2657119

slido

Join at slido.com #2657119

Click **Present with Slido** or install our Chrome extension to display joining instructions for participants while presenting.

2657119

LECTURE 14

Gradient Descent - Part 2 & Feature Engineering

Building models in code. Transforming data to improve model performance.

Data 100/Data 200, Spring 2025 @ UC Berkeley

Narges Norouzi and Josh Grossman

Content credit: Acknowledgments

Midterm Announcements

Please read the Midterm Logistics post for full midterm logistics.

- **Midterm is next Wednesday, March 12th from 8-10 PM.**

Midterm Logistics!

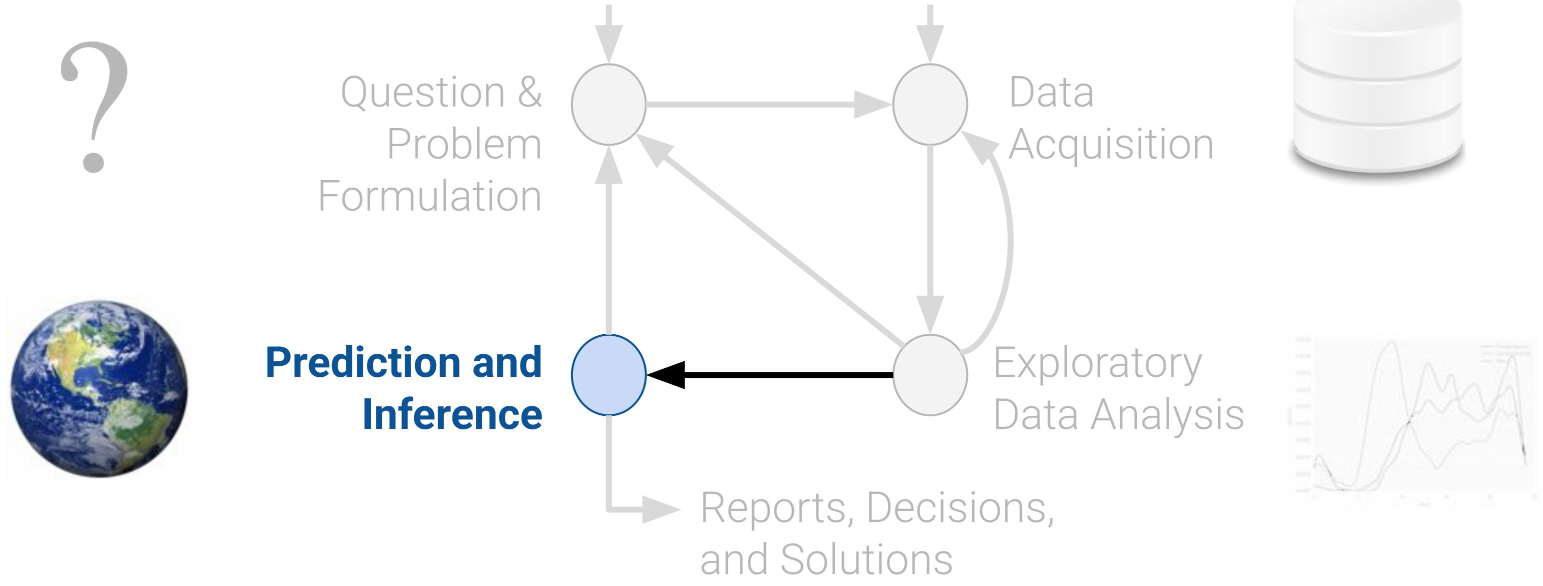
- **Midterm review session is Friday, 3/7 in Stanley 105 from 4:00 - 5:30 PM.**
  - Seats available on a first-come, first-serve basis. (approx 290 seats)
  - **Recording** and **slides** will be **posted on Ed afterwards**

- **Seating assignments** for the in-person midterm will be sent out next **Monday, 3/10**.

- **No lecture next Tuesday, March 11th!**

Model Implementation

2657119



Model Implementation I:
scikit-learn
Gradient Descent I

Model Implementation II:
Gradient descent II
Feature Engineering

Gradient descent on multi-dimensional models

Lecture 14, Data 100 Spring 2025

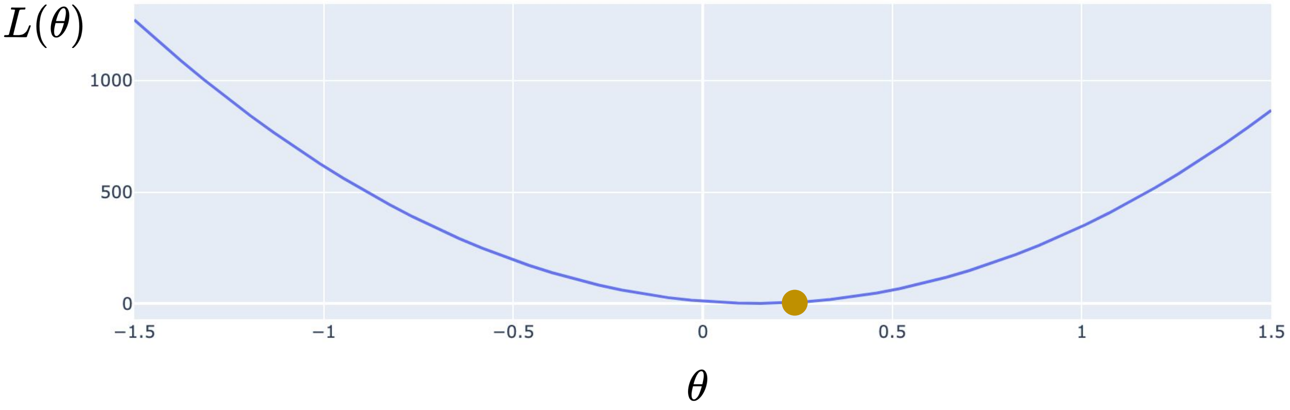
2657119

- **Gradient descent on multi-dimensional models**
- Batch, mini-batch, and stochastic gradient descent
- Feature Engineering
    - One-Hot Encoding
    - Polynomial Features
    - Complexity and Overfitting

Last Lecture: Optimization on Loss Functions

Recall: When modeling, we aim to identify the model parameters that minimize a *loss function*. 2657119

Goal: choose the value of θ that minimizes $L(\theta)$, the model's **loss** on the dataset



Last Lecture: From Arbitrary Functions to Loss Functions

2657119

Goal: Choose the value of θ that minimizes $L(\theta)$, the model's loss on the dataset

Gradient: The direction of steepest ascent for a function at some specific input.

The **1D gradient descent** algorithm:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \frac{d}{d\theta} L\left(\theta^{(t)}\right)$$

Last Lecture: Gradient Descent in Multiple Dimensions

2657119

Recall our 1D update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \frac{d}{d\theta} L\left(\theta^{(t)}\right)$$

Now, for models with multiple parameters, we work in terms of vectors:

$$\begin{bmatrix} \theta_0^{(t+1)} \\ \theta_1^{(t+1)} \\ \vdots \end{bmatrix} = \begin{bmatrix} \theta_0^{(t)} \\ \theta_1^{(t)} \\ \vdots \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \\ \vdots \end{bmatrix}$$

Written in a more compact form:

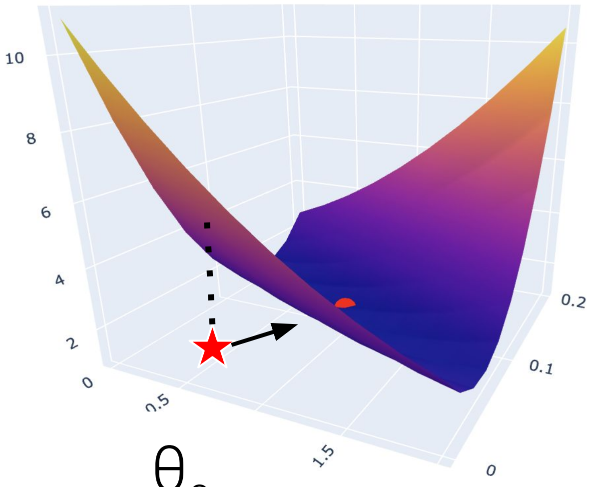
$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right)$$

Last Lecture: The Gradient Vector

Before, the derivative of the loss function guided us towards the minimizing parameter value.

On a 3D (or higher) loss surface, the **gradient** is described by a **vector** of partial derivatives.

