Distributed Optimization for Machine Learning

Lecture 2 - Class Survey and Machine Learning Basics

Tianyi Chen

School of Electrical and Computer Engineering
Cornell Tech, Cornell University

August 27, 2025

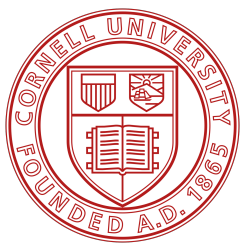
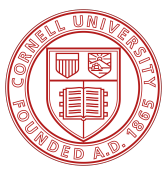


Table of Contents

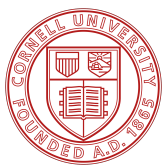


Active participation of course survey

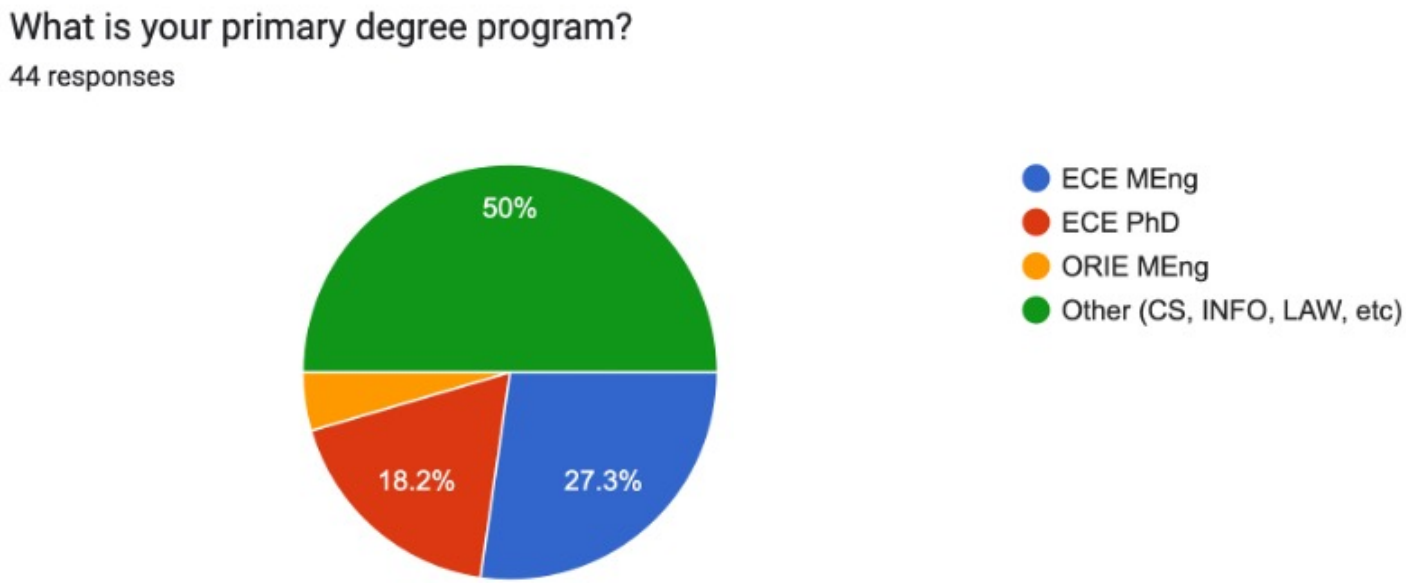
Thank you for filling out the course survey!

99% participation!

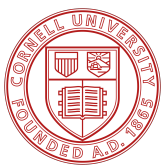
It gives me a great snapshot of who is in the room with us today.



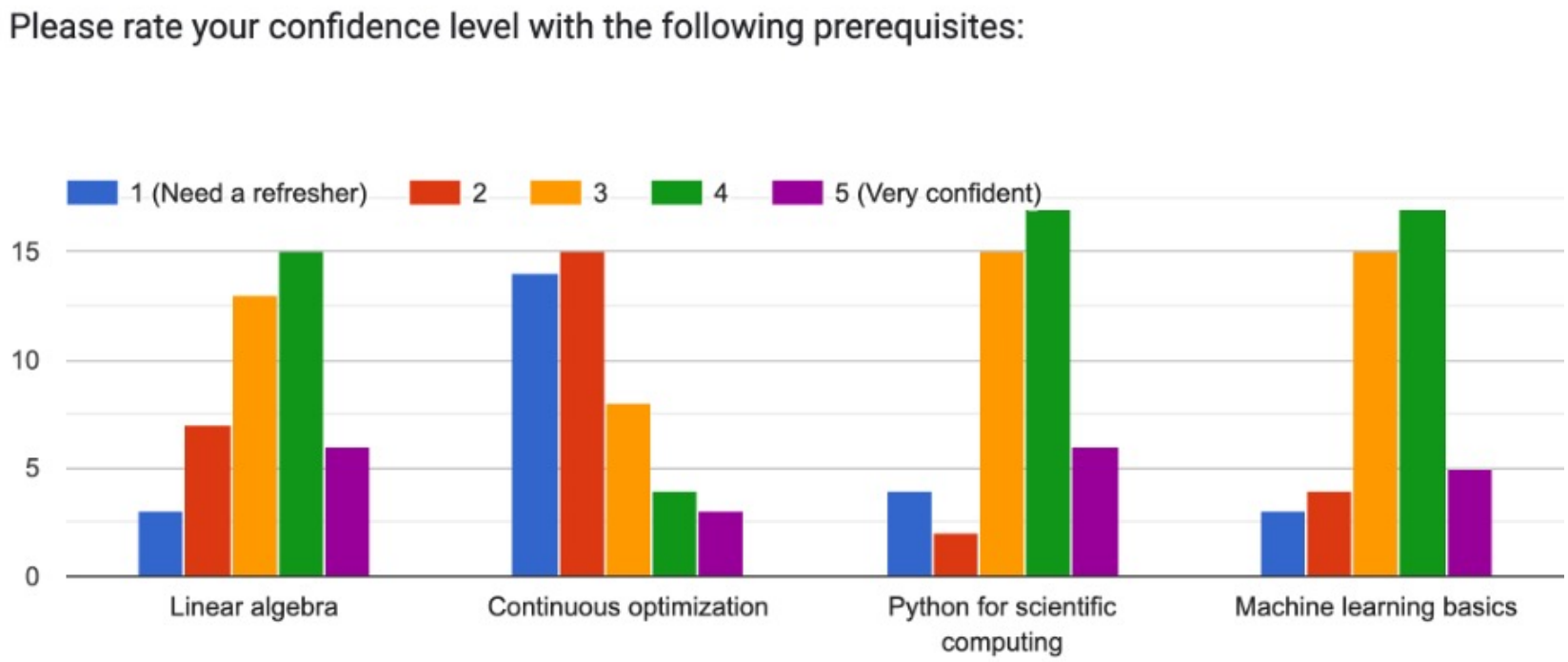
Who you are: A profile of our class



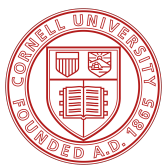
We have diverse backgrounds, with many from ECE PhD and ECE MEng, plus a strong showing from ORIE, CS and INFO.



