Machine Learning Basics

Chia-Chi Tsai (蔡家齊)
cctsai@gs.ncku.edu.tw
AI System Lab
Department of Electrical Engineering
National Cheng Kung University

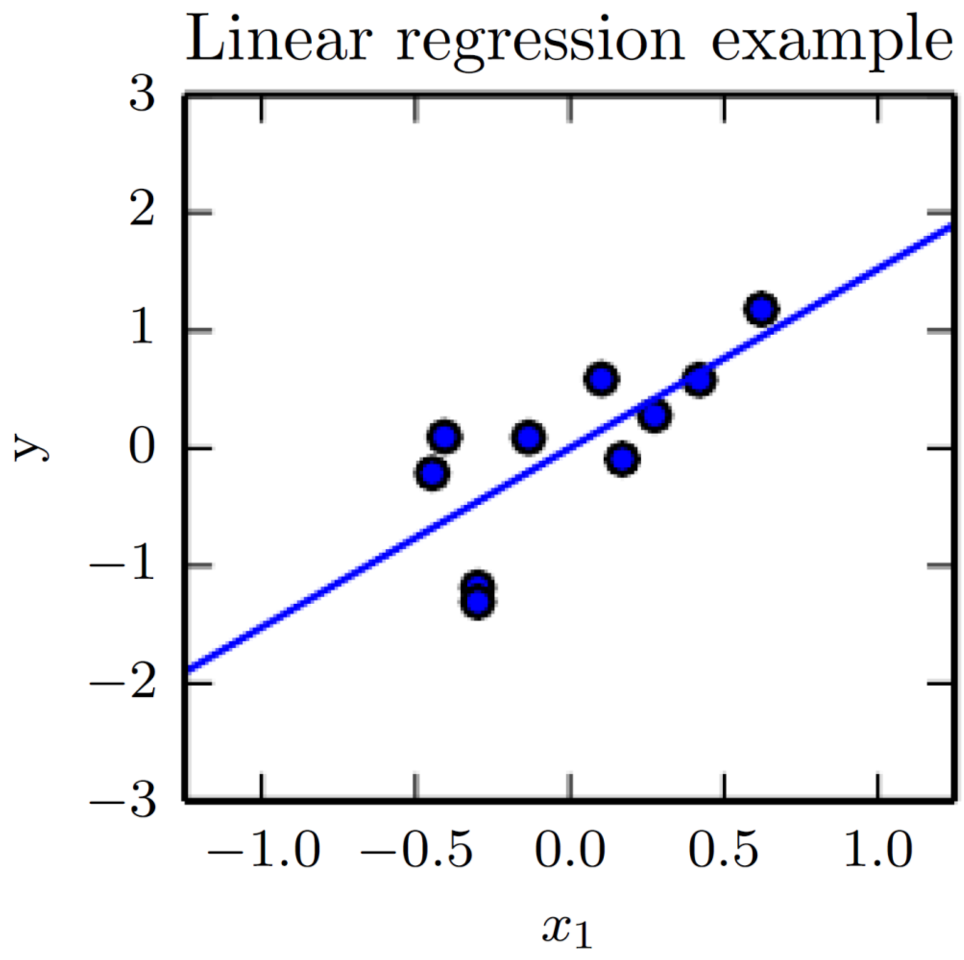
Outline

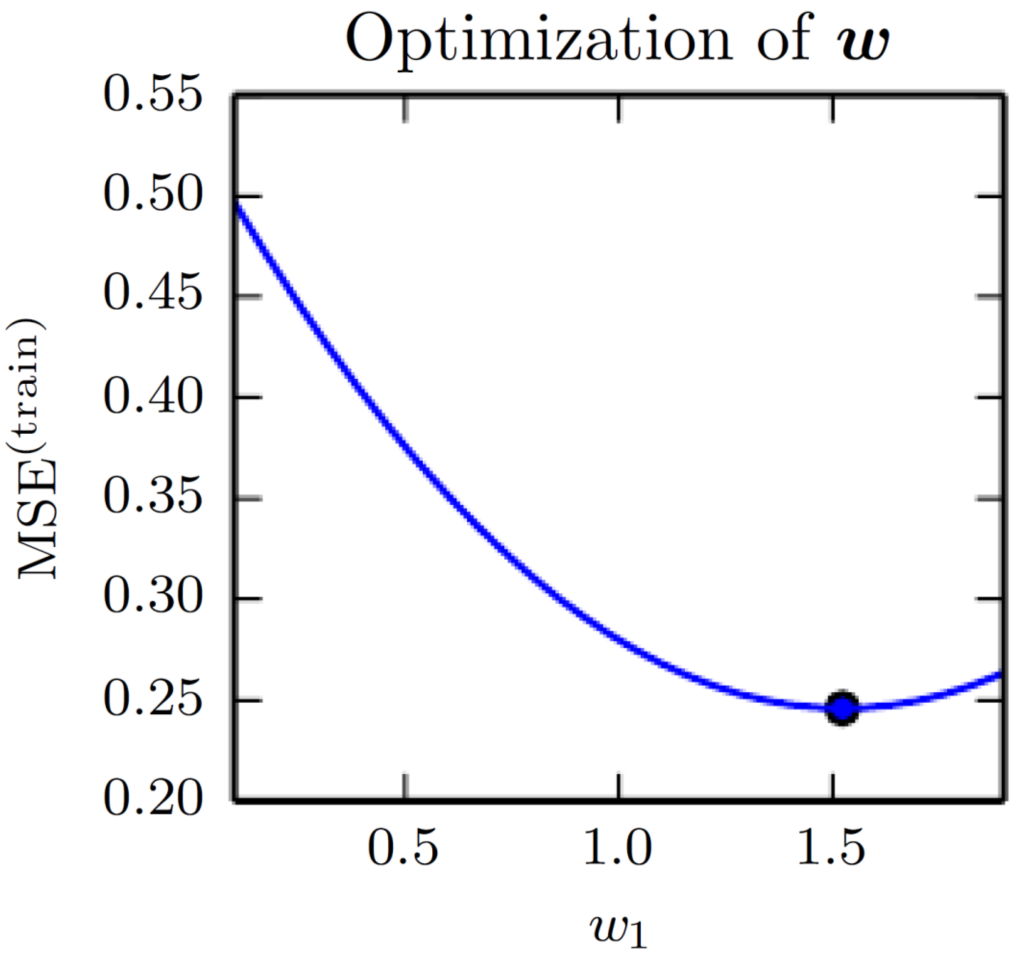
- Learning Algorithm
- Estimator, Bias and Variance
- Maximum Likelihood Estimation
- Bayesian Statistics
- Supervised/Unsupervised Learning
- Challenges Motivating Deep Learning

Learning Algorithms

(Mitchell, 1997) 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

A Linear Regression Problem





Example - Linear Regression

- Example Linear Regression

$$\begin{tikzcd} & & & & & t \arrow[d] & \\ x \arrow[r] & \boxed{\text{Rep}} \arrow[r] & \phi(x) \arrow[r] & \boxed{\text{Model}} \arrow[r] & y \arrow[r] & \boxed{-} \arrow[r] & Cost \end{tikzcd}$$

  - Task T: To predict y from x by outputting
$$\hat{y} = w^T\phi(x) = \phi(x)^T w$$
  - Experience E: To learn w by minimizing, over a training set $\left(X^{(train)}, y^{(train)}\right)$,
$$MSE^{(train)} = \frac{1}{m^{(train)}}\left\|\hat{y}^{(train)} - y^{(train)}\right\|_2^2$$

where

$$\hat{y}^{(train)} = \Phi^{(train)}w, \Phi^{train} = \begin{bmatrix} \phi(x_0^{(train)})^T \\ \phi(x_1^{(train)})^T \\ \vdots \\ \phi(x_{m-1}^{(train)})^T \end{bmatrix}$$

$$y^{(train)} = (y_0^{(train)}, y_1^{(train)}, \dots, y_{m-1}^{(train)})$$

Example - Linear Regression

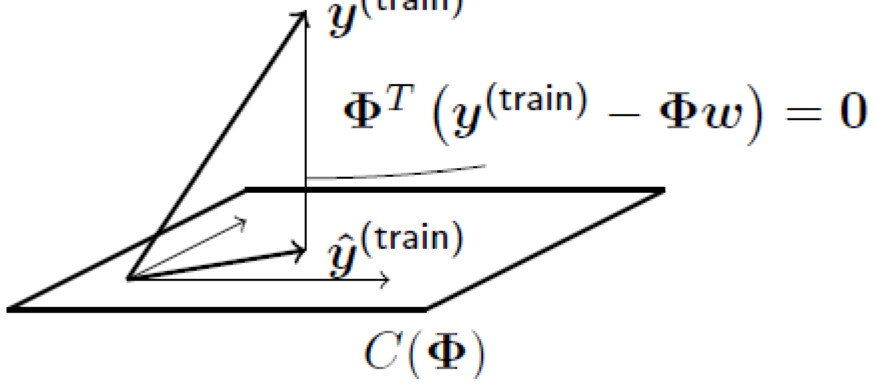
- Performance P: To measure mean squared error on a test set $\left(X^{(test)}, y^{(test)}\right)$, i.e.,

$$MSE^{(test)} = \frac{1}{m^{(test)}} \left\| \hat{y}^{(test)} - y^{(test)} \right\|_2^2$$

- To minimize $MSE^{(train)}$, w can be solved by setting

$$\nabla_w MSE^{(train)} = 0$$

- A geometrical view is to solve $\hat{y}^{(train)}$as the projection of $y^{(train)}$ onto the column space of $\Phi^{(train)}$



$y^{(\text{train})}$
$\Phi^T \left(y^{(\text{train})} - \Phi w\right) = 0$
$\hat{y}^{(\text{train})}$
$C(\Phi)$

- We then have

$$w = (\Phi^{(train)^T} \Phi^{(train)})^{-1} \Phi^{(train)^T} y^{train}$$

Example - Linear Regression

- The present model can be extended to include a bias term b

$$\hat{y} = w^{\mathrm{T}}\phi(x) + b = \widetilde{w}^{T}\tilde{\phi}(x),$$

with

$$\widetilde{w} = \begin{bmatrix} w \\ b \end{bmatrix}, \tilde{\phi}(x) = \begin{bmatrix} \phi(x) \\ 1 \end{bmatrix}$$

- Prediction with a polynomial of degree 2

$$\tilde{y} = w_2 x^2 + w_1 x^1 + b = \widetilde{w}^{T}\tilde{\phi}(x)$$

where

$$\widetilde{w} = \begin{bmatrix} w_2 \\ w_1 \\ b \end{bmatrix}, \tilde{\phi}(x) = \begin{bmatrix} \phi_2(x) \\ \phi_1(x) \\ 1 \end{bmatrix} = \begin{bmatrix} x^2 \\ x^1 \\ 1 \end{bmatrix}$$

Dataset

- ML tasks are usually described in terms of how the ML system should process an example.
- A dataset is a collection of many examples.
- ML algorithms can be broadly categorized as **supervised** and **unsupervised** by what kind of dataset they process.
  - Supervised: each example of the dataset is associated with a label or target
    - E.g., Classification, regression
  - Unsupervised: experience dataset without labels
    - E.g., Clustering
  - Reinforcement: Not a fixed dataset, interact with an environment
- Partition the dataset into:
  - Training set: where the model was trained
  - Test set: where the trained model was tested

Generalization

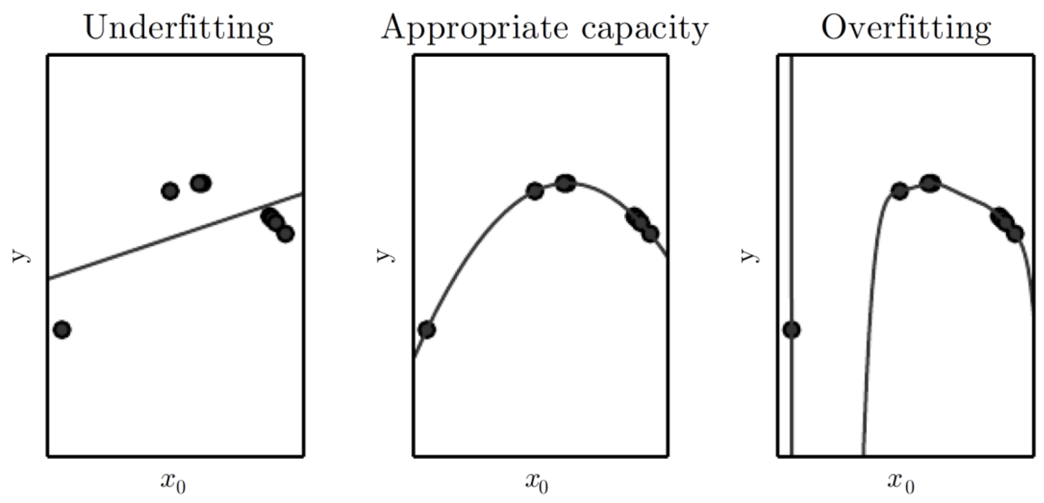
- The model's ability to perform well on **new**, previously **unseen** inputs
- Generalization error is defined to be the expected **value** of the **error** on a **new input** and is typically estimated by measuring the performance on a **test set** collected separately from the **training set**
- Examples (x, y) in the training and test sets are assumed to be drawn independently from the same distribution, $p_{data}(x, y)$
- **Bayes error**
  - Minimum generalization error achieved by an oracle model having knowledge of $p_{data}(x, y)$
  - E.g.: if $y = w^T \phi(x) + \varepsilon$ and ε is Gaussian noise independent of x, Bayes error = $Var(\varepsilon)$ with MSE measure

Training Error and Test Error

- Pitfall
    - At first glance, the expected test error should be the same as the expected training error for a given model, because the data in these sets are drawn from the same distribution
- In practice, we sample the training set, use it to train the model, and then sample the test set to measure test error
- Generally, test error ≥ training error

