Introduction to Neural Networks
Linear representations of high-dimensional patterns
Lateral inhibition

Introduction

Last time

Developed a generic neuron model and from there built a network. The model was "structure-less, continuous signal, and discrete time".

Basic linear algebra review. Motivated linear algebra concepts from neural networks.

Today

We'll first review some basics of linear algebra with a view to understanding one way to think about the information that a population of neurons encodes about a meaningful pattern. We will look at an example of how neural populations involved in image processing by the brain can be viewed either as analyzing image information or as representing the image.

Then we will change gears a bit, and look at an explanation of a human perceptual phenomenon called Mach bands, that involves a linear approximation based on a real neural network. This is an example of neural filtering found in early visual coding. We will study two types of network that may play a role in the perception of Mach bands: 1) feedforward; 2) feedback. The feedforward network will be a straightforward elaboration of the linear neural network model -- "applied linear algebra". The feedback system will provide our first example of a dynamical system. We will also look at how these networks are related to the neural processing in the horseshoe crab (*limulus*) compound eye. Despite the (apparent) enormous difference between your visual system and that of the horseshoe crab, our visual system shares a fundamental image processing function with that of this lowly crustacean (and virtually all other studied animals that process image patterns).

You will also see how two different mechanisms can produce equivalent input-output behavior.

And how a choice of parameters can produce winner-take-all behavior in a dynamic recurrent network.

Basis sets

Suppose we have a vector, **g**, that represents the inputs to a network of neurons. To be concrete, as earlier, assume our input vectors live in an 8-dimensional space. Further, suppose that each of 8 neurons receives information from all 8 inputs. Consider two different ways in which the weights (or receptive fields, if a sensory neuron) might be organized. The collection of weights could be cartesian:

```
In[267]:= u1 = {1,0,0,0,0,0,0,0};
          u2 = {0,1,0,0,0,0,0,0};
          u3 = {0,0,1,0,0,0,0,0};
          u4 = {0,0,0,1,0,0,0,0};
          u5 = {0,0,0,0,1,0,0,0};
          u6 = {0,0,0,0,0,1,0,0};
          u7 = {0,0,0,0,0,0,1,0};
          u8 = {0,0,0,0,0,0,0,1};
          cartesianset = {u1,u2,u3,u4,u5,u6,u7,u8};
```

or they could be like the walsh set:

```
In[276]:= nwalshset = HadamardMatrix[8];
          {w1, w2, w3, w4, w5, w6, w7, w8} = nwalshset;
```

Both of these sets of vectors have the properties: 1) each vector in the set is orthogonal to every other; 2) the vector lengths are all equal to 1; 3) The set of 8 vectors is said to be *"complete",* in that they completely "*span"* the 8 dimensional space. Some basis sets are "over-complete", e.g. if you added 9th vector to the set. The 9th vector would be redundant in that you could always express it in terms of the others. In this case the set of 9 is said to be "linearly dependent". Our original set(s) of 8 are linearly independent. (See previous lecture.)

In general, a set of vectors used in a linear weighted sum to represent an N-dimensional vector pattern is called a *basis set* for that N-dimensional space.

It is pretty clear that given any vector whatsoever in 8-space, you can specify how much of it gets projected in each of the eight directions specified by the cartesian set of unit vectors u1, u2, ...u8. But you can also build back up an arbitrary vector by adding up all the contributions from each of the component vectors. This is a consequence of vector addition and can be easily seen to be true in 2 dimensions.

We can verify it ourselves. Pick an arbitrary vector **g**, project it onto each of the basis vectors, and then add them back up again:

```
In[278]:= g = {2,6,1,7,11,4,13, 29};

In[279]:= (g.u1) u1 + (g.u2) u2 + (g.u3) u3 + (g.u4) u4 +
          (g.u5) u5 + (g.u6) u6 + (g.u7) u7 + (g.u8) u8

Out[279]= {2, 6, 1, 7, 11, 4, 13, 29}
```

▶ 1. Exercise

Reconstruction works for any orthogonal basis set, such as {v1,v2,...v8}. What happens if you project **g** onto the normalized Walsh basis set defined by **{w1,w2,...}** above, and then add up all 8 components?

```
In[14]:= (g.w1) w1 + (g.w2) w2 + (g.w3) w3 + (g.w4) w4 +
         (g.w5) w5 + (g.w6) w6 + (g.w7) w7 + (g.w8) w8
```

Looks complicated, but if you simplify it:

```
In[280]:= Simplify[%]
Out[280]= {2, 6, 1, 7, 11, 4, 13, 29}
```

you get back the original vector.

{2, 6, 1, 7, 11, 4, 13, 29}

Spectrum

The projections, {**g.w**$_\mathbf{i}$} are sometimes called the **spectrum** of **g**. This terminology comes from the Fourier basis set used in Fourier analysis. Also recall white light is composed of a combination of light of different frequencies (colors). A discrete version of a Fourier basis set is similar to the Walsh set, except that the elements fit a sine wave pattern, and so are not binary-valued.

Representations of patterns

Motivation: representation of high-dimensional visual information

Analyzing image input

In Lecture 4, we noted that the simplest feedforward neural network can often be usefully approximated over a linear regime by a matrix operation:

$$y_i = \sigma\left(\sum_{j=1}^{n} w_{ij}\, x_j\right) \sim \sum_{j=1}^{n} w_{ij}\, x_j$$

In vector-matrix notation as: y = W.x, where W is a matrix and x and y are vectors.

It is hard to believe that much can be done with something so simple. The reason is that the complexity lies in the potentially rich structure of the connections, W, and how they map input patterns to useful and meaningful outputs. We turn to visual coding to motivate this.

To help keep in mind that we are using an image processing example, we'll change notation so that:

$y_i \to s_i$ and $x_i \to$ flattened representation of image intensities $I(x, y)$. Flattened meaning that we re-represent the 2D input image I(x,y) by mapping it into a 1D vector, **I**, that we label in bold and use later.

$I(x, y)$ can be thought of as the intensity at pixel location (x,y). Then a visual transformation (or filtering) of I(x,y) can be represented by a mapping of image intensities (or input activities I(x,y)) to firing rates s_i:

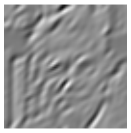
$$s_i = \sum_{x,y} W_i(x,y) I(x,y).$$

W_i(x,y) represents the effective synaptic weights (or spatial receptive field) of neuron i.

But what does W_i(x,y) mean? What does it do to the incoming image? One example in vision is that neurons in primary visual cortex can be modeled by a set of receptive field weights, W_i(x,y) which acts as a spatial filtering operation that amplifies responses to edges of different orientations indexed by i. In fact all the built-in function GaborFilter[] does is exactly that. Under the hood, GaborFilter[ ] treats the input image as a vector, and figures out what values to put in a matrix to multiply it with to amplify spatial changes in intensity, i.e. the big values corresponding to the strengths of edges in a particular direction.

