

CS 2750: Machine Learning

Linear Algebra and Matlab

Prof. Adriana Kovashka
University of Pittsburgh
January 10, 2017

Announcement

- TA won't be back until the last week of January
- Skype office hours:
 - Tuesday/Wednesday 9:30am to 11:30am
 - Username: **vicl720**
- Please do Doodle:
 - <http://doodle.com/poll/gtaxphum9mutd74x>

Linear Algebra Primer

Professor Fei-Fei Li
Stanford

Another, very in-depth linear algebra review from CS229 is available here:

<http://cs229.stanford.edu/section/cs229-linalg.pdf>

And a video discussion of linear algebra from EE263 is here (lectures 3 and 4):

<http://see.stanford.edu/see/lecturelist.aspx?coll=17005383-19c6-49ed-9497-2ba8bfcfe5f6>

Vectors and Matrices

- Vectors and matrices are just collections of ordered numbers that represent something: movements in space, scaling factors, word counts, movie ratings, pixel brightnesses, etc. We'll define some common uses and standard operations on them.

Vector

- A column vector $\mathbf{v} \in \mathbb{R}^{n \times 1}$ where

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

- A row vector $\mathbf{v}^T \in \mathbb{R}^{1 \times n}$ where

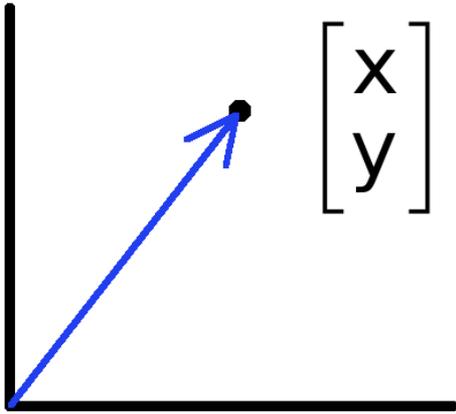
$$\mathbf{v}^T = [v_1 \quad v_2 \quad \dots \quad v_n]$$

T denotes the transpose operation

Vector

- You'll want to keep track of the orientation of your vectors when programming in MATLAB.
- You can transpose a vector V in MATLAB by writing V' .

Vectors have two main uses



- Vectors can represent an offset in 2D or 3D space
- Points are just vectors from the origin
- Data can also be treated as a vector
- Such vectors don't have a geometric interpretation, but calculations like "distance" still have value

Matrix

- A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is an array of numbers with size $m \downarrow$ by $n \rightarrow$, i.e. m rows and n columns.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & & & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

- If $m = n$, we say that \mathbf{A} is square.

Matrix Operations

- Addition

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a + 1 & b + 2 \\ c + 3 & d + 4 \end{bmatrix}$$

- Can only add a matrix with matching dimensions, or a scalar.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + 7 = \begin{bmatrix} a + 7 & b + 7 \\ c + 7 & d + 7 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times 3 = \begin{bmatrix} 3a & 3b \\ 3c & 3d \end{bmatrix}$$

