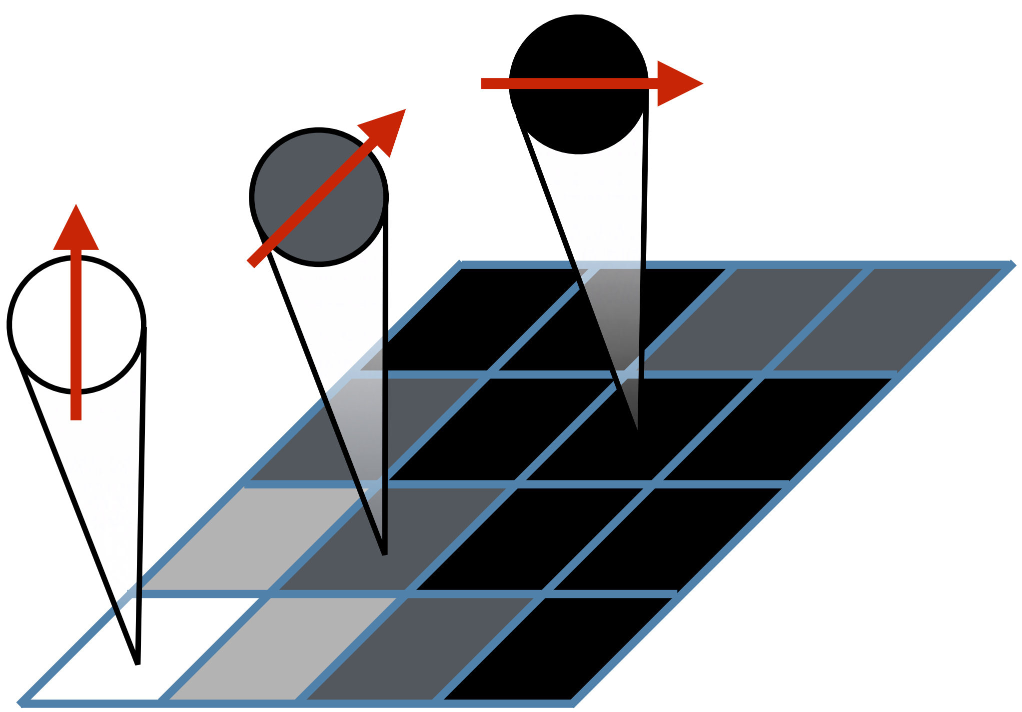
Tensor Networks and Applications



E.M. Stoudenmire

Apr 2018 - Sao Paulo

FLATIRON INSTITUTE

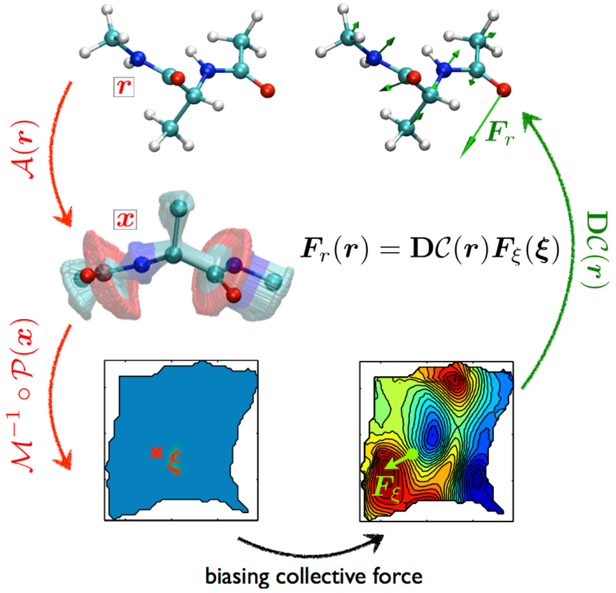
SIMONS FOUNDATION

Machine learning galvanizing industry & science

Language Processing

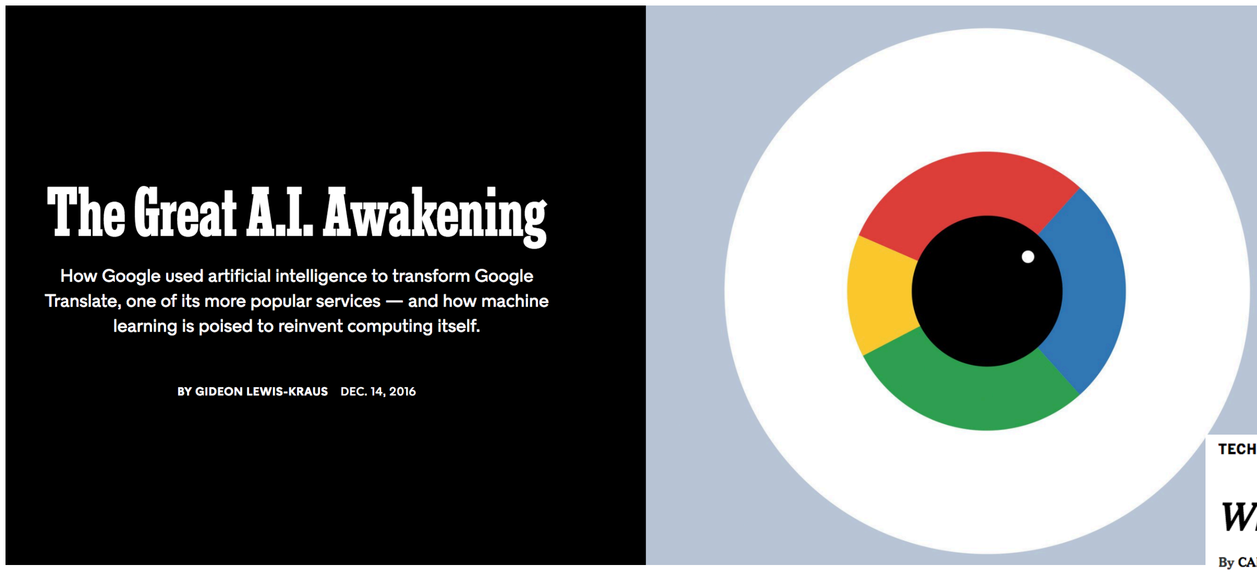
Self-driving cars

Medicine



Materials Science / Chemistry

Google rebranded a "machine learning first company"

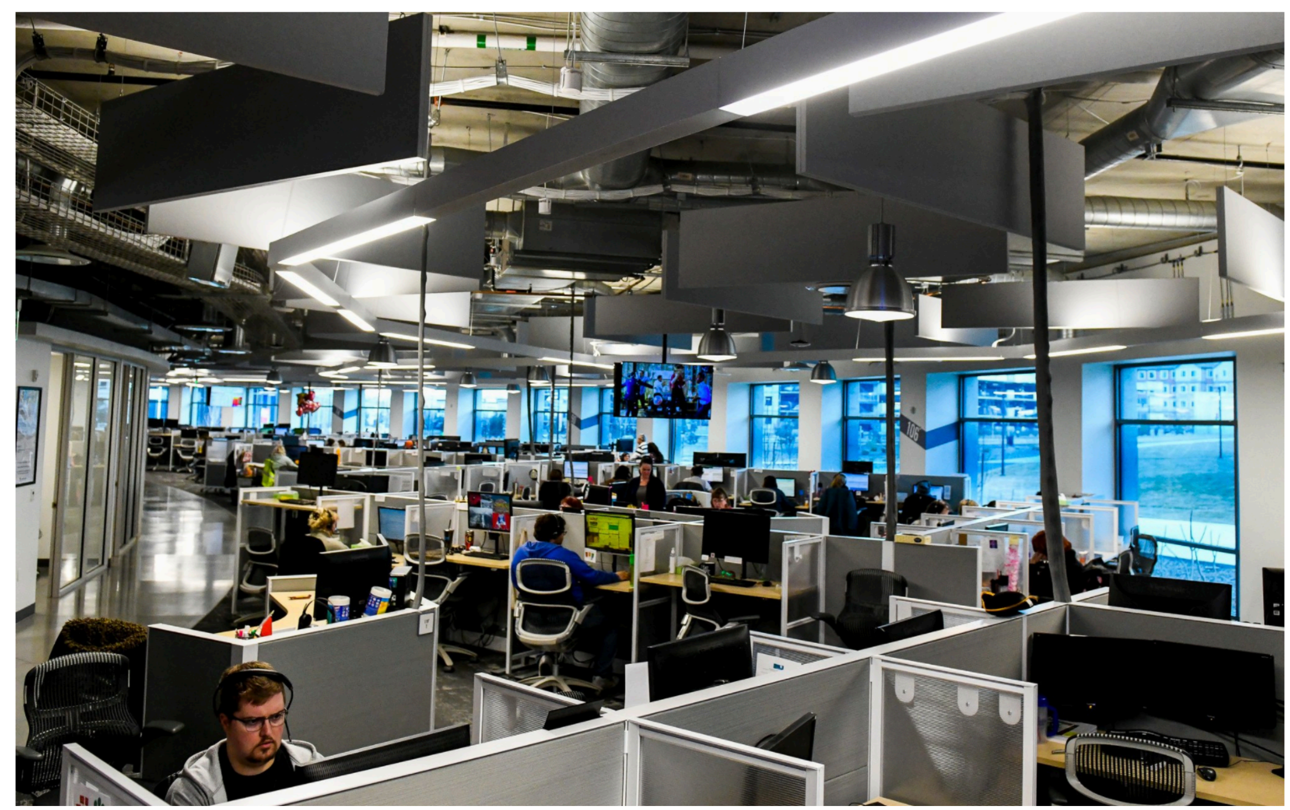


Neural nets replace linguistic approach to Google Translate

TECHNOLOGY

Why A.I. Researchers at Google Got Desks Next to the Boss

By CADE METZ FEB. 19, 2018



arXiv.org > quant-ph > arXiv:1802.06002

Quantum Physics

Classification with Quantum Neural Networks on Near Term Processors

Edward Farhi, Hartmut Neven

(Submitted on 16 Feb 2018)

Quantum machine learning

Examples of Machine Learning

Image recognition

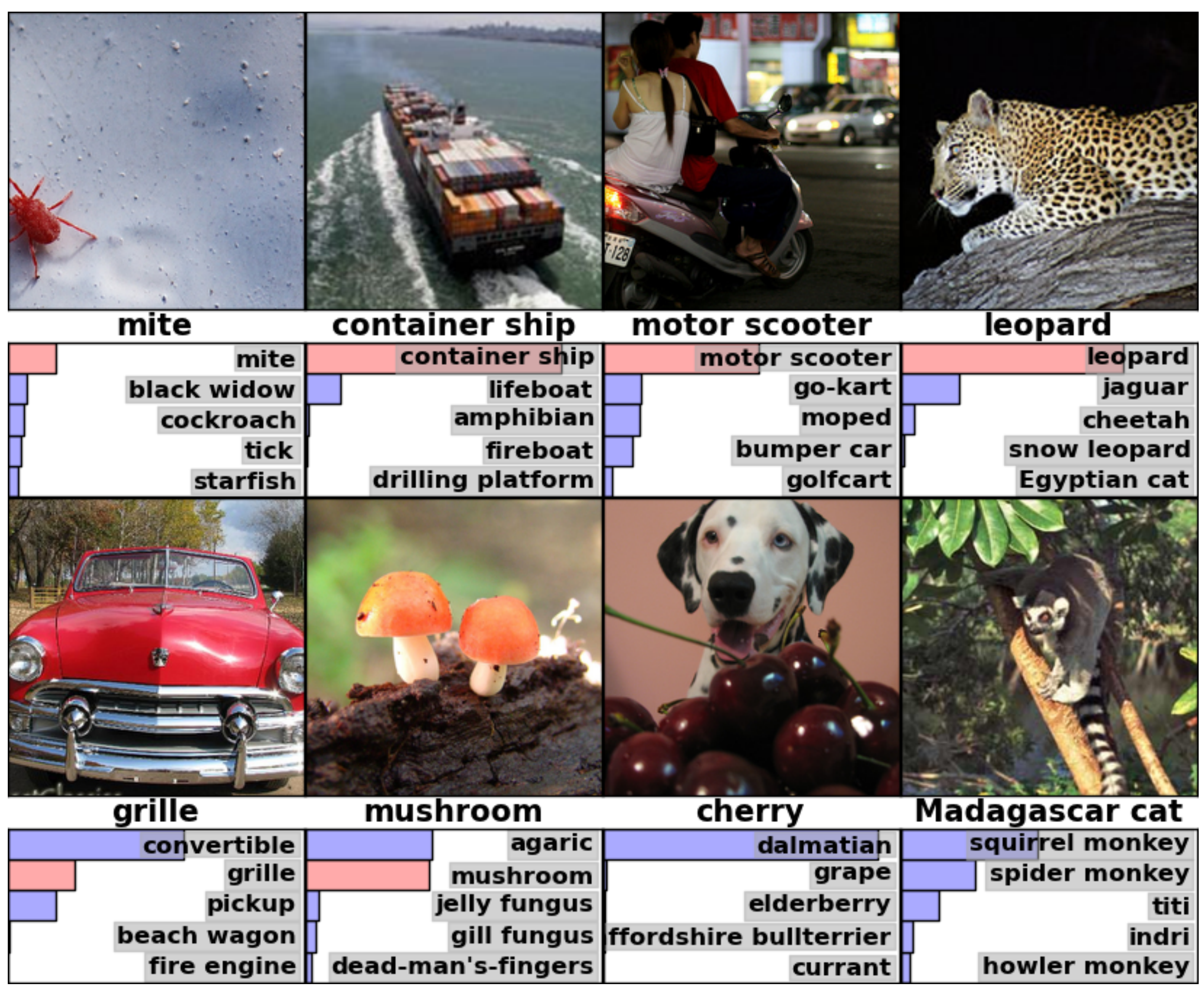
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

2012 paper that launched recent deep learning craze (20k citations)



ImageNet:

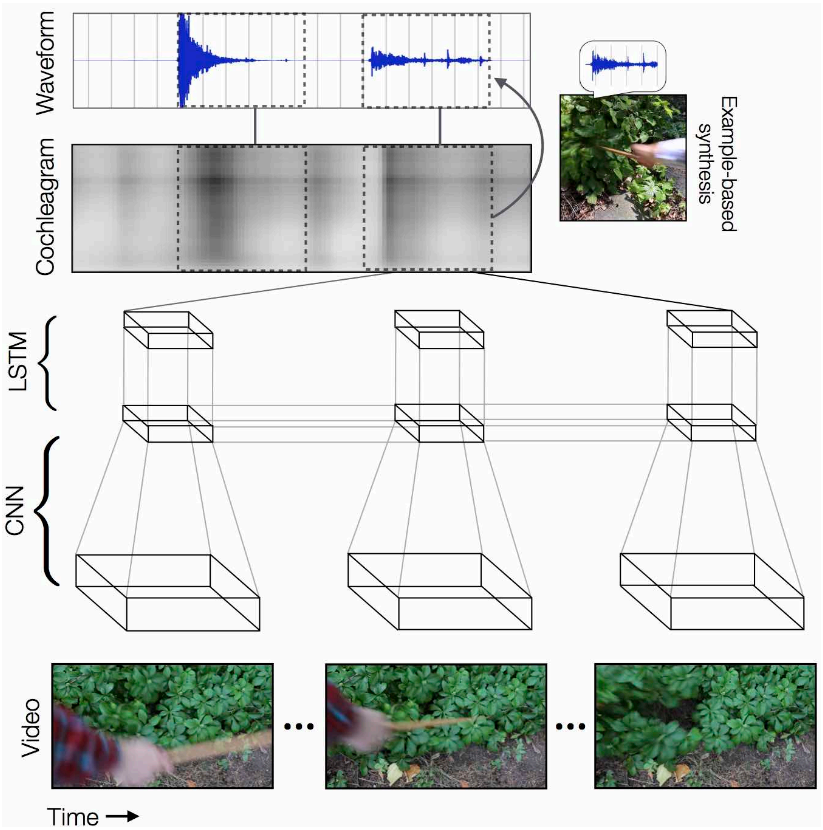
- 1.2 million training images (150k test)
- 1000 categories
- 15% neural net error
- 26% next best error

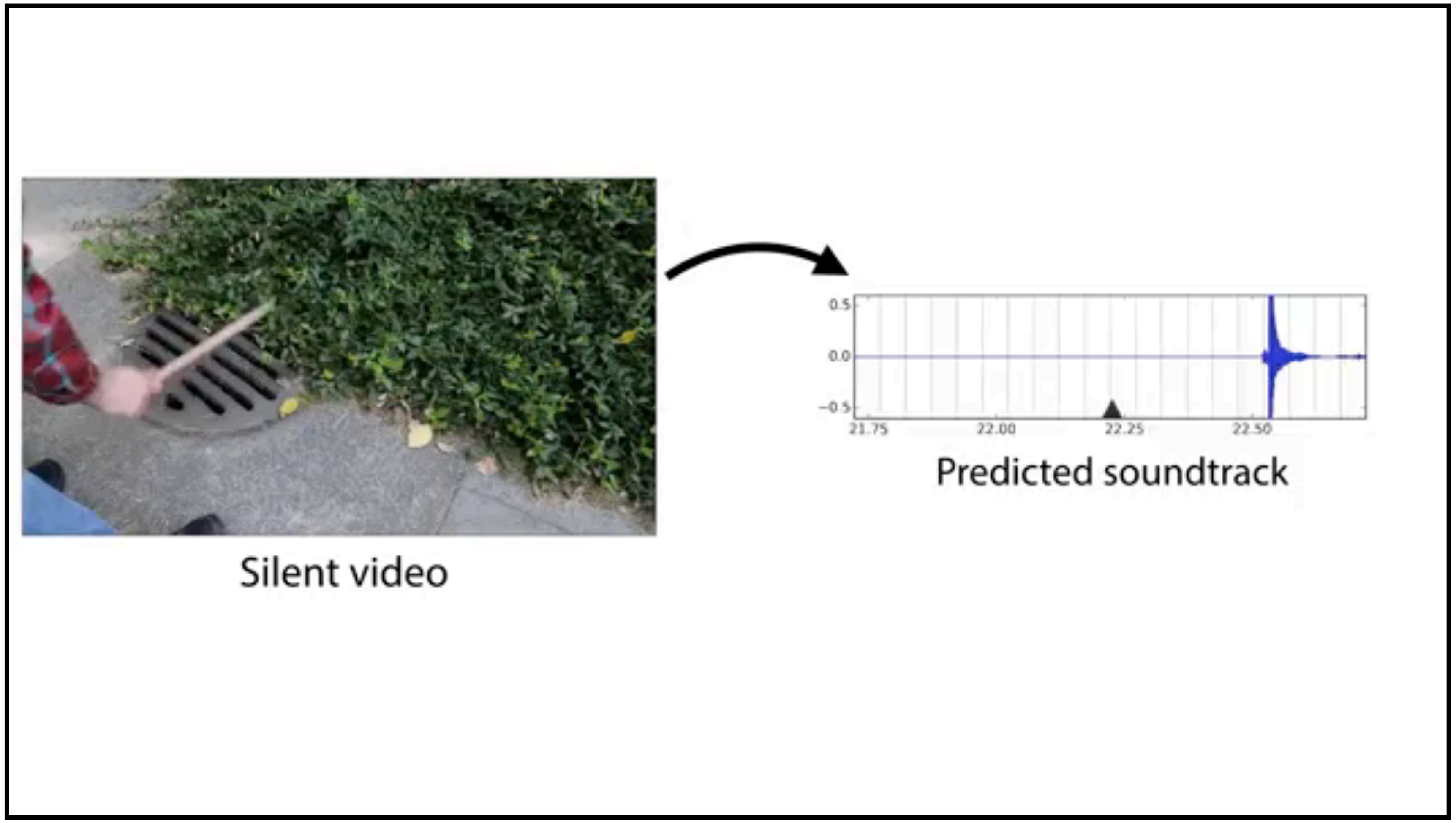
Sound prediction

Visually Indicated Sounds

Andrew Owens[1] Phillip Isola[2,1] Josh McDermott[1]
Antonio Torralba[1] Edward H. Adelson[1] William T. Freeman[1,3]

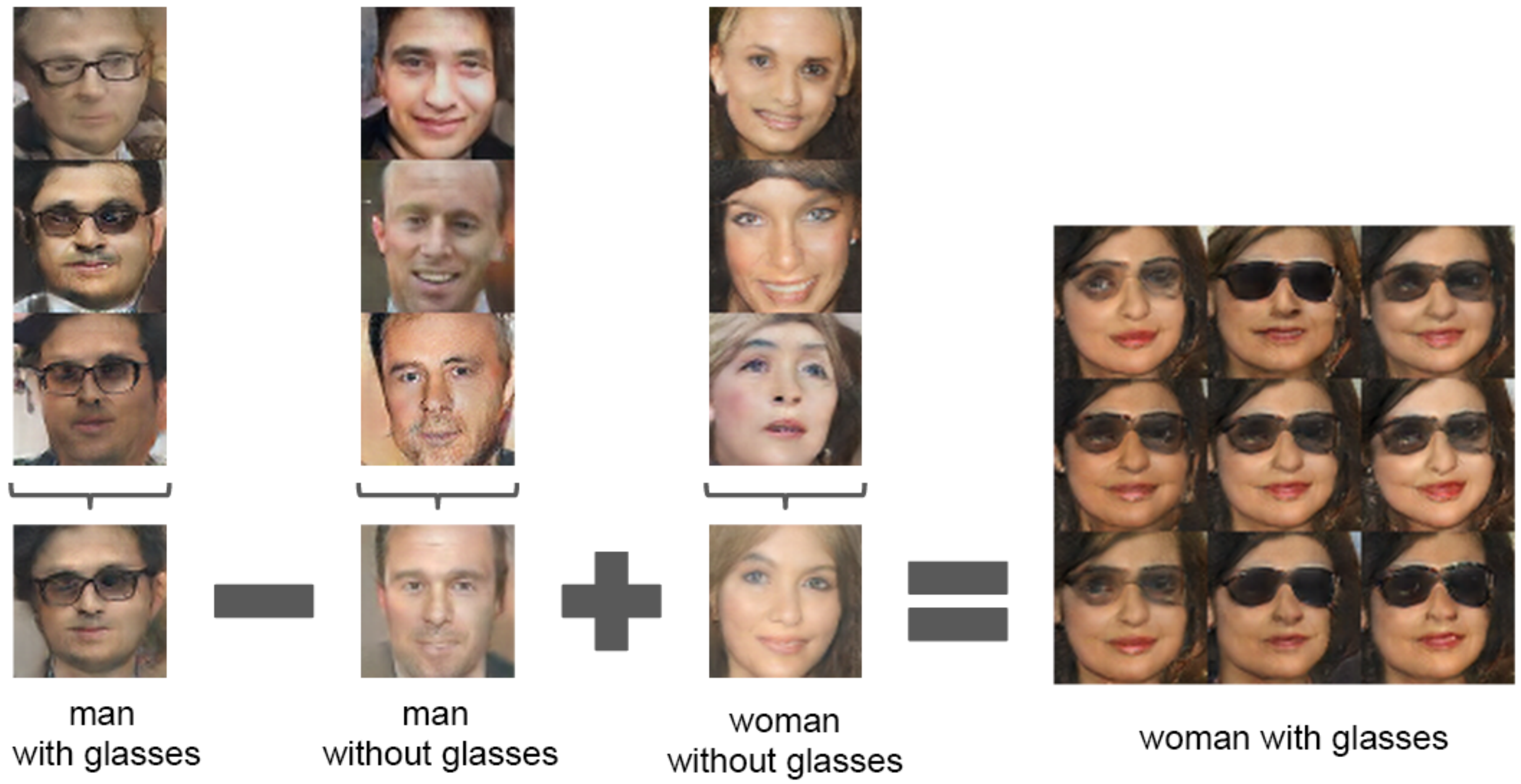
[1]MIT [2]U.C. Berkeley [3]Google Research





http://vis.csail.mit.edu

Image Generation



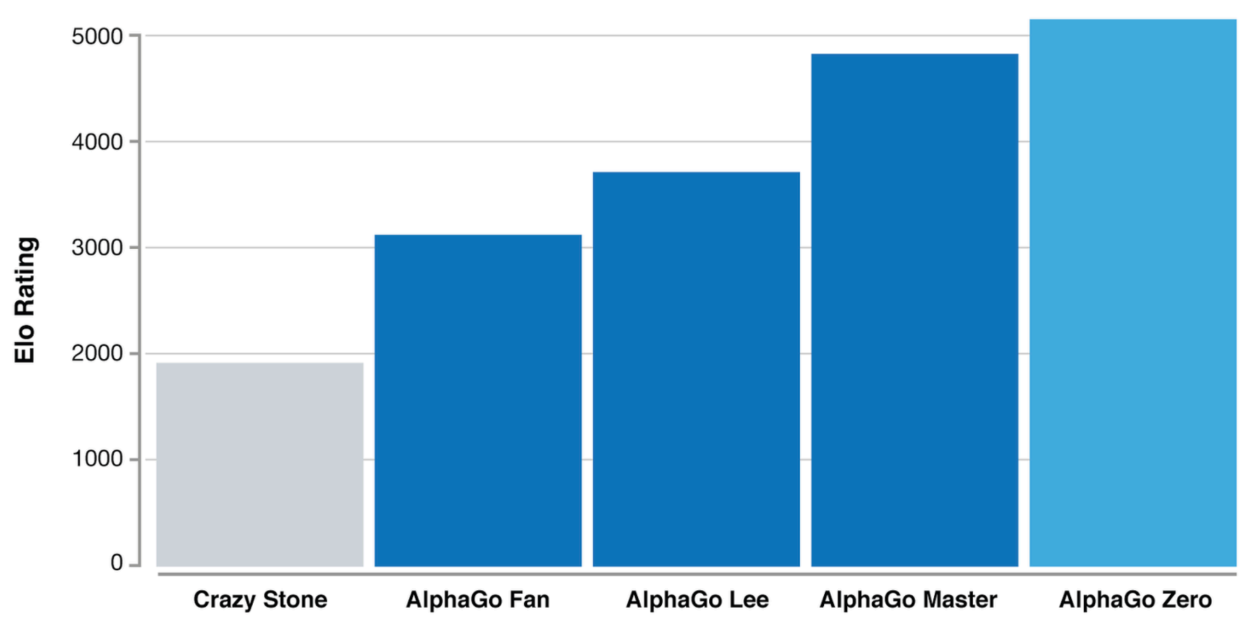
UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

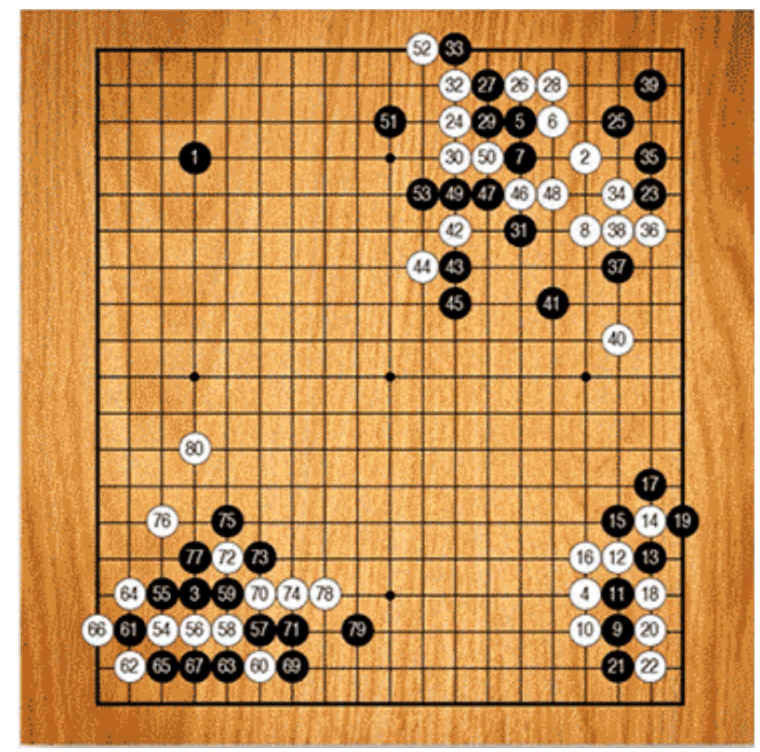
Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

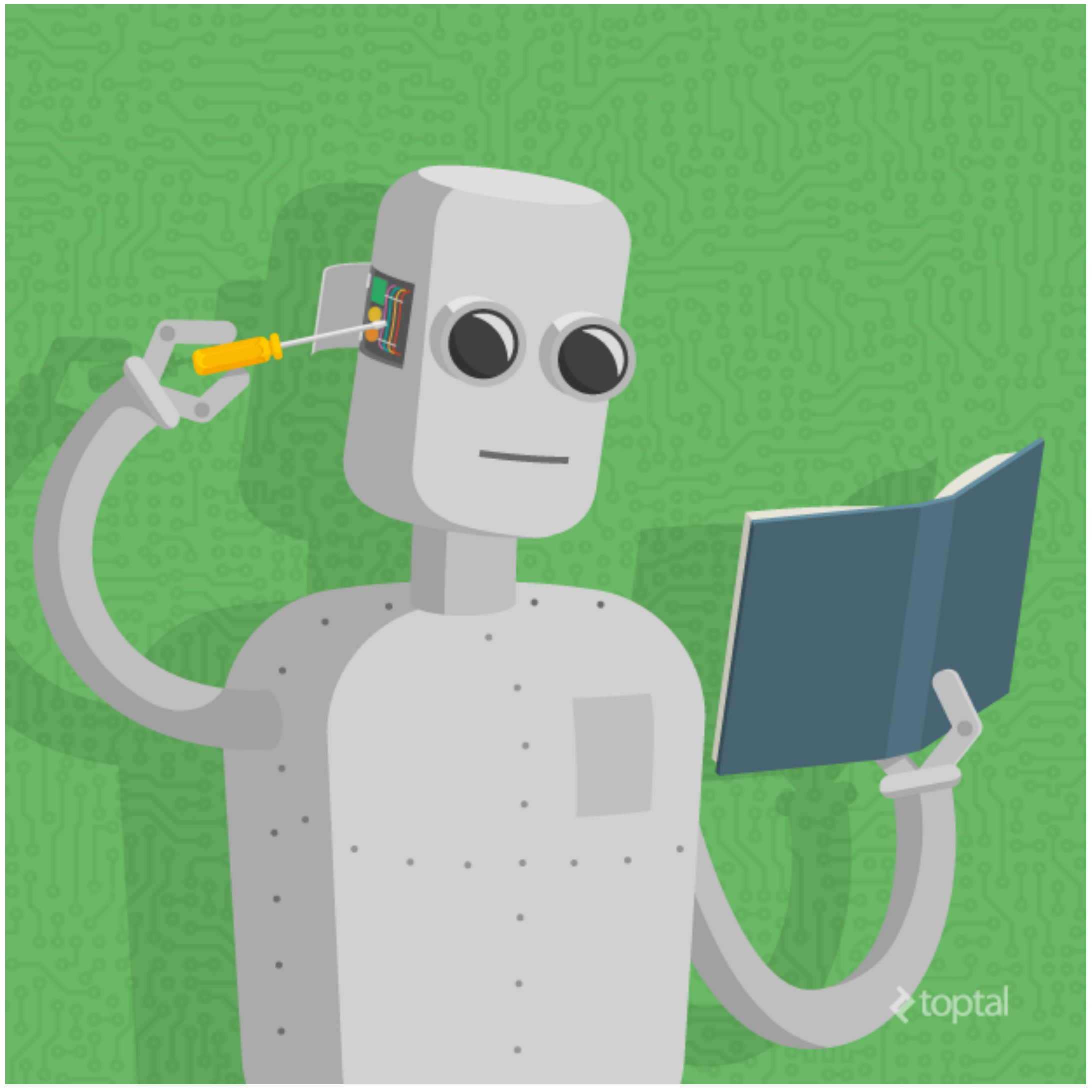
Success at tasks previously thought impossible

AlphaGo

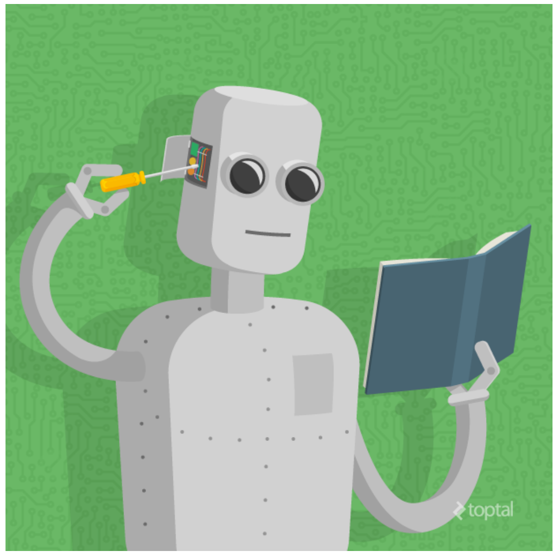




What is machine learning?



