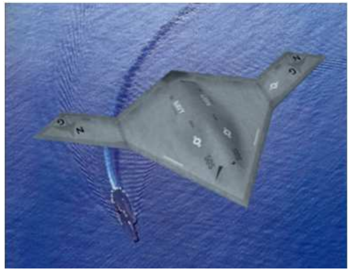
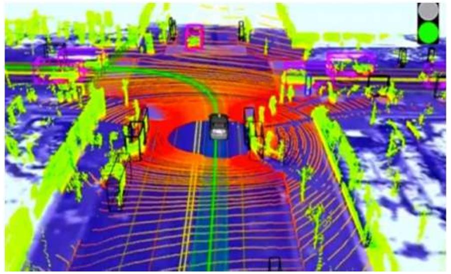
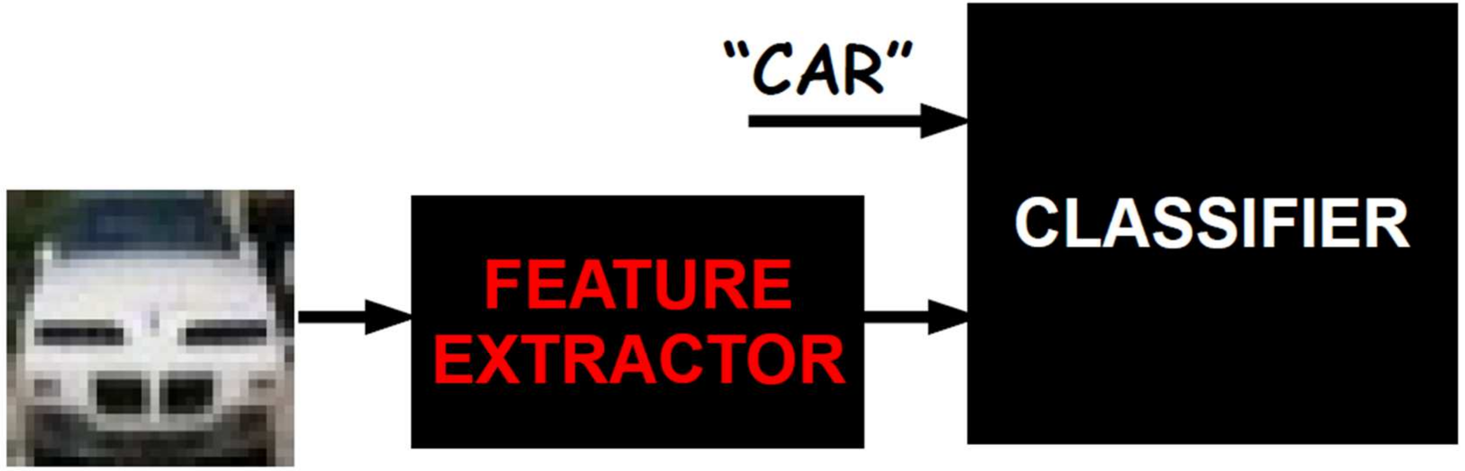
Deep Learning
Restricted Boltzmann Machine

布树辉 bushuhui@nwpu.edu.cn
http://www.adv-ci.com

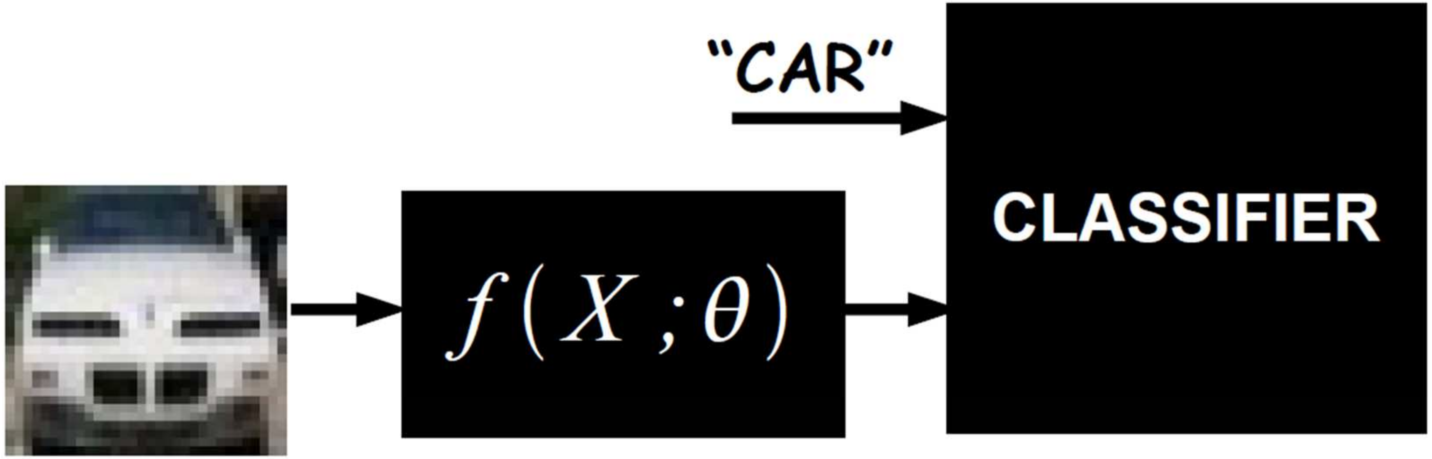



Building an object recognition system



IDEA: Use data to optimize features for the given task.

Building an object recognition system

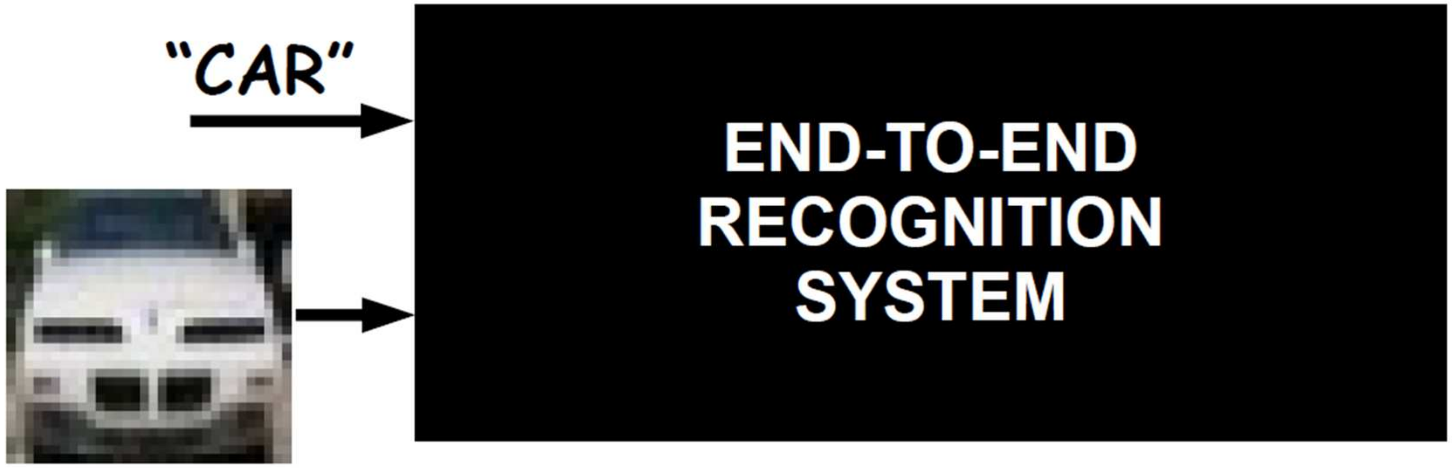


What we want: Use parameterized function such that

a) features are computed efficiently

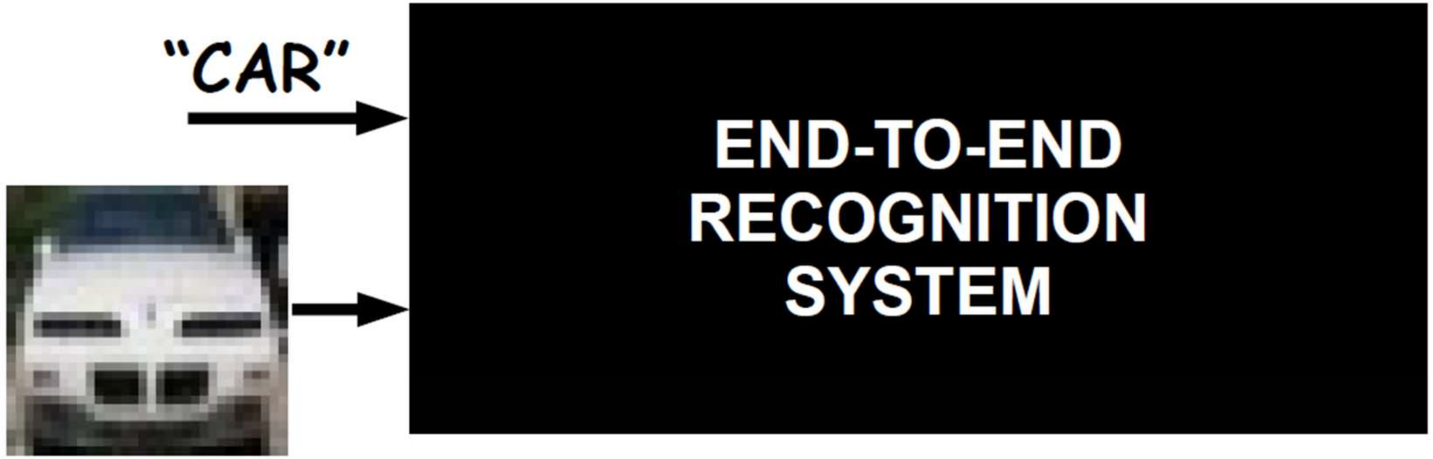
b) features can be trained efficiently

Building an object recognition system



- Everything becomes adaptive.
- No distiction between feature extractor and classifier.
- Big non-linear system trained from raw pixels to labels.

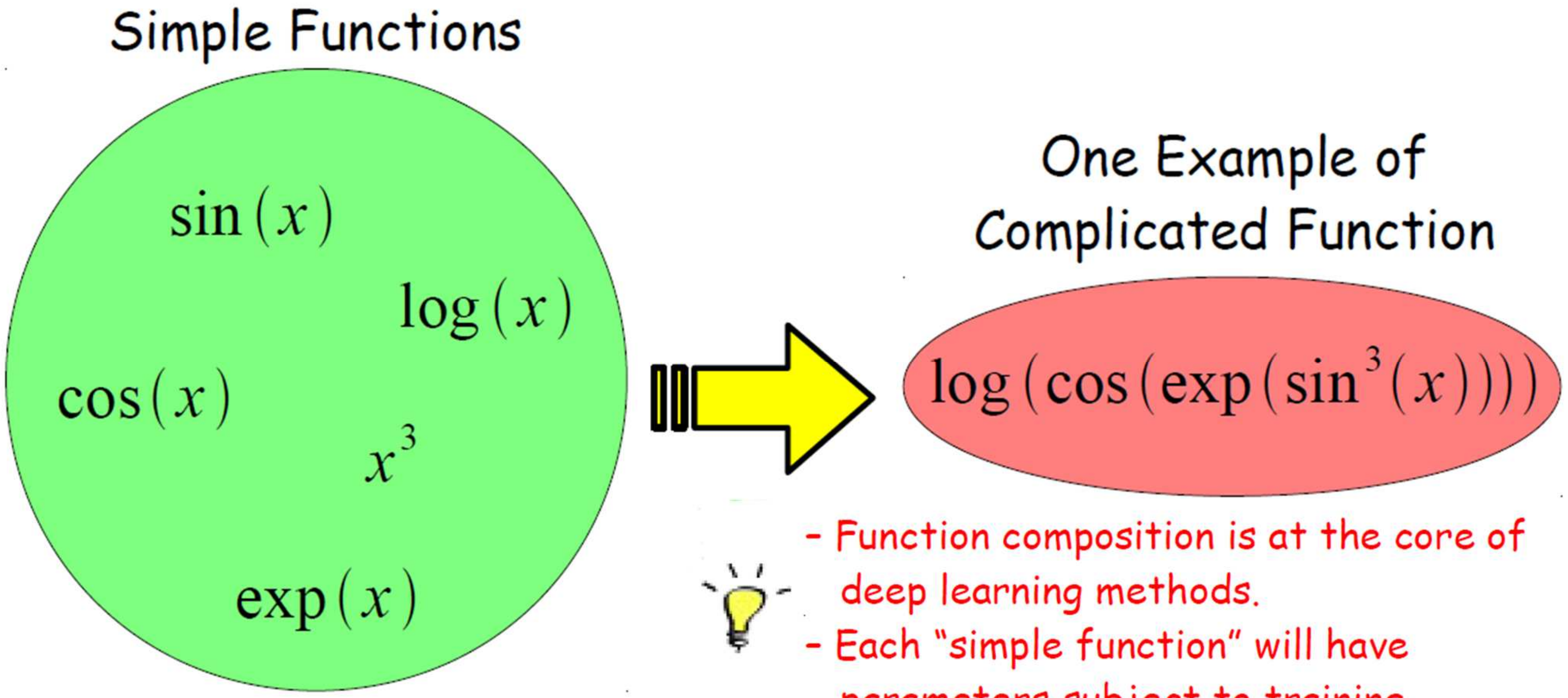
Building an object recognition system



Q: How can we build such a highly non-linear system?

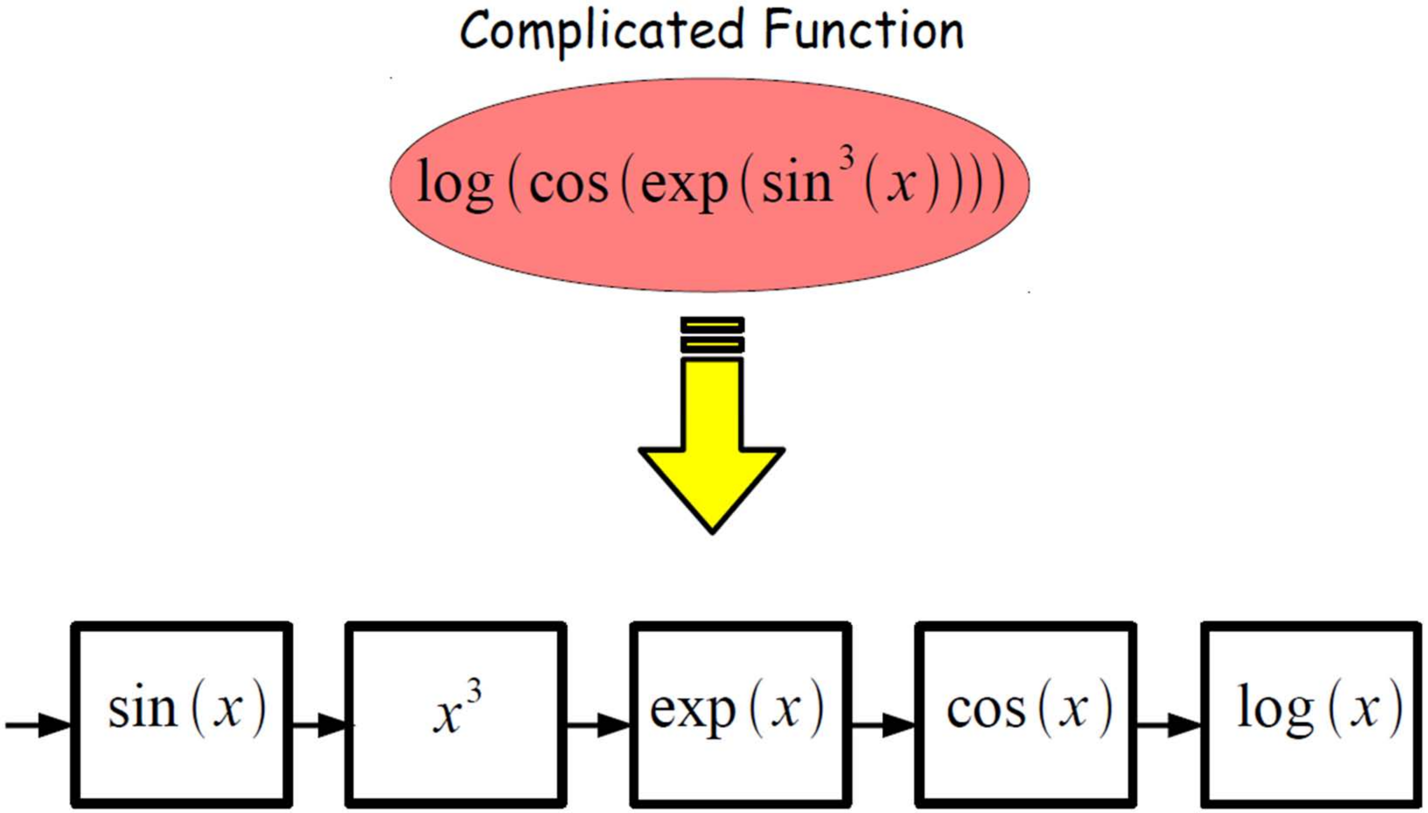
A: By combining simple building blocks we can make more and more complex systems.

Building a complicated function

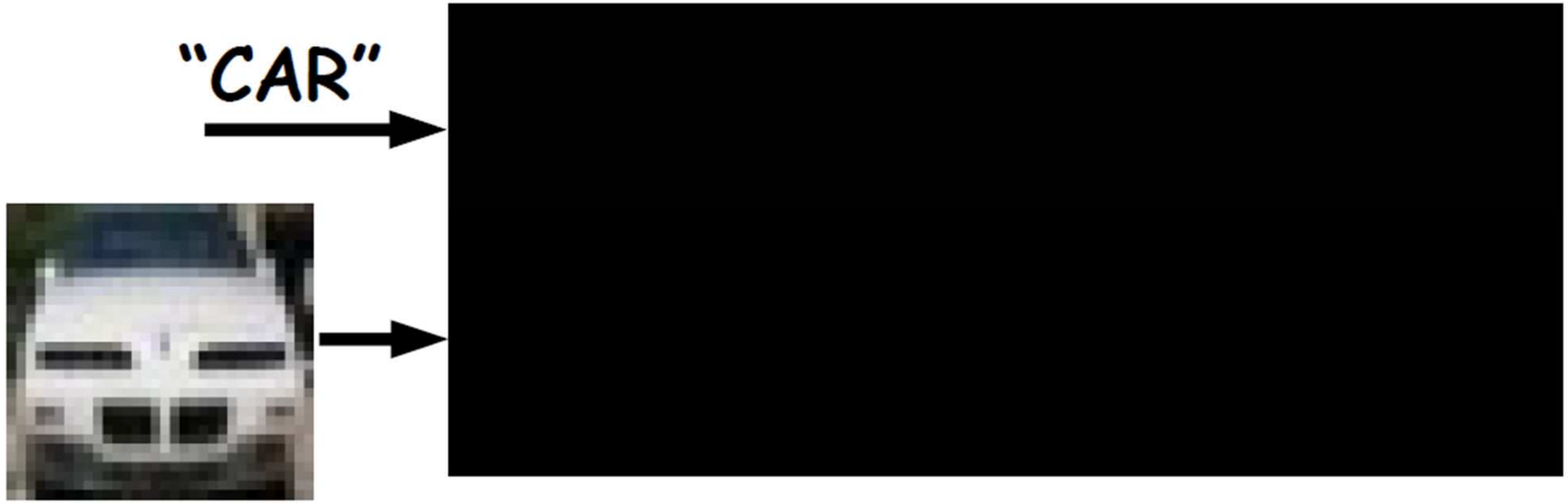

Simple Functions
$\sin(x)$
$\log(x)$
$\cos(x)$
x^3
$\exp(x)$
One Example of Complicated Function
$\log(\cos(\exp(\sin^3(x))))$

- Function composition is at the core of deep learning methods.
- Each "simple function" will have parameters subject to training.

