MAT 180 FALL 2022

Mathematics of Machine Learning

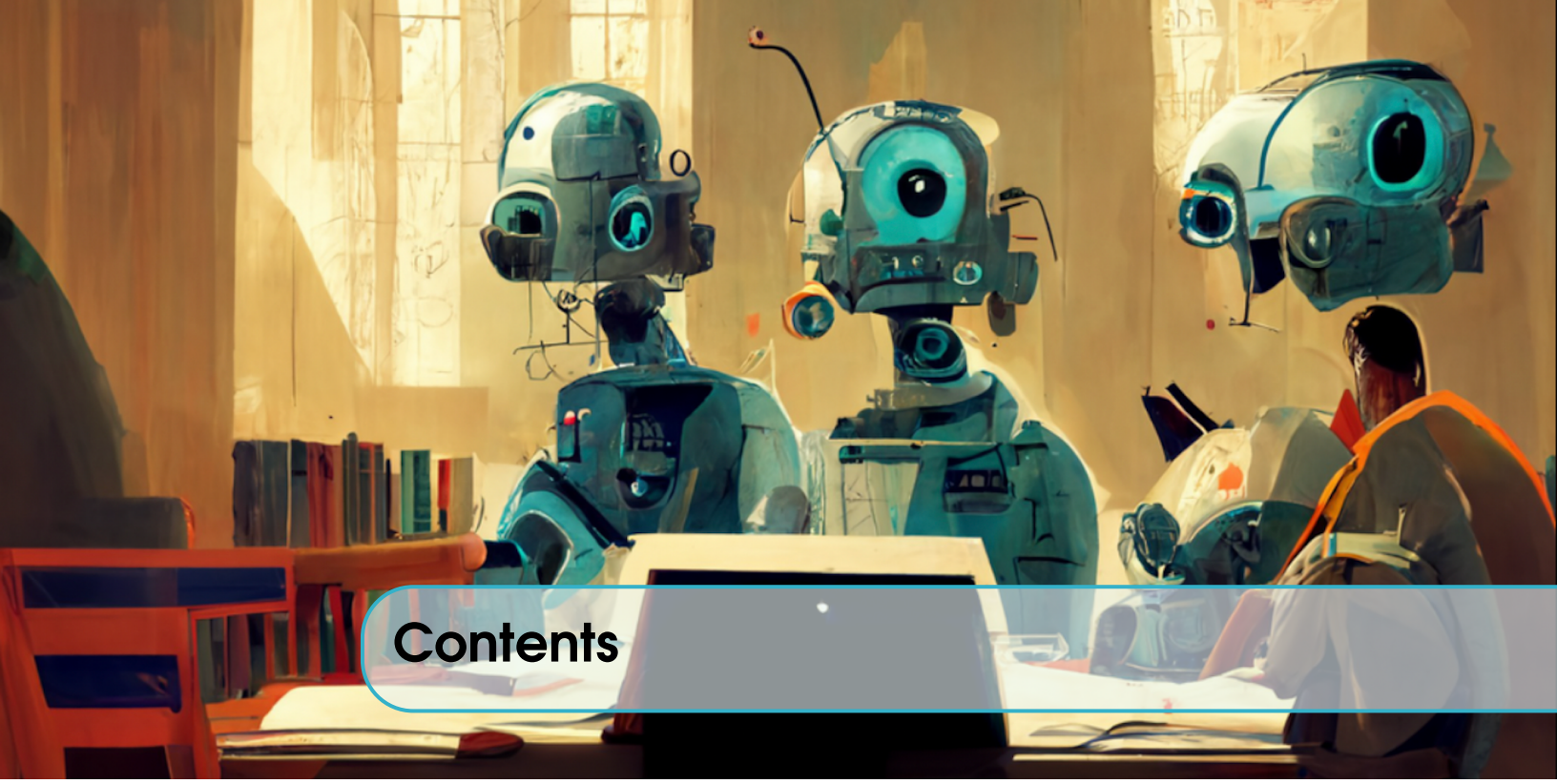
Lecture Notes

SPECIAL TOPICS COURSE, UNIVERSITY OF CALIFORNIA, DAVIS

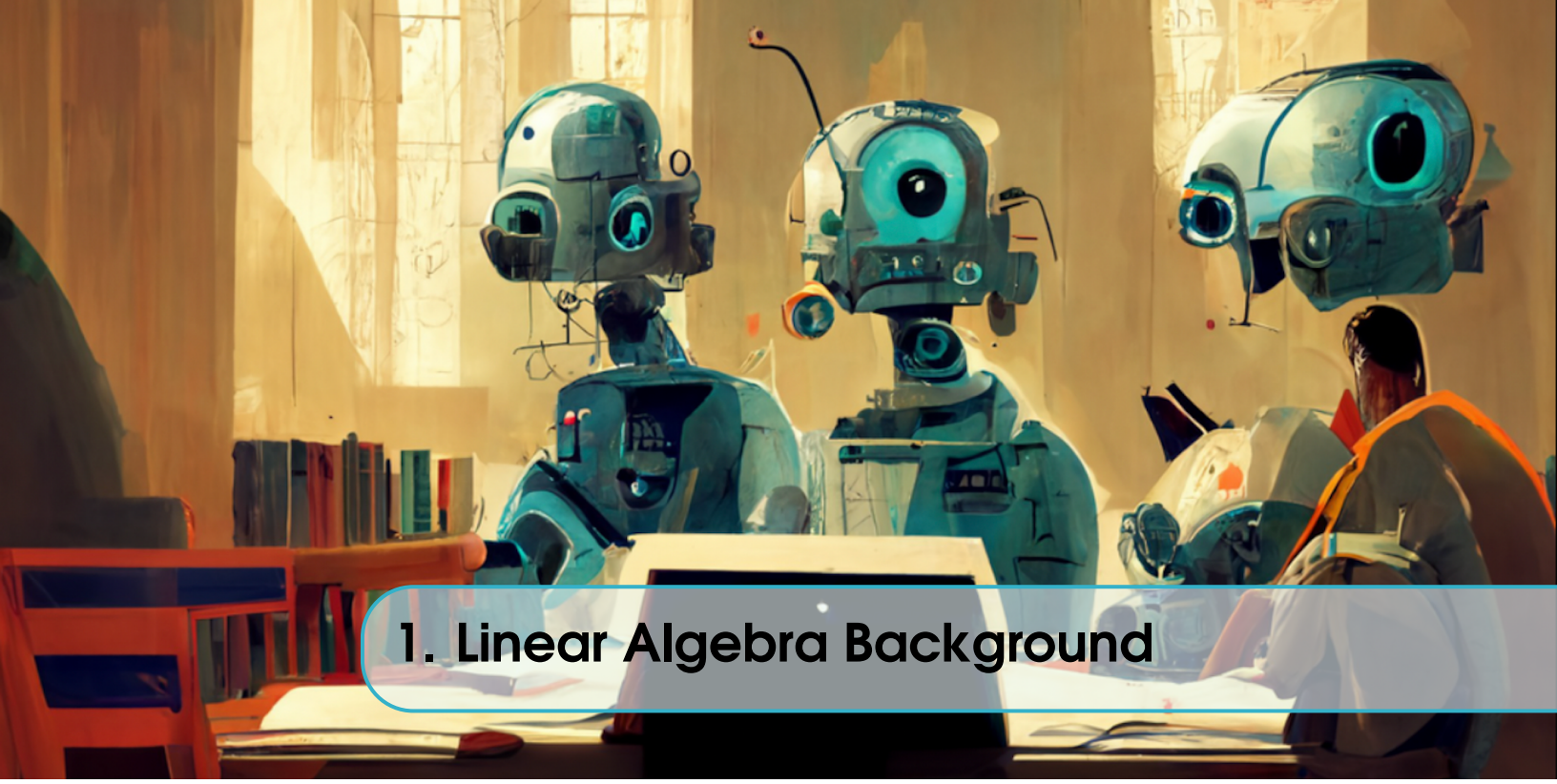
GITHUB.COM/ALEXCHANDLER100

This course was taught by Dr. Alex Chandler as MAT 180: The Mathematics of Machine Learning in the Fall 2022 Quarter at UC Davis with the aid of the teaching assistant Gregory DePaul.

First release, January 2023



Contents



1. Linear Algebra Background

In this chapter we go through a quick overview of the mathematical foundations needed in order to properly understand the fine details of deep learning. In Section 1.1 we recall the basics of vector spaces, norms, and linear maps. This leads us to the important concepts of eigenvalue and singular value decompositions in Section 1.3.

1.1 Vector Spaces and Linear Maps

We assume the reader has some familiarity with linear algebra so we simply review some of the major concepts needed to proceed, often without proof. The main object of study in linear algebra is the vector space. We begin our study with the definition of a vector space.

Definition 1.1.1 A *vector space* over a field $\mathbb{F}$ is a set V closed under addition and scalar multiplication satisfying the properties: For any $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$ and $c, d \in \mathbb{F}$,

1. $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$
2. $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$
3. There exists a zero vector $\mathbf{0}$ such that $\mathbf{u} + \mathbf{0} = \mathbf{u}$
4. For each $\mathbf{u} \in V$, there exists a vector $-\mathbf{u}$ such that $\mathbf{u} + (-\mathbf{u}) = \mathbf{0}$
5. $c(\mathbf{u} + \mathbf{v}) = c\mathbf{u} + c\mathbf{v}$
6. $(c + d)\mathbf{u} = c\mathbf{u} + d\mathbf{u}$
7. $c(d\mathbf{u}) = (cd)\mathbf{u}$
8. $1\mathbf{u} = \mathbf{u}$

R One often studies vector spaces over more general fields, but for our purposes we work only over $\mathbb{R}$ and $\mathbb{C}$.

The most common vector space we will use is the space $\mathbb{R}^n$ of all n-tuples of real numbers. We

always write the elements of $\mathbb{R}^n$ as column vectors between square brackets:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

but sometimes we may wish to save space and write them as row vectors, in which case we may use the transpose notation (which we wait until Definition 1.2.5 to introduce formally)

$$\mathbf{x} = [x_1, x_2, \dots x_n]^\top.$$

■ **Example 1.1** In $\mathbb{R}^2$ we have

$$[1,2]^\top + [-3,4]^\top = [-2,6]^\top$$

$$10 \cdot [-1,1]^\top = [-10,10]^\top$$

■

1.1.1 Span, Linear Independence, Basis

There is no preferred coordinate system in an abstract vector space. In order to prescribe meaningful coordinate representations of vectors, we need the concept of a basis. For this, we first need to discuss the concepts of linear independence and span.

Definition 1.1.2 Suppose we are given a set of vectors $S = \{\mathbf{v}_1, \dots, \mathbf{v}_n\} \subset V$. Then the *span* of S is the set

$$\text{span}(S) := \{a_1\mathbf{v}_1 + \dots + a_k\mathbf{v}_k : \mathbf{v}_i \in S, a_i \in \mathbb{R}\ \forall i\ \}$$

■ **Example 1.2** The typical well at a cocktail bar contains at least four ingredients at the bartender's disposal; vodka, tequila, orange juice, and grenadine. Assuming we have this well, we can represent drinks as points in $\mathbb{R}^4$, with one element for each ingredient. For instance, a tequila sunrise can be represented using the point

$$[0, 1.5, 6, 0.75]^\top$$

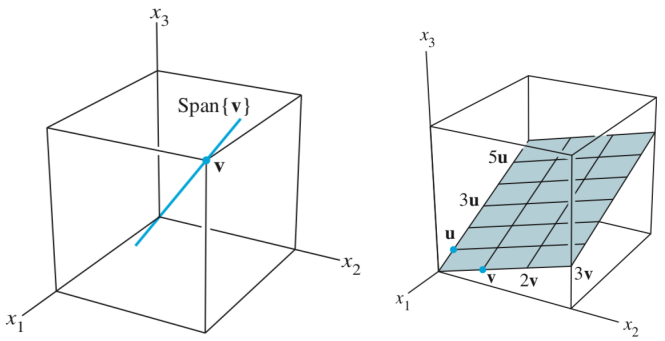
representing amounts of vodka, tequila, orange juice, and grenadine (in ounces). The set of drinks can be represented as a subset of the span

$$\text{span}(\{[1,0,0,0]^\top, [0,1,0,0]^\top, [0,0,1,0]^\top, [0,0,0,1]^\top\})$$

(consisting of vectors with non-negative coefficients). Further, the bartender might be able to save time by making the observation that many drinks have the same orange juice - to - grenadine ratio, and therefore mix the two. So they might simplify their well by mixing the two:

$$\text{span}(\{[1,0,0,0]^\top, [0,1,0,0]^\top, [0,0,6,0.75]^\top\})$$

Notice, it is now easier to pour drinks but this bartender can no longer make as many drinks, such as a screwdriver which contains orange juice but no grenadine. ■



(a) Span of one vector (b) Span of two vectors

Figure 1.1: Linear Spans

Definition 1.1.3 A set $S \subset V$ of vectors is *linearly dependent* if there exists a non-empty linear combination of elements $\mathbf{v}_k \in S$ yielding

$$\sum_{k=1}^{m} c_k \mathbf{v}_k = 0$$

where $c_k \neq 0$ for all k. A set that is not linearly dependent is called *linearly independent*.

Definition 1.1.4 The *dimension* of V is the maximal size $|S|$ of a linearly independent set $S \subset V$ such that $\text{span}(S) = V$. Any linearly independent set S of maximal size $|S|$ with $\text{span}(S) = V$ is called a *basis* of V.

■ **Example 1.3** The standard basis for $\mathbb{R}^n$ is the set of vectors of the form

$$\mathbf{e}_k = [\underbrace{0,\ldots,0}_{k-1 \text{ elements}}, 1, \underbrace{0,\ldots,0}_{n-k \text{ elements}}]^\top$$

for $1 \leq k \leq n$. ■

1.1.2 Vector Norms

One often needs the concept of length or distance when discussing vectors in a vector space. There are many different meaningful ways to define these concepts for a given vector space. To this end, we introduce vector norms.

Definition 1.1.5 A *vector norm* is a function $\|\cdot\| : \mathbb{R}^n \to [0,\infty)$ satisfying the following conditions:

1. $\|\mathbf{x}\| = 0$ if and only if $x = 0$ (Nondegeneracy)
2. $\|c\mathbf{x}\| = |c|\,\|\mathbf{x}\|$ for all scalars $c \in \mathbb{R}, \mathbf{x} \in \mathbb{R}^n$ (Absolutely scalability)
3. $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ for all $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$ (Triangle Inequality)

■ **Example 1.4** For $p \geq 1$, $p \in \mathbb{Z}$ we define the *p-norm* on $\mathbb{R}^n$ by

$$\|\mathbf{x}\|_p := (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{\frac{1}{p}} \tag{1.1}$$

and the *infinity norm*

$$\|\mathbf{x}\|_\infty := \max(|x_1|, |x_2|, \ldots, |x_n|). \tag{1.2}$$

We leave it to the reader to verify that these are indeed vector norms according to Definition 1.1.5. One should observe that the limit as p approaches infinity of the p-norms of a vector is equal to its infinity norm. The most common measure of length is the 2-norm. Given a norm $\|\cdot\|$ one can define the distance between two vectors $\mathbf{x}, \mathbf{y}$ with respect to the norm $\|\cdot\|$ as $\|\mathbf{x}-\mathbf{y}\|$. The most commonly used distance is the 2-norm distance. ■

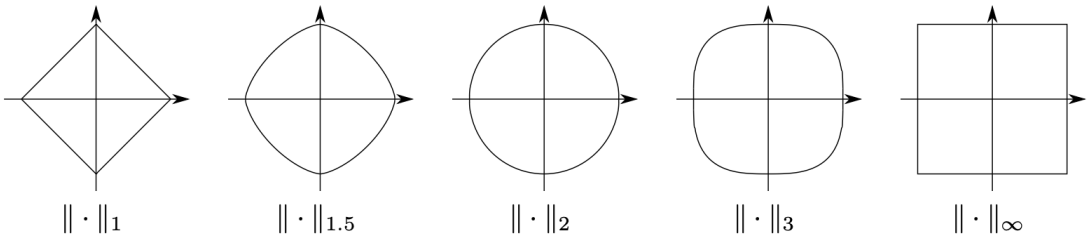


Figure 4.7 The set $\{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\| = 1\}$ for different vector norms $\|\cdot\|$.

Two norms $\|\cdot\|$ and $\|\cdot\|'$ are said to be equivalent if there exists constants c_{low} and c_{high} such that $c_{\text{low}}\|\mathbf{x}\| \leq \|\mathbf{x}\|' \leq c_{\text{high}}\|\mathbf{x}\|$ for all $\mathbf{x} \in \mathbb{R}^n$. The equivalence theorem of norms in finite dimension tells us that all norms on $\mathbb{R}^n$ are equivalent. We leave it to the reader to seek a proof of this surprising fact, if desired.

1.2 Linear Mappings

In mathematics, there is the common notion that to properly study objects (vector spaces in our case), one should study structure preserving maps between the objects. This leads us to the notion of linear maps.

Definition 1.2.1 Suppose V and V' are vector spaces. Then a mapping $T : V \to V'$ is *linear* if it satisfies the following for all $\mathbf{v}_1, \mathbf{v}_2 \in V$ and $c \in \mathbb{C}$:

- $T(\mathbf{v}_1 + \mathbf{v}_2) = T(\mathbf{v}_1) + T(\mathbf{v}_2)$
- $T(c\mathbf{v}) = cT(\mathbf{v})$

■ **Example 1.5** Consider the map $T : \mathbb{R}^2 \to \mathbb{R}^3$ defined by $T([x,y]^\top) = [3x, 2x+y, -y]^\top$. More generally, let $A \in \mathbb{R}^{m\times n}$ be a matrix and define a map $T_A : \mathbb{R}^n \to \mathbb{R}^m$ by $T_A(x) = Ax$. We leave it to the reader to verify these maps are indeed linear according to Definition 1.2.1. ■

Proposition 1.2.1 A linear mapping T on $\mathbb{R}^n$ is completely determined by its action on the standard basis vectors $\mathbf{e}_k$.

Proof. For any $\mathbf{v} \in \mathbb{R}^n$ we have

$$T(\mathbf{v}) = T(\sum_k v_k \mathbf{e}_k) = \sum_k T(v_k \mathbf{e}_k) = \sum_k v_k T(\mathbf{e}_k)$$

■

■ **Example 1.6** Returning the previous example:

$$T([x,y]^\top) = xT(\mathbf{e}_1) + yT(\mathbf{e}_2) = x[3,2,0]^\top + y[0,1,-1]^\top.$$

■

1.2.1 Matrices

Once we choose bases, linear maps between vector spaces can be represented by 2-dimensional arrays called matrices. Linear maps are often easier to study by thinking about the matrices which represent them.

Definition 1.2.2 The *space of matrices* $\mathbb{R}^{m\times n}$ is the set composed of matrices of the form:

$$\mathbf{V} = \begin{bmatrix} V_{11} & V_{12} & \dots & V_{1n} \\ V_{21} & V_{22} & \dots & V_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ V_{m1} & V_{m2} & \dots & V_{mn} \end{bmatrix}$$

with $V_{i,j} \in \mathbb{R}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. For $1 \leq i \leq m$ we let $\mathbf{V}_{i,*} = [V_{i,1}, \dots, V_{i,n}]^\top$ denote the ith row of $\mathbf{V}$ and for $1 \leq j \leq n$ we let $\mathbf{V}_{*,j} = [V_{1,j}, \dots, V_{m,j}]^\top$ denote the jth column of $\mathbf{V}$. We follow the common convention that both are interpreted as column vectors.

Definition 1.2.3 Matrix to vector multiplication is defined by

$$\begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ \mathbf{V}_{*,1} & \mathbf{V}_{*,2} & \dots & \mathbf{V}_{*,n} \\ \vdots & \vdots & \dots & \vdots \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = c_1\mathbf{V}_{*,1} + c_2\mathbf{V}_{*,2} + \ldots + c_n\mathbf{V}_{*,n}.$$

The Riesz representation theorem links the concepts of linear mappins with matrices. Riesz states that any linear map can be expressed as matrix to vector multiplication by a fixed matrix (once a basis is picked for both the domain and codomain).

Theorem 1.2.2 — Riesz Representation. Every linear mapping from a vector space V to V', there exists a unique linear matrix A (up to a chosen bases on the domain and codomain) such that

$$T\left(\begin{bmatrix} x \\ \vdots \\ y \end{bmatrix}\right) = A \begin{bmatrix} x \\ \vdots \\ y \end{bmatrix}$$

■ **Example 1.7** The linear map T from Example 1.5 can be written

$$T\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 3 & 0 \\ 2 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

■

In many ways, this theorem is profound. If you want to define a linear function from $\mathbb{R}^3$ to $\mathbb{R}^{10000000000000}$, you simply have to define exactly 3 points, each corresponding to the evaluation of the function on each of the basis vectors. In the context of machine learning, this theorem provides us a one-to-one correspondence to represent every (linear) function simply as an array.

Definition 1.2.4 Matrix to matrix multiplication is defined by natural extension of matrix to vector multiplication

$$\mathbf{M}\begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_n \\ \vdots & \vdots & \dots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ \mathbf{M}\mathbf{v}_1 & \mathbf{M}\mathbf{v}_2 & \dots & \mathbf{M}\mathbf{v}_n \\ \vdots & \vdots & \dots & \vdots \end{bmatrix}$$

A matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is said to be *invertible* if there exists a matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$ such that $\mathbf{AB} = \mathbf{I} = \mathbf{BA}$ where $\mathbf{I}$ is the identity matrix ($I_{i,j}$ is 1 if $i = j$ and 0 otherwise). In this case one will write $\mathbf{B} = \mathbf{A}^{-1}$.

■ **Example 1.8** Returning to the cocktail example, suppose we make two drinks from our 3 defined wells of liquid (vodka, tequila, and the mix of grenadine and orange juise). Then to find the basic ingredients we simply use matrix multiplication:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 6 \\ 0 & 0 & 0.75 \end{bmatrix}\begin{bmatrix} 0 & 0.75 \\ 1.5 & 0.75 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 0.75 \\ 1.5 & 0.75 \\ 6 & 12 \\ 0.75 & 1.5 \end{bmatrix}$$

■

1.2.2 Some Common Matrix Operations

In this section we introduce the transpose, trace, induced matrix norm, and determinant, along with some useful properties of the common operations.

Definition 1.2.5 The *transpose* of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is the matrix $\mathbf{A}^\top \in \mathbb{R}^{n\times m}$ with elements defined by

$$A^\top_{i,j} = A_{j,i}$$

for all $1 \le i \le m$ and $1 \le j \le n$.

Definition 1.2.6 The *trace* of a matrix $\mathbf{A}$ is $\mathrm{Tr}(\mathbf{A}) = \sum_i \mathbf{A}_{ii}$.

The reader can verify that for all matrices $\mathbf{A}, \mathbf{B}$ we have

1. $(\mathbf{A}^\top)^\top = \mathbf{A}$
2. $(\mathbf{A}+\mathbf{B})^\top = \mathbf{A}^\top + \mathbf{B}^\top$
3. $(\mathbf{AB})^\top = \mathbf{B}^\top\mathbf{A}^\top$
4. $\mathrm{Tr}(\mathbf{A}) = \mathrm{Tr}(\mathbf{A}^\top)$
5. $\mathrm{Tr}(\mathbf{AB}) = \mathrm{Tr}(\mathbf{BA})$

We found norms to be a useful operation on vectors, and we would like to bootstrap that construction to get a similar operation on matrices.

Definition 1.2.7 The *induced matrix norm on* $\mathbb{R}^{m\times n}$ by a vector norm $\|\cdot\|$ is given by

$$\|\mathbf{A}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|.$$

■ **Example 1.9** The reader may wish to prove the following as an exercise:

1. $\|\mathbf{A}\|_1 = \max_{1\le j\le n} \sum_{i=1}^m |A_{i,j}|$
2. $\|\mathbf{A}\|_\infty = \max_{1\le i\le m} \sum_{j=1}^n |A_{i,j}|$

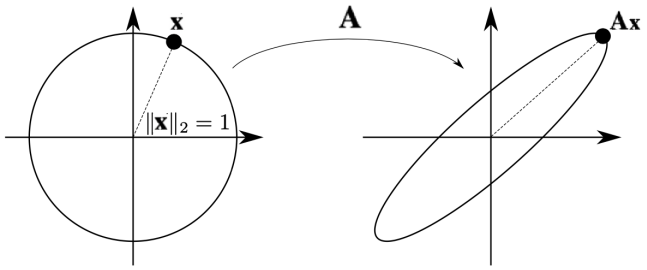


Figure 1.2: This figure illustrates the intuition of the matrix norm of a 2 by 2 matrix $\mathbf{A}$. We simply look at the image of the unit circle and find the distance of the point furthest from the origin.

In general, the p-norm has no such formula, but we will find soon that the SVD gives a nice way to compute the 2-norm. ∎

Another useful norm for matrices is given by treating a matrix as if it were a vector and simply taking the vector norm.

Definition 1.2.8 The *Frobenius norm* is defined by $\|\mathbf{A}\|_{\text{Fro}} := \sqrt{\sum_{i,j} A_{i,j}^2}$.

R The Frobenius norm cannot be induced from a vector norm.

The Frobenius norm has a useful relationship to the trace and transpose operations discussed previously.

Proposition 1.2.3 For any matrix $\mathbf{A}$ we have

$$\|\mathbf{A}\|_{\text{Fro}} = \sqrt{\text{Tr}(\mathbf{A}^\top \mathbf{A})}.$$

Here we introduce the determinant of a matrix. We skip any motivation for the definition and simply write it down along with a few common consequences. The reader can consult any linear algebra textbook for more details.

Definition 1.2.9 Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, the *determinant* of $\mathbf{A}$ is

$$\det(\mathbf{A}) = \sum_{\pi \in \mathfrak{S}_n} (-1)^{\text{inv}(\pi)} \prod_i A_{i,\pi(i)}$$

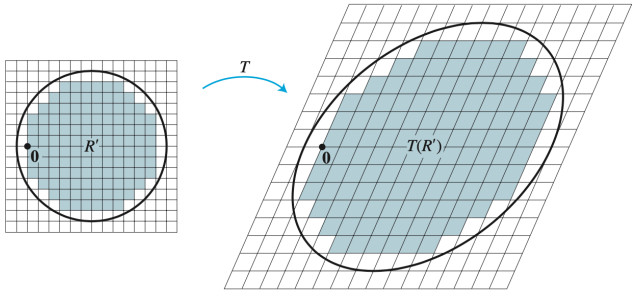
where $\mathfrak{S}_n$ is the set of permutations on $\{1,\dots,n\}$ and $\text{inv}(\pi)$ is the number of inversions of π, that is, the number of pairs $i < j$ such that $\pi(i) > \pi(j)$.

■ **Example 1.10** For $n = 2$ there are two permutations $(1,2)$ and $(2,1)$ on the set $\{1,2\}$ and we have $\text{inv}(1,2) = 0$ and $\text{inv}(2,1) = 1$. We find that

$$\det\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right) = 1\cdot 4 - 3\cdot 2 = -2.$$

∎

R The geometric interpretation of the determinant is that it is the ratio volume of the image of the unit sphere after the action by the matrix A divided by the volume of the unit sphere within the ambient vector space. There is a minus sign involved if the transformation reverses the orientation of the sphere.



We state without proof that for any matrices $\mathbf{A}, \mathbf{B}$ of compatible dimensions, that

$$\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B}). \tag{1.3}$$

We encourage the reader to seek out a proof of this fundamental result, along with the fact that a matrix has nonzero determinant if and only if that matrix is invertible.

1.2.3 Orthogonality and The Inner Product Space $\mathbb{R}^n$

One of the most important constructions we are leading towards is the singular value decomposition (SVD) of a matrix. In order to define the SVD, we need the concept of orthogonality.

Definition 1.2.10 The *dot product* of two vectors $\mathbf{u}$ and $\mathbf{v}$ in $\mathbb{R}^n$ is given by

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^\top \mathbf{v}.$$

■ **Example 1.11** In $\mathbb{R}^2$, we see

$$[1,2]^\top \cdot [-2,6]^\top = 10$$

■

Notice that for $\mathbf{u} \in \mathbb{R}^n$ we have $\|\mathbf{u}\|_2 = \sqrt{u_1^2 + \ldots + u_n^2} = \sqrt{\mathbf{u} \cdot \mathbf{u}}$. Geometrically, we may remember the following relationship:

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|_2 \|\mathbf{v}\|_2 \cos\theta$$

where θ is the angle between $\mathbf{u}$ and $\mathbf{v}$. When $\cos\theta = 0$, we see that the dot product is also zero. This motivates the following definition:

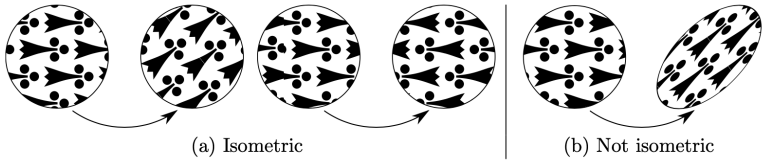
Definition 1.2.11 Two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ are *orthogonal* when $\mathbf{u} \cdot \mathbf{v} = 0$. A set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ is *orthonormal* if $\|\mathbf{v}_i\|_2 = 1$ for all i and $\mathbf{v}_i \cdot \mathbf{v}_k = 0$. A square matrix whose columns are orthonormal is called an *orthogonal matrix*.

Proposition 1.2.4 A matrix $\mathbf{Q}$ is orthogonal if and only if $\mathbf{Q}$ is invertible and $\mathbf{Q}^{-1} = \mathbf{Q}^\top$.

Proof. This follows immediately from the definition of orthogonal matrix. The reader should write out the details as an exercise. ■

Definition 1.2.12 An *isometry* on $\mathbb{R}^n$ is a distance preserving bijection $f : \mathbb{R}^n \to \mathbb{R}^n$. That is,

$$\|f(\mathbf{x}) - f(\mathbf{y})\|_2 = \|\mathbf{x} - \mathbf{y}\|_2 .$$



(a) Isometric (b) Not isometric

We end this section by stating without proof two commonly used facts about orthogonal matrices.

Proposition 1.2.5

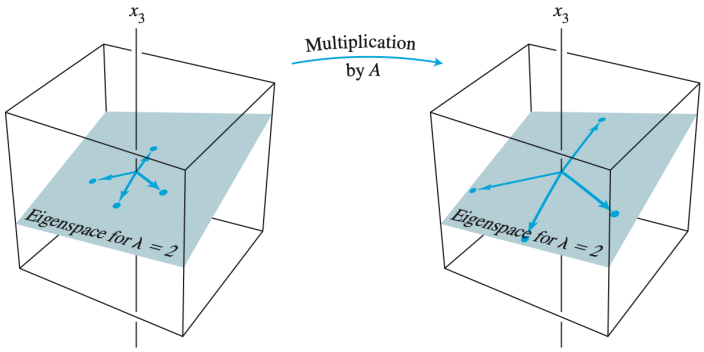
1. If $\mathbf{Q}$ is orthogonal, then the function $\mathbf{x} \mapsto \mathbf{Q}\mathbf{x}$ is an isometry on $\mathbb{R}^n$.
2. Every isometry on $\mathbb{R}^n$ can be written as $\mathbf{x} \mapsto \mathbf{A} + \mathbf{Q}\mathbf{x}$ for some $\mathbf{A}, \mathbf{Q} \in \mathbb{R}^{n\times n}$ with $\mathbf{Q}$ orthogonal.

1.3 Eigenvalue and Singular Value Decompositions

Square matrices are often best understood by writing down a special decomposition called the eigenvector decomposition. To this end we define eigenvectors and eigenvalues. We will find especially nice decompositions for symmetric matrices.

1.3.1 Eigenvalues and Eigenvectors

Definition 1.3.1 An *eigenvector* of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a nonzero vector v such that $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ for some scalar λ. A scalar λ is called an *eigenvalue* of $\mathbf{A}$ if there is a nonzero vector $\mathbf{v}$ such that $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$.



Proposition 1.3.1 Every matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ has at least one (potentially complex) eigenvector.

Proof. Take any vector $x \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ and assume $\mathbf{A} \neq \mathbf{0}$ since this matrix trivially has eigenvalue 0. The set $\{\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}, \ldots, \mathbf{A}^n\mathbf{x}\}$ must be linearly dependent because it contains $n+1$ vectors in n dimensions. So there exists constants $c_0, \ldots, c_n \in \mathbb{R}$ not all zero such that $\mathbf{0} = c_0\mathbf{x} + c_1\mathbf{A}\mathbf{x} + c_2\mathbf{A}^2\mathbf{x} + \ldots + c_n\mathbf{A}^n\mathbf{x}$. We define the polynomial $f(z) = c_0 + c_1 z + \ldots + c_n z^n$ By the fundamental theorem of Algebra, there exist $m \geq 1$ roots $z_i \in \mathbb{C}$ and $c \neq 0$ such that $f(z) = c(z - z_1)(z - z_2)\ldots(z - z_m)$. Applying this factorization, we see that $\mathbf{0} = c_0\mathbf{x} + c_1\mathbf{A}\mathbf{x} + \ldots + c_n\mathbf{A}^n\mathbf{x} = c(\mathbf{A} - z_1\mathbf{I})\ldots(\mathbf{A} - z_m\mathbf{I})\mathbf{x}$. In this form, at least one $\mathbf{A} - z_i\mathbf{I}$ has a null space, since otherwise each term would be invertible, forcing $\mathbf{x} = \mathbf{0}$, which we already assumed against. So if we take $\mathbf{v}$ to be a nonzero vector in the null space of $\mathbf{A} - z_i\mathbf{I}$, then by construction $\mathbf{A}\mathbf{v} = z_i\mathbf{v}$. ■

We recall here some basic facts one should be aware of, which hopefully the reader has seen in a previous course in linear algebra. Suppose V is a complex vector space and $T : V \to V$ is a linear map. Let $\lambda_1, \ldots, \lambda_m$ denote the distinct eigenvalues of T, with multiplicities $d_1, \ldots, d_m$. Then the polynomial

$$(z - \lambda_1)^{d_1} \ldots, (z - \lambda_m)^{d_m}$$

is called the *characteristic polynomial* of T. A scalar λ is an eigenvalue of and $n \times n$ matrix A if and only if λ satisfies the characteristic equation $\det(A - \lambda I) = 0$. The eigenvectors corresponding to distinct eigenvalues of a given matrix are linearly independent. To understand why this is true, suppose otherwise. Then there exists eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ with distinct eigenvalues $\lambda_1, \ldots, \lambda_k$ that are linearly dependent. This implies that there are coefficients $c_1, \ldots, c_k$ not all zero such that $\mathbf{0} = c_1\mathbf{v}_1 + \ldots + c_k\mathbf{v}_k$. Notice, for any $i \neq j$, we see that $(\mathbf{A} - \lambda_i\mathbf{I})x_j = \mathbf{A}\mathbf{x}_j - \lambda_i\mathbf{x}_j = (\lambda_j - \lambda_i)\mathbf{x}_j$ We can then isolate one of the coefficients $\mathbf{0} = (\mathbf{A} - \lambda_2\mathbf{I}) \ldots (\mathbf{A} - \lambda_k\mathbf{I})c_1\mathbf{v}_1 + \ldots + c_k\mathbf{v}_k = c_1(\lambda_1 - \lambda_2) \ldots (\lambda_1 - \lambda_k)$ Since λ_1 does not equal any of the other distinct eigenvalues, then $c_1 = 0$ We can repeat this for all the either eigenvalues.

1.3.2 Symmetric and Positive Semi-definite Matrices

Here we recall some facts and definitions about symmetric and positive (semi)-definite matrices. Recall a square matrix $\mathbf{A}$ is *symmetric* if $\mathbf{A} = \mathbf{A}^\top$. A symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is *positive semi-definite* if for every $\mathbf{x} \in \mathbb{R}^n$ it follows that $\mathbf{x}^\top\mathbf{A}\mathbf{x} \geq 0$, and is called *positive definite* if furthermore the only vector $\mathbf{x}$ with $\mathbf{x}^\top\mathbf{A}\mathbf{x} = 0$ is the zero vector $\mathbf{x} = \mathbf{0}$.

Theorem 1.3.2

1. All eigenvalues of symmetric matrices are real.
2. Eigenvectors corresponding to distinct eigenvalues of symmetric matrices are orthogonal.
3. If A is positive-semidefinite, then A has nonnegative real eigenvalues.
4. If A is positive-definite, then A has positive eigenvalues.
5. For any $\mathbf{A} \in \mathbb{R}^{m\times n}$, the matrix $\mathbf{A}^\top\mathbf{A}$ is positive semidefinite.
6. For any $\mathbf{A} \in \mathbb{R}^{m\times n}$, the matrix $\mathbf{A}^\top\mathbf{A}$ is positive definite provided the columns of $\mathbf{A}$ are linearly independent.

1.3.3 Diagonalization and Spectral Theorem

A matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is said to be *diagonalizable* if $\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^{-1}$ for some invertible matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$ and some diagonal matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$.

Theorem 1.3.3 An $n \times n$ matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is diagonalizable if and only if $\mathbf{A}$ has n linearly independent eigenvectors. In this case, $\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^{-1}$ where the columns of $\mathbf{P}$ are exactly the eigenvectors of $\mathbf{A}$ and the diagonal entries of $\mathbf{D}$ are the eigenvalues of $\mathbf{A}$.

Theorem 1.3.4 — Spectral Theorem. Suppose $\mathbf{A} \in \mathbb{C}^{n\times n}$ is symmetric. Then $\mathbf{A}$ has exactly n orthonormal real eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ with (possibly repeated) eigenvalues $\lambda_1, \ldots, \lambda_n$. In other words, there exists an orthogonal matrix $\mathbf{Q}$ of eigenvectors and a diagonal matrix $\mathbf{D}$ such that $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$.

Proofs of these very important theorems can be found in any linear algebra textbook.

