Linear Algebra

Foundations of Machine Learning

Paul A. Jensen
University of Illinois at Urbana-Champaign

Spring 2022 Edition

Contents

Part III

High-Dimensional Systems

Chapter 17

Vector Spaces, Span, and Basis

17.1 Vector Spaces

Vector spaces are collections of vectors. The most common spaces are $\mathbb{R}^2$, $\mathbb{R}^3$, and $\mathbb{R}^n$ — the spaces that include all 2-, 3-, and n-dimensional vectors. We can construct *subspaces* by specifying only a subset of the vectors in a space. For example, the set of all 3-dimensional vectors with only integer entries is a subspace of $\mathbb{R}^3$.

Remember that $\mathbb{R}^2$ is not a subspace of $\mathbb{R}^3$; they are completely separate, non-overlapping spaces.

17.2 Span

A set of m vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$ is said to *span* a space V if any vector $\mathbf{u}$ in V can be written as a linear combination of the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$. This is equivalent to saying there exist scalars $a_1, a_2, \ldots, a_m$ such that

$$\mathbf{u} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_m\mathbf{v}_m.$$

Writing a vector $\mathbf{u}$ as a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$ is called *decomposing* $\mathbf{u}$ over $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$. If a set of vectors spans a space, they can be used to decompose any other vector in the space.

We've already seen vector composition using a special set of vectors $\hat{\mathbf{e}}_j$, the unit vectors with only one nonzero entry at element j. For example, the vector

$$\begin{pmatrix} -2 \\ 4 \\ 5 \end{pmatrix} = -2\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 4\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + 5\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Thus the vectors $\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \hat{\mathbf{e}}_3$ spans $\mathbb{R}^3$. In general, the set of vectors $\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \ldots, \hat{\mathbf{e}}_n$ spans the space $\mathbb{R}^n$. Are these the only sets of vectors that span these spaces?

No, there are infinitely many sets of vectors that span each space. Consider the vectors $\begin{pmatrix}1\\1\end{pmatrix}$ and $\begin{pmatrix}-1\\1\end{pmatrix}$. We can show that these vectors span $\mathbb{R}^2$ by showing that any vector $\mathbf{u}$ in $\mathbb{R}^2$ can be written as a linear combination

$$\mathbf{u} = \begin{pmatrix}u_1\\u_2\end{pmatrix} = a_1\begin{pmatrix}1\\1\end{pmatrix} + a_2\begin{pmatrix}-1\\1\end{pmatrix}.$$

Finding the coefficients a_1 and a_2 is akin to solving the system of linear equations

$$\begin{aligned} a_1 - a_2 &= u_1 \\ a_1 + a_2 &= u_2 \end{aligned}$$

which has the unique solution

$$a_1 = \frac{u_1 + u_2}{2}, \quad a_2 = \frac{u_2 - u_1}{2}.$$

To demonstrate, let $\mathbf{u} = \begin{pmatrix}-2\\4\end{pmatrix}$. Then $a_1 = 1$ and $a_2 = 3$, so

$$a_1\begin{pmatrix}1\\1\end{pmatrix} + a_2\begin{pmatrix}-1\\1\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix} + 3\begin{pmatrix}-1\\1\end{pmatrix} = \begin{pmatrix}1-3\\1+3\end{pmatrix} = \begin{pmatrix}-2\\4\end{pmatrix}.$$

We've shown that there are least two sets of vectors that span $\mathbb{R}^2$, $\left\{\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}\right\}$ and $\left\{\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}-1\\1\end{pmatrix}\right\}$. How can we say there are infinitely many? If vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ span a space V, then the vectors $k_1\mathbf{v}_1$ and $k_2\mathbf{v}_2$ also span V for any scalars k_1 and k_2. To prove this, remember that any vector $\mathbf{u}$ can be decomposed onto $\mathbf{v}_1$ and $\mathbf{v}_2$, i.e.

$$\begin{aligned} \mathbf{u} &= a_1\mathbf{v}_1 + a_2\mathbf{v}_2 \\ &= \frac{a_1}{k_1}(k_1\mathbf{v}_1) + \frac{a_2}{k_2}(k_2\mathbf{v}_2) \end{aligned}$$

Since (a_1/k_1) and (a_2/k_2) are simply scalars, we've shown that $\mathbf{u}$ can be decomposed onto the vectors $k_1\mathbf{v}_1$ and $k_2\mathbf{v}_2$. Therefore, $k_1\mathbf{v}_1$ and $k_2\mathbf{v}_2$ must also span $\mathbb{R}^2$. For example, the vectors $\begin{pmatrix}3\\0\end{pmatrix}$ and $\begin{pmatrix}0\\-1/2\end{pmatrix}$ are scalar multiples of $\begin{pmatrix}1\\0\end{pmatrix}$ and $\begin{pmatrix}0\\1\end{pmatrix}$. The

Since k_1 and k_2 are arbitrary, this allows us to generate infinitely many sets of vectors that span any space from a single spanning set.

former two vectors must therefore span $\mathbb{R}^2$, so we can decompose the vector $\begin{pmatrix}-2\\4\end{pmatrix}$ onto them:

$$\begin{pmatrix}-2\\4\end{pmatrix} = -\frac{2}{3}\begin{pmatrix}3\\0\end{pmatrix} - 8\begin{pmatrix}0\\1/2\end{pmatrix}.$$

Similarly, if $\mathbf{v}_1$ and $\mathbf{v}_2$ span a space V, the vectors $\mathbf{v}_1$ and $(\mathbf{v}_1 + \mathbf{v}_2)$ also span V:

$$\begin{aligned}\mathbf{u} &= a_1\mathbf{v}_1 + a_2\mathbf{v}_2\\ &= a_1\mathbf{v}_1 + a_2(\mathbf{v}_1 + \mathbf{v}_2) - a_2\mathbf{v}_1\\ &= (a_1 - a_2)\mathbf{v}_1 + a_2(\mathbf{v}_1 + \mathbf{v}_2)\end{aligned}$$

If scalars a_1 and a_2 decompose $\mathbf{u}$ over $\mathbf{v}_1$ and $\mathbf{v}_2$, then $(a_1 - a_2)$ and a_2 decompose $\mathbf{u}$ over $\mathbf{v}_1$ and $(\mathbf{v}_1 + \mathbf{v}_2)$.

17.3 Review: Linear Independence

We said before that vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ are *linearly independent* if and only if

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n = \mathbf{0}$$

implies that all coefficients $a_1, a_2, \ldots, a_n$ are zero. No linear combination of a set of linearly independent vectors can be the zero vector except for the trivial case where all the coefficients are zero. We often say that a set of vectors are linearly dependent if one of the vectors can be written as a linear combination of the others, i.e.

$$\mathbf{v}_i = a_1\mathbf{v}_1 + \cdots + a_{i-1}\mathbf{v}_{i-1} + a_{i+1}\mathbf{v}_{i+1} + \cdots + a_n\mathbf{v}_n.$$

Moving the vector $\mathbf{v}_i$ to the right hand side

$$\mathbf{0} = a_1\mathbf{v}_1 + \cdots + a_{i-1}\mathbf{v}_{i-1} - \mathbf{v}_i + a_{i+1}\mathbf{v}_{i+1} + \cdots + a_n\mathbf{v}_n$$

we see 1.) a linear combination of the vectors sums to the zero vector on the left, and 2.) at least one of the coefficients (the -1 in front of $\mathbf{v}_i$) is nonzero. This is consistent with are above definition of linear independence. We said these vectors were not linearly independent, so it is possible for a linear combination to sum to zero using at least one nonzero coefficient.

17.4 Basis

The concepts of span and linear independence are a powerful combination. Any linearly independent set of vectors that span a space V are called a *basis* for V.

Any vector in a space be decomposed over a set of vectors that span the space. **However, every vector in a space has a unique decomposition over an associated basis.** Said another way, for every vector in a space, there are only one set of coefficients $a_1, a_2, \ldots, a_n$ that decompose it over a basis $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$.

We can prove that a decomposition over a basis is unique by contradiction. Suppose there were two sets of coefficients – $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ – that decomposed a vector $\mathbf{u}$ over a basis $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$. Then

$$\mathbf{u} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n = b_1\mathbf{v}_1 + b_2\mathbf{v}_2 + \cdots + b_n\mathbf{v}_n.$$

We can move all the right hand size over to the left and group terms to give

$$(a_1 - b_1)\mathbf{v}_1 + (a_2 - b_2)\mathbf{v}_2 + \cdots + (a_n - b_n)\mathbf{v}_n = \mathbf{0}.$$

Remember that $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ is a basis, so the vectors must be linearly independent. By the definition of linear independence, the only way the above equation can be true is if all the coefficients are zero. This implies that $a_1 = b_1$, $a_2 = b_2$, and so on. Clearly this is a violation of our original statement that $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots b_n$ where different. Therefore, there can only be one way to decompose any vector onto a basis.

17.4.1 Testing if vectors form a basis

Every basis for a vector space has the same number of vectors. This number is called the *dimension* of the vector space. For standard vectors spaces like $\mathbb{R}^n$, the dimension is n. The dimension of $\mathbb{R}^2$ is 2, and the dimension of $\mathbb{R}^3$ is 3.

Most people think of dimension as the number of elements in a vector. While the true definition of dimension is the number of vectors in the basis, counting elements in a vector works for spaces like $\mathbb{R}^n$. To see why, remember that the Cartesian unit vectors $\hat{\mathbf{e}}_i$ form a basis for $\mathbb{R}^n$, but we need n of these vectors, one per element.

Any set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ is a basis for a space V if and only if:

1. The number of vectors (n) matches the dimension of V.
2. The vectors span V.
3. The vectors are linearly independent.

Proving any two of the above statements automatically implies the third is true.

We get to choose which two of the above three statements to prove when verifying that $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ is a basis. The first statement is usually trivial — does the number of vectors match the dimension? We almost always choose to prove the first statement. Proving that vectors are linearly independent is always easier that proving the vectors span the space. If we collect the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ into a matrix, the rank of this matrix should be n if the vectors are linearly independent.

17.4.2 Decomposing onto a basis

How do we decompose a vector onto a basis? Remember that decomposing $\mathbf{u}$ over $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ is equivalent to finding a set of coefficients $a_1, a_2, \ldots, a_n$ such that

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n = \mathbf{u}.$$

Let's collect the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ into a matrix $\mathbf{V}$, where each vector $\mathbf{v}_i$ is the ith column in $\mathbf{V}$. Then

$$\begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \mathbf{V}\mathbf{a} = \mathbf{u}.$$

We see that finding the coefficients $a_1, a_2, \ldots, a_n$ that decompose a vector $\mathbf{u}$ onto a basis $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ is equivalent to solving the linear system $\mathbf{V}\mathbf{a} = \mathbf{u}$.

By formulating vector decomposition as a linear system, we can easily see why the decomposition of a vector over a basis is unique. In $\mathbb{R}^n$, the basis contains n vectors, each with n elements. So, the matrix $\mathbf{V}$ is a square, $n \times n$ matrix. Since the vectors in the basis (and therefore the columns in $\mathbf{V}$) are linearly independent, the matrix $\mathbf{V}$ has full rank. Thus, the solution to $\mathbf{V}\mathbf{a} = \mathbf{u}$ must be unique, implying that the decomposition of every vector onto a basis is unique.

Since $\mathbf{V}$ is square and full rank, its inverse ($\mathbf{V}^{-1}$) must exist. The system has a unique solution $\mathbf{a} = \mathbf{V}^{-1}\mathbf{u}$.

17.5 Orthogonal and Orthonormal Vectors

A set of vectors is an *orthogonal set* if every vector in the set is orthogonal to every other vector in the set. If every vector in an orthogonal set has been normalized, we say the vectors form an *orthonormal set*. Orthogonal and orthonormal sets are ideal candidates for basis vectors. Since there is no "overlap" among the vectors, we can easily decompose other vectors onto orthogonal basis vectors.

