# DD2437 – Artificial Neural Networks and Deep Architectures (annda)

## Lecture 8b: **Boltzmann machines and RBMs, autoencoders**

Pawel Herman

Computational Science and Technology (CST)

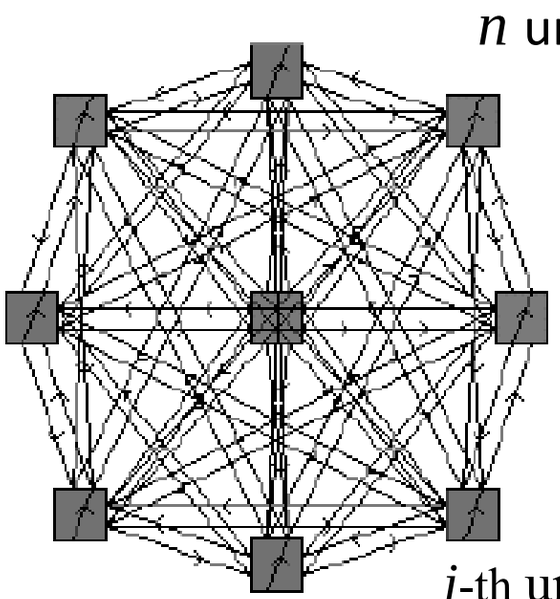
KTH Royal Institute of Technology

September 2019

# Lecture overview

- Boltzmann machine
- Restricted Boltzmann machine, RBM
- Autoencoders

# Hopfield network




$$\forall_i \, w_{i,i} = 0$$ no self-connections

$$\vec{x}' = \mathrm{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

# From Hopfield networks to Boltzmann machines

- Continuous Hopfield network

$$x_i = \frac{1}{1+e^{-a}} \quad \text{instead of} \quad x_i = \text{sgn}(a)$$

- Stochastic component

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ -1, & \text{with probability } 1\text{-}p_i \end{cases} \qquad p_i = \frac{1}{1+e^{-\frac{1}{T}\sum_j w_{i,j}x_j}}$$

$T$ is a positive temperature const.

# From Hopfield networks to Boltzmann machines

- Stochastic component

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ -1, & \text{with probability } 1\text{-}p_i \end{cases}$$

$$p_i = \frac{1}{1 + e^{-\frac{1}{T}\sum_j w_{i,j} x_j}}$$

$$p(\nu) = \frac{1}{1+e^{-\nu}} \quad \text{where} \quad \nu = \frac{1}{T}\sum_j w_{i,j} x_j$$

$T$ controls the level of randomness

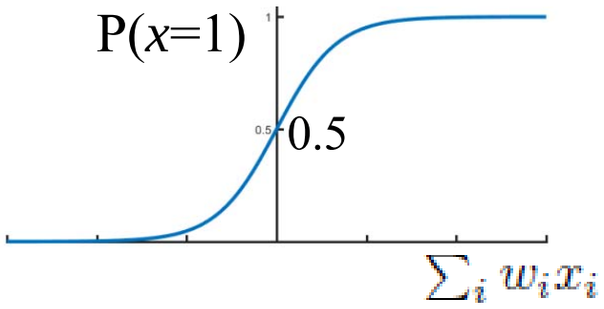
# From Hopfield networks to Boltzmann machines

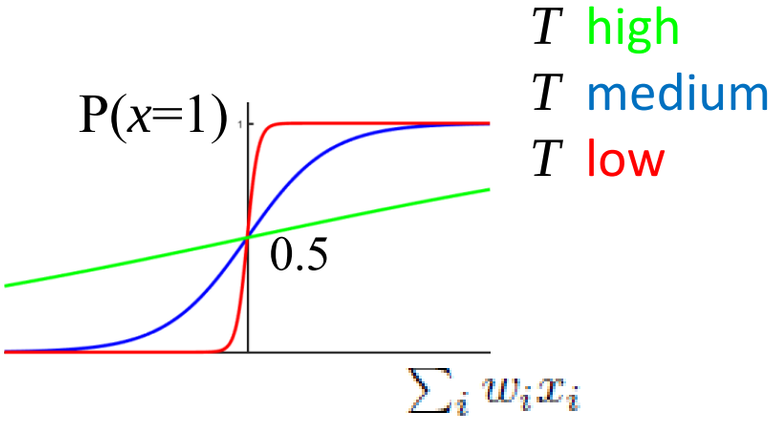
- Stochastic component

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ -1, & \text{with probability } 1\text{-}p_i \end{cases}$$

$$p_i = \frac{1}{1 + e^{-\frac{1}{T}\sum_j w_{i,j} x_j}}$$

$$p(v) = \frac{1}{1 + e^{-v}} \quad \text{where} \quad v = \frac{1}{T}\sum_j w_{i,j} x_j$$



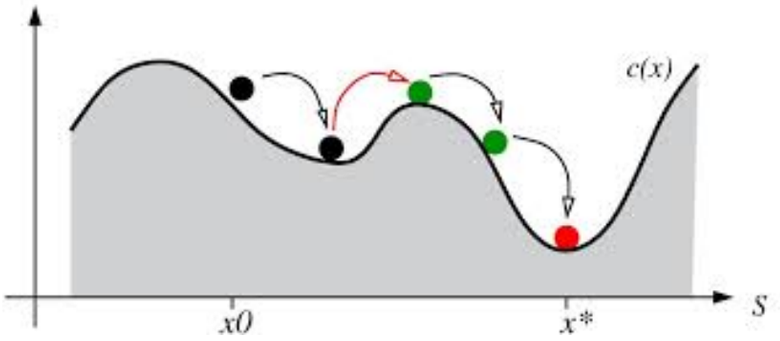
# From Hopfield networks to Boltzmann machines

- Stochastic component

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ -1, & \text{with probability } 1\text{-}p_i \end{cases}$$

$$p_i = \frac{1}{1 + e^{-\frac{1}{T}\sum_j w_{i,j} x_j}}$$

Analogy to simulated annealing (relaxation technique common in metallurgy)

