Oct 20, 2025

Selected Topics in Computational Quantum Physics

量子物理计算方法选讲

Shuo Yang（杨硕）

Department of Physics, Tsinghua University

Email: shuoyang@tsinghua.edu.cn

WeChat: condmat-ys

Machine learning meets physics

The Nobel Prize in Physics 2024

Ill. Niklas Elmehed © Nobel Prize Outreach

John J. Hopfield

Prize share: 1/2

Ill. Niklas Elmehed © Nobel Prize Outreach

Geoffrey Hinton

Prize share: 1/2

The Nobel Prize in Physics 2024 was awarded jointly to John J. Hopfield and Geoffrey E. Hinton "for foundational discoveries and inventions that enable machine learning with artificial neural networks"

https://www.nobelprize.org/prizes/physics/2024/popular-information/

Machine learning meets physics

- neural network, supervised learning, back propagation
- unsupervised learning, restricted Boltzmann machine
- recent progress in physics

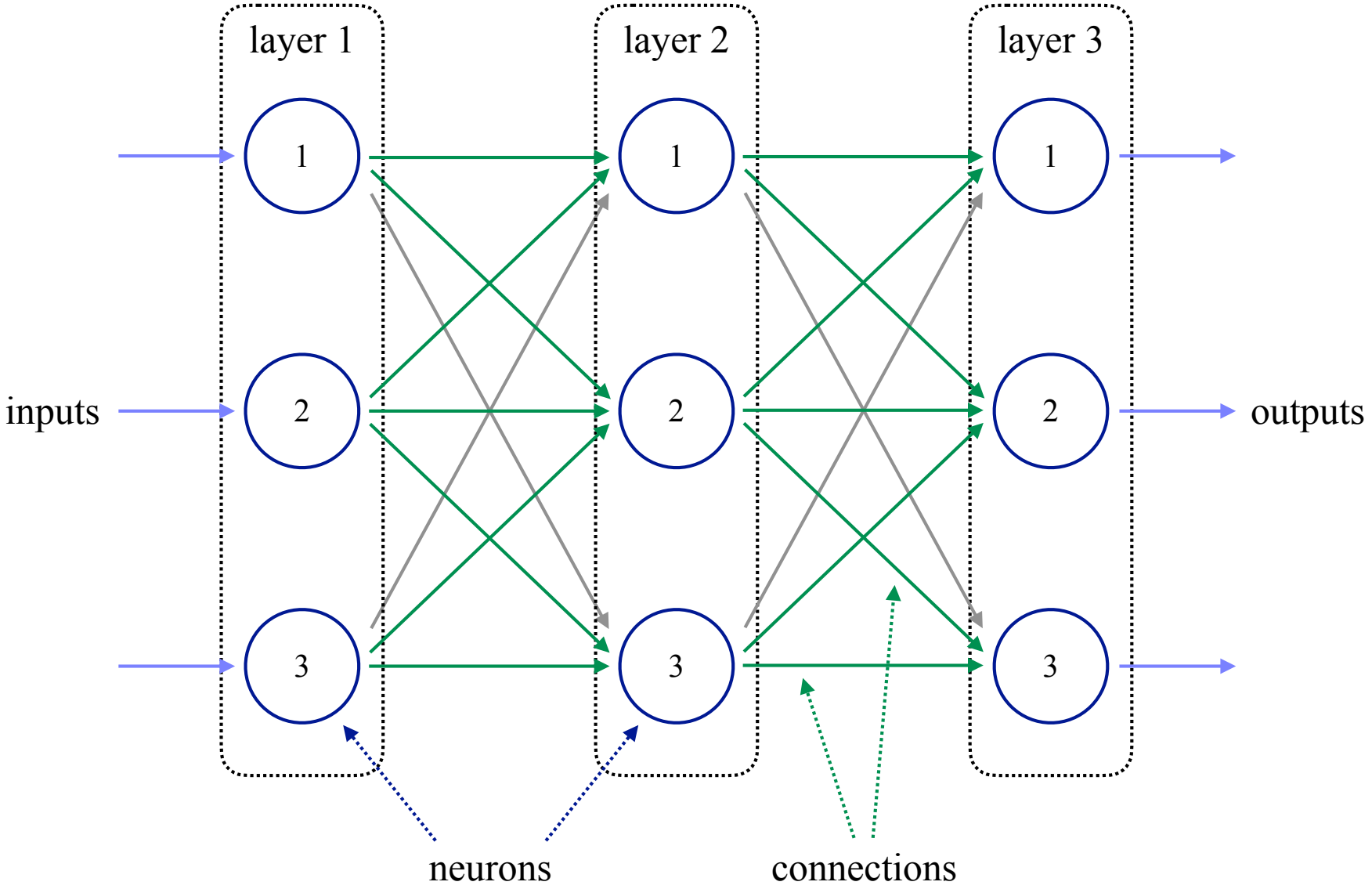
Selected references:
books:
Make your own neural network
https://neuralnetworksanddeeplearning.com

websites:
https://machine-learning-for-physicists.org/
https://physicsml.github.io/
http://deeplearning.net/tutorial/rbm.html

review articles:
Physics Reports 810 (2019) 1-124
Rev. Mod. Phys. 91, 045002 (2019)
arXiv:2112.00851

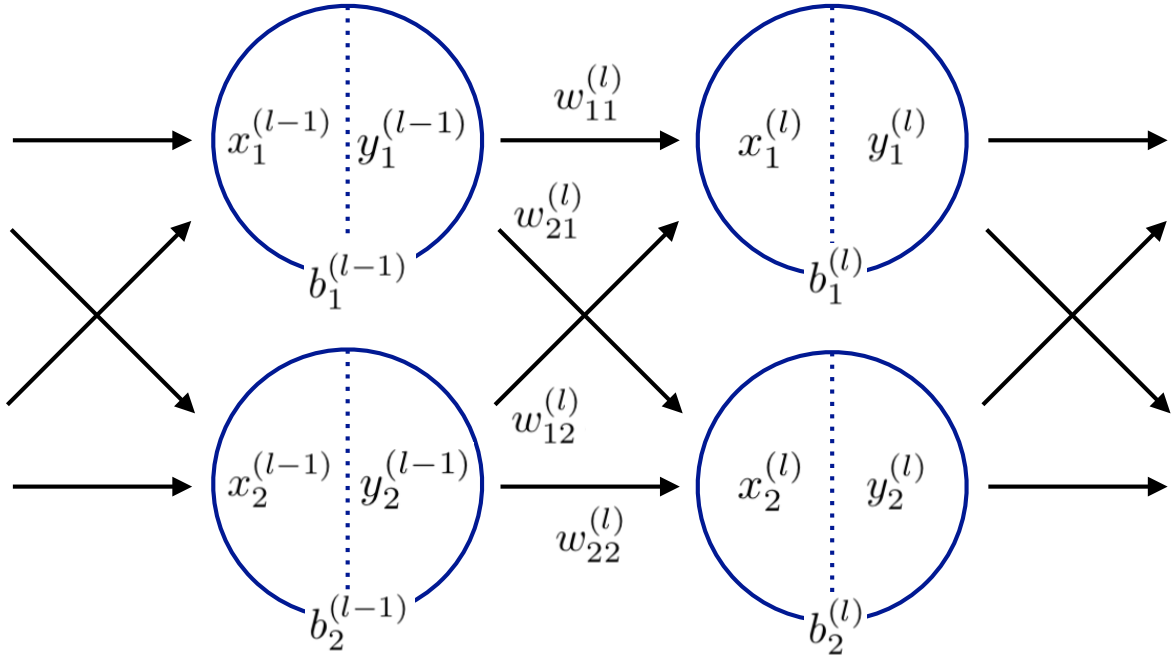
Deep neural network

- a three-layer neural network



Deep neural network

- simplest case: 2 layers, each with 2 neurons



- x and y are related by

$$\begin{aligned} x_1^{(l)} &= w_{11}^{(l)} y_1^{(l-1)} + w_{12}^{(l)} y_2^{(l-1)} + b_1^{(l)} \\ x_2^{(l)} &= w_{21}^{(l)} y_1^{(l-1)} + w_{22}^{(l)} y_2^{(l-1)} + b_2^{(l)} \end{aligned} \quad \Rightarrow \quad \begin{pmatrix} x_1^{(l)} \\ x_2^{(l)} \end{pmatrix} = \begin{pmatrix} w_{11}^{(l)} & w_{12}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} \end{pmatrix} \begin{pmatrix} y_1^{(l-1)} \\ y_2^{(l-1)} \end{pmatrix} + \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \end{pmatrix}$$

