# Deep Neural Networks

## DT2118 Speech and Speaker Recognition

Giampiero Salvi

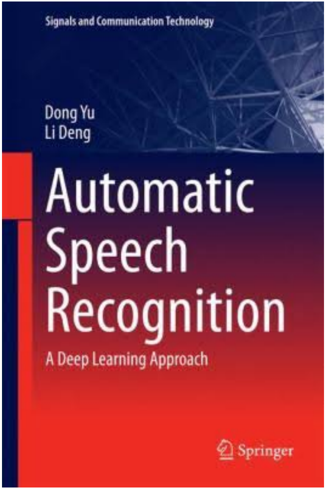
KTH/CSC/TMH giampi@kth.se

VT 2016

# Literature

D. Yu and L. Deng. *Automatic Speech Recognition, a Deep Learning Approach*. Springer, 2015

Available in PDF through KTH Library



# Outline

- State-to-Output Probability Model
- Artificial Neural Networks
  - Perceptron
  - Multi Layer Perceptron
  - Error Backpropagation
  - Hybrid HMM-MLP
- Deep Learning (Initialization)
  - Deep Neural Networks
  - Restricted Boltzmann Machines
  - Deep Belief Networks

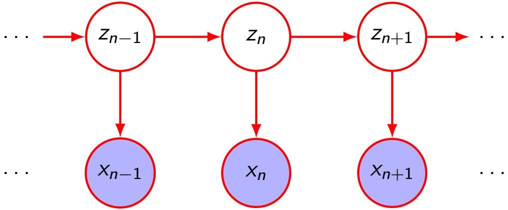
# Outline

State-to-Output Probability Model

Artificial Neural Networks
- Perceptron
- Multi Layer Perceptron
- Error Backpropagation
- Hybrid HMM-MLP

Deep Learning (Initialization)
- Deep Neural Networks
- Restricted Boltzmann Machines
- Deep Belief Networks

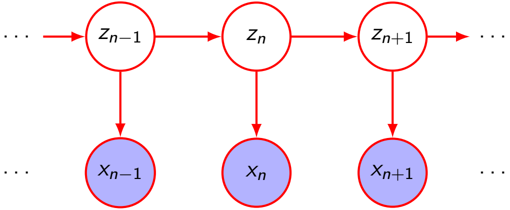
# State-to-Output Probability Model




is responsible for the discriminative power of the whole model

- GMMs used because easy to train and adapt
- discriminative training can improve results

# State-to-Output Probability Model




is responsible for the discriminative power of the whole model

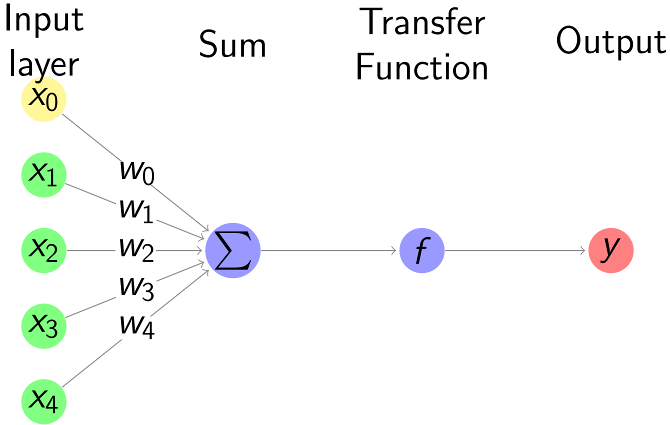
Alternatives:

- artificial neural networks (ANNs)
- deep neural networks (DNNs)
- support vector machines (SVMs) not used for ASR

# Outline

# Perceptron

Known since the 1950's [7]




[7] F. Rosenblatt. *The perceptron: A perceiving and recognizing automaton*. Tech. rep. 85-460-1. Cornell Aeronautical Laboratory, 1957