1.3.4 Singular Value Decomposition

For matrices which are not square, unfortunately the theory of diagonalization cannot be applied. The generalization to the setting of non-square matrices is called the singular value decomposition.

Theorem 1.3.5 Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, then there exist orthogonal $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ such that

$$\mathbf{U}^\top \mathbf{A}\mathbf{V} = \Sigma := \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$$

where $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \ldots \geq \sigma_p \geq 0$.

Proof. Since $\mathbf{A}^\top\mathbf{A}$ is a symmetric matrix, we can apply the Spectral Theorem 1.3.4 to get $\mathbf{A}^\top\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^\top$ where $\mathbf{V}$ is an orthogonal matrix. Further, since $\mathbf{A}^\top\mathbf{A}$ is positive semidefinite, we know all of the entries of $\mathbf{D}$ are nonnegative. We now define the set $\{\mathbf{v}_i\}_{i=1}^n$ to be the set of eigenvectors that make up the columns of $\mathbf{V}$. Define

$$\mathbf{u}_i = \frac{\mathbf{A}\mathbf{v}_i}{\sqrt{\mathbf{D}_{ii}}}$$

for all i up until $\mathbf{D}_{ii}$ potentially becomes 0. Observe,

$$\|\mathbf{u}_i\| = \left|\frac{1}{\sqrt{D_{ii}}}\right| \|\mathbf{A}\mathbf{v}_i\| = \left|\frac{1}{\sqrt{D_{ii}}}\right| \sqrt{\mathbf{v}_i^\top \mathbf{A}^\top \mathbf{A}\mathbf{v}_i} = \left|\frac{1}{\sqrt{D_{ii}}}\right| \sqrt{D_{ii}\mathbf{v}_i^\top \mathbf{v}_i} = 1$$

$$\mathbf{u}_i^\top \mathbf{u}_j = \frac{\mathbf{v}_i^\top \mathbf{A}^\top \mathbf{A}\mathbf{v}_j}{\sqrt{D_{ii}D_{jj}}} = \frac{\mathbf{v}_i^\top D_{jj}\mathbf{v}_j}{\sqrt{D_{ii}D_{jj}}} = \frac{D_{jj}}{\sqrt{D_{ii}D_{jj}}}\mathbf{v}_i^\top \mathbf{v}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

So we see that $\{\mathbf{u}_i\}_i$ is a set of orthonormal vectors. Use the Gram-Schmidt algorithm to extend this to an orthonormal basis of $\mathbb{R}^m$, and define the matrix $\mathbf{U}$ to have this basis as its set of column vectors where the vectors are ordered in decreasing order of the corresponding eigenvalues. Lastly, define $\sigma_i = \sqrt{D_{ii}}$. Then we see that

$$\mathbf{U}\Sigma = \begin{bmatrix} \vdots & \vdots & \ldots & \vdots \\ \mathbf{u}_1 & \mathbf{u}_2 & \ldots & \mathbf{u}_m \\ \vdots & \vdots & \ldots & \vdots \end{bmatrix} \operatorname{diag}(\sigma_1, \ldots, \sigma_n) = \begin{bmatrix} \vdots & \vdots & \ldots & \vdots \\ \mathbf{A}\mathbf{v}_1 & \mathbf{A}\mathbf{v}_2 & \ldots & \mathbf{A}\mathbf{v}_n \\ \vdots & \vdots & \ldots & \vdots \end{bmatrix} = \mathbf{A}\mathbf{V}.$$

■

We continue to use the notation in the above proof, where the $\mathbf{u}_i$ denote the columns of $\mathbf{U}$ and $\mathbf{v}_j$ denote the columns of $\mathbf{V}$. Another often more convenient way to interpret the SVD is as follows. If $\mathbf{U}^\top\mathbf{A}\mathbf{V} = \Sigma$ for $\mathbf{A} \in \mathbb{R}^{m\times n}$, then for $1 \leq i \leq n, \mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$ and $\mathbf{A}^\top\mathbf{u}_i = \sigma_i\mathbf{v}_i$.

Proposition 1.3.6 If $\mathbf{A} \in \mathbb{R}^{m\times n}$, then $\|\mathbf{A}\|_2 = \sigma_1$ and $\|\mathbf{A}\|_{\text{Fro}} = \sqrt{\sigma_1^2 + \ldots + \sigma_p^2}$

Proof. We present a proof of the second fact, leaving the first to the reader to confirm

$$\|\mathbf{A}\|_{\text{Fro}} = \sqrt{\sum_{i,j} A_{i,j}^2} = \sqrt{\operatorname{Tr}(\mathbf{A}^\top\mathbf{A})} = \sqrt{\operatorname{Tr}(\mathbf{V}\Sigma\mathbf{U}^\top\mathbf{U}\Sigma\mathbf{V}^\top)} = \sqrt{\operatorname{Tr}(\mathbf{V}\Sigma^2\mathbf{V}^\top)}$$

$$= \sqrt{\operatorname{Tr}(\mathbf{V}^\top\mathbf{V}\Sigma^2)} = \sqrt{\operatorname{Tr}(\Sigma^2)} = \sqrt{\sigma_1^2 + \ldots + \sigma_p^2}$$

■

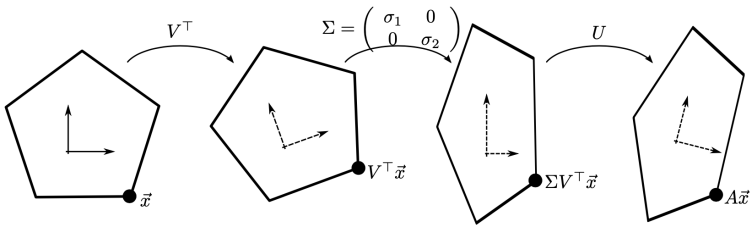


Figure 1.3: Visually, for 2 by 2 matrices, one can see the SVD as a decomposition of the action of a matrix $\mathbf{A}$ on vectors in $\mathbb{R}^2$ as being composed by a rotation, a stretching factor, and another rotation (also possibly reversing the orientation).

It may be interesting to note that since orthogonal matrices preserve 2-norm distance, we have

$$\|\Sigma\|_2 = \left\|\mathbf{U}\mathbf{A}\mathbf{V}^\top\right\|_2 = \|\mathbf{A}\|_2 .$$

Recall the *kernel* of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is the set $\ker(\mathbf{A}) = \{\mathbf{v} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{v} = \mathbf{0}\}$ and the *image* of $\mathbf{A}$ is the set $\mathrm{im}(\mathbf{A}) = \{\mathbf{A}\mathbf{v} \mid \mathbf{v} \in \mathbb{R}^n\} \subseteq \mathbb{R}^m$. Both the kernel and the image of $\mathbf{A}$ are subspaces. The kernels and images of $\mathbf{A}$ and $\mathbf{A}^\top$ are often called the four fundamental subspaces of $\mathbf{A}$. The reader should verify the following important result.

Theorem 1.3.7 — Four Fundamental Subspaces. If $\mathbf{A}$ has r positive singular values where $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$ is an SVD for $\mathbf{A}$, then $\mathrm{rank}(\mathbf{A}) = r$ and

$$\ker(\mathbf{A}) = \mathrm{span}\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n\}$$

$$\mathrm{im}(\mathbf{A}) = \mathrm{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$$

$$\ker(\mathbf{A}^\top) = \mathrm{span}\{\mathbf{u}_{r+1}, \ldots, \mathbf{u}_m\}$$

$$\mathrm{im}(\mathbf{A}^\top) = \mathrm{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$$

Notice that the theorem indicates a fundamental conservation property about these fundamental subspaces. Specifically,

Corollary 1.3.8 — Rank-Nullity Theorem. If $A \in \mathbb{R}^{m\times n}$, then

$$dim(\ker(\mathbf{A})) + dim(\mathrm{im}(\mathbf{A})) = n$$

$$dim(\ker(\mathbf{A}^\top)) + dim(\mathrm{im}(\mathbf{A}^\top)) = m$$

Proposition 1.3.9 If $\mathbf{A} \in \mathbb{R}^{m\times n}$, with $\mathrm{rank}(\mathbf{A}) = r$, then $\mathbf{A} = \sum_{k=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$.

Proof.

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sigma_1\mathbf{u}_i & \ldots & \sigma_r\mathbf{u}_r & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \ldots & \mathbf{v}_1^\top & \ldots \\ & \vdots & \\ \ldots & \mathbf{v}_n^\top & \ldots \end{bmatrix} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$$

■

Adjusting the upper index of summation in the above formula turns out to have an interesting interpretation. The very useful matrix approximation theorem states that the matrix $\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ is the closest approximation to the matrix $\mathbf{A}$ among all matrices of rank k. This holds true for both the 2-norm and the Frobenius norm:

$$\mathbf{A}_k = \underset{\text{rank}(\mathbf{M})=k}{\text{argmin}} \; ||\mathbf{A}-\mathbf{M}||_{\text{Fro}}$$

$$\mathbf{A}_k = \underset{\text{rank}(\mathbf{M})=k}{\text{argmin}} \; ||\mathbf{A}-\mathbf{M}||_{2}.$$

1.4 Exercises

1. Consider the vector space $\mathscr{P}_3(x) = \{a+bx+cx^2 \mid a,b,c,d \in \mathbb{R}\}$ consisting of polynomials $f(x)$ of maximum degree less than 3. Consider the set

$$\mathscr{B} = \left\{1,\ -x+1,\ \frac{1}{2}(x^2-4x+2)\right\}$$

consisting of the first three Laguerre polynomials.
   (a) Show that $\mathscr{B}$ is a basis for $\mathscr{P}_3(x)$.
   (b) Consider the map $E : \mathscr{P}_3(x) \to \mathbb{R}^4$ by

$$E(a+bx+cx^2) = [a, a-b, b-c, c]^\top$$

   for a polynomial $a+bx+cx^2 \in \mathscr{P}_3(x)$. Show that the map E is linear.
   (c) Compute the matrix of E with respect to the basis $\mathscr{B}$ for $\mathscr{P}_3(x)$ and the standard basis $\mathscr{B}_{\text{std}}$ for $\mathbb{R}^4$.
   (d) Determine the rank of E. Is E invertible?
2. **Examples of Positive Semi-Definite Matrices**
   (a) Let $z \in \mathbb{R}^n$ be an n-vector. Show that $A = zz^T$ is positive semidefinite.
   (b) Let $z \in \mathbb{R}^n$ be a non-zero n-vector. Let A = zz^T. What is the null-space of A? What is the rank of A?
   (c) Let $A \in \mathbb{R}^{n\times n}$ be positive semidefinite and $B \in \mathbb{R}^{m\times n}$ be arbitrary, where $m,n \in \mathbb{N}$ Is BAB^T Positive Semi-Definite? If so, prove it. If not, give a counterexample with explicit A,B.
3. Let $\mathbf{U}$ be an upper triangular matrix with with positive entries on its diagonal and define $\mathbf{A} = \mathbf{U}^\top\mathbf{U}$. Use the matrix $\mathbf{U}$ to define matrices $\mathbf{L}$ and $\mathbf{D}$ such that $\mathbf{L}$ is lower unitriangular (lower triangular with 1s along the diagonal), $\mathbf{D}$ is diagonal with all positive entries, and $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^\top$.
4. Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ where $m \geq n$, with SVD

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top.$$

   Express the column vector $\mathbf{A}_{*,k}$ as a linear combination of the columns of $\mathbf{U}$. That is, show that

$$\mathbf{A}_{*,k} = \sum_{i=1}^{n} \beta_{i,k}\mathbf{U}_{*,i}$$

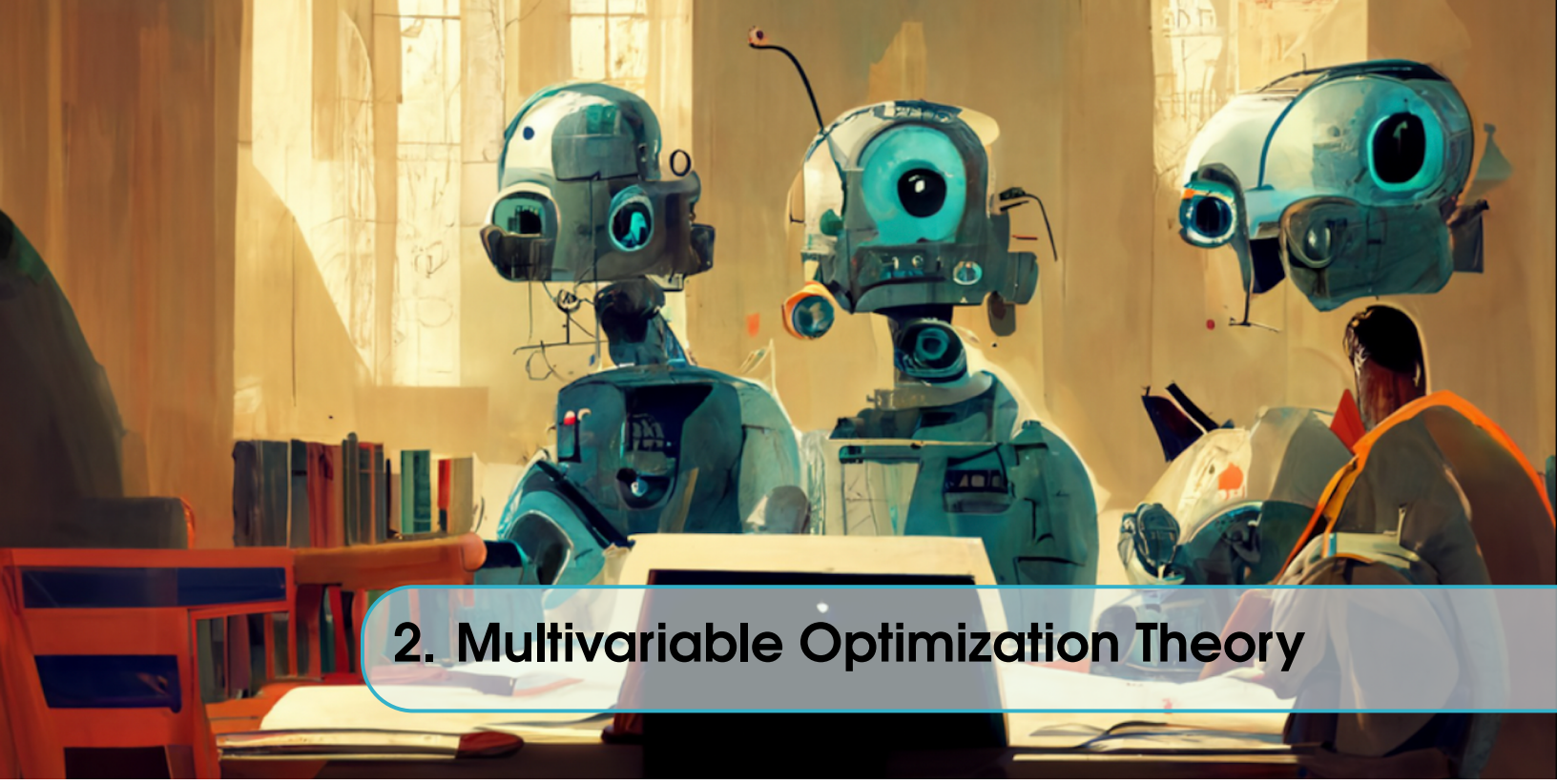
   and determine the coefficients $\beta_{i,k}$.

1.5 Solutions to Exercises

1. (a) To show that $\mathscr{B}$ is a basis, suppose that $a+b(1-x)+c(\frac{1}{2}(x^2-4x+2)=0$ for some $a,b,c\in\mathbb{R}$. Then we have $(a+b+c)+(-b-2c)x+\frac{c}{2}x^2=0$ and we find that $a=b=c=0$. This shows that $\mathscr{B}$ is linearly independent. Now we can simply observe that since $\{1,x,x^2\}$ is a basis and $\mathscr{B}$ is a linearly independent set with the same number of elements as $\{1,x,x^2\}$, $\mathscr{B}$ must also be spanning and therefore a basis. (b) To show that E is linear, take any polynomials $f,g\in\mathscr{P}_3(x)$ and $a,b\in\mathbb{R}$. Write $f=f_0+f_1x+f_2x^2$ and $g=g_0+g_1x+g_2x^2$ where $f_i,g_j\in\mathbb{R}$ for all i,j. Then we observe that $E(af+bg)=E(a(f_0+f_1x+f_2x^2)+b(g_0+g_1x+g_2x^2))=E(af_0+bg_0+(af_1+bg_1)x+(af_2+bg_2)x^2)=[af_0+bg_0,a(f_0-f_1)+b(g_0-g_1),a(f_1-f_2)+b(g_1-g_2),af_2+bg_2]^\top=aE(f)+bE(g)$. (c) The matrix of E with respect to $\mathscr{B}$ and the standard basis for $\mathbb{R}^4$ is

$$\begin{bmatrix}E(1) & E(1-x) & E(\frac{1}{2}(x^2-4x+2)\end{bmatrix}=\begin{bmatrix}1 & 1 & 1\\ 0 & 2 & 3\\ 0 & -1 & -\frac{9}{2}\\ 0 & 0 & \frac{1}{2}\end{bmatrix}.$$

   (d) The columns of the matrix of E are linearly independent and therefore the rank of E is equal to the number of columns, so $\text{rank}(E)=3$. Since the matrix of E is not square, E is not invertible.
2. **Fill In**
3. Let $\mathbf{D}=\text{diag}(\mathbf{U})^2$. Then $\mathbf{D}^{-1/2}\mathbf{U}$ is upper unitriangular. Define $\mathbf{L}=(\mathbf{D}^{-1/2}\mathbf{U})^\top$. Then $\mathbf{LDL}^\top=(\mathbf{D}^{-1/2}\mathbf{U})^\top\mathbf{D}(\mathbf{D}^{-1/2}\mathbf{U})=\mathbf{U}^\top(\mathbf{D}^{-1/2})\mathbf{D}\mathbf{D}^{-1/2}\mathbf{U}=\mathbf{U}^\top\mathbf{U}=\mathbf{A}$.
4. We have $\mathbf{A}=\sum_{i=1}^r\sigma_i\mathbf{u}_i\mathbf{v}_i^\top$ so $\mathbf{A}_{*,k}=\sum_{i=1}^n\sigma_iV_{i,k}\mathbf{u}_i=\sum_{i=1}^n\sigma_iV_{i,k}\mathbf{U}_{*,i}$ so we have $\beta_{i,k}=\sigma_iV_{i,k}$ for $1\leq i,j\leq n$.



2. Multivariable Optimization Theory

2.1 Matrix Differentiation

Theorem 2.1.1 Given a constant matrix $\mathbf{A}$:

$$\frac{\partial}{\partial \mathbf{y}}(\mathbf{y}^\top \mathbf{A}\mathbf{y}) = \mathbf{y}^\top(\mathbf{A}+\mathbf{A}^\top)$$

Theorem 2.1.2 For functions f and g and matrix $\mathbf{A}$ with compatible dimensions:

$$\frac{\partial}{\partial \mathbf{y}} f(\mathbf{y})^\top \mathbf{A} g(\mathbf{y}) = g(\mathbf{y})^\top \mathbf{A}^\top \frac{\partial}{\partial \mathbf{y}} f(\mathbf{y}) + f(\mathbf{y})^\top \mathbf{A} \frac{\partial}{\partial \mathbf{y}} g(\mathbf{y})$$

2.2 Optimization Theory

2.2.1 Optimization with No Constraints

2.2.2 Constrained Optimization

In the previous sections, we discussed optimization problems looking for solutions in $\mathbb{R}$ to attain the maximum or minimum value of a function. Sometimes, we want to optimize f over some restriction of its domain. For example, we may want to satisfy a financial budget for a project. These problems are called *constrained optimization* problems.

Definition 2.2.1 A *constrained optimization* problem is an optimization problem for which we want to find the maximal or minimal value of a function $f(\mathbf{x})$ over a subset of all possible values of $\mathbf{x}$. An optimization problem that is not constrained is called an *unconstrained optimization* problem.

Consider $f : \mathbb{R}^n \to \mathbb{R}$ and a set $S \subseteq \mathbb{R}^n$. We want to find:

$$x^* = \underset{\mathbf{x}\in S}{\operatorname{argmin}} f(\mathbf{x})$$

or

$$f_{min} = \min_{\mathbf{x} \in S} f(\mathbf{x})$$

One approach is to modify the gradient descent algorithm. At each step, we check if $f(\mathbf{x})$ is in S. If it is not, then we project $\mathbf{x}$ back into S before taking the next step. Unsurprisingly, this is called *projected gradient descent*. One way to do the projection is to find the value $\mathbf{x} \in S$ such that:

$$\operatorname*{argmin}_{\mathbf{x} \in S} \|\mathbf{x} - \mathbf{x}_{i-1}\|$$

where $\mathbf{x}_{i-1}$ is the result of the previous gradient descent step and is not in S.

Note that the projection step is another optimization problem. When our constraint is simple, projected gradient descent is a useful method. However, when our constraint set S is complicated, determining the projection step may add significant overhead to our algorithm. Examples for which projected gradient fails will be explored in exercise JANICE (In solution, talk about convexity. If Q is a convex set, the optimization problem has a unique solution. I If Q is nonconvex, the solution to PQ(x0) may not be unique: it gives more than one solution.)!!!. In these cases, we'd hope to use a different constrained optimization method.

One such approach is to construct an unconstrained optimization problem whose solutions are able to be transformed into solutions of the original, constrained problem. This can be done in many ways. We will discuss the *Karush-Kuhn-Tucker* (KKT) method, which constructs a new problem with identical solutions to the original.

In order to apply KKT, S must be describable in the following way:

$$S = \{\mathbf{x} | g_i(\mathbf{x}) = 0 \text{ and } h_j(\mathbf{x}) \leq 0 \text{ for } 1 \leq i \leq n,\ 1 \leq j \leq k\}$$

Then, we may construct the following function:

Definition 2.2.2 Define the *Lagrangian* to be the function $L(\mathbf{x}; \lambda, \alpha)$ such that

$$L(\mathbf{x}; \lambda, \alpha) = f(\mathbf{x}) + \sum_i \lambda_i g_i(\mathbf{x}) + \sum_i \alpha_i h_i(\mathbf{x})$$

The parameters α and λ are called *KKT multipliers*. The equalities involving g_i are called the *equality constraints*, and the inequalities involving h_i are called the *inequality constraints*.

R In the case where S can be described exclusively by the equality constraints, the Lagrangian reduces to

$$L(\mathbf{x}; \lambda) = f(\mathbf{x}) + \sum_i \lambda_i g_i(\mathbf{x})$$

In this case, λ is called the *Lagrangian multiplier*.

Proposition 2.2.1 Let S, $L(\mathbf{x}; \lambda, \alpha)$ be defined as described in the KKT method. Then,

$$\min_{\mathbf{x} \in \mathbb{R}^n} \max_{\lambda} \max_{\alpha : \alpha \geq 0} L(\mathbf{x}; \lambda, \alpha)$$

has the same solutions as $\min_{\mathbf{x} \in S} f(\mathbf{x})$.

The justification is as follows:

If $\mathbf{x} \notin S$, then $g_i \neq 0$. The parameter λ is unbounded by definition, resulting in

$$\max_{\lambda} \max_{\alpha:\alpha\geq 0} L(\mathbf{x};\lambda,\alpha) = \infty$$

So, any $\mathbf{x} \notin S$ cannot be an optimal solution.

If $\mathbf{x} \in S$, then $\sum_i \lambda_i g_i(\mathbf{x}) = 0$. Note that since $\alpha \geq 0$ by definition, we also have $\max_{\alpha:\alpha\geq 0} \alpha_i h_i(\mathbf{x}) = 0$. So,

$$\max_{\lambda} \max_{\alpha:\alpha\geq 0} L(\mathbf{x};\lambda,\alpha) = f(\mathbf{x})$$

Thus when $\mathbf{x} \notin S$, $\mathbf{x}$ cannot be an optimal solution for $\min_{\mathbf{x}\in S} f(\mathbf{x})$, and when $\mathbf{x} \in S$, the optimal solutions are equivalent.

R We can construct the Lagrangian to adjust the maximums and minimums involved:

$$\min_{\mathbf{x}} \max_{\lambda} \max_{\alpha:\alpha\geq 0} -f(\mathbf{x}) + \sum_i \lambda_i g_i(\mathbf{x}) + \sum_i \alpha_i h_i(\mathbf{x})$$

$$\max_{\mathbf{x}} \min_{\lambda} \min_{\alpha:\alpha\geq 0} f(\mathbf{x}) + \sum_i \lambda_i g_i(\mathbf{x}) - \sum_i \alpha_i h_i(\mathbf{x})$$

Now, we have reduced our constrained optimization problem to an unconstrained optimization problem. However, not all Lagrangian optimization problems can be easily solved. Fortunately, the KKT method also provides us with a set of conditions that describe the optimal solutions.

Theorem 2.2.2 — Karush-Kahn-Tucker Conditions. Let $f : \mathbb{R}^n \to \mathbb{R}$ and $S \in \mathbb{R}^n$. Let $L(\mathbf{x};\lambda,\alpha)$ be a Lagrangian function for f, S. If $\mathbf{x}$ is a solution for $\operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$, then the following are true:

- $\nabla x L(\mathbf{x};\lambda,\alpha) = 0$
  - In the case where f is not differentiable, the gradient is defined as a set of possible slopes, and this conditions states that 0 is an element of the gradient.
- $\mathbf{x} \in S$ and the constraints on the KKT multipliers are satisfied.
- $\sum_i \alpha_i h_i(\mathbf{x}) = 0$ (this is also known as the *complementary slackness* condition).

These conditions sometimes allow us to determine the solution analytically. If we are unable to analytically determine the solution, these conditions still give us useful information for computing the solution numerically.

2.3 Convex Optimization

2.3.1 Convex Functions

Definition 2.3.1 A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if for all $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$ and all $0 \leq t \leq 1$ we have:

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y})$$

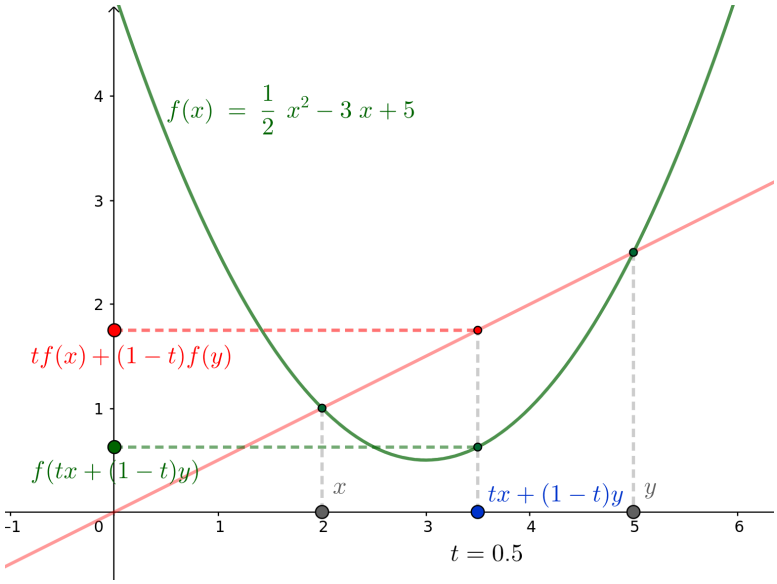
and f is strictly convex if

$$f(t\mathbf{x} + (1-t)\mathbf{y}) < tf(\mathbf{x}) + (1-t)f(\mathbf{y})$$

Intuitively, f is *convex* if the line segment between any two points on the graph function is above

the graph between the two points.

■ **Example 2.1** Example of the convexity of the function $f(x) = \frac{1}{2}x^2 - 3x + 5$



Here we clearly see that :

- $f(tx+(1-t)y) \leq tf(x)+(1-t)f(y)$
- The line segment between f(x) and f(y) is above the graph between these two points

To prove that f is convex, this must be true for all x and y. ■

This definition of convexity applies to functions that are not necessarily differentiable, which is sometimes the case with the functions used in deep learning. For example, an often used activation function is ReLu (Rectified Linear Unit), which is not differentiable.

■ **Example 2.2** Let's show that $f(x) = |x|$, is convex. To do so, we can use the above definition of convexity. Let $x, y \in \mathbb{R}$ and $t \in [0,1]$:

$$\begin{aligned} f(tx+(1-t)y) &= |tx+(1-t)y| \\ &\leq |tx| + |(1-t)y| \qquad \text{using the triangle inequality} \\ &= t|x| + (1-t)|y| \qquad \text{as } t \in [0,1] \\ &= tf(x) + (1-t)f(y) \end{aligned}$$

We have $f(tx+(1-t)y) \leq tf(x)+(1-t)f(y)$ so f is convex. ■

Proposition 2.3.1 A differentiable function $f : \mathbb{R} \to \mathbb{R}$ is convex if and only if its graph lies above all of its tangents:

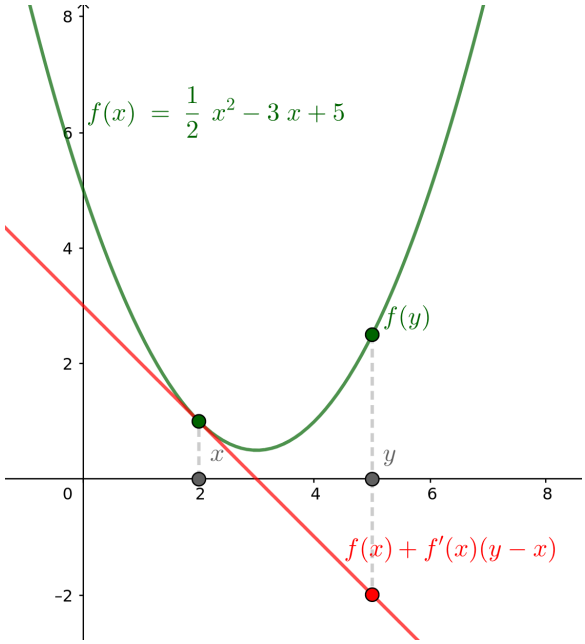
$$f(y) \geq f(x) + f'(x)(y-x), \forall x, y \in \mathbb{R}$$

This property can be generalized to functions of several variables. For functions $f : \mathbb{R}^n \to \mathbb{R}$, this becomes

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla_{\mathbf{x}} f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x})$$

R This property means that the first Taylor expansion at any point of a convex function, is a global under-estimator of the function.

■ **Example 2.3** Example with the same function $f(x) = \frac{1}{2}x^2 - 3x + 5$



Here we clearly see that :

- $f(y) \geq f(x) + f'(x)(y-x)$
- The graph lies above the tangent on x.

Again, to prove that f is convex, this must be true for all x and y. ■

Theorem 2.3.2 A twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if $H(f)(\mathbf{x})$ is positive semi-definite for all $\mathbf{x} \in \mathbb{R}^n$. If $H(f)(\mathbf{x})$ is positive definite, then f is strictly convex.

R One way to see this theorem, is that the first derivative of a convex function is always increasing, so the second derivative is always positive.

2.3.2 Consequences of Convexity

Definition 2.3.2 A function $f : \mathbb{R}^n \to \mathbb{R}$ is $\mathscr{L}$-Lipschitz continuous if $\exists \mathscr{L} > 0$, such that $\forall \vec{x}, \vec{y} \in \mathbb{R}^n, ||f(\vec{x}) - f(\vec{y})|| \geq \mathscr{L}||\vec{x} - \vec{y}||_2$

■ **Example 2.4** Suppose constant $A \in \mathbb{R}^n$ that is greater than 0.

$$f(\vec{x}) = A\vec{x} + \vec{b} \tag{2.1}$$

$$||f(\vec{x}) - f(\vec{y})|| = ||A\vec{x} + \vec{b} - A\vec{y} - \vec{b}|| \tag{2.2}$$

$$= ||A(\vec{x} - \vec{y}) + \vec{b} - \vec{b}|| \tag{2.3}$$

$$= A||(\vec{x} - \vec{y})|| \tag{2.4}$$

Since A is a constant, there must exist an $\mathscr{L}$ that is less than or equal to A, implying

$$A||(\vec{x}-\vec{y})|| \geq A||(\vec{x}-\vec{y})|| \tag{2.5}$$

. Thus $f(\vec{x})$ must be $\mathscr{L}$-Lipschitz continuous. ∎

Definition 2.3.3 $f : \mathbb{R}^n \to \mathbb{R}$ is convex if $H(f)(\vec{x}))$ is a positive semidefinite for all $\vec{x} \in \mathbb{R}$

∎ **Example 2.5** Consider $f(\vec{x}) = b^T\vec{x} + \vec{x}^T A\vec{x} + c$

$$\begin{aligned}\frac{df}{d\vec{x}}(\vec{x}) &= b^T + \vec{x}(A + A^T) \\ &= b^T + 2\vec{x}^T A \\ &= \frac{df}{d\vec{x}^2}(\vec{x}) = 2A^T = 2A\end{aligned}$$

so f is convex iff A is positive semi-definite ∎

∎ **Example 2.6** $f(\vec{x}) = x^T\vec{x} = ||\vec{x}||^2{}_2$ is convex $\vec{x}$ ∎

2.4 Exercises

1. Let $f(x) = \frac{1}{2}x^T Ax + b^T x$ where A is a symmetric matrix and $b \in \mathbb{R}^n$ is a vector. What is $\nabla_x f(x)$? What is $\nabla_x^2 f(X)$?
2. Let $f(x) = g(h(x))$ where $g : \mathbb{R} \to R$ is differentiable and $h : \mathbb{R}^n \to \mathbb{R}$ is differentiable. What is $\nabla_x f(x)$?
3. Let $f(x) = g(a^T x)$ where $g : \mathbb{R} \to \mathbb{R}$ is continuously differentiable and $a \in \mathbb{R}^n$ is a vector. What is $\nabla_x f(x)$ and $\nabla^2 f(x)$?
4. Consider the average empirical loss (the risk) for logistic regression:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \log(1 + e^{-y^{(i)}\theta^T x^{(i)}} = \frac{-1}{m}\sum_{i=1}^{m} \log(h_\theta(y^{(i)x^{(i)}})$$

   where $h_\theta(x) = g(\theta^T x)$ and $g(z) = \frac{1}{(1+e^{-z})}$.
   (a) Find the Hessian H of $J(\theta)$.
   (b) Should that H is positive semi-definite.
5. To prove the theorem (4.3.2), we use the proposition (2.3.1) about the tangents of convex functions. Prove this proposition by showing that $\forall x, y \in \mathbb{R}$ we have

$$f(y) \geq f(x) + f'(x)(y-x) \equiv f \text{ is convex}$$

6. Let $F : \mathbb{R}^3 \to \mathbb{R}$ defined by $F(x,y,z) = x^2 + y^2 + z^2 - xy - z$. Show that F is convex using the theorem (2.3.2).

2.5 Solutions to Exercises

1.
2.
3.

4.

5. To prove that equivalence, we need to prove
   - f convex $\implies f(y) \geq f(x) + f'(x)(y-x)$
   - $f(y) \geq f(x) + f'(x)(y-x) \implies f$ convex

   Let's start with the first implication. Let $x, y \in \mathbb{R}$, $x \neq y$ and $0 \leq t \leq 1$. Let's use the definition of convexity (2.3.1)

$$f((1-t)x + ty) \leq (1-t)f(x) + tf(y)$$

   As $(1-t)x + ty = x + t(y-x)$ we an write

$$\begin{aligned} f(x + t(y-x)) &\leq f(x) - tf(x) + tf(y) \\ f(x + t(y-x)) - f(x) &\leq t(f(y) - f(x)) \\ \frac{f(x + t(y-x)) - f(x)}{t} &\leq f(y) - f(x) \end{aligned}$$

   Now we want to use the definition of the derivative of f: $f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$. So here, we want h to be $t(y-x)$. To do so, we can multiply the left part of the inequality by $\frac{(y-x)}{(y-x)}$.

$$\frac{f(x + t(y-x)) - f(x)}{t(y-x)}(y-x) \leq f(y) - f(x)$$

   If $t \to 0$ then $t(y-x) \to 0$ so we can write

$$\begin{aligned} \lim_{t \to 0} \frac{f(x + t(y-x)) - f(x)}{t(y-x)}(y-x) &\leq \lim_{t \to 0} f(y) - f(x) \\ f'(x)(y-x) &\leq f(y) - f(x) \\ f(y) &\geq f(x) + f'(x)(y-x) \end{aligned}$$

   We proved that f convex $\implies f(y) \geq f(x) + f'(x)(y-x)$. Now let's prove that $f(y) \geq f(x) + f'(x)(y-x) \implies f$ convex. Let $x, y \in \mathbb{R}$, $z = tx + (1-t)y$ and $0 \leq t \leq 1$. So we have:

$$f(x) \geq f(z) + f'(z)(x-z) \tag{2.6}$$
$$f(y) \geq f(z) + f'(z)(y-z) \tag{2.7}$$

   We can multiply (2.6) by t:

$$tf(x) \geq tf(z) + tf'(z)(x-z)$$

   And multiply (2.7) by $(1-t)$:

$$(1-t)f(y) \geq (1-t)f(z) + (1-t)f'(z)(y-z)$$

Now let's sum the two inequalities:

$$\begin{aligned} tf(x)+(1-t)f(y) &\geq tf(z)+tf'(z)(x-z)+(1-t)f(z)+(1-t)f'(z)(y-z) \\ &\geq tf(z)+(1-t)f(z)+f'(z)(t(x-z)+(1-t)(y-z)) \\ &\geq f(z)+f'(z)(tx+y-z-ty) \\ &\geq f(z)+f'(z)(tx+(1-t)y-z) \end{aligned}$$

Since $z = tx+(1-t)y$ we have

$$\begin{aligned} &\geq f(z)+f'(z)(z-z) \\ &\geq f(z) \end{aligned}$$

So we have

$$tf(x)+(1-t)f(y) \geq f(tx+(1-t)y)$$

We proved that $f(y) \geq f(x)+f'(x)(y-x) \implies f$ convex. Thus, we proved that $f(y) \geq f(x)+f'(x)(y-x)$ is equivalent to f being convex.

