

Restricted Boltzmann Machines

Patrick Huembeli

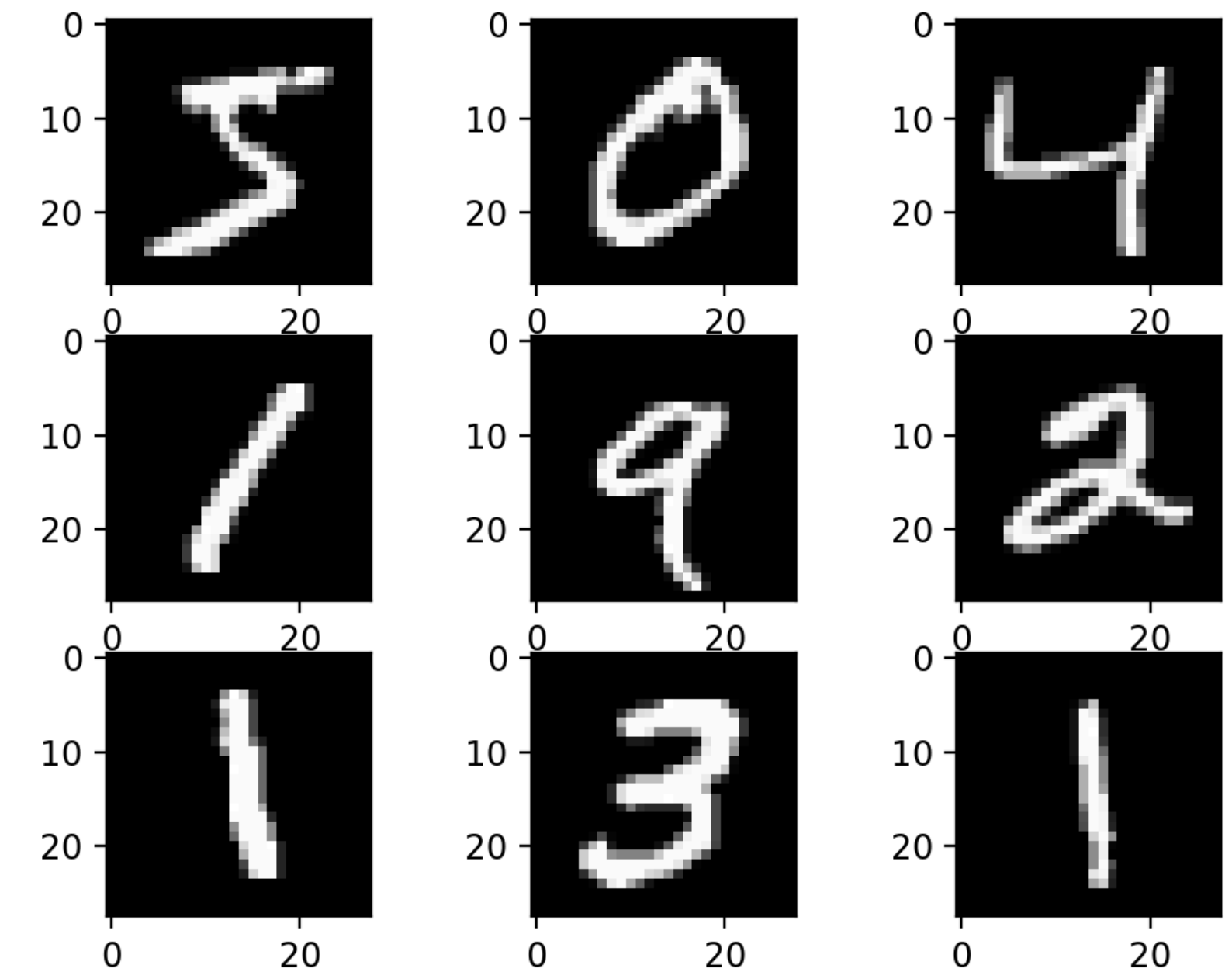
Recap: Hopfield Networks

- Simple energy based model
- Input with N pixels / spins (e.g. MNIST 28x28)

- Binarize input $x_i \in \{-1, +1\}$

- Spin Hamiltonian
$$E(\mathbf{x}) = \sum_{i,j} w_{i,j} x_i x_j + \sum_i b_i x_i$$

- To store an image we need to find the parameters $w_{i,j}$ and b_i that makes an image minimum energy



The role of Temperature

- So far Hopfield networks were at Zero temperature: No stochasticity
- Update rule only accepts a single spin update if it lowers energy:
 - Update $x_i \rightarrow -x_i$ is accepted if $E(-x_i) < E(x_i)$
- We can go to non-zero Temperature via the Boltzmann distribution

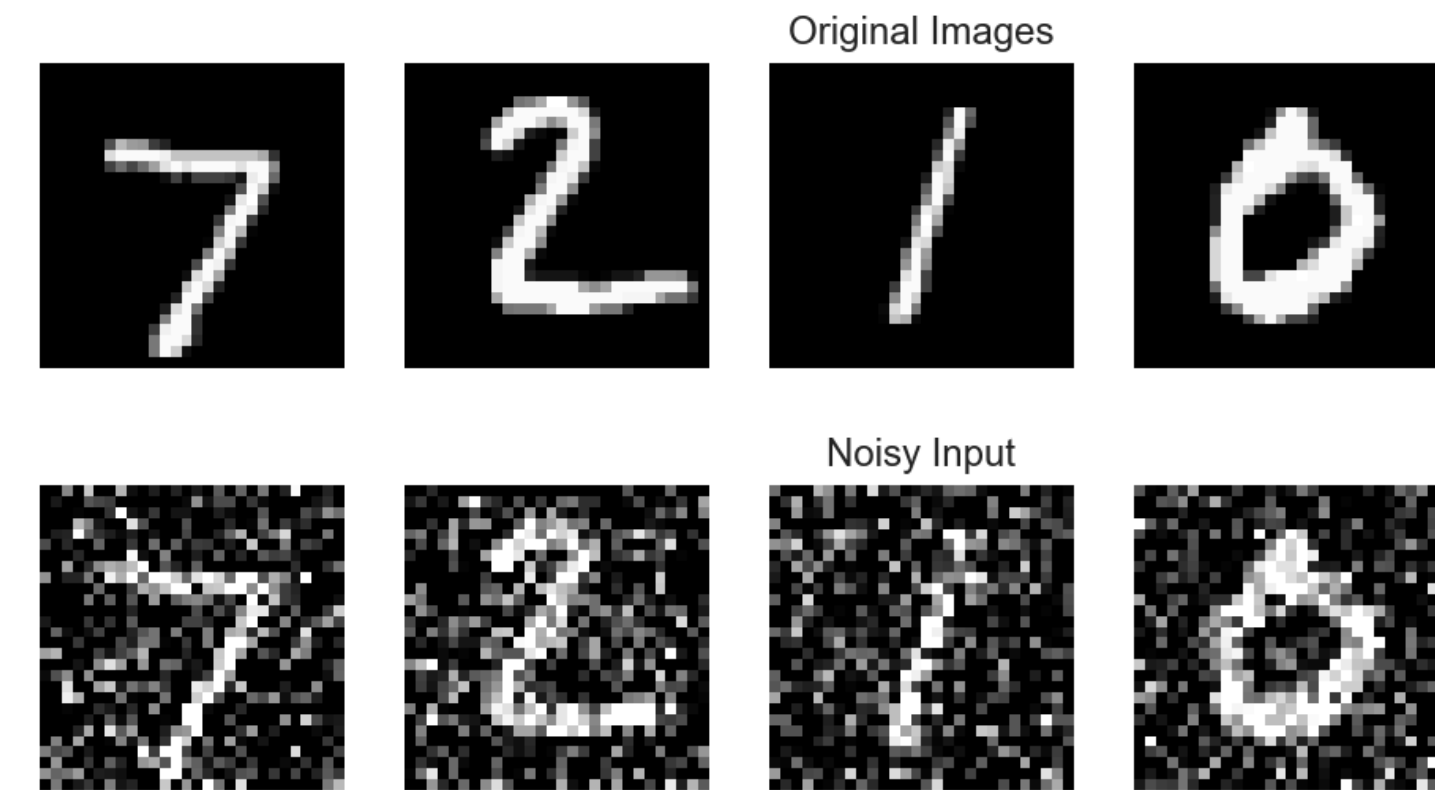
$$p(x) = \frac{e^{-\frac{1}{T}E(x)}}{Z}$$

