CS290D – Advanced Data Mining

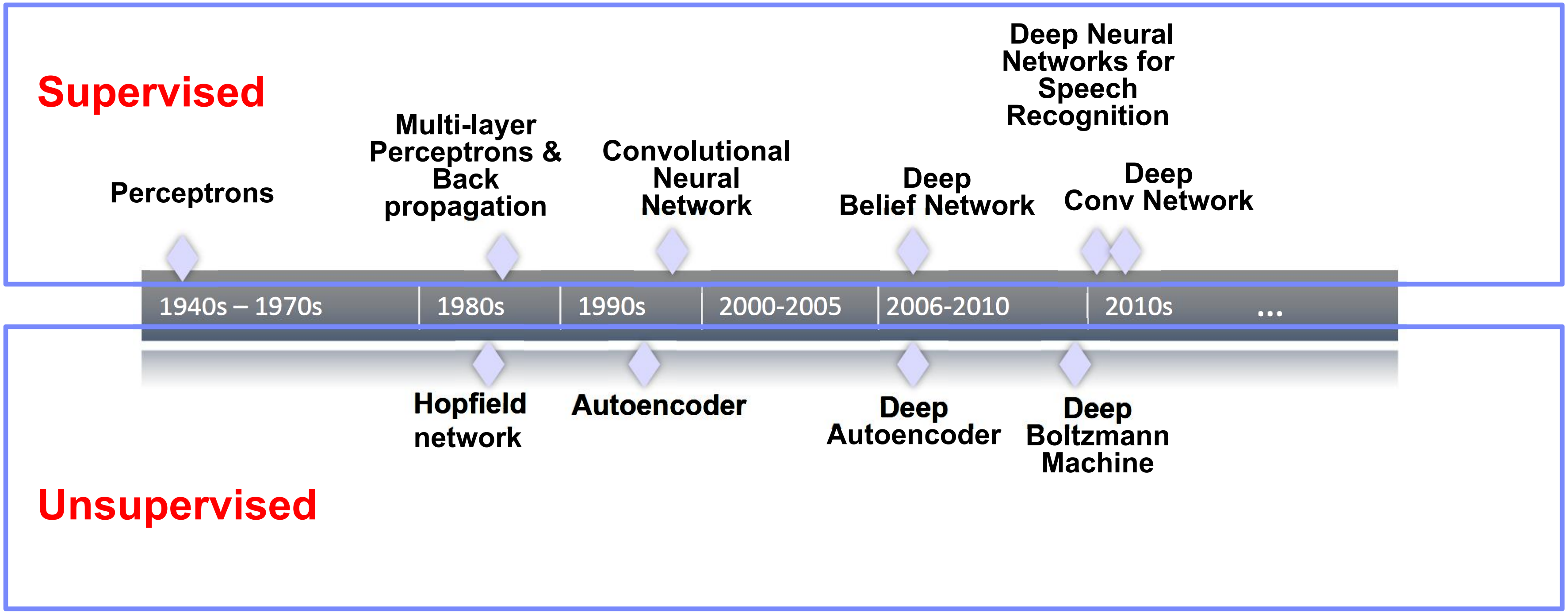
Instructor: Xifeng Yan
Computer Science
University of California at Santa Barbara

Deep Learning

Lecturer: Fangqiu Han
Computer Science
University of California at Santa Barbara

- The slides are made from:
  - Coursera online course, '**Neural Networks for Machine Learning',** Geoffrey Hinton
  - Deep Learning – ICML 2013 Tutorial, Yann LeCun

Neural network timeline



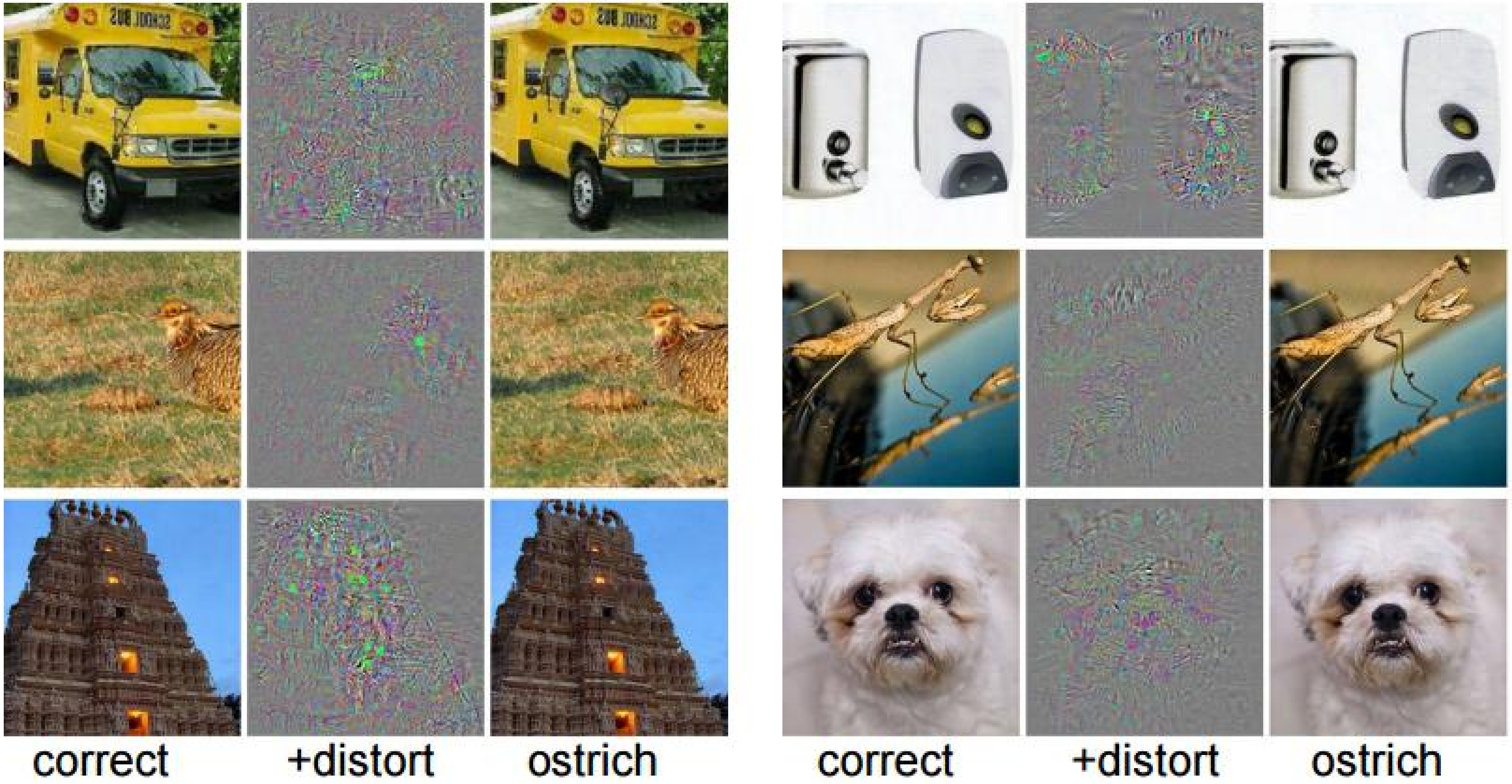
Why unsupervised learning?

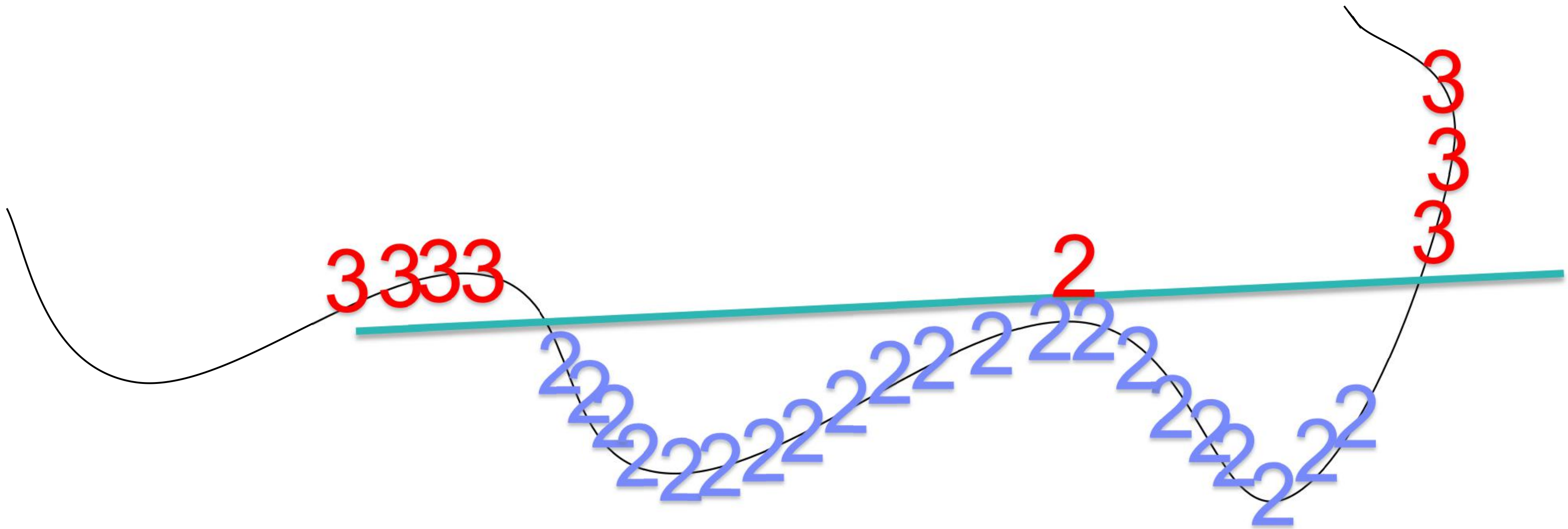
- Most data are unlabeled. We need unsupervised learning to help on supervised tasks.

- It is likely to be much more common in the brain than supervised learning.

