Introduction to Neural Networks

Probability and neural networks

Initialize standard library files:

```
Off[General::spell1];
SetOptions[ContourPlot, ImageSize → Small];
SetOptions[Plot, ImageSize → Small];
SetOptions[ListPlot, ImageSize → Small];
```

Introduction

Last time

From energy to probability. From Hopfield net to Boltzmann machine. We showed how the Hopfield network, which minimized an energy function, could be viewed as finding states that increase probability.

Neural network computations from the point of view of statistical inference

By treating neural network learning and dynamics in terms of probability computations, we can begin to see how a common set of tools and concepts can be applied to:
1. **Inference**: as a process which makes the best guesses given data. E.g. find H such that p(H | data) is biggest.
2. **Learning**: a process that discovers the parameters of a probability distribution. E.g. find weights such that
p(weights | data) is biggest
3. **Generative modeling**: a process that generates data from a probability distribution.
E.g. draw samples from p(data | H)

Today

Boltzmann machine: learning the weights
Probability and statistics overview
Introduction to drawing samples

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 (supervised learning).

If they are just stimulus units, then the network observes and organizes its interpretation of the stimuli that arrive (unsupervised learning).

Our goal is to have the hidden units discover the structure of the environment. Once learned, if the network was left to run freely using stochastic sampling, 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, V:

P(V) - probability of visible units taking on certain values determined by the environment.
P'(V) - 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").

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

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 pij is the probability of Vi and Vj both being 1 when environment is clamping the states at thermal equilibrium averaged over many samples. p'ij is the probability of Vi and Vj being 1 when the network is running freely without the environment at equilibrium.

Derive an expression for the KL divergence between two univariate gaussian distributions of means u1 and u2, each with a standard deviation of sd.

Relatives and descendants of Boltzmann machines

As noted earlier, convergence through simulated annealing can be impractically slow. The mean field approximation is one technique used to improve convergence (cf. Ripley, 1996). Mean field methods come from physics where one simplifies the problem by replacing node values with averages, similar to the transition from the discrete to analog Hopfield networks.

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 where some of the variables can be designated inputs, others outputs, and others hidden. Learning large scale joint distributions is known to be a hard problem in statistics. We'll discuss the general problem later. 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 special case called Restricted Boltzmann Machines (RBMs) have been shown to be very useful for a number of problems. RBMs are divided up into layers of units that have connections between nearby layers but not within layers. RBMs have been used to discover efficient representations of inputs that can then be fine tuned for a task, such as digit classification, using backpropagation (Hinton, G. E., & Salakhutdinov, R. R., 2006).

See also sleep-wake algorithms & "deep belief" networks references.

For a computational example of an application to learning and inference, see:

http://www.cs.toronto.edu/~hinton/digits.html

And for a related neuroscience study see: Berkes, P., Orbán, G., Lengyel, M., & Fiser, J. (2011). Spontaneous cortical activity reveals hallmarks of an optimal internal model of the environment. Science, 331(6013), 83–87

Probability and statistics overview

To understand modern neural network theory, we need to understand probablistic modeling.Why?

Basic rules of probability

Joint probabilities. Suppose we know everything there is to know about a set of variables (A,B,C,D,E). What does this mean in terms of probability? It means that we know the joint distribution, p(A,B,C,D,E).

In other words, for any particular combination of values (A=a,B=b, C=c, D=d,E=e), we can calculate, look up in a table, or determine some way or another the number p(A=a,B=b, C=c, D=d,E=e), for any particular instances, a, b, c, d, e.

Rule 1: Conditional probabilities from joints: The product rule

Probability about an event changes when new information is gained.

Prob(X given Y) = p(X|Y)

$$p(X \mid Y) = \frac{p(X, Y)}{p(Y)}$$

$$p(X, Y) = p(X \mid Y)\, p(Y)$$

The form of the product rule is the same for densities as for probabilities.

Independence

Knowledge of one event doesn't change the probability of another event. This can be expressed in terms
of conditional probability as:

p(X)=p(X|Y) which by the product rule is:

p(X,Y)=p(X)p(Y)

Rule 2: Lower dimensional probabilities from joints: The sum rule (marginalization)

$$p(X) = \sum_{i=1}^{N} p(X, Y(i))$$

$$p(x) = \int_{-\infty}^{\infty} p(x, y)\, dy$$

These distributions on the left are called *marginal distributions*. They are obtained by "*summing over"* or "*integrating out*" the variables we are not interested in.

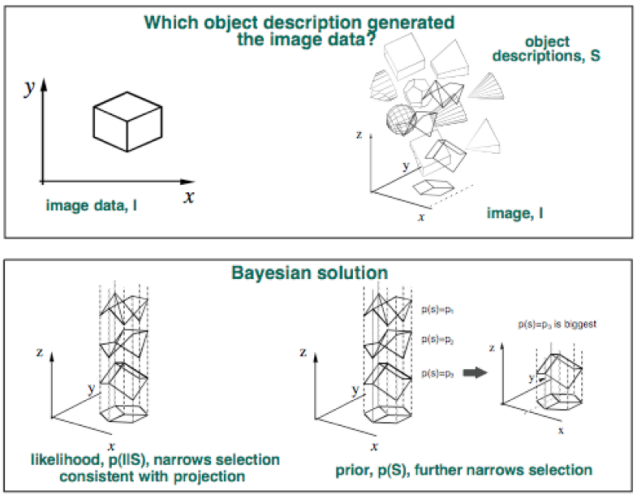
Rule 3: Bayes' rule

From the product rule, and since p[X,Y] = p[Y,X], we have:

$$p(Y \mid X) = \frac{p(X \mid Y)\, p(Y)}{p(X)}, \text{ and using the sum rule, } p(Y \mid X) = \frac{p(X \mid Y)\, p(Y)}{\sum_Y p(X, Y)}$$

Developing intutions: Bayes inference applied to visual perception

Consider the problem of interpreting the line drawing shown in the upper left part of the figure below.



