Helmholtz Machines

15-486/782 Artificial Neural Networks
David S. Touretzky
Fall 2006

Unsupervised Learning: Discovering Structure in the World

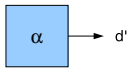
- Start with a set of input patterns, or “observations”.
- Try to “explain” them by inferring some structure by which they can be described.
- Examples:
  - Clustering models: competitive learning or k-means
    - Assign each point to one of k clusters (classification)
    - What's learned: an “optimal” clustering
  - Mixture models: Expectation-Maximization
    - Gaussian mixtures: parameters α_i, μ_i, σ_i
    - What's learned: mixture parameters to maximize likelihood of data
  - Generative models (this lecture)

Hermann von Helmholtz

- Helmholtz' theory of perception: analysis by synthesis.
- The brain “understands” sensory input by figuring out what process could have generated that input.
- Machine learning implementation: construct a stochastic parameterized generative model that can mimic the distribution of observed input patterns.
- Then we can describe/explain/transmit a pattern by giving the parameter settings that generate it.

Generative Models

- Build a generator that reproduces the input distribution.
- The generator's variables α serve as a description of the output pattern: an “explanation”.



- Recognition: map input pattern d to “explanation” α.
- Generation: map variable values α to pattern d'.
- Can transmit d efficiently by transmitting α plus an error term: the difference between d and d'.

Generative Model Example

- d = 64x64 bit image
  α = "cat", "dog", "bird", "fish"
- Dog's ears can be up or down.
- Transmit "dog" plus correction for ears. Cost?

$$C(\alpha,d|\theta) = C(\alpha) + C(d|\alpha,\theta)$$

Cost to transmit α

Cost to transmit correction given α and a known generative model θ

- Learning: train generator θ so as to minimize C(d|α,θ).

Review of Information Theory

Want to transmit 1 of N symbols with uniform prob.

Message x has probability p(x) = 1/N.

bits required to transmit x is the entropy:
H(x) = – log p(x) = – log(1/N) = log N

Let X be a random variable distributed as p(x).

$$\text{Entropy } H(X) = E_p[H(x)]$$

$$= E_p[-\log p(x)]$$

$$= -\sum_x p(x)\log p(x)$$

Example: 8-Sided Die

$$x \in \{1,2,3,4,5,6,7,8\}, \quad p(x) = \frac{1}{8}$$

$$H(5) = -\log p(5) = -\log\frac{1}{8} = 3 \text{ bits.}$$

$$H(X) = -\sum_x p(x)\log p(x)$$

$$= -\sum_{x=1}^{8}\frac{1}{8}\log\frac{1}{8}$$

$$= -\frac{1}{8}\sum_{x=1}^{8}\log\frac{1}{8}$$

$$= -\frac{1}{8}\cdot 8\cdot -3 = 3 \text{ bits}$$

Entropy of Binary Random Variables

$x \in \{0,1\}$, x=1 with probability p

So x=0 with probability $(1-p)$

$$H(x) = H(1) + H(0)$$

$$= -p\log p + -(1-p)\log(1-p)$$

Entropy of a Biased Coin

$$p(1) = \frac{1}{4}, \quad p(0) = \frac{3}{4}$$

$$\begin{aligned} H(x) &= -\frac{1}{4}\log\frac{1}{4} + -\frac{3}{4}\log\frac{3}{4} \\ &= -\frac{1}{4}(-2) + -\frac{3}{4}(-0.415) \\ &= 0.5 + 0.31 \\ &= 0.81 \text{ bits} \end{aligned}$$

Biased coins generate less information than a fair coin (1 bit).

Cost of Transmitting a Binary Variable

If bit s_j is 1, the cost is $-\log p_j$

If bit s_j is 0, the cost is $-\log(1-p_j)$

Can combine these cases and write the cost as:

$$C(s_j) = s_j \log p_j + (1-s_j)\log(1-p_j)$$

Kullback-Leibler Divergence

Difference between true distribution P and approximation Q.

$$\begin{aligned} KL[Q,P] &= \sum_\alpha Q(\alpha)\log\frac{Q(\alpha)}{P(\alpha)} \\ &= \sum_\alpha Q(\alpha)\log Q(\alpha) - \sum_\alpha Q(\alpha)\log P(\alpha) \\ &= E_Q[\log Q] - E_Q[\log P] \end{aligned}$$

Always non-negative. Zero iff $P \equiv Q$. Not symmetric.