Underfitting vs. Overfitting

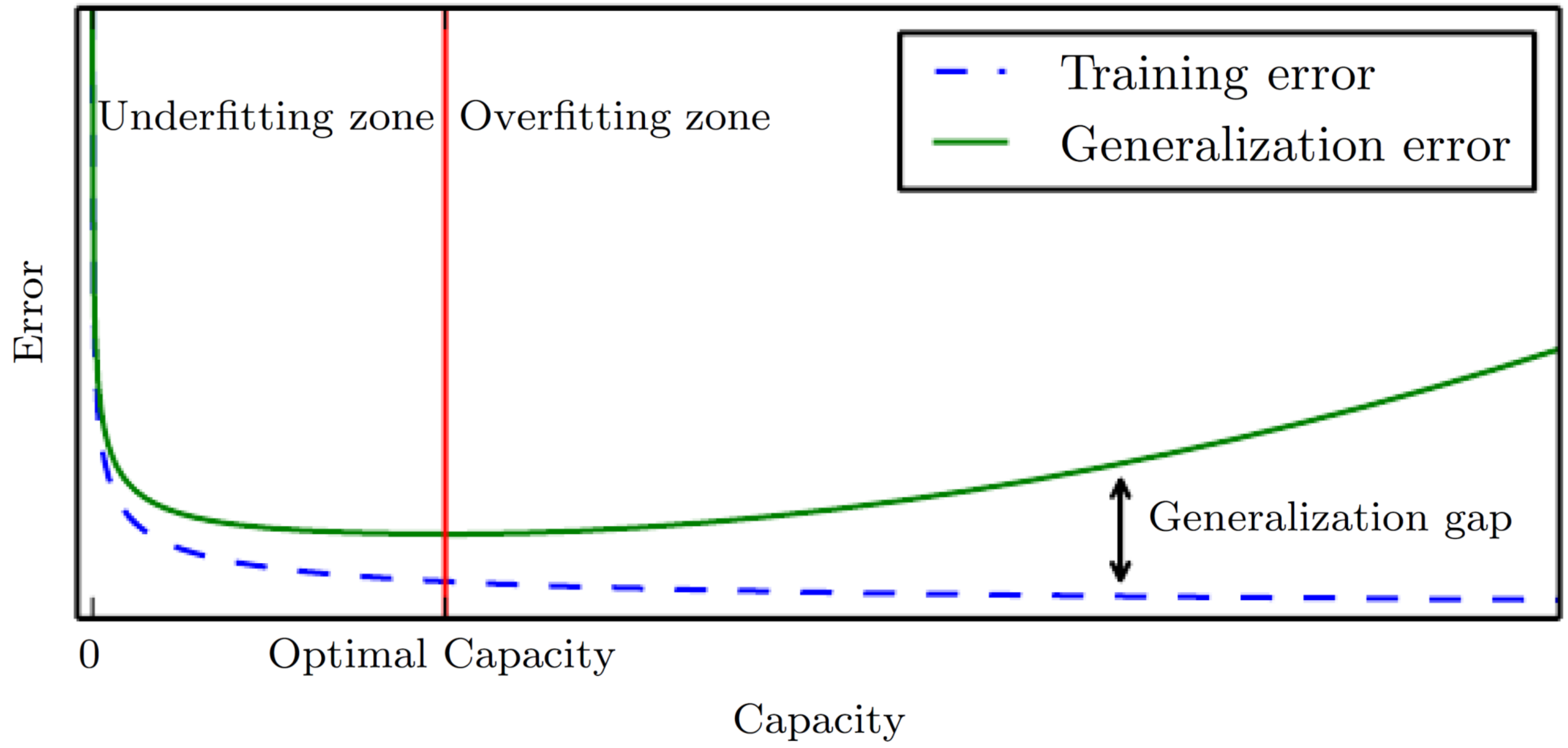
- Two objectives to achieve in designing a model
  1. Make training error small to avoid underfitting
  2. Make gap between training and test error small to avoid overfitting
- Trade-off can be made by altering the **model capacity**, which refers broadly to a model's ability to fit a wide variety of functions
- Example: Fitting a polynomial model to quadratic data



$$\hat{y} = \sum_{i=1}^{d} w_i x_0^i + b$$

Capacity and Error

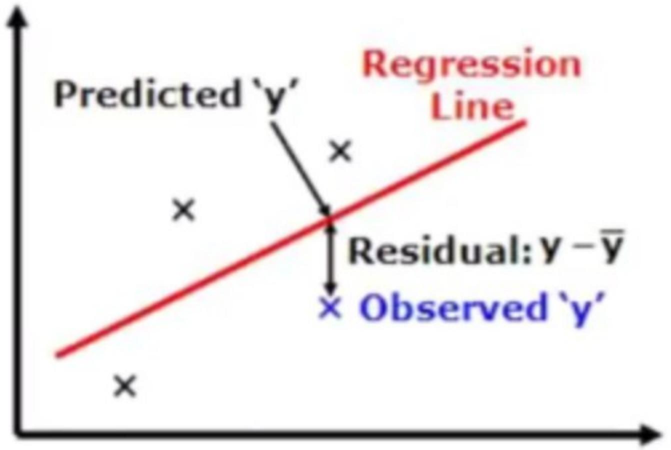
- Typical relation between capacity and error



Cost Function

- Measures the performance of a ML model for given data
- Quantifies the error between predicted and expected values in the **training** set.
- Example

$$Cost\ Function = MSE_{train} = \frac{1}{m}\sum_{i=0}^{m}(\hat{y}_i - y_i)^2$$

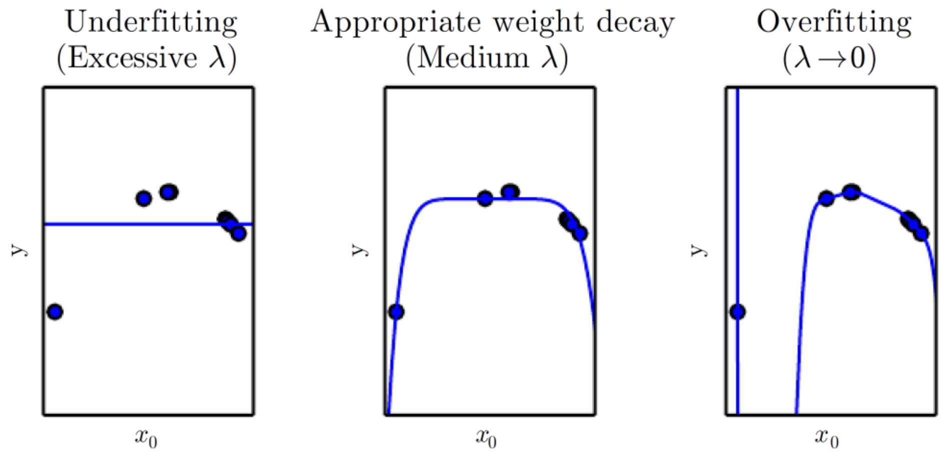


Regularization

- Modification made to the learning algorithm to reduce generalization error (usually at the cost of higher training error)
  - Not reducing the training error
- Example: To include **weight decay** in the training criterion

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

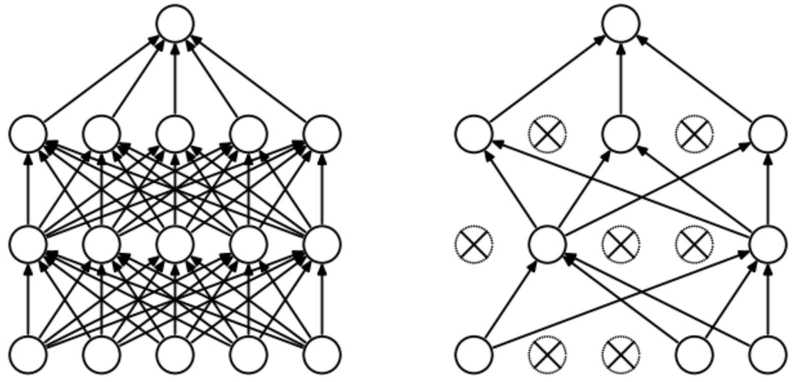
where λ controls preference for small w and is determined a priori

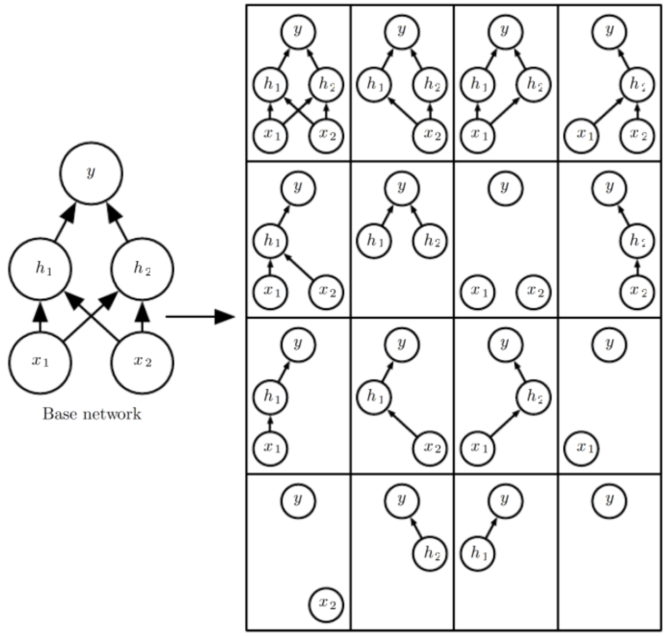


A degree-9 polynomial model fitted to quadratic data

Regularization - Examples

- Weight decay (L2/L1 regularization)
  - Expressing preferences of smaller weights
  - $Cost\ Function = MSE^{(train)} + \lambda \sum_i w_i^2\ \ (L2)$
  - $Cost\ Function = MSE^{(train)} + \lambda \sum_i |w_i|\ (L1)$
- Dropout
  - Temporally remove nodes from network
  - Train a large ensemble of models that share parameters


(a) Standard Neural Net (b) After applying dropout.



Estimators

- Point estimation: To provide a **single best estimate** of some quantity from observing independent and identically distributed (i.i.d.) samples
  - Consider m i.i.d samples $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ drawn from a Bernoulli distribution with mean θ

$$P\left(x^{(i)}; \theta\right) = \theta^{x^{(i)}}(1-\theta)^{1-x^{(i)}}, x^{(i)} = \{1,0\}$$

  - The **sample mean** can be used to give a point estimate of θ

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

Estimators

- A point estimator $\hat{\theta}_m$ of a parameter θ is any function of the observed samples $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$$\hat{\theta}_m = g(x^{(1)}, x^{(2)}, \dots, x^{(m)})$$

  - This definition of a point estimator is very general and allows the designer of an estimator great flexibility
    - Almost any function thus qualifies as an estimator
    - A good estimator is a function whose output is close to the true underlying θ that generated the training data.
  - Let's the the **frequentist** perspective on statistics
    - θ is fixed but unknown
    - $x^{(1)}, x^{(2)}, \dots, x^{(m)}$are seen samples of a random variable
      - Since the data is drawn from a random process, any function of the data is random.
    - As a result, $\hat{\theta}_m$ is a random variable

Estimation

- Function Estimation: Approximating f with a model or estimate $\hat{f}$
  - Assume that there is a function $f(x)$ that describes the approximate relationship between y and x
    - Predicting a variable y given an input vector x.
  - Assuming $y = f(x) + \epsilon$
    - ϵ stands for the part of y that is not predictable from x

- The function estimator $\hat{f}$ is simply a point estimator in function space
  - Example: The linear regression example can be interpreted as
    - Estimating a parameter w (point estimation)
    - Estimating a function $\hat{f}$ mapping from x to y (function estimation)

Bias

- The bias of the estimator $\hat{\theta}_m$ is defined as
$$bias(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta$$
where the expectation E(·) is taken w.r.t. $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$
  - $\hat{\theta}_m$ is **unbiased** if $bias(\hat{\theta}_m) = 0$
  - $\hat{\theta}_m$is **asymptotically unbiased** $if \lim_{m\to\infty} bias(\hat{\theta}_m) = 0$
- In the Bernoulli example, the sample mean is an unbiased estimator

$$E\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] = \frac{1}{m}\sum_{i=1}^{m} E\left[x^{(i)}\right] = \theta$$

Bernoulli Distribution Example

- Consider a set of sample $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ that are i.i.d according to a Bernoulli distribution with mean θ:

$$P\left(x^{(i)};\theta\right) = \theta^{x^{(i)}}(1-\theta)^{1-x^{(i)}}$$

- A common estimator for the θ parameter of this distribution is the mean of the training samples:

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

- In the Bernoulli example, the sample mean is an unbiased estimator

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

$$= E\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] - \theta$$

$$= \frac{1}{m}\sum_{i=1}^{m} E\left[x^{(i)}\right] - \theta$$

$$= \frac{1}{m}\sum_{i=1}^{m}\sum_{x^{(i)}=0}^{1}\left(x^{(i)}\theta^{x^{(i)}}(1-\theta)^{1-x^{(i)}}\right) - \theta$$

$$= \frac{1}{m}\sum_{i=1}^{m}(\theta) - \theta$$

$$= \theta - \theta \quad = 0$$

Consistency

- An estimator $\hat{\theta}_m$ is said to be consistent in probability if

$$\lim_{m\to\infty} P\left(\left|\hat{\theta}_m - \theta\right| > \varepsilon\right) = 0, \varepsilon > 0$$

- In the Bernoulli example, the sample mean is consistent
  - Chebyshev's inequality

$$P(|X - \mu_X| > \varepsilon) \leq \frac{\sigma_X^2}{\varepsilon^2}$$

  - $\hat{\theta}_m$ has mean θ, variance $\theta(1-\theta)/m$

$$P\left(\left|\hat{\theta}_m - \theta\right| > \varepsilon\right) \leq \frac{\theta(1-\theta)}{m\varepsilon^2}$$

$$\lim_{m\to\infty} P\left(\left|\hat{\theta}_m - \theta\right| > \varepsilon\right) = 0$$

Consistency

- Consistency ensures that the bias of the estimator diminishes as the number of data samples grows
  - However, the reverse is not true – asymptotic unbiasedness does not imply consistency
- Example – consider estimating the mean parameter μ of a normal distribution $\mathrm{N}(x; \mu, \sigma^2)$ with a dataset consisting of m samples $\{x^{(1)}, \dots, x^{(m)}\}$
  - Use the first sample $x^{(1)}$ of the dataset as an unbiased estimator: $\hat{\theta} = x^{(1)}$
  - $E\left(\hat{\theta}_m\right) = \mu$ => estimator is unbiased no matter how many data points are seen
    - Imply that the estimate is asymptotically unbiased
  - However, this is not a consistent estimator as it is not the case that $\hat{\theta}_m \rightarrow \mu \; as \; m \rightarrow \; \infty$

Estimators for Gaussian Distribution

- Gaussian probability density function

$$N\left(x^{(i)};\mu,\sigma^2\right)=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\frac{(x^{(i)}-\mu)^2}{\sigma^2}\right)$$

- Common estimator for mean: Sample mean (unbiased)

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

  - Unbiased estimator

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

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

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

$$=\left(\frac{1}{m}\sum_{i=1}^{m}\mu\right)-\mu$$

$$=\mu-\mu=0$$

Estimators for Gaussian Distribution

- First estimator of variance : Sample variance

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

  - Asymptotically unbiased

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

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

$$\lim_{m\to\infty} bias(\hat{\sigma}_m^2) = \lim_{m\to\infty} E[\tilde{\sigma}_m^2] - \sigma^2 = \lim_{m\to\infty} \frac{m-1}{m}\sigma^2 - \sigma^2 = \lim_{m\to\infty} -\frac{\sigma^2}{m} = 0$$

Estimators for Gaussian Distribution

- Alternative estimator of variance : unbiased sample variance estimator

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

  - Unbiased

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

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

$$bias(\tilde{\sigma}_m^2) = E[\tilde{\sigma}_m^2] - \sigma^2 = \sigma^2 - \sigma^2 = 0$$

While unbiased estimators are clearly desirable, they are not always the “best” estimators.
As we will see we often use biased estimators that possess other important properties.

