LINEAR ALGEBRAIC METHODS IN DATA SCIENCE AND NEURAL NETWORKS

A Thesis
Submitted to the Faculty
of
Washington and Lee University

By

Jackson Gazin

In Partial Fulfillment of the Requirements
for the Degree of
BACHELOR OF SCIENCE
WITH HONORS

Major Department: Mathematics
Thesis Advisor: Dr. Michael R. Bush
Second Reader: Dr. Cody Watson

May 2022

ABSTRACT

This thesis is about some of the methods and concepts of linear algebra that are particularly helpful for data analysis. After a brief review of some linear algebra concepts in chapter 1, the second chapter of the thesis centers around the singular value decomposition (SVD) which expresses any matrix A as a product of an orthogonal matrix, a diagonal matrix, and another orthogonal matrix. Understanding the SVD requires understanding the properties of symmetric matrices, which are explained first. Chapter 3 focuses on the applications of the SVD. It begins with using the SVD for low-rank approximation, and then explores how the SVD is applied in principle component analysis.

Chapter 4 introduces neural networks, a machine learning architecture useful for image recognition among other applications. It introduces the structure of a neural network in linear algebraic notation; one of the main goals of chapter 4 is to reinforce the idea that neural networks can be seen as compositions of matrix transformations with non-linear activation functions. We then introduce how the parameters in a neural network are optimized. Chapter 5 deals with convolutional neural networks. It also focuses heavily on circulant matrices, and the relationship between convolution and circulant matrices. Understanding the properties of circulant matrices will be instrumental in understanding the benefits of convolution. We finish the chapter by showing how PCA can be used as a data pre-processing tool before running the data through a convolutional neural network.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

This chapter will introduce some basic ideas from linear algebra that will be necessary to understand the content in this thesis. A reader who has taken a college-level linear algebra course should be familiar with these ideas and could skip this chapter if they wanted. However, this may be helpful for a reader who hasn't encountered linear algebra in a while.

The first section of the chapter will review what eigenvectors and eigenvalues are, how to solve for them, and their implications. It will also introduce what it means to diagonalize a matrix which will become particularly important when we introduce the idea of singular value decomposition.

The second section of the chapter introduces the concept of orthogonality. Most importantly, it displays the Gram-Schmidt algorithm which presents an algorithm to construct an orthogonal basis from any basis. This algorithm will also be extremely important when we present the concept of singular value decomposition. The final section of this chapter will introduce some different miscellaneous ideas that are referenced in the thesis.

We omit most of the proofs in this chapter. A more detailed discussion can be found in various introductory texts including [9] and [6].

1.1. Eigenvectors

Definition 1.1. An **eigenvector** of an $n \times n$ matrix A is a nonzero vector $\mathbf{x} \in \mathbb{R}^n$ such that $A\mathbf{x} = \lambda\mathbf{x}$ for some scalar λ. We call λ an **eigenvalue**, and we also say that $\mathbf{x}$ is a $\lambda-$eigenvector

Lemma 1.2. *λ is an eigenvalue of A if and only if λ is the root of*

$$c_A(\lambda) = det\,(A - \lambda I)\,.$$

Lemma 1.3. *The λ-eigenvectors $\boldsymbol{x}$ are the non-zero solutions to the homogeneous system*

$$(\lambda I - A)\,\boldsymbol{x} = \boldsymbol{0}.$$

Note: The λ- eigenspace E_λ is the full set of solutions including $\boldsymbol{x} = \boldsymbol{0}$

Definition 1.4. An eigenvalue λ of a square matrix A is said to have **multiplicity** m if it occurs m times as a root of the characteristic polynomial $c_A(\lambda)$.

Definition 1.5. A square matrix A is said to be **diagonalizable** if $A = PDP^{-1}$ for some invertible matrix P and some diagonal matrix D.

Proposition 1.6. *An $n \times n$ matrix A can be diagonalized when A has eigenvectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ such that the matrix $P = [\boldsymbol{x}_1 \boldsymbol{x}_2 \ldots \boldsymbol{x}_n]$ is invertible. Then $A = PDP^{-1}$, with*

$$D = P^{-1}AP = diag\,(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

where $\boldsymbol{x}_i$ is a λ_i-eigenvector for $1 \leq i \leq n$.

Definition 1.7. For any eigenvalue λ, the eigenspace E_λ is defined

$$E_\lambda = \{\mathbf{x} \in \mathbb{R}^n | (\lambda I - A)\mathbf{x} = \mathbf{0}\}.$$

Proposition 1.8. *An $n \times n$ matrix A is diagonalizable if and only if every eigenvalue λ of multiplicity m_λ yields an eigenspace E_λ such that $\dim(E_\lambda) = m_\lambda$; that is if and only if the general solution of the system $(\lambda I - A)\,\boldsymbol{x} = \boldsymbol{0}$ has exactly m_λ parameters.*

Lemma 1.9. *An $n \times n$ matrix matrix A is diagonalizable if and only if $\mathbb{R}^n$ has a basis $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ consisting of eigenvectors of A*

Algorithm 1.10. Diagonalization Algorithm

- Step 1) Find the distinct eigenvalues λ of A
- Step 2) Compute each basis for each eigenspace E_λ by solving the solutions to the homogeneous system
$$(\lambda I - A)\,\mathbf{x} = \mathbf{0}$$
- Step 3) If the total number of eigenvectors from step 2 is equal to n, then the matrix is diagonalizable.
- Step 4) If A is diagonalizable, the $n \times n$ matrix P with these basic eigenvectors as its columns is the **diagonalizing matrix** for A, meaning that P is invertible and $P^{-1}AP$ is diagonal.

1.2. Orthogonality

Definition 1.11. For two vectors $\mathbf{x} = [x_1, \ldots, x_n]$ and $\mathbf{y} = [y_1, \ldots, y_n]$, their dot product $\mathbf{x} \cdot \mathbf{y}$ is defined

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n.$$

Definition 1.12. Two vectors $\mathbf{u}$ and $\mathbf{v} \in \mathbb{R}^n$ are **orthogonal** if $\mathbf{u} \cdot \mathbf{v} = 0$.

If a vector $\mathbf{z}$ is orthogonal to every vector in a subspace W of $\mathbb{R}^n$ then $\mathbf{z}$ is said to be orthogonal to W. The set of all vectors $\mathbf{z}$ that are orthogonal to W is called to **orthogonal complement of W** and is denoted by $W^\perp$. $W^\perp$ is a subspace of $\mathbb{R}^n$

Definition 1.13. A set of vectors $\{\mathbf{u}_1, \ldots., \mathbf{u}_p\}$ in $\mathbb{R}^n$ is said to be an **orthogonal set** if each pair of distinct vectors from the set is orthogonal.

Lemma 1.14. *If $S = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p\}$ is an orthogonal set of nonzero vectors in $\mathbb{R}^n$, then S is linearly independent and hence is a basis for the subspace spanned by S.*

Definition 1.15. An **orthogonal basis** for a subspace W of $\mathbb{R}^n$ is a basis for W that is also an orthogonal set.

Lemma 1.16. *Let $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p\}$ be an orthogonal basis for a subspace W of $\mathbb{R}^n$. For each $\boldsymbol{y}$ in W, the weights in the linear combination $\boldsymbol{y} = c_1\boldsymbol{u}_1 + \ldots + c_p\boldsymbol{u}_p$ are given by*

$$c_j = \frac{\boldsymbol{y} \cdot \boldsymbol{u}_j}{\boldsymbol{u}_j \cdot \boldsymbol{u}_j} \quad (j = 1, \ldots, p)\,.$$

Lemma 1.17. *Let W be a subspace of $\mathbb{R}^n$ and $\boldsymbol{x}$ be a vector in $\mathbb{R}^n$. Then we can write $\boldsymbol{x}$ uniquely as*

$$\boldsymbol{x} = \boldsymbol{x}_W + \boldsymbol{x}_{W^\perp}$$

where $\boldsymbol{x}_W \in W$ and $\boldsymbol{x}_{W^\perp} = \boldsymbol{x} - \boldsymbol{x}_W \in W^\perp$.

The vector $\mathbf{x}_{W^\perp}$ is called the orthogonal projection of $\mathbf{x}$ on W. We have:

Proposition 1.18. *For W a subspace of $\mathbb{R}^n$ with an orthogonal basis $\{\boldsymbol{u}_1, \ldots., \boldsymbol{u}_p\}$ the orthogonal projection of a vector $\boldsymbol{x}$ in $\mathbb{R}^n$ onto W is*

$$\text{proj}_W \boldsymbol{x} = \left(\frac{\boldsymbol{x} \cdot \boldsymbol{u}_1}{\boldsymbol{u}_1 \cdot \boldsymbol{u}_1}\right)\boldsymbol{u}_1 + \ldots + \left(\frac{\boldsymbol{x} \cdot \boldsymbol{u}_p}{\boldsymbol{u}_p \cdot \boldsymbol{u}_p}\right)\boldsymbol{u}_p.$$

Note that $\text{proj}_W \boldsymbol{x} \in W$ since $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p$ are all $\in W$

Lemma 1.19. *Let W be a subspace of $\mathbb{R}^n$, let $\boldsymbol{y}$ be any vector in $\mathbb{R}^n$ and let $\hat{\mathbf{y}} = proj_W(\boldsymbol{y})$. Then $\hat{\mathbf{y}}$ is the closest point in W to $\boldsymbol{y}$ in the sense that*

$$||\boldsymbol{y} - \hat{\mathbf{y}}|| < ||\boldsymbol{y} - \boldsymbol{v}||$$

for all $\boldsymbol{v}$ in W distinct from $\hat{\mathbf{y}}$.

Lemma 1.20. *Let $\{\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_m\}$ be an orthogonal set of nonzero vectors in $\mathbb{R}^n$. Given $\boldsymbol{x}$ in $\mathbb{R}^n$, write*

$$\boldsymbol{f}_{m+1} = \boldsymbol{x} - \left(\frac{\boldsymbol{x} \cdot \boldsymbol{f}_1}{\boldsymbol{f}_1 \cdot \boldsymbol{f}_1}\right)\boldsymbol{f}_1 - \left(\frac{\boldsymbol{x} \cdot \boldsymbol{f}_2}{\boldsymbol{f}_2 \cdot \boldsymbol{f}_2}\right)\boldsymbol{f}_2 - \ldots - \left(\frac{\boldsymbol{x} \cdot \boldsymbol{f}_m}{\boldsymbol{f}_m \cdot \boldsymbol{f}_m}\right)\boldsymbol{f}_m$$

Then:

1. *$\boldsymbol{f}_{m+1} \cdot \boldsymbol{f}_k = 0$ for $k = 1, 2, \ldots, m$*
2. *If $\boldsymbol{x}$ is not in $\text{span}\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m, \boldsymbol{f}_{m+1}\}$, then $\boldsymbol{f}_{m+1} \neq \boldsymbol{0}$ and orthogonal to $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_m$*

This tells us that for the orthogonal set $S \in \mathbb{R}^n$, we can use a vector $\mathbf{x} \in \mathbb{R}^n$ where $\mathbf{x} \notin \text{span(S)}$ to add another vector to the set, while keeping the set orthogonal. Vectors in orthogonal sets are non-zero. If $\{\mathbf{f}_1, \ldots, \mathbf{f}_m\}$ is an orthogonal set of vectors and $\mathbf{f}_{m+1} \notin Span\{\mathbf{f}_1, \ldots, \mathbf{f}_m\}$ then $\{\mathbf{f}_1, \ldots, \mathbf{f}_m, \mathbf{f}_{m+1}\}$ will also be an orthogonal set of non-zero vectors.

Lemma 1.21. *Every subspace has an orthogonal basis, and we can find the orthogonal basis applying the Gram–Schmidt algorithm to any basis.*

Algorithm 1.22. Gram-Schmidt Algorithm: If $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ is any basis of a subspace U of $\mathbb{R}^n$, construct $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_m$ in U successively as follow

$\mathbf{f}_1 = \mathbf{x}_1$

$\mathbf{f}_2 = \mathbf{x}_2 - \frac{\mathbf{x}_2 \cdot \mathbf{f}_1}{||\mathbf{f}_1||^2}\mathbf{f}_1$

$\mathbf{f}_3 = \mathbf{x}_3 - \frac{\mathbf{x}_3 \cdot \mathbf{f}_1}{||\mathbf{f}_1||^2}\mathbf{f}_1 - \frac{\mathbf{x}_3 \cdot \mathbf{f}_2}{||\mathbf{f}_2||^2}\mathbf{f}_2$

.

.

where in general we have

$\mathbf{f}_k = \mathbf{x}_k - \frac{\mathbf{x}_k \cdot \mathbf{f}_1}{||\mathbf{f}_1||^2}\mathbf{f}_1 - \frac{\mathbf{x}_k \cdot \mathbf{f}_2}{||\mathbf{f}_2||^2}\mathbf{f}_2 - \ldots - \frac{\mathbf{x}_k \cdot \mathbf{f}_{k-1}}{||\mathbf{f}_{k-1}||^2}\mathbf{f}_{k-1}$

for each $k = 2, 3, \ldots, m$ then

1) $\{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_m\}$ is an orthogonal basis of U

2) $\text{span}\{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_k\} = \text{span}\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ for each $k = 1, 2, \ldots, m$.

The definition of an orthogonal projection of a vector $\mathbf{x}$ in $\mathbb{R}^n$ onto a subspace W of $\mathbb{R}^n$ leads directly to the Gram-Schmidt Process. If we have an orthogonal set of vectors, we can add another vector to the set and keep the set orthogonal by choosing a vector outside of the subspace spanned by the original set $\{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_k\}$ and subtracting it from the orthogonal projection of $\mathbf{x}$ onto that subspace. We know that this will not only be outside of the original subspace, but it will also be orthogonal to every basis vector for the subspace. The Gram-Schmidt process is saying we can extend an orthogonal set which spans W by adding the vector $\mathbf{x} - proj_W\mathbf{x}$ to the set, where $\mathbf{x}$ is a vector outside of W.

1.3. Matrix as a Linear Transformation and Change of Basis Matrix.

If $T : V \to W$ is a linear transformation where $V = \mathbb{R}^n$ and $W = \mathbb{R}^m$, we can always describe T as a multiplication by an $m \times n$ matrix A given bases for V and W.

Lemma 1.23. *Suppose the set $\mathcal{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_p\}$ is a basis for the subspace H. For each $\boldsymbol{x}$ in H, **the coordinate of $\boldsymbol{x}$ relative to $\mathcal{B}$** is defined*

$$[\boldsymbol{x}]_{\mathcal{B}} = \begin{bmatrix} c_1 \\ \vdots \\ c_p \end{bmatrix}$$

where $\boldsymbol{x} = c_1\boldsymbol{b}_1 + \ldots + c_p\boldsymbol{b}_p$

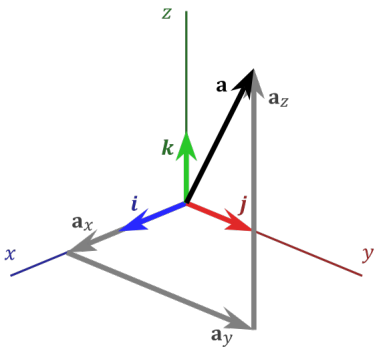


Figure 1. Viewing the vector $\mathbf{a}$ as vector $\mathbf{a}_z$ defined by the basis with a new coordinate space

Let $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be a basis for V and let $\mathcal{E}$ be a basis for W. If $T : V \to W$ is a linear transformation and $\mathbf{x} = d_1\mathbf{b}_1 + \ldots d_n\mathbf{b}_n$ then

$$T(\mathbf{x}) = T(d_1\mathbf{b}_1 + \ldots d_n\mathbf{b}_n) = d_1T(\mathbf{b}_1) + \ldots d_nT(\mathbf{b}_n)$$

and

$$[T(\mathbf{x})]_\mathcal{E} = [d_1T(\mathbf{b}_1) + \ldots d_nT(\mathbf{b}_n)]_\mathcal{E} = d_1[T(\mathbf{b}_1)]_\mathcal{E} + \ldots + d_n[T(\mathbf{b}_n)]_\mathcal{E} = M \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} = M[\mathbf{x}]_\mathcal{E}$$

where

$$M = [[T(\mathbf{b}_1)]_\mathcal{E} \ldots [T(\mathbf{b}_n)]_\mathcal{E}]$$

and we call M the matrix for T with respect to the bases $\mathcal{B}$ and $\mathcal{E}$.

