Introduction to Neural Networks

Sculpting energy/cost landscapes: interpolation and gradient descent

Initialization

```
Off[SetDelayed::write]
Off[General::spell1]
```

Overview

Last time

Probability, statistics overview
Simple sampling

Mathematica functions for multivariate distributions

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 graphically illustrate some of the above ideas, and help build intuitions.

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:

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

where Σ^{-1}plays the role, in n dimensions, of the reciprocal of variance $\left(\sigma^2\right)^{-1}=\left(1/\sigma^2\right)$ in one dimension. $(x-\mu)^T\,\Sigma^{-1}\,(x-\mu)$ is called the Mahalanobis_distance of x from μ. "Distance" in the sense that if the mean, μ, is a zero vector, you can think of $x^T\Sigma^{-1}x$ as a measure of how far x is away from zero in "units of standard deviation" in that direction.

The covariance matrix Σ can't be any matrix, it has to be symmetric and "positive definite". The latter means that $x^T\Sigma x > 0$ for any vector x consisting of real numbers (and for which no columns of Σ are zero). Suppose for the moment that Σ is a scalar, not a matrix. If $x^2\,\Sigma > 0$ for all x, then Σ would also have to be positive, i.e. $\Sigma > 0$. Positive definite is a generalization to matrices. In general $x^T\Sigma x$ is a sum of linear plus quadratic terms. In general, such a sum can define a planar surface (if all the squared and cross-product terms are zero), a valley or ridge, or a convex mound, or a concave bowl. In other words, in 2D want a "bowl", and as a consequence, p(x) will be a convex mound. This is important for the existence and uniqueness of a mode. Most sampled covariance matrices will be positive definite.

Mathematica has a built-in function, MultinormalDistribution[μ, Σ]. 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.

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

You can ask for the Covariance to be returned:

```
    Covariance[MultinormalDistribution[{μ1, μ2}, Σ]] // MatrixForm
```

$$\begin{pmatrix} \sigma_{11}^2 & \rho\, \sigma_{11}\, \sigma_{22} \\ \rho\, \sigma_{11}\, \sigma_{22} & \sigma_{22}^2 \end{pmatrix}$$

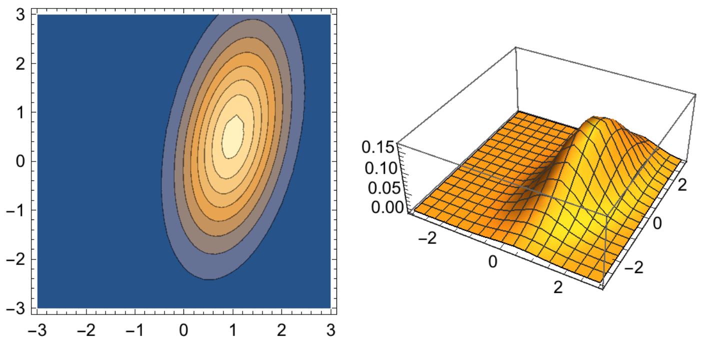
Examples of PDF, CDF

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}]}]
```



▶ 1. Use PositiveDefiniteMatrixQ[r]; Plot Σ from above: ContourPlot[{x1, x2}.r.{x1, x2}, {x1, -6, 6}, {x2, -6, 6}]

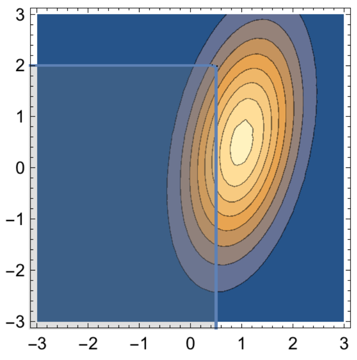
Calculating probabilities using the CDF

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:

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

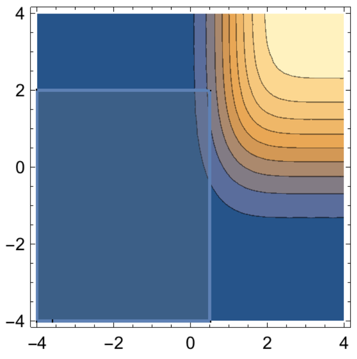
So the answer is the area under the PDF shown below.

```
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 over the plot to see the value of the CDF at $\{x_1, x_2\}$ = {.5, 2.0}.

```
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

Recall 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}}
```

Drawing samples

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

```
RandomVariate[ndist]
```

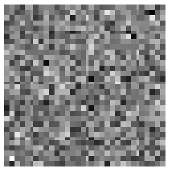
```
{1.27588, 1.6535}
```

or

```
RandomReal[ndist]
```

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 the others.