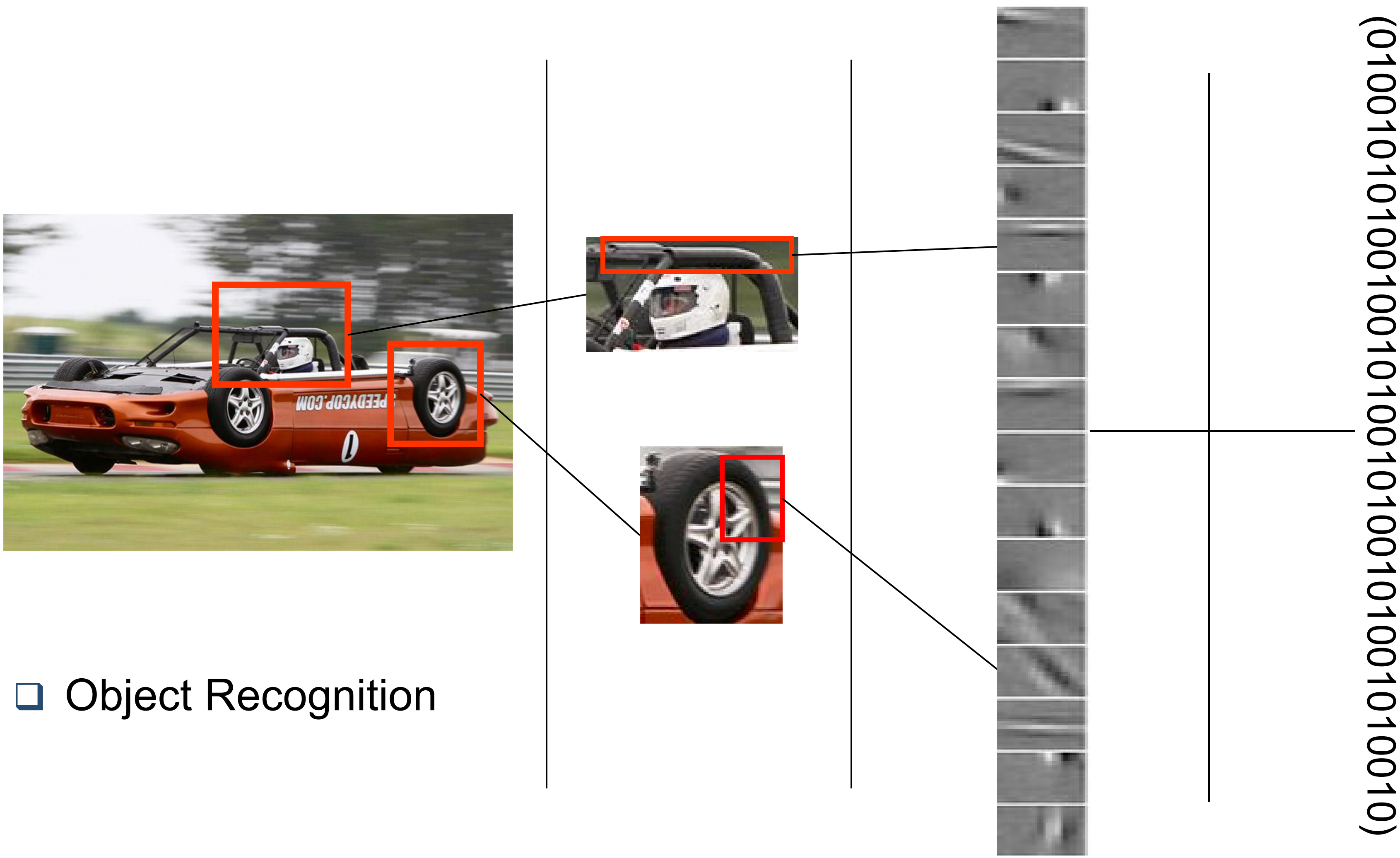
- It is easy to fool supervised learning methods as they only learn decision boundaries.

[Intriguing properties of neural networks, Szegedy et al., 2013]



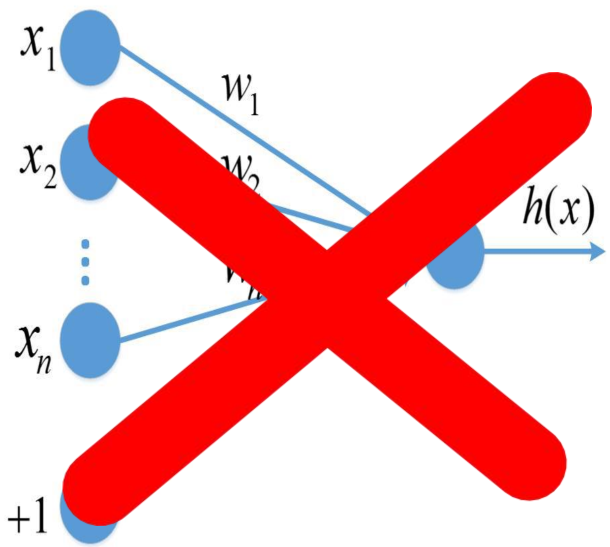

3333
2
3
3
3
2222222222222222222222222222

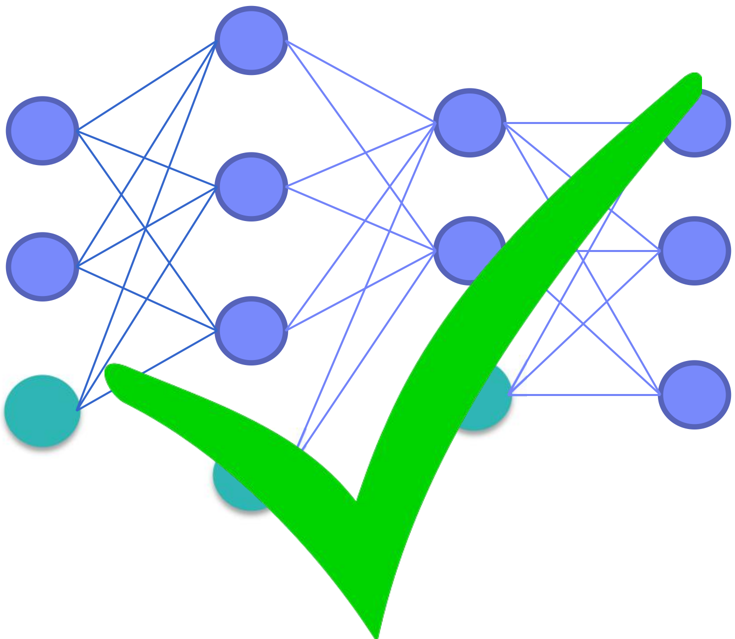
Why deep learning – Recognizing deep features



- ❑ Object Recognition

What is deep?

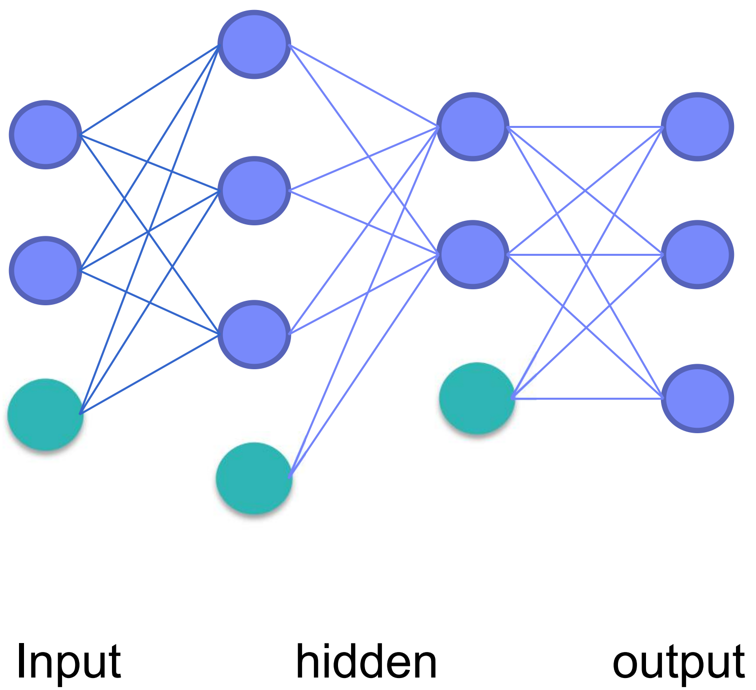




Which Models are Deep?

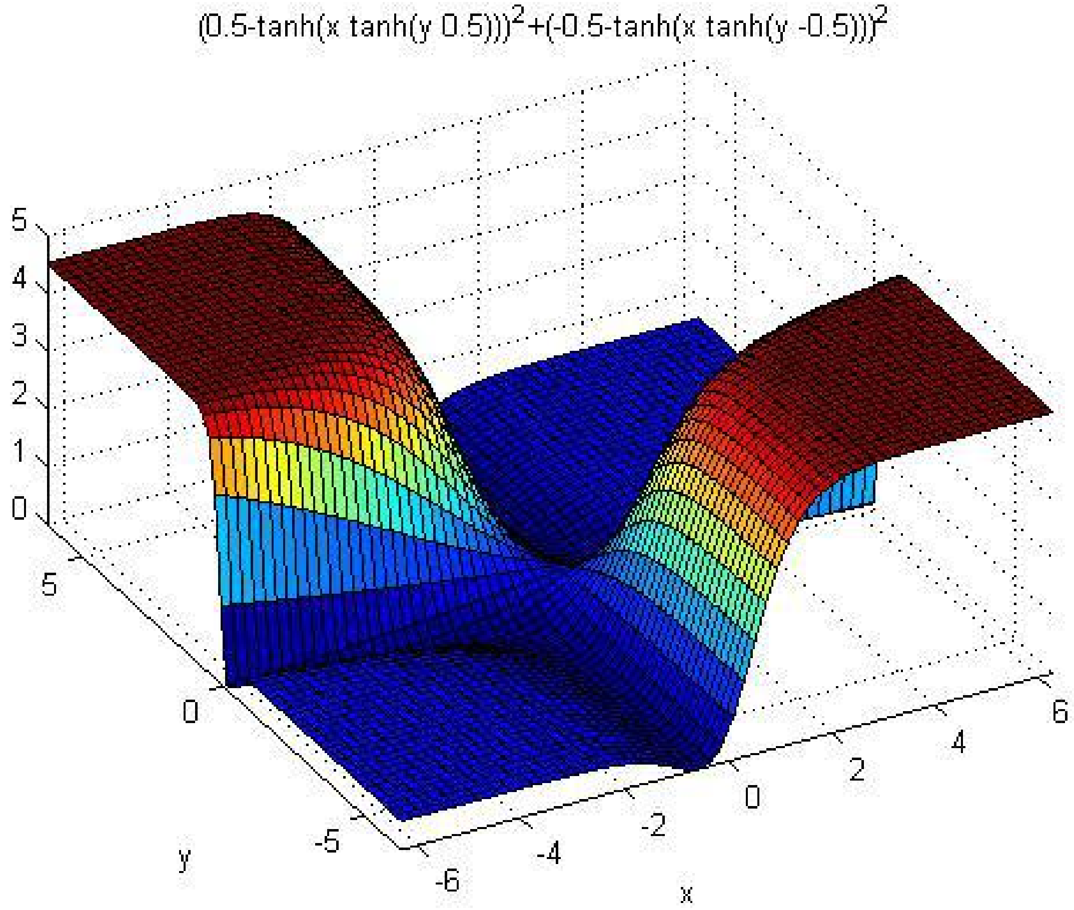
- Neural nets with 1 hidden layer are not deep
  - Because there is no feature hierarchy
- SVMs and Kernel methods are not deep
  - Layer1: kernels; layer2: linear
- Classification trees are not deep
  - No hierarchy of features. All decisions are made in the input space.