- Update $x_i \rightarrow -x_i$ is accepted if $E(-x_i) < E(x_i)$

or if

$$\frac{e^{-\frac{1}{T}(E(x_i) - E(-x_i))}}{Z} > R$$

R is randomly drawn from [0, 1]



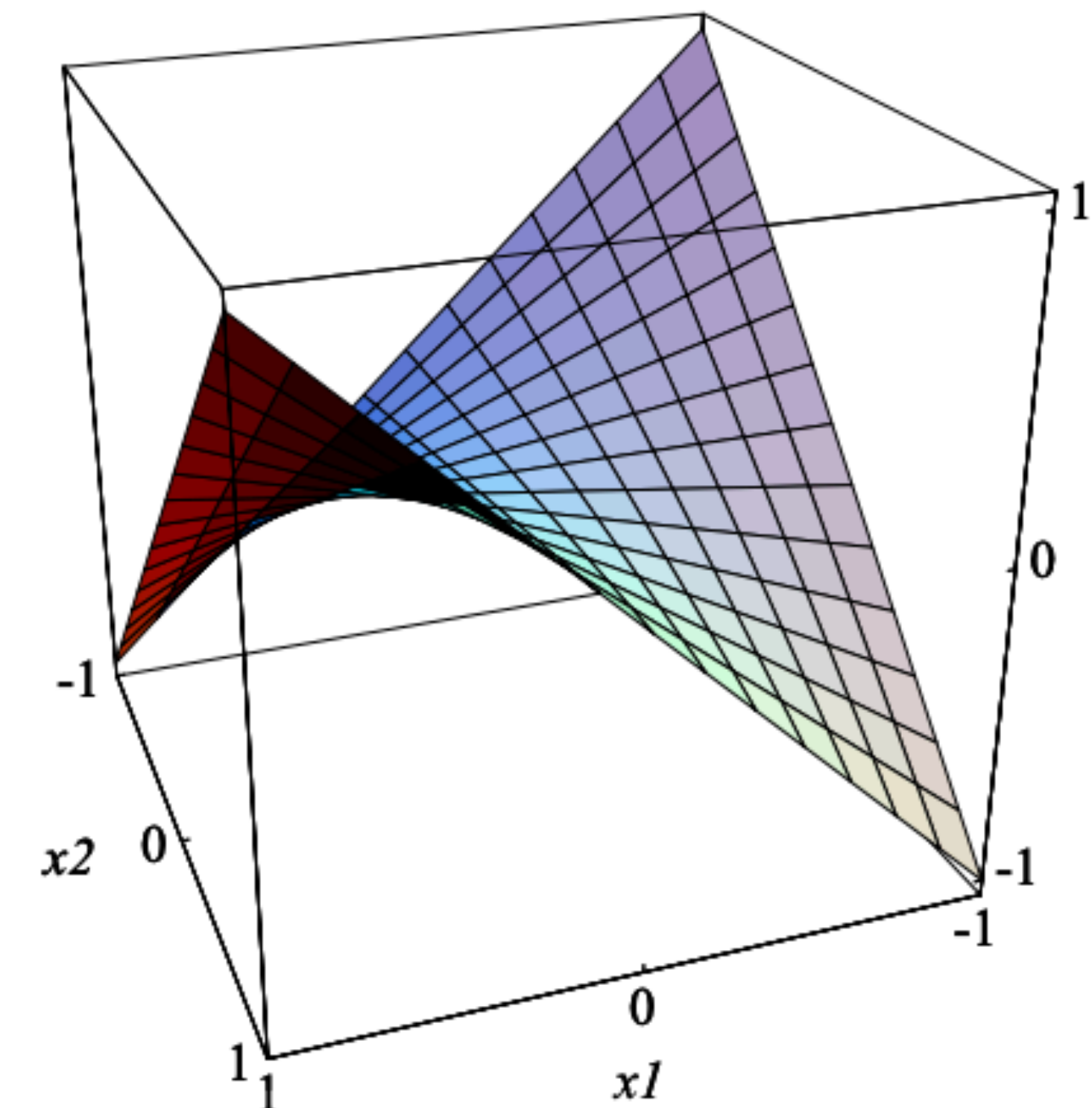
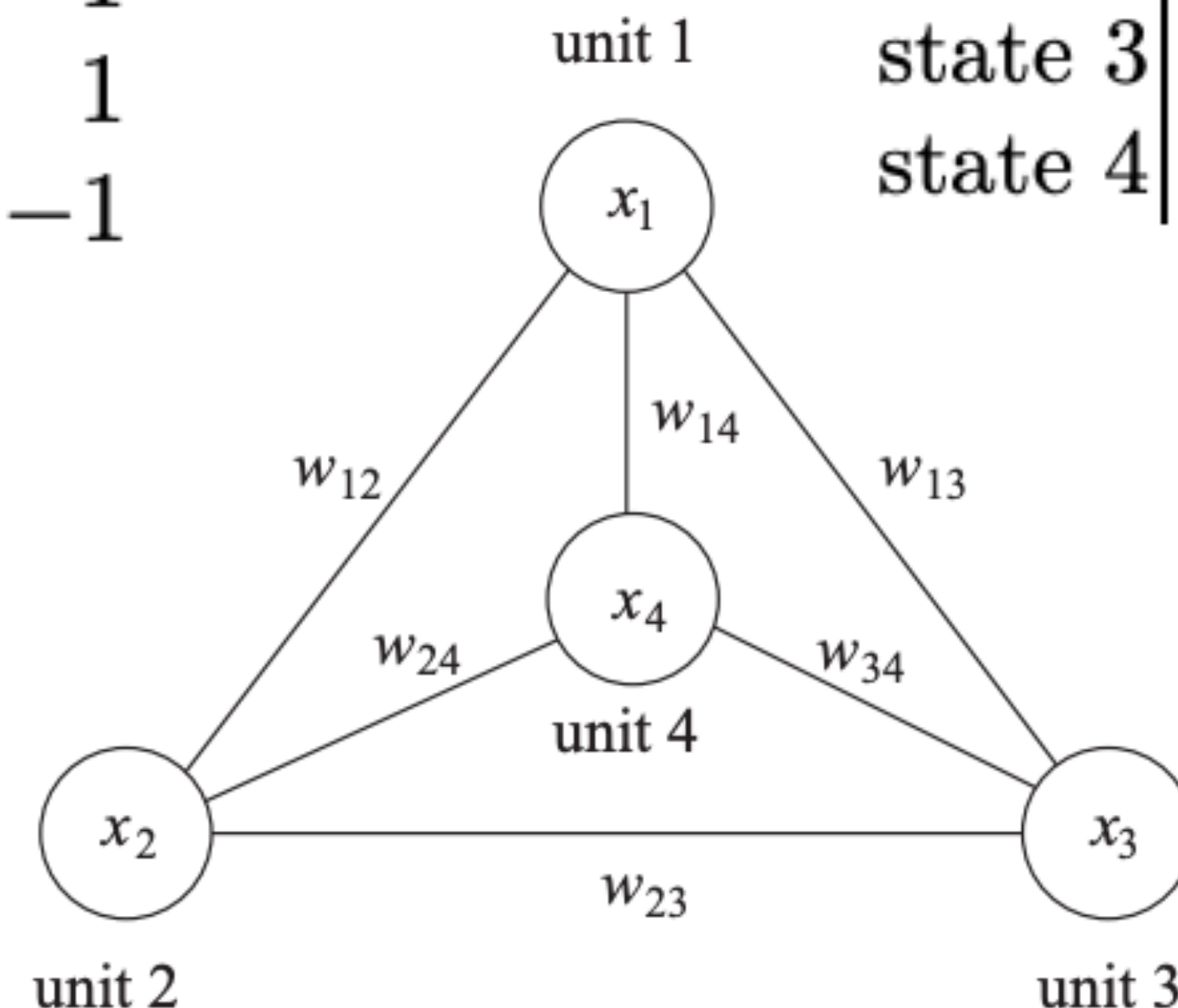
Show Sampling Graphic

