

Introduction to Neural Networks

Daniel Kersten

Introduction

Last time

Modeled aspects of spatial lateral inhibition in the visual system:

Simple linear network: intensity \rightarrow spatial filter \rightarrow response

intensity and response were represented as vectors

the spatial filter was represented as a matrix

If the spatial filter is a neural network, then the elements of the matrix can be interpreted as strengths of synaptic connections. The idea is to represent the synaptic weights for the i^{th} output neuron by the values in the i^{th} row of the matrix. The matrix of weights is sometimes called a connection matrix just because non-zero weights are between connected neurons, and one can assign zero weights between neurons that are not connected.

This model is what we called a linear feedforward network.

We generalized the linear discrete model to a linear continuous one with feedback. Although more complex, its steady-state solution is identical to the linear feedforward model.

We studied the continuous system by approximating it as a discrete time system with $\epsilon = \Delta t$ as a free parameter that represented how finely we sliced time in the simulations.

Today

There is a large body of mathematical results on linear algebra and matrices, and it is worth our while to spend some time going over some of the basics of matrix manipulation.

We will first review matrix arithmetic (addition and multiplication). We will review the analog of division, namely finding the inverse of a matrix--something that was used in Lecture 5 to show how the steady-state solution of the feedback model of lateral inhibition was equivalent to a feedforward model with the appropriate weights.

You may wonder at times how all this is used in neural modeling. As this course goes on, we will see how otherwise obscure notions of things like an "outer product" between two vectors, or the "eigenvectors" of a matrix are meaningful for neural networks. For example, the "outer product" between two vectors can be used in modeling learning, and the eigenvectors corresponding to a "matrix of memories" can represent stored prototypes, and be used in making drastic reductions in the number of features needed to represent a collection of patterns.

Basic matrix arithmetic

A short motivation, review and preview

In Lecture 4, we developed the generic neural model. If we leave out the non-linear squashing function and noise, we have a linear neural network.

$$y_i = \sum w_{i,j} x_j$$

In Lecture 5, we noted that the feedforward model of lateral inhibition does a good job of characterizing the neural response of visual receptors in the limulus. We implemented the feedforward linear model of lateral inhibition by taking the dot product of a weight vector with the input vector at one location, then we shifted the weight vector over, took a new dot product, and so forth.

This operation can be expressed as a matrix, \mathbf{W} , times a vector, \mathbf{e} , using the linear model of a neural network:

$$y_i = \sum W_{i,j} e_j$$

$\mathbf{W.e}$ is short-hand for: $\sum W_{i,j} e_j$

To illustrate, define \mathbf{e} as we did in the previous lecture, but over a shorter width (so the output display isn't too big) :

```
In[168]:= width = 24; low = 0.2; hi = .8; lowx = 12; hix = 16;
e = Table[Piecewise[{{low, x < lowx}, {(hi - low) / (hix - lowx)) x - ((hi - low) lowx) /
(hix - lowx) + low, x >= lowx && x < hix}, {hi, x >= hix}], {x, 1, width}];
In[169]:= e // MatrixForm;
```

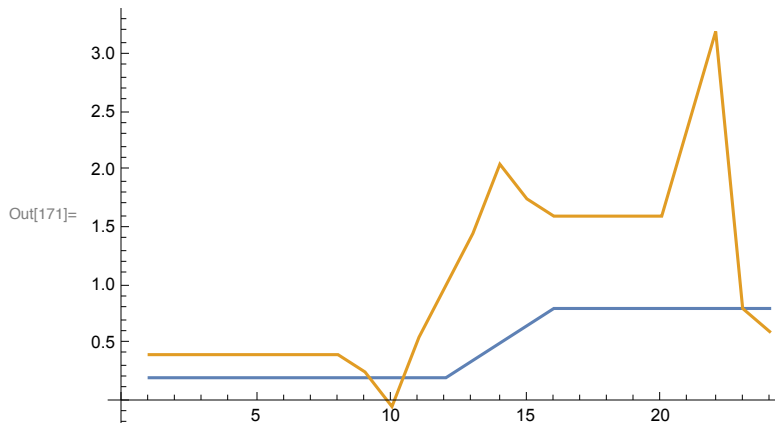
Take the same weights we used in the previous lecture and make a 24x24 matrix in which the first row is: -1,-1,6,1,1,0,0,..... , and the rows above are versions of the above row shifted one place to the right.

Use the **Toeplitz[]** function as a “one-liner” to fill up the matrix with the appropriate weights.

```
In[170]:= W = RotateLeft[ToeplitzMatrix[
{6, -1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, 2];
```

Look up the definitions of a **Toeplitz** and a **Circulant** matrix.

```
In[171]:= ListPlot[{e, W.e}, PlotJoined -> True]
```



Note the boundary effect on the right. How could we get rid of it?