Why not multi-layer model with back-propagation

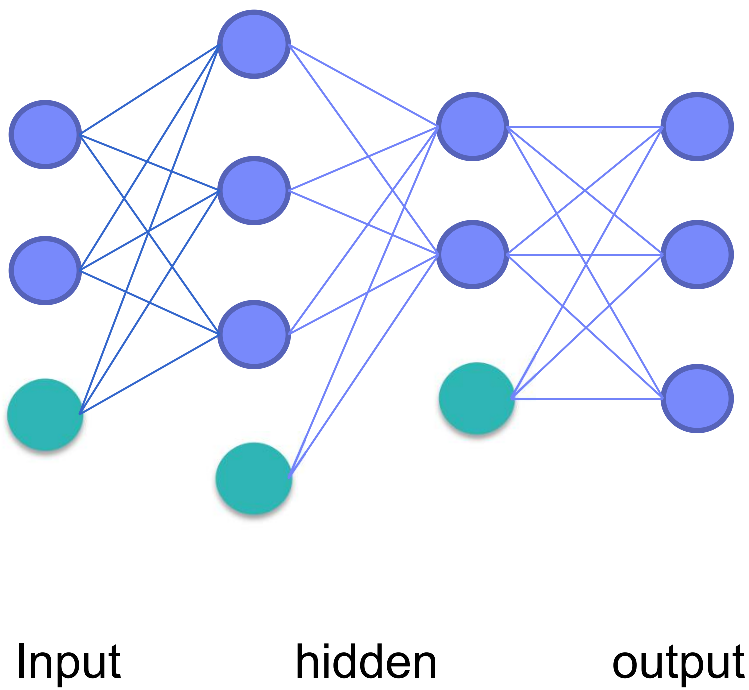


- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.

Deep Supervised Learning is Non-Convex

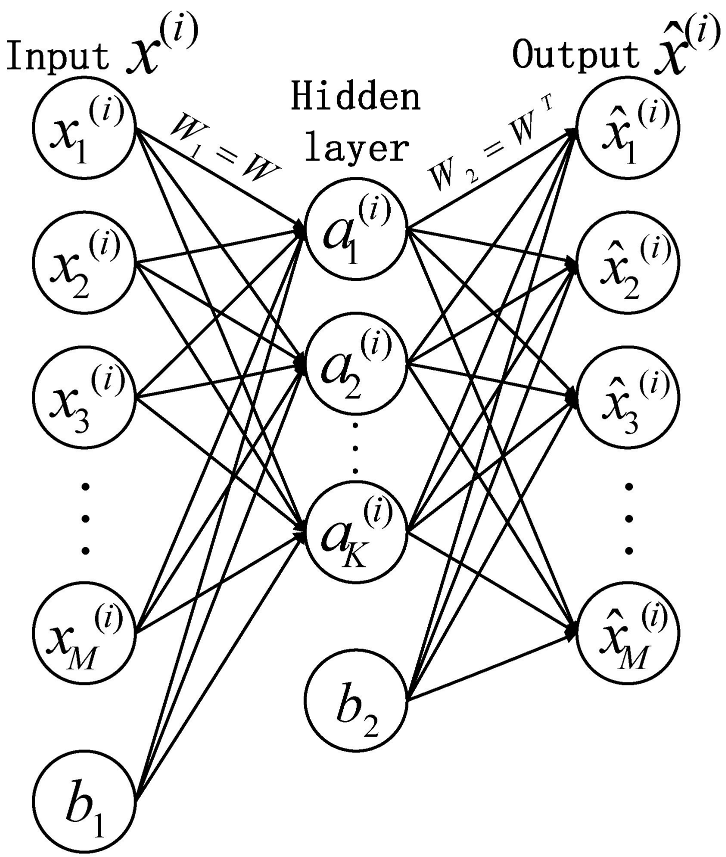


Why not multi-layer model with back-propagation



- 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.
- It requires labeled training data.
- Solutions:
  - Reduce the number of parameters
  - Use better initialization
  - Find other method for optimization
  - Find better structures

Autoencoder



Activation function:

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

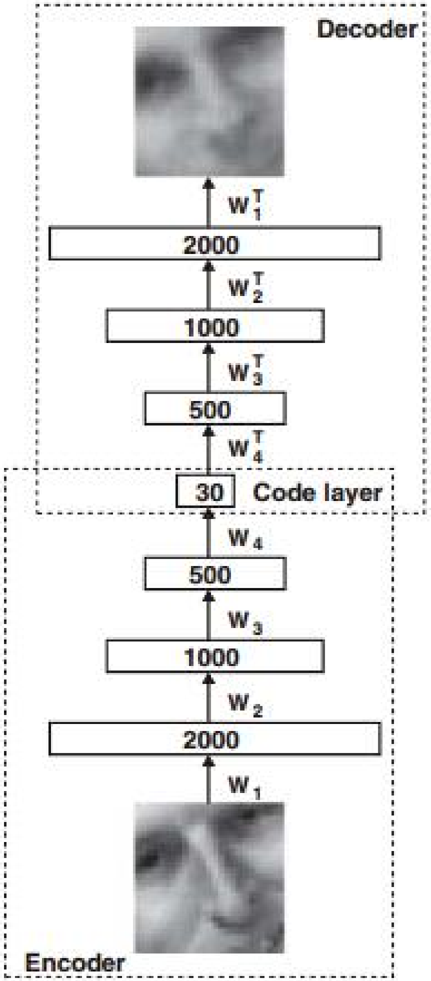
Forward pass:

$$\hat{x} = f(W^T f(W\,x))$$

Objective function:

$$\underset{W,b_1,b_2}{argmin} \quad H = \frac{1}{2N} * \sum_{n=1}^{N} \sum_{m=1}^{M} (\hat{x}_m^{(n)} - x_m^{(n)})^2 \quad (i)$$
$$+ \frac{\lambda}{2} * \|W\|_F^2 \quad (ii)$$

Deep Autoencoder



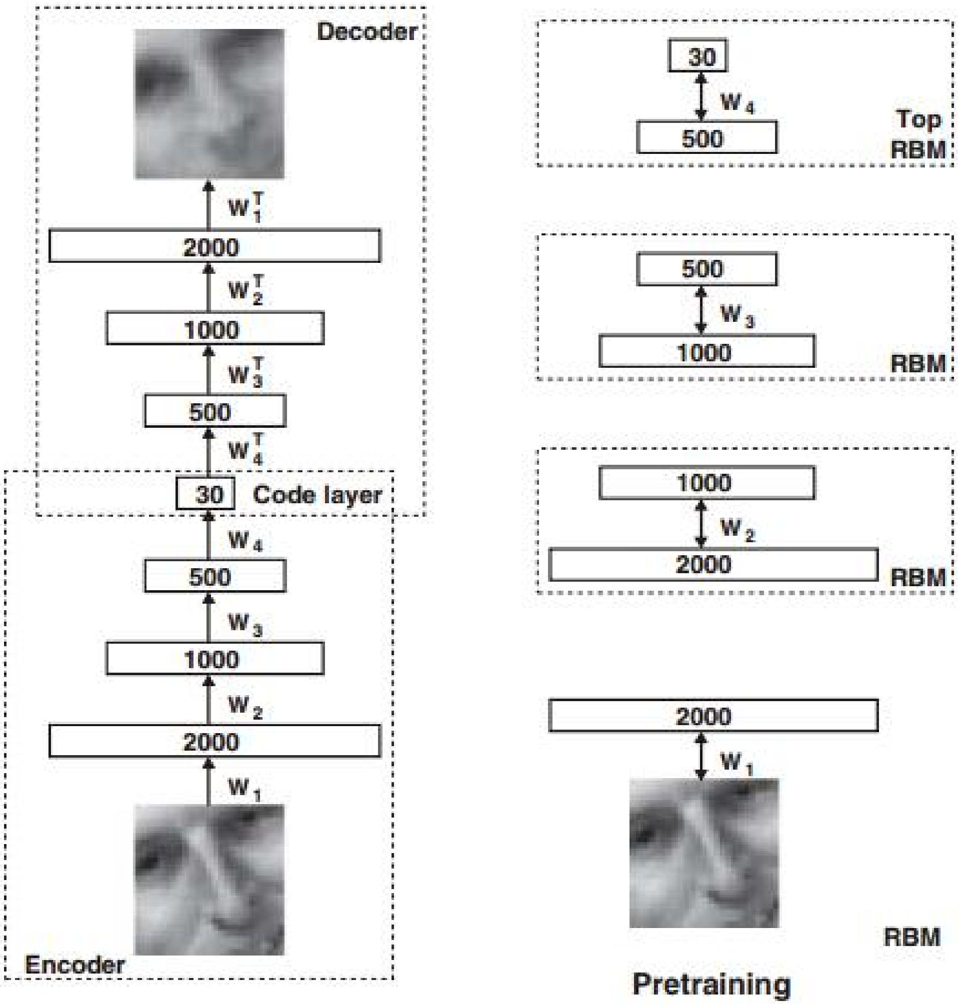
Forward pass:

$$\hat{X} = f(W_1{}^T f(W_2{}^T f(W_2 f(W_1\, x))))$$

Objective function:

$$\underset{W,b_1,b_2}{argmin} \quad H = \frac{1}{2N} * \sum_{n=1}^{N} \sum_{m=1}^{M} (\hat{x}_m^{(n)} - x_m^{(n)})^2 \quad (i)$$
$$+ \frac{\lambda}{2} * \|W\|_F^2 \quad (ii)$$

