Computational Vision
U. Minn. Psy 5036
Daniel Kersten
Lecture 3: The Ideal Observer

■ **Initialize standard library files:**

In[24]:=
```
<< "BarCharts`";
<< "Histograms`";
```

Goals

Last time

■ **Limits to light discrimination**

In the famous experiment of Hecht, Schlaer, and Pirenne, measurements were made of the proportion of times observers saw a dim small briefly flashed spot of light. Under the right conditions, a person can detect an amazingly small number of photons. Perhaps even more remarkable was the discovery that single photoreceptors are capable of transducing single photons. We singled out the problem of variability in the photon emissions and absorptions as a key computational problem to understand.

Why was the subject response curve sigmoidal? If there was a specific light intensity (or number of photons) at transition from invisible to visible, we should see a "step" function--but we don't. The answer is *variability or "noise"*. But the variability may be due to several causes: the statistics of photon emission and absorption we've mentioned, "noisiness" in human responses, or in neural transmission. The physical limits to visual detection and discrimination can be traced to the stochastic nature of photon absorption and emission. This stochastic nature or variability is a consequence of the particle nature of light. A careful understanding of these limits in conjunction with psychophysical measurements is an essential component to understanding human sensitivity. Let's try to understand how to model variability in the physical stimulus and thereby understand the ideal observer's limitations. Eventually we will return to the question of what are the primary sources of variability in the human discrimination of light intensity. The theory will provide a springboard to analyze the limits of human ability to do other more complex tasks.

The computational theory of signal detection and estimation was developed in the 1940's and 1950's. More information about **Signal Detection Theory (SDT)** can be found in a now classic book by Green, D. M., & Swets, J. A. (1974). Treatments from an engineering perspective can be found in Van Trees, H. L. (1968) and Poor (1994).

Although, Hecht et al. measured thresholds for absolute detection; now we will study discrimination because it makes the key points clearer and is more general.

Introduced basic strategy of our approach to vision:

A: Understand the information processing problem

1) Generative modeling--Understand the image (data) formation process: S -> I

e.g. for light detection, S is the switch value, which can take on two values: "on" or "off", and I is the number of photons.

The generative model was a "Poisson process". Of particular importance is noting that there is variability in the generative process itself.

2) Inference/estimation modeling--Understand how to best guess causes of data: I -> S'

e.g. I is a measurement of the number of photons, and S' is the guess of the switch value. In general, the observer introduces additional variability when making an inference.

B: Understand how well human vision handles the information (I->S''), and from there how it works.

Generative model terminology: Some conceptually related terms:

S -> I

(Causes, scene properties, signals, hypotheses, states of the world) -> (data, image intensities, image measurements, features)

Today

I am going to introduce some of the key concepts in Signal Detection Theory (SDT) and ideal observer analysis in the context of light intensity discrimination. The essence of the ideal observer approach is to ask: What are the theoretical limits to performance on a task? What is the optimal or ideal strategy for this task? In particular for our example, what are the limitations on the discrimination of light levels that any observer must face?

Our goals are:

o Practice some *Mathematica* tools useful for modeling variability and ideal decision making.

o Intuitive introduction to the basics of probability distributions, cumulative distributions.

o Introduce the idea of an **ideal observer** that models "external variability" and makes the best guesses, in the sense of minimizing error rate.

Our generalization of the ideal observer assumes that to minimize error, a decision should choose the most probable signal (e.g. switch setting) given a measurement. The jargon is that the ideal makes a decision that maximizes the "posterior probability".

o Introduce the yes/no task, "hits" and "false alarms" (also called "false positives"). Hit and false alarm rates completely characterizes the ideal's performance (Recall Hecht et al.'s experiment only reported hit rate, the "proportions of yes's" when the light was flashed). We'll also define "correct rejections" and "misses".

Motivation -- DEMO : competing with an ideal observer

Preview of needed concepts

Basic *Mathematica* concepts

- **Built-in functions.** *Mathematica* has a very large library of built-in functions. They all begin with an uppercase letter. You can get information about a function through the **Help Browser** under the **Help** menu item. You can also get, help on-line, e.g. for the exponential of a function, or for plotting graphs:

In[37]:=
```
?Exp
```

Exp[z] is the exponential function. ≫

Exp[z] is the exponential function.

```
?Plot
```

Plot[f, {x, x_{min}, x_{max}}] generates a plot of f as a function of x from x_{min} to x_{max}. Plot[{f_1, f_2, …}, {x, x_{min}, x_{max}}] plots several functions f_i. ≫

Plot[f, {x, xmin, xmax}] generates a plot of f as a
function of x from xmin to xmax. Plot[{f1, f2, ...},
{x, xmin, xmax}] plots several functions fi.

If you type two question marks before a function, ??Plot, you'll get more information. Try it. What does the **RandomReal** function do?. **Add-in** function, such as **BarChart[]** need to be loaded in from standard packages (e.g. as above, << "BarCharts`")

- **Defining functions.** You can define your own functions. Here is a gaussian function:

In[38]:=
```
gauss[x_, mean_, std_] := Exp[-((x - mean)^2) / (2 std^2)] / (std Sqrt[2 Pi]);
```

And you can view the definition in standard form by evaluating:

In[39]:=
```
gauss[x, μ, σ]
```

Out[39]=
$$\frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}\ \sigma}$$

And you can check a your own definition:

In[40]:=
```
? gauss
```

The underscore, **x_ is important** because it tells Mathematica that x represents a slot, not an expression. Note that we've used a colon followed by equals (:=) instead of just an equals sign (=). When you use an equals sign, the value is calculated immediately. When there is a colon in front of the equals, the value is calculated only when called on later. So here we use := because we need to define the function for later use. A double equals, ==, is a relational operator used for testing whether an equation is true or not.

