Linear Algebra for Machine Learning

Importance of Linear Algebra in ML

But what is RIGHT? And is that enough? (Image: Machine Learning, XKCD)

Topics in Linear Algebra for ML

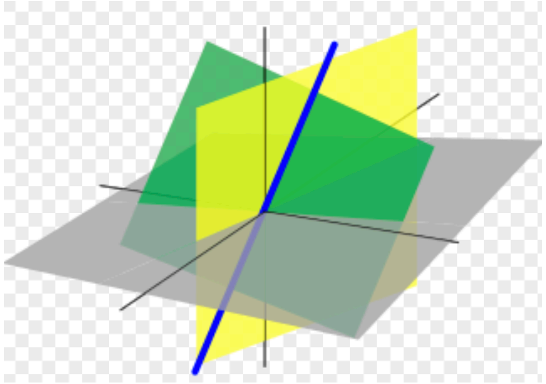
- Why do we need Linear Algebra?
- From scalars to tensors
- Flow of tensors in ML
- Matrix operations: determinant, inverse
- Eigen values and eigen vectors
- Singular Value Decomposition
- Principal components analysis

What is linear algebra?

- Linear algebra is the branch of mathematics concerning linear equations such as

$$a_1x_1+\ldots..+a_nx_n=b$$

  - In vector notation we say $\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}=b$
  - Called a linear transformation of $\boldsymbol{x}$
- Linear algebra is fundamental to geometry, for defining objects such as lines, planes, rotations



Linear equation $a_1x_1+\ldots..+a_nx_n=b$ defines a plane in $(x_1,..,x_n)$ space
Straight lines define common solutions to equations

Why do we need to know it?

- Linear Algebra is used throughout engineering
  - Because it is based on continuous math rather than discrete math
    - Computer scientists have little experience with it
- Essential for understanding ML algorithms
  - E.g., We convert input vectors $(x_1,..,x_n)$ into outputs by a series of linear transformations
- Here we discuss:
  - Concepts of linear algebra needed for ML
  - Omit other aspects of linear algebra

Linear Algebra Topics

- Scalars, Vectors, Matrices and Tensors
- Multiplying Matrices and Vectors
- Identity and Inverse Matrices
- Linear Dependence and Span
- Norms
- Special kinds of matrices and vectors
- Eigendecomposition
- Singular value decomposition
- The Moore Penrose pseudoinverse
- The trace operator
- The determinant
- Ex: principal components analysis

Scalar

- Single number
  - In contrast to other objects in linear algebra, which are usually arrays of numbers
- Represented in lower-case italic x
  - They can be real-valued or be integers
    - E.g., let $x \in \mathbb{R}$ be the slope of the line
      - Defining a real-valued scalar
    - E.g., let $n \in \mathbb{N}$ be the number of units
      - Defining a natural number scalar

Vector

- An array of numbers arranged in order
- Each no. identified by an index
- Written in lower-case bold such as $\boldsymbol{x}$
  - its elements are in italics lower case, subscripted

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

- If each element is in R then $\boldsymbol{x}$ is in R^n
- We can think of vectors as points in space
  - Each element gives coordinate along an axis

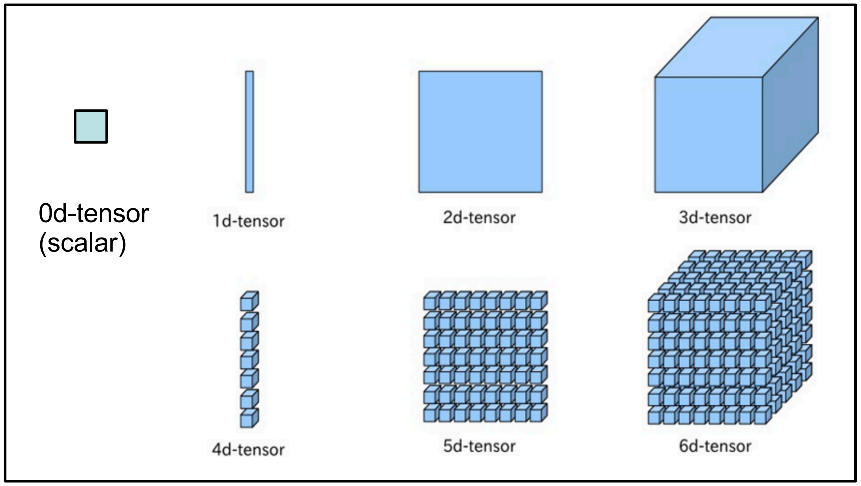
Matrices

- 2-D array of numbers
  - So each element identified by two indices
- Denoted by bold typeface $\boldsymbol{A}$
  - Elements indicated by name in italic but not bold
    - $A_{1,1}$ is the top left entry and $A_{m,n}$ is the bottom right entry
    - We can identify nos in vertical column *j* by writing : for the horizontal coordinate
    - E.g., $\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$
    - $A_{i:}$ is i^{th} row of A, $A_{:j}$ is j^{th} column of $\boldsymbol{A}$
- If A has shape of height m and width n with real-values then $\boldsymbol{A} \in \mathbb{R}^{m \times n}$

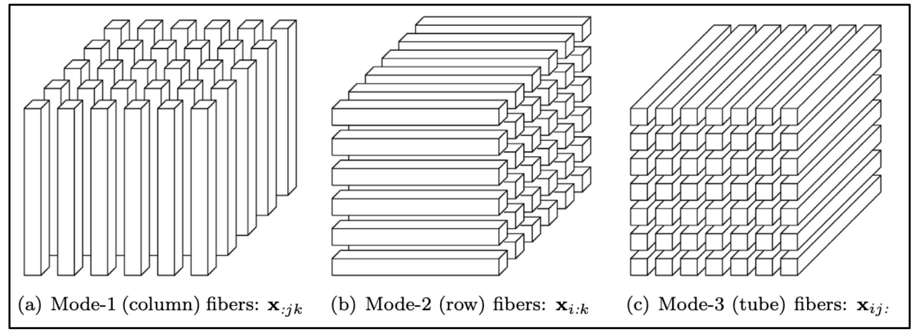
Tensor

- Sometimes need an array with more than two axes
  - E.g., an RGB color image has three axes
- A tensor is an array of numbers arranged on a regular grid with variable number of axes
  - See figure next
- Denote a tensor with this bold typeface: $\mathbf{A}$
- Element (i,j,k) of tensor denoted by $\mathbf{A}_{i,j,k}$

