Advanced Machine Learning
Lecture 20

Restricted Boltzmann Machines

01.02.2016

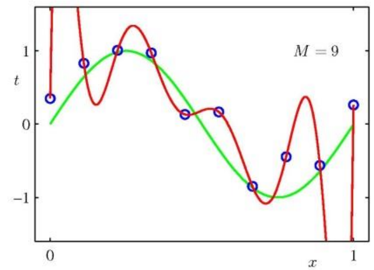
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de/

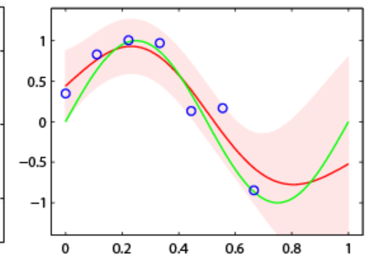
leibe@vision.rwth-aachen.de

This Lecture: *Advanced Machine Learning*

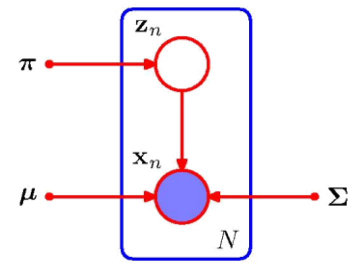
- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Gaussian Processes

$$f : \mathcal{X} \to \mathbb{R}$$

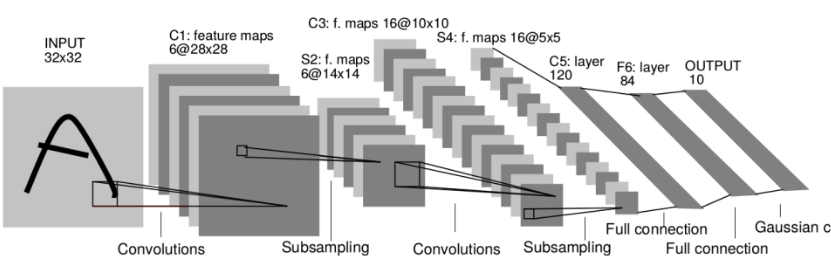




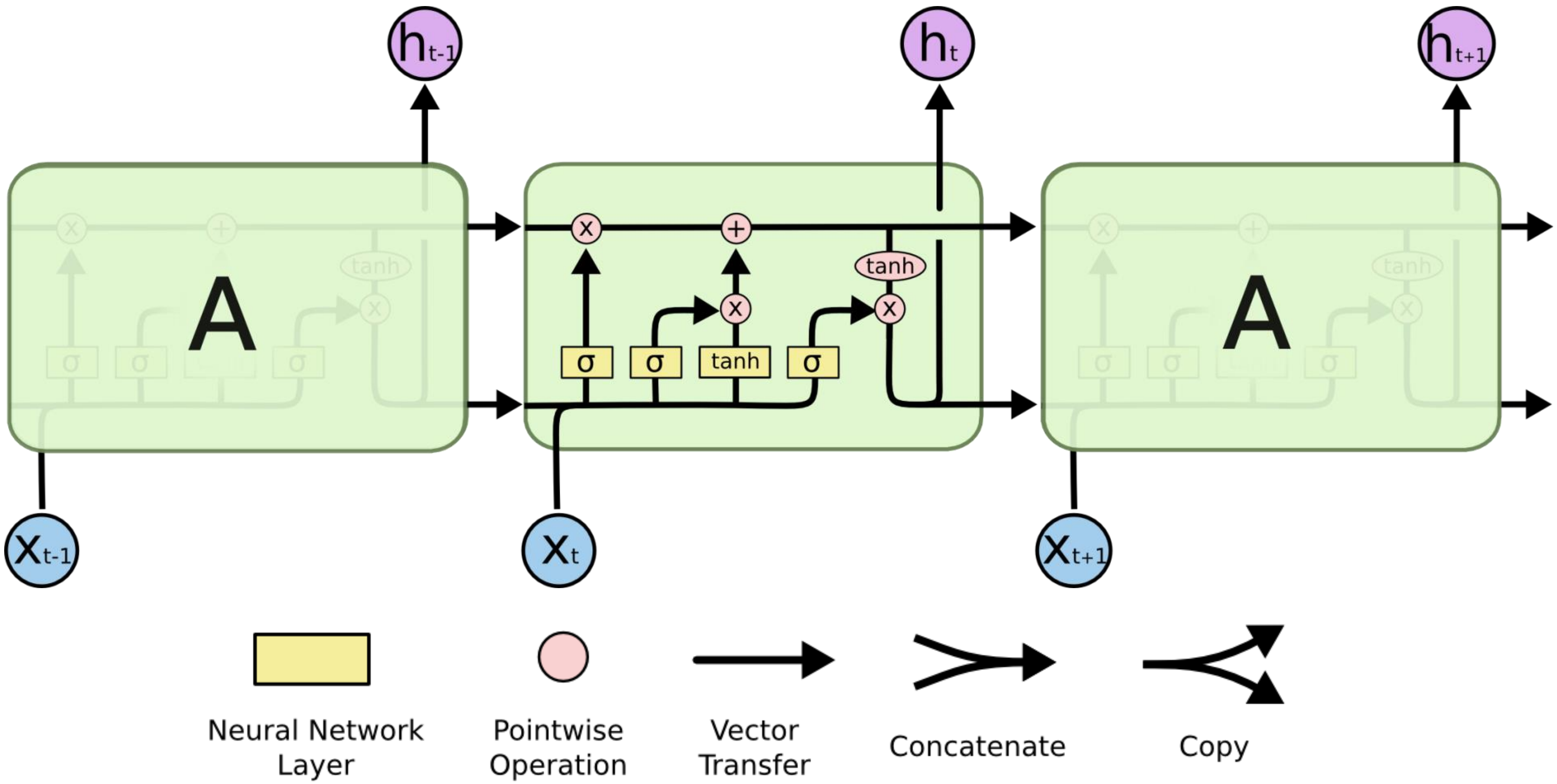
- Learning with Latent Variables
  - Prob. Distributions & Approx. Inference
  - Mixture Models
  - EM and Generalizations



- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, RNNs, RBMs, etc.



Recap: Long Short-Term Memory

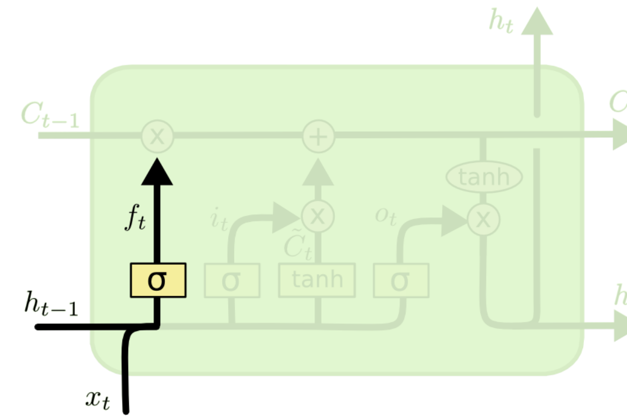


- **LSTMs**
  - **Inspired by the design of memory cells**
  - **Each module has 4 layers, interacting in a special way.**

Image source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

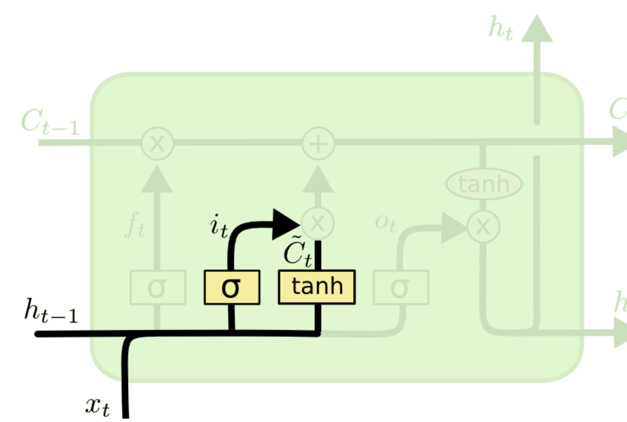
Recap: Elements of LSTMs

- **Forget gate** layer
  - Look at $\mathbf{h}_{t\text{-}1}$ and $\mathbf{x}_t$ and output a number between 0 and 1 for each dimension in the cell state $\mathbf{C}_{t\text{-}1}$.
    - 0: completely delete this,
    - 1: completely keep this.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \;+\; b_f\right)$$

- **Update gate** layer
  - Decide what information to store in the cell state.
  - Sigmoid network (input gate layer) decides which values are updated.
  - tanh layer creates a vector of new candidate values that could be added to the state.