Definition of a matrix: a list of scalar lists

Traditional and Standard form output format defaults.

Defining arrays or lists of function outputs using indices

As we've already seen, `Table[]` can be used to generate a matrix, or list of lists. We can control the format. Unlike `TraditionalForm[]`, `StandardForm[]` gives a unique representation of an expression which can be used as input.

`MatrixForm[]` can also be used. Generally, keep things in the default `StandardForm`, but use the others as an alternative visualization.

```
In[172]:= H = Table[i^2 + j^2, {i, 1, 3}, {j, 1, 3}]
H // TraditionalForm
H // StandardForm
H // MatrixForm
```

```
Out[172]= {{2, 5, 10}, {5, 8, 13}, {10, 13, 18}}
```

```
Out[173]//TraditionalForm=
```

$$\begin{pmatrix} 2 & 5 & 10 \\ 5 & 8 & 13 \\ 10 & 13 & 18 \end{pmatrix}$$

```
Out[174]//StandardForm=
```

```
{{2, 5, 10}, {5, 8, 13}, {10, 13, 18}}
```

```
Out[175]//MatrixForm=
```

$$\begin{pmatrix} 2 & 5 & 10 \\ 5 & 8 & 13 \\ 10 & 13 & 18 \end{pmatrix}$$

Defining arrays or lists of symbols

Although most of the time, we'll be working with numerical matrices, *Mathematica* also allows one to specify arrays or lists of variables.

An $m \times n$ matrix has m rows, and n columns. Here is a 3×4 matrix of symbolic elements $w[i,j]$:

```
In[176]:= Array[w, {3, 4}]
```

```
Out[176]:= {{w[1, 1], w[1, 2], w[1, 3], w[1, 4]},
            {w[2, 1], w[2, 2], w[2, 3], w[2, 4]}, {w[3, 1], w[3, 2], w[3, 3], w[3, 4]}}
```

You can produce a similar array with: **Table[w[i,j],{i,1,3},{j,1,4}]**. You can also have *Mathematica* display a subscripted list

```
In[177]:= L = Table[wi,j, {i, 1, 3}, {j, 1, 4}];
          L // TraditionalForm
```

```
Out[178]//TraditionalForm=
```

$$\begin{pmatrix} 3 & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & 3. & \text{hi} \\ w_{3,1} & w_{3,2} & w_{3,3} & \pi \end{pmatrix}$$

Mathematica lists can have elements of with a variety types. But if the entries are not scalar numbers, then it is list, but not a matrix. We can change the elements like this:

```
In[179]:= w2,3 = 3.0;
          w3,4 = Pi;
          w2,4 = "hi";
          w1,1 = 3;
```

```
In[183]:= L // TraditionalForm
```

```
Out[183]//TraditionalForm=
```

$$\begin{pmatrix} 3 & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & 3. & \text{hi} \\ w_{3,1} & w_{3,2} & w_{3,3} & \pi \end{pmatrix}$$

Or we can assign expressions by row and column number:

```
In[184]:= L[[1, 2]] = "row 1, column 2";
          L // TraditionalForm
```

```
Out[185]//TraditionalForm=
```

$$\begin{pmatrix} 3 & \text{row 1, column 2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & 3. & \text{hi} \\ w_{3,1} & w_{3,2} & w_{3,3} & \pi \end{pmatrix}$$

Adding, subtracting and multiplying by a scalar

As with vectors, matrices are added, subtracted, and multiplied by a scalar component by component:

```
In[186]:= A = {{a,b},{c,d}};
          B = {{x,y},{u,v}};
```

Add **A** to **B**:

```
In[188]:= A+B//TraditionalForm
```

```
Out[188]//TraditionalForm=
```

$$\begin{pmatrix} \left\{ a + 1, a + \frac{1}{6}(-3 - \sqrt{33}) \right\} & b + y \\ c + u & d + v \end{pmatrix}$$

Subtract **B** from **A**:

```
In[189]:= A-B//TraditionalForm
```

```
Out[189]/TraditionalForm=
```

$$\begin{pmatrix} \left\{ a-1, a + \frac{1}{6}(3 + \sqrt{33}) \right\} & b-y \\ c-u & d-v \end{pmatrix}$$

Multiply **A** by 3:

```
In[190]:= 3 A//TraditionalForm
```

```
Out[190]/TraditionalForm=
```

$$\begin{pmatrix} 3a & 3b \\ 3c & 3d \end{pmatrix}$$

Multiplying two matrices

We have already seen how to multiply a vector by a matrix: we replace the i^{th} “row” (i.e. i^{th} element) of the output vector by the inner (dot) product of the i^{th} row of the matrix with the input vector.

