

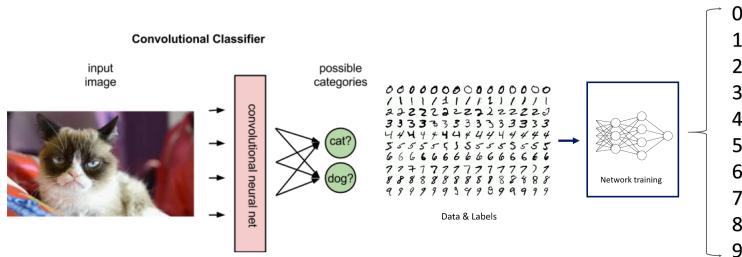
Introduction to Probabilistic Models

F. Noé¹

Deep Learning Classes, FU Berlin 2018

Classification

- Classification problems have discrete output variables (categories).
- Previously, we have used ordinary regression norms, such as $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$. Here, we want to go to a probabilistic description.



Reminder

Minimization of cost functions

Symbols

Meaning	Symbol	Shape
Feature vector	\mathbf{x}	\mathbb{R}^n
Feature matrix	\mathbf{X}	$\mathbb{R}^{N \times n}$
Label vector	\mathbf{y}	\mathbb{R}^I
Label matrix	\mathbf{Y}	$\mathbb{R}^{N \times I}$
Parameters	θ	\mathbb{R}^d
Loss / Cost function	$C(\mathbf{X}, \mathbf{Y}; \theta)$	\mathbb{R}

Cost function C quantifies how well a given model with parameters θ explains the observations \mathbf{X} .

Model fitting

$$\hat{\theta} = \arg \min_{\theta} C(\mathbf{X}, \mathbf{Y}; \theta)$$

Minimizing the *cost* C is equivalent to maximizing the *score* $-C$.

Logistic Regression

Shallow-learning basis of classification problems

- **Input:** Training data (\mathbf{X}, \mathbf{y}) with feature matrix $\mathbf{X} \in \mathbb{R}^{N \times n}$ and one-dimensional binary outputs $y \in \{0, 1\}$ (2 classes).
- **Goal:** Predict $\mathbf{x} \rightarrow \hat{y}$ with minimal error.
- **Idea 1: Perceptron.** Define **linear model** with weights $\mathbf{w} \in \mathbb{R}^n$ and offset $b \in \mathbb{R}$

$$s_i = \mathbf{x}_i^\top \mathbf{w} + b.$$

Map s_i through sign function in order to turn into a classifier:

$$\hat{y}_i = \text{sign}(s_i) = \begin{cases} 1 & s_i \geq 0 \\ 0 & s_i < 0 \end{cases}$$

Logistic Regression

Shallow-learning basis of classification problems

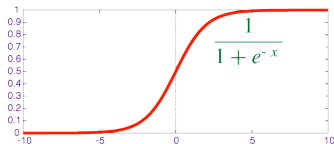
- It is often desirable to have a **soft** classifier with $\hat{y} \in [0, 1] \subset \mathbb{R}$:
 - noisy data \rightarrow not all data points can be unambiguously assigned.
 - enables computation of gradients \rightarrow can use deep structures with backpropagation.
- **Idea 2:** Define **linear model** with weights $\mathbf{w} \in \mathbb{R}^n$ and offset $b \in \mathbb{R}$

$$s_i = \mathbf{x}_i^\top \mathbf{w} + b.$$

Map s_i through **logistic (sigmoid)** function:

$$\hat{y}_i = \sigma(s_i) = \frac{1}{1 + e^{-s_i}}$$

Note: $1 - \sigma(s_i) = \sigma(-s_i)$



- Statistical mechanical system with two states 0 and 1 and **energies** $\varepsilon_0 = 0$ (reference) and ε_1 .
- **Boltzmann weights:**

$$w_0 = e^{-\varepsilon_0} = 1$$

$$w_1 = e^{-\varepsilon_1}$$

- **Probability:**

$$p_1 = \frac{w_1}{w_0 + w_1} = \frac{e^{-\varepsilon_1}}{1 + e^{-\varepsilon_1}} = \frac{1}{e^{\varepsilon_1} + 1} = \sigma(-\varepsilon_1)$$

- Logistic regression is obtained with model $\varepsilon_1 = -\mathbf{x}_i^\top \mathbf{w} - b$.