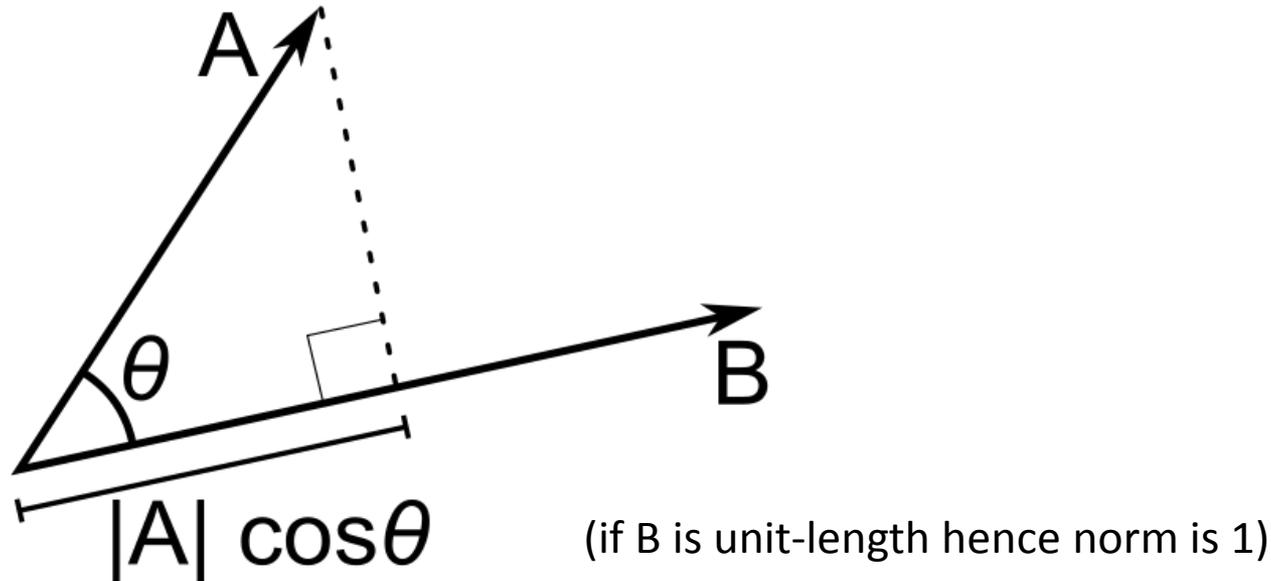
Matrix Operations

- Inner product (dot product) of vectors
 - Multiply corresponding entries of two vectors and add up the result
 - $\mathbf{x} \cdot \mathbf{y}$ is also $|\mathbf{x}| |\mathbf{y}| \cos(\text{the angle between } \mathbf{x} \text{ and } \mathbf{y})$

$$\mathbf{x}^T \mathbf{y} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^n x_i y_i \quad (\text{scalar})$$

Matrix Operations

- Inner product (dot product) of vectors
 - If B is a unit vector, then $A \cdot B$ gives the length of A which lies in the direction of B (projection)



Norms

- L1 norm

$$\|\mathbf{x}\|_1 := \sum_{i=1}^n |\mathbf{x}_i|$$

- L2 norm

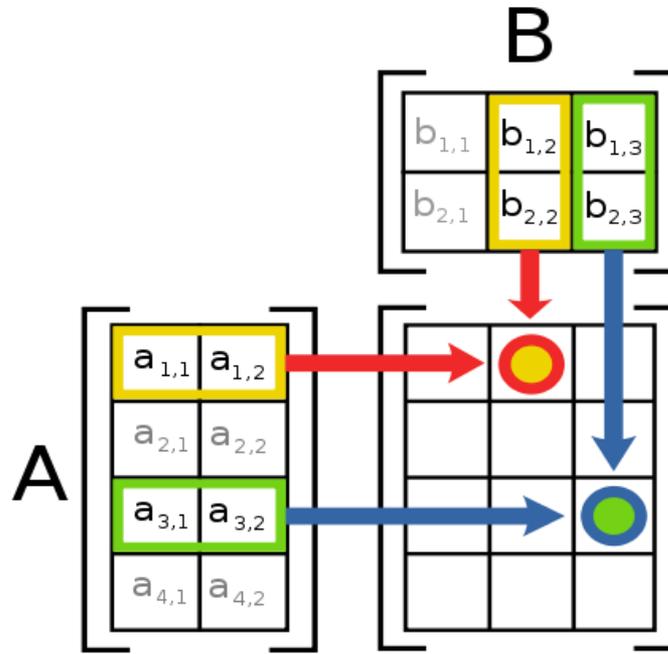
$$\|\mathbf{x}\| := \sqrt{\mathbf{x}_1^2 + \cdots + \mathbf{x}_n^2}$$

- L^p norm (for real numbers $p \geq 1$)

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^n |\mathbf{x}_i|^p \right)^{1/p}$$

