Centre for Quantum Technologies

NUS National University of Singapore

Tensor network and machine learning

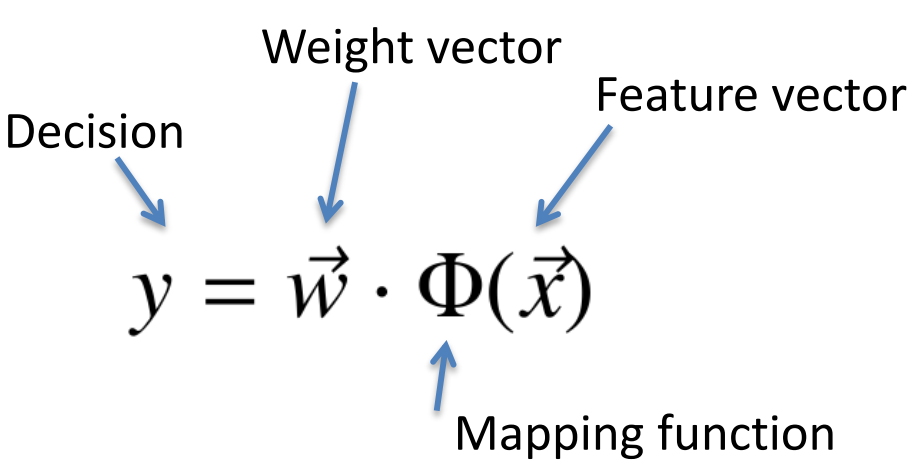
Supervised Learning With Quantum-Inspired Tensor Networks
arXiv:1605.05775
E. Stoudenmire and D. Schwab

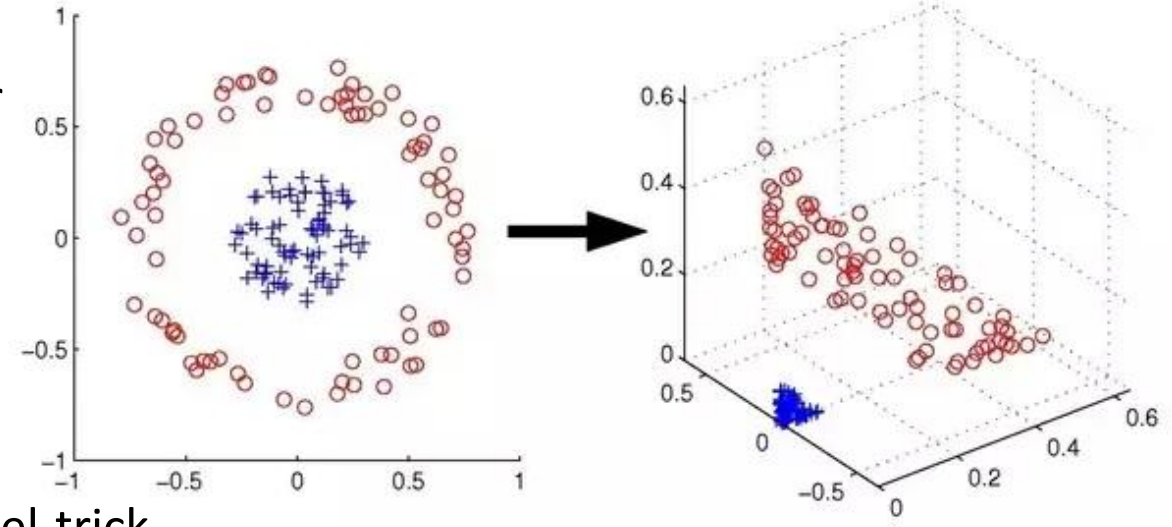
Tobias Haug

05.10.2018

Machine learning

- Classification: Find (complicated) decision boundary
- One solution: Map to a linear problem in higher dimensions
- High dimension scale badly→Use tricks
  - Use good mapping function $\Phi(\mathbf{x})$
  - Kernel trick (minimizing powers of inner products corresponds to higher dimension)
  - (Shallow) Neuronal network representation, etc…
- Can classical methods for quantum many-body problems help?