In order to multiply a matrix **A**, by another matrix **B** to get **C = AB**, we calculate the ij^{th} component of the output matrix by taking the inner product of the i^{th} row of **A** with the j^{th} column of **B**:

```
In[191]:= Print[A // TraditionalForm, B // TraditionalForm, " = ", A.B // TraditionalForm]
```

```
Dot::rect : Nonrectangular tensor encountered. >>
```

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} \left\{ 1, \frac{1}{6}(-3 - \sqrt{33}) \right\} & y \\ u & v \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} \left\{ 1, \frac{1}{6}(-3 - \sqrt{33}) \right\} & y \\ u & v \end{pmatrix}$$

Note that in general, **AB** is not equal to **BA**:

```
In[192]:= A.B//TraditionalForm
```

```
B.A//TraditionalForm
```

```
Dot::rect : Nonrectangular tensor encountered. >>
```

```
Out[192]/TraditionalForm=
```

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} \left\{ 1, \frac{1}{6}(-3 - \sqrt{33}) \right\} & y \\ u & v \end{pmatrix}$$

```
Dot::rect : Nonrectangular tensor encountered. >>
```

```
Out[193]/TraditionalForm=
```

$$\begin{pmatrix} \left\{ 1, \frac{1}{6}(-3 - \sqrt{33}) \right\} & y \\ u & v \end{pmatrix} \cdot \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Laws of commutation, association and distribution

In particular, look at the element in the upper left of the matrix **BA** above--there is no reason, in general, for $bu+ax$ to equal $ax + cy$. That is unlike scalar multiplication, matrix multiplication does not *commute*.

Apart from commutation for matrix multiplication, the usual laws of commutation, association, and distribution that hold for scalars hold for matrices. It is easy to see that matrix addition and subtraction

do commute--e.g. compare $\mathbf{A+B}$ with $\mathbf{B+A}$. Matrix multiplication is associative, so $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$. The distributive law works too: $\mathbf{A}(\mathbf{B+C}) = \mathbf{AB} + \mathbf{AC}$

Non-square matrices

As we noted in the previous lecture, is not necessary for \mathbf{A} and \mathbf{B} to be square matrices (i.e. have the same number of rows as columns) to multiply them. They can be rectangular.

But if \mathbf{A} is an $m \times n$ matrix, then \mathbf{B} has to be an $n \times p$ matrix in order for \mathbf{AB} to make sense. In other words, the left matrix (\mathbf{A}) has to have the same number of columns as the right matrix (\mathbf{B}) has rows. For example, here \mathbf{F} is a 3×2 matrix, and \mathbf{G} is a 2×4 matrix.

```
In[194]:= Clear[e];
          F = {{a,b},{c,d},{e,f}}
          G = {{p,q,r,s},{t,u,v,w}}
```

```
Out[195]= {{a, b}, {c, d}, {e, f}}
```

```
Out[196]= {{p, q, r, s}, {t, u, v, w}}
```

Check the dimensions with

```
In[197]:= Dimensions[F]
          Dimensions[G]
```

```
Out[197]= {3, 2}
```

```
Out[198]= {2, 4}
```

Because \mathbf{F} has 2 columns, and \mathbf{G} has 2 rows, it makes sense to multiply \mathbf{G} by \mathbf{F} :

```
In[199]:= Print["F.G = ", F // TraditionalForm,
               G // TraditionalForm, " =", F.G // TraditionalForm]
```

$$\mathbf{F.G} = \begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} \begin{pmatrix} p & q & r & s \\ t & u & v & w \end{pmatrix} = \begin{pmatrix} ap+bt & aq+bu & ar+bv & as+bw \\ cp+dt & cq+du & cr+dv & cs+dw \\ ep+ft & eq+fu & er+fv & es+fw \end{pmatrix}$$

However, because the number of columns of \mathbf{G} (4) do not match the number of rows of \mathbf{F} (3), $\mathbf{G.F}$ is not well-defined.

What happens when you try this?

```
In[200]:= Print[F, G, " =", G.F];
```

Inverse of a Matrix

Recall that in the previous lecture, we used the identity matrix, and matrix inversion to show that the steady-state solution of the recurrent lateral inhibition network was mathematically equivalent to a linear feedforward network.