Imagine you have an orthogonal set of vectors you want to use as a basis. We'll assume you have the correct number of vectors (equal to the dimension of your space) for this to be possible. Based on the above requirements for basis vectors, we need only to show that these vectors are linearly independent. If so, they are a basis. For a set of n orthogonal vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$, linear independence requires that

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n = \mathbf{0}$$

if and only if all the coefficients $a_1, a_2, \ldots, a_n$ are equal to zero. Let's take the dot product of both sides of the above equation with the vector $\mathbf{v}_1$:

$$(a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n) \cdot \mathbf{v}_1 = \mathbf{0} \cdot \mathbf{v}_1.$$

On the right hand size, we know that $\mathbf{0} \cdot \mathbf{v}_1 = 0$ for any vector $\mathbf{v}_1$. We also distribute the dot product on the left hand side to give

$$a_1\mathbf{v}_1 \cdot \mathbf{v}_1 + a_2\mathbf{v}_2 \cdot \mathbf{v}_1 + \cdots + a_n\mathbf{v}_n \cdot \mathbf{v}_1 = 0.$$

Since all the vectors are orthogonal, $\mathbf{v}_i \cdot \mathbf{v}_1$ is zero except when $i = 1$. Canceling out all the dot products equal to zero shows that

$$a_1\mathbf{v}_1 \cdot \mathbf{v}_1 = a_1 \|\mathbf{v}_1\|^2 = 0.$$

We know that $\|\mathbf{v}_1\|^2 \neq 0$, so the only way $a_1\mathbf{v}_1 \cdot \mathbf{v}_1$ can equal zero is if a_1 is zero. If we repeat this entire process by taking the dot product with $\mathbf{v}_2$ instead of $\mathbf{v}_1$, we will find that $a_2 = 0$. This continues with $\mathbf{v}_3, \ldots, \mathbf{v}_n$ until we can say that $a_1 = a_2 = \cdots = a_n = 0$. Therefore, if the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ are an orthogonal (or orthonormal) set, they are linearly independent.

17.5.1 Decomposing onto orthonormal vectors

We saw previously that finding the coefficients to decompose a vector onto a basis requires solving a system of linear equations. For high-dimensional spaces, solving such a system can be computationally expensive. Fortunately, decomposing a vector onto an orthonormal basis is far more efficient.

Theorem. *The decomposition of a vector* $\mathbf{u}$ *onto an orthonormal basis* $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_n$ *given by*

$$\mathbf{u} = a_1\hat{\mathbf{v}}_1 + a_2\hat{\mathbf{v}}_2 + \cdots + a_n\hat{\mathbf{v}}_n$$

We use the symbol $\hat{\mathbf{v}}_i$ for vectors in an orthonormal set to remind us that each vector has been normalized.

has coefficients

$$\begin{aligned} a_1 &= \mathbf{u} \cdot \hat{\mathbf{v}}_1 \\ a_2 &= \mathbf{u} \cdot \hat{\mathbf{v}}_2 \\ &\vdots \\ a_n &= \mathbf{u} \cdot \hat{\mathbf{v}}_n \end{aligned}$$

Proof. We use a similar strategy as when we proved the linear independence of orthogonal sets. Let's start with the formula for vector decomposition

$$\mathbf{u} = a_1\hat{\mathbf{v}}_1 + a_2\hat{\mathbf{v}}_2 + \cdots + a_n\hat{\mathbf{v}}_n.$$

Taking the dot product of both sides with the vector $\hat{\mathbf{v}}_1$ yields (after distributing the right hand side)

$$\mathbf{u} \cdot \hat{\mathbf{v}}_1 = a_1\hat{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_1 + a_2\hat{\mathbf{v}}_2 \cdot \hat{\mathbf{v}}_1 + \cdots + a_n\hat{\mathbf{v}}_n \cdot \hat{\mathbf{v}}_1.$$

Because all the vectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_n$ are orthogonal, the only nonzero term on the right hand side is $a_1\hat{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_1$, so

$$\mathbf{u} \cdot \hat{\mathbf{v}}_1 = a_1\hat{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_1.$$

By definition of the dot product, $\hat{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_1 = \|\hat{\mathbf{v}}_1\|^2$. Since $\hat{\mathbf{v}}_1$ is a unit vector, $\|\hat{\mathbf{v}}_1\|^2 = 1$. Thus $a_1 = \mathbf{u} \cdot \hat{\mathbf{v}}_1$. By repeating the same procedure with $\hat{\mathbf{v}}_2$, we find that $a_2 = \mathbf{u} \cdot \hat{\mathbf{v}}_2$, and so on. □

Decomposing vectors over an orthonormal basis is efficient, requiring only a series of dot products to compute the coefficients. For example, we can decompose the vector $\mathbf{u} = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix}$ over the orthonormal basis $\left\{ \hat{\mathbf{v}}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \hat{\mathbf{v}}_2 = \begin{pmatrix} 0 \\ 3/5 \\ 4/5 \end{pmatrix}, \hat{\mathbf{v}}_3 = \begin{pmatrix} 0 \\ 4/5 \\ -3/5 \end{pmatrix} \right\}$.

$$a_1 = \mathbf{u} \cdot \hat{\mathbf{v}}_1 = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 7 + 0 + 0 = 7$$

$$a_2 = \mathbf{u} \cdot \hat{\mathbf{v}}_2 = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 3/5 \\ 4/5 \end{pmatrix} = 0 - 3 + 8 = 5$$

$$a_3 = \mathbf{u} \cdot \hat{\mathbf{v}}_3 = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 4/5 \\ -3/5 \end{pmatrix} = 0 - 4 - 6 = -10$$

The decomposition of $\mathbf{u}$ is

$$\mathbf{u} = 7\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 5\begin{pmatrix} 0 \\ 3/5 \\ 4/5 \end{pmatrix} - 10\begin{pmatrix} 0 \\ 4/5 \\ -3/5 \end{pmatrix} = \begin{pmatrix} 7+0+0 \\ 0+3-8 \\ 0+4+6 \end{pmatrix} = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix}.$$

17.5.2 Checking an orthonormal set

Given a set of vectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_n$, how can we verify that they are orthonormal? We need to test two things.

1. All vectors are normalized ($\|\hat{\mathbf{v}}_i\| = 1$ for all $\hat{\mathbf{v}}_i$).
2. All vectors are mutually orthogonal ($\hat{\mathbf{v}}_i \cdot \hat{\mathbf{v}}_j = 0$ for all $i \neq j$).

The first test is straightforward. The second can be a little cumbersome, as we need to test all $n^2 - n/2$ pairs of vectors for orthogonality. A simpler, albeit more sometimes more computationally intensive approach, is to collect the vectors into a matrix $\mathbf{V} = \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{pmatrix}$. Then the set of vectors is orthonormal if an only if $\mathbf{V}^{-1} = \mathbf{V}^{\mathsf{T}}$. While inverting a matrix is "expensive," for small to medium size vector sets this method avoids the need to iterate over all pairs of vectors to test for orthonormality.

Interestingly, the proof of this method (not shown here) reveals that if the columns of $\mathbf{V}$ are an orthonormal set, the rows of $\mathbf{V}$ are also an orthonormal set!

17.5.3 Projections

Our next goal will be to create orthonormal sets of vectors from sets that are not orthogonal. Before introducing such an algorithm, we need to develop a geometric tool — the vector *projection*. The projection of vector $\mathbf{v}$ onto vector $\mathbf{u}$ is a vector that points along $\mathbf{u}$ with length equal to the "shadow" of $\mathbf{v}$ onto $\mathbf{u}$. Previously we used the dot product to calculate the magnitude of the projection of $\mathbf{v}$ onto $\mathbf{u}$, which was a scalar equal to $\|\mathbf{v}\| \cos\theta$, where θ is the angle between $\mathbf{v}$ and $\mathbf{u}$. To calculate the actual projection, we multiply the magnitude of the projection ($\|\mathbf{v}\| \cos\theta$) by a unit vector that points along $\mathbf{u}$. Thus the projection of $\mathbf{v}$ onto $\mathbf{u}$ is defined as

$$\text{proj}_{\mathbf{u}}(\mathbf{v}) = (\|\mathbf{v}\| \cos\theta)\hat{\mathbf{u}}.$$

By definition, $\hat{\mathbf{u}} = \mathbf{u}/\|\mathbf{u}\|$. Also, we note that $\mathbf{v}\cdot\mathbf{u} = \|\mathbf{v}\|\,\|\mathbf{u}\| \cos\theta$, so the expression $\|\mathbf{u}\| \cos\theta$ can be written in terms of the dot product $(\mathbf{v}\cdot\mathbf{u})/\|\mathbf{u}\|$. We can rewrite our formula for the projection using only dot products:

$$\text{proj}_{\mathbf{u}}(\mathbf{v}) = (\|\mathbf{v}\| \cos\theta)\hat{\mathbf{u}} = \frac{\mathbf{v}\cdot\mathbf{u}}{\|\mathbf{u}\|}\frac{\mathbf{u}}{\|\mathbf{u}\|} = \frac{\mathbf{v}\cdot\mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u} = \frac{\mathbf{v}\cdot\mathbf{u}}{\mathbf{u}\cdot\mathbf{u}}\mathbf{u}.$$

We can use the projection to make any two vectors orthogonal, as demonstrated by the following theorem.

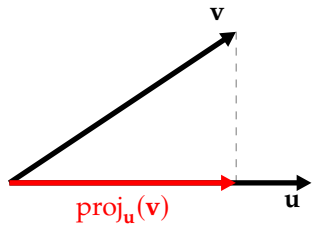


Figure 17.1: The projection is a vector "shadow" of one vector onto another.

Theorem. *Given any vectors* $\mathbf{v}$ *and* $\mathbf{u}$*, the vector* $\mathbf{v}$- $\text{proj}_{\mathbf{u}}(\mathbf{v})$ *is orthogonal to* $\mathbf{u}$.

Proof. If the vector $\mathbf{v} - \text{proj}_{\mathbf{u}}(\mathbf{v})$ is orthogonal to $\mathbf{u}$, the dot product between these vectors must be zero.

$$\begin{aligned}
\left(\mathbf{v} - \text{proj}_{\mathbf{u}}(\mathbf{v})\right)\cdot\mathbf{u} &= \left(\mathbf{v} - \frac{\mathbf{v}\cdot\mathbf{u}}{\mathbf{u}\cdot\mathbf{u}}\mathbf{u}\right)\cdot\mathbf{u} \\
&= \mathbf{v}\cdot\mathbf{u} - \frac{\mathbf{v}\cdot\mathbf{u}}{\mathbf{u}\cdot\mathbf{u}}\mathbf{u}\cdot\mathbf{u} \\
&= \mathbf{v}\cdot\mathbf{u} - \mathbf{v}\cdot\mathbf{u} \\
&= 0
\end{aligned}$$

□

Subtracting the projection of $\mathbf{v}$ onto $\mathbf{u}$ from the vector $\mathbf{v}$ "corrects" $\mathbf{v}$ by removing its overlap with $\mathbf{u}$. The resulting vector is a vector closest to $\mathbf{v}$ that is still orthogonal to $\mathbf{u}$.

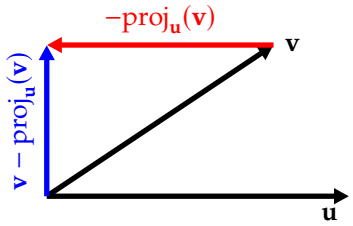


Figure 17.2: Subtracting the projection of $\mathbf{v}$ onto $\mathbf{u}$ from $\mathbf{v}$ makes the vectors orthogonal.

We must begin with linearly independent vectors. Otherwise, orthogonalization will turn one of the vectors into the zero vector, which is not allowed in a basis.