What is machine learning?



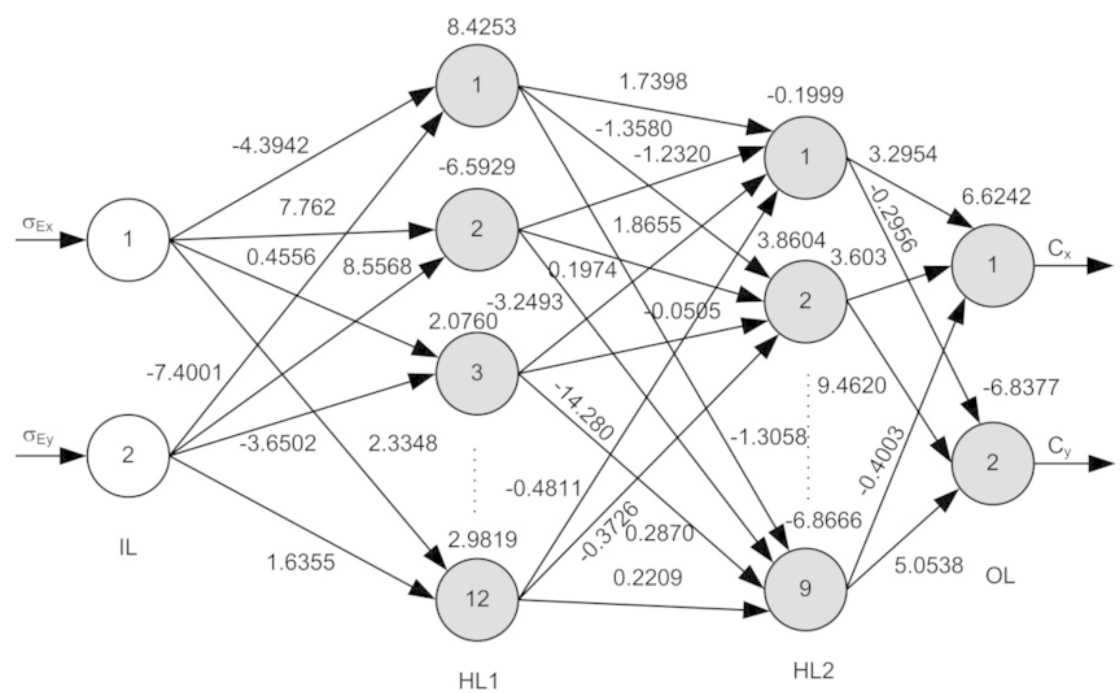
Data driven problem solving

Any system that, given more data, performs increasingly better at some task

Framework / philosophy, not single method

Software 1.0

Software 2.0



Andrej Karpathy Follow

Director of AI at Tesla. Previously Research Scientist at OpenAI and PhD student at Stanford. I like to train deep neural nets on large datasets.

Nov 11, 2017 · 7 min read

https://medium.com/@karpathy/software-2-0-a64152b37c35

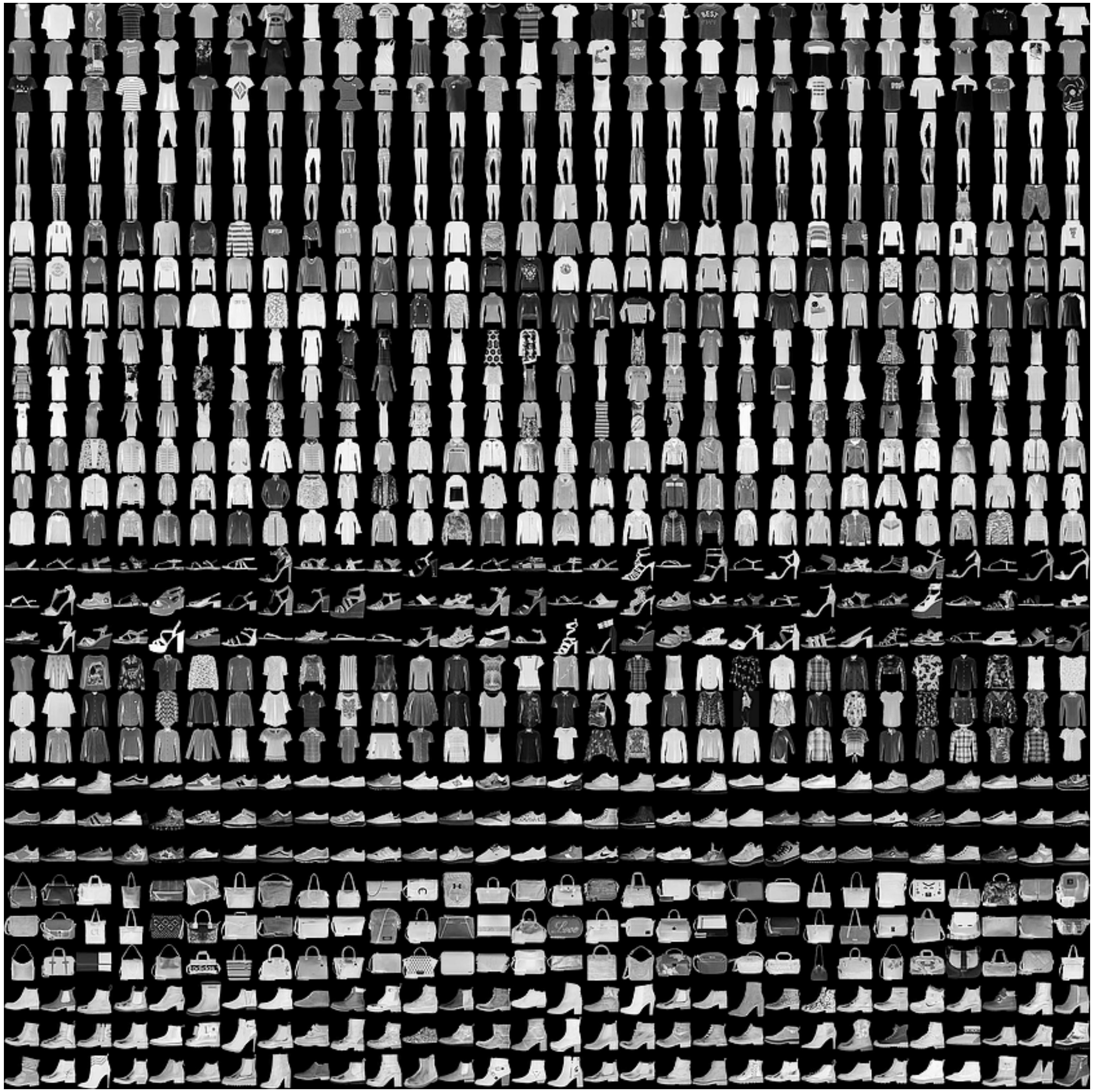
Basics of Machine Learning

Example of a Dataset – Fashion MNIST

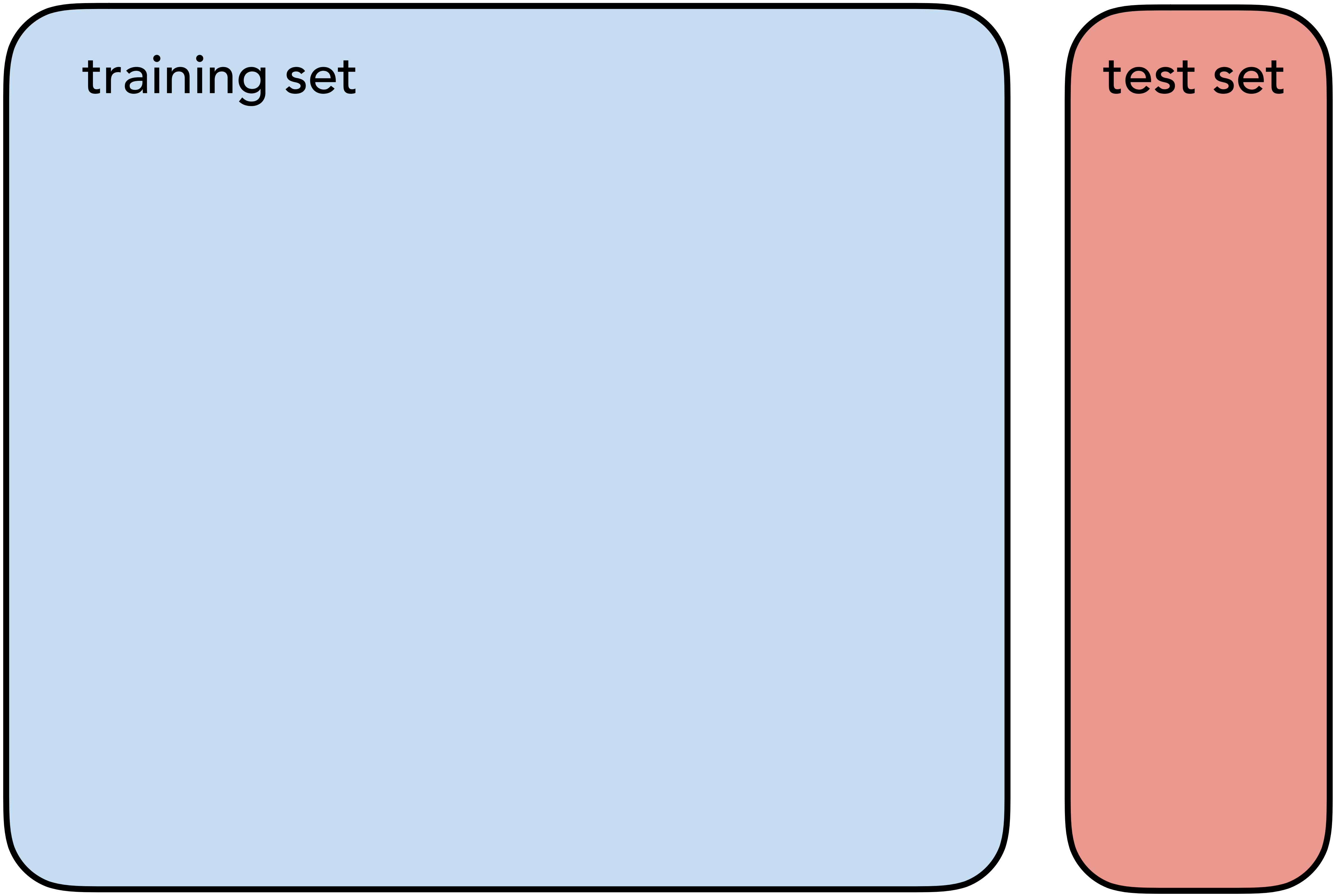
10 categories (labels)

28x28 grayscale

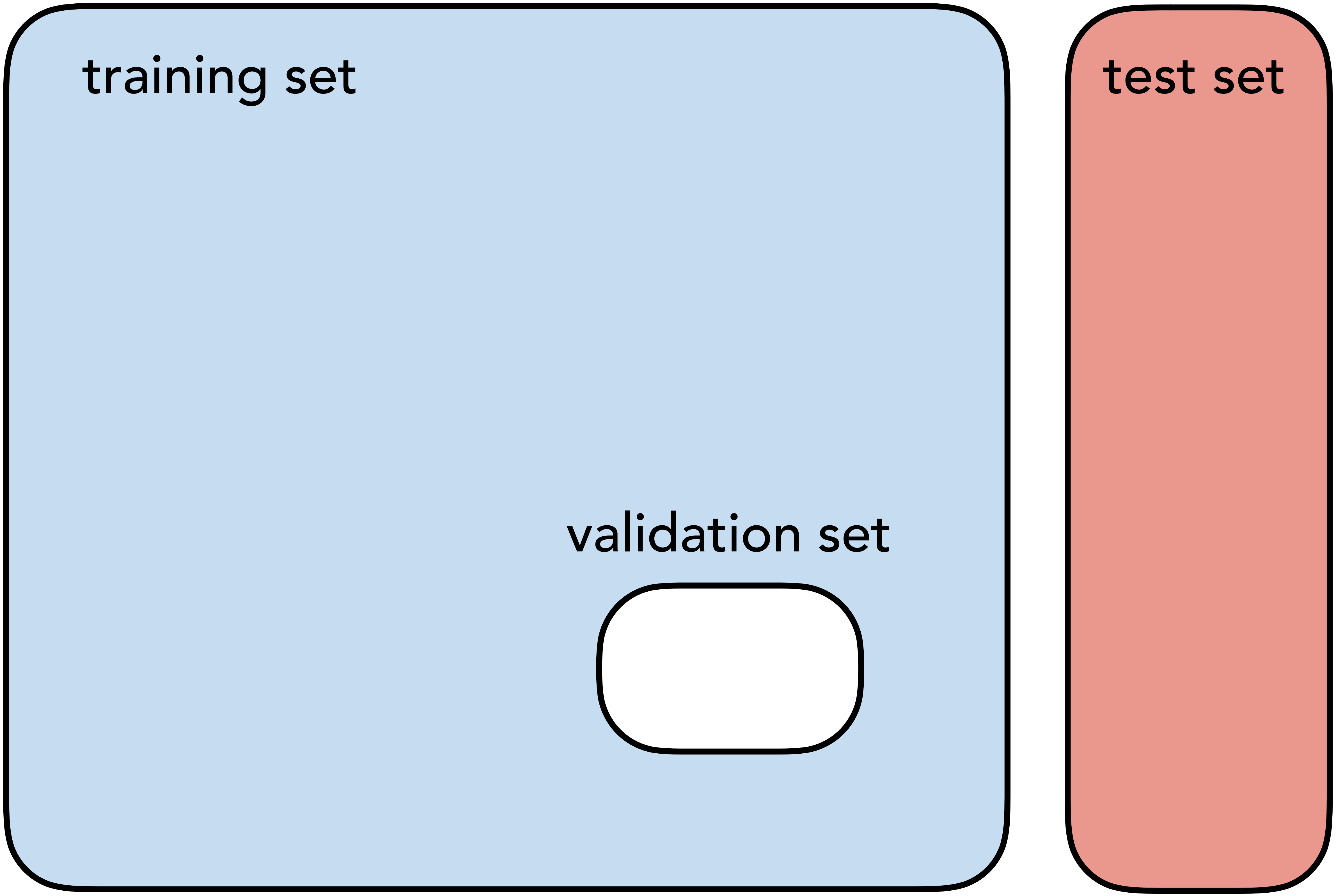
70000 labeled images



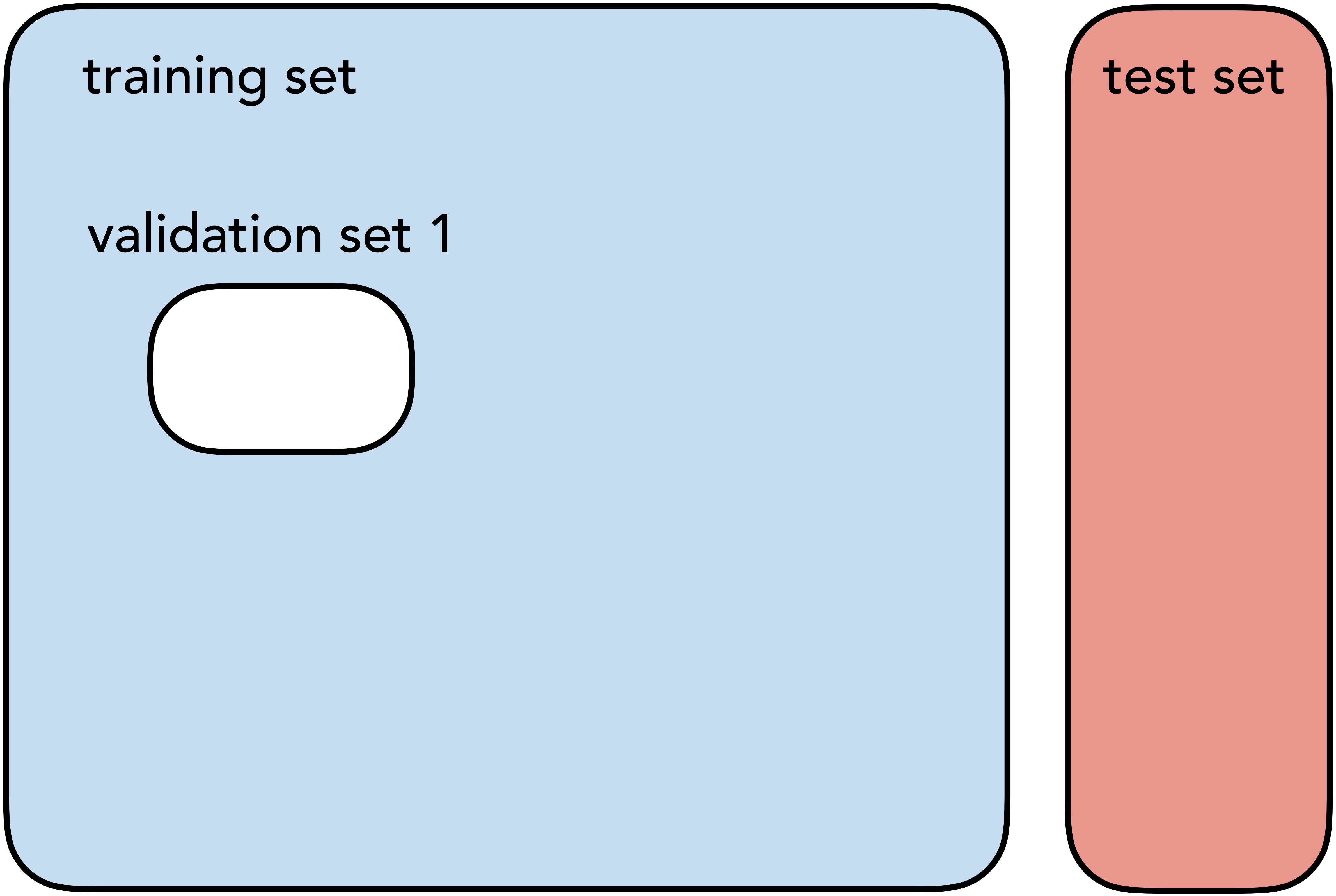
Anatomy of a Dataset



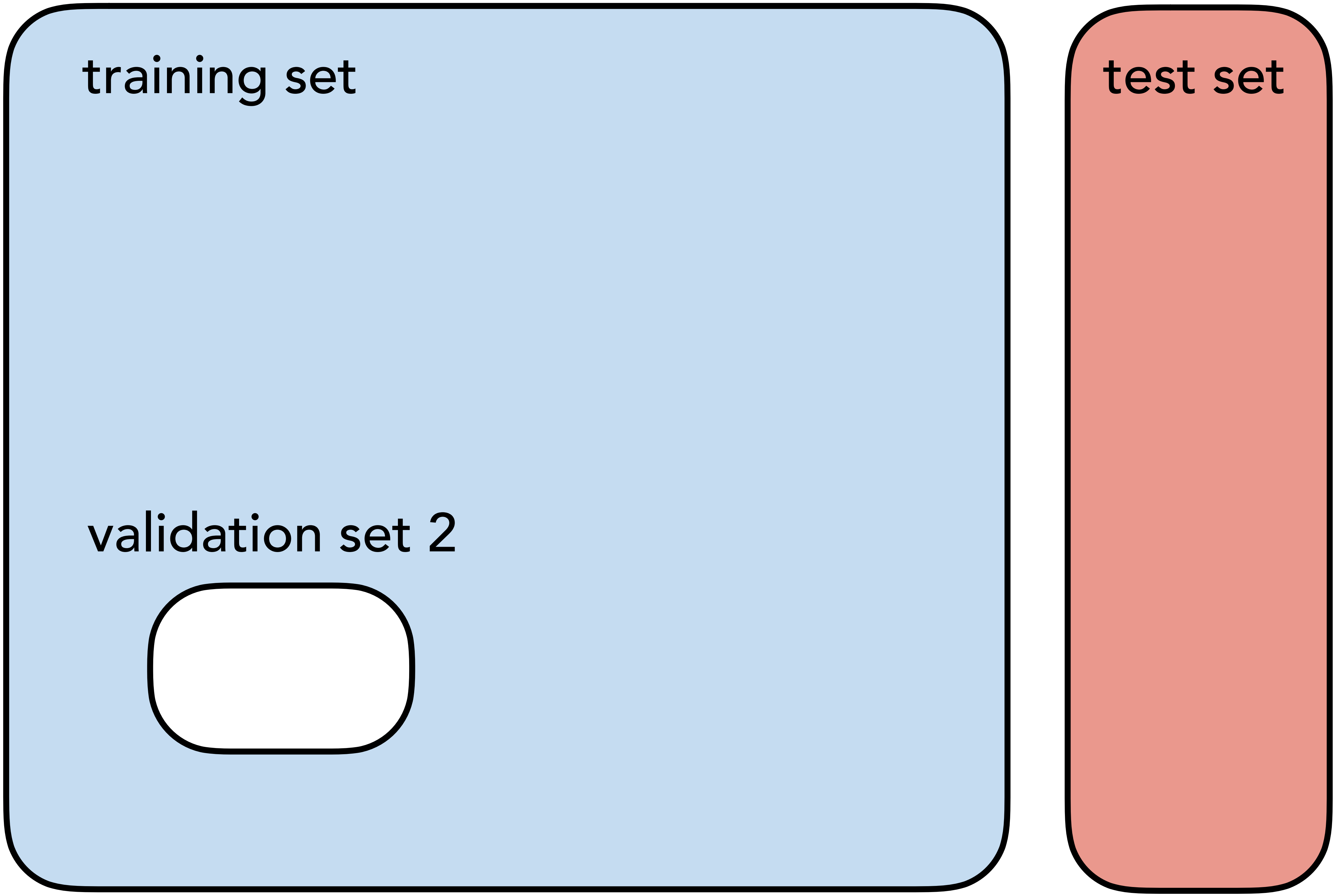
Anatomy of a Dataset



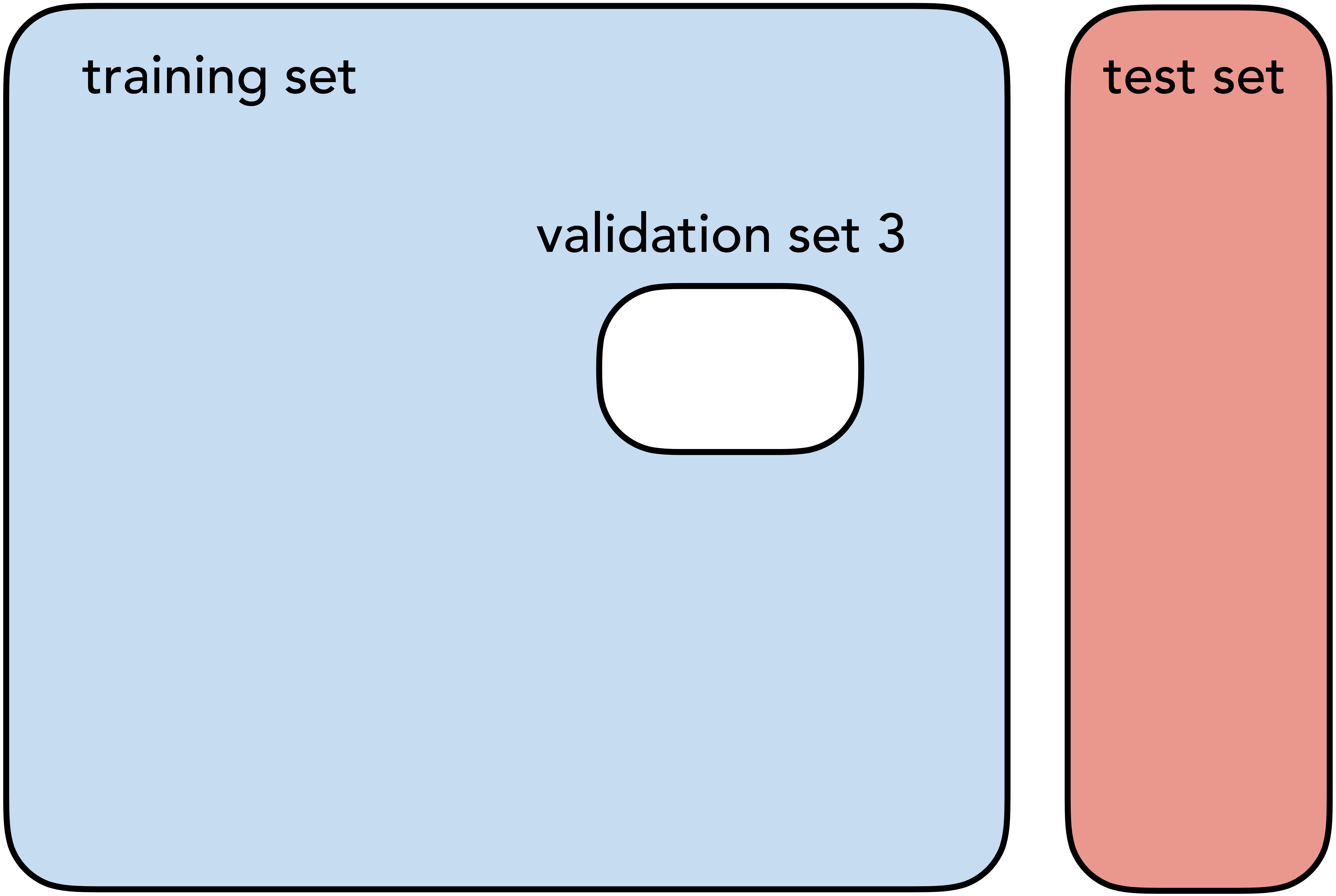
Anatomy of a Dataset



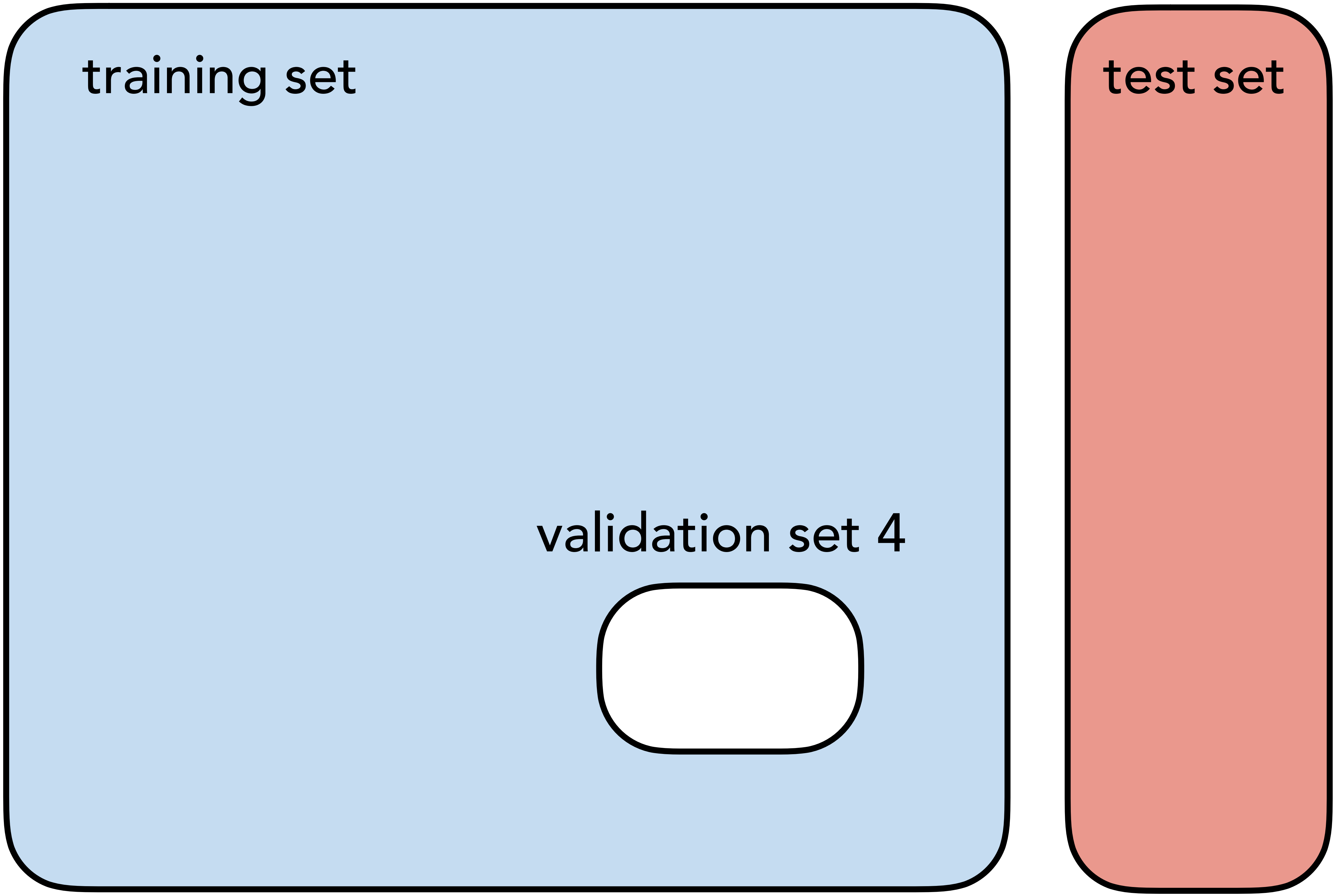
Anatomy of a Dataset



Anatomy of a Dataset

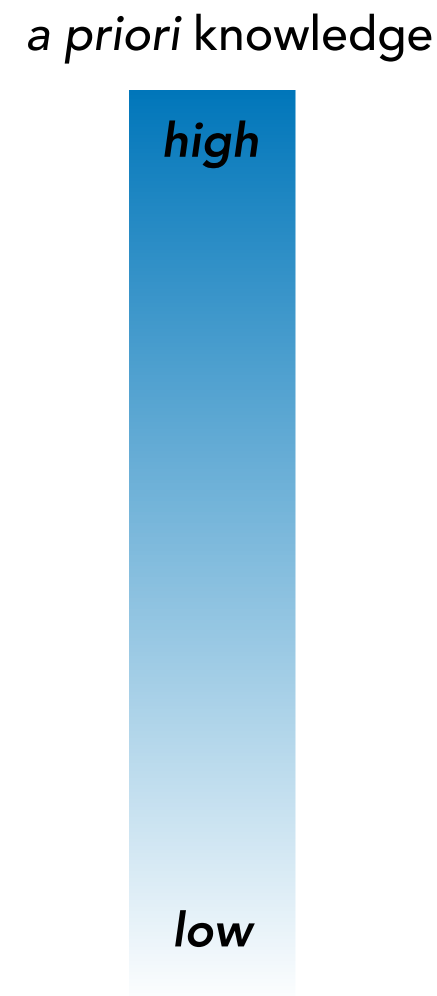


Anatomy of a Dataset



Types of learning tasks:

- Supervised learning (labeled data)
- Unsupervised learning (unlabeled data)
- Reinforcement learning ('reward' data)



