Machine Learning for Computational Linguistics

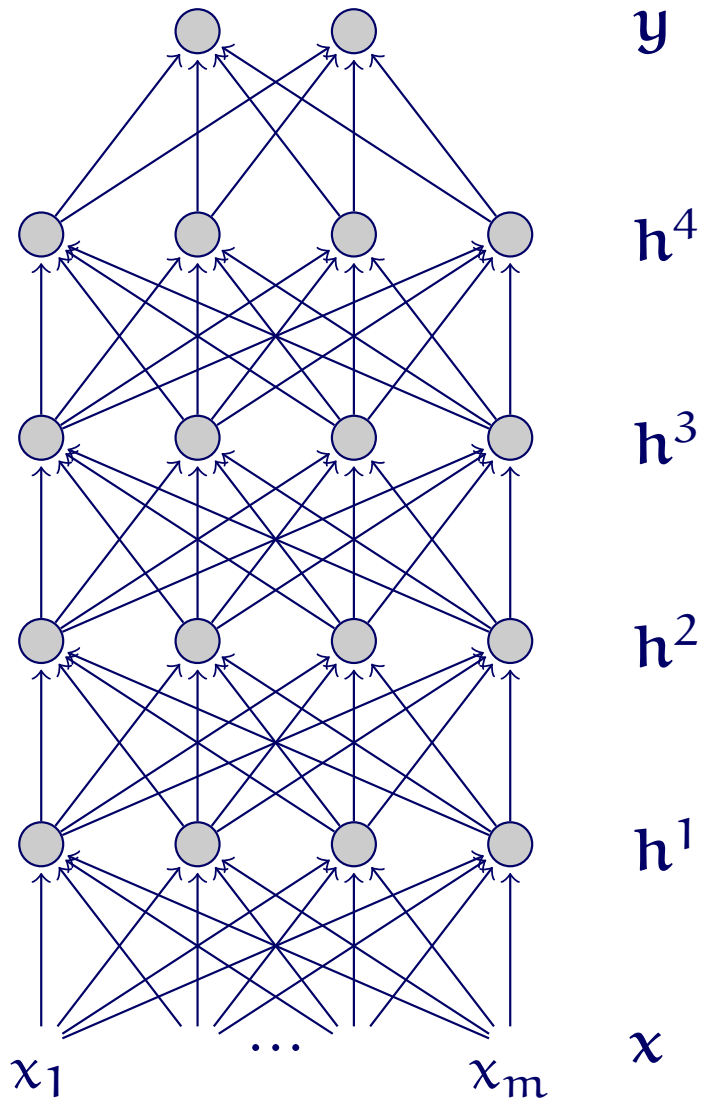
Autoencoders + deep learning summary

Çağrı Çöltekin

University of Tübingen
Seminar für Sprachwissenschaft

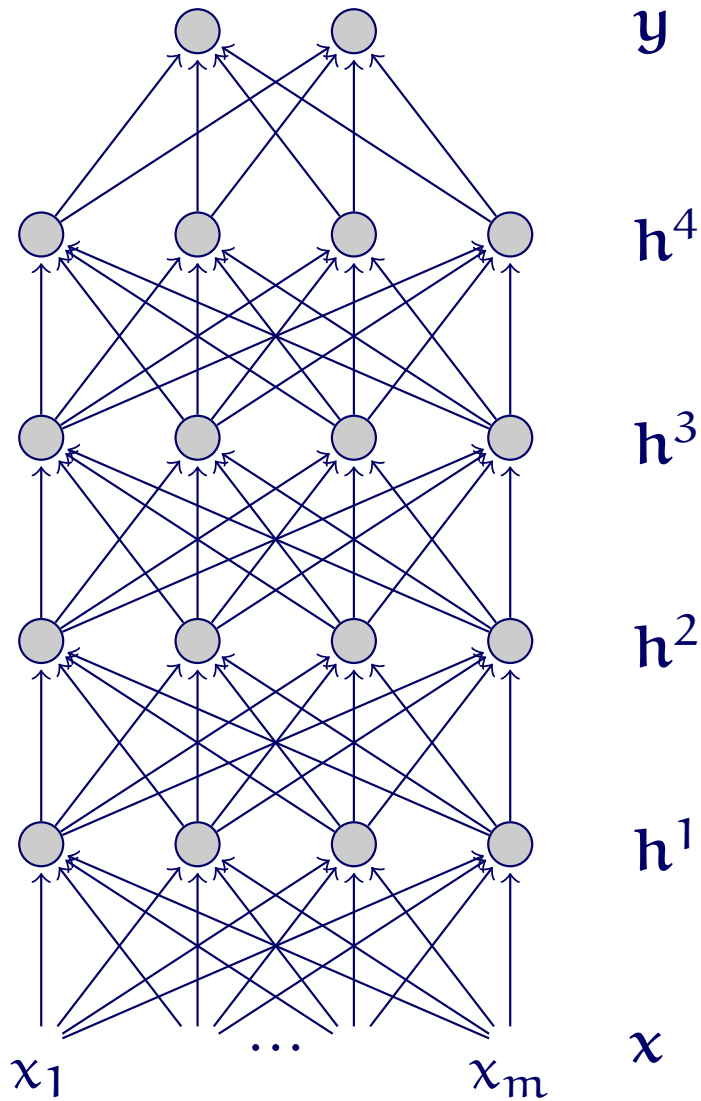
July 12, 2016

(Deep) neural networks so far



- $\mathbf{x}$ is the input vector
- $\mathbf{y}$ is the output vector
- $\mathbf{h}^1 \ldots \mathbf{h}^m$ are the hidden layers (learned/useful representations)
- The network can be fully connected, or may can use sparse connectivity
- The connections can be feed-forward, or may include recurrent links

(Deep) neural networks so far



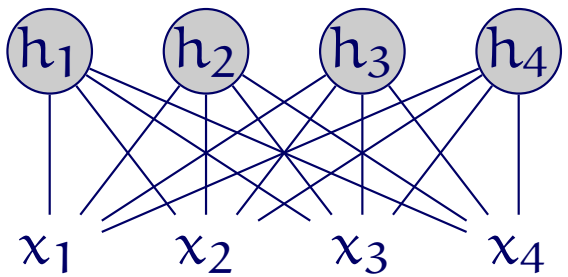
- $\mathbf{x}$ is the input vector
- $\mathbf{y}$ is the output vector
- $\mathbf{h}^1 \ldots \mathbf{h}^m$ are the hidden layers (learned/useful representations)
- The network can be fully connected, or may can use sparse connectivity