It is important to remember we can think of any basis as a way to describe a coordinate system. More specifically, the basis will tell us the direction and unit of change for each axis in the coordinate system. Once we pick a basis, each vector in the vector space can be described as a coordinate vector with respect to that basis. Often, when we see any random vector, we are assuming it to be a coordinate vector for the standard basis. This means we are defining the vector with respect to the traditional coordinate system. However, if we pick a different basis, our coordinate system reflects that basis. Thus, if $T : V \to W$ is a linear transformation where $V = \mathbb{R}^n$ and $W = \mathbb{R}^m$, we can always describe T as a multiplication by an $m \times n$ matrix A. However, this matrix A is completely determined by the bases we pick for both V and W.

1.4. Miscellaneous

Definition 1.24. For an $m \times n$ matrix A, the transpose A^T is a matrix where $[A^T]_{ji} = A_{ij}$ for all i, j.

Lemma 1.25. $(AB)^T = B^TA^T$

Definition 1.26. An $n \times n$ square matrix A is an orthogonal matrix if the columns of A are orthogonal to the rows of A^T. This occurs when $AA^T = I$, equivalently, $A^T = A^{-1}$

Definition 1.27. The column space (also called the range or image) of a matrix A is the span (set of all possible linear combinations) of its column vectors.

Definition 1.28. The row space of a matrix A is the span of the row vectors in A.

Definition 1.29. Given two vectors of size $m \times 1$ and size $n \times 1$ respectively

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}, \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

their outer product $\mathbf{u} \otimes \mathbf{v}$ is defined:

$$\mathbf{u} \otimes \mathbf{v} = A = \begin{bmatrix} u_1v_1 & u_1v_2 & \dots & u_1v_n \\ u_2v_1 & u_2v_2 & \dots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \dots & u_mv_n \end{bmatrix}$$

Lemma 1.30. *If $\boldsymbol{u}$ and $\boldsymbol{v}$ are both non-zero vectors, then their outer product $\boldsymbol{u} \otimes \boldsymbol{v}$ has rank 1.*

CHAPTER 2. SINGULAR VALUE DECOMPOSITION

The singular value decomposition (SVD) of a matrix is a decomposition of the matrix into a product of an orthogonal matrix, a diagonal matrix, and another orthogonal matrix. It is one of the most powerful ideas in linear algebra. However, to understand it fully one must first understand certain facts about symmetric matrices. Thus, our first section will show that all symmetric matrices are orthogonally diagonalizable. Not only can we construct a basis of eigenvectors for any symmetric matrix, but the the matrix formed out of these vectors, P, will be an orthogonal matrix! We will then make this relationship between orthogonal diagonalization and symmetric matrices even tighter; a matrix is orthogonally diagonalizable if and only if it is a symmetric matrix. This result is known as the spectral theorem.

The second section introduces the singular value decomposition of a matrix. It relies on the concepts of symmetric matrices and the fact that for any matrix A both AA^T and A^TA will be symmetric matrices. The SVD says that for any $m \times n$ matrix A, $A = U\Sigma V^T$ where U is made of an orthogonal basis of eigenvectors for AA^T and V is made up of an orthogonal basis of eigenvectors for A^TA.

We finally introduce the idea that when we represent any linear transformation as a matrix transformation, the SVD helps us map this transformation through all fundamental subspaces. The idea that the spectral theorem and the singular value decomposition can help us choose the right coordinate systems for each fundamental subspace is explained further in [5].

The final section on quadratic forms will be helpful when we consider the applications of the singular value decomposition such as low-rank approximation and principle component analysis. This section will also introduce the idea of an orthogonal change of basis matrix.

2.1. Symmetric Matrices

Definition 2.1. A **symmetric matrix** is a $n \times n$ matrix A that is equal to its transpose. This means that for all $1 \leq i, j \leq n$ $a_{ij} = a_{ji}$.

Definition 2.2. A matrix A is **orthogonally diagonalizable** if there exists an orthogonal matrix P and a diagonal matrix D such that $A = PDP^{-1} = PDP^T$.

First, we see that orthogonally diagonalizable matrices are always symmetric matrices. The diagonal matrix D is always symmetric.

Lemma 2.3. *If a matrix is orthogonally diagonalizable, it is a symmetric matrix*

Proof. If $A = PDP^T$ and D is diagonal, then $A^T = (PDP^T)^T = P^{TT}D^TP^T = PDP^T = A$ □

In fact, the converse holds. Before showing this, we need to prove other results about symmetric matrices.

Lemma 2.4. *For any real symmetric matrix A, if λ is a complex eigenvalue of A, then λ is real.*

Proof.

Our goal is to prove that $\overline{\lambda} = \lambda$. $\overline{A} = A$ because A is real. Let $\mathbf{x}$ be a (possibly complex) eigenvector corresponding to λ so that $\mathbf{x} \neq \mathbf{0}$ and $A\mathbf{x} = \lambda\mathbf{x}$. Let $c = \mathbf{x}^T\overline{\mathbf{x}}$. If $\mathbf{x}^T = [z_1, \dots z_n]$ then

$$\mathbf{x} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

and

$$c = \mathbf{x}^T\overline{\mathbf{x}} = z_1\overline{z_1} + z_2\overline{z_2} + ... + z_n\overline{z_n} = |\overline{z_1}|^2 + |\overline{z_2}|^2 + ... + |\overline{z_n}|^2$$

Thus, c is a real number and $c > 0$ since $z_i \neq 0$ for some i (as $\mathbf{x} \neq \mathbf{0}$). We can now show that $\lambda = \overline{\lambda}$. Observe that

$$\lambda c = \lambda(\mathbf{x}^T\overline{\mathbf{x}}) = (\lambda\mathbf{x})^T\overline{\mathbf{x}} = (A\mathbf{x})^T\overline{\mathbf{x}} = \mathbf{x}^T A^T\overline{\mathbf{x}}$$

Since $A = \overline{A} = A^T$, we have

$$\mathbf{x}^T A\overline{\mathbf{x}} = \mathbf{x}^T(\overline{A}\overline{\mathbf{x}}) = \mathbf{x}^T(\overline{A\mathbf{x}}) = \mathbf{x}^T(\overline{\lambda\mathbf{x}}) = \mathbf{x}^T(\overline{\lambda}\overline{\mathbf{x}}) = \overline{\lambda}\mathbf{x}^T\overline{\mathbf{x}} = \overline{\lambda}c$$

Thus, $\lambda c = \overline{\lambda}c$ meaning that $\lambda = \overline{\lambda}$ since $c \neq 0$.

This means that $A - \lambda I_n$ is a real matrix, and $det(A - \lambda I) = 0$ yields non-zero real solutions. Thus, there are also real eigenvectors for each eigenvalue λ. □

Lemma 2.5. *Let $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_k\}$ be a linearly independent set of eigenvectors of an $n \times n$ matrix A, and extend it to the basis $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_k, ...\boldsymbol{x}_n\}$ of $\mathbb{R}^n$ and let*

$$P = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \dots & \boldsymbol{x}_n \end{bmatrix}$$

be the invertible $n \times n$ matrix with $\boldsymbol{x}_i$ as its columns. If $\lambda_1, \lambda_2, ..., \lambda_k$ are the (not necessarily distinct) eigenvalues of A corresponding to $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_k$ then $P^{-1}AP$ has block form

$$\begin{bmatrix} diag(\lambda_1, \lambda_2..., \lambda_k) & B \\ 0 & A_1 \end{bmatrix}$$

where B has size $n \times n - k$ and A_1 has size $n - k \times n - k$.

Now we have the facts to prove that not only are all orthogonally diagonalizable matrices are symmetric, but all symmetric matrices are orthogonally diagonalizable.

Theorem 2.6. ***Spectral Theorem*** *The following conditions are equivalent for any $n \times n$ matrix A*

1. *A has an orthonormal set of n eigenvectors*

2. *A is orthogonally diagonalizable*

3. *A is symmetric*

Proof. We need to prove that (1) $\Leftrightarrow$ (2) $\Leftrightarrow$ (3). Well, we have that (1) $\Rightarrow$ (2): Let $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ be orthonormal set of eigenvectors of A. If we let $P = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]$ then P is orthogonal and

$$AP = [A\mathbf{x}_1, ..., A\mathbf{x}_n] = [\lambda_1\mathbf{x}_1, ..., \lambda_n\mathbf{x}_n] = PD$$

where $D = \text{diag}(\lambda_1, ..., \lambda_n)$. Since $P^{-1} = P^T$ we have $A = PDP^T$, meaning that A is orthogonally diagonalizable.

(2) $\Rightarrow$ (1): Let $P^{-1}AP$ be diagonal where P is orthogonal. If $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ are the columns of P then $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ is an orthonormal basis of $\mathbb{R}^n$ that consists of eigenvectors of A.

(2) $\Rightarrow$ (3): This statement is Lemma 2.3 which we have proved.

Now, we have (1) $\Leftrightarrow$ (2) and (2) $\Rightarrow$ (3). If we show that (3) $\Rightarrow$ (2), by the transitive property we will have (1) $\Leftrightarrow$ (2) $\Leftrightarrow$ (3) which means we have proven the spectral theorem.

Base Case: $n = 1$. If $A = [a]$, then $A = [1][a][1]^T$. Thus, if A is a 1×1 (symmetric) matrix, it will be orthogonally diagonalizable.

Inductive Step: Assume that all $n-1 \times n-1$ symmetric matrices where $n - 1 \geq 1$ are orthogonally diagonalizable.

Now, consider an $n \times n$ symmetric matrix A. We know that A has a real eigenvalue λ_1 by Lemma 2.4. Let $A\mathbf{x}_1 = \lambda_1\mathbf{x}_1$ where $|\mathbf{x}_1| = 1$ is a normalized eigenvector for λ_1.

We can add vectors to $\mathbf{x}_1$ to find a basis for $\mathbb{R}^n$, and then use the Gram-Schmidt algorithm to find an orthonormal basis $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ for $\mathbb{R}^n$. So $P_1 = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]$ is an orthonormal matrix where x_1 is an eigenvector, and the rest of the column vectors are **not** necessarily eigenvectors. Observe that

$$AP_1 = P_1 \begin{bmatrix} \lambda_1 & B \\ 0 & A_1 \end{bmatrix}$$

for some B, A. We can rewrite this as

$$P_1^T AP_1 = \begin{bmatrix} \lambda_1 & B \\ 0 & A_1 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ B^T & A_1^T \end{bmatrix}$$

This means that $B = 0$, $B^T = 0$, and $A_1 = A_1^T$. Thus, we have $P_1^T AP_1 = \begin{bmatrix} \lambda_1 & 0 \\ 0 & A_1 \end{bmatrix}$ where A_1 is a symmetric $n-1 \times n-1$ matrix.

By induction, since A_1 is an $n-1 \times n-1$ symmetric matrix, there exists an $n-1 \times n-1$ orthogonal matrix Q such that $Q^T A_1 Q = D_1$

Let $P_2 = \begin{bmatrix} 1 & 0 \\ 0 & Q \end{bmatrix}$. The matrix P_2 is orthogonal and so $P_1 P_2$ will also be orthogonal as products of orthogonal matrices are also orthogonal.

Let $P_1 P_2 = U$. We have

$$\begin{aligned} U^T A U = (P_1 P_2)^T A (P_1 P_2) = P_2^T P_1^T A P_1 P_2 = P_2^T (P_1^T A P_1) P_2 \\ = \begin{bmatrix} 1 & 0 \\ 0 & Q^T \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & A_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & Q \end{bmatrix} \\ = \begin{bmatrix} \lambda_1 & 0 \\ 0 & Q^T A_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & Q \end{bmatrix} \\ = \begin{bmatrix} \lambda_1 & 0 \\ 0 & Q^T A_1 Q \end{bmatrix} \\ = \begin{bmatrix} \lambda_1 & 0 \\ 0 & D_1 \end{bmatrix} \end{aligned}$$

and $\begin{bmatrix} \lambda_1 & 0 \\ 0 & D_1 \end{bmatrix}$ is diagonal! This completes the inductive step. Therefore, any symmetric matrix A is orthogonally diagonalizable. Thus, (3) $\Leftrightarrow$ (2), and we have proven the spectral theorem! □

Now we understand that a matrix can be orthogonally diagonalized if and only if it is a symmetric matrix. Let's look at a method for finding an orthogonal diagonalization. This will rely on Lemma 2.7 which will allow us to prove Lemma 2.8 which will be crucial to orthogonally diagonalizing a symmetric matrix.

Lemma 2.7. *If A is an $n \times n$ symmetric matrix, then*

$$(A\boldsymbol{x}) \cdot \boldsymbol{y} = \boldsymbol{x} \cdot (A\boldsymbol{y})$$

for all columns $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbb{R}^n$

Proof. We have $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y}$ so if $A = A^T$ then $(A\mathbf{x}) \cdot \mathbf{y} = (A\mathbf{x})^T \mathbf{y} = \mathbf{x}^T A^T \mathbf{y} = \mathbf{x}^T A \mathbf{y} = \mathbf{x} \cdot (A\mathbf{y})$ □

This implies the following statement.

Lemma 2.8. *If A is a symmetric matrix, then the eigenvectors of A corresponding to distinct eigenvalues are orthogonal.*

Proof.

Let $A\mathbf{x} = \lambda\mathbf{x}$ and $A\mathbf{y} = \mu\mathbf{y}$ where $\lambda \neq \mu$.

Then $\lambda(\mathbf{x} \cdot \mathbf{y}) = (\lambda\mathbf{x}) \cdot y = (A\mathbf{x}) \cdot \mathbf{y}$. By Lemma 2.7 $(A\mathbf{x}) \cdot \mathbf{y} = \mathbf{x} \cdot (A\mathbf{y})$

Therefore, $\lambda(\mathbf{x} \cdot \mathbf{y}) = \mu(\mathbf{x} \cdot \mathbf{y})$ so $(\lambda - \mu)(\mathbf{x} \cdot \mathbf{y}) = 0$. It follows that $\mathbf{x} \cdot \mathbf{y} = 0$ since $\lambda \neq \mu$ □

We can now describe a simple algorithm to orthogonally diagonalize any symmetric matrix A.

Algorithm 2.9. Orthogonal diagonalization of a symmetric matrix algorithm

- Step 1) Find the eigenvalues λ_i for A.
- Step 2)For each λ_i, if its corresponding eigenspace E_{λ_i} has rank > 1, use the use the Gram-Schmidt process to construct an orthogonal basis.
- Step 3) Take the union of all the bases for each eigenspace E_{λ_i}. By Lemma 2.8, this set will be orthogonal and form a basis for $\mathbb{R}^n$.

We can now also define the spectral decomposition of a symmetric matrix A.

Definition 2.10. For a symmetric $n \times n$ matrix A, we define a **spectral decomposition** of A as being a sum of the form

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

where $P = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n]$ is an orthogonal set of unit eigenvectors, and $\lambda_1, \lambda_2, ..., \lambda_n$ are the eigenvalues of A corresponding to P. The spectral decomposition is in fact found by orthogonally diagonalizing A.

$$A = PDP^T = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n] \begin{bmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & .... & \lambda_n \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ ... \\ \mathbf{u}_n^T \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1\mathbf{u}_1 & \lambda_2\mathbf{u}_2 & ... & \lambda_n\mathbf{u}_n \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ ... \\ \mathbf{u}_n^T \end{bmatrix} = \lambda_1\mathbf{u}_1\mathbf{u}_1^T + \lambda_2\mathbf{u}_2\mathbf{u}_2^T + ...\lambda_n\mathbf{u}_n\mathbf{u}_n^T$$

.