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



Marginalizing and conditioning normal distributions

Normal or gaussian distributions are the workhorses of probabilistic modeling involving continuous variables. And it isn’t just because of the central limit theorem--in fact most natural processes are non-gaussian and non-linear. There is a close relationships between linearity and gaussianity. So despite the world being non-linear and non-gaussian, linear and gaussian assumptions are practically useful in many contexts. More importantly, gaussian models provide a useful basis on which to build models of gaussian processes and non-gaussian distributions, e.g. mixture distributions.

This section demonstrates two basic properties of multivariate distributions, namely that they behave nicely when subjected to both the product and sum rules. Specifically, conditioning or marginalizing a multivariate gaussian results in another gaussian distribution.

Marginalization

Suppose, given the joint PDF[x_1,x_2], 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 the marginal distribution of x_1. Below we integrate out the other variable, $x_2$from the joint, p(x_1, x_2) = PDF[x_1, x_2]. Similarly, we can calculate PDF[x_2].

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

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

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

Note that both marginals are also normally distributed:

```
{marginal[x1], marginal2[x2]}
```

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

*This is a very important general point. In fact, you can project the variables of a bivariate normal on to any line (a*x1+b*x2 = c), and the resulting marginal will still be normally distributed. And more generally, the projection of high dimensional multivariate gaussian onto any of its dimensions will be a gaussian.*

We can sample from the marginals:

```
RandomVariate[marginalM, {10}]
```

```
{0.405627, 0.785671, 1.52309, 1.69814,
 2.16477, 1.6117, 0.447167, 1.4135, 0.701329, 1.18288}
```

And we can calculate modes. What is the mode of PDF[x_2]?

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

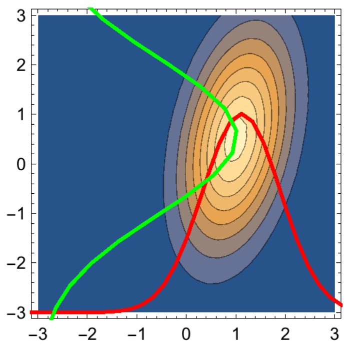
```
{0.282095, {zz → 0.5}}
```

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]}}]}]
```



▶ 2. Use the built-in function, MarginalDistribution[ ], to calculate and plot the marginals:

```
marginalM = MarginalDistribution[ndist, 1];
marginal2M = MarginalDistribution[ndist, 2];
```

Conditioning

Given the joint PDF[x_1,x_2], we want the conditional distribution given x_2, PDF[x_1, x_2] = $\frac{\text{PDF}[x_1, x_2]}{\text{PDF}[x_2]}$. The joint is:

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

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

and the marginal is:

```
marginal2[x2]
```

$$\frac{e^{-\frac{1}{16}(1-2\,x2)^2}}{2\sqrt{\pi}}$$

so the conditional distribution is:

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

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

Looks more complicated than it is, but it the conditional is still a gaussian.

```
FullSimplify[
```

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

$$-\frac{1}{128}(11-12\,x1+2\,x2)^2$$

..although we'd still want to show that we can write the argument in standard form.

And we can check to make sure the area under the curve along x1 is unity:

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

```
 {x1, -Infinity, Infinity}]
```

```
1
```

▶ 3. Calculate the condition with PDF[ndist,{x1,x2}] and PDF[marginal2M,x2]:

```
PDF[ndist, {x1, x2}] / PDF[marginal2M, x2]
```

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

Use FullSimplify[] on the exponential arguments to show that the answers are the same.

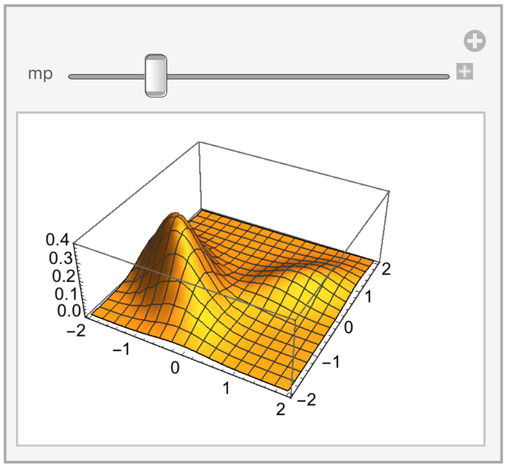
Mixtures of gaussians with MultinormalDistribution[ ]

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. Mixtures can be used to model clusters of feature values for object categories. The feature values for each category corresponds to a multivariate gaussian.