Matrix Operations

- Multiplication
- The product AB is:



- Each entry in the result is (that row of A) dot product with (that column of B)

Matrix Operations

- Multiplication example:

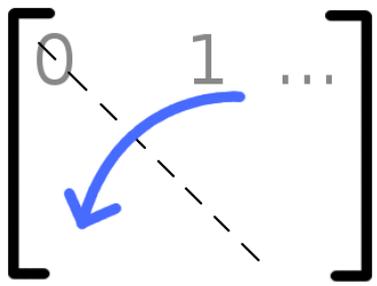
$$\begin{array}{ccc} A & \times & B \\ \downarrow & & \swarrow \\ \begin{bmatrix} 0 & 2 \\ 4 & 6 \end{bmatrix} & & \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} \\ & & \begin{bmatrix} \square & 14 \\ \square & \square \end{bmatrix} \end{array}$$

$$0 \cdot 3 + 2 \cdot 7 = 14$$

- Each entry of the matrix product is made by taking the dot product of the corresponding row in the left matrix, with the corresponding column in the right one.

Matrix Operations

- Transpose – flip matrix, so row 1 becomes column 1


$$\begin{bmatrix} 0 & 1 & \dots \\ 2 & 3 & \\ 4 & 5 & \end{bmatrix}^T = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

- A useful identity:

$$(ABC)^T = C^T B^T A^T$$

Special Matrices

- Identity matrix \mathbf{I}
 - Square matrix, 1's along diagonal, 0's elsewhere
 - $\mathbf{I} \cdot [\text{another matrix}] = [\text{that matrix}]$

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

- Diagonal matrix
 - Square matrix with numbers along diagonal, 0's elsewhere
 - A diagonal \cdot [another matrix] scales the rows of that matrix

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2.5 \end{bmatrix}$$

Special Matrices

- Symmetric matrix

$$\mathbf{A}^T = \mathbf{A}$$

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 7 \\ 5 & 7 & 1 \end{bmatrix}$$

Inverse

- Given a matrix \mathbf{A} , its inverse \mathbf{A}^{-1} is a matrix such that $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$
- E.g. $\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$
- Inverse does not always exist. If \mathbf{A}^{-1} exists, \mathbf{A} is *invertible* or *non-singular*. Otherwise, it's *singular*.

Matrix Operations

- MATLAB example:

$$AX = B$$

$$A = \begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

```
>> x = A\B
```

```
x =
```

```
    1.0000
```

```
   -0.5000
```

Pseudo-inverse

- Say you have the matrix equation $AX=B$, where A and B are known, and you want to solve for X
- You could use MATLAB to calculate the inverse and premultiply by it: $A^{-1}AX=A^{-1}B \rightarrow X=A^{-1}B$
- MATLAB command would be **`inv(A)*B`**
- But calculating the inverse for large matrices often brings problems with computer floating-point resolution, or your matrix might not even have an inverse. Fortunately, there are workarounds.
- Instead of taking an inverse, directly ask MATLAB to solve for X in $AX=B$, by typing **`A\B`**
- MATLAB will try several appropriate numerical methods (including the pseudoinverse if the inverse doesn't exist)
- MATLAB will return the value of X which solves the equation
 - If there is no exact solution, it will return the closest one
 - If there are many solutions, it will return the smallest one

