Introduction to Neural Networks
U. Minn. Psy 5038
Spring, 1998

Lateral inhibition

Introduction

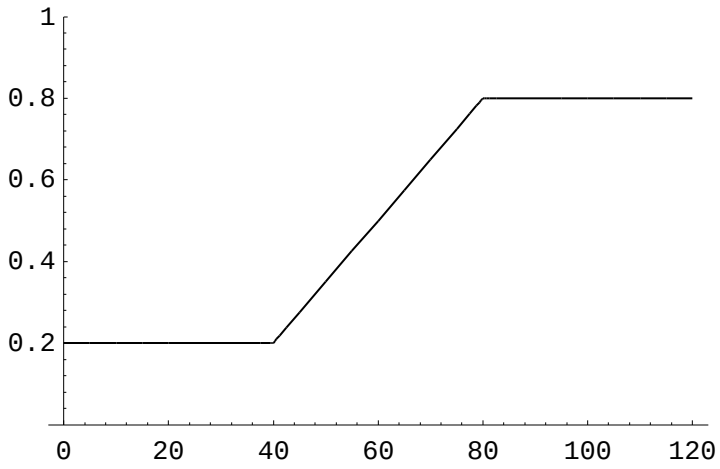
We are going to look at an explanation of a perceptual phenomenon called Mach bands, that involves a good linear approximation based on a real neural network. This is an example of neural filtering found in early visual coding. We will study two types of network that may account for Mach bands: 1) feedforward; 2) feedback. The feedback system will provide our first example of a dynamical system.

Mach bands & perception

Ernst Mach was an Austrian physicist and philospher. In addition to being well-known today for a unit of speed, he is also known for several visual illusions. One illusion is called "Mach bands". Let's make some.

```
Clear[y];
low = 0.2; hi = 0.8;
y[x_] := low /; x<40
y[x_] :=
    ((hi-low)/40) x + (low-(hi-low)) /; x>=40 && x<80
y[x_] := hi /; x>=80
```

```
Plot[y[x],{x,0,120},PlotRange->{0,1}];
```



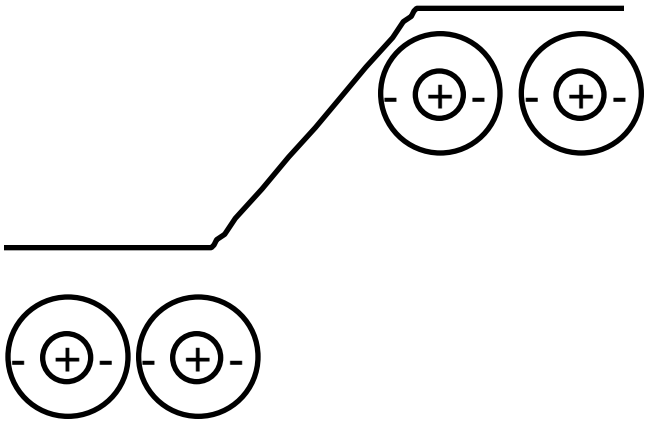

```
size = 120;
e := Table[y[i],{i,1,size}];
```

Let's make a 2D gray-level picture displayed with **ListDensityPlot** to experience the Mach bands for ourselves. **PlotRange** allows us to scale the brightness.

```
e1 = e;
picture = Table[e1,{i,1,60}];
ListDensityPlot[picture,Frame->False,Mesh->False,
                PlotRange->{0,1}];
```

What Mach noticed was that the left knee of the ramp looked too dark, and the right knee looked too bright. Objective light intensity did not predict apparent brightness.

■ **Mach's explanation**



Neural basis?

Limulus (horseshoe crab)--Hartline

Frog - Barlow

Cat --Kuffler

Feedforward model

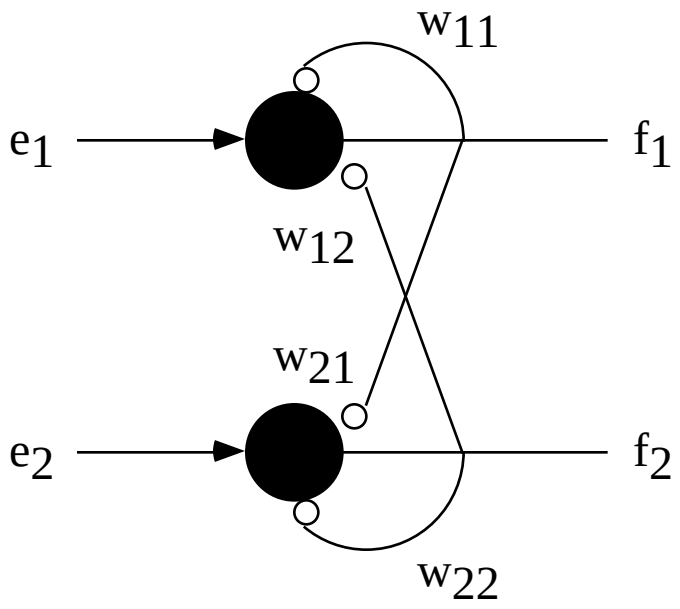
Two types of models: feedforward

y = w.e

where **e** is a vector representing the input intensities, **w** is a suitably chosen set of weights (i.e. excitatory center and inhibitory surround as shown on the above figure), and **y** is the output.

