The Neural Network, its Techniques and Applications

Casey Schafer

April 12, 2016

Contents

1 Introduction **3**

2 Linear Algebra Terminology and Background **3**

2.1 Linear Independence and Spanning . . . 3
2.2 Basis . . . 4
2.3 Example: Too Many Columns . . . 4
2.4 Example: Too Many Rows . . . 5
2.5 Reconstructions . . . 5
2.6 Change of Basis Theorem . . . 6
2.7 Example: Changing a Basis . . . 7

3 Singular Value Decomposition **8**

3.1 The Symmetric Matrix . . . 9
3.2 Singular Value Decomposition . . . 10
3.3 Example: SVD in Action . . . 12

4 Applications of the SVD **13**

4.1 The Psuedoinverse . . . 13
4.2 The Covariance Matrix . . . 14
4.3 The Best Basis . . . 15
4.4 Example: Finding the Best Basis . . . 17
4.5 SV-Spectral Decomposition . . . 19
4.6 Practical Applications . . . 20

5 Neural Networks **23**

5.1 Example: Cancer . . . 24
5.2 Neural Network Theory . . . 25
5.3 Build up to Gradient Descent . . . 28
5.4 Gradient Descent . . . 29
5.5 Example: Cancer Revisited . . . 31
5.6 Example: Function Fitting . . . 34

6 Conclusion **36**

Alphabetical Index **37**

1 Introduction

A **neural network** is a powerful mathematical model combining linear algebra, biology and statistics to solve a problem in a unique way. The network takes a given amount of inputs and then calculates a specified number of outputs aimed at targeting the actual result. Problems such as pattern recognition, linear classification, data fitting and more can all be conquered with a neural network. This paper will aim to answer these questions: what is a neural network, what is its mathematical background, and how can we use a neural network in application?

My paper will be split roughly into two parts. The first entails a host of linear algebra including sections on bases, symmetric matrices, the singular value decomposition, the covariance matrix and the psuedoinverse. All these concepts tie together to form an amazingly simple way to solve a pertinent problem. The second part will consist of extensive talk on neural networks, an often more powerful way to solve the same problem. In the end we will see the benefits and drawbacks of both methods and realize that it is important to know everything in this paper so as to tackle any problem thrown at us. The paper will start with many key concepts from linear algebra. A typical linear algebra student will have seen some, but not all, of these ideas.

2 Linear Algebra Terminology and Background

Although I cannot go over all of linear algebra in this paper, I would like to go over a few concepts that are imperative to understand. Linear algebra is essentially the study of matrices. Throughout this paper I will refer to a matrix, A, of size $m \times n$, which is interpreted as having m rows and n columns.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

We would say this matrix maps a vector $\mathbf{x}$ in $\mathbb{R}^n$ to a vector $A\mathbf{x}$ in $\mathbb{R}^m$. A column vector, normally just called a **vector**, is simply a matrix of size $m \times 1$. We would normally say an $m \times n$ matrix is comprised of n different vectors, which we would denote $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$. Two notable concepts related to the columns of a matrix are linear independence and spanning.

2.1 Linear Independence and Spanning

If a column of a matrix can be written as the sum of scalar multiples of other columns then we say the columns of this matrix are linearly dependent. Formally, a set of columns are said to be linearly dependent if the equation,

$$0 = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \ldots + c_n\mathbf{v}_n,$$

has a solution where not all constants, c_i, are equal to 0. Note that if we put one of the terms, $c_i\mathbf{v}_i$, on the left side and divide by $-c_i$, assuming $c_i \neq 0$, we are saying that we can write the vector $\mathbf{v}_i$ as the sum of constant multiples of the other vectors. If all c_i's are equal to 0, we cannot divide by any c_i and thus cannot write one vector as the sum of constant multiples of the others. When this happens, we call the set of vectors **linearly independent** and formally say the equation

$$0 = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \ldots + c_n\mathbf{v}_n,$$

has only the solution where all c_i's are equal to zero [1]. Linear independence is a useful property columns of a matrix can have. Essentially it is the most efficient matrix possible, as each vector contributes something new.

Another property is the **span** of the columns of a matrix. The span of a set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ is the set of all **linear combinations** of the vectors, or the collection of vectors of the form:

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \ldots + c_n\mathbf{v}_n[1].$$

While linear independence might be an indicator of efficiency, the span tells us how much total data our vectors can represent. An ideal matrix would have columns with a large span and linear independence, both explaining a lot of data and doing so without any unnecessary vectors. This brings us to the idea of a basis.

2.2 Basis

A **basis** of a subspace H of $\mathbb{R}^m$ is a linearly independent set of vectors in H that spans H [1]. Essentially this means that any vector in H can be written as a linear combination of the vectors from the basis and that it is the smallest such set. Let's look at a few examples to understand the concepts of linear independence, spanning and a basis.

2.3 Example: Too Many Columns

Let $A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$. We could also think of this matrix as a collection of three column vectors $\mathbf{a}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\mathbf{a}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, and $\mathbf{a}_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

Clearly we can write any point in $\mathbb{R}^2$ using the first two vectors as such:

$$\begin{bmatrix} x \\ y \end{bmatrix} = c_1\mathbf{a}_1 + c_2\mathbf{a}_2.$$

Therefore this matrix spans $\mathbb{R}^2$. However notice we can write $\mathbf{a}_3 = \mathbf{a}_1 + \mathbf{a}_2$. The third vector can be written as a combination of the first two and so we have linear dependence. This matrix, while spanning $\mathbb{R}^2$ is not a basis of $\mathbb{R}^2$. If we took only the first two columns where $\mathbf{a}_1$ and $\mathbf{a}_2$ are in fact linearly independent we would have a basis for $\mathbb{R}^2$.

2.4 Example: Too Many Rows

Here's another matrix, $B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$. Notice it is the **transpose** of A, denoted A^T, meaning the rows and columns are swapped. To test linear independence,we could try writing one vector as a scalar multiple of the other. But notice the presence of 0 in one of the columns and the absence of 0 in the other column for the same row. Thus there is no way to do this, so these columns are linearly independent. What this matrix lacks is a spanning of $\mathbb{R}^3$. Last time we wanted a span of $\mathbb{R}^2$ because there were 2 rows. Now since there are three rows we want a span of $\mathbb{R}^3$. However try writing a point like $(2, 3, 4)$ as a combination of the two vectors. The 2 in the first row and the fact that $\mathbf{v}_1$ has a 0 in the first row would require $\mathbf{v}_2$'s constant to be 2. Similarly the second row indicates $\mathbf{v}_1$'s constant is 3. With those already set, we are unable to get 4 in the last row (but we could get 5). We found a point in $\mathbb{R}^3$ that couldn't be expressed as a linear combination of our vectors, therefore B does not span $\mathbb{R}^3$ and again there is no basis for $\mathbb{R}^3$. However we do have a basis for a subspace similar to $\mathbb{R}^2$. We cannot write a point from $\mathbb{R}^2$ given as $(x, y, 0)$ as a collection of these vectors. However there is a plane in $\mathbb{R}^3$ that contains points of the form $(x, y, x+y)$ and these two vectors would suffice to explain any point on this plane. This space is in fact isomorphic to $\mathbb{R}^2$, a useful fact, but for the non-mathematical readers we will just say that this space exhibits properties very similar but not identical to $\mathbb{R}^2$.

In these two examples something was slightly off. In the first we had an additional vector that prevented us from a basis. In the second we could not form a basis for all of $\mathbb{R}^3$. An astute mind might be able to guess that the reasons for these problems was because we had too many columns in the first example and too many rows in the second. This observation led to part of the invertible matrix theorem that says: if the columns of a square matrix (a matrix with the same number of rows and columns, n) are linearly independent and span $\mathbb{R}^n$ then those columns form a basis for $\mathbb{R}^n$. This is not directly tied to my project, but it is a nice theorem to know regardless. What is pertinent to this project is the takeaway that square matrices are nicer to work with than their non square counterparts. When a matrix is not square we usually have to sacrifice something, either linear independence as in the first example or the ability to span $\mathbb{R}^m$ as in the second example. And because matrices in the real world are often not square, these sacrifices are often a necessity.

2.5 Reconstructions

When we have a basis of a space, there are specific vectors that determine the entire space, call these $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$. These are known as the **basis vectors** for a space. If we want to write any point $\mathbf{x}$ in our space as a combination of these vectors, we would use a set of **coordinates**, $\{c_1, c_2, \ldots, c_n\}$, and write:

$$\mathbf{x} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \ldots + c_n\mathbf{v}_n$$

Whether you know it or not, you have seen this before. Think about $\mathbb{R}^3$. Any point can be expressed as a linear combination of the three basis vectors, $\begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$, $\begin{bmatrix}0 & 1 & 0\end{bmatrix}^T$, and $\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$. Similarly the basis vectors for $\mathbb{R}^2$ are $\begin{bmatrix}1 & 0\end{bmatrix}^T$ and $\begin{bmatrix}0 & 1\end{bmatrix}^T$. However often we are able to write a point typically found in one coordinate system using a different coordinate system with different basis vectors. The point $(2, 3, 5)$ could be written as such in $\mathbb{R}^3$, or could be written using the coordinate system defined by the basis vectors in the previous example, $\begin{bmatrix}0\\1\\1\end{bmatrix}$, $\begin{bmatrix}1\\0\\1\end{bmatrix}$, as $2\mathbf{v}_1 + 3\mathbf{v}_2$. The coordinates using these new basis vectors would now be $(2, 3)$. What is the point of this? Notice we can now express a point in $\mathbb{R}^3$ using only 2 coordinates instead of 3. That might not seem like a big deal, but as our matrices get bigger the opportunity to express the same point using less values will be crucial. The down side? We can only explain a select few points in $\mathbb{R}^3$; $(2, 3, 4)$ for instance will never be able to be fully explained with this coordinate system.

This is a fundamental point in my project: we want to explain as much data as possible but we don't want to use a ridiculously sized coordinate system to do so. Often we will 'go down' in coordinate systems to express the data using less points, losing a little of what we can explain in the process. This is wrapped up nicely using the Change of Basis Theorem.

2.6 Change of Basis Theorem

First we need a few more definitions. Look at the basis vectors for $\mathbb{R}^2$ that we used previously. These vectors have two special properties. The first is that each vector is a unit, or has length 1. The **length**, or **norm**, of a vector is given using the distance formula, the square root of sum of squares of each term. When a vector's length is 1, it is a **unit vector** like the basis vectors in $\mathbb{R}^2$, $\mathbb{R}^3$ and beyond. Luckily any vector can be made into a unit vector by dividing each term by the vector's length. The basis vectors for $\mathbb{R}^2$ are also **orthogonal**, that is perpendicular, which happens when the dot product of the two is equal to 0. The dot product is just the sum of the product of each term in a vector with its corresponding term in the other vector.