What is the “data”? What is the perceptual hypothesis? What are the probablilistic relationships between them? If we can answer these questions, we can first pose problems of perception as problems of perceptual inference. Then perhaps later we can investigate the possibility that a neural network is solving the perceptual inference problem.

$$p[\text{hypothesis} \mid \text{data}] = \frac{p[\text{data} \mid \text{hypothesis}]\, p[\text{hypothesis}]}{p[\text{data}]}$$

$$p[S \mid I] = \frac{p[I \mid S]\, p[S]}{p[I]}$$

Usually, we will be thinking of the **“hypothesis”** term as a random variable over the hypothesis space, and “**data”** comes from some measurements. So for visual inference, **S = hypothesis** (the scene), and **I = data** (the image data), and **I = f(S)**. In perceptual inference, optimal decisions are based on:

p(S|I),

the **posterior** probability of the scene given the image, i.e. what you get when you condition the joint by the image data. The posterior is often what we'd like to base our decisions on, because as we discuss below, picking the hypothesis **S** which maximizes the posterior (i.e. maximum a posteriori or **MAP** estimation) minimizes the average probability of error.

p(S) is the **prior** probability of the scene.

p(I|S) is the **likelihood** of the scene. Note this is a probability over values of **I**, but not of **S**. So $\int$ **p(I|S)dI = 1.**

A MAP decision based on p(**S**|**I**) does not necessarily involve explicit computations of the likelihood and prior as in Bayes rule. Such inferential processes are sometimes called “discriminative”. Generative inferences rely on explicit knowledge of how the data might have been generated, as represented by the likelihood p(**I** | **S**), perhaps together with a model of the prior, p(**S**).

In general, **S** is a high-dimensional description of scene parameters, and **I** is a vector of image measurements.

One of the challenges of visual inference is that the data by itself is insufficient to decide the value(s) of the hypothesis. In other words, the image measurements **I** don't uniquely determine the scene description **S**. A visual system must make a good guess. To do this well, it needs to take into account all the ways the image might have been generated--i.e. the causes of image measurements, together with a model of the prior probabilities of those causes. To give some intuition, here is an example of how the likelihood and priors represent constraints in Bayesian visual inference:

We will return to examples of Bayesian inference in perception later. But first, let's go over some basic definitions in statistics.

Statistics

The definitions and properties of statistics are important to know about in the context of what it means to have "image data". Image data can come in many forms, such as the raw intensity values in a picture, or some simple measurement like how far a pixel intensity is from its neighbors. Often one needs to summarize some quantity in a complex pattern, like an image. These summaries often take the form of statistics. Statistics like the mean (or expectation) and variance can be sufficient to characterize a whole distribution, such as a gaussian. This assumption is used in parametric statistics. But other times there is no simple distribution to characterize the underlying probablistic generative process. Then we might represent the distribution in terms of frequencies with respect to some relevant measurements. A histogram of image intensities is an example--i.e. how frequently do graylevels 0, 1, 2, ...255 occur in the image?

Expectation & variance

Analogous to center of mass:

Definition of expectation or average:

$$\text{Average}[X] = \overline{X} = E[X] = \Sigma\, x[i]\, p[x[i]] \sim \sum_{i=1}^{N} x_i / N$$

$$\mu = E[X] = \int x\, p(x)\, dx$$

Some rules:

E[X+Y] = E[X] + E[Y]
E[aX] = a E[X]
E[X+a] = a + E[X]

Definition of variance:

$$\sigma^2 = \text{Var}[X] = E[[X-\mu]\text{^}2] = \Sigma_{j=1}^{N} p_j (x_j - \mu))^2 \sim \sum_{i=1}^{N} (x_i - \mu)^2 / N$$

$$\text{Var}[X] = \int (x-\mu)^2\, p(x)\, dx$$

Standard deviation:

$$\sigma = \sqrt{\text{Var}[X]}$$

Some rules:

$$\text{Var}[X] = E\left[X^2\right] - E[X]^2$$
$$\text{Var}[aX] = a^2\, \text{Var}[X]$$

Covariance & Correlation

Covariance:

Cov[X,Y] =E[[X - μ_X] [Y - μ_Y]]

Correlation coefficient:

$$\rho[X, Y] = \frac{\text{Cov}[X, Y]}{\sigma_X \sigma_Y}$$

Covariance matrix

Suppose now that X is a vector: {X_1, X_2,...}

Then we can describe the covariance between pairs of elements of X:

$$\Sigma_{ij} = \text{cov}[X_i, X_j] = E[[X_i - \mu_{X_i}] [X_j - \mu_{X_j}]] \sim \sum_{n=1}^{N} (x_i^n - \mu_{X_i}) (x_j^n - \mu_{X_i})^T / N$$

In matrix form, the covariance can be written:

$$\Sigma = \text{cov}[X] = E[(X-E[X])(X - E[X])^T]$$

In other words, the covariance matrix can be approximated by the average outer product. In the language of neural networks, it is proportional to a Hebbian matrix memory of pair-wise relationships.