There is also neurophysiological evidence for an implementation of lateral inhibition via feedback or *recurrent lateral inhibition*.

Mach bands & physiology: Recurrent lateral inhibition



Steady state

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

$$f_1 = e_1 + \left(w_{12} f_2 + w_{11} f_1\right)$$
$$f_2 = e_2 + \left(w_{21} f_1 + w_{22} f_2\right)$$

or in summation notation:

$$f_i = e_i + \sum_j w_{ij} f_j$$

or in matrix notation:

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

or in vector and *Mathematica* notation:

f = e + W.f

Note that by expressing f in terms of e, this is equivalent to another linear matrix equation,the feedforward solution:

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

where

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

The **-1** exponent means the inverse of the matrix in brackets. **I** is the identity matrix.

We will see later how to find the inverse of a matrix.

■ Dynamical system -- coupled differential equations

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:

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

"Steady state" just means that the values of **f(t)** are not changing, i.e. **df/dt = 0**. In terms of the components, the equations can also be written:

$$\frac{df_i(t)}{dt} = e_i(t) + \sum_{j=1}^{n} w_{ij} f_j(t) - f_i(t)$$

We are going to develop a solution to this set of equations using a discrete time approximation.

The state vector at time t+Δt (ε = Δt) can be approximated as:

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

We are going to fix or "clamp" the input, start with arbitrary position of the state vector, and seek a stable 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. In particular,

If ε is big (near 1), our coupled equations are unstable

If w's are small, f ~e

If w's are big (lots of inhibition), e gets swamped out.

This simple system of equations leads 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

Simulation 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 epsilon:

The input stimulus

```
size = 30;
spaceconstant =5;
maxstrength = 0.05;
iterations = 10;
epsilon = .3;
```

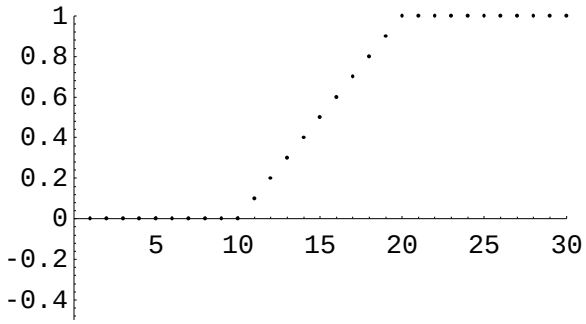
```
e   = Join[Table[0,{i,N[size/3]}],Table[i/N[size/3],
        {i,N[size/3]}], Table[1,{i,N[size/3]}]];
g0  = ListPlot[e, PlotRange -> {{0,30},{-0.5,1.0}},
        DisplayFunction -> Identity];
picture = Table[e,{i,1,30}];
```

We've stored the graphic g0 of the input for later use. The option

DisplayFunction -> Identity prevents the display. We can turn it on later with:

DisplayFunction -> $DisplayFunction.

```
Show[g0, DisplayFunction -> $DisplayFunction];
```



The weights

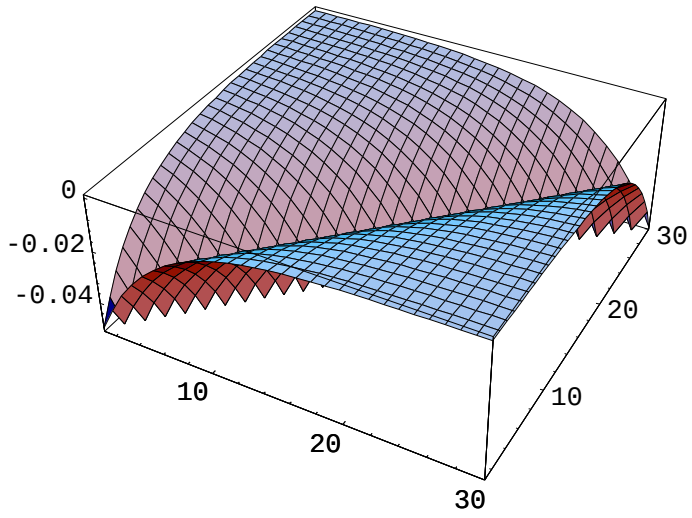
Now we'll initialize the starting values of the output **f** to be random real numbers between 0 and 1, drawn from a uniform distribution.

```
f   = Table[Random[],{size}];
```

Now let's set up synaptic weights which drop off exponentially away from each neuron:

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

```
ListPlot3D[W];
```

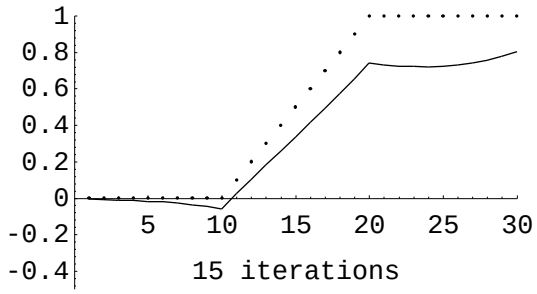


Simulating the response

We are going to use the *Mathematica* function **Nest[]** to iterate through the limulus equations. In order to do this we have to express the dynamical system in terms of a function, **T**, which gets applied repeatedly to itself. For example, **Nest[T,x,4]** produces as output **T[T[T[T[x]]]]**.