$$y_1^{(l)} = \text{sigmoid}\left(x_1^{(l)}\right)$$

$$y_2^{(l)} = \text{sigmoid}\left(x_2^{(l)}\right)$$

Deep neural network

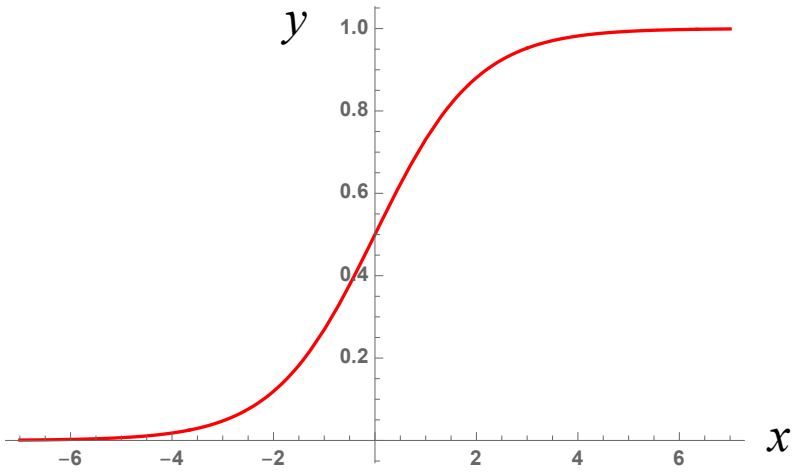
- in general

$$x_k^{(l)} = \sum_j w_{kj}^{(l)} y_j^{(l-1)} + b_k^{(l)}$$

$$\mathbf{x}^{(l)} = \mathbf{w}^{(l)} \mathbf{y}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{y}^{(l)} = \text{sigmoid}\left(\mathbf{x}^{(l)}\right)$$

- sigmoid function



$$y = \text{sigmoid}\,(x) = \frac{1}{1+e^{-x}}$$

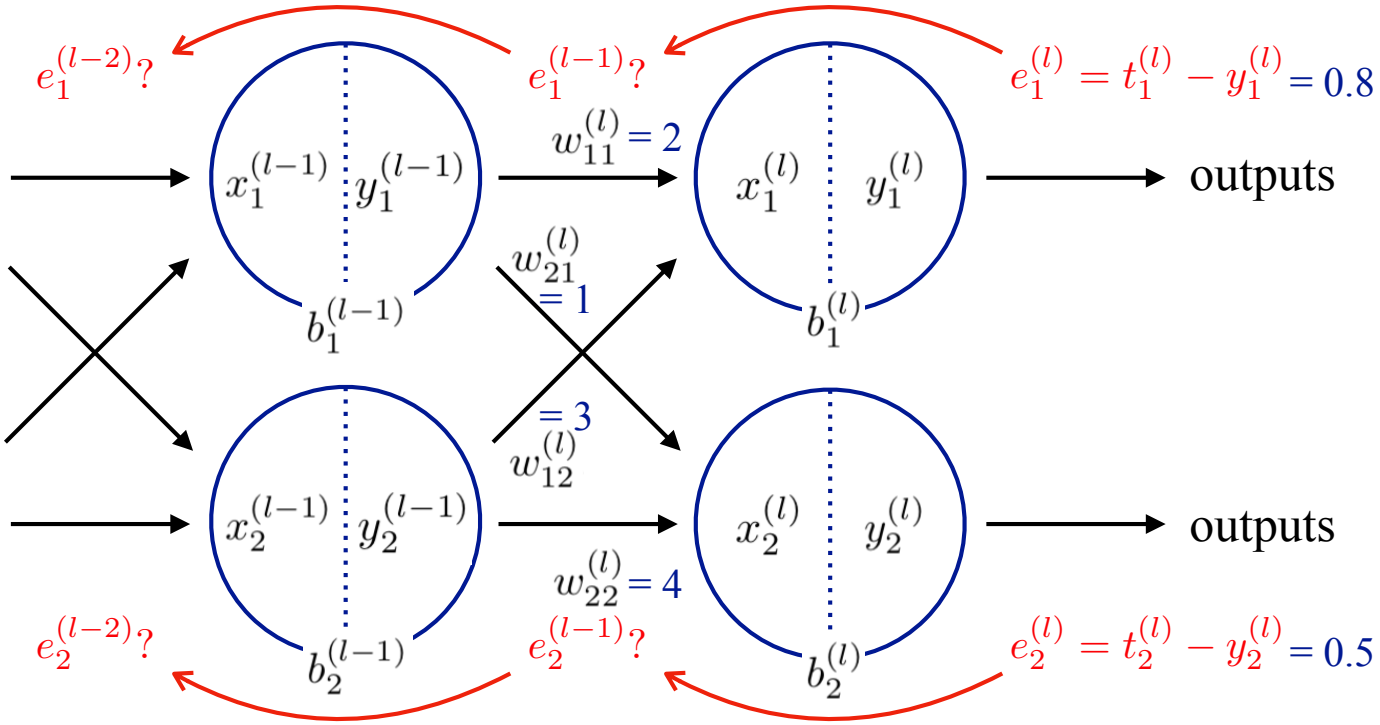
- the derivative of the sigmoid function has a simple form

$$\frac{\partial y}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{1+e^{-x}}\right) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}}$$

$$= \text{sigmoid}\,(x)\,(1-\text{sigmoid}\,(x)) = y\,(1-y)$$

Back propagation

- if the output layer has some error, how does it propagate to the previous layers ?



the larger the weight, the more of the error is carried back to the previous layer

$$e_1^{(l-1)} = 0.8 \times \frac{2}{2+3} + 0.5 \times \frac{1}{1+4} = 0.32 + 0.1 = 0.42$$

$$e_2^{(l-1)} = 0.8 \times \frac{3}{2+3} + 0.5 \times \frac{4}{1+4} = 0.48 + 0.4 = 0.88$$

