PhD in Computer Science and Engineering
Bologna, April 2016

Machine Learning

Marco Lippi

marco.lippi3@unibo.it

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Deep Learning: a revolution for AI

The history of Artificial Neural Networks...

1943 **McCulloch and Pitts** – The Artificial Neuron
1958 **Rosenblatt** – Perceptron
1969 **Minsky and Papert** – Limits of the perceptron

AI Winter ☹

1974 **Werbos** – First ideas of backpropagation
1984 **Hopfield** – Hopfield Networks
1986 **Rumelhart, Hinton, Williams** – Backpropagation ☺

Kernel Machines and Statistical Learning boom... 😐

2006 **Hinton, Bengio, LeCun, Ng,** ... Deep Learning 😎

Bringing AI in the news...

NYU "Deep Learning" Professor LeCun Will Head Facebook's New Artificial Intelligence Lab

Posted Dec 9, 2013 by Josh Constine (@joshconstine)

857 SHARES

By teaching a computer to think, Facebook hopes to better understand how its users do too. So today the company announced that one of the world's leading deep learning and machine learning scientists, NYU's Professor Yann LeCun, will lead its new artificial intelligence laboratory.

Google Buys U.K. Artificial Intelligence Company DeepMind

GOOGLE'S ARTIFICIAL BRAIN LEARNS TO FIND CAT VIDEOS

Scientists See Promise in Deep-Learning Programs

By JOHN MARKOFF NOV. 23, 2012

A voice recognition program translated a speech given by Richard F. Rashid, Microsoft's top scientist, into Mandarin Chinese. Hao Zhang/The New York Times

Deep Learning

Breakthrough in many AI applications

- Speech recognition
- Image classification
- Object detection
- Video classification
- Scene understanding
- Natural language understanding
- Machine translation
- ...

What is REALLY new ?

Deep Learning = neural networks with many layers ?

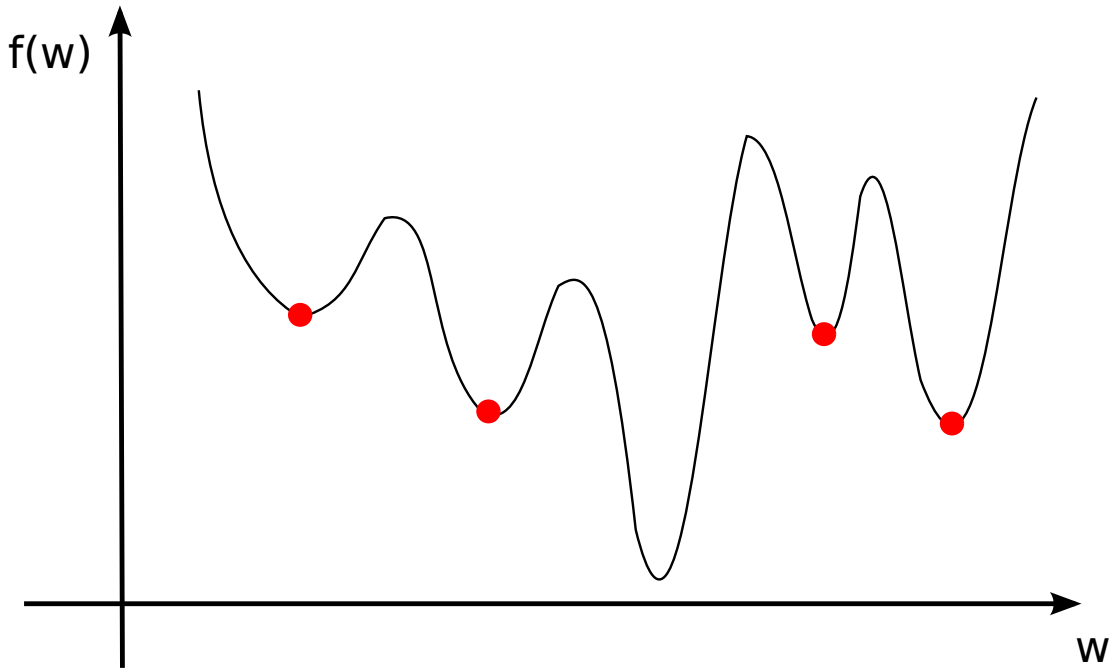
Deep neural networks were already known since Multi-Layer Perceptron, so **why** has deep learning become popular **only now** ?

Main limitations of (deep) neural networks in the 90s:

1. Local minima
2. Vanishing gradients
3. Curse of dimensionality
4. Computational requirements

Local minima

Attractors for the classic backpropagation algorithm...

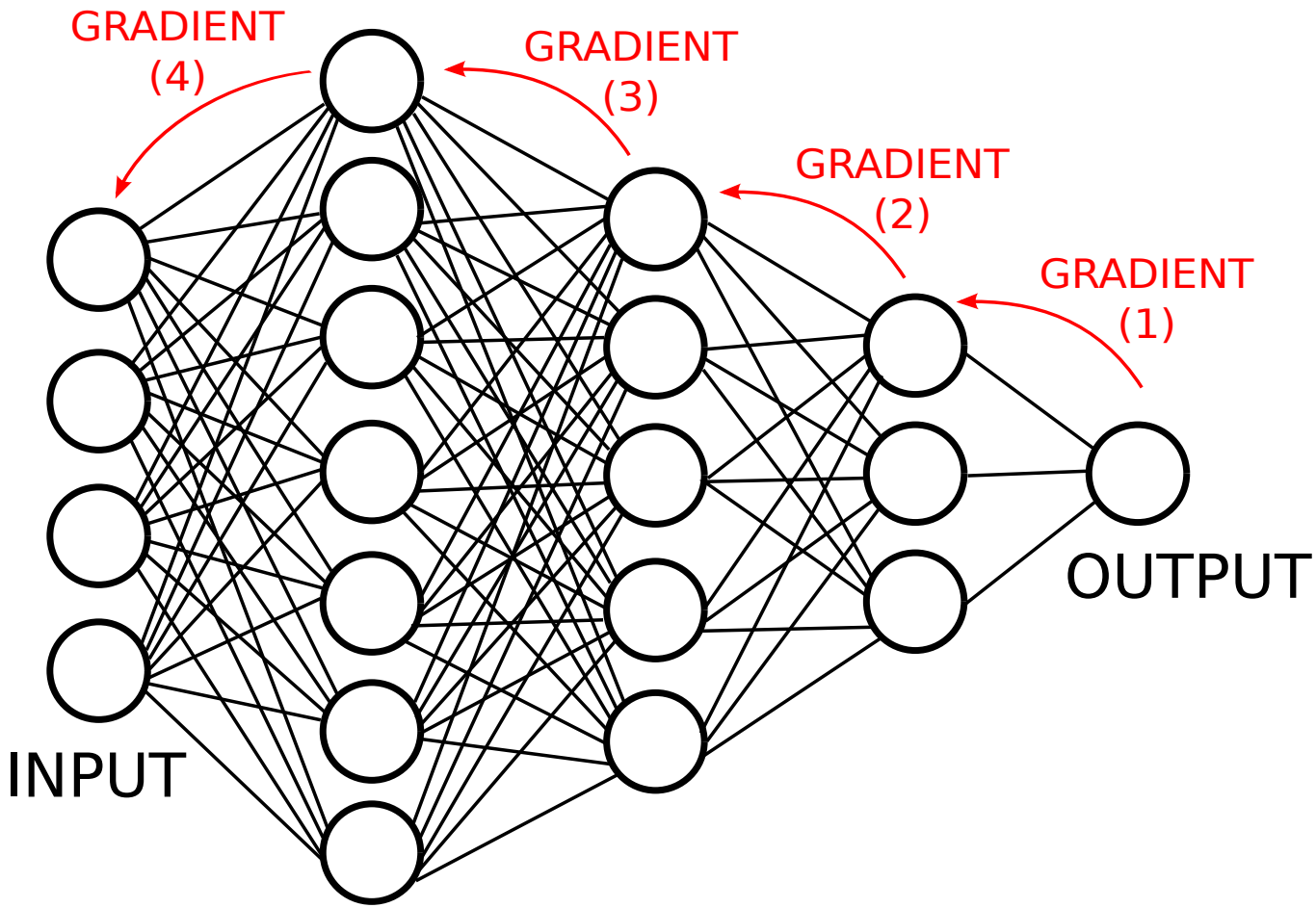


A few solutions...

- Multi-start with random initializations
- Momentum term $\longrightarrow w_{ij}^{t} = w_{ij}^{t-1} - \eta \Delta w_{ij}^{t} + \mu \Delta w_{ij}^{t-1}$

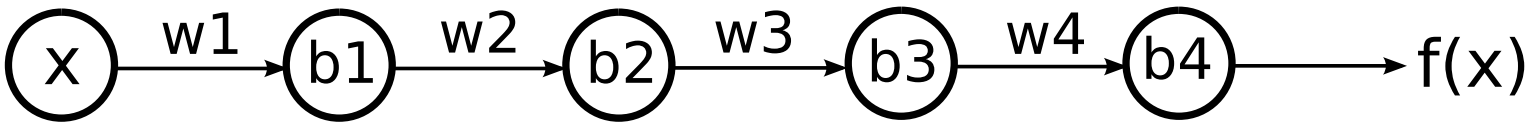
Vanishing gradients

The composition of several gradient contributions can make the values either explode or vanish to zero (happening also for activation functions)



Vanishing gradients

Consider an ANN with 4 layers and **a single unit** in each layer.



We name x the input, w_j and b_j the weight and bias of layer j.
The output of each layer can be computed as:

$$z_j = \sigma(w_j z_{j-1} + b_j)$$

where $z_0 = x$.

[Example by M. Nielsen]

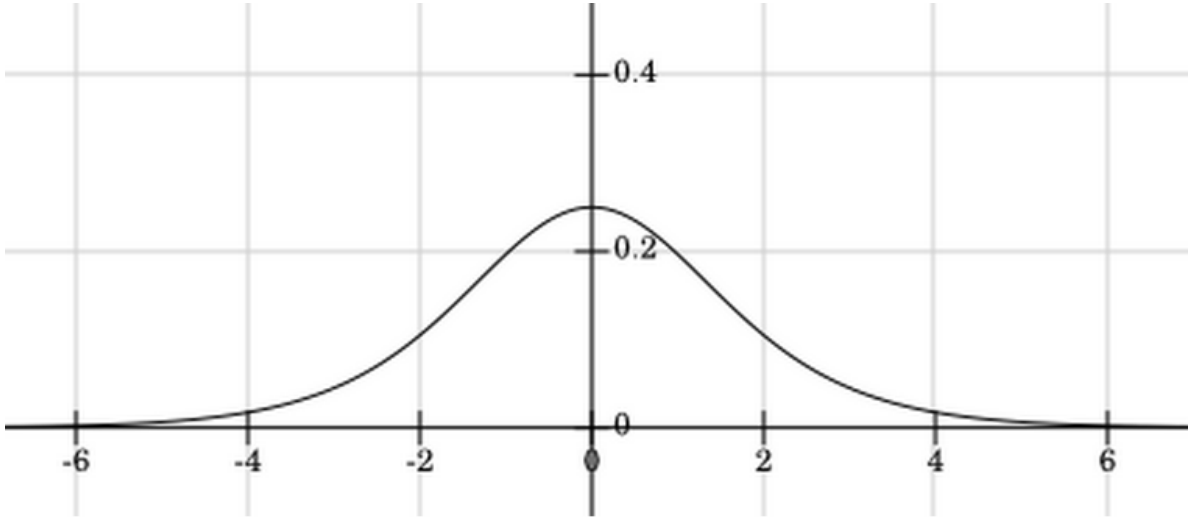
Let us compute the gradient of f with respect to b_1 and b_3:

$$\frac{\partial f}{\partial b_1} = \sigma'(z_1) \cdot w_2 \cdot \sigma'(z_2) \cdot w_3 \cdot \sigma'(z_3) \cdot w_4 \cdot \sigma'(z_4) \cdot \frac{\partial f}{\partial z_4}$$

$$\frac{\partial f}{\partial b_3} = \sigma'(z_3) \cdot w_4 \cdot \sigma'(z_4) \cdot \frac{\partial f}{\partial z_4}$$

Vanishing gradients

If we now consider the derivative of the sigmoid function:



we have that, with typical weight initialization (randomly sampled from a Gaussian with mean 0 and standard deviation 1):

$$|w_j \sigma'(z_j)| < \frac{1}{4}$$

Vanishing gradients

Now going back to the gradient computation:

$$\frac{\partial f}{\partial b_1} = \sigma'(z_1) \cdot w_2 \cdot \sigma'(z_2) \cdot w_3 \cdot \sigma'(z_3) \cdot w_4 \cdot \sigma'(z_4) \cdot \frac{\partial f}{\partial z_4}$$