```
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:

```
Clear[𝒟, w, x, y];
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 application of mixture models to discovering visual features in natural images.

Using energy and gradient descent to derive update rules

In previous inference examples, weights were determined by Hebbian learning. So the energy landscape was sculpted by state vectors to be stored.

But we can also do things the other way around. Rather than figuring out the weights for a Hopfield-style network that has a known relationship to an energy function, we first specify the energy function, and then figure out an update rule that will descend the energy landscape. From the point of view of neural networks, this update rule may look nothing like what neurons do. But it may be the best way to start--that is, by sculpting the energy function directly, and then see what emerges in terms of an update rule.

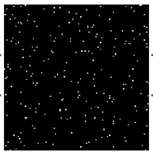
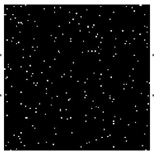
We are going to follow this strategy in this notebook on a simple problem of interpolation. It will turn out that our update rule is the simplest neural model--a linear summer. Although limited, this kind of analysis provides a starting point for more complicated energy functions with correspondingly non-linear update rules.

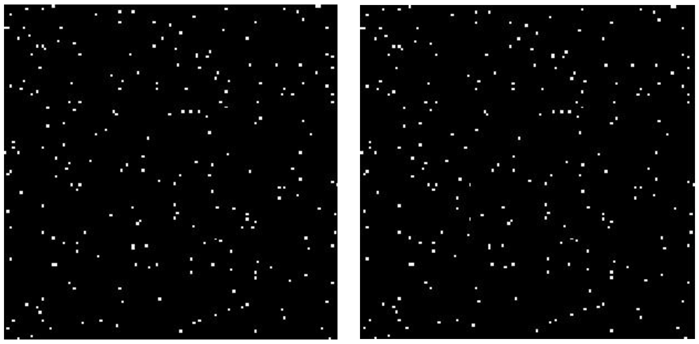
The energy function is sometimes referred to as a cost function or objective function.

Motivation: Interpolation problems in perception

A recurring theme in many of the problems in visual perception that the data available (“features”) in the image do not fully constrain the estimate of the surface or surface properties one would like to compute.

A random dot stereogram can have many local features densely packed, with substantial ambiguity in matching left eye pixels to those in the right. Here another stereo example in which the features (dots, and their disparities) are sparsely packed.

```
GraphicsRow[{gleft = Image[
```

```
], gright = Image[
```

```
]}]
```

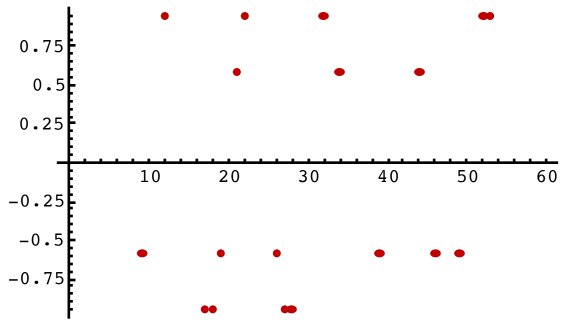


If you cross your eyes so the left and right images go to your right and left eyes, respectively, you may be able to see a small square surface, textured with dots, floating out in front of a background of dots. In addition to an illusory surface, you may also see illusory boundaries around the small square--i.e. edges where there are no changes in intensity. The visual system seems to *interpolate* a surface between salient points--the dots. The Kanizsa triangle is another example of perceptual interpolation.

We will study a simpler case of interpolation--namely filling in a line between feature points in just 1D. We will do this by constructing an energy function first, and then calculating an update rule by gradient descent. But the same principles apply to computational models of stereopsis, shape-from-shading, optic flow and other problems in early visual processing.

Energy: Data and smoothness terms

Suppose we have a set of points $\{i, d_i\}$, where i indexes the point, and d_i is the height at i.



We would like to find a smooth function, f[i] ~ di, that approximately fits the data points. However, we'll assume we don't have specific knowledge of the parametric form of the underlying function. In other words, we don't want to assume linear, quadratic, or general polynomial fits, or sinusoidal fits. We could just connect the dots. But if there is any wiggly noise, we'd have lots of wiggles that shouldn't be there. Further, the model probably wouldn't generalize well. We want a "smoother fit".

We assume that there are two constraints that will guide the problem of sculpting an energy function to reconstruct a line (or surface):

1. Fidelity to the data;
2) Smoothness of the fit.

Suppose $\{d_i\}$ are the data points, where the i's come from a subset, D of the total domain over which our reconstructed function, f is defined. Fidelity to the data can be represented by an energy or cost term that is big if the estimate of f is too far away from the data:

$$E_d = \sum_{i \in D} \left(f_i - d_i\right)^2$$

Smoothness can be represented by an energy cost in which near-by values of the estimate, f, are required to be close:

$$E_s = \sum_i \left(f_i - f_{i+1}\right)^2$$

We combine two constraints by adding:

$$E = E_d + E_s = \sum_{i \in D} \left(f_i - d_i\right)^2 + \lambda \sum_i \left(f_i - f_{i+1}\right)^2 \qquad (1)$$

λ is a free parameter that allows us to control how much the smoothing should dominate the data or fidelity term. Note we assume that the smoothness term is independent of the data, and is equivalent to assuming a Bayesian prior in statistics (Poggio et al., 1988; Kersten et al., 1987).

If we wish to start off with some initial guess for the values of f, we can successively improve our estimate by sliding down the slope of E in the steepest direction. As we saw in an earlier lecture, this says that the rate of change of f_i in time should be proportional to the negative slope of E in the direction of f_i

$$\frac{df_i}{dt} = -\frac{\partial E}{\partial f_i}$$

Recall the proof:

$$\Delta E = \nabla E \bullet d\vec{f} = \left|\nabla E\right|\left|d\vec{f}\right|\cos\theta$$

For step Δt, the biggest decrease in E is when cosθ = -1. In other words, when the vectors $\overrightarrow{\nabla E}$ and $\overrightarrow{df}$ are pointing in opposite directions. So if we make a change df in proportion to -∇E for each time step Δt, we will reduce the energy.

$\overrightarrow{df} \propto -\nabla E$

In the limit Δt→0, we can set:

$$\left(\frac{\partial E}{\partial f_1}, \frac{\partial E}{\partial f_2}, \ldots\right) = -\left(\frac{df_1}{dt}, \frac{df_2}{dt}, \ldots\right)$$

As mentioned above, we've encountered gradient descent before when we calculated the derivative of the error function with respect to the weights when we studied linear regression and the widrow-hoff rule. But now were are descending in activity, rather than weight space.

Taking the derivative of the above cost function:

$$\frac{df_i}{dt} \propto -\left(f_i - d_i\right) - \lambda\left(2f_i - f_{i+1} - f_{i-1}\right)$$

(One can use *Mathematica* to verify the pattern of the terms in the derivative.) Or in discrete time steps of dt,

$$f_i(t+dt) = f_i + dt\left\{-2\left(f_i - d_i\right) - 2\lambda\left(2f_i - f_{i+1} - f_{i-1}\right)\right\} \tag{2}$$

(Note: We aren’t really doing interpolation yet--for that we need to some book-keeping in that, we have to pay attention to when we are updating f 's that don't have data support. We show one way to handle that below.)

You can see that the value of f at time t+dt, is just the weighted sum of the values of f at time t, and the data, d. A weighted sum calculation is exactly what our linear model of a neuron does.

So we've derived an update rule for our energy function that could be implemented with a standard linear neural model.

► 4. What is the weight matrix? Are the diagonals zero? Is the matrix symmetric?

As we point out later in the low level vision subsection, the power of this approach is that one can construct more complicated energy functions, and derive gradient descent update rules that may not be linear but are useful to solutions. For example, the world we see is not constructed from one smooth surface, but is better modeled as a set of surfaces separated by discontinuities. An energy function can be constructed that explicitly models these discontinuities and their effects on the interpolated values. These are sometimes called *“weak membrane”* models. The energy function in this case is no longer quadratic, and the update rule computes more than a weight sum.

Example: Reconstructing a smooth line, “interpolation” using smoothness

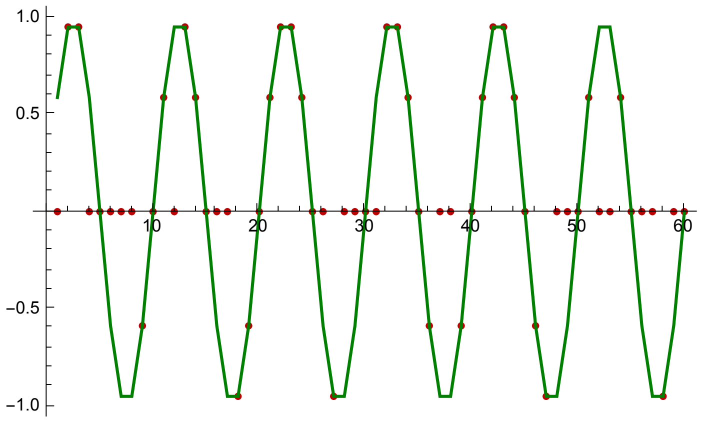
First-order smoothness

A generative model: Sine wave with random missing data points

Suppose we have sampled a function at a discrete random set of points **xs**. Multiplying the sine function by the vector **xs** picks out the values at the sample points at each location of the vector where **xs** is one, and sets the others to zero.