- The connections can be feed-forward, or may include recurrent links

So far, we only studied *supervised* models

Unsupervised learning in ANNs

- Restricted Boltzmann machines (RBM)
  similar to the latent variable models (e.g., Gaussian mixtures), consider the representation learned by hidden layers as hidden variables ($\mathbf{h}$), and learn $p(\mathbf{x}, \mathbf{h})$ that maximize the probability of the (unlabeled)data
- Autoencoders
  train a constrained feed-forward network to predict its output

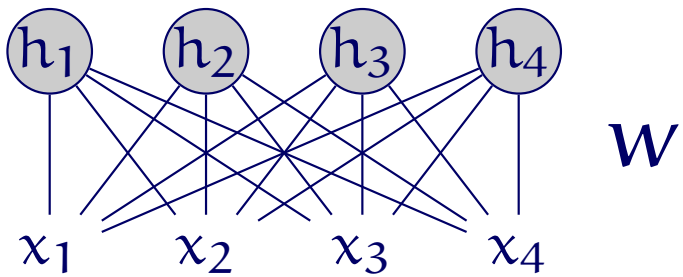
Restricted Boltzmann machines (RBMs)



- RBMs are unsupervised latent variable models, they learn only from unlabeled data
- They are generative models of the joint probability $p(\mathbf{h}, \mathbf{x})$
- They correspond to undirected graphical models
- No links within layers
- The aim is to learn useful features ($\mathbf{h}$)

* As usual, biases are omitted from the diagrams and the formulas.