XOR problem and hidden nodes

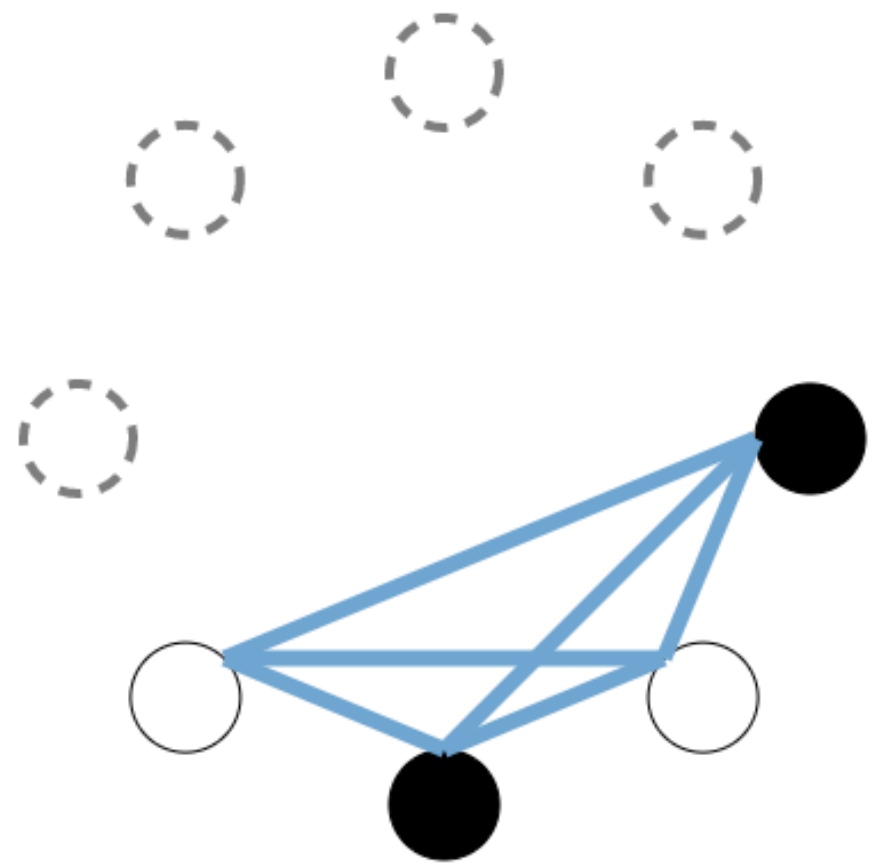
- The Hopfield network can't learn the truth table of an XOR gate.
- If we add an additional spin it can

unit	1	2	3
state 1	-1	-1	-1
state 2	1	-1	1
state 3	-1	1	1
state 4	1	1	-1

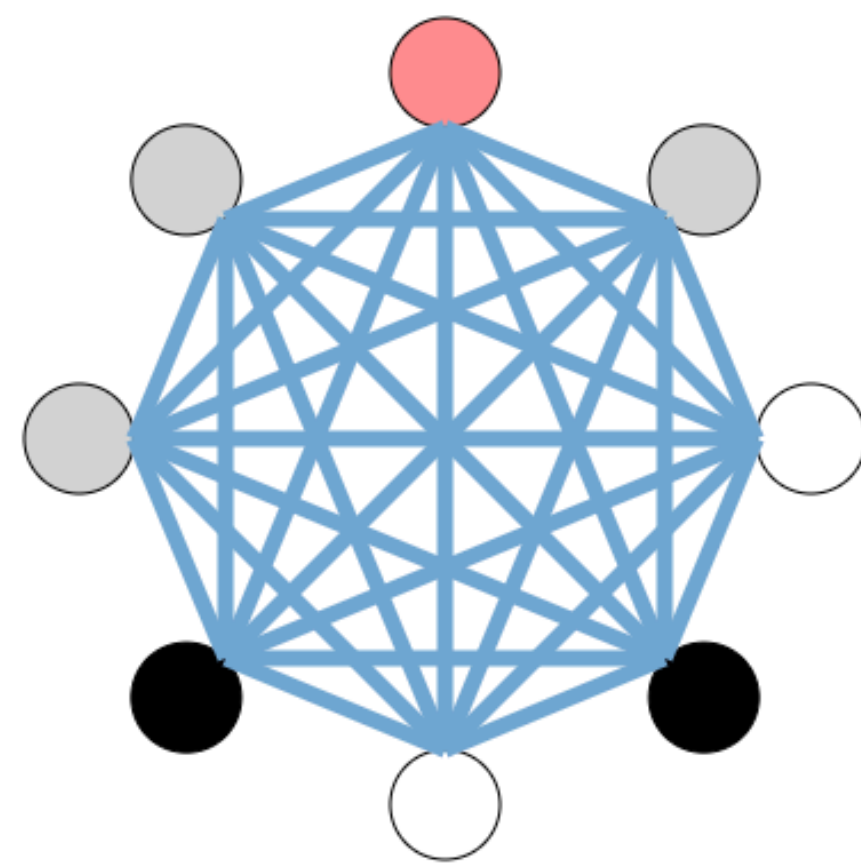
unit	1	2	3	4
state 1	-1	-1	-1	1
state 2	1	-1	1	1
state 3	-1	1	1	1
state 4	1	1	-1	-1



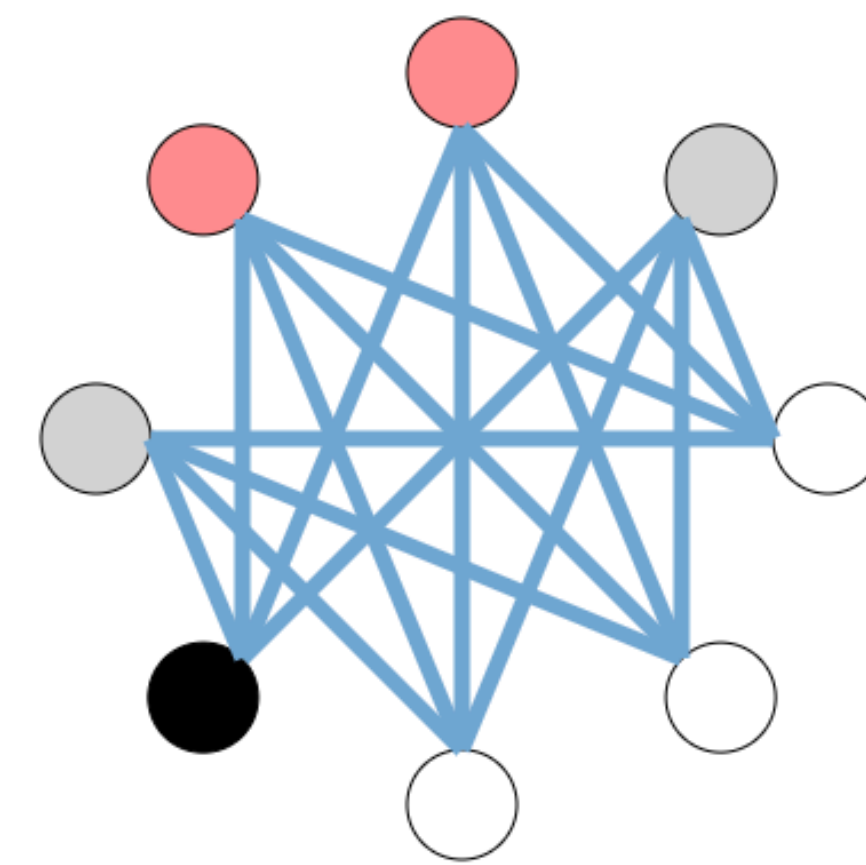
Homework: Convince yourself that this is true!
<https://page.mi.fu-berlin.de/rojas/neural/chapter/K13.pdf>