# Perceptron input/output

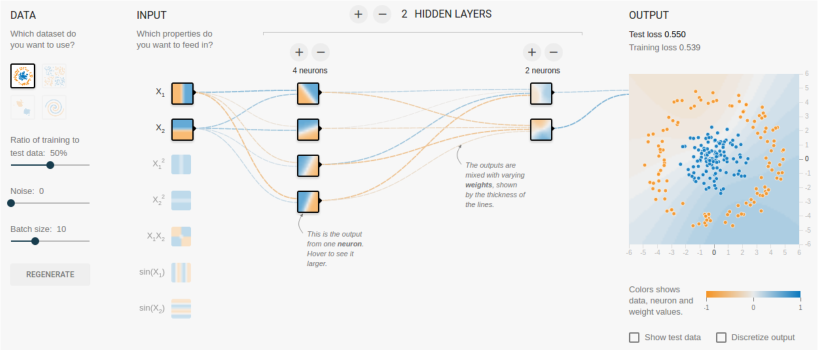
$$y = f\left(b + \sum_i w_i x_i\right)$$

where

$$\begin{aligned}
f(z) &= \frac{1}{1+e^{-z}} \quad \text{sigmoid}\\
f(z) &= \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad \text{hyperbolic tangent}\\
f(z) &= \max(0, z) \quad \text{rectified linear unit}
\end{aligned}$$

Equivalent to logistic regression ($b = w_0 x_0$ bias)

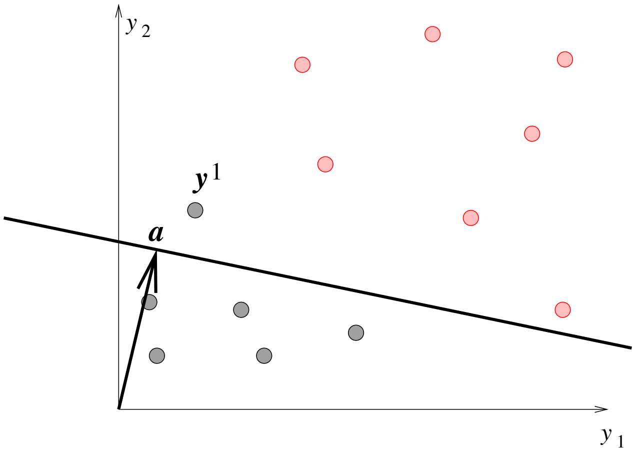
# Preceptron: Illustration



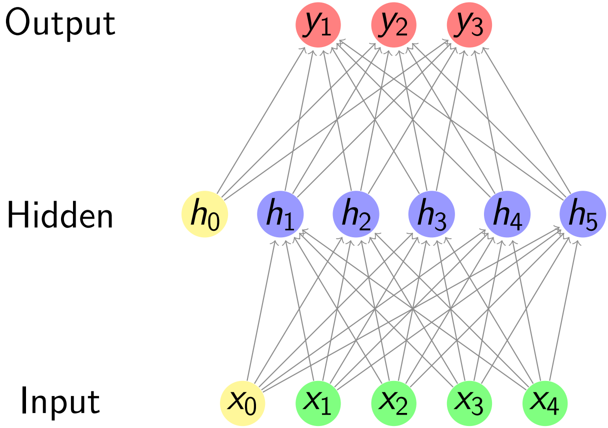

http://playground.tensorflow.org/

# Perceptron: Linear Classification

Learning adjust weights to correct errors



# Multi-layer Perceptron [6]




[6] F. Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms.* Tech. rep. DTIC Document, 1961

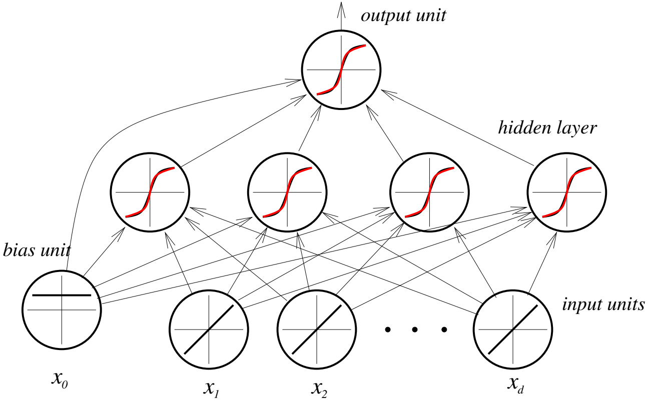
# Universal Approximation Theorem

- First proposed by Gybenko [3]
- one single hidden layer and finite but appropriate number of neurons
- can approximate any function in $\mathbb{R}^N$ with mild constraints

