University of Zurich[UZH]

Titus Neupert

December 10–11, 2018

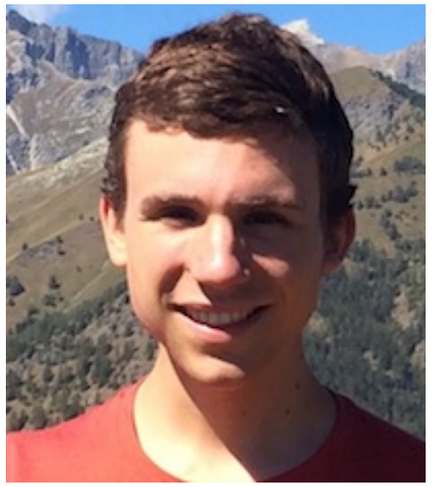
Mini-workshop

Machine-learning for
experimental condensed matter physics

Team for the workshop

Mark Fischer
(Uni Zurich)

Eliska Greplova
(ETH Zurich)



Frank Schindler
(Uni Zurich)

Kenny Choo
(Uni Zurich)

PROGRAM

Monday December 10 Y10-G-03/04 Seminarraum

14.30-15.30 **Titus Neupert: Lecture I**
15.30-16.00 break
16.00-17.15 **Titus Neupert: Lecture II**

Tuesday December 11 Y03-G-91 Seminarraum

Google Colab notebooks on
physik.uzh.ch/en/groups/neupert/Mini-workshop.html
[need a Google account]

13.30-15.00 **Kenny Choo and Mark H. Fischer: Code intro lesson**
15.00-15.30 break
15.30-16.30 **Dr. Eliska Greplova: Flake searching with AI**
16.45-18.15 **Frank Schindler: hands-on exercise session**

OVERVIEW

1: NN fundamentals

- network structure (variational function)
- activation functions
- layer types: dense, convolutional, drop-out, pooling
- cost function (loss): quadratic, cross-entropy
- optimizer: gradient descent (with momentum)

2: unsupervised techniques

- autoencoders
- dreaming
- vulnerability of NN
- principle component analysis

3: NN in condensed matter physics

- phase classification
- applications to material discovery
- variational quantum states, quantum state tomography
- device design with machine learning

AI is everywhere

… but not equally useful everywhere

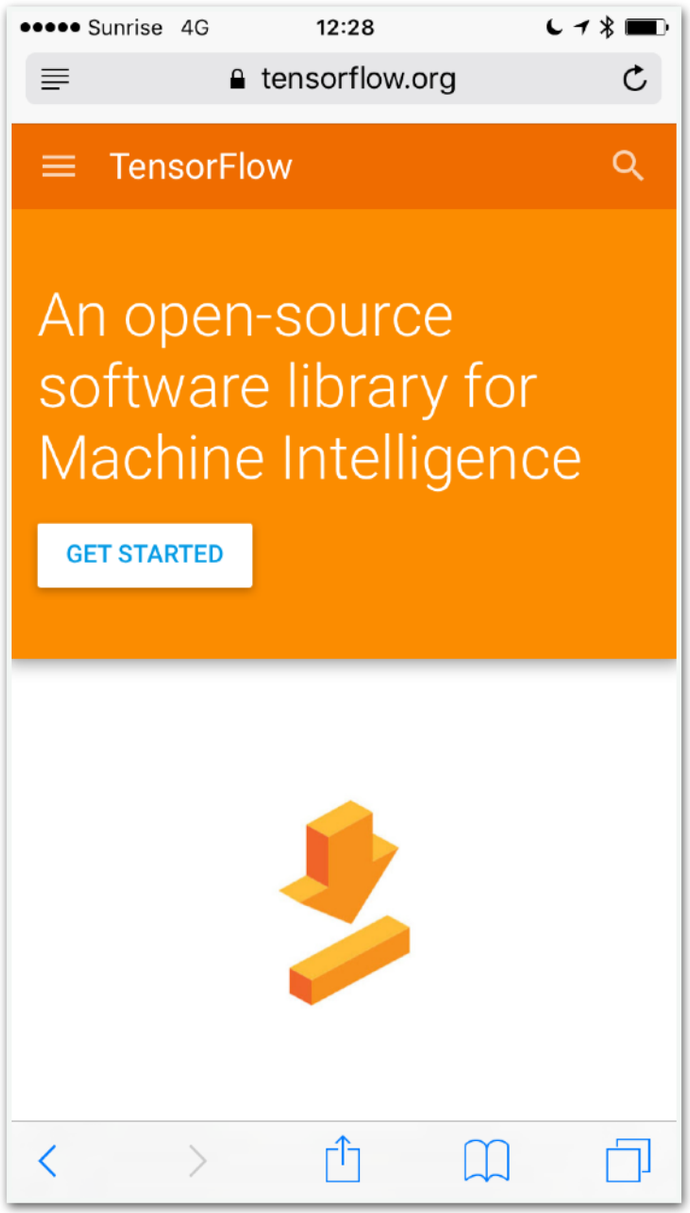
Picture/pattern/face recognition
Voice recognition
Translations
Recommendation systems
Spam filters
…

A lot of jargon, but simple math behind
(There is more to it than neural networks)

BIG data technique

Codes

Low entry barrier through packages like **TensorFlow, Keras, Mathematica**, etc. that work out of the box



Companies using TensorFlow

Google

IBM Power Systems

airbnb

AIRBUS DEFENCE & SPACE

intel

JD.COM 京东

ARM

CEVA

QUALCOMM

quantiphi

CI&T

DeepMind

SAP

Dropbox

ebay

UBER

Structure of neural networks

NN = class of variational function (many parameters) $\vec{x} \mapsto \vec{y}$

some can be shown to approximate smooth functions arbitrarily well

layered structure successive application of different functions (**layers**)

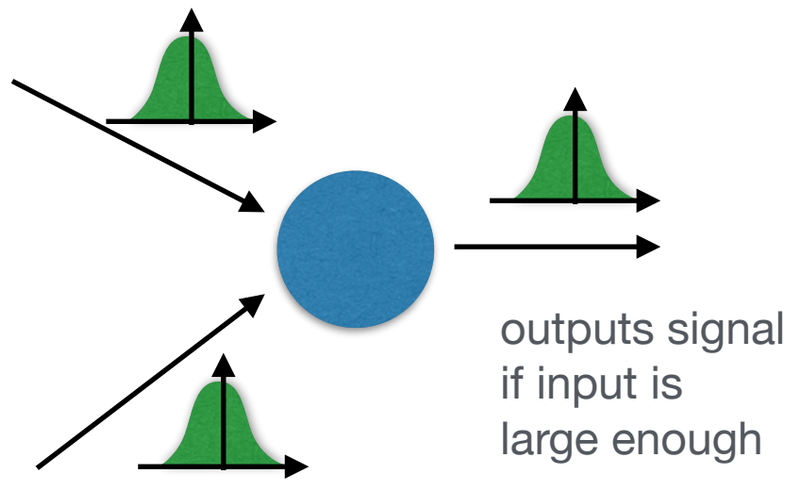
NEURONS inspired from biology

$$f(\vec{x}) = g\left(\sum_i w_i x_i + b\right)$$

activation function **weights** **bias**

fixed by network structure

variational freedom

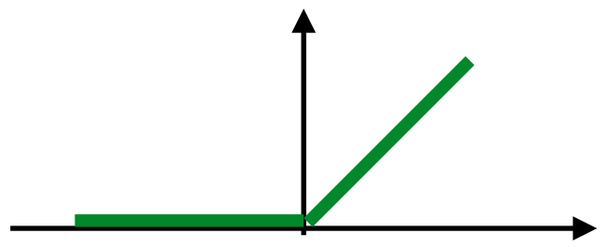


Structure of neural networks

Activation functions

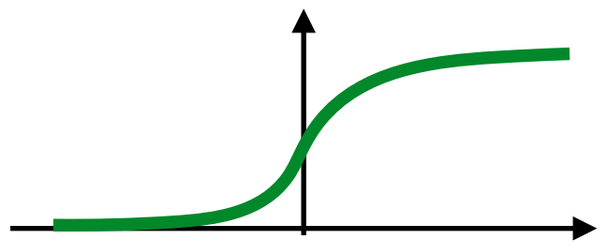
ReLu

$$g(s) = s\ \Theta(s)$$



Sigmoid

$$g(s) = \frac{1}{1 + e^{-s}}$$



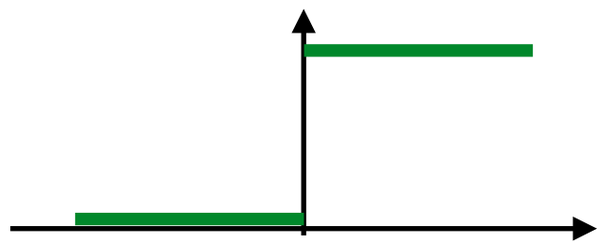
Softmax

$$g_i(\vec{s}) = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

Output sums to 1, good for probability distributions

Step function (perceptron)

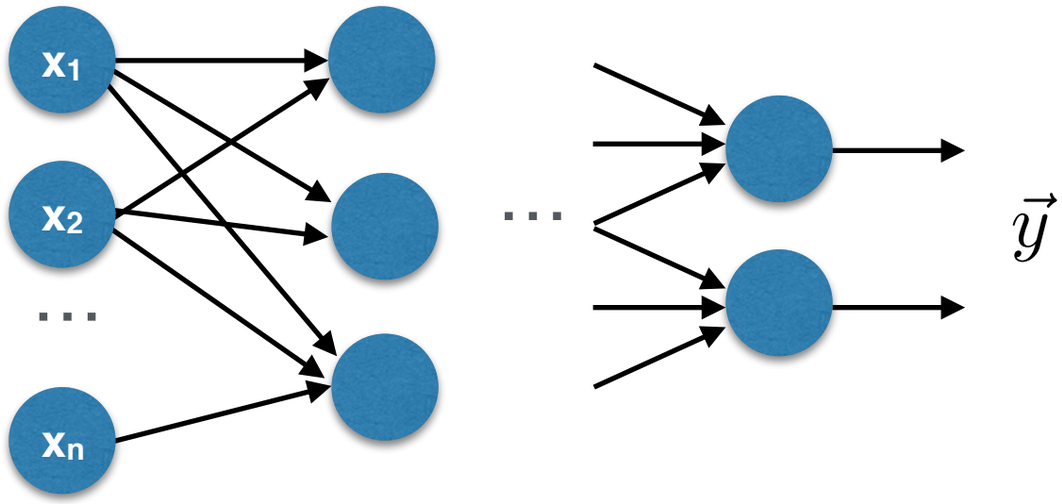
$$g(s) = \Theta(s)$$



…

Structure of neural networks

Choose number N of neurons in each layer



Layer terminology

Feed-forward NN if no “loops” in flow/variable dependence

Deep NN if “many” layers

Input layer does nothing

Output layer last layer, has y as output

Hidden layer all layers between in- and output (one in above example if there was no …)

Fully connected layer takes input from ALL previous layer outputs to each of its neurons

… variety of layer types to come

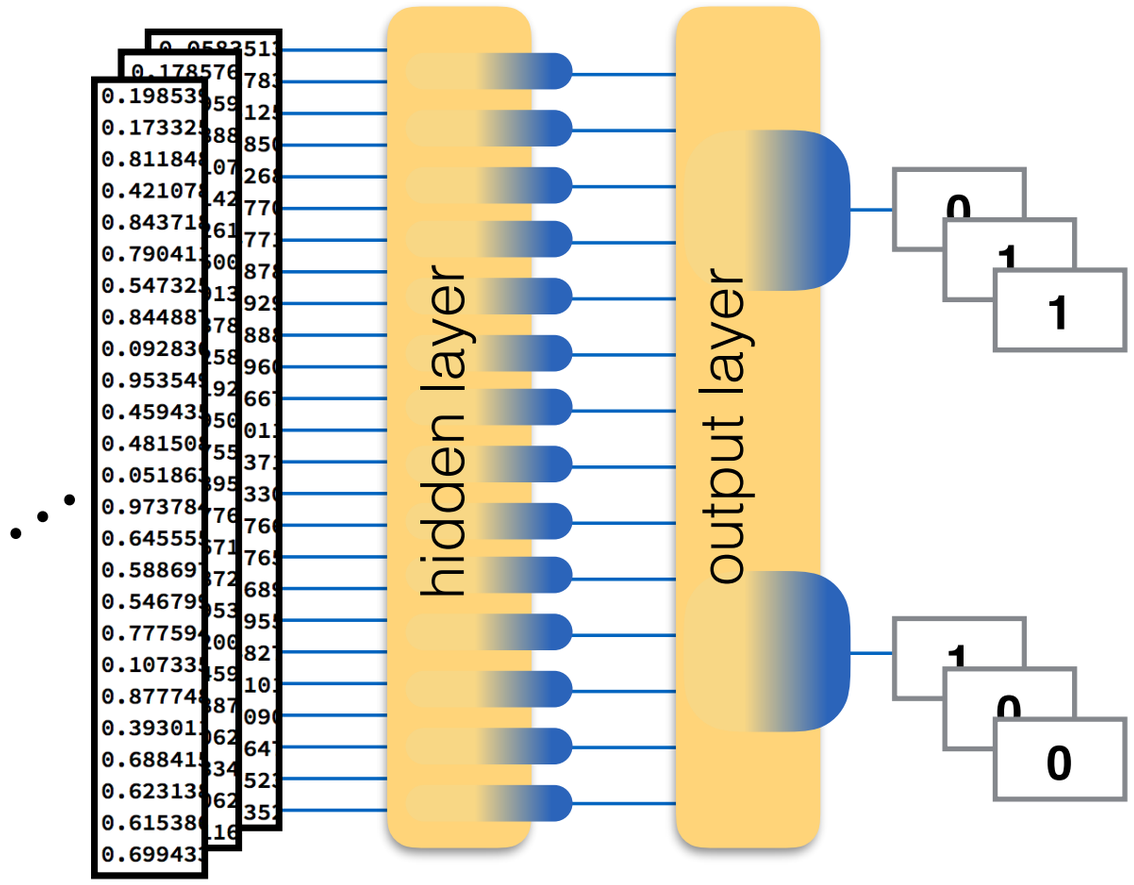
Example: fully connected NN with one hidden layer

$$y_i = f_i(\vec{x}) = g^{(2)}\left[\sum_{j=1}^{N_2} W^{(2)}_{i,j}\, g^{(1)}\left(\sum_{k=1}^{N_1} W^{(1)}_{j,k}\, x_k + b_j\right) + b^{(2)}_i\right]$$

$\vec{x}$	N_1-vector	dimension of input layer **N_1**
$b^{(1)}$	N_2-vector	“number of hidden units” **N_2** (free parameter of network structure)
$W^{(1)}$	$N_1 \times N_2$-matrix	
$b^{(2)}$	N_3-vector	
$W^{(2)}$	$N_2 \times N_3$-matrix	dimension of output vector **N_3**