Matrix Operation Properties

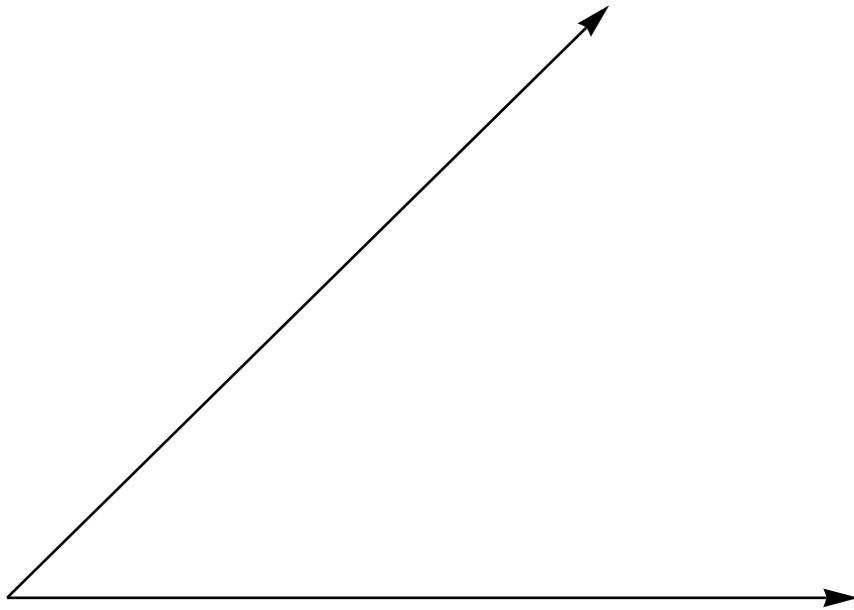
- Matrix addition is commutative and associative
 - $A + B = B + A$
 - $A + (B + C) = (A + B) + C$
- Matrix multiplication is associative and distributive but *not* commutative
 - $A(B * C) = (A * B)C$
 - $A(B + C) = A * B + A * C$
 - $A * B \neq B * A$

Linear independence

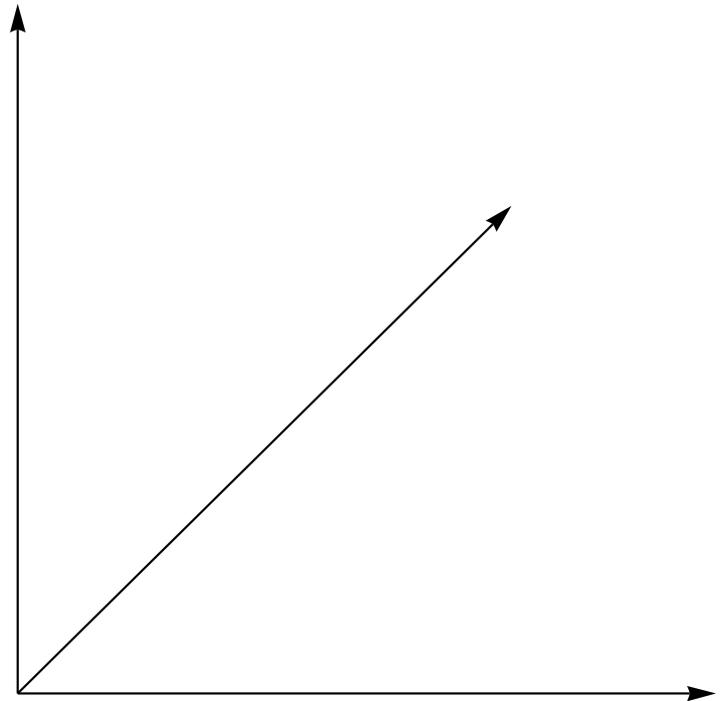
- Suppose we have a set of vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$
- If we can express \mathbf{v}_1 as a linear combination of the other vectors $\mathbf{v}_2 \dots \mathbf{v}_n$, then \mathbf{v}_1 is linearly *dependent* on the other vectors.
 - The direction \mathbf{v}_1 can be expressed as a combination of the directions $\mathbf{v}_2 \dots \mathbf{v}_n$. (E.g. $\mathbf{v}_1 = .7 \mathbf{v}_2 - .7 \mathbf{v}_4$)
- If no vector is linearly dependent on the rest of the set, the set is linearly *independent*.
 - Common case: a set of vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ is always linearly independent if each vector is perpendicular to every other vector (and non-zero)