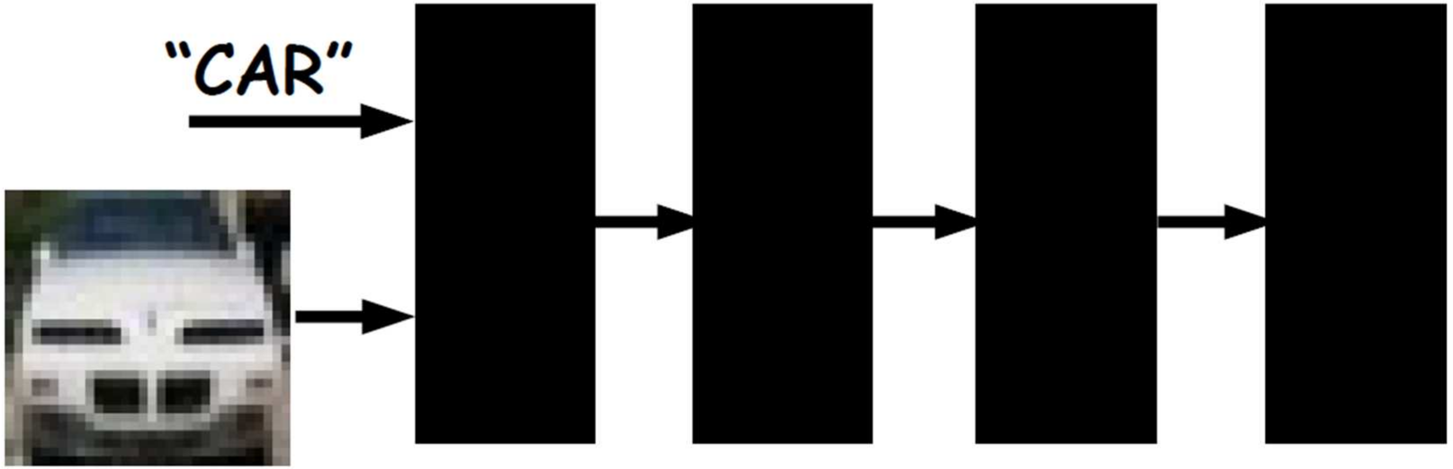
Implementing a complicated function



Intuition behind deep neural network

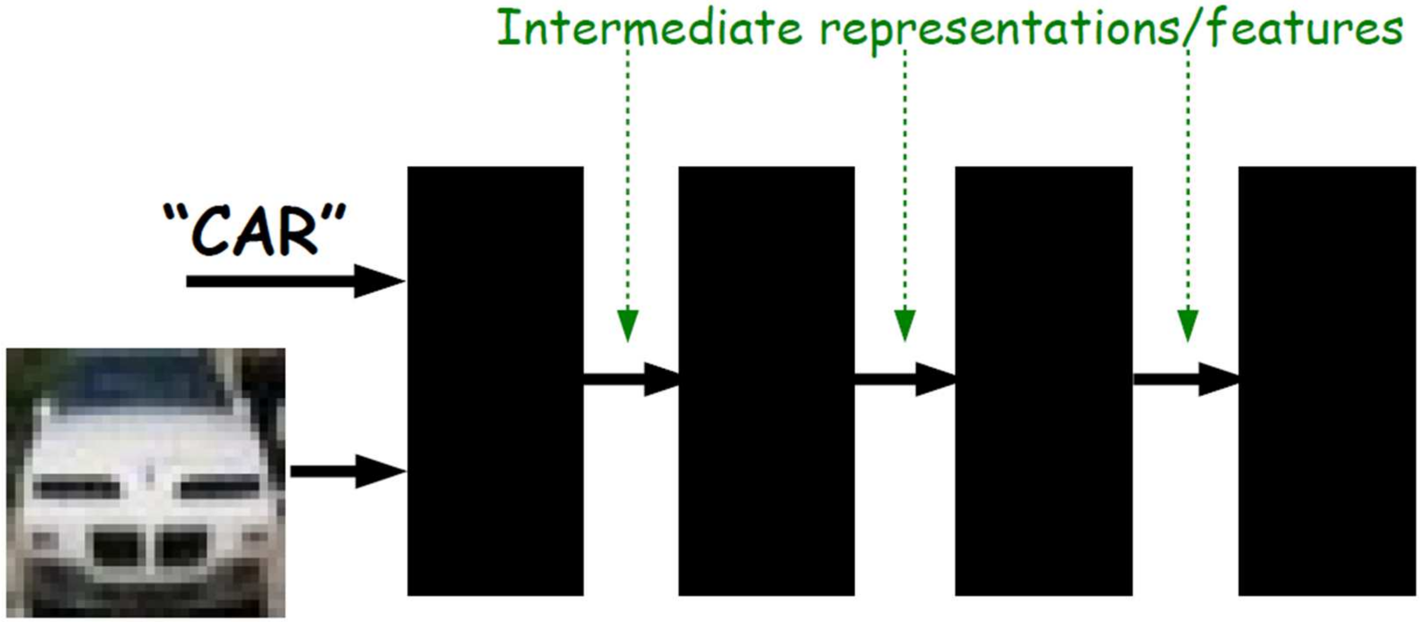


Intuition behind deep neural network



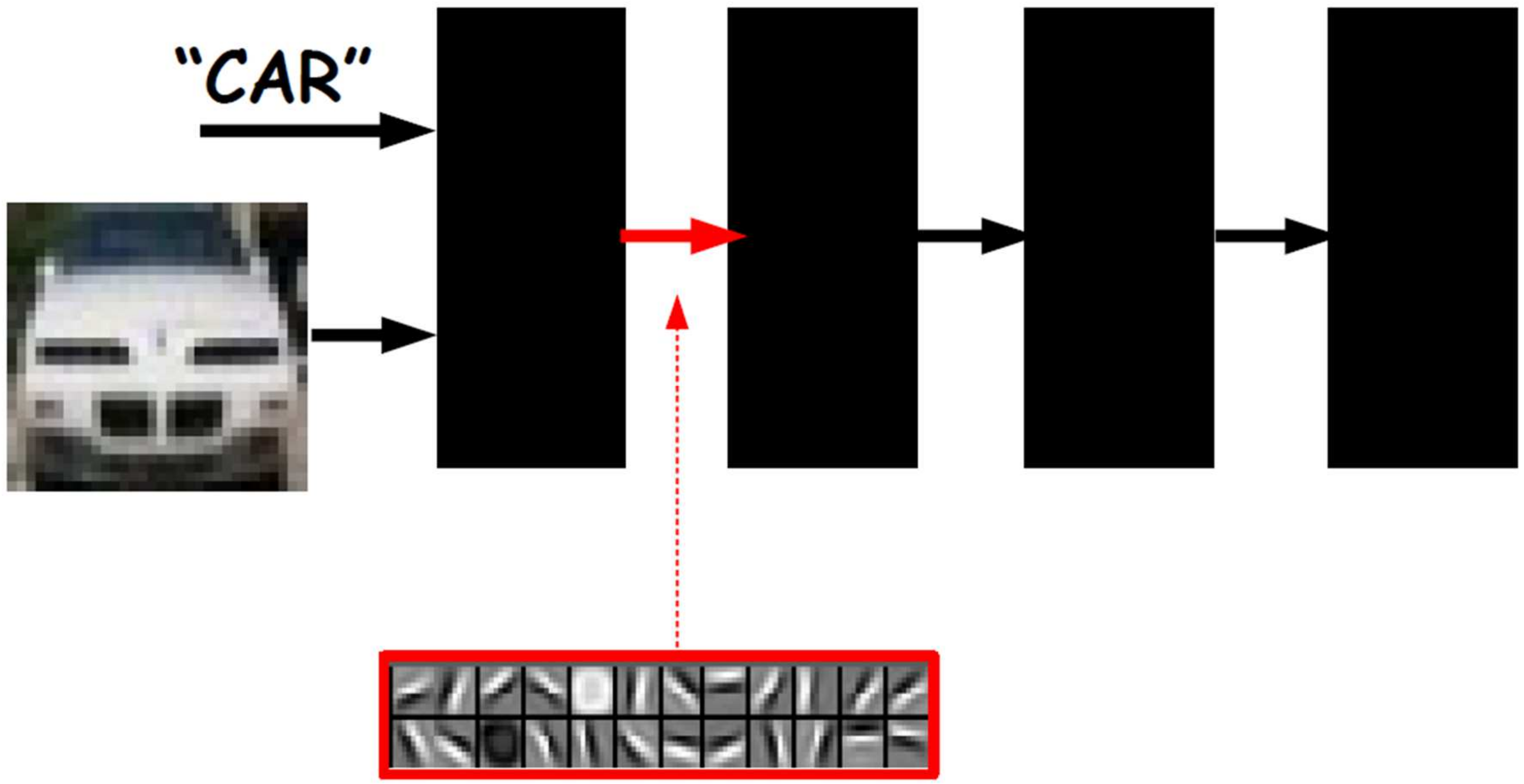
NOTE: Each black box can have trainable parameters.
Their composition makes a highly non-linear system.

Intuition behind deep neural network

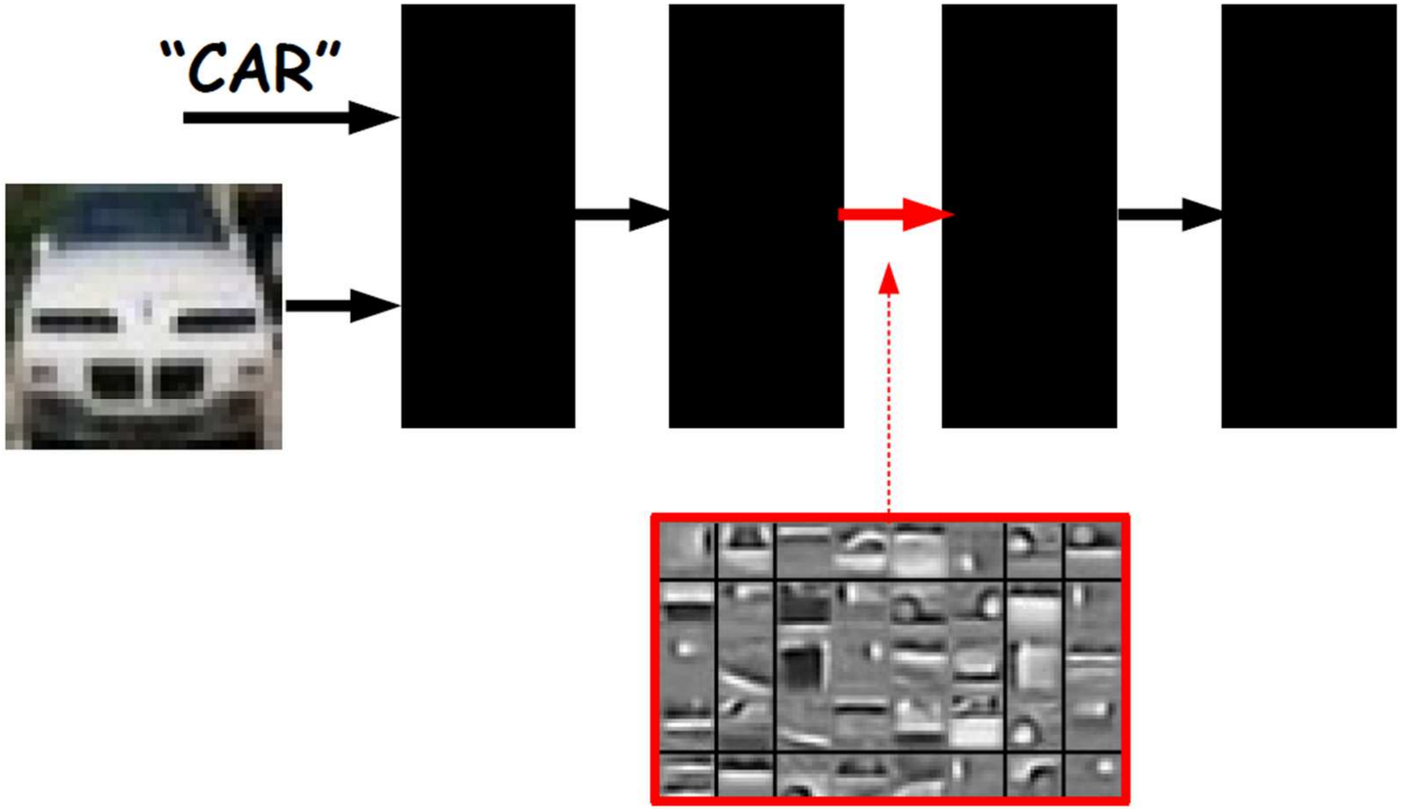


NOTE: System produces a hierarchy of features.

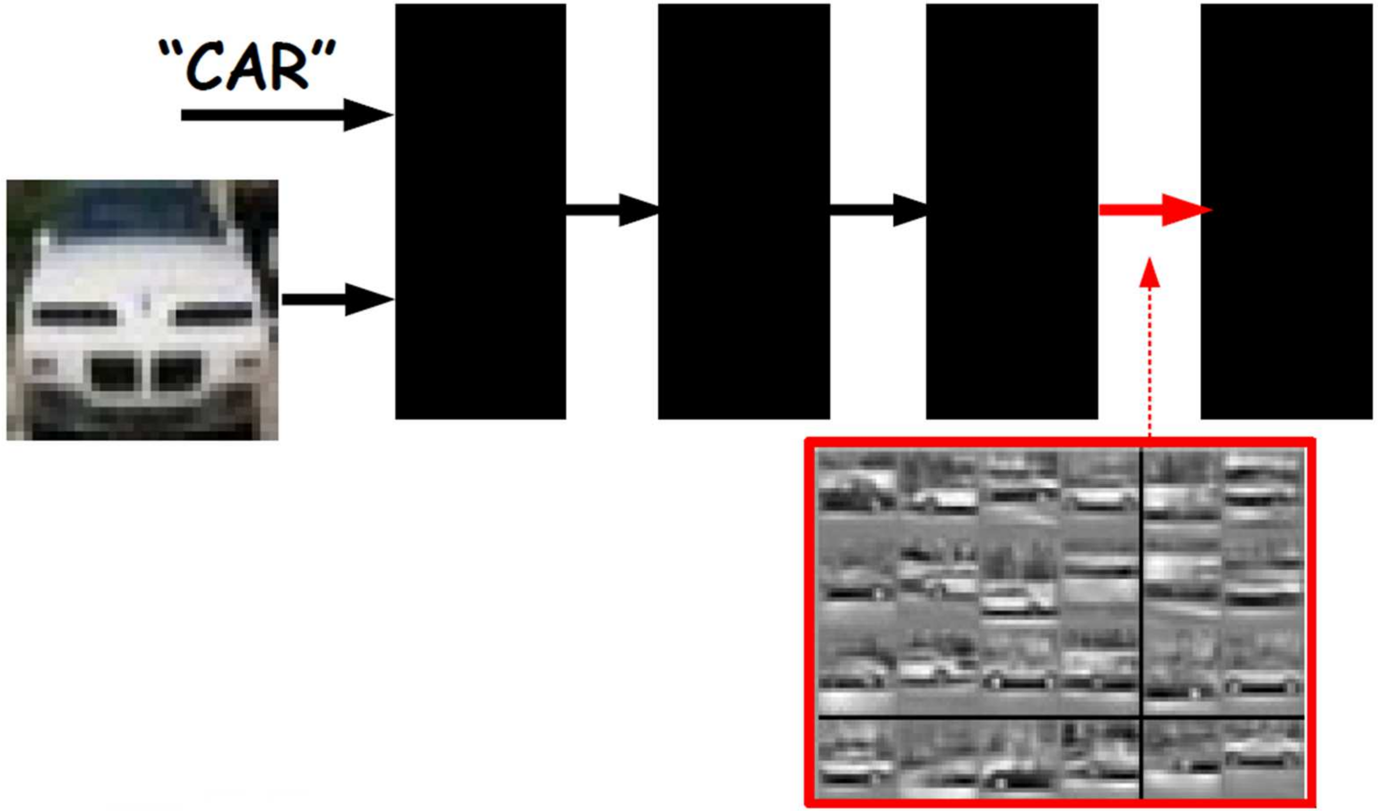
Intuition behind deep neural network



Intuition behind deep neural network



Intuition behind deep neural network



Key ideas of neural networks

IDEA # 1

Learn features from data

IDEA # 2

Use differentiable functions that produce features efficiently

IDEA # 3

End-to-end learning:
no distinction between feature extractor and classifier

IDEA # 4

"Deep" architectures:
cascade of simpler non-linear modules

Key questions

- What is the input-output mapping?

- How are parameters trained?

- How computational expensive is it?

- How well does it work?

Contents

- How to learn multi-layer generative models of unlabelled data by learning one layer of features at a time.
  - How to add Markov Random Fields in each hidden layer.
- How to use generative models to make discriminative training methods work much better for classification and regression.
  - How to extend this approach to Gaussian Processes and how to learn complex, domain-specific kernels for a Gaussian Process.
- How to perform non-linear dimensionality reduction on very large datasets
  - How to learn binary, low-dimensional codes and how to use them for very fast document retrieval.
- How to learn multilayer generative models of high-dimensional sequential data.