KL Divergence for Binary Distributions

$$KL[q,p] = q\log\frac{q}{p} + (1-q)\log\frac{1-q}{1-p}$$

Suppose $p(1)=\frac{1}{3}$ and $p(0)=\frac{2}{3}$

While $q(1)=\frac{1}{4}$ and $q(0)=\frac{3}{4}$

$$\begin{aligned} KL[q,p] &= \frac{1}{4}\log\left(\frac{1/4}{1/3}\right) + \frac{3}{4}\log\left(\frac{3/4}{2/3}\right) \\ &= \frac{1}{4}\log(3/4) + \frac{3}{4}\log(9/8) \\ &= 0.0237 \\ KL[p,q] &= 0.0251 \end{aligned}$$

Generative Distribution P

$$\log p(d|\theta) = \log\left(\sum_\alpha p(\alpha|\theta)p(d|\alpha,\theta)\right)$$

Define the energy of explanation α for d as:

$$E_\alpha = -\log p(\alpha|\theta)p(d|\alpha,\theta) = -\log p(\alpha,d|\theta)$$

By Boltzmann, $P_\alpha = \dfrac{\exp(-E_\alpha)}{\sum_\beta \exp(-E_\beta)} = \dfrac{p(\alpha,d|\theta)}{\sum_\beta p(\beta,d|\theta)}$

$$\log p(d|\theta) = -\left[\underbrace{\sum_\alpha P_\alpha E_\alpha - \left(-\sum_\alpha P_\alpha \log P_\alpha\right)}\right]$$

Helmholtz free energy F(d; θ,P)

Helmholtz Free Energy

- Helmholtz free energy is the difference between
  - the expected energy of the explanations for d, and
  - the entropy of the probability distribution across explanations.
- Analogy from statistical mechanics:

  As a hot gas expands and pushes a cylinder, a certain amount of energy is required to reconfigure the gas molecules.

  What's left, the free energy, is available to do work such as moving the cylinder.

Problem Estimating P_α

- Computing P_α may be intractable: there can be an exponential number of states.
- Solution: pick a distribution Q that is tractable to compute. If Q is close to P, we may be okay.

$$\log p(d|\theta) = \underbrace{-\sum_\alpha Q_\alpha E_\alpha - \sum_\alpha Q_\alpha \log Q_\alpha} + \sum_\alpha Q_\alpha \log[Q_\alpha/P_\alpha]$$

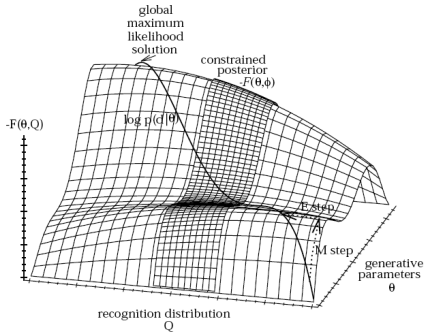
$$= \underbrace{-F(d;\theta,Q)} + KL[Q,P]$$

Helmholtz free energy

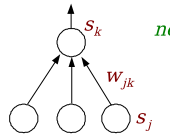
Recognition Distribution Q

- Goal is to train a generative model to mimic the distribution of patterns d, then use its variable values α to transmit d efficiently.
- But we can't compute the probability distribution P, so we pick a simpler distribution Q that we can compute.
- We will learn Q using a recognition model trained on the input patterns d.
- Train recognition and generative models in parallel.

Training the Generative Weights To Minimize the Free Energy



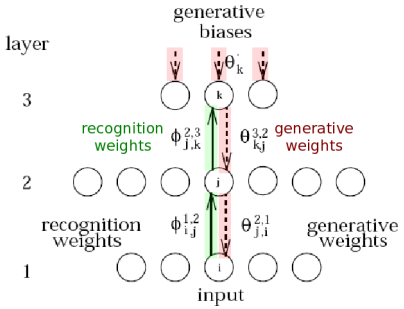
The Helmholtz Machine: Hinton, Dayan, Zemel, Frey, and Neal



$$net_k = \sum_j s_j w_{jk}$$

$$p(s_k=1) = \frac{1}{1+\exp(-net_k)}$$

Helmholtz Machine



Recognition weights ϕ are feed-forward. Given d, compute α.
Generative weights θ are feed-backward. Given α, generate d'.
Generative biases θ_k correspond to class priors.
Any number of layers allowed.
No recurrence: strictly feed-forward and feed-back.

