Linear Algebra Review

CSC2515 – Machine Learning – Fall 2002

Abstract— **This tutorial note provides a quick review of basic linear algebra concepts. It is quite condensed, as it attempts to do in a few pages what Strang's book does very well in 500.**

I. Vectors and Matrices

Linear algebra is the study of vectors and matrices and how they can be manipulated to perform various calculations. What do the two words "linear" and "algebra" have to do with vectors and matrices? Consider functions which take several input arguments and produce several output arguments. If we stack up the input arguments into a vector $\mathbf{x}$ and the outputs into a vector $\mathbf{y}$ then a function $\mathbf{y} = f(\mathbf{x})$ is said to be *linear* if:

$$f(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha f(\mathbf{x}) + \beta f(\mathbf{y}) \tag{1}$$

for all scalars α, β and all vectors $\mathbf{x}, \mathbf{y}$. In other words, *scaling the input scales the output* and *summing inputs sums their outputs*. Now here is the amazing thing. All functions which are linear, in the sense defined above, can be written in the form of a matrix F which left multiplies the input argument $\mathbf{x}$:

$$\mathbf{y} = \mathrm{F}\mathbf{x} \tag{2}$$

Here F has as many rows as outputs and as many columns as inputs. Furthermore, all matrix relations like the one above represent linear functions from their inputs to their outputs. [Try to show both directions of this equivalence.]

Another interesting fact is that the composition of two linear functions is still linear [try to show this also]: $g(f(x)) = \mathrm{GF}\mathbf{x} = \mathrm{H}\mathbf{x} = h(\mathbf{x})$. This means that if we think of the inputs and outputs as values running along "wires" and the functions as "components" we can build any "circuit" we like (assuming the values on the wires add when they meet) and it will still be linear. The manipulations of matrix multiplication and vector addition correspond to running some wires through a component and to connecting wires together. This use of multiplication and addition of vectors is why we use the word "algebra" in linear algebra.

Hence the entire study of mutiple-input multiple-output linear functions can be reduced to the study of vectors and matrices.

II. Multiplication, Addition, Transposition

Adding up two vectors or two matrices is easy: just add their corresponding elements. (Of course the two things being added have to be exactly the same size.) Multiplying a vector or matrix by a scalar just multiplies each element by the scalar. So we are left with matrix-vector multiplication and matrix-matrix multiplications.

The best way to think of an n by m matrix F is as a machine that eats m sized vectors and spits out n sized vectors. This conversion process is known as "(left) multiplying by F" and has many similarities to scalar multiplication, but also a few differences. First of all, the machine only accepts inputs of the right size: you can't multiply just any vector by just any matrix. The length of the vector must match the number of columns of the matrix to its left (or the number of rows if the matrix is on the right of the vector).

We can flip, or "transpose" a matrix if we want to interchange its rows and columns. Usually we will write $\mathrm{F}^\top : (\mathrm{F}^\top)_{ij} = \mathrm{F}_{ji}$.

Like scalar multiplication, matrix multiplication is *distributive* and *associative*:

$$\mathrm{G}(\mathrm{F}\mathbf{a}) = (\mathrm{GF})\mathbf{a} \tag{3}$$
$$\mathrm{F}(\mathbf{a} + \mathbf{b}) = \mathrm{F}\mathbf{a} + \mathrm{F}\mathbf{b} \tag{4}$$
$$\tag{5}$$

Which means you can think of the matrix product GF as the equivalent linear operator you get if you compose the action of F followed by the action of G.

Matrix-matrix multiplication as a sequence of matrix-vector multiplications, one for each column whose results get stacked beside each other in columns to form a new matrix. In general, we can think of column vectors of length k as just k by 1 and row vectors as 1 by k matrices; this eliminates any distinction between matrix-matrix multiplication and matrix-vector multiplication.

Of course, unlike scalar multiplication, matrix multiplication is not *commutative*:

$$\mathrm{F}\mathbf{a} \neq \mathbf{a}\mathrm{F} \tag{6}$$

Multiplying a vector by itself gives a scalar $\mathbf{x}^\top\mathbf{x}$ which is known as the (squared) *norm* or squared length of the vector and is written $\|\mathbf{x}\|^2$. This measure adds up the sum of the squares of the elements of the vector. The *Frobenius norm* of a matrix $\|\mathrm{A}\|^2$ does the same thing, adding up the squares of all the matrix elements.