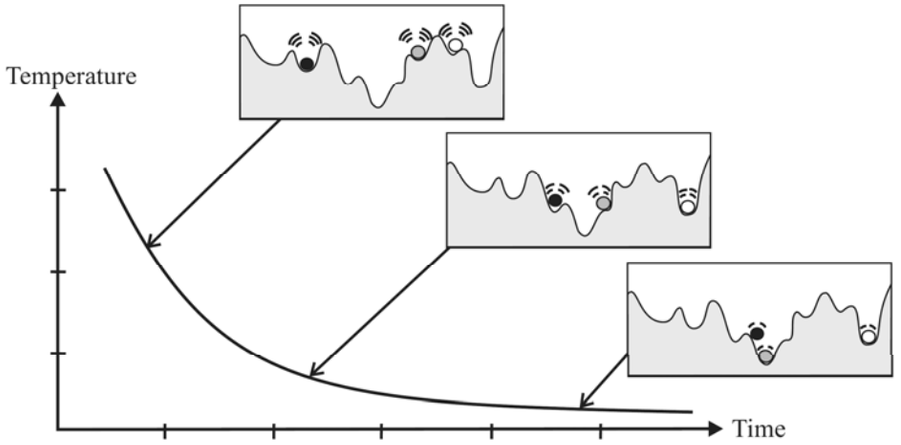



$$p_{\Delta E} = \frac{1}{1 + e^{\Delta E / T}}$$

*"When optimising a large complex system with many degrees of freedom, instead of always going downhill, try to go downhill most of the time"*

# Simulated annealing to reach the global energy min

The critical role of temperature $T$.

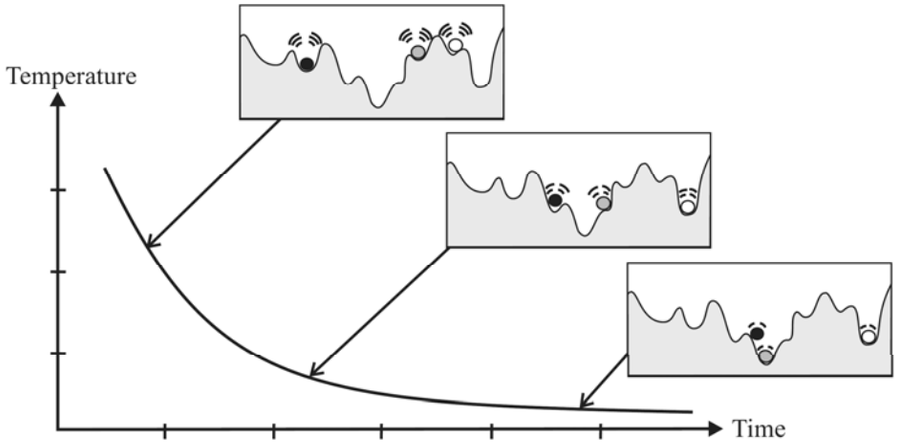


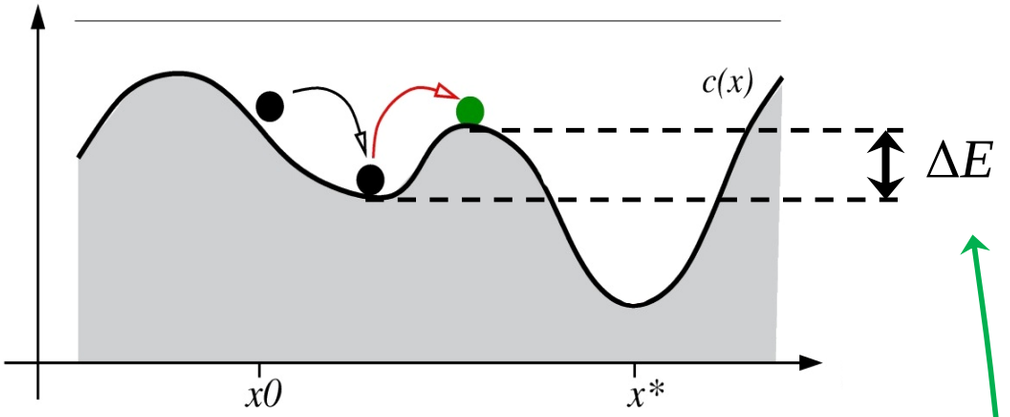

$T$ decreases over time

$$p_{\Delta E} = \frac{1}{1 + e^{\Delta E / T}}$$

# Simulated annealing to reach the global energy min

The critical role of temperature $T$.




$$p_{\Delta E} = \frac{1}{1 + e^{\Delta E / T}}$$

# From Hopfield networks to Boltzmann machines

- Energy of this stochastic network is the same as before

$$E = -\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}x_i x_j$$

- The key difference is a stochastic nature of transitions

from state $s1$ to $s2$:

$$p_{s1\to s2} = \frac{1}{1+e^{(E_2-E_1)/T}} = \frac{1}{1+e^{\Delta E/T}}$$

# From Hopfield networks to Boltzmann machines

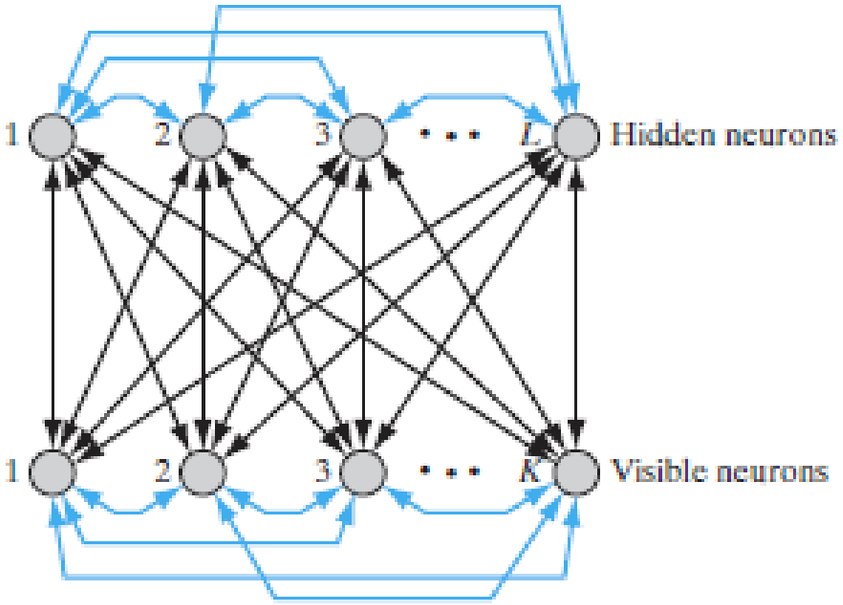
- Energy of this stochastic network is the same as before

$$E = -\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}x_i x_j$$

- Given a set of examples $\{\vec{x}_i\}_1^m$ the idea is to adjust $\mathbf{W}$ to describe data distribution (well matched to these examples)