Variance of the Estimator

- **Variance** of the estimator indicates how much the estimator varies as a function of the samples
- The variance of an estimator is simply the variance

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

  - Its squared root is called **standard error**

$$SE(\hat{\theta}) = \sqrt{Var(\hat{\theta})}$$

- Example: Standard error of the sample mean $\hat{\mu}_m$

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

where σ is usually estimated by $\sqrt{\tilde{\sigma}_m^2}$

  - Noted that σ^2 is the true variance of the samples $x^{(i)}$

- However, $Var(\hat{\theta})$ and $SE(\hat{\theta})$ provide an unbiased estimate of the standard deviation
  - Both **underestimate** the true standard deviation
  - For large m the approximation is quite reasonable

Variance of the Estimator

- In ML experiments
  - Generalization error usually estimated by sample mean of the error on test set
  - The number of examples in the test set determines the accuracy of this estimate
- By the central limit theorem,

$$\frac{\hat{\mu}_m - 0}{SE(\hat{\mu}_m)} \sim N(0,1)$$

- The 95 percent confidence interval can thus be derived as

$$(\hat{\mu}_m - 1.96\, SE(\hat{\mu}_m), \hat{\mu}_m + 1.96\, SE(\hat{\mu}_m))$$

Variance of the Estimator

In ML experiments, it is common to say algorithm A performs better than B if its 95 percent upper bound of the test error is smaller than the lower bound of B’s test error

Variance – Bernoulli Distribution Example

- Consider a set of i.i.d samples $\{x^{(1)}, \dots, x^{(m)}\}$ drawn from a Bernoulli distribution

$$P\left(x^{(i)}; \theta\right) = \theta^{x^{(i)}}(1-\theta)^{\left(1-x^{(i)}\right)}$$

- Estimator

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

- Variance

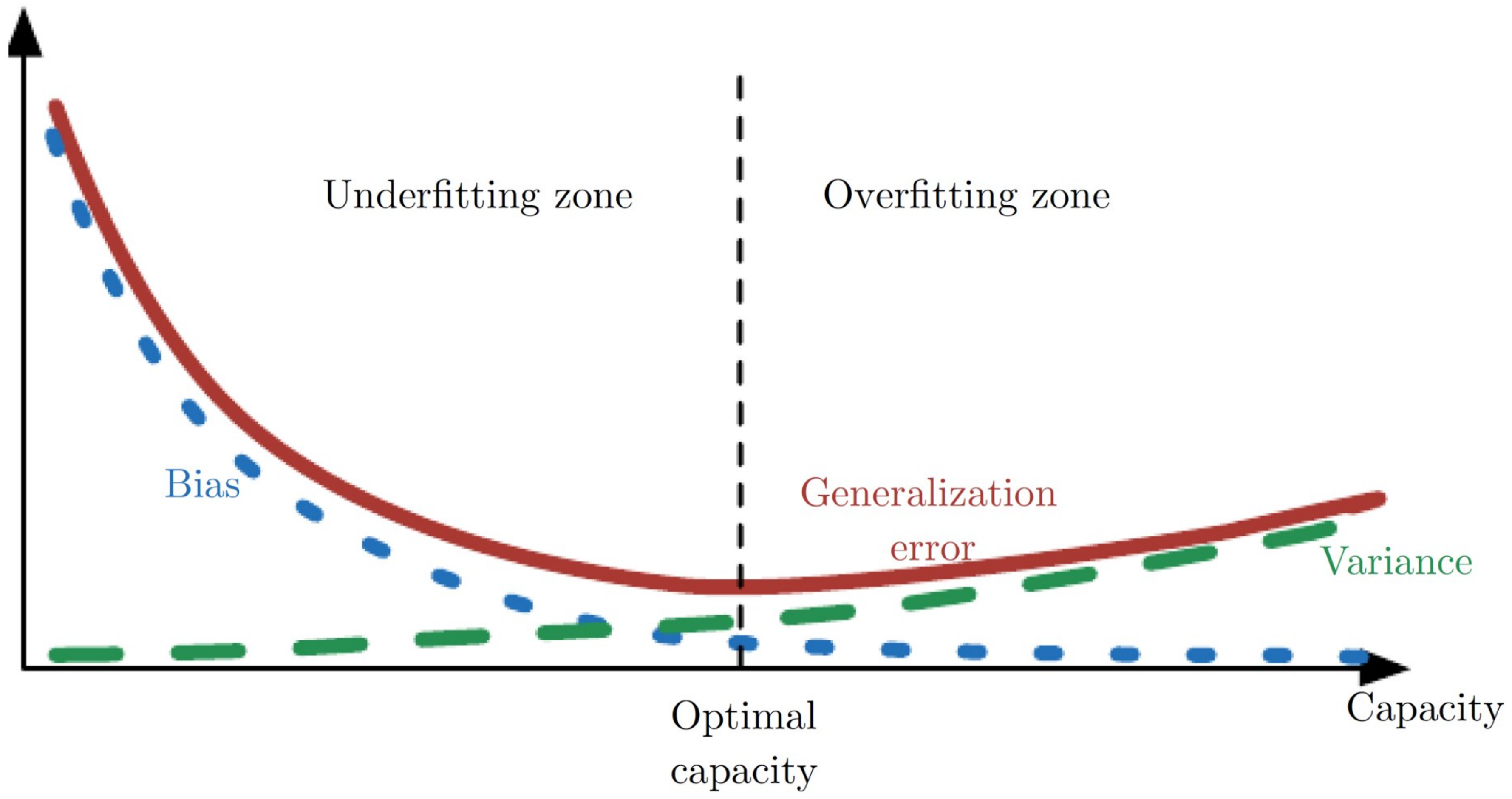
$$Var\left(\hat{\theta}_m\right) = Var\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right) = \frac{1}{m^2}\sum_{i=1}^{m} Var\left(x^{(i)}\right) = \frac{1}{m^2}\sum_{i=1}^{m} \theta(1-\theta)$$

$$= \frac{1}{m^2} m\theta(1-\theta) = \frac{1}{m}\theta(1-\theta)$$

- The variance of the estimator decrease as a function of m

Tradeoff Bias and Variance to Minimize MSE

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



Maximum Likelihood (ML) Estimation

- Consider examples $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ drawn independently from a distribution $p_{model}(x; \theta)$
  - With parameter θ being fixed but unknown
- The maximum likelihood estimator θ_{ML} for θ is defined as

$$\theta_{ML} = arg \max_{\theta} p_{model}(x^{(1)}, x^{(2)}, \dots, x^{(m)};\ \theta)$$

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

$$= arg \max_{\theta} \sum_{i=1}^{m} \log p_{model}(x^{(i)}; \theta)$$

  - Where $\sum_{i=1}^{m} \log p_{model}(x)^{i}; \theta)$ is the log-likelihood function of θ

Maximum Likelihood Estimation

- The sum over examples can be written as expectation w.r.t. the empirical data distribution $\hat{p}_{data}$

$$\theta_{ML} = arg\max_{\theta} E_{x\sim\hat{p}_{data}} \log p_{model}(x^{(i)};\theta)$$

- **Maximizing the log-likelihood** can be viewed as **minimizing the dissimilarity** between the empirical data distribution $\hat{p}_{data}$(defined by the training set) and the model distribution p_{model} in terms of KL divergence:

$$D_{KL}(\hat{p}_{data}||p_{model}) = E_{x\sim\hat{p}_{data}} \underbrace{\boxed{[\log \hat{p}_{data}(x)}}_{\text{Function only of the data generating process}} - \log p_{model}(x)]$$

=>Minimize $D_{KL}(\hat{p}_{data}||p_{model})$ = minimize $-E_{x\sim\hat{p}_{data}}[\log p_{model}(x;\theta)]$

Cross-entropy

- **Cross-entropy** - Any loss consisting of a negative log-likelihood
  - Between the empirical distribution defined by the training set and the probability distribution defined by model.
- E.g., MSE is the cross-entropy between the empirical distribution and a Gaussian model
- Pitfalls
  - Use the term cross-entropy to identify specifically the negative log-likelihood of a Bernoulli or softmax distribution
  - Just a misnomer
- Back to ML, minimize KL divergence corresponds exactly to minimizing the cross-entropy between the distributions
  - Minimize $D_{KL}(\hat{p}_{data}||p_{model})$ = minimize $-E_{x\sim\hat{p}_{data}}[\log p_{model}(x;\theta)]$ = minimize cross-entropy

Remarks for ML

- In information theory, $D_{KL}(\hat{p}_{data}||p_{model})$ denotes the extra amount of information (in bits when using log_2 base) needed to send a message x drawn from $\hat{p}_{data}$ with a code optimized for messages drawn from p_{model}
- The cross-entropy $H(\hat{p}_{data}, p_{model})$ between $\hat{p}_{data}$ and p_{model} is defined as

$$-E_{x\sim\hat{p}_{data}}[\log p_{model}(x;\theta)]$$

- It is easy to show that

$$-E_{x\sim\hat{p}_{data}}[\log p_{model}(x;\theta)] = \underbrace{\boxed{H(\hat{p}_{data})}}_{\text{Entropy of } \hat{p}_{data}} + D_{KL}(\hat{p}_{data}||p_{model})$$

- Minimizing the cross-entropy $H(\hat{p}_{data}, p_{model})$ is equivalent to maximizing the log-likelihood function $E_{x\sim\hat{p}_{data}}[\log p_{model}(x;\theta)]$ and thus minimizing the $D_{KL}(\hat{p}_{data}||p_{model})$

Conditional Log-Likelihood Estimation

- The ML estimator generalized to learn a conditional probability $p_{model}(y|x;\theta)$ in order to predict y given x
- For i.i.d samples $X\text{: }all\ our\ inputs\ , Y\text{: }all\ our\ observed\ targets$, conditional maximum likelihood estimator θ_{ML} is

$$\theta_{ML} = arg\max_{\theta} p_{model}(Y|X;\theta)$$

$$= arg\max_{\theta} \sum_{i=1}^{m} \log p_{model}(y^{(i)}|x^{(i)};\theta)$$

$$= arg\max_{\theta} E_{x,y\sim\hat{p}_{data}(x,y)}[\log p_{model}(\theta)]$$

Conditional Log-Likelihood Estimation Linear Regression Example

- Linear Regression as ML

$$y = \hat{y}(x; w) + \varepsilon = w^T \phi(x) + \varepsilon$$

  - Where $\varepsilon \sim N(0, \sigma^2)$ is independent of x
- Instead of producing a single prediction $\hat{y}$, think of the model as producing a conditional distribution $p(y|x)$
- It can be shown that $p(y|x; w) = N(y; \hat{y}(x; w), \sigma^2)$
  - $\hat{y}(x; w)$ gives the prediction of the mean of the Gaussian
  - Assume variance is fixed to some σ^2

Conditional Log-Likelihood Estimation Linear Regression Example

- The conditional log-likelihood of $p_{model}(Y|X;w)$ is given by

$$\sum_{i=1}^{m} \log p\left(y^{(i)} \middle| x^{(i)}; w\right)$$

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

Normal Distribution

$$N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

- Maximizing the log-likelihood w.r.t. w leads to the same problem of minimizing

$$MSE^{(train)} = \frac{1}{m^{(train)}} \left\| \hat{y}^{(train)} - y^{(train)} \right\|_2^2$$

- Therefore, the data model $y = \hat{y}(x; w) + \varepsilon$ provides an alternative view of the linear regression problem

Bayesian Statistics

- Assume the true parameter θ is a random variable governed by a prior probability distribution $p(\theta)$
  - Instead of estimate a single value of θ and making prediction based on it, consider all possible values θ and making a prediction → **Bayesian Statistics**
  - Reflect degrees of certainty of states of knowledge
  - The dataset is directly observed and so is not random → the true θ is unknown and thus is represented as random variable
- The goal is to infer the posterior distribution $p(\theta|X)$ of θ by combining the prior $p(\theta)$ and the data likelihood $p(\mathrm{X}|\theta)$ via Bayes' rule:

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)}$$

where set of data samples $X = (x^{(1)}, \dots x^{(m)})$

Bayesian Statistics - Prior

- We represent our knowledge of θ using the **prior probability distribution (the prior)** $p(\theta)$
- Generally, the machine learning practitioner selects a prior distribution that is quite broad to reflect a high degree of uncertainty in the value of θ before observing any data
  - E.g., a priori that θ lies in some finite range or volume with a uniform distribution
- Many priors instead reflect a preference for "simpler" solutions
  - E.g., smaller magnitude coefficients, or a function that is closer to being constant

Bayesian Statistics - Properties

- Bayesian estimation offers two important differences from ML
  1. Make predictions using a full distribution over θ instead of a point estimate of θ
     - After observing m samples, the predicted distribution over the next data sample $x^{(m+1)}$ is given by

$$p\left(x^{(m+1)} \middle| x^{(1)}, \dots, x^{(m)}\right) = \int p\left(x^{(m+1)} | \theta\right) p\left(\theta \middle| x^{(1)}, \dots, x^{(m)}\right) d\theta$$

     - if we are still quite uncertain about the value of θ, then this uncertainty is incorporated directly into any predictions we might make
  2. The prior has an influence by shifting probability mass density towards regions of the parameter space that are preferred a priori
     - Due to the contribution of the Bayesian prior distribution
     - The prior often expresses a preference for models that are simpler or more smooth
     - The prior might be a source of subjective human judgment impacting the predictions
- Bayesian methods typically generalize much better when limited training data is available, but typically suffer from high computational cost when the number of training examples is large.