Hopfield network: The number of nodes is equal to the size of the input data. There are no hidden nodes (dashed) contributing to the energy, which limits the expressive power of this model. You can hover with the mouse over the nodes and the connections for extra information and clicking on the nodes changes their values



Boltzmann machine: The Boltzmann machine network is fully connected. The visible nodes (black or white) are clamped to the input data and the hidden nodes (red & grey) are free parameters. We keep this color scheme throughout this article to distinguish between hidden and visible nodes. Boltzmann machines are a powerful model, but challenging to train.



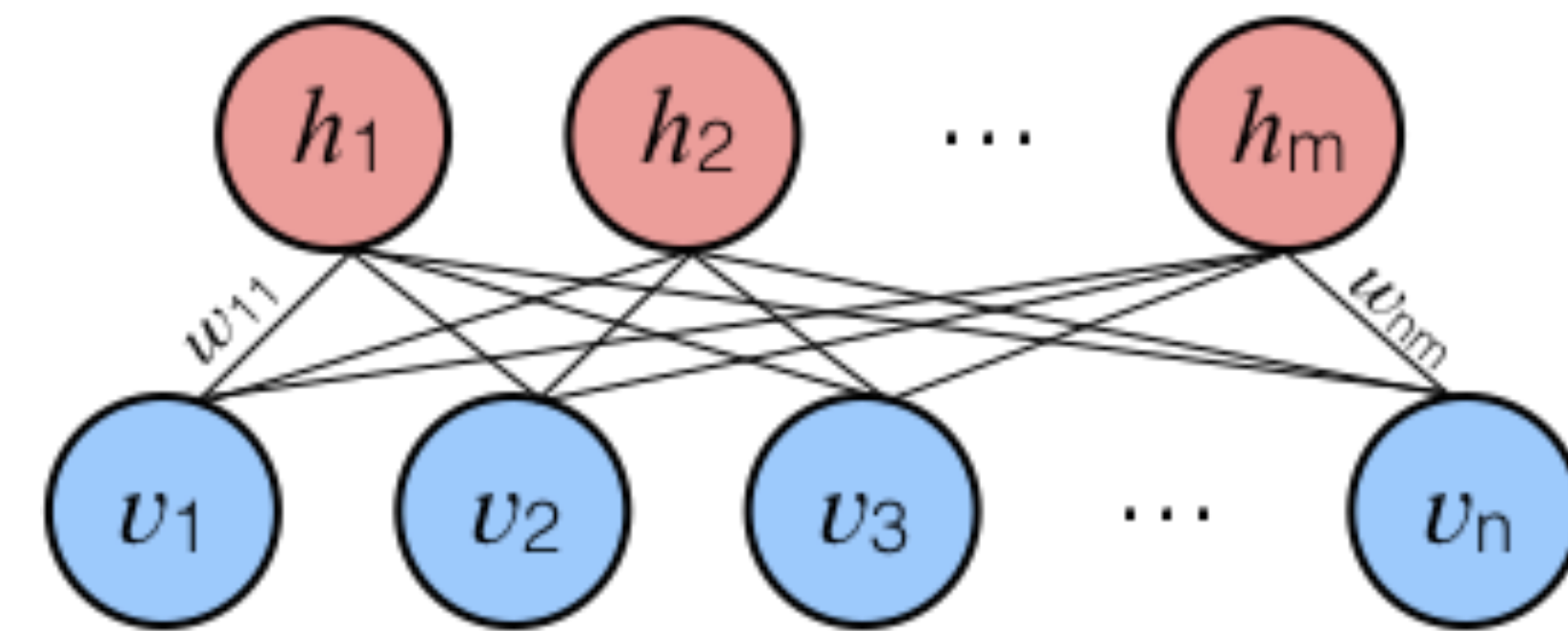
Restricted Boltzmann machine: RBMs use hidden and visible nodes, but connections are not allowed within the same layer, i.e., the network is bipartite. This restriction greatly simplifies training.

Build an RBM

<https://medium.com/@MeTroFuN/python-mxnet-tutorial-1-restricted-boltzmann-machines-using-ndarray-f77578648ecf>

Ingredients:

- Bipartite graph: hidden and visible
- We separate now $\mathbf{x} = (\mathbf{v}, \mathbf{h})$
- And the data corresponds only to \mathbf{v}
- Binary units $\{0,1\}$ or $\{-1,1\}$
- Probabilistic model
 - Normal Neural Networks are deterministic
 - We want to find parameters of model that make training data most likely
- Maximize $p(\mathbf{v})$ if \mathbf{v} is data that we want to learn



Parametrise weights and biases with:

$$\lambda = (\mathbf{W}, \mathbf{b}, \mathbf{c})$$

Build an RBM

$$E_{\lambda}(\mathbf{v}, \mathbf{h}) = - \sum_{i,j} w_{ij} v_i h_j - \sum_i b_i v_i - \sum_j c_j h_j$$

Boltzmann distribution

$$p_{\lambda}(\mathbf{v}, \mathbf{h}) = e^{-E_{\lambda}(\mathbf{v}, \mathbf{h})} / Z$$

Partition Function

$$Z = \sum_{\mathbf{v}, \mathbf{h}} p_{\lambda}(\mathbf{v}, \mathbf{h})$$

$$\rightarrow p_{\lambda}(\mathbf{v}) = \sum_{\mathbf{h}} p_{\lambda}(\mathbf{v}, \mathbf{h}) = e^{-\mathcal{E}(\mathbf{v})} / Z$$

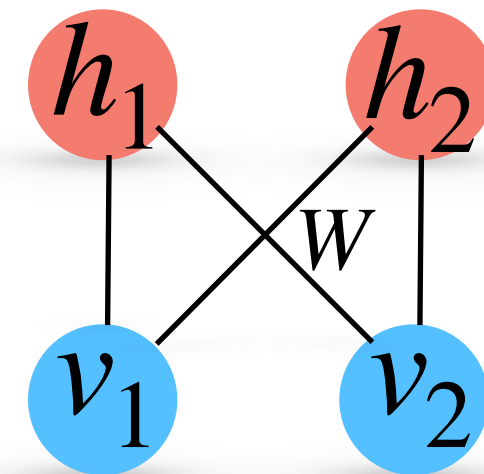
$$\mathcal{E}(\mathbf{v}) = - \sum_i b_i v_i - \sum_j \text{softplus}(c_j + \sum_i w_{i,j} v_i)$$

$$\text{softplus}(x) = \ln(1 + \exp(x))$$

$$\lambda = (\mathbf{W}, \mathbf{b}, \mathbf{c})$$

Min. Energy gives max. Probability !!!

Example:



$$w_{i,j} = 1, \forall i,j$$

$$E = -v_1 h_1 - v_1 h_2 - v_2 h_1 - v_2 h_2$$

$$b_i = c_i = 0$$

Which configuration is most likely??