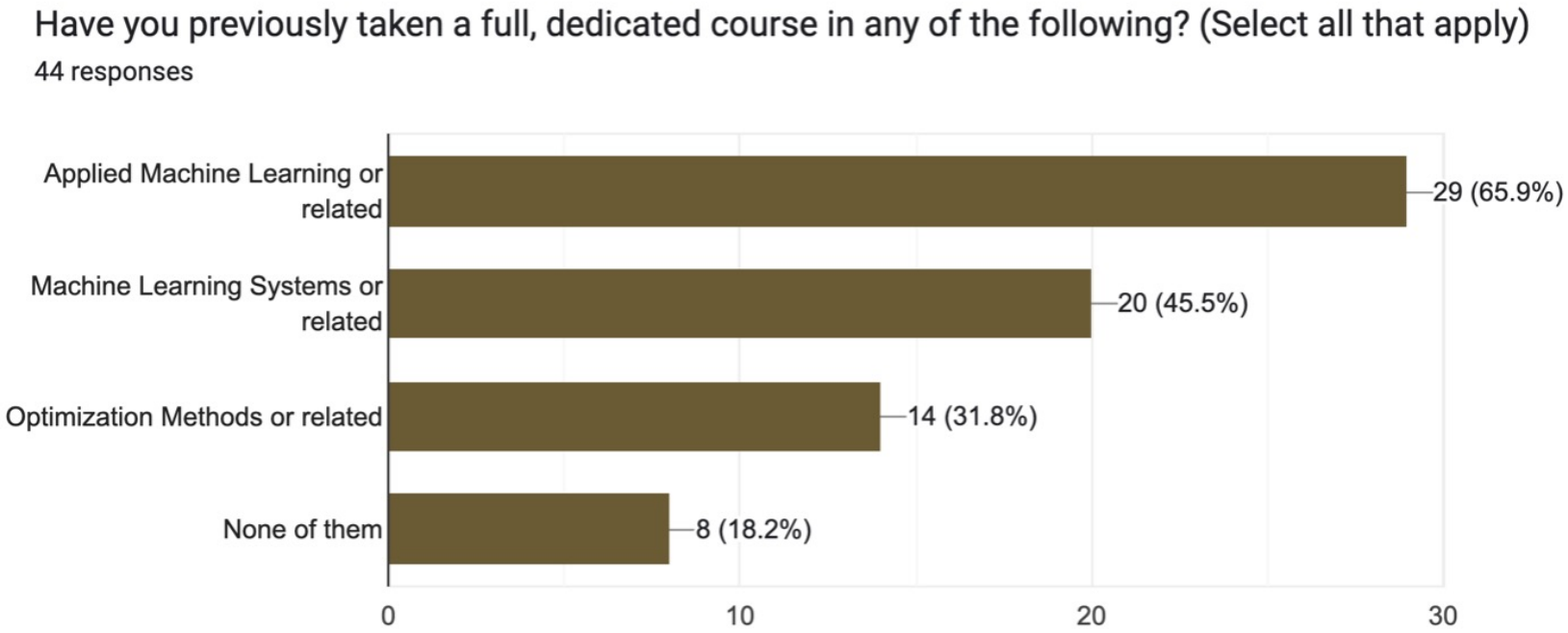
Your confidence with prerequisites



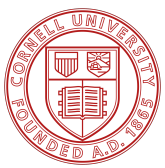
On average, the class feels most confident in Linear Algebra and Machine Learning basics. It looks like Continuous Optimization is the area where you'd most appreciate a review, which I will keep in mind.



Your relevant courses

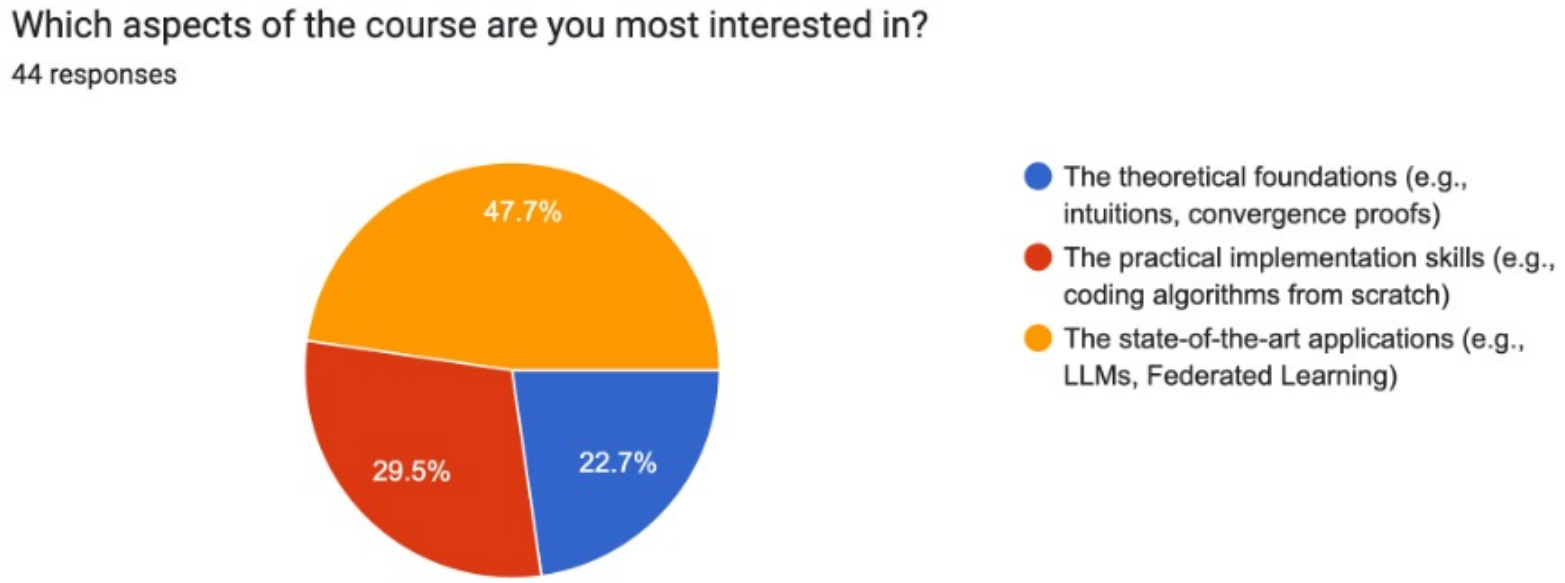


On average, the majority of class have taken Applied Machine Learning.
It looks like Optimization-related Course is less popular here.

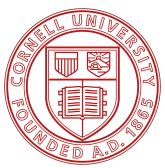


What you're interested in

Tailor the course examples to what's most relevant and exciting for you.