In[281]:=
```
GaborFilter[
```

```
, 4, {1, 1}] // ImageAdjust
```

Out[281]=


In other words, switching to a vector representation of input **I** and a set of receptive fields **W**, the activity of the population of neurons can be written:

s = **W**.**I**

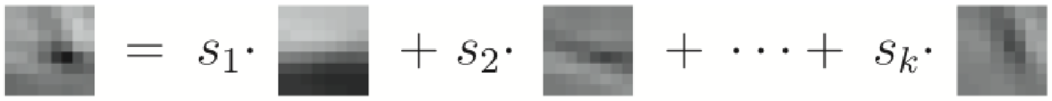
where again s and I are vectors, and W is a matrix. s= W.I describes how a set of effective weights (e.g. synaptic weights or "receptive field" weights) turns the image input I into a pattern of neural responses--a "neural image" s (equivalent to the notation, where now s=y, and x=I).

Synthesizing image input

It is useful to distinguish between 1) the weights $W_i(x, y)$ that analyze an image to produce a pattern of neural activity s_i and 2) the “features”, call them $A_i(x, y)$ that *explain, generate, or synthesize* an input *I*.

Assume that an input image I(x,y) can be represented (synthesized) by a linear combination of a limited set of “features”, A_i(x,y) :

$$I(x,y) = \sum_{i=1}^{n} A_i(x,y) s_i$$



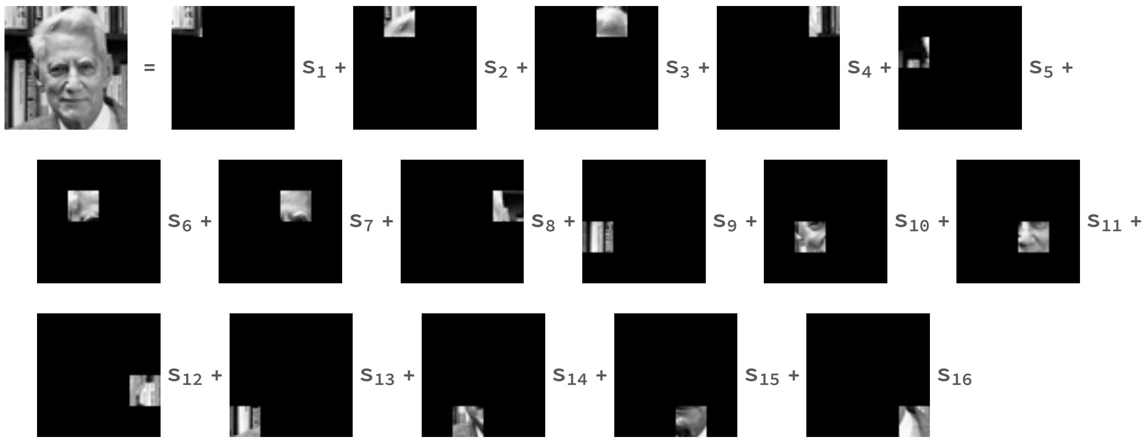
Below we tie these “basis images” to the columns of a feature matrix.

For the moment, think of the A’s as providing a dictionary or vocabulary useful to describe any image in some perhaps restricted set, say “human faces” or “natural images”, in terms of a weighted sum. The A’s are a special collection of image patches or “dictionary” for which any image (in the set of interest) can be efficiently composed. Fourier synthesis is a special example where an arbitrary image is expressed as the sum of sinusoidal images weighted by a scalar (s_i) that says how much of each needs to get added to equal the arbitrary image. The collection of sinusoidal images is the basis set.

Such a model is sometimes called a generative model for the patterns in the set of interest. (The term “generative” is more specifically used in the context where one also has a probabilistic description of generating parameters, s_i, *and* the outputs).

See: Hyvärinen, A. (2010). Statistical Models of Natural Images and Cortical Visual Representation. Topics in Cognitive Science, 2(2), 251–264. doi:10.1111/j.1756-8765.2009.01057.x

Here is a simple example where the features are non-overlapping 16 x 16 pixel patches:



With the above sixteen highly specific dictionary elements (each of which is 16x16 pixels), we have only 16 degrees of freedom, so would be very limited. You couldn’t “span” the space of human faces with this set.

However, you could expand the set to have many more patches, no longer orthogonal to each other, and cover a larger range of the space of faces. At the far extreme, you could have a unique basis vector for every face. Then the s_i value would be 1 for the ith face, and zero for all the others. But this set would be vastly “overcomplete”. It would be a grandmother cell code. Alternatively, note that if the patches were each just 1x1 pixels, you would have a cartesian basis set and could synthesize any 64 x 64 image (within the range of values of s_i). But now it may be too general--the human visual system doesn’t need to represent any image (think of huge space of TV “snow”). Getting the “right” number of basis vectors to efficiently represent a set of interest has been an active topic of research in signal processing for many decades. In past studies, the “set of interest” has been defined to be “the set of human faces”, “the set of english letters”, and even “the set of natural images”. Efficiency can be interpreted in terms of “sparseness”--i.e. the number of active features to describe any image in the set of interest, but more on this later. (There are small basis sets, around the same order of 16 elements, that can do quite well on databases on human faces; cf. Sirovich, L., & Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. Journal of the Optical Society of America A, 4(3), 519–524.).

Now let’s reformulate the example in terms of simple vectors and matrices. We “flatten” I(x,y) and each A_i(x,y) to turn them into vectors and then write:

I = A.s,

where we identify the features A_i(x,y) as the "columns" of A, $\{A_j\}$, and the elements of s, $\{s_j\}$, represent the activity of the neurons representing how much each of those features contribute to (or explain) **I**:

I = s1 * A_1 + s2 *A_2 +...

Again, we use the word "features" because we think of the input as being made up of (linear) combinations of these *particular* patterns in the world.

Here is an example in which there are 5 feature vectors for an 4-dimensional input space:

$$\begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{pmatrix} . \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix} = s_1 . \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{pmatrix} + s_2 . \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \\ a_{42} \end{pmatrix} + \ldots = s_1 . A_1 + s_2 . A_2 + \ldots$$

where

$A_i = \{a_{1\,i}, a_{2\,i}, a_{3\,i}, a_{4\,i}\};$

So the actual neural representation of an image I is only implicit in the firing rates and a definition of what each A_j means. In other words, the activities s are a simple "code" for I which can be recovered using the above formula...IF we know A. Neural models of image representation in the primary visual cortex have been analyzed in terms of whether the high-dimensional images received (at the retina) are projected into lower or higher dimensional spaces in cortex, and what the consequences might be for biological image processing (Hyvärinen, 2010). The notion of a linear representation of features for an input signal class has had a major impact on theories of neural representation in primary visual cortex, as well as image processing more generally.

OK, now let’s make some *strong, but greatly simplifying* assumptions which will allow us to exploit standard results in linear algebra.