Logistic Regression

Shallow-learning basis of classification problems

- Probability to belong to category 0 or 1:

$$p(y_i = 1 \mid \mathbf{x}_i, \theta) = \sigma(\mathbf{x}_i^\top \mathbf{w} + b) = \frac{1}{1 + e^{-\mathbf{x}_i^\top \mathbf{w} - b}}$$

$$p(y_i = 0 \mid \mathbf{x}_i, \theta) = 1 - p(y_i = 1 \mid \mathbf{x}_i; \theta)$$

- **Likelihood** of data set $\{\mathbf{x}_i, y_i\}_{i=1 \dots N}$ under the model:

$$p(\{\mathbf{x}_i, y_i\} \mid \theta) = \prod_{i=1}^N \left[\sigma(\mathbf{x}_i^\top \mathbf{w} + b) \right]^{y_i} \left[1 - \sigma(\mathbf{x}_i^\top \mathbf{w} + b) \right]^{1-y_i}$$

Logistic Regression

Shallow-learning basis of classification problems

- **Log-likelihood:**

$$L(\theta) = \sum_{i=1}^N y_i \log \sigma(\mathbf{x}_i^\top \mathbf{w} + b) + (1 - y_i) \log [1 - \sigma(\mathbf{x}_i^\top \mathbf{w} + b)]$$

- **Maximum likelihood estimator:**

$$\hat{\theta} = \arg \max_{\theta} L(\theta) = \arg \min_{\theta} \{-L(\theta)\}$$

- **Cross-entropy:**

$$\begin{aligned} C(\theta) &= -L(\theta) \\ &= - \sum_{i=1}^N y_i \log \sigma(\mathbf{x}_i^\top \mathbf{w} + b) + (1 - y_i) \log [1 - \sigma(\mathbf{x}_i^\top \mathbf{w} + b)] \end{aligned}$$

- As in linear regression, the cross-entropy is often equipped with regularizers.

Logistic Regression

Shallow-learning basis of classification problems

- Minimize cross-entropy, using $\sigma'(x) = \sigma(x)(1 - \sigma(x))$:

$$\begin{aligned}\frac{\partial C}{\partial w_k} &= - \sum_{i=1}^N y_i \frac{\sigma'(\mathbf{x}_i^\top \mathbf{w} + b)}{\sigma(\mathbf{x}_i^\top \mathbf{w} + b)} x_{ik} + (1 - y_i) \frac{-\sigma'(\mathbf{x}_i^\top \mathbf{w} + b)}{1 - \sigma(\mathbf{x}_i^\top \mathbf{w} + b)} x_{ik} \\ &= - \sum_{i=1}^N y_i \left[1 - \sigma(\mathbf{x}_i^\top \mathbf{w} + b) \right] x_{ik} - (1 - y_i) \sigma(\mathbf{x}_i^\top \mathbf{w} + b) x_{ik} \\ &= \sum_{i=1}^N \left[\sigma(\mathbf{x}_i^\top \mathbf{w} + b) - y_i \right] x_{ik}\end{aligned}$$

- Likewise

$$\frac{\partial C}{\partial b} = \sum_{i=1}^N \sigma(\mathbf{x}_i^\top \mathbf{w} + b) - y_i$$

- Using $\theta = (b, w_1, \dots, w_n)$, results in the gradient:

$$\nabla C(\theta) = \sum_{i=1}^N \left[\sigma(\mathbf{x}_i^\top \mathbf{w} + b) - y_i \right] \begin{pmatrix} 1 \\ \mathbf{w} \end{pmatrix}$$

No closed-form solution \rightarrow numerical optimization.