Supervised learning

$$\{(\vec{x}_\alpha, \vec{y}_\alpha)\}_{\alpha=1}^{M}$$



Network learns function implicitly characterized by the data

No exact science

pairs $(\vec{x}, \vec{y})$ of input and function values:
labelled data (y is label)

Goal: fit network parameters (weights and biases) such that

1) network output for data is as close to the label as possible

2) network generalizes to previously not seen similar data

Supervised learning $\{(\vec{x}_\alpha, \vec{y}_\alpha)\}_{\alpha=1}^{M}$

COST FUNCTION (LOSS): objective for learning is minimization of cost function (a minimum should be reached when labels agree with network output for data)

$$C(W^{(1)}, W^{(2)}, \dots, b^{(1)}, b^{(2)}, \dots)$$

choice of cost function important part of learning problem definition

Needs to be smooth in W, b (not integer-valued…)

Ex 1: Quadratic cost function

$$C(W,b) = \frac{1}{2M} \sum_\alpha ||\vec{f}(\vec{x}_\alpha) - \vec{y}_\alpha||^2$$

W, b dependence

minimum at 0

slow learning for vastly wrong output (initial phase)

Ex 2: Kullback-Leibler divergence/categorical cross-entropy

$$C(W,b) = -\frac{1}{M} \sum_\alpha \sum_i [f_i(\vec{x}_\alpha) \log y_{\alpha,i} + (1 - f_i(\vec{x}_\alpha)) \log (1 - y_{\alpha,i})]$$

often better behaved than Ex.1

Supervised learning

OPTIMIZATION: finding W, b by minimizing C over data

gradient descent

$$\Delta C = \sum_r \frac{\partial C}{\partial \nu_r} \Delta \nu_r \qquad \Delta \nu_r = -\eta \nabla_r C$$

ν_r … collection of network parameters W, b

η … **leaning rate**, small positive number

example of **hyper parameter** that controls learning process

Global choice of learning rate difficult; improve by including **momentum**

$$\Delta \nu_r^{(t)} = -\eta \nabla_r C + \gamma \Delta \nu_r^{(t-1)}$$

keeps going in previous direction

γ … **coefficient of momentum/inertia**

Supervised learning

Problem: computation of gradients of C extremely costly for large data sets and many variational parameters

Stochastic gradient descent: use only a small, randomly chosen subset of the data **(“batch”/ “mini-batch”)** to evaluate the gradients approximately

$$\nabla_r C = \frac{1}{M} \sum_{\alpha} \cdots \quad \longrightarrow \quad \nabla_r C \approx \frac{1}{m} \sum_{\alpha \in B} \cdots$$

m … size of batch B

Many variants with various tweaks, for instance **ADAM**

size m of batch is another important hyper-parameter

Supervised learning

Not all data is used for training (= **training set**); small part is set apart as a **test set** to estimate whether network **generalizes** well to previously unknown data.

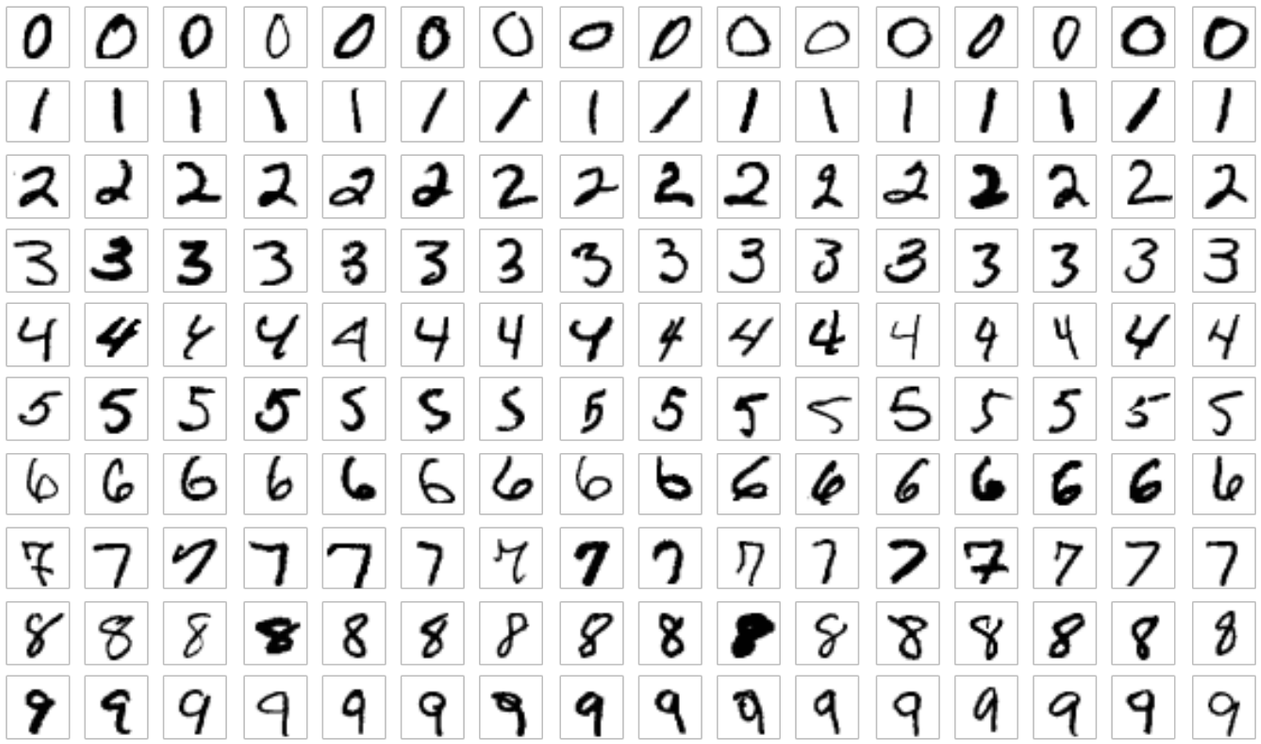
Repetition of the above is typically referred to as **cross-validation**

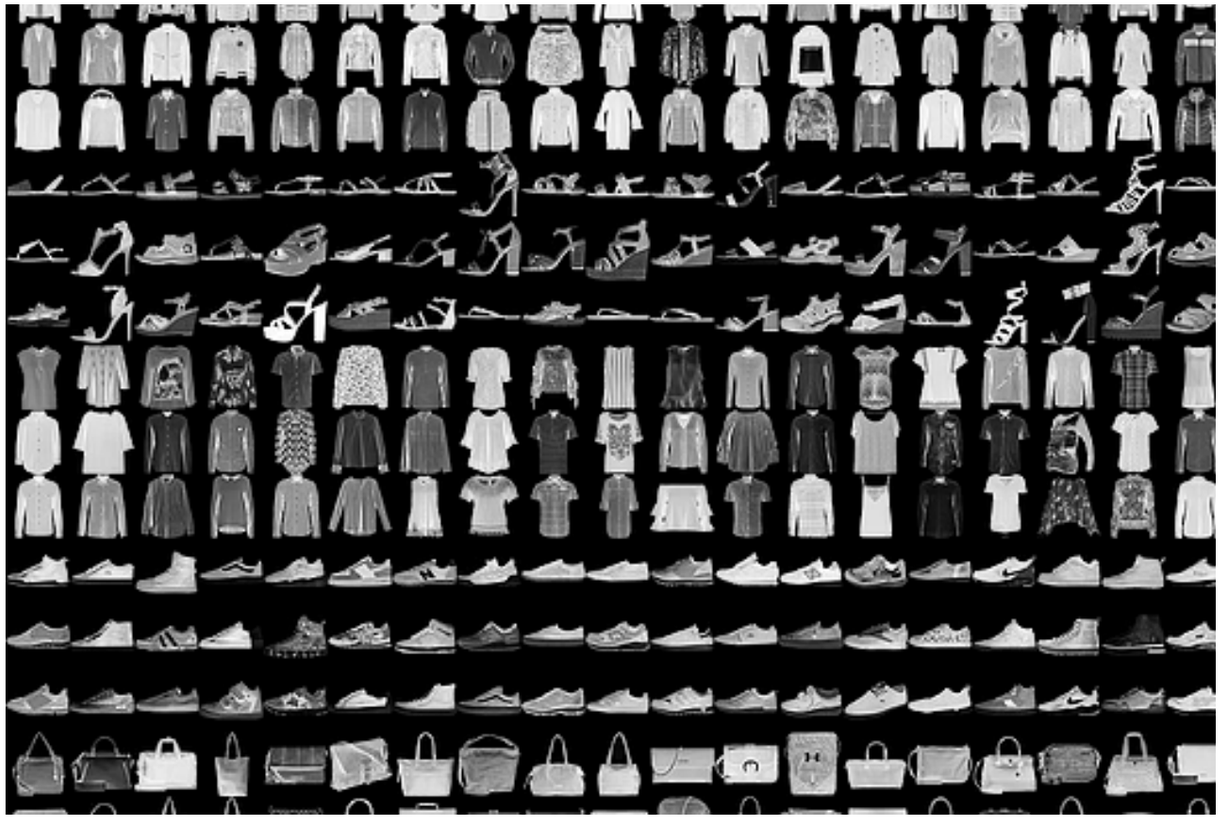
Can set apart another **validation set** to verify that optimization of hyperparameters (via repeated evaluation of test set) has not led to overfitting as well

If network does not perform well on test set, it has learned undesired specifics of the data: **overfitting**

Naive computation of C gradients is costly. Due to layered structure of the network, one can use the **chain rule** to speed it up a lot: **backpropagation algorithm**

Benchmarks: MNIST and fashion MNIST





60 000 training, 10 000 test images, 28x28 pixel

best algorithm: 0.23% error rate

best algorithms: ~10% error rate

Regularization methods

Fight overfitting

0) More training data

1) Weight decay: add term to the cost function

$$C \to C + \frac{\lambda}{2M} \sum_{i,j,\ell} \left| W_{i,j}^{(\ell)} \right|^2$$

λ … regularization parameter

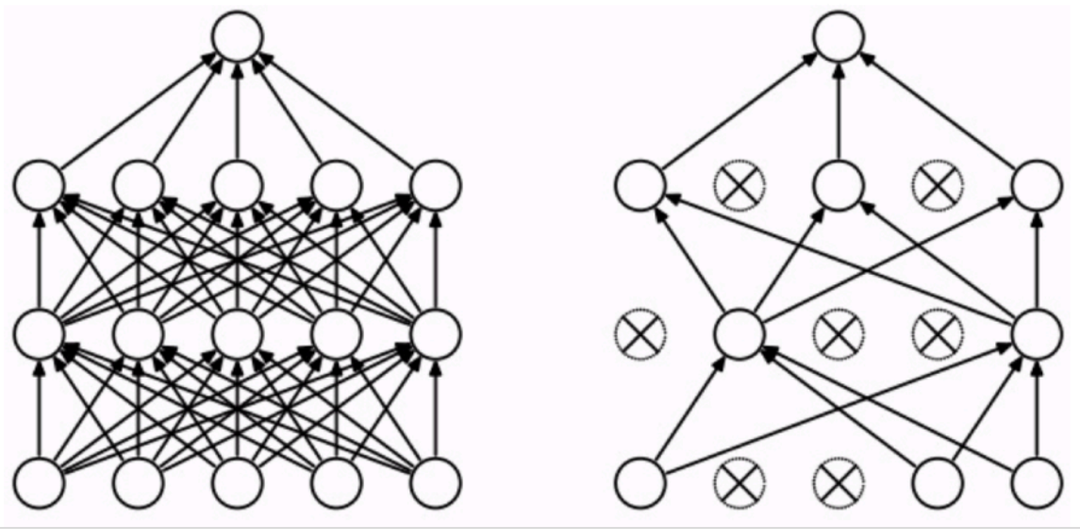
encourages using as few weights as possible

2) Dropout layers:

sets inputs to the subsequent layer
to 0 randomly during each evaluation

used during training

trades training performance for
generalization



fully connected layers
without and with dropout

Convolutional networks

applies a number of filters to input data detect local features

1D example

$$y_r^{(l)} = g\left(\sum_i^n W_i^{(l)} x_{rs+i}\right) \qquad l = 1, \cdots, d \qquad r \approx 1, \cdots, N/s$$

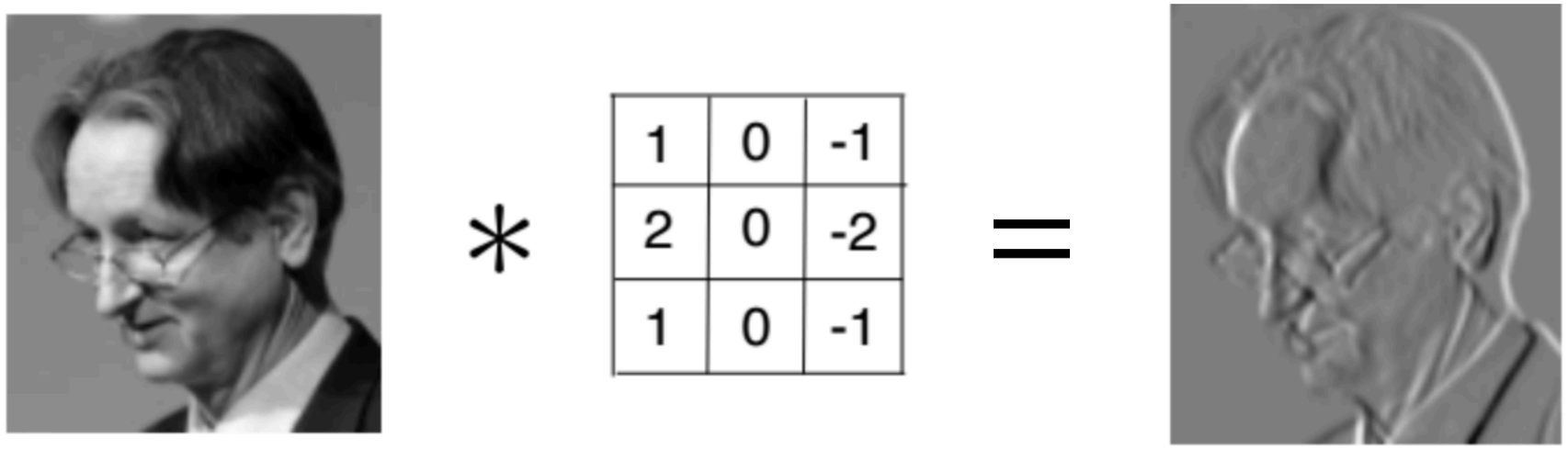
$y_r^{(l)}$: **feature maps** — $W_i^{(l)}$: **convolutional filters**

n … filter size, smaller than N (length of x)

s … **stride**: determines by how much filtered regions overlap (s = 1,…,n)

d … **depth**: number of filters/feature maps processed in parallel

2D example



input data may be padded before

Convolutional networks

Pooling layer applies an operation like **max** or taking the **average** to subsequent subsets of the input vector

typically used with stride s>1 to reduce the size of the data

e.g. max pooling in 1D:

$$y_r = \max\{x_{rs+1}, \cdots, x_{rs+n}\}$$

Typical structure: convolution-pooling-convolution-pooling …

OVERVIEW

1: NN fundamentals

- network structure (variational function)
- activation functions
- layer types: dense, convolutional, drop-out, pooling
- cost function (loss): quadratic, cross-entropy
- optimizer: gradient descent (with momentum)