- we ignore the normalization factor

$$\begin{pmatrix} e_1^{(l-1)} \\ e_2^{(l-1)} \end{pmatrix} = \begin{pmatrix} w_{11}^{(l)} & w_{21}^{(l)} \\ w_{12}^{(l)} & w_{22}^{(l)} \end{pmatrix} \begin{pmatrix} e_1^{(l)} \\ e_2^{(l)} \end{pmatrix}$$

$$\mathbf{e}^{(l-1)} = \mathbf{w}^{\mathrm{T}(l)} \mathbf{e}^{(l)}$$

Optimization

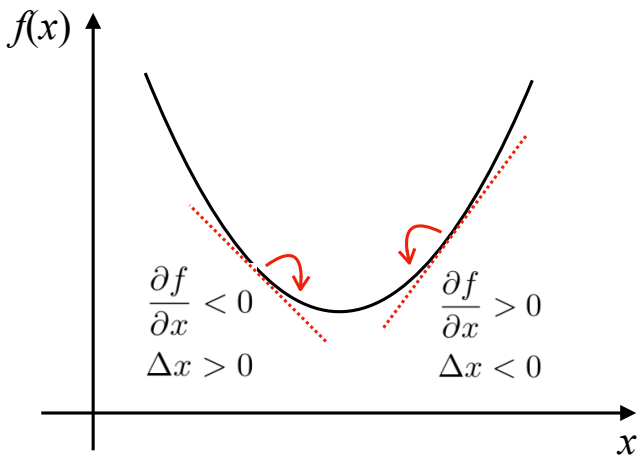
- define the cost function

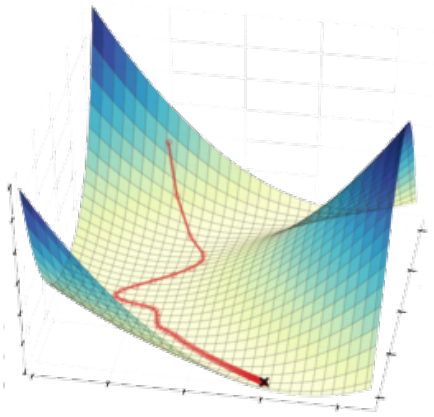
network output	target output	(target - actual)	|target - actual|	(target - actual)^2
0.4	0.5	0.1	0.1	0.01
0.8	0.7	-0.1	0.1	0.01
1.0	1.0	0	0	0
sum		0	0.2	0.02

$$E = \sum_n e_n^2 = \sum_n (t_n - y_n)^2$$

$$e_n = t_n - y_n$$

- find the minimal by gradient descent





$$x_{\text{new}} = x_{\text{old}} - \alpha \frac{\partial f}{\partial x}$$

Update weight and bias

- given the error $E^{(l)}$ at the l-th layer, we will update $w_{kj}^{(l)}$ and $b_k^{(l)}$
  j belongs to the (l-1)-th layer and k belongs to the l-th layer

- we have shown

$$E^{(l)} = \sum_k \left(e_k^{(l)}\right)^2 = \sum_k \left(t_k^{(l)} - y_k^{(l)}\right)^2$$

$$\frac{\partial E^{(l)}}{\partial y_k^{(l)}} = -2\left(t_k^{(l)} - y_k^{(l)}\right) = -2e_k^{(l)}$$

$$x_k^{(l)} = \sum_j w_{kj}^{(l)} y_j^{(l-1)} + b_k^{(l)}$$

$$\mathbf{e}^{(l-1)} = \mathbf{w}^{\mathrm{T}(l)} \mathbf{e}^{(l)}$$

$$y_k^{(l)} = \text{sigmoid}\left(x_k^{(l)}\right)$$

$$\frac{\partial y_k^{(l)}}{\partial x_k^{(l)}} = y_k^{(l)}\left(1 - y_k^{(l)}\right)$$

- we finally get

$$\frac{\partial E^{(l)}}{\partial w_{kj}^{(l)}} = \frac{\partial E^{(l)}}{\partial y_k^{(l)}} \frac{\partial y_k^{(l)}}{\partial x_k^{(l)}} \frac{\partial x_k^{(l)}}{\partial w_{kj}^{(l)}} = \left[-2e_k^{(l)}\right]\left[y_k^{(l)}\left(1 - y_k^{(l)}\right)\right]\left[y_j^{(l-1)}\right] \propto -e_k^{(l)} y_k^{(l)}\left(1 - y_k^{(l)}\right) y_j^{(l-1)}$$

$$\frac{\partial E^{(l)}}{\partial b_k^{(l)}} = \frac{\partial E^{(l)}}{\partial y_k^{(l)}} \frac{\partial y_k^{(l)}}{\partial x_k^{(l)}} \frac{\partial x_k^{(l)}}{\partial b_k^{(l)}} = \left[-2e_k^{(l)}\right]\left[y_k^{(l)}\left(1 - y_k^{(l)}\right)\right][1] \propto -e_k^{(l)} y_k^{(l)}\left(1 - y_k^{(l)}\right)$$

$$w_{k,j,\text{new}}^{(l)} = w_{k,j,\text{old}}^{(l)} - \alpha \frac{\partial E^{(l)}}{\partial w_{kj}^{(l)}}$$

$$b_{k,\text{new}}^{(l)} = b_{k,\text{old}}^{(l)} - \alpha \frac{\partial E^{(l)}}{\partial b_k^{(l)}}$$

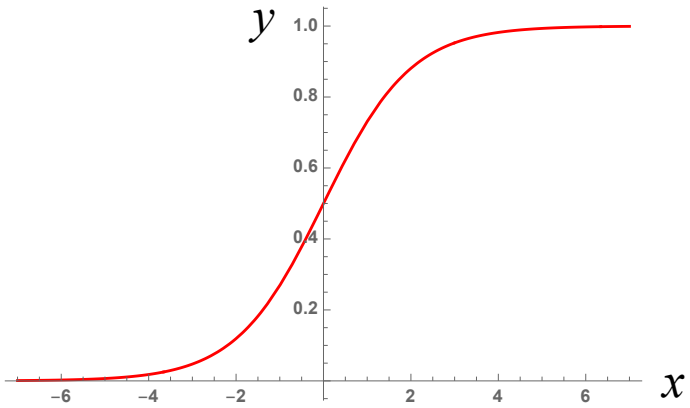
Preparing initial data

- inputs

  large x → small gradient → reduced learning

  keep the inputs small but not zero

  rescale to [0.01, 0.99] or [-1, +1]

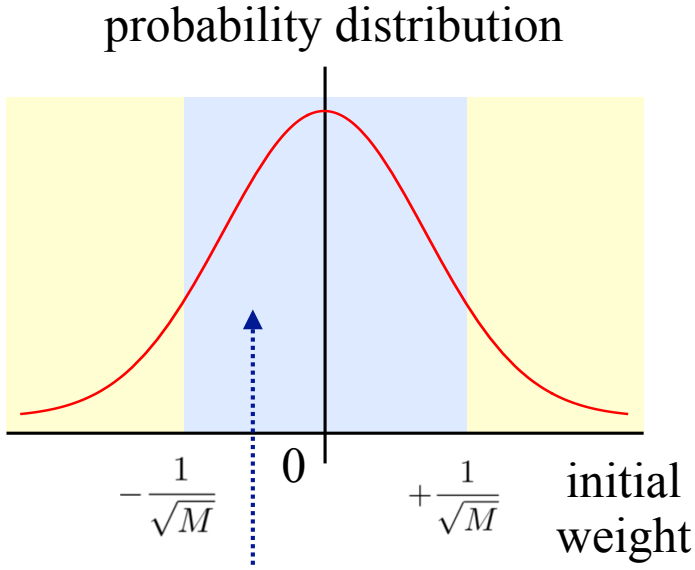


- outputs

  rescale target values to match the outputs possible from the activation function, e.g. [0.01, 0.99]

