Introduction to Neural Networks
Daniel Kersten
Belief Propagation

Initialize

In[132]:=
```
Off[General::spell1];
Needs["ErrorBarPlots`"]
```

Last time

Bayesian inference and conjugate priors

Graphical models

(Bias/variance trade-off and over-fitting)

Today

Bayesian learning of the mean of a gaussian

Belief propagation on a graph

Final projects

Bayesian learning of univariate Gaussian mean: MAP

We'll spend more time on unsupervised learning later, but it is useful to see how to apply what we've learned about to one of the simplest applications of Bayes to unsupervised learning.

From a statistical point of view, one form of unsupervised learning is "density estimation" which can be done (non-parametrically) from histogram measurements. In high dimensions this is hard, but is easier if we have a low-dimensional parametric model for the density--i.e. the density is modeled in terms of a few parameters. So for example, the 1D Gaussian could be approximated by a huge list of numbers ("statistics")--a frequency of occurrence for each bin with a range Δx, each number representing an estimate of the probability of the value of the random variable falling in that bin. But because it is Gaussian, we can be more efficient by representing the density in terms of just two numbers (also "statistics", but just the mean and variance), and a formula. In this context, learning becomes *parameter estimation*.

A Bayesian learning example: Suppose we believe the data comes from a Gaussian generative process, but we don't know the mean.

This example follows the same logic as the coin example in the previous lecture, except now we work with continuous random variables, and gaussian rather than the binomial distribution.

For simplicity, suppose we have a set of samples that come from a Gaussian distribution with known variance σ^2, but unknown mean μ.

x_i = noise, where noise ~ $N[\mu, \sigma]$, or equivalently

$x_i = \mu +$ noise, where noise ~ $N[0, \sigma]$

```
In[57]:= ndist0 = NormalDistribution[μ, σ];
```

This is the forward model. We now want to do “inverse probability”, i.e. estimate the mean and its distribution based on prior assumptions and data.

So lets treat the unknown mean, μ as a random variable, and assume a Gaussian prior on it:

$p(\mu)$, with $\mu \sim N[\mu_0, \sigma_0]$

```
In[60]:= ndistμ = NormalDistribution[μ0, σ0];
         PDF[ndistμ, μ]
```

Out[61]=

$$\frac{e^{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}}}{\sqrt{2\pi}\,\sigma_0}$$

We can think of $\mu0$ as an initial guess of the mean's mean, with standard deviation ($\sigma0$). But we are willing to change our estimate of the mean given new data. If we are really uncertain at the beginning,, we can start of assuming a large standard deviation σ_0, and as we gather data, the uncertainty about the value of the mean will decrease.

Suppose the generative model $N[\mu, \sigma]$ produces three i.i.d. (independent, identically distributed) samples x_1, x_2, x_3.What is the MAP estimate of μ? Which value of μ makes the posterior biggest? We use Bayes rule:

$$p(\mu \mid x_1, x_2, x_3) = \frac{p(x_1, x_2, x_3 \mid \mu)\, p(\mu)}{p(x_1, x_2, x_3)}$$

$p(x_1 \mid \mu)$ is given by :

```
PDF[ndist0, x1]
```

$$\frac{e^{-\frac{(-\mu+x_1)^2}{2\sigma^2}}}{\sqrt{2\pi}\,\sigma}$$

Because the samples are drawn independently, the $p(x_1, x_2, x_3 \mid \mu)$ is the product of three terms, so the numerator is $p(x_1 \mid \mu)\, p(x_2 \mid \mu)\, p(x_3 \mid \mu)$ times the prior $p(\mu)$:

```
PDF[ndist0, x1] * PDF[ndist0, x2] * PDF[ndist0, x3] * PDF[ndistμ, μ]
```

$$\frac{e^{-\frac{(-\mu+x_1)^2}{2\sigma^2}-\frac{(-\mu+x_2)^2}{2\sigma^2}-\frac{(-\mu+x_3)^2}{2\sigma^2}-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}}}{4\pi^2\sigma^3\sigma_0}$$

Calculating the MAP estimate of mean

To find the value of the mean that is largest given our three samples, and our prior assumption, we need to find μ where $p(x_1, x_2, x_3 \mid \mu)\, p(\mu)$ is biggest. This is the same as finding the value where the log of $p(x_1, x_2, x_3 \mid \mu)\, p(\mu)$ is biggest--a simpler expression. And with calculus, we can do that by finding where the slope of the derivative of the log with respect to μ is zero.

```
In[62]:= g = PDF[ndist0, x1] * PDF[ndist0, x2] * PDF[ndist0, x3] * PDF[ndistμ, μ];
         t = Log[g]
```

Out[63]=
$$\text{Log}\left[\frac{e^{-\frac{(-\mu+x_1)^2}{2\sigma^2}-\frac{(-\mu+x_2)^2}{2\sigma^2}-\frac{(-\mu+x_3)^2}{2\sigma^2}-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}}}{4\pi^2\sigma^3\sigma_0}\right]$$

```
In[64]:= t1 = PowerExpand[t]
```

Out[64]=
$$-2\,\text{Log}[2]-2\,\text{Log}[\pi]-3\,\text{Log}[\sigma]-\text{Log}[\sigma_0]-\frac{(-\mu+x_1)^2}{2\sigma^2}-\frac{(-\mu+x_2)^2}{2\sigma^2}-\frac{(-\mu+x_3)^2}{2\sigma^2}-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}$$

```
In[65]:= t2 = D[t1, μ]
```

Out[65]=
$$\frac{-\mu+x_1}{\sigma^2}+\frac{-\mu+x_2}{\sigma^2}+\frac{-\mu+x_3}{\sigma^2}-\frac{(\mu-\mu_0)\left(1-\text{Subscript}^{(1,0)}[\mu,0]\right)}{\sigma_0^2}$$

...Mathematica is having problems with subscripts, so get rid of them:

In[67]:=
$$\text{Solve}\left[-\frac{-\mu+x_1}{\sigma^2}+\frac{-\mu+x_2}{\sigma^2}+\frac{-\mu+x_3}{\sigma^2}-\frac{(\mu-\mu0)}{\sigma0^2}==0,\ \mu\right]$$

Out[67]=
$$\left\{\left\{\mu\to\frac{\mu0\,\sigma^2-\sigma0^2\,x_1+\sigma0^2\,x_2+\sigma0^2\,x_3}{\sigma^2+\sigma0^2}\right\}\right\}$$

Divide the above expression by $\sigma0^2\,\sigma^2$.