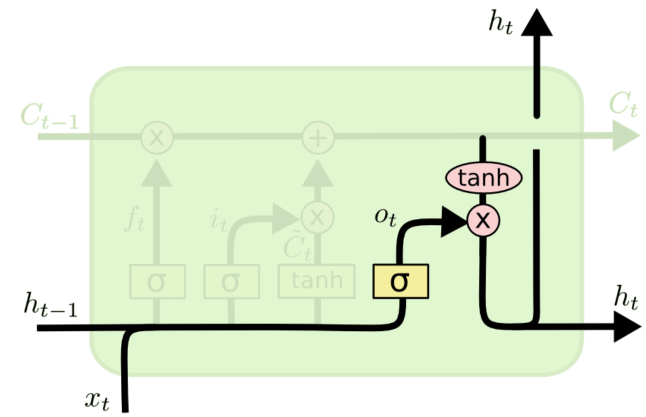
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

Source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recap: Elements of LSTMs

- **Output gate layer**
  - Output is a filtered version of our gate state.
  - First, apply sigmoid layer to decide what parts of the cell state to output.
  - Then, pass the cell state through a tanh (to push the values to be between -1 and 1) and multiply it with the output of the sigmoid gate.



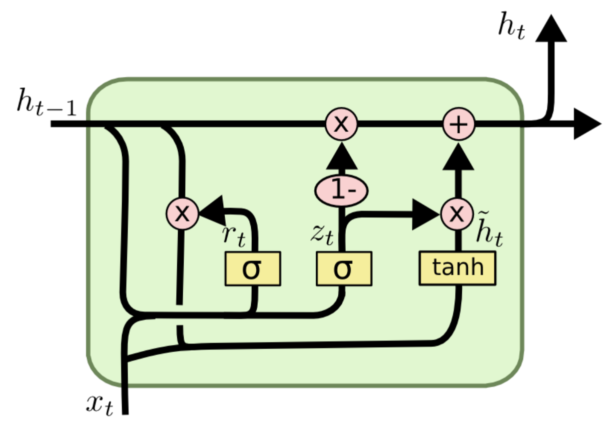
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recap: Gated Recurrent Units (GRU)

- Simpler model than LSTM
  - Combines the forget and input gates into a single **update gate** z_t.
  - Similar definition for a **reset gate** r_t, but with different weights.
  - In both cases, merge the cell state and hidden state.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Empirical results
  - Both LSTM and GRU can learn much longer-term dependencies than regular RNNs
  - GRU performance similar to LSTM (no clear winner yet), but fewer parameters.

Source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Topics of This Lecture

- Unsupervised Learning
  - Motivation
- Energy based Models
  - Definition
  - EBMs with Hidden Units
  - Learning EBMs
- Restricted Boltzmann Machines
  - Definition
  - RBMs with Binary Units
  - RBM Learning
  - Contrastive Divergence

RWTH AACHEN UNIVERSITY

Looking Back...

- We have seen very powerful deep learning methods.
  - Deep MLPs
  - CNNs
  - RNNs (+LSTM, GRU)
  - (When used properly) they work very well and have achieved great successes in the last few years.

- But...
  - All of those models have many parameters.
  - They need A LOT of training data to work well.
  - Labeled training data is very expensive.

  $\Rightarrow$ *How can we reduce the need for labeled data?*

Reducing the Need for Labeled Data

- Reducing Model Complexity
  - E.g., GoogLeNet: big reduction in the number of parameters compared to AlexNet (60M $\rightarrow$ 5M).
  - $\Rightarrow$ More efficient use of the available training data.
- Transfer Learning
  - Idea: Pre-train a model on a large data corpus (e.g., ILSVRC), then just fine-tune it on the available task data.
  - This is what is currently done in Computer Vision.
  - $\Rightarrow$ Benefit from generic representation properties of the pre-trained model.
- Unsupervised / Semi-supervised Learning
  - Idea: Try to learn a generic representation from unlabeled data and then just adapt it for the supervised classification task.