[3] G. Gybenko. "Approximation by superposition of sigmoidal functions". In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314

# Multi-layer Perceptron: Training

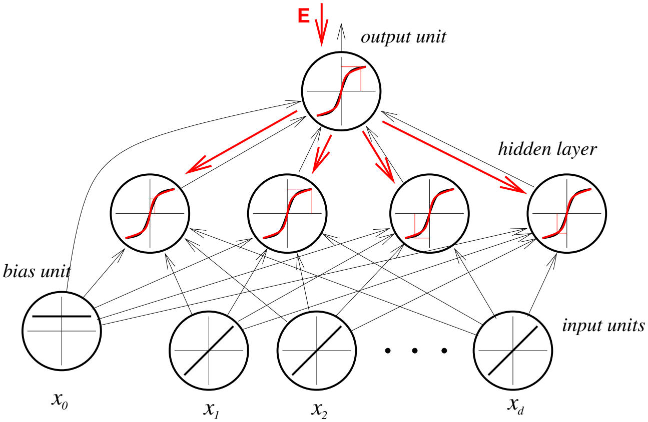
## Backpropagation algorithm [8]




[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. Tech. rep. DTIC Document, 1985

# Multi-layer Perceptron: Training

## Backpropagation algorithm [8]




[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. Tech. rep. DTIC Document, 1985

# Learning Criteria

Ideally minimise Expected Loss:

$$J_{\text{EL}} = \mathbb{E}\big[J(W, B, o, y)\big] = \int_o J(W, B, o, y) p(o) do$$

where $o$ = features, $y$ = labels

but we do not know $p(o)$

Use empirical learning criteria instead:

- Mean Square Error (MSE)
- Cross Entropy (CE)

# Mean Square Error Criterion

$$J_{\mathrm{MSE}} = \frac{1}{M} \sum_{m=1}^{M} J_{\mathrm{MSE}}(W, B, o^m, y^m)$$

$$\begin{aligned} J_{\mathrm{MSE}}(W, B, o^m, y^m) &= \frac{1}{2} \left\| v^L - y \right\|^2 \\ &= \frac{1}{2} \left( v^L - y \right)^T \left( v^L - y \right) \end{aligned}$$

# Cross Entropy Criterion

$$J_{\mathrm{CE}} = \frac{1}{M}\sum_{m=1}^{M} J_{\mathrm{CE}}(W, B, o^m, y^m)$$

$$J_{\mathrm{CE}}(W, B, o^m, y^m) = -\sum_{i=1}^{C} y_i \log v_i^L$$

Equivalent to minimising Kullback-Leibler divergence (KLD)

# Update rules

$$\begin{aligned} W_{t+1}^{l} &\leftarrow W_t^l - \epsilon \Delta W_t^l \\ b_{t+1}^{l} &\leftarrow b_t^l - \epsilon \Delta b_t^l \end{aligned}$$

To compute $\Delta W_t^l$ and $\Delta b_t^l$ we need the gradient of the criterion function.
Key trick: chain rule of gradients $f(g(x))$:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

# Backpropagation: Properties

- weights only depend on neighbouring variables
- algorithm finds local optimum
- sensitive to initialisation

# Practical Issues

- preprocessing: Cepstral Mean Normalisation
- initialisation: random (symmetry breaking), linear range of activation function
- regularisation (weight decay, dropout)
- batch size selection
- sample randomisation
- momentum
- learning rate and stopping criterion