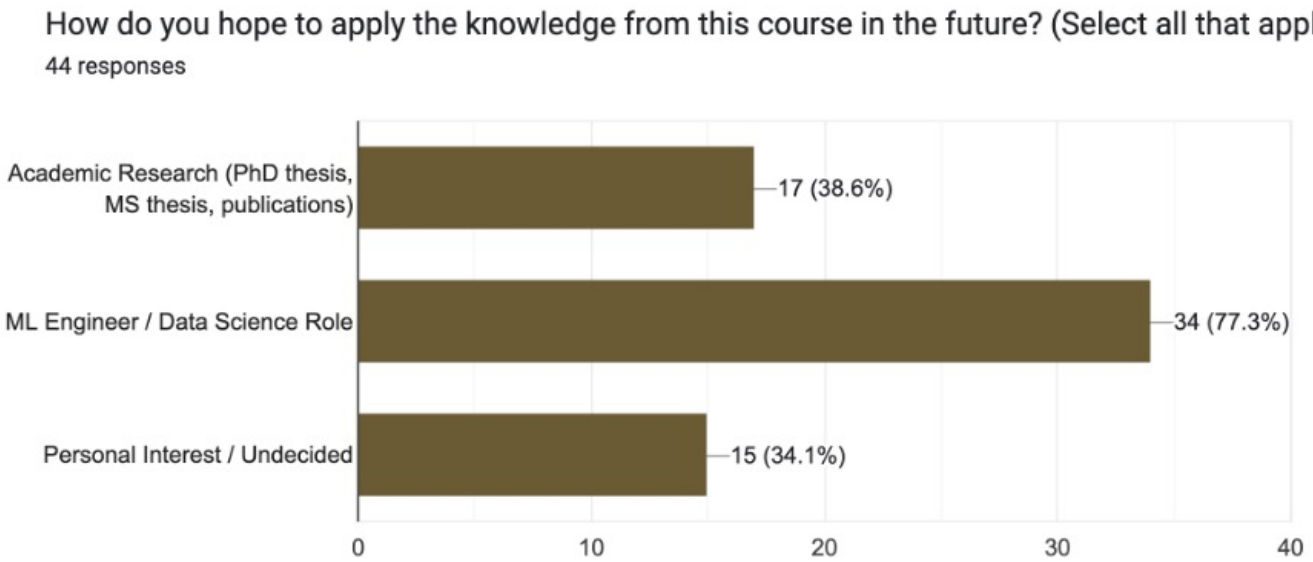


There's a very strong interest in the state-of-the-art applications (like LLMs and Federated Learning), balanced with a desire to learn both the theoretical foundations and practical implementation skills.

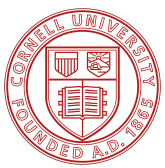


How you hope to apply this knowledge

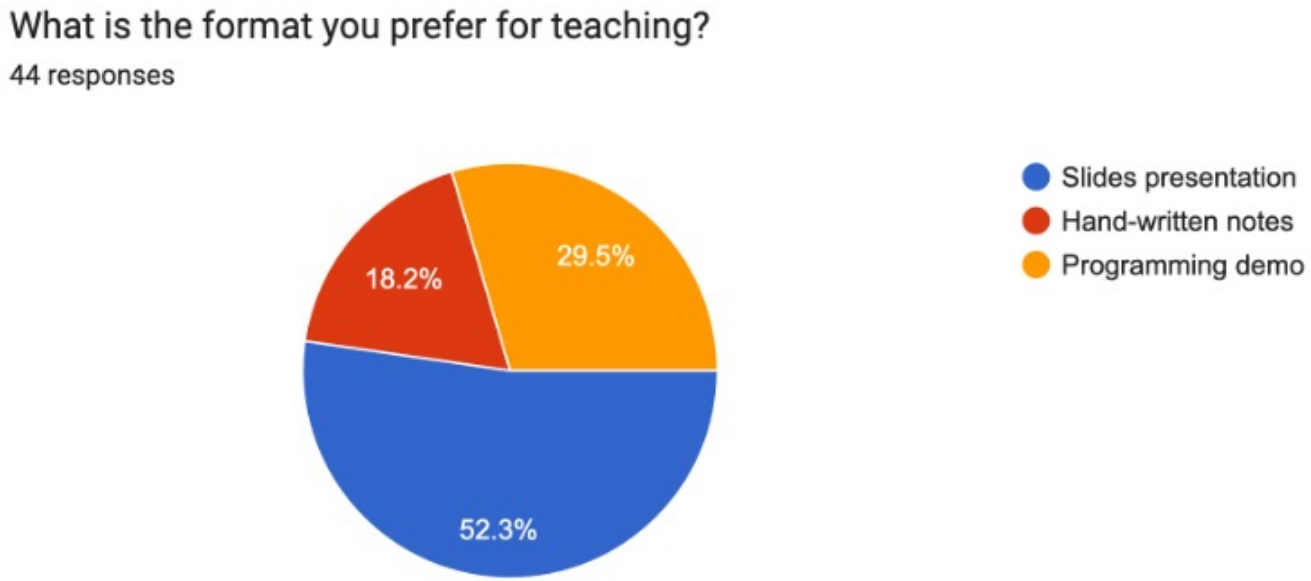
It's motivating to see your future goals, as I can connect the material directly to your potential career paths.



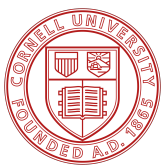
Many of you are planning to apply this knowledge in an ML Engineer/Data Science role or in Academic Research. The skills you'll learn are directly applicable to both industry and academia.



What is you preferred the presentation format?

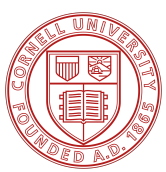


Indeed, the presentation format will follow this partition!



Effective learning for a math-heavy class

- **Clear Explanations & Examples:** Step-by-step explanations, specific examples, case studies, and demonstrations.
- **Problem Solving & Exercises:** Practice problems, problem sets, and guided exercises with examples and solutions.
- **Practical Application:** Hands-on practice, coding, projects, implementation, and real-world demos.
- **Engaging and Interactive Learning:** Class activities, discussions, and student presentations to promote engagement.



Any specific questions about the course?

- If there a lot of overlapping with several ML courses in this semester?
- Would I need background knowledge of hardware for this course?
- What libraries in python and matlab that we should acquaint ourselves with for optimization?

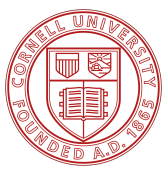
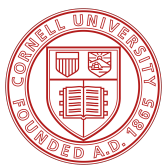


Table of Contents



Calibrate "optimization" from engineers and theorists

The engineer's view

Goal: Get a high-performing model quickly using real-world resources.

```
[41]  1 net = UNet(12)
      2 net = net.to(device)
      3 optimizer = optim.Adam(net.parameters(), lr = 8e-5)
      4 if train_epoch != None:
      5     checkpoint = torch.load(checkpoint_dir+'/0150-model.ckpt')
      6     net.load_state_dict(checkpoint['model_state_dict'])
      7     optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
      8     train_epoch = checkpoint['epoch']
      9     loss = checkpoint['loss']

      1 loss

      tensor(0.0400, device='cuda:0', requires_grad=True)
```

The theorist's view

Goal: Understand the fundamental performance of an algorithm.

1. Lower bound for D_k with a Bregman term.

For convex L-smooth f, the Bregman divergence obeys

$$\frac{1}{2L}\|\nabla f(x) - \nabla f(y)\|^2 \;\le\; f(x) - f(y) - \langle \nabla f(y), x - y\rangle$$

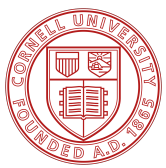
(see the inequality used repeatedly in §3, Eq. (3) / Theorem 2.1.5 of Nesterov as cited there). Applying it with $x = x_k$, $y = x_{k+1}$ and noting $x_k - x_{k+1} = \eta g_k$ gives

$$D_k \;\ge\; \eta\,\langle g_{k+1}, g_k\rangle \;+\; \frac{1}{2L}\|\Delta_k\|^2. \quad \text{(A)}$$

2. Upper bound for D_{k+1} by convexity.

By convexity, $f(x) - f(y) \le \langle \nabla f(x), x - y\rangle$, so with $x = x_{k+1}$, $y = x_{k+2}$ and $x_{k+1} - x_{k+2} = \eta g_{k+1}$,

$$D_{k+1} \;\le\; \eta\,\|g_{k+1}\|^2. \quad \text{(B)}$$

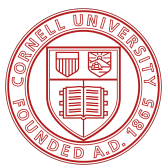


Going back to our "old-school" ML task

Machine learning (ML) learns pattern from historical data. Assume an old-school ML task - model **House Price** based on **Square Footage**.

Square Footage	**Price ($1000s)**
800	150
1200	250
1500	280
2000	350
2400	450
3000	500

- **Feature (x):** Input variable for predictions (e.g., *square footage*).
- **Label (y):** The "answer" or output we want to predict (e.g., *price*).
- **Training Example:** A single row of data, like (1200 sq. ft., $250k).