Also note that we could define our function with **N[]**. *Mathematica* trys to keep everything exact as long as possible and thus will try to do symbol manipulation if we don't explicitly tell it that we want numbers. Try it:

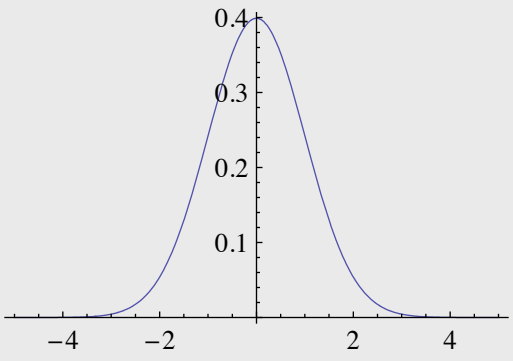
In[41]:=
```
ngauss[x_, mean_, std_] :=
  N[Exp[-((x - mean)^2) / (2 std^2)] / (std Sqrt[2 Pi])];
ngauss[x, u, σ]
```

Out[42]=
$$\frac{0.398942\ 2.71828^{-\frac{0.5\,(x-1.\,u)^2}{\sigma^2}}}{\sigma}$$

■ **Graphics.** Let's plot the graph of the gaussian function:

In[43]:=
```
Plot[gauss[x, 0, 1], {x, -5, 5}]
```

Out[43]=



Lists. We are going to do a lot of work with lists, in particular with vectors and matrices to represent images. Sometimes we'll use vectors to represent 2D images and use matrices to do transformations on those images. Other times, we'll use matrices to represent images. Here is a four-dimensional vector which we'll call **x**. **x** could represent the pixel intensities in an image, or the input signal strengths at 4 receptors:

```
x = {2,3,0,1};
```

By ending a line with a semi-colon, you suppress the output after hitting the return key. Now let's make another vector, this one will be a list of "weights", say, representing the efficiency with which the inputs are weighted before being summed in the visual receptive field:

```
w = {2,1,-2,3};
```

Now the output of a model neuron that simply takes a weighted sum of the inputs is just the dot product of the input with the weights:

```
y = w.x
```

```
10
```

This kind of operation is sometimes referred to as a "cross-correlator". It takes a signal **x**, and cross-correlates it with a template, **w**. In your homework you calculate the performance of a cross-correlator. It is one of the simplest measures of similarity between two patterns. A cross-correlator is the simplest model of what is called the receptive field of a neuron. But more on this later.

- **Using Tables to make Lists.** We will need to convenient ways of making vectors and matrices (e.g. in order to synthesize an image or a filter). The **Table[]** function does this for us. For example, make a list whose elements are the squares of the element location.

```
s = Table[x^2,{x, 1, 8}]
```

```
{1, 4, 9, 16, 25, 36, 49, 64}
```

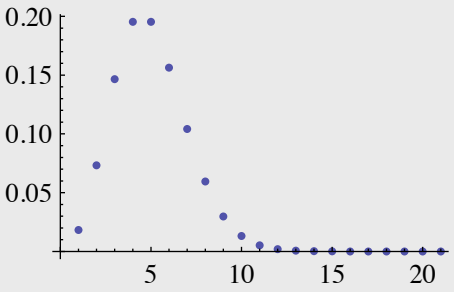
We will use **Table[ ]** quite a bit, because it provides a compact way to iterate function values into an array. To see the result, we use **ListPlot[]** which takes a list of numerical values, rather than a function as the argument for plotting.

```
Clear[x];
poisson[x_,a_] := Exp[-a] a^x / Factorial[x]
poisson[x,a]
p1 = Table[N[poisson[x, 4]], {x, 0, 20}];
ListPlot[p1]
```

Out[46]=

$$\frac{a^x e^{-a}}{x!}$$

Out[47]=

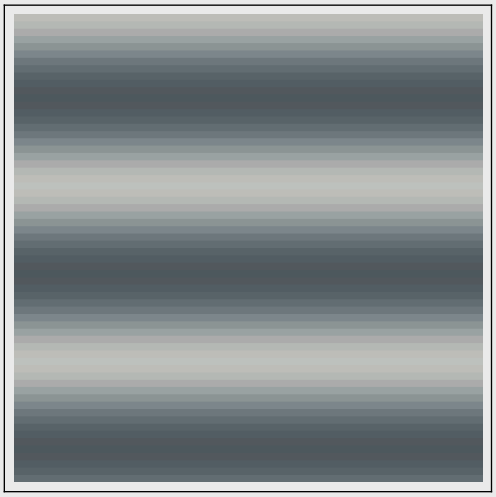


Later, we will use the add-in function, **BarChart[]**. You can also use **Table[]** to make a list of lists, e.g. to define a 2D image. Here is a 64x64 matrix that we can plot up using **ArrayPlot** (you can also use **DensityPlot[]**):

In[48]:=

```
A = Table[Cos[2 π x / 24], {x, 1, 64}, {y, 1, 64}];
ArrayPlot[A, Mesh → False, ColorFunction → "GrayTones",
 PlotRange → {-2, 2}]
```

Out[48]=



■ **Below we are going to use the following functions:**

Built-in Functions: Table, Range, Map, Count, ArrayPlot, Show, PoissonDistribution

We will also use Function[], but in short-hand form.

Add-on function: BarChart

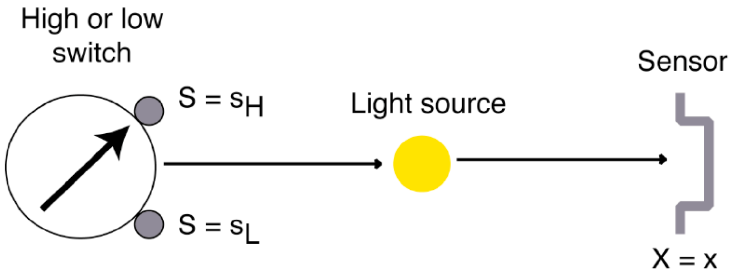
Probability concepts

In the next lecture, we will systematically go over the elements of probability and inference in detail. But today, we will take an intuitive and inductive approach, letting the light discrimination problem motivate the need for the concepts we need. We will talk about: **Random variable**, **Probability distribution**, **Conditional probability**, **Prior probability**, **Posterior probability**, and let the problem context help to develop an understanding.