Homework:











**Calculate Probabilities for this case for all configurations.
To do so you will have to calculate the partition function
and the energies for all possible configurations.**

Do the same with $w_{i,j} = -1, \forall i,j$

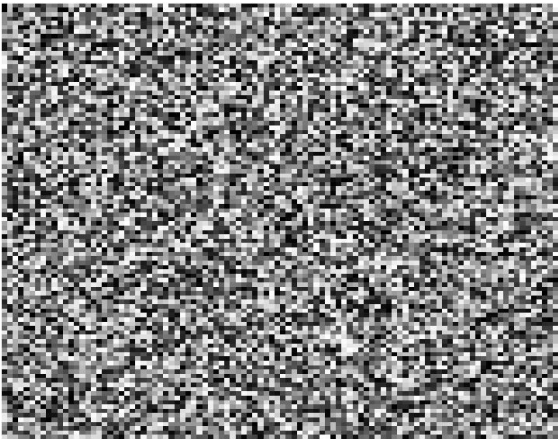
Why is this useful?

- RBM can learn probability distributions $p_{\lambda}(\mathbf{v})$ by finding the parameters that:


$\mathbf{v} =$

Num: 0	Num: 1	Num: 2	Num: 3	Num: 4
				
Num: 5	Num: 6	Num: 7	Num: 8	Num: 9
				

Make this configuration very likely



Make this configuration very unlikely

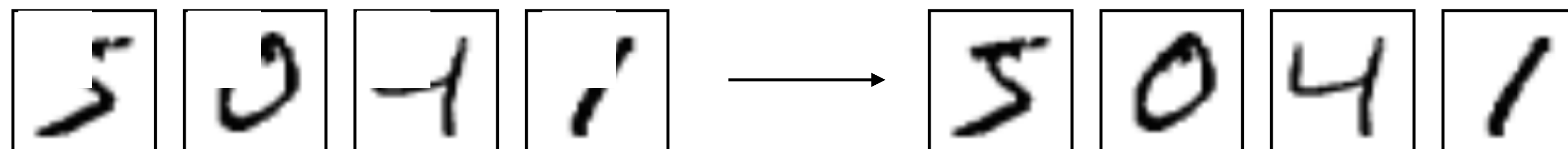


Make this configuration quite likely

- Learning is unsupervised, no labels needed

Why is this useful?

- We can sample from the learned distribution via Gibbs sampling
- Get new configurations
- Generate new data
- Restore 'defect' data (Homework)



1st Summary

- RBMs are probabilistic models
- We here only look at binary RBMs, which means all nodes are either 0 or 1
- We define energy and adjust parameters such that the configurations of our data have the smallest energy
 - Which is the highest probability
 - This is inspired by statistical physics (e.g. classical Spins)
 - Preferred configurations have lowest energy
- Think of training data as probability distribution.
 - Distribution of pixels that we want to learn

Build an RBM

Probabilities:

$$p_{\lambda}(\mathbf{v}, \mathbf{h}) = p_{\lambda}(\mathbf{v} | \mathbf{h}) p_{\lambda}(\mathbf{h})$$

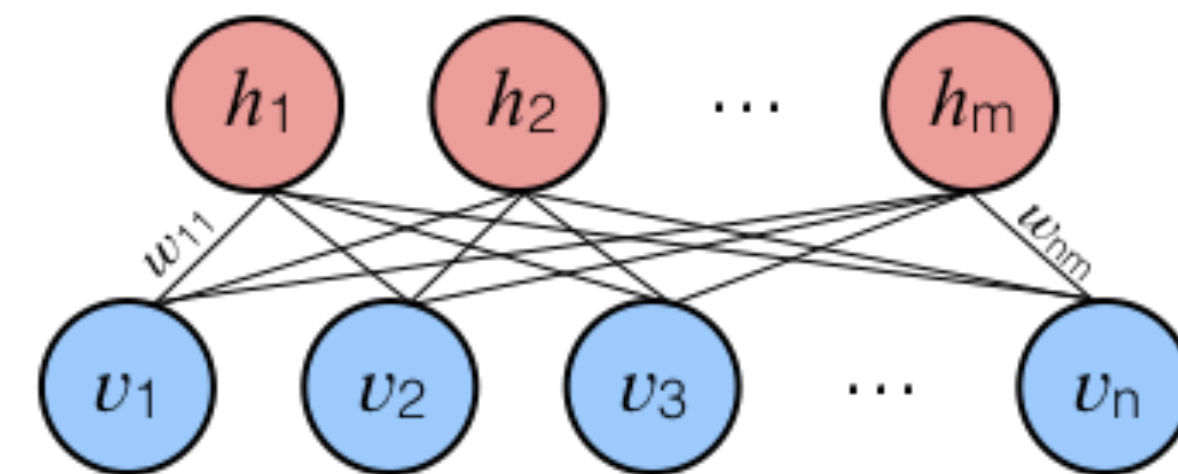
$$p_{\lambda}(\mathbf{h}, \mathbf{v}) = p_{\lambda}(\mathbf{h} | \mathbf{v}) p_{\lambda}(\mathbf{v})$$

In Hopfield network this was simply $p_{\lambda}(\mathbf{x})$
Now we have bipartition in hidden and visible units

Because there are no connections between the units of the same layer, the conditional distributions factorise:

$$p_{\lambda}(\mathbf{v} | \mathbf{h}) = \prod_i p_{\lambda}(v_i | \mathbf{h})$$

$$p_{\lambda}(\mathbf{h} | \mathbf{v}) = \prod_j p_{\lambda}(h_j | \mathbf{v})$$



Build an RBM

- Calculate conditional probabilities

$$p_{\lambda}(\mathbf{v}_i = 1 \mid \mathbf{h}) = \mathcal{S} \left(b_i + \sum_j h_j w_{i,j} \right)$$

$$p_{\lambda}(\mathbf{h}_i = 1 \mid \mathbf{v}) = \mathcal{S} \left(c_j + \sum_i v_j w_{i,j} \right)$$

Because of the restriction that there are no connections within visible or hidden nodes, it is possible to obtain this simple form for the conditional probabilities. And this allows us to sample them relatively easy.

