Introduction to Neural Networks
U. Minn. Psy 5038

Probability, Energy, & the Boltzmann machine

Initialize

■ **Spell check off. Plots small.**

In[47]:=
```
Off[General::spell1];
SetOptions[Plot, ImageSize → Small];
SetOptions[ArrayPlot, ColorFunction → Hue, Mesh → True, ImageSize → Tiny];
```

Statistical physics, computation, and statistical inference

At the beginning of this course, we noted that John von Neumann, one of the principal minds behind the architecture of the modern digital computer, wrote that brain theory and theories of computation would eventually come to more resemble statistical mechanics or thermodynamics than formal logic. We have already seen in the Hopfield net, the development of the analogy between statistical physics systems and neural networks. In the past 15 years, the relationship between computation and statistical physics has received considerable study (cf. Hertz et al., 1991). We are going to look at a neural network model that exploits the relationship between thermodynamics and computation both to find global minima and to modify weights. Further, we will see how relating energy to probability leads naturally to statistical inference theory. Much of the current research in neural network theory is done in the context of statistical inference (Bishop, 1995; Ripley, 1996).

Probability Preliminaries

Random variables, discrete probabilities, probability densities, cumulative distributions

■ Discrete distributions: random variable X can take on a finite set of discrete values

X = {x(1),...,x(N)]

$$\sum_{i=1}^{N} p_i = \sum_{i=1}^{N} p(X = x(i)) = 1$$

■ Continuous densities: X takes on continuous values, x, in some range.

Density: p(x)

Analogous to material mass,we can think of the probability over some small domain of the random variable as "probability mass":

$$\text{prob}(x < X < \text{dx} + x) = \int_{X}^{\text{dX}+X} p(x)\, dx$$

$$\text{prob}(x < X < \text{dx} + x) \simeq p(x)\, \text{dx}$$

By analogy with discrete distributions, the area under $p(x)$ must be unity :

$$\int_{-\infty}^{\infty} p(x)\, dx = 1$$

like an object always weighing 1.

Cumulative distribution:

$$\text{prob}\,(X < x) = \int_{-\infty}^{x} p(X)\, dX$$

■ Densities of discrete random variables

The Dirac Delta function, $\delta[\bullet]$, allows us to use the mathematics of continuous distributions for discrete ones, by defining the density as:

$$\text{p[x]}=\sum_{i=1}^{N} p_i \delta[\text{x - x[i]}], \text{ where } \delta[\text{x - x[i]}] = \begin{cases} \infty & \text{for } x = x[i] \\ 0 & \text{for } x \neq x[i] \end{cases}$$

Think of the delta function, $\delta[\bullet]$, as ϵ wide and $1/\epsilon$ tall, and then let $\epsilon \to 0$, so that:

$$\int_{-\infty}^{\infty} \delta(y)\, dy = 1$$

The above density, p[x], is a series of spikes. It is infinitely high only at those points for which x = x[i], and zero elsewhere. But "infinity" is scaled so that the local mass or area around each point x[i], is p_i.

Check out *Mathematica*'s functions: DiracDelta, KroneckerDelta. What is the relationship of KroneckerDelta to IdentityMatrix?

■ Joint probabilities

Prob (X AND Y) = $p(X, Y)$
Joint density : $p(x, y)$

Probability and energy

■ Conditional probabilities

Two events, a and b, are said to be independent if the probability of their occurring together (i.e. their "joint probability") is equal to the product of their probabilities:

$$p(a\&b) = p(a)p(b)$$

By definition, the conditional probability of a given b (or "the conditional probability of a on b") is:

$$p(a|b) = \frac{p(a\&b)}{p(b)}$$

If a and b are independent, what is the conditional probability of a given b? The intuition is that knowledge of b provides no help with making statistical decisions about a.

■ Probabilities of hypotheses contingent on data, Bayes' rule

Conditional probabilities are central to modeling statistical decisions about hypotheses that depend on data. For example, the posterior probability of H, given data d is:

$$p(H|d)$$

It is called "posterior", because it is the probability *after* one knows the data. It is more constrained than the *prior* probability, p(H).

If one has a formula for the posterior probability, then it is possible to devise optimal strategies to achieve well-defined goals of inference. For example, imagine the data are given and thus fixed. A device that picks the hypothesis, H, that makes the posterior probability biggest is optimal in the sense that it makes the fewest errors on average. In this case, the goal is well-defined--to achieve the least average error rate. This kind of decision maker is called a *maximum a posteriori*

(or MAP) estimator.

Often it is easier to find a formula for p(d|H), then the other way around. In other words, we might have a good idea of the generative model--p(d|H) and p(H).

If the prior, p(H), is known, one can still do MAP estimation because of Bayes' rule:

$$p(H|d) = \frac{p(d|H)p(H)}{p(d)}$$

There are two assumptions that can simplify MAP estimation. First, p(d) is fixed at some constant value (we have the data and it isn't changing while we try to decide on the best hypothesis to explain the data). Further, we often don't have reason to prefer one hypothesis H=H', over any other, say H=H". So p(H) is constant. If these two conditions hold, then MAP estimation is equivalent to finding the H that makes p(d|H) biggest. This latter strategy is called *maximum likelihood* estimation.

