

Introduction to Neural Networks

More sampling

Overview

Last time

Scientific python; Python, IPython, numpy, matplotlib, scipy.

Today

More sampling, leading to MCMC applications

Next time: MCMC sampling with the Python module pymc

Why sampling?

Boltzmann machine's update rule review

Recall the Boltzmann machine was a fully connected network consisting of binary units. The update rule was to set

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

where p_i is:

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

where

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

T (without the indices) is analogous to temperature in thermodynamics. For $T=1$, this update rule should by now be quite familiar. If we add a bias term, replace V_j and T_{ij} with vector x and matrix w , it is the same expression that we saw logistic regression, and for the simple probabilistic model of neural spike generation (Lecture 18) where the probability of a neuron firing is:

$$p(y_i = 1 | x) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

To implement the update rule for the Boltzmann machine, we draw a uniformly distributed number between 0 and 1, and if that number is less than $p_i = p(\Delta E_i)$ we set the output of the neuron, V_i , to 1; otherwise, set it to zero:

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

This is an example of the use of Monte Carlo sampling to do inference. But sampling can be applied in many more ways, for example to learn the weights given training data, and more ways described below.

Earlier we also saw how to sample in one dimension. But sampling gets much harder as the dimensionality increases. To further motivate sampling, consider several applications:

Pattern synthesis

Draw samples from a generative model which might be complex. E.g. involving lots of lateral interactions as in an undirected graph. Or involving a top-down directed graph that specifies how causes create samples.

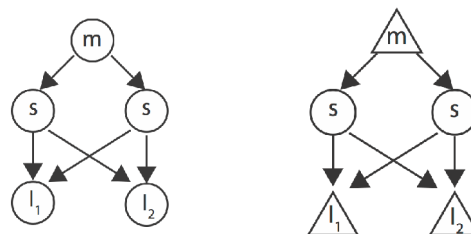
This can be quite useful in vision to check whether one's model is capturing the regularities seen in the data. Texture synthesis is one example.

Learning

Given a model, one uses stochastically generated (synthesized) samples in order to compare with data so as to adjust parameters (i.e. weights). We saw this in the original Boltzmann machine.

Optimization

We already saw an example using the Boltzmann machine to do optimization, where we had multiple constraints. For non-convex distributions, one may need annealing. Even with low-dimensional architectures (graphical models), the math can be too hard, and simulations are required. For example, it is one thing to posit and write down the joint distribution corresponding to this graph:



but quite another to calculate needed values for unknown variables, with others fixed, and still others unwanted. Recall that unwanted variables need to be integrated out in order to calculate ideal performance. Practical solutions to solving these kinds of problems is what we are aiming for here and in the later material on MCMC sampling.

Integration (summing over) in very high dimensions

Integration or summing over many variables is a recurring problem. Such as finding the volume of a function p :

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} p(x_1, x_2, x_3)$$

Often there is no formula for p or closed-form solution for its volume.

Or suppose you want to calculate the average value of a function, $f(x)$ of a random vector. (When you calculate the mean, $f(x) = x$.) and you have an expression for the joint distribution, say $p(x)$, where $x = (x_1, x_2, x_3, \dots, x_N)$.

$$E[f(x)] = \sum_{x_1} \sum_{x_2} \sum_{x_3} \dots \sum_{x_N} f(x_1, x_2, x_3, \dots, x_N) p(x_1, x_2, x_3, \dots, x_N)$$

Even if one had formulas for the functions, N can be very, very big.

Summing over gets computationally expensive the more variables we need to sum over. If we have $p(x_1, x_2, x_3, \dots, x_N)$ and we would like to sum over N variables, each of which can take on M values. We have to calculate and sum M^N values. For example, this would be impossible for even a small image.

Also, if we want to compare the probability of two models given some data, we need to integrate out the parameters, θ , that determine the models, m . For example, one model might be a quadratic function with three parameters, and another the sum of three sinusoids, whose parameters are the coefficients. Think of the parameters as values to be estimated (or explanations of the data) under model m . Bayes rule says that given data x :

$$p(\theta | x, m) = \frac{p(x | \theta, m) p(\theta | m)}{p(x | m)}$$

The denominator is called the “evidence”. For a fixed model, we usually don’t need to know the denominator. But if we want to decide which model is better we need to evaluate $p(x|m)$ for the various m ’s. And this may not be easy:

$$p(x | m) = \sum_{\theta} p(x, \theta | m) = \sum_{\theta} p(x | \theta, m) p(\theta | m)$$

And $\theta = (\theta_1, \theta_2, \dots)$ may have many dimensions. We’ll talk about “model selection” later.

Sampling as a metaphor for bottom-up, top-down neural processing

One can either fix causes and draw samples of the data (top-down), or fix the data and draw samples of possible causes.

Samples as the basis for representations and decisions.

See Sundareswara & Schrater (2008), Battaglia et al. (2011); Vul et al. (2012).

Sampling

What is Monte Carlo?

Let's consider the Monte Carlo approach to the integration problem. For example, given a line that divides a table into two unequal parts, one could estimate the relative proportions of one of the areas by randomly throwing lots of balls on to the table and recording the proportion that settled in the area of interest.

