# Computational Vision
# U. Minn. Psy 5036
# Daniel Kersten
# Lecture 11: Image Coding

### Initialize

■ **Read in Statistical Add-in packages:**

In[1]:=
```
Off[General::spell1];
<< Statistics`DescriptiveStatistics`
<< Statistics`DataManipulation`
<< Statistics`NormalDistribution`
<< Statistics`ContinuousDistributions`
<< Graphics`Graphics`
```

## Outline

### Last time

Final projects

Image operations

Point non-linearities, and simple statistics

Using function interpolation for image morphing, symbolic differentiation.

Properties of the fourier transforms of images (in the online class tutorial: Fourier_neural_image.nb )

E.g., if you haven't tried it yet, look at the relative importance of amplitude and phase spectra in object recognition

### Today

Natural image statistics and efficient coding

Understanding first-order statistics in terms of efficient coding of natural images

## Efficient coding: 1rst order statistics & point operations

In 1981, Simon Laughlin published a paper in which he showed that the contrast response function of  LMC interneurons of the fly's ommatidium ("large monopolar cells") had a sigmoidal non-linearity as shown below. This kind of non-linearity wasn't new, and is common, especially in sensory response functions, that it is a de facto standard point non-linearity in generic neural network models.

But why the sigmoidal shape? The mechanistic explanation for this kind of sigmoidal shape was and has been that small signals tend to get suppressed (e.g. a "soft" threshold), and at the high end, signals get saturated.

Laughlin came up with a different kind of answer, based on a functional argument that went along the following lines: the fly lives in a visual world in which big contrasts (negative or positive) are less common than contrasts near zero, so neurons should devote more of the resolving capacity to the middle contrasts. This kind of argument is based on information theory. In this lecture, we'll develop some basic tools of information theory to understand his model and others like it.

In image processing jargon, the fly's visual neuron is doing "histogram equalization". Let's see what that means.

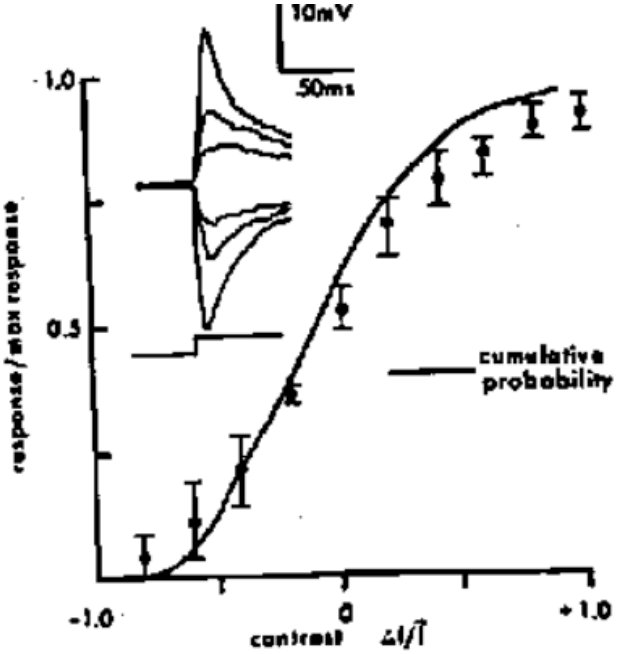



Fig. 2. The contrast-response function of light adapted fly LMC's compared to the cumulative probability function for natural contrasts (50° interval). Response is normalised with the maximum amplitude to light off as 0.0, and the maximum amplitude to an increment as 1.0. Data points averaged from 6 cells; range bars show total scatter. Inset

## Important caveat

In the next two lectures we are going to study  statistical regularities found in natural images. Efficient coding takes advantage of regularities to represent images with fewer bits. For instructional purposes, we'll illustrate these regularities with arbitrary images that haven't been digitally calibrated. Further, depending on the kind of compression (e.g. "lossy"), images like jpeg images have statistical properties that deviate from the original uncompressed image. There exist several carefully calibrated image data sets, such as the "van Hateren" database: http://hlab.phys.rug.nl/archive.html.

## Import image file "granite.jpg"

### ■ Using Import[]

As noted earlier, you can use "Get File Path" from the "Input" menu to paste in the directory name of a file, or to use as an argument for SetDirectory. You can use **Import[ ]** to read in image files of various formats.

