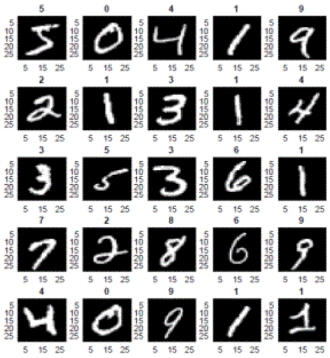
Goal: construct a network that can recognize handwritten numbers $\in \{0, 1, \dots, 9\}$
from MNIST (Modified National Institute of Standards and Technology) data set.



- contains 60000 training images, labeled by 'image ID' $n = 1, \dots, N$

  and 10000 testing images
- 28x28 pixels, labeled by 'pixel ID' $\ell = 1, \dots, 784 := L$
- each pixel contains grey-scale value $x^\ell_n \in (\underset{\text{white}}{0}, \underset{\text{black}}{1}) := \underset{\text{unit interval}}{I} \subset \mathbb{R}$
- image n is represented by 'image vector' $\vec{x}_n = (x^1_n, \dots, x^L_n) \in I^L$
- each image has been assigned a 'target name' $\vec{t}_n \in \{\vec{e}_0, \dots, \vec{e}_9\}$,

  where $\vec{e}_j = (0, 0, \dots, 1, \dots, 0)$, a basis vector in $\mathbb{N}^{10}$, represents the number $j \in \{0, \dots, 9\}$

Goal: find 'decision function' $\vec{f}$ that maps image vector to 'predicted name',

$$\vec{f} : I^L \to \mathbb{N}^{10}, \qquad \vec{x}_n \longmapsto \vec{f}(\vec{x}_n) := \vec{f}_n$$

'predicted name'

while minimizing the cost function $$C = \sum_{n=1}^{N} (\vec{f}_n - \vec{t}_n)^2$$

target name

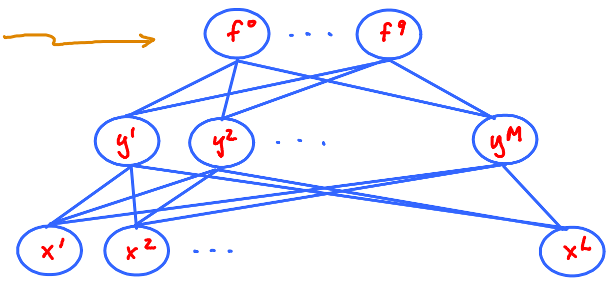
[Alternatively, choose $\vec{f} \in I^{10}$, $|\vec{f}| = 1$ then f^j = probability that image is the number j]

1. Neural network

'output layer': $\vec{f} = (f^0, \dots, f^9) \in \mathbb{N}^{10}$

'hidden layer': $\vec{y} = (y^1, \dots, y^M) \in I^M$

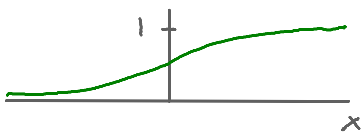
'input layer': $\vec{x} = (x^1, \dots, x^L) \in I^L$



Non-linear transformation: $$y^k = \sigma(b^k + w^k_\ell x^\ell)$$

'bias' 'weight' 'input'

with $\sigma(x) = \dfrac{1}{1 + e^{-x}}$ 'sigmoid function'



mimics neuron: 'fires' when input is above threshold 'soft-max layer': $f^j = \dfrac{e^{(a^j + u^j_\ell y^\ell)}}{\sum_{i=0}^{9} e^{(a^i + u^i_\ell y^\ell)}}$

use of exponentials emphasizes largest output at expense of others

$\vec{v} = (b, w, a, u)$ are variational parameters, used to minimize C (e.g. by gradient descent)

$\Longrightarrow$ 'train the network' = 'supervised learning'

<u>Multilayer networks</u> (many layers = 'deep learning')

All of the above is just one possible Ansatz.
Many others can and have been tried.