2.2. Singular Value Decomposition

The singular value decomposition (SVD) is a decomposition that exists for any $m \times n$ matrix A. It can be viewed as a generalization of the spectral theorem and we will make use of this using the following definition.

Theorem 2.11. *Let A be any $m \times n$ matrix with rank r. Then, $A = U\Sigma V^T$ where U is an $m \times m$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and Σ is an $m \times n$ matrix such that*

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

where D is a $r \times r$ diagonal matrix. The remaining $m - r$ rows and $n - r$ columns of Σ will be 0. D will be the first r non-zero singular values of A, $(\sigma_1, \ldots, \sigma_r)$, such that

$$\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$$

*We call $A = U\Sigma V^T$ a **singular value decomposition** of A.*

The power of the singular value decomposition is that it exists for any matrix without restrictions. Because of this, the applications of the singular value decomposition are extremely powerful for data analysis.

Proof. First, for any $m \times n$ matrix A, the matrix $A^T A$ will be a $n \times n$ symmetric matrix

$$(A^T A)^T = A^T A^{TT} = A^T A$$

We can therefore apply the Spectral Theorem to $A^T A$

$$V^T A^T A V = D$$

where the columns of V $[\mathbf{v}_1 \ldots \mathbf{v}_n]$ form an orthogonal basis of eigenvectors for $A^T A$. It can be shown that the set $\{A\mathbf{v}_i | A\mathbf{v}_i \neq \mathbf{0}\}$ forms an orthogonal basis for the column space of A.

If $i \neq j$, we have

$$(A\mathbf{v}_i)^T (A\mathbf{v}_j) = \mathbf{v}_i^T A^T A \mathbf{v}_j = \mathbf{v}_i^T (\lambda_j \mathbf{v}_j) = \lambda_j (\mathbf{v}_i \cdot \mathbf{v}_j) = 0$$

since $\mathbf{v}_i \cdot \mathbf{v}_j = 0$.

We can order the vectors so that $A\mathbf{v}_i \neq \mathbf{0}$ for $1 \leq i \leq r$ and $A\mathbf{v}_i = \mathbf{0}$ for $i > r$.

Secondly, we can write any vector $\mathbf{x} \in \mathbb{R}^n$ as a linear combination of $\mathbf{v}_1, ..., \mathbf{v}_n$. If

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

then

$$A\mathbf{x} = A(c_1 \mathbf{v}_1 + ... + c_1 \mathbf{v}_n) = Ac\mathbf{v}_i + ... + Ac\mathbf{v}_r + 0 + ...0$$

Therefore, $A\mathbf{v}_i, ..., A\mathbf{v}_r$ is a basis for the column space of A

If we now define

$$\mathbf{u}_i = \frac{1}{||A\mathbf{v}_i||} A\mathbf{v}_i$$

for $0 \leq i \leq r$, then $\{\mathbf{u}_i, \ldots, \mathbf{u}_r\}$ will be an orthonormal basis for the column space of A.

Let $\sigma_i = ||A\mathbf{v}_i||$. Then $A\mathbf{v}_i = \sigma_i \mathbf{u}_i$. We can use the Gram-Schmidt process to extend $\{\mathbf{u}_i, \ldots, \mathbf{u}_r\}$ to , an orthonormal basis $\{\mathbf{u}_i, \ldots, \mathbf{u}_r, \ldots, \mathbf{u}_m\}$ for $\mathbb{R}^m$. Let $U = [\mathbf{u}_1, ..., \mathbf{u}_m]$. Let D be a diagonal matrix containing σ_i for $0 \leq i \leq r$ and 0s along the diagonal. Observe that

$$U\Sigma = [\mathbf{u}_1\mathbf{u}_2 \ldots \mathbf{u}_m] \begin{bmatrix} \sigma_1 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 & 0 & \ldots & 0 \\ \ldots & & & & & & \\ 0 & 0 & \ldots & \sigma_r & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ \ldots & & & & & & \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \end{bmatrix} = [\sigma_1\mathbf{u}_1 \ldots \sigma_r\mathbf{u}_r\mathbf{0} \ldots \mathbf{0}] = AV$$

□

Let's define some terms used in the SVD.

Definition 2.12. The columns of U are called **left singular vectors** of A. The columns of V are called **right singular vectors** of A.

Lemma 2.13. *Left singular vectors of A are eigenvectors for AA^T and right singular vectors of A are the eigenvectors for A^TA*

Proof. Let $A = U\Sigma V^T$ be a singular value decomposition. Then

$$A = U\Sigma V^T, AA^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T$$

Therefore, $AA^TU = UD$ where $D = \Sigma\Sigma^T = \Sigma^L$. Thus the columns of U form an orthogonal eigenbasis for A^TA. If we apply the singular value decomposition to A^TA, we get

$$A^TA = V\Sigma U^T U\Sigma V^T = V\Sigma\Sigma^T V^T$$

Therefore, V is equal to the orthogonal basis of eigenvectors for A^TA. □

Lemma 2.13 displays how important symmetric matrices are to the SVD. We could think of the singular value decomposition as just orthogonally diagonalizing two different matrices.

Multiplication by an orthogonal matrix will always preserve the length of a vector and can be thought of as a rotation or reflection. Multiplication by a diagonal matrix always scales the components of the vector by the entries along its diagonal. Therefore, the singular value decomposition tells us that any matrix can be decomposed into a product of a rotation/reflection, a strength, and another rotation/reflection.

Lemma 2.14. *The column space is spanned by the first r columns of U. The row space is spanned by the first r columns of V, or the first r rows of* V^T

Proof. If we take a SVD of a matrix A, we will have $A = U\Sigma V^T$. V^T is an invertible matrix and therefore can be broken up as a product of elementary matrices. This action will not change the column space. Therefore, $U\Sigma V^T$ must have the same column space as $U\Sigma$. Σ is a diagonal matrix where the first r diagonal entries are non-zero. Therefore, only the first r columns of the product $U\Sigma$ will be non-zero. Because U is orthogonal and therefore invertible, the first r columns of U will also be independent, and therefore span the column space for A. The row space of A is equal to the column space of A^T. If $A = U\Sigma V^T$, then $A^T = V\Sigma^T U^T = V\Sigma U^T$. We can apply a similar argument for the row space. □

Lemma 2.15. *The last* $n-r$ *columns of V or the last* $n-r$ *rows of* V^T *span the null space of A. The last* $m-r$ *columns of U span the null space of* A^T.

Proof. For any right singular vector $\mathbf{v}_i$ it is true that

$$A\mathbf{v}_i = U\Sigma V^T\mathbf{v}_i = U\Sigma\mathbf{e}_i = U\sigma_i\mathbf{e}_i. \tag{1}$$

If $i > r$, then $\sigma_i = 0$ and so $A\mathbf{v}_i = \mathbf{0}$. Therefore, the span of the last $n-r$ columns of V will be contained in the null space. Since these columns are a subset of the invertible matrix V, they will be independent. Therefore, they will span the null space since the dimension of the null space is $n-r$.

For the second statement, observe that $A^T = V\Sigma U^T$. The same argument can be applied with U in place of V to prove that the last $m-r$ columns of U span the null space of A^T. □

See below an example of constructing an SVD.

Example 2.16. Let's find the SVD of $\begin{bmatrix} 5 & 4.6 \\ 2.5 & 3 \end{bmatrix}$

- Step 1) Orthogonally Diagonalize A^TA

$$\begin{bmatrix} 5 & 4.6 \\ 2.5 & 3 \end{bmatrix}^T \begin{bmatrix} 5 & 4.6 \\ 2.5 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} -0.713395 & -0.700762 \\ -0.700762 & 0.713395 \end{bmatrix} \begin{bmatrix} 61.2099 & 0 \\ 0 & .20001 \end{bmatrix} \begin{bmatrix} -0.713395 & -0.700762 \\ -0.700762 & 0.713395 \end{bmatrix}^T$$

Note, that $\begin{bmatrix} 61.2099 & 0 \\ 0 & .20001 \end{bmatrix} = \begin{bmatrix} \sigma_0^2 & 0 \\ 0 & \sigma_1^2 \end{bmatrix}$

- Step 2) Set up V and Σ

$$V = \begin{bmatrix} -0.713395 & -0.700762 \\ -0.700762 & 0.713395 \end{bmatrix}, \Sigma = \begin{bmatrix} \sqrt{61.2099} & 0 \\ 0 & \sqrt{.20} \end{bmatrix} = \begin{bmatrix} 7.823 & 0 \\ 0 & .4476 \end{bmatrix}$$

- Step 3) Construct U

$$\mathbf{u}_1 = \frac{1}{7.823} \begin{bmatrix} 5 & 4.6 \\ 2.5 & 3 \end{bmatrix} \begin{bmatrix} -.713395 \\ -.700762 \end{bmatrix} = \frac{1}{7.823} \begin{bmatrix} -6.79048 \\ -3.88577 \end{bmatrix} = \begin{bmatrix} -.86794 \\ -.496669 \end{bmatrix}$$

$$\mathbf{u}_2 = \frac{1}{.44736} \begin{bmatrix} 5 & 4.6 \\ 2.5 & 3 \end{bmatrix} \begin{bmatrix} -0.700762 \\ 0.713395 \end{bmatrix} = \frac{1}{.44736} \begin{bmatrix} -0.222193 \\ 0.38828 \end{bmatrix} = \begin{bmatrix} -0.496669 \\ 0.86794 \end{bmatrix}$$

Thus we have

$$A = \begin{bmatrix} -.86794 & -0.496669 \\ -.496669 & 0.86794 \end{bmatrix} \begin{bmatrix} 7.823 & 0 \\ 0 & .4476 \end{bmatrix} \begin{bmatrix} -0.713395 & -0.700762 \\ -0.700762 & 0.713395 \end{bmatrix}$$

One important factor is that the methods that we use to find the SVD of a matrix by hand will be often very different for how a computer will compute the SVD. The study of numerical linear algebra often surrounds the most efficient computational algorithms to solve or closely estimate decompositions like the SVD.

2.3. Quadratic Forms

We can now use the quadratic form to describe any any function's effect on any vector with a symmetric matrix. This concept will be extremely important when we examine the applications of the SVD; the quadratic form is often used to map simplify datasets.

Definition 2.17. A **quadratic form** on $\mathbb{R}^n$ is a function Q defined on $\mathbb{R}^n$ whose value at a vector $\mathbf{x}$ in $\mathbb{R}^n$ can be computed by an expression of the form $Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x}$ where A is an $n \times n$ symmetric matrix. The matrix A is called the **matrix of the quadratic form**.

The quadratic form helps us understand any vector $\mathbf{x}$ in $\mathbb{R}^n$ as a product of its transpose, an $n \times n$ symmetric matrix A, and itself.

When $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, the quadratic form, $Q(\mathbf{x})$ can also be written as

$$\sum_{i,j}^{n} a_{ij} x_i x_j = [x_1, ..., x_n] \begin{bmatrix} a_{11} & ... & a_{1n} \\ ..... & .... & .... \\ a_{n1} & ... & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$\begin{aligned} &= a_{11}x_1^2 + a_{12}x_1x_2 + \ldots + a_{1n}x_1x_n \\ &+ a_{21}x_2x_1 + a_2 2x_2^2 + \ldots + a_{2n}x_2x_n \\ &+ \ldots + \ldots + \ldots + \ldots \\ &+ \ldots + \ldots + \ldots + \ldots \\ &+ \ldots + \ldots + \ldots + \ldots \\ &+ a_{n1}x_nx_1 + a_{n2}x_nx_2 + \ldots + a_{nn}x_n^2 \end{aligned}$$

Example 2.18. Let $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. Compute Let's find the SVD of $\mathbf{x}^T A\mathbf{x}$ for $\begin{bmatrix} 3 & -2 \\ -2 & 7 \end{bmatrix}$

$$\mathbf{x}^T A\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \begin{bmatrix} 3 & -2 \\ -2 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix} \begin{bmatrix} 3x_1 - 2x_2 \\ -2x_1 + 7x_2 \end{bmatrix}$$

$$= x_1(3x_1 - 2x_2) + x_2(-2x_1 + 7x_2) = 3x_1^2 - 4x_1x_2 + 7x_2^2$$

Definition 2.19. If $\mathbf{x}$ represents a variable vector in $\mathbb{R}^n$, then a **change of variable** is an equation of the form $\mathbf{x} = P\mathbf{y}$, or $\mathbf{y} = P^{-1}\mathbf{x}$ where P is an invertible matrix and $\mathbf{y}$ is a new variable vector in $\mathbb{R}^n$. More precisely, $\mathbf{y}$ is the coordinate vector of $\mathbf{x}$ relative to the basis of $\mathbb{R}^n$ determined by the columns of P.

Now, if we plug in the change of variable equation into the quadratic form we have

$$\mathbf{x}^T A\mathbf{x} = (P\mathbf{y})^T A(P\mathbf{y}) = \mathbf{y}^T P^T AP\mathbf{y} = \mathbf{y}^T (P^T AP)\mathbf{y}$$

Thus, the quadratic form can also be represented in a way with no "cross" terms as $\mathbf{y}^T D\mathbf{y}$ will be a diagonal matrix. Furthermore, if we choose P, such that P is the orthogonal matrix, made up of an orthogonal basis of eigenvectors for $\mathbb{R}^n$. Then plugging in $\mathbf{x} = P\mathbf{y}$ gives us

$$\mathbf{x}^T A\mathbf{x} = \mathbf{y}^T D\mathbf{y} = \lambda_1 \mathbf{y}_1^2 + \lambda_2 \mathbf{y}_2^2 + \ldots + \lambda_n \mathbf{y}_n^2$$

where $\lambda_1, \lambda_2, ..., \lambda_n$ are the eigenvalues corresponding to the eigenvectors that form the columns of P.

Example 2.20. Make a change of variable that transforms the quadratic form to have no "cross terms" for the matrix

$$\begin{bmatrix} 1 & -4 \\ -4 & -5 \end{bmatrix}$$

- Orthogonally diagonalize A. Its eigenvalues are $\lambda = 3$ and $\lambda = -7$. Its corresponding unit eigenvectors are

$$\lambda = 3 : \begin{bmatrix} 2\sqrt{5} \\ -1\sqrt{5} \end{bmatrix} \lambda = -7 : \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{bmatrix}$$

Construct an orthonormal basis for $\mathbb{R}^2$

$$P = \begin{bmatrix} 2\sqrt{5} & \frac{1}{\sqrt{5}} \\ -1\sqrt{5} & \frac{2}{\sqrt{5}} \end{bmatrix} D = \begin{bmatrix} 3 & 0 \\ 0 & -7 \end{bmatrix}$$

- $\mathbf{x} = P\mathbf{y}$ where $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$

- Thus

$$\begin{aligned} \mathbf{x}^T A\mathbf{x} = x_1^2 - 8x_1x_2 - 5x_2^2 = (P\mathbf{y})^T A(P\mathbf{y}) \\ = \mathbf{y}^T P^T AP\mathbf{y} = \mathbf{y}^T D\mathbf{y} = 3y_1^2 - 7y_2^2 \end{aligned}$$

This means that for any vector $\mathbf{x}$ in $\mathbb{R}^n$, if we orthogonally diagonalize any $n \times n$ symmetric matrix A, such that

$$A = PDP^T$$

we can represent $\mathbf{x}$ as the product of the transpose of the coordinate vector of $\mathbf{x}$ relative to P, the symmetric matrix A, and the coordinate vector of $\mathbf{x}$ relative to P. The value of this quadratic form for $\mathbf{x} \neq \mathbf{0}$ is completely determined by what symmetric matrix A we choose.