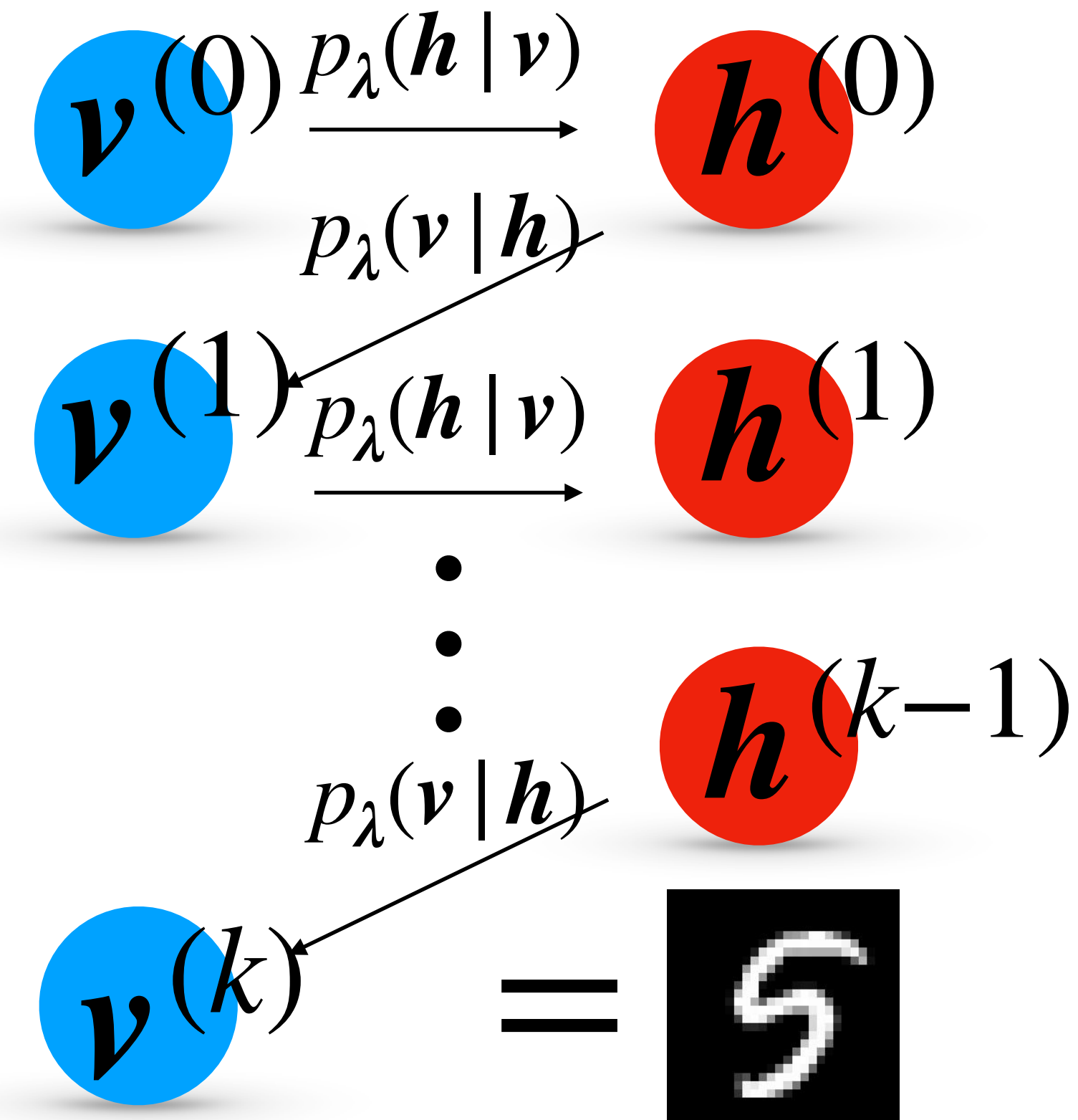
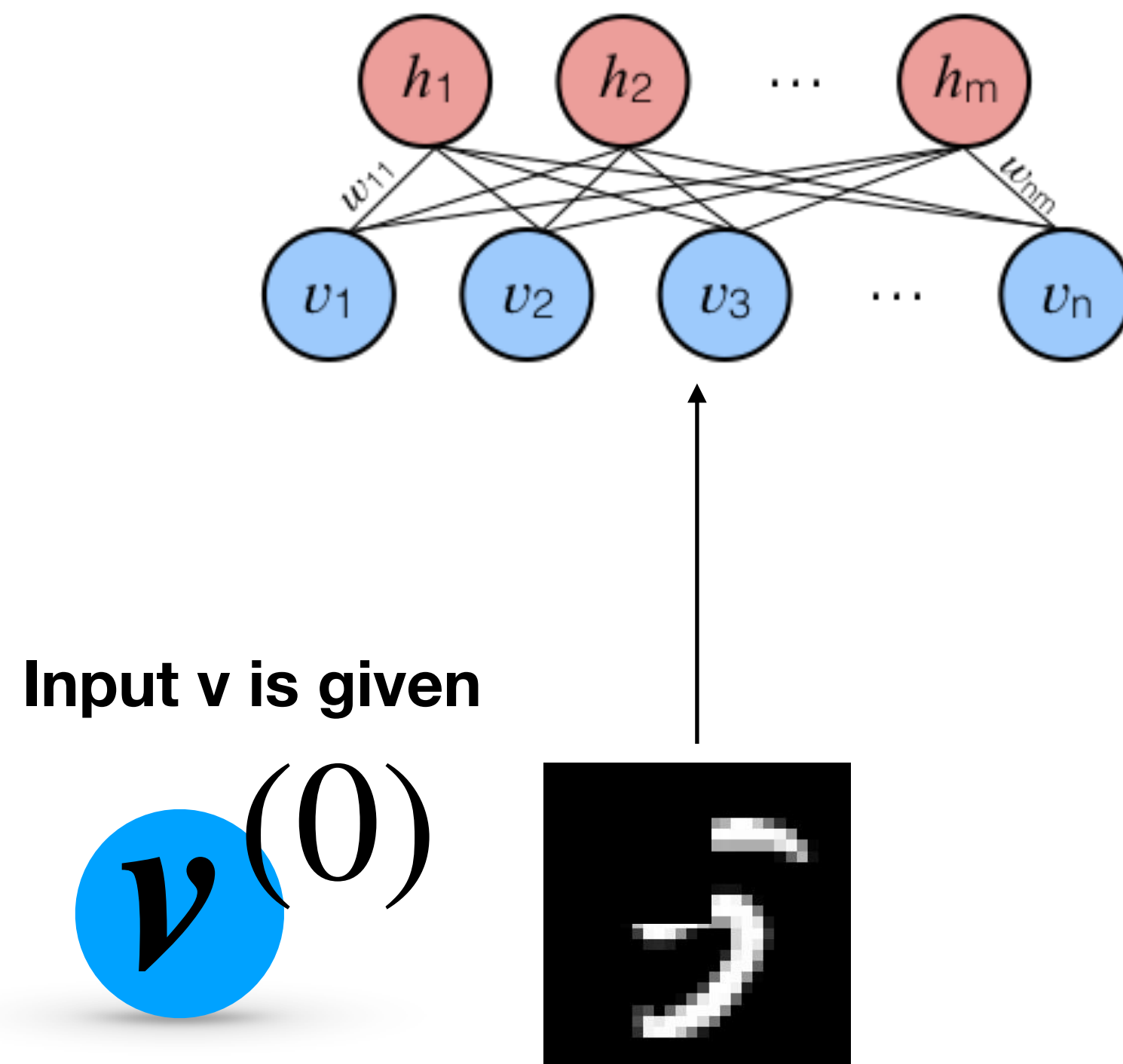
$$\mathcal{S}(x) = \frac{1}{1 + e^{-x}}$$

Derivation:

<http://www.iro.umontreal.ca/~bengioy/ift6266/H14/ftml-sec5.pdf>

Build an RBM: Example

- After RBM is trained on MNIST, the configurations of MNIST are most likely!



Markov chain converges to stationary solution!
Which means low energy solution

Show Equilibration Graphic

Train an RBM

‘Cost function’:

$$C_{\lambda} = D_{KL}(q || p_{\lambda}) = \sum_{\mathbf{v}} q(\mathbf{v}) \log \left(\frac{q(\mathbf{v})}{p_{\lambda}(\mathbf{v})} \right)$$

q

is the data distribution, for example MNIST. How are the pixels normally arranged in a MNIST data set? What is the probability $q(\mathbf{v})$ of a configuration \mathbf{v} in your data set?

p_{λ}

is the distribution learned by the RBM. The parameters λ determine the probability of a certain configuration \mathbf{v} .

Train an RBM

- Minimize Kullback-Leibler (KL) divergence
- To calculate gradient of KL-Divergence we need to calculate expectation values over the model distribution.

