Linear Algebra/Matrix Calculus Recitation

Cathy Ma

Caltech CS/CNS/EE 155

Winter 2018

Linear Algebra Outline

We will review some basic linear algebra concepts that are useful in this course and cover how they can be applied using Numpy and Python. We will skip some concepts which are not relevant to the course, such as independence/dependence, rank, and linear maps.

- Basic Linear Algebra + Numpy Tools
  - Matrix and Vectors
  - Matrix Multiplication
  - Basic Operators
  - Special Matrices
  - Vector Spaces
  - Vector Norms
  - Eigenvalues and Eigenvectors

Matrix and Vectors

- A matrix A^{mxn} is a m by n rectangular array of numbers
- A column vector $x_C \in \mathbb{R}^m$
- A row vector $x_R \in \mathbb{R}^n$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, x_C = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, x_R = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}$$

Matrix and Vectors (code)

In Numpy:

- Matrix:

```
A = np.array([[1, 2], [3, 4]])
```

- Vector:

```
>>> np.array([1, 2, 3]).reshape((3,1)) # Column Vector
array([[1],
       [2],
       [3]])
>>> np.array([1, 2, 3]).reshape((1,3)) # Row Vector
array([[1, 2, 3]])
```

Matrix Multiplication

- If A^{mxn}, B^{nxp}, then $C^{mxp} = AB$ with

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Properties:

- Associativity:$(AB)C = A(BC)$
- Distributivity: $A(B + C) = AB + AC$
- No Commutativity: $AB \neq BA$

Matrix Multiplication (Code)

For matrix multiplication, np.dot is the best option. Note that np.multiply is not matrix multiplication, but rather element-wise multiplication. Also, np.matrix is unwieldy for > 2 dimensions, so we recommend using np.array.

```
>>> A = np.array([[1, 2, 3], [4, 5, 6]])
>>> A.shape
(2, 3)
>>> B = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10],
[11, 12, 13, 14, 15]])
>>> B.shape
(3, 5)
>>> np.dot(A, B).shape
(2, 5)
>>> np.dot(A,B)
array([[46, 52, 58, 64, 70],
[100, 115, 130, 145, 160]])
```

Basic Operators

Transpose: if $A \in \mathbb{R}^{m \times n}$, then $A^T \in \mathbb{R}^{n \times m}, (A^T)_{ij} = A_{ji}$

- $(A^T)^T = A$
- $(AB)^T = B^T A^T$
- $(A + B)^T = A^T + B^T$
- In Numpy:

```
>>> A = np.array([[1, 2, 3], [4, 5, 6]])
>>> A
array([[1, 2, 3],
[4, 5, 6]])
>>> A.T
array([[1, 4],
[2, 5],
[3, 6]])
```

Basic Operators (continued)

Trace: if $A \in \mathbb{R}^{n \times n}$, then $tr(A) = \sum_{i=1}^{n} A_{ii}$

- $tr(A + B) = tr(A) + tr(B)$
- In Numpy:

```
>>> A = np.array([[1, 2], [3, 4]])
>>> np.trace(A)
5
```

Basic Operators (continued)

A matrix $A \in \mathbb{R}^{nxn}$ is invertible if $\exists B \in \mathbb{R}^{nxn}$ such that $AB = BA = I$. B is called the inverse of A and is written as $B = A^{-1}$.

- $(A^{-1})^{-1} = A$
- $(AB)^{-1} = B^{-1}A^{-1}$
- $(A^{-1})^T = (A^T)^{-1}$
- In Numpy:

```
>>> A = np.array([[1, 2], [3, 4]])
>>> B = np.linalg.inv(A)
>>> B
array([[-2., 1.],
[1.5, -0.5]])
>>> np.dot(A, B)
array([[1, 0],
[0, 1]])
```

Basic Operators (continued)

What is the easiest way to check is a matrix A is invertible?

- Check $det(A)$. If $det(A) \neq 0$, then A is invertible.
- In Numpy:

```
>>> A = np.array([[1, 2], [3, 4]])
>>> np.linalg.det(A)
-2
```

- Invertible Matrix Theorem

Special Matrices

If $A \in \mathbb{R}^{n \times n}$,

- A is a **diagonal** matrix if all its non-diagonal entries are zero, $\forall i \neq j : A_{ij} = 0$
- A is a **symmetric** matrix if $A = A^T$
- A is an **orthogonal** matrix if $A^T = A^{-1}$ (i.e. $AA^T = A^T A = I$)

Vector Spaces

- A vector space is a set of vectors that is closed under addition and scalar multiplication.
- Vector spaces also satisfy other axioms. Given a vector space V, for any $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$:
  - Associativity of addition: $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$
  - Commutativity of addition: $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$
  - Identity element of addition: $\exists \mathbf{0} \in V$ such that $\mathbf{v} + \mathbf{0} = \mathbf{v}$
  - Inverse element of addition: $\exists -\mathbf{v}$ such that $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$
  - Associativity of scalar multiplication: $a(b\mathbf{v}) = (ab)\mathbf{v}$
  - Identity element of multiplication: $1\mathbf{v} = v$
  - Distributivity: $a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + b\mathbf{v}, (a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$
- $\mathbb{R}^n$ is a vector space.

