}{\partial A^{i_n}_{d_{n-1}d_n}} = -\frac{1}{N_T}\sum_n \frac{\partial}{\partial A^{i_n}_{d_{n-1}d_n}} \log$$ 

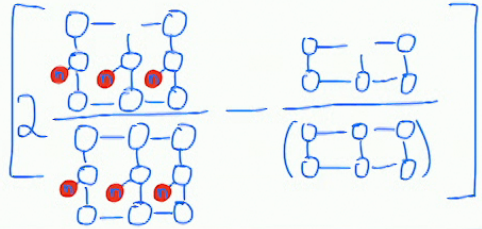
$A^{i_n}_{d_{n-1}d_n}$

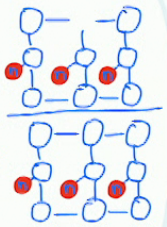
samples

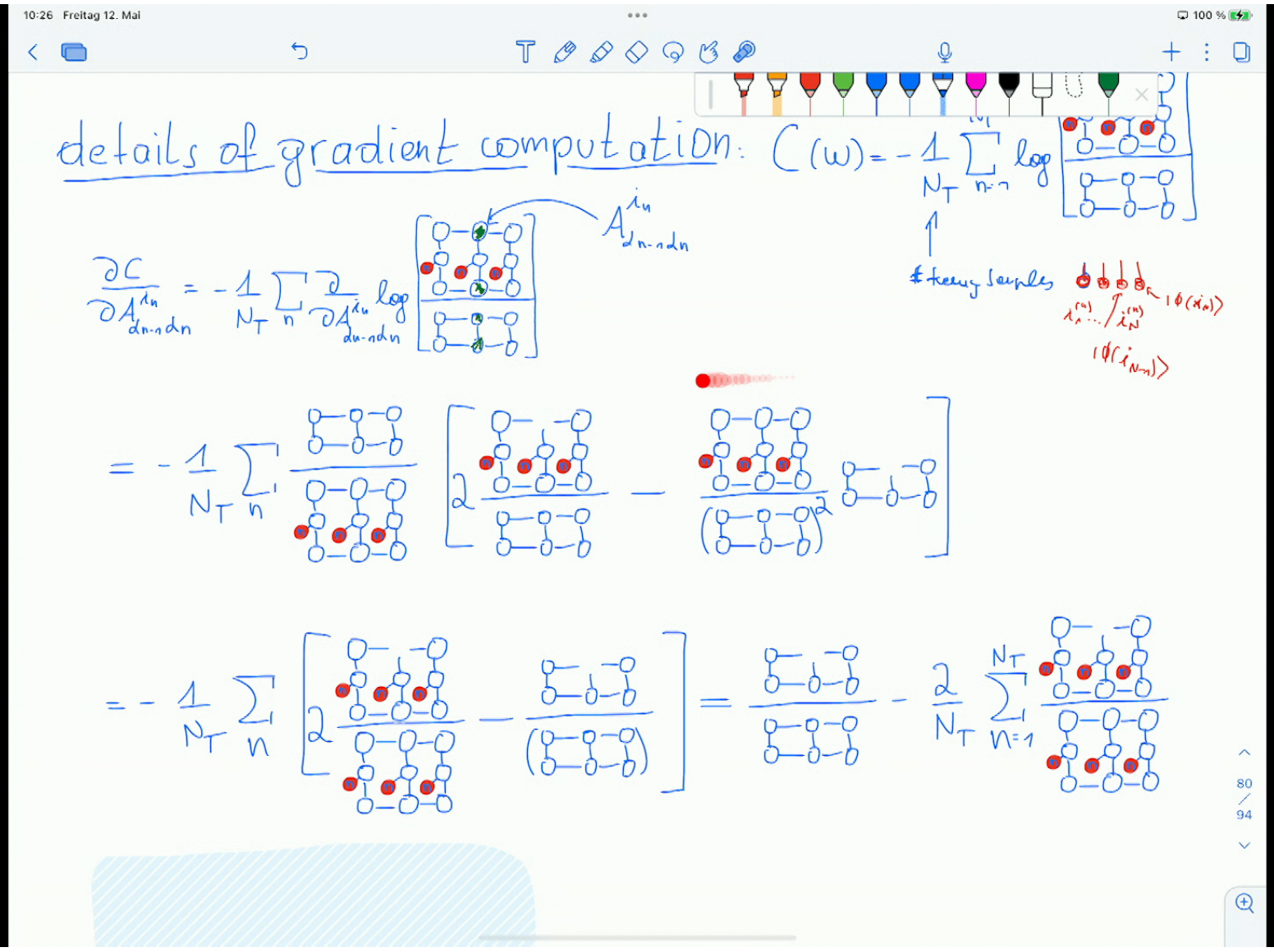
$|\phi(i_n)\rangle$

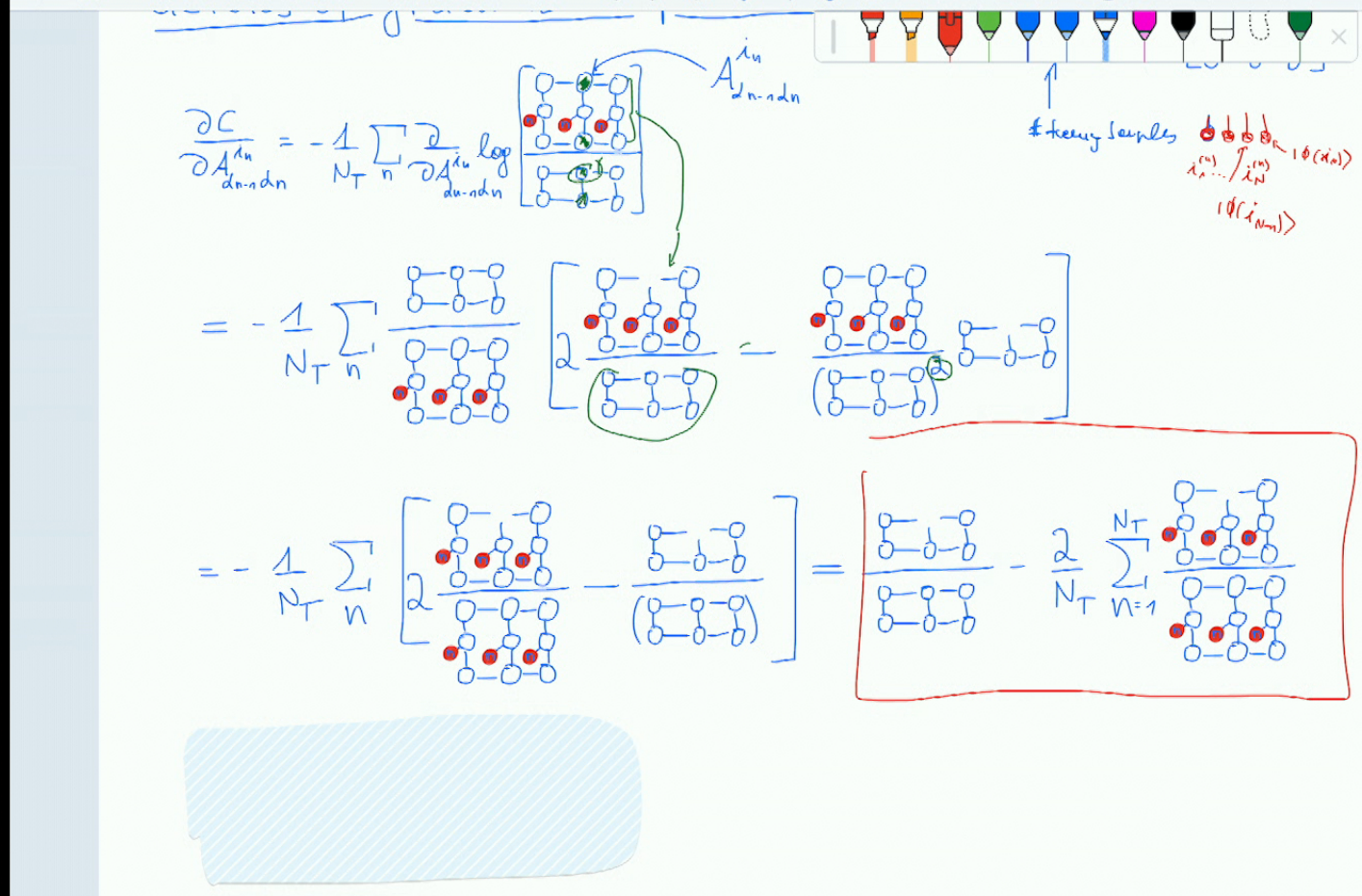
$|\phi(i_{N-1})\rangle$

$$= -\frac{1}{N_T}\sum_n$$ 

$$= -\frac{1}{N_T}\sum_n$$ 

$$= \frac{2}{N_T}\sum_{n=1}^{N_T}$$ 


details of gradient computation: $C(w) = -\frac{1}{N_T}\sum_{n=1}^{N_T} \log$
$\frac{\partial C}{\partial A^{i_n}_{d_{n-1}d_n}} = -\frac{1}{N_T}\sum_n \frac{\partial}{\partial A^{i_n}_{d_{n-1}d_n}} \log$
$A^{i_n}_{d_{n-1}d_n}$
samples
$|\phi(i_n)\rangle$
$|\phi(i_{N-1})\rangle$
$= -\frac{1}{N_T}\sum_n$
$= -\frac{1}{N_T}\sum_n$
$= \frac{2}{N_T}\sum_{n=1}^{N_T}$


$\frac{\partial C}{\partial A^{i_n}_{d_{n-1}d_n}} = -\frac{1}{N_T}\sum_n \frac{\partial}{\partial A^{i_n}_{d_{n-1}d_n}} \log$
$A^{i_n}_{d_{n-1}d_n}$
training samples
$|\phi(x_1)\rangle$
$|\phi(x_{N_T})\rangle$
$= -\frac{1}{N_T}\sum_n$
$= -\frac{1}{N_T}\sum_n$
$= \frac{2}{N_T}\sum_{n=1}^{N_T}$

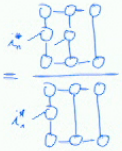
Perfect sampling (Sandvik, Vidal)