A spectrum of machine learning tasks

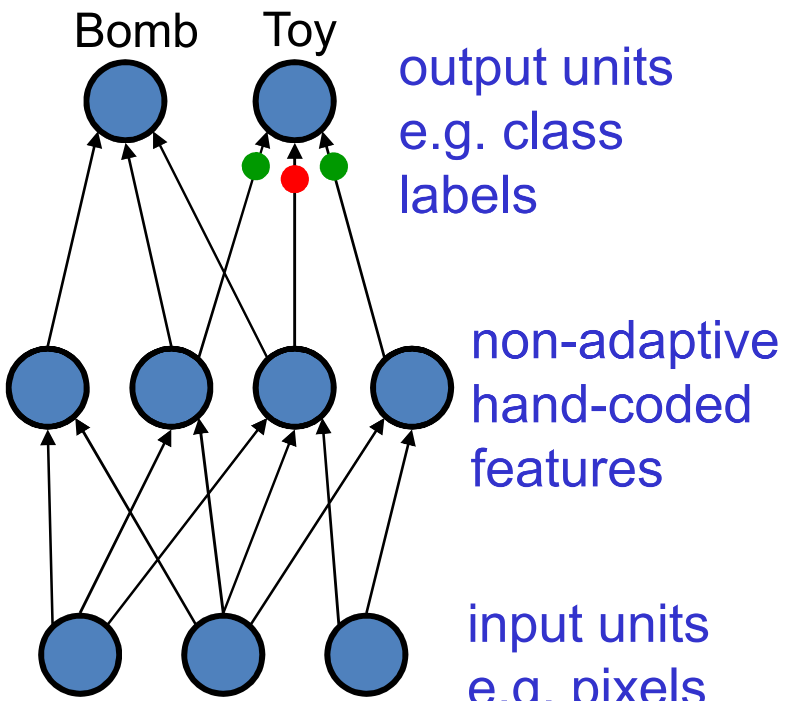
Typical Statistics ------------ Artificial Intelligence

- Low-dimensional data (e.g. less than 100 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.
- The main problem is distinguishing true structure from noise.

- High-dimensional data (e.g. more than 100 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model.
- The main problem is figuring out a way to represent the complicated structure so that it can be learned.

Historical background: First generation neural networks

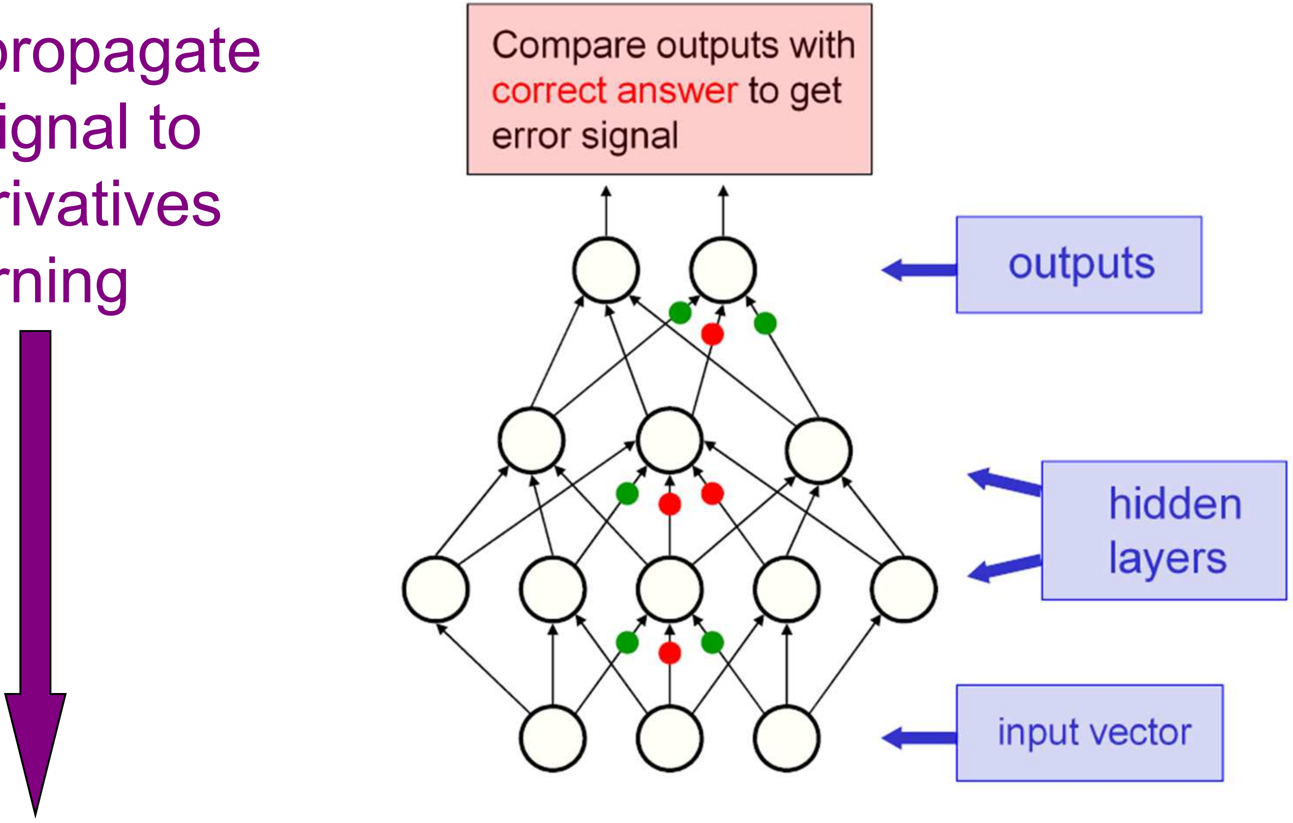
- Perceptrons (~1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features.
  - There was a neat learning algorithm for adjusting the weights.
  - But perceptrons are fundamentally limited in what they can learn to do.



Sketch of a typical perceptron from the 1960's

Second generation neural networks (~1985)

Back-propagate error signal to get derivatives for learning



A temporary digression

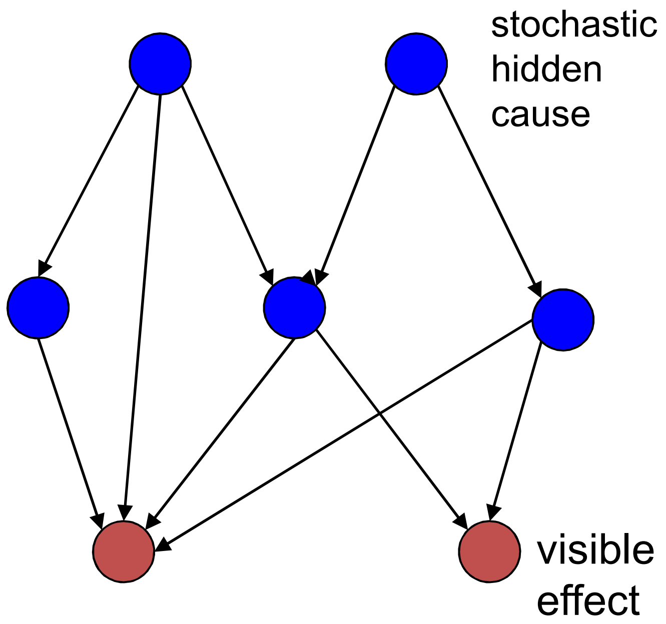
- Vapnik and his co-workers developed a very clever type of perceptron called a Support Vector Machine.
  - Instead of hand-coding the layer of non-adaptive features, each training example is used to create a new feature using a fixed recipe.
    - The feature computes how similar a test example is to that training example.
  - Then a clever optimization technique is used to select the best subset of the features and to decide how to weight each feature when classifying a test case.
    - But its just a perceptron and has all the same limitations.
- In the 1990's, many researchers abandoned neural networks with multiple adaptive hidden layers because Support Vector Machines worked better.

What is wrong with back-propagation?

- It requires labeled training data.
  - Almost all data is unlabeled.
- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.

Belief Networks

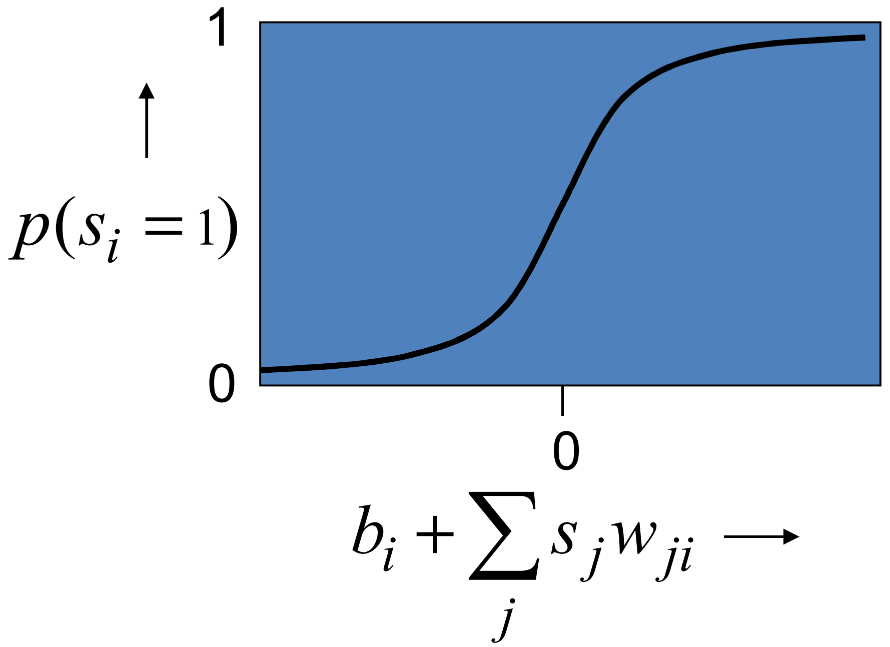
- A belief net is a directed acyclic graph composed of stochastic variables.
- We get to observe some of the variables and we would like to solve two problems:
- The inference problem: Infer the states of the unobserved variables.
- The learning problem: Adjust the interactions between variables to make the network more likely to generate the observed data.



We will use nets composed of layers of stochastic binary variables with weighted connections. Later, we will generalize to other types of variable.

Stochastic binary units (Bernoulli variables)

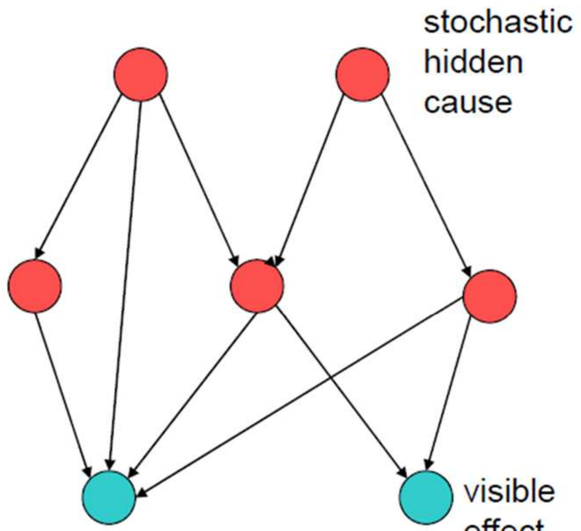
- These have a state of 1 or 0.
- The probability of turning on is determined by the weighted input from other units (plus a bias)



$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_j s_j w_{ji})}$$

Learning Deep Belief Nets

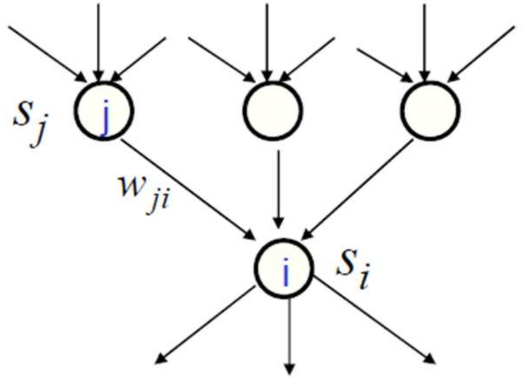
- It is easy to generate an unbiased example at the leaf nodes, so we can see what kinds of data the network believes in.
- It is hard to infer the posterior distribution over all possible configurations of hidden causes.
- It is hard to even get a sample from the posterior.
- So how can we learn deep belief nets that have millions of parameters?



The learning rule for sigmoid belief nets

- Learning is easy if we can get an unbiased sample from the posterior distribution over hidden states given the observed data.

- For each unit, maximize the log probability that its binary state in the sample from the posterior would be generated by the sampled binary states of its parents.



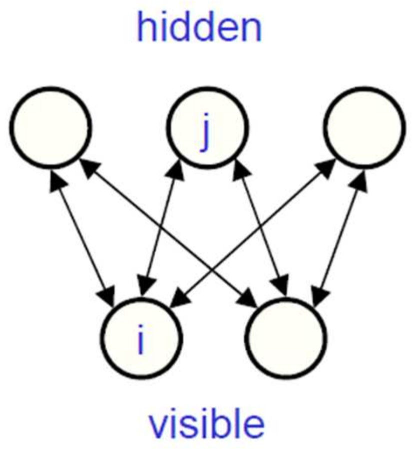
$$p_i \equiv p(s_i = 1) = \frac{1}{1 + \exp(-\sum_j s_j w_{ji})}$$

$$\Delta w_{ji} = \varepsilon \, s_j (s_i - p_i)$$

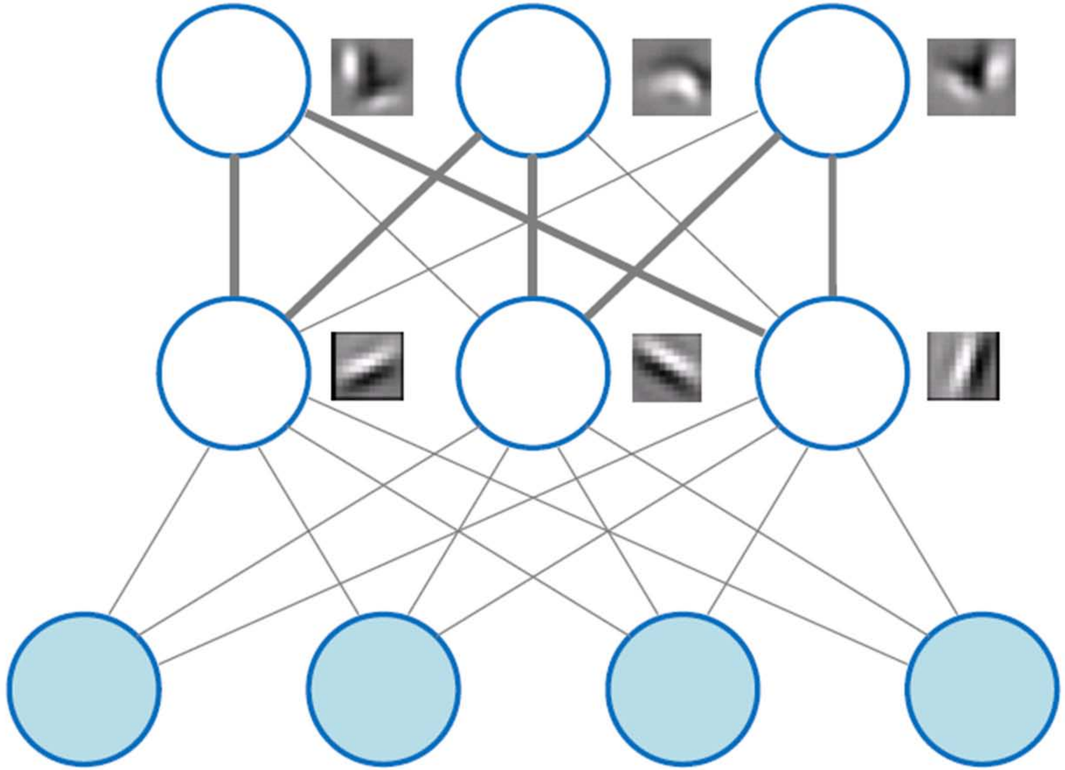
⇧
learning rate

Restricted Boltzmann Machines

- We restrict the connectivity to make learning easier.
  - Only one layer of hidden units.
    - We will deal with more layers later
  - No connections between hidden units.
- In an RBM, the hidden units are conditionally independent given the visible states.
  - So we can quickly get an unbiased sample from the posterior distribution when given a data-vector.
  - This is a big advantage over directed belief nets



Learning feature hierarchy



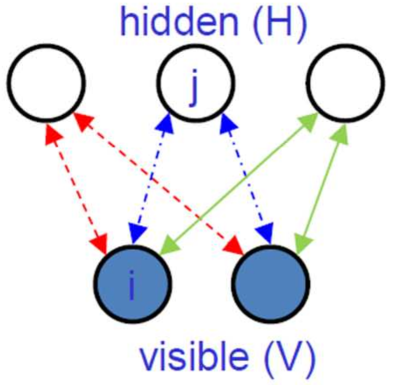
Higher layer: DBNs (Combinations of edges)

First layer: RBMs (edges)

Input image patch (pixels)

Restricted Boltzmann Machines with binary-valued input data

- Representation
  - Undirected bipartite graphical model
  - $\mathbf{v} \in \{0,1\}^D$: observed (visible) binary variables
  - $\mathbf{h} \in \{0,1\}^K$: hidden binary variables.



$$P(\mathbf{v},\mathbf{h}) = \frac{1}{Z}\exp(-E(\mathbf{v},\mathbf{h}))$$

$$E(\mathbf{v},\mathbf{h}) = -\sum_{ij} v_i W_{ij} h_j - \sum_j b_j h_j - \sum_i c_i v_i$$

$$= -\mathbf{v}^T W \mathbf{h} - \mathbf{b}^T \mathbf{h} - \mathbf{c}^T \mathbf{v}$$

$$Z = \sum_{\mathbf{v}\in\{0,1\}^D} \sum_{\mathbf{h}\in\{0,1\}^K} \exp(-E(\mathbf{v},\mathbf{h}))$$

The energy of a joint configuration

binary state of visible unit i

binary state of hidden unit j

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij}$$

Energy with configuration v on the visible units and h on the hidden units

weight between units i and j

$$-\frac{\partial E(v,h)}{\partial w_{ij}} = v_i h_j$$

Weights -> Energies -> Probabilities

- Each possible joint configuration of the visible and hidden units has an energy
  - The energy is determined by the weights and biases (as in a Hopfield net).
- The energy of a joint configuration of the visible and hidden units determines its probability:

$$p(v,h) \propto e^{-E(v,h)}$$

- The probability of a configuration over the visible units is found by summing the probabilities of all the joint configurations that contain it.