https://www.quora.com/What-is-the-kernel-trick

Quantum states and tensors

- Number of many-body states scale exponentially, e.g. N spins as 2^N
- Each state is represented by a entry in the state vector

For N two-level systems

$$y = (x_1, x_2, \cdots, x_N)$$

$$|\Psi_1\rangle = |0, 0, \cdots, 0, 0\rangle$$

$$|\Psi_2\rangle = |1, 0, \cdots, 0, 0\rangle$$

$$|\Psi_N\rangle = |1, 1, \cdots, 1, 1\rangle$$

- Scales badly….
- Instead: Describe state by multiplications of matrices (Matrix Product state)

$$|\psi\rangle = \sum_{\sigma_1,\ldots,\sigma_L} c_{\sigma_1\ldots\sigma_L} |\sigma_1, \ldots, \sigma_L\rangle \longrightarrow |\Psi\rangle = \sum_{\sigma_1,\sigma_2,\cdots,\sigma_N} \mathrm{tr}\left(A_{\sigma_1} A_{\sigma_2} \cdots A_{\sigma_N}\right) |\sigma_1, \sigma_2, \cdots, \sigma_N\rangle$$

Vector size: 2^N

Number of matrices: 2*N

Multiply matrices together, then take trace

- Matrices could be also generalized to tensors (“higher dimensional matrices”)

Example for MPS

$$|\Downarrow\rangle = |0, 0, \cdots, 0, 0\rangle \qquad |\Uparrow\rangle = |1, 1, \cdots, 1, 1\rangle$$

- First Example: $|\Psi\rangle = |\Downarrow\rangle + |\Uparrow\rangle$

$$A_0 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$|\Psi\rangle = \sum_{\sigma_1, \sigma_2, \cdots, \sigma_N} \mathrm{tr}\left(A_{\sigma_1} A_{\sigma_2} \cdots A_{\sigma_N}\right) |\sigma_1, \sigma_2, \cdots, \sigma_N\rangle$$

- Second: $|\Psi\rangle = \sum_{\text{all permutations of 1}} |0, 0, \cdots, 0, 1, 0, \cdots, 0, 0\rangle$

$$A_1 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad A_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \Phi_{\mathrm{L}} = \begin{pmatrix} 1 & 0 \end{pmatrix} \qquad \Phi_{\mathrm{R}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$|\Psi\rangle = \sum_{\sigma_1, \sigma_2, \cdots, \sigma_N} \vec{\Phi}_{\mathrm{L}} A_{\sigma_1} A_{\sigma_2} \cdots A_{\sigma_N} \vec{\Phi}_{\mathrm{R}} |\sigma_1, \sigma_2, \cdots, \sigma_N\rangle$$

How to construct MPS

- SVD: $M_{x,y} = \sum_a U_{x,a} S_{a,a} V^{\dagger}_{a,y}$ ***S*** is a diagonal matrix, higher values contribute more → Approximate ***M*** by throwing away small entries

$$|\Psi\rangle = \sum_{\sigma_1,\sigma_2,\cdots,\sigma_N} c_{\sigma_1\sigma_2\cdots\sigma_N} |\sigma_1\sigma_2\cdots\sigma_N\rangle \longrightarrow c_{\sigma_1\sigma_2\cdots\sigma_N} = \phi_{\sigma_1,(\sigma_2\cdots\sigma_N)}$$

$$c_{\sigma_1\sigma_2\cdots\sigma_N} = \phi_{\sigma_1,(\sigma_2\cdots\sigma_N)} = \sum_{a_1}^{d} U_{\sigma_1,a_1} S_{a_1,a_1} (V^{\dagger})_{a_1,(\sigma_2\cdots\sigma_N)} \equiv \sum_{a_1}^{d} U_{\sigma_1,a_1} c_{a_1\sigma_2\cdots\sigma_N}$$

