Introduction to Neural Networks
U. Minn. Psy 5038

Daniel Kersten

Introduction

Last time

Modeled aspects of spatial lateral inhibition in the visual system:

Simple linear network: intensity->spatial filter->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 interperted 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 neurons that aren't connected get fixed zero weights.

This model is what we called a linear feedforward network, and used the generic neuron model.

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

We studied the continous system by approximating it as a discrete time system with $\epsilon = \Delta t$ as a free parameter.

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. But 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 in dimensionality reduction.

Basic matrix arithmetic

Definition of a matrix: a list of scalar lists

■ 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. E.g.

In[1]:=
```
H = Table[i^2 + j^2, {i, 1, 3}, {j, 1, 3}]
```

Out[1]=

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

And we can view it in the traditional MatrixForm either by: **Table[i^2+j^2,{i,1,3},{j,1,3}]//MatrixForm**, or

In[2]:=
```
MatrixForm[H]
```

Out[2]//MatrixForm=

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

Before going on, use the Cell menu to set: **Cell->Default Output FormatType->TraditionalForm**. This will cause vectors and matrices to be printed out in traditional mathematics format. Then we don't have to explicitly type MatrixForm every time to see the output in traditional matrix form. But if you want to see the output in standard *Mathematica* form, use **//StandardForm** or **StandardForm[ ]**.

■ 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 mxn matrix has m rows, and n columns. Here is a 3x4 matrix of symbolic elements w[i,j]:

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

Out[3]=
$$\begin{pmatrix} w(1,1) & w(1,2) & w(1,3) & w(1,4) \\ w(2,1) & w(2,2) & w(2,3) & w(2,4) \\ w(3,1) & w(3,2) & w(3,3) & w(3,4) \end{pmatrix}$$

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

In[4]:=
```
L = Table[w_{i,j}, {i, 1, 3}, {j, 1, 4}]
```

Out[4]=
$$\begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \end{pmatrix}$$

Mathematica lists can have elements of almost any type:

In[5]:=
```
w_{2,3} = 3.0;
w_{3,4} = Pi;
w_{2,4} = "hi";
w_{1,1} = 3;
```

In[9]:=
```
L
```

Out[9]=
$$\begin{pmatrix} 3 & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & 3. & \text{hi} \\ w_{3,1} & w_{3,2} & w_{3,3} & \pi \end{pmatrix}$$

Adding, subtracting and multiplying by a scalar

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

In[10]:=
```
A = {{a,b},{c,d}};
B = {{x,y}, {u,v}};
```

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

In[12]:=
```
A+B
```

Out[12]=
$$\begin{pmatrix} a+x & b+y \\ c+u & d+v \end{pmatrix}$$

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

In[13]:=
```
A-B
```

Out[13]=
$$\begin{pmatrix} a-x & b-y \\ c-u & d-v \end{pmatrix}$$

Multiply **A** by 3:

In[14]:=
```
3 A
```

Out[14]=
$$\begin{pmatrix} 3a & 3b \\ 3c & 3d \end{pmatrix}$$

Multiplying two matrices

We have already seen how to multiply a vector by a matrix: we replace the i^{th} row of the output vector by the inner product of the i^{th} row of the matrix with the 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[15]:=
```
A.B
```

Out[15]=
$$\begin{pmatrix} bu+ax & bv+ay \\ du+cx & dv+cy \end{pmatrix}$$

Note that **AB** is not necessarily equal to **BA**:

In[16]:=
```
B.A
```

Out[16]=

$$\begin{pmatrix} a x + c y & b x + d y \\ a u + c v & b u + d v \end{pmatrix}$$

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 **ax+bu** to equal **ax + cy**. That is, 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. Matrix addition and subtraction do commute. Matrix multiplication is associative, so (**AB**)**C = A**(**BC**). The distributive law works too:

A(B+C) = AB + AC

Non-square matrices

It is not necessary for **A** and **B** to be square matrices (i.e. have the same number of rows as columns) to multiply them. But if **A** is an mxn matrix, then **B** has to be an nxp matrix in order for **AB** to make sense. For example, here **F** is a 3x2 matrix, and **G** is a 2x4 matrix.