Autoregressive property: $p(x_1, \dots, x_N) = p(x_1)p(x_2|x_1)p(x_3|x_2,x_1)\cdots p(x_N|x_{N-1},\dots,x_1)$

$p(i_1, \dots, i_N) =$  (assume $Z=1$, i.e. implicit normalization)

1) sample i_1^* from $p(i_1) =$ 

2) fix value of i_1 to i_1^*

3) compute $p(i_2|i_1^*) = \frac{p(i_2, i_1^*)}{p(i_1^*)} =$ 

4) sample i_2^* from $p(i_2^*|i_1)$

5) fix $i_1 = i_1^*, i_2 = i_2^*$, compute $p(i_3|i_1^*, i_2^*)$ a.s.o.

$i_1^*, i_2^*, \dots, i_N^*$ is an unbiased sample from $p(i_1, \dots, i_N)$

Generalizations & applications

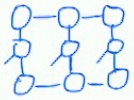
uniform MPS: use infinite MPS for sequence modelling of arbitrary length; technically very similar to finite MPS approach (Miller, Rabusseau, Terilla)

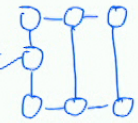
PEPS: similar idea, but technically more involved
employs multimodal representation $p(\vec{i}) = \sum_k d_k p_k(\vec{i}),\ \sum_k d_k = 1$
sampling & optimization can only be done approximately
Applied to e.g. image generation

TTNs: due to loop-free property of TTNs, the MPS approach can be lifted to the TTN case with only minor modifications (Cheng, Wang, Xiang, Zhang, '19)
im[illegible]

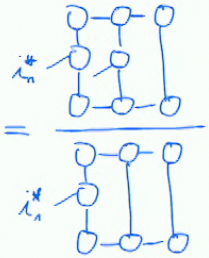
Perfect sampling (Sundaram, Wilde)

Autoregressive property: $p(x_1, \dots, x_N) = p(x_1)\, p(x_2|x_1)\, p(x_3|x_2 x_1) \cdots p(x_N|x_{N-1} \cdots x_1)$

$p(i_1, \dots, i_N) =$  (assume $Z=1$, i.e. implicit normalization)

1) sample i_1^* from $p(i_1) =$ 

2) fix value of i_1 to i_1^*

3) compute $p(i_2|i_1^*) = \dfrac{p(i_2, i_1^*)}{p(i_1^*)} =$ 

4) sample i_2^* from $p(i_2^*|i_1)$

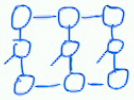
5) fix $i_1 = i_1^*, i_2 = i_2^*$, compute $p(i_3|i_1^*, i_2^*)$ a.s.o.

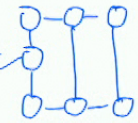
$i_1^*, i_2^*, \dots, i_N^*$ is an unbiased sample from $p(i_1, \dots, i_N)$

Generalizations & applications

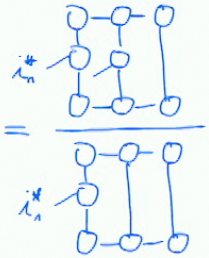
Perfect sampling

Autoregressive property: $p(x_1, \dots, x_N) = p(x_1)\, p(x_2|x_1)\, p(x_3|x_2 x_1) \cdots p(x_N|x_{N-1} \dots x_1)$

$p(i_1, \dots, i_N) =$  (assume $Z=1$, i.e. implicit normalization)

1) sample i_1^* from $p(i_1) =$ 

2) fix value of i_1 to i_1^*

3) compute $p(i_2|i_1^*) = \dfrac{p(i_2, i_1^*)}{p(i_1^*)} =$ 

4) sample i_2^* from $p(i_2^*|i_1)$

5) fix $i_1 = i_1^*, i_2 = i_2^*$, compute $p(i_3|i_1^*, i_2^*)$ a.s.o.

$i_1^*, i_2^*, \dots, i_N^*$ is an unbiased sample from $p(i_1, \dots, i_N)$

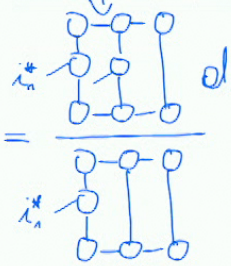
Generalizations & applications

1) sample i_1^* from $p(i_1)$ =



2) fix value of i_1 to i_1^*

3) compute $p(i_2 | i_1^*) = \frac{p(i_2, i_1^*)}{p(i_1^*)}$ =



draw sample i_2^*

4) sample i_2^* from $p(i_2^* | i_1)$

5) fix $i_1 = i_1^*, i_2 = i_2^*$, compute $p(i_3 | i_1^*, i_2^*)$ a.s.o.