2: unsupervised techniques

- autoencoders
- dreaming
- vulnerability of NN
- principle component analysis

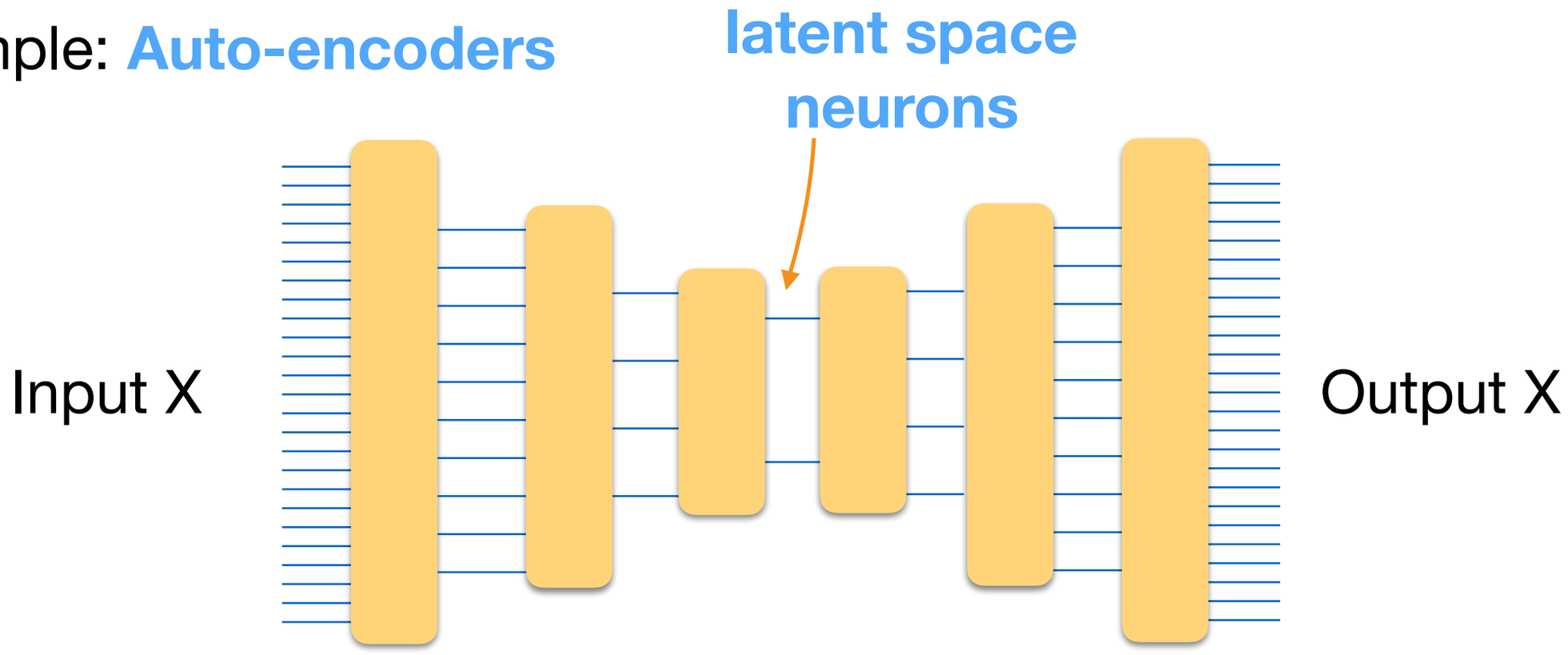
3: NN in condensed matter physics

- phase classification
- applications to material discovery
- variational quantum states, quantum state tomography
- device design with machine learning

Unsupervised learning

Find structure in data without any labels

Example: **Auto-encoders**



Train on input = output: represents identity on data set

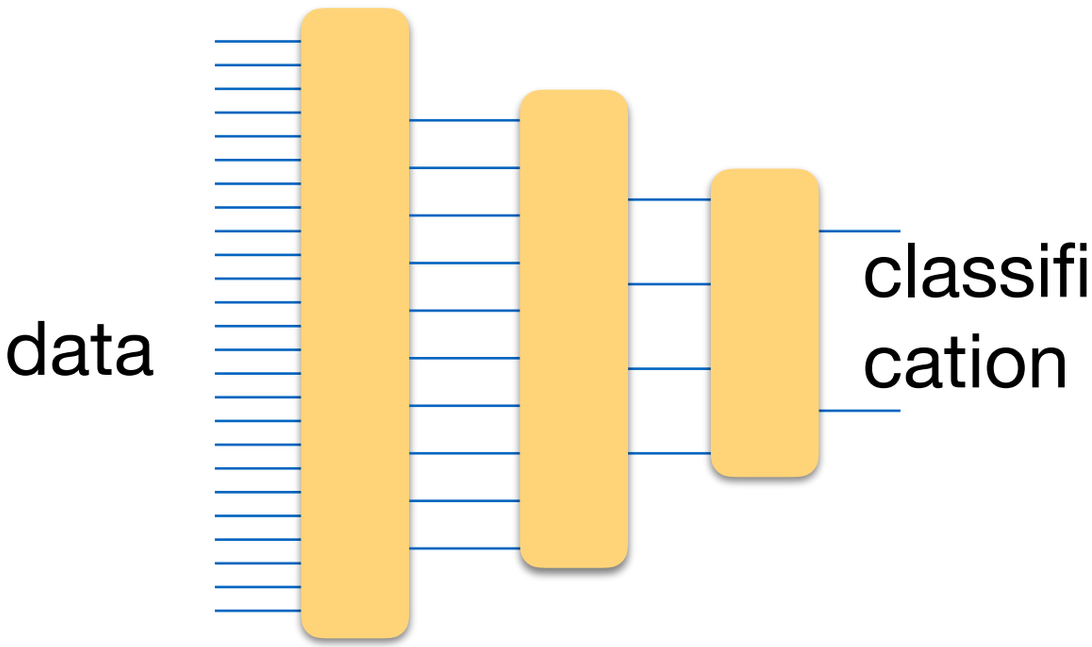
Information has to be compressed

How many latent space neurons needed for this?

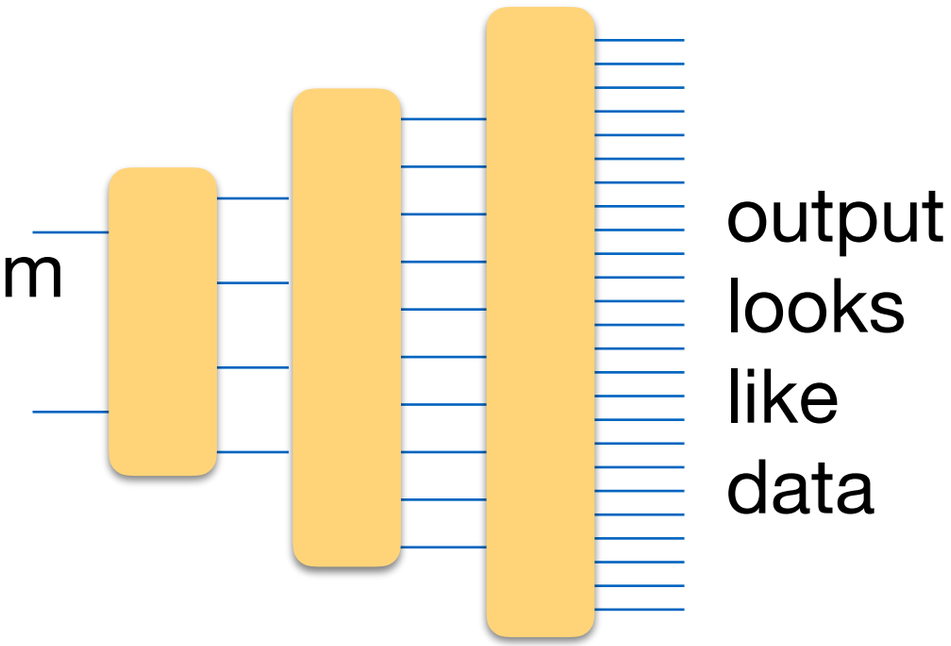
Unsupervised learning

Two uses of the trained network:

Encoder (Classifier)



Decoder (Generative network)



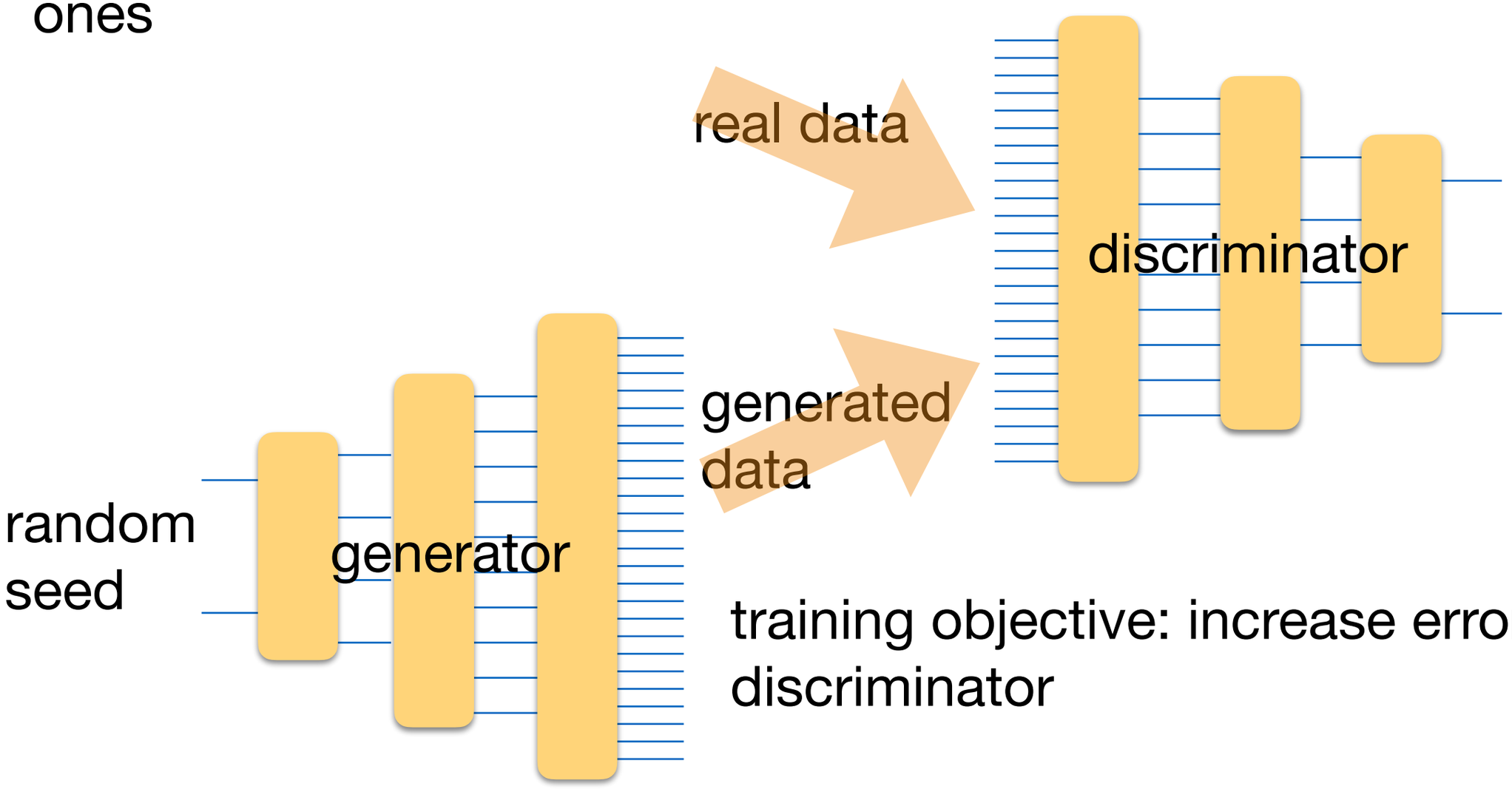
Variational autoencoder [2013]

ensures Gaussian distribution of latent space variables over data by adding KL divergence between unit Gaussian and latent variables to cost function

Unsupervised learning

Generative adversarial network (GAN) [2013/14]

