Introduction to Neural Networks

Linear systems analysis

Introduction to Learning and Memory

Introduction

Last time

Matrix algebra review. Finished up with "outer product", "eigenvectors" of a matrix. Useful for neural networks and natural computation. Most important points required for today's lecture:

1. **Symmetric matrix**

If the transpose of a matrix equals itself, $W^T = W$, W is said to be a symmetric matrix. I.e. $w_{ij} = w_{ji}$ for each i and j.

Symmetric matrices occur quite often in physical systems (e.g. the force on particle i by particle j is equal to the force of j on i). The elements of a symmetric matrix H look like they are reflected about the diagonal.

We constructed a symmetric matrix when we modeled the exponential drop-off in lateral inhibition.

2. **Eigenvalues/eigenvectors**. An eigenvector **x** of a matrix **W** points in the same direction as **x**:

$$\mathbf{W.x} = \lambda \mathbf{x}$$

The scale factor λ is the eigenvector. Eigenvectors of a symmetric matrix are orthogonal.

3. **Outer product of two matrices**

For example, the outer product between two 3-dimensional vectors **f** and **g** is:

$\mathbf{f g^T} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} (g_1\ g_2\ g_3)$. In *Mathematica* it is written Outer[Times,**f**,**g**]

```
In[1]:= Clear[f1, f2, f3, g1, g2, g3]
        Outer[Times, {f1, f2, f3}, {g1, g2, g3}] // TraditionalForm
Out[2]//TraditionalForm=
```

$$\begin{pmatrix} f1\,g1 & f1\,g2 & f1\,g3 \\ f2\,g1 & f2\,g2 & f2\,g3 \\ f3\,g1 & f3\,g2 & f3\,g3 \end{pmatrix}$$

Today

Linear systems--as an abstraction of matrix operations in linear neural networks. Useful for the *analysis* of systems, including neural systems.
In this sense, linear systems analysis is a special case of “systems identification”.

Will provide insight into how to think about experimental "black box" tests for whether it makes sense to model a neural or behavioral process as a linear system.

Brief overview of learning and memory
Modeling associative memory

Linear systems, matrices & neural networks

Motivation

Linear systems analysis is an experimental technique that has been applied in numerous areas in the cognitive and neurosciences. Examples include characterizing human ability to detect visual patterns, hear sounds, and changes in skin pressure, spiking responses of single neurons to sensory inputs, and the hemodynamic response in functional magnetic resonance imaging of the human visual cortex. Let’s look at the underlying theory.

Introduction

We've seen that the basic linear network is represented by a matrix of weights that operates on a vector of input activities. One property of such a system is that it satisfies the fundamental abstract definition of a "linear system".

In this lecture we will look at linear systems theory as conceptual tools to *analyze*, rather than model a neural system. A network may not be linear, but it can still be useful to use linear systems methods to measure how it differs from linearity.

The world of input/output systems can be divided up into linear and non-linear systems. Linear systems are nice because the mathematics that describes them is not only well-known, but also has a mature elegance. On the other hand, it is a fair statement to say that most real-world systems are not linear, and thus hard to analyze...but fascinating if for that reason alone. Scientists were lucky with the limulus eye. That nature is usually non-linear doesn't mean one shouldn't familiarize oneself with the basics of linear system theory. Many times a non-linear system has a sufficiently smooth mapping, over some domain, that it can be approximated by a linear function. So precisely what is a "linear system"?

The notion of a "linear system" can be thought of as a *generalization of the input/output properties of a straight line passing through zero*. The matrix equation **W.x** = **y** represents a discrete n-dimensional linear system. There are two basic requirements of a linear system that can be induced from matrix/vector multiplication. Suppose that **W** is a matrix, **x1** and **x2** are vectors, and a and b are scalars. Then we know that:

$$\mathbf{W}.(a\,\mathbf{x}1) = a\mathbf{W}.(\,\mathbf{x}1)$$

This is the *scaling* or *homogeneity* requirement for a linear system.

The second basic property of a linear system is *additivity*:

$$\mathbf{W}.(\mathbf{x}1 + \mathbf{x2}) = \mathbf{W.x1} + \mathbf{W.x2}$$

Together these specify the *superposition principle*:

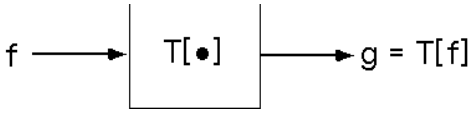
$$\mathbf{W}.(a\ \mathbf{x}1 + b\ \mathbf{x2}) = a\ \mathbf{W.x1} + b\ \mathbf{W.x2}$$

This so far is just a consequence of the laws of matrix algebra.

Generalization

The idea of a linear system can be generalized beyond matrix algebra to include continuous systems. This is useful mathematically, but the generalization also provides insight into basic experimental tests that psychologists and neuroscientists use to test linearity in a system, where they may not care or know about the dimensionality of the system. Linear systems analysis provides a set of tools for the experimental study of a system. Tests of linearity have been applied to behavioral responses, neuron responses (recall the experimental graphs showing linearity and departures from linearity in the slow potentials of a neuron), and neuroimaging responses (e.g. functional magnetic resonance imaging of BOLD signals).

Imagine we have a box that takes inputs such as f, and outputs g = T[f], where f and g could be vectors or continuous functions.



The abstract definition of a linear system is that it satisfies the *superposition principle*:

$$T[a\ f_1 + b\ f_2] = a\ T[f_1] + b\ T[f_2]$$

where T is the transformation that takes the sum of scaled inputs f_1, f_2 (which can be functions or vectors) to the sum of the scaled transformation of f_1 and f_2. The fact that linear systems show superposition is good for doing theory, is often a good approximation, but as we will see later, it limits the kind of computations that can be done with linear systems, and thus with linear neural network models. But we can analyze a network to see how it deviates from linearity.

Let's get a graphical view.

Homogeneity

One way of analyzing a system is to measure its response to a single point in time or space.

For example, suppose we have an input **f1** representing an impulse at time i=33 (i.e. f1 is a vector with 0s everywhere but at i=33, where i=1). (A brief pulse can be thought of as an approximation to a "delta function"). And suppose the measured response is **g1** as shown in the first figure below. This is sometimes called an *impulse response*. For example given an amplifier, a click on a microphone yields a response on the speaker, which isn't exactly the same click as the input--it may be spread out in time and have lost some high frequencies.