Train Deep Autoencoder



- Layer wised pretaining from lower level to higher level using Restricted Boltzmann machines (RBM)
- Fine tuning all the weights using back-propagation algorithm

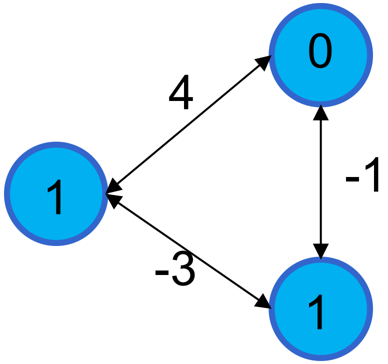
Outline

- Hopfield Network

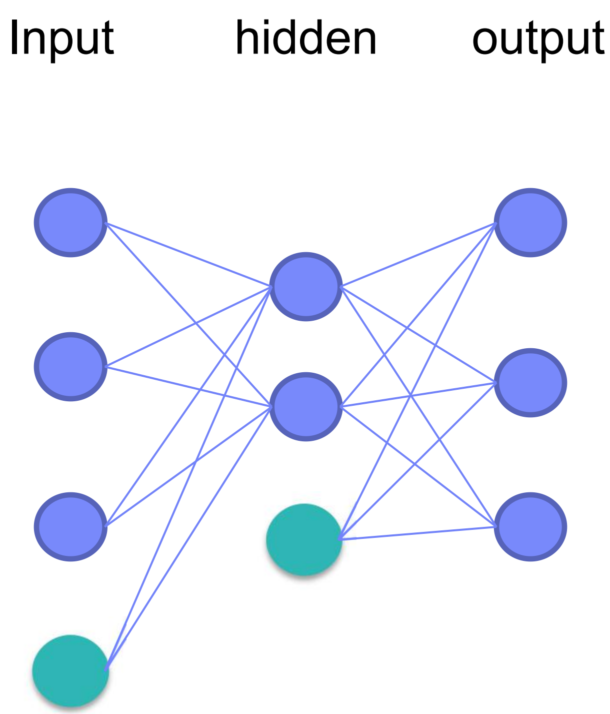
- Boltzmann machines:
  A stochastic Hopfield net with hidden units.

- Restricted Boltzmann machines:
  A variant of Boltzmann machines, with the restriction that their neurons must form a bipartite graph.

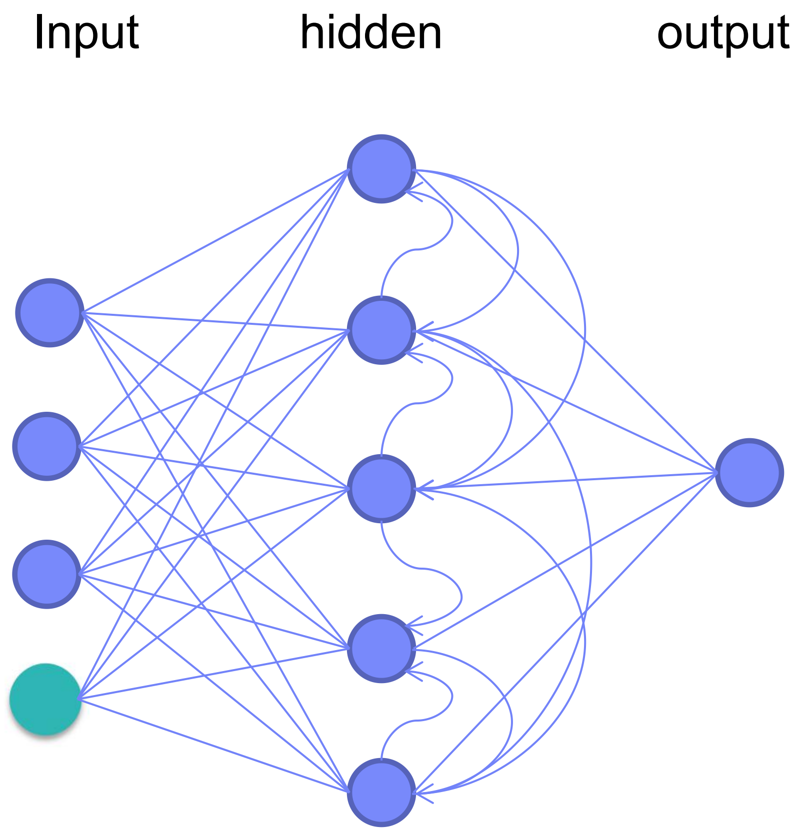
Hopfield Network



- A Hopfield net is composed of binary threshold units with recurrent connections between them.



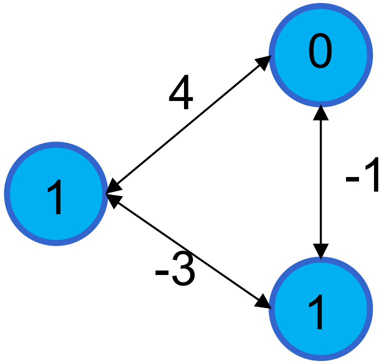
Multi-layer Perceptrons



Recurrent neural networks

Hopfield Network

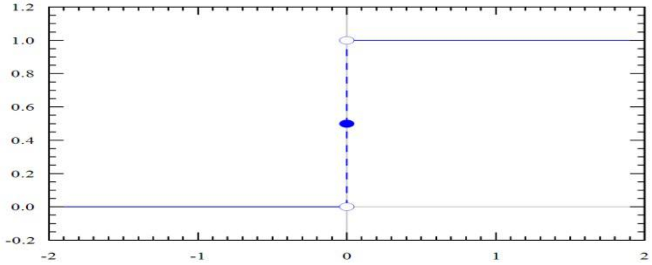
- A Hopfield net is composed of binary threshold units with recurrent connections between them.



Units

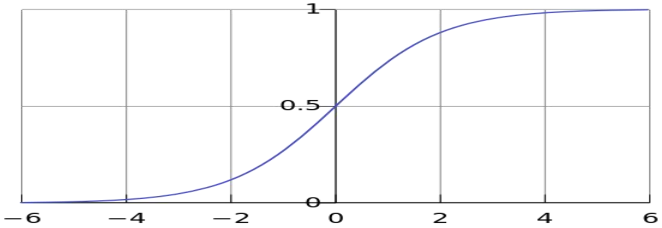
- Binary threshold units

$$f(z) = \begin{cases} 1, z > 0 \\ 0, z \leq 0 \end{cases}$$

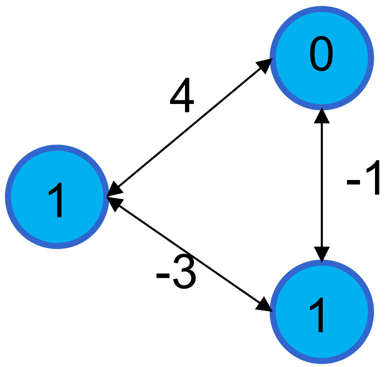


- Stochastic binary units

$$p(f(z) = 1) = \frac{1}{1 + e^{-z}}$$

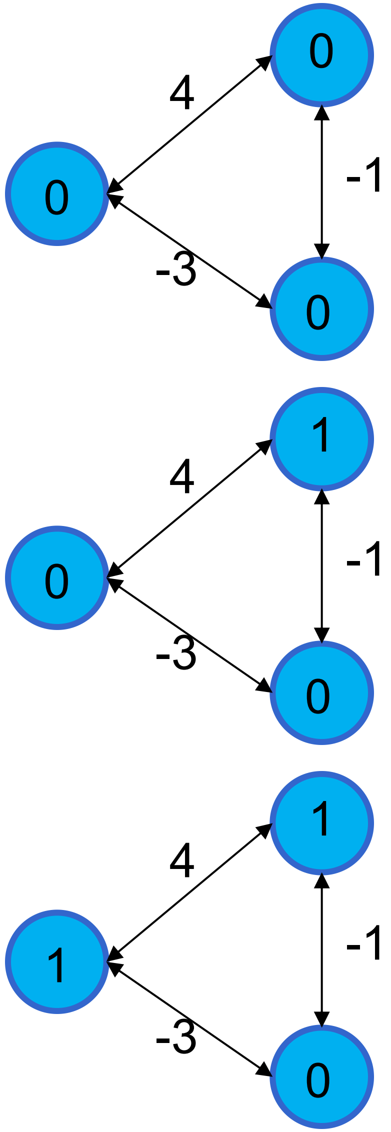


Hopfield Network



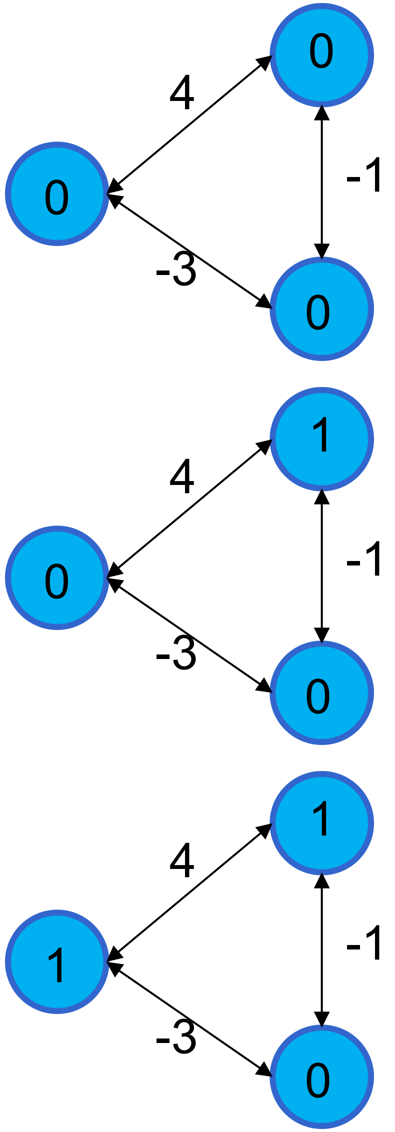
- A Hopfield net is composed of binary threshold units with recurrent connections between them.
- Configuration: an assignment of binary values to each neural.