Cost Function

Description length (cost) of unit j in state α:

$$C(s_j^\alpha) = -s_j^\alpha \log p_j^\alpha - (1-s_j^\alpha)\log(1-p_j^\alpha)$$

Description length for input vector d using the representation α is cost of describing hidden states plus cost of describing d given those hiddens.

$$C(\alpha, d) = C(\alpha) + C(d|\alpha)$$

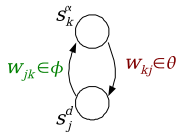
$$= \underbrace{\sum_j C(s_j^\alpha)}_{\text{From recognition model}} + \underbrace{\sum_i C(s_i^d|\alpha)}_{\text{Based on output of generative model given state } \alpha}$$

Training the Generative Model

- Start with input pattern d.
- Recognition model with weights ϕ generates an explanation α.
- Adjust generative model weights θ to minimize cost

$$C(\alpha, d) = \underbrace{C(\alpha)}_{\text{Explanation from recognition model}} + \underbrace{C(d|\alpha)}_{\text{Difference between d and output d' of generative model given } \alpha}$$

Learning to Reduce Generator Cost



$$\frac{\partial}{\partial w_{kj}} C(\alpha, d) = \frac{\partial}{\partial w_{kj}} \left[C(\alpha) + C(d|\alpha) \right]$$

Doesn't depend on w_{kj}

$$= \frac{\partial}{\partial w_{kj}} C(s_j^d|\alpha)$$

From recognition model

$$= \frac{\partial}{\partial w_{kj}} \left[-s_j^d \log p_j^\alpha - (1 - s_j^d) \log (1 - p_j^\alpha) \right]$$

$$p_j^\alpha = \sigma\left(-\sum_k s_k^\alpha w_{kj} \right) = \sigma(net_j^\alpha)$$

Interesting property of the sigmoid:

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

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

So $\frac{\partial}{\partial w_{kj}} \log p_j^\alpha = \frac{1}{p_j^\alpha} \cdot \frac{\partial}{\partial w_{kj}} p_j^\alpha$

$$= \frac{1}{p_j^\alpha} \cdot p_j^\alpha \cdot (1 - p_j^\alpha) \cdot \frac{\partial}{\partial w_{kj}} net_k^\alpha$$

$$= (1 - p_j^\alpha) \cdot s_k^\alpha$$

Learning Rule for Generative Weights

$$\frac{\partial}{\partial w_{kj}} C(\alpha, d) = \frac{\partial}{\partial w_{kj}} \left[-s_j^d \log p_j^\alpha - (1 - s_j^d) \log (1 - p_j^\alpha) \right]$$

$$= s_k^\alpha (s_j^d - p_j^\alpha)$$

So the weight update rule for the generative weights is:

$$\Delta w_{kj} = \epsilon \cdot s_k^\alpha (s_j^d - p_j^\alpha)$$

What About the Recognition Model?

- Given a fixed generative model, so that C(α,d) is known, what is the best way to select an α for a given d?
- Should use a Boltzmann distribution

$$P(\alpha|d) = \frac{\exp(-C(\alpha, d))}{\sum_{\beta} \exp(-C(\beta, d))}$$

because it minimizes the cost C(d):

$$C(d) = \underbrace{\sum_{\alpha} Q(\alpha|d)\, C(\alpha, d)}_{\text{Expected cost to transmit d}} - \underbrace{(-\sum_{\alpha} Q(\alpha|d) \log Q(\alpha|d))}_{\text{Entropy of the distribution Q}}$$

$Q(\alpha|d)$ = prob. of choosing explanation α to describe d

Where Does Q Come From?

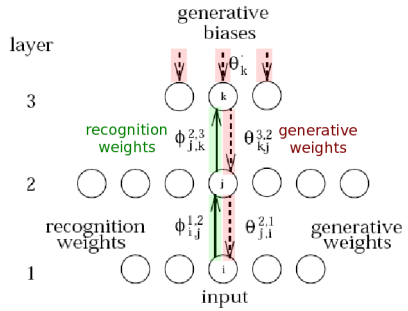
Ideally, $Q(\alpha|d) \equiv P(\alpha|d)$

But we can't afford to measure $P(\alpha|d)$.

So we can't train Q to match P. What can we do instead?

- Run the generative model to produce explanation γ.
- Train the recognition weights ϕ to minimize the cost of transmitting γ given d', the generated pattern.
- So we adjust Q to match the generator's distribution rather than the real distribution.

Training the Recognition Weights



Wake-Sleep Algorithm

- Wake phase: recognition weights determine explanation α. Train the generative weights:

$$\Delta w_{kj} = \epsilon\ s_k^{\alpha}(s_j^{\alpha} - p_j^{\alpha})$$

- Sleep phase: generative weights drive the network and produce a generated state γ. Train the recognition weights:

$$\Delta w_{jk} = \epsilon\ s_k^{\gamma}(s_k^{\gamma} - q_k^{\gamma})$$

where γ is the generated state and q_k^{γ} is the probability that unit k would be turned on by the recognition weights in layer j given s_j^{γ}.