Problem!

$$\nabla_{\lambda} C_{\lambda} \approx = \langle \nabla_{\lambda} \mathcal{E}_{\lambda}(\mathbf{v}) \rangle_{\mathcal{D}} - \langle \nabla_{\lambda} \mathcal{E}_{\lambda}(\mathbf{v}) \rangle_{p_{\lambda}}$$

$$p_{\lambda}(\mathbf{v}) = e^{-\mathcal{E}(\mathbf{v})/Z}$$

- Partition function Z generally not accessible
- Derivation from : https://qucumber.readthedocs.io/en/stable/static/RBM_tutorial.pdf

Train RBM

- Derivation on https://github.com/PatrickHuembeli/QML-Course-UPC-2018/blob/master/RBM_Slides_and_Notes/RBM_gradient_derivation.pdf

Alternative Motivation and Recap Hopfield Network

- Hopfield Network was trained with Hebbian rule
 - For data set with N vectors $\mathcal{D} = \{\mathbf{x}^{(k)}\}_{k=1}^N$

$$w_{i,j} = -\frac{1}{N} \sum_{k=1}^N x_i^{(k)} x_j^{(k)} = \langle x_i x_j \rangle_{\mathcal{D}}$$

- If vectors $\mathbf{x}^{(k)}$ are not perfectly orthogonal, spurious minima appear
- Spurious minima: Energy minimas that don't correspond to a training vector $\mathbf{x}^{(k)}$

Motivation and Recap

Hopfield Network

- We can unlearn spurious minima
- Sample m configurations $\{\tilde{\mathbf{x}}^{(l)}\}_{l=1}^m$ from model by initialising it randomly and let it equilibrate.
- This set contains now configurations coming from spurious minima and from “good” minima.
- Add a correction to weights:

$$w_{i,j} = \langle x_i x_j \rangle_{\mathcal{D}} - \varepsilon \frac{1}{m} \sum_{l=1}^m \tilde{x}_i^{(l)} \tilde{x}_j^{(l)} = \langle x_i x_j \rangle_{\mathcal{D}} - \varepsilon \langle x_i x_j \rangle_{\text{model}}$$

Unlearning naturally appears from gradient

- The gradient of the RBM reads:

$$\nabla_{\lambda} C_{\lambda} \approx = \langle \nabla_{\lambda} \mathcal{E}_{\lambda}(\boldsymbol{v}) \rangle_{\mathcal{D}} - \langle \nabla_{\lambda} \mathcal{E}_{\lambda}(\boldsymbol{v}) \rangle_{p_{\lambda}}$$

- For specific weights $\nabla_{w_{i,j}} C_{\lambda} \approx = \langle v_i h_j \rangle_{\mathcal{D}} - \langle v_i h_j \rangle_{p_{\lambda}}$
- To update the weights we use gradient descent:

$$w_{i,j} \leftarrow w_{i,j} - \eta \nabla_{w_{i,j}} C_{\lambda}$$

- Therefore a gradient descent step moves the weights by a small amount η in the direction of the Hebbian weight with unlearning

Reminder: Hebbian rule with unlearning

$$w_{i,j} = \langle x_i x_j \rangle_{\mathcal{D}} - \epsilon \langle x_i x_j \rangle_{\text{model}}$$

Show Figure Hebbian Learning

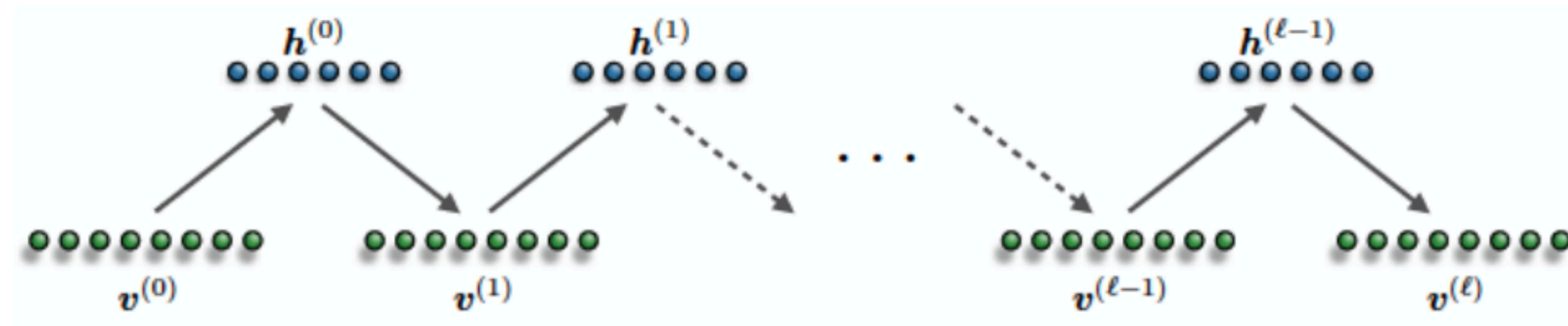
Train RBM

- The problem is we cannot calculate $p_{\lambda}(\mathbf{v})$
- Because to calculate $Z = \sum_{\mathbf{v}, \mathbf{h}} p(\mathbf{v}, \mathbf{h})$ sum grows exponentially
- Approximate expectation value with estimator sampled from model using Gibbs sampling

$$\langle \nabla_{\lambda} \mathcal{E}_{\lambda}(\mathbf{v}) \rangle_{p_{\lambda}} = \sum_{\mathbf{v}} p_{\lambda}(\mathbf{v}) \mathcal{E}_{\lambda}(\mathbf{v}) \approx \frac{1}{M} \sum_{\mathbf{v}^{(l)} \in \mathcal{M}} \mathcal{E}_{\lambda}(\mathbf{v}^{(l)})$$

Gibbs sampling?

- Ideally Markov chain converges to stationary solution



- If we sample back and forth, we approximate the distribution of the RBM
- Initial vector is a training sample (Homework)
- Empirics show, that one Gibbs step is enough

Contrastive divergence

- Instead of taking the model distribution $p_\lambda(\mathbf{v})$

- Estimate expectation value with M samples.

$$\langle \nabla_\lambda \mathcal{E}_\lambda(\mathbf{v}) \rangle_{p_\lambda} = \sum_{\mathbf{v}} p_\lambda(\mathbf{v}) \nabla_\lambda \mathcal{E}_\lambda(\mathbf{v}) \approx \frac{1}{M} \sum_{\mathbf{v}^{(k)} \in \mathcal{M}} \nabla_\lambda \mathcal{E}_\lambda(\mathbf{v}^{(k)})$$

- The number of Gibbs steps for each sample is k
- The more steps we take the closer we converge to the actual model distribution, i.e. equilibrium.
- But it also takes more time
- Contrastive divergence is often referred to as CD- k , where k gives the number of steps

Calculate gradients explicitly

$$\frac{\partial \mathcal{E}_\lambda(\mathbf{v})}{\partial w_{i,j}} \approx -v_i h_i$$

$$\frac{\partial \mathcal{E}_\lambda(\mathbf{v})}{\partial c_j} \approx -h_j$$

$$\frac{\partial \mathcal{E}_\lambda(\mathbf{v})}{\partial b_i} = -v_i$$

Exact gradients: In code for home work we need this exact definition:

Simplified version is often sufficient, because we will average over many samples in the contrastive divergence

$$\frac{\partial \mathcal{E}_\lambda(\mathbf{v})}{\partial w_{i,j}} = -v_i \cdot p_\lambda(h_j = 1 | \mathbf{v})$$

$$\frac{\partial \mathcal{E}_\lambda(\mathbf{v})}{\partial c_j} = -p_\lambda(h_j = 1 | \mathbf{v})$$

$$\frac{\partial \mathcal{E}_\lambda(\mathbf{v})}{\partial b_i} = -v_i$$

Check as Homework!

