# COMP0137 Machine Vision Lecture Notes

*Melodie Li*

UCL Department of Computer Science

Dec 2024

# Contents

# Chapter 1

# Probability Distribution

## 1.1 Probability Fundamentals

### 1.1.1 Random Variables

**Definition 1.1.1.** A **random variable** $X$ represents a quantity whose outcome is uncertain. It can be:

- The result of an experiment, such as flipping a coin.
- Real-world measurements, like temperature readings.

**Remark.**
- ***Discrete Random Variables:*** *Take on a countable number of distinct values.*
- ***Continuous Random Variables:*** *Can take any value in a continuous range.*

### 1.1.2 Joint Probability Distribution

**Definition 1.1.2.** For two random variables $X$ and $Y$, the **joint probability distribution** $\Pr(X, Y)$ represents the probability of both variables taking specific values simultaneously.

**Remark.** *Observing paired instances of $X$ and $Y$ reveals that some combinations are more likely. This relationship is encapsulated in the joint probability distribution.*

### 1.1.3 Marginal Probability Distribution

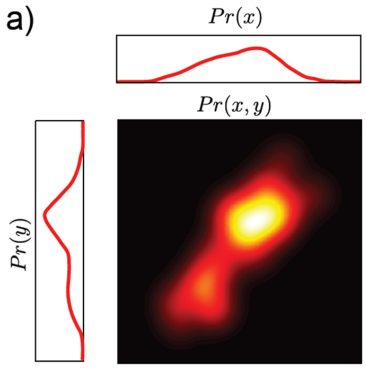



Figure 1.1: Marginal Probability Distribution Visualization 1.

**Theorem 1.1.3.** *The marginal probability of a variable in a joint distribution is obtained by summing or integrating over the other variables:*

- ***Discrete case:***

$$\Pr(X) = \sum_{Y} \Pr(X, Y)$$

- ***Continuous case:***

$$\Pr(X) = \int \Pr(X, Y)\, dY$$

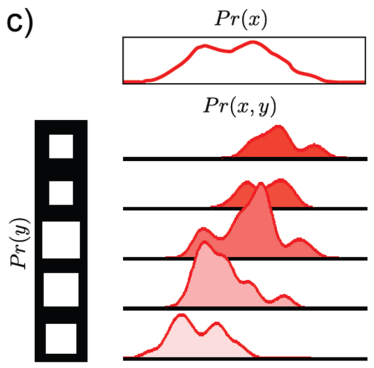



Figure 1.2: Marginal Probability Distribution Visualization 2.

### 1.1.4 Conditional Probability Distribution

**Definition 1.1.4.** The **conditional probability** of $X$ given $Y = y^*$ is the likelihood of $X$ taking specific values, given that $Y$ is fixed:

$$\Pr(X \mid Y = y^*) = \frac{\Pr(X, Y = y^*)}{\Pr(Y = y^*)}.$$

Equivalently, using integration for continuous random variables:

$$\Pr(X \mid Y = y^*) = \frac{\Pr(X, Y = y^*)}{\int \Pr(X, Y = y^*)\, dx}.$$

This can also be written compactly as:

$$\Pr(X \mid Y) = \frac{\Pr(X, Y)}{\Pr(Y)}.$$

Additionally, the relationship can be rearranged to express the joint probability as:

$$\Pr(X, Y) = \Pr(X \mid Y) \Pr(Y),$$

or:

$$\Pr(X, Y) = \Pr(Y \mid X) \Pr(X).$$

This idea can be extended to more than two variables. For instance:

$$\Pr(W, X, Y, Z) = \Pr(W, X, Y \mid Z) \Pr(Z),$$

which can be further expanded as:

$$\Pr(W, X, Y, Z) = \Pr(W, X \mid Y, Z) \Pr(Y \mid Z) \Pr(Z),$$

and finally:

$$\Pr(W, X, Y, Z) = \Pr(W \mid X, Y, Z) \Pr(X \mid Y, Z) \Pr(Y \mid Z) \Pr(Z).$$

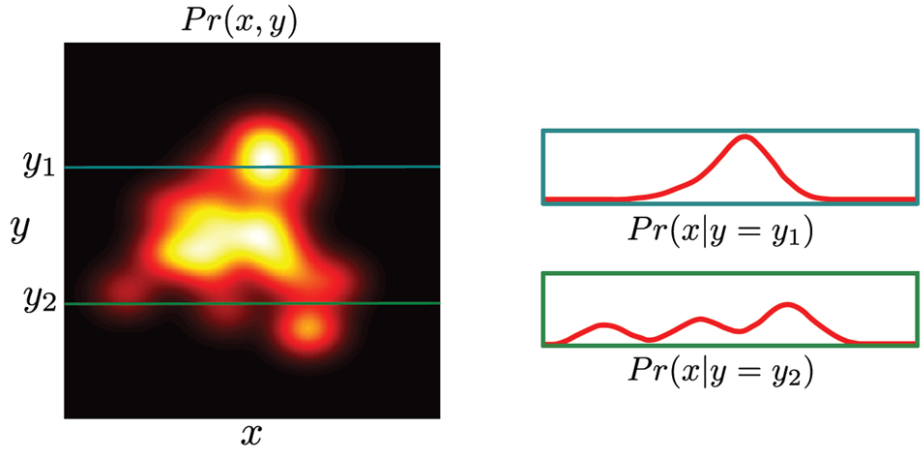



Figure 1.3: Conditional Probability Distribution Visualization.

### 1.1.5 Bayes' Rule

**Theorem 1.1.5.** *Bayes' rule relates conditional and marginal probabilities:*

$$\Pr(X \mid Y) = \frac{\Pr(Y \mid X)\Pr(X)}{\Pr(Y)}.$$

- **Prior:** $\Pr(X)$ — Probability of $X$ before observing $Y$.
- **Likelihood:** $\Pr(Y \mid X)$ — Probability of $Y$ given $X$.
- **Posterior:** $\Pr(X \mid Y)$ — Updated probability of $X$ after observing $Y$.
- **Evidence:** $\Pr(Y)$ — Normalizing constant ensuring a valid distribution.

### 1.1.6 Independence

**Definition 1.1.6.** Two variables $X$ and $Y$ are **independent** if:

$$\Pr(X, Y) = \Pr(X)\Pr(Y).$$

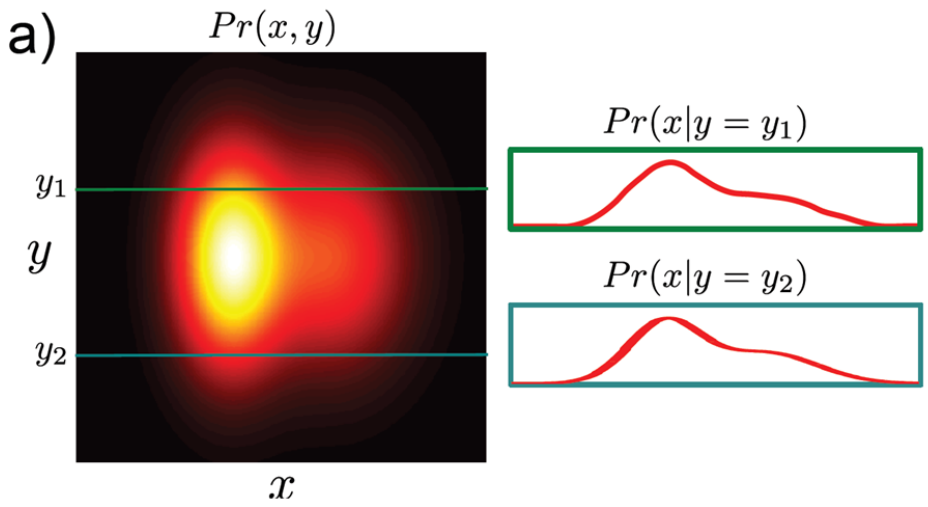



Figure 1.4: Independence Illustration.

### 1.1.7 Expectation

**Definition 1.1.7.** The **expectation** of a function $f(X)$ with respect to the probability distribution of $X$ is:

$$\mathbb{E}[f(X)] = \int f(X)\,\Pr(X)\,dX \quad \text{(continuous case)},$$

or

$$\mathbb{E}[f(X)] = \sum_X f(X)\,\Pr(X) \quad \text{(discrete case)}.$$

- **Linear Properties:**

$$\begin{aligned} \mathbb{E}[a] &= a, \quad \text{(constant expectation)} \\ \mathbb{E}[af(X)] &= a\mathbb{E}[f(X)], \\ \mathbb{E}[f(X) + g(X)] &= \mathbb{E}[f(X)] + \mathbb{E}[g(X)]. \end{aligned}$$

- **Independence Rule:** If $X$ and $Y$ are independent:

$$\mathbb{E}[f(X)g(Y)] = \mathbb{E}[f(X)]\mathbb{E}[g(Y)].$$

| Function $f[\bullet]$ | Expectation |
|---|---|
| $x$ | mean, $\mu_x$ |
| $x^k$ | $k^{th}$ moment about zero |
| $(x-\mu_x)^k$ | $k^{th}$ moment about the mean |
| $(x-\mu_x)^2$ | variance |
| $(x-\mu_x)^3$ | skew |
| $(x-\mu_x)^4$ | kurtosis |
| $(x-\mu_x)(y-\mu_y)$ | covariance of $x$ and $y$ |

Figure 1.5: Expectation: Common Cases.