"Dividing" a matrix by a matrix: the identity matrix & matrix inverses

The matrix corresponding to 1 or unity is the **identity matrix**. Like 1, the identity matrix is fundamental

enough, that *Mathematica* provides a special function to generate n-dimensional identity matrices. Here is a 2x2 identity matrix:

`IdentityMatrix[2] // MatrixForm`

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

It is easy to show that the identity matrix plays the role for matrix arithmetic that the scalar 1 plays for scalar arithmetic:

`IdentityMatrix[2].A // MatrixForm`

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

which is just **A**.

What could it mean to divide one matrix, say **B**, by another, say **A**?

We can divide numbers, *b* by *a*, by *multiplying* *b* times the inverse of *a*, i.e. $a^{-1} = 1/a$.

So to do the equivalent of dividing **B** by **A**, we need to find a matrix **Q** (written \mathbf{A}^{-1}) such that when **A** is multiplied by **Q**, we get the matrix equivalent of unity, i.e. the identity matrix. Then "**B/A**" can be achieved by calculating the matrix product: **B.Q**.

Let

`A = {{a,b},{c,d}};`

Mathematica provides a built-in function to compute matrix inverses:

`Q = Inverse[A]`

$$\left\{ \left\{ \frac{d}{-bc+ad}, -\frac{b}{-bc+ad} \right\}, \left\{ -\frac{c}{-bc+ad}, \frac{a}{-bc+ad} \right\} \right\}$$

We can test to see whether the product of a **A** and **Q** is the identity matrix, but *Mathematica* won't go through the work of simplifying the algebra in this case, unless we specifically ask it to.

`Q.A`

$$\left\{ \left\{ -\frac{bc}{-bc+ad} + \frac{ad}{-bc+ad}, 0 \right\}, \left\{ 0, -\frac{bc}{-bc+ad} + \frac{ad}{-bc+ad} \right\} \right\}$$

`Simplify[Q.A] // TraditionalForm`

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

Here is a simple numerical example. Verify that R is the inverse of B by seeing whether the product is the Identity matrix.

```

B = {{1, -1}, {3, 2}};
R = Inverse[B]
%/MatrixForm

{{2/5, 1/5}, {-3/5, 1/5}}

( 2/5  1/5
  -3/5  1/5 )

```

Review the logic of the proof in Lecture 5 that the steady-state solution for a discretized recurrent model of lateral inhibition is equivalent to a linear feedforward model:

The steady state solution is: $\mathbf{f} = \mathbf{e} + \mathbf{W}\mathbf{f}$.
 Let \mathbf{I} be the identity matrix:

$$\mathbf{I}\mathbf{f} = \mathbf{e} + \mathbf{W}\mathbf{f},$$

Use the laws of matrix algebra to show:

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

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

And assuming that $(\mathbf{I} - \mathbf{W})$ has an inverse:

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

Singular and badly conditioned matrices

You might have noticed that elements of the inverse matrix could have zero denominators if we happened to have unfortunate values of a, b, c, d . In general, the inverse of a matrix is not guaranteed to exist. Let's look at the conditions for which the inverse is not defined.

What if one row is a scaled version of another row?

```
In[201]:= B1 = {{1.5, 1}, {3, 2.0}}
```

```
Out[201]= {{1.5, 1}, {3, 2.}}
```

Then the rows of the matrix are not linearly independent. In other words, one row can be expressed as a scaled version of the other row.

In general, *a set of row vectors is linearly independent if for any row you can't write it as a linear combination of any of the others.*

If the rows are not linearly independent, the inverse is not defined and the matrix is said to be *singular*.

What happens if you ask for the inverse of B1?

```
In[202]:= Inverse[B1]//TraditionalForm
```

Sometimes the rows are almost, but not quite, linearly dependent (can arise because the elements are represented as approximate floating point approximations to the actual values, and the computer has insufficient numerical precision). *Mathematica* may warn you that the matrix is badly conditioned. *Mathematica* may try to find a solution to the inverse, but you should be suspicious of the solution. In general, one has to be careful of badly conditioned matrices.

Try finding the inverse of the following matrix:

```
In[203]:= B2 = {{-2,-1},{4.00000000000001,2.0}};
Inverse[B2]
```

Were we lucky? Find out:

```
In[205]:= B2.Inverse[B2]
```

Determinant of a matrix

There is a scalar function of a matrix called the determinant. If a matrix has an inverse, then its determinant is non-zero. Does **B2** have an inverse?

```
Det[B2]
```

```
9.76996 × 10-15
```

Yes, but it is suspiciously small. But **B** has a solid non-zero determinant.

```
Det[B]
```

```
5
```