Modeling external variability: The Ideal Observer

Physical generative model for stimulus: Given the switch setting, what measurements result?

■ **Schematic of set-up for ideal photon counter**



■ **Demonstration of the idea: two switch settings produce various intensities**

As at the end of the last lecture, define a Poisson distribution with a mean of **mean**, with a function to draw a sample from this distribution:

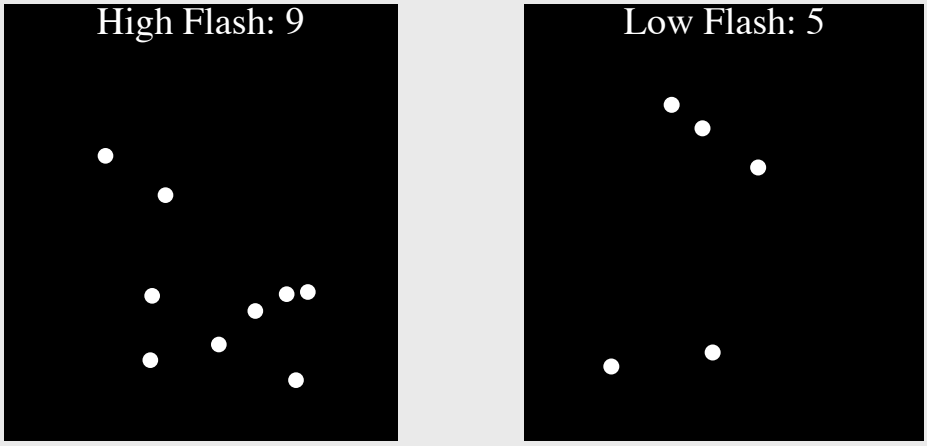
In[49]:=
```
numberofphotons[mean_] := RandomInteger[PoissonDistribution[mean]];
```

Let highmean = 7 for the "high" setting, and lowmean =5 for the "low" setting. Now, turn the switch to "high", draw a

sample, and then to "low" and draw a sample. Let's plot up the two:

In[50]:=
```
highmean = 7; numhighsample = numberofphotons[highmean];
lowmean = 5; numlowsample = numberofphotons[lowmean];
highsample = Table[RandomReal[{0, 1}, 2], {numhighsample}];
lowsample = Table[RandomReal[{0, 1}, 2], {numlowsample}];
highg = Graphics[{PointSize[0.04], White, Point /@ highsample},
   AspectRatio → 1, Frame → False, FrameTicks → None,
   Background → GrayLevel[0.], PlotRange → {{-0.2, 1.2}, {-0.2, 1.2}},
   PlotLabel → "High Flash: " <> ToString[numhighsample],
   LabelStyle → Directive[White]];
lowg = Graphics[{PointSize[0.04], White, Point /@ lowsample},
  AspectRatio → 1, Frame → False, FrameTicks → None,
  Background → GrayLevel[0.`], PlotRange → {{-0.2`, 1.2`}, {-0.2`, 1.2`}},
  PlotLabel → "Low Flash: " <> ToString[numlowsample],
  LabelStyle → Directive[White]];
Show[GraphicsRow[{highg, lowg}, Spacings → Scaled[0.3`]],
 Frame → False]
```

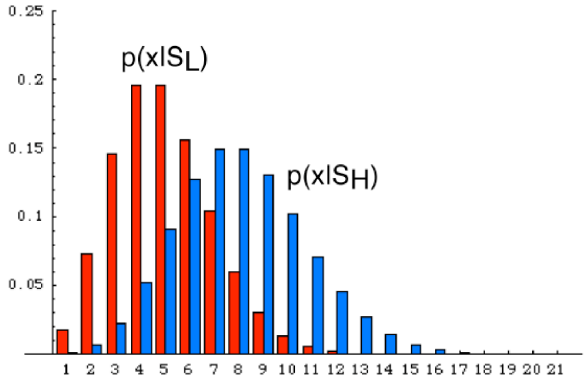
Out[55]=



■ Histograms give an estimate of the conditional probabilities

We use *Mathematica* to study the properties of photon (or dot) counts. Imagine first that we do an experiment as described above.

First set the light level to S_L. Now we begin to send flashes by opening and closing the shutter. No matter how hard we try to make the apparatus perfect, we would discover that the photon counter doesn't always record the same number of photons for each flash. Then we do the same for the switch setting, S_H. So, we compile a histogram of photon counts and find something like this:



The main observation is: photon detection is inherently statistical, and in fact because of the independence of photon absorption, the histogram can be modeled as a *Poisson distribution*. As the number of samples (i.e. test flashes in our case) grows, the better the model fits the histogram data. The Poisson distribution is characterized by the mean (or expected) number of photons flashed when the switch is set to S_L. The spread in the distribution (measured by the standard deviation) is equal to the square root of the mean. As above, the probability of k photons being detected for a Poisson distribution is given by:

```
a^k/(E^a*k!)
```

with a mean level and variance of a. The standard deviation is the square root of the variance. (There are many standard statistical distributions that describe various kinds of uncertainty.

Let's simulate the process of photon generation and do some histogram counting using *Mathematica*.