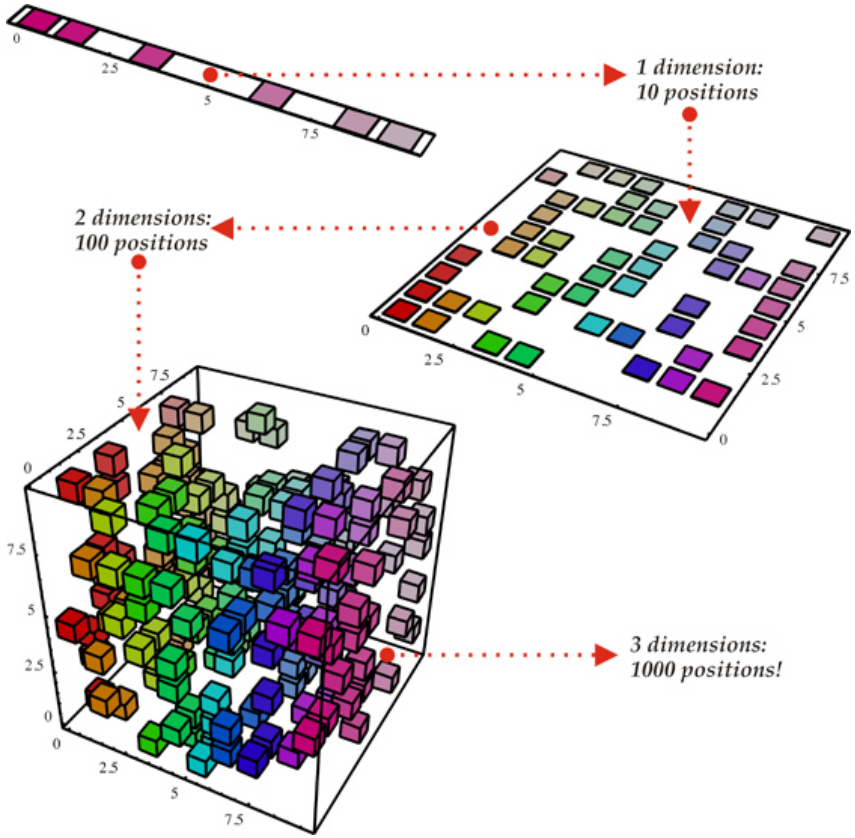
$$\frac{\partial f}{\partial b_3} = \sigma'(z_3) \cdot w_4 \cdot \sigma'(z_4) \cdot \frac{\partial f}{\partial z_4}$$

Lower layers tend to have either vanishing or exploding gradients !

Exhaustive analysis for Recurrent Neural Networks in [Hochreiter et el., 2001]

Curse of dimensionality

input combinations grows non-linearly with # input variables

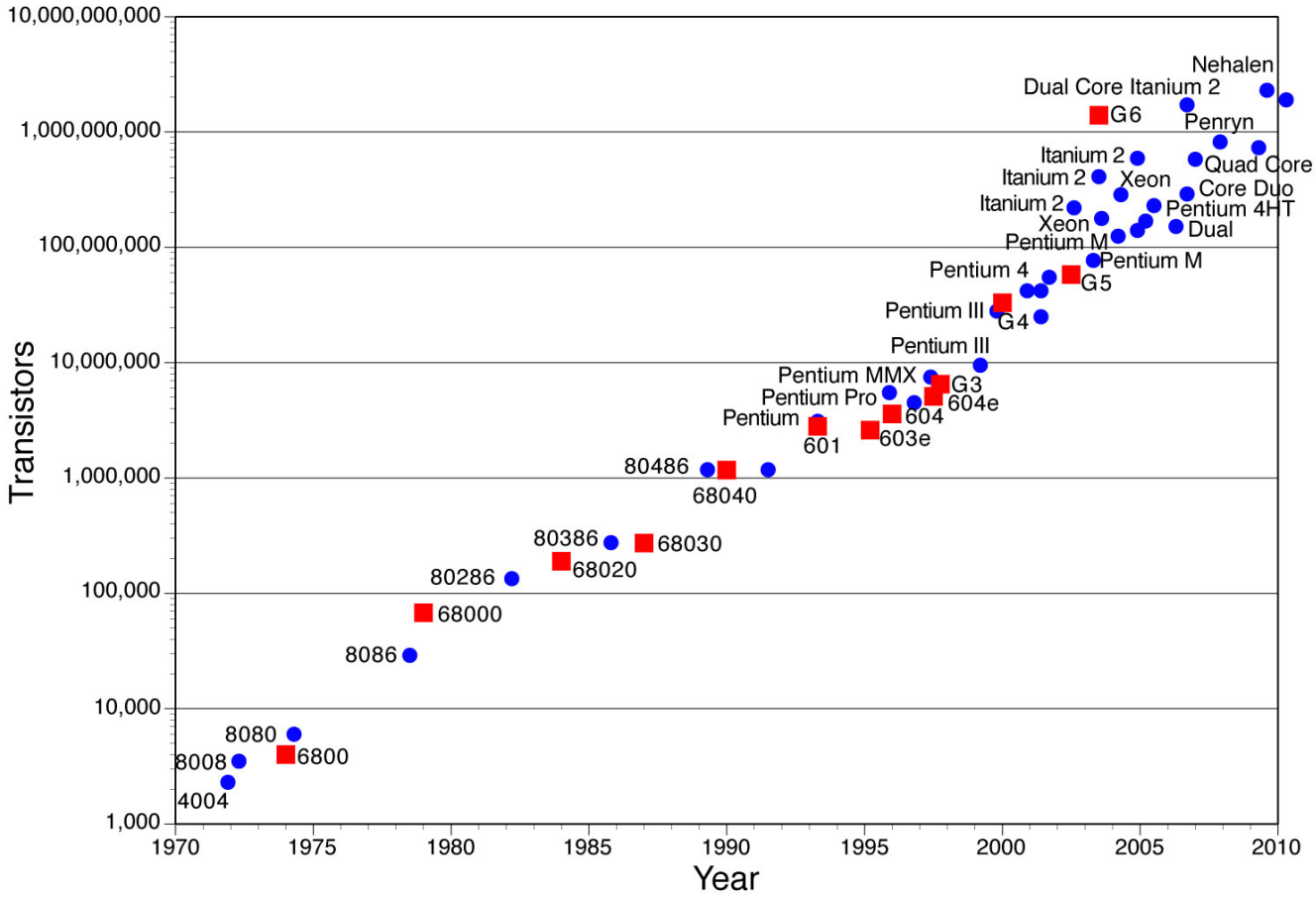


[Figure by Y. Bengio]

Computational requirements

Training deep models in the 90s was extremely time-consuming...

... Now computers are faster ! ☺



Theoretical arguments

An ANN with a **single hidden layer** is a universal approximator...
... but it may require an exponential number of hidden neurons...
... and therefore it may be much more difficult to learn !

Analogy with digital circuits $\rightarrow$ N-bit parity:

- requires N-1 gates with depth $\log(N)$
- requires exponential # gates with 2 layers only

Theoretical arguments

Advantages of deep architectures:

- representing functions with a lower number of elements
- more compact and more efficient model (fewer parameters)

Observation

The number of affordable parameters of a neural network depends on the number of training examples that can be used to tune them

$\Longrightarrow$ Insufficiently deep architecture leads to **poor generalization** (Occam's razor)

Training deep models

Since the development of multi-layer perceptrons, only **1 or 2 hidden layers** had obtained successful results...

- One exception: Convolutional Neural Networks (next lecture)

Breakthrough in 2006 with Deep Belief Networks (DBNs) developed at the University of Toronto [Hinton et al., 2006]:

- train **one layer at time**
- exploit **unsupervised learning**
- use supervisions only for a **fine tuning** of the network

Supervised vs. unsupervised learning

Supervised learning

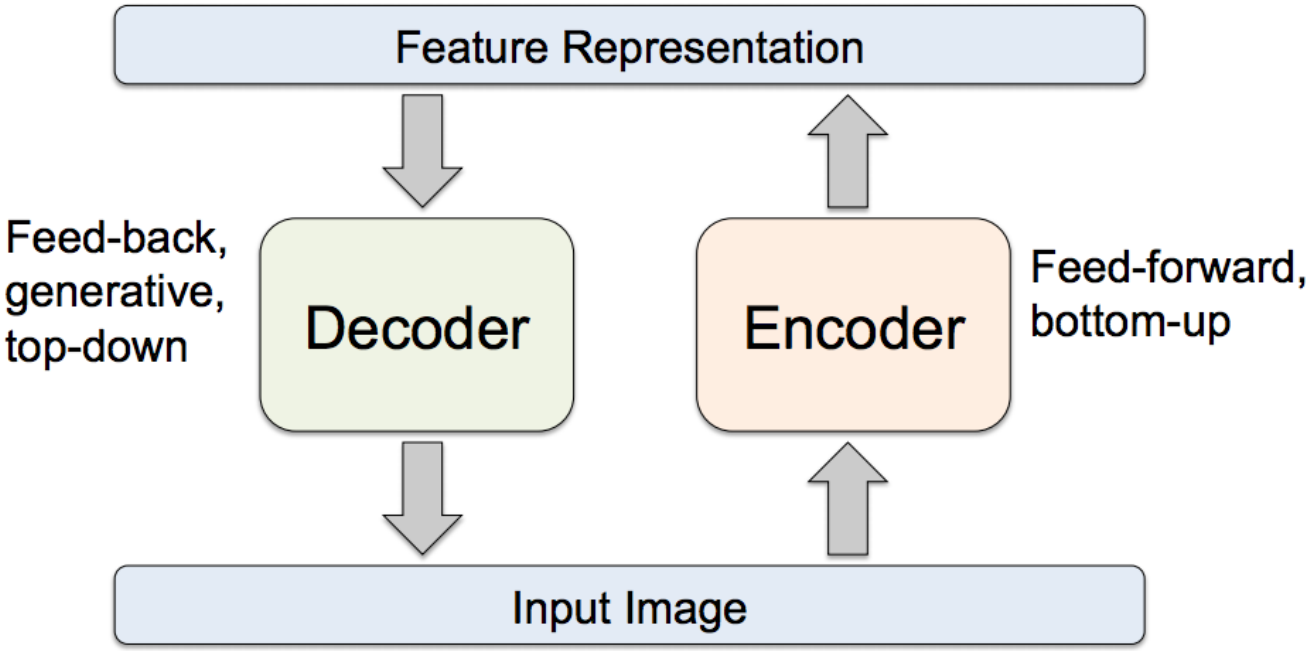
- data consists in **observations** $\mathcal{X}$ and **labels** $\mathcal{Y}$
- given an observation $x \in \mathcal{X}$, the goal is to **predict** $y \in \mathcal{Y}$
- learn a **function** $f : \mathcal{X} \to \mathcal{Y}$ from a data set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$

Unsupervised learning

- data consists in observations $\mathcal{X}$
- find **regularities** and **patterns** in data
- understand **which features are most important**
- learn a **representation** of data

Auto-Encoders (more on this later)

An **Auto-Encoder** is an ANN that learns **a representation of the input** through which **it is possible to reconstruct the input**



[Figure by R. Salakhutdinov]

It is an example of unsupervised learning !

