Teaching Portfolio

Samuel Deng
samdeng@cs.columbia.edu
samuel-deng.github.io

In this document, I've collected a range of representative artifacts that I believe represent my teaching. In particular, I've attempted to collect material that I feel encapsulate the core principles of my **teaching philosophy**:

1. **A driving and cohesive narrative should propel all parts of a course.**
2. **Ideas should be presented as if the student could've discovered them themselves.**
3. **An instructor should never forget how they first struggled when learning the same ideas.**

Table of Contents

This document is quite large, so please sample whatever you find relevant and ignore whatever you do not. It should operate a bit like a webpage. All **blue** links lead to other parts of this document. All **orange** links lead to external links.

Summary of Courses and Evaluations

Below is a table that contains my “Overall Instructor Quality” for all the courses that I’ve TA’d or taught. These all come from end-of-semester anonymous teaching evaluations solicited by Columbia, and the scale given is: *(1) Poor (2) Fair (3) Good (4) Very Good (5) Excellent*.

Course	**Semester**	**Role**	**Overall Instructor Quality**	**Number of Respondents**	**Number of Students**
Math for ML	**Summer 2024**	Course Designer/ Insructor	4.83/5	7	30
Computational Linear Algebra	**Fall 2022**	Head TA	4.63/5	51	130
Natural and Artificial Neural Networks (Lab)	**Spring 2022**	Co-Course Designer/Co-Instructor *	5/5	3	15
Machine Learning	Summer 2020	Head TA			
Discrete Math	Spring 2020	Head TA			
Discrete Math	**Fall 2019**	Head TA	4.21/5	38	287
Machine Learning	**Spring 2019**	TA	4.83/5	18	259
Discrete Math	Fall 2018	TA			

- All courses I’ve designed have their materials all available online; just click the **orange** links to view all the materials.
- The **orange** link for Computational Linear Algebra directs to a **YouTube playlist** complete with recordings of all the weekly recitations I designed and taught throughout the semester, as well as a guest lecture I did on eigenvectors and eigenvalues.
- Click the **blue** links to be directed to the full set of evaluations for that semester’s class.
- The **greyed out** boxes were evaluations that I unfortunately couldn’t find in the system. The Spring 2020 in particular had no final evaluations because of the COVID-19 pandemic.
- * The Spring 2022 semester of Natural and Artificial Neural Networks was a companion “lab” session to a seminar titled Natural and Artificial Neural Networks. The instructors never figured out how to separate the lab session and list us as “Instructors” on the official listing, which is why our official evaluation designates us as “TAs.”

Jump To: Table of Contents

Summary of Teaching Awards, Certification, and Other Service

Awards

My work as a teacher has been recognized over the years by various awards and fellowships:

- **Teaching Assistant Fellowship (2019).** Awarded to "exceptional" teaching assistants in the computer science department, providing full funding for several semesters of my M.S.
- **Andrew P. Kosoresow Award for Excellence in Teaching and Service (2021).** Our computer science department's highest award for teaching, awarded to students "*for outstanding contributions to teaching [and exemplary service] in the Department.*
- **SEAS Doctoral Teaching Fellowship (2024).** School-wide, faculty-nominated fellowship awarded to PhD students who who have demonstrated "*excellence in teaching*," meant to allow students to further develop their pedagogy.

Six of the seven anonymous respondents in Math for ML also elected to nominate me for a SEAS Distinguished Faculty Award:

10 - Would you nominate this professor for the SEAS Distinguished Faculty Award?

Samuel Deng

Response Option	Weight	Frequency	Percent	Percent Responses	Means
Yes	(1)	6	85.71%		1.14
No	(2)	1	14.29%		
				0 25 50 100	Question

Response Rate	Mean	STD	Median
7/30 (23.33%)	1.14	0.38	1.00

Teaching Certification

Over the past four years, I've participated in Columbia's Center for Teaching and Learning's **Teaching Development Program (TDP),** an evidence-based, multi-year teaching certification program for PhD students across the university. The TDP focuses on cultivating, documenting, and reflecting upon evidence-based, student-centered teaching. I have completed the requirements for the fundamental **Foundational Track** and am slated to complete the **Advanced Track** early Summer 2025. The Advanced Track is the CTL's highest certification.

Teaching-related Service

My proudest service contribution has been my five semesters coordinating the **Emerging Scholars Program (ESP)**, Columbia's peer-led workshop and discussion seminar for first-year undergraduates. ESP provides introductory computer science students an opportunity to learn about a wide range of computer science topics beyond programming, build problem-solving confidence, receive personalized mentorship, and form close-knit peer groups. Its motivation stems from the recognition that computer science students come from a plethora of academic and personal backgrounds, and large introductory courses lack the close-knit environment that fosters connections with peers and group problem-solving. Alongside fellow PhD student Hadleigh Schwartz, each semester I led a team of eight to ten undergraduate teaching assistants and coordinated the program across as many as ten sessions of 100 total students.

Jump To: Table of Contents

Student Evaluations: Math for Machine Learning

Course: **Mathematics for Machine Learning** *(click to access all materials)*
Semester: Summer 2024
Role: Instructor/Course Designer
Course Size: 30

I have condensed all the responses into tables to save space; the original evaluation is available upon request.

Prompt	(1) Poor	(2) Fair	(3) Good	(4) Very Good	(5) Excellent	Mean	Resp. Rate
Course: Amount Learned	0	0	0	1	6	4.86/5	7
Course: Appropriateness of Workload	0	0	0	1	6	4.86/5	7
Course: Fairness of Grading Process	0	0	1	1	5	4.57/5	7
Course: Overall Quality	0	0	0	1	6	4.86/5	7
Instructor: Organization and Preparation	0	0	0	1	6	4.86/5	7
Instructor: Classroom Delivery	0	0	0	1	6	4.86/5	7
Instructor: Approachability	0	0	1	1	5	4.57/5	7
Instructor: Overall Quality	0	0	0	1	6	4.86/5	7

Responses to *"Enter any additional comments here*:"

- Sam is a tremendous lecturer; he is extremely knowledgeable, prepared, energetic, engaged, and accessible. This is course is marketed to students preparing for COMS 4771, but I think its value far exceeds just that individual course. Make no mistake, there is a ton of content covered in this course and it is probably better suited for a 12-week session, but this course is a tremendous value in that it cuts through the filler of at least three other standalone courses and gets us straight to the most important, fundamental aspects of ML math. With that said, in large part the course is manageable because of Sam - I really appreciate how thoroughly prepared he is, and the course website is among the best that I've seen. We were sort of guinea pigs in this inaugural cohort of ours, so naturally there were some typos and

Jump To: Table of Contents

errors in problem sets that needed to be cleaned up on the go, but Sam is approachable and it never felt burdensome to ask clarifying questions or make suggestions on the content, problem sets, or his delivery. He's a great lecturer, math courses can be hit or miss and often tedious - that was not the case, and if you missed lectures his recordings are just as clear and engaging as if you were in the classroom. All around, just a great job - he's going to be an awesome Professor, someday!

- The class was extremely well organized, starting from basics and leading to an overall understanding of bigger math concepts. All HW problems were helpful and well-guided - the problem sets were long but they were divided into smaller sections which reduced unnecessary time spent going the wrong way (it was very clear if I was going in the right direction). The coding assignments were also very clear and we could immediately see the results and learn from it. Unlike some other classes were coding homework feels distanced from the content, the coding part here well-matched the concepts discussed and the way it had explanations in between each sections of the coding helped with understanding the features we are building.

- This course have bolstered my confidence in approaching the material covered in machine learning.
- Sam is an excellent instructor, and this class was extremely enjoyable. I look forward to taking any other courses Sam prepares.

Additional question for this course was *"Would you nominate this professor for the SEAS Distinguished Faculty Award?"*

10 - Would you nominate this professor for the SEAS Distinguished Faculty Award?

Samuel Deng

Response Option	Weight	Frequency	Percent
Yes	(1)	6	85.71%
No	(2)	1	14.29%

Means: Question 1.14

Response Rate	Mean	STD	Median
7/30 (23.33%)	1.14	0.38	1.00

The answers to: *"If so, please explain why"*:

- He brings both energy and clear expectations to the classroom.
- Sam's as good as it gets and he's genuinely interested in how we're doing, what we're interested in, and how he can help us along our journey.
- It really felt like the instructor was prepared to teach the class - the contents were not only organized but it had story to it. It worked up its way to a bigger concept. He had amazing slides and each concepts were supported with examples that he clearly worked through in class. Since it was a summer class and not many people took it (thus had only one TA), there were not many office hours available compared to some other CS classes during the regular semesters. However, he was always available through Ed and scheduled extra office hours if students requested.

Jump To: Table of Contents

- Sam is a fantastic instructor, in and out of the classroom. His lectures are excellent, his course is interesting and necessary, and his is prepared with information beyond the scope of the class.

Additional Feedback: Math for Machine Learning

I have also included additional feedback I've received in the form of emails, an anonymous end-of-course survey I used to solicit more course-specific feedback, and even a **reddit post**.

Emails

LIONMAIL @COLUMBIA

Samuel Deng <sd3013@columbia.edu>

Thank you for the course!
2 messages

[redacted] Wed, Aug 7, 2024 at 5:39 PM
To: Samuel Deng <samdeng@cs.columbia.edu>

Hi Sam,

Just wanted to send a quick note to say thank you for an extremely well-designed and executed course. I haven't been able to audit in-person in the last few weeks of the course because of my internship, but I've been following along online. I can clearly see the dedication you put into your teaching --- and it's very much appreciated!

If you're amenable, I'd love to grab a coffee on or near campus to discuss future study opportunities and to get to know you better. Please let me know what your schedule looks like say, next week?

I've answered the SEAS survey.

Thanks so much!

[redacted]

LIONMAIL @COLUMBIA

Samuel Deng <sd3013@columbia.edu>

Thank you
2 messages

[redacted] Mon, Aug 12, 2024 at 5:19 PM
To: Samuel Deng <samdeng@cs.columbia.edu>

Hello Samuel,

I hope you are doing well given the end of the summer semester and the number of papers that need to be graded.

I wanted to send this email to thank you for this class. I really enjoyed it and thought that I learned a lot. While writing my final evaluation, I was actually amazed by how much more of the paper I understood. In the beginning it all truly looked like gibberish. But now, I could honestly follow what the authors were talking about and understand what computations were being made. I am more confident in my Linear Algebra, Calculus, and Probability and Statistics.

I'm sorry I couldn't see you during the last couple weeks. I got very sick after coming back from DC when we had the online week.

Hope to see you on campus sometime. Enjoy the rest of your summer.

Thank you again,
[redacted]

Jump To: Table of Contents

LIONMAIL @COLUMBIA

Samuel Deng <sd3013@columbia.edu>

Updates
3 messages

[redacted] Wed, Dec 4, 2024 at 12:24 PM
To: Samuel Deng <samdeng@cs.columbia.edu>

Hey Sam!
Saw your email about checking in. I didn't take 4771 but I am taking Foundations of Optimization with Santiago following your suggestion.

I can confidently say I would not have survived this class without preparation in Mathematics for ML. Without the time we spent formalizing and visualizing convexity, the material I am tackling now might as well be hieroglyphics. I think your emphasis on the intuition and visualization of convexity paid the biggest dividends.

How is your research at Berkeley going?

Cheers,
[redacted]

LIONMAIL @COLUMBIA

Samuel Deng <sd3013@columbia.edu>

ML feedback
2 messages

[redacted] Wed, Dec 4, 2024 at 4:13 PM
To: Samuel Deng <samdeng@cs.columbia.edu>

Hi Sam,

Thanks for reaching out!

I'd love to give feedback on your Math for ML course's prep for the COMS 4771 course.
I took it this semester with Prof. Verma. My high-level comments would be:

1. I think your course actually covered a lot more mathematical material than what was strictly required for COMS 4771 --- this may be because Prof. Verma skipped dimensionality reduction and graphical models units in its entirety. Either way, I think it's actually a positive that I felt "over-prepared" in terms of mathematical breadth.

2. Specifically, I think your course went into a lot more detail with regards to multivariate calculus and OLS than what was necessary for this semester's COMS 4771 course. Again, I think that's a good thing. Your probability and stats unit was just about right in terms of breadth and depth.

3. Overall, I think your course was incredibly good prep for ML. I definitely felt a lot more prepared mathematically, although some of Verma's problems were way too hard...

I'd love to fill out the survey.

Thanks!

[redacted]

Jump To: Table of Contents

Math For Machine Learning - COMS4995

5 messages

[redacted] Mon, Jul 22, 2024 at 11:50 AM
To: Robert Kramer <rk3281@columbia.edu>
Cc: Samuel Deng <sd3013@columbia.edu>

Hi Robert,

I hope you are doing well. I am reaching out to share my positive experience auditing the "Math for Machine Learning" course with Professor Samuel Deng.

This course has been incredible in helping me understand many foundational concepts necessary for "Machine Learning for Data Science". It would be an excellent recommendation for students with no solid mathematical foundation or those like me who took Linear Algebra, Calculus and Optimization over a decade ago.

The best part of the course is that it focuses on developing intuition, which helps students go deeper into other classes of the MS in Data Science Program. Professor Verma could even go much deeper in his lectures, as this course already covers many technical aspects that he had to spend time on.

I've copied Professor Deng, to whom I am grateful for allowing me to audit the course. I believe it's an outstanding recommendation for upcoming students.

Kind regards,
[redacted]

--
[redacted]

Robert Kramer <rk3281@columbia.edu> Mon, Jul 22, 2024 at 4:16 PM
[redacted]
Cc: Samuel Deng <sd3013@columbia.edu>

Hi [redacted]

Thank you for your email and for letting me know; I am very glad to hear the course went well!

Samuel, please feel free to let me know if you will be teaching the course in the future. I am happy to inform our students if you ever have any open seats, as I think some of our MS Data Science students would be very interested in possibly taking the class during the Fall semester, prior to taking their Machine Learning course.

Kindly,
Rob

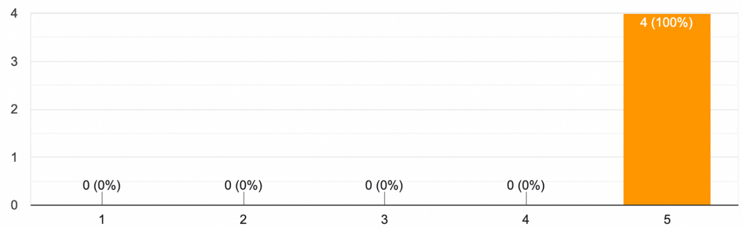
[Quoted text hidden]
--
Best,
Robert Kramer (he/his)
Associate Director of Admissions and Academic Affairs

End-of-course Survey

My foundations in linear algebra have improved.

Copy chart

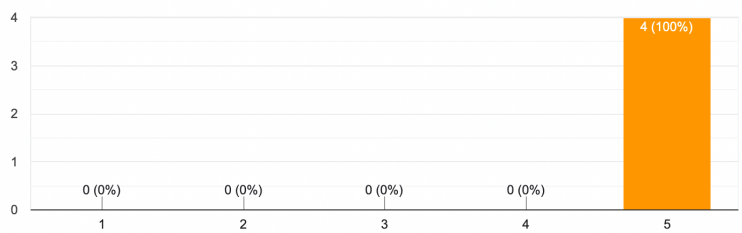
4 responses



My foundations in calculus and optimization have improved.

Copy chart

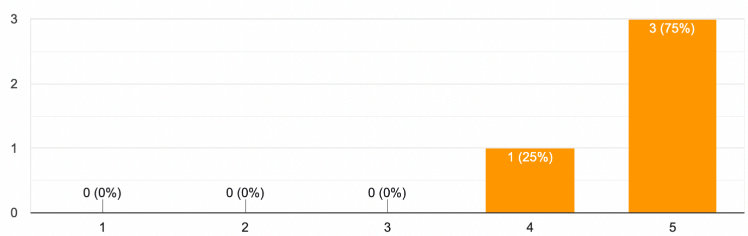
4 responses



My foundations in probability and statistics have improved.

Copy chart

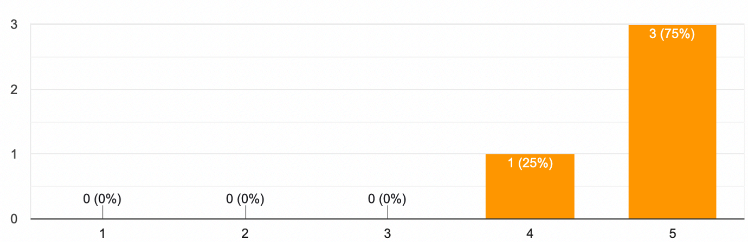
4 responses



I feel more confident with my mathematical foundations and tackling future ML courses or research.

Copy chart

4 responses



Jump To: Table of Contents

Reddit Post

r/columbia • 8 mo. ago
TheMostEquivocal

If offered again, don't pass up on COMS 'Math for ML'

academic tips

This is a PSA for potential future MSCS students interested in ML. If you ever have the opportunity to take Math for ML taught by Samuel Deng, do not pass up on it. I was among a lucky few to take this class over this summer, and I'll explain below. Sam might offer this class in 2025/2026.

Course Overview:

This is probably one of the best courses I have taken to develop a solid introductory understanding in convex optimization, and the groundwork for the math you will need to excel in any other machine learning class. Simply put, there is nothing like it. In fact, Sam created this class with the express goal of helping strengthen the core fundamentals you need for classes like 4771. While this course is meant to help 'prime' you with the necessary theory needed to excel in more advanced classes, it is not to be underestimated. It is *rigorous* in its coverage of fundamental concepts. Those concepts are the bedrock for building up to all the advanced ML in future classes. Sam goes into extensive detail about connecting several pillars in ML together to paint a cohesive picture on why we do what we do with models. What made the course special, was that Sam was there to guide us throughout the process. This class was the ideal balance of being challenging in a manner that is justified, motivated, inspiring, and (with a good work ethic) very doable.

As an instructor:

At the time of writing this, Sam is a PhD student. I can say quite confidently that he is probably of the best instructors I have personally encountered. I always found him super approachable and incredibly passionate. He went out of his way to help us understand concepts , going so far as to re-derive entire theorems in OH. The course itself is very well organized. It definitely puts you into a better position to understand how to approach things like research papers. To give you an idea of how committed he is, his lectures are accompanied with interactive 3D renderings of graphs he made himself, and problem sets that have entire expositions written to guide you through each concept. He has put in the work to give you the best experience you can have to learn as much as possible.

Workload:

Bear in mind this is likely subject to change depending on if/how Sam decides to re-create the course for a future semester schedule.

This is a challenging class.

- 6 problem sets

We write out all our PSets in Latex (something you will learn in PSet 0 if you have never done it before.) You will be deriving a lot of foundational proofs. Each PSet also includes a programming section in python. You need to put aside adequate time to understand the lectures and complete these questions. For some this may come easier than others (i.e you have great mathematical intuition these Psets will be readily manageable.) For many, it will take some grit and effort to work your way through them. Having said that, I felt motivated going through the PSets. They felt meaningful, and I learned a great deal.

- 2 paper evaluations

The paper evaluations are a way to encourage you to choose a paper and dissect it in some detail. It was a great exercise to better understand how to apply what we learnt in class to interpreting literature.

Overall:

I wanted to write this because I honestly felt so lucky to have taken this course and I really want more people to take it if it's offered in the future. Get your moneys worth, don't pass up on it if it's offered again.

Jump To: Table of Contents

Student Evaluations: Computational Linear Algebra

Course: Computational Linear Algebra *(click to access recitations and guest lecture)*
Semester: Fall 2022
Role: Head TA
Course Size: 130

I have condensed all the responses into tables to save space; the original evaluation is available upon request.