Putting these two concepts together, a set of vectors form an **orthonormal set** if each vector is a unit vector and is orthogonal to every other vector in the set. The basis vectors for $\mathbb{R}^2$ would then be orthonormal basis vectors.

This leads us to the Change of Basis Theorem. This theorem is hugely important to this project. Essentially it allows, given certain circumstances, a vector normally in one basis to be written in another, often using fewer points.

After the theorem we will see an example that demonstrates the benefits of the Change of Basis Theorem.

Change of Basis Theorem. *Let U be an $m \times n$ matrix with $m \geq n$ that has n orthonormal basis vectors for H, a subspace of $\mathbb{R}^m$. Then for $\boldsymbol{x} \in H$, the* ***low dimensional representation of $\boldsymbol{x}$*** *is the coordinate vector $\boldsymbol{c} \in \mathbb{R}^n$ where*

$$\boldsymbol{c} = U^T \boldsymbol{x}$$

The ***reconstruction*** *of x given the coordinates is*

$$\boldsymbol{x} = U\boldsymbol{c}$$

Another way we could write the latter formula is $\mathbf{x} = UU^T\mathbf{x}$. It might not seem useful to write $\mathbf{x}$ as a function of itself, but the matrix UU^T projects $\mathbf{x}$ into the **column space** of U if it does not already exist there. The column space of a matrix is simply the set of all linear combinations of its columns [1]. While it might not be clear what this means, a couple examples will hopefully illuminate the use of this formula and in particular the matrix UU^T.

2.7 Example: Changing a Basis

Let $\mathbf{x}_1 = \begin{bmatrix} 3 \\ 2 \\ 3 \end{bmatrix}$ and $U = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 \\ \frac{1}{\sqrt{2}} & 0 \end{bmatrix}$

Let's make sure this fulfills the requirements of the Change of Basis Theorem. U is a 3×2 matrix, so $m = 3$ and $n = 2$ and indeed $m \geq n$. We could separate U into vectors, $\mathbf{u}_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix}$ and $\mathbf{u}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$. The first vector in U has length $d = \sqrt{(\frac{1}{\sqrt{2}})^2 + (\frac{1}{\sqrt{2}})^2} = 1$ and the second clearly does too, so they are both unit vectors. Their dot product is $\mathbf{u}_1 \cdot \mathbf{u}_2 = (\frac{1}{\sqrt{2}})(0) + (0)(1) + (\frac{1}{\sqrt{2}})(0) = 0$, so they are orthogonal. Because of this and the fact that they are basis vectors for a subspace H of $\mathbb{R}^3$, then the columns of U are $n = 2$ orthonormal basis vectors for H. The vector $\mathbf{x}_1$ is also in this subspace, although that may not be obvious yet.

With this out of the way, we can safely use the Change of Basis Theorem to find a low dimensional representation of $\mathbf{x}_1$ in the coordinate system given by the columns of U. To find the coordinates of $\mathbf{x}_1$ in a space isomorphic to $\mathbb{R}^2$, we just need to compute $U^T\mathbf{x}_1$.

$$\mathbf{c} = U^T\mathbf{x}_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{6}{\sqrt{2}} & 2 \end{bmatrix}$$

Thus the coordinates of $\mathbf{x}_1$ in the basis given by $\mathbf{u}_1$ and $\mathbf{u}_2$ are $(\frac{6}{\sqrt{2}}, 2)$. We can write $\mathbf{x}_1 = \frac{6}{\sqrt{2}}\mathbf{u}_1 + 2\mathbf{u}_2$. As mentioned previously, the reason for finding a lower dimensional representation such as this is to express $\mathbf{x}_1$ using less points, which we did. To confirm that this is the same $\mathbf{x}_1$ we had before, we can find the reconstruction of $\mathbf{x}_1$ in our original basis (the basis vectors of $\mathbb{R}^3$) by computing $U\mathbf{c}$:

$$\mathbf{x}_1 = U\mathbf{c} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 \\ \frac{1}{\sqrt{2}} & 0 \end{bmatrix} \begin{bmatrix} \frac{6}{\sqrt{2}} & 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 3 \end{bmatrix}$$

The reconstruction of $\mathbf{x}_1$ is the same as the $\mathbf{x}_1$ we started with, as expected. This example was meant to show how a vector can be expressed as the combination of basis vectors and how, in this case, no data was lost switching between $\mathbb{R}^3$ and a subspace of $\mathbb{R}^3$ isomorphic to $\mathbb{R}^2$ [2]. But remember that the downside is that we lose some data when we go down in the size of our basis. What does that mean?

Well take a different point, say $\mathbf{x}_2 = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T$. Using the same method we calculate $\mathbf{c} = \begin{bmatrix} \frac{4}{\sqrt{2}} & 2 \end{bmatrix}$. However when we go back to reconstruct $\mathbf{x}_2$, we get $\mathbf{x}_2 = \begin{bmatrix} 2 & 2 & 2 \end{bmatrix}^T$. That's not right! When we went to a lower dimensional basis we lost some of the data. This is because $\mathbf{x}_2$ was not in the subspace H, spanned by the columns of U, a prerequisite for the Change of Basis Theorem. When this happens, we look back to the matrix UU^T and the column space of U. Notice $\mathbf{u}_1$ has the first and third elements the same, whereas $\mathbf{u}_2$ has 0's in those places. That means no matter what constant we put in front of $\mathbf{u}_1$, the set of linear combinations of $\mathbf{u}_1$ and $\mathbf{u}_2$, $c_1\mathbf{u}_1 + c_2\mathbf{u}_2$ (by definition the column space of U), will as well have the first and third elements the same. The only points that can be perfectly reconstructed are the ones in the column space of U, like $\mathbf{x}_1$, which have the first and third elements the same. All other points will be projected onto this space losing a little bit of data along the way. Notice that $\mathbf{x}_2$ started out with distinct first and third elements but ended up with identical elements in those spots during the reconstruction. As said before, this is because multiplying by the matrix UU^T projects a vector so that it is in the column space of U. Sometimes this is a necessary sacrifice to make. Usually if we have a large set of data we find it easier to work with in a lower dimensional basis. Often we will choose the size of the basis that we want to work with, perhaps saying 'I would like my data to be represented using k coordinates'. Then the next question is how do we find the best lower dimensional basis, that is the basis that loses the least amount of data, in a space isomorphic to $\mathbb{R}^k$. Well, that is what much of this paper is about after all.

3 Singular Value Decomposition

The path to the best basis is a long and arduous one. It starts with more linear algebra, except this time we will be exploring concepts that the typical linear algebra student has not seen yet. Singular value decomposition is at the forefront, but as always there are terms and concepts that need to be understood before we can talk about the SVD.

3.1 The Symmetric Matrix

Recall that a square matrix has an equal number of rows and columns. When a square matrix is invertible, it has several nice properties given by the Invertible Matrix Theorem. A special kind of square matrix is a **symmetric matrix** , a square matrix, A, such that $A = A^T$. Symmetric matrices are invertible square matrices and so have all the properties from the Invertible Matrix Theorem as well as many more. These are given by the very handy Spectral Theorem, which will appear shortly. First there are a few definitions to go over.

Eigenvalues and **eigenvectors** are important terms. An eigenvalue, λ, is a constant value with a corresponding eigenvector, $\mathbf{v}$, such that $A\mathbf{v} = \lambda\mathbf{v}$ for some matrix A [2]. Notice that for an eigenvector of size $n \times 1$, the matrix A must necessarily be square of size $n \times n$. The **eigenspace** would be the span of the eigenvectors. We also say that a matrix is **diagonalizable** if it can be written as $A = PDP^{-1}$, where P is a square matrix composed of the eigenvectors of A and D is a diagonal matrix with the corresponding eigenvalues on the main diagonal. Diagonal implies that there are 0's in all other spots other than the main diagonal. A matrix is **orthogonally diagonalizable** if $P^{-1} = P^T$.

We are now ready for the Spectral Theorem. This theorem provides many useful properties that any symmetric matrix will have.

The Spectral Theorem. *An $n \times n$ symmetric matrix A has the following properties:*

a) A has n real eigenvalues, counting multiplicities.

b) The dimension of the eigenspace for each eigenvalue λ equals the multiplicity of λ as a root of the characteristic equation.

c) The eigenspaces are mutually orthogonal, in the sense that eigenvectors corresponding to different eigenvalues are orthogonal.

d) A is orthogonally diagonalizable [1].

An $n \times n$ invertible matrix need not have n real eigenvalues, so part *a*) is useful to know. Part *c*) gives us that the eigenvectors of a symmetric matrix are orthogonal with each other, therefore if we also make them unit vectors we could have an orthonormal basis, a prerequisite for the Change of Basis Theorem. Part *d*), is also quite nice, as we can now write any symmetric matrix A as PDP^T as in the definition of an orthogonally diagonalizable matrix.

The Spectral Theorem also leads to a nice property known as spectral decomposition. From part *d*) we know any symmetric matrix, A, can be written as PDP^T. Since we know there are n real eigenvalues and corresponding eigenvectors from part *a*), we let the eigenvectors of A be $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$ and the eigenvalues of A be $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. Then:

$$A = U\Lambda U^T = \begin{bmatrix}\mathbf{u}_1 & \dots & \mathbf{u}_n\end{bmatrix} \begin{bmatrix}\lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n\end{bmatrix} \begin{bmatrix}\mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_n^T\end{bmatrix}.$$

Noting that the front product is the same as multiplying each eigenvector in P by its corresponding eigenvalue we get:

$$= \begin{bmatrix}\lambda_1\mathbf{u}_1 & \dots & \lambda_n\mathbf{u}_n\end{bmatrix} \begin{bmatrix}\mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_n^T\end{bmatrix}.$$

Then from a theorem known as the column-row expansion theorem we can write A as follows:

$$A = \lambda_1\mathbf{u}_1\mathbf{u}_1^T + \lambda_2\mathbf{u}_2\mathbf{u}_2^T + \dots + \lambda_n\mathbf{u}_n\mathbf{u}_n^T.$$

This is called the **spectral decomposition** of a matrix A [1]. It can often be useful to write out a matrix using only its eigenvalues and eigenvectors. This is only possible for a symmetric matrix, as a matrix is symmetric if it is orthogonally diagonalizable (the converse of part d, resulting in double implication).

The Spectral Theorem and spectral decomposition are excellent properties for symmetric matrices, but rarely is anything in the real world so nicely laid out for us. Often times the matrices we use are not even square, much less symmetric. What we need is to generalize the concepts of eigenvalues, eigenvectors, diagonalization and spectral decomposition to matrices of any size. Enter singular value decomposition ...