Supervised Learning

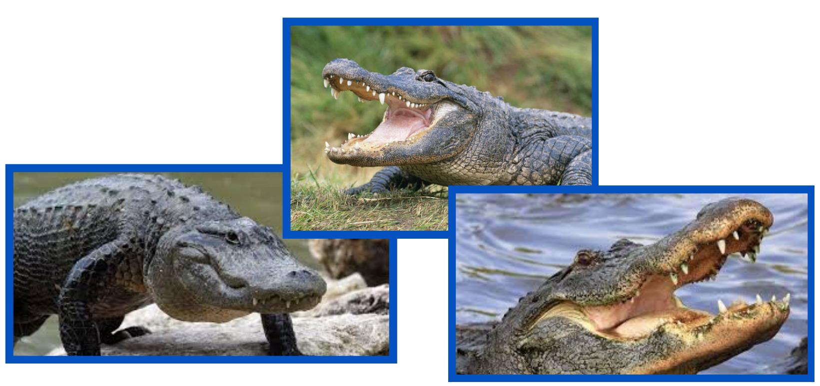
Given labeled training data (labels A and B)

Find *decision function* $f(\mathbf{x})$

$$f(\mathbf{x}) > 0 \qquad \mathbf{x} \in A$$

$$f(\mathbf{x}) < 0 \qquad \mathbf{x} \in B$$

Example: identify photos of alligators and bears



Supervised Learning

Typical strategy:

given training set $\{\mathbf{x}_j, y_j\}$, minimize cost function

$$C = \frac{1}{N_T} \sum_j (f(\mathbf{x}_j) - y_j)^2 \qquad\qquad y_j = \begin{cases} +1 & \mathbf{x}_j \in A \\ -1 & \mathbf{x}_j \in B \end{cases}$$

by varying adjustable params of f

Cost function measures distance of trial function $f(\mathbf{x}_j)$ from idealized "indicator" function y_j

Unsupervised Learning

Given unlabeled training data $\{\mathbf{x}_j\}$

- Find function $f(\mathbf{x})$ such that $f(\mathbf{x}_j) \simeq p(\mathbf{x}_j)$
- Find function $f(\mathbf{x})$ such that $|f(\mathbf{x}_j)|^2 \simeq p(\mathbf{x}_j)$
- Find data clusters and which data belongs to each cluster
- Discover reduced representations of data for other learning tasks (e.g. supervised)

Unsupervised Learning

Typical approach for inferring $p(\mathbf{x})$

Given data $\{\mathbf{x}_j\}$, maximize log likelihood

$$\mathcal{L} = \sum_j \log p(\mathbf{x}_j)$$

by varying p

Can view log likelihood as distance measure between

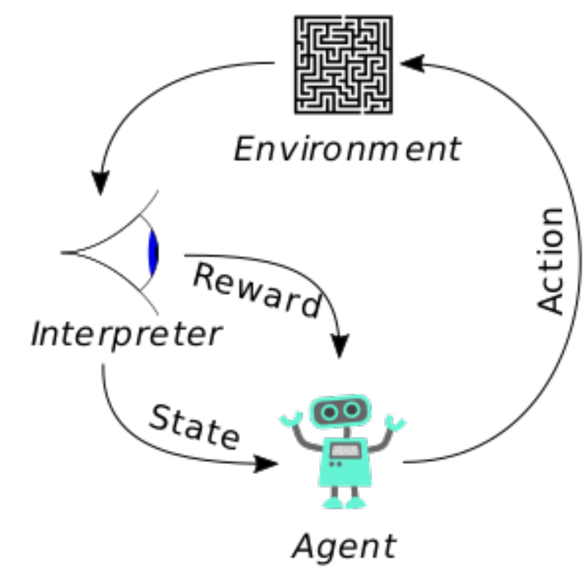
$p(\mathbf{x})$ and $p_{\mathrm{data}}(\mathbf{x}) = \sum_j \delta(\mathbf{x} - \mathbf{x}_j)$

("Kullback-Leibler divergence")

Reinforcement learning

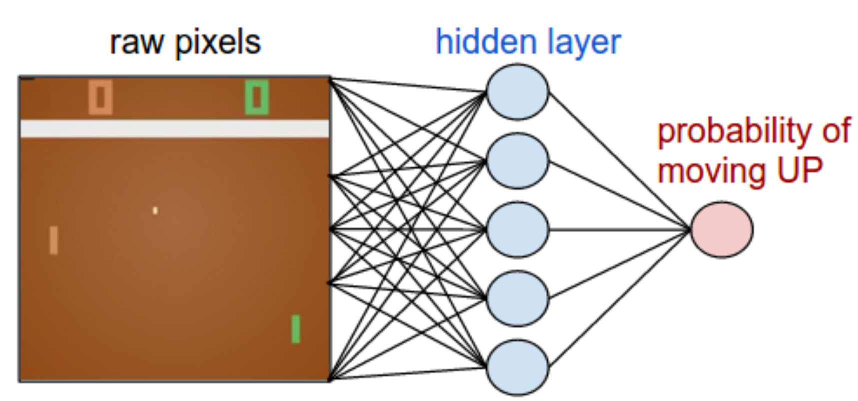
Many flavors, but have common features

- environment & agent with states s_n
- agent actions a_n
- reward $R(s_n)$ for being in state s_n



Goal: determine a policy $P(s_n) \longrightarrow a_n$,
best actions to maximize reward in fewest steps

Example: learning "Pong"
by observing screen state



General Philosophy of Machine Learning

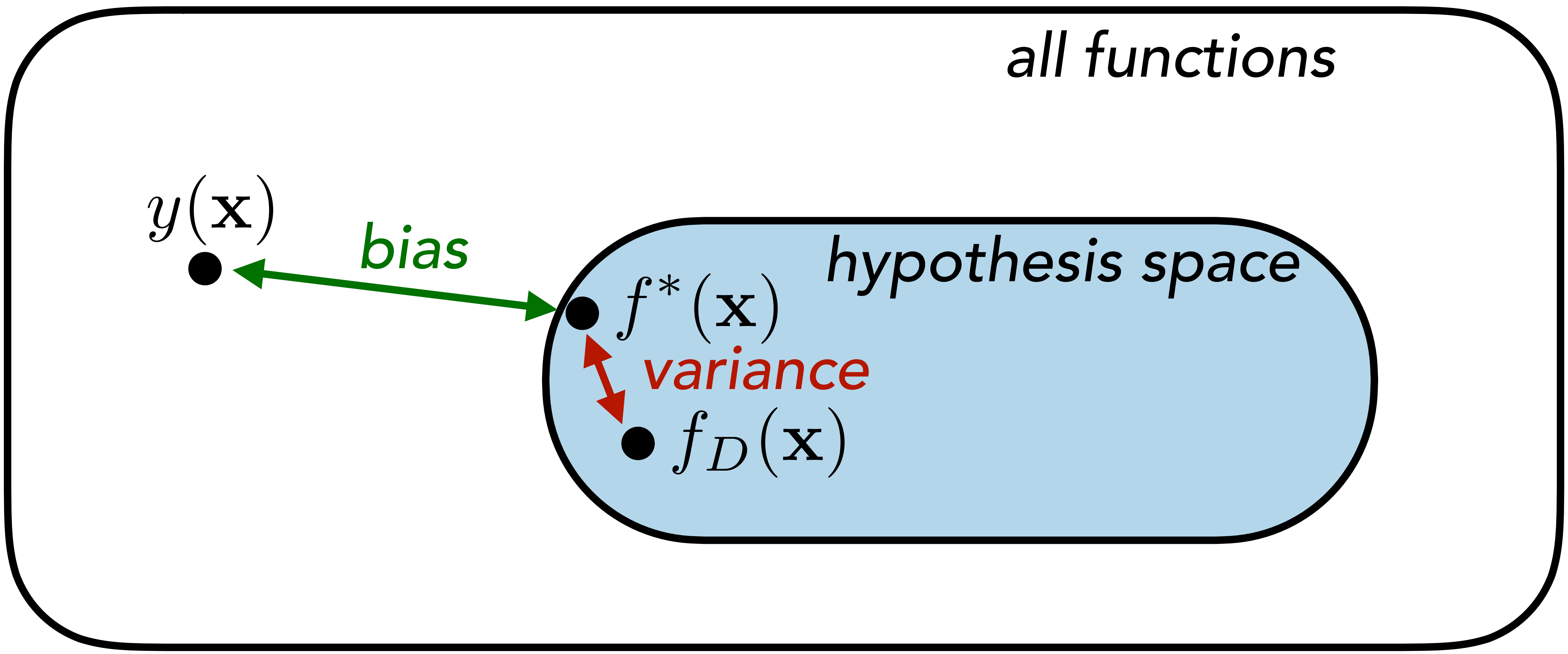
- Solution to problem just some function $y(\mathbf{x})$
- Parameterize very flexible functions $f(\mathbf{x})$ (prefer convenient over "correct")
- Of all f that come closest to y for training data, prefer the simplest f

Bias-Variance Tradeoff

$y(\mathbf{x})$ – ideal solution function

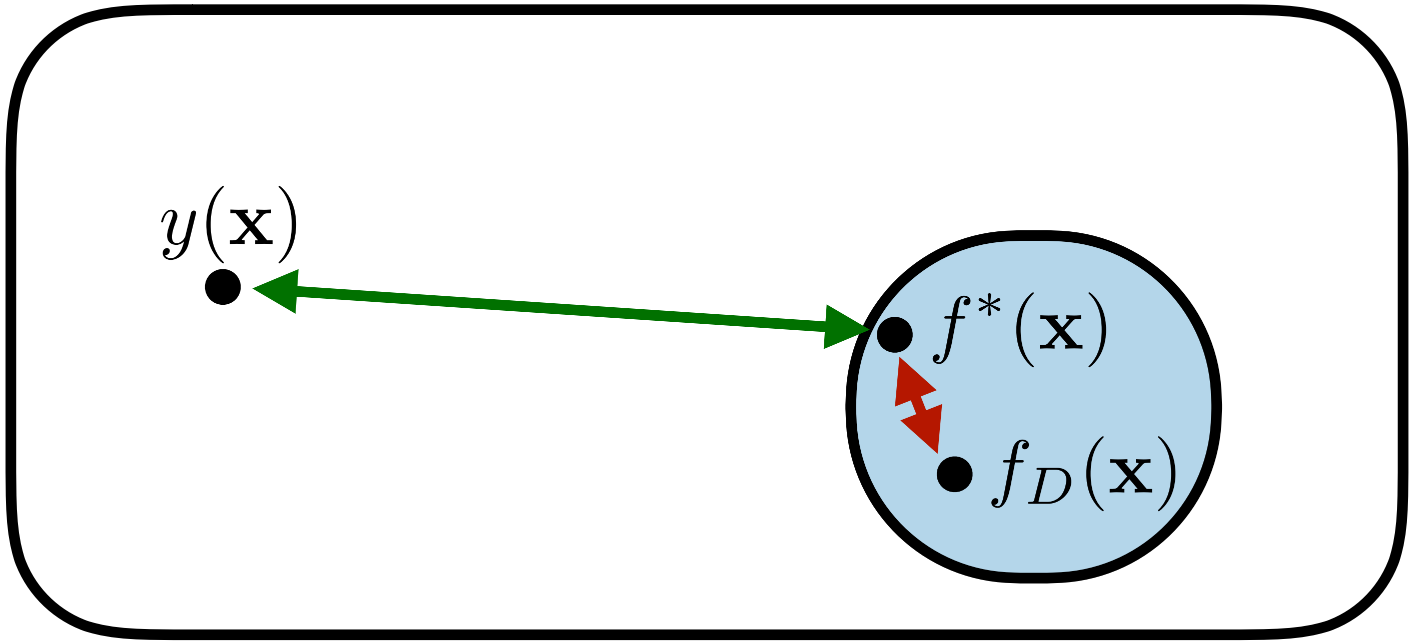
$f^*(\mathbf{x})$ – best possible hypothesis

$f_D(\mathbf{x})$ – best hypothesis given training data



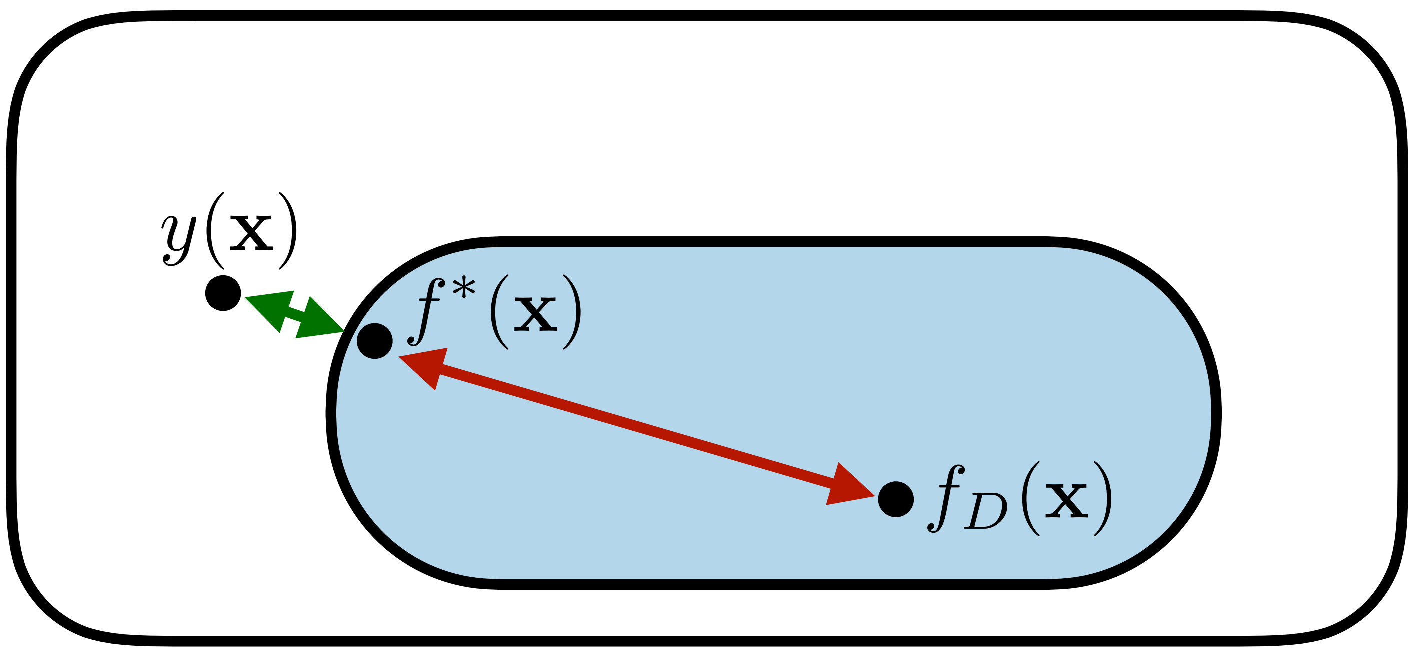
Bias-Variance Tradeoff

Two extreme situations



low variance: will generalize!

high bias: poor results



low bias: good result possible

high variance: might overfit

Model Architectures

Let's discuss the 3 most used types of models (increasing complexity)

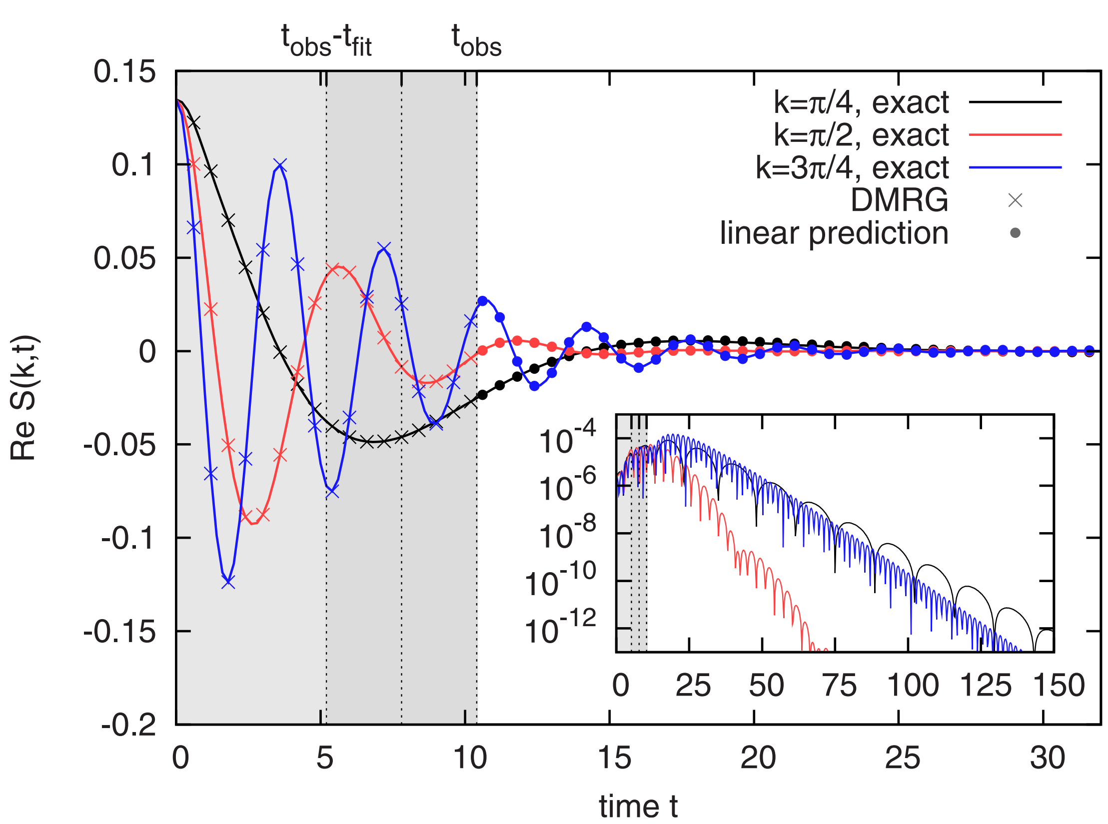
- The linear model
- Kernel learning / support vector machines
- Neural networks

The linear model

$$f(\mathbf{x}) = W \cdot \mathbf{x} + W_0$$

Where W and W_0 are the weights to be learned

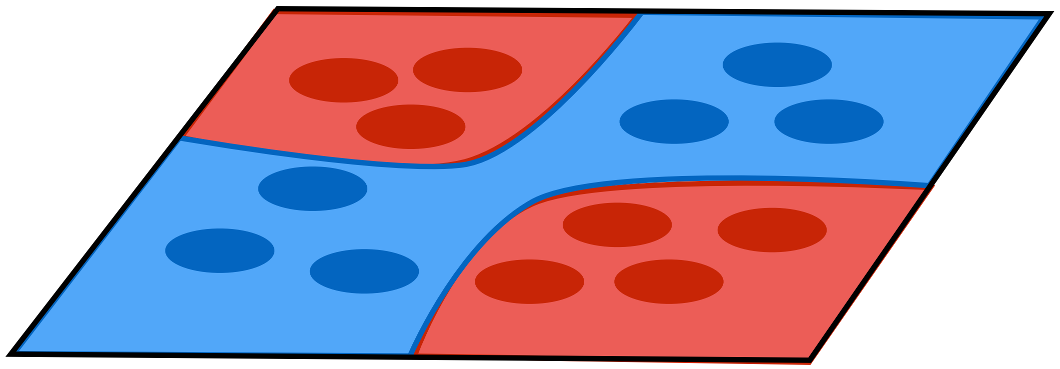
Can be surprisingly powerful, and a useful starting point



Barthel, Schollwöck, White, PRB 79, 245101

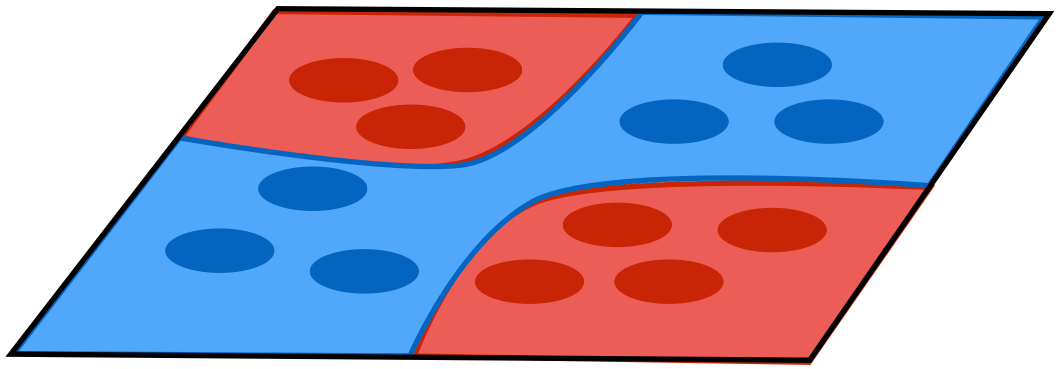
Kernel learning

Want $f(\mathbf{x})$ to separate classes, say



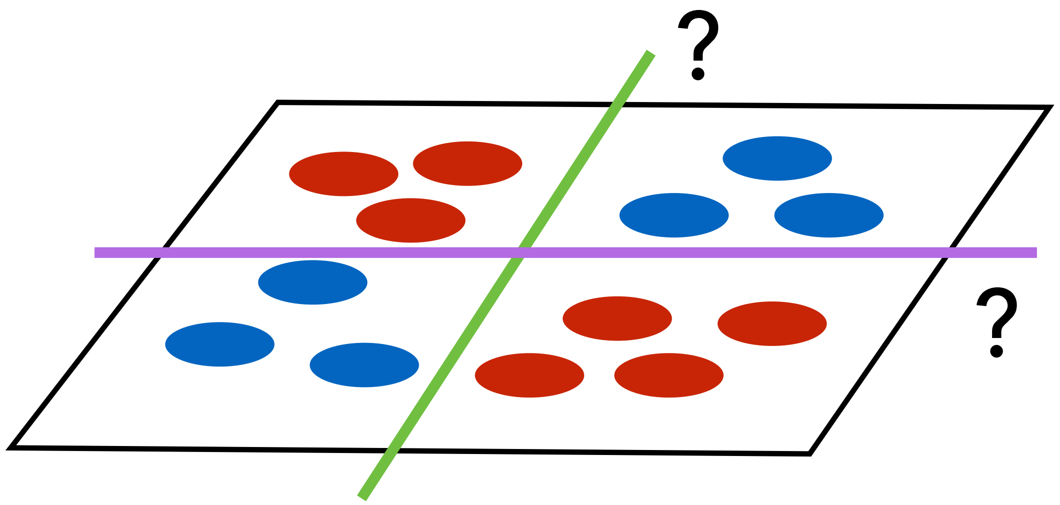
Kernel learning

Want $f(\mathbf{x})$ to separate classes, say



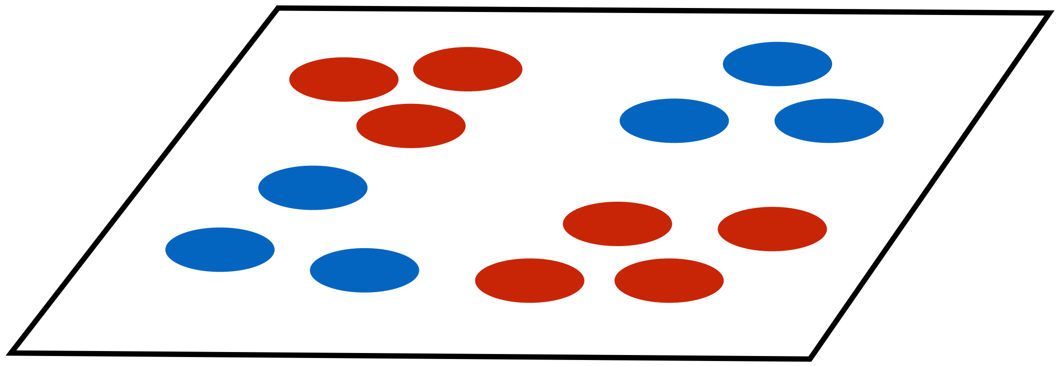
Linear classifier
may be insufficient

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



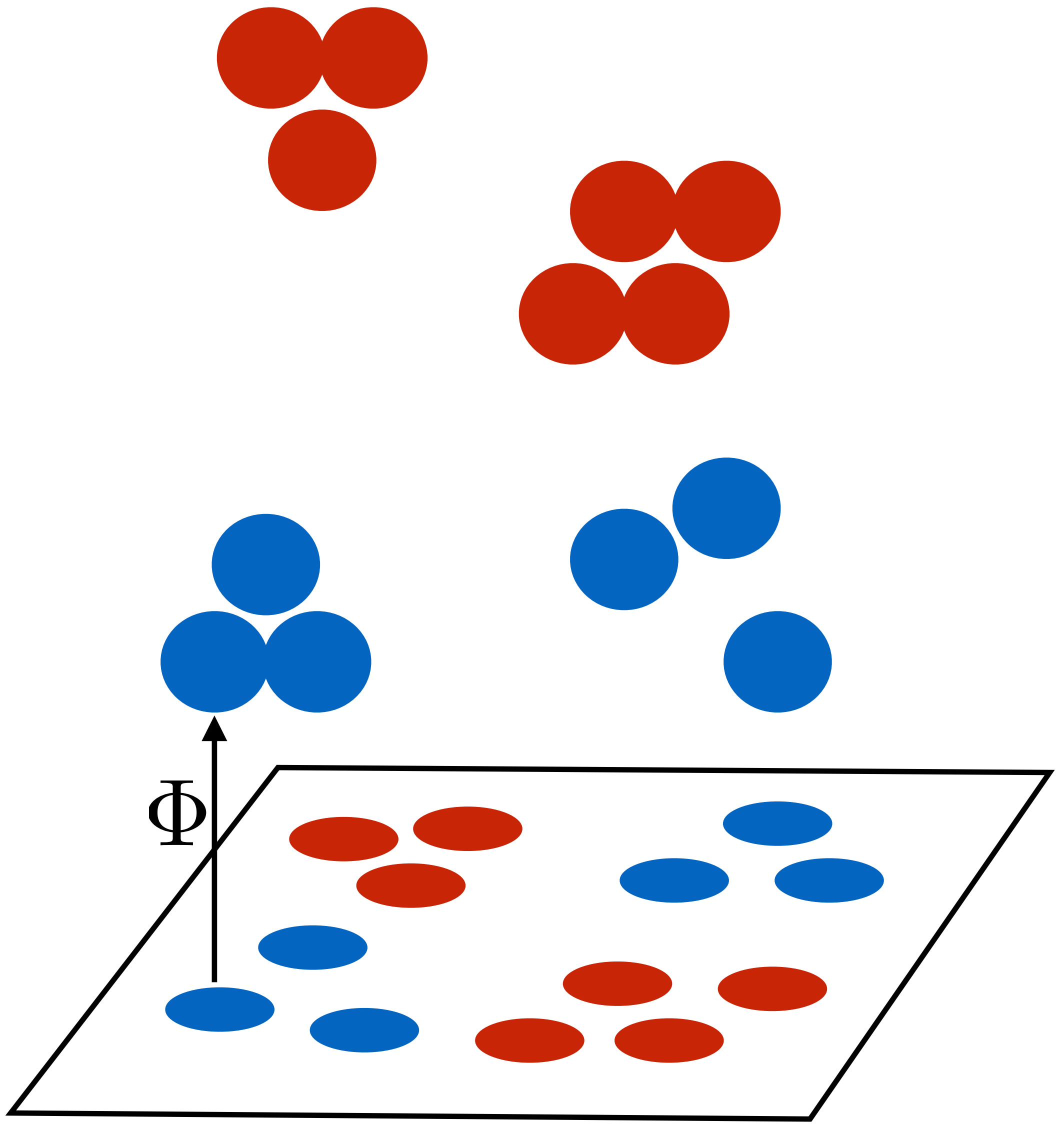
Kernel learning

Apply non-linear "feature map" $\mathbf{x} \rightarrow \Phi(\mathbf{x})$



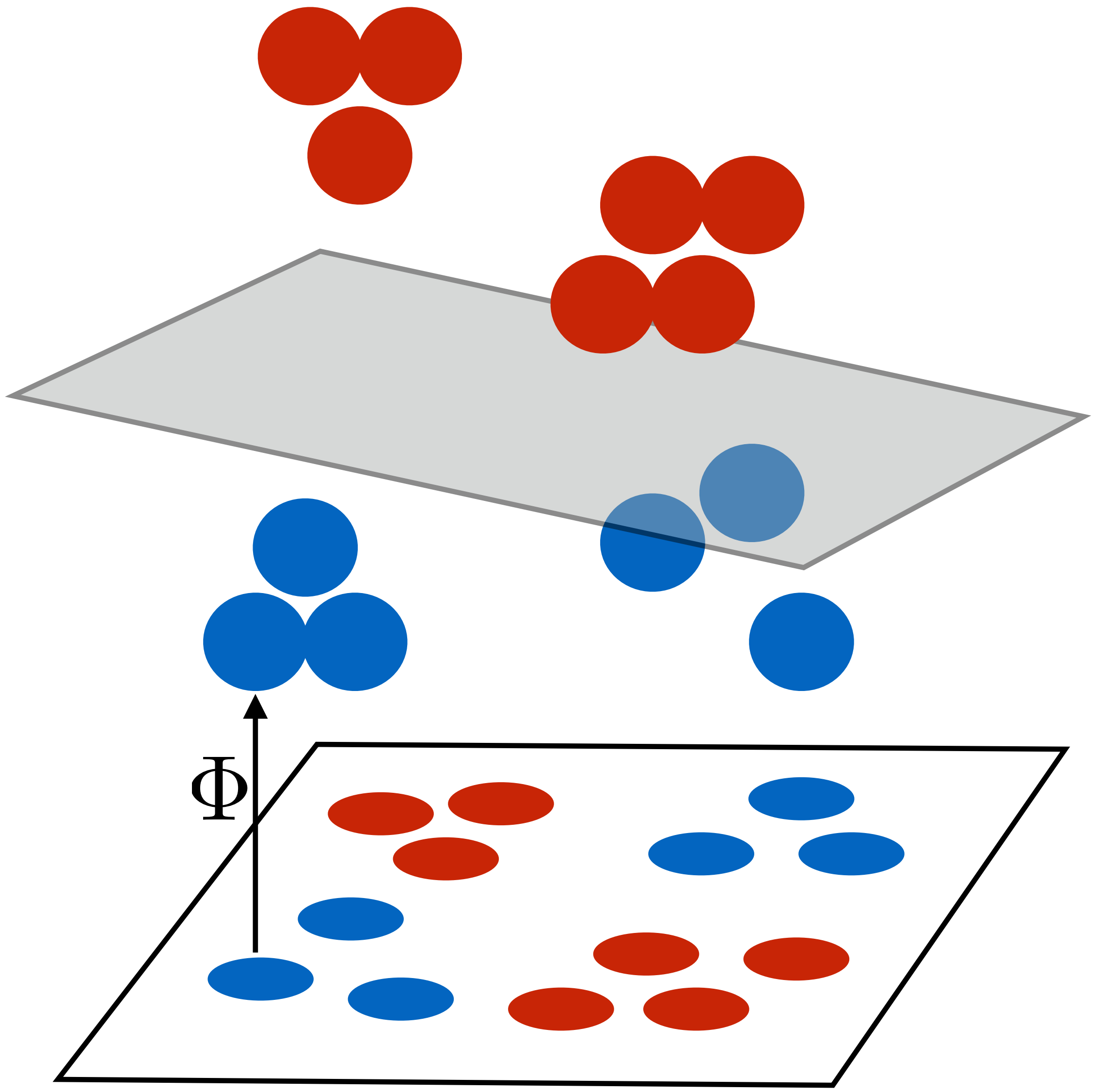
Kernel learning

Apply non-linear "feature map" $\mathbf{x} \rightarrow \Phi(\mathbf{x})$



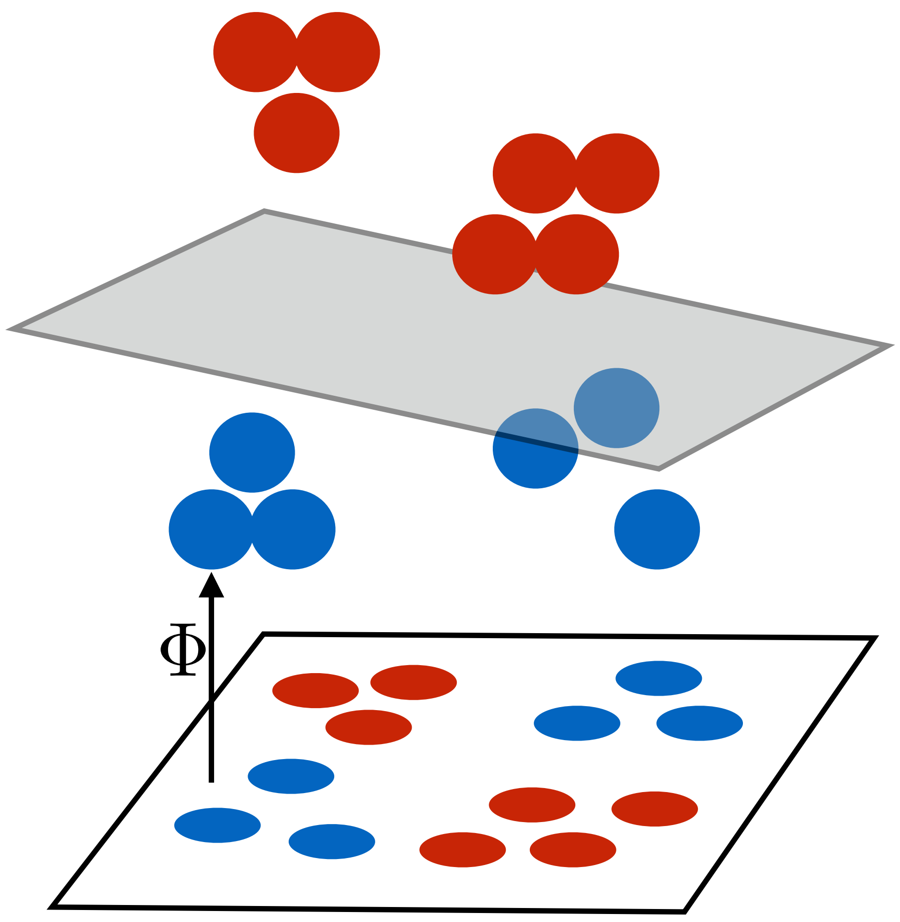
Kernel learning

Apply non-linear "feature map" $\mathbf{x} \to \Phi(\mathbf{x})$



Decision function $$f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})$$

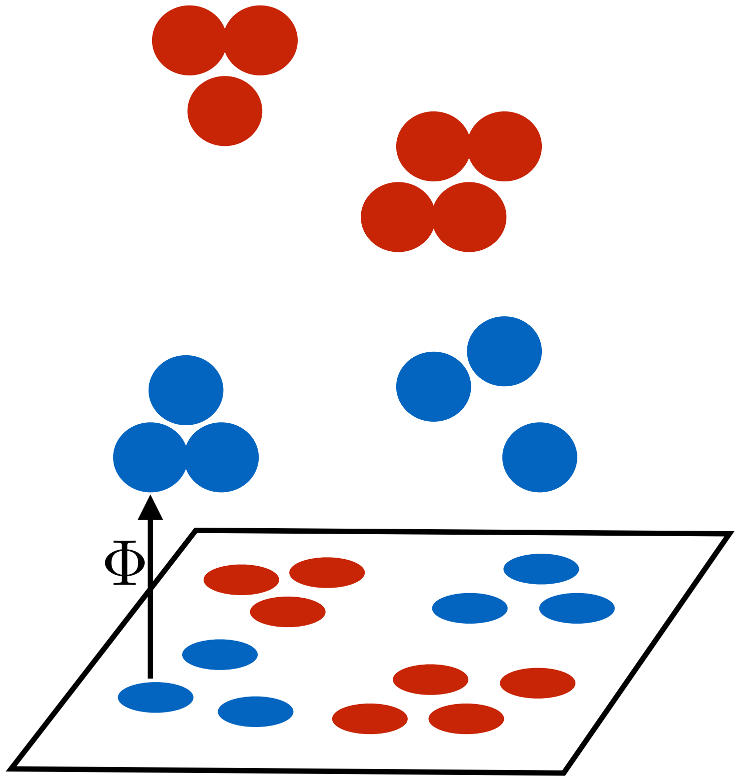
Kernel learning



Decision function $$f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})$$

Linear classifier in *feature space*

Kernel learning



Example of *feature map*

$$\mathbf{x} = (x_1,\, x_2,\, x_3)$$

$$\Phi(\mathbf{x}) = (1,\, x_1,\, x_2,\, x_3,\, x_1x_2,\, x_1x_3,\, x_2x_3)$$

$\mathbf{x}$ is "lifted" to feature space

Kernel learning

Technical notes:

- Also called "support vector machine" when using a particular choice of cost function
- Name "kernel learning" comes from idea that $\Phi(\mathbf{x})$ may be too high dimensional, yet $K_{ij} = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ may be efficiently computable, enough to optimize
- Very generally, optimal weights have the form

$$W = \sum_j \alpha_j \Phi(\mathbf{x}_j)$$

a result known as the "representer theorem"

Kernel learning

Kernel learning still popular among academics & for certain applications (e.g. life sciences)

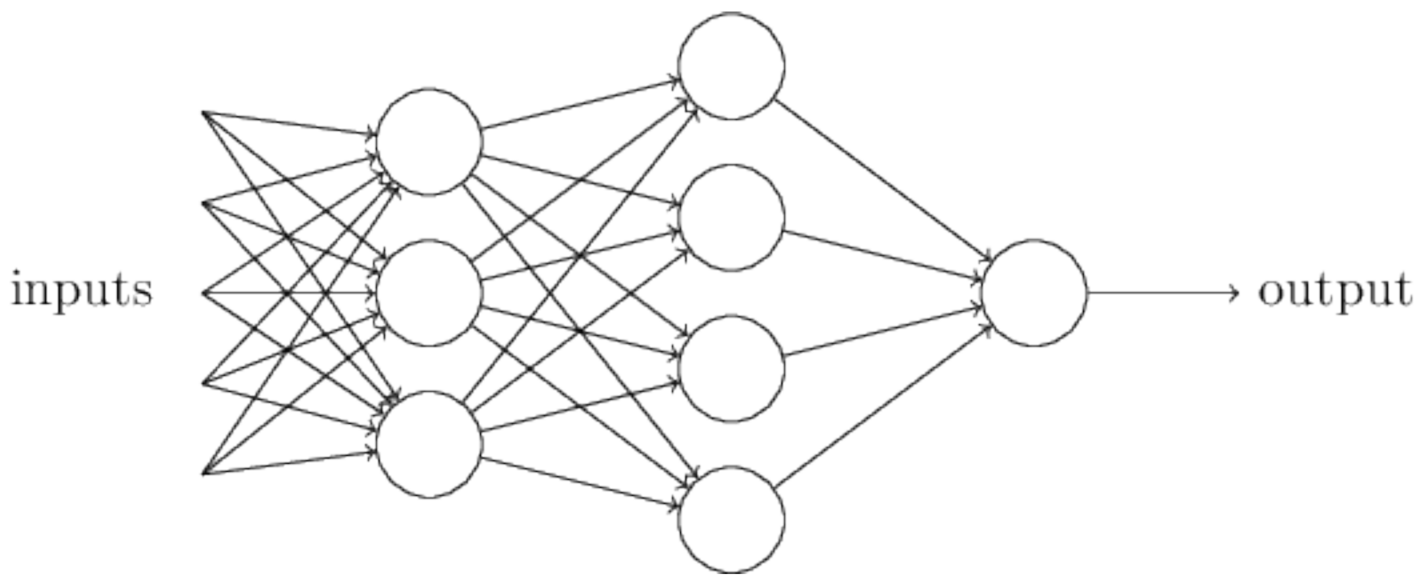
But "kernelization" approach scales as N^3 where N is size of training set – very costly!

Thus kernel methods not popular with engineers

Tomorrow: learning kernel models with tensor network weights

Neural networks

Current favorite of M.L. engineers



Often notated diagrammatically
(not a tensor diagram!)

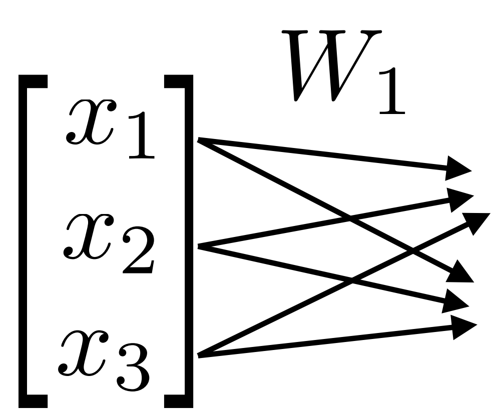
Neural networks

Actually very simple: compute a function $f(\mathbf{x})$ as

Neural networks

Actually very simple: compute a function $f(\mathbf{x})$ as

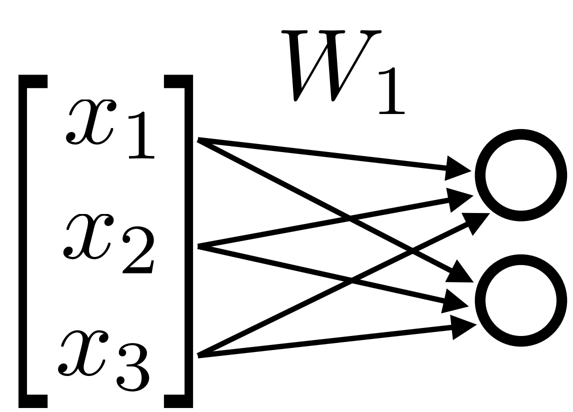
- Multiply input $\mathbf{x}$ by rectangular "weight" matrix W_1



Neural networks

Actually very simple: compute a function $f(\mathbf{x})$ as

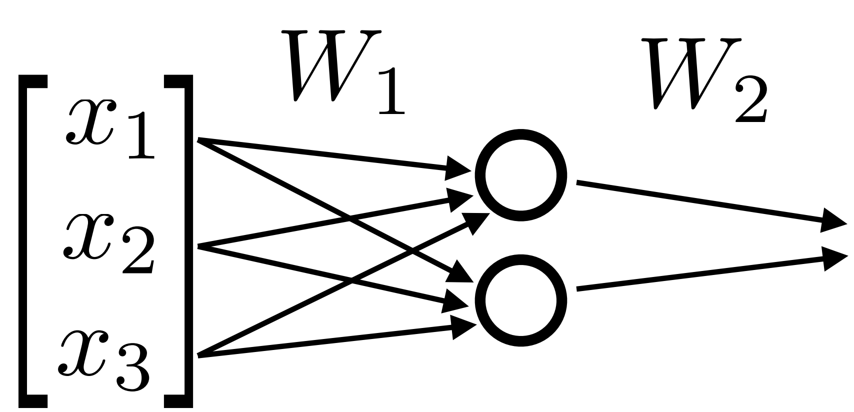
- Multiply input $\mathbf{x}$ by rectangular "weight" matrix W_1
- Point-wise evaluate components of $\mathbf{x}' = W_1\mathbf{x}$ by some non-linear function [e.g. $\sigma(x_j') = 1/(1 - e^{x_j' - b})$]



Neural networks

Actually very simple: compute a function $f(\mathbf{x})$ as

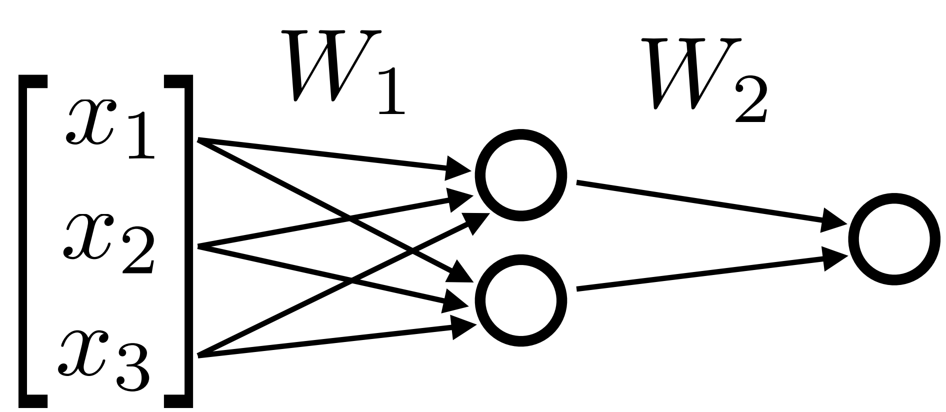
- Multiply input $\mathbf{x}$ by rectangular "weight" matrix W_1
- Point-wise evaluate components of $\mathbf{x}' = W_1\mathbf{x}$ by some non-linear function [e.g. $\sigma(x'_j) = 1/(1 - e^{x'_j - b})$]
- Multiply result by second weight matrix W_2



Neural networks

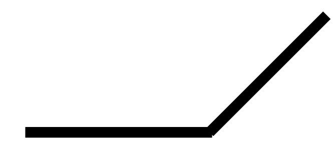
Actually very simple: compute a function $f(\mathbf{x})$ as

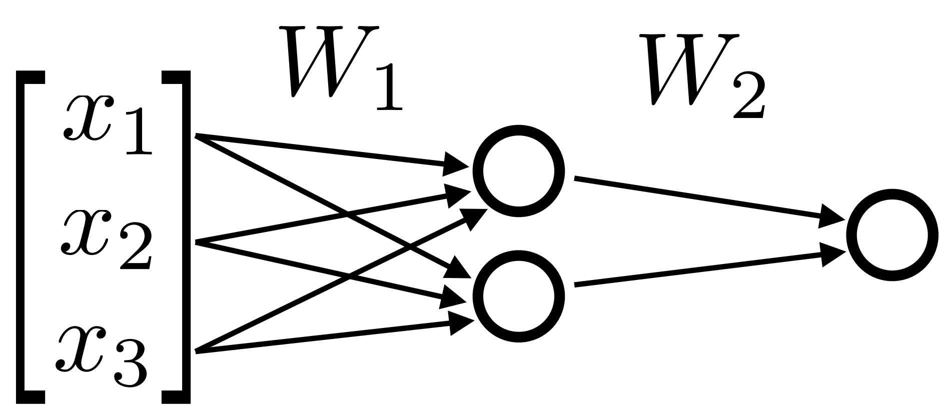
- Multiply input $\mathbf{x}$ by rectangular "weight" matrix W_1
- Point-wise evaluate components of $\mathbf{x}' = W_1\mathbf{x}$ by some non-linear function [e.g. $\sigma(x'_j) = 1/(1 - e^{x'_j - b})$]
- Multiply result by second weight matrix W_2
- Plug new components into non-linearities, etc.



Neural networks

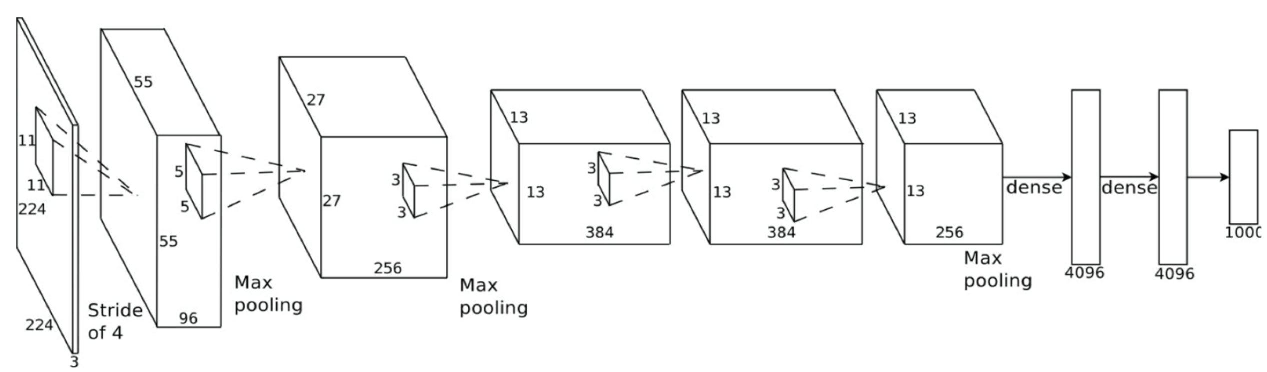
Additional facts:

- Non-linearities $\sigma(x)$ called "neurons"
- Other neurons include tanh and ReLU 
- Neural net with more than one weight matrix is "deep"
- Number of neurons is arbitrary, but with enough can represent any function



Neural networks

Many successful neural nets include "convolutional layers"
These have sparser weight layers with few parameters.



Recent upsurge of neural nets since 2012 (ImageNet paper)

"Deep learning" often associated with 3 researchers:

Yann LeCun (Facebook)

Geoff Hinton (Vector/Google)

Yoshua Bengio (Montreal)

Other model types

Graphical models

very similar to tensor networks, except

- always interpreted as probability
- non-negative parameters only

Boltzmann machines

identical to random-bond classical Ising (T=1)

J_{ij} values learnable parameters

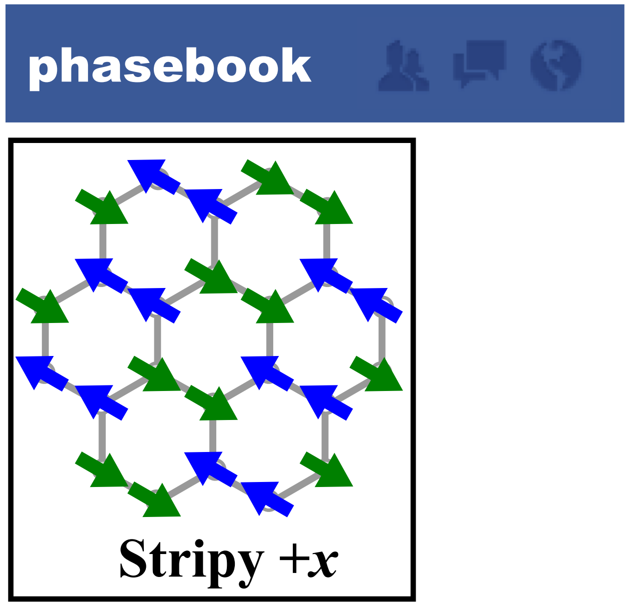
generate data by sampling subset of spins

Decision trees

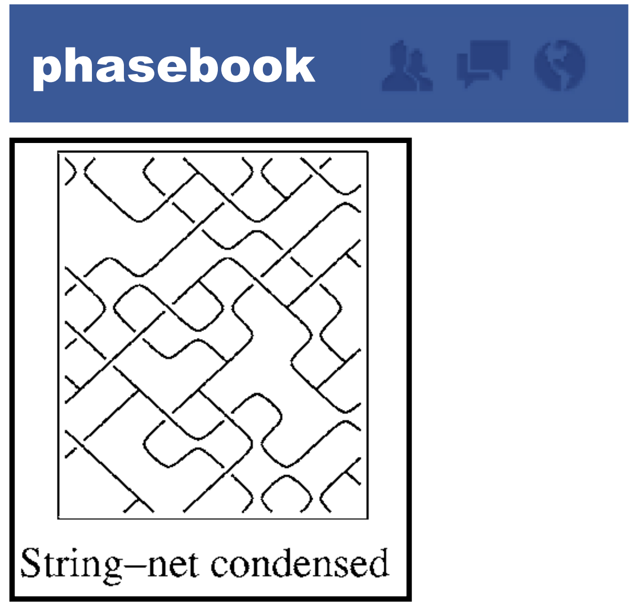
make decisions about input by taking forking paths

Selected Physics Applications

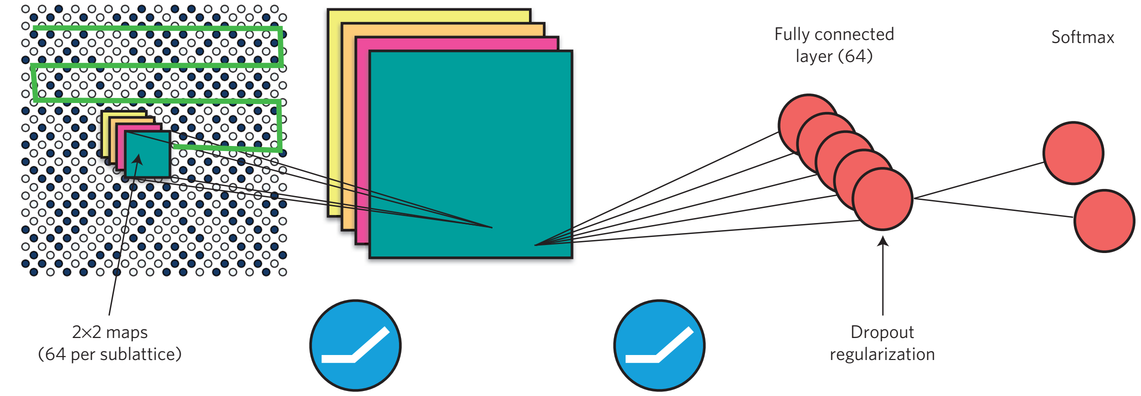
Phase recognition



Friends:
Lev Landau
Werner Heisenberg



Friends:
Michael Levin
Xiao-Gang Wen



View Monte Carlo configurations as input data, train model (supervised or unsupervised) to distinguish phases

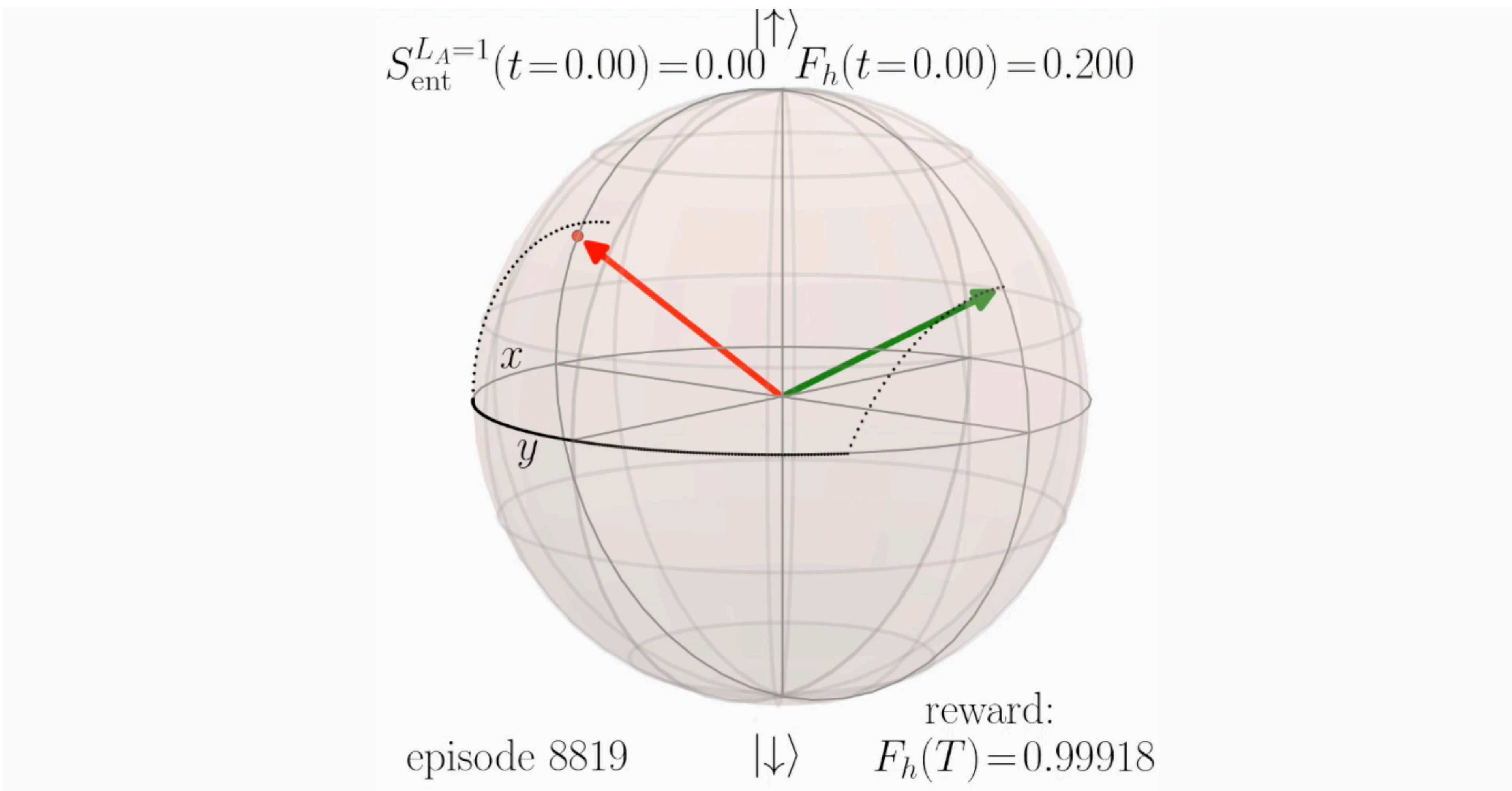
Some relevant papers:

- Carrasquilla, Melko, Nature Phys. (2017) [**supervised**]
- Wang, PRB 94, 195105 [**unsupervised**]
- Broecker, Carrasquilla, Melko, Trebst Scientific Reports 7, 8823 (2017) [**from aux. field QMC**]
- Broecker, Assaad, Trebst arxiv:1707.00663 [**unsupervised**]
- … and quite a few others …

Learning to Control Quantum Systems

How to apply time-dependent field to quantum system and reach some target state?

Treat fidelity as "reward" and train reinforcement learning agent to work out best protocol



Bukov, Day, et al., arxiv:1705.00565

Many Other Creative Ideas

Learning quantum Monte Carlo updates

J. Liu, Y. Qi, et al. arxiv:1610.03137

L. Huang, L. Wang, arxiv:1610.02746

L. Wang, arxiv:1702.08586

H. Shen, J. Liu, L. Fu, arxiv:1801.01127

Neural Net Representations of Wavefunctions

G. Carleo, M. Troyer, arxiv:1606.02318

D. Deng, X. Li, S. Das Sarma, arxiv:1609.09060, arxiv: 1701.04844

S. Clark, arxiv:1710.03545

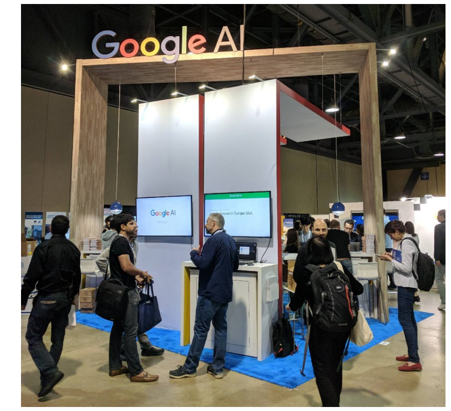
Learning Density Functionals

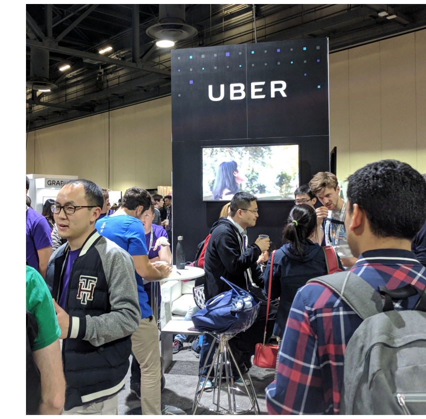
J. Snyder, et al., arxiv:1112.5441

F. Brockherde, et al., arxiv:1609.02815

L. Li, et al., arxiv:1609.03705

Machine Learning Research Culture





One sub-community is academic: papers often involve theorems

Another community is engineering-oriented: papers focus on results, developments are intuitive/faddish

Conference talks/posters valued above journal articles

Strong industry ties: Google, Microsoft, etc. have booths at conferences, grad students poached often

Recommended Resources

- Online book by Michael Nielsen (quant. computing author)
  http://neuralnetworksanddeeplearning.com

- Caltech Lectures by Yaser Abu-Mostafa CS 156
  Available on YouTube. Companion book "Learning from Data"

- M.L. review article by Pankaj Mehta, David Schwab
  aimed at physicists

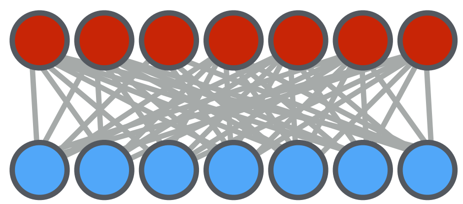
- TensorFlow examples (MNIST demo)

- Blogs of Chris Olah and Andrej Karpathy

Tensor Network Machine Learning

Stoudenmire, Schwab, *Advanced in Neural Information Processing Systems (NIPS)*, **29**, 4799 [arxiv:1605.05775]

Many physics ideas in machine learning

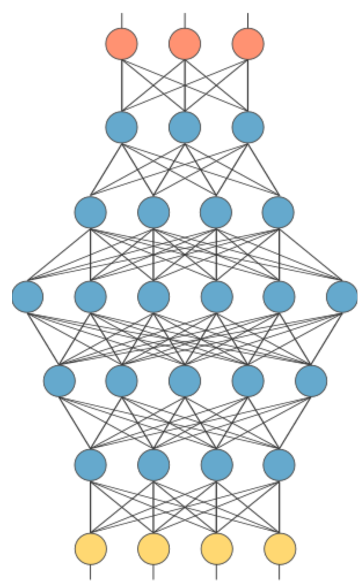


Boltzmann Machines

=

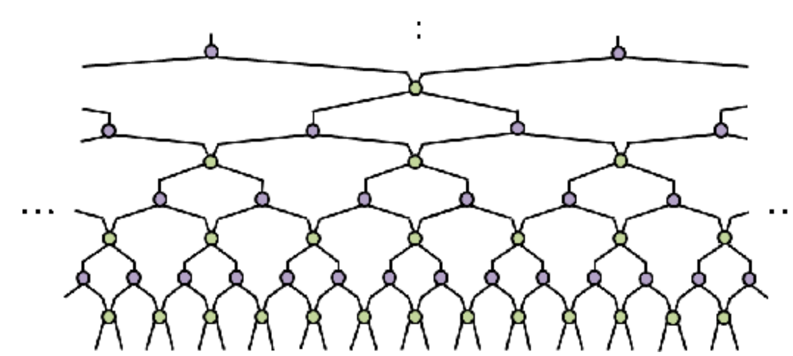
↓ ↑ ↓ ↓ ↑ ↑ ↓ ↑

Ising Model



Deep Belief Networks

=



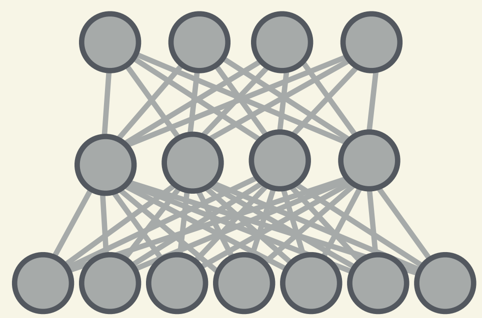
Renormalization Group

Mehta, Schwab, arxiv:1410.3831

Let's apply more ideas to M.L!