- weights

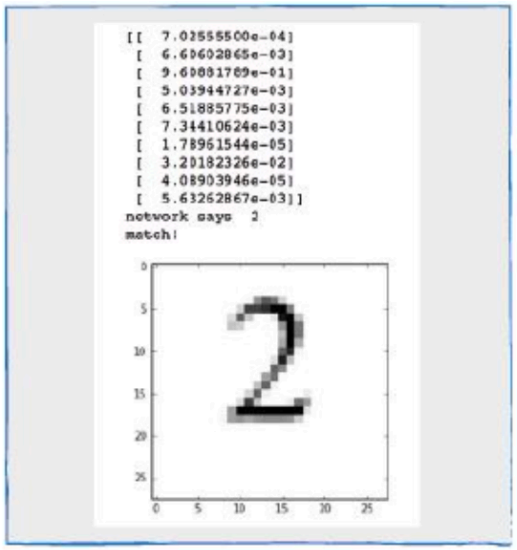
1. randomly and uniformly from [-1,+1]
2. randomly sampling from $\left[-\frac{1}{\sqrt{M}}, +\frac{1}{\sqrt{M}}\right]$
3. combine with normal distribution

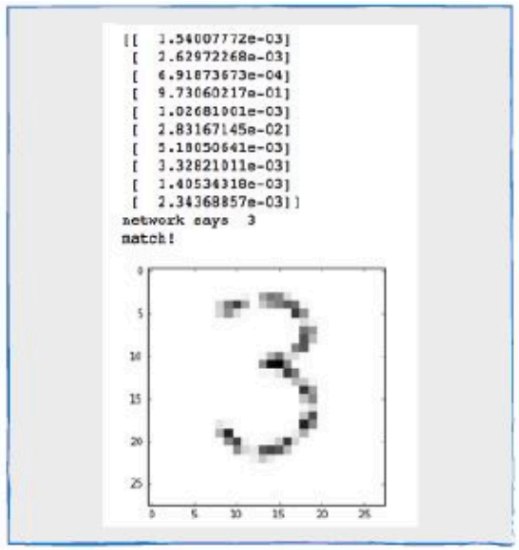


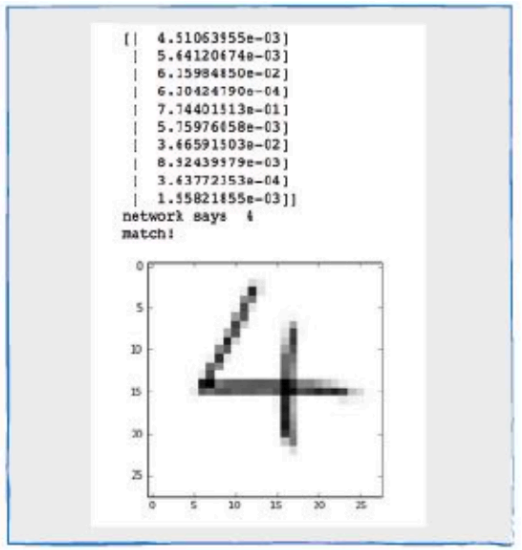
range for initial weight avoiding zero

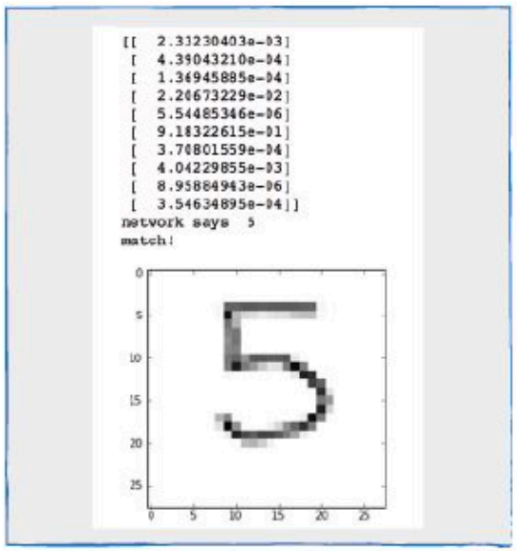
Example: handwriting recognition

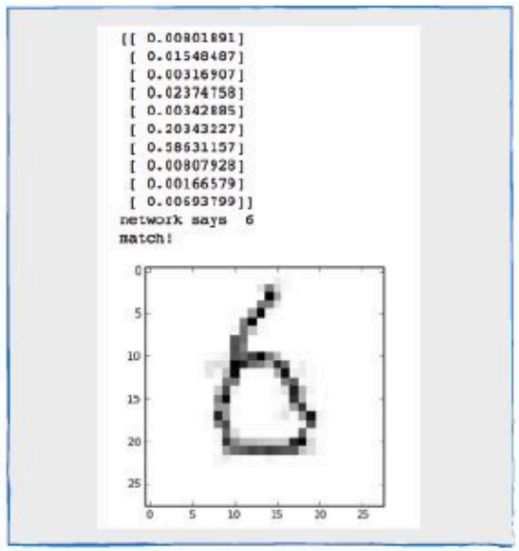
https://github.com/makeyourownneuralnetwork/makeyourownneuralnetwork/blob/master/part3_neural_network_mnist_and_own_data.ipynb

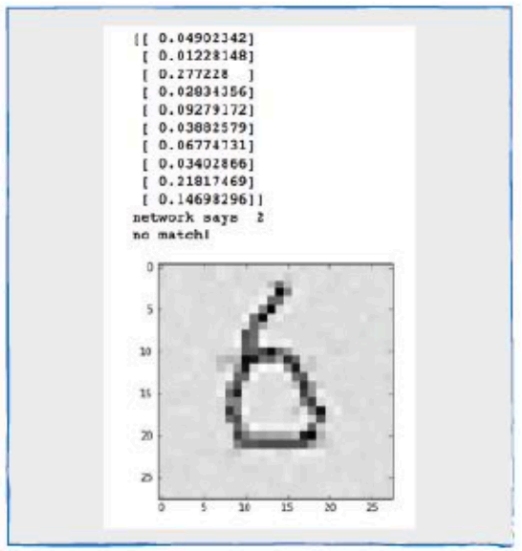












Machine learning meets physics

- neural network, supervised learning, back propagation
- unsupervised learning, restricted Boltzmann machine
- recent progress in physics

Selected references:
books:
Make your own neural network
https://neuralnetworksanddeeplearning.com

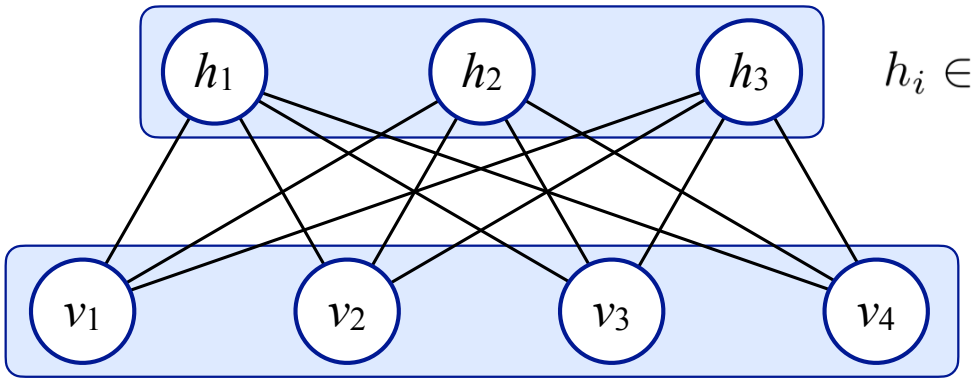
websites:
https://machine-learning-for-physicists.org/
https://physicsml.github.io/
http://deeplearning.net/tutorial/rbm.html

review articles:
Physics Reports 810 (2019) 1-124
Rev. Mod. Phys. 91, 045002 (2019)
arXiv:2112.00851