E.g. on my computer, **granite = Import["PowerBookHD:Research:kersten-lab:courses:Psy5036:Lectures:11. Efficient coding:granite64x64.jpg"];**

Alternatively, you can use the following to open a dialog window to look for the image file you want. I'm going to read in: GrayAlpine256x256.jpg

```
In[7]:= graniteg = Import[Experimental`FileBrowse[False]];
```

```
In[8]:= Show[graniteg];
```

graniteg is of type **-** Graphics **-**. To manipulate pixel values, we need to pull out the first element [[1,11]], which is the image matrix of pixel values that we want:

In[9]:=
```
granite = graniteg[[1, 1]];
ListDensityPlot[granite, Mesh → False];
```

Let's calculate the max, mean and standard deviation. The standard deviation gives us a measure of contrastiness (see previous lecture's definitions of contrast)

In[11]:=
```
ListDensityPlot[granite, Mesh → False, Frame → False];
Max[Flatten[granite]]
N[Mean[Flatten[granite]]]
N[StandardDeviation[Flatten[granite]]]
```

Out[12]= 255

Out[13]= 128.805

Out[14]= 46.5557

## Histogram

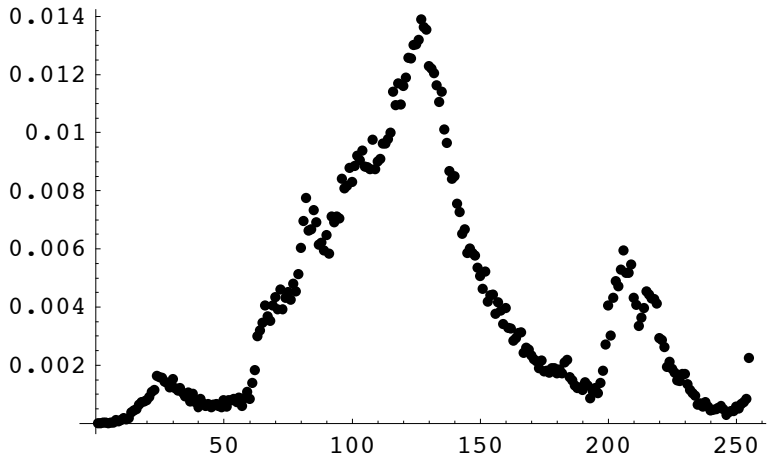
Now let's calculate some first-order statistics. First-order means that we are looking at the frequency of occurrence of pixel intensity values. In the next lecture we'll will look at second-order statistics that describe how pixel intensities at one location are related to nearby ones. One of *Mathematica*'s add-on functions is Histogram[ ], but we'll build our own using BinCounts[ ].

■ **Define histogram function that accepts an image with graylevels [lowlimt,highlimit], and outputs a histogram normalized to 1:**

In[15]:=
```
histogram[image_, binsize_, lowlimit_, highlimit_] :=
  Module[{histx},
   histx = BinCounts[Flatten[image], {lowlimit, highlimit, binsize}];
   Return[N[histx / Plus @@ histx]];
  ];
```

The histogram gives us an estimate of $p_i$, the probability of the ith intensity, e.g. the probability of a pixel's value being say graylevel 103.

In[16]:=
```
histograniteimage = histogram[granite, 1, 0, 255];
ListPlot[histograniteimage, PlotStyle → PointSize[0.015]];
```



**What is the frequency of occurrence of graylevel 103?**

---

## Entropy

Entropy provides a scalar measure of the degree of disorder in an image. "Disorder" isn't necessarily bad because it is closely related to the degree of novelty or surprise in a pattern. We want a measure of information that reflects degree of surprise in a formal objective sense.

A highly ordered or regular pattern is by definition somewhat predictable--there is a "pattern there". A highly probable event doesn't convey as much information as a low probability event. "I will go to sleep tonight" vs. "I will parachute tonight". The second statement conveys more information than the first. An event for us will be a pixel taking on a particular graylevel value, say i, where we have a space of 256 possible events.

If a pixel can take on any graylevel with equal probability, like the noise images you have generated, the entropy is high (and in fact maximum). On the other hand, if pixels can take on only one value, entropy is low (and in fact 0). (Entropy for a continuous random variable is different--the entropy doesn't have a natural lower bound, and the minimum can be negative)

*entropy* is defined as the average (expected) information over all N events, where information of an event i is measured as:

$$-\text{Log}_2\ p_i\,.$$

This formula satisfies at least one requirement for a measure of information, that it should be monotonically related to the degree of surprise.

So by the definition of expectation (or average):

$$entropy = -\sum_{i=1}^{N} p_i \text{Log}_2\ p_i \tag{1}$$

Information and entropy are measured in bits. Again, recall that an event for us is a particular graylevel.

Entropy reflects the amount of information conveyed on average by a set of signals. A 256x256 gaussian white noise image may look boring and unsurprising to you, but a true sample is actually completely novel--you've never seen it before, and you'll never see it again (or at least the odds are vanishingly low).