3.2 Singular Value Decomposition

Singular value decomposition is a process similar to diagonalizability that can be used on any matrix, regardless of its size. Start with an $m \times n$ matrix, A, where $m \neq n$. Note that A^TA is square and size $n \times n$. Furthermore it is symmetric since $(A^TA)^T = A^TA^{TT} = A^TA$. Similarly AA^T is symmetric and size $m \times m$ since $(AA^T)^T = A^{TT}A^T = AA^T$. The matrix A^TA is then an $n \times n$ square matrix with eigenvalues $\{\lambda_1, ...\lambda_n\}$ and eigenvectors $\{\mathbf{v}_1, ...\mathbf{v}_n\}$ guaranteed by the Spectral Theorem and AA^T has eigenvalues $\{\zeta_1, ...\zeta_m\}$ and eigenvectors $\{\mathbf{u}_1, ...\mathbf{u}_m\}$. Also because of the Spectral Theorem, both these matrices can be diagonalized, so $A^TA = V\Lambda V^T$ and $AA^T = UZU^T$.

First let us note that $A^TA\mathbf{v}_i = \lambda_i\mathbf{v}_i$, since $\mathbf{v}_i$ and λ_i are an eigenvector and eigenvalue respectively for A^TA. Front multiplying by A leaves $(AA^T)A\mathbf{v}_i = \lambda_iA\mathbf{v}_i$. Thus $A\mathbf{v}_i$ is an eigenvector for AA^T with the same nonzero eigenvalues as A^TA, λ_i. We called those eigenvectors $\mathbf{u}_i$, so now we have the relationship $A\mathbf{v}_i = \mathbf{u}_i$. Almost. If we assume $\mathbf{v}_i$ and $\mathbf{u}_i$ are unit vectors then we must divide by the magnitude $||A\mathbf{v}_i||$ to maintain the unit vector property.

Let's note something interesting about $||A\mathbf{v}_i||^2$:

$$||A\mathbf{v}_i||^2 = (A\mathbf{v}_i)^T A\mathbf{v}_i = \mathbf{v}_i^T A^T A\mathbf{v}_i.$$

The first equality is how we represent squares in matrix form and the second equality is how multiple terms transposed is evaluated. Noting that since $\mathbf{v}_i$ is an eigenvector of A^TA, then $A^TA\mathbf{v}_i = \lambda_i\mathbf{v}_i$ by definition of eigenvectors and eigenvalues. Thus we are left with: $||A\mathbf{v}_i||^2 = \mathbf{v}_i^T\lambda_i\mathbf{v}_i$. The constant eigenvalue can be pulled to the front leaving $\mathbf{v}_i^T\mathbf{v}_i = ||\mathbf{v}_i||^2$, the reverse of what we did to $A\mathbf{v}_i$ initially. But since $\mathbf{v}_i$ is a unit, $||\mathbf{v}_i||^2 = 1$. We are left with $||A\mathbf{v}_i||^2 = \lambda_i$ or

$$||Av_i|| = \sqrt{\lambda_i}.$$

The **singular values**, denoted σ_i, of A are defined as the square roots of the eigenvalues of A^TA, $\sigma_i = \sqrt{\lambda_i}$, also equivalent to $||A\mathbf{v}_i||$. This means that $\mathbf{u}_i = \frac{A\mathbf{v}_i}{\sigma_i}$ and $\mathbf{u}_i$ is a unit vector as desired. The entire U matrix is then constructed using $\mathbf{u}_i$'s as the columns.

$$U = [\mathbf{u}_1, \ldots, \mathbf{u}_m].$$

V is done similarly using $\mathbf{v}_i$'s as the columns:

$$V = [\mathbf{v}_1, \ldots, \mathbf{v}_n].$$

Notice now that $AV = \begin{bmatrix} A\mathbf{v}_1 & \ldots & A\mathbf{v}_m \end{bmatrix} = \begin{bmatrix} \sigma_1\mathbf{u}_1 & \ldots & \sigma_m\mathbf{u}_m \end{bmatrix}$. If we introduce another matrix, Σ, with the singular values on the diagonal then suddenly we get $AV = U\Sigma$. Note that to make the matrix multiplication work we have to expand Σ to size $m \times n$ and so we fill in 0's elsewhere. Finally noting that V is orthogonal we get $VV^T = I$ (where I is the identity matrix). Then $AVV^T = A = U\Sigma V^T$. We have reached singular value decomposition.

Singular Value Decomposition. *Any matrix A can be decomposed as $U\Sigma V^T$ where:*

$$U = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m]$$

$$V = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]$$

$$\Sigma = \begin{bmatrix} D & 0 & \ldots \\ 0 & 0 & \ldots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

$$D = \begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_m \end{bmatrix}.$$

One important thing to note is that when there are not m nonzero eigenvalues, D will be a smaller size. Since Σ is constructed with 0's around D this does not matter in practice, but does raise a question of what we want to include as part of the SVD. Above is what is known as **full** singluar value decomposition, where we take all possible eigenvalues and eigenvectors. In this case Σ is the same size as A, $m \times n$, and U and V are square with all eigenvectors included. Sometimes we get eigenvalues of 0, which are not helpful and so we do not want to include them nor their eigenvectors. A **reduced** singular value decomposition, if there are r nonzero eigenvalues, would have U be $m \times r$, V be $n \times r$ and Σ be $r \times r$, getting rid of all the excess 0's. It should be noted in either case that r, the number of nonzero eigenvalues, will never exceed the minimum of n and m, as both U and V have the same nonzero eigenvalues despite their different sizes.

Singular value decomposition is an immensely powerful and useful tool. The main takeaway should be that any matrix can be singular value decomposed regardless of its size. We will see how techniques such as the covariance matrix and the psuedoinverse use the SVD to great effect. But before we start talking about its applications, let's go through an example to see how any given matrix would be constructed [1].

3.3 Example: SVD in Action

Let $A = \begin{bmatrix} 4 & 11 & 14 \\ 8 & 7 & -2 \end{bmatrix}$. The first thing to do is calculate A^TA and find its eigenvalues and eigenvectors. A^TA turns out to be $\begin{bmatrix} 80 & 100 & 40 \\ 100 & 170 & 140 \\ 40 & 140 & 200 \end{bmatrix}$ and the eigenvalues turn out to be $\lambda_1 = 360, \lambda_2 = 90$ and $\lambda_3 = 0$. The singular values are then $\sigma_1 = 6\sqrt{10}$, $\sigma_2 = 3\sqrt{10}$ and $\sigma_3 = 0$. The eigenvectors are then $\mathbf{v}_1 = \begin{bmatrix} 1/3 \\ 2/3 \\ 2/3 \end{bmatrix}$, $\mathbf{v}_2 = \begin{bmatrix} -2/3 \\ -1/3 \\ 2/3 \end{bmatrix}$ and $\mathbf{v}_3 = \begin{bmatrix} 2/3 \\ -2/3 \\ 1/3 \end{bmatrix}$. Notice these eigenvectors form an orthonormal set and their collection forms the matrix V^T:

$$V^T = \begin{bmatrix} 1/3 & 2/3 & 2/3 \\ -2/3 & -1/3 & 2/3 \\ 2/3 & -2/3 & 1/3 \end{bmatrix}.$$

Calculating D with this information is also quite trivial. We have two nonzero singular values so those go in our diagonal matrix, D:

$$D = \begin{bmatrix} 6\sqrt{10} & 0 \\ 0 & 3\sqrt{10} \end{bmatrix}$$

Σ is always the same size as our initial matrix in the full SVD case, so it would be:

$$\Sigma = \begin{bmatrix} 6\sqrt{10} & 0 & 0 \\ 0 & 3\sqrt{10} & 0 \end{bmatrix}$$

Finally U is comprised of $\frac{Av_i}{\sigma_1}$. So $u_1 = \frac{1}{6\sqrt{10}}\begin{bmatrix}18\\6\end{bmatrix}$ and $u_2 = \frac{1}{3\sqrt{10}}\begin{bmatrix}3\\-9\end{bmatrix}$.

$$U = \begin{bmatrix}3/\sqrt{10} & 1/\sqrt{10}\\ 1/\sqrt{10} & -3/\sqrt{10}\end{bmatrix}.$$

Putting all the pieces together we get:

$$A = U\Sigma V^T = \begin{bmatrix}3/\sqrt{10} & 1/\sqrt{10}\\ 1/\sqrt{10} & -3/\sqrt{10}\end{bmatrix}\begin{bmatrix}6\sqrt{10} & 0 & 0\\ 0 & 3\sqrt{10} & 0\end{bmatrix}\begin{bmatrix}1/3 & 2/3 & 2/3\\ -2/3 & -1/3 & 2/3\\ 2/3 & -2/3 & 1/3\end{bmatrix}.$$

Note that since the final singular value is 0, we leave it out as we do not want to divide by 0 when finding U. In general we only focus on the nonzero singular values, as those are the ones shared by AA^T and A^TA.

Now that the theory behind singular value decomposition has been explored, it is now time to move onto some of its applications. This starts with the psuedoinverse before jumping into the covariance matrix.

4 Applications of the SVD

4.1 The Psuedoinverse

A pertinent application of singular value decomposition to our talk on neural networks is the **psuedoinverse**. This involves finding a solution to the equation $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$. If A is an invertible matrix then we can easily say $\mathbf{x} = A^{-1}\mathbf{b}$, however this rarely happens. Just like singular value decomposition generalized the idea of diagonalization to matrices of any size, so too does the psuedoinverse generalize the idea of an inverse.

To see this, the equation $A\mathbf{x} = \mathbf{b}$ can be written using the SVD of A

$$U\Sigma V^T\mathbf{x} = \mathbf{b}.$$

It is important to note that in this case we are using the reduced SVD where Σ has all the nonzero singular values on the diagonal and no additional 0's. This allows Σ to have an inverse.

To go further, we must show $U^TU = I$. This comes straight from the fact that the columns of U form an orthonormal set by construction.

$$\begin{bmatrix}\mathbf{u}_1^T\\ \vdots\\ \mathbf{u}_r^T\end{bmatrix}\begin{bmatrix}\mathbf{u}_1 & \dots & \mathbf{u}_r\end{bmatrix}.$$

Notice that $\mathbf{u}_i^T\cdot\mathbf{u}_i = ||\mathbf{u}_i||^2 = 1$ because $\mathbf{u}_i$ is a unit vector. That 1 would go in the ith row, ith column spot, forming 1's on the diagonal. Also notice that $\mathbf{u}_i\cdot\mathbf{u}_j = 0$, because the columns are orthogonal. Therefore everywhere besides the diagonal is filled with 0's and we are left with the identity matrix. Note this would also work for V^TV.