Softmax Regression

Shallow-learning basis of classification problems

- Regression to multiple classes: $\{1, 2, \dots, I\}$.
- **One-hot encoding:** $\mathbf{y}_i \in \mathbb{R}^I$ with:

$$y_{ik} = \begin{cases} 1 & c_i = k \\ 0 & \text{else} \end{cases}.$$

- Probability of \mathbf{x}_i to be in class k : **Softmax function**

$$p(y_{ik} = 1 \mid \mathbf{x}_i; \theta) = S_k(\mathbf{x}_i; \theta) = \frac{e^{\mathbf{x}_i^\top \mathbf{w}_k + b_k}}{\sum_{j=1}^I e^{\mathbf{x}_i^\top \mathbf{w}_j + b_j}}$$

- Corresponds to statistical mechanics model with energies $\varepsilon_j = -\mathbf{x}_i^\top \mathbf{w}_j - b_j$.
- Likelihood:

$$p(\{\mathbf{x}_i, y_i\} \mid \theta) = \prod_{i=1}^N \prod_{k=1}^I [S_k(\mathbf{x}_i; \theta)]^{y_{ik}} [1 - S_k(\mathbf{x}_i; \theta)]^{1-y_{ik}}$$

Softmax Regression

Shallow-learning basis of classification problems

- **Loss function:**

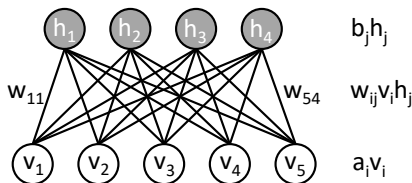
$$C(\theta) = - \sum_{i=1}^N \sum_{k=1}^I y_{ik} \log S_k(\mathbf{x}_i; \theta) + (1 - y_{ik}) \log [1 - S_k(\mathbf{x}_i; \theta)]$$

- For $I = 1$, we recover the cross-entropy for logistic regression.
- For $I = 2$, use that $y_{i1} = 1 - y_{i2}$ and $S_2(\mathbf{x}_i; \theta) = 1 - S_1(\mathbf{x}_i; \theta)$, and obtain:

$$C(\theta) = -2 \sum_{i=1}^N y_{i1} \log S_1(\mathbf{x}_i; \theta) + (1 - y_{i1}) \log [1 - S_1(\mathbf{x}_i; \theta)]$$

- Equivalent with cross-entropy for logistic regression (up to constant factor).
- For a two-class classification problem, it is equivalent to use one output neuron with logistic activation or two output neurons with softmax activation.

Restricted Boltzmann Machine (RBM)



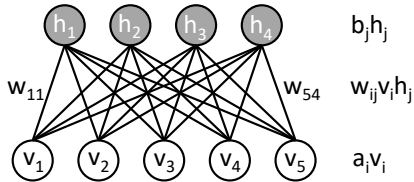
- **Energy-based model.** Visible units $v_i \in \{0,1\}$, hidden units $h_j \in \{0,1\}$.
- Bipartite interaction graph: visible-hidden interactions, but no hidden-hidden or visible-visible.
- Standard **energy function** with biases a_i, b_j and weights w_{ij} :

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}) &= - \sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} w_{ij} v_i h_j \\ &= -\mathbf{a}^\top \mathbf{v} - \mathbf{b}^\top \mathbf{h} - \mathbf{v}^\top \mathbf{W} \mathbf{h}. \end{aligned}$$

Continuous-variable versions and different energy functions exist.

- Each hidden unit can be thought as a representative of a data pattern or feature.
- Closely related to the Hopfield memory model

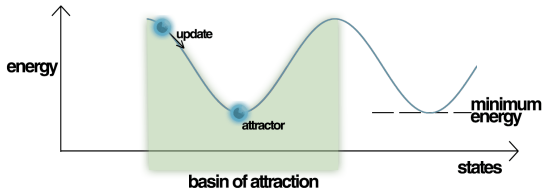
Restricted Boltzmann Machine (RBM)



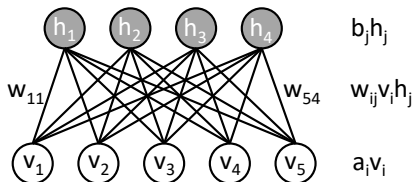
Energy function

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^\top \mathbf{v} - \mathbf{b}^\top \mathbf{h} - \mathbf{v}^\top \mathbf{W} \mathbf{h}.$$

Defines the localations of minima (attractors in Hopfield model, metastable states in RBM)



Restricted Boltzmann Machine



- Probability of state (\mathbf{v}, \mathbf{h}) :

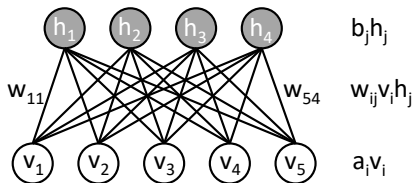
$$p(\mathbf{v}, \mathbf{h}) = Z^{-1} e^{-E(\mathbf{v}, \mathbf{h})}$$

- Partition function Z sums over all visible and hidden states:

$$Z = \sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$$

Restricted Boltzmann Machine (RBM)

What can a RBM represent?