## 1.2 Common Probability Distributions

### 1.2.1 Bernoulli Distribution

**Definition 1.2.1.** The **Bernoulli distribution** models binary outcomes $y \in \{0, 1\}$ with a single parameter $p$, the probability of success:

$$\Pr(y) = \begin{cases} p & \text{if } y = 1, \\ 1-p & \text{if } y = 0. \end{cases}$$

For short, it is written as:

$$y \sim \text{Bernoulli}(p).$$

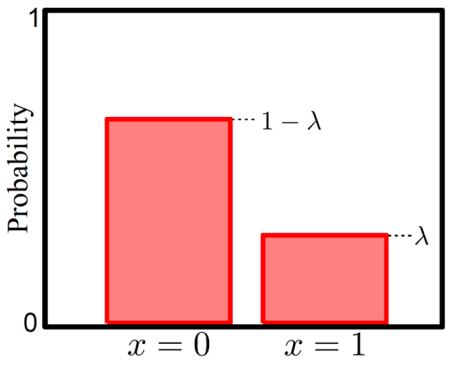



Figure 1.6: Bernoulli Illustration.

### 1.2.2 Beta Distribution

**Definition 1.2.2.** The **Beta distribution** is defined over the interval $\lambda \in [0, 1]$ (commonly used as the parameter of a Bernoulli distribution) and is parameterized by two positive parameters $\alpha$ and $\beta$. Its probability density function is given by:

$$\Pr(\lambda) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\lambda^{\alpha-1}(1-\lambda)^{\beta-1},$$

where $\Gamma(z)$ is the Gamma function:

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt.$$

For short, this can be written as:

$$\Pr(\lambda) = \text{Beta}(\lambda \mid \alpha, \beta).$$

The Beta distribution has the following properties:

- **Mean:** The expected value of $\lambda$ is:
$$\mathbb{E}[\lambda] = \frac{\alpha}{\alpha+\beta}.$$
- **Concentration:** The variance of the distribution is inversely proportional to the sum of the parameters, $\alpha+\beta$, indicating that larger values of $\alpha+\beta$ result in higher concentration around the mean.
- **Interpretation:**
  - The relative values of $\alpha$ and $\beta$ determine the skewness of the distribution.
  - Higher $\alpha$ emphasizes values closer to $1$, while higher $\beta$ emphasizes values closer to $0$.

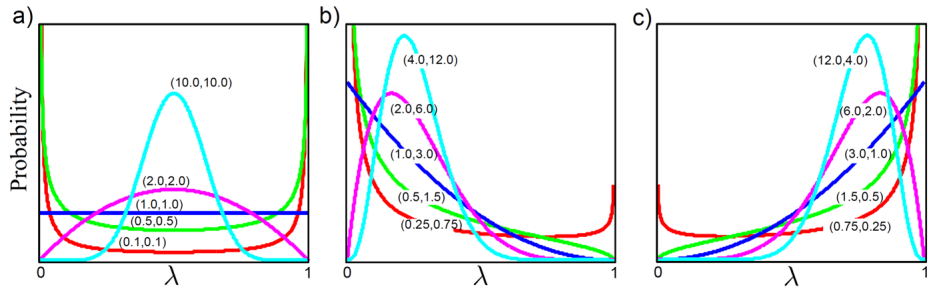



Figure 1.7: Beta Illustration.

### 1.2.3 Categorical Distribution

**Definition 1.2.3.** The **Categorical distribution** describes a situation with $K$ possible outcomes, where each outcome $k$ is associated with a probability $\lambda_k$. The probabilities satisfy:

$$\sum_{k=1}^{K} \lambda_k = 1, \quad \lambda_k \in [0, 1].$$

The probability of a specific outcome $x = k$ is:

$$\Pr(x = k) = \lambda_k.$$

Alternatively, for a unit vector representation $\mathbf{x}$ (where only the $k$th element is $1$ and all others are $0$), the probability can be expressed as:

$$\Pr(x = \mathbf{e}_k) = \prod_{j=1}^{K} \lambda_j^{x_j} = \lambda_k,$$

where $x_j$ is the $j$th element of $\mathbf{x}$, i.e., a binary indicator (0 or 1).

For short, we write:

$$\Pr(x) = \mathsf{Cat}_x(\boldsymbol{\lambda}),$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_K)$.

The Categorical distribution is a generalization of the Bernoulli distribution to $K$ outcomes, and each $\lambda_k$ determines the likelihood of observing the corresponding outcome.

### 1.2.4 Dirichlet Distribution

**Definition 1.2.4.** The **Dirichlet distribution** is a generalization of the Beta distribution to $K$ dimensions, modeling probabilities $\boldsymbol{\lambda}$ over a simplex, where:

$$\lambda_k \geq 0, \quad \sum_{k=1}^{K} \lambda_k = 1.$$

The probability density function is defined as:

$$\Pr(\lambda_1, \ldots, \lambda_K) = \frac{\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \lambda_k^{\alpha_k - 1},$$

where $\alpha_1, \alpha_2, \ldots, \alpha_K > 0$ are the parameters of the distribution, and $\Gamma(z)$ is the Gamma function:

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t}\, dt.$$

For short, this is denoted as:

$$\Pr(\lambda_1, \ldots, \lambda_K) = \text{Dir}(\boldsymbol{\lambda} \mid \alpha_1, \alpha_2, \ldots, \alpha_K).$$

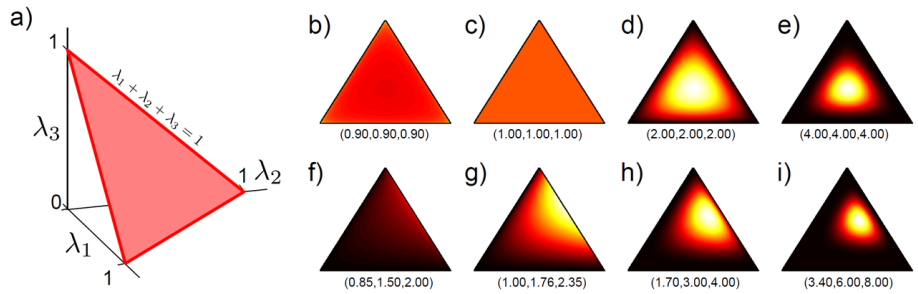



Figure 1.8: Dirichlet Illustration.

### 1.2.5 Univariate Normal Distribution

**Definition 1.2.5.** The **Univariate Normal Distribution** models a single continuous variable $x$:

$$\Pr(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

It is parameterized by the mean $\mu$ and variance $\sigma^2 > 0$.

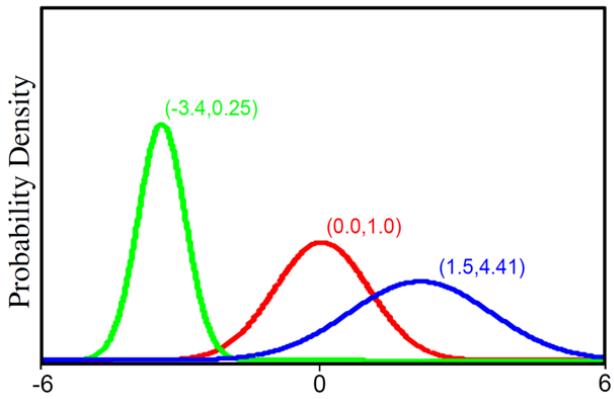



Figure 1.9: Univariate Normal Illustration.

### 1.2.6 Normal Inverse Gamma Distribution

**Definition 1.2.6.** The **Normal Inverse Gamma distribution** is a joint distribution over a mean $\mu$ and variance $\sigma^2 > 0$, parameterized by four positive parameters $\alpha$, $\beta$, $\gamma$, and $\delta$. The probability density function is given by:

$$\Pr(\mu, \sigma^2) = \sqrt{\frac{\gamma}{2\pi}} \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma^2}\right)^{\alpha+1} \exp\left(-\frac{2\beta + \gamma(\delta - \mu)^2}{2\sigma^2}\right).$$

For short, this is denoted as:

$$\Pr(\mu, \sigma^2) = \text{NormInvGam}(\mu, \sigma^2 \mid \alpha, \beta, \gamma, \delta).$$

Key properties of the Normal Inverse Gamma distribution:

- **Support:** $\sigma^2 > 0$ and $\mu \in \mathbb{R}$.
- **Parameters:**
  - $\alpha$ and $\beta$ control the shape and scale of the variance $\sigma^2$.
  - $\gamma$ and $\delta$ influence the mean $\mu$ and its variance.

