Introduction to Neural Networks
U. Minn. Psy 5038
Spring, 1999
Daniel Kersten
Lecture 6: Matrices

Introduction

We have seen how the synaptic connections between neurons can be represented by a connection matrix. The idea is to represent the synaptic weights for the i^{th} output neuron by the values in the i^{th} row of the connection matrix. Because of the large body of mathematical results on matrix algebra, 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). Towards the end 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.

Basic matrix arithmetic

Definition of a matrix: a list of scalar lists

An mxn matrix has m rows, and n columns. Here is a 3x4 matrix:

```
MatrixForm[
    Table[W[i,j],{i,1,3},{j,1,4}]]
```

$$\begin{pmatrix} W(1,1) & W(1,2) & W(1,3) & W(1,4) \\ W(2,1) & W(2,2) & W(2,3) & W(2,4) \\ W(3,1) & W(3,2) & W(3,3) & W(3,4) \end{pmatrix}$$

The matrix can be written in standard form using subscripts as:

$$\begin{pmatrix} w_{11} w_{12} w_{13} w_{14} \\ w_{21} w_{22} w_{23} w_{24} \\ w_{31} w_{32} w_{33} w_{34} \end{pmatrix}$$

And as we have seen, *Mathematica* represents a matrix as a list of lists:

```
%//StandardForm
```

```
{{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]}}
```

Remember, the symbol % in *Mathematica* stands for the most recent output, %% stands for the second to last output, and so forth.

Adding, subtracting and multiplying by a scalar

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

```
A = {{a,b},{c,d}};
B = {{x,y}, {u,v}};
```

Let's add **A** to **B**:

```
A+B
```

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

Subtracting **B** from **A**:

```
A-B
```

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

And if we multiply **A** by 3:

```
3 A
```

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

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**:

```
A.B
```

$$\begin{pmatrix} bu + ax & bv + ay \\ du + cx & dv + cy \end{pmatrix}$$

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

```
B.A
```

$$\begin{pmatrix} ax + cy & bx + dy \\ au + cv & bu + dv \end{pmatrix}$$

Laws of commutation, association and distribution

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.

```
F = {{a,b},{c,d},{e,f}};
G = {{p,q,r,s},{t,u,v,w}};
```

```
Dimensions[F]
```

{3, 2}

```
Dimensions[G]
```

{2, 4}

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

```
F.G
```

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

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:

```
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.

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

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:

```
IdentityMatrix[2]
```

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

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**.

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

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

```
Q = Inverse[A]
```

$$\begin{pmatrix} \frac{d}{ad-bc} & -\frac{b}{ad-bc} \\ -\frac{c}{ad-bc} & \frac{a}{ad-bc} \end{pmatrix}$$

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.

```
Q.A
```

$$\begin{pmatrix} \frac{ad}{ad-bc} - \frac{bc}{ad-bc} & 0 \\ 0 & \frac{ad}{ad-bc} - \frac{bc}{ad-bc} \end{pmatrix}$$

```
Simplify[Q.A]
```

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

Here is a simple numerical example:

```
B = {{1,-1},{3,2}};
R = Inverse[B]
```

$$\begin{pmatrix} \frac{2}{5} & \frac{1}{5} \\ -\frac{3}{5} & \frac{1}{5} \end{pmatrix}$$

```
B.R
```

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

Badly conditioned matrices

What if one row is a scaled version of another row? Then the rows are not linearly independent. In this case, the inverse is not defined.

```
B1 = {{1.5,1},{3,2.0}}
```

$$\begin{pmatrix} 1.5 & 1 \\ 3 & 2. \end{pmatrix}$$

```
Inverse[B1]
```

Inverse::sing : Matrix $\begin{pmatrix} 1.5 & 1 \\ 3 & 2. \end{pmatrix}$ is singular.

$$\begin{pmatrix} 1.5 & 1 \\ 3 & 2. \end{pmatrix}^{-1}$$

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:

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

Inverse::luc : Result for Inverse of badly conditioned matrix $\begin{pmatrix} -2 & -1 \\ 4.00000000000001 & 2. \end{pmatrix}$ may contain significant numerical errors.

$$\begin{pmatrix} 2.04709 \times 10^{14} & 1.02355 \times 10^{14} \\ -4.09418 \times 10^{14} & -2.04709 \times 10^{14} \end{pmatrix}$$

```
B2 . Inverse[B2]
```

Inverse::luc : Result for Inverse of badly conditioned matrix $\begin{pmatrix} -2 & -1 \\ 4.00000000000001 & 2. \end{pmatrix}$ may contain significant numerical errors.

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

Were we lucky or not?

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]
```

$$9.76996 \times 10^{-15}$$

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

```
Det[B1]
```

0.