Goal: learn how to generate datasets indistinguishable from existing ones



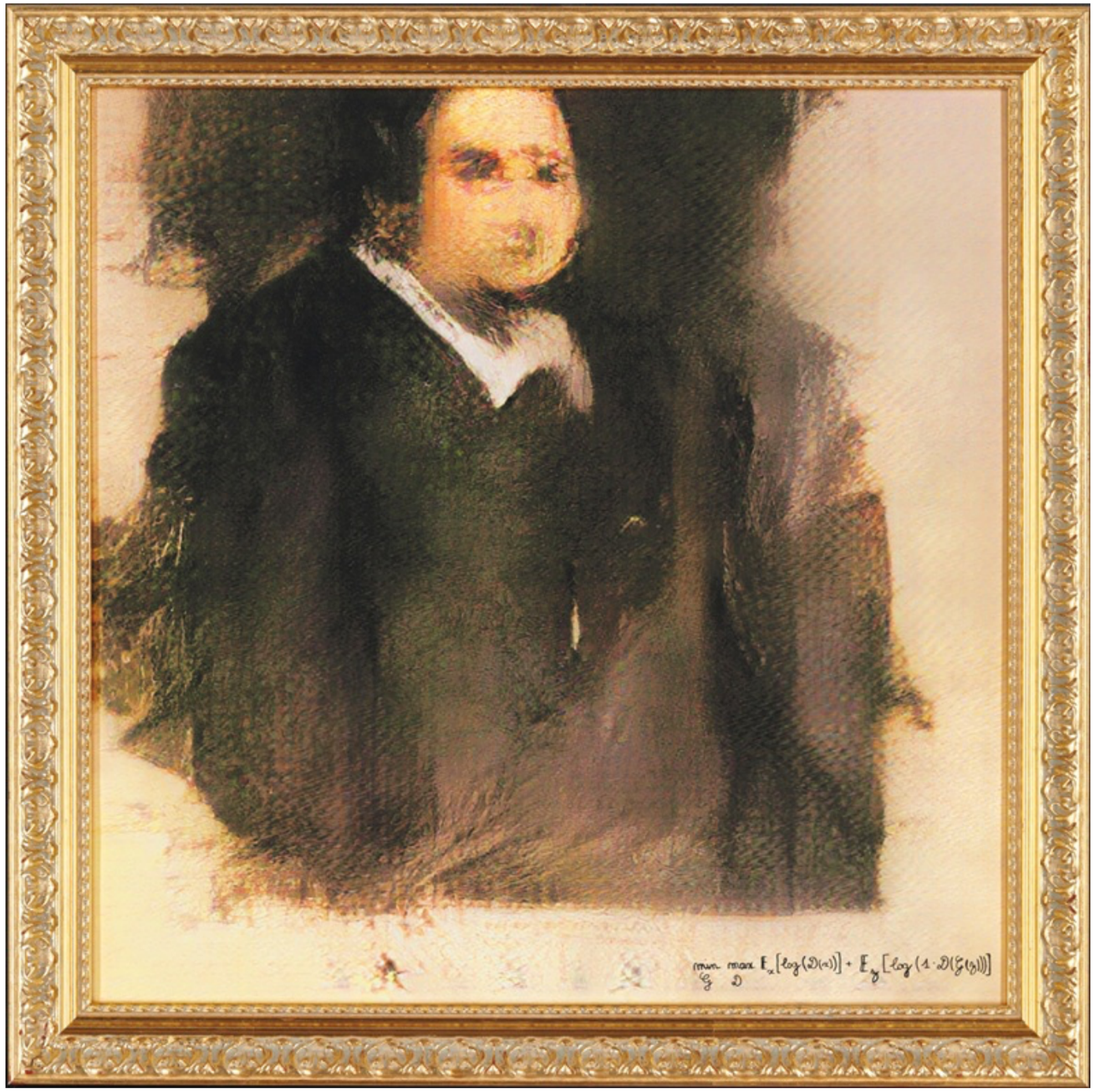
training objective: increase error rate of discriminator

training improves the ability of generator to fake data and of discriminator to distinguish fake from real data

Unsupervised learning

Applications of GANs:

Googles new **Alpha GO**



Edmond de Belamy
GAN created canvas painting
sold at Christie's for 432,500 $

Routinely implemented in NN codes

Unsupervised learning

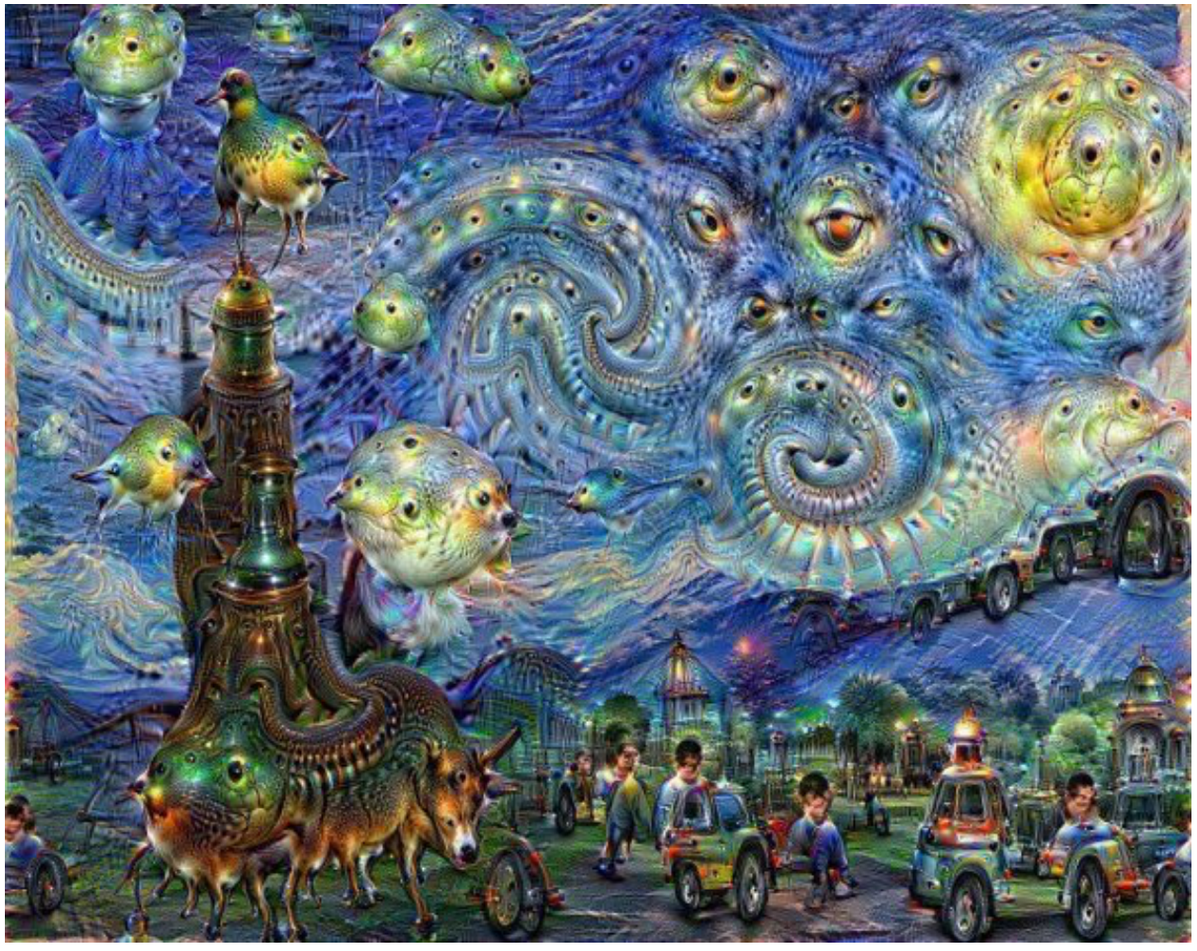
Dreaming

(hallucinogenic dreaming)

Use fully (e.g., supervised) trained network; fix all its parameters

- fix output to desired one
- change input (from arbitrary seed) until cost function is minimized

Shows what network has learned



Vulnerability of NN (1)

1

network trained to distinguish pictures from Reese Witherspoon and Russell Crow

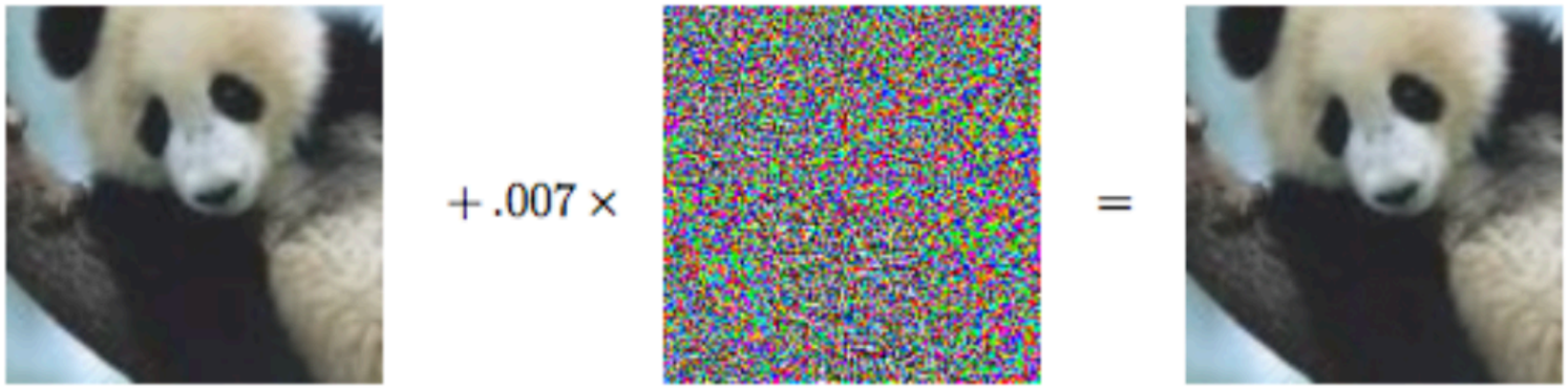
2

dream glasses of Reese Witherspoon until network thinks for certain it is **Russell Crow**

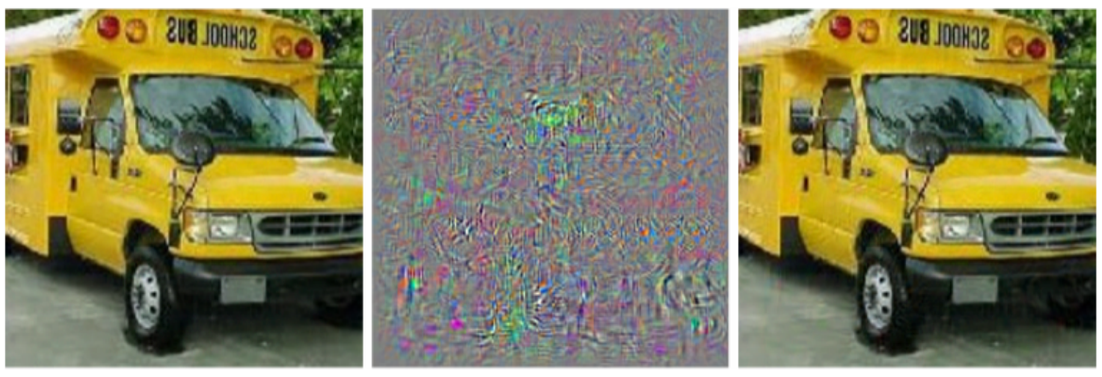
[Sharif et al. http://dx.doi.org/10.1145/2976749.2978392]

Vulnerability of NN (2)

Misclassification after adding small noise to the data



[Goodfellow et al. https://arxiv.org/abs/1412.6572]



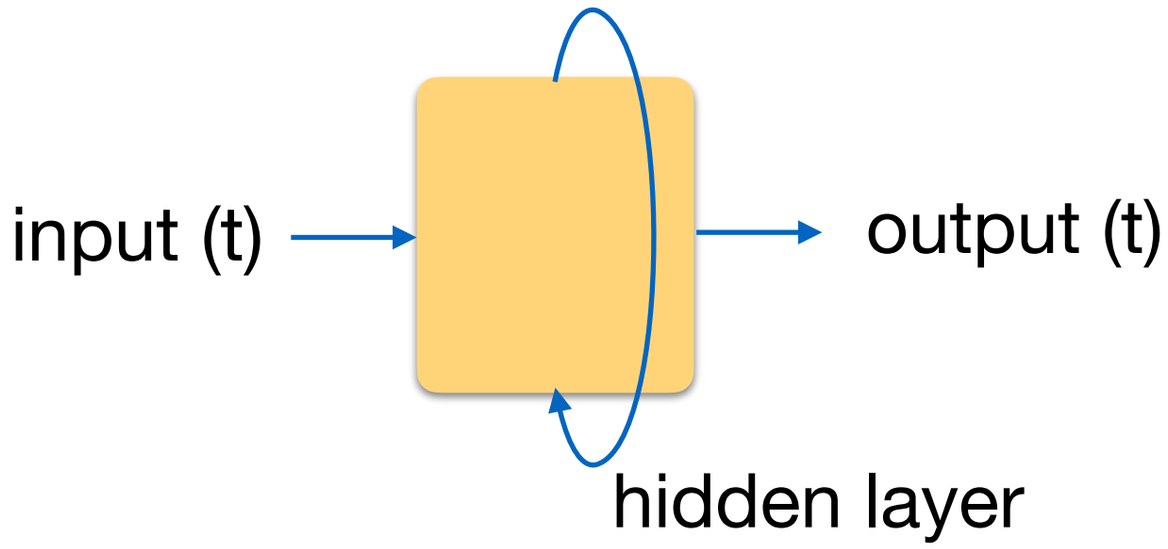
[Szegedy et al. https://arxiv.org/pdf/1312.6199v1.pdf]

Importance of regularization methods (dropout, add noise to data …)

Time series: recurrent NNs

NN so far only work with fixed size input/output

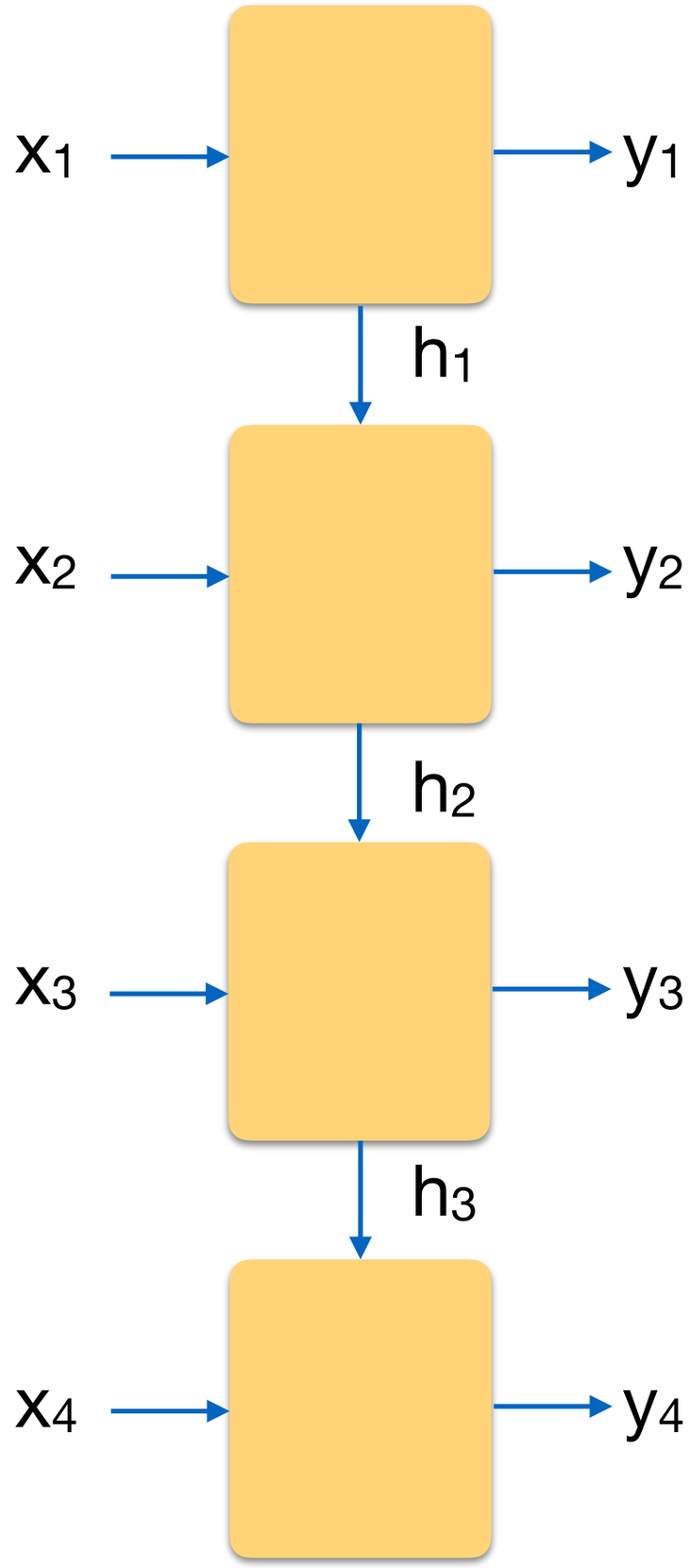
recurrent NNs are useful for time series



simple example without output:

$$\boldsymbol{h}_t = g\left(W_{hh}\boldsymbol{h}_{t-1} + W_{xh}\boldsymbol{x}_t\right)$$

applications: natural language processing, image recognition, video processing, machine translation …