$i_1^*, i_2^*, \dots, i_N^*$ is an unbiased sample from $p(i_1, \dots, i_N)$

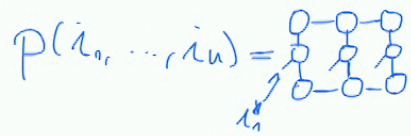
Generalizations & applications

uniform MPS: use infinite MPS for sequence modelling of arbitrary length; technically very similar to finite MPS approach (Miller, Rabusseau, Terilla)

PEPS: similar idea, but technically more involved

employs multimodal representation $p(\vec{i}) = \sum_k \alpha_k p_k(\vec{i}), \sum_k \alpha_k = 1$

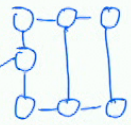
sampling & optimization [illegible] approximately



(assume $z=1$, i.e. ...)

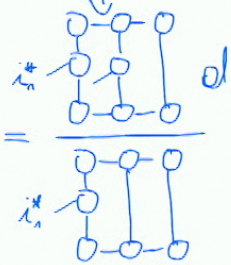
$i_1^*, i_2^*, \dots, i_N^* \sim p(i_1, \dots, i_N)$

1) sample i_1^* from $p(i_1)$ =



2) fix value of i_1 to i_1^*

3) compute $p(i_2 | i_1^*) = \frac{p(i_2, i_1^*)}{p(i_1^*)}$ =



draw sample i_2^*

4) sample i_2^* from $p(i_2^* | i_1)$

5) fix $i_1 = i_1^*, i_2 = i_2^*$, compute $p(i_3 | i_1^*, i_2^*)$ a.s.o.

$i_1^*, i_2^*, \dots, i_N^*$ is an unbiased sample from $p(i_1, \dots, i_N)$

Generalizations & applications

uniform MPS: use infinite MPS for sequence modelling of arbitrary length; technically very similar to finite MPS approach (Miller, Rabusseau, Terilla)

Generalizations & applications

uniform MPS: use infinite MPS for sequence modelling of arbitrary length; technically very similar to finite MPS approach (Miller, Rabusseau, Terilla)

PEPS: similar idea, but technically more involved
employs multimodal representation $p(\vec{i}) = \sum_k \alpha_k p_k(\vec{i})$, $\sum_k \alpha_k = 1$
sampling & optimization can only be done approximately
Applied to e.g. image generation

TTNs: due to loop-free property of TTNs, the MPS approach can be lifted to the TTN case with only minor modifications (Cheng, Wang, Xiang, Zhang, '19)
image generation

numerical mathematics, high dimensional integration, PDEs

numerical mathematics, high dimensional integration, PDEs

Tensor networks, in particular MPS, find increasing applications in numerical mathematics.
Two applications discussed in the following:

a) high dimensional integration of multivariate functions

b) solving partial differential equations (Navier-Stokes equation)

Strategy: represent/approximate a multivariate function $f(x_1, x_2, \dots, x_N)$ of N continous variables as an MPS/TT.

a) Tensor-train cross approximation & high dimensional integration

Task: given a multivariate function $f(x_1, x_2, \dots, x_N)$ compute its definite integral $\int dx_1 \dots dx_N f(x_1, \dots, x_N)$

The complexity of this problem is

Strategy: 1) approximate $f(x_1, x_2, \dots, x_N)$ as an MPS/TT using TT-cross approximation
2) perform integration using efficient TN methods

- discretize $x_i \in [x_i^{min}, x_i^{max}]$ into d_i discrete values

numerical mathematics, high dimensional integration, PDEs

Tensor networks, in particular MPS, find increasing applications in numerical mathematics.

Two applications discussed in the following:

a) high dimensional integration of multivariate functions

b) solving partial differential equations (Navier-Stokes equation)

Strategy: represent/approximate a multivariate function $f(x_1, x_2, \dots, x_N)$ of N continous variables as an MPS/TT.

a) Tensor-train cross approximation & high dimensional integration

Task: given a multivariate function $f(x_1, x_2, \dots, x_N)$ compute its definite integral $\int dx_1 \dots dx_N f(x_1, \dots, x_N)$

The complexity of this problem is

Strategy: 1) approximate $f(x_1, x_2, \dots, x_N)$ as an MPS/TT using TT-cross approximation
2) perform integration using efficient TN methods

- discretize $x_i \in [x_i^{min}, x_i^{max}]$ into d_i discrete values

Partial differential equations & function encoding

Gourianov et. al. 2022, Garcia Ripoll 2021, Lotorre 2005

Task: find solution to a partial differential equation, e.g. Navier-Stokes:

$f(x_1, x_2, x_3) \hat{=}$ velocity field $\vec{v}(\vec{x}) = \begin{bmatrix} v_x(\vec{x}) \\ v_y(\vec{x}) \\ v_z(\vec{x}) \end{bmatrix}$

example: Incompressible NS equations:

$$\vec{\nabla} \vec{v} = 0$$

$$\frac{\partial \vec{v}}{\partial t} + \vec{v}(\vec{\nabla}\vec{v}) = -\vec{\nabla} p + \nu \vec{\nabla}^2 \vec{v}$$

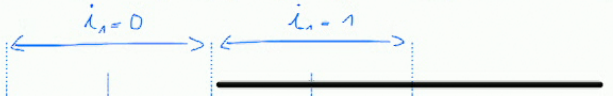
ν: kinematic viscosity (material constant)

Strategy: a) encode each component $v_i(\vec{x})$ into an MPS

b) adapt standard approaches to integrate INSE to MPS setting use standard MPS techniques to carry out integration of PDE in the MPS parametrization

a) **MPS-encoding of $f(x_i)$** (discretization)

