BST 261: Data Science II

Lecture 1

Heather Mattie

Department of Biostatistics
Harvard T.H. Chan School of Public Health
Harvard University

March 19, 2018

Class Introduction

Course Overview

Instructor:

- Heather Mattie
- Instructor of Biostatistics and Data Science
- Email: hemattie@hsph.harvard.edu
- Office: Room 421A, Department of Biostatistics (Building 1)
- Office hour: TBD

Course Overview

Teaching Assistants:

- Stephanie Chan: stephaniechan@fas.harvard.edu
- Office hours: TBD
- Office hour location: TBD
- Please fill out this form with your preferred office hour days and times: `https://docs.google.com/forms/d/e/1FAIpQLSdRiqeLyEBIg7PQLdq8dOlnf3xvim1hulOmFS6NMNJxTM-EZg/viewform?usp=sf_link`

Course Overview

Course Website and Communication

- Canvas: The Canvas site is an important learning tool for this course where students will access course materials, submit course assignments and share other resources with the class. Course announcements will be posted on the site and students will be required to check the course website on a weekly basis.

- GitHub: All Jupyter notebooks used for this course (in-class examples, homework assignments, group project) are available on the course GitHub repository here: `https://github.com/hmattie/BST261Spring2018`
    - Note that assignments will not be submitted to this repository, but in Canvas.

- Slack: Students may utilize the discussion boards in Canvas, or ask questions on the course Slack workspace: `https://join.slack.com/t/bst261fall2018/signup`

Course Overview

- Course content:
  - Computational and mathematical foundations of deep learning
  - Deep neural networks
  - Deep learning research
- Most of the content is based on this excellent new book: Deep Learning. Ian Goodfellow, Yoshua Bengio, Aaron Courville. MIT Press, 2016.
- Book available at `http://www.deeplearningbook.org`
- Examples mainly from the new book: Deep Learning with Python. François Chollet. Manning Publications, 2017.
- The first few chapters are available at `https://www.manning.com/books/deep-learning-with-python`

Course Overview

Course content (cont.):

- Computational and mathematical foundations of deep learning
  - Review of Python 3
  - Scientific computation in Python
  - Linear algebra and probability
  - Machine learning
- Deep neural networks
  - Feedforward networks
  - Convolutional networks
  - Recurrent and recursive networks
- Deep learning research
  - Autoencoders
  - Approximate inference
    - Parameter inference
    - Model selection
  - Deep generative models (if time)

Course Overview

Lectures:

- Held on Mondays & Wednesdays from 9:45 - 11:15am in Kresge 200
- Slides and in-class examples will be available on the course Canvas and GitHub before each lecture
- Approximately half of the lectures will be theory and half applied
- All coding examples will be in Python and use Keras with TensorFlow backend

Course Overview

Homework assignments:

- Homework 1 due Monday April 9
- Homework 2 due Monday April 30
- Output is a Jupyter notebook consisting of code and text

Course Overview

Group project:

- Maximum of 4 people in each group
- Each group will choose a data set to use
- Preloaded data sets in Keras are not allowed
- Output is a Jupyter notebook consisting of code and text (deadline: May 14 - note this is the Monday after the semester ends)
- Group presentations on the last day of class (May 9)

Course Overview

Grading:

- 2 homework assignments (70%)
- Group project (30%)
- Credits: 2.5
- There is no exam

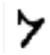
What is deep learning?

Introduction to Deep Learning

Calculate 1,293 * 2,374 * 0.43

Introduction to Deep Learning

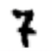
What digit is this? How do you know?



Introduction to Deep Learning

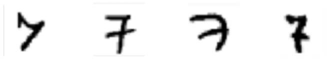
Introduction to Deep Learning

Introduction to Deep Learning



Introduction to Deep Learning

How would you tell a computer that these are all the same digit?



