Lecture 1: ECE C147/C247, Neural Networks & Deep Learning

General info:

ECE C147/C247, Neural Networks & Deep Learning
Winter quarter, UCLA AY 2021-22

Instructor:

Prof. Jonathan Kao
OH: TBD

TAs:

Yang Li
Pan Lu
Tonmoy Monsoor
Tianyi Wang

Some high-level thoughts about this class:

- Deep learning is a very relevant topic for many different applications today, and we hope this class will be useful for people going to a diversity of areas.
- We're living in a renaissance of deep learning, and that's exciting.
- This class will (hopefully) be fun, but also a lot of work.
- Ask questions!

Zoom online lectures

A handful of lectures, discussions, and office hours, will be carried out on Zoom. These lectures will be recorded. Note that lectures held in Mong Learning Center will be recorded, too.

We encourage you to attend lectures live (MW 4-5:50pm Pacific Time), as there will be an opportunity for interaction and asking questions. If you do not wish to appear in a lecture video, because we are recording them for the benefit of all students, we ask that you opt out by not attending.

As per notice by UCLA's Office of Information Technology:

> *Where recording is permitted, it is permitted only by the host (typically instructor or meeting chair). Students in a class and/or meeting participants and any student-hosted meetings are prohibited from recording of any kind.*

Lectures will be held at `https://ucla.zoom.us/j/93700609639`.

Zoom online lectures (cont.)

By default, all students will be muted during live lectures. There are two ways of having questions throughout lecture answered:

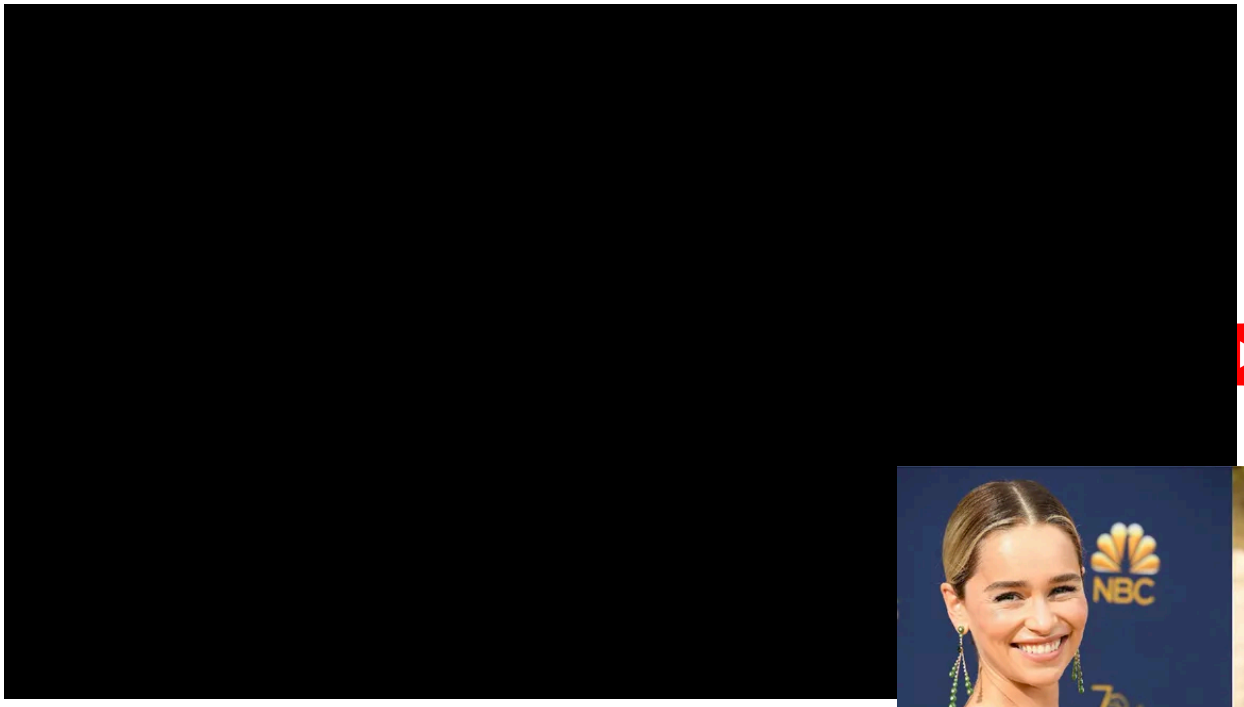
1. Raise your (virtual) hand.

   If you are on a mobile device, there is a "Raise Hand" button. If you are on a Desktop, click on "Participants" and then "Raise Hand." If you raise your hand, but you no longer need to ask your question, you can undo the "Raise Hand." When there is a natural breaking point in the lecture, Prof. Kao will unmute students with their hands raised, so that they can ask their question. Note, you will also have to unmute yourself.

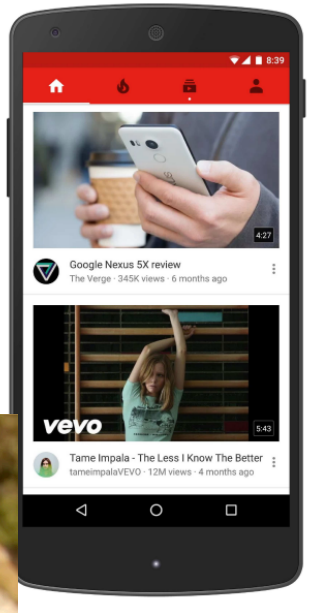
2. Through the chat functionality.

   During lecture, you will be able to ask questions seen by everyone, or only by the TAs. We encourage you to ask questions seen by everyone, which may benefit others. Do not send chat messages to Prof. Kao, as he will not actively monitor these during the lecture. The TAs will answer questions over chat.

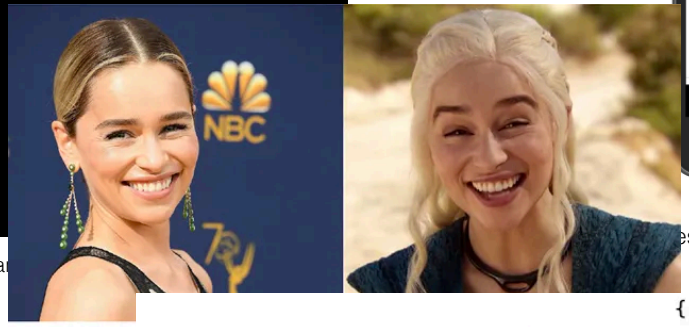
Deep learning is regularly used in modern AI.



https://engineering.fb.com/2016/06/01/core-data/introducing-deeptext-facebook-s-text-understa



esearch.google.com/en//pubs/archive/45530.pdf



Meta

https://www.forbes.com/sites/bernardmarr/2016/12/29/4-amazing-ways-facebook-uses-deep-learning-to-learn-everything-about-you/?sh=478f6eddccbf

https://viso.ai/computer-vision/deepface/


```
{
  "emotion":{
     "angry":7.603101671639384e-14,
     "disgust":2.7474185705216866e-21,
     "fear":1.688688161735822e-14,
     "happy":100.0,
     "sad":4.205067717644173e-10,
     "surprise":7.103817571484745e-13,
     "neutral":4.4851553027136504e-08
  },
  "dominant_emotion":"happy",
  "age":31,
  "gender":"Woman",
  "race":{
     "asian":0.9087088517844677,
     "indian":1.1444833129644394,
     "black":0.09399998234584928,
     "white":66.56872034072876,
     "middle eastern":16.655877232551575,
     "latino hispanic":14.628209173679352
  },
  "dominant_race":"white"
}
```

Why a class on deep learning?

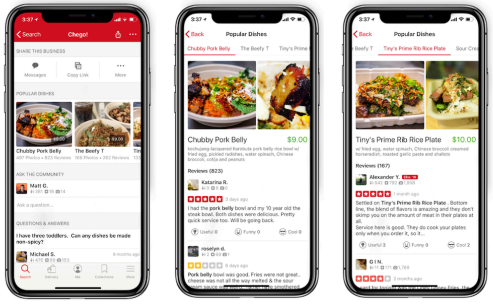
https://www.youtube.com/watch?v=40riCqvRoMs

Deep learning is regularly used in modern AI.

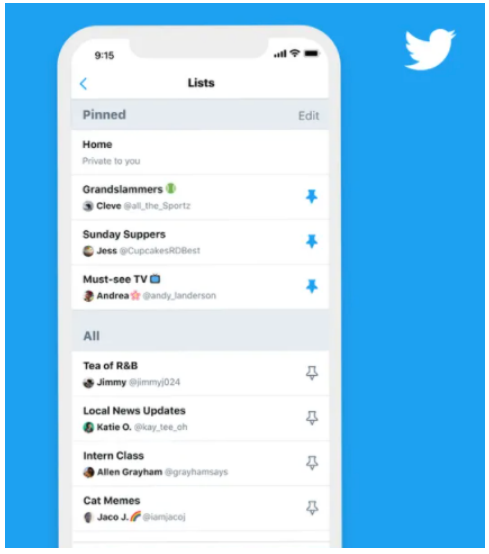
Discovering Popular Dishes with Deep Learning

Introduction

Yelp is home to nearly 200 million user-submitted reviews and even more photos. This data is rich with information about businesses and user opinions. Through the application of cutting-edge machine learning techniques, we're able to extract and share insights from this data. In particular, the Popular Dishes feature leverages Yelp's deep data to take the guesswork out of what to order. For more details on the product itself, check out our product launch blog post.

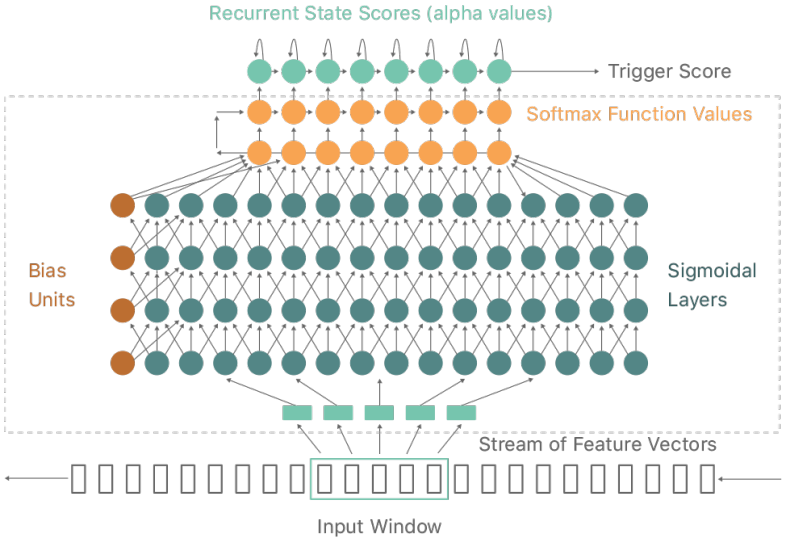


https://engineeringblog.yelp.com/2019/10/discovering-popular-dishes-with-deep-learning.html



https://blog.twitter.com/engineering/en_us/topics/insights/2017/using-deep-learning-at-scale-in-twitters-timelines

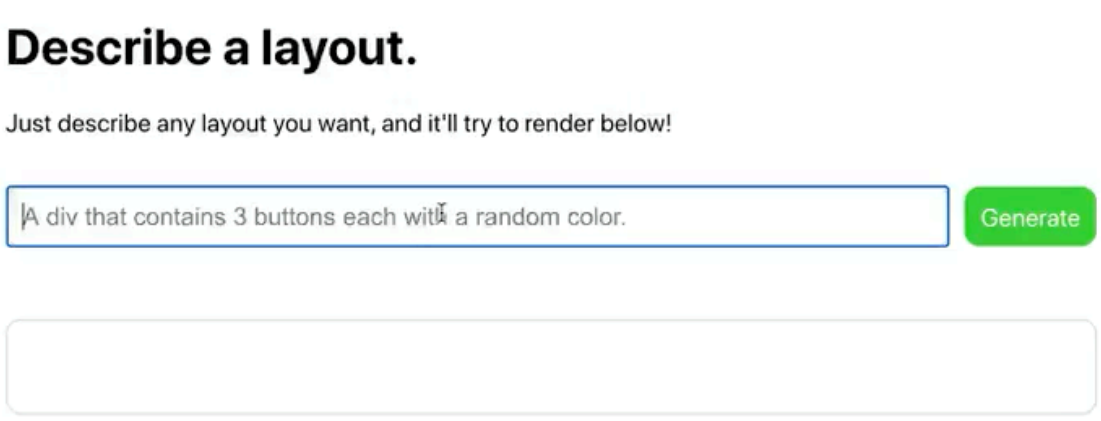
Alexa, Siri, Cortana



https://machinelearning.apple.com/2017/10/01/hey-siri.html

Netflix vectorflow (https://medium.com/@NetflixTechBlog/introducing-vectorflow-fe10d7f126b8) and likely recommendation (https://becominghuman.ai/how-netflix-uses-ai-and-machine-learning-a087614630fe)

Deep learning is regularly used in modern AI.



Deep learning is regularly used in modern AI.

TEXT PROMPT an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES

TEXT PROMPT an armchair in the shape of an avocado. . . .

AI-GENERATED IMAGES

TEXT PROMPT a store front that has the word 'openai' written on it. . . .

AI-GENERATED IMAGES

https://www.machinecurve.com/index.php/2021/01/05/dall-e-openai-gpt-3-model-can-draw-pictures-based-on-text/

Why a class on deep learning?

Its applications are diverse.

Why a class on deep learning?

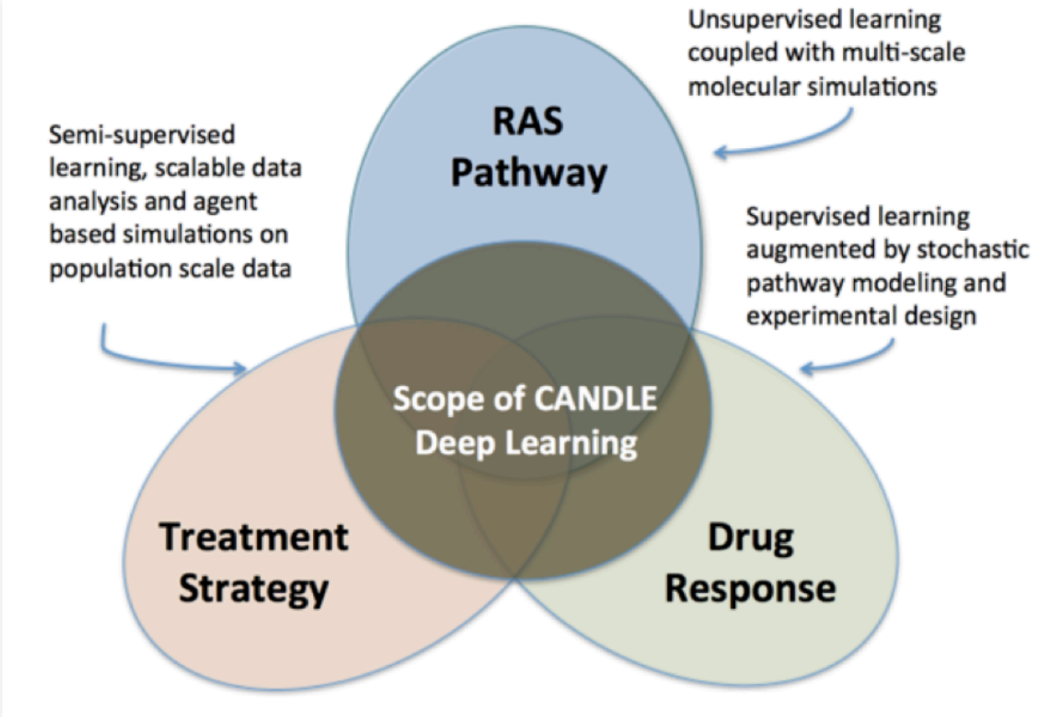
Its applications are diverse.

https://www.youtube.com/watch?v=4HCE1P-m1I8

Why a class on deep learning?

Deep learning in important applications.

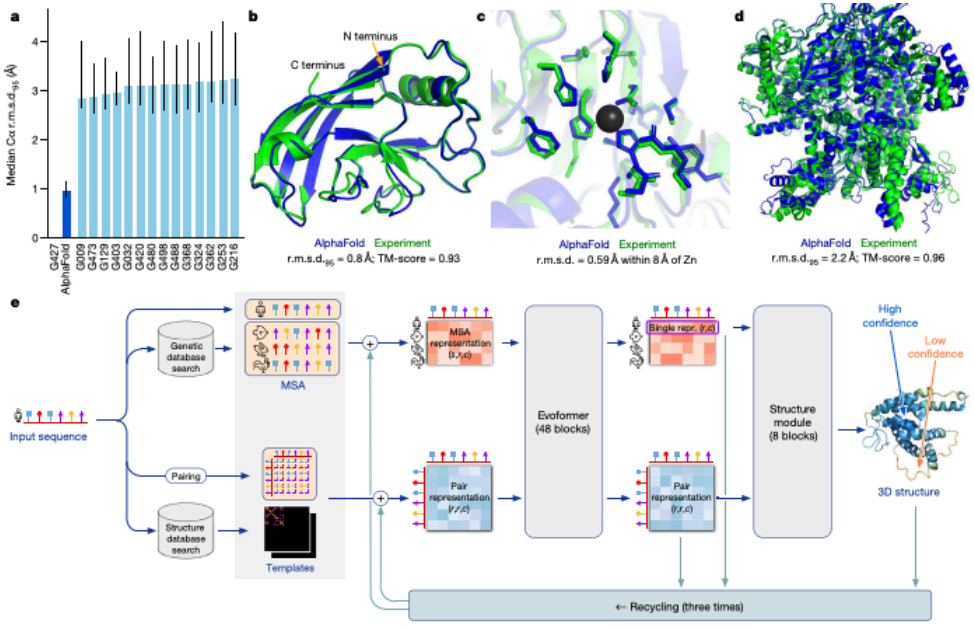
- Fraud detection (e.g., https://www.technologyreview.com/s/545631/how-paypal-boosts-security-with-artificial-intelligence/)
- Many medical endeavors, including:
  - Cancer moonshot: http://candle.cels.anl.gov/



Why a class on deep learning?

Deep learning in important applications.

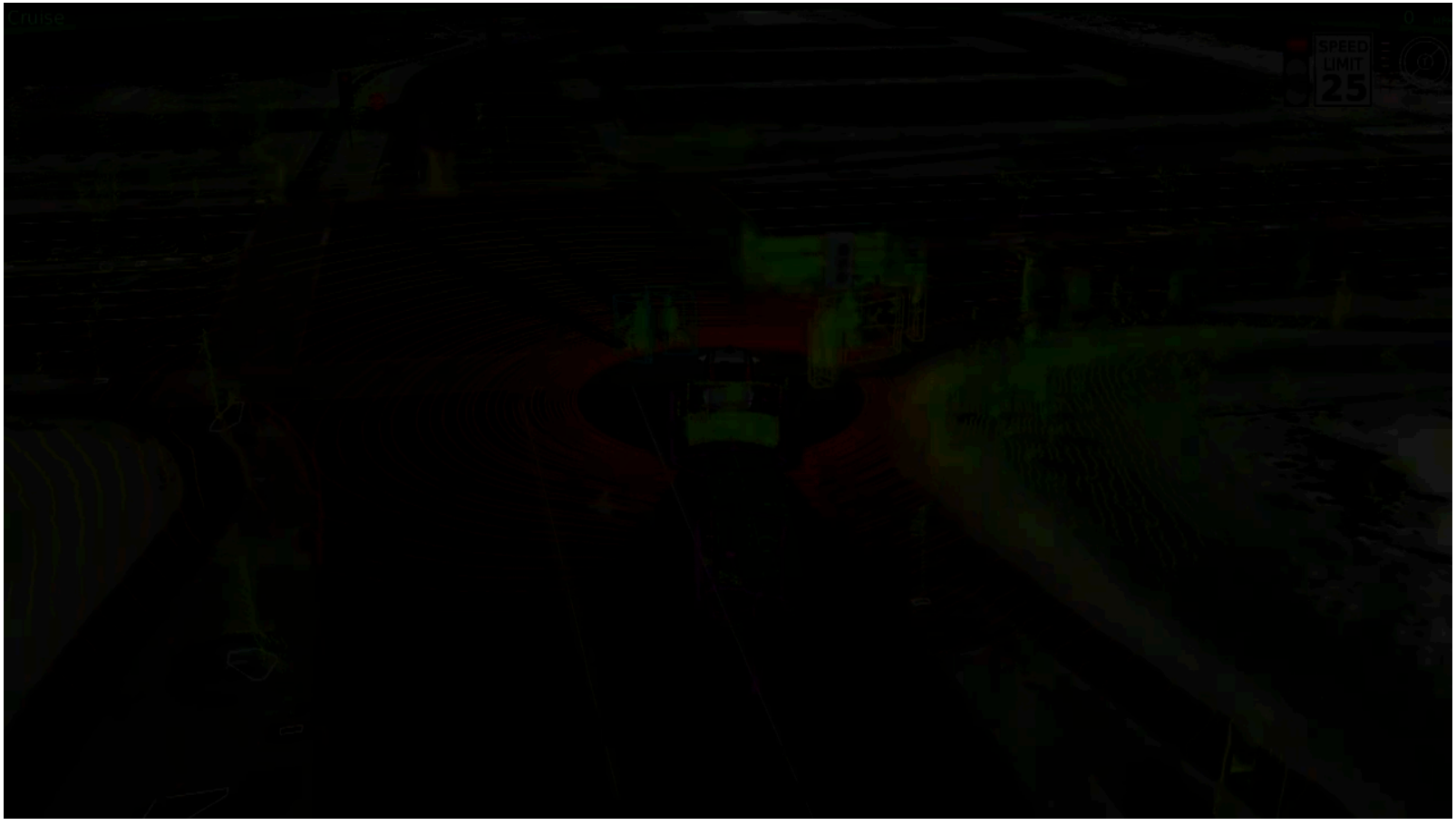
- Fraud detection (e.g., https://www.technologyreview.com/s/545631/how-paypal-boosts-security-with-artificial-intelligence/)
- Many medical endeavors, including:
  - Cancer moonshot: http://candle.cels.anl.gov/
  - Drug discovery (e.g., http://onlinelibrary.wiley.com/doi/10.1002/minf.201501008/full)
  - Protein folding (https://www.nature.com/articles/s41586-021-03819-2)



Why a class on deep learning?

New technologies.

- Self-driving vehicles.

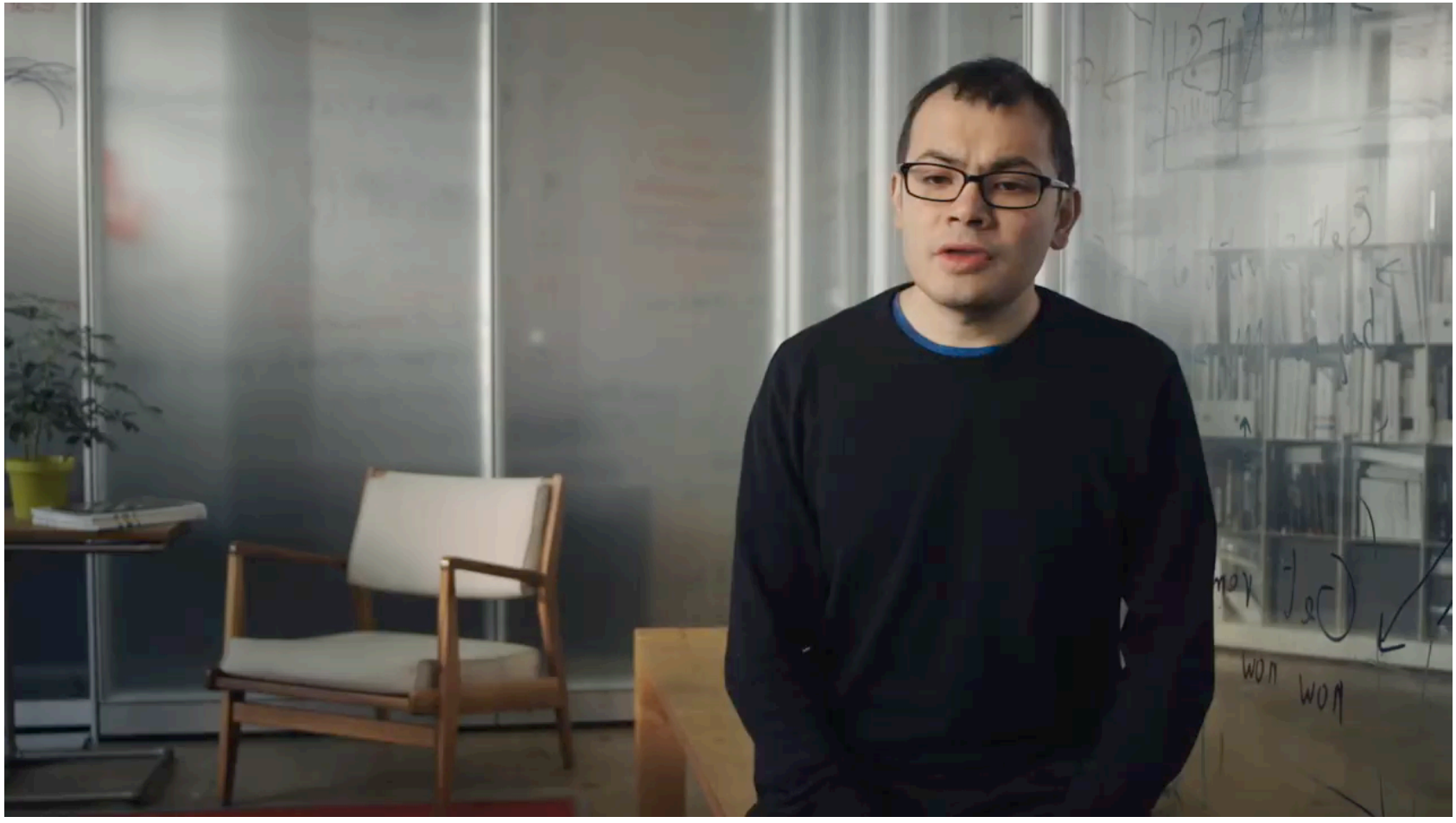


Chris Urmson TED talk, https://www.youtube.com/watch?v=tiwVMrTLUWg

Why a class on deep learning?

Things of the future becoming real today.

- The game of Go.

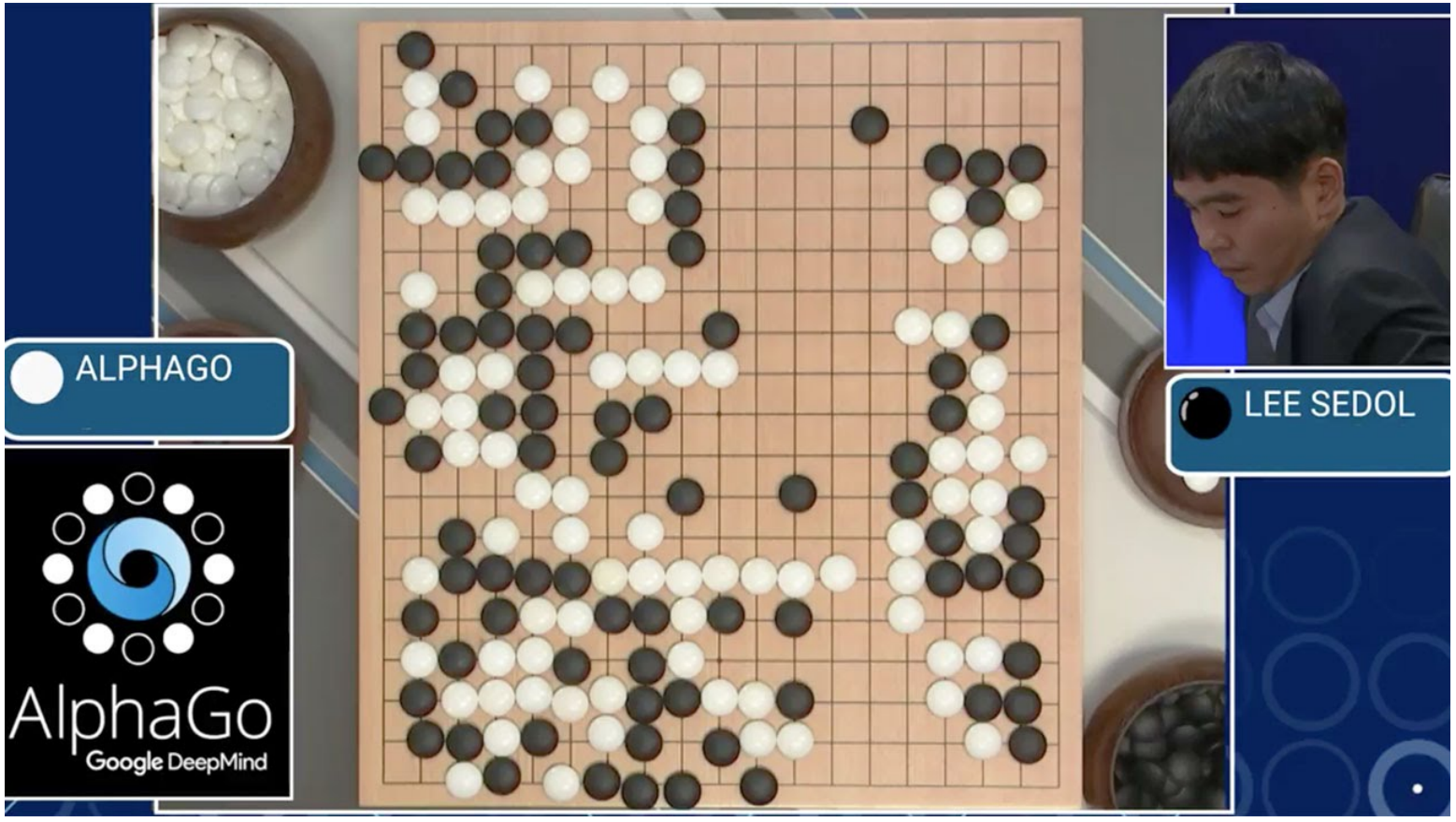


https://www.youtube.com/watch?v=SUbqykXVx0A

Why a class on deep learning?