L(θ)



θ1

θ0

For the *vector* of parameter values:

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

Find the *partial derivative* of loss with respect to each parameter:

$$\nabla_{\vec{\theta}} L(\theta_0, \theta_1) = \begin{bmatrix} \frac{\partial}{\partial \theta_0} L(\theta_0, \theta_1) \\ \frac{\partial}{\partial \theta_1} L(\theta_0, \theta_1) \end{bmatrix}$$

$-\nabla_{\vec{\theta}} L$ always points towards the **downhill direction** of the surface.

Note: The 2D gradient is not tangent to the 3D loss surface!

Suppose the gradient of the loss function $L(\theta_0, \theta_1)$ at a particular combination of θ_0 and θ_1 is [-3 2]. Which of the following is true?

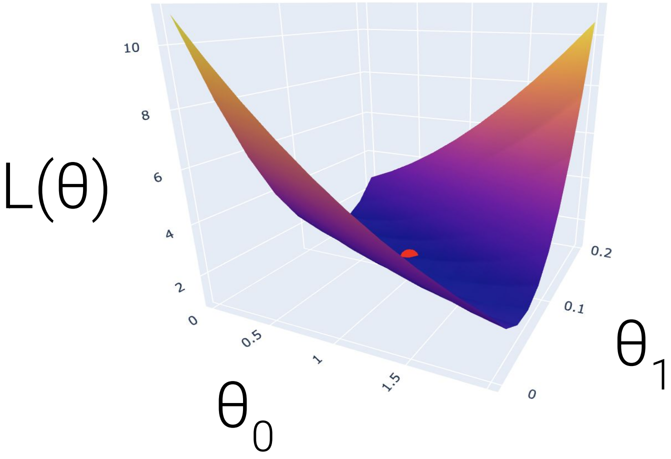
Last Lecture: How to Interpret Gradients

2657119

Element 1: If I were to barely increase θ_0, how does loss change?

Element 2: If I were to barely increase θ_1, how does loss change?

$$\nabla_{\vec{\theta}} L(\theta_0, \theta_1) = \begin{bmatrix} \frac{\partial}{\partial \theta_0} L(\theta_0, \theta_1) \\ \frac{\partial}{\partial \theta_1} L(\theta_0, \theta_1) \end{bmatrix}$$



Putting It All Together

2657119

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1$$

$$l\left(y, \hat{y}\right) = \left(y - \hat{y}\right)^2$$

$$L\left(\vec{\theta}\right) = \frac{1}{n}\sum_{i=1}^{n} l\left(y_i, \hat{y}_i\right)$$

Model+loss

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)$$

$$\nabla_{\vec{\theta}} l\left(\vec{\theta}^{(t)}, \vec{x}_i, y_i\right) = \text{???}$$

Compute the gradient at each time step

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right)$$

$$\begin{bmatrix} \theta_0^{(t+1)} \\ \theta_1^{(t+1)} \end{bmatrix} = \begin{bmatrix} \theta_0^{(t)} \\ \theta_1^{(t)} \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \end{bmatrix}$$

Gradient descent update rule

You Try:

2657119

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1 \qquad l\left(y, \hat{y}\right) = \left(y - \hat{y}\right)^2$$

$$l\left(\vec{\theta}, \vec{x}_i, y_i\right) =$$

$$\frac{\partial}{\partial \theta_0} l\left(\vec{\theta}, \vec{x}_i, y_i\right) =$$

$$\frac{\partial}{\partial \theta_1} l\left(\vec{\theta}, \vec{x}_i, y_i\right) =$$

$$\nabla_{\vec{\theta}} l\left(\vec{\theta}, \vec{x}_i, y_i\right) =$$

You Try:

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1 \qquad l\left(y, \hat{y}\right) = \left(y - \hat{y}\right)^2$$

$$l\left(\vec{\theta}, \vec{x}_i, y_i\right) = \left(y_i - \theta_0 x_{i0} - \theta_1 x_{i1}\right)^2$$

$$\frac{\partial}{\partial \theta_0} l\left(\vec{\theta}, \vec{x}_i, y_i\right) = 2\left(y_i - \theta_0 x_{i0} - \theta_1 x_{i1}\right)\left(-x_{i0}\right)$$

$$\frac{\partial}{\partial \theta_1} l\left(\vec{\theta}, \vec{x}_i, y_i\right) = 2\left(y_i - \theta_0 x_{i0} - \theta_1 x_{i1}\right)\left(-x_{i1}\right)$$

$$\nabla_{\vec{\theta}} l\left(\vec{\theta}, \vec{x}_i, y_i\right) = \begin{bmatrix} -2\left(y_i - \theta_0 x_{i0} - \theta_1 x_{i1}\right)\left(x_{i0}\right) \\ -2\left(y_i - \theta_0 x_{i0} - \theta_1 x_{i1}\right)\left(x_{i1}\right) \end{bmatrix}$$

2657119

Putting it all together

2657119

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1$$

$$l(y, \hat{y}) = (y - \hat{y})^2$$

$$L\left(\vec{\theta}\right) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, \hat{y}_i)$$

Model+loss

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\vec{\theta}} l(y_i, \hat{y}_i)$$

$$\nabla_{\vec{\theta}} l\left(\vec{\theta}^{(t)}, \vec{x}_i, y_i\right) = \begin{bmatrix} -2\left(y_i - \theta_0^{(t)} x_{i0} - \theta_1^{(t)} x_{i1}\right)(x_{i0}) \\ -2\left(y_i - \theta_0^{(t)} x_{i0} - \theta_1^{(t)} x_{i1}\right)(x_{i1}) \end{bmatrix}$$

Compute the gradient at each time step

See skipped slides for derivation! Nothing new.

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right)$$

$$\begin{bmatrix} \theta_0^{(t+1)} \\ \theta_1^{(t+1)} \end{bmatrix} = \begin{bmatrix} \theta_0^{(t)} \\ \theta_1^{(t)} \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \end{bmatrix}$$

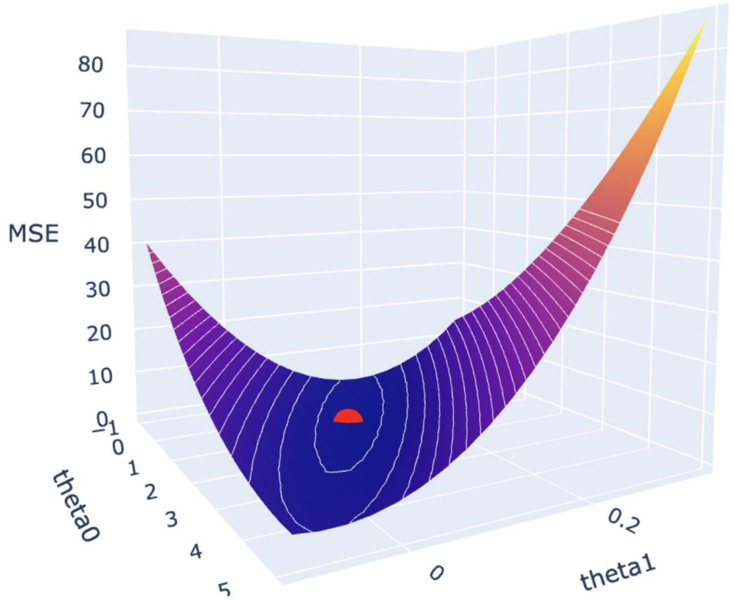
Gradient descent update rule

Demo Slides

lec13.ipynb

Gradient Descent with 3D Loss Surface

2657119



Similar Local Minima Concerns with Non-Convex 3D Loss Surfaces!