More generally, we can approximate a density $p(x)$ with a weighted set of samples from a potentially very high dimensional space by:

$$p_N(\mathbf{x}) \sim \sum_{i=1}^N \delta(\mathbf{x}^i - \mathbf{x}) p(\mathbf{x}^i)$$

and using the definition of the delta function, one can show that we can approximate high-dimensional integrals with computable sums:

$$I_N(f) = \frac{1}{N} \sum_{i=1}^N f(x^{(i)}) \xrightarrow[N \rightarrow \infty]{a.s.} I(f) = \int_{\mathcal{X}} f(x)p(x) dx$$

Let's first look at a univariate example to illustrate concepts that can be carried to more complicated cases. Of course to draw samples, we could use the inverse CDF method in an earlier lecture, but let's try something different.

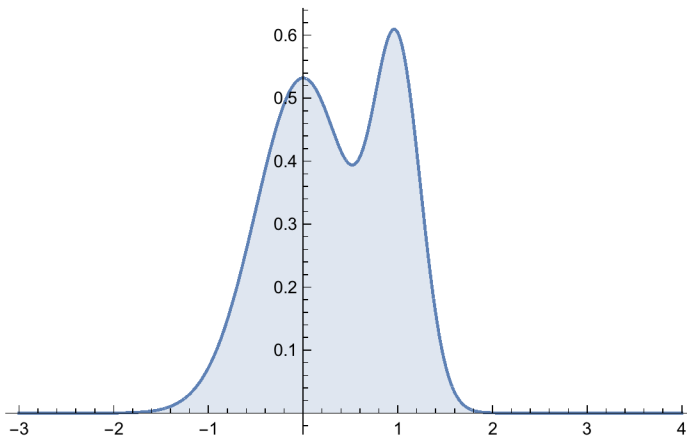
We want to draw samples from a target distribution that is hard to sample from. But we'll assume a potential sample can be drawn from another distribution that is easy to sample from, called the proposal distribution. Given sample from the proposal distribution, we can calculate its probability from the target distribution and then keep the sample if it is a true draw, but reject it if not. How do we know if it is a true draw?

Suppose we want to draw samples from this target distribution:

```

pD = MixtureDistribution[{2, 1},
  {NormalDistribution[0, 1/2], NormalDistribution[1, 1/4]};
Plot[PDF[pD, x], {x, -3, 4}, Filling -> Axis]

```



but we are only allowed to draw samples from this one:

```

qD = NormalDistribution[.5, 1]
NormalDistribution[0.5, 1]

```

How can we do it? One straightforward way is rejection sampling.

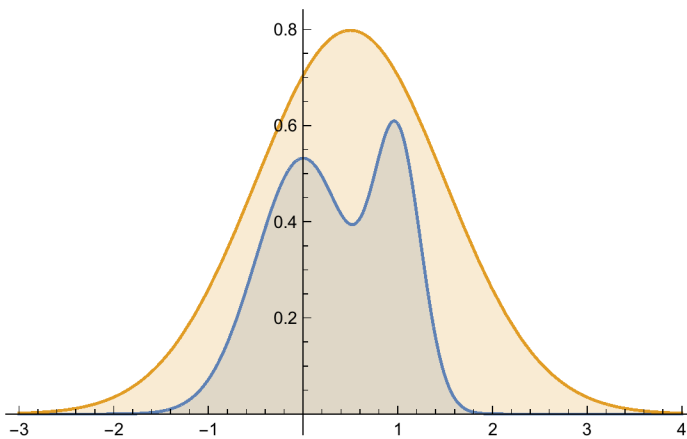
Rejection sampling

The idea is to first find M such that $\text{PDF}[pD, x] < M * \text{PDF}[qD, x]$.

```

Plot[{PDF[pD, x], 2 * PDF[qD, x]}, {x, -3, 4}, Filling -> Axis]

```



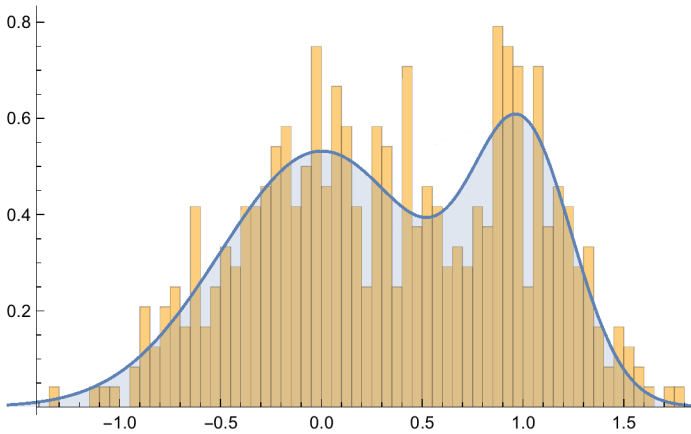
Draw a sample x' from qD and only keep it if $\text{PDF}[pD, x'] < M * \text{PDF}[qD, x']$.

```

u := RandomVariate[UniformDistribution[]];
x := RandomVariate[qD];
samples =
  Table[If[(u < PDF[pD, x0 = x] / (20 * PDF[qD, x0])) && Abs[x] < 3, x0, Null], {10 000}];
gH = Histogram[samples, 48, "PDF"];

```

```
gpD = Plot[{PDF[pD, x]}, {x, -3, 4}, Filling -> Axis];
Show[{gH, gpD}]
```



You can see that the rejection sampling is most efficient if M times the proposal distribution isn't too much higher than the target distribution over most of the range.