$$c_{\sigma_1\sigma_2\cdots\sigma_N} = \sum_{a_1}^{d} U_{\sigma_1,a_1} c_{a_1\sigma_2\cdots\sigma_N} = \sum_{a_1}^{d} A^{\sigma_1}_{a_1} \phi_{(a_1\sigma_2),(\sigma_3\cdots\sigma_N)}$$

$$c_{\sigma_1\sigma_2\cdots\sigma_N} = \sum_{a_1}^{d} \sum_{a_2}^{d^2} A^{\sigma_1}_{a_1} U_{(a_1\sigma_2),a_2} S_{a_2,a_2} (V^{\dagger})_{a_2,(\sigma_3\cdots\sigma_N)} \equiv \sum_{a_1}^{d} \sum_{a_2}^{d^2} A^{\sigma_1}_{a_1} A^{\sigma_2}_{a_1,a_2} c_{a_2\sigma_3\cdots\sigma_N}$$

$$|\Psi\rangle = \sum_{\sigma_1,\sigma_2,\cdots,\sigma_N} A_{\sigma_1} A_{\sigma_2} \cdots A_{\sigma_N} |\sigma_1\sigma_2\cdots\sigma_N\rangle$$

How does it help?

- Of course, this method does not reduce complexity, as the matrices will in general increase in dimension with N and the complexity of the wavefunction
- But: Allows for efficient approximation, converges to exact state
- The **entanglement S** of the **ground state** of **gapped** (energy of ground state is non-degenerate) and **local** (there is no long-range interactions) **Hamiltonians** is proportional to the **boundary of the D-dimensional space**

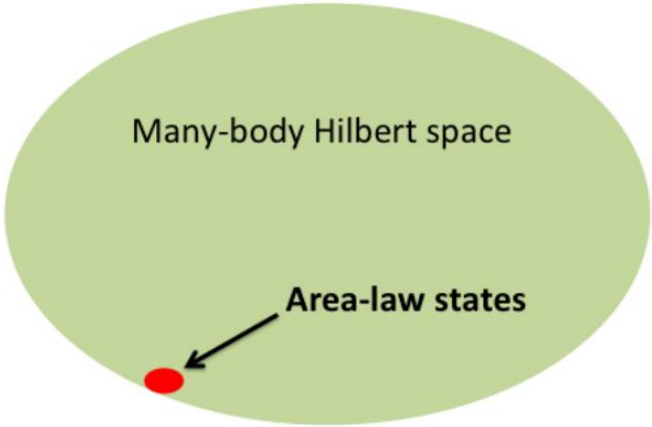
→ $\mathbf{S \sim N^{D-1}}$

- Entanglement of ground state of 1D Hamiltonian is independent of system size→Complexity of wavefunction does not increase with size!

  (e.g. my recent publication quasi-1D $\dim(H)=10^{65}$)
- MPS is optimal representation in 1D. For 2D and more finding optimal representation is NP-hard (but there are generalized MPS for 2D and 3D called PEPS, MERA, etc…)

Area law

- MPS works efficiently in a small sub space of the Hilbert space
- Fails for highly entangled states (e.g. thermal, non-equilibrium)