Linear independence

Linearly independent set



Not linearly independent



Matrix rank

- Column/row rank

col-rank(\mathbf{A}) = the maximum number of linearly independent column vectors of \mathbf{A}
row-rank(\mathbf{A}) = the maximum number of linearly independent row vectors of \mathbf{A}

- Column rank always equals row rank

- Matrix rank $\text{rank}(\mathbf{A}) \triangleq \text{col-rank}(\mathbf{A}) = \text{row-rank}(\mathbf{A})$

- If a matrix is not full rank, inverse doesn't exist
 - Inverse also doesn't exist for non-square matrices

Singular Value Decomposition (SVD)

- There are several computer algorithms that can “factor” a matrix, representing it as the product of some other matrices
- The most useful of these is the Singular Value Decomposition
- Represents any matrix **A** as a product of three matrices: **$U\Sigma V^T$**
- MATLAB command: **$[U,S,V] = \text{svd}(A);$**

Singular Value Decomposition (SVD)

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

- Where \mathbf{U} and \mathbf{V} are rotation matrices, and $\mathbf{\Sigma}$ is a scaling matrix. For example:

$$\begin{matrix} U & & \Sigma & & V^T & & A \\ \begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix} & \times & \begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix} & \times & \begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix} & = & \begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix} \end{matrix}$$

Singular Value Decomposition (SVD)

- In general, if \mathbf{A} is $m \times n$, then \mathbf{U} will be $m \times m$, $\mathbf{\Sigma}$ will be $m \times n$, and \mathbf{V}^T will be $n \times n$.

$$\begin{matrix} U \\ \begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix} \end{matrix} \times \begin{matrix} \Sigma \\ \begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} \end{matrix} \times \begin{matrix} V^T \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{matrix} = \begin{matrix} A \\ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \end{matrix}$$

Singular Value Decomposition (SVD)

- **U** and **V** are always rotation matrices.
 - Geometric rotation may not be an applicable concept, depending on the matrix. So we call them “unitary” matrices – each column is a unit vector.
- **Σ** is a diagonal matrix
 - The number of nonzero entries = rank of **A**
 - The algorithm always sorts the entries high to low

$$\begin{matrix} U \\ \begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix} \end{matrix} \times \begin{matrix} \Sigma \\ \begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} \end{matrix} \times \begin{matrix} V^T \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{matrix} = \begin{matrix} A \\ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \end{matrix}$$

Singular Value Decomposition (SVD)

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

SVD Applications

- We've discussed SVD in terms of geometric transformation matrices
- But SVD of a data matrix can also be very useful
- To understand this, we'll look at a less geometric interpretation of what SVD is doing

SVD Applications

$$\begin{matrix} U & & \Sigma & & V^T & & A \\ \begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix} & \times & \begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} & \times & \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} & = & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
 \end{matrix}$$

- Look at how the multiplication works out, left to right:
- Column 1 of \mathbf{U} gets scaled by the first value from Σ .

$$\begin{matrix} \swarrow U\Sigma \\ \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}
 \end{matrix}$$

- The resulting vector gets scaled by row 1 of \mathbf{V}^T to produce a contribution to the columns of \mathbf{A}

SVD Applications

$$\begin{array}{l}
 \begin{array}{c} U\Sigma \\ \left[\begin{array}{ccc} -3.67 & -0.71 & 0 \\ -8.8 & 0.30 & 0 \end{array} \right] \times \begin{array}{c} V^T \\ \left[\begin{array}{ccc} -0.42 & -0.57 & -0.70 \\ 0.81 & 0.11 & -0.58 \\ 0.41 & -0.82 & 0.41 \end{array} \right] \end{array} \\
 \\
 + \begin{array}{c} U\Sigma \\ \left[\begin{array}{ccc} -3.67 & -0.71 & 0 \\ -8.8 & 0.30 & 0 \end{array} \right] \times \begin{array}{c} V^T \\ \left[\begin{array}{ccc} -0.42 & -0.57 & -0.70 \\ 0.81 & 0.11 & -0.58 \\ 0.41 & -0.82 & 0.41 \end{array} \right] \end{array} \\
 \\
 = \begin{array}{c} A_{\text{partial}} \\ \left[\begin{array}{ccc} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{array} \right] \\
 \\
 A_{\text{partial}} \\ \left[\begin{array}{ccc} -0.6 & -0.1 & 0.4 \\ 0.2 & 0 & -0.2 \end{array} \right] \\
 \\
 \left[\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array} \right]
 \end{array}
 \end{array}$$