Topics of This Lecture

- Unsupervised Learning
  - Motivation
- Energy based Models
  - Definition
  - EBMs with Hidden Units
  - Learning EBMs
- Restricted Boltzmann Machines
  - Definition
  - RBMs with Binary Units
  - RBM Learning
  - Contrastive Divergence

Energy Based Models (EBM)

- **Energy Based Probabilistic Models**
  - Define the joint probability over a set of variables $\mathbf{x}$ through an energy function

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

where the normalization factor Z is called the partition function

$$Z = \sum_{\mathbf{x}} e^{-E(\mathbf{x})}$$

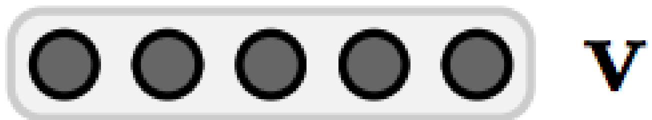
  - An EBM can be learned by performing (stochastic) gradient descent on the negative log-likelihood of the training data

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{N} \sum_{x_n \in \mathcal{D}} \log p(x_n)$$

using the stochastic gradient $-\frac{\partial \log p(x_n)}{\partial \theta}$

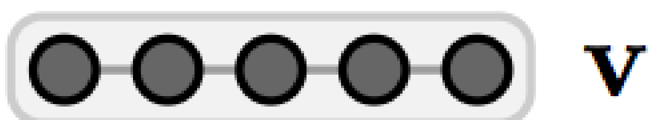
Energy Based Models: Examples

- **We have been using EBMs all along...**
  - E.g., Collections of independent variables



$$E(\mathbf{v}) = \sum_i f_i(v_i; b_i)$$ where f_i encodes the NLL of v_i

  - E.g., Markov Random Fields



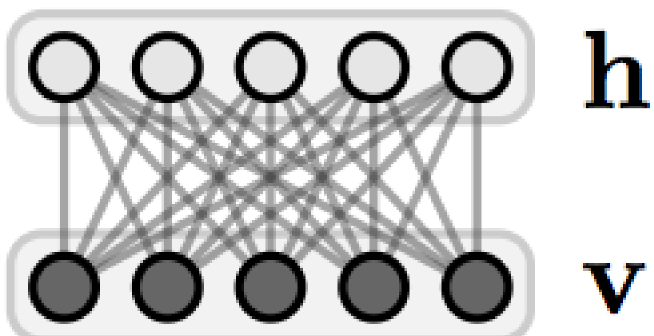
MRF

$$E(\mathbf{v}) = \sum_i f_i(v_i; b_i) + \sum_{(i,j)} f_{ij}(v_i, v_j; w_{ij})$$

Image Source: S.M. A. Eslami et al.

EBMs with Hidden Units

- In the following
  - We want to explore deeper models with (multiple layers of) hidden units
  - E.g., Restricted Boltzmann machines



  - This will lead to Deep Belief Networks (DBN) that were popular until very recently.

EBMs with Hidden Units

- **Hidden variable formulation**
  - In many cases of interest, we do not observe the examples fully
  - Split them into an observed part $\mathbf{x}$ and a hidden part $\mathbf{h}$:

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

- **Notation**
  - We define the free energy (inspired by physics)

$$\mathcal{F}(\mathbf{x}) = -\log \sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}$$

and write the joint probability as

$$p(\mathbf{x}) = \frac{e^{-\mathcal{F}(\mathbf{x})}}{Z} \qquad \text{with} \qquad Z = \sum_{\mathbf{x}} e^{-\mathcal{F}(\mathbf{x})}.$$

EBMs with Hidden Units

- Expressing the gradient
  - Free energy formulation of the joint probability

$$p(\mathbf{x}) = \frac{e^{-\mathcal{F}(\mathbf{x})}}{Z} \quad \text{with} \quad Z = \sum_{\mathbf{x}} e^{-\mathcal{F}(\mathbf{x})}.$$

  - The negative log-likelihood gradient then takes the following form

$$-\frac{\partial \log p(\mathbf{x})}{\partial \theta} = \underbrace{\frac{\partial \mathcal{F}(\mathbf{x})}{\partial \theta}}_{\text{Positive phase}} - \underbrace{\sum_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}}) \frac{\partial \mathcal{F}(\tilde{\mathbf{x}})}{\partial \theta}}_{\text{Negative phase}}.$$

(The names do not refer to the sign of each term, but to their effect on the probability density defined by the model)