2657119

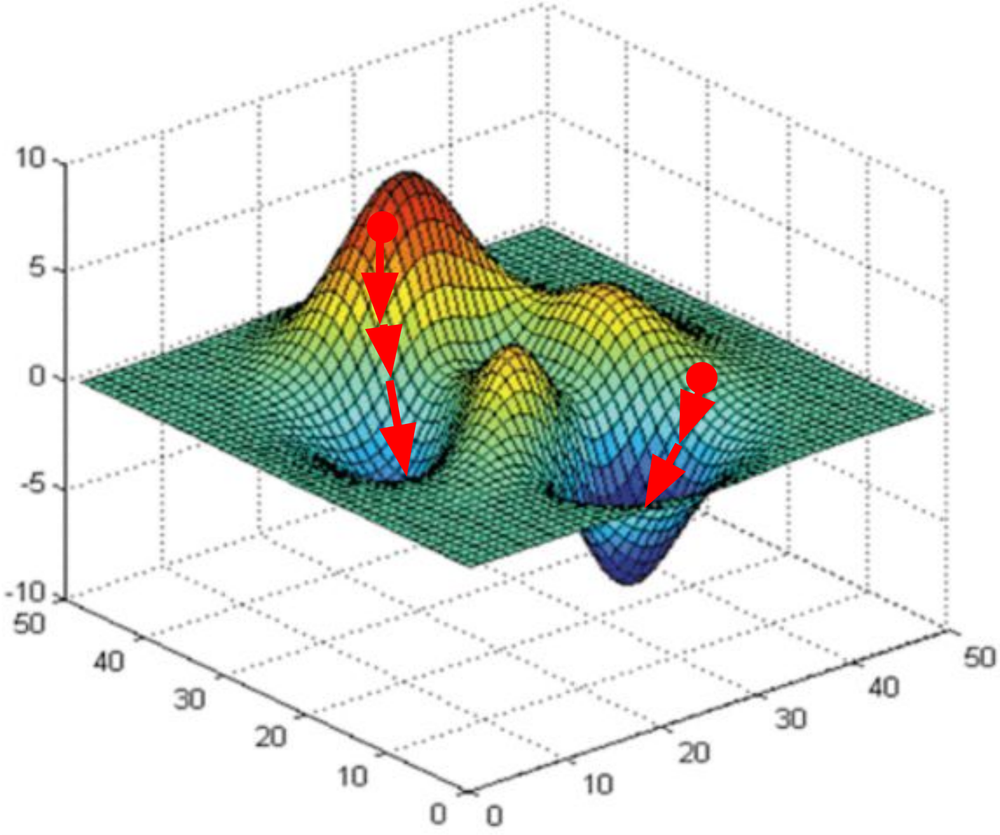


Image Credit: Narges Norouzi

Mini-batch Gradient Descent

Lecture 14, Data 100 Spring 2025

- Gradient descent on multi-dimensional models
- **Batch, mini-batch, and stochastic gradient descent**
- Feature Engineering
  - One-Hot Encoding
  - Polynomial Features
  - Complexity and Overfitting

2657119

Complexity of Gradient Descent

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \underbrace{\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right)}$$

n loss computations for **d** parameters

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)}_{O(nd)}$$

If we run *T* for iterations, then the final complexity is O(Tnd)

- Typically, n is the largest term, by far!

Can we **reduce the cost** of this algorithm using a technique from Data 100?

Epochs and Updates

Epoch: One pass through <u>all</u> n data points while updating the parameters.

For example: We can calculate the exact gradient with all n data points, as before:

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)$$

Then, we could make one update to the parameter vector:

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right)$$

This is **one epoch** of gradient descent with **one update** to the parameter vector.

Epochs and Updates

Dataset with **n** data points

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)$$

One *exact* update to the parameters per **one** epoch of gradient descent.

Dataset **randomly** split into **two** pieces

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) \approx \frac{1}{n/2}\sum_{i=1}^{n/2} \nabla_{\vec{\theta}}\, l\left(y_i, \hat{y}_i\right) \qquad \nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t+1)}\right) \approx \frac{1}{n/2}\sum_{i=1}^{n/2} \nabla_{\vec{\theta}}\, l\left(y_i, \hat{y}_i\right)$$

Two *approximate* updates per **one** epoch! A free lunch?

Mini-batch Gradient Descent: *Sampling the Gradient!*

2657119

True Gradient: $\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)$ O(nd)

We can use a **random sample** to **approximate the gradient**!

"Mini-batch Gradient"

Sample *b* records: $\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) \approx \underbrace{\frac{1}{b}\sum_{i=1}^{b} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)}_{O(bd)}$

Big b: More accurate updates.
Small b: Faster updates.
Tradeoff of accuracy + speed.

Comparing Gradient Descent Techniques

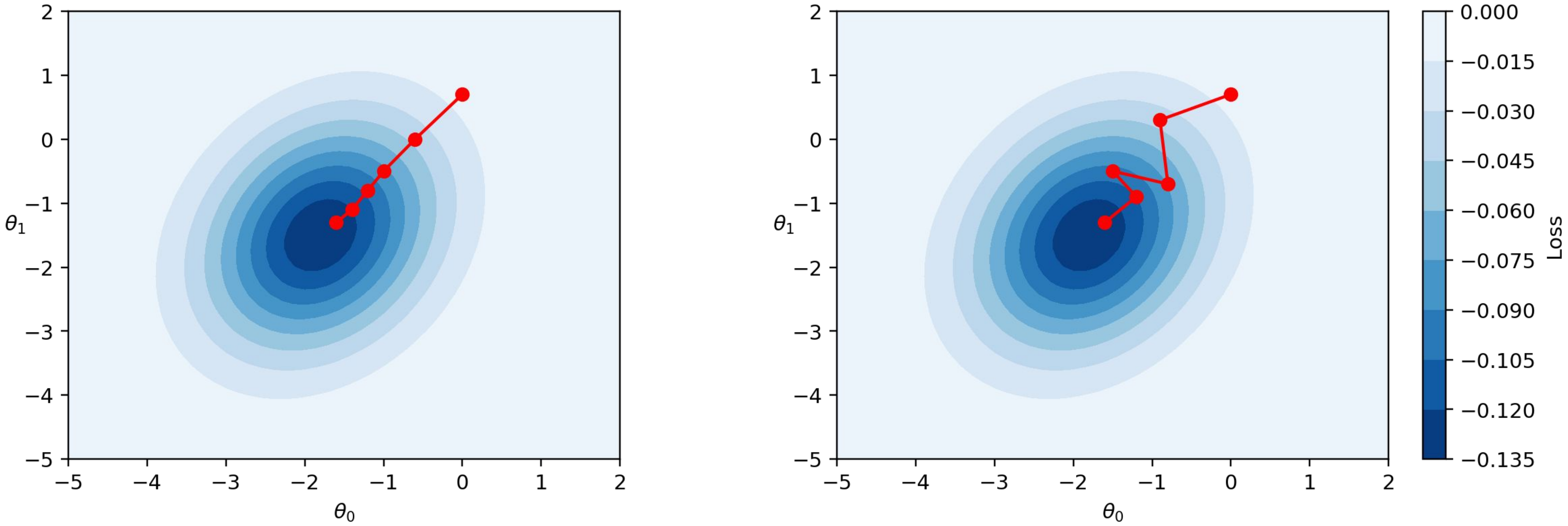
2657119

(Batch) **Gradient Descent:**

- Computes the **true gradient**
- Always descends towards the true minimum of the loss

Mini-batch **Gradient Descent:**

- ***Approximates*** the true gradient
- May not descend towards the true minimum with each update



2657119

In general, which of these is the least computationally expensive?

Computational Expense of Gradient descent

In general, which of these is the least computationally expensive?

A. One update of gradient descent.
B. One update of mini-batch gradient descent.
C. They are about the same.

One update of mini-batch gradient descent requires us to average the gradient of loss of **b** data points to approximate the true gradient. **b < # of data points**

Computation required for one update:

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) \approx \frac{1}{b} \sum_{i=1}^{b} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right) \qquad \vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right)$$

Gradient descent requires loss computations for all **n** data points.

2657119

In general, which of these is the least computationally expensive?

Computational Expense of Gradient descent

2657119

In general, which of these is the least computationally expensive?

A. One epoch of gradient descent updates.
B. One epoch of mini-batch gradient descent updates.
C. They are about the same.

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) \approx \frac{1}{b} \sum_{i=1}^{b} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)$$

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right)$$

Solution: Both A and B require computation of loss for all **n** data points.

However, one epoch of mini-batch gradient descent consists of **n / b** updates to the parameters, while gradient descent consists of just **one** update.

Within a single update, the gradient of loss on individual data points can be computed in **parallel**. The order of these computations does not matter—we will just average them.

But, multiple updates cannot be computed in parallel; they must be done one after the other (i.e., **serially**). Each parameter update depends on the previous update!

2657119

In general, which of these will get you closer to a minimum of a function?

Efficiency of Gradient Descent

2657119

In general, which of these will get you closer to a minimum of a function?