Bayesian Statistics – Linear Regression Example

- Linear Regression

$$y = \hat{y}(x; w) + \varepsilon = w^T \phi(x) + \varepsilon$$

- Again, $p(y|x, w) = N(y; \hat{y}(x; w), \sigma^2)$

$$\propto \exp\left(-\frac{1}{2}\left(y - \hat{y}(x; w)\right)^T \left(y - \hat{y}(x; w)\right)\right)$$

  - assuming Gaussian variance on y is one($\sigma^2 = 1$)

- Assume $p(w) = N(w; \mu_o, \Lambda_0)$

$$\propto \exp\left(-\frac{1}{2}(w - \mu_0)^T \Lambda_0^{-1} (w - \mu_0)\right)$$

  - μ_0: prior distribution mean vector
  - Λ_0: prior distribution covariance matrix

Bayesian Statistics – Linear Regression Example

- Consider x to be deterministic data
- The posterior distribution $p(w|X^{(train)}, y^{(train)})$ is given by

$$
\begin{aligned}
& p\left(w \middle| X^{(train)}, y^{(train)}\right) \\
& \propto p(w)p\left(y^{(train)} \middle| X^{(train)}, w\right) \\
& \propto \exp\left(-\frac{1}{2}(w-\mu_0)^T\Lambda_0^{-1}(w-\mu_0)\right)\exp\left(-\frac{1}{2}(y-\Phi w)^T(y-\Phi w)\right) \\
& \propto \exp\left(-\frac{1}{2}(-2y^TXw + w^TX^TXw + w^T\Lambda_0^{-1}w - 2\mu_0^T\Lambda_0^{-1}w)\right) \\
& \propto \exp\left(-\frac{1}{2}(w-\mu_m)^T\Lambda_m^{-1}(w-\mu_m) + \frac{1}{2}\mu_m^T\Lambda_m^{-1}\mu_0\right) \\
& \propto \exp\left(-\frac{1}{2}(w-\mu_m)^T\Lambda_m^{-1}(w-\mu_m)\right) \\
& = N(w;\mu_m,\Lambda_m)
\end{aligned}
$$

- When $\mu_0 = 0$ and $\Lambda_0 = \frac{1}{\lambda}I$, μ_m leads to the same solution as minimizing

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

Where

$$
\begin{aligned}
&\Phi = \Phi^{(train)}, \qquad y = y^{(train)} \\
&\Lambda_m = (\Phi^T\Phi + \Lambda_0^{-1})^{-1}, \\
&\mu_m = \Lambda_m\left(\Phi^T y^{(train)} + \Lambda_0^{-1}\mu_0\right) \\
&\Phi = \begin{bmatrix} \phi\left(x_0^{(train)}\right)^T \\ \vdots \\ \phi\left(x_{m-1}^{(train)}\right)^T \end{bmatrix}, y^{(train)} = \begin{bmatrix} y_0^{(train)} \\ \vdots \\ y_{m-1}^{(train)} \end{bmatrix}
\end{aligned}
$$

Bayesian Statistics

- Given $p(w|X^{(train)}, y^{(train)})$, one can infer the probability distribution of $y^{(new)}$ at unseen x^{new} by

$$
\begin{aligned}
&p\left(y^{(new)}\middle|x^{(new)}, X^{(train)}, y^{(train)}\right) \\
&= \int p\left(y^{(new)}, w|x^{(new)}, X^{(train)}, y^{(train)}\right)dw \\
&= \int p\left(w|X^{(train)}, y^{(train)}\right)p\left(y^{(new)}\middle|x^{(new)}, w\right)dw \\
&= \int N(w; \mu_m, \Lambda_m)N\left(y^{(new)}; \hat{y}\left(x^{(new)}; w\right), 1\right)dw
\end{aligned}
$$

Bayesian Statistics

- Equivalently, this is to ask about the distribution of

$$y^{(new)} = \hat{y}\left(x^{(new)}; w\right) + \varepsilon = w^T \phi\left(x^{(new)}\right) + \varepsilon$$

  - Given $\left(X^{(train)}, Y^{(train)}\right)$ with

$$p\left(w \middle| X^{(train)}, Y^{(train)}\right) = N(w; \mu_m, \Lambda_m)$$
$$p\left(\varepsilon \middle| X^{(train)}, Y^{(train)}\right) = N(\varepsilon; 0,1)$$

  - And w, ε being conditionally independent

$$p\left(w, \varepsilon \middle| X^{(train)}, Y^{(train)}\right) = p\left(w \middle| X^{(train)}, Y^{(train)}\right) p(\varepsilon | X^{(train)}, Y^{(train)})$$

Bayesian Statistics

- We further recognize that
  - $w^T\phi\left(x^{(new)}\right)|X^{(train)}, Y^{(train)}$ is a Gaussian

$$w^T\phi\left(x^{(new)}\right)|X^{(train)}, Y^{(train)}$$

$$\sim N\left(\mu_m^T\phi\left(x^{(new)}\right), \phi\left(x^{(new)}\right)^T\Lambda_m\phi\left(x^{(new)}\right)\right)$$

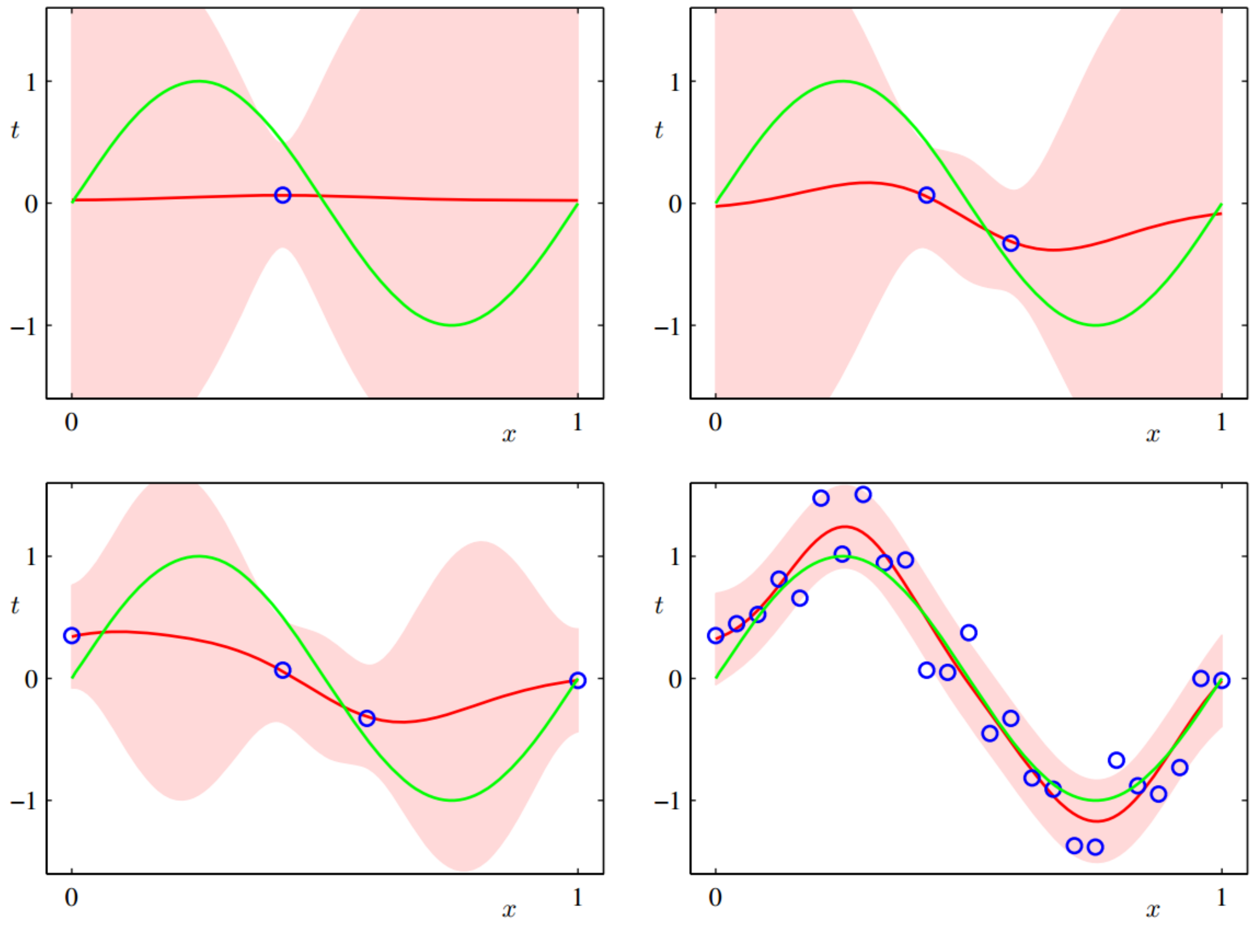
  - $w^T\phi\left(x^{(new)}\right) + \varepsilon|X^{(train)}, Y^{(train)}$ (sum of two conditionally independent Gaussian's) is another Gaussian

$$w^T\phi\left(x^{(new)}\right) + \varepsilon|X^{(train)}, Y^{(train)}$$

$$\sim N(\mu_m^T\phi\left(x^{(new)}\right), \phi\left(x^{(new)}\right)^T\Lambda_m\phi\left(x^{(new)}\right) + 1)$$

- This concludes our evaluation for $p\left(y^{(new)}\middle|x^{(new)}, X^{(train)}, y^{(train)}\right)$

Bayesian Statistics



Ground truth : Green

$\mu_{y(new)}$: Red

Data : Blue

$\sigma_{y(new)}$: Pink

Maximum A Posteriori(MAP) Estimation

- The full Bayesian treatment may sometimes be intractable
  - A point estimate offers a tractable approximation.
- To offer a point estimate in the Bayesian framework, we usually choose

$$
\begin{aligned}
\theta_{MAP} &= arg \max_{\theta} p(\theta|X) \\
&= arg \max_{\theta} \frac{p(X|\theta)p(\theta)}{p(\mathrm{X})} \\
&= arg \max_{\theta} \log p(X|\theta) + \log p(\theta)
\end{aligned}
$$

MAP – Regularization

- Many regularized estimation strategies (e.g. ML+ weight decay) can be interpreted as making the MAP inference, if the regularization term (e.g. weight decay) admits an explanation of $\log p(\theta)$

- Example – weight decay regularization
  - Consider a linear regression model with a Gaussian prior on the weights w
  - The prior is given by $N\left(w; 0, \frac{1}{\lambda} I^2\right)$

$$\log p(\theta) \propto \lambda w^T w$$

=> Proportional to the familiar weight decay penalty

  - MAP Bayesian inference with a Gaussian prior on the weights thus corresponds to weight decay

Maximum A Posteriori(MAP) Estimation

- Consider Linear Regression with point estimate in Bayesian framework

$$\begin{aligned} w_{MAP} &= arg\max_w p(w|X^{(train)}, y^{(train)}) \\ &= arg\max_w N(w;\mu_m, \Lambda_m) \\ &= \mu_m \end{aligned}$$

- We may then choose the prediction model to be

$$\hat{y}\left(x^{(new)}; w\right) = w_{MAP}^T \phi\left(x^{(new)}\right) = \mu_m^T \phi(x^{(new)})$$

  - Which in the present case coincides with the mean of the posterior distribution

$$p\left(y^{(new)} \middle| x^{(new)}, X^{(train)}, y^{(train)}\right) \sim N\left(\mu_m^T \phi\left(x^{(new)}\right), \phi\left(x^{(new)}\right)^T \Lambda_m \phi\left(x^{(new)}\right)\right)$$

Maximum A Posteriori(MAP) Estimation

- From the earlier derivation and definition,

$$\mu_m = \Lambda_m\left(\Phi^T y^{(train)} + \Lambda_0^{-1}\mu_0\right)$$

$$\Phi = \begin{bmatrix} \phi\left(x_0^{(train)}\right)^T \\ \phi\left(x_1^{(train)}\right)^T \\ \vdots \\ \phi\left(x_{m-1}^{(train)}\right)^T \end{bmatrix}, y^{(train)} = \begin{bmatrix} y_0^{(train)} \\ y_1^{(train)} \\ \vdots \\ y_{m-1}^{(train)} \end{bmatrix}$$

Maximum A Posteriori(MAP) Estimation

- Assuming $\mu_0 = 0$, we have

$$\hat{y}\left(x^{(new)}; w\right) = \phi\left(x^{(new)}\right)^T \Lambda_m \Phi^T y^{(train)}$$

$$= \sum_{i=0}^{m-1} \boxed{\phi\left(x^{(new)}\right)^T \Lambda_m \phi\left(x_i^{(train)}\right)} y_i^{(train)}$$

$$= \sum_{i=0}^{m-1} \underbrace{\boxed{k\left(x^{(new)}, x_i^{(train)}\right)}}_{\text{Kernel}} y_i^{train}$$

  - Where

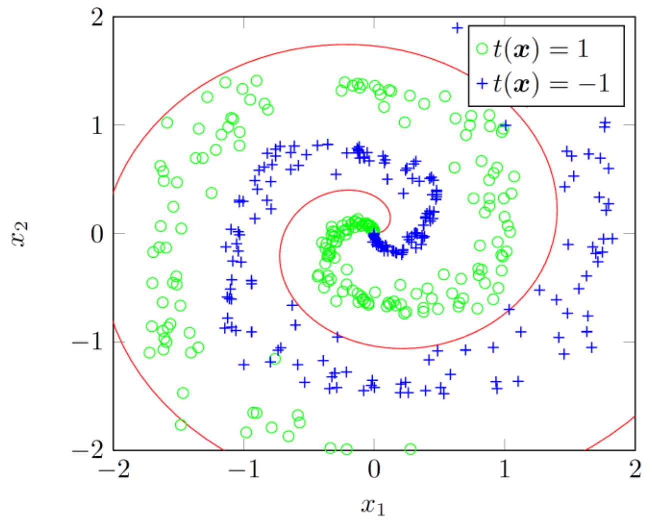
$$k\left(x^{(new)}, x_i^{(train)}\right) = \phi\left(x^{(new)}\right)^T \Lambda_m \phi\left(x_i^{(train)}\right)$$