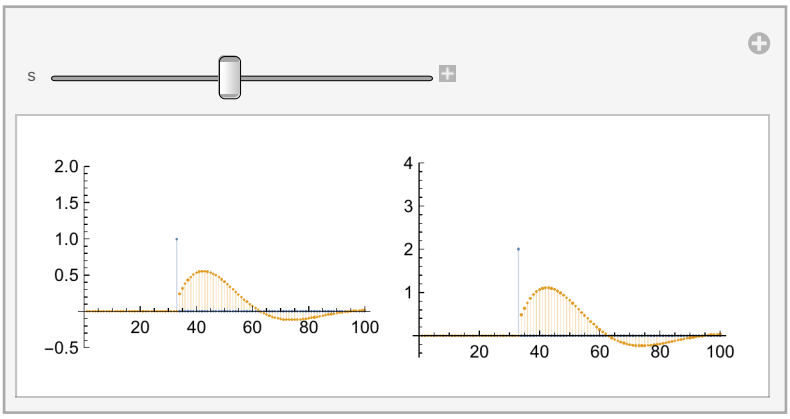
A second example is looking at a point of light such as a star. Due to optical aberrations in your eye, the image of the star on your retina isn't a point, but is spread out.

A third example crops up in the analysis of functional magnetic resonance images (fMRI) of the brain. A short burst of neural activity results in a BOLD (blood oxygen level dependent) response that is delayed and spread out in time. The response (the hemodynamic response function) has a shape similar to that in the graph below. However, it is an experimental question as to whether any of these systems shows homogeneity (cf. Boynton et al., 1996).

If the system is linear, and in particular respects homogeneity, then if we double the input 2 x **f1**, the response should be 2 x **g1**. The blue dot represents the input impulse, and the tan curve the output.

In[3]:=
```
f1 = Table[If[i == 33, 1, 0], {i, 1, 100}];
g1 = Table[If[i ≤ 33, 0, -e^(-1/20 Abs[i-33])] Sin[i/10], {i, 1, 100}];
Manipulate[GraphicsRow[{ListPlot[{f1, g1}, PlotRange → {-0.5, 2}, Filling → Axis],
ListPlot[{s f1, s g1}, PlotRange → {-0.5, 4.0}, Filling → Axis]}], {{s, 2}, .25, 4.0}]
```

Out[5]=

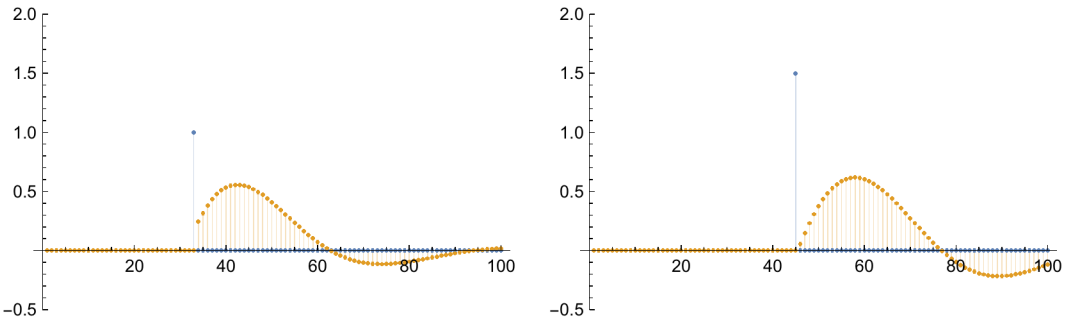


Additivity

Now imagine an input impulse **f1** at time i=33, followed by another **f2** at time i=45. These lead to responses **g1** and **g2**.

In[6]:=
```
f1 = Table[If[i == 33, 1, 0], {i, 1, 100}];
f2 = Table[If[i == 45, 1.5, 0], {i, 1, 100}];
g1 = Table[If[i ≤ 33, 0, -e^(-1/20 Abs[i-33])] Sin[i/10], {i, 1, 100}];
g2 = Table[If[i ≤ 45, 0, -e^(-1/30 Abs[i-45])] Sin[(i-14)/10], {i, 1, 100}];
GraphicsRow[{ListPlot[{f1, g1}, Filling → Axis, PlotRange → {-0.5, 2}],
ListPlot[{f2, g2}, Filling → Axis, Filling → Axis, PlotRange → {-0.5, 2}]}]
```

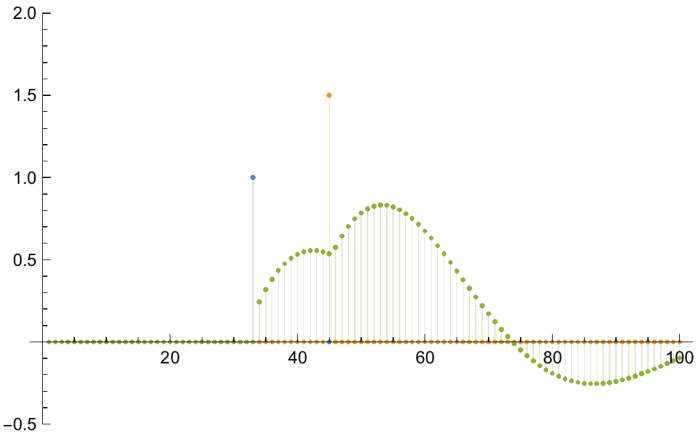
Out[10]=



If the system shows additivity, what would the response look like?

In[11]:=
```
ListPlot[{f1, f2, g1 + g2}, Filling → Axis, PlotRange → {-0.5, 2}]
```

Out[11]=

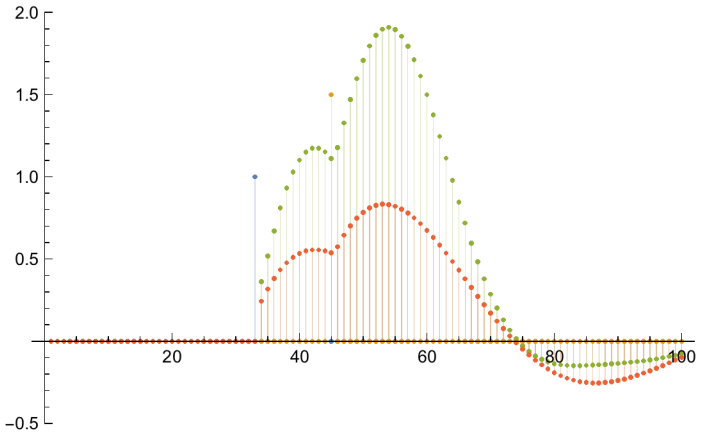


Non-linear, non-additivity example

Here's an example of a response that has a non-linear component (because of the quadratic terms). The output (no longer a simple sum) is shown in green. The original is orange.