Hopfield Network



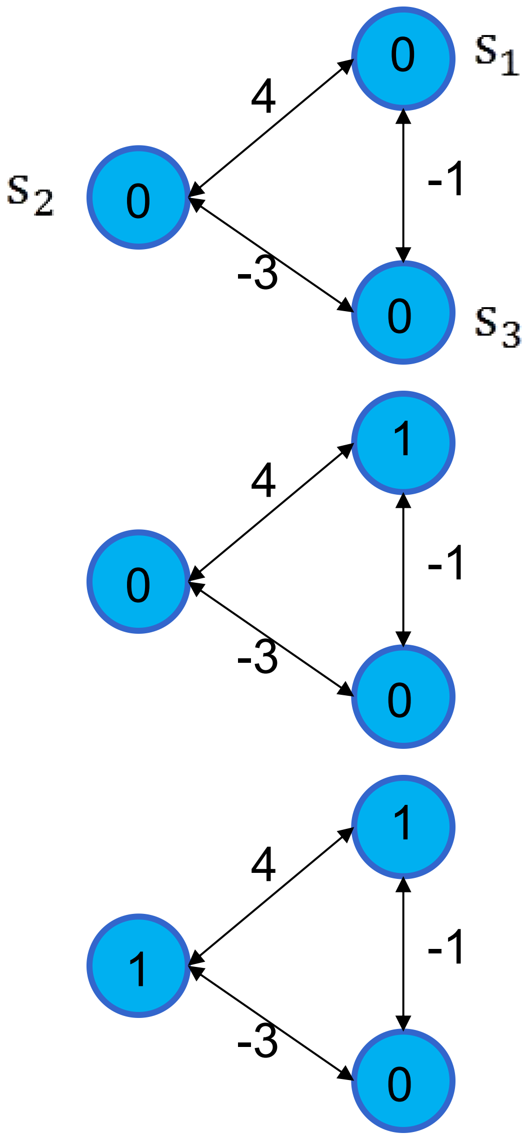
- A Hopfield net is composed of binary threshold units with recurrent connections between them.
- Configuration: an assignment of binary values to each neural.
- 'Run' the network: sequentially update each unit using activation function.

Hopfield Network



- Recurrent networks of non-linear units are generally very hard to analyze. They can behave in many different ways:
  - Settle to a stable state
  - Oscillate
  - Follow chaotic trajectories that cannot be predicted far into the future.
- John Hopfield (and others) realized that if the connections are symmetric, the network will converge.
  - There is a global energy function.
  - The binary threshold decision rule causes the network to settle to a minimal of this energy function

Hopfield Network Energy Function



- Each configuration has an energy:

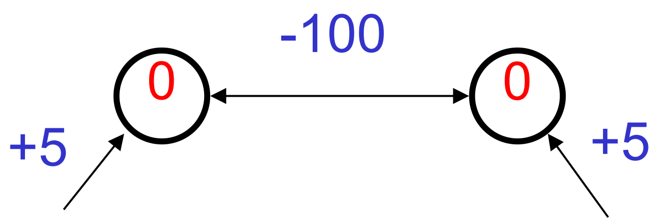
$$E = -\sum_i s_i b_i - \sum_{i<j} s_i s_j w_{ij}$$

- Energy gap:

$$\Delta E_i = E(s_i = 0) - E(s_i = 1) = b_i + \sum_j s_j w_{ij}$$

Why do the decisions need to be sequential?

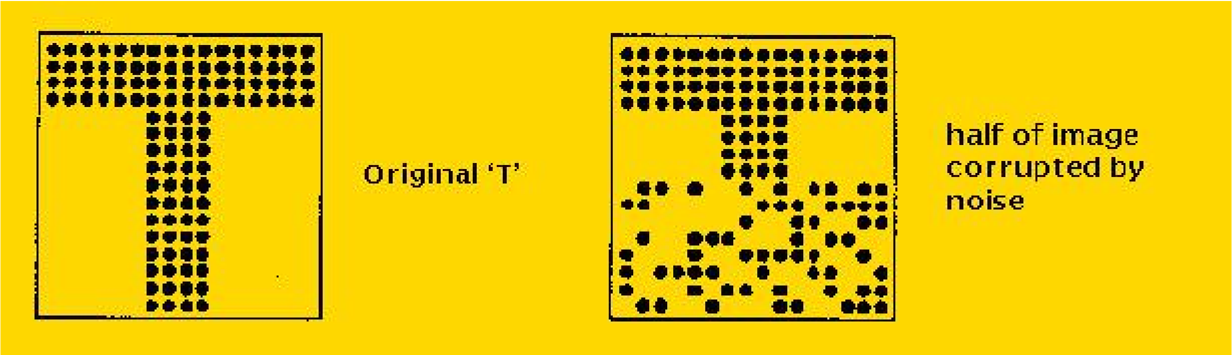
- If units make simultaneous decisions the energy could go up.
- With simultaneous parallel updating we can get oscillations.
- If the updates occur in parallel but with random timing, the oscillations are usually destroyed.



At the next parallel step, both units will turn on. This has very high energy, so then they will both turn off again.

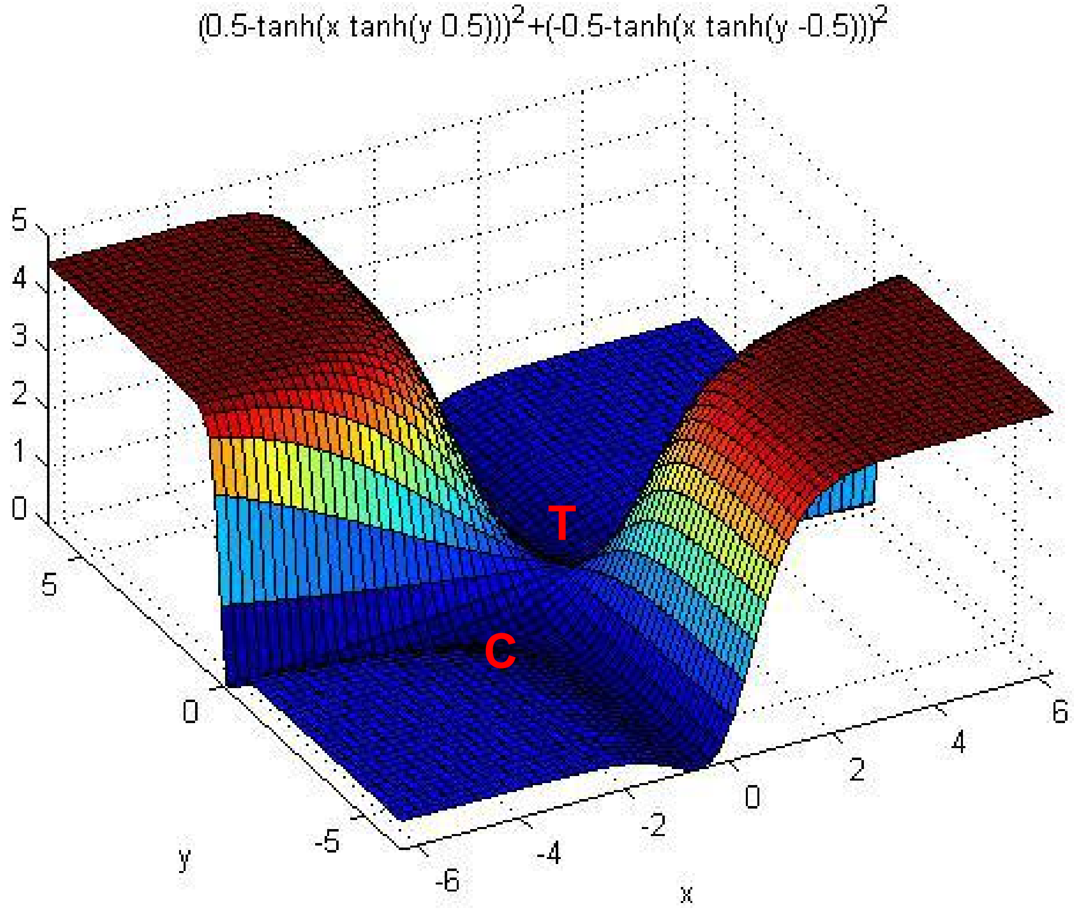
How to use Hopfield nets?

- Hopfield (1982) proposed that memories could be energy minima of a neural net.
  - The binary threshold decision rule can then be used to “clean up” incomplete or corrupted memories.
  - An item can be accessed by just knowing part of its content.



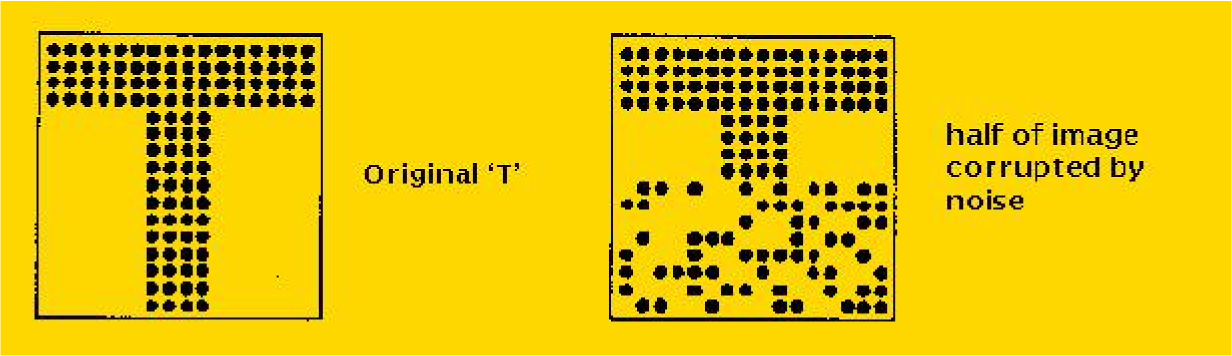
- Training: find {w, b} such that all data points will have low energy.

Hopfield net learning



How to use Hopfield nets?

- Hopfield (1982) proposed that memories could be energy minima of a neural net.
  - The binary threshold decision rule can then be used to “clean up” incomplete or corrupted memories.
  - An item can be accessed by just knowing part of its content.



- Training: find {w, b} such that all data points will have low energy.

Hopfield nets training

- Objective is to minimize:

$$\sum_k E(\text{data}_k)$$

- By definition:

$$E = -\sum_i s_i b_i - \sum_{i<j} s_i s_j w_{ij}$$

- Then we have

$$\Delta w_{ij} = -\frac{1}{m}\sum_k s_i^{(k)} s_j^{(k)}$$

Outline

- Hopfield Network
- Boltzmann machines:
  A stochastic Hopfield net with hidden units.
- Restricted Boltzmann machines:
  A variant of Boltzmann machines, with the restriction that their neurons must form a bipartite graph.