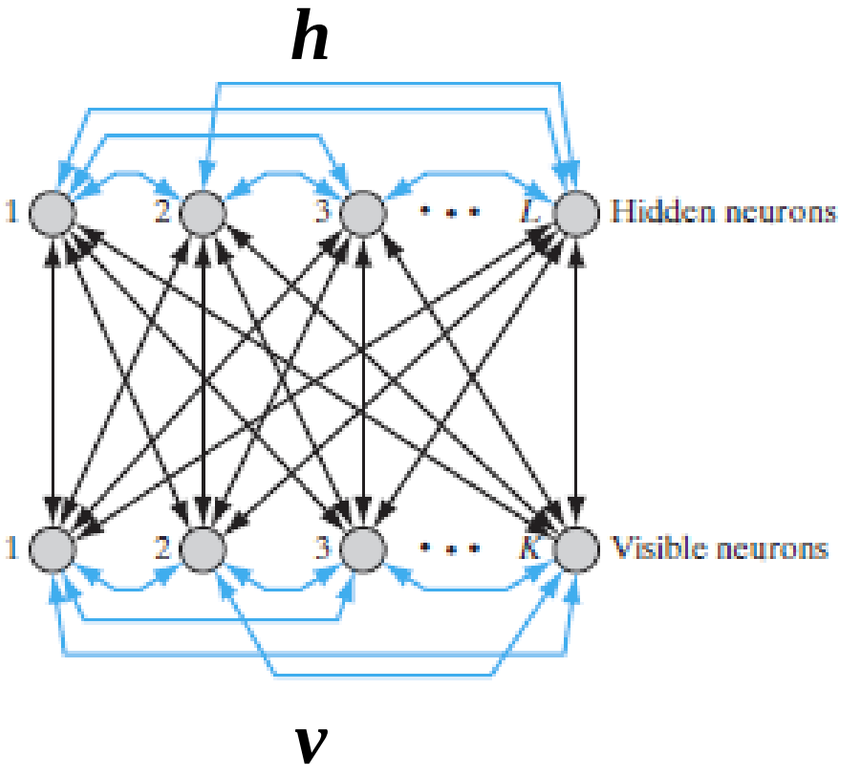
$$P(\vec{x}\,|\,\mathbf{W}) = \frac{e^{-E}}{Z} = \frac{1}{Z(\mathbf{W})}\exp\left(\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x}\right)$$

# Hidden and visible units



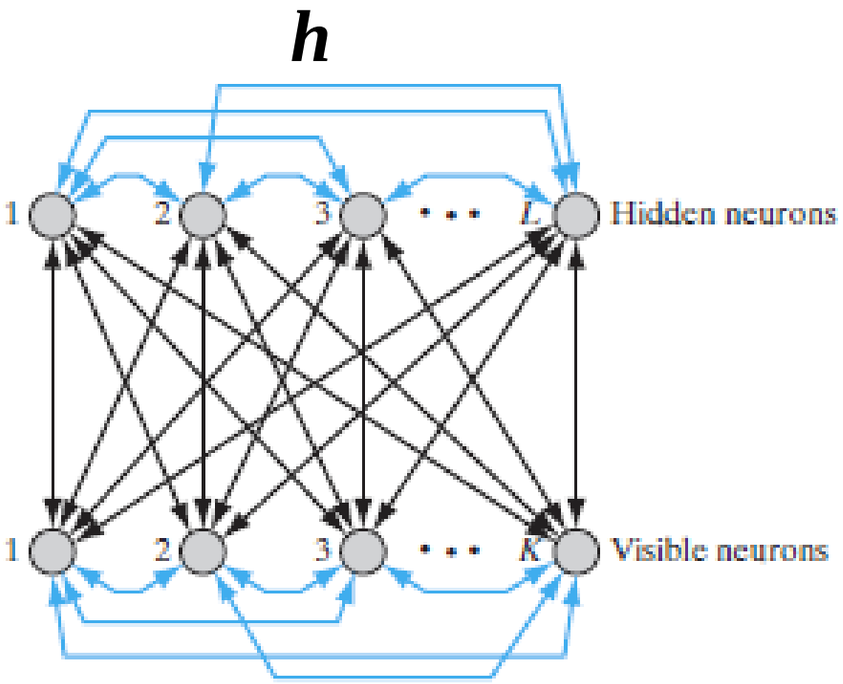

- Symmetric connections between visible, $\boldsymbol{v}$, and hidden neurons, $\boldsymbol{h}$
- Hidden neurons help account for higher-order correlations in the input vectors (data)
- Visible units provide interface to the external world – environment (data, $\boldsymbol{v}=\boldsymbol{x}$)
- Hidden units operate freely and are used to explain environmental input vectors

# Hidden and visible units




- Symmetric connections between visible, $\boldsymbol{v}$, and hidden neurons, $\boldsymbol{h}$
- Hidden neurons help account for higher-order correlations in the input vectors (data)
- Visible units provide interface to the external world – environment (data, $\boldsymbol{v}=\boldsymbol{x}$)
- Hidden units operate freely and are used to explain environmental input vectors

# Hidden and visible units




$$\boldsymbol{v}^{(p)} = \boldsymbol{x}^{(p)}$$

$$\Downarrow$$

$$\boldsymbol{y}^{(p)} = [\boldsymbol{x}^{(p)}, \boldsymbol{h}]$$

- Symmetric connections between visible, $\boldsymbol{v}$, and hidden neurons, $\boldsymbol{h}$
- Hidden neurons help account for higher-order correlations in the input vectors (data)
- Visible units provide interface to the external world – environment (data, $\boldsymbol{v}$=$\boldsymbol{x}$)
- Hidden units operate freely and are used to explain environmental input vectors
- Modelling a probability distribution (and hidden representation) by clamping patterns onto the visible units $\boldsymbol{v}^{(p_i)} = \boldsymbol{x}^{(p_i)}$

# Boltzmann learning

- The primary goal is to correctly model input patterns according to Boltzmann distribution
  - each input pattern is assumed to last long enough (it might have to be clamped for long) for the network to reach thermal equilibrium (converge) at temperature T
  - to reduce this time, simulated annealing is used with a sequence decreasing temperatures (from “hot” to “cold”)
- Essentially, hidden units learn probabilistically representation of data (seen through visible units)

# Boltzmann learning

- The idea is to maximise log-likelihood, $L(\mathbf{W})=\log\left(P(\mathbf{X})|\mathbf{W}\right)$

$$\Delta w_{ji} = \varepsilon \frac{\partial L(\mathbf{W})}{\partial w_{ji}} = \eta(\rho^{+}_{j,i} - \rho^{-}_{j,i}), \qquad \eta = \varepsilon / T$$