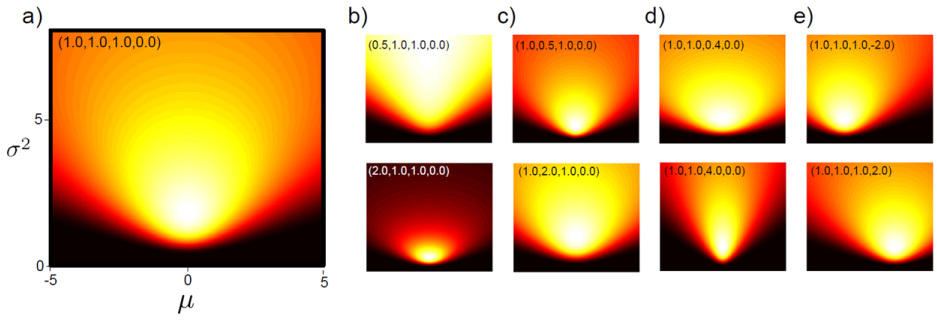



Figure 1.10: Normal Inverse Gamma Illustration.

### 1.2.7 Multivariate Normal Distribution

### 1.2.8 Multivariate Normal Distribution

**Definition 1.2.7.** The **Multivariate Normal Distribution** generalizes the univariate normal distribution to $D$ dimensions. Its probability density function is given by:

$$\Pr(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^D \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right),$$

For short, this is denoted as:

$$\Pr(\mathbf{x}) = \text{Norm}_{\mathbf{x}}(\boldsymbol{\mu}, \Sigma).$$

Key properties:

- The covariance matrix $\Sigma$ determines the spread and orientation of the distribution. It must be symmetric and positive-definite, meaning for any real vector $\mathbf{z}$:

$$\mathbf{z}^\top \Sigma \mathbf{z} > 0.$$

- The mean vector $\boldsymbol{\mu}$ specifies the location of the peak of the distribution in $D$-dimensional space.

**Types of Covariance Matrices**

The covariance matrix $\Sigma$ can take different forms:

- **Spherical:** All dimensions have the same variance, and there are no covariances between dimensions:

$$\Sigma_{\text{spher}} = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}.$$

- **Diagonal:** Each dimension has its own variance, but there are no covariances between dimensions:

$$\Sigma_{\text{diag}} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}.$$

- **Full:** Allows for variances and covariances between dimensions:

$$\Sigma_{\text{full}} = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12} \\ \sigma_{21} & \sigma_{22}^2 \end{bmatrix}.$$

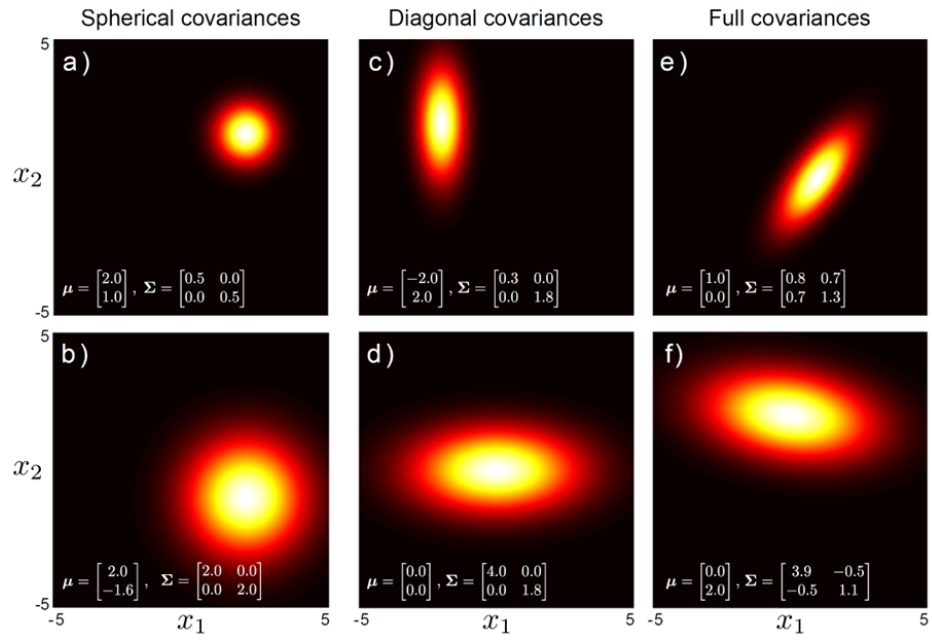



Figure 1.11: Multi-Normal Types of Covariances.

### 1.2.9 Normal Inverse Wishart Distribution

**Definition 1.2.8.** The **Normal Inverse Wishart distribution** is a joint distribution over a mean vector $\boldsymbol{\mu} \in \mathbb{R}^D$ and a symmetric positive-definite covariance matrix $\Sigma \in \mathbb{R}^{D \times D}$. It is parameterized by four values: a positive scalar $\alpha$, a positive-definite scale matrix $\Psi$, a positive scalar $\gamma$, and a vector $\boldsymbol{\delta} \in \mathbb{R}^D$.

The probability density function is given by:

$$\Pr(\boldsymbol{\mu}, \Sigma) = \frac{\gamma^{D/2}|\Psi|^{\alpha/2}|\Sigma|^{-(\alpha+D+2)/2}}{(2\pi)^{D/2}2^{\alpha\gamma/2}\Gamma_D(\alpha/2)} \exp\left\{-\frac{1}{2}\left(\mathrm{Tr}(\Psi\Sigma^{-1}) + \gamma(\boldsymbol{\mu}-\boldsymbol{\delta})^\top\Sigma^{-1}(\boldsymbol{\mu}-\boldsymbol{\delta})\right)\right\},$$

where:

- $|\Sigma|$ denotes the determinant of $\Sigma$,
- $\Gamma_D(\cdot)$ is the multivariate Gamma function:

$$\Gamma_D(a) = \pi^{D(D-1)/4}\prod_{j=1}^{D}\Gamma\left(a - \frac{j-1}{2}\right).$$

For short, this is denoted as:

$$\Pr(\boldsymbol{\mu}, \Sigma) = \text{NormInvWish}(\boldsymbol{\mu}, \Sigma \mid \alpha, \Psi, \gamma, \boldsymbol{\delta}).$$

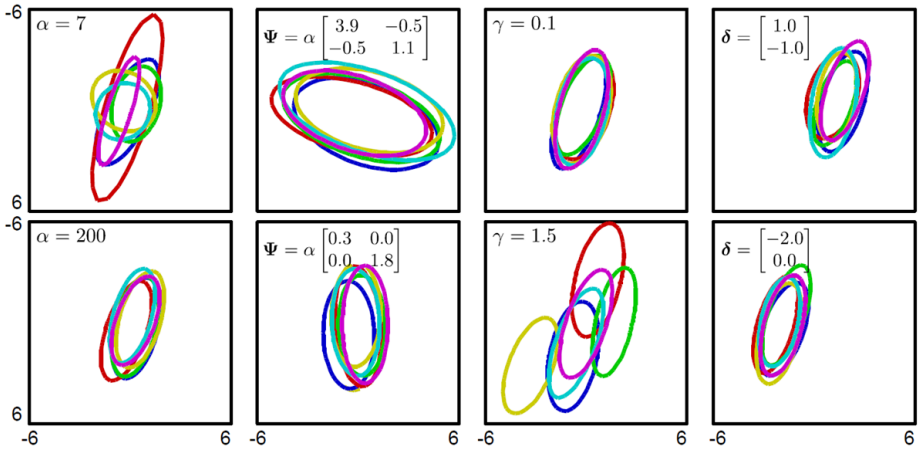



Figure 1.12: Normal Inverse Wishart Illustration.

### 1.2.10 Conjugate Distributions

**Remark.** *Conjugate distributions simplify Bayesian updates. Examples:*

- ***Beta-Bernoulli:*** *A Beta prior with a Bernoulli likelihood produces a Beta posterior.*
- ***Dirichlet-Categorical:*** *A Dirichlet prior with a Categorical likelihood produces a Dirichlet posterior.*
- ***Normal-Inverse Gamma:*** *A Normal-Inverse Gamma prior with a Univariate Normal likelihood produces a Normal-Inverse Gamma posterior.*
- ***Normal-Inverse Wishart:*** *A Normal-Inverse Wishart prior with a Multivariate Normal likelihood produces a Normal-Inverse Wishart posterior.*

### 1.2.11 Key Relationships Between Distributions

- The Beta distribution is used to model the success probability $p$ of a Bernoulli trial.
- The Dirichlet distribution generalizes the Beta distribution to multiple categories.
- The Normal distribution is self-conjugate with respect to the mean, making Bayesian updates straightforward.

| Data Type | Domain | Distribution |
|---|---|---|
| univariate, discrete, binary | $x \in \{0, 1\}$ | Bernoulli |
| univariate, discrete, multi-valued | $x \in \{1, 2, \ldots, K\}$ | categorical |
| univariate, continuous, unbounded | $x \in \mathbb{R}$ | univariate normal |
| univariate, continuous, bounded | $x \in [0, 1]$ | beta |
| multivariate, continuous, unbounded | $\mathbf{x} \in \mathbb{R}^K$ | multivariate normal |
| multivariate, continuous, bounded, sums to one | $\mathbf{x} = [x_1, x_2, \ldots, x_K]^T$<br>$x_k \in [0, 1], \sum_{k=1}^K x_k = 1$ | Dirichlet |
| bivariate, continuous,<br>$x_1$ unbounded,<br>$x_2$ bounded below | $\mathbf{x} = [x_1, x_2]$<br>$x_1 \in \mathbb{R}$<br>$x_2 \in \mathbb{R}^+$ | normal-scaled<br>inverse gamma |
| multivariate vector $\mathbf{x}$ and matrix $\mathbf{X}$<br>$\mathbf{x}$ unbounded,<br>$\mathbf{X}$ square, positive definite | $\mathbf{x} \in \mathbb{R}^K$<br>$\mathbf{X} \in \mathbb{R}^{K \times K}$<br>$\mathbf{z}^T\mathbf{X}\mathbf{z} > 0 \quad \forall\, \mathbf{z} \in \mathbb{R}^K$ | normal<br>inverse Wishart |

**Table 2.1:** Common probability distributions: the choice of distribution depends on the type/domain of data to be modeled.