III. Inverses and Determinants

Two more important concepts to introduce before we get to use matrices and vectors for some real stuff. The first is the concept of reversing or undoing or *inverting* the function represented by a matrix A. For a function to be invertible, there needs to be a one-to-one relationship between inputs and outputs so that given the output you can always say exactly what the input was. In other words, we need a function which, when composed with A gives back the original vector. Such a function – if it exists – is called the *inverse* of A and the matrix corresponding to it is the *matrix inverse* or just *inverse* of A, denoted A^{-1}. In matrix terms, we seek a matrix that left multiplies A to give the identity matrix:

$$\mathrm{A}^{-1}\mathrm{A} = \mathrm{I} \tag{7}$$

where I is the *identity matrix* $\mathrm{I}_{ij} = \delta_{ij}$, corresponding to the identity (do-nothing) function.

Only a very few, special linear functions are invertible. For starters, they must have at least as many outputs as inputs (think about why), in other words the matrix must have at least as many rows as columns. Also, they must not map any two inputs to the same output. Technically this is means they must have *full rank*, a concept which is explained in the appendix.

The last important concept is that of a matrix determinant. This is a nonnegative scalar quantity, normally denoted $|\mathrm{A}|$ or $\det(A)$ which measures how much the matrix "stretches" or "squishes" volume as it transforms its inputs to its outputs. Matrices with large determinants do (on average) a lot of stretching and those with small determinants to a lot of squishing. Matrices with zero determinant have rank less than the number of rows and actually collapse some of their input space into a line or hyperplane (pancake) in the output space, and thus can be thought of as doing "infinite squishing". Conventionally, the determinant is only defined for square matrices, but there is a natural extension to rectangular ones using the *singular value decomposition* which is a topic for another chapter.

IV. Fundamental Matrix Equations

The two most important matrix equations are the system of linear equations:

$$\mathrm{A}\mathbf{x} = \mathbf{b} \tag{8}$$

and the eigenvector equation:

$$\mathrm{A}\mathbf{x} = \mathbf{x} \tag{9}$$

which betweeen them cover a large number of optimization and constraint satisfaction problems. As we've written them above, $\mathbf{x}$ is a vector but these equations also have natural extensions to the case where there are many vectors simultaneously satisfying the equation: $\mathrm{AX} = \mathrm{B}$ or $\mathrm{AX} = \mathrm{X}$.

V. Systems of Linear Equations

A central problem in linear algebra is the solution of a system of linear equations like this:

$$\begin{aligned} 3x + 4y + 2z &= 12 \\ x + y + z &= 5 \end{aligned}$$

Typically, we express this system as a single *matrix equation* something like this: $\mathrm{A}\mathbf{x} = \mathbf{b}$, where A is an m by n matrix, $\mathbf{x}$ is an n column vector and $\mathbf{b}$ is an m column vector. The number of unknowns is n and the number of equations or constraints is m. Here is another simple example:

$$\begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix} \tag{10}$$

How do we go about "solving" this system of equations? Well, if A is known, then we are trying to find an $\mathbf{x}$ corresponding to the $\mathbf{b}$ on the right hand side. (Why? Well, Finding $\mathbf{b}$ given A and $\mathbf{x}$ is pretty easy–just multiply. And for a single $\mathbf{x}$ there are usually a great many matrices A which satisfy the equation: one example – assuming the elements of $\mathbf{x}$ do not sum to zero – is $\mathbf{b}\mathbf{1}^\top / \sum(\mathbf{x})$. The only interesting question problem left, then, is to find $\mathbf{x}$.) This kind of equation is really a problem statement. It says "hey, we applied the function A and got the output $\mathbf{b}$; what was the input $\mathbf{x}$?" The matrix A is dictated to us by our problem, and represents our model of how the system we are studying converts inputs to outputs. The vector $\mathbf{b}$ is the output that we observe (or desire) – we know it. The vector $\mathbf{x}$ is the set of inputs – it is what we are trying to find.