Rejection sampling becomes increasingly inefficient as the dimensionality increases when too many samples get rejected.

Importance "sampling"

Perhaps should be called "importance approximation", because we don't use it to directly produce samples from a target distribution, but use samples from a proposal distribution to estimate expectations with respect to the target distribution.

$$E[f] = \int f(x) p(x) dx = \int f(x) \frac{p(x)}{q(x)} q(x) dx = \int f(x) w(x) q(x) dx$$

We can approximate the density $p(x)$ in terms of a weighted sum of point masses drawn from $q(x)$:

$$p_N(x) \sim \sum_{i=1}^N \delta(x^i - x) w(x^i)$$

So if we sample from $q(x)$, but weight the sample according to $w(x)$, we can estimate

$$E[f] \sim \int \sum_i^N f(x^i) w(x^i) \delta(x^i - x) dx \sim \sum_i^N f(x^i) w(x^i)$$

Lets look at an example where we find the average of a random variable from target distribution pD , but we don't sample from it directly--as before, we sample from the "proposal" distribution, call it qD .

```
In[500]:= qD = NormalDistribution[0, 3];
pD = NormalDistribution[3, 1];
```

```
In[502]:= u := RandomVariate[qD];
weights = Table[{x0 = u, PDF[pD, x0] / PDF[qD, x0]}, {1000}];
Mean[weights[[;;, 1]] * weights[[;;, 2]]]
```

```
Out[504]= 2.95723
```

Compare with the mean when we are allowed to directly sample from $p_{\mathcal{D}}$:

```
In[505]:= x := RandomVariate[pD];
xsamples = Table[{x}, {1000}];
```

```
In[507]:= Mean[xsamples]
```

```
Out[507]= {2.97845}
```

Show that we don't need to know the normalizing constants for either the target or proposal distributions.

What is the importance of importance sampling?

Suppose you need to sample from a truncated normal distribution with support from 0 to 1--call it $p_T(x)$. You could draw samples from the normal and throw out ones outside of the (0,1) range. This could waste many samples.

Alternatively, you could use the uniform distribution over (0,1) as the proposal distribution, and weight each sample by $p_T(x^i)/1$. You can show that the mean is:

$$(1 / N) \sum_i^N x^i p_T(x^i)$$

And every sample gets used.

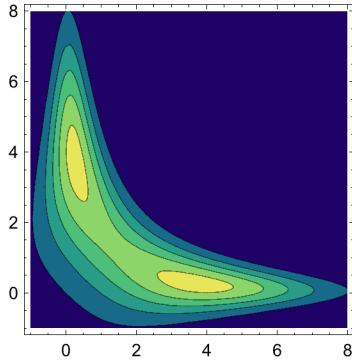
Estimate the variance of the target distribution of the above truncated normal using importance sampling