- It is seen that the prediction $\hat{y}(x^{(new)}; w)$ is a weighted combination of the training data $y_i^{(train)}$ with weights determined by the kernel function $k(\cdot)$ measuring a certain type of distance between $x^{(new)}$ and $x^{(train)}$

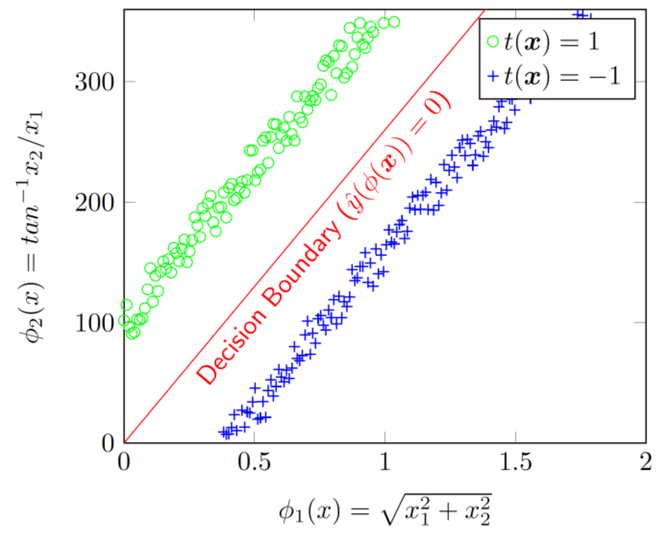
Support Vector Machines (SVM)

- One of the most influential approaches to supervised learning
- Idea – To find a hyperplane (in feature space) for classifying merely separable training data according to the sign of $\hat{y}(\phi(x))$

$$\hat{y}(\phi(x)) = w^T\phi(x) + b$$



Raw data domain



Feature domain

Support Vector Machines (SVM)

- The hyperplane, known as the decision boundary, is defined by
$$\hat{y}\big(\phi(x)\big) = w^T\phi(x) + b = 0$$
- w is a vector orthogonal to every vector in the decision boundary
- Any point $\phi(x)$ to the decision boundary has a distance
$$\frac{|\hat{y}(\phi(x))|}{||w||}$$
- **Maximum margin classifiers**: Maximizing the smallest distance between the decision boundary and any of the training samples $\phi(x_n)$
$$arg\max_{w,b}\min_{n}\frac{t_n\hat{y}(\phi(x_n))}{||w||} = arg\max_{w,b}\left\{\frac{1}{||w||}\min_{n}[t_n(w^T\phi(x_n)+b)]\right\}$$
  - Where $t_n \in \{-1,1\}$ is the ground-truth label associated with $\phi(x_n)$

Support Vector Machines (SVM)

- Noting that w, b can be scaled simultaneously $(w \to kw, b \to kb)$ without changing the resulting distance, we can choose a k such that $t_n(w^T\phi(x_n) + b) = 1$ for the closest point $\phi(x)$ to the boundary
- This allow us to reformulate the problem as a **constrained optimization problem**

$$\underset{w,b}{argmin} \frac{1}{2}\|w\|_2^2 \text{ , subject to } t_n(w^T\phi(x_n) + b) \geq 1, \forall n$$

- Using the method of Lagrange multipliers, which will be introduced next, the solution for $\hat{y}(\phi(x))$ can be solved as

$$\hat{y}(\phi(x)) = \sum_{n=1}^{N} a_n t_n \boxed{\phi(x_n)^T\phi(x)} + b = \sum_{n=1}^{N} a_n t_n \boxed{k(x_n, x)} + b$$

Kernel

  - Where $a_n \geq 0, \forall n$ and the most of them are zero
- **Support vectors** refer to those $\phi(x_n)$'s whose $a_n \neq 0$

Constrained Optimization

- To find the maximal/minimal value of $f(x)$ for x (known as feasible points) in some set $\mathbb{S}$, e.g.

$$\arg\min_{x} f(x), \text{ subject to}$$
$$g^{(i)}(x) = 0, i = 1, \dots, m$$
$$h^{(j)}(x) \leq 0, j = 1, \dots, n$$

  - Where $\mathbb{S} = \{x | \forall i, g^{(i)}(x) = 0, \forall j, h^{(j)}(x) \leq 0\}$ are defined by m **equality constraints** and n **inequality constraints**

Constrained Optimization

- The **Karush-Kuhu-Tucker (KKT)** approach obtains a solution by solving the unconstrained optimization of the Lagrangian function

$$\min_{x} \max_{\lambda} \max_{\alpha, \alpha \geq 0} \mathrm{L}(x, \lambda, \alpha)$$

  - Where

$$L(x, \lambda, \alpha) = f(x) + \sum_{i=0}^{m} \lambda_i g^{(i)}(x) + \sum_{j=0}^{n} \alpha_j h^{(j)}(x)$$

  - And λ and α are termed Lagrange multiplier

Constrained Optimization

- The optimal solution satisfies (necessary conditions)
  1. $\nabla L(x, \lambda, \alpha) = 0$
  2. All constraint on x and Lagrange multiplier are met
  3. Complementary slackness: $\alpha \odot h(x) = 0$, i.e. $\alpha_j = 0$ for $h^{(i)}(x) < 0$ (inactive) and $\alpha_j \geq 0$ for $h^{(i)}(x) = 0$ (active)
- It is easy to see that any constraint is violated,

$$\max_{\lambda} \max_{\alpha, \alpha \geq 0} L(x, \lambda, \alpha) = \infty$$

  - Which excludes infeasible point from being considered
- On the other hand, when the constraint are all satisfied,

$$\max_{\lambda} \max_{\alpha, \alpha \geq 0} L(x, \lambda, \alpha) = f(x)$$

  - Which ensures the optimum within feasible points in unchanged

Constrained Optimization

- Example: to solve w, b

$$\min \frac{1}{2}\|w\|_2^2, \text{ subject to } t_n(w^T\phi(x_n) + b) \geq 1, \forall n$$

- We set to zero the gradient w.r.t

$$L(w, b, \alpha) = \frac{1}{2}\|w\|_2^2 + \sum_{j=1}^{n} \alpha_j(1 - t_n(w^T\phi(x_n) + b))$$

  - And arrive at

$$\nabla_w L(w, b, \alpha) = 0 \Rightarrow w = \sum_{j=1}^{n} \alpha_j t_n \phi(x_n)$$

$$\nabla_b L(w, b, \alpha) = 0 \Rightarrow \sum_{j=1}^{n} \alpha_j t_n = 0$$

- At this point, we already have the form of the optimal w

Constrained Optimization

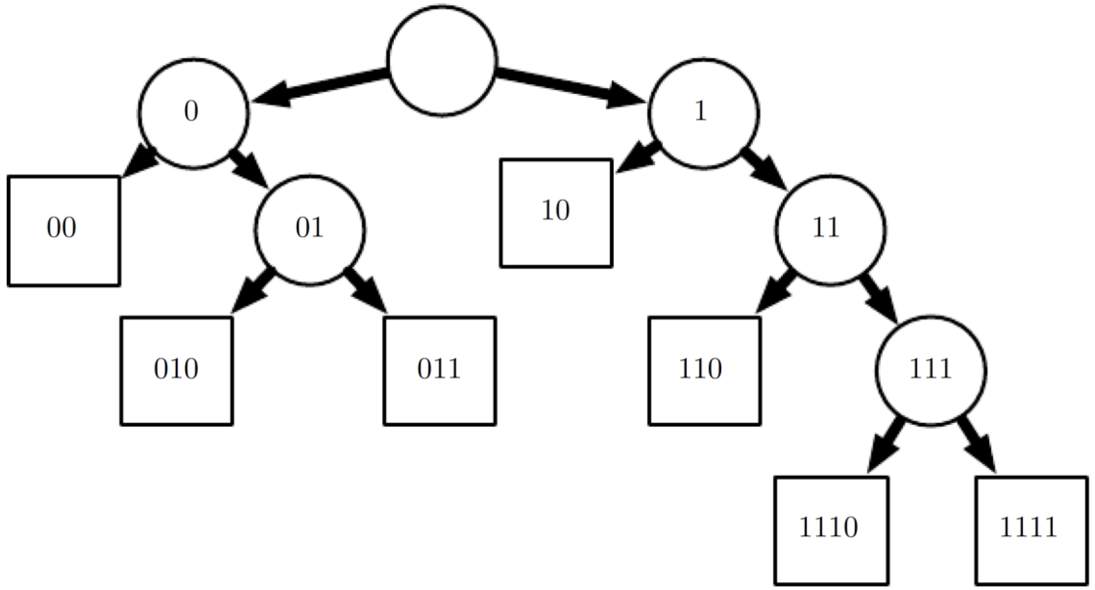
- To solve for α, we can substitute this w back to the Lagrangian

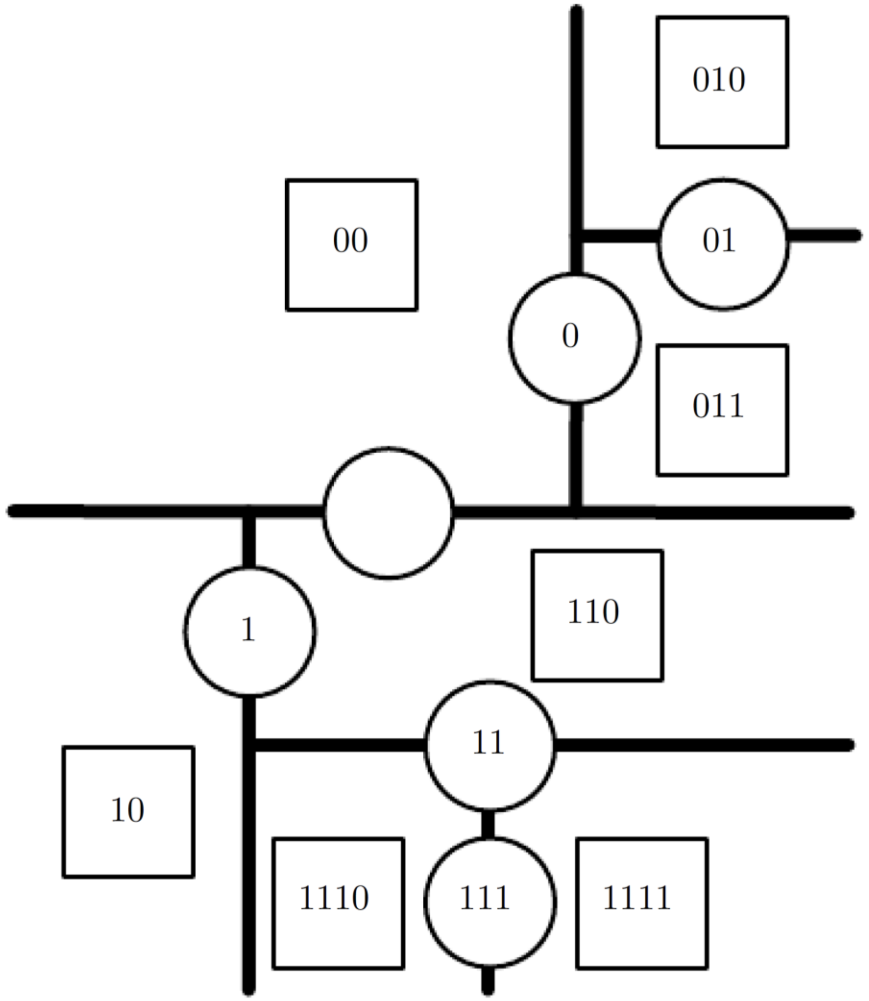
$$\max_{\alpha} \sum_{j=1}^{n} \alpha_j - \frac{1}{2} \sum_{p=1}^{m} \sum_{q=1}^{m} \alpha_p \alpha_q t_p t_q \underbrace{\boxed{\phi(x_p)^T \phi(x_q)}}_{\text{Kernel function}},$$

  - Subject to $\alpha_i \geq 0, \forall i$
    and $\sum_{j=1}^{n} \alpha_j t_j = 0$
  - Kernel function: $k(x, x') = \phi(x)^T \phi(x')$

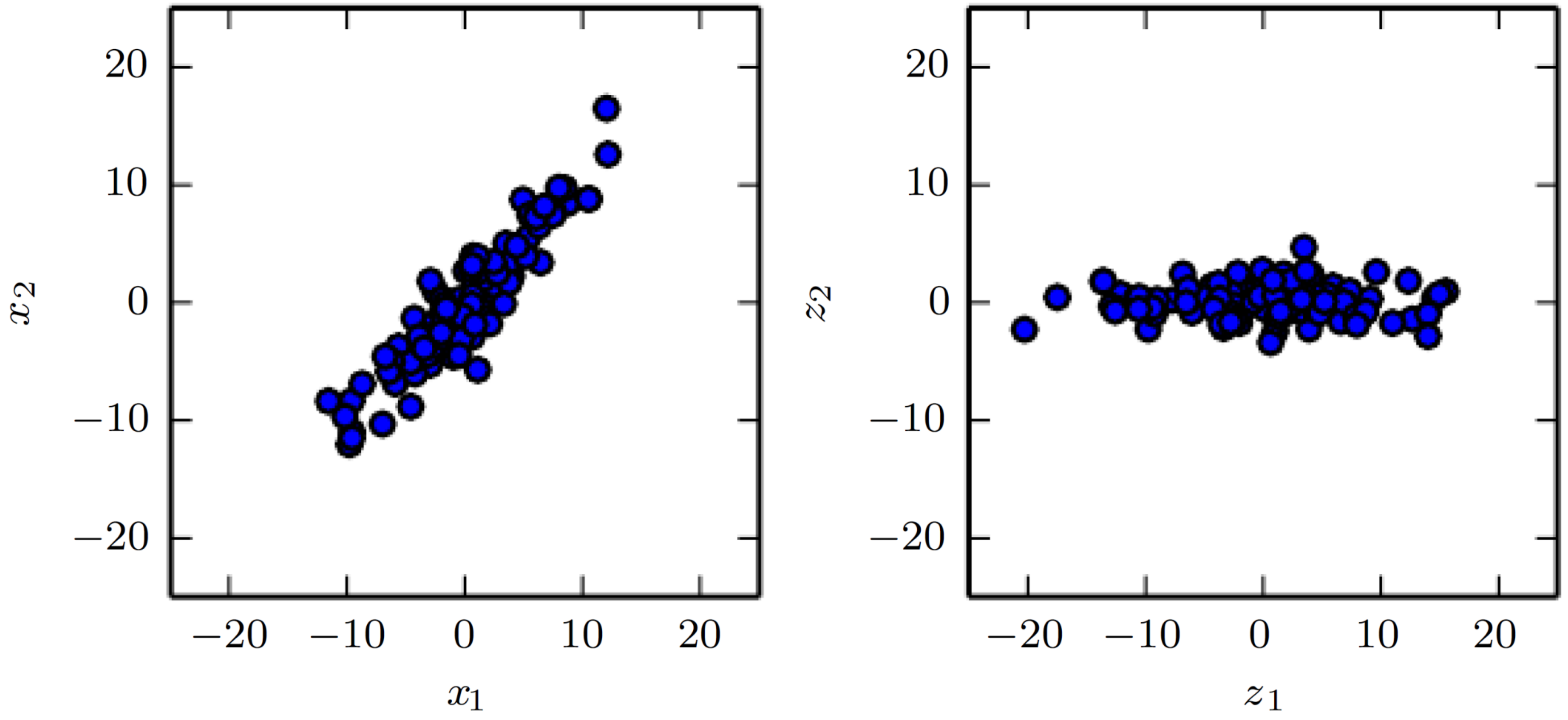
- How to sole b?