Remember that there are two ways of thinking about this kind of equation. One is *rowwise* as a set of m equations, or constraints that correspond geometrically to m intersecting constraint surfaces:

$$\begin{bmatrix} 2x_1 - x_2 &= 1 \\ x_1 + x_2 &= 5 \end{bmatrix}$$

The goal is to find the point(s), for example (x_1, x_2) above, which are at the intersection of all the constraint surfaces. In the example above, these surfaces are two lines in the plane. If the lines intersect then there is a solution, if they are parallel, there is not, if they are coincident then there are infinite solutions. In higher dimensions there are more geometric cases, but in general there can be no solutions, one solution, or infinite solutions.

The other way is *columnwise* in which we think of the entire system as a single vector relation:

$$x_1 \begin{bmatrix} 2 \\ 1 \end{bmatrix} + x_2 \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$$

The goal here is to discover which linear combination(s) (x_1, x_2), if any, of the n column vectors on the left will give the one on the right.

Either way, the matrix equation $\mathrm{A}\mathbf{x} = \mathbf{b}$ is an almost ubiquitous problem whose solution comes up again and again in theoretical derivations and in practical data analysis problems. One of the most important applications is the minimization of quadratic energy functions: if A is symmetric positive definite then the quadratic form $\mathbf{x}^\top \mathrm{A}\mathbf{x} - 2\mathbf{x}^\top \mathbf{b} + \mathbf{c}$ is *minimized* at the point where $\mathrm{A}\mathbf{x} = \mathbf{b}$. Such quadratic forms arise often in the study of linear models with Gaussian noise since the log likelihood of data under such models is always a matrix quadratic.

A. *Least squares: solving for* $\mathbf{x}$

Consider the case of a single $\mathbf{x}$ first. Geometrically you can think of the rows of of the system as encoding constraint surfaces in which the solution vector $\mathbf{x}$ must lie and what we are looking for is the point(s) at which these surfaces intersect. Of course, they may not intersect, in which case there is no exact solution satisfying the equation; then we typically need some way to pick the "best" approximate solution. The constraints may also intersect along an entire line or surface in which case there are an infinity of solutions; once again we would like to think about which one might be best.

Let's consider finding exact solutions first. The most naive thing we could do is to just find the inverse of A and solve the equations as follows:

$$\mathrm{A}^{-1}\mathrm{A}\mathbf{x} = \mathrm{A}^{-1}\mathbf{b} \tag{11}$$

$$\mathrm{I}\mathbf{x} = \mathrm{A}^{-1}\mathbf{b} \tag{12}$$

$$\mathbf{x} = \mathrm{A}^{-1}\mathbf{b} \tag{13}$$

which is known as *Cramer's rule*.

There are several problems with this approach. Most importantly, many interesting functions are not invertible. In other words, given the output there might be several inputs which could have generated it or no inputs which could have. But beyond that, matrix inversion is a difficult and potentially numerically unstable operation.

In fact, there is a much better way to define what we want as a solution. We will say that we want a solution $\mathbf{x}^*$ which minimizes the error:

$$e = \|\mathrm{A}\mathbf{x}^* - \mathbf{b}\|^2 = \mathbf{x}^\top \mathrm{A}^\top \mathrm{A}\mathbf{x} - 2\mathbf{x}^\top \mathrm{A}^\top \mathbf{b} + \mathbf{b}^\top \mathbf{b} \tag{14}$$

This problem is known as *linear least squares*, for obvious reasons. If there is an exact solution (one giving zero error) it will certainly minimize the above cost, but if there is not, we can still find the best possible approximation. If we take the matrix derivative (see Chapter ??) of this expression, we can find the best soltiuon:

$$\mathbf{x}^* = (\mathrm{A}^\top \mathrm{A})^{-1}\mathrm{A}^\top \mathbf{b} \tag{15}$$

which takes advantage of the fact that even if A is not invertible, $\mathrm{A}^\top \mathrm{A}$ may be.

But what if the problem is degererate. In other words, what if is there more than one exact solution (say a family of them), or indeed more than one inexact solution which all achieve the same minimum error. How can this occur? Imagine an equation like this:

$$[1, -1]\mathbf{x} = 4 \tag{16}$$

in which $\mathrm{A} = [1, -1]$. This equation constrains the difference between the two elements of $\mathbf{x}$ to be 4 but the sum can be as large or small as we want. As you can read in the appendix, this happens because the matrix A has a *null space* and we can add any amount of any vector in the null space to $\mathbf{x}$ without affecting $\mathrm{A}\mathbf{x}$.