Introduction to Deep Learning

- Computers can solve problems that are intellectually difficult for humans
  - Ex: multiplication with large numbers and decimals, chess, etc.
  - Problems that can be described by a list of formal, mathematical rules
- Humans can solve problems that are intuitive, but difficult to describe formally and thus difficult for computers to solve
  - Ex: handwriting recognition, speech recognition, etc.
- Deep learning is a solution to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts
- The hierarchy of concepts builds on itself, producing a deep graph with many layers, leading to the concept of ***deep learning***

Introduction to Deep Learning

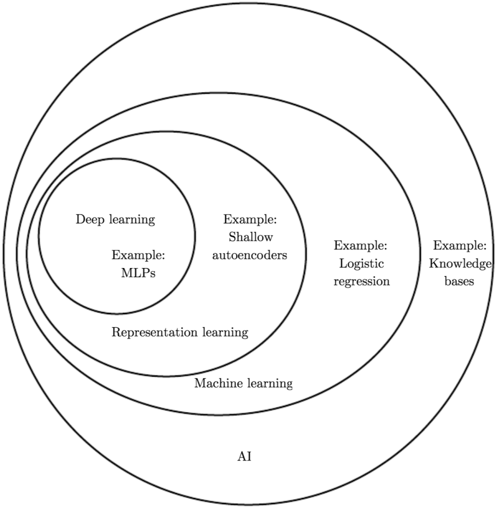


Figure: AI venn diagram

Symbolic AI vs Machine Learning

- Symbolic AI:
  - Input: data and rules
  - Output: answers
- Machine Learning
  - Input: data and answers
  - Output: rules
- A machine-learning system is *trained*, or *learned*, rather than explicitly programmed

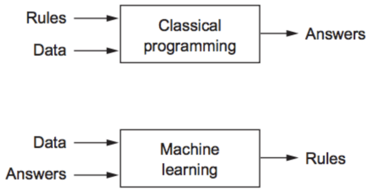


Figure: Machine learning: a new programming paradigm

Features

What number is this: CMXCIX

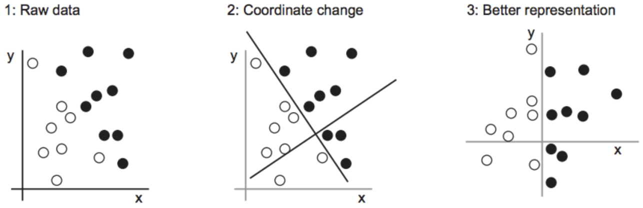
Features

What number is this: CMXCIX
And this: 999

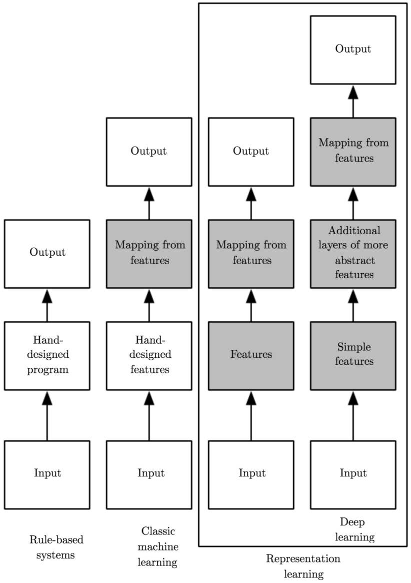
Features

What number is this: CMXCIX
And this: 999
Which was easier to read and process?