Simpler machine learning: PCA

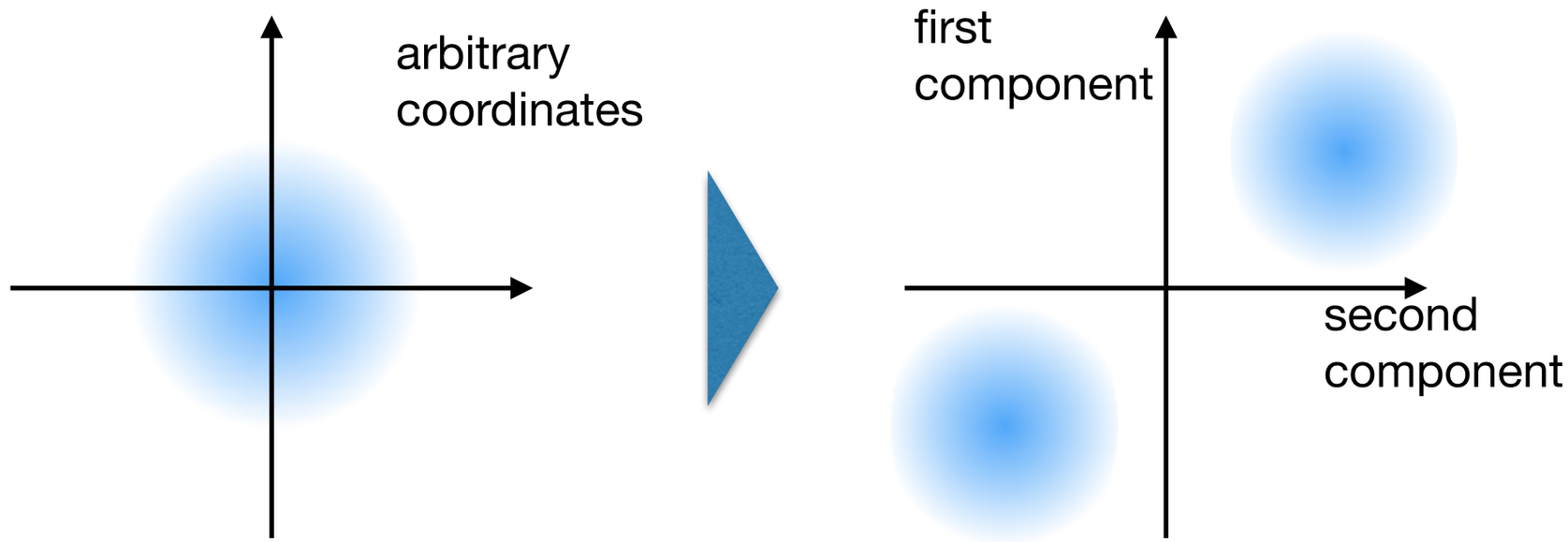
Principle component analysis (PCA) is a simple form of finding structure in data

data matrix
(each row a dataset) X ▶ eigenvalue
decomposition of $X^{\mathsf{T}}X$

linear transformation that best separates data points (finds the best separating hyperplane through data space)

project all data onto largest eigenvector, then onto subsequent ones…

"**score**" $X.\boldsymbol{w}^{(1)}$



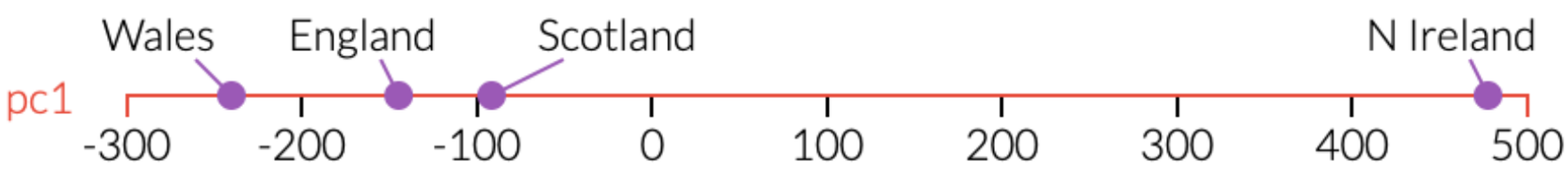
Simpler machine learning: PCA

Example: eating habits in UK
[from: http://setosa.io/ev/principal-component-analysis/]

	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	874
Fresh Veg	253	143	171	265
Other meat	685	586	750	803
Other Veg	488	355	418	570
Processed potatoes	198	187	220	203
Processed Veg	360	334	337	365
Soft drinks	1374	1506	1572	1256
Sugars	156	139	147	175

difficult to read off anything

easy to see N Ireland is outlier



nonlinear generalization: **support vector machines**

Intermediate summary

- network variants are countless and evolving
- supervised learning most well defined problem
- unsupervised techniques more open challenge/problem

- performance is main objective
- interpretability of networks is a frontier (physicist's view)
  - dreaming
  - study weights directly
  - study convolutional filters individual actions

- BIG data is the best way to improve network performance
- libraries optimized for GPUs

OVERVIEW

1: NN fundamentals

- network structure (variational function)
- activation functions
- layer types: dense, convolutional, drop-out, pooling
- cost function (loss): quadratic, cross-entropy
- optimizer: gradient descent (with momentum)

2: unsupervised techniques

- autoencoders
- dreaming
- vulnerability of NN
- principle component analysis

3: NN in condensed matter physics

- phase classification
- applications to material discovery
- variational quantum states, quantum state tomography
- device design with machine learning

- finding phases and phase boundaries

- experimental data analysis of various kinds

- automated materials discovery from big databases

- quantum state representation/compression

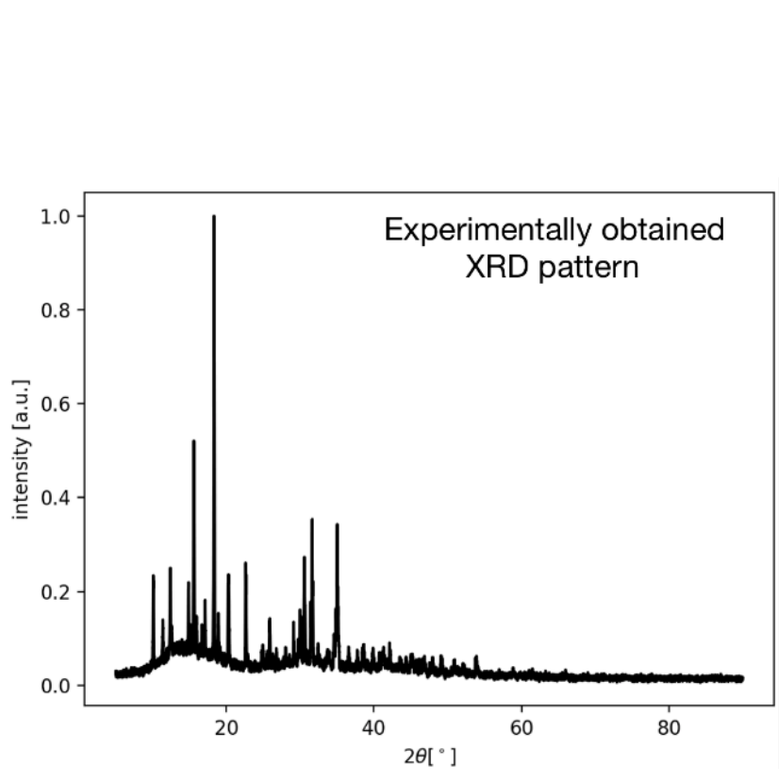
- quantum state tomography (used, e.g., with quantum simulation devices)

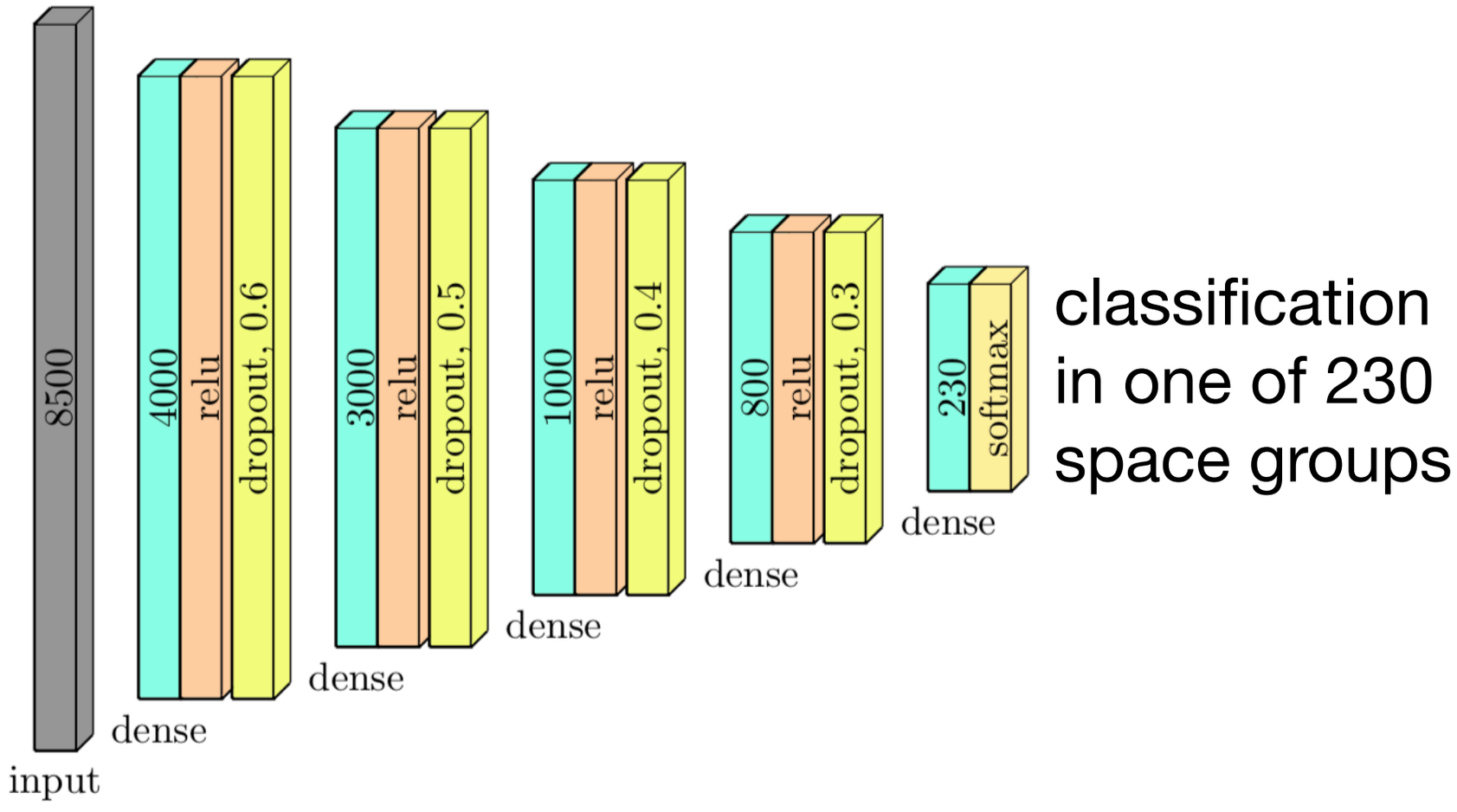
- …

Phase classification (fully supervised)

[Vecsei et al., to appear]

Example: find **crystal structure** (space group/crystal system classification) from X-ray diffraction (XRD) patterns





classification in one of 230 space groups

train with theoretically computed data (~100 000 datasets)

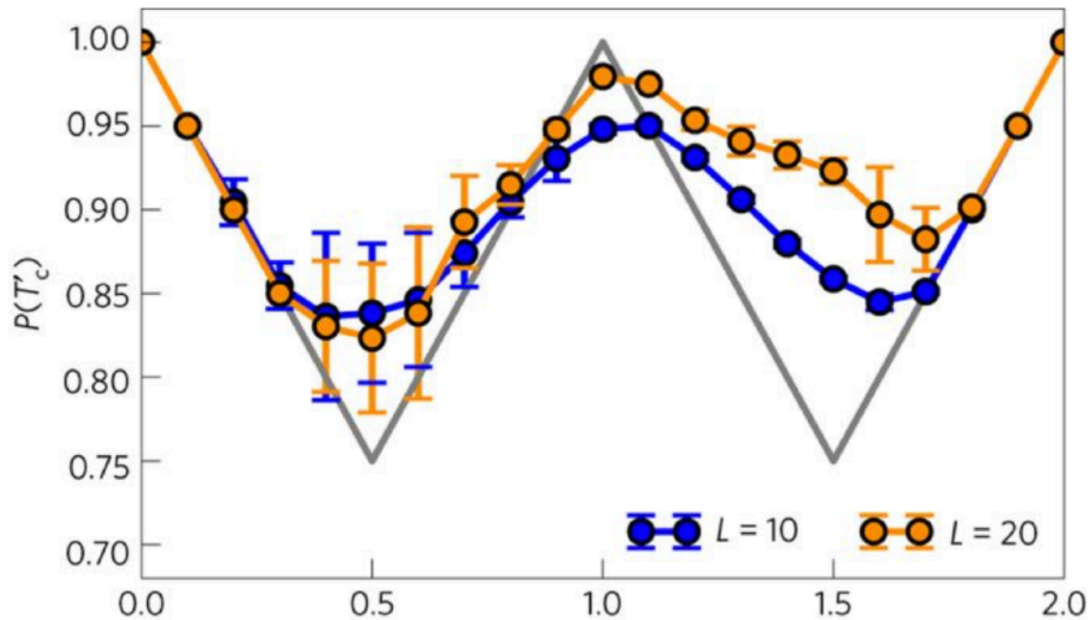
	Crystal systems		Space groups	
	Test set	RRUFF	Test set	RRUFF
Convolutional	85%	56%	76%	42%
Dense	73%	70%	57%	54%