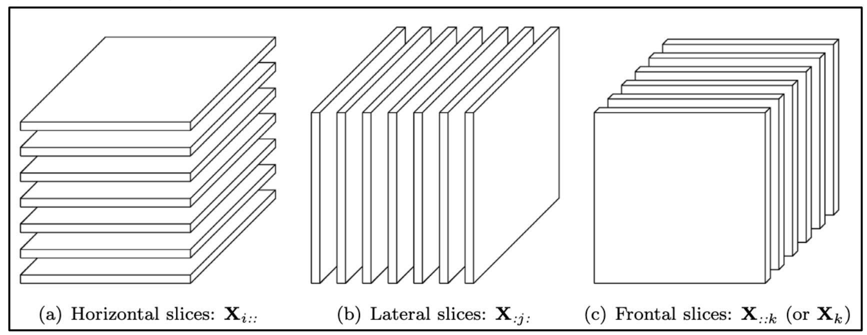
Dimensions of Tensors



Fibers of a 3rd order tensor



Slices of a 3rd order tensor



1	5	2	7	11	24	25	12

One dimensional Tensor

Collection of one dimensional tensors gives two dimensional tensor

1	5	2	7	11	24	25	12
2	3	35	7	14	0	2	15
5	25	3	1	13	28	3	16

Two dimensional tensor

Collection of two dimensional tensors gives two dimensional tensor

1	5	2	7	11	24	25	12
2	3	35	7	14	0	2	15
5	25	3	1	13	28	3	16

1	5	2	7	11	24	25	12
2	3	35	7	14	0	2	15
5	25	3	1	13	28	3	16

Three dimensional tensor

Numpy library in Python for tensors

– Zero-dimensional tensor

- import numpy as np
  x = np.array(100)
  print("Array:", x)
  print("Dimension:", x.ndim)
- **Output**
  Array: 100
  Dimension 0

– One-dimensional tensor

- import numpy as np
  x = np.array([1,5,2,7,11,24,25,12])
  print("Array:", x)
  print("Dimension:", x.ndim)
- **Output**
  Array: [1 5 2 7 11 24 25 12]
  Dimension 1

– Two-dimensional tensor

- import numpy as np
  x = np.array(
  [
  [1,5,2,7,11,24,25,12],
  [1,2,3,4,5,6,7,8]
  ]
)
- print("Array:", x)
  print("Dimension:", x.ndim)
- **Output**
  Array: [[1 5 2 7 11 24 25 12] [1 2 3 4 5 6 7 8]]

  Dimension 2

Transpose of a Matrix

- An important operation on matrices
- The transpose of a matrix $\boldsymbol{A}$ is denoted as $\boldsymbol{A}^{\mathrm{T}}$
- Defined as

$$(\mathbf{A}^{\mathrm{T}})_{i,j}=A_{j,i}$$

  – The mirror image across a diagonal line
    - Called the main diagonal , running down to the right starting from upper left corner

$$A=\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \Rightarrow A^T=\begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \\ A_{1,3} & A_{2,3} & A_{3,3} \end{bmatrix}$$

$$A=\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow A^T=\begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

Vectors as special case of matrix

- Vectors are matrices with a single column
- Often written in-line using transpose

$$\boldsymbol{x} = [x_1,..,x_n]^{\mathrm{T}}$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \\ \\ x_n \end{bmatrix} \Rightarrow \boldsymbol{x}^T = \left[x_1, x_2, ..x_n\right]$$

- A scalar is a matrix with one element

$$a = a^{\mathrm{T}}$$

Matrix Addition

- We can add matrices to each other if they have the same shape, by adding corresponding elements
  - If A and B have same shape (height m, width n)

$$C = A + B \Rightarrow C_{i,j} = A_{i,j} + B_{i,j}$$

- A scalar can be added to a matrix or multiplied by a scalar $D = aB + c \Rightarrow D_{i,j} = aB_{i,j} + c$
- Less conventional notation used in ML:
  - Vector added to matrix $C = A + \boldsymbol{b} \Rightarrow C_{i,j} = A_{i,j} + b_j$
    - Called broadcasting since vector $\boldsymbol{b}$ added to each row of A

Multiplying Matrices

- For product $\boldsymbol{C}=\boldsymbol{AB}$ to be defined, $\boldsymbol{A}$ has to have the same no. of columns as the no. of rows of $\boldsymbol{B}$
  - If $\boldsymbol{A}$ is of shape mxn and B is of shape nxp then *matrix product* $\boldsymbol{C}$ is of shape mxp

$$C = AB \Rightarrow C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

  - Note that the standard product of two matrices is not just the product of two individual elements
    - Such a product does exist and is called the element-wise product or the Hadamard product $\boldsymbol{A}\odot\boldsymbol{B}$

Multiplying Vectors

- Dot product between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of same dimensionality is the matrix product $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$
- We can think of matrix product $C=AB$ as computing C_{ij} the dot product of row i of A and column j of B

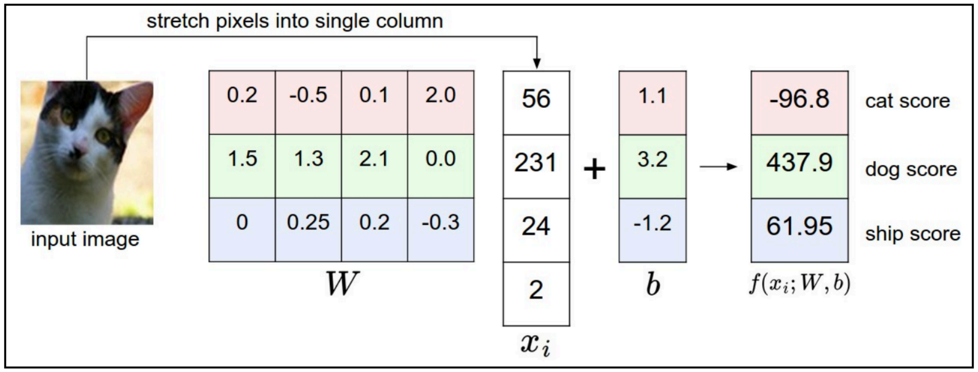
Matrix Product Properties

- Distributivity over addition: $A(B+C)=AB+AC$
- Associativity: $A(BC)=(AB)C$
- Not commutative: $AB=BA$ is not always true
- Dot product between vectors is commutative: $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}=\boldsymbol{y}^{\mathrm{T}}\boldsymbol{x}$
- Transpose of a matrix product has a simple form: $(AB)^{\mathrm{T}}=B^{\mathrm{T}}A^{\mathrm{T}}$

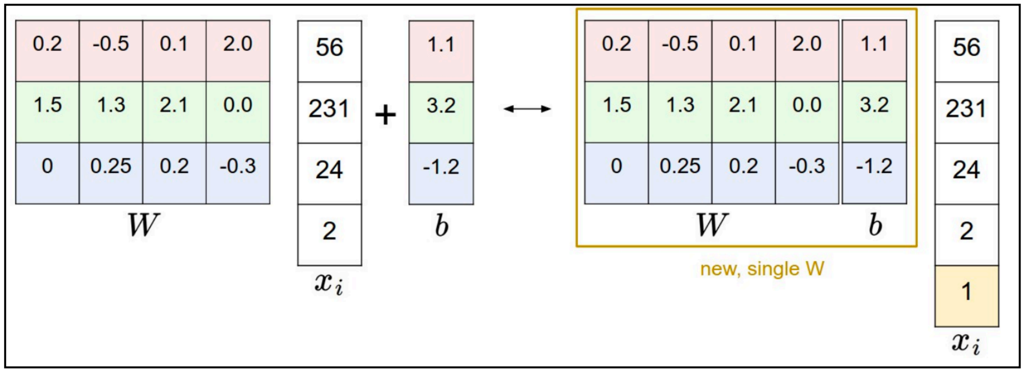
Example flow of tensors in ML

Vector $\boldsymbol{x}$ is converted into vector $\boldsymbol{y}$ by multiplying $\boldsymbol{x}$ by a matrix $\boldsymbol{W}$