From a linear algebra perspective, if I and s have the same number of elements, A is "square", and if A is invertible, then W = A^{-1}. (We’ll cover the definition of an inverse matrix later, but you can think of it as analogous to division for scalars, but for matrices-- i.e. for scalars, if y=A.x, then one could solve for x as x=(1/A).y or x=A^{-1}.y)

Further, there is a theorem that says if the columns of A are orthogonal, then W is just the transpose of A. The transpose $A^T$of a matrix A just turns the rows of A into columns of A^T. This means that the "features" or basis vectors used to represent I (columns of A) are the same as the receptive field weights (rows of W) used to analyze I. Under these assumptions, we can think of the pattern of a receptive field as representing an image feature. Let’s now summarize the basic linear algebra of vector representation in terms of basis vectors.

Mach bands & perception

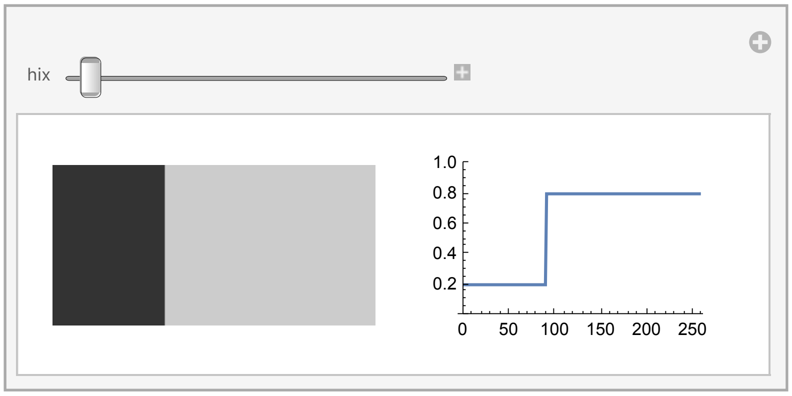
Now let’s take a look at a famous problem in human visual perception and its relation to a linear neural network model developed to explain image processing by the limulus or horseshoe crab. We first go back to the 19th century.

Ernst Mach was an Austrian physicist and philosopher. In addition to being well-known today for a unit of speed, and *Mach's Principle* in theoretical physics, he is also known for several visual illusions. One illusion is called "*Mach bands*". Let's make and experience one.

```
In[283]:= width = 256;
          y[x_, hix_] := Module[{low, hi, lowx},
             low = 0.2; hi = 0.8;
             lowx = .35 * width; Piecewise[{{low , x < lowx}, {((hi - low) / (hix - lowx)) x -
                 ((hi - low) lowx) / (hix - lowx) + low , x >= lowx && x < hix}, {hi, x >= hix}}]];

In[285]:= Manipulate[
           e3 = Table[y[i, hix], {i, 1, width}];
           picture2 = Table[e3, {i, 1, 60}];
           GraphicsGrid[{{Graphics[Raster[picture2, {{1, 1}, {120, 60}}, {0, 1}]], Plot[y[x,
                 hix], {x, 1, width}, PlotRange → {0, 1}]}}], {{hix, 87}, width / 3., width * 3 / 4}]
```

Out[285]=

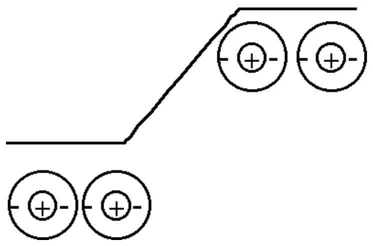


PlotRange is used to scale the brightness.

What Mach noticed was that the left “knee” of the ramp looked too dark, and the right knee looked too bright to be explained by the light intensity. *Objective* light intensity did not predict *subjective* brightness.

(Perceptual psychologists sometimes distinguish between “brightness” and “lightness, cf. Boyaci et al., 2007 and Knill and Kersten, 1994. Lightness and related phenomena involve visual “filling-in”, believed to involve cortical processes, cf. Komatsu et al., 2006. We’ll look at models for filling-in later.)

Mach's explanation in terms of lateral inhibition



Mach’s interpretation was that somewhere in the visual system, there are units whose outputs reflect a process in which the light intensity at a point gets replaced by the weighted sum of intensities at nearby points. The weights have a center-surround organization, in which weights are positive near the point of interest, and negative further away. These so-called center-surround filters are illustrated in the above concentric circles, where + indicates positive weights, and - indicates negative weights. This process is called *lateral inhibition.* If you work through the consequences of a lateral inhibition filtering operation, you will see that there is more inhibition (negative weight contributions) at the left knee, than just to the left. And there is more positive contribution at the right knee, than at the point just to the right. In general, lateral inhibition increases contrast at edges.

Neural basis?
Early visual neurons (e.g. ommatidia in horseshoe crab, ganglion cells in the mammalian retina and even later cells in the lateral geniculate neurons of the thalamus, and some cells in V1 or primary visual cortex of the monkey) have receptive fields with Mach's center surround organization. I.e. approximately circular excitatory centers and inhibitory surrounds. Or the opposite polarity, inhibitory centers and excitatory surrounds.

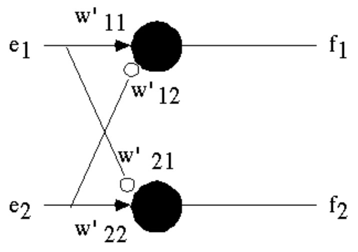
Some history:
Limulus (horseshoe crab)--Hartline won the 1967 Nobel prize for this work that began in the 20's with publications up to the 70's.
(See http://hermes.mbl.edu/marine_org/images/animals/Limulus/vision/index.html).
Frog -- Barlow, H. B. (1953). Summation and inhibition in the frog's retina. J Physiol, 119, 69-88.
Cat --S. W. Kuffler (1953). Discharge patterns and functional organization of mammalian retina . Journal of Neurophysiology, 16:37--68.

Feedforward model

We'll consider two types of models for lateral inhibition: feedforward and feedback (in our context, "recurrent lateral inhibition").
Let's look at a simple feedforward model. Let

f = *w*′.e

where **e** is a vector representing the input intensities (the **e3** or **y[]** values in the above demo), ***w***′ is a suitably chosen set of weights (i.e. excitatory center and inhibitory surround as shown in the above figure), and **f** is the output. The connectivity for two neurons would look like this:

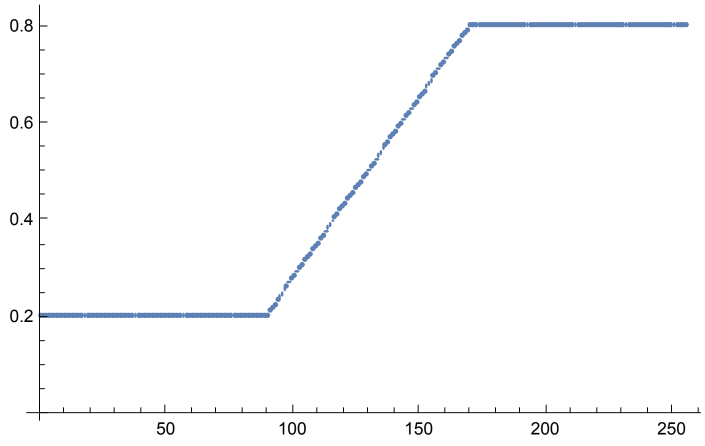


but of course you'd have to add more neurons to get the circularly symmetric weights described by Mach.