Features


1: Raw data
2: Coordinate change
3: Better representation
y
x
y
x
y
x

Workflows


Output
Mapping from features
Output
Output
Mapping from features
Mapping from features
Additional layers of more abstract features
Output
Hand-designed program
Hand-designed features
Features
Simple features
Input
Input
Input
Input
Deep learning
Rule-based systems
Classic machine learning
Representation learning

Handwritten Digit Classification

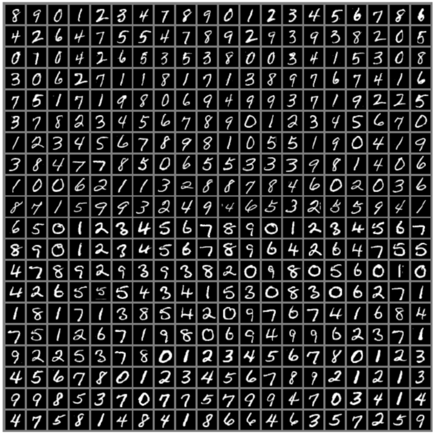
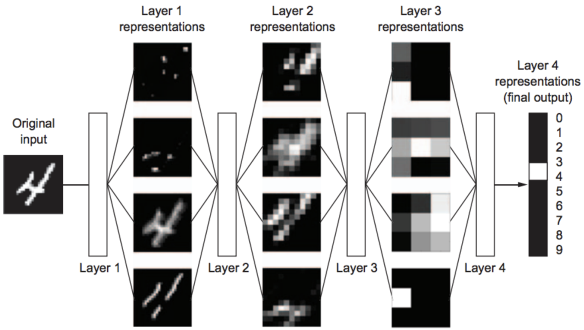
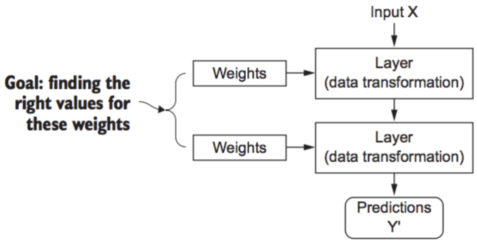


Figure: Example inputs from the MNIST dataset. The “NIST” stands for National Institute of Standards and Technology, the agency that originally collected this data.The “M” stands for “modified," since the data has been preprocessed for easier use with machine learning algorithms. The MNIST dataset consists of scans of handwritten digits and associated labels describing which digit 0-9 is contained in each image. This simple classification problem is one of the simplest and most widely used tests in deep learning research.

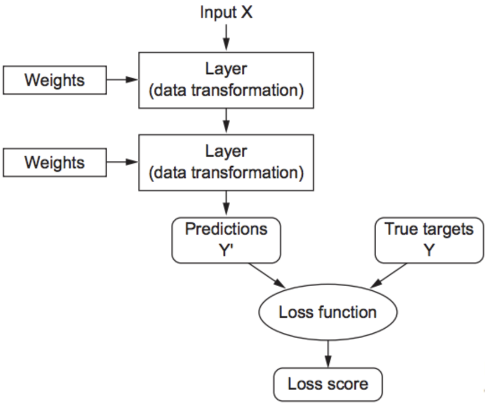
Handwritten Digit Classification



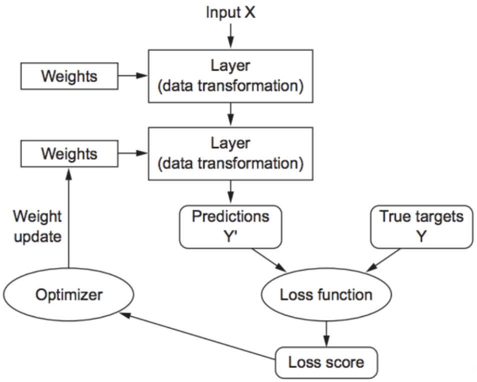
More Concrete Workflow


Input X
Goal: finding the right values for these weights
Weights
Layer (data transformation)
Weights
Layer (data transformation)
Predictions Y'

More Concrete Workflow



More Concrete Workflow



Why all the hype now?