Matrix transpose

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

```
X = Table[w_{i,j}, {i, 1, 3}, {j, 1, 4}]
```

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

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:

```
Transpose[X];
```

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

$$X^T$$

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

What do the ouputs, **X** and X^T look like in StandardForm, MatrixForm?

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 **X** is:

```
X[[2]]
```

$$\{w_{2,1}, w_{2,2}, w_{2,3}, w_{2,4}\}$$

What about i^{th} 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:

```
Transpose[X][[3]]
```

$$\{w_{1,3}, w_{2,3}, w_{3,3}\}$$

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:

```
H = Table[N[Exp[-Abs[i-j]],1],{i,5},{j,5}]
```

$$\begin{pmatrix} 1. & 0.4 & 0.2 & 0.05 & 0.01 \\ 0.4 & 1. & 0.4 & 0.2 & 0.05 \\ 0.2 & 0.4 & 1. & 0.4 & 0.2 \\ 0.05 & 0.2 & 0.4 & 1. & 0.4 \\ 0.01 & 0.05 & 0.2 & 0.4 & 1. \end{pmatrix}$$

■ 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:

```
f = {f1,f2,f3};
g = {g1,g2,g3};
```

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

```
h = Outer[Times,f,g]
```

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

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

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

Eigenvectors and 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 direction as **x**:

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

where λ is a scalar that adjusts the length change of **x**.

There can't be any more than n distinct eigenvectors for an nxn matrix--and there may be less.

```
A = {{1,2},{3,4}};
```

The *Mathematica* function **Eigenvectors[A]** returns the eigenvectors of matrix **A** as the rows of a matrix, which we'll call **eig**:

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

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

We can verify that **eig[[1]]** and **A.eig[[1]]** lie along the same direction by taking the dot product of the unit vectors pointing in the directions of each:

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

normalize[eig[[1]]].normalize[A.eig[[1]]];
N[%]
```

```
-1.
```

■ Eigenvalues

The eigenvalues are given by:

```
Eigenvalues[A]
```

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

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. In *Mathematica*, imaginary numbers are represented by multiples (or fractions) of **I**, the square root of -1:

```
Sqrt[-1]
Sqrt[-1]//StandardForm
```

$$i$$

```
I
```

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

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

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.

Also next time, 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.

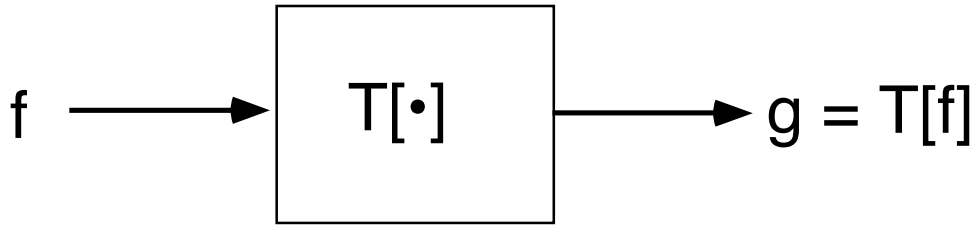
The world of input/output systems can be divided up into linear and non-linear systems. Linear systems are nice because the mathematics that describes them is not only well-known, but also has a mature elegance. On the other hand, it is a fair statement to say that most real-world systems are not linear, 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 exactly 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.

Optional Exercises

Exercise 1: Fun with eigenvectors: Preview of self-organization

■ Autocorrelation matrix is symmetric

Eigenvectors crop up in many different domains. In statistics, 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 ijth element is big if the ith pixel of an image is a good predictor of the jth. One measure of predictability is the autocorrelation matrix:

W = Average[Outer[Times,x1,x1], Outer[Times,x2,x2],...]

Later on in this course, we will see how neural networks can be devised that find the hidden structure in a set of vectors **x1, x2**....by analyzing the autocorrelation matrix. What these networks do is to self-organize to discover the eigenvectors of the autocorrelation matrix of the ensemble of patterns they are exposed to. The eigenvectors represent the "prototypical" dimensions along which the ensemble varies. Statisticians call this "Principal Components Analysis" or PCA.

I won't show how it all works yet, but let's try a simple exercise that demonstrates that there can be surprising structure buried in an autocorrelation matrix.

We mentioned earlier that symmetric matrices are nice. One reason is that:

the eigenvalues of a symmetric matrix are real and the eigenvectors are orthogonal (if they come from distinct eigenvalues).

I'd like to show you in this section that symmetric matrices are not only nice, but they can be almost magical.

Autocorrelation matrices are symmetric. A good model for the autocorrelation matrix for the intensity contrast of natural images is exponential drop-off:

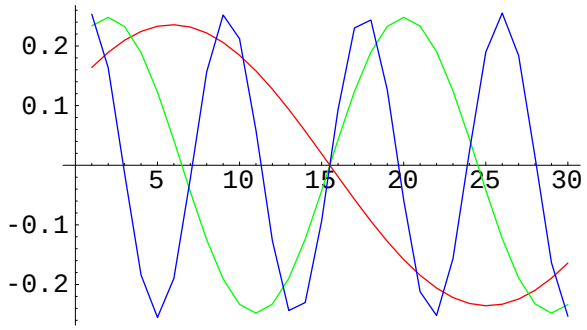
■ Plotting the eigenvectors of the symmetric matrix for exponential correlation drop-off