Things of the future becoming real today.

- Game of Go solved. (AlphaGo def Lee Sedol 4-1, 2016.)



Game 3 of AlphaGo vs Lee Sedol, https://www.youtube.com/watch?v=_OV0Hlj8Fb8

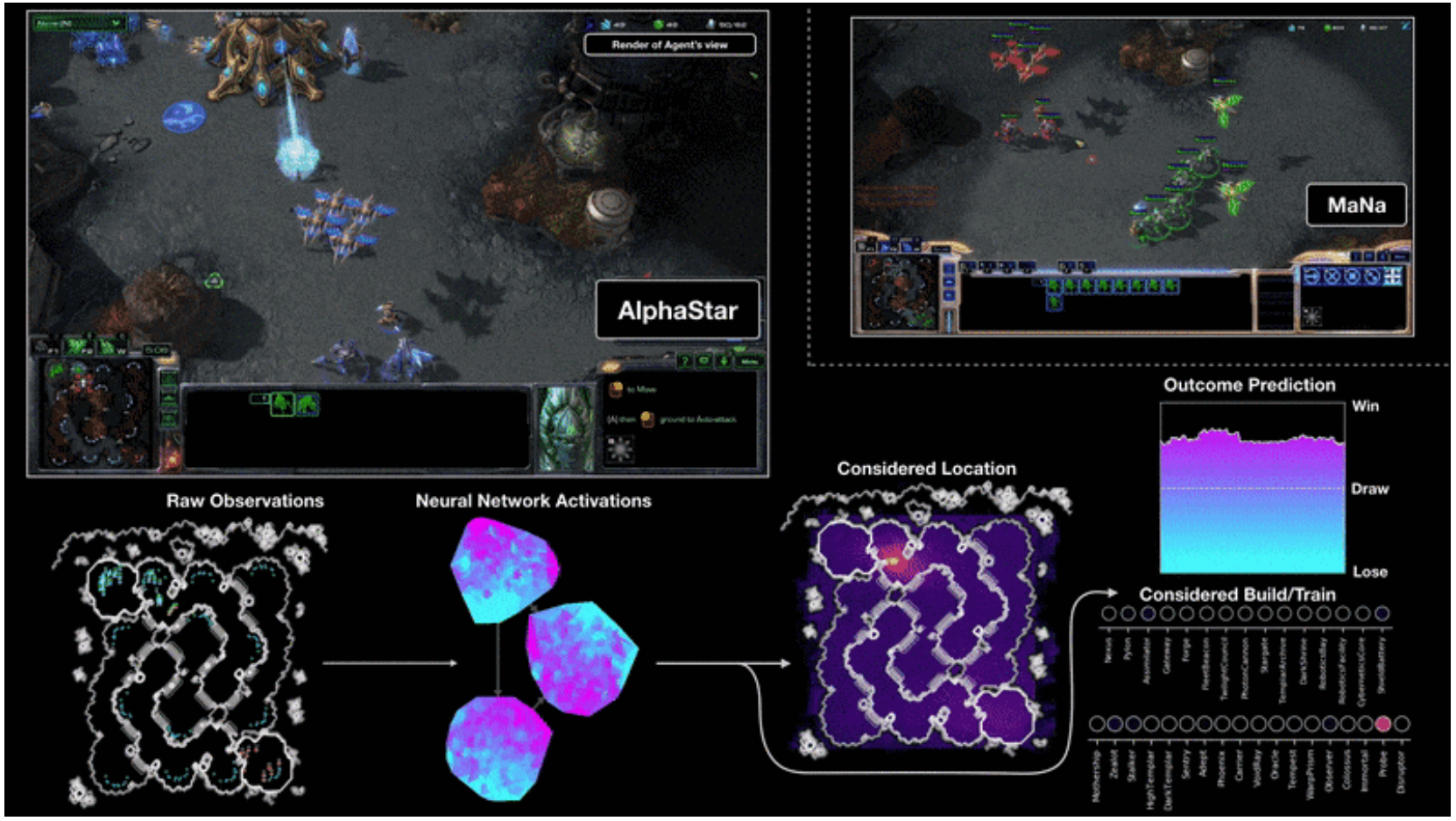
Why a class on deep learning?

Things of the future becoming real today.

- AI's not even requiring human expert data anymore.

Why a class on deep learning?

AI starcraft (https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii)

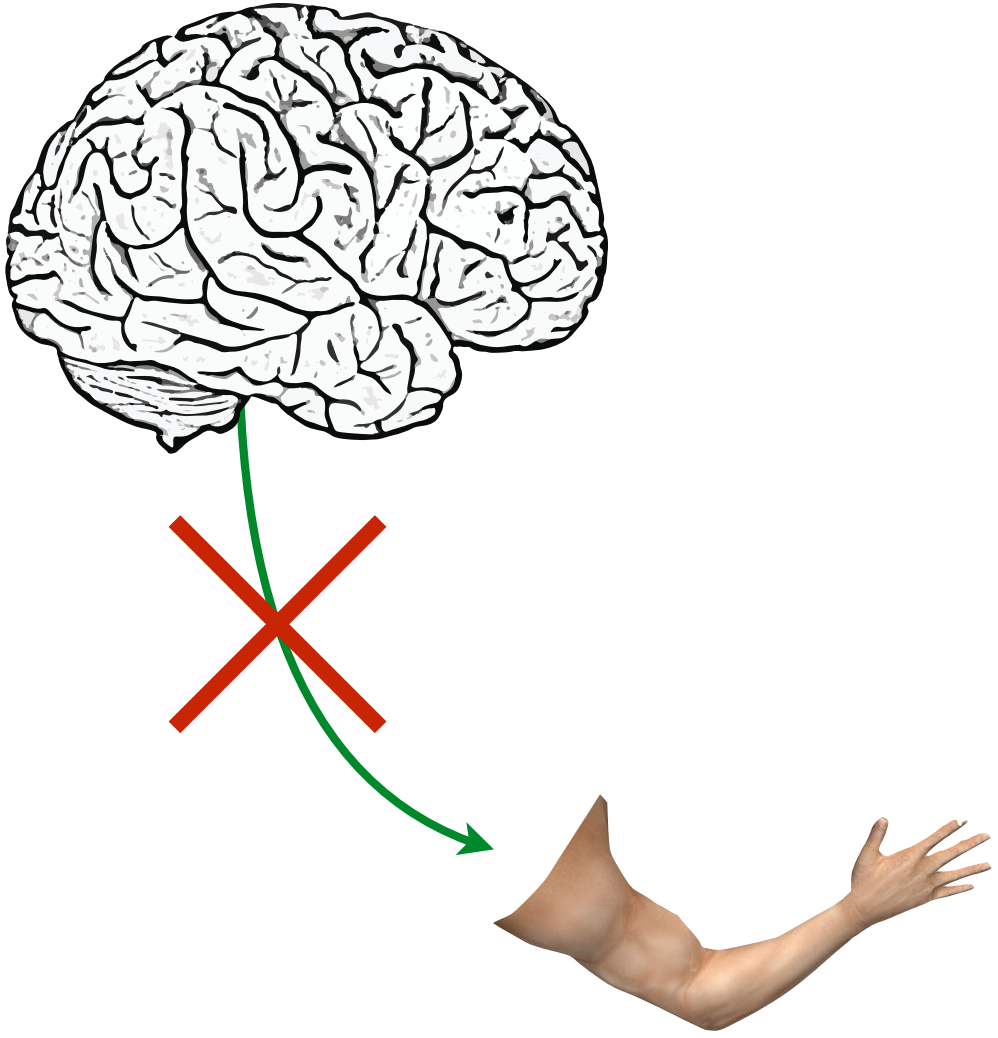


Why a class on deep learning?

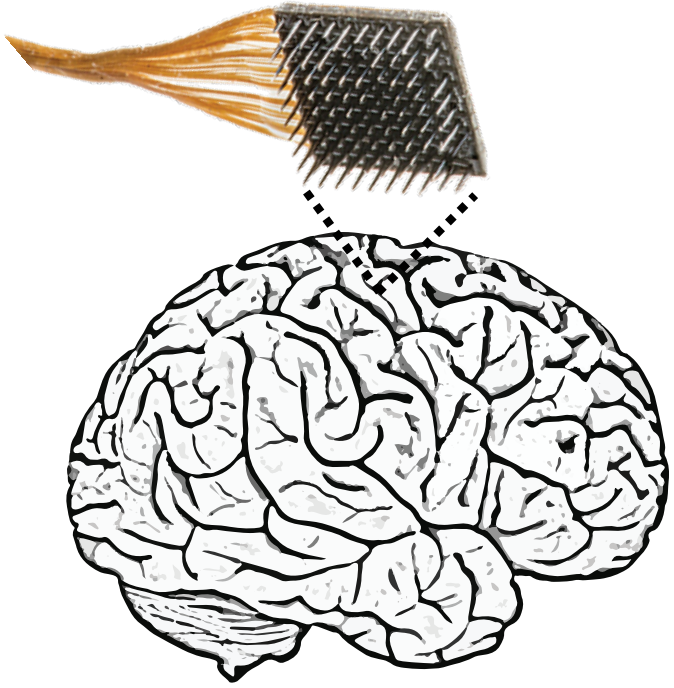
Something closer to my area of expertise:

- Brain-machine interfaces.

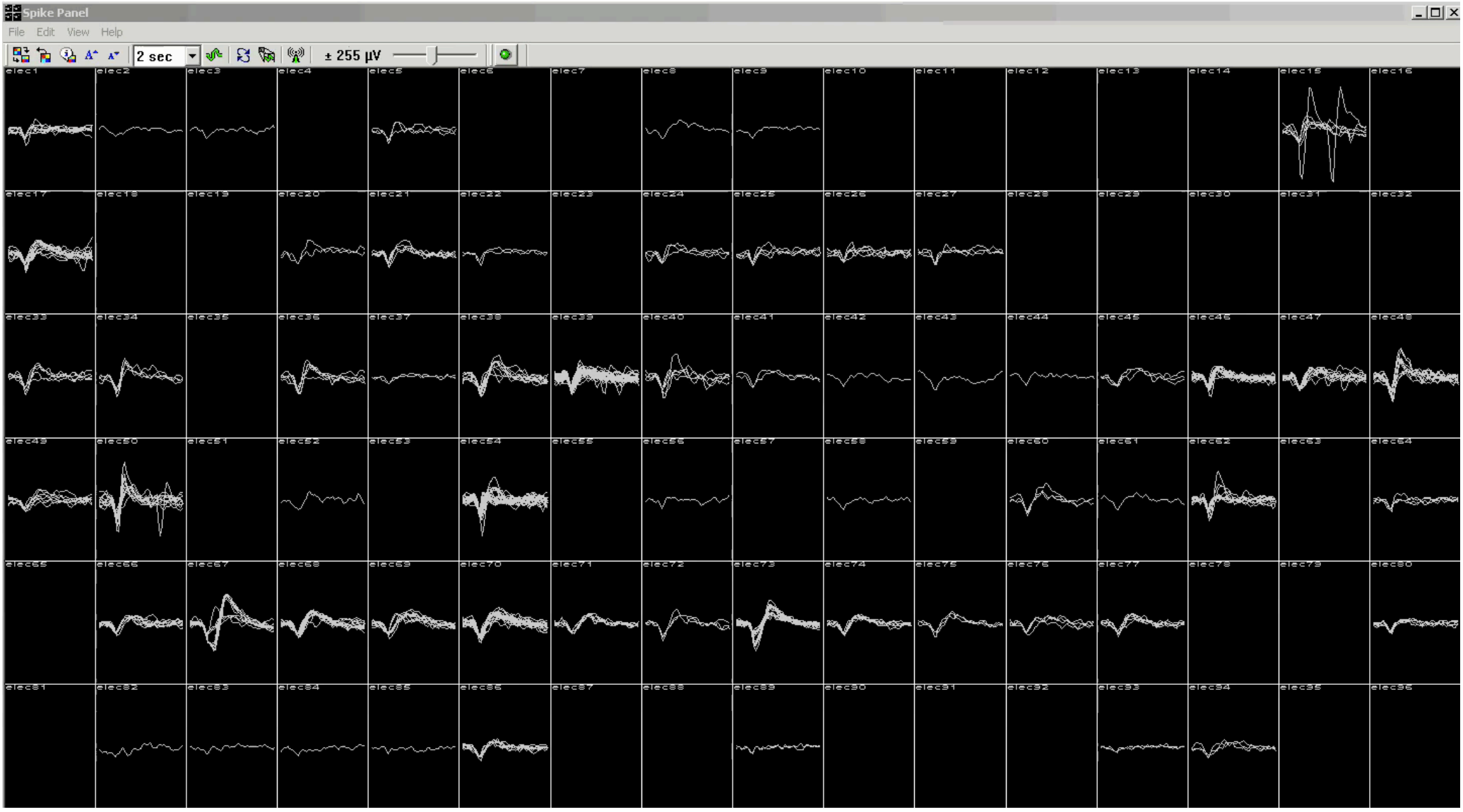
Why a class on deep learning?



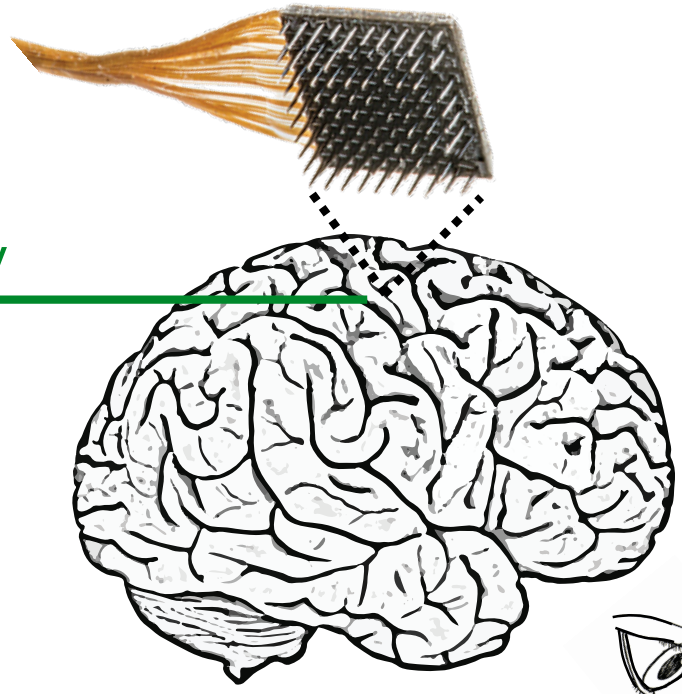
Why a class on deep learning?



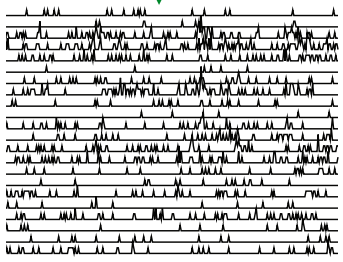
Why a class on deep learning?



Why a class on deep learning?

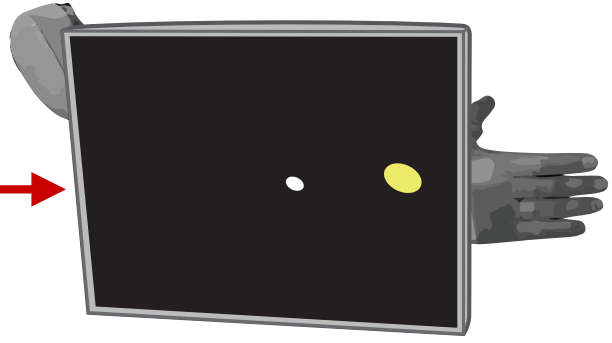


neural activity



Decoder

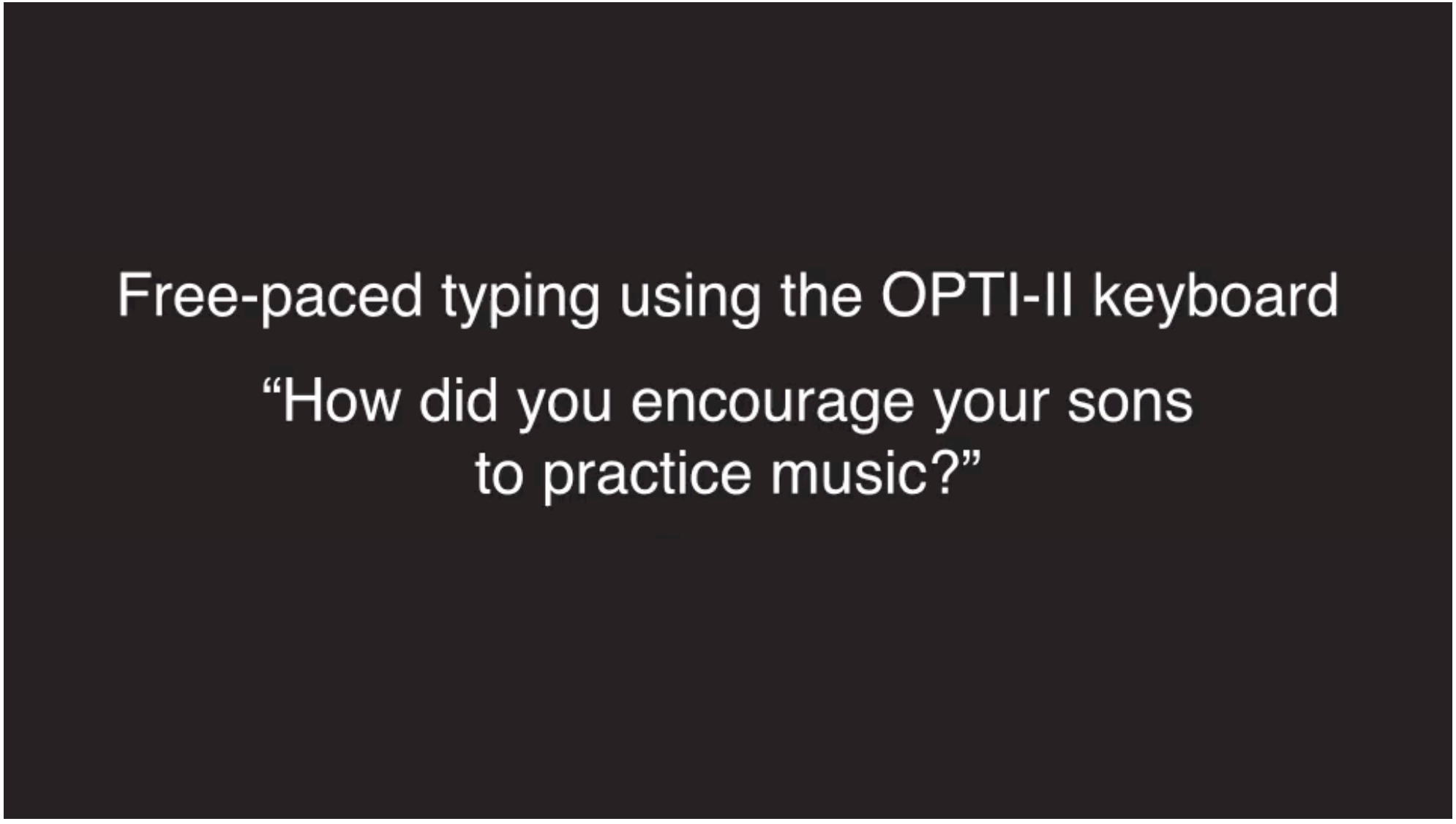
control signals



visual feedback

- Wessberg...Nicolelis, *Nature*, 2000
- Taylor...Schwartz, *Science*, 2002
- Serruya...Donoghue, *Nature*, 2002
- Carmena...Nicolelis, *PLoS Biology*, 2003
- Hochberg...Donoghue, *Nature*, 2006
- Velliste...Schwartz, *Nature*, 2008
- Kim...Black, *J Neural Eng*, 2008
- Ganguly & Carmena, *PLoS Bio*, 2009
- O'Doherty...Nicolelis, *Nature*, 2011
- Ethier...Miller, *Nature*, 2012
- Hochberg...Donoghue, *Nature*, 2012
- Collinger…Schwartz, *The Lancet*, 2013
- Wodlinger…Collinger, *J Neural Eng*, 2015
- Aflalo…Anderson, *Science*, 2015

Why a class on deep learning?



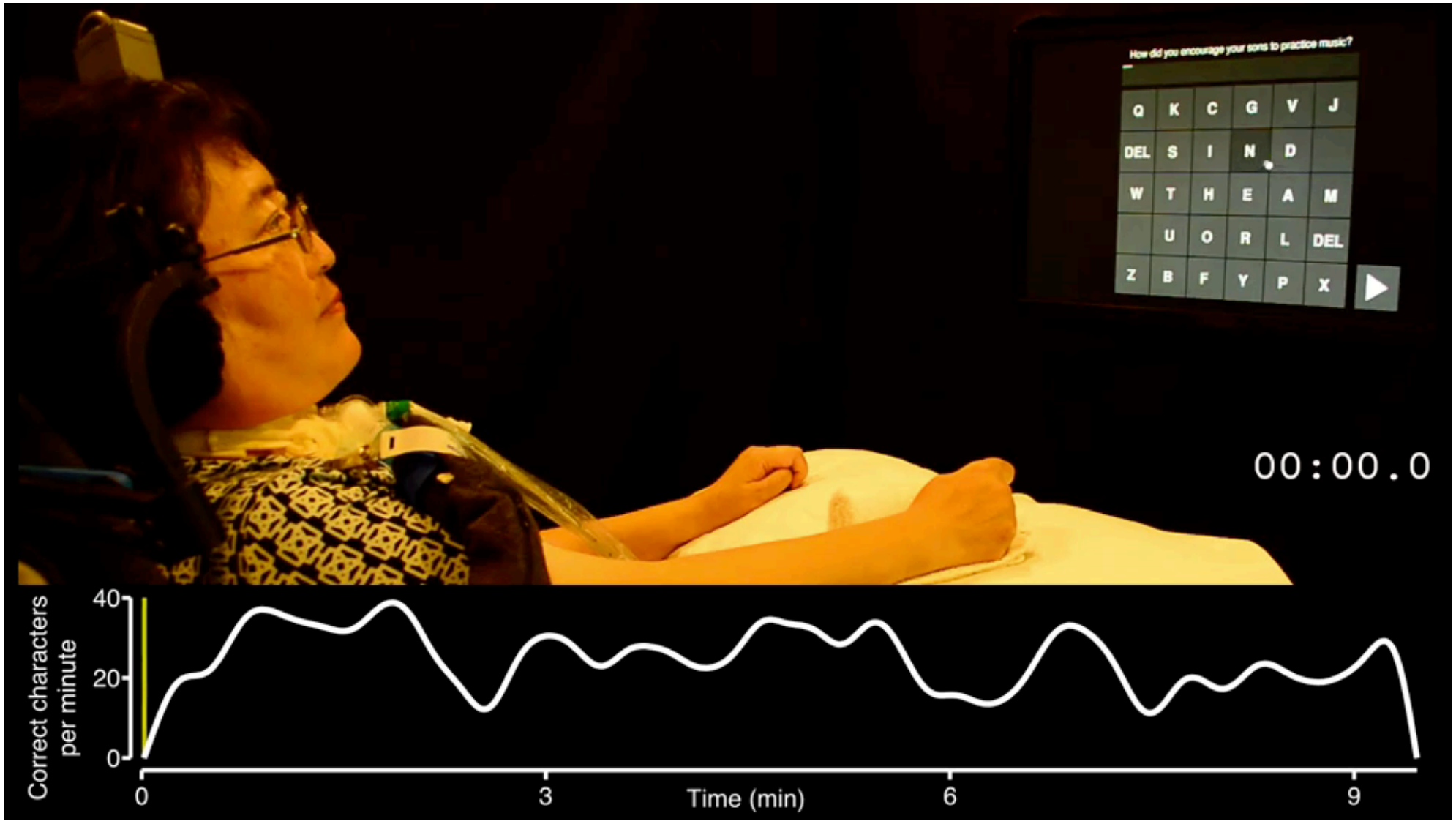
Algorithm: ReFIT-KF + HMM

(Kao, Nuyujukian*, et al., IEEE TBME 2016)*

Pandarinath, Nuyujukian*, et al., eLife 2017*

BrainGate2 Pilot Clinical Trial at Stanford University
Caution: Investigational Device.
Limited by Federal Law to Investigational Use.

Why a class on deep learning?



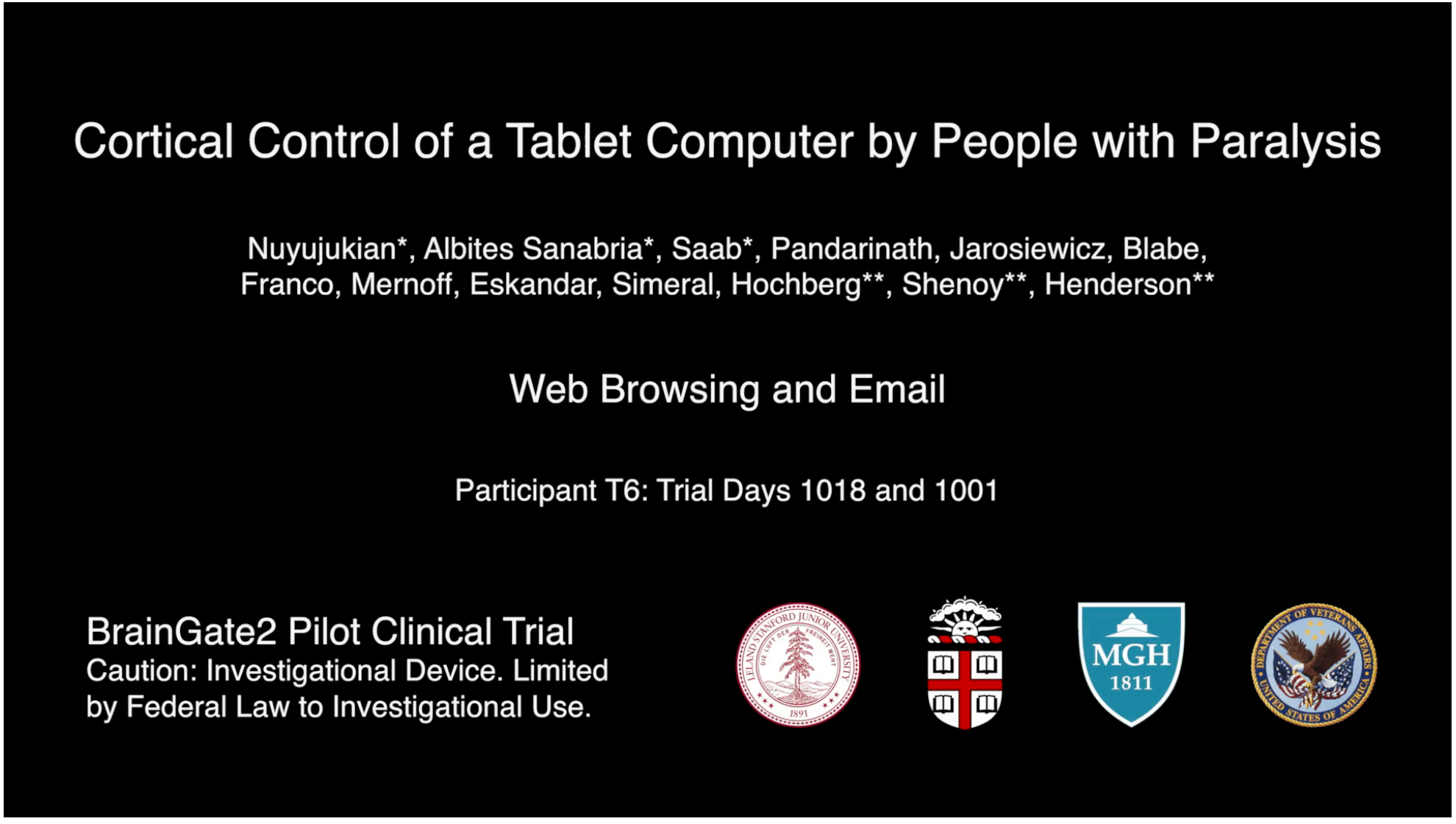
Algorithm: ReFIT-KF + HMM

(Kao, Nuyujukian*, et al., IEEE TBME 2016)*

Pandarinath, Nuyujukian*, et al., eLife 2017*

BrainGate2 Pilot Clinical Trial at Stanford University
Caution: Investigational Device.
Limited by Federal Law to Investigational Use.