Prompt	(1) Poor	(2) Fair	(3) Good	(4) Very Good	(5) Excellent	Mean	Resp. Rate
Overall Quality	0	0	4	11	36	4.63/5	51
Knowledgeability	0	0	3	11	37	4.67/5	51
Approachability	0	1	3	11	36	4.67/5	51
Availability	0	3	5	10	32	4.42/5	50
Communication	0	1	6	8	35	4.54/5	50

Responses to *"Comments*:"

- Sam was easily one of the best TAs I've had at Columbia. He explained things in a clear and concise manner and was clearly very passionate about the subject. Attending his recitations was my favorite part of this class!
- Sam is a superstar. He was great in his recitations and the guest lecture he did. He's patient and a great communicator. If he's not on a professorial track, I hope he considers it. Also, as a commuting GS student, I was appreciative to have recitations available on Zoom and recorded.
- Excellent lecturer and very good at explaining tricky concepts in recitation
- Probably the best instructor and TA that I've had the pleasure of learning from. Very rarely does an instructor (professor or otherwise) come as well prepared in terms of lesson materials and knowledge whilst maintaining approachability and affability. Very responsive to questions and shares his thought process regarding topics at hand. If there is a TA of the year sticker Sam should definitely get it.
- TA Sam is the reason that I understood half of the material in this course. His explanations always made the most sense, and he really went above and beyond to make sure that we understood everything, through extra videos and lengthy Ed responses. I can't explain how grateful I am to have had Sam to help me understand CLA.
- Samuel is really good at teaching. Not only does he have the knowledge base, but he also has a very good energy about him while he's teaching that draws you into the material. Also, he can dumb things down "simple stupid" which make it easier to broadly grasp a concept before building upon its intricacies that make it complex.
- Informally, Sam is the best. I literally might have pulled the chute on this course if I didn't have him to pull me through this course kicking and screaming. Sam and I spent no less then

3ish hours every Friday going over course material. Don't get me wrong Daniel Hsu is great, but Sam could have absolutely taught this course for Daniel without an issue. 9.7/10.
- nice
- Very well prepared and patient!
- Always explained everything really well!
- Sam has been an incredible TA. He is super caring and knowledgeable, providing multiple ways to understand a topic.
- This man went above and beyond as a TA. He saw lots of messages on the Ed that a particular Problem Set was hard so he made a video giving a high level overview of the homework. Another Problem Set was hard and no one really understood the solutions (because Hsu releases solutions without work/explanation) so Deng made a video of him going through the solutions with work. Great TA. Was very responsive to the needs of the students!
- He's so knowledgeable, approachable, and friendly -- like no matter how silly a question may seem, he will answer it with patience and do his best to make sure you understand. His review sessions were life savers and he organizes his office hours so well so everyone who needs help will get help in a timely manner - overall one of the best TAs I've learned from!
- king
- INCREDIBLE! An amazing teacher. I wouldn’t have understood the material nearly as well if not for Sam. Thank you Sam!!! It was a pleasure!

Student Evaluations: Natural and Artificial Neural Networks Lab

Course: Natural and Artificial Neural Networks Lab *(click to access all materials)*
Semester: Spring 2022
Role: Co-Instructor/Co-Course Designer/TA[1]
Course Size: 15

I have condensed all the responses into tables to save space; the original evaluation is available upon request.

Prompt	(1) Poor	(2) Fair	(3) Good	(4) Very Good	(5) Excellent	Mean	Resp. Rate
Overall Quality	0	0	0	0	3	5/5	3
Knowledgeability	0	0	0	0	3	5/5	3
Approachability	0	0	0	0	3	5/5	3
Availability	0	0	0	0	3	5/5	3
Communication	0	0	0	0	3	5/5	3

Responses to *Comments:*

- great TA. really knows his stuff.

[1] With fellow PhD student Clayton Sanford. This was a companion two-hour “lab” course that we created all the materials and taught every week. Every session involved a short lecture and then an interactive Python “lab.” We also served as TAs to the main seminar course.

Jump To: Table of Contents

Student Evaluations: Discrete Mathematics

Course: Discrete Mathematics
Semester: Fall 2019
Role: Head TA
Course Size: 287

I have condensed all the responses into tables to save space; the original evaluation is available upon request.

Prompt	(1) Poor	(2) Fair	(3) Good	(4) Very Good	(5) Excellent	Mean	Resp. Rate
Overall Quality	1	0	9	8	20	4.21/5	38
Knowledgeability	0	1	7	6	21	4.34/5	35
Approachability	1	0	9	4	21	4.26/5	35
Availability	1	0	6	6	20	4.33/5	33
Communication	1	0	8	4	20	4.27/5	33

Responses to *Comments:*

- Very good at giving hints that don't give the answers away, very helpful and great teacher, all around cool guy
- Best TA ever!!
- He answered questions on piazza well.

Jump To: Table of Contents

Student Evaluations: Machine Learning

Course: Machine Learning
Semester: Spring 2019
Role: TA
Course Size: 259

I have condensed all the responses into tables to save space; the original evaluation is available upon request.

Prompt	(1) Poor	(2) Fair	(3) Good	(4) Very Good	(5) Excellent	Mean	Resp. Rate
Overall Quality	0	0	1	1	16	4.83/5	18
Knowledgeability	0	0	1	2	15	4.78/5	18
Approachability	0	0	1	1	16	4.83/5	18
Availability	0	0	1	1	16	4.83/5	18
Communication	0	0	1	1	16	4.83/5	18

Responses to *Comments:*

- Thank you!!!
- Good
- Thanks for helping with the homework!
- sammy d is my homie g five stars
- He was super friendly and approachable, always willing to help at office hours or even outside of class

Jump To: Table of Contents

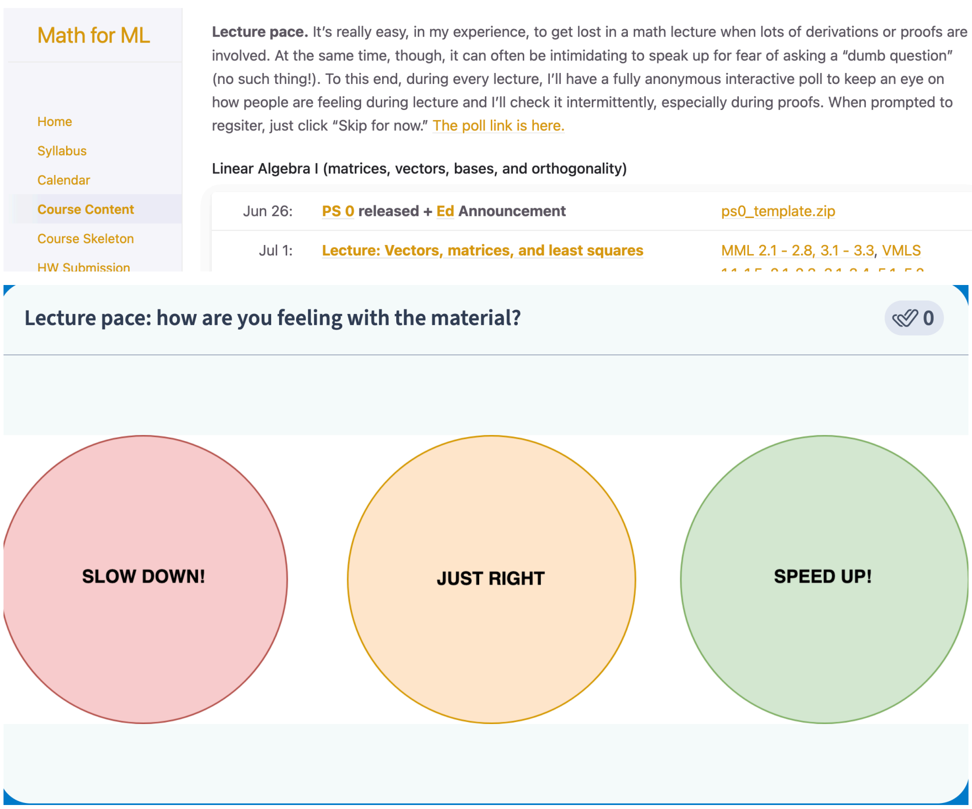
Math for Machine Learning Lectures

In this section, I present a 15 minute representative lecture of my teaching and give a broad overview of three representative lectures[2] of Math for ML that exhibit my teaching principle: **(1) A driving and cohesive narrative should propel all parts of a course.** If you'd like to access my course in its entirety:

- All lecture slides are available here.
- A complete YouTube playlist including all the recorded lectures is available here.

Red-light, yellow-light, green-light system

During lectures, one practice that embodies my teaching principle **(3) An instructor should never forget how they first struggled when learning the same ideas** is a "red-light, yellow-light, green-light system" I've developed for students.



I make this poll available at the start of each lecture for students to access on their phones.

On my PC, a synchronously updated version of this poll is within sight at all times.

This system comes from the understanding that, oftentimes, students may be insecure or shy about expressing confusion. It's greatly helped me calibrate the pacing during difficult sections of the class.

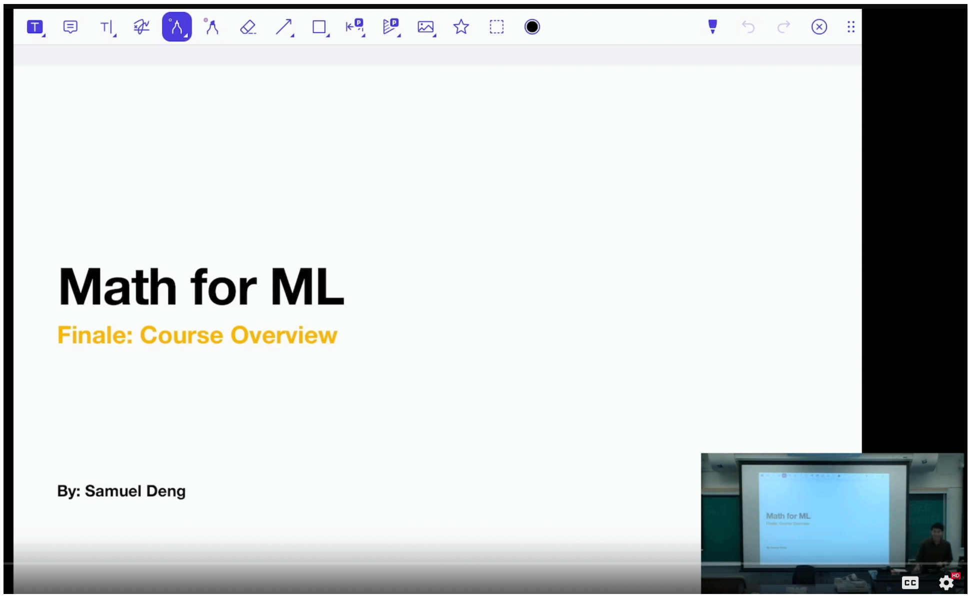
---

[2] Because this was a summer course, the classes were 3 hours long and included the content of two traditional class sessions.

Jump To: Table of Contents

15-minute Representative Lecture

Here is a link to a 15-minute representative lecture that was a part of my final session of Math for ML Summer 2024, Lecture 6.2: Multivariate Gaussian and Finale. This clip reviews the motivation behind the course and the key developments in the first third on linear algebra. **The full 3 hour lecture video can be found here.**



Overview: Three Representative Lectures

For more detail, I'll present a broad overview of three representative lectures that show how I spin a narrative around a central idea of the course: ordinary least squares. From the syllabus:

> *This is a course with a loose story. The course is structured around two main ideas that underlie modern machine learning: least squares regression and gradient descent. Very informally, least squares regression is a classic way of modeling problems in machine learning (the "what"), and gradient descent is the workhorse algorithm that drives much of modern machine learning (the "how"). Every week, we'll develop and motivate these two ideas in lecture with the tools and concepts you learn from each part of the course. As the class goes on, you'll develop different perspectives on these two ideas from, first, what we learn in linear algebra, then calculus and optimization, and, finally, probability and statistics. The hope is that, by the end of the course, you'll have a deep understanding of both these ideas in ML while also having two concrete "applications" to motivate all the abstract mathematical tools and concepts you learn in the course.*

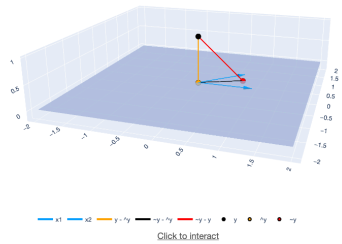
The three representative lectures are:

1. Lecture 1.1: Vectors, Matrices, and Least Squares (video, slides)
2. Lecture 3.1: Differentiation and Vector Calculus (video, slides)
3. Lecture 4.2: Convexity and Convex Optimization (video, slides)

Jump To: Table of Contents

Lesson Overview

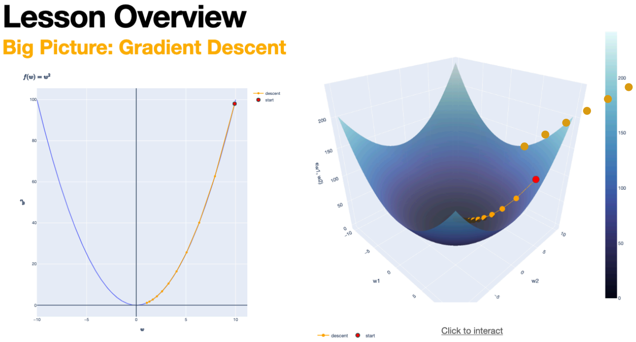
Big Picture: Least Squares



[Lecture 1.1: Vectors, Matrices, and Least Squares]() is the very first lecture.

Lesson Overview

Big Picture: Gradient Descent



Every lesson begins with an updated "big picture" of the two main narratives of the course: least squares and gradient descent.

All of the 3D renderings are [available]() for students to play with in the "Story Thus Far" sections of Course Content.

Ordinary Least Squares

Main Theorem FULL THRM.

Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ with $n \geq d$ and $\text{rank}(\mathbf{X}) = d$ (the columns of $\mathbf{X}$ are linearly independent.

Then, the solution $\hat{\mathbf{w}} \in \mathbb{R}^d$ that minimizes $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|$, i.e.

$$\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}\| \leq \|\mathbf{X}\mathbf{w} - \mathbf{y}\| \text{ for all } \mathbf{w} \in \mathbb{R}^d,$$

is given by:

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

$$(X^T X)\hat{w} = X^T y$$

In this lesson, students prove a solution to ordinary least squares purely from geometric intuition and linear algebra.

To arrive at this, I continually reference this [3D rendering]().

Lesson Overview

Takeaways

Regression. The basic problem in machine learning is regression. We have *training data* in the form of a data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ and labels $\mathbf{y} \in \mathbb{R}^n$. We seek a model $\hat{\mathbf{w}} \in \mathbb{R}^d$ such that $\mathbf{X}\hat{\mathbf{w}} \approx \mathbf{y}$.

Least squares. One way to find a model for the data is through *least squares*: choose $\hat{\mathbf{w}}$ that minimizes $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$.

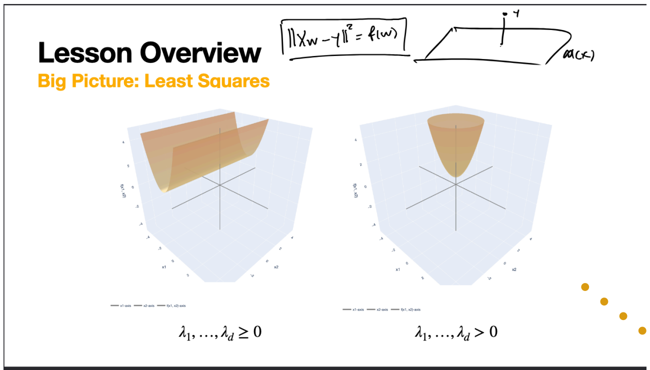
Span and orthogonality. We can solve least squares by noticing that $\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}$ is *orthogonal* to $\text{span}(\text{cols}(\mathbf{X}))$. This gives us the normal equations: $\mathbf{X}^\top\mathbf{X}\hat{\mathbf{w}} = \mathbf{X}^\top\mathbf{y}$.

Linear independence. To solve the normal equations, we need $\mathbf{X}$ to be full *rank* (its d columns are *linearly independent*). Then, we can invert and solve the normal equations.

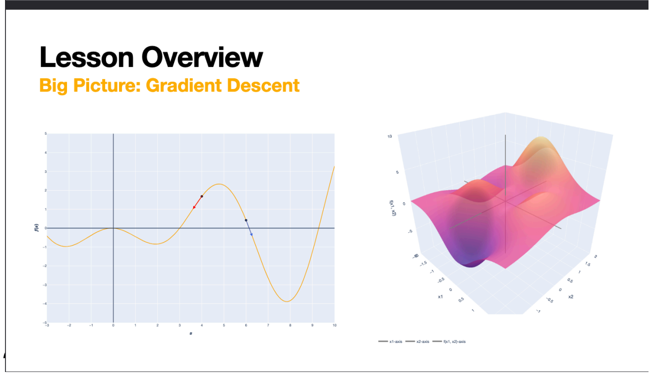
$$\hat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}.$$

I close every lesson with a recap of the important concepts learned. These are tracked in an ongoing [course skeleton]().

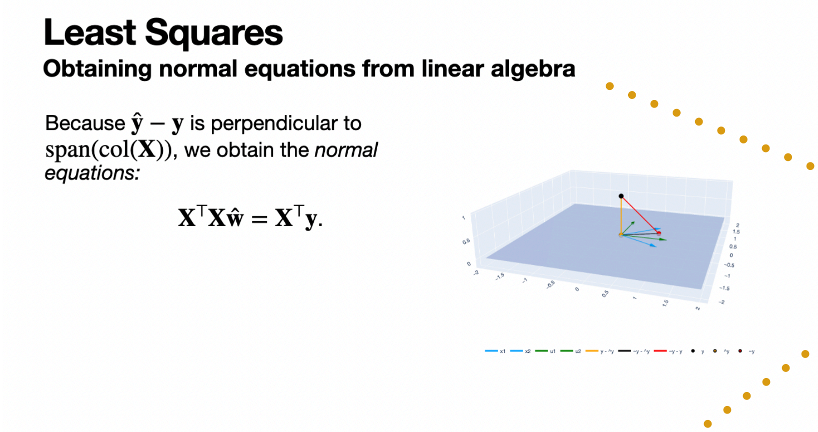
[Jump To: Table of Contents]()



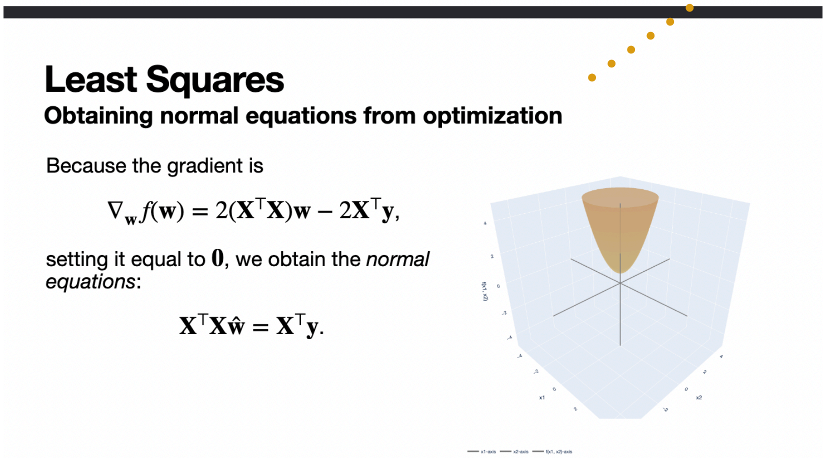
Lecture 3.1: Differentiation and Vector Calculus is the first lecture of the second third of the course on calculus and optimization.