- Each product of (*column i of \mathbf{U}*)·(*value i from Σ*)·(*row i of \mathbf{V}^T*) produces a component of the final \mathbf{A} .

SVD Applications

$$\begin{array}{c}
 U\Sigma \\
 \begin{bmatrix} -3.67 & -0.71 & 0 \\ -8.8 & 0.30 & 0 \end{bmatrix} \times \begin{array}{c} V^T \\ \begin{bmatrix} -0.42 & -0.57 & -0.70 \\ 0.81 & 0.11 & -0.58 \\ 0.41 & -0.82 & 0.41 \end{bmatrix} \end{array} \\
 \begin{array}{c} A_{\text{partial}} \\ \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix} \end{array}
 \end{array}
 \qquad
 \begin{array}{c}
 A \\
 \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
 \end{array}$$

$$\begin{array}{c}
 U\Sigma \\
 \begin{bmatrix} -3.67 & -0.71 & 0 \\ -8.8 & 0.30 & 0 \end{bmatrix} \times \begin{array}{c} V^T \\ \begin{bmatrix} -0.42 & -0.57 & -0.70 \\ 0.81 & 0.11 & -0.58 \\ 0.41 & -0.82 & 0.41 \end{bmatrix} \end{array} \\
 \begin{array}{c} A_{\text{partial}} \\ \begin{bmatrix} -0.6 & -0.1 & 0.4 \\ 0.2 & 0 & -0.2 \end{bmatrix} \end{array}
 \end{array}$$

- We're building **A** as a linear combination of the columns of **U**
- Using all columns of **U**, we'll rebuild the original matrix perfectly
- But, in real-world data, often we can just use the first few columns of **U** and we'll get something close (e.g. the first **A_{partial}**, above)

SVD Applications

$$\begin{array}{c}
 U\Sigma \\
 \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{array}{c} V^T \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{array} \\
 \\
 U\Sigma \\
 \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{array}{c} V^T \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{array} \\
 \\
 A_{\text{partial}} \\
 \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix} \\
 \\
 A \\
 \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
 \end{array}$$

- We can call those first few columns of \mathbf{U} the *Principal Components* of the data
- They show the major patterns that can be added to produce the columns of the original matrix
- The rows of \mathbf{V}^T show how the *principal components* are mixed to produce the columns of the matrix

SVD Applications

$$\begin{matrix} U \\ \begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix} \end{matrix} \times \begin{matrix} \Sigma \\ \begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} \end{matrix} \times \begin{matrix} V^T \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{matrix} = \begin{matrix} A \\ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \end{matrix}$$

We can look at Σ to see that the first column has a large effect

while the second column has a much smaller effect in this example

Principal Component Analysis

$$\begin{matrix} U\Sigma \\ \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \end{matrix} \times \begin{matrix} V^T \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{matrix} = \begin{matrix} A_{\text{partial}} \\ \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix} \end{matrix}$$

- Remember, columns of \mathbf{U} are the *Principal Components* of the data: the major patterns that can be added to produce the columns of the original matrix
- One use of this is to construct a matrix where each column is a separate data sample
- Run SVD on that matrix, and look at the first few columns of \mathbf{U} to see patterns that are common among the columns
- This is called *Principal Component Analysis* (or PCA) of the data samples