In[12]:=
```
ListPlot[{f1, f2, g1 + g2 + 2 (g1 g1) + 2 (g2 g2) + 2 (g1 g2), g1 + g2},
 Filling → Axis, PlotRange → {-0.5`, 2}]
```

Out[12]=



Mathematica note: we used the following syntax to multiply two n-dimensional vectors element by element to produce an n-dimensional output:

In[13]:=
```
{a, b} {x, y}
```

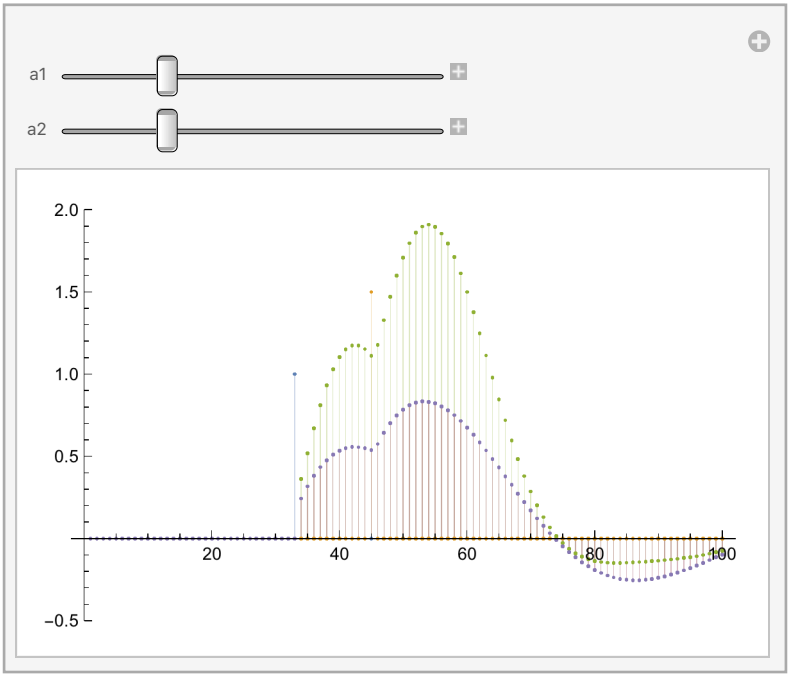
Out[13]=
```
{a x, b y}
```

Compare this with {a,b}.{x,y}

Can you find a linear solution? Manipulate the a1 and a2 sliders to get the orange close to the green.

In[14]:=
```
Manipulate[
 ListPlot[{f1, f2, g1 + g2 + 2 (g1 g1) + 2 (g2 g2) + 2 (g1 g2), (a1 * g1 + a2 * g2), g1 + g2},
  Filling → Axis, PlotRange → {-0.5, 2}], {{a1, 1.0}, 0, 4.}, {{a2, 1.0}, 0, 4.}]
```

Out[14]=



► 1. Find the least squares linear solution

While one can't get a perfect fit above, sometimes a linear approximation is good enough. One can solve the above problem in a well-defined way by calculating a linear fit that minimizes the sum of squared errors. Of course, this solution is for the particular pair of inputs. How general is the fit?

In[15]:=
```
Clear[a1, a2];
nonlinearoutput = N[g1 + g2 + 2 (g1 g1) + 2 (g2 g2) + 2 (g1 g2)];
bestleastsquaresfit = a1 * N[g1] + a2 * N[g2];
sumsquares = Total[(nonlinearoutput - bestleastsquaresfit) ^ 2];
FindMinimum[sumsquares, {a1, a2}]
```

Out[19]=
```
{2.46813, {a1 → 2.08742, a2 → 2.01621}}
```

Or with the built-in function LeastSquares[], you can do it in one line:

In[20]:=
```
N[LeastSquares[Transpose[{g1, g2}], nonlinearoutput]]
```

Out[20]=
```
{2.08742, 2.01621}
```

Characterizing a linear system by its response to an orthonormal basis set

Suppose we have an unknown physical or biological system that we are studying in the lab. We'll keep it simple and assume just 8 inputs (e.g. if it is a visual system, you could think of the system having 8 light sensitive receptors). We might do a a simple test of superposition and see that it behaves linearly, at least for the pair of stimuli we've tried. So tentatively we assume it is a linear system. We really don't know what the full characteristics are, but let's represent it by a (square) matrix **T** filled with numbers that are assumed to be *unknown* to us as experimenters:

In[21]:=
```
T = Table[RandomReal[], {i, 1, 8}, {j, 1, 8}];
```

Think of **T** as a particular, but arbitrary (i.e. not special), experimental system under study. The numbers in the matrix T are hidden, and we would like to make a simple set of measurements that could characterize **T** in such a way that we could predict the output of **T** to any input. This is the sort of task that engineers face when wanting to characterize, say a audio amplifier (as a model linear system), so that the output sound can be predicted for any input sound. Or an optical system, like the lens of a camera.

What kind of measurements would tell us what **T** is? Well, we could make a huge look-up table that enumerates the response to each possible input. That's impractical, and also unnecessary if the system is linear (or is approximately so).

Another thing we could do is "stimulate" the system with cartesian vectors {1,0,0,0,0,0,0,0}, {0,1,0,0,0,0,0,0}, ..., and so forth and collect the responses. The responses would be the columns of **T**. Once we have those, we can predict the response to any input. This is in fact the kind of thing illustrated above with pulses at just two of 100 possible time points, i.e. i = 33 and i = 45.

This is theoretically fine, but can lead to two practical problems:

1) for a real physical system, such as your audio player, or a neuron in the limulus eye, this would require concentrating all of the physical stimulus energy into one small stimulus, i.e. very short duration in time or small in space. A strong input may be needed just to measure a reliable response. This isn't always a problem, but stimulation with a high-intensity audio or light intensity spike could drive the system into a non-linear range, or worse damage what you are trying to study. For example, if you

have 1 million receiving elements, getting a response by stimulating only 1 part in a million may require concentrating lots of energy on just one element to get a measurement above noise levels. In the theoretical limit, this corresponds to stimulating it with a "delta function", a spike which is infinitely narrow and infinitely high.

2) Characterizing the linear system by a matrix **T**, requires n^2numbers, where **n** is the input signal vector length (because each of the n inputs generates an output vector consisting of n numbers)--and **n** can be pretty big for both audio and visual systems. Problem 2) has a more efficient solution when **T** is symmetric, and even nicer solution if the rows are shifted versions of each other, as with the model for lateral inhibition (this is addressed later). Problem 1) can be addressed by showing that we can characterize **T** with any basis set--so we can pick one that more uniformly distributes energy over the inputs in the the physical system being tested.