With that being proved we can left multiply both sides by U^T to get:

$$\Sigma V^T\mathbf{x} = U^T\mathbf{b}.$$

Σ has all nonzero eigenvalues of A on the diagonal and so has an inverse, which we can multiply both sides by to get:

$$V^T\mathbf{x} = \Sigma^{-1}U^T\mathbf{b}.$$

Lastly, since $VV^T \neq I$, we cannot simply do as we did with U and have a clean solution for $\mathbf{x}$. However, remember that multiplying a vector by a matrix of the form VV^T is the same as projecting that vector onto the column space of V. Therefore if we project $\mathbf{x}$ onto the column space of V, we get:

$$\hat{\mathbf{x}} = V\Sigma^{-1}U^T\mathbf{b}.$$

The pseudoinverse is the term $V\Sigma^{-1}U^T$. We throw a hat on $\mathbf{x}$ to show that it is just a projection of $\mathbf{x}$, and not the actual value we were looking for. This is a case where we *had* to use a lower dimensional basis representation to get a close estimation of $\mathbf{x}$ in terms of V's column space. The problem with this is that we do not know how good the column space of V is at explaining the data in $\mathbf{x}$. It could be when representing $\mathbf{x}$ in this new basis that we lose a lot of information. What we really want is the basis that loses the least amount of information about $\mathbf{x}$, and a little look at the covariance matrix shows us that what we are looking for is indeed the matrix V used in the psuedoinverse.

4.2 The Covariance Matrix

Now we come to the **covariance matrix**, a hugely important topic involving singular value decomposition. For an $m \times n$ matrix, A, the covariance matrix, C, can be written as:

$$C = \frac{1}{n-1}AA^T.$$

The $\frac{1}{n-1}$ term comes from the definition of a sample covariance when data is sampled from a population and is used to give an unbiased estimation of the population covariance. However this is not extremely important to this paper, so we can think of it as a row wise average. This formula assumes that the matrix is **mean subtracted**, that is the sum across any row i is equal to 0:

$$\frac{1}{n}\sum_{k=1}^{n}\mathbf{a}_{i,k} = 0.$$

Unsurprisingly, to make sure a matrix is mean subtracted we calculate the mean of a row and literally subtract the mean from each value in the row.

The covariance matrix measures how correlated two rows in A are. Since A has m row vectors, there are m^2 combinations of two vectors to test correlation between, including testing a vector against itself. This gives an intuitive reason for why C is an $m \times m$ matrix: the value in $c_{i,j}$ measures the correlation between the ith and jth row vector in A. Since this correlation will be independent of which vector comes first, this also gives an intuitive reason for why C is symmetric, which it clearly is because AA^T is symmetric as shown previously. The covariance matrix can also be written as:

$$C = \frac{1}{n-1} A^T A,$$

to measure the correlation between column vectors. These formulas can be used more or less interchangeably with the same properties for each. Depending on the situation, it can be beneficial to use one equation or the other but in this paper we will use the former.

Plugging the singular value decomposition of A into the original equation for C results in:

$$C = U(\frac{1}{n-1}\Sigma^2)U^T.$$

A few things to note: since C is symmetric it can be written as PDP^T by the Spectral Theorem, where P is orthonormal and D is diagonal with the eigenvalues of C. The matrix U is orthonormal and $\frac{1}{n-1}\Sigma^2$ is diagonal, and so its diagonal consists of the eigenvalues of C. Since Σ is calculated by the singular value decomposition of A, an important realization occurs: we can calculate the eigenvalues and eigenvectors of a covariance matrix using the singular value decomposition of its original matrix. In fact if we let λ_i be an eigenvalue of the covariance matrix and remember σ_i is the singular value of A, the relationship is quite simple:

$$\lambda_i = \frac{1}{n-1}\sigma_i^2.$$

Since the singular values are in fact the square root of the eigenvalues, the relationship is even simpler: **the eigenvalues of the covariance matrix are the same as the eigenvalues of its original matrix** with an additional $\frac{1}{n-1}$ term added in [2].

We now have a handy way to calculate the eigenvalues of a covariance matrix, which can often be helpful to do especially when m is much larger than n. The next question is, why do we care about the covariance matrix and its eigenvalues? The answer has to do with the search for a best basis.

4.3 The Best Basis

Continuing with our talk on the covariance matrix, our goal in this section is to find the best projection onto a one dimensional subspace. Suppose A is a set of

n vectors in $\mathbb{R}^m$ and the data is mean subtracted. If we project the data onto any unit vector, $\mathbf{u}$, then the variance, $S^2_{\mathbf{u}}$, is given as:

$$S^2_{\mathbf{u}} = \mathbf{u}^T C \mathbf{u},$$

where C is of course the covariance matrix. When $\mathbf{u}$ is the first eigenvector of C, $\mathbf{v}_1$, then $\mathbf{v}_1^T C \mathbf{v}_1 = \mathbf{v}_1^T \lambda_1 \mathbf{v}_1$ by the definition of eigenvectors and eigenvalues. This in turn equals λ_1, because $\mathbf{v}_1$ is just a unit, so $\mathbf{v}_1^T \mathbf{v}_1 = 1$. The conclusion here is that the variance of data projected to an eigenvector is the corresponding eiegenvalue of C.

Next we want to estimate a vector $\mathbf{x}$ using parts of an orthonormal basis, $\{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$. First we can write the equation:

$$\mathbf{x} = c_1 \mathbf{u}_1 + \ldots + c_n \mathbf{u}_n.$$

The constant vector, $\mathbf{c}$ can then be written as $U^T \mathbf{x}$ using the change of basis theorem to get:

$$\mathbf{x} = \mathbf{u}_1^T \mathbf{x} \mathbf{u}_1 + \ldots + \mathbf{u}_n^T \mathbf{x} \mathbf{u}_n.$$

We can dot both sides with $\mathbf{x}$ to find the magnitude, noting that since we have an orthonormal basis, all terms combining two different vectors of U will be 0. We are left with:

$$||\mathbf{x}||^2 = (\mathbf{u}_1^T \mathbf{x})(\mathbf{x}^T \mathbf{u}_1) + \ldots + (\mathbf{u}_n^T \mathbf{x})(\mathbf{x}^T \mathbf{u}_n).$$

The parentheses are not relevant, but do show how we got this mess from the simple equation $\mathbf{a}^2 = \mathbf{a}\mathbf{a}^T$.

We can choose to drop all but one term and add an error in place. The reason we do this is because our goal from the start of this section was to find any given point using just one basis vector. We are left with:

$$||\mathbf{x}||^2 = \mathbf{u}_i^T \mathbf{x}\mathbf{x}^T \mathbf{u}_i + ||\mathbf{x}_{err}||^2.$$

Noting that the covariance matrix is equal to $\frac{1}{n-1}\mathbf{x}\mathbf{x}^T$ all we have to do is divide everything by $n-1$ to get the following equivalent equation:

$$\frac{1}{n-1}||\mathbf{x}||^2 = \mathbf{u}_i^T C \mathbf{u}_i + \frac{1}{n-1}||\mathbf{x}_{err}||^2,$$

where C is the covariance matrix for $\mathbf{x}$.

Of the three terms (one on the left of the equal sign, two on the right), the middle term should look familiar. It's the variance of $\mathbf{x}$ projected onto a unit vector, which $\mathbf{u}_1$ clearly is coming from the orthonormal set. Finding the best projection obviously involves minimizing the error term. However since $\mathbf{x}$ is given, the left most term is a constant whereas the other two terms depend on which vector, $\mathbf{u}_i$, is picked. Therefore minimizing the error is equivalent to maximizing the variance. Let's do that.

Since C is symmetric, its eigenvectors form an orthonormal basis for $\mathbb{R}^n$ from the Spectral Theorem and we can write any vector as a combination of these vectors using the Change of Basis Theorem. Therefore $\mathbf{u} = V\mathbf{c}$ where $\mathbf{c}$ is just a vector of constants and V is a matrix with the eigenvectors of C in the columns. Furthermore $C = VDV^T$ from the Spectral Theorem. Then we get:

$$\mathbf{u}^T C\mathbf{u} = (V\mathbf{c})^T(VDV^T)(V\mathbf{c}) = \mathbf{c}^T(V^TV)D(V^TV)\mathbf{c} = \mathbf{c}^T D\mathbf{c}.$$

Then noting that D is just a diagonal matrix with the eigenvalues of C we get:

$$= c_1^2\lambda_1 + c_2^2\lambda_2 + \ldots + c_n^2\lambda_n.$$

We also want this whole term to be a unit vector, so we can say that this quantity is equal to $p_1\lambda_1 + p_2\lambda_2 + \ldots + p_n\lambda_n$ where $\sum p_i = 1$. This is because the whole term is a unit vector, and all p_i are nonnegative because all the c_i's are squared. Interestingly this turns out to be like a probability function. From here it is pretty easy to see that maximizing this function is solved by finding the biggest eigenvalue, λ_j, and letting $p_j = 1$ while all other $p_i = 0$. If we note that the eigenvalues are listed in decreasing order in D by construction of the covariance matrix, then we know λ_1 is the biggest eigenvalue.

What did we do this all for? We were trying to find the vector that maximized the variance term to find the best 1 dimensional basis for $\mathbb{R}^n$. This best basis turned out to be created by the eigenvector corresponding to the first (and largest) eigenvalue of the covariance matrix, and generally **the best k-dimensional basis is found by taking the first k eigenvectors of the covariance matrix** [2].

This fact, coupled with our earlier discovery on the covariance matrix, is hugely important. Not only do we know the best k-dimensional basis comes from the first k eigenvalues of C, an extremely helpful fact for representing data using far fewer points, but also the eigenvalues of C come straight from the singular value decomposition of the original matrix.

So far this paper has looked at the Change of Basis Theorem, singular value decomposition, the psuedoinverse and the covariance matrix as main points of interest. How does they all tie together? Well the V term from the SVD allows the psuedoinverse to project a matrix X onto the column space of V using the Change of Basis Theorem. The matrix V consists of the eigenvectors of AA^T, which we said could be used to calculate the eigenvectors of the covariance matrix. We also said when changing to a different basis that the first k eigenvectors of the covariance matrix were the best basis to change to in order to minimize the error lost. Therefore the psuedoinverse is more powerful than we thought. Since V is the best basis, the projection of X is actually the best we can get. We will use the psuedoinverse later to solve the same problem a neural network would be used for, but in a different way. Because of this, it is nice to know the psuedoinverse is mathematically the best we can do.