- Hardware
- Algorithmic advances
- Data (availability and volume)

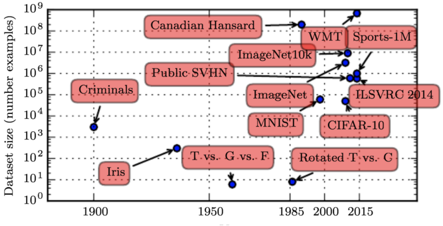


Figure: Sizes of data sets over time

More Data = Better Performance

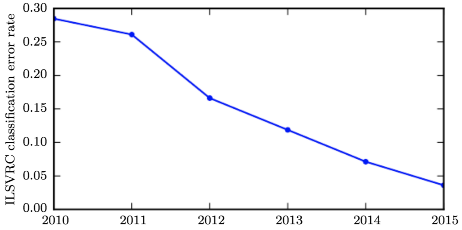


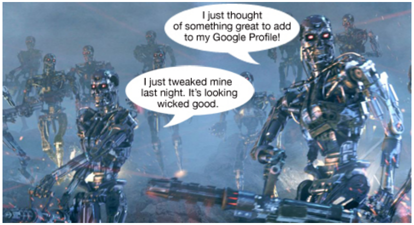
Figure: Decreasing error rate over time. Since deep networks reached the scale necessary to compete in the ImageNet Large Scale Visual Recognition Challenge, they have consistently won the competition every year, yielding lower and lower error rates each time.

Deep Learning Successes

- Near-human-level image classification
- Near-human-level speech recognition
- Near-human-level handwriting transcription
- Improved machine translation
- Improved text-to-speech conversion
- Digital assistants such as Google Now and Amazon Alexa
- Near-human-level autonomous driving
- Improved ad targeting, as used by Google, Baidu, and Bing
- Improved search results on the web
- Ability to answer natural-language questions
- Superhuman Go playing

Don't Fret

Machines won't take over the world ... in this lifetime.


I just thought of something great to add to my Google Profile!
I just tweaked mine last night. It's looking wicked good.

LOOKS LIKE COMPUTERS WILL BEAT HUMANS AT GO PRETTY SOON.
WOW.
THAT'S THE LAST OF THE BIG ONES.
YEAH.
WELL, AT LEAST HUMANS ARE STILL BETTER AT, UH,
COMING UP WITH REASSURING PARABLES ABOUT THINGS HUMANS ARE BETTER AT?
HMM.
TYPE TYPE
I MADE A PYTHON SCRIPT THAT GENERATES THOUSANDS OF REASSURING PARABLES PER SECOND.
DAMMIT.
COMPUTERS WILL NEVER UNDERSTAND A SONNET COMPUTERS WILL NEVER ENJOY A SALAD COMP—

Cautious Optimism

- "There has been a great deal of *hype* surrounding neural networks, making them seem magical and mysterious. As we make clear in this section (11.3), they are just nonlinear statistical models ... "
  - The Elements of Statistical Learning - Hastie, Tibshirani, Friedman
- "Although deep learning has led to remarkable achievements in recent years, expectations for what the field will be able to achieve in the next decade tend to run much higher than what will likely be possible. ... as technology fails to deliver, research investment will dry up, slowing progress for a long time. This has happened twice before. ..."
  - Deep Learning with Python - Chollet
- "The years ahead are full of challenges and opportunities to improve deep learning even further and to bring it to new frontiers."
  - Deep Learning - Goodfellow, Bengio, Courville

Linear Algebra

Reading: Chapter 2 Goodfellow

Scalars, Vectors, Matrices

- A **scalar** is a single number
- We use $s \in \mathbb{R}$ to denote a real-valued scalar and $n \in \mathbb{N}$ to denote a natural number scalar
- A **vector** is an array of numbers that are arranged in order
- We use $\boldsymbol{x} \in \mathbb{R}^n$ to denote a vector, where $\mathbb{R}^n$ is the set formed by taking the Cartesian product of $\mathbb{R}$ a total of n times
- Elements of vector $\boldsymbol{x}$ are identified with $x_1, x_2, \ldots$
- We can think of vectors as identifying points in space, with each element giving the coordinate along a different axis
- All vectors we deal with are column vectors (rather than row vectors) unless we specify otherwise
- We can index a set of elements of a vector by defining a set containing the indices and writing the set as a subscript
- For example we can define $S = \{1, 3, 6\}$ and then write $\boldsymbol{x}_S$
- We use the $-$ sign to index the complement of a set, for example, $\boldsymbol{x}_{-S}$ is the vector containing all elements of $\boldsymbol{x}$ except for x_1, x_3, and x_6