A linear classifier $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}^{\mathrm{T}} + \boldsymbol{b}$



A linear classifier with bias eliminated $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}^{\mathrm{T}}$



Linear Transformation

- $\boldsymbol{Ax}=\boldsymbol{b}$
  - where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{b} \in \mathbb{R}^{n}$
  - More explicitly

$$
\begin{aligned}
A_{11}x_1 + A_{12}x_2 + ... + A_{1n}x_n &= b_1 \\
A_{21}x_1 + A_{22}x_2 + ... + A_{2n}x_n &= b_2 \\
\\
A_{n1}x_1 + A_{m2}x_2 + ... + A_{n,n}x_n &= b_n
\end{aligned}
$$

n equations in
n unknowns

$$
A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \vdots & \vdots \\ A_{n,1} & \cdots & A_{nn} \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}
$$

n x n n x 1 n x 1

Can view A as a *linear transformation* of vector $\boldsymbol{x}$ to vector $\boldsymbol{b}$

- Sometimes we wish to solve for the unknowns $\boldsymbol{x}=\{x_1,..,x_n\}$ when A and $\boldsymbol{b}$ provide constraints

Identity and Inverse Matrices

- Matrix inversion is a powerful tool to analytically solve $A\boldsymbol{x}=\boldsymbol{b}$
- Needs concept of Identity matrix
- Identity matrix does not change value of vector when we multiply the vector by identity matrix
  - Denote identity matrix that preserves n-dimensional vectors as I_n
  - Formally $I_n \in \mathbb{R}^{n\times n}$ and $\forall \mathbf{x} \in \mathbb{R}^n, I_n\mathbf{x} = \mathbf{x}$
  - Example of I_3 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Matrix Inverse

- Inverse of square matrix A defined as $A^{-1}A = I_n$
- We can now solve $A\boldsymbol{x}=\boldsymbol{b}$ as follows:

$$A\boldsymbol{x} = \boldsymbol{b}$$
$$A^{-1}A\boldsymbol{x} = A^{-1}\boldsymbol{b}$$
$$I_n\boldsymbol{x} = A^{-1}\boldsymbol{b}$$
$$\boldsymbol{x} = A^{-1}\boldsymbol{b}$$

- This depends on being able to find A^{-1}
- If A^{-1} exists there are several methods for finding it

Solving Simultaneous equations

- $\boldsymbol{Ax} = \boldsymbol{b}$

  where $\boldsymbol{A}$ is $(M+1) \times (M+1)$

  $\boldsymbol{x}$ is $(M+1) \times 1$: set of weights to be determined

  $\boldsymbol{b}$ is $N \times 1$

Example: System of Linear Equations in Linear Regression

- Instead of ***Ax*=*b***
- We have $\boxed{\Phi \mathbf{w} = \mathbf{t}}$
  - where Φ is *m* x *n* design matrix of m features for *n* samples $\boldsymbol{x}_j$, j=1,..n
  - $\boldsymbol{w}$ is weight vector of *m* values
  - ***t*** is target values of sample, $\boldsymbol{t}=[t_1,..t_n]$
  - We need weight ***w*** to be used with *m* features to determine output

$$\boxed{y(\boldsymbol{x},\boldsymbol{w})=\sum_{i=1}^{m} w_i x_i}$$

Closed-form solutions

- Two closed-form solutions
  1. Matrix inversion $\boldsymbol{x}=A^{-1}\boldsymbol{b}$
  2. Gaussian elimination

Linear Equations: Closed-Form Solutions

1. Matrix Formulation: ***Ax=b***
Solution: ***x=A***$^{-1}$***b***

$$\begin{aligned} a_{11}x_1 + \; a_{12}x_2 + \cdots + \; a_{1n}x_n &= b_1 \\ a_{21}x_1 + \; a_{22}x_2 + \cdots + \; a_{2n}x_n &= b_2 \\ \vdots \qquad \vdots \qquad\qquad \vdots \quad &\;\; \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

2. Gaussian Elimination followed by back-substitution

$$\begin{aligned} x + 3y - 2z &= 5 \\ 3x + 5y + 6z &= 7 \\ 2x + 4y + 3z &= 8 \end{aligned}$$

$L_2 - 3L_1 \rightarrow L_2$ $\quad L_3 - 2L_1 \rightarrow L_3$ $\quad -L_2/4 \rightarrow L_2$

$$\left[\begin{array}{ccc|c} 1 & 3 & -2 & 5 \\ 3 & 5 & 6 & 7 \\ 2 & 4 & 3 & 8 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 3 & -2 & 5 \\ 0 & -4 & 12 & -8 \\ 2 & 4 & 3 & 8 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 3 & -2 & 5 \\ 0 & -4 & 12 & -8 \\ 0 & -2 & 7 & -2 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 3 & -2 & 5 \\ 0 & 1 & -3 & 2 \\ 0 & -2 & 7 & -2 \end{array}\right]$$

$$\sim \left[\begin{array}{ccc|c} 1 & 3 & -2 & 5 \\ 0 & 1 & -3 & 2 \\ 0 & 0 & 1 & 2 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 3 & -2 & 5 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & 2 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 3 & 0 & 9 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & 2 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 0 & 0 & -15 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

Disadvantage of closed-form solutions

- If A^{-1} exists, the same A^{-1} can be used for any given ***b***
  - But A^{-1} cannot be represented with sufficient precision
  - It is not used in practice
- Gaussian elimination also has disadvantages
  - numerical instability (division by small no.)
  - $O(n^3)$ for n x n matrix
- Software solutions use value of ***b*** in finding ***x***
  - E.g., difference (derivative) between ***b*** and output is used iteratively

How many solutions for *Ax=b* exist?

- System of equations with
  - n variables and m equations is:

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + ... + A_{1n}x_n &= b_1 \\ A_{21}x_1 + A_{22}x_2 + ... + A_{2n}x_n &= b_2 \\ A_{m1}x_1 + A_{m2}x_2 + ... + A_{mn}x_n &= b_m \end{aligned}$$

- Solution is $\boldsymbol{x}=A^{-1}\boldsymbol{b}$
- In order for A^{-1} to exist $A\boldsymbol{x}=\boldsymbol{b}$ must have *exactly* one solution for every value of $\boldsymbol{b}$
  - It is also possible for the system of equations to have *no solutions* or an *infinite no. of solutions* for some values of $\boldsymbol{b}$
    - It is not possible to have more than one but fewer than infinitely many solutions
  - If $\boldsymbol{x}$ and $\boldsymbol{y}$ are solutions then $z=\alpha\,\boldsymbol{x} + (1-\alpha)\,\boldsymbol{y}$ is a solution for any real α