We can take things one step further to get around this problem also. The answer is to ask for the *minimum norm* vector $\mathbf{x}$ that still minimizes the above error. This breaks the degeneracies in both the exact and inexact cases and leaves us with solution vectors that have no projection into the null space of A. In terms of our cost function, this corresponds to adding an infinitesial penalty on $\mathbf{x}^\top \mathbf{x}$:

$$e = \lim_{\epsilon \to 0} [\mathbf{x}^\top \mathrm{A}^\top \mathrm{A}\mathbf{x} - 2\mathbf{x}^\top \mathrm{A}^\top \mathbf{b} + \mathbf{b}^\top \mathbf{b} + \epsilon \mathbf{x}^\top \mathbf{x}] \tag{17}$$

And the optimal solution becomes

$$\mathbf{x}^* = \lim_{\epsilon \to 0} [(\mathrm{A}^\top \mathrm{A} + \epsilon \mathrm{I})^{-1}\mathrm{A}^\top \mathbf{b}] \tag{18}$$

Now, of course actually computing these solutions efficiently and in a numerically stable way is the topic of much study in numerical methods. However, in MATLAB you don't have to worry about any of this, you can just type `xx=AA \ bb` and let someone else worry about it.

B. *Linear Regression: solving for* A

Now consider what happens if we have many vectors $\mathbf{x}_n$ and $\mathbf{b}_n$, all of which we want to satisfy the some equation $\mathrm{A}\mathbf{x}_n = \mathbf{b}_n$. If we stack the vectors $\mathbf{x}_n$ beside each other as the columns of a large matrix X and do the same for $\mathbf{b}_n$ to form B, we can write the problem as a large matrix equation:

$$\mathrm{AX} = \mathrm{B} \tag{19}$$

There are two things we could do here. If, as before, A is known, we could find X given B. (Once again finding B given X is trivial.) To do this we would just need to apply the techniques above to solve the system $\mathrm{A}\mathbf{x}_n = \mathbf{b}_n$ independently for each column n.

But there is something else we could do. If we were given *both* X and B we could try to find a *single* A which satisfied the equations. In essence we are fitting a linear function give its inputs X and corresponding outputs B. This problem is called *linear regression*. (Don't forget to add a column of ones to X if you want to fit an affine function, i.e.one with an offset.)

Once again, there are only very few cases in which there exists an A which exactly satisfies the equations. (If there is, X will be square and invertible.)

But we can set things up the same way as before and ask for the *least-squares* A which minimizes:

$$e = \sum_n \|\mathrm{A}\mathbf{x}_n - \mathbf{b}_n\|^2 \tag{20}$$

Once again, using matrix calculus we can derive the optimal solution to this problem. The answer, one of the most famous formulas in all of mathematics, is known as the *discrete Wiener filter*:

$$\mathrm{A}^* = \mathrm{BX}^\top(\mathrm{XX}^\top)^{-1} \tag{21}$$

Once again, we might have invertibility problems in $\mathrm{XX}^\top$; this corresponds to having fewer equations than unknowns in our linear system (or duplicated equations), thus leaving some of the elements of A unconstrained. We can get around this in the same way as with linear least squares by adding a small amount of penalty on the norm of the elements in A.

$$e = \sum_n \|\mathbf{b}_n - \mathrm{A}\mathbf{x}_n\|^2 + \epsilon\|\mathrm{A}\|^2 \tag{22}$$

Which means we are asking for the matrix of minimum norm which still minimizes the sum squared error on the outputs. Under this cost, the optimal solution is:

$$\mathrm{A}^* = \mathrm{BX}^\top(\mathrm{XX}^\top + \epsilon \mathrm{I})^{-1} \tag{23}$$

which is known as *ridge regression*. Often it is a good idea to use a small nonzero value of ϵ even if $\mathrm{XX}^\top$ is technically invertible, because this gives more stable solutions by penalizing large elements of A that aren't doing much to reduce the error. In neural networks, this is known as *weight decay*. You can also interpret it as having a Gaussian prior with mean zero and variance $1/2\epsilon$ on each element of A.