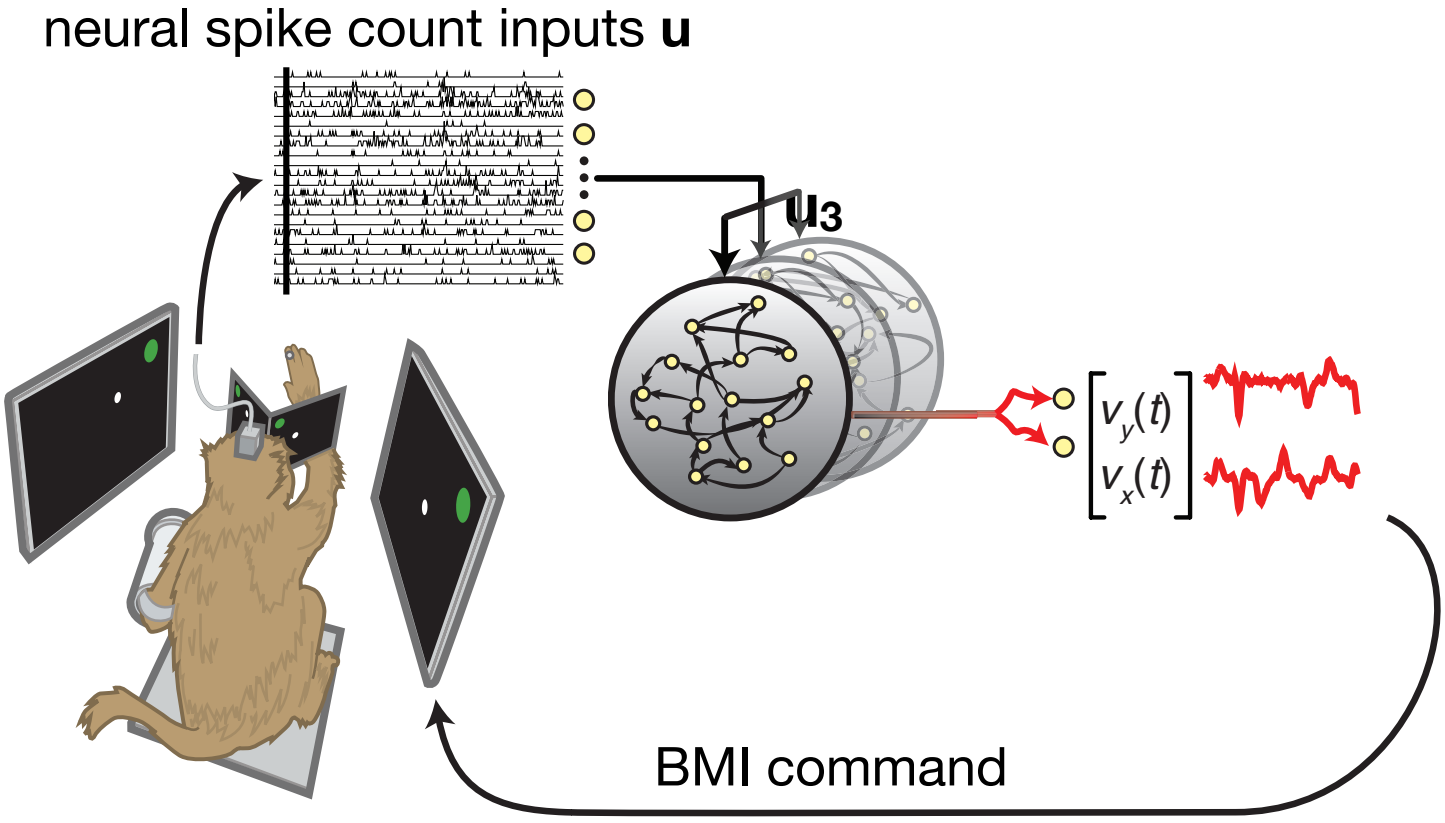
Why a class on deep learning?



Nuyujukian et al., PLoS ONE, 2018

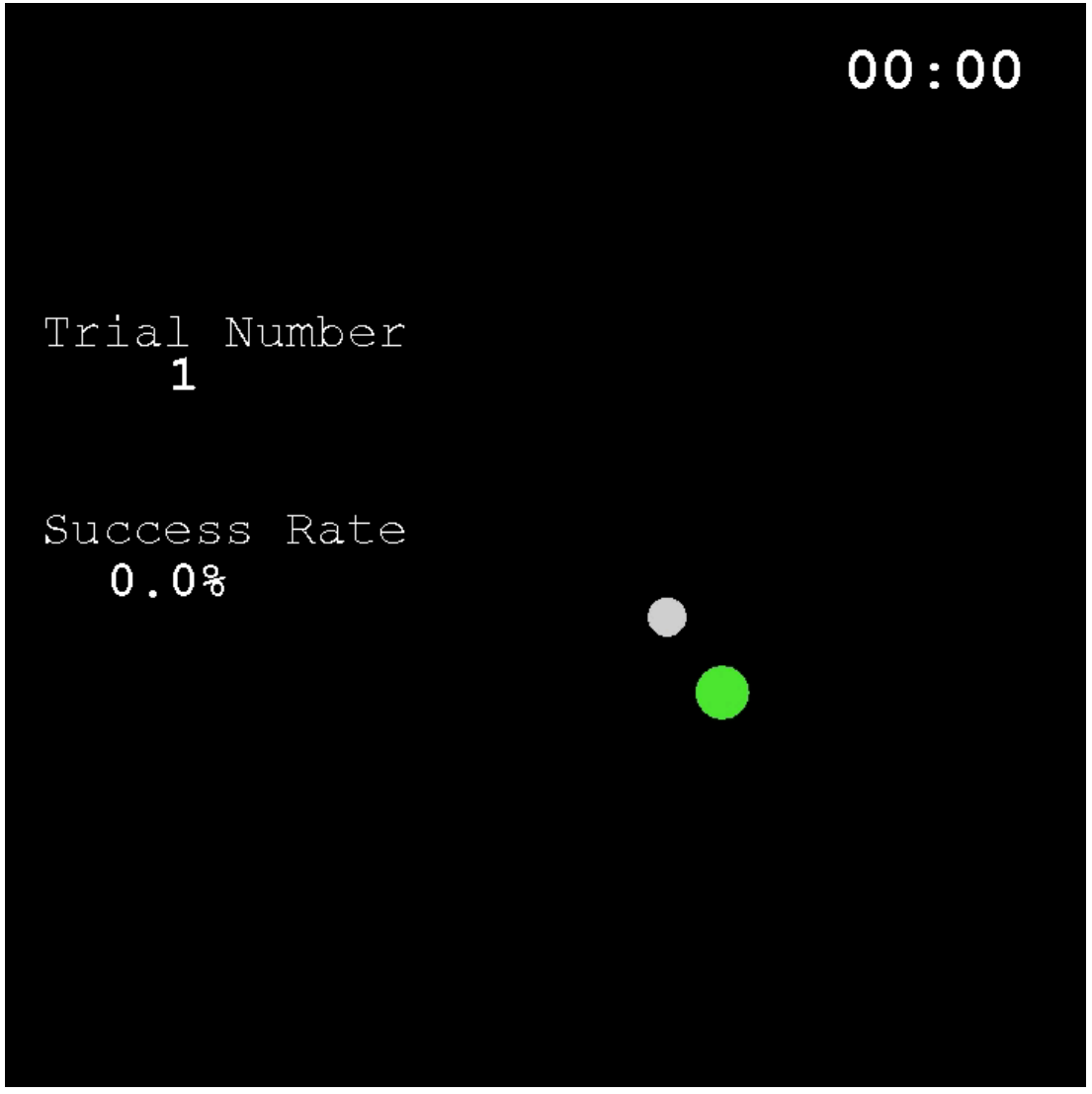
BrainGate2 Pilot Clinical Trial at Stanford University
Caution: Investigational Device.
Limited by Federal Law to Investigational Use.

Why a class on deep learning?



Sussillo*, Stavisky*, Kao*, Ryu, Shenoy, Nat Comm 2016

Why a class on deep learning?



Sussillo*, Stavisky*, Kao* (co-first author), Ryu, Shenoy, Nat Comm 2016

ECE C143A/C243A:
(Spring quarter 2021)

Deep Learning is becoming ubiquitous

- We see that deep learning is enabling AI's to become an everyday part of life.
- It has resulted in key breakthroughs in many areas; in many different fields, many groups are looking towards deep learning to achieve better performance.
- Deep learning may be useful to research in your area / future work.

We will study deep learning in the context of computer vision

- Neural network research dates back to 1943 (McCullough and Pitts, a logical calculus of nervous activity) and 1958 (Rosenblatt's perceptron)

- Example video from 1989 of using a multi-layer perceptron (fully connected neural network) to do classification of digits.

- Fukushima's NeoCognitron (1982), Yann LeCun's "LeNet" (1998) which is the modern convolutional architecture, existed well before today's deep learning renaissance.

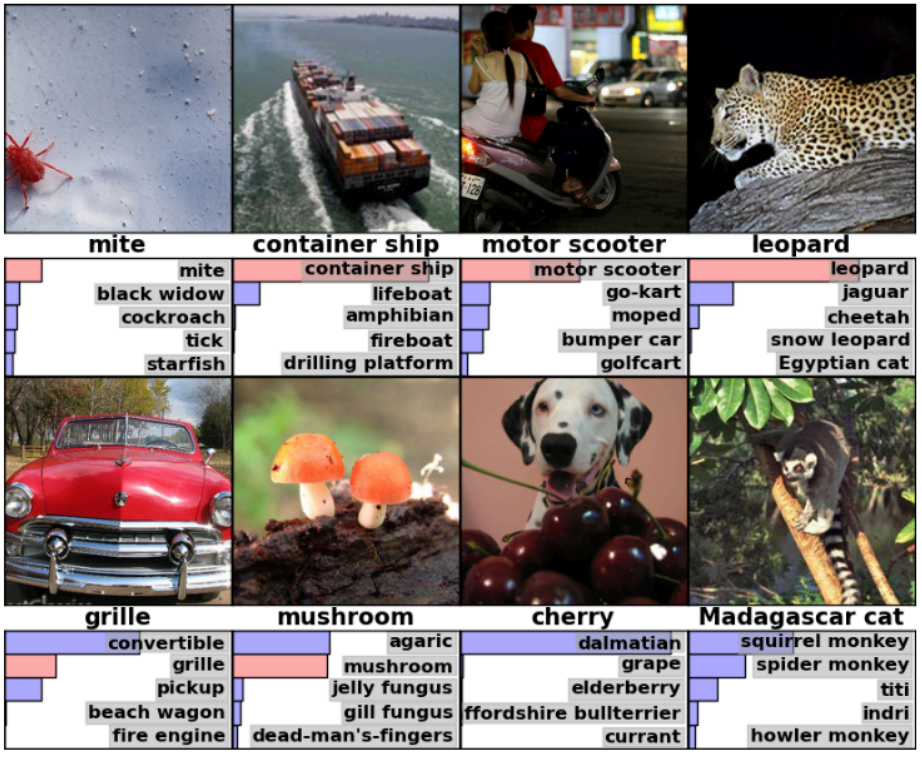
Tackling computer vision

- IMAGENET

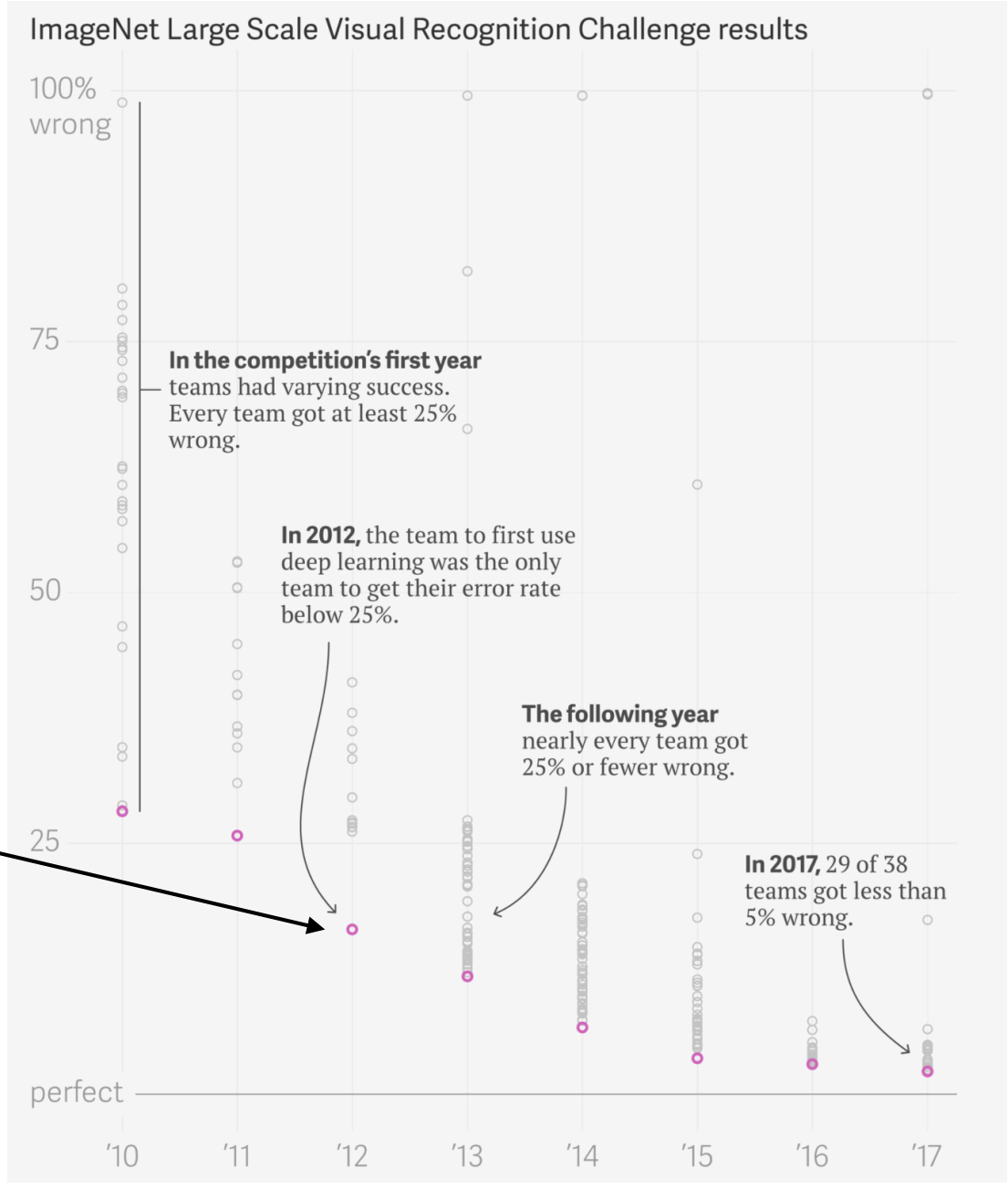
Overall

- Total number of non-empty synsets: 21841
- Total number of images: 14,197,122
- Number of images with bounding box annotations: 1,034,908

http://image-net.org/about-stats



A big splash



AlexNet

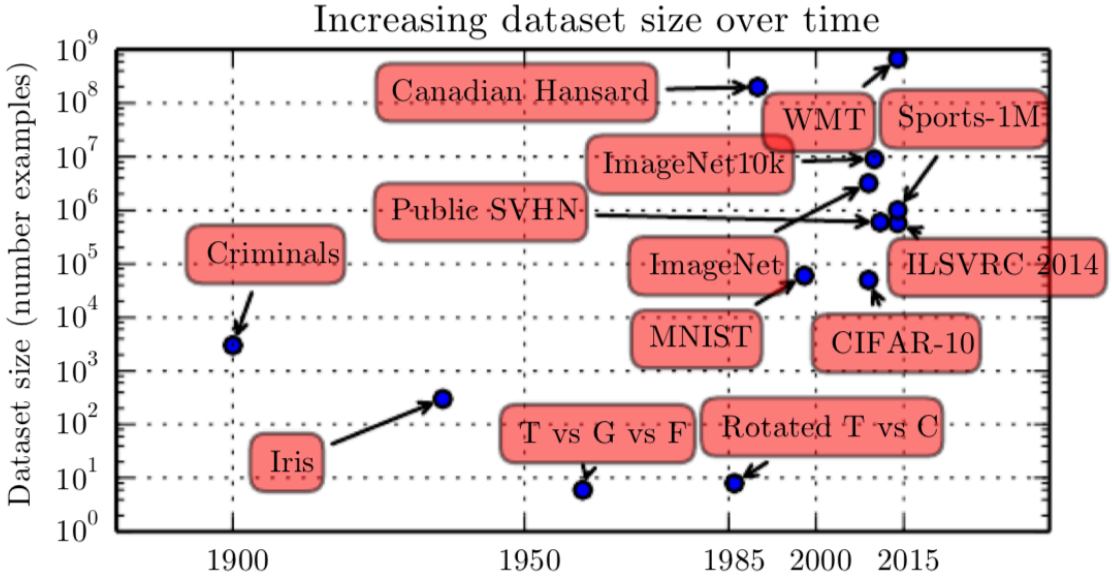
Data and technology

- Part of what has driven deep learning today is the amount of data we have and the computational power we have to process it.
- This includes larger models as computing infrastructure improves.

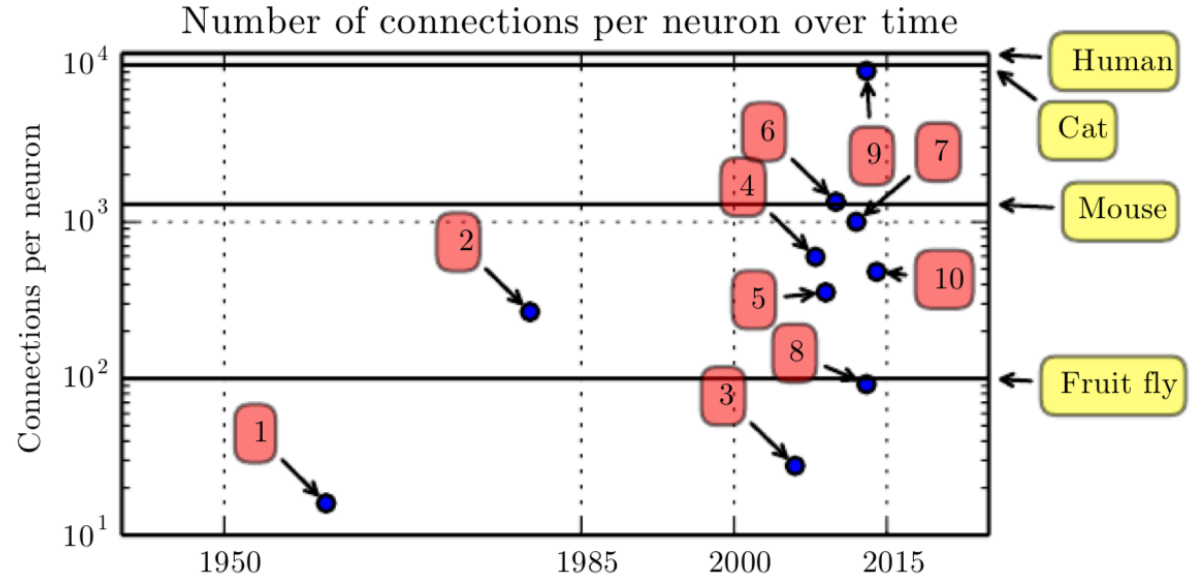
Fortunately, the amount of skill required [to get good performance from a deep learning algorithm] reduces as the amount of training data increases.
(Goodfellow, p. 18)

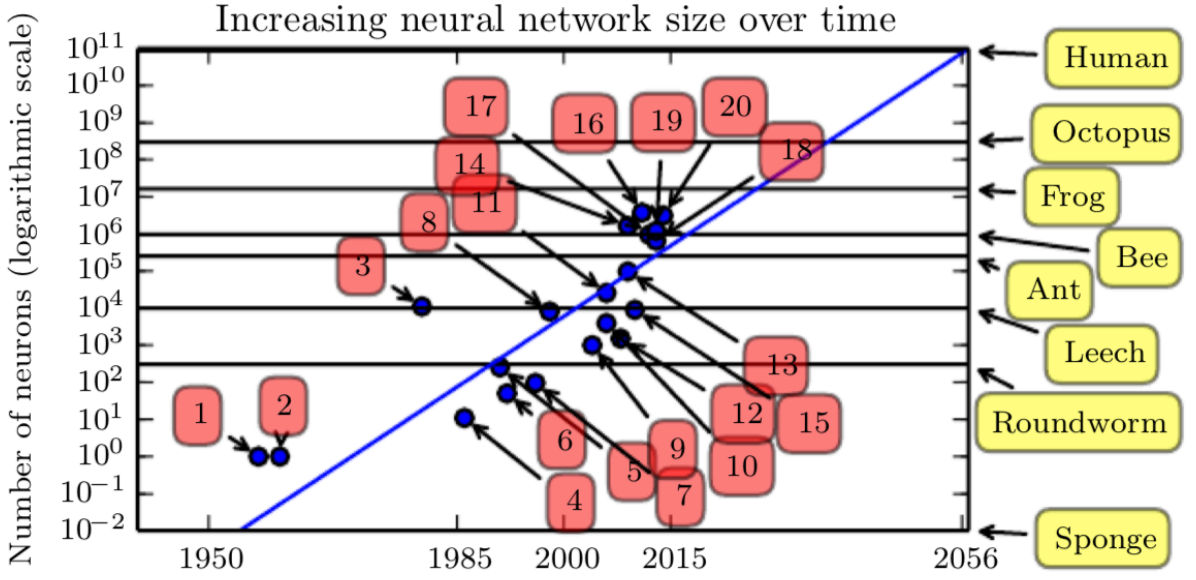
Larger dataset sizes

Fortunately, the amount of skill required [to get good performance from a deep learning algorithm] reduces as the amount of training data increases.
(Goodfellow, p. 18)



Larger networks



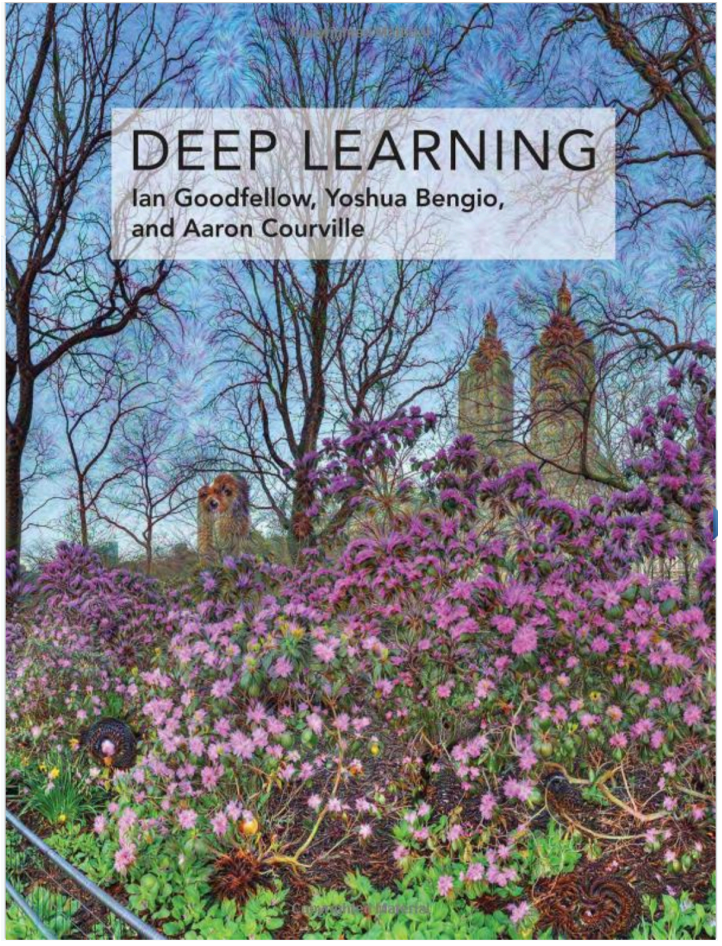


Breaks.

- I plan to give a 5 minute break in the middle of lecture.
- I will show a timer up here on the screen.

Rough class schedule

Date (2022)	Lecture	Content
03 Jan	1	Overview to deep learning (ZOOM)
05 Jan	2	Machine learning refresher I (ZOOM)
10 Jan	3	Supervised Classification & Gradient descent principles (ZOOM)
		HW #1 released
12 Jan	4	Fully connected neural networks (ZOOM)
17 Jan	-	No class (MLK holiday)
		HW #2 released
19 Jan	5	Backpropagation
24 Jan	6	Regularizations for training neural networks
		HW #3 released
26 Jan	7	Optimization for training neural networks
31 Jan	8	Convolutional Neural Networks I (ZOOM)
		HW #4 released
02 Feb	9	Convolutional Neural Networks II (ZOOM)
07 Feb	10	Convolutional Networks III
		HW #5 released
09 Feb	11	Recurrent Neural Networks
14 Feb	12	Recurrent neural networks II
16 Feb	M	Midterm, in class
		Project released
21 Feb	-	Veteran's day holiday
23 Feb	14	PyTorch and Tensorflow
28 Feb	15	Variational autoencoders
02 Mar	16	Generative adversarial networks
07 Mar	17	Survey of advanced topics in deep learning
09 Mar	18	Overview



http://www.deeplearningbook.org/

Class logistics

- The class will be done entirely in Python and we will provide resources for setting up Python 3, as well as packages needed to be installed for each assignment.
- We will post a set of “formal notes” on Bruin Learn.
- Lecture notes “in class” are informal and not publicly posted. I ask that you please not publicly post them, though we will distribute them on Bruin Learn.

On discussion sections

- We have 10 total discussion sections, but in our experience, the discussion sessions are not robustly attended, especially for graduate classes
- We will therefore upload a discussion video each week that goes over material related to the homework.
- We will hold **some** of the 10 discussion sections as live discussion sections based on a poll we send out to the class.
- We will take the remaining time we would have spent holding discussions and instead hold extra OH. We will post a full office hours schedule by end of week after completion of our polls.

Grading

You will be graded on three components:

1. **Homework (40%)**. Homework will contain both written components as well as `Python` components.
   - Assignments are due (i.e., submitted to Gradescope) by 11:59pm on the day they are due. PRINT JUPYTER NOTEBOOK TO PDF.
   - To accomodate for unexpected or unforeseen circumstances, we will give *three late days* to every student. These late days should only be used in extenuating circumstances. We will not grant additional late days beyond these.
   - You may use **at most** 2 late days on any given assignment.
   - Any assignment more than two days late receives a grade of **zero**.
2. **Midterm exam (30%)**, in class.
3. **Final project (30%)**, details to be released.

Grading (cont.)

The class is graded on an absolute scale. The scale may be relaxed but it will not be made more stringent.