In[68]:=
$$\left\{\left\{\mu\to\frac{\frac{\mu0}{\sigma0^2}+\frac{1}{\sigma^2}(x_1+x_2+x_3)}{\frac{3}{\sigma^2}+\frac{1}{\sigma0^2}}\right\}\right\}$$

Out[68]=
$$\left\{\left\{\mu\to\frac{\frac{\mu0}{\sigma0^2}+\frac{x_1+x_2+x_3}{\sigma^2}}{\frac{3}{\sigma^2}+\frac{1}{\sigma0^2}}\right\}\right\}$$

We've done the calculation with just three data points, but one can see the pattern. So in general, μ can be estimated from n samples in batch mode by the following rule:

$$\left\{\left\{\mu\to\frac{\frac{\mu0}{\sigma0^2}+\frac{1}{\sigma^2}\sum_{i=1}^{n}x_i}{\frac{n}{\sigma^2}+\frac{1}{\sigma0^2}}\right\}\right\}$$

With a bit more effort, one can also derive the solution to estimate the standard deviation too, when it is unknown.

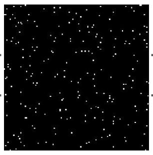
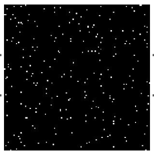
So to sum up, we have a method to estimate the probability distribution of the mean given the data. (For the multi-variate case, cf. Duda and Hart.)

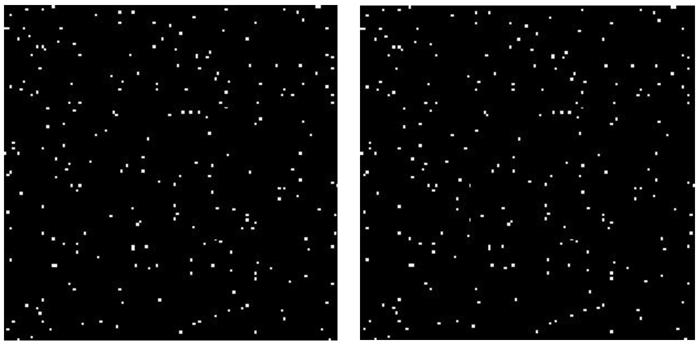
► 1. What is the influence of the initial estimate of the mean as learning goes on? What is the estimate of the mean as n gets large?

► 2. Formulate the estimation process as a sequential rule where the posterior gets iteratively updated as each new data point arrives.

Interpolation and perceptual surface completion revisited

Recall this previous example of your visual system interpolating a smooth surface from sparse data (cf. Nakayama and Shimojo, 1992). The data is sparse because the information about depth comes from the disparities in the left and right vertical edges of the horizontal rectangle. The random dot stereogram illustrated here is “dense” rather than sparse. The data is dense because there were lots of potential features to match throughout the background and the square that floated out in depth. Here’s the example of a sparse random dot stereogram seen earlier:

```
GraphicsRow[{Image[
```

```
], Image[
```

```
]}]
```



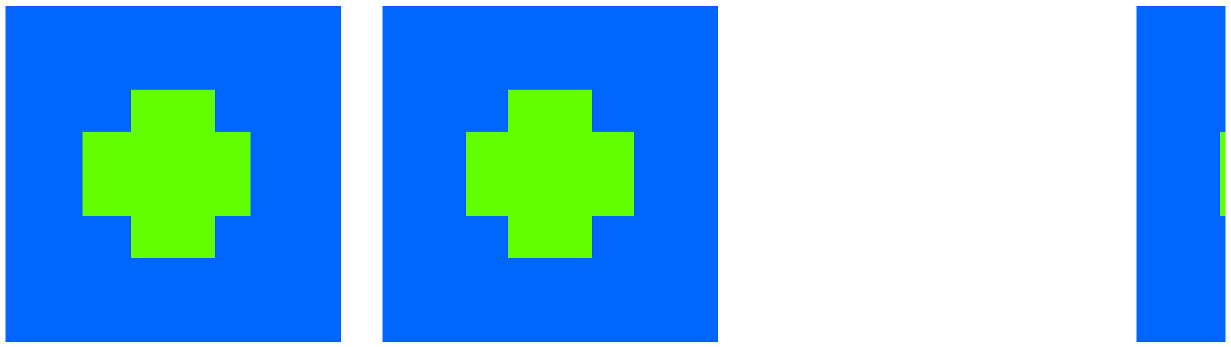
One sees the white points on dark, central surface floating in front of a background surface, also with a white point texture. How can one model processes of interpolation? Earlier we approached the problem by constructing a cost function to be minimized.

Here’s another demonstration. When you view the left and right images below in stereo, you may see a horizontal rectangle floating out in front of the vertical rectangle in the back:

In[69]:=
```
backwidth = 50; backheight = 50;
{x0, y0, vwidth, vheight, vxoff, vyoff, hwidth, hheight, hxoff, hyoff} =
```
$$\left\{\frac{backwidth}{2}, \frac{backheight}{2}, \frac{backwidth}{4}, \frac{backheight}{8}, \frac{backwidth}{2}, \frac{backheight}{2}, \frac{backwidth}{8}, \frac{backheight}{4}, \frac{backwidth}{2}, \frac{backheight}{2}\right\};$$

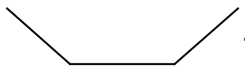
```
In[71]:= gleft =
           Show[Graphics[{Hue[0.6`], Rectangle[{0, 0}, {backwidth, backheight}], Hue[0.27`],
               Rectangle[{x0 - vwidth, y0 - vheight}, {x0 + vwidth, y0 + vheight}], Hue[0.27`],
               Rectangle[{x0 - hwidth, y0 - hheight}, {x0 + hwidth, y0 + hheight}]}],
             AspectRatio → Automatic];
         gright = Show[Graphics[{Hue[0.6`], Rectangle[{0, 0}, {backwidth, backheight}],
               Hue[0.27`], Rectangle[{x0 - vwidth - 1, y0 - vheight},
                 {x0 + vwidth - 1, y0 + vheight}], Hue[0.27`], Rectangle[{x0 - hwidth, y0 - hheight},
                 {x0 + hwidth, y0 + hheight}]}], AspectRatio → Automatic];
         gappearance = Show[Graphics[{Hue[0.6`], Rectangle[{0, 0}, {backwidth, backheight}],
               Hue[0.42`], Rectangle[{x0 - hwidth, y0 - hheight}, {x0 + hwidth, y0 + hheight}],
               Hue[0.27`], Rectangle[{x0 - vwidth, y0 - vheight}, {x0 + vwidth, y0 + vheight}]}],
             AspectRatio → Automatic, ImageSize → Small];
         Show[GraphicsRow[{gright, gleft, "", gappearance}]]
```