# Boltzmann learning

- The idea is to maximise log-likelihood, $L(\mathbf{W})=\log\left(P(\mathbf{X})|\mathbf{W}\right)$

$$\Delta w_{ji} = \varepsilon \frac{\partial L(\mathbf{W})}{\partial w_{ji}} = \eta(\rho^{+}_{j,i} - \rho^{-}_{j,i}), \quad \eta = \varepsilon / T$$

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \underbrace{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})}}_{\text{positive}} - \underbrace{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})}}_{\text{negative}} \right\}$$

**positive** phase (awake), with clamping, $\boldsymbol{v}^{(p)}=\boldsymbol{x}^{(p)}$

**negative** phase (sleep) free running

$$\left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})} = \sum_p \sum_h P\left(\boldsymbol{h} \mid \boldsymbol{v}=\boldsymbol{x}^{(p)}\right) y_i y_j$$

$$\left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})} = \sum_p \sum_y P(\boldsymbol{y}) y_i y_j$$

# Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})} - \left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})} \right\}$$

$$\Delta w_{i,j} \propto \left\langle y_i, y_j \right\rangle_{\text{data}} - \left\langle y_i, y_j \right\rangle_{\text{model}}$$

# Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})} - \left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})} \right\}$$

$$\Delta w_{i,j} \propto \left\langle y_i, y_j \right\rangle_{\text{data}} - \left\langle y_i, y_j \right\rangle_{\text{model}}$$

- **Positive** phase implies clamping the inputs (relative fast)

$$\left\langle y_i, y_j \right\rangle_{data}$$ ← Expected value at thermal equilibrium

- **Negative** phase involves updating all the units (can be very slow)

$$\left\langle y_i, y_j \right\rangle_{\text{model}}$$

# Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})} - \left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})} \right\}$$

$$\Delta w_{i,j} \propto \left\langle y_i, y_j \right\rangle_{\text{data}} - \left\langle y_i, y_j \right\rangle_{\text{model}}$$

- **Positive** phase implies clamping the inputs (relative fast)
- **Nega**

Expected value at thermal equilibrium

*Thermal equilibrium* does not imply only that the system settles down into the lowest energy state.

It is about the convergence of probability distribution over different configurations.

# Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})} - \left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})} \right\}$$

$$\Delta w_{i,j} \propto \left\langle y_i, y_j \right\rangle_{\text{data}} - \left\langle y_i, y_j \right\rangle_{\text{model}}$$

- **Positive** phase implies clamping the inputs (relative fast)

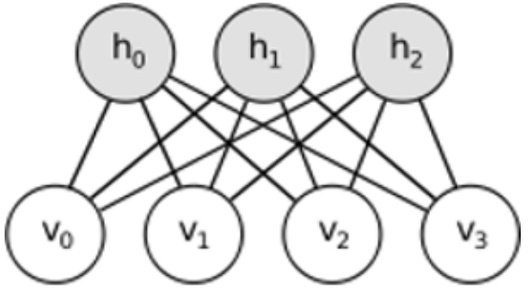
  *"Hebbian learning"* $\left\langle y_i, y_j \right\rangle_{data}$

- **Negative** phase involves updating all the units (can be very slow)

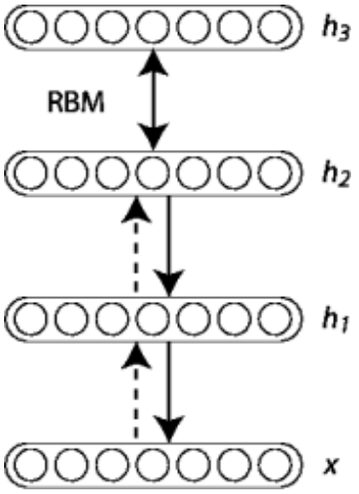
  *"Hebbian forgetting"* $\left\langle y_i, y_j \right\rangle_{\text{model}}$ prevent from learning false, spontaneously generated states

# Next step to decrease the complexity

- Restricted Boltzmann machines

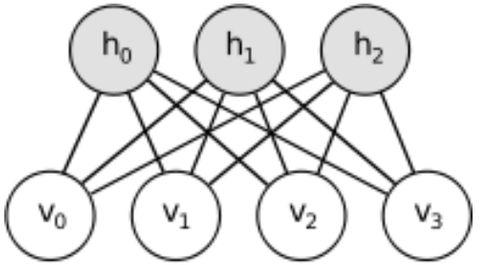



- Deep belief networks

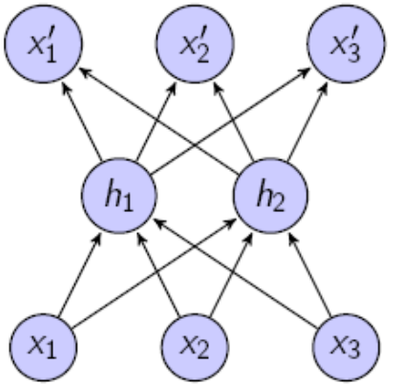


# Computational blocks for learning representations

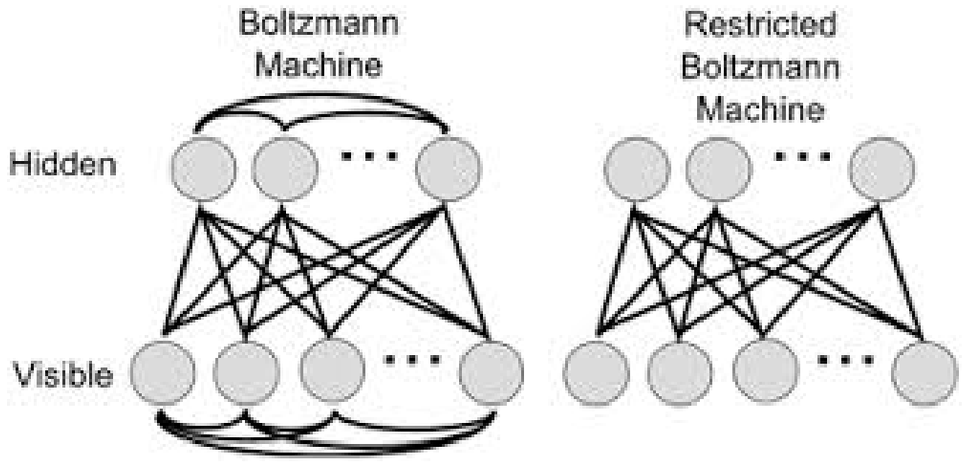
- Two key approaches to greedy layer-wise pretraining
  - regularized Boltzmann machines (RBMs)