Final score	Letter grade
93 – 100	A
90 – 93	A-
86 – 90	B+
83 – 86	B
80 – 83	B-
76 – 80	C+
73 – 76	C
70 – 73	C-
66 – 70	D+
63 – 66	D
60 – 63	D-
< 60	F

- We award A+ to the class's top students.
- I will not change your final grade unless I made a calculation error, in accordance with UCLA Academic Senate Regulation A-313 and strict rules governing the integrity of the grading process.

Grading (cont.)

In addition to these grading scales, we will also award bonuses on top of your final grade as follows.

Bonuses (2 points)

- (Feedback) You earn a bonus of +0.5% for filling out the class evaluation at the end of class.
- (Piazza) You receive a bonus of at most +1.5% for participating on Piazza. While your answers to others will be anonymous, they will be known to the instructors, who will determine an appropriate number of points for instructor-approved student replies. Your bonus will be based on your participation on Piazza, which will be curved.
- (Piazza, cont.) Please do not conspire to post and answer questions for extra credit. We will be able to detect this. We do not want the Piazza forums to be spammed; this makes it more difficult for all students to find helpful questions.

Some notes on the class

- Piazza should be the primary means of asking and getting questions answered in the class. We would like Piazza to be student-driven. TAs will answer any questions that other students are unable to answer.
- If you have other course-related questions that are not appropriate to Piazza, please e-mail the entire teaching staff (rather than any one of us individually). If you have any personal matters to discuss, please e-mail Prof. Kao directly.

Some notes on the class

- 5 HW assignments.
- Assignments will contain both written components as well as coding components.
- Code in the HW will walk you through Jupyter Notebooks, and usually it'll be clear if you're doing things correctly or not.
    - Note, we will release solutions for written HW exams, but we will **not** release solution code for HW's.
    - These assignments are planned to be used in future years.
- HW will be submitted via Gradescope; they are due at 11:59PM Pacific Time on the stated due date.
- Any late assignment will receive a grade of **zero**.
- We understand that life happens. We are giving **three late days to every student**, with the intent that these are to be used only in extenuating circumstances. You may use at most 2 late days on any given assignment.

Some notes on the class

- C147 and C247 students will be graded on different scales.
    - C147 students will also have one question of one homework be optional.
    - The classes are otherwise the same.
- All HWs are to be submitted to Gradescope.
- In general, I will post the non-annotated versions of the slide prior to lecture, so you can take notes if you want. I will always post the annotated version of the slides as well as the lecture videos (unless there are technical failures).
- We will list readings in the textbook at the start of each different lecture topic.
- This class has 4 total TAs. Two TAs are regular TAs, and two TAs are for the online version of the class. These 4 TAs will service the whole class, and you are welcome to attend any of their office hours. Note that by policy, TAs in the online version of the class will prioritize questions from MS Online students first.

Some notes on the class

- This class will focus on implementation of neural networks and on algorithms to train them.
- It will **not** cover theory of deep learning (CS 269).
- It will **not** be a theoretical class in general.
    - We aim to be rigorous in our use of notation and math.
    - But we won't be doing proofs.
    - We're more focused on application and equipping you with tools that may be helpful in your research or future industry positions.
- This is not a statement on the importance of theory. It merely reflects the priorities for this class alone.

Academic integrity

UCLA embraces the core values of integrity, excellence, accountability, respect, and service through the True Bruin program

`http://www.truebruin.ucla.edu`

I take academic integrity very seriously; students caught cheating or violating these principles will face disciplinary action. Please refer to the UCLA student conduct code:

`https://deanofstudents.ucla.edu/student-conduct-code`

In this class, unacceptable behavior includes plagiarizing the work of others, plagiarizing code, and copying another person's exam. In accordance with UCLA policy, any instance of suspected academic dishonesty will be immediately reported to the Dean of Students Office and zero credit will be given for any work determined to be dishonest.

What I assume you know

Pre-requisites

This class requires a solid understanding in probability (131A) and linear algebra (133A or 205A), as well as prior exposure to machine learning (M146). This class will be very difficult (but not impossible) if you do not have prior machine learning background. We will spend two lectures doing machine learning review to ensure we are familiar with concepts we will expand upon in machine learning.

It also requires coding experience. The class will be taught entirely in `Python`. If you have only had exposure to `MATLAB`, there will be ramp up time to familiarize yourself with `Python`. You should factor this into your course load.

Pre-requisite topics I will **assume** you know.

- Probability: independence, conditional probability, Bayes rule, multivariate Gaussian distribution, marginalization, expectation, variance
- Linear algebra: basic matrix operations, span, rank, range and null space, eigenvalue decomposition, singular value decomposition, pseudoinverse

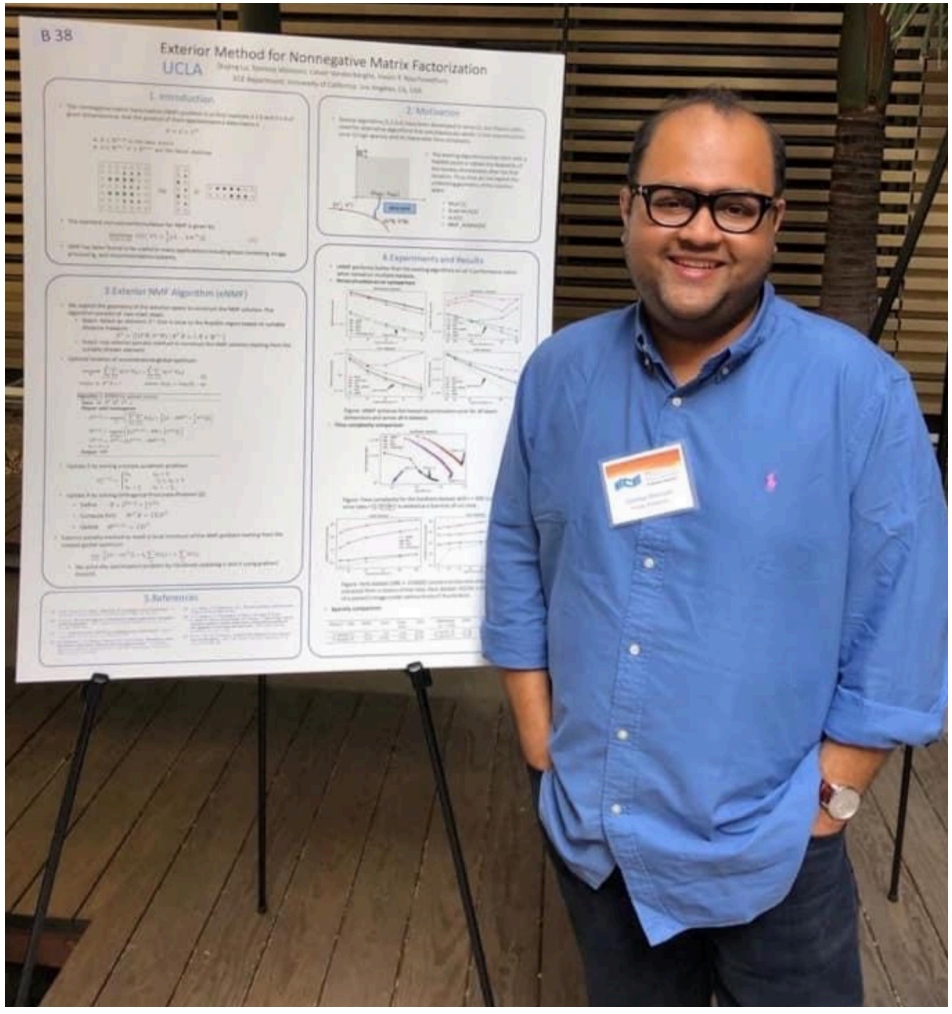
A few last notes about this class

- Common student feedback is that, even if they were familiar with `MATLAB`, it was still time-consuming to transition to `Python`. Please consider this seriously as you plan your schedule for assignments. `Python` is the standard language for machine learning research today, and the best deep learning packages are specifically designed for `Python`.
- We know, and consistently receive feedback, that this class is a lot of work and is time-consuming. I want to state this up front so you can plan accordingly. We will aim to keep the stated HW schedule, following the assignments and schedule used last year.

About me

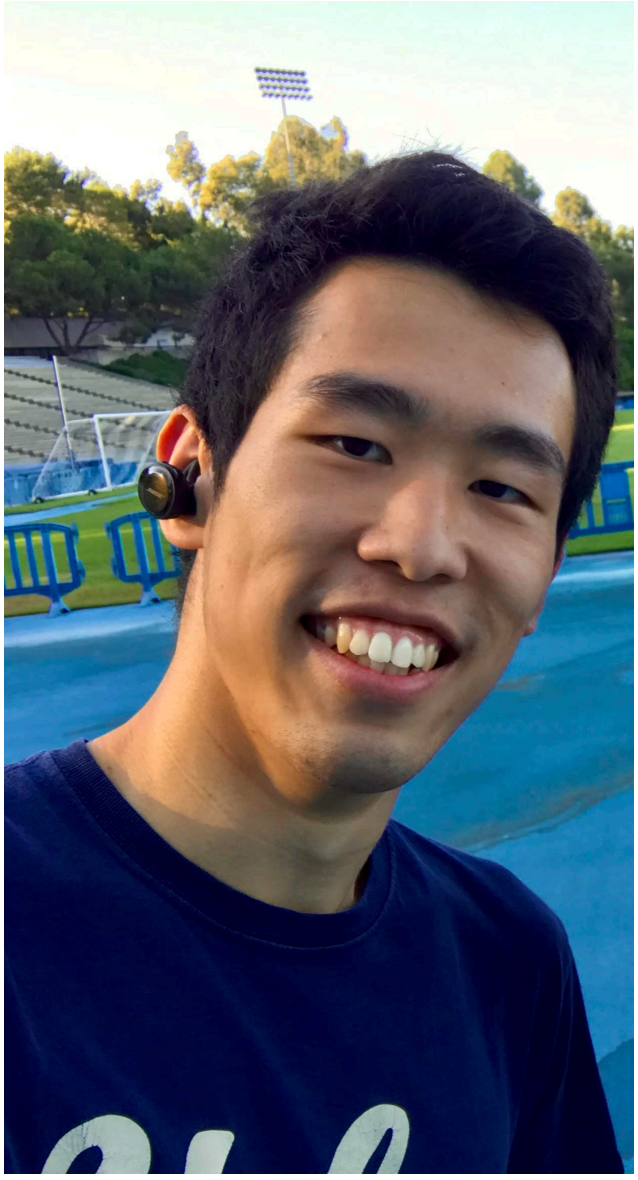
- This is my fifth time teaching this class at UCLA.
- Though this class focuses on deep learning for engineering purposes, I'm interested in deep learning because of its ties to the brain.
    - I am primarily a computational neuroscientist and neural engineer.
    - My work includes brain-machine interfaces and applying machine learning to understand how neurons in the brain communicate.
    - We use deep learning as a model to understand the brain.

Tonmoy Monsoor



- Graduate student researcher at Big data and Complex Networks Group
- Fourth year Ph.D. student advised by Professor Vwani Roychowdhury
- Research interests
  - Reinforcement learning
  - Distributed optimization
  - Pattern extraction in dynamic networks
- Favorite classes at UCLA
  - Convex optimization, ECE 236B
  - Neural signal processing, ECE 243A

Tianyi Wang

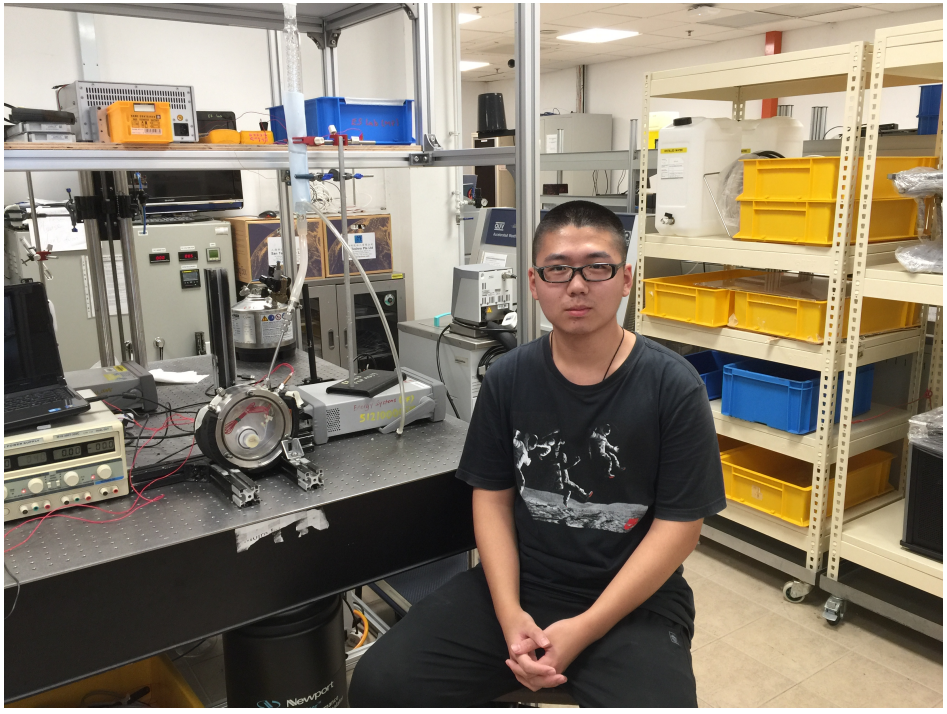


- Fourth year PhD student advised by Prof. Vwani Roychowdhury in Department of Electrical and Computer Engineering
- Research Interests:
  - Physics informed deep learning
  - Cognition-inspired natural language understanding and multi-modal learning
  - Stochastic optimization

Pan Lu

- 3rd-year Ph.D. candidate in Computer Science advised by Prof. Song-Chun Zhu
- Graduate student researcher at VCLA Team
- Research Interests
  - Commonsense reasoning
  - Mathematical reasoning for education
  - Multimodal learning in vision and language
- Second time being a TA, and I'd like to help you!

Yang Li



- 2nd year PhD student in Mechanical Engineering department
- Graduate student researcher in H-Lab
- Research interests:
  - Material science
  - Density Functional Theory for thermal property calculation
  - Neural Network Potential for thermal property prediction
- Second time as a TA

Questions?

Lecture 2: Machine learning refresher

This lecture gives a refresher on key concepts from machine learning.

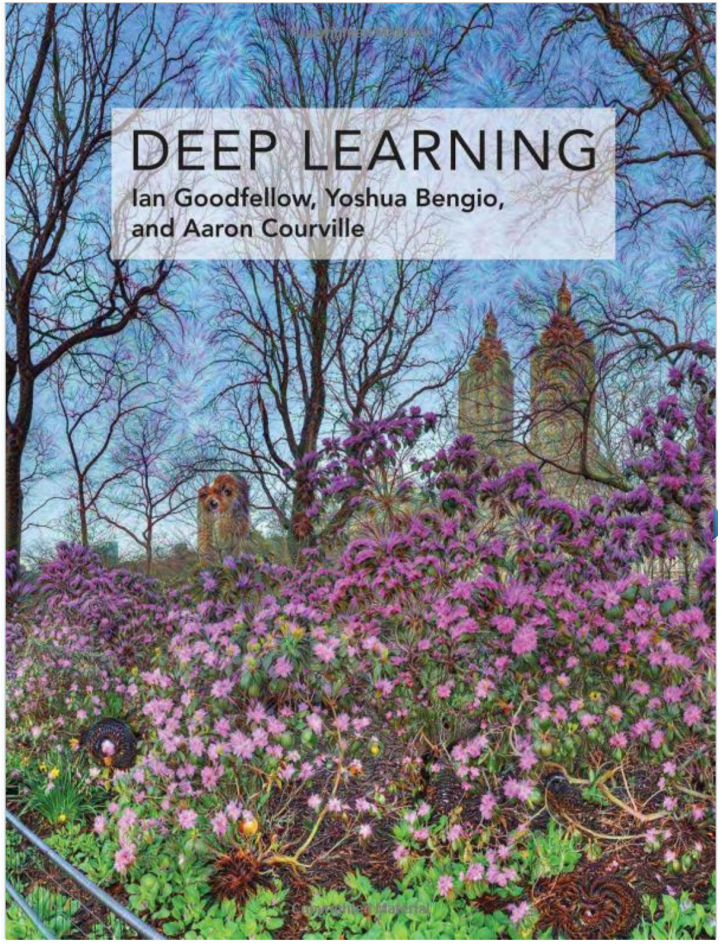
- Introduction to concepts in machine learning
- Cost functions
- Example: linear and polynomial regression
- Model complexity and overfitting
- Training set, validation set, test set
- Dealing with probabilistic cost functions and models
- Example: maximum-likelihood classification

Lecture 2: Basics of machine learning

Reading:

Deep Learning, chapter 5 (up to and including section 5.5).

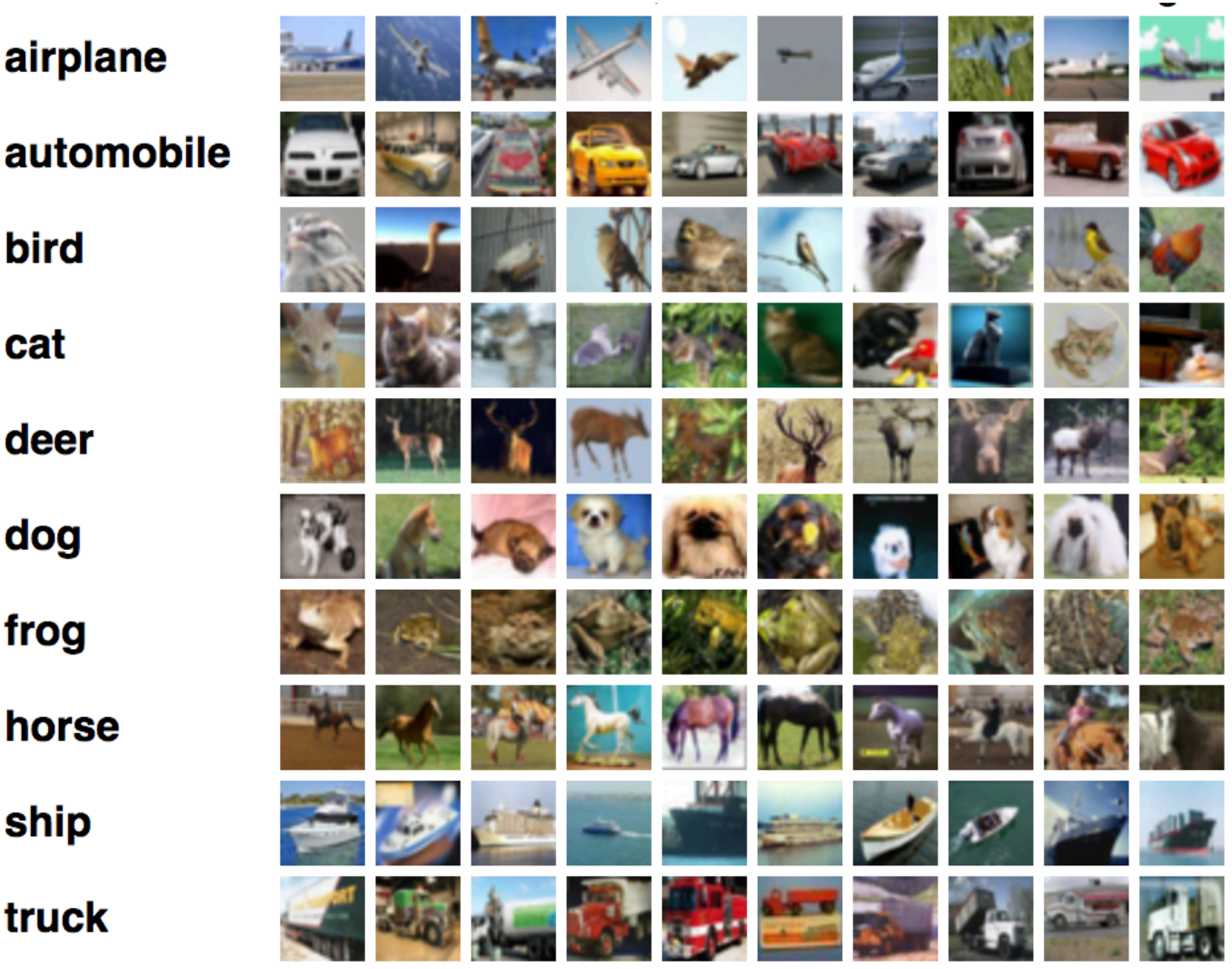
To refresh your linear algebra and probability, look at chapters 2 and 3 respectively.



Focus of this class:

- This class will focus on **supervised** learning problems.
- This class will also focus on **classification** largely (e.g., when considering convolutional neural networks) as well as some instances of **regression** (e.g., when considering recurrent neural networks)

Classification



CIFAR-10 dataset, https://www.cs.toronto.edu/~kriz/cifar.html

↳ 10 CLASSES

INPUT: $\vec{x} \rightarrow 32 \times 32 \times 3$ TENSORS

↳ RGB

$\Big\Vert$ RESHAPE

$\downarrow$

$\vec{x} \in \mathbb{R}^{3072}$

CLASSIFICATION:

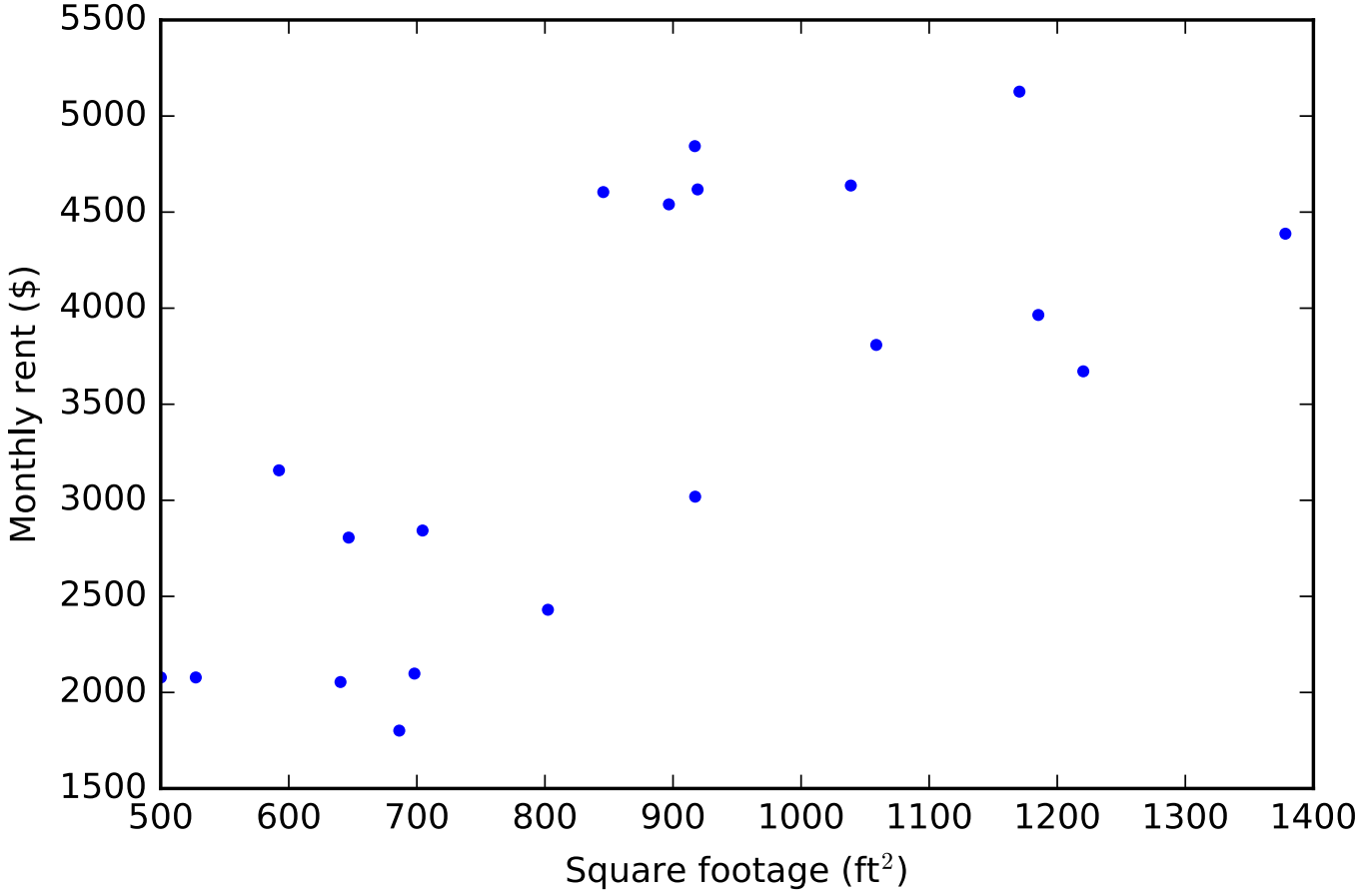
$f(\vec{x}) \longrightarrow$ CLASS (OUTPUT)

OUTPUT: What class the image belongs to?

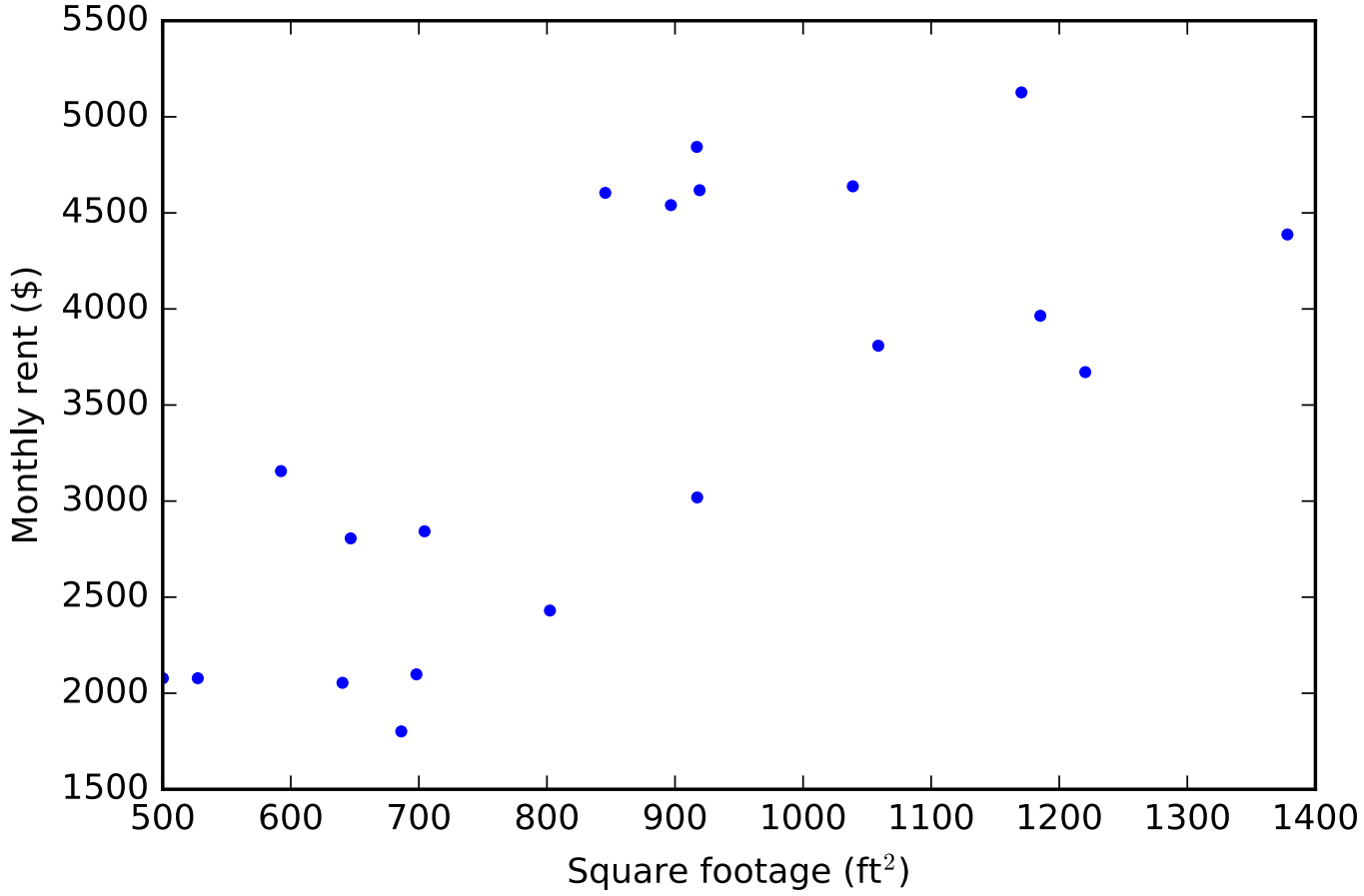
GOAL: FIND WHAT CAN BE USED AS "f"!