The set of Walsh functions we looked at earlier is just one of many possible basis sets with the advantage that the elements that contribute to the "energy", i.e. (the square of the vector length) are distributed across the vector. Here is a

In[22]:=
```
nwalshset = HadamardMatrix[8];
{w1, w2, w3, w4, w5, w6, w7, w8} = N@nwalshset;
```

If you look at the values of the elements in the **w**'s, you'll see that their magnitudes are spread out over the 8 inputs, in contrast to the cartesian set.

We have already seen that the vector basis set $\{w_i\}$ spans 8-space in such a way that we can express any vector, **g**, as a linear sum of these basis vectors.

$$g = \sum (g.w_i)\, w_i$$

So as we saw in an earlier lecture, an arbitrary vector, **g**

In[24]:=
```
g = {2,6,1,7,11,4,13,29};
```

is the sum of its own projections onto the basis set:

In[25]:=
```
(g.w1) w1  + (g.w2) w2  +(g.w3) w3  +(g.w4) w4  +
(g.w5) w5  +(g.w6) w6  +(g.w7) w7  +(g.w8) w8
```

Out[25]=
```
{2., 6., 1., 7., 11., 4., 13., 29.}
```

Suppose we now do an "experiment" to find out how **T** transforms the vectors of our basis set, and we put all of these transformed basis elements into a new set of vectors **newW[[i]]**, where **newW** is a matrix for which each row is the response of **T** to a basis vector.

In other words, we will treat the basis set as our key set of test stimuli.

In[26]:=
```
newW = {T.w1,T.w2,T.w3,T.w4,T.w5,T.w6,T.w7,T.w8};
```

Note that **newW** is an 8x8 matrix.

So how can we calculate the output of **T**, given an arbitrary **g** without actually "going back to the lab" and running **g** through our experimental system **T**?

By the principle of linearity (or the properties of matrix algebra), we can calculate the output by finding the "spectrum" of **g,** (i.e. $\{g.w_i\}$ as in Problem Set 1), and then scaling each of the transformed basis elements by the spectrum and adding them up:

$$T.g = T.\left\{\sum (g.w_i)\, w_i\right\} = \sum (g.w_i)\, T.w_i$$

In[27]:=
```
(g.w1) T.w1 + (g.w2) T.w2 + (g.w3) T.w3 + (g.w4) T.w4 +
(g.w5) T.w5 + (g.w6) T.w6 + (g.w7) T.w7 + (g.w8) T.w8
```

Out[27]= {22.7215, 37.0806, 27.1971, 32.349, 47.9475, 48.5157, 9.14456, 32.0964}

Now here's the interesting point. Of course, we have already done our "experiment" to characterize **T** in terms of the responses to the basis vectors. They are $\{\mathbf{T}.w_i\}$ -- the responses of **T** to the basis vectors $\{w_i\}$--and we've stored them as rows of the matrix **newW** = $\{\mathbf{T}.w_1, \mathbf{T}.w_2, ..., \mathbf{T}.w_8\}$. Because we have our key test set of vectors, {w1,w2,w3,w4,w5,w6,w7,w8}, know we can calculate the spectrum {**g.w**i} of **g**.

So the output of **T** can be calculated as:

In[28]:=
```
(g.w1) newW[[1]] + (g.w2) newW[[2]] + (g.w3) newW[[3]] +
(g.w4) newW[[4]] + (g.w5) newW[[5]] + (g.w6) newW[[6]] +
(g.w7) newW[[7]] + (g.w8) newW[[8]]
```

Out[28]= {22.7215, 37.0806, 27.1971, 32.349, 47.9475, 48.5157, 9.14456, 32.0964}

So we don't have to "go back to the lab" to test the response to **g**...at least not if we believe the system is linear.

Note: If we do "go back to the lab" and run the input through **T** we can check to see if this is right:

In[29]:=
```
T.g
```

Out[29]= {22.7215, 37.0806, 27.1971, 32.349, 47.9475, 48.5157, 9.14456, 32.0964}

► 2. Show that one can summarize reconstruction of the input vector g using more concise matrix notation, where W = {w1,w2,w3,w4,w5,w6,w7,w8}.

Let the basis vectors be the rows of a matrix **W**:

In[30]:=
```
W = {w1, w2, w3, w4, w5, w6, w7, w8};
```

So again, we can project **g** onto the rows of **W**, and then reconstitute it in terms of **W** to get **g** back again:

In[31]:=
```
(W.g).W
```

Out[31]= {2., 6., 1., 7., 11., 4., 13., 29.}

But as before, but more concisely, by calculating the spectrum, and multiplying it by our stored representation of what the system does to the basis vectors -- **newW**, we can predict the output:

In[32]:=
```
g.Transpose[W].newW
```

Out[32]= {22.7215, 37.0806, 27.1971, 32.349, 47.9475, 48.5157, 9.14456, 32.0964}

► 3. Show that **T==Transpose[newW].W,** and that the system output is thus: Transpose[newW].W.g

► 4. Are the rows of newW above orthogonal?

Summary: *An elegant way to characterize an unknown linear system T is by its response to orthonormal input vectors.*

We next look at an even more elegant solution. *Namely, if **T** is symmetric, then measure the response*

with respect to the eigenvectors of a symmetric matrix (e.g. T itself).

What if the choice of basis set used to "stimulate" T is the set of eigenvectors of T?

In this section we are going to see why sine waves are used so often in linear systems analysis.

These new basis vectors (rows of newW: $\boldsymbol{T}.w_1$, $\boldsymbol{T}.w_2$,...) do span 8-space, but they are not necessarily orthogonal or normal. Under what conditions would they be orthogonal? There is a theorem that says the eigenvectors of a symmetric matrix are orthogonal. Let's assume that the matrix **T** is symmetric. Rather than picking some arbitrary orthogonal basis set to analyze T and to represent inputs, we choose the eigenvectors of **T** as the basis set.

We've just seen how linearity provides us with a method for characterizing a linear system in terms of the responses of the system to a set of orthogonal basis vectors. One problem is that if the input signals are long vectors, say with dimension 40,000, then this set of basis vector responses is really big--1.6×10^9 numbers.

Suppose though that we have a symmetric matrix transformation, **T**. And let $\{\boldsymbol{e_i}\}$ be the eigenvectors of **T**. You should be able to show that if vectors of the basis set are the eigenvectors of **T**, $\{\mathbf{e}_i\}$, then the transformation of any arbitrary input vector **x** is given by:

$$\boldsymbol{T.x} = \boldsymbol{T.}\sum(\boldsymbol{x.e_i})\,\boldsymbol{e_i} = \sum(\boldsymbol{x.e_i})\,\boldsymbol{T.e_i} = \sum(\boldsymbol{x.e_i})\,\lambda_i\,\boldsymbol{e_i} = \sum\alpha_i\,\lambda_i\,\boldsymbol{e_i}$$

Where the α_i are the projections of **x** onto each eigenvector (α_i=**x.e**$_i$). Having the eigenvectors of **T** enables us to express the input *and* outputs of **T** in terms of the *same basis set--the eigenvectors*. We can see that all **T** does to the input is to scale its projection onto each eigenvector by the eigenvalue for that eigenvector.