Analogy between wavefunctions & M.L. models

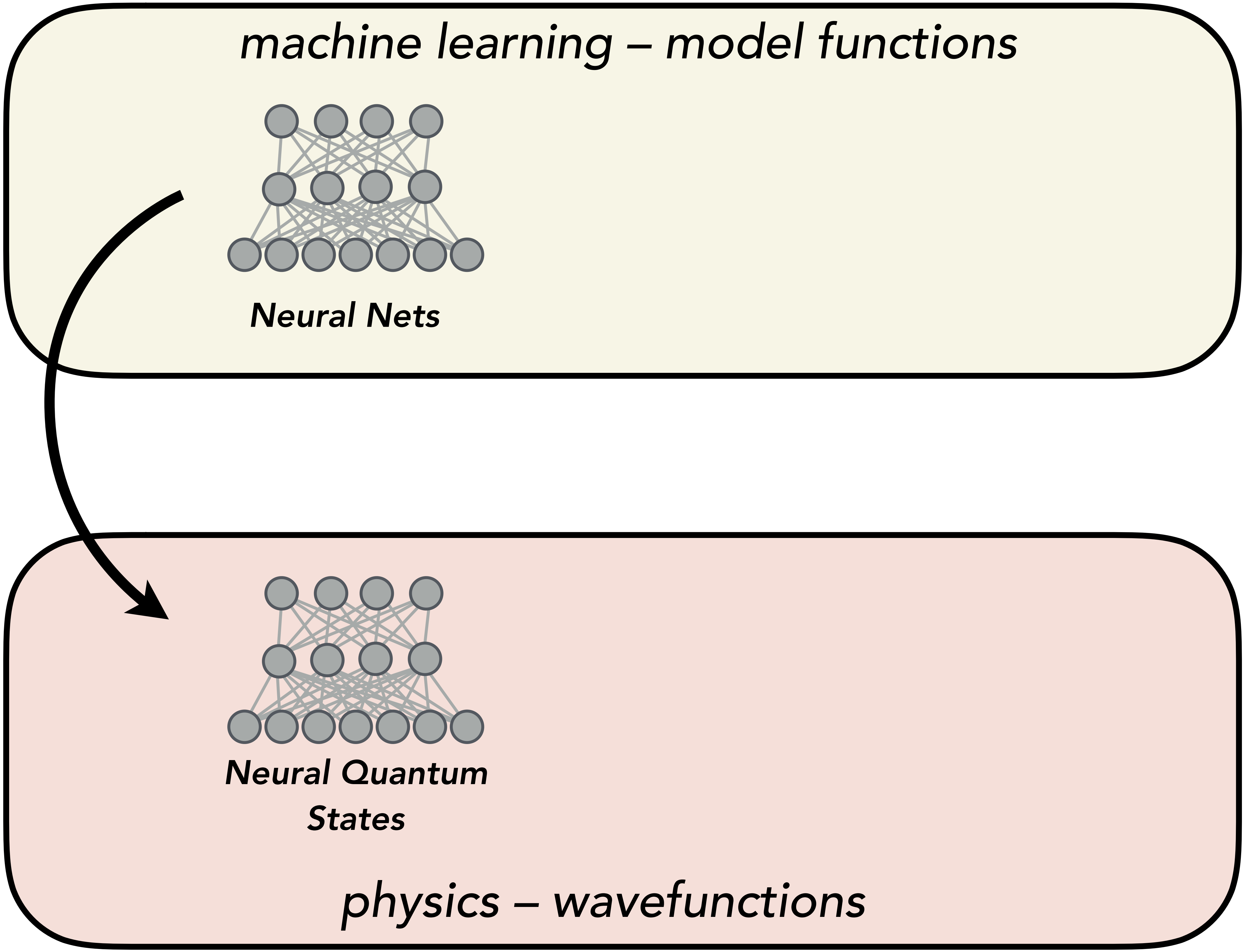
machine learning – model functions



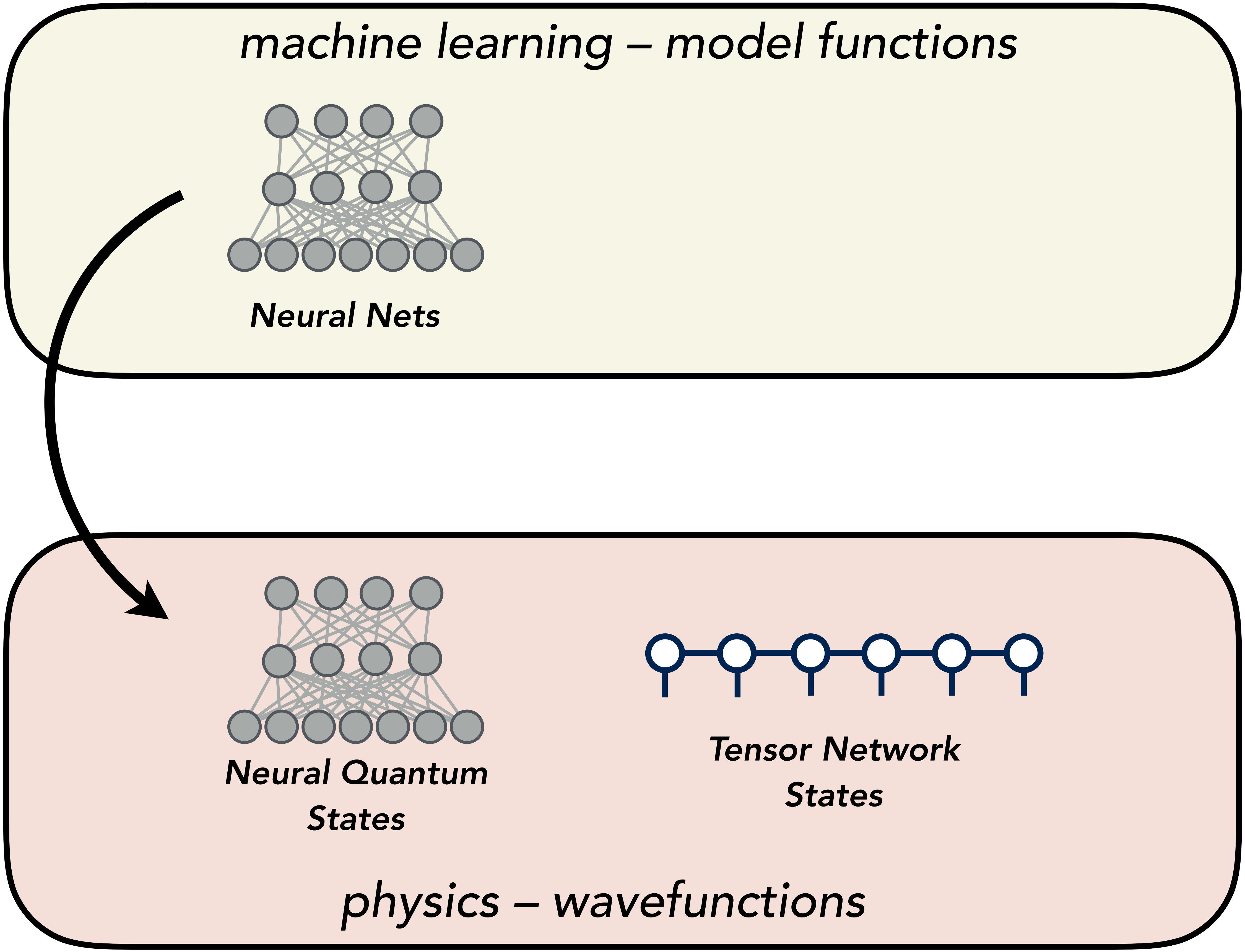
Neural Nets

physics – wavefunctions

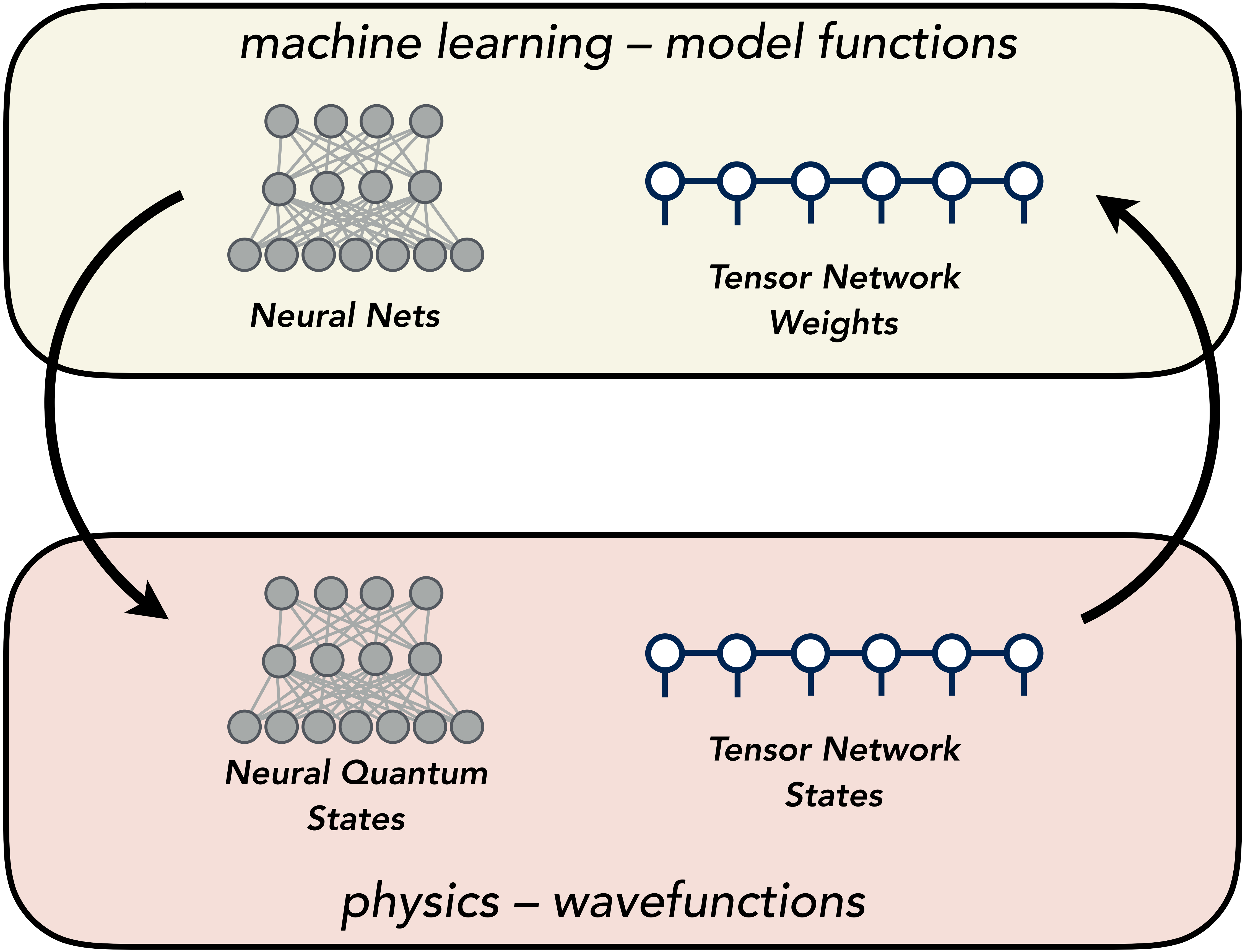
Analogy between wavefunctions & M.L. models



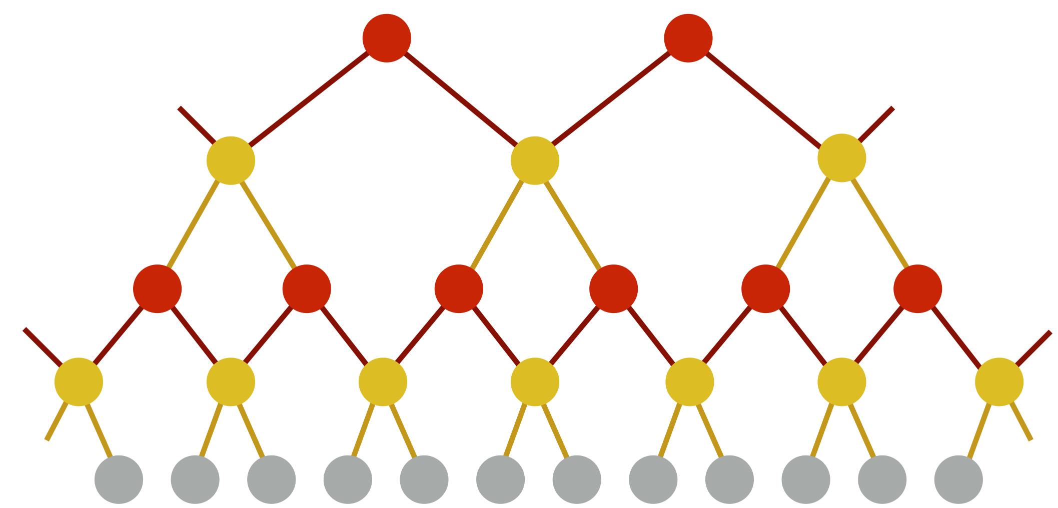
Analogy between wavefunctions & M.L. models



Analogy between wavefunctions & M.L. models



Are tensor networks useful for machine learning?



"MERA" tensor network

Tensor networks can represent weights of useful and interesting machine learning models

Realized benefits:

- Linear scaling
- Adaptive weights
- Learning data "features"

Future benefits?

- Interpretability / theory
- Better algorithms
- Quantum computing

Raw data vectors

$$\mathbf{x} = (x_1, x_2, x_3, \ldots, x_N)$$

Example: grayscale images, components of $\mathbf{x}$ are pixels

$$x_j \in [0, 1]$$

Propose following model

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

$$= \sum_{\mathbf{s}} W_{s_1 s_2 s_3 \cdots s_N} \; x_1^{s_1} x_2^{s_2} x_3^{s_3} \cdots x_N^{s_N} \qquad s_j = 0, 1$$

Weights are N-index tensor
Like N-site wavefunction

Cohen et al. arxiv:1509.05009
Novikov, Trofimov, Oseledets, arxiv:1605.03795
Stoudenmire, Schwab, arxiv:1605.05775

N=3 example:

$$f(\mathbf{x}) = W \cdot \Phi(\mathbf{x}) = \sum_{\mathbf{s}} W_{s_1 s_2 s_3}\; x_1^{s_1} x_2^{s_2} x_3^{s_3}$$

$$= W_{000} + W_{100}\, x_1 + W_{010}\, x_2 + W_{001}\, x_3$$

$$+ W_{110}\, x_1 x_2 + W_{101}\, x_1 x_3 + W_{011}\, x_2 x_3$$

$$+ W_{111}\, x_1 x_2 x_3$$

Contains linear classifier, plus other "feature maps"

More generally, apply local "feature maps" $\phi^{s_j}(x_j)$

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

$$= \sum_{\mathbf{s}} W_{s_1 s_2 s_3 \cdots s_N} \phi^{s_1}(x_1) \phi^{s_2}(x_2) \phi^{s_3}(x_3) \cdots \phi^{s_N}(x_N)$$

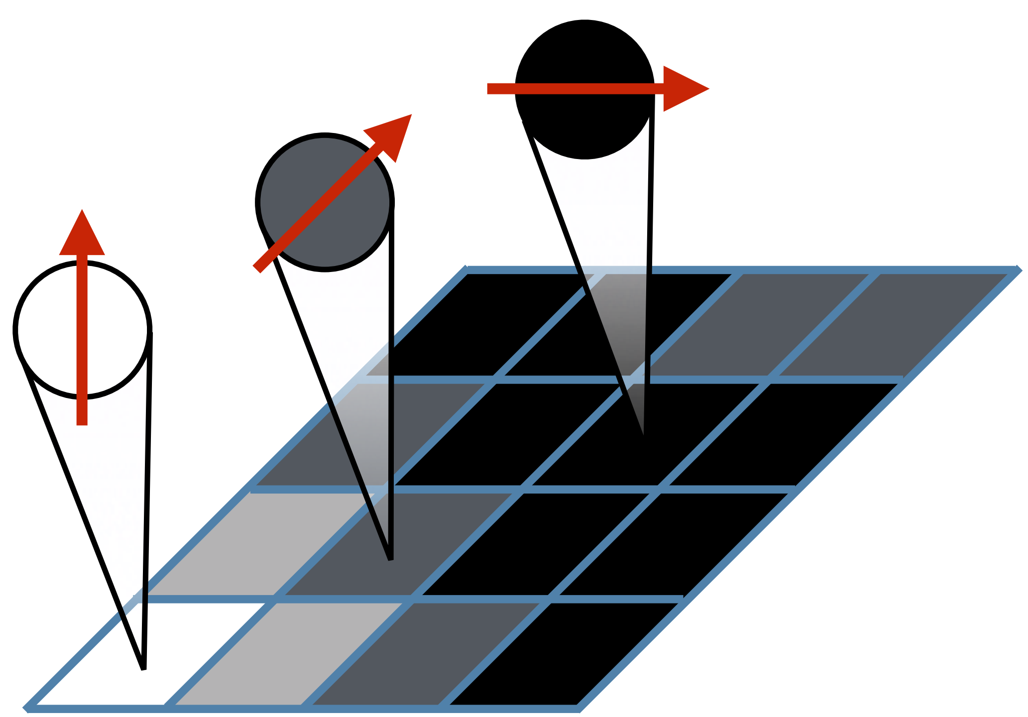
Highly expressive!

$\mathbf{x} =$ input

For example, following local feature map

$$\phi(x_j) = \left[\cos\left(\frac{\pi}{2}x_j\right), \sin\left(\frac{\pi}{2}x_j\right)\right] \qquad x_j \in [0,1]$$

Picturesque idea of pixels as "spins"



$\mathbf{x} =$ input

$\phi =$ local feature map

Total feature map $\Phi(\mathbf{x})$

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

- Tensor product of local feature maps / vectors
- Just like product state wavefunction of spins
- Vector in 2^N dimensional space

$\mathbf{x} =$ input

$\phi =$ local feature map

Total feature map $\Phi(\mathbf{x})$

More detailed notation

$$\mathbf{x} = [x_1, \quad x_2, \quad x_3, \quad \ldots \quad , \quad x_N]$$

raw inputs

$$\Phi(\mathbf{x}) = \begin{bmatrix} \phi_1(x_1) \\ \phi_2(x_1) \end{bmatrix} \otimes \begin{bmatrix} \phi_1(x_2) \\ \phi_2(x_2) \end{bmatrix} \otimes \begin{bmatrix} \phi_1(x_3) \\ \phi_2(x_3) \end{bmatrix} \otimes \cdots \otimes \begin{bmatrix} \phi_1(x_N) \\ \phi_2(x_N) \end{bmatrix}$$

feature vector

$\mathbf{x} =$ input

$\phi =$ local feature map

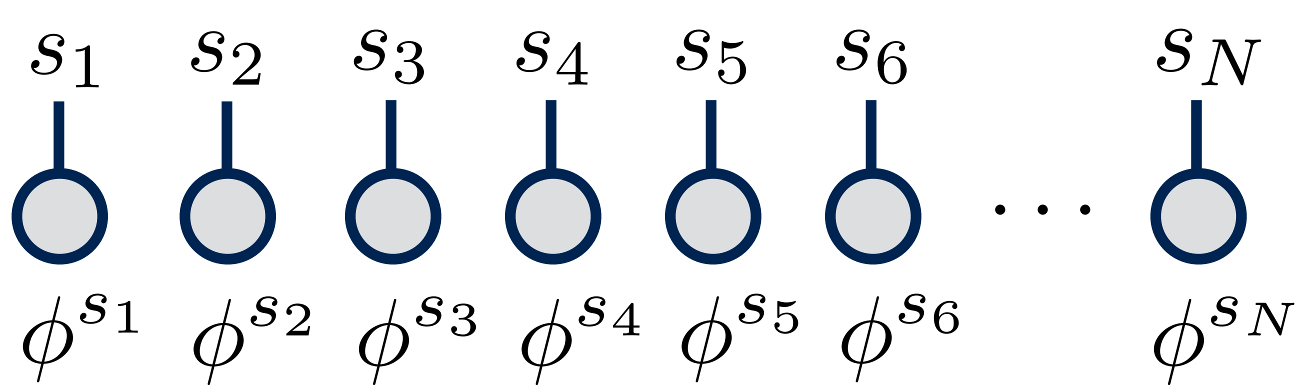
Total feature map $\Phi(\mathbf{x})$

Tensor diagram notation

$$\mathbf{x} = [x_1, \quad x_2, \quad x_3, \quad \ldots \quad , \quad x_N]$$

raw inputs

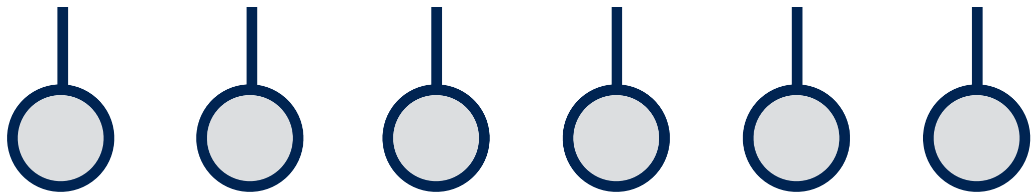
$\Phi(\mathbf{x}) =$



feature vector

Construct decision function

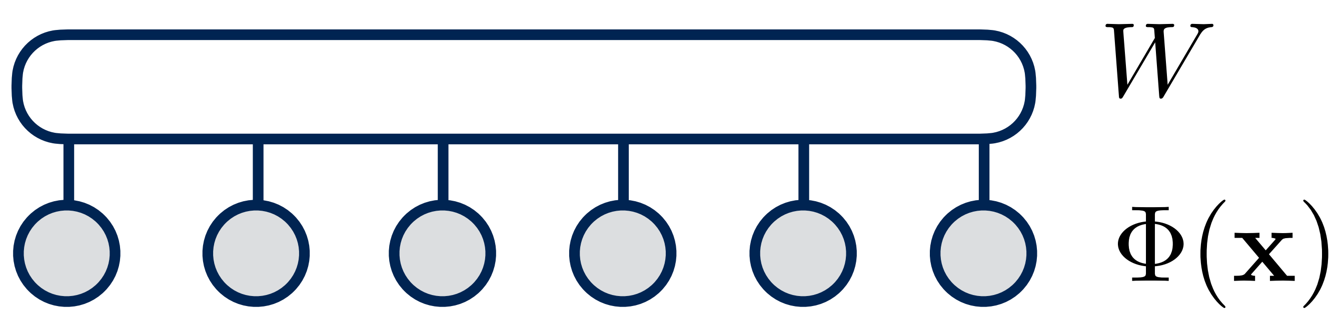
$$f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})$$



$\Phi(\mathbf{x})$

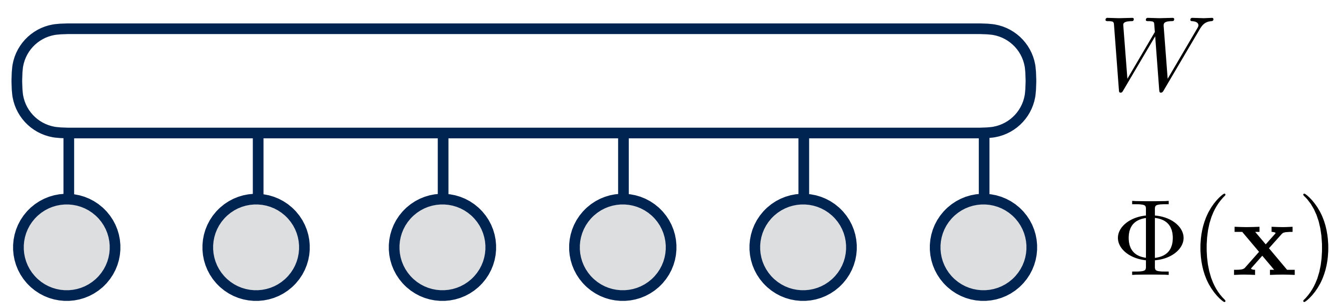
Construct decision function

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



Construct decision function

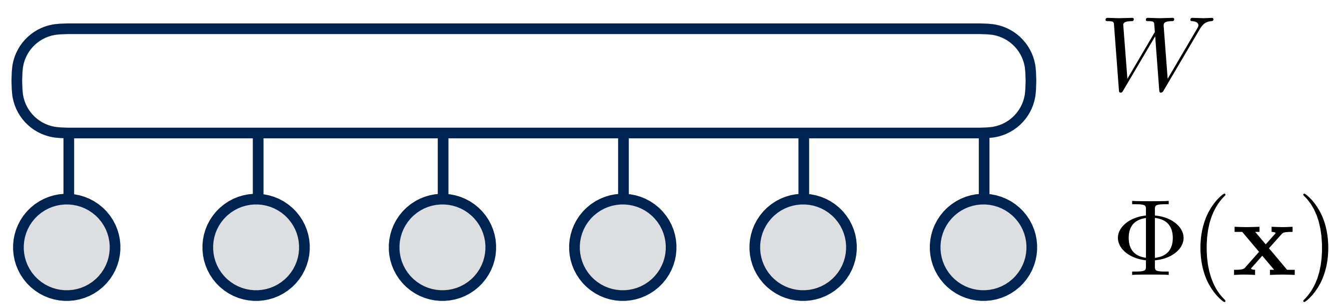
$$f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})$$

$f(\mathbf{x}) =$ 

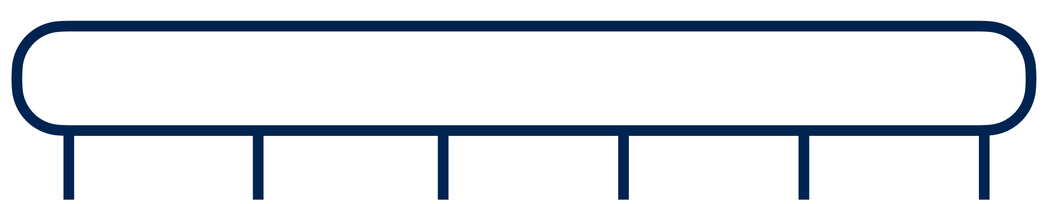
Construct decision function

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

$f(\mathbf{x}) =$



$W =$

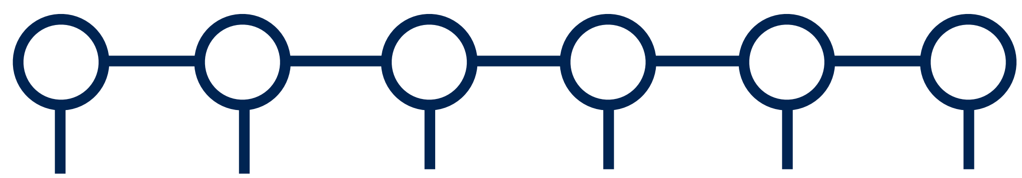


Main approximation

$W =$

order-N tensor

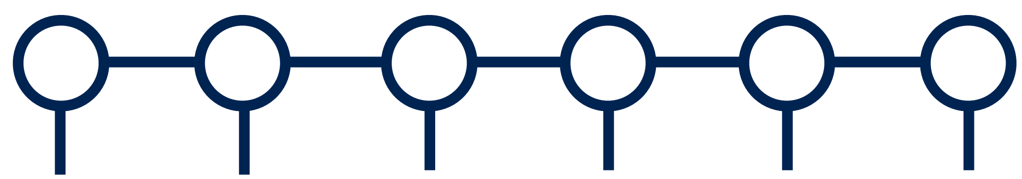
$\approx$



matrix product state (MPS)

Main approximation

$W =$  *order-N tensor*

$\approx$  *matrix product state (MPS)*

$\Big(\approx$  *PEPS* $\Big)$

Linear scaling

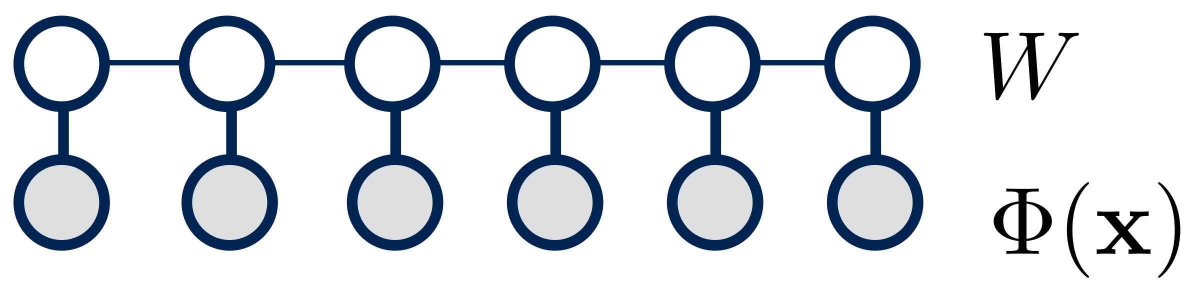
Can use algorithm similar to DMRG to optimize

Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$ 

Linear scaling

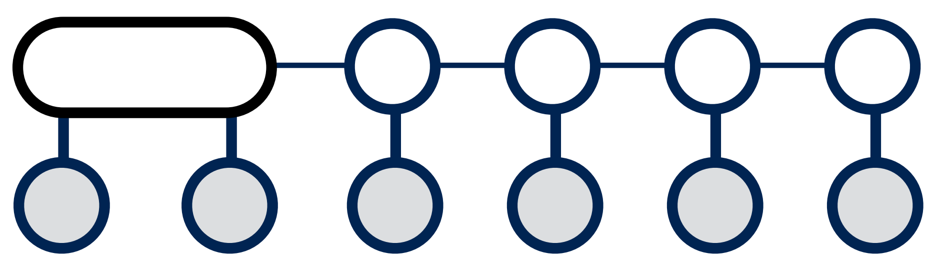
Can use algorithm similar to DMRG to optimize

Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

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

Linear scaling

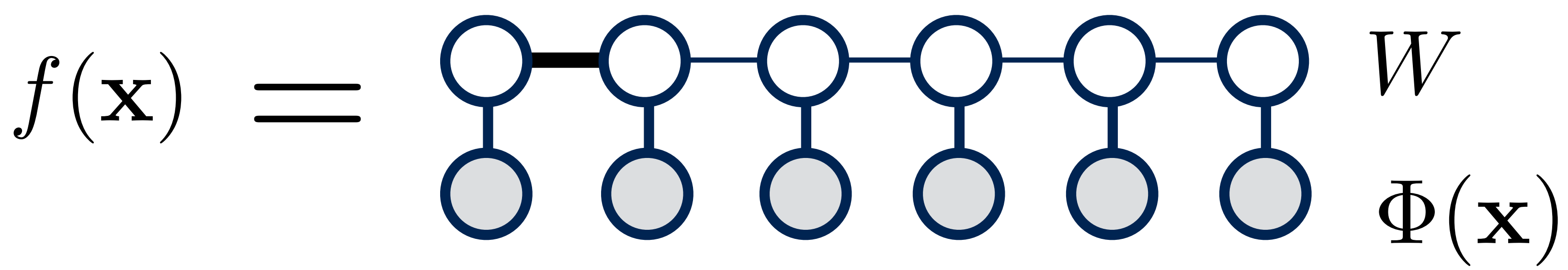
Can use algorithm similar to DMRG to optimize

Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension



Linear scaling

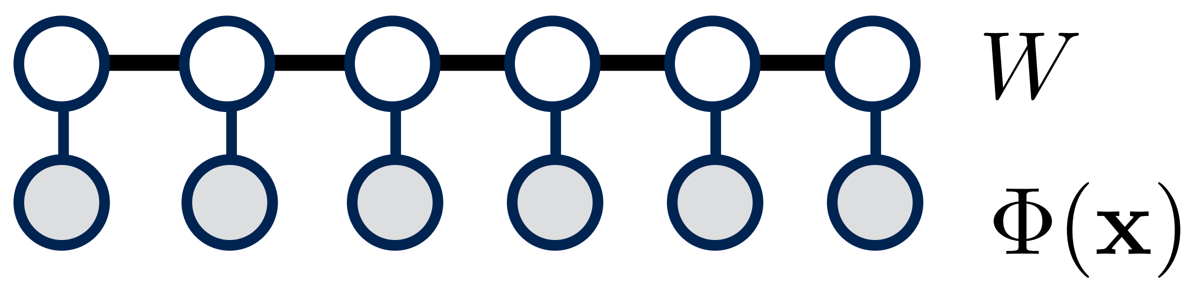
Can use algorithm similar to DMRG to optimize

Scaling is $N \cdot N_T \cdot m^3$

$N =$ size of input

$N_T =$ size of training set

$m =$ MPS bond dimension

$f(\mathbf{x}) =$ 

Why should this work at all?