Scalars, Vectors, Matrices

- A **matrix** is a 2-D array of numbers
- We use $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ to denote a real-valued matrix $\boldsymbol{A}$ of m rows and n columns
- We identify the elements of a matrix using its name and the indices, for example, $A_{1,1}$ is the entry in the first row and first column of $\boldsymbol{A}$

$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \tag{1}$$

- We can identify all values of a given index using :
- For example, $\boldsymbol{A}_{i,:}$ denotes row i of $\boldsymbol{A}$ and $\boldsymbol{A}_{:,j}$ denotes column j of $\boldsymbol{A}$
- We use subscripts to index matrix-valued expressions, for example, $f(\boldsymbol{A})_{i,j}$ gives element (i, j) of the matrix computed by applying the function f to $\boldsymbol{A}$
- **Tensors** are multidimensional arrays
- We use $A_{i,j,k}$ to identify element (i, j, k) of $\boldsymbol{A}$

Scalars, Vectors, Matrices

- Let $\boldsymbol{A}$ be an $m \times n$ matrix
- The $n \times m$ matrix $\boldsymbol{B}$ with elements $B_{j,i} = A_{i,j}$ is called the **transpose** of $\boldsymbol{A}$ and it is denoted by $\boldsymbol{B} = \boldsymbol{A}^{\mathrm{T}}$
- Taking the transpose of a matrix amounts to changing rows into columns and vice versa
- The transpose of a matrix is the mirror image of the matrix across a diagonal line, called the **main diagonal**, which runs from the top left element to the bottom right element
- Vectors can the thought of as matrices that contain only one column, and the transpose of a column vector is a row vector and vice versa
- A scalar can be thought of as a matrix with only a single entry, which is its own transpose: $a = a^{\mathrm{T}}$

Scalars, Vectors, Matrices

- Matrices that have the same number of rows and the same number of columns are said to be **conformal** for addition or subtraction
- Matrix addition and subtraction are defined for any two conformal matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, and these operations are performed element-wise: $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$ where $C_{i,j} = A_{i,j} + B_{i,j}$
- Matrix addition is commutative, that is, $\boldsymbol{A} + \boldsymbol{B} = \boldsymbol{B} + \boldsymbol{A}$
- A matrix can be multiplied by a scalar by performing that operation on each element of the matrix: $\boldsymbol{D} = a \cdot \boldsymbol{B}$ where $D_{i,j} = a \cdot B_{i,j}$
- Additionally some non-conventional shorthand notation is occasionally used:
    - Matrix $+$ scalar: $\boldsymbol{D} = \boldsymbol{B} + c$ where $D_{i,j} = B_{i,j} + c$

Multiplying Matrices and Vectors

- The **matrix product** of an $m \times n$ matrix $\boldsymbol{A}$ and an $n \times p$ matrix $\boldsymbol{B}$ is the $m \times p$ matrix $\boldsymbol{C}$, where the product operation is defined by

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j} \tag{2}$$

- We write this as $\boldsymbol{AB} = \boldsymbol{C}$ and refer to it as pre-multiplication of $\boldsymbol{B}$ by $\boldsymbol{A}$ or post-multiplication of $\boldsymbol{A}$ by $\boldsymbol{B}$
- The **element-wise product**, also known as the **Hadamard product**, of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ by is denoted by $\boldsymbol{A} \odot \boldsymbol{B}$
- The **dot product** between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of the same dimensionality is the matrix product $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$
- We can therefore think of the matrix product $\boldsymbol{C} = \boldsymbol{AB}$ as computing $C_{i,j}$ as the dot product between row i of $\boldsymbol{A}$ and column j of $\mathbf{B}$