Out[74]=



If you can cross your eyes, so that the left image is in the right eye, and the right image in the left, you will see a green horizontal bar floating out in front of a green vertical bar. So-called "free-fusing" isn't easy, but when you've got it, you should see a total of three green crosses. The one in the middle is the one in which the two images are fused by your brain--and this is the one we are talking about. The interesting point here is that even though there is no local information in the image to support the percept of a horizontal occluding bar, observers still see an illusory completion.

It looks a bit like the figure on the far right, except that the color of the horizontal bar is changed here slightly just for illustration.

▶ 3. Why doesn't the cross appear to look more like wings? I.e. if seen from below,  .

One's first guess might be that observers should not see the horizontal bar as a plane, but as shown in the figure, rather they would interpolate a surface that on the left side is close to the viewer, but then descends back towards the depth of the vertical bar, stays at that constant far depth left to right, and then comes back towards the viewer on the right (see Nakayama, K., & Shimojo, S., 1992.) This suggests a Bayesian prior that captures long-range, or global constraints, in addition to the local ones studied in this lecture.

Below we first review how to solve the interpolation problem using standard gradient descent on a cost function, and then re-formulate the problem on a graphical model to introduce Bayesian Belief Propagation.

Review of interpolation using smoothness: Gradient descent

For simplicity, we'll assume 1-D as in the lecture on sculpting the energy function. In anticipation of formulating the problem in terms of a graph that represents conditional probability dependence, we represent *observable* depth cues by *y**, and the true ("*hidden*") depth estimates by y.

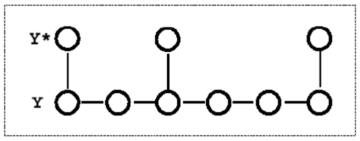


Figure from Weiss (1999).

We want to estimate the underlying function using only a *local* smoothness constraint. This is in contrast to linear regression, which assumes fitting the data with a straight line--a *global* constraint.

First-order smoothness

Earlier we saw that under specific assumptions, biologically plausible neural updating can be seen to decrease the value of an energy or cost function. One can also start off with an assumed cost function, determined by a set of constraints, and use gradient descent to derive an update rule that minimizes the cost.

For an interpolation problem, we can write the energy or cost function by:

$$J(Y) = \sum_k w_k (y_k - y_k^*)^2 + \lambda \sum_i (y_i - y_{i+1})^2$$

where w_k(= xs[[k]] below) is an "indicator function", and y_k^*= d, are the data values. The indicator function is 1 if where is data available, and zero otherwise. The second sum represents the sum of the squared differences between neighboring y-values. Minimizing the first sum encourages the estimates of y to be close to the measured values. Minimizing the second sum encourages nearby y-values to be the same. Thus minimizing J(Y) encourages fidelity to the data where present, and similarity to nearby values where there is no data.

Gradient descent gives the following local update rule:

$$y_k \leftarrow y_k + \eta_k (\lambda(\frac{y_{k-1} + y_{k+1}}{2} - y_k) + w_k(y_k^* - y_k))$$

As before, λ is a free parameter that controls the degree of smoothness, i.e. smoothness at the expense of fidelity to the data.

At each step, the rule encourages the estimates to get closer to the data (where it exists), and also closer to the values of neighbors.

There are various choices for how to change the smoothness as a function of iterations.

E.g. standard methods in numerical analysis include:
Gauss-Seidel: η[k_] :=1 /(λ+xs[[k]]). And successive over-relaxation (SOR): η2[k_]:=1.9/(λ+xs[[k]]);

A simulation: Straight line with random missing data points

We look at an example where we have only two data points, both at the boundaries.

Make the data

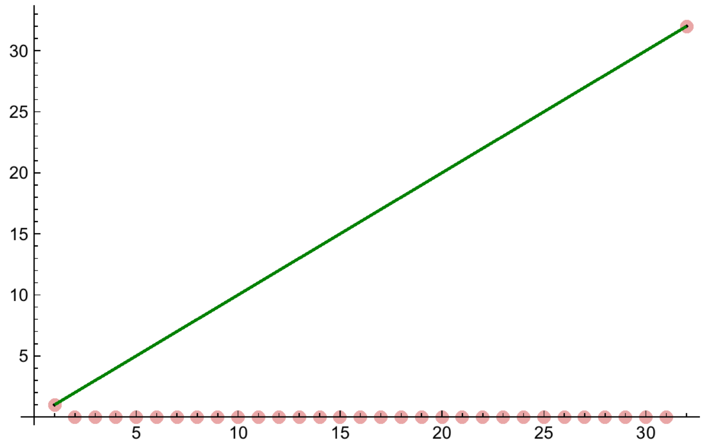
Consider the problem of interpolating a set of points with missing data, marked by an indicator function with the following notation:
w_k= **xs[[k]], y* = data, y=f.**

We'll assume the true model is that f = y = j, where j=1 to size. data is a function of the sampling process on f = j

```
In[75]:= size = 32;
xs = Table[0, {i, 1, size}];
xs⟦1⟧ = 1;
xs⟦size⟧ = 1;
data = Table[N[j] xs⟦j⟧, {j, 1, size}];
g3 = ListPlot[Table[N[j], {j, 1, size}],
   Joined → True, PlotStyle → {RGBColor[0, 0.5, 0]}];
g2 = ListPlot[data, Joined → False,
   PlotStyle → {Opacity[0.35], RGBColor[0.75, 0., 0], PointSize[Large]}];
```

The green line shows the a straight line connecting the data points. The red dots on the abscissa mark the points where data are missing.

```
Show[{g2, g3}, ImageSize → Medium]
```