Parameter update

- Like in stochastic gradient descent $\lambda \leftarrow \lambda - \eta \nabla_{\lambda} C_{\lambda}$
- Reminder: $\nabla_{\lambda} C_{\lambda} \approx \langle \nabla_{\lambda} \mathcal{E}_{\lambda}(\mathbf{v}) \rangle_{\mathcal{D}} - \langle \nabla_{\lambda} \mathcal{E}_{\lambda}(\mathbf{v}) \rangle_{p_{\lambda}}$
- E.g. weight update

$$w_{i,j} \leftarrow w_{i,j} - \eta \left\langle \frac{\partial \mathcal{E}_{\lambda}(\mathbf{v})}{\partial w_{i,j}} \right\rangle_{\mathcal{D}} + \eta \left\langle \frac{\partial \mathcal{E}_{\lambda}(\mathbf{v})}{\partial w_{i,j}} \right\rangle_{p_{\lambda}}$$

$$w_{i,j} \leftarrow w_{i,j} + \eta \langle v_i \cdot h_j \rangle_{\mathcal{D}} - \eta \langle v_i \cdot h_j \rangle_{p_{\lambda}}$$

$$w_{i,j} \leftarrow w_{i,j} + \eta \frac{1}{|\mathcal{D}|} \sum_{v_i \in \mathcal{D}} v_i \cdot h_j - \eta \frac{1}{|\mathcal{M}|} \sum_{v_i^{(k)} \in \mathcal{M}} v_i \cdot h_j$$

**Sum over batch from
data**

**Sum over batch of
Gibbs samples**

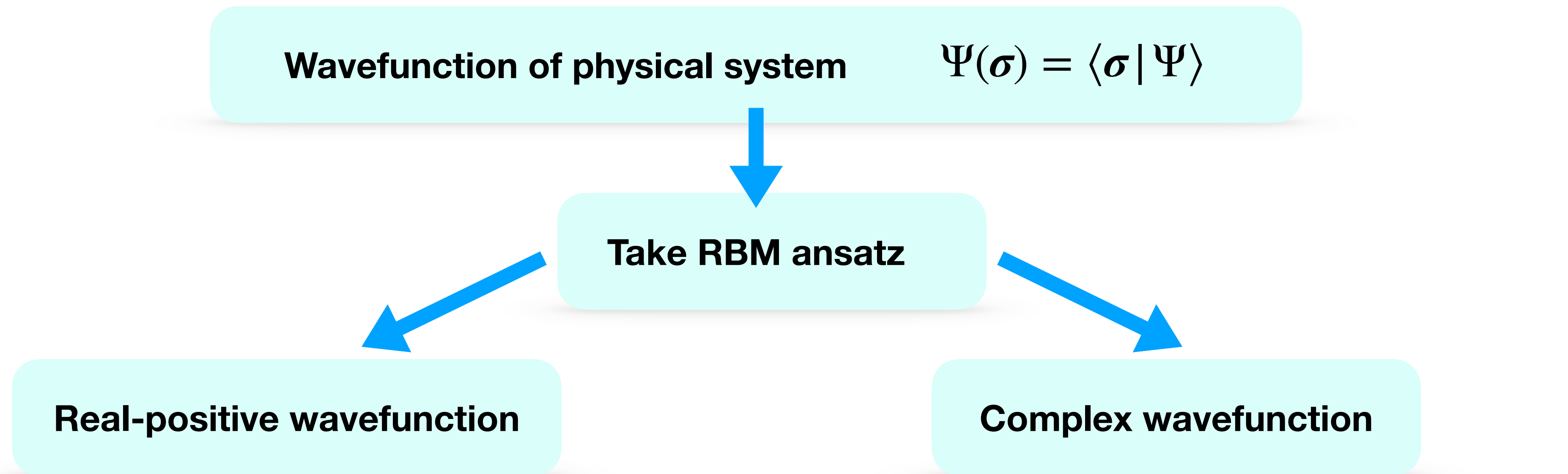
Restricted Boltzmann Machines in Physics

RBM as ansatz for wavefunctions

$$\Psi = \sum_{\sigma} \Psi(\sigma) |\sigma\rangle \quad P(\sigma) \propto |\Psi(\sigma)|^2 \quad \psi_{\lambda}(\sigma) = \sqrt{\frac{p_{\lambda}(\sigma)}{Z_{\lambda}}}$$
$$\Psi(\sigma) = \langle \sigma | \Psi \rangle$$

- Variational Monte Carlo and minimize e.g. energy (Troyer et al.)
- Learn state from samples / measurements
- Ansatz is efficient.
 - ➡ Grows polynomial in system size

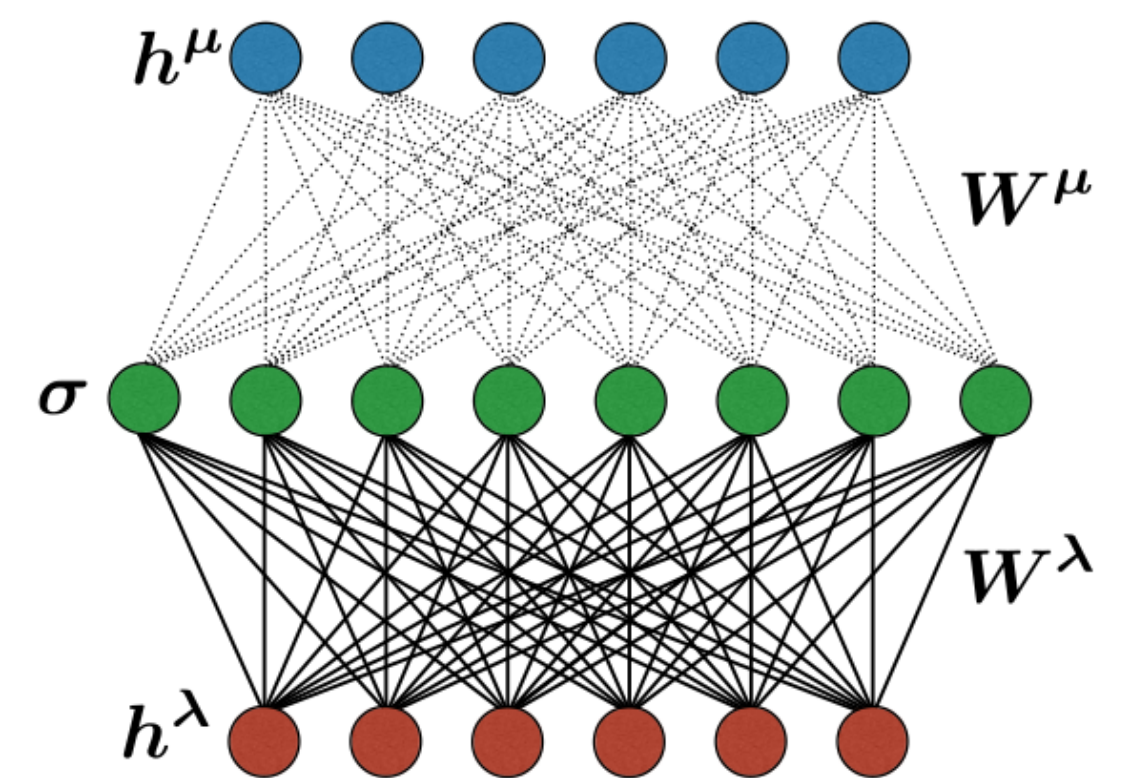
RBM as ansatz for wavefunctions



$$\psi_{\lambda}(\sigma) = \sqrt{\frac{p_{\lambda}(\sigma)}{Z_{\lambda}}}$$

$$\psi_{\lambda,\mu}(\sigma) = \sqrt{\frac{p_{\lambda}(\sigma)}{Z_{\lambda}}} e^{i\phi_{\mu}(\sigma)}$$

$$\phi_{\mu}(\sigma) = \log p_{\mu}(\sigma)$$



State tomography

- For phases we need to change basis.
- Make measurements on system in different basis
 $b = 0, 1, \dots$

$$P_b(\boldsymbol{\sigma}^{[b]}) \propto \left| \Psi(\boldsymbol{\sigma}^{[b]}) \right|^2$$

- Learn this distribution with RBM

$$\psi_{\lambda,\mu}(\boldsymbol{\sigma}^{[b]}) = \sum_{\boldsymbol{\sigma}} U_b(\boldsymbol{\sigma}, \boldsymbol{\sigma}^{[b]}) \psi_{\lambda,\mu}(\boldsymbol{\sigma})$$

- In contrast to positive-real wavefunction, the unitaries will be part of the training.

Results

- RBMs are efficient representation for quantum states
- We can sample from it
- We can calculate any expectation value via estimators
- Open source software for QST

➡ <https://github.com/PIQuIL/QuCumber>