6. Computing the gradient gives:

$$\nabla F(x,y,z) = \begin{bmatrix} 2x-y \\ 2y-x \\ 2z-1 \end{bmatrix}$$

Then, computing the hessian gives:

$$H(f)(x,y,z) = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

$H(f)(x,y,z)$ is symmetric and has an orthonormal eigenvector decomposition, and real eigenvalues. To determine if the matrix is positive definite, we can use the Sylvester's criterion : If all of the leading principal minors are positive (determinant of the sub-square matrices), then the matrix has positive eigenvalues, and is positive definite.

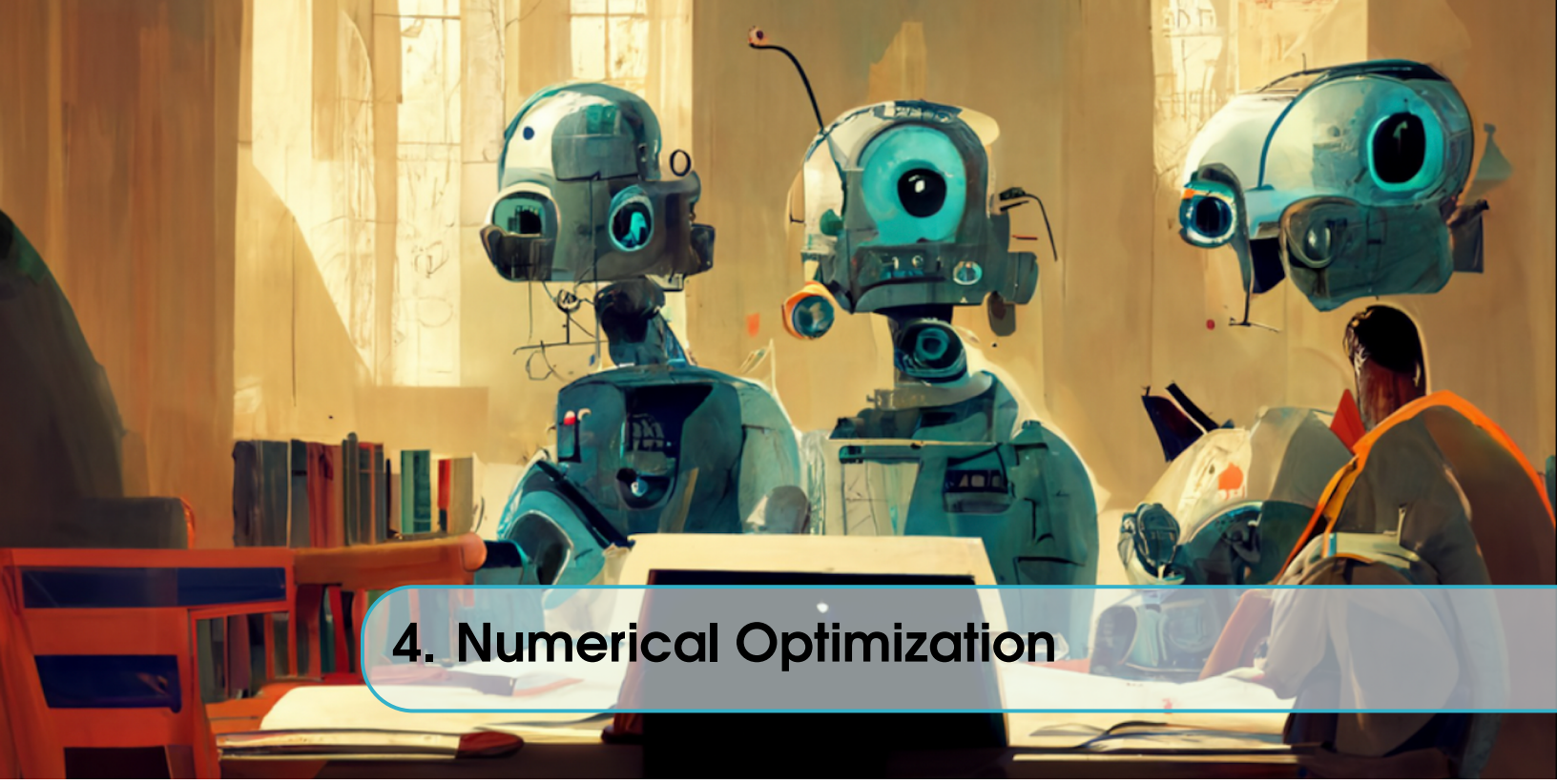
$$\det(2) = 2 > 0$$

$$\det\left(\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}\right) = 3 > 0$$

$$\det\left(\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}\right) = 6 > 0$$

All the leading principal minors are positive, so $H(f)(x,y,z)$ is positive definite, and f is strictly convex.

3. Probability Background



4. Numerical Optimization

4.1 Floating Point Calculations

4.2 Newton's Method

4.3 Gradient Based Optimization

4.3.1 More on Gradient Descent

Suppose that we let $\vec{x_0} \in \mathbb{R}^n$ to be our starting condition, $g = \nabla_x f(\vec{x_0})$, and $f : \mathbb{R}^n \to \mathbb{R}$. Our goal is to use gradient descent to minimize $f(\vec{x_0})$.

Thus we have the Hessian: $H = H(f)(\vec{x_0})$, and a direction to step in: $\hat{u} = \frac{\vec{x}-\vec{x_0}}{||\vec{x}-\vec{x_0}||_2}$. Then $f(\vec{x}) \approx f(\vec{x_0}) + (\vec{x}-\vec{x_0})^T\vec{g} + \frac{1}{2}(\vec{x}-\vec{x_0})^T H(\vec{x}-\vec{x_0})$. Note that this is only accurate locally. In other words, $f(\vec{x})$ is the composition of the initial condition, the gradient (first derivative) in direction $\vec{x}-\vec{x_0}$, and the Hessian (second derivative) in direction $\vec{x}-\vec{x_0}$.

Definition 4.3.1 Gradient Descent Step

Consider $\vec{x}$ as a combination of the initial position with a step in a certain direction. Let α be the step and $\vec{g}$ be the direction. Thus $f(x_0 - \alpha\vec{g}) \approx f(x_0) - \alpha\vec{g}^T\vec{g} + \frac{1}{2}\alpha^2\vec{g}^T H\vec{g}$ Assuming the Hessian is not zero there are 2 cases.

Case 1: Suppose f is positive definite. That implies that $\vec{g}^T H\vec{g} > 0$. Then to compute the optimal step size find where the derivative is 0:

$$\frac{d}{d\alpha} f(\vec{x_0} - \alpha\vec{g}) = -\vec{g}^T\vec{g} + \alpha\vec{g}^T H\vec{g}$$

$$\implies \alpha = \frac{\vec{g}^T\vec{g}}{\vec{g}^T H\vec{g}}$$

Case 2: Suppose $\vec{g}^T H\vec{g} < 0$. Then, by Taylor series, this will cause $f(\vec{x})$ decrease forever. (Note that this is probably not always the case since this approximation is only accurate locally)

Thus by finding what direction $\vec{g}$ the gradient is changing in, we can eventually find the extrema of $f(\vec{x})$

Recall, the 2nd Derivative Test for convexity from your calculus test, similarly we have the 2nd Derivative Test with Hessians.

Theorem 4.3.1 2nd Derivative Test
Suppose $\nabla_{\vec{x}} f(x) = \vec{0}$

1. If $H(f)(\vec{x_0})$ is positive definite, meaning $\vec{g}^T H \vec{g} > 0$, the $\vec{x_0}$ is a local minimum.
2. If $H(f)(\vec{x_0})$ is negative definite, meaning $\vec{g}^T H \vec{g} < 0$, the $\vec{x_0}$ is a local maximum.
3. If $H(f)(\vec{x_0})$ has negative and positive eigenvalues, meaning H is increasing and decreasing give different directions, the $\vec{x_0}$ is a saddle point.

R The condition number $\max_{i,j} |\frac{\lambda_i}{\lambda_j}|$ of H indicates how well a gradient descent step will preform. Specifically a large condition number would imply that the gradient step will over step.

■ **Example 4.1** Suppose in 2D $H(f) = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}$ ■

The issue here is... Fixes:

1. Use the Hessian in the algorithm
2. Normalize the data (similar to the change of variables which makes the level curves more circle; Hint: Logical Regression)

So, when can we guarantee that gradient descent or some variation with converge? -> in deep learning this will never happen.

4.3.2 Gradient Descent Convergence

How can we guarantee that gradient descent converges? It is impossible to guarantee for all functions. But for a specific class of functions, using convex optimization, we can provide such a theorem:

Theorem 4.3.2 — Gradient Descent Guarentee. Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable, convex, whose gradint is $\mathscr{L}$-Lipschitz continuous, and has a global minimum $\mathbf{x}^* \in \mathbb{R}^n$. If $\mathbf{x}_k$ is the k^{th} step of gradient descent with fixed step size $\alpha < \frac{1}{L}$, and $\mathbf{x}_0$ is the initialization point, then we have

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \frac{\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2}{2\alpha k}$$

R By choosing k large enough (performing several gradient descent steps), we can make $\frac{\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2}{2\alpha k}$ as small as necessary (as everything else in this expression is constant). That means that by going far enough in gradient descent, it is possible to guarentee that the value of $f(\mathbf{x}_k)$ is within any desired distance from the global minimum.

Proof. First, we need to use the fact that the gradient $\nabla_{\mathbf{x}} f(\mathbf{x})$ is $\mathscr{L}$-Lipschitz continuous implies that $H(f)(\mathbf{x}) - \mathscr{L} I$ is negative semi-definite. To prove this fact, let's introduce some notation. Let $f : \mathbb{R} \to \mathbb{R}, f_{\mathbf{x},\hat{\mathbf{u}}}(t) = f(\mathbf{x} + t\hat{\mathbf{u}}), \forall \mathbf{x}, \hat{\mathbf{u}} \in \mathbb{R}^n$ where $\hat{\mathbf{u}}$ is a unit vector. Then we have its derivative in

$t = 0$ equal to $f'_{\mathbf{x},\hat{\mathbf{u}}}(0) = \hat{\mathbf{u}}^T \nabla_{\mathbf{x}} f(\mathbf{x})$ which is the directional derivative in the direction of $\hat{\mathbf{u}}^T$. With this, we can show that $H(f)(\mathbf{x}) - \mathscr{L}I$ is negative semi-definite by proving that $\hat{\mathbf{u}}^T(H(f)(\mathbf{x}) - \mathscr{L}I)\hat{\mathbf{u}}$ is always lesser than or equal to 0:

$$\begin{aligned}\hat{\mathbf{u}}^T(H(f)(\mathbf{x}) - \mathscr{L}I)\hat{\mathbf{u}} &= \hat{\mathbf{u}}^T H(f)(\mathbf{x})\hat{\mathbf{u}} - \hat{\mathbf{u}}^T \mathscr{L}I\hat{\mathbf{u}} \\ &= \hat{\mathbf{u}}^T H(f)(\mathbf{x})\hat{\mathbf{u}} - \mathscr{L}\end{aligned}$$

Here, $\hat{\mathbf{u}}^T H(f)(\mathbf{x})\hat{\mathbf{u}}$ is the second derivative of f in direction of $\hat{\mathbf{u}}$. So this expression is actually equal to $f''_{\mathbf{x},\hat{\mathbf{u}}}(0) - \mathscr{L}$ and we can write

$$\begin{aligned}\hat{\mathbf{u}}^T(H(f)(\mathbf{x}) - \mathscr{L}I)\hat{\mathbf{u}} &= f''_{\mathbf{x},\hat{\mathbf{u}}}(0) - \mathscr{L} \\ &= \lim_{h\to 0} \frac{f'_{\mathbf{x},\hat{\mathbf{u}}}(h+0) - f'_{\mathbf{x},\hat{\mathbf{u}}}(0)}{h} - \mathscr{L} \\ &= \lim_{h\to 0} \hat{\mathbf{u}} \cdot \frac{\nabla_{\mathbf{x}} f(\mathbf{x}+h\hat{\mathbf{u}}) - \nabla_{\mathbf{x}} f(\mathbf{x})}{h} - \mathscr{L} \\ &= \lim_{h\to 0} \|\hat{\mathbf{u}}\|_2 \frac{\|\nabla_{\mathbf{x}} f(\mathbf{x}+h\hat{\mathbf{u}}) - \nabla_{\mathbf{x}} f(\mathbf{x})\|_2}{h} \cos(\theta) - \mathscr{L}\end{aligned}$$

With θ the angle between $\hat{\mathbf{u}}$ and $\frac{\nabla_{\mathbf{x}} f(\mathbf{x}+h\hat{\mathbf{u}}) - \nabla_{\mathbf{x}} f(\mathbf{x})}{h}$. Here, the maximum value that this expression can take is when these two vectors are parallel, which gives $\cos(\theta) = 1$. We can also note that $\|\hat{\mathbf{u}}\|_2$ is equal to 1, as $\hat{\mathbf{u}}$ is a unit vector. So we can write

$$\hat{\mathbf{u}}^T(H(f)(\mathbf{x}) - \mathscr{L}I)\hat{\mathbf{u}} \leq \lim_{h\to 0} \frac{\|\nabla_{\mathbf{x}} f(\mathbf{x}+h\hat{\mathbf{u}}) - \nabla_{\mathbf{x}} f(\mathbf{x})\|_2}{h} - \mathscr{L}$$

And by the Lipschitz condition we have

$$\begin{aligned}\hat{\mathbf{u}}^T(H(f)(\mathbf{x}) - \mathscr{L}I)\hat{\mathbf{u}} &\leq \lim_{h\to 0} \frac{\mathscr{L}\|h\hat{\mathbf{u}} - 0\|_2}{h} - \mathscr{L} \\ &\leq \lim_{h\to 0} \frac{\mathscr{L}h}{h} - \mathscr{L} = 0\end{aligned}$$

Thus, we showed that $\hat{\mathbf{u}}^T(H(f)(\mathbf{x}) - \mathscr{L}I)\hat{\mathbf{u}} \leq 0$, so $H(f)(\mathbf{x}) - \mathscr{L}I$ is negative semi-definite. Now we can use this fact. Let's start by writing this fact in a convenient form for the proof:

$$\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n, (\mathbf{x}-\mathbf{y})^T(H(f)(\mathbf{z}) - \mathscr{L}I)(\mathbf{x}-\mathbf{y}) \leq 0$$

Now we will use Taylor's reminder theorem. In $\mathbb{R}$, the Taylor reminder theorem says that $\forall x, y \in \mathbb{R}$:

$$f(y) = \sum_{i=0}^{n-1} \frac{f^{(i)}(x)}{i!}(y-x)^i + R_{n-1}(y)$$

and $\exists z \in [x, y]$ such that

$$R_{n-1}(y) = \frac{f^{(n)}(z)}{n!}(y-x)^n$$

where $R_{n-1}(x)$ is reminder.
So here, $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ by the Taylor reminder theorem, $\exists \mathbf{z} \in \mathbb{R}^n$ such that $\|\mathbf{z}-\mathbf{x}\|_2 \leq \|\mathbf{y}-\mathbf{x}\|_2$ (or equivalently $\mathbf{z} \in N_x(\|\mathbf{y}-\mathbf{x}\|_2)$. By performing a second order expansion of f around $f(\mathbf{x})$ we have

$$f(\mathbf{y}) = f(\mathbf{x}) + \nabla_{\mathbf{x}} f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + \frac{1}{2}(\mathbf{y}-\mathbf{x})^T H(f)(\mathbf{z})(\mathbf{y}-\mathbf{x}) \tag{4.1}$$

Here, the reminder is controlled by the quadratic term $\frac{1}{2}(\mathbf{y}-\mathbf{x})^T H(f)(\mathbf{z})(\mathbf{y}-\mathbf{x})$. As $\nabla_{\mathbf{x}} f(\mathbf{x})$ is $\mathscr{L}$-Lipschitz continuous, we showed that it implies that $H(f)(\mathbf{x}) - \mathscr{L} I$ is negative semi-definite. From this, we have

$$\begin{aligned}(\mathbf{x}-\mathbf{y})^T (H(f)(\mathbf{z}) - \mathscr{L} I)(\mathbf{x}-\mathbf{y}) &\leq 0 \\ (\mathbf{x}-\mathbf{y})^T (H(f)(\mathbf{z}))(\mathbf{x}-\mathbf{y}) &\leq \mathscr{L} \|\mathbf{y}-\mathbf{x}\|_2^2\end{aligned}$$

So we can plug this result into (4.1)

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla_{\mathbf{x}} f(\mathbf{x})^T (\mathbf{y}-\mathbf{x}) + \frac{1}{2}\mathscr{L} \|\mathbf{y}-\mathbf{x}\|_2^2$$

Now, we want to look at what happens on a gradient descent step by setting $\mathbf{y} = \mathbf{x} - \alpha \nabla_{\mathbf{x}} f(\mathbf{x})$:

$$\begin{aligned}f(\mathbf{x} - \alpha \nabla_{\mathbf{x}} f(\mathbf{x})) &\leq f(\mathbf{x}) + \nabla_{\mathbf{x}} f(\mathbf{x})^T (\mathbf{x} - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}) - \mathbf{x}) + \frac{1}{2}\mathscr{L} \|\mathbf{x} - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}) - \mathbf{x}\|_2^2 \\ &= f(\mathbf{x}) - \nabla_{\mathbf{x}} f(\mathbf{x})^T \alpha \nabla_{\mathbf{x}} f(\mathbf{x}) + \frac{1}{2}\mathscr{L} \|\alpha \nabla_{\mathbf{x}} f(\mathbf{x})\|_2^2 \\ &= f(\mathbf{x}) - \alpha \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2^2 + \frac{1}{2}\mathscr{L}\alpha^2 \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2^2 \\ &= f(\mathbf{x}) - (1 - \frac{1}{2}\mathscr{L}\alpha)\alpha \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2^2 \end{aligned} \tag{4.2}$$

Using $0 < \alpha \leq \frac{1}{\mathscr{L}}$, we can write

$$\begin{aligned}0 < \frac{1}{2}\mathscr{L}\alpha &\leq \frac{1}{2} \\ 0 > -\frac{1}{2}\mathscr{L}\alpha &\geq -\frac{1}{2} \\ 1 > 1 - \frac{1}{2}\mathscr{L}\alpha &\geq \frac{1}{2}\end{aligned}$$

Plugging this result into (4.2) gives

$$f(\mathbf{x} - \alpha \nabla_{\mathbf{x}} f(\mathbf{x})) \leq f(\mathbf{x}) - \frac{1}{2}\alpha \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2^2 \tag{4.3}$$

Since $\frac{1}{2}\alpha \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2^2$ is always positive (unless $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$), we can say that the sequence $(f(x_i))_{i\geq 0}$ (where x_i is the i^{th} step of gradient descent) is weakly decreasing. As $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla_{\mathbf{x}_i} f(\mathbf{x}_i)$, we can rewrite (4.3):

$$f(\mathbf{x}_{i+1}) = f(\mathbf{x}_i - \alpha \nabla_{\mathbf{x}_i} f(\mathbf{x}_i)) \leq f(\mathbf{x}_i) - \frac{1}{2}\alpha \|\nabla_{\mathbf{x}_i} f(\mathbf{x}_i)\|_2^2 \tag{4.4}$$

Now let's use f($\mathbf{x}^*$) the global minimum to bound $f(\mathbf{x} - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}))$. To do so, we will use the property (2.3.1). Since f is convex, we can write:

$$\begin{aligned}f(\mathbf{x}^*) &\geq f(\mathbf{x}) + \nabla_{\mathbf{x}} f(\mathbf{x})^T (\mathbf{x}^* - \mathbf{x}) \\ f(\mathbf{x}) &\leq f(\mathbf{x}^*) + \nabla_{\mathbf{x}} f(\mathbf{x})^T (\mathbf{x} - \mathbf{x}^*)\end{aligned}$$

Plugging this into (4.4) gives

$$f(\mathbf{x}_{i+1}) \leq f(\mathbf{x}^*) + \nabla_{\mathbf{x}_i} f(\mathbf{x}_i)^T (\mathbf{x}_i - \mathbf{x}^*) - \frac{1}{2}\alpha \|\nabla_{\mathbf{x}_i} f(\mathbf{x}_i)\|_2^2$$

Since $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla_{\mathbf{x}_i} f(\mathbf{x}_i)$, we have $\nabla_{\mathbf{x}_i} f(\mathbf{x}_i) = \frac{\mathbf{x}_i - \mathbf{x}_{i+1}}{\alpha}$. So we can write

$$\begin{aligned}
f(\mathbf{x}_{i+1}) &\leq f(\mathbf{x}^*) + \frac{1}{\alpha}(\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}^*) - \frac{1}{2\alpha}\|\mathbf{x}_i - \mathbf{x}_{i+1}\|_2^2 \\
&= f(\mathbf{x}^*) + \frac{1}{\alpha}(\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}^*) - \frac{1}{2\alpha}(\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}_{i+1}) \\
&= f(\mathbf{x}^*) - \frac{1}{2\alpha}(-2(\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}^*) + (\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}_{i+1}))
\end{aligned}$$

Here, we notice that $(\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}_{i+1})$ is a quadratic term, and $(\mathbf{x}_i - \mathbf{x}_{i+1})^T$ is a linear term. So we can complete the square:

$$\begin{aligned}
f(\mathbf{x}_{i+1}) &\leq f(\mathbf{x}^*) - \frac{1}{2\alpha}((\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}_{i+1}) - 2(\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}^*)) \\
&= f(\mathbf{x}^*) - \frac{1}{2\alpha}((\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}_{i+1}) - 2(\mathbf{x}_i - \mathbf{x}_{i+1})^T(\mathbf{x}_i - \mathbf{x}^*) \\
&\qquad + (\mathbf{x}_i - \mathbf{x}^*)^T(\mathbf{x}_i - \mathbf{x}^*) - (\mathbf{x}_i - \mathbf{x}^*)^T(\mathbf{x}_i - \mathbf{x}^*)) \\
&= f(\mathbf{x}^*) - \frac{1}{2\alpha}(((\mathbf{x}_i - \mathbf{x}_{i+1}) - ((\mathbf{x}_i - \mathbf{x}^*))^T((\mathbf{x}_i - \mathbf{x}_{i+1}) - ((\mathbf{x}_i - \mathbf{x}^*)) \\
&\qquad - (\mathbf{x}_i - \mathbf{x}^*)^T(\mathbf{x}_i - \mathbf{x}^*)) \\
&= f(\mathbf{x}^*) - \frac{1}{2\alpha}((\mathbf{x}^* - \mathbf{x}_{i+1})^T(\mathbf{x}^* - \mathbf{x}_{i+1}) - (\mathbf{x}_i - \mathbf{x}^*)^T(\mathbf{x}_i - \mathbf{x}^*)) \\
f(\mathbf{x}_{i+1}) - f(\mathbf{x}^*) &\leq -\frac{1}{2\alpha}(\|\mathbf{x}^* - \mathbf{x}_{i+1}\|_2^2 - \|\mathbf{x}_i - \mathbf{x}^*\|_2^2)
\end{aligned}$$

This inequality holds for x_{i+1} on every iteration of gradient descent. If k steps of gradient descent is performed, we can sum this inequality over all the k iterations which gives:

$$\sum_{i=0}^{k}(f(\mathbf{x}_i) - f(\mathbf{x}^*)) \leq \sum_{i=0}^{k}\frac{1}{2\alpha}(\|\mathbf{x}^* - \mathbf{x}_i\|_2^2 - \|\mathbf{x}^* - \mathbf{x}_{i+1}\|_2^2)$$

If we expand the sum, we notice that this is a telescoping sum, so most of the terms cancels each other:

$$\begin{aligned}
&\|\mathbf{x}^* - \mathbf{x}_0\|_2^2 - \|\mathbf{x}^* - \mathbf{x}_1\|_2^2 \\
&+ \|\mathbf{x}^* - \mathbf{x}_1\|_2^2 - \|\mathbf{x}^* - \mathbf{x}_2\|_2^2 \\
&+ \qquad\qquad \vdots \\
&+ \|\mathbf{x}^* - \mathbf{x}_{k-1}\|_2^2 - \|\mathbf{x}^* - \mathbf{x}_k\|_2^2 \\
&= \|\mathbf{x}^* - \mathbf{x}_0\|_2^2 - \|\mathbf{x}^* - \mathbf{x}_k\|_2^2
\end{aligned}$$

So we can remove the right summation and obtain:

$$\begin{aligned}
\sum_{i=0}^{k}(f(\mathbf{x}_i) - f(\mathbf{x}^*)) &\leq \frac{1}{2\alpha}(\|\mathbf{x}^* - \mathbf{x}_0\|_2^2 - \|\mathbf{x}^* - \mathbf{x}_k\|_2^2) \\
&\leq \frac{1}{2\alpha}\|\mathbf{x}^* - \mathbf{x}_0\|_2^2
\end{aligned}$$

Finally, we can use the fact that the sequence $(f(x_i))_{i\geq 0}$ is weakly decreasing on every iteration, and say that $\forall i < k$:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq f(\mathbf{x}_i) - f(\mathbf{x}^*)$$
$$\sum_{i=1}^{k}(f(\mathbf{x}_k) - f(\mathbf{x}^*)) \leq \sum_{i=1}^{k}(f(\mathbf{x}_i) - f(\mathbf{x}^*))$$
$$k(f(\mathbf{x}_k) - f(\mathbf{x}^*)) \leq \frac{1}{2\alpha}\|\mathbf{x}^* - \mathbf{x}_0\|_2^2$$
$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \frac{1}{2\alpha k}\|\mathbf{x}^* - \mathbf{x}_0\|_2^2$$

Which conclude the proof. ■

4.4 Exercises

1.
2. Consider the function $f : \mathbb{R} \to \mathbb{R}$ defined by $f(y) = \sqrt{y}$.
   (a) Find the second order Taylor approximation at $x = 5$
   (b) Use the Taylor's reminder theorem to deduce how accurate this approximation is for $4 \leq y \leq 6$
3. Consider using an iterative optimization algorithm (such as Newton's method, or gradient descent) to minimize some continuously differentiable function $f(x)$. Suppose we initialize the algorithm at $x^{(0)} = 0$. When the algorithm is run, it will produce a value of $x \in \mathbb{R}^n$ for each iteration: $x^{(1)}, x^{(2)}, \ldots$.
   Now, let some non-singular square matrix $A \in \mathbb{R}^{n\times n}$ be given, and define a new function

   $$g(z) = f(Az)$$

   Consider using the same iterative optimization algorithm to optimize g
   (with initialization $z(0) = 0$). If the values $z^{(1)}, z^{(2)}, \ldots$ produced by this method necessarily satisfy

   $$z(i) = A^{-1}x^{(i)}$$

   for all i, we say this optimization algorithm is invariant to linear reparameterizations.
   (a) Show that Newton's method (applied to find the minimum of a function) is invariant to linear reparameterizations.
   (b) Is gradient descent invariant to linear reparameterizations? Justify your answer.
4. Consider the projected gradient decent approach to constrained optimization as discussed in 2.2.1. Provide an example for when this method fails to find a global maximum. Hint: consider optimizing over a non-convex set.
5. Let S be the unit circle in $\mathbb{R}^2$. Set up the Lagrangian for optimizing $f(x,y)$ over the interior of SS
6. Minimize $f(\mathbf{x}) = -x_1 + -x_2$ subject to the constraints $x_1 + 2x_2 \leq 3$ and $3x_1 + x_2 \leq 5$ using KKT multipliers.

4.5 Solutions to Exercises

1. (placeholder for #1 in Exercises)

2. (a) The second order Taylor approximation of a function f is

$$P_2(y) = f(x) + \frac{f'(x)}{1!}(y-x) + \frac{f''(x)}{2!}(y-x)^2$$

Here we have $f(x) = \sqrt{x}$ and $x = 5$.

$$\begin{aligned} f(x) &= \sqrt{x} & f(5) &= \sqrt{5} = 2.2361 \\ f'(x) &= \frac{1}{2}x^{-1/2} & f'(5) &= \frac{1}{2}5^{-1/2} = 0.2236 \\ f''(x) &= -\frac{1}{4}x^{-3/2} & f''(5) &= -\frac{1}{4}5^{-3/2} = -0.0224 \end{aligned}$$

Thus, the second order Taylor approximation is

$$f(y) \approx P_2(y) = \sqrt{x} + \frac{1}{2}x^{-1/2}(y-x) - \frac{1}{8}x^{-3/2}(y-x)^2$$

With $x = 5$ we have

$$P_2(y) = \sqrt{5} + \frac{1}{2\sqrt{5}}(y-5) - \frac{1}{8 \times 5^{3/2}}(y-5)^2$$

(b) By the Taylor's reminder theorem, $\exists\, z$ between 5 and x such that

$$R_2(y) = \frac{f^{(3)}(z)}{3!}(y-5)^3$$

Here, $f^{(3)}(z) = \frac{3}{8}z^{-5/2} = \frac{3}{8z^{2/5}}$ so we have

$$R_2(y) = \frac{(x-5)^3}{16z^{2/5}}$$

For y we have

$$4 \leq y \leq 6$$

$$-1 \leq y-5 \leq 1$$

So we have $|y-5| \leq 1$ and $|y-5|^3 \leq 1$. Since $y \geq 4$ we have $z^{5/2} \geq 4^{5/2} = 4^2\sqrt{4} = 32$. Then, we can write

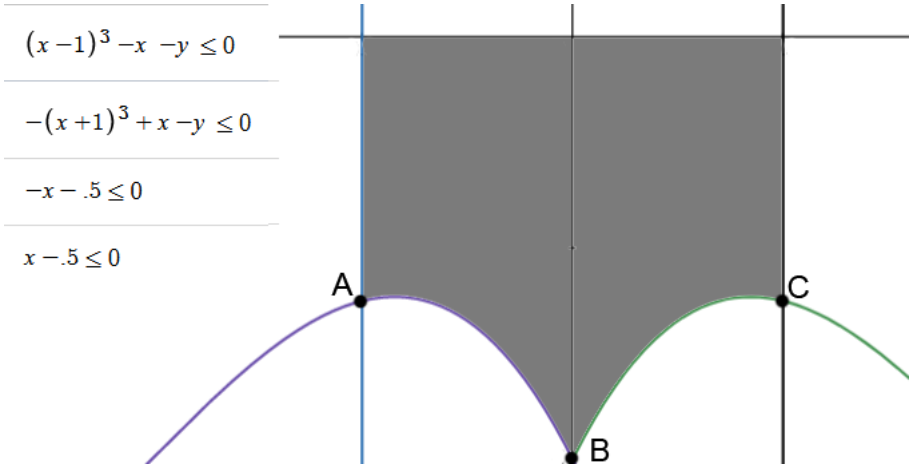
$$\begin{aligned} |R_2(y)| &= \frac{|y-5|^3}{16z^{5/2}} \\ &\leq \frac{1}{16 \times 32} \\ &\leq 0.002 \end{aligned}$$

So if $4 \leq y \leq 6$, the Taylor approximation $P_2(y)$ is accurate within 0.002. We can verify this upper bound by computing the real error for $y = 4$ and $y = 6$:

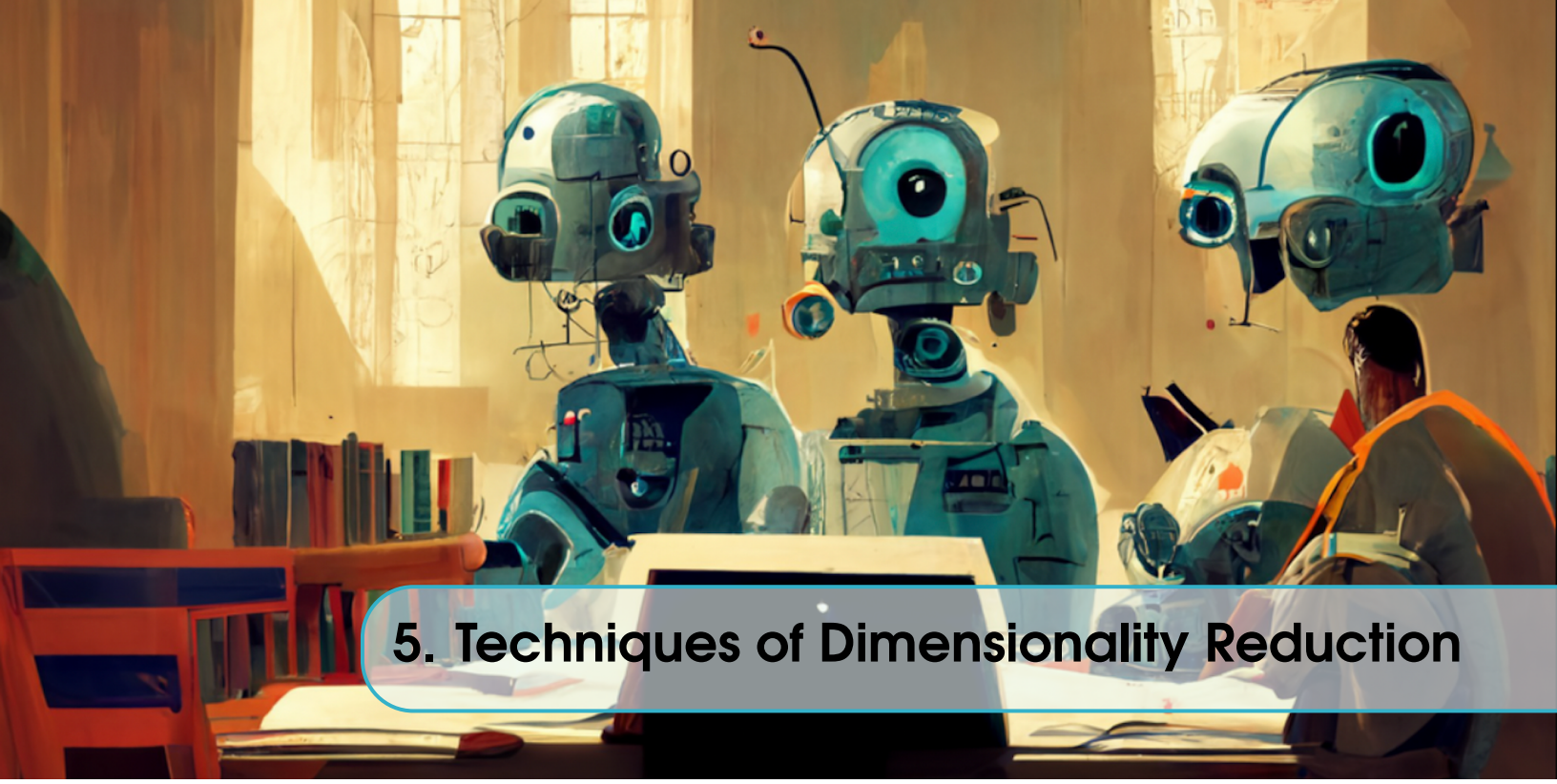
$$|\sqrt{4} - P_2(4)| = 0.0013 < 0.002$$

$$|\sqrt{6} - P_2(6)| = 0.0001 < 0.002$$

3. Consider minimizing y over a set of this form:



   Note the global minimum B and points A and C which lie on the boundary and lead to local minimums outside the boundary. Running projected gradient decent in a neighborhood of A or C will result in getting stuck against the regions boundary.
4. $L(f) = f + \alpha(x^2 - y^2 - 1)$
5. $$L(f) = -x_1 + -x_2 + \alpha_1(x_1 + 2x_2 - 3) + \alpha_2(3x_1 + x_2 - 5)$$
   $$\frac{\partial L(f)}{\partial x_1} = -1 + \alpha_1 + 3\alpha_2$$
   $$\frac{\partial L(f)}{\partial x_2} = -1 + 2\alpha_2 + \alpha_2$$
   solving this linear system yields $\alpha_1 = \frac{2}{5}\alpha_2 = \frac{1}{5}$. By the complementary slackness condition of KKT, this gives us $x_1 + 2x_2 - 3 = 0$ and $3x_1 + x_2 = 0$. solving this linear system yields $x_1 = \frac{11}{5}$ and $x_2 = \frac{-4}{5}$



5. Techniques of Dimensionality Reduction

5.1 Principal Component Analysis

Principal component analysis (PCA) is closely related to singular value decomposition (SVD). The goal is compression. Often, we don't know which components of our data will be important, so for each data point, we record more information than necessary. A single point might have thousands of components of information. Principle Component Analysis ranks each component by importance; then we trim the unhelpful components. This simplifies our data so that we can easily correlations. In short, principal component analysis analyzes our data to highlight the important/principal components.

We begin with a data set of m elements. Each element has n components. So our data set is $\mathbf{X} \in \mathbb{R}^{m \times n}$, and we will trim this to a set of k elements. We also want the trimming to be reversible, so we need two functions, one that trims/encodes the data and one that rebuilds/decodes it. $g : \mathbb{R}^n \to \mathbb{R}^k$ will be our encoding function, and $f : \mathbb{R}^k \to \mathbb{R}^n$ will be our decoding function. We cannot perfectly reverse the trimming, so when defining these functions, we want to minimize the loss. We want:

$$f(g(\mathbf{x})) \approx \mathbf{x}$$

To achieve this, we define $g(\mathbf{x})$ as the vector that minimizes the difference between $\mathbf{x}$ and $f(g(\mathbf{x}))$:

$$g(\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}} \|\mathbf{x} - f(\mathbf{y})\|_2^2$$

(argmin means the argument $\mathbf{y}$ that minimizes the expression $\|\mathbf{x} - f(\mathbf{y})\|_2^2$. Also, we do not need to square $\|\mathbf{x} - f(\mathbf{y})\|_2$, but doing so simplifies computations and the minimum $\mathbf{y}$ will still be the same.)