17.5.4 Creating orthonormal basis vectors

We can make any two vectors orthogonal by adjusting one based on its projection onto the other. We can apply these corrections iteratively to make any set of linearly independent vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ into an orthonormal basis set $\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_n$. First, we set

$$\mathbf{u}_1 = \mathbf{v}_1.$$

We leave this first vector unchanged. All other vectors will be made orthogonal to it (and each other). Next, we create $\mathbf{u}_2$ by making $\mathbf{v}_2$ orthogonal to $\mathbf{u}_1$:

$$\mathbf{u}_2 = \mathbf{v}_2 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_2).$$

Now we have two orthogonal vectors, $\mathbf{u}_1$ and $\mathbf{u}_2$. We continue by creating $\mathbf{u}_3$ from $\mathbf{v}_3$, but this time we must make $\mathbf{v}_3$ orthogonal to both $\mathbf{u}_1$ and $\mathbf{u}_2$:

$$\mathbf{u}_3 = \mathbf{v}_3 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \text{proj}_{\mathbf{u}_2}(\mathbf{v}_3).$$

We continue this process for all n vectors, making each vector $\mathbf{v}_i$ orthogonal to all the newly created orthogonal vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{i-1}$. This approach is called the Gram-Schmidt algorithm. Given a set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$, we create a set of orthogonal vectors

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{v}_1 \\
\mathbf{u}_2 &= \mathbf{v}_2 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_2) \\
\mathbf{u}_3 &= \mathbf{v}_3 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \text{proj}_{\mathbf{u}_2}(\mathbf{v}_3) \\
&\vdots \\
\mathbf{u}_i &= \mathbf{v}_i - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_i) - \cdots - \text{proj}_{\mathbf{u}_{i-1}}(\mathbf{v}_i) \\
&\vdots \\
\mathbf{u}_n &= \mathbf{v}_n - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_n) - \cdots - \text{proj}_{\mathbf{u}_{n-1}}(\mathbf{v}_n)
\end{aligned}$$

The Gram-Schmidt algorithm products an orthogonal set of vectors. To make the set orthonormal, we must subsequently normalize each vector.

Said more succinctly,

$$\mathbf{u}_i = \mathbf{v}_i - \sum_{k=1}^{i-1} \text{proj}_{\mathbf{u}_k}(\mathbf{v}_i)$$

Chapter 18

Eigenvectors and Eigenvalues

Consider the matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix}.$$

Multiplying $\mathbf{A}$ by the vector $\mathbf{x}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$ gives an interesting result.

$$\mathbf{A}\mathbf{x}_1 = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ -5 \end{pmatrix} = -5\begin{pmatrix} -1 \\ 1 \end{pmatrix} = -5\mathbf{x}_1.$$

Similarly, with $\mathbf{x}_2 = \begin{pmatrix} -7 \\ 1 \end{pmatrix}$:

$$\mathbf{A}\mathbf{x}_2 = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix}\begin{pmatrix} -7 \\ 1 \end{pmatrix} = \begin{pmatrix} -7 \\ 1 \end{pmatrix} = \mathbf{x}_2.$$

In both cases, multiplication by $\mathbf{A}$ returned a scalar multiple of the vector (-5 for $\mathbf{x}_1$ and 1 for $\mathbf{x}_2$). This is not a property of solely the matrix $\mathbf{A}$, since the vector $\mathbf{x}_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ is not transformed by a single scalar.

$$\mathbf{A}\mathbf{x}_3 = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 9 \\ 5 \end{pmatrix} \neq \lambda\mathbf{x}_2$$

Similarly, the results we are seeing are not properties of the vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, since they do not become scalar multiples of themselves when multiplied by other

matrices.

$$\mathbf{B} = \begin{pmatrix} 2 & 1 \\ -3 & 0 \end{pmatrix}$$

$$\mathbf{B}\mathbf{x}_1 = \begin{pmatrix} 2 & 1 \\ -3 & 0 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 3 \end{pmatrix} \neq \lambda\mathbf{x}_1$$
$$\mathbf{B}\mathbf{x}_2 = \begin{pmatrix} 2 & 1 \\ -3 & 0 \end{pmatrix}\begin{pmatrix} -7 \\ 1 \end{pmatrix} = \begin{pmatrix} -13 \\ 21 \end{pmatrix} \neq \lambda\mathbf{x}_2$$

The phenomena we're observing is a result of the paring between the matrix $\mathbf{A}$ and the vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. In general, we see that multiplying a vector by a matrix returns a scalar multiple of the vector, or

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

Any vector $\mathbf{x}$ that obeys the above relationship is called an *eigenvector* of the matrix $\mathbf{A}$. The scalar λ is called the *eigenvalue* associated with the eigenvector $\mathbf{x}$. The vector $\mathbf{x}$ is an eigenvector of the matrix $\mathbf{A}$; it is not generally an eigenvector of other matrices.

Eigenvectors were originally called *characteristic* vectors, as they describe the character of the matrix. German mathematicians dropped this nomenclature in favor of the German prefix "eigen-", which mean "own". An eigenvector can be viewed as one of a matrix's "own" vectors since it is not rotated when transformed by multiplication.

In the example above, the matrix $\mathbf{A} = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix}$ has two eigenvectors, $\mathbf{v}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$ with eigenvalue $\lambda_1 = -5$, and $\mathbf{v}_2 = \begin{pmatrix} -7 \\ 1 \end{pmatrix}$ with eigenvector $\lambda_2 = 1$.

18.1 Properties of Eigenvectors and Eigenvalues

Only square matrices have eigenvectors and eigenvalues. To understand why the matrix must be square, remember that a non-square matrix with m rows and n columns transforms an n-dimensional vectors into an m-dimensional vector. Clearly, the m-dimensional output cannot be the n-dimensional input multiplied by a scalar!

An n by n matrix of real numbers can have up to n distinct eigenvectors. Each eigenvector is associated with an eigenvalue, although the eigenvalues can be duplicated. Said another way, two eigenvectors $\mathbf{v}_1$ and $\mathbf{v}_2$ of a matrix will never be the same, but the corresponding eigenvalues λ_1 and λ_2 can be identical.

Although the number of eigenvectors may vary, all eigenvectors for a matrix are linearly independent. Thus, if an n by n matrix has n eigenvectors, these vectors can be used as a basis (called an *eigenbasis*). If an eigenbasis exists for a matrix, decomposing vectors over this basis simplifies the process of matrix

An n by n matrix with n eigenvectors and n distinct eigenvalues is called a *perfect matrix*. As the name implies, perfect matrices are great to find, but somewhat uncommon.

multiplication. To illustrate, imagine we decompose the vector $\mathbf{x}$ over a set of eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$. Decomposing $\mathbf{x}$ means we can find coefficients $a_1, \ldots, a_n$ such that

$$\mathbf{x} = a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n.$$

Now let's compute the product $\mathbf{Ax}$. We multiply both sides of the decomposition by $\mathbf{A}$.

$$\mathbf{Ax} = \mathbf{A}\left(a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n\right)$$

We distribute the matrix $\mathbf{A}$ into the sum on the right hand side and note that the constants a_i can be moved in front of the matrix multiplication.

$$\mathbf{Ax} = a_1\mathbf{Av}_1 + \cdots + a_n\mathbf{Av}_n$$

Remember that $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are eigenvectors of $\mathbf{A}$, so $\mathbf{Av}_i = \lambda_i\mathbf{v}_i$. We can simplify the previous expression to

$$\mathbf{Ax} = a_1\lambda_1\mathbf{v}_1 + \cdots + a_n\lambda_n\mathbf{v}_n.$$

We don't need to perform the multiplication at all! Instead, we can scale each eigenvector by the eigenvalue. Multiplying again by the matrix $\mathbf{A}$ multiplies each eigenvector by its eigenvalue.

We use the notation $\mathbf{A}^2$ to denote $\mathbf{AA}$, $\mathbf{A}^3$ for $\mathbf{AAA}$, and $\mathbf{A}^k$ for the product of k matrices $\mathbf{A}$.

$$\mathbf{A}^2\mathbf{x} = a_1\lambda_1^2\mathbf{v}_1 + \cdots + a_n\lambda_n^2\mathbf{v}_n$$

$$\mathbf{A}^k\mathbf{x} = a_1\lambda_1^k\mathbf{v}_1 + \cdots + a_n\lambda_n^k\mathbf{v}_n$$

18.2 Computing Eigenvectors and Eigenvalues

We can use the relationship between matrix multiplication and eigenvalues to find eigenvectors for any matrix. Our computational approach is based on the following theorem.

Theorem. *Given any (random) vector* $\mathbf{b}$*, repeated multiplication by the matrix* $\mathbf{A}$ *will converge to the eigenvector of* $\mathbf{A}$ *with the largest magnitude eigenvalue – provided the largest eigenvalue is unique. Said another way,*

$$\lim_{k\to\infty} \mathbf{A}^k\mathbf{b} = \mathbf{v}_{\max}.$$

Proof. We know that the product $\mathbf{Ax}$ can be expressed as a linear combination of the eigenvectors and eigenvalues of $\mathbf{A}$, i.e. $\mathbf{Ax} = a_1\lambda_1\mathbf{v}_1 + \cdots + a_n\lambda_n\mathbf{v}_n$. Thus

$$\lim_{k\to\infty} \mathbf{Ab} = \lim_{k\to\infty} \left(a_1\lambda_1^k\mathbf{v}_1 + \cdots + a_n\lambda_n^k\mathbf{v}_n\right).$$

As k increases, the values λ_i^k grow very large. However, the λ_i to not grow at the same rate. The largest eigenvalue will grow the fastest. At very large values of k, the term associated with the largest eigenvalue will dominate the entire sum, so the result will point in only the direction of the associated eigenvector. Note that convergence to a single eigenvector requires that the largest eigenvalue be distinct. If two eigenvectors have the same (largest) eigenvalue, both terms in the above sum would "blow up" at the same rate. Repeated multiplications by $\mathbf{A}$ would then converge to the sum of the two associated eigenvectors. □

The above theorem allows us to find the eigenvector paired with the largest eigenvalue. While the direction of the eigenvector doesn't change, its magnitude grows as the number of multiplication of $\mathbf{A}$ increases. If convergence is slow, we might need to work with numbers before finding the eigenvector. To avoid numerical difficulties, we renormalize the vector after every multiplication by $\mathbf{A}$. This algorithm is called the Power Iteration method, which proceeds as follows:

1. Choose a random vector $\mathbf{b}_0$. For fastest convergence, it helps to choose a vector close to $\mathbf{v}_{\max}$ if possible. Normalize this vector to product $\hat{\mathbf{b}}_0 = \mathbf{b}_0/\|\mathbf{b}_0\|$.
2. Compute vector $\mathbf{b}_1 = \mathbf{A}\hat{\mathbf{b}}_0$. Normalize this vector to give $\hat{\mathbf{b}}_1 = \mathbf{b}_1/\|\mathbf{b}_1\|$.
3. Repeat step 2 to product $\hat{\mathbf{b}}_2, \hat{\mathbf{b}}_3, \ldots, \hat{\mathbf{b}}_k$. Stop when all entries of $\hat{\mathbf{b}}_k$ do not change from the entries in $\hat{\mathbf{b}}_{k-1}$. The vector $\hat{\mathbf{b}}_k$ is an eigenvector of $\mathbf{A}$.

The eigenvector associated with the largest magnitude eigenvalue is called the *leading eigenvector*.

Now that we have the eigenvector $\mathbf{v}_{\max}$, how do we find the associated eigenvalue $\lambda_{\max}$? We know that $\mathbf{v}_{\max}$ is an eigenvector of $\mathbf{A}$, to $\mathbf{A}\mathbf{v}_{\max} = \lambda_{\max}\mathbf{v}_{\max}$. The ith element in $\mathbf{A}\mathbf{v}_{\max}$ should be equal to $\lambda_{\max}$ times the ith element in $\mathbf{v}_{\max}$. However, since we only found a numerical approximation to the $\mathbf{v}_{\max}$, the estimate for $\lambda_{\max}$ from each element in $\mathbf{v}_{\max}$ might differ slightly. To "smooth out" these variations, compute the eigenvalue using the Rayleigh quotient:

$$\lambda_{\max} = \frac{\mathbf{v}_{\max} \cdot \mathbf{A}\mathbf{v}_{\max}}{\mathbf{v}_{\max} \cdot \mathbf{v}_{\max}}.$$

To see why the Raleigh quotient works, consider an eigenvector $\mathbf{v}$ for matrix $\mathbf{A}$ with associated eigenvalue λ. Then

$$\frac{\mathbf{v} \cdot \mathbf{A}\mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} = \frac{\mathbf{v} \cdot (\lambda\mathbf{v})}{\mathbf{v} \cdot \mathbf{v}} = \lambda\frac{\mathbf{v} \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} = \lambda.$$

The dot product in the Rayleigh quotient averages out all of the small discrepancies between our estimate $\mathbf{v}_{\max}$ and the true largest eigenvector. The Rayleigh quotient provides a numerically stable estimate of the largest eigenvalue.