Gibbs sampling

Sometimes we can calculate the conditionals

Gibbs sampling simple bivariate demo

```
weird2ddist[x_, y_] := Exp[-(x * x * y * y + x * x * y * y - 8 * x - 8 * y) / 2];
gcontour = ContourPlot[weird2ddist[x, y]^2, {x, -1, 8},
  {y, -1, 8}, ColorFunction -> "BlueGreenYellow", ImageSize -> Small]
```



```

n = 50 000;
x = ConstantArray[0.0, n];
y = ConstantArray[0.0, n];

x[[1]] = 1.;
y[[1]] = 6.;
sig[z_, i_] := Sqrt[1. / (1. + z[[i]] * z[[i]])];
mu[z_, i_] := 4. / (1. + z[[i]] * z[[i]]);

For[i = 2, i < n - 1, i = i + 2,
  sigx = sig[y, i - 1];
  mux = mu[y, i - 1];

  x[[i]] = RandomVariate[NormalDistribution[mux, sigx]];
  y[[i]] = y[[i - 1]];

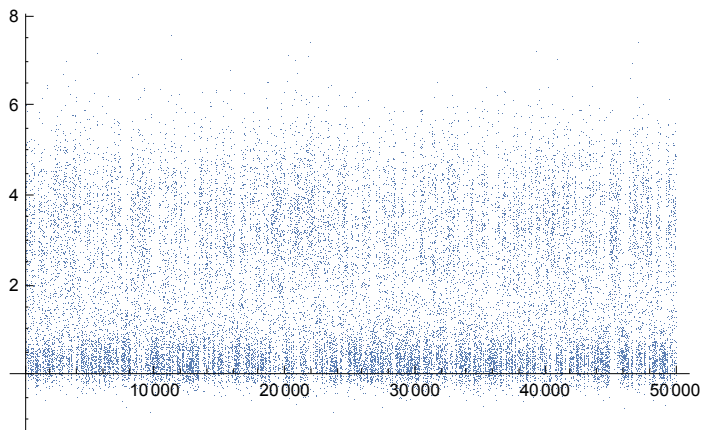
  sigy = sig[x, i];
  muy = mu[x, i];

  y[[i + 1]] = RandomVariate[NormalDistribution[muy, sigy]];
  x[[i + 1]] = x[[i]]; (**)

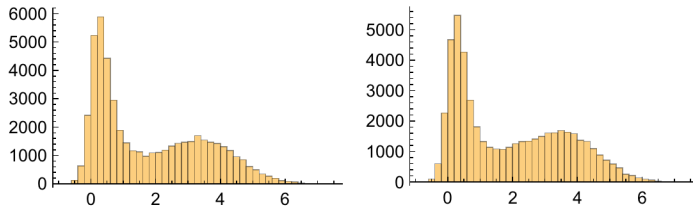
]

ListPlot[x]

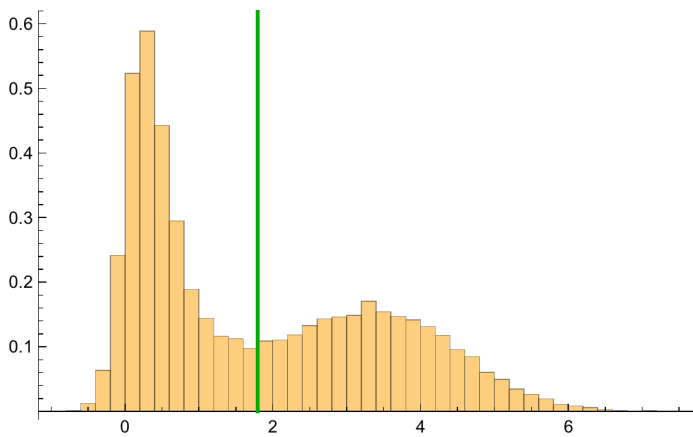
```




```
GraphicsRow[{Histogram[x, 50], Histogram[y, 50]}]
```



```
{bins, heights} = HistogramList[x, 50, "PDF"];
maxFreq = N[Max[heights]]
Histogram[x, {bins}, heights &,
  Epilog -> {{Thick, Darker[Green], Line[{{Mean[x], 0}, {Mean[x], maxFreq + 2}}]}}]
0.5886
```

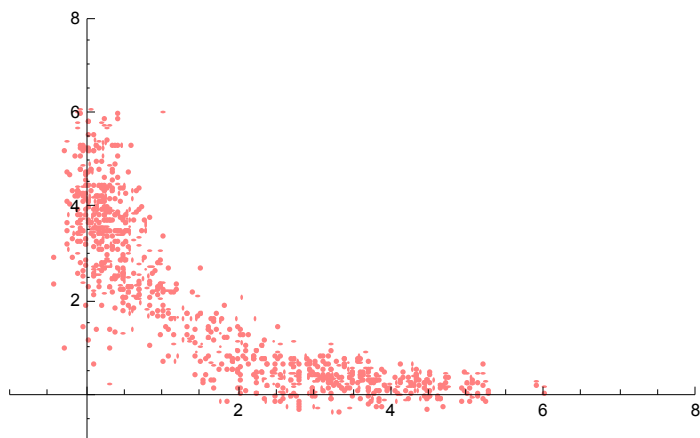


```
{Mean[x], Mean[y]}
```

```
{1.79745, 1.92216}
```

```
Dimensions[x][[2]]
```

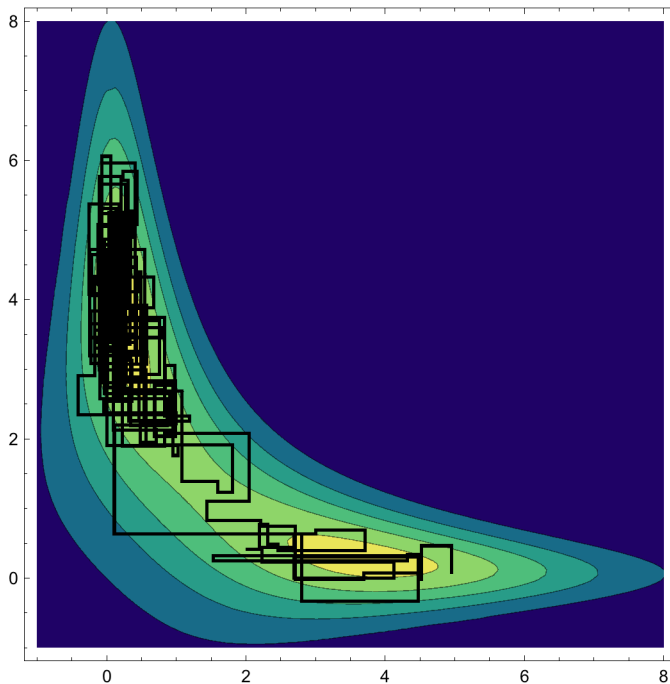
```
traj = Transpose[Join[{x[[1 ;; 1000]], y[[1 ;; 1000]]}]];
gtraj0 = ListPlot[traj, Joined -> False,
  PlotRange -> {{-1, 8}, {-1, 8}}, PlotStyle -> Pink, Axes -> True]
```



```

traj2 = Take[traj, {Dimensions[traj][[1]] - 300, Dimensions[traj][[1]], 1}];
gtraj = ListPlot[traj2, Joined → True, PlotRange → {{-1, 8}, {-1, 8}},
  PlotStyle → {Black, Thickness[.005]}, Axes → False];
Show[{gcontour, gtraj}]