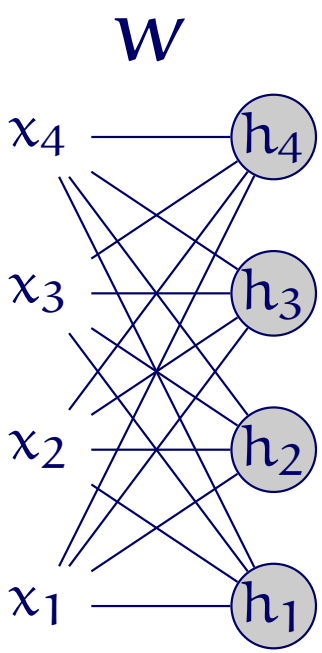
Restricted Boltzmann machines (RBMs)



- RBMs are unsupervised latent variable models, they learn only from unlabeled data
- They are generative models of the joint probability $p(\mathbf{h}, \mathbf{x})$
- They correspond to undirected graphical models
- No links within layers
- The aim is to learn useful features ($\mathbf{h}$)

* As usual, biases are omitted from the diagrams and the formulas.

The distribution defined by RBMs



$$p(\mathbf{h}, \mathbf{x}) = \frac{e^{h^T W x}}{Z}$$

which is intractable (Z is difficult to calculate).
But conditional distributions are easy to calculate

$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x}) \quad = \frac{1}{1 + e^{W_j x}}$$

$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h}) \quad = \frac{1}{1 + e^{W_k^T h}}$$

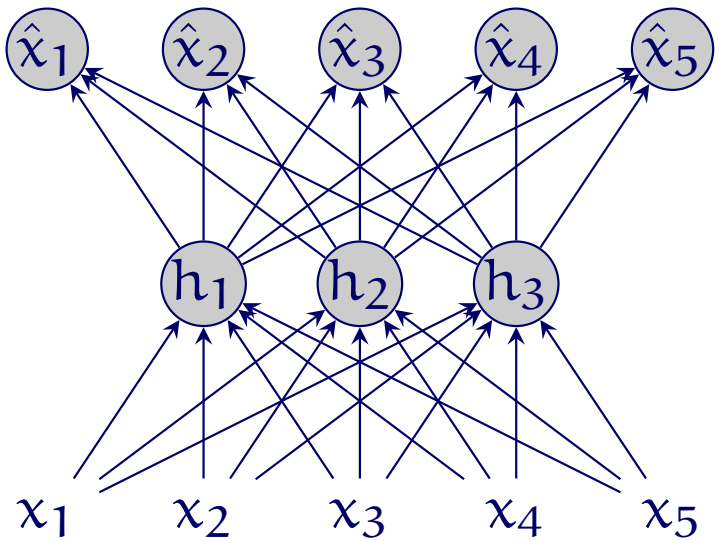
Learning in RBMs: contrastive divergence algorithm

- We want to maximize the probability that the model assigns to the input, $p(x)$, or equivalently minimize $-\log p(x)$
- In general, this is not tractable. But efficient approximate algorithms exist