Now that we've found the first eigenvector, how do we find the others? If we start the Power Iteration method over again using the matrix $(\mathbf{A} - \lambda_{\max}\mathbf{I})$ instead of $\mathbf{A}$, the algorithm will converge to the eigenvector associated with the second largest eigenvalue. We can subtract this eigenvalue from $\mathbf{A}$ and repeat to find

the third eigenvector, and so on. Proving that Power Iteration is able to find subsequent eigenvectors is beyond the scope of this book. However, as we'll see later, finding only the first eigenvector is sufficient for addressing a number of interesting problems.

18.2.1 Eigenvalues and Eigenvectors in MATLAB

The MATLAB function `eig` computes eigenvalues and eigenvectors. The statement `[V,L] = eig(A)` involving an n by n matrix `A` returns two n by n matrices:

- Each column of the matrix `V` is an eigenvector `A`.
- The matrix `L` is a diagonal matrix. The ith entry on the diagonal is the eigenvalue associated with the ith column in `V`.

Remember that any vector that points in the same direction as an eigenvector of a matrix is also an eigenvector of that matrix. If the eigenvectors returned by computational systems like MATLAB are not what you expect, remember that they may be normalized or scaled – but still point along the same direction.

18.3 Applications

Eigenvalue and eigenvectors can be used to solve a number of interesting engineering and data science problems.

18.3.1 Solving Systems of ODEs

Consider the linear system of ODEs

$$\frac{dx_1}{dt} = x_1 + 2x_2$$
$$\frac{dx_2}{dt} = 3x_1 + 2x_2$$

with initial conditions $x_1(0) = 0$ and $x_2(0) = -4$. We can write this system using vectors and matrices as

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}, \quad \mathbf{x}(0) = \mathbf{x}_0$$

where for the example above

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 2 \end{pmatrix}, \quad \mathbf{x}_0 = \begin{pmatrix} 0 \\ -4 \end{pmatrix}.$$

If we know the eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ and eigenvalues $\lambda_1, \ldots, \lambda_n$ for the matrix $\mathbf{A}$, we can compute the solution as

$$\mathbf{x}(t) = c_1\mathbf{v}_1 e^{\lambda_1 t} + c_2\mathbf{v}_2 e^{\lambda_2 t} + \cdots + c_n\mathbf{v}_n e^{\lambda_n t}.$$

This solution requires the matrix $\mathbf{A}$ be perfect and therefore have a complete set of eigenvectors.

The scalars $c_1, \ldots, c_n$ are the constants of integration. To find these values, notice what happens to our solution at time $t = 0$:

$$\mathbf{x}(0) = \mathbf{x}_0 = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n.$$

At $t = 0$, the right hand side is a decomposition of the initial conditions $\mathbf{x}_0$. If we collect the eigenvectors as columns of a matrix $\mathbf{V} = (\mathbf{v}_1\mathbf{v}_2 \ldots \mathbf{v}_n)$, we can find the constants $c_1, \ldots, c_n$ by solving the linear system

$$\mathbf{V}\begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \mathbf{x}_0.$$

Returning to our original example, the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 2 \end{pmatrix}$$

has eigenvalue/eigenvector pairs

$$\lambda_1 = -1, \quad \mathbf{v}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \qquad \text{and} \qquad \lambda_2 = 4, \quad \mathbf{v}_2 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}.$$

The integration constants c_1 and c_2 are defined by the system $\mathbf{Vc} = \mathbf{x}_0$, which for this example is

$$\begin{pmatrix} -1 & 2 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 0 \\ -4 \end{pmatrix}.$$

Solving the above equations reveals $c_1 = -8/5$ and $c_2 = -4/5$. The final solution to this systems of ODEs is

$$\mathbf{x}(t) = -\frac{8}{5}\begin{pmatrix} -1 \\ 1 \end{pmatrix} e^{-t} - \frac{4}{5}\begin{pmatrix} 2 \\ 3 \end{pmatrix} e^{4t}.$$

The function $f(t) = e^{\lambda t}$ is an *eigenfunction* of the derivative operator, i.e.

$$\frac{d}{dt} f(t) = \lambda e^{\lambda t} = \lambda f(t)$$

. The solution of a system of linear ODEs is the product of the eigenvectors of $\mathbf{A}$ and the eigenfunctions of $\frac{d\mathbf{x}}{dt}$.

18.3.2 Stability of Linear ODEs

The eigenvalues of $\mathbf{A}$ are sufficient to tell if the system $\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}$ is stable. For a system of linear ODEs to be stable, all eigenvalues of $\mathbf{A}$ must be nonpositive. If the eigenvalues are all negative, each term $e^{\lambda_i t}$ goes to zero at long times, so all variables in the system to go zero. If any of the eigenvalues are zero, the system is still stable (provided all other eigenvalues are negative), but the system will go to a constant value $c_i\mathbf{v}_i$, where $\mathbf{v}_i$ is the eigenvector associated with the zero eigenvalue.

18.3.3 Positive Definite Matrices

A symmetric matrix $\mathbf{A}$ is *positive definite* (p.d.) if $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} > 0$ for all nonzero vectors $\mathbf{x}$. If a matrix $\mathbf{A}$ satisfies the slightly relaxed requirement that $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} \geq 0$ for all nonzero $\mathbf{x}$, we say that $\mathbf{A}$ is *positive semi-definite* (p.s.d.).

Remember that a matrix $\mathbf{A}$ is symmetric if $\mathbf{A} = \mathbf{A}^\mathsf{T}$.

Knowing that a matrix is positive (semi-)definite is useful for quadratic programming problems like the Support Vector Machine classifier. The quadratic function $f(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{Q}\mathbf{x}$ is convex if and only if the matrix $\mathbf{Q}$ is positive semi-definite. For optimization problems like quadratic programs, the convexity of the objective function has enormous implications. Convex quadratic programs must only have global optima, making them easy to solve using numerical algorithms.

If $\mathbf{Q}$ is positive definite (rather than just positive semi-definite) then $\mathbf{x}^\mathsf{T}\mathbf{Q}\mathbf{x}$ is strictly convex.

Determining if a matrix is positive (semi-)definite can be difficult unless we use eigenvalues. Any matrix with only positive eigenvalues is positive definite, and any matrix with only nonnegative eigenvalues is positive semi-definite. For example, consider the 2×2 identity matrix

$$\mathbf{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

The product $\mathbf{x}^\mathsf{T}\mathbf{I}\mathbf{x}$ is

$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = x_1^2 + x_2^2.$$

Since $x_1^2 + x_2^2$ is greater than zero for all nonzero inputs x_1 and x_2, the matrix $\mathbf{I}$ is positive definite and all its eigenvalues should be positive. Indeed, the eigenvalues for the identity matrix are $\lambda_1 = \lambda_2 = 1$.

As another example, consider the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}.$$

The product $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} = x_1^2 - 4x_1x_2 + x_2^2$, which is not always positive. When $x_1 = x_2 = 1$, we see that $x_1^2 - 4x_1x_2 + x_2^2 = -2$. We know that $\mathbf{A}$ is not positive definite (or even

positive semi-definite), so $\mathbf{A}$ should have at least one negative eigenvalue. As expected, the eigenvalues for $\mathbf{A}$ are $\lambda_1 = 3$ and $\lambda_2 = -1$.

18.3.4 Network Centrality

Networks are represented by collections of *nodes* connected by *edges*. When analyzing a network, it is common to ask which node occupies the most important position in the network. For example, which airport would cause the most problems if closed due to weather? In biological networks, the importance or *centrality* of an enzyme is used to prioritize drug targets.

We can quantify the centrality of each node in a network using random walks. We start by choosing a random node in the network. Then we randomly choose one of the edges connected to the node and travel to a new node. This process repeats again and again as we randomly traverse nodes and edges. The centrality of each node is proportional to the number of times we visit the node during the random walk. In the airport analogy, randomly traveling to cities across the country means you will frequently visit major hub airports.

Measuring centrality by random walks is easy to understand but impractical for large networks. It could take millions of steps to repeatedly reach all the nodes in networks with only a few thousand of nodes. In practice, we use eigenvectors to calculate centrality for networks. We begin by constructing an *adjacency matrix* for the network. The adjacency matrix encodes the connections (edges) between the nodes. The adjacency matrix is square with rows and columns corresponding to nodes in the network. The (i,j) entry in the network is set to 1 if there is an edge connecting node i with node j. Otherwise, the entries are zero. Note that we only consider direct connections. If node 1 is connected to node 2 and node 2 is connected to node 3, we do not connect nodes 1 and 3 unless there is a direct edge between them. Also, no node is connected to itself, so the diagonal elements are always zero.

Consider the four node network shown in Figure 18.1. The four node network has the following adjacency matrix.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

To identify the most central node in the network, we find the eigenvector that corresponds to the largest eigenvalue ($\lambda_{\max}$). For the above adjacency matrix,

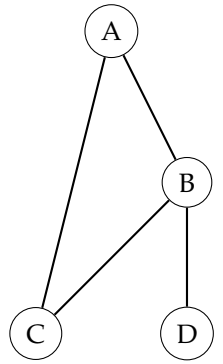


Figure 18.1: Sample network with four nodes and four edges.

$\lambda_{\max} \approx 2.2$, and the associated eigenvector is

$$\mathbf{v}_{\max} = \begin{pmatrix} 0.52 \\ 0.61 \\ 0.52 \\ 0.28 \end{pmatrix}.$$

The entries in the eigenvector $\mathbf{v}_{\max}$ are called the *eigencentrality* scores. The largest entry corresponds to the most central node, and the smallest entry is associated with the least central node. We see that node B (entry 2) is most central in Figure 18.1 and node D (entry 4) is least central.

Centrality requires only the leading eigenvector of a network's adjacency matrix. Power Iteration (section 18.2) can find the leading eigenvector efficiently for large networks.

In simple networks like Figure 18.1, the most central node also has the most direct connections. This is not always the case. Eigencentrality considers not only the number of connections but also their importance. Each edge is weighted by the centrality of the nodes it connect. Connections from more central nodes are more important, just as flights between major hub cities usually have the highest passenger volumes. Eigencentrality has numerous applications including web searching. Google uses a modified version of centrality (called PageRank) to determine which results should be displayed first to users.

18.4 Geometric Interpretation of Eigenvalues

Consider a matrix $\mathbf{A} \in \mathbb{R}^2$ with eigenvalues λ_1 and λ_2 and corresponding eigenvectors $\mathbf{v}_1$ and $\mathbf{v}_2$. Let's take a vector $\mathbf{x}$ and decompose it over the eigenvectors.

$$\mathbf{x} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2$$

We can represent the vector $\mathbf{x}$ by plotting it; however, instead of using the normal Cartesian unit vectors as axes, we will use the eigenvectors. The values of $\mathbf{x}$ along the "eigenaxes" are a_1 and a_2. What happens when we multiply $\mathbf{x}$ and $\mathbf{A}$?

$$\mathbf{A}\mathbf{x} = \mathbf{A}(a_1\mathbf{v}_1 + a_2\mathbf{v}_2) = \lambda_1 a_1\mathbf{v}_1 + \lambda_2 a_2\mathbf{v}_2$$

Visually, multiplying by $\mathbf{A}$ scales the values along the eigenvector axes. The scaling factor along each axis is the corresponding eigenvalue. To quantify the overall effect of multiplying by the matrix $\mathbf{A}$, we can compare the areas swept out by the vector $\mathbf{x}$ before and after multiplication. The area is simply the product of the values along each axis, so the ratio becomes

$$\frac{\text{area}(\mathbf{A}\mathbf{x})}{\text{area}(\mathbf{x})} = \frac{\lambda_1 a_1 \lambda_2 a_2}{a_1 a_2} = \lambda_1\lambda_2.$$

We can repeat the same calculation in three dimensions by looking at the ratio of the volume before and after multiplying by the matrix $\mathbf{A}$.

$$\frac{\text{volume}(\mathbf{Ax})}{\text{volume}(\mathbf{x})} = \frac{\lambda_1 a_1 \lambda_2 a_2 \lambda_3 a_3}{a_1 a_2 a_3} = \lambda_1 \lambda_2 \lambda_3$$

In general, the product of the eigenvalues of a matrix describe to overall effect of multiplying a vector by the matrix. The product of the eigenvalues of a matrix $\mathbf{A}$ is called the *determinant* of $\mathbf{A}$, or $\det(\mathbf{A})$.

$$\det(\mathbf{A}) = \lambda_1 \lambda_2 \cdots \lambda_n$$

If the determinant of a matrix is large, multiplying a vector by the matrix enlarges the volume swept out by the vector. If the determinant is small, the volume contracts.