In[17]:=
```
F = {{a,b},{c,d},{e,f}}
G = {{p,q,r,s},{t,u,v,w}}
```

Out[17]=

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

Out[18]=

$$\begin{pmatrix} p & q & r & s \\ t & u & v & w \end{pmatrix}$$

Check the dimensions with

In[19]:=
```
Dimensions[F]
Dimensions[G]
```

Out[19]= {3, 2}

Out[20]= {2, 4}

Because **F** has 2 columns, and **G** has 2 rows, it makes sense to multiply **G** by **F**:

In[21]:=
```
Print[F, G, " =", F.G]
```

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix}\begin{pmatrix} p & q & r & s \\ t & u & v & w \end{pmatrix} = \begin{pmatrix} a p + b t & a q + b u & a r + b v & a s + b w \\ c p + d t & c q + d u & c r + d v & c s + d w \\ e p + f t & e q + f u & e r + f v & e s + f w \end{pmatrix}$$

However, because the number of columns of **G** (4) do not match the number of rows of **F** (3), **G.F** is not well-defined:

Try this:

In[22]:=
```
Print[F, G, " =", G.F]
```

Dot::dotsh : Tensors $\begin{pmatrix} p & q & r & s \\ t & u & v & w \end{pmatrix}$ and $\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix}$ have incompatible shapes.

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix}\begin{pmatrix} p & q & r & s \\ t & u & v & w \end{pmatrix} = \begin{pmatrix} p & q & r & s \\ t & u & v & w \end{pmatrix}.\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix}$$

Inverse of a Matrix

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:

In[23]:=
```
IdentityMatrix[2]
```
Out[23]= $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

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

How can one divide one matrix, say **B**, by another, say **A**? We can divide numbers, x by y, by multiplying x times the inverse of y, i.e. 1/y. So to do the equivalent of dividing **B** by **A**, we need to find a matrix **Q** 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**.

In[24]:=
```
A = {{a,b},{c,d}};
```

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

In[25]:=
```
Q = Inverse[A]
```
Out[25]= $\begin{pmatrix} \frac{d}{a d-b c} & -\frac{b}{a d-b c} \\ -\frac{c}{a d-b c} & \frac{a}{a d-b c} \end{pmatrix}$

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.

In[26]:=
```
Q.A
```
Out[26]= $\begin{pmatrix} \frac{a d}{a d-b c}-\frac{b c}{a d-b c} & 0 \\ 0 & \frac{a d}{a d-b c}-\frac{b c}{a d-b c} \end{pmatrix}$

In[27]:=
```
Simplify[Q.A]
```
Out[27]= $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

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

In[28]:=
```
B = {{1,-1},{3,2}};
R = Inverse[B]
```
Out[29]= $\begin{pmatrix} \frac{2}{5} & \frac{1}{5} \\ -\frac{3}{5} & \frac{1}{5} \end{pmatrix}$

Singular and badly conditioned matrices

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

In[30]:=
```
B1 = {{1.5,1},{3,2.0}}
```
Out[30]= $\begin{pmatrix} 1.5 & 1 \\ 3 & 2. \end{pmatrix}$

Then the rows of the matrix are not linearly independent. In this case, the inverse is not defined.

Ask for the inverse of B1

In[31]:=
```
Inverse[B1]
```
Inverse::sing : Matrix $\begin{pmatrix} 1.5 & 1. \\ 3. & 2. \end{pmatrix}$ is singular.

Out[31]= $\begin{pmatrix} 1.5 & 1 \\ 3 & 2. \end{pmatrix}^{-1}$

Sometimes the rows are almost, but not quite, linearly dependent (because the elements are represented as approximate floating point approximations to the actual values). *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.

Let's try finding the inverse of the following matrix:

In[32]:=
```
B2 = {{-2,-1},{4.00000000000001,2.0}};
Inverse[B2]
```

Inverse::luc :
Result for Inverse of badly conditioned matrix $\begin{pmatrix} -2. & -1. \\ 4. & 2. \end{pmatrix}$ may contain significant numerical errors.