# Output Layer

Regression tasks: Linear layer

$$v^L = z^L = W^L v^{L-1} + b^L$$

Classification tasks: Softmax layer

$$v_i^L = \mathrm{softmax}_i(z^L) = \frac{e^{z_i^L}}{\sum_{j=1}^{C} e^{z_j^L}}$$

# Probabilistic Interpretation

1. $v_i^L \in [0, 1] \quad \forall i$
2. $\sum_{j=1}^{C} v_i^L = 1$

Output activations are posterior probabilities of the classes given the observations

$$v_i^L = P(i|o)$$

In speech: $P(\text{state}|\text{sounds})$

# Hybrid HMM+Multi Layer Perceptron

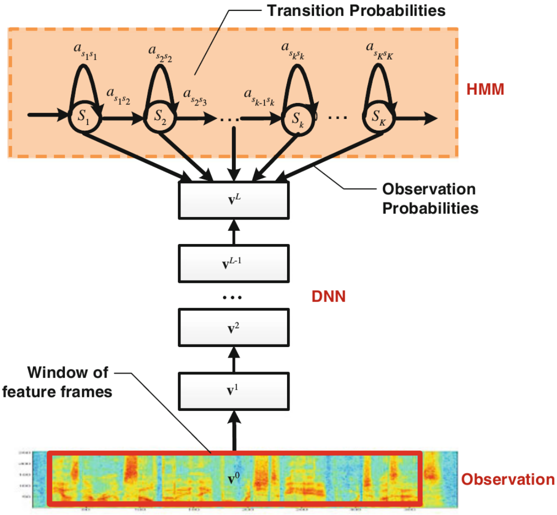



Figure from Yu and Deng

# Combining probabilities [1]

- HMMs use likelihoods $P(\text{sound}|\text{state})$
- MLPs and DNNs estimate posteriors $P(\text{state}|\text{sound})$

We can combine with Bayes:

$$P(\text{sound}|\text{state}) = \frac{P(\text{state}|\text{sound})P(\text{sound})}{P(\text{state})}$$

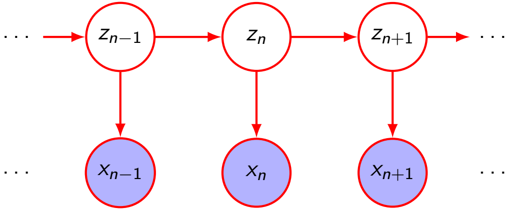
- $P(\text{state})$ can be estimated from the training set
- $P(\text{sound})$ is constant and can be ignored

Use scaled likelihoods:

$$\bar{P}(\text{sound}|\text{state}) = \frac{P(\text{state}|\text{sound})}{P(\text{state})}$$

[1] H. Bourland and C. J. Wellekens. "Links Between Markov Models and Multilayer Perceptrons". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 12.12 (1990)

# State-to-Output Probability Model




Use ANNs for $P(x_n|z_n)$

# Time-Delayed NNs [11]

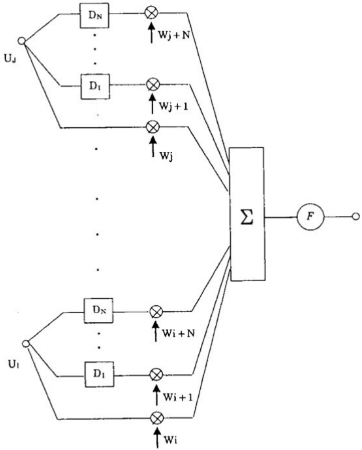
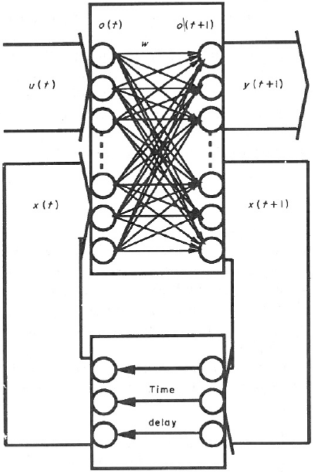



Fig. 1. A Time-Delay Neural Network (TDNN) unit.