Restricted Boltzmann Machine (RBM)

hidden layer

visible layer



$h_i \in \{0, 1\}$

$v_j \in \{0, 1\}$

- the joint probability distribution is given by the Gibbs distribution

$$p(\mathbf{v}, \mathbf{h}) = \exp(-E(\mathbf{v}, \mathbf{h}))/Z \qquad Z = \sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$$

with the energy function

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}) &= -\sum_{i=1}^{M}\sum_{j=1}^{N} h_i w_{ij} v_j - \sum_{i=1}^{M} b_i h_i - \sum_{j=1}^{N} c_j v_j \\ &= -\mathbf{h}^T \mathbf{w}\mathbf{v} - \mathbf{b}^T \mathbf{h} - \mathbf{c}^T \mathbf{v} \end{aligned}$$

- RBM has no connections between two variables of the same layer
  hidden variables are independent given the visible variables, and vice versa

$$p(\mathbf{h} \mid \mathbf{v}) = \prod_{i=1}^{M} p(h_i \mid \mathbf{v}) \qquad p(\mathbf{v} \mid \mathbf{h}) = \prod_{j=1}^{N} p(v_j \mid \mathbf{h})$$

Conditional probability

$$
\begin{aligned}
p(\mathbf{h} \mid \mathbf{v}) &= \frac{p(\mathbf{v}, \mathbf{h})}{p(\mathbf{v})} = \frac{p(\mathbf{v}, \mathbf{h})}{\sum_{\mathbf{h}'} p(\mathbf{v}, \mathbf{h}')} \\
&= \frac{\exp\left(\mathbf{h}^T \mathbf{W} \mathbf{v} + \mathbf{b}^T \mathbf{h} + \mathbf{c}^T \mathbf{v}\right) / Z}{\sum_{\mathbf{h}'} \exp\left(\mathbf{h}'^T \mathbf{W} \mathbf{v} + \mathbf{b}^T \mathbf{h}' + \mathbf{c}^T \mathbf{v}\right) / Z} = \frac{\exp\left[\sum_i \left(h_i \mathbf{W}_i \mathbf{v} + b_i h_i\right)\right]}{\sum_{\mathbf{h}'} \exp\left[\sum_i \left(h'_i \mathbf{W}_i \mathbf{v} + b_i h'_i\right)\right]} \\
&= \frac{\prod_i \exp\left(h_i \mathbf{W}_i \mathbf{v} + b_i h_i\right)}{\prod_i \left[\sum_{h'_i} \exp\left(h'_i \mathbf{W}_i \mathbf{v} + b_i h'_i\right)\right]} = \prod_i \left[\frac{\exp\left(h_i \mathbf{W}_i \mathbf{v} + b_i h_i\right)}{1 + \exp\left(\mathbf{W}_i \mathbf{v} + b_i\right)}\right] = \prod_i p(h_i \mid \mathbf{v})
\end{aligned}
$$

$$
p(h_i = 1 \mid \mathbf{v}) = \frac{\exp\left(\mathbf{W}_i \mathbf{v} + b_i\right)}{1 + \exp\left(\mathbf{W}_i \mathbf{v} + b_i\right)} = \frac{1}{1 + \exp\left[-\left(\mathbf{W}_i \mathbf{v} + b_i\right)\right]} = \text{sigmoid}\left(\mathbf{W}_i \mathbf{v} + b_i\right)
$$

$$
\begin{aligned}
p(\mathbf{v} \mid \mathbf{h}) &= \frac{p(\mathbf{v}, \mathbf{h})}{p(\mathbf{h})} = \frac{p(\mathbf{v}, \mathbf{h})}{\sum_{\mathbf{v}'} p(\mathbf{v}', \mathbf{h})} \\
&= \frac{\exp\left(\mathbf{h}^T \mathbf{W} \mathbf{v} + \mathbf{b}^T \mathbf{h} + \mathbf{c}^T \mathbf{v}\right) / Z}{\sum_{\mathbf{v}'} \exp\left(\mathbf{h}^T \mathbf{W} \mathbf{v}' + \mathbf{b}^T \mathbf{h} + \mathbf{c}^T \mathbf{v}'\right) / Z} = \frac{\exp\left[\sum_j \left(\mathbf{h}^T \mathbf{W}_j v_j + c_j v_j\right)\right]}{\sum_{\mathbf{v}'} \exp\left[\sum_j \left(\mathbf{h}^T \mathbf{W}_j v'_j + c_j v'_j\right)\right]} \\
&= \frac{\prod_j \exp\left(\mathbf{h}^T \mathbf{W}_j v_j + c_j v_j\right)}{\prod_j \left[\sum_{v'_j} \exp\left(\mathbf{h}^T \mathbf{W}_j v'_j + c_j v'_j\right)\right]} = \prod_i \left[\frac{\exp\left(\mathbf{h}^T \mathbf{W}_j v_j + c_j v_j\right)}{1 + \exp\left(\mathbf{h}^T \mathbf{W}_j + c_j\right)}\right] = \prod_j p(v_j \mid \mathbf{h})
\end{aligned}
$$

$$
p(v_j = 1 \mid \mathbf{h}) = \frac{\exp\left(\mathbf{h}^T \mathbf{W}_j + c_j\right)}{1 + \exp\left(\mathbf{h}^T \mathbf{W}_j + c_j\right)} = \frac{1}{1 + \exp\left[-\left(\mathbf{h}^T \mathbf{W}_j + c_j\right)\right]} = \text{sigmoid}\left(\mathbf{h}^T \mathbf{W}_j + c_j\right)
$$

Log-likelihood and KL-divergence

- unsupervised learning means learning an unknown distribution q based on sample data $S = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_l\}$

- training corresponds to finding the parameters that maximize the likelihood

$$\mathcal{L}(\theta \mid S) = \prod_{i=1}^{l} p(\mathbf{v}_i \mid \theta)$$

- maximizing the likelihood is the same as maximizing the log-likelihood

$$\ln \mathcal{L}(\theta \mid S) = \ln \prod_{i=1}^{l} p(\mathbf{v}_i \mid \theta) = \sum_{i=1}^{l} \ln p(\mathbf{v}_i \mid \theta)$$

- maximizing the log-likelihood corresponds to minimizing the Kullback-Leibler (KL) divergence

$$\mathrm{KL}(q \mid p) = \sum_{\mathbf{v}} q(\mathbf{v}) \ln \frac{q(\mathbf{v})}{p(\mathbf{v})} = \sum_{\mathbf{v}} q(\mathbf{v}) \ln q(\mathbf{v}) - \sum_{\mathbf{v}} q(\mathbf{v}) \ln p(\mathbf{v})$$

- KL-divergence is a non-symmetric measure of the difference between two distributions, always ≥ 0

  KL-divergence = 0 if and only if the two distributions are the same

Derivative of log-likelihood

- we calculate the derivative of the log-likelihood

since $$p(\mathbf{v} \mid \theta) = p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h})}{Z} = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}$$

we have $$\frac{\partial \ln p(\mathbf{v} \mid \theta)}{\partial \theta} = \frac{\partial}{\partial \theta}\left[\ln \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\right] - \frac{\partial}{\partial \theta}\left[\ln \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\right]$$

$$= -\sum_{\mathbf{h}} \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{h}'} e^{-E(\mathbf{v},\mathbf{h}')}} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} + \sum_{\mathbf{v},\mathbf{h}} \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v}',\mathbf{h}'} e^{-E(\mathbf{v}',\mathbf{h}')}} \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta}$$

$$= -\sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} + \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta}$$

$$= -\sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} + \sum_{\mathbf{v}} p(\mathbf{v}) \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta}$$

where $$E(\mathbf{v},\mathbf{h}) = -\sum_{i=1}^{M}\sum_{j=1}^{N} h_i w_{ij} v_j - \sum_{i=1}^{M} b_i h_i - \sum_{j=1}^{N} c_j v_j$$