Decision Trees





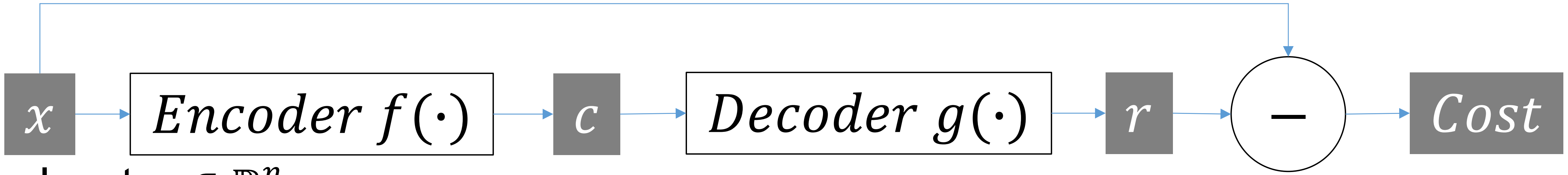
Principle Component Analysis (PCA)



PCA learns a linear projection that aligns the direction of greatest variance with the axes of the new space.

Principle Component Analysis (PCA)

- An unsupervised learning algorithm that learns a data representation



  - Input: $x \in \mathbb{R}^n$
  - Representation of input: $c \in \mathbb{R}^l$
  - Decoder: $g(c) = Dc$ with $D \in \mathbb{R}^{n \times l}$ composed of orthonormal columns
  - Cost: $\|x - g(c)\|_2^2$
  - Encoder has the form $f(x) = D^T x$ when Cost is minimized
- Objective: given training data $X^{(train)}$, we wish to find D

$$arg \min_D \left\| X^{(train)} - X^{(train)} D D^T \right\|_F^2, s.t. D^T D = I$$

$\|\cdot\|_F$: Frobenius norm

Principle Component Analysis (PCA)

- Recall that

$$X^{(train)} = \begin{bmatrix} x_0^{(train)^T} \\ x_1^{(train)^T} \\ \vdots \\ x_{m-1}^{(train)^T} \end{bmatrix}, \|A\|_F^2 = Tr\{AA^T\} = \sum_i A_{i,i}^2$$

- By simple algebra, we have

$$\left\|X^{(train)} - X^{(train)}DD^T\right\|_F^2$$
$$= Tr\left(X^{(train)}X^{(train)T}\right) - Tr(X^{(train)}DD^T X^{(train)T})$$

  - Where the first term has nothing to do with D

Principle Component Analysis (PCA)

- The initial problem them simplified to

$$arg \max_{D} Tr\left(X^{(train)} D D^T X^{(train)T}\right), s.t. D^T D = I$$

- Observing that the Trace operator has the property

$$Tr(ABC) = Tr(BCA) = Tr(CAB),$$

as long as all matrix multiplications are allowed

- We arrive at

$$arg \max_{D} Tr\left(D^T X^{(train)^T} X^{(train)} \mathrm{D}\right), s.t. D^T D = I$$

Principle Component Analysis (PCA)

- By a further application of Singular Value Decomposition(SVD) to

$$X_{m\times n}^{(train)} = U_{m\times \mathrm{m}}\Sigma_{m\times n}\mathrm{V}_{n\times n}^{T}$$

  - U is the eigenvector matrix of $X_{m\times n}^{(train)}X_{m\times n}^{(train)^{T}}$ and satisfies
$$UU^{T} = U^{T}U = I_{m\times m}$$
  - V is the eigenvector matrix of $X_{m\times n}^{(train)^{T}}X_{m\times n}^{(train)}$ and satisfies
$$VV^{T} = V^{T}V = I_{n\times n}$$
  - Σ is the singular matrix given by
$$\left[\begin{array}{cccc:c} \sigma_1 & 0 & \cdots & 0 & \\ 0 & \sigma_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & \\ 0 & 0 & 0 & \sigma_r & \\ \hdashline & & 0 & & 0 \end{array}\right]_{m\times n}$$

Principle Component Analysis (PCA)

- We see that the column of D has an obvious choice of the l columns in V which corresponds to the largest l singular value

$$arg \max_D Tr(D^T V \Sigma^T \Sigma V^T D), s.t. D^T D = I$$

  - Where

$$\Sigma^T \Sigma = \left[\begin{array}{cccc:c} \sigma_1^2 & 0 & \cdots & 0 & \\ 0 & \sigma_2^2 & \cdots & 0 & \\ \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & 0 & \sigma_r^2 & \\ \hdashline \multicolumn{4}{c:}{0} & 0 \end{array}\right]_{n \times n}$$

Optimization

- Learning algorithms often seek to minimize/maximize some objective function w.r.t. the model parameter, e.g.

$$arg\min_{w} J(w) = -E_{x,y\sim\hat{p}_{data}}[p_{model}(y|x)]$$

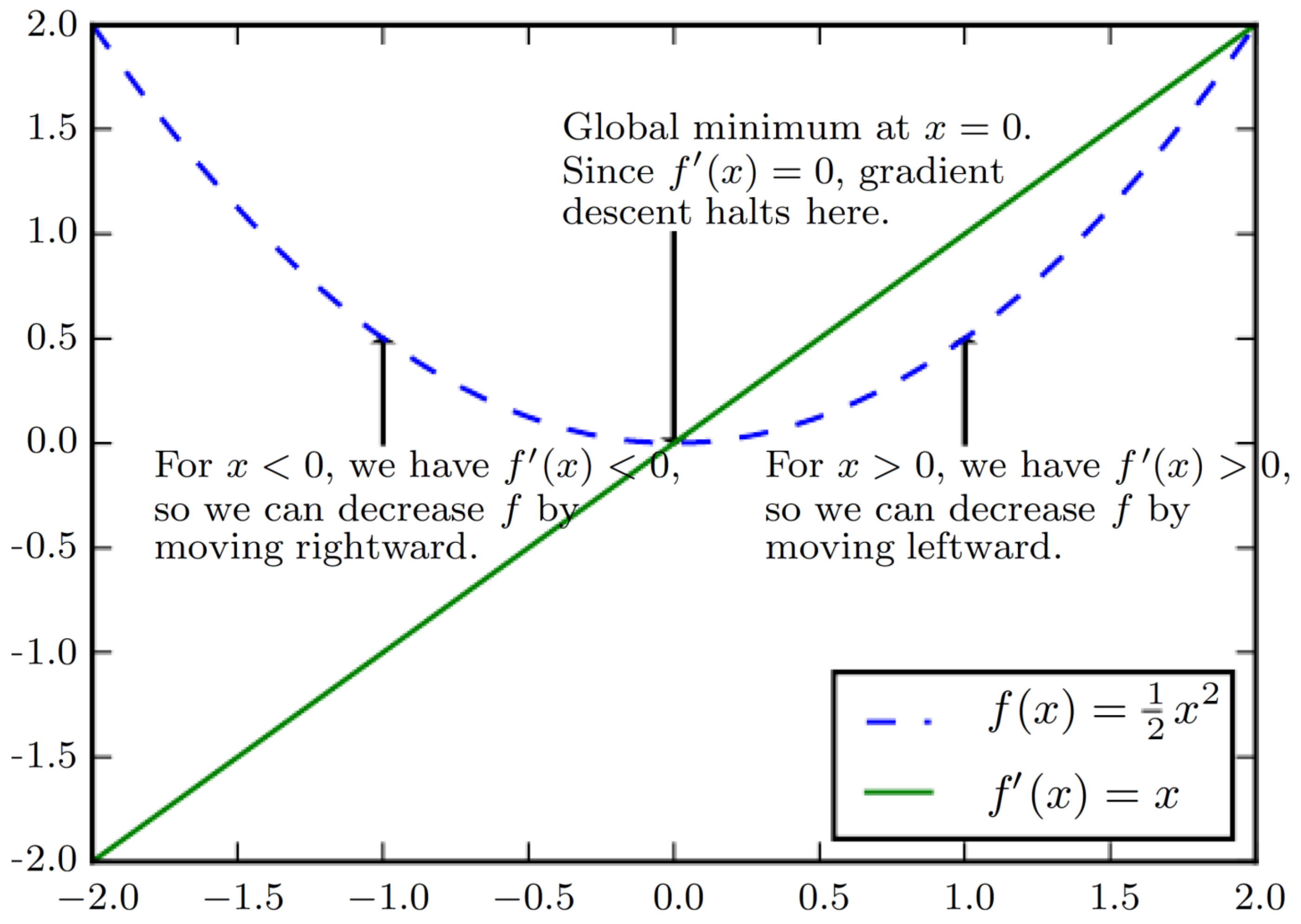
- Very often, there is no closed-form solution
- Gradient-based learning algorithms are thus called for to update estimates of the solution via an iterative procedure

Gradient-based Optimization

- Nonlinearity of NN causes non-convex loss functions
- Method to minimize the **cost function** by updating weights
- **No convergence guarantee**
- Sensitive to initial values of parameters
  - Weight - small random values
  - Bias - zero of small positive values
- Gradient descent:
  - Iteratively moving in the direction of steepest descent as defined by the negative of the gradient
- Stochastic gradient descent (SGD)
  - To handle large training sets
  - Only run a subset of the training sets (i.e., batch/minibatch) for each update
  - Easier to converge

Gradient Descent

- Follow the slope



Gradient Descent (Steepest Descent)

- For functions with multiple inputs, we must make use of the concept of partial derivatives
- The partial derivative $\frac{\partial}{\partial x_i} f(x)$
  - Measures how f changes as only the variable x_i increases at point x
- The **gradient** generalizes the notion of derivative to the case where the derivative is with respect to a vector
  - $\nabla_x f(x)$: The gradient of f is the vector containing all of the partial derivatives
  - Element i of the gradient is the partial derivative of f with respect to x_i
  - Critical points : the points where every element of the gradient is equal to zero

Gradient Descent (Steepest Descent)

- To decrease $J(w)$ in the direction in which it decreases the fastest

$$w^{(n+1)} = w^{(n)} - \epsilon \nabla_w J\left(w^{(n)}\right)$$

  where ϵ controls the step size for each update

- The negative gradient$-\nabla_w J\left(w^{(n)}\right)$points to the direction in which $J(w)$ decreases the fastest at $w^{(n)}$
  - To see this, we define the directional directive in a unit vector u at w_0, with respect to α

$$\frac{\partial}{\partial \alpha} J(w_o + \alpha u)$$

  - Using the chain rule, it can then be evaluated as

$$\frac{\partial}{\partial \alpha} J(w_0 + \alpha u) = u^T \nabla_w J(w_0), \qquad when\ \alpha = 0.$$

Gradient Descent (Steepest Descent)

- To minimize J

$$\min_{u, u^T u=1} u^T \nabla_x J(w_0) = \min_{u, u^T u=1} \|u\|_2 \|\nabla_x J(w_0)\|_2 \cos\theta \;, where\ \|u\|_2 = 1$$

$$= \min_{u} \cos\theta$$

,where θ is the angle between u and the gradient

  - This is minimized when u points in the opposite direction as the gradient
  - In other words, the gradient points directly uphill, and the negative gradient points directly downhill

- From Taylor expansion viewpoint
  - Using Taylor-1 approximation at w_0

$$J(w) \approx J(w_0) + (w - w_0)^T \nabla_w J(w_0)$$

  - The unit vector u that points in the direction $-\nabla_w J(w_0)$ yields a minimal directive among other unit vectors

Gradient Descent (Steepest Descent)

- Instead of using a fixed ϵ, we can use line search to adapt its value to the curvature of $J(w)$ along $-\nabla_w J(w_o)$
- This relies on approximating $J(w)$ at w_0 with Taylor-2 approximation

$$J(w) \approx J(w_0) + (w - w_0)^T \nabla_w J(w_0) + \frac{1}{2}(w - w_0)^T H (w - w_0)$$

  - Where H is the Hessian matrix defined as

$$\mathrm{H}\big(J(w_0)\big)_{i,j} = \frac{\partial^2}{\partial w_i \partial w_j} J(w_0)$$