Once again, in MATLAB you don't have to worry about any of this, just type `AA = YY/ XX` and presto! linear regression. Notice that this is a forward slash, while least squares used a backslash. (Can you figure out how to do ridge regression this way, without using `inv()`?)

VI. Eigenvector Problems

under construction

VII. Singular Value Decomposition

under construction

Appendix: Fundamental Spaces

First of all remember that if A is m by n in our equation $\mathrm{A}\mathbf{x} = \mathbf{b}$ then $\mathbf{x}$ is an n-dimensional vector, i.e.the vectors we are loking for live in an n-dimensional space; similarly $\mathbf{b}$ is an m-dimensional vector. Left multiplying by the matrix A takes us from the $\mathbf{x}$ space ($\Re^n$) into the $\mathbf{b}$ space ($\Re^m$). Just by looking at its dimensions, you can tell that left multiplying by A^T would take us from the $\mathbf{b}$ space to the $\mathbf{x}$ space. Careful though, it is only very special matrices[1] that have the property $\mathrm{A}^T = \mathrm{A}^{-1}$ so that in general $\mathrm{A}^T\mathrm{A}\mathbf{x} \neq \mathbf{x}$. In other words, if we send a vector from $\Re^n$ to $\Re^m$ using A and then bring it back to $\Re^n$ using A^T we can't be sure that we have the original vector again.

So now we know what matrix multiplication does in terms of the *size* of its inputs and outputs. But we still need an understanding of what is actually going on. The answer is closely related to the idea of the *fundamental spaces* of a matrix A. Here is an informal summary of what happens, using the concept of the *rank* r of a matrix and these spaces. These terms are explained further below.

The action of an m by n matrix A of rank r is to take an input vector $\mathbf{x}$ (n-dimensional) to an output vector $\mathbf{b}$ (m-dimensional) through an r-dimensional "bottleneck". You can think of this as happening in two steps. First, A "crushes" part of $\mathbf{x}$ to bring it into an r-dimensional subspace of the input space $\Re^n$. Then it invertibly (one-to-one) maps the crushed $\mathbf{x}$ into an r-dimensional subspace of the output space $\Re^m$. The part of $\mathbf{x}$ that is "crushed"

[1] Called *orthogonal* or in the complex case *unitary* matrices.

is its projection into a space called the *null space* of A which is an $(n - r)$-dimensional subspace of the input space that you "cannot come from". The part of $\mathbf{x}$ that is "kept" is its projection into a space called the *row space* of A which is an r-dimensional subspace of the input space. The output subspace where all the $\mathbf{x}$'s end up is called the *column space* of A, also r-dimensional. You "cannot get to" anywhere outside the column space. If $r = n$ then no part of $\mathbf{x}$ is crushed and the row space fills the entire input space; i.e.you can "come from everywhere". If $r = m$ then the column space fills the entire output space; i.e.you can "get to everywhere". If $r = n = m$ then the entire input space is mapped one-to-one onto the entire output space and A is called an *invertible matrix*. Figure 1 (inspired by Strang) shows this graphically.

Basically, if you ask the matrix A, there are three classes of citizens in the input vector space $\Re^n$. There is the "unfortunate" class (of dimension $n - r$) who live purely in a place called the *null space* of A. All vectors from this class automatically get mapped onto the zero-vector in $\Re^m$. In otherwords, anyone who lives in the null space part of the input space gets "killed" by A's mapping. There is also the "lucky" class (of dimension r) who live purely in a place called *row space* of A. Any vector from this class gets mapped invertibly (one-to-one) into the *column space* in $\Re^m$. Finally, there is the "average" class who live in all the rest of the input space. Before telling you what happens to the average vectors, let me point out some surprising but true facts about the first two classes:

- The place where the "unfortunate" class lives, (i.e.the null space of A) is actually a *subspace*. This means that all linear combinations of vectors in the null space are still in the null space. No amount of cross-breeding amongst this class can ever produce anyone outside of it. Similarly, the place where the "lucky" class lives (the row space of A) is also a subspace and all linear combinations of vectors from the row space are confined to be still in the row space.
- The classes "unfortunate" and "lucky" are *orthogonal*, meaning that any vector in one class' subspace has *no projection* onto the other class' subspace. Members of the two classes have no attributes in common.
- The classes "unfortunate" and "lucky" *span* the entire input space, meaning that all other vectors in the input space (i.e.all the members of the class "average") can be written as linear combinations of vectors from the null space and the row space.