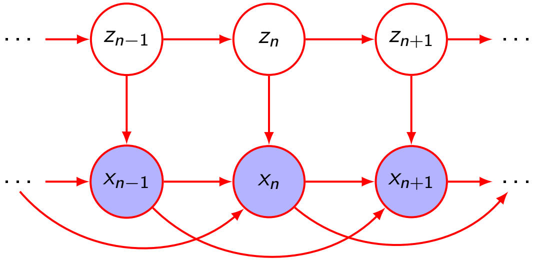

[11] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. "Phoneme Recognition Using Time-Delay Neural Networks". In: *IEEE Trans. Acoust., Speech, Signal Process.* 37.3 (1989)

# Recurrent ANNs [5]




[5] T. Robinson and F. Fallside. "A recurrent error propagation network speech recognition system". In: *Computer Speech and Language* 5.3 (1991), pp. 259–274

# HMM + RNN Dependencies




How do the two models interact? [9]

[9] G. Salvi. "Dynamic Behaviour of Connectionist Speech Recognition with Strong Latency Constraints". In: *Speech Communication* 48.7 (July 2006), pp. 802–818

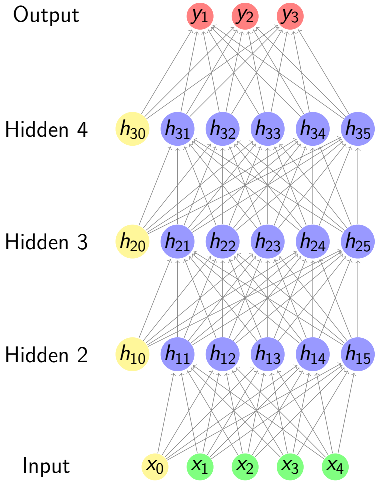
# Outline

State-to-Output Probability Model

Artificial Neural Networks
- Perceptron
- Multi Layer Perceptron
- Error Backpropagation
- Hybrid HMM-MLP

Deep Learning (Initialization)
- Deep Neural Networks
- Restricted Boltzmann Machines
- Deep Belief Networks

# Deep Neural Network



# DNN: Motivation

- depth $\sim$ abstraction
- good initialisation (see later)
- fast computers, large datasets

# DNN and MLPs

- no conceptual difference from MLPs
- Backpropagation alone not powerful enough [2]
- local minima
- vanishing gradients

(later BP has been proven to be sufficient)

[2] D. Erhan, Y. Bengio, A. Courville, P.-A. Mansagol, and P. Vincent. "Why Does Unsupervised Pre-training Help Deep Learning?" In: *Journal of Machine Learning Research* 11 (2010), pp. 625–660

# Deep Learning possible with Pre-Training

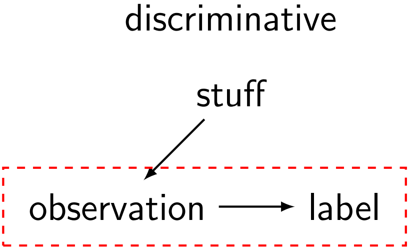
- pioneered by Geoffry Hinton (Univ. Toronto)
- unifies properties of generative and discriminative models
- most of the large companies are using it (Microsoft, Google, Nuance, IBM)
- most promising technique at the moment

# Pre-Training: Idea

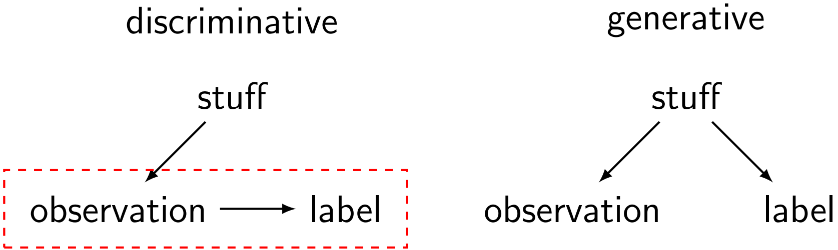
discriminative

$$\begin{tikzcd}[column sep=small] & \text{stuff} \arrow[dl] & \\ \text{observation} \arrow[rr] & & \text{label} \end{tikzcd}$$