- assume x_i is bounded, i.e. $x_i^{min} \leq x_i \leq x_i^{max}$
- discretize x_i using binary representation, i.e. for $x_i \in [x_i^{min}, x_i^{max}]$ discretize interval into 2^N gridpoints $\{x_i^{(k)}\}$ labelled by binary string $i_1 i_2 \dots i_N$, $i_k = \{0,1\}$; increments $x_i^{(k+1)} - x_i^{(k)} = \epsilon$



Task: find solution to a partial different

$f(x_1, x_2, x_3) \hat{=}$ velocity field $\vec{v}(\vec{x}) = \begin{bmatrix} v_x(\vec{x}) \\ v_y(\vec{x}) \\ v_z(\vec{x}) \end{bmatrix}$

example: Incompressible NS equations:

$$\vec{\nabla}\vec{v} = 0$$

$$\frac{\partial \vec{v}}{\partial t} + \vec{v}(\vec{\nabla}\vec{v}) = -\vec{\nabla}p + \nu \vec{\nabla}^2 \vec{v}$$

ν: kinematic viscosity (material constant)

Strategy: a) encode each component $v_i(\vec{x})$ into an MPS

b) adapt standard approaches to integrate INSE to MPS setting
use standard MPS techniques to carry out integration of PDE in the MPS parametrization

a) **MPS-encoding of $f(x_i)$ (discretization)**

- assume x_i is bounded, i.e. $x_i^{min} \leq x_i \leq x_i^{max}$
- discretize x_i using binary representation, i.e. for $x_i \in [x_i^{min}, x_i^{max}]$ discretize interval into 2^N gridpoints $\{x_i^{(k)}\}$ labelled by

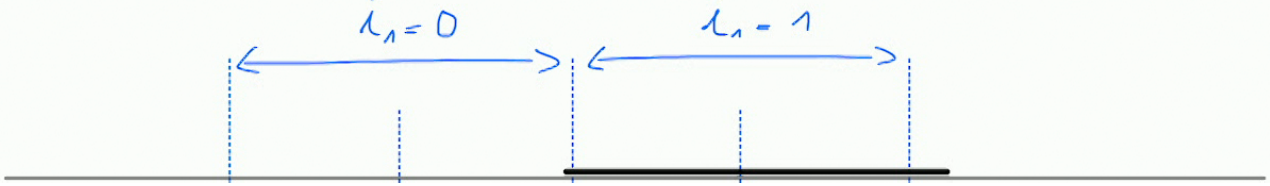
$\frac{\partial}{\partial t} + \nu(\ldots) =$ ↑ kinematic viscosity (material constant)

Strategy: a) encode each component $v_i(\vec{x})$ into an MPS

b) adapt standard approaches to integrate INSE to MPS setting
use standard MPS techniques to carry out integration of PDE in the parametrization

a) MPS-encoding of $f(x_i)$ (discretization)

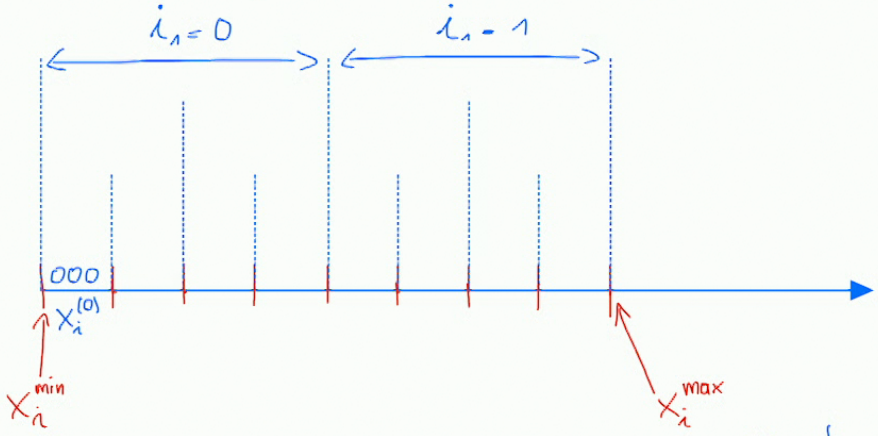
- assume x_i is bounded, i.e. $x_i^{min} \leq x_i \leq x_i^{max}$
- discretize x_i using binary representation, i.e. for $x_i \in [x_i^{min}, x_i^{max}]$ discretize interval into 2^N gridpoints $\{x_i^{(k)}\}$ labelled by binary string $i_1 i_2 \ldots i_N$, $i_k = \{0,1\}$; increments $x_i^{(k+1)} - x_i^{(k)} =$



parametrization

a) MPS-encoding of $f(x_i)$ (discretization)

- assume x_i is bounded, i.e. $x_i^{min} \leq x_i \leq x_i^{max}$
- discretize x_i using binary representation, i.e. for $x_i \in [x_i^{min}, x_i^{max}]$ discretize interval into 2^N gridpoints $\{x_i^{(k)}\}$ labelled by binary string $i_1 i_2 \dots i_N$, $i_k = \{0,1\}$; increments $x_i^{(k+1)} - x_i^{(k)} = \epsilon$