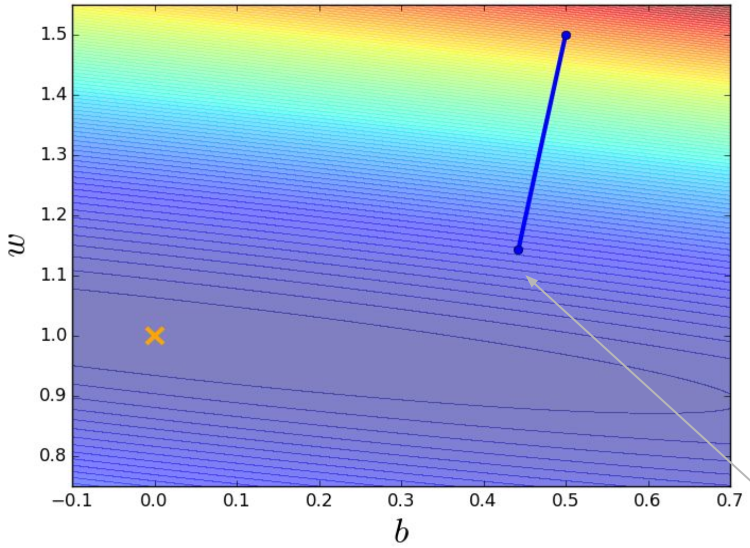
A. One epoch of gradient descent updates.
B. One epoch of mini-batch gradient descent updates.
C. They are about the same.

Solution: Even though each **update** of mini-batch gradient descent is only approximate, mini-batch gradient descent is more efficient **per epoch**. See next slide!

Mini-batch Gradient Descent is More Efficient Per Epoch

2657119

Gradient Descent
One Epoch, **One** *Exact* Update



Mini-batch Gradient Descent
One Epoch, **Many** *Approximate* Updates

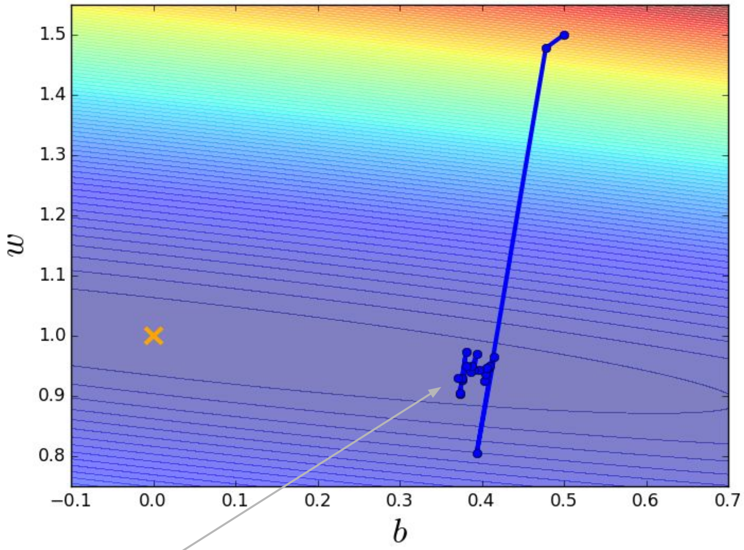


Image Credit: Narges Norouzi

One epoch of mini-batch GD gets us closer to the minimum!

Taking Mini-batch Gradient Descent to the Extreme

2657119

Dataset with **n** data points

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)$$

One *exact* update to the parameters per **one** epoch of gradient descent.

Dataset **randomly** shuffled and split into **n** pieces

. . . . .

$$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) \approx \nabla_{\vec{\theta}} l\left(y_1, \hat{y}_1\right) \quad \nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t+1)}\right) \approx \nabla_{\vec{\theta}} l\left(y_2, \hat{y}_2\right) \cdots\cdots \quad \nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t+n-1)}\right) \approx \nabla_{\vec{\theta}} l\left(y_n, \hat{y}_n\right)$$

Use the gradient of loss on a **single data point** to approximate the exact gradient.
n *approximate* updates per **one epoch**!

Stochastic Gradient Descent (SGD)

2657119

True Gradient: $$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)$$ O(nd)

We can use a **random sample** to **approximate the gradient**!

"Stochastic Gradient"

Sample just *1* record $$\nabla_{\vec{\theta}} L\left(\vec{\theta}^{(t)}\right) \approx \underbrace{\color{red}{\nabla_{\vec{\theta}} l\left(y_i, \hat{y}_i\right)}}_{O(d)}$$

In other words, stochastic gradient descent (SGD) is Mini-batch Gradient Descent with b=1.

2657119

Suppose we fit an Ordinary Least Squares model to a 8192 datapoint dataset using **stochastic** gradient descent. How many gradient updates are there per epoch?

Feature Engineering

Lecture 14, Data 100 Spring 2025

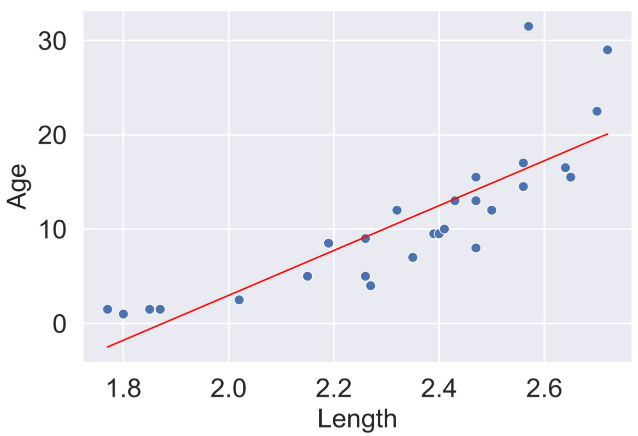
- Gradient descent on Multi-Dimensional Models
- Batch, mini-batch, and stochastic gradient descent
- **Feature Engineering**
  - One-Hot Encoding
  - Polynomial Features
  - Complexity and Overfitting

Transforming Features

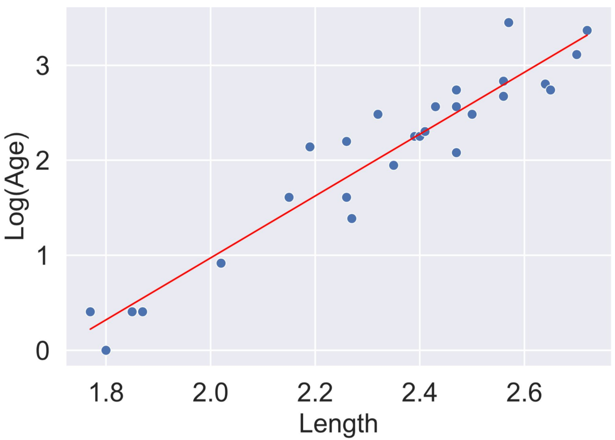
Two observations from previous lectures:

- We found that applying a transformation could help **linearize** a dataset.
- We saw that **linear modeling works best** when our dataset has **linear relationships.**

$$\widehat{\text{Age}} = \theta_0 + \theta_1 \text{Length}$$



$$\widehat{\log(\text{Age})} = \theta_0 + \theta_1 \text{Length}$$



Feature Engineering

Feature engineering is the process of **transforming raw features** into **more informative features** for use in modeling**.**

Allows us to:

- Express non-linear relationships using linear models (e.g., x → log(x))
- Use qualitative/categorical/non-numeric features in models (e.g., grade level)
- Capture domain knowledge (e.g., latitude+longitude → distance to nearest school)

Feature Functions

2657119

A **feature function** describes the transformation of raw features to transformed features.

Example: a feature function that adds a squared feature to the design matrix

	hp	mpg
0	130.00	18.00
1	165.00	15.00
2	150.00	18.00
...	...	...
395	84.00	32.00
396	79.00	28.00
397	82.00	31.00

392 rows × 2 columns

$$\Phi$$

	hp	hp^2	mpg
0	130.00	16900.00	18.00
1	165.00	27225.00	15.00
2	150.00	22500.00	18.00
...	...	...	...
395	84.00	7056.00	32.00
396	79.00	6241.00	28.00
397	82.00	6724.00	31.00

392 rows × 3 columns

Dataset of raw features:

$$\mathbb{X} \in \mathbb{R}^{n \times p}$$

After applying the feature function Φ:

$$\Phi(\mathbb{X}) \in \mathbb{R}^{n \times p'}$$

Feature Functions

2657119

A **feature function** describes the transformation of raw features to transformed features.