The set of these eigenvalues is sometimes called the *modulation transfer function* (especially in image processing) because it describes how the amplitude of the eigenvectors change as they pass through **T**.

Linear systems analysis is the foundation for *Fourier analysis*, and is why it makes sense to characterize your audio player in terms of frequency response--i.e. how much it attenuates the amplitude of a sinewave input. But your audio player isn't just any linear system--it has the special property that if you input a sound at time t and measure the response, and then you input the same sound again at a later time, you get the same response, except of course that is is shifted in time. It is said to be a *shift-invariant* system. The eigenvectors of a shift-invariant linear system are sinusoids. Further, a linear shift-invariant system can modeled as a *convolution*, as we saw in the lateral inhibition network.

(The eigenvectors of the symmetric matrix in the optional exercises of Lecture 6 were approximately sinusoids, not just because the matrix was symmetric, but also because each row of the matrix was a shifted version of the previous row--the elements along any given diagonal are identical. This is called a symmetric circulant--special case of a Toeplitz--matrix.)

So sine wave inputs are the eigenvectors of your audio player. The dimensionality, of course, is much higher--if you are interested in frequencies up to 20,000 Hz, your eigenvector for this highest frequency would have least 40,000 elements--not just 8! But the math is the same.

As mentioned in the introduction, this kind of analysis has been applied not only to physical systems, but to a wide range of neural sensory systems. For the visual system, linear systems analysis has been used to study the cat retina (Enroth-Cugell and Robson, 1964), the monkey visual cortex, and human contrast sensitivity of the system as a whole (Campbell and Robson, 1968).

Much empirical analysis has been done using linear systems theory to characterize neural sensory systems in audition and touch, and other neural systems such as those for eye movements. It works wonderfully as long as the linear system approximation holds. And it does quite well for the lateral eye of the limulus, as well as X-cells and P-cells of the mammalian visual system, and over restricted ranges for so-called "simple" cells in the visual cortex, among others. The optics of the simple eye is another example of an approximately linear system--one can predict the image contrast distortions on your fovea using linear systems theory.

In summary:

If **T** has n distinct orthogonal eigenvectors, $\mathbf{e}_i$, and known eigenvalues, λ_i, then to calculate the response to an arbitrary input **x**, do the following:

Step 1: Project **x** onto eigenvectors of **T** to get the spectrum of **x**: $\mathbf{x}.\mathbf{e}_i$
Step 2: Multiply each $\mathbf{x}.\mathbf{e}_i$ by the eigenvalue of $\mathbf{e}_i$: $\lambda_i(\mathbf{x}.\mathbf{e}_i)$
Step 3: Scale each vector $\mathbf{e}_i$ by the scalar value $\lambda_i\mathbf{x}.\mathbf{e}_i$ to get $(\lambda_i \mathbf{x}.\mathbf{e}_i)\mathbf{e}_i$
Step 4: Sum these vectors up. That's the response of T to **x**! :

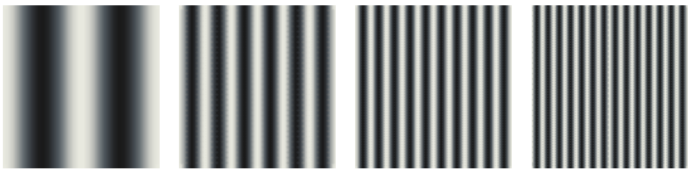
$$\sum_i (\lambda_i \mathbf{x}.\mathbf{e}_i)\,\mathbf{e}_i$$

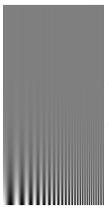
So the eigenvectors are presumed fixed (i.e. sinusoidal input vectors are just part of our standard toolbox we can apply as inputs to any system). We test our unknown, n x n system **T** to measure the eigenvalues, which in general will be system specific. To predict the response to **x,** we get another tool out of our toolbox (e.g. "spectrum analyzer") that gives us the spectrum of **x**, another set of n numbers. The product of the eigenvalues and the spectral values tell us the lengths of each eigenvector for the output.

An illustration from human vision: the contrast sensitivity function

In[33]:=
```
Grating[x_, y_, fx_, fy_] := Cos[2 Pi (fx x + fy y)];
g = Table[DensityPlot[0.025 Grating[x, y, fx, 0],
    {x, -1, 1}, {y, -1, 1}, PlotPoints → 32, Mesh → False, Frame → False,
    PlotRange → {-1, 1}, ColorFunction → "GrayTones"], {fx, 1, 7, 2}];
Show[GraphicsRow[{g[[1]], g[[2]], g[[3]], g[[4]]}]]
```

Out[35]=



In[36]:= Image[];

Neural networks have been proposed to explain the shape of the human contrast sensitivity (Campbell and Robson, 1968). One type of model is called a “single-channel” model, but it is basically the same lateral inhibition model that you have already seen. More complicated models assume that there are different populations of neurons (different “channels”), each with its distinctive weights. The weights are presumed to reflect processing by neurons in the primary visual cortex. They have a center-surround organization, but are no longer circularly symmetric. Instead they are said to have preferred orientations. The CSF above may be the envelope representing the collection of most sensitive cells to these frequencies across many channels.

Learning and memory: Brief overview

It is curious to note that historically, the simple linear model that we will discuss came after much research had been devoted to non-linear learning models, based in particular on a special case of McCulloch-Pitts neurons, used in a system called the Perceptron. Linear models have severe limitations, in particular as a consequence of the superposition principle discussed above. Nevertheless, many of the interesting properties of non-linear systems can be understood in terms of small signal linearity properties. So with that preamble, let's take a look at memory and see where linearity applies, and how far we can get with a linear model. But first some definitions, and a general overview.

Definition of learning and memory

Memory is the consistent, adaptive change in the behavior of an organism as a consequence of past experience. More precisely one can distinguish learning and memory:

Learning has to do with acquisition (encoding), and *memory* with storage and retrieval of information. Memory recall is the retrieval of information.

Psychology and biology of learning and memory

Associative and non-associative memory

In the next few lectures, we are going to be talking about *associative memory*, so it is useful to bear in mind that not all memory is considered associative.