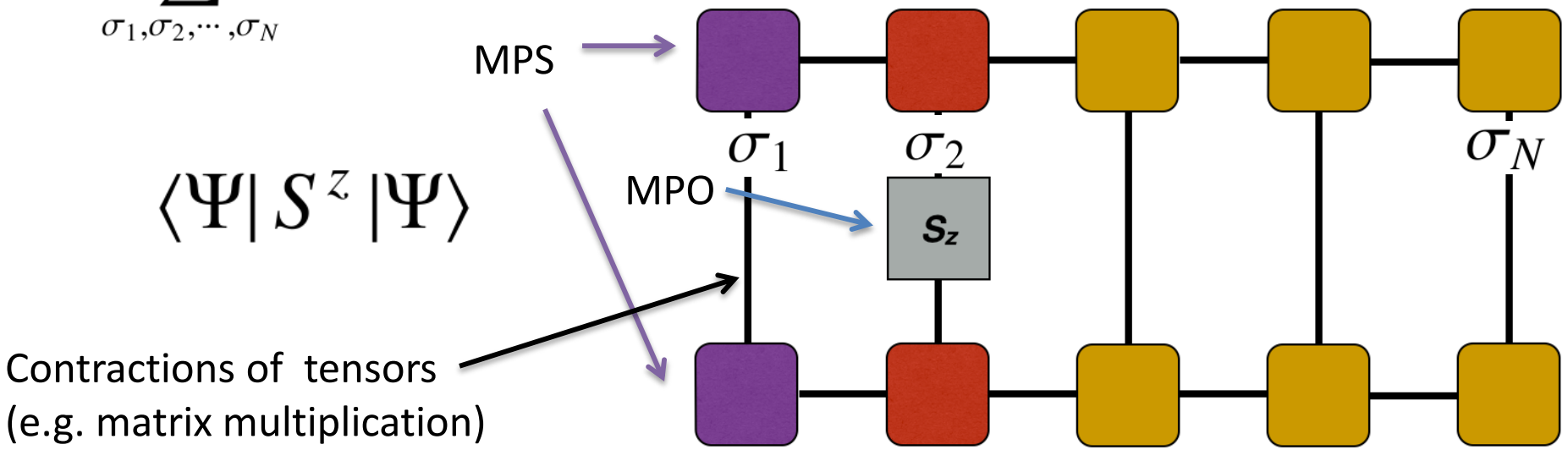


How actually used

- "Local" operators can be expressed efficiently as Matrix-product operators (MPO)
- Initialize random MPS, go through chain and apply Hamiltonian MPO locally to optimize A towards ground state
- Truncate size of matrix A (bond size) after each step
- Efficient, as MPO acts locally, involving only a few *A* elements
- Everything can be neatly represented using tensors, no need to go to "Hilbert space representation"

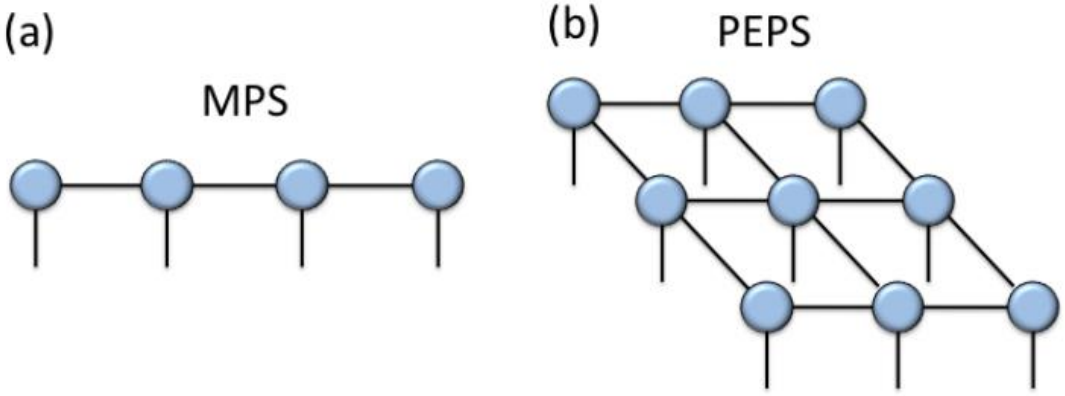
$$|\Psi\rangle = \sum_{\sigma_1,\sigma_2,\cdots,\sigma_N} A_{\sigma_1} A_{\sigma_2} \cdots A_{\sigma_N} |\sigma_1 \sigma_2 \cdots \sigma_N\rangle$$

$$\langle \Psi | S^z | \Psi \rangle$$



Some other notes

- Concept can be extended to find good representations beyond 1D (PEPS, etc.), best case entanglement grows as $\mathbf{S \sim N^{D-1}}$
- Finding optimal representation in general is NP hard
- DMRG is distinct from variatonal Monte-Carlo: It converges towards the exact wavefunction
- Restricted Boltzmann approach is variational Monte-Carlo: Assume ansatz, then optimize