Lemma 2.21. *For any $n \times n$ symmetric matrix A, the quadratic form $Q(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x}$. We have:*

a. $Q(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$ if and only if the eigenvalues of A are all positive.

b. $Q(\boldsymbol{x}) < 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$ if and only if the eigenvalues of A are all negative

c. $Q(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$ if and only if A has both positive and negative eigenvalues

Proof. First note that we proved for any symmetric matrix A, there exists an orthogonal change of matrix variable $\mathbf{x} = P\mathbf{y}$ such that

$$Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} = \mathbf{y}^T D\mathbf{y} = \lambda_1 y_1^2 + \lambda_2 y_2^2 + ... + \lambda_n y_n^2$$

Since P is invertible, there is a one-to-one correspondence between all nonzero $\mathbf{x}$ and nonzero $\mathbf{y}$. Therefore, the values of $Q(\mathbf{x})$ for $\mathbf{x} \neq \mathbf{0}$ coincide with the values of the above expression on the right. Therefore, $Q(\mathbf{x})$ is determined by the signs of the eigenvalues $(\lambda_1, \lambda_2, ...\lambda_n)$.

Consider the relationship between the SVD and the quadratic form. For $\mathbf{x} \in \mathbb{R}^n$ and the $m \times n$ matrix A we have

$$Q_A(\mathbf{x}) = \mathbf{x}^T A^T A\mathbf{x} = (A\mathbf{x})^T A\mathbf{x} = (A\mathbf{x})^T(A\mathbf{x}) = ||A\mathbf{x}|| \geq \mathbf{0} \tag{2}$$

for all $\mathbf{x} \in \mathbb{R}^n$. See that if $A = U\Sigma V^T$, then

$$S = A^T A = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma^T U^T U\Sigma V^T = V\Sigma^T \Sigma V^T = VDV^T$$

Therefore, the maximum and minimum values for $Q_S(\mathbf{x})$ for $\mathbf{x}$ such that $||\mathbf{x}|| = 1$ are the same as for $Q_0(\mathbf{x})$ since V is orthogonal implies that $||\mathbf{x}|| = 1 \Leftrightarrow ||V\mathbf{x}|| = 1$. This the maximum σ_i^2 is the maximizes the quadratic form when $||\mathbf{x}|| = 1$ and the minimum σ_i^2 minimizes the quadratic form when $||\mathbf{x}|| = 1$. □

CHAPTER 3. APPLICATIONS OF THE SINGULAR VALUE DECOMPOSITION

Chapter 3 focuses on applications of the SVD. We first introduce the concept of low-rank approximation. By low-rank, we mean that our approximation matrix has a lower rank than the original matrix A. This approximation relies heavily on the fact that the rank of any matrix is equal to the number of non-zero singular values. We show how to compute such an approximation using the SVD. We give an example using a real image quantifying how much memory we can save using this approximation. We also introduce the idea of a spectral norm and show that our rank-k approximation is the closest rank-k approximation with respect to the spectral norm. These ideas draw heavily from Section 4.7 in [3].

We then introduce the concept of principle component analysis, which allows us to use the SVD in the context of statistics. By taking any data matrix, and putting it in mean-deviation form, a concept that will be explained later, we can create the covariance matrix. If we denote the matrix in mean-deviation form as B, the covariance matrix will be equal to $\frac{1}{N-1}BB^T$, a symmetric matrix! Every entry in the covariance matrix S_{ij} is equal to the covariance between variables i and j. These statistical properties can be further analyzed by orthogonally diagonalizing the covariance matrix S. This can be used for PCA approximation as well as data-analysis.

3.1. Low-Rank Approximation

Lemma 3.1. *The rank of any matrix A is equal to the number of non-zero singular values of A.*

Proof. Given an $m \times n$ matrix A, we can find a singular value decomposition, $A = U\Sigma V^T$. Both U and V are orthogonal matrices and therefore also invertible matrices. Further note, that multiplication by invertible matrices preserves rank. Since $A = U\Sigma V^T$, we have that $\text{rank}(A) = \text{rank}(U\Sigma V^T) = \text{rank}(\Sigma V^T) = \text{rank}(\Sigma)$. The rank of a diagonal matrix is equal to the number of non-zero entries. Therefore, the rank of A is equal to the rank of Σ which is equal to the number of non-zero singular values. □

Definition 3.2. Define

$$\hat{A}(k) = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

where σ_i are the singular values for the matrix A and $\mathbf{u}_i$ and $\mathbf{v}_i$ are the left and right singular vectors of A, respectively.

Lemma 3.3. *$\hat{A}(k)$ has rank k.*

Proof. We have

$$\hat{A}(k) = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T = \begin{bmatrix} \mathbf{u}_1 & \dots & \mathbf{u}_k \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & ... & 0 \\ 0 & \sigma_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & .... & \sigma_k \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_k^T \end{bmatrix} = U_k \Sigma_k V_k^T$$

$T_{U_k}(\mathbf{x}) = U_k\mathbf{x}$ is a one-to-one transformation, just as $T_{V_k^T}(\mathbf{x}) = \mathbf{x}V_k^T$ is a one-to-one transformation. Applying a one-to-one transformations preserve the rank of a matrix A. We have

$$\text{rank}(\hat{A}(k)) = \text{rank}(U_k\Sigma_k V_k^T)$$

$$rank(\hat{A}(k)) = rank(\Sigma_k V_k^T) = rank(\Sigma_k)$$

Therefore $\text{rank}(\hat{A}(k))$ is equal to the number of non-zero singular values of Σ_k which is equal to k if $k \leq \text{rank}(A)$.s □

We will eventually prove that this approximation is the best rank-k approximation for A. However, let's first give a visual example of the effectiveness of the $\hat{A}(k)$ approximation. It is particularly helpful with image compression. If we think of an image as a $m \times n$ matrix, we can represent the "essence" of the image with a much lower rank than the original one. See the figures below, which shows different low-rank approximations of a grey-scale image of me hitting using the approximation $\hat{A}(k)$.

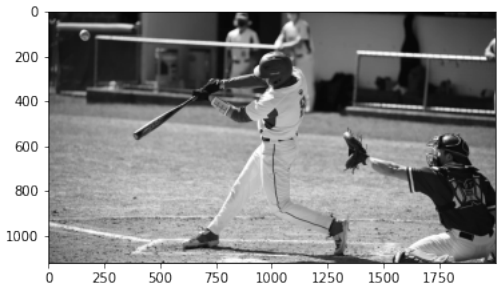


Figure 2. Rank 1 Approximation

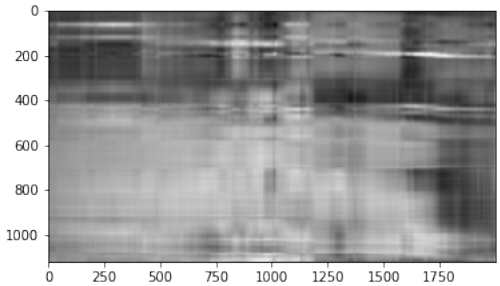


Figure 3. Rank 5 Approximation

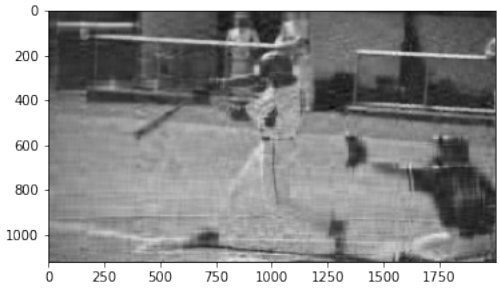


Figure 4. Rank 25 Approximation

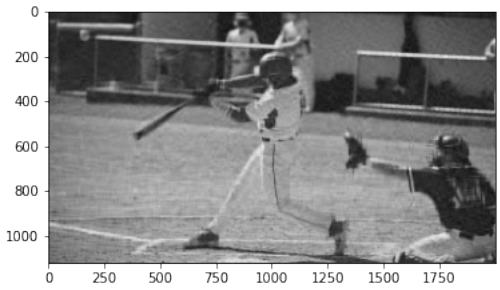


Figure 5. Rank 50 Approximation

Note that storing the original image would take $1125 * 2000$ values. However, storing a rank k approximation would only take $k \times (1125 + 2000 + 1)$ values. We can see for our rank 50 approximation, it is nearly indistinguishable from the original image to the human eye. Furthermore, it would take $50(1125 + 1200 + 1) = 116,300$ values to store it which is much less then $1125 * 2000 = 2,250,000$ to store the original image.

It will be shown that the rank k approximation $A(\hat{k})$ is the closest rank k approximation for the matrix A. First, we introduce a notation of size for matrices called the **spectral norm**.

Definition 3.4. [3, Section 10.2] The **spectral norm** of an $m \times n$ matrix A is defined

$$||A|| = max_{\mathbf{x}} \frac{||A\mathbf{x}||}{||\mathbf{x}||} \mathbf{x} \in \mathbb{R}^n.$$

The spectral norm determines the maximum scaling factor for $\mathbf{x}$ when multiplied by A.

Lemma 3.5. *Let M and m be the maximum and minimum singular values for a matrix A respectively. Then*

- *1)* $m \leq \dfrac{||A\boldsymbol{x}||}{||\boldsymbol{x}||} \leq M$ *for all* $\boldsymbol{x} \in R^n$.

- *2) The spectral norm of* A *is equal to* M.

Proof. We can restrict to $||\mathbf{x}|| = 1$. This is because for any scalar c we have $|c\mathbf{x}| = |c||\mathbf{x}|$ and $|A(c\mathbf{x})| = C|A\mathbf{x}| = |c||A\mathbf{x}|$. This means our quotient under scaling is $\dfrac{|A(c\mathbf{x})}{|c\mathbf{x}|} = \dfrac{|c||A\mathbf{x}|}{|c||\mathbf{x}|} = \dfrac{|A\mathbf{x}|}{|\mathbf{x}|}$. Thus, we can restrict to $|\mathbf{x}| = 1$ when looking for the maximum and minimum value. Further note that multiplying by an orthogonal matrix preserves length. If we write $A = V\Sigma U^T$, then we have

$$||A|| = \max_{||\mathbf{x}||=1} \frac{||V\Sigma U^T\mathbf{x}||}{1} = \max_{||\mathbf{x}||=1} ||\Sigma U^T\mathbf{x}|| = \max_{||\mathbf{x}||=1} ||\Sigma\mathbf{x}||$$

$$= \max_{||\mathbf{x}||=1} \sqrt{(\sigma_1 x_1^2 + \ldots \sigma_n x_n^2)}$$

The singular values are greater than or equal to 0. Therefore $M \geq m \geq 0$ Let $M = \max\sigma_i$ and $m = \min\sigma_i$. If $||\mathbf{x}|| = 1$, then

$$m = \sqrt{m\sum_{i=1}^{n} x_i^2} \leq \sqrt{\sum_{i=1}^{n} \sigma_i x_i^2} \leq \sqrt{M\sum_{i=1}^{n} x_i^2} = M$$

If we order the singular values so that $M = \sigma_i$, the greatest singular value, the maximum is attained at $\mathbf{x} = \mathbf{e}_1$. The maximum is achieved with the unit vector corresponding to the largest singular value of A, therefore, the spectral norm is equal to the largest singular value σ_1. □

Theorem 3.6. *Consider an* $m \times n$ *matrix* A *of rank* r. *For all* $m \times n$ *matrices* B *of rank* $k \leq r$, *we have*

$$|||A - \hat{A}(k)|| = \sigma_{k+1} \leq ||A - B||$$

Therefore, $\hat{A}(k)$ *is the best rank-k approximation for* A.

Proof. The matrix $A - \hat{A}(k)$ is a matrix containing the sum of the remaining rank-1 matrices. By Definition 3.4,

$$A - \hat{A}(k) = \sum_{i=k+1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

We see that $\{\sigma_{k+1}, \ldots, \sigma_n\}$, σ_{k+1} is the biggest singular value, so by Lemma 3.5

$$||A - \hat{A}(k)|| = \sigma_{k+1}$$

Let B be any matrix such that $\text{rank}(B) = k$. We can choose a basis, $\mathbf{x}_1, \ldots \mathbf{x}_k$, for $Col(B)$. Let $X = [\mathbf{x}_1, \ldots \mathbf{x}_k]$. Any vector in the column space can be written as a linear combination of these basis vectors. Since the columns of B are in the column

space, we can write $B = XY$ where the columns of Y are the coordinate vectors of the columns of B with respect to this basis.

Consider $Y\mathbf{v}_i \in \mathbb{R}^k$ for $1 \leq i \leq k+1$. Since $k+1 > k$, the vectors will be linearly dependent, and so there exists $w_1, \ldots, w_{k+1} \in \mathbb{R}^k$ such that

$$\sum_{i=1}^{k+1} w_i Y\mathbf{v}_i = \mathbf{0}$$

where $w_1, \ldots, w_{k+1}$ are not all 0. Let $\mathbf{w} = \sum_{i=1}^{k+1} w_i \mathbf{v}_i$. Rescale so that $||\mathbf{w}||^2 = w_1^2 + \ldots + w_{k+1} = 1$. We have $B\mathbf{w} = XY\mathbf{w} = XY\sum_{i=1}^{k+1} w_i\mathbf{v}_i = X\sum_{i}^{k+1} w_i Y\mathbf{v}_i = \mathbf{0}$. Using the fact that V is orthogonal, we have

$$A\mathbf{w} = U\Sigma V^T\mathbf{w} = U\sum_{1}^{k+1} w_i \Sigma V^T\mathbf{v}_i = U\sum_{i=1}^{k+1} w_i\sigma_i\mathbf{e}_i$$

where the vectors $\mathbf{e}_i$ are the standard basis vectors for $\mathbb{R}^m$.

Using Definition 3.4, and the fact U is orthogonal, we have

$$\begin{aligned}\|A-B\|^2 \geq \|(A-B)\mathbf{w}\|^2 = \|A\mathbf{w}\|^2 &= \left\|U\sum_{i=1}^{k+1} w_i\sigma_i\mathbf{e}_i\right\|^2 \\ &= \left\|\sum_{i=1}^{k+1} w_i\sigma_i\mathbf{e}_i\right\|^2 \\ &= \sum_{i=}^{k+1}(w_i\sigma_i)^2 \geq (\sum_{i=1}^{k+1} w_i^2)\sigma_{k+1}^2 = \sigma_{k+1}^2.\end{aligned}$$

Therefore, there exists no matrix B_k with $rk(B) = k$ such that

$$||A-B|| < ||A-\hat{A}(k)||$$

Thus $A(k)$ will be the best rank-k approximation for A. □

3.2. Principal Component Analysis

One common application of singular value decomposition is principal component analysis (PCA). PCA allows us to reduce and simplify sets of data based on their covariance and variance across the variables of the data set. The relies heavily on computing the covariance matrix, B, from the data matrix A.

Input Data Principle component analysis takes a matrix of observations as input.

Example 3.7. Suppose we took the measurement of height (inches), weight (pounds), and age (years) of four different adults in a random sample from a population. Our observation vectors would be

$$\mathbf{x}_1 = \begin{bmatrix} 72 \\ 180 \\ 25 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 65 \\ 140 \\ 68 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 70 \\ 165 \\ 45 \end{bmatrix}, \mathbf{x}_4 = \begin{bmatrix} 74 \\ 200 \\ 36 \end{bmatrix}$$

which would give us a data-matrix

$$X = \begin{bmatrix} 72 & 65 & 70 & 74 \\ 180 & 140 & 165 & 200 \\ 25 & 68 & 45 & 36 \end{bmatrix}$$

Our data-matrix is a matrix where each column is one of the observation vectors. For an $m \times n$ matrix that PCA takes as input, we will have m measurements or variables and n different observations for each variable. Thus we have the matrix, where each observation vector X_k is a column

$$X = [\mathbf{x}_1 \mathbf{x}_2 .... \mathbf{x}_n] = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1} & \vdots & \vdots & x_{mn} \end{bmatrix}$$

Data Transforming Secondly, PCA requires us first to put the original input matrix X into **mean-deviation form**. Doing this requires computing the mean of each variable. We will then create a matrix $\hat{X}$, where instead of having our columns be the observation vectors $\mathbf{x}_k$, our columns are $\hat{\mathbf{x}}_k$. Let $\hat{\mathbf{x}}_k = \mathbf{x}_k - \mathbf{m}$ where $\mathbf{m}$ is the vector where each component m_i is the mean of variable i.