Multiplying Matrices and Vectors

- Matrix multiplication is distributive:
$$\boldsymbol{A}(\boldsymbol{B}+\boldsymbol{C}) = \boldsymbol{A}\boldsymbol{B}+\boldsymbol{A}\boldsymbol{C} \tag{3}$$
- Matrix multiplication is associative:
$$\boldsymbol{A}(\boldsymbol{B}\boldsymbol{C}) = (\boldsymbol{A}\boldsymbol{B})\boldsymbol{C} \tag{4}$$
- Matrix multiplication is generally not commutative:
$$\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A} \tag{5}$$
- However the dot product between two vectors is commutative:
$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{x} \tag{6}$$
- The transpose of a matrix product:
$$(\boldsymbol{A}\boldsymbol{B})^{\mathrm{T}} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} \tag{7}$$

Multiplying Matrices and Vectors

- We can write down a system of linear equations:

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{8}$$

- Here $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is a known matrix, $\boldsymbol{b} \in \mathbb{R}^m$ is a known vector, and $x \in \mathbb{R}^n$ is an unknown vector we would like to solve for
- This can be rewritten as

$$\begin{aligned} A_{1,1}x_1 + A_{1,2}x_2 + \cdots + A_{1,n}x_n &= b_1 \\ A_{2,1}x_1 + A_{2,2}x_2 + \cdots + A_{2,n}x_n &= b_2 \\ \cdots & \\ A_{m,1}x_1 + A_{m,2}x_2 + \cdots + A_{m,n}x_n &= b_m \end{aligned} \tag{9}$$

Identity and Inverse Matrices

- The **identity matrix** is a matrix that does not change a vector when multiplied by it
- We denote the identity matrix that preserves n-dimensional vectors as $\boldsymbol{I}_n$, where $\boldsymbol{I} \in \mathbb{R}^{n \times n}$ and

$$\boldsymbol{I}_n \boldsymbol{x} = \boldsymbol{x} \; \forall \boldsymbol{x} \in \mathbb{R}^n \tag{10}$$

- In an identity matrix, all entries along the main diagonal are 1, while all the other entries are 0

$$\boldsymbol{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

- The **matrix inverse** of $\boldsymbol{A}$ is denoted as $\boldsymbol{A}^{-1}$ and defined as the matrix such that

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}_n \tag{12}$$

- This enables us to solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ as $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$

Identity and Inverse Matrices

- The inverse $\boldsymbol{A}^{-1}$ does not always exist, but when it does, several different algorithms can find it in closed form
- The obtained inverse matrix $\boldsymbol{A}^{-1}$ can then be used to solve the equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ for different values of $\boldsymbol{b}$
- In practice, due to the limited precision of a digital computer, the inverse matrix $\boldsymbol{A}^{-1}$ is rarely used in practice for most software applications
- Instead, algorithms that make use of the value of $\boldsymbol{b}$ can usually obtain numerically more accurate estimates of $\boldsymbol{x}$

Linear Dependence and Span

- A **linear combination** of some set of vectors $\{\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(n)}\}$ is given by multiplying each vector $\boldsymbol{v}^{(i)}$ by a corresponding scalar coefficient c_i and adding the results together

$$\sum_i c_i \boldsymbol{v}^{(i)} \tag{13}$$

- The **span** of a set of vectors $\{\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(n)}\}$ is the set of all points obtainable by their linear combination $\sum_i c_i \boldsymbol{v}^{(i)}$
- A system of linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ has either no solutions or has infinitely many solutions (the only two possibilities) for a given $\boldsymbol{b}$
- We can think of the columns of $\boldsymbol{A}$ as column vectors that specify different directions we can travel in from the origin, and we can think of each element of $\boldsymbol{x}$ as specifying how far we travel in each of these directions