- Define free energy of visible units $E(\mathbf{v}) = -\log p(\mathbf{v})$ and rewrite as:

$$E(\mathbf{v}) = -\sum_i a_i v_i - \sum_j \sum_n \frac{\kappa_j^{(n)}}{n!} \left(\sum_i w_{ij} v_i \right)^n$$

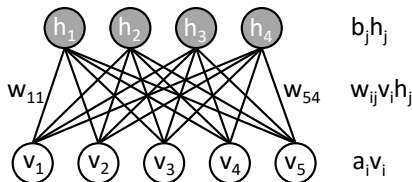
where $\kappa_j^{(n)}$ are the cumulants of the distribution of $e^{b_j h_j}$.

Cumulants are – as moments – a way to characterize probability densities (e.g., as for moments, the first two cumulants are the mean and the variance).

- $E(\mathbf{v})$ includes all orders of interactions between the visible units.
 - Each hidden unit can encode interactions of arbitrarily high order.
 - With sufficiently many hidden units, any probabilist distribution of \mathbf{v} can be encoded (compare to universal representation theorem).

Restricted Boltzmann Machine

Training



- Probability of visible vector \mathbf{v} :

$$p(\mathbf{v}) = Z^{-1} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$$

- Derivative of the log-probability with respect to the weight is:

$$\begin{aligned} \frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} &= \sum_{\mathbf{h}} p(\mathbf{h} | \mathbf{v}) v_i h_j - \sum_{\mathbf{v}', \mathbf{h}} p(\mathbf{v}', \mathbf{h}) v'_i h_j \\ &= \mathbb{E}_{\text{data}} [v_i h_j] - \mathbb{E}_{\text{model}} [v_i h_j] \end{aligned}$$

- Gradient ascent learning rules (learning rate β):

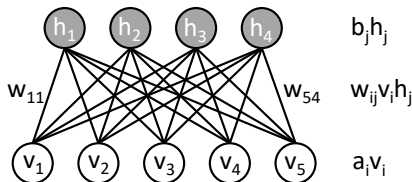
$$\Delta w_{ij} = \beta (\mathbb{E}_{\text{data}} [v_i h_j] - \mathbb{E}_{\text{model}} [v_i h_j])$$

$$\Delta a_i = \beta (\mathbb{E}_{\text{data}} [v_i] - \mathbb{E}_{\text{model}} [v_i])$$

$$\Delta b_j = \beta (\mathbb{E}_{\text{data}} [h_j] - \mathbb{E}_{\text{model}} [h_j])$$

Restricted Boltzmann Machine

Training



- Given \mathbf{v} it is easy to sample \mathbf{h} :

$$p(h_j = 1 \mid \mathbf{v}) = \frac{w(h_j = 1 \mid \mathbf{v})}{w(h_j = 0 \mid \mathbf{v}) + w(h_j = 1 \mid \mathbf{v})}$$

Using $w(h_j = 1 \mid \mathbf{v}) = \exp(-b_j - \sum_i w_{ij} v_i)$ and $w(h_j = 0 \mid \mathbf{v}) = 1$:

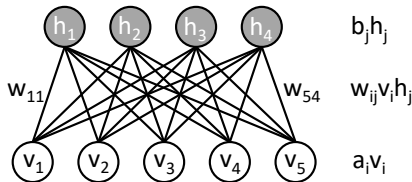
$$\begin{aligned} p(h_j = 1 \mid \mathbf{v}) &= \frac{\exp(-b_j - \sum_i w_{ij} v_i)}{1 + \exp(-b_j - \sum_i w_{ij} v_i)} \\ &= \sigma(b_j + \sum_i w_{ij} v_i) \end{aligned}$$