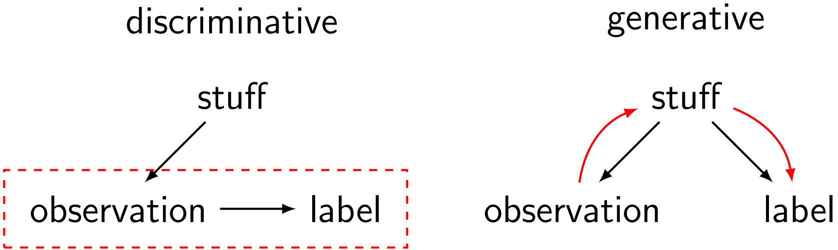
# Pre-Training: Idea



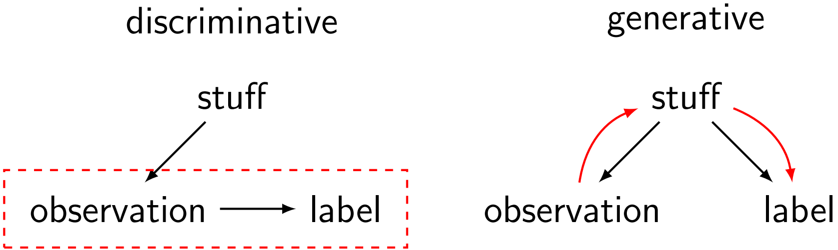
# Pre-Training: Idea



# Pre-Training: Idea



# Pre-Training: Idea



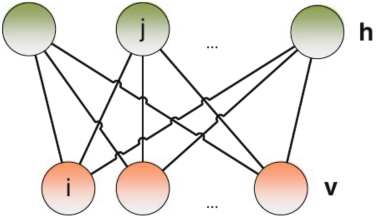

. . . but HMMs with Gaussian Mixure Models are also generative: why is this better?

# Deep Learning: Idea #2

1. initialise DNN with Restricted Boltzmann Machines (RBM) that can be trained unsupervised
2. use fast learning procedure (Hinton)
3. use ridiculous amounts of unlabelled (cheap) data to train a ridiculous number of parameters in an unsupervised fashion
4. at the end, use small amounts of labelled (expensive) data and backpropagation to learn the labels

# Restricted Boltzmann Machines (RBMs)

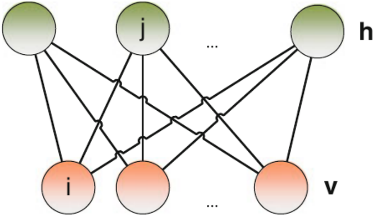
First called Harmonium [10]




- binary nodes: Bernoulli distribution
- continuous nodes: Gaussian-Bernoulli

[10] P. Smolensky. "Information processing in dynamical systems: Foundations of harmony theory". In: Department of Computer Science, University of Colorado, Boulder, 1986. Chap. 6

# Restricted Boltzmann Machines (RBMs)




Energy (Bernoulli):

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

Energy (Gaussian-Bernoulli):

$$E(\mathbf{v},\mathbf{h}) = \frac{1}{2}(\mathbf{v}-\mathbf{a})^T(\mathbf{v}-\mathbf{a}) - \mathbf{b}^T\mathbf{h} - \mathbf{h}^T\mathbf{W}\mathbf{v}$$

# RBM: Probabilistic Interpretation

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

Posteriors (conditional independence):

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

and

$$P(\mathbf{v}|\mathbf{h}) = \cdots = \prod_i P(v_i|\mathbf{h})$$

# Binary Units: Cond Prob

Posterior equals sigmoid function!!