Associative memory: What events reliably and predictably occur together? Here an animal may learn about the relationship of one stimulus to another. Or of one part of a multidimensional stimulus to another part. This is related to “content-addressable” memory, in which one part of a stored relationship between different parts of a stored input can be used to retrieve the rest of the input. An animal may also learn to establish associations between a stimulus and its own motor responses, or the association of an input stimulus with a reward.

Nonassociative memory: exposure to a single stimulus either once, or repeated offers opportunity for

learning about it

Examples of nonassociative learning

Habituation (Ivan Pavlov)

decrease in behavioral reflex response to repeated non-noxious stimulus

Sensitization (pseudo-conditioning)

exposure to noxious stimuli increases sensitivity

More complex examples of nonassociative learning

memory for sensory record...although what one calls a stimulus vs. response is problematic

imitation learning

Associative learning

Classical conditioning (Ivan Pavlov)

Smell or sight of food naturally produces salivation.

Food is said to be an unconditioned stimulus (US) for salivation

Now pair a tone with food:

(tone) -- food -- salivation

(conditioned stimulus--unconditioned stimulus--response)

CS -- US -- R

After repeated pairings, the tone can produce salivation by itself, i.e. in the absence of food.

The tone is said to now be a conditioned stimulus (CS).

Another example:

tone -- air puff - eye blink

Temporal contiguity of CS-US, and frequency important, but not the only factors.

We say that an association between the tone and salivation has been learned.

Operant conditioning (Thorndike of Columbia U.)

also called instrumental or trial-and-error learning

E.g. hungry rat in a cage with a lever

Occasionally the rat may press the lever out of curiosity, accident or whatever, but if it does and promptly receives some food (a "consequence", which in this case is a reward or a positive reinforcement), the lever pressing (called the "operant") will increase above a spontaneous rate (the rate in the control state where there is no reward).

In general, consequences can involve positive or negative reinforcement (take away an ongoing unpleasant stimulus), or positive or negative punishment (take away a pleasant stimulus).

The rat learns an association between its action and food delivery.

The idea of associative learning is very general. We will need to make finer-grained distinctions, because the mechanisms and functions of associative learning can be quite different depending on the kind of adaptive behavior under consideration. Later, for example, we will look at associations that might be established between neighboring pixels in an image, and how these can be used.

Human memory

Stages of memory

Classical view from cognitive psychology -- "block diagram" interpretation of stages of memory

Iconic -- e.g. visual afterimages (1 sec)

Working memory (short-term memory, minutes to hours)

small capacity, disrupted by being knocked unconscious

short-term neural plastic events

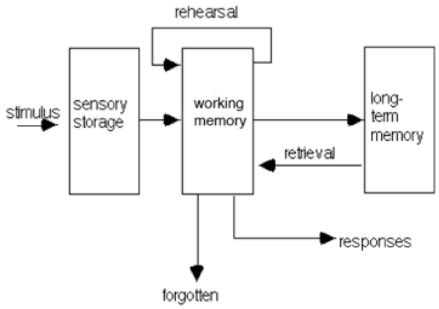
possible mechanisms?

rapid synaptic modification?

reverberating circuits?

Long-term memory

large capacity, relatively permanent (years)



Implicit (reflexive) vs. explicit (declarative) memory

reflexive -- automatic readout not dependent on awareness, or cognitive processes of comparison, evaluation

declarative -- "I saw a yellow canary yesterday" relies on inference, comparison, and evaluation

Learning to driving a car or play tennis -- memory moves from declarative towards reflexive with experience.

In our consideration of models, we will be primarily concerned with simple reflexive associative memory in which an neural input event leads to the reconstruction of a predictive response based on past experience.

Theoretical/machine learning perspective

Types of learning

http://en.wikipedia.org/wiki/Machine_learning

Unsupervised

Discover regularities in an ensemble of inputs

Example: correlations between image pixels or sound samples, efficient coding, PCA in statistics, autoassociation in neural networks.

At this point in time, the wiki link is a weak on this one, but take a look: http://en.wikipedia.org/wiki/Unsupervised_learning

From a neural perspective, in unsupervised learning synaptic weight dynamics depends only on activity of pre and post-synaptic neurons

Supervised Learning

Learning with a teacher, where the desired output is known for each input.

"Try to do this....ok, now this is the right answer, see how close you were?"

weights are adjusted to minimize the difference between desired & actual output

- Example: regression in statistics, multi-layer networks with error backpropogation learning rule, general “heteroassociation” in neural networks.

See: http://en.wikipedia.org/wiki/Supervised_learning

In terms of neurons, learning depends on pre- and post-synaptic activity and an error signal

Reinforcement Learning: sensation, action, and goal

"Reinforcement learning is learning what to do--how to map situations to actions--so as to maximize a numerical reward signal."-- Sutton, R. S., & Barto, A. G. (1998).

From a neural perspective, learning depends on pre- and post-synaptic activity, and a *reward signal*

Learning with a "critic". e.g. "You did really well" or "You really messed up on that one"

While “supervised” in the sense of a critic, in contrast to standard supervised learning, correct pairs of inputs and outputs are not provided. More representative of how biological learning typically occurs, and has distinct challenges and characteristics:

"Trial and error search", exploration/exploitation trade-off, delayed reward.

See: http://en.wikipedia.org/wiki/Reinforcement_learning

Linear model of associative memory

Basic assumptions: Knowledge is represented in a neural network in the connections of the neurons. Knowledge is acquired/learned through experience that changes in the weights of the connections.

Hebbian rule for synaptic modification

Introduction: Modeling associative learning and memory

The brain has developed mechanisms that allow the animal to distinguish events that reliably and predictably occur together from those that do not. What kinds of neural models could be used to capture and retrieve associations? How can one pattern of neural activity come to be associated with another?

Specific assumptions:

• physical basis of memory is in synaptic modification which alters the mapping of inputs to outputs

• the strength of the modification is determined by how often input and output activity occurs together, and by the strengths of the input and output activities.

The idea of learning as association goes back to William James (1890) (See Anderson).

"When two elementary brain processes have been active together or in immediate succession,

one of them on recurring, tends to propagate its excitement into the other (Psychology: Briefer Course)."-- William James (1890)

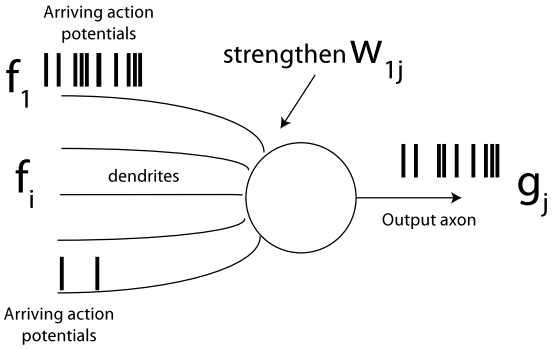
(James also has an "activation equation" that sounds a lot like our stage 1 of the 2-layer feedforward network:

"The amount of activity at any given point in the brain-cortex is the sum of the tendencies of all other points to discharge into it" -- William James (1890). Replace "point" by "neuron", and we have our limulus equation.)