- Given \mathbf{h} it is easy to sample \mathbf{v} :

$$p(v_i = 1 \mid \mathbf{h}) = \sigma(a_i + \sum_j w_{ij} h_j)$$

Restricted Boltzmann Machine

Training



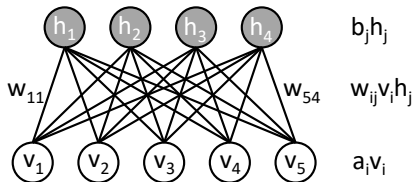
Approximate by **direct sampling** of $\mathbf{h} \mid \mathbf{v}$:

$$s_{ij} := \mathbb{E}_{\text{data}}[v_i h_j] = \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{v}) v_i h_j$$

- 1 $\mathbf{S} = 0$
- 2 For each \mathbf{v}_t in data batch $(\mathbf{v}_1, \dots, \mathbf{v}_B)$:
 - 1 Sample $\mathbf{h} \sim p(\mathbf{h} \mid \mathbf{v}_t) = \sigma(\mathbf{b} + \mathbf{W}^\top \mathbf{v}_t)$
 - 2 $\mathbf{S} \leftarrow \mathbf{S} + \mathbf{v}_t \mathbf{h}^\top$
- 3 $\mathbb{E}_{\text{data}}[\mathbf{v} \mathbf{h}^\top] \approx \frac{1}{B} \mathbf{S}$

Restricted Boltzmann Machine

Training



Approximate by **Gibbs sampling**:

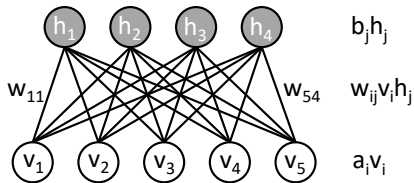
$$q_{ij} = \mathbb{E}_{\text{model}} [v_i h_j] = \sum_{\mathbf{v}, \mathbf{h}} p(\mathbf{v}, \mathbf{h}) v_i h_j$$

- ① $\mathbf{Q} = 0$, Set \mathbf{v} to initial random vector.
- ② Repeat N_s times (**until convergence**):
 - ① Sample $\mathbf{h} \sim p(\mathbf{h} | \mathbf{v}) = \sigma(\mathbf{b} + \mathbf{W}^\top \mathbf{v})$
 - ② Sample $\mathbf{v} \sim p(\mathbf{v} | \mathbf{h}) = \sigma(\mathbf{a} + \mathbf{W} \mathbf{h})$
 - ③ $\mathbf{Q} \leftarrow \mathbf{Q} + \mathbf{v} \mathbf{h}^\top$
- ③ $\mathbb{E}_{\text{model}} [v_i h_j] \approx \frac{1}{N_s} \mathbf{Q}$

This scheme converges much slower, because the samples of \mathbf{h} and \mathbf{v} are correlated!

Restricted Boltzmann Machine

Contrastive Divergence CD- n



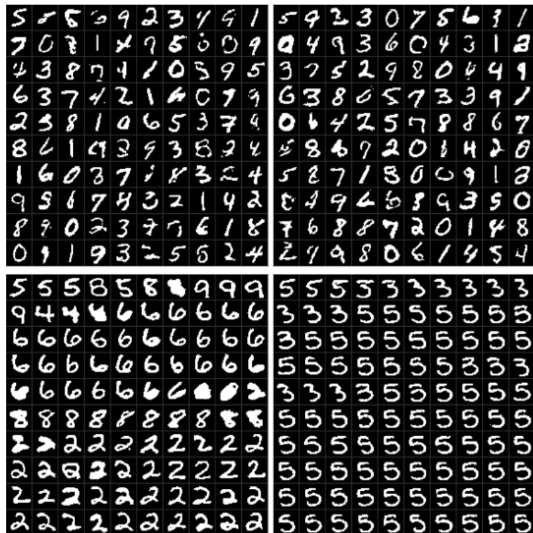
Approximate by **Contrastive divergence**:

$$q_{ij} = \mathbb{E}_{\text{model}} [v_i h_j] = \sum_{\mathbf{v}, \mathbf{h}} p(\mathbf{v}, \mathbf{h}) v_i h_j$$