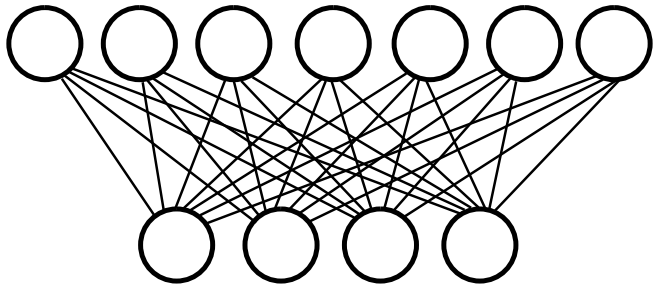
A generic architecture for deep learning

1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning

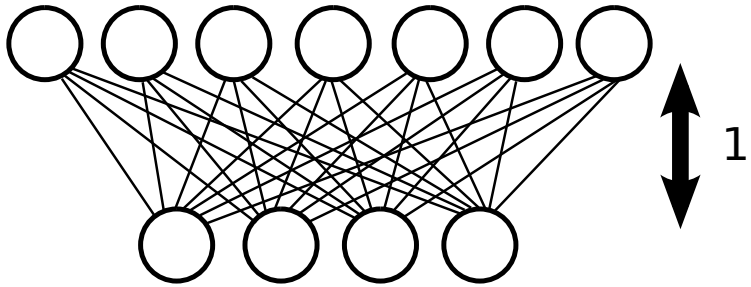
1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning



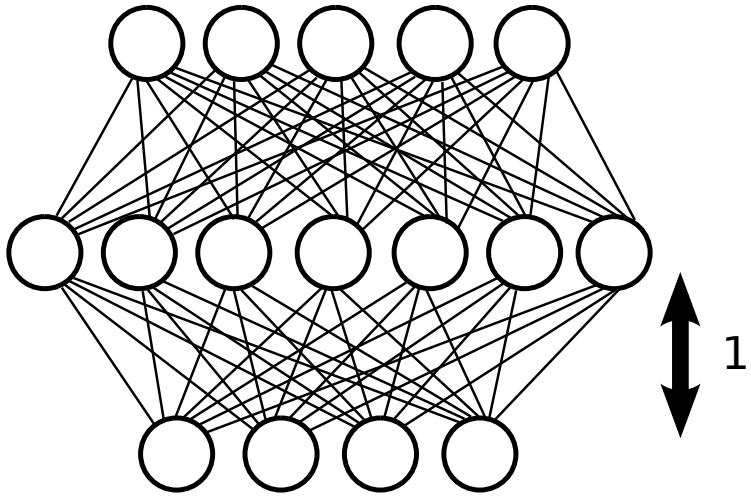
1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning



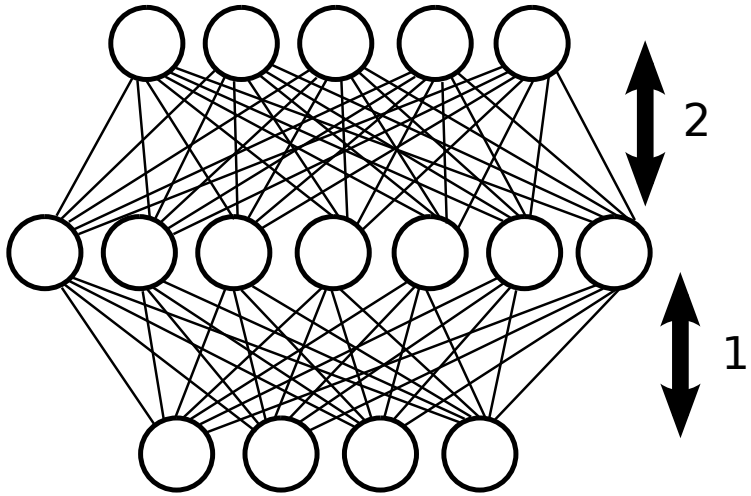
1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning



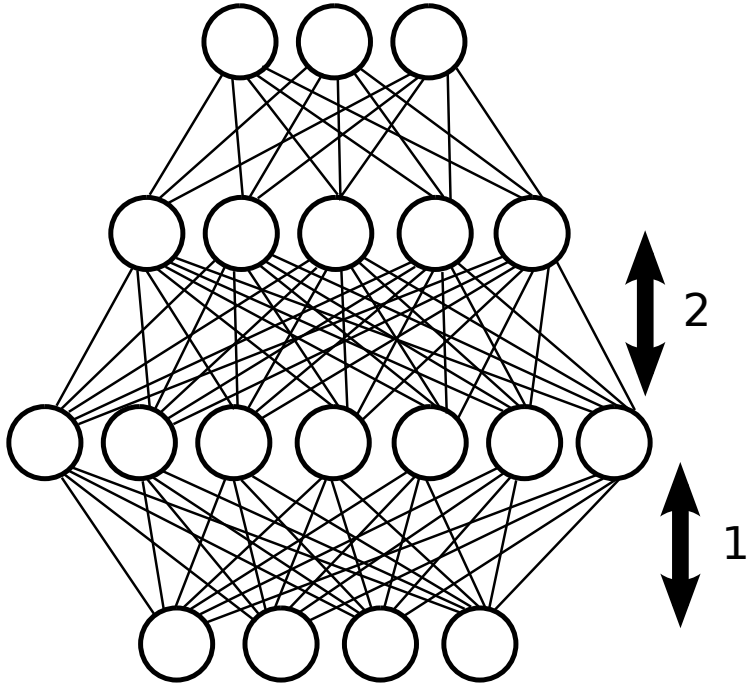
1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning



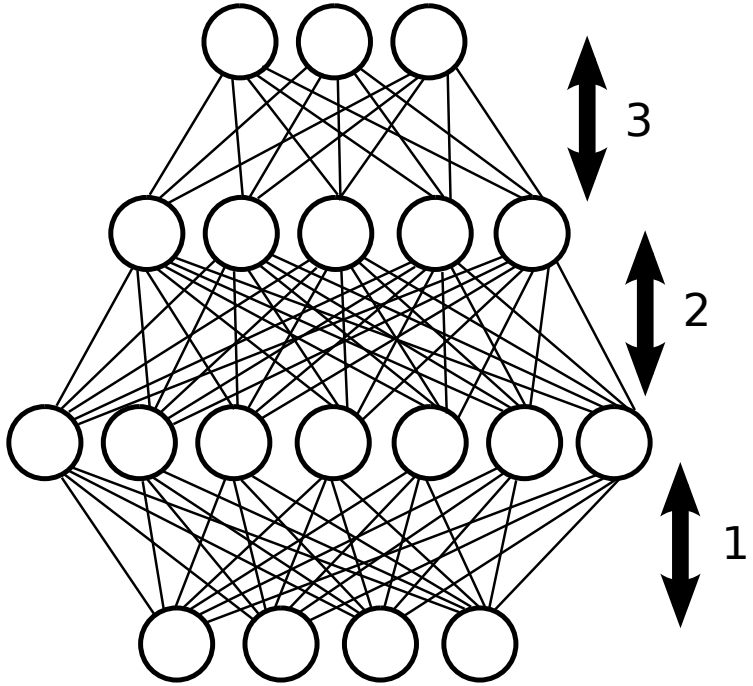
1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning



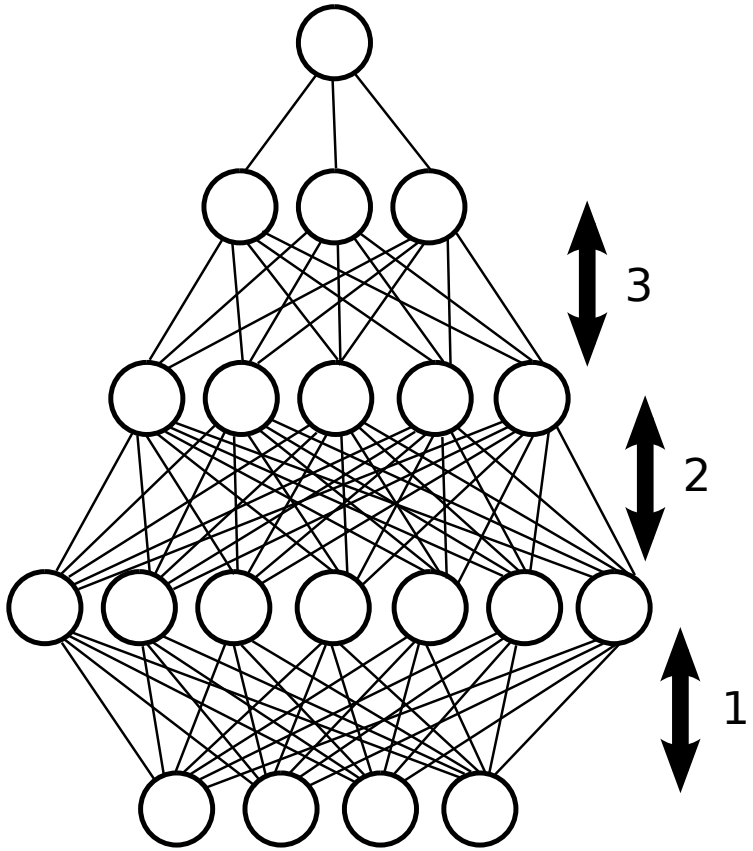
1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning



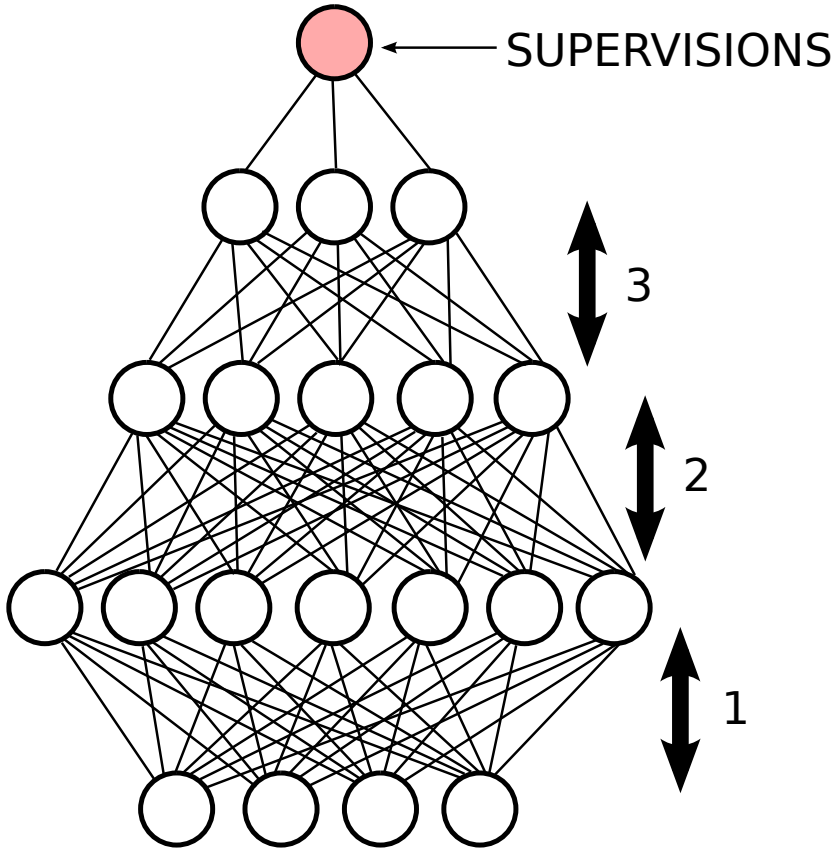
1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning



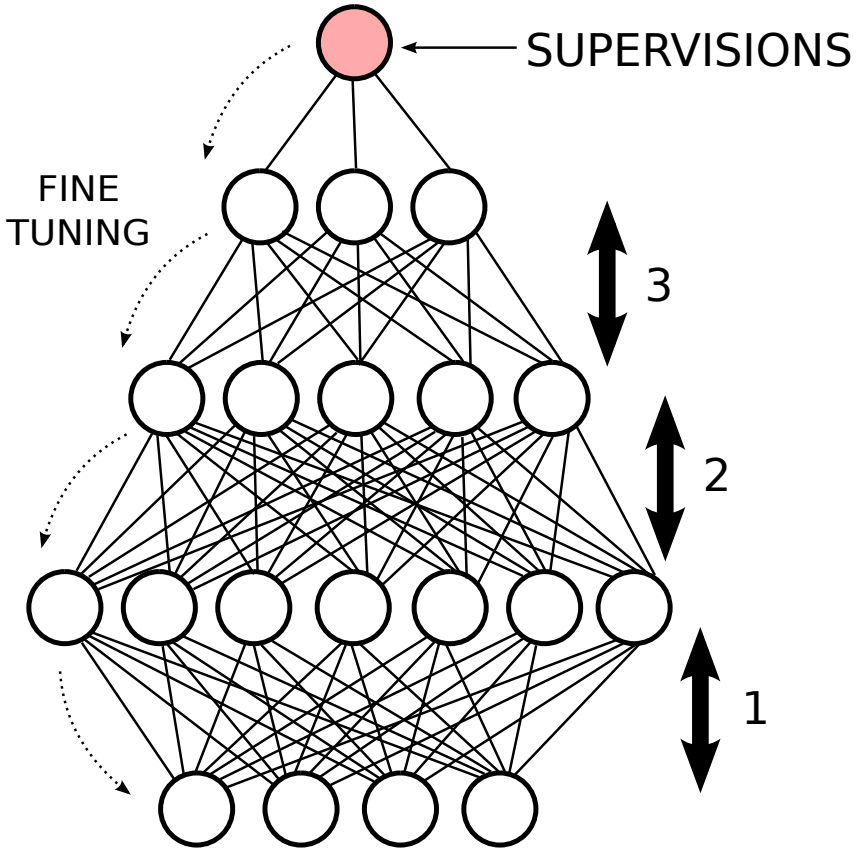
1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning



1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

A generic architecture for deep learning



1. Train one layer per time **to reconstruct the input**
2. **Unsupervised** training for each layer
3. Once a layer is trained, **pass to the upper one**
4. Finally, perform **fine-tuning** with supervisions

Representation learning

Deep learning is essentially **representation learning**

- Most machine learning approaches work well for specific tasks owing to **a great effort** in the design of **good features**
- Finding appropriate features can be **a time-consuming** but also **subtle task** even for specialists
- Representation learning tries to **automatically learn good features** from raw input data (even not knowing the task ?)
- Deep learning tries to learn a **hierarchy of multiple levels** of representation having **increasing complexity**