Using energies to define probabilities

- The probability of a joint configuration over both visible and hidden units depends on the energy of that joint configuration compared with the energy of all other joint configurations.

$$p(v,h) = \frac{e^{-E(v,h)}}{\underbrace{\sum_{u,g}}_{\text{partition function}} e^{-E(u,g)}}$$

- The probability of a configuration of the visible units is the sum of the probabilities of all the joint configurations that contain it.

$$p(v) = \frac{\sum_{h} e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}}$$

Energy based models with hidden units

To increase power of EBMs, add hidden variables.

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

By using the notation,

Free energy

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

We can rewrite p(x) in a form similar to the standard EBM,

$$P(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \text{ with } Z = \sum_x e^{-\mathcal{F}(x)}.$$

Conditional probabilities

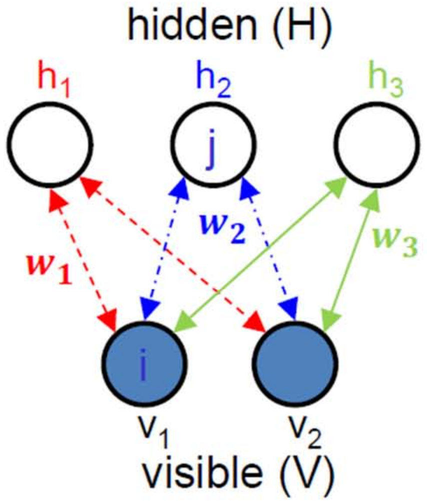
- Given $\mathbf{v}$, all the h_j are conditionally independent

$$P(h_j = 1|\mathbf{v}) = \frac{\exp(\sum_i W_{ij}v_j + b_j)}{\exp(\sum_i W_{ij}v_j + b_j) + 1}$$
$$=\text{sigmoid}(\textstyle\sum_i W_{ij}\, v_j + b_j)$$
$$=\text{sigmoid}(\mathbf{w}_j^T \mathbf{v} + b_j)$$

  – P(**h**|**v**) can be used as "features"

- Given $\mathbf{h}$, all the v_i are conditionally independent

$$P(v_i|\mathbf{h}) = \text{sigmoid}(\textstyle\sum_j W_{ij}\, h_j + c_i)$$



Tweakin' parameters

Now we need to adjust the model so it reflects our data, do ML

- Likelihood fn

$$\mathrm{L}(\theta) = \Pi_{i=1}^{n} p(x_i;\theta)$$

- Avg. Log-likelihood fn

$$\ell(\theta) = \frac{1}{n}\log(\Pi_i p(x_i;\theta)) = \frac{1}{n}\sum_i \log(p(x_i;\theta))$$

$$= \frac{1}{n}\sum_i log\frac{e^{-F(x_i)}}{Z} = \frac{1}{n}\sum_i(-F(x_i) - \log(Z))$$

Tweakin' parameters

- Take the derivative

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j} - \frac{\partial \log Z}{\partial \theta_j}\right) = \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j} + \frac{1}{Z}\frac{\partial Z}{\partial \theta_j}\right)$$

$$= \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j} + \frac{1}{Z}\sum_{\hat{x}} e^{-F(\hat{x})}\frac{\partial F(\hat{x})}{\partial \theta_j}\right)$$

$$= \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j}\right) + \sum_{\hat{x}} p(\hat{x})\frac{\partial F(\hat{x})}{\partial \theta_j}$$

$$= \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j}\right) + E_p\left[\frac{\partial F(x)}{\partial \theta_j}\right]$$

Tweakin' parameters

- Take the derivative

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j} - \frac{\partial \log Z}{\partial \theta_j}\right) = \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j} + \frac{1}{Z}\frac{\partial Z}{\partial \theta_j}\right)$$

$$= \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j} + \frac{1}{Z}\sum_{\hat{x}} e^{-F(\hat{x})}\frac{\partial F(\hat{x})}{\partial \theta_j}\right)$$

$$= \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j}\right) + \sum_{\hat{x}} p(\hat{x})\frac{\partial F(\hat{x})}{\partial \theta_j}$$

$$= \frac{1}{n}\sum_i \left(\frac{-\partial F(x_i)}{\partial \theta_j}\right) + E_p\left[\frac{\partial F(x)}{\partial \theta_j}\right]$$

Can think of as an expectation over dataset.

This is an expectation over all possible configurations of input x. Grows exponentially as function of the length of input.

Gradient revisited

$$
\begin{aligned}
\frac{\partial \log p(x)}{\partial \theta} &= \frac{\partial}{\partial \theta}\log\left(\frac{e^{-F(x)}}{Z}\right) = \frac{\partial - F(x)}{\partial \theta} - \frac{\partial log Z}{\partial \theta} \\
&= \frac{\partial log \sum_h e^{-E(x,h)}}{\partial \theta} - \frac{\partial \log \sum_{\hat{x}} \sum_h e^{-E(\hat{x},h)}}{\partial \theta} \\
&= -\frac{1}{\sum_h e^{-E(x,h)}} \sum_h e^{-E(x,h)} \frac{\partial E(x,h)}{\partial \theta} \\
&+ \frac{1}{\sum_{\hat{x},h} e^{-E(\hat{x},h)}} \sum_{\hat{x},h} e^{-E(\hat{x},h)} \frac{\partial E(\hat{x},h)}{\partial \theta} \\
&= -\sum_h p(h|x) \frac{\partial E(x,h)}{\partial \theta} + \sum_{\hat{x},h} p(\hat{x},h) \frac{\partial E(\hat{x},h)}{\partial \theta}
\end{aligned}
$$

The first term we can calculate directly from data and we sample from p(v,h) using Gibbs Sampling. [Remember that x represents the observable variables, *ie* v in RBM]

Learning rule

Recall energy function

$$E(v,h) = -\sum_i b_i v_i - \sum_j c_j h_j - \sum_{i,j} v_i h_j w_{i,j}$$

Calculating derivatives…

$$\frac{\partial E(v,h)}{\partial w_{i,j}} = v_i h_j \qquad \frac{\partial E(v,h)}{\partial b_i} = v_i$$

$$\frac{\partial E(v,h)}{\partial c_j} = h_j$$

So,

$$\Delta w_{i,j} \propto \epsilon(< v_i h_j >^0 - < v_i h_j >^\infty)$$

Inference

- Conditional Distribution: P(v|h) or P(h|v)
  - Easy to compute (see previous slides).
  - Due to conditional independence, we can sample all hidden units given all visible units in parallel (and vice versa)
- Joint Distribution: P(v,h)
  - Requires Gibbs Sampling (approximate; lots of iterations to converge).

```
Initialize with $\mathbf{v}^0$
Sample $\mathbf{h}^0$ from $P(\mathbf{h}|\mathbf{v}^0)$

Repeat until convergence (t=1,...) {
        Sample $\mathbf{v}^t$ from $P(\mathbf{v}^t|\mathbf{h}^{t-1})$
        Sample $\mathbf{h}^t$ from $P(\mathbf{h}|\mathbf{v}^t)$
}
```

Contrastive Divergence

- An approximation of the log-likelihood gradient for RBMs
  1. Replace the average over all possible inputs by samples

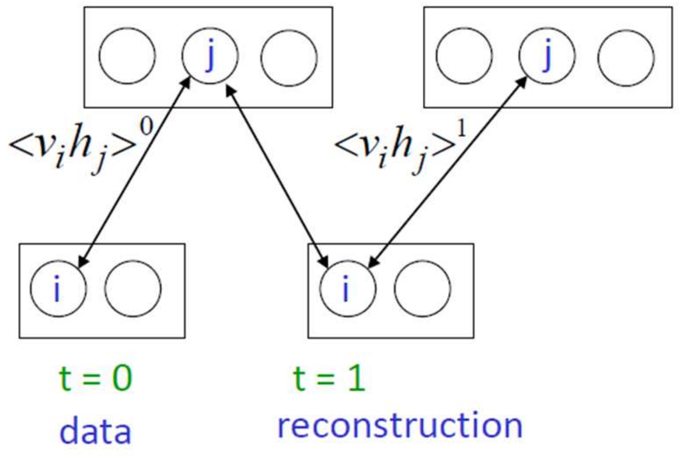
$$\frac{\partial}{\partial\theta}\log P(\mathbf{v}) = \mathbb{E}_{\mathbf{h}\sim P_\theta(\mathbf{h}|\mathbf{v})}\left[-\frac{\partial}{\partial\theta}E(\mathbf{h},\mathbf{v})\right] - \mathbb{E}_{\mathbf{v}',\mathbf{h}\sim P_\theta(\mathbf{v},\mathbf{h})}\left[-\frac{\partial}{\partial\theta}E(\mathbf{h}',\mathbf{v})\right]$$

  2. Run the MCMC chain (Gibbs sampling) for only k steps starting from the observed example

```
Initialize with $\mathbf{v}^0 = \mathbf{v}$
Sample $\mathbf{h}^0$ from $P(\mathbf{h}|\mathbf{v}^0)$

For t = 1,…,k {
        Sample $\mathbf{v}^t$ from $P(\mathbf{v}^t|\mathbf{h}^{t-1})$
        Sample $\mathbf{h}^t$ from $P(\mathbf{h}|\mathbf{v}^t)$
}
```

A quick way to learn an RBM



Start with a training vector on the visible units.

Update all the hidden units in parallel

Update the all the visible units in parallel to get a “reconstruction”.

Update the hidden units again.

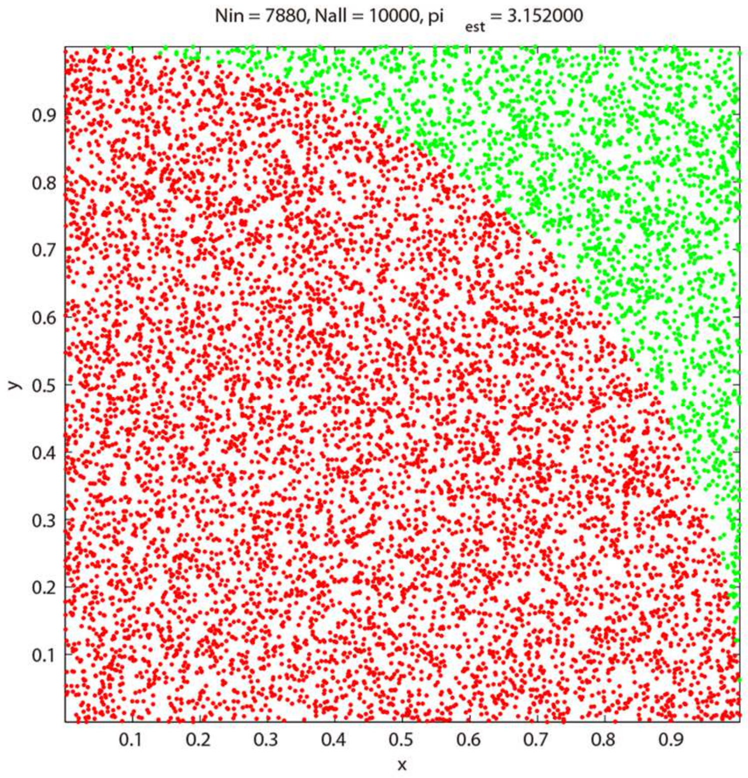
$$\Delta w_{ij} = \varepsilon \left(<v_i h_j>^0 - <v_i h_j>^1 \right)$$

This is not following the gradient of the log likelihood. But it works well. It is approximately following the gradient of another objective function (Carreira-Perpinan & Hinton, 2005).

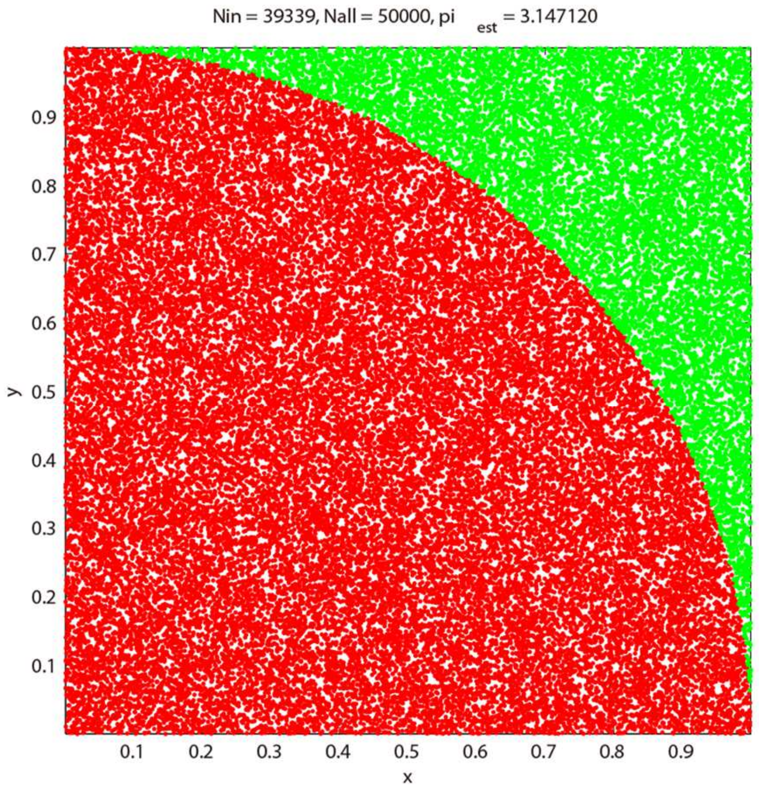
Markov Chain Monte Carlo

- Monte Carlo: sample from a distribution
  - to estimate the distribution
  - to compute max, mean
- Markov Chain Monte Carlo: sampling using "local" information
  - Generic "problem solving technique"
  - decision/optimization/value problems
  - generic, but not necessarily very efficient

Monte Carlo



Nin = 7880, Nall = 10000, pi est = 3.152000
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
y
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
x



Nin = 39339, Nall = 50000, pi est = 3.147120
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
y
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
x

Monte Carlo

Generate samples from distribution *p(x). But not always samples can be generated by formula.*

$$p(x) = \frac{\tilde{p}(x)}{\int \tilde{p}(x)dx}$$

P(x,y) is a 2-D distribution, which is difficult to represent, but p(x|y) and p(y|x) is easy to calculated.

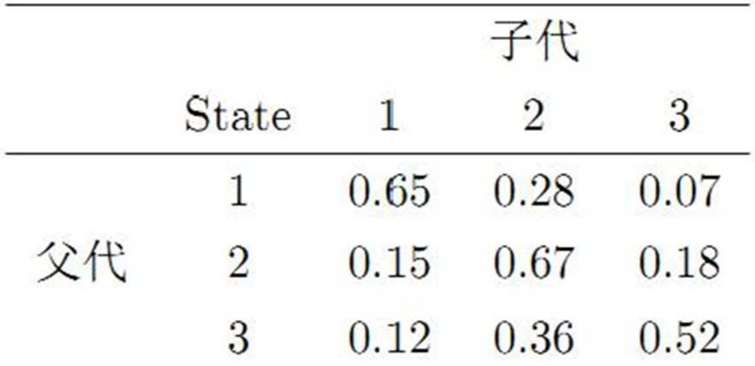
Markov chain and stationary distribution

$$P(X_{t+1}=x|X_t,X_{t-1},\cdots)=P(X_{t+1}=x|X_t)$$

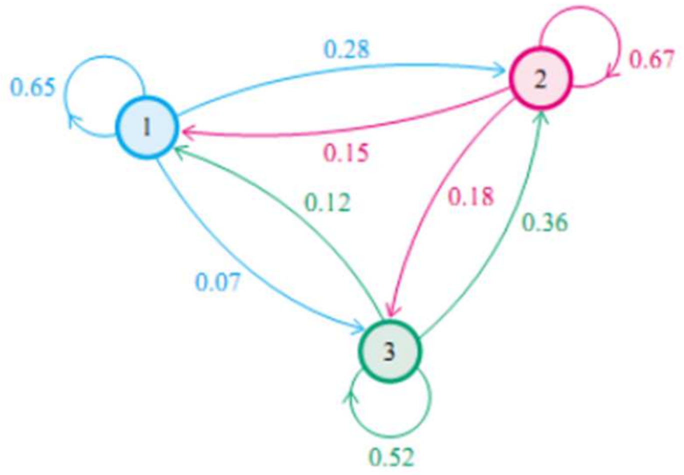
State transiting probability just depend on previous state.