An example of supervised learning

Let's say we want to rent a home in Westwood, and we wanted to know if we were getting a good deal. **(Warning: this data is synthetic! Scrape the real data if you're curious.)**



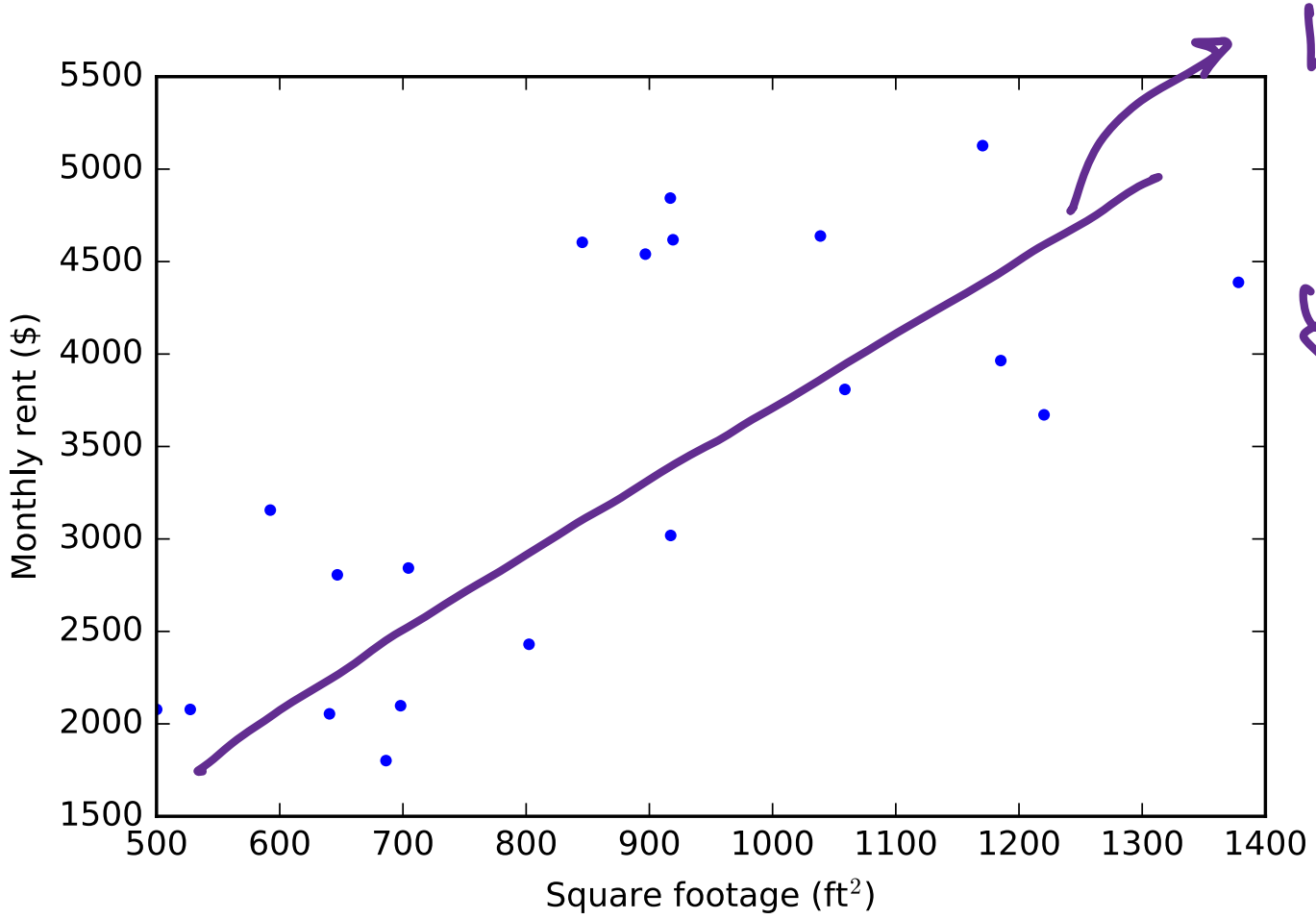
An example of supervised learning



How should we model this data?

- Inputs, **x**? Outputs, **y**?
- What model should we use?
- How do we assess how good our model is?

An example of supervised learning



LINEAR MODEL

$y = ax + b$

PARAMETERS

How should we model this data?

- Inputs, **x**? Outputs, **y**?
- What model should we use? → LINEAR
- How do we assess how good our model is?

- Inputs, **x**? Outputs, **y**?

INPUT: Sq. ft.
(x)

OUTPUT: RENT
(y)

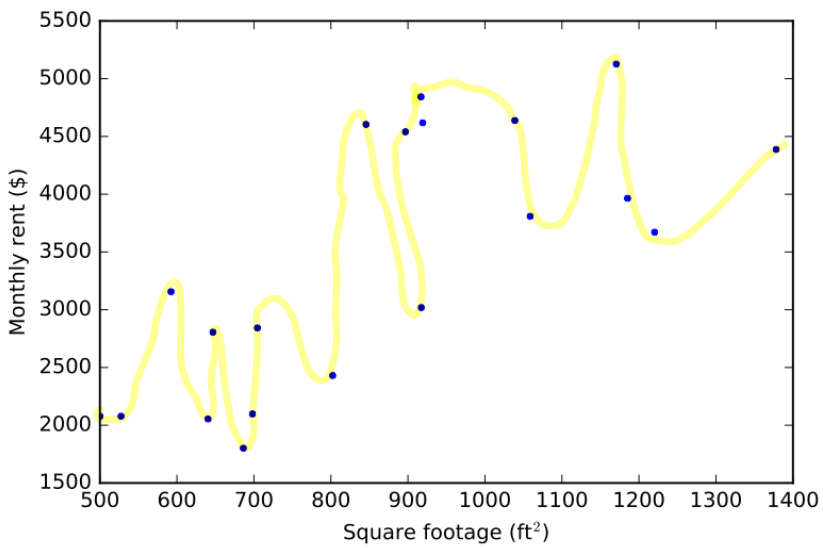
GOAL: LEARN f

RENT ← f(sq. ft.)

- What model should we use?

MODEL: → LINEAR: $y = ax + b$

→ POLYNOMIAL: $y = b + a_{100}x^{100} + a_{99}x^{99} + \dots + a_1 x$



THIS MAY NOT BE GOOD!

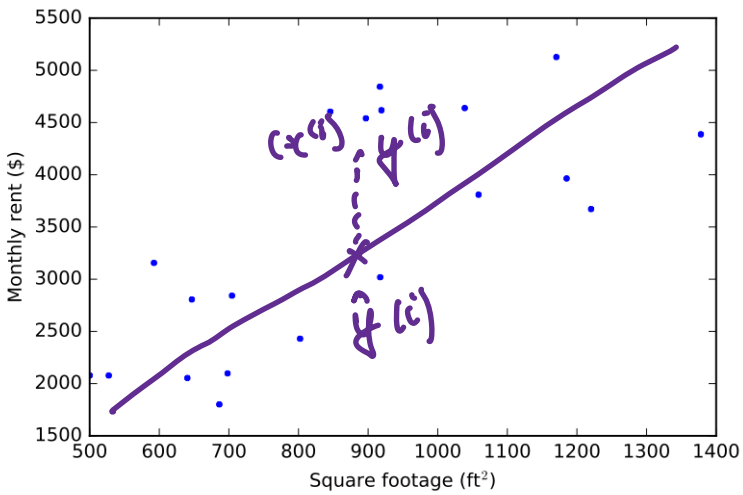
- How do we assess how good our model is?

INPUT : $x^{(i)}$

OUTPUT : $y^{(i)}$

$\hat{y}^{(i)} = f(x^{(i)})$

↳ MODEL PREDICTION



$y = b + ax$

$\varepsilon_i = (y^{(i)} - \hat{y}^{(i)})^2$

$\text{LOSS} = \text{COST} = \frac{1}{N} \sum_{i}^{N} (y^{(i)} - f(x^{(i)}))^2$

GOAL: FIND a, b ∀ LOSS IS MINIMUM //

An example of supervised learning

A simple linear example:

$\theta = \begin{bmatrix} a \\ b \end{bmatrix} \in \mathbb{R}^2$

- Model:

$$\begin{aligned} \hat{y} &= ax + b \\ &= \theta^T \hat{\mathbf{x}} \end{aligned}$$

$\hat{x} : \begin{bmatrix} x \\ 1 \end{bmatrix}$

- Cost function:

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{2} \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2 \\ &= \frac{1}{2} \sum_{i=1}^{N} (y^{(i)} - \theta^T \hat{\mathbf{x}}^{(i)})^2 \end{aligned}$$

↳ CHOOSE θ TO MAKE MODEL FIT DATA

⇓

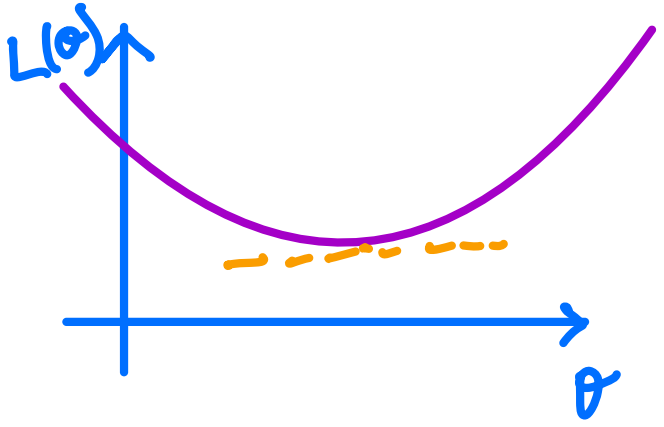
TO MAKE $L(\theta)$ AS SMALL AS POSSIBLE

How do we learn the parameters of this model?

An example of supervised learning

Construct it as an optimization problem.

Our goal is to choose the parameters to make the loss as small as possible.



Thus our strategy to find the best θ is to:

- Calculate

$$\frac{d\mathcal{L}}{d\theta}$$

- Solve for θ such that

$$\frac{\partial \mathcal{L}}{\partial \theta} = 0$$

$L(\theta) \rightarrow$ CONVEX

① GLOBAL MIN

② $\partial^2 L / \partial \theta^2 > 0$

However, θ is a vector, so how do we take derivatives with respect to it?

Aside: vector and matrix derivatives

In machine learning, we often take derivatives with respect to vectors and matrices.

These are typically called *gradients*, and in this class we'll use the following notation to denote derivatives with respect to vectors and matrices.

In this class, we will use both the differentiation operator ∂ and ∇ to denote derivatives. If y is a scalar, and $\mathbf{x}$ is a vector, then the gradient of y with respect to $\mathbf{x}$ is denoted as both:

$$\frac{\partial y}{\partial \mathbf{x}} \quad \text{and} \quad \nabla_{\mathbf{x}} y$$

The gradient of y with respect to $\mathbf{x}$ is itself a vector with the same dimensionality as $\mathbf{x}$.

Aside: vector and matrix derivatives

The gradient

The gradient generalizes the scalar derivative to multiple dimensions. Suppose $f : \mathbb{R}^n \to \mathbb{R}$ transforms a vector $\mathbf{x} \in \mathbb{R}^n$ to a scalar. If $y = f(\mathbf{x})$, then the gradient is:

$$\nabla_{\mathbf{x}} y = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix}$$

$\in \mathbb{R}$ $\in \mathbb{R}^n$

Δy due to Δx_2 is $\approx \partial y / \partial x_2$

if $dy = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ so changing x_2, y doesn't change

Aside: vector and matrix derivatives

In other words, the gradient is:

- A *vector* that is the same size as $\mathbf{x}$, i.e., if $\mathbf{x} \in \mathbb{R}^n$ then $\nabla_{\mathbf{x}} y \in \mathbb{R}^n$.
- Each dimension of $\nabla_{\mathbf{x}} y$ tells us how small changes in $\mathbf{x}$ in that dimension affect y. i.e., changing the ith dimension of $\mathbf{x}$ by a small amount, Δx_i, will change y by

$$\frac{\partial y}{\partial x_i} \Delta x_i$$

We may also denote this as:

$$(\nabla_{\mathbf{x}} y)_i \, \Delta x_i$$

$\vec{x}$ becomes $\vec{x} + \Delta\vec{x}$

How much does y change

$$\Delta y \approx (\Delta_x y)^T \Delta \vec{x} \approx \sum_{i=1}^{n} \frac{\partial y}{\partial x_i} \Delta x_i$$

Aside: vector and matrix derivatives

Example: derivative with respect to a vector

Let $f(\mathbf{x}) = \theta^T \mathbf{x}$. What is $\nabla_{\mathbf{x}} f(\mathbf{x})$?

$\theta, \vec{x} \in \mathbb{R}^n$

$y = f(x) = \theta^T x$

$\nabla_x y \in \mathbb{R}^n$

$f(x) = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

$$\nabla_x f(x) = \begin{bmatrix} \partial f(x)/\partial x_1 \\ \partial f(x)/\partial x_2 \\ \vdots \\ \partial f(x)/\partial x_n \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} = \theta$$

Aside: vector and matrix derivatives

Example: derivative with respect to a vector

Let $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A}\mathbf{x}$. What is $\nabla_{\mathbf{x}} f(\mathbf{x})$?

$\vec{x} \in \mathbb{R}^n \quad A \in \mathbb{R}^{n \times n}$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$

$$f(x) = x^T A x = \sum_i \sum_j a_{ij} x_i x_j$$

$$\frac{\partial f(x)}{\partial x_1} = 2a_{11}x_1 + \sum_{j=2}^{n} a_{1j}x_j + \sum_{i=2}^{n} a_{i1}x_i$$

$$= \sum_{j=1}^{n} a_{1j}x_j + \sum_{i=1}^{n} a_{i1}x_i .$$

Aside: vector and matrix derivatives

CASE 1: $i = j = 1$ $\quad a_{11} x_1^2$

$$\frac{\partial a_{11} x_1^2}{\partial x_1} = 2 a_{11} x_1$$

CASE 2: $i = 1, j \neq 1$ $\quad \sum_{j=2}^{n} a_{ij} x_1 x_j$

$$\frac{\partial \sum_{j=2}^{n} a_{ij} x_1 x_j}{\partial x_i} = \sum_{j=2}^{n} a_{ij} x_j$$

CASE 3: $i \neq 1$ $\quad j = 1$

$$\sum_{i=2}^{n} a_{i1} x_i x_1$$

$$\frac{\partial f(x)}{\partial x_1} = \sum_{j=1}^{n} a_{1j} x_j + \sum_{i=1}^{n} a_{i1} x_i$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

A $\qquad$ x

$$(Ax)_1 \qquad + \qquad (A^T x)_1$$

$$\frac{\partial f(x)}{\partial x_1} = (Ax)_1 + (A^T x)_1$$

$$\frac{\partial f(x)}{\partial x_2} = (Ax)_2 + (A^T x)_2$$

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

$$\begin{bmatrix} \partial f(x)/\partial x_1 \\ \partial f(x)/\partial x_2 \\ \vdots \\ \partial f(x)/\partial x_n \end{bmatrix} = \begin{bmatrix} (Ax)_1 + (A^T x)_1 \\ (Ax)_2 + (A^T x)_2 \\ \vdots \\ (Ax)_n + (A^T x)_n \end{bmatrix}$$

Aside: vector and matrix derivatives

First, we note that $\mathbf{x}^T\mathbf{A}\mathbf{x} = \sum_i \sum_j a_{ij} x_i x_j$. Then, we have

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} 2a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + a_{21}x_2 + \cdots + a_{n1}x_n \\ 2a_{22}x_2 + a_{21}x_1 + \cdots + a_{2n}x_n + a_{12}x_1 + \cdots + a_{n2}x_n \\ \vdots \\ 2a_{nn}x_n + a_{n1}x_1 + \cdots + a_{n,n-1}x_{n-1} + a_{1n}x_1 + \cdots + a_{n-1,n}x_{n-1} \end{bmatrix}$$

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

IF A is symmetric

$$\nabla_x f(x) = 2Ax.$$

SANITY CHECK:

Assume scalars ($n = 1$)

$$f(x) = ax^2 \qquad \frac{df(x)}{dx} = 2ax.$$

Matrix derivatives

$y = f(x)$ eg. $z^T A x$ $x \in \mathbb{R}^n$

$z \in \mathbb{R}^m$

$A \in \mathbb{R}^{m \times n}$

$$\nabla_A y = \begin{bmatrix} \partial y/\partial a_{11} & \partial y/\partial a_{12} & \cdots & \partial y/\partial a_{1n} \\ \vdots & \ddots & & \vdots \\ \partial y/\partial a_{m1} & \cdots & & \partial y/\partial a_{mn} \end{bmatrix}$$

if $A \in \mathbb{R}^{m \times n}$ $\nabla_A y \in \mathbb{R}^{m \times n}$ "DENOMINATOR LAYOUT"

↑ Used in theory

↑ PYTHON uses this

"NUMERATOR LAYOUT"

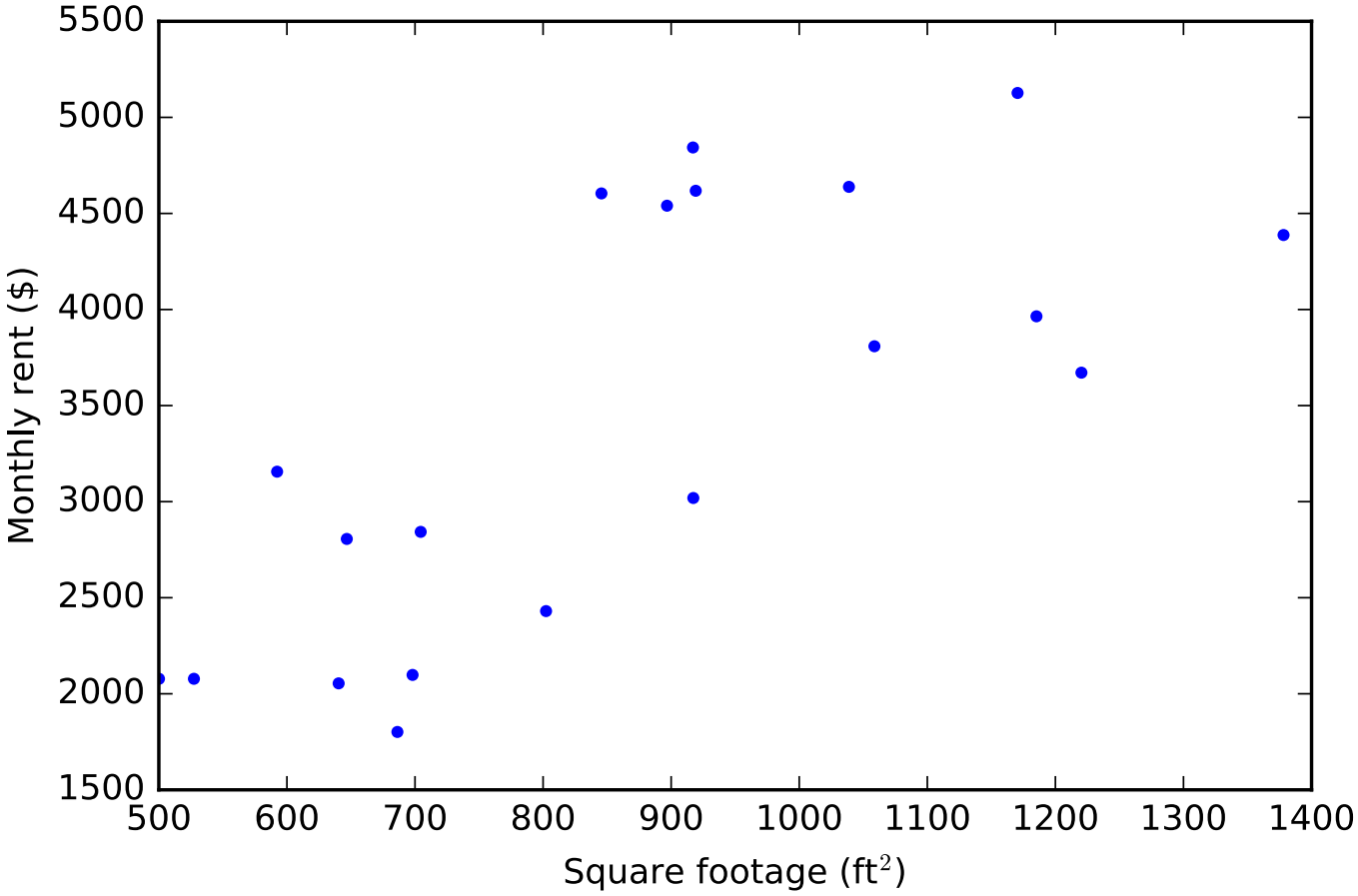
$x \in \mathbb{R}^{n \times 1}$

$A \in \mathbb{R}^{m \times n}$

$\nabla_x y \in \mathbb{R}^{1 \times n}$

$\nabla_A y \in \mathbb{R}^{n \times m}$

Back to our supervised learning example



Thus our strategy to find the best θ is to:

- Calculate

$$\frac{d\mathcal{L}}{d\theta}$$

- Solve for θ such that

$$\frac{\partial \mathcal{L}}{\partial \theta} = 0$$

Back to our supervised learning example

Re-writing the cost function, we have:

$$\mathcal{L}(\theta) = \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \theta^T \hat{\mathbf{x}}^{(i)})^2$$

$\theta = \begin{bmatrix} a \\ b \end{bmatrix}$

$\hat{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$

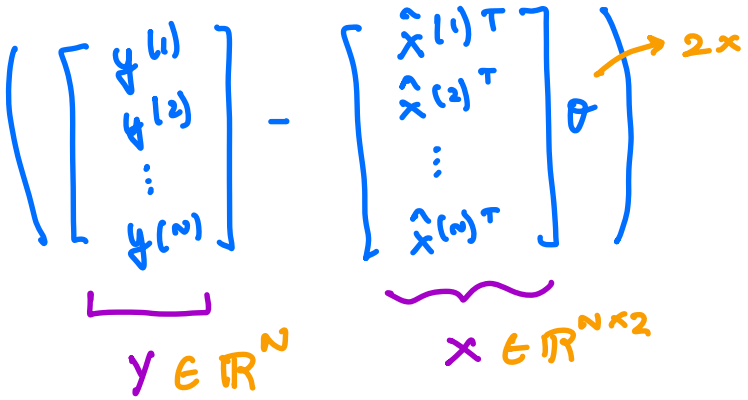
$$L(\theta) = \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \theta^T \hat{x}^{(i)})^T (y^{(i)} - \theta^T \hat{x}^{(i)})$$

$$= \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \hat{x}^{(i)T}\theta)(y^{(i)} - \hat{x}^{(i)T}\theta)$$

Scalar, so $x = x^T$

VECTORIZATION:

$$L(\theta) = \frac{1}{2}\left(\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} - \begin{bmatrix} \hat{x}^{(1)T} \\ \hat{x}^{(2)T} \\ \vdots \\ \hat{x}^{(N)T} \end{bmatrix}\theta\right)^T x$$



$$L(\theta) = \frac{1}{2}(y - x\theta)^T (y - x\theta)$$

$$L(\theta) = \frac{1}{2}\left(y^T y - 2y^T x\theta + \theta^T x^T x\theta\right)$$

$(1\times N)(N\times 2)(2\times 1)$

(1×1)

SCALAR

Back to our supervised learning example

Let's now take derivatives:

$$\frac{1}{2}\left(Y^TY - 2Y^T\mathbf{X}\theta + \theta^T\mathbf{X}^T\mathbf{X}\theta\right)$$

Y^T → z^T, $\mathbf{X}^T\mathbf{X}$ → A

$w = z^T\theta \quad \frac{\partial w}{\partial \theta} = z$

$z^T = y^T x$

$$\frac{1}{2}\left(y^Ty - 2z^T\theta + \theta^TA\theta\right)$$

$$\frac{\partial \theta^TA\theta}{\partial \theta} = (A + A^T)\theta$$

$$\frac{\partial L}{\partial \theta} = \frac{1}{2}\left(0 - 2z + (A + A^T)\theta\right)$$

$z^T = y^T x$, $z = x^T y$

Back to our supervised learning example

$$A = X^T X$$

$$\frac{\partial L}{\partial \theta} = \frac{1}{2}\left[0 - 2X^T y + (X^T X + X^T X)\theta\right]$$

$$= -X^T y + X^T X \theta$$

$$\text{Set} \quad \frac{\partial L}{\partial \theta} = 0$$

$$X^T y = X^T X \theta$$

$$\theta = (X^T X)^{-1} X^T y.$$

Back to our supervised learning example

This solution is called least-squares, and it appears in a variety of linear applications.

$$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T Y$$

2 × n   n × 2   2 × n   n × 1

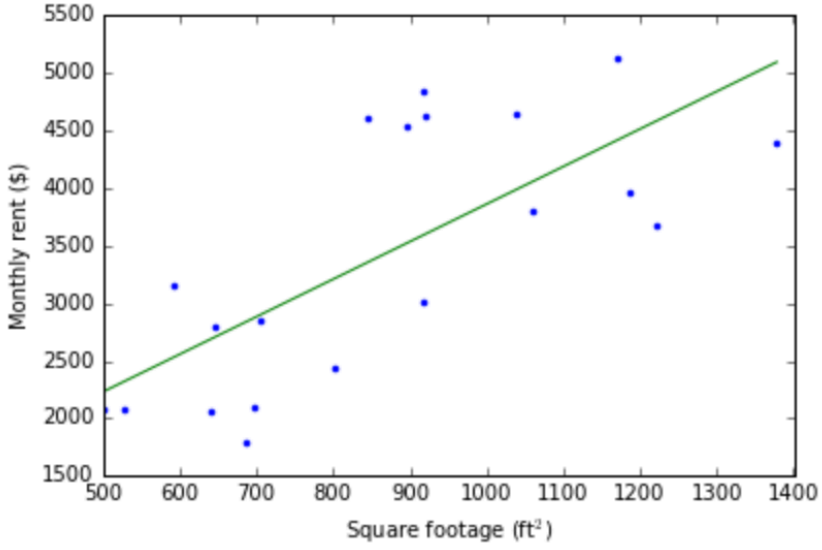
2 × 2

Back to our supervised learning example

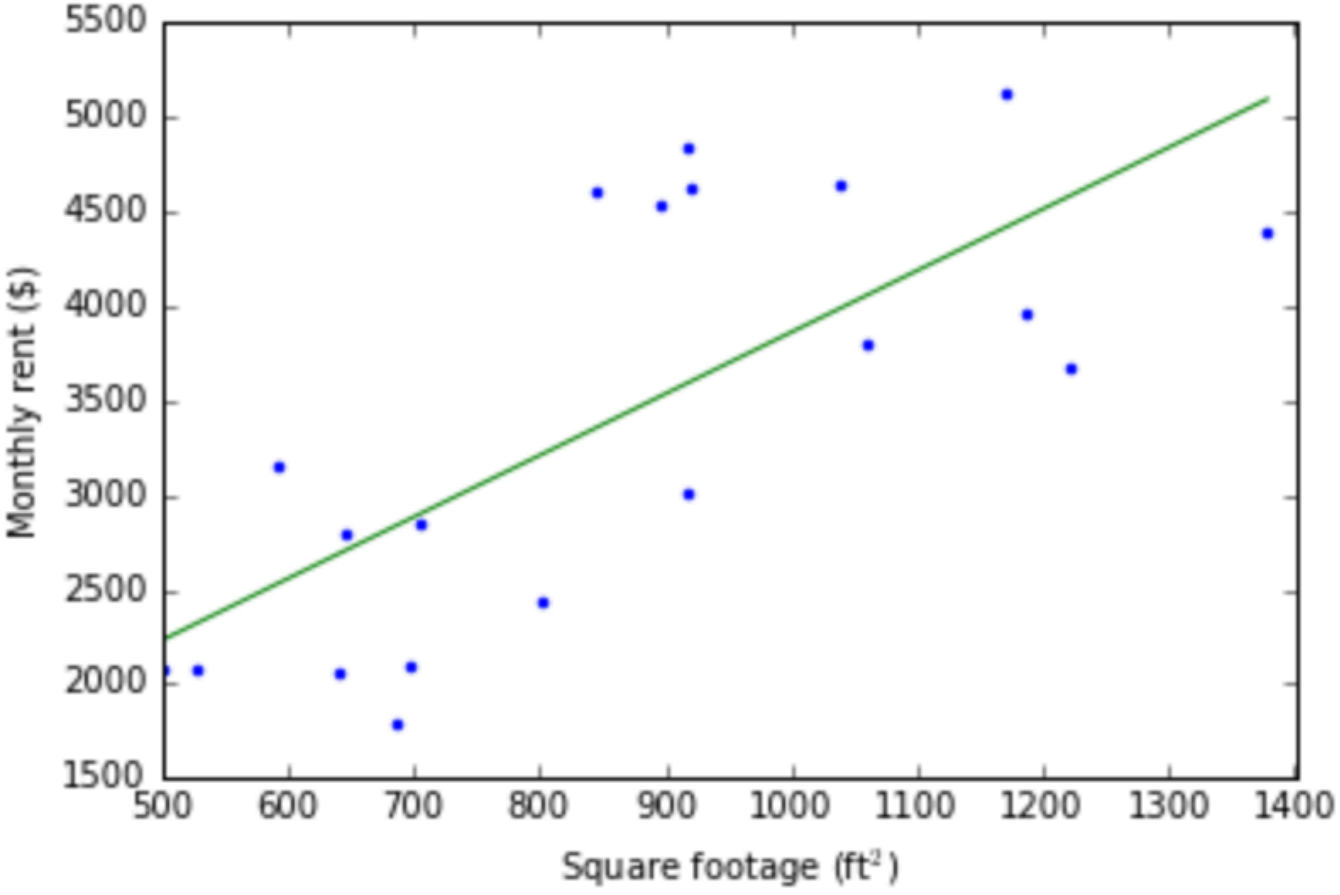
$\hat{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$

```
# Given paired data, with x being square footages and y being monthly rents
# Fit a linear model
xhat = np.vstack((x, np.ones_like(x)))
theta_lin = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y))

# Plot the data
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('Square footage (ft$^2$)')
ax.set_ylabel('Monthly rent ($)')
f.savefig('ml-basics_rent.pdf')

# Plot the regression line
xs = np.linspace(min(x), max(x),50)
xs = np.vstack((xs, np.ones_like(xs)))
plt.plot(xs[0,:], theta_lin.dot(xs))

print theta_lin
```