There are similar statements elsewhere (e.g. Kenneth Craik of Cambridge in the 1940's).

But the clearest and most explicit statement of an associative learning rule is credited to Canadian Psychologist Donald Hebb:

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells, such that A's efficiency as one of the cells firing B, is increased" (Hebb in the "Organization of Behavior", 1949).



For many years there was little evidence to support this notion of synaptic modification. But direct tests of Hebbian conjecture have now been made. (See Sjöström et al., 2008 for review; Paulsen & Sejnowski, 2000))

Hebbian synaptic modification: Modeling using the outer product

The fundamental Hebbian assumption is that synaptic strength grows with co-activity of pre- and post-synaptic activity. How can we quantify this? We will use a simple model of synaptic modification that assumes that a change in connection strength between neuron i and j is proportional to the product of the input and output activities.

$$\Delta W_{ij} = \alpha\, f_i g_j$$

If you remember the definition of the outer product of two vectors, you can see that the above rule is just that--an outer product of the input and output activities.

The linear model

Heteroassociation (supervised) vs. autoassociation (unsupervised)

We will look at two types of associative memory models. In heteroassociation, an input **f** is associated with **g**. So at some later time, when the system is stimulated with **f**, it should produce **g**. In autoassocia-

tion, an input **f** is associated with itself.

On the face of it, autoassociation sounds a bit silly--"what do you learn"? But in the next lecture, we will see the use of autoassociation.

Summary of linear association model

1. **Learning.** Let $\{\mathbf{f}_n, \mathbf{g}_n\}$ be a set of input/output activity pairs, where n = 1 to N. Memories are stored by superimposing new weight changes on old ones. Information from many associations is present in *each* connection strength.

$$\mathbf{W}_{n+1} = \mathbf{W}_n + \alpha\, g_n\, \mathbf{f}_n^T = \mathbf{W}_n + g_n\, \mathbf{f}_n^T$$

(For simplicity, we'll set the learning rate coefficient $\alpha = 1$.)

2. **Recall.** Let **f** be an input possibly associated with output pattern **g**. A linear model for recall is:

$$\mathbf{g} = \mathbf{W}_N \cdot \mathbf{f}$$

$$g_i = \sum_j w_{ij}\, f_j$$

3. **Condition for perfect recall**

If the input (i.e. stimulus) vectors $\{\mathbf{f}_n\}$ are orthonormal, the system shows perfect recall:

$$\begin{aligned}\mathbf{W}_n\, \mathbf{f}_m &= \left(\mathbf{g}_1\, \mathbf{f}_1^T + \mathbf{g}_2\, \mathbf{f}_2^T + \ldots + \mathbf{g}_n\, \mathbf{f}_n^T\right)\, \mathbf{f}_m \\ &= \mathbf{g}_1\, \mathbf{f}_1^T\, \mathbf{f}_m + \mathbf{g}_2\, \mathbf{f}_2^T\, f_m + \ldots + \mathbf{g}_m\, \mathbf{f}_m^T\, \mathbf{f}_m + \ldots + \mathbf{g}_n\, \mathbf{f}_n^T\, \mathbf{f}_m \\ &= \mathbf{g}_m\end{aligned}$$

since,

$$\mathbf{f}_n^T\, \mathbf{f}_m = \begin{cases} 1, & n = m \\ 0, & n \neq m \end{cases}$$

So an **nxn** matrix has a memory capacity of **n** for orthogonal inputs. For random vectors, it is about 10-20% of **n**.

Linear association is also useful as a mapping, i.e. as *linear regression*, in which one wants to generalize in a smooth (actually linear) way to novel inputs. For example, linear regression in 2D can be done with a 2x2 matrix. It can be "trained" with a few input/output pairs of points {x,y}. Once trained, if the data are well-modeled by a straight line, the network will do a nice job of "predicting" the output value (y) given a novel input value (x).

Next time

Simulations of associative memory

Heteroassociative memory

Autoassociative memory

Appendix

Eigenvectors, eigenvalues: algebraic manipulation with Mathematica

Last time we used the *Mathematica* function **Eigenvectors[]** and **Eigenvalues[]** to produce the eigenvectors and eigenvalues for the matrix equation: **Ax = λx**. It is worth spending a little time to understand what is being done algebraically. Consider the following pair of equations specified by a 2x2 matrix **W** acting on **xv** to produce a scaled version of **xv**:

In[37]:=
```
W1 = {{1,2},{2,1}};
xv := {x,y};
lambda xv == W1.xv
```

Out[39]= $\{\text{lambda}\, x, \text{lambda}\, y\} == \{x + 2\, y, 2\, x + y\}$

Finding the eigenvalues is a problem in solving this set of equations. If we eliminate x and y from the pair of equations, we end up with a quadratic equation in lambda:

Eliminate[] & Solve[]

In[40]:=
```
Eliminate[{lambda xv == W1.xv, lambda != 0}, {x, y}]
```

Out[40]= $(-3 + \text{lambda} \neq 0 \&\& \text{lambda} \neq 0 \&\& 1 + \text{lambda} \neq 0) \,||\, -2\, \text{lambda} + \text{lambda}^2 == 3$

In[41]:=
```
Solve[-2 lambda + lambda^2 == 3, lambda]
```

Out[41]= $\{\{\text{lambda} \to -1\}, \{\text{lambda} \to 3\}\}$

So our eigenvalues are -1 and 3. We can plug these values of lambda into our equations to solve for the eigenvectors:

In[42]:=
```
Solve[{-x == x + 2 y, - y == 2 x + y}, {x,y}]
Solve[{3 x == x + 2 y, 3 y == 2 x + y}, {x,y}]
```

Solve: Equations may not give solutions for all "solve" variables.

Out[42]= $\{\{y \to -x\}\}$

Solve: Equations may not give solutions for all "solve" variables.

Out[43]= $\{\{y \to x\}\}$

Reduce

Mathematica is smart enough that we can use **Reduce[]** to do it all in one line:

In[44]:=
```
Reduce[{lambda xv == W1.xv}, {x,y,lambda}]
```

Out[44]= $(x == 0 \&\& y == 0) \,||\, \left((y == -x \,||\, y == x) \&\& x \neq 0 \&\& \text{lambda} == \frac{x + 2\, y}{x}\right)$

The eigenvectors are unique only up to a scale factor, so one can choose how to normalize them. For example, we could arbitrarily set x to 1, and then the eigenvectors are: {1,1}, and {1,-1}. Alternatively, we could normalize them to {1/Sqrt[2], 1/Sqrt[2]} and {1/Sqrt[2], -1/Sqrt[2]}.