Linear classifier $f(\mathbf{x}) = V \cdot \mathbf{x}$ exactly m=2 MPS

$$W = \begin{bmatrix} V_0 & 1 \end{bmatrix} \begin{bmatrix} \hat{1} & 0 \\ \hat{V}_1 & \hat{1} \end{bmatrix} \begin{bmatrix} \hat{1} & 0 \\ \hat{V}_2 & \hat{1} \end{bmatrix} \begin{bmatrix} \hat{1} & 0 \\ \hat{V}_3 & \hat{1} \end{bmatrix} \cdots$$

$$\hat{1} = [1 \;\; 0]$$

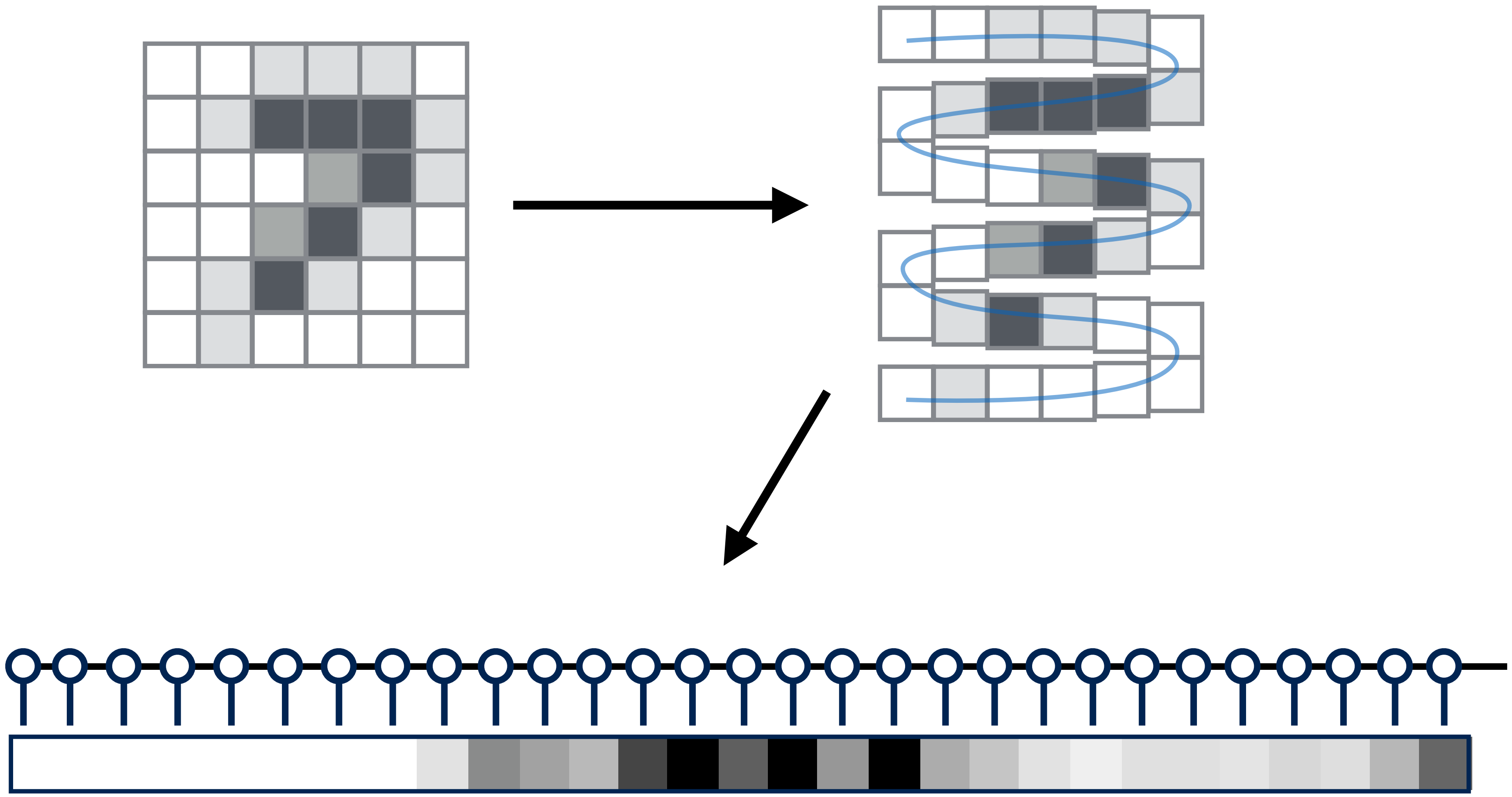
$$\hat{V}_j = [0 \;\; V_j]$$

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

$$\phi^{s_j}(x_j) = [1\,,\; x_j]$$

Novikov, Trofimov, Oseledets, arxiv:1605.03795

Experiment: handwriting classification (MNIST)



Train to 99.95% accuracy on 60,000 training images

Obtain **99.03%** accuracy on 10,000 test images (only 97 incorrect)

Stoudenmire, Schwab, arxiv:1605.05775

Papers using tensor network machine learning

Expressivity & priors of TN based models

- *Levine et al., "**Deep Learning and Quantum Entanglement: Fundamental Connections with Implications to Network Design**" arxiv:1704.01552*
- *Cohen, Shashua, "**Inductive Bias of Deep Convolutional Networks through Pooling Geometry**" arxiv:1605.06743*
- *Cohen et al., "**On the Expressive Power of Deep Learning: A Tensor Analysis**" arxiv:1509.05009*

Generative Models

- *Han et al., "**Unsupervised Generative Modeling Using Matrix Product States**" arxiv:1709.01662*
- *Sharir et al., "**Tractable Generative Convolutional Arithmetic Circuits**" arxiv:1610.04167*

Supervised Learning

- *Novikov et al., "**Expressive power of recurrent neural networks**", arxiv:1711.00811*
- *Liu et al., "**Machine Learning by Two-Dimensional Hierarchical Tensor Networks: A Quantum Information Theoretic Perspective on Deep Architectures**", arxiv:1710.04833*
- *Stoudenmire, Schwab, "**Supervised Learning with Quantum-Inspired Tensor Networks**", arxiv:1605.05775*
- *Novikov et al., "**Exponential Machines**", arxiv: 1605.03795*

Related uses of tensor networks

Compressing weights of neural nets (& other models)

Yu et al., Advances in Neural Information Processing (2017), arxiv:1711.00073

Izmailov et al., arxiv:1710.07324 (2017)

Yang et al., arxiv:1707.01786 (2017)

Garipov et al., arxiv:1611.03214 (2016)

Novikov et al., Advances in Neural Information Processing (2015) (arxiv:1509.06569)

Large scale linear algebra (PCA/SVD)

Lee, Cichocki, arxiv: 1410.6895 (2014)

Feature extraction & tensor completion

Bengua et al., arxiv:1606.01500, arxiv:1607.03967, arxiv:1609.04541 (2016)

Phien et al., arxiv:1601.01083 (2016)

Bengua et al., IEEE Congress on Big Data (2015)

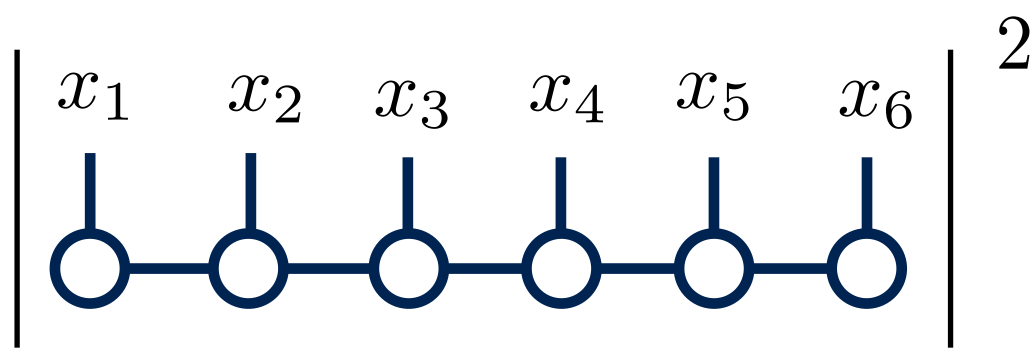
Tensor Network Machine Learning Studies

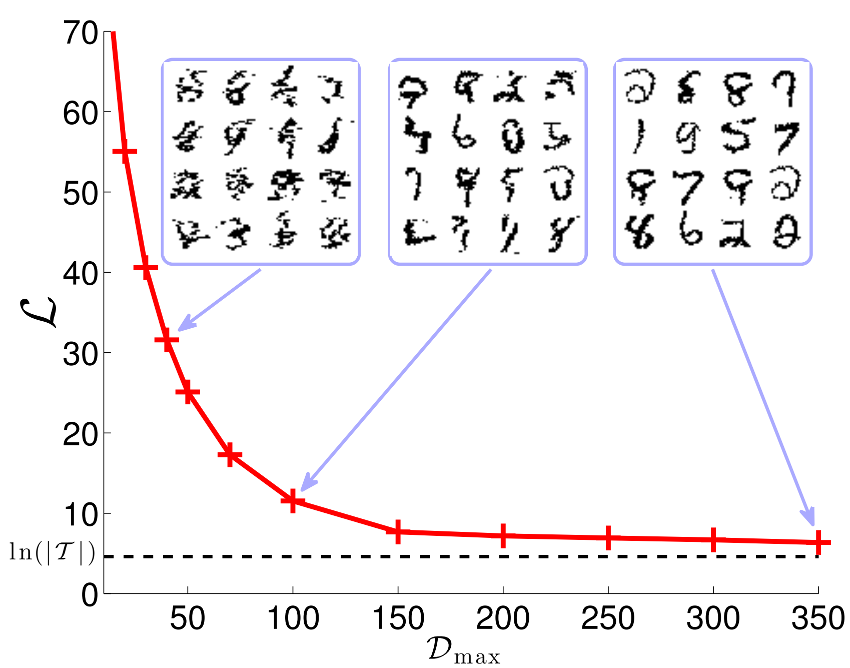
Unsupervised Generative Modeling Using MPS

Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, Pan Zhang

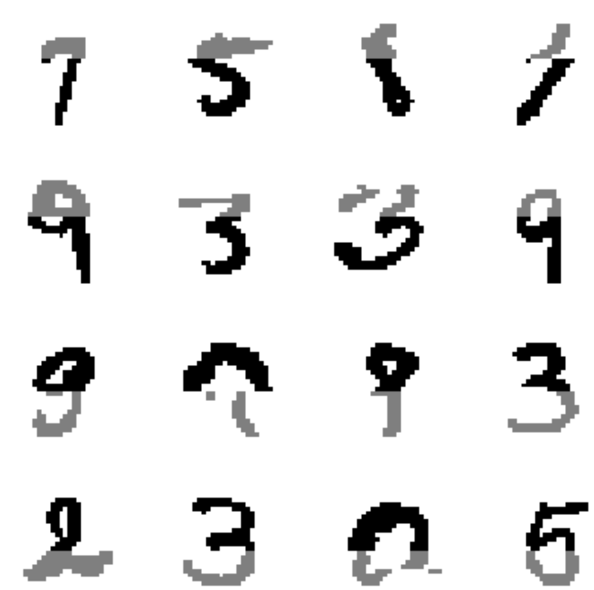
- Map data to product state, tensor network weights
- <u>Squared</u> output is probability – "Born machine"
- "Perfect" sampling (no autocorrelation)

$p(\mathbf{x}) =$





Negative Log-Likelihood

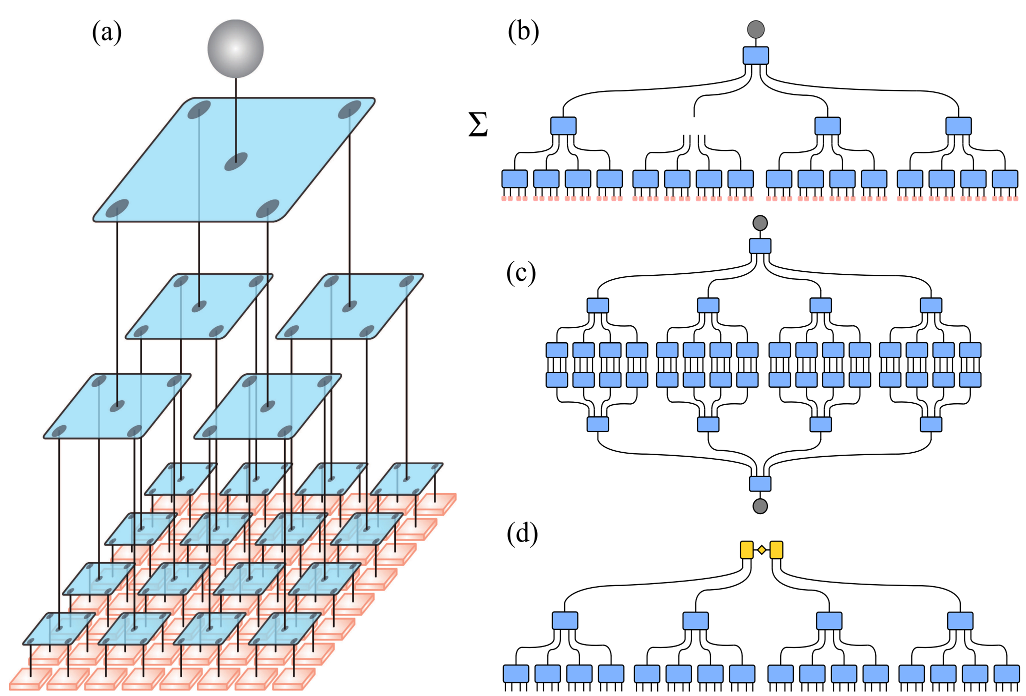


Reconstructing Testing Images

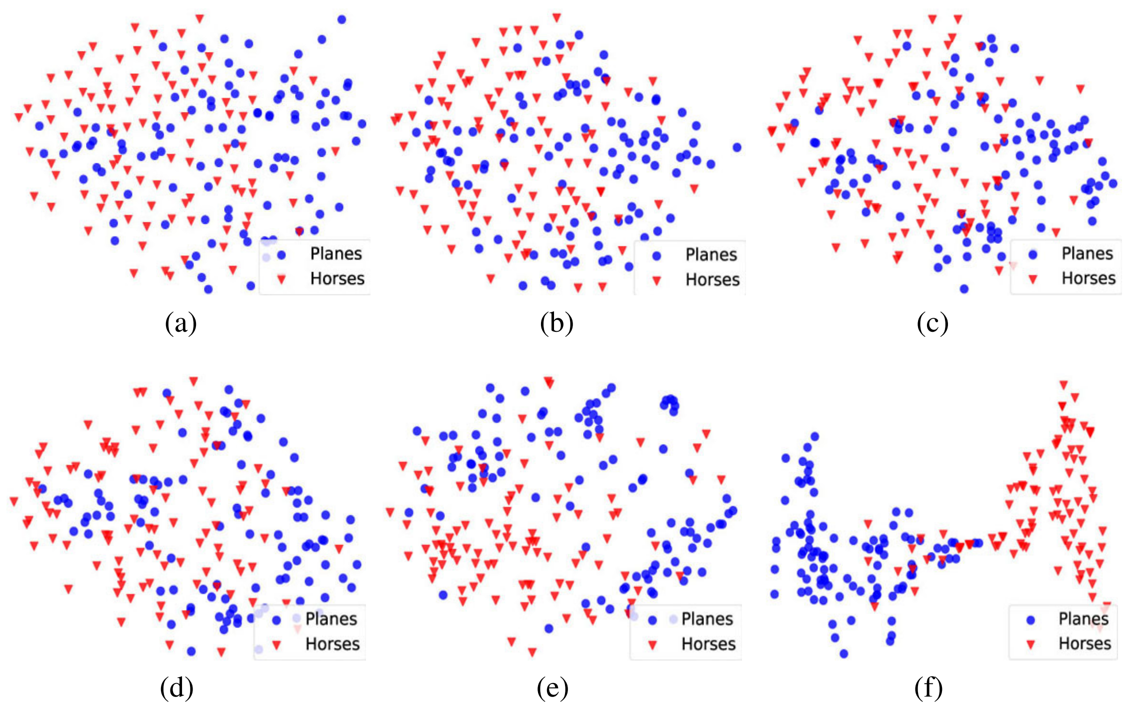
Machine Learning By Hierarchical Tensor Networks...

Ding Liu, Shi-Ju Ran, Peter Wittek, Cheng Peng, Raul Blazquez Garcia, Gang Su, Maciej Lewenstein

- Supervised learning with tree tensor networks
- Tests on MNIST, CIFAR-10
- Studied properties of the trained model (feature representations, entanglement)



Model Architecture

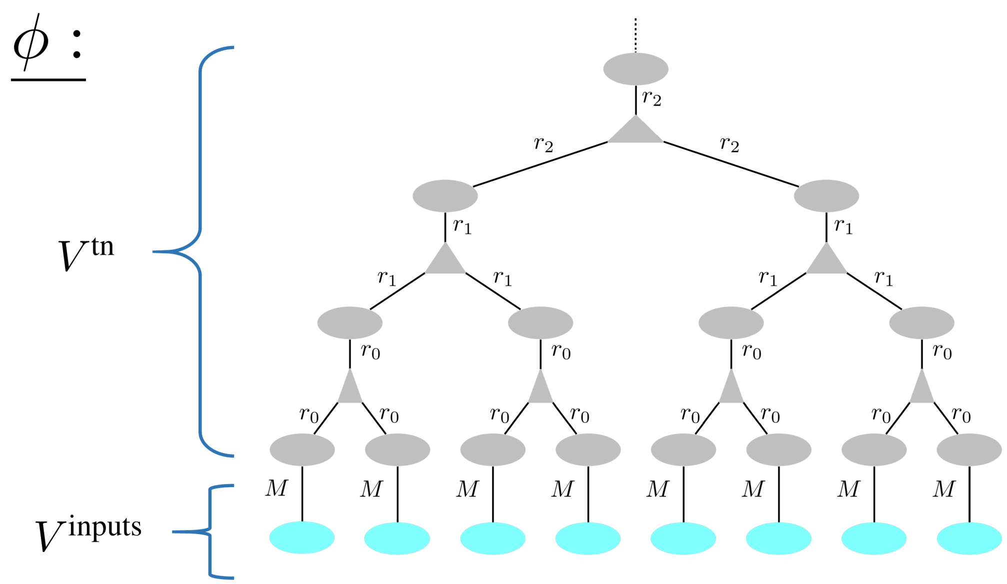


Data Representation at Different Scales

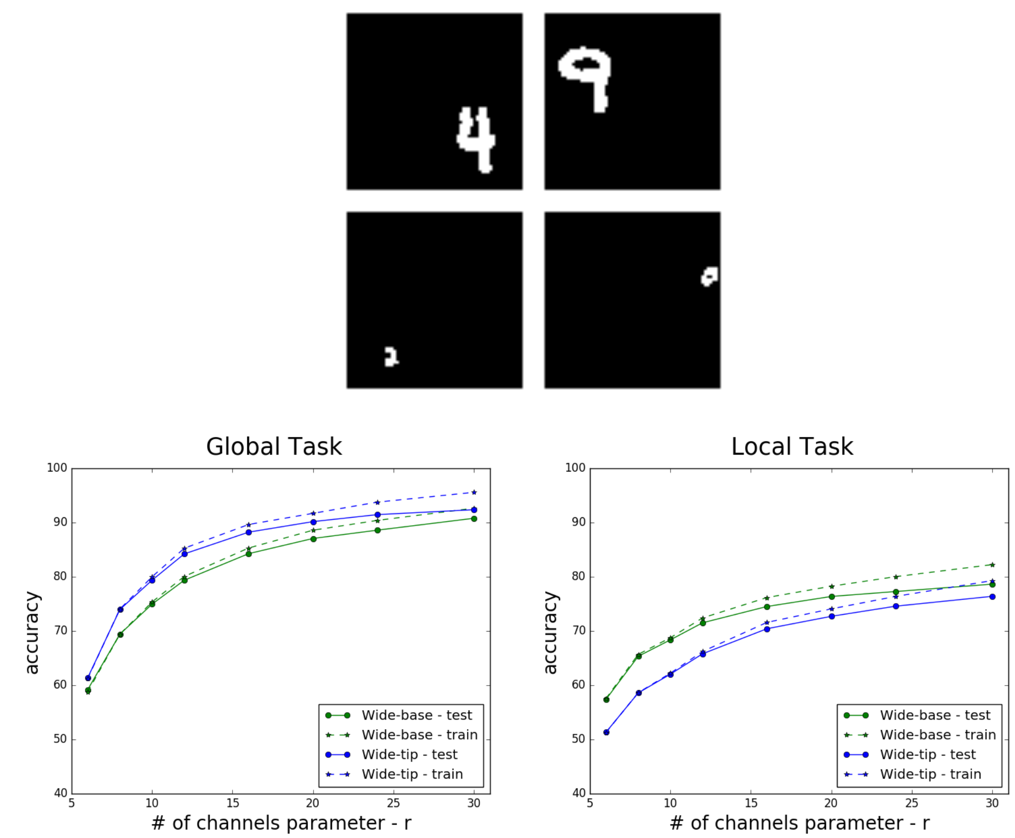
Deep Learning and Quantum Entanglement...

Yoav Levine, David Yakira, Nadav Cohen, Amnon Shashua

- "ConvAC" deep neural net = tree tensor network
- Tensor network rank as capacity of model
- Experiment on "inductive bias" of model architecture



Tree Network as a Deep Neural Net

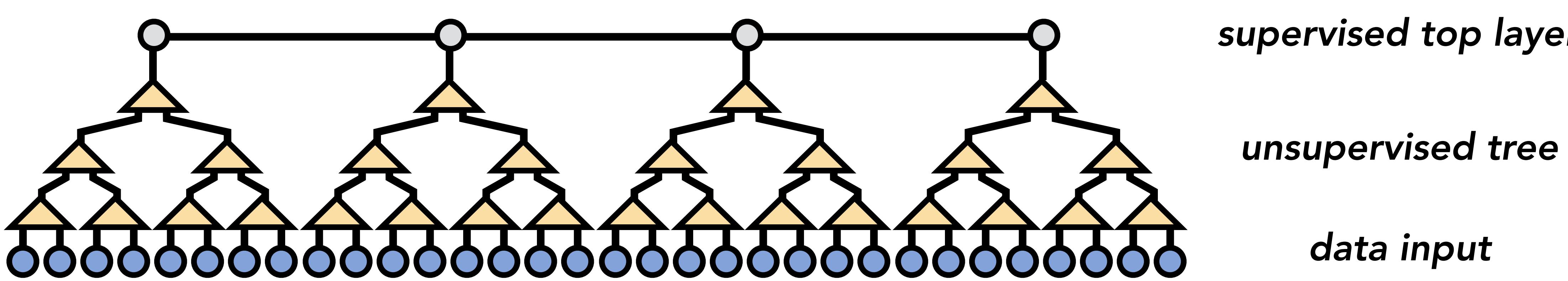


Inductive Bias Experiment

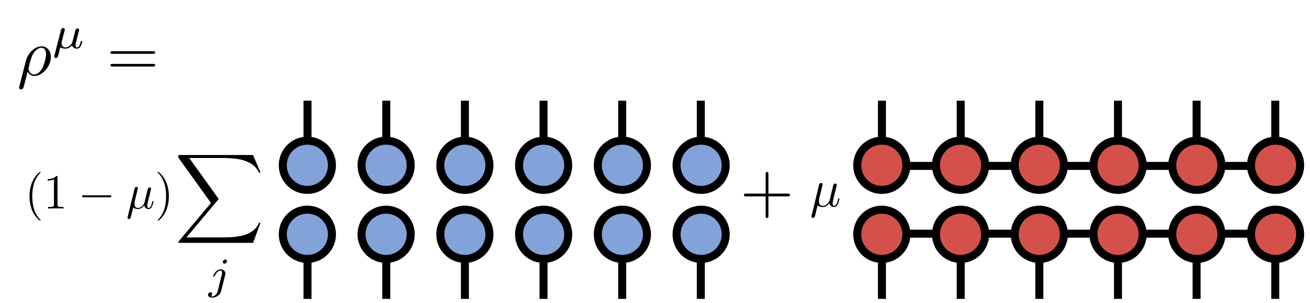
Learning Relevant Features of Data...

E.M. Stoudenmire

- Unsupervised determination of tree tensor network (compress data)
- Supervised training of top layer
- Excellent performance with "features" determined by tree tensors



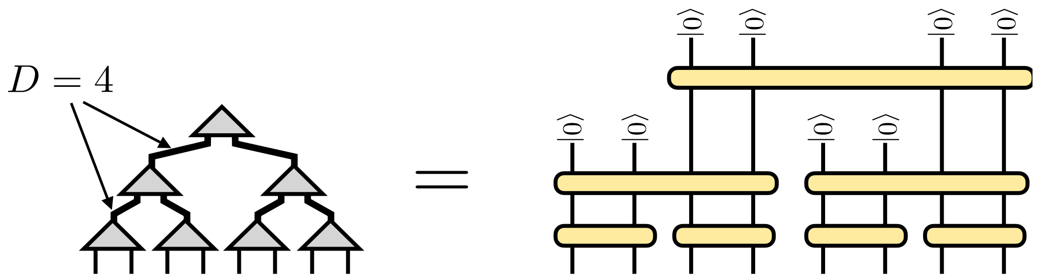
89% accuracy on
Fashion MNIST data set



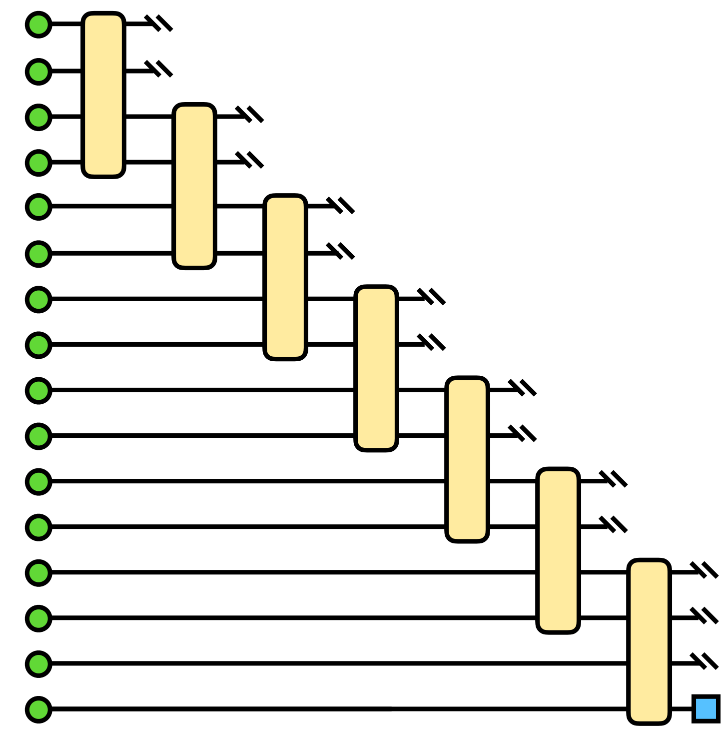
mixed training
supervised / unsupervised

Tensor Network Learning on Quantum Computers

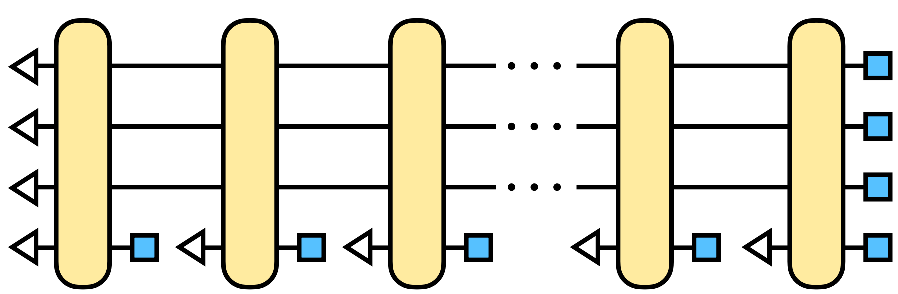
Tensor networks equivalent to quantum circuits



Proposal for learning based on MPS:



Qubit-efficient
generative model:



Huggins, Patil, Whaley, Stoudenmire, arxiv:1803.11537

Grant, Benedetti, et al., arxiv:1804.03680

Conclusions & Future Directions

- Quantum-inspired tensor networks an intriguing alternative to traditional machine learning models

- Better scaling, interesting algorithms, opportunities for theoretical insights

- Continue pushing interpretability, algorithms

- Promising as a framework for machine learning with quantum computing

Learning Relevant Features of Data

For a model $f(\mathbf{x}) = W \cdot \Phi(\mathbf{x})$

Given training data $\{\mathbf{x}_j\}$

Can show optimal W is of the form

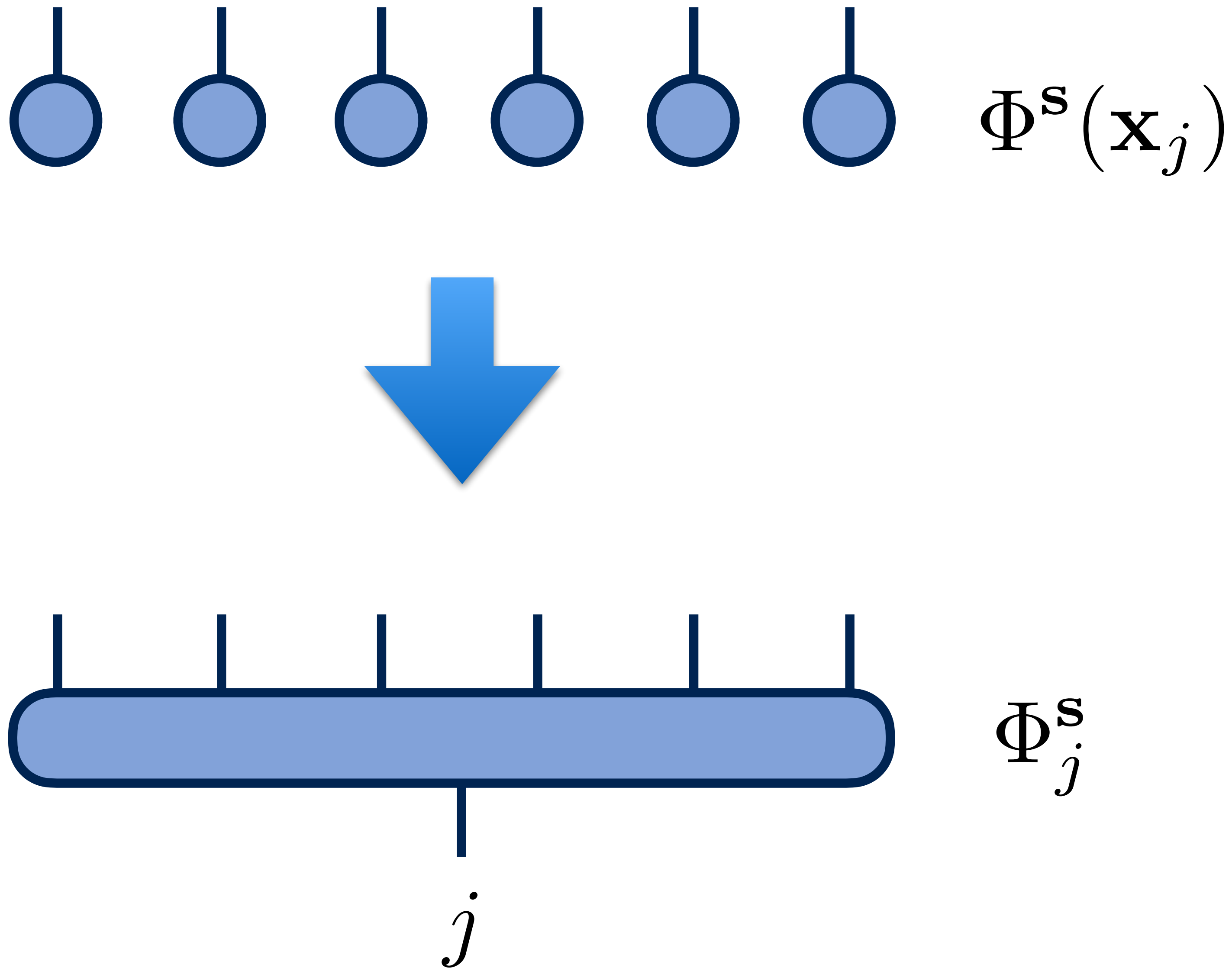
$$W = \sum_j \alpha_j \, \Phi(\mathbf{x}_j)$$