  - autoencoders



# Restricted Boltzmann machine (RBM)

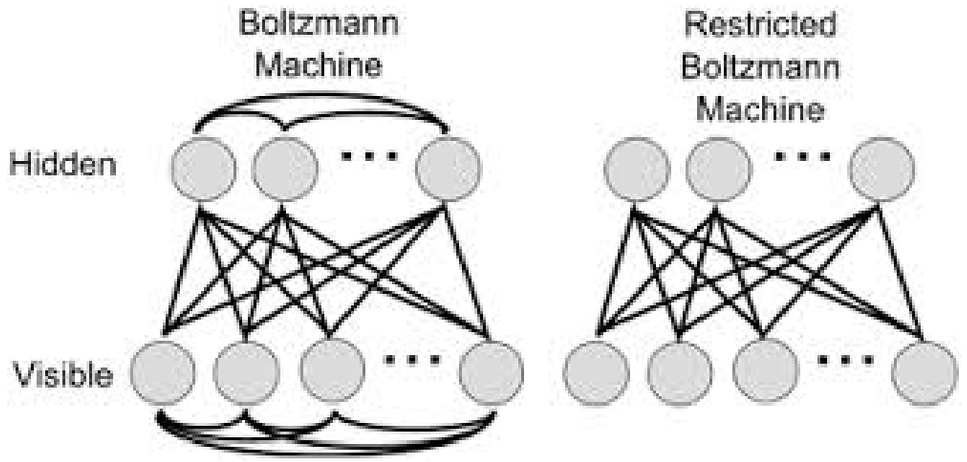



Visible and hidden units are conditionally independent given one another

$$p(\boldsymbol{h} \mid \boldsymbol{v}) = \prod_i p(h_i \mid \boldsymbol{v})$$

$$p(\boldsymbol{v} \mid \boldsymbol{h}) = \prod_j p(v_j \mid \boldsymbol{h})$$

# Restricted Boltzmann machine (RBM)




Visible and hidden units are conditionally independent given one another

$$p(\boldsymbol{h} \mid \boldsymbol{v}) = \prod_i p(h_i \mid \boldsymbol{v})$$

$$p(\boldsymbol{v} \mid \boldsymbol{h}) = \prod_j p(v_j \mid \boldsymbol{h})$$

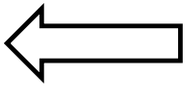
Following the same principle of maximising log likelihood by means of gradient ascent, one obtains:

$$\Delta w_{ji} = \varepsilon \frac{\partial L(\mathbf{W})}{\partial w_{ji}} = \varepsilon \left( \left\langle v_j h_i \right\rangle_{\text{data}} - \left\langle v_j h_i \right\rangle_{\text{model}} \right)$$

# Restricted Boltzmann machine (RBM)

$$P(h_i = 1 \mid \boldsymbol{v}) = \frac{1}{1 + \exp\left(-bias_{h_i} - \boldsymbol{v}^{\mathrm{T}} \mathbf{W}_{:,i}\right)}$$

$$P(v_j = 1 \mid \boldsymbol{h}) = \frac{1}{1 + \exp\left(-bias_{v_j} - \mathbf{W}_{j,:} \boldsymbol{h}\right)}$$



Visible and hidden units are conditionally independent given one another

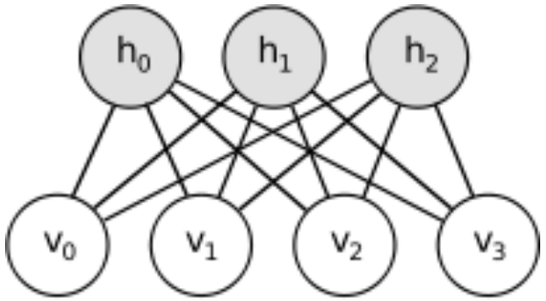
$$p(\boldsymbol{h} \mid \boldsymbol{v}) = \prod_i p(h_i \mid \boldsymbol{v})$$

$$p(\boldsymbol{v} \mid \boldsymbol{h}) = \prod_j p(v_j \mid \boldsymbol{h})$$

Following the same principle of maximising log likelihood by means of gradient ascent, one obtains:

$$\Delta w_{ji} = \varepsilon \frac{\partial L(\mathbf{W})}{\partial w_{ji}} = \varepsilon \left( \left\langle v_j h_i \right\rangle_{\text{data}} - \left\langle v_j h_i \right\rangle_{\text{model}} \right)$$

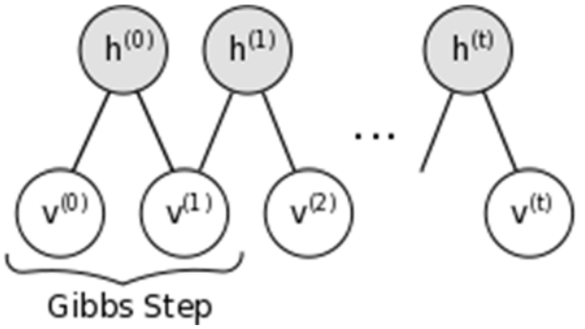
# RBM learning with Contrastive Divergence (CD)

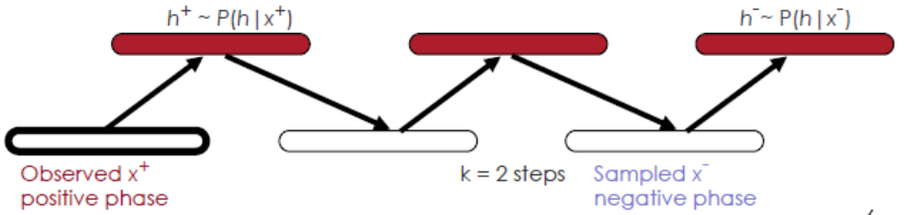


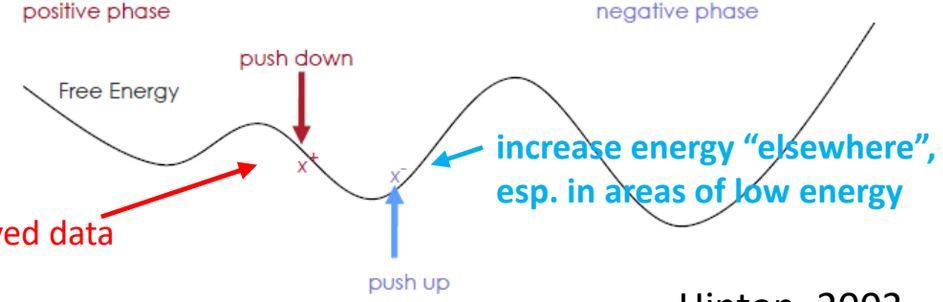

$$P(h_i = 1 \mid \boldsymbol{v}) = \frac{1}{1 + \exp\left(-bias_{h_i} - \boldsymbol{v}^{\mathrm{T}} \mathbf{W}_{:,i}\right)}$$

$$P(v_j = 1 \mid \boldsymbol{h}) = \frac{1}{1 + \exp\left(-bias_{v_j} - \mathbf{W}_{j,:}\, \boldsymbol{h}\right)}$$