Out[33]= $\begin{pmatrix} 2.04709\times 10^{14} & 1.02355\times 10^{14} \\ -4.09418\times 10^{14} & -2.04709\times 10^{14} \end{pmatrix}$

Were we lucky or not?

In[34]:=
```
B2.Inverse[B2]
```

Inverse::luc :
Result for Inverse of badly conditioned matrix $\begin{pmatrix} -2. & -1. \\ 4. & 2. \end{pmatrix}$ may contain significant numerical errors.

Out[34]= $\begin{pmatrix} 1. & 0. \\ 0. & 1. \end{pmatrix}$

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?

In[35]:=
```
Det[B2]
```

Out[35]= 9.76996×10^{-15}

Why do we get a zero determinant for **B1**, but not for **B2**?

In[36]:=
```
Det[B1]
```

Out[36]= 0.

Matrix transpose

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

In[37]:=
```
Y = Table[yi,j , {i, 1, 3}, {j, 1, 4}]
```

Out[37]= $\begin{pmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} \end{pmatrix}$

You may have noticed that in the Cell menu, you can convert to various cell types.

In StandardForm on the input line, the transpose is written:

In[38]:=
```
Transpose[Y]
```

Out[38]= $\begin{pmatrix} y_{1,1} & y_{2,1} & y_{3,1} \\ y_{1,2} & y_{2,2} & y_{3,2} \\ y_{1,3} & y_{2,3} & y_{3,3} \\ y_{1,4} & y_{2,4} & y_{3,4} \end{pmatrix}$

The output default is TradtionalForm. On the input line, in TraditionalForm, the transpose is written:

In[39]:= Y^T // StandardForm

Out[39]//StandardForm=
```
{{y1,1 , y2,1 , y3,1 }, {y1,2 , y2,2 , y3,2 }, {y1,3 , y2,3 , y3,3 }, {y1,4 , y2,4 , y3,4 }}
```

What do the ouputs, X and X^T look like in StandardForm? (Make sure your input form is appropriate)

Getting parts of a matrix

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[40]:=
```
Y[[2]]
```
Out[40]= $\{y_{2,1}, y_{2,2}, y_{2,3}, y_{2,4}\}$

What about ith column of a matrix? There is no equally simple way of getting the column of a matrix in *Mathematica*, but we can use the transpose operation to do it. **Transpose[W] [[i]]** produces the ith column of matrix **W**. For example, the 3nd column of **X** is:

In[41]:=
```
Transpose[Y][[3]]
```
Out[41]= $\{y_{1,3}, y_{2,3}, y_{3,3}\}$

Try putting **//ColumnForm** after the above expression.

Pull out the element $x_{2,3}$ from Y

In[42]:=
```
Y[[2, 3]]
```
Out[42]= $y_{2,3}$

Symmetric matrices

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

If the transpose of a matrix equals itself, $\mathbf{H}^T = \mathbf{H}$, **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 **H** actually look like they are reflected about the diagonal. We constructed a symmetric matrix in Lecture 5:

In[43]:=
```
H = Table[N[Exp[-Abs[i-j]],1],{i,5},{j,5}]
```
Out[43]=
$$\begin{pmatrix} 1. & 0.367879 & 0.135335 & 0.0497871 & 0.0183156 \\ 0.367879 & 1. & 0.367879 & 0.135335 & 0.0497871 \\ 0.135335 & 0.367879 & 1. & 0.367879 & 0.135335 \\ 0.0497871 & 0.135335 & 0.367879 & 1. & 0.367879 \\ 0.0183156 & 0.0497871 & 0.135335 & 0.367879 & 1. \end{pmatrix}$$

The notion of a Hermitian matrix is a generalization of symmetric, where H* = H, and H* is the transpose of the complex conjugate of H.