Figure 1.13:

## 1.3 Fitting Probability Models

### 1.3.1 Maximum Likelihood Estimation (MLE)

**Definition 1.3.1.** The **Maximum Likelihood Estimation** method seeks parameters $\theta$ that maximize the likelihood function:

$$\mathcal{L}(\theta) = \Pr(\text{data} \mid \theta).$$

**Remark.** *For independent data points $x_1, x_2, \ldots, x_N$, the likelihood becomes:*

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} \Pr(x_i \mid \theta).$$

*To simplify computations, we often maximize the log-likelihood:*

$$\log \mathcal{L}(\theta) = \sum_{i=1}^{N} \log \Pr(x_i \mid \theta).$$

### 1.3.2 Maximum A Posteriori (MAP)

**Definition 1.3.2.** The **Maximum A Posteriori** method finds parameters $\theta$ that maximize the posterior probability:

$$\Pr(\theta \mid \text{data}) = \frac{\Pr(\text{data} \mid \theta)\Pr(\theta)}{\Pr(\text{data})}.$$

**Remark.** *The posterior combines the likelihood* $\Pr(\textit{data} \mid \theta)$ *and the prior* $\Pr(\theta)$*. Since the evidence* $\Pr(\textit{data})$ *is constant for a given dataset, we maximize:*

$$\Pr(\theta \mid \textit{data}) \propto \Pr(\textit{data} \mid \theta)\Pr(\theta).$$

### 1.3.3 Bayesian Approach

**Definition 1.3.3.** The **Bayesian Approach** computes the posterior distribution over all possible parameter values using Bayes' rule:

$$\Pr(\theta \mid \text{data}) = \frac{\Pr(\text{data} \mid \theta)\Pr(\theta)}{\Pr(\text{data})}.$$

**Remark.** *Instead of finding a single parameter value, this approach:*

- *Captures the uncertainty in parameter estimation.*
- *Uses the posterior to make predictions by integrating over all possible parameters.*

#### Predictive Density in Bayesian Approach

- Prediction for a new data point $x^*$:

$$\Pr(x^* \mid \text{data}) = \int \Pr(x^* \mid \theta)\Pr(\theta \mid \text{data})\, d\theta.$$

- This is a weighted sum of predictions for all parameter values, weighted by their posterior probabilities.

### 1.3.4 Example: Fitting a Normal Distribution

#### Maximum Likelihood Estimation (MLE)

The Maximum Likelihood Estimation aims to maximize the likelihood of the data given the parameters $\mu$ and $\sigma^2$. The likelihood is given by:

$$\Pr(\mathbf{x}_1, \ldots, \mathbf{x}_I \mid \mu, \sigma^2) = \prod_{i=1}^{I} \Pr(x_i \mid \mu, \sigma^2),$$

where:

$$\Pr(x \mid \mu, \sigma^2) = \text{Norm}_x(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right).$$

Taking the logarithm for computational convenience, the log-likelihood becomes:

$$\log \Pr(\mathbf{x}_1, \ldots, \mathbf{x}_I \mid \mu, \sigma^2) = -\frac{I}{2}\log(2\pi) - \frac{I}{2}\log\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{I}(x_i - \mu)^2.$$

To maximize, take the derivatives with respect to $\mu$ and $\sigma^2$ and equate to zero:

$$\hat{\mu} = \frac{1}{I}\sum_{i=1}^{I} x_i, \quad \hat{\sigma}^2 = \frac{1}{I}\sum_{i=1}^{I}(x_i - \hat{\mu})^2.$$

The MLE solution is simply the sample mean and sample variance.

**Maximum A Posteriori (MAP)**

In MAP, we incorporate prior information about the parameters by maximizing the posterior distribution:

$$\Pr(\mu, \sigma^2 \mid \mathbf{x}_1, \ldots, \mathbf{x}_I) \propto \Pr(\mathbf{x}_1, \ldots, \mathbf{x}_I \mid \mu, \sigma^2)\Pr(\mu, \sigma^2).$$

Using a Normal-Inverse-Gamma prior:

$$\Pr(\mu, \sigma^2) = \text{NormInvGam}(\mu, \sigma^2 \mid \alpha, \beta, \gamma, \delta),$$

the posterior distribution becomes:

$$\Pr(\mu, \sigma^2 \mid \mathbf{x}_1, \ldots, \mathbf{x}_I) \propto \prod_{i=1}^{I} \Pr(x_i \mid \mu, \sigma^2)\Pr(\mu, \sigma^2).$$

Taking the logarithm and maximizing:

$$\hat{\mu} = \frac{\sum_{i=1}^{I} x_i + \gamma\delta}{I + \gamma}, \quad \hat{\sigma}^2 = \frac{\sum_{i=1}^{I}(x_i - \mu)^2 + 2\beta + \gamma(\delta - \mu)^2}{I + 3 + 2\alpha}.$$

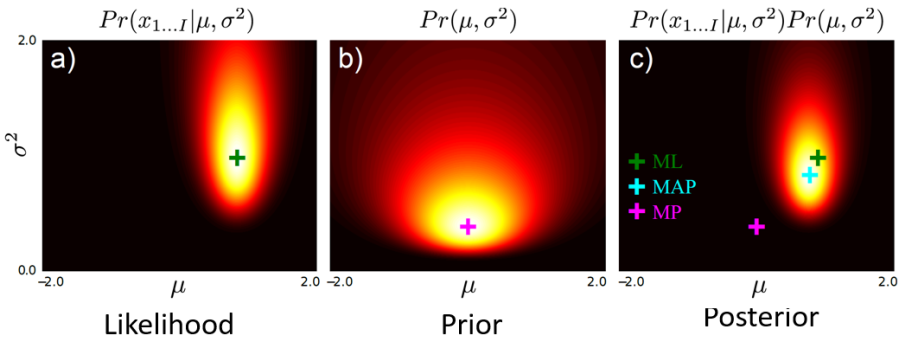



Figure 1.14: Fitting Normal Distribution: MAP.

The MAP solution incorporates prior beliefs about $\mu$ and $\sigma^2$.

**Bayesian Approach**

In the Bayesian approach, we compute the posterior distribution of the parameters using Bayes' rule:

$$\Pr(\mu, \sigma^2 \mid \mathbf{x}_1, \ldots, \mathbf{x}_I) = \frac{\Pr(\mathbf{x}_1, \ldots, \mathbf{x}_I \mid \mu, \sigma^2) \Pr(\mu, \sigma^2)}{\Pr(\mathbf{x}_1, \ldots, \mathbf{x}_I)}.$$

Assuming a Normal-Inverse-Gamma prior:

$$\Pr(\mu, \sigma^2) = \text{NormInvGam}(\mu, \sigma^2 \mid \alpha, \beta, \gamma, \delta),$$

the posterior distribution remains a Normal-Inverse-Gamma distribution:

$$\Pr(\mu, \sigma^2 \mid \mathbf{x}_1, \ldots, \mathbf{x}_I) = \text{NormInvGam}(\mu, \sigma^2 \mid \widetilde{\alpha}, \widetilde{\beta}, \widetilde{\gamma}, \widetilde{\delta}),$$

where the updated parameters are:

$$\widetilde{\alpha} = \alpha + I/2, \quad \widetilde{\beta} = \beta + \frac{\sum_{i=1}^{I} x_i^2}{2}, \quad \widetilde{\gamma} = \gamma + I, \quad \widetilde{\delta} = \frac{\gamma\delta + \sum_{i=1}^{I} x_i}{\gamma + I}.$$

The posterior combines the likelihood of the data and the prior, reflecting both the observed data and prior beliefs.