```



Metropolis-Hastings

We've gone from special cases to more general. It can be shown that Gibbs Sampling, for example, is a special case of a more general class of samplers called Metropolis-Hastings.

The origins

```
In[448]:= Clear[x, x1, xx, uu, x0, u, pD, qD, A, A0];
```

The first function below is the target distribution. We pretend we can't sample from it. However, given a possible sample we can evaluate its probability:

```
In[449]:= pD = MixtureDistribution[{2, 1},
  {NormalDistribution[], NormalDistribution[2, 1 / 4]}];
```

This next function below is the proposal distribution. We can sample from it and given a possible sample value as input, we can calculate its value.

```
In[450]:= qD[x_] := NormalDistribution[x, 4];
```

```
In[451]:= x0 = -1.0; (*initial value*)
uu := RandomVariate[UniformDistribution[]];
(*xx[x_] := RandomVariate[qD[x]];*)
```

Define an "acceptance function" A

```
In[453]:= (*A[x1_, x0_] := Min[1.0, PDF[pD, x1]/PDF[pD, x0]];*)
```

```
In[454]:= A0[x1_, x0_] :=
  Min[1.0, (PDF[pD, x1] * PDF[qD[x0], x1]) / (PDF[pD, x0] * PDF[qD[x1], x0])];
```

The above acceptance function is the general case -- it puts the Hastings in Metropolis-Hastings.

If you check the above **A0[]** function, you'll see that if the proposal distribution is symmetric (you can swap x_1 and x_0 , and the computed value stays the same). So in our case of a symmetric normal distribution for sampling, we can use:

```
In[455]:= A[x1_, x0_] := Min[1.0, (PDF[pD, x1]) / (PDF[pD, x0])];
```

This is sometimes called random-walk Metropolis.

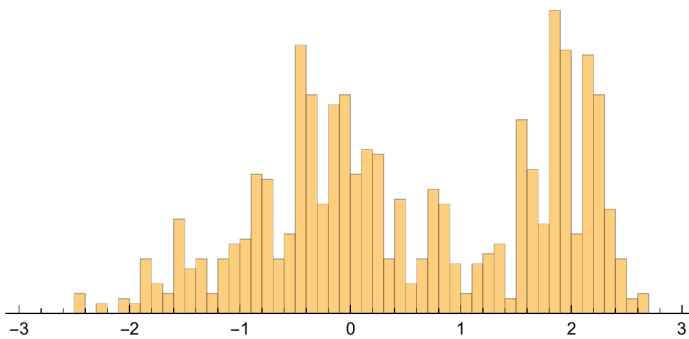
```
In[456]:= n = 1000;
samples = ConstantArray[0.0, n];

samples[[1]] = x0;
For[i = 1, i < n, i++,
  x1 = RandomVariate[qD[samples[[i]]]];

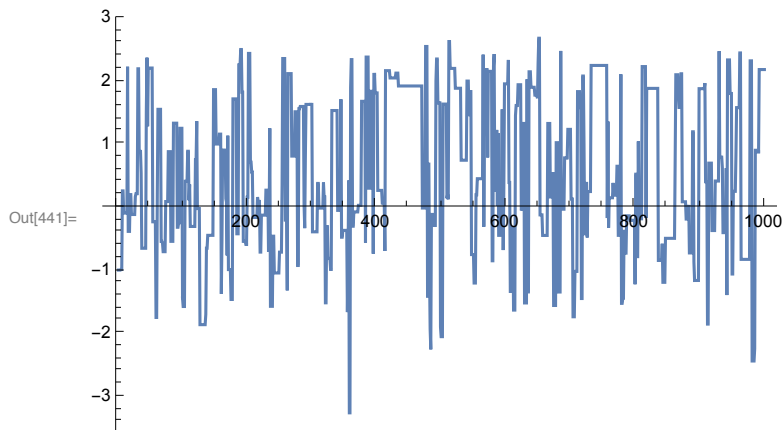
  If[uu < A[x1, samples[[i]]], samples[[i + 1]] = x1, samples[[i + 1]] = samples[[i]]];
```

```
In[423]:= gH = Dynamic[Histogram[samples, 48, "PDF", PlotRange -> {{-3, 3}, {0, .8}}]]
```