Linear models for House Price predictions

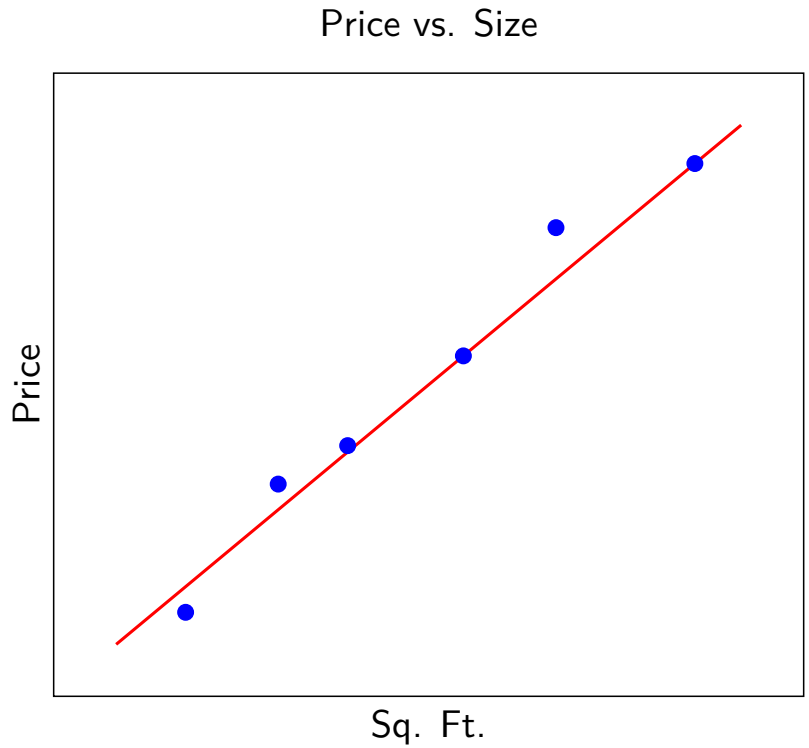
Our ML task is to model **House Price** based on **Square Footage**.

Square Footage $\rightarrow x$

Price $\rightarrow y$

Base Price $\rightarrow \theta_0$

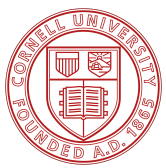
Price per SqFt $\rightarrow \theta_1$



Our Model:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Linear model $h_\theta(x)$ predicts the price via base price plus square footage.



Multiple features for House Price predictions

Let's make our price predictor more realistic by adding more features.

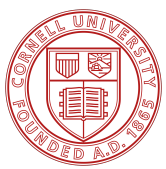
Size (sq. ft.)	**# Bedrooms**	**Age (years)**	**Price ($k)**
1200	3	10	250
2000	4	5	350
800	2	25	150

Feature vector $\mathbf{x}$ and parameter vector $\boldsymbol{\theta}$ now have multiple dimensions:

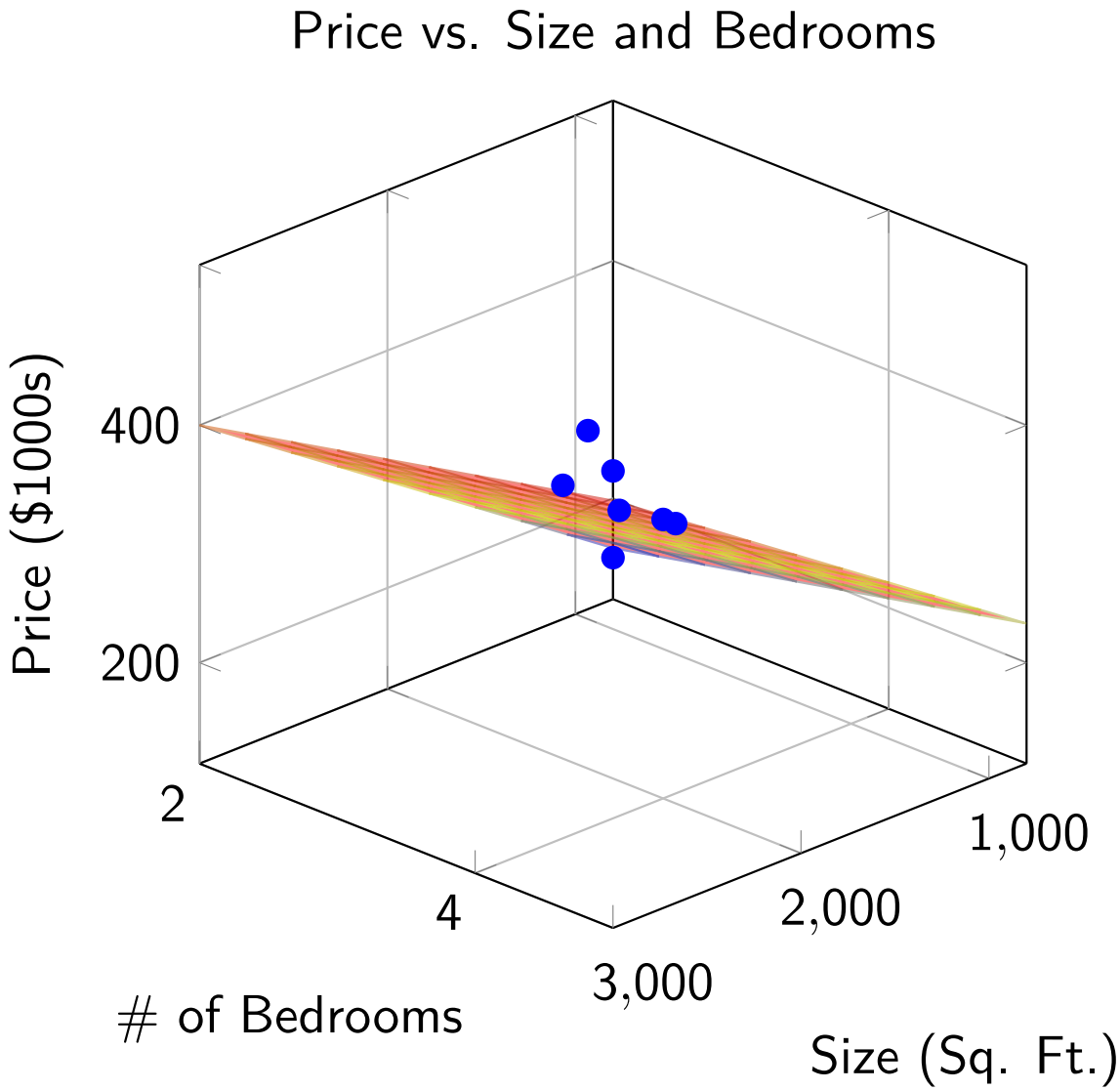
$$\mathbf{x} = \begin{pmatrix} x_1 \text{ (size)} \\ x_2 \text{ (beds)} \\ x_3 \text{ (age)} \end{pmatrix}, \quad \boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}$$

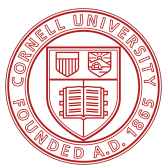
The model becomes a weighted sum of these features (Assuming $x_0 = 1$):

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 = \boldsymbol{\theta}^T \mathbf{x}$$