Holds for wide variety of cost functions / tasks

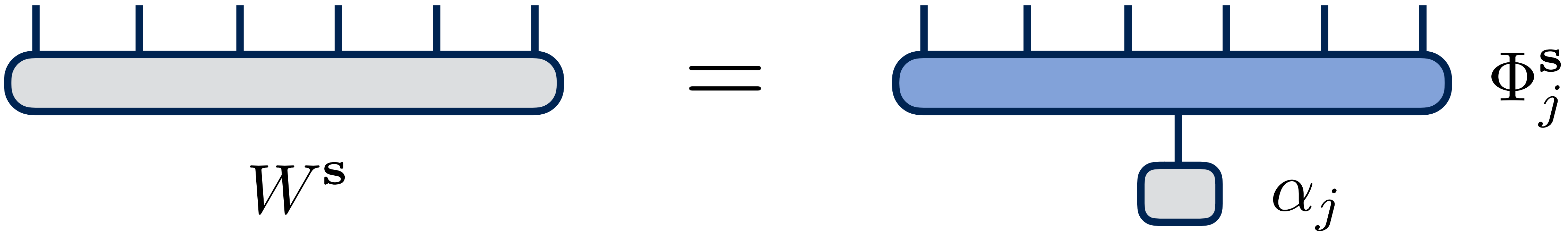
"representer theorem"

Schölkopf, Smola, Müller, Neural Comp. 10, 1299 (1998)

View $\Phi^{\mathbf{s}}(\mathbf{x}_j) = \Phi^{\mathbf{s}}_j$ as a tensor

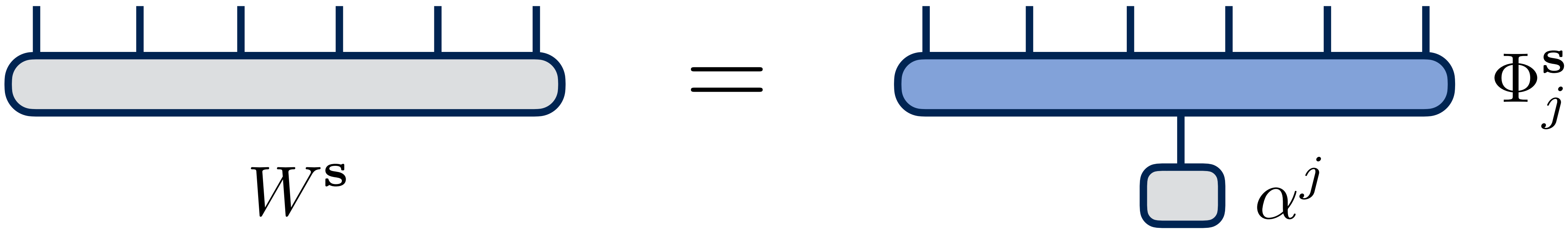


Representer theorem says

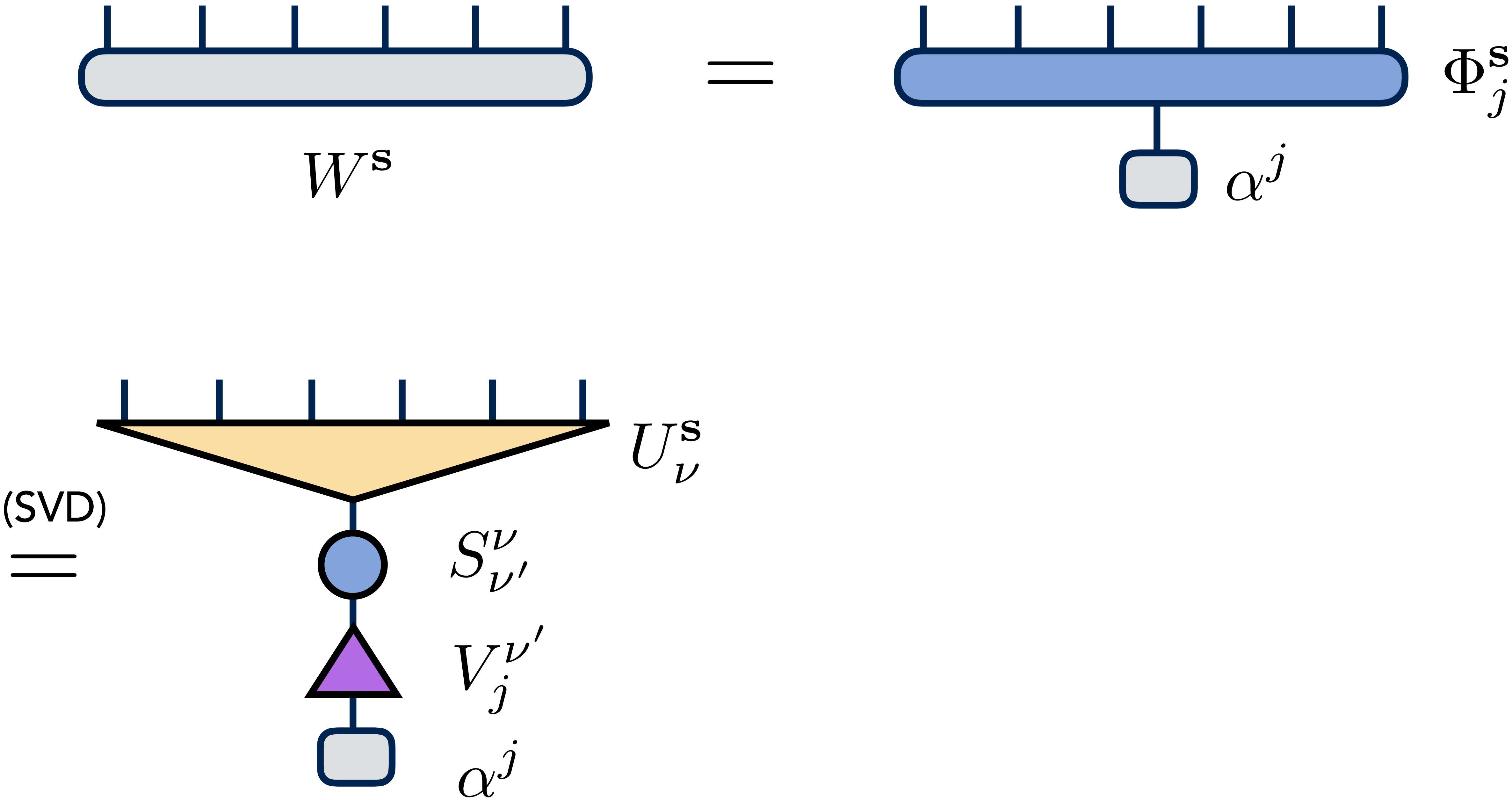


Really just says weights in the span of $\{\Phi_j^{\mathbf{s}}\}$

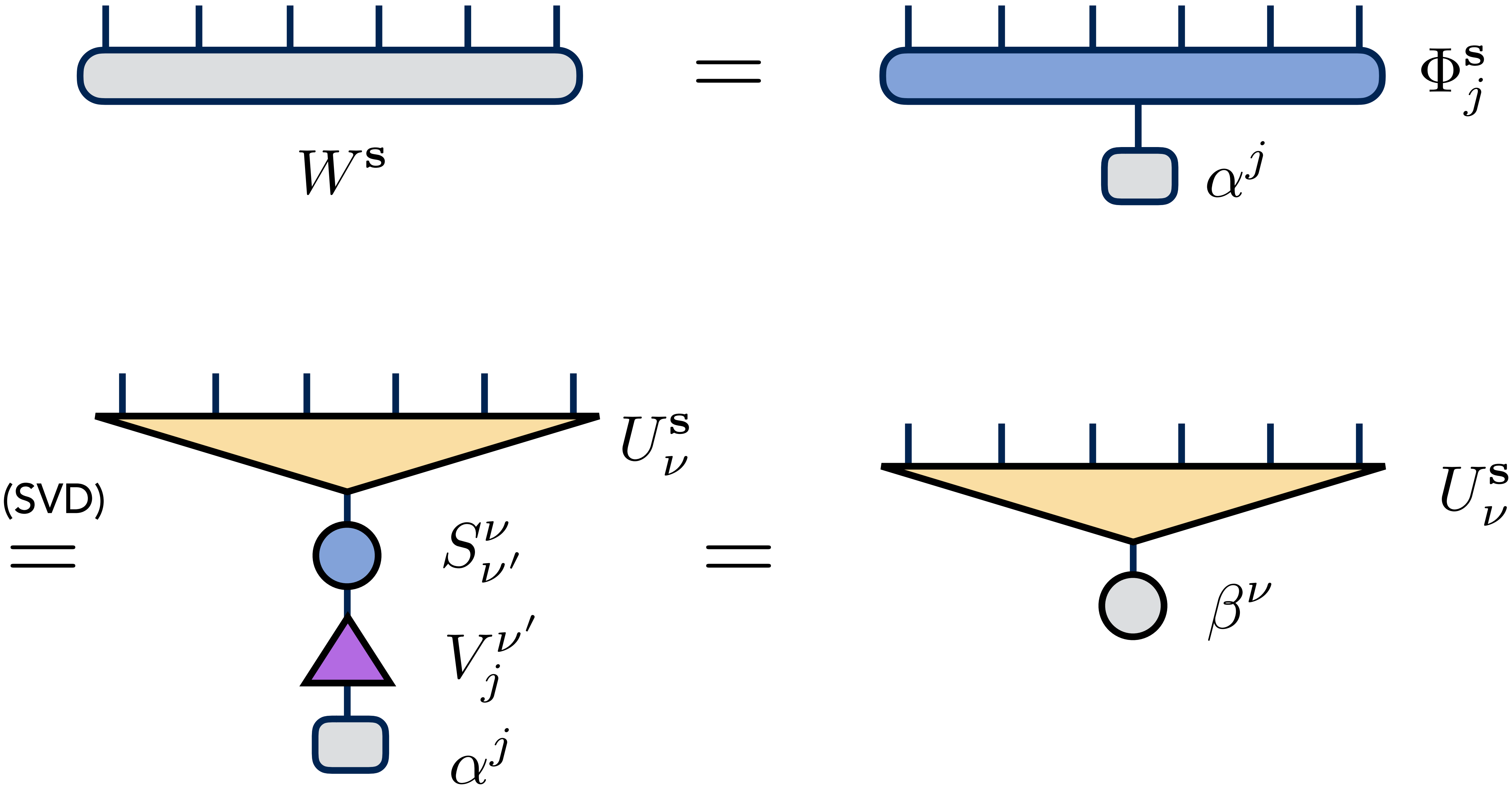
Can choose any basis for span of $\{\Phi_j^{\mathbf{s}}\}$



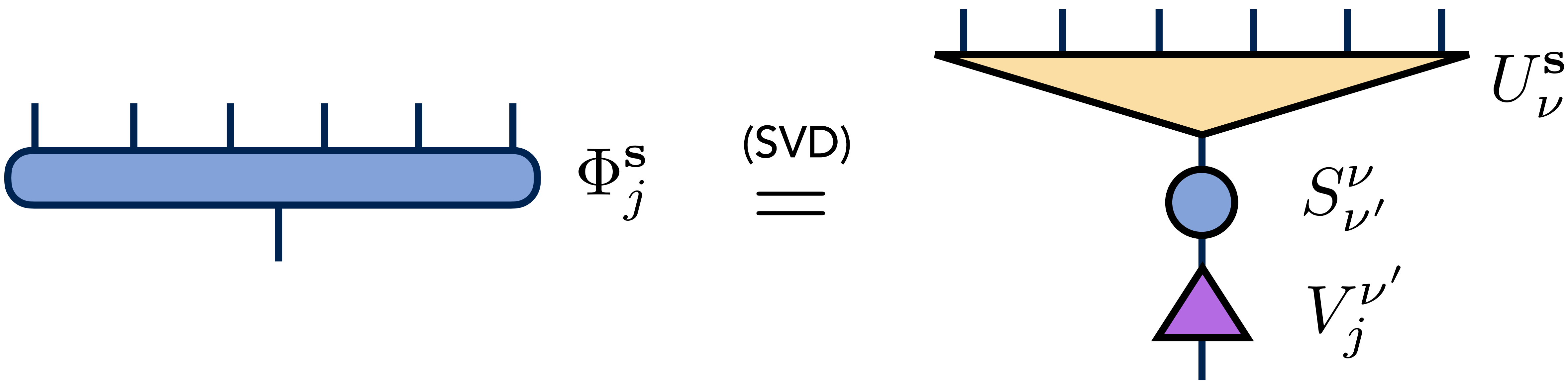
Can choose any basis for span of $\{\Phi^{\mathbf{s}}_j\}$



Can choose any basis for span of $\{\Phi_j^{\mathbf{s}}\}$



Why switch to $U_\nu^{\mathbf{s}}$ basis?



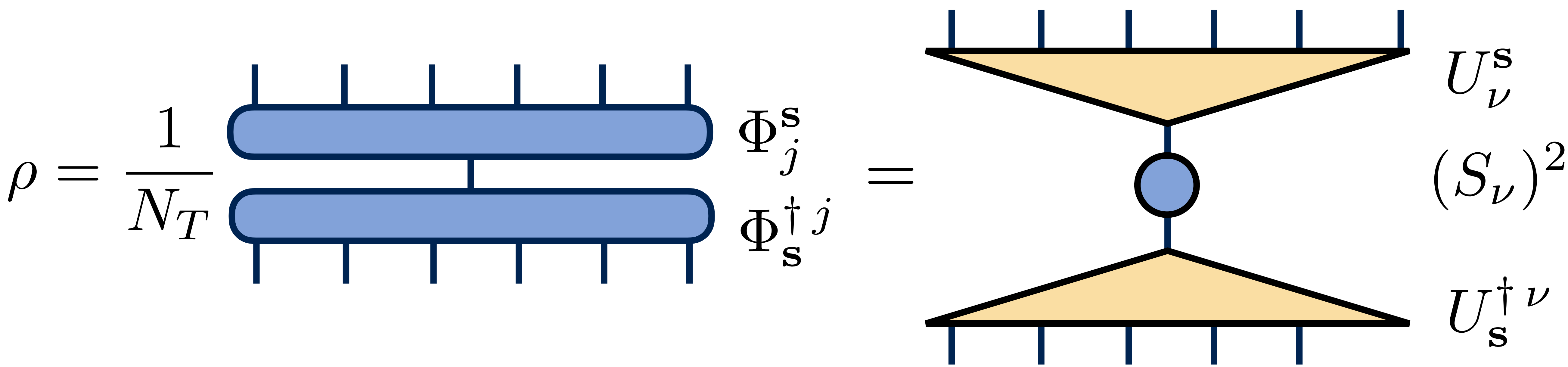
Orthonormal basis

Can discard basis vectors corresponding to small s. vals.

Can compute $U_\nu^{\mathbf{s}}$ fully or partially using *tensor networks*

Computing $U_\nu^{\mathbf{s}}$ efficiently

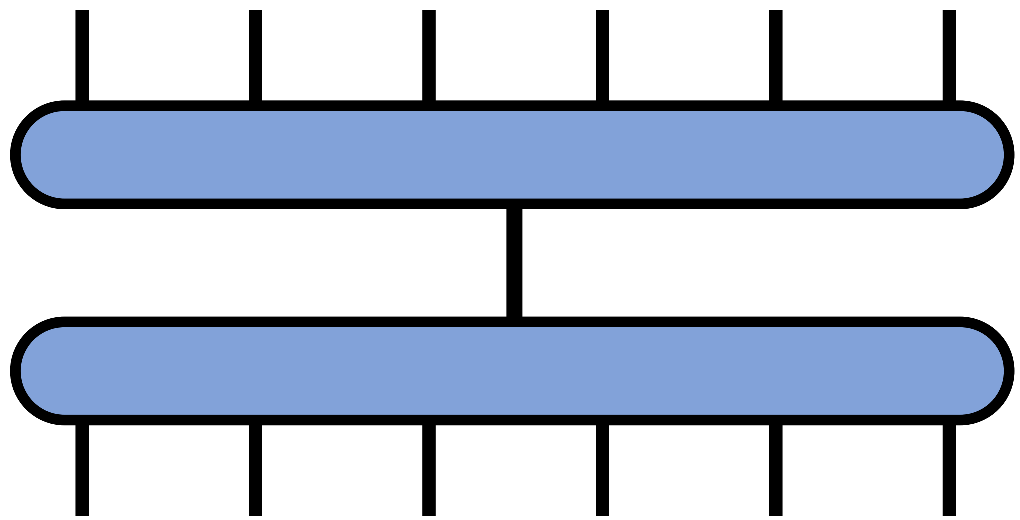
Define *feature space covariance matrix*
(similar to density matrix)



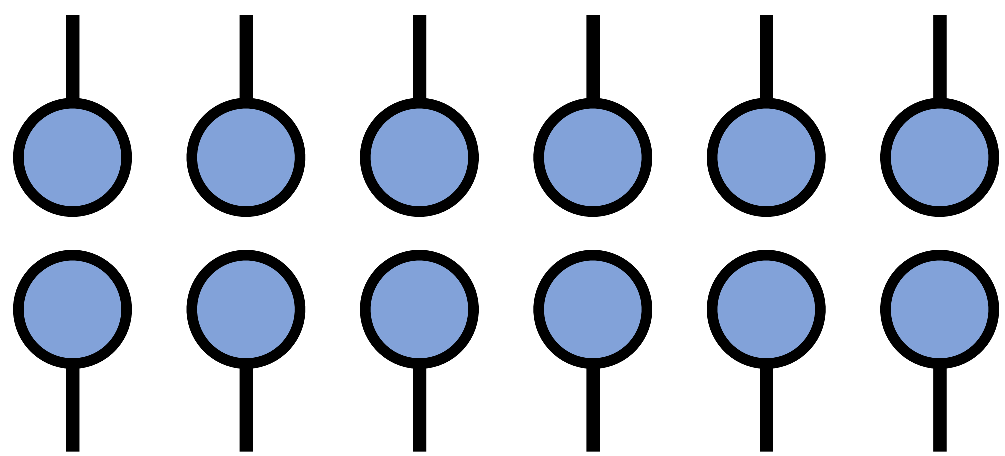
Strategy: compute $U_\nu^{\mathbf{s}}$ iteratively as a layered (tree) tensor network

For efficiency, exploit product structure of Φ

$$\rho = \Phi\Phi^\dagger = \frac{1}{N_T}$$



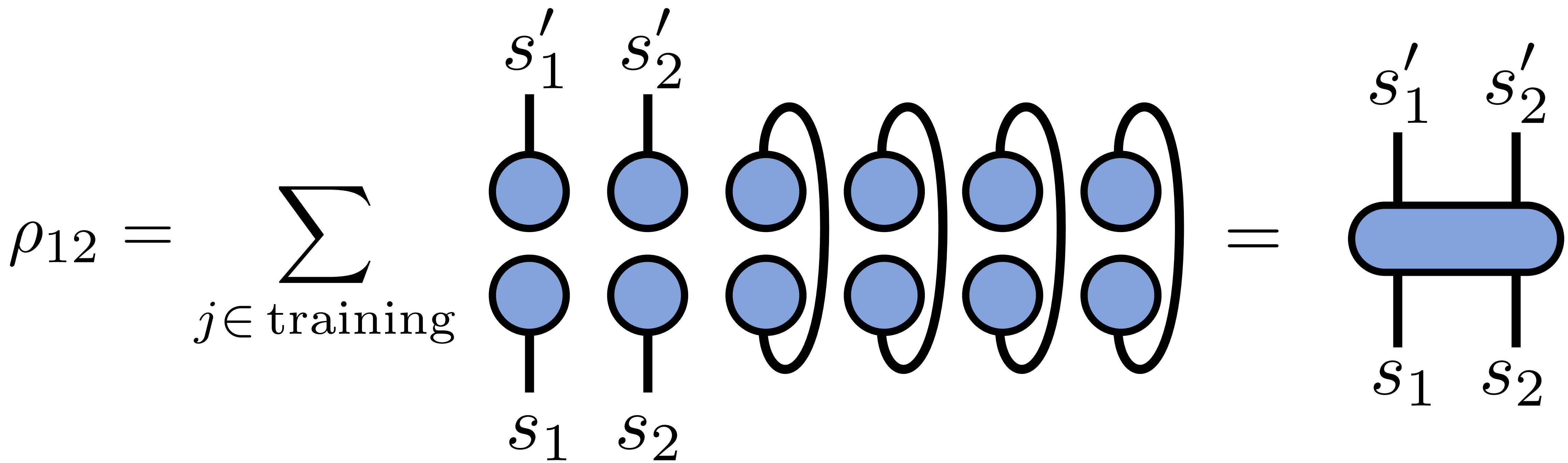
$$= \frac{1}{N_T}\sum_{j=1}^{N_T}$$

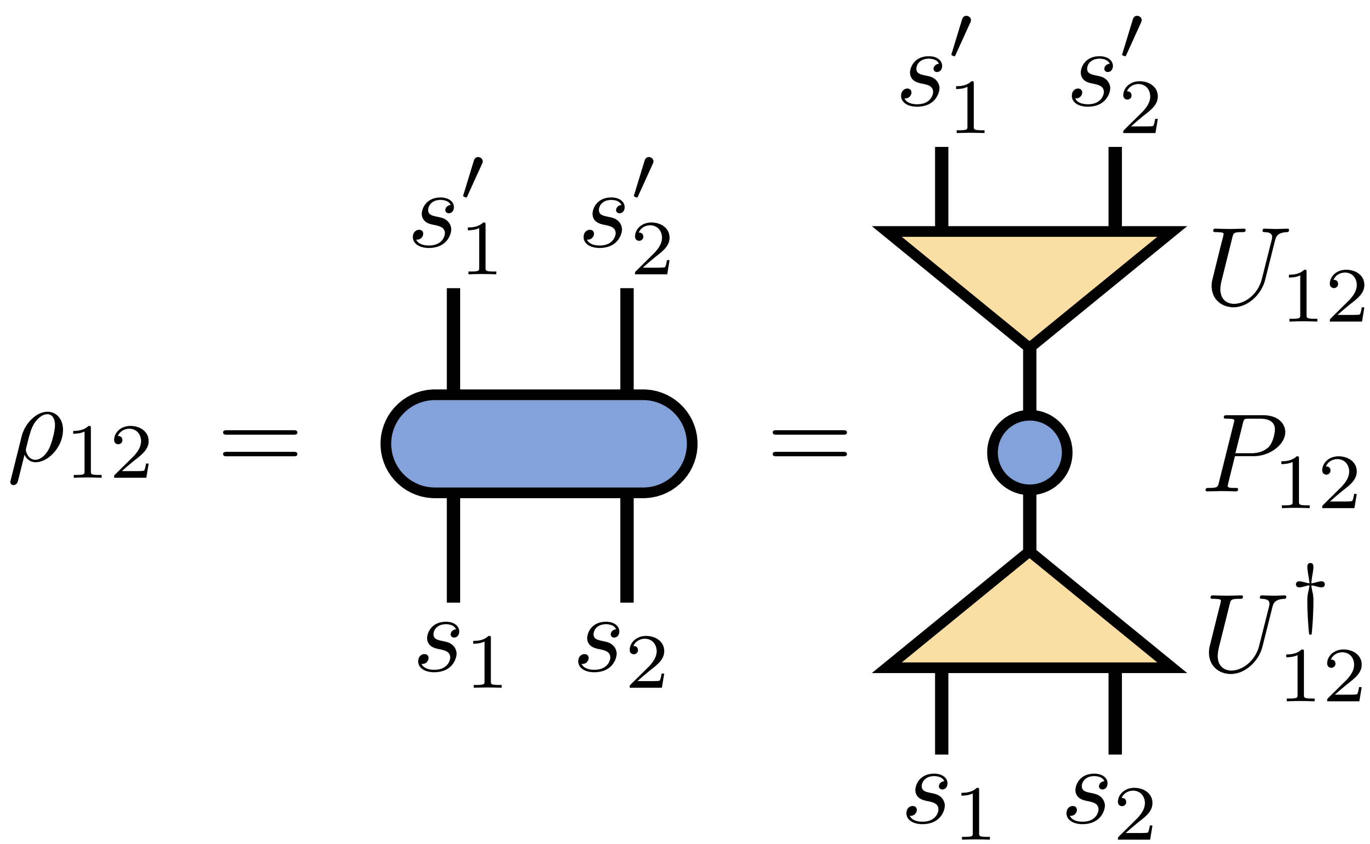


$\Phi(\mathbf{x}_j)$

$\Phi^\dagger(\mathbf{x}_j)$

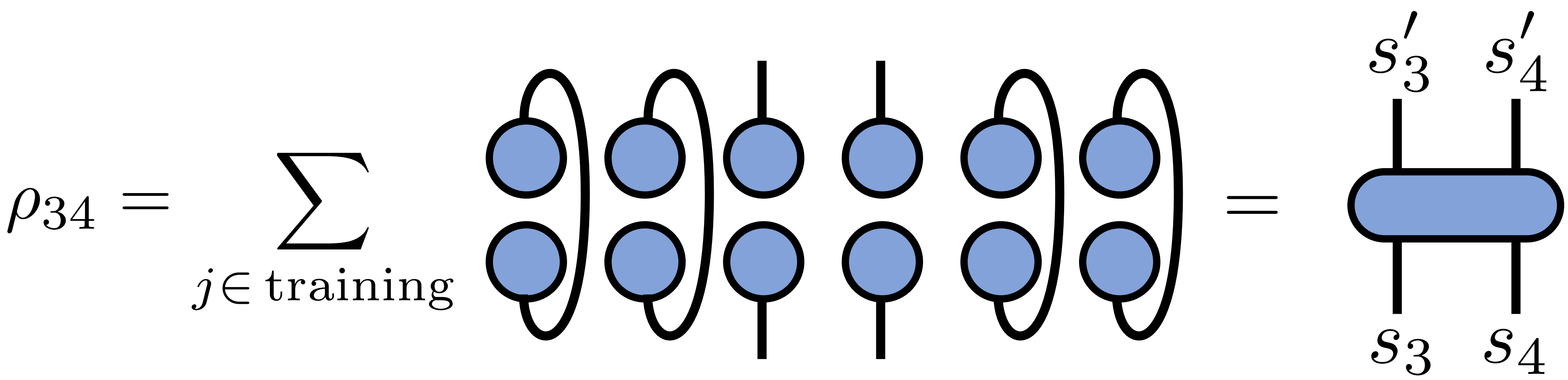
Compute tree tensors from reduced matrices

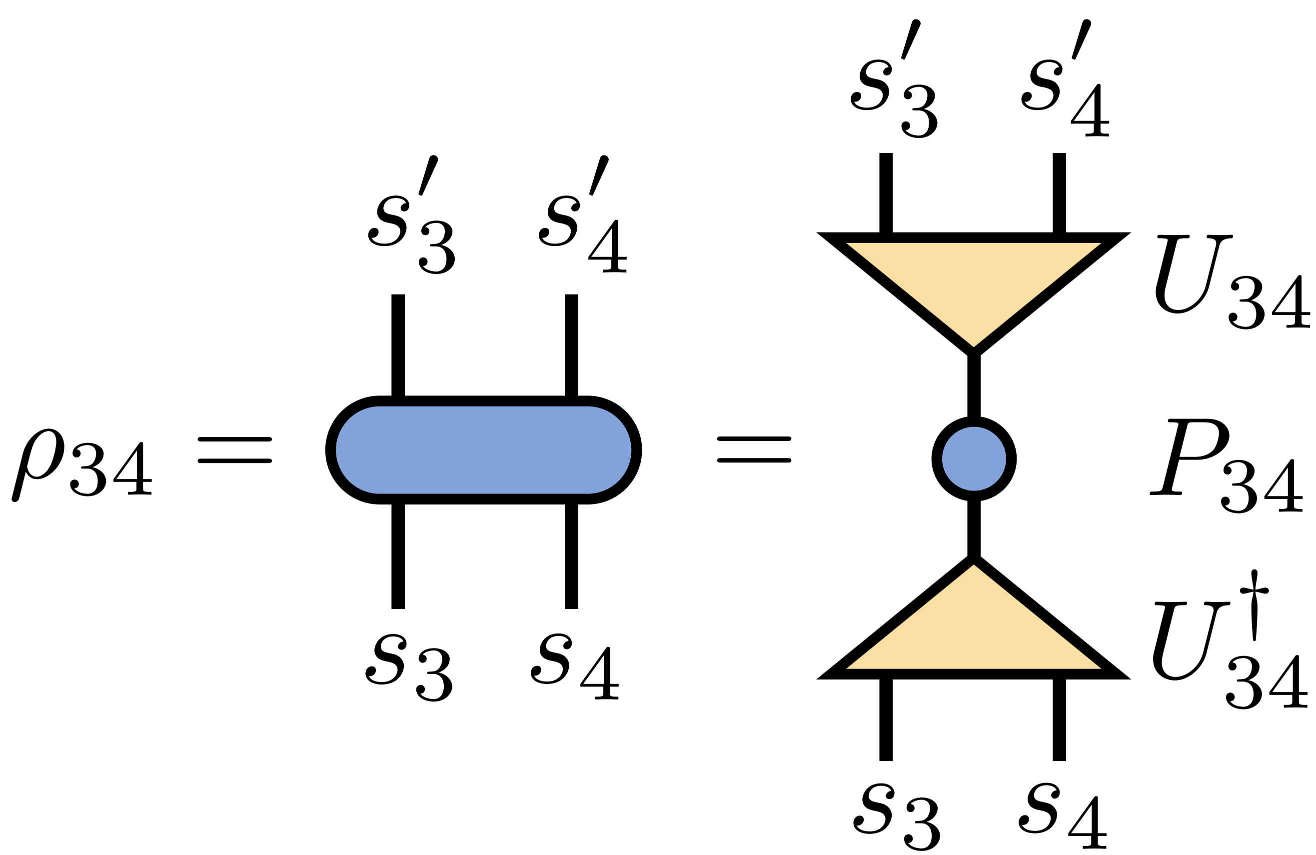




Truncate small eigenvalues

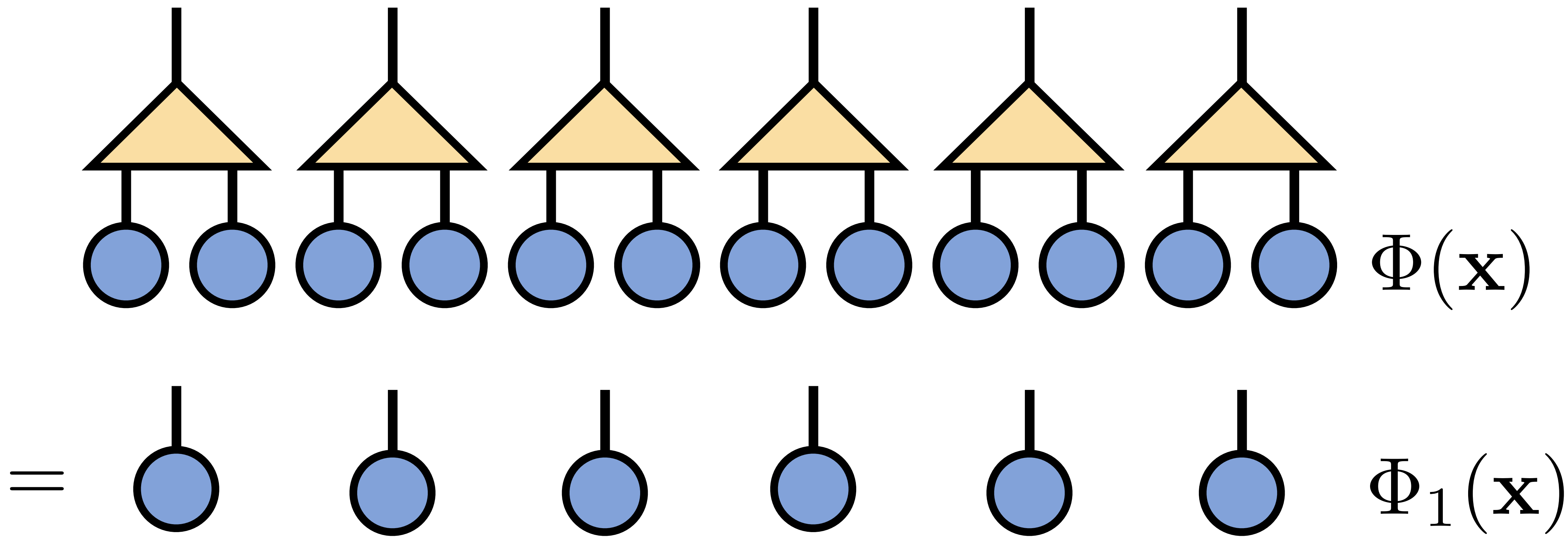
Compute tree tensors from reduced matrices