■ Putting probability and energy together: The Gibbs distribution

Let $E(V_1,..,V_n)$ be an energy function for a network, as in a Hopfield model. Then we can write a probability function, called the Gibb's distribution, for the network as:

$$p(V_1,\ldots,V_n) = \kappa \exp\left(\frac{-E(V_1,\ldots,V_n)}{T}\right)$$

T is a parameter that controls the "peakedness" of the probability distribution (e.g. later we'll see that if Vs are continuous and the energy is a quadratic function of the V's, the Gibbs distribution becomes a multivariate Gaussian, and if each unit has the same variance, and they are independent, we have $T = 2\sigma^2$). From the physicist's point of view, T is temperature--e.g. the hotter the matter, the more variance there is in the particle velocities. For a magnetic material, an increase in thermal fluctuations makes it more likely for little atomic magnets to flip out of their otherwise regular arrangement. κ is a normalization constant determined by the constraint that the total probability over all possible states must equal one.

Question: So if the Hopfield net seeks states that minimize energy, what kind of statistical estimator is the Hopfield net?

Now imagine that some of the values of our units are given and fixed. In other words a subset of the units are declared to be the input, and are "clamped" at specific levels. Call these V_i^s. These values are fixed, and the others vary. The conditional probability is written as:

$$p(V_1,\ldots,V_m|V_1^s,\ldots,V_k^s) = \kappa' \exp\left(\frac{-E(V_1,\ldots,V_m;V_1^s,\ldots,V_k^s)}{T}\right)$$

So from a statistician's point of view, *a network that is evolving to minimize an energy function, is doing a particular form of Bayesian estimation, MAP*.

Sidenote on terminology

We've already noted that energy is equivalent to the Lyapunov function of dynamical systems. Other analogous terms you may run into are: Hamiltonian (in statistical mechanics), and cost or objective functions in optimization theory.

Boltzmann machine: Constraint satisfaction

Introduction

We've seen how local minima in an energy function can be useful stable points for storing memories. However, for constraint satisfaction problems such as the stereo example, local minima can be a real annoyance--one would like to find the *global* minimum, because this corresponds to the state-vector that should best satisfy the constraints inherent in the weights and the data input.

One of the early contributions to improving the odds of finding the global minimum was an algorithm called the Boltzmann machine by Ackley et al. (1985). Like the Hopfield network, the Boltzmann machine is a recurrent network with units connected to each other with symmetric weights. The units are binary threshold logic units. Unlike the 1982 Hopfield net, the Boltzmann machine uses a stochastic update rule that allows occasional increases in energy.

But more importantly, the Boltzmann machine had a feature that made it a more potentially powerful learning machine. Although fully connected, yet unlike Hopfield, some units could be "hidden" and others "visible". Similar to the back-prop nets, the hidden units could learn to capture statistical dependencies of higher order than two.

First we take a look at the update rule, and then the learning rule. The learning rule will lead us to a different view of supervised learning, in which the goal is to model the state of the environment--the joint probability of the various possible combinations of inputs.

Finding the global minimum: Theory

■ TLUs and energy again

The starting point is the discrete Hopfield net (1982), with a view towards solving constraint satisfaction, rather than memory problems. Energy is then a measure of the extent to which a possible combination of hypotheses violates the constraints of the problem. We've seen this with the stereo problem. Let V1 and V2 be the outputs of neural elements 1 and 2. These two outputs can be thought to represent *local* "hypotheses". A positive connection weight (e.g. T_{12}) means that local hypotheses, V1 and V2 support one another. A negative weight would mean that the two hypotheses inhibit each other--and should discourage both from being accepted at the same time.

Some of the inputs can be clamped, and the rest allowed to evolve. In this way the network finds the conditional local minimum (or equivalently, the maximum of the corresponding conditional probability).

$$V_i = \begin{cases} V_i^c & \text{if } i \text{ is a clamped unit} \\ 1 & \text{if } \Sigma\, T_{\text{ij}}\, V_j \geq U_i \\ 0 & \text{if } \Sigma\, T_{\text{ij}}\, V_j < U_i \end{cases}$$

$$E = -\sum_{i<j} T_{ij} V_i V_j + \sum_i U_i V_i$$

(Notation: the sum for i<j, is the same as the 1/2 the sum for i not equal to j, because the weight matrix is assumed to be symmetric, and the diagonals are not included. So this may look different than the Hopfield energy, but it isn't.).

As we have done several times before, we can remove the explicit dependence on the threshold U_i, by including weights $-U_i$, and a clamped input of 1 that effectively biases the unit. Then, as we saw for the Hopfield net:

$$E = -\sum_{i<j} T_{ij} V_i V_j$$

Consider the ith unit. If V_i goes from 1 to 0, the energy gap between two states corresponding to Vi being off (hypothesis i rejected) or on (hypothesis i accepted) is:

$$\Delta E_i = \sum_j T_{ij} V_j$$

■ Boltzmann update rule

Recall that under certain conditions, Hopfiled showed that energy can't increase.