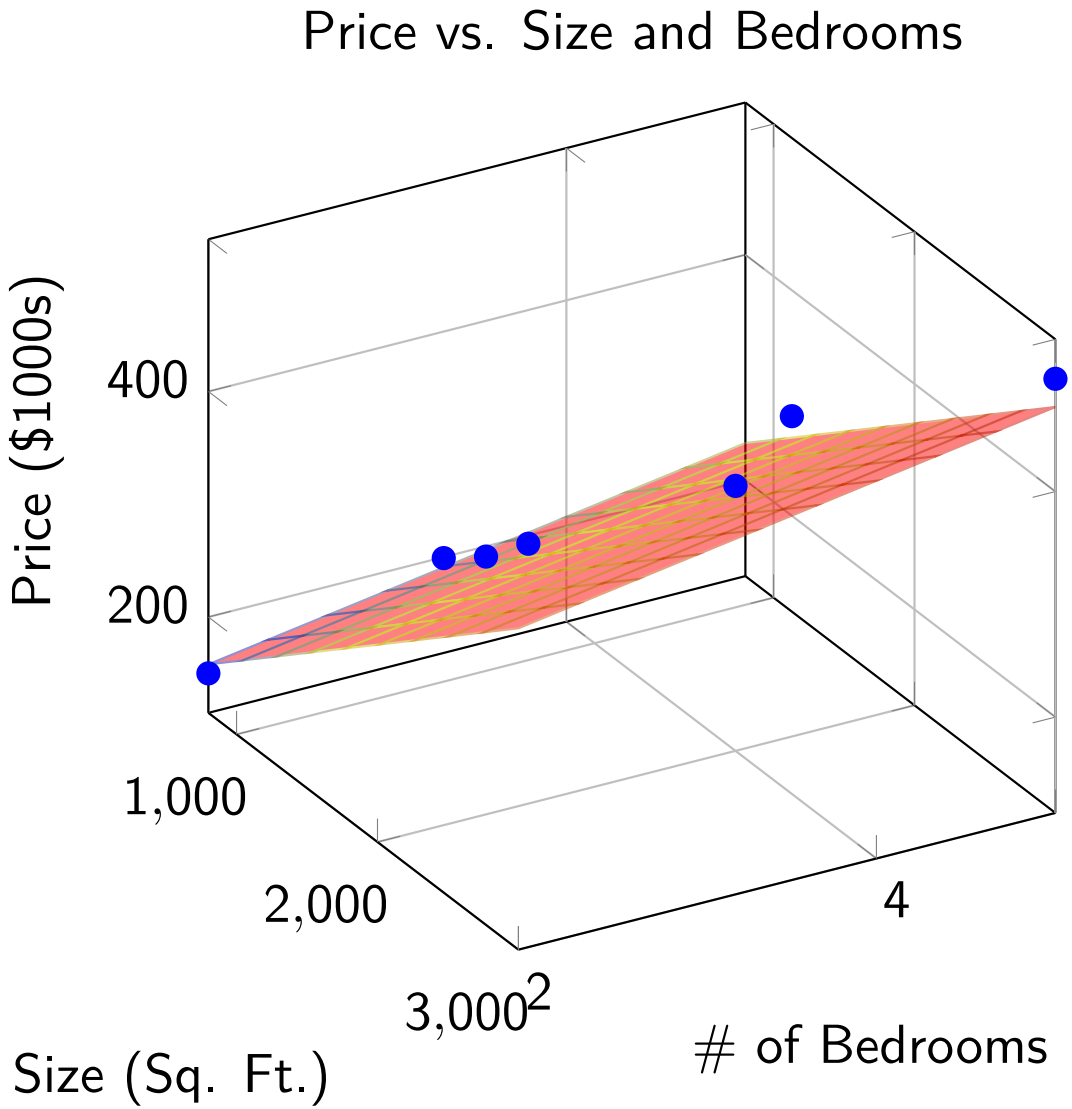


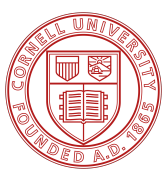
Lines become hyperplanes





Lines become hyperplanes





Training linear models for predictions

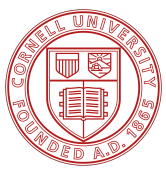
Our goal is to find the best parameter vector $\boldsymbol{\theta}$ by minimizing the **Mean Squared Error (MSE)** loss function.

$$L(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

where the model is a weighted sum of these features (Assuming $x_0 = 1$):

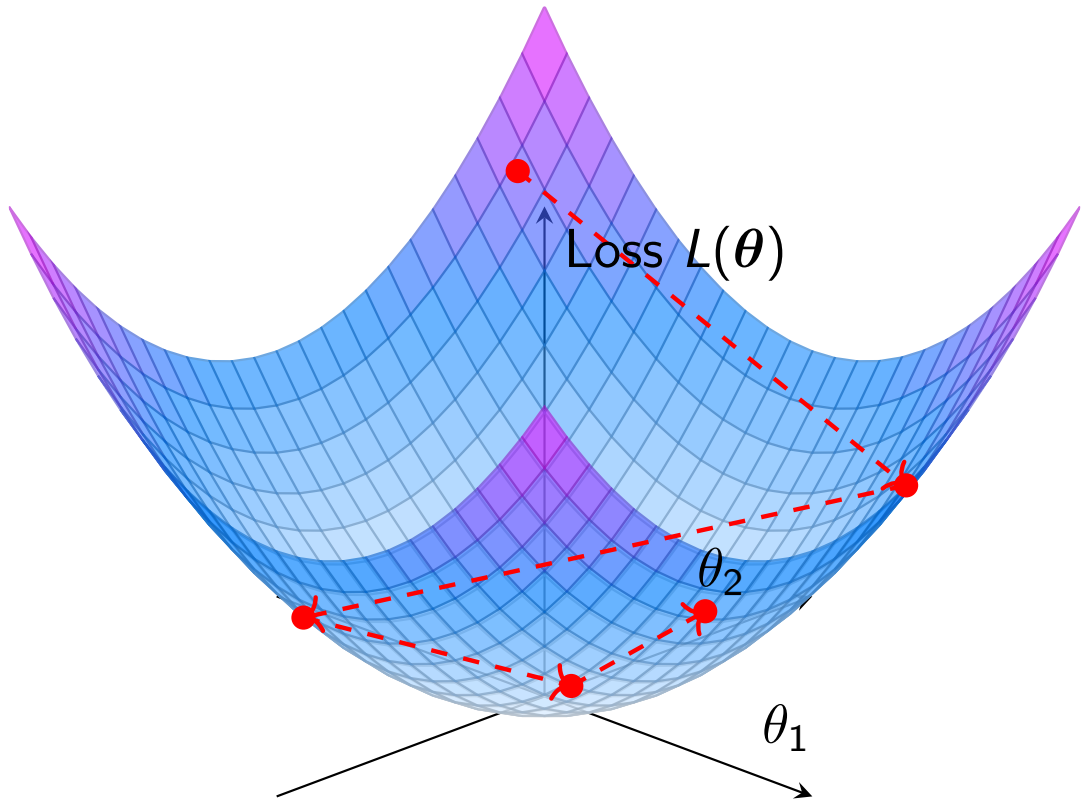
$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 = \boldsymbol{\theta}^T \mathbf{x}$$

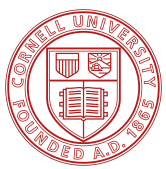
How do we minimize this? PyTorch Optimizer?



Trial-and-Error?