Sometimes probabilities are 0, so we set 0 Log(0) = 0, and then the following function calculates entropy for a list of probabilities:

In[18]:=
```
entropy[probdist_] := Plus @@ (If[# == 0, 0, -# Log[2, #]] & /@ probdist)
```

**Plus@@list** is short-hand for **Apply[Plus,list]**. They both return the sum of the elements in the **list**.

**If[#==0,0,-# Log[2,#]]&** is short-hand for a function to be applied to elements of our list. # is a placeholder for the variable that gets plugged in. We could have defined a function: **information[p_]:=If[p==0,0,-p Log[2,p]];**

**/@** is short-hand for **Map[]**. I.e. we could have done: **Apply[Plus, Map[information, probdist]]**, or even longer, we could have written a loop.

We can use our entropy function to verify that the entropy is biggest when the probability distribution is uniform, i.e. when $p_i$=1/N. So the maximum entropy for our graylevel pictures would be: **entropy[Table[1/256, {256}]] = 8** bits.

**Verify this and calculate: entropy[Table[1/256, {256}]]**

---

Let's calculate the "first-order" entropy of our grayalpine image:

```
In[19]:= entropy[histograniteimage]
```

```
Out[19]= 7.39632
```

## Histogram equalization

### ■ Theory

In the subsequent analysis, we will treat images as having continuous, rather than discrete intensity values. (When we calculate entropy, we'll first bin the intensities into 256 bins). Consider for the moment, a large set of images {g(x,y)} with continuous intensity values. Suppose that the distribution of intensities was truly gaussian with a particular mean graylevel and standard deviation:

$$p_g(\mathrm{g}) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2\sigma^2}(g-\mu)^2}.$$

If we map f = $\phi$(g), we'd get a new set of images {f(x,y)}. What would the histogram look like over this set? The answer comes from the density mapping theorem. (See the Probability Overview in Lecture 5) Let's call the new density of f: $p_f(f)$. Assume $\phi$() is monotonic. The fundamental idea is that probability mass should be conserved,

$$p_f(f)\Delta\mathrm{f} \simeq p_g(\mathrm{g})\Delta\mathrm{g}. \tag{2}$$

What if we want $p_f(f)=\epsilon$, a constant over some domain?

$$\epsilon\,\mathrm{df} = p_g(g)\,\mathrm{dg}, \text{ then we integrate both sides to get:} \tag{3}$$

$$\mathrm{f(g)} \propto \int_{-\infty}^{g} p_g(g')\,dg' \tag{4}$$

Thus f is proportional to the function given by the cumulative distribution. In other words, if we draw a gaussian number, g, and run it through the point-wise non-linearity given by the formula for the cumulative gaussian,

$$\mathrm{f(g)} = \frac{1}{2}\left(1 + \mathrm{Erf}\left[\frac{g-\mu}{\sqrt{2}\,\sigma}\right]\right) \tag{5}$$

f(g) will be uniformly distributed. Thus the function f gives us the mapping rule to convert a non-uniform histogram to a uniform one, i.e. to do "histogram equalization" for an image.

(Note that if we use the inverse $f^{-1}$, we have the means to generate gaussian random variables from uniformly distributed ones. See the Exercises section of the ProbabiltyOverview.nb)

## ■ Synthetic images: Gaussian white noise (discretized & clipped)

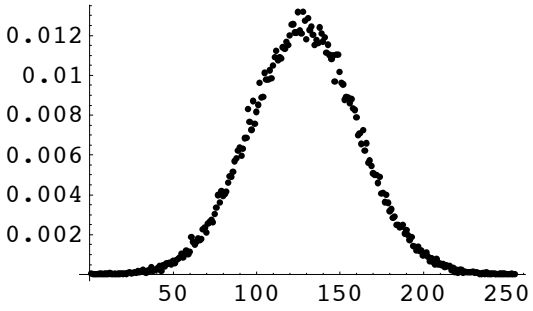
Recall that "white" means that the pixel intensities are independent of each other. We'll make this notion precise in the next lecture when we learn about 2nd order statistics. For our purposes here, it just means we randomly draw samples for each pixel independent of what any of the previous draws were.

Because a display graylevel is in the range [0,255], we clip the extreme samples using **Which[]** in the sampling function **randomguess**:

In[35]:=
```
ndist = NormalDistribution[128, 32];
randomgauss := Which[(r = Random[ndist]) > 255, 1, r < 0, 0, True, r];
```

In[37]:=
```
gaussimage = Table[randomgauss, {i, 1, 256}, {j, 1, 256}];
```

In[38]:=
```
histogaussimage = histogram[gaussimage, 1, 0, 255];
ListPlot[histogaussimage, PlotStyle → PointSize[0.015]];
entropy[histogaussimage]
```