Programming implementation

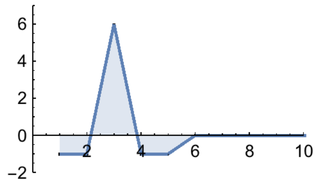
Because changes in the stimulus are one-dimensional, we'll simulate the response in one dimension. We specify the input vector **e** as above:

In[286]:=
```
e4 := Table[y[i, 170], {i, 1, width}];
ListPlot[e4]
```

Out[287]=


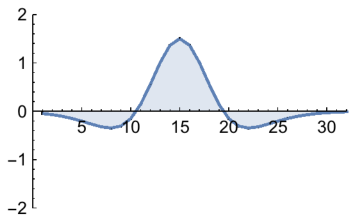
Let's assign weights consistent with Ernst Mach's 19th century hypothesis. The receptive field for one output unit will be represented by: 5 weights, with a center value of 6, and 4 surround values of -1:

In[288]:=
```
wp = Table[0, {i, 1, Length[e4]}];
wp[[1]] = -1;
wp[[2]] = -1;
wp[[3]] = 6;
wp[[4]] = -1;
wp[[5]] = -1;
ListPlot[wp, Joined → True, PlotRange → {{0, 10}, {-2, 7}}, Filling → Axis]
```

Out[288]=


...or we could get a little more sophisticated, and use a common formula to model center-surround organization, the difference of two Gaussian functions with different widths, a so-called “DOG” filter. Here is a specific choice that specifies a wider filter than above:

In[289]:=
```
wp = Table[2.5 * Exp[- ((i - 15) / 4) ^ 2] - Exp[- ((i - 15) / 8) ^ 2], {i, 1, Length[e4]}];
ListPlot[wp, Joined → True, PlotRange → {{0, width / 8}, {-2, 2}}, Filling → Axis]
```

Out[290]=


The plot shows the "center-surround" organization of the filter. Filters of this sort that have an antagonistic center-surround organization are sometimes referred to as "Mexican hat" filters.

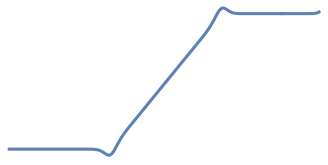
Now assume that all units have the same weights, and calculate the response at each point by shifting the weight filter **wp** right one by one, and taking the dot product with the input pattern **e4**, each time:

In[291]:=
```
response = Table[RotateRight[wp,i].e4,{i,1,width-30}];
```

This way we can mimic the response we want:

In[292]:=
```
ListPlot[response, Joined → True, Axes → False]
```

Out[292]=



Note that we cut the rotation short to eliminate boundary effects.

► 2. Show that you can do this operation as matrix multiplication, where each subsequent row of a matrix **W** is the vector wp shifted over by one.

Convolution

This kind of operation where the same filter gets applied repeatedly at different positions is common in signal processing. It is called discrete *convolution*. (Convolution can be defined in the continuous domain as a specific kind of integration. More later.). Convolution is used in neural network modeling whenever 1) a linear model is reasonable; 2) the same filter gets applied repeatedly across space (or time), as one might expect when processing an image.

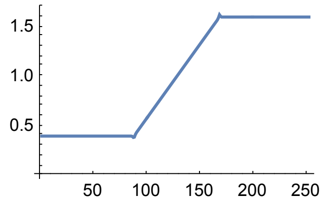
Mathematica has a function **ListConvolve[ ]** that does discrete convolutions. It has additional arguments that allow for handling of the boundaries at the beginning and end of an input vector.

What should you do when the filter gets close to the end of the stimulus? A common default is to let the filter wrap around. Another common solution is to "pad" the ends of e with fixed values, such as zero. What does the retina do?

Here's ListPlot[] with the simpler center-surround receptive field, {-1, -1, 6, -1, -1}. In mathematics, the filter vector used in convolution is sometimes called a "kernel".

In[293]:=
```
ListPlot[ListConvolve[{-1, -1, 6, -1, -1}, e4], Joined → True]
```

Out[293]=



► 3. What is the response to the ramp if the sum of the weights is zero? Build a simple edge detector. Let **kernel={-1,2,-1}** and plot ListConvolve[kernel, e4].

Feedback model: Recurrent lateral inhibition

The above model abstracts away any dependence on the temporal dynamics of lateral inhibitory circuitry.

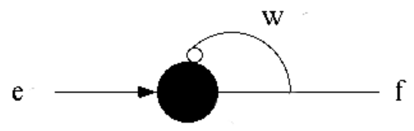
Let’s develop a different, dynamical model for lateral inhibition that includes time and feedback, that is nonetheless still linear, but now captures the behavior of the network as it evolves in time. In other words, we will model the temporal evolution of the state vector given an input.

There is neurophysiological evidence for an implementation of lateral inhibition via feedback, called *recurrent lateral inhibition.*

We’ll first assume time is discrete, and then generalize to continuous time.

Dynamical systems: difference equation for one neuron

State of neuron output **f** at discrete time k.



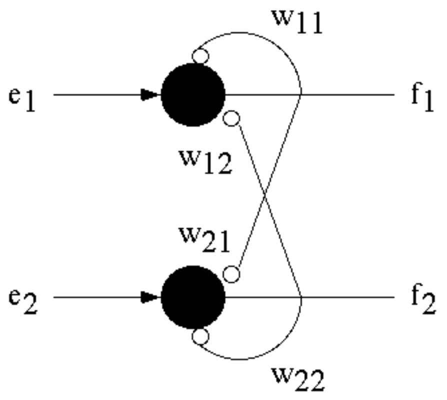
For one neuron, let the output at time interval k+1 be:

$$f[k+1] = e[k] + w\,f[k] \tag{1}$$

► 4. Suppose the initial state f[0] is known and e[k] is zero, can you find an expression for f[k]? What happens if w is less than one? Greater than one?

Dynamical systems: Coupled difference equations for interconnected neurons

Now consider a two neuron system. The formalism will extend naturally to higher dimensions. To keep this simple, the weights for the inputs **e** are fixed at one, but we will specify weights for the newly added feedback connections in order to capture the behavior of lateral inhibition:



Let **e** be the input activity vector to the neurons, **f** is the n-dimensional state vector representing output activity and **W** is a fixed nxn weight matrix. Then for a two neuron network we have:

$$f_1[k+1] = e_1[k] + w_{12}\,f_2[k] + w_{11}\,f_1[k]$$
$$f_2[k+1] = e_2[k] + w_{21}\,f_1[k] + w_{22}\,f_2[k]$$

or in terms of vectors and matrices

$$\begin{pmatrix} f_1[k+1] \\ f_2[k+1] \end{pmatrix} = \begin{pmatrix} e_1[k] \\ e_2[k] \end{pmatrix} + \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \begin{pmatrix} f_1[k] \\ f_2[k] \end{pmatrix}$$

or in summation notation:

$$f_i[k+1] = e_i[k] + \sum_j w_{ij} . f_j[k] \tag{2}$$

or in concise vector-matrix (and *Mathematica*) notation:

$$\mathbf{f}[k+1] = \mathbf{e}[k] + \mathbf{W}.\mathbf{f}[k] \tag{3}$$

where $\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$

This equation is an example of a simple *dynamical* system, with state vector **f**. The state of the dynamical system changes with time (i.e. with each iteration k).