Supervised Learning With Quantum-Inspired Tensor Networks

E. Miles Stoudenmire[1,2] and David J. Schwab[3]

[1] *Perimeter Institute for Theoretical Physics, Waterloo, Ontario, N2L 2Y5, Canada*
[2] *Department of Physics and Astronomy, University of California, Irvine, CA 92697-4575 USA*
[3] *Dept. of Physics, Northwestern University, Evanston, IL*

(Dated: May 22, 2017)

Tensor networks are efficient representations of high-dimensional tensors which have been very successful for physics and mathematics applications. We demonstrate how algorithms for optimizing such networks can be adapted to supervised learning tasks by using matrix product states (tensor trains) to parameterize models for classifying images. For the MNIST data set we obtain less than 1% test set classification error. We discuss how the tensor network form imparts additional structure to the learned model and suggest a possible generative interpretation.

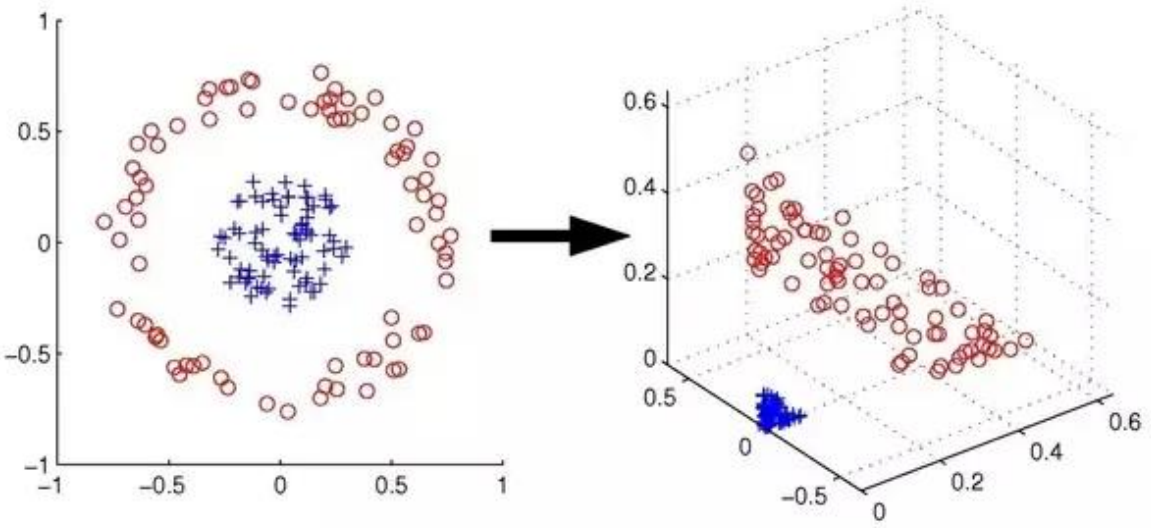
- Classification of a large input vector x with a weight matrix W, to find decision function f of label l

$$f^{\ell}(\mathbf{x}) = W^{\ell} \cdot \Phi(\mathbf{x})$$

- $\Phi(\mathbf{x})$ maps N-dimensional x to some higher dimension
- Train W such that it fits labels (e.g. identify images for cats and dogs

Whats this problem again?