Out[40]= 7.04778

**How does the entropy change if the standard deviation is smaller, say 8, instead of 32?**

---

## ■ Non-linearity

The cumulative gaussian is the probability of an intensity being higher than x. The cumulative gives us the form for the non-linearity (i.e. f=y):

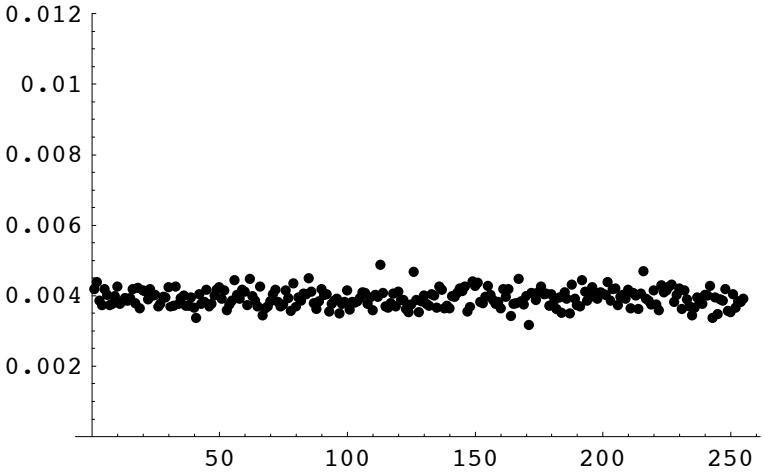
$$y[x\_, \mu\_, \sigma\_] := 256 * \frac{1}{2}\left(1 + \text{Erf}\left[\frac{x - \mu}{\sqrt{2}\,\sigma}\right]\right);$$

(Note that *Mathematica* has a built-in add-on function for the cumulative gaussian, so we could have used that).

If the standard deviation exactly matches that of the gaussian white noise parameters, we can make newgaussimage by running the old pixel values through the (sigmoidal point) non-linearity specified by y[] and produce a new image with a uniform white noise spectrum:

```
newgaussimage = Round[y[gaussimage, 128, 32]];
```

```
histogaussimage = histogram[newgaussimage, 1, 0, 255];
ListPlot[histogaussimage, PlotStyle → PointSize[0.015],
  PlotRange → {0, 0.012}];
entropy[histogaussimage]
```




```
7.99136
```

...now the entropy is near the maximum of Log[2,256] = 8 bits.

**Plot up gaussimage and newgaussimage. Which one appears "contrastier"? Which one has the greater standard deviation?**

---

## ■ Equalize an imported natural image, e.g. grayalpine

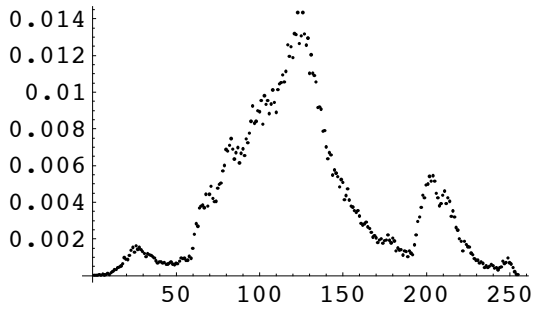
We'll first add a tiny bit of Gaussian noise to granite. This is more realistic, and it gives us a larger intensity vocabulary.

In[41]:=
```
Clear[randomgauss];
noise = Table[Random[NormalDistribution[0, 2]], {i, 1, 256}, {j, 1, 256}];
```

In[43]:=
```
granite = granite + noise;
granite = 255.0 * granite / Max[granite];
Max[granite]
```

Out[45]=
255.

In[46]:=
```
histogranite = histogram[granite, 1, 0, 255];
ListPlot[histogranite, PlotStyle → PointSize[0.008]];
```




In[48]:=
```
entropy[histogranite]
```

Out[48]=
7.37776

Calculate the cumulative distribution.

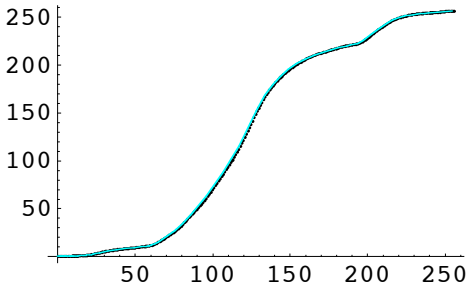
In[49]:=
```
cumulhistogranite = 256.0 * FoldList[Plus, histogranite[[1]], histogranite];
g1 = ListPlot[cumulhistogranite, DisplayFunction → Identity];
```

Use **ListInterpolation** to fit a continuous function to cumulative distribution, and plot up data with function fit.