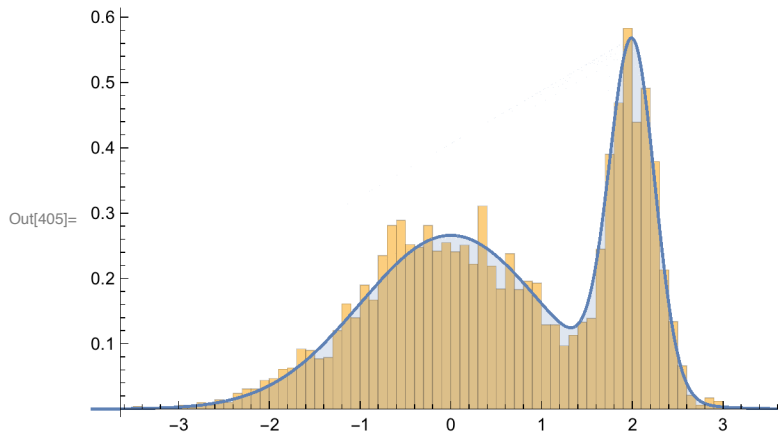
Out[423]=



```
In[441]:= Dynamic[ListPlot[samples, Joined → True]]
```



```
In[404]:= gpD = Plot[{PDF[pD, x]}, {x, -5, 6}, Filling → Axis];
Show[{Histogram[samples, 48, "PDF"], gpD}]
```



Side note: The original Metropolis et al. paper and simulated annealing papers are among the top 100 most cited papers according to Web of Science and Nature: <http://www.nature.com/news/the-top-100-papers-1.16224>

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. , Equation of state calculations by fast computing machines. J. Chem. Phys. 21,1087–1092.

Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. Optimization by simulated annealing. Science, 220, 671–680.

Also, try scholar.googleing: geman

So what is a Markov Chain?

Again, we've introduced material inductively. In fact, Gibbs Sampling and the Metropolis-Hastings methods are both examples Markov Chain Monte Carlo. Where does the Markov Chain come in?

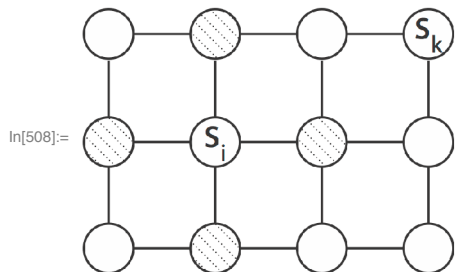
$$p(x^{(i)} | x^{(i-1)}, \dots, x^{(1)}) = T(x^{(i)} | x^{(i-1)})$$

Under certain conditions, with large enough i , the evolution of samples x^i will come to reflect the distribution of an “invariant distribution”, which with the right design corresponds to our desired target distribution. Once we have an estimate of that, we can calculate various statistics on it, such as mean values of marginals. The conditions can be illustrated with searches on the web (Andriew et al., 2003). A transition takes us from one node (link) to the next. One would like to avoid getting trapped in loops and one would like to be able to access any link in the net. These two constraints correspond to two conditions, aperiodicity and irreducibility which in Markov Chains are required for the chain to evolve such that samples are drawn from the target distribution. Google’s original PageRank algorithm used a transition matrix with added noise to avoid aperiodicity and irreducibility.

Gibbs sampling texture synthesis demo

Let’s look at an application to pattern synthesis. We’ve already looked at the problem of visual interpolation. This was an example of a more general principle of perception in which similar features reinforce each other, providing the basis for grouping. We discuss this more later when we talk about lateral organization in neural architectures for vision.

Grouping similar features can be represented by an undirected graph:



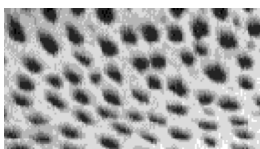
Suppose the nodes represent pixel values over some probability model of images, such as all the images of “grass”. In general, the probability of activation, s_i , of node i will depend on the neighbors j in some neighborhood of i : $j \in N_i$. A Markov Random Field (MRF) has the following property:

$$p(s_i | s_j; \text{all } j \neq i) = p(s_i | s_j; j \in N_i)$$

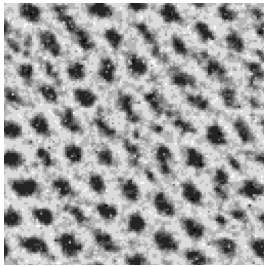
In other words, to draw samples from $p(s_i | \text{given all the other pixels})$, it is sufficient to draw from $p(s_i | \text{in a neighborhood of } i)$.

Texture perception relies in part on local regularities in images: MRFs

Texture modeling has received considerable attention in biological vision and computer graphics. 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:



In general, it is a hard problem to draw true samples from high dimensional spaces. One needs a quantitative model of the distribution and a method such as Gibbs sampling to draw samples.