Linear models trained on transformed data often written using Φ ("phi") instead of X:

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2$$
$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

$$\hat{y} = \theta_0 + \theta_1 \phi_1 + \theta_2 \phi_2$$
$$\hat{\mathbb{Y}} = \Phi\theta$$

Shorthand for “the design matrix after feature engineering”

2657119

One-Hot Encoding

Lecture 14, Data 100 Spring 2025

- Gradient descent on Multi-Dimensional Models
- Batch, mini-batch, and stochastic gradient descent
- **Feature Engineering**
  - **One-Hot Encoding**
  - Polynomial Features
  - Complexity and Overfitting
- Bonus

Regression Using Non-Numeric Features

2657119

Think back to the `tips` dataset we used when first exploring regression

	total_bill	tip	sex	smoker	day
0	16.99	1.01	Female	No	Sun
1	10.34	1.66	Male	No	Sun
2	21.01	3.50	Male	No	Sun
3	23.68	3.31	Male	No	Sun
4	24.59	3.61	Female	No	Sun

Before, we were limited to only using numeric features in a model (e.g., `total_bill`)

By performing feature engineering, we can incorporate **qualitative** features like the day of the week (`day`).

One-hot Encoding

2657119

One-hot encoding is a feature engineering technique to transform **qualitative data** into numeric features for modeling

- Each category of a categorical variable gets its own feature
  - Value = 1 if a row belongs to the category
  - Value = 0 otherwise

	Original data	Sunday	Thursday	Friday	Saturday
i=1	Sunday	**1**	0	0	0
i=2	Sunday	**1**	0	0	0
i=3	Thursday	0	**1**	0	0
i=4	Friday	0	0	**1**	0
i=5	Saturday	0	0	0	**1**

One-hot encoding

Regression Using the One-Hot Encoding

2657119

The one-hot encoded features can then be used in the design matrix to train a model

	total_bill	size	day_Fri	day_Sat	day_Sun	day_Thur
0	16.99	2	0.0	0.0	1.0	0.0
1	10.34	3	0.0	0.0	1.0	0.0
2	21.01	3	0.0	0.0	1.0	0.0
3	23.68	2	0.0	0.0	1.0	0.0
4	24.59	4	0.0	0.0	1.0	0.0

Raw features

One-hot encoded features

$$\hat{y} = \theta_1(\text{total\_bill}) + \theta_2(\text{size}) + \theta_3(\text{day\_Fri}) + \theta_4(\text{day\_Sat}) + \theta_5(\text{day\_Sun}) + \theta_6(\text{day\_Thur})$$

Why is the bias column missing? Answered in a few slides!

2657119

What tip would the model predict for a party with size 3 and a total bill of $50 eating on a Friday?

Regression Using the One-Hot Encoding

Party of 3, $50 total bill, eating on a Friday:

$$\hat{y} = \theta_1(\text{total\_bill}) + \theta_2(\text{size}) + \theta_3(\text{day\_Fri}) + \theta_4(\text{day\_Sat}) + \theta_5(\text{day\_Sun}) + \theta_6(\text{day\_Thur})$$

$$\hat{y} = 0.09(50) + 0.19(3) + 0.75(1) + 0.62(0) + 0.73(0) + 0.67(0)$$

	Model Coefficient
Feature	
total_bill	0.09
size	0.19
day_Fri	0.75
day_Sat	0.62
day_Sun	0.73
day_Thur	0.67

$$\hat{y} = 5.82$$

Regression Using the One-Hot Encoding

2657119

Interpreting the `day_Fri` coefficient:

$$\hat{y} = \theta_1(\text{total\_bill}) + \theta_2(\text{size}) + \theta_3(\text{day\_Fri}) + \theta_4(\text{day\_Sat}) + \theta_5(\text{day\_Sun}) + \theta_6(\text{day\_Thur})$$

	Model Coefficient
Feature	
total_bill	0.09
size	0.19
day_Fri	0.75
day_Sat	0.62
day_Sun	0.73
day_Thur	0.67

Interpretation**:** *0.75 is the predicted tip for a $0 bill with 0 customers on Friday.*

Not a very useful interpretation!

One-hot Encode Wisely!

2657119

Any set of one-hot encoded columns will always sum to a column of all ones:

Sunday	Thursday	Friday	Saturday	Bias
1	0	0	0	1
1	0	0	0	1
0	1	0	0	1
0	0	1	0	1
0	0	0	1	1

If we also include a bias column in the design matrix, there will be **linear dependence** in the model. $\mathbb{X}^\top\mathbb{X}$ is not invertible, and our OLS estimate $\hat{\theta} = (\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\mathbb{Y}$ fails.

One-hot Encode Wisely!

2657119

How to resolve?
1. Designate a one-hot encoded column as a **reference level** and remove the column from X.
2. Do not include a bias column (i.e., **remove the intercept**).
3. Regularize. Coming in a few lectures 🙂.

Sunday	Thursday	~~Friday~~	Saturday	Bias
1	0	~~0~~	0	1
1	0	~~0~~	0	1
0	1	~~0~~	0	1
0	0	~~1~~	0	1
0	0	~~0~~	1	1

or

Sunday	Thursday	Friday	Saturday	~~Bias~~
1	0	0	0	~~1~~
1	0	0	0	~~1~~
0	1	0	0	~~1~~
0	0	1	0	~~1~~
0	0	0	1	~~1~~

No information is lost. The omitted column is a linear combination of the remaining columns.

One-Hot Encoding with Omitted Column

What happens if we omit the `day_Fri` column and include an intercept?

$$\hat{y} = \theta_0 + \theta_1(\text{ total\_bill }) + \theta_2(\text{ size }) + \theta_3(\text{ day\_Sat }) + \theta_4(\text{ day\_Sun }) + \theta_5(\text{ day\_Thur })$$

	Model Coefficient
Feature	
Intercept	0.75
total_bill	0.09
size	0.19
day_Sat	-0.12
day_Sun	-0.01
day_Thur	-0.08

Interpretation: *For a $0 bill for a party of size 0, what is the predicted tip for the **reference level** day (i.e., Friday)?*

Interpretation: *For the same bill and group size, how much larger are predicted Saturday tips, **relative to Friday**?*

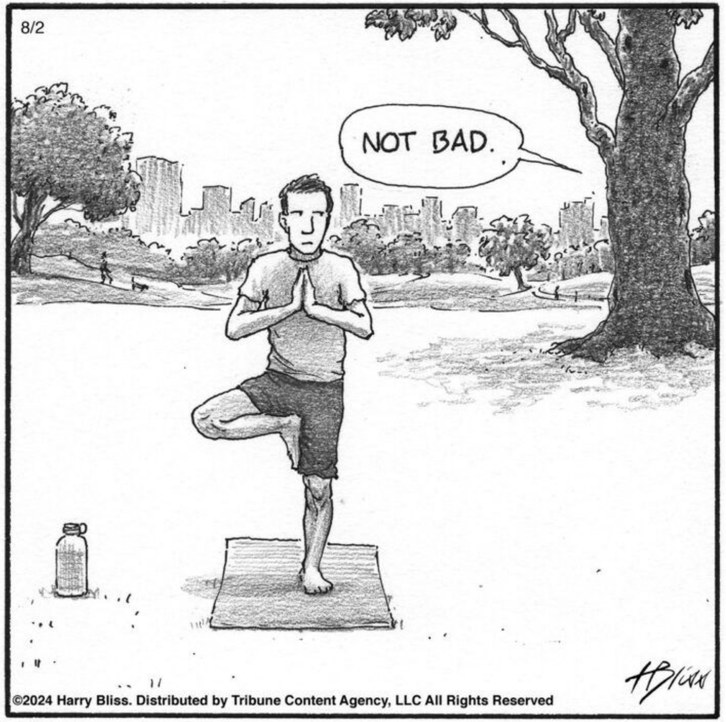
More useful interpretation than before!

The omitted column approach gives exactly the same predictions as before, but is better for **interpretability**.

Do you need a **left-handed desk** or other **non-DSP accommodation** for the midterm? Fill out this form! (Also linked on Ed)

2-minute stretch break!