In[51]:=
```
fcumulhistogranite = ListInterpolation[cumulhistogranite, {{0, 255}}];
```

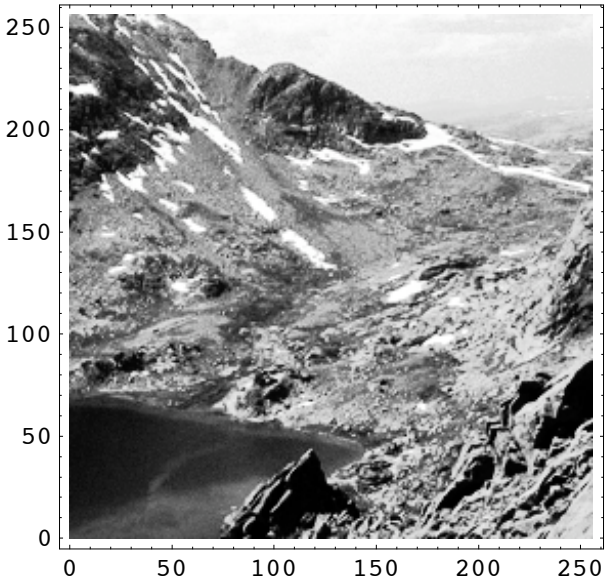
In[52]:=
```
g2 = Plot[fcumulhistogranite[x], {x, 0, 255}, PlotStyle → Hue[0.5],
   DisplayFunction → Identity];
```

In[53]:=
```
Show[g1, g2, DisplayFunction → $DisplayFunction];
```




Use cumulative function to re-map intensities:

In[54]:=
```
equalgranite = Round[Map[fcumulhistogranite, granite, {2}]];
ListDensityPlot[equalgranite, Mesh → False, PlotRange → {0, 255}];
```



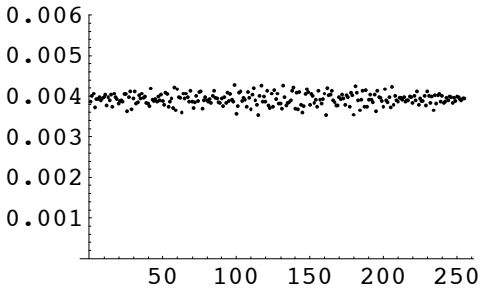

As we'd expect, the histogram equalization increases contrast as compared with the original image:

**Compare above image with original: ListDensityPlot[granite,Mesh→False, PlotRange→{0,255}];**

---

Check final histogram and entropy:

In[56]:=
```
histoequalgranite = histogram[equalgranite, 1, 0, 255];
ListPlot[histoequalgranite , PlotStyle → PointSize[0.01],
  PlotRange → {0, 0.006}];
```




In[58]:=
```
entropy[histoequalgranite]
```

Out[58]= 7.99337

(If we had kept our 256 gray-level intensity "vocabulary", the non-linearity leaves out lots of gray-levels in the output. Contrast would still be enhanced, but the mapping can actually reduce entropy. A one-to-one re-mapping of labels shouldn't affect entropy at all.)

## ■ Does histogram equalization improve image quality?

There used to be a fair amount of research investigating how various forms of histogram equalization might be used to improve the visibility of important information in degraded images, such as medical radiograms. The short answer, is that histogram equalization does little to help.

## Histogram equalization by the fly's eye

The sigmoidal non-linearity of photoreceptors found in the fly (and other animals) is a good design to efficiently code the range of light typically found in an image (Laughlin, 1981; Richards, 1981; see figure below). The idea is that if you made a plot of the distribution of light intensities in a typical scene about some mean level, you might find something like the gaussian distribution shown below. The grayapline wasn't gaussian at all, but it was roughly more peaked in the middle range. The granite image (see Appendix) has an approximately gaussian histogram.

Even if the intensity values are continuous, the effective resolution of intensity may be limited by a constant amount of noise, reducing the effective response range (like quantization). If you wish to divide up the response range efficiently, it would make sense to devote larger portions of the range where you are most likely to encounter the more frequently occuring intensities. Intensities that are unlikely should get squeezed off into the lower and upper parts of the response range. Such a strategy could be implemented with a sigmoidal point mapping function as shown in the figure. In fact, the optimal non-linearity would correspond to the cumulative probability distribution. Or in other words, the fly's light transducer should be performing "histogram equalization".

Laughlin was able to show a good correspondence between theory and actual photoreceptor non-linearity in the fly.

The non-linear coding serves to improve the cell's information capacity. Entropy has measured in the responses is higher--the average "surprise value" of a response value is higher than without the non-linearity.

Are there ways in which the retina could take advantage of other statistical regularities in natural images to efficiently encode image information? Next time we will see how information can be encoded even more efficiently by taking advantage of the regularities across space and time.