Stochastic binary neural to escape from local minima

- A Hopfield net always makes decisions that reduce the energy.
  - This makes it impossible to escape from local minima.

- We can use random noise to escape from poor minima.
  - Start with a lot of noise so its easy to cross energy barriers.
  - Slowly reduce the noise so that the system ends up in a deep minimum. This is also called the "simulated annealing" (Kirkpatrick et.al. 1981).

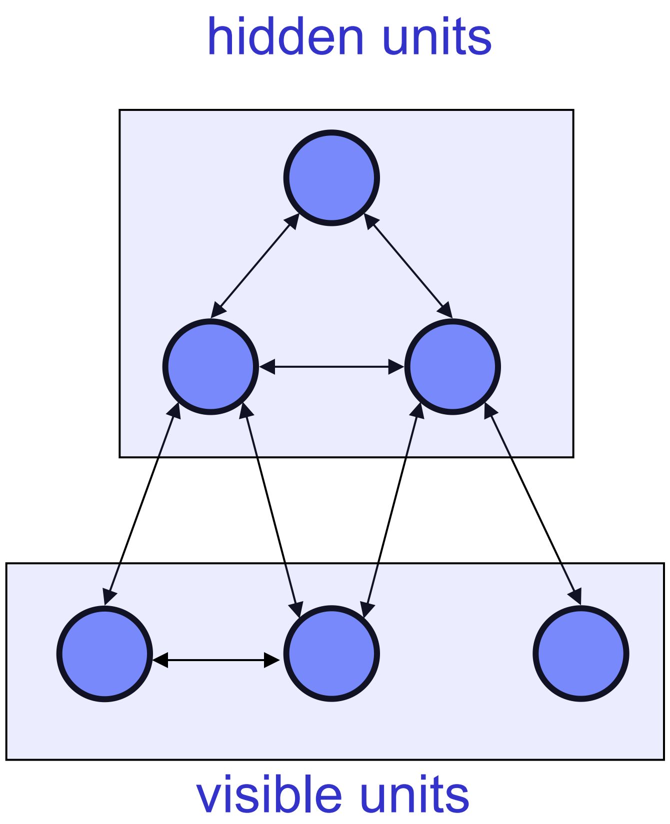
Stochastic binary neural

- Replace the binary threshold units by binary stochastic units that make biased random decisions.

$$p(f(z)=1)=\frac{1}{1+e^{-\Delta E/T}}$$

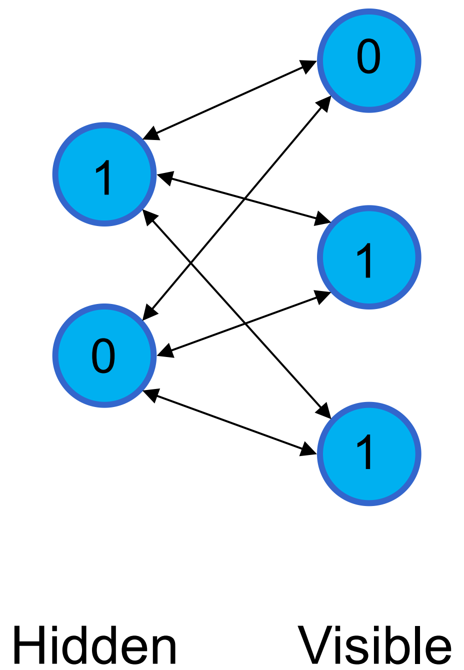
  - The "temperature" controls the amount of noise
  - Raising the noise level is equivalent to decreasing all the energy gaps between configurations.
  - Example

Boltzmann machines



- Instead of using the net to store memories, use it to construct interpretations of sensory input.
  - The input is represented by the visible units.
  - The interpretation is represented by the states of the hidden units.

Restricted Boltzmann Machines



- Notations:

$h_i, v_j \in \{0, 1\}$ = the activation of neural
w_{ij} = weight between hidden and visible layer

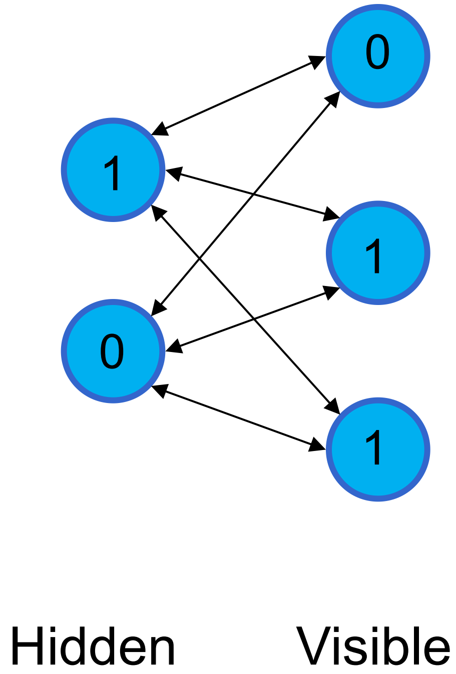
- Activation function :

$$\mathrm{P}(f(z) = 1) = \frac{1}{1 + e^{-z}}$$

- Definitions:

Configuration = an assignment of binary values to each neural

The energy function



- Each configuration has an energy

$$E(v,h) = -\sum b_i h_i - \sum a_i v_i - \sum_{i<j} w_{ij} h_i v_j$$

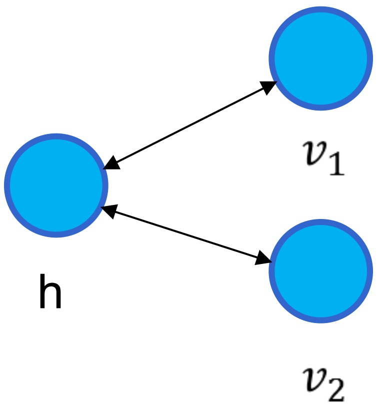
- Each configuration has a probability defined on its energy:

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

- Each data point (visible configuration) has a probability:

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

Model binary data



Hidden Visible

- Example
  - data point: (1,1) (0,0)
  - parameters (after training):

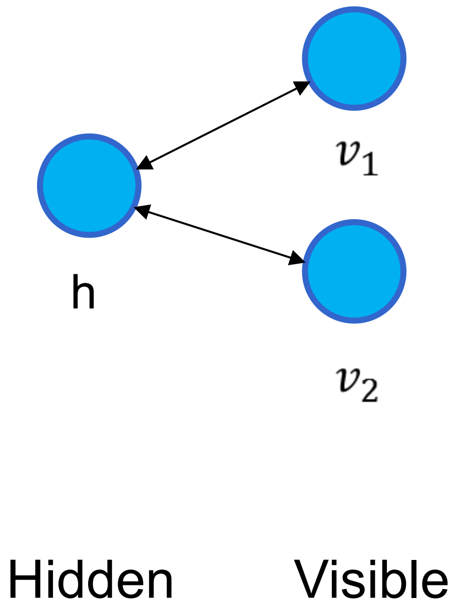
$$w_1 = w_2 = 20, a_1 = a_2 = -10, b = -20$$

  - $-\mathrm{E}(v,h) = \sum b_i h_i + \sum a_i v_i + \sum w_{ij} h_i v_j$

$$= 20hv_1 + 20hv_2 - 10v_1 - 10v_1 - 20h$$

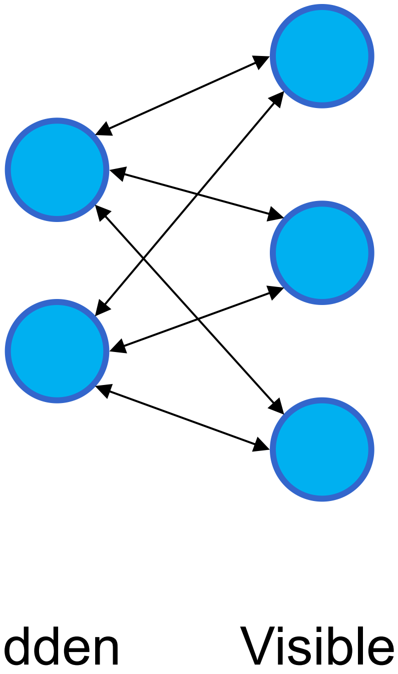
Model binary data

- $-\mathrm{E}(v,h) = 20hv_1 + 20hv_2 - 10v_1 - 10v_1 - 20h$



v	h	$-E$	e^{-E}	$p(\mathrm{v},\mathrm{h})$	$p(\mathrm{v})$
1 1	1	0	1	0.5	0.5
1 1	0	-20	0	0	
0 0	1	-20	0	0.5	0.5
0 0	0	0	1	0	
1 0	1	-10	0.00004	0	0
0 1	0	-10	0.00004	0	
0 1	1	-10	0.00004	0	0
1 0	0	-10	0.00004	0	
			2		

Sampling Restrict Boltzmann Machine



- Given activation in the hidden(visible) layer, all neural in the visible (hidden) layer are independent.
- Sample one neural is simple:

$$P(v_i = 1 \mid h) = f(\sum_j w_{ij} h_j + a_i)$$

$$P\left(h_j = 1 \mid v\right) = f(\sum_i w_{ij} v_i + b_j)$$

A very surprising fact about learning

$$\frac{\partial \log(p(v))}{\partial w_{ij}} = \langle v_i h_j \rangle_v - \langle v_i h_j \rangle_{model}$$

Derivative of log probability of one training vector, v under the model.

Expected value of product of neural i and j when v is clamped on the visible units

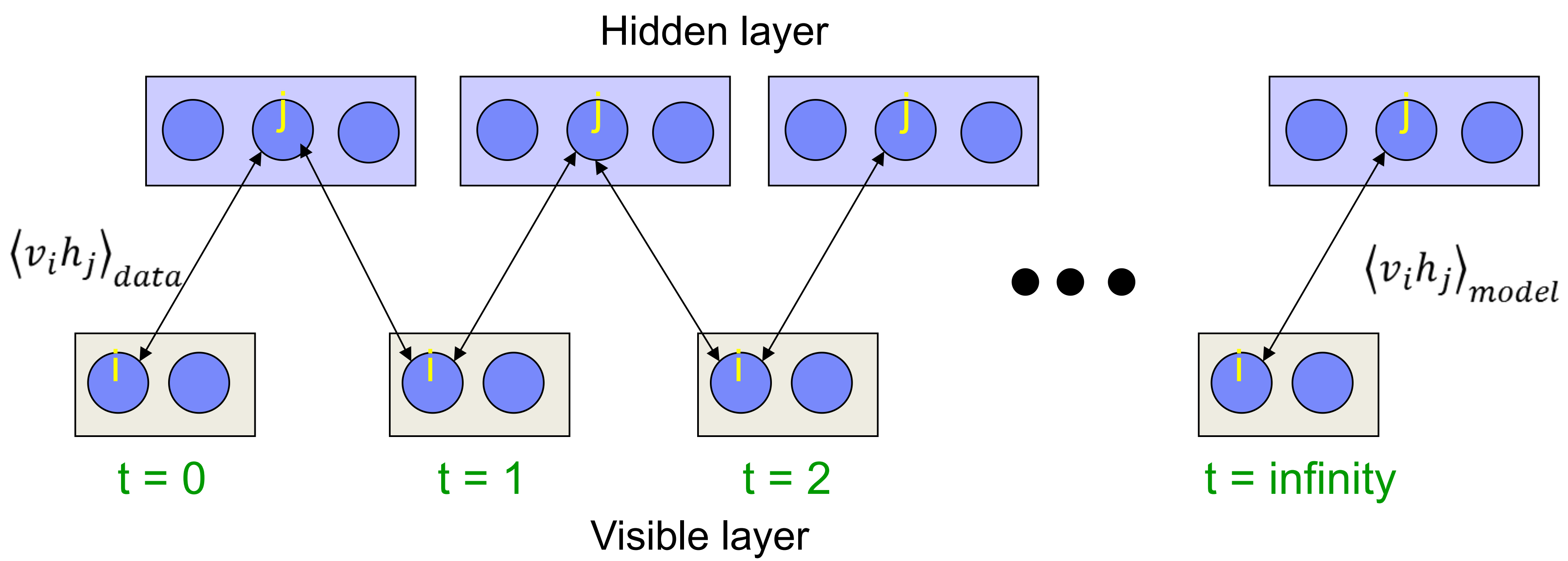
Expected value of product neural i and j with no clamping

$$-\mathrm{E}(v,h) = \sum b_i h_i + \sum a_i v_i + \sum_{i<j} w_{ij} h_i v_j$$

Contrastive Divergence (CD) Algorithm

- Positive phase $\langle v_i h_j \rangle_{data}$
  - Clamp a data vector v on the visible units.
  - Compute the exact value of $v_i h_j$ for all pairs of visible unit i and hidden unit j.
  - For every connected pair of units, average $\langle v_i h_j \rangle_{data}$ over all data.
- Negative phase $\langle v_i h_j \rangle_{model}$
  - Sampling RBM with no clamping.
  - Compute the exact value of $v_i h_j$ for all pairs of visible unit i and hidden unit j.
  - For every connected pair of units, average $\langle v_i h_j \rangle_{model}$ over all data.

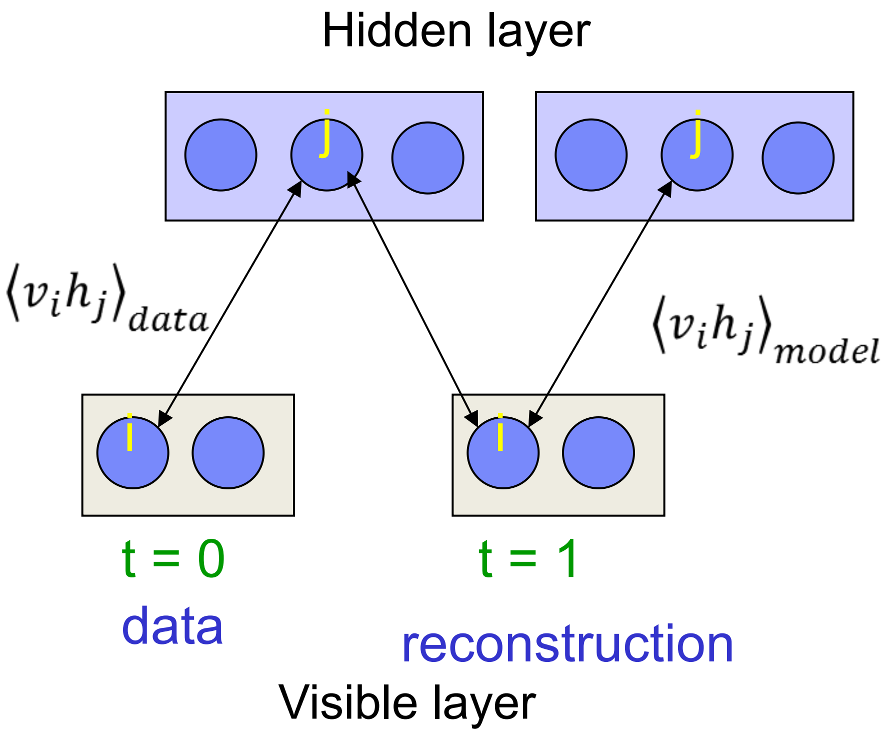
Contrastive Divergence (CD) Algorithm



Start with a training vector on the visible units. Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

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

A very surprising short-cut



Start with a training vector on the visible units.

Update all the hidden units in parallel.

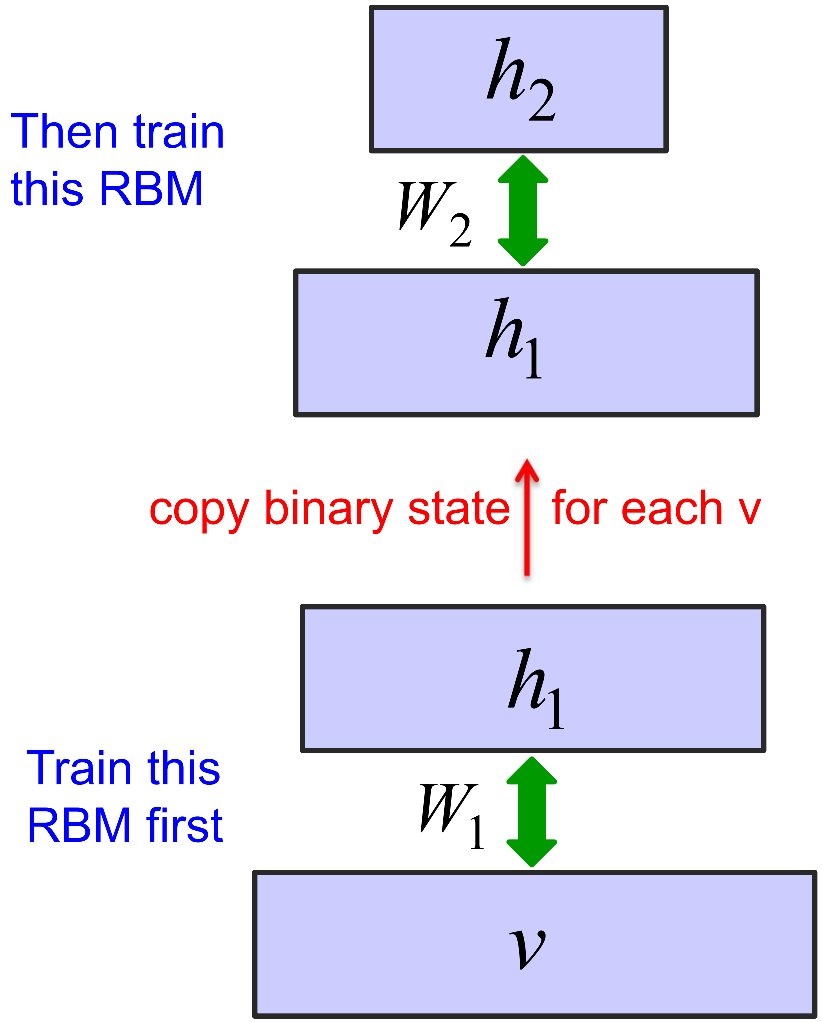
Update the all the visible units in parallel to get a "reconstruction".

Update the hidden units again.

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

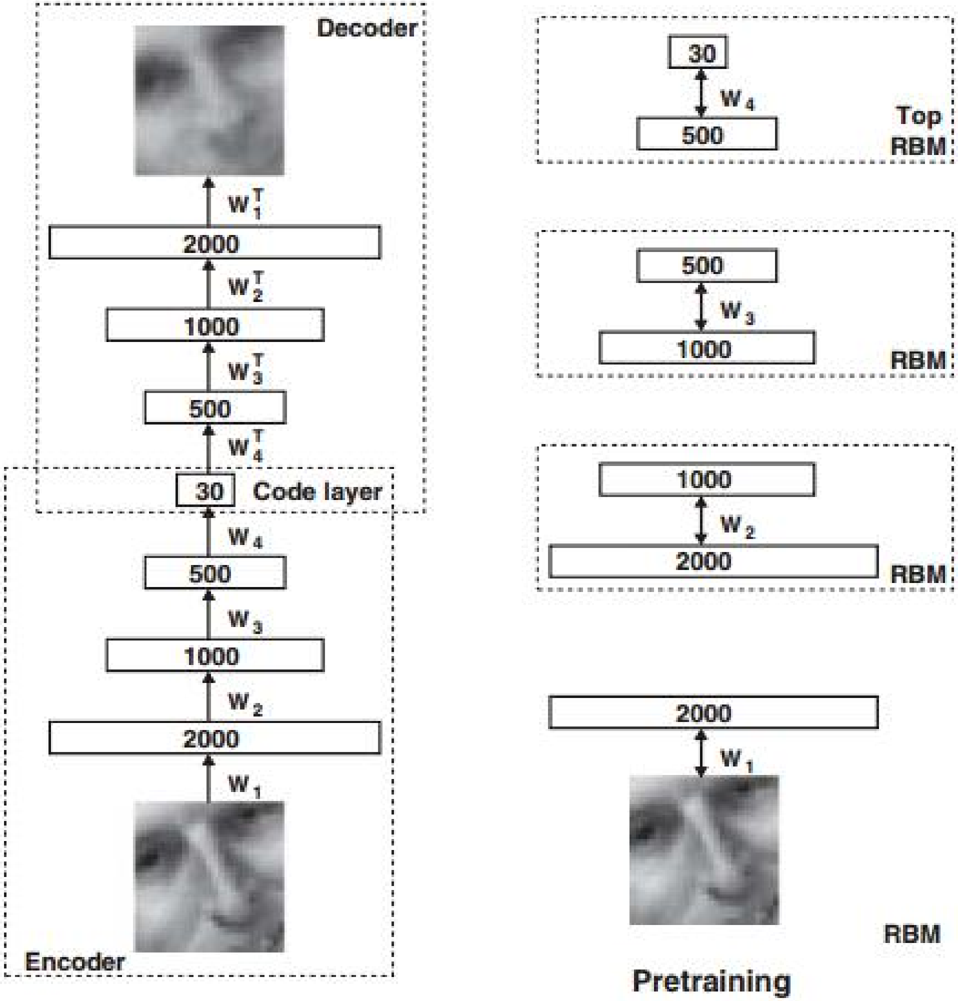
This is not following the gradient of the log likelihood. But it works well.

Pretraining a deep network by stacking RBMs

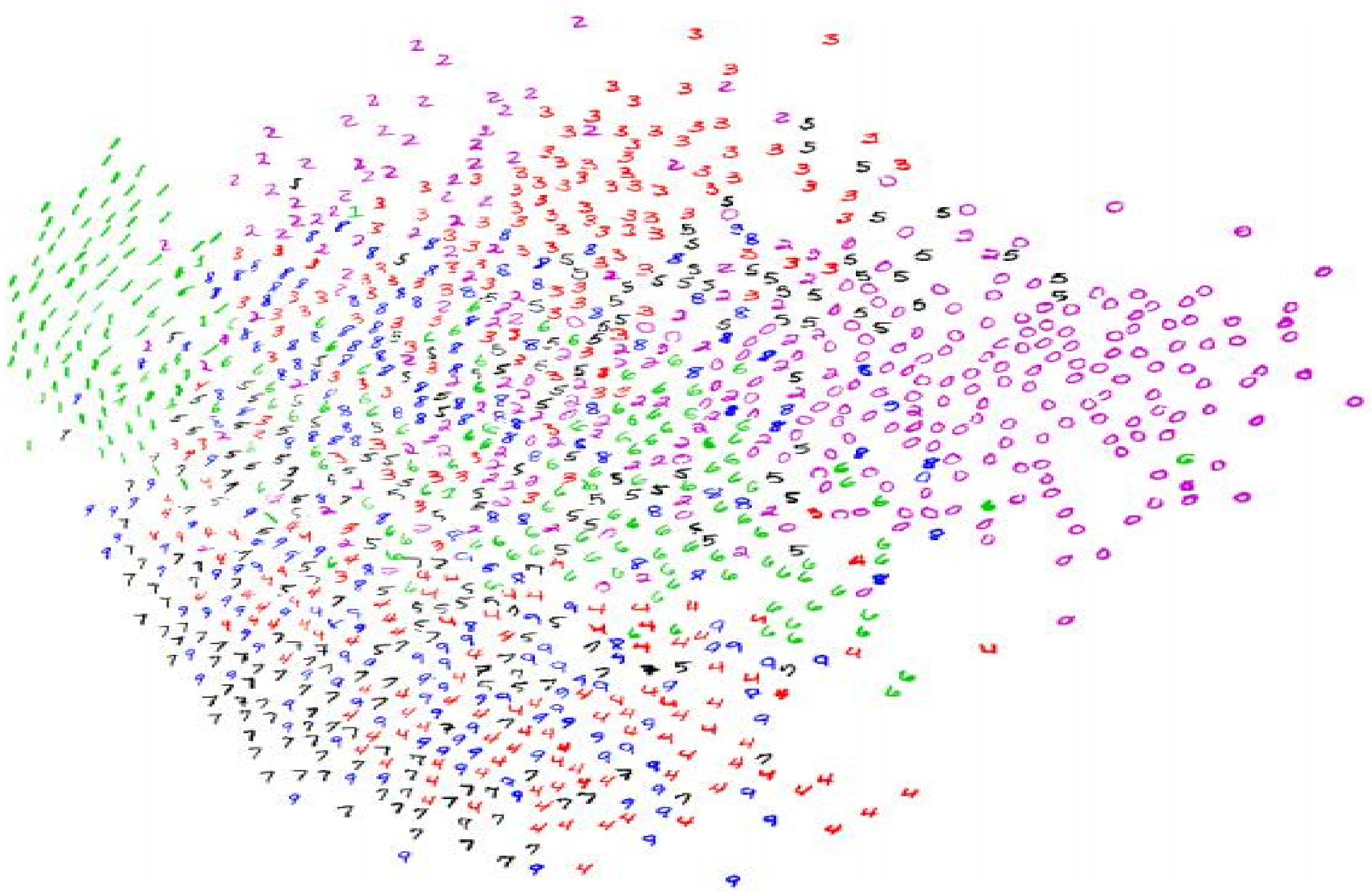


- First train a layer of features that receive input directly from the pixels.
- Then treat the activations of the trained features as if they were pixels and learn features of features in a second hidden layer.
- Then do it again.

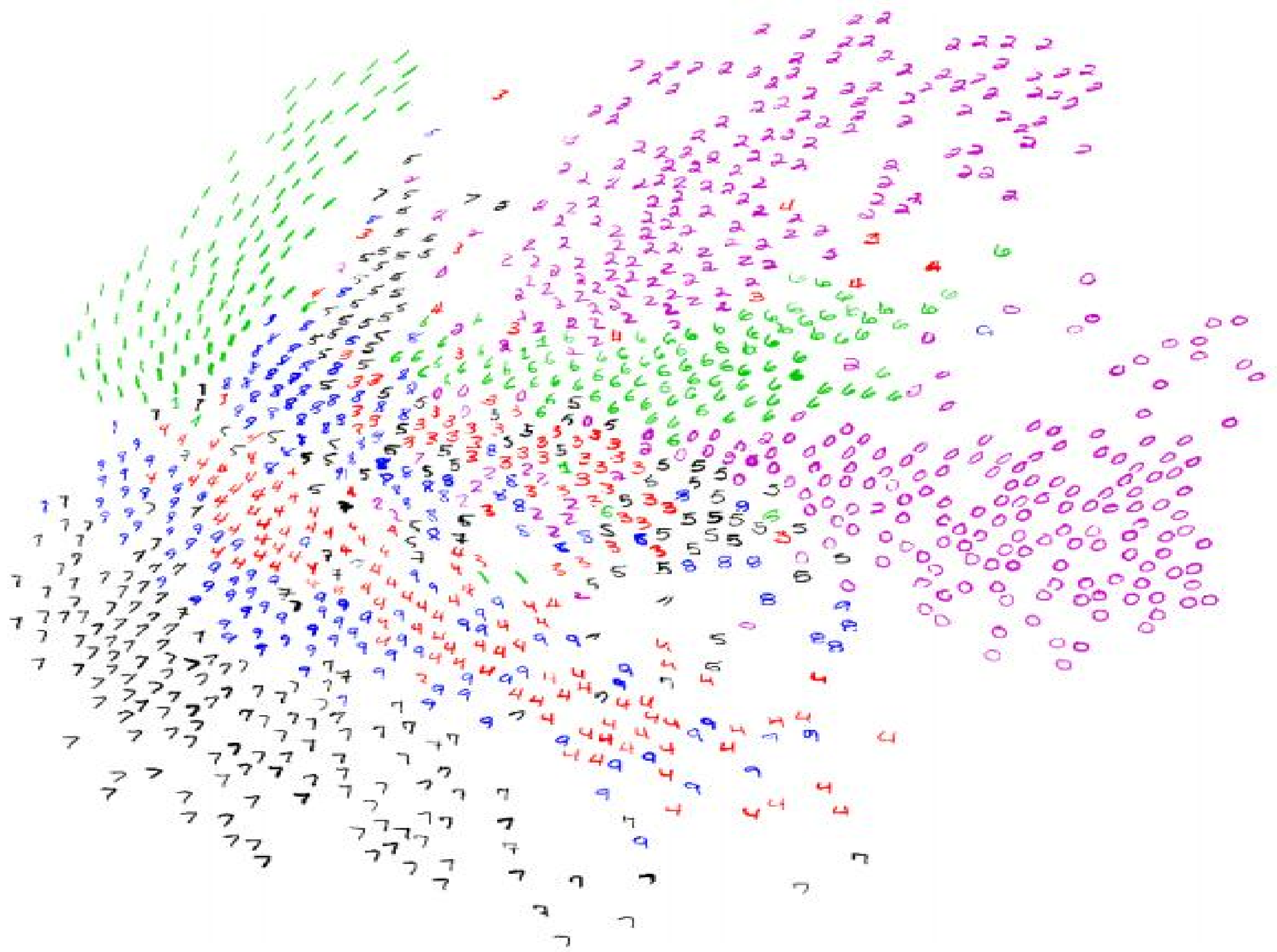
Train Deep Autoencoder



- Layer wised pretaining from lower level to higher level using Restricted Boltzmann machines (RBM)
- Fine tuning all the weights using back-propagation algorithm



Visualization of the 2-D codes produced 2-D PCA



Visualization of the 2-D codes produced by a 784-1000-500-250-2 AutoEncoder

Deep learning and feature leaning today

- Deep Learning has been the hottest topic in speech recognition in the last several years
  - A few long-standing performance records were broken with deep learning methods
  - Microsoft and Google have both deployed DL-based speech recognition system in their products
  - Microsoft, Google, IBM, Nuance, AT&T, and all the major academic and industrial players in speech recognition have projects on deep learning
- Deep Learning is the hottest topic in Computer Vision
  - Record holders on ImageNet and Semantic Segmentation are convolutional Neural nets
- Deep Learning is the hottest topic in Natural Language Processing

Deep Learning – A Theoretician`s Nightmare

- Deep Learning involves non-convex loss functions
- No generalization bounds
- It is hard to prove anything about deep learning systems
- If we only study models for which we can prove things, we wouldn't have speech, handwriting, and visual object recognition systems today

Recommendation readings / videos

- Coursera:
  - Neural Networks for Machine Learning, Geoffrey Hinton
  - Machine Learninig, Andrew Ng
- Tutorial:
  - http://deeplearning.net/tutorial
  - Deep Learning for NLP - NAACL 2013 Tutorial, Richard Socher and Christopher Manning
  - A tutorial on Deep Learning – NIPS 2009 Tutorial, Geoffrey Hinton
  - Representation Learning Tutorial – ICML 2012 Tutorial, Yoshua Bengio
  - Deep Learning – ICML 2013 Tutorial, Yann LeCun

Questions?