For a recent example of an application of texture synthesis to understanding the networks of the brain’s visual system, see Freeman, J & Simoncelli, E. P. (2011)

Often it isn’t important to be sure that one has a “true sample”, and other techniques can be used to produce random patterns. E.g. one can first draw sample of white noise, and then iteratively adjust the texture to match statistics of the model, e.g. <http://www.cns.nyu.edu/~lcv/texture/>

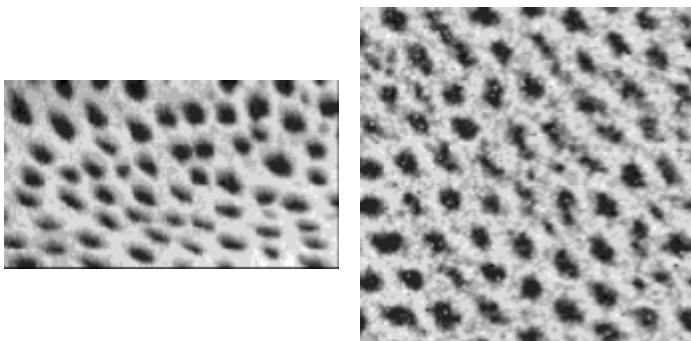
Texture synthesis: An example of generative modeling of data

Last time we took a quick look at the application of the Boltzmann machine 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.

Here is an observed sample on the left, and a synthesized sample after Minimax entropy learning on the right:

```
In[509]:= GraphicsRow[ { Image[  ], Image[  ] }
```

Out[509]=



We drew the analogy to dreaming or conscious imagery. A potentially related phenomenon is hypnagogic imagery.

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

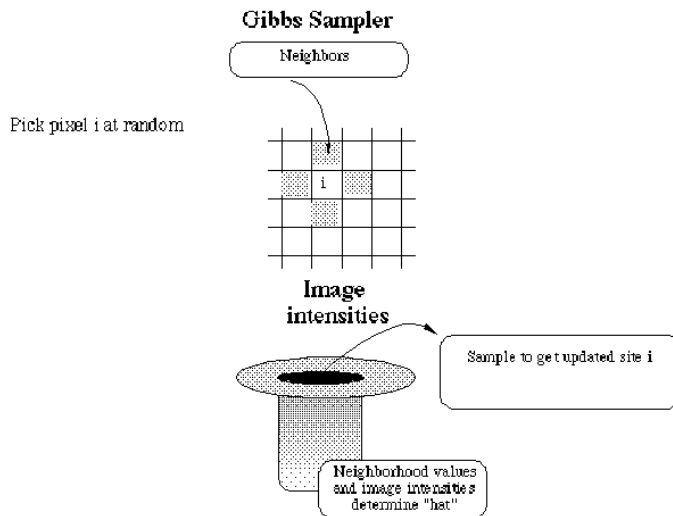
$$\text{Local energy (potential) at location } i = \sum_{j \in N(i)} f(V_i - V_j)$$

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

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

Sampling from textures using local updates

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



Demo

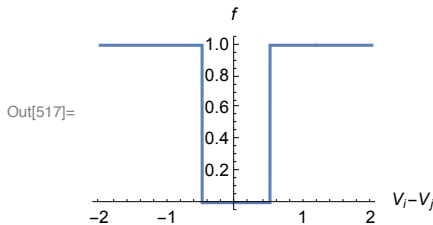
Set up image arrays and useful functions

```
In[510]:= size = 64; 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;
```

We can try several types of potentials.

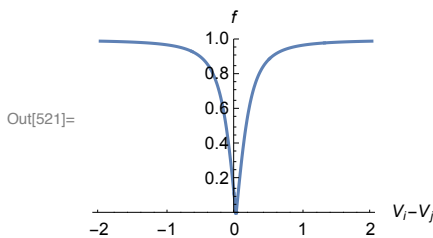
Ising potential

```
In[514]:= 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}]
```



Geman & Geman potential

```
In[518]:= Clear[f];
f[x_, s_, n_] := N[ $\sqrt{\left(\text{Abs}\left[\frac{x}{s}\right]^n / \left(1 + \text{Abs}\left[\frac{x}{s}\right]^n\right)\right)}$ ];
s0 = 0.25; n0 = 2;
Plot[f[x, s0, n0], {x, -2, 2}, PlotRange -> {0, 1}, AxesLabel ->
  {"\!\(\(*SubscriptBox[\(V\), \(\i\)]\) - \!\(\(*SubscriptBox[\(V\), \(\j\)]\) \)", f}]
```



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

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

```
In[522]:= 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).

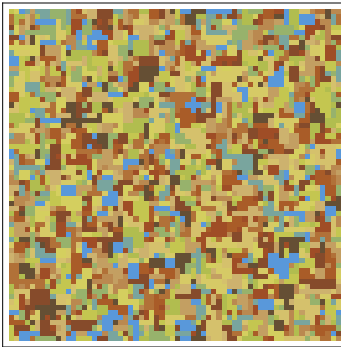

```

In[524]:= 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[525]:= gg = Dynamic[
  ArrayPlot[brown, Mesh -> False, ColorFunction -> ColorData["SouthwestColors"],
  PlotRange -> {1, ngray}, ImageSize -> Small]]

```

Out[525]=