Example 3.8. $\mathbf{m} = \begin{bmatrix} 70.25 \\ 171.25 \\ 43.5 \end{bmatrix}$ since $mean_h = 70.25$, $mean_a = 43.5$, $mean_w = 171.25$.

Thus, our matrix $\hat{X}$ would be $\begin{bmatrix} 1.5 & -5.5 & -.5 & 3.5 \\ 8.75 & -31.25 & -6.25 & 28.75 \\ -18.5 & 24.5 & 1.5 & -7.5 \end{bmatrix}$

Let m_{rowj} be the mean of row $0 \leq j \leq m$. Our mean deviation matrix will be $\hat{X}$ where $\hat{X}$ is

$$[\hat{\mathbf{x}}_1 \hat{\mathbf{x}}_2 \dots \hat{\mathbf{x}}_n] = \begin{bmatrix} x_{11} - m_{row1} & x_{12} - m_{row1} & \dots & x_{1n} - m_{row1} \\ x_{21} - m_{row2} & x_{22} - m_{row2} & \dots & x_{2n} - m_{row2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1} - m_{rowm} & \vdots & \vdots & x_{mn} - m_{rowm} \end{bmatrix}$$

We call the $m \times n$ matrix S the the **(sample) covariance matrix** where

$$S = \frac{1}{N-1}\hat{X}\hat{X}^T$$

This is not only a symmetric matrix, but it also contains important information about the variance of the data. First note, that for matrix multiplication of two matrices AB, the matrix product AB_{ij}, will be

$$\sum_{k=1}^{n}(a_{ik})(b_{kj})$$

Secondly note that $(A^T)_{ij} = A_{ji}$. Thus, S_{ij} will be equal to

$$\frac{1}{n-1}([BB^T]_{ij}) = \frac{1}{n-1}\sum_{k=1}^{n}(B_{ik})((B^T)_{kj}) = \frac{1}{n-1}\sum_{k=1}^{n}(B_{ik})((B)_{jk})$$

which is equal to

$$\frac{1}{n-1}\sum_{k=1}^{n}(x_{ik} - m_{row1})(x_{jk} - m_{rowj})$$

This is equivalent to the covariance of variables i and j. Thus, each index S_{ij} of S is equal to the covariance of variables i and j!

Example 3.9. For our data-matrix X our covariance matrix S would be

$$\frac{1}{4-1}\hat{X}\hat{X}^T = \begin{bmatrix} 1.5 & -5.5 & -.5 & 3.5 \\ 8.75 & -31.25 & -6.25 & 28.75 \\ -18.5 & 24.5 & 1.5 & -7.5 \end{bmatrix}\begin{bmatrix} 1.5 & -5.5 & -.5 & 3.5 \\ 8.75 & -31.25 & -6.25 & 28.75 \\ -18.5 & 24.5 & 1.5 & -7.5 \end{bmatrix}^T$$

which is equal to

$$\frac{1}{3}\begin{bmatrix} 45 & 288.75 & -189.5 \\ 288.75 & 1918.75 & -1152.5 \\ -189.5 & -1152.5 & 1001 \end{bmatrix} = \begin{bmatrix} 15 & 96.25 & -63.1667 \\ 96.25 & 639.583 & -384.167 \\ -63.1667 & -384.167 & \frac{1001}{3} \end{bmatrix}$$

Well, what does the covariance represent? The covariance measures the joint-variability of two random variables. A positive covariance means that variables tend to be above or below their mean values at the same time, while a negative covariance means that one variables tends to above the mean while the other is below, and vice-versa.

Because our data matrix, B is an $m \times n$ matrix, consider the SVD of B

$$B = U\Sigma V^T$$

Furthermore, since $S = \frac{1}{n-1}\hat{X}\hat{X}^T$, we say that

$$S = \frac{1}{n-1}U\Sigma V^T V\Sigma^T U^T = \frac{1}{n-1}U\Sigma\Sigma^T U^T$$

Since $S = \hat{X}\hat{X}^T$ is symmetric, we have an orthogonal diagonalization of S

$$S = PDP^T = U\Sigma\Sigma^T U^T$$

The matrix U is made up of orthogonal eigenvectors of S and the eigenvalues of S are related to the singular values of X like

$$\lambda_d = \frac{\sigma_d^2}{n-1}$$

Example 3.10. The SVD of S in our example involving height, age, and weight is

$$U = \begin{bmatrix} -0.125805 & .12312 & .984385 \\ -.82414 & -.565327 & -.03461 \\ .55223 & -.815627 & .172589 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 908.437 & 0 & 0 \\ 0 & 129.913 & 0 \\ 0 & 0 & 12.9343 \end{bmatrix}$$

$$V^T = \begin{bmatrix} .0468423 & .829633 & .556341 \\ -.827098 & -.28008 & .487303 \\ .560102 & -.482975 & .673068 \end{bmatrix}^T$$

We can also observe that the total variance of the data matrix is equal to the trace of the covariance matrix.

Lemma 3.11. *The total variance of the data, is equal to the trace of the covariance matrix, $S = \frac{1}{n-1}XX^T$ which is equal to the sum of the eigenvalues of S*

Proof. Remember that the trace of a square matrix is equal to the sum of the elements across the diagonal. Note that for the covariance matrix S, the elements across the diagonal S_{ij} where $i = j$, is equal to the variance of each variable in the data. Also note that the trace of a matrix products is commutative; $Tr(AB) = Tr(BA)$. Since S is a symmetric matrix we have $Tr(S) = Tr(PDP^T) = Tr(DPP^T) = Tr(D)$. Thus, $Tr(S) = \sum_{i=1}^{n} \lambda_i$. Since, $Tr(S) = \sum_{i=1}^{n} \lambda_i = \sum_{i}^{n} S_{ii}$, it is true that the total variance across all variables of the data is equal to the sum of the eigenvalues of the covariance matrix. □

Since $S = PDP^T$, where P is an orthogonal matrix that forms a basis for R^m, any observation vector $\mathbf{X}_K$ can be written as $\mathbf{X}_k = P\mathbf{y}$ such that

$$\begin{bmatrix} x_{0k} \\ x_{1k} \\ ..... \\ x_{mk} \end{bmatrix} = [\mathbf{u}_1\mathbf{u}_2....\mathbf{u}_m] \begin{bmatrix} y_1 \\ y_2 \\ ..... \\ y_m \end{bmatrix}$$

where the new variables $y_1, ..., y_p$ are uncorrelated and are arranged in order of decreasing variance. Note that for each $\mathbf{X}_k$ it is true that $\mathbf{Y}_k = P^{-1}\mathbf{X}_k = P^T\mathbf{X}_k$

Each observation vector $\mathbf{X}_k$ will now receive a "new name", $\mathbf{Y}_k$ such that $\mathbf{X}_K = P\mathbf{Y}_k$, where $\mathbf{Y}_k$ is the coordinate vector of $\mathbf{X}_k$ with respect to the columns of P, as $\mathbf{Y}_k = P^{-1}\mathbf{X}_k = P^T\mathbf{X}_k$ for $k = 1, ..., m$.

We can also see that the covariance matrix of $\mathbf{Y}_1, ..., \mathbf{Y}_n$ is P^TSP.

Proof. Let $S' = \frac{1}{n-1}YY^T$ which means that $S' = \frac{1}{n-1}PX(PX)^T = P((\frac{1}{n-1})XX^T)P^T = PSP^T$ □

By definition, P^TSP will be a diagonal matrix and contains the eigenvalues of S on the diagonal arranged so that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p \geq 0$ where P is made up of the unit eigenvectors of S. Further note that because this matrix is diagonal, each variable will be uncorrelated; the co-variances for difference variables will be 0. Furthermore, we can also say that this new matrix has the same total variance as the original matrix S

Proof. Let $S' = YY^T$ we showed that this equals PSP^T. Thus $Tr(S') = Tr(USU^T) = Tr(S)$. Thus, S and S' will have the same trace and the same eigenvalues which summed together will equal to the total variance of the variables. □

We call the unit vectors $\mathbf{u}_1, ...\mathbf{u}_m$ of the covariance matrix S, the **principal components** of the data (in the matrix of observations). The **first principle component** is the eigenvector corresponding to the largest eigenvalue of S, the **second principal component** is the eigenvector corresponding to the largest eigenvalue and so in.

The first principal component $\mathbf{u}_1$ determines the new variable y_1 in the following way. Let $c_1, ..., c_m$ be the entries in $\mathbf{u}_1$. Since $\mathbf{u}_1^T$ is the first row of P^T, the equation* $Y = P^TX$ shows that

$$y_1 = \mathbf{u}_1^T X = c_1x_1 + c_2x_2 + ... + c_px_p.$$

3.3. Using PCA to Reduce the Dimension of Multivariate Data

Since $Y = P^TX$, and P^T is orthogonal, it will not change the trace of Y, and will therefore keep the total variance (the sum of the eigenvalues the same). Further note that since Y is diagonal, the covariance of different variables will be 0, and the variables will be uncorrelated. Thus, we can use PCA to create a new Data-matrix with the same total variance, but where the co-variance between each variable is 0. Thus we could use this matrix Y to create low-rank approximations of the data, which is not only accurate, but maintains the co-variance among different variables to be 0.

CHAPTER 4. NEURAL NETWORKS

Neural networks are machine-learning models which are inspired by how neurons fire in the brain to make decisions. Nevertheless, the actual difference between neural networks and the brain are numerous and distinct. Before we introduce their structure, let's consider why neural networks are important. Neural Networks are extremely powerful, particularly when it comes to image processing and pattern recognition. Their structure allows computers to effectively recognize patterns through automatic extraction of features of the input. These features can then be used for classification or prediction. Let's now introduce their structure.

4.1. The Components of a Neural Network

A neural network is composed of neurons and edges with the neurons usually organized in layers and the directed edges connecting neurons from one layer to the next. We can think of neurons as variables with assigned values which we calculate through "forward propagation" which will be defined later. They are also called activation units. We can think of edges as variables whose value indicates how strongly one neuron influences another. The weights will serve as a type of scalar to the neuron it recieves.

These edges' values will be used to define functions that take the values of the neurons in one layer and use these to define values in the next layer. There will be a pre-determined number of layers in the network and the activation units in the final layer will signify something about the data inputted into the neural network.

For example, suppose we have a data point with two variables and we want to classify the point as "on" or "off". Consider figure 6 which displays a neural network with one "hidden layer", the layers that do not contain input or output neurons and with three neurons inside this layer.

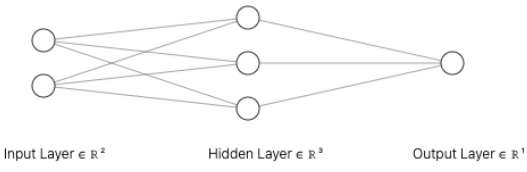
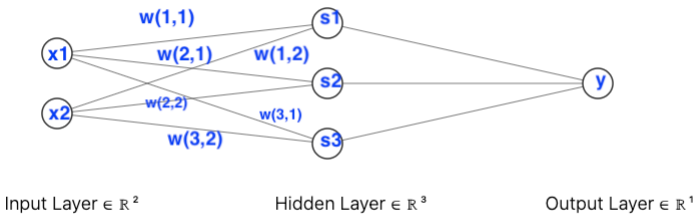


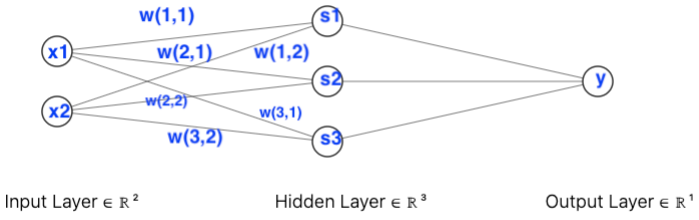
Figure 6. Structure of a Neural Network

The **depth** of the network is equal to the total number of layers in the network. Each layer will also have a **width** which is based on the number of neurons at each layer. We call the value of the edges that connect neurons to different layers, the **weights** of the network. The weights are used to define a function that uses one layer to define how one input neuron becomes another input neuron. We have x_1, x_2

as the input neurons, where w_{ij} represents the weights applied to the them. Also, s_1, s_2, s_3 are the neurons at the hidden layer, and y is the output neuron. Consider figure 4.1



How do these weights transform the neurons? The best way to understand this is by viewing a neural network as simply layers of matrix-vector multiplication composed together. If we have a data set the entire data set will usually be a data matrix X, where each vector is a data point. In our previous example, each data point would have two variables, x_1 and x_2. Thus, we can think of each layer of neurons as a vector. If a layer has 3 neurons, it would be represented as a vector with dimension 3. In a fully connected layer, there is an edge between each input neuron and each output neuron. In this case, we can represent the edges together as a matrix as well. We will call this the weight matrix. Thus, the action of the weights on the first layer becomes



$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \\ w_{31}x_1 + w_{32}x_2 \end{bmatrix}$$

which would then undergo another matrix multiplication to produce the output neuron y.