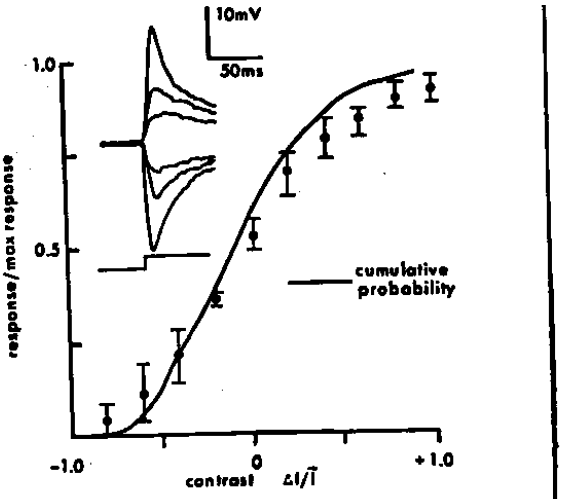



Fig. 2. The contrast-response function of light adapted fly LMC's compared to the cumulative probability function for natural contrasts (50° interval). Response is normalised with the maximum amplitude to light off as 0.0, and the maximum amplitude to an increment as 1.0. Data points averaged from 6 cells: range bars show total scatter. Inset

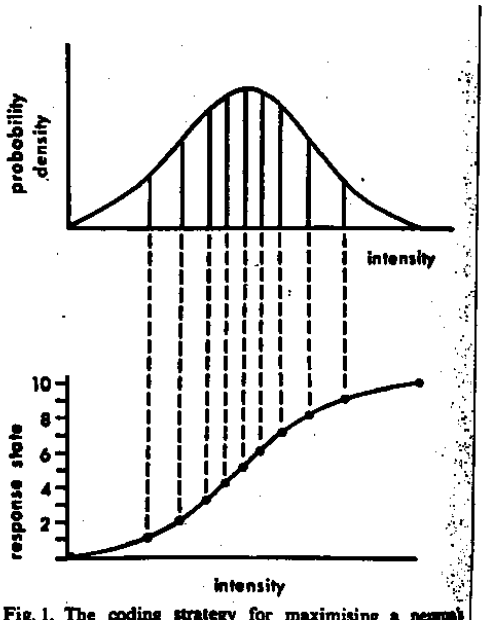



Fig. 1. The coding strategy for maximising a neural

# Sparseness of difference filters for natural images

When Hubel and Wiesel began their seminal recordings of the responses of neurons in visual cortex to images, they made a very important observation: It was really hard to get a neuron to fire. They needed to shwo the neurons edges of just the right orientation and location. Why are cortical V1 neurons usually so quiet?

We can get a preliminary understanding by calculating the histograms of the responses to linear difference filters to natural image input.

## Are images Gaussian?

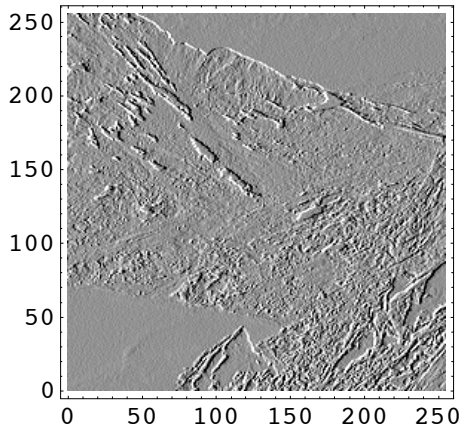
The weighted sum (or difference) of two Gaussian random variables is also a Gaussian. Thus if we compute a new image that is a linear combination of pixel intensities in a gaussian image (e.g. as in any convolution), the responses should have a gaussian histogram. Let's take a look at the histogram of a simple difference operator on the grayalpine image. Here is a discrete approximation of a short vertically oriented "edge detector", or first derivative:

In[59]:=
```
kern = {{1, -1}}
```

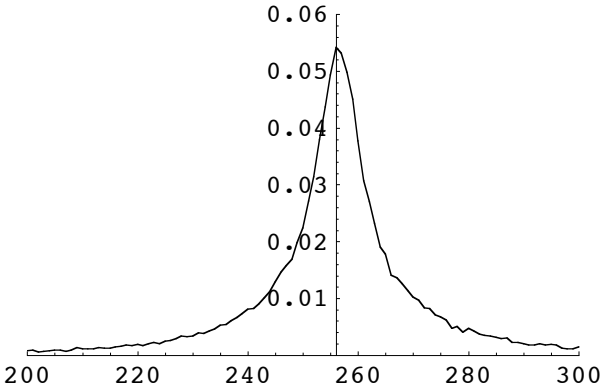
Out[59]=
$$(1 \quad -1)$$

In[60]:=
```
diffgranite = ListConvolve[kern, granite];
```

In[61]:=
```
ListDensityPlot[diffgranite, Mesh → False];
```



In[62]:=
```
empiricalhistg = ListPlot[histogram[diffgranite, 1, -256, 255],
   PlotRange → {{200, 300}, {0, .06}}, PlotJoined → True,
   AxesOrigin → {256, 0}];
```




This histogram is strikingly regular (compared with the intensity histogram), with a fairly sharp peak and long tails. The histogram is said to have large "kurtosis".