- represent $f(x_i)$ as a "quantum state":

can be easily extended to 2d

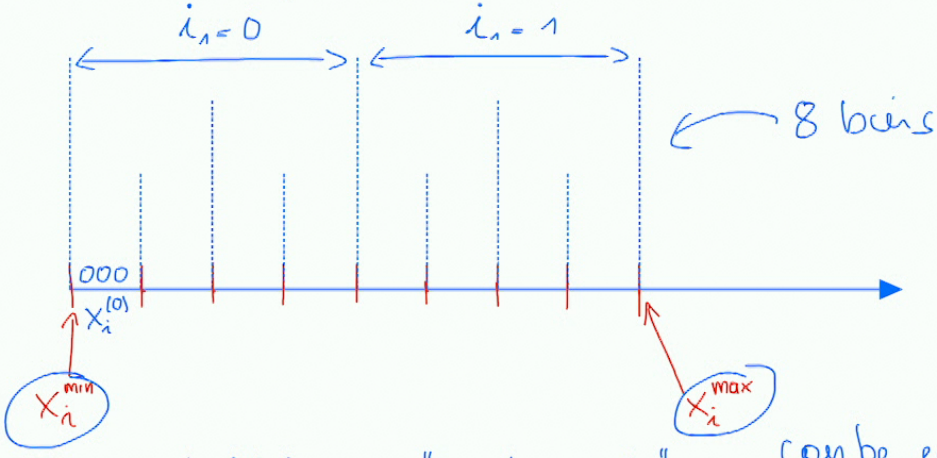
$$|f\rangle = \sum_{k=1}^{2^N-1} f(x_i^{(k)}) |\text{bin-repr}(k)\rangle =$$



$\approx$

a) MPS-encoding of $f(x_i)$ (discretization)

- assume x_i is bounded, i.e. $x_i^{min} \leq x_i \leq x_i^{max}$
- discretize x_i using binary representation, i.e. for $x_i \in [x_i^{min}, x_i^{max}]$ discretize interval into 2^N gridpoints $\{x_i^{(k)}\}$ labelled by binary string $i_1 i_2 \cdots i_N$, $i_k = \{0,1\}$; increments $x_i^{(k+1)} - x_i^{(k)} = \varepsilon$



← 8 bins $= 2^3$ bins

- represent $f(x_i)$ as a "quantum state":

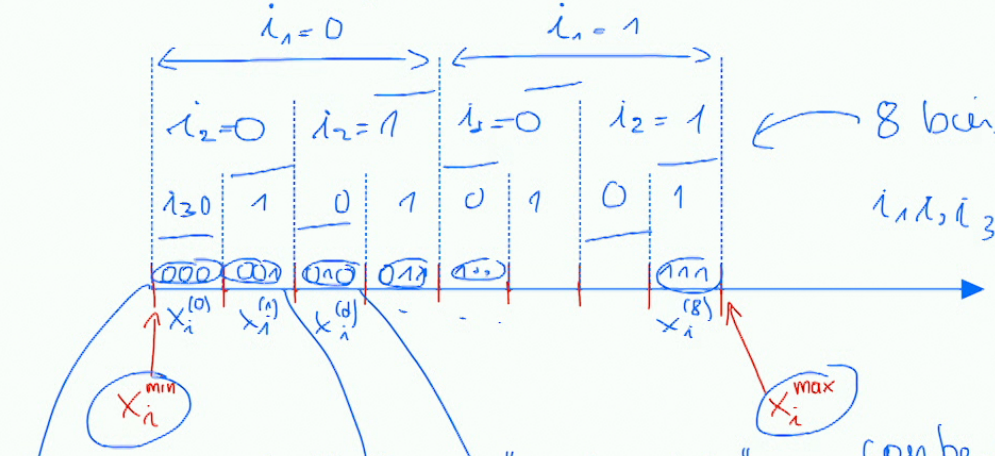
can be easily extended to 2d

$$|f\rangle = \sum_{k=0}^{2^N-1} f(x_i^{(k)}) \underbrace{|\text{bin-repr}(k)\rangle}_{|i_1^{(k)} i_2^{(k)} \cdots i_N^{(k)}\rangle} = \quad \approx$$



discretize interval into 2^N gridp

binary string $i_1 i_2 \cdots i_N$, $i_k = \{0,1\}$; increments $x_i^{(k+1)} - x_i^{(k)} = \epsilon$



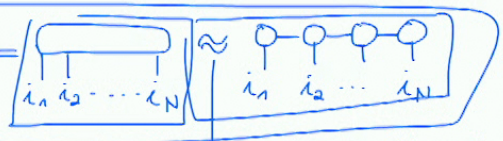
8 bins $= 2^3$ bins

$i_1 i_2 i_3$

- represent $f(x)$ as a "quantum state":

can be easily extended to 2d

$$|f\rangle = \sum_{k=0}^{2^N-1} f(x_i^{(k)}) \, |\text{bin-repr}(k)\rangle = |i_1^{(k)} i_2^{(k)} \cdots i_N^{(k)}\rangle$$