4.4 Example: Finding the Best Basis

Before heading into bigger examples it is beneficial to understand how these concepts work. Suppose we have a matrix:

$$A = \begin{bmatrix} 4 & 1 & 1 \\ 3 & 5 & 1 \\ 4 & 2 & 6 \\ 1 & 1 & 1 \\ 3 & 3 & 6 \end{bmatrix}.$$

The covariance matrix assumes that A is mean subtracted, so we can calculate the mean row-wise and subtract it from each term in the row. Now we get:

$$A = \begin{bmatrix} 2 & -1 & -1 \\ 0 & 2 & -2 \\ 0 & -2 & 2 \\ 0 & 0 & 0 \\ -1 & -1 & 2 \end{bmatrix}.$$

Here we have 3 vectors in $\mathbb{R}^5$. Let's say we want to find a 1-dimensional representation of our data. Firstly, of course, we must find the eigenvalues of the covariance matrix. We could do that by calculating the covariance matrix,

$$C = \begin{bmatrix} 3 & 0 & 0 & 0 & -3/2 \\ 0 & 4 & -4 & 0 & -3 \\ 0 & -4 & 4 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ -3/2 & -3 & 3 & 0 & 3 \end{bmatrix},$$

and then solving for its eigenvalues. But notice that involves dealing with a 5×5 matrix. Instead we could calculate the singular values of A by first finding $A^T A$:

$$A^T A = \begin{bmatrix} 5 & -1 & -4 \\ -1 & 10 & -9 \\ -4 & -9 & 13 \end{bmatrix}.$$

The benefit is that we now only have to calculate the eigenvalues for a 3×3 matrix. It should be clear how useful this becomes when the number of rows greatly exceeds the number of columns.

The eigenvalues for $A^T A$ are $21, 7$ and 0. Now all we must do is divide by $n - 1 = 2$ to find the eigenvalues for C. These are $\frac{21}{2}, \frac{7}{2}$ and 0.

The best basis is then constructed by the eigenvector associated with the largest eigenvalue, in this case:

$$\mathbf{v}_1 = \sqrt{\frac{21}{2}} \begin{bmatrix} -1/2 \\ -2 \\ 5/2 \end{bmatrix},$$

after normalizing the vector. However this is the eigenvector of V in the singular value decomposition of A, and since we want our vectors in $\mathbb{R}^5$ we must multiply by A to get to the U in the SVD, by the equation $\mathbf{u}_i = \frac{A\mathbf{v}_i}{\sigma_i}$:

$$\mathbf{u}_1 = \sqrt{\frac{41}{2}} \begin{bmatrix} 7/2 \\ 1 \\ -1 \\ 0 \\ 5/2 \end{bmatrix}.$$

Using the Change of Basis Theorem, let's see what happens when we try to represent a column of A using our low dimensional representation.

The coordinate for $\mathbf{a}_1$ in the space spanned by $\mathbf{v}_1$ would be:

$$c_1 = \mathbf{u}_1^T \mathbf{a}_1 = \sqrt{\frac{41}{2}} \begin{bmatrix} 7/2 & 1 & -1 & 0 & -5/2 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix} = \frac{19}{2\sqrt{\frac{41}{2}}}.$$

Reconstructing $\mathbf{a}_1$, denoted $\hat{\mathbf{a}}_1$ results in:

$$\hat{\mathbf{a}}_1 = \begin{bmatrix} 1.62 \\ .46 \\ -.46 \\ 0 \\ -1.16 \end{bmatrix}.$$

This is fairly close to the original $\mathbf{a}_1 = \begin{bmatrix} 2 & 0 & 0 & 0 & -1 \end{bmatrix}^T$, a minor miracle considering we only have one vector with which to explain it and the other two vectors. The other vectors, $\hat{\mathbf{a}}_2$ and $\hat{\mathbf{a}}_3$, are a little further off from their original vectors, $\mathbf{a}_2 = \begin{bmatrix} -1 & 2 & -2 & 0 & -1 \end{bmatrix}^T$ and $\mathbf{a}_3 = \begin{bmatrix} -1 & -2 & 2 & 0 & 2 \end{bmatrix}^T$ respectively, but still decent considering the circumstances:

$$\hat{\mathbf{a}}_2 = \begin{bmatrix} .51 \\ .15 \\ -.15 \\ 0 \\ -.37 \end{bmatrix}, \hat{\mathbf{a}}_3 = \begin{bmatrix} -2.13 \\ -.61 \\ .61 \\ 0 \\ 1.52 \end{bmatrix}.$$

Not bad at all considering the restraints given by the one vector. Keep in mind this $\mathbf{v}_1$ is the best one dimensional basis for the data residing in A. Hopefully this small example provided clarity to the power of the best basis.

4.5 SV-Spectral Decomposition

The idea of the best k-dimensional basis given by the first k eigenvalues of the covariance matrix is tied to something we have already seen. One nice fallout from the Spectral Theorem we saw earlier was spectral decomposition, where any symmetric matrix could be written as a sum of products:

$$A = \lambda_1 \mathbf{u}_1 \mathbf{u}_1^T + \lambda_2 \mathbf{u}_2 \mathbf{u}_2^T + \ldots + \lambda_n \mathbf{u}_n \mathbf{u}_n^T.$$

Like we saw with the psuedoinverse, the SVD can be used to generalize this idea to matrices of any size. The following formula allows us to write a matrix of any size as a sum of its singular values and eigenvectors from its singular value decomposition. This allows us to easily and immediately find the best estimation of a matrix in any k-dimensional space by taking the first k terms of the sum.

Singular Value Decomposition Theorem.

$$A = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \sigma_2 \boldsymbol{u}_2 \boldsymbol{v}_2^T + \ldots + \sigma_r \boldsymbol{u}_r \boldsymbol{v}_r^T.$$

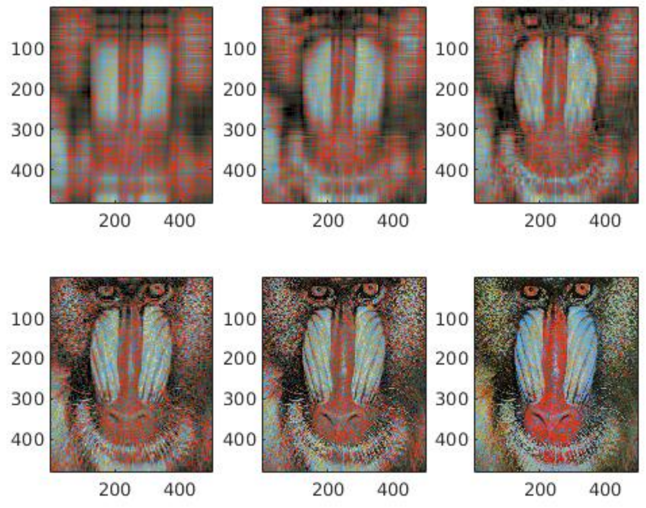
where the σ's are the singular values of A and the $\boldsymbol{u}$'s and $\boldsymbol{v}$'s come from the singular value decomposition [1].

One thing to note is that the terms are arranged in decreasing order with σ_1 the largest singular value. From our discussion on the covariance matrix we know the first k eigenvalues best describe the data. Therefore we could represent a matrix using the first k terms in the previous decomposition.

4.6 Practical Applications

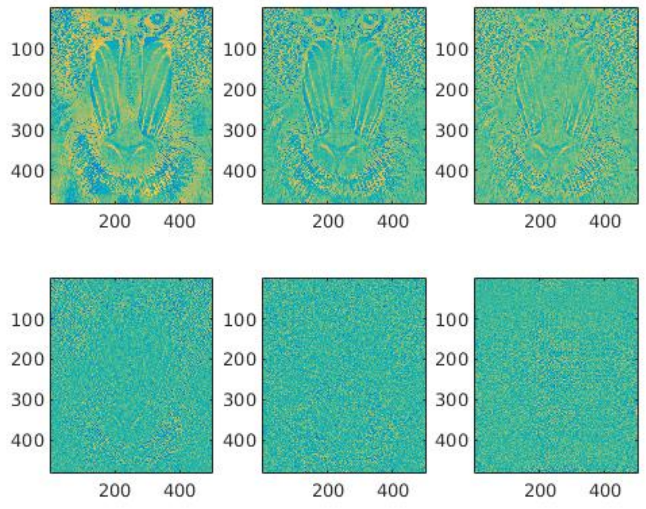
Here we now switch gears away from the theoretical linear algebra to applications of what has been covered so far. Take this picture of a mandrill for instance:

This image is a 480 x 500 matrix, where each entry is an integer in $[0, 220]$ representing a color. We can use the previous formula to estimate the image using the first k terms. Below are the estimations of the mandrill for $k = 3, 6, 10, 50, 150$ and 450 respectively.



At $k = 50$ we can very clearly make out the mandrill. The amazing thing is that we used 1/10th the information of the original matrix. It may not be quite as pretty, but it gets the job done and saves a lot of effort in the process. This is a larger, visual example of how powerful the best basis can be.

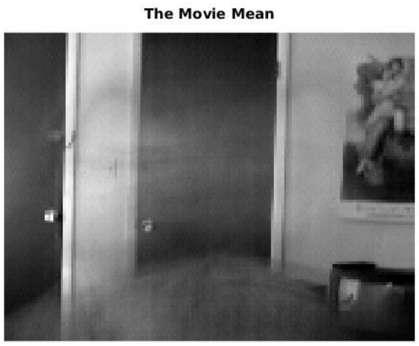
However when going down in dimension, some data is lost. The following image represents the error - the data left out - for each of the previous estimations in the same order.



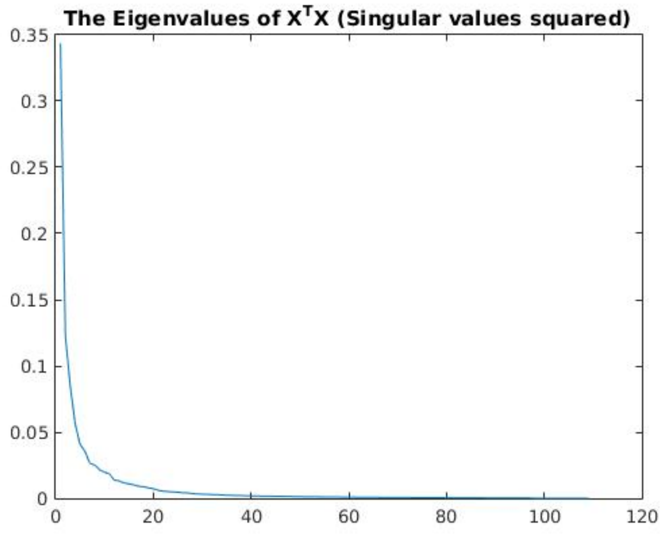
At $k = 50$ and above the error is essentially just noise. Clearly after a certain point it becomes more and more difficult to extract data that really adds a lot. This realization is of use to us: if we can estimate a matrix of any data using far less dimension while keeping the general meaning of the data in tact, that will save time, memory and effort.

Another example of an application of the best basis is in a video. Here is a data set representing a short video, 109 frames long, of Professor Hundley. With each frame being 120 x 160 pixels, or 19200 pixels total, this data set is a matrix of size 19200 x 109, again of integers representing a color (but this time) in grayscale. We can think of this as a matrix,X, in $\mathbb{R}^{19200}$. One of the 109 frames is shown below:

We can also look at the mean of the data, interestingly just the background



Like the example of the mandrill, we want to find a lower dimensional basis of the data but instead we will look at how a different number of eigenvalues affects the quality of the data. Below is a graph of the eigenvalues by their order.