At this point, the linear algebra third of the course has finished. I've hinted at this **"bowl-shaped" representation** of the least squares error function, but students don't have the formal tools (yet) to analyze it. This lecture will give them these tools.



The first "narrative" of the course takes a twist: least squares can be solved either: completely linear algebraically using pure geometric intuition or using the tools of calculus!



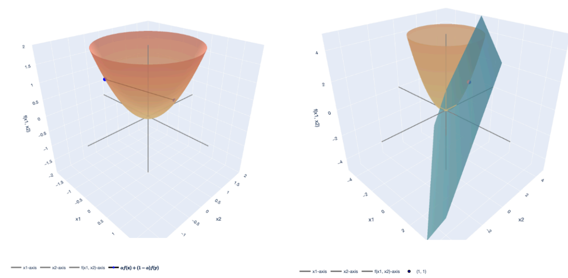
By the end of the course, the goal is for students to be able to see *least squares* and *gradient descent* from as many perspectives as possible.

These perspectives motivate which "characters" I introduce each lecture: they see a *gradient* for the first time in service of discovering a bit more about least squares.

Jump To: Table of Contents

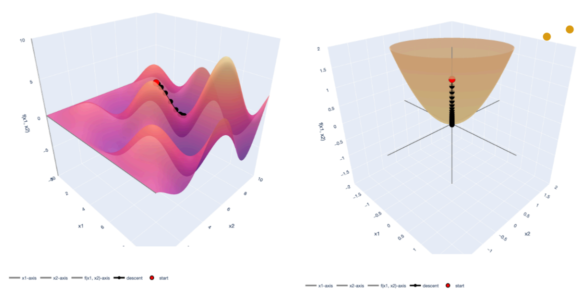
Lesson Overview

Big Picture: Least Squares



Lesson Overview

Big Picture: Gradient Descent



Gradient Descent and OLS

Uniting our two stories

Theorem (GD applied to OLS). Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{y} \in \mathbb{R}^n$ be fixed. Let the maximum eigenvalue $\lambda_{\max}$ of $\mathbf{X}^\top \mathbf{X}$ satisfy $\lambda_{\max} \leq \beta/2$. Let $\mathbf{w}^*$ be a (global) minimizer of $f(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$, satisfying:

$$\|\mathbf{X}\mathbf{w}^* - \mathbf{y}\|^2 \leq \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 \text{ for all } \mathbf{w} \in \mathbb{R}^d.$$

If we run gradient descent with step size $\eta = 1/\beta$ and initial point $\mathbf{w}_0 \in \mathbb{R}^d$ for T iterations, we have:

$$\|\mathbf{X}\mathbf{w}_T - \mathbf{y}\|^2 - \|\mathbf{X}\mathbf{w}^* - \mathbf{y}\|^2 \leq \frac{\beta}{2T}\left(\|\mathbf{w}_0 - \mathbf{w}^*\|^2 - \|\mathbf{w}_T - \mathbf{w}^*\|^2\right).$$

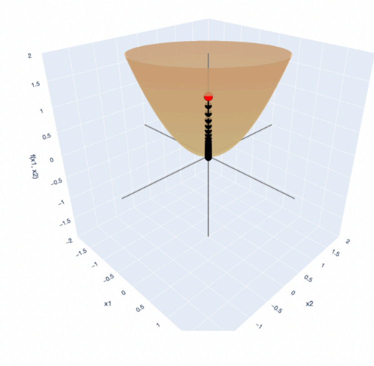
Gradient Descent

Algorithm for OLS

Make an initial guess $\mathbf{w}_0$.

For $t = 1,2,3,\ldots$

- Compute:
  $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - 2\eta \mathbf{X}^\top\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$.
- Stopping condition: If $\|\mathbf{w}_t - \mathbf{w}_{t-1}\| \leq \epsilon$, then return $f(\mathbf{w}_t)$.



Lecture 4.2: Convexity and Convex Optimization is the final lecture of the calculus and optimization unit of the course.

The big picture slides now hint that our least squares picture is showing up in a "crossover" with gradient descent.

The second third of the course culminates in the two stories of the course coming together: *gradient descent* applied to *least squares.*

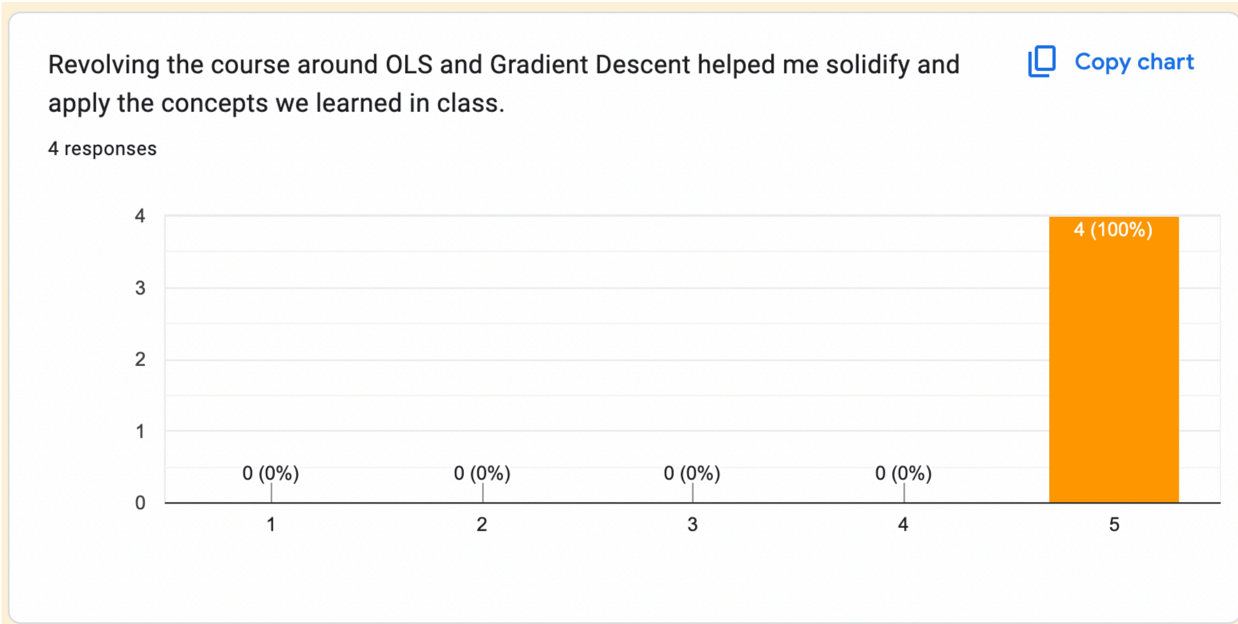
From the very first lecture, I hinted at the algorithm of gradient descent purely with hand-wavy intuition: "rolling a marble down a bowl."

This lecture gives students the mathematical tools to prove why gradient descent converges, and, specifically, why it works so well with least squares. Students investigate this connection further in **Problem Set 4**.

One student reported that their "mind was blown" at this, which is all I can ask for.

Jump To: Table of Contents

This philosophy that **(1) A driving and cohesive narrative should propel all parts of a course** was well-received by students in an end-of-semester anonymous survey. All four respondents appreciated this overarching narrative.



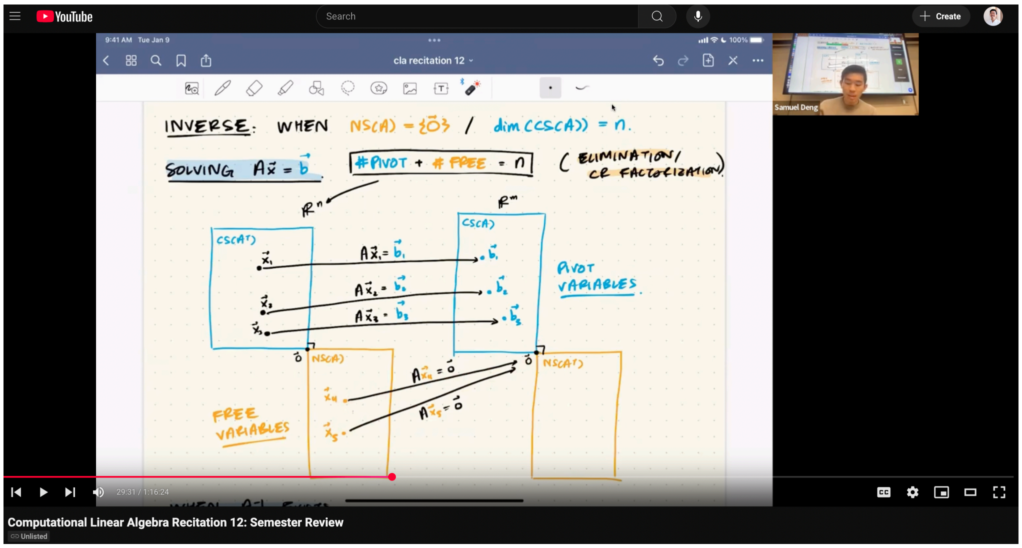
This was a broad overview of three lectures at various points through the semester. For more details on how I structure my material for an *individual lecture,* jump to **Lecture Slides: Math for ML (Subspaces, bases, orthogonality)**.

Jump To: Table of Contents

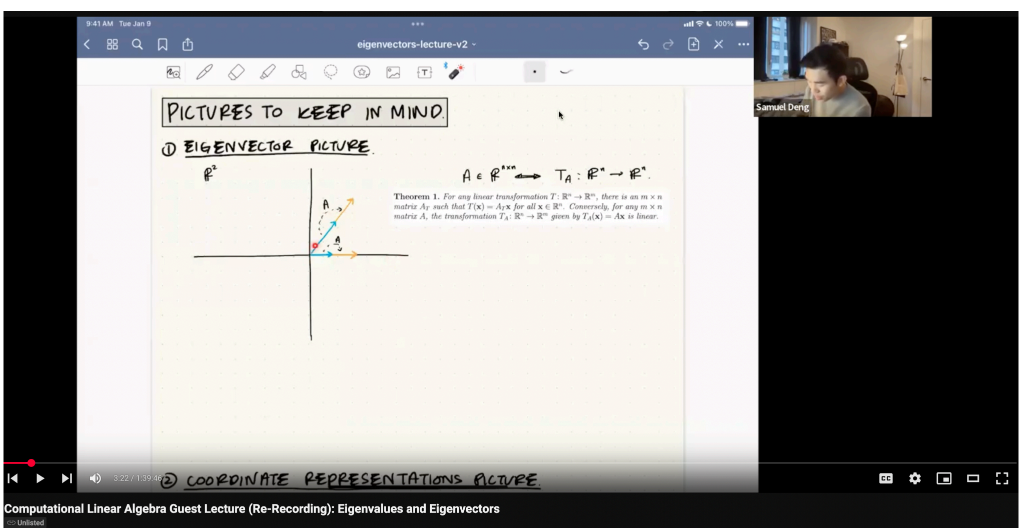
Computational Linear Algebra Recitations and Guest Lecture

This section includes some video recordings of my teaching during Computational Linear Algebra (CLA), where I held a weekly recitation session and delivered a guest lecture. [A playlist of all my CLA teaching can be found here.]

I'm particularly proud of my [final recitation lecture] for CLA, where I tied together each unit of the class into an overarching "big picture" revolving around the four fundamental subspaces.



I also took the opportunity to have my teaching observed and critiqued from the Center for Teaching and Learning as part of their Teaching Development Program's observation requirement. This happened during my [guest lecture] on eigenvalues and eigenvectors. Unfortunately, the sound didn't pick up in lecture, so this is a re-recording of the same content.



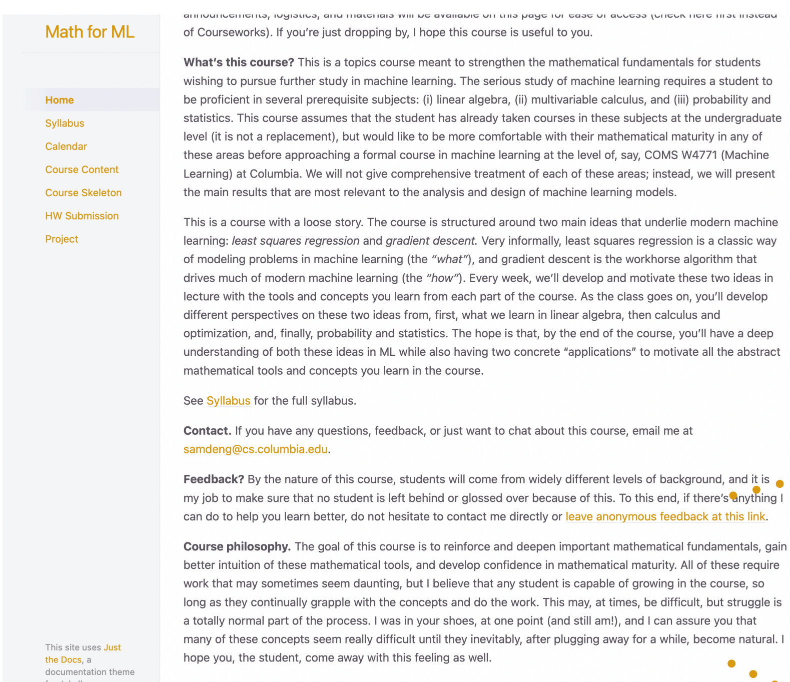
[Jump To: Table of Contents]

Math for Machine Learning Syllabus

This section includes a few annotated snippets of my syllabus and course website for Math for ML. A couple details on how this course came to be:

- I actually had the inkling of an idea for this course in my undergraduate senior year, Fall 2018, after I somehow hobbled through our Machine Learning course without ever taking probability and statistics. It was brutal, and conversations with peers from that point on showed me that I was not alone: many undergraduates and Master's students at Columbia felt that the jump from math prerequisites to our flagship ML course is too steep.
- Recognizing this as a pain point in our curriculum, I began constructing the course in earnest in Fall 2023, eventually ending up with this rationale.
- When I proposed this rationale to some faculty responsible for the undergraduate curriculum in the department, I was pleasantly surprised that this has been on their mind for a while, but no one had taken the initiative to do it.
- I decided to take the leap and create the course through Fall 2023 and Spring 2024, and I piloted the course during Summer 2024 under the SEAS Teaching Fellowship.

I didn't quite have the words then to express this, but, upon reflecting on this now, I was really motivated by my third teaching principle pervades every design decision in this course: **(3) An instructor should never forget how they first struggled when learning the same ideas.** I figured: if I could go from not knowing what an expectation is while taking Machine Learning to finishing a PhD in theoretical machine learning, I'm sure others could too. They just need the right preparation.



Math for ML

Home

Syllabus

Calendar

Course Content

Course Skeleton

HW Submission

Project

of Courseworks). If you're just dropping by, I hope this course is useful to you.

What's this course? This is a topics course meant to strengthen the mathematical fundamentals for students wishing to pursue further study in machine learning. The serious study of machine learning requires a student to be proficient in several prerequisite subjects: (i) linear algebra, (ii) multivariable calculus, and (iii) probability and statistics. This course assumes that the student has already taken courses in these subjects at the undergraduate level (it is not a replacement), but would like to be more comfortable with their mathematical maturity in any of these areas before approaching a formal course in machine learning at the level of, say, COMS W4771 (Machine Learning) at Columbia. We will not give comprehensive treatment of each of these areas; instead, we will present the main results that are most relevant to the analysis and design of machine learning models.

This is a course with a loose story. The course is structured around two main ideas that underlie modern machine learning: *least squares regression* and *gradient descent*. Very informally, least squares regression is a classic way of modeling problems in machine learning (the *"what"*), and gradient descent is the workhorse algorithm that drives much of modern machine learning (the *"how"*). Every week, we'll develop and motivate these two ideas in lecture with the tools and concepts you learn from each part of the course. As the class goes on, you'll develop different perspectives on these two ideas from, first, what we learn in linear algebra, then calculus and optimization, and, finally, probability and statistics. The hope is that, by the end of the course, you'll have a deep understanding of both these ideas in ML while also having two concrete "applications" to motivate all the abstract mathematical tools and concepts you learn in the course.

See Syllabus for the full syllabus.

Contact. If you have any questions, feedback, or just want to chat about this course, email me at samdeng@cs.columbia.edu.

Feedback? By the nature of this course, students will come from widely different levels of background, and it is my job to make sure that no student is left behind or glossed over because of this. To this end, if there's anything I can do to help you learn better, do not hesitate to contact me directly or leave anonymous feedback at this link.

Course philosophy. The goal of this course is to reinforce and deepen important mathematical fundamentals, gain better intuition of these mathematical tools, and develop confidence in mathematical maturity. All of these require work that may sometimes seem daunting, but I believe that any student is capable of growing in the course, so long as they continually grapple with the concepts and do the work. This may, at times, be difficult, but struggle is a totally normal part of the process. I was in your shoes, at one point (and still am!), and I can assure you that many of these concepts seem really difficult until they inevitably, after plugging away for a while, become natural. I hope you, the student, come away with this feeling as well.

This site uses Just the Docs, a documentation theme

My full syllabus and content is available online. This has actually led to some fortuitous consequences: educators at other institutions have reached out to talk about the course and its use at their schools.

An anonymous feedback form is open from day one to make sure I can adjust to the backgrounds of the class.

I make sure that the philosophy in designing the course is clear to the students.

Jump To: Table of Contents

Linear Algebra I (matrices, vectors, bases, and orthogonality)

Jun 26:	**PS 0 released + Ed Announcement**	ps0_template.zip
Jul 1:	**Lecture: Vectors, matrices, and least squares**	MML 2.1 - 2.8, 3.1 - 3.3, VMLS 1.1-1.5, 2.1-2.3, 3.1-3.4, 5.1, 5.2, 6.1-6.4, 12.1-12.4, Regression (d=2)
Jul 2:	**PS 1 released, due July 11, 11:59 PM ET** Paper reading project released. Evaluation due July 8 11:59 PM ET	ps1.pdf, ps1_template.zip, ps1.ipynb, ps1_tex.zip
Jul 3:	**Lecture: Subspaces, bases, and orthogonality**	MML 2.1 - 2.8, 3.1 - 3.3, VMLS 1.1-1.5, 2.1-2.3, 3.1-3.4, 5.1, 5.2, 6.1-6.4, 12.1-12.4, Alternate basis, 3Blue1Brown video on bases, 3Blue1Brown video on matrices as linear transformations
Jul 4:	DUE **PS 0 due**	
LS (Story thus far):	Lecture 1.1: Least squares regression can be solved geometrically with the Pythagorean Theorem. Lecture 1.2: Least squares regression has a simpler solution with orthonormal bases.	
GD (Story thus far):	Lecture 1.1, 1.2: Gradient descent with a "bowl-shaped" function gets us to the minimum.	