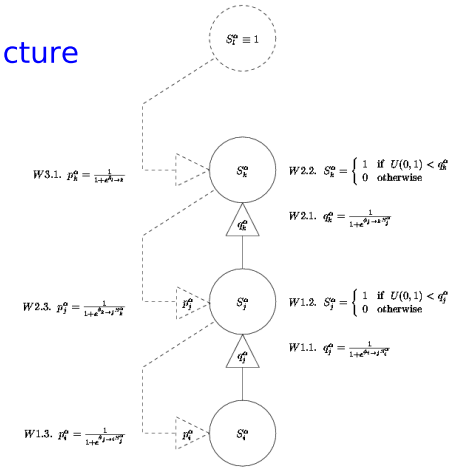
Four Sets of Numbers

α is generated in wake mode by supplying input pattern d
γ is generated in sleep mode by generating a pattern d'

q_j^α recognition probability, determines s_j^α from d
p_j^α generative probability for recognition state α
train generator weights w_{kj} using p_j^α

p_k^γ generative probability, determines s_k^γ
q_k^γ recognition probability for generative state γ
train recognizer weights w_{jk} using q_k^γ

Big Picture

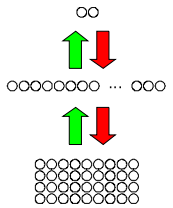


Independence Assumption

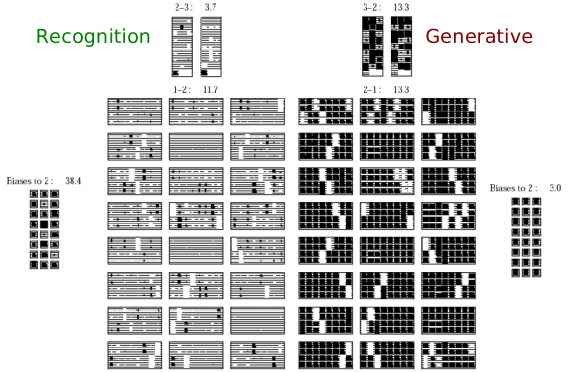
- We approximate P using Q to train the recognizer:
- We generate states γ and then use the recognition weights to estimate the probabilities q^γ.
- Problem: this assumes all the bits in each hidden layer are independent.
- In general, they're not.

Experiment: The Shifter Problem

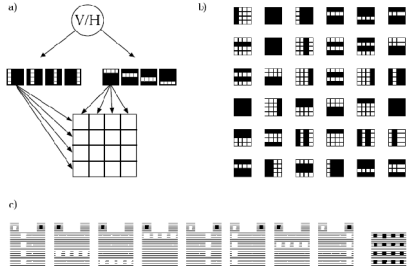
- Four rows of 8 inputs: each of two rows is copied twice.
- 24 hidden units
- 2 top level units: L and R



Learned Weights



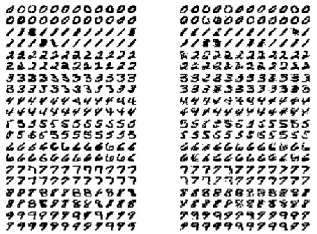
Vertical/Horizontal Bar Task



Every pixel appears in an equal number of vertical and horizontal bars.

Correct classification requires detecting co-occurrence of several pixels.

Digit Recognition Task



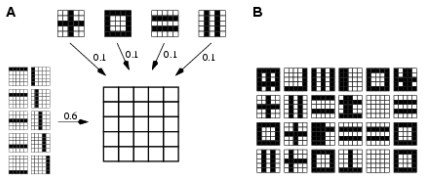
Sample Training Patterns | Generated Patterns

Error rates:

Nearest neighbor	6.7%
Backprop	5.6%
Helmholtz	4.8%

700 examples of each digit from MNIST dataset.
Separate network for each digit class.
Classification by measuring C(α,d) over 10 runs.

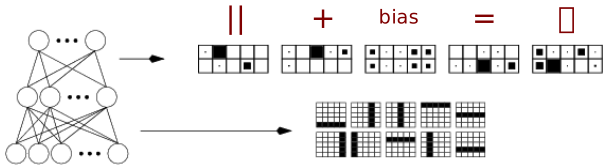
Lewicki & Sejnowski: Learning Higher Order Structure



Generate patterns that are mixtures of four higher order shapes (plus, squars, equals, and parallels) plus random single lines.

Can the network recover the underlying structure?

Results of Learning



Helmholtz Machine: Summary

- Helmholtz machines are stochastic, like Boltzmann.
- They do not use annealing, so they learn much faster.
- Rely on an approximation Q to the true generative distribution P, so they don't always work.
- Are part of a family of MDL (Minimum Description Length) models that try to optimize a cost function based on description of the data set.