Are there solutions for which the state does not change with time? If there are, these solutions are called *steady state* solutions.

In contrast to the way we set up the weights for the feedforward matrix (which included the forward excitatory weights), we are going to assume later that all of these weights are inhibitory (because we are modeling lateral *inhibition*). The positive contributions come from the input **e**.

Steady state solution is mathematically equivalent to a linear feedforward model

Recall that the feedforward solution is just a matrix operation on the inputs: **f = W'.e**, where the first row of W' has the center-surround organization of the weights, e.g. w1 = {-1,-1,6,-1,-1,0,0,0,0,0,...}, and the second row is shifted: {0,-1,-1,6,-1,-1,0,0,0,0,,...} and so forth.

Let's compare the feedforward model with the recurrent feedback after the latter has "settled down"--i.e. with its steady-state solution.

A steady-state solution simply means that the state vector **f** doesn't change with time:

$$\mathbf{f}[k+1] = \mathbf{f}[k]$$

or in vector and *Mathematica* notation:

$$\mathbf{f = e + W.f}$$

where we drop the index k. Note that by expressing **f** in terms of **e**, this is equivalent to another linear matrix equation, the feedforward solution:

$$\mathbf{I.f = e + W.f,}$$
$$\mathbf{(I - W)f = e,}$$

Assume that we can solve this equation for **f**:

$$\mathbf{f = W'.e},$$

Later when we review the basics of matrix algebra, we'll see that the solution, **W'**, is given by:

$$\mathbf{W' = (I - W)^{-1}}$$

where the **-1** exponent means the inverse of the matrix in brackets. And **I** is the identity matrix, which in two dimensions is: $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

We will review more later on how to manipulate matrices, find the inverse of a matrix, etc.. But for the time being, think of an identity matrix as the generalization of unity: **1**. **1** times **x** returns **x**. And the inverse is the generalization of taking the reciprocal of a number.

The point of the above derivation is that *the steady-state solution for recurrent inhibition (i.e. with feedback) which is equivalent to non-recurrent linear network (no feedback, just feedforward)*.

In general, there may be more than one underlying neural circuit to explain the external behavior of a network. Telling them apart requires doing the right kind of experiment.

Dynamical system with continuous time-- theory of coupled differential equations ("limulus" equations)

In our discussion of the different types of neural models, we noted that continuous time is a more realistic assumption for a neural network. So what if time is modeled continuously, and not in discrete clocked chunks? We'll extend the discrete time model to a continuous time model. But then to simulate the dynamics, we'll have to go back to a discrete approximation, while keeping in mind that the behavior might depend on the temporal grain of our discrete approximation. Temporal grain refers to how finely we chop up time.

The theory for coupled discrete equations

$$\mathbf{f}[k+1] = \mathbf{e}[k] + \mathbf{W}.\mathbf{f}[k]$$

parallels the theory for continuous differential equations where time varies continuously:

$$\frac{d\mathbf{f}}{dt} = \mathbf{e}[t] + \mathbf{W}''.\mathbf{f}[t]$$

(***W***″ is not the same matrix as ***W***.) If you want to learn more about dynamical systems, a classic text is Luenberger, 1979)

Let's see how this might work.

Let **e**(t) be the input activity to the neurons, **f**(t) is the n-dimensional state vector representing output activity, now as a function of time. **W** is a fixed nxn weight matrix. The equation in the previous section is the steady state solution to the following differential equation:

$$\frac{d\mathbf{f}}{dt} = \mathbf{e}[t] + \mathbf{W}.\mathbf{f}[t] - \mathbf{f}[t]$$

(You can see this by noting that as before, "steady state" just means that the values of **f(t)** are not changing with time, i.e. **df/dt = 0**). We are going to develop a solution to this set of equations using a discrete-time approximation.

$$\frac{d\mathbf{f}}{dt} \sim \frac{f(t+\Delta t)-f(t)}{\epsilon}$$

The state vector **f** at time t+Δt ($\epsilon = \Delta t$) can be approximated as:

$$\mathbf{f}(t+\Delta t) \cong \mathbf{f}(t) + \varepsilon[\mathbf{e}(t) + \mathbf{W}.\mathbf{f}(t) - \mathbf{f}(t)]$$

We will fix or "clamp" the input **e**, start with arbitrary position of the state vector **f**, and model how the state vector evolves through time. We'll ask whether it seeks a stable (i.e. steady) state for which **f**(t) is no longer changing with time, **f**(t + Δt) = **f**(t), i.e. when d**f**/dt = 0. In the limit as Δt (or ϵ) approaches zero, the solution is given by the steady state solution of the previous section. But neural systems take time to process their information and for the discrete time approximation, the system may not necessarily evolve to the steady state solution.

Simulation of the dynamics of recurrent lateral inhibition

First we will initialize parameters for the number of neurons (**size**), the space constant of the lateral inhibitory field (**spaceconstant**), the maximum strength of the inhibitory weights (**maxstrength**), the number of iterations (**iterations**), and the feedback delay ϵ:

The input stimulus

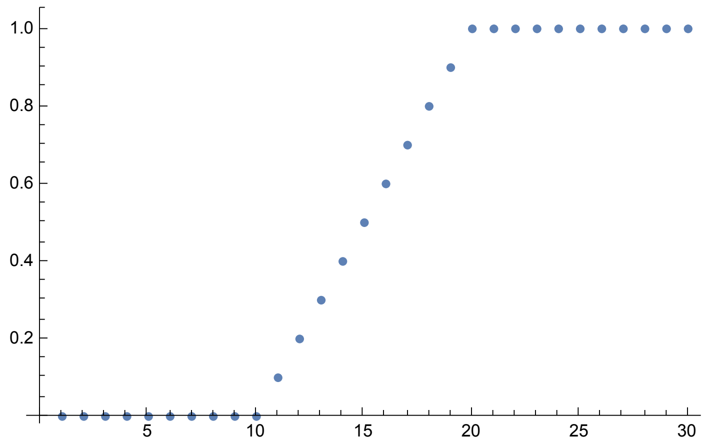
In[294]:=
```
size = 30;
spaceconstant =5;
maxstrength = 0.05;
iterations = 10;
ϵ = .3;
```

Now make the stimulus

In[299]:=
```
e    = Join[Table[0,{i,N[size/3]}],Table[i/N[size/3],
        {i,N[size/3]}], Table[1,{i,N[size/3]}]];
g0   = ListPlot[e, PlotRange -> {{0,30},{-0.5,1.1}},PlotStyle->{RGBColor[1,0,0]}];
picture = Table[e,{i,1,30}];
```

In[302]:=
```
ListPlot[e]
```

Out[302]=



We've stored the graphic **g0** of the input for later use, we can show it later with **Show[g0]**.