Linear Algebra II (singular value decomposition and eigendecomposition)

Calculus and Optimization I (differentiation and Taylor Series)

Jul 15:	**Lecture: Differentiation and vector calculus**	"Peaks" Function, Derivative Ex. 1, Derivative Ex. 2, Derivative Ex. 3, MML 5.1 - 5.5, The Matrix Cookbook
Jul 17:	**Lecture: Taylor Series, Linearization, and Gradient Descent**	GD Example 1 (big eta), GD Example 1 (small eta), GD Example 2 (big eta), GD Example 2 (small eta), Linearization in 3D, Polynomial 1, Polynomial 2, Beta-smooth function, 3Blue1Brown video on Taylor Series
Jul 18:	**PS 3 released, due July 29, 11:59 PM ET**	ps3.pdf, ps3_template.zip, ps3.ipynb, ps3_tex.zip
LS (Story thus far):	Lecture 3.1, 3.2: We can derive the exact same OLS theorem from linear algebra section from just the tools of optimization and viewing the notion of least squares error as an "objective function."	
GD (Story thus far):	Lecture 3.1: We can now write down the *algorithm* for gradient descent. Intuitively, positive semidefinite or positive definite quadratic forms seem good for gradient descent. Lecture 3.2: Using Taylor's approximations and Taylor's theorem for the first-order approximation (linearization), we can provide intuition and a formal guarantee that gradient descent makes the function values decrease. The behavior of gradient descent depends on the learning rate eta: eta too big will result in erratic behavior but small enough eta results in stable convergence.	

Calculus and Optimization II (optimization and convexity) -- SAM OUT OF TOWN

Jul 22:	**Lecture: Optimization and the Lagrangian** (recording in	Constrained least squares (ridge

The course is divided into three main parts: linear algebra, calculus and optimization, and probability and statistics.

Each lecture develops the two driving narratives of the course: *least squares* and *gradient descent.*

I visually summarize how the concept develops with a 3D rendered "big picture" that each lecture centers around.

To my delight, these 3D renderings were quite popular. One student even was able to spontaneously come up with the idea of a saddle point and better understand Lagrangian duality by playing with this visualization in office hours.

The second third on calculus and optimization builds on linear algebra by first showing that least squares can also be solved via optimization, and, by lecture 4.2, with gradient descent.

See the representative video lectures in the Math for ML Lectures section for more details on this progression.

Jump To: Table of Contents

Probability and Statistics I (basic probability theory and statistical estimation)

Jul 29:	**Lecture: Basic Probability Theory, Models, and Data** **DUE** **PS 3 due**	Regression setup w/ randomness, MML 6.1-6.4, Blitzstein and Hwang's Ch. 9 on Conditional Expectation
Jul 31:	**Lecture: Bias, Variance, and Statistical Estimators** Final paper reading evaluation released. Evaluation due August 12 11:59 PM ET	Regression (d = 2) with test point, SGD with batch size 1, SGD with batch size 10
Aug 1:	**PS 5 released, due Aug 13th, 11:59 PM ET (no programming portion)**	ps5.pdf, ps5_template.zip, ps5_tex.zip
LS (Story thus far):	Lecture 5.1: Modeled the regression problem with a linear model with random errors. Found that OLS' conditional expectation is the true linear model and its variance scales with the variance of the random errors. Lecture 5.2: OLS is the lowest variance *unbiased* linear estimator (Gauss-Markov Theorem). Derived expression for the risk (generalization error) of OLS.	
GD (Story thus far):	Lecture 5.1: Nothing new here. Lecture 5.2: Closed the story of gradient descent by defining *stochastic gradient descent*, where we use unbiased estimators of the gradient instead of the full gradient over all the data.	

Probability and Statistics II (Maximum likelihood and Gaussian distribution)

Aug 5:	**Lecture: The Central Limit Theorem, "Named" Distributions, and MLE**	MML 6.1-6.8, MML Ch. 8, 3Blue1Brown's video on the Central

In the last third of the course on probability and statistics, students finally gain the tools to ground the epistemic assumption of "random" data in the machine learning setup they've examined all class.

Conveniently, least squares is a pretty deep concept statistically: it shows up as maximum likelihood estimation under certain assumptions, and it provides nice analytic solutions that demonstrate key concepts like bias and variance.

This "inevitability" of least squares drives the last third of the class.

Math for ML

Courseworks Video Recordings Gradescope Ed Anonymous Feedback

- Home
- Syllabus
- Calendar
- Course Content
- Course Skeleton
- HW Submission
- **Project**

Paper Reading Project

1. Paper Reading Guidelines
2. List of Papers
3. Final Evaluation Outline
4. First Evaluation Outline

The project of this course will be to *attempt* to read a research paper in machine learning. Emphasis on *attempt*: there is no expectation that you will understand every single detail in the paper. However, you might be pleasantly surprised that you understand a bit more than you would've at the beginning of the course just by strengthening your mathematical foundations.

There are three parts to this project:

1. **Choose a paper.** *Within the first week.* Take a look at the list of research papers below and choose a paper based on the title and abstract. You're free to choose whatever might look interesting to you. If you need help deciding, feel free to email the instructor or TA or post on Ed.
2. **Beginning of course evaluation.** *Before the second week.* You will attempt to read the whole paper. Research papers can be intimidating if you've never read one before (and even if you've read hundreds!) so we will provide some guidance on how to read a scientific paper in machine learning. Then, you will provide a critical evaluation of the paper to the best of your current ability based on the template below. This will be graded on completion and effort – we emphasize that it does *not* matter how much you actually know or understand from the paper, just that you put a concerted effort into completing the evaluation and grappling with the paper.
3. **End of course evaluation.** *Final week.* At the end of the course, you will read the paper again. You will fill out a similar critical evaluation of the paper, per the same template, with a couple added questions. Again, you will be graded not on your understanding (though we hope it's improved the second time around!) but, rather, your concerted effort in writing an evaluation that shows that you've read and grappled with the paper to the best of *your* ability.

This project will be graded on the clarity and quality of the evaluation, but we stress that we will not focus on how much you "get" the paper. As long as you do the work of grappling with the paper and filling out the evaluation to the best of *your* ability, regardless of your understanding, you should get full marks, grade-wise. The emphasis of this project is on your own growth — hopefully, you'll find that by the end of the course your chosen paper isn't

This site uses Just the Docs, a documentation theme for Jekyll.

The summer version of this course involves a paper reading project.

The intended goal of this project is to show students how far they've come with mathematical maturity.

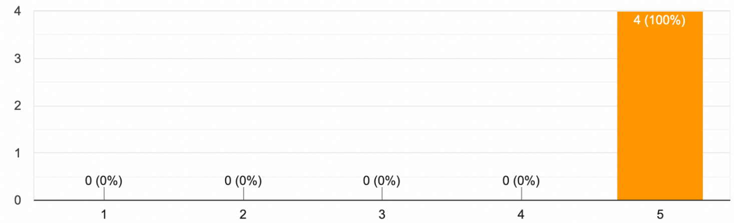
One student emailed me: *I was actually amazed by how much more of the paper I understood. In the beginning it all truly looked like gibberish. But now, I could honestly follow what the authors were talking about and understand what computations were being made.*

Jump To: Table of Contents

My foundations in linear algebra have improved.

Copy chart

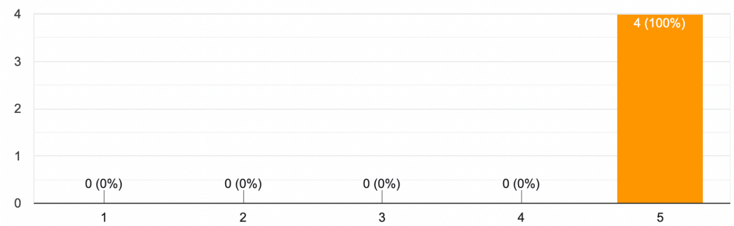
4 responses



My foundations in calculus and optimization have improved.

Copy chart

4 responses

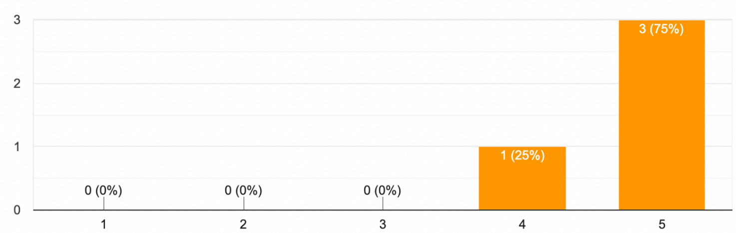


I created a course-specific feedback survey at the end of the course to solicit feedback on each part of the course.

My foundations in probability and statistics have improved.

Copy chart

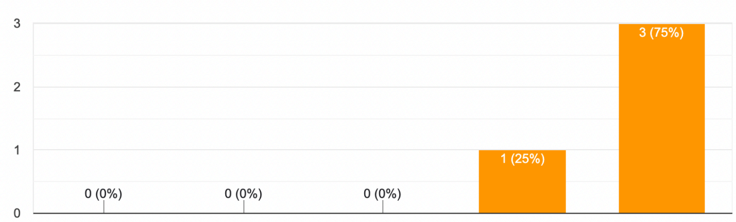
4 responses



I feel more confident with my mathematical foundations and tackling future ML courses or research.

Copy chart

4 responses

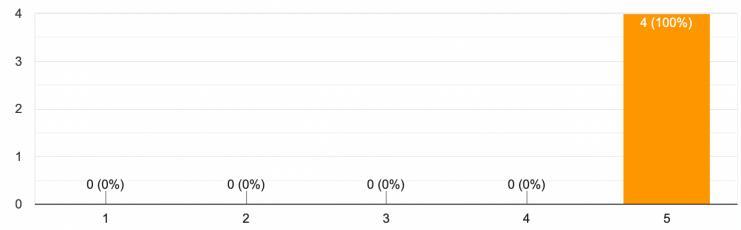


Jump To: Table of Contents

The paper reading project was a valuable exercise.

Copy chart

4 responses



In my opinion, the strength(s) of this class compared to other classes were:

3 responses

A strong emphasis on visualization, examples, and case-study type questions.

It was a clearly presented, important information. I enjoyed learning.

The Problem Sets. I think the problem sets were great. Explained the topic being used. Helped us derive the main concept from the basics.

I also made sure to get students' opinions on what might change in a future iteration.

In my opinion, the weakness(es) of this class compared to other classes were:

3 responses

Not really a weakness, but no matter how you cut it, this class was fast paced. Some of the later lectures were definitely harder to grasp at the pace.

It is a significant workload, especially for those with a rusty background in Linear Algebra. This may be due to summer having an expedited schedule.

Too quick. However that is a consequence of the summer course not the class itself.

A particular point that came up multiple times was that the accelerated summer schedule (4 full-length lectures a week in two 3-hour sessions) made the course particularly intense.

For me, the most significant obstacle(s) to my learning in this class were:

3 responses

Having time set aside to explore some of the theorems and concepts. There was little time for self exploration since the problem sets had to be immediately started. This is mostly a consequence of this being a summer class.

Time spent on problem sets. Again, summer schedule may be the cause.

Getting more confident with applying mathematical theorems.

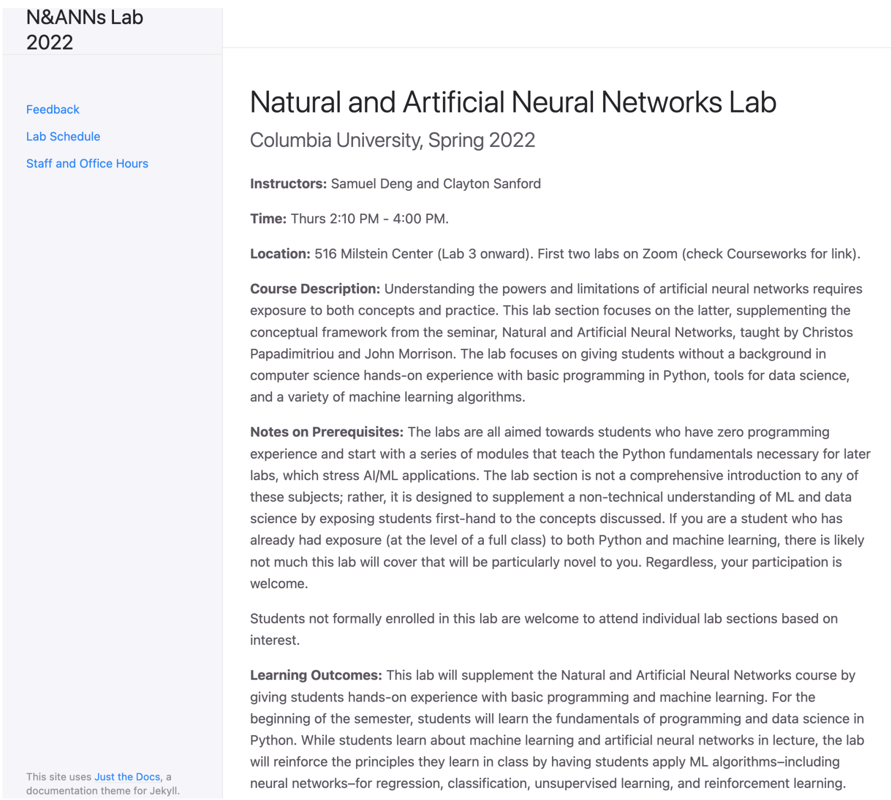
In future summer iterations, I will take this feedback to heart and adjust pacing to skip some content that's less crucial.

[Jump To: Table of Contents]

Natural and Artificial Neural Networks Syllabus

This section includes a syllabus for **Natural and Artificial Neural Networks Lab**, the companion course I co-designed and co-taught with PhD student Clayton Sanford. Some points that made this course unique were:

- This was an optional, graded companion course to a widely interdisciplinary seminar, Natural and Artificial Neural Networks, taught jointly by Christos Papadimitriou from the computer science department and John Morrison from the philosophy department.
- This course was cross-listed in many different departments, so students came from backgrounds as diverse as philosophy, computer science, law, biology, and neuroscience.
- The seminar had about 50 total students, and our lab enrolled 15 of those students.
- The purpose of the lab was to supplement the seminar with hands-on experience and build students up from potentially zero Python experience to being able to implement and play with basic neural network models with `keras` and `scikit-learn`.


N&ANNs Lab 2022

Feedback
Lab Schedule
Staff and Office Hours

Natural and Artificial Neural Networks Lab

Columbia University, Spring 2022

Instructors: Samuel Deng and Clayton Sanford

Time: Thurs 2:10 PM - 4:00 PM.

Location: 516 Milstein Center (Lab 3 onward). First two labs on Zoom (check Courseworks for link).

Course Description: Understanding the powers and limitations of artificial neural networks requires exposure to both concepts and practice. This lab section focuses on the latter, supplementing the conceptual framework from the seminar, Natural and Artificial Neural Networks, taught by Christos Papadimitriou and John Morrison. The lab focuses on giving students without a background in computer science hands-on experience with basic programming in Python, tools for data science, and a variety of machine learning algorithms.

Notes on Prerequisites: The labs are all aimed towards students who have zero programming experience and start with a series of modules that teach the Python fundamentals necessary for later labs, which stress AI/ML applications. The lab section is not a comprehensive introduction to any of these subjects; rather, it is designed to supplement a non-technical understanding of ML and data science by exposing students first-hand to the concepts discussed. If you are a student who has already had exposure (at the level of a full class) to both Python and machine learning, there is likely not much this lab will cover that will be particularly novel to you. Regardless, your participation is welcome.

Students not formally enrolled in this lab are welcome to attend individual lab sections based on interest.

Learning Outcomes: This lab will supplement the Natural and Artificial Neural Networks course by giving students hands-on experience with basic programming and machine learning. For the beginning of the semester, students will learn the fundamentals of programming and data science in Python. While students learn about machine learning and artificial neural networks in lecture, the lab will reinforce the principles they learn in class by having students apply ML algorithms–including neural networks–for regression, classification, unsupervised learning, and reinforcement learning.

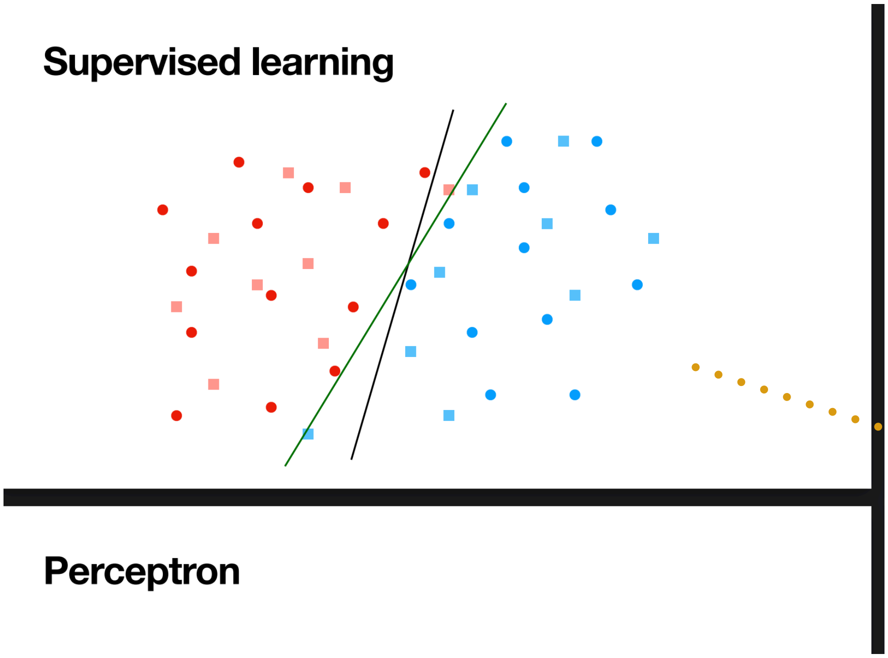
This site uses Just the Docs, a documentation theme for Jekyll.

Our course was hosted publicly online and all materials still live there now: **Natural and Artificial Neural Networks Lab**.

A main challenge of the course was to teach difficult concepts to students with a *very* wide range of backgrounds.

To design this course, Clayton and I had to constantly exercise a "beginner's mind" as PhD students in machine learning: *what parts of this might be hard to someone with little to no programming or technical background?*

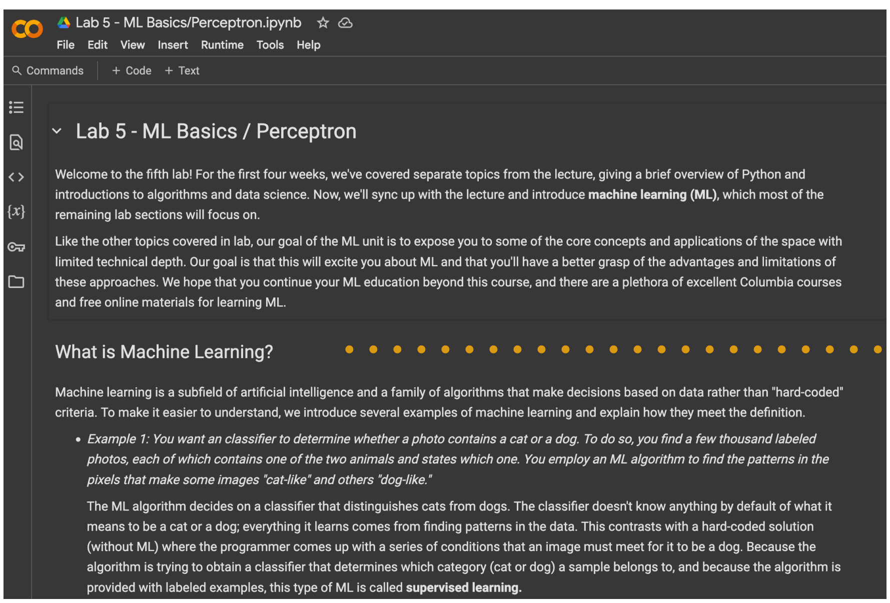
Jump To: Table of Contents



Each class session consisted of a short lecture and a hands-on interactive Python programming session.

These can all be found here.

The short lecture introduces the idea at a high level with many visuals and non-technical intuition.