Phase classification in unknown phase diagram

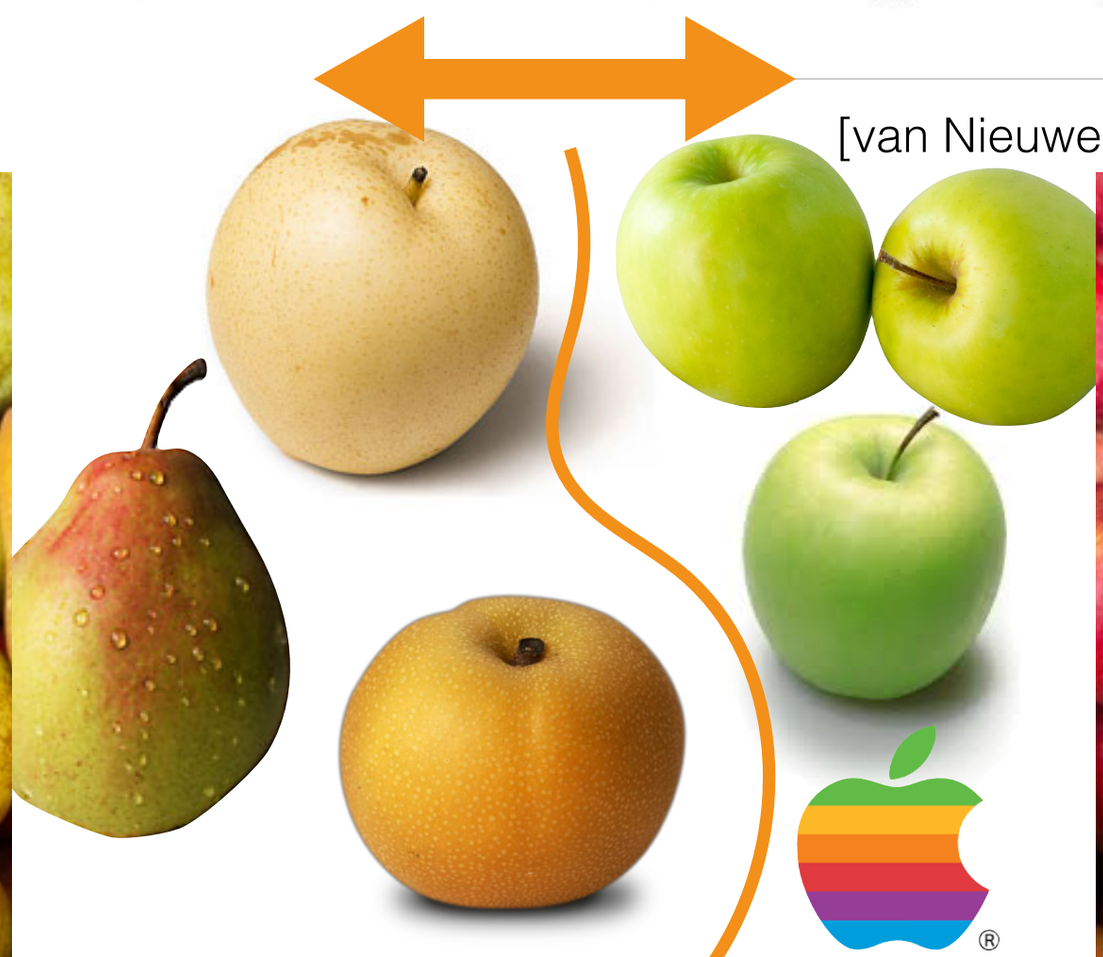
Learning by confusion

- data ordered in parameter space (e.g. by temperature)
- select putative phase boundary, train supervised NN network
- change boundary and repeat
- networks best training performance corresponds to true phase boundary

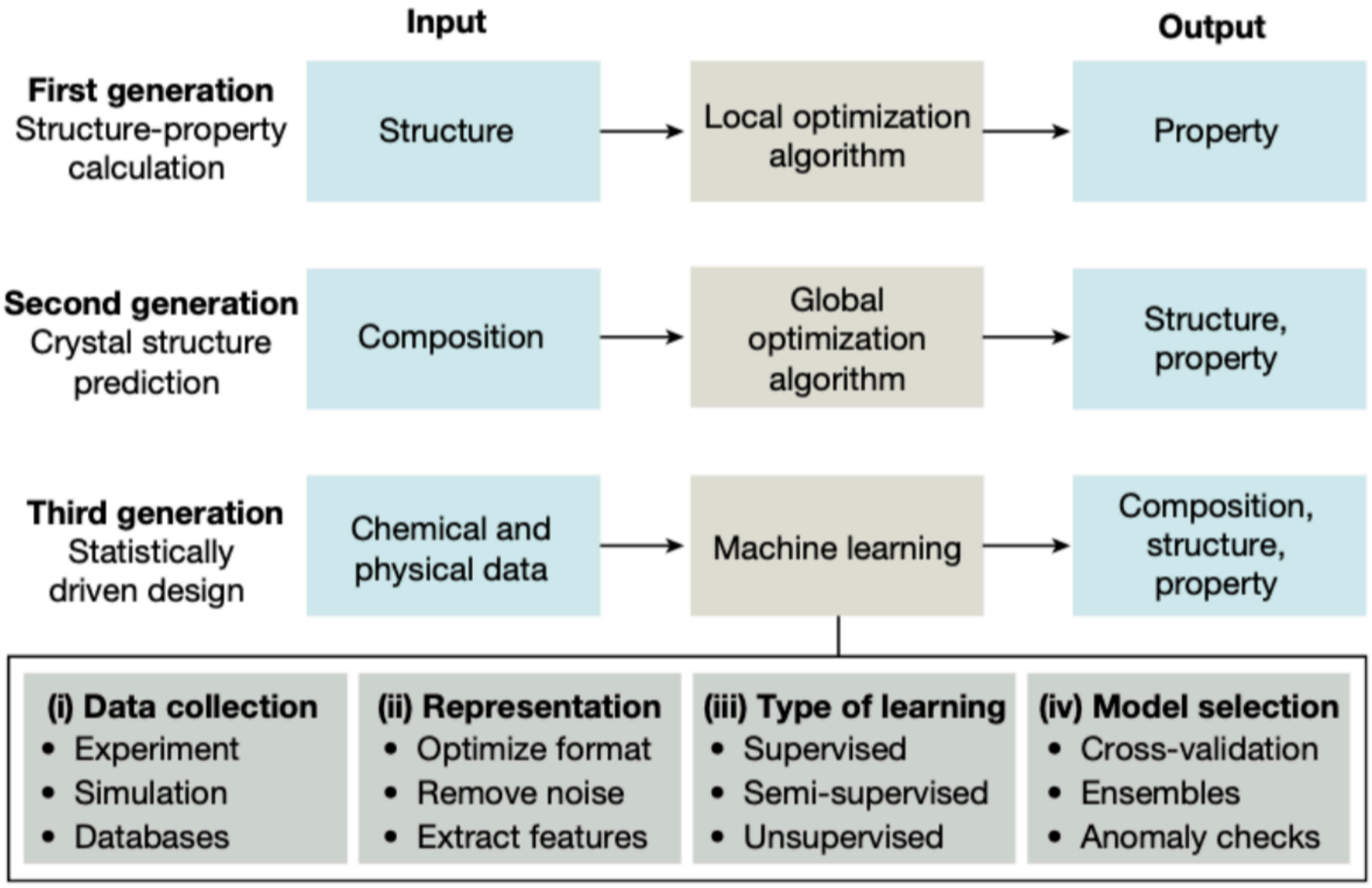
example: Ising model



[van Nieuwenburg et al. Nature Physics, 2017]



Prediction of crystallographic and physical properties



[Butler et al. Nature (Review article), 2018]

Prediction of crystallographic and physical properties

Schütt, K. T. et al. How to represent crystal structures for machine learning: towards fast prediction of electronic properties. *Phys. Rev. B* **89**, 205118 (2014).
A radial-distribution-function description of periodic solids is adapted for machine-learning models and applied to predict the electronic density of states for a range of materials.

Wicker, J. G. P. & Cooper, R. I. Will it crystallise? Predicting crystallinity of molecular materials. *CrystEngComm* **17**, 1927–1934 (2015).
This paper presents a crystal engineering application of machine learning to assess the probability of a given molecule forming a high-quality crystal.

Brockherde, F. et al. Bypassing the Kohn-Sham equations with machine learning. *Nat. Commun.* **8**, 872 (2017).
This study transcends the standard approach to DFT by providing a direct mapping from density to energy, paving the way for higher-accuracy approaches.

[Butler et al. Nature (Review article), 2018]

Machine learning modeling of T_c of superconductors

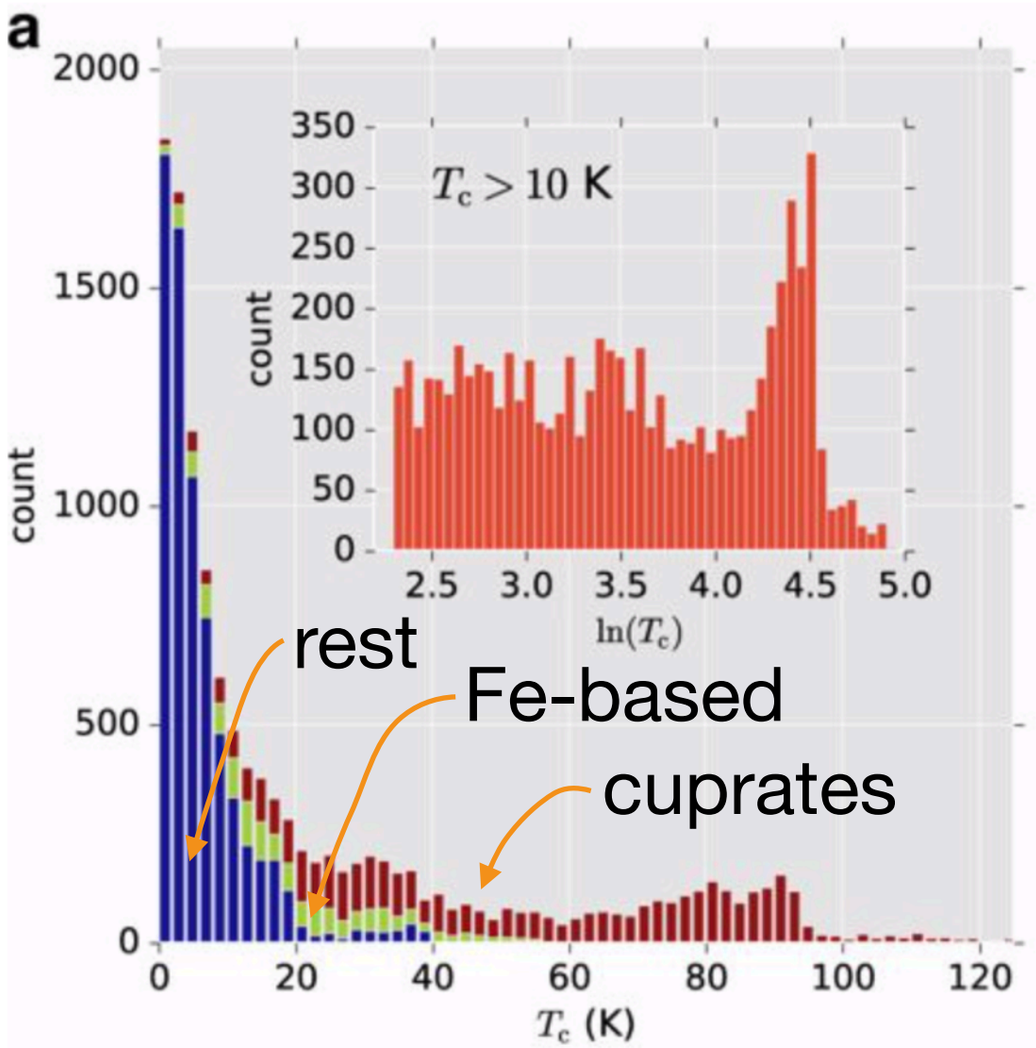
Data: 12000 superconductors
chemical composition and structure

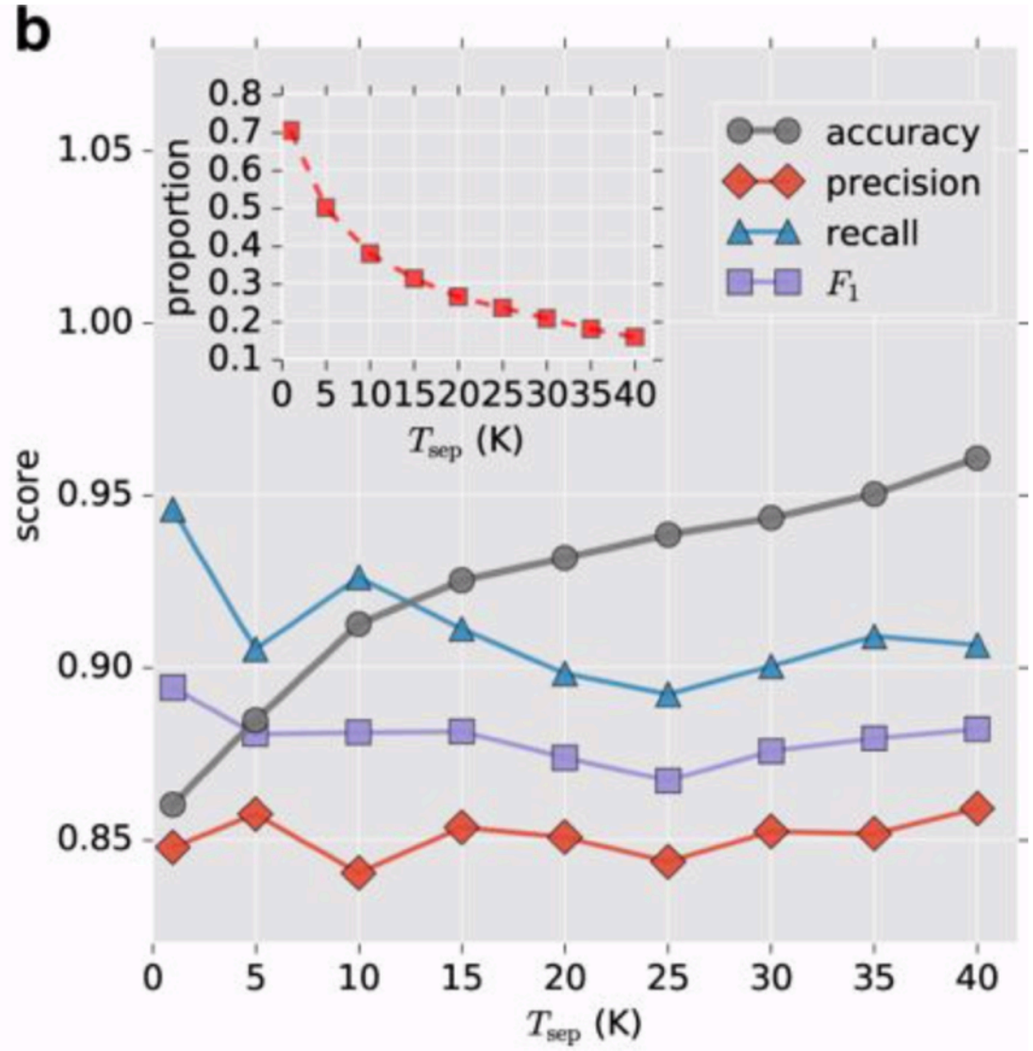
[Stanev et al. Nature Computational Materials, 2018]