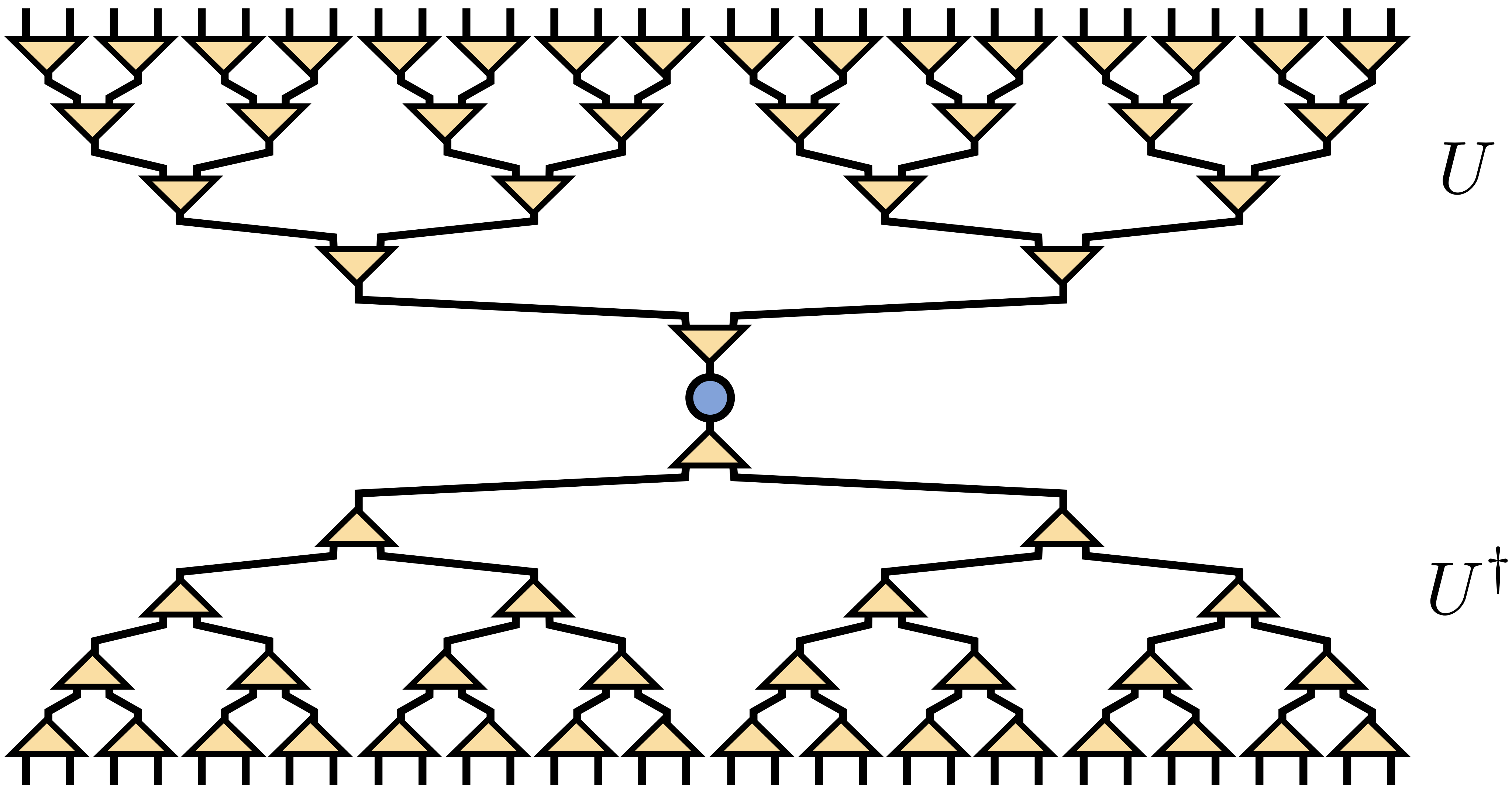
Truncate small eigenvalues

Having computed a tree layer, rescale data



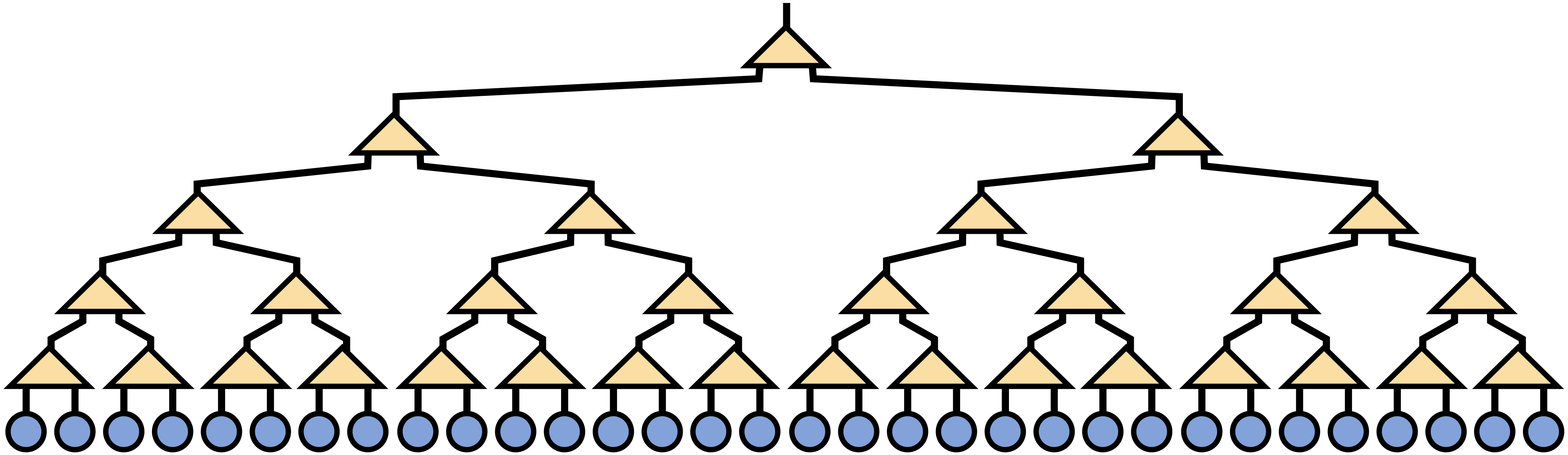
With all layers, have approximately diagonalized ρ

$\rho \simeq$



Equivalent to *kernel PCA*,
but linear scaling with size of data set

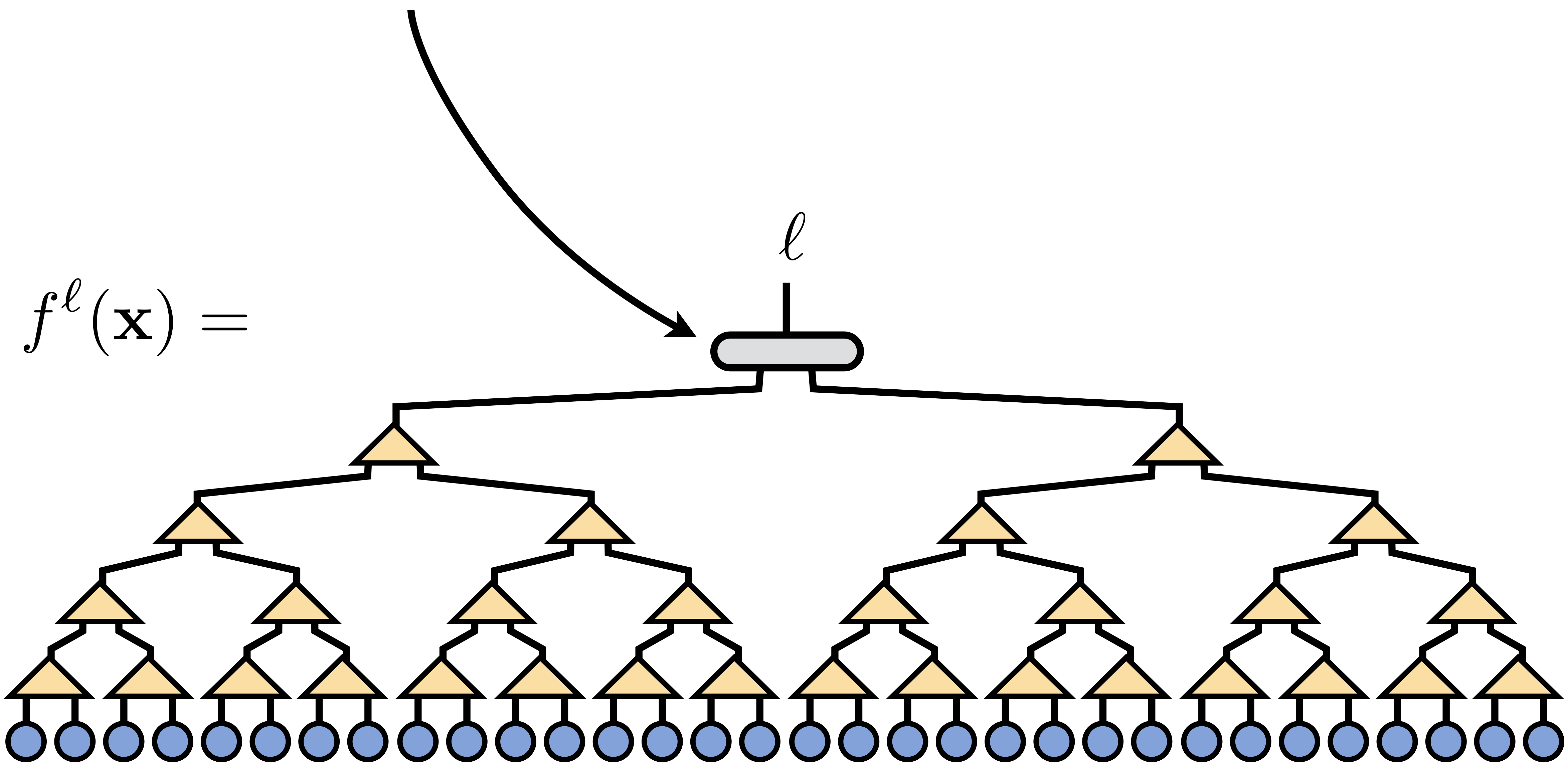
Can view as *unsupervised learning* of representation of training data



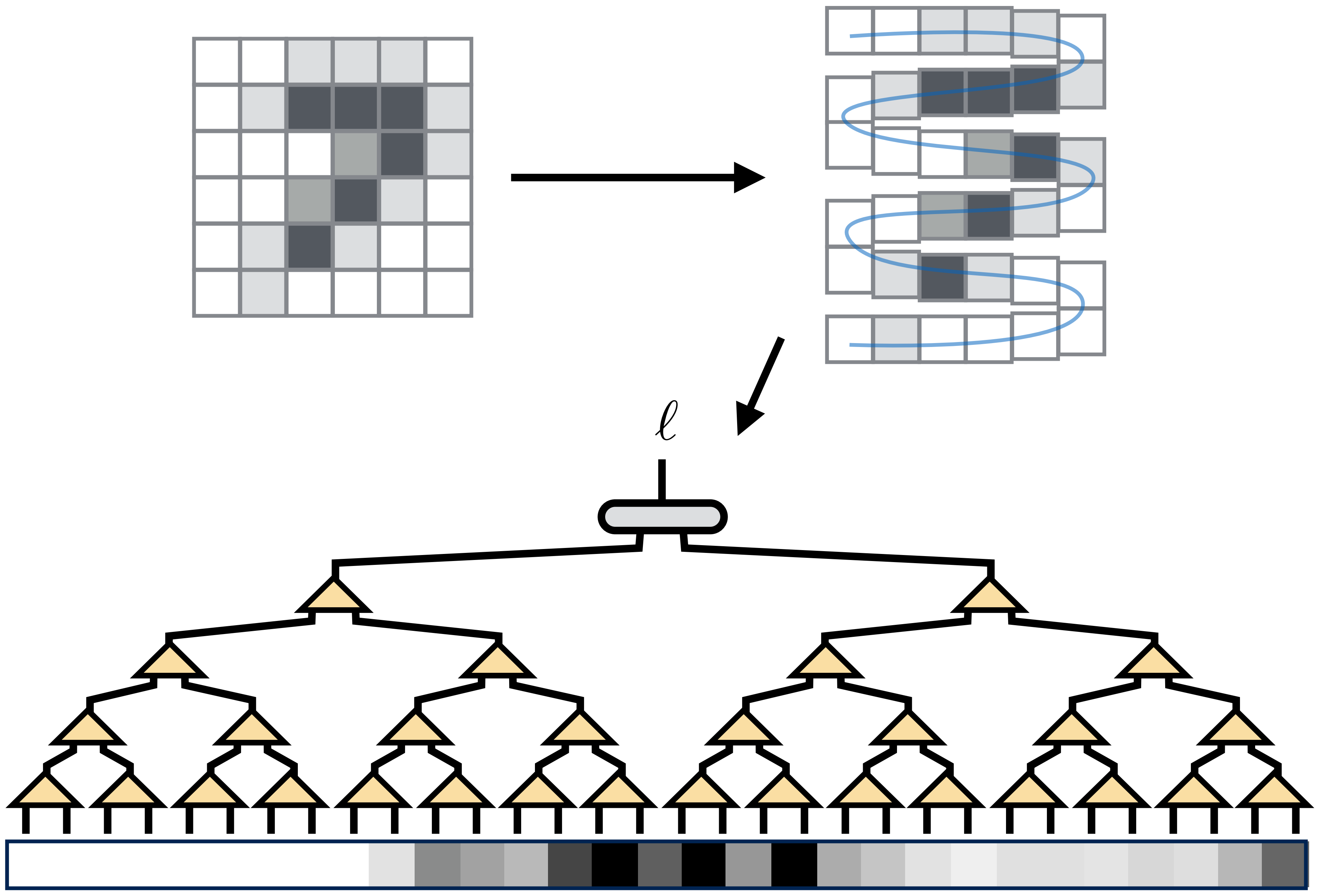
Use as starting point for supervised learning

Only train top tensor for supervised task

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



Experiment: handwriting classification (MNIST)



Cutoff 6×10^{-4} gave top indices sizes 328 and 444

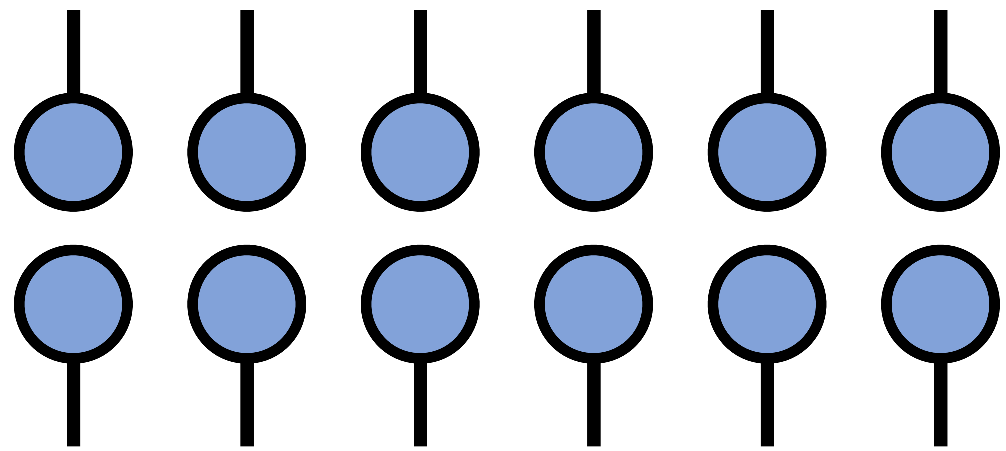
Training acc: 99.68% **Test acc:** 98.08%

Refinements and Extensions

No reason we must base tree around ρ

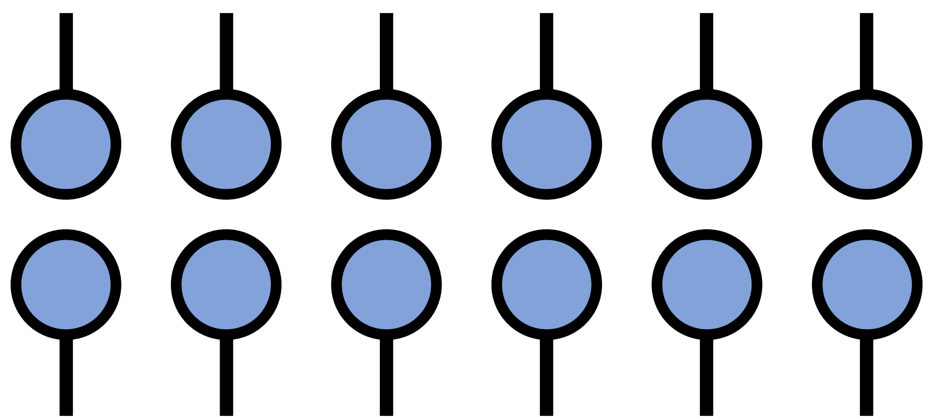
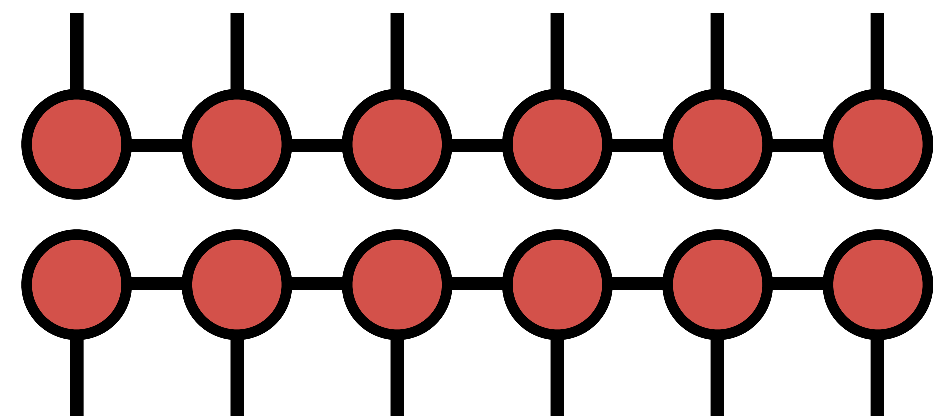
Could reweight based on importance of samples

$$\tilde{\rho} = \frac{1}{N_T} \sum_{j=1}^{N_T} w_j$$



Another idea is to mix in a "lower level" model trained on a given task (e.g. supervised learning)

$\rho^\mu =$

$(1-\mu)\sum_j$  $+\ \mu$ 

If $\mu = 1$, tree provides basis for provided weights

If $0 < \mu < 1$, tree is "enriched" by data set

Experiment: mixed correlation matrix for MNIST

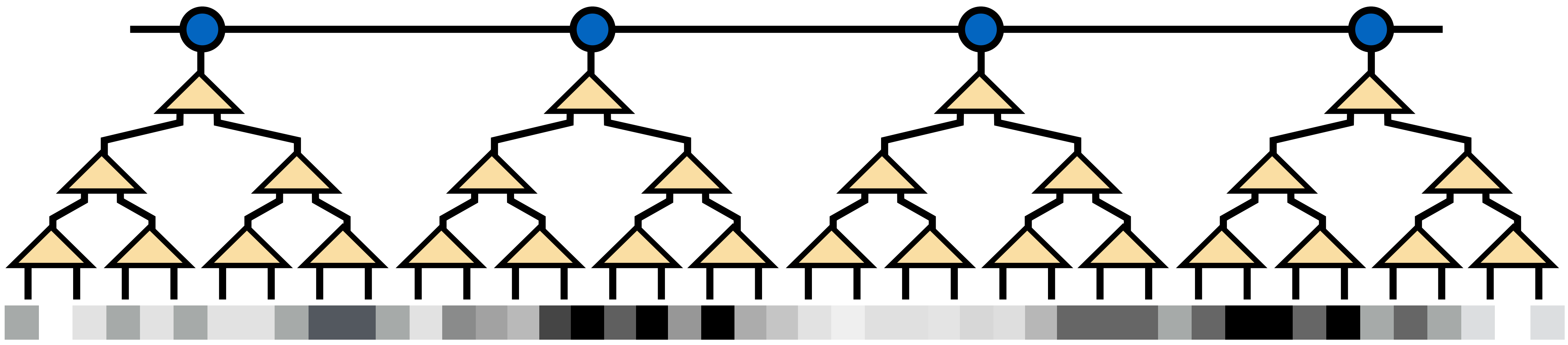
Using $\rho^{\mu} = (1-\mu)\rho + \mu \sum_{\ell} |W^{\ell}\rangle\langle W^{\ell}|$

with trial weights trained from a linear classifier and $\mu = 0.5$

Train acc: 99.798% **Test acc:** 98.110%
Top indices of size 279 and 393.

Comparable performance to unmixed case with top index sizes 328 and 444

Also no reason to build entire tree



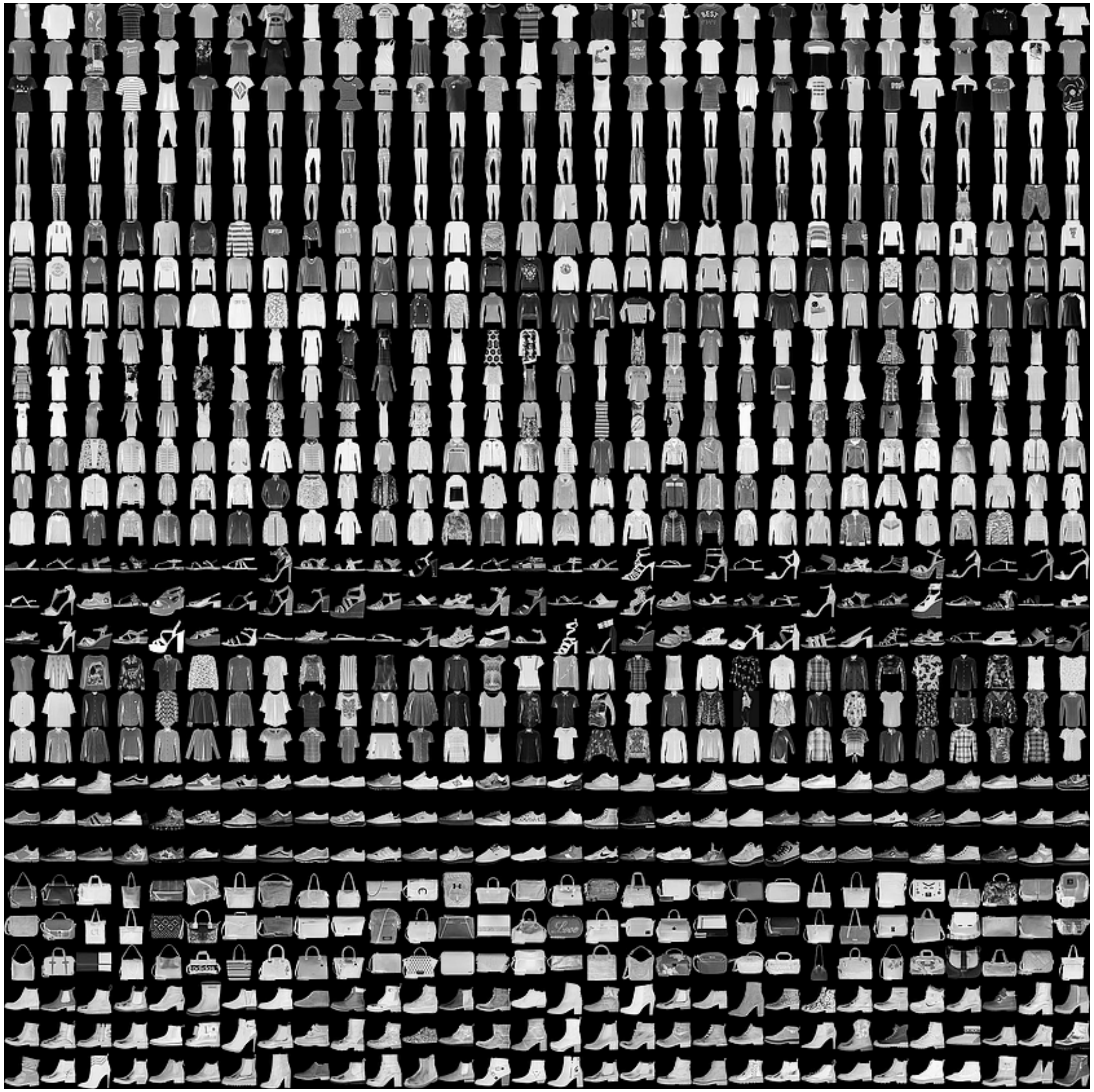
Approximate top tensor by MPS

Experiment: "fashion MNIST" dataset

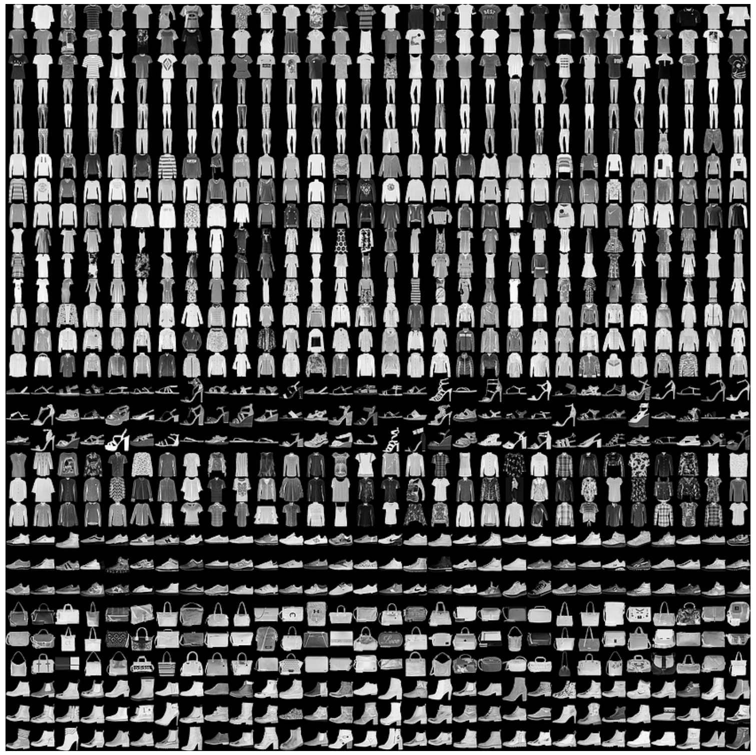
28x28 grayscale

60,000 training images

10,000 testing images

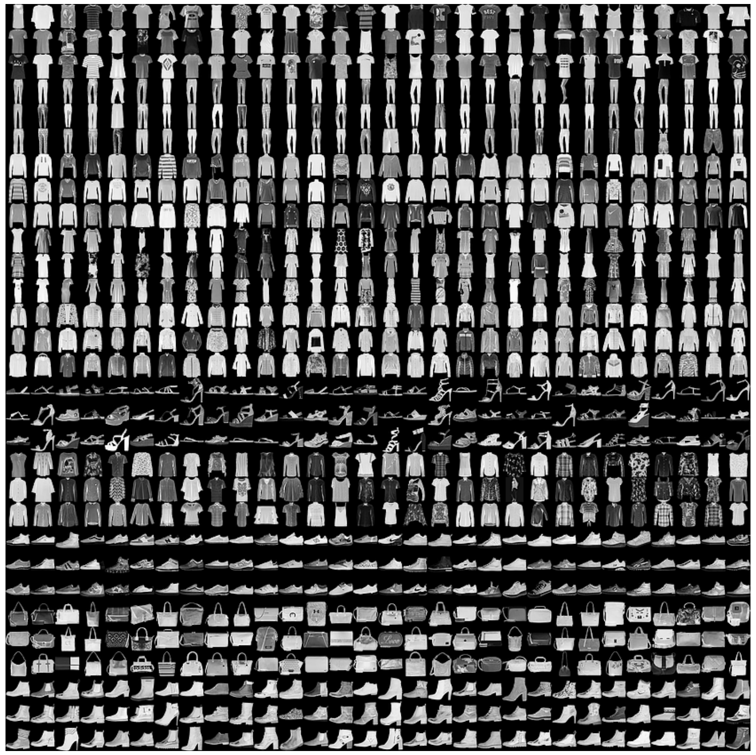


Experiment: "fashion MNIST" dataset



- Used 4 tree tensor layers
- Dimension of top "site" indices ranged from 11 to 30
- Top MPS bond dimension of 300 and 30 sweeps

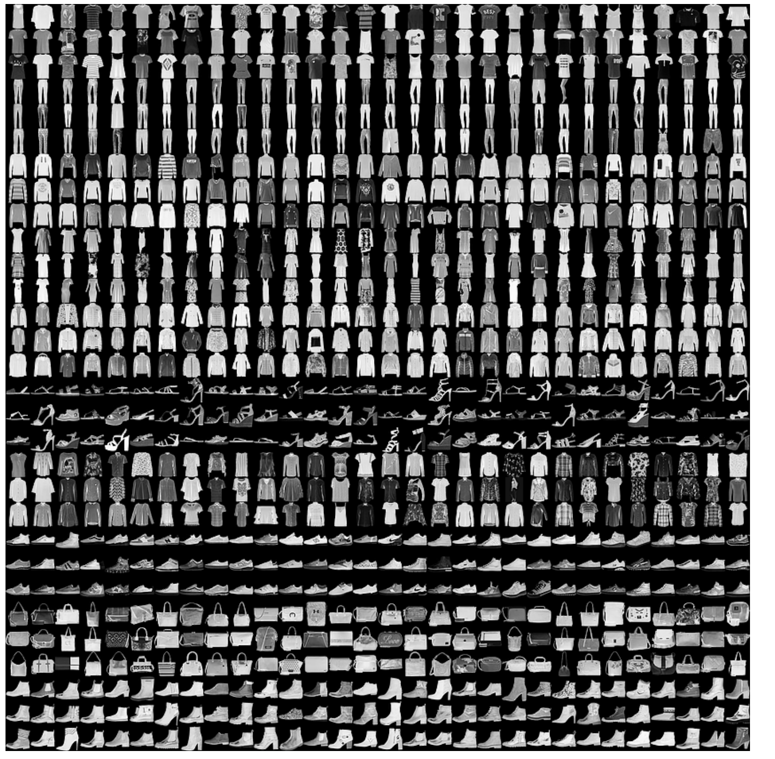
Experiment: "fashion MNIST" dataset



- Used 4 tree tensor layers
- Dimension of top "site" indices ranged from 11 to 30
- Top MPS bond dimension of 300 and 30 sweeps

Train acc: 95.38% **Test acc:** 88.97%

Experiment: "fashion MNIST" dataset



- Used 4 tree tensor layers
- Dimension of top "site" indices ranged from 11 to 30
- Top MPS bond dimension of 300 and 30 sweeps

Train acc: 95.38% **Test acc:** **88.97%**

Comparable to XGBoost (**89.8%**), AlexNet (**89.9%**), Keras Conv Net (**87.6%**)

Best (w/o preprocessing) is GoogLeNet at **93.7%**

Much Room for Improvement

- Use MERA instead of tree layers
- Optimize all layers, not just top, for specific task
- Iterate mixed approach: feed trained network into new covariance/density matrix
- Stochastic gradient based training

Recap & Future Directions

- Trained layered tensor network on real-world data in unsupervised fashion

- Specializing top layer gives very good results on challenging supervised image recognition tasks

- Linear tensor network approach gives enormous flexibility. Progress toward interpretability.