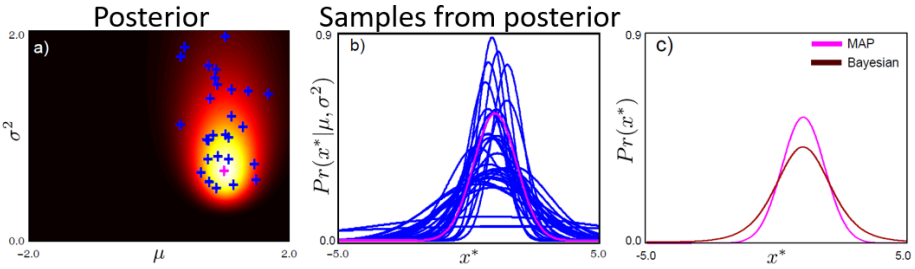



Figure 1.15: Fitting Normal: Bayesian Approach.

**Predictive Density**

To make predictions, we compute the predictive density by integrating over the posterior distribution:

$$\Pr(x^* \mid \mathbf{x}_1, \ldots, \mathbf{x}_I) = \int\int \Pr(x^* \mid \mu, \sigma^2) \Pr(\mu, \sigma^2 \mid \mathbf{x}_1, \ldots, \mathbf{x}_I)\, d\mu\, d\sigma^2.$$

Using the posterior distribution, the predictive density simplifies to:

$$\Pr(x^* \mid \mathbf{x}_1, \ldots, \mathbf{x}_I) = \text{Norm}\left(x^* \mid \widetilde{\mu}, \widetilde{\sigma}^2\right),$$

where:

$$\widetilde{\mu} = \frac{\gamma\delta + \sum_{i=1}^{I} x_i}{\gamma + I}, \quad \widetilde{\sigma}^2 = \frac{\beta + \sum_{i=1}^{I}(x_i - \widetilde{\mu})^2}{\widetilde{\alpha} - 1}.$$

The Bayesian approach naturally incorporates uncertainty in parameter estimation, resulting in a predictive density that reflects both the data and prior beliefs.

### 1.3.5 Example: Fitting a Categorical Distribution

- Parameters: $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_K)$, where $\sum_{k=1}^{K} \theta_k = 1$.
- Likelihood:

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{k=1}^{K} \theta_k^{N_k},$$

  where $N_k$ is the count of observations in category $k$.
- MLE:

$$\theta_k = \frac{N_k}{\sum_{j=1}^{K} N_j}.$$

- MAP:
$$\theta_k = \frac{N_k + a_k - 1}{\sum_{j=1}^{K}(N_j + a_j - 1)},$$
where $a_k$ are parameters of the Dirichlet prior.

### 1.3.6 Conclusion

**Remark.** *Three main approaches to fitting probability models:*

- ***MLE:*** *Directly maximizes likelihood, often leading to overfitting.*
- ***MAP:*** *Incorporates prior beliefs, balancing likelihood with prior knowledge.*
- ***Bayesian:*** *Provides a complete description of parameter uncertainties, requiring computationally intensive methods like sampling.*

## 1.4 The Normal Distribution

### 1.4.1 Univariate Normal Distribution

**Definition 1.4.1.** The **Univariate Normal Distribution** models a single continuous variable $x$:

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

where:

- $\mu$: Mean (central tendency).
- $\sigma^2$: Variance (spread), with $\sigma^2 > 0$.

For short, it is written as:

$$x \sim \mathcal{N}(\mu, \sigma^2).$$

### 1.4.2 Multivariate Normal Distribution

**Definition 1.4.2.** The **Multivariate Normal Distribution** generalizes the normal distribution to $D$ dimensions:

$$\Pr(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^D \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

where:

- $\mathbf{x}$: $D$-dimensional vector of random variables.
- $\boldsymbol{\mu}$: $D$-dimensional mean vector.
- $\Sigma$: Positive-definite covariance matrix.

For short:

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma).$$

**Types of Covariance Structures**

- **Spherical:** $\Sigma = \sigma^2 I$, implying equal variance in all directions and no correlations.
- **Diagonal:** Non-zero entries only along the diagonal of $\Sigma$, representing uncorrelated variables with different variances.
- **Full:** General case with off-diagonal entries, allowing for correlations between variables.

### 1.4.3 Key Properties of the Normal Distribution

**Theorem 1.4.3** (Marginal Distributions)**.** *The marginal distribution of a subset of variables from a multivariate normal is also normal.*

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma) \implies \mathbf{x}_A \sim \mathcal{N}(\boldsymbol{\mu}_A, \Sigma_{AA}), \tag{1.1}$$

*where $\mathbf{x}_A$ represents the subset of variables.*

**Theorem 1.4.4** (Conditional Distributions)**.** *The conditional distribution of one set of variables given another is normal:*

$$\Pr(\mathbf{x}_A \mid \mathbf{x}_B) \sim \mathcal{N}(\boldsymbol{\mu}_A + \Sigma_{AB}\Sigma_{BB}^{-1}(\mathbf{x}_B - \boldsymbol{\mu}_B), \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA}), \tag{1.2}$$

*where:*

- $\boldsymbol{\mu}_A$, $\boldsymbol{\mu}_B$*: Means of the subsets.*
- $\Sigma_{AB}$*: Covariance between subsets $A$ and $B$.*
- $\Sigma_{AA}$, $\Sigma_{BB}$*: Covariance within each subset.*

### 1.4.4 Transformations of Variables

**Theorem 1.4.5** (Linear Transformations)**.** *A linear transformation of a normal random variable results in another normal random variable. If:*

$$\mathbf{y} = A\mathbf{x} + \mathbf{b},$$

*where $A$ is a matrix and $\mathbf{b}$ is a vector, then:*

$$\mathbf{y} \sim \mathcal{N}(A\boldsymbol{\mu} + \mathbf{b}, A\Sigma A^\top).$$

**Remark.** *This property is useful for generating samples from an arbitrary normal distribution by transforming samples from the standard normal distribution.*

### 1.4.5 Product of Two Normals

**Theorem 1.4.6** (Self-Conjugacy)**.** *The product of two normal distributions in the same variable is proportional to a third normal distribution. Specifically:*

$$\mathcal{N}(\mu_1, \sigma_1^2) \times \mathcal{N}(\mu_2, \sigma_2^2) \propto \mathcal{N}(\mu_c, \sigma_c^2),$$

*where:*

$$\mu_c = \frac{\mu_1/\sigma_1^2 + \mu_2/\sigma_2^2}{1/\sigma_1^2 + 1/\sigma_2^2},$$
$$\sigma_c^2 = \frac{1}{1/\sigma_1^2 + 1/\sigma_2^2}.$$

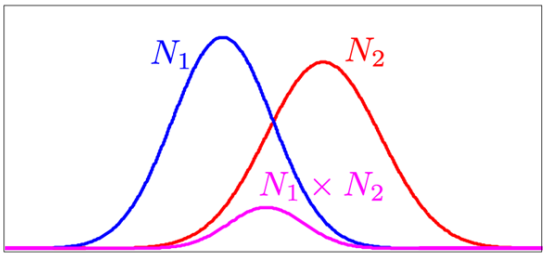



Figure 1.16: Product of Two Normals.

### 1.4.6 Change of Variables

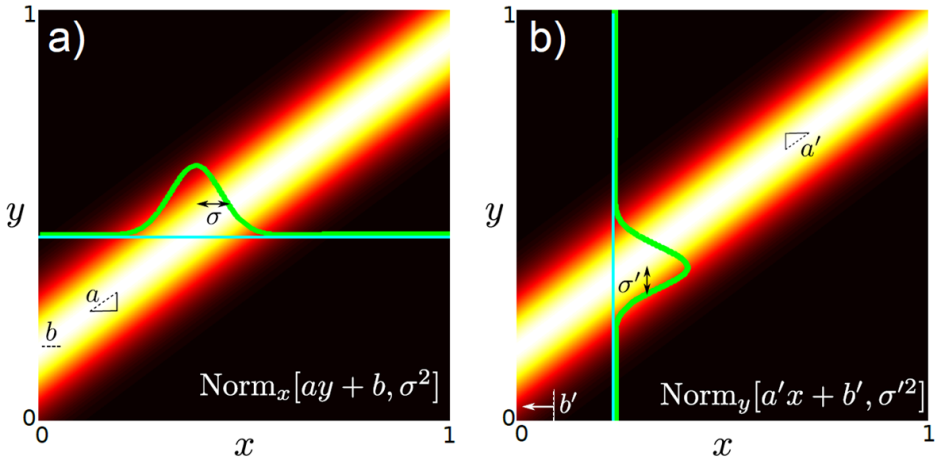



Figure 1.17: Change of Variables.

If the mean of a normal distribution in $x$ is proportional to $y$, then the distribution can be re-expressed as a normal distribution in $y$ that is proportional to $x$. This transformation is expressed as:

$$\mathrm{Norm}_x(\mathbf{A}\mathbf{y}+\mathbf{b}, \Sigma) = \kappa\, \mathrm{Norm}_y(\mathbf{A}'x+\mathbf{b}', \Sigma'),$$

where:

$$\mathbf{A}' = \Sigma\mathbf{A}^\top(\mathbf{A}\Sigma\mathbf{A}^\top)^{-1}, \quad \mathbf{b}' = -\Sigma\mathbf{A}^\top(\mathbf{A}\Sigma\mathbf{A}^\top)^{-1}\mathbf{b}, \quad \Sigma' = (\mathbf{A}\Sigma\mathbf{A}^\top)^{-1}.$$

This transformation allows us to switch between equivalent representations of the normal distribution in different variable spaces, preserving the underlying relationships between variables.