Independent random variables

If p(X,Y) = p(X)p(Y), then

$$E[X\,Y] = E[X]\,E[Y] \text{ (uncorrelated)}$$
$$\text{Cov}[X, Y] = \rho[X, Y] = 0$$
$$\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y]$$

If two random variables are uncorrelated (ρ[X,Y] = 0), they are not necessarily independent.

Two random variables are said to be orthogonal if their correlation is zero.

Degree of belief vs., relative frequency

What is the probability that it will rain tomorrow? Assigning a number between 0 and 1 is assigning a degree of belief. These probabilities are also called subjective probabilities. It is nonsense to think of “repeating” tomorrow to see how frequently it rains.

What is the probability that a coin will come up heads? In this case, we can imagine actually doing an experiment to find out. Flip the coin n times, and count the number of heads, say h[n], and then set the probability, p ≃ h[n]/n -- the relative frequency (which is a statistic). Of course, if we did it again, we may not get the same estimate of p. One solution often given is:

$$p = \lim_{n \to \infty} \frac{h(n)}{n}$$

A problem with this, is that in general there is no guarantee that *a* well – defined limit exists.

In some domains we may have enough data that we can measure statistics, and model probabilities of both inputs and outputs. So the relative frequency interpretation seems reasonable. In practice, the dimensions of many problems in perception, cognition, language, and memory are so high, that it is

impractical to do this.

Suppose you wanted to estimate the joint probabilities of 6 letter combinations from a database of english words (or worse yet, 8x8 pixel images). There are over 300 million possible combinations of 26 letters--i.e. over 300 million "bins" for your word counts. Most of these would have zero or near-zero entries, and it would be hard to get good estimates of most of the joint probabilities of 6 letter combinations. Although there are ways to estimate "objective priors" in high-dimensional spaces (see below), once we use the statistical framework to model perception, say of a particular cue, then probabilities can become more like "subjective unconscious beliefs". From a modeling perspective, one can treat a specific prior as an assumed ingredient, and test to see how well the model accounts for the data, and how well it predicts new data. If it the model does a poor job empirically, then one can go back and question whether an alternative prior could improve the predictions.

Sidenote: Parametric vs. non-parametrics statistics and neural networks

Principle of insufficient reason

Principle of symmetry

Suppose we have N events, x[1], x[2], x[3],..., x[N] that are all physically identical except for the label. In the absence of any other information about these events, it seems reasonable to assume that

$$\text{prob}(x(1)) = \text{prob}(x(2)) = \text{prob}(x(3)) = \text{prob}(x(N)) = \frac{1}{N}$$

In other words, intuition tells us that if we have no additional pertinent data about the events, we should assume that they are uniformly distributed. I.e., assume a *uniform prior*.

What about the continuous case where there is no reason to assume any particular value at all between -∞ and +∞? If one assumes the prior is constant, then it has to approach zero everywhere to make sure the sum between -∞ and +∞ is 1...mmm...
A constant value is often assumed anyway, and this is called an improper prior.

Information theory and Maximum entropy

Information theory provides a powerful extension to the principle of symmetry. Information of event (or signal) X is:

$$\text{Information}[X] = -\log_2(p(X))$$

The more improbable an event, the more “surprising” it is in some sense, and the more information it provides.

Using the definition of expectation above, we can specify the expectation of information (or average information), which is called entropy. Entropy of a random variable X with probability distribution p[X] is:

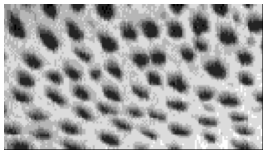
$$H(X) = \text{Average}(\text{Information}[X]) = -\sum_X p(X) \log_2(p(X))$$

It can be shown that out of all possible probability distributions for our above case, H(X) is biggest for the uniform distribution, p(X)=1/N. Maximum entropy is looking like the symmetry principle.

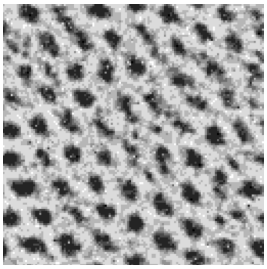
Indeed it turns out that a useful generalization of the principle of symmetry is maximum entropy. For

example, out of all possible probability distributions of a random variable with infinity range, but with a specific mean and standard deviation, the Gaussian is unique in having the largest entropy. If the range goes from zero to infinity, and we know the mean, the maximum entropy distribution is an exponential (Cover and Thomas).

An interesting application of the maximum entropy principle is to learning image textures joint probabilities: p(I[1],..., I[N]), where N is very big, but where one has only a relatively small number of measured statistics relative to the number of possible images (which is really huge). The measurements underdetermine the dimensionality of the probability space--i.e. there are many different probability distributions that give the same statistics. So the principle of symmetry, (also called insufficient reason, principle of indifference), says to choose the one with the maximum entropy. This provides a way to model high dimensional priors. And in fact, this has been done for texture models briefly described in a previous lecture (Zhu et al.). 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. And in the above text example, the sample was not drawn from an explicit probability distribution. The previous lecture mentioned an example of an application of texture synthesis to understanding the networks of the brain's visual system, see Freeman, J & Simoncelli, E. P. (2011)

Mathematica functions for multivariate distributions & exploring marginals

Later we'll tackle the problem of drawing samples from high-dimensional, but structured distributions. But here we'll use *Mathematica* built-in functions to illustrate some of the above ideas.

