Machine Learning Basics

Lecture slides for Chapter 5 of *Deep Learning*
www.deeplearningbook.org
Ian Goodfellow

Adapted by m.n. for CMPS 392

Machine Learning

- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E (Tom M. Michell, 1997)
- Example
  - E: the experience of playing thousands of games
  - T: playing checkers game
  - *P:* the fraction of games it wins against human opponents

The task, *T*

- Tasks are usually described in terms of how the machine learning should process an example: $x \in \mathbb{R}^n$ where each entry x_i is a feature
  - **Classification:** Learn $f: \mathbb{R}^n \rightarrow \{1, .., \mathrm{k}\}$
    - $\mathrm{y} = f(x)$: assigns the input to the category with numerical code y
    - Example: object recognition
  - **Classification with missing inputs:** learn a set of functions
    - Example: medical diagnosis

The task, *T*

- **Regression:** Learn $f: \mathbb{R}^n \rightarrow \mathbb{R}$
  - Example: Predict claim amount for an insured person
- **Transcription:** unstructured representation to discrete textual form
  - Example: optical character recognition, speech recognition
- **Machine Translation:** Sequence to sequence
  - Example: translate English to French

The task, *T*

- **Structured output:** output is a vector or data structure of values that are tightly interrelated
  - Example: parsing natural language sentence into a tree that describes grammatical structure by tagging nodes of the tree as being verbs, nouns, etc.
  - image segmentation: assigning a pixel to a segment
  - Image captioning

The task, *T*

- **Anomaly detection:** flag unusual or atypical events
  - Credit card fraud detection
- **Synthesis and sampling:** generate examples that are similar to those in the training data
  - Genreate textures for video games
  - Speech synthesis
- **Imputation of missing values:** predict the values of the missing entries

The task, *T*

- **Denoising:** f: $\tilde{x} \in \mathbb{R}^n \rightarrow x \in \mathbb{R}^n$
  - Predict clean example from corrupted example
  - $p(x|\tilde{x}) =?$
- **Density estimation** or **probability mass estimation**
  - $p_{model}(\boldsymbol{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}$ (x can be discrete or continuous)
  - Example: $p_{model}(x_i|\boldsymbol{x}_{-i})$

The performance measure, *P*

- **Accuracy**:
  - The proportion of examples for which the model produces the correct output
- **Error rate** (0-1 loss):
  - The proportion of examples for which the model produces incorrect output
- **Average log-probability** of some examples (for density estimation)
- It is difficult sometimes, to decide what should be measured
- It is imparactical sometimes to measure an implicit performance metric
- Evaluate the performance measure using a **test set**

The experience, *E*

- **Supervised learning:** $p(y|\boldsymbol{x})$
  - Experience is a labeled dataset (or datapoints)
  - Each datapoint has a label or target
- **Unsupervised learning:** $p(\boldsymbol{x})$
  - Experience is an unlabeled dataset
  - Clustering, learning probability distribution, denoising, etc.
- The line between supervised and unsupervised is often blurred
  - $p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i|x_1, \ldots, x_{i-1})$
  - $p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x},y)}{\sum_{y'} p(\boldsymbol{x},y')}$

The experience, *E*

- **Semi-Supervised learning:**
  - Some examples include supervision targets, but others do not.
- **Multi-instance learning:**
  - A collection of examples is labeled as containing an example of a class
- **Reinforcement learning:**
  - Interaction with an environment

A dataset

- Design matrix **X**:
  - Each row is an example
  - Each column is a feature
  - Example: iris dataset: $X \in \mathbb{R}^{150 \times 4}$
- Vector of labels **y**
  - Example: 0 is a person, 1 is a car, 2 is a cat, etc.
  - The label can be a set of words (e.g. transcription)

Example: Linear regression

- **Task:** regression problem $\mathbb{R}^n \rightarrow \mathbb{R}$
  - $\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$
- **Performance measure:**
  - Have a test set: $\boldsymbol{X}^{\boldsymbol{test}}, \boldsymbol{y}^{\boldsymbol{test}}$

$$MSE_{test} = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}}^{\boldsymbol{test}} - \boldsymbol{y}^{\boldsymbol{test}})_i^2$$

$$= \frac{1}{m} \|\hat{\boldsymbol{y}}^{\boldsymbol{test}} - \boldsymbol{y}^{\boldsymbol{test}}\|_2^2$$

- **Experience**
  - $\boldsymbol{X}^{\boldsymbol{train}}, \boldsymbol{y}^{\boldsymbol{train}}$
  - Minimize MSE_{train}

Linear regression

- $\nabla_{\boldsymbol{w}} MSE_{train} = 0$
- $\frac{1}{m_{train}} \nabla_{\boldsymbol{w}} \left(\boldsymbol{X^{train}} \boldsymbol{w} - \boldsymbol{y^{train}} \right)^T \left(\boldsymbol{X^{train}} \boldsymbol{w} - \boldsymbol{y^{train}} \right) = 0$
- $(\boldsymbol{X^{train^T}} \boldsymbol{X^{train}}) \boldsymbol{w} = \boldsymbol{X^{train^T}} \boldsymbol{y^{train}}$ *(normal equations)*

- We can solve for $\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$ using a simple trick
- *b* is called the bias term *(to not confound with statistical bias that will be discussed later)*

Linear Regression

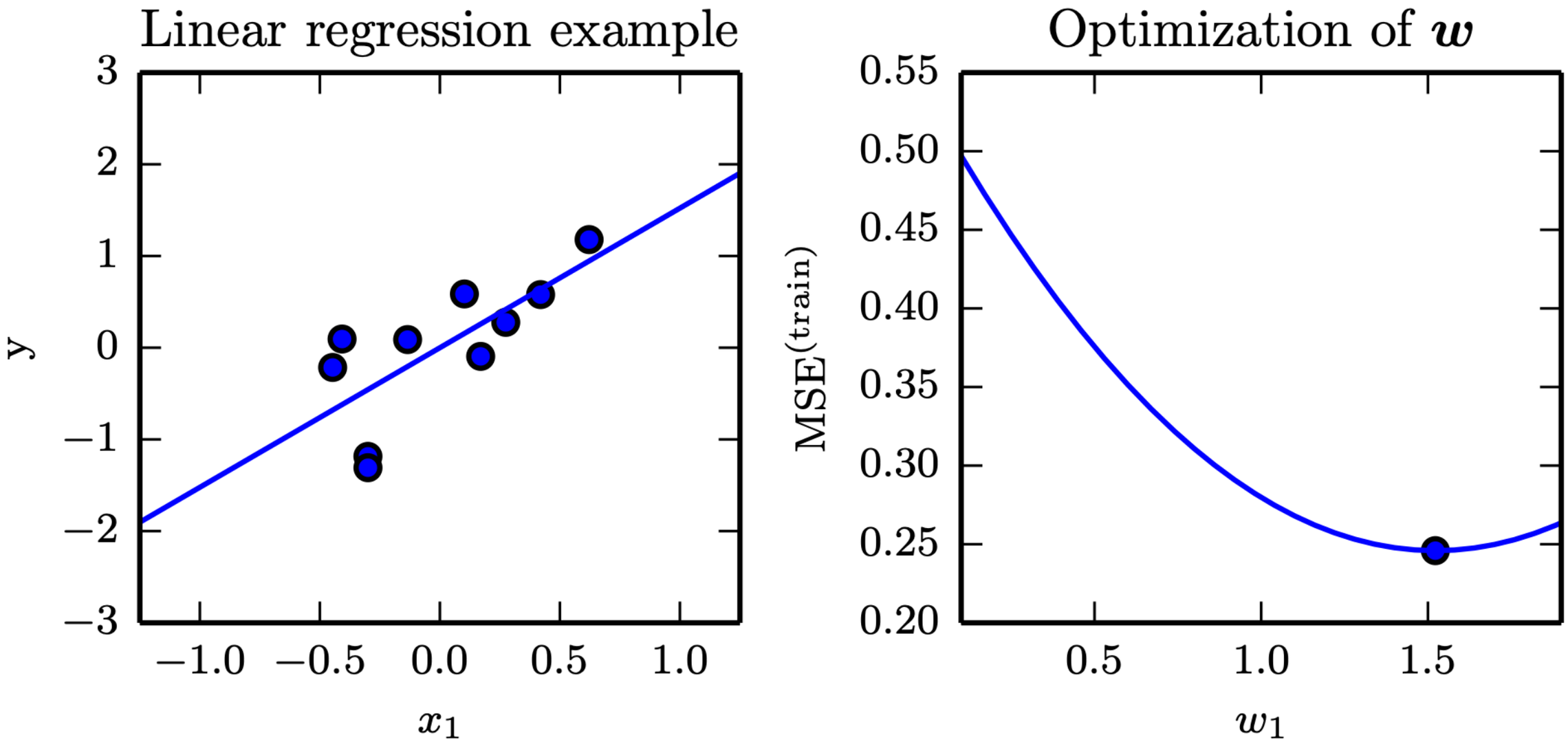


Figure 5.1

Capacity, overfitting and underfitting

- **Generalization:** ability to perform well on previously unobserved data
- **i.i.d assumptions:**
  - The examples in each dataset are independent from each other,
  - and that the train set and test set are identically distributed, drawn from the same probability distribution as each other.
  - We call that shared underlying distribution the data generating distribution, denoted p_{data}

Capacity, overfitting and underfitting

- For some fixed value $\boldsymbol{w}$, the expected training set error is exactly the same as the expected test set error, because both expectations are formed using the same dataset sampling process.
- In practice:
  - expected test set error > expected train set error
- We need ability to:
  - 1. Make the training error small. **(underfitting)**
  - 2. Make the gap between training and test error small. **(overfitting)**