You can try to fit this with a Gaussian, but you will discover that the tails of the distribution are too extended for a Gaussian.

It turns out that most natural images when filtered with a spatial filter whose area sums to zero (i.e. has as much positive as negative weights, as in the center-surround filters we've studied) produces a neural image with high kurtosis. An interesting neural interpretation is that such a neural code produces sparse responses, i.e. there are a few units with really big responses, but most responses huddle near the mean (which is zero). David Mumford has called this the "blue sky effect". But there is more to the story which you can read about in: Simoncelli, E. P., & Olshausen, B. A. (2001), and Simoncelli (1999, 2003).

A big take-home message is that: ***most natural images are decidedly non-Gaussian.***

### Can the above empirical distribution be fit by a Gaussian, a Laplacian?

---

In[63]:=
```
PDF[LaplaceDistribution[υ, β], x]
```

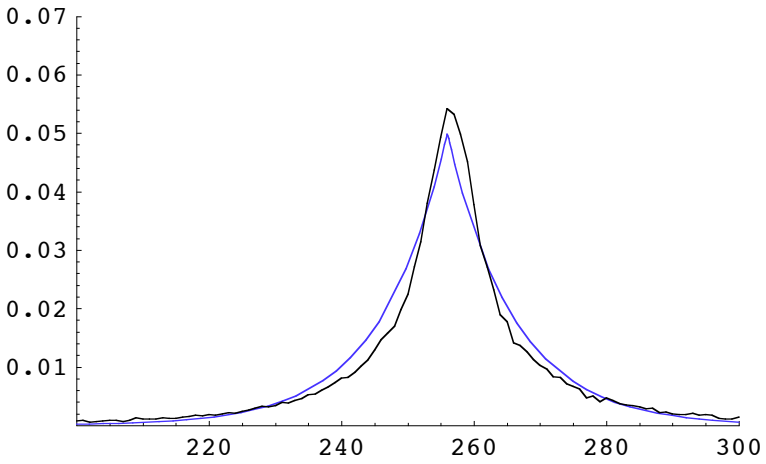
Out[63]=
$$\frac{e^{-\frac{(x-\upsilon)\,\mathrm{sgn}(x-\upsilon)}{\beta}}}{2\beta}$$

■ **Here are some results with this kernel**

In[64]:=
```
kern = {{1, -1}, {1, -1}, {1, -1}}
```

Out[64]=
$$\begin{pmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{pmatrix}$$

In[67]:=
```
theoreticalhistg = Plot[PDF[LaplaceDistribution[256, 10], x],
   {x, 200, 300}, PlotRange → {{200, 300}, {0, .07}}, PlotStyle → Hue[0.7],
   DisplayFunction → Identity];
Show[theoreticalhistg, empiricalhistg, DisplayFunction → $DisplayFunction];
```




If not, try the **generalized laplace distribution**, with a p value between 0.5 and 0.8 (Simoncelli, 1999);

```
pdf[x_, s_, p_] := Exp[-Abs[x / s]^p] / (Gamma[1 / p] * 2 * s / p);
```

### Plot the histogram for a center-surround filtered natural image

---

```
kern = {{0, -1, 0}, {-1, 4, -1}, {0, -1, 0}};
diffgranite = ListConvolve[kern, granite];
```

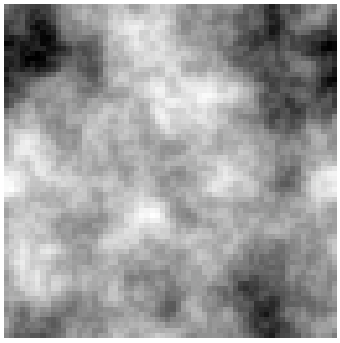
### Plot a difference histogram for a random gaussian fractal image

---

The next lecture shows how to introduce simple correlations in an image using a fractal model.

■ **Initialize gaussianfractalimage by executing the following cell**

```
ListDensityPlot[gaussianfractalimage, Mesh → False, Frame → False];
```



**Even tho' the above image isn't white gaussian noise, it is gaussian. If it is, a linear combination of its pixels should still be gaussian. Check this out using one of the above difference kernels.**

---

## Next time

Efficient coding: 2nd order statistics and area-based operations

Lateral inhibition does "predictive coding"

Cortical wavelet-like basis sets: sparse, distributed representation that "decorrelates"

Color Trichromacy: decorrelation by principal components analysis

# Appendices

## Breaking an image into a series of subimages

### ■ The input 64x64 image: face

Get dimensions of face

```
width = Dimensions[face][[1]]; hsize = width/2;
height = Dimensions[face][[2]];
(* Short[face,1 check out the first few lines*)
```

Scale so image ranges from 0 to 255.

```
α = 255 / (Max[face] - Min[face]); β = -α Min[face];
face256 = α face + β;
```

Scale face so that the average value is zero, and the r.m.s. contrast is 1:

```
tempm = Mean[Flatten[face]];
tempsd = StandardDeviation[Flatten[face]];
face = (face - tempm) / tempsd;
```

```
nregions = 4;
swidth = width / nregions;
```

```
subface = Table[Take[face256, {i * swidth + 1, i * swidth + swidth},
    {j * swidth + 1, j * swidth + swidth}], {i, 0, nregions - 1},
   {j, 0, nregions - 1}];
```

```
Dimensions[subface]
```

```
{4, 4, 16, 16}
```

## ■ Using Raster[]

```
Show[Graphics[Raster[0.5 face]], AspectRatio → 1];
```

## ■ Exporting images

Export["granite64x64.tif",granite,"TIFF"];

## ■ Inverse histogram

```
temp = Table[{fcumulhistogranite[x], x}, {x, 0, 255}];
ListPlot[temp]
```

## ■ Color images

Read in: ColorAlpine256x256.jpg

```
coloralpine = Import[Experimental`FileBrowse[False]];
```

```
Show[coloralpine];
```

Convert RGB to graylevel by taking a weighted average of the three color channels:

$$\texttt{grayalpine = Map}\left[\frac{\texttt{0.3 \#[[1]] + 0.6 \#[[2]] + 0.1 \#[[3]]}}{\texttt{255}}\ \texttt{\&, N[coloralpine[[1, 1]]], \{2\}}\right];$$

```
ListDensityPlot[grayalpine, Mesh → False, Frame → False];
```

**Try histogram equalization with a different image. To redefine granite, first making this cell evaluatable, and then initialize it (**Graygranite256x256.jpg **)**

## References

Atick, J. J., & Redlich, A. N. (1990). Towards a theory of early visual processing. Neural Computation, 2(3), 308-320.

Atick, J. J., & Redlich, A. N. (1992). What does the retina know about natural scenes? Neural Computation, 4(2), 196-210.

Buchsbaum, G., & Gottschalk, A. (1983). Trichromacy, Opponent Colour Coding and Optimum Information Transmission in the Retina. Proceedings of the Royal Society, London B, 220, 89-113.

Gopen, George D. & Swan, Judith A. The Science of Scientific Writing. American Scientist, 78, 550-558.

Hacker, Diana. A Pocket Style Manual 2nd Edition. Bedford Books. Scientific American/St. Martin's College Publishing Group.

Kersten, D. J. (1987). Predictability and Redundancy of Natural Images, Journal of the Optical Society of America A, 4, 2395-2400.

Laughlin, S. B. (1981). A simple coding procedure enhances a neuron's information capacity, Z. Naturforsch.

Laughlin, S. B., de Ruyter van Steveninck, R. R., & Anderson, J. C. (1998). The metabolic cost of neural information. *Nat Neurosci, 1*(1), 36-41.

Linsker, R. (1990). Perceptual neural organization: some approaches based on network models and information theory. Annual Review of Neuroscience, 13, 257-281.

Olshausen, Bruno A., Field, David J. (2000) Vision and the coding of natural images. American Scientist, 88, 238-245.

Simoncelli, E. P., & Olshausen, B. A. (2001). Natural image statistics and neural representation. *Annu Rev Neurosci, 24*, 1193-1216.

Simoncelli, E. (1999, July, 1999). *Modeling the Joint Statistics of Images in the Wavelet Domain*. Paper presented at the Proc. SPIE 44th Annual Meeting, Denver, CO.

Simoncelli, E. P. (2003). Vision and the statistics of the visual environment. *Curr Opin Neurobiol, 13*(2), 144-149.

Srinivasan, M. V., Laughlin, S. B., & Dubs, A. (1982). Predictive coding: A fresh view of inhibition in the retina. Proceedings of the Royal Society London B, 216, 427-459.

van Hateren, J. H., & van der Schaaf, A. (1998). Independent component filters of natural images compared with simple cells in primary visual cortex. *Proc R Soc Lond B Biol Sci, 265*(1394), 359-366.

http://www.cis.upenn.edu/~eero/ABSTRACTS/simoncelli90-abstract.html

http://library.wolfram.com/howtos/images/#histograms