Lecture 14, Data 100 Spring 2025



2657119

Demo: One-Hot Encoding

lec14.ipynb

Sunday	Thursday	Friday	Saturday	Bias
1	0	0	0	**1**
1	0	0	0	**1**
0	**1**	0	0	**1**
0	0	**1**	0	**1**
0	0	0	**1**	**1**

2657119

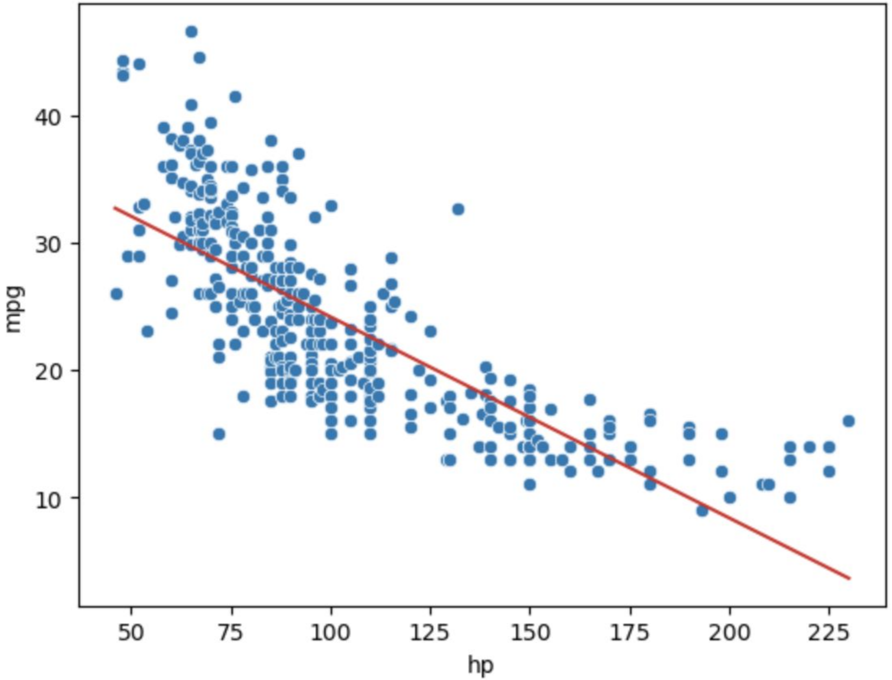
Polynomial Features

Lecture 14, Data 100 Spring 2025

- Gradient descent on Multi-Dimensional Models
- Batch, mini-batch, and stochastic gradient descent
- **Feature Engineering**
  - One-Hot Encoding
  - **Polynomial Features**
  - Complexity and Overfitting

2657119

We've seen a few cases now where models with linear features have performed poorly on datasets with a clear non-linear curve.



$$\hat{y} = \theta_0 + \theta_1(\text{hp})$$

MSE: 23.94

When our model uses only a single linear feature (**hp**), it cannot capture non-linearity in the relationship

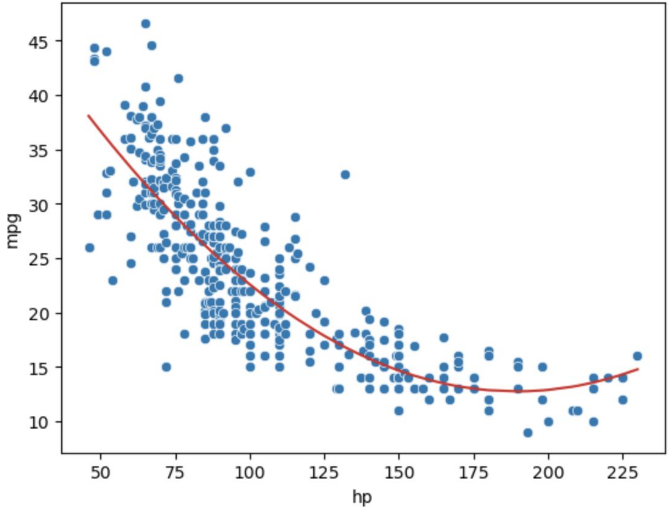
Solution: Incorporate a non-linear feature!

Polynomial Features

We create a new feature: the square of the **hp**

$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2)$$

This is still a **linear model**. Even though there are non-linear *features*, the model is linear with respect to θ

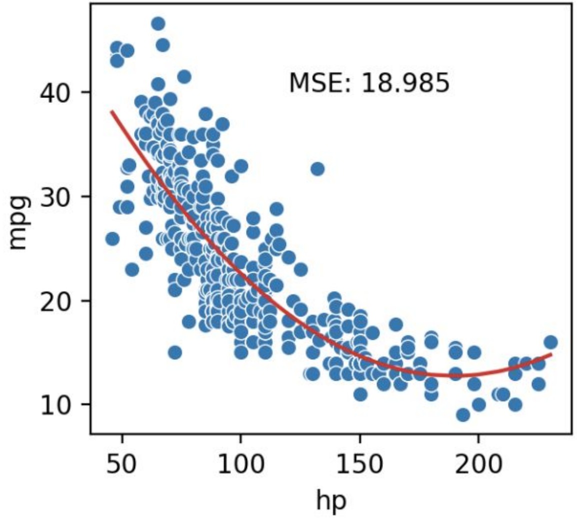


Degree of model: 2
MSE: 18.98

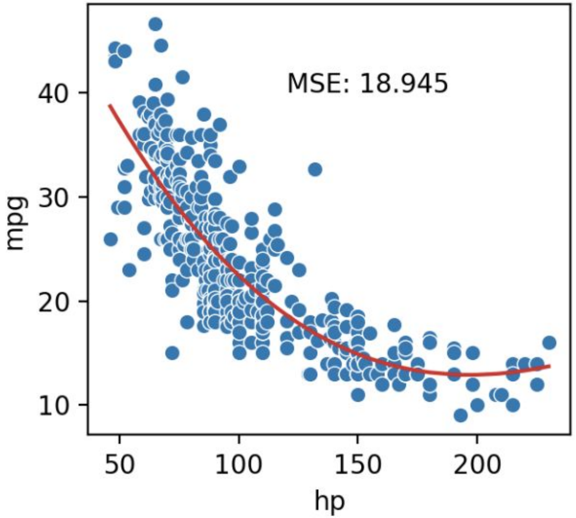
Looking a lot better: Our predictions capture the curvature of the data.

What if we add more polynomial features?

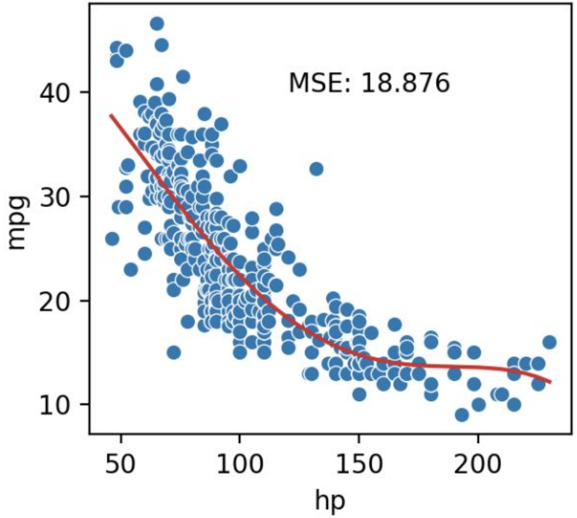
$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2)$$



$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2) + \theta_3(\text{hp}^3)$$



$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2) + \theta_3(\text{hp}^3) + \theta_4(\text{hp}^4)$$



MSE continues to decrease with each additional polynomial term. Should we keep going?