A sample of Markov chain

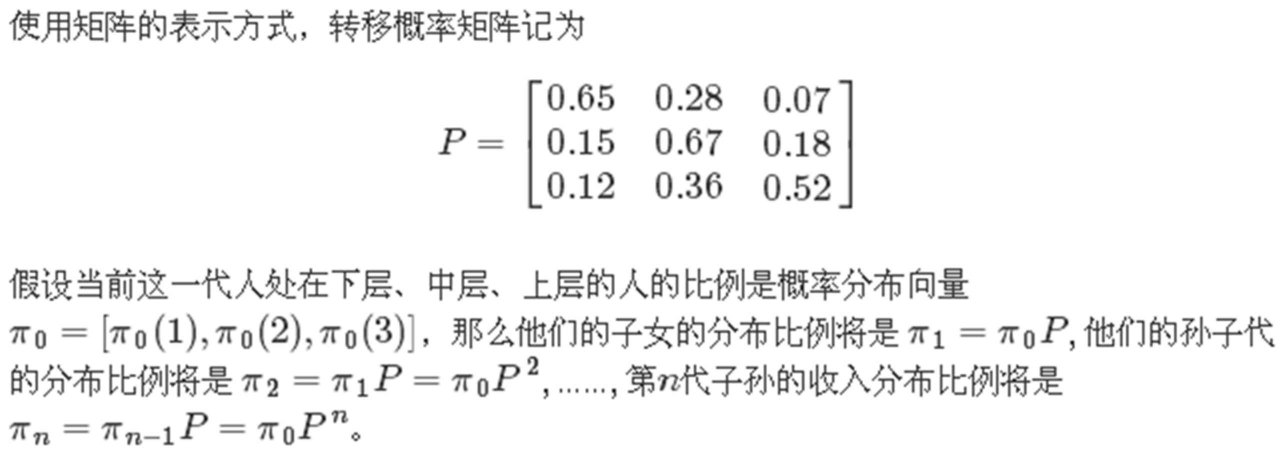
我们先来看马氏链的一个具体的例子。社会学家经常把人按其经济状况分成3类：下层(lower-class)、中层(middle-class)、上层(upper-class)，我们用1,2,3 分别代表这三个阶层。社会学家们发现决定一个人的收入阶层的最重要的因素就是其父母的收入阶层。如果一个人的收入属于下层类别，那么他的孩子属于下层收入的概率是 0.65，属于中层收入的概率是 0.28，属于上层收入的概率是 0.07。事实上，从父代到子代，收入阶层的变化的转移概率如下



		子代		
	State	1	2	3
	1	0.65	0.28	0.07
父代	2	0.15	0.67	0.18
	3	0.12	0.36	0.52



A sample of Markov chain

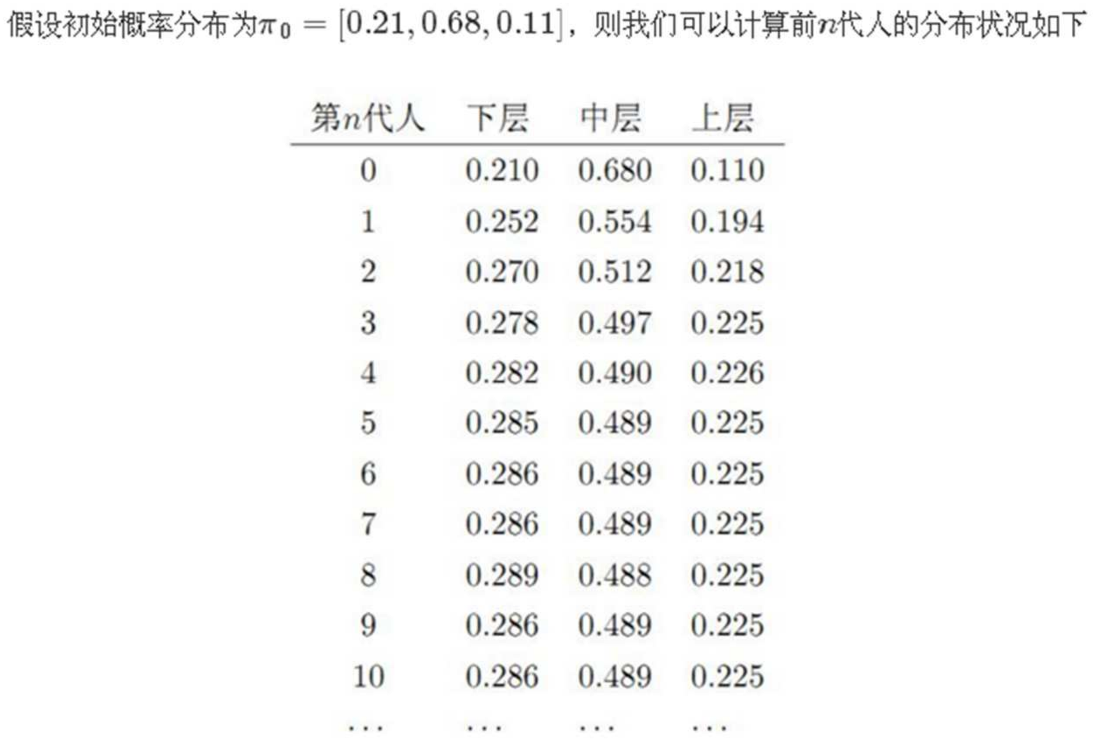


使用矩阵的表示方式，转移概率矩阵记为

$$P = \begin{bmatrix} 0.65 & 0.28 & 0.07 \\ 0.15 & 0.67 & 0.18 \\ 0.12 & 0.36 & 0.52 \end{bmatrix}$$

假设当前这一代人处在下层、中层、上层的人的比例是概率分布向量 $\pi_0 = [\pi_0(1), \pi_0(2), \pi_0(3)]$，那么他们的子女的分布比例将是 $\pi_1 = \pi_0 P$, 他们的孙子代的分布比例将是 $\pi_2 = \pi_1 P = \pi_0 P^2$, ……, 第$n$代子孙的收入分布比例将是 $\pi_n = \pi_{n-1} P = \pi_0 P^n$。

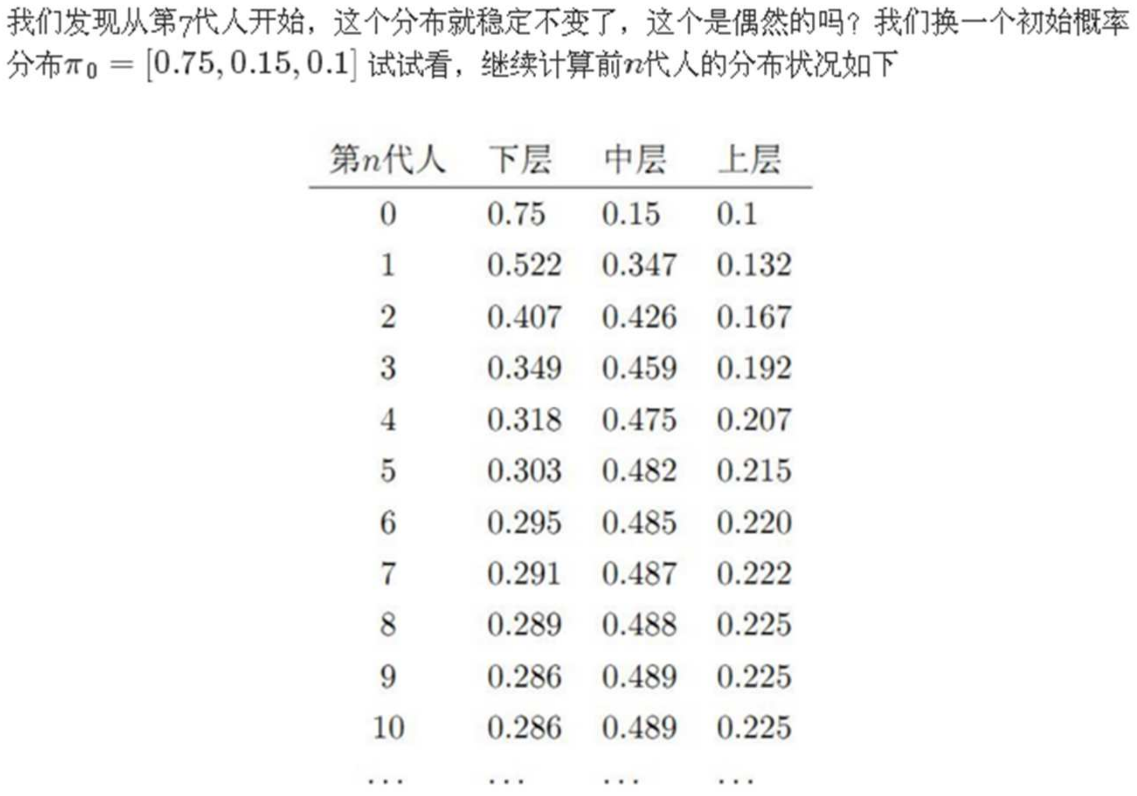
A sample of Markov chain



假设初始概率分布为$\pi_0 = [0.21, 0.68, 0.11]$，则我们可以计算前$n$代人的分布状况如下

第n代人	下层	中层	上层
0	0.210	0.680	0.110
1	0.252	0.554	0.194
2	0.270	0.512	0.218
3	0.278	0.497	0.225
4	0.282	0.490	0.226
5	0.285	0.489	0.225
6	0.286	0.489	0.225
7	0.286	0.489	0.225
8	0.289	0.488	0.225
9	0.286	0.489	0.225
10	0.286	0.489	0.225
⋯	⋯	⋯	⋯

A sample of Markov chain



我们发现从第7代人开始，这个分布就稳定不变了，这个是偶然的吗？我们换一个初始概率分布$\pi_0 = [0.75, 0.15, 0.1]$ 试试看，继续计算前n代人的分布状况如下

第n代人	下层	中层	上层
0	0.75	0.15	0.1
1	0.522	0.347	0.132
2	0.407	0.426	0.167
3	0.349	0.459	0.192
4	0.318	0.475	0.207
5	0.303	0.482	0.215
6	0.295	0.485	0.220
7	0.291	0.487	0.222
8	0.289	0.488	0.225
9	0.286	0.489	0.225
10	0.286	0.489	0.225
⋯	⋯	⋯	⋯

A sample of Markov chain

我们发现，到第9代人的时候, 分布又收敛了。最为奇特的是，两次给定不同的初始概率分布，最终都收敛到概率分布 $\pi = [0.286, 0.489, 0.225]$，也就是说收敛的行为和初始概率分布 π_0 无关。这说明这个收敛行为主要是由概率转移矩阵P决定的。我们计算一下 P^n

$$P^{20} = P^{21} = \cdots = P^{100} = \cdots = \begin{bmatrix} 0.286 & 0.489 & 0.225 \\ 0.286 & 0.489 & 0.225 \\ 0.286 & 0.489 & 0.225 \end{bmatrix}$$

我们发现，当 n 足够大的时候，这个P^n矩阵的每一行都是稳定地收敛到 $\pi = [0.286, 0.489, 0.225]$ 这个概率分布。自然的，这个收敛现象并非是我们这个马氏链独有的，而是绝大多数马氏链的共同行为，关于马氏链的收敛我们有如下漂亮的定理：

A sample of Markov chain

马氏链定理： 如果一个非周期马氏链具有转移概率矩阵P,且它的任何两个状态是连通的，那么 $\lim\limits_{n\to\infty} P_{ij}^n$ 存在且与i无关，记 $\lim\limits_{n\to\infty} P_{ij}^n = \pi(j)$, 我们有

1. $$\lim_{n\to\infty} P^n = \begin{bmatrix} \pi(1) & \pi(2) & \cdots & \pi(j) & \cdots \\ \pi(1) & \pi(2) & \cdots & \pi(j) & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \pi(1) & \pi(2) & \cdots & \pi(j) & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

1. $$\pi(j) = \sum_{i=0}^{\infty} \pi(i) P_{ij}$$
2. π 是方程 $\pi P = \pi$ 的唯一非负解

其中,

$$\pi = [\pi(1), \pi(2), \cdots, \pi(j), \cdots], \quad \sum_{i=0}^{\infty} \pi_i = 1$$

π称为马氏链的平稳分布。

A sample of Markov chain

从初始概率分布 π_0 出发，我们在马氏链上做状态转移，记X_i的概率分布为π_i, 则有

$$\begin{aligned} X_0 &\sim \pi_0(x) \\ X_i &\sim \pi_i(x), \qquad \pi_i(x) = \pi_{i-1}(x)P = \pi_0(x)P^n \end{aligned}$$

由马氏链收敛的定理, 概率分布$\pi_i(x)$将收敛到平稳分布 $\pi(x)$。假设到第n步的时候马氏链收敛，则有

$$\begin{aligned} X_0 &\sim \pi_0(x) \\ X_1 &\sim \pi_1(x) \\ &\cdots \\ X_n &\sim \pi_n(x) = \pi(x) \\ X_{n+1} &\sim \pi(x) \\ X_{n+2} &\sim \pi(x) \\ &\cdots \end{aligned}$$

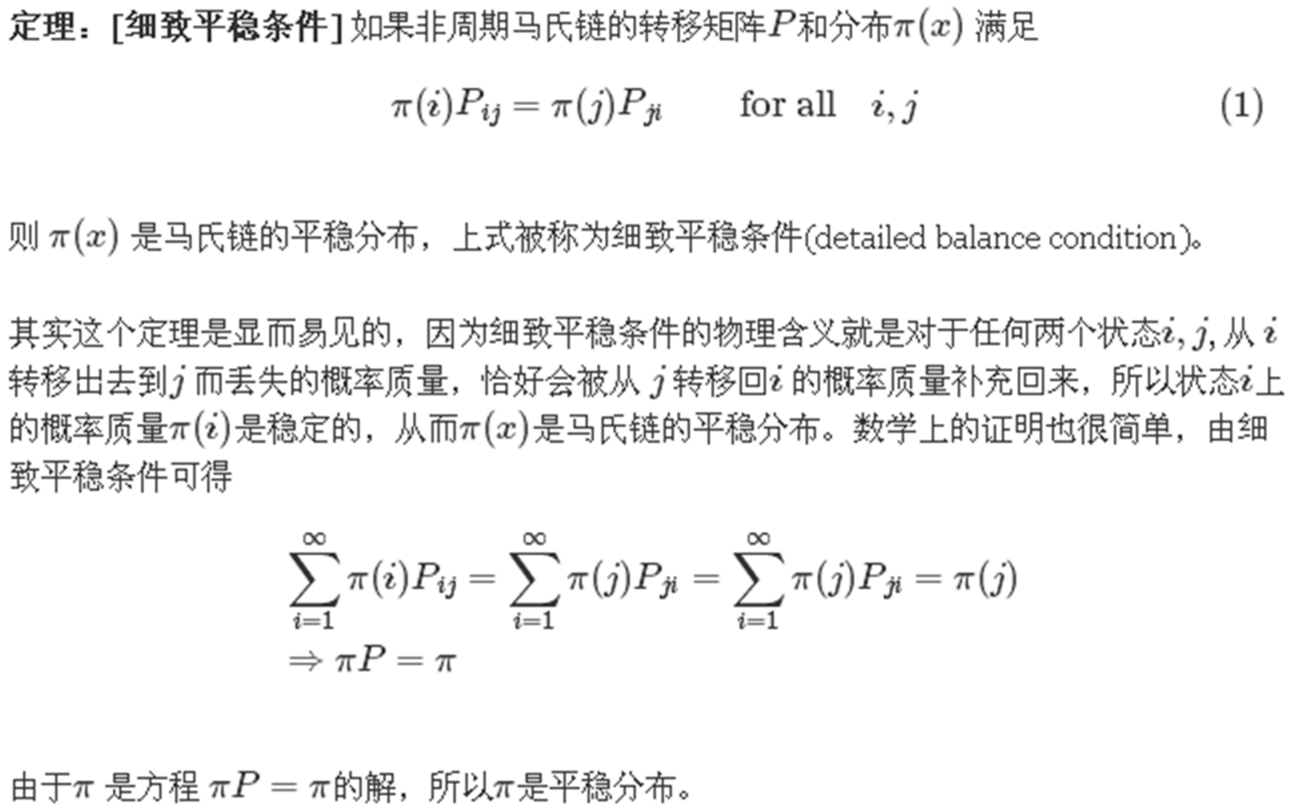
所以 $X_n, X_{n+1}, X_{n+2}, \cdots \sim \pi(x)$ 都是同分布的随机变量，当然他们并不独立。如果我们从一个具体的初始状态 x_0 开始,沿着马氏链按照概率转移矩阵做跳转，那么我们得到一个转移序列 $x_0, x_1, x_2, \cdots x_n, x_{n+1} \cdots$, 由于马氏链的收敛行为， $x_n, x_{n+1}, \cdots$ 都将是平稳分布 $\pi(x)$ 的样本。

Markov chain Monte Carlo

对于给定的概率分布$p(x)$,我们希望能有便捷的方式生成它对应的样本。由于马氏链能收敛到平稳分布， 于是一个很的漂亮想法是：如果我们能构造一个转移矩阵为P的马氏链，使得该马氏链的平稳分布恰好是$p(x)$, 那么我们从任何一个初始状态x_0出发沿着马氏链转移, 得到一个转移序列 $x_0, x_1, x_2, \cdots x_n, x_{n+1} \cdots,$， 如果马氏链在第$n$步已经收敛了，于是我们就得到了 $\pi(x)$ 的样本$x_n, x_{n+1} \cdots$。

这个绝妙的想法在1953年被 Metropolis想到了，为了研究粒子系统的平稳性质，Metropolis 考虑了物理学中常见的波尔兹曼分布的采样问题，首次提出了基于马氏链的蒙特卡罗方法，即Metropolis算法，并在最早的计算机上编程实现。Metropolis 算法是首个普适的采样方法，并启发了一系列 MCMC方法，所以人们把它视为随机模拟技术腾飞的起点。Metropolis的这篇论文被收录在《统计学中的重大突破》中， Metropolis算法也被遴选为二十世纪的十个最重要的算法之一。

Markov chain Monte Carlo



定理：[细致平稳条件] 如果非周期马氏链的转移矩阵P和分布$\pi(x)$ 满足

$$\pi(i)P_{ij} = \pi(j)P_{ji} \qquad \text{for all} \quad i,j \tag{1}$$

则 $\pi(x)$ 是马氏链的平稳分布，上式被称为细致平稳条件(detailed balance condition)。

其实这个定理是显而易见的，因为细致平稳条件的物理含义就是对于任何两个状态i,j, 从 i 转移出去到j 而丢失的概率质量，恰好会被从 j 转移回i 的概率质量补充回来，所以状态i上的概率质量$\pi(i)$是稳定的，从而$\pi(x)$是马氏链的平稳分布。数学上的证明也很简单，由细致平稳条件可得

$$\sum_{i=1}^{\infty}\pi(i)P_{ij} = \sum_{i=1}^{\infty}\pi(j)P_{ji} = \sum_{i=1}^{\infty}\pi(j)P_{ji} = \pi(j)$$
$$\Rightarrow \pi P = \pi$$

由于π 是方程 $\pi P = \pi$的解，所以π是平稳分布。

Markov chain Monte Carlo

假设我们已经有一个转移矩阵为Q马氏链($q(i,j)$表示从状态 i转移到状态j的概率，也可以写为 $q(j|i)$或者$q(i \rightarrow j)$), 显然，通常情况下

$$p(i)q(i,j) \neq p(j)q(j,i)$$

也就是细致平稳条件不成立，所以 $p(x)$ 不太可能是这个马氏链的平稳分布。我们可否对马氏链做一个改造，使得细致平稳条件成立呢？譬如，我们引入一个 $\alpha(i,j)$, 我们希望

$$p(i)q(i,j)\alpha(i,j) = p(j)q(j,i)\alpha(j,i) \quad (*) \tag{2}$$

取什么样的 $\alpha(i,j)$ 以上等式能成立呢？最简单的，按照对称性，我们可以取

$$\alpha(i,j) = p(j)q(j,i)\,, \quad \alpha(j,i) = p(i)q(i,j)$$

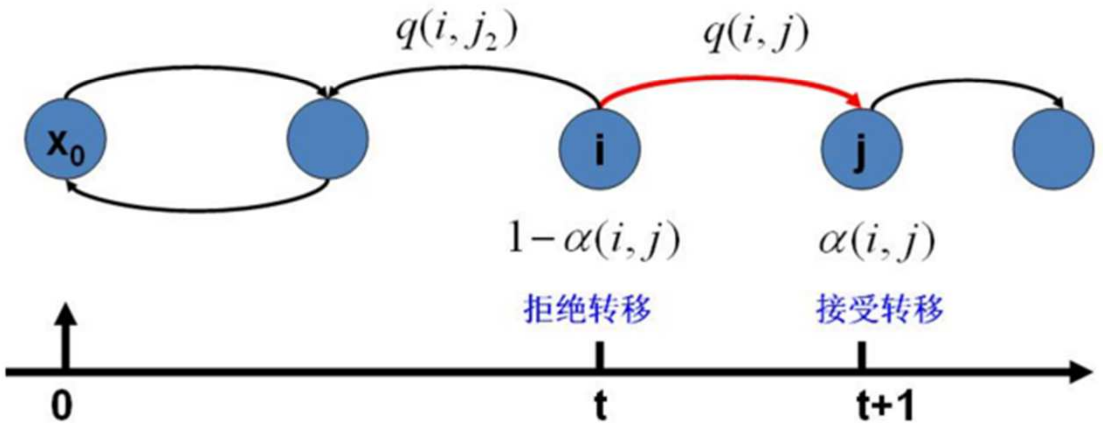
于是(*)式就成立了。所以有

$$p(i)\underbrace{q(i,j)\alpha(i,j)}_{Q'(i,j)} = p(j)\underbrace{q(j,i)\alpha(j,i)}_{Q'(j,i)} \quad (**) \tag{3}$$