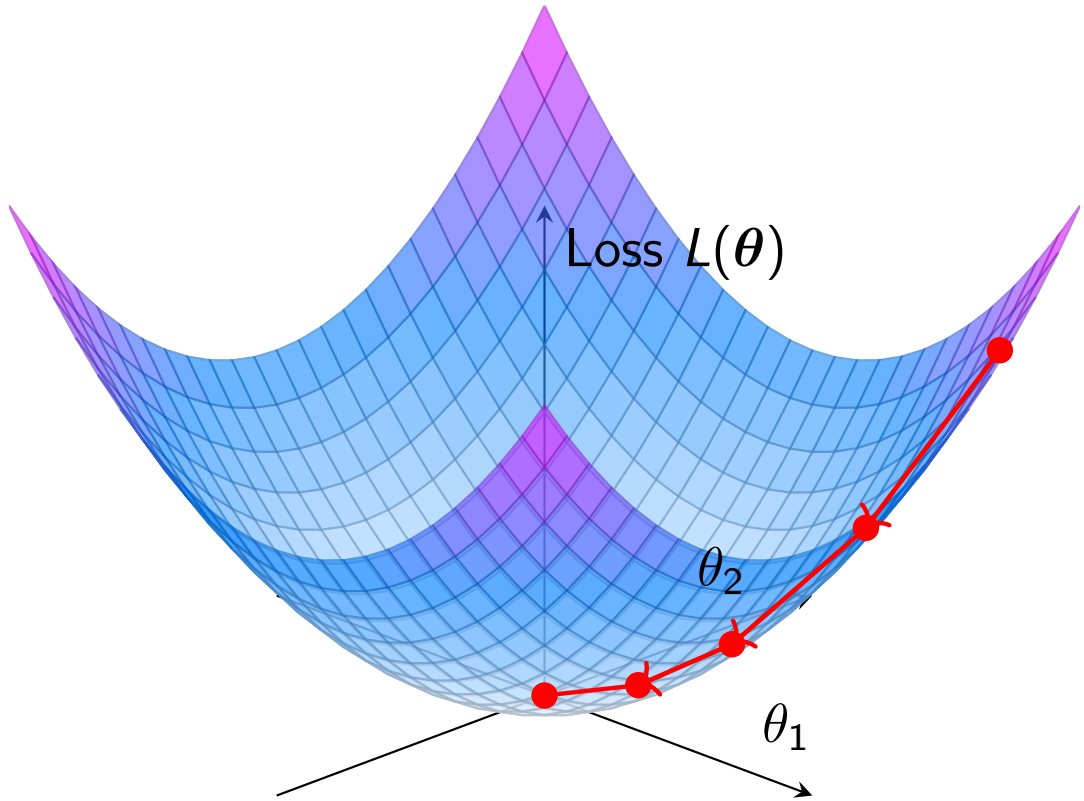
How about we could try random guesses and then compare their loss values? It works, but that's inefficient.

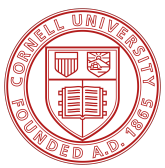




Gradient Descent: Walking downhill

We use an algorithm that "walks downhill" on the loss surface. The direction of steepest descent is given by the **gradient** $\nabla L(\boldsymbol{\theta})$.



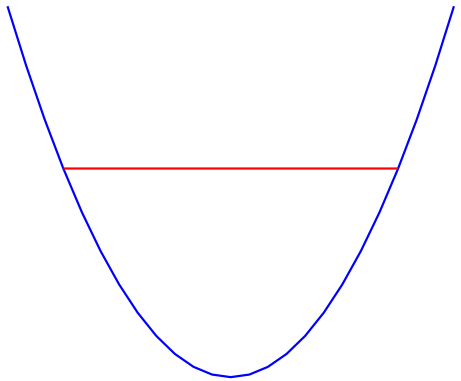


An Important Property: Convexity

Geometric intuition:

- A function is **convex** if the line segment connecting any two points on its graph lies *on or above* the graph.
- Think of it as a "bowl" shape.

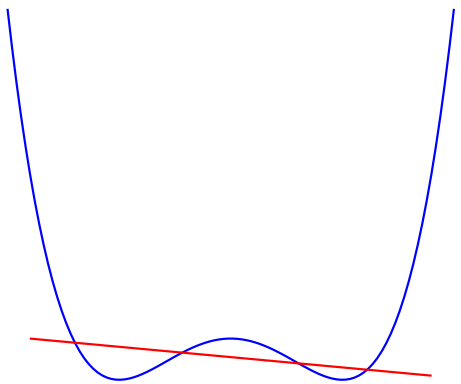
Convex

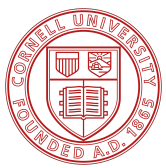


Good news for walk downhill:

- A convex function has no "bad" local minima, and we are guaranteed to find the single best solution!

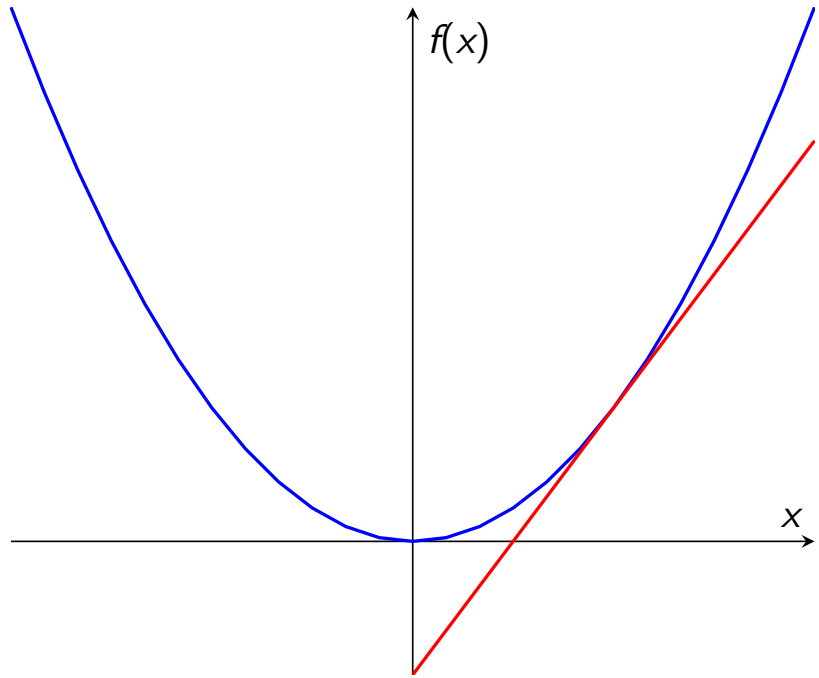
Non-Convex

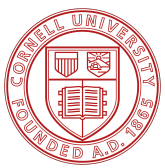




Math Review: The Derivative

- The derivative of a function $f(x)$, written as $f'(x)$, measures its instantaneous **rate of change** or **slope** at a point.
- It tells us how the output of the function will change if we make a tiny change to its input.
- **Example:** For $f(x) = x^2$, the derivative is $f'(x) = 2x$.





Math Review: The Second Derivative

- The second derivative, written as $f''(x)$, is the derivative of the first derivative.
- Measure **rate of change of the slope**. Or, the function's **curvature**.
- If $f''(x) > 0$, the slope is increasing, and the function is curving **upwards** (convex, like a bowl).
- If $f''(x) < 0$, the slope is decreasing, and the function is curving **downwards** (concave, like a dome).