Matrix transpose

Transpose definition

We will use the transpose operation quite a bit in this course. It interchanges the rows and columns of a matrix:

```
In[220]:= Y = Table[yi,j, {i, 1, 3}, {j, 1, 4}];
% // TraditionalForm
```

```
Out[221]//TraditionalForm=
```

$$\begin{pmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} \end{pmatrix}$$

```
In[222]:= Transpose[Y] // TraditionalForm
```

```
Out[222]/TraditionalForm=
```

$$\begin{pmatrix} y_{1,1} & y_{2,1} & y_{3,1} \\ y_{1,2} & y_{2,2} & y_{3,2} \\ y_{1,3} & y_{2,3} & y_{3,3} \\ y_{1,4} & y_{2,4} & y_{3,4} \end{pmatrix}$$

Recall that the idea of a transpose came up in the previous lecture where we contrasted the difference between a synthetic representation of a pattern in terms of a weighted sum of features with an analytic decomposition of the pattern in terms of neuron firing rates for each receptive field.

Getting parts of a matrix

We covered some of this last time. Here is some more practice.

(See the *Mathematica* documentation: tutorial/GettingAndSettingPiecesOfMatrices)

We've seen before how to get the ij th element, in other words the element in the i th row and j th column. We type **W[[i,j]]** or **W[[i]][[j]]**.

We can pull out the i^{th} row of a matrix in *Mathematica* by simply writing **W[[i]]**. For example, the 2nd row of **Y** is:

```
In[223]:= Y[[2]]
```

```
Out[223]= {Y2,1, Y2,2, Y2,3, Y2,4}
```

What about the j^{th} column of a matrix?

W[[All,j]] produces the j th column of matrix **W**. To get the 3rd column of a matrix in *Mathematica*, type:

```
In[224]:= Y[[All, 3]]
```

```
Out[224]= {Y1,3, Y2,3, Y3,3}
```

We can also use the transpose operation to do it. **Transpose[W][[i]]** produces the j th column of matrix **W**. For example, the 3rd column of **Y** is:

```
In[225]:= Transpose[Y][[3]]
```

```
Out[225]= {Y1,3, Y2,3, Y3,3}
```

To pull out a submatrix with rows i_0 through i_1 , and columns j_0 through j_1 , type: **W[[i₀ ;; i₁, j₀ ;; j₁]]**

```
In[226]:= Y[[2 ;; 3, 1 ;; 3]] // TraditionalForm
```

```
Out[226]/TraditionalForm=
```

$$\begin{pmatrix} y_{2,1} & y_{2,2} & y_{2,3} \\ y_{3,1} & y_{3,2} & y_{3,3} \end{pmatrix}$$

And here is yet another way:

```
In[227]:= Y[[Range[2, 3], Range[1, 3]]]
```

```
Out[227]= {{Y2,1, Y2,2, Y2,3}, {Y3,1, Y3,2, Y3,3}}
```

Pull out the element $y_{2,3}$ from \mathbf{Y}

Compare

```

 $\mathbf{Y}[\{\mathbf{A11}, 1\}]$ 
 $\mathbf{Y}[1, \mathbf{A11}]$ 
{Y1,1, Y2,1, Y3,1}
{Y1,1, Y1,2, Y1,3, Y1,4}

```

Note that unlike *Matlab*, *Mathematica* doesn't distinguish row from column formats. This can get confusing when switching between programming environments.

Symmetric matrices

For a square matrix, the diagonal elements remain the same under transpose.

If the transpose of a matrix equals itself, $\mathbf{H}^T = \mathbf{H}$, \mathbf{H} is said to be a **symmetric matrix**. 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). This means that the elements of a symmetric matrix \mathbf{H} look like they are reflected about the diagonal.

(The notion of a Hermitian matrix is a generalization of symmetric, where $\mathbf{H}^* = \mathbf{H}$, and \mathbf{H}^* is the transpose of the complex conjugate of \mathbf{H} . If interested in the terminology, look up imaginary numbers.)

Neural networks and symmetric connections

Do neural systems have symmetric connections?

We constructed a symmetric matrix in Lecture 5 when we modeled the exponential drop-off in lateral inhibition:

```

 $\mathbf{H} = \text{Table}[\mathbf{N}[\text{Exp}[-\text{Abs}[i-j]], 1], \{i, 5\}, \{j, 5\}];$ 
 $\%//\text{TraditionalForm}$ 

```

$$\begin{pmatrix} 1. & 0.4 & 0.1 & 0.05 & 0.02 \\ 0.4 & 1. & 0.4 & 0.1 & 0.05 \\ 0.1 & 0.4 & 1. & 0.4 & 0.1 \\ 0.05 & 0.1 & 0.4 & 1. & 0.4 \\ 0.02 & 0.05 & 0.1 & 0.4 & 1. \end{pmatrix}$$

Most real neural networks probably do not have symmetric connections. However, when the nature of the processing would not be expected to favor a particular asymmetry, we might expect that there should be symmetric connections on average. We made this assumption when setting up our lateral inhibition weight matrix, as seen above for \mathbf{H} . Symmetric matrices have many nice properties, making them appealing to neural modelers on theoretical grounds. We'll see this later when we study the Hopfield networks. Lack of symmetry can have profound effects on the dynamics of non-linear networks and can produce instabilities, i.e. lack of simple steady-state solutions, such as "chaotic" trajectories of the state vector.