Span of a set of vectors

- Span of a set of vectors: set of points obtained by a *linear combination* of those vectors
  - A linear combination of vectors $\{\boldsymbol{v}^{(1)},..,\boldsymbol{v}^{(n)}\}$ with coefficients c_i is $\sum_i c_i \boldsymbol{v}^{(i)}$
  - System of equations is $\boldsymbol{Ax}=\boldsymbol{b}$
    - A column of $\boldsymbol{A}$, i.e., $A_{:i}$ specifies travel in direction i
    - How much we need to travel is given by x_i
    - This is a linear combination of vectors $\boldsymbol{Ax} = \sum_i x_i A_{:,i}$
  - Thus determining whether $\boldsymbol{Ax}=\boldsymbol{b}$ has a solution is equivalent to determining whether $\boldsymbol{b}$ is in the span of columns of $\boldsymbol{A}$
    - This span is referred to as *column space* or *range* of $\boldsymbol{A}$

Conditions for a solution to ***Ax=b***

- Matrix must be square, i.e., *m=n* and all columns must be *linearly independent*
  - Necessary condition is $n \geq m$
    - For a solution to exist when $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ we require the column space be all of $\mathbb{R}^m$
  - Sufficient Condition
    - If columns are linear combinations of other columns, column space is less than $\mathbb{R}^m$
      - Columns are linearly dependent or matrix is *singular*
    - For column space to encompass $\mathbb{R}^m$ at least one set of m *linearly independent* columns
- For non-square and singular matrices
  - Methods other than matrix inversion are used

Use of a Vector in Regression

- A design matrix
  - N samples, D features

	# hours studied	# hours playing games	# classes missed
Student #1	10	3	0
Student #2	8	20	2
Student #3	5	1	5

→

Grade
87
75
63

    - Feature vector has three dimensions
- This is a regression problem

Norms

- Used for measuring the size of a vector
- Norms map vectors to non-negative values
- Norm of vector $\boldsymbol{x} = [x_1,..,x_n]^T$ is distance from origin to $\boldsymbol{x}$
  - It is any function f that satisfies:

$$f(\boldsymbol{x}) = 0 \Rightarrow \boldsymbol{x} = 0$$

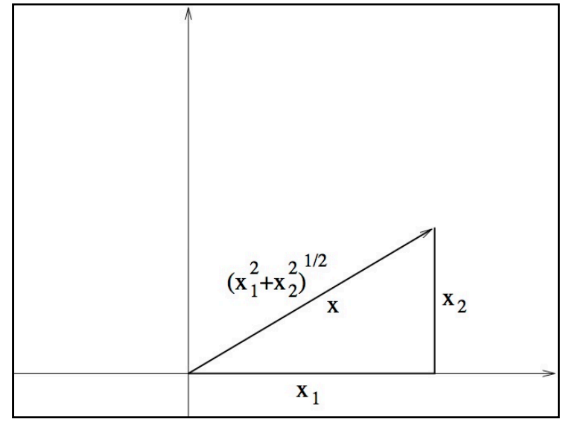
$$f(\boldsymbol{x}+\boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y}) \quad \text{Triangle Inequality}$$

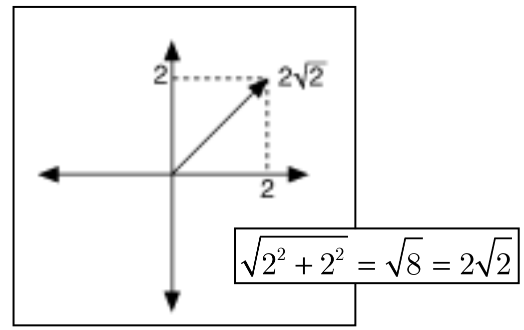
$$\forall \alpha \in R \quad f(\alpha \boldsymbol{x}) = |\alpha| f(\boldsymbol{x})$$

L^p Norm

- Definition:

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





- L^2 Norm
  - Called Euclidean norm
    - Simply the Euclidean distance between the origin and the point ***x***
    - written simply as $||\boldsymbol{x}||$
    - Squared Euclidean norm is same as $\boldsymbol{x}^T\boldsymbol{x}$
- L^1 Norm
  - Useful when 0 and non-zero have to be distinguished
    - Note that L^2 increases slowly near origin, e.g., $0.1^2=0.01$)
- L^∞ Norm

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

  - Called max norm

Use of norm in Regression

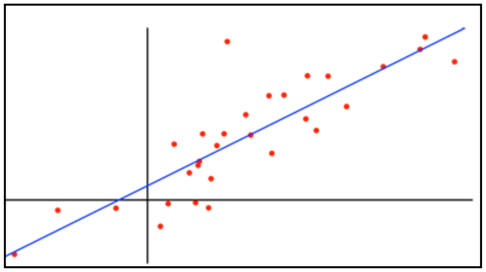
- Linear Regression

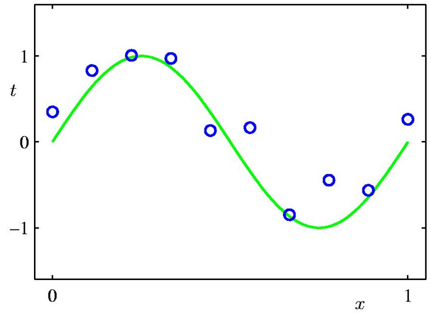
  x: a vector, ***w***: weight vector

  $y(\boldsymbol{x},\boldsymbol{w}) = w_0 + w_1 x_1 + .. + w_d x_d = \boldsymbol{w}^T \boldsymbol{x}$

  With nonlinear basis functions ϕ_j

$$y(\boldsymbol{x},\boldsymbol{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\boldsymbol{x})$$





- Loss Function

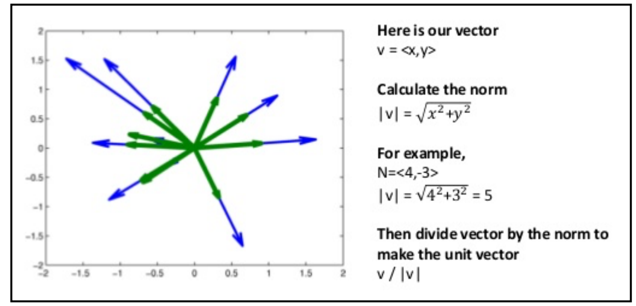
$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\boldsymbol{x}_n,\boldsymbol{w}) - t_n\}^2 + \frac{\lambda}{2} \, || \, \boldsymbol{w}^2 \, ||$$

  Second term is a weighted norm

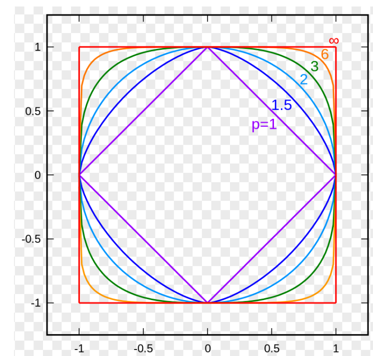
  called a regularizer (to prevent overfitting)

L^P Norm and Distance

- Norm is the length of a vector



- We can use it to draw a unit circle from origin
  - Different *P* values yield different shapes
    - Euclidean norm yields a circle