$$
\begin{aligned}
P(h_i = 1|\mathbf{v}) &= \frac{e^{(b_i 1 + 1\mathbf{W}_{i,*}\mathbf{v})}}{e^{(b_i 1 + 1\mathbf{W}_{i,*}\mathbf{v})} + e^{(b_i 0 + 0\mathbf{W}_{i,*}\mathbf{v})}} \\
&= \frac{e^{(b_i 1 + 1\mathbf{W}_{i,*}\mathbf{v})}}{e^{(b_i 1 + 1\mathbf{W}_{i,*}\mathbf{v})} + 1} \\
&= \sigma(b_i 1 + 1\mathbf{W}_{i,*}\mathbf{v})
\end{aligned}
$$

Same as Multi Layer Perceptron (viable for initialisation!)

# Gaussian Units: Cond Prob

$$P(\mathbf{v}|\mathbf{h}) \;=\; \mathcal{N}(\mathbf{v}; \mu, \mathbf{\Sigma})$$

with

$$\mu = \mathbf{W}^T\mathbf{h} + \mathbf{a}$$
$$\mathbf{\Sigma} = \mathbf{I}$$

# RBM Training

Stochastic Gradient Descend (minimise the negative log likelihood)

$$J_{\text{NLL}}(\mathbf{W}, \mathbf{a}, \mathbf{b}, \mathbf{v}) = -\log P(\mathbf{v}) = F(\mathbf{v}) + \log \sum_{\mathbf{v}} e^{-F(\mathbf{v})}$$

where

$$F(\mathbf{v}) = -\log \left( \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \right)$$

is the free energy of the system.

BUT: the gradient can not be computed exactly

# RBM Gradient

$$\frac{\partial J_{\text{NLL}}(\mathbf{W}, \mathbf{a}, \mathbf{b}, \mathbf{v})}{\partial \theta} = \frac{\partial F(\mathbf{v})}{\partial \theta} - \sum_{\tilde{\mathbf{v}}} p(\tilde{\mathbf{v}}) \frac{\partial F(\tilde{\mathbf{v}})}{\partial \theta}$$

- first term increases prob of training data
- second term decreases prob density defined by the model

# RBM Stochastic Gradient

The general form is:

$$
\begin{aligned}
\nabla_\theta J_{\text{NLL}}(\mathbf{W}, \mathbf{a}, \mathbf{b}, \mathbf{v}) \;=\; -\Bigg[ & \left\langle \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta} \right\rangle_{\text{data}} \\
& - \left\langle \frac{\partial E(\mathbf{v}, \mathbf{h})}{\partial \theta} \right\rangle_{\text{model}} \Bigg]
\end{aligned}
$$

Example: visible layer

$$
\begin{aligned}
\nabla_{w_{ij}} J_{\text{NLL}}(\mathbf{W}, \mathbf{a}, \mathbf{b}, \mathbf{v}) \;=\; -\big[ & \langle v_i h_j \rangle_{\text{data}} \\
& - \langle v_i h_j \rangle_{\text{model}} \big]
\end{aligned}
$$

# Gibbs Sampling

$\langle v_i h_j \rangle_{\text{model}}$ computed with sampling

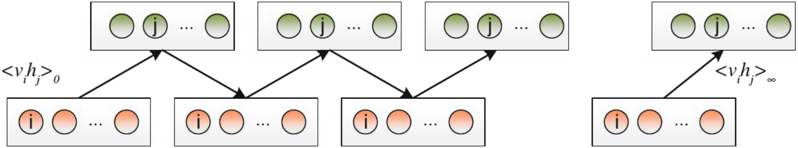
Sample joint distribution of $N$ variables, one at a time:

$$P(X_i | X_{-i})$$

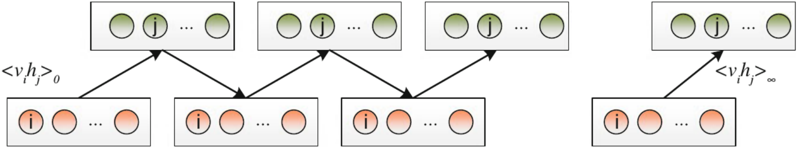
where $X_{-i}$ are all the other variables

