Regression

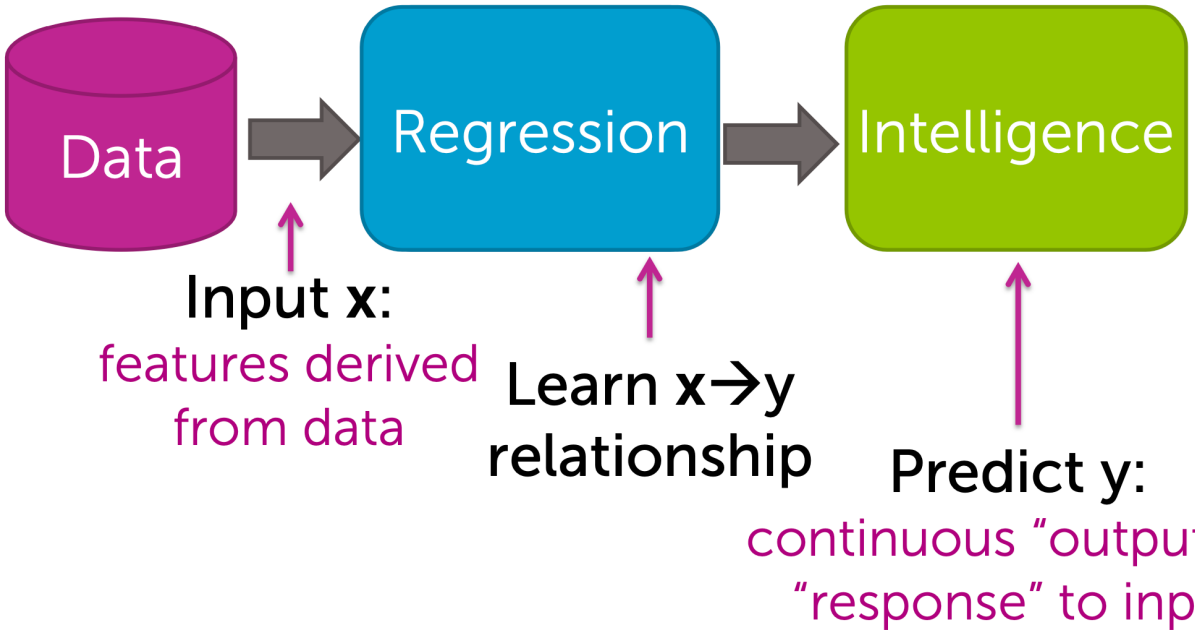
K Kotecha

https://www.linkedin.com/in/ketankotecha/

- Material taken in this slides are from various sources. Its only used for education purpose.

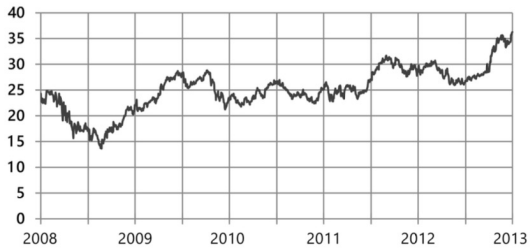
What is regression?

From features to predictions



Stock prediction

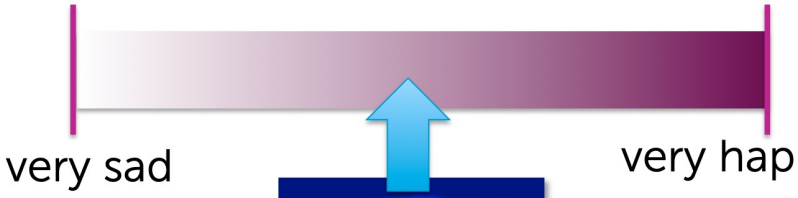
- Predict the price of a stock (y)
- Depends on **x** =
  - Recent history of stock price
  - News events
  - Related commodities

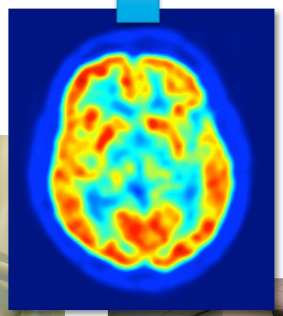


Tweet popularity

- How many people will retweet your tweet? (y)
- Depends on **x** = # followers,
  # of followers of followers,
  features of text tweeted,
  popularity of hashtag,
  # of past retweets,...

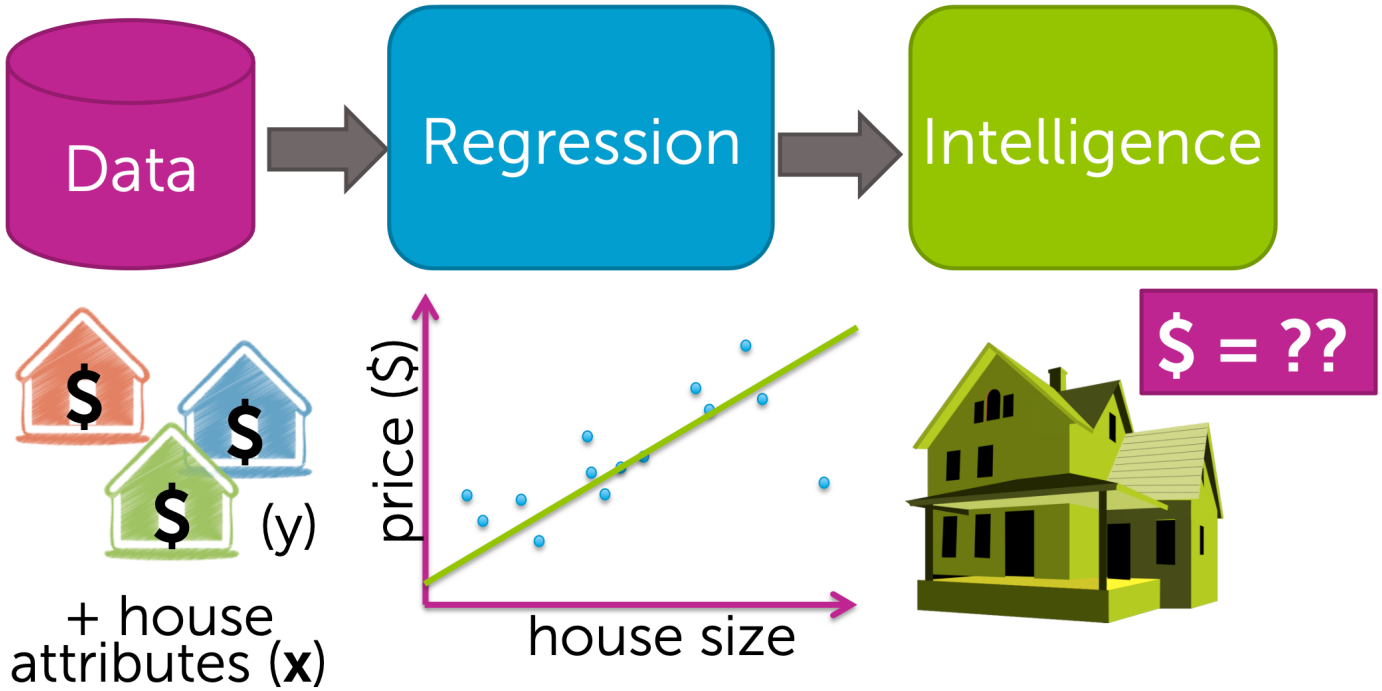
Reading your mind



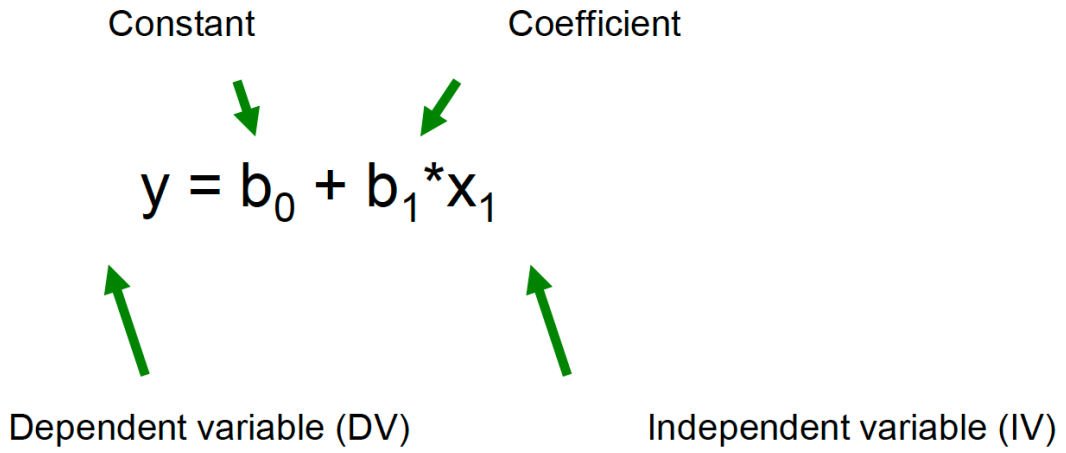


Inputs x are brain region intensities

Case Study:
Predicting house prices



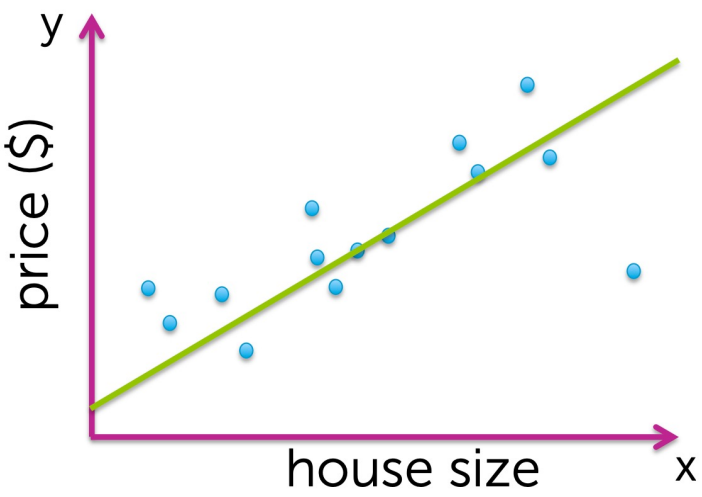
Simple Linear Regression



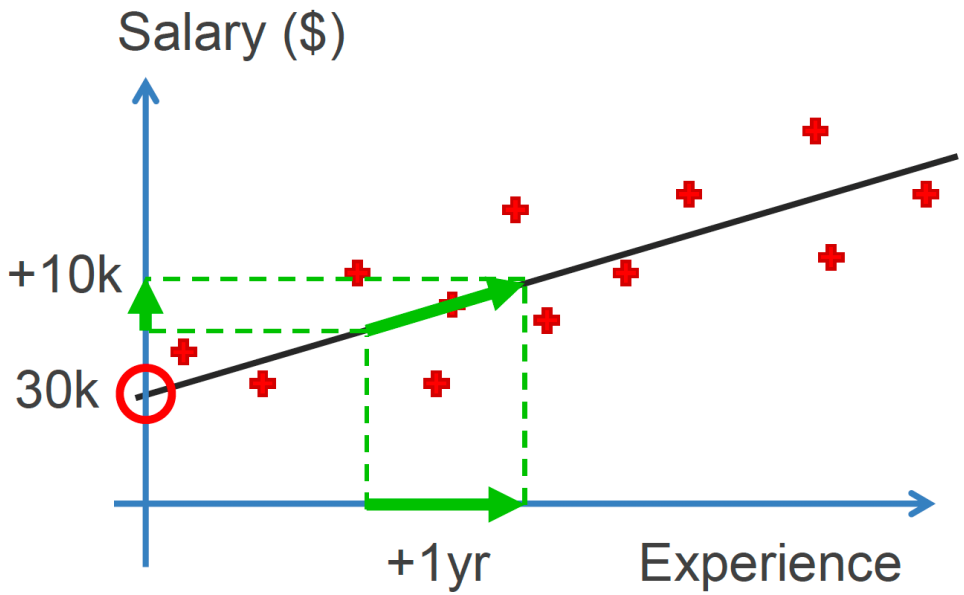
Simple Regression

What makes it simple?

1 input and just fit a line to data



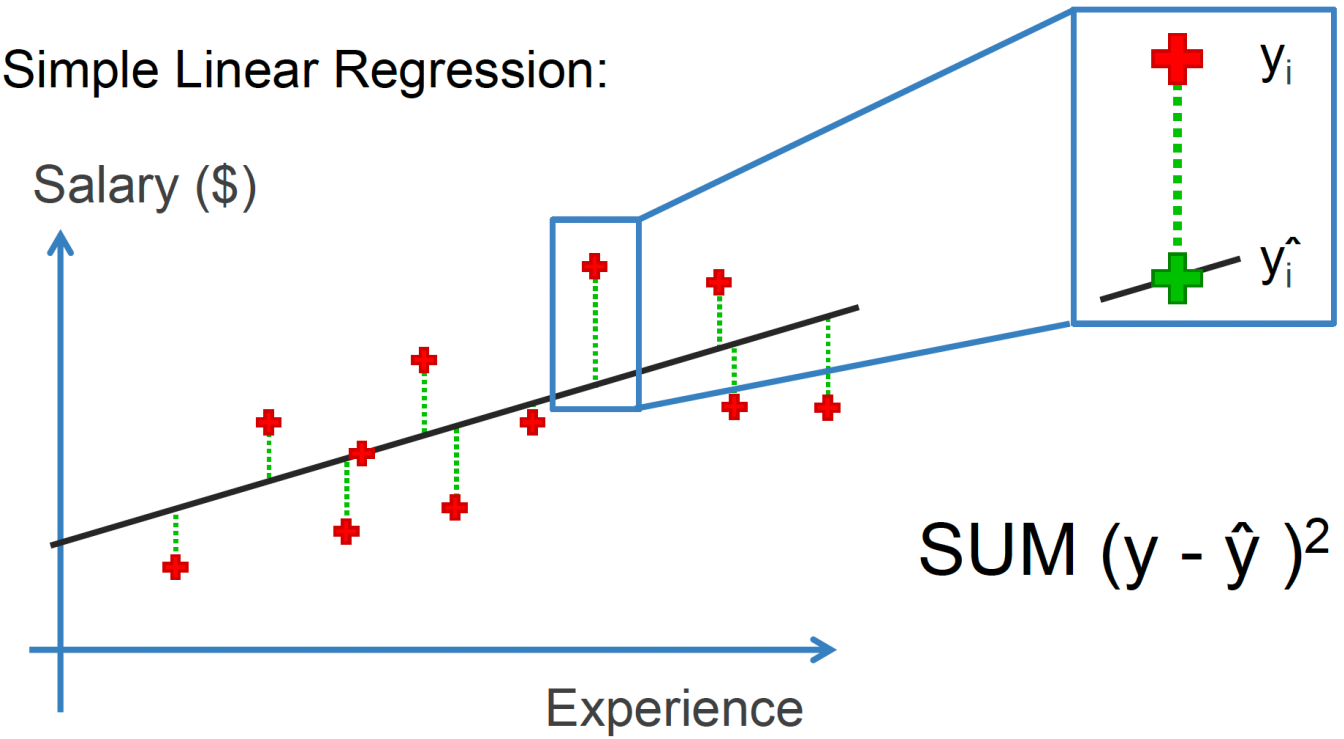
Simple Linear Regression:



$$y = b_0 + b_1 * x$$

Salary = b_0 + b_1 *Experience

Simple Linear Regression:



$$SUM\ (y - \hat{y})^2 \quad \text{-> min}$$

How much is my house worth?

Look at recent sales in my neighborhood

- How much did they sell for?