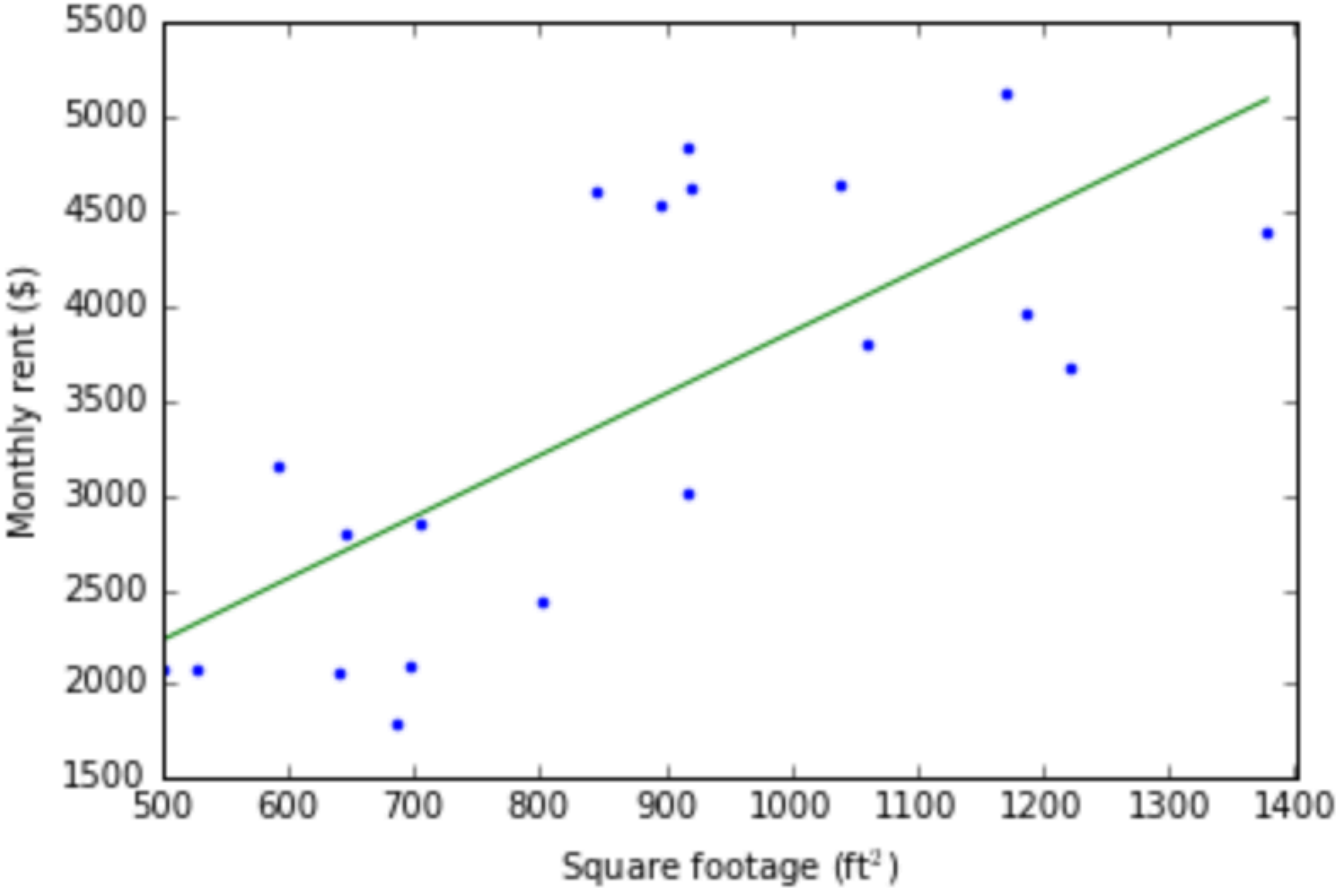
```
[   3.25900793  601.85544895]
```



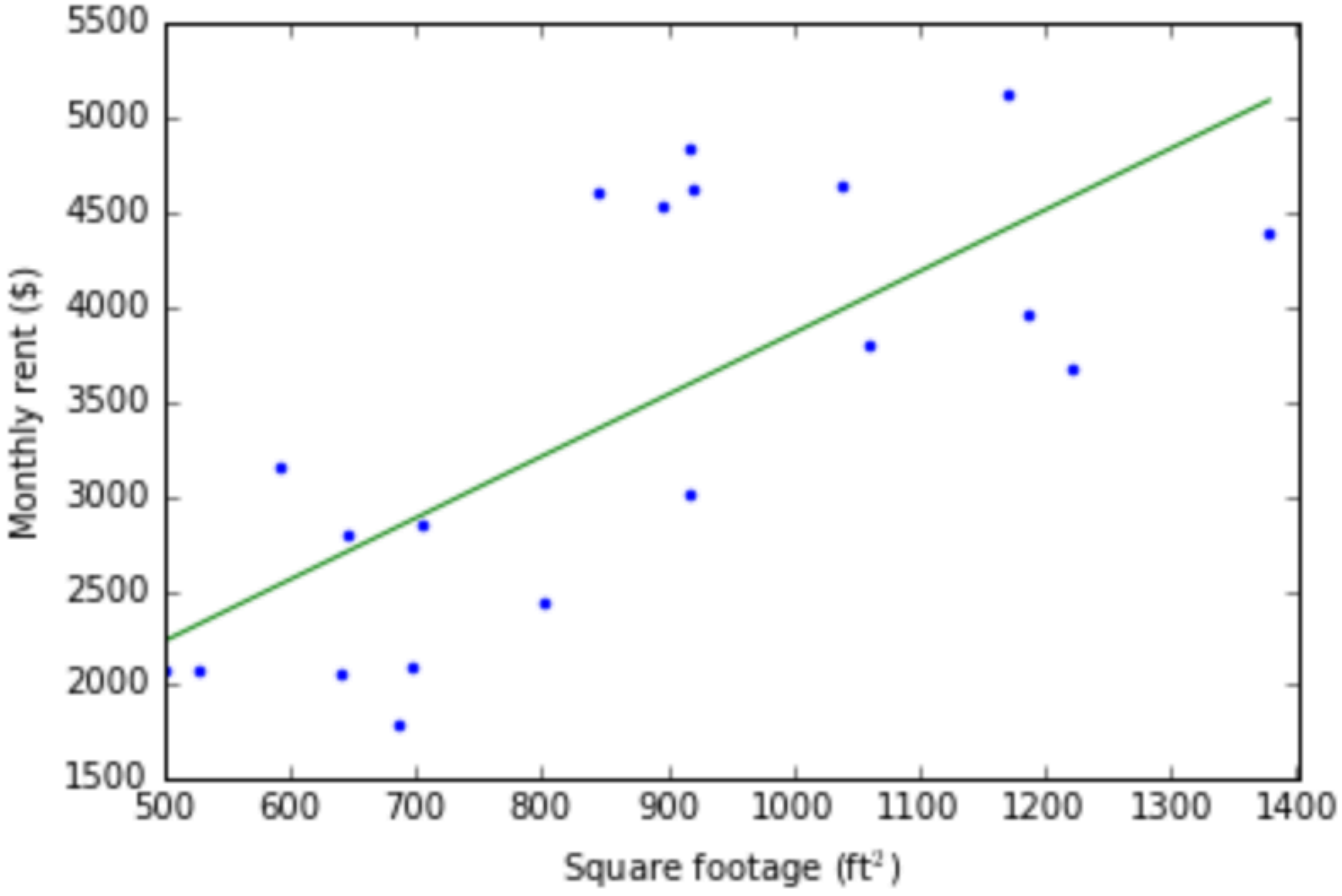
Can’t we do better?



Can’t we do better?



Zoom poll:



Question: Does our current least-squares formula allow for learning nonlinear polynomial fits, e.g.,

$$y = b + a_1 x_1 + a_2 x^2 + \cdots + a_n x^n$$

More generally…

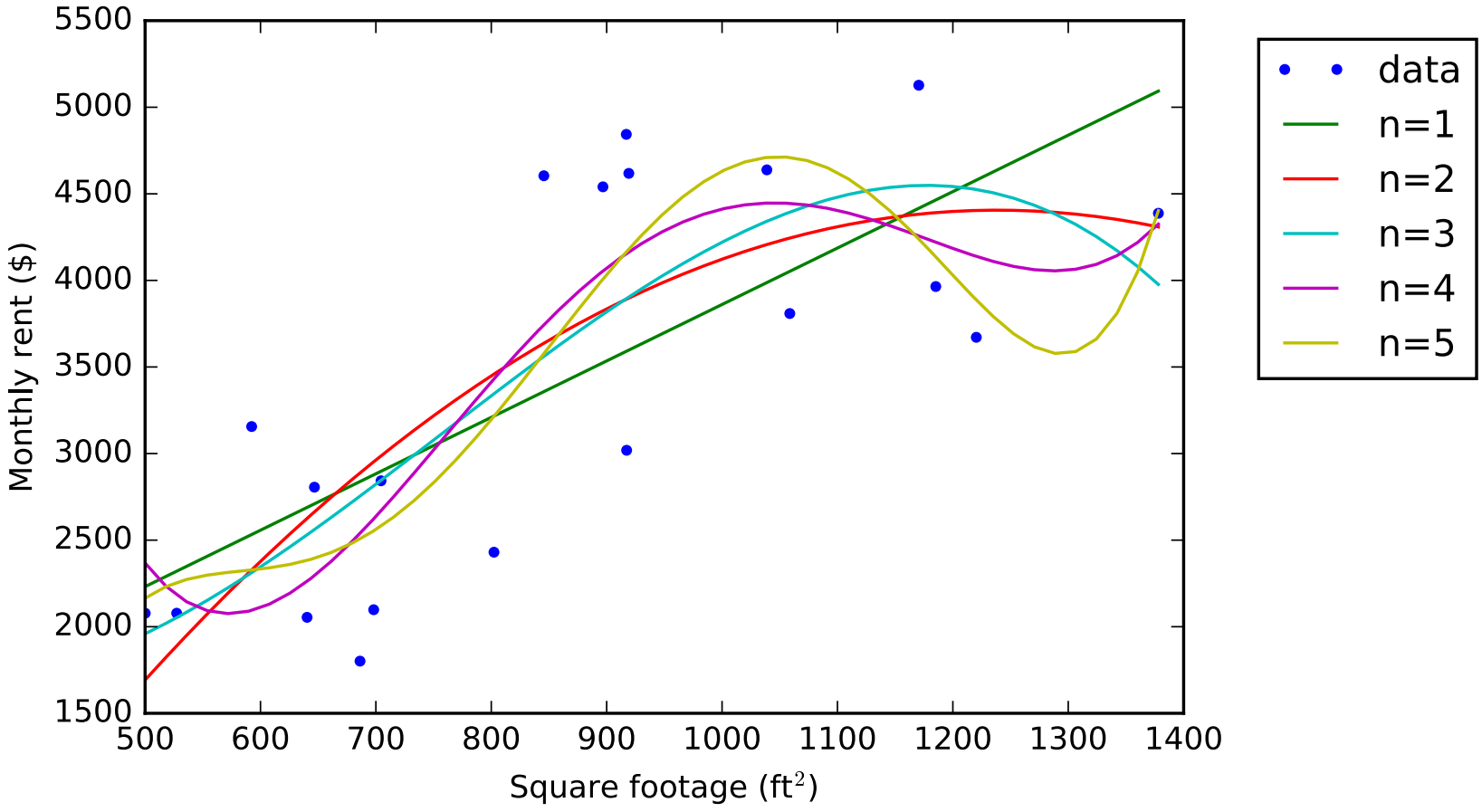
With our problem setup, it's straightforward to generalize to higher degree polynomial models, e.g.,

$$y = b + a_1 x_1 + a_2 x^2 + \cdots + a_n x^n$$

$\hat{x} = \begin{bmatrix} 1 \\ x \\ \vdots \\ x^n \end{bmatrix}$ $\theta = \begin{bmatrix} b \\ a_1 \\ \vdots \\ a_n \end{bmatrix}$

REST ARE SAME.

More generally…



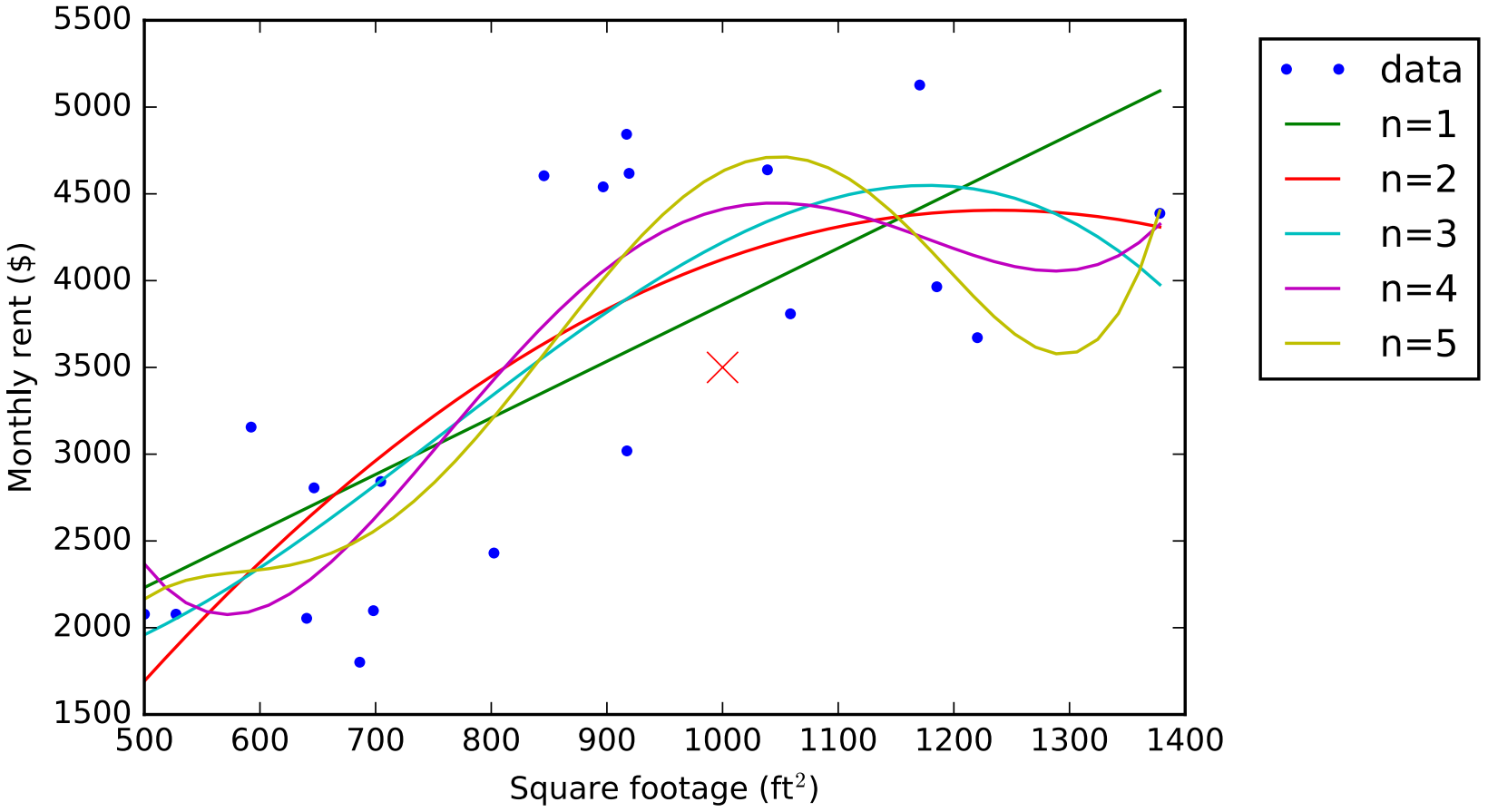
Now, a higher degree polynomial will *always* fit the **provided** data better. (Why)?

$L(\theta)$ for 3^{RD} order $\leq$ $L(\theta)$ for 1^{ST} polynomial

$b + a_1 x_1 + a_2 x_2^2 + a_3 x_3^3$

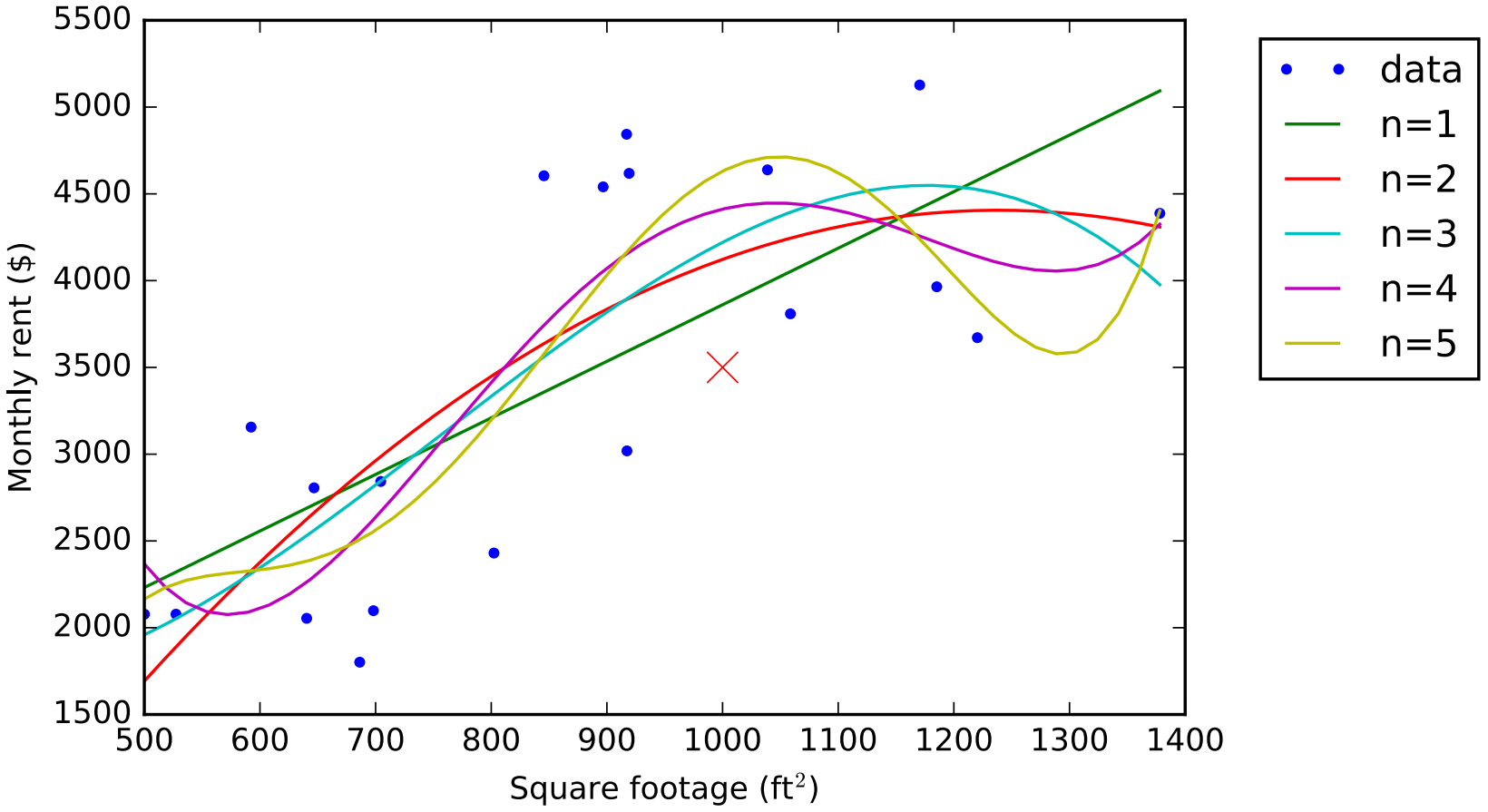
can just set $a_2 = a_3 = 0$ and same as 1st.

Now, let's say a new house pops up for rent…



In this scenario, the linear model best predicts the price of the new house ('x')

Model generalization



The fundamental problem here is that the more complex models may **not generalize as well** if the data come from a different ~~model~~ dataset.

Training and testing data

Overfitting

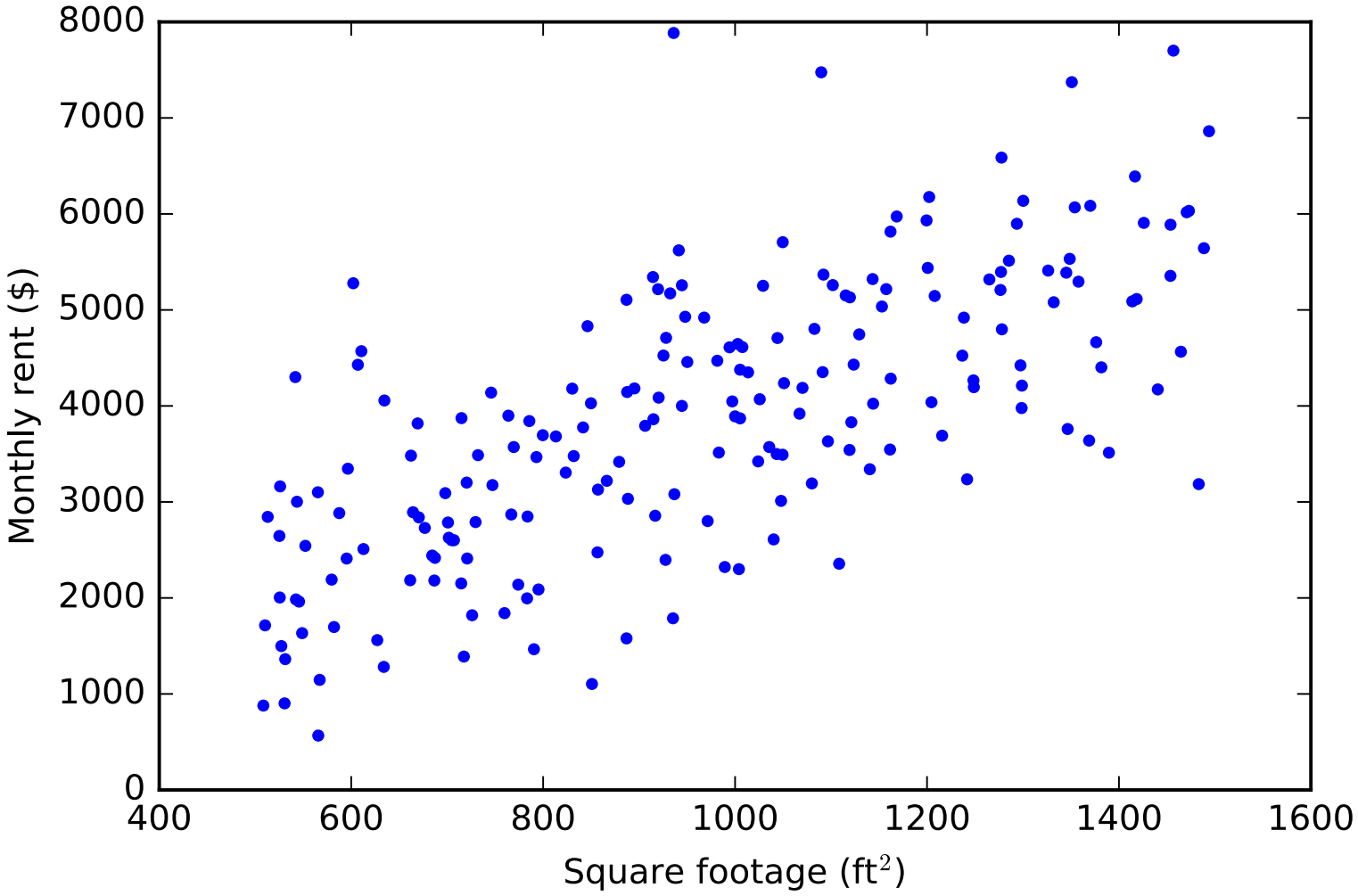
This idea of generalization can be made more formal by introducing the concepts of a *training set* and *testing set*.

- **Training data** is data that is used to learn the parameters of your model.
- **Testing data** is data that is excluded in training and used to score your model.

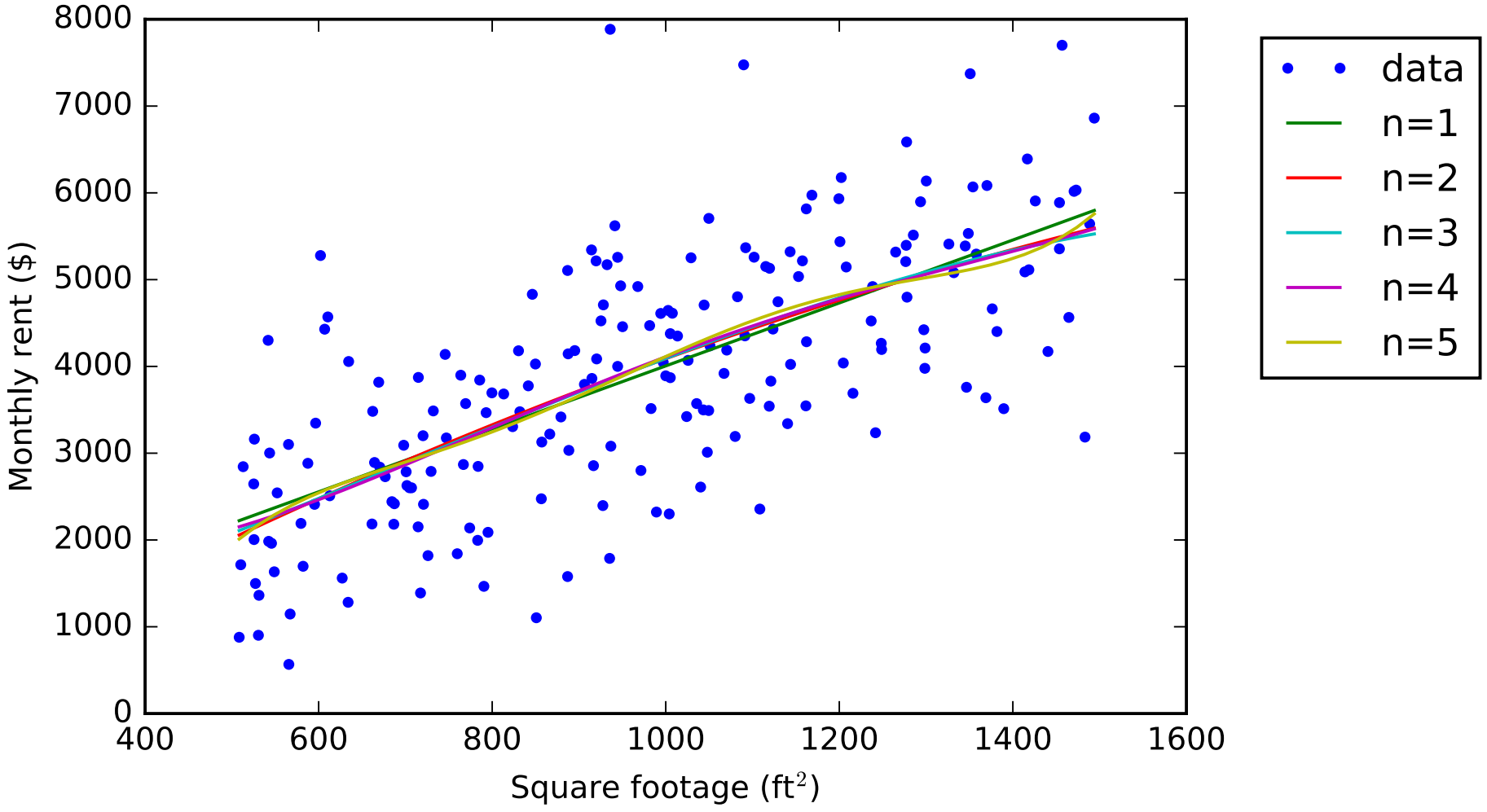
There is also a notion of validation data, which we will get to later.