We previously only described the interactions between neurons and edges as matrix-transformation. However, neural networks will be made up of non linear

transformations. The goal of many neural networks is to identify complicated patterns to solve complicated problems. Having our functions limited to be linear functions would severely restrict the ability for neural networks to identify complicated patterns that will most likely **not** be linear. Therefore, at each layer we introduce, **non linear activation functions** which transform our linear functions into non linear functions. Consider the activation function σ as $\mathbb{R} \to \mathbb{R}$. Let $\sigma_{\mathbf{b}} : \mathbb{R}^n \to \mathbb{R}$ where $\mathbf{b} \in \mathbb{R}^n$. We now describe the interaction between neurons and edges as an non linear function where 4.1 becomes

$$\sigma_{\mathbf{b}}\left(\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \\ w_{31}x_1 + w_{32}x_2 \end{bmatrix}\right) = \sigma_{\mathbf{b}}(\begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \\ w_{31}x_1 + w_{32}x_2 \end{bmatrix}) = \begin{bmatrix} \sigma(w_{11}x_1 + w_{12}x_2 - b_1) \\ \sigma(w_{21}x_1 + w_{22}x_2 - b_2) \\ \sigma(w_{31}x_1 + w_{32}x_2 - b_3) \end{bmatrix}$$

where $\sigma_{\mathbf{b}}$ maps the dimension of the output neurons to the same dimension. Thus if we have $y_1, \dots y_n$ output neurons at each layer, we have

$$\sigma_b(\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}) = \begin{bmatrix} \sigma(y_1 - b_1) \\ \vdots \\ \sigma(y_n - b_n) \end{bmatrix}$$

Thus, equation 4.1 becomes

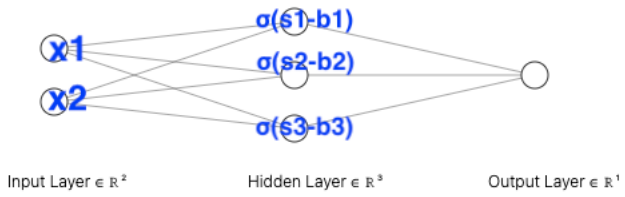


Figure 7. Non Linear Transformation

Example 4.1. Let's consider a helpful, but slightly unrealistic example. Suppose we have images representing two numbers: a 1 and a 0. A one would be a 3×3 matrix with values down the middle. A 0 will also be a 3×3 matrix but with values all the way across the perimeter. Let's consider that our data set has just a 1 and a 2.

$$1 = \begin{bmatrix} 0 & 5 & 0 \\ 0 & 8 & 0 \\ 0 & 1 & 0 \end{bmatrix}, 0 = \begin{bmatrix} 1 & 3 & 4 \\ 1 & 0 & 5 \\ 3 & 7 & 6 \end{bmatrix}$$

Note that if we vectorize both matrices, which is common in neural networks, the number in vector for will be

$$1 = \begin{bmatrix} 0 \\ 5 \\ 0 \\ 0 \\ 8 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad 0 = \begin{bmatrix} 1 \\ 3 \\ 4 \\ 1 \\ 0 \\ 5 \\ 3 \\ 7 \\ 6 \end{bmatrix}$$

Imagine we constructed a two-layer neural network, with 9 input neurons and one output neuron. See

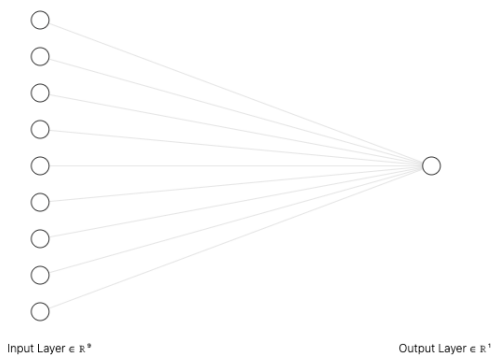


Figure 8. Neural Network Example

As explained later, the values of our weights will be randomly initialized. Nevertheless, their values would determine the value of the neurons in the final layer.

$$\sigma\left(\begin{bmatrix} 4 & 4 & 3 & 3 & 1 & 6 & 7 & 9 & 10 \end{bmatrix} \begin{bmatrix} 0 \\ 5 \\ 0 \\ 0 \\ 8 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}\right) = \sigma\left(\begin{bmatrix} 37 \end{bmatrix}\right)$$

$$\sigma\left(\begin{bmatrix}4 & 4 & 3 & 3 & 1 & 6 & 7 & 9 & 10\end{bmatrix}\begin{bmatrix}1\\3\\4\\1\\0\\5\\3\\7\\6\end{bmatrix}\right) = \sigma\left(\begin{bmatrix}205\end{bmatrix}\right)$$

Both the non linear activation function and the bias vector work in tandem to improve our network's ability to make accurate predictions on complicated problems. Applying the activation function first introduces this non linearity to our function, allowing our network to recognize more complex patterns. Secondly, together with the bias vector it helps normalize the values of neurons between a certain range. The bias vector helps determine the cut-off in how neurons will transitions between a certain range. For example, consider a commonly used activation function where each neuron is scaled to a value between 0 and 1.

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

It's graph looks like

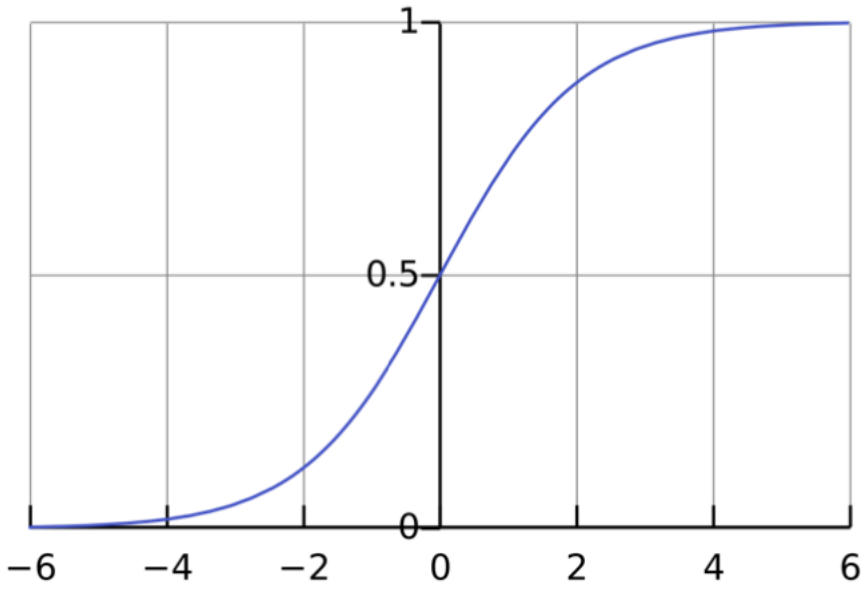


Figure 9. Logistic Sigmoid Function

In this example, the activation function transitions between 0 and 1 at the x value of 0. This would represent the bias vector having values of 0. Suppose we

wanted the transition to occur at some value b. We would set the value of the bias vector to b, and our graph would be

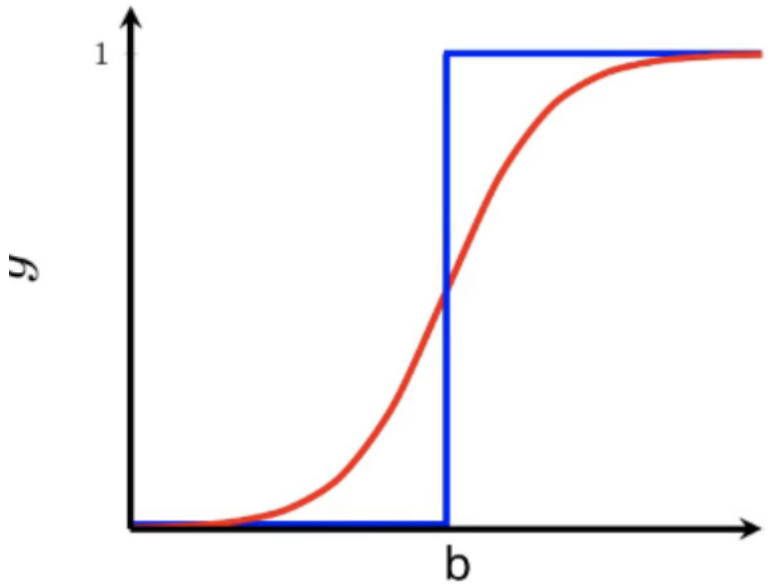


Figure 10. Logistic Sigmoid Function Shifted

Thus, while the activation function determines the shape of the curve that will normalize the values of our neurons, the bias vector gives us flexibility in how these neurons are normalized. They determine the cut-off which transitions numbers between a certain range.

The sigmoid function allows us to interpret outputs as probabilities between 0 and 1. This sigmoid function, is mainly used for binary classification problems; when we want our output to be one of two classification. The soft max activation function is a combination of multiple sigmoid functions. It can can be used to assign probabilities in classification problems with more than two outputs.

$$\sigma(\mathbf{z})_j = \frac{e^{zj}}{\sum_{k=1}^{K} e^{zk}}$$

There are many other activation functions that can be used to introduce non linearity to each layer. However, it is important that these functions are differentiable almost everywhere. As we will see later, this will make the process of optimizing the network much easier.

Let's formalize the architecture of a neural network.

Let $a_j^{(l)}$ be the value of the j-th activation unit in the l-th layer. Note that activation units in each layer will affect activation units in the next layers. If there are $M^{(l-1)}$ activation units at the $(l-1)$-th layer, we have

$$a_j^{(l)} = \sigma\left(\sum_{i=0}^{M^{(l-1)}} w_{ji}^{(l)} a_i^{(l-1)} + w_{j0}\right)$$

We can view the bias w_{j0} as a special weight parameter by setting $a_0^{(l)} = 1$ in each layer. Then

$$a_j^{(l)} = \sigma\left(\sum_{i=0}^{M^{(l-1)}} w_{ji}^{(l)} a_i^{(l-1)}\right).$$

for $1 \leq l \leq L$, where L is thte total number of layers.

This equation is stating that every activation unit is the result of summing all the weight-input vector connections and applying the activation function to the result. A neural network takes values in the input layer and uses **forward propagation** through the hidden layers. We define **forward propagation** as the action of neurons "propagating" forward to the next layer through the edges connected to them. Consider the previous picture

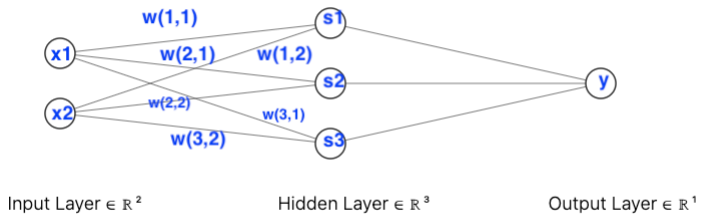


Figure 11. Forward Propagation

with

$$\sigma_{\mathbf{b}}\left(\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \\ w_{31}x_1 + w_{32}x_2 \end{bmatrix}\right) = \sigma_{\mathbf{b}}(\begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \\ w_{31}x_1 + w_{32}x_2 \end{bmatrix}) = \begin{bmatrix} \sigma(w_{11}x_1 + w_{12}x_2 - b_1) \\ \sigma(w_{21}x_1 + w_{22}x_2 - b_2) \\ \sigma(w_{31}x_1 + w_{32}x_2 - b_3) \end{bmatrix}$$

Consider that $a_1 = \sigma(w_{11}x_1 + w_{12}x_2 - b_1)) = \sum_{i=1}^{2} w_1 i x_i - w_{10}$. where $w_{10} = b_1$.

4.2. Gradient Descent

We previously described how the process of forward propagation depends on the weights in the network. We will now describe how these weights are selected.

In neural networks, we utilize **gradient descent** to take an initial selection of weights whose values are often initialized randomly and update them through an iterative process until our total error is close to 0. This error is computed with respect to a set a data that includes input values and the correct output values which we call the "training data."

Before we consider gradient descent in the context of neural networks, let's consider the problem of optimizing general function $f : \mathbb{R}^N \to \mathbb{R}$. Recall the definition of the gradient.

Definition 4.2. The gradient of a scalar-valued differentiable function f of several variables is the vector $\bigtriangledown f$ whose value at point p is the vector whose components are the partial derivative of f at point p. For $f : \mathbb{R}^n \to \mathbb{R}$, its gradient $\bigtriangledown f$ is defined at the point $p = (x_1, x_2, \ldots, x_n)$ for the n-dimensional vector

$$\bigtriangledown f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \frac{\partial f}{\partial x_2}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

We now describe the process for finding a local optimum $f(\mathbf{w}_*)$ of a function, $f : \mathbb{R}^n \to \mathbb{R}$, if it exists. We start with an initial guess $\mathbf{w}_0$ and then iterate according to the rule

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda_i (\bigtriangledown f)(\mathbf{w}_i))^T$$

which, under the right conditions, the the sequence $f(\mathbf{w}_0) \geq f(\mathbf{w}_1) \geq \ldots$ will converge to a local minimum.

The variable λ is called the step size. This will tell us the rate we will walk against the gradient. It is also called the **learning rate**. This is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

In simple terms, the gradient helps us known what direction to walk in to find the local minimum, and the learning rate tells us how big a "step" we will take in that direction. If our learning rate is too high, we could imagine over-stepping the local minimum. If our learning rate is too low, it would be computationally inefficient to get to the minimum. See the image below

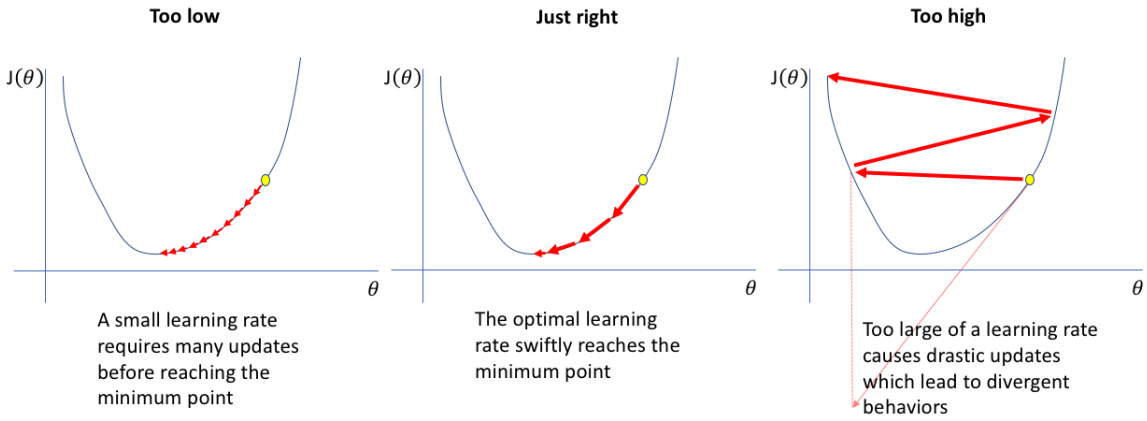


Figure 12. Learning Rate

In practice, if a suitable learning rate is chosen, gradient descent should return a vector of weights which minimizes the error.

Given a weight vector $\mathbf{w}$, we define $E(\mathbf{x}_n, \mathbf{w}, \mathbf{t}_m) = \frac{1}{2}||y(\mathbf{x}_n, \mathbf{w}) - t_n||^2$, then $E(\mathbf{w})$ can be defined $E(\mathbf{w}) = \sum_{n=1}^{N} E(\mathbf{x}_n, \mathbf{w}, \mathbf{t}_n)$ where $\mathbf{x}_n, \mathbf{t}_n$ with $n = 1, \ldots n$ are a collection of inputs and desired outputs respectively. $E(\mathbf{w})$ quantifies how close the network is getting to the desired output in the training data. We will use gradient descent to minimize the error function. Note that $\mathbf{y}(\mathbf{x}_m, \mathbf{w})$ will be the vector $\mathbf{y}$ where each $y_j = a_j^{(L)}$.

Our job is to choose the weights of the neural network which minimizes $E(\mathbf{w})$. Gradient descent, tells us which direction to move the weights in so that $E(\mathbf{w})$ moves closer towards its local minimum.

Note that by equation 4.1,

$$a_j^{(l)} = \sigma\left(\sum_{i=0}^{M^{(l-1)}} w_{ji}^{(l)} a_i^{(l-1)}\right)$$

Assuming the network has l layers and layer $l-1$ has N neurons, we have

$$y_j = a_j^{(l)} \sigma\left(\sum_{i=0}^{N} w_{ji} a_i^{(l-1)}\right).$$

We have

$$\bigtriangledown E(\mathbf{x}) = \frac{\partial E(\mathbf{x}, \mathbf{w}, \mathbf{t})}{\partial w_{ji}^{(l)}}$$

for all $l = 1, \ldots, L$ and all i, j.

4.3. Backpropagation

One of the most popular methods to evaluate the gradient of the error function $E(\mathbf{w})$ for a neural network is known as **backpropagation**. In particular it computes, $\frac{\partial E(\mathbf{w})}{\partial w_{jk}^{(l)}}$ by evaluating

$$\frac{\partial E(\mathbf{x}_n, \mathbf{w}, \mathbf{t}_n)}{\partial w_{ji}^{(l)}}$$

for $1 \leq n \leq N$.

Applying the chain rule, we have

$$\frac{\partial E(\mathbf{x}_n, \mathbf{w}, \mathbf{t}_n)}{\partial w_{ji}^{(l)}} = \frac{\partial E}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}}$$

In what follows, we will be evaluating various auxiliary partial derivatives using a recursive process. The following notation will be helpful. Let $\delta_j^{(l)} = \frac{\partial E}{\partial a_j^{(l)}}$. Now, we have $\frac{\partial E(\mathbf{x}_n, \mathbf{w}, \mathbf{t}_n)}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}}$.