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

```
iterations = 15;
g1 = ListPlot[Nest[T, f, iterations],PlotJoined->True,
        PlotRange -> {{0,30},{0,1.0}},
        DisplayFunction -> Identity];
Show[g0,g1, Graphics[Text[iterations "iterations",
        {size/2,-0.4}]],
        DisplayFunction -> $DisplayFunction];
```



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 we would like the neuron with biggest value to remain high, while all others are suppressed. The limulus equation can be set up to act as such a "winner-take-all" network. The inhibition strength needs to be large, the space constant big, and self-inhibition should be removed.

```
size = 32;
spaceconstant =2;
maxstrength = 2.0;
iterations = 19;
epsilon = .25;
```

■ **Make a "tepee" stimulus and intialize the neural starting values**

```
e   = Join[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]}]];
g0  = ListPlot[e, PlotRange -> {{0,32},{-0.5,2.0}},
        DisplayFunction -> Identity];
f   = Table[N[0],{size}];
```

■ **Set up weights**

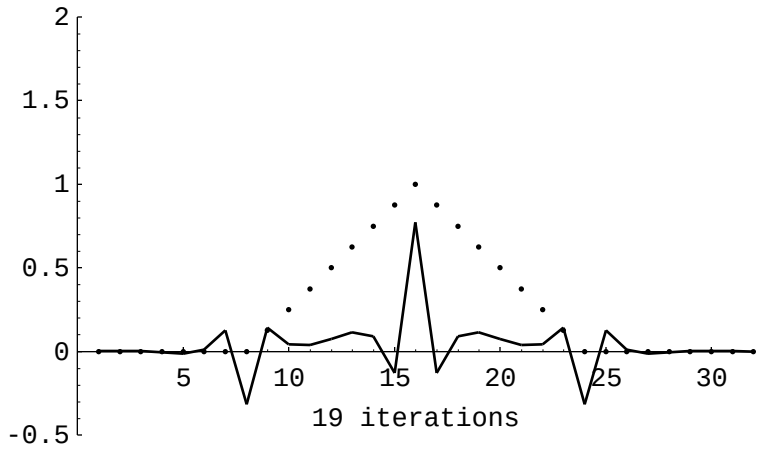
```
e2 = Table[-maxstrength,{i,size}];
w0 = DiagonalMatrix[e2];
W = Table[N[-maxstrength Exp[-Abs[i-j]/spaceconstant],1],
        {i,size},{j,size}];
W = W - w0 ;
```

■ **Define network output values, f**

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

■ **Run simulation**

```
g1 = ListPlot[Nest[T, f, iterations],
        PlotJoined->True,
        PlotRange -> {{0,32},{-0.5,2.0}},
        DisplayFunction -> Identity];
Show[g0,g1, Graphics[Text[iterations "iterations",
        {size/2,-0.4}]],
        DisplayFunction -> $DisplayFunction];
```



The network is evolving in the right direction, but the suppression of the low-rate neurons isn't complete. Can you modify the network parameters to do a better job?

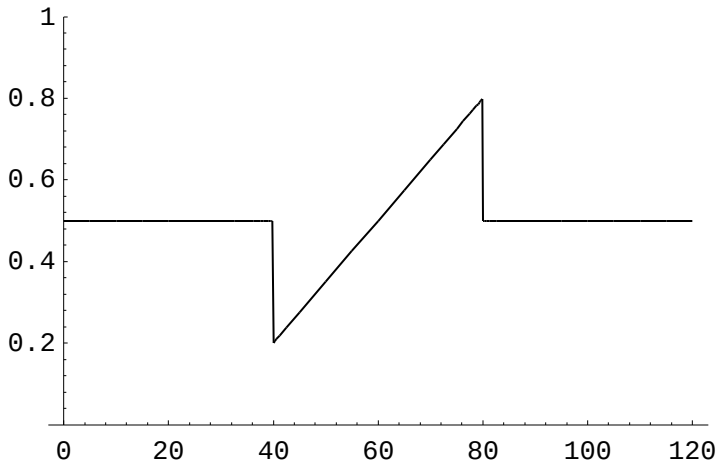
Optional Exercises

■ **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?

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

```
Plot[y[x],{x,0,120},PlotRange->{0,1}];
```



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?

```
width = 5; gap = 1; nsquares = 6;
```

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

```
Show[Graphics[hermann, AspectRatio → 1]];
```