Markov chain Monte Carlo

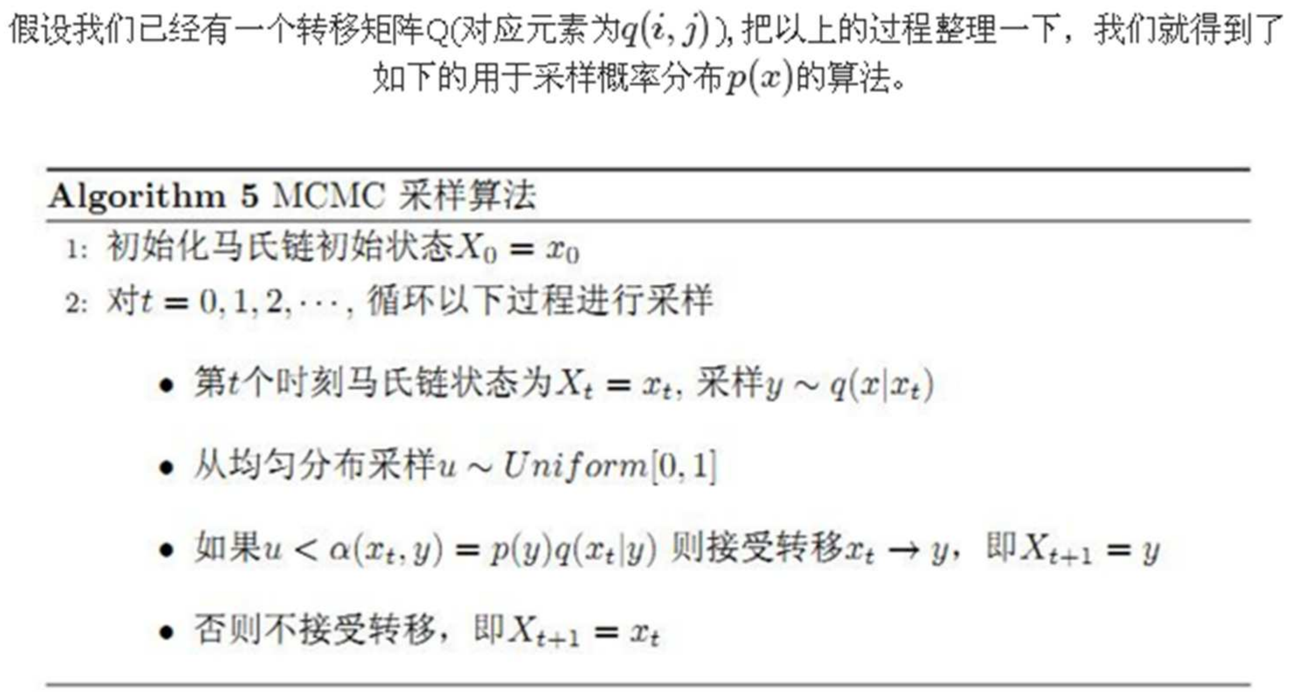
于是我们把原来具有转移矩阵Q的一个很普通的马氏链，改造为了具有转移矩阵Q'的马氏链，而 Q'恰好满足细致平稳条件，由此马氏链Q'的平稳分布就是$p(x)$！

在改造 Q 的过程中引入的 $\alpha(i,j)$称为接受率，物理意义可以理解为在原来的马氏链上，从状态 i 以$q(i,j)$ 的概率转跳转到状态j 的时候，我们以$\alpha(i,j)$的概率接受这个转移，于是得到新的马氏链Q'的转移概率为$q(i,j)\alpha(i,j)$。



马氏链转移和接受概率

Markov chain Monte Carlo



假设我们已经有一个转移矩阵Q(对应元素为$q(i,j)$), 把以上的过程整理一下，我们就得到了如下的用于采样概率分布$p(x)$的算法。

Algorithm 5 MCMC 采样算法

```
1: 初始化马氏链初始状态$X_0 = x_0$
2: 对$t = 0, 1, 2, \cdots$, 循环以下过程进行采样
      • 第$t$个时刻马氏链状态为$X_t = x_t$, 采样$y \sim q(x|x_t)$
      • 从均匀分布采样$u \sim Uniform[0, 1]$
      • 如果$u < \alpha(x_t, y) = p(y)q(x_t|y)$ 则接受转移$x_t \to y$，即$X_{t+1} = y$
      • 否则不接受转移，即$X_{t+1} = x_t$
```

Markov chain Monte Carlo

以上的 MCMC 采样算法已经能很漂亮的工作了，不过它有一个小的问题：马氏链Q在转移的过程中的接受率 $\alpha(i,j)$ 可能偏小，这样采样过程中马氏链容易原地踏步，拒绝大量的跳转，这使得马氏链遍历所有的状态空间要花费太长的时间，收敛到平稳分布$p(x)$的速度太慢。有没有办法提升一些接受率呢?

假设 $\alpha(i,j)=0.1, \alpha(j,i)=0.2,$ 此时满足细致平稳条件，于是

$$p(i)q(i,j)\times 0.1 = p(j)q(j,i)\times 0.2$$

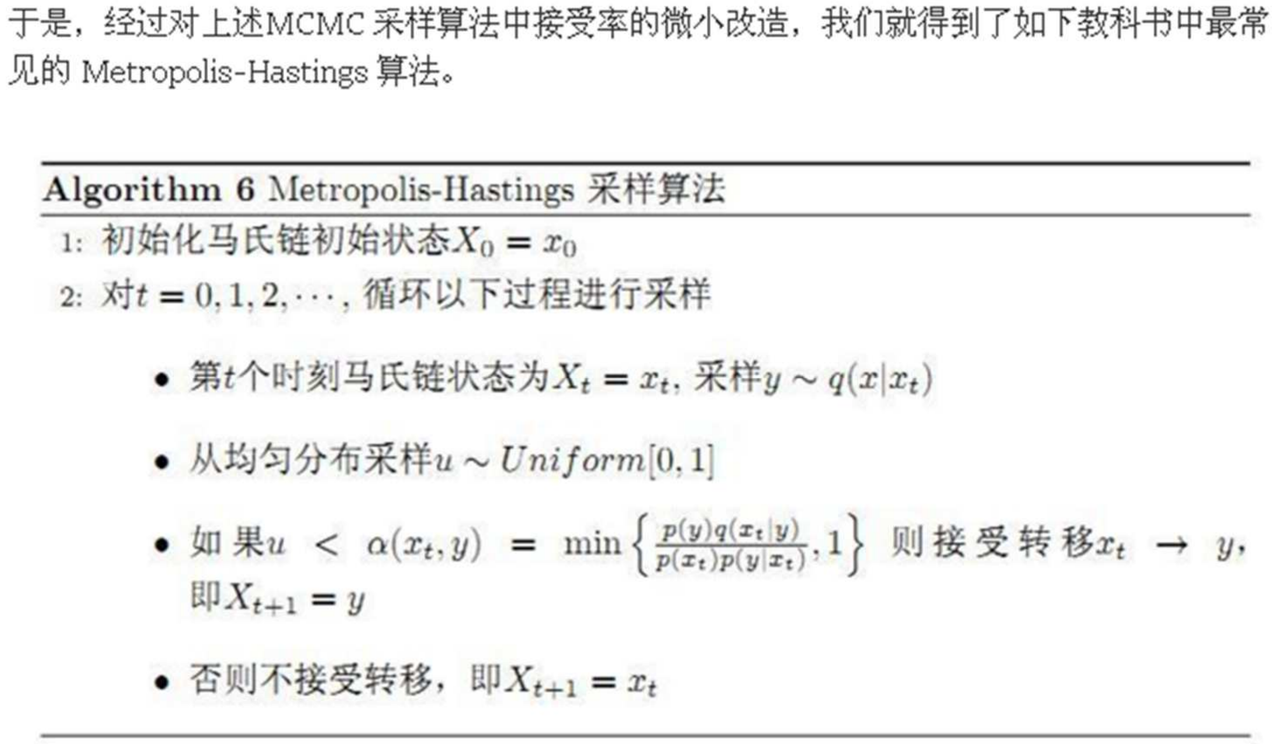
上式两边扩大5倍，我们改写为

$$p(i)q(i,j)\times 0.5 = p(j)q(j,i)\times 1$$

看，我们提高了接受率，而细致平稳条件并没有打破！这启发我们可以把细致平稳条件(**)式中的$\alpha(i,j), \alpha(j,i)$ 同比例放大，使得两数中最大的一个放大到1，这样我们就提高了采样中的跳转接受率。所以我们可以取

$$\alpha(i,j)=\min\left\{\frac{p(j)q(j,i)}{p(i)q(i,j)},1\right\}$$

Markov chain Monte Carlo


于是，经过对上述MCMC 采样算法中接受率的微小改造，我们就得到了如下教科书中最常见的 Metropolis-Hastings 算法。

Algorithm 6 Metropolis-Hastings 采样算法

```
1: 初始化马氏链初始状态$X_0 = x_0$
2: 对$t = 0, 1, 2, \cdots$, 循环以下过程进行采样
    • 第$t$个时刻马氏链状态为$X_t = x_t$, 采样$y \sim q(x|x_t)$
    • 从均匀分布采样$u \sim Uniform[0, 1]$
    • 如果$u < \alpha(x_t, y) = \min\left\{\frac{p(y)q(x_t|y)}{p(x_t)p(y|x_t)}, 1\right\}$ 则接受转移$x_t \to y$, 即$X_{t+1} = y$
    • 否则不接受转移，即$X_{t+1} = x_t$
```

Gibbs Sampling

对于高维的情形，由于接受率 α的存在(通常 $\alpha < 1$), 以上 Metropolis-Hastings 算法的效率不够高。能否找到一个转移矩阵Q使得接受率 $\alpha = 1$ 呢？我们先看看二维的情形，假设有一个概率分布 $p(x, y)$, 考察x坐标相同的两个点$A(x_1, y_1), B(x_1, y_2)$，我们发现

$$p(x_1, y_1)p(y_2|x_1) = p(x_1)p(y_1|x_1)p(y_2|x_1)$$
$$p(x_1, y_2)p(y_1|x_1) = p(x_1)p(y_2|x_1)p(y_1|x_1)$$

所以得到

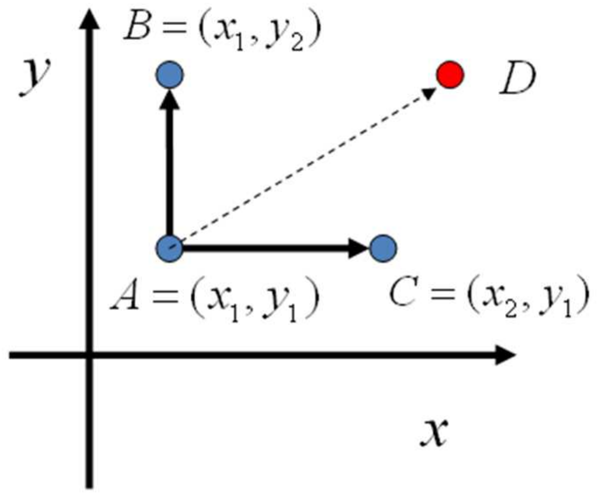
$$p(x_1, y_1)p(y_2|x_1) = p(x_1, y_2)p(y_1|x_1) \quad (***) \tag{4}$$

即

$$p(A)p(y_2|x_1) = p(B)p(y_1|x_1)$$

基于以上等式，我们发现，在 $x = x_1$ 这条平行于 y轴的直线上，如果使用条件分布$p(y|x_1)$做为任何两个点之间的转移概率，那么任何两个点之间的转移满足细致平稳条件。同样的，如果我们在 $y = y_1$ 这条直线上任意取两个点 $A(x_1, y_1), C(x_2, y_1)$,也有如下等式

$$p(A)p(x_2|y_1) = p(C)p(x_1|y_1).$$



平面上马氏链转移矩阵的构造

Gibbs Sampling

于是我们可以如下构造平面上任意两点之间的转移概率矩阵Q

$$\begin{aligned} Q(A\rightarrow B) &= p(y_B|x_1) & \text{如果} \quad x_A = x_B = x_1 \\ Q(A\rightarrow C) &= p(x_C|y_1) & \text{如果} \quad y_A = y_C = y_1 \\ Q(A\rightarrow D) &= 0 & \text{其它} \end{aligned}$$

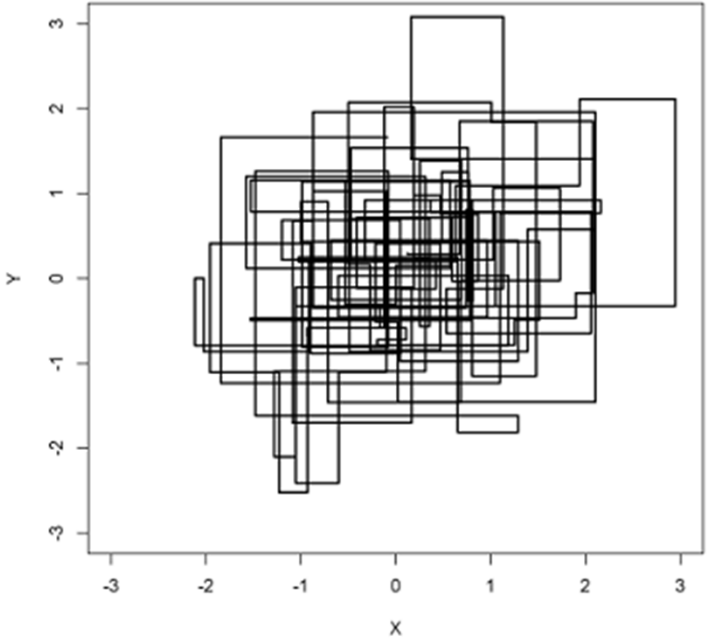
有了如上的转移矩阵 Q, 我们很容易验证对平面上任意两点 X, Y, 满足细致平稳条件

$$p(X)Q(X\rightarrow Y) = p(Y)Q(Y\rightarrow X)$$

于是这个二维空间上的马氏链将收敛到平稳分布 $p(x, y)$。而这个算法就称为 Gibbs Sampling 算法,是 Stuart Geman 和Donald Geman 这两兄弟于1984年提出来的，之所以叫做Gibbs Sampling 是因为他们研究了Gibbs random field, 这个算法在现代贝叶斯分析中占据重要位置。

Algorithm 7 二维Gibbs Sampling 算法

```
1: 随机初始化$X_0 = x_0 Y_0 = y_0$
2: 对$t = 0, 1, 2, \cdots$ 循环采样
      1. $y_{t+1} \sim p(y|x_t)$
      2. $x_{t+1} \sim p(x|y_{t+1})$
```



二维Gibbs Sampling算法中的马氏链转移

Gibbs Sampling

以上的过程我们很容易推广到高维的情形，对于(***)式，如果x_1变为多维情形$\mathbf{x_1}$，可以看出推导过程不变，所以细致平稳条件同样是成立的

$$p(\mathbf{x_1}, y_1)p(y_2|\mathbf{x_1}) = p(\mathbf{x_1}, y_2)p(y_1|\mathbf{x_1}) \tag{5}$$

此时转移矩阵 Q 由条件分布 $p(y|\mathbf{x_1})$ 定义。上式只是说明了一根坐标轴的情形，和二维情形类似，很容易验证对所有坐标轴都有类似的结论。所以n维空间中对于概率分布 $p(x_1, x_2, \cdots, x_n)$ 可以如下定义转移矩阵

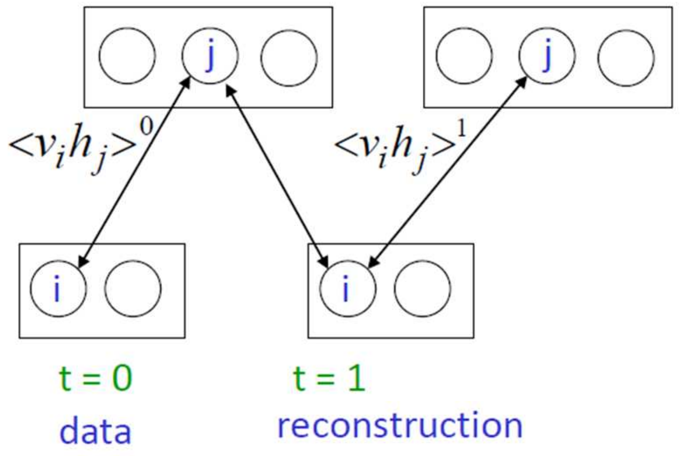
1. 如果当前状态为$(x_1, x_2, \cdots, x_n)$，马氏链转移的过程中，只能沿着坐标轴做转移。沿着 x_i 这根坐标轴做转移的时候，转移概率由条件概率 $p(x_i|x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$ 定义；
2. 其它无法沿着单根坐标轴进行的跳转，转移概率都设置为 0。

于是我们可以把Gibbs Smapling 算法从采样二维的 $p(x, y)$ 推广到采样n 维的 $p(x_1, x_2, \cdots, x_n)$

Algorithm 8 n维Gibbs Sampling 算法

```
1: 随机初始化$\{x_i : i = 1, \cdots, n\}$
2: 对$t = 0, 1, 2, \cdots$ 循环采样
        1. $x_1^{(t+1)} \sim p(x_1|x_2^{(t)}, x_3^{(t)}, \cdots, x_n^{(t)})$
        2. $x_2^{(t+1)} \sim p(x_2|x_1^{(t+1)}, x_3^{(t)}, \cdots, x_n^{(t)})$
        3. $\cdots$
        4. $x_j^{(t+1)} \sim p(x_j|x_1^{(t+1)}, \cdots, x_{j-1}^{(t+1)}, x_{j+1}^{(t)}, \cdots, x_n^{(t)})$
        5. $\cdots$
        6. $x_n^{(t+1)} \sim p(x_n|x_1^{(t+1)}, x_2^{t}, \cdots, x_{n-1}^{(t+1)})$
```

A quick way to learn an RBM



Start with a training vector on the visible units.