- Find non-linear decision boundary by mapping problem to higher dimension (Support Vector machine, Neuronal network…)
- High dimension scale badly→Use tricks to deal with high dimension
  - Use good mapping $\Phi(\mathbf{x})$
  - Kernel trick (use powers of inner products)
  - Neuronal network representation, etc…



https://www.quora.com/What-is-the-kernel-trick

Example

Local dimension (here d=2)

whiteness

blackness

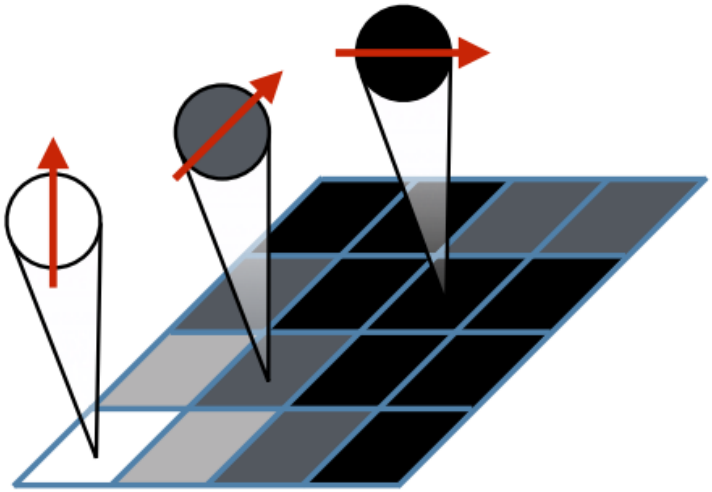
$$\phi^{s_j}(x_j) = \left[\cos\left(\frac{\pi}{2}x_j\right),\ \sin\left(\frac{\pi}{2}x_j\right)\right]$$

- x_j:color of pixel j (0…1)
- Map to a "qubit"
- Map N pixels to d^N dimensional space

$$\Phi^{s_1 s_2 \cdots s_N}(\mathbf{x}) = \phi^{s_1}(x_1) \otimes \phi^{s_2}(x_2) \otimes \cdots \phi^{s_N}(x_N)$$

- This paper: 196 pixel, dim=10^{59}

  (number atoms sun: 10^{57})



$$f^{\ell}(\mathbf{x}) = W^{\ell} \cdot \Phi(\mathbf{x})$$

- Represent weight vector as MPS

$$W^{\ell}_{s_1 s_2 \cdots s_N} = \sum_{\{\alpha\}} A^{\alpha_1}_{s_1} A^{\alpha_1 \alpha_2}_{s_2} \cdots A^{\ell;\alpha_j \alpha_{j+1}}_{s_j} \cdots A^{\alpha_{N-1}}_{s_N}$$

- 2D image NIST dataset
- Order image into 1D chain
- Algorithm to optimize:
  - take 2 neighboring A
  - project input data onto local representation of that neighborhood (information about the rest of the chain is stored into $\tilde{\Phi}_n$
  - use cost function to optimize weight matrices locally via gradient, then truncate
  - Move on to next A, adapt projection

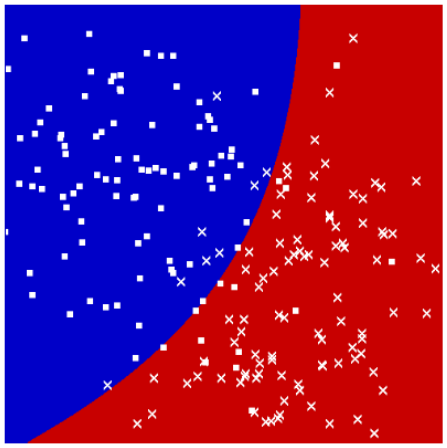
$$C = \frac{1}{2}\sum_{n=1}^{N_T}\sum_{\ell}(f^{\ell}(\mathbf{x}_n) - \delta^{\ell}_{L_n})^2$$