The Python programming session is the focus of each session. Students step through a machine learning concept through supportive exposition and hands-on exercises. Because the course was small, Clayton and I were able to individually help students with these labs and provide one-on-one instructional feedback in a “flipped classroom” setting.

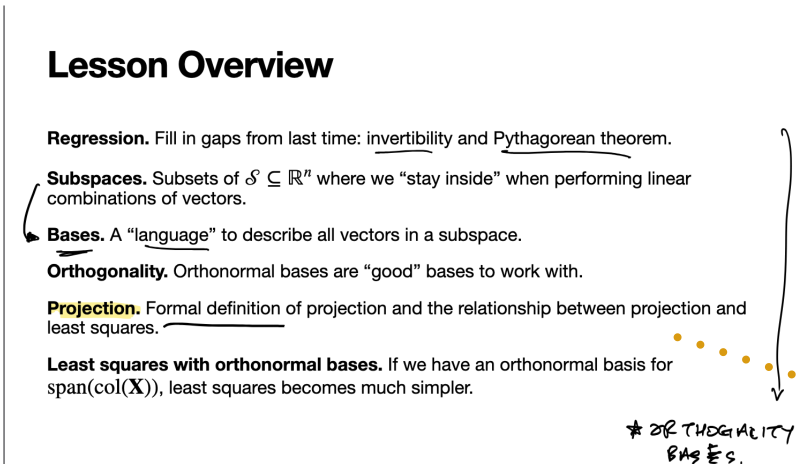
Jump To: Table of Contents

Lecture Slides: Math for ML (Subspaces, bases, orthogonality)

In this section, I go over snippets of a single lecture in Math for ML that illustrate two of my core teaching principles at the level of an individual lecture.

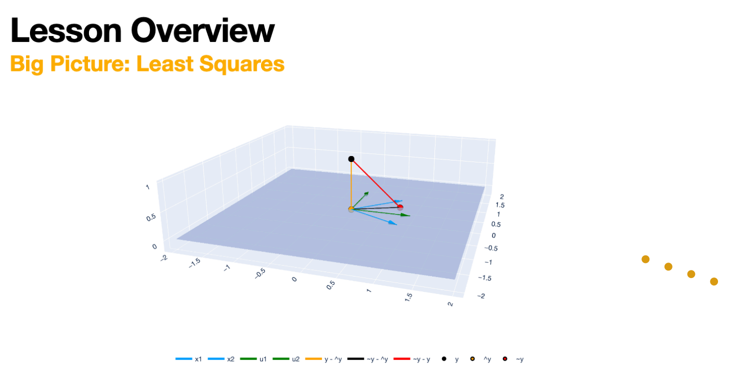
1. **A driving and cohesive narrative should propel all parts of a course.**
2. **Ideas should be presented as if the student could've discovered them themselves.**

The **lecture video can be found here**, and the **lecture slides can be found here**.

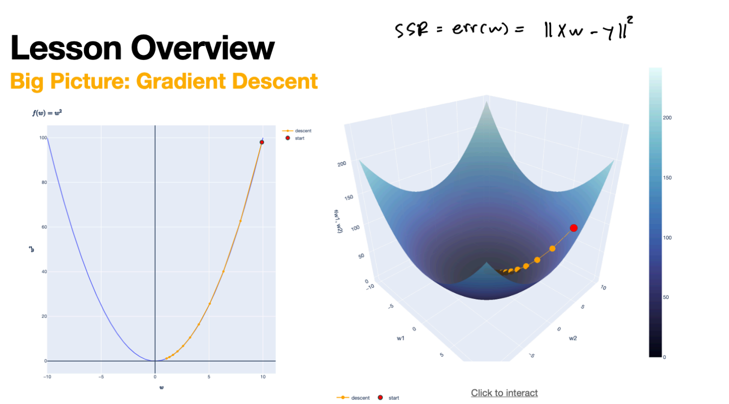


In a normal semester, this lesson takes place in the second week after students review basic linear algebra (vectors, matrices, dot products, etc.)

I begin every lesson with a lesson overview that includes all the core concepts of the lecture. These get compiled week-to-week in an evolving **course skeleton**.



Each lesson also begins with two big picture 3D visualizations that summarize the lesson in view of the main narratives of the course: *least squares* and *gradient descent.*



Students learn *exactly* all the tools (no more, no less) they need to develop the main picture, so everything is well-motivated and can be recalled in context.

I call this "teaching with Chekhov's gun."

Jump To: Table of Contents

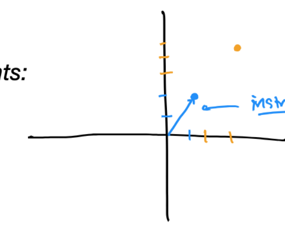
The lecture then moves to a review of the previous lecture's material with some simple sketched examples.

Least Squares
A Quick Review

Vectors
Review from linear algebra

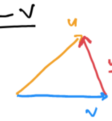
$\mathbb{R}^2$ v = (1, 2) New York
u = (3, 5) Boston.

Vectors can interchangeably thought of as *points:*



or "*arrows*":

subtract: u − v
points.



→ distance: $\|u - v\|$.

To get a direction from 2 points → $\boxed{\text{SUBTRACT}}$

Regression
Setup

Every math concept in the class is taught in service of the ML setup of *regression,* so I try to re-introduce it in these early lectures to make sure it's crystal clear.

n datapoints → d features / measurements.

Observed: Matrix of *training samples* $\mathbf{X} \in \mathbb{R}^{n \times d}$ and vector of *training labels* $\mathbf{y} \in \mathbb{R}^{\cancel{d}\, n}$.

$$\mathbf{X} = \begin{bmatrix} \uparrow & & \uparrow \\ \mathbf{x}_1 & \cdots & \mathbf{x}_d \\ \downarrow & & \downarrow \end{bmatrix} = \begin{bmatrix} \leftarrow & \mathbf{x}_1^\top & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{x}_n^\top & \rightarrow \end{bmatrix}. \Big\} \mathbb{R}^{n \times d}$$

$\boxed{X} \rightarrow W \leftarrow \boxed{Y}$, $W \rightarrow w^T x_0 \leftarrow x_0$, $w^T x_0 \rightarrow \boxed{\hat{Y}_0}$

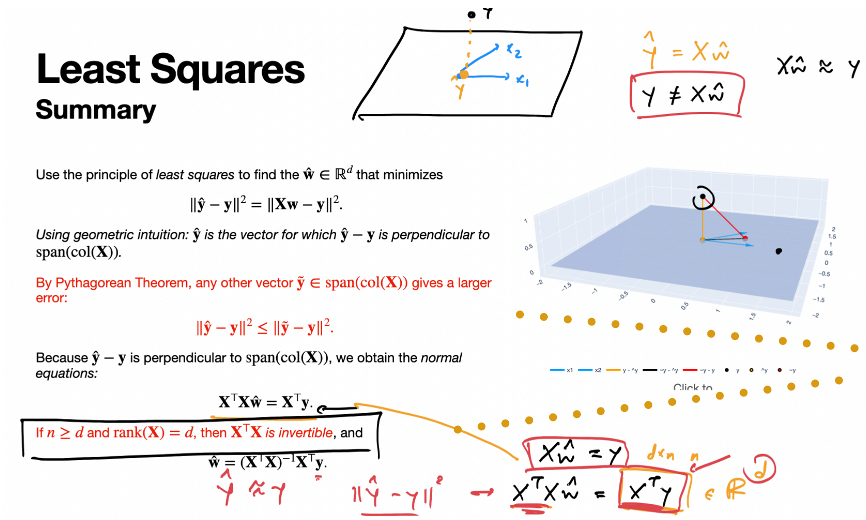
Unknown: *Weight vector* $\mathbf{w} \in \mathbb{R}^d$ with weights $w_1, \ldots, w_d$.

Goal: For each $i \in [n]$, we predict: $\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i = w_1 x_{i1} + \ldots + w_d x_{id} \in \mathbb{R}$.

Choose a weight vector that "fits the training data": $\mathbf{w} \in \mathbb{R}^d$ such that $y_i \approx \hat{y}_i$ for $i \in [n]$, or:

Test: $\boxed{x_0 \quad w^T x_0 = \hat{y}_0}$ $\boxed{\mathbf{X}\mathbf{w} = \hat{\mathbf{y}} \approx \mathbf{y}.}$ $Xw = \hat{y} \approx y$

Jump To: Table of Contents



At the end of the previous lecture, students learned the statement of the first main theorem for least squares, but two main parts were missing.

Now for the new material, and the usual cadence of how I teach. I begin by motivating why we need the math of the lesson. In this case, it's to plug up the holes needed to completely prove their first major theorem.

Least Squares

First missing item: invertibility of $\mathbf{X}^\top\mathbf{X}$

$$\boxed{\text{If } n \geq d \text{ and } \mathrm{rank}(\mathbf{X}) = d, \text{ then } \mathbf{X}^\top\mathbf{X} \textit{ is invertible.}}$$

"If there are no redundant features, then we can invert the normal equations"

All the individual math concepts should be motivated by the need to understand the bigger picture.

This structures the lectures as if students were discovering these ideas themselves.

To understand the first missing item, we need concepts of rank, invertibility, and subspace.

Whenever possible, I also always try to give a "plain English" description of mathematical facts.

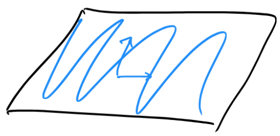
Jump To: Table of Contents

Subspaces
Idea

$\mathbb{R}^n$ = Euclidean space (n dimensions)

$S \subseteq \mathbb{R}^n$

A ***subspace*** is a set of vectors that "stays within" the set under all linear combinations of the vectors.



To get to the bottom of the first missing item, one mathematical concept we need is *subspace*.

Whenever introducing an individual mathematical concept, I start with the *idea*: a "plain English" description of its high-level intuition.

Subspaces
Definition

A ***subspace*** $\mathcal{S} \subseteq \mathbb{R}^n$ is a subset of vectors that satisfies the property: if $\mathbf{v}, \mathbf{w} \in \mathcal{S}$, then $\alpha\mathbf{v} + \beta\mathbf{w} \in \mathcal{S}$ for any $\alpha, \beta \in \mathbb{R}$.

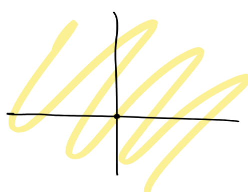
Then, I move onto the formal *definition*, making sure to emphasize how it expresses the idea.

$v - v = \vec{0}$

Subspaces
Examples

$\mathbb{R}^2 \subseteq \mathbb{R}^2$

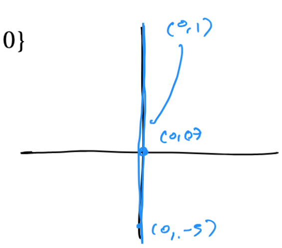
Example: $\mathcal{S}_0 := \mathbb{R}^2$



Finally, I present several simple examples to make the concept concrete.

Subspaces
Examples

Example: $\mathcal{S}_1 := \{\mathbf{v} \in \mathbb{R}^2 : v_1 = 0\}$



This isn't rocket science or anything particularly innovative — some variant of this is usually how math classes go.

However, I try to present these ideas with as little visual clutter and as many intuitive means (3D renderings, etc.) as possible.

Jump To: Table of Contents

Least Squares

First missing item: invertibility of $\mathbf{X}^\top\mathbf{X}$

$\text{rank}(X) \leq \min\{n, d\}$

Theorem (Invertibility of $\mathbf{X}^\top\mathbf{X}$). Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a matrix, with columns $\mathbf{x}_1, \ldots, \mathbf{x}_d \in \mathbb{R}^n$. If $n \geq d$ and $\text{rank}(\mathbf{X}) = d$, then $\mathbf{X}^\top\mathbf{X}$ *is invertible.*

Proof. To show that $\mathbf{X}^\top\mathbf{X}$ is invertible, show $\mathbf{X}^\top\mathbf{X}$ has d linearly independent columns.

$$\mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{0} \implies \mathbf{w} = \mathbf{0}.$$

Suppose $\mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{0}$. Let $\mathbf{w} \in \mathbb{R}^d$ be any vector. Take a dot product of both sides with $\mathbf{w}$:

$$\|\mathbf{X}\mathbf{w}\|^2 \implies \mathbf{X}\mathbf{w} = \mathbf{0}.$$

$Xw = \vec{0} \Rightarrow \vec{w} = \vec{0}$

But $\text{rank}(\mathbf{X}) = d$, so $\mathbf{X}$ has d linearly independent columns. Therefore, $\mathbf{w} = \mathbf{0}$.

$$Xw = \begin{bmatrix} | & & | \\ x_1 & \cdots & x_d \\ | & & | \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix} = \vec{0}.$$

After learning the appropriate mathematical concepts in sequence, we bring it back to prove the "first missing item." It should be clear how everything fits into the broader puzzle.

Least Squares

Summary

Use the principle of *least squares* to find the $\hat{\mathbf{w}} \in \mathbb{R}^d$ that minimizes

$$\|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2.$$

Using geometric intuition: $\hat{\mathbf{y}}$ *is the vector for which* $\hat{\mathbf{y}} - \mathbf{y}$ *is perpendicular to* $\text{span}(\text{col}(\mathbf{X}))$.

By Pythagorean Theorem, any other vector $\tilde{\mathbf{y}} \in \text{span}(\text{col}(\mathbf{X}))$ gives a larger error:

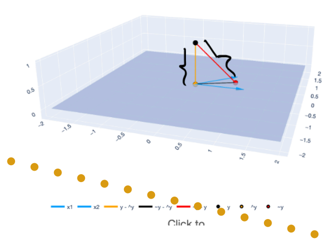
$$\|\hat{\mathbf{y}} - \mathbf{y}\|^2 \leq \|\tilde{\mathbf{y}} - \mathbf{y}\|^2.$$

Because $\hat{\mathbf{y}} - \mathbf{y}$ is perpendicular, we obtain the *normal equations:*

$$\boxed{\mathbf{X}^\top\mathbf{X}\hat{\mathbf{w}} = \mathbf{X}^\top\mathbf{y}.}$$

If $n \geq d$ and $\text{rank}(\mathbf{X}) = d$, then $\mathbf{X}^\top\mathbf{X}$ *is invertible*, and

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



I very frequently call back and show the "big picture" 3D renderings to orient students. The renderings are clickable and interactive on the slide PDF itself, so students can interact with it.

See how I do this in [the lecture video] or [try it yourself]!

I make sure to loop back around to what statements are still pending, indicating in green what we have proven, and in red the component that remains.

Least Squares

Second missing item: Pythagorean Theorem

By Pythagorean Theorem, any other vector $\tilde{\mathbf{y}} \in \text{span}(\text{col}(\mathbf{X}))$ gives a larger error:

$$\boxed{\|\hat{\mathbf{y}} - \mathbf{y}\|^2 \leq \|\tilde{\mathbf{y}} - \mathbf{y}\|^2.}$$

"The vector closest to $\mathbf{y}$ in the subspace is perpendicular."

I then repeat the process with the other missing piece.

Jump To: Table of Contents

Least Squares
Summary

Use the principle of *least squares* to find the $\hat{\mathbf{w}} \in \mathbb{R}^d$ that minimizes

$$\|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2.$$

Using geometric intuition: $\hat{\mathbf{y}}$ *is the vector for which* $\hat{\mathbf{y}} - \mathbf{y}$ *is perpendicular to* $\mathrm{span}(\mathrm{col}(\mathbf{X}))$.

By Pythagorean Theorem, any other vector $\tilde{\mathbf{y}} \in \mathrm{span}(\mathrm{col}(\mathbf{X}))$ gives a larger error:

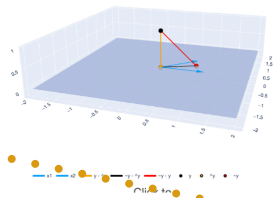
$$\|\hat{\mathbf{y}} - \mathbf{y}\|^2 \leq \|\tilde{\mathbf{y}} - \mathbf{y}\|^2.$$

Because $\hat{\mathbf{y}} - \mathbf{y}$ is perpendicular, we obtain the *normal equations:*

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

If $n \geq d$ and $\mathrm{rank}(\mathbf{X}) = d$, then $\mathbf{X}^\top\mathbf{X}$ *is invertible*, and

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



We've proven both pieces of our puzzle! Students now hopefully feel the satisfaction that all of the abstract math they learned was in service of a broader story.

Hopefully they also feel that they could've discovered this themselves by emulating how I broke this nontrivial statement into two modular chunks.

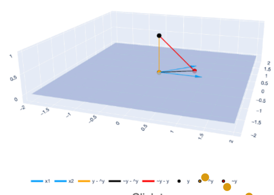
Least Squares
Summary

Goal: Find the $\hat{\mathbf{w}} \in \mathbb{R}^d$ that minimizes

$$\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2.$$

<u>**Theorem (OLS).**</u> If $n \geq d$ and $\mathrm{rank}(\mathbf{X}) = d$, then:

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



Most theorem statements then add something to the "big picture" 3D renderings. This example didn't but, later in the same lecture, it gets slightly updated when students learn an orthonormal basis.

Lesson Overview

Regression. Fill in gaps from last time: invertibility and Pythagorean theorem.

Subspaces. Subsets of $\mathcal{S} \subseteq \mathbb{R}^n$ where we "stay inside" when performing linear combinations of vectors.

Bases. A "language" to describe all vectors in a subspace.

Orthogonality. Orthonormal bases are "good" bases to work with.

Projection. Formal definition of projection and the relationship between projection and least squares.

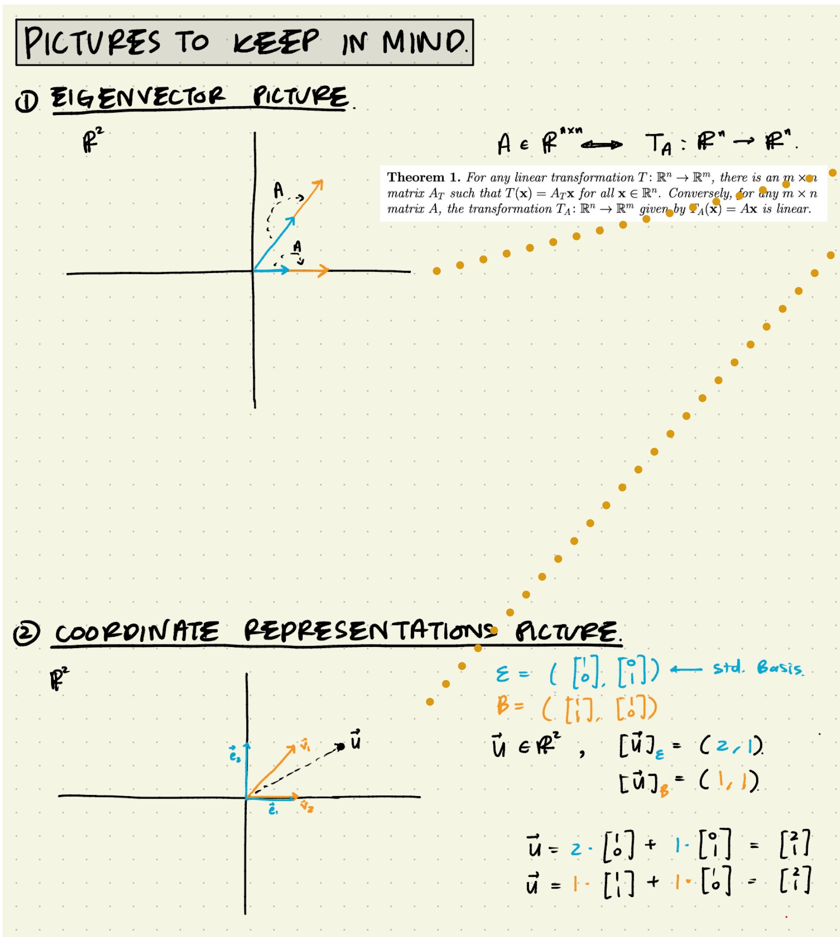
Least squares with orthonormal bases. If we have an orthonormal basis for $\mathrm{span}(\mathrm{col}(\mathbf{X}))$, least squares becomes much simpler.