- for RBM, θ can be weight or bias

$$\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}} = -h_i v_j, \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_i} = -h_i, \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_j} = -v_j$$

Derivative of log-likelihood

- given a single training pattern $\mathbf{v}$, we get the derivative of the log-likelihood

$$\begin{aligned}
\frac{\partial \ln p(\mathbf{v} \mid w_{ij})}{\partial w_{ij}} &= \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) h_i v_j - \sum_{\mathbf{v}} p(\mathbf{v}) \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) h_i v_j \\
&= p(h_i = 1 \mid \mathbf{v}) v_j - \sum_{\mathbf{v}} p(\mathbf{v}) p(h_i = 1 \mid \mathbf{v}) v_j \\
\frac{\partial \ln p(\mathbf{v} \mid b_i)}{\partial b_i} &= \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) h_i - \sum_{\mathbf{v}} p(\mathbf{v}) \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) h_i \\
&= p(h_i = 1 \mid \mathbf{v}) - \sum_{\mathbf{v}} p(\mathbf{v}) p(h_i = 1 \mid \mathbf{v}) \\
\frac{\partial \ln p(\mathbf{v} \mid c_j)}{\partial c_j} &= \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) v_j - \sum_{\mathbf{v}} p(\mathbf{v}) \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) v_j \\
&= v_j - \sum_{\mathbf{v}} p(\mathbf{v}) v_j
\end{aligned}$$

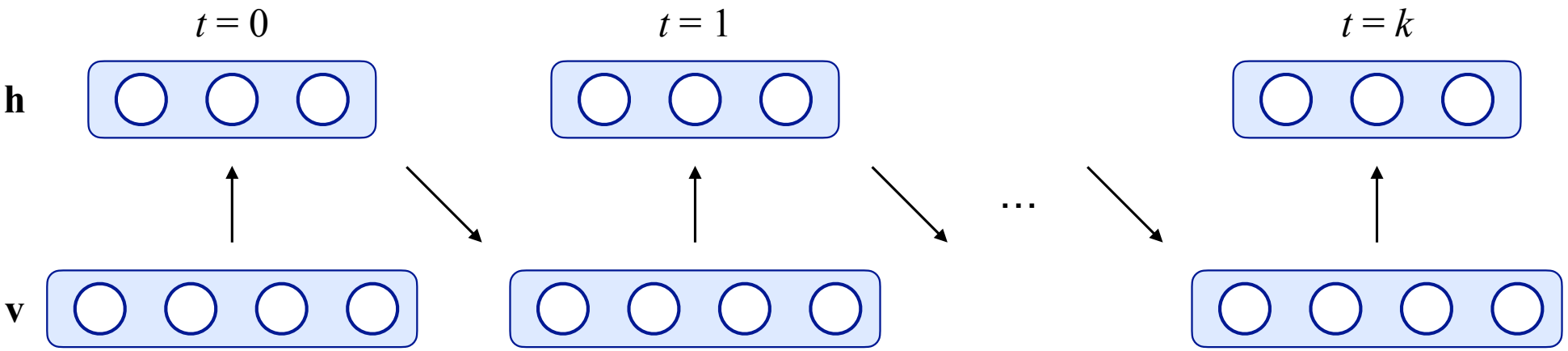
- the mean of the derivative over a training set $S = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m\}$ is

$$\frac{1}{m} \sum_{\mathbf{v} \in S} \frac{\partial \ln p(\mathbf{v} \mid w_{ij})}{\partial w_{ij}} = \frac{1}{m} \sum_{\mathbf{v} \in S} \left[\langle h_i v_j \rangle_{p(\mathbf{h} \mid \mathbf{v})} - \langle h_i v_j \rangle_{p(\mathbf{h}, \mathbf{v})} \right] = \langle h_i v_j \rangle_{\text{data}} - \langle h_i v_j \rangle_{\text{model}}$$

- for each $\mathbf{v}_i$, the first term of the derivative only depends on $\mathbf{v}_i$
  while the second term of the derivative depends on the whole Markov chain

Approximating the log-likelihood gradient

- obtaining unbiased estimates of log-likelihood gradient using MCMC methods typically requires many sampling steps
- however, after running the chain for a few steps can be sufficient for model training → contrastive divergence (CD) learning
- *k*-step Contrastive Divergence (CD-*k*): Gibbs chain is run for only *k* steps
- Gibbs chain is initialized with a training example $\mathbf{v}^{(0)}$ and yields $\mathbf{v}^{(k)}$ after *k* steps
- in each step *t*, sampling $\mathbf{h}^{(t)}$ from $p\left(\mathbf{h} \mid \mathbf{v}^{(t)}\right)$ and sampling $\mathbf{v}^{(t+1)}$ from $p\left(\mathbf{v} \mid \mathbf{h}^{(t)}\right)$



k-step contrastive divergence learning

- the log-likelihood gradient

$$\frac{\partial \ln p(\mathbf{v} \mid \theta)}{\partial \theta} = -\sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta} + \sum_{\mathbf{v}} p(\mathbf{v}) \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta}$$

  is approximated by

$$\mathrm{CD}_k\left(\theta, \mathbf{v}^{(0)}\right) = -\sum_{\mathbf{h}} p\left(\mathbf{h} \mid \mathbf{v}^{(0)}\right) \frac{\partial E\left(\mathbf{v}^{(0)}, \mathbf{h}\right)}{\partial \theta} + \sum_{\mathbf{h}} p\left(\mathbf{h} \mid \mathbf{v}^{(k)}\right) \frac{\partial E\left(\mathbf{v}^{(k)}, \mathbf{h}\right)}{\partial \theta}$$

- when θ is taken as weight and bias

$$\begin{aligned}
\Delta w_{ij} &= p\left(h_i = 1 \mid \mathbf{v}^{(0)}\right) v_j^{(0)} - p\left(h_i = 1 \mid \mathbf{v}^{(k)}\right) v_j^{(k)} \\
\Delta b_i &= p\left(h_i = 1 \mid \mathbf{v}^{(0)}\right) - p\left(h_i = 1 \mid \mathbf{v}^{(k)}\right) \\
\Delta c_j &= v_j^{(0)} - v_j^{(k)}
\end{aligned}$$