In[45]:=
```
Eigenvectors[W1]
```
Out[45]=
```
{{1, 1}, {-1, 1}}
```

Side note: Solve[] vs. Reduce[]

Solve[] makes assumptions about constraints left unspecified, so the following returns the solution true for any lambda:

In[46]:=
```
Solve[{lambda x == x + 2 y, lambda y == 2 x + y},
    {x,y,lambda}]
```
Solve: Equations may not give solutions for all "solve" variables.

Out[46]=
```
{{y → -x, lambda → -1}, {y → x, lambda → 3}, {x → 0, y → 0}}
```

In[47]:=
```
Solve[{lambda x == x + 2 y, lambda y == 2 x + y,
lambda !=0, x != 0, y != 0}, {x,y,lambda}]
```
Solve: Equations may not give solutions for all "solve" variables.

Out[47]=
```
{{y → -x, lambda → -1}, {y → x, lambda → 3}}
```

Reduce[] gives all the possibilities without making specific assumptions about the parameters:

Either of the following forms will work too:

In[48]:=
```
Reduce[lambda xv == W1.xv, {xv[[1]],xv[[2]],lambda}]
Reduce[{{lambda x, lambda y} == {x + 2 y, 2 x + y}},
    {x,y,lambda}]
```

Out[48]=
$$(x == 0 \&\& y == 0) \,||\, \left((y == -x \,||\, y == x) \&\& x \neq 0 \&\& \text{lambda} == \frac{x + 2\,y}{x}\right)$$

Out[49]=
$$(x == 0 \&\& y == 0) \,||\, \left((y == -x \,||\, y == x) \&\& x \neq 0 \&\& \text{lambda} == \frac{x + 2\,y}{x}\right)$$

Determinant solution

Ax = λx can be written: **(A - Iλ).x, w**here **I** is the identity matrix. The interesting values of **x** that satisfy this equation are the ones that aren't zero. For this to be true, **(A - Iλ)** must be singular (i.e. no inverse). And this is true if the determinant of **(A - Iλ)** is zero

Answers

In[50]:=
```
G = RandomReal[{0, 1}, {8, 8}];
```

In[51]:=
```
W[[1]]
```
Out[51]=
```
{0.353553, 0.353553, 0.353553, 0.353553, 0.353553, 0.353553, 0.353553, 0.353553}
```

In[52]:=
```
G[[1]]
```
Out[52]=
```
{0.253055, 0.697285, 0.379601, 0.148807, 0.313116, 0.540876, 0.253647, 0.203795}
```

In[53]:=
```
Wmemory = Table[0, {8}, {8}];
For[i = 1, i ≤ 8, i++,
  Wmemory = Wmemory + Outer[Times, G[[i]], W[[i]]]];
```

In[55]:=
```
Wmemory.W[[3]]
G[[3]]
```

Out[55]= {0.987643, 0.371271, 0.66457, 0.706767, 0.265348, 0.333793, 0.710247, 0.696239}

Out[56]= {0.987643, 0.371271, 0.66457, 0.706767, 0.265348, 0.333793, 0.710247, 0.696239}

Linear memory model exercises

- 5. Exercise: Verify that linear memory model shows perfect recall for orthonormal inputs
- 6. Define an 8x8 matrix G whose rows are 8 dimensional random output vectors
- 7. Print out row 3: G[[3]]
- 8. Define an 8x8 matrix Wmemory whose elements is the sum of the outer products of the rows of G with the rows of W (the orthonormal Walsh set defined earlier). This corresponds to Step 1 in the previous section
- 9. Show that G[[3]]=Wmemory.W[[3]]
- 10. Discussion question: What if the input vectors are not orthonormal?
- 11. Exercise: Pencil and paper

Let f1 and f2 be two orthogonal, normalized input patterns:

$$f_1 = \frac{1}{\sqrt{3}}\begin{pmatrix}1\\-1\\1\end{pmatrix}$$

$$f_2 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\\0\end{pmatrix}$$

and g1, g2 two output patterns:

$$g_1 = \begin{pmatrix}3\\1\\2\end{pmatrix}$$

$$g_2 = \begin{pmatrix}-1\\-1\\2\end{pmatrix}$$

Form the outer product:

$$W = g_1 f_1^T$$

Test for "recall" by feeding f1 as input to W. Stimulate W with f2. What happens? Add the outer product:

$$g_2 f_2^T$$

to the previous W matrix. Now test for recall on stimulation with f1, and f2. What do you find?

References

Boynton, G. M., Engel, S. A., Glover, G. H., & Heeger, D. J. (1996). Linear systems analysis of functional magnetic resonance imaging in human V1. J Neurosci, 16(13), 4207-4221.

Campbell, F. W., & Robson, J. R. (1968). Application of Fourier Analysis to the Visibility of Gratings. Journal of Physiology , 197, 551-566.

Carandini, M., Heeger, D. J., & Movshon, J. A. (1997). Linearity and normalization in simple cells of the macaque primary visual cortex. J Neurosci, 17(21), 8621-44.

Enroth-Cugell, C., & Robson, J. G. (1966). The contrast sensitivity of retinal ganglion cells of the cat, Journal of Physiology, London, 187, 517-552.

Gaskill, J. D. (1978). Linear Systems, Fourier Transforms, and Optics . New York: John Wiley & Sons.

Mackintosh, N. J. (1994). Animal learning and cognition. San Diego: Academic Press.

Nykamp, D. Q., & Ringach, D. L. (2002). Full identification of a linear-nonlinear system via cross-correlation analysis. J Vis, 2(1), 1-11.

Paulsen, O., & Sejnowski, T. J. (2000). Natural patterns of activity and long-term synaptic plasticity. Current Opinion in Neurobiology, 10(2), 172–179.

Silverman, M.S., Grosof, D.H., DeValois, R.L., & Elfar, S.D. (1989). Spatial-frequency organization in primate striate cortex. Proceedings of the National Academy of Science USA, 86, 711-715.

Sjöström, P. J., Rancz, E. A., Roth, A., & Häusser, M. (2008). Dendritic excitability and synaptic plasticity. Physiological reviews, 88(2), 769–840. doi:10.1152/physrev.00016.2007

Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning : an introduction. Cambridge, Mass.: MIT Press.

http://onlinebooks.library.upenn.edu/webbin/book/lookupid?key=olbp29573