- Distance between two vectors (***v,w***)
  - dist(***v,w***)=||***v-w***||

$$= \sqrt{(v_1 - w_1)^2 + .. + (v_n - w_n)^2}$$

Distance to origin would just be sqrt of sum of squares

Size of a Matrix: Frobenius Norm

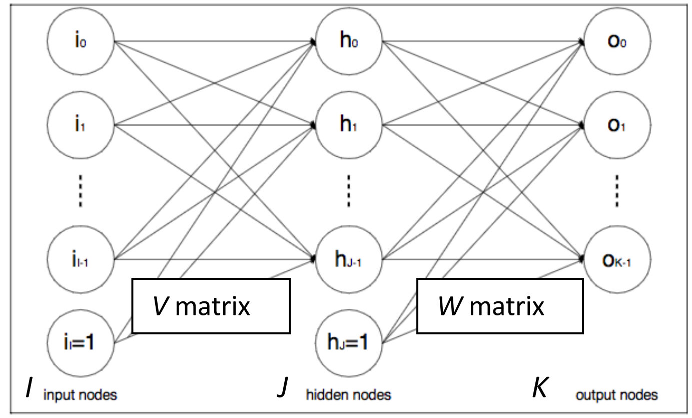
- Similar to L^2 norm

$$\|A\|_F = \left(\sum_{i,j} A_{i,j}^2\right)^{\frac{1}{2}}$$

$$A = \begin{bmatrix} 2 & -1 & 5 \\ 0 & 2 & 1 \\ 3 & 1 & 1 \end{bmatrix} \qquad \|A\| = \sqrt{4+1+25+..+1} = \sqrt{46}$$

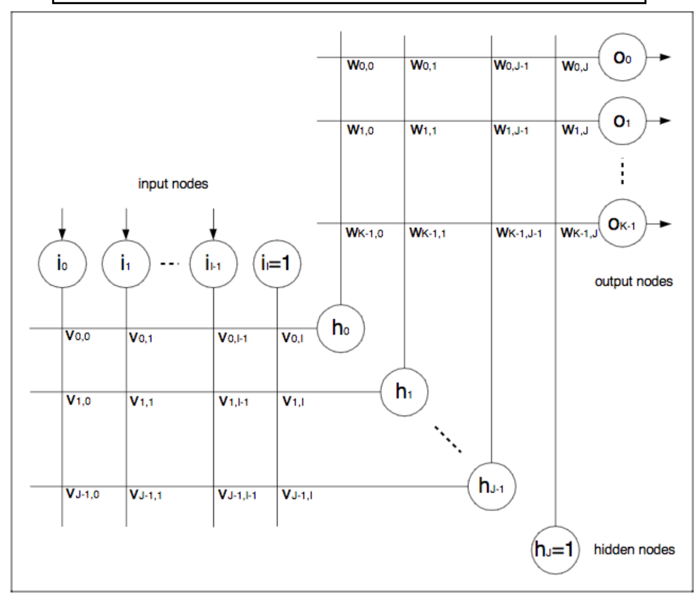
- Frobenius in ML
  - Layers of neural network involve matrix multiplication
  - Regularization:
    - minimize Frobenius of weight matrices $||W(i)||$ over L layers

$$J_R = J + \lambda \sum_{i=1}^{L} \left\| W^{(i)} \right\|_F$$



$$I_{1\times(I+1)} \times V_{(I+1)\times J} = net_J$$

$$h_j = f(net_j) \qquad f(x) = 1/(1+e^{-x})$$



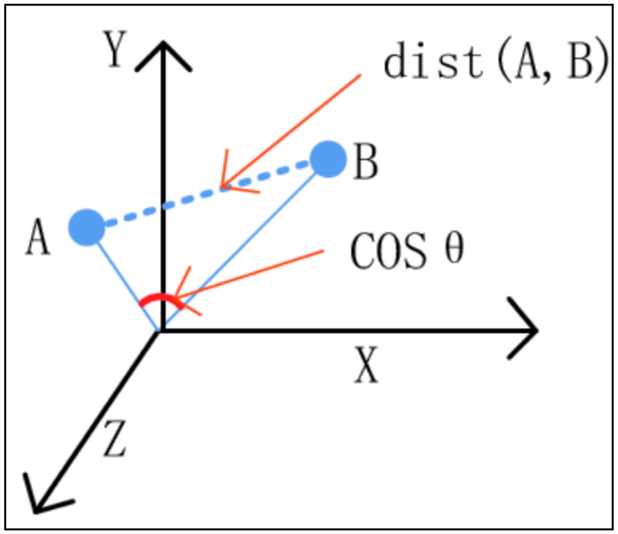
Angle between Vectors

- Dot product of two vectors can be written in terms of their L^2 norms and angle θ between them

$$x^T y \Rightarrow ||x||_2 ||y||_2 \cos\theta$$

- Cosine between two vectors is a measure of their similarity

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$



Special kind of Matrix: Diagonal

- Diagonal Matrix has mostly zeros, with non-zero entries only in diagonal
  - E.g., identity matrix, where all diagonal entries are 1

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

  - E.g., covariance matrix with independent features

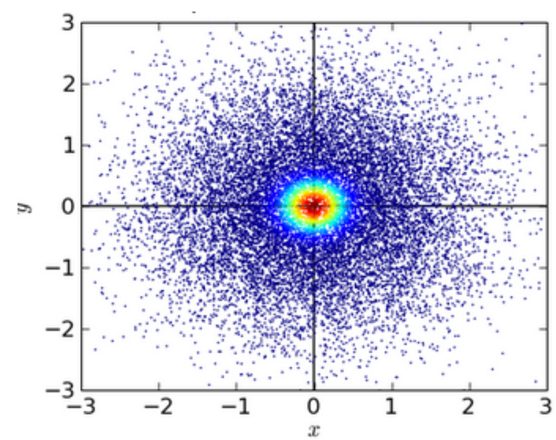
$$Cov(X,Y) = \sigma_{XY} = E[(X-\mu_X)(Y-\mu_Y)]$$

$$\text{Covariance} = \frac{\sum(x_i - x_{avg})(y_i - y_{avg})}{n-1}$$

$$\text{Covariance} = \frac{-64.57}{8}$$

$$\text{Covariance} = \boxed{-8.07}$$

$$\begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_K^2 \end{bmatrix}$$



If Cov(X,Y)=0 then E(XY)=E(X)E(Y)

$$N(\mathbf{x}\,|\,\boldsymbol{\mu},\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

Efficiency of Diagonal Matrix

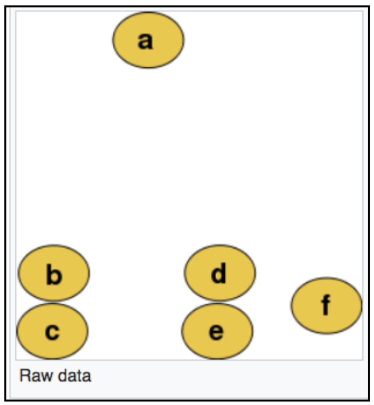
- diag (***v***) denotes a square diagonal matrix with diagonal elements given by entries of vector ***v***
- Multiplying vector ***x*** by a diagonal matrix is efficient
  - To compute diag(***v***)***x*** we only need to scale each x_i by v_i

$$\operatorname{diag}(\boldsymbol{v})\boldsymbol{x} = \boldsymbol{v} \odot \boldsymbol{x}$$