BUT: it takes exponential time to compute exactly

# Contrastive Divergence



# Contrastive Divergence




Two tricks:

1. initialise the chain with a training sample
2. do not wait for convergence

# RBMs and Deep Belief Networks

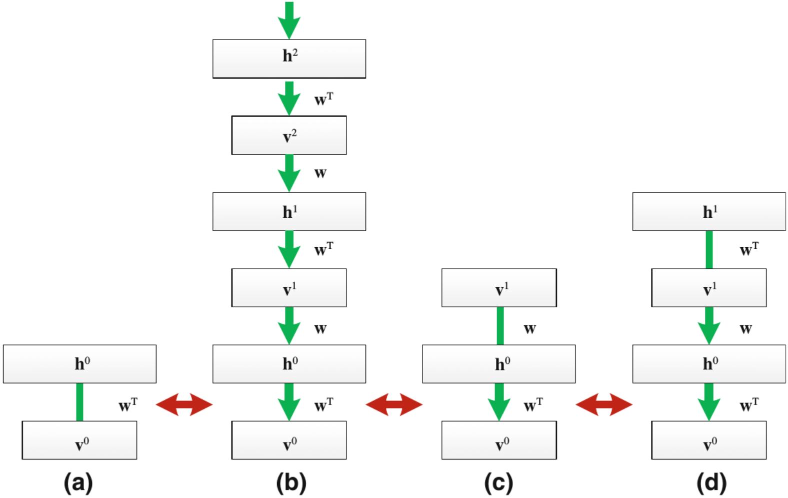


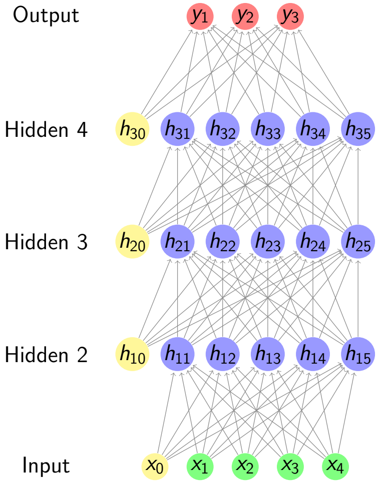

Figure from Yu and Deng

# Deep Belief Networks: Training

Yee-Whye Teh (one of Hinton's students) observed that DBNs can be trained greedily for each layer:

1. train a RBM unsupervised
2. excite the network with training data to produce outputs
3. use the outputs to train next RBM

# Final Step: Supervised Training



# The importance of pre-training

- 2006: Backpropagation alone not powerful enough [2]
- 2016: Backpropagation on large data sets (with tricks) is good enough (missing citation)

[2] D. Erhan, Y. Bengio, A. Courville, P.-A. Mansagol, and P. Vincent. "Why Does Unsupervised Pre-training Help Deep Learning?" In: *Journal of Machine Learning Research* 11 (2010), pp. 625–660

# Deep Learning: Performance

- state-of-the-art on most ASR tasks
- Made people from University of Toronto, Microsoft, Google and IBM write a paper together [4]
- experiments with learning the features from speech signal

Yu and Deng's book has many examples

[4] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. "Deep Neural Networks for Acoustic Modeling in Speech Recognition". In: *IEEE Signal Processing Magazine* (2012)

# ANNs in ASR: Advantages

- discriminative in nature
- powerful time model:
- Time-Delayed Neural Networks (TDNNs)
- Recurrent Neural Networks (RNNs)

# ANNs in ASR: Disadvantages

- training requires state level annotations (no EM available)
- usually annotations obtained with forced alignment (Viterbi training)
- not easy to adapt
- we still need GMM-HMMs for the training

# Typical Training Procedure

1. train a full context dependent GMM-HMM system
2. cluster CD HMM states into senones (order of 1000)
3. use senones to define output of DNN
4. run forced alignment with GMM-HMMs
5. train DNN with forced aligned transcriptions