Capacity, overfitting and underfitting

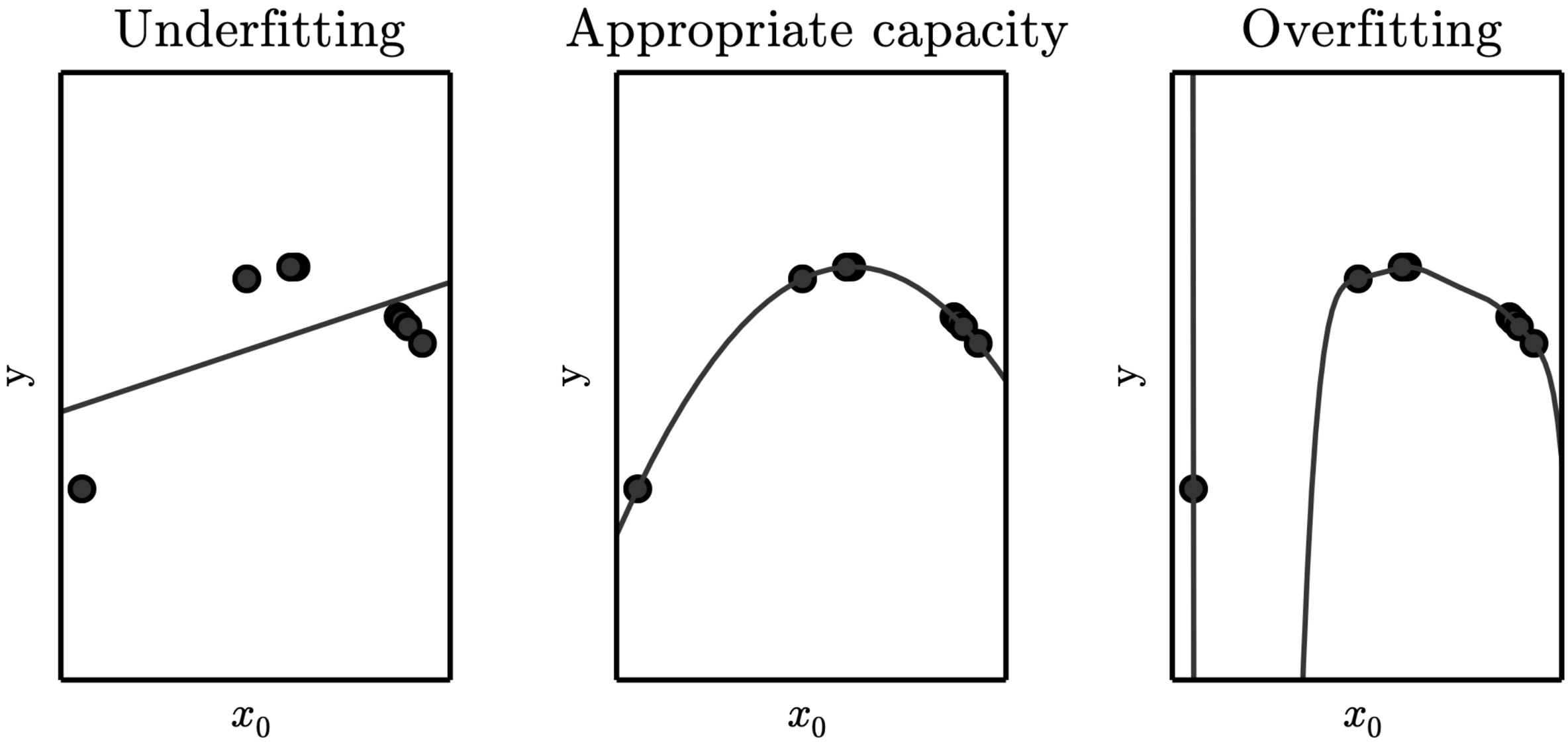
- **Underfitting** occurs when the model is not able to obtain a sufficiently low error value on the training set.
- **Overfitting** occurs when the gap between the training error and test error is too large.
- Informally, a model's **capacity** is its ability to fit a wide variety of functions.
  - Low capacity may cause underfitting
  - High capacity may cause overfitting

Hypothesis space

- One way to control the capacity of a learning algorithm is by choosing its **hypothesis space**,
  - the set of functions that the learning algorithm is allowed to select as being the solution.
  - For example, linear regression has the set of all linear functions of its input
  - To increase capacity: (model is polynomial of degree 9)

$$\hat{y} = b + \sum_{i=1}^{9} w_i x^i$$

Underfitting and Overfitting in Polynomial Estimation



Degree 1

Degree 2

Degree 9: infinitely many solutions

Capacity

- **Representational capacity:** finding the best function within a family of functions
- **Effective capacity** may be less than the representational capacity because it is hard to find the best function
- **Occam's razor:** Among competing hypotheses that explain known observations equally well, one should choose the "**simplest**" one.
- **Statistical learning theory** shows that the discrepancy between training error and generalization error is bounded from above by a quantity that grows as the model capacity grows but shrinks as the number of training examples increases

Generalization and Capacity

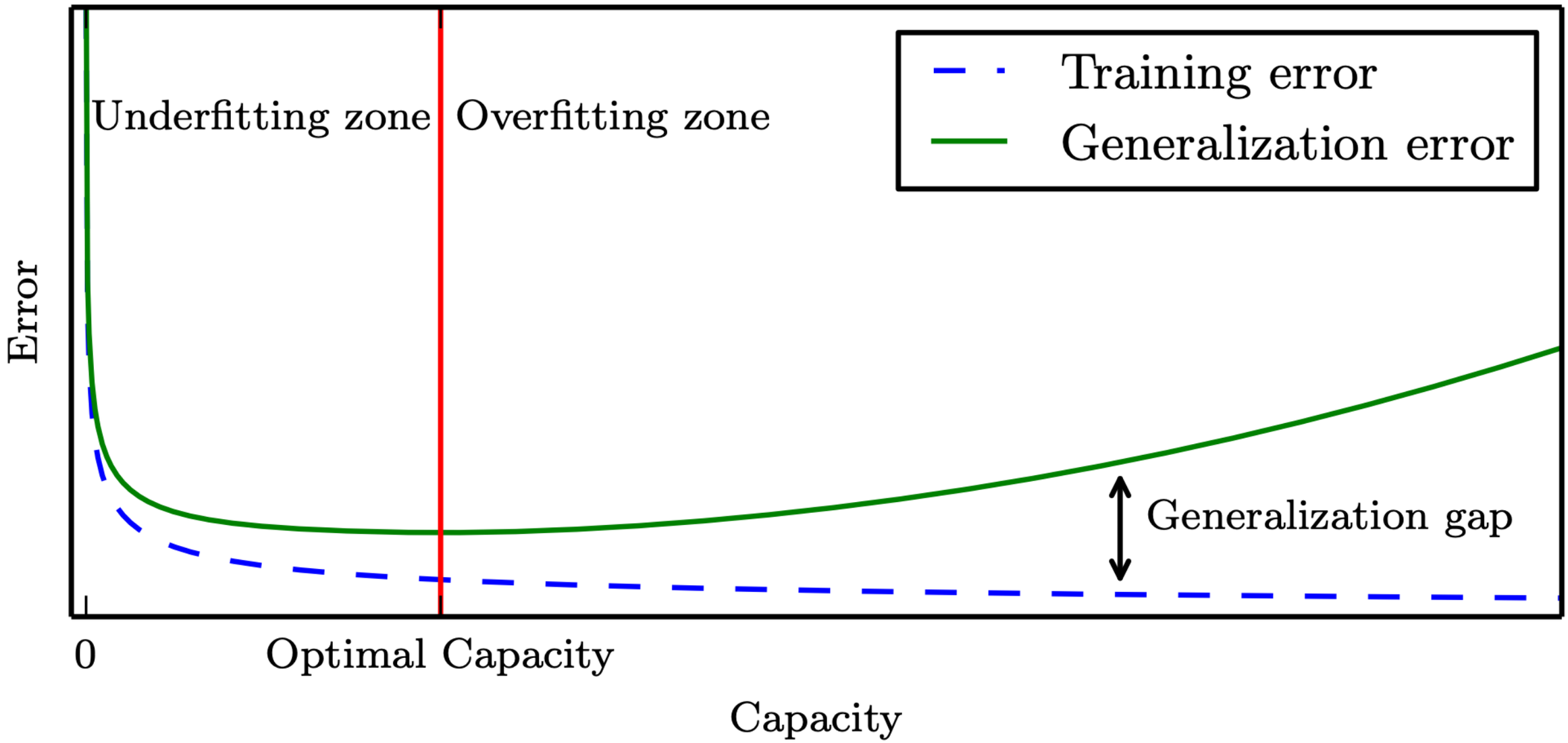


Figure 5.3

High capacity: non-parametric models

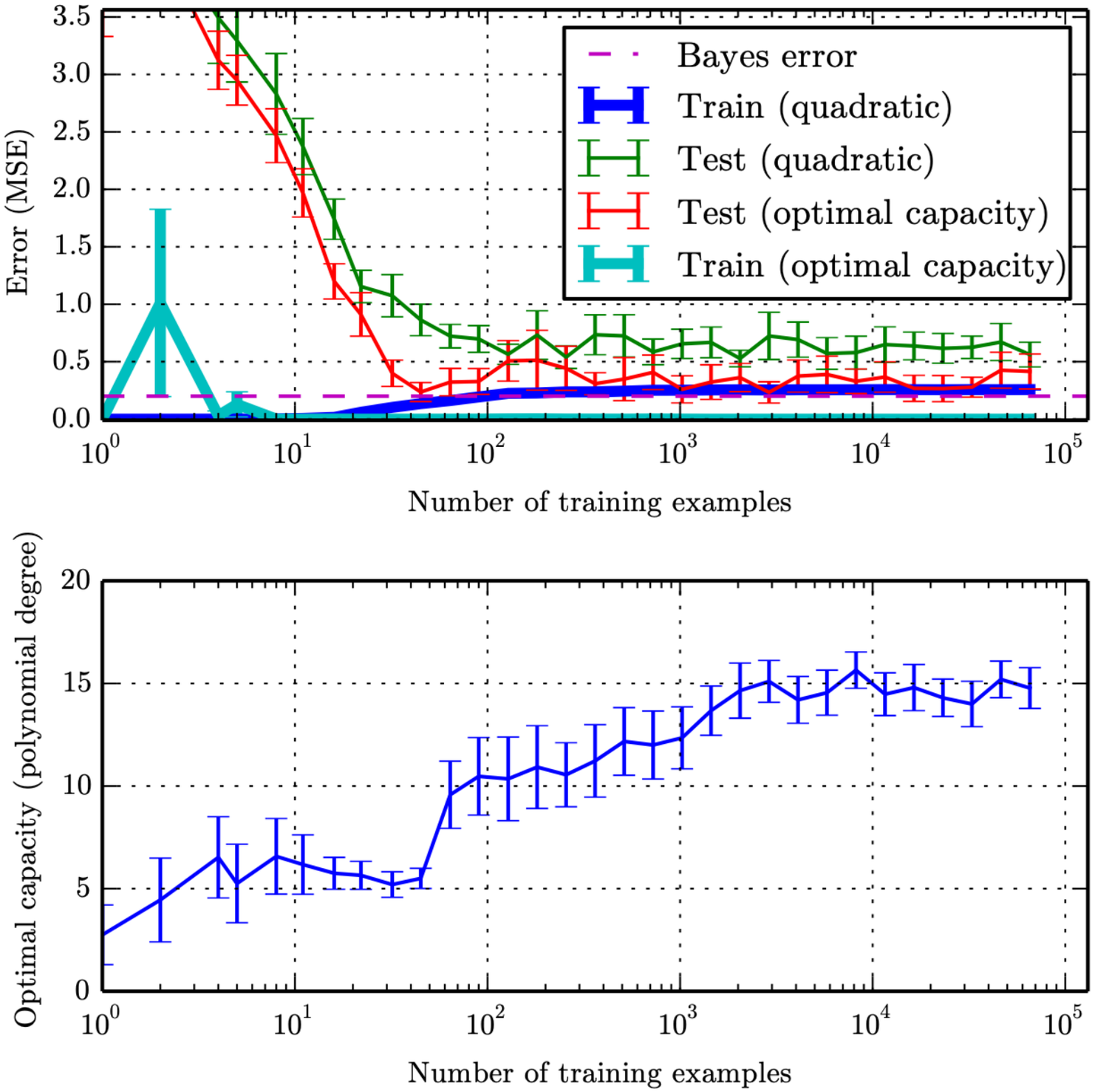
- Nearest neighbor regression

$$\hat{y} = y_i \text{ where } i = \text{argmin} \left\| \boldsymbol{X}_{\boldsymbol{i},:} - \boldsymbol{x} \right\|_2^2$$

Bayes error

- The error incurred by an oracle making predictions from the true distribution $p(\boldsymbol{x}, y)$.
- This error is due to:
  - There may still be noise in the distribution
  - y might be inherently stochastic
  - y may be deterministic but involves other variables besides $\boldsymbol{x}$

Training Set Size



moderate amount of noise added to a degree-5 polynomial

The no Free lunch theorem

- Averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.