```
size = 60;
xs = Table[RandomInteger[1], {i, 1, size}];
data = Table[N[Sin[2 π j / 10] xs⟦j⟧], {j, 1, size}];
g3 = ListPlot[Table[N[Sin[2 π j / 10]], {j, 1, size}], Joined → True,
  DisplayFunction → Identity, PlotStyle → {RGBColor[0, 0.5`, 0]}];
g2 = ListPlot[data, Joined → False, PlotStyle → {RGBColor[0.75`, 0.`, 0]},
  Prolog → AbsolutePointSize[5]];
```

```
Show[g2, g3]
```



We are not going to fit the dots on the abscissa. These are places where data is missing, analogous to the stereo interpolation regions with uniform regions where there are no points to match. The non-zero data are analogous to the disparity values from the two eyes.

We would like to find a smooth function, **f[]**, approximation to the non-zero data points, given the assumption that we don't know what the underlying function actually is.

We have one constraint already--the fidelity constraint that requires that the function **f** should be close to the non-zero data, **d**. We will use this to construct the "energy" term that measures how close they are in terms of the sum of the squared error.

We need another constraint--smoothness--to get the in-between points. There are many ways of doing this. If we had a priori knowledge that the underlying curve was periodic, we'd try fitting the data with some combination of sinusoids. Suppose we don't know this, but do have reason to believe that the underlying function is smooth in the sense of nearby points being close. As above, let's assume that the difference between nearby points should be small. That is, the sum of the squared errors, **f[i+1] - f[i]**, gives us the second part of our energy function.

Let's make up a small 8 element energy vector:

```
Clear[f];
energyvector =
Table[(f[i+1] - f[i])^2 + s[i] (d[i] - f[i])^2,
            {i,1,8}];
```

```
energy = Sum[energyvector[[j]],{j,1,8}];
```

The **s[i]** term is the "filter" (the same as **xs** above) that only includes data points in the data part of the energy function. It is zero for i's where there are no data, and one for the points where there are data.

We would like to find the **f[]** that makes this energy a minimum. We can do this by calculating the derivative of the energy with respect to each component of **f**, and moving the state vector in a direction to minimize the energy--i.e. in the direction of the negative of the gradient.

It can be messy to keep track of all the indices in these derivatives, so let's let *Mathematica* calculate the derivative for f[3]. From this we can see the pattern for any index.

```
D[energy, f[3]]
```

```
2 (-f[2] + f[3]) - 2 (-f[3] + f[4]) - 2 (d[3] - f[3]) s[3]
```

```
FullSimplify[%]
```

```
-2 (f[2] + f[4] + d[3] s[3] - f[3] (2 + s[3]))
```

We can use this pattern to see how to set up the Tm matrix below.

Given the derivatives how to solve for f? Now we could generate the full set of derivatives, set them equal to zero and solve for **f**, using standard linear algebra to solve a set of linear equations as we saw in the lecture on regression and Widrow-Hopf. . This will work because the energy function is quadratic in elements of **f**, and thus the derivatives are linear in **f**. The interpolation function will then be a matrix operation on the data. And we can solve our interpolation/approximation problem with a one shot, feedforward operation. As mentioned before, life won't always be that easy, and the situation often arises in which the energy function is not quadratic.

So the alternative, which can work in a non-linear case too, is to iteratively update the activity pattern f with an update rule--gradient descent. We can do this by expressing the derivative in terms of two matrix operations: One on the function to be estimated, f, and one on the data. Let's set up these two matrices, **Tm** and **Sm** such that the gradient of the energy is equal to: **Tm . f - Sm . data**

Sm will be our filter to exclude non-data points. **Tm** will express the "smoothness" constraint.

```
Sm = DiagonalMatrix[xs];
Tm = Table[0,{i,1,size},{j,1,size}];
For[i=1,i<=size,i++,Tm[[i,i]] = xs[[i]]+2];
For[i=1,i<size,i++, Tm[[i+1,i]] = -1];
For[i=1,i<size,i++, Tm[[i,i+1]] = -1];
```

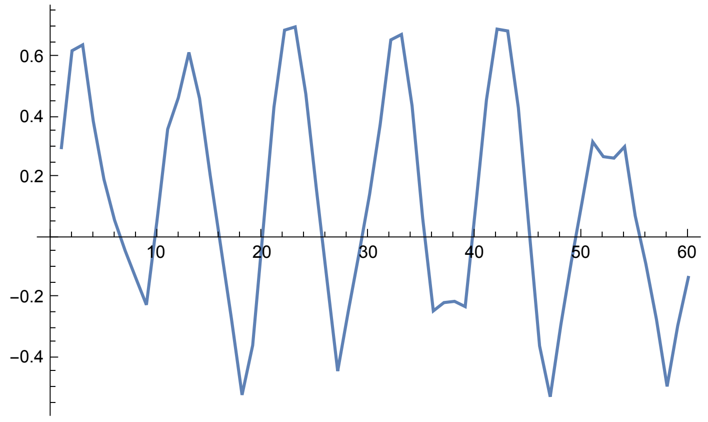
```
dt = 0.1;
Tf[f1_] := f1 - dt (Tm.f1 - Sm.data);
```

We will initialize the state vector to zero, and then run the network for 30 iterations:

```
f = Table[0,{i,1,size}];
result = Nest[Tf,f,30];
```

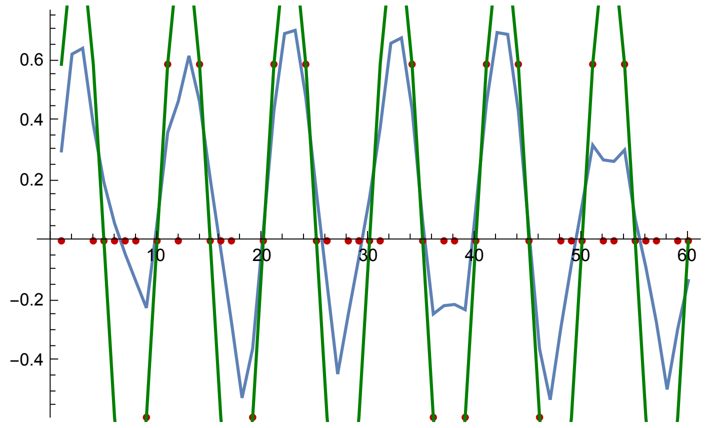
Here is our smoothed function:

```
g1 = ListPlot[result, Joined → True]
```



You can see below that the function was *not interpolated* in the strict sense--the fit doesn't not pass through the data points. If we wanted more fidelity to the data, we could control this by increasing the weight given to the data part of the energy term relative to the smoothness part.

```
Show[{g1, g2, g3}]
```



► 5. Exercise:

Decrease the parameter controlling smoothness to see if you get better fidelity.

Sculpting for interpolation using second order smoothness constraints

This section elaborates the energy function (and weight matrix) to require that both first and second order differences be small.

```
Clear[energy,energyvector,f,d,s,data];
```

```
energyvector =
Table[(f[i+1] - f[i])^2 + (f[i+2] - 2 f[i] + f[i+1])^2 + s[i] (d[i] - f[i])^2,{i,1,8}];
```

```
energy = Sum[energyvector[[j]],{j,1,8}];
```

By taking the derivative of the energy with respect to one of the interpolation depths, say f[3], we can see the pattern of the weights for the gradient descent update rule:

```
D[energy, f[3]]
```

```
2 (-f[2] + f[3]) + 2 (-2 f[1] + f[2] + f[3]) - 2 (-f[3] + f[4]) +
 2 (-2 f[2] + f[3] + f[4]) - 4 (-2 f[3] + f[4] + f[5]) - 2 (d[3] - f[3]) s[3]
```

```
Simplify[%]
```

```
-2 (2 f[1] + 2 f[2] - 8 f[3] + 2 f[4] + 2 f[5] + d[3] s[3] - f[3] s[3])
```

Now we simulate the sampled data, and then set up the weight matrix:

```
size = 120;
xs = Table[RandomInteger[1], {i, 1, size}];
```

$$\text{data} = \text{Table}\left[\text{N}\left[\text{Sin}\left[\frac{2\,\pi\,j}{20}\right]\text{xs}[\![j]\!]\right], \{j, 1, \text{size}\}\right];$$

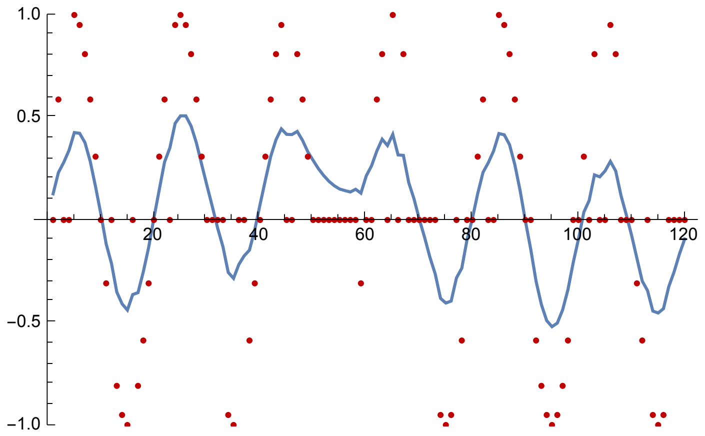
```
Sm = DiagonalMatrix[xs];
Tm = Table[0, {i, 1, size}, {j, 1, size}];
For[i = 1, i ≤ size, i++, Tm[[i, i]] = xs[[i]] + 8];
For[i = 1, i < size, i++, Tm[[i + 1, i]] = -2];
For[i = 1, i < size, i++, Tm[[i, i + 1]] = -2];
For[i = 1, i < size - 1, i++, Tm[[i + 2, i]] = -2];
For[i = 1, i < size - 1, i++, Tm[[i, i + 2]] = -2];
```

We will give the smoothness term a little more weight relative to the fidelity term:

```
dt = 0.1;
λ=1.;
Tf[f1_] := f1 - dt (λ*Tm.f1 - Sm.data);
```

```
f = Table[0,{i,1,size}];
result = Nest[Tf,f,30];
```

```
g1 = ListPlot[result, Joined → True, PlotRange → {-1, 1}, DisplayFunction → Identity];
g2 = ListPlot[data, Joined → False, PlotRange → {-1, 1}, PlotStyle → {RGBColor[0.75`,
      0.`, 0]}, Prolog → AbsolutePointSize[5], DisplayFunction → Identity];
Show[g1, g2, DisplayFunction → $DisplayFunction]
```



▶ 6. Try different values of the smoothing weight

Probabilistic, Bayesian interpretation

When we know the structure of a problem, and want to construct a Bayesian model, it is often easier to begin by constructing a cost or energy function as we did above. And if needed (e.g. if you want to draw samples), from there construct a Bayesian posterior.

Minimizing

$$E = E_d + E_s = \sum_{i \in D} (f_i - d_i)^2 + \lambda \sum_i (f_i - f_{i+1})^2 \tag{3}$$

is a special case of finding values of f = $\{f_1, f_2, \ldots\}$, that maximize:

$$p(f \mid d) = \frac{p(f, d)\, p(f)}{p(d)} \propto \prod_i e^{-\frac{(f_i - d_i)^2}{2\sigma_d^2}} \times \prod_i e^{-\frac{(f_i - f_{i+1})^2}{2\sigma_p^2}}$$

given data d = $\{d_1, d_2, \ldots\}$, and a standard deviation (σ_d) for assumed additive noise in the measurements d, and a standard deviation (σ_p) representing prior uncertainty in the smoothness of f.

To see this, note that the logarithm of p(f |d) is proportional to the energy--a sum of the same quadratic terms as in equation (3), and that $\lambda = (\sigma_p / \sigma_d)^2$, which specifies the trade-off between the noisiness of the data and the strength of your prior beliefs. If the data is very noisy, you may want to rely more on the prior--i.e. more on “smoothing”. If the data is both accurate and dense, you wouldn’t even need a prior.

Note that minimizing a quadratic function leads to a linear estimate--i.e. the solution is a matrix operation on the data, d.
In fact, there is a close relationship between normal distributions and linear estimates. Just as the

assumption of linearity makes computations fast and convenient, so does the normal assumption.

Low-level vision

Smoothing and regularization

The solutions of many of computational problems of early vision can be formulated in terms of maximum a posteriori estimates of scene attributes (Poggio, Torre and Koch, 1985). These problems include: detecting edges in images, perceiving/estimating optic flow, perceiving surface color, depth, and stereo vision.

A linear solution starts with the assumption that the generative model can be described as a matrix multiplication, where the image I is matrix mapping A of a scene vector S:

$$I = \mathbf{A}S$$

$$E = (I - \mathbf{A}S)^T (I - \mathbf{A}S) + \lambda S^T \mathbf{B} S$$

The scene vector could represent true depth. Then a solution corresponded to minimizing a cost function E, that simultaneously tries to minimize the cost due to reconstructing the image from the current hypothesis S, and a prior "smoothness" constraint on S, as we saw above. λ is a (often free) parameter that determines the balance between the two terms. If there is reason to trust the data, then λ is small; but if the data is unreliable, then more emphasis should be placed on the prior, thus λ should be bigger. For example, S could correspond to representations of shape, stereo, edges, or motion field, and smoothness be modeled in terms of nth order derivatives, approximated by finite differences in matrix B.

The Bayesian interpretation comes from multivariate gaussian assumptions on the generative model:

$$p(I \mid S) = k \times \exp\left[-\frac{1}{2\sigma_n^2}(I - \mathbf{A}S)^T (I - \mathbf{A}S)\right]$$

$$p(S) = k' \times \exp\left[-\frac{1}{2\sigma_s^2} S^T \mathbf{B} S\right]$$

From Poggio, Torre & Koch, 1985

In edge detection, noise can create spurious edges. The way to deal with that is by blurring the image and then applying a spatial derivative. The above constraint says to assume there is an underlying “image”, f, that has the “true perfectly sharp edges”, which have gotten smoothed out by filter S to produce i; and because there is noise, we should find the find f that is consistent with this generative assumption, but restricted to the f which is most smooth. This latter constraint is measured by the square of the second spatial derivative of f: f_{xx}.

For optic flow (area based), the gradient constraint is what we have seen before: $i_x u + i_y v + i_t = 0$. The smoothness constraint here is expressed as:

$u_x^2 + u_y^2 + v_x^2 + v_y^2$, which discourages rapid spatial changes in the optic flow vectors. The notation above is: $\frac{\partial f}{\partial x} \longleftrightarrow f_x$.

Yuille, A. L., & Grzywacz, N. M. (1988) proposed including all derivatives, which would include this, as well as the “slow” and “smooth” assumptions in the work by Weiss et al. (2002).

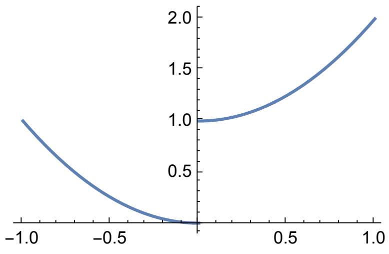
Above we looked at the one-dimensional analog of surface reconstruction where we contrasted the cost function minimization, similar to that above, with a probablistic formulation solved through belief-propagation.

A key point is that the maximum a posteriori solution based on equations 1 and 2 above is linear. Thus given the “right” representation, a broad range of estimation problems can be modeled as simple linear networks. However, we noted early on that there are also severe limitations to linear estimation.

An example of an extension to piece-wise smooth interpolation

Many natural processes are smooth within limited domains, then make a jump to another domain. The depths of surfaces, and colors of surfaces have this property. Here’s a 1D example:

```
Plot[Piecewise[{{x^2, x < 0}, {x^2 + 1, x > 0}}], {x, -1, 1}]
```



How can we estimate piecewise functions from data? Suppose our data is represented by a set of noisy measurements I at locations x, and we want to estimate a function J(x). We we will also want to estimate the locations of the discontinuities, represented by l(x) which is one at points of discontuity--and it is at these points that we don’t want to encourage nearby values of J to be similar--i.e. we want to discourage J(x+1) from being close to J(x). These constraints suggest posterior and energy functions below.

$$P(\boldsymbol{J},\boldsymbol{l}|\boldsymbol{I}) = \frac{1}{Z}\exp\{-E[\boldsymbol{J},\boldsymbol{l}:\boldsymbol{I}]/T\}$$

$$E[\boldsymbol{J},\boldsymbol{l}:\boldsymbol{I}] = \sum_x (I(x)-J(x))^2 + A\sum_x (J(x+1)-J(x))^2(1-l(x)) + B\sum_x l(x)$$

See the “Line Process Model” section of: http://gandalf.psych.umn.edu/users/kersten//kersten-lab/papers/YuilleKerstenFinalChapter2016.pdf

Next time

Discrete Bayes
Graphical models

References

Duda, R. O., & Hart, P. E. (1973). Pattern classification and scene analysis . New York.: John Wiley & Sons.
Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley. (Amazon.com)
Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the theory of neural computation* (Santa Fe Institute Studies in the Sciences of Complexity ed. Vol. Lecture Notes Volume 1). Reading, MA: Addison-Wesley Publishing Company.
Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv Preprint arXiv:1207.0580.
Poggio, T., Torre, V., & Koch, C. (1985). Computational vision and regularization theory. Nature, 317, 314-319.
Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. Neural Computation, 4(1), 1-58.
Kersten, D., O'Toole, A. J., Sereno, M. E., Knill, D. C., & Anderson, J. A. (1987). Associative learning of scene parameters from images. Appl. Opt., 26, 4999-5006.
Kersten, D. J. (1991). Transparency and the Cooperative Computation of Scene Attributes. In M. Landy, & A. Movshon (Ed.), Computational Models of Visual Processing (pp. 209-228). Cambridge, Massachusetts: M.I.T. Press.
Kersten, D., & Madarasmi, S. (1995). The Visual Perception of Surfaces, their Properties, and Relationships. In I. J. Cox, P. Hansen, & B. Julesz (Eds.), Partitioning Data Sets: With applications to psychology, vision and target tracking, (pp. 373-389): American Mathematical Society.
Koch, C., Marroquin, J., & Yuille, A. (1986). Analog "neuronal" networks in early vision. Proc. Natl. Acad. Sci. USA, 83, 4263- 4267.) See also, Hertz et al., page 82-87.
Nakayama, K., & Shimojo, S. (1992). Experiencing and perceiving visual surfaces. Science, 257, 1357-1363.
MacKay, D. J. C. (1992). Bayesian interpolation. *Neural Computation, 4*(3), 415-447.
Weiss, Y., Simoncelli, E. P., & Adelson, E. H. (2002). Motion illusions as optimal percepts. Nature Neuroscience, 5(6), 598–604. doi:10.1038/nn858