Challenge for Learning

$$-\frac{\partial \log p(\mathbf{x})}{\partial \theta} = \frac{\partial \mathcal{F}(\mathbf{x})}{\partial \theta} - \sum_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}}) \frac{\partial \mathcal{F}(\tilde{\mathbf{x}})}{\partial \theta}.$$

- **Problem**
  - **Difficult to determine this gradient analytically.**
  - **Computing it would involve evaluating**

$$\mathbb{E}_p\left[\frac{\partial \mathcal{F}(\mathbf{x})}{\partial \theta}\right]$$

i.e., the expectation over all possible configurations of the input $\mathbf{x}$ under the distribution p formed by the model!

⇒ Often infeasible.

Steps Towards a Solution...

- **Monte Carlo approximation**
  - Estimate the expectation using a fixed number of model samples for the negative phase gradient ("negative particles")

$$-\frac{\partial \log p(\mathbf{x})}{\partial \theta} \approx \underbrace{\frac{\partial \mathcal{F}(\mathbf{x})}{\partial \theta}}_{\text{free energy at current point}} - \underbrace{\frac{1}{|\mathcal{N}|} \sum_{\tilde{\mathbf{x}} \in \mathcal{N}} \frac{\partial \mathcal{F}(\tilde{\mathbf{x}})}{\partial \theta}}_{\text{avg. free energy for all other points}}.$$

  - With this, we almost have a practical stochastic algorithm for learning an EBM.
  - We just need to define how to extract the negative particles $\mathcal{N}$.
    - Many sampling approaches can be used here.
    - MCMC methods are especially well-suited.

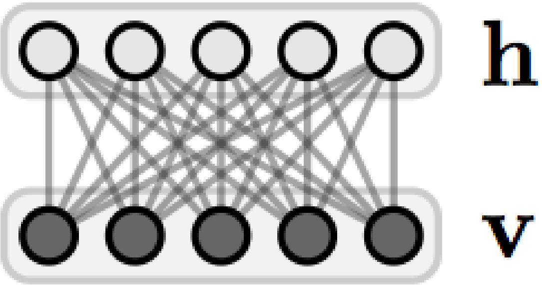
And this is where all parts of the lecture finally come together...

Topics of This Lecture

- Unsupervised Learning
  - Motivation
- Energy based Models
  - Definition
  - EBMs with Hidden Units
  - Learning EBMs
- **Restricted Boltzmann Machines**
  - Definition
  - RBMs with Binary Units
  - RBM Learning
  - Contrastive Divergence

Restricted Boltzmann Machines (RBM)

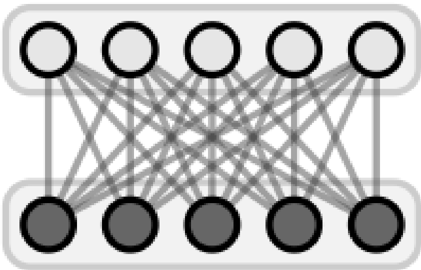
- **Boltzmann Machines** (BM)
  - BMs are a particular form of log-linear MRF, for which the free energy is linear in its free parameters.
  - To make them powerful enough to represent complicated distributions, we consider some of the variables as hidden.
  - In their general form, they are very complex to handle.
- **Restricted Boltzmann Machines** (RBM)
  - RBMs are BMs that are restricted not to contain visible-visible and hidden-hidden connections.



  - This makes them far easier to work with.

Restricted Boltzmann Machines (RBM)

- Properties
  - Components
    - Visible units $\mathbf{v}$ with offsets $\mathbf{b}$
    - Hidden units $\mathbf{h}$ with offsets $\mathbf{c}$
    - Connection matrix W



  - Energy Function of an RBM

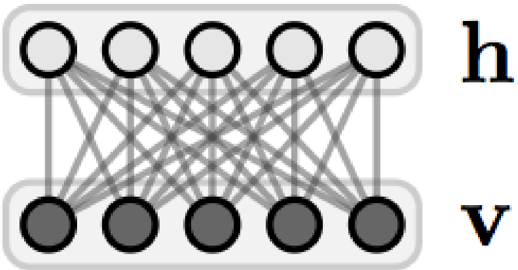
$$E(\mathbf{v},\mathbf{h}) = -\sum_i b_i v_i - \sum_j c_j h_j - \sum_{i,j} w_{ij} v_i h_j$$
$$= -\mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h} - \mathbf{h}^\top W \mathbf{v}$$

  - This translates to a free energy formula

$$\mathcal{F}(\mathbf{v}) = -\mathbf{b}^\top \mathbf{v} - \sum_i \log \sum_{h_i} e^{h_i (c_i + W_i \mathbf{v})}.$$

Restricted Boltzmann Machines (RBM)

- Properties (cont'd)
  - Because of their specific structure, visible and hidden units are conditionally independent given one another.