```
size = 30;
length = 5;
w   = Table[N[Exp[-Abs[i-j]/length],2],
        {i,size},{j,size}];
```

```
e = Eigenvectors[w];
```

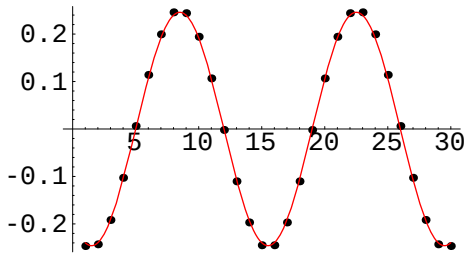
Let's plot a few of the eigenvectors of **w**:

```
g1 = ListPlot[e[[2]], PlotJoined -> True, PlotStyle ->
   {RGBColor[1, 0, 0]}, DisplayFunction -> Identity];
g2 = ListPlot[e[[4]],
 PlotJoined -> True, PlotStyle ->
   {RGBColor[0, 1, 0]}, DisplayFunction -> Identity];
g3 = ListPlot[e[[8]],
 PlotJoined -> True, PlotStyle ->
   {RGBColor[0, 0, 1]}, DisplayFunction -> Identity];
Show[g1, g2, g3, DisplayFunction -> $DisplayFunction];
```



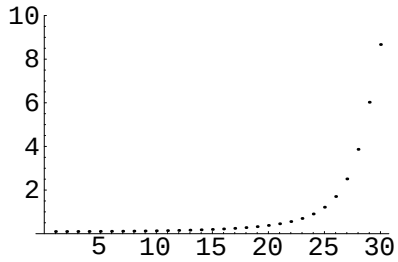
They sure look a lot like sinewaves, and in fact, they are pretty close. We will return to the significance of this later for models of self-organization of the weights of neurons in visual cortex.

```
g4 = ListPlot[e[[5]],PlotJoined->False,Prolog->PointSize[
0.02],DisplayFunction -> Identity];
g5 = Plot[-Max[e[[5]]] Cos[(2 Pi (x-1.5))/14],{x,1,30},PlotStyle ->
     {RGBColor[1, 0, 0]},DisplayFunction -> Identity];
Show[g4,g5, DisplayFunction -> $DisplayFunction];
```



The eigenvalues for **w** have an interesting form as well:

```
lambda = Sort[Eigenvalues[w]];
ListPlot[N[lambda,1],PlotRange->{0,10}];
```



Exercise 2

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

Exercise 3

Try replacing the exponential in **Exp[-Abs[i-j]]** with another function, thereby defining a completely different symmetric matrix. Compare the plots of the eigenvectors for this symmetric matrix with those from the one above.

Exercise 4

Show that the output vector of a linear neural network is the weighted sum of the columns of the weight matrix, where the weights are the input values.

Exercise 5

Construct an example of a 3x3 matrix in which the third row is a linear combination of the first two rows. Try to compute the inverse and determinant.

Notice that for bigger matrices, it isn't immediately obvious whether the rows of a matrix are linearly independent or not--and thus not immediately apparent whether the inverse exists. That is one reason why routines to calculate the determinant are useful.

References

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