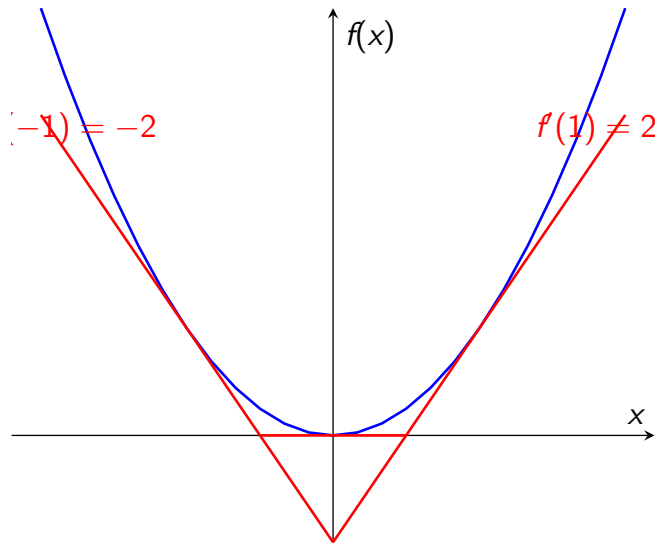
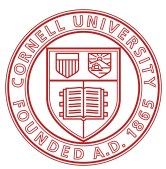


Figure: For $f(x) = x^2$, the slope increases from negative to positive. As the slope is always increasing, $f''(x) > 0$, $f(x)$ curves upwards.

How about the vector-input function?



Math Review: Vectors & Matrices

Vectors

- An ordered list of numbers.
- In ML, represents a single data point (feature vector) or the model's parameters.

Matrices

- A rectangular array of numbers.
- For now, represents the entire dataset, where each row is a training example.

Feature Vector

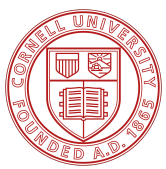
$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

Parameter Vector

$$\boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{pmatrix}$$

Dataset Matrix

$$\mathbf{X} = \begin{pmatrix} \leftarrow & (\mathbf{x}^{(1)})^T & \rightarrow \\ \leftarrow & (\mathbf{x}^{(2)})^T & \rightarrow \\ & \vdots & \\ \leftarrow & (\mathbf{x}^{(m)})^T & \rightarrow \end{pmatrix}$$



Math Review: Partial Derivatives & The Gradient

Partial Derivatives

- For a function with multiple inputs, a partial derivative ($\frac{\partial f}{\partial x_j}$) is the derivative with respect to *one* variable, treating others as constants.
- **Example:** For $f(x, y) = 3x^2 + 2y$:

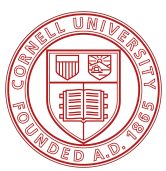
$$\frac{\partial f}{\partial x} = 6x, \quad \frac{\partial f}{\partial y} = 2$$

The Gradient (∇f)

- The **gradient** is a vector containing *all* of the partial derivatives

$$\nabla f = \begin{pmatrix} \partial f/\partial x \\ \partial f/\partial y \end{pmatrix} = \begin{pmatrix} 6x \\ 2 \end{pmatrix}$$

- **Key Insight:** The gradient vector ∇f always points in the direction of the **steepest ascent** of the function.

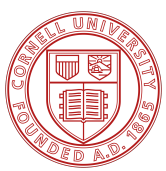


Math Review: Jacobian and Hessian

1. **The Jacobian Matrix (J)**
   - The matrix of all first-order partial derivatives of a **vector-valued function** ($f: \mathbb{R}^n \to \mathbb{R}^m$). It generalizes the gradient.
   - **Example:** For $f(x, y) = \begin{pmatrix} x^2 y \\ 5x + \sin(y) \end{pmatrix}$, the Jacobian is:

$$\mathbf{J} = \begin{pmatrix} \partial f_1/\partial x & \partial f_1/\partial y \\ \partial f_2/\partial x & \partial f_2/\partial y \end{pmatrix} = \begin{pmatrix} 2xy & x^2 \\ 5 & \cos(y) \end{pmatrix}$$

2. **The Hessian Matrix (H or $\nabla^2 f$) - Jacobian of the gradient**
   - The matrix of all second-order partial derivatives of a **scalar-valued function** ($f: \mathbb{R}^n \to \mathbb{R}$) - the multi-variable version of 2nd derivative.
   - **Key Insight:** The Hessian describes the local **curvature** of a function. If the function looks like a bowl, a dome, or a saddle point.



Math Review: Positive Semi-Definite (PSD) Matrices

This is a crucial property of a symmetric matrix (like the Hessian) that formalizes the "bowl shape."

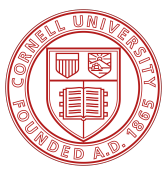
Definition: A matrix $\mathbf{H}$ is Positive Semi-Definite (PSD) if for any non-zero vector $\mathbf{v}$, the following holds:

$$\mathbf{v}^T\mathbf{H}\mathbf{v} \geq 0$$

Intuition: This means that from any point on a function's surface, the curvature is *never downwards* in any direction. Either flat or curving up.

Connection to Convexity

If the Hessian matrix of a function is Positive Semi-Definite everywhere, then the function is **convex**.



Is the linear regression loss convex?

For a twice-differentiable function $f(\boldsymbol{\theta})$, it is convex if and only if its Hessian matrix $\nabla^2 f(\boldsymbol{\theta})$ is **positive semi-definite (PSD)**.

1. **Simplify:** Only need to show convexity for a single data point's loss.

$$L_i(\boldsymbol{\theta}) = (\boldsymbol{\theta}^T \mathbf{x} - y)^2$$

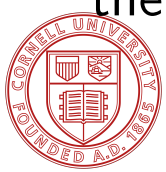
2. **Calculate the Gradient** and **the Hessian**:

$$\nabla L_i(\boldsymbol{\theta}) = 2(\boldsymbol{\theta}^T \mathbf{x} - y)\mathbf{x}; \quad \nabla^2 L_i(\boldsymbol{\theta}) = 2\mathbf{x}\mathbf{x}^T$$

3. **Check if the Hessian is PSD:** For any vector $\mathbf{v} \in \mathbb{R}^d$:

$$\mathbf{v}^T(2\mathbf{x}\mathbf{x}^T)\mathbf{v} = 2(\mathbf{v}^T\mathbf{x})(\mathbf{x}^T\mathbf{v}) = 2(\mathbf{v}^T\mathbf{x})^2 \geq 0$$

Since $(\mathbf{v}^T\mathbf{x})^2$ is a squared number, it is always non-negative. Therefore, the Hessian is positive semi-definite, and the MSE loss is **convex**.

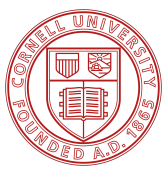


Training Step 1: The Gradient

For our linear model with MSE loss, the gradient component for a single parameter θ_j is:

$$\frac{\partial}{\partial \theta_j} L(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

This tells us how much the error changes with respect to a small change in that specific parameter (e.g., the weight for 'size' or 'age').

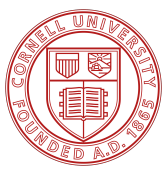


Training Step 2: The Update Rule

Gradient Descent works by taking small, iterative steps. In each step, we update every parameter θ_j using the following rule:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} L(\boldsymbol{\theta})$$

- θ_j is the parameter's current value.
- α is **learning rate**, a small number (e.g., 0.01) controls the stepsize.
- $\frac{\partial}{\partial \theta_j} L(\boldsymbol{\theta})$ is the gradient component on the previous slide.



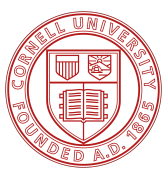
The Full Gradient Descent Algorithm

The complete Gradient Descent algorithm is:

1. Initialize parameters $\boldsymbol{\theta}$ (e.g., to all zeros).
2. Repeat for many iterations:
   - Compute the gradient $\frac{\partial}{\partial \theta_j} L(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$
   - Update parameter θ_j using the rule:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

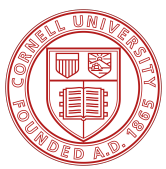
   - Repeat this for all parameters $(\theta_0, \theta_1, \theta_2, \theta_3)$ simultaneously.
3. Stop until some stopping criteria are satisfied.