To allow escapes from local minima, the idea is to allow occasional *increases* in energy, an idea that goes back to 1953 (Metropolis et al., 1953). Let's see how it works.

Set

$$V_i = 1 \text{ with probability } p_i$$

where

$$p_i = p(\Delta E_i) = \frac{1}{1 + e^{-\Delta E_i / T}}$$

T is a free parameter that plays the role of temperature in thermodynamics. When we implement the algorithm below, we draw a (uniformly distributed) number between 0 and 1 "out of a hat", and if that number is less than p_i, we set V_i to 1. Otherwise, set it to zero. Specifically, the update rule is:

$$V_i = \begin{cases} 1 & \text{if Random[Real]} < p(\Delta E_i) = p(\sum_j T_{ij} V_j) \\ 0 & \text{otherwise} \end{cases}$$

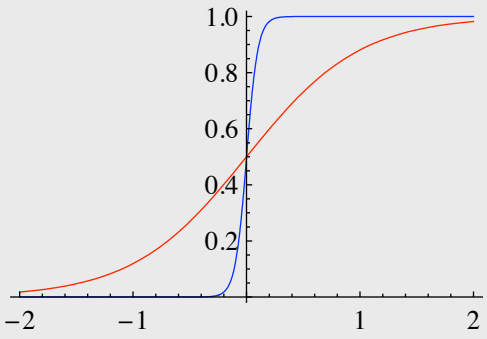
If the temperature is very low (T about zero), the update rule is the same as that for a deterministic TLU. This is because if the weighted sum of inputs is bigger than zero, p is virtually at 1. The probability of setting V to 1 is then guaranteed. If the weighted sum is less than zero, p is zero, and the probability of setting V to 1 is nil.

In[50]:=
```
boltz[x_,T_] := 1/(1 + Exp[-x/T]);
```

Here is a plot of p_i, with a high and low temperatures:

In[57]:=
```
Plot[Tooltip[{boltz[x, 0.5], boltz[x, 0.05]}],
 {x, -2, 2}, PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 0, 1]}]
```

Out[57]=



■ Simulated annealing

Now if we just let the Boltzmann rule update the state vector at a high temperature, the network will never settle to a stable point in state space. Conversely, if we set the temperature low, the network is likely to get stuck in a local minimum. The key idea behind introducing the notion of temperature is to start off with a high temperature, and then gradually cool the network. This simulates the physical process of annealing. If one heats metal, and then cools it rapidly, there is less crystalline structure or alignment of the atoms. This is a high energy state, and is desirable for making strong metal. The steel has been tempered. Slower cooling allows the substance to achieve a lower energy state with more alignment, with correspondingly more potential fractures. Although bad for metal strength, slow annealing is good for constraint satisfaction problems.

The "Gibbs Sampler" is the more general form of updating for n-valued nodes making up a Markov Random Field (Geman and Geman, 1984). It has been shown that a suitably slow annealing schedule will guarantee convergence (Geman and Geman, 1984):

$$T(n) > \frac{c}{\log(1+n)}$$

This annealing schedule, however, can be VERY slow, in fact too slow to be usually practical, except for small scale problems.

Local minimum demonstration

Let's look at a simple example where the standard discrete Hopfield net gets stuck in a local minimum, but the Boltzmann machine with annealing gets out of it. Let's construct in a 2D grid (similar to the stereo example). Each unit gets connected to its four nearest neighbors with weights given by **weight** (=1). As illustrated in an exercise below, the global minimum for this network is when all the units are turned off. But does the TLU update rule get us there from all initial states?

■ Initialization

We will use a toroidal geometry to keep the programming simple.

In[6]:=
```
Mod2[x_,n_] := Mod[x-1,n] + 1;
threshold[x_] := N[If[x>=0,1,-1]];
```

In[8]:=
```
weight = 1;
size = 16;
numiterations = 10;
```

Below, we will deliberately construct a weight matrix so that the energy function has a local minimum at the following state vector:

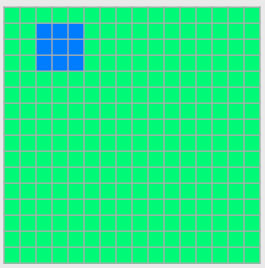
In[11]:=
```
V = Table[-1,{i,size},{j,size}];

V[[2,3]] = 1; V[[2,4]]= 1;  V[[2,5]] = 1;
V[[3,3]] = 1; V[[3,4]] = 1; V[[3,5]] = 1;
V[[4,3]] = 1; V[[4,4]] = 1; V[[4,5]] = 1;
```

Here is a picture of the state vector with the local minimum:

In[15]:=
```
ArrayPlot[V, PlotRange → {-5, 5}]
```

Out[15]=



Asynchronous updating without annealing: Getting stuck

Each unit is connected to its four nearest neighbors with weights given by **weight** (=1). The rest of the weights are zero. So the update rule is:

In[16]:=
```
update[Vv_,ii_,jj_] :=
            threshold[weight (Vv[[ Mod2[ii + 1,size],
             jj  ]] +
            Vv[[  Mod2[ii - 1,size], jj  ]] +
            Vv[[  ii, Mod2[jj - 1,size]   ]] +
            Vv[[  ii,  Mod2[jj + 1,size]   ]])];
```