■ Simulation of the generative process-- Photon counts, and histograms corresponding to high and low switch settings

We compile a histogram of photon counts conditional on the switch setting. Define high and low mean models:

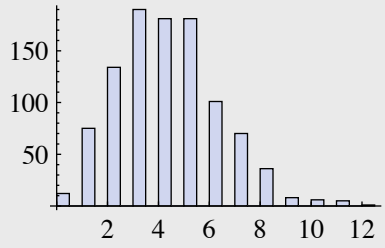
In[56]:=
```
highpdist = PoissonDistribution[7];
lowpdist = PoissonDistribution[4];
nsamples = 1000;
```

Draw nsamples (e.g. = 1000 flashes) for each switch setting.

In[59]:=
```
sample[ntimes_] :=
    Table[Random[lowpdist],{ntimes}];

zlow = sample[nsamples];
domain = Range[0,15];
lowFreq = N[Map[Count[zlow,#]&,domain]/Length[zlow]];
```

In[63]:=
```
Histogram[zlow]
```

Out[63]=



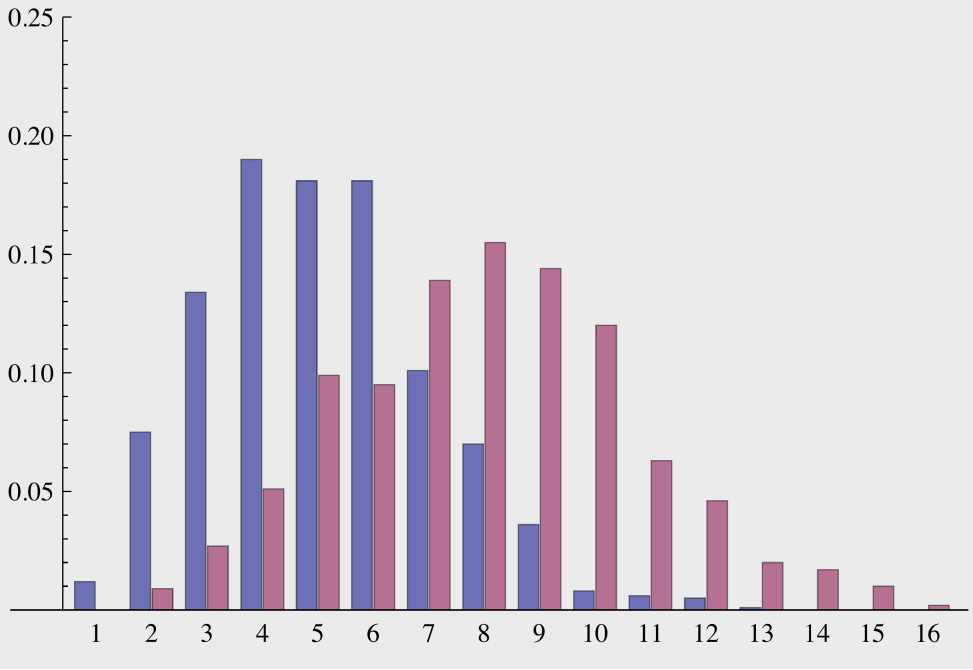
In[64]:=
```
sample[ntimes_] :=
    Table[Random[highpdist],{ntimes}];

zhigh = sample[nsamples];
domain = Range[0,15];
highFreq = N[Map[Count[zhigh,#]&,domain]/Length[zhigh]];
```

Use the Help Browser to look up the definitions of: **Count[ ]**, **Map[ ]**, and **Function[ ]**.

Plot up both of the histograms showing the frequency that the various possible photon counts occur:

In[68]:=
```
BarChart[lowFreq, highFreq, PlotRange → {0, 0.25}]
```

Out[68]=



■ Theoretical -- Compare Poisson probability distributions to high and low simulated histograms

We don't have to remember the formula for the Poisson distribution above. The symbolic definition of a PoissonDistribution is built in for us and can be used to define the probability distribution function (PDF):

```
?? PoissonDistribution
```

PoissonDistribution[μ] represents a Poisson distribution with mean μ. ≫

```
Attributes[PoissonDistribution] = {Protected, ReadProtected}
```

In[69]:=
```
PDF[PoissonDistribution[u], x]
```

Out[69]=
$$\frac{e^{-u}\, u^{x}}{x!}$$

This is the same formula we wrote down earlier. In standard probability notation, if the mean value is a, then the probability of random variable X taking on the value of x photons is:

$$\text{p(X=x photons)} = \frac{a^{x}\, e^{-a}}{x!}$$

The mean or expectation of X is $E(X) = a$, and for a Poisson distribution, the variance, $var(X) = E[(X - a)^2]$ is also equal to a.

The mean gives a measure of "central tendency", and the variance is measure of "spread" of the distribution. (Earlier we wrote our own poisson function simply as: **poisson[x_,a_] := Exp[-a] a^x / Factorial[x]**);

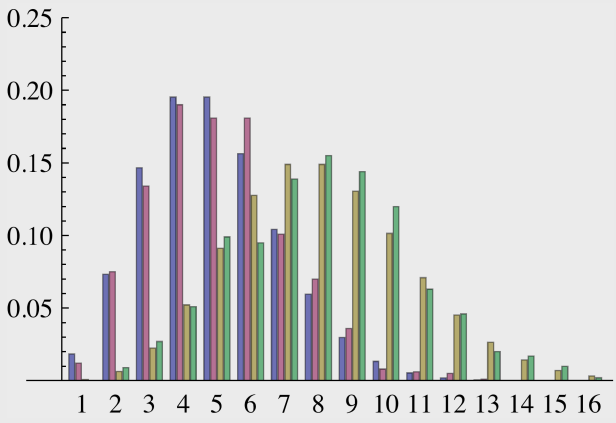
Then we make two lists corresponding to means of 4 and 7:

In[70]:=
```
theorylowFreq = Table[N[PDF[PoissonDistribution[4], x]], {x, 0, 15}];
theoryhighFreq = Table[N[PDF[PoissonDistribution[7], x]], {x, 0, 15}];
```

Compare simulated photon count probabilities with theory predictions. What does the plot look like if nsamples is much smaller, say nsamples = 50?