Outer product of two vectors

We've already seen that the inner product of two vectors produces a scalar. In the next lecture, we will begin our discussion of Hebbian learning in neural networks. In this context, the **outer product** of an input and output vector will be used to model synaptic modification. Consider two vectors:

```
fv = Table[fi, {i, 1, 3}]
```

```
gv = Table[gi, {i, 1, 3}]
```

{**f**₁, **f**₂, **f**₃}

{**g**₁, **g**₂, **g**₃}

The outer product is all of the pairwise products of the elements of **f** and **g** arranged in a nice (and special) order:

```
h = Outer[Times, fv, gv];
```

```
%//TraditionalForm
```

$$\begin{pmatrix} f_1 g_1 & f_1 g_2 & f_1 g_3 \\ f_2 g_1 & f_2 g_2 & f_2 g_3 \\ f_3 g_1 & f_3 g_2 & f_3 g_3 \end{pmatrix}$$

To get a preview of things to come, we'll see how to model some aspects of learning using the idea of a "Hebbian synapse". The idea in Hebbian learning is that an adjustment to the strength of a connection between say the *i*th and *j*th neurons is proportional to the product of the strength of pre- and post-synaptic activities, represented by g_i and f_j .

The outer product is also written in traditional row and column format as: $\mathbf{f} \mathbf{g}^T$

$$\mathbf{f} \mathbf{g}^T = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} \begin{pmatrix} g_1 & g_2 & g_3 \end{pmatrix} = \begin{pmatrix} f_1 g_1 & f_1 g_2 & f_1 g_3 \\ f_2 g_1 & f_2 g_2 & f_2 g_3 \\ f_3 g_1 & f_3 g_2 & f_3 g_3 \end{pmatrix}$$

Recall that the dot product can be written in traditional form as: $\mathbf{a}^T \mathbf{b}$.

We'll also use outer product to calculate autocorrelation matrices.

Try using `Outer[]` with a different operation, such as `Plus`

Eigenvectors and eigenvalues

Basic idea

Eigenvalues and eigenvectors crop up in many ways. For example, the solution to a set of linear first-order differential equations (i.e. no derivative orders bigger than 1) can be found in terms of eigenvalues and eigenvectors. For a simplified version of the limulus equations

$$\frac{d\mathbf{f}}{dt} = \mathbf{A} \cdot \mathbf{f}$$

the solution is determined by the eigenvalues and eigenvectors of \mathbf{A} and the initial conditions.

Eigenvectors also crop up in statistics and related neural network algorithms. For example, principal components analysis (PCA) is used in dimensionality reduction in statistics--also important in neural networks and statistical analysis of natural patterns of images and sounds.

So what are eigenvalues, eigenvectors? An eigenvector, \mathbf{x} , of a square matrix, \mathbf{A} , is vector that when you multiply it by \mathbf{A} , you get an output vector that points in the same (or opposite, depending on the sign of λ) direction as \mathbf{x} :

$$\mathbf{Ax} = \lambda\mathbf{x}$$

where λ is an *eigenvalue*--a scalar that adjusts the length change of \mathbf{x} . The eigenvalues are found by solving:

$$\text{Det}[\mathbf{A}-\lambda\mathbf{Id}]=0, \text{ for } \lambda \text{ where } \mathbf{Id} \text{ is the identity matrix.}$$

Thought experiment. What forms of images remain unchanged if you blur them? These patterns are the eigenvectors of the linear transformation for blur.

Mathematica provides a one-step function to return the eigenvalues and eigenvectors of a matrix. But let's review how one might go about finding the eigenvalues of a matrix \mathbf{A} in smaller steps:

$$\mathbf{A} = \{\{1, 2\}, \{3, 4\}\};$$

$$\mathbf{Id} = \text{IdentityMatrix}[2]$$

$$\{\{1, 0\}, \{0, 1\}\}$$

$$\text{Solve}[\text{Det}[\mathbf{A} - \lambda * \mathbf{Id}] == 0, \lambda]$$

$$\left\{ \left\{ \lambda \rightarrow \frac{1}{2} (5 - \sqrt{33}) \right\}, \left\{ \lambda \rightarrow \frac{1}{2} (5 + \sqrt{33}) \right\} \right\}$$