# Chapter 2

# Learning and Inference

## 2.1 Components of Solution

To solve the problem, we need the following components:

- **A model:** A mathematical representation that relates the visual data $\mathbf{x}$ to the world state $\mathbf{w}$. The model specifies a family of relationships, where the particular relationship depends on parameters $\boldsymbol{\theta}$.
- **A learning algorithm:** Fits the parameters $\boldsymbol{\theta}$ using paired training examples $(\mathbf{x}_i, \mathbf{w}_i)$.
- **An inference algorithm:** Uses the model to compute $\Pr(\mathbf{w} \mid \mathbf{x})$ for new observed data $\mathbf{x}$.

## 2.2 Types of Models

**Definition 2.2.1.** Models mathematically relate data $x$ to world states $w$. There are three key types:

1. **Discriminative models:** Directly model $\Pr(w \mid x)$.
2. **Joint models:** Model the joint distribution $\Pr(x, w)$.
3. **Generative models:** Model $\Pr(x \mid w)$ and use Bayes' rule for inference.

### 2.2.1 Discriminative Models: $\Pr(w \mid x)$

**Remark.** *Discriminative models directly compute the posterior probability of $w$ given $x$. The components include:*

- ***Formulation:*** *Choose a parametric form for $\Pr(w \mid x)$.*

- ***Learning:*** *Estimate parameters $\theta$ from training data $(x, w)$.*
- ***Inference:*** *Evaluate $\Pr(w \mid x)$ for new observations $x$.*

**Example 2.2.2** (Linear Regression)**.** For continuous $w$, the model is:

$$\Pr(w \mid x) \sim \mathcal{N}(\mu, \sigma^2), \quad \mu = f_0 + f_1 x,$$

where:

- $f_0$: Offset parameter.
- $f_1$: Slope parameter.
- $\sigma^2$: Variance, assumed constant.

The learning algorithm estimates $f_0$, $f_1$, and $\sigma^2$ using methods like MLE or MAP.

**Example 2.2.3** (Logistic Regression)**.** For binary classification, $w \in \{0, 1\}$, the model is:

$$\Pr(w \mid x) = \frac{1}{1 + \exp(-(f_0 + f_1 x))}.$$

Parameters $f_0$ and $f_1$ are learned using MLE, MAP, or Bayesian methods.

### 2.2.2 Joint (Generative) Models: $\Pr(x, w)$

**Remark.** *Joint models describe the combined behavior of $x$ and $w$. Key steps include:*

- *Concatenate $x$ and $w$ to form $z = [x^\top, w^\top]^\top$.*
- *Model the joint distribution $\Pr(z)$, typically using a multivariate normal distribution.*
- *Use Bayes' rule to compute the posterior $\Pr(w \mid x)$.*

**Example 2.2.4** (Multivariate Normal Distribution)**.** If $x$ and $w$ are continuous, their joint distribution is:

$$\Pr(z) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma),$$

where $\boldsymbol{\mu}$ is the mean vector, and $\Sigma$ is the covariance matrix. Inference for $\Pr(w \mid x)$ involves conditioning on $x$.

### 2.2.3 Generative Models: $\Pr(x \mid w)$

**Remark.** *Generative models describe how data $x$ is generated given $w$. They require:*

- *A prior distribution $\Pr(w)$.*
- *Use of Bayes' rule to compute $\Pr(w \mid x)$:*

$$\Pr(w \mid x) = \frac{\Pr(x \mid w)\Pr(w)}{\Pr(x)}.$$

**Example 2.2.5** (Gaussian Likelihood)**.** For continuous $x$, the likelihood is modeled as:

$$\Pr(x \mid w) \sim \mathcal{N}(m, \sigma^2), \quad m = f_0 + f_1 w.$$

Parameters $f_0$, $f_1$, and $\sigma^2$ are learned from data.

## 2.3 Worked Examples: Regression and Classification

### 2.3.1 Regression

**Remark.** *Regression tasks predict continuous $w$. Applications include:*

- *Estimating human pose from silhouette images.*
- *Predicting head pose from facial orientation.*

**Discriminative Model:** $\Pr(w \mid x)$

To model $\Pr(w \mid x)$:

1. Choose an appropriate form for $\Pr(w)$.
2. Make the parameters a function of $x$.
3. Use a function parameterized by $\boldsymbol{\theta}$ to define its shape.

Assuming a normal distribution over $w$ with constant variance, the model becomes:

$$\Pr(w \mid x, \boldsymbol{\theta}) = \text{Norm}_w\left[\phi_0 + \phi_1 x, \sigma^2\right],$$

where:

- $\phi_0$ is the intercept.

- $\phi_1$ is the slope.
- $\sigma^2$ is the constant variance.

This model is referred to as linear regression.

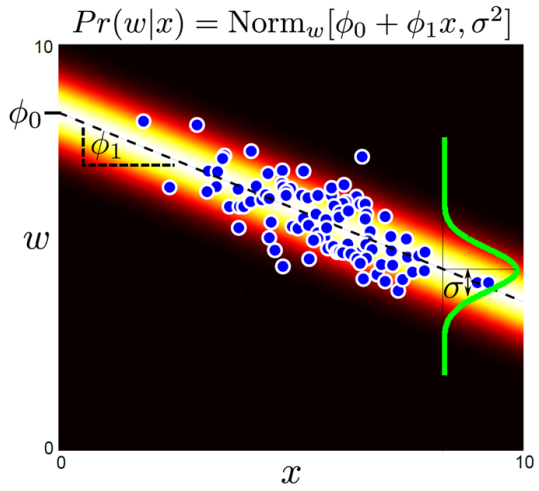



Figure 2.1: Discriminative Model: Regression.

**Generative Model:** $\Pr(x, w)$

To model $\Pr(x, w)$:

1. Concatenate $x$ and $w$ into a single vector $\mathbf{z} = [x, w]^\top$.
2. Model the probability density function of $\mathbf{z}$ as a multivariate normal distribution.
3. Parameterize the pdf using parameters $\boldsymbol{\theta}$.

The resulting model is:

$$\Pr(x, w) = \text{Norm}_{[x,w]}\left[\boldsymbol{\mu}, \Sigma\right],$$

where:

- $\boldsymbol{\mu}$ is the mean vector.
- $\Sigma$ is the covariance matrix.

The inference algorithm computes $\Pr(w \mid x)$ using Bayes' rule:

$$\Pr(w \mid x) = \frac{\Pr(x, w)}{\Pr(x)} = \frac{\Pr(x, w)}{\int \Pr(x, w) dw}.$$

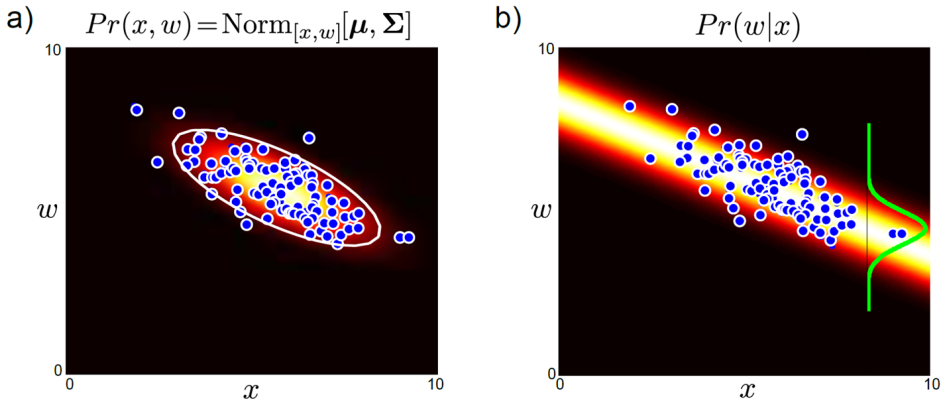



Figure 2.2: Joint Generative Model: Regression

**Generative Model:** $\Pr(x \mid w)$

To model $\Pr(x \mid w)$:

1. Choose an appropriate form for $\Pr(x)$.
2. Make the parameters a function of $w$.
3. Use a function parameterized by $\boldsymbol{\theta}$ to define its shape.

Assuming a normal distribution over $x$ with constant variance, the model becomes:

$$\Pr(x \mid w, \boldsymbol{\theta}) = \text{Norm}_x \left[\phi_0 + \phi_1 w, \sigma^2\right],$$

where:

- $\phi_0$ is the intercept.
- $\phi_1$ is the slope.
- $\sigma^2$ is the constant variance.

The inference algorithm computes $\Pr(w \mid x)$ using Bayes' rule:

$$\Pr(w \mid x) = \frac{\Pr(x \mid w) \Pr(w)}{\int \Pr(x \mid w) \Pr(w) dw}.$$

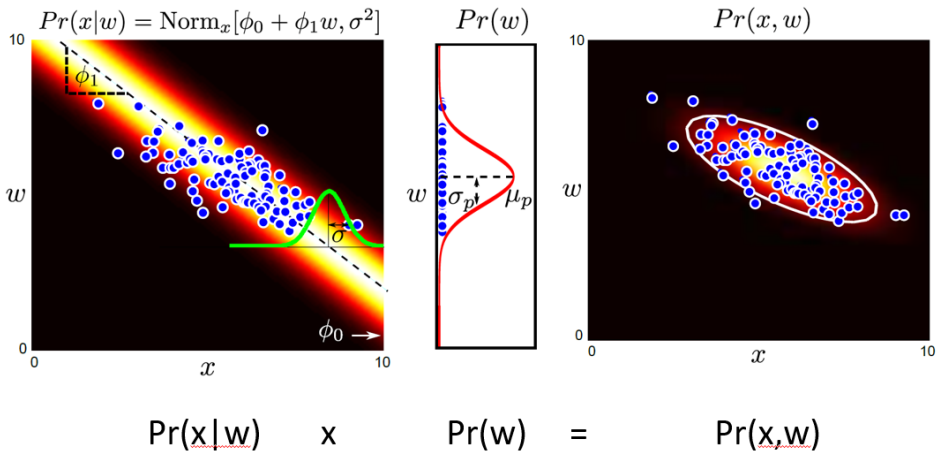



Figure 2.3: Generative Model: Regression.

### 2.3.2 Classification

**Remark.** *Classification tasks predict discrete $w$. Examples include:*

- *Detecting faces in an image.*
- *Identifying pedestrians.*
- *Recognizing faces.*
- *Performing semantic segmentation.*

**Discriminative Model:** $\Pr(w \mid x)$

To model $\Pr(w \mid x)$:

1. Choose an appropriate form for $\Pr(w)$.
2. Make the parameters a function of $x$.
3. Use a function parameterized by $\boldsymbol{\theta}$ to define its shape.

Assuming a Bernoulli distribution for $w$, the model becomes:

$$\Pr(w \mid x, \boldsymbol{\theta}) = \text{Bern}_w\left[\text{sigmoid}(\phi_0 + \phi_1 x)\right] = \text{Bern}_w\left[\frac{1}{1+\exp(-\phi_0 - \phi_1 x)}\right],$$

where:

- $\phi_0$ and $\phi_1$ are the parameters.

This model is known as logistic regression. Parameters $\boldsymbol{\theta} = \{\phi_0, \phi_1\}$ are learned using standard methods like MLE, MAP, or Bayesian inference.

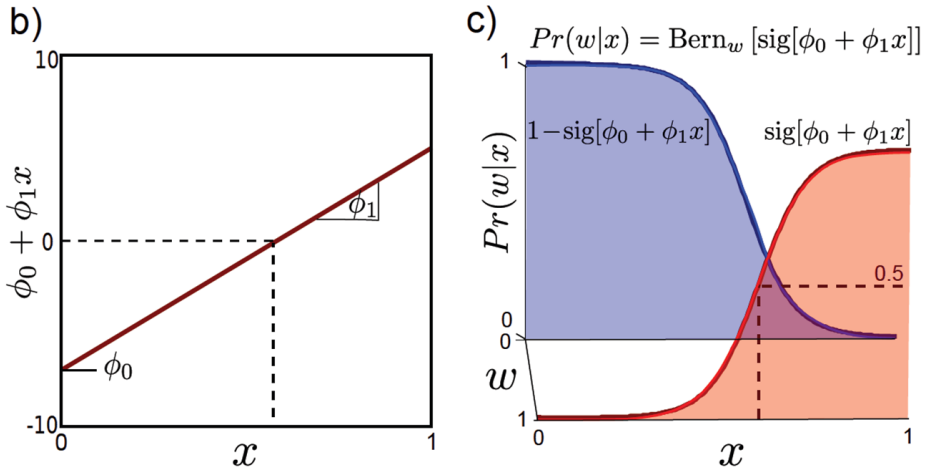



Figure 2.4: Discriminative Model: Classification.

**Generative Model:** $\Pr(x, w)$

A generative model for classification can be challenging to construct because it involves modeling the joint distribution of continuous and discrete variables:

1. Concatenate the continuous vector $x$ and discrete $w$ to form $\mathbf{z}$.
2. There is no straightforward probability distribution to model this joint probability.

**Generative Model:** $\Pr(x \mid w)$

To model $\Pr(x \mid w)$:

1. Choose an appropriate form for $\Pr(x)$.
2. Make the parameters a function of $w$.
3. Use a function parameterized by $\boldsymbol{\theta}$ to define its shape.

Assuming a Gaussian distribution for $x$, the model becomes:

$$\Pr(x \mid w, \boldsymbol{\theta}) = \text{Norm}_x\left[\mu_w, \sigma_w^2\right],$$

where:

- $\mu_w$ and $\sigma_w^2$ are the mean and variance, respectively, conditioned on $w$.

The inference algorithm computes $\Pr(w \mid x)$ using Bayes' rule:

$$\Pr(w \mid x) = \frac{\Pr(x \mid w)\Pr(w)}{\sum_w \Pr(x \mid w)\Pr(w)}.$$

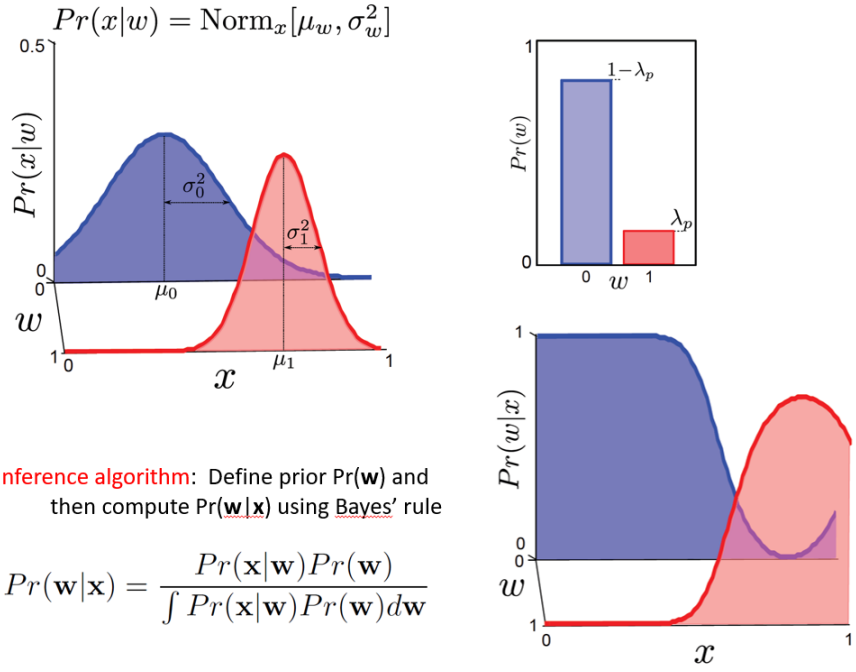



Figure 2.5: Generative Model: Classification.

## 2.4 Modeling Complex Densities

**Remark.** *Complex densities arise in real-world applications where simple models like a single Gaussian are insufficient. This chapter focuses on techniques for modeling such densities, which include:*

- *Mixture of Gaussians (MoG).*
- *t-distributions for robustness to outliers.*
- *Factor analysis for modeling covariance in subspaces.*
- *The Expectation-Maximization (EM) algorithm, a general approach for fitting models involving latent variables.*

*These methods are used in applications such as classification, segmentation, and face recognition.*

### 2.4.1 Densities for Classification

**Remark.** *Classification models rely on computing* $\Pr(w \mid x)$*, which involves:*

- ***Prior:*** $\Pr(w)$*, representing prior knowledge about classes.*
- ***Class-conditional densities:*** $\Pr(x \mid w)$*, describing the likelihood of observing* $x$ *for a given class* $w$*.*

*Using Bayes' rule:*

$$\Pr(w \mid x) = \frac{\Pr(x \mid w)\Pr(w)}{\Pr(x)}.$$

*Class-conditional densities can be modeled using Gaussian or* $t$*-distributions, among others.*

### 2.4.2 Face Detection Experiment

- **Dataset:** 1000 face and 1000 non-face images, each represented as a $10800 \times 1$ vector (60x60x3 images).
- **Model:** Gaussian class-conditional densities with diagonal covariance matrices.
- **Performance:** Achieved 75% accuracy with equal priors $\Pr(w = 0) = \Pr(w = 1) = 0.5$, indicating the need for more sophisticated models.

## 2.5 Models with Hidden Variables

**Definition 2.5.1.** Hidden (or latent) variables $h$ allow complex densities to be modeled as marginalizations:

$$\Pr(x) = \sum_{h} \Pr(x, h),$$

where $h$ represents unobserved variables that influence the data-generation process.