In[72]:=
```
BarChart[theorylowFreq, lowFreq, theoryhighFreq, highFreq,
 PlotRange → {0, 0.25`}]
```

Out[72]=



You can play with the **nsamples** variable. Note that the more samples we draw, the better the experimental histograms match the theoretical. Try a small value of nsamples, say **nsamples = 100**;

The main point is: photon production is inherently statistical, and in fact because of the independence of photon absorption, the histogram for ideal photon counting can be modeled as a Poisson distribution.

■ Conditional probabilities: a preview

p(x|S_L), p(x|S_H) are conditional probabilities. I.e. they represent the probability of observing a photon count of x, given (or "conditional on") the switch setting H = {S_L or S_H}. We will treat the switch setting as an hypothesis, which is also a random variable, i.e. H.

A note on terminology: p(x|S_L) is the probability of x conditional on S_L. But below when we consider optimal inference, we will talk about likelihood functions, where p(x|S_L) is the *likelihood* of S_L given a measurement of x.

The knowledge from a generative model gets embodied in the conditional probability--given a cause H (where H = S_L or S_H), the conditional probability tells us the probability of a measurement x. In a concise form, p(x|H), is one component of the generative model, and as we see below, p(H) (the "prior") is the other part.

Inference model for light intensity discrimination: Given a measurement, what was the switch setting?

■ Yes/no task

In a yes/no task, we designate one switch position as the "signal" and the other as "not the signal", or "noise". We'll have the observer say "yes" when the switch is set to high.

■ Posterior probability distribution

Given a measurement, what was the switch setting? We've seen that because of variability in the measurement, it is impossible to answer this question correctly every time just from the data. However, we can ask how the decision should be made so that an observer is correct as often as possible. More precisely, we can ask:

What decision strategy will minimize the probability of error?

The observer that achieves a pre-defined optimality criterion like this is called an "*ideal observer*". We'll see other ideal criteria later.

Intuitively one might guess (correctly) that, given a photon measurement count, the ideal observer would pick the switch setting that has the biggest probability of the two. In other words:

If P(S_L| x) > P(S_H|x), choose S_L

otherwise, choose S_H.

P(S_L | x) is "the probability of the switch taking on the value of S_L, conditional on a measurement x", and similarly for P(S_H | x). This rule is called the "maximum *a posteriori*" or MAP rule.

P(S_L | x) and P(S_H | x) are called *posterior* probabilities. Why "posterior"? These probabilities are "informed by the data".

Even without any data measurements, P(S_L) and P(S_H) provides important information for optimal decisions, and are called *prior* probabilities, i.e. the information *prior* to the arrival of the data measurements.

For the problems of visual inference, it is often hard to directly specify the posterior probability, and it is easier to model the prior probabilities and the likelihoods, and then use Bayes rule to relate them to each other:

$$p(H \mid x) = \frac{p(x \mid H)\, p(H)}{p(x)}$$

Bayes' rule is a classic rule in probability theory, attributed to the Rev. Thomas Bayes (1702-1761). We'll use it a lot in this course. After we've reviewed the basics of probability in the next few lectures, you'll be able to easily prove it.

Optimal performance is determined by the posterior probability, and that is why Bayes' rule is of central importance in determining optimal theories of inference. Let's consider the roles the priors and likelihoods play separately.

■ Prior only

Suppose the light transmission switch is set to S_L 4/5 of the time, and to S_H 1/5 of the time.

In[81]:=
```
pH = {4 / 5, 1 / 5};
```

But the photon counter isn't working at all--so there are no data. What is the observers's optimal strategy to guess whether it was the high or bright flash? First we have to specify the receiver's goal, and then it makes sense to answer this question. If the receiver wants to minimize the average probability of error, then it should always pick S_L.

Like the maximum a posteriori rule, the rule that minimizes the error rate is:

choose S_H if p (S_H) > p(S_L)

choose S_L if p(S_L) > p(S_H)

So if p(S_H) > p(S_L), the rule says to always choose S_H. Interesting enough, people often don't do this. With feedback, they tend to match probabilities. This is a phenomenon which has been studied by cognitive psychologists.

Following the above rule, S_H is presented p(S_H) fraction of the time--so this is exactly the error rate, i.e. the fraction of the time that the observer gets the wrong answer using the minimum error rule.

Exercise: What is the probability of error if the probabilities are matched?

■ Likelihood

We use *Mathematica* to define **p(x|H) = poisson[x,a]**, where the hypothesis or switch setting H, determines the average number of samples, x:

In[74]:=
```
poisson[x_, a_] := PDF[PoissonDistribution[a], x]
```

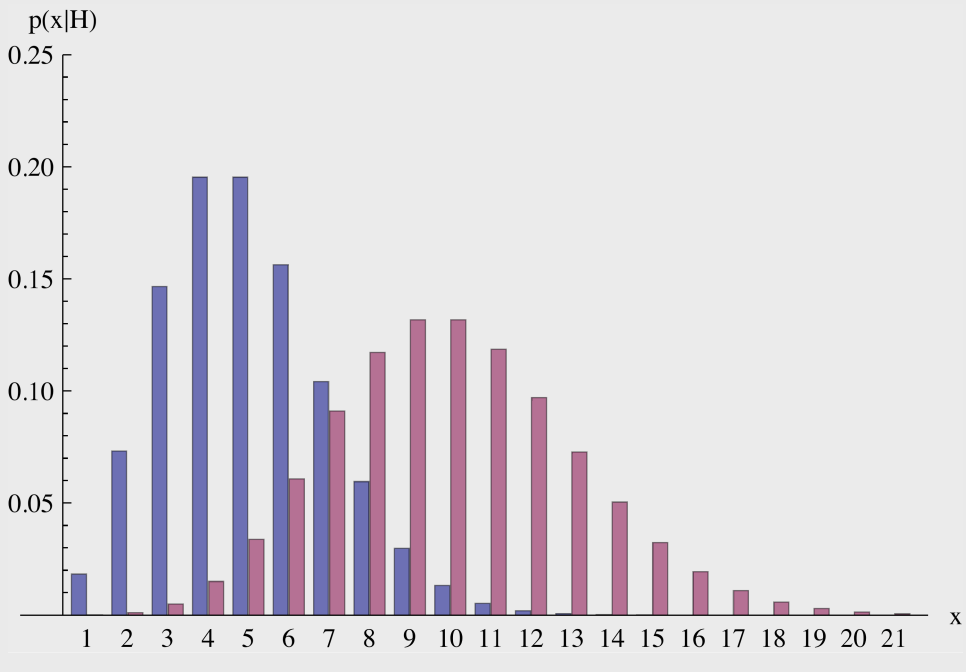
Then we make two lists of probabilities corresponding to means of 4 and 9:

In[77]:=
```
p1 = Table[N[poisson[x, 4]], {x, 0, 20}];
p2 = Table[N[poisson[x, 9]], {x, 0, 20}];
```

The as before, use BarChar to check the distributions.

```
BarChart[p1, p2, PlotRange → {0, 0.25}, AxesLabel → {"x", "p(x|H)"}]
```

Out[79]=



Note that the plot shows the frequencies of observing x photons under the two possible switch settings, low and high.

Let's put the two likelihoods together into one 2x21 array:

In[82]:=
```
poispxH = {p1, p2};
```

Choosing the hypothesis with the largest likelihood will minimize the probability of error when the prior probabilities are uniform--i.e. maximum likelihood decisions are the same as maximum a posterior decisions in this case. But what if the priors are not uniform (i.e. the same constant probability for all outcomes)?

■ Combining the prior and likelihood: Maximum a posteriori (MAP) rule

Suppose (as we assumed for the light discrimination example) that we know the likelihoods: $\mathbf{p(x|}S_L\mathbf{)}$ and $\mathbf{p(x|}S_H\mathbf{)}$. Bayes' rule tells us how to combine them to obtain the a posteriori probability of the hypotheses conditional on the data:

$$p(H \mid x) = \frac{p(x \mid H)\, p(H)}{p(x)} = \frac{p(x \mid H)\, p(H)}{\sum_H p(x \mid H)\, p(H)}$$

(We'll go over this and the basic rules of probability in the next lecture.)

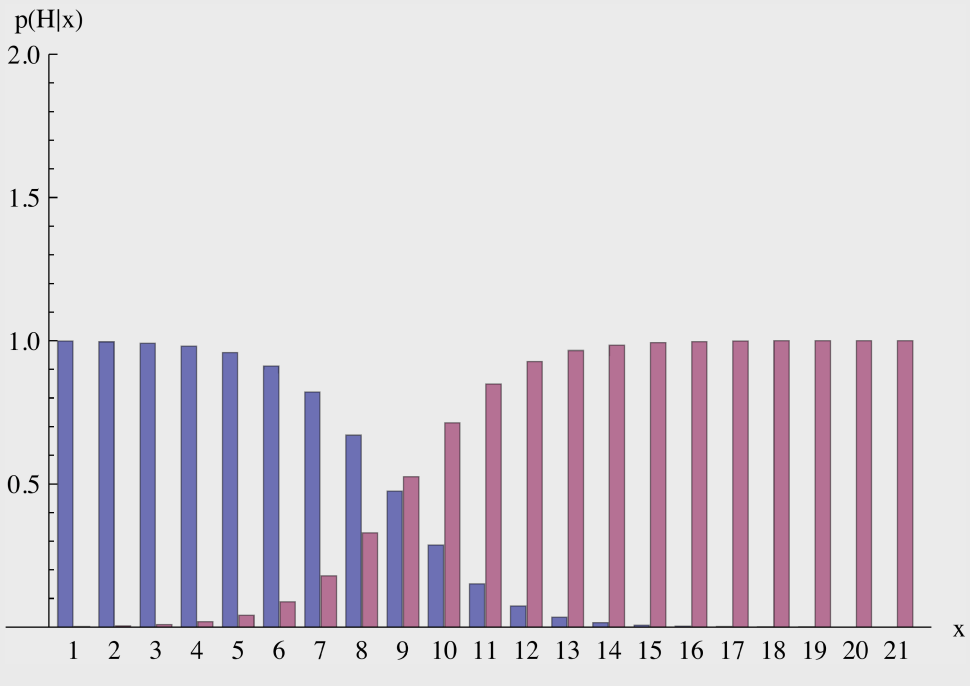
We can calculate the posterior using the following, somewhat opaque, line of *Mathematica*:

In[83]:=
```
posterior = Transpose[Transpose[(poispxH pH)] / Plus @@ (poispxH pH)];
```

Below is a plot of the posterior probabilities, $\mathbf{p(}S_H\mathbf{|x)}$ and $\mathbf{p(}S_L\mathbf{|x)}$, as function of the data **x**:

In[84]:=
```
BarChart[posterior[[1]], posterior[[2]], PlotRange → {0, 2},
 AxesLabel → {"x", "p(H|x)"}]
```

Out[84]=



Note that the plot shows the probability of the switch being low or high, given that x photons were counted. (Why does the posterior plot look so different from the likelihood plot?)

Let's apply the Maximum a posteriori rule (MAP)

For maximum a posteriori estimation, the rule is:

choose S_H if p(S_H|x) > p(S_L|x)

choose S_L if p(S_L|x) > p(S_H|x)

More generally, a function which chooses the value of argument (H = S_L or S_H) that maximizes a function is sometimes written as:

$$\underset{H}{\text{argmax}}\ p(H|x)$$

Suppose 8 photons were counted, the MAP rules says to pick "low", because the blue bar is higher than the red bar.