Remember that the volume we're discussing here is the volume when a vector is plotted using the matrix's eigenvectors as axes.

18.5 Properties of the Determinant

The determinant is a powerful property of a matrix. Determinants can be easily calculated for a matrix and contain useful information about the matrix.

In MATLAB, the function `det` calculates the determinant of a matrix.

Let's say a vector $\mathbf{y} = \mathbf{Ax}$. We know the determinant of the matrix $\mathbf{A}$ is the ratio of the volumes of $\mathbf{x}$ and $\mathbf{y}$.

$$\det(\mathbf{A}) = \frac{\text{volume}(\mathbf{Ax})}{\text{volume}(\mathbf{x})} = \frac{\text{volume}(\mathbf{y})}{\text{volume}(\mathbf{x})}$$

If the inverse of $\mathbf{A}$ exists, we know that $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$. Thus, the determinant of the inverse matrix $\mathbf{A}^{-1}$ is

$$\det(\mathbf{A}^{-1}) = \frac{\text{volume}(\mathbf{A}^{-1}\mathbf{y})}{\text{volume}(\mathbf{y})} = \frac{\text{volume}(\mathbf{x})}{\text{volume}(\mathbf{y})} = \frac{1}{\det(\mathbf{A})}.$$

The determinant of $\mathbf{A}^{-1}$ is the inverse of the determinant of $\mathbf{A}$. If the determinant of a matrix is zero, this property indicates there is a problem with the inverse of the matrix.

$$\det(\mathbf{A}) = 0 \Rightarrow \det(\mathbf{A}^{-1}) = \frac{1}{\det(\mathbf{A})} = \frac{1}{0} \rightarrow \text{undefined}$$

Although we won't prove it in this course, **a matrix has an inverse if and only if the determinant of the matrix is nonzero**. The following statements are, in fact, equivalent for a square matrix $\mathbf{A}$:

- $\mathbf{A}$ can be transformed into the identity matrix by elementary row operations.
- The system $\mathbf{Ax} = \mathbf{y}$ is solvable and has a unique solution.
- $\mathbf{A}$ is full rank.
- $\mathbf{A}^{-1}$ exists.
- $\det(\mathbf{A}) \neq 0$.

A matrix with a determinant equal to zero has a geometric interpretation. Remember that the determinant is the product of the eigenvalues. It is the product of the scaling factors of the matrix along each eigenvector. If one of the eigenvalues is zero, we are missing information about how the matrix scales along at least one eigenvector. Our knowledge of the transformation is incomplete, which is why we cannot find a unique solution for the corresponding linear system.

Using the determinant we can concisely state our last field axiom. Recall that for scalars we required a multiplicative inverse exist for any nonzero member of the field, i.e.

$$\text{For all scalars } a \neq 0 \text{ there exists } a^{-1} \text{ such that } aa^{-1} = 1.$$

For vector spaces, we have the following:

$$\begin{gathered}\text{For all square matrices } \mathbf{A} \text{ where } \det(\mathbf{A}) \neq 0, \\ \text{there exists } \mathbf{A}^{-1} \text{ such that } \mathbf{AA}^{-1} = \mathbf{I} = \mathbf{A}^{-1}\mathbf{A}.\end{gathered}$$

Chapter 19

Matrix Decompositions

19.1 Eigendecomposition

Let's discuss a square, $n \times n$ matrix $\mathbf{A}$. Provided $\mathbf{A}$ is not defective, it has n linearly independent eigenvectors which we will call $\mathbf{v}_1, \ldots, \mathbf{v}_n$. The eigenvectors are linearly independent and therefore form a basis for $\mathbb{R}^n$ (an *eigenbasis*). We said in the last chapter that any vector $\mathbf{x}$ can be decomposed onto the eigenbasis by finding coefficients $a_1, \ldots, a_n$ such that

$$\mathbf{x} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n.$$

Multiplying the vector $\mathbf{x}$ by the matrix $\mathbf{A}$ is equivalent to scaling each term in the decomposition by the corresponding eigenvalue (λ_i).

$$\mathbf{A}\mathbf{x} = a_1\lambda_1\mathbf{v}_1 + a_2\lambda_2\mathbf{v}_2 + \cdots + a_n\lambda_n\mathbf{v}_n$$

We can think of matrix multiplication as a transformation with three steps.

1. Decompose the input vector onto the eigenbasis of the matrix.
2. Scale each term in the decomposition by the appropriate eigenvalue.
3. Reassemble, or "un-decompose" the output vector.

Each of these steps can be represented by a matrix operation. First, we collect the n eigenvalue into a matrix $\mathbf{V}$.

$$\mathbf{V} = (\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n)$$

Each column in the matrix $\mathbf{V}$ is an eigenvector of the matrix $\mathbf{A}$. To decompose the vector $\mathbf{x}$ onto the columns of $\mathbf{V}$ we find the coefficients $a_1, \ldots, a_n$ by solving the linear system

$$\mathbf{Va} = \mathbf{x}$$

where $\mathbf{a}$ is a vector holding the coefficients $a_1, \ldots, a_n$. The matrix $\mathbf{V}$ is square and has linearly independent columns (the eigenvectors of $\mathbf{A}$), so its inverse exists. The coefficients for decomposing the vector $\mathbf{x}$ onto the eigenbasis of the matrix $\mathbf{A}$ are

$$\mathbf{a} = \mathbf{V}^{-1}\mathbf{x}.$$

If the inverse matrix $\mathbf{V}^{-1}$ decomposes a vector into a set of coefficients $\mathbf{a}$, then multiplying the coefficients vector $\mathbf{a}$ by the original matrix must reassemble the vector $\mathbf{x}$. Looking back at the three steps we defined above, we can use multiplication by $\mathbf{V}^{-1}$ to complete step 1 and multiply by $\mathbf{V}$ to perform step 3. For step 2, we need to scale the individual coefficients by the appropriate eigenvalues. We define a scaling matrix $\mathbf{\Lambda}$ as a diagonal matrix of the eigenvalues:

In other words, if $\mathbf{V}^{-1}$ decomposes a vector, $(\mathbf{V}^{-1})^{-1} = \mathbf{V}$ must undo the decomposition.

We use the uppercase Greek lambda ($\mathbf{\Lambda}$) to denote the matrix of eigenvalues λ_i (lowercase lambda).

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}.$$

Notice that

$$\mathbf{\Lambda a} = \begin{pmatrix} \lambda_1 a_1 \\ \lambda_2 a_2 \\ \vdots \\ \lambda_n a_n \end{pmatrix}$$

so the matrix $\mathbf{\Lambda}$ scales the ith entry of the input vector by the ith eigenvalue.

We now have matrix operations for decomposing onto an eigenbasis ($\mathbf{V}^{-1}$), scaling by eigenvalues ($\mathbf{\Lambda}$), and reassembling the output vector ($\mathbf{V}$). Putting everything together, we see that matrix multiplication can be expressed as an *eigendecomposition* by

Eigendecomposition is the last time we will use the prefix "eigen-". Feel free to use it on other everyday words to appear smarter.

$$\mathbf{Ax} = \mathbf{V\Lambda V}^{-1}\mathbf{x}.$$

Equivalently, we can say that the matrix $\mathbf{A}$ itself can be as the product of three matrices ($\mathbf{A} = \mathbf{V\Lambda V}^{-1}$) if $\mathbf{A}$ has a complete set of eigenvectors. There are two ways to interpret the dependence on a complete set of eigenvectors. Viewed technically, the matrix $\mathbf{V}$ can only be inverted if it is full rank, so $\mathbf{V}^{-1}$ does not exist if one or more eigenvectors is missing. More intuitively, the eigendecomposition defines

a unique mapping between the input and output vectors. Uniqueness requires a basis, since a vector decomposition is only unique if the set of vectors form a basis. If the matrix $\mathbf{A}$ is defective, its eigenvectors do not form an eigenbasis and there cannot be a unique mapping between inputs and outputs.

19.2 Singular Value Decomposition

The eigendecomposition is limited to square matrices with a complete set of eigenvectors. However, the idea that matrices can be factored into three operations (decomposition, scaling, and reassembly) generalizes to all matrices, even non-square matrices. The generalized equivalent of the eigendecomposition is called the *Singular Value Decomposition*, or (SVD).

Singular Value Decomposition. *Any $m \times n$ matrix $\mathbf{A}$ can be factored into the product of three matrices*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$$

where

- $\mathbf{U}$ *is an orthogonal $m \times m$ matrix.*
- $\mathbf{\Sigma}$ *is a diagonal $m \times n$ matrix with nonzero entries.*
- $\mathbf{V}$ *is an orthogonal $n \times n$ matrix.*

The square matrices $\mathbf{U}$ and $\mathbf{V}$ are *orthogonal*, i.e. their columns form an orthonormal set of basis vectors. As we discussed previously, the inverse of an orthogonal matrix is simply its transpose, so $\mathbf{U}^{-1} = \mathbf{U}^\mathsf{T}$ and $\mathbf{V}^{-1} = \mathbf{V}^\mathsf{T}$. The $\mathbf{V}^\mathsf{T}$ term in the decomposition has the same role as the $\mathbf{V}^{-1}$ matrix in an eigendecomposition — projection of the input vector onto a new basis. The matrix $\mathbf{U}$ in SVD reassembles the output vector analogous to the vector $\mathbf{V}$ in an eigendecomposition.

If the entries in $\mathbf{A}$ were complex numbers, the matrices $\mathbf{U}$ and $\mathbf{V}$ would be *unitary*. The inverse of a unitary matrix is the complex conjugate of the matrix transpose.

The matrix $\mathbf{\Sigma}$ is diagonal but not necessarily square. It has the same dimensions as the original matrix $\mathbf{A}$. For a 3×5 matrix, the $\mathbf{\Sigma}$ has the form

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 \end{pmatrix}.$$

If the matrix $\mathbf{A}$ was 5×3, we would have

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The elements along the diagonal of $\mathbf{\Sigma}$ are called *singular values*. If $\mathbf{A}$ is an $m \times n$ matrix, the maximum number of nonzero singular values is $\min\{m, n\}$. The are the analogues of eigenvalues for non-square matrices. However, the singular values for a square matrix are not equal to the eigenvalues of the same matrix. Singular values are always nonnegative. If we arrange $\mathbf{\Sigma}$ such that the singular values are in descending order, the SVD of a matrix is unique.

The columns in $\mathbf{U}$ and $\mathbf{V}$ are called the left and right *singular vectors*, respectively. Just as there is a relationship between eigenvalues and eigenvectors, the columns in $\mathbf{U}$ and $\mathbf{V}$ are connected by the singular values in $\mathbf{\Sigma}$. If $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$, then

$$\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i$$

and

$$\mathbf{A}^\mathsf{T}\mathbf{u}_i = \sigma_i \mathbf{v}_i$$

where $\mathbf{v}_i$ is ith right singular vector (the ith column in $\mathbf{V}$); $\mathbf{u}_i$ is the ith left singular vector (the ith column in $\mathbf{U}$); and σ_i is the ith singular value (the ith nonzero on the diagonal of $\mathbf{\Sigma}$).

19.3 Applications of the SVD

19.3.1 Rank of a matrix

The rank of a matrix $\mathbf{A}$ is equal to the number of nonzero singular values (the number of nonzero values along the diagonal of $\mathbf{\Sigma}$). This is true for both square and nonsquare matrices. Notice that the way we defined the diagonal matrix $\mathbf{\Sigma}$ implies that the number of singular values must be at most $\min\{m, n\}$ for an $m \times n$ matrix. This requirement agrees with our knowledge that $\text{rank}(\mathbf{A}) \leq \min\{m, n\}$.