Update all the hidden units in parallel

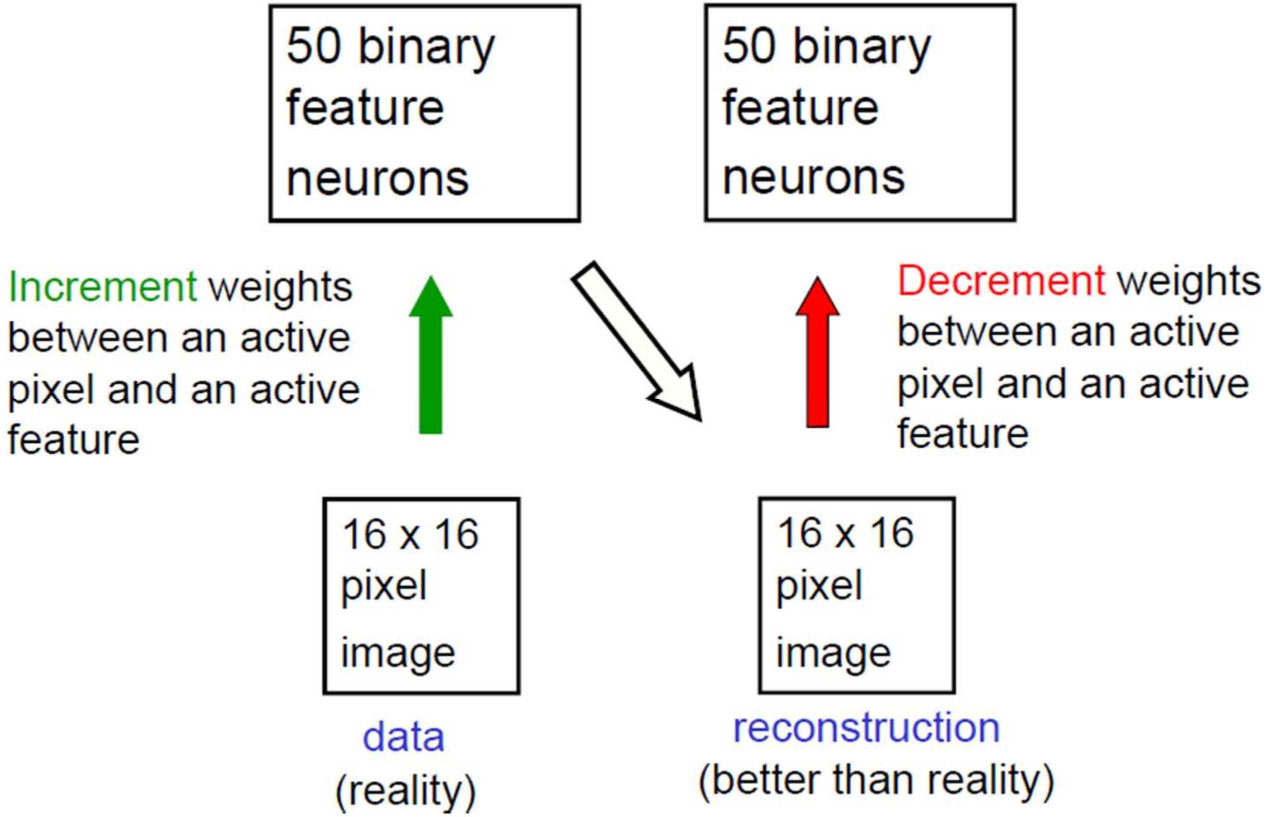
Update the all the visible units in parallel to get a "reconstruction".

Update the hidden units again.

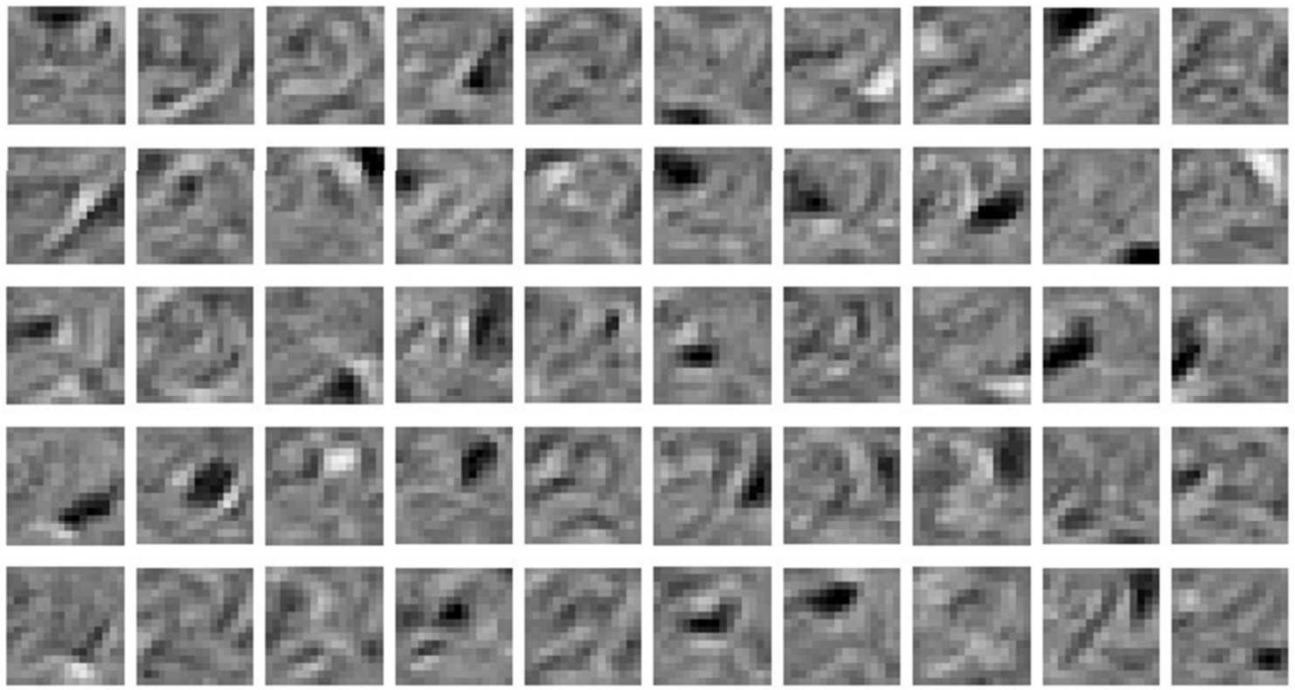
$$\Delta w_{ij} = \varepsilon \left(<v_i h_j>^0 - <v_i h_j>^1 \right)$$

This is not following the gradient of the log likelihood. But it works well. It is approximately following the gradient of another objective function (Carreira-Perpinan & Hinton, 2005).

How to learn a set of features that are good for reconstructing images of the digit 2


50 binary feature neurons
50 binary feature neurons
Increment weights between an active pixel and an active feature
Decrement weights between an active pixel and an active feature
16 x 16 pixel image
16 x 16 pixel image
data
(reality)
reconstruction
(better than reality)

The final 50 x 256 weights

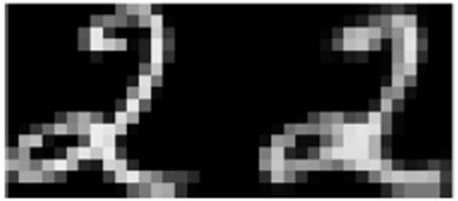


Each neuron grabs a different feature.

How well can we reconstruct the digit images from the binary feature activations?

Data

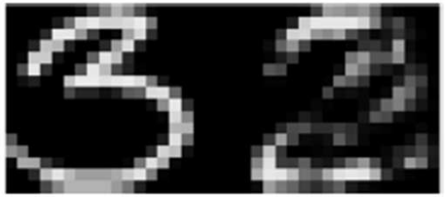
Reconstruction from activated binary features



New test images from the digit class that the model was trained on

Data

Reconstruction from activated binary features



Images from an unfamiliar digit class (the network tries to see every image as a 2)

Three ways to combine probability density models

- **Mixture:** Take a weighted average of the distributions.
  - It can never be sharper than the individual distributions. It's a very weak way to combine models.
- **Product:** Multiply the distributions at each point and then renormalize.
  - Exponentially more powerful than a mixture. The normalization makes maximum likelihood learning difficult, but approximations allow us to learn anyway.
- **Composition:** Use the values of the latent variables of one model as the data for the next model.
  - Works well for learning multiple layers of representation, but only if the individual models are undirected.

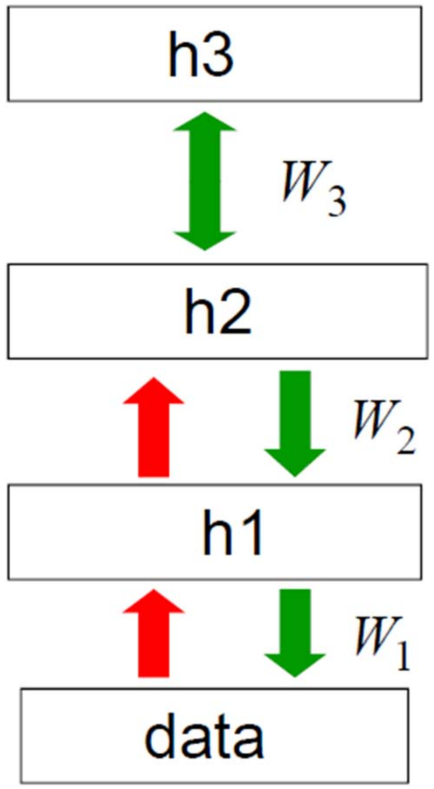
Training a deep network

- First train a layer of features that receive input directly from the pixels.
- Then treat the activations of the trained features as if they were pixels and learn features of features in a second hidden layer.
- It can be proved that each time we add another layer of features we improve a variational lower bound on the log probability of the training data.
  - The proof is slightly complicated.
  - But it is based on a neat equivalence between an RBM and a deep directed model (described later)

The generative model after learning 3 layers

- To generate data:
  1. Get an equilibrium sample from the top-level RBM by performing alternating Gibbs sampling for a long time.
  2. Perform a top-down pass to get states for all the other layers.

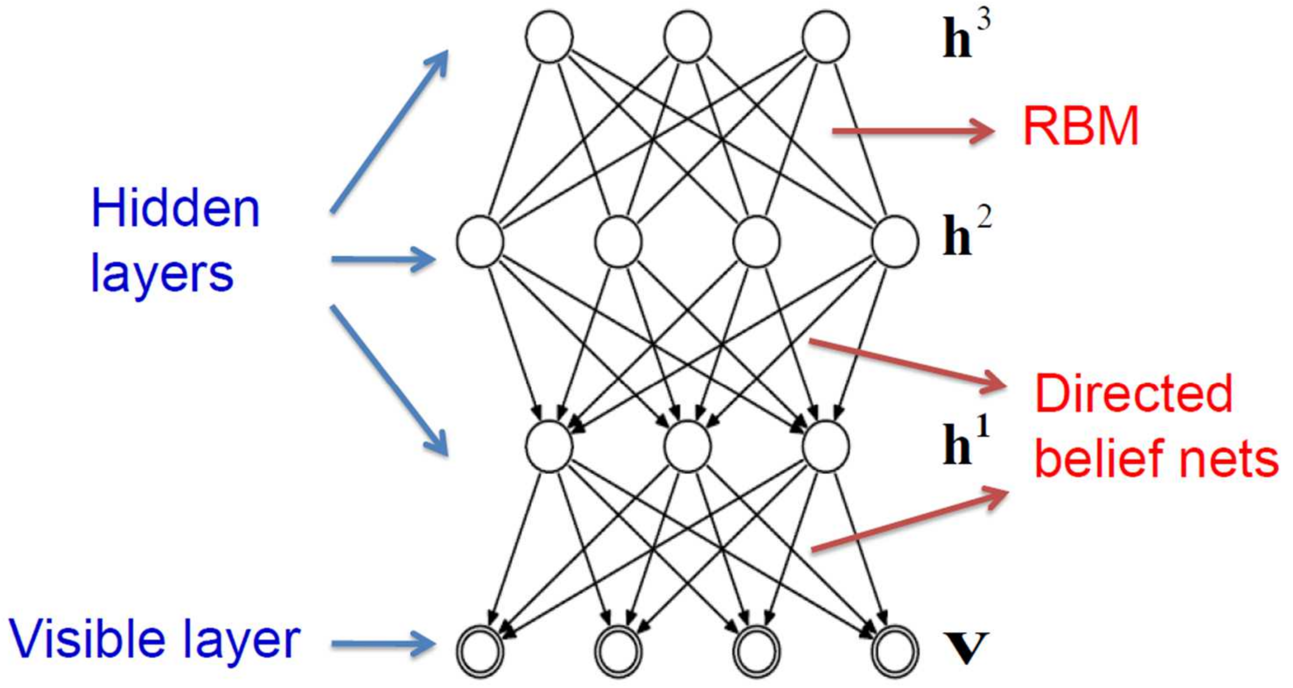
  So the lower level bottom-up connections are not part of the generative model. They are just used for inference.



Deep Belief Networks (DBNs)

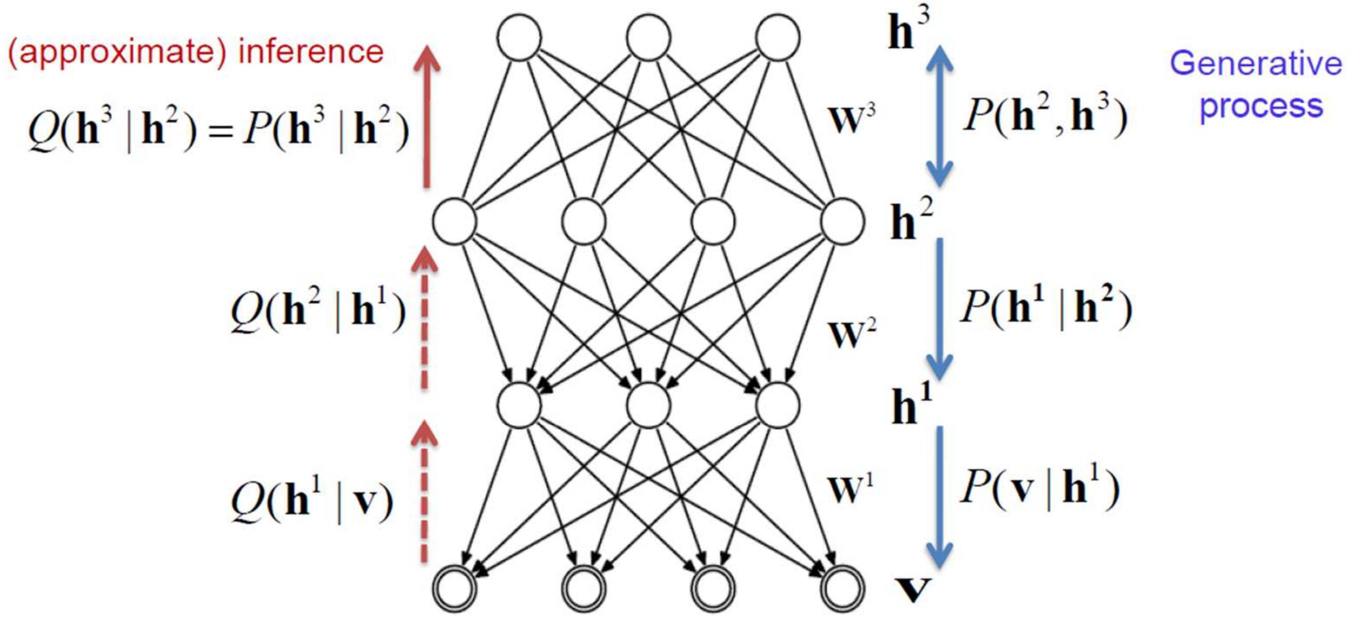
- Probabilistic generative model
- Deep architecture – multiple layers
- Unsupervised pre-learning provides a good initialization of the network
  - maximizing the lower-bound of the log-likelihood of the data
- Supervised fine-tuning
  - Generative: Up-down algorithm
  - Discriminative: backpropagation

DBN structure



$$P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \ldots, \mathbf{h}^l) = P(\mathbf{v} \mid \mathbf{h}^1) P(\mathbf{h}^1 \mid \mathbf{h}^2) \ldots P(\mathbf{h}^{l-2} \mid \mathbf{h}^{l-1}) P(\mathbf{h}^{l-1}, \mathbf{h}^l)$$

DBN structure



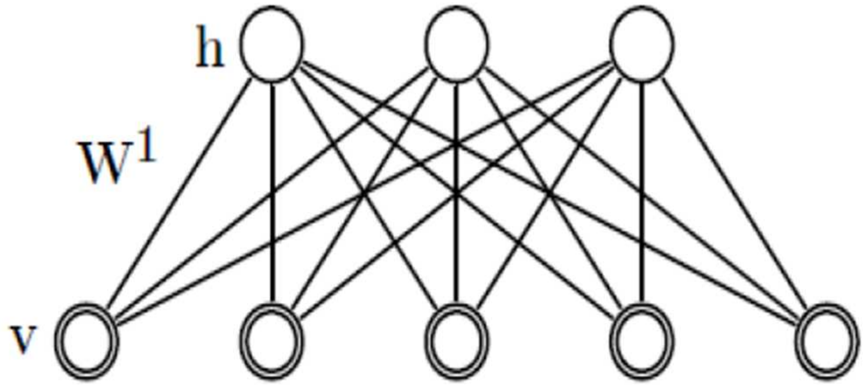
(approximate) inference
$Q(\mathbf{h}^3 \mid \mathbf{h}^2) = P(\mathbf{h}^3 \mid \mathbf{h}^2)$
$Q(\mathbf{h}^2 \mid \mathbf{h}^1)$
$Q(\mathbf{h}^1 \mid \mathbf{v})$
$\mathbf{h}^3$
$\mathbf{W}^3$
$\mathbf{h}^2$
$\mathbf{W}^2$
$\mathbf{h}^1$
$\mathbf{W}^1$
$\mathbf{v}$
$P(\mathbf{h}^2, \mathbf{h}^3)$
$P(\mathbf{h}^1 \mid \mathbf{h}^2)$
$P(\mathbf{v} \mid \mathbf{h}^1)$
Generative process

$$P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \ldots, \mathbf{h}^l) = P(\mathbf{v} \mid \mathbf{h}^1) P(\mathbf{h}^1 \mid \mathbf{h}^2) \ldots P(\mathbf{h}^{l-2} \mid \mathbf{h}^{l-1}) P(\mathbf{h}^{l-1}, \mathbf{h}^l)$$

$$Q(\mathbf{h}^i \mid \mathbf{h}^{i-1}) = \prod_j sigm(\mathbf{b}_j^{i-1} + \mathrm{W}_j^i \mathbf{h}^{i-1}) \quad P(\mathbf{h}^{i-1} \mid \mathbf{h}^i) = \prod_j sigm(\mathbf{b}_j^i + \mathrm{W}_{.j}^i{}' \mathbf{h}^i)$$