We will now use the chain rule to compute the derivatives on the right side.
Using $a_j^{(l)} = \sigma\left(\sum_{r=1}^{M^{(l-1)}} w_{jr}^{(l)} a_r^{(l-1)}\right)$, we have $\frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = \sigma'\left(\sum_{r=1}^{M^{(l-1)}} w_{jr}^{(l)} a_r^{(l-1)}\right) a_i^{(l-1)}$.
For $l < L$, we compute $\delta_j^{(l)}$ recursively in terms of $\delta_i^{(l+1)}$, $(1 \leq i \leq M^{(l+1)})$, as follows

$$\delta_j^{(l)} = \sum_{i=1}^{M^{(l+1)}} \frac{\partial E}{\partial a_i^{(l+1)}} \frac{\partial a_i^{(l+1)}}{\partial a_j^{(l)}} = \sum_{i=1}^{M^{(l+1)}} \delta_i^{(l+1)} \frac{\partial a_i^{(l+1)}}{\partial a_j^{(l)}}$$

where $\frac{\partial a_i^{(l+1)}}{\partial a_j^{(l)}} = \sigma'\left(\sum_{r=1}^{M^{(l)}} w_{ir}^{(l+1)} a_r^{(l)})\right) w_{ij}^{(l+1)}$, since $a_i^{(l+1)} = \sigma\left(\sum_{r=1}^{M^{(l+1)}} w_{ir}^{(l+1)} a_r^{(l)}\right)$.
For $l = L$, we have $\delta_j^{(L)} = \frac{\partial E}{\partial a_j^{(L)}} = \frac{\partial}{\partial a_j^{(L)}}\left(\sum_{r=1}^{M^{(L)}} (a_r^{(L)} - \mathbf{t}_n)^2\right) = a_j^{(L)} - (\mathbf{t}_n)_j$.

Note that if are trying to find the value of some $\frac{\partial E(\mathbf{x}_n, \mathbf{w}, \mathbf{t}_n)}{\partial w_{ji}^{(l)}}$, and have already computed $\frac{\partial E(\mathbf{x}_n, \mathbf{w}, \mathbf{t}_n)}{\partial w_{ji}^{(l+1)}}$, $\left(\sum_i \delta_i^{(l+1)} \frac{\partial a_i^{(l+1)}}{\partial a_j^{(l)}}\right)$ should already be known! If we start by computing the weights at the last layer, computing the weights of layers after are much more computationally efficient!

Lemma 4.3. *The backpropagation algorithm goes as follow*

1) *Apply an input vector $\boldsymbol{x}_n$ to the network and forward propagate through the network to find the activations of all the hidden and output units.*

2) *Evaluate δ_k for all the output units.*

3) *Backpropagate the δs to obtain δ_j for each hidden unit.*

4) *Use these values to evaluate $\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}}$ for each $w_{ji}^{(l)}$ at each layer l.*

In conclusion, we will determine a certain number of data points which we will perform gradient descent and backpropagation on. For methods like batch we will use every data point for this process. Methods like stochastic gradient descent, will choose a random number of data points to conduct this process. However, what matters is that for every data point, we will utilize backpropagation to compute the gradients for each weight. We will then use gradient-descent to move each weight in the direction away from the gradient closer to the local minimum. We will then repeat this process for some list of data points, at each data point moving closer to the local minimum.

In practice, the training data-sets can be very large. In the example above, we utilized **batch-gradient descent** and defined the loss function with respect to all

the data-points. However, this can be computationally expensive. A different idea is to define the cost with respect to some randomly selected subset of data points at each step of the descent. This is called **stochastic gradient descent**.

CHAPTER 5. CONVOLUTIONAL NEURAL NETWORKS

5.1. Convolution as a Sliding Dot Product

Particularly for image-processing, most neural networks have multiple layers that involve convolution before they reach fully-connected layers. The purpose of convolutional layers is to extract features from the input image. The input layer will be some input image which can be represented with an $m \times n$ matrix A where each entry corresponds to a pixel in the image. This is the standard for grey-scale pictures. However, for color images where we are using the RGB color model, each RGB component of the image is represented by a matrix. Viewing these three separate $m \times n$ matrices as one object, we obtain a higher-dimensional version of a matrix called a tensor.

Consider the case of a grey-scale image represented by the following matrix

$$A = \begin{bmatrix} 3 & 3 & 2 & 1 & 0 & 5 \\ 0 & 0 & 1 & 3 & 1 & 6 \\ 3 & 1 & 2 & 2 & 3 & 7 \\ 2 & 0 & 0 & 2 & 2 & 8 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{bmatrix}$$

We can consider the following 9 different sub-matrices $A_1, \ldots A_{12}$ respectively

$$\begin{pmatrix} \boxed{\begin{matrix} 3 & 3 & 2 \\ 0 & 0 & 1 \\ 3 & 1 & 2 \end{matrix}} & 1 & 0 & 5 \\ 3 & 1 & 2 & 2 & 3 & 7 \\ 2 & 0 & 0 & 2 & 2 & 8 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 3 & \boxed{\begin{matrix} 3 & 2 & 1 \\ 0 & 1 & 3 \\ 1 & 2 & 2 \end{matrix}} & 0 & 5 \\ 3 & 1 & 2 & 2 & 3 & 7 \\ 2 & 0 & 0 & 2 & 2 & 8 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 3 & 3 & \boxed{\begin{matrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 2 & 2 & 3 \end{matrix}} & 5 \\ 3 & 1 & 2 & 2 & 3 & 7 \\ 2 & 0 & 0 & 2 & 2 & 8 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 3 & 3 & 2 & \boxed{\begin{matrix} 1 & 0 & 5 \\ 3 & 1 & 6 \\ 2 & 3 & 7 \end{matrix}} \\ 3 & 1 & 2 & 2 & 3 & 7 \\ 2 & 0 & 0 & 2 & 2 & 8 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 3 & 2 & 1 & 0 & 5 \\ \boxed{\begin{matrix} 0 & 0 & 1 \\ 3 & 1 & 2 \\ 2 & 0 & 0 \end{matrix}} & 3 & 1 & 6 \\ 2 & 2 & 3 & 7 \\ 2 & 2 & 8 \\ 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 3 & 3 & 2 & 1 & 0 & 5 \\ 0 & \boxed{\begin{matrix} 0 & 1 & 3 \\ 1 & 2 & 2 \\ 0 & 0 & 2 \end{matrix}} & 1 & 6 \\ 3 & 1 & 2 & 2 & 3 & 7 \\ 2 & 0 & 0 & 2 & 2 & 8 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 3 & 3 & 2 & 1 & 0 & 5 \\ 0 & 0 & \boxed{\begin{matrix} 1 & 3 & 1 \\ 1 & 2 & 2 \\ 0 & 2 & 2 \end{matrix}} & 6 \\ 3 & 1 & 2 & 2 & 3 & 7 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 3 & 3 & 2 & 1 & 0 & 5 \\ 0 & 0 & 1 & \boxed{\begin{matrix} 3 & 1 & 6 \\ 2 & 3 & 7 \\ 2 & 2 & 8 \end{matrix}} \\ 3 & 1 & 2 & 2 & 3 & 7 \\ 2 & 0 & 0 & 2 & 2 & 8 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 3 & 2 & 1 & 0 & 5 \\ 0 & 0 & 1 & 3 & 1 & 6 \\ \boxed{\begin{matrix} 3 & 1 & 2 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{matrix}} & 2 & 3 & 7 \\ 2 & 2 & 8 \\ 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 3 & 3 & 2 & 1 & 0 & 5 \\ 0 & 0 & 1 & 3 & 1 & 6 \\ 3 & \boxed{\begin{matrix} 1 & 2 & 2 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{matrix}} & 3 & 7 \\ 2 & 2 & 8 \\ 1 & 9 \end{pmatrix} \begin{pmatrix} 3 & 3 & 2 & 1 & 0 & 5 \\ 0 & 0 & 1 & 3 & 1 & 6 \\ 3 & 1 & \boxed{\begin{matrix} 2 & 2 & 3 \\ 2 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}} & 7 \\ 2 & 0 & 0 & 2 & 2 & 8 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{pmatrix} \begin{pmatrix} 3 & 3 & 2 & 1 & 0 & 5 \\ 0 & 0 & 1 & 3 & 1 & 6 \\ 3 & 1 & 2 & \boxed{\begin{matrix} 2 & 3 & 7 \\ 2 & 2 & 8 \\ 0 & 1 & 9 \end{matrix}} \\ 2 & 0 & 0 & 2 & 2 & 8 \\ 2 & 0 & 0 & 0 & 1 & 9 \end{pmatrix}$$

Convolution involves performing a dot-product operation between each sub-matrix and a pre-determined kernel matrix. The kernel matrix is the matrix that slides through every sub-matrix and performs a dot-product operation. The kernel represents some feature in the image that we are trying to recognize. Each entry in

the output matrix says something about the similarity between the kernel and the corresponding sub-matrix that was used to compute the dot product. Suppose our kernel matrix is

$$K = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

. The result of the convolution of A with k would be

$$\begin{bmatrix} A_1 \cdot K & A_2 \cdot K & A_3 \cdot K & A_4 \cdot K \\ A_5 \cdot K & A_6 \cdot K & A_7 \cdot K & A_8 \cdot K \\ A_9 \cdot K & A_{10} \cdot K & A_{11} \cdot K & A_{12} \cdot K \end{bmatrix} = \begin{bmatrix} 12 & 12 & 17.0 & 35 \\ 10.0 & 17.0 & 19.0 & 41 \\ 9.0 & 6.0 & 14.0 & 44 \end{bmatrix}$$

Consider how we got $A_4 \cdot K$

$$1 * 0 + 0 * 1 + 2 * 5 + 2 * 3 + 1 * 2 + 6 * 0 + 2 * 0 + 1 * 3 + 7 * 2$$

Note that the dot product of a matrix with itself is the square of its magnitude, so values in the matrix output that are close to the magnitude squared of the kernel, indicate that that part of the matrix held some important pattern. This is why convolution is so effective at feature extraction. As we will explain later, convolutional neural networks still have fully-connected layers at the end of the network.

5.2. Convolution as Matrix Multiplication

It can be shown that convolving a vector with another vector is equivalent to multiplying a vector by a **circulant matrix**.

Let's define a circulant matrix

Definition 5.1. A circulant matrix, C, is a matrix of the form

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & & \dots & c_{n-1} \\ c_{n-1} & c_0 & c_1 & c_2 & & \vdots \\ & c_{n-1} & c_0 & c_1 & \ddots & \\ \vdots & \ddots & \ddots & \ddots & & c_2 \\ & & & & & c_1 \\ c_1 & \dots & & c_{n-1} & & c_0 \end{bmatrix}$$

where each row is a cyclic shift of the row above it. The structure can also be characterized by noting that (k, j)-entry C_{kj} is given by

$$C_{kj} = c_{j-k \pmod n}$$

We write $j - k \pmod n$ to refer to the index $0 \leq i \leq n - 1$ such that $i \equiv j - k \pmod n$.

Definition 5.2. Given $\mathbf{a}, \mathbf{x} \in \mathbb{R}^n$, we define the convolution of $\mathbf{a}$ and $\mathbf{x}$, denoted $\mathbf{a} \star \mathbf{x}$, to be the vector whose kth component is $\sum_{l=1}^{n} a_{k-l} x_l$ for $1 \leq k \leq n$.

The fact that definition 5.2 implies the following lemma can be easily verified.

Lemma 5.3. *For $\boldsymbol{a}, \boldsymbol{x} \in \mathbb{R}^n$, we have $\boldsymbol{a} \star \boldsymbol{x} = \boldsymbol{y} = C_a\boldsymbol{x}$ where C_a where C_a is a circulant matrix made up of the a_k*

$$C_a = \begin{bmatrix} a_0 & a_{n-1} & \dots & a_1 \\ a_1 & a_0 & \dots & a_2 \\ \vdots & & \dots & \vdots \\ a_{n-1} & a_{n-2} & \dots & a_0 \end{bmatrix}$$

$$\boldsymbol{y} = \boldsymbol{a} \star \boldsymbol{x} = C_a\boldsymbol{x} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} a_0 & a_{n-1} & \dots & a_1 \\ a_1 & a_0 & \dots & a_2 \\ \vdots & & \dots & \vdots \\ a_{n-1} & a_{n-2} & \dots & a_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = C_a\boldsymbol{x}.$$

Thus, the convolution of one vector convolved over the other is equivalent to a matrix vector product, where the corresponding matrix is a **circulant matrix**. However, in most image processing applications, our input will not be a n dimensional vector, but rather a $m \times n$ matrix.

5.3. Convolution of Grey-Scale Images

Let's return to our original example. If our training data set is a collection of grey-scale images with corresponding labels, each image in the set would be a matrix. Like the example we introduced on page 39, see example 5.3, we would be convolving some matrix kernel over a each image matrix. It can be shown that we can also represent this operation as matrix multiplication, albeit with a slight twist.

Lemma 5.4. *For any kernel K, the linear transform for the convolution by K is represented by the following block matrix*

$$A = \begin{bmatrix} Circ(K_0,:) & Circ(K_1,:) & \dots & Circ(K_{n-1},:) \\ Circ(K_{n-1},:) & Circ(K_0,:) & \dots & Circ(K_{n-2},:) \\ \vdots & \vdots & \vdots & \vdots \\ Circ(K_1,:) & Circ(K_2,:) & \dots & Circ(K_0,:) \end{bmatrix}$$

that is if X is an $n \times n$ matrix, and Y is the result of a $2-d$ convolution of X with K where for all i, j we have

$$Y_{ij} = \sum_{p\in[n]} \sum_{q\in[n]} X_{i+p,j+q} K_{p,q}$$

then $\text{vec}(Y) = A\ \text{vec}(X)$

Thus, the vectorized output of convolution on a matrix input, is equivalent to vectorizing the original input and multiplying it by a doubly block circulant matrix.

Let's bring up some important properties of convolution; **sparse connectivity**, **parameter sharing**, and **equivariant representation**. These ideas are explored in greater detail in [4].

First, note that convolutional layers, unlike fully-connected layers, involve sparse connectivity. Compare this to when we viewed convolution as matrix-multiplication from 5.3.

$$\begin{bmatrix} a_1 \cdot k & a_2 \cdot k & a_3 \cdot k & a_4 \cdot k \\ a_5 \cdot k & a_6 \cdot k & a_7 \cdot k & a_8 \cdot k \\ a_9 \cdot k & a_{10} \cdot k & a_{11} \cdot k & a_{12} \cdot k \end{bmatrix}$$

In a fully-connected layer every output entry is calculated using all of the input entries. However, note that in the matrix above, every entry is **not** calculated using every entry from the input matrix A. Rather, each convolution output entry depends only on the entries from the corresponding sub-matrix.

The figure below further visualizes the idea of sparse connectivity. The first image in the figure shows sparsely connected neurons, while the second image shows a fully-connected layer, where every input neuron is connected to every output neuron.

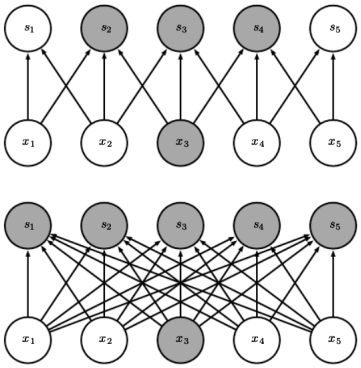


Figure 13. Sparse Connectivity

Similarly, convolutional layers involve parameter sharing. This means that we are using the same set of weights applied to different input elements. This idea is made obvious first when viewing convolution as a sliding dot-product; we are applying the same kernel across every sub-matrix. Also, consider an arbitrary circulant matrix

$$\begin{bmatrix} c_0 & c_1 & c_2 & & \cdots & c_{n-1} \\ c_{n-1} & c_0 & c_1 & c_2 & & \vdots \\ & c_{n-1} & c_0 & c_1 & \ddots & \\ \vdots & \ddots & \ddots & \ddots & & c_2 \\ & & & & & c_1 \\ c_1 & \cdots & & c_{n-1} & & c_0 \end{bmatrix}$$

Notice that each row of the circulant matrix contains the same set of weights. Therefore, the resulting vector product only uses weights $c_0, \dots c_{n-1}$ for every computation!