A model which has very low training error but high testing error is called *overfit*.

More data helps to avoid overfitting



More data helps to avoid overfitting



More data helps to avoid overfitting

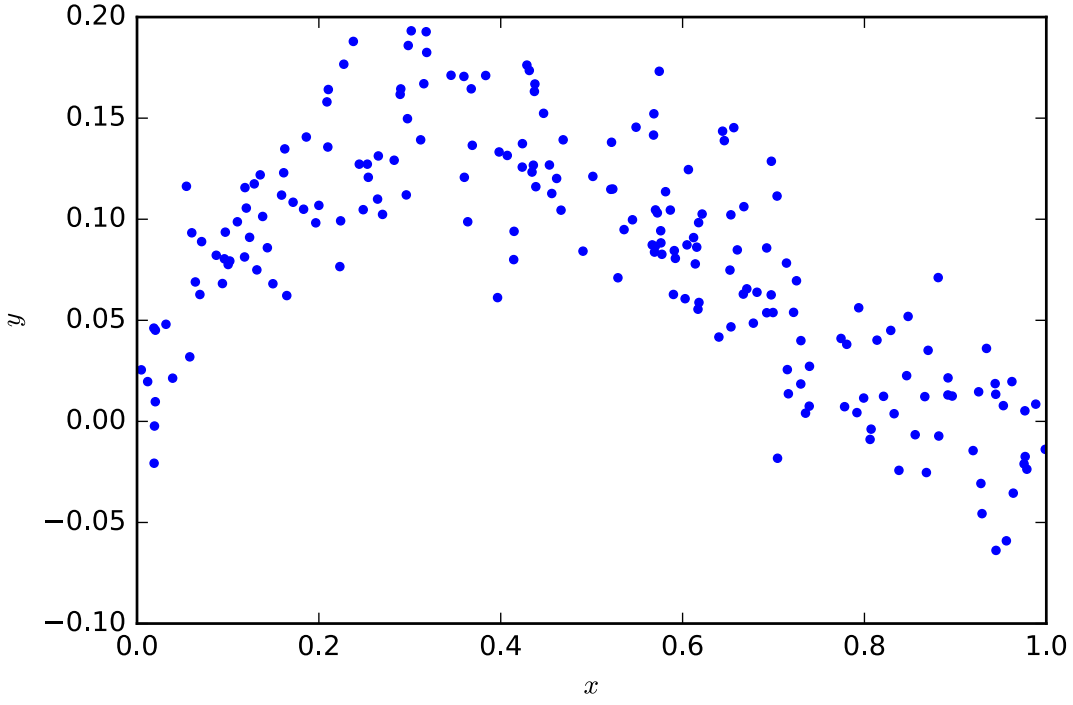
This suggests that when *a lot* of data is available, it may be appropriate to use more complex models. (Another technique we will discuss later, regularization, also helps with overfitting.) This is a shadow of things to come (i.e., neural networks which have large complexity).

Underfitting

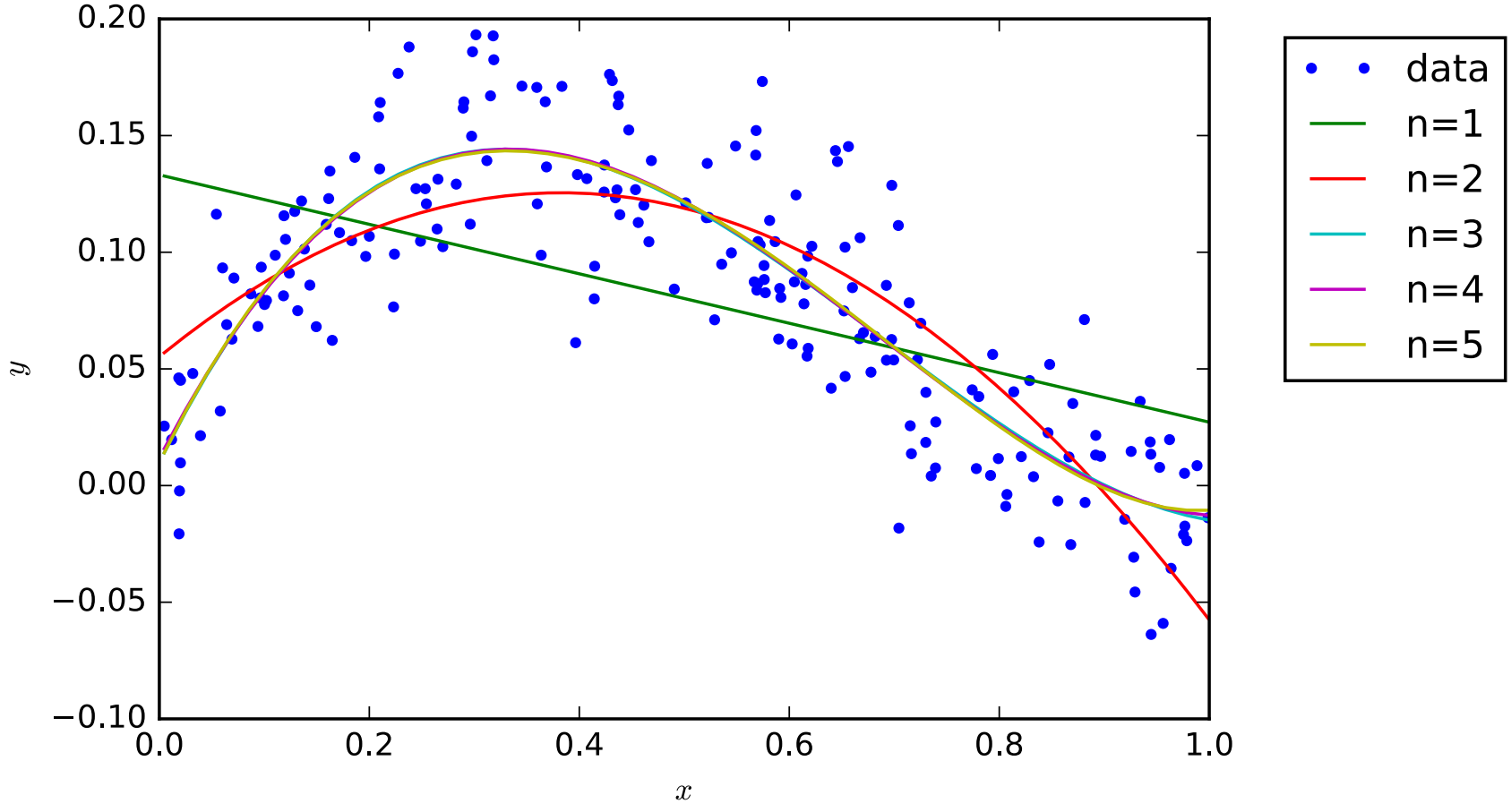
At the same time, we can not make the model *overly simple* as then we will underfit the data. This corresponds to a model that has both high training and high testing error, and the fit generally will not improve with more training data because the model is not expressive enough.

Underfitting

```
np.random.seed(0)  # Sets the random seed.
num_train = 200      # Number of training data points

# Generate the training data
x = np.random.uniform(low=0, high=1, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
f.savefig('ml-basics_underfitting-data.pdf')
```



Underfitting



How do we think of overfitting and underfitting?

Overfitting and underfitting are important problems to address in machine learning.

A more rigorous framework to think of these is as the *bias* and *variance* of the estimate of our parameters.

Picking a best model

From this picture, a reasonable way to pick a model is to assess its generalization error, and pick a setting of the parameters that results in minimal value.

A brief aside:

Sometimes there may be scenarios where dataset size is so limited that it is not possible to have a reasonably good test dataset. In these cases, we may utilize *model selection*, which is itself an entire field.

At a high level, the principles of model selection is to find a way to *penalize* the model for being overly complex. There is an art to designing some penalty terms.

Criterion for selecting models include:

- Bayes information criterion
- Akaike information criterion
- Deviance information criterion

My recommendation: if you have enough data to make a test set, do it.

Evaluating generalization error

Training, validation, and testing data

The standard for training, evaluating, and choosing models is to use different datasets for each step.

- **Training data** is data that is used to learn the parameters of your model.
- **Validation data** is data that is used to optimize the hyperparameters of your model. This avoids the potential of overfitting to nuances in the testing dataset.
- **Testing data** is data that is used to score your model.

Note, in many cases, testing and validation datasets are used interchangeably as datasets that are used to evaluate the model. In this scenario, hyperparameters would also be optimized using training data.

design choices made before training.

Evaluating generalization error

Assume: we have separate testing data

k-fold cross validation

Training set → Split into training and validation data.

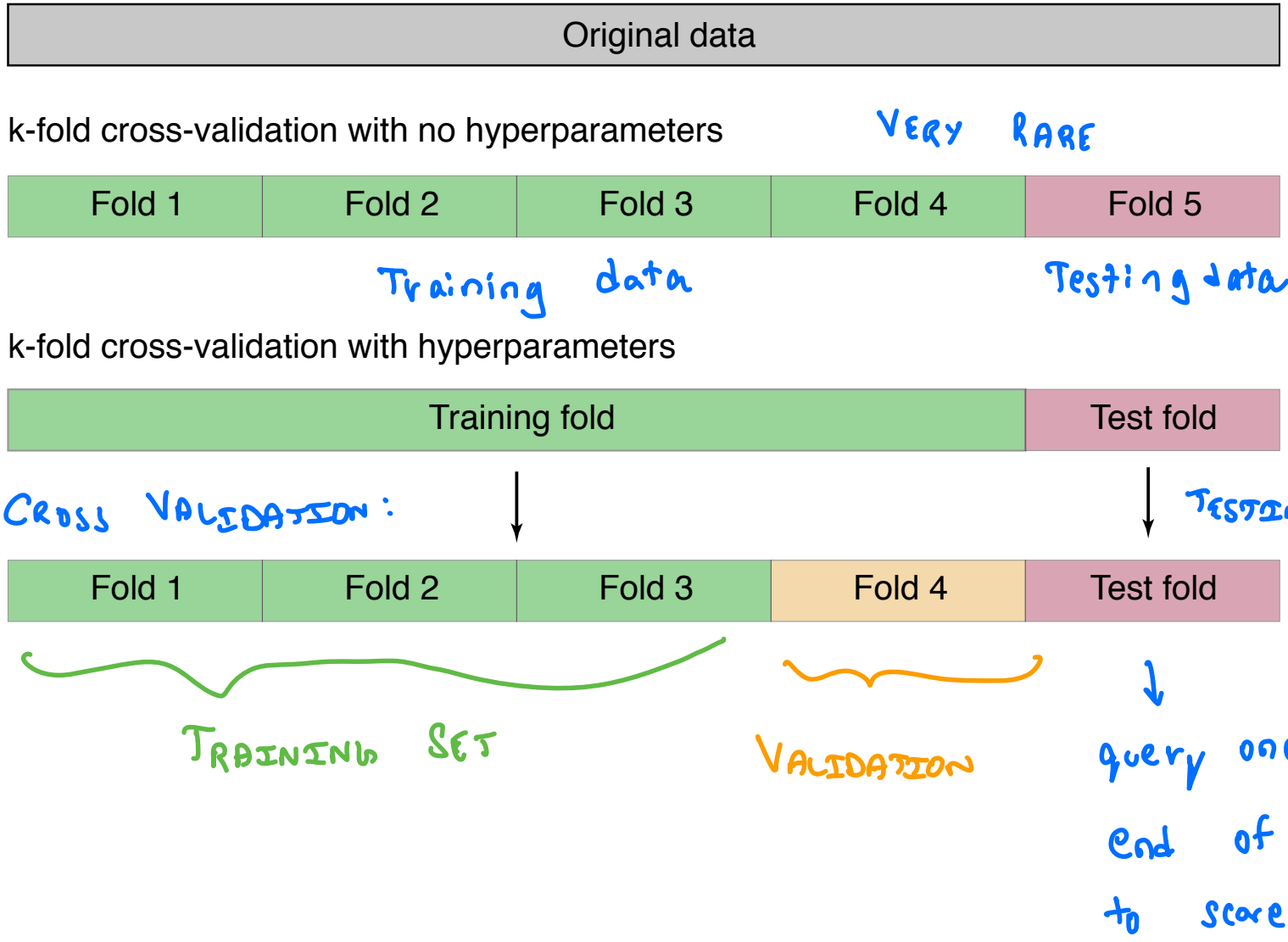
In a common scenario, you will be given a training dataset and a testing dataset. To train a model using this dataset, one common approach is k-fold cross validation. Then the procedure looks as follows:

- Let the training dataset contain N examples.
- Then, split the data into k equal sets, each of N/k examples. Each of these sets is called a "fold."
- $k-1$ of the folds are datasets that are used to train the model parameters.
- The remaining fold is a ~~testing~~ validation dataset used to evaluate the model.
- You may repeatedly train the model by choosing which folds comprise the training folds and the testing fold.

Assume: $N=1000$, $k=5$

split into 5 folds each with 200 data points

Evaluating generalization error



VERY RARE

Training data

Testing data

IMPORTANT

4-FOLD CROSS VALIDATION:

TESTING DATA

TRAINING SET

VALIDATION

query once at end of training to score your model.

$n = 1, 2, \ldots 5$ COULD BE THE POLYNOMIAL ORDERS OF THE LINEAR REGRESSION

↳ θ

VALIDATION:

LET ERRORS BE $L_1, L_2, \ldots L_5$

ON RUN 2,

FOLD 1 CAN BE THE VALIDATION SET.

THEN FOLD 2 CAN BE THE VALIDATION SET.

FIND Average L's accross the runs as $L_1, L_2 \ldots L_5$.

k-fold cross validation from housing example

Routines for training models and ~~testing~~ validating data.

```
def train_models(x, y):
    # Fit a polynomial model up to N=5
    N = 5
    thetas = []
    for i in np.arange(N):
        xhat = np.vstack((x, np.ones_like(x))) if i == 0 else np.vstack((x**(i+1), xhat))
        thetas.append(np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y)))

    return thetas

def ~~test~~ val_models(x, y, thetas):
    N = 5
    errors = []
    for i in np.arange(N):
        xhat = np.vstack((x, np.ones_like(x))) if i == 0 else np.vstack((x**(i+1), xhat))
        errors.append(np.sqrt(np.sum((y - thetas[i].dot(xhat))**2) / len(y)))
    return errors
```

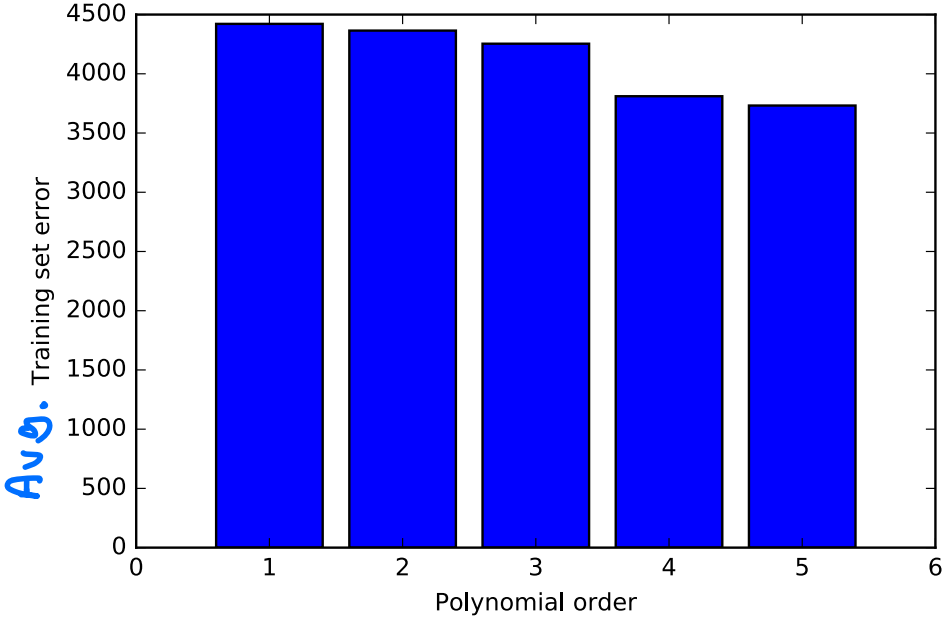
k-fold cross validation from housing example

Its all validation not test.

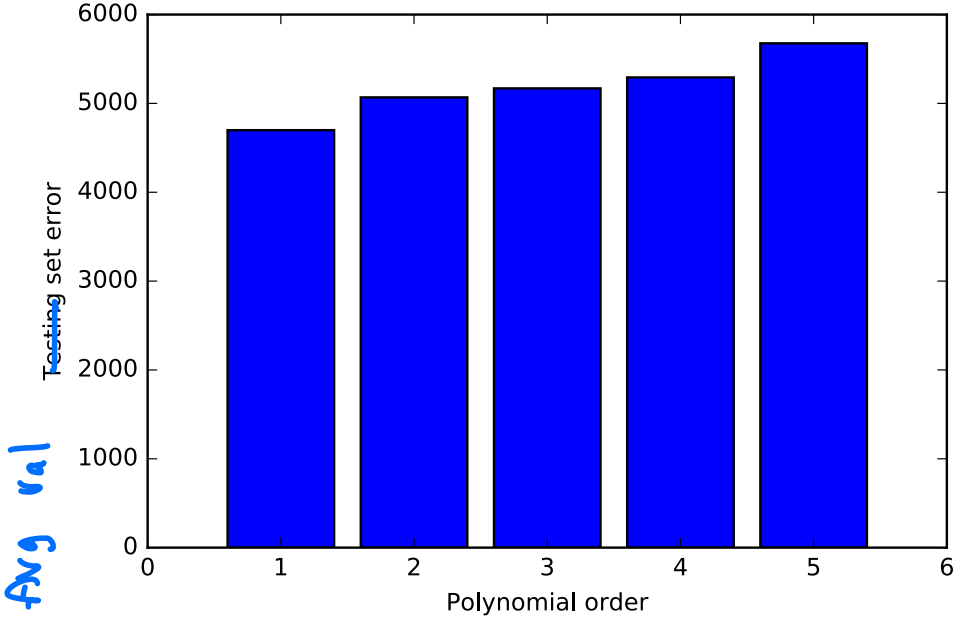
```
# (x,y) are the training data pairs from the supervised housing rent example
# num_train = len(x) is the number of examples
num_folds = 5
cv_idx = np.arange(num_train)
np.random.shuffle(cv_idx)
fold_size = num_train // 5
train_error, test_error = (np.zeros(5), np.zeros(5))
for i in np.arange(num_folds):
    test_idx = cv_idx[i*fold_size:(i+1)*fold_size]
    train_idx = np.concatenate((cv_idx[:i*fold_size], cv_idx[(i+1)*fold_size:]))
    x_test = x[test_idx]
    y_test = y[test_idx]
    x_train = x[train_idx]
    y_train = y[train_idx]
    thetas = train_models(x_train, y_train)
    train_error += test_models(x_train, y_train, thetas)
    test_error += test_models(x_test, y_test, thetas)

print train_error
print test_error
f = plt.figure()
ax = f.gca()
ax.bar(np.arange(5) + 1, train_error, width=0.8, align='center')
ax.set_xlabel('Polynomial order')
ax.set_ylabel('Training set error')
f.savefig('ml-basics_cv-train.pdf')
f = plt.figure()
ax = f.gca()
ax.bar(np.arange(5) + 1, test_error, width=0.8, align='center')
ax.set_xlabel('Polynomial order')
ax.set_ylabel('Testing set error')
f.savefig('ml-basics_cv-test.pdf')
```

```
[ 4421.38242236  4364.65797472  4254.01787959  3810.5847096   3731.95920537]
[ 4698.83840383  5067.8036503   5169.2079499   5292.40392825  5676.26492547]
```

k-fold cross validation from housing example



Avg.



Avg val

$y = ax + b$ is the best here.

COMMON METHODS:

* 5-FOLD

* LEAVE ONE OUT:

N = 1000

train on 999

validate on 1

repeat for different 1.

Other types of optimization

We've talked about examples where we want to *minimize* a mean-square error or distance metric.

Another metric that we may want to ~~minimize~~ maximize is the *probability of having observed the data*. In this framework, the data is modeled to have some distribution with parameters. We choose the parameters to maximize the probability of having observed our training data.

Assume we have a weighted coin and we want to find $P(H)$

DATA : H T H H T T H T

Seeing the data we might guess $P(H) = 1/2$.

$P(H) = 0.99$ is not consistent with the data.

MODEL : $x^{(i)} = \begin{cases} 0, & \text{with } P = 1-\theta \\ 1, & \text{with } P = \theta \end{cases}$

Maximum-likelihood introduction

Find θ that maximizes the likelihood of seeing the data.

LIKELIHOOD: Say, Model 1 : $\theta = 1$

Model 2 : $\theta = 0.75$

Model 3 : $\theta = 0.5$

Model 1 : 1.0.1.1.0.0.1.0

$= 0$

0 likelihood to see the data given $\theta = 1$

Model 2 : $(0.75)^4 (0.25)^4 = 0.00124$

Model 3 : $(0.5)^4 (0.5)^4 = 0.0039$

Data has highest chance of occuring for model 3.

But there are ∞ possible models.

L = Likelihood $= \theta^4 (1-\theta)^4$

$$\frac{\partial L}{\partial \theta} = 0$$

$$\log L = 4\log\theta + 4\log(1-\theta)$$

$$\frac{\partial \log L}{\partial \theta} = \frac{4}{\theta} - \frac{4}{1-\theta} = 0$$

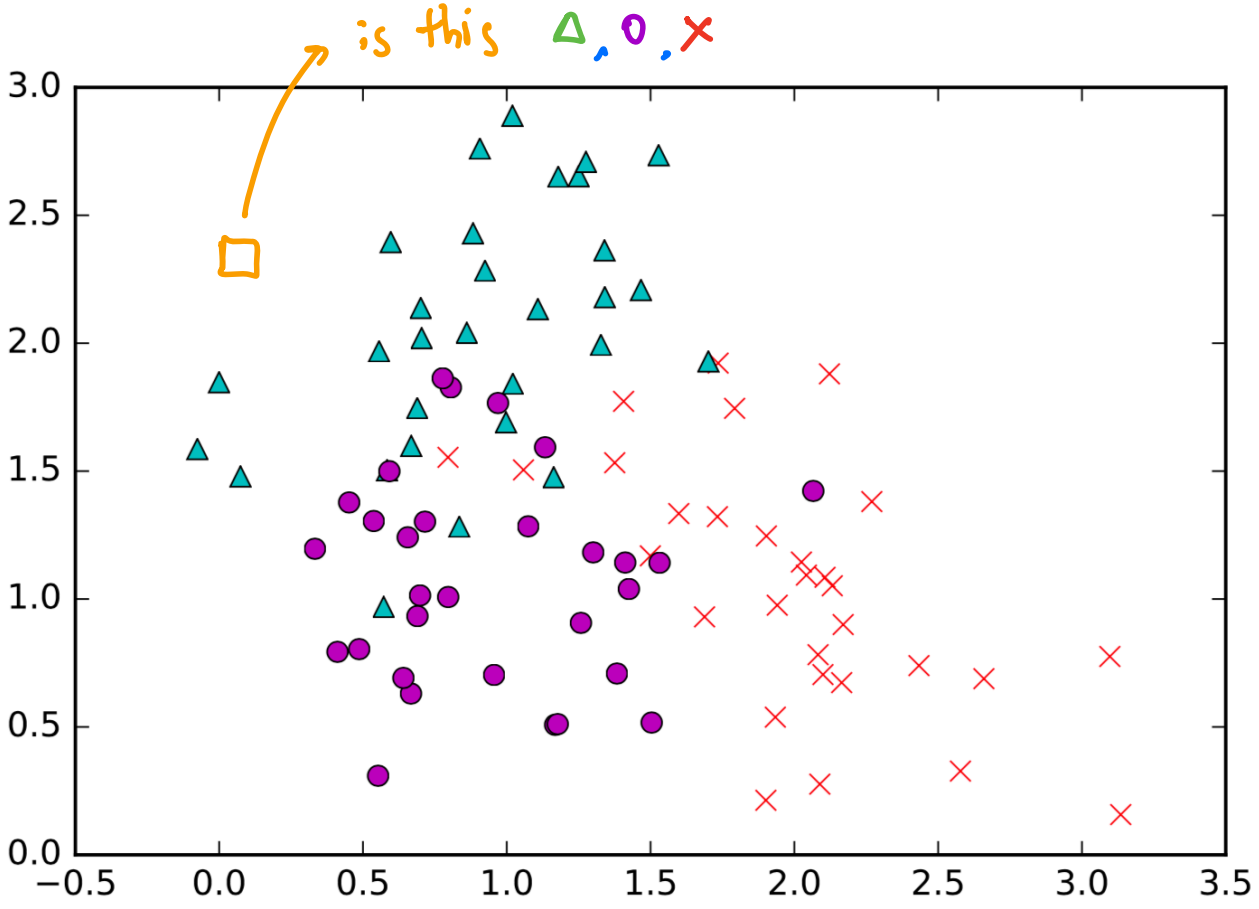
$$\theta = 1-\theta$$

$$\theta = 1/2$$

Maximum-likelihood estimation

Example of ML estimation

Say we receive paired data $\{\mathbf{x}^{(i)}, y^{(i)}\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^2$ is a data point that belongs to one of three classes, $y^{(i)} \in \{1, 2, 3\}$. Classes $1, 2, 3$ denote the three possible classes (or labels) that a data point could belong to. The drawing below (with appropriate labels) represents this:



Chain rule for probability

$$Pr(A=a) = p_A(a) = p(a)$$

$$Pr(B=b) = p_B(b) = p(b)$$

$$Pr(A=a \text{ and } B=b) = p_{A,B}(a,b) = p(a,b)$$

$$Pr(A=a \text{ and } B=b) = Pr(A=a) \cdot Pr(B=b \text{ given } A=a)$$

$$p(a,b) = p(a) \cdot p(b|a)$$

$$= p(b) \cdot p(a|b)$$

$$p(a,b,c) = p(c) \cdot p(a|c) \cdot p(b|c,a)$$

$$= p(a) \cdot p(b|a) \cdot p(c|a,b)$$

$$= p(a,c) \cdot p(b|a,c)$$

Chain rule for probability

1. $P(b,c|d,e) = \frac{?}{P(d) \cdot P(e|d)}$

$\rightarrow P(d,e)$

$? = P(b,c,d,e)$ //

2. $P(d,e) \cdot ? = \frac{P(a,b,c,d,e)}{P(a|b,c,d,e)}$

$P(d,e) \cdot ? \cdot P(a|b,c,d,e) = P(a,b,c,d,e)$

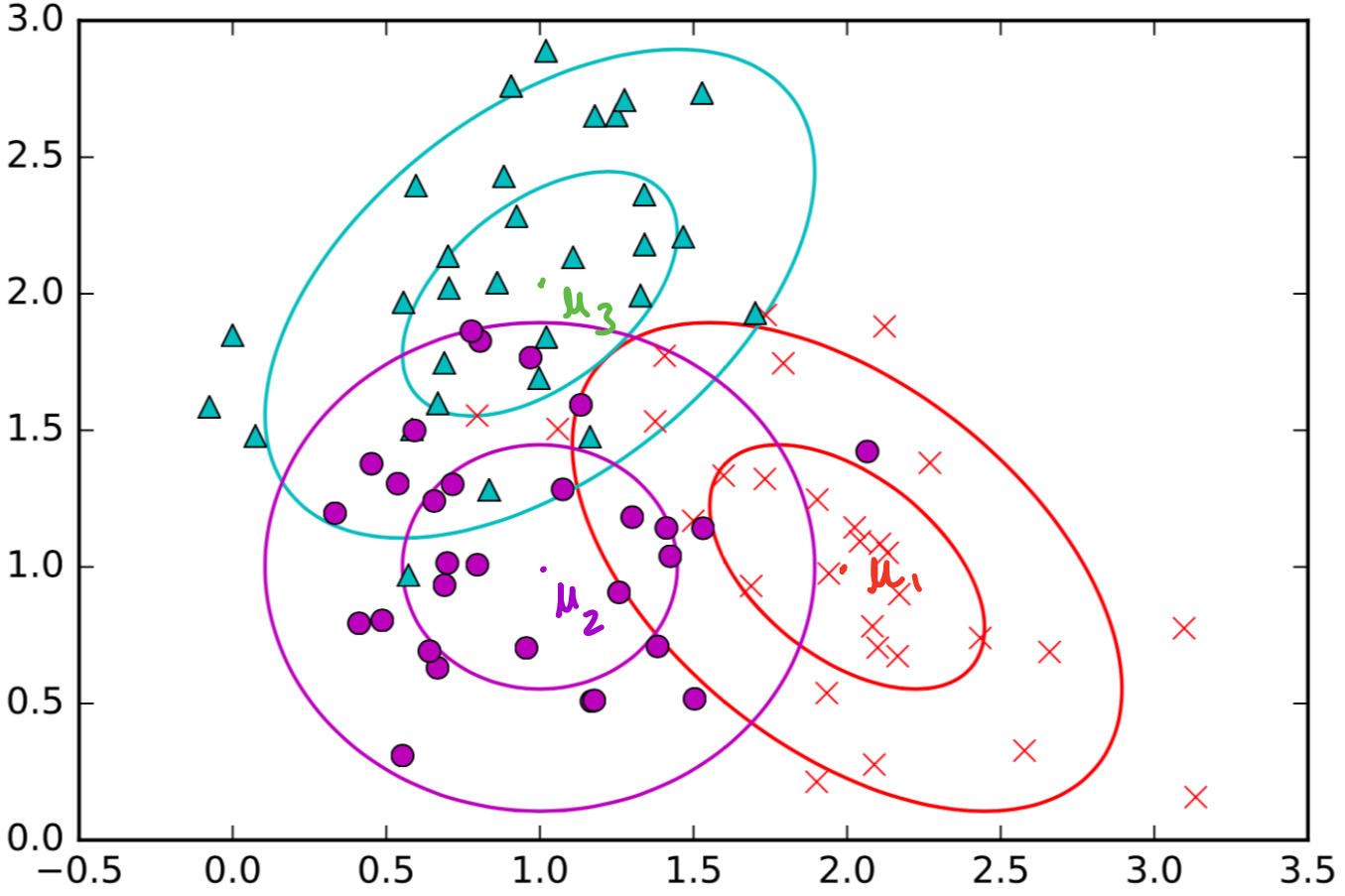
$P(a|b,c,d,e) \times P(b,c,d,e)$

$P(b,c|d,e) \times P(d,e)$

$?$

Example of ML estimation (cont.)