Gradient Descent: Iteration 1 (Setup)

Size (sq. ft.)	**# Bedrooms**	**Age (years)**	**Price ($k)**
x_1	x_2	x_3	y
1200	3	10	250
2000	4	5	350
800	2	25	150

1. **Initial State:**
   - Learning Rate $\alpha = 10^{-7}$ (a tiny number due to large feature values).
   - Initial Parameters $\boldsymbol{\theta}^{(0)} = (0, 0, 0, 0)$.



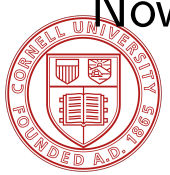
Gradient Descent: Iteration 1 (Setup)

Size (sq. ft.) x_1	**# Bedrooms** x_2	**Age (years)** x_3	**Price ($k)** y
1200	3	10	250
2000	4	5	350
800	2	25	150

2. Calculate predictions and errors for all training examples:
Since our initial parameters are all zero, the first prediction is $h_{\boldsymbol{\theta}}(\mathbf{x}) = 0$.

House	**Prediction (h)**	**Actual (y)**	**Error ($h - y$)**
1	0	250	-250
2	0	350	-350
3	0	150	-150

Now we use these errors to compute the gradient for the entire batch.

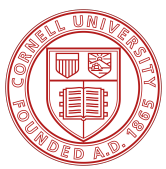


Gradient Descent: Iteration 1 (Calculating the gradient)

3. Calculate the gradient via the formula $\frac{1}{m}\sum(\text{error}^{(i)}) \cdot x_j^{(i)}$ for each j.

- **For θ_0 (bias, $x_0 = 1$):**
  $\frac{1}{3}[(-250 \cdot 1) + (-350 \cdot 1) + (-150 \cdot 1)] = -250$
- **For θ_1 (size, x_1):**
  $\frac{1}{3}[(-250 \cdot 1200) + (-350 \cdot 2000) + (-150 \cdot 800)] = -373,333$
- **For θ_2 (beds, x_2):**
  $\frac{1}{3}[(-250 \cdot 3) + (-350 \cdot 4) + (-150 \cdot 2)] = -817$
- **For θ_3 (age, x_3):**
  $\frac{1}{3}[(-250 \cdot 10) + (-350 \cdot 5) + (-150 \cdot 25)] = -2,667$

The full gradient vector is $\nabla L(\boldsymbol{\theta}^{(0)}) = (-250, -373333, -817, -2667)$.



Gradient Descent: Iteration 1 (The update)

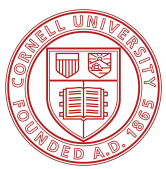
4. Update the parameters using the rule $\boldsymbol{\theta}^{(\text{new})} = \boldsymbol{\theta}^{(\text{old})} - \alpha \nabla L(\boldsymbol{\theta}^{(\text{old})})$.

$$\begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} - (10^{-7}) \cdot \begin{pmatrix} -250 \\ -373,333 \\ -817 \\ -2,667 \end{pmatrix}$$

The new parameter vector after one step is:

$$\boldsymbol{\theta}^{(1)} = \begin{pmatrix} 0.000025 \\ 0.037333 \\ 0.000082 \\ 0.000267 \end{pmatrix}$$

For Iteration 2, we would repeat this entire process starting with $\boldsymbol{\theta}^{(1)}$.



Gradient Descent: Iteration 2 (New predictions & loss)

1. **Starting Point:** Begin with the parameters from the first iteration:

$$\boldsymbol{\theta}^{(1)} = (0.000025, 0.0373, 0.000082, 0.000267)$$

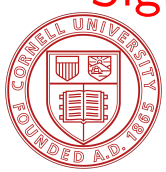
2. **Calculate new predictions and errors:**

House	Prediction (h)	Actual (y)	Error ($h - y$)
1 (1200 sqft)	44.80	250	-205.20
2 (2000 sqft)	74.67	350	-275.33
3 (800 sqft)	29.87	150	-120.13

3. **Calculate the new loss:** Using the new errors, the MSE is:

$$L(\boldsymbol{\theta}^{(1)}) = \frac{1}{2 \cdot 3} \left((-205.20)^2 + (-275.33)^2 + (-120.13)^2 \right) \approx 22,057$$

Significant Improvement! Loss at Iteration 0: $L(\boldsymbol{\theta}^{(0)}) = 34,583$



Gradient Descent: Iteration 2 (New gradient & update)

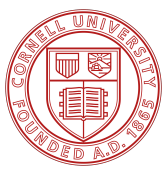
4. **Calculate the new gradient** using the errors from the previous step.
   The new gradient is $\nabla L(\boldsymbol{\theta}^{(1)}) = (-200, -297,640, -665, -2,125)$.

(Note: gradients are smaller than before, (as expected?) as the error is lower).

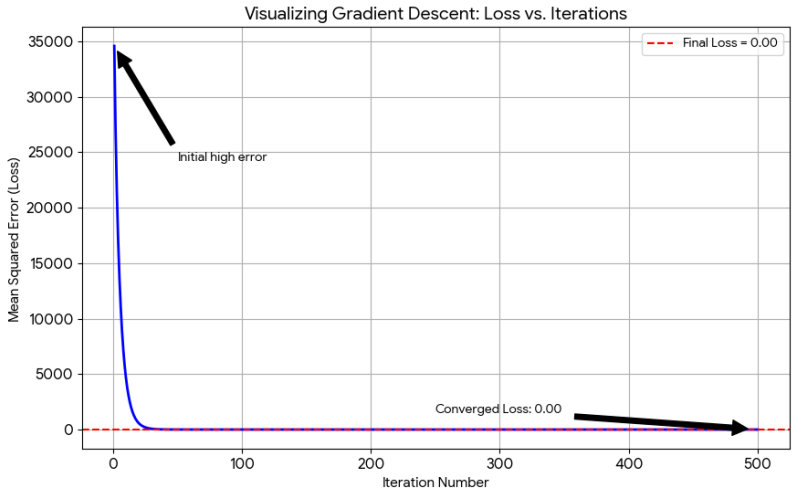
5. **Perform the Second Update** to get $\boldsymbol{\theta}^{(2)} = \boldsymbol{\theta}^{(1)} - \alpha \nabla L(\boldsymbol{\theta}^{(1)})$:

$$\boldsymbol{\theta}^{(2)} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}^{(2)} = \begin{pmatrix} 0.000025 \\ 0.0373 \\ 0.000082 \\ 0.000267 \end{pmatrix} - (10^{-7}) \cdot \begin{pmatrix} -200 \\ -297,640 \\ -665 \\ -2,125 \end{pmatrix} = \begin{pmatrix} 0.000045 \\ 0.0671 \\ 0.000148 \\ 0.000479 \end{pmatrix}$$

The model continues this process, with each step making a smaller, more refined adjustment to the parameters until the loss converges.



The quality of a converged model



After running hundreds of iterations, the gradient descent algorithm converges.

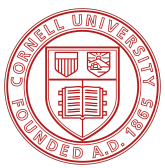
Final Learned Parameters (θ):

- θ_0 (Base Price): 250.0
- θ_1 (for Size): 84.3
- θ_2 (for Beds): 21.2
- θ_3 (for Age): -22.9

Final loss (MSE): 73.55

Final Model Predictions:

Actual	Predicted	Error
\$250k	\$257k	-7k
\$350k	\$343k	+7k
\$150k	\$150k	0



Recap and fine-tuning

- What we have talked about today?
  - $\Rightarrow$ How to solve linear regression via gradient descent?
  - $\Rightarrow$ How to extended to nonlinear regression?
  - $\Rightarrow$ How about other loss functions?

Welcome anonymous survey!