The update rule is linear, so we can represent in terms of two matrices, **Tm** and **Sm** such that the gradient of the energy is equal to:
Tm . f - Sm . data. As before, **Sm** is the filter to exclude non-data points specified by the indicator function w_k= **xs[[k]]** . **Tm** expresses the "smoothness" constraint.

```
In[76]:= Sm = DiagonalMatrix[xs];
         Tm = Table[0,{i,1,size},{j,1,size}];
         For[i=1,i<=size,i++,Tm[[i,i]] = 2];
         Tm[[1,1]]=1;Tm[[size,size]]=1; (*Adjust for the boundaries*)
         For[i=1,i<size,i++, Tm[[i+1,i]] = -1];
         For[i=1,i<size,i++, Tm[[i,i+1]] = -1];
```

Run gradient descent

```
In[82]:= Clear[f, d, λ]
         (λ * Tm.Array[f, size] - Sm.((Array[d, size]) - Array[f, size])) // MatrixForm ;

In[84]:= Clear[Tf,f1,j];
         dt = 1; λ=2;
         Tf[f1_] := f1 - dt*(1/(λ+xs))*(Tm.f1 - λ*Sm.(data-f1));
```

We will initialize the state vector to zero, and then run the network for **iter** iterations:

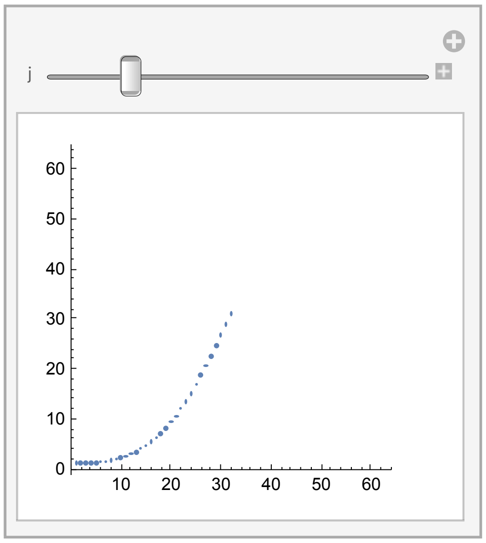
```
In[87]:= f0 = Table[0,{i,1,size}];
         (*f0 = Table[RandomReal[{0,30}],{i,1,size}];*)
         result=f0;
         f=f0;
         iter=25;
```

Now update the result after each iteration, and plot the interpolated function with each iteration step.

```
In[92]:= result = NestList[Tf, f, 30 * iter];

In[93]:= Manipulate[
          g1 = ListPlot[result[[j]], Joined → False, AspectRatio → Automatic,
            PlotRange → {{0, size}, {-1, size + 1}}, ImageSize → Small],
          {j, 1, Dimensions[result][[1]], 1}]
```

Out[93]=



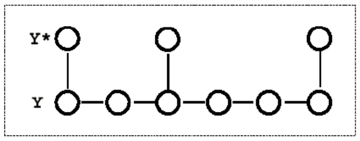
► 4. Try starting with f = random values. Try various numbers of iterations.

▶ 5. Try different sampling functions xs[[i]].

Interpolation using Belief Propagation

Same interpolation problem, but now using belief propagation

Example is taken from Yair Weiss. (Weiss, 1999). The data variables are on top, and the hidden variables to be estimated at the bottom of the graph.



Probabilistic generative model

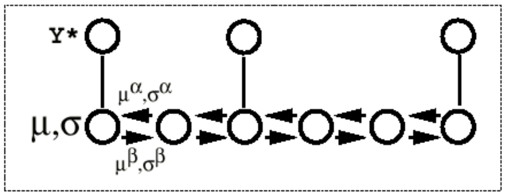
```
data[[i]] = y*[i] = xs[[i]] y[[i]] + dnoise,  dnoise~N[0, σD]
y[[i + 1]] = y[[i]] + znoise,  znoise~N[0, σR]
```

The first term is the "data formation" model, i.e. how the data is directly influenced by the interaction of the underlying influences or causes:

y* is determined by an underlying hidden “y” which can’t be directly measured. But we assume we can measure y*, which is determined by sampling some values of y and adding noise.

The second term reflects our prior assumptions about the smoothness of y, i.e. nearby y's are correlated, and in fact assumed identical except for some added noise. So with no noise the prior reflects the assumption that lines are horizontal--all y's are the same. This is sometimes called a “soft constraint”, because it is a tendency--we don’t insist that it be satisfied exactly.

Summary of the message passing rules



We'd like to know the distribution of the random variables at each node i, conditioned on all the data: I.e. we want the posterior

p(y_i=u | all the data)

If we could find this, we'd be able to: *1) say what the most probable value of y is at each node, and 2) give a measure of confidence.*

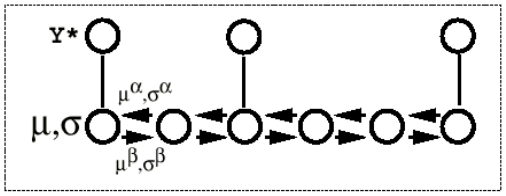
The appendix derives a solution, based on belief propagation, which sequentially updates local conditional distributions for the estimates of the mean of Y at each point, as well as estimating its standard deviation.

Let’s go over the rules of message passing. We follow Weiss, and make a (hopefully not too confusing) notation change to avoid the square superscripts, using the notation substitution:

$$\sigma_D^2 \to \sigma_D,\ \sigma_R^2 \to \sigma_R.$$

So σ_D represents the variability or uncertainty in the data. If σ_D is small, we trust the data Y^*. σ_R represents our prior belief
in the “roughness” of the curve--the Y’s--to be estimated. If σ_R is big, then we tolerate more deviations from straightness, while small values would bias the estimates towards smooth, straight lines. In other words, if the variance of the difference between neighboring estimates is small, then they would tend to have nearly the same values.