In[17]:=
```
iter=1;
```

In[18]:=
```
updateAndPlot := Module[{}, For[i = 1, i ≤ size size, i++,
    iindex = RandomInteger[size - 1] + 1; jindex = RandomInteger[size - 1] + 1;
    V[[iindex, jindex]] = update[V, iindex, jindex];];
   Print[ArrayPlot[V, PlotRange → {-5, 5},
     Epilog → Inset[iter++, {size - 2, size - 2}]]]];
```

In[19]:=
```
Button["Update and Plot", updateAndPlot]
```

Out[19]=

Update and Plot

■ Asynchronous updating with annealing: Getting unstuck

In[20]:=
```
temp0 = 1;
iter = 1;
temp = temp0 / Log[1 + iter];
```

In[23]:=
```
updateAndPlotAnneal := Module[{},
   temp = temp0/Log[1 + iter];
   For[i = 1, i ≤ size size, i++,
    iindex = RandomInteger[size - 1] + 1; jindex = RandomInteger[size - 1] + 1;
    pdeltaE = N[boltz[weight (V[[Mod2[iindex + 1, size], jindex]] +
          V[[Mod2[iindex - 1, size], jindex]] + V[[iindex, Mod2[jindex - 1,
            size]]] + V[[iindex, Mod2[jindex + 1, size]]]), temp]];
    V[[iindex, jindex]] = If[pdeltaE ≥ RandomReal[], 1, -1];];

   Print[ArrayPlot[V, PlotRange → {-5, 5},
     Epilog → Inset[iter++, {size - 2, size - 2}]]]];
```

In[24]:=
```
Button["Update with Annealing and Plot", updateAndPlotAnneal]
```

Out[24]=

Update with Annealing and Plot

Optional exercise 1: Energy function

Write a function to calculate the energy function for the above network.

What is the energy of the ground state?

What is the energy of the local minimum we constructed above?

Optional exercise 2 : Thermal equilbrium

Make two versions of the above simulation in which 1) the temperature is fixed; 2) the temperature is gradually lowered (as above). Start with a random initial setting of the network.

Pattern synthesis using Markov Random Field models & Gibbs sampling

Let's see how the ideas in the Boltzmann machine can be applied to problems of pattern synthesis. This involves a different perspective on the same network--now instead of viewing it as doing inference, the network is a generative model. Let's let the units take on continuous values. Consider the quantized case. Suppose rather than just binary values, our units can take on a range of values. We'll introduce the idea with a demonstration of a pattern synthesizer for texture generation.

■ Local energy

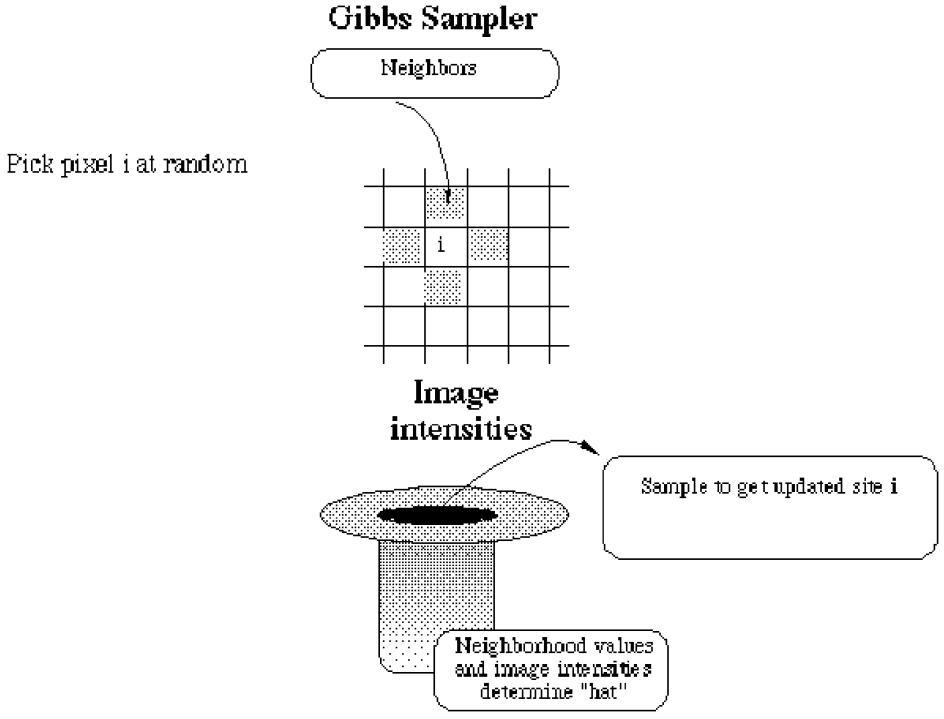
$$\text{Local energy (potential) at location } i = \sum_{j \in N(i)} f\left(V_i - V_j\right) \tag{2}$$