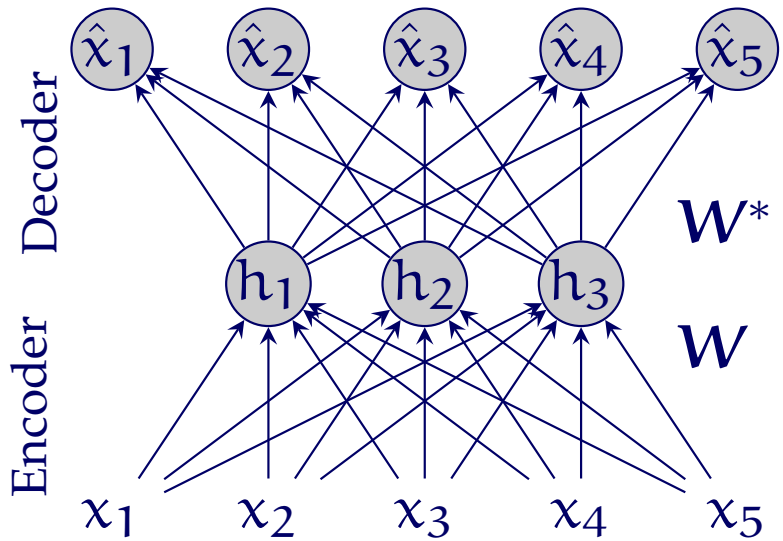
Contrastive divergence algorithm

1. Given a training example $\mathbf{x}$, calculate the probabilities of hidden units, and sample a hidden activation, $\mathbf{h}$, from this distribution
2. Sample a *reconstruction*, $\mathbf{x}'$ from $p(\mathbf{x}|\mathbf{h})$, and re-sample $\mathbf{h}'$ using $\mathbf{x}'$
3. Set the update rule to $\Delta w_{ij} = (x_i v_j - x'_i h'_j)\epsilon$

Autoencoders

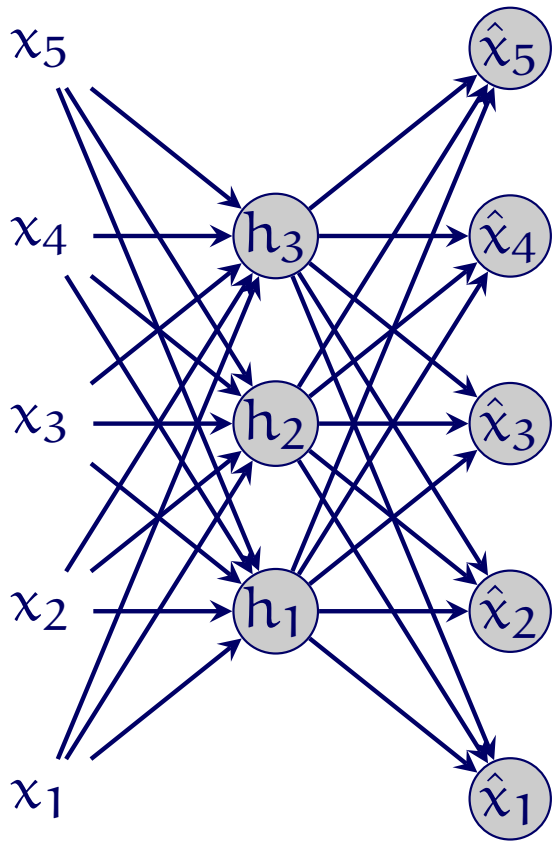


Autoencoders



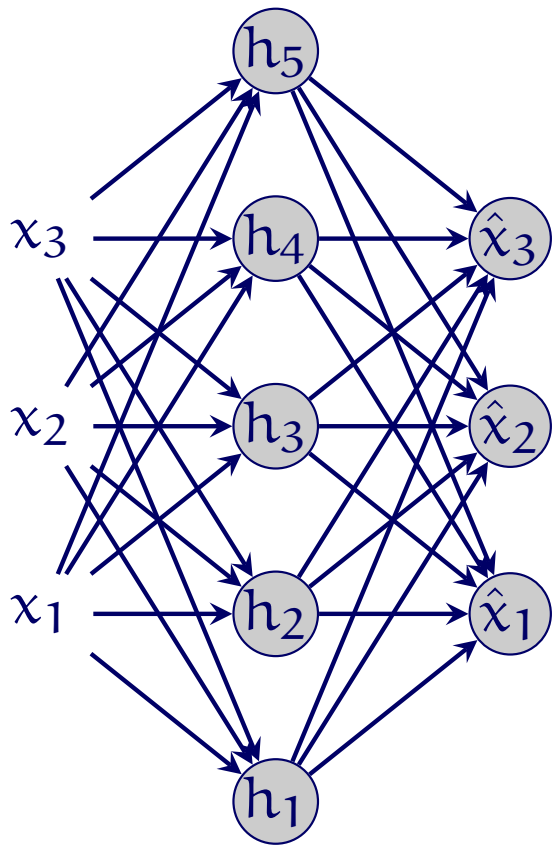
- Autoencoders are standard feed-forward networks
- The main difference is that they are trained to predict their input (they try to learn the identity function)
- The aim is to learn useful representations of input at the hidden layer
- Typically weights are tied ($W^* = W^\top$)