Now pick one of the solutions for λ , find the vector $\mathbf{x}=\{x_1, x_2\}$ that satisfies the basic definition, $\mathbf{A}\mathbf{x}=\lambda\mathbf{x}$

$$\text{Solve}\left[\left\{\mathbf{A} \cdot \{\mathbf{x}_1, \mathbf{x}_2\} == \frac{1}{2} (5 - \sqrt{33}) * \{\{1, 0\}, \{0, 1\}\} \cdot \{\mathbf{x}_1, \mathbf{x}_2\}\right\}, \{\mathbf{x}_1\}\right] // \text{Simplify}$$

$$\left\{ \left\{ \mathbf{x}_1 \rightarrow -\frac{1}{6} (3 + \sqrt{33}) \mathbf{x}_2 \right\} \right\}$$

So any multiple of

$$\mathbf{x} = \left\{ 1, -\frac{1}{6} (\sqrt{33} + 3) \right\}$$

$$\left\{ 1, \frac{1}{6} (-3 - \sqrt{33}) \right\}$$

is a valid eigenvector.

From algebra, because the $\text{Det}[\mathbf{J}]=0$ is a polynomial of order n , there can't be any more than n distinct solutions, i.e. eigenvectors for an $n \times n$ matrix--and there may be less.

We can similarly find the second distinct eigenvector for $\lambda \rightarrow \frac{1}{2} (5 + \sqrt{33})$, but *Mathematica* provides a

simpler way of getting eigenvalues and eigenvectors.

Built-in functions for finding eigenvalues and eigenvectors

Eigenvalues[]

The eigenvalues are given by:

```
Eigenvalues[A]
```

$$\left\{ \frac{1}{2} (5 + \sqrt{33}), \frac{1}{2} (5 - \sqrt{33}) \right\}$$

Eigenvalues and eigenvector elements do not have to be real numbers. They can be complex, that is an element can be the sum of a real and imaginary number. Imaginary numbers are represented by multiples (or fractions) of **I**, the square root of -1:

```
Sqrt[-1]
```

```
Sqrt[-1]//OutputForm
```

```
i
```

```
I
```

```
B = {{1,2},{-3,4}};
```

```
Eigenvalues[B]
```

$$\left\{ \frac{1}{2} (5 + i \sqrt{15}), \frac{1}{2} (5 - i \sqrt{15}) \right\}$$

Eigenvectors[]

The built-in *Mathematica* function **Eigenvectors**[A] returns the eigenvectors of matrix **A** as the rows of a matrix, lets call **eig**:

```
eig = Eigenvectors[A]
```

$$\left\{ \left\{ \frac{1}{6} (-3 + \sqrt{33}), 1 \right\}, \left\{ \frac{1}{6} (-3 - \sqrt{33}), 1 \right\} \right\}$$

Verify that **eig[[1]]** and **A.eig[[1]]** lie along the line (i.e. in the same or opposite directions) by taking the dot product of the unit vectors pointing in the directions of each. Use the **Normalize**[] function:

Preview of eigenvectors and dimensionality reduction

A measure of the structure of an ensemble of signals (e.g. of vectors) is the degree of correlation between their elements. Signals, such as sounds and visual images, have correlational structure that is taken advantage of in sound and image compression algorithms. One simple kind of statistical structure is characterized by the degree to which one can predict one element of a vector from a nearby element. For images, the color of a pixel at location *i* is a pretty good predictor of the color at location *j* = *i*+1. As *j* gets far from *i*, however, the prediction gets worse. It is possible to characterize the degree of predictability by a matrix whose *ij*th element is big if the *i*th pixel of an image is a good predictor of the *j*th. Natural sounds also have a similar correlational structure. One measure of predictability is the autocorrelation

matrix where, for example, the i,j th element represents the degree of correlation between the i th and j th pixels, i.e. that are $|j - i|$ apart. An autocorrelation matrix has real elements and is symmetric. The eigenvectors of the matrix capture the subspace with most of the variance, i.e. where the "action" is. The eigenvalues correspond to the amount of "action"--i.e. the variance in the eigenvector directions.

Verify that the eigenvectors of the symmetric matrix above (**H**) are orthogonal.

Practice with useful matrix operations and simple functional programming

Rotations

Find the *Mathematica* function that outputs a rotation matrix.

Look up `RotationTransform`.