- CD is a biased approximation, the bias vanishes as $k \to \infty$

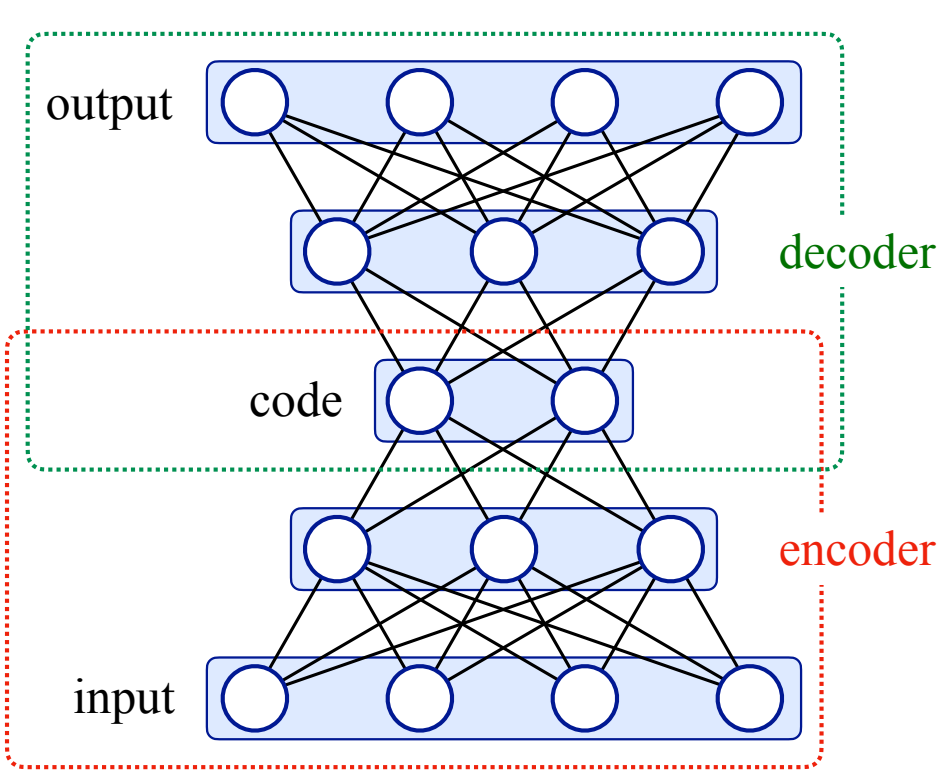
  it does not maximize the likelihood of the data under the model

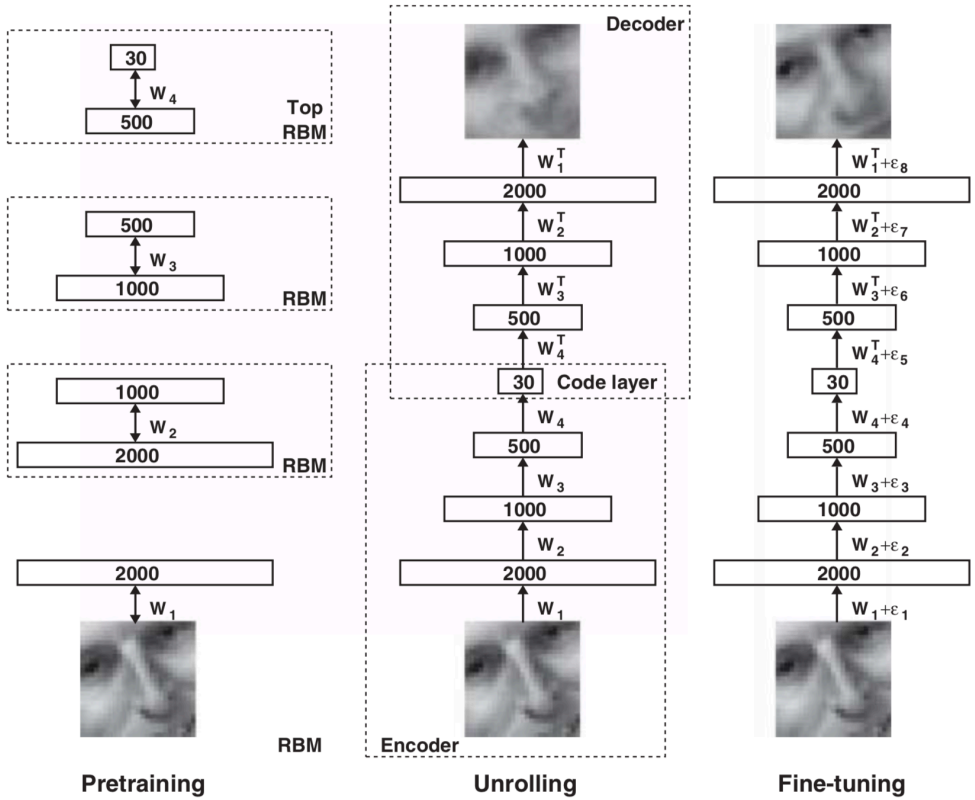
  but minimize the difference of two KL-divergences $\mathrm{KL}(q \mid p) - \mathrm{KL}(p_k \mid p)$

  q is the empirical distribution and p_k is the distribution of the visible layer after k steps of the Markov chain

Autoencoder

- autoencoder learns to compress data from the input layer into a short code and then uncompress it into something closely matches the original data





- dimension reduction by learning how to ignore noise → similar to RG
  use for image recognition or discover interesting structure about the data

- how to learn: pretraining RBM → unrolling → fine-tuning

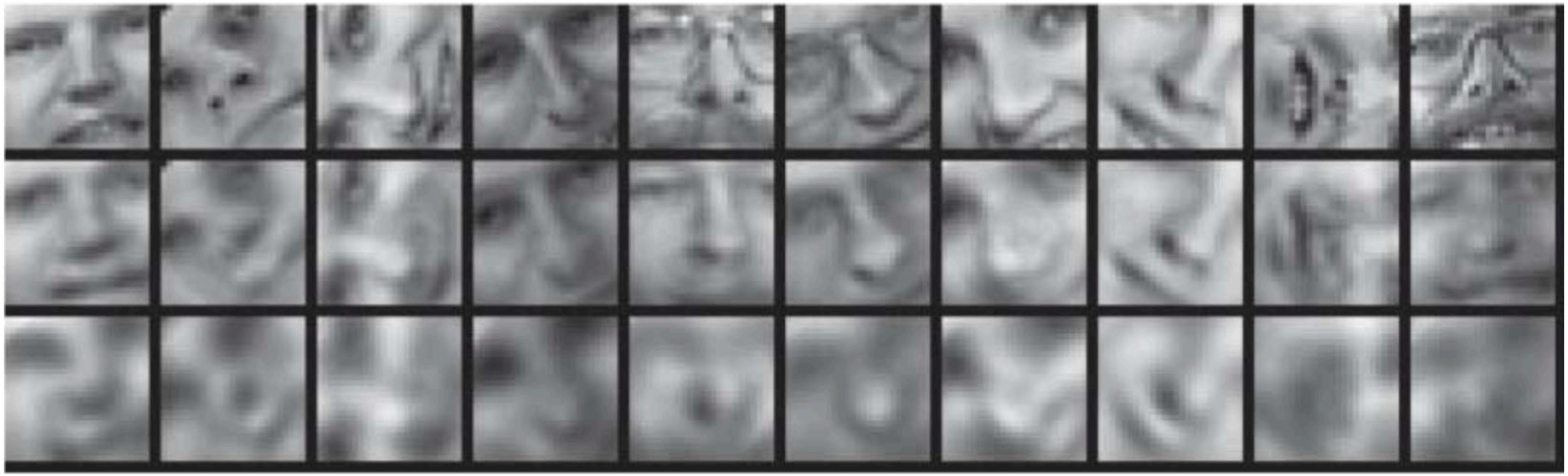
Example

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

504 28 JULY 2006 VOL 313 SCIENCE www.sciencemag.org



sample from test data set

reconstruction by autoencoder

reconstruction by PCA (Principal Components Analysis)