Let’s look at the messages passed for our example:



The Appendix derives a rule that tells us how to update the mean and variance parameters of the previous node, going right to left:

$$\mu\alpha \leftarrow \frac{\mu\alpha_p\,\sigma_D^2 + y_p\,\sigma\alpha_p^2}{\sigma_D^2 + \sigma\alpha_p^2} = \frac{\frac{\mu\alpha_p\,\sigma_D^2}{\sigma\alpha_p^2\,\sigma_D^2} + \frac{y_p\,\sigma\alpha_p^2}{\sigma\alpha_p^2\,\sigma_D^2}}{\frac{\sigma_D^2}{\sigma\alpha_p^2\,\sigma_D^2} + \frac{\sigma\alpha_p^2}{\sigma\alpha_p^2\,\sigma_D^2}} = \frac{\frac{\mu\alpha_p}{\sigma\alpha_p^2} + \frac{y_p}{\sigma_D^2}}{\frac{1}{\sigma\alpha_p^2} + \frac{1}{\sigma\alpha_p^2}}$$

The update rule for the variance is:

$$\sigma\alpha^2 \leftarrow \sigma_R^2 + \frac{1}{\frac{1}{\sigma_D^2} + \frac{1}{\sigma\alpha_p^2}}$$

And by symmetry, we have a similar pair of update rules for the messages $\mu\beta$ and $\sigma\beta^2$ passed from left to right (see Appendix).

Recall that sometimes we have data and sometimes we don’t. So replace:

$$y_p \to w_{i-1}\, y_{i-1}^*$$

- Side note: The underlying operation in which estimates get combined weighted by uncertainty, is similar to how we updated the mean in the Bayesian learning example. This is a recurring theme. In your assignment, you show that the maximum a posteriori estimate for cue combination is:

$$\mu = \frac{1/\sigma_1^2}{1/\sigma_1^2 + 1/\sigma_2^2}\,\mu_1 + \frac{1/\sigma_2^2}{1/\sigma_1^2 + 1/\sigma_2^2}\,\mu_2 = \frac{\mu_1/\sigma_1^2 + \mu_2/\sigma_2^2}{1/\sigma_1^2 + 1/\sigma_2^2}$$

  where $r_i = 1/\sigma_i^2$, and μ_1 and μ_2 are the estimates of the means obtained separately. How this could be done neurally has inspired recent work in neural population codes, mentioned below. Combining information weighted by reliability underlies a number of models of visual and multimodal cue integration in psychophysics over the past decade.

A simulation: Belief propagation for interpolation with missing data

Here’s an implementation of the Bayesian belief updates applied to the above interpolation problem. We change some of the notation for the program, with equivalent symbols:

$y_p \to = w_{i-1}\, y^*_{i-1}$ = **xs**[i − 1] **data**[i − 1]

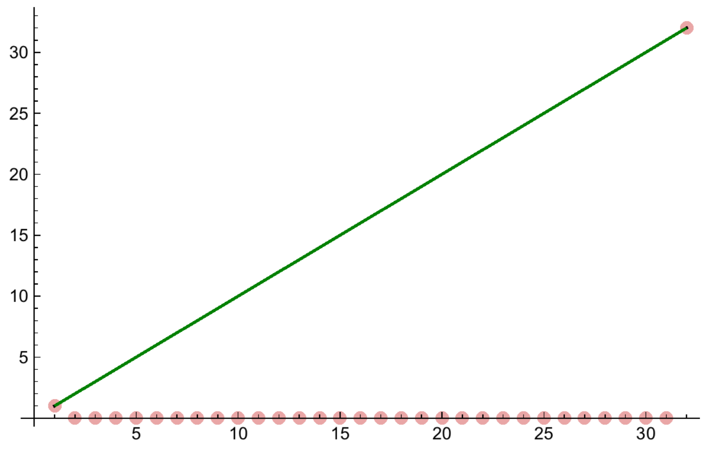
y → **yfit**

```
In[134]:= size = 32;
xs = Table[0, {i, 1, size}];
xs[[1]] = 1;
xs[[size]] = 1;
data = Table[N[j] xs[[j]], {j, 1, size}];
g3bp = ListPlot[Table[N[j], {j, 1, size}],
   Joined → True, PlotStyle → {RGBColor[0, 0.5, 0]}];
g2bp = ListPlot[data, Joined → False,
   PlotStyle → {Opacity[0.35], RGBColor[0.75, 0., 0], PointSize[Large]}];
```

The green line shows the a straight line connecting the data points. The red dots on the abscissa mark the points where data are missing.

```
In[135]:= Show[{g2bp, g3bp}, ImageSize → Medium]
```

Out[135]=



Initialization

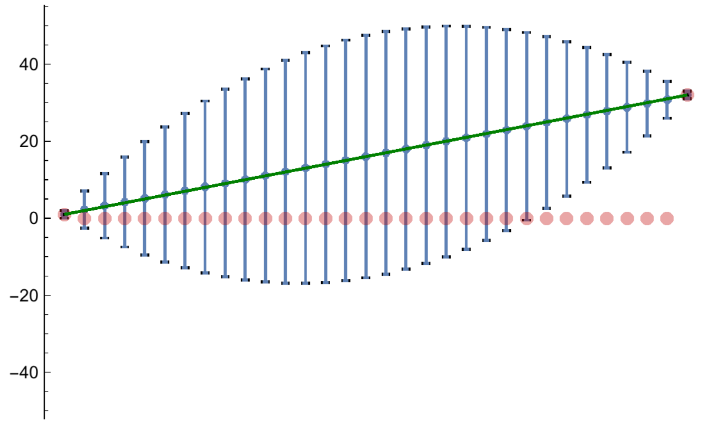
```
In[136]:= size = 32;
μ0 = 1;
μα = 1; σα = 100 000; (*large uncertainty *)
μβ = 1; σβ = 100 000; (*large*)
σR = 4.0; σD = 1.0;
μ = Table[μ0, {i, 1, size}];
σ = Table[σα, {i, 1, size}];
μα = Table[μ0, {i, 1, size}];
σα = Table[σα, {i, 1, size}];
μβ = Table[μ0, {i, 1, size}];
σβ = Table[σβ, {i, 1, size}];
iter = 0;
i = 1;
j = size;
```

The code below implements the above iterative equations, taking care near the boundaries. The plot shows the estimates of y_i= μ, and the error bars show $\pm\sigma_i$.