The lesson always ends with a recap of the main ideas again.

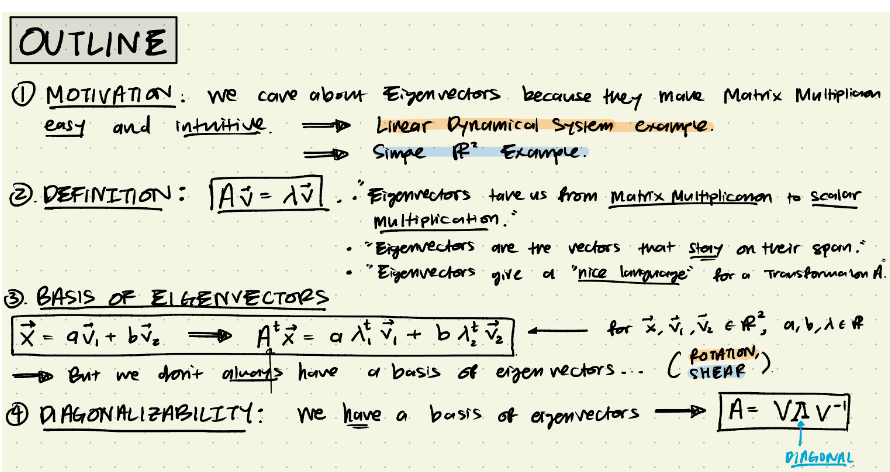
Jump To: Table of Contents

Lecture Notes: Computational Linear Algebra

Another example of an individual lecture is my **guest lecture on eigenvalues and eigenvectors** for Computational Linear Algebra. This was a more traditional mathematics lecture that I gave on my iPad.



The lesson begins with two key pictures that I continually reference. I didn't have the 3D rendering chops in Fall 2022, but these 2D doodles served the same purpose.

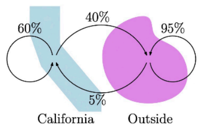


I motivate why they need to learn arguably one of the most abstract parts of linear algebra: eigenvectors and diagonalization.

Jump To: Table of Contents

EXAMPLE: PS1, PROBLEM 1.7

- of those who start a year in California, 60 percent stay in and 40 percent move out by the end of the year; and
- of those who start a year outside California, 95 percent stay out and 5 percent move in by the end of the year.



If we know the situation at the beginning of a year, say 300 million outside and 40 million in, then it is easy to find the numbers x and y that are outside and inside California by the end of the year:

$$\text{number outside} = x = 0.95 \times 300{,}000{,}000 + 0.4 \times 40{,}000{,}000$$
$$\text{number inside} = y = 0.05 \times 300{,}000{,}000 + 0.6 \times 40{,}000{,}000.$$

$\vec{x}_1 = A\vec{x}_0$

$\vec{x}_2 = A\vec{x}_1 = A(A\vec{x}_0) = A^2\vec{x}_0$

$\vec{x}_3 = A\vec{x}_2 = A(A^2\vec{x}_0) = A^3\vec{x}_0$

...

$\vec{x}_t =$ Population at time t.

$\vec{x}_0 =$ initial population.

Let $\vec{x}_t = \begin{bmatrix} x_{IN} \\ x_{OUT} \end{bmatrix}$ be the populations at time t. Then, $\vec{x}_0$ are initial populations.

(E.g. $\vec{x}_0 = \begin{bmatrix} 40 \times 10^6 \\ 300 \times 10^6 \end{bmatrix}$).

TRANSITION MATRIX.

$$A = \begin{bmatrix} \text{IN CA} \to \text{IN CA} & \text{OUT CA} \to \text{IN CA} \\ \text{IN CA} \to \text{OUT CA} & \text{OUT CA} \to \text{OUT CA} \end{bmatrix}$$

$$= \boxed{\begin{bmatrix} 0.6 & 0.05 \\ 0.4 & 0.95 \end{bmatrix}}.$$

HARD TO DO: $\boxed{\vec{x}_t = \begin{bmatrix} 0.6 & 0.05 \\ 0.4 & 0.95 \end{bmatrix}^t \vec{x}_0}$

ASSUME I TOLD YOU...

① For $\vec{u} = (1, 8)$, $A\vec{u} = \begin{bmatrix} 0.6 & 0.05 \\ 0.4 & 0.95 \end{bmatrix}\begin{bmatrix} 1 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 \\ 8 \end{bmatrix}$.

$\longrightarrow$ $A\vec{u} = \vec{u}$ (keeps $\vec{u}$ the same).

A previous problem set "seeded" this idea early on.

I recap this problem that students already solved and how it relates to eigenvectors, giving a concrete example about population change from New York to California.

DEFINITIONS: EIGENVECTORS / EIGENVALUES.

For $n \times n$ Matrix A: $\boxed{A\vec{v} = \lambda\vec{v}}$

EIGENVECTOR EIGENVALUE.

EXAMPLES ABOVE:

$\underbrace{\begin{bmatrix} -1/2 & 5/2 \\ 0 & 2 \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} 1 \\ 1 \end{bmatrix}}_{\vec{v}} = \underbrace{2}_{\lambda}\underbrace{\begin{bmatrix} 1 \\ 1 \end{bmatrix}}_{\vec{v}}$ and $\underbrace{\begin{bmatrix} -1/2 & 5/2 \\ 0 & 2 \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\vec{v}} = \underbrace{-1/2}_{\lambda}\underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\vec{v}}$.

DEF (EIGENVECTOR/ EIGENVALUE). A nonzero vector $\vec{v} \in \mathbb{R}^n$ is an eigenvector of $n \times n$ Matrix A if there exists a scalar $\lambda \in \mathbb{R}$ s.t. $\boxed{A\vec{v} = \lambda\vec{v}}$. The scalar λ is the eigenvalue corresponding to $\vec{v}$.

INTUITION: Eigenvectors are the vectors that stay on their span (either "shrunk" or "stretched" by the transformation corresponding to A.

- $\vec{v}$ is an eigenvector, w/ eigenvalue λ $\Longrightarrow$ All nonzero $\vec{v}' \in \text{span}(\{\vec{v}\})$ are also eigenvectors w/ eigenvalue λ.
- MATRIX MULTIPLICATION (HARD!) $\Rightarrow$ SCALAR MULTIPLICATION (EASY!)

Finally, after providing the requisite motivation and intuition, I give the definition.

I try to emphasize that this definition seems inevitable if we want to think of the "fixed point" of transformations, or the "vectors that stay on their span."

I stress how this definition is natural after the motivation and intuition, leading students to think that they could've formalized these intuitive ideas themselves.

[Jump To: Table of Contents]

FOR OUR ONGOING EXAMPLE:

$A = \begin{bmatrix} -1/2 & 5/2 \\ 0 & 2 \end{bmatrix}$ Eigenvalues: $\lambda_1 = 2$, $\lambda_2 = -1/2$

Eigenvectors: $\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\vec{v}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ ← Linearly independent!

$\mathcal{E}$ = standard basis = $(\vec{e}_1, \vec{e}_2) = ((1,0),(0,1))$

$\mathcal{B}$ = eigenvector basis = $(\vec{v}_1, \vec{v}_2) = ((1,1),(1,0))$

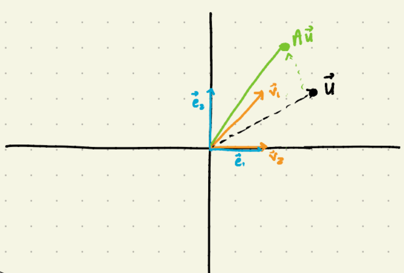
$$\underbrace{\begin{bmatrix} -1/2 & 5/2 \\ 0 & 2 \end{bmatrix}}_{[T]_{\mathcal{E}\to\mathcal{E}}}^{A} = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}_{[id]_{\mathcal{B}\to\mathcal{E}}}^{V} \underbrace{\begin{bmatrix} 2 & 0 \\ 0 & -1/2 \end{bmatrix}}_{[T]_{\mathcal{B}\to\mathcal{B}}}^{\Lambda} \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{-1}}_{[id]_{\mathcal{E}\to\mathcal{B}}}^{V^{-1}} \Longleftrightarrow \underbrace{\begin{bmatrix} 2 & 0 \\ 0 & -1/2 \end{bmatrix}}_{[T]_{\mathcal{B}\to\mathcal{B}}}^{\Lambda} = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{-1}}_{[id]_{\mathcal{E}\to\mathcal{B}}}^{V^{-1}} \underbrace{\begin{bmatrix} -1/2 & 5/2 \\ 0 & 2 \end{bmatrix}}_{[T]_{\mathcal{E}\to\mathcal{E}}}^{A} \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}_{[id]_{\mathcal{B}\to\mathcal{E}}}^{V}$$

☆ APPLY Λ to any vector $[\vec{u}]_{\mathcal{B}}$ (represented in the "language" of $\mathcal{B}$).

For example: $[\vec{u}]_{\mathcal{E}} = (2,1) \Longleftrightarrow [\vec{u}]_{\mathcal{B}} = (1,1)$.

$$\begin{bmatrix} -1/2 & 5/2 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3/2 \\ 2 \end{bmatrix} \leftarrow \mathcal{E} \qquad \begin{bmatrix} 2 & 0 \\ 0 & -1/2 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ -1/2 \end{bmatrix} \leftarrow \mathcal{B}$$

☆ SAME TRANSFORMATION, DIFFERENT "INTERPRETATION."



$\mathcal{E} = \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right)$

$\mathcal{B} = \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right)$

$$\begin{bmatrix} 2 & 0 \\ 0 & -1/2 \end{bmatrix}^t = \begin{bmatrix} 2^t & 0 \\ 0 & (-1/2)^t \end{bmatrix}$$

With a topic as abstract as eigenvalues and eigenvectors, it helps tremendously to step through a representative simple example.

In this case, I drew on my third teaching principle: **(3) An instructor should never forget how they first struggled when learning the same ideas.** For me, a simple 2D example helps tremendously when learning a new theorem or definition.

Walking through this example should be an active experience: I frequently stop and ask students if they know the "next step" in the computations.

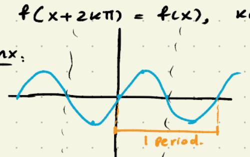
ABSTRACT V.S. EXAMPLE: CONVOLUTION.

$W = C_{\text{periodic}}([-\pi,\pi], \mathbb{R})$.

(vector space of continuous, 2π-periodic, real-valued functions on $[-\pi,\pi]$.)

VECTORS: continuous, periodic $f: [-\pi,\pi] \to \mathbb{R}$. $f(x+2k\pi) = f(x)$, $k \in \mathbb{Z}$.

Ex $f(x) = \sin x$.



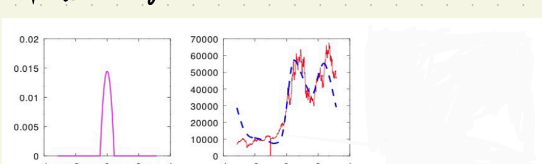
INNER PROD.: $\langle f, g \rangle_W = \int_{-\pi}^{\pi} f(t)g(t)\,dt$

DEF (CONVOLUTION OPERATOR): The convolution operator is the linear transformation $T_h: W \to W$ for a fixed kernel function $h \in W$, defined:

$$\boxed{T_h(f) = g, \quad \text{where} \quad g(t) = \int_{-\pi}^{\pi} f(t-x)\,h(x)\,dx}$$

$f(-x)\,h(x)\,dx$

- What does convolution do?. Transform "rough" function into a "smoother" function by convolving with a "nice" kernel:



3BLUE1BROWN

$$\boxed{T_h(f) = \lambda f}$$

scalar

same function.

I close the lecture with a very nontrivial application of eigenvalues and eigenvectors to convolutions, which motivates and connects it to concepts earlier in the class (in particular, abstract vector spaces).

[Jump To: Table of Contents]

Problem Set: Math for ML

In this section, I present a representative problem from Problem Set 1 of Math for ML. They all abide by my second teaching principle: **(2) Ideas should be presented as if the student could've discovered them themselves.**

All the problems sets in the course aren't just sequences of exercises: they model the process of mathematical discovery by giving a nontrivial result or theorem as a problem, but then guiding students through the process of: (i) testing it on simple examples (ii) proving key lemmas (iii) piecing the lemmas together to complete the theorem.

My other problem sets can be found at this link (under PS #).

Problem 2

Linear transformations and matrices (26 points total). The property that underlies all of linear algebra is *linearity*. In this problem, we will attempt to understand the relationship between matrices and linear transformations.

Many common functions in the real world are linear. Cooking is one of them. Consider the following example. Suppose that we have $d = 7$ ingredients to make some classic NYC fare: bacon, egg, cheddar cheese, cream cheese, bagel, Kaiser roll, and lox. Consider four recipes we can make with these ingredients, represented by the vectors $\mathbf{r}, \mathbf{c}, \mathbf{b}$ and $\mathbf{l}$. The d ingredients are ordered as above; for example, to make a bacon, egg and cheese on a roll (vector $\mathbf{r}$), we need one unit each of bacon, egg, cheddar cheese, and Kaiser roll, with zero units of the other ingredients.

$$\text{bacon, egg, and cheese on roll:} \quad \mathbf{r} = \begin{bmatrix}1\\1\\1\\0\\0\\1\\0\end{bmatrix} \qquad \text{cream cheese on bagel:} \quad \mathbf{c} = \begin{bmatrix}0\\0\\0\\2\\1\\0\\0\end{bmatrix}$$

$$\text{bacon, egg, and cheese on bagel:} \quad \mathbf{b} = \begin{bmatrix}1\\1\\1\\0\\1\\0\\0\end{bmatrix} \qquad \text{lox sandwich on bagel:} \quad \mathbf{l} = \begin{bmatrix}0\\0\\0\\1\\1\\0\\1\end{bmatrix}.$$

Suppose the vector $\mathbf{u} = (4, 4, 4, 5, 6, 2, 3)$ describes how much of each ingredient we have in supply today (four units of bacon, four units of egg, etc.).

> **Problem 2(a) [2 points].** We would like to use as many of our ingredients as possible to make as many of the above recipes as possible. How many of each recipe can we make with zero surplus (or shortfall) of each ingredient? Set up a system of linear equations for this question in matrix-vector form.

> **Problem 2(b) [2 points].** Does the system of equations in Problem 2(a) have a solution? If so, write down a solution. If not, explain why. Feel free to use `numpy` or any other numerical computing software to help you solve the system.

As we can see from the above example, matrix-vector multiplication has the nice property that, if you add the inputs, you add the outputs (if we wanted twice as many of each recipe,

Most problems begin with a motivating example that the student can easily step through mechanically. The example should capture the essence of the idea for this particular problem — in this case, *linearity.*

Problem 2(a) and Problem 2(b) are easy points but make sure that the student has taken time to play with the example. In proving a new theorem for my research, I find this is usually the first step.

The problem is interlaced with exposition and a loose "narrative" that drives the discovery. The course doesn't have an official textbook, so this is a good way to have students actively read supporting material.

Jump To: Table of Contents

Let $T : \mathbb{R}^d \to \mathbb{R}^n$ be a function (also referred to as a "mapping" or "transformation"). Functions can be arbitrarily complicated; a function need only map inputs in $\mathbb{R}^d$ to outputs in $\mathbb{R}^n$. *Linear transformations* (a.k.a. "linear functions" or "linear maps") are restricted to obey two rules that force them to behave nicely:

$$T(\mathbf{x}+\mathbf{y}) = T(\mathbf{x}) + T(\mathbf{y}) \quad \text{and} \quad T(\alpha\mathbf{x}) = \alpha T(\mathbf{x})$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ and scalars $\alpha \in \mathbb{R}$.

Problem 2(c) [**8 points**] Determine whether the following transformations are linear. If a function is linear, give a proof by showing the function satisfies the properties of linearity. If not, state which property of linearity fails and give a specific pair of vectors $\mathbf{x}, \mathbf{y}$ or a scalar α and vector $\mathbf{x}$ for which it fails.

- $T : \mathbb{R} \to \mathbb{R}$ defined $T(x) := 2x - 1$.
- $T : \mathbb{R}^2 \to \mathbb{R}^2$ defined as $T(x_1, x_2) := (x_2, x_1 + x_2)$.
- $T : \mathbb{R}^d \to \mathbb{R}$ defined $T(x) := \frac{1}{d}(x_1 + \cdots + x_d)$.
- $T : \mathbb{R}^d \to \mathbb{R}$ defined $T(x_1, \ldots, x_d) := x_d - x_1$.

Taken as functions, inner products and matrix-vector products are also linear. For a given vector $\mathbf{a} \in \mathbb{R}^d$, let the function $T_{\mathbf{a}} : \mathbb{R}^d \to \mathbb{R}$ be defined as:

$$T_{\mathbf{a}}(\mathbf{x}) := \mathbf{a}^\top \mathbf{x}. \tag{3}$$

For a given matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, let the function $T_{\mathbf{A}} : \mathbb{R}^d \to \mathbb{R}^n$ be defined as:

$$T_{\mathbf{A}}(\mathbf{x}) := \mathbf{A}\mathbf{x}. \tag{4}$$

Problem 2(d) [**4 points**] Prove that the function defined by inner products in Equation (3) and the function defined by matrix-vector products in Equation (4) are linear transformations. For Equation (4), you may use any of the equivalent characterizations of matrix-vector multiplication shown in class.

In this way, any matrix defines a linear transformation. This is important — perhaps in your introductory linear algebra class, matrices were introduced as just a way to organize a system of linear equations, like $\mathbf{A}\mathbf{x} = \mathbf{b}$. Equation (4) tells us that we can actually think of a matrix as an object that *does something* to vectors. Given a matrix, matrix-vector multiplication is a linear transformation. Surprisingly, the reverse is true as well: *any* linear transformation has an associated matrix!

The problems gradually ramp up in difficulty. Problem 2(c) is easy but no longer purely mechanical, and it asks for short proofs.

A learning goal of this course is to develop students' mathematical maturity, broadly speaking. The problem sets attempt to do this by modeling problem-solving skills such as experimenting with simple examples and proving helper lemmas.

Text like "*this is important*" and "*surprisingly*" liven the problem's exposition and point out the gut feelings a student should be feeling.

Jump To: Table of Contents

Consider the following example. Let $\mathbf{e}_1 = (1,0,0), \mathbf{e}_2 = (0,1,0)$, and $\mathbf{e}_3 = (0,0,1)$ denote the standard basis vectors in $\mathbb{R}^3$. Let $T : \mathbb{R}^3 \to \mathbb{R}^2$ be the linear transformation defined as:

$$T(x_1, x_2, x_3) := (2x_1, x_2 + x_3).$$

Problem 2(e) [1 point] Where does T map the basis vectors to? That is, compute $T(\mathbf{e}_1)$, $T(\mathbf{e}_2)$, and $T(\mathbf{e}_3)$.

We move onto proving the surprising fact that any linear transformation has an associated matrix. Again, start with simple examples to get a feel.