Vector Norm

A norm of a vector space V is a function $||.|| : V \rightarrow R^{+}$ such that

- $||x|| = 0$ iff $x = 0$
- $||\alpha x|| = |\alpha| * ||x||$
- $||x + y|| \leq ||x|| + ||y||$
- In Numpy:

```
>>> x = np.array([1, 2, 3])
>>> np.linalg.norm(x, ord=None)
3.7416573867739413
>>> np.linalg.norm(x, ord=1)
6.0
```

- The "ord" parameter allows you to select what type of norm (l_2, l_1, etc.)
- defaults to the Frobenius norm (2-norm)

Vector Norm (continued)

The norm of a vector is a measure of its magnitude. Formally,

- l_p norm: $||x||_p = (\sum_{i=1}^{n} x_i^p)^{1/p}$

The l_2 and l_1 norms, which are used in Ridge and Lasso regularization respectively, are the most common.

Eigenvalues and Eigenvectors

Let $A \in \mathbb{R}^{n \times n}$. A scalar λ is called an **eigenvalue** of A if there exists $v \in \mathbb{R}^n$, $v \neq 0$, such that $Av = \lambda v$. Any vector $v \in \mathbb{R}^n$, $v \neq 0$, satisfying this equation is called an **eigenvector** associated with eigenvalue λ.

- We can rewrite the above equation as $(A - \lambda I_n)v = 0$.
- Strategy:
  - Find all λ for which the characteristic polynomial $p_A(\lambda) = det(A - \lambda I_n) = 0$.
  - For each λ, solve $(A - \lambda I_n)v = 0$ for v.
- In Numpy:

```
>>> A = np.array([[0.8, 0.3], [0.2, 0.7]])
>>> np.linalg.eig(A)[0]
array([1., 0.5])
>>> np.linalg.eig(A)[1]
array([[0.83205029, -0.70710678],
       [0.5547002, 0.70710678]])
```

Matrix Calculus Outline

- The Gradient
- The Hessian Matrix
- Gradient for Vector-Valued Functions
- Handy Matrix Differentiation Rules
- Useful Resources
- Examples

The Gradient

Consider a differentiable real function $f : \mathbb{R}^{mxn} \to \mathbb{R}$. Then, for $x \in \mathbb{R}^{mxn}$, the gradient $\nabla : \mathbb{R}^{mxn} \to \mathbb{R}^{mxn}$ of f is by definition

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_{11}} & \frac{\partial f(x)}{\partial x_{12}} & \cdots & \frac{\partial f(x)}{\partial x_{1n}} \\ \frac{\partial f(x)}{\partial x_{21}} & \frac{\partial f(x)}{\partial x_{22}} & \cdots & \frac{\partial f(x)}{\partial x_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(x)}{\partial x_{m1}} & \frac{\partial f(x)}{\partial x_{m2}} & \cdots & \frac{\partial f(x)}{\partial x_{mn}} \end{bmatrix}$$

In most cases in this course, x will be a vector $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$. In this case, the gradient $\nabla : \mathbb{R}^n \to \mathbb{R}^n$ of f is

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

The Hessian Matrix

Let x be a vector $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$. Then, the Hessian matrix is $\nabla^2 : \mathbb{R}^n \to \mathbb{R}^{nxn}$ of f is

$$\nabla_x^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

Gradient for Vector-Valued Functions

For $x \in \mathbb{R}$ and $v : \mathbb{R} \to \mathbb{R}^N$, the gradient $\nabla : \mathbb{R} \to \mathbb{R}^N$ of v is

$$\nabla_x v(x) = \begin{bmatrix} \frac{\partial v_1(x)}{\partial x} & \frac{\partial v_2(x)}{\partial x} & \cdots & \frac{\partial v_N(x)}{\partial x} \end{bmatrix}$$