$$d^3\, m^3\, N\, N_L\, N_T$$

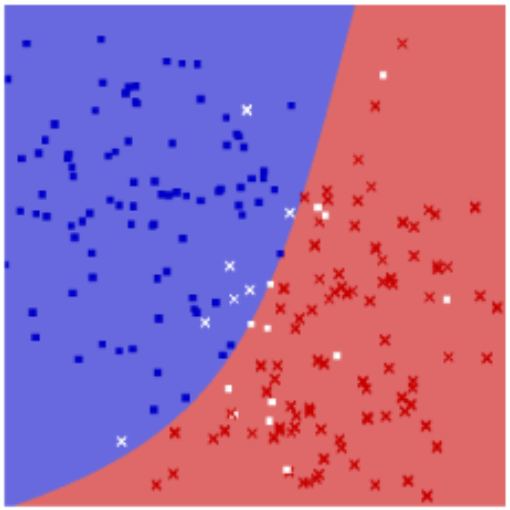
Test error rates also decreased rapidly with the maximum MPS bond dimension m. For $m = 10$ we found both a training and test error of about 5%; for $m = 20$ the error dropped to only 2%. The largest bond dimension we tried was $m = 120$, where after three sweeps we obtained a test error of 0.97% (97 misclassified images out of the test set of 10,000 images); the training set error was 0.05% or 32 misclassified images.

Increasing local dimension

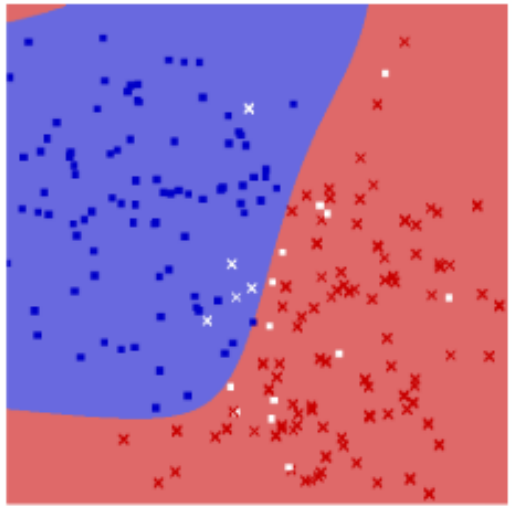
- 2 parameter toy model



d=2



d=3

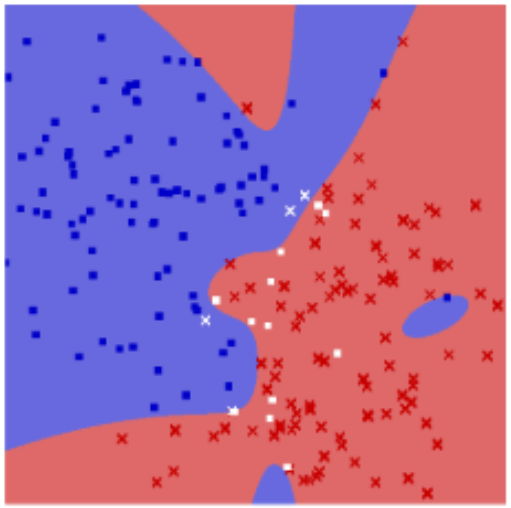


d=2

$$\phi^{s_j}(x_j) = \left[\cos\left(\frac{\pi}{2}x_j\right),\ \sin\left(\frac{\pi}{2}x_j\right)\right]$$