Belief Propagation Routine

```
In[149]:= yfit = Table[{0, 0}, {i1, 1, size}];
g1b = ErrorListPlot[{yfit}];
Dynamic[Show[
  {g1b, g2bp, g3bp, Graphics[{Text["Iteration=" <> ToString[iter], {size/2, size}]}]},
  PlotRange → {-50, 50}, Axes → {False, True}]]
```

Out[151]=



Execute the next cell to run 31 iterations. The display is slowed down so that you can see the progression of the updates in the above graph.

```
In[152]:= Do[
  Pause[.5];
  μ⟦i⟧ = (xs⟦i⟧ data⟦i⟧/σD + μα⟦i⟧/σα⟦i⟧ + 1. μβ⟦i⟧/σβ⟦i⟧)/(xs⟦i⟧/σD + 1/σα⟦i⟧ + 1/σβ⟦i⟧);
 σ⟦i⟧ = 1./(xs⟦i⟧/σD + 1/σα⟦i⟧ + 1/σβ⟦i⟧);
 μ⟦j⟧ = (xs⟦j⟧ data⟦j⟧/σD + μα⟦j⟧/σα⟦j⟧ + 1. μβ⟦j⟧/σβ⟦j⟧)/(xs⟦j⟧/σD + 1/σα⟦j⟧ + 1/σβ⟦j⟧);
 σ⟦j⟧ = 1./(xs⟦j⟧/σD + 1/σα⟦j⟧ + 1/σβ⟦j⟧);
 nextj = j - 1;
 μα⟦nextj⟧ = (xs⟦j⟧ data⟦j⟧/σD + 1. μα⟦j⟧/σα⟦j⟧)/(xs⟦j⟧/σD + 1/σα⟦j⟧);
 σα⟦nextj⟧ = σR + 1./(xs⟦j⟧/σD + 1/σα⟦j⟧);
 nexti = i + 1;
 μβ⟦nexti⟧ = (xs⟦i⟧ data⟦i⟧/σD + 1. μβ⟦i⟧/σβ⟦i⟧)/(xs⟦i⟧/σD + 1/σβ⟦i⟧);
 σβ⟦nexti⟧ = σR + 1./(xs⟦i⟧/σD + 1/σβ⟦i⟧);
 j--;
 i++;
 iter++;
 yfit = Table[{μ⟦i1⟧, σ⟦i1⟧}, {i1, 1, size}];
 g1b = ErrorListPlot[{yfit}];
  , {size - 1}];
```

Relation to neural networks

We've seen two very different ways of estimating states through iterative updating. It is easy to see how the first, derived from gradient descent, is related to updating in a traditional neural network. The second, belief propagation, gives us a very different view of how information might be represented and updated in a network. For example, in the second case applied to interpolation, the nodes represent probability distributions over depths (summarized by the mean and variance) not just estimates of depth. A critical new feature is the explicit representation of uncertainty, i.e. in the standard deviations of the node values. Later in the course, we will discuss whether the brain might have explicit representations of such uncertainties, how these might be represented in populations of neurons, and how the brain might do computations on these. Does the brain do belief propagation, and if so how?

For a preview of the general question of whether the brain represents and computes on distributions, see: Ma et al. (2006, 2012), Knill & Pouget (2004), Zemel & Pouget (1998).

Discussion: relation to behavior?

Yair Weiss and colleagues showed how the probablistic combination of local motion cues and prior motion assumptions could explain a number of visual illusions: Weiss, Y., Simoncelli, E. P., & Adelson, E. H. (2002). Motion illusions as optimal percepts. Nature Neuroscience.

But their conclusion (and probably all other similar applications to perceptual behavior) didn’t depend on the algorithm.

Can you think of an application and test of belief propagation to perception?

More examples and exercises

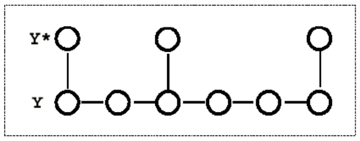
Final projects

See guides on class web page. Outline due November 21.

Appendix

Derivation of the update rules, called “message passing”

Updating the mean and variance given the data at point i, and current beliefs about mean and variance before and after i



In[1]:=

Let p(Y_i=u | all the data) be normally distributed: NormalDistribution[μi,σi].

Consider the ith unit. The posterior p(Y_i=u | all the data) =

p(Y_i=u| all the data) ∝ p(Y_i=u | data before i) p(data at i | Y_i=u) p(Y_i=u | data after i)

(“before” and “after” means to the left and right of i, respectively.)

Suppose that p(Y_i=u | data before i) is also gaussian:

p(Y_i=u | data before i) = α_i[u] ~ NormalDistribution[$\mu\alpha$, $\sigma\alpha$]

and so is probability conditioned on the data after i:

p(Y_i=u | data after i)= β_i[u] ~ NormalDistribution[$\mu\beta$, $\sigma\beta$]

And the noise model for the data:

p(data at i | Y_i=u) = L_i[u]~ `NormalDistribution[yp, σD]`

yp=data[[i]]

So in terms of these functions, the posterior probability of the ith unit taking on the value u can be expressed as proportional to a product of the three factors:

p(Y_i=u | all the data) ∝ α_i[u]*L_i[u]*β_i[u]

```
In[37]:= ClearAll[μα, σα, μβ, σβ, yp, σD];

αudist = NormalDistribution[μα, σα];
α[u] = PDF[αudist, u];

Ddist = NormalDistribution[yp, σD];
L[u] = PDF[Ddist, u];

βudist = NormalDistribution[μβ, σβ];
β[u] = PDF[βudist, u];

α[u] * L[u] * β[u]
```

Out[44]=
$$\frac{e^{-\frac{(u-yp)^2}{2\,\sigma D^2}-\frac{(u-\mu\alpha)^2}{2\,\sigma\alpha^2}-\frac{(u-\mu\beta)^2}{2\,\sigma\beta^2}}}{2\sqrt{2}\,\pi^{3/2}\,\sigma D\,\sigma\alpha\,\sigma\beta}$$

This just another gaussian distribution on Y_i=u. What is its mean and variance? Finding the root enables us to complete the square to see what the numerator looks like. In particular, what the mode (=mean for gaussian) is.