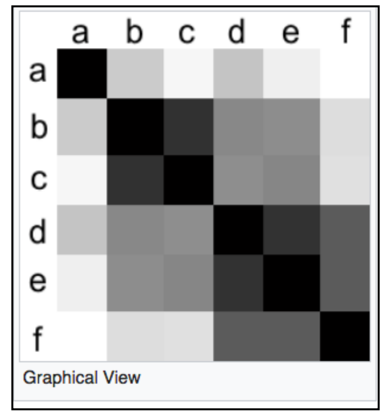
- Inverting a square diagonal matrix is efficient
  - Inverse exists iff every diagonal entry is nonzero, in which case diag (***v***)$^{-1}$=diag ($[1/v_1,..,1/v_n]^T$)

Special kind of Matrix: Symmetric

- A symmetric matrix equals its transpose: $A=A^T$
  - E.g., a distance matrix is symmetric with $A_{ij}=A_{ji}$



	a	b	c	d	e	f
a	0	184	222	177	216	231
b	184	0	45	123	128	200
c	222	45	0	129	121	203
d	177	123	129	0	46	83
e	216	128	121	46	0	83
f	231	200	203	83	83	0



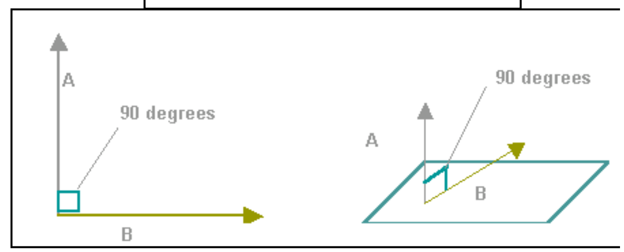
  - E.g., covariance matrices are symmetric

$$\Sigma = \begin{pmatrix} 1 & .5 & .15 & .15 & 0 & 0 \\ .5 & 1 & .15 & .15 & 0 & 0 \\ .15 & .15 & 1 & .25 & 0 & 0 \\ .15 & .15 & .25 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & .10 \\ 0 & 0 & 0 & 0 & .10 & 1 \end{pmatrix}$$

Special Kinds of Vectors

- Unit Vector
  - A vector with unit norm $\|x\|_2 = 1$
- Orthogonal Vectors
  - A vector ***x*** and a vector ***y*** are orthogonal to each other if $\boldsymbol{x}^T\boldsymbol{y}=0$
    - If vectors have nonzero norm, vectors at 90 degrees to each other

$$\begin{bmatrix} 2 \\ -3 \\ -2 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 4 \\ -2 \\ 7 \end{bmatrix}$$



  - Orthonormal Vectors
    - Vectors are orthogonal & have unit norm
    - Orthogonal Matrix
      - A square matrix whose rows are mutually orthonormal: $A^TA=AA^T=I$
      - $A^{-1}=A^T$

Orthogonal matrices are of interest because their inverse is very cheap to compute

Matrix decomposition

- Matrices can be decomposed into factors to learn universal properties, just like integers:
  - Properties not discernible from their representation

1. Decomposition of integer into prime factors
   - From $12=2\times 2\times 3$ we can discern that
     - 12 is not divisible by 5 or
     - any multiple of 12 is divisible by 3
     - But representations of 12 in binary or decimal are different

2. Decomposition of Matrix *A* as $A=V\text{diag}(\lambda)V^{-1}$
   - where *V* is formed of eigenvectors and λ are eigenvalues, e.g,

$A=\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ has eigenvalues λ=1 and λ=3 and eigenvectors *V*: $v_{\lambda=1}=\begin{bmatrix} 1 \\ -1 \end{bmatrix}, v_{\lambda=3}=\begin{bmatrix} 1 \\ 1 \end{bmatrix}$

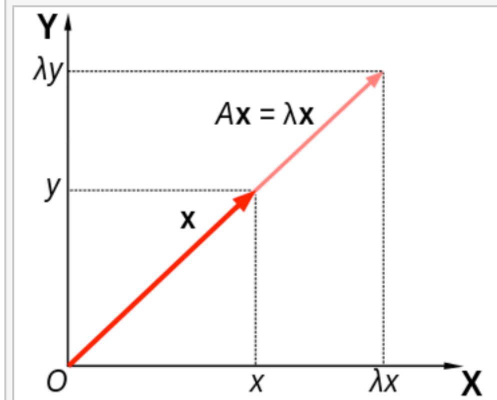
Eigenvector

- An eigenvector of a square matrix ***A*** is a non-zero vector ***v*** such that multiplication by ***A*** only changes the scale of ***v***

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

  – The scalar λ is known as eigenvalue

- If ***v*** is an eigenvector of ***A***, so is any rescaled vector ***sv***. Moreover ***sv*** still has the same eigen value. Thus look for a unit eigenvector



Matrix *A* acts by stretching the vector *x*, not changing its direction, so *x* is an eigenvector of *A*.

Wikipedia

Eigenvalue and Characteristic Polynomial

$$A = \begin{bmatrix} A_{1,1} & L & A_{1,n} \\ M & M & M \\ A_{n,1} & L & A_{nn} \end{bmatrix} \quad \boldsymbol{v} = \begin{bmatrix} v_1 \\ M \\ v_n \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} w_1 \\ M \\ w_n \end{bmatrix}$$

- Consider $A\boldsymbol{v}=\boldsymbol{w}$
- If $\boldsymbol{v}$ and $\boldsymbol{w}$ are scalar multiples, i.e., if $A\boldsymbol{v}=\lambda\boldsymbol{v}$
    - then $\boldsymbol{v}$ is an eigenvector of the linear transformation A and the scale factor λ is the eigenvalue corresponding to the eigen vector
- This is the *eigenvalue equation* of matrix A
    - Stated equivalently as $(A-\lambda I)\boldsymbol{v}=0$
    - This has a non-zero solution if $|A-\lambda I|=0$ as
        - The polynomial of degree n can be factored as

$$|A-\lambda I| = (\lambda_1-\lambda)(\lambda_2-\lambda)\ldots(\lambda_n-\lambda)$$

- The $\lambda_1, \lambda_2 \ldots \lambda_n$ are roots of the polynomial and are eigenvalues of A

Example of Eigenvalue/Eigenvector

- Consider the matrix $$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$
- Taking determinant of $(A-\lambda I)$, the char poly is

$$|A - \lambda I| = \begin{bmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{bmatrix} = 3 - 4\lambda + \lambda^2$$

- It has roots $\lambda=1$ and $\lambda=3$ which are the two eigenvalues of A
- The eigenvectors are found by solving for ***v*** in $A\boldsymbol{v}=\lambda\boldsymbol{v}$, which are

$$v_{\lambda=1} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, v_{\lambda=3} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Eigendecomposition