$$\boldsymbol{A}\boldsymbol{x} = \sum_i x_i \boldsymbol{A}_{:,i} \tag{14}$$

Linear Dependence and Span

- Determining the number of solutions is then equivalent to determining the number of ways in which we can combine the columns of $\boldsymbol{A}$ to reach $\boldsymbol{b}$
- Put differently, determining whether $\boldsymbol{Ax} = \boldsymbol{b}$ has a solution amounts to testing whether $\boldsymbol{b}$ is in the span of the columns of $\boldsymbol{A}$, where the span of the columns of $\boldsymbol{A}$ is known as the **column space** or the **range** of $\boldsymbol{A}$
- In order for the system $\boldsymbol{Ax} = \boldsymbol{b}$ to have a solution for all values of $\boldsymbol{b} \in \mathbb{R}^m$, we require that the column space of $\boldsymbol{A}$ be all of $\mathbb{R}^m$
- If any point in $\mathbb{R}^m$ is excluded from the column space, that point is a potential value of $\boldsymbol{b}$ that has no solution
- The requirement that the column space of $\boldsymbol{A}$ be all of $\mathbb{R}^m$ implies immediately that $\boldsymbol{A}$ must have at least m columns
- For example, for a 3×2 matrix $\boldsymbol{A}$, the target $\boldsymbol{b}$ is in 3 dimensions but $\mathbf{x}$ is only in 2 dimensions, so modifying the value of $\boldsymbol{x}$ at best enables us to trace out a 2-D plane within $\mathbb{R}^3$; the equation has a solution if and only if $\boldsymbol{b}$ lies on that plane

Linear Dependence and Span

- A set of vectors is **linearly independent** if no vector in the set is a linear combination of the other vectors
- This means that for the column space of $\boldsymbol{A}$ to encompass all of $\mathbb{R}^m$, the matrix must contain at least one set of m linearly independent columns, which is both necessary and sufficient for $\boldsymbol{Ax} = \boldsymbol{b}$ to have a solution for every value of $\boldsymbol{b}$
- The requirement is for exactly m linearly independent columns; no set of m-dimensional vectors can have more than m linearly independent columns, but a matrix with more than m columns may have more than one such set
- For $\boldsymbol{A}$ to have an inverse, we additionally need to ensure that $\boldsymbol{Ax} = \boldsymbol{b}$ has at most one solution for each value of $\boldsymbol{b}$, meaning that the matrix has at most m columns
- Taken together, this means that the matrix must be square and all the columns must be linearly independent
- A square matrix with linearly dependent columns is called **singular**
- If $\boldsymbol{A}$ is not square, or is square but singular, solving the equation is still possible, but we cannot use matrix inversion to find the solution

Norms

- We often need to measure the size of a vector
- This is done using a function called a **norm**, which maps a vector to a non-negative value
- The L^p norm, for $p \in \mathbb{R}$, $p \geq 1$, is given by

$$\|\boldsymbol{x}\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}} \tag{15}$$

- The L^2 norm is known as the **Euclidean norm**, which is simply the Euclidean distance from the origin to the point identified by $\boldsymbol{x}$
- Since this norm is used so frequently, it is sometimes denoted as $\|\boldsymbol{x}\|$ with the subscript 2 omitted
- The squared L^2 norm is more convenient to work with mathematically and computationally than the L^2 norm itself, and it can be calculated simply as $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$