Initializing the state vector and specifying the weights

Now we'll initialize the starting values of the output **f** to be random real numbers between 0 and 1, drawn from a uniform distribution.

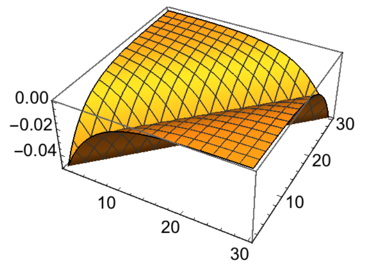
In[303]:=
```
f = RandomReal[{0, 1}, size];
```

Now let's set up synaptic weights which are negative, but become weaker the further they get from the neuron. We assume that the weights drop off exponentially away from each neuron:

In[304]:=
```
W =
Table[N[-maxstrength Exp[-Abs[i-j]/spaceconstant],1],
        {i,size},{j,size}];

ListPlot3D[W,ImageSize->Small]
```

Out[306]=



Note how the weight structure assumes “self-inhibition” corresponding to the figures of the one and two-neuron models above.

Simulating the response

We are going to use the *Mathematica* function **Nest[ ]** to iterate through the limulus equations. Nest[*f*, *expr*, *n*] gives an expression with *f* applied *n* times to *expr*. For example, if we have defined a function **T[ ]**, **Nest[T,x,4]** produces as output:

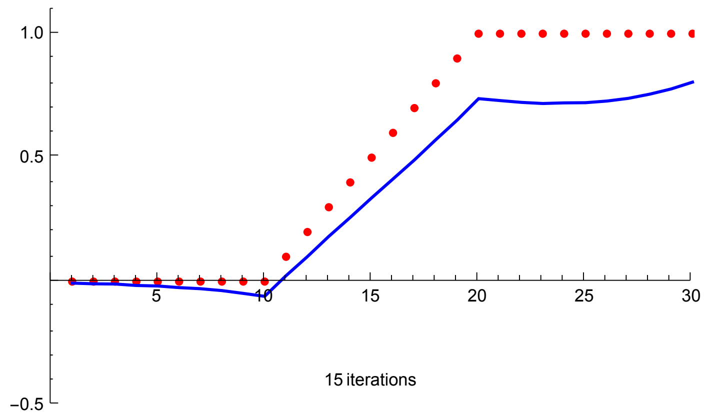
T[T[T[T[x]]]].

Let's express our discrete approximation for the limulus dynamical system in terms of a function, **T**, which will get applied repeatedly to itself with Nest:

In[307]:=
```
T[f_] := f + ϵ (e + W.f - f);
```

In[308]:=
```
iterations = 15;
g1 = ListPlot[Nest[T, f, iterations],PlotJoined->True,
        PlotRange -> {{0,30},{-.5,1.0}},PlotStyle->{RGBColor[0,0,1]}];
Show[g0,g1, Graphics[Text[iterations "iterations",
        {size/2,-0.4}]]]
```

Out[310]=



How does the simulation match up to data from the Limulus eye?

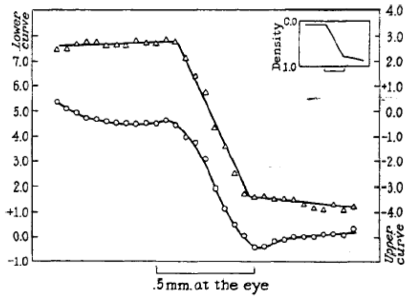


Fig. 8. Contrast phenomena, analogous to Mach bands, demonstrated by patterns of optic nerve fiber activity in the eye *of Limulus.* The discharge of impulses from a receptor was recorded as the eye was caused to scan slowly a pattern of illumination containing a simple gradient of intensity shown in the inset, upper right. When all the receptors were masked except the one from which activity was being recorded, a faithful representation of the actual physical distribution of light was obtained (upper graph, triangles). With the mask removed, so that all the receptors viewed the pattern, the lower graph (circles) was obtained, with a maximum and a minimum where Mach bands are seen by a human observer viewing the same pattern. (From Ratliff and Hartline[21])

From Hartline's Nobel lecture http://www.nobel.se/medicine/laureates/1967/hartline-lecture.pdf. Figure from: F. Ratliff and H.K. Hartline, *J. Gen. Physiol.*, 42 (1959) 1241.

▶ 5. Google limulus

Explore the parameter space

The effect of ϵ, strength of inhibition, and number of iterations

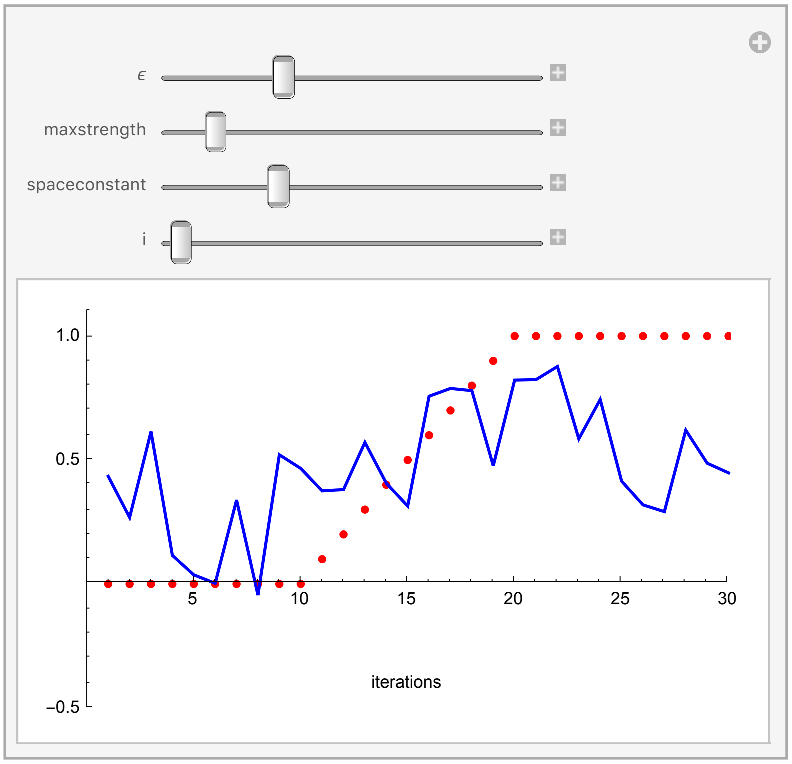
Define a function with inputs: ϵ, maxstrength and iterations, and outputs: a plot of response

We can use the **Module[ ]** function to define a routine with local variables and a set of other functions to define limulus[ϵ_,maxstrength_,iterations_]:

```
In[311]:= limulus[ϵ_, maxstrength_, spaceconstant_, iterations_] :=
  Module[{f, W}, W = Table[N[-maxstrength ⅇ^(-Abs[i-j]/spaceconstant), 1], {i, size}, {j, size}];
   f = RandomReal[{0, 1}, size];
   T[f_] := f + ϵ (e + W.f - f);
   g1 = ListPlot[Nest[T, f, iterations], Joined → True,
     PlotRange → {{0, 30}, {-.5, 1.}}, PlotStyle → {RGBColor[0, 0, 1]}];
   Show[g0, g1, Graphics[Text[iterations "iterations", {size/2, -0.4}]]]]
```

In[312]:=
```
Manipulate[limulus[ϵ, maxstrength, spaceconstant, i], {{ϵ, .3}, 0, 1},
  {{maxstrength, .05}, 0, .5}, {{spaceconstant, 5}, 1, 15, 1}, {i, 1, 15, 1}]
```

Out[312]=



▶ 6. What does the steady state response look like if the inhibition is small (i.e. small maxstrength)?