In[45]:=
$$\text{Solve}\left[-D\left[-\frac{(u-\mu\alpha)^2}{2\,\sigma\alpha^2}-\frac{(u-\mu\beta)^2}{2\,\sigma\beta^2}-\frac{(u-y_p)^2}{2\,\sigma_D^2},\,u\right]==0,\,u\right]$$

Out[45]=
$$\left\{\left\{u\to\frac{\frac{\mu\alpha}{\sigma\alpha^2}+\frac{\mu\beta}{\sigma\beta^2}+\frac{y_p}{\sigma_D^2}}{\frac{1}{\sigma\alpha^2}+\frac{1}{\sigma\beta^2}+\frac{1}{\sigma_D^2}}\right\}\right\}$$

This suggests that if we had estimates of $\mu\alpha$, $\mu\beta$, $\sigma\alpha^2$, $\sigma\beta^2$ and the data, we could update the mean of node i using:

$$u \leftarrow \frac{\frac{\mu\alpha}{\sigma\alpha^2}+\frac{\mu\beta}{\sigma\beta^2}+\frac{y_p}{\sigma_D^2}}{\frac{1}{\sigma\alpha^2}+\frac{1}{\sigma\beta^2}+\frac{1}{\sigma_D^2}}$$

Similarly, the update rule for the variance is:

$$\sigma^2 \leftarrow \frac{1}{\sigma\alpha^2} + \frac{1}{\sigma\beta^2} + \frac{1}{\sigma_D^2}$$

How do we get $\mu\alpha$, $\mu\beta$, $\sigma\alpha$, $\sigma\beta$?

We express the probability of the ith unit taking on the value **u** in terms of the values of the neighbor before, conditioning on what is known (the observed measurements), and marginalizing over what isn't (the previous "hidden" node value, **v,** at the i-1th location).

We have three terms to worry about that depend on nodes in the neighborhood preceding i:

$$\alpha[u] = \int_{-\infty}^{\infty} \alpha_p[v] * S[u] * L[v]\, dv \propto \int_{-\infty}^{\infty} e^{-\frac{(v-y_p)^2}{2\sigma_D^2} - \frac{(u-v)^2}{2\sigma_R^2} - \frac{(v-\mu\alpha_p)^2}{2\sigma\alpha_p^2}}\, dv$$

$\alpha_p = \alpha_{i-1}$. S[u] is our smoothing term, or transition probability : S[u] = p (u | v) .
L[v] is the likelihood of v given the data previous at the previous node.

In[46]:=
```
Rdist = NormalDistribution[v, σR];
S[u] = PDF[Rdist, u];

αvdist = NormalDistribution[μαp, σαp];
αp[v] = PDF[αvdist, v];

Lp[v] = PDF[Ddist, v];
```

In[51]:=
```
Integrate[αp[v] * S[u] * Lp[v], {v, -Infinity, Infinity}]
```

Out[51]=
$$\text{ConditionalExpression}\left[\frac{e^{-\frac{u^2\,\sigma D^2 + yp^2\,\sigma_R^2 + \mu\alpha_p^2\,(\sigma D^2+\sigma_R^2) - 2\,\mu\alpha_p\,(u\,\sigma D^2 + yp\,\sigma_R^2) + u^2\,\sigma\alpha_p^2 - 2\,u\,yp\,\sigma\alpha_p^2 + yp^2\,\sigma\alpha_p^2}{2\,(\sigma D^2\,\sigma\alpha_p^2 + \sigma_R^2\,(\sigma D^2 + \sigma\alpha_p^2))}}}{2\,\pi\,\sigma D\,\sigma_R\,\sqrt{\frac{1}{\sigma D^2} + \frac{1}{\sigma_R^2} + \frac{1}{\sigma\alpha_p^2}}\;\sigma\alpha_p},\ \text{Re}\left[\frac{1}{\sigma D^2} + \frac{1}{\sigma_R^2} + \frac{1}{\sigma\alpha_p^2}\right] \geq 0\right]$$

Some uninspired *Mathematica* manipulations

Let’s find an expression for the mode of the above calculated expression for α[u]

In[52]:=
```
D[-((u - μαp)^2 σD^2 + μαp^2 σR^2 + u^2 σαp^2 + yp^2 (σR^2 + σαp^2) - 2 yp (μαp σR^2 + u σαp^2)) /
    (2 (σR^2 σαp^2 + σD^2 (σR^2 + σαp^2))), u]
```

Out[52]=
$$-\frac{2\,(u - \mu\alpha_p)\,\sigma_D^2 + 2\,u\,\sigma\alpha_p^2 - 2\,y_p\,\sigma\alpha_p^2}{2\,(\sigma_R^2\,\sigma\alpha_p^2 + \sigma_D^2\,(\sigma_R^2 + \sigma\alpha_p^2))}$$

In[53]:=
```
Solve[-% == 0, u]
```

Out[53]=
$$\left\{\left\{u \to \frac{\frac{\mu\alpha_p\,\sigma_D^2}{\sigma_R^2\,\sigma\alpha_p^2 + \sigma_D^2\,(\sigma_R^2+\sigma\alpha_p^2)} + \frac{y_p\,\sigma\alpha_p^2}{\sigma_R^2\,\sigma\alpha_p^2 + \sigma_D^2\,(\sigma_R^2+\sigma\alpha_p^2)}}{\frac{\sigma_D^2}{\sigma_R^2\,\sigma\alpha_p^2 + \sigma_D^2\,(\sigma_R^2+\sigma\alpha_p^2)} + \frac{\sigma\alpha_p^2}{\sigma_R^2\,\sigma\alpha_p^2 + \sigma_D^2\,(\sigma_R^2+\sigma\alpha_p^2)}}\right\}\right\}$$

In[54]:= Simplify$\left[\left(\frac{\mu\alpha_p\,\sigma_D^2}{\sigma_R^2\,\sigma\alpha_p^2+\sigma_D^2\,(\sigma_R^2+\sigma\alpha_p^2)}+\frac{y_p\,\sigma\alpha_p^2}{\sigma_R^2\,\sigma\alpha_p^2+\sigma_D^2\,(\sigma_R^2+\sigma\alpha_p^2)}\right)\Big/\left(\sigma_D^2 * \sigma\alpha_p^2\right)\right]$

Out[54]= $\frac{\mu\alpha_p\,\sigma_D^2+y_p\,\sigma\alpha_p^2}{\sigma_D^2\,\sigma_R^2\,\sigma\alpha_p^4+\sigma_D^4\,\sigma\alpha_p^2\,(\sigma_R^2+\sigma\alpha_p^2)}$