Our decoding function f will simply multiply g with some matrix $\mathbf{D}$:

$$f(g(\mathbf{x})) = \mathbf{D}g(\mathbf{x})$$

(In general, $f(\mathbf{y}) = \mathbf{D}\mathbf{y}$.) $\mathbf{D}$ is a matrix in $\mathbb{R}^{n \times k}$ with orthonormal columns. (I.e., the columns of $\mathbf{D}$ are all orthogonal with each other and are all unit vectors.)

Now, we can evaluate (step-by-step) $g(\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}} \|\mathbf{x} - f(\mathbf{y})\|_2^2$ to find the minimum $\mathbf{y}$:

$$g(\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}} \|\mathbf{x} - f(\mathbf{y})\|_2^2$$

$\underset{\mathbf{y}}{\operatorname{argmin}} \|\mathbf{x} - f(\mathbf{y})\|_2^2$ returns the largest eigenvalue of the matrix $(\mathbf{x} - f(\mathbf{y}))^\top (\mathbf{x} - f(\mathbf{y}))$, but in this case $(\mathbf{x} - f(\mathbf{y}))^\top (\mathbf{x} - f(\mathbf{y}))$ evaluates to a 1×1 matrix, so the matrix itself is also the eigenvalue. Therefore, $\underset{\mathbf{y}}{\operatorname{argmin}} \|\mathbf{x} - f(\mathbf{y})\|_2^2 = (\mathbf{x} - f(\mathbf{y}))^\top (\mathbf{x} - f(\mathbf{y}))$, so:

$$\begin{aligned} g(\mathbf{x}) &= \underset{\mathbf{y}}{\operatorname{argmin}} ((\mathbf{x} - f(\mathbf{y}))^\top (\mathbf{x} - f(\mathbf{y}))) \\ &= \underset{\mathbf{y}}{\operatorname{argmin}} (\mathbf{x}^\top \mathbf{x} - f(\mathbf{y})^\top \mathbf{x} - \mathbf{x}^\top f(\mathbf{y}) + f(\mathbf{y})^\top f(\mathbf{y})) \end{aligned}$$

Note: The actual value of this expression does not matter; we are just looking for the $\mathbf{y}$ that minimizes it, so the constant terms like $\mathbf{x}^\top \mathbf{x}$ have no effect on the result, and we can safely eliminate them:

$$g(\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}} (-f(\mathbf{y})^\top \mathbf{x} - \mathbf{x}^\top f(\mathbf{y}) + f(\mathbf{y})^\top f(\mathbf{y}))$$

We can add $f(\mathbf{y})^\top \mathbf{x}$ and $\mathbf{x}^\top f(\mathbf{y})$ since both are equivalent to $f(\mathbf{y}) \cdot \mathbf{x}$.

$$\begin{aligned} g(\mathbf{x}) &= \underset{\mathbf{y}}{\operatorname{argmin}} (-2\mathbf{x}^\top f(\mathbf{y}) + f(\mathbf{y})^\top f(\mathbf{y})) \\ &= \underset{\mathbf{y}}{\operatorname{argmin}} (-2\mathbf{x}^\top (\mathbf{D}\mathbf{y}) + (\mathbf{D}\mathbf{y})^\top (\mathbf{D}\mathbf{y})) \\ &= \underset{\mathbf{y}}{\operatorname{argmin}} (-2\mathbf{x}^\top \mathbf{D}\mathbf{y} + \mathbf{y}^\top \mathbf{D}^\top \mathbf{D}\mathbf{y}) \end{aligned}$$

Since $\mathbf{D}$ is orthogonal $\mathbf{D}^\top \mathbf{D} = \mathbf{I}$.

$$g(\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}} (-2\mathbf{x}^\top \mathbf{D}\mathbf{y} + \mathbf{y}^\top \mathbf{y})$$

We can now find the minimum $\mathbf{y}$ by taking the derivative with respect to $\mathbf{y}$ and setting it equal to 0.

$$0 = \frac{\partial}{\partial \mathbf{y}} (-2\mathbf{x}^\top \mathbf{D}\mathbf{y} + \mathbf{y}^\top \mathbf{y}) = -2\frac{\partial}{\partial \mathbf{y}} (\mathbf{x}^\top \mathbf{D}\mathbf{y}) + \frac{\partial}{\partial \mathbf{y}} (\mathbf{y}^\top \mathbf{y})$$

We use two rules of matrix differentiation to evaluate this. For $\frac{\partial}{\partial \mathbf{y}} (\mathbf{y}^\top \mathbf{y})$, we use:

Theorem 5.1.1 Given a constant matrix $\mathbf{A}$:

$$\frac{\partial}{\partial \mathbf{y}} (\mathbf{y}^\top \mathbf{A}\mathbf{y}) = \mathbf{y}^\top (\mathbf{A} + \mathbf{A}^\top)$$

For $\mathbf{A}$ we can substitute the identity matrix to get:

$$0 = -2\frac{\partial}{\partial \mathbf{y}} (\mathbf{x}^\top \mathbf{D}\mathbf{y}) + \mathbf{y}^\top (\mathbf{I} + \mathbf{I}^\top) = -2\frac{\partial}{\partial \mathbf{y}} (\mathbf{x}^\top \mathbf{D}\mathbf{y}) + 2\mathbf{y}^\top$$

We use a more complicated rule for the remaining derivative:

Theorem 5.1.2 For functions f and g and matrix $\mathbf{A}$ with compatible dimensions:

$$\frac{\partial}{\partial \mathbf{y}} f(\mathbf{y})^\top \mathbf{A} g(\mathbf{y}) = g(\mathbf{y})^\top \mathbf{A}^\top \frac{\partial}{\partial \mathbf{y}} f(\mathbf{y}) + f(\mathbf{y})^\top \mathbf{A} \frac{\partial}{\partial \mathbf{y}} g(\mathbf{y})$$

(In our case, $f(\mathbf{y}) = \mathbf{x}, \mathbf{A} = \mathbf{D}$, and $g(\mathbf{y}) = \mathbf{y}$.) We then get the following:

$$\begin{aligned}
0 &= -2\frac{\partial}{\partial \mathbf{y}}(\mathbf{x}^\top \mathbf{D}\mathbf{y}) + 2\mathbf{y}^\top \\
&= -2\left(\mathbf{y}^\top \mathbf{D}^\top \frac{\partial}{\partial \mathbf{y}}\mathbf{x} + \mathbf{x}^\top \mathbf{D}\frac{\partial}{\partial \mathbf{y}}\mathbf{y}\right) + 2\mathbf{y}^\top \\
&= -2(0 + \mathbf{x}^\top \mathbf{D}\mathbf{I}) + 2\mathbf{y}^\top \\
&= -2\mathbf{x}^\top \mathbf{D} + 2\mathbf{y}^\top \\
&= -\mathbf{x}^\top \mathbf{D} + \mathbf{y}^\top
\end{aligned}$$

This implies $\mathbf{x}^\top \mathbf{D} = \mathbf{y}^\top$, and we can take the transpose of both sides and get $\mathbf{y} = \mathbf{D}^\top \mathbf{x}$. Therefore, our optimal encoding function is:

$$g(\mathbf{x}) = \mathbf{D}^\top \mathbf{x}$$

Now we need to solve for $\mathbf{D}$. Recall that our goal is to get $f(g(\mathbf{x}))$ as close as possible to $\mathbf{x}$, but we are trying to do this for every data point in our set. So we want the matrix $\mathbf{D}$ that accumulates the least error over all the our data points:

$$\begin{aligned}
\mathbf{D} &= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} \sum_{i=1}^{m} \|\mathbf{x}^{(i)} - f(g(\mathbf{x}^{(i)}))\|_2^2 \\
&= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} \sum_{i=1}^{m} \|\mathbf{x}^{(i)} - f(\mathbf{C}^\top \mathbf{x}^{(i)})\|_2^2 \\
&= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} \sum_{i=1}^{m} \|\mathbf{x}^{(i)} - \mathbf{C}\mathbf{C}^\top \mathbf{x}^{(i)}\|_2^2
\end{aligned}$$

($\mathbf{C}$ is just another name for our matrix $\mathbf{D}$, and $\mathbf{x}^{(i)} \in \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(m)}\} \in \mathbb{R}^n$ is the ith data point of our data set. We are squaring the expression for convenience, which does not affect the result.)

We do not want to deal with the complications of a summation; we can rewrite the equation in terms of matrices. We essentially replace $\mathbf{x}^{(i)}$ with our entire data set $\mathbf{X} \in \mathbb{R}^{m \times n}$:

$$\mathbf{X} = \begin{bmatrix} \dots & \mathbf{x}^{(1)} & \dots \\ & \vdots & \\ \dots & \mathbf{x}^{(m)} & \dots \end{bmatrix}$$

Which allows us to simplify our definition of $\mathbf{D}$ to be:

$$\begin{aligned}\mathbf{D} &= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} \|\mathbf{X}-\mathbf{X}\mathbf{C}\mathbf{C}^\top\|_{\text{Fro}}^2 \\ &= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} \operatorname{Tr}((\mathbf{X}-\mathbf{X}\mathbf{C}\mathbf{C}^\top)^\top(\mathbf{X}-\mathbf{X}\mathbf{C}\mathbf{C}^\top)) \\ &= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} \operatorname{Tr}((\mathbf{X}^\top-\mathbf{C}\mathbf{C}^\top\mathbf{X}^\top)(\mathbf{X}-\mathbf{X}\mathbf{C}\mathbf{C}^\top)) \\ &= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} \operatorname{Tr}(\mathbf{X}^\top\mathbf{X}-\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top-\mathbf{C}\mathbf{C}^\top\mathbf{X}^\top\mathbf{X}+\mathbf{C}\mathbf{C}^\top\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top)\end{aligned}$$

As before, we can eliminate constant terms because this will not change how $\mathbf{C}$ minimizes the expression:

$$\mathbf{D} = \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} \operatorname{Tr}(-\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top-\mathbf{C}\mathbf{C}^\top\mathbf{X}^\top\mathbf{X}+\mathbf{C}\mathbf{C}^\top\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top)$$

We can also distribute the trace function since it is linear:

$$\mathbf{D} = \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} (-\operatorname{Tr}(\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top)-\operatorname{Tr}(\mathbf{C}\mathbf{C}^\top\mathbf{X}^\top\mathbf{X})+\operatorname{Tr}(\mathbf{C}\mathbf{C}^\top\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top))$$

Now, we use the property $\operatorname{Tr}(\mathbf{AB}) = \operatorname{Tr}(\mathbf{BA})$ to move $\mathbf{C}\mathbf{C}^\top$ to the end of the second and third terms. In the third term, this allows us to use the fact $\mathbf{C}^\top\mathbf{C} = \mathbf{I}$:

$$\begin{aligned}\mathbf{D} &= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} (-\operatorname{Tr}(\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top)-\operatorname{Tr}(\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top)+\operatorname{Tr}(\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top\mathbf{C}\mathbf{C}^\top)) \\ &= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} (-\operatorname{Tr}(\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top)-\operatorname{Tr}(\mathbf{C}\mathbf{C}^\top\mathbf{X}^\top\mathbf{X})+\operatorname{Tr}(\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top)) \\ &= \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmin}} (-\operatorname{Tr}(\mathbf{X}^\top\mathbf{X}\mathbf{C}\mathbf{C}^\top))\end{aligned}$$

We remove the negative sign by changing argmin to argmax:

$$\mathbf{D} = \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmax}} (\operatorname{Tr}(\mathbf{C}^\top\mathbf{X}^\top\mathbf{X}\mathbf{C}))$$

We have thus proven the following lemma:

Lemma 5.1.3 The optimal encoding function is $g(\mathbf{x}) = \mathbf{D}^\top\mathbf{x}$, where $\mathbf{D} = \underset{\mathbf{C}:\mathbf{C}^\top\mathbf{C}=\mathbf{I}}{\operatorname{argmax}} (\operatorname{Tr}(\mathbf{C}^\top\mathbf{X}^\top\mathbf{X}\mathbf{C})$.

This is close to the final result, but we can do more to find $\mathbf{D}$. We want to show that the columns of $\mathbf{D}$ are the largest k eigenvalues of $\mathbf{X}^\top\mathbf{X}$. The first step is to show that this is true for $k = 1$. In this case, $\mathbf{C}$ is simply a unit vector, so $\mathbf{C}^\top\mathbf{X}^\top\mathbf{X}\mathbf{C}$ is a 1×1 matrix. This then implies that $\operatorname{Tr}(\mathbf{C}^\top\mathbf{X}^\top\mathbf{X}\mathbf{C}) = \mathbf{C}^\top\mathbf{X}^\top\mathbf{X}\mathbf{C}$. We observe that the matrix $\mathbf{X}^\top\mathbf{X}$ is symmetric, so by Spectral Theorem, $\mathbf{X}^\top\mathbf{X}$ has n orthonormal eigenvectors with non-negative eigenvalues. We denote the eigenvectors as $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n$, where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n \geq 0$.

Because the eigenvectors are orthogonal, they form a basis of $\mathbb{R}^{n\times n}$. This implies that $\mathbf{C} \in \mathbb{R}^n$ is a linear combination of the eigenvectors:

$$\mathbf{C} = \sum_{i=1}^{n} a_i\mathbf{v}_i$$

(Since $\mathbf{C}$ is a unit vector, $\sum_{i=1}^{n} a_i^2 = 1$.) We make the above substitution and continue with the goal of maximizing $\mathbf{C}^\top \mathbf{X}^\top \mathbf{X}\mathbf{C}$:

$$\mathbf{C}^\top \mathbf{X}^\top \mathbf{X}\mathbf{C} = \left(\sum_{i=1}^{n} a_i \mathbf{v}_i\right)^\top \mathbf{X}^\top \mathbf{X} \left(\sum_{j=1}^{n} a_j \mathbf{v}_j\right)$$

We can rewrite this as a single summation:

$$\mathbf{C}^\top \mathbf{X}^\top \mathbf{X}\mathbf{C} = \sum_{i,j=1}^{n} a_i \mathbf{v}_i^\top \mathbf{X}^\top \mathbf{X} a_j \mathbf{v}_j$$

$$= \sum_{i,j=1}^{n} a_i \mathbf{v}_i^\top \mathbf{X}^\top \mathbf{X} a_j \mathbf{v}_j$$

$$= \sum_{i,j=1}^{n} a_i a_j \mathbf{v}_i^\top \mathbf{X}^\top \mathbf{X} \mathbf{v}_j$$

Since $\mathbf{v}_j$ is an eigenvector, $\mathbf{X}^\top \mathbf{X}\mathbf{v}_j = \lambda_j \mathbf{v}_j$:

$$\mathbf{C}^\top \mathbf{X}^\top \mathbf{X}\mathbf{C} = \sum_{i,j=1}^{n} a_i a_j \mathbf{v}_i^\top \lambda_j \mathbf{v}_j = \sum_{i,j=1}^{n} a_i a_j \lambda_j \mathbf{v}_i^\top \mathbf{v}_j$$

Because the eigenvectors are orthonormal, $\mathbf{v}_i^\top \mathbf{v}_j = 0$ except when $i = j$, in which case $\mathbf{v}_i^\top \mathbf{v}_i = 1$.

$$\mathbf{C}^\top \mathbf{X}^\top \mathbf{X}\mathbf{C} = \sum_{i=1}^{n} a_i a_i \lambda_i = \sum_{i=1}^{n} a_i^2 \lambda_i$$

Because λ_1 is defined to be the largest eigenvalue and $\sum_{i=1}^{n} a_i^2 = 1$, $\mathbf{C}^\top \mathbf{X}^\top \mathbf{X}\mathbf{C}$ is maximized in the case where $a_1 = 1$ and $a_2 = a_3 = \ldots = a_n = 0$. This gives us $\mathbf{C}^\top \mathbf{X}^\top \mathbf{X}\mathbf{C} = \lambda_1$, but more importantly, these conditions give us the maximizing $\mathbf{C}$:

$$\mathbf{C} = \sum_{i=1}^{n} a_i \mathbf{v}_i = \mathbf{v}_1$$

This resulting vector is known as our first principal component. This leads to our final conclusion:

Theorem 5.1.4 For principal component analysis, the optimal encoding function is $g : \mathbb{R}^n \to \mathbb{R}^k$ with:

$$g(\mathbf{x}) = \mathbf{D}^\top \mathbf{x},$$

and the optimal decoding function is $f : \mathbb{R}^k \to \mathbb{R}^n$ with:

$$f(\mathbf{y}) = \mathbf{D}\mathbf{y},$$

where $\mathbf{D} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_k \end{bmatrix}$ and $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ correspond to the largest k eigenvalues of $\mathbf{X}^\top \mathbf{X}$ in descending order. ($\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ are the first k principal components of $\mathbf{X}$.)

The principal components are the important features of our data points, which we want to keep; we can trim the rest using our encoding function.

(Note: We have only proven this theorem for the case $k = 1$. A full proof is left to the reader.)

5.2 Exercises

1. Given is the data set below, find the first two principal components; then use the optimal encoding function g to fit the data into $\mathbb{R}^{4\times 2}$. (Ignore the "Player Name" column for your calculations.)

Sport Wins			
Player Name	Golf	100 yd Sprint	Hide & Seek
Peter	0	0	4
Max	0	2	0
Sebastian	0	2	0
Timmy	1	0	0
Matt	3	0	0

2. Use f to decode the your encoded data set from Exercise 1.
3. The functions g and f encode/decode one data point at a time. Define functions G and F that encode/decode the entire data set at once.

5.3 Solutions to Exercises

1. The data set we are using is:

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 4 \\ 0 & 2 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{5\times 3}$$

We first calculate $\mathbf{X}^\top\mathbf{X}$:

$$\mathbf{X}^\top\mathbf{X} = \begin{bmatrix} 0 & 0 & 0 & 1 & 3 \\ 0 & 2 & 2 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 4 \\ 0 & 2 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 16 \end{bmatrix}$$

Now, we want the eigenvectors and eigenvalues. In this case, the matrix is diagonal, so the eigenvalues are simply the diagonal entries (10 , 8, and 16), and the eigenvectors are $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^\top$, $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^\top$, and $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$. We arrange the eigenvalues in descending order, so we have $\lambda_1 = 16, \lambda_2 = 10$, and $\lambda_3 = 8$ with corresponding eigenvectors $\mathbf{v}_1 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$, $\mathbf{v}_2 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^\top$, and $\mathbf{v}_3 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^\top$. Therefore, the first two principal components are $\mathbf{v}_1$ and $\mathbf{v}_2$, and our encoding function is:

$$g(\mathbf{x}) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}^\top \mathbf{x} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \mathbf{x}$$

We now encode each data point.

$$g\left(\begin{bmatrix}0\\0\\4\end{bmatrix}\right) = \begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\end{bmatrix}\begin{bmatrix}0\\0\\4\end{bmatrix} = \begin{bmatrix}4\\0\end{bmatrix}$$

$$g\left(\begin{bmatrix}0\\2\\0\end{bmatrix}\right) = \begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\end{bmatrix}\begin{bmatrix}0\\2\\0\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

$$g\left(\begin{bmatrix}0\\2\\0\end{bmatrix}\right) = \begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\end{bmatrix}\begin{bmatrix}0\\2\\0\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

$$g\left(\begin{bmatrix}1\\0\\0\end{bmatrix}\right) = \begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix}$$

$$g\left(\begin{bmatrix}3\\0\\0\end{bmatrix}\right) = \begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\end{bmatrix}\begin{bmatrix}3\\0\\0\end{bmatrix} = \begin{bmatrix}0\\3\end{bmatrix}$$

Thus, our encoded data set is:

$$\begin{bmatrix}4 & 0\\0 & 0\\0 & 0\\0 & 1\\0 & 3\end{bmatrix}$$

2. We apply f to every data point in our new set.

$$f\left(\begin{bmatrix}4\\0\end{bmatrix}\right) = \begin{bmatrix}0 & 1\\0 & 0\\1 & 0\end{bmatrix}\begin{bmatrix}4\\0\end{bmatrix} \approx x^{(1)}$$

$$f\left(\begin{bmatrix}0\\0\end{bmatrix}\right) = \begin{bmatrix}0 & 1\\0 & 0\\1 & 0\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} \approx x^{(2)}$$

$$f\left(\begin{bmatrix}0\\0\end{bmatrix}\right) = \begin{bmatrix}0 & 1\\0 & 0\\1 & 0\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} \approx x^{(3)}$$

$$f\left(\begin{bmatrix}0\\1\end{bmatrix}\right) = \begin{bmatrix}0 & 1\\0 & 0\\1 & 0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} \approx x^{(4)}$$

$$f\left(\begin{bmatrix}0\\3\end{bmatrix}\right) = \begin{bmatrix}0 & 1\\0 & 0\\1 & 0\end{bmatrix}\begin{bmatrix}0\\3\end{bmatrix} \approx x^{(5)}$$

Thus, our decoded data set is:

$$\begin{bmatrix} 0 & 0 & 4 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix}$$

3. $G(\mathbf{X}) = \mathbf{XD} = \mathbf{X}_{encoded}$
   $F(\mathbf{X}_g) = \mathbf{X}_g\mathbf{D}^\top = \mathbf{X}_{decoded}$

6. Regression and Classification

6.1 Basics of Machine Learning

6.1.1 Learning Algorithms

To understand machine learning, we must begin with the definition of a learning algorithm.

Definition 6.1.1 A *learning algorithm* is a computer program which accomplishes some task (which we will denote T) based on some experience (E) with respect to some performance measure (P), such that performance P increases as we add experience to E.

R This is not a true definition of a learning algorithm, but we will understand what is meant by E, T, and P in this section through examples.

6.1.2 Experience

In a machine learning context, experience is a set of examples that we wish to learn from. Similar to the colloquial use, a learning algorithm 'experiences' these examples and uses them for a variety of tasks, including making predictions or generating new information.

Definition 6.1.2 *Experience* E is a dataset $\mathbf{X} \in \mathbb{R}^{m \times n}$, where each column is called a *feature* and each row is called an *example*.

■ **Example 6.1** A classic example is the Iris dataset. Suppose we have the following measurements for 150 Iris plants.

This dataset is our experience. It may also include a vector $\mathbf{y}$ that catagorizes the plants. For

$$\mathbf{X} = \begin{bmatrix} 2.5 & 1.2 & 4.5 & 2.6 \\ 3 & .5 & 3.4 & 4.3 \\ 4.2 & 1.3 & 4.3 & 1.2 \\ \vdots & \vdots & \vdots & \vdots \\ 3.1 & .7 & 3 & 4.1 \end{bmatrix}$$

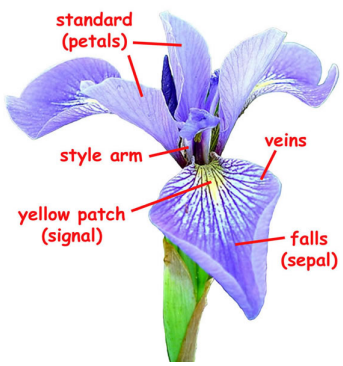


Figure 6.1: (Left) Dataset of Iris feature measurements with columns petal length, petal with, sepal length, and sepal width in that order. (Right) Diagram of an Iris with features labeled.

example, we could know that each flower is one of three species, and have the following vector $\mathbf{y}$:

$$\mathbf{X} = \begin{bmatrix} 3 \\ 1 \\ 2 \\ \vdots \\ 1 \end{bmatrix}$$

■

Definition 6.1.3 A learning algorithm with both $\mathbf{X}, \mathbf{y}$ is called *supervised*. A learning algorithm with $\mathbf{X}$ and no label vector is called *unsupervised*.

When using an unsupervised algorithm, we determine a probability distribution $p(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^n$ corresponds to $x_{i,*}$. When using a supervised algorithm, we usually want to determine the probability $p(y|\mathbf{x})$.

Definition 6.1.4 If vy takes discrete values, we call this a *classification* problem. If vy takes continuous values, we call this *regression* problem.

6.1.3 Task

The next part to understand about learning algorithms is the task T. Again, we will improve our understanding through the examples of classification and regression.

The task T of (supervised) classification is generally the following: Given $(\mathbf{X}, \mathbf{y})$ with $y_i \in \{1, 2, \ldots, k\}$, produce a function $f : \mathbb{R}^n \to \{1, 2, \ldots, k\}$ such that $f(\mathbf{x}) = y$ performs well as a prediction.

■ **Example 6.2** Suppose $\mathbf{X}_{i,*}$ contains the pixel data of a grayscale image, and $\mathbf{y}$ has the labels for cats and dogs, say 1 for cats and 2 for dogs arbitrarily. Then one possible task T for a learning algorithm could be to classify a given image $\mathbf{x}$ as a 1 or 2, or a dog or a cat. ■

The task T of regression is similar to that of classification: Given $(\mathbf{X}, \mathbf{y})$ with $y_i \in \mathbb{R}$, produce a function $f : \mathbb{R}^n \to \mathbb{R}$ such that $f(\mathbf{x}) = y$ performs well as a prediction.

■ **Example 6.3** Housing prices are notoriously difficult to predict, but you might want to use data about houses to determine a good price. One way to do this is to collect data on houses and their

prices, such as square footage, number of bedrooms, number of bathrooms, and proximity to schools. Each house's data would be kept in $\mathbf{X}_{i,*}$, with the corresponding price in y_i. The task T of a learning algorithm with this information could be to predict the price of a house given the data. ∎

There are many other possible examples for regression tasks, such as producing text in Unicode given an image of text, translation between languages, generating new examples from data in experience E, filling in missing entries from $\mathbf{X}$, or removing unwanted noise from $\mathbf{X}$. In general, defining a task is important because gives meaning to the algorithm. That being said, it is not possible to know how well a given algorithm is doing its task without a performance measure.

6.1.4 Performance Measure

Performance measures P generally depend on the task T. For classification algorithms, we often use accuracy, precision, or recall as the performance measure (we will discuss these in future sections). For regression, we often use some sort of average squared error in prediction. This performance is not measured from the experience set E, but from a different set of examples, $\mathbf{X}^{test}$ and the corresponding $\mathbf{y}^{test}$.

∎ **Example 6.4** Let's continue the example for housing prices. Suppose we used a supervised algorithm using experience E, say $\mathbf{X}^{train}$ and $\mathbf{y}^{train}$. The task T is to generate a function $f : \mathbb{R}^n \to \mathbb{R}$. The performance measure P is the average squared error between $f(\mathbf{X}^{test})$ and $\mathbf{y}^{test}$. With this information, we can build a learning algorithm, a computer program, that will attempt to minimize this average squared error. ∎

6.2 Linear Regression

Having covered the basics, we can now discuss how learning algorithms can optimize this performance measure based on the task and experience. For the supervised linear regression model we built in the previous example, we will assume that y_i depends linearly on $\mathbf{X}_{i,*}$. That is, there exist vectors $\mathbf{w}, \mathbf{b}$ such that

$$y_i \approx \mathbf{X}_{i,*}\mathbf{w} + \mathbf{b}$$

This is somewhat inconvenient to write with $\mathbf{b}$ being its own vector, so we introduce the notation

$$\hat{\mathbf{X}} := \begin{bmatrix} 1 & X_{1,1} & X_{1,2} & \cdots & X_{1,n} \\ 1 & X_{2,1} & X_{2,2} & \cdots & X_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{m,1} & X_{m,2} & \cdots & X_{m,n} \end{bmatrix}$$

and by optimizing $\mathbf{w}$ (now an element of $\mathbb{R}^{n+1}$), we get the same result as optimizing $\mathbf{w}$ and $\mathbf{b}$ as before. Our optimization will consist of finding the optimal $\mathbf{w}^*$ such that

$$\frac{1}{m}\sum_{i=1}^{m} \|\hat{\mathbf{X}}_{i,*}\mathbf{w} - y_i\|_2^2 = \frac{1}{m}\|\hat{\mathbf{X}}\mathbf{w} - \mathbf{y}\|_2^2$$

is minimized. We will call this our cost function.

Definition 6.2.1 A *cost function* is a function $J : \mathbb{R}^n \to \mathbb{R}$ that takes parameters from experience E and has an argument $\mathbf{w}$ for which it returns the "cost."

R J takes on many names, such as cost function, loss function, or error function. The task T is usually related to minimizing J, but if we wanted to maximize J, we might not call it a cost function, as our intuition tells us that we want to minimize costs. One way to get around this is to minimize the negative of J, where $-J$ would be the cost function. This will maximize J.

For this particular cost function, we take parameters $\mathbf{X}$ (changed to $\hat{\mathbf{X}}$) and $\mathbf{y}$ from E. One solution to finding the optimal $\mathbf{w}$ to minimize this function is to use gradient descent to approximate $\mathbf{w}^*$.

Proposition 6.2.1 J is twice-differentiable, convex, has Lipschitz-continuous gradient, has gradient

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{2}{m}(\hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{w} - \hat{\mathbf{X}}^\top \mathbf{y})$$

and has Hessian

$$H(J)(\mathbf{w}) = \frac{2}{m}\hat{\mathbf{X}}^\top \hat{\mathbf{X}}$$

Proof. Let $J(\mathbf{w}) = \frac{1}{m}(\|\hat{\mathbf{X}}\mathbf{w} - \mathbf{y}\|_2^2) = \frac{1}{m}((\hat{\mathbf{X}}\mathbf{w} - \mathbf{y})^\top(\hat{\mathbf{X}}\mathbf{w} - \mathbf{y})) = \frac{1}{m}(\mathbf{w}^\top \hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{w} - \mathbf{w}^\top \hat{\mathbf{X}}^\top \mathbf{y} - \mathbf{y}^\top \hat{\mathbf{X}}\mathbf{w} + \mathbf{y}^\top \mathbf{y})$. Because this is a real value ($J(\mathbf{w}) \in \mathbb{R}$), we can rewrite this as

$$J(\mathbf{w}) = \frac{1}{m}(\mathbf{w}^\top \hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{w} - 2\mathbf{y}^\top \hat{\mathbf{X}}\mathbf{w} + \mathbf{y}^\top \mathbf{y})$$

Using our matrix differentiation rules discussed in Section 2.1, we can find

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}} J(\mathbf{w})^\top = \frac{1}{m}(\mathbf{w}^\top(\hat{\mathbf{X}}^\top \hat{\mathbf{X}} + (\hat{\mathbf{X}}^\top \hat{\mathbf{X}})^\top) - 2\mathbf{y}^\top \hat{\mathbf{X}})^\top = \frac{2}{m}(\hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{w} - \hat{\mathbf{X}}^\top \mathbf{y})$$

Now to see this is Lipschitz continuous, let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^{n+1}$. Then

$$\|\nabla_{\mathbf{v}} J(\mathbf{v}) - \nabla_{\mathbf{w}} J(\mathbf{w})\|_2 = \frac{2}{m}\|(\hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{v} - \hat{\mathbf{X}}^\top \mathbf{y}) - (\hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{w} - \hat{\mathbf{X}}^\top \mathbf{y})\|_2 = \frac{2}{m}\|\hat{\mathbf{X}}^\top \hat{\mathbf{X}}(\mathbf{v} - \mathbf{w})\|_2$$

$$\leq \frac{2}{m}\|\hat{\mathbf{X}}^\top \hat{\mathbf{X}}\|_2 \|\mathbf{v} - \mathbf{w}\|_2$$

By setting $\mathscr{L} = \frac{2}{m}\|\hat{\mathbf{X}}^\top \hat{\mathbf{X}}\|_2$ (a constant), we see the gradient is $\mathscr{L}$-Lipschitz continuous. Now we calculate the Hessian, $H(J)(\mathbf{w})$ with our matrix differentiation rules again.

$$H(J)(\mathbf{w}) = \frac{\partial^2}{\partial \mathbf{w}^2} J(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}}\left(\frac{2}{m}(\hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{w} - \hat{\mathbf{X}}^\top \mathbf{y})\right) = \frac{2}{m}(\hat{\mathbf{X}}^\top \hat{\mathbf{X}})$$

$\hat{\mathbf{X}}^\top \hat{\mathbf{X}}$ is positive semidefinite because $\mathbf{w}^\top \hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{w} = \|\hat{\mathbf{X}}\mathbf{w}\|_2^2 \geq 0$. Because the Hessian is positive semidefinite, we know J is convex. Therefore, we have shown that J is twice-differentiable, convex, has Lipschitz-continuous gradient, has the gradient and Hessian shown above. ■

From the section on Gradient Descent Convergence, we have the properties to know that gradient descent will converge to the global minimum, $\mathbf{w}^*$, for this particular cost function.

Gradient descent is a computationally expensive algorithm, and depending on the choices of stopping conditions, it may not converge in a reasonable amount of time. Another solution to finding $\mathbf{w}^*$ is to calculate it directly.

Proposition 6.2.2 If $\hat{\mathbf{X}}$ has linearly independent columns, and if J has a global minimum, $\mathbf{w}^*$, then $\mathbf{w}^* = (\hat{\mathbf{X}}^\top \hat{\mathbf{X}})^{-1}\hat{\mathbf{X}}^\top \mathbf{y}$.

Proof. Since J is differentiable, if $\mathbf{w}^*$ is a global minimum, then $\mathbf{w}^*$ is a local minimum, so $\nabla_{\mathbf{w}^*} J(\mathbf{w}^*) = \mathbf{0}$. We calculated $\nabla_{\mathbf{w}^*} J(\mathbf{w}^*)$ in the previous problem, so we simply calculate:

$$\nabla_{\mathbf{w}^*} J(\mathbf{w}^*) = \frac{2}{m}(\hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{w}^* - \hat{\mathbf{X}}^\top \mathbf{y}) = \mathbf{0}$$

$$\iff \hat{\mathbf{X}}^\top \hat{\mathbf{X}}\mathbf{w}^* = \hat{\mathbf{X}}^\top \mathbf{y}$$

Because $\hat{\mathbf{X}}$ has linearly independent columns, we know $\hat{\mathbf{X}}^\top \hat{\mathbf{X}}$ is positive definite. This means it is invertible, so we have

$$\mathbf{w}^* = (\hat{\mathbf{X}}^\top \hat{\mathbf{X}})^{-1} \hat{\mathbf{X}}^\top \mathbf{y}$$

Thus we can calculate the minimum, $\mathbf{w}^*$, directly. It can, however, be computationally more expensive to invert matrices, so sometimes it is faster if we simply run gradient descent, which we know will converge. ■

In the case that y_i is a polynomial function of $\mathbf{X}_{i,*}$, suppose it has maximum total degree m. That is,

$$f(x_1, x_2, \ldots, x_n) = f(\mathbf{x}) = \sum_{\alpha \in \mathbb{Z}^n} c_\alpha x^\alpha,$$

where $1 \leq \sum_{i=1}^{n} \alpha_i \leq m$. Here is an example of one term of such a polynomial:

■ **Example 6.5**

$$\alpha = (1,0,2,3) \implies x^\alpha = x_1^1 x_2^0 x_3^2 x_4^3 = x_1 x_3^2 x_4^3$$

The power of x_i is determined by the i-th index of α. ■

To solve the case where we assume y_i is a polynomial function of $\mathbf{X}_{i,*}$, we add columns to $\mathbf{X}$, one for each valid $\alpha \in \mathbb{Z}^n$. Again, we use an example to show what we mean:

■ **Example 6.6** Suppose $\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 4 \end{bmatrix}$. If we assume the maximum total degree is 2, we add columns to $\mathbf{X}$ and fill in the values:

$$\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 4 \end{bmatrix} \rightarrow \begin{matrix} x_1 & x_2 & x_1^2 & x_1 x_2 & x_2^2 \end{matrix} \\ \begin{bmatrix} 1 & 2 & 1 & 2 & 4 \\ 2 & 1 & 4 & 2 & 1 \\ 3 & 4 & 9 & 12 & 16 \end{bmatrix}$$

■

Doing this sets up a linear relationship between y_i and the new $\mathbf{X}_{i,*}$ because it now contains all of the possible polynomial terms up to degree m of the columns. Then we can form $\hat{\mathbf{X}}$ as before and solve as in the case of a linear relationship. There is, of course, the problem of picking the "best" maximum total degree m, but this problem will be covered in Section 7.2 when we discuss hyperparameters.

6.3 Logistic Regression

Logistic Regression is an instantiation of a Classification Algorithm. The Learning Algorithm is setup as follows

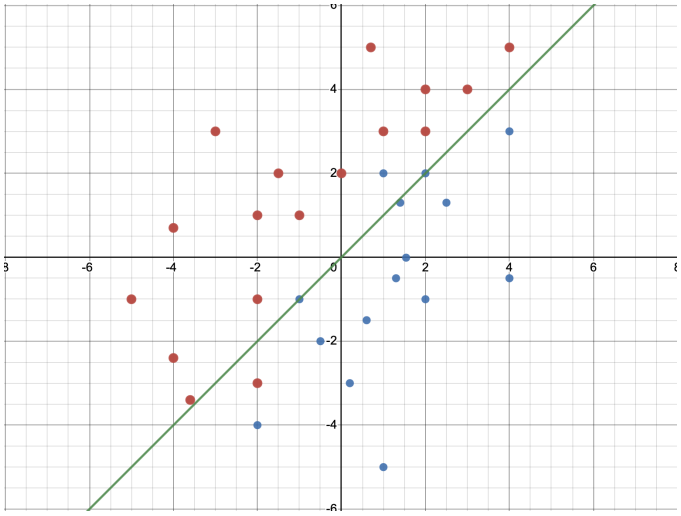
1. Experience (E): The dataset X is setup as a m instances, each with n features. More formally, $X \in \mathbb{R}^{m \times n}$. The goal of this algorithm is to predict a binary label for each instance. So, the ground-truth $\mathbf{y}$ is a vector containing only ones and zeros. So, $\mathbf{y} \in \{0,1\}^m$.
2. Task (T): By observation from the dataset formulation, it becomes clear the goal is to produce a function $f : \mathbb{R}^n \to \{0,1\}$ learned from each instance m. We can then use pass this value through a piece-wise function to assign labels.
3. Performance Measure (P): Many choices exist, let's go with accuracy:

$$\mathbf{Acc} = \frac{\#\{i \mid \text{pred}_f(\mathbf{X}^{(\text{test})}_{i,*}) = \mathbf{y}^{(\text{test})}_i\}}{\#\text{rows in } \mathbf{X}^{(\text{test})}}$$

6.3.1 One Solution: Logistic Regression

Now, let's proceed with an illustration of one solution to the desiderata presented above. Here, there are two cases depending on the nature of the data.