Compare this plot, which assumed pH = {4/5, 1/5}, with: pH={1/2, 1/2} and a measurement of 8.

■ Let's apply the Maximum likelihood rule

In maximum likelihood estimation, the rule is:

choose S_H if p(x | S_H) > p(x | S_L)

choose S_L if p(x | S_L) > p(x | S_H)

Because of Bayes rule, if the prior probabilities are equal, then MAP is equivalent to maximum likelihood.

p(x | S_H)/p(x | S_L) is called the likelihood ratio.

Suppose 8 photons were counted, what is the maximum likelihood decision? (See p(x|H) plots above, or recompute p(H|x) using pH={1/2,1/2}.)

■ Simple OPTIMAL Rule for light discrimination: Compare the photon count to a criterion.

Let's assume the priors are equal, so we base our decisions using the maximum likelihood rule. We will use *Mathematica* to do some symbolic manipulations to find the transition point in our data measurements (i.e. photon count) where the likelihood goes from favoring a low to a high switch setting. This is where the likelihood ratio is 1, or the log of the ratio is zero.

In[85]:=
```
loglikelihoodratio = Log[poisson[x1, b] / poisson[x1, d]]
```

Out[85]=
$$\log\left(b^{x1}\, d^{-x1}\, e^{d-b}\right)$$

In[86]:=
```
eloglikelihoodratio = PowerExpand[loglikelihoodratio]
```

Out[86]=
$$-b + d + \text{x1}\log(b) - \text{x1}\log(d)$$

There is a simple point to make here--the photon or dot count, x1, is monotonically related to the log likelihood. This means that our intuition that we can't do any better than measuring the dot count was correct. But we need to decide on an appropriate criterion.

In[87]:=
```
eloglikelihoodratio
```

Out[87]=
$$-b + d + \text{x1}\log(b) - \text{x1}\log(d)$$

In[88]:=
```
Solve[eloglikelihoodratio == 0, x1]
```

Out[88]=
$$\left\{\left\{\text{x1} \to \frac{b-d}{\log(b)-\log(d)}\right\}\right\}$$

If d and b are the average number of photons for the low and high switch settings, and the photon counts are distributed according to a Poisson distribution, then minimal error will result on average with the following rule:

Say "high" if

$$\texttt{x} > \frac{\texttt{b-d}}{\texttt{Log[b]-Log[d]}}$$

Say "low" otherwise. The photon count, x, is said to be the *decision variable*. Because it is monotonically related to the likelihood ratio, it is an optimal decision variable. The transition value is called the decision *criterion*:

$$\text{decision criterion} = \frac{\texttt{b-d}}{\texttt{Log[b]-Log[d]}}$$

So now you have all you need to know to write a program that would make optimal decisions about high vs. low--it would simply count photons, and base its decision on whether the count was greater or less than the decision criterion.

■ Summarizing performance

So far so good. But we would like to go a littler farther so that we can calculate theoretically what the average performance of an ideal observer would be. In a yes/no task, an ideal observer can make two kinds of mistakes. It can decide the signal was there when it wasn't (a false alarm or false positive), or it can decide it wasn't there when it was (a miss). These two determine the error rate. An ideal can also be correct in two ways. It can have a hit or a correct rejection. We'll see later that you only need to measure two of these four statistics, because the hit + miss rate have to add up to 100% (if the signal is presented 100 times, you either say yes or no, so the sum of the hit and misses has to be 100).

In the next section you can try competing against the ideal, and have the opportunity to calculate hit and false alarm rates.

Try your luck against the Ideal Observer: Yes/no task

■ Set up a mini-experiment

Let's pretend that photons are like dots. In 1977, Barlow published a paper which did just that. He pointed out that humans can efficiently, but not perfectly, discriminate between patterns of differing dot density.

As above, let's define a Poisson distribution with a mean of **mean**, with a function to draw a sample from this distribution. Let highmean = 220 for the "high" setting, and darkmean =200 for the "low" setting. Then let's initialize an array called **data**, to accumulate the human, ideal, and true answers, and the corresponding hit, false alarm, correct rejection (cr), and miss events.

In[26]:=
```
dotsize = 0.01;

numberofphotons[mean_] := RandomInteger[PoissonDistribution[mean]];
highmean = 220; darkmean = 100;
criterion = N[ - (-highmean + darkmean) / (Log[highmean] - Log[darkmean])];
data = {{"myanswer", "idealanswer", "trueanswer", "ideal hit?",
    "my hit?", "ideal false alarm?", "my false alarm?", "ideal cr?",
    "my cr?", "ideal miss?", "my miss?"}};
```

We'll define a blank screen to turn the display on and off. Set the variable flash to blank.

In[31]:=
```
blank = Graphics[{PointSize[dotsize], Black, Point /@ {{}, {}}},
   AspectRatio → 1, Frame → False, FrameTicks → None,
   Background → GrayLevel[0.0], PlotRange → {{-0.2, 1.2}, {-0.2, 1.2}}];
flash = blank;
```

Now create a window whose contents contain the graphics structure "flash". We make flash dynamic so that the display will get automatically updated whenever we change the value of flash.

In[33]:=
```
nb3 = CreateWindow[DocumentNotebook[Dynamic[flash]],
   ShowCellBracket → False, WindowSize → {300, 300}, WindowElements → {},
   Background → Black, NotebookFileName → "Yes/No Flash Display",
   WindowMargins → {{600, 200}, {120, 120}}];
```

■ Define a trial that shows one flash, either high or low

Now, randomly turn the switch to "high" or "low", draw a sample. We put in a dialog box so that you can input your response (click"yes" for "high" and "no" for "low).