Gibbs sampling



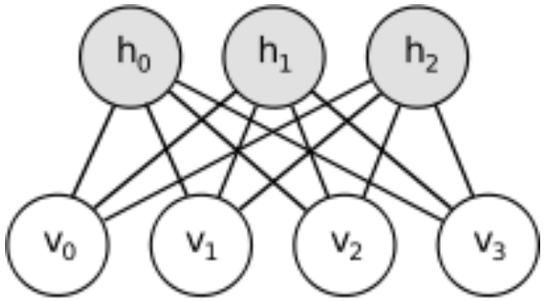




for the observed data

increase energy "elsewhere", esp. in areas of low energy

Hinton, 2003

# RBM learning with Contrastive Divergence (CD)



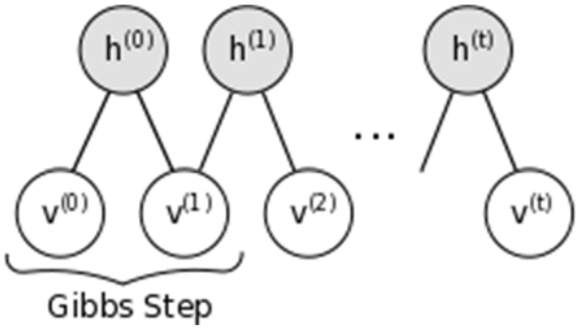

$$P(h_i = 1 \mid \boldsymbol{v}) = \frac{1}{1 + \exp\left(-bias_{h_i} - \boldsymbol{v}^{\mathrm{T}} \mathbf{W}_{:,i}\right)}$$

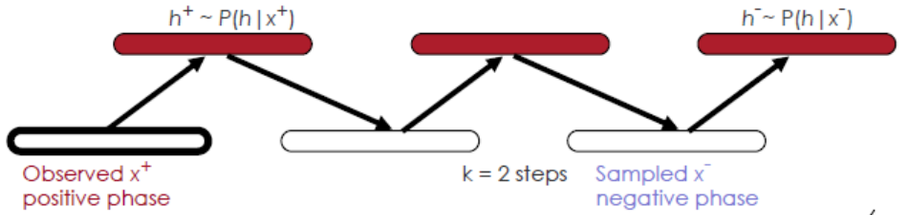
$$P(v_j = 1 \mid \boldsymbol{h}) = \frac{1}{1 + \exp\left(-bias_{v_j} - \mathbf{W}_{j,:}\, \boldsymbol{h}\right)}$$

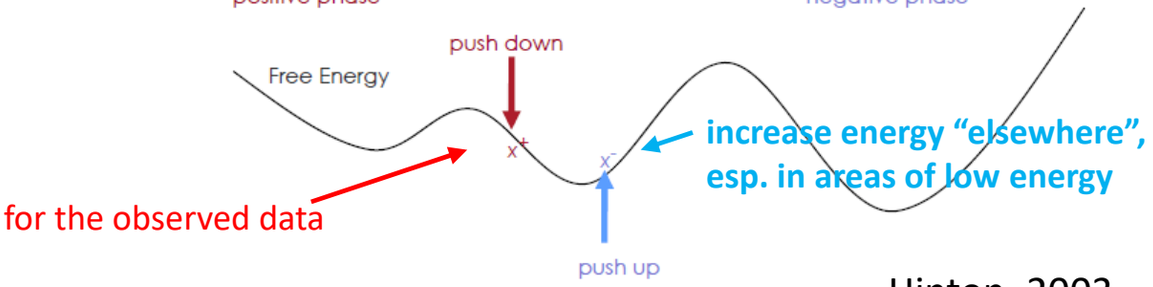
GOOD TO KNOW:

Contrastive Divergence does not optimise the likelihood but it works effectively!

Gibbs sampling



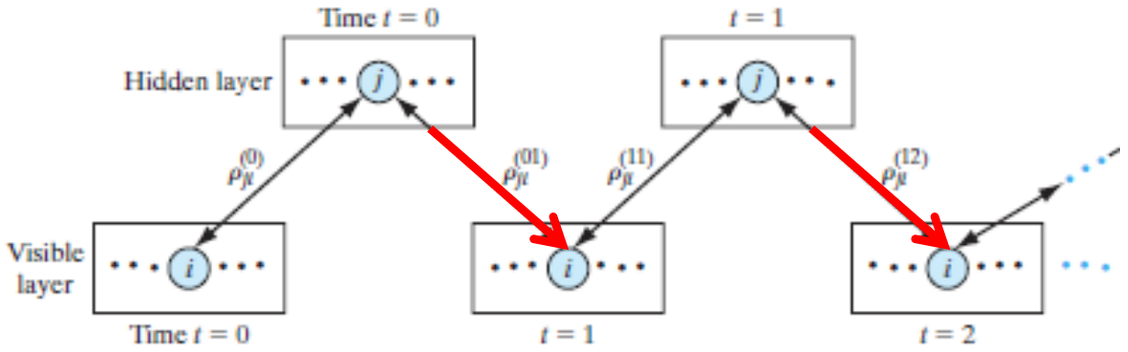




Hinton, 2003

# $CD_k$ recipe for training RBM

## Gibbs sampling




1) Clamp the visible units with an input vector and update hidden units.

$$P(h_i = 1 \mid \boldsymbol{v}) = \left(1 + \exp\left(-bias_{h_i} - \boldsymbol{v}^{\mathrm{T}} \mathbf{W}_{:,i}\right)\right)^{-1}$$

2) Update all the visible units in parallel to get a **reconstruction**.

$$P(v_j = 1 \mid \boldsymbol{h}) = \left(1 + \exp\left(-bias_{v_j} - \mathbf{W}_{j,:}\, \boldsymbol{h}\right)\right)^{-1}$$

3) Collect the statistics for correlations after *k* steps using mini-batches and update weights:

$$\Delta w_{j,i} = \frac{1}{N} \sum_{n=1}^{N} \left( v_j^{(n)} h_i^{(n)} - \hat{v}_j^{(n)} \hat{h}_i^{(n)} \right)$$

# From RBM to Gaussian-Bernoulli RBM

Bernoulli-Bernoulli (binary-binary)

$$p(v_i = 1|\mathbf{h}) = g\left(\sum_j W_{ij} b_j + b_i\right)$$

$$p(b_j = 1|\mathbf{v}) = g\left(\sum_i W_{ij} v_i + a_j\right)$$

Gaussian-Bernoulli (real/cont.-binary)

$$p(v_i = x|\mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{\left(x - b_i - \sigma_i \sum_j b_j W_{ij}\right)^2}{2\sigma_i^2}\right),$$

$$p(b_j = 1|\mathbf{v}) = g\left(b_j + \sum_i W_{ij} \frac{v_i}{\sigma_i}\right),$$

Visible units are real-valued whereas hidden units remain binary.

Salakhutdinov, 2015

# From RBM to Gaussian-Bernoulli RBM

Bernoulli-Bernoulli (binary-binary)

$$p(v_i = 1|\mathbf{h}) = g\left(\sum_j W_{ij} b_j + b_i\right)$$

$$p(b_j = 1|\mathbf{v}) = g\left(\sum_i W_{ij} v_i + a_j\right)$$

Gaussian-Bernoulli (real/cont.-binary)

$$p(v_i = x|\mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{\left(x - b_i - \sigma_i \sum_j b_j W_{ij}\right)^2}{2\sigma_i^2}\right),$$

$$p(b_j = 1|\mathbf{v}) = g\left(b_j + \sum_i W_{ij} \frac{v_i}{\sigma_i}\right),$$

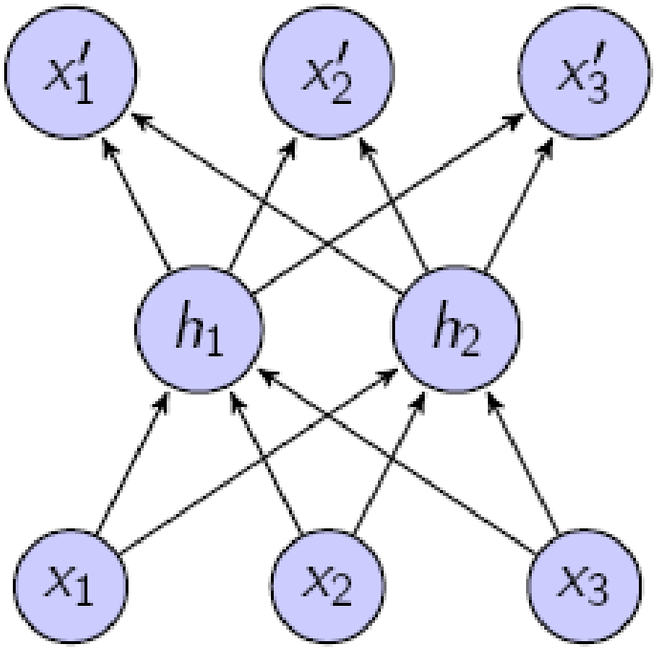
Visible units are real-valued whereas hidden units remain binary.

The derivative of the log-likelihood:

$$\frac{\partial \log P(\mathbf{v};\theta)}{\partial W_{ij}} = \mathbb{E}_{P_{\text{data}}}\left[\frac{1}{\sigma_i} v_i b_j\right] - \mathbb{E}_{P_{\text{model}}}\left[\frac{1}{\sigma_i} v_i b_j\right]$$

Salakhutdinov, 2015

# Autoencoders – principles



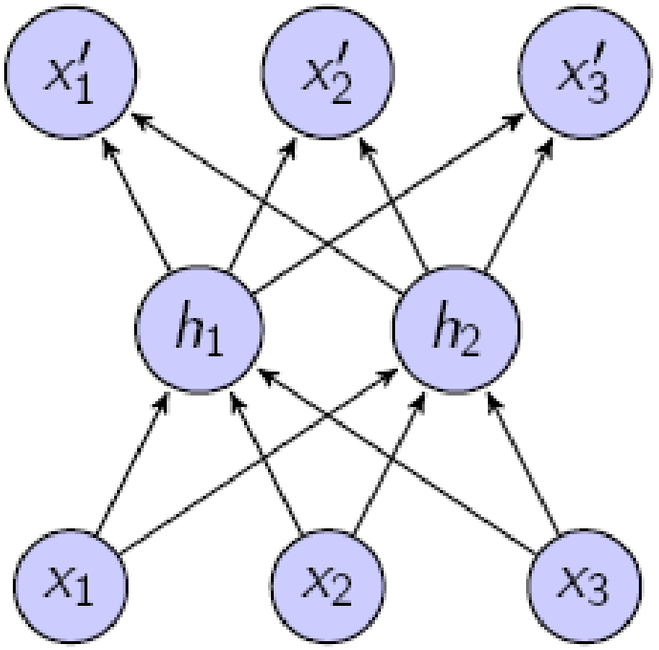

Decoder: $x' = f(x)$

Encoder: $h = g\big(f(x)\big)$

The idea is to minimise the loss function, $L$:

$$L\big(x, g\big(f(x)\big)\big)$$

# Autoencoders




Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Encourage $h$ to give small reconstruction error:

- e.g. $Loss = \sum_m ||x^{(m)} - DECODER(ENCODER(x^{(m)}))||^2$
- Reconstruction: $x' = \sigma(W'\sigma(Wx + b) + d)$

# Different types of autoencoders

- Undercomplete autoencoders
  - hidden layer is smaller than the input dimensionality
- Overcomplete regularised autoencoders
  - Larger hidden layer size with the regularisation (to avoid overfitting and copying input to the output)

$$L\Big(x, g\big(f(x)\big)\Big) + \Omega(h), \quad h = f(x)$$

  - Sparse autoencoders, denosing autoencoders
  - Deep autoencoders

# Sparse autoencoders

- Penalizing non-sparse solutions can be seen as adding latent variables with a prior and maximising likelihood

$$\log p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x}) = \log p_{\text{model}}(\boldsymbol{h}) + \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$$

*for example:* $$p_{\text{model}}(\boldsymbol{h}) = \prod_i \frac{\lambda}{2} e^{-\lambda |h_i|} \Rightarrow \quad \Omega(\boldsymbol{h}) = \lambda \sum |h_i|$$

# Denoising autoencoders

$$L\Big(\boldsymbol{x}, g\big(f(\tilde{\boldsymbol{x}})\big)\Big), \quad h = f(\tilde{\boldsymbol{x}})$$

Corrupted copy of $x$

Autoencoders have to undo this corruption beyond simply coping the input.

# Denoising autoencoders

$$L\Big(\boldsymbol{x}, g\big(f(\tilde{\boldsymbol{x}})\big)\Big), \quad h = f(\tilde{\boldsymbol{x}})$$

Corrupted copy of $x$

Autoencoders have to undo this corruption beyond simply coping the input.

1. A training sample is sampled from the training data.
2. A corrupted version of the sample $\boldsymbol{x}$ is drawn from some corruption process
$$C(\tilde{\boldsymbol{x}} \mid \boldsymbol{x} = \boldsymbol{s})$$
3. $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ is used as a training sample to estimate the autoencoder's reconstruction distribution $p_{reconstruction}(\tilde{\boldsymbol{x}} \mid \boldsymbol{x}) = p_{decoder}(\boldsymbol{x} \mid \boldsymbol{h}), \quad \boldsymbol{h} = f(\tilde{\boldsymbol{x}})$

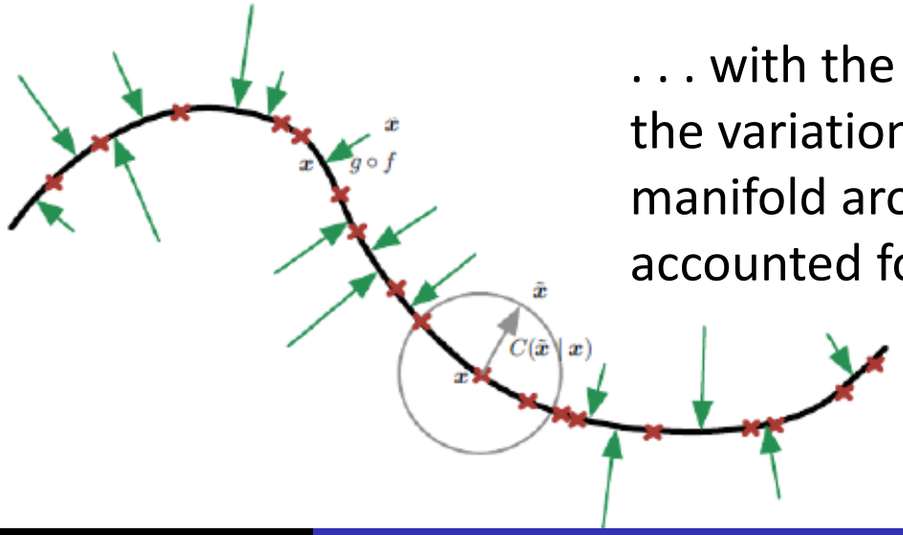
# Denoising autoencoders

$$L\left(\boldsymbol{x}, g\left(f(\tilde{\boldsymbol{x}})\right)\right), \quad h = f(\tilde{\boldsymbol{x}})$$

Corrupted copy of $x$

Autoencoders have to undo this corruption beyond simply coping the input.

Learning a *vector field* around a *low-dimensional manifold* . . .




. . . with the principle that only the variations tangent to the manifold around ***x*** should be accounted for by changes in ***h***

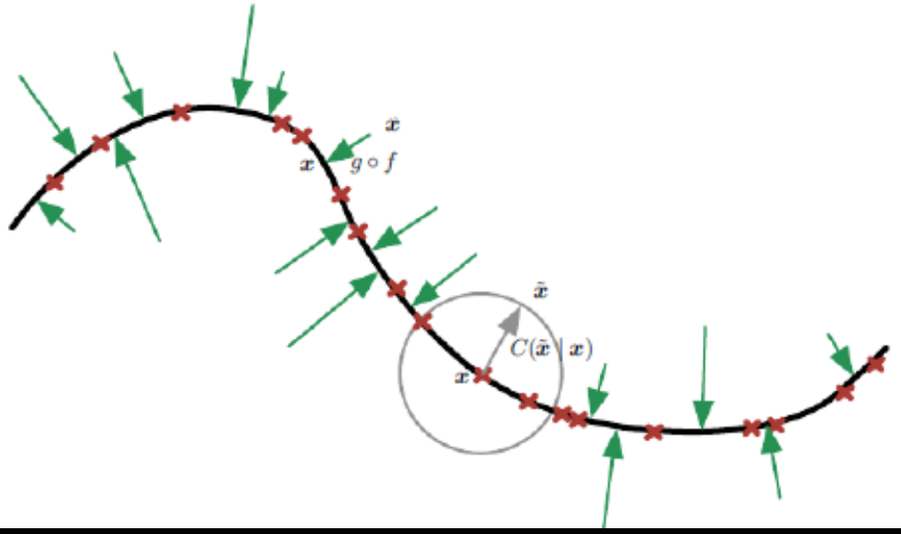
Goodfellow et al.

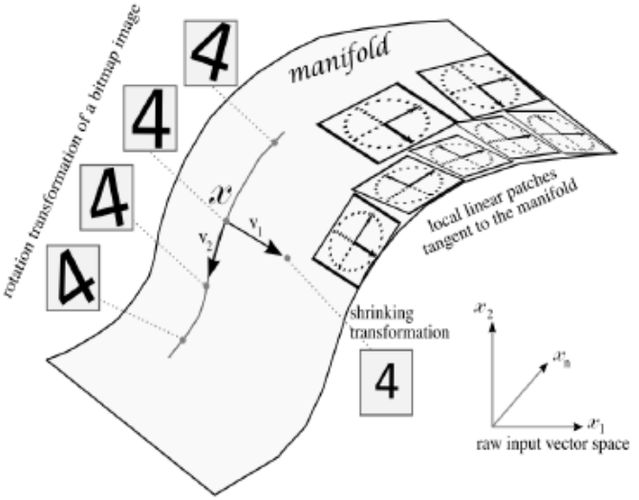
# Denoising autoencoders

When training autoencoders there is a **compromise**

I. Need to approximately recover ***x*** – *reconstruction* force

II. Need to satisfy the regularization term – *regularisation* force.




Goodfellow et al.