

CSE 446

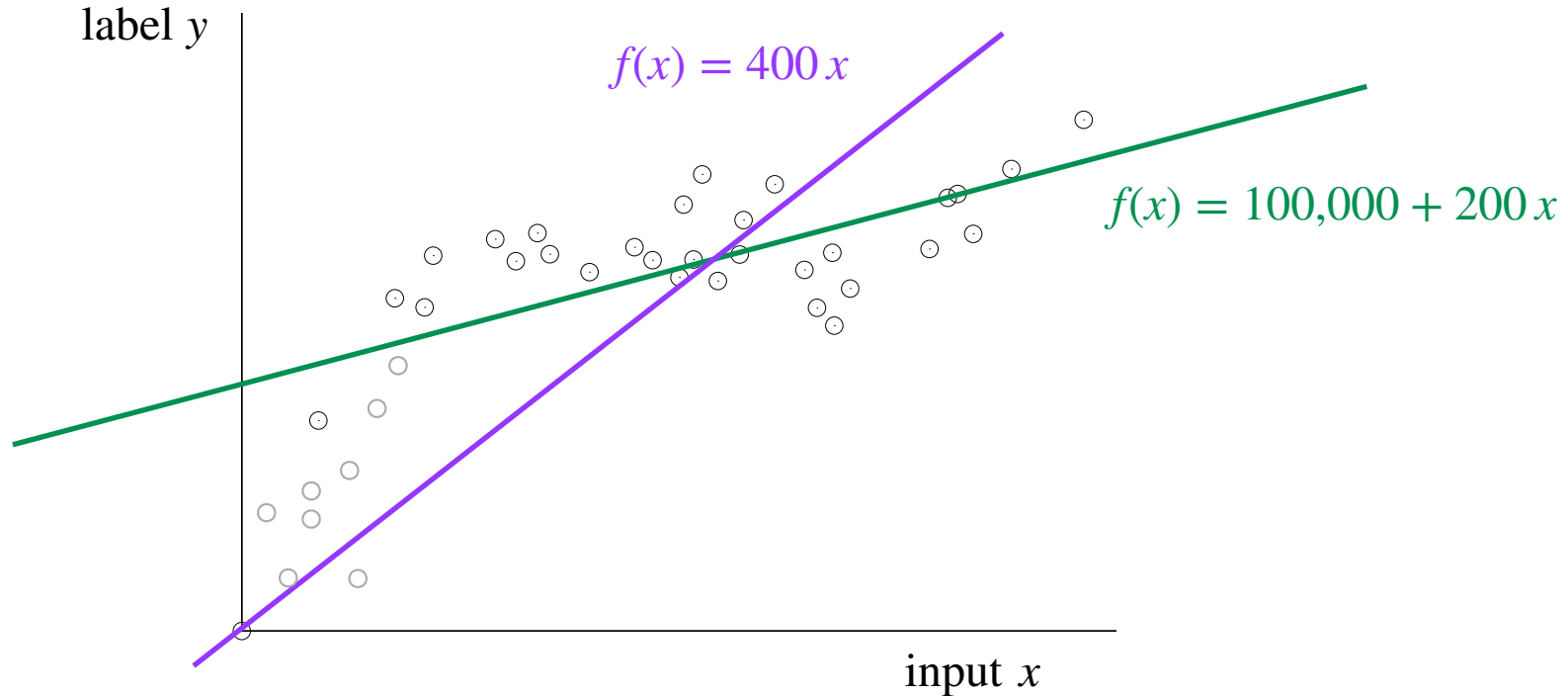
- Linear Regression w/ Basis Functions
- Model Evaluation

Natasha Jaques

Polynomial Features



Recap: Linear Regression



- In general high-dimensions, we fit a linear model with intercept $y_i \simeq w^T x_i + b$, or equivalently $y_i = w^T x_i + b + \epsilon_i$ \leftarrow with model parameters ($\underline{w} \in \mathbb{R}^d, \underline{b} \in \mathbb{R}$) that minimizes ℓ_2 -loss

$$\mathcal{L}(w, b) = \sum_{i=1}^n \underbrace{(y_i - (w^T x_i + b))^2}_{\text{error } \epsilon_i}$$

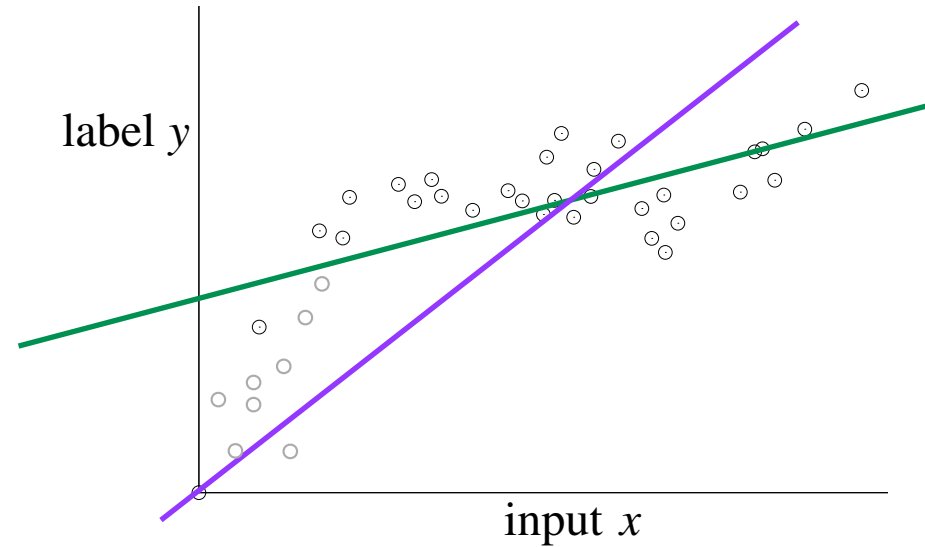
MSE

Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **Linear model with parameter (b, w_1) :**

- $\hat{y}_i = \underline{b} + \underline{w_1 x_i}$



Quadratic regression in 1-dimension

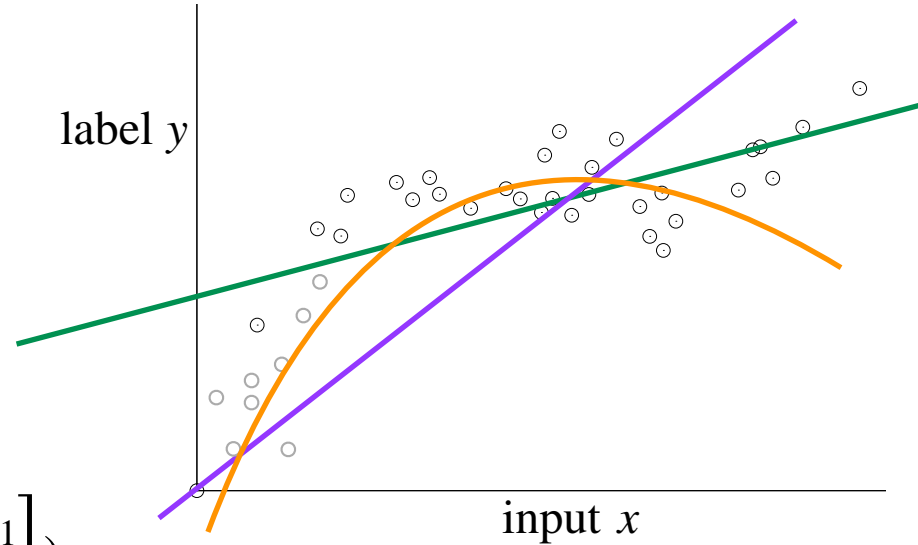
- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **Linear model with parameter (b, w_1) :**

- $\hat{y}_i = b + w_1 x_i$

- **Quadratic model with parameter $(b, w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix})$:**

- $\hat{y}_i = b + w_1 x_i + w_2 x_i^2$



$$h(x_i) = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \end{bmatrix} \in \mathbb{R}^3$$

$$\hat{y} = h(x_i)^T \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix}$$

w

// still linear regression

Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **Linear model with parameter (b, w_1) :**

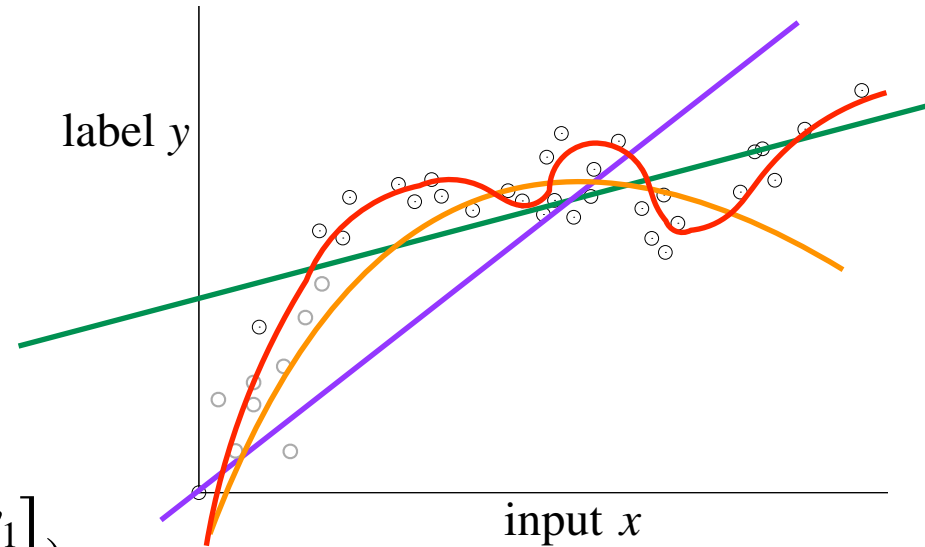
- $\hat{y}_i = b + w_1 x_i$

- **Quadratic model with parameter $(b, w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix})$:**

- $\hat{y}_i = b + w_1 x_i + w_2 x_i^2$

- **Degree-p polynomial model with parameter $(b, w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix})$:**

- $\hat{y}_i = b + w_1 x_i + w_2 x_i^2 + \dots + w_p x_i^p$



Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **Linear model with parameter (b, w_1) :**

- $\hat{y}_i = b + w_1 x_i$

- **Quadratic model with parameter $(b, w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix})$:**

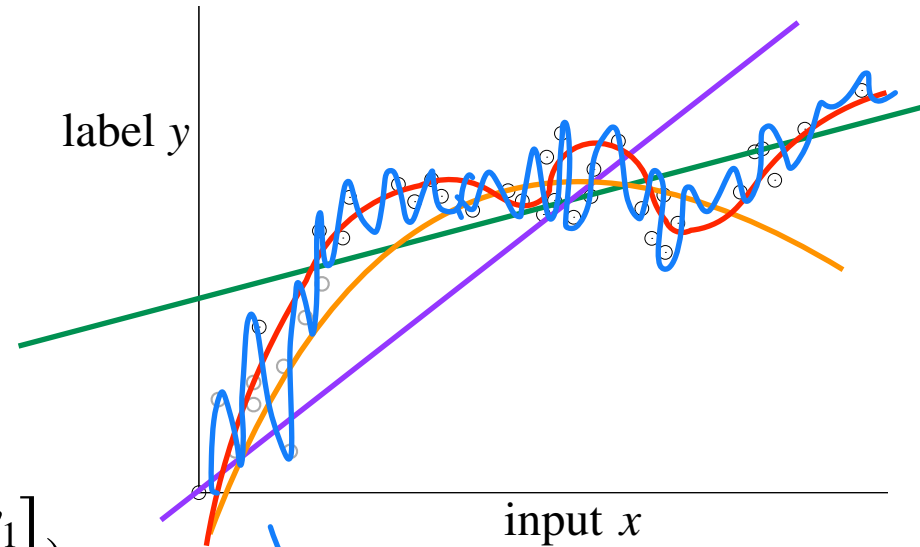
- $\hat{y}_i = b + w_1 x_i + w_2 x_i^2$

- **Degree-p polynomial model with parameter $(b, w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix})$:**

- $\hat{y}_i = b + w_1 x_i + w_2 x_i^2 + \dots + w_p x_i^p$

- **General p-features with parameter $w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$:**

- $\hat{y}_i = \langle w, h(x_i) \rangle$ where $h : \mathbb{R} \rightarrow \mathbb{R}^p$



overfitting
random
fluctuations
(measurement
error) in
training data

Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **General p-features with parameter** $w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$:

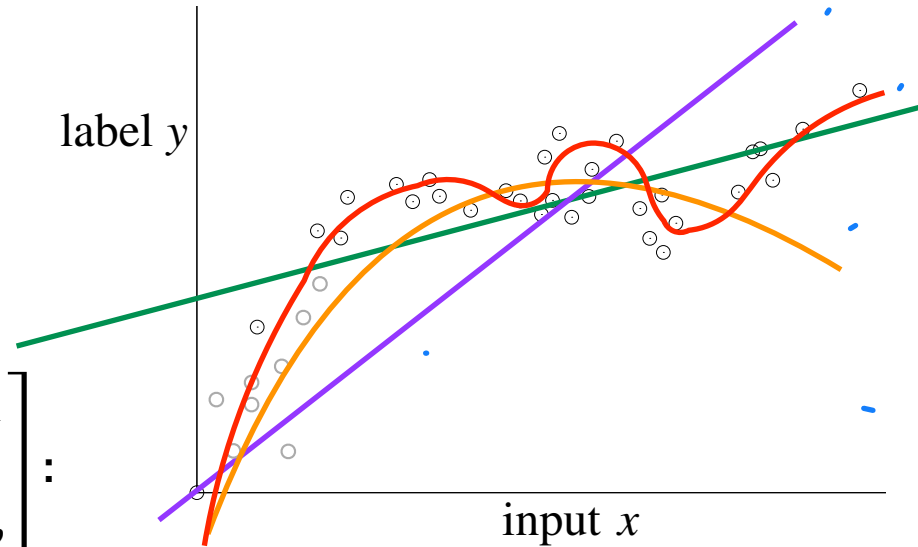
- $\hat{y}_i = \langle w, h(x_i) \rangle$ where $h : \mathbb{R} \rightarrow \mathbb{R}^p$

Note: h can be arbitrary non-linear functions!

$$h(x) = \left[\log(x), x^2, \sin(x), \sqrt{x} \right]^T$$

↳ transform features however

$\sin(x)$? cyclic



zip code?
↳ lat/long

PLOT

Quadratic regression in 1-dimension

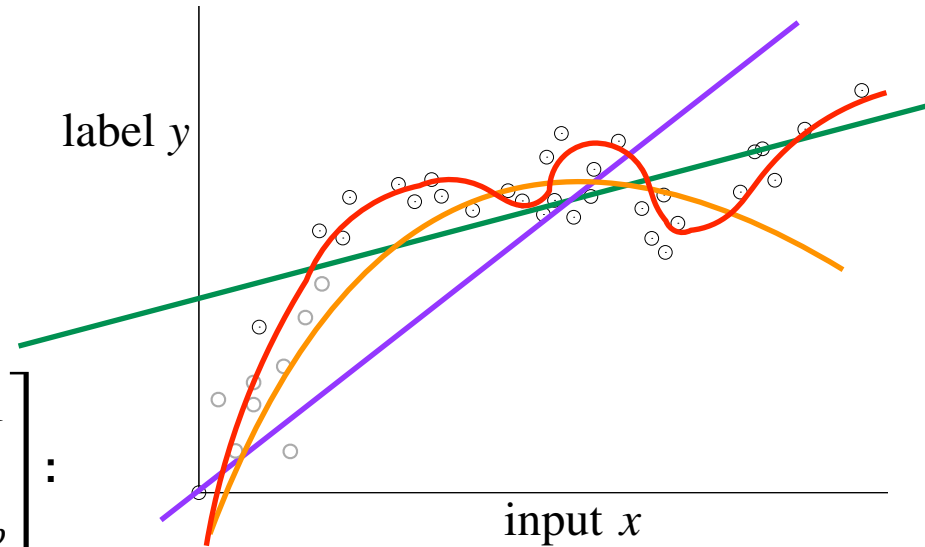
- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **General p-features with parameter** $w =$

$$\begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix} :$$

- $\hat{y}_i = \langle w, h(x_i) \rangle$ where $h : \mathbb{R} \rightarrow \mathbb{R}^p$

How do we learn w ?



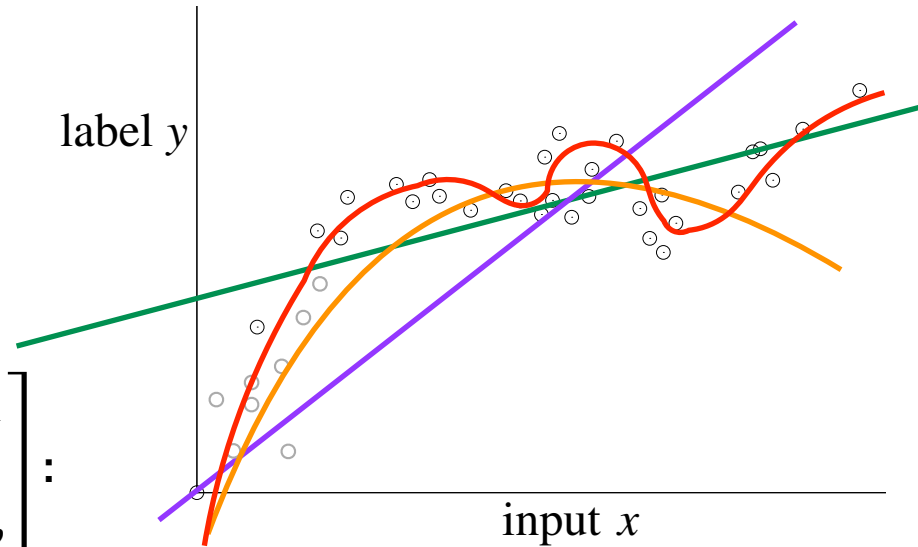
Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **General p-features with parameter $w =$**

$$\begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix} :$$

- $\hat{y}_i = \langle w, h(x_i) \rangle$ where $h : \mathbb{R} \rightarrow \mathbb{R}^p$



How do we learn w ?

$$\mathbf{H} = \begin{bmatrix} \text{---} h(x_1)^\top \text{---} \\ \vdots \\ \text{---} h(x_n)^\top \text{---} \end{bmatrix} \in \mathbb{R}^{n \times p}$$

$$\hat{w} = \arg \min_w \|\mathbf{H}w - \mathbf{y}\|_2^2$$

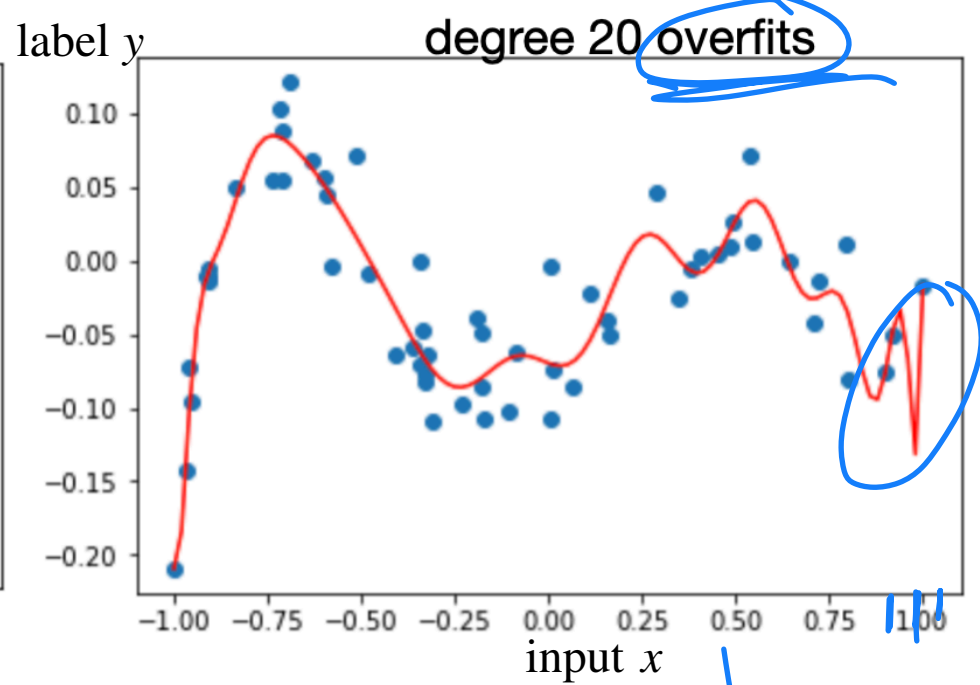
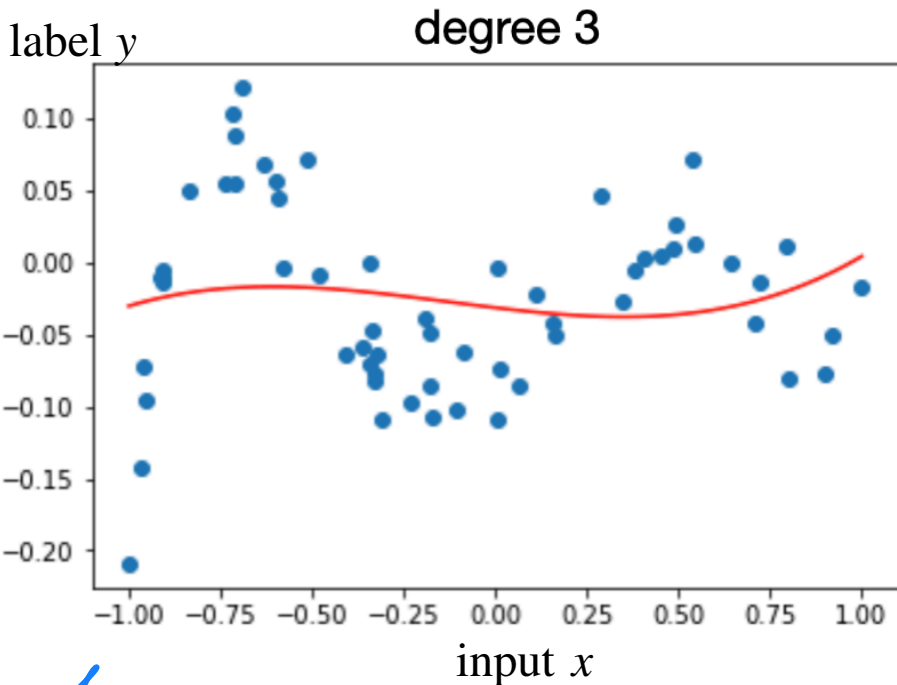
For a new test point x , predict
 $\hat{y} = \langle \hat{w}, h(x) \rangle$

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

Which p should we choose?

- First instance of class of models with different representation power = model complexity

UNDERFIT



- How do we determine which is better model?

not fitting training data

fitting "idiosyncrasy" of training data

Generalization



test

- we say a predictor **generalizes** if it performs as well on unseen data as on training data (we will formalize the next lecture)
- the data used to train a predictor is **training data** or **in-sample data**
- we want the predictor to work on **out-of-sample data**
- we say a predictor **fails to generalize** if it performs well on in-sample data but does not perform well on out-of-sample data

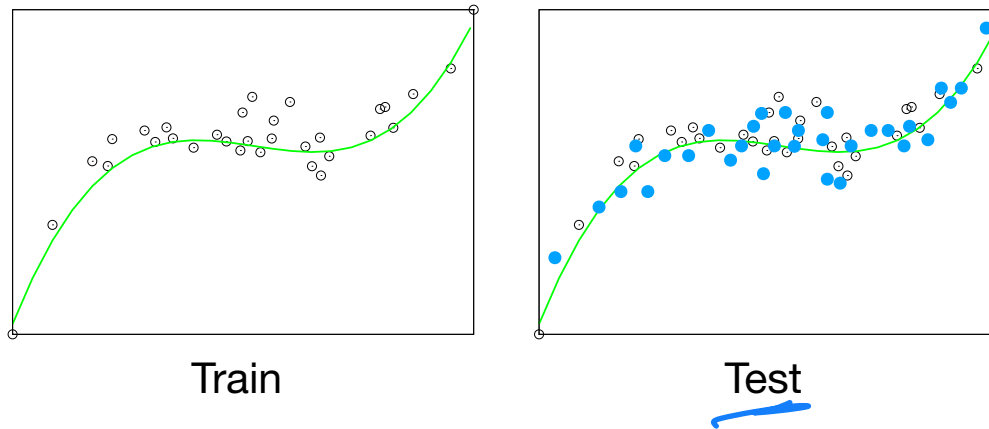
Generalization

- we say a predictor **generalizes** if it performs as well on unseen data as on training data (we will formalize the next lecture)
- the data used to train a predictor is **training data** or **in-sample data**
- we want the predictor to work on **out-of-sample data**
- we say a predictor **fails to generalize** if it performs well on in-sample data but does not perform well on out-of-sample data

Generalization

Consider the following:

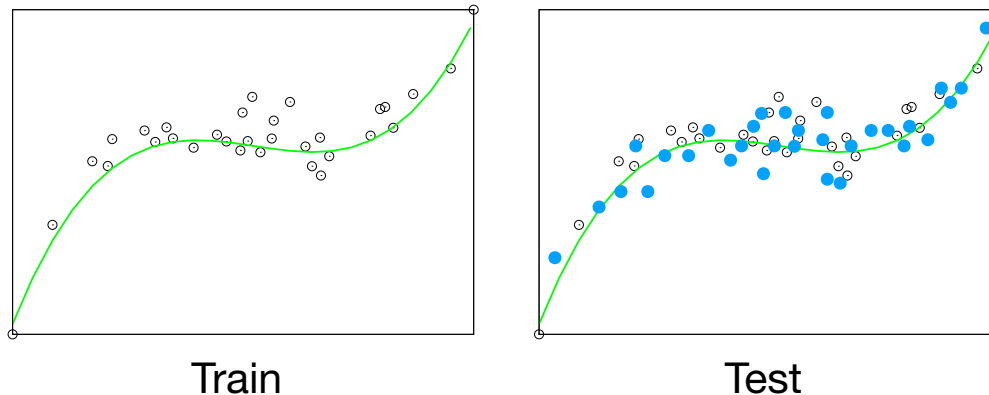
- **train** a cubic predictor on 32 (**in-sample**) white circles: Mean Squared Error (MSE) 174
- **predict** label y for 30 (**out-of-sample**) blue circles: MSE 192
- **Would you conclude this predictor generalizes effectively?**



Generalization

Consider the following:

- **train** a cubic predictor on 32 (**in-sample**) white circles: Mean Squared Error (MSE) 174
- **predict** label y for 30 (**out-of-sample**) blue circles: MSE 192
- **Would you conclude this predictor generalizes effectively?**
 - **Yes**, as in-sample MSE \simeq out-of-sample MSE



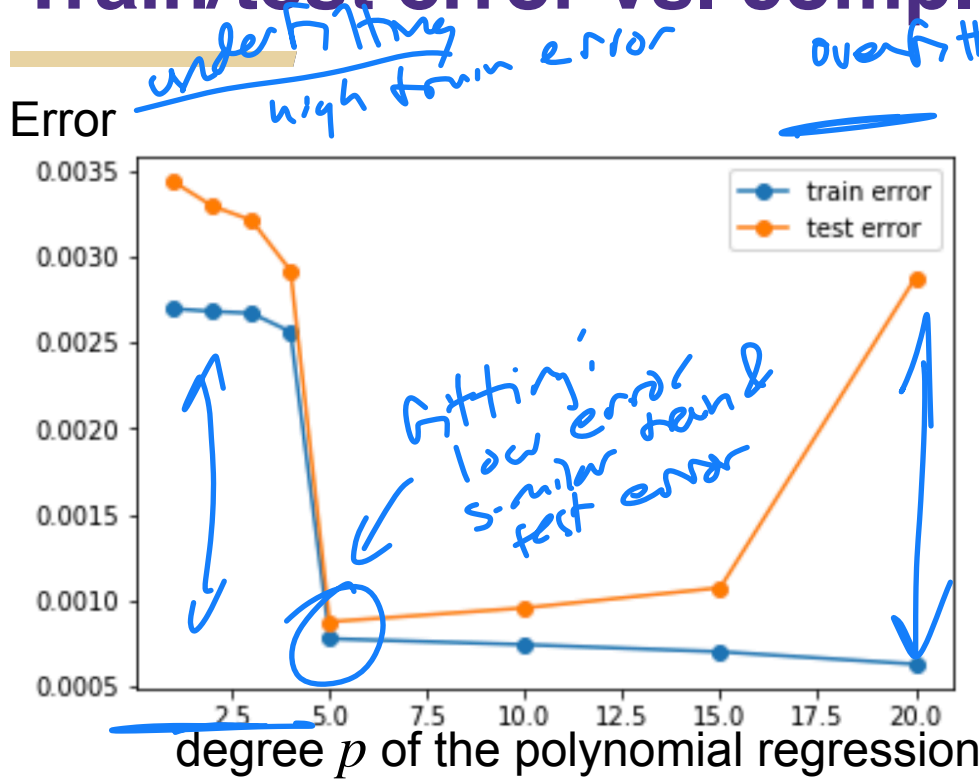
Split the data into **training** and **testing**

- a way to mimic how the predictor performs on unseen data
- given a single dataset $S = \{(x_i, y_i)\}_{i=1}^n$
- we split the dataset into two: training set and test set (e.g., 90/10)
- **training set** used to train the model
 - minimize $\mathcal{L}_{\text{train}}(w) = \frac{1}{|S_{\text{train}}|} \sum_{i \in S_{\text{train}}} (y_i - x_i^T w)^2$
→ At the params of model
- **test set** used to evaluate the model
 - $\mathcal{L}_{\text{test}}(w) = \frac{1}{|S_{\text{test}}|} \sum_{i \in S_{\text{test}}} (y_i - x_i^T w)^2$
- this assumes that test set is similar to unseen data
- **test set should never be used in training or picking unknowns**

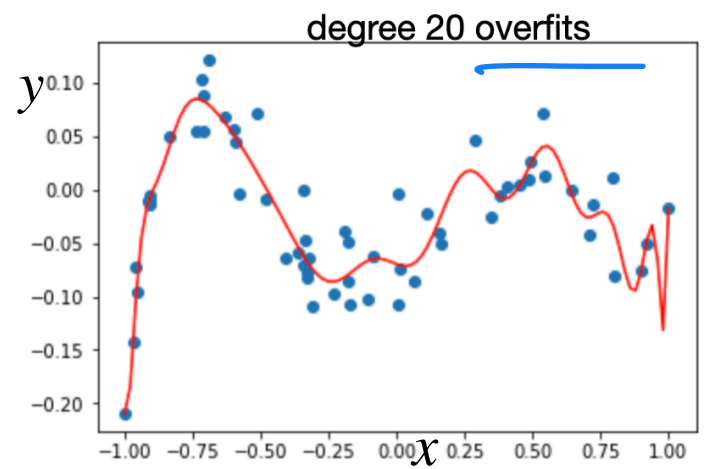
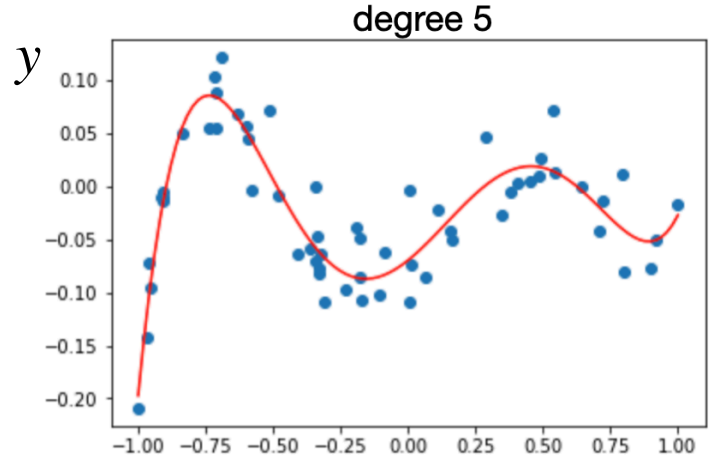
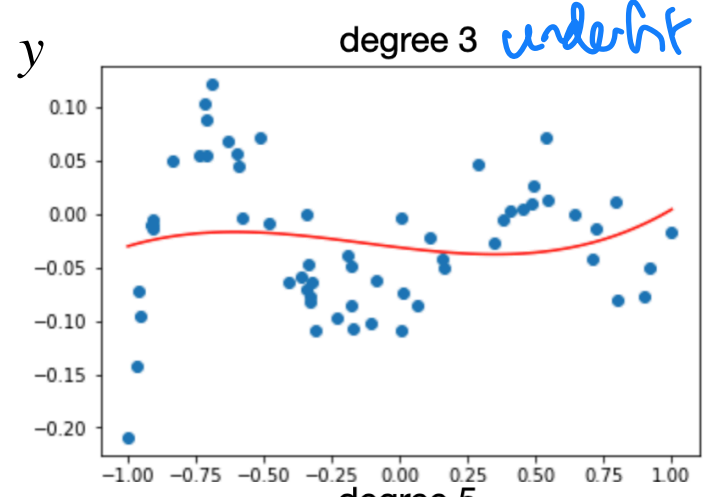
Demo 2: Polynomial Regression

See [colab notebook demo](#)

Train/test error vs. complexity



high gap btw train & test



- Degree $p = 5$, since it achieves **minimum test error**
- **Train error** monotonically decreases with model complexity
- **Test error** has a U shape. There is a best value

test set should never be used in training or picking degree

Cross-Validation

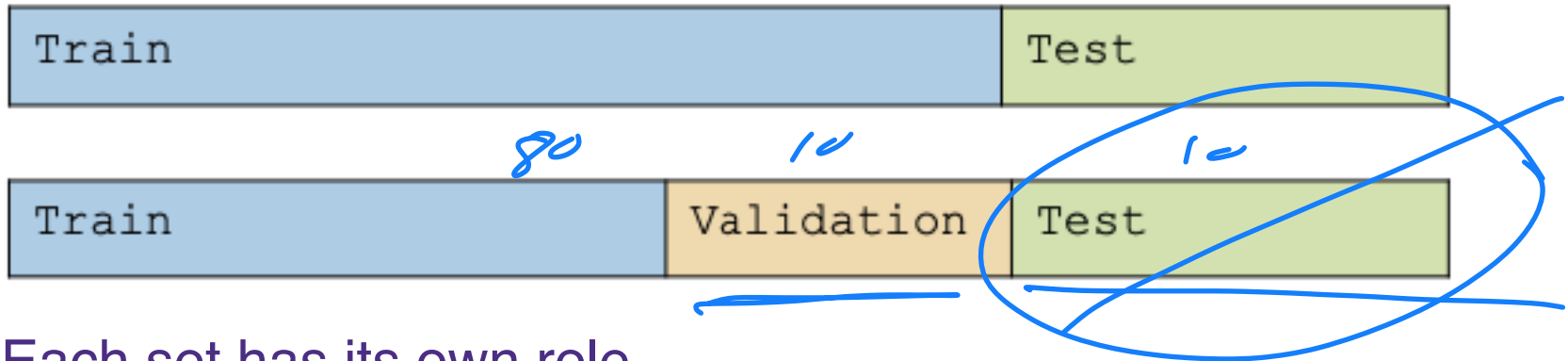
How... How... How????????

- > How do we pick the number of basis functions...
- > We could use the test data, but...

Validation Set

parameters = ω
fit ω / ML
hyperparameter: polynomial degree
chosen by ML researcher

- > Simplest approach is to add another dataset partition called the **validation set**



- > Each set has its own role
 - > **Train:** Used to train a model of each model complexity
 - > **Validation:** Used as a hold-out to evaluate each model. Generally choose model complexity with lowest validation error.
 - > **Test:** Once the model is chosen, can get an estimate of future error by test. This would be what you report. **Only do this once!**

(LOO) Leave-one-out cross validation

- > Consider a validation set with 1 example:
 - D – training data *n samples*
 - $D \setminus j$ – training data with j th data point (x_j, y_j) moved to validation set
- > Learn model $f_{D \setminus j}$ with $D \setminus j$ dataset *n-1* →
- > Estimate true error as squared error on predicting y_j :
 - Unbiased estimate of error_{true}($f_{D \setminus j}$)!

computationally \$\$

(LOO) Leave-one-out cross validation

- > Consider a validation set with 1 example:
 - D – training data
 - $D \setminus j$ – training data with j th data point (x_j, y_j) moved to validation set
- > Learn model $f_{D \setminus j}$ with $D \setminus j$ dataset
- > Estimate true error as squared error on predicting y_j :
 - Unbiased estimate of error_{true}($f_{D \setminus j}$)!
- > LOO cross validation: Average over all data points j :
 - For each data point you leave out, learn a new model $f_{D \setminus j}$
 - Estimate error as:

$$\underline{\text{error}_{LOO}} = \frac{1}{n} \sum_{j=1}^n (y_j - f_{D \setminus j}(x_j))^2$$

LOO cross validation is (almost) unbiased estimate!

- > When computing LOOCV error, we only use $N-1$ data points
 - So it's not estimate of true error of learning with N data points
 - Usually pessimistic, though – learning with less data typically gives worse answer

- > LOO is almost unbiased! Why not use LOO error for model selection!
 - E.g., picking degree

Computational cost of LOO

- > Suppose you have 100,000 data points
- > You implemented a great version of your learning algorithm
 - Learns in only 1 second ←
- > Computing LOO will take about 1 day!!!
 -

Use k -fold cross validation

> Randomly divide training data into k equal parts

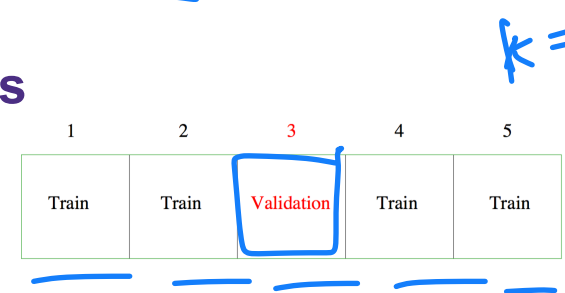
– D_1, \dots, D_k

> For each i

– Learn model $f_{D \setminus D_i}$ using data point not in D_i

– Estimate error of $f_{D \setminus D_i}$ on validation set D_i :

$$\text{error}_{D_i} = \frac{1}{|D_i|} \sum_{(x_j, y_j) \in D_i} (y_j - f_{D \setminus D_i}(x_j))^2$$



Use k -fold cross validation

> Randomly divide training data into k equal parts

- D_1, \dots, D_k

> For each i


- Learn model $f_{D \setminus D_i}$ using data point not in D_i
- Estimate error of $f_{D \setminus D_i}$ on validation set D_i :

$$\text{error}_{D_i} = \frac{1}{|D_i|} \sum_{(x_j, y_j) \in D_i} (y_j - f_{D \setminus D_i}(x_j))^2$$

> k -fold cross validation error is average over data splits:

$$\text{error}_{k\text{-fold}} = \frac{1}{k} \sum_{i=1}^k \text{error}_{D_i}$$

> k -fold cross validation properties:

- Much faster to compute than LOO 
- More (pessimistically) biased – using much less data, only $n(k-1)/k$
- Usually, $k = 10$

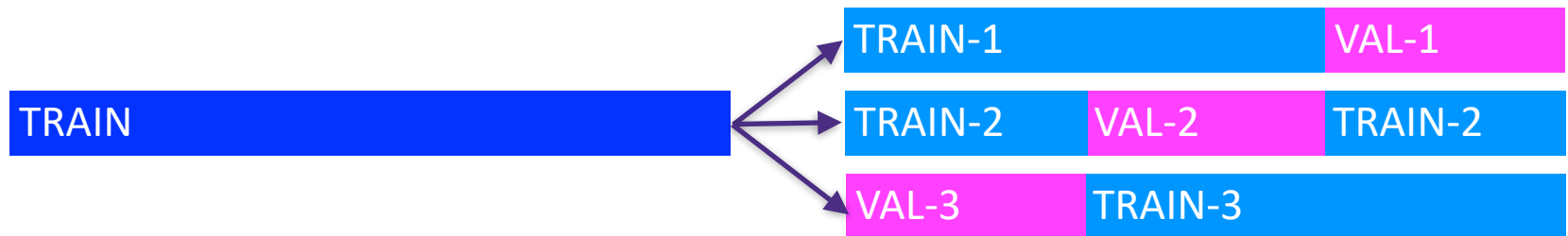
1	2	3	4	5
Train	Train	Validation	Train	Train

Recap

- > Given a dataset, begin by splitting into



- > Model selection: Use k-fold cross-validation on **TRAIN** to train predictor and choose magic parameters such as degree



- > Model assessment: Use **TEST** to assess the accuracy of the model you output

- **Never ever ever ever ever train or choose parameters based on the test data**

Example 1

- > You wish to predict the stock price of zoom.us given historical stock price data
- > You use all daily stock price up to Jan 1, 2020 as **TRAIN** and Jan 2, 2020 - April 13, 2020 as **TEST**
- > What's wrong with this procedure?

IID

OOD
out of
distribution

Example 2

$x_i \in \mathbb{R}^d = 10,000$
FIRST holdout test set

10,000

- > Given 10,000-dimensional data and n examples, we pick a subset of 50 dimensions that have the highest correlation with labels in the entire dataset:

50 indices j that have largest

$$\frac{|\sum_{i=1}^n x_{i,j} y_i|}{\sqrt{\sum_{i=1}^n x_{i,j}^2}}$$

E

- > After picking our 50 features, we then break data into train and test dataset.
- > We train linear regression on these selected features on the training set. We compute the test error and report it
- > What's wrong with this procedure?

90%
contaminating test
60%

Recap

> Learning is...

- Collect some data
 - > E.g., housing info and sale price
- Randomly split dataset into TRAIN, VAL, and TEST
 - > E.g., 80%, 10%, and 10%, respectively
- Choose a hypothesis class or model
 - > E.g., linear with non-linear transformations
- Choose a loss function
 - > E.g., least squares on TRAIN
- Choose an optimization procedure
 - > E.g., set derivative to zero to obtain estimator, cross-validation on VAL to pick num. features
- > Justifying the accuracy of the estimate
 - > E.g., report TEST error

estimate generalization error
- pick hparams