Norms

- One other norm that arises is the L^∞ norm, also known as the **max norm**, which simplifies to the absolute value of the element with the largest magnitude in the vector:

$$\|\boldsymbol{x}\|_\infty = \max_i |x_i| \tag{16}$$

- The most commonly used matrix norm is the **Frobenius norm**

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} A_{i,j}^2} \tag{17}$$

Special Matrices and Vectors

- **Diagonal** matrices consist mostly of zeros and have nonzero entries only along the main diagonal
- We write $\mathrm{diag}(\boldsymbol{c})$ to denote a square diagonal matrix whose diagonal entries are given by the entries of the vector $\boldsymbol{v}$
- Diagonal matrices are of interest in part because multiplying by a diagonal matrix is computationally efficient
- For example, $\mathrm{diag}(\boldsymbol{v})\boldsymbol{x} = \boldsymbol{v} \odot \boldsymbol{x}$
- The inverse of a square diagonal matrix exists only if every diagonal entry is nonzero, and in that case $\mathrm{diag}(\boldsymbol{v})^{-1} = \mathrm{diag}([1/v_1, \ldots, 1/v_n]^{\mathrm{T}})$
- Not all diagonal matrices need to be square, i.e., we can have a rectangular diagonal matrix
- Non-square diagonal matrices do not have inverses, but we can still multiply them computationally efficiently

Special Matrices and Vectors

- A **symmetric matrix** is any matrix that is equal to its own transpose: $\boldsymbol{A} = \boldsymbol{A}^{\mathrm{T}}$
- Distance matrices are symmetric because distance functions are symmetric
- A **unit vector** is a vector with **unit norm**:

$$\|\boldsymbol{x}\|_2 = 1 \tag{18}$$

- Vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are **orthogonal** to each other if $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = 0$
- If both vectors have nonzero norm, this means that they are at a 90 degree angle to each other
- If the vectors are orthogonal and also have unit norm, we call them **orthonormal**

Special Matrices and Vectors

- An **orthogonal matrix** is a square matrix whose rows are mutually orthonormal and whose columns are mutually orthonormal (counterintuitively, not merely orthogonal!)

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} = \boldsymbol{I} \tag{19}$$

- This implies that

$$\boldsymbol{A}^{-1} = \boldsymbol{A}^{\mathrm{T}} \tag{20}$$

- Inverses of orthogonal matrices are very cheap to compute

Eigendecomposition

- Many mathematical objects can be understood better by breaking them into constituents parts (e.g., decomposing integers into prime factors)
- An **eigenvector** of a square matrix $\boldsymbol{A}$ is a nonzero vector $\boldsymbol{v}$ such that multiplication by $\boldsymbol{A}$ alters only the scale of $\boldsymbol{v}$:

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v} \tag{21}$$

- Here the scalar λ is the **eigenvalue** corresponding to eigenvector $\boldsymbol{v}$
- Since a vector obtained by rescaling $\boldsymbol{v}$ is also an eigenvector, we usually look for unit eigenvectors

Eigendecomposition

- Suppose that a matrix $\boldsymbol{A}$ has n linearly independent eigenvectors $\{\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(n)}\}$ with corresponding eigenvalues $\{\lambda_1, \dots, \lambda_n\}$
- We can concatenate all the eigenvectors to form a matrix $\boldsymbol{V}$ with one eigenvector per column, and we can concatenate the eigenvalues to form a vector $\boldsymbol{\lambda}$
- The **eigendecomposition** of $\boldsymbol{A}$ is then given by

$$\boldsymbol{A} = \boldsymbol{V}\operatorname{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1} \tag{22}$$

- Not every matrix can be decomposed into eigenvalues and eigenvectors
- Every real symmetric matrix can be decomposed using only real-valued eigenvectors and eigenvalues

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\mathrm{T}} \tag{23}$$

- Here $\boldsymbol{Q}$ is an orthogonal matrix composed of eigenvectors of $\boldsymbol{A}$ and $\boldsymbol{\Lambda}$ is a diagonal matrix, where the eigenvalue $\boldsymbol{\Lambda}_{i,i}$ is associated with the eigenvector in column i of $\boldsymbol{Q}$

Matrix Cookbook

- Contains literally every matrix operation you can think of and can come in very handy:
- `http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf`