

CSE/ISyE 6740
Computational Data Analysis

Bias-Variance Tradeoff

10/12/2026

Kai Wang, Assistant Professor in Computational Science and Engineering
kwang692@gatech.edu

Outline

- **Supervised Learning**
 - **Bias-variance tradeoff**
 - Bias-variance tradeoff
 - Methods to mitigate overfitting

Bias-Variance Tradeoff

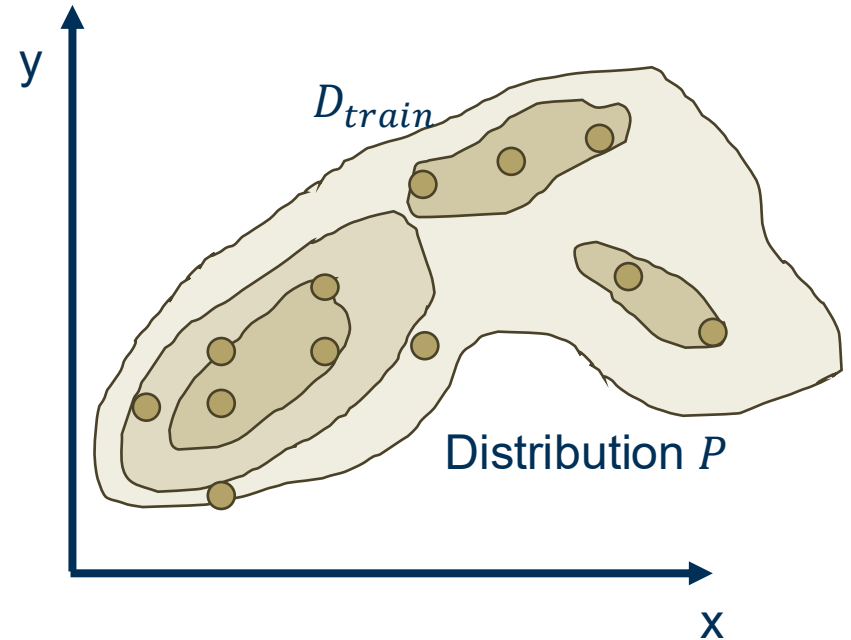
What is Wrong with MSE / Maximum Likelihood?

- Given n data points, find θ that minimizes the mean square error.

$$\hat{\theta} = \operatorname{argmin}_{\theta} \hat{L}(\theta) := \frac{1}{n} \sum_{i=1}^n (y^i - \theta^{\top} x^i)^2$$

- The loss function is measured on the distribution P , not just on training samples

$$L(\theta) := \mathbb{E}_{(x,y) \sim P} \left[(y^i - \theta^{\top} x^i)^2 \right]$$



Empirical Risk Minimization v.s. True Loss

- **Empirical risk minimization:** estimate your function from a finite data set D

$$\hat{f}_D = \operatorname{argmin}_{f \in \mathcal{H}} \hat{L}(f) := \frac{1}{n} \sum_{i=1}^n \left(y^i - f(x^i) \right)^2$$

$\hat{f}_D$ is a function in the hypothesis class $\mathcal{H}$, e.g., all polynomial with degree d .

- **True loss of $\hat{f}_D$:**

$$L(\hat{f}_D) := \underbrace{\mathbb{E}_{(x,y) \sim P}} \left[\left(y - \hat{f}_D(x) \right)^2 \right]$$

Evaluate the performance of $\hat{f}_D$
on the true distribution P

Decomposition of Expected Loss

- **Empirical risk minimization:** estimate your function from a finite data set D

$$\hat{f}_D = \operatorname{argmin}_{f \in \mathcal{H}} \hat{L}(f) := \frac{1}{n} \sum_{i=1}^n \left(y^i - f(x^i) \right)^2$$

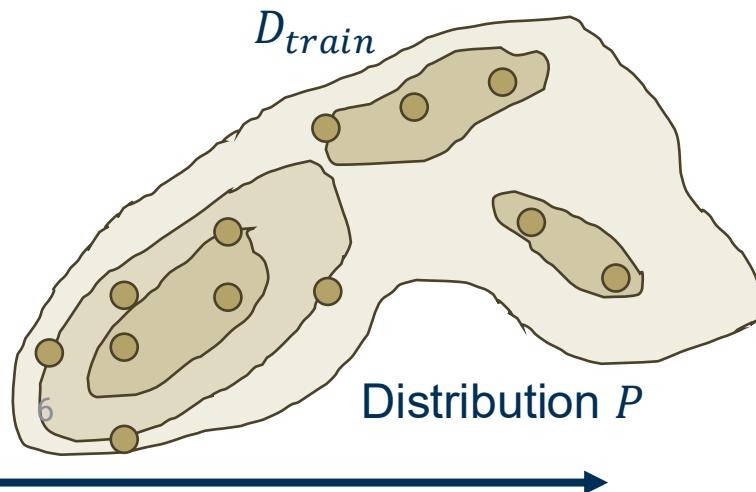
$\hat{f}_D$ is a function in the hypothesis class $\mathcal{H}$, e.g., all polynomial with degree d .

- **True loss of $\hat{f}_D$:**

Taking the randomness of the dataset D into account.

$$L := \mathbb{E}_D [L(\hat{f}_D)] = \mathbb{E}_D \underbrace{\mathbb{E}_{(x,y) \sim P} \left[\left(y - \hat{f}_D(x) \right)^2 \right]}$$

Evaluate the performance of $\hat{f}_D$ on the true distribution P



Decomposition of Expected Loss

- **Empirical risk minimization:** estimate your function from a finite data set D

$$\hat{f}_D = \operatorname{argmin}_{f \in \mathcal{H}} \hat{L}(f) := \frac{1}{n} \sum_{i=1}^n \left(y^i - f(x^i) \right)^2$$

$\hat{f}_D$ is a function in the hypothesis class $\mathcal{H}$, e.g., all polynomial with degree d .

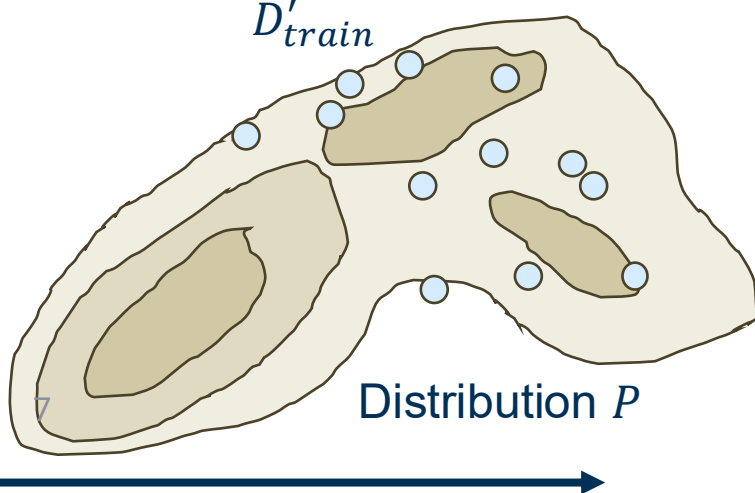
- **True loss of $\hat{f}_D$:**

Taking the randomness of the dataset D into account.

Unlucky training samples D'_{train}

$$L := \mathbb{E}_D [L(\hat{f}_D)] = \mathbb{E}_D \underbrace{\mathbb{E}_{(x,y) \sim P} \left[\left(y - \hat{f}_D(x) \right)^2 \right]}$$

Evaluate the performance of $\hat{f}_D$ on the true distribution P



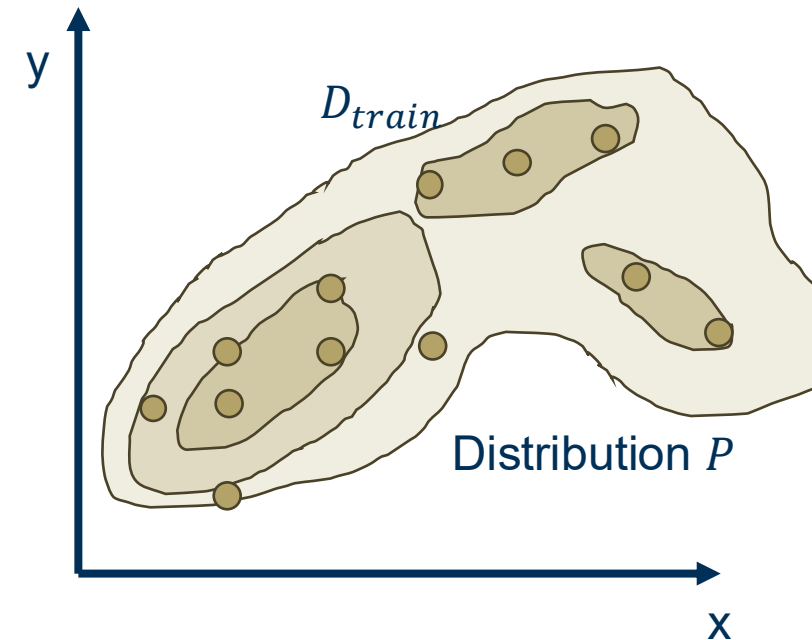
Bias-Variance Decomposition

- **This is the first step to understand (not to mitigate yet) the issue of overfitting!**
Expected loss = (bias)² + variance + noise

$$\begin{aligned} L &:= \mathbb{E}_D \mathbb{E}_{(x,y) \sim P(x,y)} \left[\left(y - \hat{f}_D(x) \right)^2 \right] \\ &= \mathbb{E}_x \left[\left(\mathbb{E}_D [\hat{f}_D(x)] - h(x) \right)^2 \right] \quad (\text{bias}^2) \\ &\quad + \mathbb{E}_x \mathbb{E}_D \left[\left(\hat{f}_D(x) - \mathbb{E}_D [\hat{f}_D(x)] \right)^2 \right] \quad (\text{variance}) \\ &\quad + \mathbb{E}_{(x,y) \sim P(x,y)} \left[(h(x) - y)^2 \right] \quad (\text{noise}) \end{aligned}$$

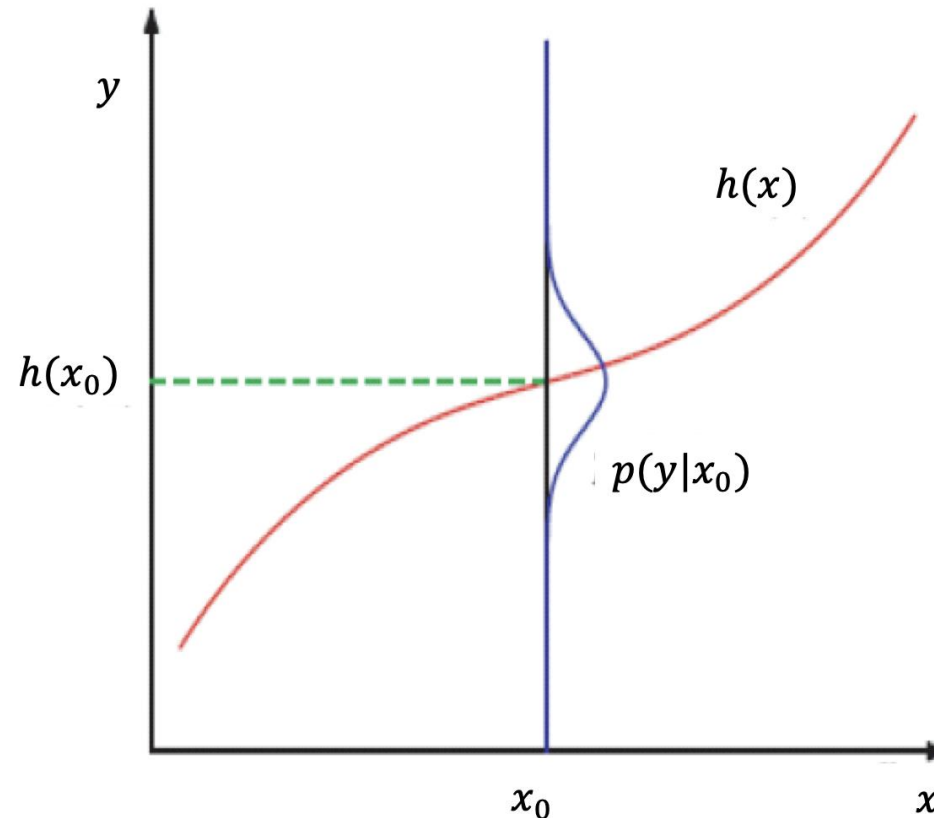
- Key quantities:

- $\hat{f}_D(x)$: predictor from dataset D
- $\mathbb{E}_D [\hat{f}_D(x)]$: expected predictor
- $h(x) = \mathbb{E}[y|x]$: optimal predictor



The Best Predictor is the Expected Value

- Let's denote $h(x) = \mathbb{E}[y|x]$, i.e., the expected value of y given a particular x . Setting $\hat{f} = h$ is the best we can do for minimizing the loss.



Meaning of Bias-Variance Tradeoff

$$\text{Bias}^2: \mathbb{E}_x \left[\left(\mathbb{E}_D[\hat{f}(x)] - h(x) \right)^2 \right]$$

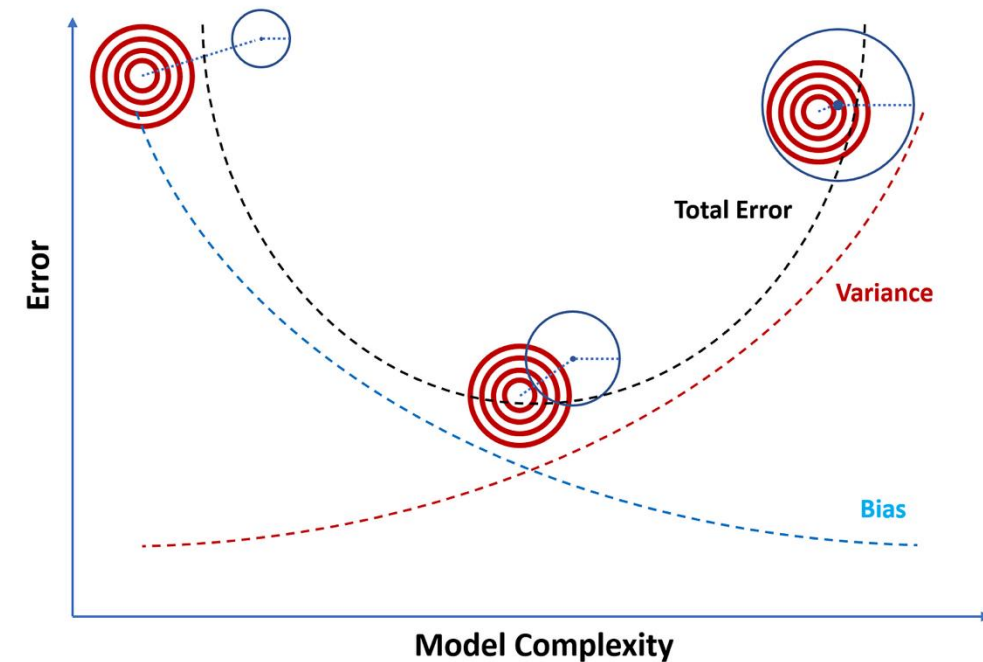
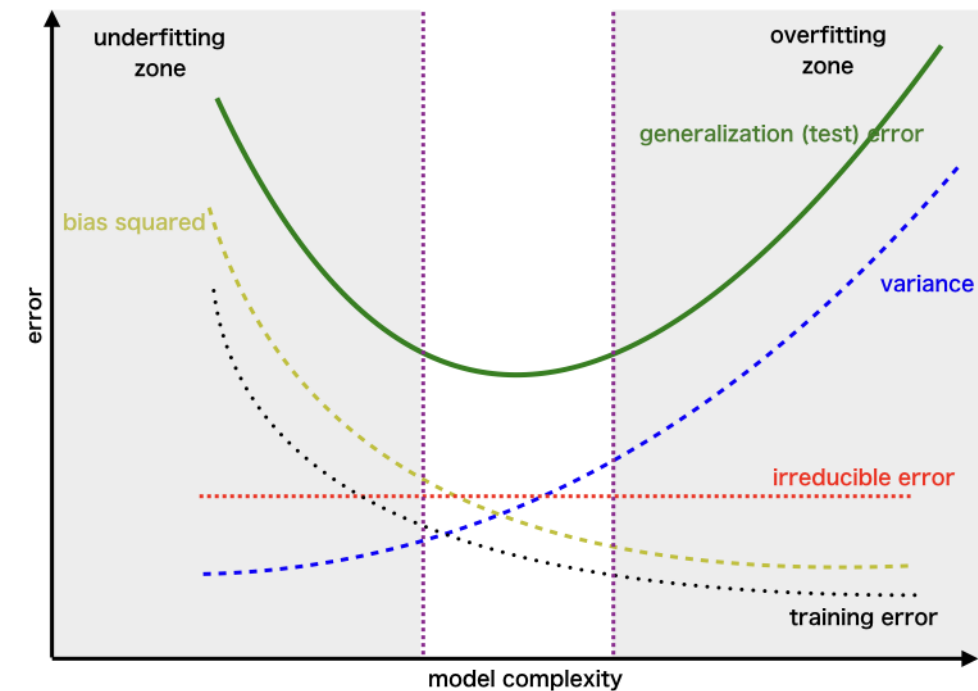
- Bias measures how far off the predictions are from the actual values on average.

$$\text{Variance: } \mathbb{E}_x \mathbb{E}_D \left[\left(\hat{f}(x) - \mathbb{E}_D[\hat{f}(x)] \right)^2 \right]$$

- Variance measures how much the predictions vary for a fixed x across different realizations of the model.

$$\text{Noise: } \mathbb{E}_{(x,y) \sim P(x,y)} [(h(x) - y)^2]$$

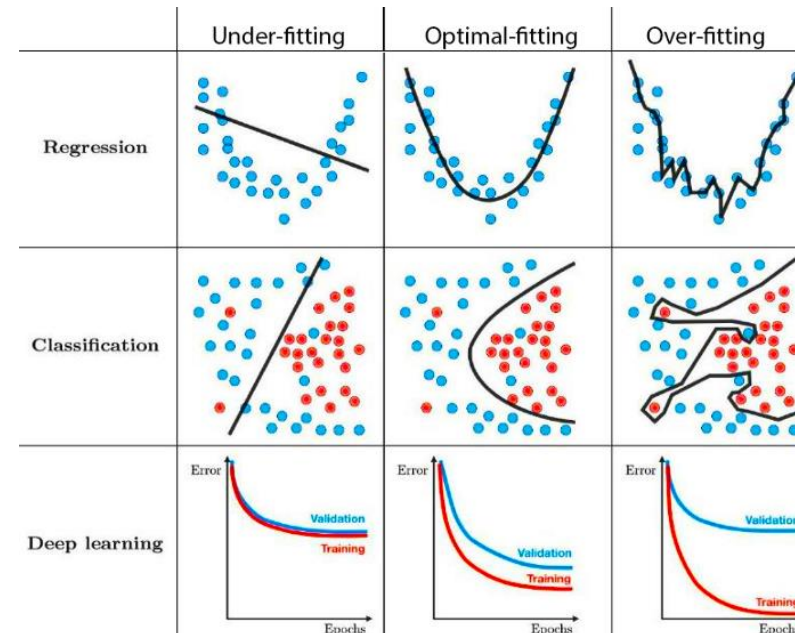
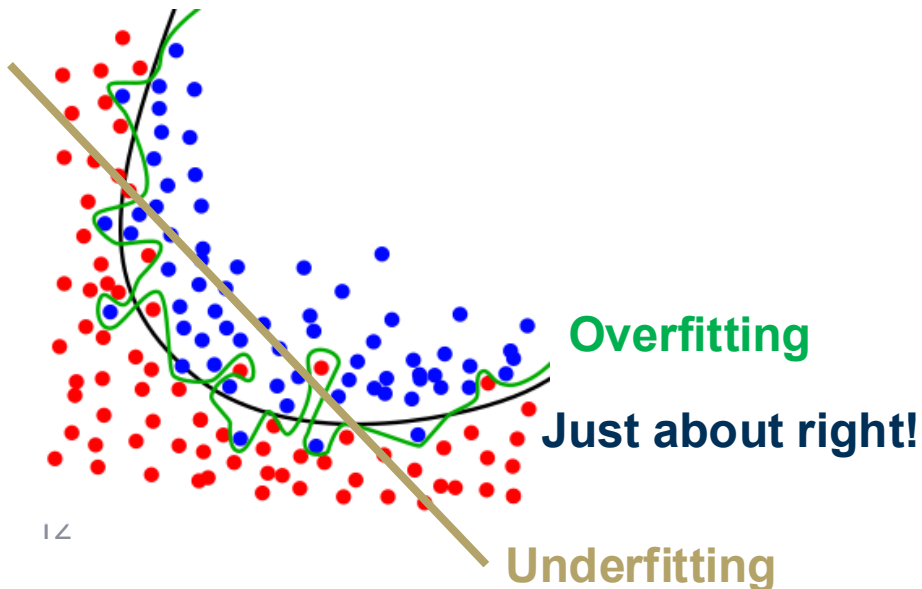
- This is the error inherent in the problem, often due to randomness or natural variability in the system.