Under-complete autoencoders



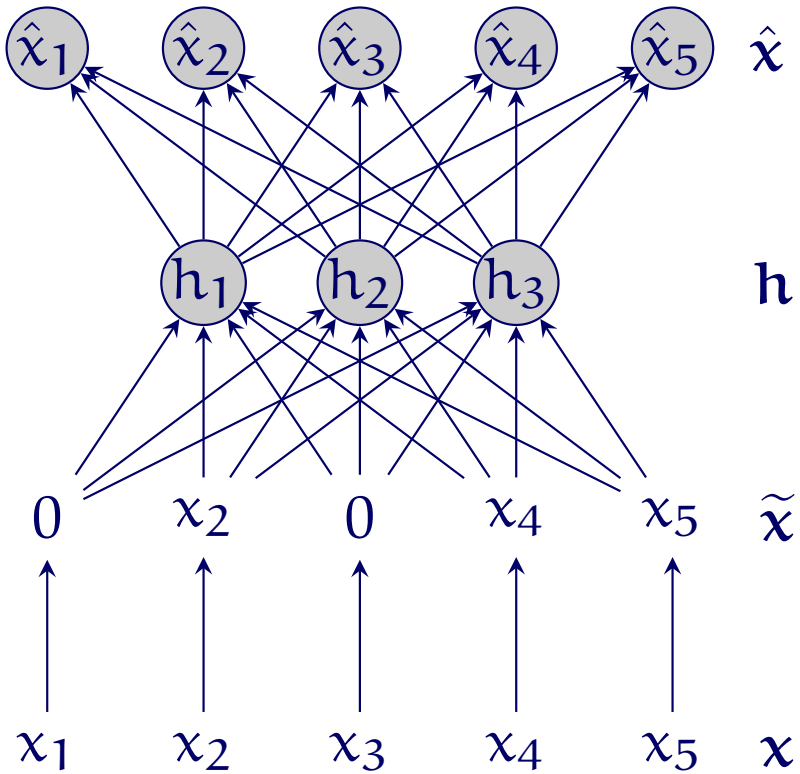
- An autoencoder is said to be *under-complete* if there are fewer hidden units than inputs
- The network is forced to learn a more compact representation of the input (compress)
- An autoencoder with a single hidden layer is equivalent to PCA
- We need multiple layers for learning non-linear features

Over-complete autoencoders



- An autoencoder is said to be *over-complete* if there are more hidden units than inputs
- The network can normally memorize the input perfectly
- This type of networks are useful if trained with a regularization term resulting in sparse hidden units (e.g., L1 regularization)

Denoising autoencoders



- Instead of providing the exact input we introduce noise by
  - randomly setting some inputs to 0 (dropout)
  - adding random (Gaussian) noise
- Network is still expected to reconstruct the original input (without noise)