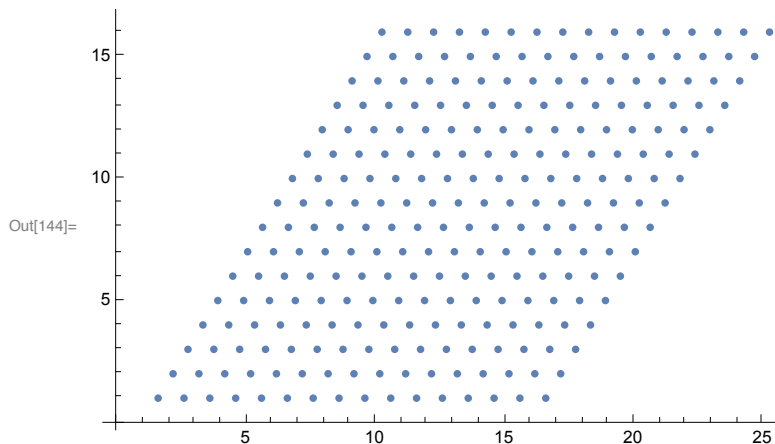
Find related transformations, for **scaling**, **shearing**, and **translation**.

Look up **affine** transforms.

Using `Flatten[]`, `Table[]`, `ListPlot[]` and `Dot[]` together with a pure function, `Function[]` (or its short-hand `&` with `#`), and `Map[]` (or its short-hand `/@`), make a 16x16 grid of coordinates, and apply `ShearingMatrix[30 Degree, {1,0}, {0,1}]` to it.

```
grid = Flatten[Table[{i, j}, {i, 1, 16}, {j, 1, 16}], 1];
ListPlot[grid, Axes -> False];
```

It should look like this:



Now try operating on the same grid with a `RotationMatrix` of your choice.

What happens if you put arbitrary numbers in a 2x2 matrix and apply it to elements in the grid?

Next time

Linear systems analysis & introduction to learning/memory

In the next lecture, we'll apply what we've learned about matrices and vectors to two problems. We'll first take an in-depth look at the properties of the linear neural network to see how it relates to general linear systems. We'll also show the advantages of using the eigenvectors of the linear transformation matrix as a useful coordinate system to represent input and output pattern vectors. We will introduce the topic of learning and memory, and see how the outer product rule can be used to model associative learning.

Preview of linear systems analysis

Linear systems are an important, and tractable class of input/output models. Matrix operations on vectors provide the prototype linear system.

Consider the generic 2-layer network, i.e. with 1 input set of units, and 2 output set. Each output neuron takes a weighted average of the inputs (stage 1), followed by a point-nonlinearity (the squash function of stage 2), and added noise (stage 3). Although later we will see how the non-linearity enables computations that are not possible without it, useful functions can be realized with just the linear or stage 1 part of the network. We've seen one application already with the model of the limulus eye. In the next lecture, we will see how linear networks can be used to model associative memory. But first, let us take what we've learned so far about modeling linear networks and look at in the general context of linear systems theory. Our 2-layer net is a matrix of weights that operates on a vector of input activities by computing a weighted sum. One property of such a system is that it satisfies the fundamental definition of a "linear system", which we will define shortly.

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 non-linear systems can be approximated by a linear one over some restricted range of parameter values.

So what is a "linear system"? The notion of a "linear system" is a generalization of the input/output properties of a straight line passing through zero. The matrix equation $\mathbf{W}\cdot\mathbf{x} = \mathbf{y}$ is a linear system. This means that if \mathbf{W} is a matrix, $\mathbf{x1}$ and $\mathbf{x2}$ are vectors, and a and b are scalars:

$$\mathbf{W}\cdot(a \mathbf{x1} + b \mathbf{x2}) = a \mathbf{W}\cdot\mathbf{x1} + b \mathbf{W}\cdot\mathbf{x2}$$

This is a consequence of the laws of matrix algebra. The idea of a linear system has been generalized beyond matrix algebra. Imagine we have a box that takes inputs such as f , and outputs $g = T[f]$.

The abstract definition of a linear system is that it satisfies:

$$T[a f + b g] = a T[f] + b T[g]$$

where T is the transformation that takes the sum of scaled inputs f , g (which can be functions or vectors) to the sum of the scaled transformation of f and g . The property, that the output of a sum is the sum of the outputs, is sometimes known as the *superposition principle* for linear systems. The fact that linear systems show superposition is good for doing theory, and modeling some kinds of neural processing, but as we will see later, superposition limits the kind of computations that can be done with linear systems, and thus with linear neural network models.

References

Linear Algebra and Its Applications by Gilbert Strang, 3rd Edition , Hardcover, 505 pages, Published by Hbj College & School Div. Publication date: February 1988

© 1998, 2001, 2003, 2005, 2007, 2009, 2011, 2012, 2014 Daniel Kersten, Computational Vision Lab, Department of Psychology, University of Minnesota.