Labels:

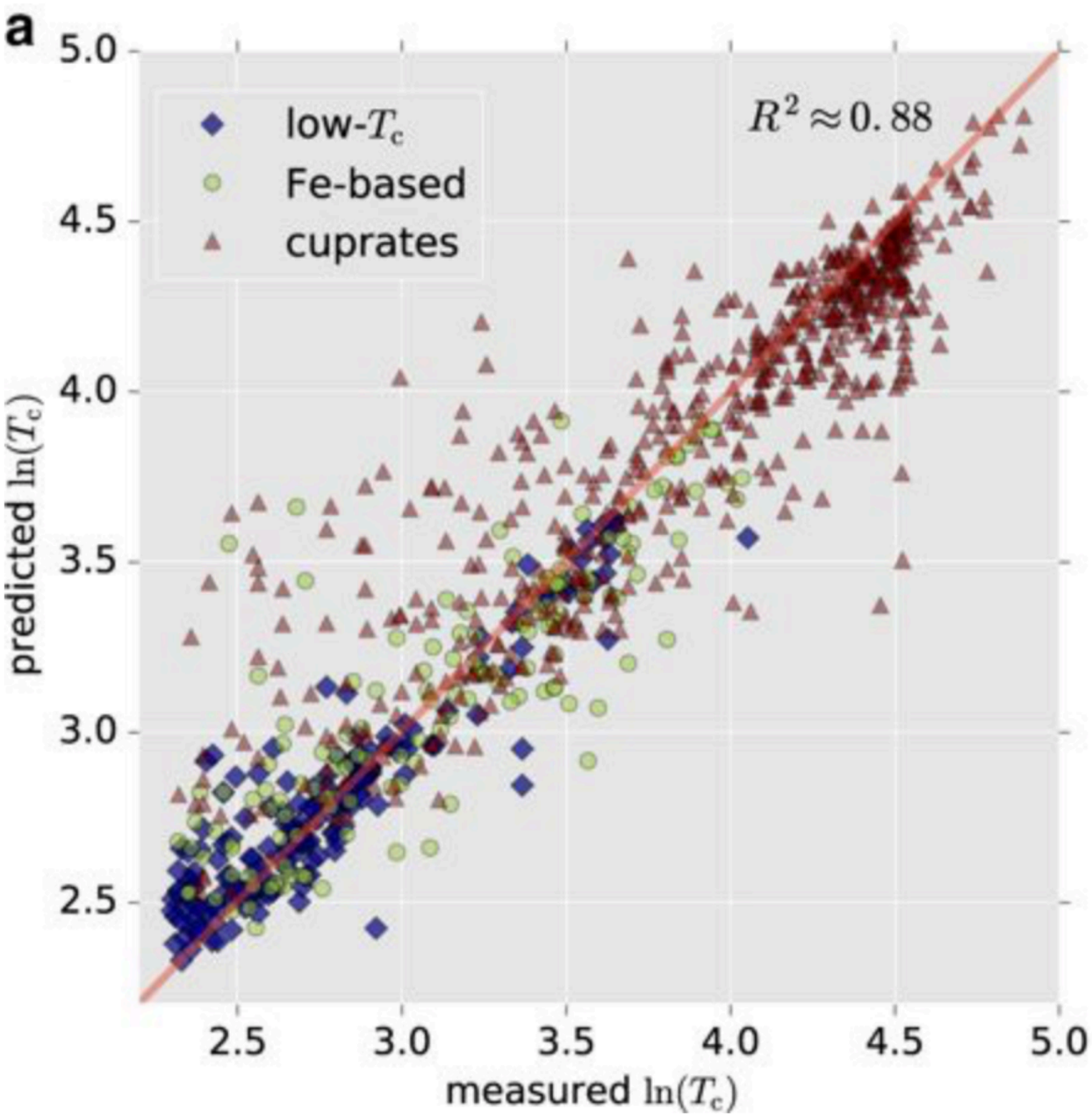
T_c (divided in low and high T_c class)

Problem: no information, what is NOT a superconductor





Machine learning modeling of T_c of superconductors



NN interpolates well but **cannot extrapolate** in complex parameter space

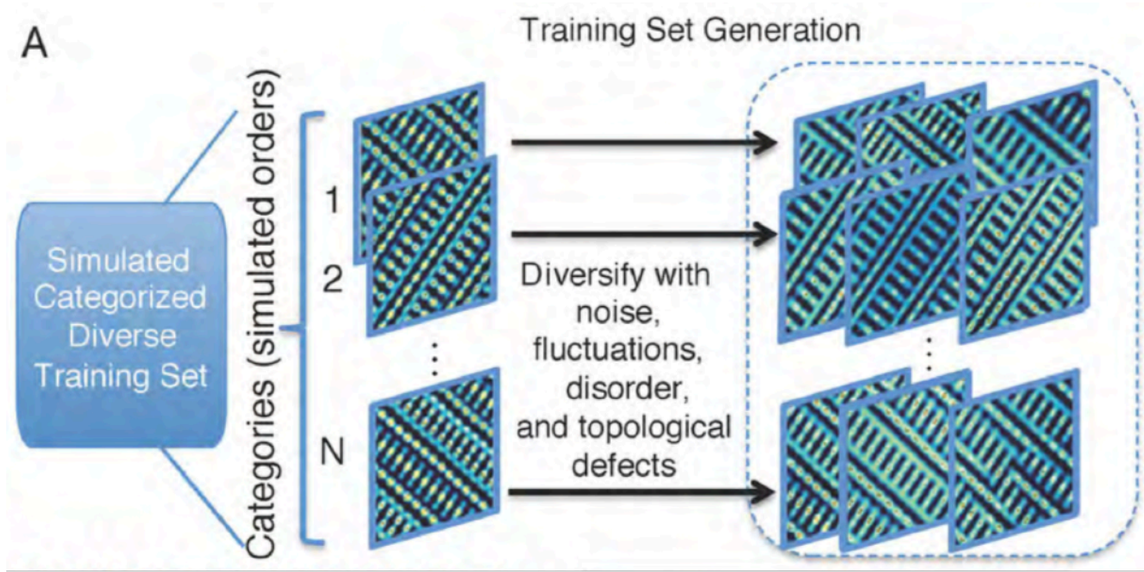
Processing spectroscopic data

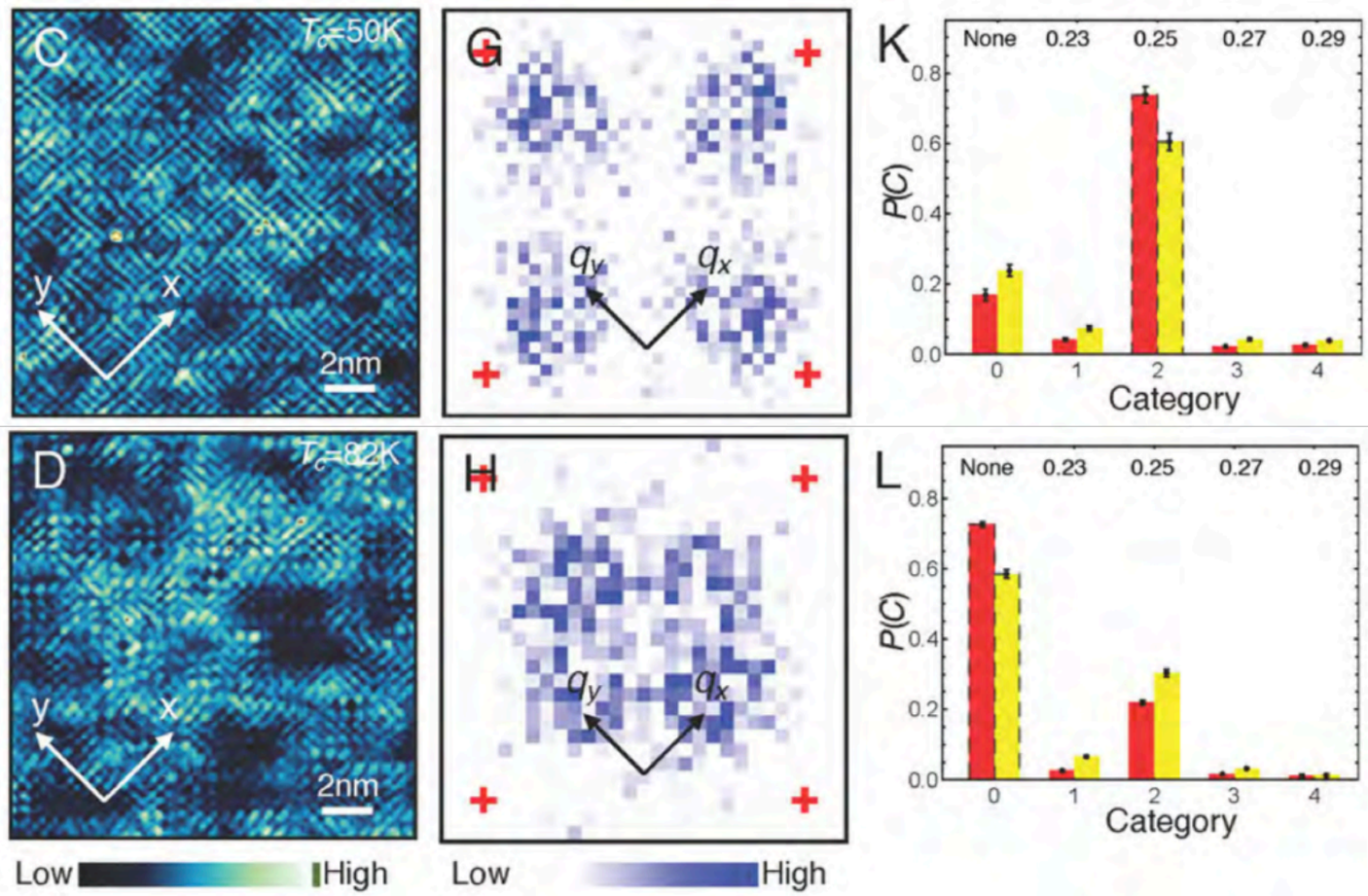
atomic-scale STM images are scanned for ordering phenomena breaking crystal symmetries

large data to be scanned (various energies, doping levels)

bottle neck is training data, overcome by generating artificial data

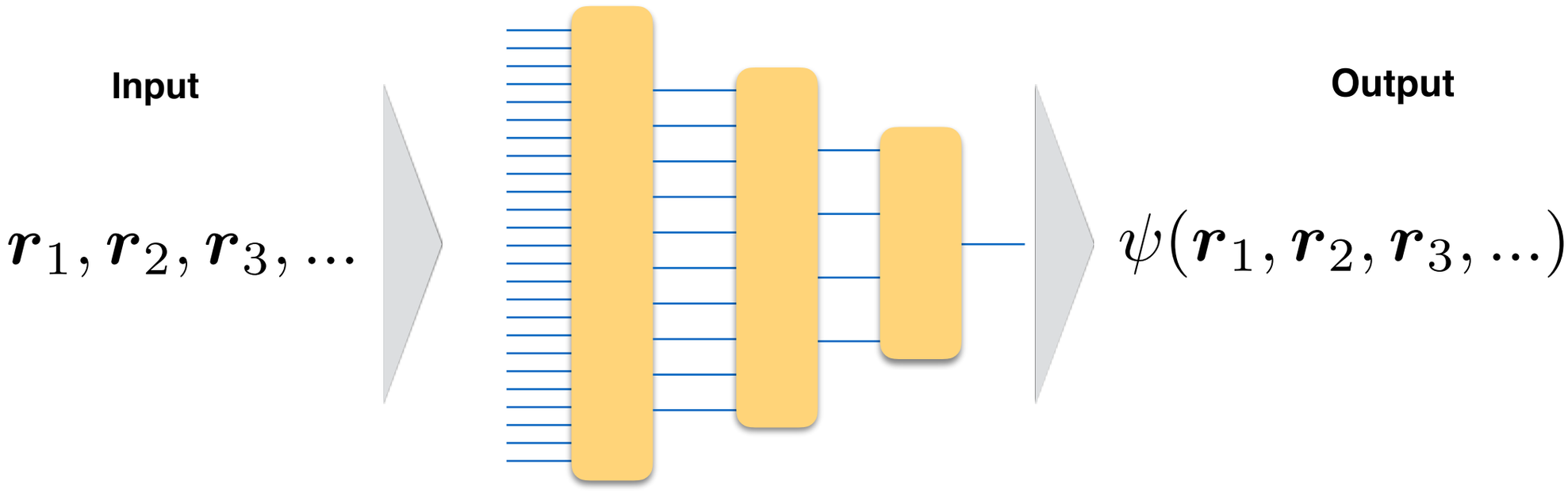
applied to cuprates and Mott insulators





[Zhang et al. https://arxiv.org/abs/1808.00479]

Neural networks as variational quantum states



Network represents one (compressed) many-body quantum state

Determine eigenstates of a given Hamiltonian variationally

Demonstrated first for Heisenberg model in various dimensions (ground state energy and time evolution for Hamiltonian with manifestly positive real elements)

[G. Carleo and M. Troyer, Science **355** (2017)]

Neural networks as variational quantum states
Network architecture

Random Boltzmann machine, RBM, (here one hidden layer)

$$\Psi(\boldsymbol{\sigma}) = \sum_{\boldsymbol{h}} e^{\sum_j a_j \sigma_j + \sum_i b_i h_j + \sum_{ij} h_i W_{ij} \sigma_j}$$

hidden spins

complex weights/biases

$$\log(\Psi(\boldsymbol{\sigma})) = \sum_j a_j \sigma_j + \sum_i \log\left[\cosh\left(b_i + \sum_j W_{ij}\sigma_j\right)\right]$$