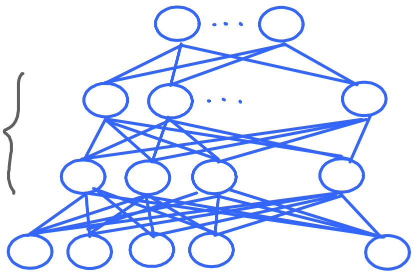
E.g.: multilayer networks:

hope is: will capture hierarchical structure better

output layer:

hidden layers:

input layer:



As before, sigmoid functions can be used to map input to output from one layer to the next.

Optimize cost function using gradient descent: $C = C(\vec{v})$

parameters of network (a, u, b, w)

Gradient: $-\vec{\nabla} C = -\left(\frac{\partial C}{\partial v^1}, \frac{\partial C}{\partial v^2}, \dots \right)$ points in direction of steepest descent:

New variables: $\vec{v}\,' = \vec{v} - \eta \vec{\nabla} C$

'learning rate' (should be neither too small, nor too large)

2. Supervised learning with tensor networks

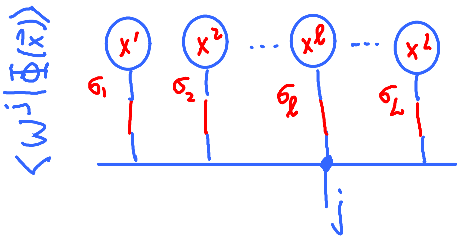
[Novikov2016], [Stoudenmire2017] with Schwab; [Maier2017] Bachelor thesis of David Maier

Goal: construct decision function $\vec{f}$ using a tensor network (here MPS);
train network using optimization techniques familiar from DMRG

Ansatz: $\vec{f} : \mathbb{I}^L \longmapsto \mathbb{I}^{10}$, (1)

$\vec{x} \longmapsto \vec{f}(\vec{x}) := \langle \vec{W} | \Phi(\vec{x}) \rangle$ (2)

image vector — predicted name



where right-hand side involves two separate maps:

'feature map' $\Phi : \vec{x} \mapsto |\Phi(\vec{x})\rangle$: encodes greyscale input data into L -leg MPS, $|\Phi(\vec{x})\rangle$ (3)

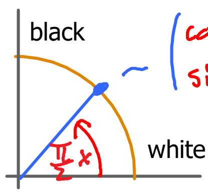
'weight vector' $\vec{W} : |\Phi(\vec{x})\rangle \mapsto f^j(\vec{x}) := \langle W^j | \Phi(\vec{x}) \rangle$, $j = 0, \dots, 9$ (4)

converts feature map into predicted name via inner product with an L-leg MPS, $|W^j\rangle$

'predicted name': that label j for which f^j is maximal.

Feature map: encoding input data

map color range
(0,1) = (white, black)
to quarter-unit-circle,

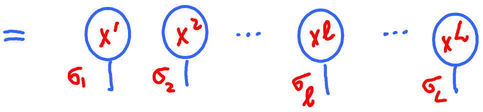


$\sim \begin{pmatrix} \cos \frac{\pi}{2} x \\ \sin \frac{\pi}{2} x \end{pmatrix} =: \begin{pmatrix} \phi_+(x) \\ \phi_-(x) \end{pmatrix} =: |\phi(x)\rangle$ (5)

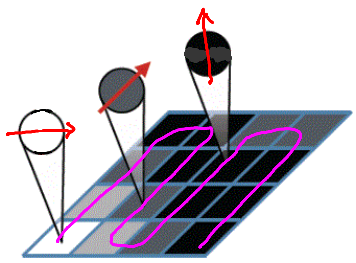
so that $\langle \phi(x') | \phi(x) \rangle = \sum_{\sigma = \pm} \phi_\sigma(x') \, \phi_\sigma(x) = \begin{cases} 1 & \text{if } x \simeq x' \\ 0 & \text{if } x \simeq \text{white}, \quad x' \simeq \text{black} \end{cases}$ (6)

Choose 'snake-ordering' of pixels,
and encode image in a product state MPS: $(d = 2)$

$|\Phi(\vec{x})\rangle = |\phi(x^1)\rangle \otimes |\phi(x^2)\rangle \otimes \dots \otimes |\phi(x^L)\rangle$ (7)



(8)



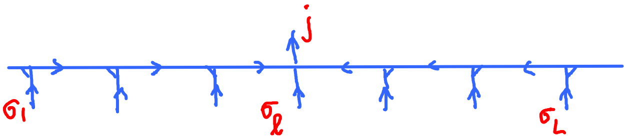
This construction for $|\Phi(\vec{x})\rangle$ is not unique. Other constructions are possible, provided that

$\langle \Phi(\vec{x}') | \Phi(\vec{x}) \rangle$ is a smooth and slowly varying function of $\vec{x}$ and $\vec{x}'$

which induces a 'distance matrix' in feature space which tends to cluster similar images together.