- In some sense, no machine learning algorithm is universally any better than any other.

- The most sophisticated algorithm we can conceive of has the same average performance (over all possible tasks) as merely predicting that every point belongs to the same class.

No free lunch

- The goal of machine learning research is not to seek a universal learning algorithm or the absolute best learning algorithm.
- Instead, our goal is to understand what kinds of distributions are relevant to the “real world”
  - and what kinds of machine learning algorithms perform well on data drawn distributions we care about.

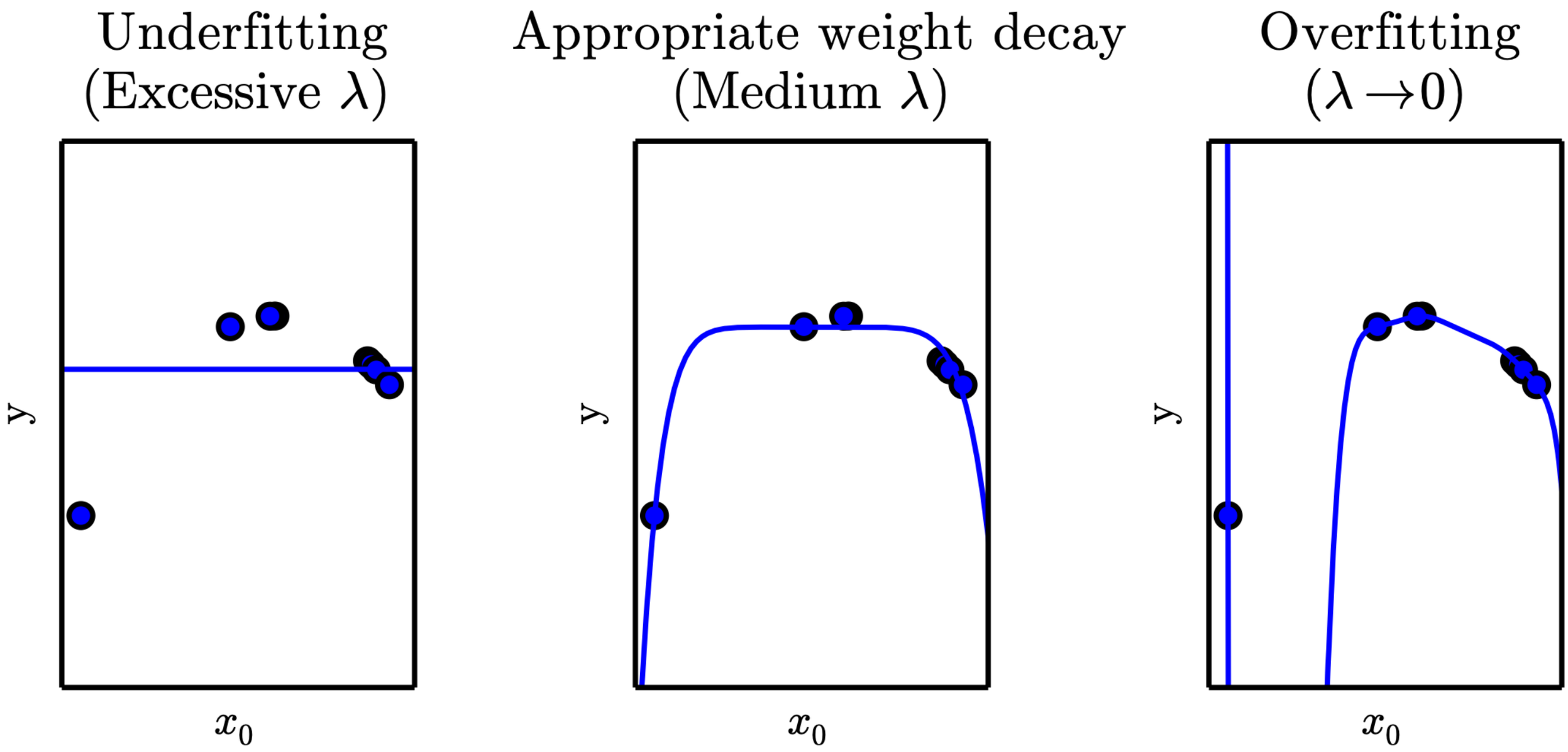
Regularization

- We can give a learning algorithm a preference for one solution in its hypothesis space to another.
  - both functions are eligible, but one is preferred.
- Example: linear regression with weight decay:

$$J(w) = MSE_{train} + \lambda \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{w}$$

- We are expressing a preference for the weights to have smaller L^2 norm
  - Larger λ forces the weights to become smaller

Weight Decay



The true function is quadratic,
but here we use only models with degree 9

Regularization

- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
- Regularization is one of the central concerns of the field of machine learning

Hyperparameters

- The values of hyperparameters are not adapted by the learning algorithm itself
- Polynomial regression hyperparameters:
  - Degree of the polynomial
  - λ (control of the weight decay)
- To set the hyperparameters, we need a validation set
  - Typically, one uses about 80% of the training data for training and 20% for validation.

Cross validation

- It is used to estimate generalization error of a learning algorithm A when the given dataset D is too small for a simple train/test or train/valid split to yield accurate estimation of generalization error

- In k-fold cross-validation a partition of the dataset is formed by splitting it into k non-overlapping subsets.
  - The test error may then be estimated by taking the average test error across k trials. On trial i, the i-th subset of the data is used as the test set and the rest of the data is used as the training set.

Cross validation

Define `KFoldXV`$(\mathbb{D}, A, L, k)$:
Require: $\mathbb{D}$, the given dataset, with elements $\boldsymbol{z}^{(i)}$
Require: A, the learning algorithm, seen as a function that takes a dataset as input and outputs a learned function
Require: L, the loss function, seen as a function from a learned function f and an example $\boldsymbol{z}^{(i)} \in \mathbb{D}$ to a scalar $\in \mathbb{R}$
Require: k, the number of folds
Split $\mathbb{D}$ into k mutually exclusive subsets $\mathbb{D}_i$, whose union is $\mathbb{D}$.
for i from 1 to k **do**
  $f_i = A(\mathbb{D} \backslash \mathbb{D}_i)$
  for $\boldsymbol{z}^{(j)}$ in $\mathbb{D}_i$ **do**
    $e_j = L(f_i, \boldsymbol{z}^{(j)})$
  end for
end for
Return $\boldsymbol{e}$

Point estimation

- Provides the single “best” prediction of some quantity of interest
- can be a single parameter or a vector of parameters in some parametric model
- A **point estimator** or **statistic** is any function of the data (assuming i.i.d. samples):

$$\widehat{\boldsymbol{\theta}} = g(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)})$$

- Frequentist perspective: we assume that the true parameter value θ is fixed but unknown
- Function estimation is really just the same as estimating a parameter θ; the function estimator is simply a point estimator in function space.

Statistical bias

Expectation over the data $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}$

$$bias\left(\widehat{\boldsymbol{\theta}}_{\mathrm{m}}\right) = \mathbb{E}\left(\widehat{\boldsymbol{\theta}}_{\mathrm{m}}\right) - \boldsymbol{\theta}$$

- An estimator $\widehat{\boldsymbol{\theta}}_{\mathrm{m}}$ is said to be **unbiased** if $bias\left(\widehat{\boldsymbol{\theta}}_{\mathrm{m}}\right) = 0$
  - which implies that $\mathbb{E}\left(\widehat{\boldsymbol{\theta}}_{\mathrm{m}}\right) = \boldsymbol{\theta}$.
- An estimator $\widehat{\boldsymbol{\theta}}_{\mathrm{m}}$ is said to be **asymptotically unbiased** if $\lim_{m \to \infty} bias\left(\mathbb{E}\left(\widehat{\boldsymbol{\theta}}_{\mathrm{m}}\right)\right) = 0$

Example: Bernoulli distribution

- Consider a set of samples $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ that are i.i.d according to a Bernoulli distribution with mean θ
  - $P(x(i);\, \theta) = \theta^{x(i)}\,(1 - \theta)^{1-x(i)}$
  - A common estimator of θ is:

$$\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\begin{aligned}
\text{bias}(\hat{\theta}_m) &= \mathbb{E}[\hat{\theta}_m] - \theta && (5.23)\\
&= \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] - \theta && (5.24)\\
&= \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}\left[x^{(i)}\right] - \theta && (5.25)\\
&= \frac{1}{m}\sum_{i=1}^{m} \sum_{x^{(i)}=0}^{1} \left(x^{(i)}\theta^{x^{(i)}}(1-\theta)^{(1-x^{(i)})}\right) - \theta && (5.26)\\
&= \frac{1}{m}\sum_{i=1}^{m} (\theta) - \theta && (5.27)\\
&= \theta - \theta = 0 && (5.28)
\end{aligned}$$

Example: Gaussian distribution - mean

- Consider a set of samples $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ that are i.i.d according to a a Gaussian distribution:
  - $p\left(x^{(i)}\right) = \mathcal{N}\left(x^{(i)}, \mu, \sigma^2\right)$
- $\hat{\mu}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$

The sample mean is an unbiased estimator of Gaussian mean parameter.

$$
\begin{aligned}
\text{bias}(\hat{\mu}_m) &= \mathbb{E}[\hat{\mu}_m] - \mu \\
&= \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] - \mu \\
&= \left(\frac{1}{m}\sum_{i=1}^{m} \mathbb{E}\left[x^{(i)}\right]\right) - \mu \\
&= \left(\frac{1}{m}\sum_{i=1}^{m} \mu\right) - \mu \\
&= \mu - \mu = 0
\end{aligned}
$$

Example: Gaussian distribution - variance

- Biased estimator: $\hat{\sigma}_m^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2$

$$\text{bias}(\hat{\sigma}_m^2) = \mathbb{E}[\hat{\sigma}_m^2] - \sigma^2$$

$$\mathbb{E}[\hat{\sigma}_m^2] = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2\right]$$

$$= \frac{m-1}{m}\sigma^2$$

the bias of $\hat{\sigma}_m^2$ is $-\sigma^2/m$.