Multivariate gaussian probability density

An *n*-variate multivariate gaussian (multinormal) distribution with mean vector μ and covariance matrix Σ is denoted $N_n(\mu, \Sigma)$. The density is:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}\,\mathrm{Det}[\Sigma]^{1/2}}\,\mathrm{Exp}\left[-\frac{1}{2}(\mathbf{x}-\mu)^{T}\,\Sigma^{-1}\,(\mathbf{x}-\mu)\right]$$

You can read in the *Mathematica* package which has predefined **MultinormalDistribution[μ, Σ].**

```
(*Needs["MultivariateStatistics`"]*)
```

Then, for example, you can define the probability density function for mean vector $\{\mu_1, \mu_2\}$, and covariance matrix $\{\{\sigma_{11}^2, \rho * \sigma_{11} * \sigma_{22}\}, \{\rho * \sigma_{11} * \sigma_{22}, \sigma_{22}^2\}\}$, where ρ parameterizes correlation.

In[285]:=
```
Σ = {{σ11^2, ρ * σ11 * σ22}, {ρ * σ11 * σ22, σ22^2}};
PDF[MultinormalDistribution[{μ1, μ2}, Σ], {x, y}]
```

Out[286]=
$$\frac{1}{2\,\pi\,\sqrt{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}}$$
$$e^{\frac{1}{2}\left(-\frac{5.25423\times10^{-9}\,(0.202363-1.\,\mu_1)\,(1.90323\times10^{8}\,y\,\rho\,\sigma_{11}-1.90323\times10^{8}\,\rho\,\mu_2\,\sigma_{11}-3.85144\times10^{7}\,\sigma_{22}+1.90323\times10^{8}\,\mu_1\,\sigma_{22})}{(-1.+\rho^2)\,\sigma_{11}^2\,\sigma_{22}}-\frac{5.25423\times10^{-9}\,(y-1.\,\mu_2)\,(-1.90323\times10^{8}\,y\,\sigma_{11}+1.90323\times10^{8}\,\mu_2\,\sigma_1}{(-1.+\rho^2)\,\sigma_{11}\,\sigma_{22}^2}\right.}$$

In[287]:=
$$e^{\frac{1}{2}\left(-(y-\mu_2)\left(\frac{(y-\mu_2)\,\sigma_{11}^2}{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}-\frac{\rho\,(x-\mu_1)\,\sigma_{11}\,\sigma_{22}}{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}\right)-(x-\mu_1)\left(-\frac{\rho\,(y-\mu_2)\,\sigma_{11}\,\sigma_{22}}{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}+\frac{(x-\mu_1)\,\sigma_{22}^2}{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}\right)\right)}\Big/\left(2\,\pi\,\sqrt{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}\right)$$

Out[287]=
$$e^{\frac{1}{2}\left((-y+\mu_2)\left(\frac{(y-\mu_2)\,\sigma_{11}^2}{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}-\frac{\rho\,(1.29075-\mu_1)\,\sigma_{11}\,\sigma_{22}}{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}\right)-(-0.40977-\mu_1)\left(-\frac{\rho\,(y-\mu_2)\,\sigma_{11}\,\sigma_{22}}{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}+\frac{(1.22789-\mu_1)\,\sigma_{22}^2}{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}\right)\right)}\Big/$$
$$\left(2\,\pi\,\sqrt{\sigma_{11}^2\,\sigma_{22}^2-\rho^2\,\sigma_{11}^2\,\sigma_{22}^2}\right)$$

In[288]:=
```
Σ
```

Out[288]=
$$\{\{\sigma_{11}^2,\ \rho\,\sigma_{11}\,\sigma_{22}\},\ \{\rho\,\sigma_{11}\,\sigma_{22},\ \sigma_{22}^2\}\}$$

In[289]:=
```
Covariance[MultinormalDistribution[{μ1, μ2}, Σ]]
```

Out[289]=
$$\{\{\sigma_{11}^2,\ \rho\,\sigma_{11}\,\sigma_{22}\},\ \{\rho\,\sigma_{11}\,\sigma_{22},\ \sigma_{22}^2\}\}$$
$$\{\{\sigma_{11}^2,\ \rho\,\sigma_{11}\,\sigma_{22}\},\ \{\rho\,\sigma_{11}\,\sigma_{22},\ \sigma_{22}^2\}\}$$
$$\{\{\sigma_{11}^2,\ \rho\,\sigma_{11}\,\sigma_{22}\},\ \{\rho\,\sigma_{11}\,\sigma_{22},\ \sigma_{22}^2\}\}$$

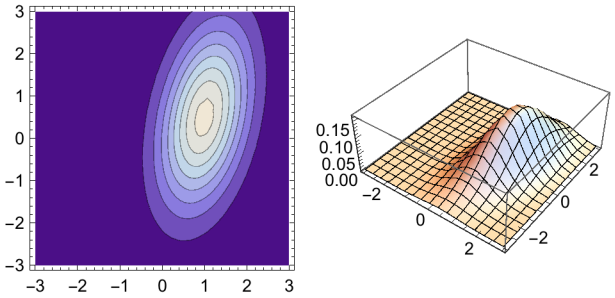
Examples of PDF, CDF

A specific example, MultinormalDistribution[μ, Σ]

```
m1 = {1,1/2};
r=(1/2)*{{1,2/3},{2/3,4}};
ndist = MultinormalDistribution[m1, r];
pdf = PDF[ndist, {x1, x2}]
```

$$\frac{1}{4\,\sqrt{2}\,\pi}3\,e^{\frac{1}{2}\left(-(-1+x1)\left(\frac{9}{4}\,(-1+x1)-\frac{3}{8}\left(-\frac{1}{2}+x2\right)\right)-\left(-\frac{3}{8}\,(-1+x1)+\frac{9}{16}\left(-\frac{1}{2}+x2\right)\right)\left(-\frac{1}{2}+x2\right)\right)}$$

```
GraphicsRow[{g1 = ContourPlot[PDF[ndist, {x1, x2}], {x1, -3, 3}, {x2, -3, 3}],
   g13D = Plot3D[PDF[ndist, {x1, x2}], {x1, -3, 3}, {x2, -3, 3}]}]
```