So what happens to a "average" vector under A's mapping? Well, first it gets *projected* into the row space and then mapped into the column space. This means that all of its null space components disappear and all of its row space components remain. In other words, A cleans it up by first removing any of its "unfortunate" attributes until it looks just like one of the "lucky" vectors. Then A maps this cleaned up version of "average" into the column space in $\Re^m$.

The number of *linearly independent* rows (or columns) of A is called the *rank* (denoted r above) and it is the dimension of the column space and also of the row space. The rank is of course no bigger than the smaller dimension of A. It is the dimension of the bottleneck through which vectors processed by A must pass.

The *column space* (or *range*) of A is the space spanned by its column vectors, or in other words, all the vectors that could ever be created as linear combinations of its columns. It is a subspace of the entire $\mathbf{b}$ space $\Re^m$. So when we form a product like $\mathrm{A}\mathbf{x}$, *no matter what we pick for* $\mathbf{x}$ we can only end up in a limited subspace of $\Re^m$ called the column space. The *row space* is a similar thing, except that it is the space spanned by the rows of A. It is of the same dimension as the column space but not necessarily the same space as the column space. When we form a product $\mathrm{A}\mathbf{x}$, *no matter what we pick for* $\mathbf{x}$ only the part of $\mathbf{x}$ that lives in row space determines what the answer is, the part of $\mathbf{x}$ that lives outside the rowspace (the null space component) is irrelevant because it gets projected out by the matrix.

It is clear that the zero vector is in every column space since we can combine any columns to get it by simply setting the coefficient of every column to zero, namely $\mathbf{x} = \mathbf{z}$. The smallest possible column space is produced by the zero matrix: its column space consists of only the zero vector. The largest possible column space is produced by a square matrix A with linearly independent columns; its column space is all of $\Re^n$ (where n is the size of A).

However, it may be possible to combine the columns of a matrix using some *nonzero* coefficients and still have them all cancel each other out to give zero; any such solutions for $\mathbf{x}$ are said to lie in the *null space* of the matrix A. That is, all solutions to $\mathrm{A}\mathbf{x} = \mathbf{z}$ except $\mathbf{x} = \mathbf{z}$ form the null space. The null space is the part of the input space that is orthogonal to the rowspace. Intuitively, any vectors that lie purely in the null space are "killed" (projected out) by A since they map to the zero vector. A completely complementary picture exists when we talk about the space $\Re^m$ and the matrix A^T. In particular, A^T has a row space (which is the column space of A) and a column space (which is the row space of A) and also a null space (which is curiously called the *left null space* of