We may want to model each cluster as being *generated* by a multivariate Gaussian distribution. Concretely, in this example, there are three Gaussian clusters, each one describing the distribution of points for that class.



Example of ML estimation (cont.)

The model setup is as follows:

- Each data point $\mathbf{x}^{(i)}$ belongs to class $y^{(i)}$.
- Each class y_i is parametrized according to a distribution.

$$\begin{aligned} \mathbf{x}^{(i)}|y^{(i)}=1 &\sim \mathcal{N}(\mu_1,\Sigma_1) \\ \mathbf{x}^{(i)}|y^{(i)}=2 &\sim \mathcal{N}(\mu_2,\Sigma_2) \\ \mathbf{x}^{(i)}|y^{(i)}=3 &\sim \mathcal{N}(\mu_3,\Sigma_3) \end{aligned}$$

Thus, the parameters we can choose to optimize our model are $\theta = \{\mu_1, \Sigma_1, \mu_2, \Sigma_2, \mu_3, \Sigma_3\}$.

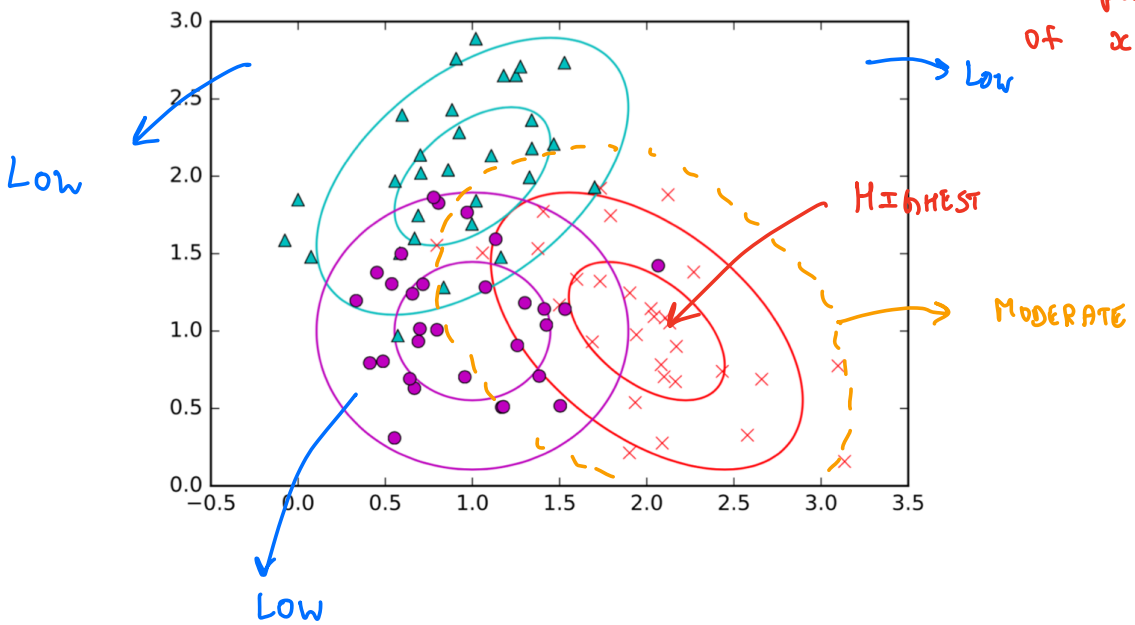
- We'll assume all classes are equally probable a priori (so that maximum likelihood estimation and maximum a posteriori estimation are equivalent).
- Finally, we'll assume each data point is independent, so that we can easily write out probabilities, i.e.,

$$p\left(\{\mathbf{x}^{(i)}, y^{(i)}\}, \{\mathbf{x}^{(j)}, y^{(j)}\}\right) = p\left(\mathbf{x}^{(i)}, y^{(i)}\right) p\left(\mathbf{x}^{(j)}, y^{(j)}\right)$$

$P(x_i \mid y_i = 1)$ = Probability of the data point x_i which is of class i to be at various points in the space

↓ pdf

$Pr(x_i \mid y_i = 1)$ values at various points of x_i

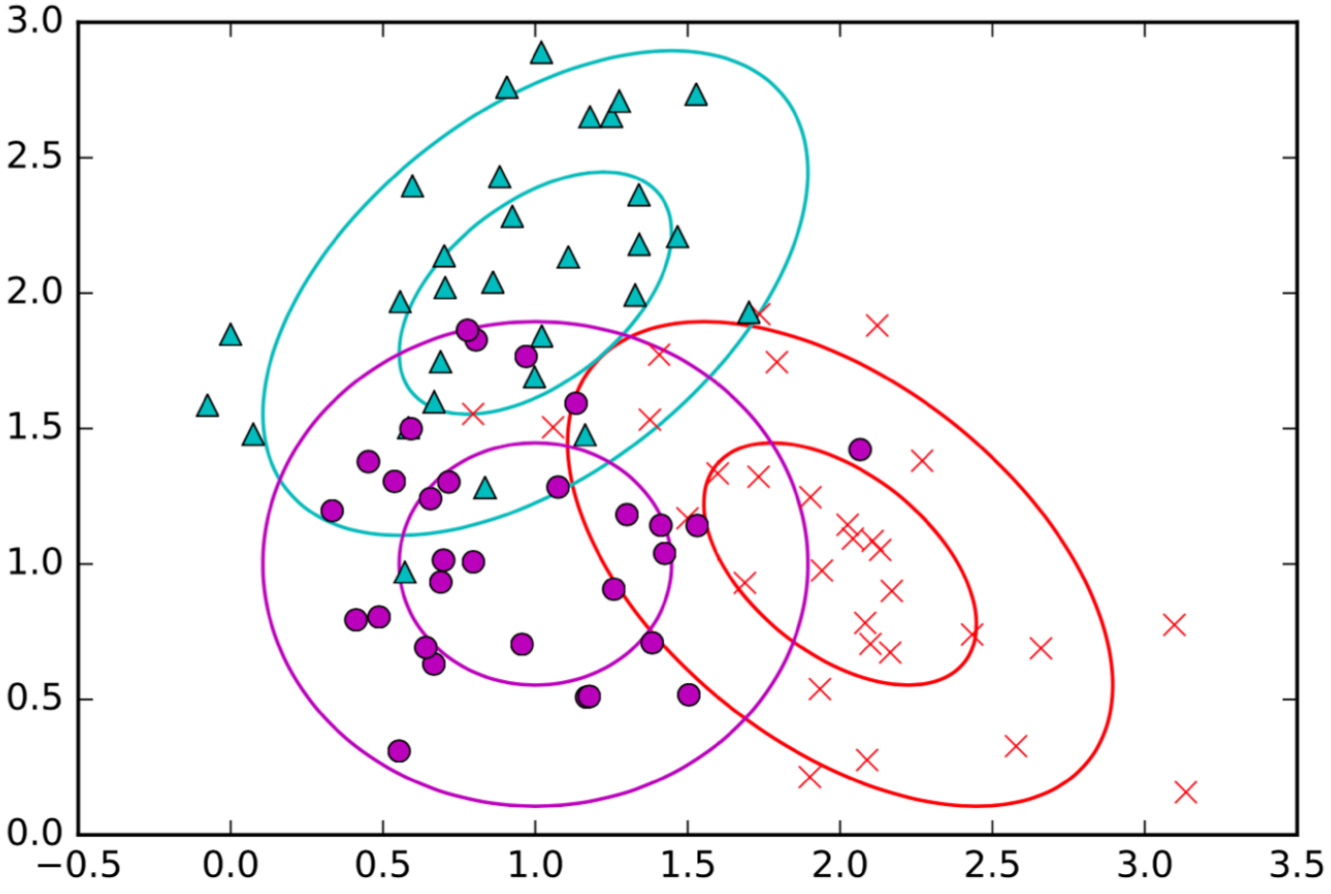


If A is discrete Random variable/vector,

$P()$ → probability (probability mass function)

A → continuous, $P()$ → probability density function.

Maximum-likelihood estimation



Maximum-likelihood estimation

Example of ML estimation (cont.)

We'd like to maximize the likelihood of having seen the dataset. This can be written as

$$\mathcal{L} = p\left(\{\mathbf{x}^{(1)}, y^{(1)}\}, \{\mathbf{x}^{(2)}, y^{(2)}\}, \ldots, \{\mathbf{x}^{(N)}, y^{(N)}\}\right)$$

θ is dropped for writing convenience.

$$L = \prod_{i=1}^{N} p(x^{(i)}, y^{(i)} | \theta)$$

$$\log L = \sum_{i=1}^{N} \log p(x^{(i)}, y^{(i)} | \theta) = \sum_{i=1}^{N} \log\left(p(y^{(i)} | \theta) \cdot p(x^{(i)} | y^{(i)}, \theta)\right)$$

$$= \sum_{i=1}^{N} \left(\log p(y^{(i)} | \theta) + \log p(x^{(i)} | y^{(i)}, \theta)\right)$$

$$p(y^{(i)} | \theta) = 1/3$$

$$= k + \sum_{i=1}^{N} \log \mathcal{N}(\mu_{y^{(i)}}, \Sigma_{y^{(i)}})$$

$$k = \sum_{i=1}^{N} \log(1/3)$$

Example of ML estimation (cont.)

Now, to simplify the probabilities, we evaluate $\log p(\mathbf{x}_i|y_i)$. We first observe:

$$\log p(\mathbf{x}^{(i)}|y^{(i)}=j) \quad = \quad \log \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)\right)$$

Maximum-likelihood estimation

Example of ML estimation (cont.)

Now, to simplify the probabilities, we evaluate $\log p(\mathbf{x}_i|y_i)$. We first observe:

$$\begin{aligned}
\log p(\mathbf{x}^{(i)}|y^{(i)}=j) &= \log \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)\right) \\
&= -\frac{n}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j) \\
&= k_2 - \frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)
\end{aligned}$$

Hence, we arrive at:

$$\log\mathcal{L} = k + \sum_{i=1}^{N} -\frac{1}{2}\log|\Sigma_{y^{(i)}}| - \frac{1}{2}(\mathbf{x}^{(i)}-\mu_{y^{(i)}})^T\Sigma_{y^{(i)}}^{-1}(\mathbf{x}^{(i)}-\mu_{y^{(i)}})$$

where $k = k_1 + Nk_2$.

Examle of ML estimation (cont.)

Let's then find the optimal parameters by differentiating and setting to zero. First, we find the optimal μ_j. We note that only the last term in the log likelihood has μ_j terms in instances where $y^{(i)} = j$.

$$\begin{aligned}
\frac{\partial \log \mathcal{L}}{\partial \mu_j} &= \frac{\partial}{\partial \mu_j} \left[\sum_{i:y^{(i)}=j} -\frac{1}{2}(\mathbf{x}^{(i)} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}^{(i)} - \mu_j) \right] \\
&= -\frac{1}{2} \frac{\partial}{\partial \mu_j} \left[\sum_{i:y^{(i)}=j} (\mathbf{x}^{(i)})^T \Sigma_j^{-1} \mathbf{x}^{(i)} - 2\mu_j^T \Sigma_j^{-1} \mathbf{x}^{(i)} + \mu_j^T \Sigma_j^{-1} \mu_j) \right] \\
&= -\frac{1}{2} \left[\sum_{i:y^{(i)}=j} -2\Sigma_j^{-1} \mathbf{x}^{(i)} + 2\Sigma_j^{-1} \mu_j \right]
\end{aligned}$$

Setting the derivative to zero, we have that:

$$\mu_j = \frac{1}{N_j} \sum_{i:y^{(i)}=j} \mathbf{x}^{(i)}$$

where $N_j = \sum_{i:y^{(i)}=j} 1$, i.e., the number of training examples where $y^{(i)} = j$. Does this answer make sense?

Maximum-likelihood estimation

Examle of ML estimation (cont.)

We next take the derivative with respect to Σ, using the following facts (for more information, see "Tools" lecture notes on derivatives w.r.t. matrices):

$$\begin{aligned}\frac{\partial}{\partial \Sigma}\mathrm{tr}(\Sigma^{-1}\mathbf{A}) &= -\Sigma^{-T}\mathbf{A}^T\Sigma^{-T} \\ \frac{\partial}{\partial \Sigma}\log|\Sigma| &= \Sigma^{-T}\end{aligned}$$

Now, differentiating:

$$\begin{aligned}\frac{\partial \log \mathcal{L}}{\partial \Sigma_j} &= \frac{\partial}{\partial \Sigma_j}\left[\sum_{i:y^{(i)}=j} -\frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)\right] \\ &= -\frac{1}{2}\frac{\partial}{\partial \Sigma_j}\left[\sum_{i:y^{(i)}=j} \log|\Sigma_j| + \mathrm{tr}\left((\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)\right)\right] \\ &= -\frac{1}{2}\frac{\partial}{\partial \Sigma_j}\left[\sum_{i:y^{(i)}=j} \log|\Sigma_j| + \mathrm{tr}\left(\Sigma_j^{-1}(\mathbf{x}^{(i)}-\mu_j)(\mathbf{x}^{(i)}-\mu_j)^T\right)\right] \\ &= -\frac{1}{2}\left[\sum_{i:y^{(i)}=j} \Sigma_j^{-T} - \Sigma_j^{-T}(\mathbf{x}^{(i)}-\mu_j)(\mathbf{x}^{(i)}-\mu_j)^T\Sigma_j^{-T}\right]\end{aligned}$$

Example of ML estimation (cont.)

Setting the derivative equal to zero, we arrive at:

$$\sum_{i:y^{(i)}=j} \Sigma_j^{-1} = \sum_{i:y^{(i)}=j} \Sigma_j^{-1}(\mathbf{x}^{(i)} - \mu_j)(\mathbf{x}^{(i)} - \mu_j)^T \Sigma_j^{-1}$$

Pre and post-multiplying both sides by Σ_j, we then have:

$$\Sigma_j = \frac{1}{N_j} \sum_{i:y^{(i)}=j} (\mathbf{x}^{(i)} - \mu_j)(\mathbf{x}^{(i)} - \mu_j)^T$$

Does this answer make sense?

Maximum-likelihood estimation

Example of ML estimation (cont.)

Solving for the optimal parameters μ_j and Σ_j for all classes, we arrive at the following solution:

```
# x and y are the data and their corresponding labels
# x is 2 x num_examples of the data points
# y is 1 x num_examples of the data labels

class_idxs = [np.where(y == j)[0] for j in np.unique(y)]
means = []
covs = []

for this_class in class_idxs:
    means.append(np.mean(x[:, this_class], axis=1))
    covs.append(np.cov(x[:, this_class]))
```

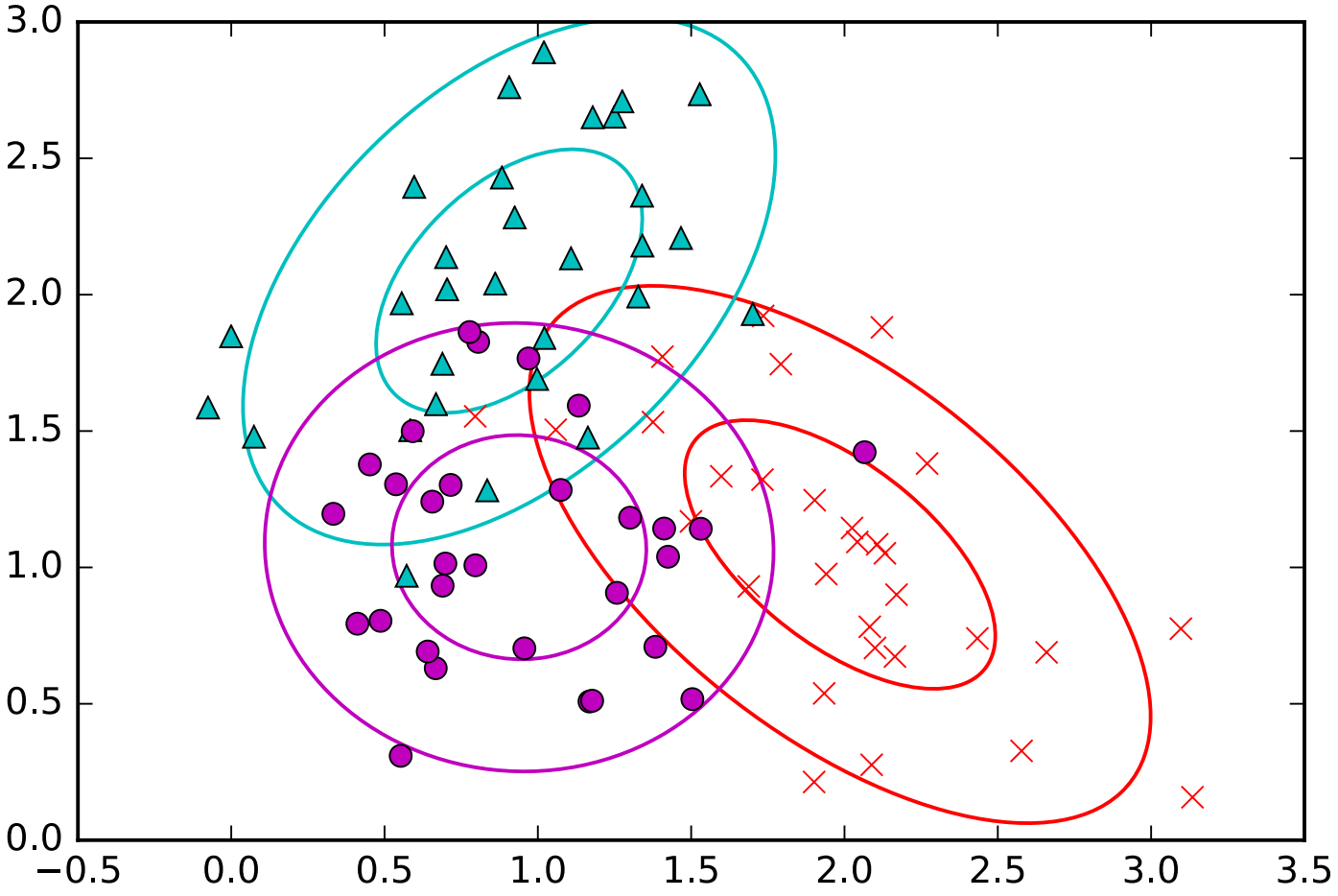
Example of ML estimation (cont.)

Solving for the optimal parameters μ_j and Σ_j for all classes, we arrive at the following solution:

```
f = plt.figure()
ax = f.gca()
ax.plot(data1[0,:], data1[1,:], marker='x', linestyle='', color=class_colors[0])
ax.plot(data2[0,:], data2[1,:], marker='^', linestyle='', color=class_colors[1])
ax.plot(data3[0,:], data3[1,:], marker='o', linestyle='', color=class_colors[2])


for i in np.arange(len(class_idxs)):
    vals, vecs = eigsorted(covs[i])
    theta = np.degrees(np.arctan2(*vecs[:,0][::-1]))
    for nstd in np.arange(3):
        w, h = 2 * nstd * np.sqrt(vals)
        e = Ellipse(xy=(means[i][0], means[i][1]),
                    width=w, height=h,
                    angle=theta, color=class_colors[i])
        e.set_facecolor('none')
        ax.add_artist(e)

f.savefig('ml-basics_maxlik-data_fit-ellipsoids.pdf')
```

Example of ML estimation (cont.)

Solving for the optimal parameters μ_j and Σ_j for all classes, we arrive at the following solution:



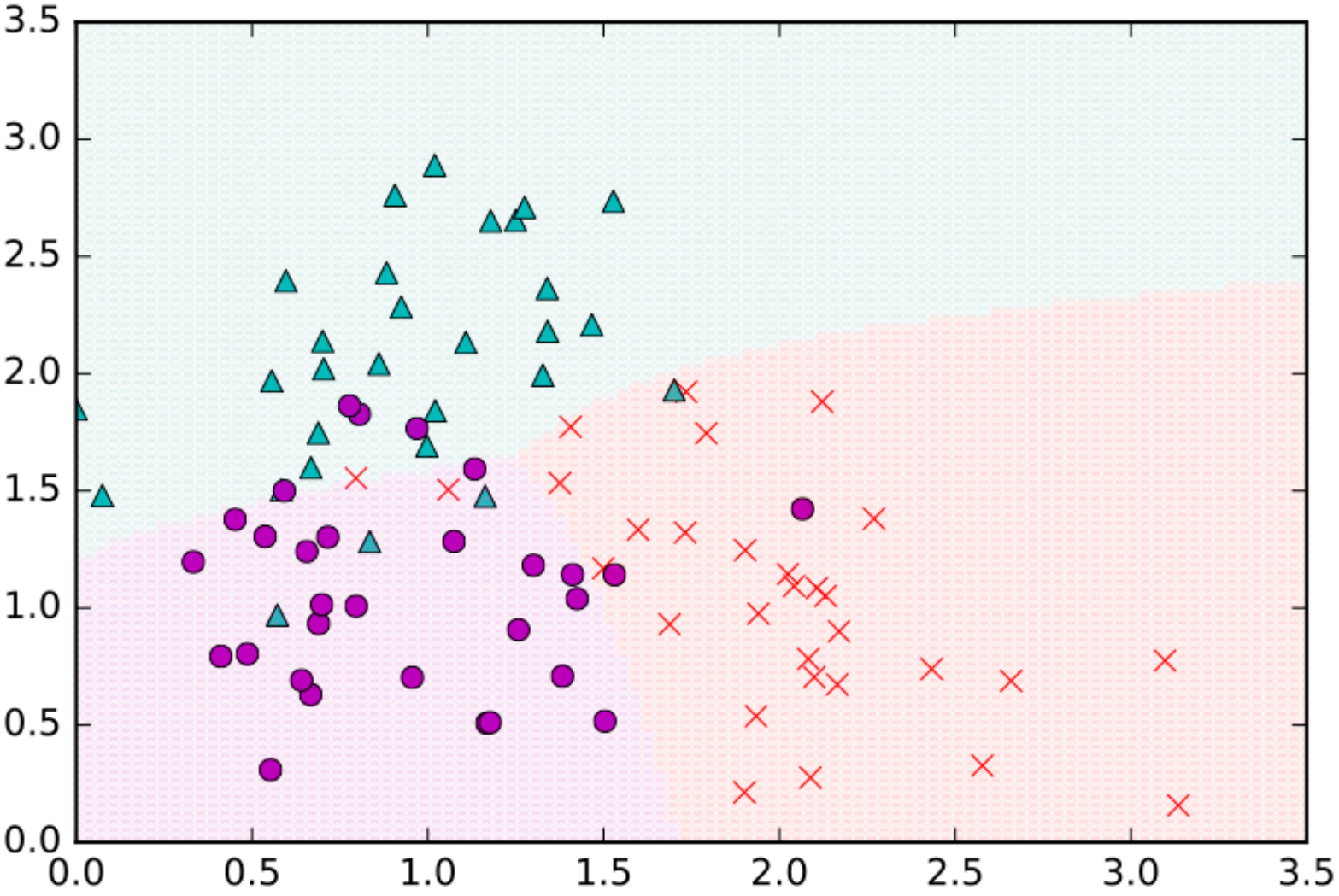
Example of ML estimation (cont.)

Imagine now a new point $\mathbf{x}$ comes in that we'd like to classify as being in one of three classes. To do so, we'd like to calculate: $\Pr(y = j|\mathbf{x})$ and pick the j that maximizes this probability. We'll denote this probability $p(j|\mathbf{x})$. We'll also assume, as earlier, that the classes are equally probable, i.e., $\Pr(y = j)$ is the same for all j.

Maximum-likelihood estimation

```
f = plt.figure()
ax = f.gca()
ax.plot(data1[0,:], data1[1,:], marker='x', linestyle='', color=class_colors[0])
ax.plot(data2[0,:], data2[1,:], marker='^', linestyle='', color=class_colors[1])
ax.plot(data3[0,:], data3[1,:], marker='o', linestyle='', color=class_colors[2])

for (i,xx) in enumerate(np.linspace(0, 3.5, 100)):
    for yy in np.linspace(0, 3.5, 100):
        xn = np.array((xx,yy))
        pmax = -np.Inf
        decoded_class = np.nan
        for c in np.arange(len(class_idxs)):
            class_prob = -np.log(np.linalg.det(covs[c])) - (xn-means[c]).T.dot(np.linalg.inv(covs[c])).dot(xn-means[c])
            if class_prob > pmax:
                pmax = class_prob
                decoded_class = c
        ax.plot(xx, yy, marker='.', color=class_colors[decoded_class], alpha=0.05)
ax.set_xlim(0,3.5)
f.savefig('ml-basics_maxlik-data_fit-ellipsoids_colorized.pdf')
```

Maximum-likelihood estimation



Where we go from here

Even more cost functions

There are even more cost functions that we could use. We'll encounter some others in this class. Others include:

- MAP estimation.
- KL divergence.
- Maximize an approximation of the distribution.

In machine learning, it is important to arrive at an appropriate model and cost function. After that, it's important to know how to optimize it. In our examples here, the models were simple enough that we could differentiate and set the derivative equal to zero. In future lectures, we'll discuss more general ways to learn parameters of models when they are not so simple.