Now, consider the input vector $\mathbf{x} = (3, 2, -1)$. Because $\mathbf{x} \in \mathbb{R}^3$ and $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ are a basis for $\mathbb{R}^3$, we can write $\mathbf{x}$ as a linear combination of $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$. Using this example, we'll try to "guess" the matrix that corresponds to T.

Problem 2(f) [1 point] Write the matrix $\mathbf{A} \in \mathbb{R}^{2\times 3}$ such that:

$$T(\mathbf{x}) = \mathbf{A}\mathbf{x}.$$

for $\mathbf{x} = (3, 2, -1)$.

Hint: Write $\mathbf{x}$ as a linear combination of $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$, i.e.,

$$\mathbf{x} = \alpha_1\mathbf{e}_1 + \alpha_2\mathbf{e}_2 + \alpha_3\mathbf{e}_3, \tag{5}$$

where $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}$ are scalars. Apply $T(\cdot)$ to both sides of Equation (5), and use linearity to get the right-hand side to be a sum of three terms.

Hints point students toward what intuitive "next steps" might be in a proof or derivation.

Problem 2(f) shows us that $T(\mathbf{x})$ is just a linear combination of $T(\mathbf{e}_1)$, $T(\mathbf{e}_2)$, and $T(\mathbf{e}_3)$. It turns out that, in general, if we are given a linear transformation and want to find its corresponding matrix $\mathbf{A}$, we only need to see what that linear transformation does to the standard basis vectors.

Problem 2(g) [4 points] Prove that any linear transformation $T : \mathbb{R}^d \to \mathbb{R}^n$ is given by matrix-vector multiplication by a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$:

$$T(\mathbf{x}) = \mathbf{A}\mathbf{x},$$

where the ith column of $\mathbf{A}$ is $T(\mathbf{e}_i)$.

Another key skill in discovering and proving results in math is going from the specific to the general. By guiding students to do this, students first "get a feel" for the proof and then can "take off the training wheels" to prove the abstract, general result.

Together, Equation (4) and Problem 2(g) give us a central theorem of linear algebra: the equivalence of matrices and linear transformations:

(a) Any matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ defines a linear transformation $T : \mathbb{R}^d \to \mathbb{R}^n$ through matrix-vector multiplication:

$$T(\mathbf{x}) = \mathbf{A}\mathbf{x}.$$

It turns out that Problem 2(g), which the students have now done themselves, was important to a "central theorem of linear algebra" all along!

Jump To: Table of Contents

(b) Any linear transformation $T : \mathbb{R}^d \to \mathbb{R}^n$ is given by matrix-vector multiplication by a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$:

$$T(\mathbf{x}) = \mathbf{A}\mathbf{x},$$

where the ith column of $\mathbf{A}$ is $T(\mathbf{e}_i)$.

The claim in (b) is particularly interesting — it tells us that, *any* linear transformation can be pinned down (by a concrete box of $n \times d$ numbers) just by seeing how that transformation acts on the standard basis vectors. Just by imposing the property of linearity on functions, we can treat them as matrices which we can easily write down! This perspective on matrices as linear transformations (and vice versa) is very helpful in understanding many of the definitions and theorems of linear algebra.

One such operation that we've already studied is the *projection* operation. Informally, we compute a projection of a point onto a subspace by seeing where a perpendicular line from the point intersects the subspace. Formally, for the subspace $S \subseteq \mathbb{R}^d$, the projection $\Pi_S(\mathbf{x})$ of $\mathbf{x} \in \mathbb{R}^d$ onto S satisfies:

$$(\mathbf{x} - \Pi_S(\mathbf{x}))^\top \mathbf{u}, \quad \text{for all } \mathbf{u} \in S.$$

The theorem we proved above shows us that we can determine the exact projection matrix if we know what a transformation does to the standard basis vectors.

Problem 2(h) **[2 points]** Consider the linear transformation in $T : \mathbb{R}^2 \to \mathbb{R}^2$ that takes any point $\mathbf{x} \in \mathbb{R}^2$ and outputs its projection onto the x-axis, i.e. the subspace spanned by the vector $\mathbf{u} = (1, 0)$. Find the matrix $\mathbf{A} \in \mathbb{R}^{2\times 2}$ that corresponds to this transformation. Find the explicit rule $T(x_1, x_2)$ that corresponds to this transformation.

Hint: What does this transformation do to $\mathbf{e}_1$? What does it do to $\mathbf{e}_2$? It may help to draw a picture.

Problem 2(i) **[2 points]** Consider the linear transformation in $T : \mathbb{R}^2 \to \mathbb{R}^2$ that takes any point $\mathbf{x} \in \mathbb{R}^2$ and outputs its projection onto the $y = x$ line, i.e. the subspace spanned by the vector $\mathbf{u} = (1, 1)$. Find the matrix $\mathbf{A} \in \mathbb{R}^{2\times 2}$ that corresponds to this transformation. Find the projection of the vector $\mathbf{x} = (3, -1)$ onto this subspace.

Other properties of matrices also become more intuitive when we conceive of matrices in $\mathbb{R}^{n\times d}$ as linear transformations from $\mathbb{R}^d$ to $\mathbb{R}^n$. For example, one of the concepts we've learned is *rank*, the number of linearly independent columns of a matrix. From (b), the columns of a matrix are exactly where the standard basis vectors "land" after the associated transformation. Therefore, a matrix that is not full-rank transforms the standard basis such that some of them are linearly dependent after the transformation.

Commit the theorem you proved above to memory — it's at the very heart of linear algebra!

Some more expository text emphasizes that the student has shown something nontrivial.

The problem closes by connecting the statement the student has just proven back to lecture — in this case, Problems 2(h) and 2(i) walk the student through the theorem's relation to projection from Lecture 1.2.

This problem also connects back to the idea of *rank* from lecture. Most of the problems in the problem sets do something similar, re-contextualizing ideas students have learned in the slides.

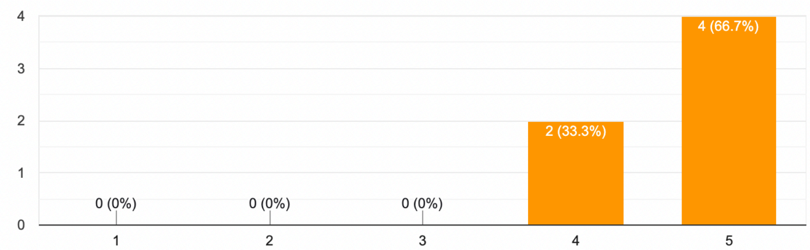
Finally, the hope is that, by proving the theorem bit-by-bit, the student comes away feeling like they *own* the statement. Recalling something you truly understand and own is much easier.

Jump To: Table of Contents

How did you like the exposition-problem-exposition format of PS1?

6 responses

Copy chart



This particular problem set was well-received by students in a mid-course survey.

In my opinion, the strength(s) of this class compared to other classes were:

3 responses

A strong emphasis on visualization, examples, and case-study type questions.

It was a clearly presented, important information. I enjoyed learning.

The Problem Sets. I think the problem sets were great. Explained the topic being used. Helped us derive the main concept from the basics.

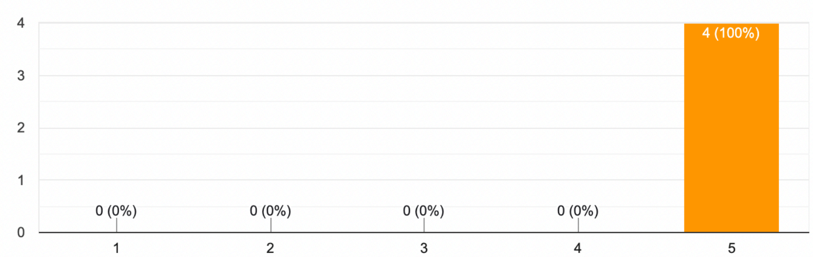
Overall, students seemed to find the problem sets a highlight of the course.

Class and lectures overall

The exposition-question-exposition format of the problem sets helped me understand and reinforce concepts better than a traditional problem set.

4 responses

Copy chart



Jump To: Table of Contents

Python Lab: Natural and Artificial Neural Networks

As an **example of a programming assignment** I co-designed with my co-instructor Clayton Sanford, this section steps through a mid-semester programming lab that introduces students to the Perceptron algorithm. These labs were done in-class, over two hours of "flipped classroom" instruction. One thing to note is that students came from a very wide range of backgrounds and programming experience.

All the Python labs for Natural and Artificial Neural Networks can be found here.

Lab 5 - ML Basics / Perceptron

Welcome to the fifth lab! For the first four weeks, we've covered separate topics from the lecture, giving a brief overview of Python and introductions to algorithms and data science. Now, we'll sync up with the lecture and introduce **machine learning (ML)**, which most of the remaining lab sections will focus on.

Like the other topics covered in lab, our goal of the ML unit is to expose you to some of the core concepts and applications of the space with limited technical depth. Our goal is that this will excite you about ML and that you'll have a better grasp of the advantages and limitations of these approaches. We hope that you continue your ML education beyond this course, and there are a plethora of excellent Columbia courses and free online materials for learning ML.

What is Machine Learning?

Machine learning is a subfield of artificial intelligence and a family of algorithms that make decisions based on data rather than "hard-coded" criteria. To make it easier to understand, we introduce several examples of machine learning and explain how they meet the definition.

- *Example 1: You want an classifier to determine whether a photo contains a cat or a dog. To do so, you find a few thousand labeled photos, each of which contains one of the two animals and states which one. You employ an ML algorithm to find the patterns in the pixels that make some images "cat-like" and others "dog-like."*

  The ML algorithm decides on a classifier that distinguishes cats from dogs. The classifier doesn't know anything by default of what it means to be a cat or a dog; everything it learns comes from finding patterns in the data. This contrasts with a hard-coded solution (without ML) where the programmer comes up with a series of conditions that an image must meet for it to be a dog. Because the algorithm is trying to obtain a classifier that determines which category (cat or dog) a sample belongs to, and because the algorithm is provided with labeled examples, this type of ML is called **supervised learning.**

- *Example 2: You own a restaurant and receive thousands of reviews online. You do not have time read them individually, but you want to know roughly what they cover. You apply an ML algorithm that groups together similar reviews based on their word choices, their tones, and their overall topics. For instance, one category may include a group of reviews that complain about the desserts. Another compliments the decor, and still another whines about the noise from the nearby train.*

The labs begin with motivation and exposition for the upcoming concepts, with a view that students in this course come from a very wide range of backgrounds in programming, math prerequisites, and academic discipline.

Linear Algebra and Vectors

Understanding machine learning in depth requires a background in probability, algorithms, and linear algebra. This section gives an overview of the basics of linear algebra needed to understand the ML algorithms that you'll try out today.

What is linear algebra? Linear algebra is a field of math which generalizes what you learned in high school algebra to other concepts besides one-dimensional numbers. For instance, elementary knowledge about linear operations allows a high school student to conclude that the equality $4x + 2 = 6$ is satisfied when $x = 1$. Linear algebra asks similar questions for the case where x is a **vector**, or an ordered tuple of numbers. The more precise term for "one-dimensional number" in linear algebra is **scalar.**

Vectors are useful in ML because we're rarely interested in performing inference on a scalar input. For instance, if we want distinguish between cats and dogs (as discussed in Example 1), the input to the algorithm is an image. The image can be precisely represented as a grid of pixels. If each images is composed of (say) 256 pixels by 256 pixels, then a total of $256 \cdot 256 = 65536$ pixels exactly represent the image. What is a pixel? A pixel is represented by three numbers, each corresponding to the amount of red, green, and blue light in the pixel. Therefore, the image can be represented exactly as an ordered collection of $65536 \cdot 3 = 196608$ numbers, which can be thought of as a 196608-dimensional vector. In order to reason about how the model processes this input, we need a mathematical language for thinking about these objects and what can be done with them. That is linear algebra.

As an example, we say that $\mathbf{x} = [1, 2, 3]$ is a three-dimensional vector, which we express as $\mathbf{x} \in \mathbb{R}^3$. ($\mathbb{R}$ represents the collection of all real numbers; $\mathbb{R}^3$ is any triple of real numbers, or a 3d coordinate.) More generally, we can let $\mathbf{x} = [x_1, x_2, \dots, x_d] \in \mathbb{R}^d$ be a d-dimensional vector with first component x_1, second component x_2, and ith component x_i.

We define three operations on vectors, which can also be done for one-dimensional numbers: *addition, scalar multiplication*, and *dot product.*

- **Addition:** If x and y are scalars, we can express their sum as $x + y$, which simply adds together the two scalars. We can do the same for vectors. If $\mathbf{x} = [x_1, x_2, \dots, x_d] \in \mathbb{R}^d$ and $\mathbf{y} = [y_1, y_2, \dots, y_d] \in \mathbb{R}^d$ are d-dimensional vectors, then their sum is computed by adding components *element-wise.* That is,

$$\mathbf{x} + \mathbf{y} = [x_1 + y_1, x_2 + y_2, \dots, x_d + y_d] \in \mathbb{R}^d.$$

  This sum can only be computed if $\mathbf{x}$ and $\mathbf{y}$ are of the same dimension; it's meaningless to think about combining two vectors of different sizes.

  When thinking about vectors, you typically want to think of each component as having some distinct meaning. For instance, maybe $d = 12$ and x_i represents the price of an Absolute bagel during the ith month of 2021. Similarly, y_i represents the price of a coffee during the ith month. Then, $x_i + y_i$ is the price of a bagel + coffee breakfast in the ith month, and $\mathbf{x} + \mathbf{y}$ helpfully tells you all of the breakfast costs in every month.

Some students were at a higher base level mathematically, so we were able to put some notation in the exposition. Because the course was small (15 students), Clayton and I were also able to give one-on-one guidance to students without the same math background, pointing out which parts were "fine to gloss over" or providing more concrete examples.

Jump To: Table of Contents

Vectors in Python

Now that you know the basics of vectors, it's time to look at how to work with them in Python. You might be thinking, "I don't need anything fancy and new for vectors; I already have lists." And that's true! You already have a fantastic data structure for storing collections of numbers. However, there are a number of things that make lists not the most ideal data structure for the job.

- Lists can hold a mixture of different data types. `[1, 2, "cat", [1, -1]]` is a valid list. Vectors should be more structured and should only contain scalars.
- Vectors make a point of being fixed size. d-dimensional vectors can only be added to other d-dimensional vectors, and it's uncommon to add more elements. On the other hand, lists are variable size, and are built around adding more elements.
- Indeed, lists make it much easier to add new elements than to perform operations on current elements. Just as we add numbers together with `x1 + x2` in Python, one might want to add vectors together elementwise with `v1 + v2`. The syntax for lists does not give you that. Try running the following code:

```
v1 = [1, 2, 3]
v2 = [1, -1, 1]
v1 + v2
```

- In order to add up the elements of that list, we would need to write a for loop.
- Same goes for multiplication:

```
3 * v1
```

While you can write for loops and functions to make those operations possible with lists, it's not ideal because (1) the code will be redundant, and (2) a data structure optimized for vectors can be setup to do those things much more efficiently.

To magically solve our problems, we introduce the **Numpy** package. This provides an `nparray` data structure that stores information in vectors. These arrays behave roughly as we'd want them to, and they make it elegant and efficient to perform mathematical operations on them.

Like `matplotlib` before it, we must start our code with `import numpy` to get access to its tools. We can convert a python list into a numpy array using the function `np.asarray`. Run the following blocks of code to see how vector operations are cleanly implemented.

The Python labs all followed the structure of: exposition, simple coded examples, and exercises for students to do themselves.

For the exercise parts, students all worked on their own laptops as Clayton and I walked around to make sure everyone could pass the exercise and receive individualized attention.

Before moving on to machine learning, we have a few exercises that teach you how to program with numpy and arrays/vectors. A key goal of this section is to solve the following problems using only vector operations, without for loops.

```
# EXERCISE: A course at Columbia has two midterms and a final. Each student
# receives a grade between 0 and 100 on each exam. Create a function which,
# given three vectors containing the score of each student on each exam, returns
# a new vector of the same size that contains the total grade for every student.
# The total grade is 0.2*(first midterm score) + 0.3*(second) + 0.5*(final).
# Your code should work for any number of students in the class.
# Make sure it passes our test case!

def total_grades(midterm1_scores, midterm2_scores, final_scores):
  ...
```

```
midterm1_scores = np.asarray([100, 90, 70])
midterm2_scores = np.asarray([90, 80, 90])
final_scores = np.asarray([80, 80, 100])
grades = np.asarray([87, 82, 91])
assert((total_grades(midterm1_scores, midterm2_scores, final_scores) == grades).all())
```

```
# OPTIONAL EXERCISE: Given a list of vectors (yes, that is possible), return the
# vector that has the smallest norm.
def smallest_norm(list_of_vector):
  ...
```

CHECKPOINT #3.

[Jump To: Table of Contents]

Perceptron: A Classification Algorithm

In 1958, Frank Rosen published a paper that introduced the **Perceptron**, a neurologically-inspired model for learning and information storage that he claimed would form the foundation of artificial intelligence. Check out the paper; he has a lot to say about neuroscience, and he speaks of his model very highly:

> The present theory, being derived from basic physical variables, is not specific to any one organism or learning situation. It can be generalized in principle to cover any form of behavior in any system for which the physical parameters are known. A theory of learning, constructed on these foundations, should be considerably more powerful than any which has previously been proposed.

He defines a Perceptron as an artificial neuron, which takes many signals as input and gives off an output. (The inputs are analogous to the dendrites of a neuron, and the output its axon and resulting impulse.) For our purposes, we let a perceptron be a function $f_{\mathbf{w}}$ parameterized by d-dimensional $\mathbf{w}$ that maps a d-dimensional input $\mathbf{x}$ to a scalar output with

$$f_{\mathbf{w}}(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{x} < 0. \end{cases}$$

This Perceptron is also frequently called a **linear threshold function** or a **halfspace** because it subdivides the input space into two regions, one labeled 1 and the other -1. This is a *classifier* because the output belongs to one of two discrete categories.

How might this be phrased as a machine learning problem? Using the notation from before, let the inputs be d-dimensional vectors and the labels be +1 and -1 (which implies Boolean-valued labels). For an unknown test set

$$(\mathbf{x}^{(1)\prime}, y^{(1)\prime}), (\mathbf{x}^{(2)\prime}, y^{(2)\prime}), \ldots, (\mathbf{x}^{(m)\prime}, y^{(m)\prime}),$$

our goal is to find some $\mathbf{w}$ such that $f_{\mathbf{w}}(\mathbf{x}^{(i)\prime}) = y^{(i)\prime}$ for most choices of i. Because we don't know that data, our best hope is to find Perceptron that perfectly fits a training set,

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)}).$$

That is, $f_{\mathbf{w}}(\mathbf{x}^{(i)}) = y^{(i)}$ for *all* choices of i. The **Perceptron learning algorithm** is a procedure that obtains such a $\mathbf{w} \in \mathbb{R}^d$ from a training sample. We refer to $\mathbf{w}$ as a *hypothesis* throughout.

Here's some rough intuition for the algorithm: We start with some hypothesis $\mathbf{w}$ and predict the outcomes of training samples one-by-one. If we correctly guess the label of an input, that's great! The hypothesis was correct, and there is no need to change it. If not, then we modify the hypothesis to make it more likely to classify the sample right the next time we see it. After looking at all the samples, we start over and continue looping through the training data until we complete a full loop without classifying any of the training samples right. Then, we're

Here's one potential red flag about the above algorithm: How do we know that it will terminate? Indeed, there's a very real concern with the Perceptron algorithm that it may not converge, and you could encounter an infinite loop. Work through the following two exercises and come up with some ideas about what causes the Perceptron learning algorithm to terminate or not. When you finish the two, chat with a TA and explain your thoughts.