19.3.2 The matrix pseudoinverse

Our definition of a matrix inverse applies only to square matrices. For nonsquare matrices we can use the SVD to construct a pseudoinverse. We represent the pseudoinverse of a matrix $\mathbf{A}$ as $\mathbf{A}^+$. We simply reverse and invert the factorization of $\mathbf{A}$, i.e.

$$\mathbf{A}^+ = (\mathbf{V}^\mathsf{T})^{-1}\mathbf{\Sigma}^+\mathbf{U}^{-1}$$

We can simplify this expression with knowledge that $\mathbf{V}$ and $\mathbf{U}$ are orthogonal, so $(\mathbf{V}^\mathsf{T})^{-1} = (\mathbf{V}^\mathsf{T})^\mathsf{T} = \mathbf{V}$ and $\mathbf{U}^{-1} = \mathbf{U}^\mathsf{T}$. Thus

$$\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^\mathsf{T}.$$

The matrix $\mathbf{\Sigma}^+$ is the pseudoinverse of the diagonal matrix $\mathbf{\Sigma}$. This is simply the transpose of $\mathbf{\Sigma}$ where each entry on the diagonal is replaced by its multiplicative inverse. For example, a 3×5 matrix $\mathbf{\Sigma}$:

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 \end{pmatrix}$$

the pseudoinverse $\mathbf{\Sigma}^+$ is

$$\mathbf{\Sigma}^+ = \begin{pmatrix} 1/\sigma_1 & 0 & 0 \\ 0 & 1/\sigma_2 & 0 \\ 0 & 0 & 1/\sigma_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Chapter 20

Low Rank Approximations

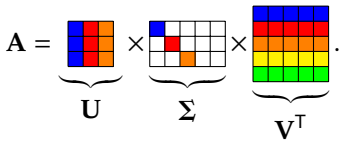
The previous chapter introduced the Singular Value Decomposition (SVD) and showed how every matrix can be decomposed into the product of three matrices. Any $m \times n$ matrix $\mathbf{A}$ is equal to $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ where

1. An $m \times m$ orthogonal matrix $\mathbf{U}$ that forms a basis for the output dimension.
2. An $m \times n$ diagonal matrix $\mathbf{\Sigma}$ that holds the singular values.
3. An $n \times n$ orthogonal matrix $\mathbf{V}$ that forms a basis for the input dimension.

We can use the SVD to help us understand how multiplication transforms vectors between dimensions. Assume that a matrix $\mathbf{A}$ has dimensions 3×5. If $\mathbf{y} = \mathbf{A}\mathbf{x}$, the input vector $\mathbf{x}$ is 5-dimensional and the output vector $\mathbf{y}$ is 3-dimensional. The input vector $\mathbf{x}$ lost two dimensions somewhere in the matrix $\mathbf{A}$. Where did they go?

Let's look at the SVD of the matrix $\mathbf{A}$. We're not interested in the particular numbers in the matrices—just the overall structure. Visually,

We're showing $\mathbf{V}^T$ in this diagram, so the column vectors in $\mathbf{V}$ appear as row vectors in $\mathbf{V}^T$.



There's lots to analyze here. First, only the colored entries contain nonzero numbers. The white squares in $\mathbf{\Sigma}$ and zero since the singular values only appear along the diagonal. Also note that the number of singular values never exceeds the *smaller* dimension of the original matrix. Since $\mathbf{A}$ is a 3×5 matrix, there can be no more than three nonzero singular values.

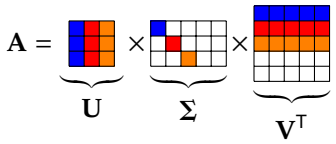
The columns of zeros in the matrix $\mathbf{\Sigma}$ will "zero-out" the last two rows (green and yellow) of the matrix $\mathbf{V}^T$ during multiplication. Thus the green and yellow rows do not contribute at all to the output vector. This is how the matrix $\mathbf{A}$ moves from five to three dimensions:

1. The input vector is decomposed onto the 5-dimensional basis formed by the columns in $\mathbf{V}$. Normally this decomposition requires multiplying the input vector by the inverse $\mathbf{V}^{-1}$; however, since $\mathbf{V}$ is an orthogonal matrix, its inverse equals its transpose.

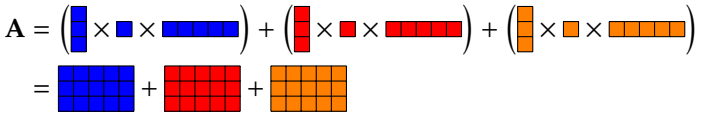
2. Multiplying by $\mathbf{\Sigma}$ scales the first three dimensions by the corresponding singular values. The last two dimensions are dropped.

3. The surviving three dimensions are projected into the output space by the basis vectors in the matrix $\mathbf{U}$.

Since the green and yellow rows in $\mathbf{V}^T$ are going to be zeroed-out, let's change their color to white to match the zeros in the matrix $\mathbf{\Sigma}$. Then they'll be easier to ignore.



The second thing to note is that the blue entries in all three matrices are only multiplied by other blue entries. The same is true for the red and orange entries. You can follow through the multiplication and separate it into three parts, one blue, one red, and one orange.



So the matrix $\mathbf{A}$ is actually the sum of three separate matrices—one blue, one red, and one orange. The blue column in the matrix $\mathbf{U}$ is the first left singular vector ($\mathbf{u}_1$). The blue square in $\mathbf{\Sigma}$ is the first singular value (σ_1). The blue row in the matrix $\mathbf{V}^T$ is actually a column vector in the untransposed matrix $\mathbf{V}$ and is the first right singular vector ($\mathbf{v}_1$).

Let's write this all out again using mathematical symbols instead of boxes for the matrices. We'll retain the same color scheme as a guide and drop rows 4–5 in

$\mathbf{V}^T$ since these rows are zeroed-out in the final product.

$$\mathbf{A} = \underbrace{\begin{pmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3\end{pmatrix}}_{\mathbf{U}} \underbrace{\begin{pmatrix}\sigma_1 & & \\ & \sigma_2 & \\ & & \sigma_3\end{pmatrix}}_{\mathbf{\Sigma}} \underbrace{\begin{pmatrix}\mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T\end{pmatrix}}_{\mathbf{V}^T}$$

$$= \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \sigma_3 \mathbf{u}_3 \mathbf{v}_3^T$$

The SVD decomposes any matrix into a weighted sum of matrices created by the pairs of singular vectors $\mathbf{u}_i \mathbf{v}_i^T$. The weights in this sum are the singular values σ_i. These scalars define how much each pair of singular vectors ($\mathbf{u}_i$ and $\mathbf{v}_i$) contribute to the matrix. Some pairs have large singular values and therefore contribute a lot. Pairs with small singular values contribute very little. As an extreme, any pair of singular vectors associated with a zero singular value *does not contribute at all*.

The singular values in the matrix $\mathbf{\Sigma}$ are by convention ordered by decreasing magnitude ($\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$). Since the singular vectors are the weights for the singular vectors, the singular vectors are similarly ordered by decreasing importance. The first pair of singular vectors $\mathbf{u}_1$, $\mathbf{v}_1$ has the largest influence on the overall matrix, followed by the second pair $\mathbf{u}_2$, $\mathbf{v}_2$, and so on. Some pairs of singular vectors are associated with such a small singular value that they can be removed with little effect, as demonstrated by the following numerical example.

Remember that singular values are always nonnegative.

In this example, we used the SVD to decompose a 4×3 matrix. Since the output dimension (4) is larger than the input dimension (3), the matrix $\mathbf{U}$ contains "extra" columns that are zeroed-out by the matrix $\mathbf{\Sigma}$.

$$\mathbf{A} = \begin{pmatrix} 0.67 & 0.99 & 0.61 \\ 0.70 & 0.13 & 0.81 \\ 0.12 & 0.18 & 0.11 \\ 0.24 & 0.77 & 0.12 \end{pmatrix}$$

$$= \underbrace{\begin{pmatrix} -0.75 & -0.23 & 0.62 & 0.03 \\ -0.51 & 0.79 & -0.33 & 0.09 \\ -0.14 & -0.04 & -0.15 & -0.98 \\ -0.39 & -0.57 & -0.70 & 0.18 \end{pmatrix}}_{\mathbf{U}} \underbrace{\begin{pmatrix} 1.76 & 0 & 0 \\ 0 & 0.75 & 0 \\ 0 & 0 & 0.01 \\ 0 & 0 & 0 \end{pmatrix}}_{\mathbf{\Sigma}} \underbrace{\begin{pmatrix} -0.55 & 0.33 & -0.76 \\ -0.64 & -0.75 & 0.13 \\ -0.53 & 0.57 & 0.63 \end{pmatrix}}_{\mathbf{V}^T}$$

Notice how small the third singular value is ($\sigma_3 = 0.01$) compared to the other two singular values ($\sigma_1 = 1.76$ and $\sigma_2 = 0.75$). The singular vectors associated with σ_3 (in orange) contribute little to the overall matrix $\mathbf{A}$ because they are multiplied by such a small singular value. We could probably change the singular value to zero and not change the matrix much. Let's delete σ_3 and call the resulting matrix $\mathbf{A}_2$ since it contains only two of the three original singular values.

Notice that we call the inner matrix $\mathbf{\Sigma}_2$ since we've modified it to only have two singular values.

$$\mathbf{A}_2 = \underbrace{\begin{pmatrix} -0.75 & -0.23 & 0.62 & 0.03 \\ -0.51 & 0.79 & -0.33 & 0.09 \\ -0.14 & -0.04 & -0.15 & -0.98 \\ -0.39 & -0.57 & -0.70 & 0.18 \end{pmatrix}}_{\mathbf{U}} \underbrace{\begin{pmatrix} 1.76 & 0 & 0 \\ 0 & 0.75 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{\mathbf{\Sigma}_2} \underbrace{\begin{pmatrix} -0.55 & 0.33 & -0.76 \\ -0.64 & -0.75 & 0.13 \\ -0.53 & 0.57 & 0.63 \end{pmatrix}}_{\mathbf{V}^\mathsf{T}}$$

$$= \begin{pmatrix} 0.67 & 0.99 & 0.61 \\ 0.70 & 0.13 & 0.81 \\ 0.12 & 0.18 & 0.11 \\ 0.24 & 0.77 & 0.12 \end{pmatrix}$$

Deleting the small singular value σ_3 had almost no effect on the matrix. The matrices $\mathbf{A}$ and $\mathbf{A}_2$ appear identical due to rounding, but there are some small differences. To emphasize the connection between the singular values and the corresponding singular vectors, we can also zero-out the orange values in $\mathbf{U}$ and $\mathbf{V}$ since they are only multiplied by the singular value σ_3 that we've set to zero. While we're at it, we can zero-out the fourth column in $\mathbf{U}$ which is never used because there are only three singular values for the 4×3 matrix.

$$\mathbf{A}_2 = \underbrace{\begin{pmatrix} -0.75 & -0.23 & 0 & 0 \\ -0.51 & 0.79 & 0 & 0 \\ -0.14 & -0.04 & 0 & 0 \\ -0.39 & -0.57 & 0 & 0 \end{pmatrix}}_{\mathbf{U}_2} \underbrace{\begin{pmatrix} 1.76 & 0 & 0 \\ 0 & 0.75 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{\mathbf{\Sigma}_2} \underbrace{\begin{pmatrix} -0.55 & 0.33 & -0.76 \\ -0.64 & -0.75 & 0.13 \\ 0 & 0 & 0 \end{pmatrix}}_{\mathbf{V}_2^\mathsf{T}}$$

$$= \begin{pmatrix} 0.67 & 0.99 & 0.61 \\ 0.70 & 0.13 & 0.81 \\ 0.12 & 0.18 & 0.11 \\ 0.24 & 0.77 & 0.12 \end{pmatrix}$$

Matrices like $\mathbf{A}_2$ are called *low rank approximations* to the original matrix $\mathbf{A}$. In section 19.3 we said that the rank of a matrix is equal to the number of nonzero

singular values in its SVD. Zeroing-out a singular value creates a new matrix with a lower rank. In the example above, the matrix $\mathbf{A}_2$ is a "rank 2" approximation to the matrix $\mathbf{A}$.