For $x \in \mathbb{R}^K$ and $h : \mathbb{R}^K \to \mathbb{R}^N$, then the gradient $\nabla : \mathbb{R}^K \to \mathbb{R}^{NxK}$ of h is

$$\nabla_x h(x) = \begin{bmatrix} \frac{\partial h_1(x)}{\partial x_1} & \frac{\partial h_1(x)}{\partial x_2} & \cdots & \frac{\partial h_1(x)}{\partial x_K} \\ \frac{\partial h_2(x)}{\partial x_1} & \frac{\partial h_2(x)}{\partial x_2} & \cdots & \frac{\partial h_2(x)}{\partial x_K} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_N(x)}{\partial x_1} & \frac{\partial h_N(x)}{\partial x_2} & \cdots & \frac{\partial h_N(x)}{\partial x_K} \end{bmatrix}$$

Gradient for Vector-Valued Functions (continued)

There are two main notational conventions in matrix calculus.
For $x \in \mathbb{R}^K$ and $h : \mathbb{R}^K \to \mathbb{R}^N$ Then the gradient of h is
Numerator Layout : $\nabla : \mathbb{R}^K \to \mathbb{R}^{NxK}$

$$\nabla_x h(x) = \begin{bmatrix} \frac{\partial h_1(x)}{\partial x_1} & \frac{\partial h_1(x)}{\partial x_2} & \cdots & \frac{\partial h_1(x)}{\partial x_K} \\ \frac{\partial h_2(x)}{\partial x_1} & \frac{\partial h_2(x)}{\partial x_2} & \cdots & \frac{\partial h_2(x)}{\partial x_K} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_N(x)}{\partial x_1} & \frac{\partial h_N(x)}{\partial x_2} & \cdots & \frac{\partial h_N(x)}{\partial x_K} \end{bmatrix}$$

Denominator Layout : $\nabla : \mathbb{R}^K \to \mathbb{R}^{KxN}$

$$\nabla_x h(x) = \begin{bmatrix} \frac{\partial h_1(x)}{\partial x_1} & \frac{\partial h_2(x)}{\partial x_1} & \cdots & \frac{\partial h_N(x)}{\partial x_1} \\ \frac{\partial h_1(x)}{\partial x_2} & \frac{\partial h_2(x)}{\partial x_2} & \cdots & \frac{\partial h_N(x)}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_1(x)}{\partial x_K} & \frac{\partial h_2(x)}{\partial x_K} & \cdots & \frac{\partial h_N(x)}{\partial x_K} \end{bmatrix}$$

You can use whichever you prefer, but be consistent.

Handy Matrix Differentiation Rules

Note that we are using numerator layout.

- $y = Ax$ where A does not depend on x.

$$\frac{\partial y}{\partial x} = A$$

- $y = x^T A$ where A does not depend on x.

$$\frac{\partial y}{\partial x} = A^T$$

- $y = A^T x B$ where A and B do not depend on x.

$$\frac{\partial y}{\partial x} = AB^T$$

- $y = x^T Ax$ where A does not depend on x.

$$\frac{\partial y}{\partial x} = x^T(A + A^T)$$

Useful Resources

- All of the matrix calculus rules you need to know for this class can be found in "The Matrix Cookbook" (matrixcookbook.com).
- The Wikipedia article for matrix calculus is a very good introduction to this material.

Examples (Least Squares)

- Real valued function $f : \mathbb{R}^n \to \mathbb{R}$ where for $x \in \mathbb{R}^n$

$$
\begin{aligned}
f(x) &= ||Ax - b||_2^2 && (1)\\
&= (Ax - b)^T (Ax - b) && (2)\\
&= (x^T A^T - b^T)(Ax - b) && (3)\\
&= x^T A^T Ax - b^T Ax - x^T A^T b + b^T b && (4)
\end{aligned}
$$

- To find the x that minimizes f, we need to compute the gradient with respect to x.

$$
\begin{aligned}
\nabla_x f(x) &= \nabla_x (x^T A^T Ax - b^T Ax - x^T A^T b + b^T b) && (5)\\
&= \boldsymbol{2A^T Ax - 2A^T b} && (6)
\end{aligned}
$$

Examples (Vector Valued Functions)

- Consider the vector valued function $v : \mathbb{R} \to \mathbb{R}^K$ where for $x \in \mathbb{R}$,

$$v(x) =< x, x^2, \ldots, x^n >$$

Then,

$$\nabla_x v(x) =< 1, 2x, \ldots, nx^{n-1} >$$

- Consider the vector valued function $h : \mathbb{R}^N \to \mathbb{R}^K$ where for $x \in \mathbb{R}^N$,

$$h(x) = ABx + A^T xB$$

$$\nabla_x h(x) = AB + AB^T$$

Generally and for most problems in this class, the gradients of matrix products will only require applying basic matrix calculus rules.