Learning manifolds

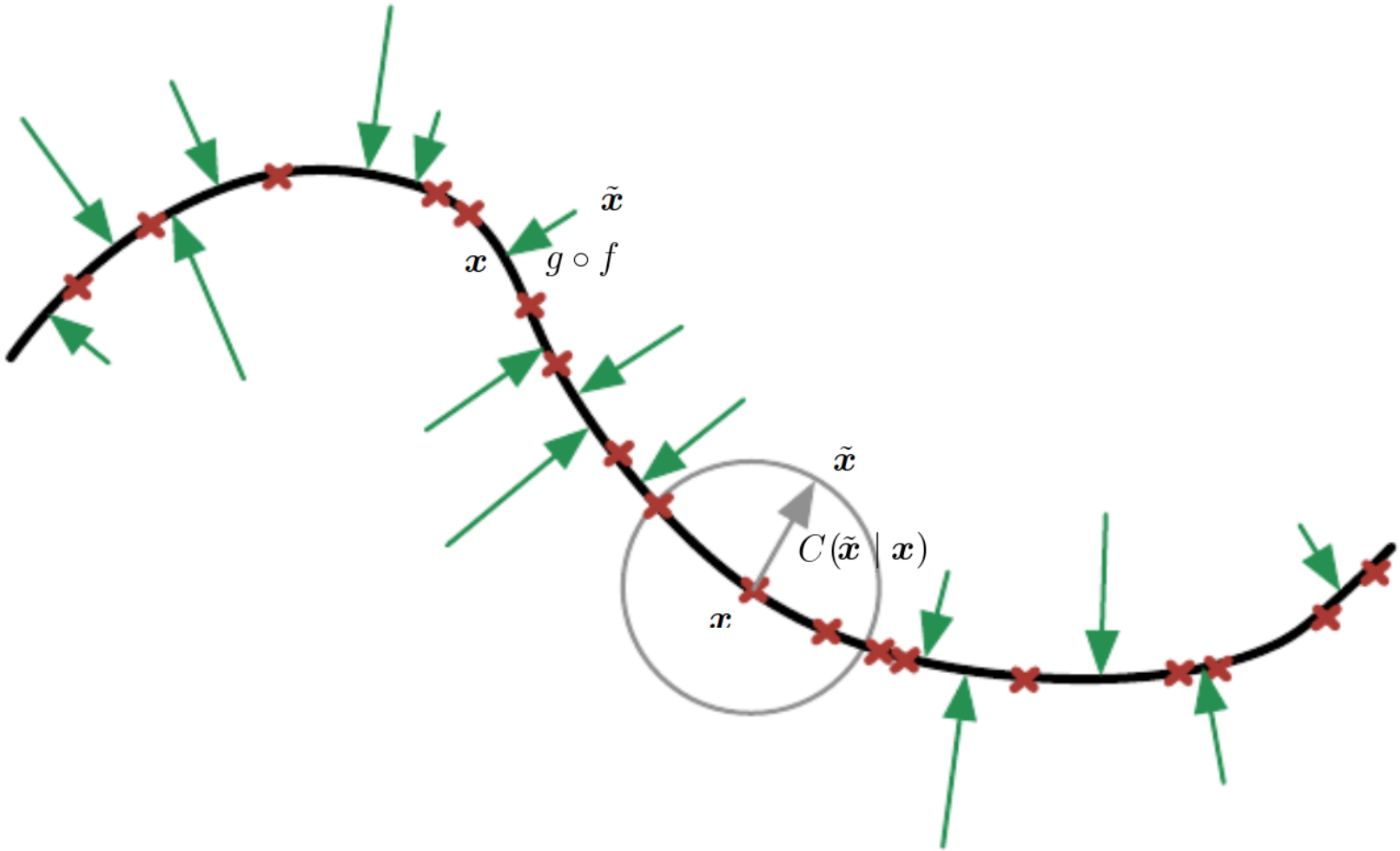


Figure: Goodfellow et al. (2016)

Unsupervised pre-training

Deep belief networks or stacked autoencoders

- A common use case for RBMs and autoencoders are as pre-training methods for supervised networks
- Autoencoders or RBMs are trained using unlabeled data
- The weights learned during the unsupervised learning is used for initializing the weights of a supervised network
- This approach has been one of the reasons for success of deep networks

Deep unsupervised learning

- Both autoencoders and RBMs can be 'stacked'
- Learn the weights of the first hidden layer from the data
- Freeze the weights, and using the hidden layer activations as input, train another hidden layer, …
- This approach is called *greedy layer-wise training*
- In case of RBMs resulting networks are called *deep belief networks*
- Deep autoencoders are called *stacked autoencoders*

Why use pre-training?

- Pre-training does not require labeled data
- It can be considered as a form of regularization
- Unsupervised methods may reduce the dimensionality, allowing in efficient computation for the supervised phase
- Unsupervised learning on large-scale data may find the manifold that contains input data, counteracting curse of dimensionality