Applications in physics

- many-body quantum matter

  representation theory for neural-network quantum states

  classifying many-body quantum phases

  variational ansatz for many-body simulations

  Monte Carlo update proposals

  tensor networks for machine learning

- quantum computing

  quantum state tomography

  controlling and preparing qubits

  error correction

- miscellaneous

  material and chemistry discoveries

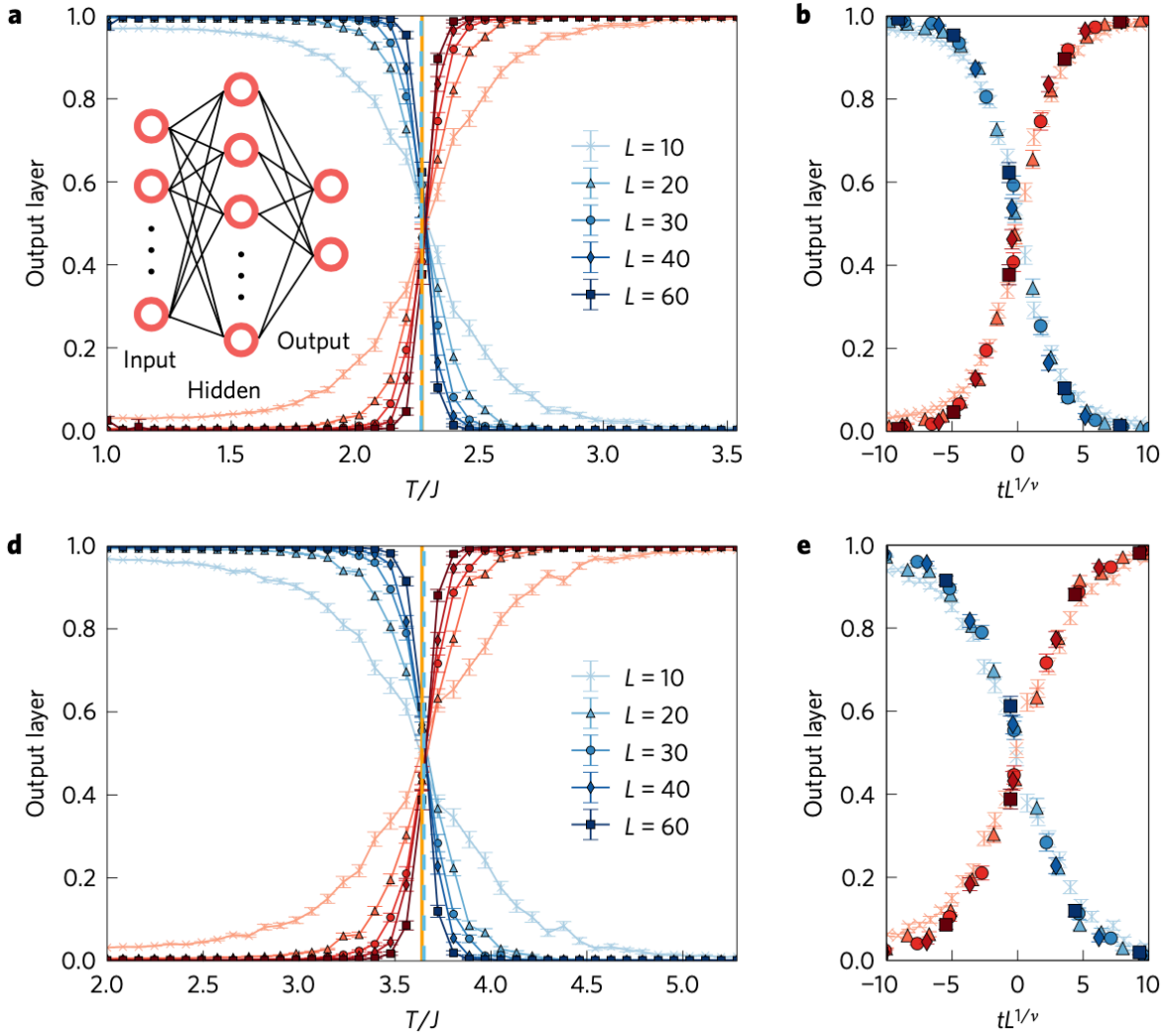
  particle physics and cosmology

  …

Machine learning phases of matter

- train the network with low-T and high-T typical configurations

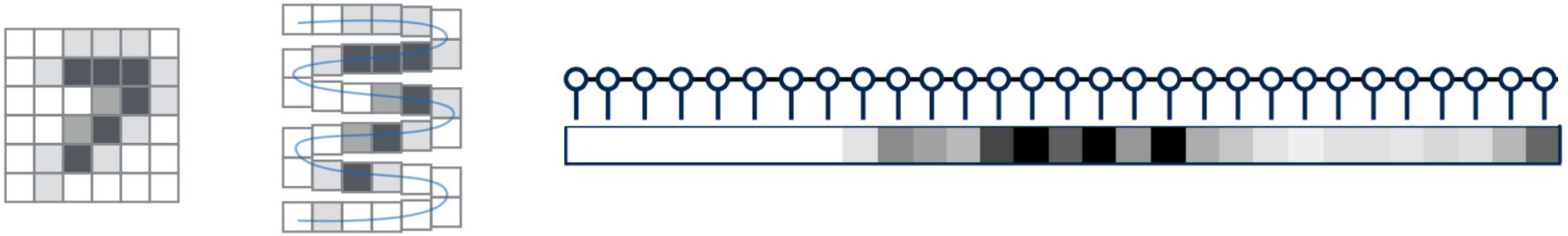
  the trained neural network can be used to detect the phase transition



J. Carrasquilla and R. G. Melko, Nature Physics 13, 431–434 (2017)

Supervised Learning with Tensor Networks

- view MNIST data as MPS



- apply MPS algorithms for supervised learning

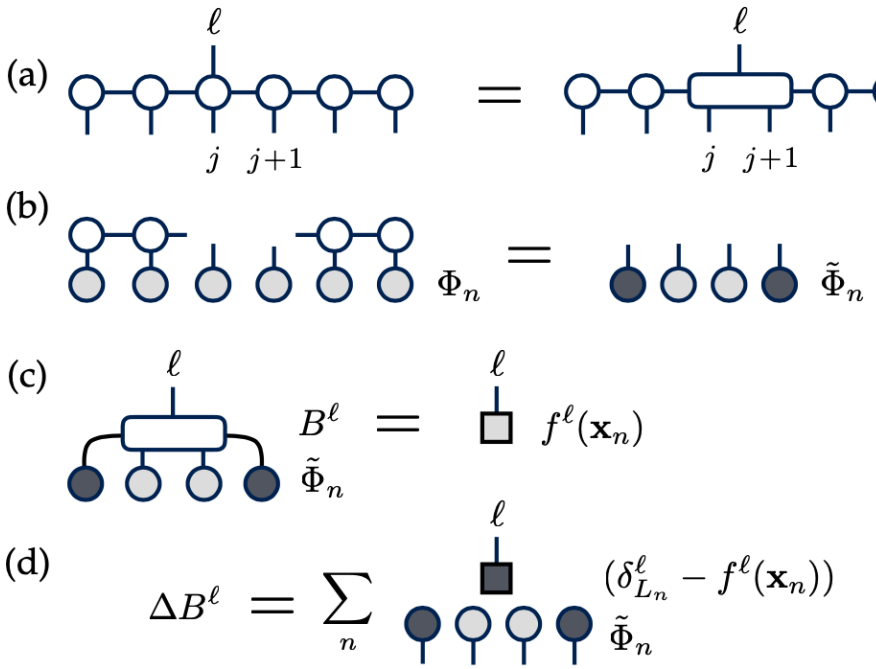


FIG. 6. Steps leading to computing the gradient of the bond tensor B^ℓ at bond j: (a) forming the bond tensor; (b) projecting a training input into the "MPS basis" at bond j; (c) computing the decision function in terms of a projected input; (d) the gradient correction to B^ℓ. The dark shaded circular tensors in step (b) are "effective features" formed from m different linear combinations of many original features.

E.M. Stoudenmire and D.J. Schwab, Advances in Neural Information Processing Systems 29, 4799 (2016), arXiv:1605.05775