Finally, consider the idea of translation equivariance. To do this, let us introduce the idea that **circulant matrices commute**.

Definition 5.5. Define the circular left-shift operator, $S^\star : \mathbb{R}^n \to \mathbb{R}^n$ by

$$S^\star[x_0 x_1 \dots x_{n-1}]^T = [x_1 \dots x_{n-1} x_0]^T$$

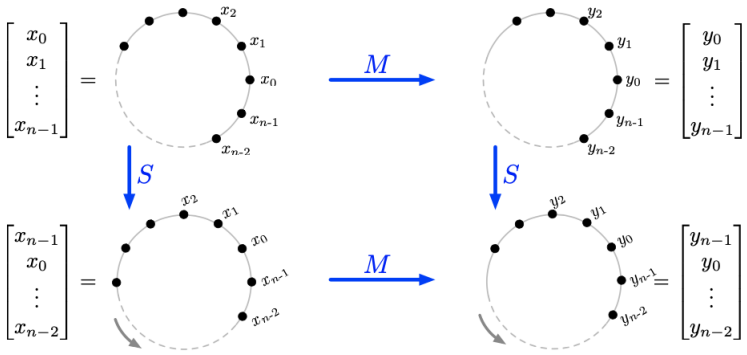
so we have $(S^\star \mathbf{x})_k = \mathbf{x}_{k+1}$ for all $\mathbf{x} \in \mathbb{R}^n$.

Definition 5.6. Define the circular right-shift operator, $S : \mathbb{R}^n \to \mathbb{R}^n$ by

$$S[x_0 \dots x_{n-2} x_{n-1}]^T = [x_{n-1} x_0 \dots x_{n-2}]^T$$

so we have $S(\mathbf{x})_k = \mathbf{x}_{k-1}$ for all $\mathbf{x} \in \mathbb{R}^n$.

Lemma 5.7. *[?] A matrix M is circulant $\Leftrightarrow$ it commutes with a circulant shift operator, S such that $SM = MS$. This also leads to shift invariance or shift equivariance. Think of an $n-$ vector $\boldsymbol{x}$ as as a function of Z_n then $SM\boldsymbol{x} = MS\boldsymbol{x}$. Taking an input vector and shifting it a certain number and then multiplying it by a circulant matrix is equivalent to shifting everything in the circulant matrix by the same number and then multiplying it by the vector $\boldsymbol{x}$*



[1]

Figure 14. Geometric view of Convolution

Lemma 5.8. *If $C(\boldsymbol{u})$ is a circulant matrix and $C(\boldsymbol{v})$ is also a circulant matrix it is true that*

$$C(\boldsymbol{u})C(\boldsymbol{v}) = C(\boldsymbol{u})C(\boldsymbol{v})$$

Example 5.9. Consider two circulant matrices C_1 and C_2 where

$$C_1 = \begin{bmatrix} 1 & 2 & 1 & 4 \\ 4 & 1 & 2 & 1 \\ 1 & 4 & 1 & 2 \\ 2 & 1 & 4 & 1 \end{bmatrix}, C_2 = \begin{bmatrix} 3 & 7 & 0 & 1 \\ 1 & 3 & 7 & 0 \\ 0 & 1 & 3 & 7 \\ 7 & 0 & 1 & 3 \end{bmatrix}$$

We can see that

$$C_1 \times C_2 = \begin{bmatrix} 33 & 14 & 21 & 20 \\ 20 & 33 & 14 & 21 \\ 21 & 20 & 33 & 14 \\ 14 & 21 & 20 & 33 \end{bmatrix}, C_2 \times C_1 = \begin{bmatrix} 33 & 14 & 21 & 20 \\ 20 & 33 & 14 & 21 \\ 21 & 20 & 33 & 14 \\ 14 & 21 & 20 & 33 \end{bmatrix}$$

Thus, it is obvious why circulant matrices are powerful for image recognition! They are able to find patterns no matter where they appear because they are translationally equivariant!

5.4. Computing Convolutions using Fast Fourier Transform

We previously discussed how multiplying a circulant matrix by a shifted vector is equivalent to multiplying a circulant matrix by the original vector and then shifting the product.

Note that both shift operators can be represented as **circulant matrices**! To see, examine the matrix representations of the shift operators

$$Sx = \begin{bmatrix} 0 & & & 1 \\ 1 & & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} x_{n-1} \\ x_0 \\ \vdots \\ x_{n-2} \end{bmatrix}$$

$$S^\star x = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & & 1 \\ 1 & & & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \\ x_0 \end{bmatrix}$$

It can be seen that $S^\star$ is the transpose of S. Furthermore, note that both matrix representations are circulant matrix as $S = C_{0,1,0,\ldots,0}$ and $S^\star = C_{0,\ldots,0,1}$

Lemma 5.10. *We can construct a basis of n eigenvectors for the left-shift operator $S^\star$ on $\mathbb{R}^n$ where each eigenvector is $\boldsymbol{w}_m$ where*

$$\boldsymbol{w}_m = \begin{bmatrix} 1 & p^m & p^{2m} & \ldots & p^{m(n-1)} \end{bmatrix}^T, m = 0, \ldots, n-1,$$

where

$$p = e^{i\frac{2\pi}{n}}.$$

Lemma 5.11. *Each eigenvalue of a left shift matrix can be written as $\lambda_m = e^{i\frac{2\pi}{n}m}$*

How does this help us easily compute circulant matrices? Consider that we can write any circulant matrix as a linear combination of left shift matrices. Therefore, we can write any matrix-vector product of a circulant matrix and a vector $C_x(\mathbf{x})$

as multiplying the linear combination of left-shift matrices and the vector which is equivalent to retaining the weights of the combination and replacing each S with its corresponding eigenvalue and multiplying by the vector. Consider the example below where we have

Example 5.12.

$$C = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix} = 1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + 2 \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} + 3 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$= 1I + 2S + 3S^2$$

We can see that we can write C as a linear combination of the left shift matrices such that $C = 1I + 2S + 3S^2$.

Note that now we have that for any vector $\mathbf{w}_m$

$$C\mathbf{v}_m = (I + 2S + 3S^2)\mathbf{v}_m$$

which is equivalent to

$$1I\mathbf{w}_m + 2S\mathbf{w}_m + 3S^2\mathbf{w}_m$$

Further note that we should be able to write each $S\mathbf{w}_m$ as $\lambda_m\mathbf{v}_m$. Therefore, we write $C\mathbf{w}_m$, finally, as

$$C\mathbf{w}_m = 1\mathbf{w}_m + 2\lambda_m\mathbf{w}_m + 3\lambda_m^2\mathbf{w}_m = (1 + 2\lambda_m + 3\lambda_m^2)\mathbf{w}_m$$

Remark 5.13. *As in the previous example, every circulant matrix can be decomposed as a polynomial in S. It is then easily verified that all circulant matrices commute (Lemma 5.8).*

The lemma below formalizes Example 5.12.

Lemma 5.14. *The eigenvalues of a circulant matrix μ_k are equal to*

$$\mu_k = \sum_{l=0}^{n-1} x_l e^{-i\frac{2\pi}{n}kl}, k = 0, 1, \ldots, n-1$$

where each row of the circulant matrix is some permutation of $x_0, \ldots, x_l$

Proof. Note that we can find find the eigenvalues of any circulant matrix C_x such from the eigenvectors of the shift operator $\mathbf{w}^m$ where

$$C_x\mathbf{w}^m = \lambda_m\mathbf{w}^m \Leftrightarrow \begin{bmatrix} x_0 & x_{n-1} & \ldots x_1 & \\ x_1 & x_0 & & x_2 \\ \vdots & & \ddots & \vdots \\ x_{n-1} & x_{n-2} & \ldots & x_0 \end{bmatrix} \begin{bmatrix} 1 \\ p_m \\ \vdots \\ p_m^{n-1} \end{bmatrix} = \lambda_m \begin{bmatrix} 1 \\ p_m \\ \vdots \\ p_m^{n-1} \end{bmatrix}$$

Consider the first row of the equation*. It is

$$\lambda_m(x_0 + x_{n-1}p_m + \ldots + x_1 p_m^{n-1} = x_0 + x_1 p_m^{-1} + \ldots + x_{n-1}p_m^{-(n-1)}$$

which is equal to

$$\sum_{l=0}^{n-1} x_l p_m^{-l} = \sum_{l=0}^{n-1} x_l p^{-ml} = \sum_{l=0}^{n-1} x_l e^{-i\frac{2\pi}{n}ml} = \hat{\mathbf{x}}_m$$

□

Therefore, we can conclude that circulant matrices are simultaneously diagonalizable. We can write any circulant matrix vector product as a linear combination of the eigenvalues of the left-shift operator and the corresponding vector. This makes it computationally efficient for computer to compute circulant matrix-vector products as it simply requires recalculating the eigenvalues of shift matrices which are easily computed.

5.5. PCA in Convolutional Neural Networks

Can principle component analysis be used in conjunction with convolutional neural networks? Both principle component analysis and convolution are used to find the most important patterns in the data. For PCA, this data reduction is based on variance and co-variance, while in convolution we reduce the data by looking for particular patterns which will help lead to the correct prediction. What if we were to apply PCA to a dataset and then train a convolutional neural network on the reduced set? We performed this test on the MNIST dataset. We compared the effectiveness of using a unchanged data-set and a dataset reduced through PCA trained on the same convolutional neural network. Both networks had the same structure, as seen below.

```
model = Sequential()
model.add(Conv1D(64, (3,3), activation = ‘‘relu"))
model.add(Conv1D(64, (3,3), activation = ‘‘relu"))
model.add(MaxPooling1D(pool_size = 3, strides = 1, padding = ‘‘valid"))
model.add(Conv1D(32, (3,3), activation = ‘‘relu"))
model.add(Conv1D(32, (3,3), activation = ‘‘relu"))
model.add(MaxPooling1D(pool_size = 3, strides = 1, padding = ‘‘valid"))
model.add(Conv1D(16, (3,3), activation = ‘‘relu"))
model.add(Conv1D(16, (3,3), activation = ‘‘relu"))
model.add(MaxPooling1D(pool_size = 3, strides = 1, padding = ‘‘valid"))
model.add(Conv1D(8, (3,3), activation = ‘‘relu"))
model.add(Conv1D(8, (3,3), activation = ‘‘relu"))
model.add(MaxPooling1D(pool_size = (2,2), strides = 1, padding = ‘‘valid"))
model.add(Flatten())
model.add(Dense(16, activation = ‘‘sigmoid"))
model.add(Dense(10, activation = ‘‘softmax"))
```

Note that our dataset was made of different grey-scale images of numbers from $0-9$. Therefore, in the original data set, each data point was a $28 \times 28 \times 1$ matrix with a corresponding 10-dimensional vector which signified what number it was. See

```
: print(x.shape, y.shape)
(70000, 28, 28, 1) (70000, 10)
```

However, remember that PCA requires that each data point be a vector. Therefore, before applying PCA, we had to reshape each data point into a $28 \times 28 = 784$ dimensional vector. We then reduced the data to only include the principal components which explained 99 percent of the total variance. This reduced each data point to a 331 dimensional vector. Furthermore, even though we used the same general structure for the neural network, because our input data were now one-dimensional vectors, our convolution was no longer two-dimensional convolution, but one-dimensional convolution. Thus our structure was as follows

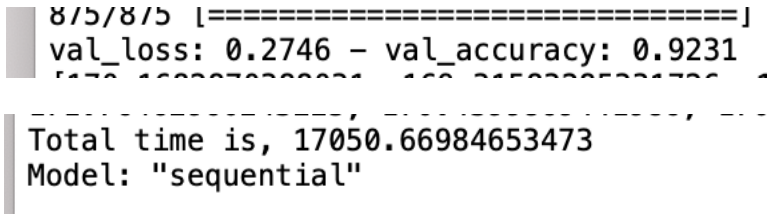
```
model = Sequential()
model.add(Conv1D(64, 3, activation = ‘‘relu"))
model.add(Conv1D(64, 3, activation = ‘‘relu"))
model.add(MaxPooling1D(pool_size = 3, strides = 1, padding = ‘‘valid"))
model.add(Conv1D(32, 3, activation = ‘‘relu"))
model.add(Conv1D(32, 3, activation = ‘‘relu"))
model.add(MaxPooling1D(pool_size = 3, strides = 1, padding = ‘‘valid"))
model.add(Conv1D(16, 3, activation = ‘‘relu"))
model.add(Conv1D(16, 3, activation = ‘‘relu"))
model.add(MaxPooling1D(pool_size = 3, strides = 1, padding = ‘‘valid"))
model.add(Conv1D(8, 3, activation = ‘‘relu"))
model.add(Conv1D(8, 3, activation = ‘‘relu"))
model.add(MaxPooling1D(pool_size = 2, strides = 1, padding = ‘‘valid"))
model.add(Flatten())
model.add(Dense(16, activation = ‘‘sigmoid"))
model.add(Dense(10, activation = ‘‘softmax"))
```

We found that our standard convolutional neural network was 98.84 percent accurate on the training data. It however took around 15 hours to train the network to the data. See

```
875/875 [==============================] -
val_loss: 0.0462 - val_accuracy: 0.9884
Total time is, 54101.34850502014
```

Note that total time is in seconds and that 54101.34850502014 seconds is equivalent to 15.028 hours.

Conversely, our convolutional neural network which was trained on the data reduced through principal component analysis was around 92 percent accurate on the training data, but only around 4.73 hours to train. (17050.66984653473 seconds is equivalent to 4.73 hours). See



Therefore, we found that although applying PCA to the data before training it on the same convolutional neural network decreased the accuracy by around 6 percent, it also had a 68 percent decrease in running time. In conclusion, it seems that principle component analysis is a valid way to reduce the running time of training a network. Some things to consider going forward is different ways to apply principle component analysis to two-dimensional data. Our method required flattening the data and and applying one dimensional rather than two-dimensional convolution to it. Methods of principle component analysis that do not require vectorized data might see a decreased loss in accuracy, while also maintaining the decrease in running time.

REFERENCES

[1] Bamieh, Bassam. (2018). “Discovering transforms: A tutorial on circulant matrices, circular convolution, and the discrete fourier transform.” arXiv preprint arXiv:1805.05533.

[2] Bronstein, M. (2022, January 2). “Deriving convolution from first principles.” Medium. Retrieved March 11, 2022, from https://towardsdatascience.com/deriving-convolution-from-first-principles-4ff124888028

[3] Faisal, A. A., Ong, C. S. (2020). “Matrix Approximation.” In “Mathematics for Machine Learning." essay, Cambridge University Press.

[4] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. (2016). “Deep learning.” MIT press.

[5] Kalman, D. (1996). “A Singularly Valuable Decomposition: The SVD of a Matrix.” The College Mathematics Journal, 27(1), 2–23.

[6] Lay, D. C., Lay, S. R., McDonald, J. (2022). “Linear algebra and its applications.” Pearson Education Limited.

[7] Murphy, Kevin P. (2012). “Machine Learning: A Probabilistic Perspective.” Cambridge: MIT Press. p. 247.

[8] Nielsen, Michael A. (2015). “Neural networks and deep learning.” Vol. 25. San Francisco, CA, USA: Determination press.

[9] Nicholson, W. K. (2018). “Linear algebra with applications ;’’ Open Textbook Library.

[10] Sharma, Sagar, Simone Sharma, and Anidhya Athaiya. (2017). “Activation functions in neural networks.” towards data science 6.12: 310-316.