Methods to Mitigate Overfitting

Bias-Variance Tradeoff is a Method to Understand Overfitting

	Bias	Variance	Noise	Model complexity	Generalizability
Underfitting: you have an overly simplified model	High	Low	Nothing we can do	Low	High
Overfitting: your model is too complicated and can be modeling the noise	Low	High	Nothing we can do	High	Low



1. How to prevent?
2. How to detect?

Many Methods in Training Neural Networks

- **Regularization**
- **Cross validation**
- **Monitoring learning curve (early stopping)**

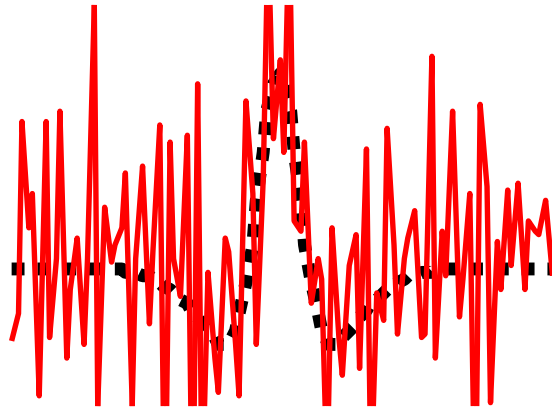
Important Principles

- Know your **model complexity** and **number of data samples**
- Do **NOT** blindly increase model complexity
- Do **NOT** use any testing samples in the training process
- Monitor your **learning curve** (training and validation)

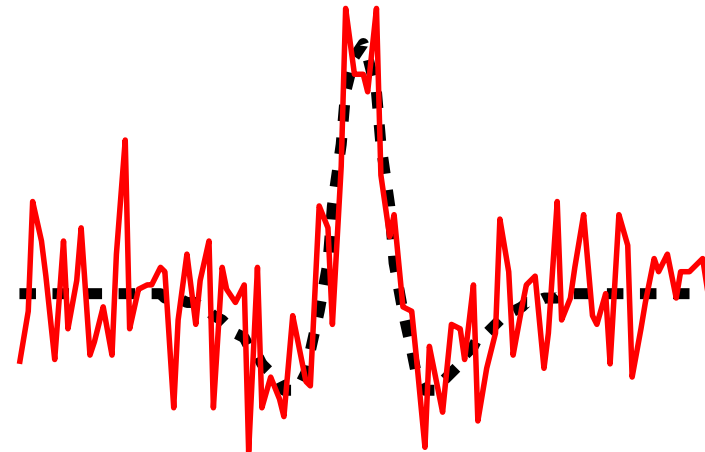
Solution 1: Regularization

- Add a regularization term $\|\theta\|^2$ (L2) or $|\theta|_1$ (L1) to penalize larger parameter θ

$$\hat{\theta} = (XX^T)^{-1}Xy$$



$$\hat{\theta} = (XX^T + \lambda I)^{-1}Xy$$

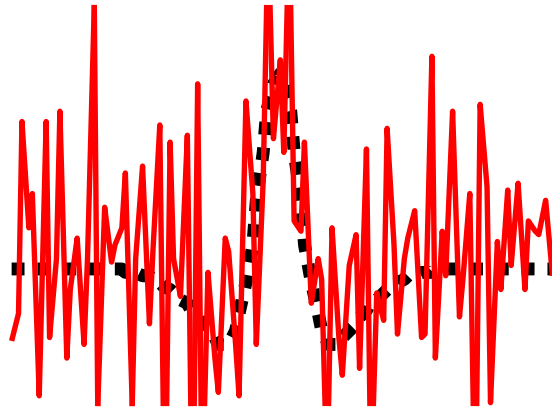


Solution 1: Regularization

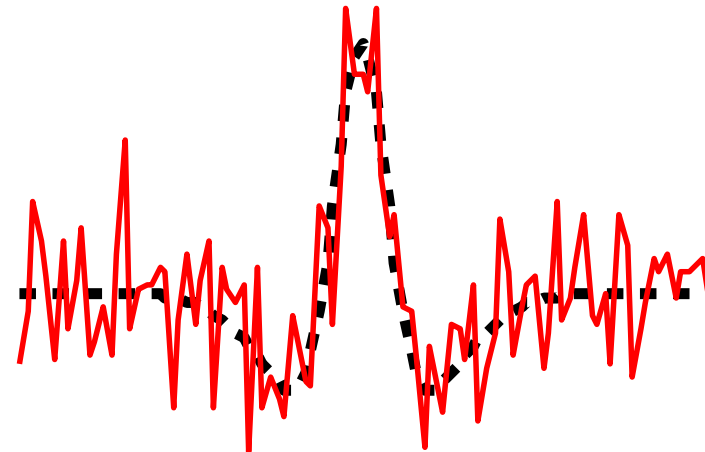
- What does regularization do in terms of bias, variance, and noise?

	Bias	Variance	Noise
Model with regularization			

$$\hat{\theta} = (XX^T)^{-1}Xy$$



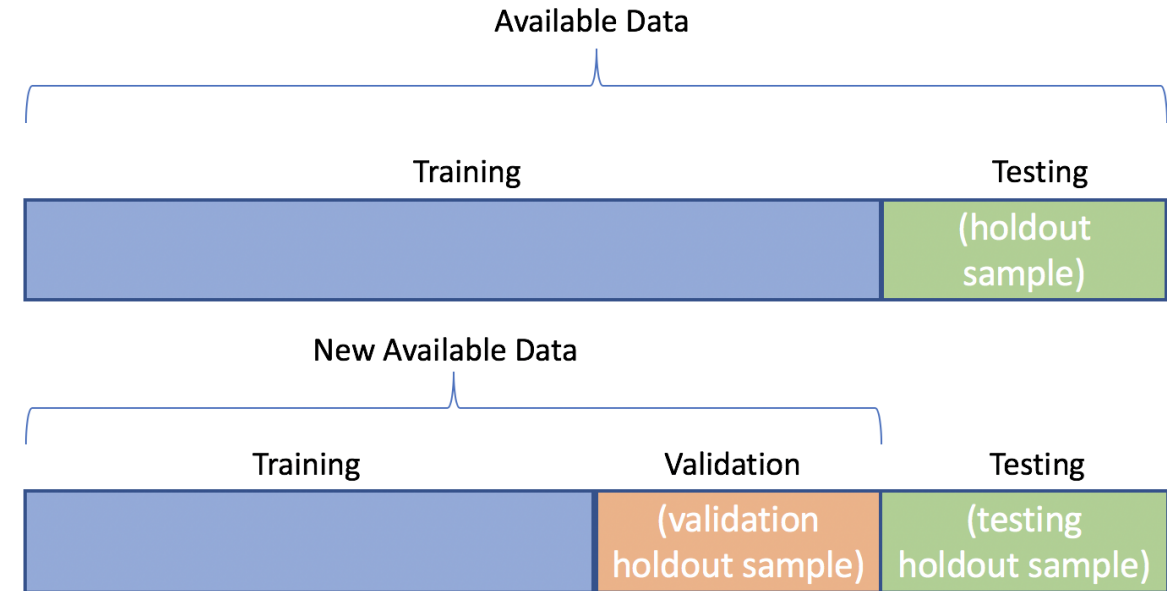
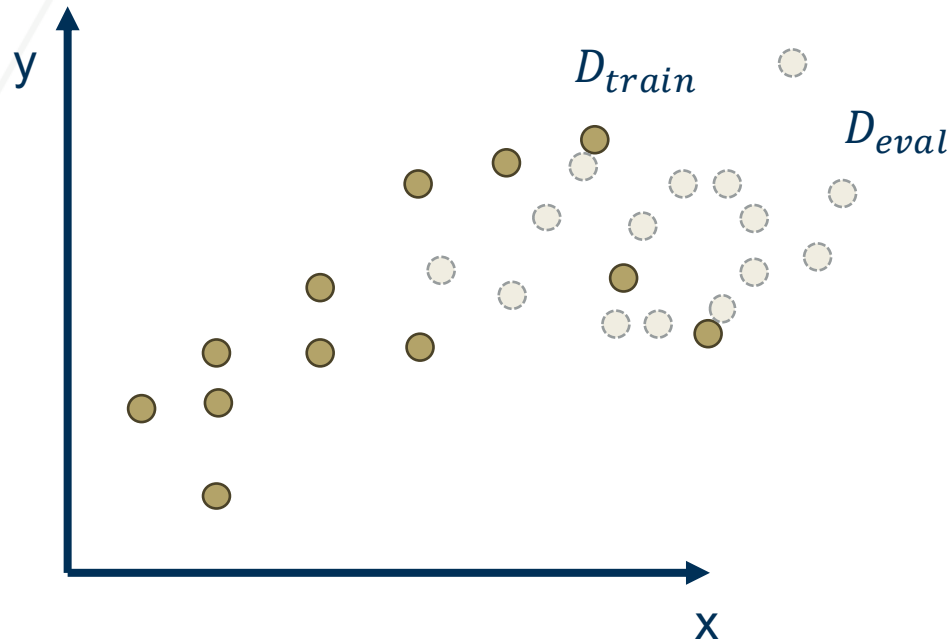
$$\hat{\theta} = (XX^T + \lambda I)^{-1}Xy$$



Solution 2: Cross-Validation

- **Train-validation split**

- Split the dataset D into a training set D_{train} and a validation set D_{val}
- Only use the training set to train.



```
torch.utils.data.random_split(dataset, lengths, generator=<torch._C.Generator object>) \[SOURCE\]
```

Randomly split a dataset into non-overlapping new datasets of given lengths.

If a list of fractions that sum up to 1 is given, the lengths will be computed automatically as $\text{floor}(\text{frac} * \text{len}(\text{dataset}))$ for each fraction provided.

After computing the lengths, if there are any remainders, 1 count will be distributed in round-robin fashion to the lengths until there are no remainders left.

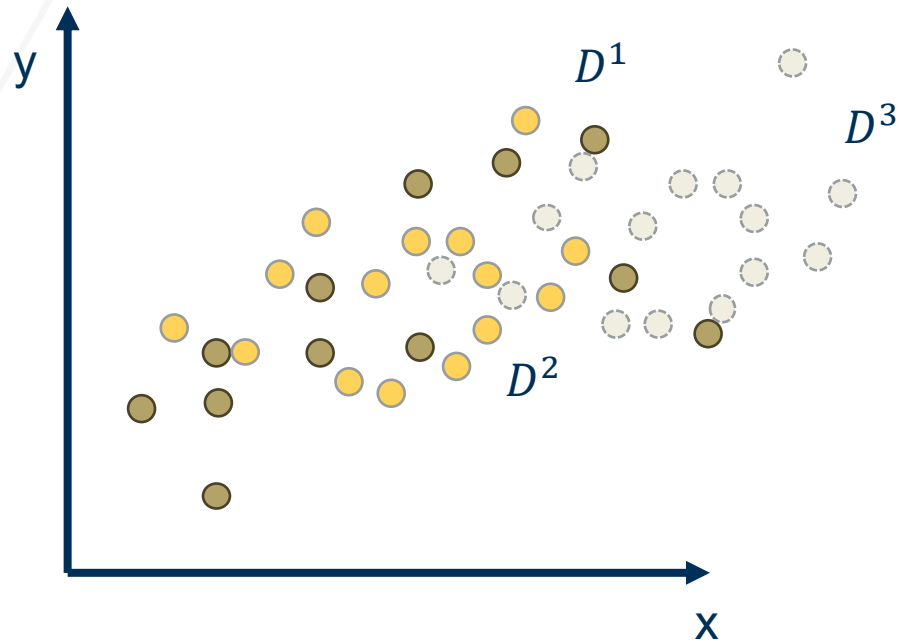
Optionally fix the generator for reproducible results, e.g.:

Example

```
>>> generator1 = torch.Generator().manual_seed(42)
>>> generator2 = torch.Generator().manual_seed(42)
>>> random_split(range(10), [3, 7], generator=generator1)
>>> random_split(range(30), [0.3, 0.3, 0.4], generator=generator2)
```

K-fold Cross Validation

- Split the dataset into K small chunks. (commonly use $K = 10$)
- Have $K - 1$ of them as training set, and the remaining one as validation set.



Pick the best one!!

Iteration 1	Test	Train	Train	Train	Train	Loss 0.8
Iteration 2	Train	Test	Train	Train	Train	Loss 0.6
Iteration 3	Train	Train	Test	Train	Train	Loss 1.1
Iteration 4	Train	Train	Train	Test	Train	Loss 0.9
Iteration 5	Train	Train	Train	Train	Test	Loss 1.2

Solution 3: Monitoring Learning Curve (Early Stopping)

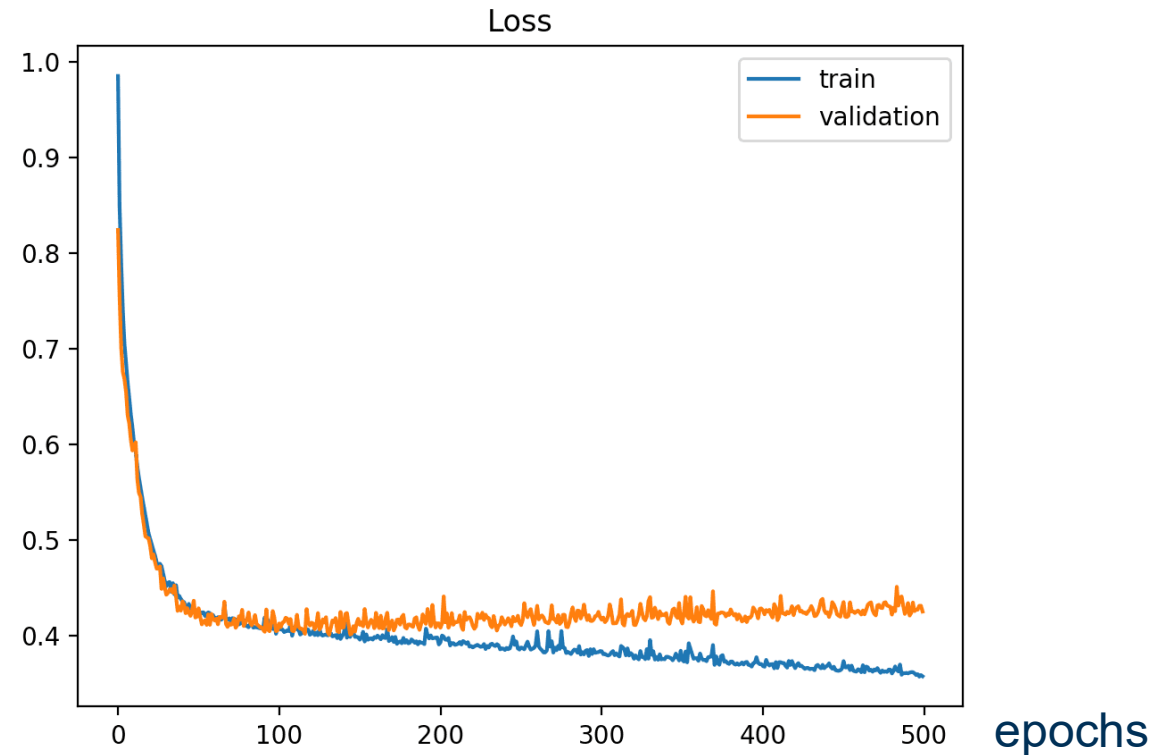
- You should also monitor the predictive error (accuracy) on the training and validation set over training epochs.

For epoch in 1,2,...

For datapoint (or batch) i in D_{train}

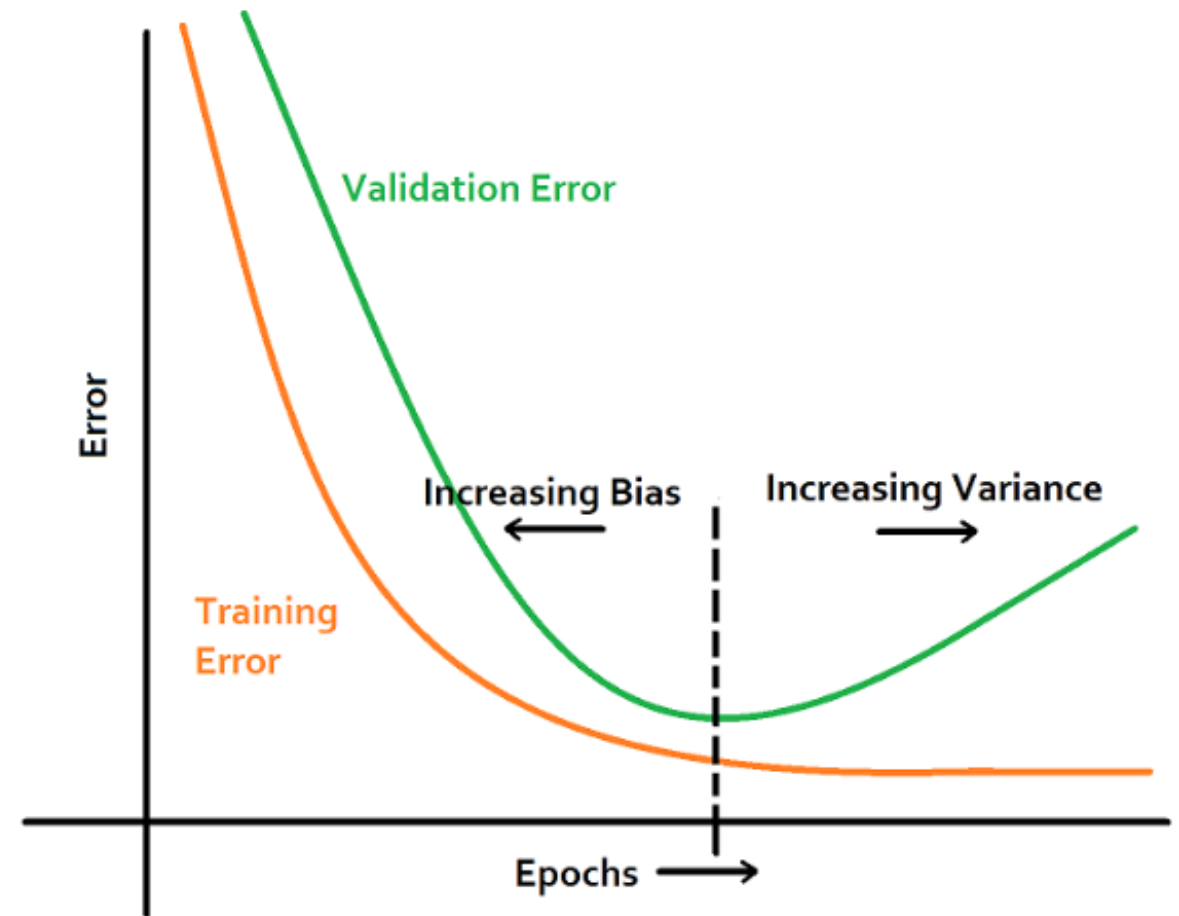
- Compute loss and gradient
- Update model parameter
- Validate on D_{val}

- Record the average predictive loss or accuracy for the current epoch



Solution 3: Monitoring Learning Curve (Early Stopping)

- **Early stopping:**
 - Stop when you see a consistent increase in the validation error (or decline in the accuracy)
 - This is also why we want to use validation set to **identify overfitting**



PyTorch Implementation

- Setup a dynamic data loader (should we load all the data into memory?)
- Do train/validation split
- Implement the training loop, **model**, and **the loss function**
- Run gradient descent
- Track the learning curve to determine when to stop

```
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

[PyTorch random_split function](#)

```
>>> generator1 = torch.Generator().manual_seed(42)
>>> generator2 = torch.Generator().manual_seed(42)
>>> random_split(range(10), [3, 7], generator=generator1)
>>> random_split(range(30), [0.3, 0.3, 0.4], generator=generator2)
```

```
def train_one_epoch(epoch_index, tb_writer):
    running_loss = 0.
    last_loss = 0.

    # Here, we use enumerate(training_loader) instead of
    # iter(training_loader) so that we can track the batch
    # index and do some intra-epoch reporting
    for i, data in enumerate(training_loader):
        # Every data instance is an input + label pair
        inputs, labels = data

        # Zero your gradients for every batch!
        optimizer.zero_grad()

        # Make predictions for this batch
        outputs = model(inputs)

        # Compute the loss and its gradients
        loss = loss_fn(outputs, labels)
        loss.backward()

        # Adjust learning weights
        optimizer.step()

        # Gather data and report
        running_loss += loss.item()
        if i % 1000 == 999:
            last_loss = running_loss / 1000 # loss per batch
            print(' batch {} loss: {}'.format(i + 1, last_loss))
            tb_x = epoch_index * len(training_loader) + i + 1
            tb_writer.add_scalar('Loss/train', last_loss, tb_x)
            running_loss = 0.

    return last_loss
```

[Fashion MNIST example](#)

Questions

- Why can regularization prevent overfitting?
- Running more epochs of gradient descents, which term among bias, variance, and noise gets reduced?