In[34]:=
```
oneflash := Module[{sample, idealanswer},
   whichflash = RandomInteger[{0, 1}];
If[whichflash == 1, numsample = numberofphotons[highmean],
    numsample = numberofphotons[darkmean]];
sample = Table[RandomReal[{0, 1}, 2], {numsample}];
flash = Graphics[{PointSize[dotsize], Red, Point /@ sample},
     AspectRatio → 1, Frame → False, FrameTicks → None,
     Background → GrayLevel[0.0], PlotRange → {{-0.2, 1.2}, {-0.2, 1.2}}];
   Pause[1];
   flash = blank;
myanswer = ChoiceDialog["High dot mean?", {"yes" → 1, "no" → 0},
     WindowMargins → {{570, 100}, {120, 120 + 300}}];
idealanswer = If[numsample > criterion, 1, 0];
trueanswer = whichflash;
data = Append[data, {myanswer, idealanswer, trueanswer,
      If[idealanswer == 1 && trueanswer == 1, 1, ""],
      If[myanswer == 1 && trueanswer == 1, 1, ""],
      If[idealanswer == 1 && trueanswer == 0, 1, ""],
      If[myanswer == 1 && trueanswer == 0, 1, ""],
      If[idealanswer == 0 && trueanswer == 0, 1, ""],
      If[myanswer == 0 && trueanswer == 0, 1, ""],
      If[idealanswer == 0 && trueanswer == 1, 1, ""],
      If[myanswer == 0 && trueanswer == 1, 1, ""]}];];
```

■ Practice run

Let's run 20 practice runs with highmean = 220; darkmean = 100;

In[35]:=
```
For[iter = 0, iter < 20, iter++, oneflash]
```

■ Display the data

In[36]:=
```
data
```

Out[36]=

myanswer	idealanswer	trueanswer	ideal hit?	my hit?	ideal false alarm?	my false alarm?	ideal cr?	m
0	0	0					1	
0	0	0					1	
0	0	0					1	
0	0	0					1	
1	0	0				1	1	
1	0	0				1	1	
1	1	1	1	1				
1	1	1	1	1				
1	1	1	1	1				
0	0	0					1	
0	1	1	1					
1	1	1	1	1				
0	1	1	1					
0	0	0					1	
0	0	0					1	
1	1	1	1	1				
1	1	1	1	1				
0	0	0					1	
0	0	0					1	
0	0	0					1	

■ Analyze the data

Let's drop the table heading stored in row 1, and then transpose the matrix so that the columns become the rows:

```
data2 = Transpose[Drop[data, 1]]
```

$$\begin{pmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & \\
1 & & & 1 & & & 1 & 1 & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & \\
 & & & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & & & \\
 & & & & & & & & 1 & 1 & & & & & & & & 1 & & 1 \\
 & & & & & & & & 1 & 1 & & & & & & & & 1 & & 1 \\
 & & & & & & & & & & & & & & & & & & & \\
 & 1 & 1 & & & 1 & 1 & & & & & & & & & & & & &
\end{pmatrix}$$

Let's use a combination of Map[ ] and Count[ ] (used earlier to make histograms) to count up all occurrences of an event type. So the total for myhits is:

```
Map[Count[data2[[5]], #] &, {1}]
```

{12}

The total times the signal was presented (i.e. switch set to high) is:

```
Map[Count[data2[[3]], #] &, {1}]
```

{16}

So what is my hit rate?

- **Now let's do it for real, where we make the task hard, but doable--100 runs with highmean = 220; darkmean = 200;**

```
highmean = 220; darkmean = 100;
criterion = N[ - (-highmean + darkmean) / (Log[highmean] - Log[darkmean])];
data = {{"myanswer", "idealanswer", "trueanswer", "ideal hit?",
    "my hit?", "ideal false alarm?", "my false alarm?", "ideal cr?",
    "my cr?", "ideal miss?", "my miss?"}};
For[iter = 0, iter < 100, iter++, oneflash]
```

Next time

In the next lecture, we are going to study hit and false alarm statistics using the Gaussian distribution, rather than the Poisson. The Gaussian approximation has the advantage of theoretical convenience and greater generality in extending to other types of signal discrimination and detection problems.

Modeling internal variability of observer: The Human

We'll see how to calculate the signal-to-noise ratio (d') for a discrimination or detection task in terms of hit and false alarm rates, and other measures.

You've probably noted above that when you are a subject in a yes/no task it is hard to figure out the criterion. Feedback helps a human observer learn, but we'll see that a better way of measuring performance is to use the two-alternative forced-choice (2AFC) task.

Compare ideal and human performance in terms of *statistical efficiency*.

References

Applebaum, D. (1996). Probability and Information . Cambridge, UK: Cambridge University Press.

Barlow, H. B. (1962). A method of determining the overall quantum efficiency of visual discriminations. Journal of Physiology (London), 160, 155-168.

Barlow, H. B. (1977). Retinal and central factors in human vision limited by noise. In B. H. B., & F. P. (Ed.), Photoreception in Vertebrates Academic Press.

Duda, R. O., & Hart, P. E. (1973). Pattern classification and scene analysis . New York.: John Wiley & Sons.

Geisler, W. (1989). Sequential Ideal-Observer analysis of visual discriminations. Psychological Review, 96(2), 267-314.

Green, D. M., & Swets, J. A. (1974). Signal Detection Theory and Psychophysics . Huntington, New York: Robert E. Krieger Publishing Company.

Poor, H. V. (1994). An introduction to signal detection and estimation (2nd ed.). New York: Springer-Verlag.

Van Trees, H. L. (1968). Detection, Estimation and Modulation Theory . New York: John Wiley and Sons.

Watson, A. B., & Pelli, D. G. (1983). QUEST: A Bayesian adaptive psychometric method. 33, 113-120.

© 2008 Daniel Kersten, Computational Vision Lab, Department of Psychology, University of Minnesota.
kersten.org