1. Case 1: If we expect $y = 1$ examples are reliably separated from $y = 0$ by a hyperplane in $\mathbb{R}^n$.



Goal: Find $\mathbf{w} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}$ such that the hyperplane $\vec{\mathbf{w}}^T \vec{\mathbf{X}} + \mathbf{b} = 0$ [1] separates the 1's from 0's "reliably". [2]

The next question that arises is how to use the optimization machinery previously discussed to learn $\vec{\mathbf{w}}$ and $\mathbf{b}$. Toward this, we need to define a cost function $J(\vec{\mathbf{w}}, \mathbf{b})$ such that minimizing this objective, maximizes our Performance Measure (Accuracy).

Observations about current setup

The key observation to be made is that $\vec{\mathbf{w}}\,\vec{\mathbf{x}} + \mathbf{b} \in \mathbb{R}$. So, $\vec{\mathbf{w}}^T \vec{\mathbf{x}} + \mathbf{b} << 0$ suggests that the model strongly predicts a 0 or $\vec{\mathbf{w}}^T \vec{\mathbf{x}} + \mathbf{b} >> 0$ suggests that the model strongly predicts a 1.

[1] Notice that $\vec{\mathbf{w}}$ is a normal vector

[2] There is no concrete definition of a reliable. In this context, we are referring to a dataset which logistic regression would be able to do well on.

However, the issue here is that we have a prediction [3] in $\mathbb{R}$ and not in $[0,1]$. So, to proceed forward we need to find a 'normalization' function.

Desiderata for the Normalization function

We want our normalization function $g : \mathbb{R} \to [0,1]$ to satisfy the following properties:

$$\lim_{z \to \infty} g(z) = 1$$

$$\lim_{z \to -\infty} g(z) = 0$$

$$g(0) = \frac{1}{2}$$

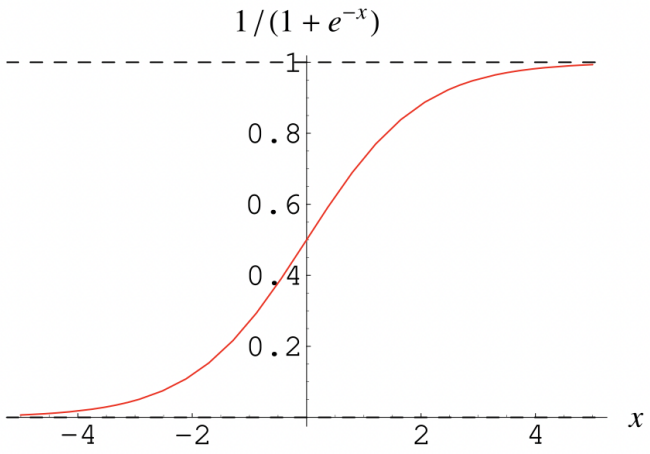
Using a function which satisfies the above properties, we are now able to assign labels through the following piece-wise function:

$$pred_f(x) = \begin{cases} 1 & \text{if } x > 0.5 \\ 0 & \text{elif } x \leq 0.5 \end{cases}$$

The Sigmoid Function

Define $g : \mathbb{R} \to [0,1]$ as:

$$g(x) = \frac{1}{1+e^{-x}}$$



Proposition 6.3.1 (a) The Sigmoid Function satisfies all the outlined desiderata of a normalization function.

(b) $g'(x) = g(x)(1 - g(x))$

Proof. This is left as an exercise to the reader. ■

[3] Such an unnormalized probability distribution is sometimes called an 'energy'. This 'energy'-based formulation of Deep Learning allows for a method to rewrite the optimization function of several algorithms such as Variational Autoencoders and certain Self-Supervised Learning algorithms as an energy function which can then be optimized using your favorite gradient-based optimizer.

Designing the Cost Function $J(\overrightarrow{\mathbf{w}}, \mathbf{b})$

For sake of notational brevity, we use the $\hat{\mathbf{X}}$ notation from previous sections. This allows the inference function to be rewritten as $\hat{\mathbf{X}}\mathbf{v}$ instead of $\overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{X}} + \mathbf{b}$.
We have to have a cost function which encourages two cases:

$$g(\overrightarrow{\mathbf{x}}^T \overrightarrow{\mathbf{v}}) >> 0 \; if \; y = 1$$

$$g(\overrightarrow{\mathbf{x}}^T \overrightarrow{\mathbf{v}}) << 1 \; if \; y = 0$$

Intuitively, we want to penalize the inverse behavior. If the ground-truth is 1 we want to penalize on learning values which would lead to a 0 prediction and vice-versa.
Another way to express this is to penalize with the following scheme:

$$\overrightarrow{\mathbf{v}} \; if \; y = 1 \; \& - \log(g(\overrightarrow{\mathbf{x}}^T \overrightarrow{\mathbf{v}})) >>> 0$$

$$\overrightarrow{\mathbf{v}} \; if \; y = 0 \; \& - \log(1 - g(\overrightarrow{\mathbf{x}}^T \overrightarrow{\mathbf{v}})) >>> 0$$

The final bit of insight we can exploit is in the nature of the problem. In all cases, $\mathbf{y}$ is either 0 or 1. So, we can use this fact to serve as a gate between the two cases of the cost function. Hence, we can penalize $\mathbf{v}$ if $-\mathbf{y}\log(g(\overrightarrow{\mathbf{x}}^T \overrightarrow{\mathbf{v}})) - (1-\mathbf{y})\log(1 - g(\overrightarrow{\mathbf{x}}^T \overrightarrow{\mathbf{v}}))$ is large.
Finally, all we are left to do is accumulating this cost function over the whole train dataset.

Definition 6.3.1 Given $\mathbf{X}^{(train)}, \mathbf{y}^{(train)}$, define $log : \mathbb{R}^m \to \mathbb{R}^m$

$$\log(x_1, ..., x_m) = (\log(x_1), ..., \log(x_m))$$

and $G : \mathbb{R}^m \to \mathbb{R}^m$

$$G(x_1, ..., x_m) = (g(x_1), ..., g(x_m))$$

$$J(\mathbf{v}) = \frac{1}{m} \sum_{i=1}^{m} -(1 - \mathbf{y}_i)\log(1 - g(\overrightarrow{\mathbf{x}_{i,*}}^T \overrightarrow{\mathbf{v}})) - \mathbf{y}\log(g(\overrightarrow{\mathbf{x}_i}^T \overrightarrow{\mathbf{v}}))$$

which can be rewritten in the more conventient notation of matrix multiplication:

$$J(\mathbf{v}) = \frac{1}{m}[-(\overrightarrow{1} - \overrightarrow{\mathbf{y}})^T \log(\overrightarrow{1} - \overrightarrow{G}(\hat{\mathbf{X}}\overrightarrow{\mathbf{v}})) - \overrightarrow{\mathbf{y}}^T \log(G(\hat{\mathbf{X}}\overrightarrow{\mathbf{v}})]$$

and find $\mathbf{v}^* = argmin(J(\mathbf{v}))$ using some form of gradient descent.

Proposition 6.3.2

$$\nabla v J(\mathbf{v}) = \frac{1}{m}(G(\hat{\mathbf{X}}\overrightarrow{\mathbf{v}}) - \overrightarrow{\mathbf{y}})^T \hat{\mathbf{X}}$$

$$H(J)(\mathbf{v}) = \frac{1}{m}\hat{\mathbf{X}}^T D(\mathbf{v}) \hat{\mathbf{X}}$$

$$D(\mathbf{v}) = diag(G'(\hat{\mathbf{X}}\overrightarrow{\mathbf{v}}))$$

$$G'(x_1, ..., x_m) = (g'(x_1), ..., g'(x_m))$$

J is twice differentiable convex, and has Lipschitz continuous gradient.
Therefore gradient descent reliably finds the global minimum (if it exists).

Proof. This proof is left as an exercise to the reader. ■

2. Case 2: A non-linear "Decision Boundary"
   In this case, one may find this nonlinear decision boundary by modifying x by adding new columns corresponding to polygons or other functions of current columns. [4]

6.4 Exercises

1. We discussed the tasks of classification and regression algorithms. Provide possible tasks for the following experience sets, and state whether the learning algorithm would be classification or regression.
   (a) $\mathbf{X}$ with red, green, and blue pixel values as the columns and $\mathbf{y}$ with numbers corresponding to colors (for example, 0 for red, 1 for blue, 2 for green, 3 for yellow)
   (b) $\mathbf{X}$ with data that seems to cluster around different areas in $\mathbb{R}^n$.
   (c) $\mathbf{X}$ with data about three notes in one bar of a 4/4 time signature and $\mathbf{y}$ with the fourth note.
2. Suppose we have $\mathbf{X}, \mathbf{y}$:
$$\mathbf{X} = \begin{bmatrix} 5 & 3 & 1 \\ 4 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 31 \\ 34 \\ 30 \end{bmatrix}$$
   and a cost function:
$$J(\mathbf{w}) = \frac{1}{3}\|\hat{\mathbf{X}}\mathbf{w} - \mathbf{y}\|_2^2$$
   Given $\mathbf{v} = \begin{bmatrix} 1 \\ 0.6 \\ 7 \\ 3 \end{bmatrix}$, calculate $J(\mathbf{v})$.
3. Define $\mathbf{X}$:
$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 2 \\ 3 & 3 & 5 \end{bmatrix}$$
   Assume there is a polynomial relationship between the associated $\mathbf{y}$ and $\mathbf{X}$, and that the maximum total degree of said polynomial is 2. Set up the new $\mathbf{X}$ such that the relationship between some y_i and $\mathbf{X}_{i,*}$ is linear but represents the polynomial combinations of the columns of the current $\mathbf{X}$.

6.5 Solutions to Exercises

1. Here are some possible solutions:
   (a) A classification algorithm that would take in $\mathbf{x} \in \mathbb{R}^3$ and output a number corresponding to the predicted color.
   (b) Suppose the data in $\mathbf{X}$ clusters around k different points. Then a regression algorithm with this dataset might first determine what the k different center points in $\mathbb{R}^n$ then, given $\mathbf{x} \in \mathbb{R}^n$, classify which of the k center points it clusters around.

[4] Playing around with the dataset in this way is called *feature engineering*. Feature Engineering can often end up being more of an art than a science as a whole lot of intuition is required in order to guess what features the model would need to then linearly separate the modified feature space.

(c) A regression algorithm that, given $\mathbf{x} \in \mathbb{R}^3$ which represents the first three notes, will predict the fourth note in the bar.

2. First we must construct $\hat{\mathbf{X}}$, then we can calculate $J(\mathbf{v})$.

$$J(\mathbf{v}) = \frac{1}{3}\left\| \begin{bmatrix} 1 & 5 & 3 & 1 \\ 1 & 4 & 4 & 2 \\ 1 & 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.6 \\ 7 \\ 3 \end{bmatrix} - \begin{bmatrix} 31 \\ 34 \\ 30 \end{bmatrix} \right\|_2^2$$

$$= \frac{1}{3}\left\| \begin{bmatrix} 31 \\ 34.4 \\ 28.6 \end{bmatrix} - \begin{bmatrix} 31 \\ 34 \\ 30 \end{bmatrix} \right\|_2^2 = \frac{1}{3}\left\| \begin{bmatrix} 0 \\ 0.4 \\ -1.4 \end{bmatrix} \right\|_2^2 = \frac{1}{3}(2.12) \approx 0.70667$$

3. Our new $\mathbf{X}$ is:

$$\mathbf{X} = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_1x_2 & x_1x_3 & x_2x_3 & x_1^2 & x_2^2 & x_3^2 \end{matrix} \\ \begin{bmatrix} 1 & 2 & 3 & 2 & 3 & 6 & 1 & 4 & 9 \\ 2 & 4 & 2 & 8 & 4 & 8 & 4 & 16 & 4 \\ 3 & 3 & 5 & 9 & 15 & 15 & 9 & 9 & 25 \end{bmatrix} \end{array}$$

7. Techniques for Improving Learning Models

7.1 Capacity, Overfitting, Underfitting

Recall, the learning algorithm (Experience, Training, Program): The program is "trained" using Experience, as in $(x^{(train)}, y^{(train)})$. But P is determined by "test" data not found in the Experience, as in $(x^{(test)}, y^{(test)})$.

Usually there will be some "cost function" which can be minimized with respect to the training set in order to determine Program. That is, the "test error" is minimized by minimizing the "training error."

In order to ensure the above process works properly, assumptions need to be made. These will be referred to as the Independent Identically Distributed (IID) assumptions. Both the training set and the test set are gotten by independently (or randomly) sampling the same probability distribution. We will refer to this probability distribution as "Pdata".

Usually our task T is accomplished by choosing some set $\overrightarrow{w}$ of parameters found by minimizing some cost function $J(\overrightarrow{w})$ on the training set.

We measure the performance of the program by measuring our cost over some training set $E^{(train)}$. The process for this follows a general formula:

1. Randomly sample P_{data} to get $E^{(train)}$
2. Find $\mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}}[J(\mathbf{w}; E^{(train)})]$
3. Sample P_{data} to get $E^{(test)}$
4. Compute $J(\mathbf{w}^*; E^{(test)})$

from this, we have that

$$\begin{aligned} E_{P_{data}}[J(\mathbf{w}^*; E^{(test)}] &\geq E_{P_{data}}[\min_{\mathbf{w}} J(\mathbf{w}; E^{(test)})] \\ &= E_{P_{data}}[\min_{\mathbf{w}} J(\mathbf{w}; E^{(train)})] && \text{by IID} \\ &= E_{P_{data}}[J(\mathbf{w}^*; E^{(train)})] \end{aligned}$$

So the value of our test error for a model is expected to be larger than the training error. Thus the performance of our model depends on

1. Minimizing the training error
2. Minimizing the gap between training error and test error

If the model struggles with minimzing the training error, we refer to this as a problem of "underfitting", if the model struggles with minimizing the gap, we refer to this as a problem of "overfitting".

7.1.1 The Capacity

We now will provide an informal sense of the capacity of a learning algorithm

Definition 7.1.1 The capacity of a learning algorithm is the "size" of the space of functions the algorithm can fit. We refer to the space of functions the algorithm can fit as the hypothesis space.

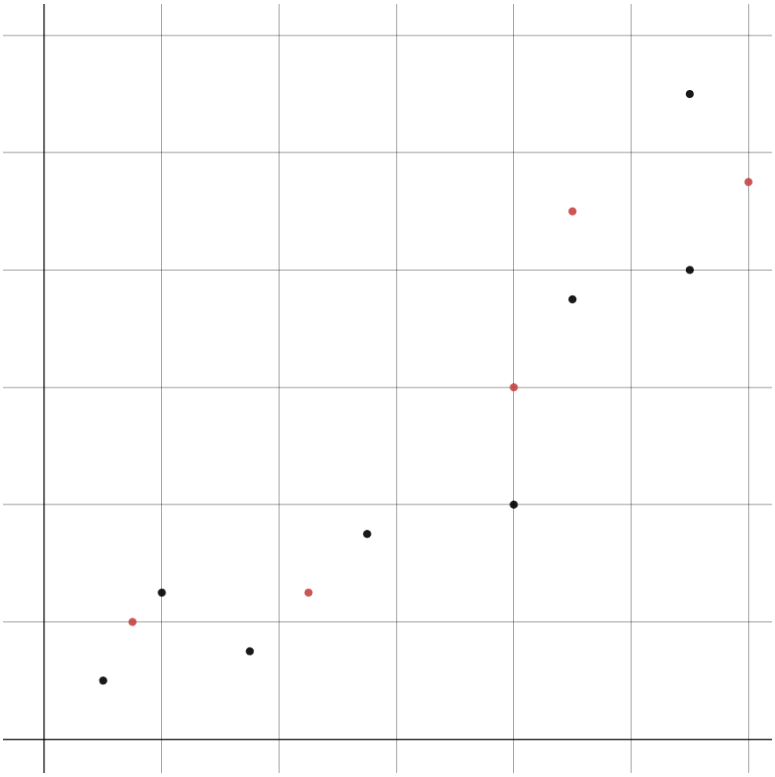
R In order to more formally definite the capacity we need statistical learning theory which is outside the scope of this book.

■ **Example 7.1** Consider linear regression with 1 variable. We have two cases

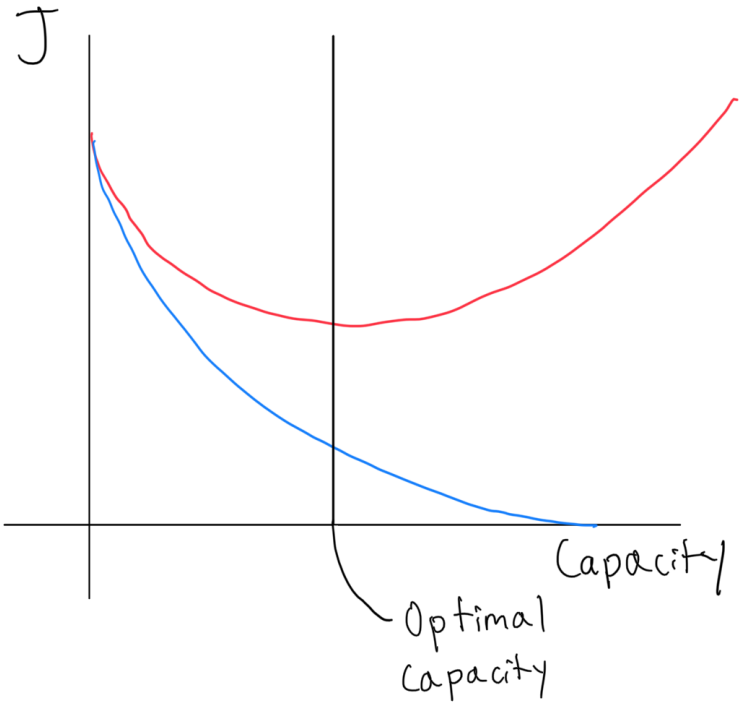
1. There are no polynomial terms. In this case, we predict with $f(x) = wx + b$. We refer to $f(x)$ as the hypothesis function. For this case, the capacity of all such functions had dimension 2.
2. There are polynomial terms. For this specific example consider polynomial terms up to degree 4. We predict with $f(x) = w_4x^4 + w_3x^3 + w_2x^2 + w_1x + b$. Now the capacity of our model is 5.

■

Suppose our dataset looks like



With the black dots being our training data and the red dots being our test data. The graph of training and test error versus capacity will look something like



with the red line representing test error and the blue line representing training error.

As you can see, there is some optimal capacity for the program in order to minimize the test error. We will further develop this "optimal model" in the next section.

7.1.2 Bayes Error

Suppose that we have P_{data}. We want to use P_{data} to make predictions

■ **Example 7.2** $f^B(\mathbf{x}) = \underset{y}{\text{argmax}} P_{data}(y|\mathbf{x})$ (supervised algo) ■

Such a model will provide a theoretical minimum cost for any prediction model.

Definition 7.1.2 We define Bayes Error to be $\min_f E_{P_{data}}[J(f;\mathbf{x},y)]$ where $J(f;\mathbf{x},y)$ is the cost function of prediction for $f(\mathbf{x}) = y$.

Thus the Bayes error is the error from the "optimal model" stated above.

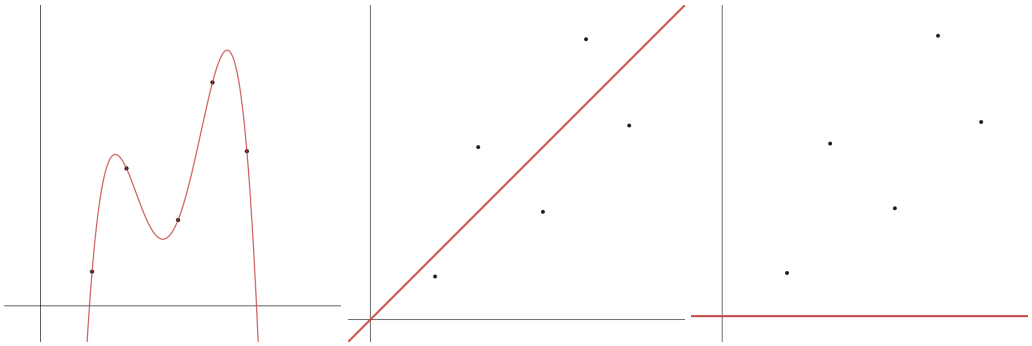
An important note is that Bayes Error can be nonzero. For example, mapping $\mathbf{x}$ to y can be nondeterministic or $\mathbf{x}$ can be mapped to y in a deterministic way but it requires more variables than we have in $\mathbf{X}$

7.1.3 Regularization

The central idea behind regularization is to add a new term with parameter λ to the cost function J to further restrict the hypothesis space.

■ **Example 7.3** We modify $J(\mathbf{w}) = \frac{1}{m}\left\|\hat{\mathbf{X}}\mathbf{w} - y\right\|_2^2 + \lambda \|\mathbf{w}\|_2^2$. Since we minimized J to find $\mathbf{w}^*$, this extra term promotes coefficients of $\mathbf{w}^*$ to be small. λ dictates how strongly this is promoted. ■

■ **Example 7.4** Using polynomial terms up to degree 10



The first graph has $\lambda = 0$ and is overfit. The third graph has a large λ and is underfit. The middle graph has a λ somewhere in between and is a good fit. ■

Using regularization we can start with a high-capacity model and "tune" λ to avoid overfitting or underfitting the data.

7.2 Hyperparameters and Validation

In the last section when we were doing regularization we added a new parameter λ. This is an example of what we call a hyperparamter.

Definition 7.2.1 *Hyperparamters* are parameters which effect/control a learning algorithm's behaviour but are not themselves learned by the algorithm in training.

So in the aforementioned regularization example for linear regression, when we added λ we were still minimizing the cost function over the parameter w and just tuning λ by hand.

■ **Example 7.5** In polynomial regression with regularization, the parameters are

$$n \text{ and } \lambda$$

where n is the maximum degree of the polynomial terms (an explicit control of capacity) and λ is the regularization parameter. ■

7.2.1 A Problem:

Hyperparameters are usually tuned by hand when experimenting in order to minimize the test error. But this creates an issue: the test set is no longer a true test set, because some model parameters are fit to the test set. And the whole point of the teat set is to approximate the generalization error, i.e. how will the model perform on new data we have never seen before.

7.2.2 A Solution and new approach:

Introduce a new test set, which we call the validation set.

Definition 7.2.2 *Training, validation, and test data set method*: Split the data E into 3 classes, $E^{(train)}$, $E^{(val)}$, and $E^{(test)}$. In terms of percentages, as a rule of thumb this is often a $80-10-10$ training-validation-test split.

1. Assuming we are trying to fit some parameters in order to create a prediction model, given hyperparameters λ, we can train a model with $E^{(train)}$ by finding $\mathbf{w}^*_\lambda = \text{argmin}_{\mathbf{w}^*} J(\mathbf{w}^*, \lambda; E^{(train)})$.
2. Evaluate performance on the validation dataset by looking at $J(\mathbf{w}^*_\lambda; E^{(val)})$. Note that J does not include the regularization term.[1]
3. Repeat 1. and 2. with different choices of λ until you find the lowest value of $J(\mathbf{w}^*_{\lambda^*}; E^{(val)})$. Call the resulting optimal hyperparameter values λ^*. This is the hyperparameter tuning phase.
4. Approximate the generalization error by computing $J(\mathbf{w}^*_{\lambda^*}; E^{(test)})$.

Tip: This all works if E is large enough to be split into 3 categories. If not, one possible solution is k-fold cross validation. This allows you to perform validation with your training set. In the python package sklearn, this is done with

```
cross_val_score()
```

7.3 Bias and Variance

Definition 7.3.1 A *point estimator* for a parameter θ associated to some distribution p is any function of random variables $x_1, \ldots, x_n$ (distributed by p). We write

$$\hat{\theta} = y(x_1, \ldots, x_n)$$

The *bias* of $\hat{\theta}$ is

$$bias(\hat{\theta}) = \mathbb{E}\left[\hat{\theta}\right] - \theta$$

The variance of $\hat{\theta}$ is

$$Var(\hat{\theta}) = \mathbb{E}\left[(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right])^2\right]$$

Definition 7.3.2 For a given estimator, if $bias(\hat{\theta}) = 0$ we call $\hat{\theta}$ an *unbiased estimator*.

■ **Example 7.6** Take $X_1, \ldots X_n$ IID random variables distributed by the Bernoulli Distribution:

$$\Omega = 0, 1$$

$$\mathscr{P}(x = 1) = \theta$$

$$\mathscr{P}(x = \chi) = \theta^{\chi}(1-\theta)^{(1-\chi)}$$

We can make a point estimate for θ by

$$\hat{\theta} = \frac{\sum_{i=1}^{n} x_i}{n}$$

[1] This is because regularization is only used during the training phase to restrict the capacity in some way, and would throw off any evaluation of performance once the parameters have already been chosen. That is, the loss function during regularization tries to penalize over/under-fitting, but once we are evaluating on a different dataset, calculating the normal loss is a much better approximation of generalization error.

Hence, our bias will be:

$$bias(\hat{\theta}) = \mathbb{E}\left[\hat{\theta}\right] - \hat{\theta}$$

$$= \mathbb{E}\left[\frac{\sum_{i=1}^{n} x_i}{n}\right] - \hat{\theta}$$

and by linearity of expectation:

$$= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\left[x_i\right] - \theta$$

$$= \frac{1}{n}\sum_{i=1}^{n} \theta - \theta$$

$$= \frac{n\theta}{n} - \theta = 0$$

With our bias calculated, let's now look at the variance:

$$Var(\hat{\theta}) = Var\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)$$

because $\frac{1}{n}$ is a constant, we can "bring" it's square out:

$$= \frac{1}{n^2}\sum_{i=1}^{n} V(x_i)$$

$$= \frac{1}{n^2}\sum_{i=1}^{n} \theta(1-\theta)$$

$$= \frac{\theta(1-\theta)}{n}$$

Notice that

$$\lim_{n\to\infty} \frac{\theta(1-\theta)}{n} = 0$$

Definition 7.3.3 The *mean squared error* (MSE) of $\hat{\theta}$ is

$$MSE(\hat{\theta}) = \mathbb{E}\left[(\hat{\theta} - \theta)^2\right]$$

Lemma 7.3.1

$$MSE(\hat{\theta}) = Var(\hat{\theta}) + Bias(\hat{\theta})^2$$

Proof.

$$MSE(\hat{\theta}) = \mathbb{E}\left[(\hat{\theta} - \theta)^2\right]$$

$$= \mathbb{E}\left[(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right] + \mathbb{E}\left[\hat{\theta}\right] - \theta)^2\right]$$

$$= \mathbb{E}\left[(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right])^2 + (\mathbb{E}\left[\hat{\theta}\right] - \theta)^2 + 2(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right])(\mathbb{E}\left[\hat{\theta}\right] - \theta)\right]$$

$$= \mathbb{E}\left[(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right])^2\right] + \mathbb{E}\left[(\mathbb{E}\left[\hat{\theta}\right] - \theta)^2\right] + 2\mathbb{E}\left[(\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right])(\mathbb{E}\left[\hat{\theta}\right] - \theta)\right]$$

Note that $\mathbb{E}\left[\hat{\theta}\right] - \theta$ is a constant:

$$= Var(\hat{\theta}) + (\mathbb{E}\left[\hat{\theta}\right] - \theta)^2 + 2(\mathbb{E}[\hat{\theta}] - \theta)\mathbb{E}\left[\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right]\right]$$

Notice $\mathbb{E}\left[\hat{\theta} - \mathbb{E}\left[\hat{\theta}\right]\right] = \mathbb{E}\left[\hat{\theta}\right] - \mathbb{E}\left[\hat{\theta}\right] = 0$, so we have:

$$= Var(\theta) + Bias(\hat{\theta})^2$$

Which is exactly what we wanted. ■

■

Finally, it is worth pointing out that, for large data sets, it might take too much computational power to use all of it during the training process. Instead, pick a subset of the data set (mini-batch) during each iteration to approximate.

7.4 Exercises

1. State the capacity for the following models:
   (a) Polynomial regression using terms up to the 8th degree
   (b) Logistic regression using terms up to the 20th degree
   (c) Linear regression using $f(x) = wx + b$
   (d) Polynomial regression using $w_7x^7 + w_4x^4 + w_1x$
2. Prove that training error can be less then Bayes error.
3. Find the Gradient and Hessian for the new cost function after adding the regularization term, $J(\mathbf{w}) = \frac{1}{m}\left\|\hat{\mathbf{X}}\mathbf{w} - y\right\|_2^2 + \lambda \left\|\mathbf{w}\right\|_2^2$
4. Consider a Gaussian distribution $N(x; \mu, \sigma^2)$. We can estimate the mean μ by taking the average of n IID random variables (all from identical Gaussian distributions):

   $$\hat{\mu} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

   Show that $Bias(\hat{\mu}) = 0$.
5. Still considering a Gaussian distribution, estimate σ^2 by

   $$\hat{\sigma^2} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$$

   Show that

   $$Bias(\hat{\sigma^2}) = -\frac{\sigma^2}{n}$$

6. Going back to the point estimate $\hat{\mu}$ for a Gaussian distribution, find the mean squared error of $\hat{\mu}$

7.5 Solutions to Exercises

1. We have that the capacity of each of the models is:
   (a) 9 as there are 8 powers of x and the constant
   (b) 21 as there are 20 powers of x and the constant
   (c) 2 as there is just 1 power of x and the constant
   (d) 3 as there are three terms total
2. Here is a possible proof: Assume that you have a nondeterministic mapping of $\mathbf{x}$ to y. Then Bayes error is nonzero. Consider a training set on this data. We could have a subset that can be deterministically mapped from $\mathbf{x}$ to y. Thus the training error could be 0, while the Bayes error is nonzero. Thus the training error can be less than the Bayes error.
3. In an earlier exercise we showed that the gradient of $\left\|\hat{\mathbf{X}}\mathbf{w}-y\right\|_2^2 = 2(\hat{\mathbf{X}}^\top\hat{\mathbf{X}}\mathbf{w}-\hat{\mathbf{X}}^\top y)$ so we only have to find the gradient of $\lambda\left\|\mathbf{w}\right\|_2^2 = \lambda\frac{\partial}{\partial\mathbf{w}}\mathbf{w}^\top\mathbf{w} = \lambda\mathbf{w}^\top(I+I^\top) = 2\lambda\mathbf{w}^\top$. Combining these gives us that the gradient is $\frac{2}{m}(\hat{\mathbf{X}}^\top\hat{\mathbf{X}}\mathbf{w}-\hat{\mathbf{X}}^\top y)+2\lambda\mathbf{w}^\top$.
   To find the Hessian we take the gradient again. So we get that the Hessian is $\frac{2}{m}\hat{\mathbf{X}}^\top\hat{\mathbf{X}}+2\lambda I^\top = \frac{2}{m}\hat{\mathbf{X}}^\top\hat{\mathbf{X}}+2\lambda$.

4. $$Bias(\hat{\mu}) = \mathbb{E}[\hat{\mu}]-\mu$$
   $$= \mathbb{E}\left[\frac{x_1+\ldots+x_n}{n}\right]-\mu$$
   and then by linearity:
   $$= \frac{\mathbb{E}[x_1]+\ldots+\mathbb{E}[x_n]}{n}-\mu$$
   $$= \frac{\mu+\ldots+\mu}{n}-\mu$$
   $$= \frac{n(\mu)}{n}-\mu = \mu-\mu$$
   $$= 0$$
   So $\hat{\mu}$ is an unbiased estimator.

5. $$Bias(\hat{\sigma^2}) = \mathbb{E}[\hat{\sigma^2}]-\sigma^2$$
   $$= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(x_i-\hat{\mu})^2\right]-\sigma^2$$
   $$= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[x_i^2-2x_i\hat{\mu}-\hat{\mu}^2\right]-\sigma^2$$
   $$= \frac{1}{n}\sum_{i=1}^{n}(\sigma^2+\mu^2))$$

6.

8. Maximum Likelihood Estimation and Naive E

8.1 Maximum Likelihood Estimation

Suppose we have a data-set under **IID** assumptions, $\mathbf{X}$, and we wish to approximate $P_{data}(\mathbf{x})$ to make reasonable predictions with an associated probability. A method for such a process is called Maximum likelihood estimation.

Definition 8.1.1 The Estimator for P_{data} is defined as: $P_{model}(\mathbf{x}, \theta)$ for a parameter, θ.

This definition however is not useful enough to use as we do not know how to find our parameter θ, nor what we are looking for to even optimize.

■ **Example 8.1** If were expectation P_{data} to be of a certain distribution, for example gaussian, then our parameter, θ, could be:

$$\theta = \begin{bmatrix} \mu_0 \\ \Sigma_0 \end{bmatrix}$$

and our associated P_{model}:

$$P_{model}(\mathbf{x}, \theta) = \mathcal{N}(\mathbf{x}\,; \mu_0, \Sigma_0)$$

■

Definition 8.1.2 The *Maximum Likelihood Estimation* for a distribution $P_data(\mathbf{x})$ and parameter θ is given by:

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}}\, P_{model}(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}; \theta)$$

This definition is essentially maximizing all the probabilities for possible points in our data with an associated θ. Note: every θ_{ML} is always defined with a data-set.

■ **Example 8.2** Definition 9.4.2 can be re-written to better reflect this using the properties of independence IID assumptions give us, $P(A,B) = P(A \bigcap B) = P(A) \cdot P(B)$:

$$\begin{aligned}\theta_{ML} &= \underset{\theta}{\text{argmax}}\ P_{model}(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}; \theta) \\ &= \underset{\theta}{\text{argmax}} \prod_{i=1}^{m} P_{model}(\mathbf{x}^{(i)}; \theta) \\ &= \underset{\theta}{\text{argmax}} \sum_{i=1}^{m} \log\left(P_{model}(\mathbf{x}^{(i)}; \theta)\right)\end{aligned}$$

■

And the specific reason we choose log here is that products are prone to underflow with $p_i \in [0,1]$, so the log eliminates this issue and does not affect θ_{ML} as it is one-to-one for positive real numbers.This P_{model} is from an unsupervised model, but can be extended to supervised models with a joint distribution.

Definition 8.1.3 Given a labeled data-set under **IID** assumptions with corresponding $\mathbf{X}, \mathbf{y}$, sampled from a joint distribution $P_{data}(\mathbf{x}, \mathbf{y})$ then $P_{model}(\mathbf{x}, \mathbf{y}; \theta)$ is an approximation for this data-set assuming $P_{data} \in \{P_{model}(\theta) | \theta \in \mathbb{R}^n\}$. Then the maximum likelihood estimation is:

$$\begin{aligned}\theta_{ML} &= \underset{\theta}{\text{argmax}}\ P_{model}(\mathbf{y}|\mathbf{X}; \theta) \\ &= \underset{\theta}{\text{argmax}} \prod_{i=1}^{m} P_{model}(y_i|\mathbf{x}^{(i)}; \theta) \\ &= \underset{\theta}{\text{argmax}} \sum_{i=1}^{m} \log\left(P_{model}(y_i|\mathbf{x}^{(i)}; \theta)\right)\end{aligned}$$

This is a very useful definition as it enables us to give a prediction given a new example $(\mathbf{x}, y)$ we can estimate y by defining:

$$\hat{y} = \underset{y}{\text{argmax}}\ P_{model}(y|\mathbf{x}; \theta_{ML})$$

and this becomes very useful when looking at some practical examples of this.

■ **Example 8.3 Linear Regression** (idea $\hat{y} = \begin{bmatrix}1\\ \mathbf{x}\end{bmatrix}^\top \theta$)

Assume $P_{model}(y|\mathbf{x}; \theta)$ is Gaussian with mean: $\mu = \begin{bmatrix}1\\ \mathbf{x}\end{bmatrix}^\top \theta$). So,

$$\begin{aligned}P_{model}(y|\mathbf{x}; \theta) &= \mathcal{N}(\mathbf{y}; \begin{bmatrix}1\\ \mathbf{x}\end{bmatrix}^\top \theta, \sigma^2) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left(\mathbf{y} - \begin{bmatrix}1\\ \mathbf{x}\end{bmatrix}^\top \theta\right)\right)^2\end{aligned}$$

and given a data-set $\mathbf{X}, \mathbf{y}$ sampled from P_{data},

$$\begin{aligned}
\theta_{ML} &= \underset{\theta}{\operatorname{argmax}}\ P_{model}(y|\mathbf{x};\theta) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} \left(\mathbf{y}_i - \begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix}^\top \theta \right) \right)^2 \right] \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} -\frac{1}{2} \log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2} \left(\mathbf{y}_i - \begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix}^\top \theta \right)^2 \\
&= -\underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \left(\mathbf{y}_i - \begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix}^\top \theta \right)^2 \\
&= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{m} \left(\mathbf{y}_i - \begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix}^\top \theta \right)^2 \\
&= \underset{\theta}{\operatorname{argmin}}\ J(\mathbf{X}, \mathbf{y}; \theta)
\end{aligned}$$

From this we see that finding the optimal least squares cost is equivalent to finding θ_{ML}. Our prediction for $\hat{y}$ can be found:

$$\begin{aligned}
\hat{y} &= \underset{y}{\operatorname{argmax}}\ P_{model}(y|\mathbf{x};\theta_{ML}) \\
&= \underset{y}{\operatorname{argmax}}\ \mathcal{N}(\mathbf{y}; \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}^\top \theta_{ML}, \sigma^2) \\
&= \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}^\top \theta_{ML}
\end{aligned}$$

As for a Gaussian distribution the maximum probability is at the mean, so our prediction $\hat{y}$ given θ_{ML} is the same prediction that the argmin over v of the cost function gives us, so we can then conclude that Linear Regression is a maximum likelihood estimator. ∎

So it seems most of our machine learning algorithms we have encountered so far have been maximum likelihood estimators. To see this further, we return to discrete classification.

∎ Example 8.4 Binary Classification

Given a labeled data-set, $\mathbf{X}, \mathbf{y}$, where $\mathbf{X}, \mathbf{y}$ are sampled from $P_{data}(\mathbf{x}, y)$ and y is 0 or 1 always. This gives us the very useful property that:

$$P_{data}(y = 1|\mathbf{x}) = 1 - P_{data}(y = 0|\mathbf{x})$$

Consider a model, $P_{model}(y = 1|\mathbf{x};\theta) = \sigma\left(\begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}^\top \theta \right)$ and $P_{model}(y = 0|\mathbf{x};\theta) = 1 - \sigma\left(\begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}^\top \theta \right)$

where $\sigma(\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{x})}$. Computing the maximum likelihood,

$$\begin{aligned}
\theta_{ML} &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log\left(P_{model}(y = y_i|\mathbf{x}^{(i)};\theta)\right) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} y_i \log\left[\sigma\left(\begin{bmatrix}1\\ \mathbf{x}^{(i)}\end{bmatrix}^{\top}\theta\right)\right] + (1-y_i)\log\left[1-\sigma\left(\begin{bmatrix}1\\ \mathbf{x}^{(i)}\end{bmatrix}^{\top}\theta\right)\right] \\
&= \underset{\theta}{\operatorname{argmin}}\ J(\mathbf{X},\mathbf{y};\theta)
\end{aligned}$$

So again we see we recovered another cost function as we did in **Example 8.3**, so we conclude that logistic regression is a maximum likelihood estimator. ∎

8.2 Bayes Estimators

Ending our discussion of MLE's we return to bias and variance.

Definition 8.2.1 For any supervised model where $\mathbf{X},\mathbf{y}$ are sampled from P_{data} under **IID** assumptions, we define the *Bayes prediction*:

$$f^B(\mathbf{x}) = \underset{y}{\operatorname{argmax}}\ P_{data}(y|\mathbf{x})$$

Here f^B is the theoretical best model as stated before in **7.1.2**.

Definition 8.2.2 Define a random variable, ε, under the assumption that ε is normally distributed with $\mu = 0$ so $\mathbb{E}_{P_{data}}[\varepsilon] = 0$, and ε is independent of $\mathbf{x},\mathbf{y}$. We define the *Noise* as:

$$\varepsilon = y - f^B(\mathbf{x})$$

We define the noise this way such that the error is distributed everywhere and not clustered randomly. Suppose we have a model, $\hat{y}$ to predict y. We can then compute the $MSE(\hat{y})$ to find the error in predicting y:

$$\begin{aligned}
MSE(\hat{y}) &= \mathbb{E}_{P_{data}}\left[(y-\hat{y})^2\right] \\
&= \mathbb{E}_{P_{data}}\left[(y - f^B(\mathbf{x}) + f^B(\mathbf{x}) - \hat{y})^2\right] \\
&\quad \text{Our goal is to have control over our model and find a separate term of solely noise...} \\
&= \mathbb{E}_{P_{data}}\left[(y-f^B(\mathbf{x}))^2\right] + \mathbb{E}_{P_{data}}\left[(f^B-\hat{y})^2\right] + 2\mathbb{E}_{P_{data}}\left[(y-f^B)(f^B-\hat{y})\right] \\
&= \mathbb{E}_{P_{data}}\left[(y-f^B(\mathbf{x}))^2\right] + \mathbb{E}_{P_{data}}\left[(f^B)-\hat{y})^2\right] + \dots \\
&\dots + 2(\mathbb{E}_{P_{data}}[yf^B(\mathbf{x})] - \mathbb{E}_{P_{data}}[f^B(\mathbf{x})^2] - \mathbb{E}_{P_{data}}[y\hat{y}] + \mathbb{E}_{P_{data}}[f^B(\mathbf{x})\hat{y}]) \\
&= \mathbb{E}_{P_{data}}\left[(y-f^B(\mathbf{x}))^2\right] + \mathbb{E}_{P_{data}}\left[(f^B(\mathbf{x})-\hat{y})^2\right] + 0 \text{ by our def. of noise} \\
&= \mathbb{E}_{P_{data}}\left[(y-f^B(\mathbf{x}))^2\right] + \mathbb{E}_{P_{data}}\left[(f^B(\mathbf{x})-\hat{y})^2\right]
\end{aligned}$$

We can then observe that the first term is simply Bayes error as defined in chapter 7. However, we have control over the second term, $\mathbb{E}_{P_{data}}\left[(f^B(\mathbf{x})-\hat{y})^2\right]$ so we will proceed with this term:

$$\mathbb{E}_{P_{data}}\left[(f^B(\mathbf{x})-\hat{y})^2\right] = \dots = (f^B(\mathbf{x}) - \mathbb{E}_{P_{data}}[\hat{y}])^2 + \mathbb{E}_{P_{data}}[(\hat{y} - \mathbb{E}_{P_{data}}[\hat{y}])^2]$$

from which we can see the first term is the bias in predicting $f^B(\mathbf{x})$ and the second term is the variance in $\hat{y}$. Thus the $MSE(\hat{y})$ in predicting y is:

$$MSE(\hat{y}) = \mathbb{E}_{P_{data}}[\varepsilon^2] + (f^B(\mathbf{x}) - \mathbb{E}_{P_{data}}[\hat{y}])^2 + \mathbb{E}_{P_{data}}[(\hat{y} - \mathbb{E}_{P_{data}}[\hat{y}])^2]$$

or more specifically the expectation of the squared noise, the bias in predicting $f^B(\mathbf{x})$, and the variance in $\hat{y}$.

8.3 Exercises

1. Find the maximum likelihood probability of a coin flip with 5 trials and 3 successes. Recall: A coin flip is modeled by a Bernoulli distribution is given by: $P(n \text{ heads}|\theta) = \frac{k!}{n!(k-n)!}\theta^n(1-\theta)^{k-n}$, where k is trials. Here, $k = 5,\ n = 3$

2. From the previous example, does this result reflect the data or the event? Is there a way to prevent this? Explain.

3. Can an MLE do this for any probability distribution? Why or why not? Give an example of a limitation and how this can be used in machine learning.

8.4 Solutions to Exercises

1. Answer: Since we are trying to find the argmax, we take the derivative with respect to our parameter, θ and set it equal to 0 to find an extremum,

$$\begin{aligned}
0 &= \frac{d}{d\theta}P(3|\theta) \\
&= \frac{d}{d\theta}\alpha\theta^3(1-\theta)^2, \alpha = \frac{5!}{3!(2)!} \\
&= 3\alpha\theta^2(1-\theta)^2 - 2\alpha(1-\theta)\theta^3 \\
2\alpha(1-\theta)\theta^3 &= 3\alpha\theta^2(1-\theta)^2 \\
2\theta(1-\theta) &= 3(1-\theta)^2 \\
2\theta &= 3(1-\theta) \\
2\theta &= 3 - 3\theta \\
\implies \theta &= \frac{3}{5}
\end{aligned}$$

   So our θ_{ML} is $3/5$. (The actual probability is $\frac{1}{2}$

2. Answer: Possible example, The maximum likelihood estimation assumes that our data is randomly sampled. Because it is random, at a low to medium sample size it may over-fit and provide a value that is not accurate to the event, such as θ_{ML} giving a result that is 10% off from the event. So for the best results, we must have a large $\mathbf{X}$ to get a good result.

3. Answer: Possible answer, Yes. With enough data(arbitrarily large amount of samples) as stated in the previous problem , one can perfectly find any distribution. However, a large

amount will suffice. A limitation of this is wide spread out data for which you don't know which kind of distribution it might fit, which is a greatly helped by IID assumptions. It can be used in machine learning as it can be used to generate new data points from some well defined P_{data} to grow a data-set with new examples.

9. Unsupervised Learning and Clustering

9.1 k-means Clustering

Suppose we have a data set $\mathbf{X} \in \mathbb{R}^{m \times n}$, where as usual each data point corresponds to $\mathbf{X}_{i,*} \in \mathbb{R}^n$, and that under reasonable conjecturing or experimentation we determine the data might be separated in groups. One such algorithm would be k-means clustering.

First , we need to initialize some parameters. Pick k, the number of clusters in which we desire to separate the data points. Choose $M = \{\mu_1, \mu_2, \cdots, \mu_k\} \subset \mathbb{R}^n$ to be arbitrary points in n-space. Define $\pi \in \mathbb{R}^m$ such that $\pi_i = j$ if $\mathbf{X}_{i,*}$ is the closest to μ_j.

The cost function that this learning algorithm will minimize is

$$J(M) = \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{X}_{i,*} - \mu_{\pi_i}\|_2$$

Finally initialize a counter variable $t = 0$, a list *costs* with the initial cost $J(M)$, and a stopping condition $\varepsilon > 0$. Then, an implementation in pseudo-code, provided below:

Algorithm 1 k-means Clustering

while True **do**
  if $t > 1$ and $|costs[t] - costs[t-1]| < \varepsilon$ **then**
    break
  end if
  for i in range(k) **do**
    $m_i = \{\#j \quad st. \quad \pi_j = i\}$
    $\mu_i = \frac{1}{m_i} \sum_{j:\pi_j = i} \mathbf{X}_{j,*}$
  end for
  Compute new values for π, J
  $t += 1$
  Append J to *costs*
end while

Without loss of generality, we will assume that the labels correspond to the first k positive integers (they need not to be, but we can always label them as such.)

First, for each label i, we want to consider all the data points that have i as a label with the current vector π, and compute the center of those points and set that new point to μ_i.

Doing so for all labels, we will obtain a new μ, for which we recalculate the vector π, and with that a new cost function value J.

Then, we repeat this process until the cost function value decreases less than our designated function value (this is determined by the size of the ε constant that we choose at the start.)

■ **Example 9.1** Consider the following simple case with the following data and $k = 3$.

The data has been marked with red, while the μ's have been marked with blue. The black line segments connecting the red points with the blue points represent the values of our vector μ (9.1).

Since the data selected exhibits very clear clustering, the algorithm terminates after running a single iteration (10.2).

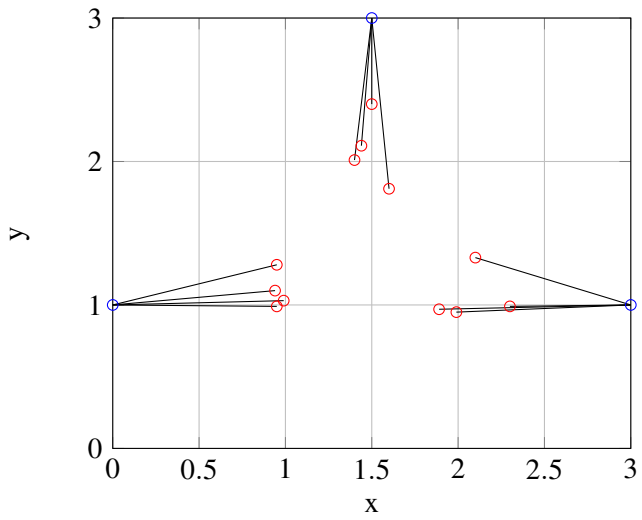


Figure 9.1: Initial μ and π

■

■ **Example 9.2** Given a data set of matrices in $\mathbb{R}^{8\times 8}$ representing an 8-by-8 black and white pixel image of handwritten digits, we would like to run 10-means clustering, in hope that each digit is reasonably clustered (i.e. two digits are close if they are the same, and far apart if they are different.)

Then, after obtaining a value for μ, we can run predictions on unlabeled data to predict the digit that it corresponds to it with the center closest to it. ■

9.2 Exercises

1. Suppose a streaming service uses cluster analysis to classify users in order to better tailor ads. Explain why this model would struggle to classify a shared account.
2. Under what circumstances would a non-convex set would not be classifiable with k-means clustering?

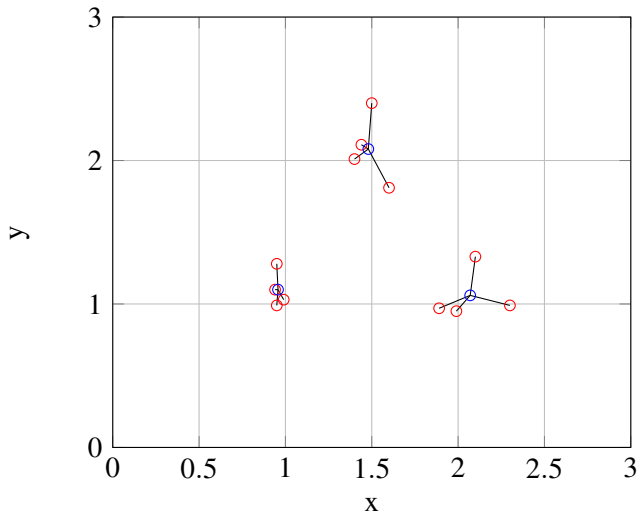
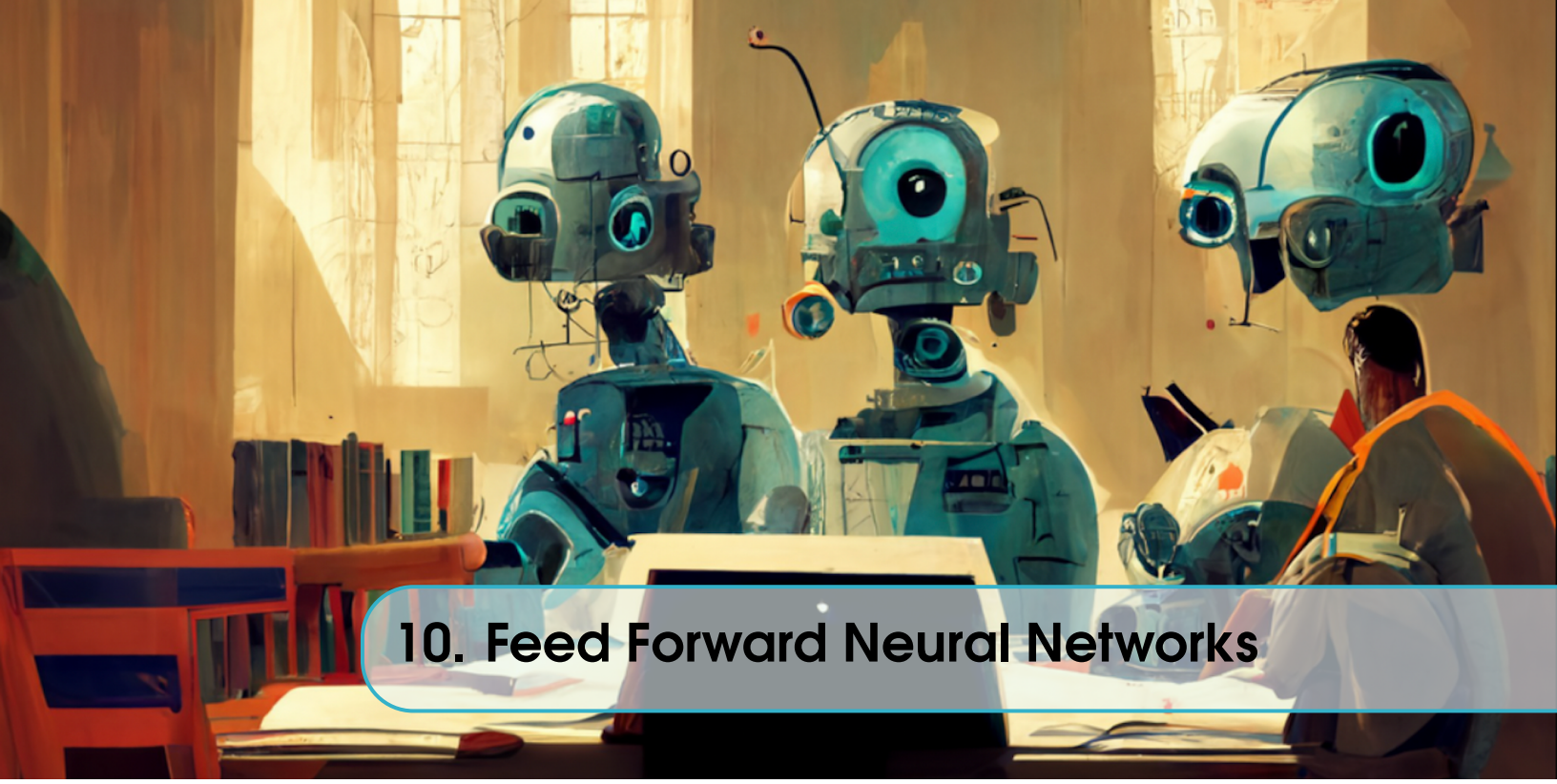


Figure 9.2: μ and π after one iteration

3. Although it was suggested that M could be chosen arbitrarily, give an example where a bad initialization of M would lead to the algorithm not producing reasonable clusters.

9.3 Solution To Exercises

1. A possible explanation might be that the metrics used are too irregular if there distinct users have sharp differences on their preferences.
2. Non-convex sets are not captured precisely if distinct groups are too close, since k-means clusters data into spheres. However if the sets are far apart from each other, it is possible for the algorithm to classify data properly.
3. If the set M is initialized with very similar centers, then the algorithm might converge into overlapping spheres.



10. Feed Forward Neural Networks

The goal of the following chapters is to introduce different types of neural network. To do so, we first introduce some definitions needed.

10.1 Abstract Neurons

Definition 10.1.1 An *abstract neuron*, or simply *neuron*, is a tuple $(\mathbf{x}, \mathbf{w}, \phi, y)$ where

- $\mathbf{x}^\top = (x_0, \cdots, x_n)$ is the *input vector*, usually $x_0 = -1$
- $\mathbf{w}^\top = (w_0, \cdots, w_n)$ is the *weight vector*, with $w_0 = b$, the bias.
- $\phi : \mathbb{R} \to \mathbb{R}$ is the *activation function* of the neuron.
- $y = \phi(\mathbf{x}^\top \mathbf{w})$ is the output of the neuron

The goal of the neuron is to take a target z, and "learn" the target by tunning $\mathbf{w}$ such that $\phi(\mathbf{x}^\top \mathbf{w}) \approx z$

Definition 10.1.2 A *perceptron* is a neuron with input $\mathbf{x} \in \{0,1\}^n$ and activation function

$$\phi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

ϕ is called the *Heaviside* step function

■ **Example 10.1** Let $f^* : \{0,1\}^2 \to \{0,1\}$ be $f^*(x,y) = x$ **and** y.
A solution using the Heaviside step function uses $\mathbf{w}^\top = (1.5, 1, 1)$. Then $\mathbf{x}^\top \mathbf{w} = -1.5 + x + y$, or equivalently, we have the line $y = 1.5 - x$ ■

■ **Example 10.2** Let $f^* : \{0,1\}^2 \to \{0,1\}$ be $f^*(x,y) = x$ **or** y.
A solution using the Heaviside step function uses $\mathbf{w}^\top = (0.5, 1, 1)$. Then $\mathbf{x}^\top \mathbf{w} = -0.5 + x + y$, or equivalently, we have the line $y = 0.5 - x$ ■

The examples above illustrate that if the points cannot be divided by a line (or hyperplane in higher dimensions) then we cannot learn the function using a single neuron. We will fix this later by

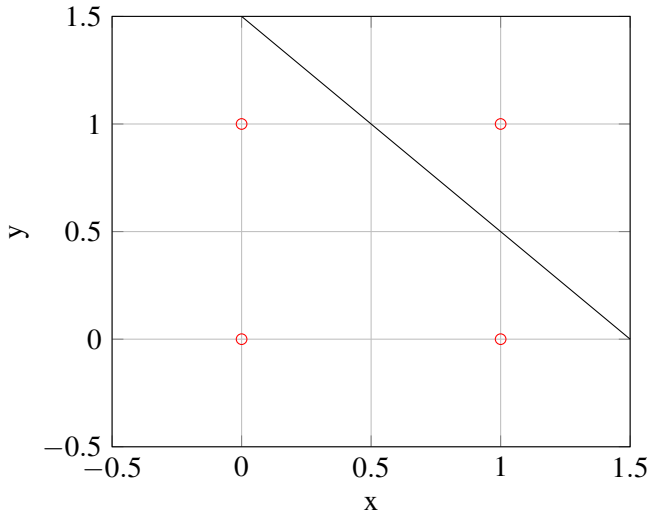


Figure 10.1: Outputs of **and** operator separated by $y = 1.5 - x$

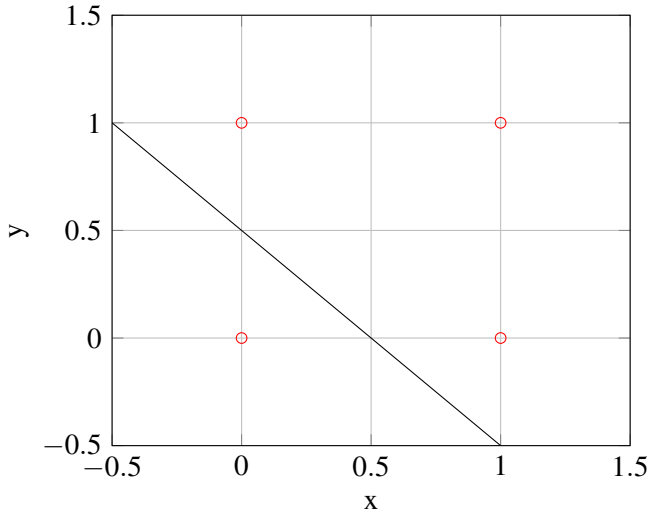


Figure 10.2: Outputs of **or** operator separated by $y = 0.5 - x$

using multiple neurons.

10.2 The Sigmoid Neuron

To better predicting the value of binary variable y, we need to make sure there is a strong gradient. So we can use sigmoid output.

Definition 10.2.1 A neuron with sigmoid activation σ
$\sigma(w^T \mathbf{x})$ is usually interpreted as the probability that target z = 1.

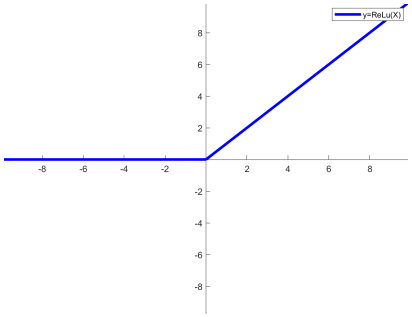
Given a supervised dataset X,$\vec{\mathbf{y}}$, for $y_i \in 0{,}1$ and $\forall$ i sampled from $P_{data}(\vec{x},\mathbf{y})$, we can try to learn target y via prediction.

$$\text{y} = 1 \text{ if } \sigma(w^T \mathbf{x}) > 0.5$$
$$\text{y} = 0 \text{ if } \sigma(w^T \mathbf{x}) < 0.5$$

Linear Neurons: Neurons with activation φ(x) = x

Definition 10.2.2 ReLU(rectified linear unit)
Activation function

$$ReLU(X) = xH(x) = \begin{cases} 1 & x < 0, \\ 0 & x > 0. \end{cases}$$

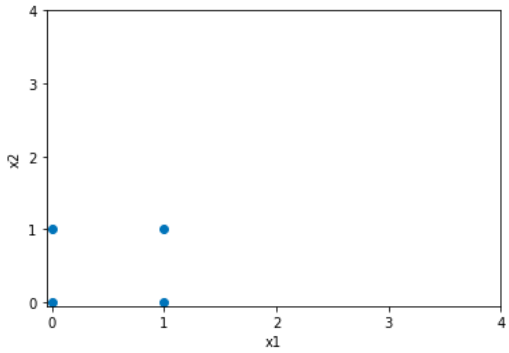


■ **Example 10.3** A Network of Neurons:

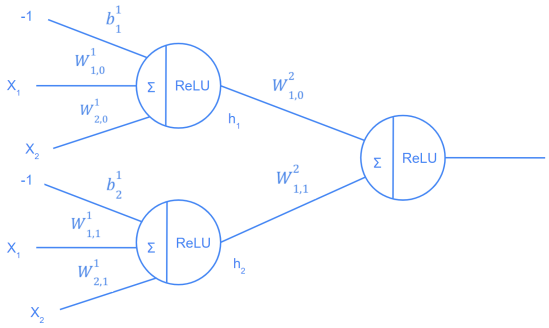
$$f(\vec{x}) = \vec{x_1} \text{ x or } \vec{x_2}, \text{ for } \vec{x} \in \{0,1\}^2$$

$$\text{dataset X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \vec{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

we can draw points in X in the folowing figure



Consider a modle f($\vec{x}$,$\vec{\theta}$) and cost function J(θ) = $\frac{1}{4}\sum(f^*(\vec{x}) - f(\vec{x};\theta))$



Idea: In x_1, x_2 - plane, the points are not separable by a line but maybe by transforing $x_1, x_2 \to h_1, h_2$, the points can be separable by a line in the h_1, h_2 - plane.

$$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = \text{ReLU}(W^{(1)}\vec{x} + \mathbf{b}^{(1)})$$

$$\text{Try } W^{(1)} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \vec{\mathbf{b}}^{(1)} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

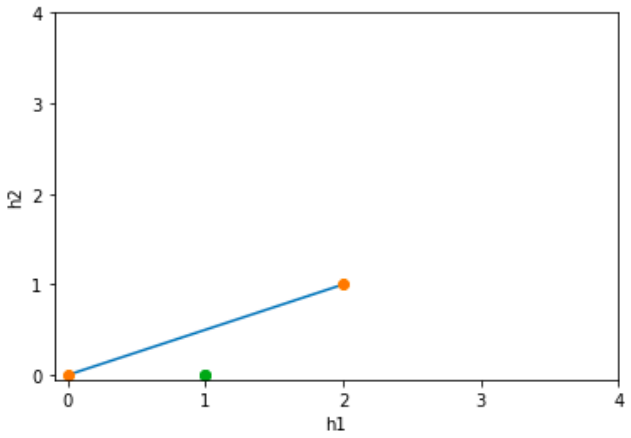
Apply to whole dataset.

$$ReLU(XW^{(1)} + \vec{1}\mathbf{b}^{(1)T}) = ReLU\left(\begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{pmatrix}\right)$$

$$= ReLU\left(\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{pmatrix}\right)$$

$$= ReLU\begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$



So we can divide the point by y = $\frac{x}{2}$

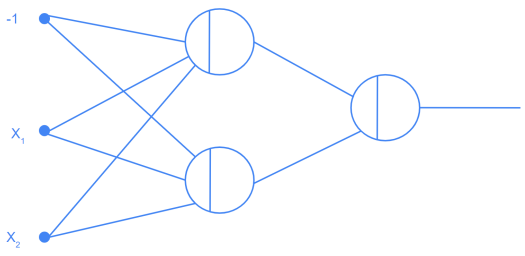
Final output is $W^{(2)} \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} + \mathbf{b}^{(2)}$. Let $W^{(2)} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$, $b^{(2)} = 0$

$$ReLU \left(\begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \end{pmatrix} \right)$$

$$= ReLU \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

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

Better way to draw this:



■

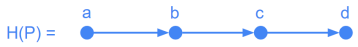
Definition 10.2.3 A partially ordered set(poset) is a pair(P, $\leq$) where P is a set and $\leq$ is a relation on P s.t.
1. x $\leq$ x, $\forall$ x $\in$ P
2. if x $\leq$ y and y $\leq$ x then x = y $\forall$ x, y $\in$ P
3.if x *leq* y and y $\leq$ z then x $\leq$ z $\forall$ x, y, z $\in$ P

Notation: Write x < y if x $\leq$ y and x $\neq$ y. Call (P, $\leq$) a total order if $\forall$ x, y $\in$ P x $\leq$ y or y $\leq$ x.

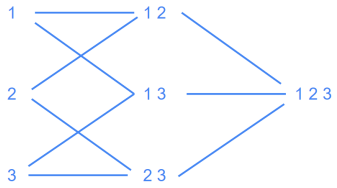
Definition 10.2.4 A cover relation is a pair (x, y) s.t. x $\lneq$ y and $\nexists$ z with x < z < y.

Definition 10.2.5 The Hasse diagram of a finite poset (P, $\leq$) consists of a vertex for each x $\in$ P and a directed edge from x to y if and only if x $\in$ y draw from left to right. Write this as H(p).

■ **Example 10.4** 1.P = {a, b, c, d} a $\lneq$ b $\lneq$ c $\lneq$ d



2. P = {S $\subseteq$ {1,2,3} | S $\neq$ $\emptyset$ } ordered by containment.

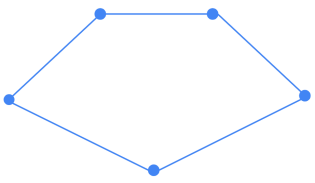


■

Definition 10.2.6 A poset is graded if $\exists$ P: P $\to$ $\mathbb{N}$ (rank function) such that
1. x < y $\implies$ ρ(x) < ρ(y), $\forall$ x, y $\in$ P
2. x $\leftarrow$ y $\implies$ ρ(x) + 1 = ρ(y), $\forall$ x, y $\in$ P

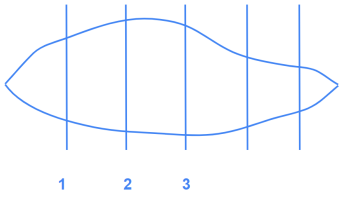
■ **Example 10.5** {S $\subseteq$ 1,2,3 | S $\neq$ $\emptyset$ } is graded
i.e. in a graded poset there is a notiton of "depth" ■

Following is an example of not graded

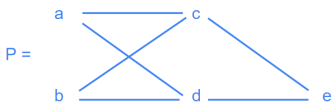


The rank decomposition of a graded poset (P, $\leq$, ρ) is

$$P = \oplus Pi, Pi = \{x \in P | \rho(x) = i\}$$



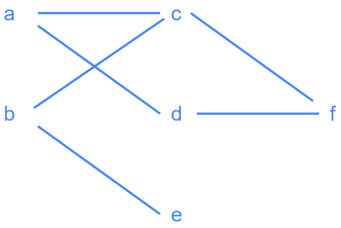
A linear extension of a poset (P, $\leq$) is a total ordering $\leq'$ on P s.t. x $\leq'$ y, for any x, y $\in$ P.
e.g
.



b < a < c < d < e

Lemma 10.2.1 A linear extension of a graded poset is determined by a choice of total ordering of each of its ranks. So there are $\prod_{i\in\mathbb{N}} |P_i|$

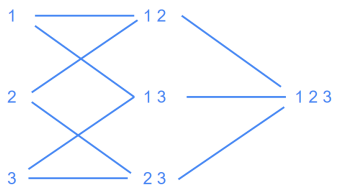
Convention Drawing a Hasse diagram determines a linear extension by ordering downwards along ranks: $a < b < c < d < e < f$.



So Hasse diagram contains all information.

Definition 10.2.7 Let $N = (H, F, G)$ be a neural network with depth D and architecture H = (P, $\leq$, ρ, $\leq'$).
The feedforward of $\mathbf{x} \in \mathbb{R}^{dim(V)_p^D}$ is:
$N\mathbf{x} = G_D F_{D-1} ... G_2 F_1 G_1 F_0 \mathbf{x} \in \mathbb{R}^{dim(V)_p^D}$

■ **Example 10.6** Consider N to be the following:



In this example,

$$\mathbf{b}^0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

and $\mathbf{b}^1 = 0$ We have

$$N \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = G^2 F^1 G^1 F^0 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

So

$$F^0\mathbf{x} = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 0 & 3 \\ 0 & -2 & -3 \end{pmatrix} \mathbf{x} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

and

$$F^1\mathbf{x} = \begin{pmatrix} 1 & -2 & 1 \end{pmatrix}$$

So now we get:

$$N \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \sigma F^1 ReLU F^0 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \sigma F^1 ReLU \begin{pmatrix} 1 \\ 6 \\ -4 \end{pmatrix} = \sigma F^1 \begin{pmatrix} 1 \\ 6 \\ 0 \end{pmatrix} = \sigma(-1\phi) = \frac{1}{1+e^{11}}$$

which is approximately equal to 0.0001.

■

■ **Example 10.7** Suppose that N has some depth D and $G^1, ..., G^D$ are all linear activations ($G^i\mathbf{x} = \mathbf{x}$ $\forall i$). Then $N\mathbf{x} = F^{D-1}F^{D-2}...F^1F^0\mathbf{x}$.

R The composition of affine maps is an affine map, so $\exists W, \mathbf{b}$ such that $N\mathbf{x} = W\mathbf{x} + \mathbf{b}$. So this Neural network can only fit affine functions, in other words, this is essentially linear regression.

■

10.3 Neural Networks as Learning Algorithms

Suppose there is a function $f^* : \mathbb{R}^n \mapsto \mathbb{R}^k$ (often k = 1) and we want to approximate or "model" f^*. Let N be a neural network $N = (H, F, G)$ with depth D and $H = (P, \leq, \rho, \leq')$ such that $\#P^0 = n$ and $\#P^D = k$.

Then we get $F^i\mathbf{x} = W^i\mathbf{x} + \mathbf{b}^i$, where nonzero entries of W^i, and $\mathbf{b}^i$ are treated as parameters.

Let $W = (W^0, ..., W^{D-1})$ and $B = (\mathbf{b}^0, ..., \mathbf{b}^{D-1})$ and define $f_{model}(\mathbf{x}; W, B) = N\mathbf{x}$. (feedforward $\mathbf{x}$).
Now how do we find the appropriate values of W,B?
Given a dataset $X \in \mathbb{R}^{mxn}$ and $Y \in \mathbb{R}^{mxk}$, we define a cost function:

$J(W,B) = \frac{1}{m}\sum_{i=1}^{m} L(f_{model}(\mathbf{x}^i; W, B), \mathbf{y}^i)$.

and try to find values for W,B which minimize this cost function using some optimization algorithm.

R We will usually need to add a regularization term, which will be mentioned later in this chapter.

A common choice for L is:
$L(f_{model}(\mathbf{x};W,B),\mathbf{y}) = \|f_{model}(\mathbf{x};W,B)-\mathbf{y}\|_2^2$

10.4 Neural Networks as a maximum likelihood estimator

Lets assume X and Y are sampled from a distribution $p_{data}(\mathbf{x},\mathbf{y})$ under I.I.D. assumptions. Now let $p_{model}(\mathbf{y}|\mathbf{x};\theta)$ be a model for $p_{data}(\mathbf{y}|\mathbf{x})$, which has been parameterized by $\theta \in \mathbb{R}^N$ for some N. Then $\theta_{ML} = \underset{\theta}{\text{argmin}}(-\sum_{i=1}^m log(p_{model}(\mathbf{y}^i|\mathbf{x}^i;\theta)))$.

■ **Example 10.8** Suppose we choose P_{model} to be Gaussian:
$P_{model}(\mathbf{y}|\mathbf{x};W,B) = \mathrm{N}(\mathbf{y};N\mathbf{x},\sum)$ where N is the normal distribution and N is the neural network.

Then $P_{model}(\mathbf{y}|\mathbf{x};W,B) = \mathrm{N}(\mathbf{y};N\mathbf{x},\sum) = \frac{1}{\sqrt{2\pi^n det\sum}} exp(-\frac{1}{2}(\mathbf{y}-N\mathbf{x})\sum^{-1}(\mathbf{y}-N\mathbf{x}))$ where
$N = (H,F,G)$,
$F_{\mathbf{x}}^i = W^i\mathbf{x}+\mathbf{b}^i$,
$W = (W^0,...,W^{D-1})$, and
$B = (\mathbf{b}^0,...,\mathbf{b}^{D-1})$.

Then finding $\theta_{ML} = (W,B)_{ML}$ is equivalent to minimizing our cost function of the form:
$J(W,B) = -\sum_{i=1}^m log(\frac{1}{\sqrt{2\pi^n det\sum}} exp(-\frac{1}{2}(\mathbf{y}^i-N\mathbf{x}^i)\sum^{-1}(\mathbf{y}^i-N\mathbf{x}^0)))$
$= \frac{1}{m}\sum -log(\frac{1}{constant} + \frac{1}{2}(\mathbf{y}^i-N\mathbf{x}^i)\sum^{-1}(\mathbf{y}^i-N\mathbf{x}^i)$, which $\frac{1}{constant}$ does not affect argmin, so we can ignore this value.
Then:
$J(W,B) = \frac{1}{m}\sum_{i=1}^m \frac{1}{2}(\mathbf{y}^i-N\mathbf{x}^i)\sum^{-1}(\mathbf{y}^i-N\mathbf{x}^i)$.

R We can usually take $\sum = id$ by first normalizing the dataset if needed.

which then we get:
$J(W,B) = \frac{1}{m}\sum_{i=1}^m \left\|\mathbf{y}^i-N\mathbf{x}^i\right\|_2^2$.

In the end, we cycle back to our previous cost function.

R So we can either specify a cost or specify a model for p_{data} and can usually go back and forth between those.

■

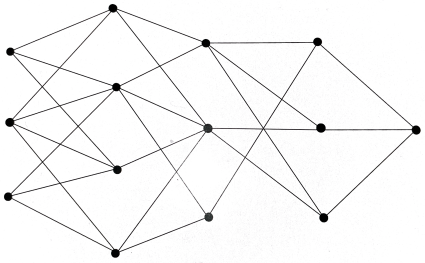
10.5 Exercises

1. Given X,**y**, using sigmoid, ReLU, then sigmoid function to rectified linear unit.

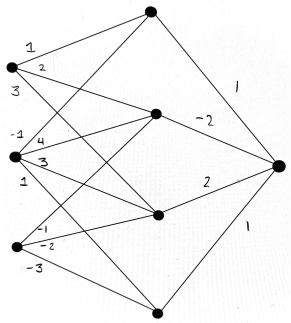
$$X = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & -3 & 2 \\ 2 & -2 & 2 \end{pmatrix}, \mathbf{b}^{(1)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \mathbf{b}^{(2)} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \mathbf{w}^{(1)} = \begin{pmatrix} 0.5 & 1 & 2 \\ 1 & 2 & 0.5 \\ 2 & 0.5 & 1 \end{pmatrix}$$

$$w^{(2)} = \begin{pmatrix} 1 & 2 & 0.5 \\ 1 & 2 & 0.5 \\ 1 & 2 & 0.5 \end{pmatrix} \mathbf{w}^{(3)} = \begin{pmatrix} 1 \\ 0.5 \\ 2 \end{pmatrix}, \mathbf{b}^{(3)} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

2. Draw a linear extension of poset (P, $\leq$), a < b < c < d < e < f
3. Explain why we use sigmoid activation function to predict value of binary variable y.
4. Determine the weight matrices $W^{(0)}, W^{(1)}, W^{(3)}, W^{(4)}$ for the following neural network if all connection weights are worth 1.



5. Find the output of the following neural network for $\mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$, $\mathbf{b}^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$ $\mathbf{b}^{(1)} = (0)$



10.6 Solutions to Exercises

1. Start with plugging in the given data matrix in to the sigmoid equation:

$$sigmoid(XW^{(1)} + \vec{1}\mathbf{b}^{(1)T}) = sigmoid\left(\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & -3 & 2 \\ 2 & -2 & 2 \end{pmatrix}\begin{pmatrix} 0.5 & 1 & 2 \\ 1 & 2 & 0.5 \\ 2 & 0.5 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}\right)$$

$$= sigmoid\begin{pmatrix} 5 & 4.5 & 6.5 \\ 2 & 2 & 1.5 \\ 2 & -5 & 1.5 \\ 4 & -1 & 6 \end{pmatrix} = \begin{pmatrix} 0.73 & 0.99 & 1.00 \\ 0.88 & 0.88 & 0.82 \\ 0.88 & 0.01 & 0.82 \\ 0.98 & 0.27 & 1.00 \end{pmatrix}$$

$$ReLU(XW^{(2)} + \vec{1}\mathbf{b}^{(2)T}) = ReLU\left(\begin{pmatrix} 0.73 & 0.99 & 1.00 \\ 0.88 & 0.88 & 0.82 \\ 0.88 & 0.01 & 0.82 \\ 0.98 & 0.27 & 1.00 \end{pmatrix}\begin{pmatrix} 1 & 2 & 0.5 \\ 1 & 2 & 0.5 \\ 1 & 2 & 0.5 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}\right)$$

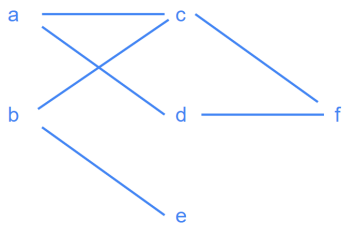
$$= ReLU\begin{pmatrix} 3.72 & 6.44 & 2.36 \\ 3.58 & 6.16 & 2.29 \\ 2.71 & 4.42 & 1.86 \\ 3.25 & 5.5 & 2.13 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

The final output is $\begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{pmatrix} W^{(3)} + \mathbf{b}^{(3)}$.

$$sigmoid(XW^{(3)} + \vec{1}\mathbf{b}^{(3)T}) = sigmoid\left(\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0.5 \\ 2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}\right)$$

$$= sigmoid\begin{pmatrix} 3.5 \\ 4.5 \\ 4.5 \\ 3.5 \end{pmatrix} = \begin{pmatrix} 0.97 \\ 0.99 \\ 0.99 \\ 0.97 \end{pmatrix}$$

2. The line extension of the given poset is:



3. The reason why we choose to use sigmoid activation function is because the gradient is 0 when $\mathbf{v}^T\mathrm{h} + \mathbf{b}$ outside the unit interval. Gradient equals to 0 means there's no direction to improve the parameter.
4.

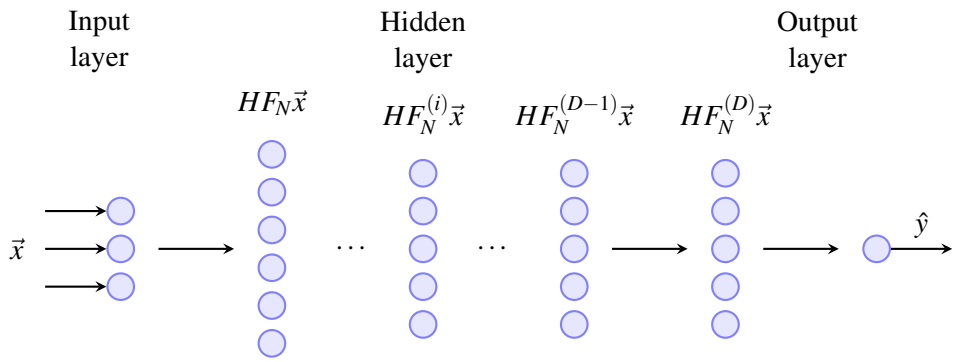
$$W^{(0)} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}, W^{(1)} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}, W^{(2)} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}, W^{(3)} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

10.7 Output Neural

Definition 10.7.1 Given a neural network $N = (H, F, G)$ with depth D , for $\vec{x} \in \mathbb{R}^{\#P_0} \cong V_P^{(0)}$ (an input vector), the hidden features associcted to $\vec{x}$ at layer i are

$$HF_N^{(i)}\vec{x} = G_i F_{i-1} \dots F_1 G_1 F_0 \vec{x}$$

where $HF_N^{(0)}\vec{x} = \vec{x}$.



So the output neural are responsible for predicting the target from $HF_N^{(D-1)}\vec{x}$. The choice of activation at the output layer depends on the type of targets possible (i.e. on $p_{\text{data}}(\vec{y})$)

■ **Example 10.9**

Input: $\vec{x} =$ (some statistics on two sports teams)

$$y = \begin{cases} 0 & \text{if team 1 wins} \\ 1 & \text{if team 2 wins} \end{cases}$$

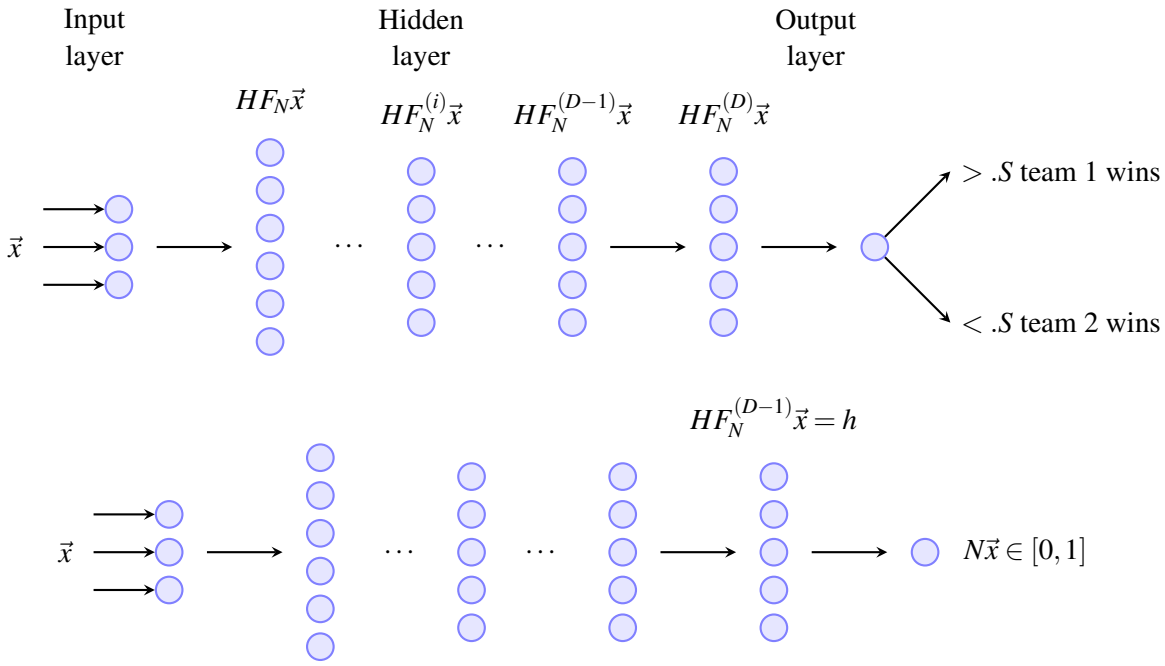
Dataset $X, \vec{y}$ of previous result. Goal: predict winners of upcoming games.
Idea: Apply a neural network N so that

$$N\vec{x} = p(y = 1 \mid \vec{x})$$

Sigmoid activation makes sense here. ■

■ **Example 10.10** Input $\vec{x}$ statstics on sports teams 1 & 2

$$y = \begin{cases} 1 & \text{team 1 wins} \\ 0 & \text{team 2 wins} \end{cases}$$



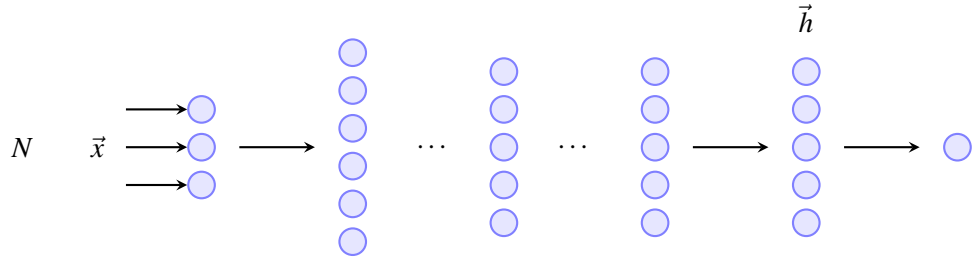
Idea: want $N\vec{x} = p(y = 1 \mid \vec{x})$

Makes sense to use sigmoid activation for $G^{(D)}$. ∎

R This is equvalent p transforming $\vec{x}$ to new features $\vec{h}$ and performing logistic regression on $\vec{h}$.

■ **Example 10.11** $\vec{x}$ statistics about a company y has stock price in 1 week (y has continuous distribution e.g. Gaussian)

linear activation $G^{(D)}(y) = y$ makes sense here.



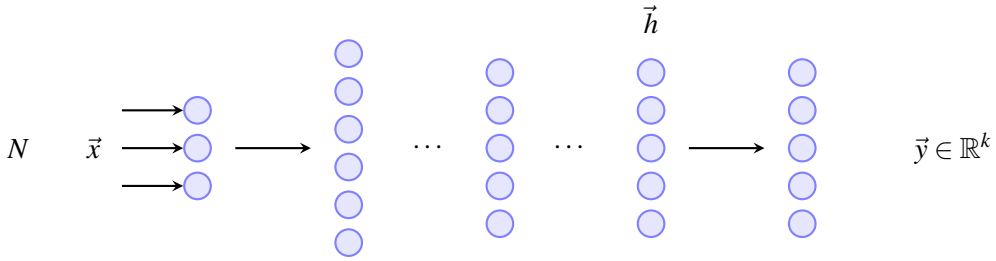
■

R This is equivalent to performing linear regression on $HF_N^{(D-1)}(\vec{x})$

■ **Example 10.12** $\vec{X}$ = vectorized image of an animals $\begin{pmatrix} k \\ \text{types} \end{pmatrix}$

$$y = \begin{cases} e_1 & \text{if cat} \\ e_2 & \text{if dog} \\ \vdots & \\ e_k & \text{if llama} \end{cases} \in \mathbb{R}^k = \langle e_1, e_2, \cdots, e_k \rangle$$

(call a one hot encoding) want $N\vec{x} = \hat{y}$ where $\hat{y}_i = p(y = e_i \mid \vec{x})$, $\left(\therefore \sum_{i=1}^k \hat{y}_i = 1\right.$



Define $G^{(0)}$ = softmax where $\text{softmax}(\vec{h}) = \frac{\exp_p(h_i)}{\sum_{j=1}^k \exp(h_j)}$ Softmax makes sense here. ■

R In each case we are choosing an activation function which will give the gradient nice properties. We want the gradient to have the property that when W, B are far from "local min, the gradient stays sufficiently large, (so GD does not "get stuck").

10.8 Optimization for Neural Networks

Let N be a neural network and $J(W,B)$ a cost function to measure "proximity" between $N_{\vec{x}}$ and target $\vec{y}$ on a dataset X, Y . To run gradient descent to find good values for W, B we need to compute

$$\frac{\partial J}{\partial W^{(i)}}, \frac{\partial J}{\partial b^{(i)}} \quad 0 \le i \le D-1$$

Definition The signal ost of layer i for input $\vec{x}$ is

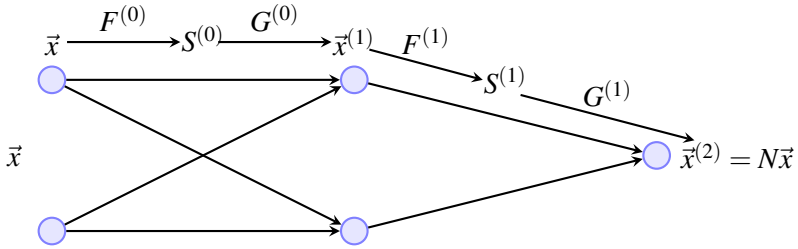
$$S^{(i)} = F^{(i)}G^{(i)}\dots G^{(1)}F^{(0)}\vec{x} \quad S^{(0)} = F^{(0)}\vec{x}$$

R $G^{(i+1)}S^{(i)} = HF_N^{(i+1)}(\vec{x}) = \vec{x}^{(i+1)}$

■ **Example 10.13**

$$W^{(0)} = \begin{pmatrix} w_{11}^0 & w_{12}^0 \\ w_{21}^0 & w_{i2}^0 \end{pmatrix} \quad \vec{b}^{(0)} = \begin{pmatrix} b_i^i \\ b_2^0 \end{pmatrix}$$
$$W^{(1)} = (w'_{11}w'_{12}) \quad \vec{b}^{(1)} = (b'_1)$$
$$S^{(0)} = W^{(0)}\vec{x} + \vec{b}^{(0)}$$
$$\vec{x}^{(1)} = G^{(1)}\left(s^{(0)}\right) = G^{(1)}\left(W^{(0)}\vec{x} + \vec{b}^{(0)}\right)$$
$$S^{(1)} = W^{(1)}G^{(1)}\left(S^{(0)}\right) + b^{(1)} = W^{(1)}\vec{x}^{(1)} + b^{(1)}$$
$$\vec{x}^{(2)} = G^{(2)}\left(s^{(1)}\right) = N\vec{x}$$

Cost function $J(W,B) = \sum_{i=1}^{m} L(N\vec{x}_i, \vec{y}_i)$. Need to find derivetives of $L(N\vec{x}, \vec{y})$ ■



R $\dfrac{\partial}{\partial v}(f \circ g)(v) = \left.\dfrac{\partial f(w)}{\partial w}\right|_{w=g(v)} \dfrac{\partial}{\partial v} g(v)$

$$\text{where } \left.\frac{\partial f(w)}{\partial w}\right|_{w=g(v)} = \frac{\partial f}{\partial g(v)}$$

10.9 Backpropagation

Backpropagation is the algorithm we use to train feed forward neural networks. This algorithm efficiently computes the gradient of the loss function with respect to the weights of the network, adjusting the weights and biases of the model. This allows us to use gradient descent or stochastic gradient descent methods to train multilayer networks, updating weights to minimize loss.

We will explore backpropagation for a network defined by the poset $P(n_0, n_1, \dots n_D)$, i.e., each layer is fully connected to adjacent layers, dataset $X \in \mathbb{R}^{m\times n}, Y \in \mathbb{R}^{m\times k}$ and a loss function

$$J(W,B) = \frac{1}{m}\sum_{i=1}^{m} L(N\mathbf{x}_i, \mathbf{y}_i)$$

Now, let's find expressions for δ, so we may use them to find expressions for the gradient of the loss function.

$$\begin{aligned}\delta^{(D-1)} &= \frac{\partial L}{\partial S^{(D-1)}} = \frac{\partial L}{\partial N\mathbf{x}_i} \cdot \frac{\partial N\mathbf{x}_i}{\partial S^{(D-1)}} \\ &= \frac{\partial L}{\partial N\mathbf{x}_i} \cdot \frac{\partial G^{(D)}}{\partial S^{(D-1)}}\end{aligned}$$

R Use $N\mathbf{x} = G^{(D)}S^{(D-1)}$

and for $0 \leq l < D-1$:

$$\begin{aligned}\delta^{(l)} &= \frac{\partial L}{\partial S^{(l)}} = \frac{\partial L}{\partial S^{(l+1)}} \cdot \frac{\partial S^{(l+1)}}{\partial S^{(l)}} \\ &= \delta^{(l+1)} W^{(l+1)} \frac{\partial G^{(l+1)}}{\partial S^{(l)}}\end{aligned}$$

R Use $S^{(l+1)} = W^{(l+1)}G^{(l+1)}S^{(l+1)} + b^{(l+1)}$

With these handy values, we can now calculate:

$$\begin{aligned}\frac{\partial L}{\partial W^{(l)}} &= \frac{\partial L}{\partial S^{(l)}} \cdot \frac{\partial S^{(l)}}{\partial W^{(l)}} \\ &= \sum_{k=1}^{n_{l+1}} \frac{\partial L}{\partial S_k^{(l)}} \cdot \frac{\partial S_k^{(l)}}{\partial W^{(l)}} \\ &= \sum_{k,j} \delta_k^{(l)} \sum_{j=1}^{n_l} \mathbf{E}_{kj}\mathbf{x}_j^{(l)} \\ &= \sum_{k=1}^{n_{l+1}} \delta_k^{(l)}\mathbf{x}_j^{(l)}\mathbf{E}_{kj} \\ &= \text{outer}(\delta^{(l)}, \mathbf{x}^{(l)})\end{aligned}$$

R Use $S_k^{(l)} = \sum_{j=1}^{n_l} W_{k,j}^{(l)} x_j^{(l)} + b_k^{(l)}$
Note that $\frac{\partial}{\partial W^{(l)}} W_{k,j}^{(l)} = \mathbf{E}_{kj}$ where $\mathbf{E}_{kj}$ is a matrix with an entry 1 in (k, j) and zeros elsewhere.

Similarly, we calculate:

$$\begin{aligned}\frac{\partial L}{\partial b^{(l)}} &= \frac{\partial L}{\partial S^{(l)}} \cdot \frac{\partial S^{(l)}}{\partial b^{(l)}} \\ &= \frac{\partial L}{\partial S^{(l)}} \\ &= \delta^{(l)}\end{aligned}$$

Equipped with these partial derivatives with respect to the weight and bias matrices, we can now calculate the gradient for our arbitrary loss function:

$$J(W,B) = \frac{1}{m}\sum_{i=1}^{m} L(N\mathbf{x}_i, \mathbf{y}_i)$$

$$\frac{\partial J}{\partial W^{(l)}} = \frac{1}{m}\sum_{i=1}^{m} \text{outer}(\delta_i^{(l)}, \mathbf{x}_i^{(l)})$$

$$\frac{\partial J}{\partial b^{(l)}} = \frac{1}{m}\sum_{i=1}^{m} \delta_i^{(l)}$$

10.10 Regularization

As we have learned with previous models, we do not want to train our feed forward neural network in such a way that the model is overfitted or underfitted. As such, we can use regularization to calibrate our model and minimize loss.

Let's consider stochastic gradient descent, a variant of gradient descent where only a few samples are selected randomly per iteration, instead of the complete data. We can take a look at the algorithm we use for each step:

Algorithm 2 Regularization for Stochastic Gradient Descent Step

Pick $\alpha > 0$
Pick an arbitrary batch Ba
Approximate $J(W,B) = \frac{1}{|Ba|}\sum_{i\in Ba} L(N\mathbf{x}_i, \mathbf{y}_i)$ and append to list of costs
for $i \in Ba$ **do**
    Compute $[(\mathbf{x}_i^{(0)}, \S_i^{(0)}), \ldots, (\mathbf{x}_i^{(D-1)}, \S_i^{(D-1)}), (\mathbf{x}_i^{(D)})]$
    Compute $[\delta_i^{(D-1)}, \ldots, \delta_i^{(0)}]$
    Compute $\frac{\partial J}{\partial W^{(l)}} = \frac{1}{|Ba|}\sum_{i\in Ba} \text{outer}(\delta_i^{(l)}, \mathbf{x}_i^{(l)})$
    Compute $\frac{\partial J}{\partial b^{(l)}} = \frac{1}{|Ba|}\sum_{i\in Ba} \delta_i^{(l)}$
end for
for l in range(D) **do**
    $W^{(l)} = W^{(l)} - \alpha\frac{\partial J}{\partial W^{(l)}}$
    $b^{(l)} = b^{(l)} - \alpha\frac{\partial J}{\partial b^{(l)}}$
end for

If the correct neural network architecture is used, stochastic gradient descent can be used to find weights and biases that provide good performance on training and testing data.

10.11 Common Loss Functions

There are multiple loss (cost) functions we can use to train a feed forward neural network. Let's take a look at a few commonly used functions:

Definition 10.11.1 *Square Error Loss* is defined by

$$L_{SE}(N\mathbf{x},\mathbf{y}) = ||N\mathbf{x}-\mathbf{y}||_2^2$$

Square Error Loss is mostly used for linear regression, and is one of the simplest loss functions. The function ensures that the trained model has no outliers with huge errors, since a larger weight is assigned to such errors (thanks to the squaring part of the function).

Definition 10.11.2 *Cross Entropy Loss* is defined by

$$\begin{aligned} L_{CE}(N\mathbf{x},\mathbf{y}) &= -\mathbf{y}^\top Log(N\mathbf{x}) \\ &= -\sum_{i=1}^{n_D} \mathbf{y}_i \log((N\mathbf{x})_i) \end{aligned}$$

Cross Entropy Loss is used for multiclass classifiers which use softmax output neurons.

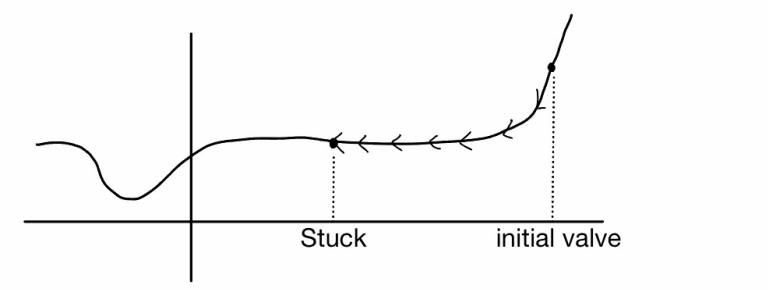
Definition 10.11.3 *Binary Cross Entropy Loss* is defined by

$$L_{BCE}(N\mathbf{x},\mathbf{y}) = -\mathbf{y}^\top \log(N\mathbf{x}) - (1-\mathbf{y})\log(1-N\mathbf{x})$$

As its name suggests, Binary Cross Entropy Loss is used for binary classifiers, which use sigmoid output neurons.

10.12 Vanishing Gradient Problem

In practice, the function of a neural network N is non-convex, so there may be flat regions one may "get stuck" in during (stuck-stic) gradient descent.



R Factors that affect the "flatness" of cost function:

1. Choice of the loss function:

$$\begin{cases} SE & \text{lot of flat regions} \\ CE & \text{less flat regions} \\ BCE & \text{least of flat regions} \end{cases}$$

2. Choice of activation(the ones we discussed so fare are good choices)

Note: Using several sigmoids in hidden layers can cause flatness.

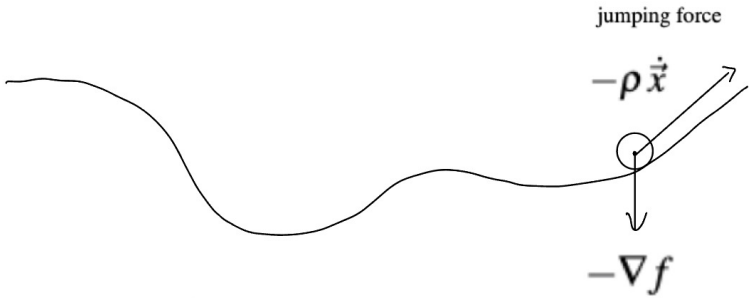
Suggestion: Use ReLu in hidden layers

3. Momentum
4. Weight Initialization

10.12.1 Momentum Method

Motivation(from physics). Momentum is one of the most popular techniques used to improve the speed of convergence of the Gradient Descent algorithm.

Consider a ball rolling on a surface



f = potential function, $f(\vec{x}) = mgx_3$

$$\sum \vec{F} = m\vec{a} \Longrightarrow \ddot{\vec{x}} = \rho\,\dot{\vec{x}} - \nabla f(\vec{x})$$

To solve: Express as a system of First order ODEs

$$\dot{\vec{V}} = -\rho\,\vec{v} - \nabla f(\vec{x})$$

$$\dot{\vec{x}} = \vec{V}$$

Write this as a finite difference system (Pick ∇t small enough so that $\rho\,\nabla t < 1$)

$$\vec{V}_{n+1} - \vec{V}_n = -\rho\,\vec{V}_n\,\nabla t - \nabla f(\vec{x}_n)\,\nabla t$$

$$\vec{X}_{n+1} - \vec{X}_n = \vec{V}_{n+1}\,\nabla t$$

Let $\varepsilon = \nabla t$, $\mu = 1 - \rho\,\varepsilon$ with $0 \leq \mu < 1$

$$\vec{V}_{n+1} = \mu\,\vec{V}_n - \varepsilon\,\nabla f(\vec{x}_n)$$

$$\vec{X}_{n+1} = \vec{X}_n + \varepsilon\,\vec{V}_{n+1}$$

let $\vec{V}\,\varepsilon = \tilde{\vec{V}}$ (new scale for velocity)

$$\frac{\tilde{\vec{V}}_{n+1}}{\varepsilon} = \frac{\mu\,\tilde{\vec{V}}_n}{\varepsilon} - \varepsilon\,\nabla f(\vec{x}n)$$

Define $\alpha = \varepsilon^2$

$$\tilde{\vec{V}}_{n+1} = \mu\,\tilde{\vec{V}}_n - \alpha\,\nabla f(\vec{x}n)$$

$$\vec{X}_{n+1} = \vec{X}_n + \tilde{\vec{V}}_{n+1}$$

if $\mu = 0$, this is just a gradient descent

■ **Example 10.14** The problem with gradient descent is that the weight update at a moment (t) is governed by the learning rate and gradient at that moment only. It doesn't take into account the past steps taken while traversing the cost space.

$$\nabla w(t) = -\eta\delta(t)$$

The gradient of the cost function at saddle points(plateau) is negligible or zero, which in turn leads to small or no weight updates. Hence, the network becomes stagnant, and learning stops. (a) How can momentum fix this?

Solution: Imagine you have a ball rolling from point A. The ball starts rolling down slowly and gathers some momentum across the slope AB. When the ball reaches point B, it has accumulated enough momentum to push itself across the plateau region B and finally follow slope BC to land at the global minima C.

(b) How can this be used and applied to Gradient Descent?

Solution: To account for the momentum, we can use a moving average over the past gradients. In regions where the gradient is high like AB, weight updates will be large. Thus, in a way we are gathering momentum by taking a moving average over these gradients. But there is a problem with this method, it considers all the gradients over iterations with equal weight-age. The gradient at t=0 has equal weight-age as that of the gradient at the current iteration t. We need to use some sort of weighted average of the past gradients such that the recent gradients are given more weight-age.

■

10.12.2 Weight Initialization

Weight initialization is an important consideration in the design of a neural network model.

Neural network models are fit using an optimization algorithm called stochastic gradient descent that incrementally changes the network weights to minimize a loss function, hopefully resulting in a set of weights for the mode that is capable of making useful predictions.

The nodes in neural networks are composed of parameters referred to as weights used to calculate a weighted sum of the inputs. Neural net performance depends heavily an $W^{(l)}$ weight initialization especially if D is large. Depends less on bias initialization $b^{(l)}$ but this depends on architecture.

Claim: ① If the initialized values of $W^{(l)}$ are too small, the variance of $J^{(l)}$ decreases at each layer ② If initialized weights are too large, variance $s^{(l)}$ amplifies at each layer.

10.12.3 Xavier Initialization

Based on the assumption that Var($s^{(\lambda)}$ should remain constant at each layer.

For P(n_0,...,n_0) entries of $W^{(l)}$ should be sampled from $N(x;0,\frac{2}{n_{l-1}+n_l})$.

Many other "suggestions" exist

- Read section 8.4 of the Deep Learning book
- Read section 6.3 of the Deep Learning Architectures

Bias can usually be initialized to $\vec{O}$

A way to do SGD on neural net N (for P($n_0,\ldots,n_0$))

Pick $\alpha_{stepsize}$<0, $\lambda_{reg-parameter}$<0, $\mu_{momentum}$<0, max_iters, batch_size
For $0 \leq l \leq D-1$ initialize $W^{(l)}$ by sampling $N(x;0,\frac{2}{n_{l-1}+n_l})$ $b^{(o)} = (\vec{O})$

initialize $V_w^{(l)} = np.zeros(W^{(l)}.shape)$
$V_b^{(l)} = np.zeros(b^{(l)}.shape)$

$epochs = 0$

While epochs <max_iters:
randomly pick $B \subseteq \{1,...,m\} with \|B\| = batch\_size$
for $i \in B$:
compute $[(x_i^{(0)}, s_i^{(0)}, (x_i^{(1)}, s_i^{(1)}), ..., (x_i^{(D-1)}, s_i^{(D-1)}), (x_i^{(0)})]$
compute $\delta_i^{(D-1)}$,...,$\delta_i^{(0)}$
compute $\frac{\partial J}{\partial W^{(l)}}$
for $0 \leq l \leq D-1$:
$V_w^{(l)} = \mu V_w^{(l)} - \alpha \frac{\partial J}{\partial W^{(l)}}$
$V_b^{(l)} = \mu V_b^{(l)} - \alpha \frac{\partial J}{\partial b^{(l)}}$
$W^{(l)} = W^{(l)} + V_W^{(l)}$
$b^{(l)} = b^{(l)} + V_b^{(l)}$
append cost to list of costs
return W, B, costs

■ **Example 10.15** Derive Xavier Initialization

$$W_{i,j}^{[l]} = N(0, \frac{1}{n^{[l-1]}}$$

Solution:

$$\begin{aligned}
Var(a_i^{[\ell-1]}) &= Var(a_i^{[\ell]}) \\
&= Var(z_i^{[\ell]}) \quad \longleftarrow \text{linearity of } \textbf{tanh} \text{ around zero } tanh(z) \approx z \\
&= Var(\sum_{j=1}^{n^{[\ell-1]}} w_{ij}^{[\ell]} a_j^{[\ell-1]}) \\
&= \sum_{j=1}^{n^{[\ell-1]}} Var(w_{ij}^{[\ell]} a_j^{[\ell-1]}) \quad \longleftarrow \text{variance of independent sum } Var(X+Y) = Var(X) + Var(Y) \\
&= \sum_{j=1}^{n^{[\ell-1]}} E[w_{ij}^{[\ell]}]^2 Var(a_j^{[\ell-1]}) + \\
&\quad E[a_j^{[\ell-1]}]^2 Var(w_{ij}^{[\ell]}) + \quad \longleftarrow \text{variance of independent product } Var(XY) = E[X]^2 Var(Y) + E[Y]^2 Var(X) + Var(X)Var(Y) \\
&\quad Var(w_{ij}^{[\ell]}) Var(a_j^{[\ell-1]}) \\
&= n^{[\ell-1]} Var(w_{ij}^{[\ell]}) Var(a_j^{[\ell-1]}) \implies Var(W) = \frac{1}{n^{[\ell-1]}}
\end{aligned}$$

■

■ **Example 10.16** How is aW_i calculated when using Xavier initialization? For a current Layer, let s be the output connections of the layer and e the input connections, then: $f(W) = \frac{2}{e+s}$

Solution:

In the case of Xavier initialization (also called "Glorot normal" in some software), the parameters are initialized as random draws from a truncated normal distribution with mean 0 and standard deviation

$\sigma = \sqrt{\frac{2}{a+b}}$

where a is the number of input units in the weight tensor, and b is the number of output units in the weight tensor. ∎

10.13 Exercises

1. Given $N\mathbf{x} = [1.0, 0.8, 0.2]$ and $\mathbf{y} = [1, 1, 0]$, compute the per-example loss for the following functions:
   (a) Square Error Loss $L_{SE}(N\mathbf{x}, \mathbf{y})$
   (b) Cross Entropy Loss $L_{CE}(N\mathbf{x}, \mathbf{y})$
2. Given $N\mathbf{x} = [0.7]$ and $\mathbf{y} = [1]$, find the Binary Cross Entropy Loss $L_{BCE}(N\mathbf{x}, \mathbf{y})$
3. How does squared error affect backpropagation? Derive an expression for $\delta^{(D-1)}$ where $L = L_{SE}$

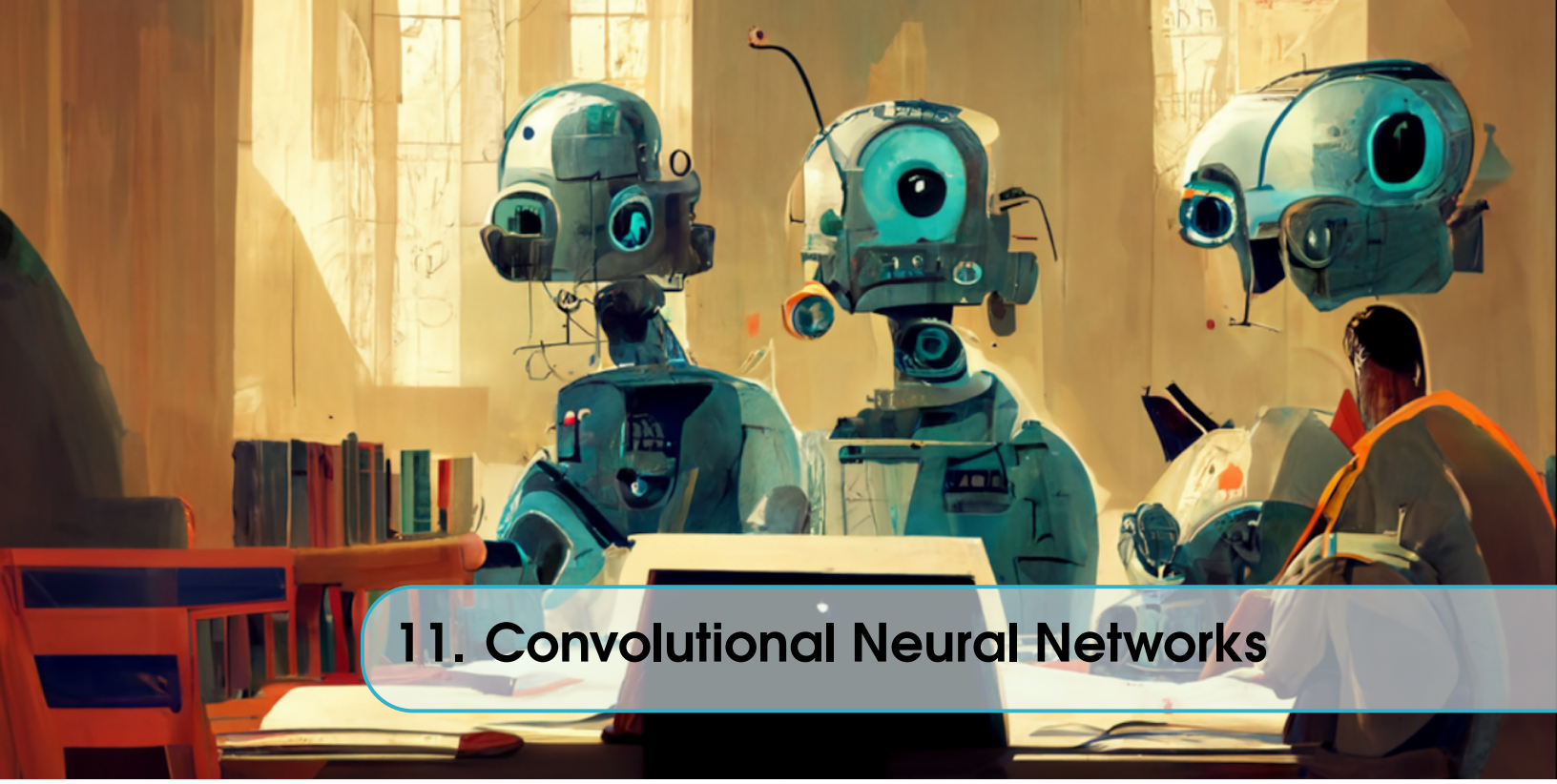
10.13.1 Solutions

1. The per-example loss for the functions is:
   (a) $L_{SE}(N\mathbf{x}, \mathbf{y}) = 0.08$
   (b) $L_{CE}(N\mathbf{x}, \mathbf{y}) = 0.2231435513142097$
2. $L_{BCE}(N\mathbf{x}, \mathbf{y}) = 0.10536051565782628$
3. First, let us compute the gradient of the loss function with respect to $N\mathbf{x}$:

$$L_{SE}(N\mathbf{x}, \mathbf{y}) = ||N\mathbf{x} - \mathbf{y}||_2^2$$
$$\frac{\partial L_{SE}}{\partial N\mathbf{x}} = 2(N\mathbf{x} - \mathbf{y})^\top$$

Then, we can substitute to get

$$\delta^{(D-1)} = \frac{\partial L_{SE}}{\partial N\mathbf{x}} \cdot \frac{\partial N\mathbf{x}}{\partial S^{D-1}}$$
$$= 2(N\mathbf{x} - \mathbf{y})^\top \cdot \frac{\partial N\mathbf{x}}{\partial S^{D-1}}$$



11. Convolutional Neural Networks

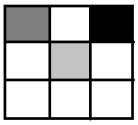
11.1 Introduction

11.1.1 Idea

In order to introduce convolutional neural networks, let's explore the role they play in neural networks. A neural network architecture should be chosen in accordance to the type of structure of input data. Consider the following example.

■ **Example 11.1** A data matrix X containing data such as statistics on houses with no real structure may use a fully connected architecture. Instead, let's take the example where X is a matrix with rows of image pixel data.

Take this image to be a black and white image thay may be represented by a 3x3 matrix of pixels where each pixel is a level of "darkness" defined by the values: 0,1,2,3.



Pixels may be placed in a row by arranging the columns in order of their index as features.

We get the vector $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$. Note that in the 3x3 matrix, x_4 is near x_1 and x_7, but the vector looses this structure.

This is our motivation for introducing a new type of layer to our neural network called a convolutional layer, in order to preserve a particular "structure" of the image.

■

11.2 Convolutional Layers

11.2.1 Convolutional Layer with 1D Inputs

Definition 11.2.1 A discrete 1-D signal is a sequence $y = (y_i)_{i\in\mathbb{Z}}$ of real numbers and is L^1-finite if $\sum |y_k| < \infty$ and compact if there exists N such that $|k| > N$ implies $y_k = 0$. A 1-D kernel is a compact support signal of weights, w. Where $w = (w_i)_{i\in\mathbb{Z}}$. Since usually we want w to be a probability distribution, then $\Sigma_i w_i = 1$.

Thus, the convolution, often called the cross correlation in signal processing, of y and w is $z = y * w$ where $z_j = \sum_{k\in\mathbb{Z}} y_{y+k} w_k$

■ **Example 11.2**
let $w_i = \begin{cases} 1/2 & \text{if} \quad i = 0,1 \\ 0 & \text{else} \end{cases}$
So $(y+w)_i = \sum_{k\in\mathbb{Z}} y_{i+k} w_k = \frac{y_i + y_{i+1}}{2}$ ■

More generally, $w_i = \begin{cases} 1/M & \text{if} \quad i = 0,\ldots,M-1 \\ 0 & \text{else} \end{cases}$
Now $(y+w)_i = \sum_{k\in\mathbb{Z}} y_{i+k} w_k = \frac{y_i + y_{i+1} + \ldots + y_{i+M-1}}{M}$

■ **Example 11.3**
let $y = (1,2,3,4,5,6)$ and $w = (\frac{1}{2}, \frac{1}{2})$
Here $y + w = (\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \frac{7}{2}, \frac{9}{2}, \frac{11}{2}, 3) = \frac{y_i + y_{i+1}}{2}$
The operation is as follows

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & & & & & \\ & \frac{1}{2} & \frac{1}{2} & & & 0 & \\ & & \frac{1}{2} & \frac{1}{2} & & & \\ & & & \frac{1}{2} & \frac{1}{2} & & \\ & & & & \frac{1}{2} & \frac{1}{2} & \\ & 0 & & & & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(0) + \frac{1}{2}(1) \\ \frac{1}{2}(1) + \frac{1}{2}(2) \\ \frac{1}{2}(2) + \frac{1}{2}(3) \\ \frac{1}{2}(3) + \frac{1}{2}(4) \\ \frac{1}{2}(4) + \frac{1}{2}(5) \\ \frac{1}{2}(5) + \frac{1}{2}(6) \\ \frac{1}{2}(6) + \frac{1}{2}(7) \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ \frac{3}{2} \\ \frac{5}{2} \\ \frac{7}{2} \\ \frac{9}{2} \\ \frac{11}{2} \\ 3 \end{pmatrix}$$ ■

When we use this in a neural network layer, we usually ignore the first and last component. We may also ignore some parts on the inside or only use alternating entries.

11.2.2 Convolutional Layer with 3D Inputs

■ **Example 11.4** In this example, we look at a neural network which uses two convolutional layers that takes images of animals and outputs a prediction of cat, dog, bear or pig.

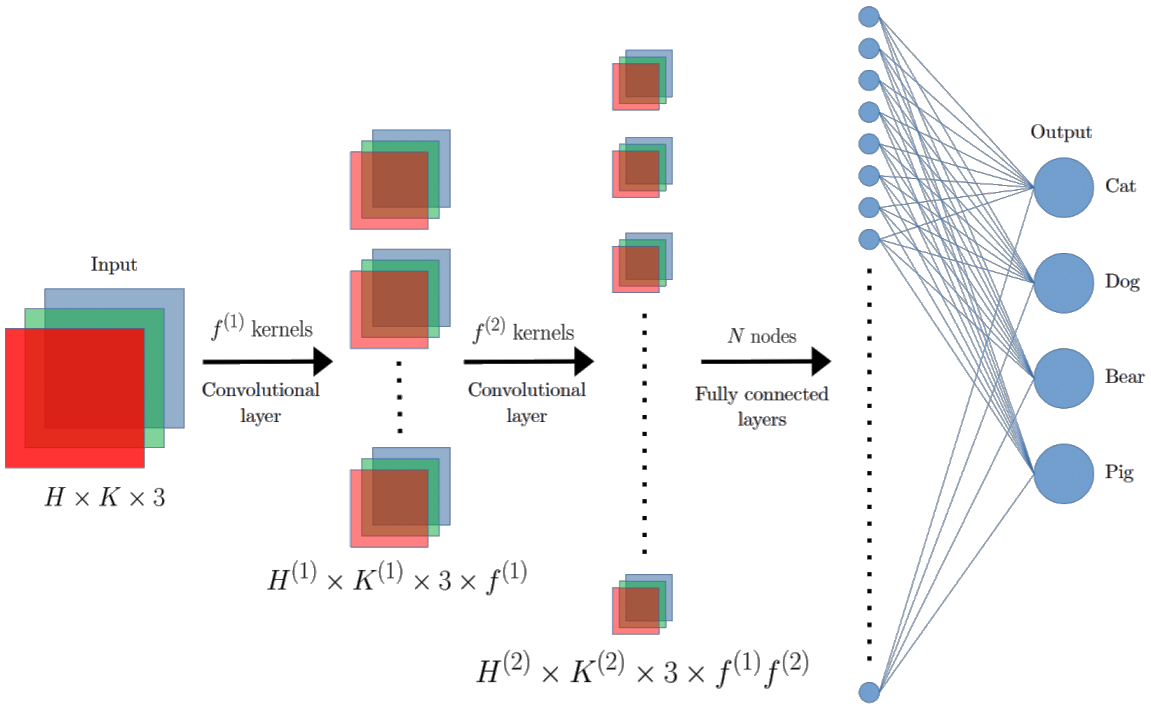


Figure 11.1: Neural network using convolutional layers to predict images of cat, dog, bear and pig.

The input consists of images of animals stored as $H \times K$ dimensions with 3 color channels containing its associated RGB data. As discussed in the previous section, every signal passing through a convolutional layer will increase the number of features multiplicatively. This can be see in the formula to compute the l-th signal $Z^{(l)}$ given kernel $W^{(l)}$, bias $b^{(l)}$ and previous signal $Z^{(l-1)}$ which has f_in features:

$$Z^{(l)}_{i,j,c,(f_{\text{in}},f_{\text{out}})} = G\left[\left(\sum_{s,p} Z^{(l)}_{i+p,j+s,c,f_{\text{in}}} W^{(l)}_{p,s,c,(f_{\text{in}},f_{\text{out}})}\right) + b^{(l)}_{i,j,c,(f_{\text{in}},f_{\text{out}})}\right].$$

In this case, and in general, we find that

$$\begin{aligned} H^{(2)} &\leq H^{(1)} \leq H \\ K^{(2)} &\leq K^{(1)} \leq K \\ f^{(1)} f^{(2)} &\geq f^{(1)} \geq 1 \end{aligned}$$

$H^{(l)}$ and $K^{(l)}$ is nonincreasing since we are applying convolution (see [relevant section when it gets written]). Sometimes, we want to preserve the size of $H^{(l)}$ and $K^{(l)}$ in all layers of our network. Moreover, the number of nodes N in the example can be very large and computationally costly. To mitigate this problem, we introduce a padding and pooling process into our neural network.

■

11.2.3 Padding

Definition 11.2.2 For any matrix $\mathbf{A} \in \mathbb{R}^{m,n}$, a *$p-$padding* applied to $\mathbf{A}$ is the transformation $\mathbf{I}_L\mathbf{A}\mathbf{I}_R$, where $\mathbf{I}_L \in \mathbb{R}^{m+2p\times m}$ and $\mathbf{I}_R \in \mathbb{R}^{n\times n+2p}$ such that

$$(\mathbf{I}_L)_{i,j} = \begin{cases} 1 & \text{if } i = j-1 \text{ and } p+1 \leq i \leq m+2p-1 \\ 0 & \text{else} \end{cases}$$

and

$$(\mathbf{I}_R)_{i,j} = \begin{cases} 1 & \text{if } j = i+1 \text{ and } p+1 \leq j \leq n+2p-1 \\ 0 & \text{else} \end{cases}$$

In simpler terms, if we have a matrix $\mathbf{A} = \begin{bmatrix} a_{1,2} & a_{2,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$ and we apply $1-$padding to this, we will obtain a result of the form

$$\mathbf{A}_{\text{padded}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & a_{1,1} & a_{1,2} & 0 \\ 0 & a_{2,1} & a_{2,2} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

That is, we “surround" the entries of $\mathbf{A}$ with zeros.

Now suppose we have a 4×4 image, and we perform a convolution with stride 1 and kernel of size (2×2). Then

$$(4\times 4) *_{(1,1)} (2\times 2) = (3\times 3)$$

Then by applying convolution to this with kernel of size (2×2), the result is of size (5×5). Unfortunately, the size is not preserved, and we instead get an increase in the image size. However, there is a p value to pad which preserves the size faithfully, and we investigate the general case for a square filter.

Proposition 11.2.1 For a matrix $\mathbf{A} \in \mathbb{R}^{H\times K}$ and kernel $\mathbf{W} \in \mathbb{R}^{h\times k}$, if $h = k$ and h is odd, then a $p-$padding with $p = \frac{h-1}{2}$ applied to $\mathbf{A}$ preserves its size after convoluting with $\mathbf{W}$.

Proof. Since the size of $\mathbf{A}$ after convoluting with $\mathbf{W}$ is $(H+2p)-h+1\times(K+2p)-h+1$, solving $H+2p-h+1 = H$ yields $p = \frac{h-1}{2}$. The restriction that h is odd ensures that p is an integer. ■

Definition 11.2.3 A padding which preserves size is called “same" padding.

11.2.4 Pooling

When dealing with very large matrices, it is often useful to reduce their size to achieve better computational efficiency whilst preserving pertinent data of the original matrix. In neural networks which receive matrices of variable data sizes, pooling can also be used to compress them into similar sizes to compute. This is the motivation behind pooling.

Definition 11.2.4 A *local pooling* consists of partitioning (in some way) $\mathbf{A}^{m\times n}$ and computing a local statistic (e.g. maximum, minimum or average).

■ **Example 11.5** Given matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 1 \\ 1 & 5 & 2 & 7 \\ 10 & 1 & 2 & 4 \\ 8 & 9 & 1 & 8 \end{bmatrix},$$

we pool $\mathbf{A}$ by partitioning it into four squares of size 2×2 each (that is, $A_1 = \{1,2,1,5\}$, $A_2 = \{3,1,2,7\}$, $A_3 = \{10,1,8,9\}$, $A_4 = \{2,4,1,8\}$) and computing the maximum of each partition and store the result into a 2×2 matrix:

$$\begin{aligned} \mathbf{A}_{\text{pool}} &= \begin{bmatrix} \max\{i : i \in A_1\} & \max\{i : i \in A_2\} \\ \max\{i : i \in A_3\} & \max\{i : i \in A_4\} \end{bmatrix} \\ &= \begin{bmatrix} 5 & 7 \\ 10 & 8 \end{bmatrix} \end{aligned}$$

■

Definition 11.2.5 A *global pooling* is a local pooling between two or more matrices.

■ **Example 11.6** Global pooling is highly useful in reducing the rank of a tensor. In **Example** 11.4, given the 4-tensor signal $Z^{(l)}_{i,j,c,(f_{\text{in}},f_{\text{out}})}$, globally pooling over i, j with respect to maximum yields $Z'^{(l)}_{c,(f_{\text{in}},f_{\text{out}})}$ which is a 2-tensor. ■

■ **Example 11.7** To illustrate the power of pooling in terms of computational efficiency, we return to the neural network given in **Example** 11.4, and introduce pooling to reduce the size of our image at each step.

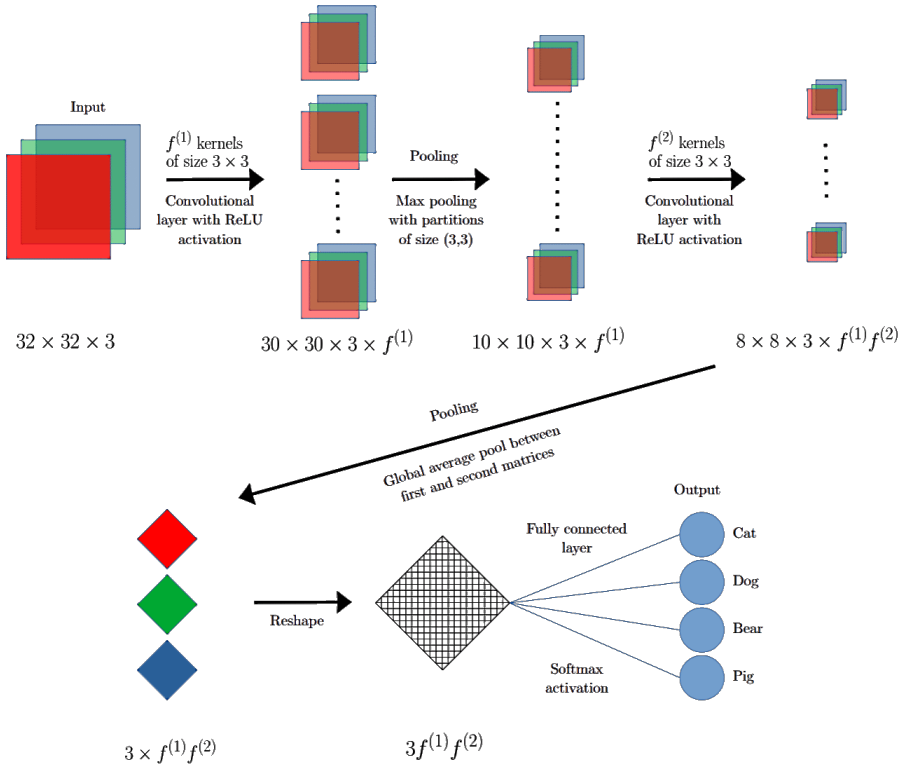


Figure 11.2: Neural network using convolutional layers and pooling to predict images of cat, dog, bear and pig. Note that the image size decreases at each step.

Now the fully connected layer at the last step involves only a single scalar output of magnitude $3f^{(1)}f^{(2)}$, instead of flattening a $H^{(2)} \times K^{(2)} \times 3 \times f^{(1)}f^{(2)}$ in the original architecture. It is good practice is to pool the data after each convolutional operation, as consecutive convolutional operations are computationally costly. ∎

11.3 Exercises

1. For matrices $\mathbf{X}_1, \ldots, \mathbf{X}_n$ with entries in $\mathbb{R}$ and size $m_{\mathbf{X}_j}$ and $n_{\mathbf{X}_j}$ respectively of $\mathbf{X}_j$, define $\max\{\mathbf{X}_1, \ldots, \mathbf{X}_n\} = \max\{i : i \in \mathbf{X}_1 \vee \ldots \vee i \in \mathbf{X}_n\}$, $\min\{\mathbf{X}_1, \ldots, \mathbf{X}_n\} = \min\{i : i \in \mathbf{X}_1 \vee \ldots \vee i \in \mathbf{X}_n\}$ and $\text{avg}\{\mathbf{X}_1, \ldots, \mathbf{X}_n\} = \frac{1}{n}\sum_{s,t}(\mathbf{X}_j)_{s,t}$.

   Compute the global pool of $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ below, storing your result in $\mathbf{v} \in \mathbb{R}^3$ where $\mathbf{v} = (\max\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}, \min\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}, \text{avg}\{\mathbf{A}, \mathbf{B}, \mathbf{C}\})$.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ -3 & 4 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 13 & -2 \\ 6 & 1 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 1 & 2 \\ 7 & 5 \end{bmatrix}$$

2. Given $\mathbf{I}_L$ and $\mathbf{I}_R$ defined in **Definition** 11.2.2, explicitly write down the entries of $\mathbf{I}_L$ and $\mathbf{I}_R$ for the $1-$padding of

$$\mathbf{A} = \begin{bmatrix} 5 & 3 \\ 8 & 2 \end{bmatrix}.$$

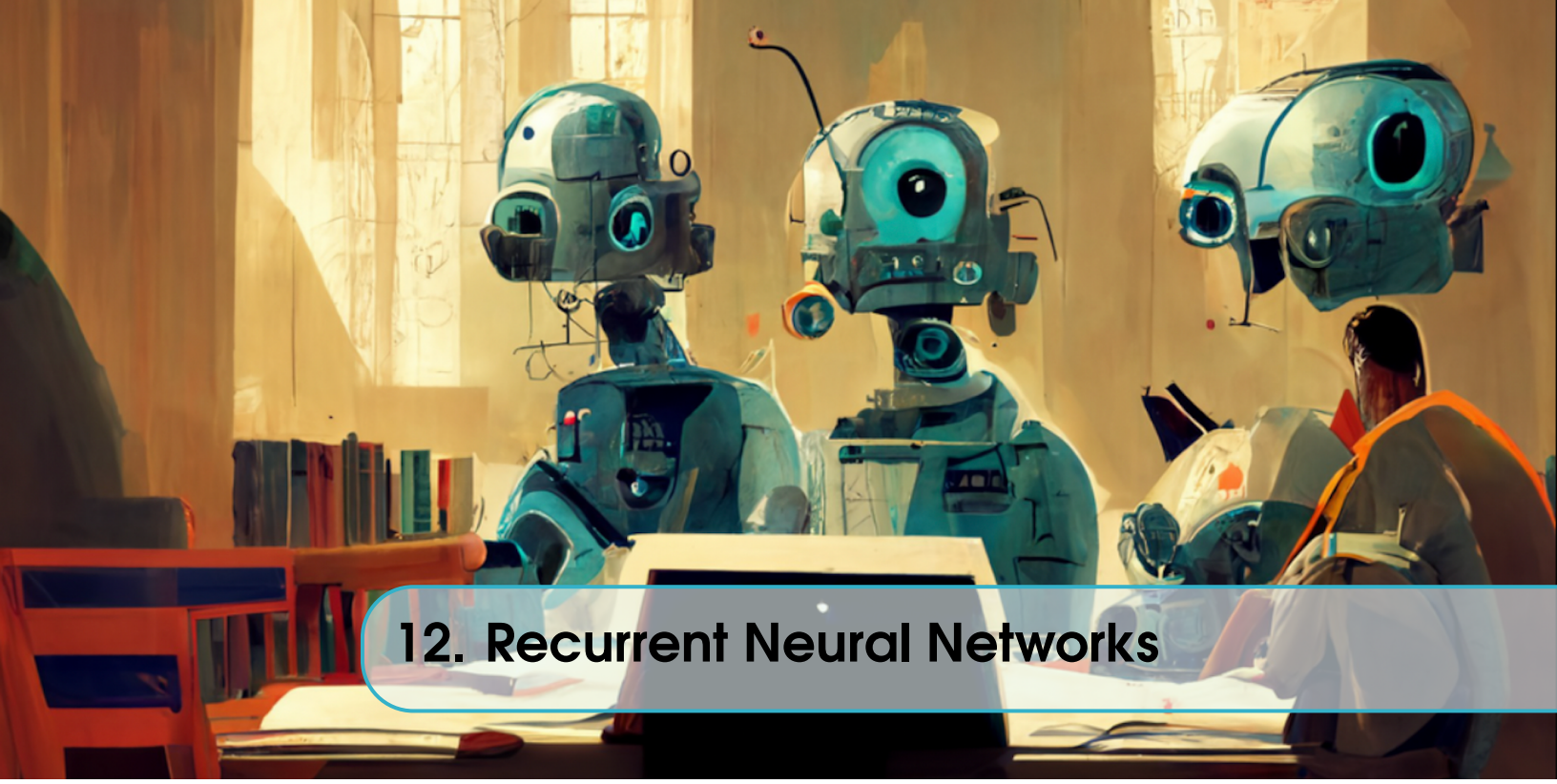
3. Write an algorithm or a pseudocode which computes the $p-$padding of a matrix. The inputs should consist of the matrix itself and p, and output should consist of the padded matrix.

11.4 Solutions to Exercises

1. $\mathbf{v} = (13, -3, \frac{37}{3})$
2. $$\mathbf{I}_L = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \mathbf{I}_R = \mathbf{I}_L^\top$$
3. Inputs are matrices $\mathbf{X}$, and a natural number p. We will use Python-like pseudocode and Numpy methods, although their operations can be generalized to any language which support such features.

Algorithm 3 $p-$padding($\mathbf{X}$,p)

m,n = $\mathbf{X}$.shape[0], $\mathbf{X}$.shape[1]
$\mathbf{I}_L, \mathbf{I}_R$ = np.eye(m), np.eye(n)
rowZero = np.atleast_2d(np.zeros($\mathbf{I}_L$.shape[1]))
colZero = np.atleast_2d(np.zeros($\mathbf{I}_R$.shape[1]))
colZero = colZero.$\top$
$\mathbf{I}_L$ = np.concatenate(($\mathbf{I}_L$,rowZero), axis=0)
$\mathbf{I}_L$ = np.concatenate((rowZero,$\mathbf{I}_L$), axis=0)
$\mathbf{I}_R$ = np.concatenate(($\mathbf{I}_R$,colZero), axis=1)
$\mathbf{I}_R$ = np.concatenate((colZero,$\mathbf{I}_R$), axis=1)
if $p > 1$ **then**
    return $p-$padding($\mathbf{I}_L\mathbf{X}\mathbf{I}_R, p-1$)
end if
return $\mathbf{I}_L\mathbf{X}\mathbf{I}_R$



12. Recurrent Neural Networks

12.1 Introduction

Consider the following situation: You are writing an email and you want to do an auto-complete. In this scenario the input would be an incomplete sentence and the output would be the next word in the sentence.

For example, let the input be:"How are". The answer/output of the auto-complete would be: "you".

A feedforward neural network is not good at solving this kind of problem. The reason is feedforward neural networks can only take fixed-sized inputs, however, a sequence can come in any length so the same network needs to be able to process any input length. Also, since sequences come in varying lengths, parameters need to be shared across each cell, a property a neural network would not support. In an RNN, the parameters in each cell are shared across the number of cells.

12.1.1 What is an RNN?

An RNN is a type of artificial neural network using sequential data for problems such as language translation, speech recognition, image captioning, etc. They are different from feedforward neural networks with their "memory" cells as information stored from previous inputs which also influences the current input and the output.

State System: Consider a dynamical system of the form:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}; \theta \text{ for } t \geq 1, (h_0 \text{ given})$$

where

- h_t is the hidden state at time t
- $f : \mathbb{R}^k \times \mathbb{R}^m$ (Borel Measureable)
- θ is a parameter of f and is independent of t

■ **Example 12.1** Let $f(x,\theta) = \sigma(\theta x), \theta \in \mathbb{R}, |\theta| < 1$. Consider $\theta = 0.5$:

$$\begin{aligned} h_0 &= 2 \\ h_1 &= \sigma(1) \approx 0.73 \\ h_2 &= \sigma(0.73 \times 0.5) \approx 0.59 \\ &\vdots \\ h_{100} &= 0.5708\ldots \end{aligned}$$

The reader can check that this converges to approximately 0.5708 and that this is surprisingly true for any value of h_0. ■

12.1.2 Representing a dynamical state system graphically

A dynamical system can be driven by an external signal X_t. The values of X could be the words in a sentence and can be represented by the following statement:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, X_t; \theta)$$

There are two ways to "graph" a dynamical system, the unfolded diagram on the right and the circuit diagram on the left. The circuit diagram is essentially a compressed version of the unfolded diagram.

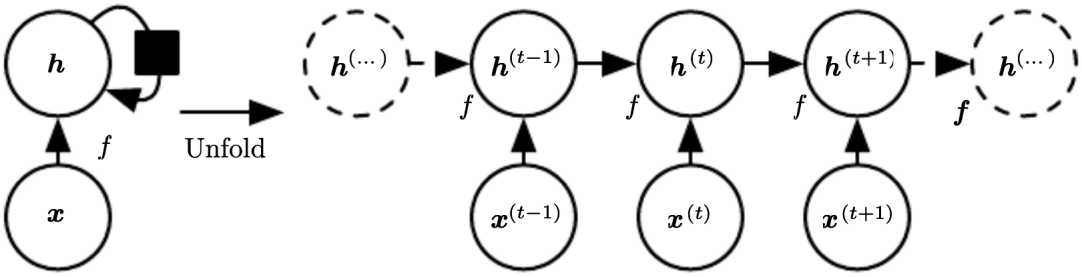


Figure 12.1: Circuit diagram (left) and Unfolded diagram (right) are two equivalent ways to represent a Recurrent Neural Network. At each time step (■), input x is incorporated into state h such that the next state h is influenced by both the current x input and previous x inputs.

12.2 Recurrent Neural Networks (RNNs)

An RNN is a state system $(h_{t-1}, X_t; \theta)$ together with a presented way to associate an output Y_t at each time step $(t > 0)$.

Notation:

- h_t : Hidden State
- X_t : t^{th} input
- Y_t : t^th output

Standard Equations

$$h_t = tanh(Wh_{t-1} + UX_t + b)$$

$$Y_t = Vh_t + C$$

$$\theta = (\mathbf{W}, \mathbf{U}, \mathbf{b}, \mathbf{c})$$

where

- $\mathbf{W}$ = hidden-to-hidden parameters
- $\mathbf{U}$ = input-to-hidden parameters
- $\mathbf{V}$ = hidden-to-output parameters (fixed)
- $\mathbf{b}, \mathbf{c}$ = bias vectors

We can also think of an RNN as a sequence of ordinary feed-forward Neural Networks each with one hidden layer as displayed by the following image:

$$\begin{pmatrix} h_0 \\ X_1 \end{pmatrix} \longrightarrow h_1 \longrightarrow Y_1$$
$$\begin{pmatrix} h_1 \\ X_2 \end{pmatrix} \longrightarrow h_2 \longrightarrow Y_2$$
$$\vdots$$

12.2.1 Loss Function

How do we actually apply a loss function to an RNN? Consider an RNN with inputs $x_1 ... x_T$ and outputs $y_1 ... y_T$ and let's consider a target $z_1 ... z_T$

A common choice for the loss function is

$$L\left[(y_1 ... y_T), (z_1 ... z_T)\right] = \sum_{t=1}^{T} \frac{1}{2} \|y_t - z_t\|_2^2$$

but one can also use Cross Entropy Loss, Binary Cross Entropy Loss, etc., depending on the situation and context. The loss function would be applied to each state and added up for the total loss.

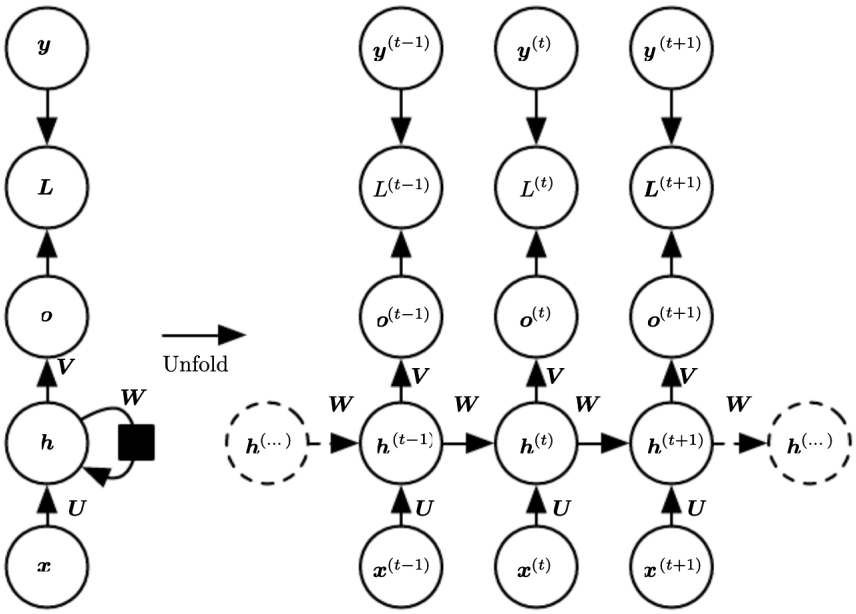


Figure 12.2: Loss is calculated between the true y values and the RNN predicted o values at each time step. If one wants a single output from the RNN, the loss can be calculated only between the last y and o values in the sequence.

■ **Example 12.2** Given a restaurant review, classify if the review is positive or negative.
Review: "It was great" ■

The RNN will take each of the words as input simultaneously and outputs a binary prediction of positive or negative.

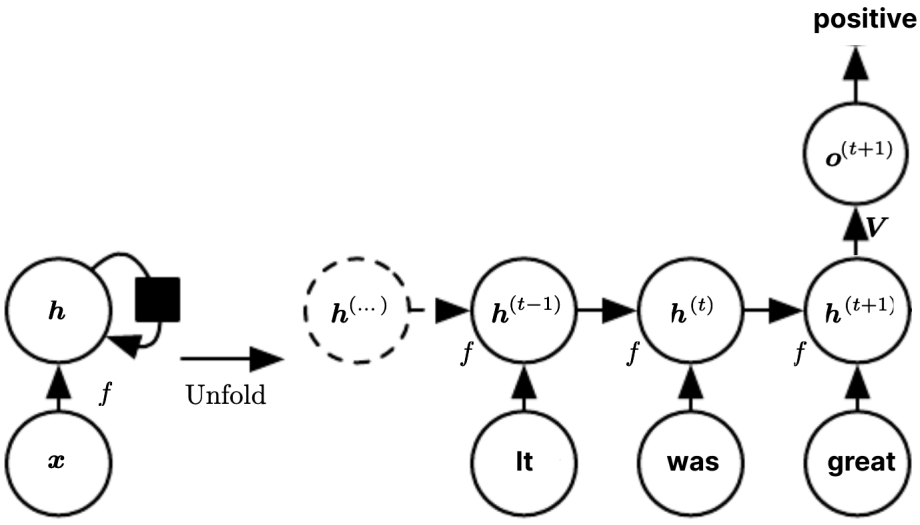
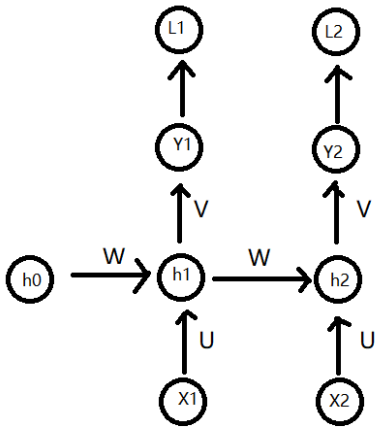


Figure 12.3: Given the sequence of words, the RNN makes a binary prediction of positive or negative.

12.3 RNN backpropagation

Now, we will be looking at RNN Back-propagation, which is called back propagation through time, where the gradients goes backward at each time steps. One can unroll the computational graph easily by applying the generalized back-propagation algorithm.

■ **Example 12.3** consider an RNN with 2 steps:



Where

$$a_1 = Wh_0 + UX_1 + b$$
$$a_2 = Wh_1 + UX_2 + b$$
$$h_1 = tanh(a_1)$$
$$h_2 = tanh(a_2)$$
$$Y_1 = Vh_1 + c$$
$$Y_2 = Vh_2 + c$$

using square error loss

$$L = \frac{1}{2}(Y_1 - Z_1)^2 + \frac{1}{2}(Y_2 - Z_2)^2 = L_1 + L_2$$

We want to compute

$$\nabla_\theta L = (\frac{\partial L}{\partial W}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial U}, \frac{\partial L}{\partial b}, \frac{\partial L}{\partial c})$$

Note that L_1 depends on W only through h_1, and L_2 depends on W through h_1 and h_2, so

$$\frac{\partial L}{\partial W} = \frac{\partial L_1}{\partial W} + \frac{\partial L_2}{\partial W} = \frac{\partial L1}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial L_2}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial L_2}{\partial h_2}\frac{\partial h_2}{\partial W}$$

$$\frac{\partial L}{\partial V} = \frac{\partial L_1}{\partial V} + \frac{\partial L_2}{\partial V} = \frac{\partial L_1}{\partial Y_1}\frac{\partial Y_1}{\partial V} + \frac{\partial L_2}{\partial Y_2}\frac{\partial Y_2}{\partial V}$$

$$\Rightarrow \frac{\partial L}{\partial V} = (Y_1 - Z_1)h_1 + (Y_2 - Z_x)h_2 = \sum_t (Y_t - Z_t)h_t$$

$$\frac{\partial L}{\partial U} = \frac{\partial L_1}{\partial U} + \frac{\partial L_2}{\partial U} = \frac{\partial L_1}{\partial h_1}\frac{\partial h_1}{\partial U} + \frac{\partial L_2}{\partial h_1}\frac{\partial h_1}{\partial U} + \frac{\partial L_2}{\partial h_2}\frac{\partial h_2}{\partial U}$$
$$\Rightarrow \frac{\partial L}{\partial U} = \frac{\partial L_1}{\partial h_1}\frac{\partial h_1}{\partial U} + \frac{\partial L_2}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial U} + \frac{\partial L_2}{\partial h_2}\frac{\partial h_2}{\partial U}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L_1}{\partial b} + \frac{\partial L_2}{\partial b} = \frac{\partial L_1}{\partial h_1}\frac{\partial h_1}{\partial b} + \frac{\partial L_2}{\partial h_1}\frac{\partial h_1}{\partial b} + \frac{\partial L_2}{\partial h_2}\frac{\partial h_2}{\partial b}$$
$$\Rightarrow \frac{\partial L}{\partial b} = \frac{\partial L_1}{\partial h_1}\frac{\partial h_1}{\partial b} + \frac{\partial L_2}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial b} + \frac{\partial L_2}{\partial h_2}\frac{\partial h_2}{\partial b}$$

$$\frac{\partial L}{\partial c} = \frac{\partial L_1}{\partial c} + \frac{\partial L_2}{\partial c} = \frac{\partial L_1}{\partial Y_1}\frac{\partial Y_1}{\partial c} + \frac{\partial L_2}{\partial Y_2}\frac{\partial Y_2}{\partial c}$$
$$\Rightarrow \frac{\partial L}{\partial c} = (Y_1 - Z_1) + (Y_2 - Z_2) = \sum_t (Y_t - Z_t)$$

With the above computed, it is suffice to compute the following:

$$\frac{\partial L_1}{\partial h_1} = \frac{\partial L_1}{\partial Y_1}\frac{\partial Y_1}{\partial h_1} = (Y_1 - Z_1)V$$

$$\frac{\partial L_2}{\partial h_2} = \frac{\partial L_2}{\partial Y_2}\frac{\partial Y_2}{\partial h_2} = (Y_2 - Z_2)V$$

$$\frac{\partial h_1}{\partial W} = \frac{\partial}{\partial W}\tanh(a_1) = sech^2(a_1)\frac{\partial a_1}{\partial W}$$
$$= (1 - \tanh^2(a_1)) = (1 - h_1^2)h_0$$

$$\frac{\partial h_2}{\partial W} = \frac{\partial}{\partial W}\tanh(a_2) = sech^2(a_2)\frac{\partial a_2}{\partial W}$$
$$= (1 - \tanh^2(a_2)) = (1 - h_2^2)h_1$$

$$\frac{\partial h_2}{\partial h_1} = \frac{\partial}{\partial h_1}\tanh(a_2) = sech^2(a_2)\frac{\partial a_2}{\partial h_1} = (1 - h_2^2)W$$

$$\frac{\partial h_1}{\partial U} = \frac{\partial}{\partial U}\tanh(a_1) = (1 - h_1^2)\frac{\partial a_1}{\partial U} = (1 - h_1^2)X_1$$

$$\frac{\partial h_2}{\partial U} = \frac{\partial}{\partial U}\tanh(a_2) = (1 - h_2^2)\frac{\partial a_2}{\partial U} = (1 - h_2^2)X_2$$

$$\frac{\partial h_1}{\partial b} = \frac{\partial}{\partial b}\tanh(a_1) = sech^2(a_1)\frac{\partial a_1}{\partial b} = (1 - h_1^2)$$

$$\frac{\partial h_2}{\partial b} = \frac{\partial}{\partial b}\tanh(a_2) = sech^2(a_2)\frac{\partial a_2}{\partial b} = (1 - h_2^2)$$

■

12.4 Exercises

1. What are Recurrent Neural Networks (RNN) used for and why can't we just use a Neural Network?
2. How is Loss calculated in an RNN?
3. Write an algorithm or pseudocode to model a state system with inputs hidden state h_0, θ, and time step T to indicate the number of iterations where $f(x,\theta) = \sigma(\theta x), \theta \in \mathbb{R}, |\theta| < 1$.

12.5 Solutions to Exercises

1. A feedforward neural network is not good at solving this kind of problem because it only can take a fixed length as input, however, a sequence has varying lengths. We need a model that is specialized in learning sequential dependencies between values in the sequence. An RNN is used for processing and handling chunks of a sequence.
2. Loss is calculated by summing each individual loss at each time step. The loss function is applied to each actual y value and the output of each cell.
3. Here is the pseudocode function for stateSystem.

Algorithm 4 $stateSystem(\mathbf{h}, theta, T)$

```
i = 0
def stateSystem(h, theta, T):
    h = h * theta
    if i == T then
        return h
```