Since this data set has been mean subtracted and the eigenvalues have been normalized, these eigenvalues actually represent the percentage of the data represented if taking just that eigenvalue as a lower dimensional basis. For example, the first five eigenvalues of $X^T X$ are:

$$.3436, .1129, .0847, .0566, .0407$$

Therefore using the first eigenvalue would explain 34.36% of the data while using the first two would keep $34.36 + 11.27 = 45.63\%$ of the data. This is the same as estimating our 109 points in $\mathbb{R}^{19200}$ using only two vectors. We would need 22 eigenvalues to explain 90% of the data, not bad at all considering we have 109 nonzero eigenvalues in the full set of data. Hopefully this example once again showed the use of the singular value decomposition theorem as a tool to approximate large data sets using far less time and effort.

5 Neural Networks

With much of the background material behind us, we can now focus on the topic of this paper: neural networks. The **neural network** is a powerful tool that finds relationships between complex data held in variables and observations. It is used for a variety of tasks such as pattern recognition, line fitting, data clustering and more. Unsurprisingly the network is rooted in linear algebra, including much of what we have just seen.

An important thing to know is that a neural network is based heavily off the biological processes of the brain. In the brain there are neurons that receive data from other neurons through connected strings called axons. Certain neurons are more connected than others, and the brain as a whole is made up of impossibly complex arrays of neurons and axons. The neural network was created to simulate how the brain works. Instead of neurons we have nodes, a collection of data for one variable for multiple observations. Nodes are classified into layers: the input layer and the output layer are necessities while a hidden layer is also often included. In the case of deep learning, there exist multiple hidden layers.

To start, let's understand the data we are looking at and what we are trying to glean from it. There are two instances of a neural network. In a **supervised** neural network (or supervised learning) we are given a collection of input data and a collection of output data. This differs from unsupervised learning where we do not have output. This paper will focus on the supervised case. The goal of the neural network is to predict the output given just the input. Since we have the output in the supervised version, we can figure out how accurate the neural network is, and then update it to make it better. This is an example of **adaptive learning**, something the neural network thrives at; as more points are added to the network, it learns from itself to produce an even better prediction of the output.

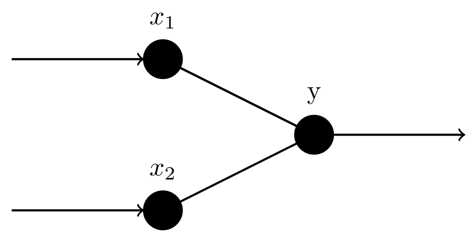
To make things easier, we put the input data in an $m \times p$ matrix, X, and the output in an $n \times p$ matrix, T (where T stands for target). The numbers m and n represent the different variables for X and T respectively, whereas p represents the different observations. For X, each of the p vectors can be thought of as all the variables for one observation. Throughout our talk on neural networks there will be an ongoing example involving cancer. To better understand the basic concepts of neural networks this example will now be presented.

5.1 Example: Cancer

With this example we are given a 9×699 matrix of input values, X, with 9 variables for each of the 699 observations. An observation in this case is a different patient we have data on. We are also given a 2×699 matrix of target values. The 9 variables from the input describe the tumor of a patient: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli and mitoses. Congratulations if you understood over half those words. These 9 variables will be used to predict T. There are two rows in T, one for if the tumor was benign (harmless) and one for if the tumor was malignant (cancerous). A 1 is placed in the row of what actually occurred, so $\begin{bmatrix}1\\0\end{bmatrix}$ would represent benign and $\begin{bmatrix}0\\1\end{bmatrix}$ would represent malignant for a given observation. Using all 699 patients, the neural network will find a relationship between the 9 predictor variables and if the patient has cancer. This is the benefit of a supervised network, as we know exactly what output corresponds to the inputs. With a large amount of data feeding into the network, we can then use the finished product to predict the type of tumor for a patient whom we do not have an output value for, given that we have the 9 predictor variables for them. This example should show how important a neural network can be in application. Now we will see how it works in theory.

5.2 Neural Network Theory

Without further ado, let's look at a visual representation of a neural network.



Here is the most basic neural network, a **perceptron**, where there is one layer of inputs and one layer of outputs. A **layer** in picture form is a series of nodes on the same vertical. Layers are represented by vectors for one observation, so the vector $\mathbf{x}$ is simply $\begin{bmatrix}x_1 & x_2\end{bmatrix}^T$. If we account for all observations, each layer represents a matrix. The matrix for the input layer would then be a $2 \times p$ matrix, X, where p is the number of observations. For the output we see a new variable, y. The matrix Y, generally $n \times p$ but in this case almost trivially $1 \times p$, is the output predicted by the neural network. Note the distinction between Y and T; T is known in supervised learning whereas Y is a prediction of T based on the inputs. The most successful network will be the one that minimizes the difference between T and Y. Otherwise put, the most successful network will be the one that is best able to predict T using just the inputs.

To do this we need a **weight** matrix, W. To map from X to Y, the weight matrix will in general be an $n \times m$ matrix, independent of the number of observations. The goal of neural networks can be reduced to finding W such that:

$$T = WX + \mathbf{b},$$

where $\mathbf{b}$ is just the resting state for each node. This is known as an affine function, meaning almost linear. To make it a linear function, we can augment the matrix W with $\mathbf{b}$, as $[W|\mathbf{b}]$ and add a row of 1's to X. Then we get the same thing as above written as a linear function:

$$T = \begin{bmatrix}W|\mathbf{b}\end{bmatrix}\begin{bmatrix}X \\ 1\end{bmatrix}.$$

We can then write this as:

$$T = WX,$$

if we note that W is now $n \times m + 1$ and X is now $m + 1 \times p$ with a row of 1's at the bottom. We do not need a neural network to 'solve' this equation for W, rather we can just compute the psuedoinverse of X. Therefore:

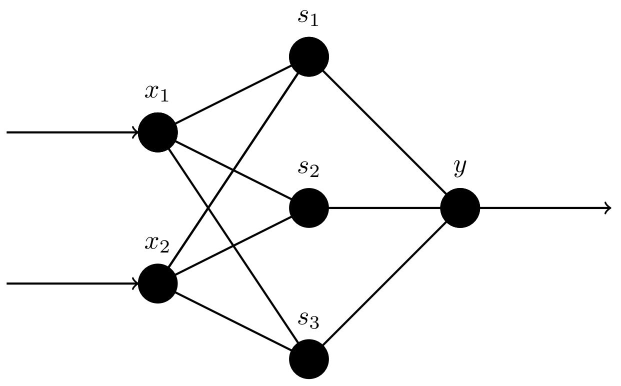
$$\hat{W} = TV\Sigma^{-1}U^T.$$

Notice that we throw a hat on W to indicate that this is only a projection of the real W onto the column space of X, as this is a necessary sacrifice that happens when we use the psuedoinverse. This new matrix $\hat{W}$ can then be used to compute Y:

$$Y = \hat{W}X.$$

The predicted output Y will be equal to the target output T only when T is in the column space of X, a rare occurrence. Otherwise Y will be as close to T as possible using the best basis previously talked about. This is one way to solve this problem, which we call the linear algebra method. It is concise and useful, but also limited. This will work the best if the relationship between X and T is linear, also a rare occurrence. To take more complex functions into account, we will use a neural network.

The pereceptron is not a very interesting case of a neural network as we can accomplish the same thing with the linear algebra method. What makes them more interesting is the addition of a hidden layer:



Note in this case we are still mapping from $\mathbb{R}^2$ to $\mathbb{R}^1$, except this time there is a middle layer of nodes that requires a transformation from $\mathbb{R}^2$ to $\mathbb{R}^3$ before going from $\mathbb{R}^3$ to $\mathbb{R}^1$. The point of a **hidden layer**, in this case S, is to add complexity to the type of relationship between X and Y. When we think X and T are complexly related then we should include a hidden layer with more nodes, whereas less nodes will indicate a simpler relationship.

As in the linear algebra method, we can calculate the product WX. Unlike last time where we found a projection of W, we will be calculating W from past iterations of itself. This goes back to the idea of adaptive learning. Once we have a pretty good W, we do not want to scrap it entirely when going through the network again. Instead we will gently change it to reflect the new data. However this method also requires that we have a W to start with, so for simplicity we initialize W to be, in this case, a 3×2 matrix of random values.

Let $P = WX$. Another thing we can do to increase the complexity is add a function in the hidden layer, applied componentwise to P. This can be extremely beneficial for linear classification problems as we will see with the cancer example. We apply the function from $\mathbb{R}$ to $\mathbb{R}$ to every element in P to get S, which will then become the input matrix for the next layer:

$$P = WX,$$

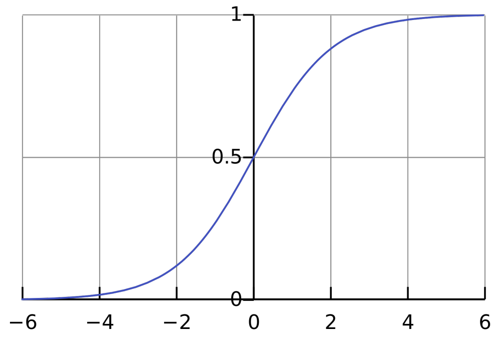
$$S = \sigma(P),$$

$$Y = \bar{W}S = \bar{W}\sigma(WX),$$

where $\bar{W}$ is the second weight matrix. We could also choose to include a function after the hidden layer, but don't in this case as it generally doesn't add as much complexity as the first function. So what is this mysterious function, σ? Well it could be any function, but typically the function we choose is a **Logistic Sigmoid** function (also called logsig). The logsig function is as follows:

$$\sigma = \frac{1}{1 + e^{-\beta x}},$$

and looks like this:



[5]

This function is special because it is asymptotically bounded below by 0 and above by 1 and is very close to one of them in most places. Furthermore, it is a strictly increasing function. This first property should remind you of a probability function. This can be very useful for figuring out the desired output for a linear classification problem. For instance a large input will result in a number very close to 1, which we can interpret as the probability that the function output is actually 1. Recall in the cancer example how benign was $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and malignant was $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. The predicted output elements of Y will very rarely be exactly 1 or exactly 0. However by using the logsig function we can find the probability that any given value in Y really should be a 1 or 0.