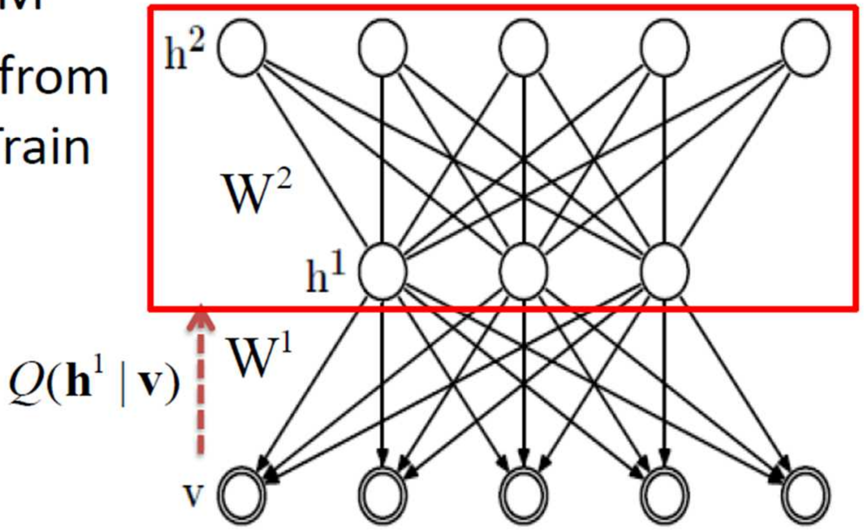
DBN greedy training

- First step:
  - Construct an RBM with an input layer **v** and a hidden layer **h**
  - Train the RBM



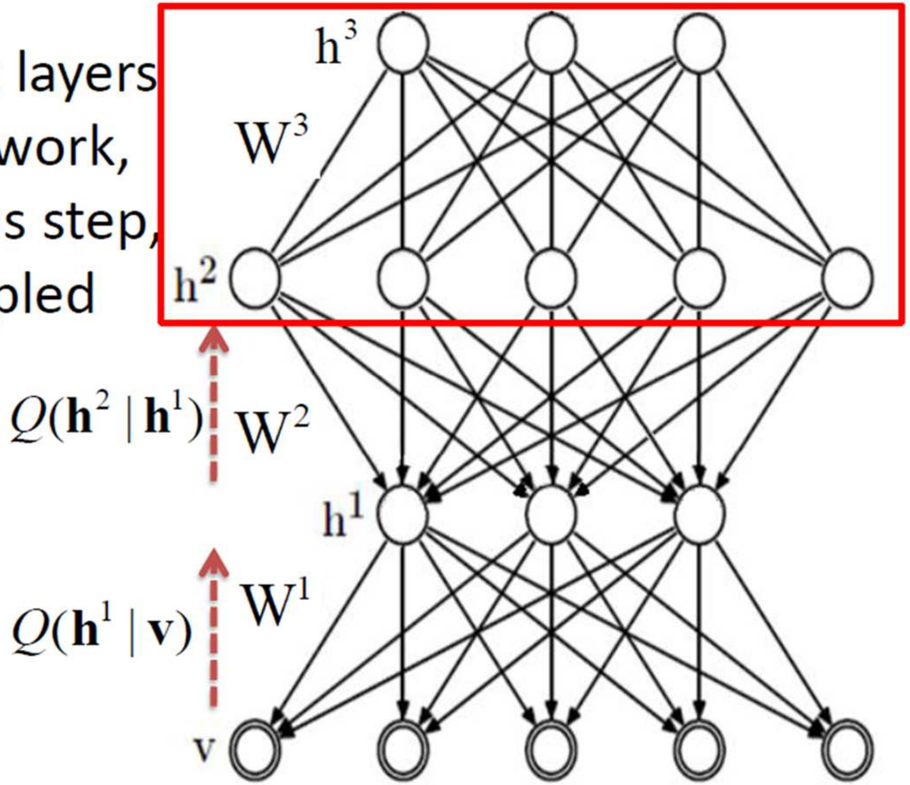
DBN greedy training

- Second step:
  - Stack another hidden layer on top of the RBM to form a new RBM
  - Fix W^1, sample $\mathbf{h}^1$ from $Q(\mathbf{h}^1 \mid \mathbf{v})$ as input. Train W^2 as RBM.



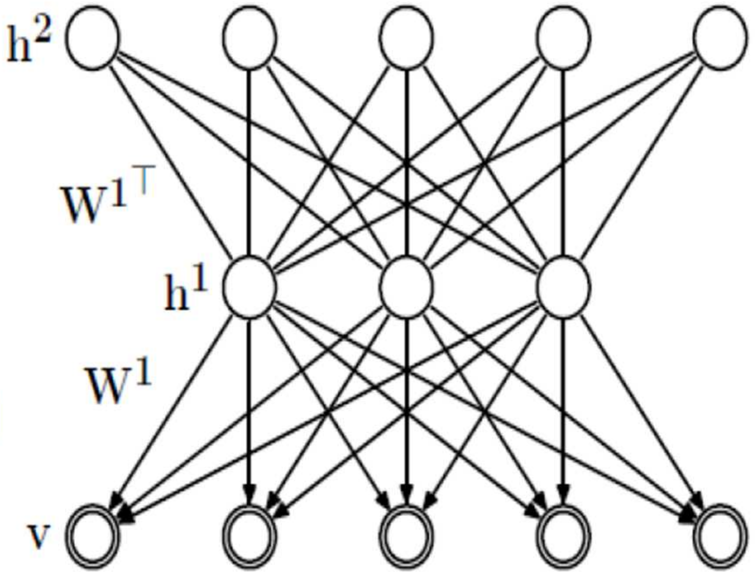
DBN greedy training

- Third step:
  - Continue to stack layers on top of the network, train it as previous step, with sample sampled from $Q(\mathbf{h}^2 \mid \mathbf{h}^1)$
- And so on…


h^3
W^3
h^2
$Q(\mathbf{h}^2 \mid \mathbf{h}^1)$
W^2
h^1
W^1
$Q(\mathbf{h}^1 \mid \mathbf{v})$
v

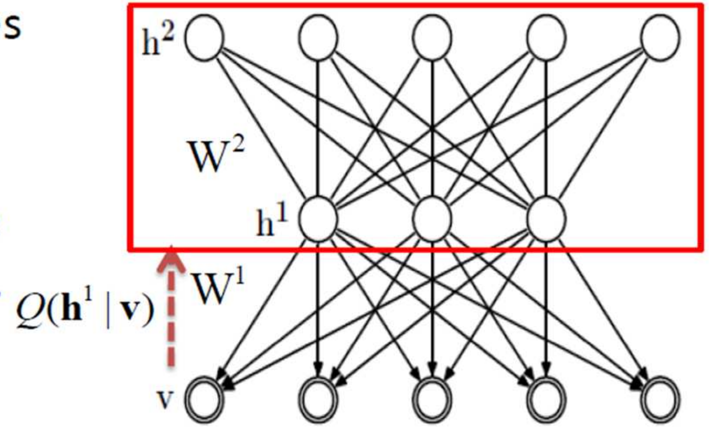
Why greedy training work

- RBM specifies P(v,h) from P(v|h) and P(h|v)
  - Implicitly defines P(v) and P(h)
- Key idea of stacking
  - Keep P(v|h) from 1st RBM
  - Replace P(h) by the distribution generated by 2nd level RBM



Why greedy training work

- Greey Training:
  - Variational lower-bound justifies greedy layerwise training of RBMs
  - Note: RBM and 2-layer DBN are equivalent when $W^2 = (W^1)^T$. Therefore, the lower bound is tight and the log-likelihood improves by greedy training.



$$\log P(\mathbf{x}) \geq H_{Q(\mathbf{h}|\mathbf{x})} + \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \left(\underline{\log P(\mathbf{h})} + \log P(\mathbf{x}|\mathbf{h}) \right)$$

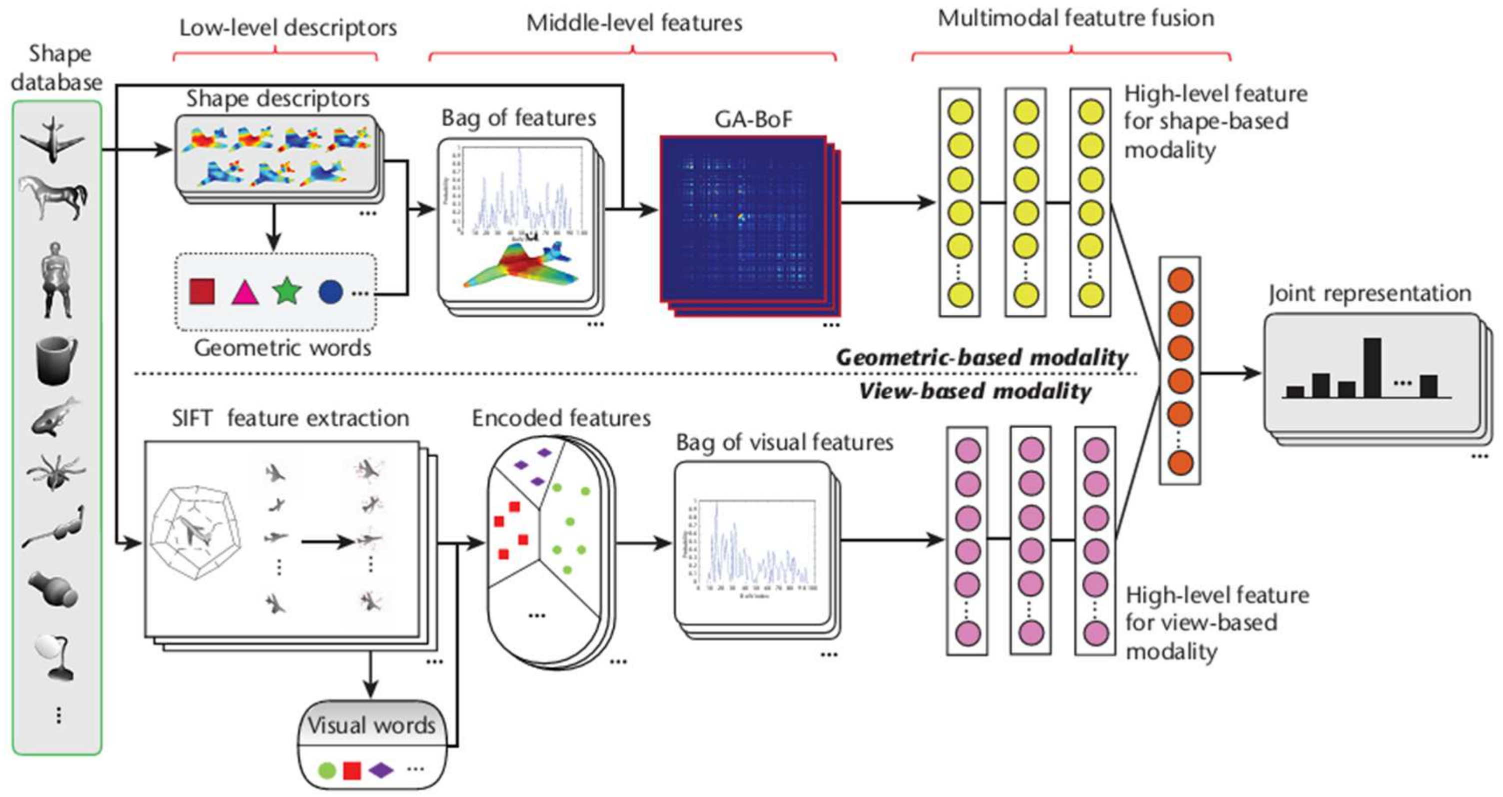
Trained by the second layer RBM

DBN and supervised fine-tuning

- Discriminative fine-tuning
  - Initializing with neural nets + backpropagation
  - Maximizes $\log P(Y \mid X)$ (X: data Y: label)

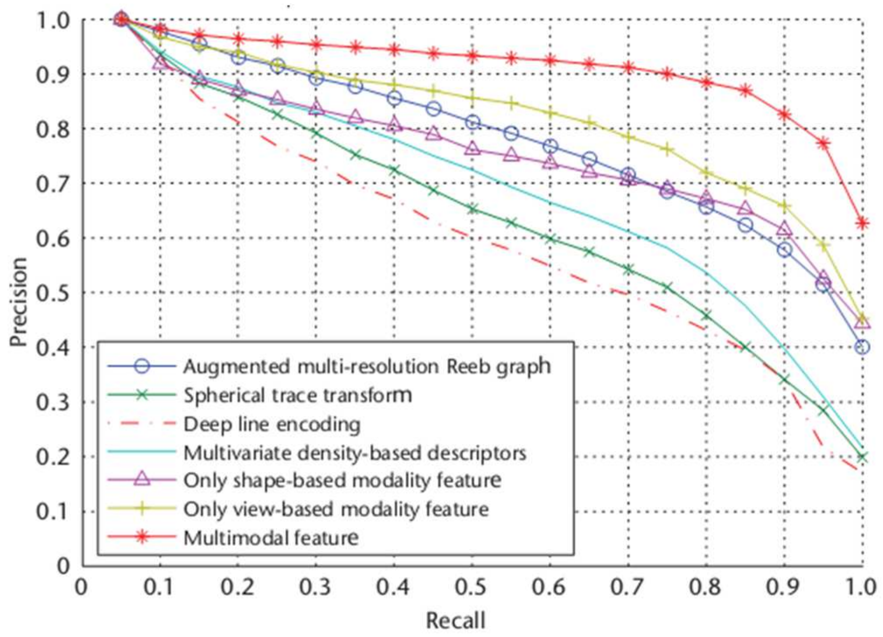
- Generative fine-tuning
  - Up-down algorithm
  - Maximizes $\log P(Y, X)$ (joint likelihood of data and labels)

Multi-modal Learning



Multi-modal Feature Fusion for 3D Shape Recognition and Retrieval

Multi-modal Learning



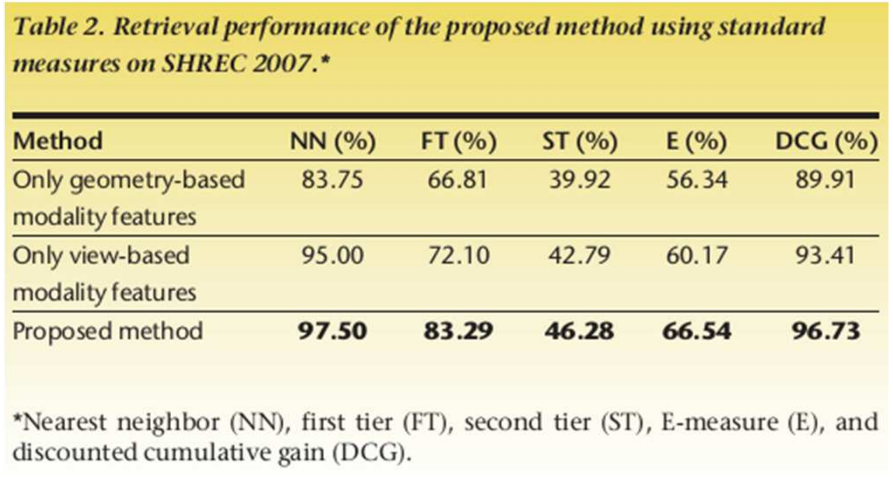


*Table 2. Retrieval performance of the proposed method using standard measures on SHREC 2007.**

Method	NN (%)	FT (%)	ST (%)	E (%)	DCG (%)
Only geometry-based modality features	83.75	66.81	39.92	56.34	89.91
Only view-based modality features	95.00	72.10	42.79	60.17	93.41
Proposed method	**97.50**	**83.29**	**46.28**	**66.54**	**96.73**

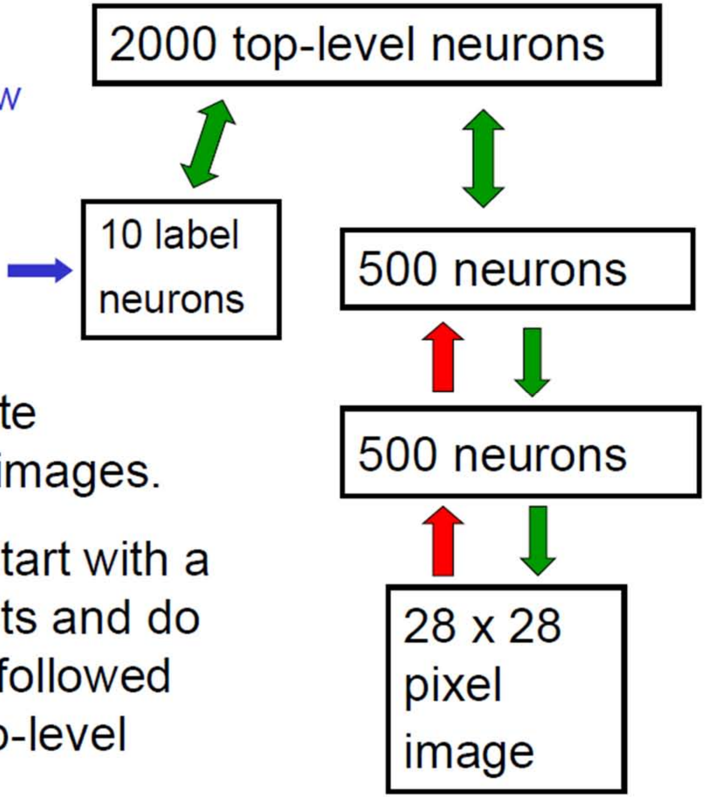
*Nearest neighbor (NN), first tier (FT), second tier (ST), E-measure (E), and discounted cumulative gain (DCG).

Multi-modal Feature Fusion for 3D Shape Recognition and Retrieval

A model for digit recognition

The top two layers form an associative memory whose energy landscape models the low dimensional manifolds of the digits.

The energy valleys have names



The model learns to generate combinations of labels and images.

To perform recognition we start with a neutral state of the label units and do an up-pass from the image followed by a few iterations of the top-level associative memory.

Result for supervised fine-tuning

• Very carefully trained backprop net with one or two hidden layers (Platt; Hinton)	1.6%
• SVM (Decoste & Schoelkopf, 2002)	1.4%
• Generative model of joint density of images and labels (+ generative fine-tuning)	1.25%
• Generative model of unlabelled digits followed by gentle backpropagation (Hinton & Salakhutdinov, Science 2006)	1.15%