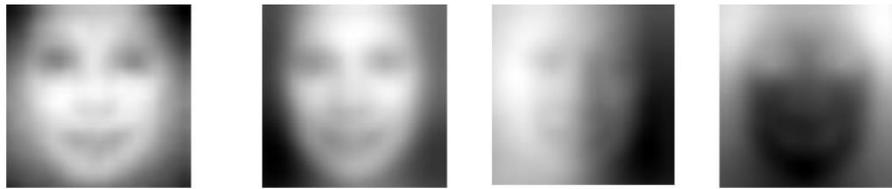
Principal Component Analysis

$$\begin{matrix} U\Sigma \\ \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \end{matrix} \times \begin{matrix} V^T \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{matrix} = \begin{matrix} A_{\text{partial}} \\ \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix} \end{matrix}$$

- Often, raw data samples have a lot of redundancy and patterns
- PCA can allow you to represent data samples as weights on the principal components, rather than using the original raw form of the data
- By representing each sample as just those weights, you can represent just the “meat” of what’s different between samples
- This minimal representation makes machine learning and other algorithms much more efficient

Example: Eigenfaces

- Images of faces: represent each as the concatenation of its pixels (column vectors)
- Stack together horizontally to get matrix A
- $[U, S, V] = \text{svd}(A)$;
- First four columns of U:



- Can represent each face as a linear combination of the first few columns of U

Addendum: How is SVD computed?

- For this class: tell MATLAB to do it. Use the result.
- But, if you're interested, one computer algorithm to do it makes use of Eigenvectors
 - The following material is presented to make SVD less of a “magical black box.” But you will do fine in this class if you treat SVD as a magical black box, as long as you remember its properties from the previous slides.

Eigenvector definition

- Suppose we have a square matrix \mathbf{A} . We can solve for vector x and scalar λ such that $Ax = \lambda x$
- In other words, find vectors where, if we transform them with \mathbf{A} , the only effect is to scale them with no change in direction
- These vectors are called eigenvectors (German for “self vector” of the matrix), and the scaling factors λ are called eigenvalues
- An $m \times m$ matrix will have $\leq m$ eigenvectors where λ is nonzero

Finding eigenvectors

- Computers can find an x such that $Ax = \lambda x$ using this iterative algorithm:
 - x = random unit vector
 - while(x hasn't converged)
 - $x = Ax$
 - normalize x
- x will quickly converge to an eigenvector
- Some simple modifications will let this algorithm find all eigenvectors

Finding SVD

- Eigenvectors are for square matrices, but SVD is for all matrices
- To do $\text{svd}(A)$, computers can do this:
 - Take eigenvectors of AA^T (matrix is always square).
 - These eigenvectors are the columns of \mathbf{U} .
 - Square root of eigenvalues are the singular values (the entries of $\mathbf{\Sigma}$).
 - Take eigenvectors of $A^T A$ (matrix is always square).
 - These eigenvectors are columns of \mathbf{V} (or rows of \mathbf{V}^T)

Finding SVD

- Moral of the story: SVD is fast, even for large matrices
- It's useful for a lot of stuff
- There are also other algorithms to compute SVD or part of the SVD
 - MATLAB's `svd()` command has options to efficiently compute only what you need, if performance becomes an issue

A detailed geometric explanation of SVD is here:

<http://www.ams.org/samplings/feature-column/fcarc-svd>

Matlab Tutorial

http://www.cs.pitt.edu/~kovashka/cs2750_sp17/tutorial.m

http://www.cs.pitt.edu/~kovashka/cs2750_sp17/myfunction.m

http://www.cs.pitt.edu/~kovashka/cs2750_sp17/myotherfunction.m

Please cover whatever we don't finish at home.

Tutorials and Exercises

- <https://people.cs.pitt.edu/~milos/courses/cs2750/Tutorial/>
- http://www.math.udel.edu/~braun/M349/Matlab_probs2.pdf
- <http://www.facstaff.bucknell.edu/maneval/heap211/basicexercises.html>
 - Do Problems 1-8, 12
 - Most also have solutions
 - Ask the TA if you have any problems