- Suppose that matrix A has n linearly independent eigenvectors $\{\boldsymbol{v}^{(1)},..,\boldsymbol{v}^{(n)}\}$ with eigenvalues $\{\lambda_1,..,\lambda_n\}$
- Concatenate eigenvectors to form matrix V
- Concatenate eigenvalues to form vector $\lambda=[\lambda_1,..,\lambda_n]$
- Eigendecomposition of A is given by

$$A=V\mathrm{diag}(\lambda)V^{-1}$$

Decomposition of Symmetric Matrix

- Every real symmetric matrix A can be decomposed into real-valued eigenvectors and eigenvalues

$$A=Q\Lambda Q^{\mathrm{T}}$$

where Q is an orthogonal matrix composed of eigenvectors of A: $\{\boldsymbol{v}^{(1)},..,\boldsymbol{v}^{(n)}\}$

orthogonal matrix: components are orthogonal or $\boldsymbol{v}^{(i)\mathrm{T}}\boldsymbol{v}^{(j)}=0$

Λ is a diagonal matrix of eigenvalues $\{\lambda_1,..,\lambda_n\}$

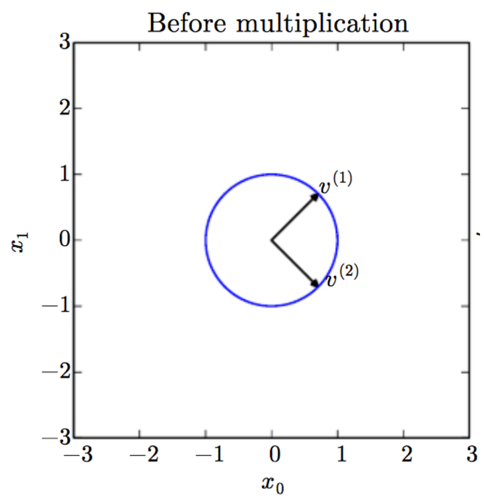
- We can think of A as scaling space by λ_i in direction $\boldsymbol{v}^{(i)}$
  - See figure on next slide

Effect of Eigenvectors and Eigenvalues

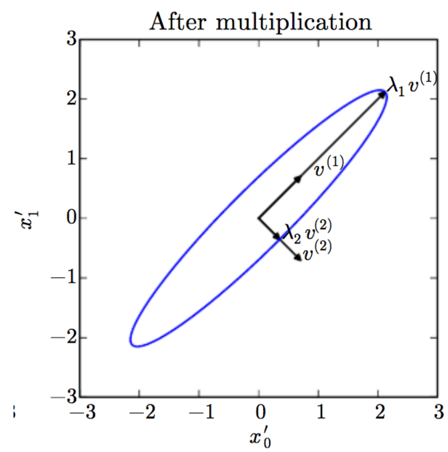
- Example of 2×2 matrix
- Matrix A with two orthonormal eigenvectors
  - $v^{(1)}$ with eigenvalue λ_1, $v^{(2)}$ with eigenvalue λ_2

Plot of unit vectors $\boldsymbol{u} \in \mathbb{R}^2$ (circle)

Plot of vectors $A\boldsymbol{u}$ (ellipse)



with two variables x_1 and x_2



Python Code for Eigenvalue/Eigenvector

- https://www.youtube.com/watch?v=mxkGMbrobY0&feature=youtu.be&fbclid=IwAR3ajOaxWmnV-rYnAa6cwYfq9j6is6-H8UhnIMCkhBu3Cqfvby_vicyU2fg

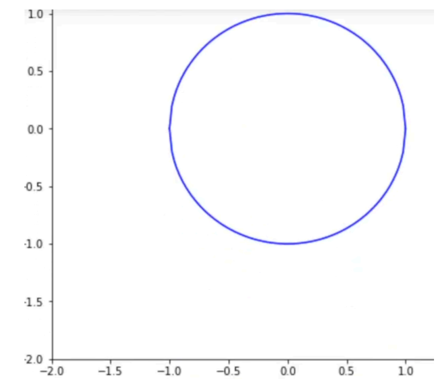
In [33]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = 8,8
```

$$\begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix}$$

In [ ]:
```
x = np.linspace(-1,1,100)
```

In [ ]:
```
y1 = np.sqrt(1 - np.square(x))
y2 = -1 * y1
```

In [ ]:
```
plt.plot(x,y1, 'b')
plt.plot(x,y2, 'b')
plt.xlim([-2, 2])
plt.ylim([-2, 2])
plt.show()
```

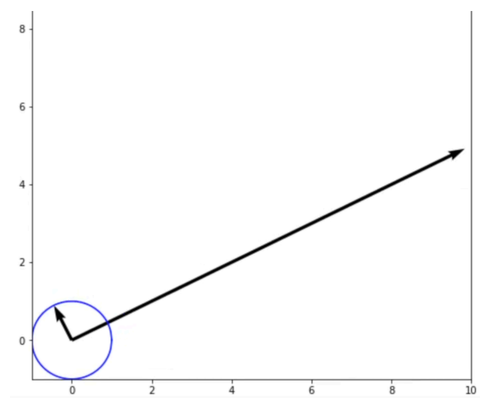


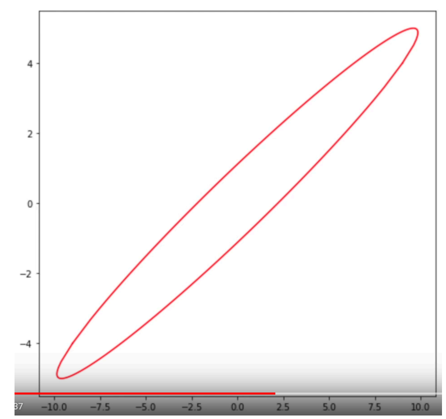
In [ ]:
```
def transformation(x,y):
    return 9*x + 4*y, 4*x + 3*y
```

In [ ]:
```
x_new1, y_new1 = transformation(x,y1)
x_new2, y_new2 = transformation(x,y2)
```

In [ ]:
```
plt.plot(x_new1,y_new1, 'r')
plt.plot(x_new2,y_new2, 'r')
```

In [ ]:
```
eig_vals, eig_vecs = np.linalg.eig(np.array([[9,4],[4,3]]))
print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```





Eigendecomposition is not unique

- Eigendecomposition is $A=Q\Lambda Q^T$
  - where Q is an orthogonal matrix composed of eigenvectors of A
- Decomposition is not unique when two eigenvalues are the same
- By convention order entries of Λ in descending order:
  - Under this convention, eigendecomposition is unique if all eigenvalues are unique

What does eigendecomposition tell us?