Discriminative vs. Generative models

Discriminative models

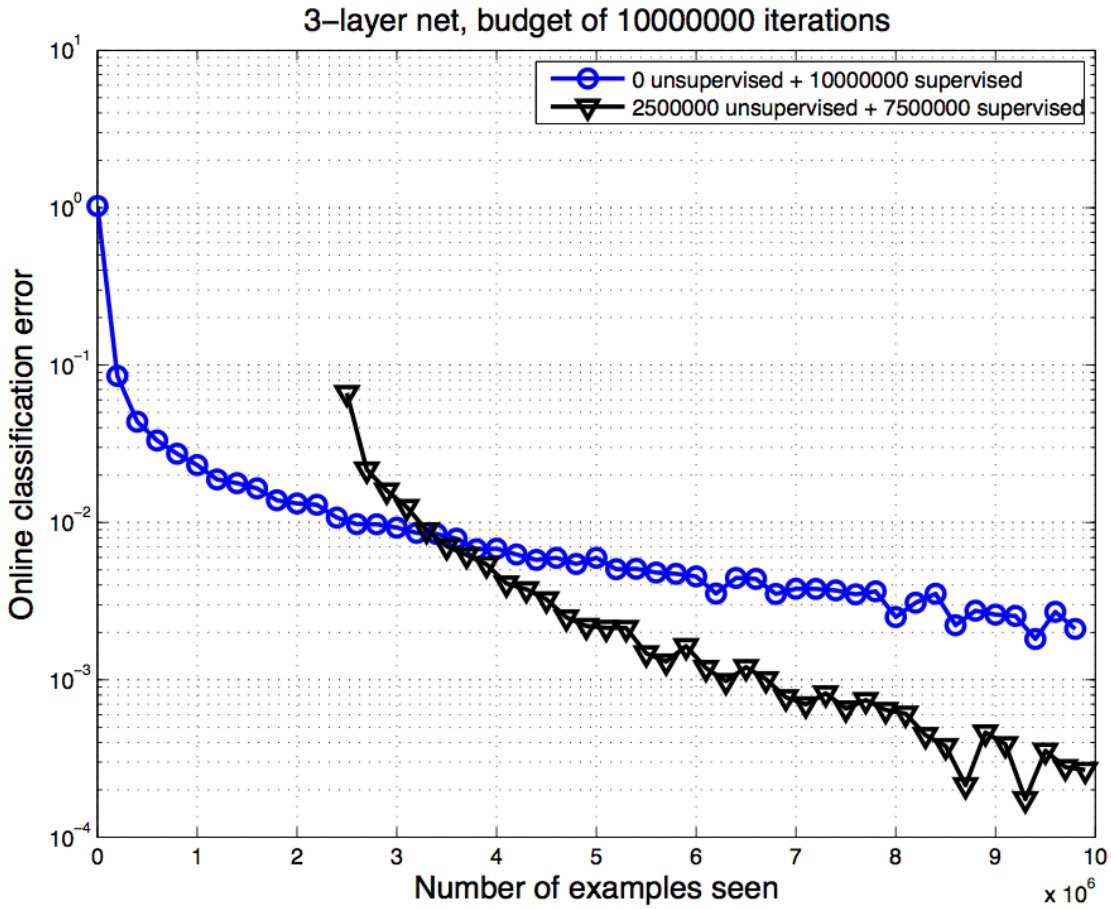
- **directly** model $P(Y|X;W)$
- need to **make assumptions** about input x

Generative models

- learn probabilistic model $P(X;W)$
- use learned parameters to **init** discriminative model
- **no knowledge** on further discriminative task !
- semi-supervised learning !

Why is unsupervised learning useful ?

For the same training error (at different points during training), test error is systematically lower with unsupervised pre-training. [Erhan et al., 2009]



Why is unsupervised learning useful ?

As discussed in [Erhan et al., 2009], unsupervised pre-training can be seen as a form of regularizer (or prior)...

Observation

Unsupervised pre-training amounts to **a constraint on the region in parameter space where a solution is allowed**.

Experiments show that the effect of unsupervised pre-training is most marked for the lower layers of a deep architecture.

Overfitting vs. underfitting

Overfitting:

- **Good training** performance, **bad generalization** capability
- Typically model too complex: too many parameters

Underfitting:

- **Bad training** performance
- Typically model too simple: too few parameters

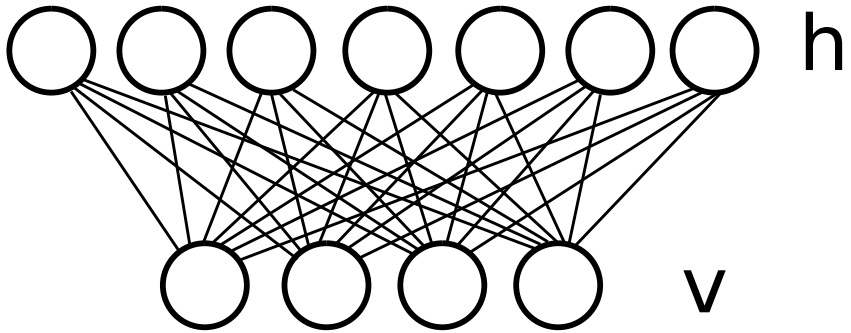
Overfitting vs. underfitting

- In a deep architecture, driving training error very low is simple, **even only with the top two layers** with enough hidden units
- Low training error and high test error $\Rightarrow$ overfitting
- Pre-training induces a data-dependent **regularization**

- Note that, with **small training set**, unsupervised pre-training can lower test error **despite a larger training error**
- With **larger training sets**, with better initialization of the lower hidden layers, both training and generalization error can be significantly decreased with unsupervised pre-training.

Deep Belief Networks, Restricted Boltzmann Machines, and Energy-Based Models

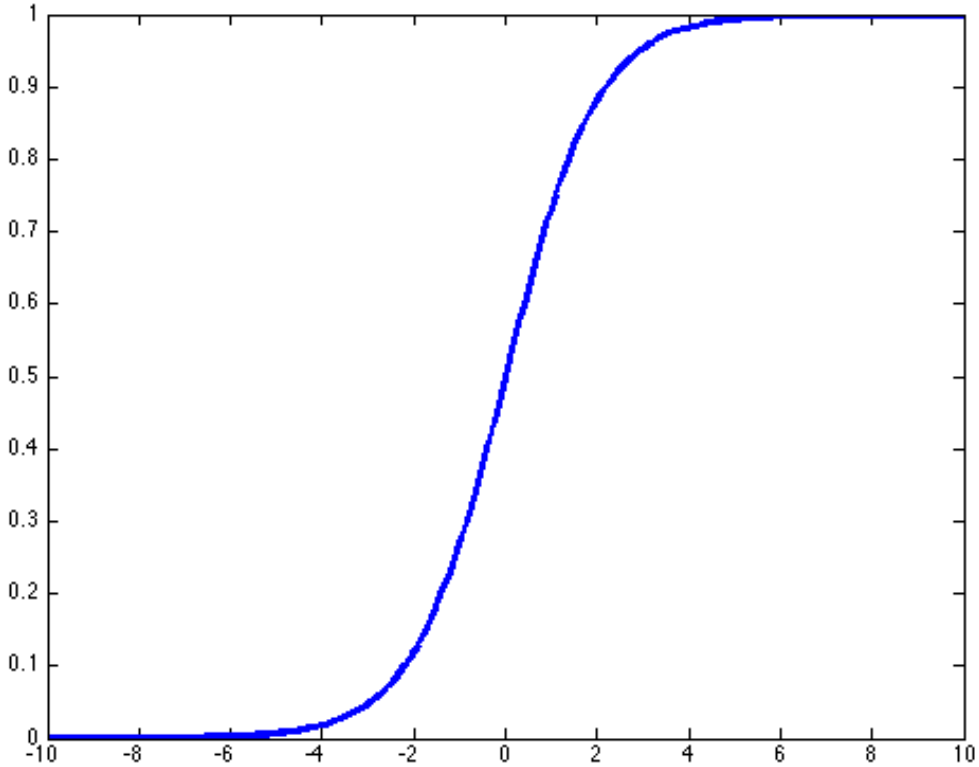
Restricted Boltzmann Machines (RBMs)



Modeling two sets of random variables V and H:

- $V = V_1, \ldots, V_m$ (**visible**)
- $H = H_1, \ldots, H_n$ (**hidden**)
- all H_i are independent when conditioning on V
- all V_j are independent when conditioning on H

In classic RBMs all units are stochastic binary (0/1) units.



$$Prob(neuron = active) = \frac{1}{1 + e^{-input}}$$

Since there are no connections between units of the same layer, the structure of the RBM induces a probability factorization, so that:

$$P(H = h|V = v) = \prod_i P(h_i|v)$$

$$P(V = v|H = h) = \prod_i P(v_i|h)$$

The probability distribution over such units is computed as:

$$P(h_j = 1|v) = \sigma(c_j + \sum_i v_i W_{ij})$$

$$P(v_i = 1|h) = \sigma(b_j + \sum_j h_j W_{ij})$$

RBMs belong to the class of **Energy-Based Models** (EBMs):

$$P(x) = \frac{e^{-Energy(x)}}{Z}$$

where

$$Z = \sum_x e^{-Energy(x)}$$

is called the **partition function**.

We will discuss EBMs in more detail later on...

When some variables are **observed** and some others are **latent**, the above definition can be re-written as:

$$P(x) = \sum_h \frac{e^{-Energy(x,h)}}{Z} = \frac{e^{-FreeEnergy(x)}}{Z}$$

being $$FreeEnergy(x) = -\log \sum_h e^{-Energy(x,h)}$$

For RBMs we have the following formulation:

$$Energy(v, h) = -\sum_i b_i v_i - \sum_j c_j h_j - \sum_i \sum_j v_i w_{ij} h_j$$

$$FreeEnergy(v, h) = -\sum_i b_i v_i - \sum_i \log \sum_{h_i} e^{h_i (c_i + \sum_k v_k w_{kj})}$$

Classic learning method for RBMs: **Maximum Likelihood**

The likelihood of the visible data v can be written as a **sum over all possible configurations of hidden states**

$$\mathcal{L} = P_{model}(v) = \sum_h p(v,h) = \frac{1}{Z} \exp\left(\sum_{ij} v_i w_{ij} h_j\right)$$

- Maximize likelihood **of given data** w.r.t. weights w_{ij}
- This is equivalent to minimize the **Kullback-Leibler divergence** between data and model distributions

We can compute the derivative of the log-likelihood w.r.t. w_{ij}:

$$\frac{\partial \log \mathcal{L}}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$$

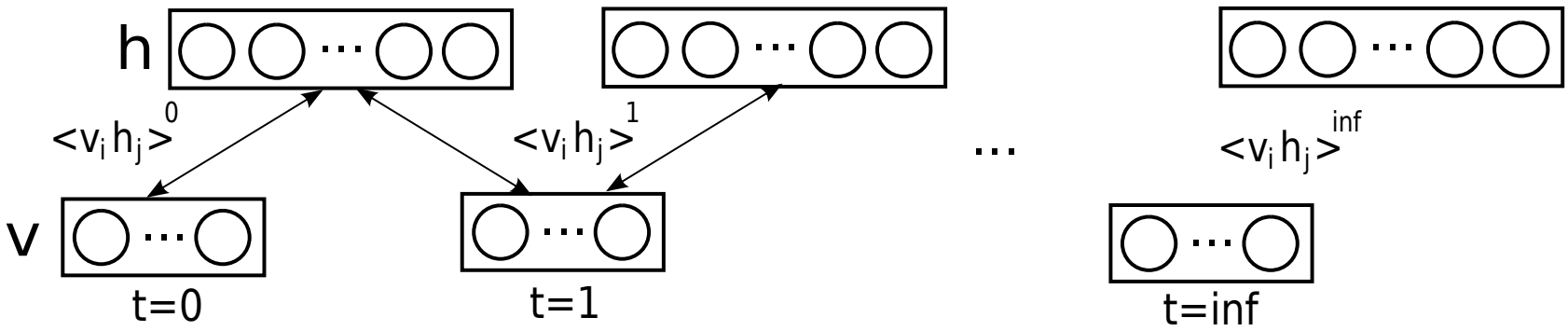
from which we can derive a **gradient update rule**:

$$\Delta w_{ij} = \rho \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right)$$

where $\langle \cdot \rangle$ indicate **expectations** over random variables.

First term: straightforward (**observed**)
Second term: needs some computations...

Classic method: **alternating Gibbs sampling**

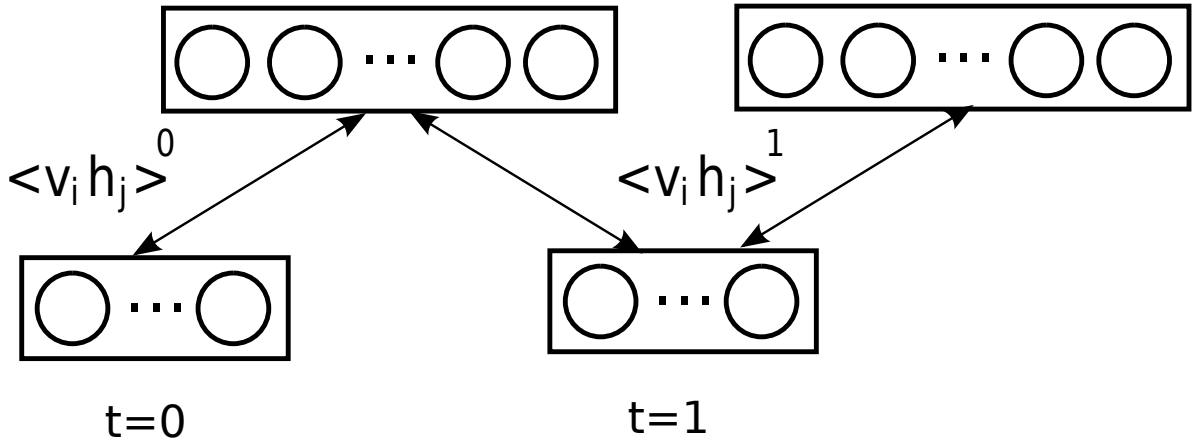


$h_i^{(n)}$ randomly chosen to be 1 with probability $\sigma(W_i' v^{(n)} + c_i)$
$v_j^{(n+1)}$ randomly chosen to be 1 with probability $\sigma(W_{\cdot j} h^{(n)} + b_j)$

$$\Delta w_{ij} = \rho\left(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty\right)$$

Proof of **convergence to the real distribution** (but slow...)

A much more efficient algorithm: **constrastive divergence**



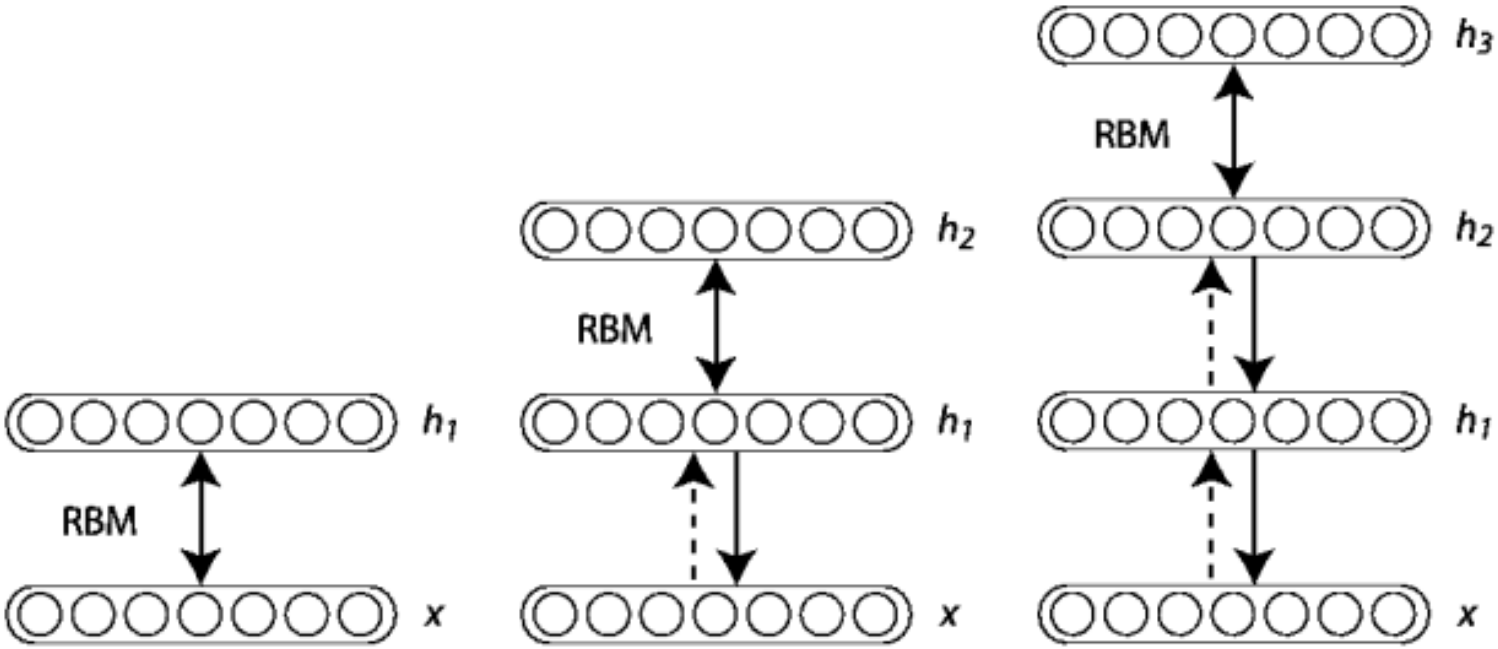
Stochastic gradient descent with update rule:

$$\Delta w_{ij} = \rho \left(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1 \right)$$

This is a very **fast** and **efficient** approach !

Deep Belief Networks [Hinton et al., 2006]

A DBN consists in a stack of RBMs



1. Layer-wise pre-training of all layers
2. Fine tuning with backpropagation

Deep Belief Networks [Hinton et al., 2006]

Experiments performed on:

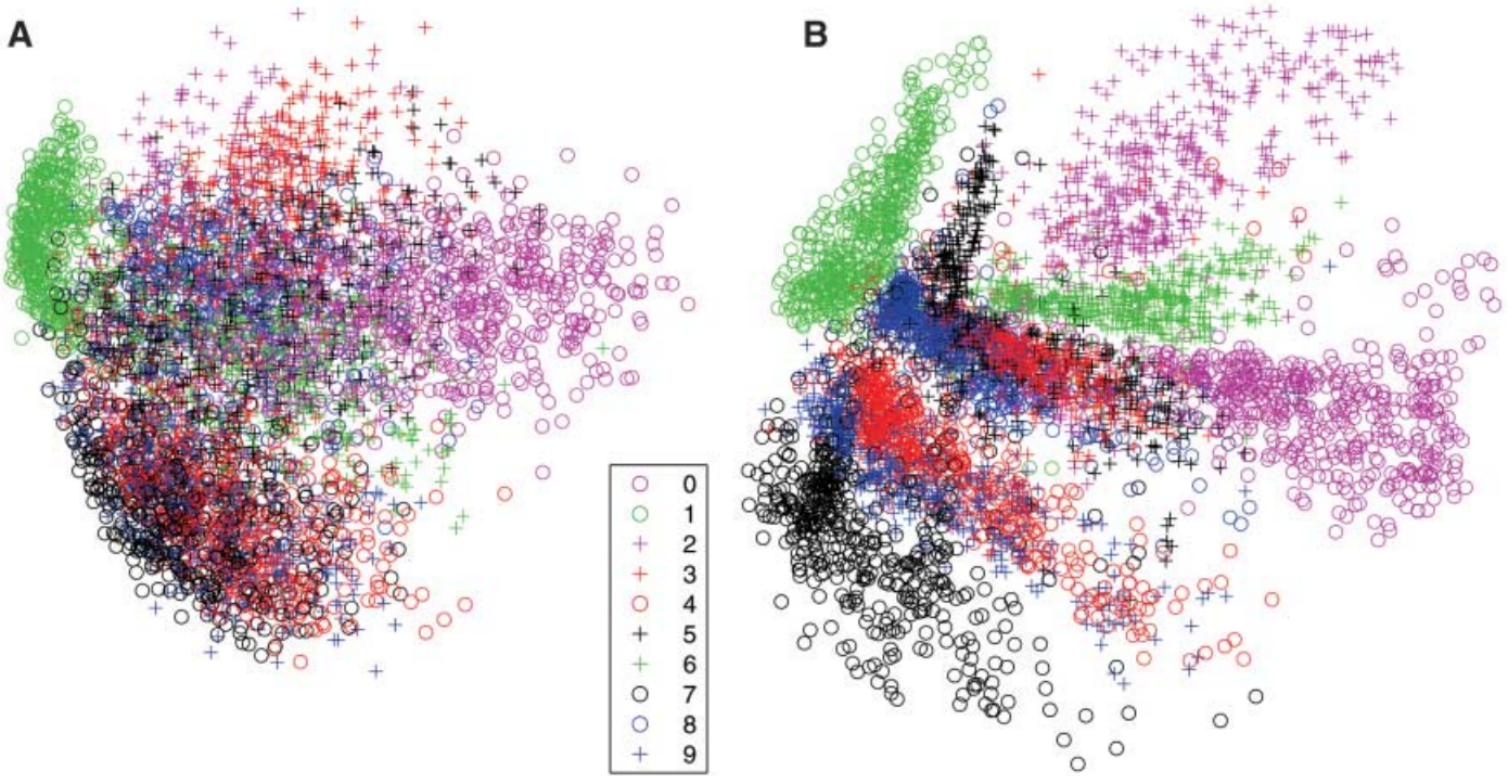
1. Image classification (Digits, Faces)
2. Document classification (Reuters corpus)

In both cases, a DBN was trained with 4 layers, with the final feature encoder consisting in only 2 dimensions:

- 784 – 1000 – 500 – 250 – 2 (images)
- 2000 – 500 – 250 – 125 – 2 (text)

Deep Belief Networks [Hinton et al., 2006]

Image classification on the MNIST digit data set

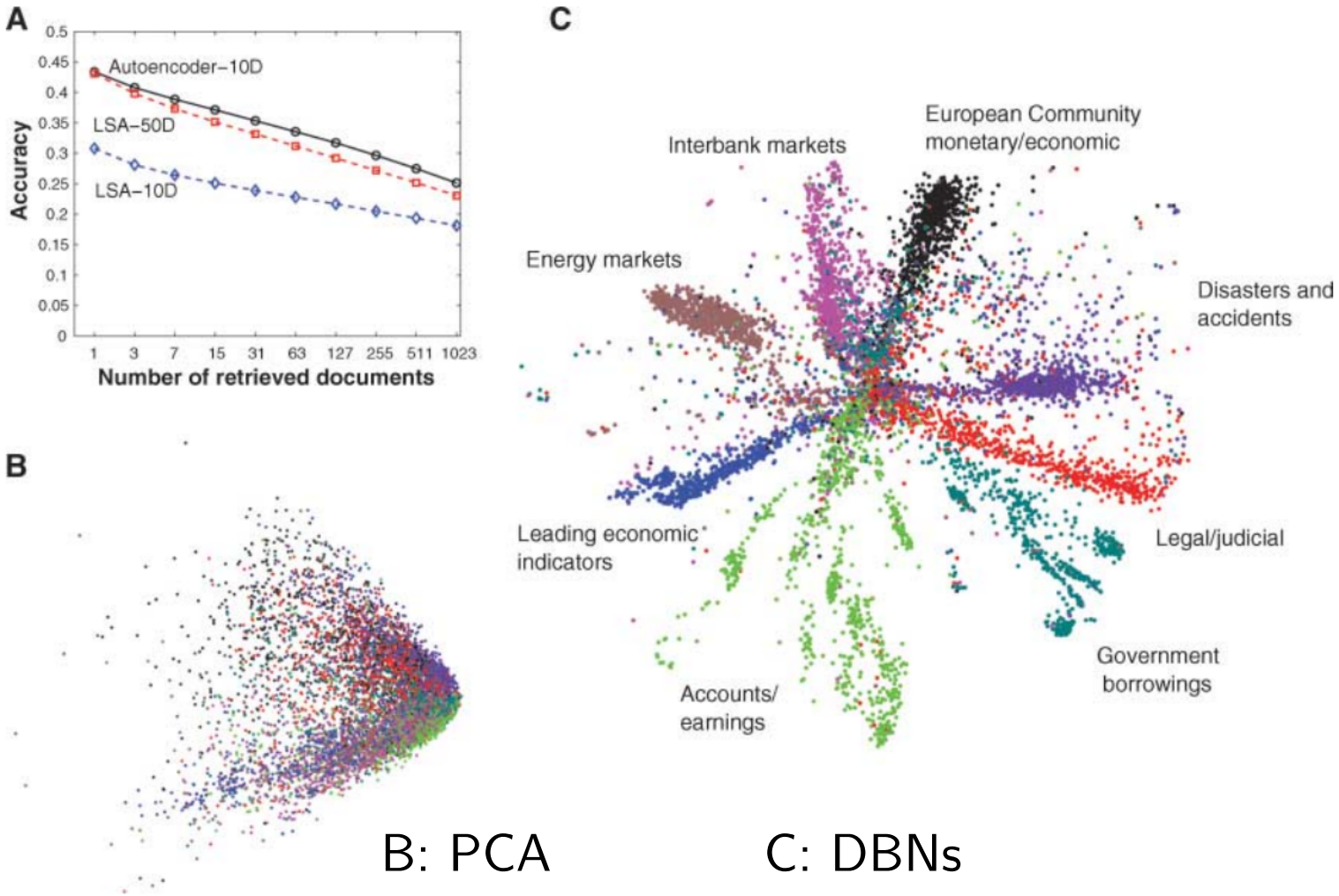


A: PCA B: DBNs

Figure by [Hinton et al., 2006]]

Deep Belief Networks [Hinton et al., 2006]

Document classification on the Reuter corpus



B: PCA C: DBNs

Figure by [Hinton et al., 2006]]

Deep Belief Networks [Hinton et al., 2006]

Digit reconstruction

Figure by [Hinton et al., 2006]]

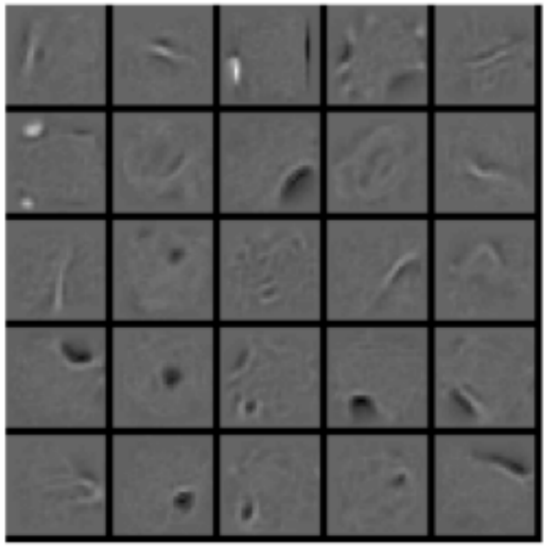
1: original images, 2: DBN (top layer 30), 3-4: PCA

Deep Belief Networks [Hinton et al., 2006]

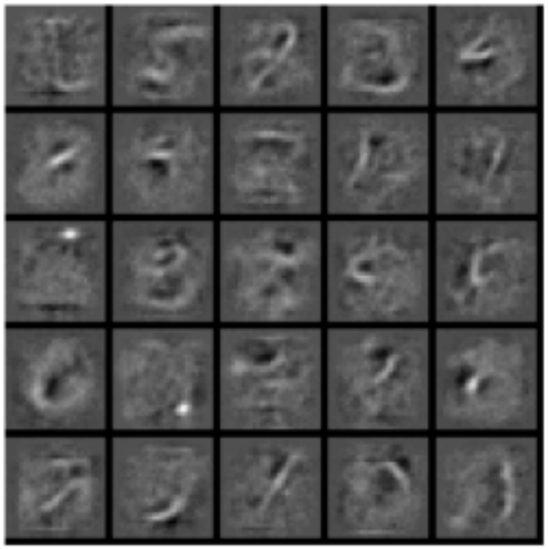
Faces reconstruction

Deep Belief Networks [Hinton et al., 2006]

1^{st}-layer features



2^{nd}-layer features



Energy-based models

Assigning a value of **energy** to each configuration of:

- **observed** variable X
- **predicted** variable Y

Inference in an energy-based model:

$$Y^* = \arg \min_{Y \in \mathcal{Y}} E(X, Y)$$

This can be computationally expensive according to $|\mathcal{Y}|$

From energies to probabilities via **Gibbs distribution**:

$$P(Y|X) = \frac{e^{-\beta E(Y,X)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(Y,X)}}$$

If we explicit the model parameters W:

$$P(Y|X,W) = \frac{e^{-\beta E(W,Y,X)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(W,y,X)}}$$

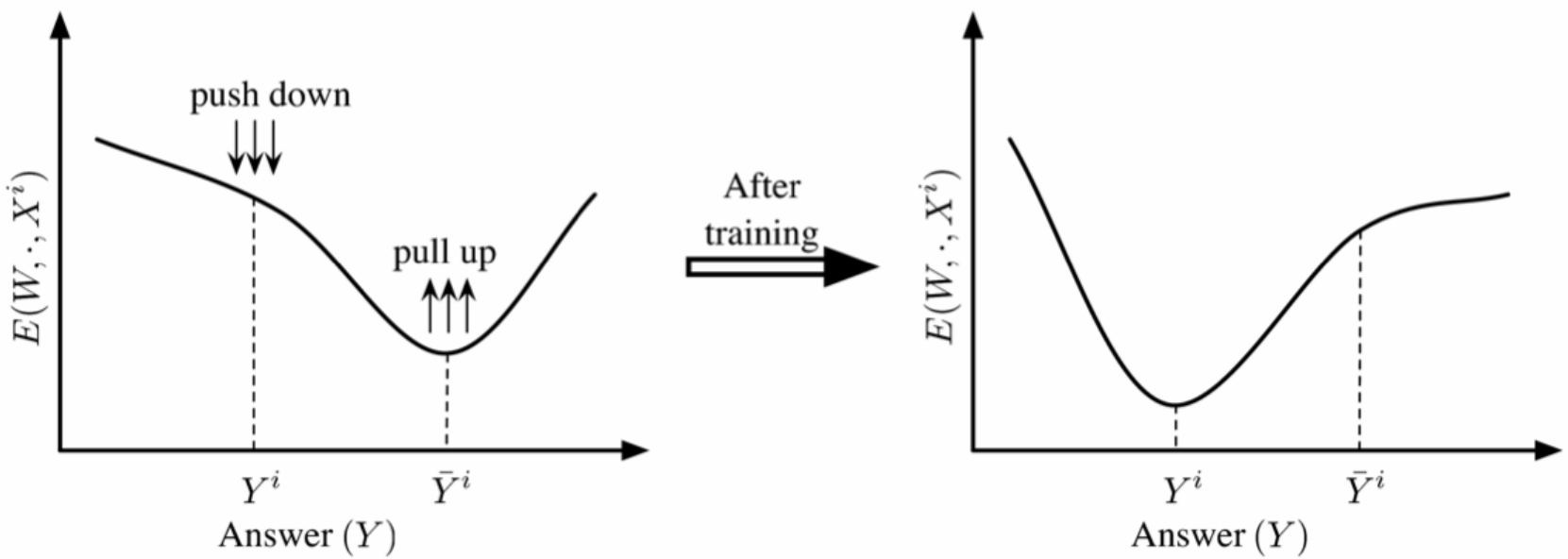
Energy-based models

Problems that can be addressed:

1. **Classification** $\rightarrow$ find Y that is most compatible with X
2. **Ranking** $\rightarrow$ decide whether Y_1 or Y_2 more compatible with X
3. **Detection** $\rightarrow$ decide whether Y is compatible with X
4. **Conditional Density Estimation** $\rightarrow$ compute $P(Y|X)$

How does learning work ?

- **Push down** on the energy of the correct answer
- **Pull up** on the energies of the incorrect answers, particularly if they are smaller than the correct one



Slide by Yann LeCun

Energy-based models

Given a dataset $\mathcal{D}$, the ingredients to be chosen are:

- a particular form of **energy** $E(W, Y, X)$
- an **inference algorithm** to find Y by minimizing the chosen energy E for any given X
- a **loss function** $\mathcal{L}(W, \mathcal{D})$ measuring the quality of E
- an **optimization method** for the loss function to find W, given $E, \mathcal{L}, \mathcal{D}$

The loss is typically designed as the sum of two terms:

$$\mathcal{L}(W, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} L(Y^i, E(W, \mathcal{Y}, X^i)) + R(W)$$

begin L a **per-loss** function and R a **regularization** term

Examples of **per-sample loss** function:

- Energy Loss
  $L(Y^i, E(W, \mathcal{Y}, X^i)) = E(W, Y^i, X^i)$
- Negative log-likelihood (or cross-entropy) Loss
  $L(Y^i, E(W, \mathcal{Y}, X^i)) = E(W, Y^i, X^i) + \frac{1}{\beta} \log \int_y e^{-\beta E(W, y, X^i)}$
- Hinge Loss
- Log Loss
- ...

Example

Minimize the **log-likelihood** on training examples
We can train the model with **gradient descent**:

$$\frac{\partial L(Y,W,X)}{\partial W} = \frac{\partial E(Y,W,X)}{\partial W} - \int_y P(y|W)\frac{\partial E(y,W,X)}{\partial W}$$

$$W \leftarrow W - \eta \frac{\partial L(Y,W,X)}{\partial W}$$

How to choose the ys to compute the second-term integral ?
One solution is given by **contrastive divergence** !

Boltzmann Machines have the following form:

$$E(v,h) = -\sum_i b_i v_i - \sum_j c_j h_j - \sum_i \sum_j v_i w_{ij} h_j$$
$$- \sum_i \sum_j v_i p_{ij} v_j - \sum_i \sum_j h_i q_{ij} h_j$$

In this case, free energy cannot be analytically computed, **which makes learning impractical** !

In RBMs, the additional assumption of **graph factorization** allows to drop the last two terms (no intra-layer links)

RBMs are universal approximators

Provided enough hidden units, an RBM can perfectly model any discrete distribution

Adding one hidden unit guarantess to **increase likelihood**, provided a **proper choice of parameters**...

Deep Boltzmann Machines are a specific case of BMs corresponding to an undirected graphical model.

Each layer is still an RBM (as in a DBN), but **the energy function is different**:

$$E(v, h^1, h^2, h^3; \theta) = -v^T W^1 h^1 - {h^1}^T W^2 h^2 - {h^2}^T W^3 h^3$$

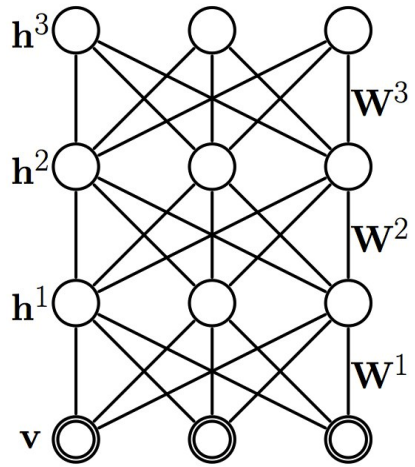


Figure by [R. Salakhutdinov & G. Hinton, 2009]

Energy-based models

It should be remarked that **a DBN is not a DBM** !

- In a DBN, there is only a **bottom-up** phase
- In a DBM, a **bottom-up** and a **top-down** phase are combined

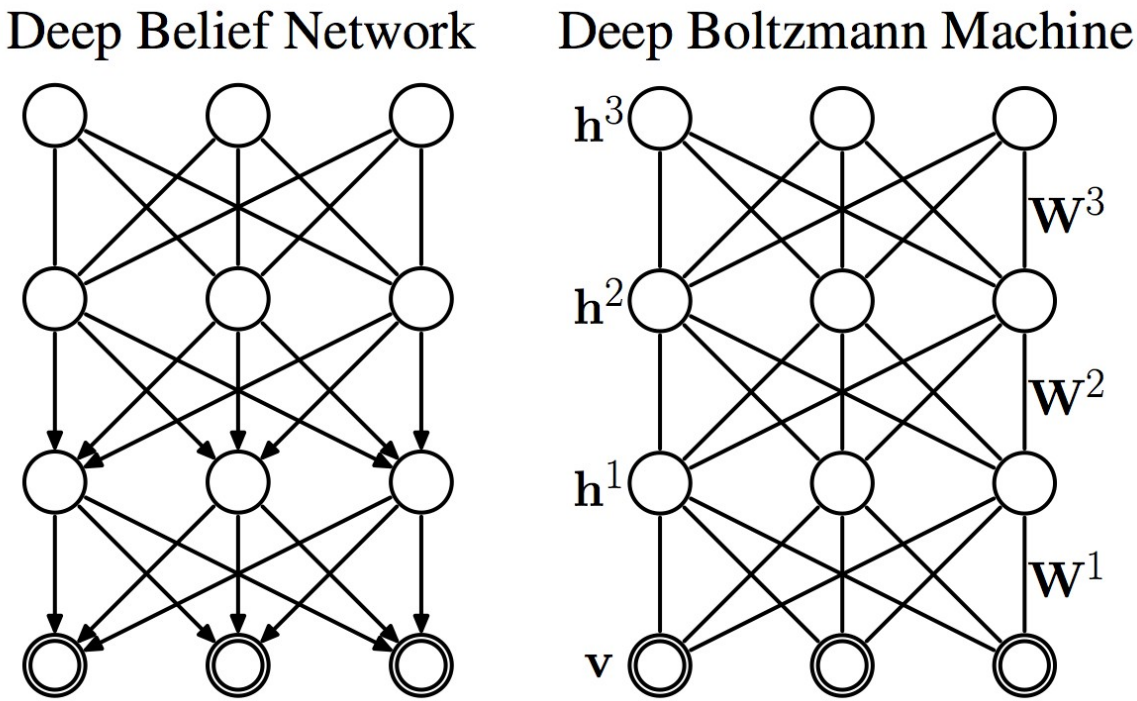


Figure by [R. Salakhutdinov & G. Hinton, 2009]

Energy-based models

Training a DBM:

- quite **hard** with classic **maximum likelihood**
- could be done with **layer-wise pre-training** as for DBNs
- a joint training exploiting **regularization** has been proposed in [Desjardins et al., 2012]

Results (image/speech recognition):

- DBNs are easier to train
- DBMs might achieve better performance

RBMs with Gaussian visible units (GRBMs)

- when dealing with **real-valued input**
- natural images, speech, ...
- learning more difficult as reconstruction is **unbounded** !
- got **unsatisfactory results**...

$$E(v,h) = \sum_{i \in vis} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{j \in hid} c_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij}$$

Hint for training: typically one should first normalize the input

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

RBMs with Gaussian visible and hidden units

- Learning becomes even more difficult

$$E(v,h) = \sum_{i \in vis} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{j \in hid} \frac{(h_j - c_j)^2}{2\sigma_j^2} - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij}$$

Mean-covariance RBM (mcRBM)

- Explicitly modeling **mean** and **covariance** of input elements
- This is captured by **two distinct hidden groups**

$$E(v, h^m, h^c) = E^c(v, h^c) + E^m(v, h^m)$$

Spike and slab RBM (ssRBM)

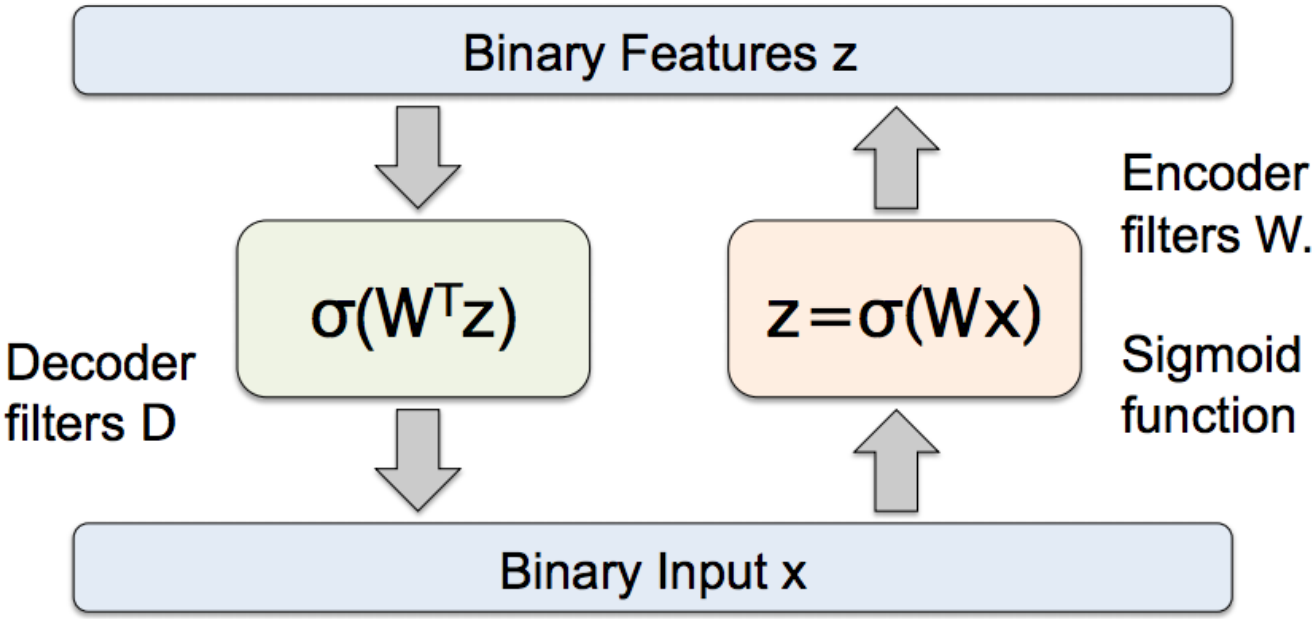
Each **hidden** neuron is associated with:

- a binary variable (**spike**) h_i
- a real-valued vector (**slab**) s_i

$$E(v, s, h) = \frac{1}{2} v^T B v - \sum_{i=1}^{N} (v^T W s_i h_i + \frac{1}{2} s_i^T \alpha_i s_i + c_i h_i)$$

Auto-Encoders

Different Auto-Encoders with different encoding/decoding blocks...



[Figure by R. Salakhutdinov]

Example: try to minimize the **reconstruction error**

$$E(W) = \frac{1}{2} \sum_{n=1}^{N} \|\sigma(W^T z_n) - x_n\|^2$$

Is it going to **learn the identity function** ?
Do we need any additional constraint ?

Note

Minimizing reconstruction error corresponds to maximizing the **mutual information** between input X and learnt representation Y

Auto-Encoders

If # hidden units is **smaller** than # input units:

- **feature selection**
- **dimensionality reduction**

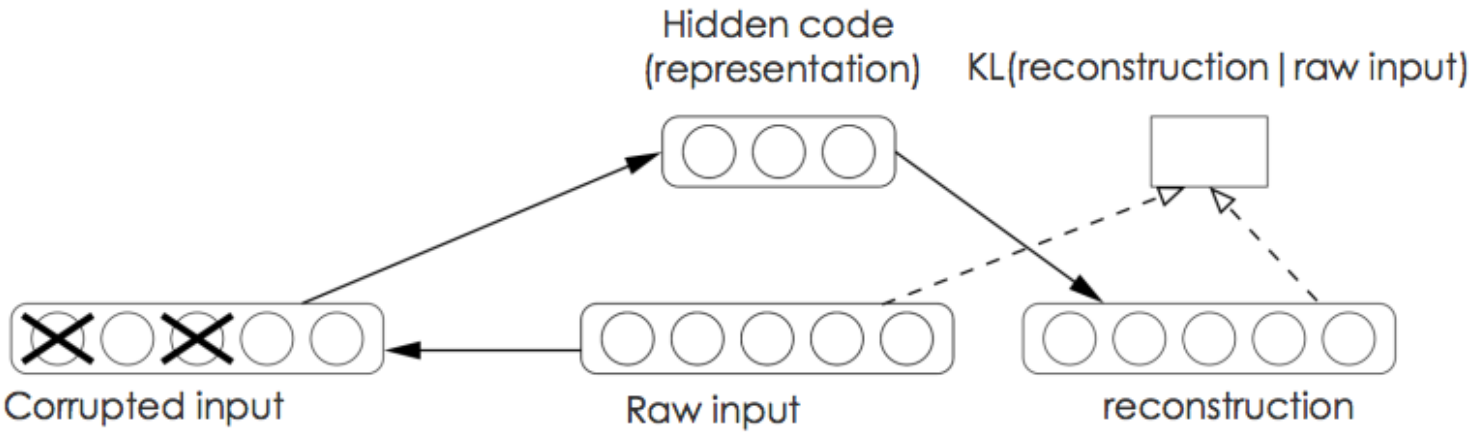
If # hidden units is **greater** than # input units (over-complete):

- typically induce **sparsity** in parameters
- biological inspiration for sparse **connectivity**
- with larger spaces units are **less entangled**
- dealing with **larger feature space** might be easier

There exist several types of Auto-Encoders

- **Ordinary** Auto-Encoders
  $\min_h \|x - \sigma(Wh)\|_2^2$
- **Sparse** Auto-Encoders
  $\min_h \|x - \sigma(Wh)\|_2^2 + \lambda\|h\|_1$
- **Denoising** Auto-Encoders
  $\min_h \|x - \sigma(Wh)\|_2^2$ where $h = \sigma(W^T\tilde{x})$

Denoising Auto-Encoders (DAEs)



[Figure by Larochelle et al., 2010]

DAEs are trained to reconstruct **stochastically corrupted** input, with uncorrupted input still **used as target**

Denoising Auto-Encoders (DAEs)

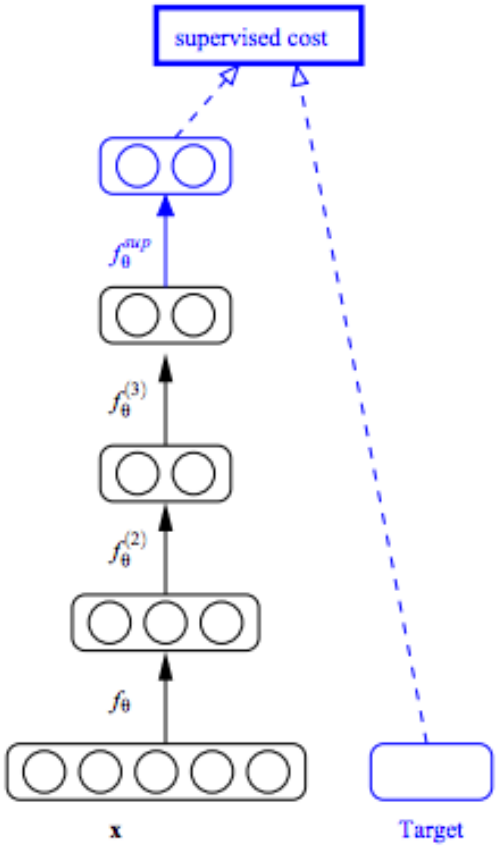
Therefore, a DAE has two intertwined goals:

1. encoding and recostructing the input
2. removing the effect of noise

The training criterion is a reconstruction log-likelihood:

$$-\log P(x|c(\tilde{x}))$$

Stacked Denoising Auto-Encoders (SDAEs)



[Figure by Larochelle et al., 2010]

Very similar to DBNs:

- Greedy layer-wise training
- Bottom-up stacking
- Fine-tuning on top

DAEs and overfitting

By adding noise to the input units of the network, DAEs:

- **perform a sort of regularization**
- make the network **more robust** to noise
- basically, they **reduce overfitting**

If there were no problem of computational requirements, overfitting could be reduced by **averaging over a set of models**.

This brings to the idea of **dropout** !

Tricks of the Trade

Ensemble learning

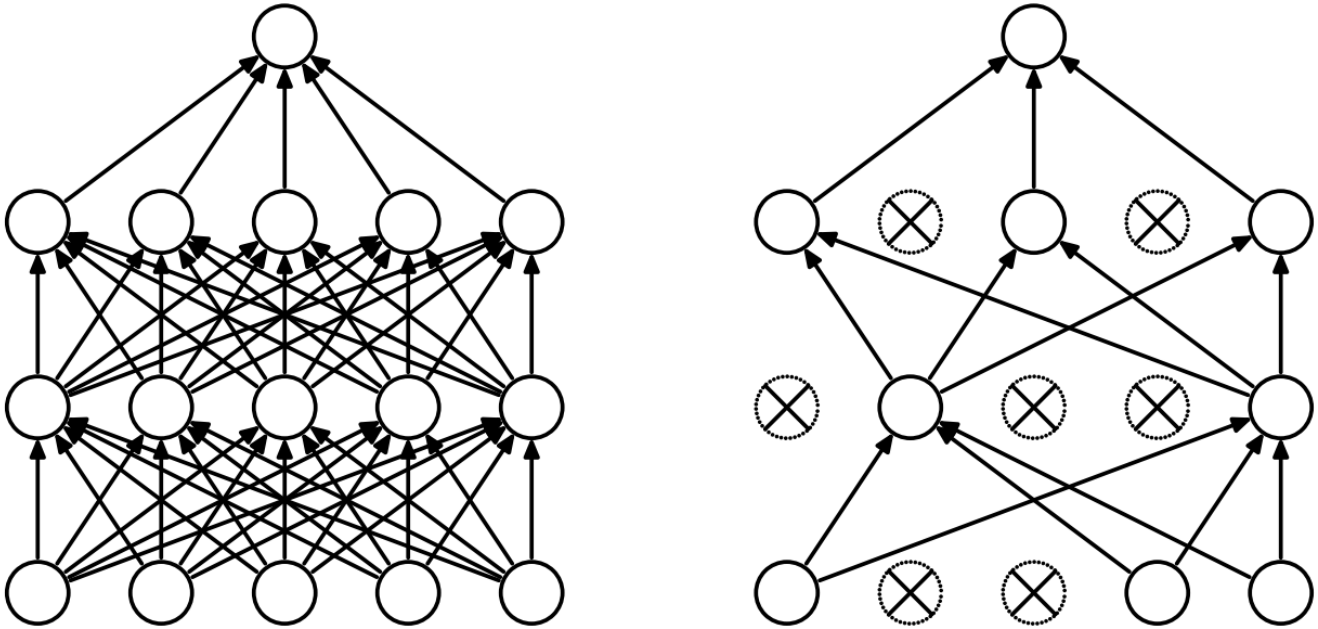
Observation

Putting together a collection of classifiers typically improves performance (e.g., see **bagging** and **boosting**)

Training a large number of deep neural networks and then combining their outputs would be **computationally unfeasible**

Dropout

The key idea of dropout is to **randomly drop units and their connections** during training.



(a) Standard Neural Net (b) After applying dropout.

[Figure by Srivastava et al., 2014]

Dropout

The mechanism transforms a deep network in a **thinned** one.

Observation

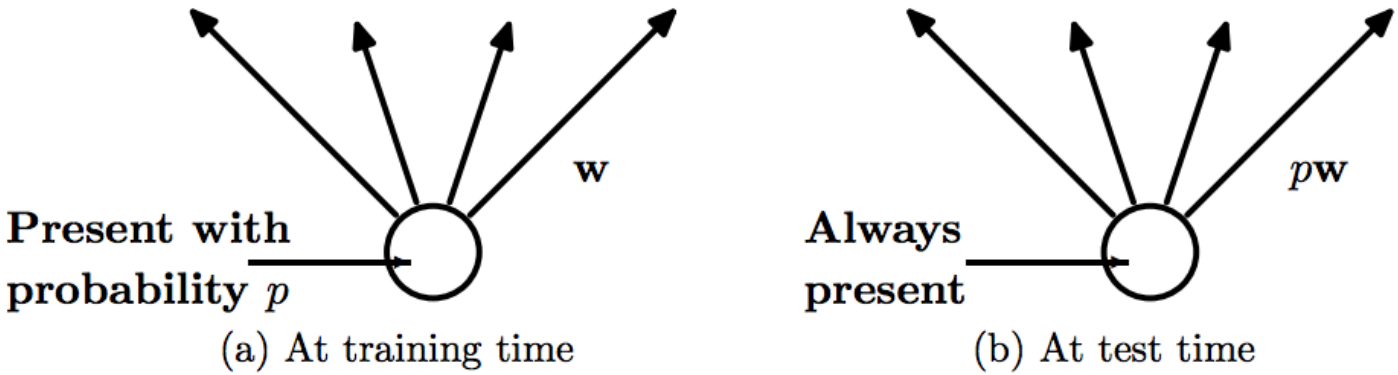
A neural network with n units can be seen as the **combination** of 2^n possible thinned neural neural networks

- Combining classifiers almost **always** improves performance
- Training different architectures with different parameters and/or inputs would be **too expensive** !

Dropout

Ok, dropping units and connections during **training**...
...But what about **test** ? Which network shall we use ?

- Use a **single** network without dropout
- **Re-scale weights** by the dropout ratio



[Figure by Srivastava et al., 2014]

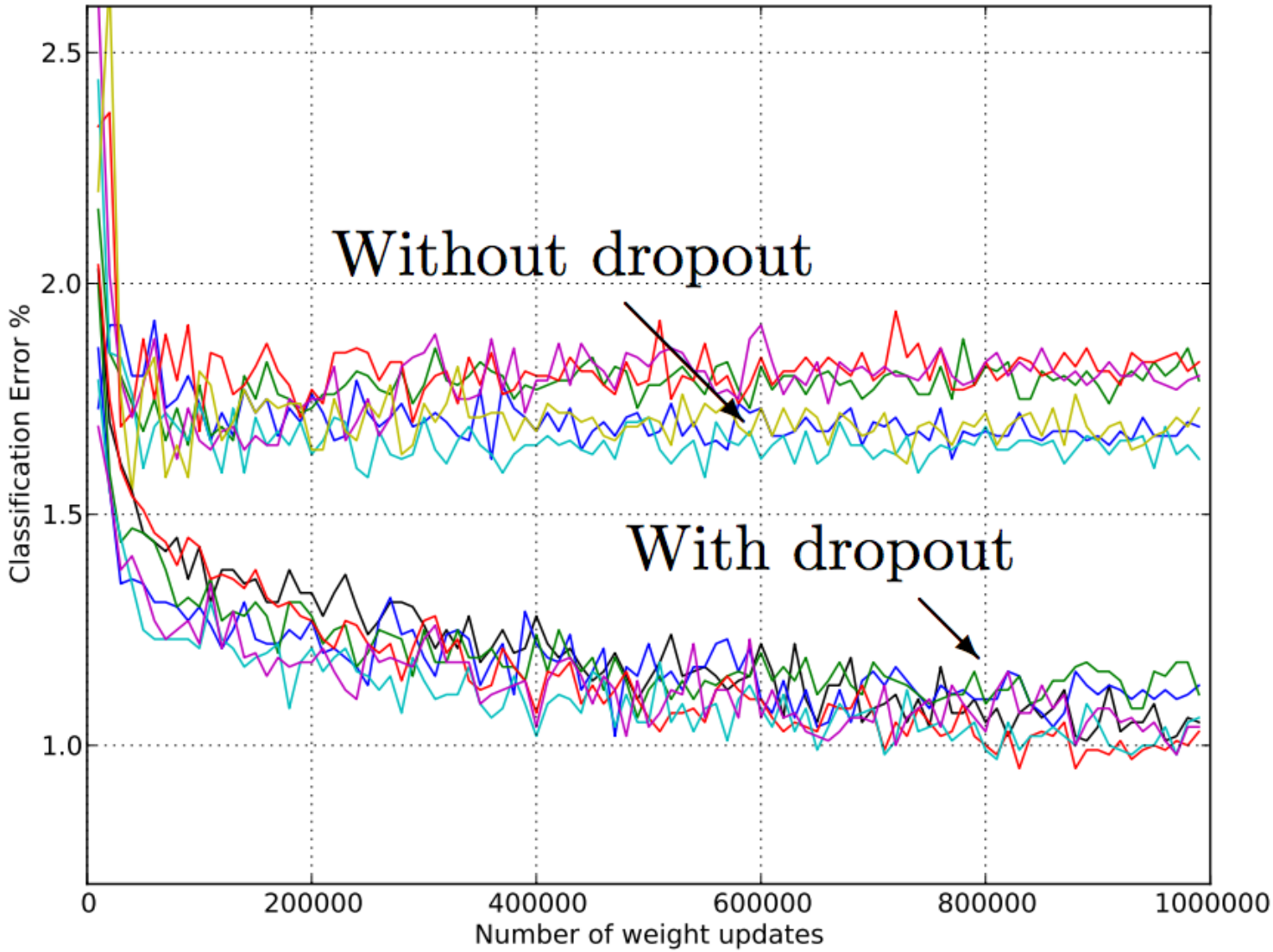
It can be applied to any deep network, thus also to DBNs

Data Set	Domain	Dimensionality	Training Set	Test Set
MNIST	Vision	784 (28 × 28 grayscale)	60K	10K
SVHN	Vision	3072 (32 × 32 color)	600K	26K
CIFAR-10/100	Vision	3072 (32 × 32 color)	60K	10K
ImageNet (ILSVRC-2012)	Vision	65536 (256 × 256 color)	1.2M	150K
TIMIT	Speech	2520 (120-dim, 21 frames)	1.1M frames	58K frames
Reuters-RCV1	Text	2000	200K	200K
Alternative Splicing	Genetics	1014	2932	733

[Table by Srivastava et al., 2014]

- Impact measured on performance on several tasks
- Dropout networks constantly outperform those without

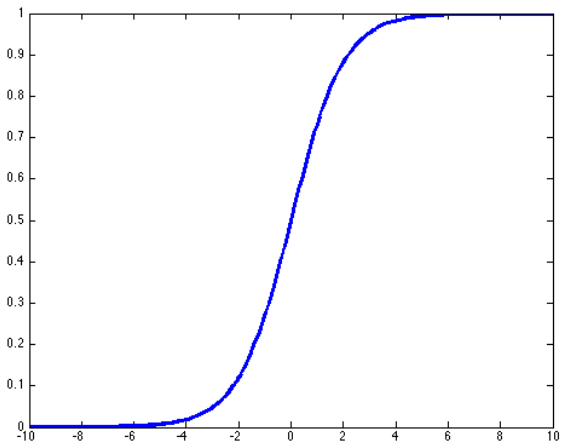
Dropout



[Figure by Srivastava et al., 2014]

Sigmoid

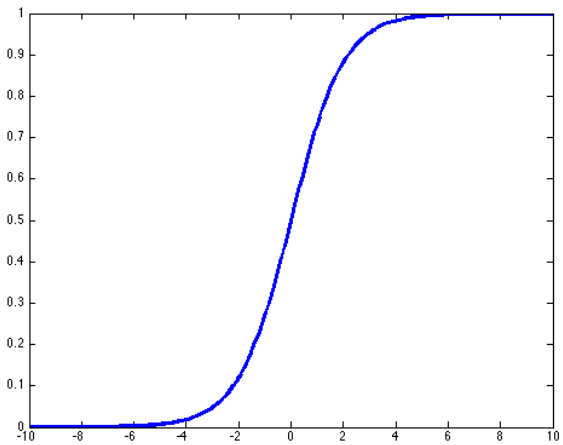
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



- Extremely popular since the pioneering ANN works
- Interpretation in terms of **saturation firing rate** of a neuron

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



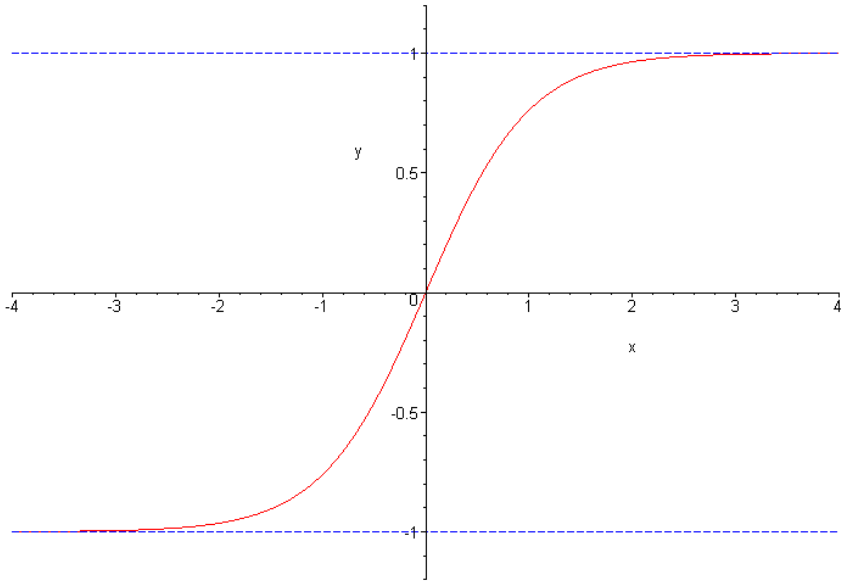
Problems:

- Saturation zones kill the gradient
- Output is not zero-mean
- exp() computationally expensive

Tanh

$$\sigma(x) = tanh(x)$$

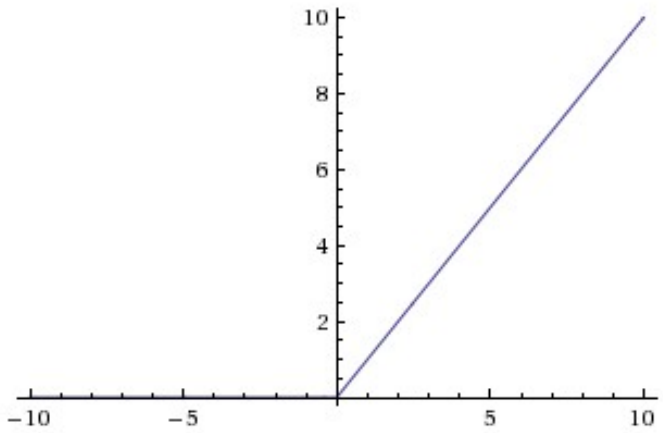
- It is zero-centered
- Still has saturation zones



Rectified Linear Unit (ReLU):

$$f(x) = max(0, x)$$

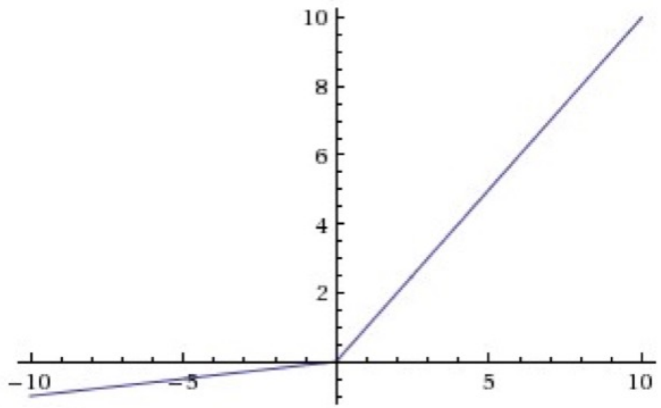
- more **biologically plausible**
- **faster** to compute
- **no vanishing gradient** problem
- **sparsity** of the solution (few neurons activated)
- suffers of a **dead zone**



Leaky ReLU:

$$f(x) = max(0.01x, x)$$

- **some gradient** also for the negative side



Other issues

- Learning rate decay
- Mini-batch dimension
- Momentum
- Dropout rate
- ...

See "Practical Recommendations for Gradient-Based Training of Deep Architectures" by Yoshua Bengio (2012)