RBMs have favorable analytical properties, allow for mathematical proofs, and admit physical interpretation

can be made deep

Neural networks as variational quantum states

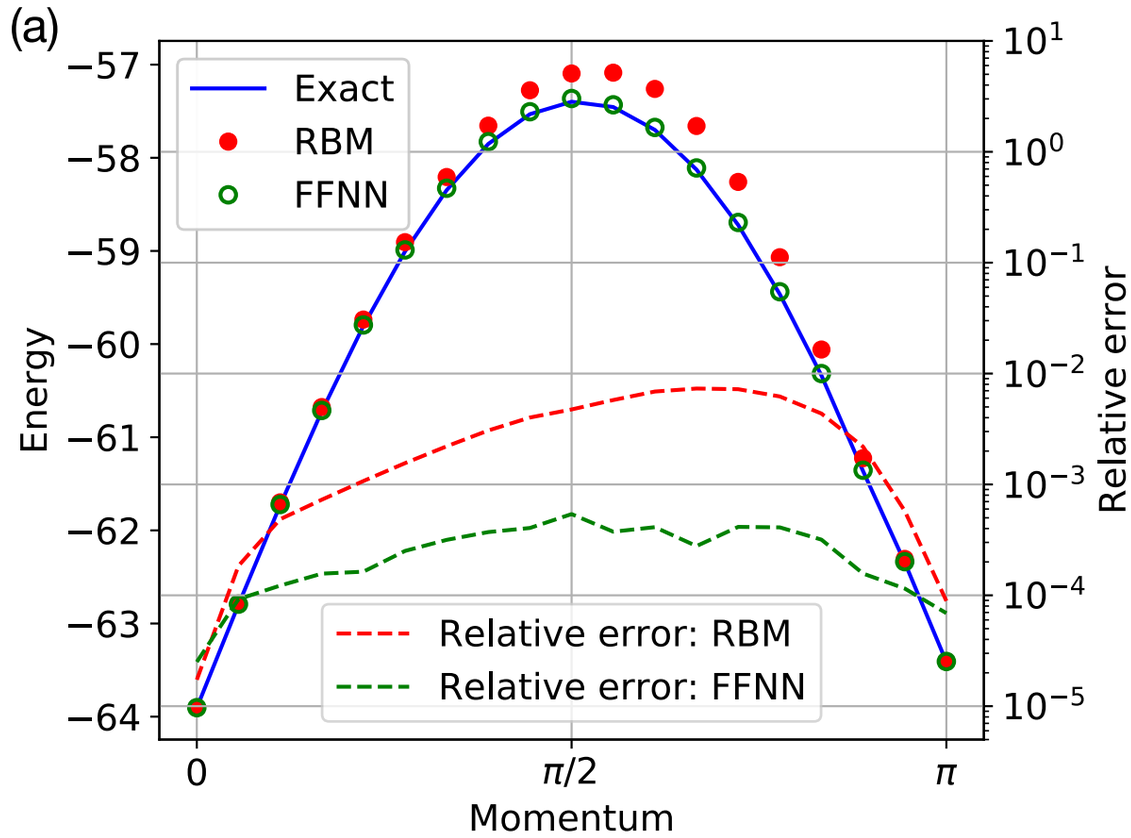
$$\hat{H} = 4 \sum_{i=1}^{L} \hat{\boldsymbol{S}}_i \cdot \hat{\boldsymbol{S}}_{i+1}$$

36 sites

~4000 network parameters

vs.

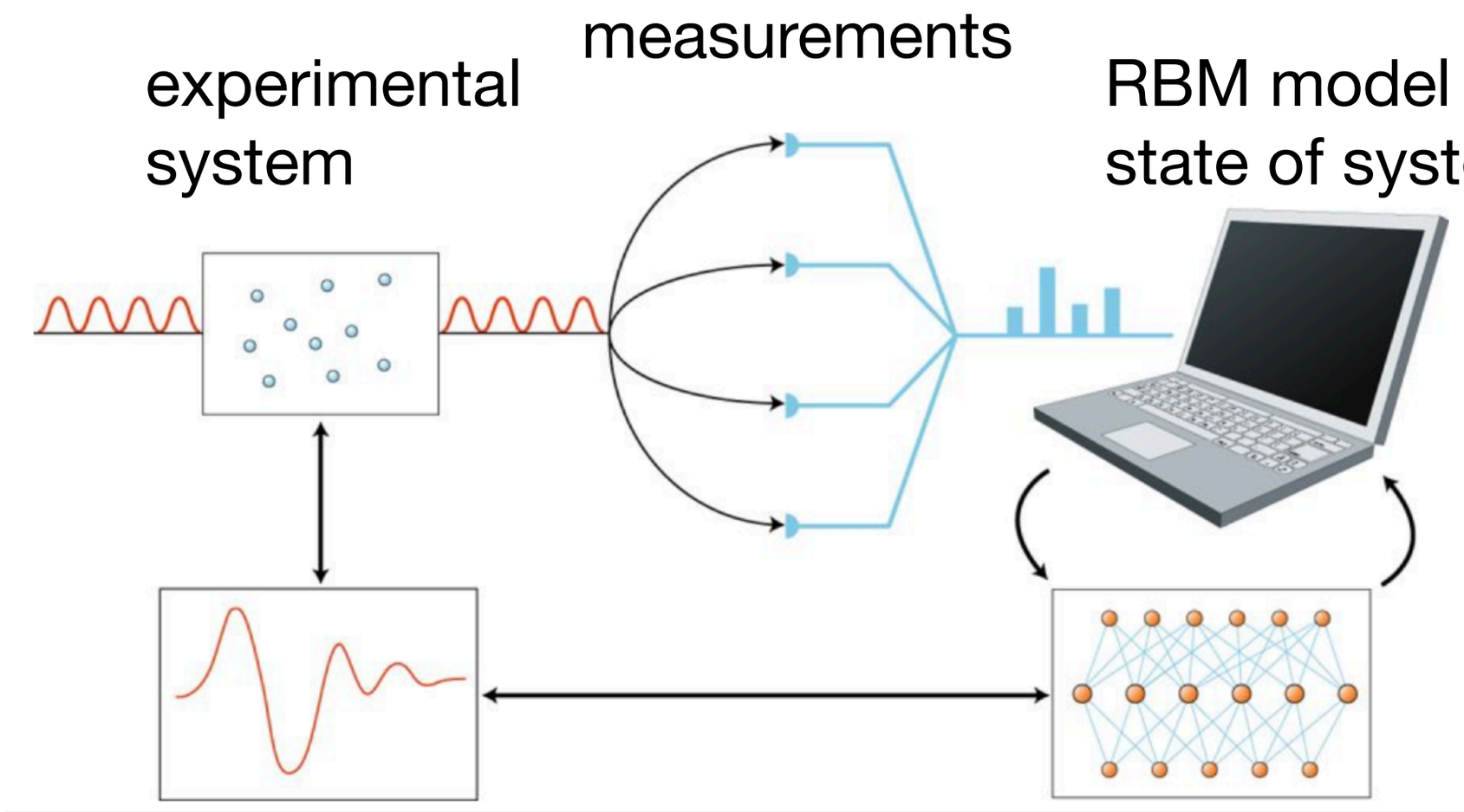
3×10^9 parameters in ED wave function



[Choo et al., PRL, 2018]

Neural networks as variational quantum states
Tomography

[Torlai et al., Nature Physics, 2018]



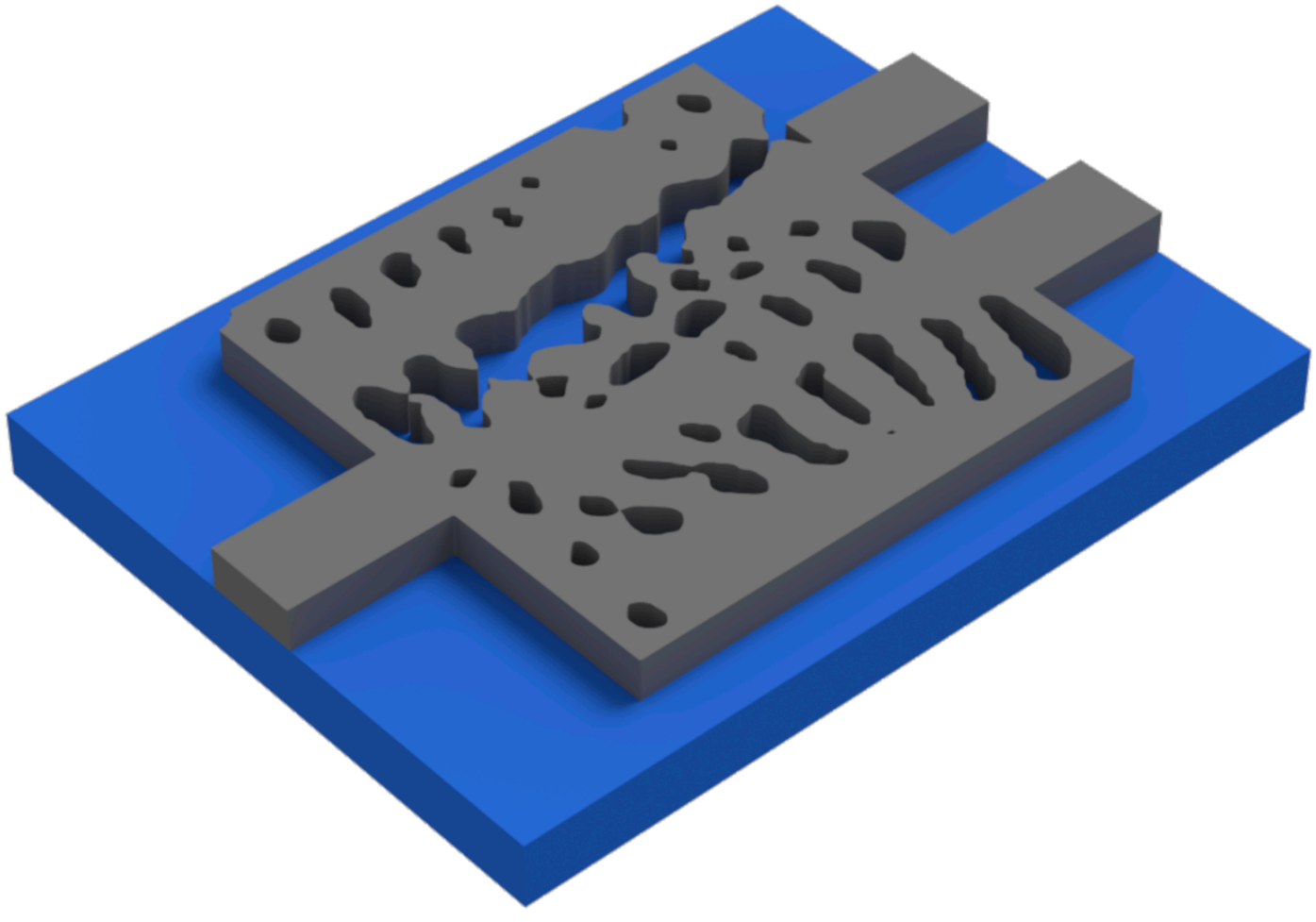
“glorious” fitting by constraints from measurements

tested theoretically on spin chains, W states, etc.

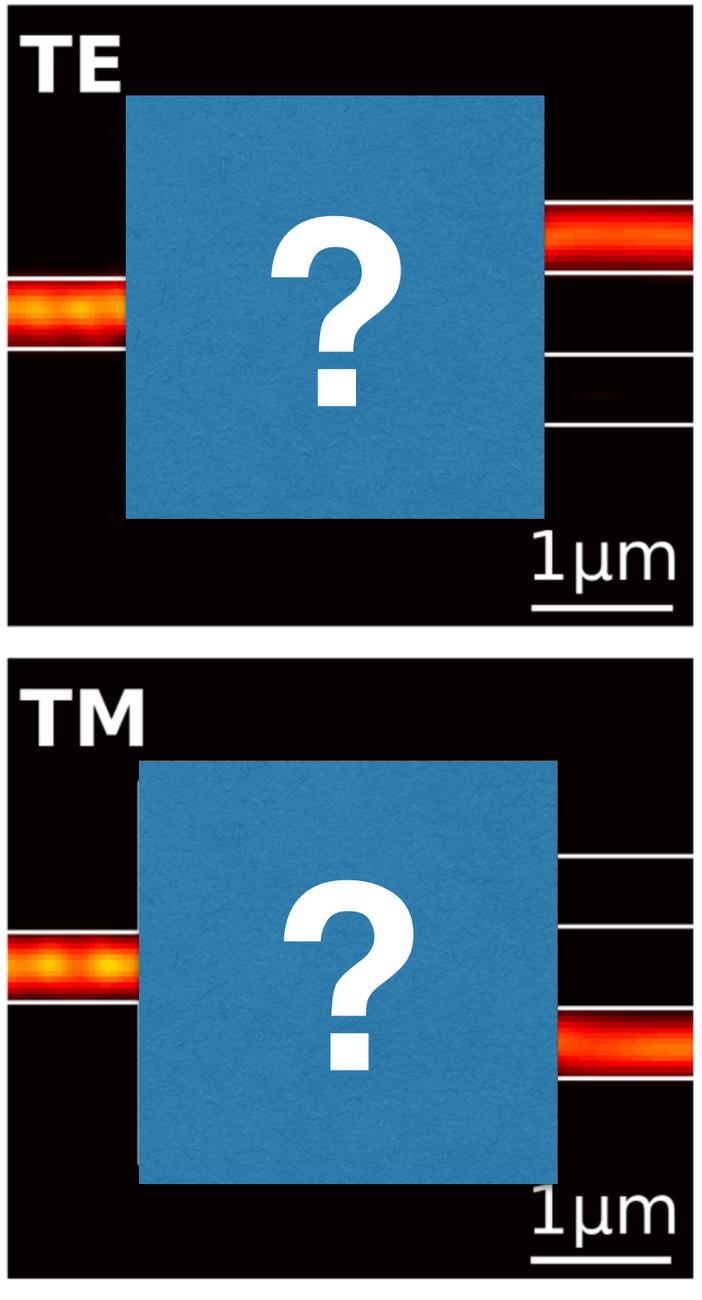
will be important for interpretation of “quantum computers”

Machine learning for device design

Objective: build photonic element that splits 1300 nm and 1550 nm light and has small footprint



performace/footprint better than conventional designs



[Molesky et al., Nature Physics, 2018]

Summary

- NNs are not the hammer for every nail, but work great in many cases
- low entry barrier: software packages even for specialized AI applications
- great variety of NN architectures and operations possible (invent your own!)
- think about: **data, network structure, loss function, training routine**
- there is more to ML than NN; try simplest first
- be prepared to give up some scientific rigor; explore what network has learned