Neural networks and symmetric connections

Do neural systems have symmetric connections? Real neural networks probably do not in general. Although, 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 **H**. Symmetric matrices have so many nice properties, that neural modelers (especially those from physics backgrounds) find the symmetry assumption almost irresistable. 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 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:

In[44]:=
```
fv = Table[fi, {i, 1, 3}];
gv = Table[gi, {i, 1, 3}];
```

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

In[46]:=
```
h = Outer[Times,fv,gv]
```
Out[46]=
$$\begin{pmatrix} f_1 g_1 & f_1 g_2 & f_1 g_3 \\ f_2 g_1 & f_2 g_2 & f_2 g_3 \\ f_3 g_1 & f_3 g_2 & f_3 g_3 \end{pmatrix}$$

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

$$\mathbf{f}\mathbf{g}^T = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} (g_1\ g_2\ g_3) \qquad (1)$$

Eigenvectors and eigenvalues

Basic idea

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

$$\frac{df}{dt} = A.f \qquad (2)$$

the solution is determined by the eigenvalues and eigenvectors of A and the initial conditions.

Eigenvectors also crop up in statistics. Principal components analysis is used in dimensionality reduction--also important in neural networks and "natural computation".

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

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

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

Det[A-λId]==0, for $\lambda$where **Id** is the identity matrix.

Let's find the eigenvalues of:

In[47]:=
```
A = {{1,2},{3,4}};
```

In[48]:=
```
Id = IdentityMatrix[2]
```

Out[48]= $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

In[49]:=
```
Solve[Det[A - λ * Id] == 0, λ]
```

Out[49]= $\{\{\lambda \to \frac{1}{2}(5-\sqrt{33})\}, \{\lambda \to \frac{1}{2}(5+\sqrt{33})\}\}$

Now pick one of the solutions for λ, find the vector x={x_1 ,x_2 } that satisfies the basic definition, **A.x=λx**

In[50]:=
```
Solve[{A.{x1, x2} == 1/2 (5 - √33) * {{1, 0}, {0, 1}}.{x1, x2}}, {x1, x2}]
```

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

Out[50]= $\{\{x_1 \to -\frac{1}{6}(3+\sqrt{33})\,x_2\}\}$

So any multiple of

In[51]:=
```
x = {1, -1/6 (√33 + 3)}
```

Out[51]= $\{1, \frac{1}{6}(-3-\sqrt{33})\}$

is a valid eigenvector.

Because the Det[]==0 is a polynomial of order n, there can't be any more than n distinct eigenvectors for an nxn matrix--and there may be less.

We can similarly find the second eigenvector for $\lambda \to \frac{1}{2}(5+\sqrt{33})$, but *Mathematica* provides a simpler way of doing things.

Built-in functions for finding eigenvalues and eigenvectors

■ Eigenvalues[ ]

The eigenvalues are given by:

In[52]:=
```
Eigenvalues[A]
```

Out[52]= $\{\frac{1}{2}(5-\sqrt{33}), \frac{1}{2}(5+\sqrt{33})\}$

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:

In[53]:=
```
Sqrt[-1]
Sqrt[-1]//OutputForm
```

Out[53]= i

Out[54]//OutputForm=
```
I
```

In[55]:=
```
B = {{1,2},{-3,4}};
Eigenvalues[B]
```

Out[56]= $\left\{\frac{1}{2}\left(5 - i\sqrt{15}\right), \frac{1}{2}\left(5 + i\sqrt{15}\right)\right\}$

■ **Eigenvectors[ ]**

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

In[57]:=
```
eig = Eigenvectors[A]
```

Out[57]= $\begin{pmatrix} \frac{1}{6}\left(-3-\sqrt{33}\right) & 1 \\ \frac{1}{6}\left(-3+\sqrt{33}\right) & 1 \end{pmatrix}$

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 this function:

In[58]:=
```
normalize[x_] := x/Sqrt[x.x];
```


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}. One measure of predictability is the autocorrelation matrix. An autocorrelation matrix has real elements and is symmetric. The eigenvectors of 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.

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. It consists of 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 ty 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 **W.x == y** is a linear system. This means that if **W** is a matrix, **x1** and **x2** are vectors, and a and b are scalars:

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

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 satsifies:

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

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, 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.

References

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