In[55]:= Simplify$\left[\left(\frac{\sigma_D^2}{\sigma_R^2\,\sigma\alpha_p^2+\sigma_D^2\,(\sigma_R^2+\sigma\alpha_p^2)}+\frac{\sigma\alpha_p^2}{\sigma_R^2\,\sigma\alpha_p^2+\sigma_D^2\,(\sigma_R^2+\sigma\alpha_p^2)}\right)\Big/\left(\sigma_D^2 * \sigma\alpha_p^2\right)\right]$

Out[55]= $\frac{\sigma_D^2+\sigma\alpha_p^2}{\sigma_D^2\,\sigma_R^2\,\sigma\alpha_p^4+\sigma_D^4\,\sigma\alpha_p^2\,(\sigma_R^2+\sigma\alpha_p^2)}$

In[56]:= $\left(\frac{\mu\alpha_p\,\sigma_D^2+y_p\,\sigma\alpha_p^2}{\sigma_D^2\,\sigma_R^2\,\sigma\alpha_p^4+\sigma_D^4\,\sigma\alpha_p^2\,(\sigma_R^2+\sigma\alpha_p^2)}\right)\Big/\left(\frac{\sigma_D^2+\sigma\alpha_p^2}{\sigma_D^2\,\sigma_R^2\,\sigma\alpha_p^4+\sigma_D^4\,\sigma\alpha_p^2\,(\sigma_R^2+\sigma\alpha_p^2)}\right)$

Out[56]= $\frac{\mu\alpha_p\,\sigma_D^2+y_p\,\sigma\alpha_p^2}{\sigma_D^2+\sigma\alpha_p^2}$

We now have a rule that tells us how to update the α(u)=p(yi=u|data before i), in terms of the mean and variance parameters of the previous node:

$$\mu\alpha \leftarrow \frac{\mu\alpha_p\,\sigma_D^2+y_p\,\sigma\alpha_p^2}{\sigma_D^2+\sigma\alpha_p^2}=\frac{\frac{\mu\alpha_p\,\sigma_D^2}{\sigma\alpha_p^2\,\sigma_D^2}+\frac{y_p\,\sigma\alpha_p^2}{\sigma\alpha_p^2\,\sigma_D^2}}{\frac{\sigma_D^2}{\sigma\alpha_p^2\,\sigma_D^2}+\frac{\sigma\alpha_p^2}{\sigma\alpha_p^2\,\sigma_D^2}}=\frac{\frac{\mu\alpha_p}{\sigma\alpha_p^2}+\frac{y_p}{\sigma_D^2}}{\frac{1}{\sigma\alpha_p^2}+\frac{1}{\sigma\alpha_p^2}}$$

The update rule for the variance is:

$$\sigma\alpha^2 \leftarrow \sigma_R^2+\frac{1}{\frac{1}{\sigma_D^2}+\frac{1}{\sigma\alpha_p^2}}$$

A similar derivation gives us the rules for $\mu\beta$, $\sigma\beta^2$

$$\mu\beta \leftarrow \frac{\frac{\mu\beta_a}{\sigma\beta_a^2}+\frac{y_a}{\sigma_D^2}}{\frac{1}{\sigma\beta_a^2}+\frac{1}{\sigma\alpha_a^2}}$$

$$\sigma\beta^2 \leftarrow \sigma_R^2+\frac{1}{\frac{1}{\sigma_D^2}+\frac{1}{\sigma\beta_a^2}}$$

Where the subscript index p (for "previous", i.e. unit i-1) is replaced by a (for "after", i.e. unit i+1).

Recall that sometimes we have data and sometimes we don't. So replace:

$$y_p \to \texttt{xs[i-1] data[i-1]} = w_{i-1}\,y_{i-1}^*$$

And similarly for y_a.

The ratio, $\left(\frac{\sigma_D}{\sigma_R}\right)^2$plays the role of λ above. If σ_D^2>>σ_R^2, there is greater smoothing. If σ_D^2<<σ_R^2, there is more fidelity to the data. (Recall $y^* \to$ data.$w_k \to$ xs[[k]]). But now we have a principled way of assigning the relative amount of smoothing.

References

Applebaum, D. (1996). Probability and Information . Cambridge, UK: Cambridge University Press.
Frey, B. J. (1998). Graphical Models for Machine Learning and Digital Communication. Cambridge, Massachusetts: MIT Press.
Jepson, A., & Black, M. J. (1993). Mixture models for optical flow computation. Paper presented at the Proc. IEEE Conf. Comput. Vsion Pattern Recog., New York.
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., & 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., & Pouget, A. (2004). The Bayesian brain: the role of uncertainty in neural coding and computation. Trends in Neurosciences, 27(12), 712–719. doi:10.1016/j.tins.2004.10.007
Ma, W. J., Beck, J. M., Latham, P. E., & Pouget, A. (2006). Bayesian inference with probabilistic population codes. Nature Neuroscience, 9(11), 1432–1438. doi:10.1038/nn1790
Ma, W. J. (2012). Organizing probabilistic models of perception. Trends in Cognitive Sciences, 16(10), 511–518. doi:10.1016/j.tics.2012.08.010
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.
Nakayama, K., & Shimojo, S. (1992). Experiencing and perceiving visual surfaces. Science, 257(5075), 1357–1363. doi:10.1126/science.1529336
Pearl, Judea. (1997) Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference. (amazon.com link)
Ripley, B. D. (1996). Pattern Reco gnition and Neural Networks. Cambridge, UK: Cambridge University Press.
Weiss Y. (1999) Bayesian Belief Propagation for Image Understanding submitted to SCTV 1999. (gzipped postscript 297K)
Weiss, Y. (1997). Smoothness in Layers: Motion segmentation using nonparametric mixture estimation. Paper presented at the Proceedings of IEEE conference on Computer Vision and Pattern Recognition.
Weiss, Y., Simoncelli, E. P., & Adelson, E. H. (2002). Motion illusions as optimal percepts. Nature Neuroscience, 5(6), 598–604. doi:10.1038/nn858
Yuille, A., Coughlan J., Kersten D.(1998) (pdf)
Zemel, R. S. & Pouget, A. (1998). Probabilistic interpretation of population codes. Neural Computation, 10(2), 403–430.

For notes on Graphical Models, see:http : // www.cs.berkeley.edu/ ~ murphyk/Bayes/bayes.html