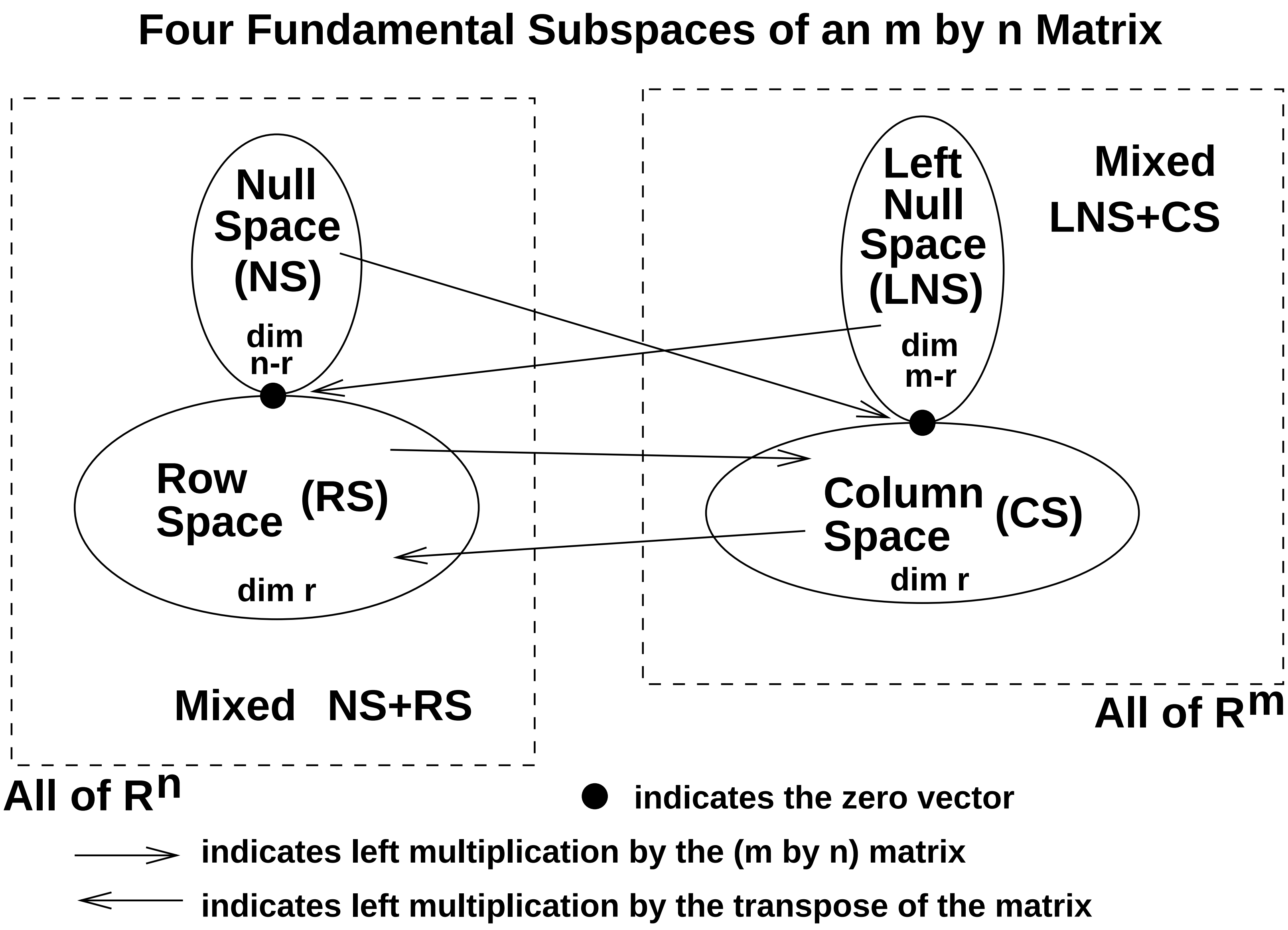

Fig. 1. The Four Fundamental Subspaces of a matrix

A).

So we have two pairs of orthogonal subspaces, one pair in $\Re^m$ which between them span $\Re^m$ and another pair in $\Re^n$ which between them span $\Re^n$. Now here is an important thing to know: *Any* matrix A maps its row space invertibly into its column space and A^T does the reverse. What does *invertibly* or *one-to-one* mean? Intuitively it means that no information is lost in the mapping. In particular, it means that each vector in the row space has exactly one corresponding vector in the column space and that no two row space vectors get mapped to the same column space vector. You can think of little strings connecting each row space vector to its column space "friend". Careful though, A^T may have a different (although still one-to-one) idea about who is friends with whom so it may not necessarily "follow the strings back from the column space to the row space", i.e.it may not be the inverse of A. If the strings all line up then $\mathrm{A}^T = \mathrm{A}^{-1}$ and we call A *orthogonal* or *unitary* in the complex case.

Invertibility

We saw above that *any* matrix maps its row space invertibly into its column space. Some special matrices map their entire input space invertibly into their entire output space. These are known as *invertible* or *full rank* or *non-singular* matrices. It is clear upon some reflection that such matrices have no null space since if they did then some non-zero input vectors would get mapped onto the zero vector and it would be impossible to recover them (making the mapping non-invertible). In other words, for such matrices, the row space fills the whole input space.

Formally, we say that a matrix A is *invertible* if there exists a matrix A^{-1} such that $\mathrm{AA}^{-1} = \mathrm{I}$. The matrix A^{-1} is called the *inverse* of A and is unique if it exists. The most common case is square, full rank matrices, for which the inverse can be found explicitly using many methods, for example Gauss-Jordan.[2] It is one of the astounding facts of computational algebra that such methods run in only $O(n^3)$ time which is the same as matrix multiplication.

REFERENCES

[1] Strang, *Linear Algebra and Applications*

[2]Write I and A side by side and do row ops on A to make it I