▶ 7. What does the steady state response look like if the inhibition is large?

▶ 8. What does the steady state response look like if the spaceconstant is very small or very large?

▶ 9. Modify the simulation to investigate what happens if the iteration step-size, ϵ, is large (e.g. 1.5). Run it limulus[ ] several times--i.e. try different initial conditions.

Neural networks as dynamical systems

We've explored a simple linear neural network that is a good model of limulus processing, and seems to provide a possible explanation for human perception of Mach bands. Real neural networks typically have non-linearities. There is no general theory of non-linear systems of difference or differential equations. But the exploration of this linear set does lead us to ask questions which are quite general about dynamical systems:

What does the trajectory in state-space look like?
Does it go to a stable point?
How many stable points or "attractors" are there?

There are non-linear systems which show more interesting behavior in which one sees:

Stable orbits

Chaotic trajectories in state-space
"Strange" attractors

We will return to some of the above questions later when we introduce Hopfield networks.

Recurrent lateral inhibition & Winner-take-all (WTA)

Sometimes one would like to have a network that takes in a range of inputs, but as output would like the neuron with biggest value to remain high, while all others are suppressed. (In computational vision, see “non-maximum suppression”, which is sometimes used in edge detection.) In other words, we want the network to make a decision. The limulus equations can be set up to act as such a "winner-take-all" network. We will remove self-inhibition by setting all the diagonal elements of **W** to zero. We will also add a non-linear thresholding function ("rectification") to set negative values to zero, and we will increase the spatial extent of the inhibition.

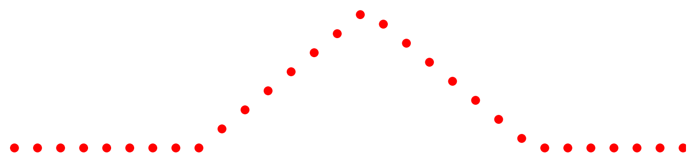
Make a rectifying threshold function

In[313]:=
```
thresh[x_] := N[If[x < 0.0, 0.0, x]];
SetAttributes[thresh, Listable];
```

Make a "tepee" stimulus and initialize the neural starting values

In[315]:=
```
size = 30;
e2  = Join[{0,0},Table[0,{i,N[size/4]}],
        Table[i/N[size/4],{i,N[size/4]}],
        Table[(N[size/4]-i)/N[size/4],{i,N[size/4]}],
        Table[0,{i,N[size/4]}]];
g2  = ListPlot[e2, PlotRange -> {{0,size},{-1,2.0}},PlotStyle→{RGBColor[1,0,0]},Axes→False
```

Out[317]=



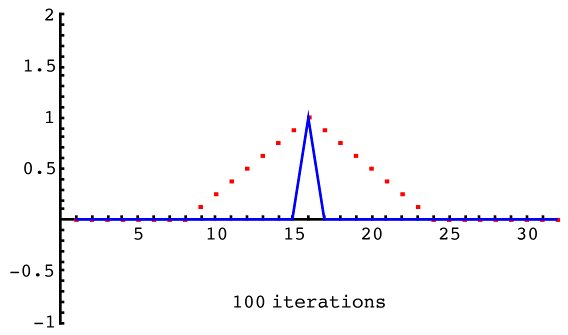
Define winnertakeall[ ] as for limulus[ ], but with no self-inhibition:

```
In[318]:= winnertakeall[ϵ_, maxstrength_, iterations_, spaceconstant_] :=
  Module[{f, W}, W = Table[N[-maxstrength ⅇ^(-Abs[i-j]/spaceconstant), 1], {i, size}, {j, size}];
   For[i = 1, i ≤ size, i++, W⟦i, i⟧ = 0.];
   f = RandomReal[{0, 1}, size];
   T[f_] := thresh[f + ϵ (e2 + W.f - f)];
   g1 = ListPlot[Nest[T, f, iterations], Joined → True,
     PlotRange → {{0, size}, {-1, 2.`}}, PlotStyle → {RGBColor[0, 0, 1]}];
   Show[g2, g1, Graphics[Text[iterations "iterations", {size/2, -0.8`}]]]]
```

▶ 10. Use ListPlot3D[W] to see the modified structure of the weight matrix

```
Wwta = W;
For[i = 1, i ≤ size, i++, Wwta⟦i, i⟧ = 0.];
ListPlot3D[Wwta, ImageSize → Small]
```

▶ 11. Run simulation: Find a set of parameters that will select the maximum response and suppress the rest

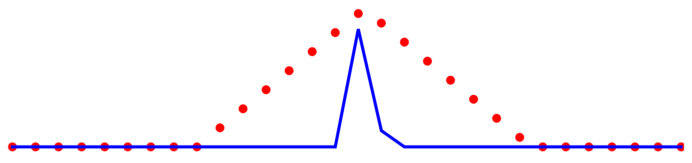


If we think of the number of iterations to steady-state as "reaction time", how is this neural network for making decisions? How sensitive is its function to the choice of parameters?

If you are having a hard time finding a good set of parameters, select the cell below, then go to **Cell->Cell Properties->Cell Open**, and then run it.

```
In[322]:= winnertakeall[.25, .95, 100, size]
```

Out[322]=



100 iterations

Next time

Review matrices. Representations of neural network weights.