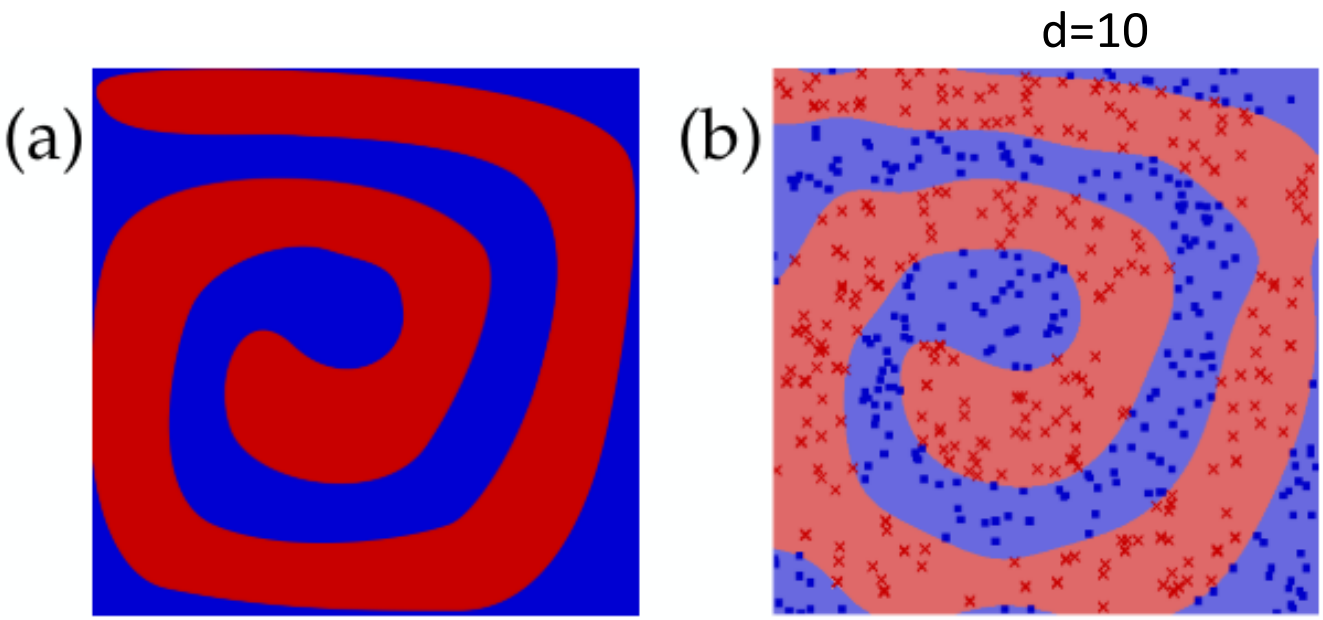
$$\Phi(x_1, x_2) = \phi^{s_1}(x_1) \otimes \phi^{s_2}(x_2)$$

d=6



Increasing local dimension

- 2 parameter toy model with non-linear distribution



Interpretation

$$W^{\ell}_{s_1 s_2 \cdots s_N} = \sum_{\{\alpha\}} U^{\alpha_1}_{s_1} \cdots U^{\alpha_i}_{\alpha_{i-1} s_i} C^{\ell}_{\alpha_i \alpha_{i+1}} V^{\alpha_{i+1}}_{s_{i+1} \alpha_{i+2}} \cdots V^{\alpha_{N-1}}_{s_N}$$

U, V "projectors" from SVD, C contains central info

- Three steps:
  1. Map feature vector x to d^N dimensional space $\Phi(\mathbf{x})$
  2. Contract $\Phi(\mathbf{x})$ with U, V, leaving only C → reduced mapping $\tilde{\Phi}(\mathbf{x})$ ~m^2
  3. Calculate f(x) with reduced mapping

- Learns both non-linear mapping and weight vector at the same time

Side note: Choosing a complete basis $\Phi(\mathbf{x})$ allows interpretation as wavefunction overlap $f^{\ell}(\mathbf{x}) = W^{\ell} \cdot \Phi(\mathbf{x})$

Centre for Quantum Technologies

NUS National University of Singapore

Neat references

- A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States, R. Orus
- Hand-waving and interpretive dance: an introductory course on tensor networks, Jacob C Bridgeman and Christopher T Chubb
- The density-matrix renormalization group in the age of matrix product states, U. Schollwoeck
- Supervised Learning With Quantum-Inspired Tensor Networks, arXiv:1605.05775, E. Stoudenmire and D. Schwab