Weight vector: encoding pattern recognition

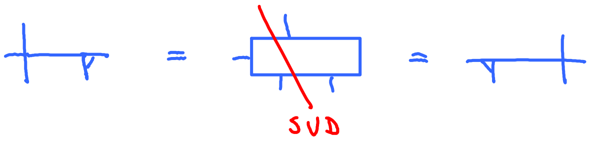
$|W^j\rangle$ = L-leg MPS =



$= |\vec\sigma\rangle A^{\sigma_1} A^{\sigma_2} \dots M^{\sigma_\ell, j} \dots B^{\sigma_{L-1}} B^{\sigma_L}$

Left-normalized A's, right-normalized B's, sandwiching a 4-leg tensor, $M^{\alpha\sigma_\ell\beta, j}$, at site ℓ

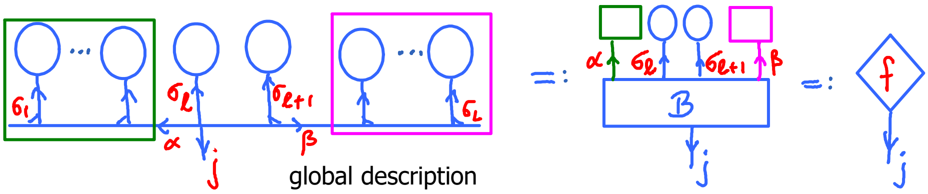
Top leg can be moved around:



Decision function:

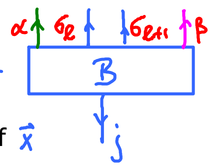
$\vec f(x)$, with components:

$f^j(\vec x) \overset{(4)}{=} \langle W^j | \Phi(\vec x)\rangle =$



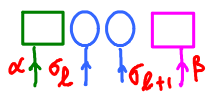
global description

$:= \langle B^j | \tilde\Phi(\vec x)\rangle$, with $\langle B^j| =$



independent of $\vec x$

, $|\tilde\Phi(x)\rangle =$



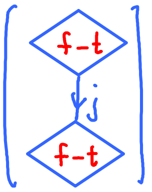
independent of j

Note: all x-dependence resides in $|\tilde\Phi(\vec x)\rangle$, all j-dependence in $|B^j\rangle$.

Location of 'central site' can be shifted (e.g. during sweeping).

Cost function

$$C = \sum_{n=1}^N \underbrace{(\vec f_n - \vec t_n)^2}_{\sum_j (f_n^j - t_n^j)^2} = \sum_{n=1}^N \Big[\ \Big]_n \quad , \quad \vec f_n := \vec f(\vec x_n)$$



evaluated at $\vec x_n$

For given set of training data $\{\vec x_n, \vec t_n \,|\, n = 1, \dots, N\}$, minimize C w.r.t. $\langle \vec W|$, or equivalently, $\langle \vec B|$.

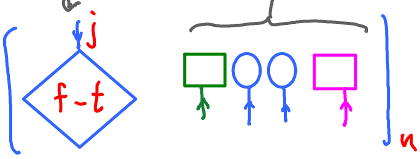
Minimize using gradient steepest descent. Compute the gradient:

$$|\nabla B^j\rangle := \frac{\partial C}{\partial B^j} := 2\sum_{n=1}^N (f_n^j - t_n^j)\frac{\partial f_n^j}{\partial B^j} = 2\sum_{n=1}^N (f_n^j - t_n^j)\, |\tilde\Phi(\vec x_n)\rangle$$

sum over training set

differs from one image to the next

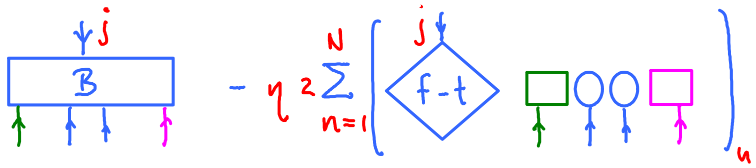
$= 2\sum_{n=1}^N \Big[\ \Big]_n$



Then update the MPS:

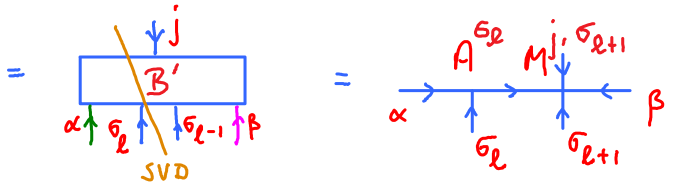
Then update the MPS:

$|B'^j\rangle = |B^j\rangle - \eta\, |\nabla B^j\rangle =$

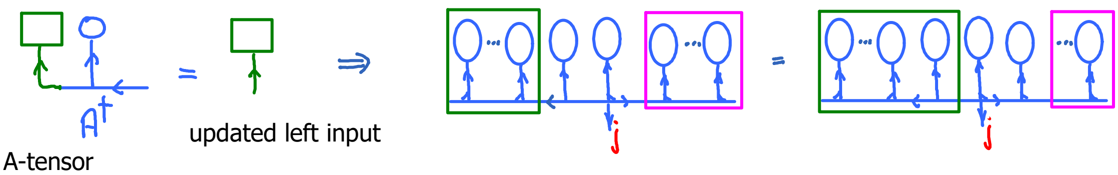


learning rate
(must be chosen very carefully!)

Advance to next site:



Update training input:



Sweep back and forth until A-tensors no longer change -- then 'training of network' is complete.

Comments

Costs: $\mathcal{O}(d^3 D^3 \cdot N \cdot L \cdot 10)$

d : physical bond dimension (here: 2)

N : number of training images

D : MPS bond dimension (free parameter)

L : number of pixels per image

Once network has been trained, prediction of a new image x proceeds simply via

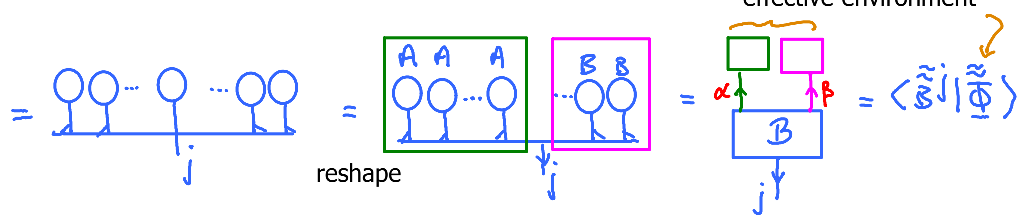
$f^j(\vec{x}) = \langle W^j | \Phi(\vec{x})\rangle$, predicted name is the j yielding maximal f^j

MNIST test:

- 28 x 28 was coarse-grained to 14 x 14 (to save resources)
- at most 5 sweeps were needed before training converges
- bond dimension $D = 10 \Rightarrow$ 5% error rate
  $20 \Rightarrow$ 2% error rate
  $120 \Rightarrow$ 0.97% error rate

Implicit feature selection - where does learning happen?

$f^j = \langle W^j | \Phi \rangle =$



$= \langle \tilde{B}^j | \tilde{\tilde{\Phi}} \rangle$

- $|\tilde{\tilde{\Phi}}\rangle$ is projection of $|\Phi\rangle$ onto space spanned by orthonormal basis, encoded in $\langle \tilde{\tilde{\xi}}_j|$

has just D^2 components lives in space of dimension 2^L lives in space of dimension D^2

- So, training an MPS model uncovers relatively small set of features, and simultaneously trains decision function using only those features.

- 'Feature selection' occurs when computing SVD: basis elements which do not contribute optimally to bond tensors are discarded

Future prospects

- try tensor networks that are designed for 2D (PEPS, TRG, MERA,)

- try other sampling schemes
- incorporate symmetries (if data set is 'invariant' under translations, rotations)
- 'unsupervised learning' with tensor networks
- ...