With the function taken care of, let's look back at our predicted output matrix, Y:

$$Y = \bar{W}S = \bar{W}\sigma(WX).$$

We have Y, now what? Recall that we wish Y to be as close to T as possible. To do this we must train the network. There are different ways to do this, but the method this paper will focus on is known as **Hebbian learning**. Unlike other types of training, Hebbian or **on-line** learning modifies W and $\bar{W}$ after looking at observations one at a time. This type of training is also dependent on the fact that we are working with a supervised network. When we know the actual value for the output, we can calculate the error term for the kth observation by subtracting the predicted value from the actual value:

$$E(W, \bar{W}) = \mathbf{t}_k - \mathbf{y}_k.$$

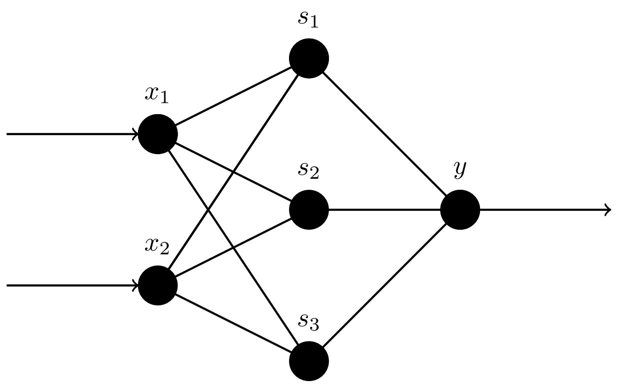
Given that some errors may be positive and some may be negative, we calculate the total error term of the function by squaring the norms of the individual errors and summing over all observations. We square the terms instead of applying the absolute value function so that we can take the derivative. In totality the error function can be given by:

$$E(W, \bar{W}) = \sum_{i=1}^{p} ||\mathbf{t}_k - \mathbf{y}_k||^2.$$

The neural network is at its best when this error term is minimized. However minimizing the function is not so simple as taking a derivative in the traditional sense. Notice that the error is a function of two matrices, each of those having multiple entries. To solve this problem, we will turn to the multivariable calculus technique of gradient descent.

5.3 Build up to Gradient Descent

Let's take a look at our example neural network again:



Notice that there are 6 lines going from the input layer to the hidden layer. These represent the 6 entries in the 3×2 weight matrix, W. Similarly the three lines going from the hidden layer to the output layer represent the 3 entries in the 1×3 matrix, $\bar{W}$. Although I have used matrix form to express the equations, I would now like to write out the problem in all its gory detail to stress the relationship between different variables.

First we start with $S = \sigma(P) = \sigma(WX)$, where it is important to remember that σ is applied componentwise, not to the whole matrix:

$$\begin{bmatrix} s_{1,k} \\ s_{2,k} \\ s_{3,k} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} \begin{bmatrix} x_{1,k} \\ x_{2,k} \end{bmatrix} \right),$$

which in turn is equal to:

$$\begin{bmatrix} s_{1,k} \\ s_{2,k} \\ s_{3,k} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{1,1}x_{1,k} + w_{1,2}x_{2,k} \\ w_{2,1}x_{1,k} + w_{2,2}x_{2,k} \\ w_{3,1}x_{1,k} + w_{3,2}x_{2,k} \end{bmatrix} \right).$$

Don't let the numerous subscripts scare you. The first subscript for each variable is the ith row, the second entry for the weight matrix is the jth column and the k just means we are looking at the kth observation. Notice then how all these terms are just single 1 dimensional values, not vectors or matrices as we've seen before.

Now it should be much clearer how each hidden layer node, $s_{i,k}$, is a function of each entry in W. Instead of one big problem involving a nasty matrix we have 3 problems each with 2 unknowns, m equations with n unknowns generally. We're only halfway through the network though. The back half written out is:

$$y_k = \bar{w}_1 s_{1,k} + \bar{w}_2 s_{2,k} + \bar{w}_3 s_{3,k}.$$

Then plugging in for $s_{i,k}$ we get:

$$y_k = \bar{w}_1\sigma(w_{1,1}x_{1,k}+w_{1,2}x_{2,k})+\bar{w}_2\sigma(w_{2,1}x_{1,k}+w_{2,2}x_{2,k})+\bar{w}_3\sigma(w_{3,1}x_{1,k}+w_{3,2}x_{2,k}).$$

A mess, to say the least, which is why we usually write it in matrix form. However this formula explicitly shows the relationship between y_k and the 9 variables (6 from W, 3 from $\bar{W}$) we want to change. The error function would be a little more complicated but would still only include these 9 variables. Now we can use multivariable calculus to find the w's and $\bar{w}$'s that minimize the error function.

5.4 Gradient Descent

The whole point of **gradient descent** is to find the W and $\bar{W}$ that minimize the error function. Recall the error term for the kth observation is $E(W, \bar{W}) = ||\mathbf{t}_k - \mathbf{y}_k||^2$, though in this example each of these vectors is just 1 point, so we can say $E(W, \bar{W}) = (t_k - y_k)^2$. Unsurprisingly, gradient descent involves computing the **gradient**, a vector that points in the direction of largest increase, though we would negate the gradient to find the direction of largest decrease. The gradient is found by taking the partial derivative of each variable in the equation. In our case we have 9 variables. We start by taking the partial derivative of E with respect to something that y is a function of (the dummy variable h) for the kth observation:

$$\frac{\partial E}{\partial h} = 2(t_k - y_k)(-\frac{\partial y_k}{\partial h})$$

With that in mind, we can now calculate $\frac{\partial y_k}{\partial h}$ in the two cases where $h = w_{i,j}$ and $h = \bar{w}_i$. The second case is a little easier. Earlier we said:

$$y_k = \bar{w}_1 s_{1,k} + \bar{w}_2 s_{2,k} + \bar{w}_3 s_{3,k}.$$

Then clearly we have:

$$\frac{\partial y_k}{\partial \bar{w}_i} = s_{i,k},$$

and in totality we get:

$$\frac{\partial E}{\partial \bar{w}_i} = 2(t_k - y_k)(-s_{i,k}).$$

This accounts for 3 of the variables, since there are 3 entries in $\bar{W}$.

Next we want to find $\frac{\partial y}{\partial w_{i,j}}$. Notice that the same formula for y_k as above and the fact that $s_{i,k}$ is a function of W implies that:

$$\frac{\partial y_k}{\partial w_{i,j}} = \bar{w}_i \frac{\partial s_{i,k}}{\partial w_{i,j}}$$

$$= \bar{w}_i \frac{\partial \sigma(p_{i,k})}{\partial w_{i,j}},$$

using our definition of $\mathbf{p}$ from before. Using the chain rule we can deconstruct the partial even further. Therefore we have:

$$\frac{\partial y_k}{\partial w_{i,j}} = \bar{w}_i \sigma'(p_{i,k}) \frac{\partial p_{i,k}}{\partial w_{i,j}}.$$

The answer to the partial on the right is hidden in our equation for $\mathbf{p}_k$:

$$\begin{bmatrix} p_{1,k} \\ p_{2,k} \\ p_{3,k} \end{bmatrix} = \begin{bmatrix} w_{1,1}x_{1,k} + w_{1,2}x_{2,k} \\ w_{2,1}x_{1,k} + w_{2,2}x_{2,k} \\ w_{3,1}x_{1,k} + w_{3,2}x_{2,k} \end{bmatrix}.$$

Simply the partial on the right is just equal to x_j. Then using the logsig function for σ:

$$\sigma = \frac{1}{1 + e^{-x}},$$

the derivative of this function conveniently turns out to be:

$$\frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x)).$$

When evaluated at $p_{i,k}$, we are able to put it back in terms of $\mathbf{s}$:

$$\sigma(p_{i,k})(1-\sigma(p_{i,k})) = s_{i,k}(1-s_{i,k}).$$

The whole partial is:

$$\frac{\partial y}{\partial w_{i,j}} = \bar{w}_i s_{i,k}(1-s_{i,k})x_{j,k},$$

and going up one more step we finally get:

$$\frac{\partial E}{\partial w_{i,j}} = 2(t_k - y_k)(-\bar{w}_i s_{i,k}(1-s_{i,k})x_{j,k}).$$

This gives us the final six partial derivatives for the six entries in W. The gradient in all its glory then is:

$$= \left\langle \frac{\partial E}{\partial w_{1,1}}, \frac{\partial E}{\partial w_{1,2}}, \frac{\partial E}{\partial w_{2,1}}, \frac{\partial E}{\partial w_{2,2}}, \frac{\partial E}{\partial w_{3,1}}, \frac{\partial E}{\partial w_{3,2}}, \frac{\partial E}{\partial \bar{w}_1}, \frac{\partial E}{\partial \bar{w}_2}, \frac{\partial E}{\partial \bar{w}_3} \right\rangle.$$

But that's not the end. Since Hebbian learning involves calculating the gradient for each observation we would then have to update the weight matrices based on this new observation. The update equation is like this:

$$w_{i,j,new} = w_{i,j,old} - \alpha \frac{\partial E}{\partial w_{i,j,old}},$$

and similarly:

$$\bar{w}_{i,new} = \bar{w}_{i,old} - \alpha \frac{\partial E}{\partial \bar{w}_{i,old}}.$$

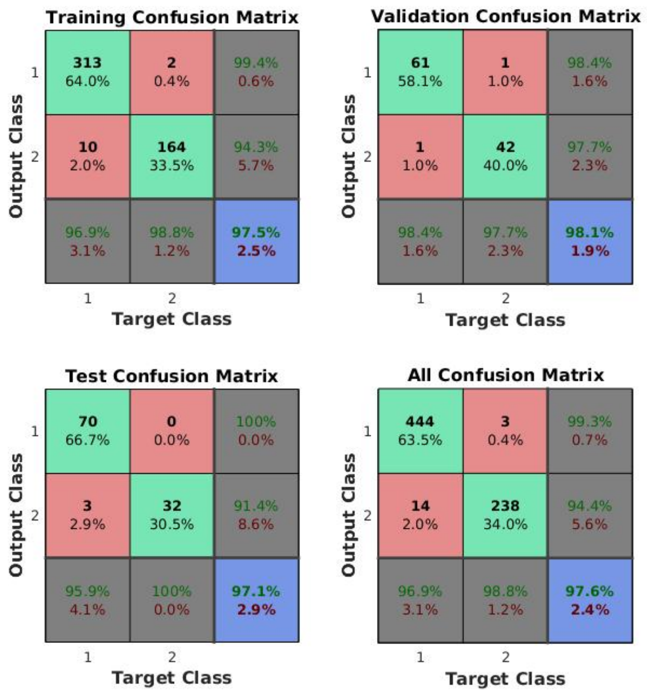
The constant α is the **learning rate**, controlling how drastic the change in weights are, where a large α would indicate a larger change of the weights. After all that we would add a new observation to the network and retrace our steps. Now aren't you glad the computer does this for us?

5.5 Example: Cancer Revisited

Recall the cancer example. Using both the linear algebra method and the neural network method we can try to predict the type of cancer based on the 9 inputs. First we will use a neural network.

Since we have 699 observations, we can divvy up the observations to train the network as just described and to test how our network performed. Generally we want more observations for training, though we can choose what percentages to use. In this case we chose to give 489 (or 70%) of the 699 samples into the network to train it. Of the remaining samples, 15% are used during the validation stage, similar to training except it halts when the accuracy of the network has stopped improving, protecting against overfitting. The rest of the samples are then used to test our model. The accuracy will be different for every trial and will depend on which samples were included in each stage. The hidden layer contains 10 nodes, a choice by us since we expect a certain level of complexity from the relationship.