Calculating probabilities using the CDF

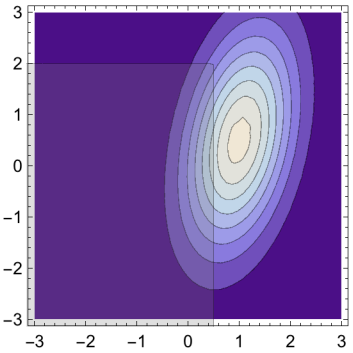
What is the probability of x_1 and x_2 taking on values in the region $x_1 < .5 \cap x_2 < 2$?
Recall that the cumulative distribution function is given by:

$$\text{CDF}(x_1, x_2) = \int_{-\infty}^{x_2} \int_{-\infty}^{x_1} p(x_1, x_2)\, dx_1\, dx_2$$

So the answer is the area under the PDF shown below.

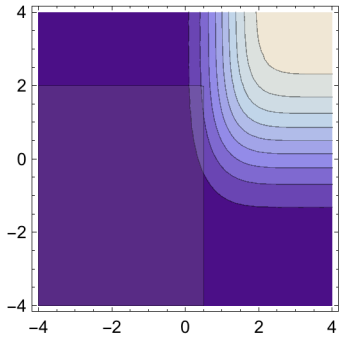
```
grp = RegionPlot[x1 < .5 && x2 < 2, {x1, -4, 4}, {x2, -4, 4},
    PlotStyle → Directive[Opacity[.25], EdgeForm[], FaceForm[Gray]]];
```

```
Show[{g1, grp}, ImageSize → Small]
```



We could numerically integrate to find the area. Alternatively, the answer can be found from the built-in cumulative distribution function, or CDF. The value corresponds to the height of the contour at $\{x_1, x_2\}$ = {.5, 2.0}. Move your mouse cursor over the contour plot below near the {.5, 2.0} point.

```
gcdf = ContourPlot[CDF[ndist, {x1, x2}], {x1, -4, 4}, {x2, -4, 4},ImageSize → Small];
Show[{ gcdf, grp}, ImageSize → Small]
```



A more precise answer is:

```
CDF[ndist, {.5, 2.0}]
```

```
0.225562
```

Finding the mode

The mode is the value of the random variable with the highest probability or probability density. For discrete distributions, think of it as the most frequent value.

(Sometimes the word “mode” is used to refer to a *local* maximum in a density function. Then the distribution is called multimodal. If it has two modes, it is called bimodal.) There may not be a unique mode--the uniform distribution is an extreme case of this.

For the Gaussian case, the mode vector corresponds to the mean vector. But we can pretend we don't know that, and use the **FindMaximum[]** function to find the maximum and the coordinates where the max occurs:

```
FindMaximum[PDF[ndist, {x1, x2}], {{x1, 0}, {x2, 0}}]
```

```
{0.168809, {x1 → 1., x2 → 0.5}}
```

Marginals

Suppose we want the distribution for just x_1, PDF[x_1] = marginal[x_1]. How do we find it?
Integrate PDF[x_1, x_2] with respect to x_2. This is called calculating the marginal distribution of x_1. It is obtained by integrating out the other variable, x_2. Similarly, we can calculate PDF[x_2].

```
Clear[x1, x2];
marginal[x1_] := ∫_-∞^∞ PDF[ndist, {x1, x2}] dx2;
marginal2[x2_] := ∫_-∞^∞ PDF[ndist, {x1, x2}] dx1;
```

What is the mode of PDF[x_2]?

```
FindMaximum[marginal2[x2], {{x2, 0}}]
```

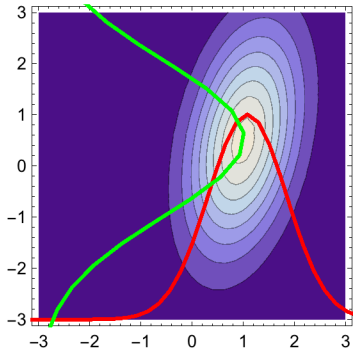
```
{0.282095, {x2 → 0.5}}
```

Now let's plot up the marginals on top of the contour plot of the joint distribution.

```
mt = Table[{x1, marginal[x1]}, {x1, -3, 3, .2}];
g2 = ListPlot[mt, Joined → True, PlotStyle → {Red, Thick}, Axes → False];

mt2 = Table[{x2, marginal2[x2]}, {x2, -3, 3, .4}];
g3 = ListPlot[mt2, Joined → True, PlotStyle → {Green, Thick}, Axes → False];

theta = Pi / 2;
Show[g1, Epilog → {Inset[g2, {0, -3}, {0, 0}], Inset[g3, {-3, 0}, {0, 0},
     Automatic, {{Cos[theta], Sin[theta]}, {Sin[theta], -Cos[theta]}}]}]
```