```

In[526]:= 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];

```

Out[526]= \$Aborted

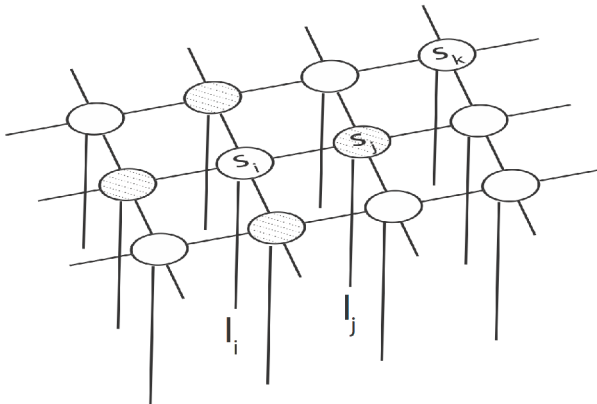
"Drawing" a pattern sample

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?

Denoising, and grouping features contingent on lower level input

A MRF can be constrained by image data. For example, suppose the true image is corrupted by noise. Fidelity to input image data can be constrained by prior assumptions about what the true image should be.

Alternatively, the nodes s could represent some useful world property to be estimated, such as a material's intrinsic reflectivity (or surface color).



Appendix

Rejection sampling with uniform proposal distribution

The uniform distribution doesn't cover the whole range of the target distribution, but may be good enough.

```
M = 4;
```

```
pD = MixtureDistribution[{2, 1},
  {NormalDistribution[], NormalDistribution[2, 1/4]}];
```

```
qD2 = UniformDistribution[{-3, 3}];
```

```
gqD = Plot[{M * PDF[qD2, x]}, {x, -3, 4}, Filling -> Axis, PlotRange -> {-5, 6}];
```

```
Clear[u, x];
```

```
u := RandomVariate[UniformDistribution[]];
```

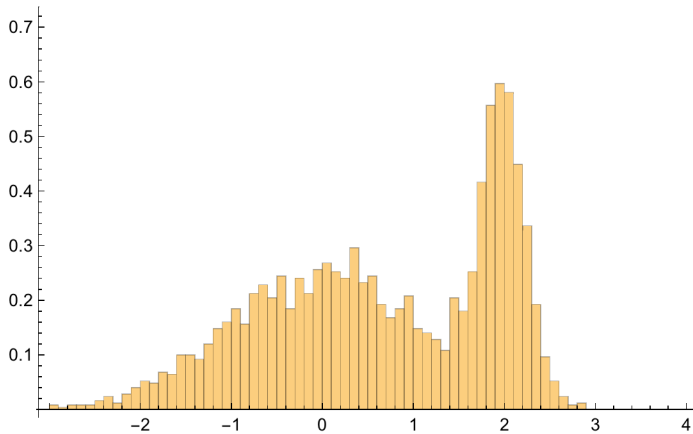
```
x := Module[{n}, n = 1;
```

```
  While[Abs[xr = RandomVariate[qD2]] > 3, xr;
```

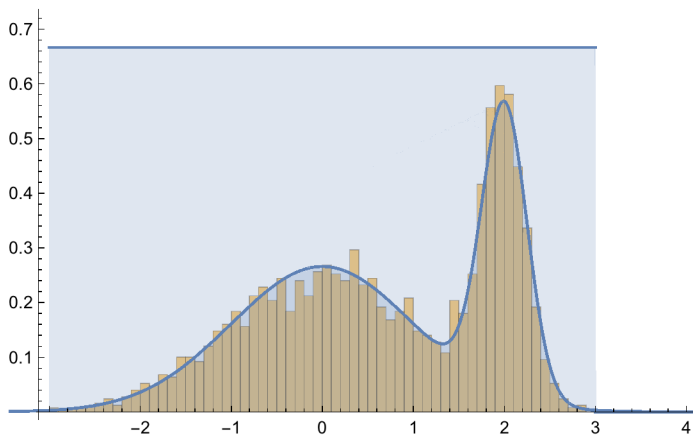
```
    n++]; Return[xr];
```

```
samples = Table[If[(u < PDF[pD, x0 = x] / (M * PDF[qD2, x0])), x0, Null], {10 000}];
```

```
gH = Histogram[samples, 48, "PDF", PlotRange -> {{-3, 4}, {0, .7}}]
```



```
gpD = Plot[{PDF[pD, x]}, {x, -5, 6}, Filling -> Axis, PlotRange -> {-3, 4}];  
Show[{gH, gpD, gqD}]
```



References

Andrieu, C., De Freitas, N., Doucet, A., & Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine Learning*, 50(1), 5–43.

Battaglia, P. W., Kersten, D., & Schrater, P. R. (2011). How haptic size sensations improve distance perception. *PLoS Computational Biology*, 7(6), e1002080. doi:10.1371/journal.pcbi.1002080

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6), 721–741.

Sundareswara, R., & Schrater, P. R. (2008). Perceptual multistability predicted by search model for Bayesian decisions. *Journal of Vision*, 8(5), 12–12. doi:10.1167/8.5.12

Vul, E., Goodman, N. D., Griffiths, T. L., & Tenenbaum, J. B. (2012). One and done? Optimal decisions from very few samples. *Proceedings of the 31st Annual Meeting of the Cognitive Science Society*,

148–153.