???

|000⟩ (001) (010)

85 / 94

b) translation of standard ut equation methods into MPS language

(0011) (0110)

b) translation of standard int egvation methods into MPS language

example: $\frac{\partial}{\partial x}$ or $\frac{\partial^2}{\partial x^2}$ (components of $\vec{\nabla}$ and ∇^2) are discretized

$$\rightarrow \frac{\partial f(x)}{\partial x} \approx \frac{f(x+\epsilon)-f(x-\epsilon)}{2\epsilon} = \frac{f(x^{(k+1)})-f(x^{(k-1)})}{x^{(k+1)}-x^{(k-1)}}$$

We'll use the definition $|k\rangle \equiv |\text{bin}(k)\rangle$, $k = 0, \dots, 2^N-1$

with the definition $S^+|k\rangle = |k+1\rangle$ we can express $\frac{\partial}{\partial x}$ as

$$|f'\rangle = \frac{\partial}{\partial x}|f\rangle = \frac{S^+|f\rangle - S|f\rangle}{2\epsilon} = |f'\rangle = \sum_{k=0}^{2^N-1} f'(x^{(k)})\,|k\rangle$$

S^+ increases $|k\rangle$ by one; it has a simple MPO representation:

How to increase a bitstring by 1?

$i_1 i_2 \dots i_N$

b) translation of standard ut equation into HPD language

example: $\frac{\partial}{\partial x}$ or $\frac{\partial^2}{\partial x^2}$ (components of $\vec{\nabla}$ and ∇^2) are discretized

$$\rightarrow \frac{\partial f(x)}{\partial x} \approx \frac{f(x+\varepsilon)-f(x-\varepsilon)}{2\varepsilon} = \frac{f(x^{(k+1)})-f(x^{(k-1)})}{x^{(k+1)}-x^{(k-1)}}$$

We'll use the definition $|k\rangle \equiv |\text{bin}(k)\rangle$, $k = 0, \ldots, 2^N-1$

with the definition $S^+|k\rangle = |k+1\rangle$ we can express $\frac{\partial}{\partial x}$ as

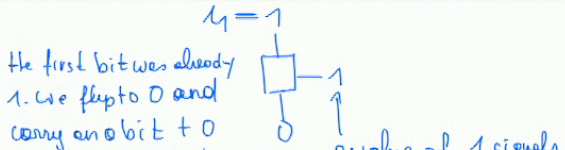
$$|f'\rangle = \frac{\partial}{\partial x}|f\rangle = \frac{S^+|f\rangle - S|f\rangle}{2\varepsilon} = |f'\rangle = \sum_{k=0}^{2^N-1} f'(x^{(k)})\,|k\rangle$$

S^+ increases $|k\rangle$ by one; it has a simple HPD representation:

How to increase a bitstring by 1?

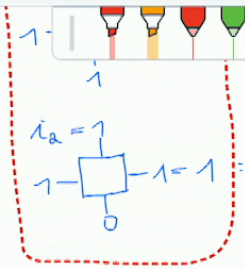
$i_1 i_2 \ldots i_N$

$|k\rangle = (000)1\ldots\rangle$



the first bit was already 1. We flip to 0 and carry an 0 bit + 0 see if the next bit should be flipped

0 value of 1 signals that we still need to apply a flip.



apply 1L to all coming sites

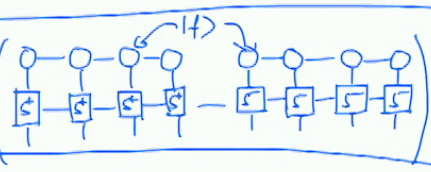
we were unable to flip the bit without carry over try at the next bit!

definition of MPO for S^+

S^+ has bond dimension of

S^- similar to S^+

$$|f'\rangle = \frac{1}{2\varepsilon}\left(S^+|f\rangle - S^-|f\rangle\right) = \frac{1}{2\varepsilon}$$



$|f'\rangle$ is an MPS with increased bond dimension, and needs to be truncated using standard methods.

∂_x and ∂_x^2 can be written as an MPO with a low bond

given velocity field $\vec{v}(t,\vec{x})$ at time t, forward integration of INSE can be carried out in multiple ways, e.g. using variational approach:

$$\vec{w}(t+\Delta t, \vec{x}) = \arg \min_{\vec{w}^*}$$

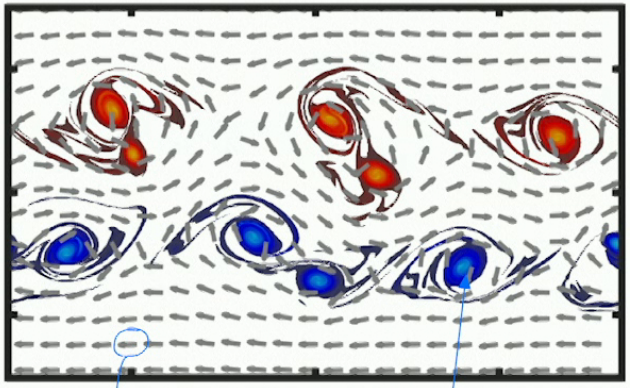
$$\theta(\vec{w}^*) = \mu \left\| \vec{\nabla} \vec{w}^* \right\|^2 + \left\| \frac{\vec{w}^* - \vec{w}}{\Delta t} + (\vec{w} \cdot \vec{\nabla}) \vec{w} - \nu \vec{\nabla}^2 \vec{w} \right\|^2$$

the minimization is carried out by employing standard MPS optimization techniques.