We always delete the smallest singular values when creating a low rank approximation. This ensures the smallest loss of information when reconstructing the original matrix. In the matrix $\mathbf{A}$ from above, the third singular value was so small that its deletion was almost unnoticeable.

How small is small for a singular value? It all depends on the size of the other singular values. We judge the size of a singular value relative to the others. For the matrix $\mathbf{A}$, the sum of the singular values was $1.76 + 0.75 + 0.01 = 2.52$, so the third singular value represented only $0.01/2.52 = 0.4\%$ of the information in the matrix. We can use singular values to assess the information distribution of a matrix since the singular vectors have been normalized. Only the singular values determine the relative contribution of the products $\sigma_i \mathbf{u}_i \mathbf{v}_i^\mathsf{T}$ that make up the original matrix.

20.1 Image Compression

A low rank approximation for a matrix contains several rows and columns of zeros. If a singular value in the matrix $\mathbf{\Sigma}$ is zero, the corresponding column in $\mathbf{U}$ and the corresponding row in $\mathbf{V}^\mathsf{T}$ can also be set to zero as shown in the previous section. We can go a step further and remove the columns of zeros entirely. The number of rows in $\mathbf{U}$ and the number of columns in $\mathbf{V}^\mathsf{T}$ determine the dimensions of the product $\mathbf{U\Sigma V}^\mathsf{T}$, so deleting a column from $\mathbf{U}$ or a row from $\mathbf{V}^\mathsf{T}$ will not change the dimensions of the product. A low rank approximation requires less memory to store on a computer and can be used as a form of image compression.

Before we see an example of image compression, we should mention that there are two types of compression. Low rank approximations are *lossy* compression schemes since they destroy information that can't be recovered. However, removing small singular values deletes only a small amount information, so the compressed image retains most of its features. *Lossless* compression retains all the information in an image by finding ways to store the image more efficiently. For example, a lossless compressor might replace frequent sequences of bits with an abbreviation to reduce the image size. We don't talk about lossless compression in this book, but many of these algorithms also use linear algebra.

A digital image can be represented as a matrix of pixels. For simplicity we'll discuss greyscale (black and white) images where each pixel is a number between zero (black) and one (white). Color images have either three (RGB) or four (CMYK) entries for each pixel to describe the color intensities.

Figure 20.1: A 512×384 pixel, grayscale image of two adorable Scottie dogs. The upper dog is Duncan. The lower dog is Rocky. They are best friends.

Figure 20.1 is an image of two Scottish Terriers looking out a window. The image is 512 pixels high and 384 pixels wide, creating a 512×384 matrix. The rank of this matrix can be no larger than $\min\{512, 384\} = 384$, so the SVD of the image matrix will have at most 384 nonzero singular values.

Let's compress this image using a low rank approximation with the following steps on the image matrix $\mathbf{A}$.

1. We decompose the matrix $\mathbf{A}$ using the SVD: $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$.
2. We keep only the k largest singular values remove the zeroed-out columns in $\mathbf{U}$ and $\mathbf{V}$ to create smaller matrices $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, and $\mathbf{V}_k$.
3. We multiply these matrices together to form a low rank approximation for the original image $\mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\mathsf{T}$.

Figure 20.2 shows six low rank approximations for the image in Figure 20.1. The first image is actually a full rank reconstruction since it uses all 384 singular values. The second image uses the largest 96 singular values, so it contains only a fraction of the information in the first image; however, the low rank approximation is almost indistinguishable from the full image. It is only at very low ranks where the image becomes blurry, although the dogs are still recognizable with only 12 singular values.

There are two reasons why we can compress an image while retaining its overall features. The first is that most of the information in an image comes from a few singular vectors. The distribution of information is skewed as shown in Figure 20.3. The smallest singular vectors give diminishing returns and contribute little to the overall information content of a matrix.

The second reason why we can compress images is that information is distributed. Each pair of singular vectors holds information about every pixel, which is expected since the product $\sigma_i\mathbf{u}_i\mathbf{v}_i$ is a matrix with the same dimensions as the original image. Figure 20.4 gives a visual representation for each pair of singular vectors. Each panel was created by zeroing-out all but one of the singular values. Most of the image has been reconstructed by the time reach the final (smallest) singular values, so representations of these singular vectors often amount to little more than noise. Many machine learning algorithms exploit this feature by "de-noising" inputs by zeroing-out any small singular vectors before using the images.

20.1.1 When does compression save space?

You might have noticed in Figure 20.2 that the SVD with all 384 singular values used *more* memory than the original image (175.2% of the original image size). To

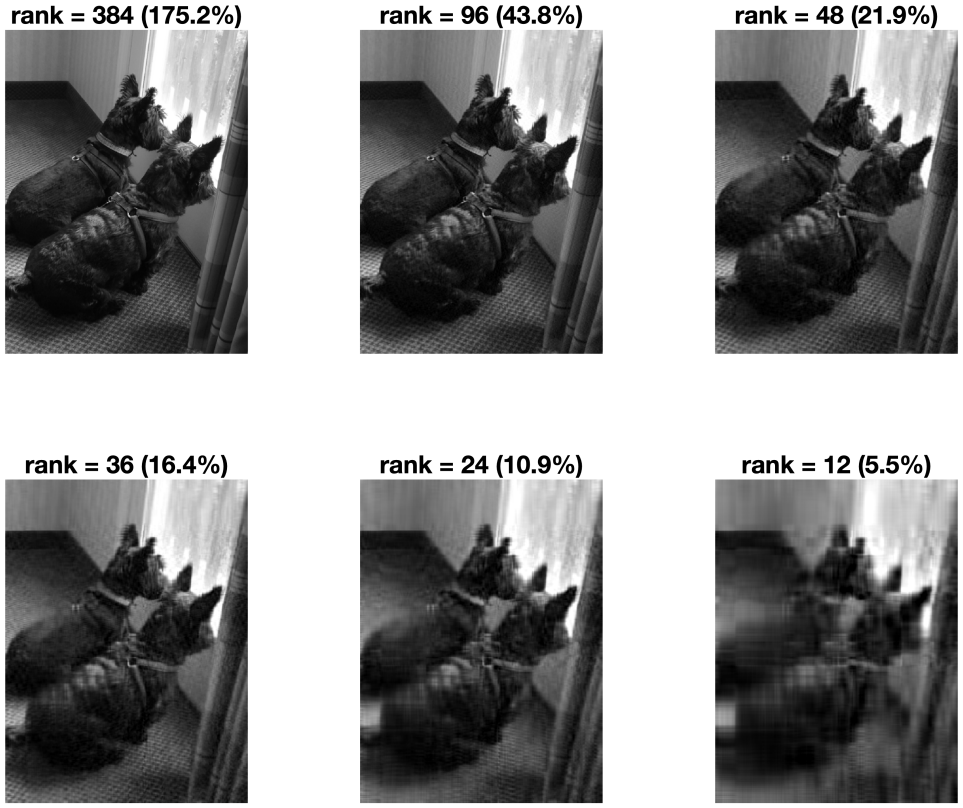


Figure 20.2: Low rank approximations for the image in Figure 20.1. The title of each panel shows the number of singular values used in the approximation (k) and the size of the compressed image relative to the original (%).

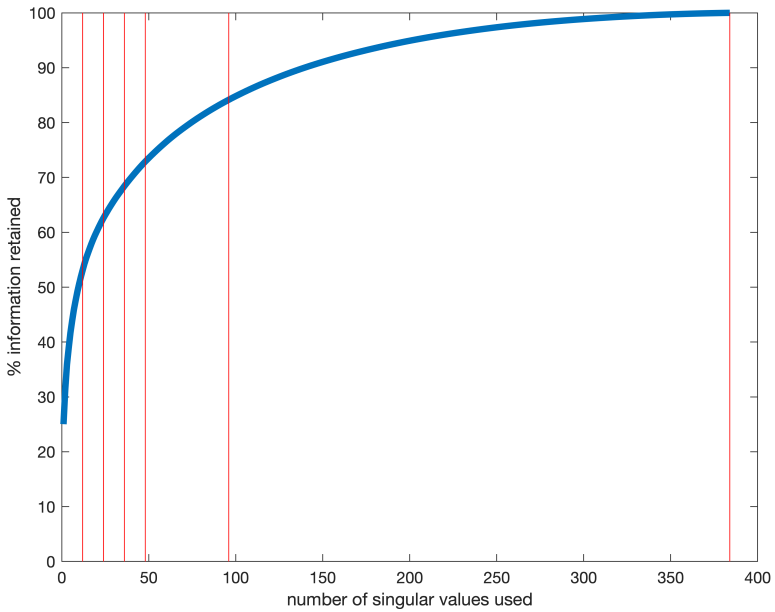


Figure 20.3: The cumulative sum of the singular values in Figure 20.1 reveals that most of the information is stored in the first few singular vectors. The red lines indicate the low rank approximations shown in Figure 20.2.

see why, consider an $m \times n$ image with $m \leq n$. (If m is the larger dimension, we can rotate the image so m is smaller.) This image requires mn units of memory, but the SVD requires

$$\underbrace{m \times m}_{\mathbf{U}} + \underbrace{m \times n}_{\mathbf{\Sigma}} + \underbrace{n \times n}_{\mathbf{V}^\mathsf{T}} .$$

units. Actually, not quite. Since $m < n$, we know that the extra $n - m$ rows in the matrix $\mathbf{V}^\mathsf{T}$ will be dropped, so we don't need to store them. (MATLAB's `svd` command removes these rows by default.) Also, The matrix $\mathbf{\Sigma}$ is diagonal, so we only need to store the m nonzero values on the diagonal. The total storage for the full SVD is therefore $m^2 + m + mn$ units of memory, which is larger than the mn units in the original image.

A low rank approximation with only k nonzero singular values keeps only the first k columns of $\mathbf{U}$, k entries in $\mathbf{\Sigma}$, and the first k rows of $\mathbf{V}^\mathsf{T}$. This requires $mk + k + kn = k(m + n + 1)$ units of storage. Compared to the mn units of memory required to store the original image, the rank k approximation by SVD requires

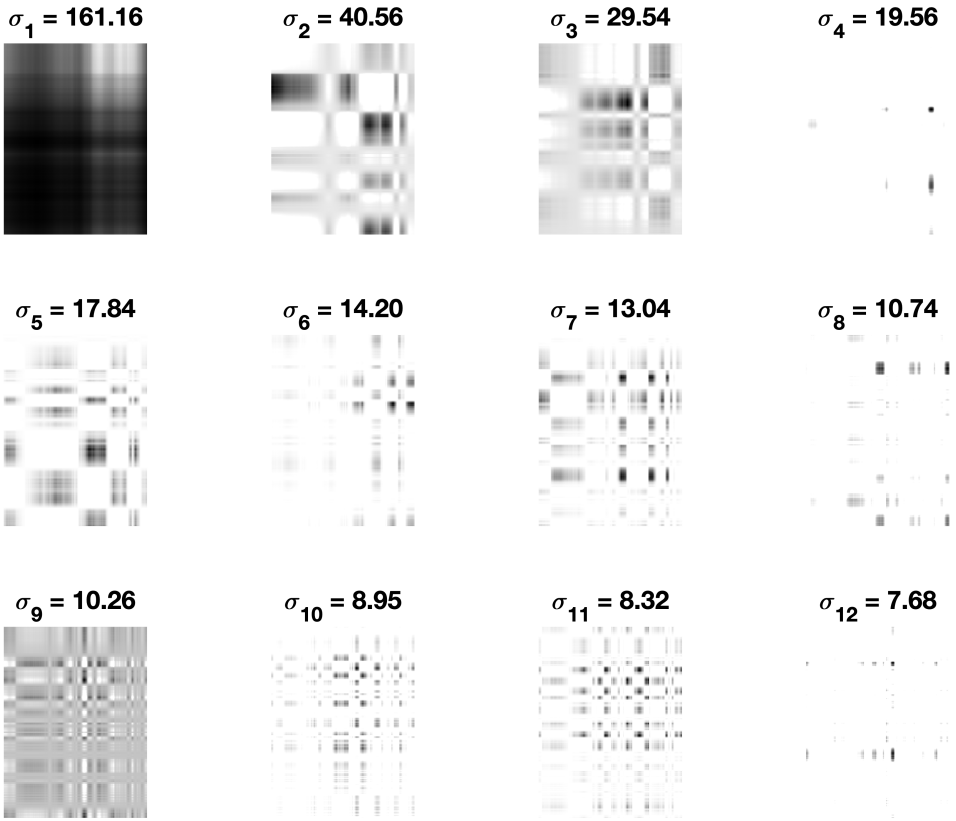


Figure 20.4: Visual representations of each pair of singular vectors. Each panel is the product $\sigma_i \mathbf{u}_i \mathbf{v}_i$ for a single index i. Notice how some of the panels map to rough shapes in the original image. The first panel (σ_1) shows the light from the window. The panels have been rescaled to $[0, 1]$ before plotting.

less memory when

$$k(m+n+1) < mn$$
$$\Rightarrow k < \frac{mn}{m+n+1}.$$

20.2 Recommender Systems

Here's a trick question: What is the missing value in the following matrix?