- Tells us useful facts about the matrix:
  1. Matrix is *singular* if & only if any eigenvalue is zero
  2. Useful to optimize quadratic expressions of form

     $f(\boldsymbol{x})=\boldsymbol{x}^T A \boldsymbol{x}$ subject to $||\boldsymbol{x}||_2=1$

     Whenever $\boldsymbol{x}$ is equal to an eigenvector, f is equal to the corresponding eigenvalue

     Maximum value of f is max eigen value, minimum value is min eigen value

     Example of such a quadratic form appears in multivariate Gaussian

$$N(\mathrm{x} \mid \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left\{ -\frac{1}{2} (\mathrm{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathrm{x}-\boldsymbol{\mu}) \right\}$$

Positive Definite Matrix

- A matrix whose eigenvalues are all positive is called *positive definite*
  - Positive or zero is called *positive semidefinite*
- If eigen values are all negative it is *negative definite*
  - Positive definite matrices guarantee that $\boldsymbol{x}^T A \boldsymbol{x} \geq 0$

Singular Value Decomposition (SVD)

- Eigendecomposition has form: $A = V\text{diag}(\lambda)V^{-1}$
  - If A is not square, eigendecomposition is undefined
- SVD is a decomposition of the form $A = UDV^T$
- SVD is more general than eigendecomposition
  - Used with any matrix rather than symmetric ones
  - Every real matrix has a SVD
    - Same is not true of eigen decomposition

SVD Definition

- Write A as a product of 3 matrices: $A=UDV^T$
  - If A is $m \times n$, then U is $m \times m$, D is $m \times n$, V is $n \times n$
- Each of these matrices have a special structure
  - U and V are orthogonal matrices
  - D is a diagonal matrix not necessarily square
    - Elements of Diagonal of D are called *singular values of A*
    - Columns of U are called *left singular vectors*
    - Columns of V are called *right singular vectors*
- SVD interpreted in terms of *eigendecomposition*
  - Left singular vectors of A are eigenvectors of AA^T
  - Right singular vectors of A are eigenvectors of A^TA
  - Nonzero singular values of A are square roots of eigen values of A^TA. Same is true of AA^T

Use of SVD in ML

1. SVD is used in generalizing matrix inversion
   - Moore-Penrose inverse (discussed next)
2. Used in Recommendation systems
   - Collaborative filtering (CF)
     - Method to predict a rating for a *user-item* pair based on the history of ratings given by the user and given to the item
     - Most CF algorithms are based on *user-item* rating matrix where each row represents a user, each column an item
       - Entries of this matrix are ratings given by users to items
     - SVD reduces no.of features of a data set by reducing space dimensions from N to K where $K < N$

SVD in Collaborative Filtering

$$\underset{m \times n}{\overset{\hat{X}}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}}} \approx \underset{m \times r}{\overset{U}{\begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix}}} \underset{r \times r}{\overset{S}{\begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix}}} \underset{r \times n}{\overset{V^{\top}}{\begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}}}$$

- X is the utility matrix
  - X_{ij} denotes how user i likes item j
  - CF fills blank (cell) in utility matrix that has no entry
- Scalability and sparsity is handled using SVD
  - SVD decreases dimension of utility matrix by extracting its latent factors
    - Map each user and item into latent space of dimension r

Moore-Penrose Pseudoinverse

- Most useful feature of SVD is that it can be used to generalize matrix inversion to non-square matrices
- Practical algorithms for computing the pseudoinverse of A are based on SVD

$$A^{+}=VD^{+}U^{T}$$

  – where U,D,V are the SVD of A
    - Pseudoinverse D^{+} of D is obtained by taking the reciprocal of its nonzero elements when taking transpose of resulting matrix

Trace of a Matrix

- Trace operator gives the sum of the elements along the diagonal

$$Tr(A) = \sum_{i,i} A_{i,i}$$

- Frobenius norm of a matrix can be represented as

$$||A||_F = \left(Tr(A)\right)^{\frac{1}{2}}$$

Determinant of a Matrix

- Determinant of a square matrix $\det(A)$ is a mapping to a scalar
- It is equal to the product of all eigenvalues of the matrix
- Measures how much multiplication by the matrix expands or contracts space

Example: PCA

- A simple ML algorithm is *Principal Components Analysis*
- It can be derived using only knowledge of basic linear algebra

PCA Problem Statement

- Given a collection of m points $\{\boldsymbol{x}^{(1)},..,\boldsymbol{x}^{(m)}\}$ in R^n represent them in a lower dimension.
  - For each point $\boldsymbol{x}^{(i)}$ find a code vector $\boldsymbol{c}^{(i)}$ in R^l
  - If l is smaller than n it will take less memory to store the points
  - This is lossy compression
  - Find encoding function $f(\boldsymbol{x}) = \boldsymbol{c}$ and a decoding function $\boldsymbol{x} \approx g(f(\boldsymbol{x}))$

PCA using Matrix multiplication

- One choice of decoding function is to use matrix multiplication: $g(\boldsymbol{c}) = D\boldsymbol{c}$ where $D \in \mathbb{R}^{n \times l}$
  - *D* is a matrix with l columns
- To keep encoding easy, we require columns of *D* to be orthogonal to each other
  - To constrain solutions we require columns of *D* to have unit norm
- We need to find optimal code $\boldsymbol{c}^*$ given *D*
- Then we need optimal *D*

Finding optimal code given D

- To generate optimal code point c^* given input $\boldsymbol{x}$, minimize the distance between input point $\boldsymbol{x}$ and its reconstruction $g(c^*)$

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} \|\boldsymbol{x} - g(\boldsymbol{c})\|_2$$

  - Using squared L^2 instead of L^2, function being minimized is equivalent to

$$(\boldsymbol{x} - g(\boldsymbol{c}))^T (\boldsymbol{x} - g(\boldsymbol{c}))$$

- Using $g(\boldsymbol{c})=D\boldsymbol{c}$ optimal code can be shown to be equivalent to

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} -2\boldsymbol{x}^T D\boldsymbol{c} + \boldsymbol{c}^T \boldsymbol{c}$$

Optimal Encoding for PCA

- Using vector calculus

$$\nabla_c(-2\boldsymbol{x}^T D\boldsymbol{c} + \boldsymbol{c}^T \boldsymbol{c}) = \boldsymbol{0}$$
$$-2D^T \boldsymbol{x} + 2\boldsymbol{c} = \boldsymbol{0}$$
$$c = D^T \boldsymbol{x}$$

- Thus we can encode $\boldsymbol{x}$ using a matrix-vector operation
  - To encode we use $f(\boldsymbol{x})=D^T\boldsymbol{x}$
  - For PCA reconstruction, since $g(\boldsymbol{c})=D\boldsymbol{c}$ we use $r(\boldsymbol{x})=g(f(\boldsymbol{x}))=DD^T\boldsymbol{x}$
  - Next we need to choose the encoding matrix D

Method for finding optimal D

- Revisit idea of minimizing L^2 distance between inputs and reconstructions
  - But cannot consider points in isolation
  - So minimize error over all points: Frobenius norm

$$D^* = \arg\min_{D} \left(\sum_{i,j} \left(x_j^{(i)} - r\left(x^{(i)}\right)_j \right)^2 \right)^{\frac{1}{2}}$$

    - subject to $D^T D = I_l$
- Use design matrix X, $X \in \mathbb{R}^{m \times n}$
  - Given by stacking all vectors describing the points
- To derive algorithm for finding D^* start by considering the case $l = 1$
  - In this case D is just a single vector $\boldsymbol{d}$

Final Solution to PCA

- For l =1, the optimization problem is solved using eigendecomposition
  - Specifically the optimal $\boldsymbol{d}$ is given by the eigenvector of X^TX corresponding to the largest eigenvalue
- More generally, matrix D is given by the l eigenvectors of X corresponding to the largest eigenvalues (Proof by induction)