The local energy determines a local conditional probability for the values at the ith site:

$$p\left(V_i \mid V_j,\ j \in N_i\right) = \kappa\, e^{-\sum_{j \in N(i)} f\left(V_i - V_j\right)}$$

■ Sampling from textures using local updates

To draw a sample at the ith node, we draw from the local (conditional) probability distribution:



The Gibbs Sampler

■ Set up image arrays and useful functions

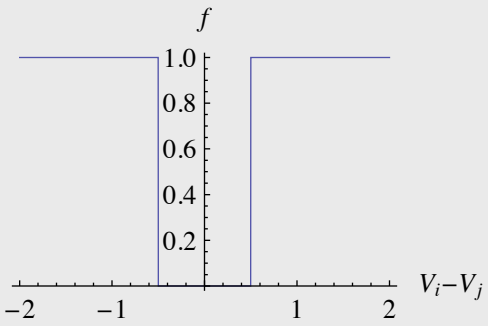
In[25]:=
```
size = 32; T0 = 1.; ngray = 16;
brown = N[Table[RandomInteger[{1, ngray}], {i, 1, size}, {i, 1, size}]];
next[x_] := Mod[x, size] + 1;
previous[x_] := Mod[x - 2, size] + 1;
 Plus @@ Flatten[brown]
----------------------;
Length[Flatten[brown]]
```

We can try several types of potentials.

■ Ising potential

In[30]:=
```
Clear[f];
f[x_, s_, n_] := If[Abs[x] < 0.5, 0, 1];
s0 = 1.`; n0 = 5;
Plot[f[x, s0, n0], {x, -2, 2}, AxesLabel →
  {"\!\(\*SubscriptBox[\(V\), \(i\)]\)-\!\(\*SubscriptBox[\(V\),
     \(j\)]\)", f}]
```

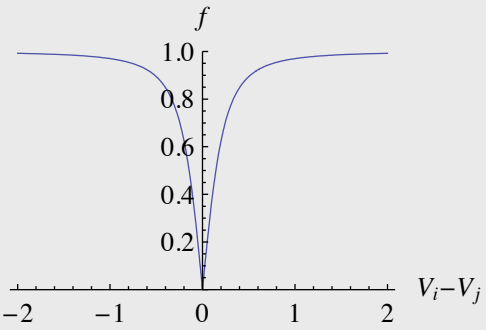
Out[33]=



■ Geman & Geman potential

In[34]:=
```
Clear[f];
f[x_, s_, n_] := N[Sqrt[(Abs[x/s]^n) / (1 + Abs[x/s]^n)]];
s0 = 0.25`; n0 = 2;
Plot[f[x, s0, n0], {x, -2, 2}, PlotRange → {0, 1},
 AxesLabel → {"\!\(\*SubscriptBox[\(V\),
     \(i\)]\)-\!\(\*SubscriptBox[\(V\), \(j\)]\)", f}]
```

Out[37]=



■ Define the potential function using nearest-neighbor pair-wise "cliques"

Suppose we are at site i. **x** is the activity level (or graylevel in a texture) of unit (or site) i. **avg** is a list of the levels at the neighbors of the site i.

In[38]:=
```
Clear[gibbspotential, gibbsdraw, tr];
gibbspotential[x_, avg_, T_] :=
  N[Exp[-(f[x - avg[[1]], s0, n0] + f[x - avg[[2]], s0, n0] +
        f[x - avg[[3]], s0, n0] + f[x - avg[[4]], s0, n0]) / T]];
```

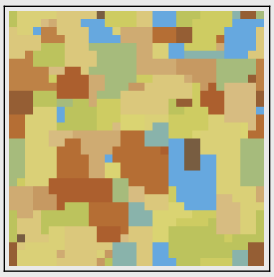
■ Define a function to draw a single pixel gray-level sample from a conditional distribution determined by pixels in neighborhood

The idea is to calculate the cumulative distribution corresponding to the local conditional probability, pick a uniformly distributed number, which determines the value of the sample x (through the distribution). To save time, we avoid having to normalize the cumulative (it should asymptote to 1) by drawing a uniformly distributed random number between the max and min values of the output of FoldList (the cumulative sum).

In[40]:=
```
gibbsdraw[avg_, T_] :=
  Module[{}, temp = Table[gibbspotential[x + 1, avg, T], {x, 0, ngray - 1}];
   temp2 = FoldList[Plus, temp[[1]], temp];
   temp10 = Table[{temp2[[i]], i - 1}, {i, 1, Dimensions[temp2][[1]]}];
   tr = Interpolation[temp10, InterpolationOrder → 0]; maxtemp = Max[temp2];
   mintemp = Min[temp2]; ri = RandomReal[{mintemp, maxtemp}];
   x = Floor[tr[ri]]; Return[{x, temp2}];];
```

In[41]:=
```
gg = Dynamic[ArrayPlot[brown, Mesh → False,
   ColorFunction → ColorData["SouthwestColors"], PlotRange → {1, ngray}]]
```

Out[41]=



In[42]:=
```
For[iter = 1, iter ≤ 10, iter++, T = 0.25;
  For[j1 = 1, j1 ≤ size size, j1++, {i, j} = RandomInteger[{1, size}, 2];
   avg = {brown[[next[i], j]], brown[[i, next[j]]], brown[[i, previous[j]]],
     brown[[previous[i], j]]}; brown[[i, j]] = gibbsdraw[avg, T][[1]];]; gg];
```

■ "Drawing" a pattern sample

Was it a true sample? Drawing true samples means that we have to allow sufficient iterations so that we end up with images whose frequency corresponds to the model. How long is long enough?

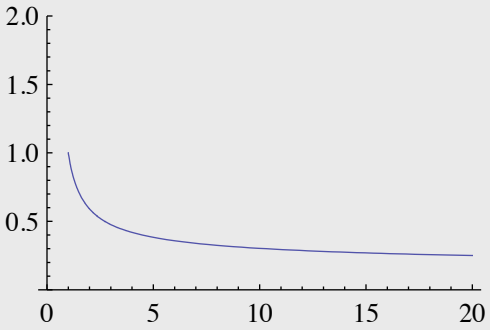
Finding modes

■ Define annealing schedule

In[43]:=
```
anneal[iter_, T0_, a_] := T0 / (a (1/a + Log[iter]));

Plot[anneal[iter, T0, 1], {iter, 1, 20},
 PlotRange → {0, 2}, AxesOrigin → {0, 0}]
```

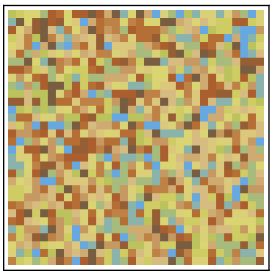
Out[44]=



■ Producing a texture sample with annealing

In the previous simulation, the temperature (variance) was fixed. We can also use annealing, to try to draw a sample near a mode.

In[45]:=
```
brown = N[Table[RandomInteger[{1, ngray}], {i, 1, size}, {i, 1, size}]];
For[iter = 1, iter ≤ 10, iter++, T = anneal[iter, T0, 1];
  For[j1 = 1, j1 ≤ size size, j1++, {i, j} = RandomInteger[{1, size}, 2];
   avg = {brown[[next[i], j]], brown[[i, next[j]]], brown[[i, previous[j]]],
     brown[[previous[i], j]]}; brown[[i, j]] = gibbsdraw[avg, T][[1]];];
  Print[ArrayPlot[brown, Mesh → False, ColorFunction →
     ColorData["SouthwestColors"], PlotRange → {1, ngray}]]];
```



Boltzmann Machine: Learning

We've seen how a stochastic update rule improves the chances of a network evolving to a global minimum. Now let's see how learning weights can be formulated as a statistical problem.

■ The Gibbs distribution again

Suppose T is fixed at some value, say T=1. Then we could update the network and let it settle to thermal equilibrium, a state characterized by some statistical stability, but with occasional jiggles. Let V_α represent the vector of neural activities. The probability of a particular state α is given by:

$$p(V_\alpha) = \kappa e^{-E_\alpha / T}$$

$$\kappa = \frac{1}{\sum_{\text{all states } k} e^{-E_k / T}}$$

Recall that the second equation is the normalization constant the ensures that the total probability (i.e. over all states) is 1.

We divide up the units into two classes: **hidden** and **visible** units. Values of the visible units are determined by the environment. If the visible units are divided up into "stimulus" and "response" units, then the network should learn associations. If they are just stimulus units ,then the network just observes and organizes its interpretation of the stimuli that arrive.

Our goal is to have the hidden units discover the structure of the environment. Once learned, if the network was left to run freely, the visible units would take on values that reflect the structure of the environment they learned. In other words, the network has a generative model of the visible structure.

Consider two probabilities over the visible units:

P - probability of visible units taking on certain values determined by the environment.

P' - probability that the visible units take on certain values while the network is running at thermal equilibrium.

If the hidden units have actually "discovered" the structure of the environment, then the probability P should match P'. How can one achieve this goal? Recall that for the Widrow-Hoff and error backpropagation rules, we started from the constraint that the network should minimize the error between the network's prediction of the output, and the actual target values supplied during training. We need some measure of the discrepancy between the desired and target states for the Boltzmann machine. The idea is to construct a measure of how far away two probability distributions are from each other--how far P is from P'. One such function is the Kullback-Leibler (KL) measure or relative entropy (also known as the "Gibbs G measure").

$$G\left(T_{12}, T_{13} \ldots, T_{ij}, \ldots\right) = \sum_{\substack{\text{all states} \\ \text{over visible units}}} P(V_\alpha) \log\left(\frac{P(V_\alpha)}{P'(V_\alpha)}\right)$$

Note: ΔT_{ij} refers to the weights and T to the temperature. Completely different.

Then we need a rule to adjust the weights so as to bring P' -> P in the sense of reducing the KL measure G. Ackley et al. derived the following rule for updating the weights so as to bring the probabilities closer together. Make weight changes ΔT_{ij} such that:

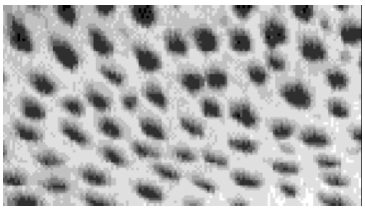
$$\Delta T_{ij} = \varepsilon\left(p_{ij} - p'_{ij}\right)$$

where p_{ij} is the probability of V_i and V_j both being 1 when environment is clamping the states at thermal equilibrium averaged over many samples. p'_{ij} is the probability of V_i and V_j being 1 when the network is running freely without the environment at equilibrium.

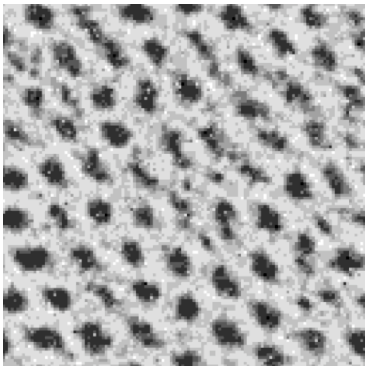
Descendants of Boltzmann machines

As noted above, convergence through simulated annealing can be impractically slow. The mean field approximation is one technique used to improve convergence (cf. Ripley, 1996). Boltzmann machines can be considered a special case of belief networks which we will study later (Ripley, 1996). Learning, as you might imagine, is also very slow because of the need to collect lots of averages before doing a weight update.

The Boltzmann machine learns to approximate the joint probability distribution on a set of binary random variables. Some of the variables are designated inputs, and others outputs. Learning large scale joint distributions is known to be a hard problem in statistics, and the success of the Boltzmann machine has been limited to small scale problems. One successor to the Boltzmann machine is the Helmholtz machine and its derivatives (Dayan et al., 1995; Hinton, 1997). A recent advance in learning pattern distributions is the Minimax theory (Zhu and Mumford, 1997). Here is an observed sample:

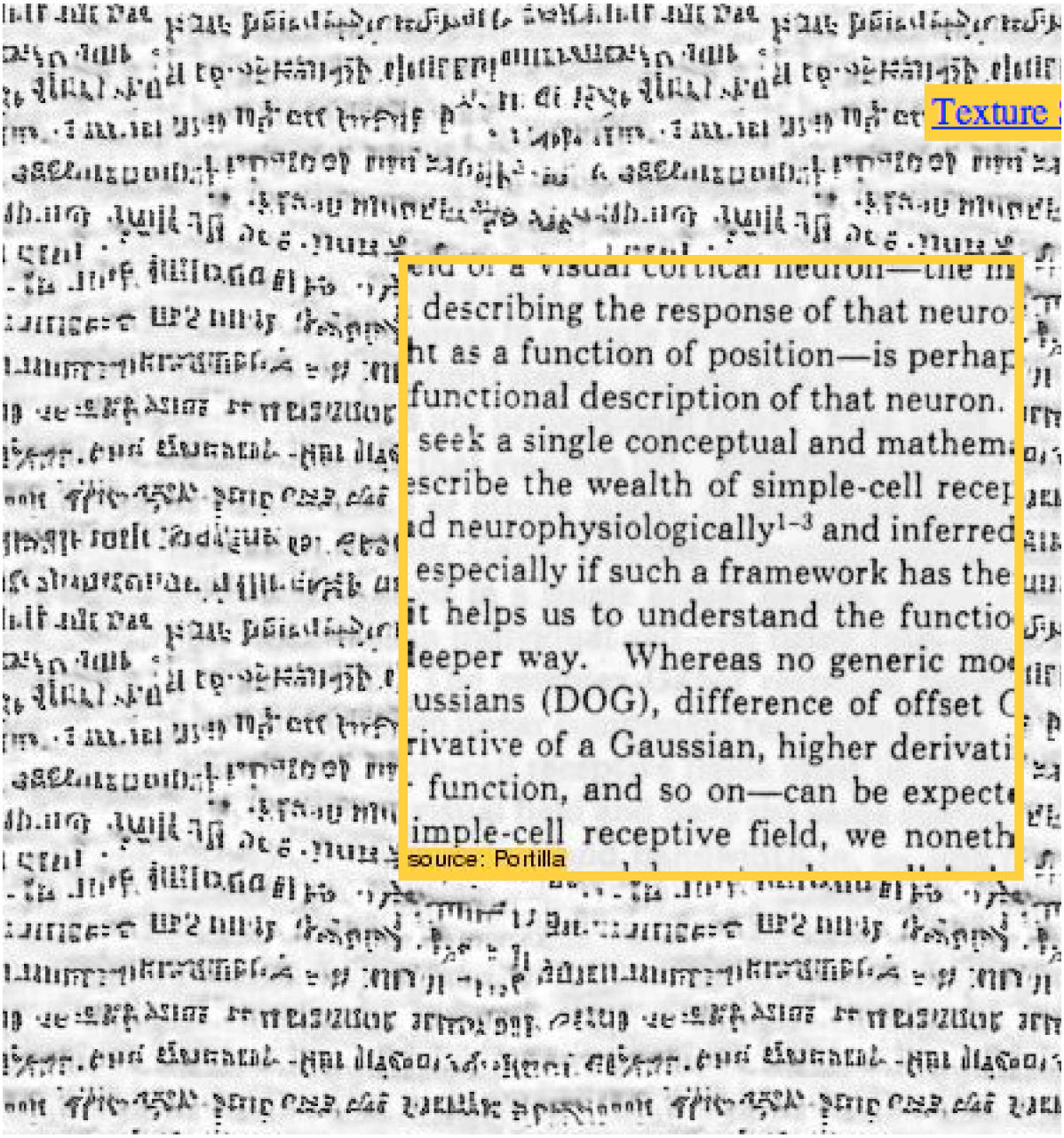


Here is a synthesized sample after Minimax entropy learning:



Below is an a sample of a "text" texture from http://www.cns.nyu.edu/~lcv/texture/

```
http : // www.cns.nyu.edu / ~lcv / texture / photo - periodic / text.o.jpg
```



In general, it is a hard problem to draw true samples from high dimensional spaces. And in the above text example, the sample was not drawn from known underlying probability distribution.

References

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. Cognitive Science, 9, 147-169.

Besag, J. (1972). Spatial interaction and the statistical analysis of lattice systems. {\it Journal of the Royal Statistical Society B}, {\bf 34}, 75-83.

Clark, James J. & Yuille, Alan L. (1990) Data fusion for sensory information processing systems. Kluwer Academic Press, Norwell, Massachusetts.

Cross, G. C., & Jain, A. K. (1983). Markov Random Field Texture Models. IEEE Trans. Pattern Anal. Mach. Intel., 5, 25-39.

Geiger, D., & Girosi. (1991). Parallel and Deterministic Algorithms from MRF's: Surface Reconstruction. I.E.E.E PAMI, 13(5).

D. Geiger, H-K. Pao, and N. Rubin (1998). Organization of Multiple Illusory Surfaces. Proc. of the IEEE Comp. Vision and Pattern Recognition, Santa Barbara.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. I.E.E.E. Transactions Pattern Analysis and Machine Intelligence, PAMI-6, 721-741.

Dayan, P., Hinton, G. E., Neal, R. M., & Zemel, R. S. (1995). The Helmholtz Machine. Neural Computation, 7(5), 889-904.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). Introduction to the theory of neural computation (Santa Fe Institute Studies in the Sciences of Complexity ed.). Reading, MA: Addison-Wesley Publishing Company.

Hinton, G. E., & Ghahramani, Z. (1997). Generative models for discovering sparse distributed representations. The Philosophical Transactions of the Royal Society, 352(1358), 1177-1190.

Kersten, D. (1990). Statistical limits to image understanding. In C. Blakemore (Ed.), Vision: Coding and Efficiency, (pp. 32-44). Cambridge, UK: Cambridge University Press.

Kersten, D. (1991) Transparency and the cooperative computation of scene attributes. In {Computation Models of Visual Processing}, Landy M., & Movshon, A. (Eds.), M.I.T. Press, Cambridge, Massachusetts.

Kersten, D., & Madarasmi, S. (1995). The Visual Perception of Surfaces, their Properties, and Relationships. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 19, 373-389.

Knill, D. C., & Kersten, D. K. (1991). Ideal Perceptual Observers for Computation, Psychophysics, and Neural Networks. In R. J. Watt (Ed.), Pattern Recognition by Man and Machine, (Vol. 14,): MacMillan Press.

Knill, D. C., Kersten, D., & Yuille, A. (1996). A Bayesian Formulation of Visual Perception. In K. D.C. & R. W. (Eds.), Perception as Bayesian Inference, (pp. Chap. 1): Cambridge University Press.

Lee, T. S., Mumford, D., & Yuille, A. Texture Segmentation by Minimizing Vector-Valued Energy Functionals: The Coupled-Membrane Model.: Harvard Robotics Laboratory, Division of Applied Sciences, Harvard University.

Madarasmi, S., Kersten, D., & Pong, T.-C. (1993). The computation of stereo disparity for transparent and for opaque surfaces. In C. L. Giles & S. J. Hanson & J. D. Cowan (Eds.), Advances in Neural Information Processing Systems 5. San

Mateo, CA: Morgan Kaufmann Publishers.

Marroquin, J. L. (1985). Probabilistic solution of inverse problems. M. I. T. A.I. Technical Report 860.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. J. Phys. Chem, 21, 1087.

Mumford, D., & Shah, J. (1985). Boundary detection by minimizing functionals. Proc. IEEE Conf. on Comp. Vis. and Patt. Recog., 22-26.

Poggio, T., Gamble, E. B., \& Little, J. J. (1988). Parallel integration of vision modules. {\it Science}, {\bf 242}, 436-440.

Ripley, B. D. (1996). Pattern Recognition and Neural Networks . Cambridge, UK: Cambridge University Press.

Zhu, S. C., Wu, Y., & Mumford, D. (1997). Minimax Entropy Principle and Its Applications to Texture Modeling. Neural Computation, 9(8), 1627-1660.

Zhu, S. C., & Mumford, D. (1997). Prior Learning and Gibbs Reaction-Diffusion. IEEE Trans. on PAMI, 19(11).