Drawing samples

As we've used in earlier lectures, drawing samples is done by:

```
RandomVariate[ndist]
```

```
{1.0705, 0.769877}
```

```
{0.936747, 0.146599}
```

```
{0.9697, -0.302945}
```

or

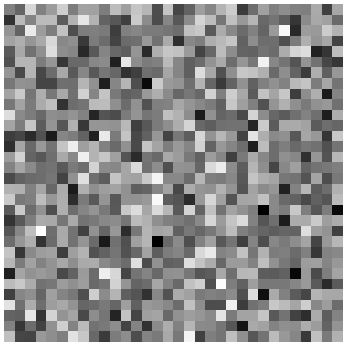
```
RandomReal[ndist]
```

```
{1.90757, -0.21533}
```

```
{2.05819, 2.35961}
```

Here's an example of a "white gaussian noise" image. It is called "white" because there are no correlations between the pixel intensities. Each one is drawn independently of any of the others.

```
width = 32;
data = RandomVariate[NormalDistribution[0, 1], width * width];
Image[Partition[data, width]] // ImageAdjust
```



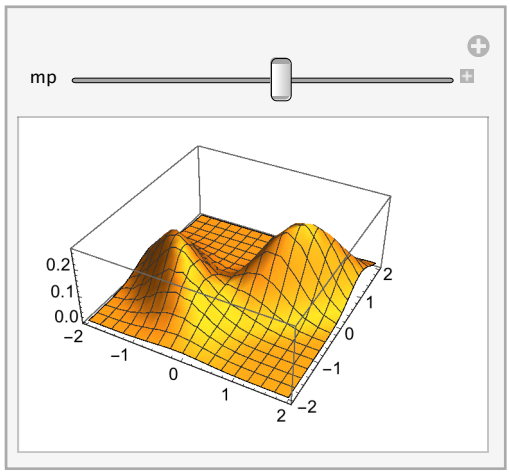
Mixtures of gaussians with MultinormalDistribution[]

Multivariate gaussian distributions are often inadequate to model real-life problems, that for example might involve more than one mode or might have non-gaussian properties. One solution is to approximate more general distributions by a sum or mixture of gaussians.

```
Clear[mix];
r1=0.4*{{1,.6},{.6,1}};
r2=0.4*{{1,-.6},{-.6,1}};
m1 = {1,.5}; m2 = {-1,-.5};
ndist1 = MultinormalDistribution[m1, r1];
ndist2 = MultinormalDistribution[m2, r2];

mix[x_,mp_] := mp*PDF[ndist1, x] + (1-mp)*PDF[ndist2, x];

Manipulate[gg1 = Plot3D[mix[{x1, x2}, mp], {x1, -2, 2},
    {x2, -2, 2}, PlotRange → Full, ImageSize → Small], {{mp, .2}, 0, 1}]
```



Mathematica has a built-in function for defining mixture distributions:

```
𝒟[w_] = MixtureDistribution[{w, 1 - w}, {MultivariatePoissonDistribution[7, {9, 10}],
     MultivariatePoissonDistribution[2, {5, 4}]}];
Manipulate[DiscretePlot3D[PDF[𝒟[w], {x, y}], {x, 0, 30},
  {y, 0, 30}, ExtentSize → Full], {w, 0, 1}]
```

See Zoran and Weiss (2011) for an interesting application of mixture models to discovering visual

features in natural images.

Sampling in low dimensions

We first go over a few basics.

Density mapping theorem

Suppose we have a change of variables that maps a discrete set of x's uniquely to y's: X->Y.

Discrete random variables

No change to probability function. The mapping just corresponds to a change of labels, so the probabilities p(X)=p(Y).

Continuous random variables

In this case, the form of probability density function changes because we require the probability "mass" to be unchanged: p(x)dx = p(y)dy

Suppose, y=f(x)

$p_Y(y)\,\delta y = p_X(x)\,\delta x$

Transformation of variables is used in making random number generators for probability densities other than the uniform distribution, such as a Gaussian.

Below we'll need to use the cumulative distribution function: CDF(x) = prob $(X < x) = \int_{-\infty}^{x} p(X)\,dX$

Univariate sampling

Making a univariate (scalar) gaussian random number generator:

We assume we have a random number generator that provides uniformly distributed numbers between 0 and 1. How can we get numbers that are Gaussian distributed?

Well, the easiest way is to use a built-in function:

```
RandomVariate[NormalDistribution[0, 1]]
```

```
0.726157
```

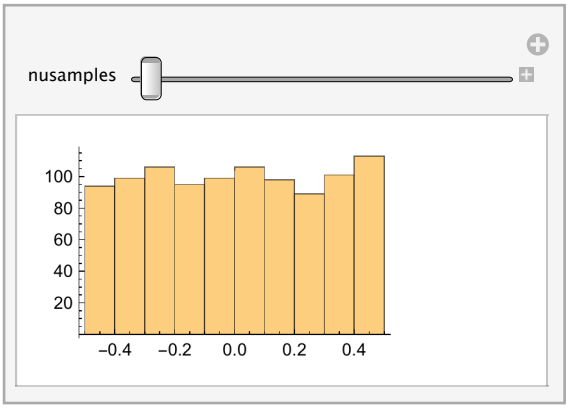
but we'd like to better understand some principles behind generating random numbers for a specified distribution.