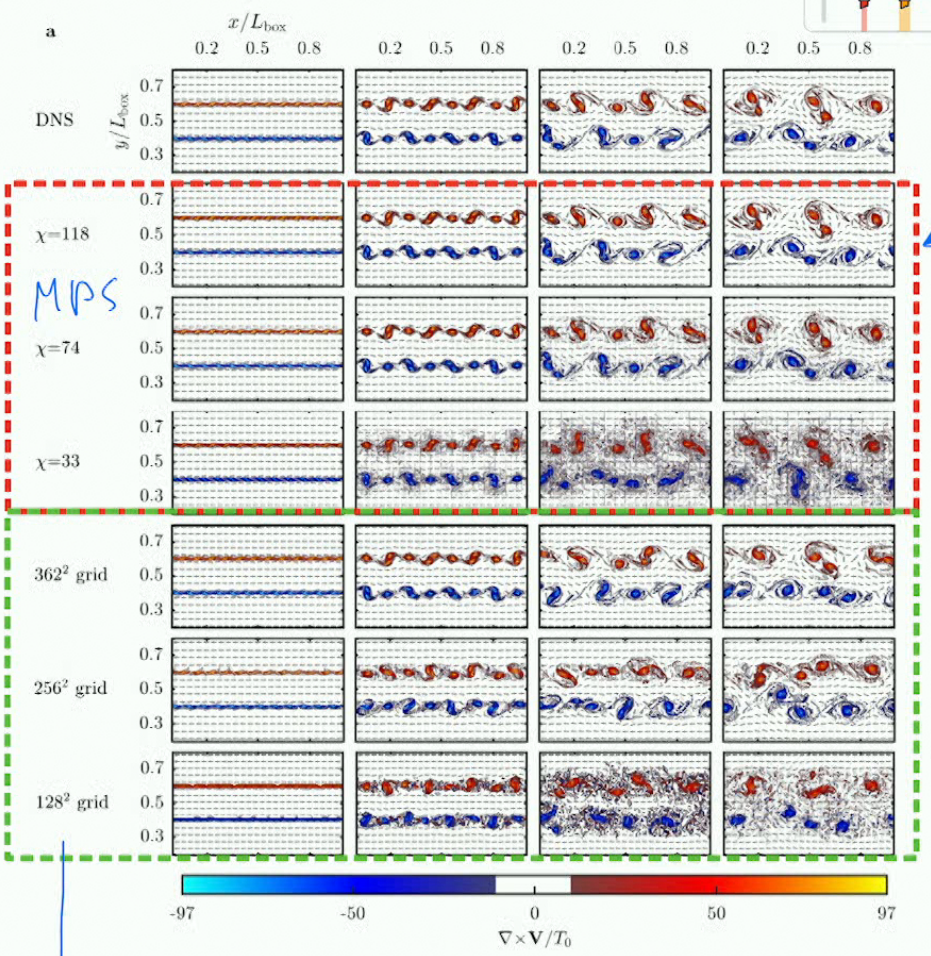
What is the benefit of this approach here?

Turbulence. solutions to NSE can exhibit turbulent structure,
The velocity field $\vec{v}(t,\vec{x})$ in these cases exhibits dynamics
on different lengths scales

(from Gourianov et. al)



MPS parametrization naturally resolves shorter and shorter scales as
one increases length of the MPS: $\bullet-\bullet-\bullet-\bullet-\bullet-\bullet$
$i_1\ i_2\ \cdots\ i_N$
scale decreases as we move
to the right



Case	Compression	Scheme	Inaccuracy
TDJ	1:1	DNS	0
TDJ	1:8	MPS	0.0119
TDJ	1:8	URDNS	0.2612
TDJ	1:16	MPS	0.0485
TDJ	1:16	URDNS	0.3333
TDJ	1:64	MPS	0.2404
TDJ	1:64	URDNS	0.3201
TGV	1:1	DNS	0.002
TGV	1:25	MPS	0.0385
TGV	1:25	URDNS	0.1599
TGV	1:49	MPS	0.0844
TGV	1:49	URDNS	0.2133
TGV	1:78	MPS	0.2618
TGV	1:78	URDNS	0.4563

TABLE 1: **Quantitative comparison betw MPS and URDNS simulations for the TD TGV flow cases.** The rows corresponding to th flow tabulate $\sigma(s,c)$, as defined in Eq. (19). $\sigma($ measures the discrepancy of the URDNS and M Reynolds stresses to that of DNS in Fig. 2b. The flow case rows tabulate $e(s,c)$, which is define Eq. (20) and represents the total numerical diffu Fig. 3b. The nearer $\sigma(s,c)$ or $e(s,c)$ is to 0, the accurate the simulation in question is.

DNS with fixed number of parameters

(Gouriano v et al)