  - Therefore the following factorization property holds:

$$p(\mathbf{h}|\mathbf{v}) = \prod_i p(h_i|\mathbf{v})$$

$$p(\mathbf{v}|\mathbf{h}) = \prod_j p(v_j|\mathbf{h}).$$

Restricted Boltzmann Machines (RBM)

- **Interpretation of RBMs**
  - Factorization property

$$p(\mathbf{h}|\mathbf{v}) = \prod_i p(h_i|\mathbf{v})$$

$$p(\mathbf{v}|\mathbf{h}) = \prod_j p(v_j|\mathbf{h}).$$

  - RBMs can be seen as a product of experts specializing on different areas.
  - Experts detect *negative constraints*, if one of them returns zero, the entire product is zero.

RBMs with Binary Units

- Binary units
  - v_j and $h_i \in \{0,1\}$ are considered Bernoulli variables.
  - This results in a probabilistic version of the usual neuron activation function

$$\begin{aligned} p(h_i = 1|\mathbf{v}) &= \sigma(c_i + W_i\mathbf{v}) \\ p(v_j = 1|\mathbf{h}) &= \sigma(b_j + W_j^\top \mathbf{h}) \end{aligned}$$

  - The free energy of an RBM with binary units simplifies to

$$\mathcal{F}(\mathbf{v}) = -\mathbf{b}^\top \mathbf{v} - \sum_i \log\left(1 + e^{(c_i + W_i\mathbf{v})}\right).$$

RBMs with Binary Units

- Binary units
  - Free energy

$$\mathcal{F}(\mathbf{v}) = -\mathbf{b}^\top \mathbf{v} - \sum_i \log\left(1 + e^{(c_i + W_i \mathbf{v})}\right).$$

  - This results in the iterative update equations for the gradient log-likelihoods

$$-\frac{\partial \log p(\mathbf{v})}{\partial W_{ij}} = \mathbb{E}_{\mathbf{v}}\left[p(h_i|\mathbf{v}) \cdot v_j\right] - v_j^{(t)} \cdot \sigma(W_i \cdot \mathbf{v}^{(t)} + c_i)$$

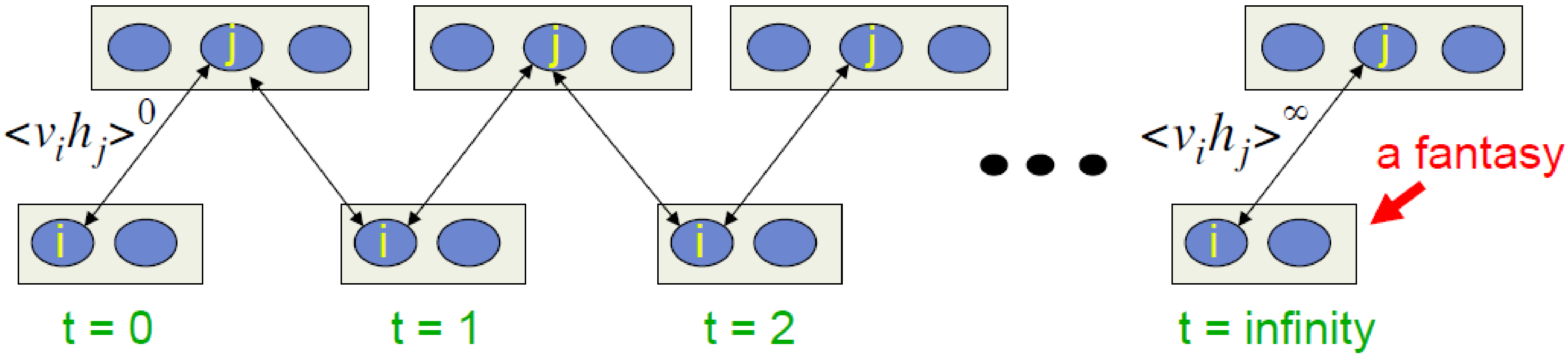
$$-\frac{\partial \log p(\mathbf{v})}{\partial c_i} = \mathbb{E}_{\mathbf{v}}[p(h_i|\mathbf{v})] - sigm(W_i \cdot \mathbf{v}^{(t)})$$