The sample variance is a biased estimator.

- Unbiased estimator:
- $\hat{\sigma}_m^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2$

Variance

- how much we expect an estimator to vary as a function of the data sample?
- $var(\hat{\theta}) = ?$
- Standard error $SE(\hat{\theta}) = ?$
- Just as we might like an estimator to exhibit low bias we would also like it to have relatively low variance.

$$\mathrm{SE}(\hat{\mu}_m) = \sqrt{\mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}},$$

Example: Bernoulli distribution

$$\begin{aligned}
\mathrm{Var}\left(\hat{\theta}_m\right) &= \mathrm{Var}\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right) \\
&= \frac{1}{m^2}\sum_{i=1}^{m} \mathrm{Var}\left(x^{(i)}\right) \\
&= \frac{1}{m^2}\sum_{i=1}^{m} \theta(1-\theta) \\
&= \frac{1}{m^2} m\theta(1-\theta) \\
&= \frac{1}{m}\theta(1-\theta)
\end{aligned}$$

- The variance of the estimator decreases as a function of m, the number of examples in the dataset.

Trading off Bias and Variance

$$\text{MSE} = \mathbb{E}[(\hat{\theta}_m - \theta)^2]$$
$$= \text{Bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m)$$

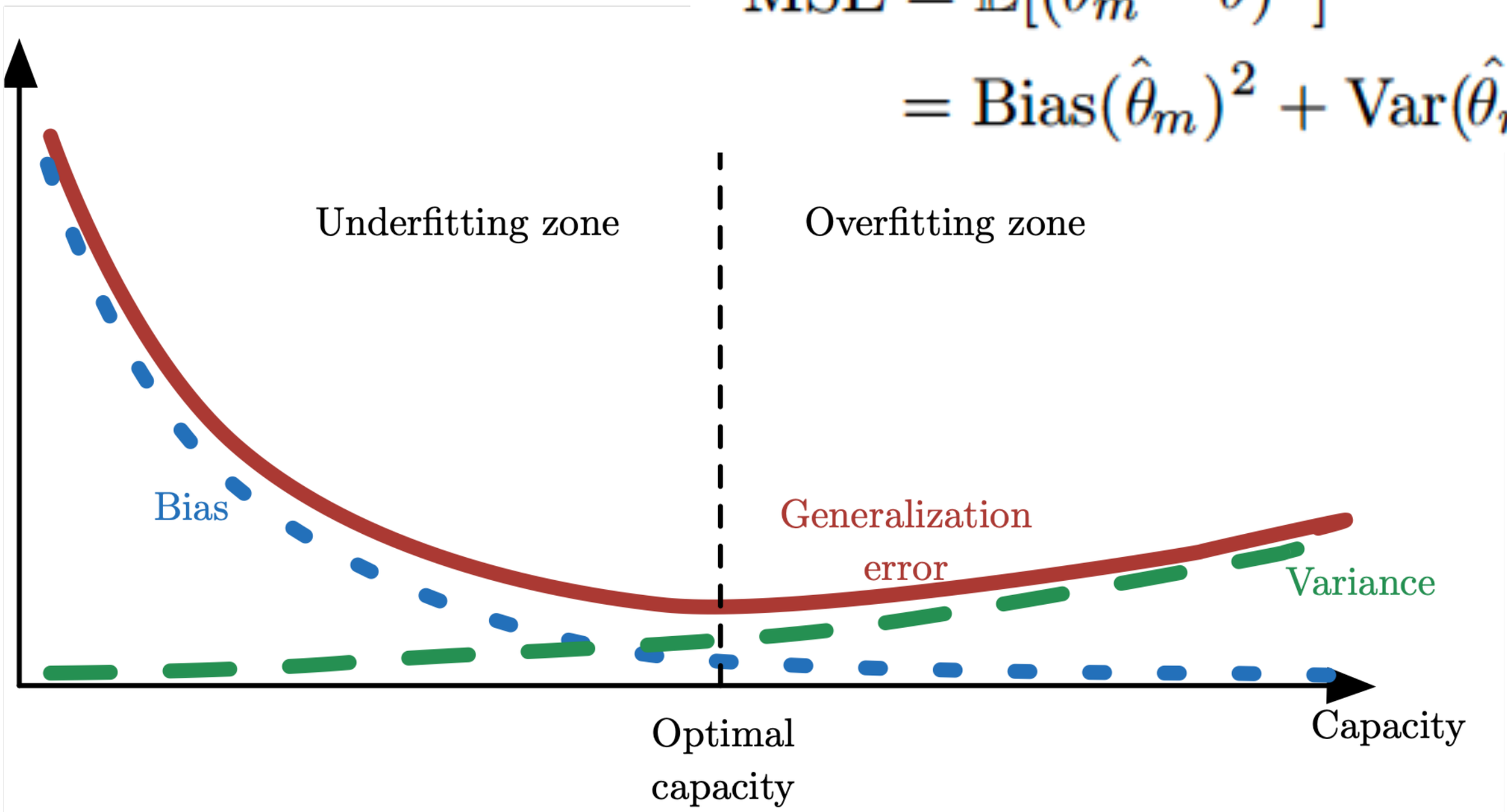


Figure 5.6

Bias vs. variance

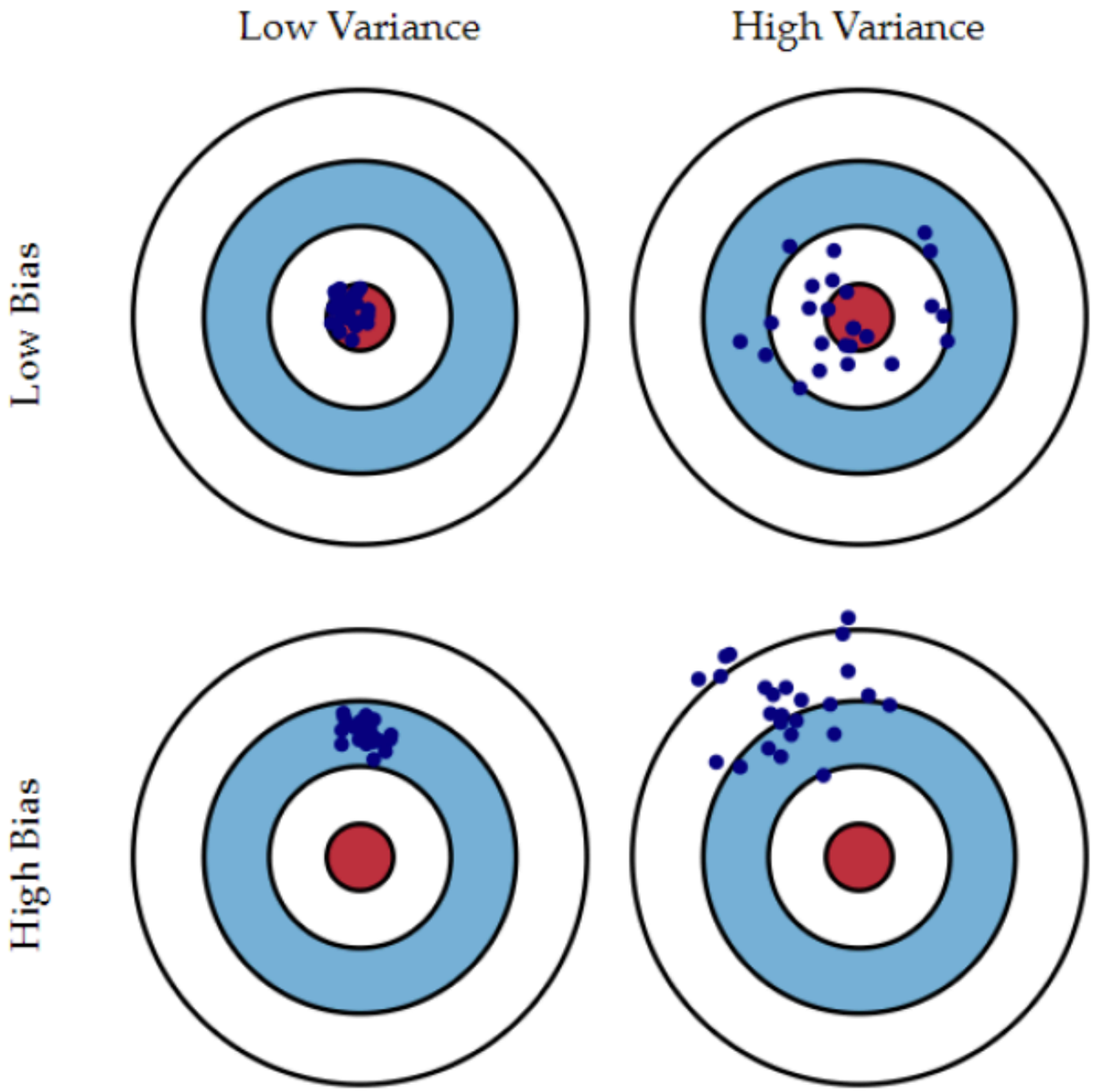
- **Bias measures the expected deviation from the true value of θ**