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 2 & 4 \\ 3 & 0 & ? \end{pmatrix}$$

It's a trick question because the missing value could be anything. Any answer is correct! But what if you also knew that the above matrix had rank 2? That's an entirely different story. There are three rows in the matrix, but only two of them would be linearly independent. Any row in the rank 2 matrix must be a linear combination of the other two rows. It appears that row three is twice the first row minus the second row ($2 \times 2 - 1 = 3$ in column one, and $2 \times 1 - 2 = 0$ in column two). So the missing entry must be $2 \times 3 - 4 = 2$. Mystery solved.

Rank deficient matrices like the one above have a remarkable property—a low rank approximation can sometimes be used to fill in missing entries. Filling in missing entries using low rank approximations is called *matrix completion*, and it's a machine learning technique worth billions of dollars. These machine learning techniques predict what a customer will want next given their history of purchases. Amazon shows you products that "you may also like"; Google targets ads based on your search results and Gmail messages; and Netflix populates its frontpage with shows it thinks you'll enjoy. Such recommendations are not guesses, but instead finely tuned algorithms that keep users buying products or spending time on streaming services. The best algorithms are worth huge sums of money, as evidenced by the "Netflix challenge," the company's million dollar contest to improve their recommender engine by 10%. The Netflix challenge was quickly shut down due to privacy concerns. A user's past preferences are so powerful that they can be used to reveal their identify even if the data are anonymized.

20.2.1 Ratings matrices

Online retailers like Amazon collect product ratings from users. Consumers can leave product reviews and rate items with 1–5 stars. We can visualize these data

as a ratings matrix like the one below, where each row is a user and each column is an item for sale.

	item 1	item 2	item 3	item 4	item 5
user 1	4		5		
user 2		1			5
user 3	3			5	
user 4		2	3		4

Ignore for a second that Amazon probably has more than four users and five items. Focus instead on how powerful it would be to have the complete ratings matrix. If we could predict how each user would rate every product before they buy it, we could suggest that they purchase the items with high ratings. Completing the ratings matrix creates personalized recommendations for every user! Systems that create recommendations through matrix completion are called *recommender systems*, and they are in use by every retailer and social media site.

There are many methods for matrix completion, but one of the most popular combines several techniques from this book. The algorithm is depicted in Figure 20.5 and is described below. A detailed description of the algorithm is available in section 20.2.4.

1. A ratings matrix $\mathbf{R}$ with m users and n items contains actual ratings for a small number of user/item pairs. The recommender randomly initializes two rank k matrices: $\mathbf{P}_k$ $(m \times k)$ and $\mathbf{Q}_k^\intercal$ $(k \times n)$.

2. The two matrices $\mathbf{P}_k$ and $\mathbf{Q}_k^\intercal$ are multiplied to produce a completed ratings matrix $\hat{\mathbf{R}}$. The complete matrix contains estimates for every user/item pair.

3. The known entries in the original ratings matrix $\mathbf{R}$ are compared with the corresponding predictions in the completed matrix $\hat{\mathbf{R}}$. The disagreement between the actual and predicted ratings are used to update the entires in the low rank approximations $\mathbf{P}_k$ and $\mathbf{Q}_k^\intercal$.

4. The process iterates until the predicted ratings match the observed ratings. The predictions for the unobserved ratings are used to make recommendations.

Recommender systems have become so ubiquitous that some databases are optimized for performing matrix algebra on their data. Thanks to recommender systems, matrix multiplications, linear models, and low rank approximations have become the fundamental operations of data mining.

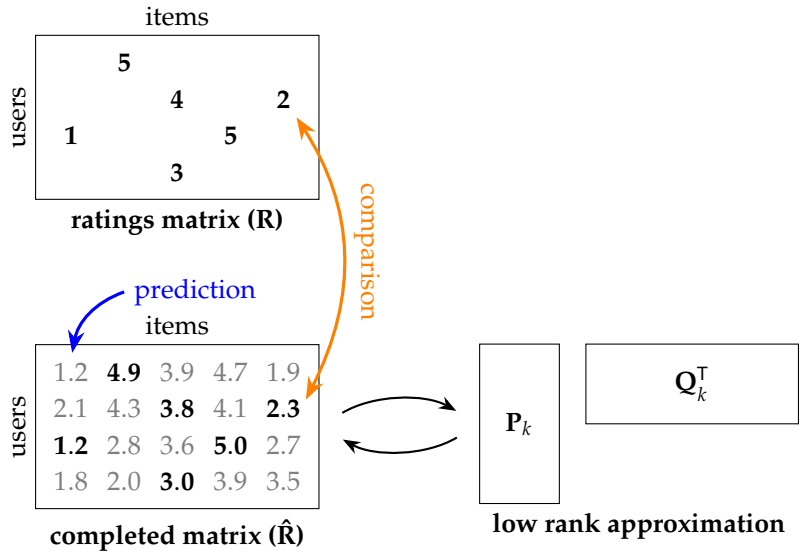


Figure 20.5: Recommender systems use low rank approximations to complete a ratings matrix. The method assumes that the completed ratings matrix is the product of smaller two matrices $\mathbf{P}$ and $\mathbf{Q}^\mathsf{T}$, each of rank k. Recommender systems search for values of the low rank matrices by comparing the known values in the ratings matrix to the corresponding predictions in the completed matrix.

20.2.2 Implicit vs. explicit ratings

Rating a product on a 1–5 scale is an *explicit* rating. Very few users provide this type of feedback, and you may be wondering how Amazon makes recommendations to you if you've never rated a product. Most recommender systems use *implicit* ratings from their users. Purchasing—or even viewing—an item is a form of rating, since users typically spend their money on things they like (or think they will like). Social media companies record how long you pause over each post when scrolling through your feed. The length of your pause is proportional to your interest in a post and can be used in a ratings matrix to predict what posts to show next. Every click, pause, or "like" on the internet is stored and used to predict ratings. Even brick-and-mortar stores use recommender systems, which is why they want you to join their store's loyalty card to track your purchases. (If you refuse, they can still track purchases across visits through your credit card number.)

The one shortcoming of recommender systems is the *cold start problem*. Before a company has any data on your preferences (implicit or explicit), they cannot make

recommendations. This explains why companies request demographic data when you sign up or frequently request ratings on your first purchases. The sooner they can collect data, the better they can customize their offerings to your preferences.

20.2.3 Why low rank approximations work

Ratings matrices are incredibly sparse. Most users have purchased only a tiny fraction of the products available on Amazon, and despite your best efforts at binge watching you have probably seen relatively few of the shows on Netflix. Recommender systems work because users and items are remarkably low rank. Even if Amazon has millions of customers, every customer can be reasonably approximated as a combination of a few thousand "customer types". A few thousand sub-genres are probably all that are necessary to make personalized recommendations for movies.

An interesting observation from recommender systems is that the number of user types must be equal to the number of categories of items. Said more technically, the user rank (row rank) must equal the item rank (column rank) for any ratings matrix, so the ranks of the matrices $\mathbf{P}$ and $\mathbf{Q}$ must be the same. Any method that finds similar users (rows of the ratings matrix) can be used equally well to find similar items (columns) by simply transposing the ratings matrix.

20.2.4 * Finding a low rank approximation for a ratings matrix

This section derives the update formulas used to find a low rank approximation for a ratings matrix $\mathbf{R}$ containing a sparse set of known ratings. We assume the user has specified a rank k for the low rank approximation—this is a hyperparameter that must be tuned to improve predictions.

The completed ratings matrix $\hat{\mathbf{R}}$ is the product of two dense, rank k matrices $\mathbf{P}_k$ and $\mathbf{Q}_k^\top$, so $\hat{\mathbf{R}} = \mathbf{P}_k\mathbf{Q}_k^\top$. Let's focus on a single known rating $\mathbf{R}(u,t)$ from user u for item t, which we'll abbreviate r_{ut}. The predicted value for this rating is

$$\hat{r}_{ut} = \mathbf{P}_k(u,:) \cdot \mathbf{Q}_k^\top(:,t).$$

We can eliminate the transpose on the matrix $\mathbf{Q}$ by remembering that column t in the transposed matrix is the same as row t in the untransposed matrix, so

$$\hat{r}_{ut} = \mathbf{P}_k(u,:) \cdot \mathbf{Q}_k(t,:).$$

The matrices $\mathbf{P}_k$ and $\mathbf{Q}_k$ are initialized with small, random values, so the predicted rating $\hat{r}_{ut}$ is likely not the same as the actual rating r_{ut}. We can update the entries

in $\mathbf{P}_k$ and $\mathbf{Q}_k$ by first calculating the loss between the predicted and actual ratings. Using a quadratic loss function, the loss for a single rating is

$$L(u,t) = (\hat{r}_{ut} - r_{ut})^2 .$$

Let's first update the entries in the matrix $\mathbf{P}_k$ using gradient descent. The entry of the gradient for entry p_{ui} of the matrix $\mathbf{P}_k$ is

$$\frac{\partial L}{\partial p_{ui}} = 2\left(\hat{r}_{ut} - r_{ut}\right)\frac{\partial \hat{r}_{ut}}{\partial p_{ui}}.$$

The partial derivative of the predicted rating is

$$\begin{aligned}\frac{\partial \hat{r}_{ut}}{\partial p_{ui}} &= \frac{\partial}{\partial p_{ui}}\mathbf{P}_k(u,:)\cdot\mathbf{Q}_k(t,:) \\ &= \frac{\partial}{\partial p_{ui}}\sum_{j=1}^{k} p_{uj}q_{tj} \\ &= q_{ti}.\end{aligned}$$

So the gradient of the loss for entry p_{ui} is

$$\frac{\partial L}{\partial p_{ui}} = 2\left(\hat{r}_{ut} - r_{ut}\right)q_{ti}.$$

Similarly, the gradient entry for value q_{ti} in the matrix $\mathbf{Q}_k$ is

$$\frac{\partial L}{\partial q_{ti}} = 2\left(\hat{r}_{ut} - r_{ut}\right)p_{ui}.$$

We can update the entries in $\mathbf{P}_k$ and $\mathbf{Q}_k$ using gradient descent with rules

$$\begin{aligned}p_{ui} &\leftarrow p_{ui} - 2\alpha(\hat{r}_{ut} - r_{ut})q_{ti} \\ q_{ti} &\leftarrow q_{ti} - 2\alpha(\hat{r}_{ut} - r_{ut})p_{ui}\end{aligned}$$

for entries $i \in \{1 \ldots k\}$. Each update uses an estimate of the total loss at a single entry in the observed ratings matrix $\mathbf{R}$. The true loss function would consider all of the known ratings, so our single-rating estimate is a *stochastic gradient*. Surprisingly, stochastic gradient descent is a better global optimizer for many non-convex machine learning problems and requires far less computation per iteration on large

problems. However, stochastic gradient descent requires more iterations than deterministic gradient descent, and the step size α often needs to be smaller to ensure stability.

Before starting gradient descent, a subset of the known ratings are held out for cross validation. Large systems with many users and items can easily be overfit because the low rank approximation will contain thousands or millions of entries. A regularization term is added to the loss function to prevent overfitting, and the best regularization parameter is determined by cross validation.

Recommender systems combine several topics from this course: matrix multiplication, rank, loss functions, gradient descent, cross validation, regularization, and matrix decompositions. They are an excellent example of how modern machine learning depends on a solid foundation in linear algebra.