$$-\frac{\partial \log p(\mathbf{v})}{\partial b_j} = \mathbb{E}_{\mathbf{v}}[p(v_j|\mathbf{h})] - \mathbf{v}_j^{(t)}$$

RWTH AACHEN UNIVERSITY

RBM Learning

- Iterative approach

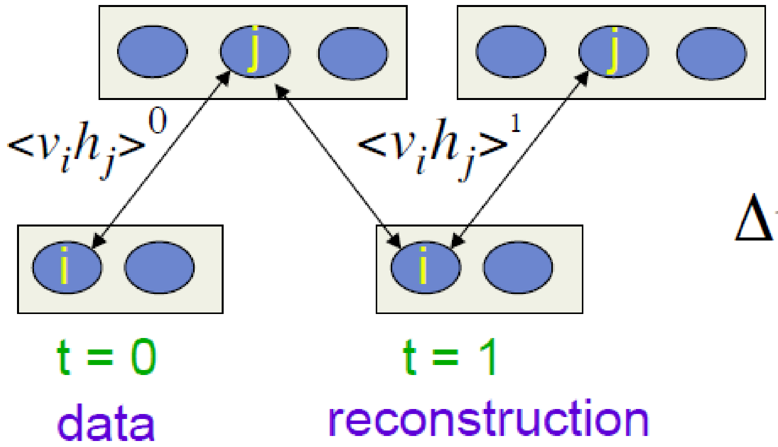


  - Start with a training vector on the visible units. Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.
  - This implements a Markov chain that we use to approximate the gradient

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = < v_i, h_j >^0 - < v_i, h_j >^\infty$$