Using a program called MatLab to run the network, the results can be visualized using a confusion matrix (no, it is not named for how confused you feel looking at it).

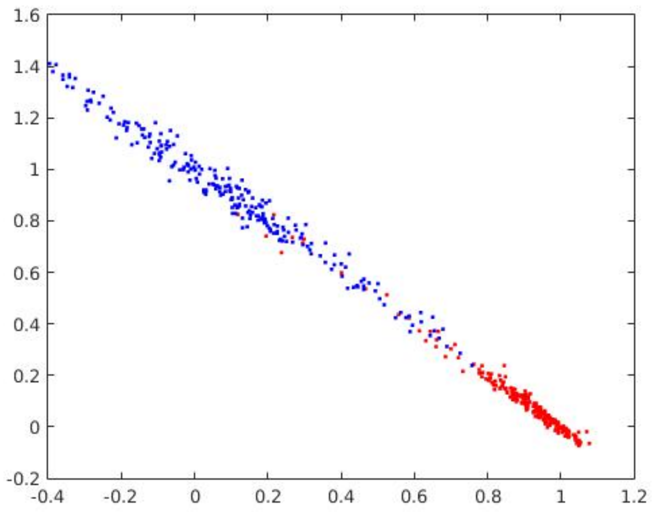


The four squares reveal the accuracy of our network at each stage (training, validation and testing) as well as overall. The target class is what actually happened where the output class is predicted by the neural network, T and Y respectively. Class 1 corresponds to benign and class 2 corresponds to malignant, so the four top left squares in each bigger square show how many points and the percentage of said points that fall into each combination of output and target. For instance when a patient's tumor is malignant (target = 2), the network correctly predicted this every single time during the testing stage. This is good; if you have cancer, you would want to know this and this network would accurately tell you (at least with this data). If a patient did have a malignant tumor and the network predicted otherwise, this error would be known as a **false negative**. The alternative error, a **false positive** would occur when malignant is predicted but is not the truth. This happened 4.1% of the time in the test stage, although one could argue that it is not as grievous of an error as the false negative. Perhaps surprising to some, the network performs very well overall, even when it's in the process of training. The blue square in the bottom right of each larger square is the total accuracy, over 97% for all. Clearly this example shows how powerful a neural network can be in predicting outcomes.

How does the linear algebra method fare compared to the neural network? As explained previously, we first calculate the psuedoinverse of $\hat{X}$ then multiply it by T, which is given, to find $\hat{W}$. Multiplying $\hat{W}$ by $\hat{X}$ gets us the predicted matrix Y. Here are the first eight columns of Y's 699 total using this method:

$$\begin{bmatrix} 1.0187 & 1.0135 & 0.9341 & 0.4624 & 0.8169 & 0.9683 & 0.1173 & -0.0468 \\ -0.0203 & -0.0062 & 0.0773 & 0.5725 & 0.1625 & 0.0120 & 0.9284 & 1.1300 \end{bmatrix}$$

Notice there are two rows. Why is this the case? For a linear classification problem with k classes we set the target value as each of the elementary basis vectors in $\mathbb{R}^k$. In this case, there are two classes (benign and malignant) and so the values for any target value is either $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ or $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Since the eight columns above are from the matrix Y, the values are not exactly either of these two vectors, but there are certain places where you can see it is close to one or the other. Graphing all 699 values in Y shows us how close they are to the two points we want.



The blue points are when the actual value was malignant while the red corresponds to benign. We would use the perpindicular line through $(.5, .5)$ as a cutoff for determining the prediction: left of the line we classify as malignant and right of the line we classify as benign. Some predictions from Y are well off the mark but when taken as a whole this is fairly accurate. It is also interesting to note that the points are all very close to or on the line going from $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ to $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. This shows that this problem is almost linear, and so the linear algebra method should perform similarly to the neural network.

We can check a confusion matrix, Co, to see how well it performed:

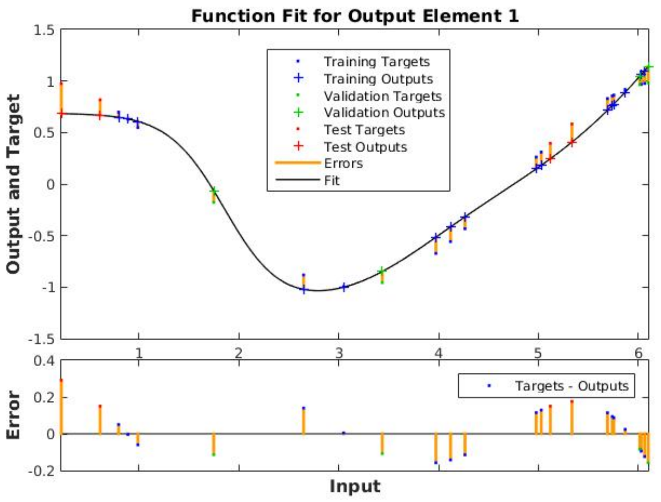
$$Co = \begin{bmatrix} 449 & 18 \\ 9 & 223 \end{bmatrix},$$

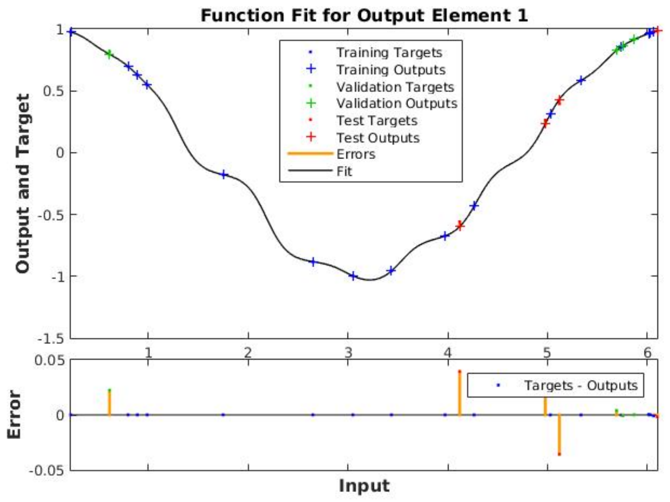
$$Co = \begin{bmatrix} 98.03\% & 7.47\% \\ 1.97\% & 92.53\% \end{bmatrix}.$$

MatLab comes with a neural network toolbox that created the last confusion matrix automatically, the reason why this one might not be as visually appealing. However it still gets the job done. Just like the last confusion matrix the columns represent what the actual value was (benign and malignant respectively) while the rows represent what was predicted. The percentages indicate the accuracy keeping target constant (the two left boxes on the bottom row in the previous confusion matrix). So the 98% could be interpreted as the percentage of the time the algorithm correctly predicted a benign tumor, whereas the 7% is the percentage the algorithm predicted benign when the tumor was actually malignant. This is not a bad method for predicting, as it produced a correct output 96% of the time. However a neural network blows it out of the water, especially when it comes to the false negative case. The 7.47% error in this case is far inferior to the 0% from the neural network, especially for something as morbidly awful as telling a patient they do not have cancer when they really do. Although the linear algebra method was fairly successful for this data set, for this and more complicated relationships a neural network is the way to go.

5.6 Example: Function Fitting

Here is another example of what a neural network can accomplish. The cancer example showed a neural network performing **linear classification**, designating output into one of two categories, but a neural network can also perform function fitting. In this example, a data set was generated using 25 random points from $[0, 2\pi]$ to be the input vector $\mathbf{x}$. The output matrix was given by the function $\mathbf{t}_1 = \cos(\mathbf{x})$ and $\mathbf{t}_2 = \sin(\mathbf{x})$ where $\mathbf{t}_1$ and $\mathbf{t}_2$ are the first and second rows of the target matrix T. In this case and unlike most other cases, there is a specific function between the inputs and outputs. We know this function, but the neural network does not. This data set then is a good test for the neural network to see how accurate it can be. To see the difference between different hidden layers, we can use different amount of nodes in our network and compare the output. First we will try a three node hidden layer, then a ten node hidden layer.





What we can see here is that the three neuron fit has far more error, but it's line is much smoother giving a better general fit for the data. The ten neuron fit is far better at reducing the error but may have overfit the data, as we know the actual relationship between input and output is not nearly as complicated as the fit it gives us. This is a good example of how the different number of neurons can affect the results of the network.

6 Conclusion

Alas, I have reached the end of what I covered this semester on neural networks. There is so much information regarding the networks, singular value decomposition, best bases and more yet there was only so much time to cover them. A more complete report would have covered back propagation of error, a technique used at the end of the neural network. There are also many different types of neural networks to explore, including different uses (data clustering for instance) and different training methods (taking all observations at once). It would also have been interesting to see the difference over many samples between the linear algebra method and the neural network, as well as exploring mathematically how the different number of nodes in the hidden layer affect the conclusions. Neural networks are a fairly new field, yet it would still be interesting to research the history of them. Certainly there is so much more about neural networks to learn and discover, yet this paper gives the basics that are essential to understand. I would like to acknowledge the help of Dylan Zukin, Professor Balof and especially Professor Hundley for editing and general teaching throughout the learning proccess.

References

David C. Lay. *Linear Algebra and its Applications Fourth Edition.* Addison-Wesley, Boston, Massachusetts, 2012.

Douglas Hundley `http://people.whitman.edu/ hundledr/courses/index.html`

Dolhansky, Brian. "Artificial Neural Networks: Linear Classification (Part 2)." RSS. N.p., 23 Sept. 2013. Web. 29 Mar. 2016.

Stansbury, Dustin. "A Gentle Introduction to Artificial Neural Networks." The Clever Machine. N.p., 11 Sept. 2014. Web. 29 Mar. 2016.

"Logistic Function." Wikipedia. Wikimedia Foundation, n.d. Web. 02 May 2016.

Alphabetical Index

Adaptive Learning, 24

Basis, 4
Basis Vectors, 6

Column Space, 7
Coordinates, 6
Covariance Matrix, 14

Eigenvalue, Eigenvector, 9

False Negative, 33
False Positive, 33
Full Singular Value Decomposition, 12

Gradient, 30
Gradient Descent, 29

Hebbian Learning, 28
Hidden Layer, 26

Learning Rate, 31
Linear Classification, 34
Linear Combination, 4
Linear Independence, 4
Logistic Sigmoid Function, 27

Mean Subtracted, 14

Neural Network, 3, 23

Orthogonally Diagonalizable, 9
Orthonormal, 7
Orthonormal Set, 6

Perceptron, 25
Psuedoinverse, 14, 25

Reconstruction, 7, 8
Reduced Singular Value Decomposition, 12

Singular Value Decomposition, 10
Spectral Decomposition, 10, 19
Supervised Network, 24, 28
Symmetric Matrix, 9

Unit Vector, 6