References

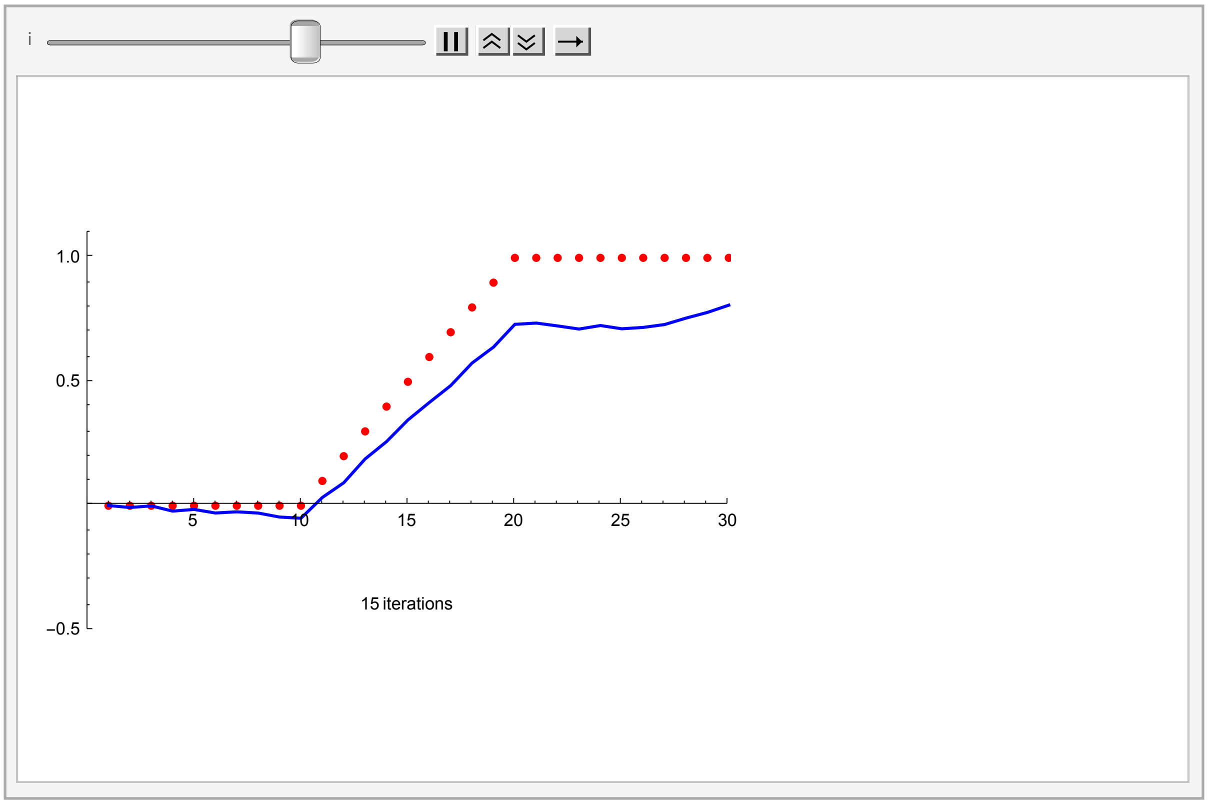
Anderson, J. A. (1995). An Introduction to Neural Networks . Cambridge, MA: MIT Press. (Chapter 4.)
Boyaci, H., Fang, F., Murray, S. O., & Kersten, D. (2007). Responses to lightness variations in early human visual cortex. Curr Biol, 17(11), 989-993.
Hartline, H. K., & Knight, B. W., Jr. (1974). The processing of visual information in a simple retina. Ann N Y Acad Sci, 231(1), 12-8.
http://www.mbl.edu/animals/Limulus/vision/index.html
Hartline, HK, Wagner, HG, & Ratliff , F. (1956) Inhibition in the Eye of Limulus, Journal of General Physiology, 39:5 pp.651-673
Komatsu, H. (2006). The neural mechanisms of perceptual filling-in. Nature Reviews Neuroscience, 7(3), 220–231. doi:10.1038/nrn1869
Knill, D. C., & Kersten, D. (1991). Apparent surface curvature affects lightness perception. Nature, 351, 228-230. (pdf) .http://gandalf.psych.umn.edu/~kersten/kersten-lab/demos/lightness.html
Luenberger, D.G. (1979). Introduction to dynamic systems : theory, models, and applications. (pp. xiv, 446). New York: Wiley.
Morrone, M. C., & Burr, D. (1988). Feature detection in human vision: A phase-dependent energy model. Proceedings of the Royal Society of London. Series B: Biological Sciences, 221–245.
Ratliff, F., Knight, B. W., Jr., Dodge, F. A., Jr., & Hartline, H. K. (1974). Fourier analysis of dynamics of excitation and inhibition in the eye of Limulus: amplitude, phase and distance. Vision Res, 14(11), 1155-68.
Wallis, S. A., & Georgeson, M. A. (2012). Mach bands and multiscale models of spatial vision: The role of first, second, and third derivative operators in encoding bars and edges. Journal of Vision, 12(13), 18–18. doi:10.1167/12.13.18

Appendix

Use NestList[ ] to store all the iterations before showing the results.

In[323]:=
```
T2[f_] := f + ϵ (e + W.f - f);
temp = NestList[T2, f, 15];
Animate[
 Show[g0, ListPlot[temp[[i]], PlotJoined → True, PlotRange → {{0, 30}, {-.5, 1.0}},
   PlotStyle → {RGBColor[0, 0, 1]}], Graphics[Text[iterations "iterations",
        {size / 2, -0.4}]]], {i, 1, 15, 1}]
```

Out[325]=

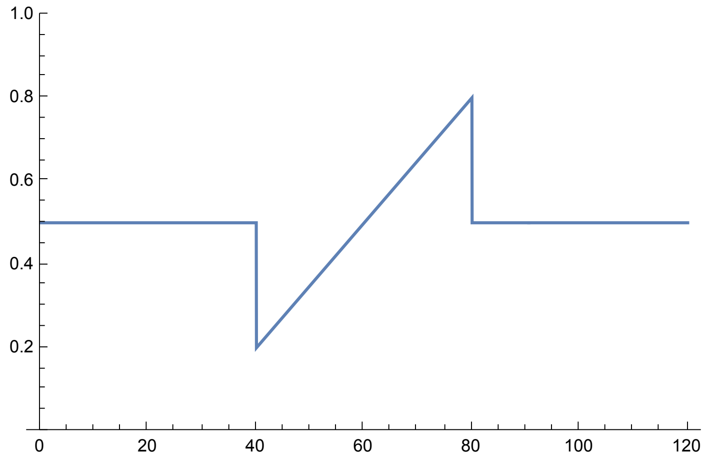


▶ 12. Exercise: Make a gray-level image of the horizontal luminance pattern shown below.

Does the left uniform gray appear to be the same lightness as the right patch? Can you explain what you see in terms of lateral inhibition?

In[326]:=
```
low = 0.2; hi = 0.8;
left = 0.5; right = 0.5;
y2[x_] := left /; x<40
y2[x_] :=
    ((hi-low)/40) x + (low-(hi-low)) /; x>=40 && x<80
y2[x_] := right /; x>=80
```

In[331]:=
```
Plot[y2[x], {x, 0, 120}, PlotRange → {0, 1}]
```

Out[331]=


▶ 13. Exercise: Hermann grid

Below is the Hermann Grid. Notice the phantom dark spots where the white lines cross. Can you explain what you see in terms of lateral inhibition?

In[332]:=
```
width2 = 5; gap = 1; nsquares = 6;
```

In[333]:=
```
hermann = Flatten[Table[{Rectangle[{x, y}, {x + width2, y + width2}]},
{x, 0, (width2 + gap) * (nsquares - 1), width2 + gap},
     {y, 0, (width2 + gap) * (nsquares - 1), width2 + gap}], 1];
```

In[334]:=
```
Show[Graphics[hermann, AspectRatio → 1]]
```

Out[334]=