Complexity and Overfitting

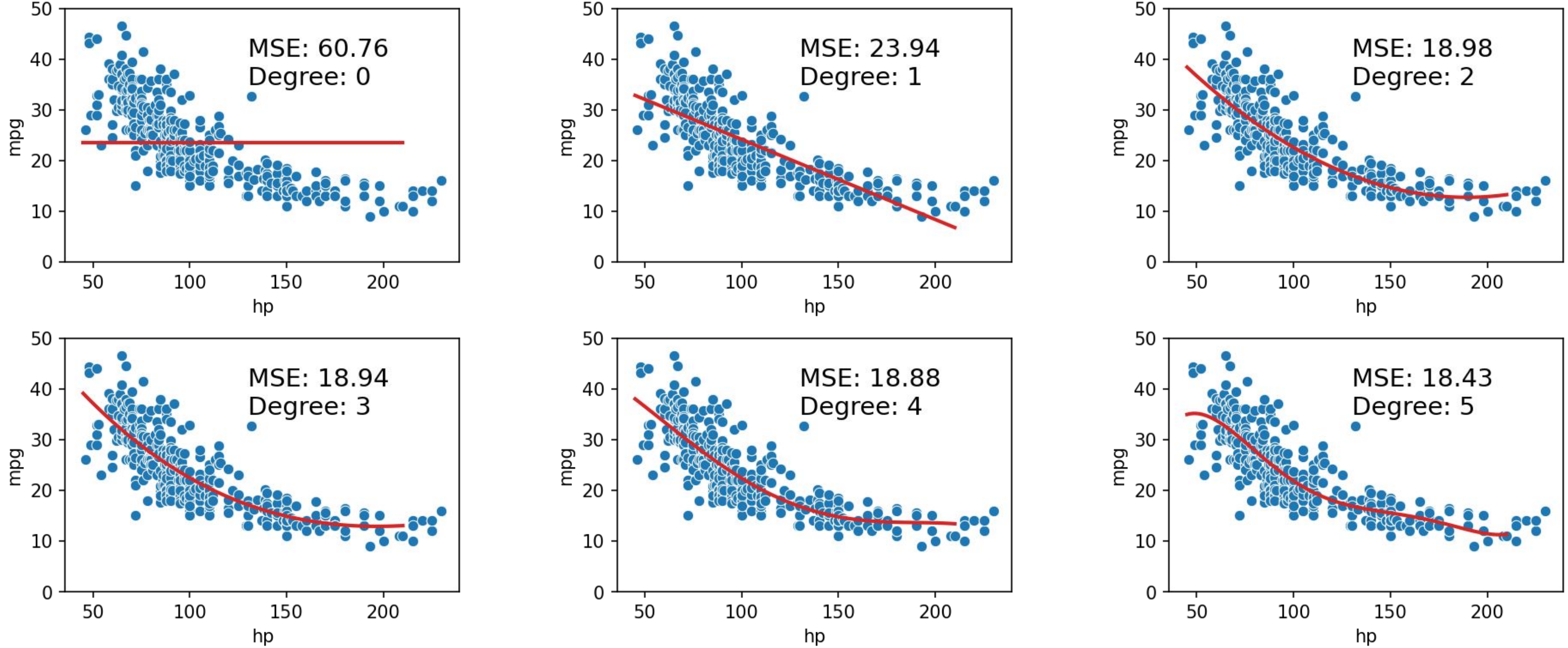
Lecture 14, Data 100 Spring 2025

- Gradient descent on Multi-Dimensional Models
- Batch, mini-batch, and stochastic gradient descent
- **Feature Engineering**
  - One-Hot Encoding
  - Polynomial Features
  - **Complexity and Overfitting**

2657119

How Far Can We Take This?

2657119



2657119

slido

Which higher-order polynomial model do you think fits best?

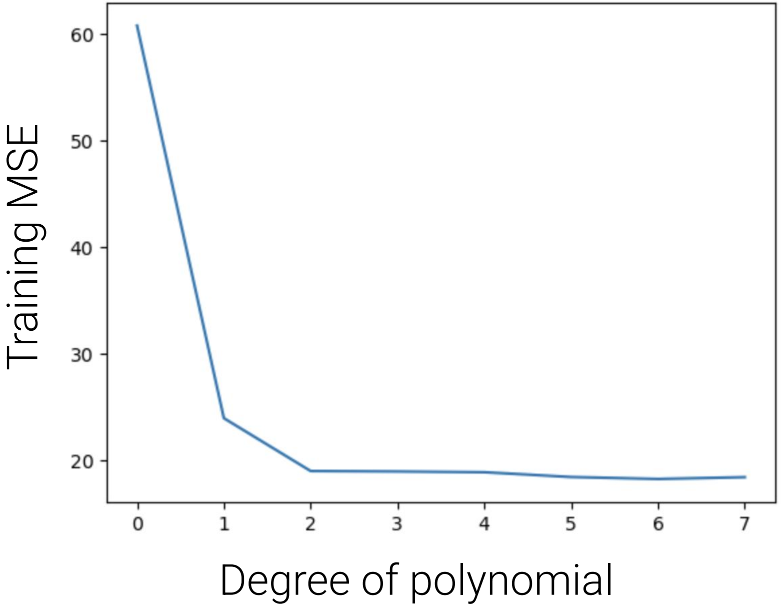
ⓘ Click **Present with Slido** or install our [Chrome extension] to activate this poll while presenting.

Model Complexity

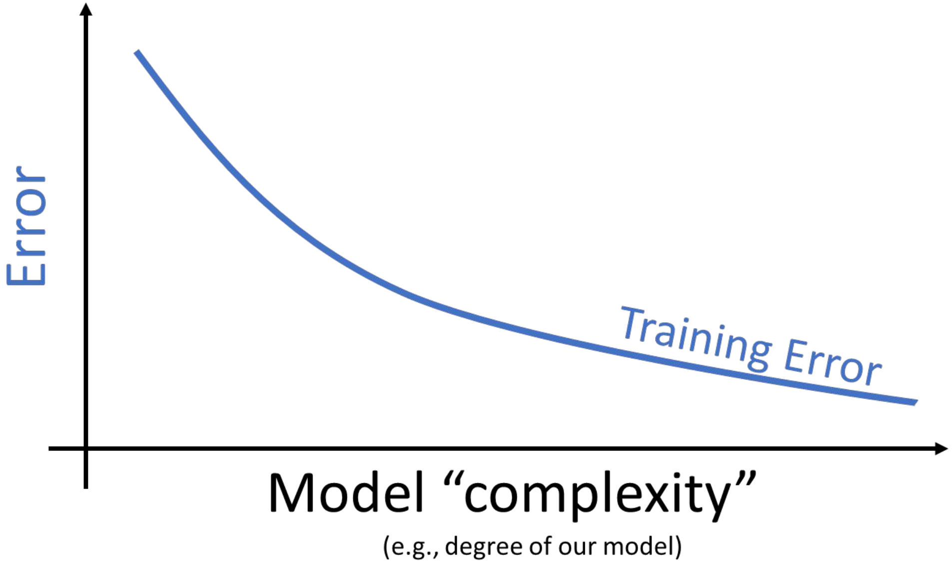
As we continue to add more polynomial features, the MSE decreases.

Equivalently: As the **model complexity** increases, its *training error* decreases.

Our experiment using `vehicles`



General trend for an arbitrary dataset

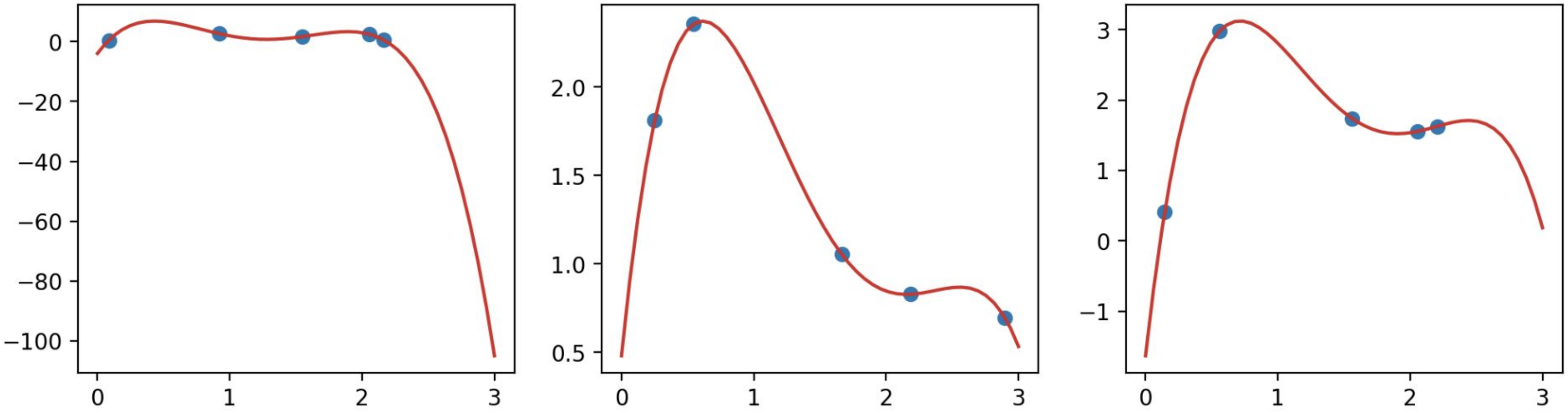


An Extreme Example: Perfect Polynomial Fits

2657119

Math fact: given **N** non-overlapping data points, we can always find a polynomial of degree **N-1** that goes through all those points.

For example, a degree-4 polynomial curve can perfectly model a dataset of 5 datapoints.