- ① $\mathbf{Q} = 0$, Set \mathbf{v} to initial random vector.
- ② Repeat n times (**not until convergence**):
 - ① Sample $\mathbf{h} \sim p(\mathbf{h} | \mathbf{v}) = \sigma(\mathbf{b} + \mathbf{W}^\top \mathbf{v})$
 - ② Sample $\mathbf{v} \sim p(\mathbf{v} | \mathbf{h}) = \sigma(\mathbf{a} + \mathbf{W} \mathbf{h})$
 - ③ $\mathbf{Q} \leftarrow \mathbf{Q} + \mathbf{v} \mathbf{h}^\top$
- ③ $\mathbb{E}_{\text{model}} [v_i h_j] \approx \frac{1}{n} \mathbf{Q}$

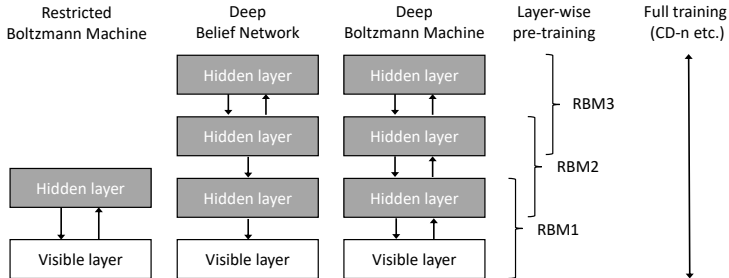
Commonly used: CD-1. Will not provide an accurate estimate of $\mathbb{E}_{\text{model}} [v_i h_j]$ at a given time, but lead to gradients that slowly follow the training process. Fast and works reasonable in practice.

Restricted Boltzmann Machine



Deep Boltzmann Machines

Deep Belief Network



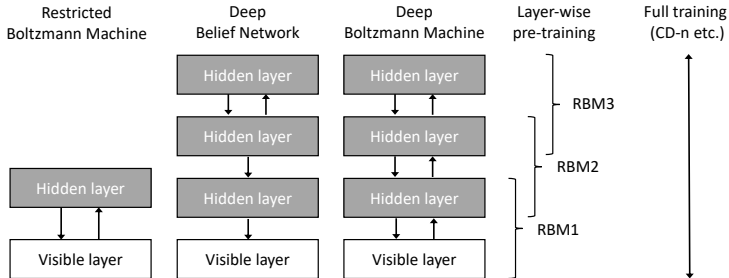
- Deep generative network, Introduction of the term “Deep learning” (Hinton et al., 2006; Hinton and Salakhutdinov, 2006)
- RBM in two top layers, deep mapping to visible nodes.
- **Pretraining** as stack of RBMs:
 - First train the bottom hidden as a normal RBM
 - Now samples can be generated for the first hidden layer, which are used to train the second hidden layer, etc.
- Use pretrained deep RBMs to define a deep MLP ($\mathbf{h}^{(0)} = \mathbf{v}$):

$$\mathbf{h}^{(l)} = \sigma(\mathbf{b}^{(l)} + \mathbf{h}^{(l-1)\top} \mathbf{W}^{(l)})$$

Then train supervised. One of the first deep learning algorithms.

Deep Boltzmann Machines

Deep Boltzmann Machine



- Undirected deep generative network
- Stack of RBMs.
- Can be rewritten in a bipartite graph by grouping visible and even hidden layers, and odd hidden layers:

$$\begin{array}{cccc} \mathbf{h}^{(1)} & \mathbf{h}^{(3)} & \mathbf{h}^{(5)} & \dots \\ \mathbf{v} & \mathbf{h}^{(2)} & \mathbf{h}^{(4)} & \dots \end{array}$$

Thus, Gibbs sampling can be used.

- Alternative: layer-wise pre-training, then CD- n for unsupervised learning.

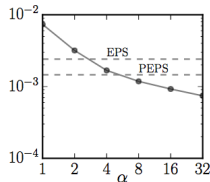
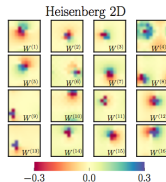
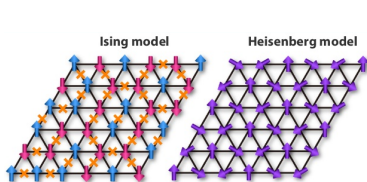
Data generation

- Reconstruction \mathbf{v}_0 of a given data point \mathbf{x} :
 - Fix visible layer $\mathbf{v} = \mathbf{x}$, use MCMC sampling to find the state of the hidden layer \mathbf{h} which maximizes the probability distribution $p(\mathbf{h}|\mathbf{v})$
 - Fixing obtained \mathbf{h} , find reconstruction \mathbf{v}_0 of original data point which maximizes the probability $p(\mathbf{v}_0|\mathbf{h})$.
- Deep Boltzmann Machine: Run forward pass to the last hidden layer, then backward pass in reverse.
- Application, e.g. image denoising. Example: randomly flip a fraction of the black&white bits in the validation data, and use Boltzmann machines to reconstruct (de-noise) the digit images.