Method 1: Just for Gaussian. Use Central Limit Theorem

If all we want to do is make a Gaussian random number generator from a uniformly distributed generator, we can use the Central Limit Theorem. The Central Limit Theorem says that the sum of a sufficiently large number of independent random variables drawn from the same underlying distribution (with finite mean and variance), will be approximately normally distributed. The approximation gets better as the number of samples increases.

Try the cell below with nusamples = 1, 2, .., 10,...

```
Manipulate[zl = Table[Sum[RandomReal[] - nusamples/2, {i, 1, nusamples}], {1000}];
 Histogram[zl, ImageSize → Small], {nusamples, 1, 30, 1}]
```



Method 2: Use Density Mapping theorem. More general.

We'll use the density mapping theorem to turn uniformly distributed random numbers **RandomReal[]** into gaussian distributed random numbers with mean =0 and standard deviation =1.

$$p_Y(y)\,\delta y = p_X(x)\,\delta x$$

$$p_Y(y)\,\frac{\delta y}{\delta x} = p_X(x)$$

Suppose $p_Y(y)$=1 (over the unit interval, but zero elsewhere). Then

$$y(x) = \int_{-\infty}^{x} p_x(x')\,dx' = P(x)$$

Thus if we sample from the uniform distribution to get y, x should be distributed according to $p_X(x)$.
To do this, we need a mapping from y->x. This is given by the inverse cumulative distribution, i.e. $P^{-1}(y)$. Let's implement this. The quick way is to use *Mathematica*'s built-in function to get the inverse cumulative.

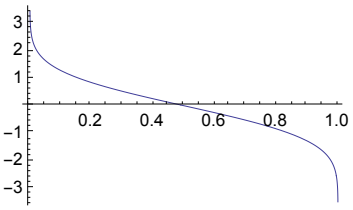
Method 2a: Applied to Gaussian

InverseErf[ ] is the inverse of :

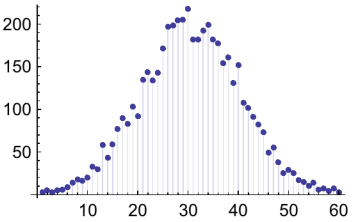
$$erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\,dt$$

We can use this to define a function for the inverse cumulative of a gaussian:

```
Clear[z];
z[p_] := √2 InverseErf[1 - 2 p];
Plot[z[y], {y, 0, 1}]
```




```
binsize = 0.1;
zl = Table[z[RandomReal[]], {5000}];
freq = BinCounts[zl, {-3, 3, binsize}];
ListPlot[freq, Filling → Axis]
```



Method 2b: From scratch: Works for almost any low dimensional distribution.

Suppose we have a discrete representation of any cumulative distribution. How can we generate samples? For illustration purposes, we'll illustrate the method with a discretization of the Gaussian.

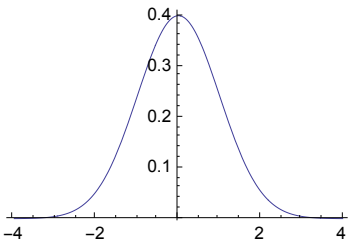
Our first goal is to produce a discrete approximation to the cumulative gaussian. To review where things come from, we'll start with the definition of a Gaussian, and make sure it is normalized.

```
Integrate[Exp[-(x - x0)^2 / (2 * σ^2)], {x, -Infinity, Infinity}]
```

$$\text{ConditionalExpression}\left[\frac{\sqrt{2\,\pi}}{\sqrt{\frac{1}{\sigma^2}}}, \text{Re}\left[\sigma^2\right] > 0\right]$$

Let x0=0 and σ=1:

```
Plot[e^(-x1^2/2) / √(2 π), {x1, -4, 4}]
```



Note that **Plot[PDF[NormalDistribution[0,1],x1],{x1,-4,4}]**; gives the same thing using the built-in normal distribution function.

Cumulative gaussian

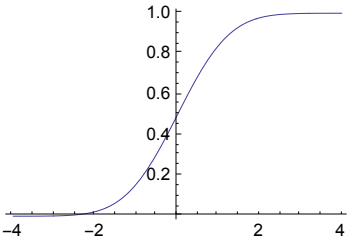
```
Clear[cumulgauss, x, x1];
cumulgauss[x_] := NIntegrate[Exp[-(x1^2) / 2] / (Sqrt[2 * Pi]), {x1, -Infinity, x}]
```

```
cumulgauss[Infinity]
```

```
1.
```

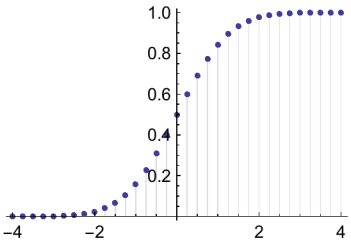
We can plot up cumulgauss:

```
Plot[cumulgauss[x], {x, -4, 4}]
```



Now make a discrete version of the cumulative distribution:

```
lcumulgauss = Table[{x, cumulgauss[x]}, {x, -4.`, 4.`, 0.25`}];
ListPlot[lcumulgauss, Filling → Axis]
```



Remember the reason we are doing this is to show how we can go from this to drawing samples. I.e. it doesn't have to
be a gaussian at this stage, the cumulative could have come from some arbitrary histogram from data gathered elsewhere.