$$bias(\hat{\theta}_m) = \mathrm{E}[\hat{\theta}_m] - \theta$$

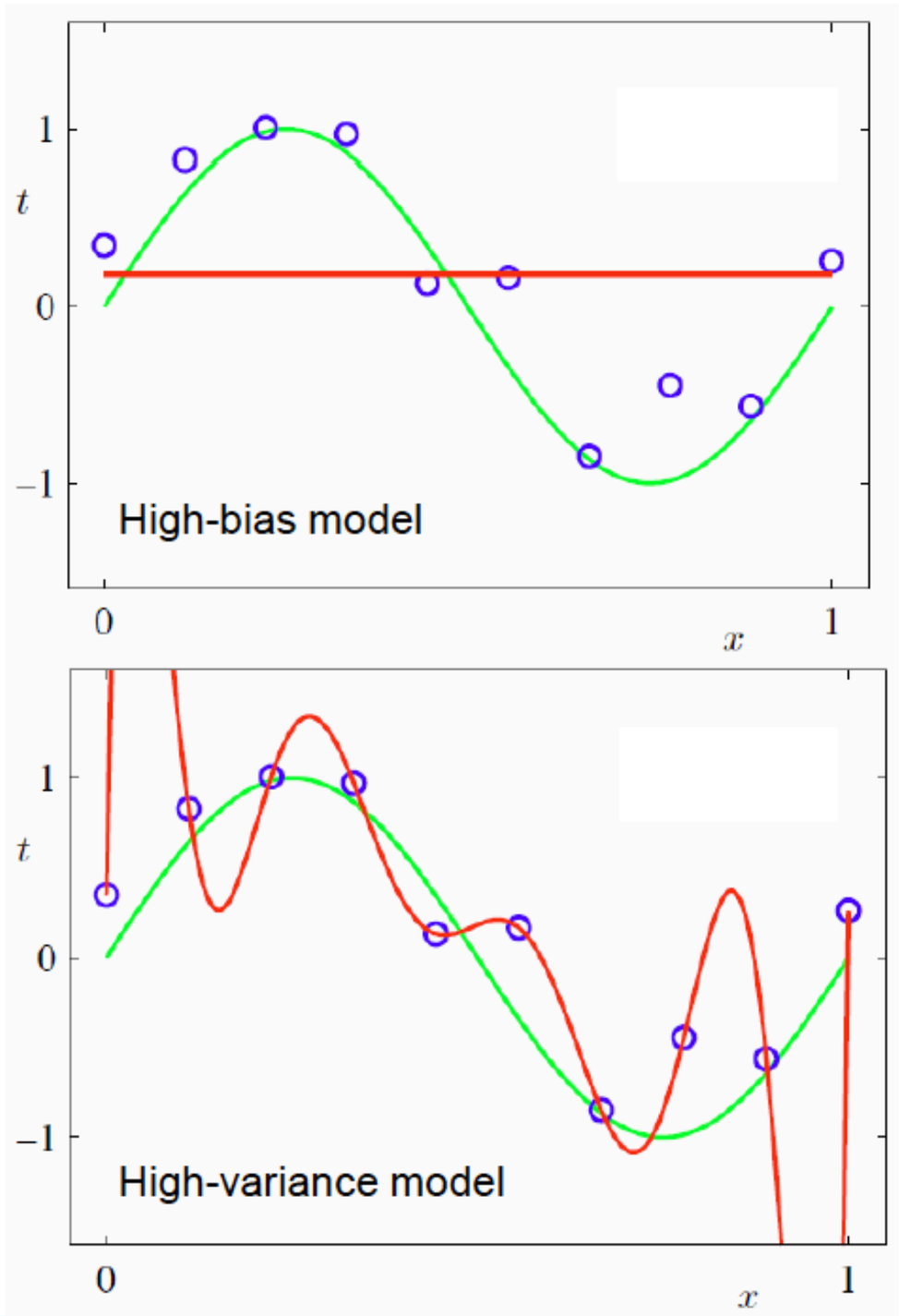
- **Variance measures the deviation from the expected estimator value that any particular sampling of the data is likely to cause**

$$Var(\hat{\theta})$$

Bias vs. variance



Scott Fortmann-Roe, Understanding the Bias-Variance Tradeoff, 2012



(Goodfellow 2016)

Maximum Likelihood Estimation

- Rather than guessing that some function might make a good estimator and then analyzing its bias and variance, we would like to have some principle from which we can derive specific functions that are good estimators for different models.

$$\mathbb{X} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$$

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\mathrm{model}}(\mathbb{X}; \boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

Same as: $\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}).$

Maximum Likelihood Estimation

- Divide by *m*: $\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta})$
- MLE minimizes the dissimilarity between the empirical distribution $\hat{p}_{data}$ and the model distribution

Empirical distribution

$$D_{\mathrm{KL}}\left(\hat{p}_{\mathrm{data}} \| p_{\mathrm{model}}\right) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\mathrm{data}}}\left[\log \hat{p}_{\mathrm{data}}(\boldsymbol{x}) - \log p_{\mathrm{model}}(\boldsymbol{x})\right]$$

Independent of θ

Any loss consisting of a negative log-likelihood is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by model.

Conditional log-likelihood

- Supervised learning $\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{Y} \mid \boldsymbol{X}; \boldsymbol{\theta})$

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log P(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}).$$

- Instead of producing a single prediction $\hat{y}$, we now think of the model as producing a conditional distribution $p(y \mid \boldsymbol{x})$.
- For linear regression, take:

Output of linear regression as the mean

fixed

$$p(y \mid \boldsymbol{x}) = \mathcal{N}(y; \hat{y}(\boldsymbol{x}; \boldsymbol{w}), \sigma^2)$$

Conditional log-likelihood for linear regression

- To maximize: $$\sum_{i=1}^{m} \log p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= -m \log \sigma - \frac{m}{2} \log(2\pi) - \sum_{i=1}^{m} \frac{\left\| \hat{y}^{(i)} - y^{(i)} \right\|^2}{2\sigma^2},$$

From the dataset

- Same as minimizing:

$$\text{MSE}_{\text{train}} = \frac{1}{m} \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||^2$$

Output of linear regression as the mean

Properties of maximum likelihood

- The best estimator asymptotically, as the number of examples $m \rightarrow \infty$, in terms of its rate of convergence as m increases.
- Consistency: $\operatorname{plim}_{m \rightarrow \infty} \hat{\theta}_m = \theta$
  - Under appropriate conditions
    - The true distribution must lie within the model family
    - The true distribution must correspond to exactly one value of θ

Bayesian statistics

- the true parameter θ is unknown or uncertain and thus is represented as a random variable.

$$p(\boldsymbol{\theta} \mid x^{(1)}, \ldots, x^{(m)}) = \frac{p(x^{(1)}, \ldots, x^{(m)} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(x^{(1)}, \ldots, x^{(m)})}$$

- It is different than the single point estimate (frequentist statistics)

$$p(x^{(m+1)} \mid x^{(1)}, \ldots, x^{(m)}) = \int p(x^{(m+1)} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid x^{(1)}, \ldots, x^{(m)}) \, d\boldsymbol{\theta}.$$

Maximum A Posteriori (MAP) Estimation

- Add the prior:

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

- MAP Bayesian inference has the advantage of leveraging information that is brought by the prior and cannot be found in the training data.
  - If this prior is given by $\mathcal{N}\left(\boldsymbol{w}, \mathbf{0}, \frac{1}{\lambda}\boldsymbol{I}^2\right)$, then the log-prior term is proportional to the familiar $\lambda w^T w$ weight decay penalty

Supervised learning examples

- Logistic regression
- Support vector machines
- Decision trees

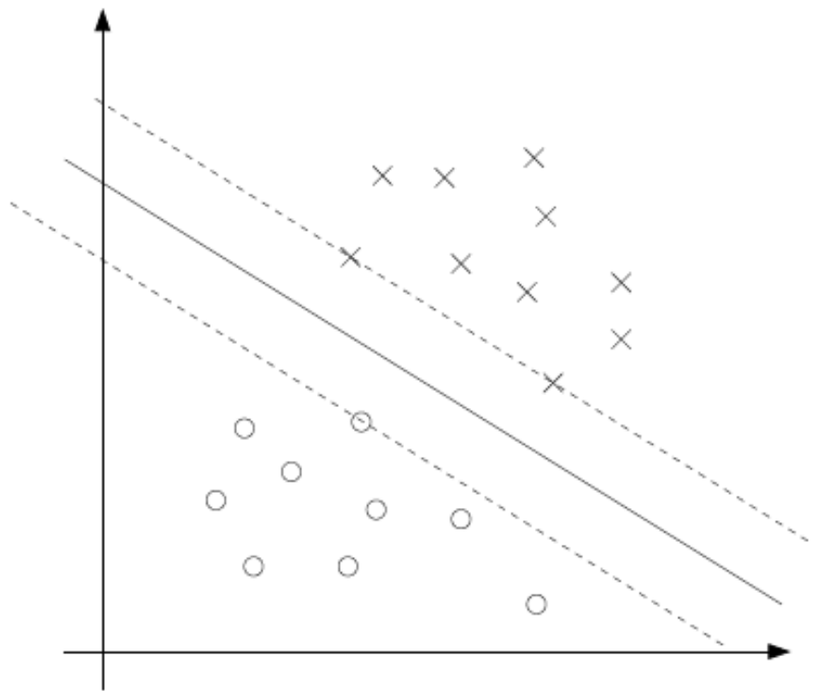
Logistic regression

$$p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}).$$

- There is no closed-form solution for the optimal weights.
- Instead, we must search for them by maximizing the log-likelihood.
- We can do this by minimizing the negative log-likelihood (NLL) using gradient descent.

Support vector machines

- Introduced by Vapnik in the 90s
- Widest street approach
- Decision rule
  - Which side of the street?
    - $\boldsymbol{wx} + b \geq 0 \Rightarrow +$
    - $\boldsymbol{wx} + b \leq 0 \Rightarrow -$



- **w** is perpendicular to the median line of the street, but which **w**?
  - We add the following scaling constraints:
    - $\boldsymbol{wx}_+ + b \geq 1$
    - $\boldsymbol{wx}_- + b \leq -1$

SVM

- $y_i = +1$ for + samples
- $y_i = -1$ for – samples
- $y_i(\boldsymbol{w}\boldsymbol{x_i} + b) \geq 1$ same equation for x+ and x-
- $y_i(\boldsymbol{w}\boldsymbol{x_i} + b) - 1 \geq 0$
- Constraints: $y_i(\boldsymbol{w}\boldsymbol{x_i} + b) - 1 = 0$ for $\boldsymbol{x_i}$ in the gutter
- $width = (\boldsymbol{x}_+ - \boldsymbol{x}_-)\frac{\mathbf{w}}{\|\boldsymbol{w}\|} = \frac{(1-\mathrm{b})}{\|\boldsymbol{w}\|} - \frac{-(1+\mathrm{b})}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$
- Maximize the width of the street:
  - $\max \frac{2}{\|\boldsymbol{w}\|} \Rightarrow \max \frac{1}{\|\boldsymbol{w}\|} \Rightarrow \min \frac{\|\boldsymbol{w}\|^2}{2}$

Lagrange multipliers

- $L = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum \alpha_i [y_i(\boldsymbol{w}\boldsymbol{x_i} + b) - 1]$ $\alpha_i \geq 0$
- $\nabla_{\boldsymbol{w}} L = \boldsymbol{w} - \sum \alpha_i y_i \boldsymbol{x_i}$
  - $\boldsymbol{w} - \sum \alpha_i y_i \boldsymbol{x_i} = \boldsymbol{0} \Rightarrow \boldsymbol{w} = \sum \alpha_i y_i \boldsymbol{x_i}$
- $\frac{\partial L}{\partial b} = \sum \alpha_i y_i \Rightarrow \sum \alpha_i y_i = 0$
- Replacing in L :
  - $L = \frac{1}{2}\sum \alpha_i y_i \boldsymbol{x_i} \sum \alpha_j y_j \boldsymbol{x_j} - \sum \alpha_i y_i \boldsymbol{x_i} \sum \alpha_j y_j \boldsymbol{x_j} - \sum \alpha_i y_i b + \sum \alpha_i$
  - $L = \sum \alpha_i - \frac{1}{2}\sum \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \boldsymbol{x_j}$
- Numerical analysts will solve this problem (max with respect to $\alpha_i \geq 0$) and give us the α_i

Kernel trick

- $\alpha_i \neq 0$ *only* for points in the gutter, also known as support vectors
- Decision for $\boldsymbol{x_j}$: $\boldsymbol{w x_j} + b = \sum \alpha_i y_i \boldsymbol{x_i x_j} + b$
- We can change the representation of $\boldsymbol{x_i}$ to be $\phi(\boldsymbol{x_i})$
- We do not need direct access to $\phi(\boldsymbol{x_i})$
  - Instead we define $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \phi(\boldsymbol{x_i})\phi(\boldsymbol{x_j})$
  - The function *k* is called kernel
- $f(\boldsymbol{x}) = \sum \alpha_i y_i k(\boldsymbol{x_i}, \boldsymbol{x}) + b$
  - This function is nonlinear with respect to $\boldsymbol{x}$

Advantages of the kernel trick

- The kernel trick is powerful for two reasons.
  - it allows us to learn models that are nonlinear as a function of $\boldsymbol{x}$ using convex optimization techniques that are guaranteed to converge efficiently
  - the kernel function often admits an implementation that is significantly more computational efficient than naively constructing two $\phi(\boldsymbol{x})$ vectors and explicitly taking their dot product.
    - In some cases, $\phi(\boldsymbol{x})$ can even be infinite dimensional

Gaussian kernel

- Gaussian kernel $k(\boldsymbol{u}, \boldsymbol{v}) = \mathcal{N}(\boldsymbol{u} - \boldsymbol{v}; 0, \sigma^2 \boldsymbol{I})$
  - Aka RBF: Radial Basis Function

- We can think of the Gaussian kernel as performing a kind of template matching.
  - A training example x associated with training label y becomes a template for class y.
  - When a test point x' is near x according to Euclidean distance, the Gaussian kernel has a large response, indicating that x' is very similar to the x template.
  - The model then puts a large weight on the associated training label y.
- Overall, the prediction will combine many such training labels weighted by the similarity of the corresponding training examples.

SVM limitations

- high computational cost of training when the dataset is large.
- The current deep learning renaissance began when Hinton et al. (2006) demonstrated that a neural network could outperform the RBF kernel SVM on the MNIST benchmark.

Watch (SVM)

- https://www.youtube.com/watch?v=_PwhiWxHK8o

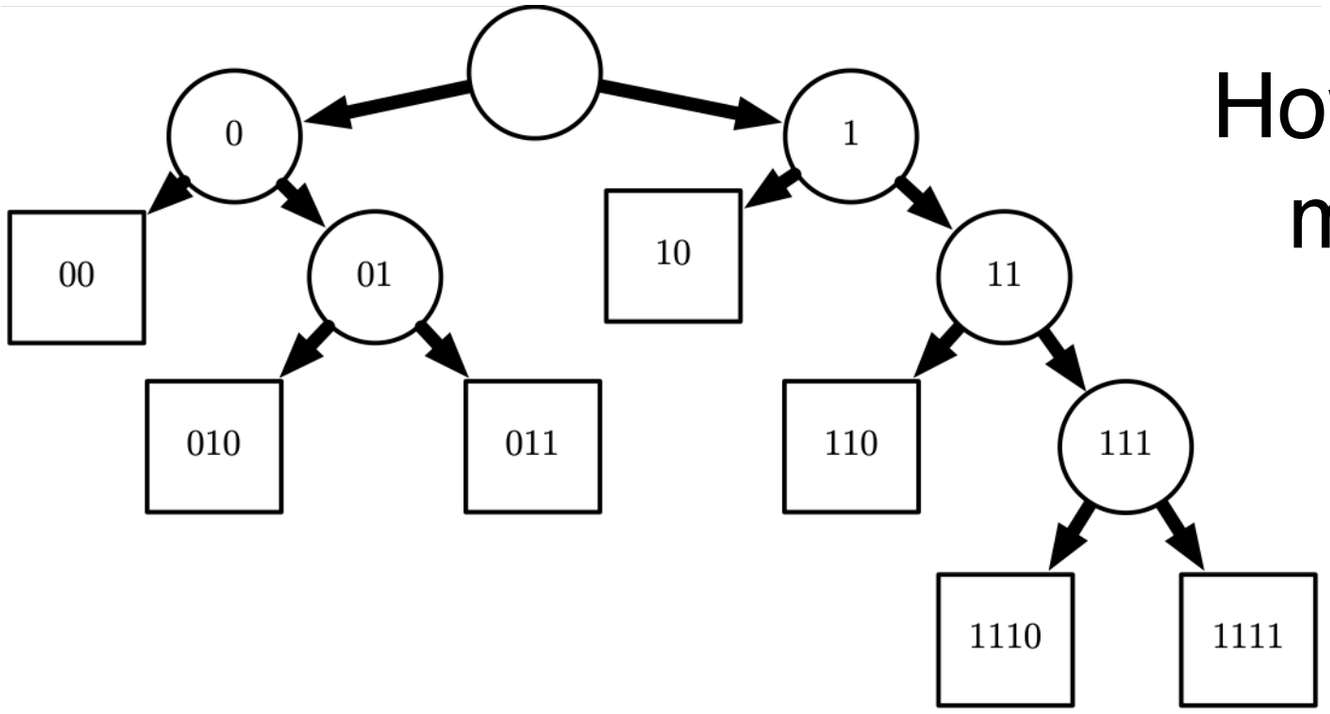
K-Nearest Neighbors

- There is not even really a training stage or learning process.
  - We find the k-nearest neighbors to $\boldsymbol{x}$ in the training data $\boldsymbol{X}$. We then return the average of the corresponding y values in the training set
- In the case of classification, we can average over one-hot code vectors
  - We can then interpret the average over these one-hot codes as giving a probability distribution over classes

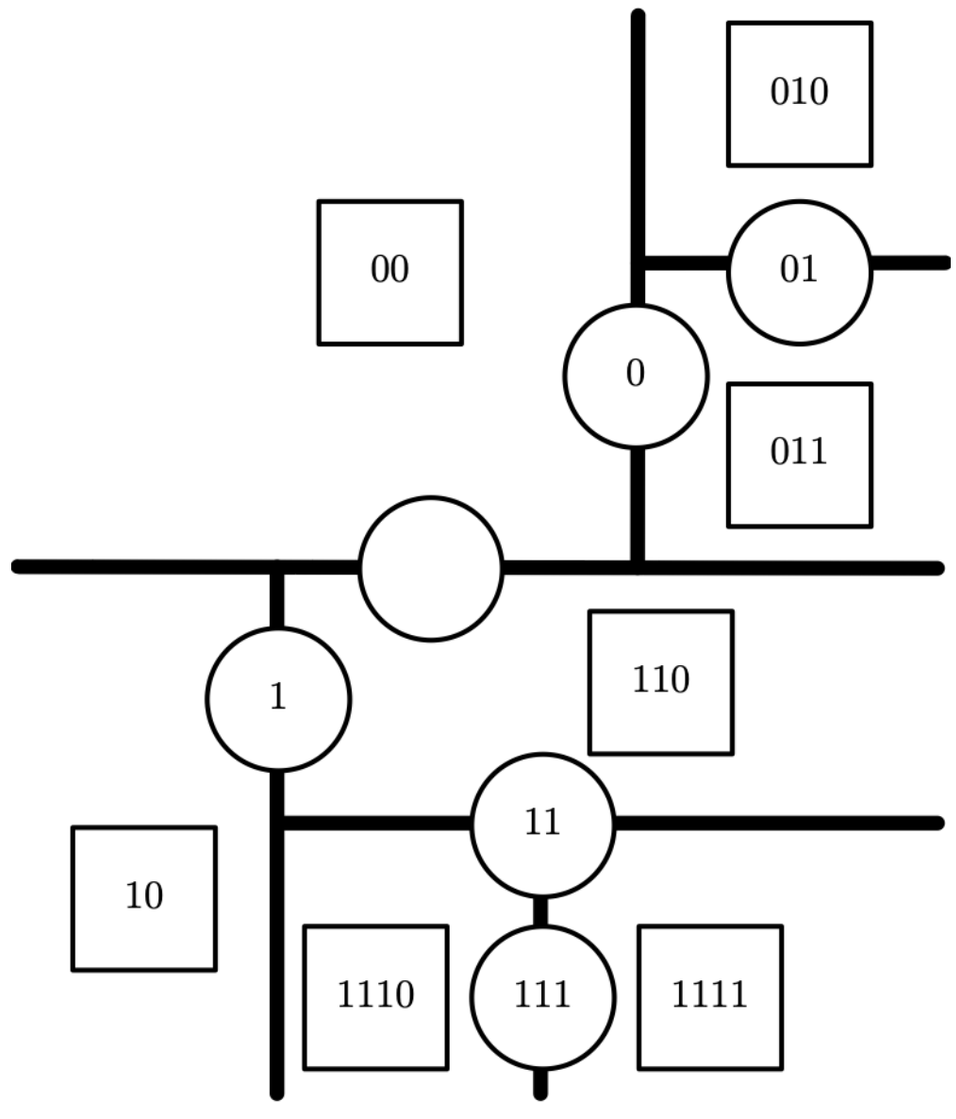
K-NN limitations

- It cannot learn that one feature is more discriminative than another.
  - For example, imagine we have a regression task with $x \in \mathbb{R}^{100}$ drawn from an isotropic Gaussian distribution,
  - but only a single variable is relevant to the output $y = x_1$
  - Nearest neighbor regression will not be able to detect this simple pattern.

Decision Trees



How a decision tree might divide $\mathbb{R}^2$.



Each leaf requires at least one training example to define,
so it is not possible for the decision tree to learn a function that has more local maxima than the number of training examples.

Figure 5.7

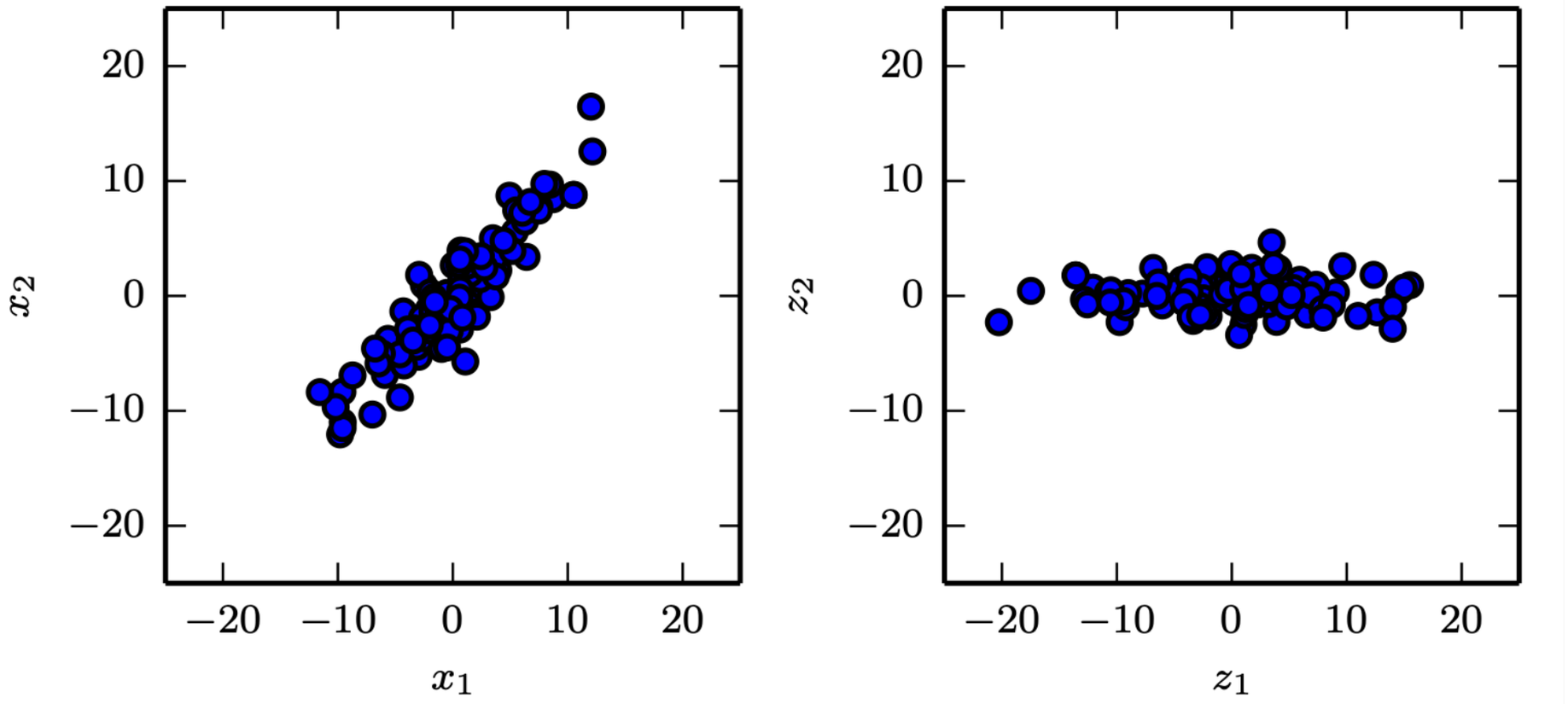
Decision trees limitations

- It struggles to solve some problems that are easy even for logistic regression.
- If we have a two-class problem and the positive class occurs wherever $x_2 > x_1$
  - the decision boundary is not axis-aligned. The decision tree will thus need to approximate the decision boundary with many splits, implementing a step function that constantly walks back and forth across the true decision function.

Unsupervided learning

- Density estimation
- Learning to draw samples from a distribution
- Learning to denoise data from some distribution
- Clustering the data into groups of related examples
- Simpler representation:
  - Lower-dimensional representation
  - Sparse representation
  - Independent representation

Principal Components Analysis



- lower dimensionality
- **no linear correlation**

No linear correlation

- Let us consider the $m \times n$-dimensional design matrix $\boldsymbol{X}$.
- We will assume that the data has a mean of zero, $\mathbb{E}[\boldsymbol{x}] = 0$.
  - If this is not the case, the data can easily be centered by subtracting the mean from all examples in a preprocessing step.
- The unbiased sample covariance matrix associated with $\boldsymbol{X}$ is given by:
  - $Var[\boldsymbol{x}] = \frac{1}{m-1} \boldsymbol{X}^T \boldsymbol{X}$
- PCA finds $\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{x}$ where $Var[\boldsymbol{z}]$ is diagonal.

Decorrelation

- PCA finds $\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{x}$ where $Var[\boldsymbol{z}]$ is diagonal.
  - Remember that $\boldsymbol{X}^T \boldsymbol{X} = \boldsymbol{W} \boldsymbol{\Lambda} \boldsymbol{W}^T$
- $Var[\boldsymbol{z}] = \frac{1}{m-1} \boldsymbol{Z}^T \boldsymbol{Z} = \frac{1}{\boldsymbol{m-1}} \boldsymbol{W}^T \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{W} = \frac{1}{m-1} \boldsymbol{W}^T \boldsymbol{W} \boldsymbol{\Lambda} \boldsymbol{W}^T \boldsymbol{W} =$
- $Var[\boldsymbol{z}] = \frac{1}{m-1} \boldsymbol{\Lambda}$ since $\boldsymbol{W}^T \boldsymbol{W} = \boldsymbol{I}$
- PCA disentangles the unknown factors of variation underlying the data.
  - this disentangling takes the form of finding a rotation of the input space that aligns the principal axes of variance with the basis of the new representation space associated with z.

k-means clustering

- Divides the training set into *k* different clusters of examples
- Provides a *k*-dimensional one-hot code vector h representing an input x.
  - If x belongs to cluster i, then $h_i = 1$ and all other entries of the representation h are zero.
  - This is an example of a sparse representation

k-means algorithm

- initialize k different centroids $\{\mu(1), \ldots, \mu(k)\}$ to different values,
- then alternate between two different steps until convergence:
  - In one step, each training example is assigned to cluster i, where i is the index of the nearest centroid $\mu(i)$.
  - In the other step, each centroid $\mu(i)$ is updated to the mean of all training examples $x(j)$ assigned to cluster i.

How to evaluate clustering?

- Suppose we have images of
  - Red trucks, gray trucks
  - Red cars, gray cars
- One clustering algorithm may find a cluster of cars and a cluster of trucks
- Another learning algorithm may find a cluster of red vehicules and a cluster of gray vehicules
- A third algorithm may find 4 clusters …
  - Red cars similarity to gray trucks is same as their similarity to gray cars!
- A **distributed representation** may have two attributes for each vehicule

Building a machine learning algorithm

- Simple recipe
  - A dataset
  - A cost function (e.g. MSE, negative log-likelihood)
  - A model (e.g. linear, non-linear)
  - An optimization procedure (closed form, gradient descent, stochastic gradient descent …)

Exercise

- Link equations and definitions

$$J(\boldsymbol{w}, b) = -\mathbb{E}_{\mathbf{x},y\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(y \mid \boldsymbol{x})$$

$$p_{\text{model}}(y \mid \boldsymbol{x}) = \mathcal{N}(y; \boldsymbol{x}^\top \boldsymbol{w} + b, 1)$$

$$J(\boldsymbol{w}) = \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}} ||\boldsymbol{x} - r(\boldsymbol{x}; \boldsymbol{w})||_2^2$$

$$J(\boldsymbol{w}, b) = \lambda ||\boldsymbol{w}||_2^2 - \mathbb{E}_{\mathbf{x},y\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(y \mid \boldsymbol{x})$$

$$r(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} \boldsymbol{w}$$

Stochastic gradient descent

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} L(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$L(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\log p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$$

Computing the gradient is O(m)

Sample a mini-batch $\mathbb{B} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')}\}$

$$\boldsymbol{g} = \frac{1}{m'}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

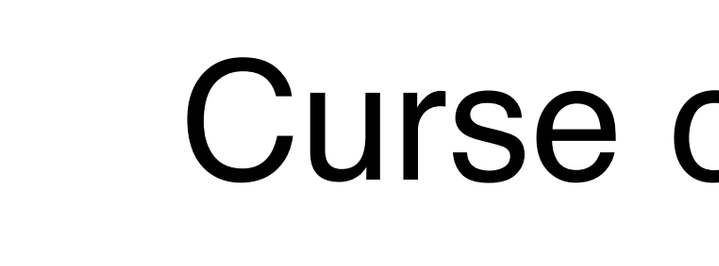
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}$$

Challenges motivating deep learning

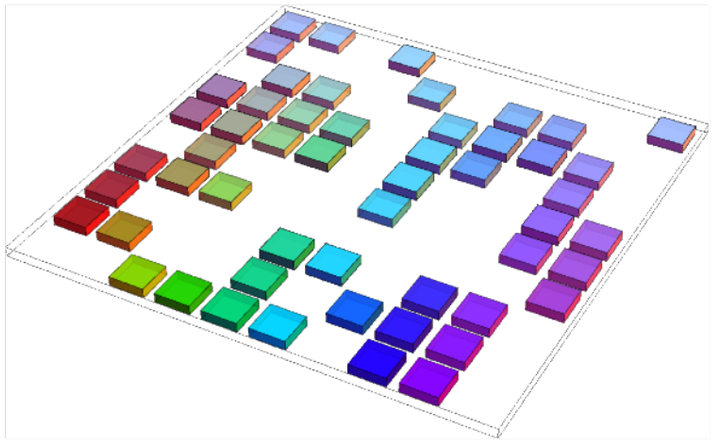
- Simple machine learning algorithms work very well on a wide variety of important problems.
- However, they have not succeeded in solving the central problems in AI, such as recognizing speech or recognizing objects.
- Generalizing to new examples becomes exponentially more difficult when working with high-dimensional data,
- Such data also often impose high computational costs.

Curse of Dimensionality

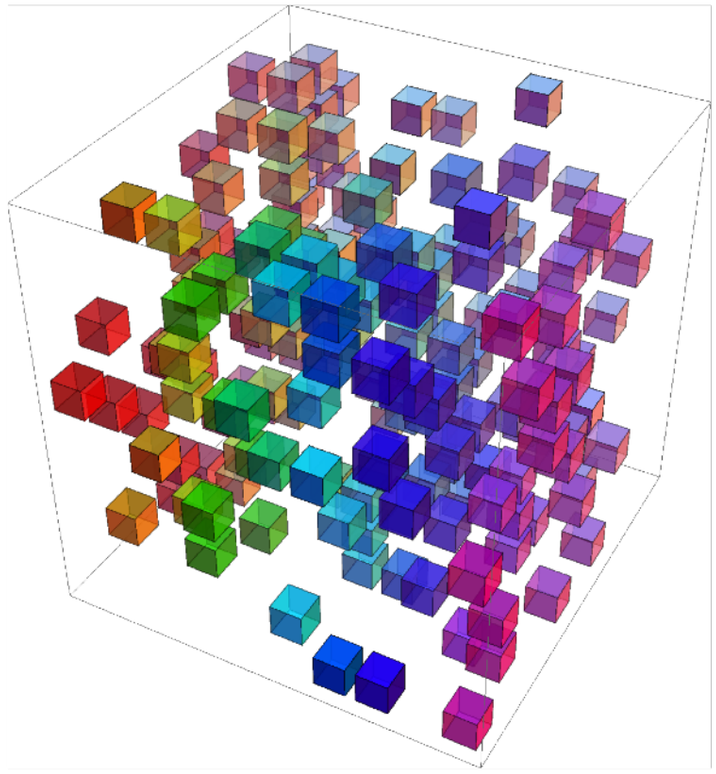
Many machine learning problems become exceedingly difficult when the number of dimensions in the data is high.



one variable, 10 regions of interest.



two variables, 10 regions of interest each.

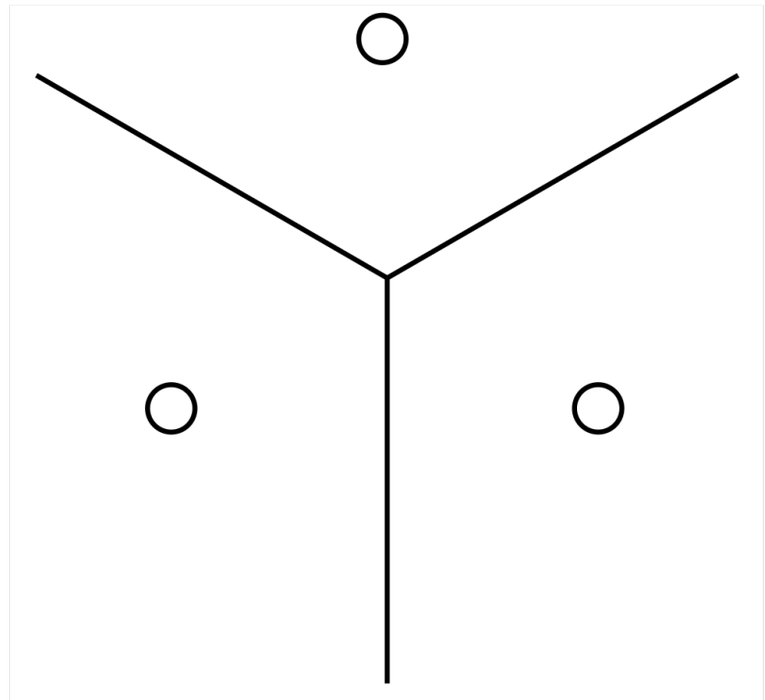


three variables, 10 regions of interest each.

For d dimensions and v values to be distinguished along each axis, we seem to need $O(v^d)$ regions and examples

Smoothness prior (aka local consistency)

- the function we learn should not change very much within a small region. $f^*(\boldsymbol{x}) \approx f^*(\boldsymbol{x} + \epsilon)$
  - KNN
  - RBF kernel
  - Decision trees
- to distinguish $O(k)$ regions in input space, all of these methods require $O(k)$ examples



Voronoi diagram

Smoothness prior is not enough!

- Is there a way to represent a complex function that has many more regions to be distinguished than the number of training examples?
  - A checkerboard contains many variations but there is a simple structure to them.
- The answer is Yes:
  - If we introduce some dependencies between the regions
  - via additional assumptions about the underlying data generating distribution

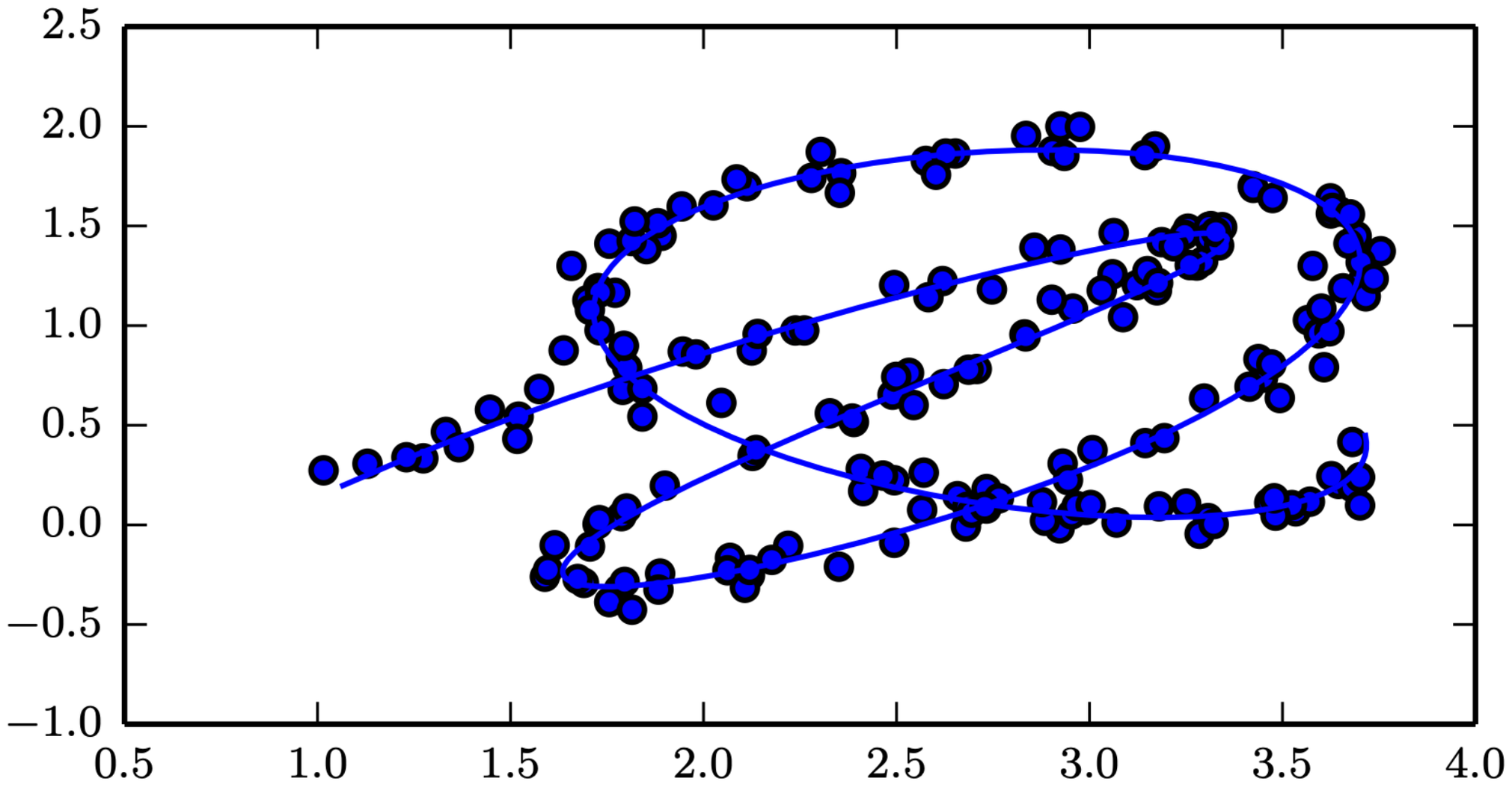
Deep learning solution

- Deep learning assumes that the data was generated by the composition of factors or features, potentially at **multiple levels in a hierarchy**
- These apparently mild assumptions allow an exponential gain in the relationship between the number of examples and the number of regions that can be distinguished.
- *Deep and distributed representations counter the curse of dimensionality.*

Manifold

- A manifold is a connected set of points
  - that can be approximated well by considering only a small number of degrees of freedom, or dimensions, embedded in a higher-dimensional space.
- A manifold is a set of points, associated with a neighborhood around each point.
  - Transformations can be applied to move on the manifold from one position to a neighboring one.
- From any given point, the manifold locally appears to be an Euclidean space.
  - In everyday life, we experience the surface of the world as a 2-D plane, but it is in fact a spherical manifold in 3-D space.

Manifold



One-dimensional manifold embedded in two dimensional space.

Manifold

- we allow the dimensionality of the manifold to vary from one point to another.
  - This often happens when a manifold intersects itself.
  - For example, a figure eight is a manifold that has a single dimension in most places but two dimensions at the intersection at the center.

Manifold learning

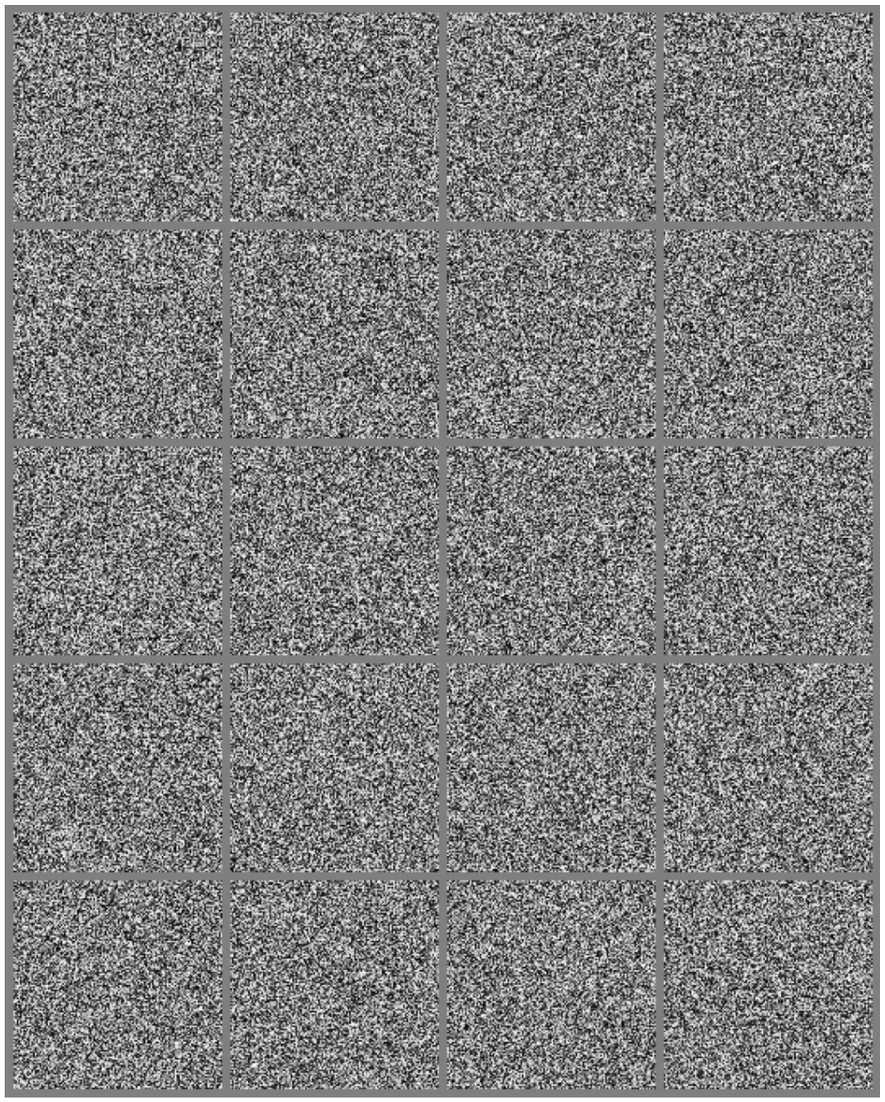
- Manifold learning algorithms assumes that most of $\mathbb{R}^n$ consists of invalid inputs,
  - interesting inputs occur only along a collection of manifolds
  - Interesting variations in the output of the learned function occurring only along directions that lie on the manifold,
  - or with interesting variations happening only when we move from one manifold to another.

the key assumption remains
that probability mass is
highly concentrated.

Arguments supporting the manifold hypothesis

- the probability distribution over images, text strings, and sounds that occur in real life is highly concentrated.

Uniformly Sampled Images



Arguments supporting the manifold hypothesis

- Examples we encounter are connected to each other by applying transformations to traverse the manifold.

QMUL Dataset

Conclusion

- The manifold assumption is at least approximately correct.
- This concludes part I of the course
- You are now prepared to embark upon your study of deep learning.

Congratulations