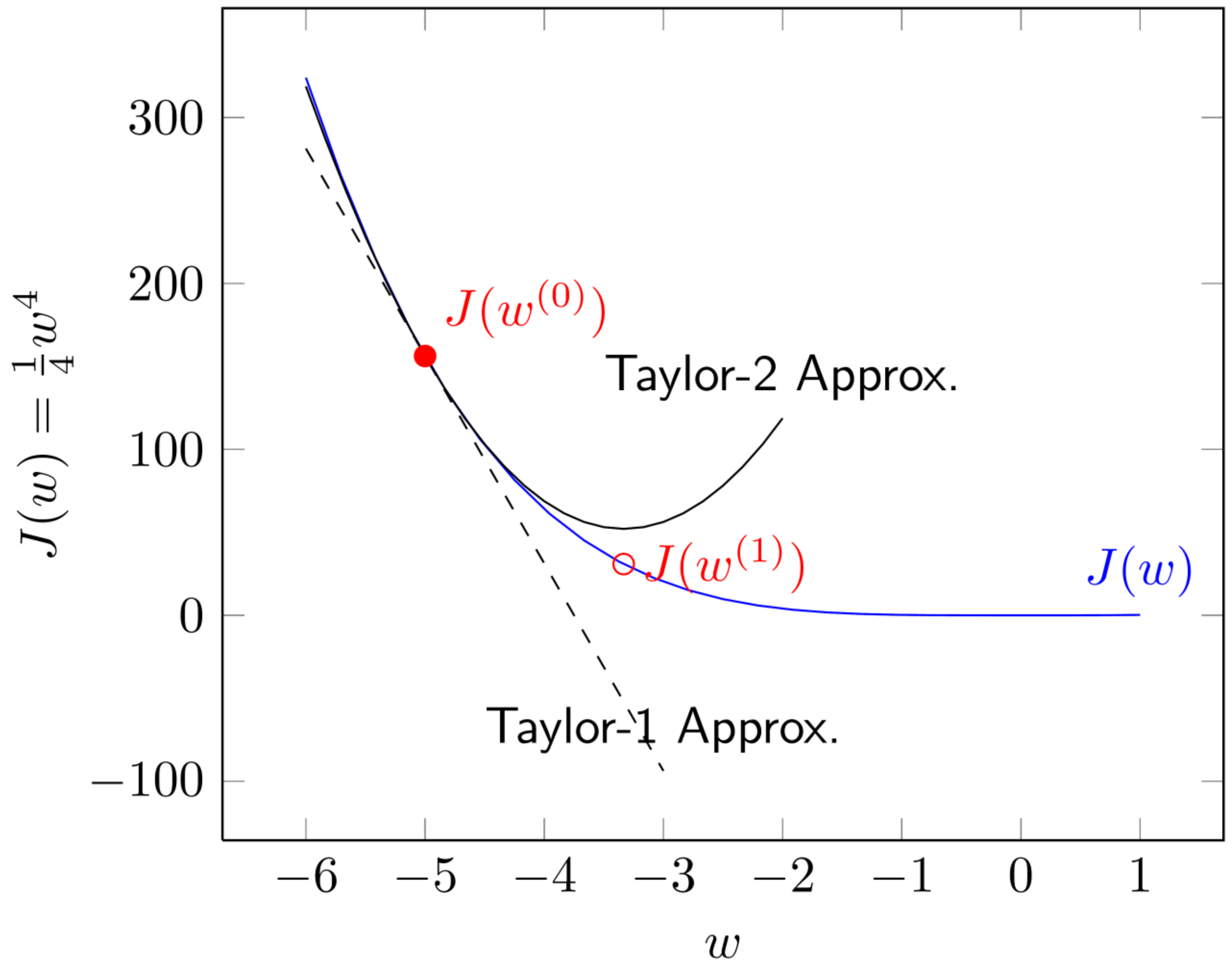
- We wish to fine a ϵ that maximize

$$J(w_0) - J\big(w_0 - \epsilon \nabla_w J(w_0)\big)$$

- The optimal ϵ is given by

$$\epsilon^2 = \frac{\nabla_w J(w_0)^T \nabla_w J(w_0)}{\nabla_w J(w_0)^T H \nabla_w J(w_0)}$$

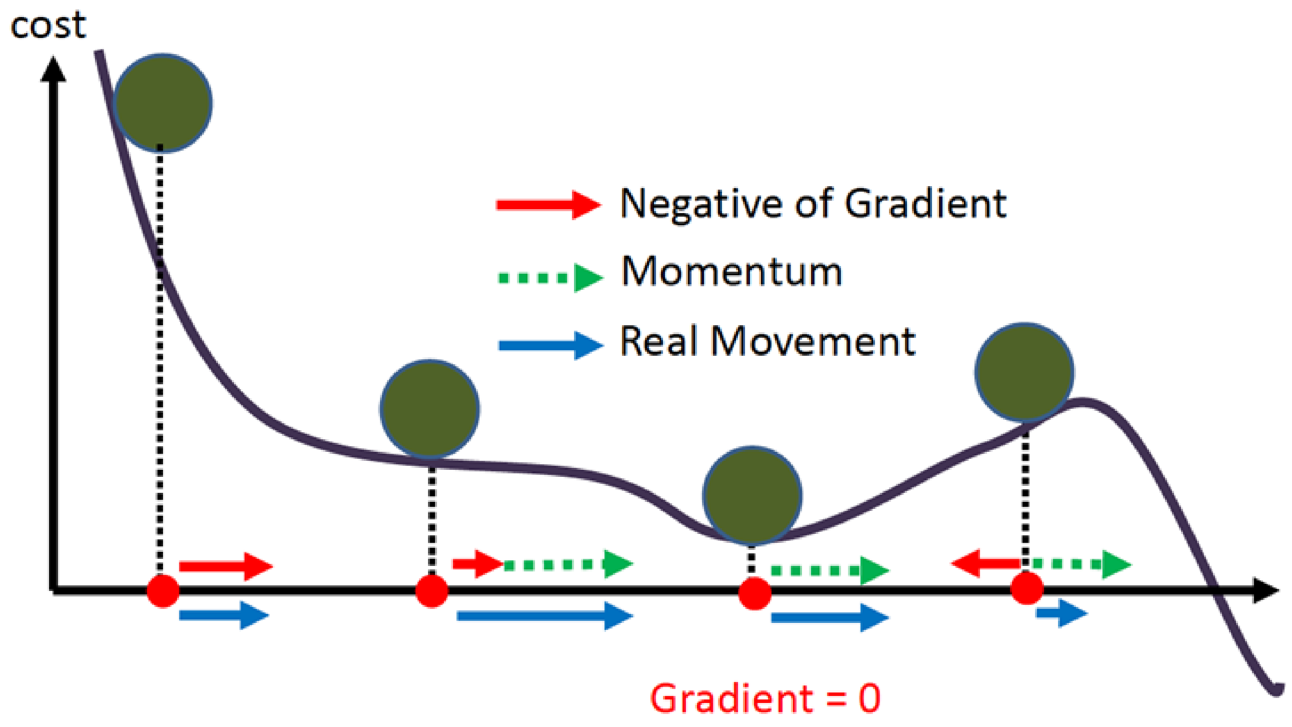
Gradient Descent with Taylor-Approximation



Momentum

- Prefers to go in a similar direction as before

Movement = Negative of Gradient + **Momentum**



Newton's Method

- To solve the critical point that minimizes $J(w)$ approximated by the Taylor-2 expansion at w_0 as the new estimate

$$arg\min_{w} J(w_0) + (w - w_0)^T \nabla_w J(w_0) + \frac{1}{2}(w - w_0)^T H(w - w_0)$$

- By setting the gradient w.r.t. w to zero, we have

$$w^* = w_0 - H\big(J(w_0)\big)^{-1} \nabla_w J(w_0)$$

- The iterative update of w then becomes

$$w^{(n+1)} = w^{(n)} - H\left(J\big(w^{(n)}\big)\right)^{-1} \nabla_w J\big(w^{(n)}\big)$$

Optimization Consideration

- First-order optimization algorithms
  - Optimization algorithms that use only the gradient
  - E.g., Gradient descent
- Second-order optimization algorithms
  - Optimization algorithms that also use the Hessian matrix,
  - E.g., Newton’s method
- Deep learning algorithms tend to lack guarantees
  - Because the family of functions used in deep learning is quite complicated
- In many other fields, the dominant approach to optimization is to design optimization algorithms for a limited family of functions

Optimization Example

- We wish to find $w^* = arg\min_w J(w)$, with

$$J(w) = \frac{1}{4}(w^T M w)^2, M = \begin{bmatrix} 6 & -4 \\ -4 & 6 \end{bmatrix}$$

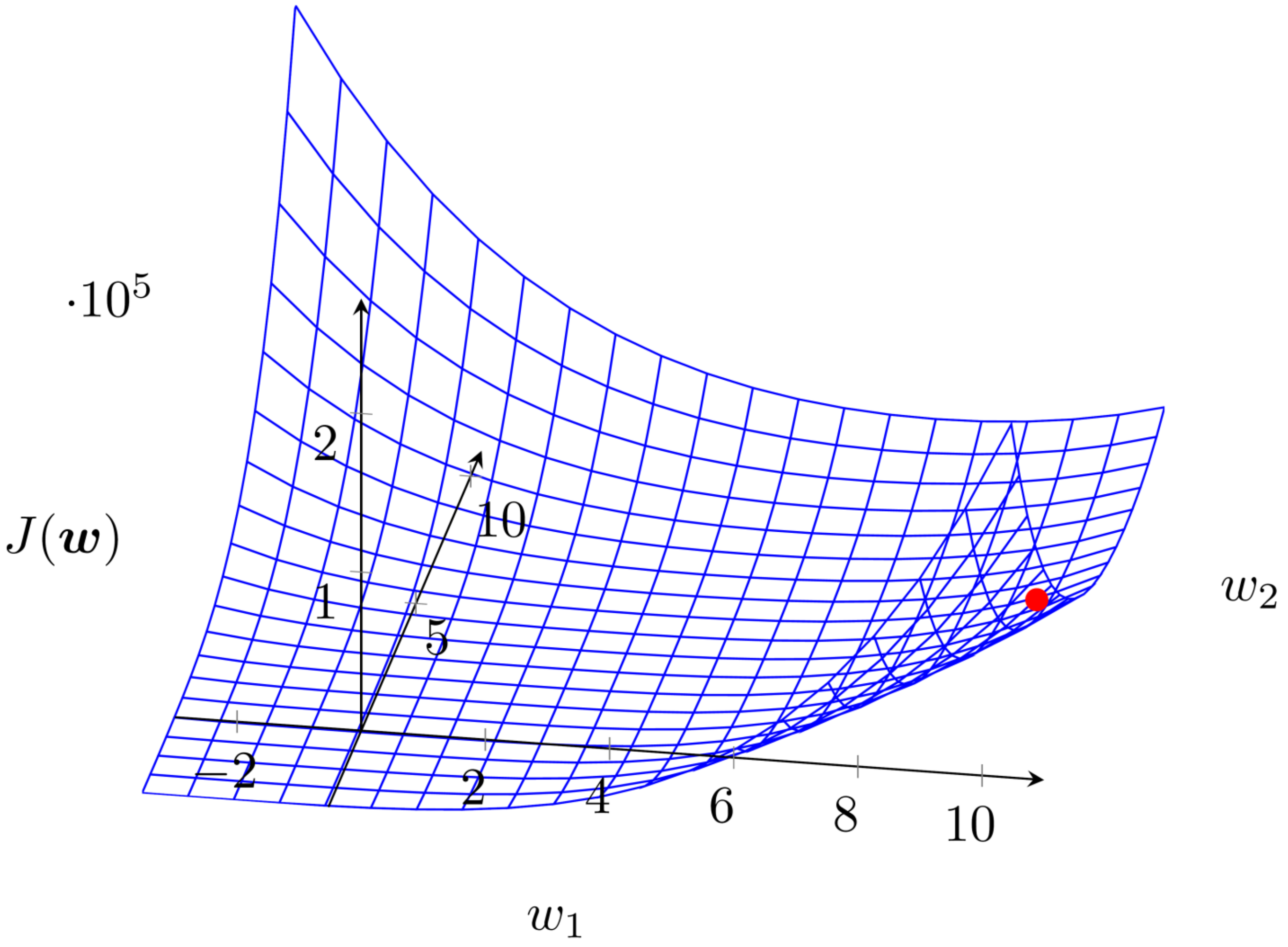
- M has the following pairs of eigenvalues and eigenvectors

$$\lambda_1 = 1 \rightarrow v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \lambda_2 = 5 \rightarrow v_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

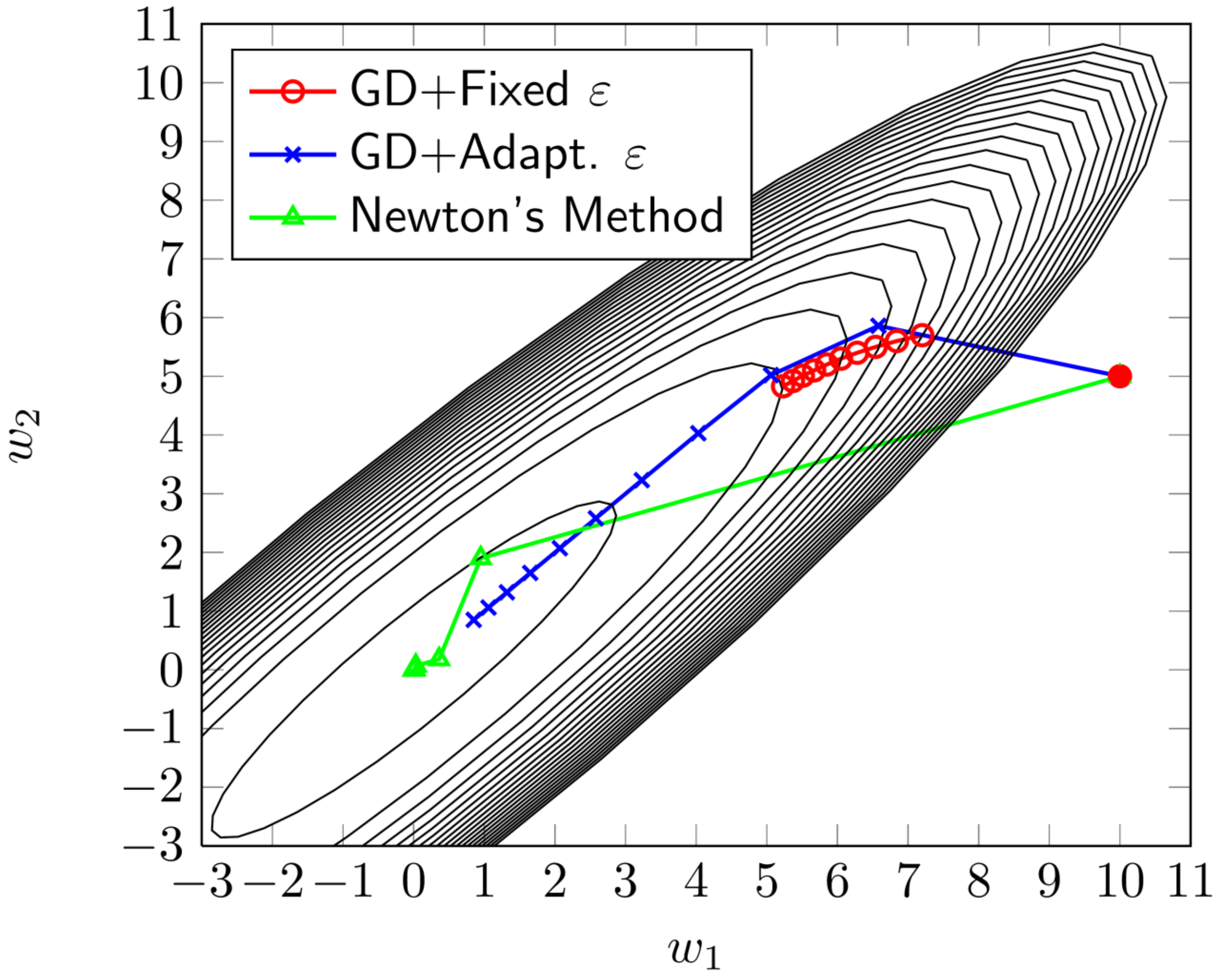
- The gradient and Hessian matrix are given by

$$\nabla_w J(w) = (w^T M w) M w$$
$$H\{J(w)\} = 2Mw(Mw)^T + w^T M w M$$

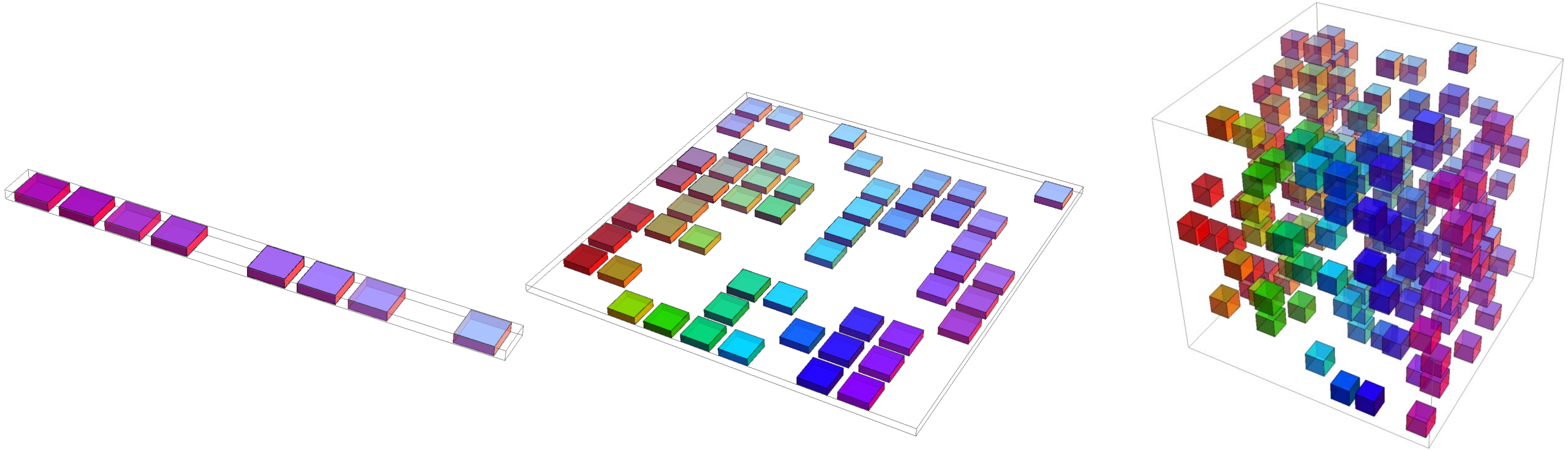
Optimization Example



Optimization Example



Curse of Dimensionality



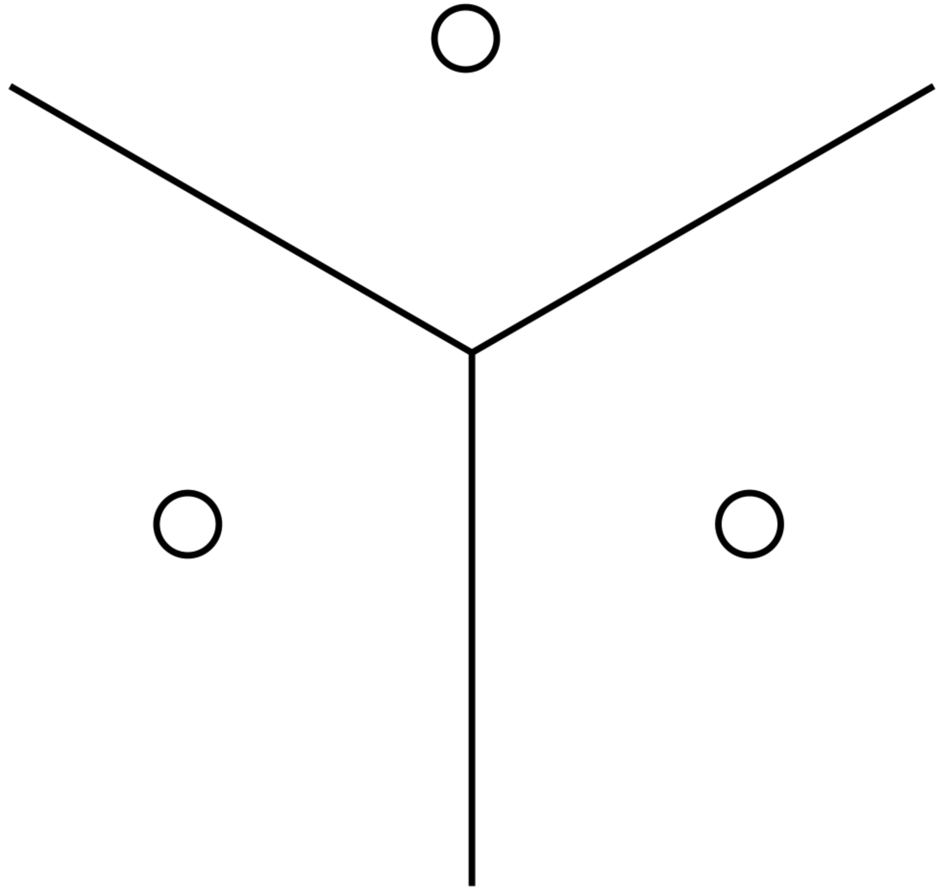
As the number of relevant dimensions of the data increases the number of configurations of interest may grow exponentially

Dealing with Missing Configuration

- Common prior beliefs
  - Local Constancy
  - Smoothness Regularization
- Task-specific assumption
  - the data was generated by the **composition of factors** or features

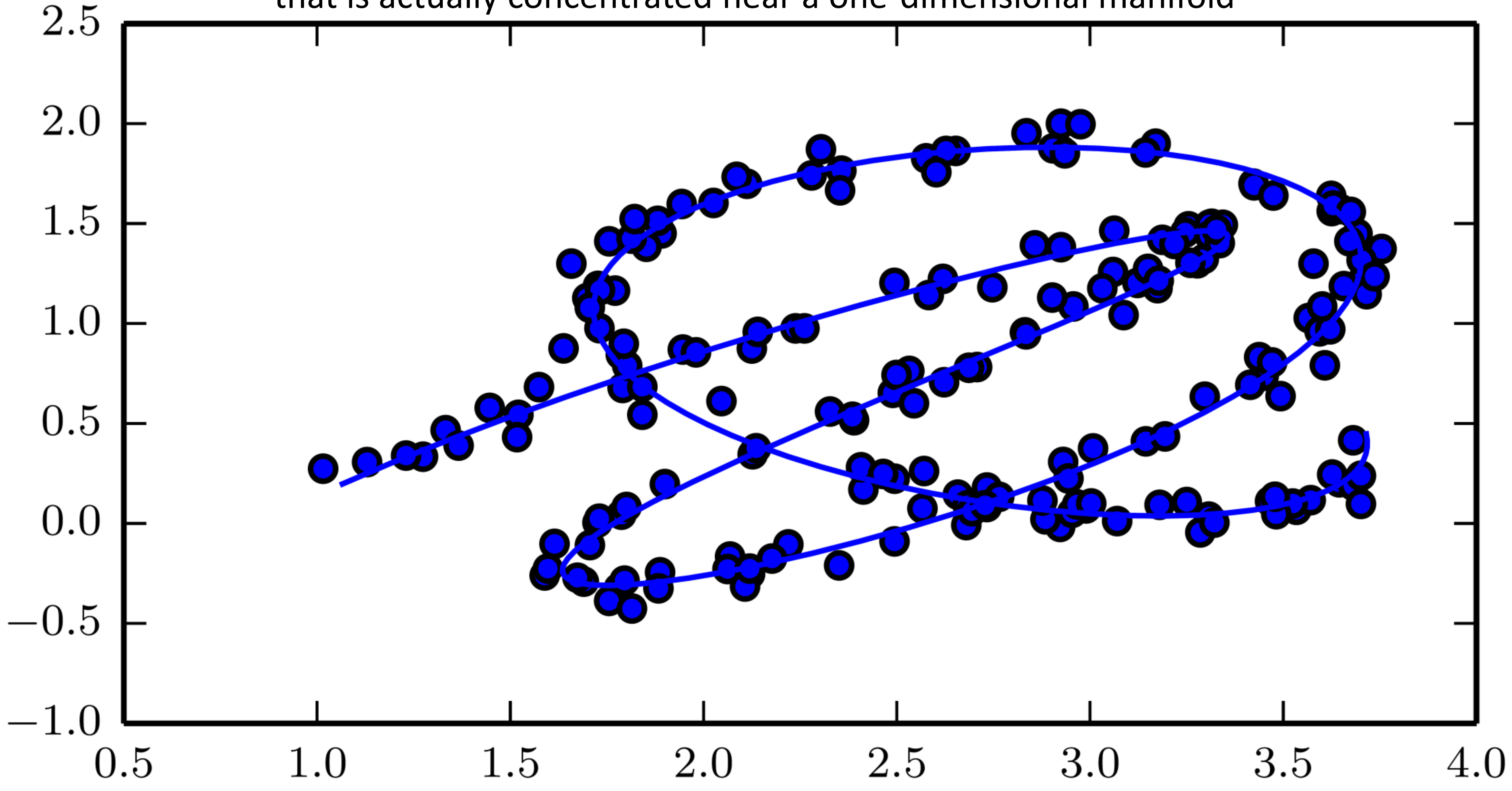
Nearest Neighbor

- breaks up the input space into regions

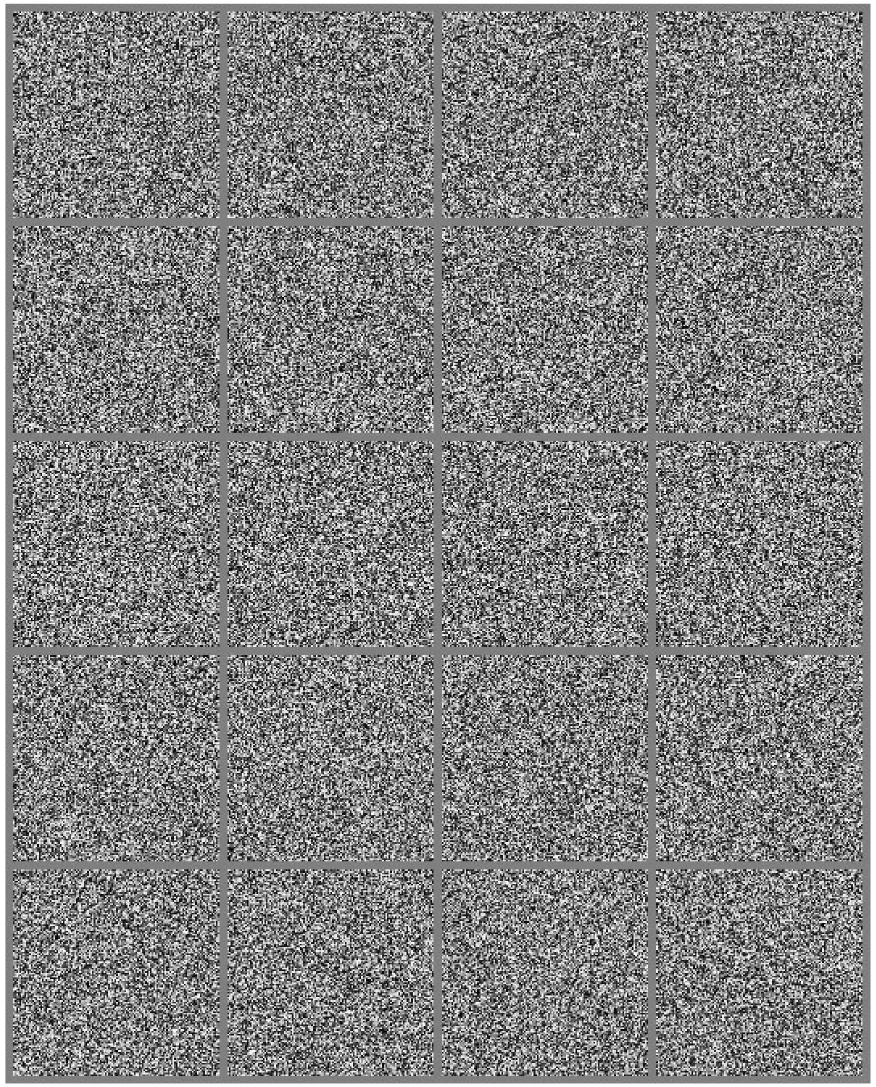


Manifold Learning

Data sampled from a distribution in a two-dimensional space
that is actually concentrated near a one-dimensional manifold



Uniformly Sampled Images



QMUL Dataset