EXERCISE: For $d = 2$ and $n = 3$, consider the training dataset $(\mathbf{x}^{(1)}, y^{(1)}) = ([1, 1], -1)$, $(\mathbf{x}^{(2)}, y^{(2)}) = ([1, -1], 1)$, $(\mathbf{x}^{(3)}, y^{(3)}) = ([-2, 1], 1)$. Work out the perceptron algorithm by hand and show how $\mathbf{w}$ will be updated until the algorithm terminates.

EXERCISE: For $d = 2$ and $n = 4$, consider the training dataset $(\mathbf{x}^{(1)}, y^{(1)}) = ([1, 0], 1)$, $(\mathbf{x}^{(2)}, y^{(2)}) = ([0, 1], -1)$, $(\mathbf{x}^{(3)}, y^{(3)}) = ([-1, 0], 1)$, $(\mathbf{x}^{(4)}, y^{(4)}) = ([0, -1], -1)$. Work out the first two rounds of Perceptron and explain why you think it will not converge.

CHECKPOINT #4.

Your next task is to implement the Perceptron learning algorithm. Like we've done before, we'll do it in pieces and combine them together to produce the algorithm. Then, we'll test it out on a cool digit-recognition application.

First, implement a Perceptron $f_{\mathbf{w}}$ for a fixed $\mathbf{w} \in \mathbb{R}^d$. $\mathbf{w}$ should be a vector of arbitrarily dimension d (not just 2). The test cases should pass.

```
[ ]
import numpy as np

# EXERCISE: Create a function that returns 1 if the dot product of x and w is at
# least 0 and -1 otherwise. (This implements f_w(x).)
def perceptron(w, x):
  ...
```

```
[ ]
# Test cases
assert(perceptron(np.asarray([1,1]), np.asarray([1,-1])) == 1)
assert(perceptron(np.asarray([1,2,3,4]), np.asarray([1,-1,1,-1])) == -1)
assert(perceptron(np.asarray([1,2,3,4,5]), np.asarray([1,-1,1,-1,1])) == 1)
```

Your `perceptron` function powers the following functions, which can be employed to draw a training or testing sample from some "true" Perceptron, `w_true`. No need to edit the code, although you should run this and the next block to ensure that your `perceptron` function works properly. Feel free to skim it to try to understand what we're doing.

An important design principle in the labs was to build up to a relatively advanced concept related to neural networks from simple modular exercises.

In this case, the example is the Perceptron algorithm, the simplest building block of a neural network.

Interactive Python code allows students to experiment and test various cases to understand an abstract idea better.

In this case, students code to try Perceptron on simple small datasets.

The one-on-one instructional aspect of the class also allowed Clayton and I to have exercises that gave students the chance to verbally “explain what this is doing.” This was helpful to know where each student was at, and to develop a closer instructor-student relationship. Clayton and I would go onto provide mentorship to several of our students!

Jump To: Table of Contents

Digit Classification: Your First "Real" ML Problem

Pat yourself on the back for what you've already accomplished today. You created an ML classification algorithm that perfectly fits training data and also performs well on test data. We'll wrap up today with a demo of the Perceptron learning algorithm on a "real" learning problem: **digit classification**.

The goal of digit classification is to assign a handwritten digit its correct numerical value (e.g. "1", "2", "3", ...). One of the most famous datasets for classification in ML is [MNIST], a large set of pixelated images of handwritten digits that people attempt to classify.

Here, we show that perceptron can make a good classifier of a simpler digit recognition task, which we source from [this tutorial]. This dataset has fewer and lower-resolution images than MNIST, which makes it easier for our purposes. We also simplify the task by distinguishing only two digits (e.g. "0" vs "1"), rather than ten digits.

```
# Choose which two digits you'd like to distinguish between.
# We use 0 vs 1 by default, but you can choose any two.
digit1 = 0
digit2 = 1
assert(digit1 != digit2)
```

With the two digits selected, we import all of those images from the dataset and organize them into training data. This requires collecting all of the pixel-vector inputs into a matrix and all labels into a +1/-1 vector. Because each digit is represented by an 8x8 image, it can be represented by a vector in $\mathbb{R}^{64}$, and we'll end up running Perceptron to learn a 64-dimensional $\mathbf{w}$.

(In ML, most of the work ends up consisting of data cleaning and processing! Next week, we'll do this more efficiently using built-in functions, but today we'll do it in a more granular way so you can more clearly see which operations are done.)

```
# Uses the sklearn package (more on that next time) to import the dataset
from sklearn import datasets
digits = datasets.load_digits()

# Filters out all digits except the two we want
target_digits = (digits.target == digit1) | (digits.target == digit2)
digits_y = digits.target[target_digits]
digits_x = digits.data[target_digits]
digits_images = digits.images[target_digits]
```

Most of the labs concluded with an application of the machine learning tool of the week to a real dataset. In this case, students applied their Perceptron algorithm to a simplified MNIST dataset of 0's and 1's. MNIST is a handwriting classification dataset where the examples are images of digits.

Students found it fascinating that they could go from zero programming experience to building a "simple neural network" that accurately classifies handwritten digits.

It's time to run the Perceptron learning algorithm to obtain some $\mathbf{w} \in \mathbb{R}^{64}$ that perfectly fits the training data.

```
w = perceptron_learning_algorithm(x_train, y_train)
```

We conclude by evaluating the performance of `w` on the test data and by visualizing some of the test data it classified wrong.

```
print("Training accuracy: {}".format(evaluate_perceptron(x_train, y_train, w)))
print("Testing accuracy: {}".format(evaluate_perceptron(x_test, y_test, w)))

n_test = len(y_test)
incorrect_test_indices = []

for i in range(n_test):
  if perceptron(w, x_test[i]) != y_test[i]:
    incorrect_test_indices.append(i)


if label is []:
  print("No errors on test data!")
else:
  y_orig_test_incorrect = y_orig_test[incorrect_test_indices]
  images_test_incorrect = images_test[incorrect_test_indices]
  _, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
  n = len(digits_y)
  perm = np.random.permutation(len(incorrect_test_indices))
  for ax, image, label in zip(axes,
                              images_test_incorrect[perm],
                              y_orig_test_incorrect[perm]):
    ax.set_axis_off()
    if label == digit1:
      pred = digit2
    else:
      pred = digit1
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title("True: {}, Predicted: {}".format(label, pred))
```

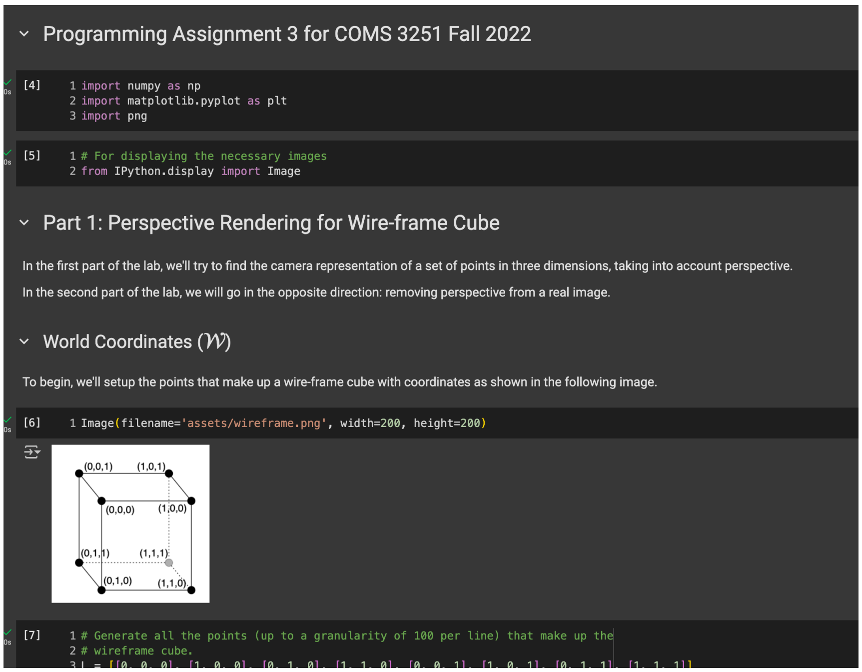
The algorithm will do much better on some pairs of numbers than others because some are much more similar than others. Try a few different pairs and see which ones it performs better on.

CHECKPOINT #6. Show a TA how your Perceptron performed.

[Jump To: Table of Contents]

Python Programming Assignment: Computational Linear Algebra

I designed this example programming assignment as a homework assignment for students in Computational Linear Algebra. Students in this class have experience coding, but possibly not in Python. At this point, they have gotten the hang of basic scientific computing libraries, so this programming assignment brought their new knowledge of Python to bear on a concrete demonstration of some abstract linear algebra ideas from class: linear transformations, bases, and change-of-bases.



Programming Assignment 3 for COMS 3251 Fall 2022

```
[4]
import numpy as np
import matplotlib.pyplot as plt
import png
```

```
[5]
# For displaying the necessary images
from IPython.display import Image
```

Part 1: Perspective Rendering for Wire-frame Cube

In the first part of the lab, we'll try to find the camera representation of a set of points in three dimensions, taking into account perspective.

In the second part of the lab, we will go in the opposite direction: removing perspective from a real image.

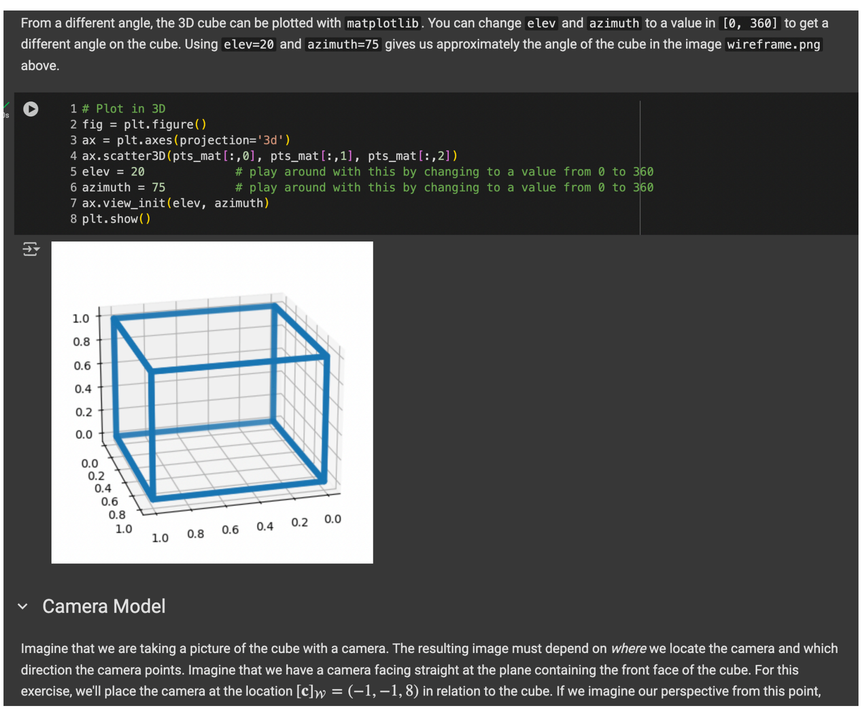
World Coordinates ($\mathcal{W}$)

To begin, we'll setup the points that make up a wire-frame cube with coordinates as shown in the following image.

```
[6]
Image(filename='assets/wireframe.png', width=200, height=200)
```



```
[7]
# Generate all the points (up to a granularity of 100 per line) that make up the
# wireframe cube.
L = [[0, 0, 0], [1, 0, 0], [0, 1, 0], [1, 1, 0], [0, 0, 1], [1, 0, 1], [0, 1, 1], [1, 1, 1]]
```



From a different angle, the 3D cube can be plotted with `matplotlib`. You can change `elev` and `azimuth` to a value in `[0, 360]` to get a different angle on the cube. Using `elev=20` and `azimuth=75` gives us approximately the angle of the cube in the image `wireframe.png` above.

```
# Plot in 3D
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(pts_mat[:,0], pts_mat[:,1], pts_mat[:,2])
elev = 20           # play around with this by changing to a value from 0 to 360
azimuth = 75        # play around with this by changing to a value from 0 to 360
ax.view_init(elev, azimuth)
plt.show()
```


Camera Model

Imagine that we are taking a picture of the cube with a camera. The resulting image must depend on *where* we locate the camera and which direction the camera points. Imagine that we have a camera facing straight at the plane containing the front face of the cube. For this exercise, we'll place the camera at the location $[\mathbf{c}]_{\mathcal{W}} = (-1, -1, 8)$ in relation to the cube. If we imagine our perspective from this point,

The point of this assignment is to connect the abstract ideas of change-of-basis from class to being able to play with the perspective of points in `matplotlib` and some digital images.

This programming assignment was dense, so my recitation for that week attempted to clarify student questions and provide an overview and hints to this assignment.

The lab begins with some interactive plotting to make sure students can grasp the inherently 3D idea at hand from as many different perspectives as possible.

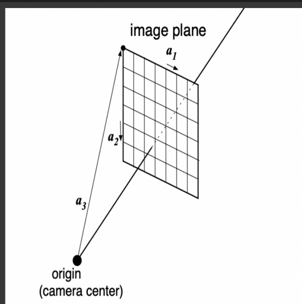
Jump To: Table of Contents

Problem 1: Finding F_3 (Camera Coordinates → Pixels)

Light travels in a straight line from $\mathbf{p}$ to the camera center $\mathbf{0}$ (we assign the origin to the camera center), and, along the way, it intersects the image plane at some point $\mathbf{q} \in \mathbb{R}^3$.

Note the difference between a point's pixel and its camera coordinates. A pixel is a rectangle on the image array canonically denoted by the 2D coordinates of its upper-left corner. Using the *camera coordinate system*, any point $\mathbf{q} \in \mathbb{R}^3$ has coordinates $[\mathbf{q}]_{\mathcal{A}} = (x_1, x_2, x_3)$, so $\mathbf{q} = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + x_3\mathbf{a}_3$. But, if $\mathbf{q}$ is on the image plane, then the coordinate x_3 is simply 1, and the pixel value of $\mathbf{q}$ on the image array is simply (x_1, x_2). In general, $F_3(x_1, x_2, x_3) = (x_1, x_2)$. This is a simple linear transformation, $F_3 : \mathbb{R}^3 \to \mathbb{R}^2$. We consider representing it from basis $\mathcal{A}$ to $\mathcal{C}$.

```
[11] Image(filename='assets/camera_coords.png', width=400, height=400)
```



We will implement F_3 below.

assume that `F3_mat` always uses `F3` when we call it.

```python
def F3(q: np.ndarray) -> np.ndarray:
    """
    Implement the function `F3`, as described above.
    """
    ...

def F3_mat(A, C, F=F3):
    """
    Implement the function `F3_mat`, as described above.
    """
    ...
    return M
```

The exercises that students were responsible for in the lab were modular functions that each implemented a key linear transformation.

The modularity was important to my design principle: I wanted students to get the sense that, by constructing the correct building blocks, they were able to do something very nontrivial by putting them together.

Like my **Problem Set: Math for ML**, the driving philosophy behind this programming assignment is giving students a chance to discover something themselves.

Displaying the Wire-Frame Cube

Now, put this all together to get the pixel coordinates for the wire-frame cube. Before we do anything, we need to account for the shift in origin from world coordinates to camera coordinates. Remember that we located our camera at $(-1, -1, -8)$ in world coordinates. In order to use the camera coordinate system, we need to locate the camera at $(0, 0, 0)$, so translate each of the points of the wire-frame cube by adding $(1, 1, 8)$ to them.

```python
[ ] # P_W is the 3x1200 matrix containing each of the points as columns [p]_W
    shifted_pts = [v + np.array([1, 1, 8]) for v in pts]
    P_W = np.array(shifted_pts).T
```

First, apply F_1 to each $[\mathbf{p}]_{\mathcal{W}}$ to get $[\mathbf{p}]_{\mathcal{A}}$, the points in camera coordinates.

```python
[ ] A = np.array([[1/100, 0, 0],
                  [0, 1/100, 0],
                  [0, 0, 1]])
```

```python
# Apply F1, the change of basis
P_A = F1(P_W, A)
```

Second, we apply F_2 to $[\mathbf{p}]_{\mathcal{A}}$ in camera coordinates to obtain $[\mathbf{q}]_{\mathcal{A}}$ in the image plane.

```python
[ ] # Apply F2, the "projection" onto the image plane
    Q_A = np.array([F2(col) for col in P_A.T]).T
```

Finally, apply F_3 to each $[\mathbf{q}]_{\mathcal{A}}$ to get $[T_3(\mathbf{q})]_{\mathcal{C}}$, the pixel coordinates of the vectors on the image plane.

```python
[ ] A = np.array([[1/100, 0., 0.],
                  [0., 1/100, 0.],
                  [0., 0., 1.]])
    C = np.array([[1/100, 0.],
                  [0., 1/100]])
    Q_C = F3_mat(A, C) @ Q_A
```

Jump To: Table of Contents

Part 2: Perspective Rectification

In **Part 1**, we found the appropriate image for a given set of points, taking into the account the perspective from a fixed camera. The goal for this main part of the lab is the opposite: we want to *remove* perspective from an image of a flat surface. Essentially, we want to synthesize a new image that completely lacks the perspective imbued from how we took the image, but shows us the flat surface (in this case, a whiteboard) head-on.

The specific image we will be playing with will be `board.png`, included in your `assets` directory.

In order to do this, we will again have to use different coordinate systems and a change of basis. Think of the original image as a grid of rectangles that are each assigned a color (the pixels). Each of these pixels corresponds to a parallelogram in the plane of the whiteboard. To get the perspective-free image, we must assign each of these parallelograms the corresponding color from the rectangle in the original image.

Our goal is thus to find a function that maps from pixel coordinates (the coordinates of a point in our image `board.png`) to coordinates that exist in the plane of the whiteboard. It is already clear that we will be dealing with *two* coordinate systems: the coordinate system for the image, and the coordinate system for the whiteboard.

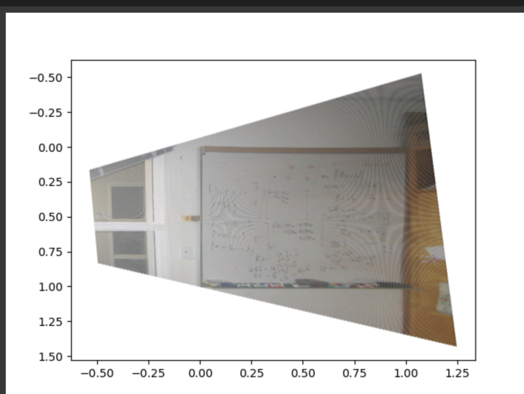
[25]
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

[26]
```
img = mpimg.imread('assets/board.png')
imgplot = plt.imshow(img)
```

Many of the coding assignments in CLA culminated in doing something to actual images, which gives students a visceral real world example of the techniques of linear algebra in action.

At the end of the assignment, you should see the whiteboard head on, with perspective removed. Here's a "spoiler" cell that shows what you should see at the end, `board_final.png`. If you don't want spoilers, you can skip the following cell until the end and just use it to check your final image. If you do want to see what should happen at the end, uncomment the line of code `Image(filename='asserts/board_final.png')` below.

[27]
```
# THIS CELL IS A *SPOILER*!
Image(filename='assets/board_final.png')
```



In this lab, students use the modular functions they implemented (each corresponding to a specific linear transformation) to “flatten” the image of a whiteboard. Pretty nontrivial!

Jump To: Table of Contents