Make inverse cumulative gaussian table

This is a useful trick whenever you want an inverse function, given a discrete representation.

```
invlcumulgauss = RotateLeft[lcumulgauss, {0, 1}];
```

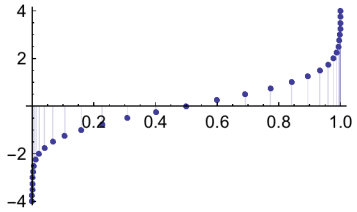
To see what this does, evaluate:

```
{{x1, y1}, {x2, y2}, {x3, y3}}
RotateLeft[{{x1, y1}, {x2, y2}, {x3, y3}}, {0, 1}]
```

```
{{x1, y1}, {x2, y2}, {x3, y3}}
```

```
{{y1, x1}, {y2, x2}, {y3, x3}}
```

```
ListPlot[invlcumulgauss, Filling → Axis]
```

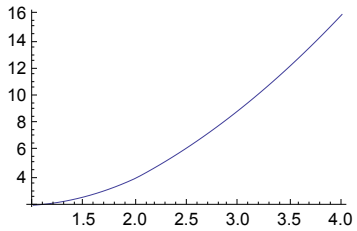


Make interpolated function of the inverse cumulative

Another useful trick.

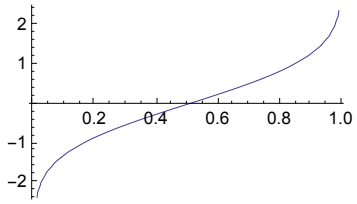
Interpolation works by fitting polynomial curves to the data. Try the test below with various interpolation orders (the default is 3)

```
test = Interpolation[{{1, 2.}, {2, 4}, {3, 9}, {4, 16}}, InterpolationOrder → 2];
Plot[test[x], {x, 1, 4}]
```




```
interinvlcumulgauss = Interpolation[invlcumulgauss];
```

```
Plot[interinvlcumulgauss[x], {x, 0.01, 0.99}]
```



Draw samples with a standard deviation of Sqrt[10]

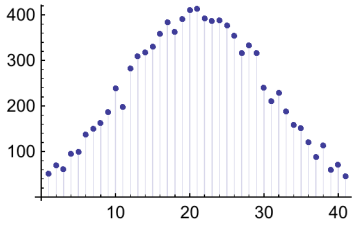
```
Round[10 interinvlcumulgauss[RandomReal[]]]
```

7

Draw a bunch of samples, and plot up histogram

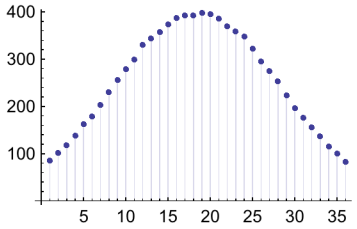
```
z = Table[Round[10 interinvlcumulgauss[RandomReal[]]], {10 000}];
domain = Range[-20, 20];
Freq = (Count[z, #1] &) /@ domain;
```

```
ListPlot[Freq, Filling → Axis]
```



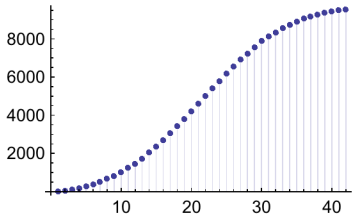
digression...a quick & dirty way to smooth is to do a moving average

```
ListPlot[MovingAverage[Freq, 6], Filling → Axis]
```



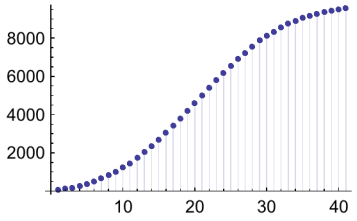
Plot up cumulative histogram

```
CumFreq = FoldList[Plus, 0, Freq];
ListPlot[CumFreq, Filling → Axis]
```



Same thing, with **Accumulate[]** :

```
CumFreq = Accumulate[Freq];
ListPlot[CumFreq, Filling → Axis]
```



Next time

Graphical models and structured distributions
Sampling in higher dimensional spaces

References

Applebaum, D. (1996). Probability and Information . Cambridge, UK: Cambridge University Press.
Cover, T. M., & Joy, A. T. (1991). *Elements of Information Theory*. New York: John Wiley & Sons, Inc.
Duda, R. O., & Hart, P. E. (1973). Pattern classification and scene analysis . New York.: John Wiley & Sons.
Berkes P, Orbán G, Lengyel M , and Fiser J. (2011) Spontaneous Cortical Activity Reveals Hallmarks of an Optimal Internal Model of the Environment. Science 7 January 2011: 331 (6013), 83-87. [DOI:10.1126/science.1195870]
Golden, R. (1988). A unified framework for connectionist systems. Biological Cybernetics, 59, 109-120.
Kersten, D. and P.W. Schrater (2000), *Pattern Inference Theory: A Probabilistic Approach to Vision*, in *Perception and the Physical World*, R. Mausfeld and D. Heyer, Editors. , John Wiley & Sons, Ltd.: Chichester. (pdf)
Kersten, D., Mamassian P & Yuille A (2004) Object perception as Bayesian inference. Annual Review of Psychology. (pdf, http://arjournals.annualreviews.org/doi/pdf/10.1146/annurev.psych.55.090902.142005)
Knill, D. C., & Richards, W. (1996). *Perception as Bayesian Inference*. Cambridge: Cambridge University Press.
Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge, UK: Cambridge University Press.
Van Trees, H. L. (1968). Detection, Estimation and Modulation Theory . New York: John Wiley and Sons.