Neural Quantum States

Carleo and Troyer, Science 2017



- Physical model with spin variables v_i
- Use RBM to represent physical spins v_i and “hidden” spins h_i .

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}; \theta) &= -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} w_{ij} v_i h_j \\ &= -\mathbf{a}^\top \mathbf{v} - \mathbf{b}^\top \mathbf{h} - \mathbf{v}^\top \mathbf{W} \mathbf{h}. \end{aligned}$$

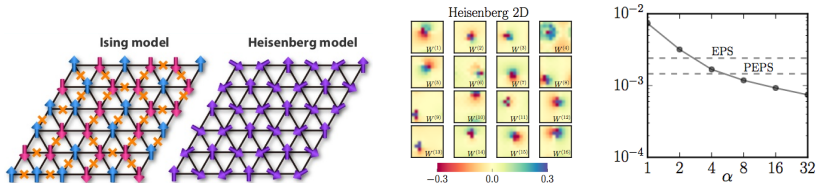
- Model the QM wavefunction ψ by the marginal spin density:

$$\psi(\mathbf{v}; \theta) = \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h}; \theta)}$$

- The network weights are complex-valued in order to provide a complete description of amplitude and phase of the wave-function.

Neural Quantum States

Carleo and Troyer, Science 2017



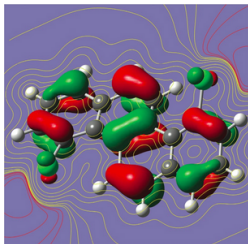
- Optimize ψ by minimizing the loss

$$\varepsilon(\theta) = \frac{\langle \psi | H | \psi \rangle}{\langle \psi | \psi \rangle} = \frac{\int \psi^*(\mathbf{v}; \theta) (H\psi)(\mathbf{v}; \theta) d\mathbf{v}}{\int \psi^*(\mathbf{v}; \theta) \psi(\mathbf{v}; \theta) d\mathbf{v}}$$

- Network is trained using the variational Quantum Monte Carlo formulation.

Project: Deep variational QMC for Molecules

Current Project (Noé group)



- Electronic many-body Hamiltonian:

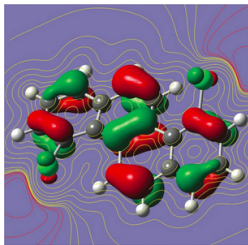
$$H = \sum_i \nabla_i^2 - \sum_i \sum_l \frac{Z_l}{|\mathbf{r}_i - \mathbf{R}_l|} + \frac{1}{2} \sum_i \sum_j \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}$$

- Find ground state (minimal) energy E and wavefunction ψ of time-stationary Schrödinger equation:

$$H\psi = \epsilon\psi$$

Project: Deep variational QMC for Molecules

Current Project (Noé group)



- Represent ψ by deep neural network and optimize by:

$$\min_{\theta} \varepsilon(\theta) = \frac{\langle \psi | H | \psi \rangle}{\langle \psi | \psi \rangle} = \frac{\int \psi^*(\mathbf{x}; \theta) (H\psi)(\mathbf{x}; \theta) d\mathbf{x}}{\int \psi^*(\mathbf{x}; \theta) \psi(\mathbf{x}; \theta) d\mathbf{x}}$$

s.t. $\psi(\mathbf{x})$ antisymmetric

where antisymmetry is defined by:

$$\psi(\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_j, \dots, \mathbf{x}_n) = -\psi(\mathbf{x}_1, \dots, \mathbf{x}_j, \dots, \mathbf{x}_i, \dots, \mathbf{x}_n)$$

- Variational Quantum Monte Carlo formulation: Sample $\mathbf{x} \sim |\psi(\mathbf{x})|^2$ and train with minibatches.