  - Better method in practice: Contrastive Divergence

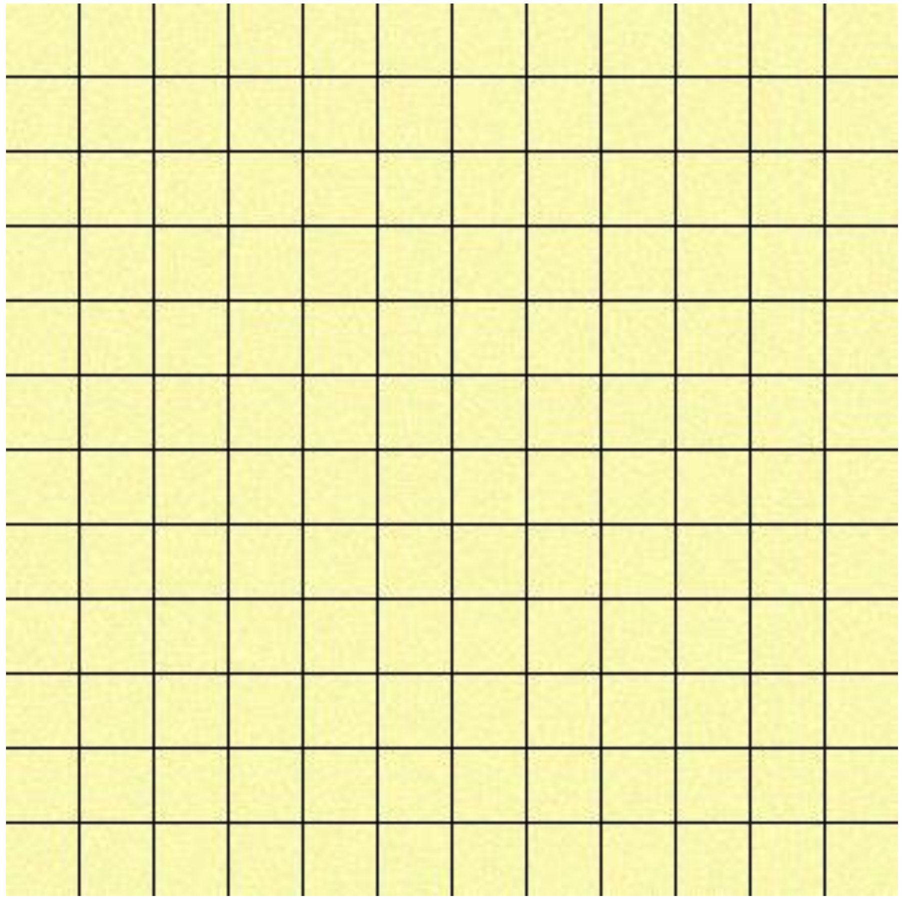
Contrastive Divergence



$$\Delta w_{ij} = \varepsilon \left(<v_i h_j>^0 - <v_i h_j>^1 \right)$$

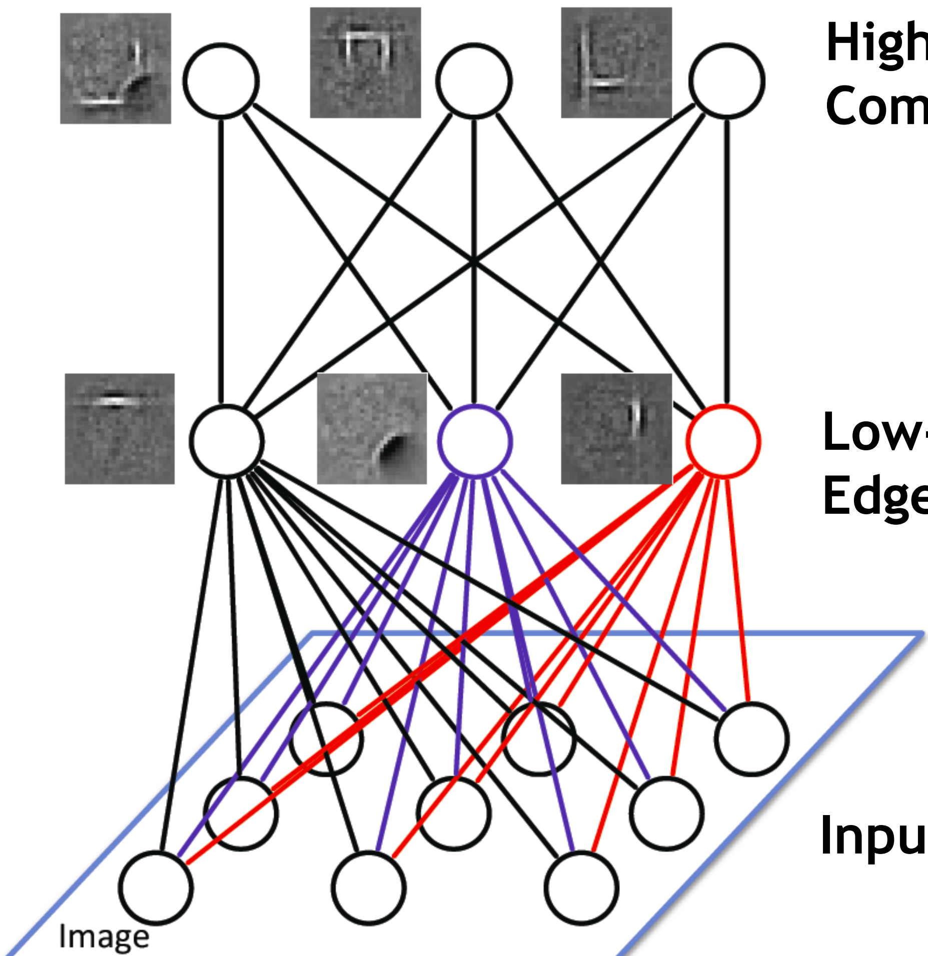
- **A surprising shortcut**
  - **Start with a training vector on the visible units.**
  - **Update all the hidden units in parallel.**
  - **Update the all visible units in parallel to get a "reconstruction".**
  - **Update the hidden units again (no further iterations).**
  - **This does not follow the gradient of the log likelihood. But it works well [Hinton].**

Example



- **RBM training on MNIST**
  - **Persistent Contrastive Divergence with chain length 15**

Extension: Deep RBMs



Higher-level features:
Combinations of edges

Low-level features:
Edges

Input: pixels

Built from unlabeled input.