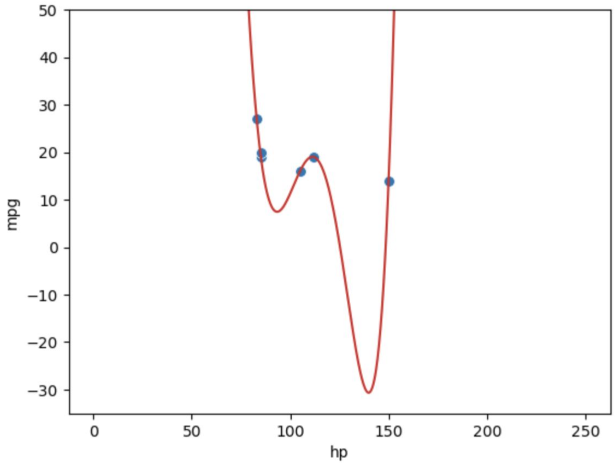
Model Performance on Unseen Data

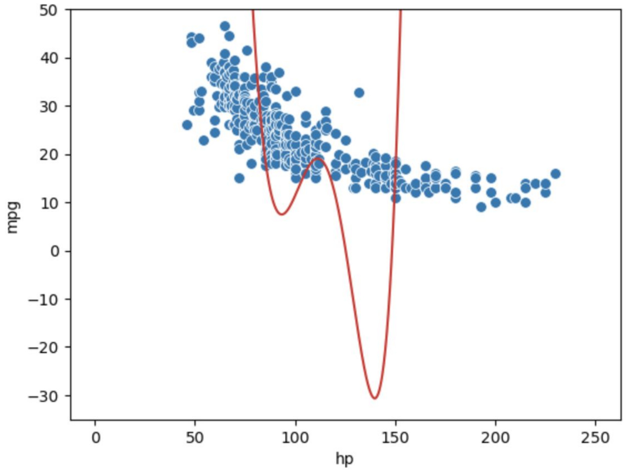
2657119

More realistic example:

- We are given a training dataset of just 6 data points.
- We want to train a model to then make predictions on a *different* set of points

We may be tempted to make a highly complex model (e.g., degree 5):





Complex model makes perfect predictions on the training data…

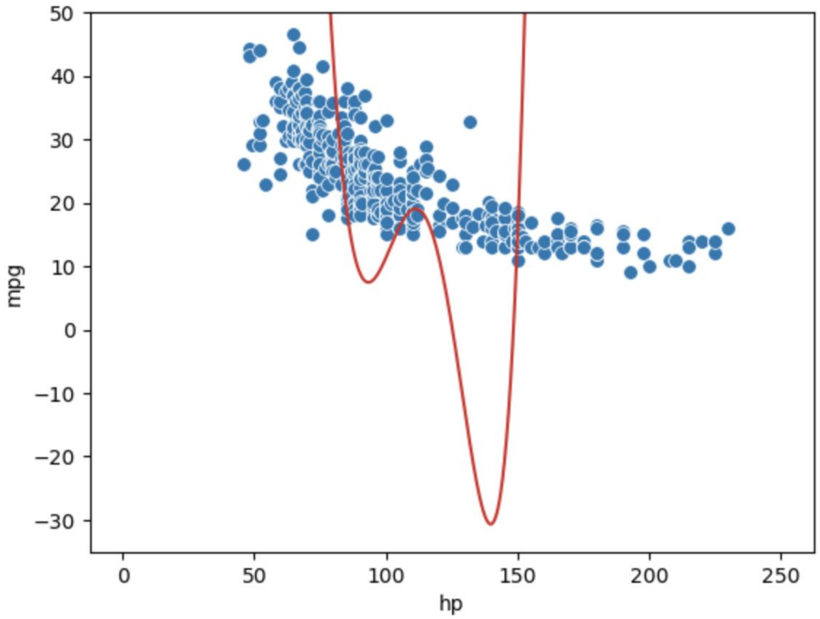
…but performs *horribly* on new data!

Model Performance on Unseen Data

2657119

What went wrong?

- The complex model **overfit** to the training data – it "memorized" the 6 training points.
- The overfitted model does not **generalize** well to new data.



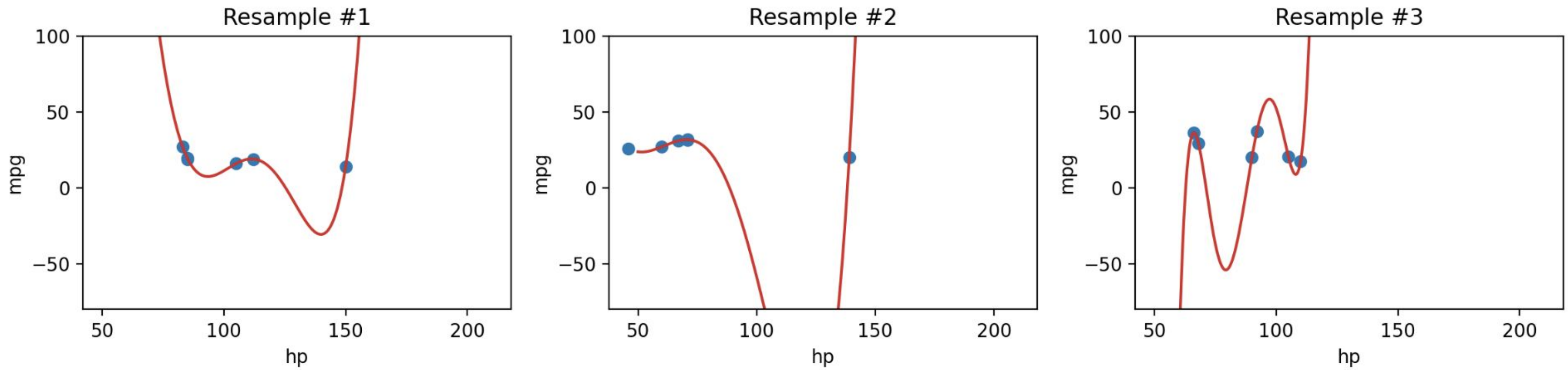
This is a problem: we want models that are generalizable to "unseen" data

Model Variance

2657119

Complex models are sensitive to the specific dataset used to train them.

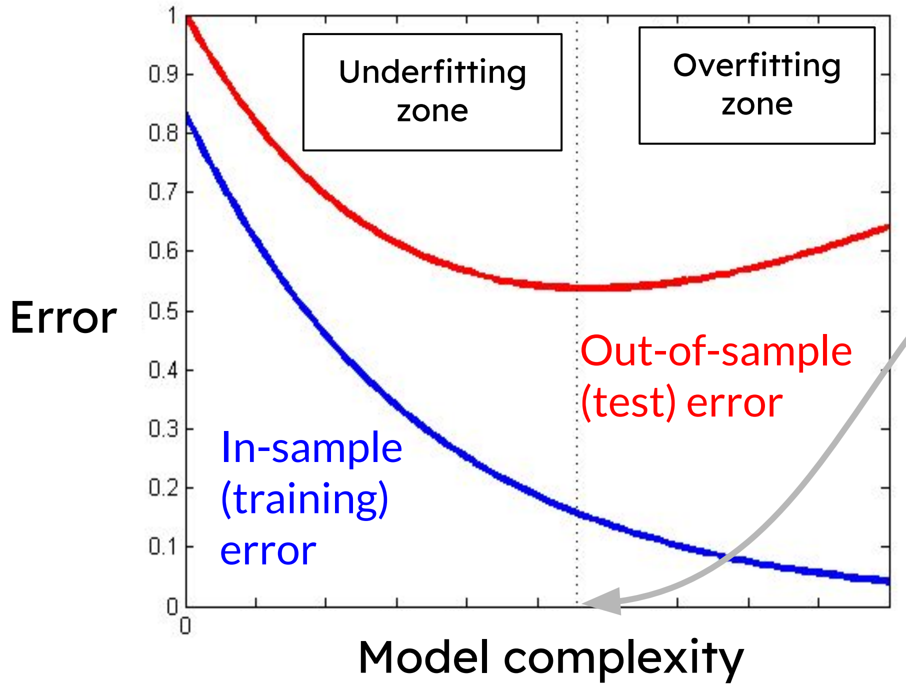
In other words, they have high **variance**, because the fitted model will *vary* a lot across different training samples, even if they were are randomly drawn from the same population!



Error, Variance, and Complexity

We know that we can **decrease training error** by increasing model complexity

However, models that are *too* complex (e.g., high-degree polynomial) do not generalize well.



Source

Our goal: Find this "sweet spot"

Future lecture!

2657119

LECTURE 14

Gradient Descent II & Feature Engineering

Content credit: Acknowledgments