Data

input *output*

$(x_1 = \text{sq.ft.}, y_1 = \$)$

$(x_2 = \text{sq.ft.}, y_2 = \$)$

$(x_3 = \text{sq.ft.}, y_3 = \$)$

$(x_4 = \text{sq.ft.}, y_4 = \$)$

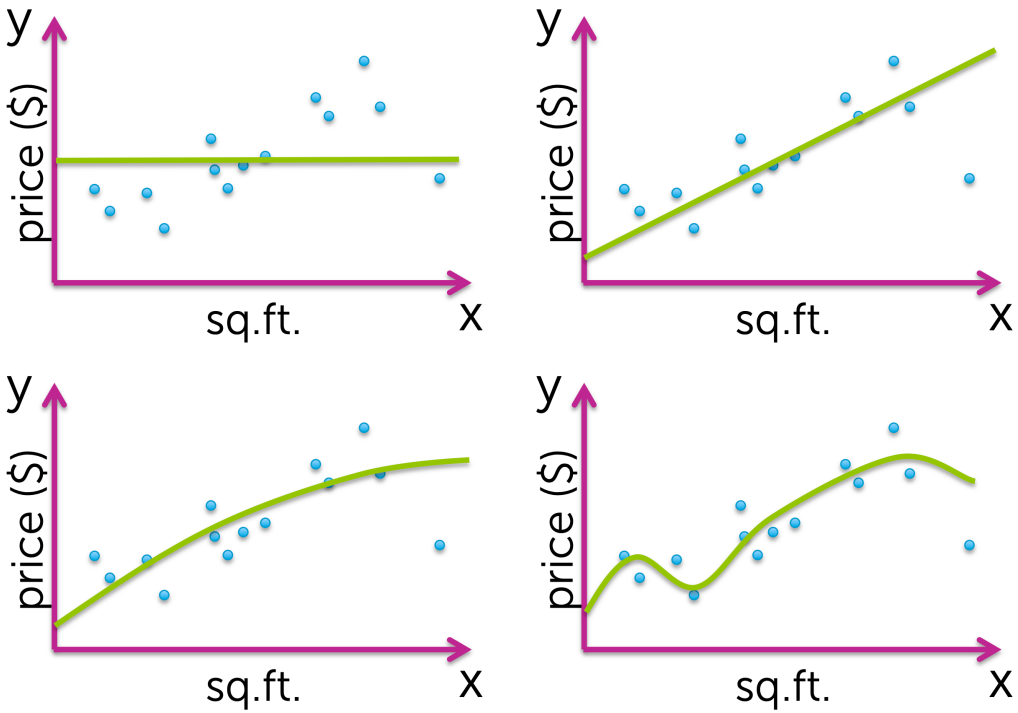
$(x_5 = \text{sq.ft.}, y_5 = \$)$

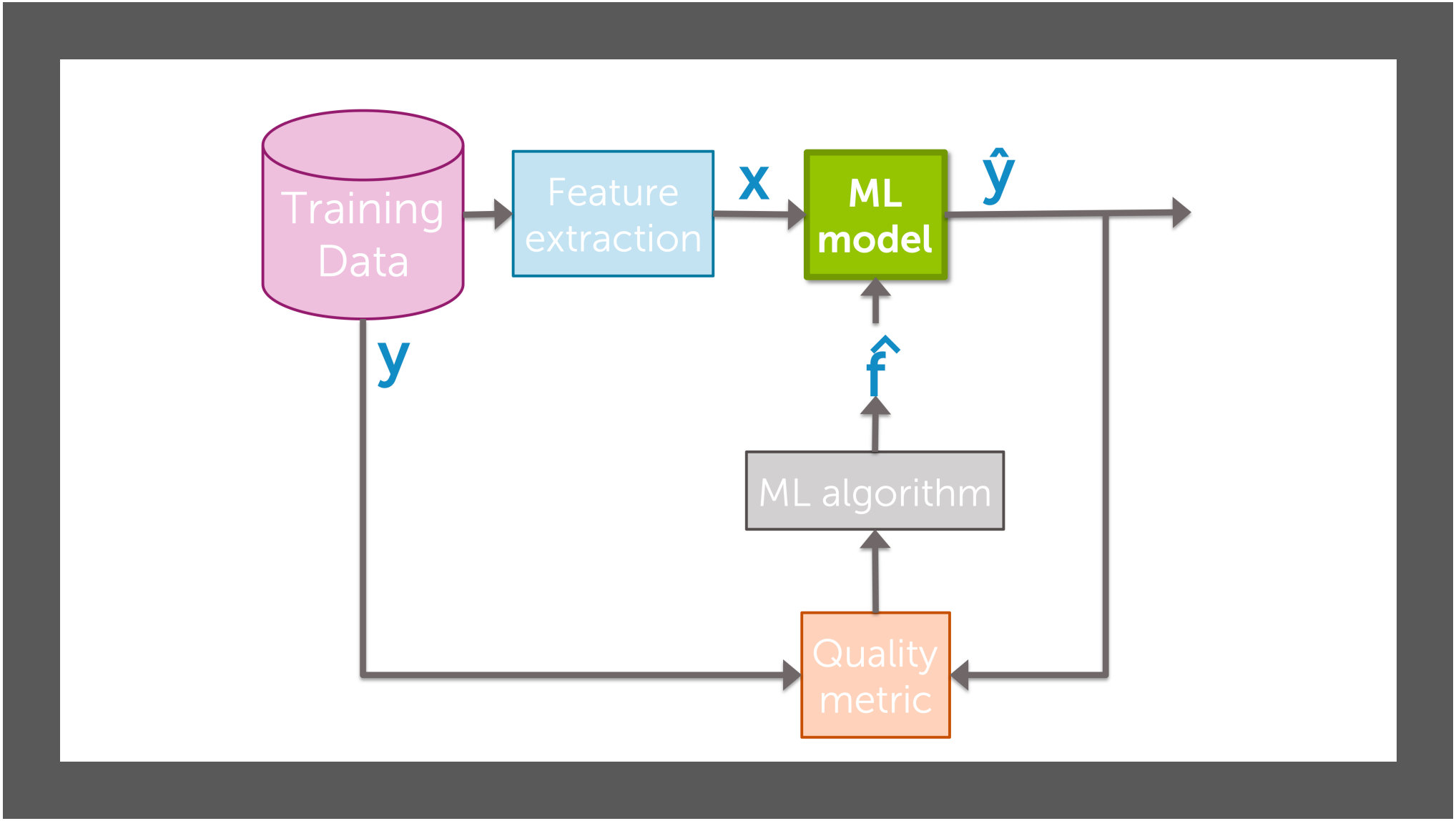
.

Input vs. Output:

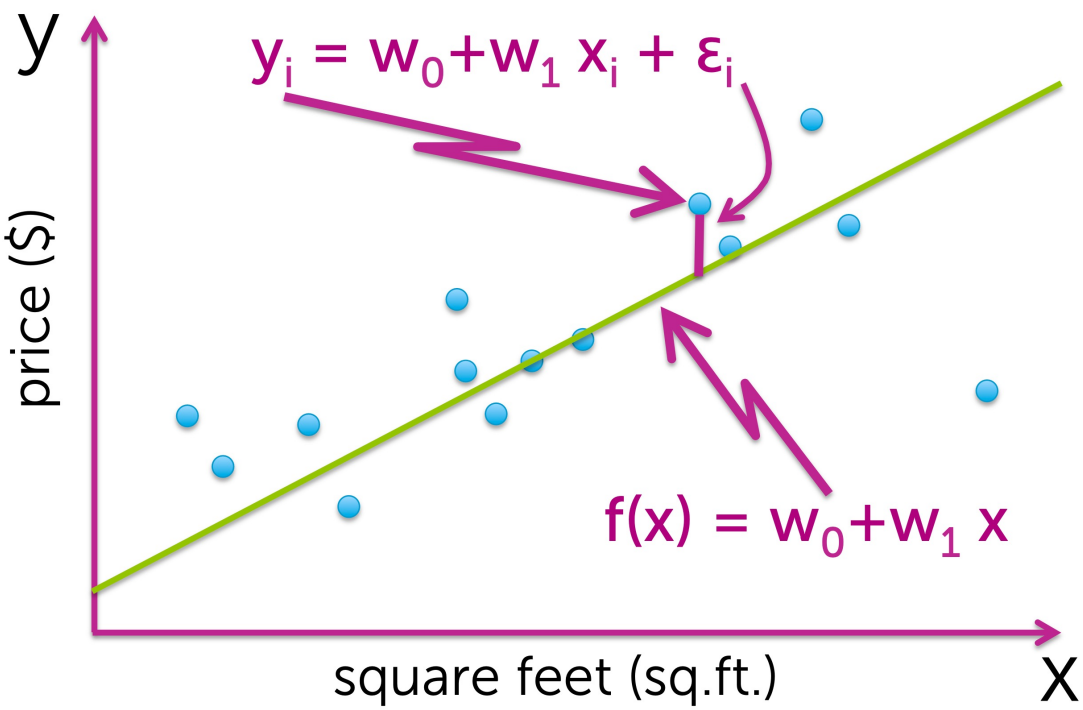
- y is the quantity of interest
- assume y can be predicted from x

Task 1–
Which model f(x)?

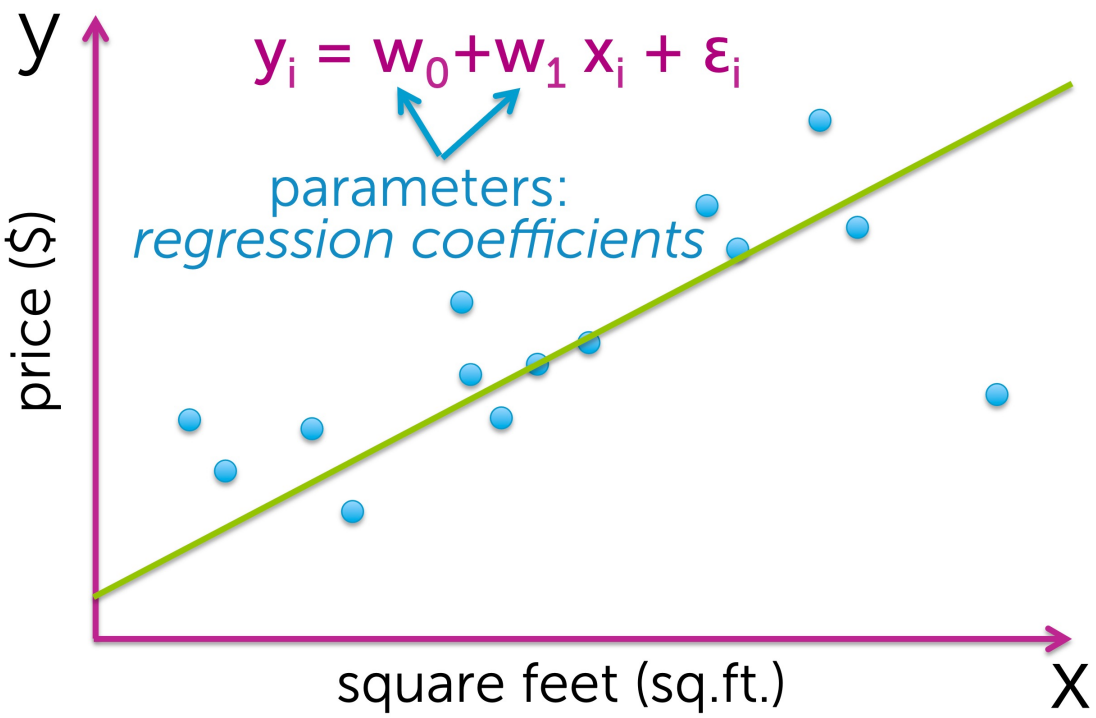


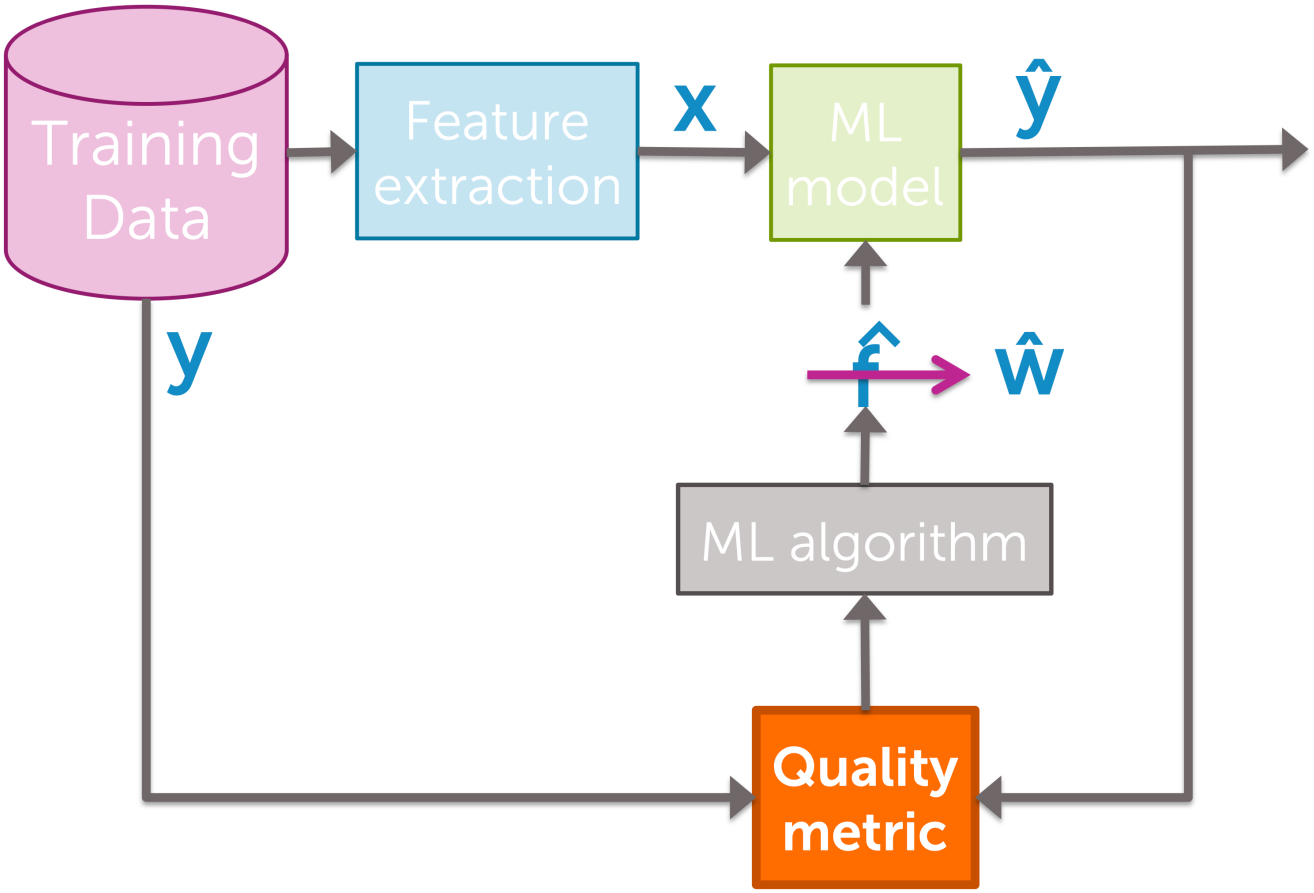

Training Data
Feature extraction
x
ML model
ŷ
y
f̂
ML algorithm
Quality metric

Simple linear regression model



Simple linear regression model




Training Data
Feature extraction
x
ML model
ŷ
y
f̂
ŵ
ML algorithm
Quality metric

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

Simple Linear Regression

$$y = b_0 + b_1 * x_1$$

Multiple Linear Regression

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Multiple Linear Regression

$$y = b_0 + b_1 * x_1 + b_2 * x_2$$

Multivariate Regression

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

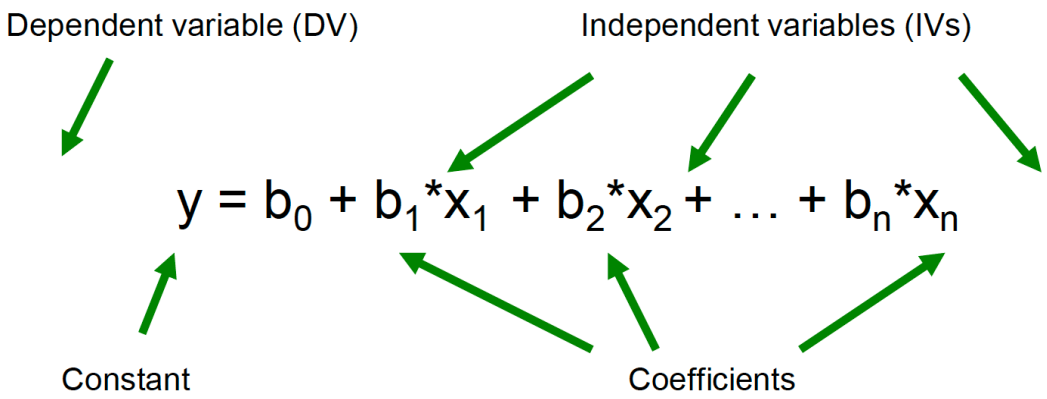
Multiple Linear Regression

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + \ldots + b_n * x_n$$

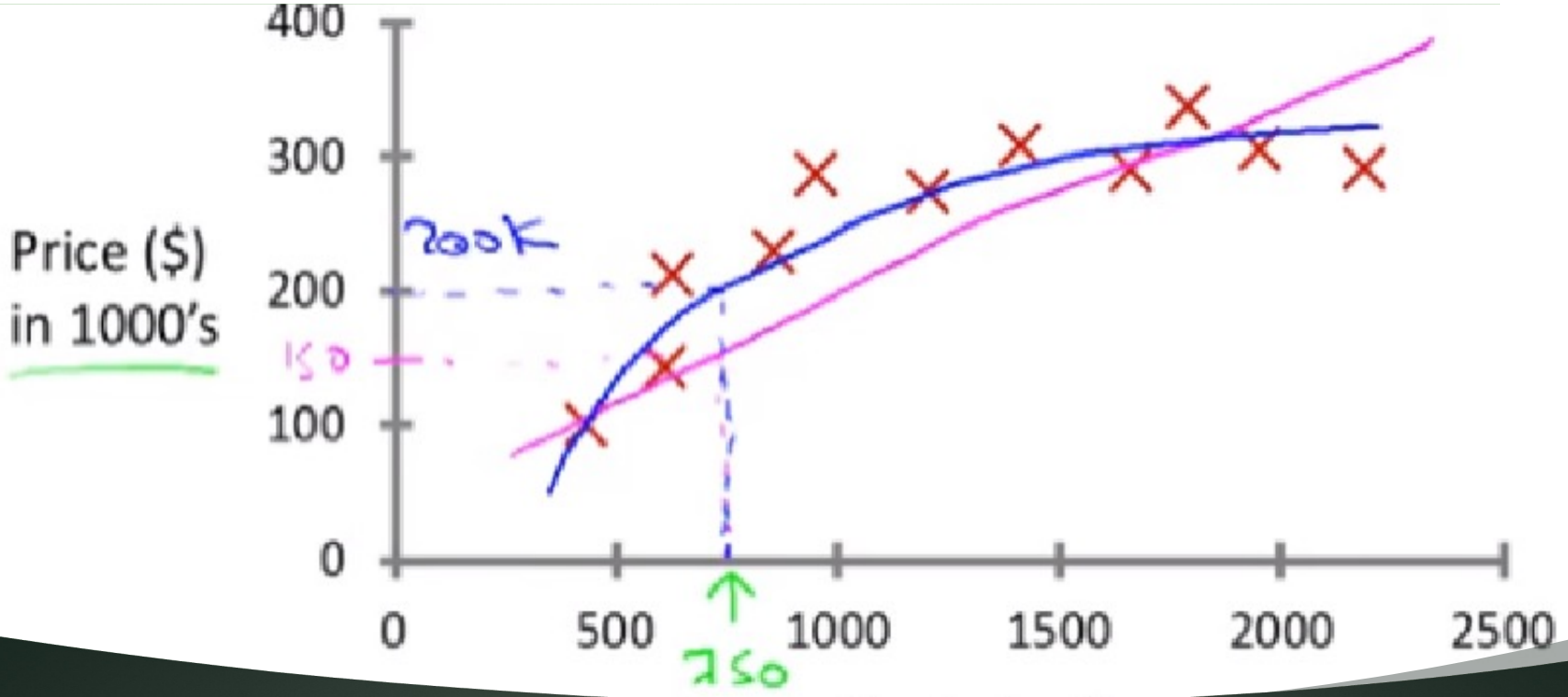
Simple Linear Regression

$$y = b_0 + b_1 * x_1$$

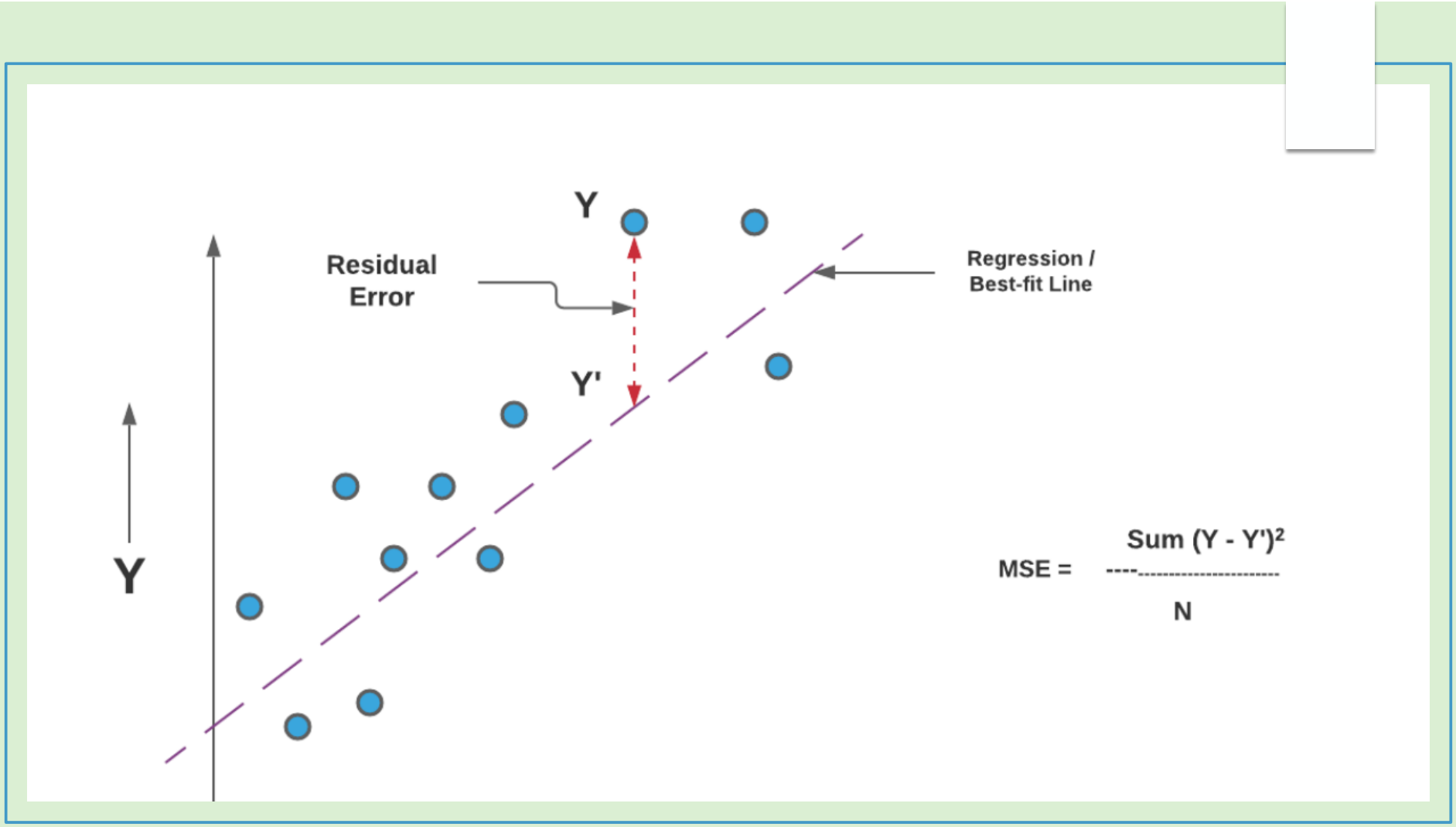
Multiple Linear Regression



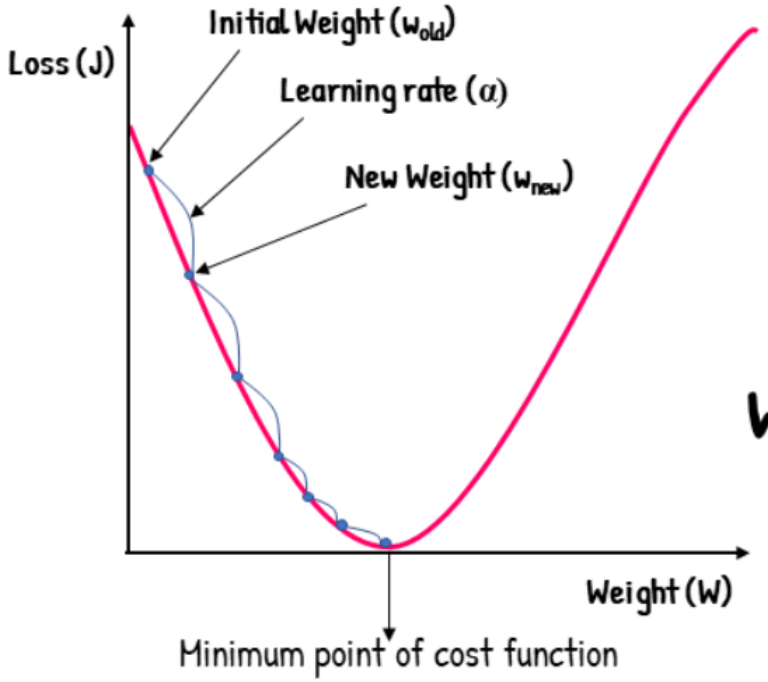
Gradient Descent Algorithm



Housing Price Prediction (Recap)


Y
Residual
Error
Regression /
Best-fit Line
Y'
Y
MSE = Sum (Y - Y')²
N

Gradient Descent



$$w_{new} = w_{old} - \alpha \frac{\delta J}{\delta w}$$

Regression loss functions

Mean Bias Error	Captures average bias in prediction. But is rarely used for training.	$\mathcal{L}_{\mathcal{MBE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))$
Mean Absolute Error	Measures absolute average bias in prediction. Also called L1 Loss.	$\mathcal{L}_{\mathcal{MAE}} = \frac{1}{N}\sum_{i=1}^{N}\lvert y_i - f(x_i)\rvert$
Mean Squared Error	Average squared distance between actual and predicted. Also called L2 Loss.	$\mathcal{L}_{\mathcal{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$
Root Mean Squared Error	Square root of MSE. Loss and dependent variable have same units.	$\mathcal{L}_{\mathcal{RMSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2}$
Huber Loss	A combination of MSE and MAE. It is parametric loss function.	$\mathcal{L}_{\text{Huberloss}} = \begin{cases} \frac{1}{2}(y_i - f(x_i))^2 & : \lvert y_i - f(x_i)\rvert \le \delta \\ \delta(\lvert y_i - f(x_i)\rvert - \frac{1}{2}\delta) & : otherwise \end{cases}$
Log Cosh Loss	Similar to Huber Loss + non-parametric. But computationally expensive.	$\mathcal{L}_{\text{LogCosh}} = \frac{1}{N}\sum_{i=1}^{N} log(cosh(f(x_i) - y_i))$

Gradient Descent

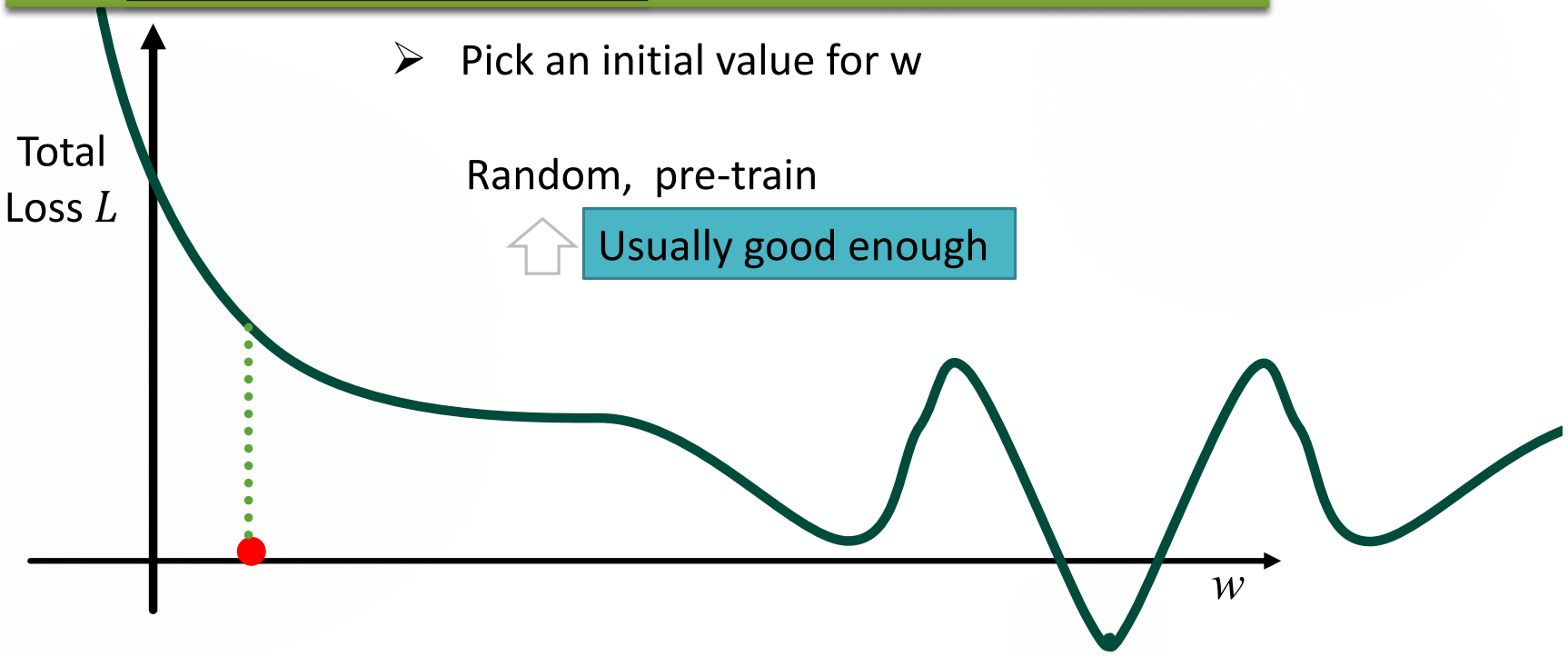
Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Find ***network parameters θ^**** that minimize total loss L

- Pick an initial value for w

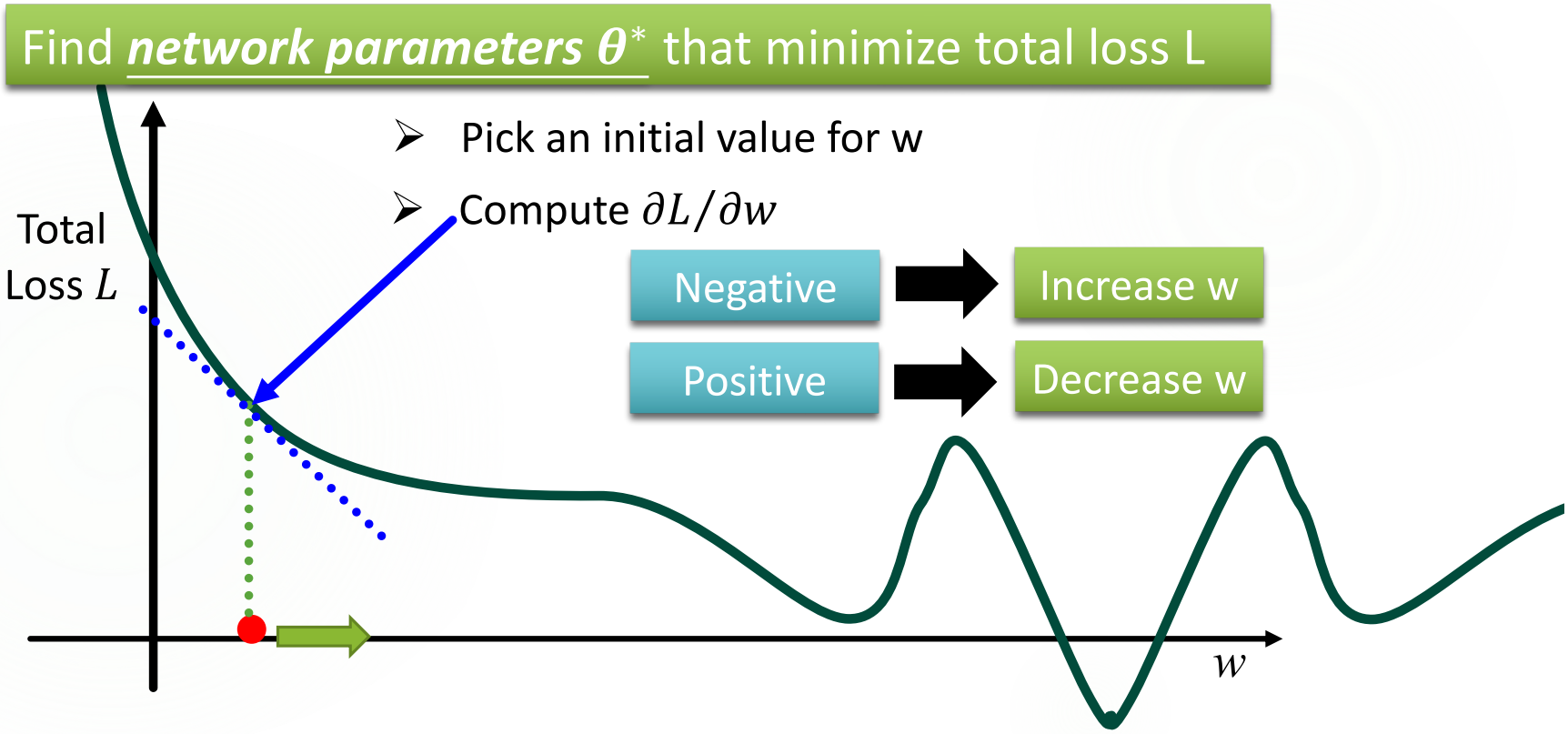
Random, pre-train

Usually good enough



Gradient Descent

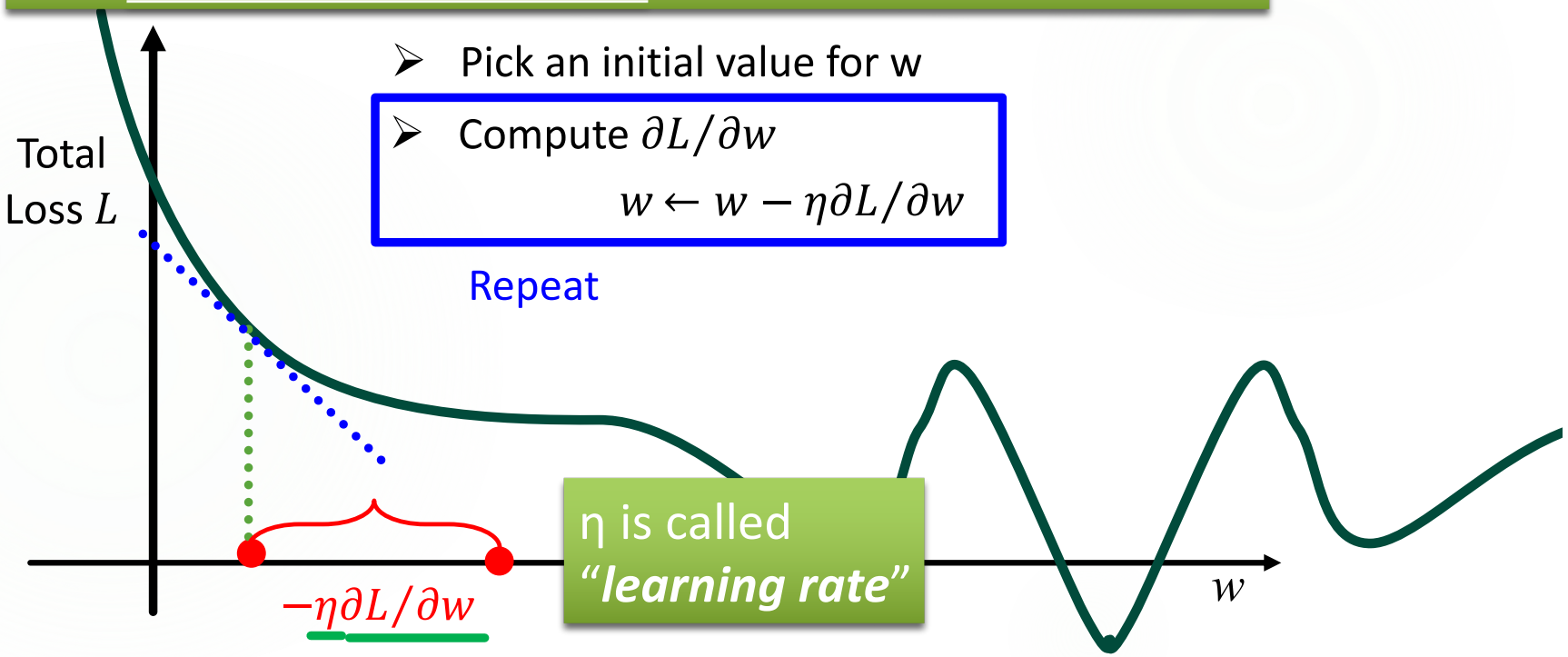
Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$



Gradient Descent

Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

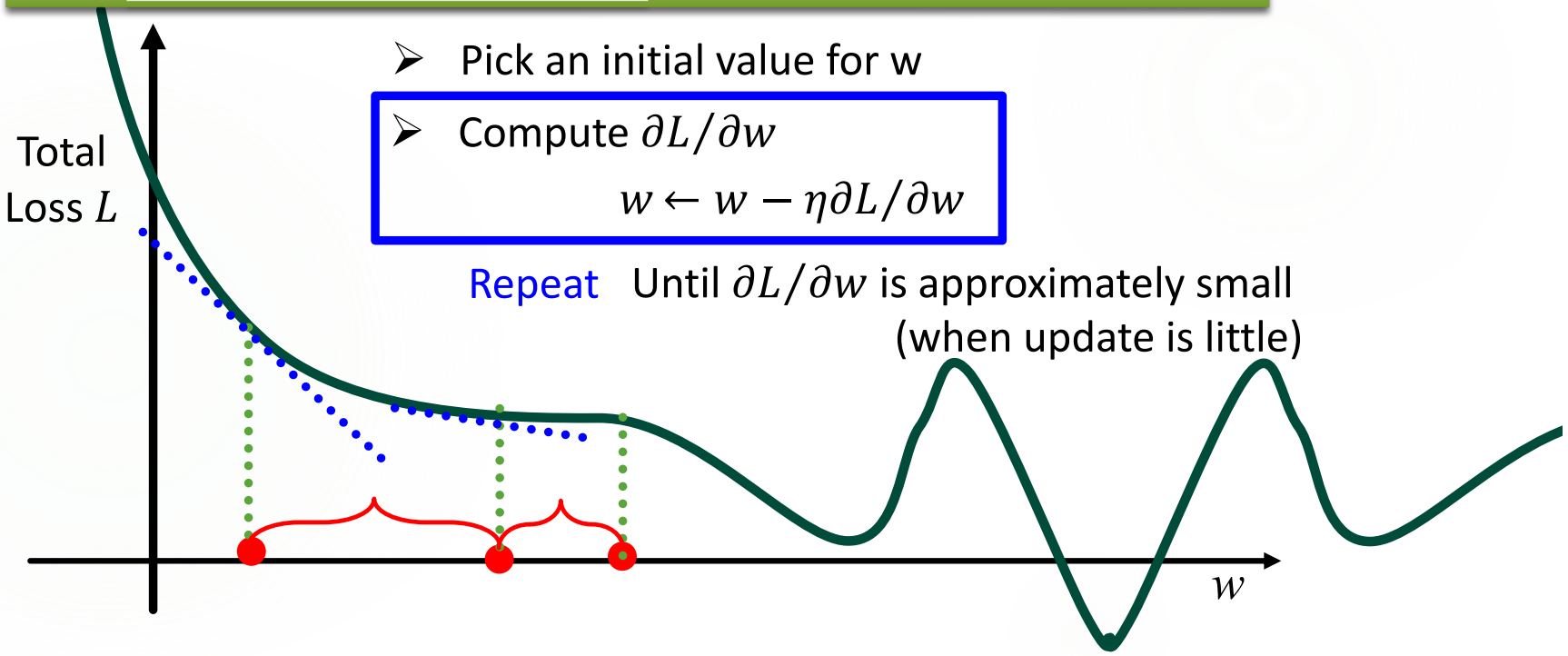
Find ***network parameters*** θ^* that minimize total loss L

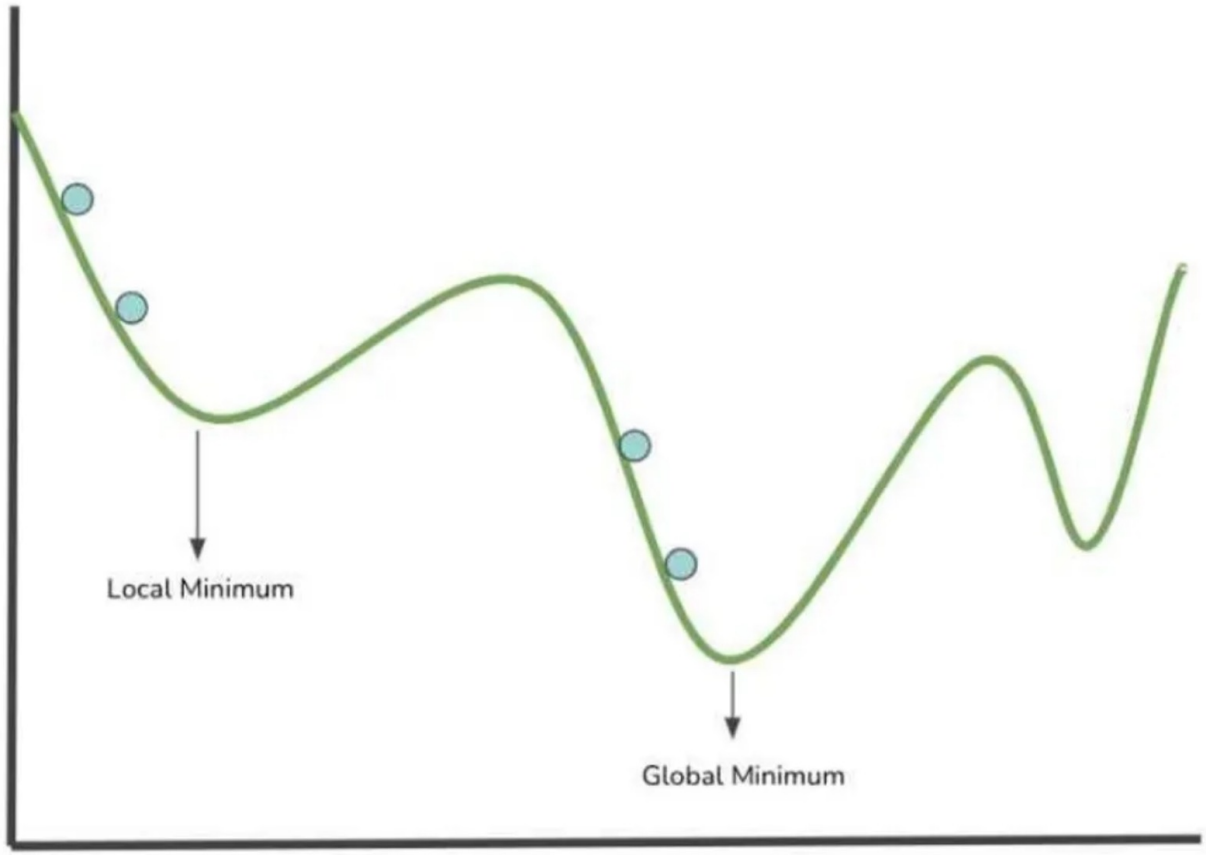


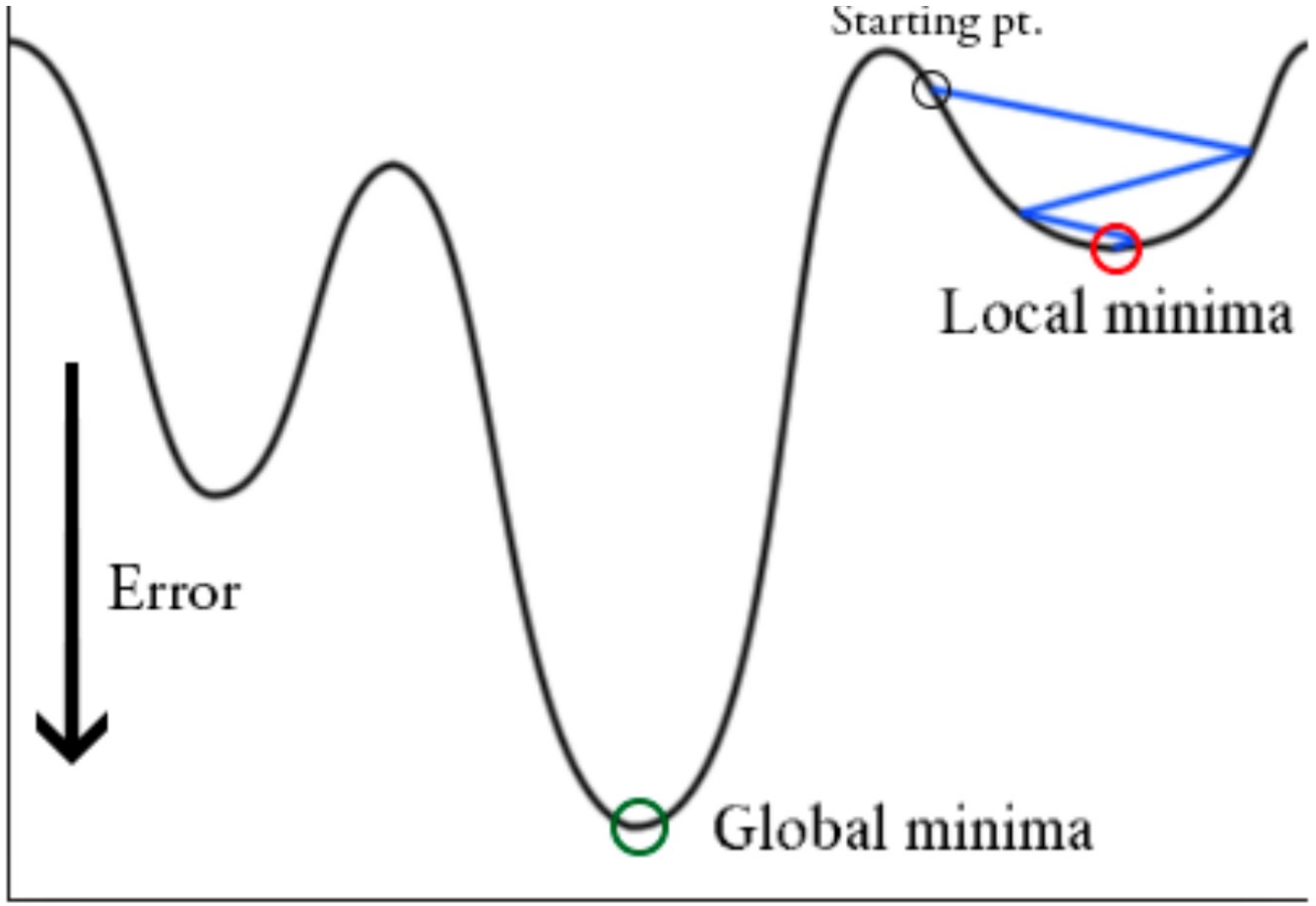
Gradient Descent

Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Find ***network parameters*** θ^* that minimize total loss L




Local Minimum
Global Minimum


Starting pt.
Local minima
Error
Global minima

SGD-illustrated

K Kotecha

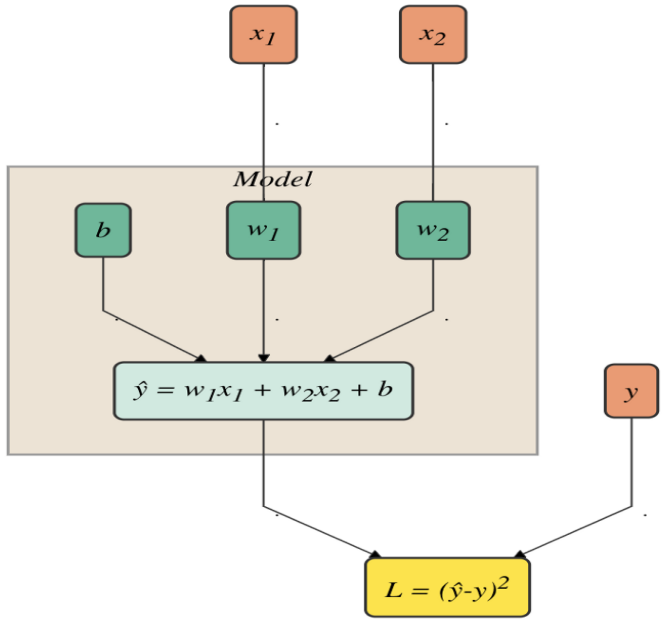
Problem

	x1	x2	y
1)	4	1	2
2)	2	8	-14
3)	1	0	1
4)	3	2	-1
5)	1	4	-7
6)	6	7	-8

$$\hat{y} = w_1 x_1 + w_2 x_2 + b$$

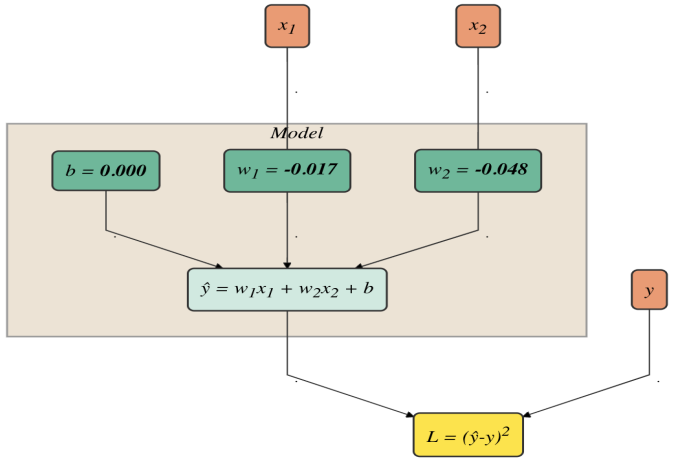
$$\hat{y} = 0.43 x_1 - 0.21 x_2 + 0.77$$

Representation



Epoch 1 with batch size 1

Epoch: - Batch: -



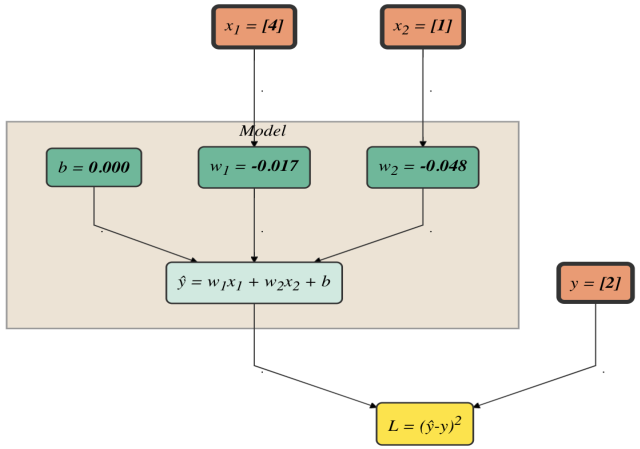
$$w_1 = -0.017$$

$$w_2 = -0.048$$

$$b = 0$$

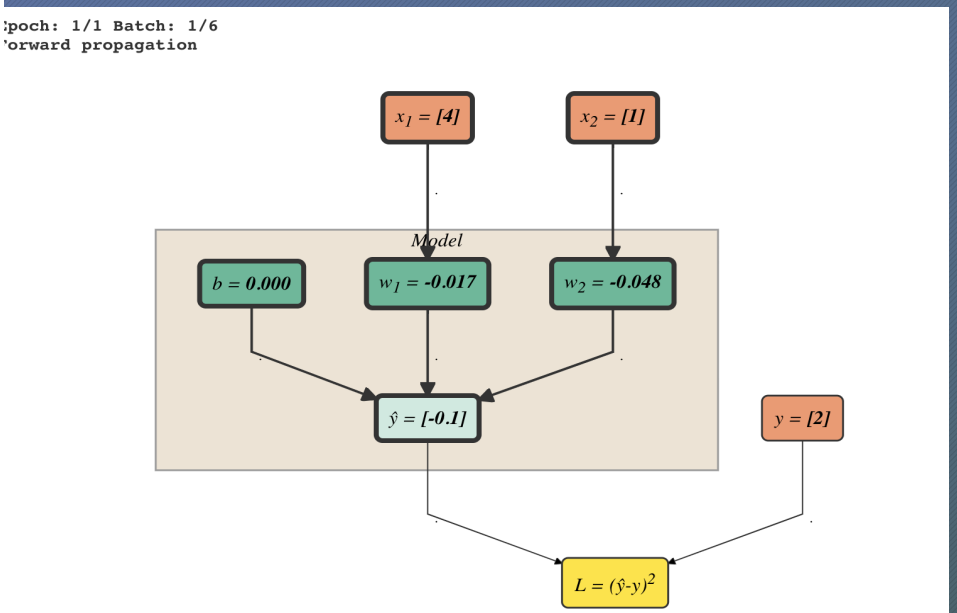
Y=2

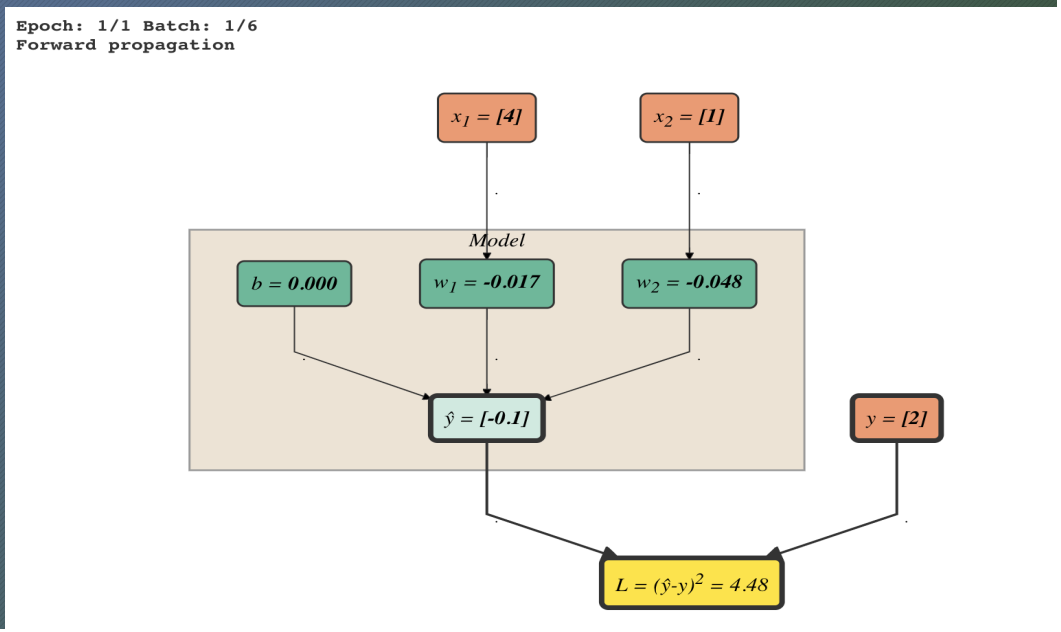
Epoch: 1/1 Batch: 1/6
Forward propagation



Forward propagation

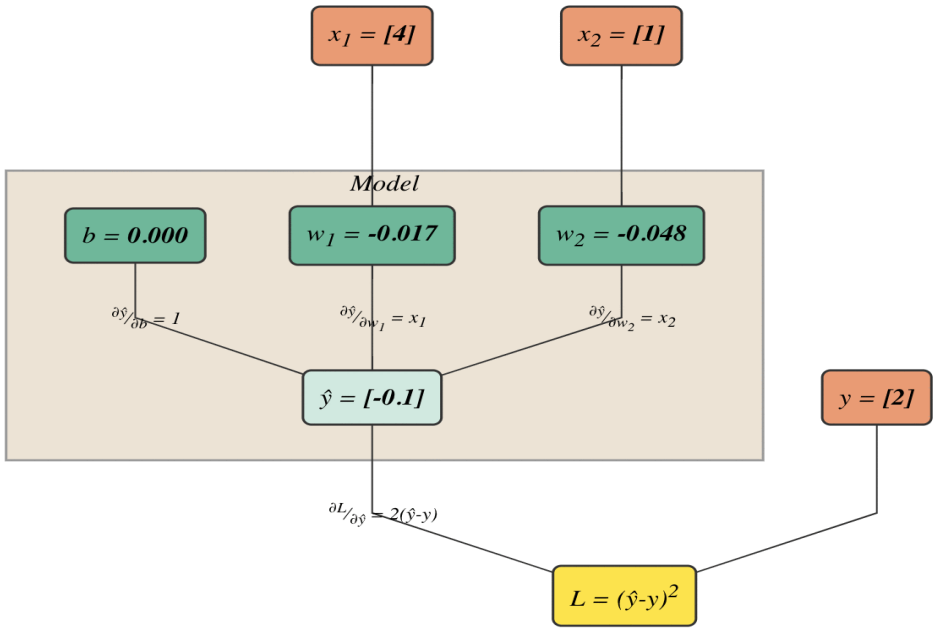
$$\begin{aligned}\hat{y} &= w_1 x_1 + w_2 x_2 + b \\ &= (-0.017) \cdot 4 + (-0.048) \cdot 1 + 0.000 \\ &= -0.116\end{aligned}$$





Derivatives

Epoch: 1/1 Batch: 1/6
Backpropagation



Updating bias

$$\hat{y} = w_1 x_1 + w_2 x_2 + b$$

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

$$b' = b - \eta \frac{\partial L}{\partial b}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial b}$$

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y) \qquad \frac{\partial \hat{y}}{\partial b} = 1$$

$$\begin{aligned} b' &= b - \eta \frac{\partial L}{\partial b} \\ &= b - \eta \left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b}\right) \\ &= b - \eta [2(\hat{y} - y) \cdot 1] \\ &= 0.000 - 0.05[2(-0.116 - 2) \cdot 1] \\ &= 0.212 \end{aligned}$$

$x_1 = 4$
$x_2 = 1$
$y = 2$
$\hat{y} = -0.116$

Updating weights

$$\hat{y} = w_1 x_1 + w_2 x_2 + b$$

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

$$\begin{aligned} w_2' &= w_2 - \eta \frac{\partial L}{\partial w_2} \\ &= w_2 - \eta(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}) \\ &= w_2 - \eta[2(\hat{y} - y) \cdot x_2] \\ &= -0.048 - 0.05[2(-0.116 - 2) \cdot 1] \\ &= 0.164 \end{aligned}$$

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\frac{\partial \hat{y}}{\partial w_2} = x_2$$

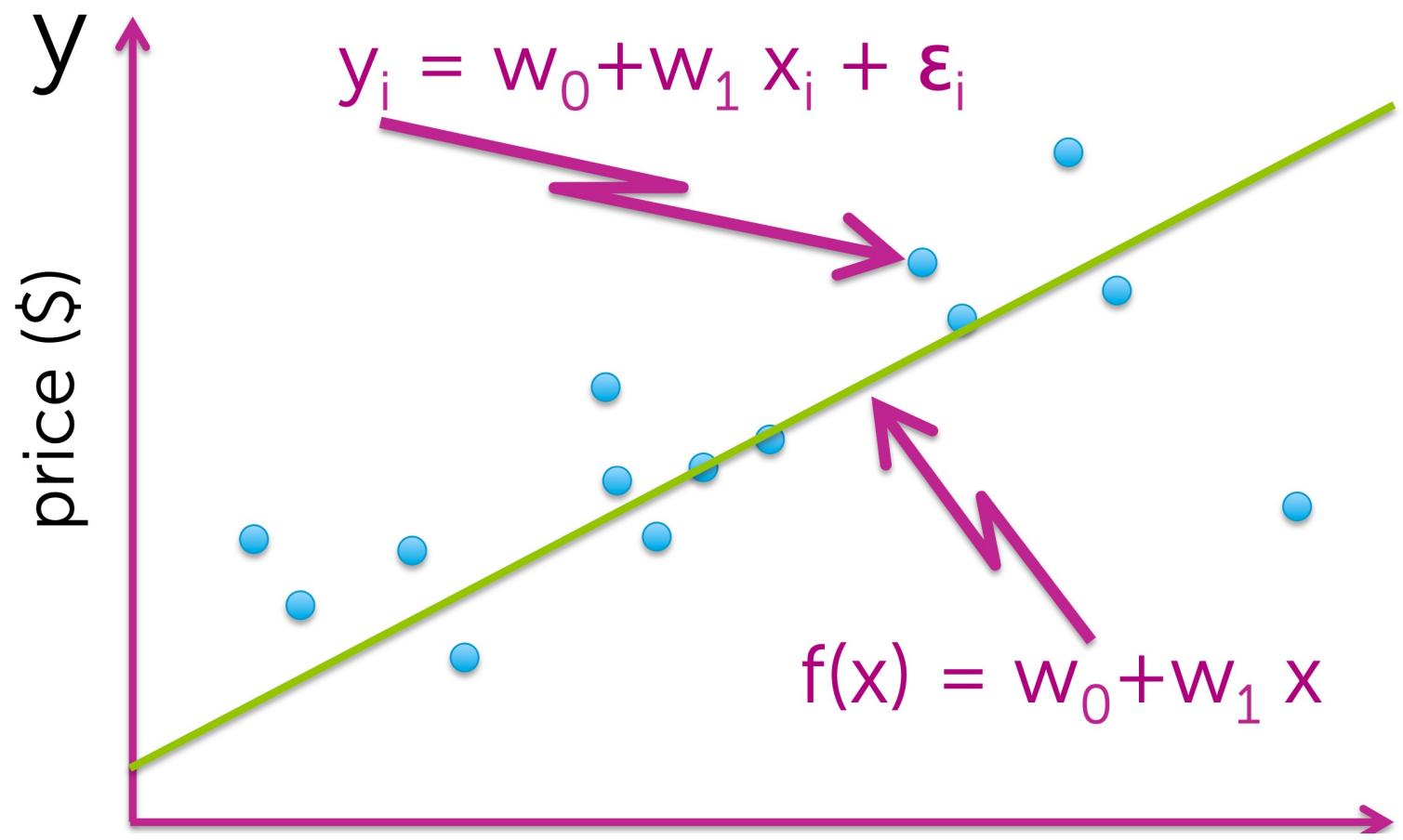
$$\begin{aligned} x_1 &= 4 \\ x_2 &= 1 \\ y &= 2 \\ \hat{y} &= -0.116 \end{aligned}$$

Like for another weight

- …

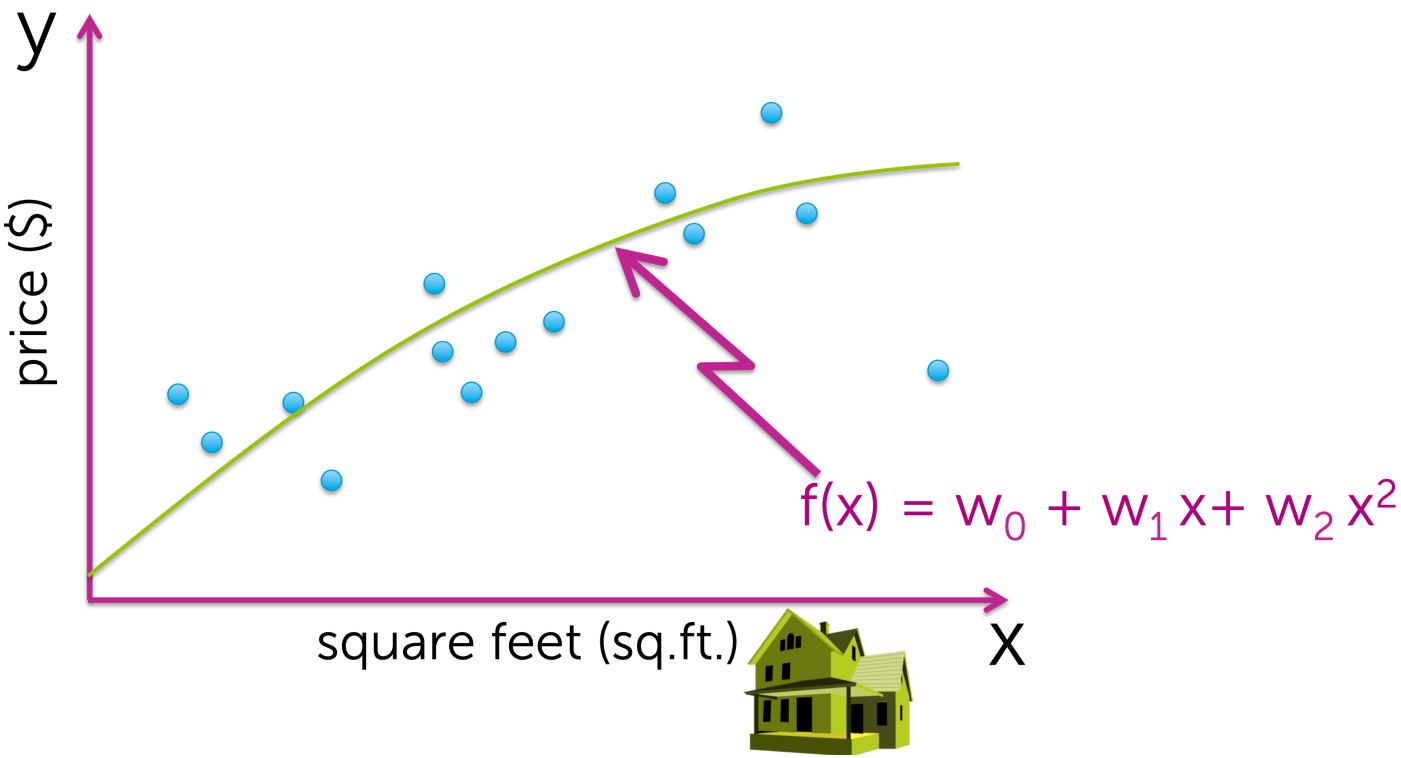
$$\hat{y} = 0.43x_1 - 0.21x_2 + 0.77$$

Simple linear regression model

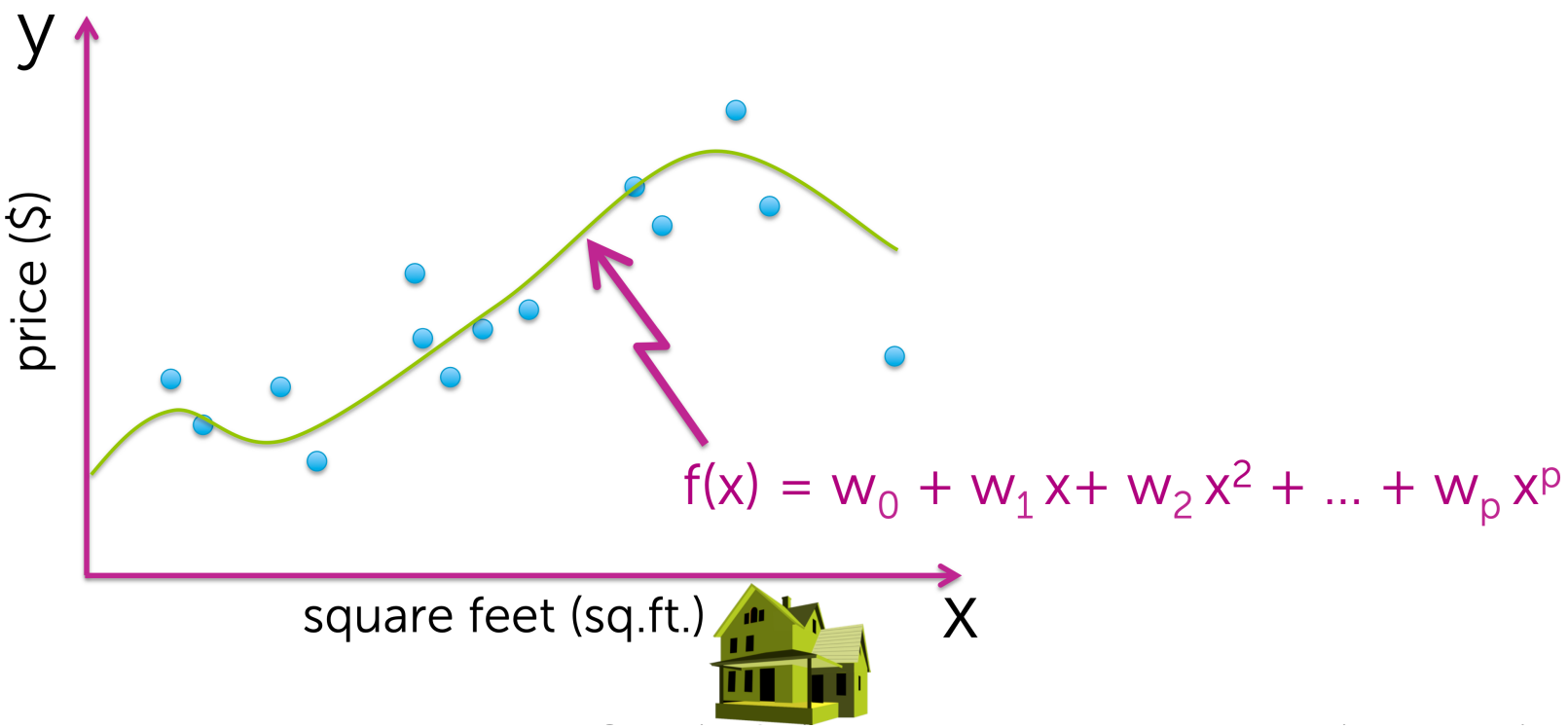


Polynomial regression

What about a quadratic function?



Even higher order polynomial



Polynomial regression

Model:

$$y_i = w_0 + w_1 x_i + w_2 x_i^2 + \ldots + w_p x_i^p + \varepsilon_i$$

treat as different **features**

feature 1 = 1 (constant) *parameter* 1 = w_0

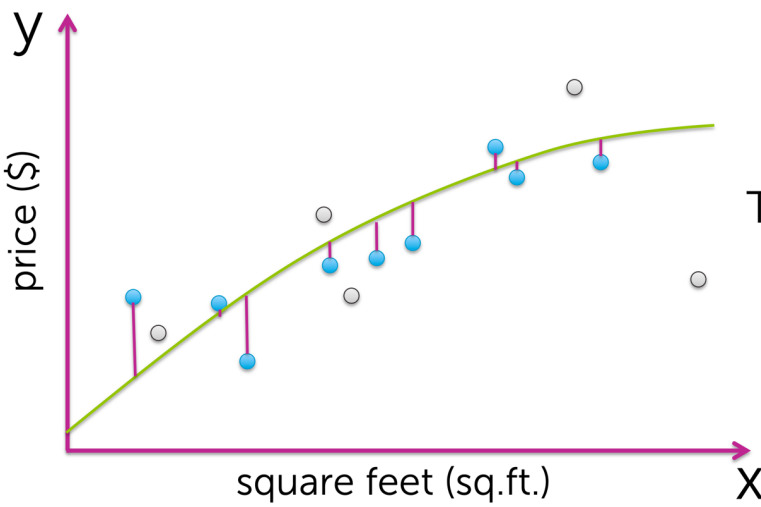
feature 2 = x *parameter* 2 = w_1

feature 3 = x^2 *parameter* 3 = w_2

... ...

feature p+1 = x^p *parameter* p+1 = w_p

Example:
Use squared error loss $(y-f_{\hat{w}}(\mathbf{x}))^2$



Training error ($\hat{w}$) = 1/N *
[($\$_{train\ 1}$-$f_{\hat{w}}(sq.ft._{train\ 1})$)^2
+ ($\$_{train\ 2}$-$f_{\hat{w}}(sq.ft._{train\ 2})$)^2
+ ($\$_{train\ 3}$-$f_{\hat{w}}(sq.ft._{train\ 3})$)^2
+ … include all
training houses]

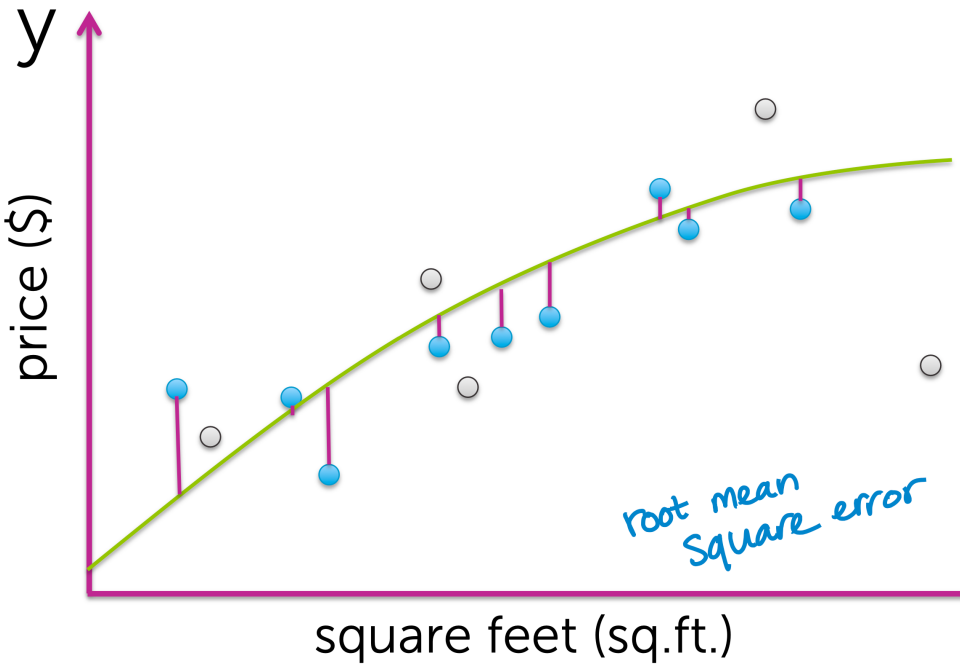
Training error

= avg. loss on houses in training set

$$= \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_{\hat{w}}(\mathbf{x}_i))$$

fit using training data

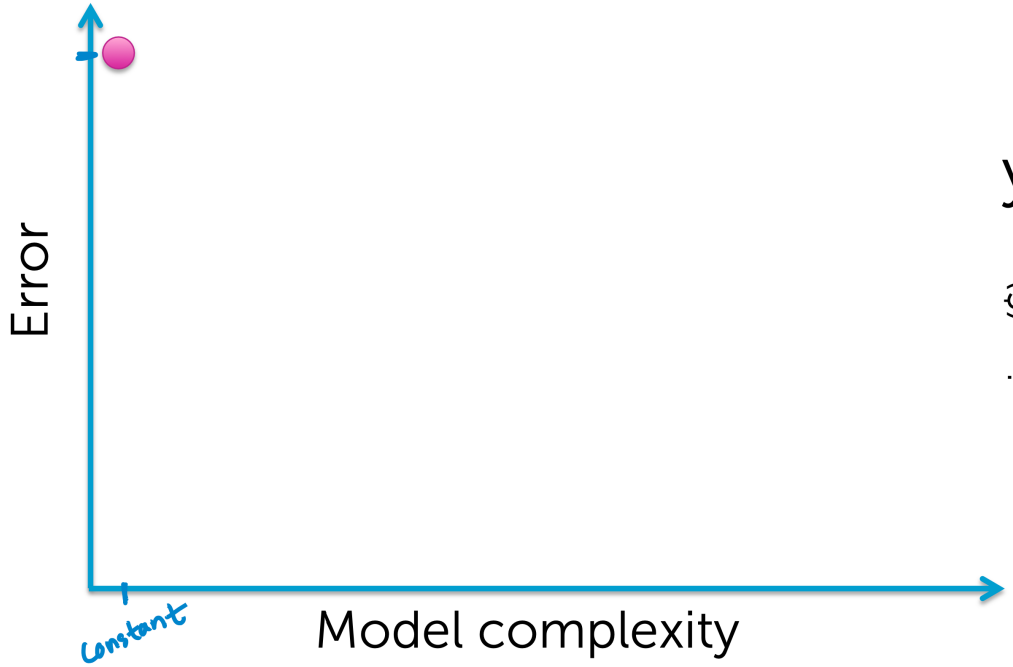
Example:
Use squared error loss $(y-f_{\hat{w}}(x))^2$

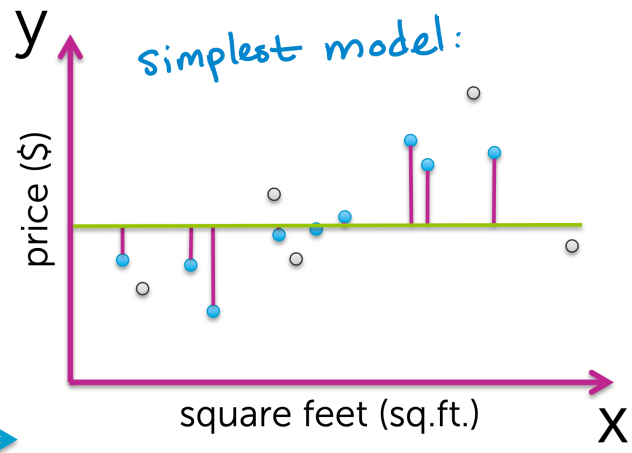


Training error ($\hat{w}$) = $\frac{1}{N}\sum_{i=1}^{N}(y_i-f_{\hat{w}}(x_i))^2$

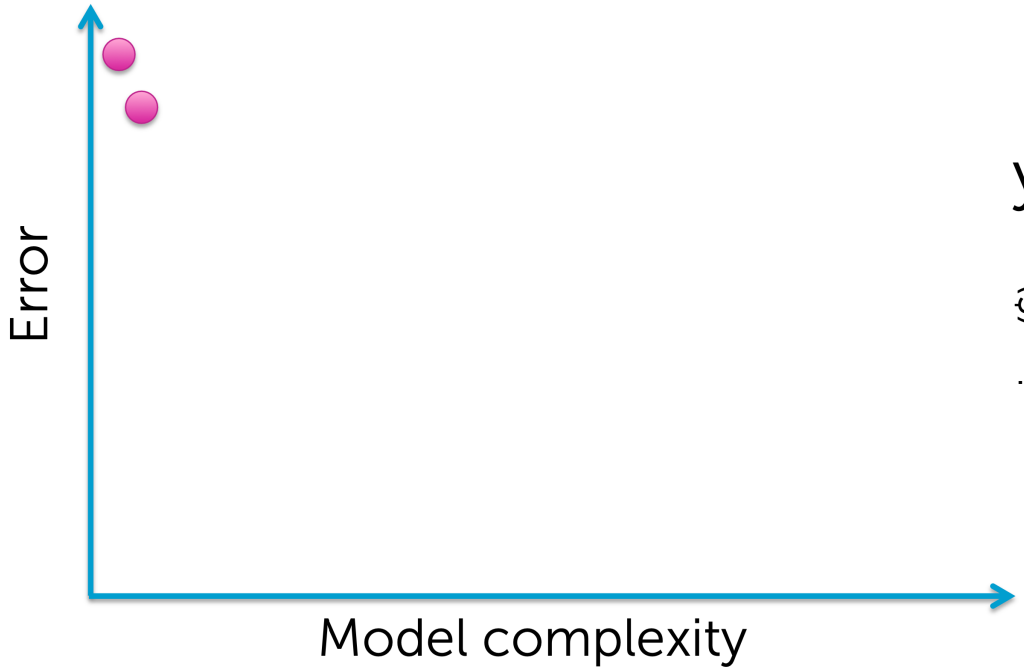
RMSE = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i-f_{\hat{w}}(x_i))^2}$

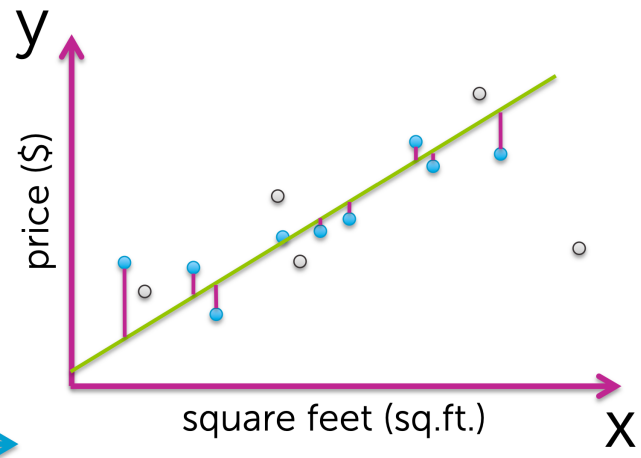
Training error vs. model complexity



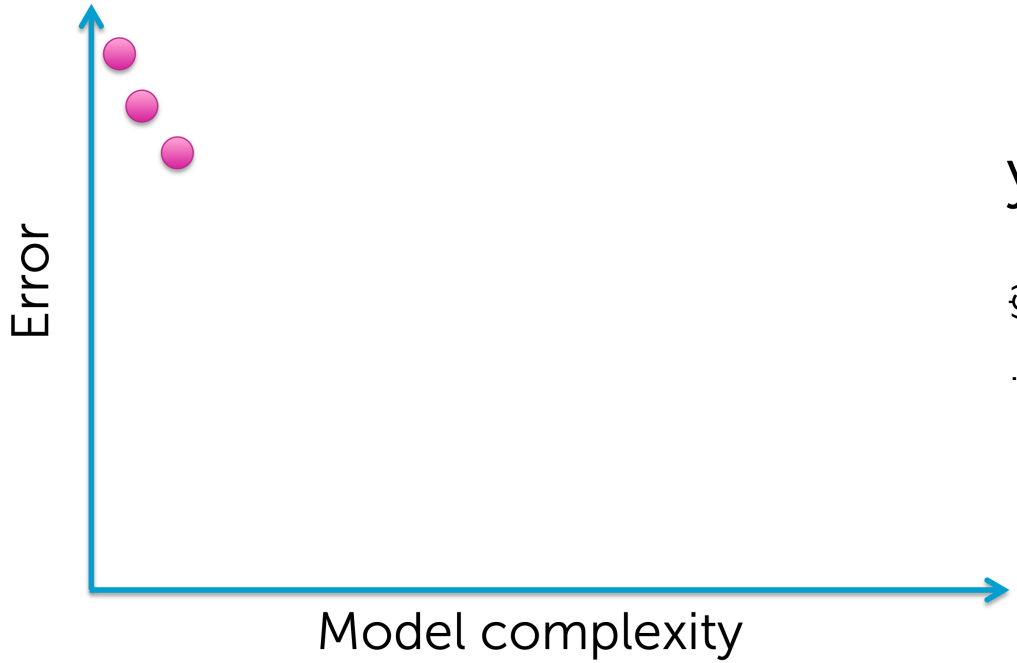


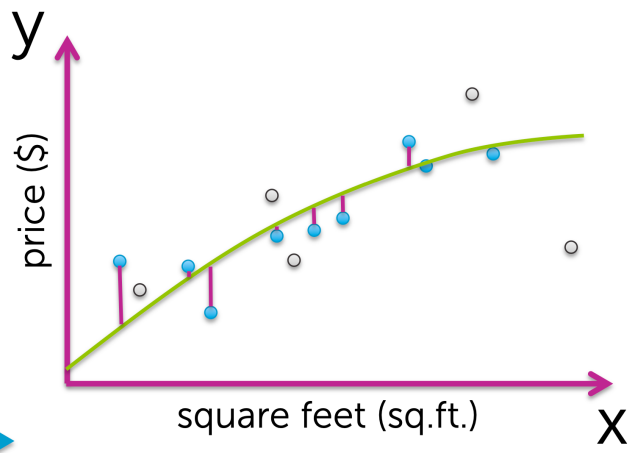
Training error vs. model complexity



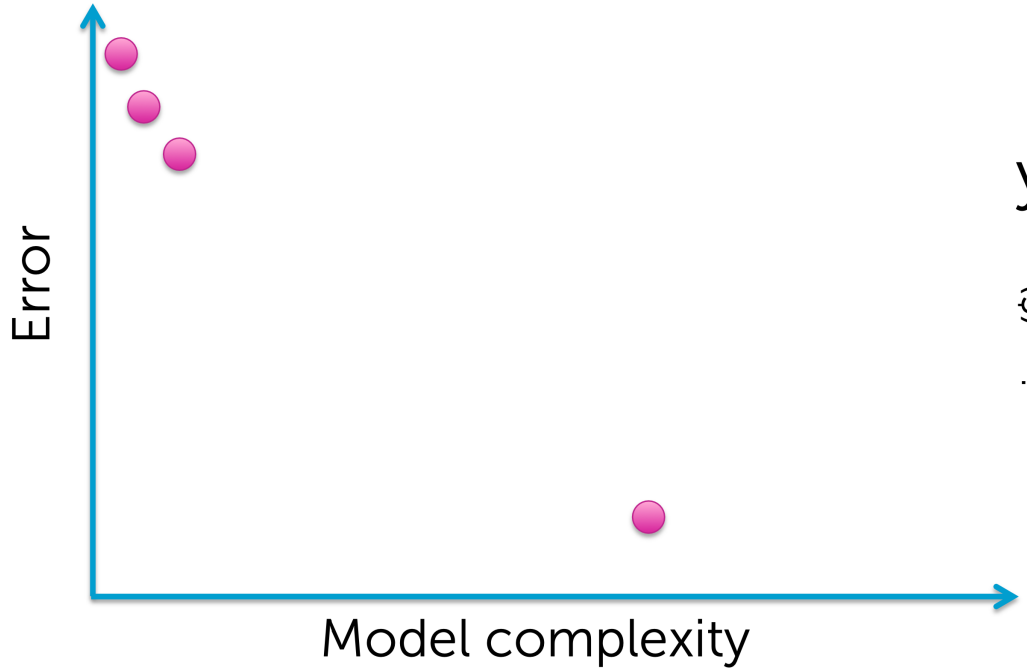


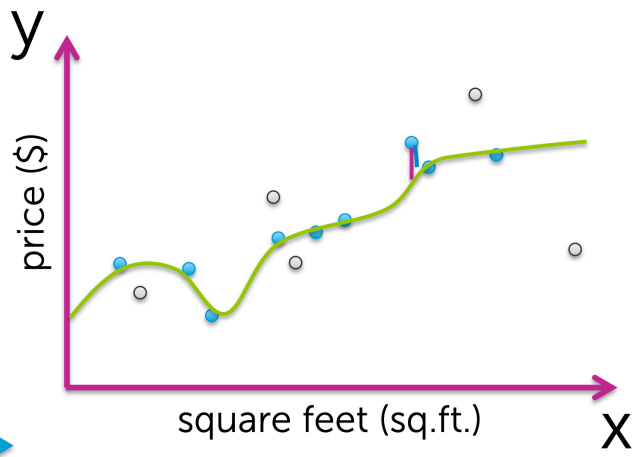
Training error vs. model complexity



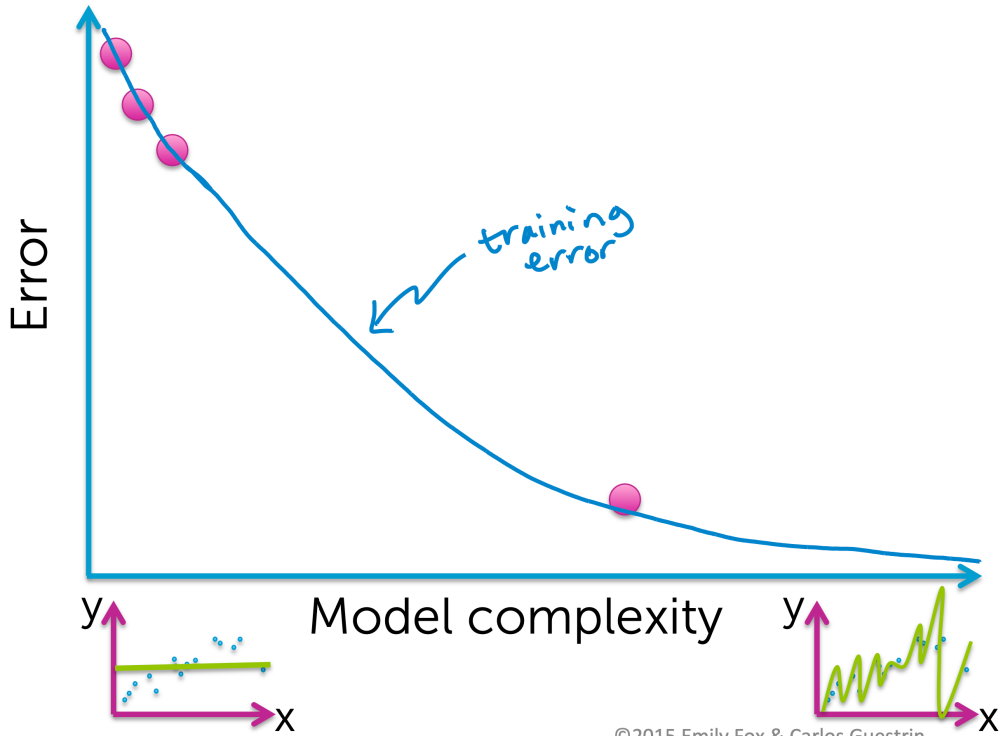


Training error vs. model complexity





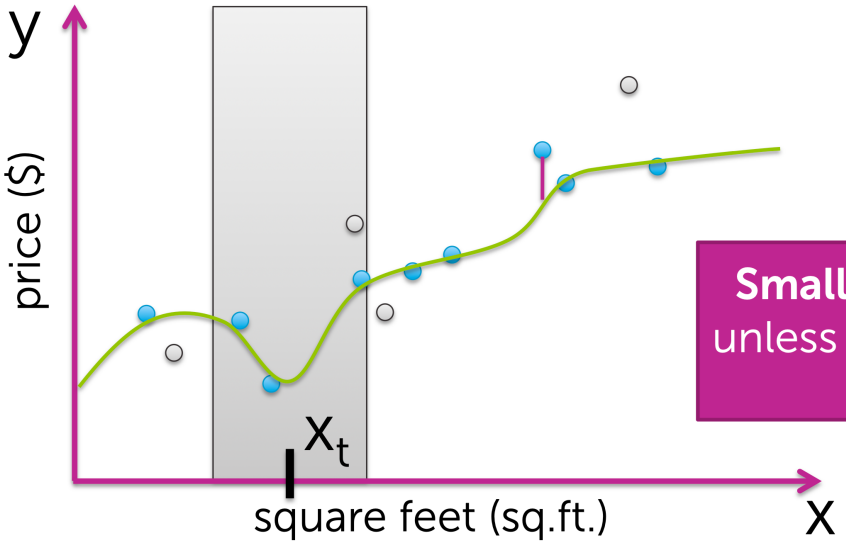
Training error vs. model complexity



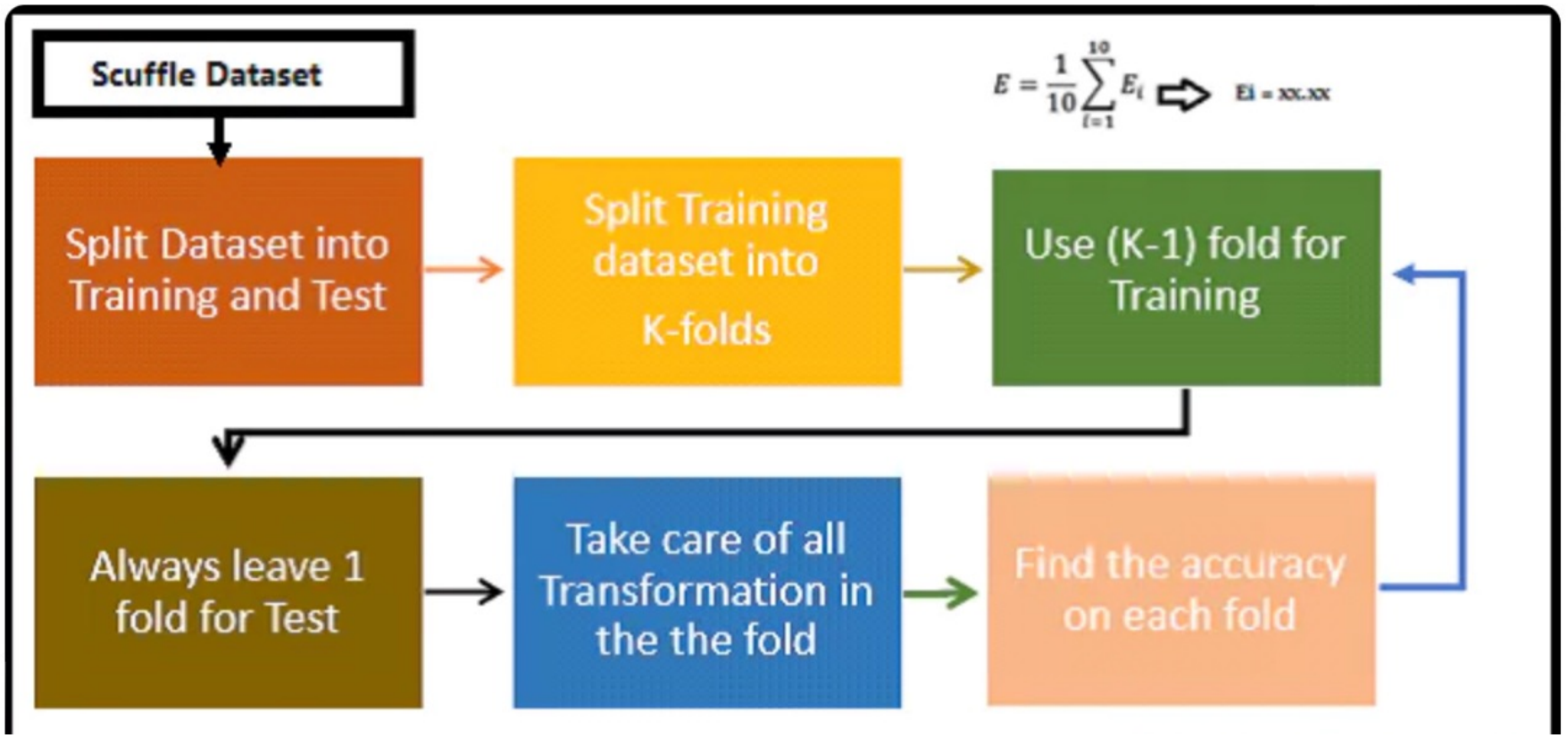
Is training error a good measure of predictive performance?

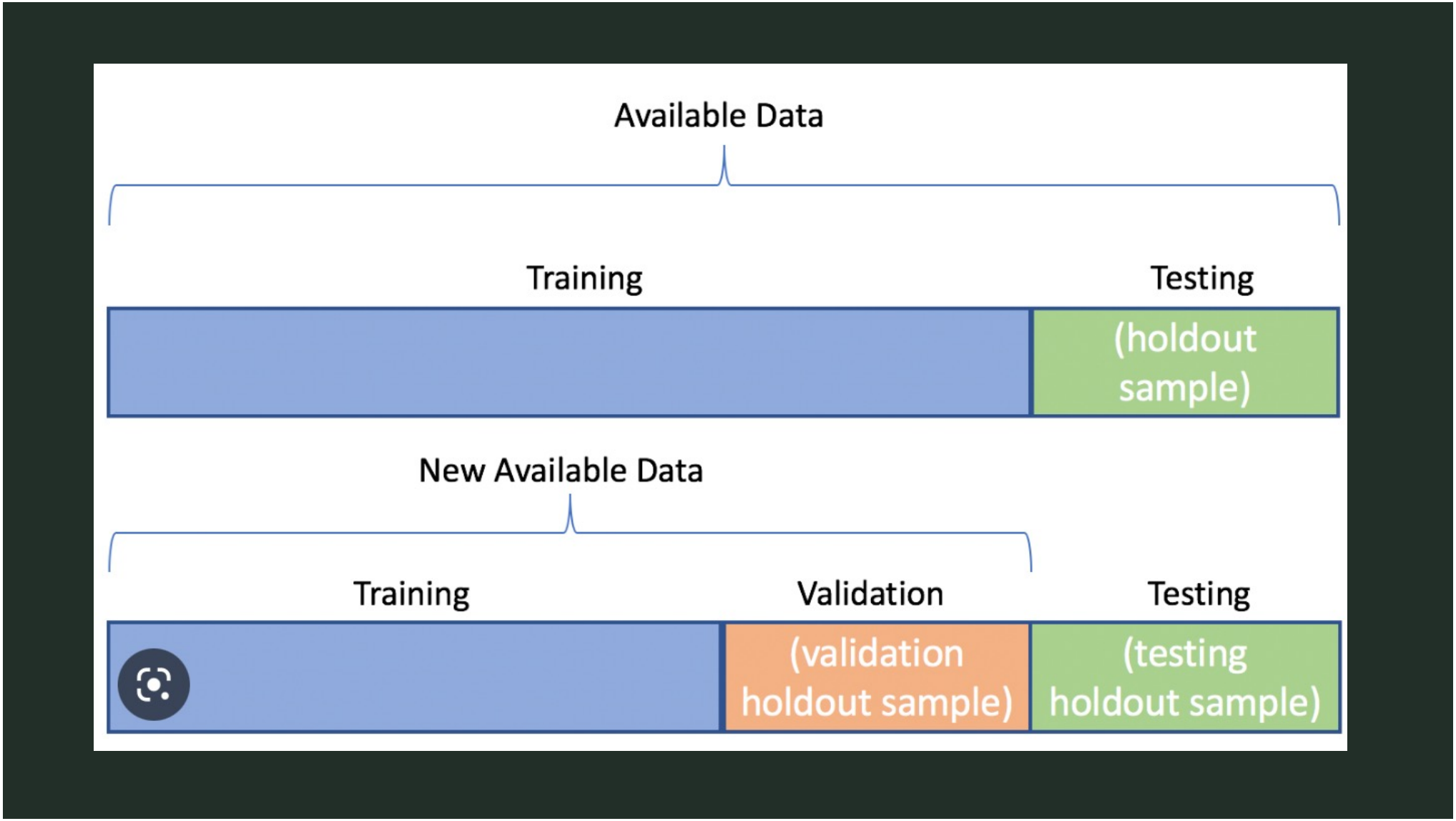
Issue: Training error is overly optimistic

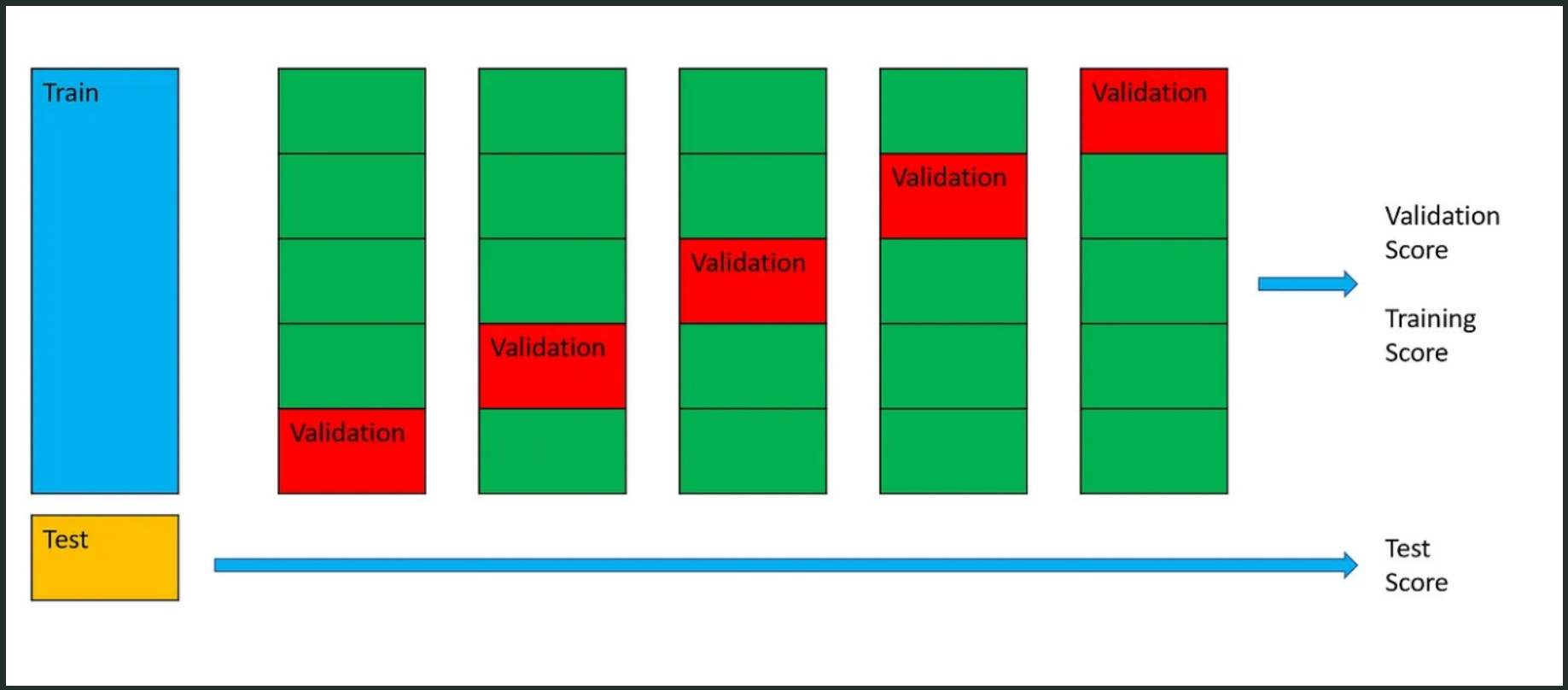
because $\hat{\mathbf{w}}$ was fit to training data



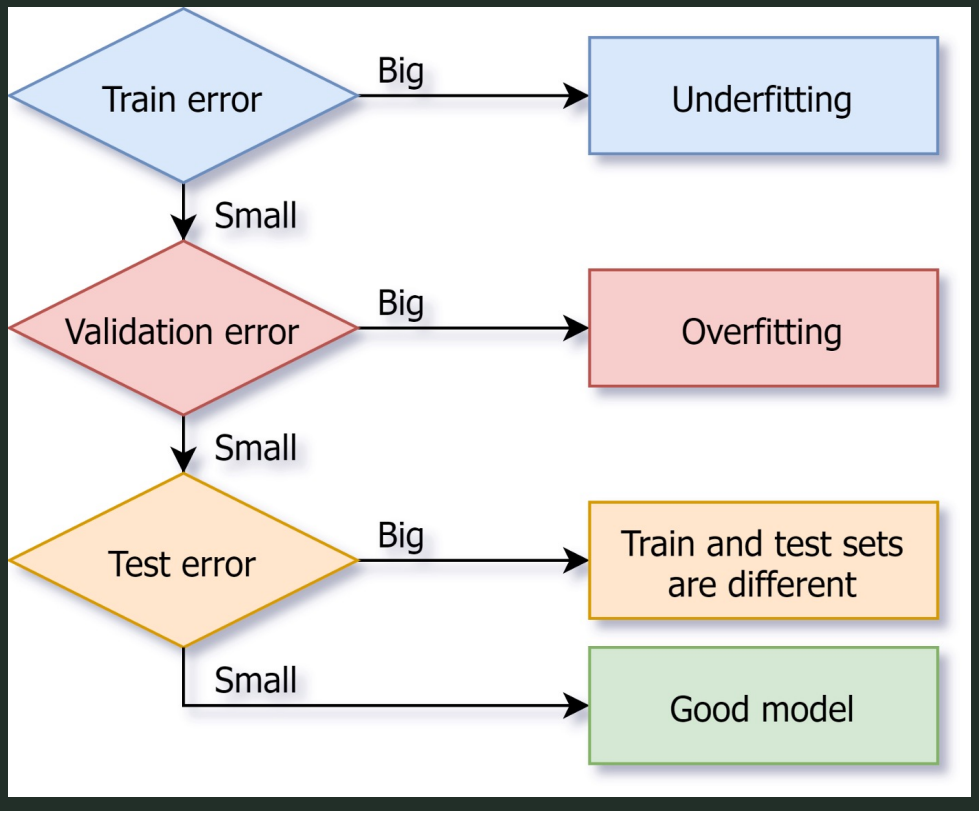
Small training error ≠> good predictions
unless training data includes everything you might ever see

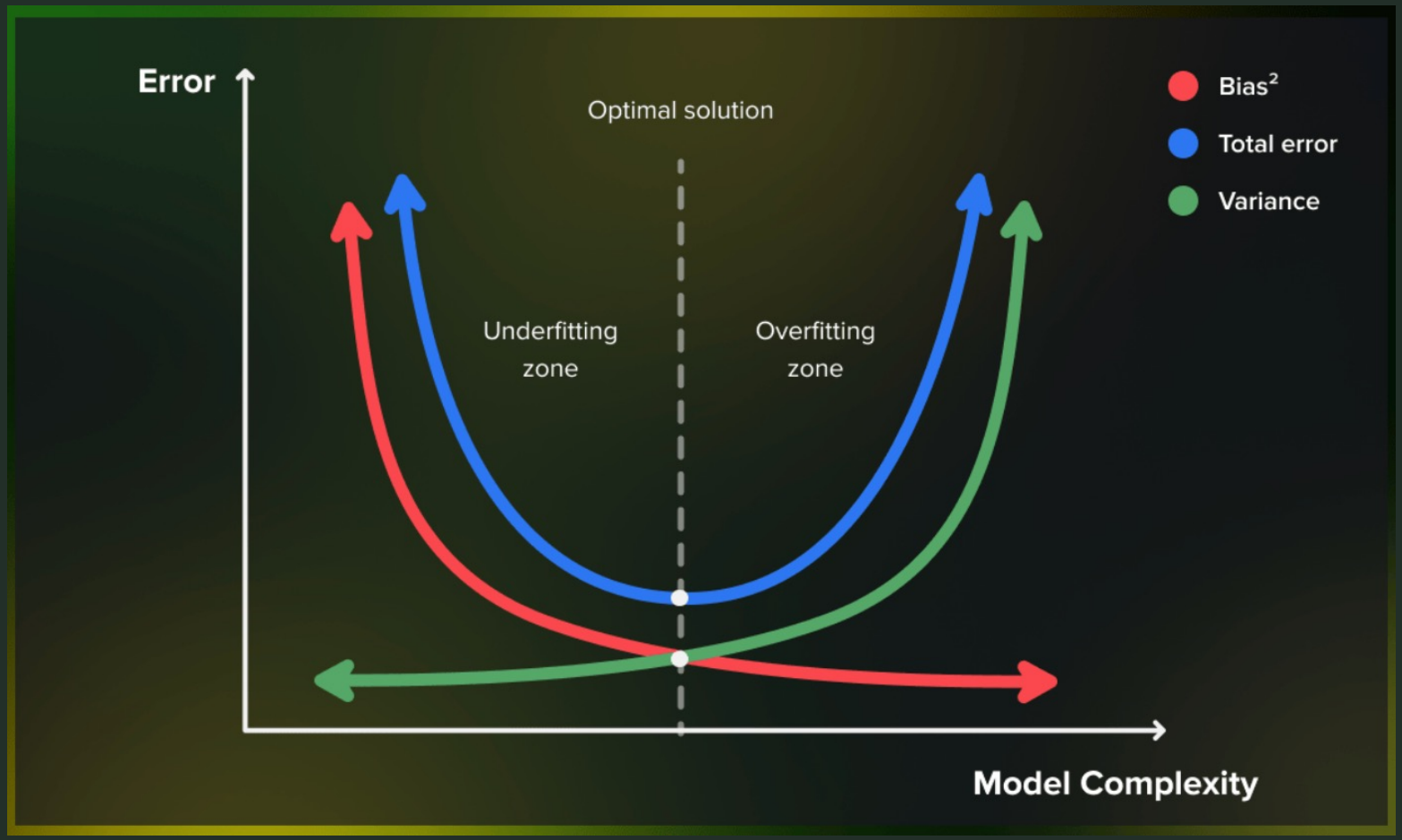

Scuffle Dataset
$E = \frac{1}{10}\sum_{i=1}^{10} E_i$
Ei = xx.xx
Split Dataset into Training and Test
Split Training dataset into K-folds
Use (K-1) fold for Training
Always leave 1 fold for Test
Take care of all Transformation in the the fold
Find the accuracy on each fold


Available Data
Training
Testing
(holdout sample)
New Available Data
Training
Validation
Testing
(validation holdout sample)
(testing holdout sample)


Train
Validation
Validation
Validation
Validation
Validation
Validation
Score
Training
Score
Test
Test
Score

Underfitting Vs Overfitting



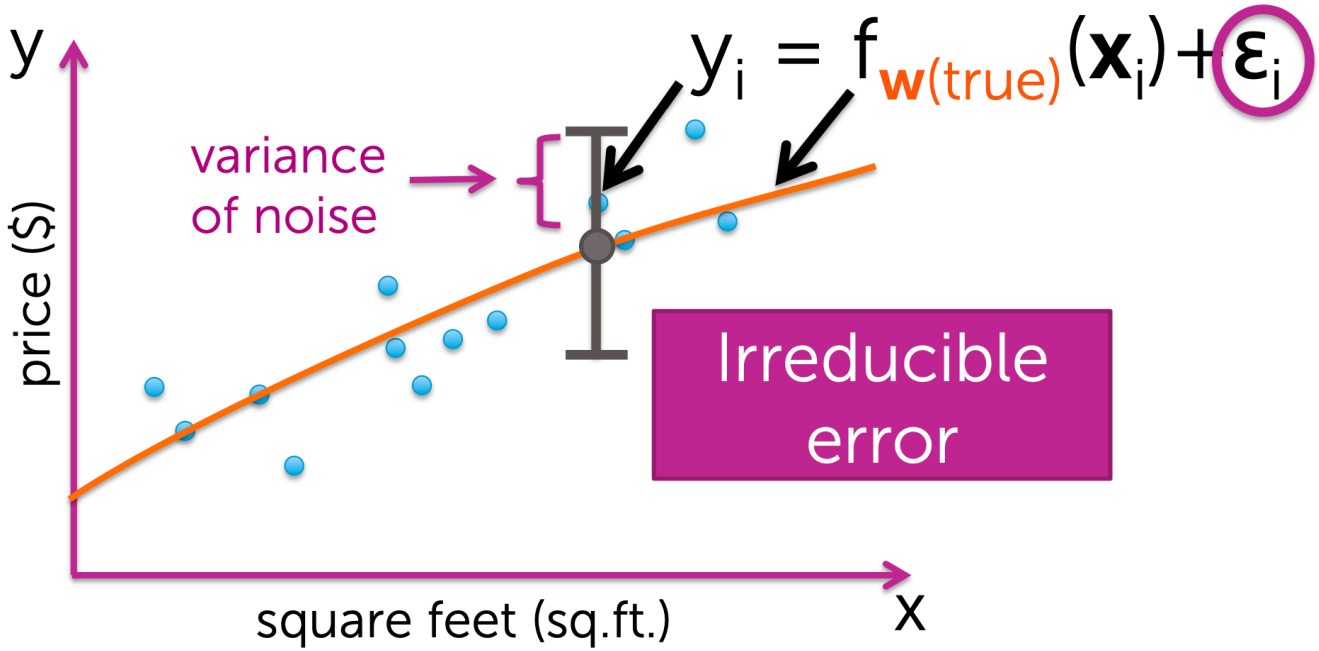

Error
Optimal solution
Bias²
Total error
Variance
Underfitting zone
Overfitting zone
Model Complexity

3 sources of error

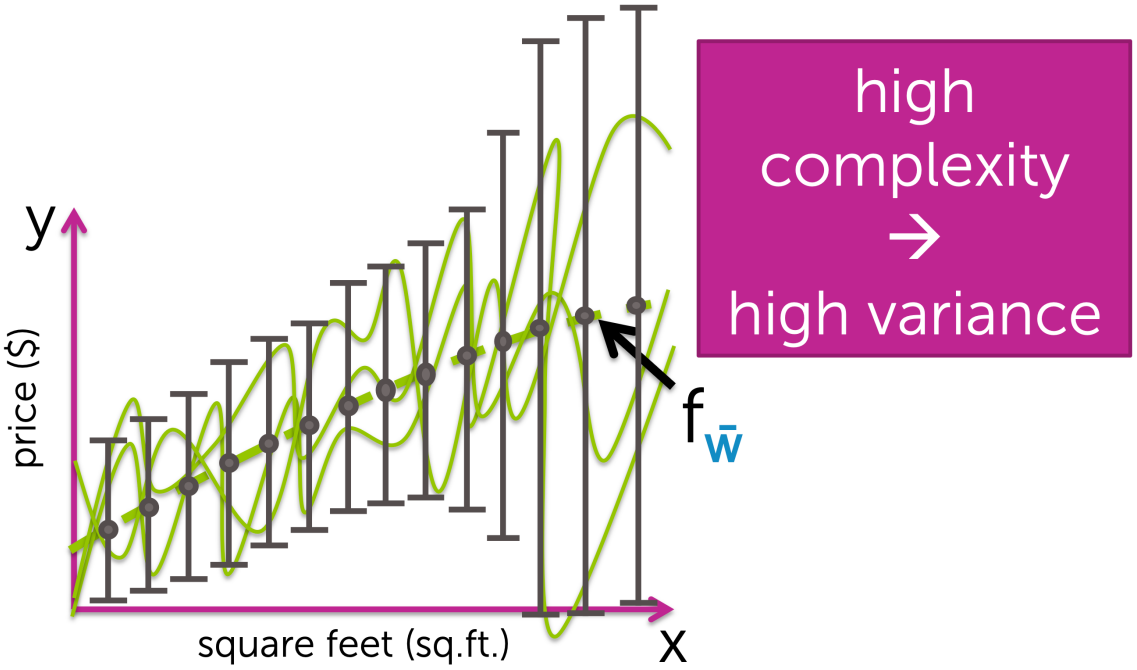
In forming predictions, there are 3 sources of error:

1. Noise
2. Bias
3. Variance

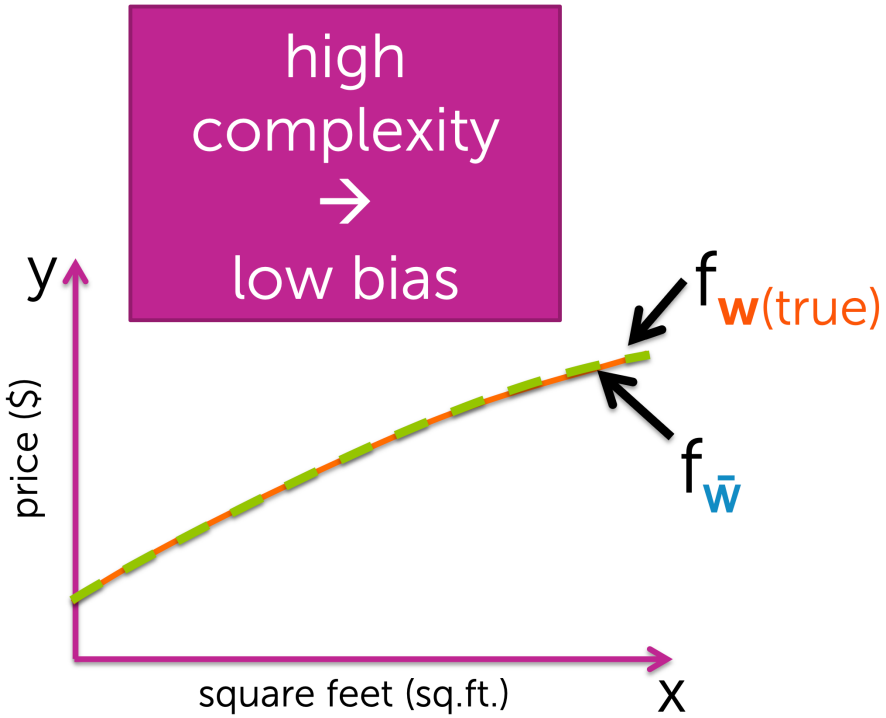
Data inherently noisy



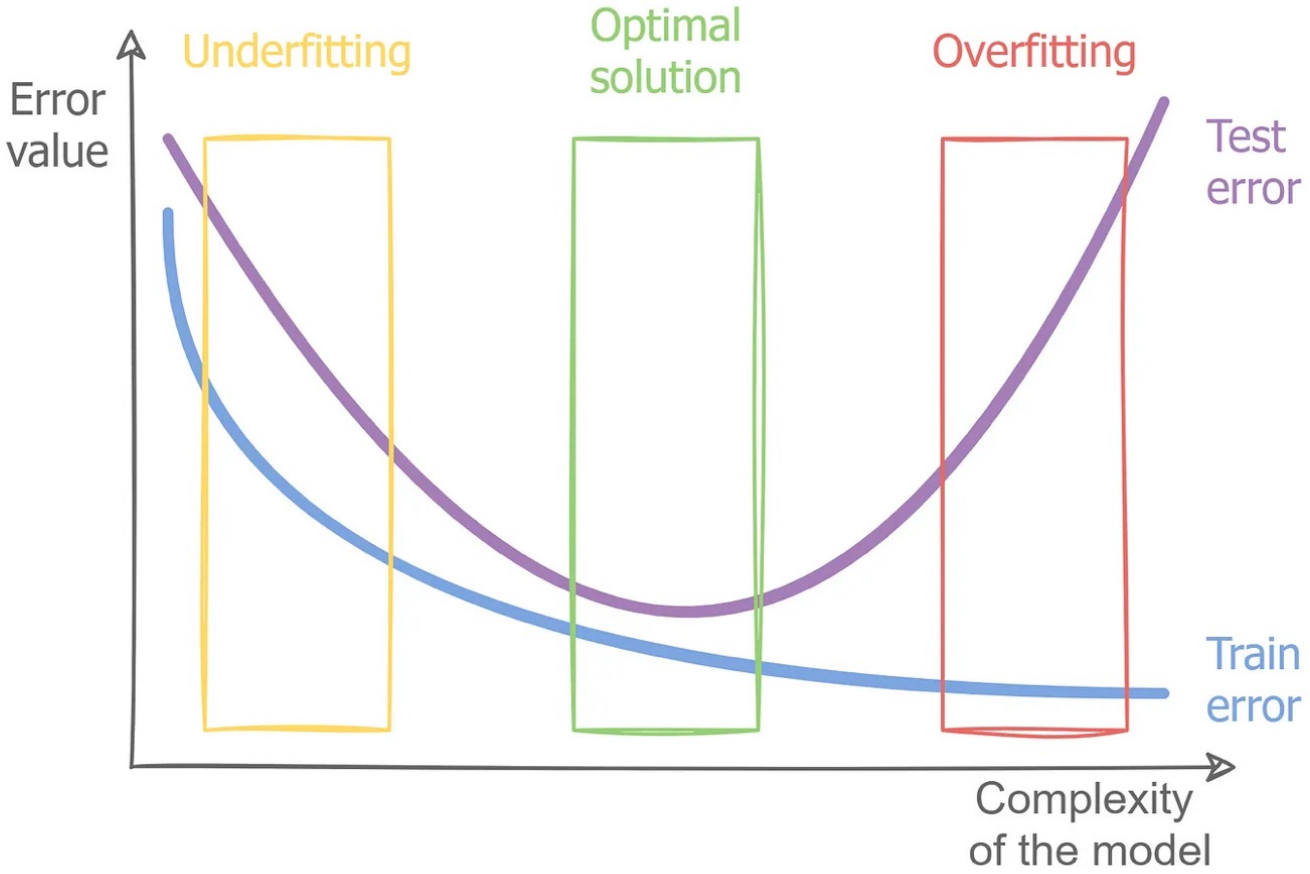
Variance of high-complexity models



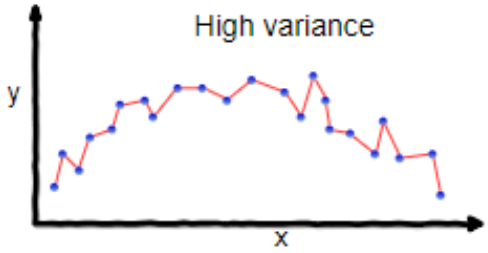
Bias of high-complexity models



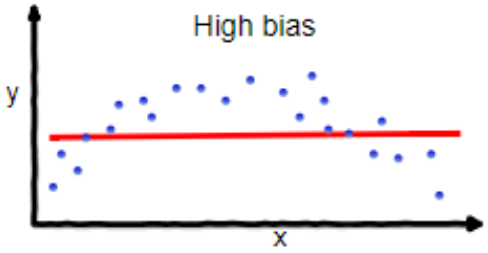
Underfitting Vs Overfitting


Underfitting
Optimal solution
Overfitting
Error value
Test error
Train error
Complexity of the model

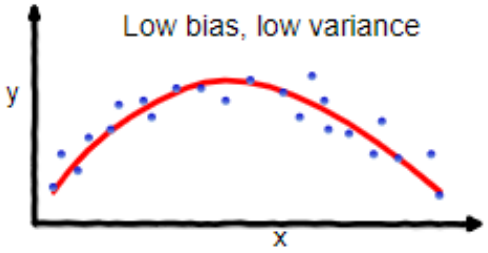
Challenges



overfitting

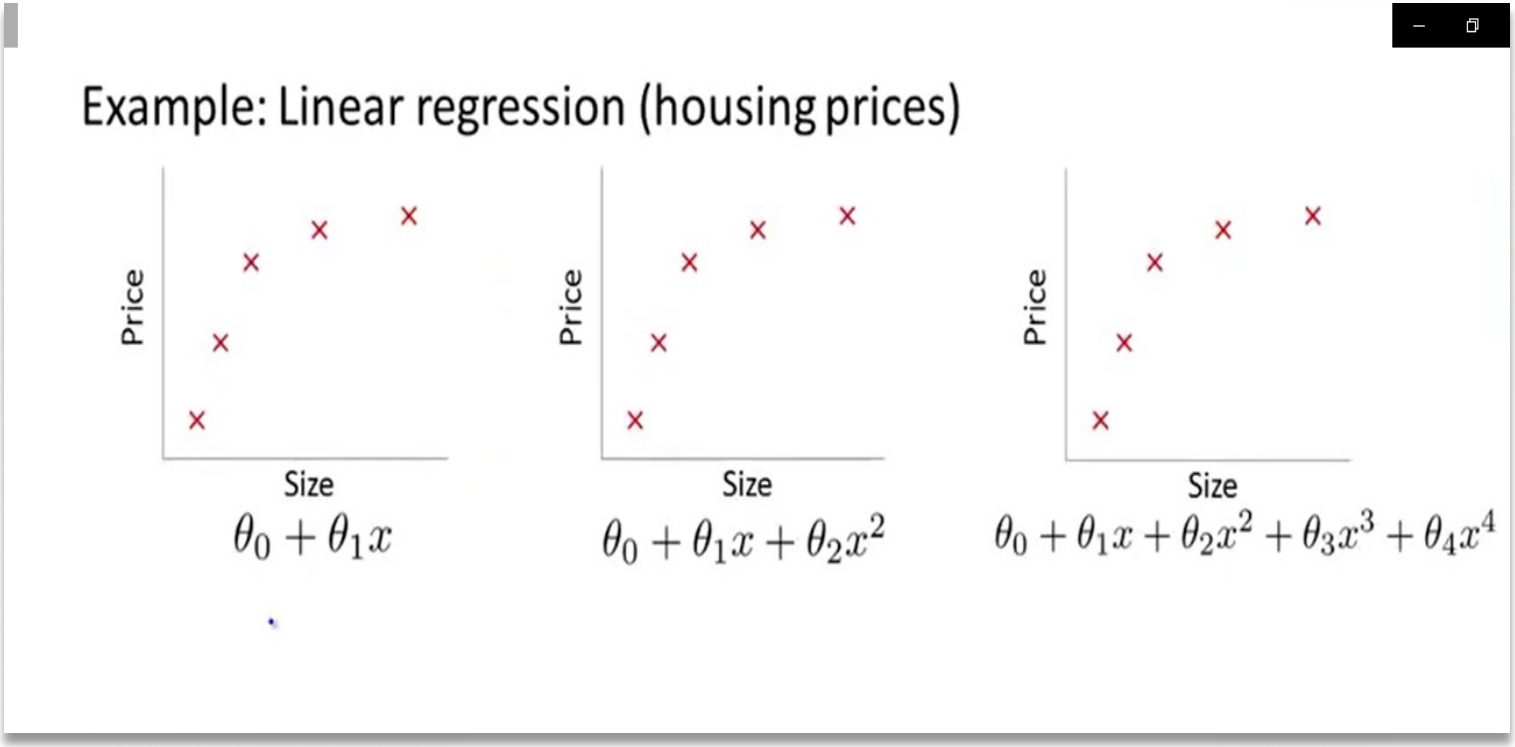


underfitting



Good balance

Underfitting Vs Overfitting in Regression



Solutions

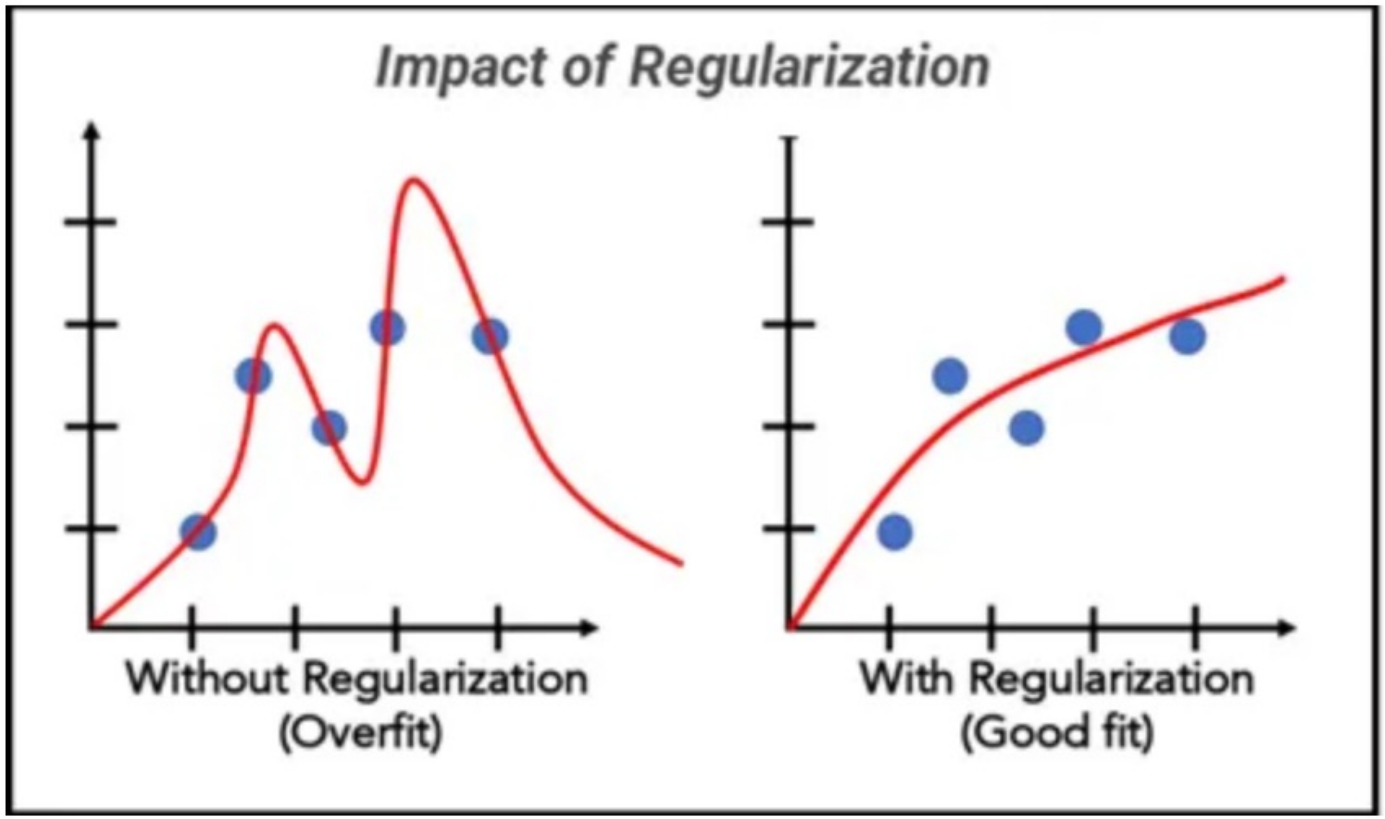
Overfitting and Underfitting are two of the main reasons machine learning models have poor performance.

Fixing **overfitting:**

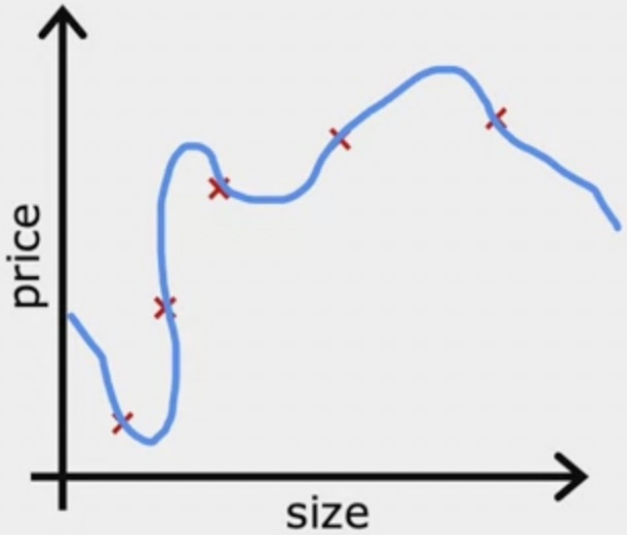
- Simplify the model (fewer parameters)
- Simplify training data (fewer attributes)
- Constrain the model (regularization)
- Use cross-validation
- Use Early stopping
- Build an ensemble
- Gather more data

Fixing **underfitting:**

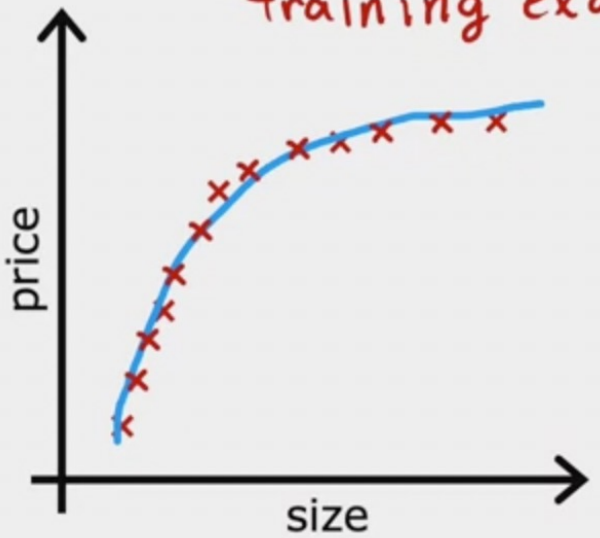
- More complex model (more parameters)
- Increase number of features
- Feature engineer should help
- Un-constrain the model (no regularization)
- Reduce noise on the data
- Train for longer


Impact of Regularization
Without Regularization
(Overfit)
With Regularization
(Good fit)

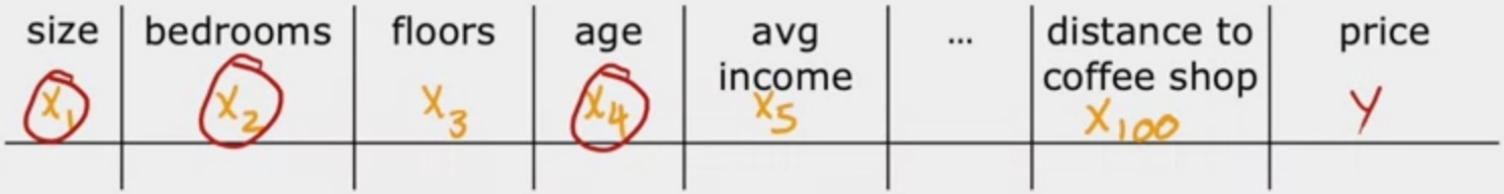
Collect more training examples



overfit



Select features to include/exclude



all features

+

insufficient data

↓

overfit

selected features

size
bedrooms
age
just right

feature selection

course 2

disadvantage

↓

useful features could be lost

Regularization

Reduce the size of parameters w_j

$$f(x) = 28x - 385x^2 + 39x^3 - 174x^4 + 100$$

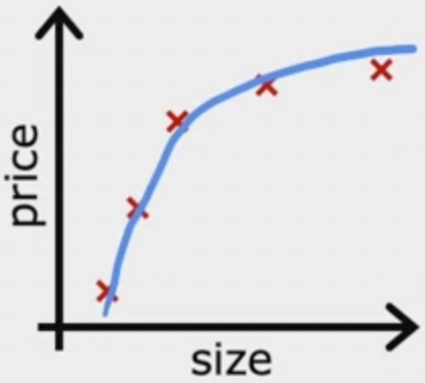
$-174x^4 \approx 0$ ← eliminate feature

large values for w_j

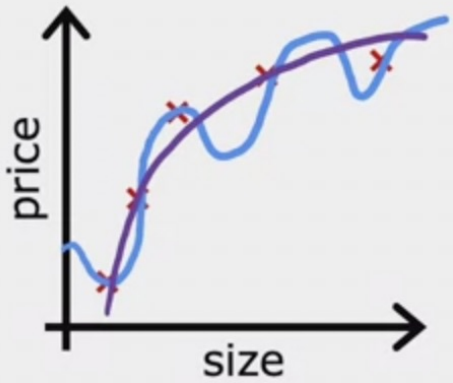
$$f(x) = 13x - 0.23x^2 + 0.000014x^3 - 0.0001\,x^4 + 10$$

small values for w_j

Intuition



$w_1 x + w_2 x^2 + b$



$w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$

make w_3, w_4 really small (≈ 0)

$$\min_{\vec{w},b} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 + 1000\, w_3^2 + 1000\, w_4^2$$

L1 Regularization(Lasso)

$$\hat{y} = w_1 x_1 + w_2 x_2 + ... + w_N x_N + b$$

$$Loss = Error(y, \hat{y})$$

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

L2 Regularization(Ridge)

$$\hat{y} = w_1 x_1 + w_2 x_2 + ... + w_N x_N + b$$

$$Loss = Error(y, \hat{y})$$

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

Elastic Net Regularization

$$Loss = Error(y, \hat{y}) + \alpha \cdot \lambda \sum_{i=1}^{N} |w_i| + (1-\alpha) \lambda \sum_{i=1}^{N} w_i^2$$