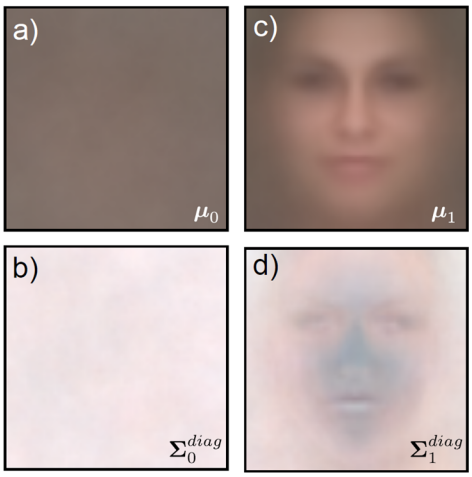



**Figure 7.2** Class conditional density functions for normal model with diagonal covariance. Maximum likelihood fits based on 1000 training examples per class. a) Mean for background data $\boldsymbol{\mu}_0$ (reshaped from $10800 \times 1$ vector to $60 \times 60$ RGB image). b) Reshaped square root of diagonal covariance for background data $\boldsymbol{\Sigma}_0$. c) Mean for face data $\boldsymbol{\mu}_1$ d) Covariance for face data $\boldsymbol{\Sigma}_1$. The background model has little structure: the mean is uniform and the variance is high everywhere. The mean of the face model clearly captures class-specific information. The covariance of the face is larger at the edges of the image which usually contain hair or background.

Figure 2.6: Results (Diagonal Covariance).

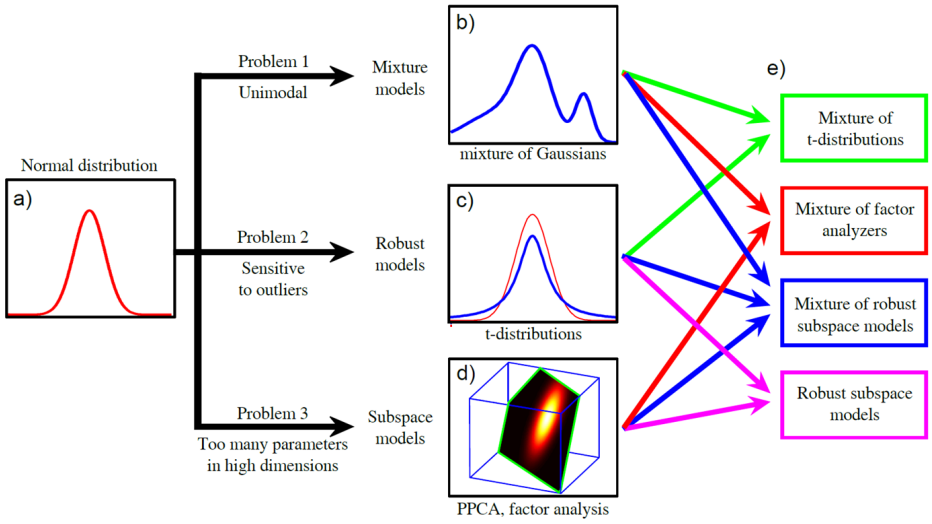



Figure 2.7: The Modeling Procedure for Classification.

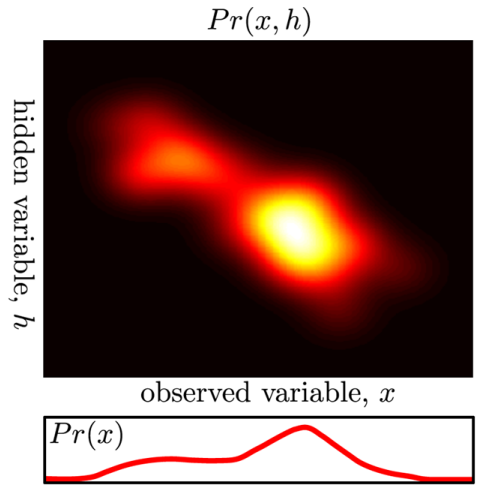



**Figure 7.4** Using hidden variables to help model complex densities. One way to model the density $Pr(x)$ is to consider the joint probability distribution $Pr(x,h)$ between the observed data $x$ and a hidden variable $h$. The density $Pr(x)$ can be considered as the marginalization of (integral over) this distribution with respect to the hidden variable $h$. As we manipulate the parameters $\boldsymbol{\theta}$ of this joint distribution, the marginal changes and the agreements with the observed data $\{x_i\}_{i=1}^I$ increases or decreases. Sometimes it is easier to fit the distribution in this indirect way than to directly manipulate $Pr(x)$.

Figure 2.8: Hidden (Latent) Variables.

### 2.5.1 Expectation-Maximization Algorithm

The Expectation-Maximization (EM) algorithm is an iterative method for finding maximum likelihood (ML) estimates of parameters in models where the data likelihood involves latent variables. EM alternates between two steps:

- **E-Step:** Estimate the posterior probabilities of the latent variables (responsibilities) given the current parameter estimates.
- **M-Step:** Update the model parameters to maximize the expected log-likelihood under the responsibilities computed in the E-Step.

The EM algorithm optimizes the log-likelihood:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{I} \log \int \Pr(x_i, h_i \mid \boldsymbol{\theta})\, dh_i,$$

where $h_i$ represents the latent variables.

#### Lower Bound

The algorithm defines a lower bound on the log-likelihood:

$$B(\{q_i(h_i)\}, \boldsymbol{\theta}) = \sum_{i=1}^{I} \int q_i(h_i) \log \frac{\Pr(x_i, h_i \mid \boldsymbol{\theta})}{q_i(h_i)}\, dh_i,$$

which satisfies:

$$B(\{q_i(h_i)\}, \boldsymbol{\theta}) \leq \sum_{i=1}^{I} \log \int \Pr(x_i, h_i \mid \boldsymbol{\theta})\, dh_i.$$

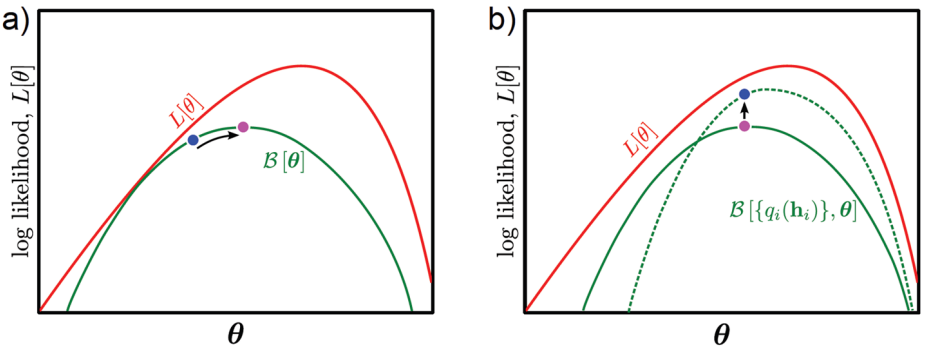



Figure 2.9: Lower Bound in EM.

The EM algorithm alternates between the following steps:

- **E-Step:** Maximize the bound with respect to the distributions $q_i(h_i)$:

$$q_i(h_i) = \Pr(h_i \mid x_i, \boldsymbol{\theta}^{[t]}).$$

- **M-Step:** Maximize the bound with respect to the parameters $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^{[t+1]} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{I} \int q_i(h_i) \log \Pr(x_i, h_i \mid \boldsymbol{\theta})\, dh_i.$$

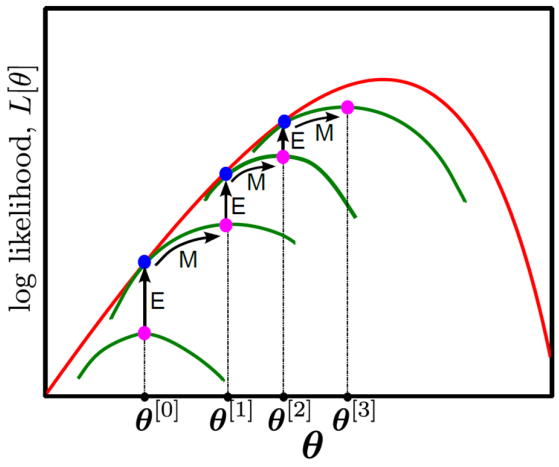



Figure 2.10: E-Step and M-Step Visual.

### 2.5.2 Mixture of Gaussians (MoG)

**Definition 2.5.2.** A **Mixture of Gaussians (MoG)** is a probabilistic model that represents the density $\Pr(x)$ as a weighted sum of $K$ Gaussian components:

$$\Pr(x \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k),$$

where:

- $\pi_k$: Mixing coefficients, with $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k \geq 0$.
- $\mu_k$: Mean of the $k$-th Gaussian.
- $\Sigma_k$: Covariance matrix of the $k$-th Gaussian.

The parameters of the MoG model are $\boldsymbol{\theta} = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$. These parameters can be estimated using maximum likelihood, but directly maximizing the log-likelihood is challenging because of the presence of a logarithm over the sum:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{I} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k).$$

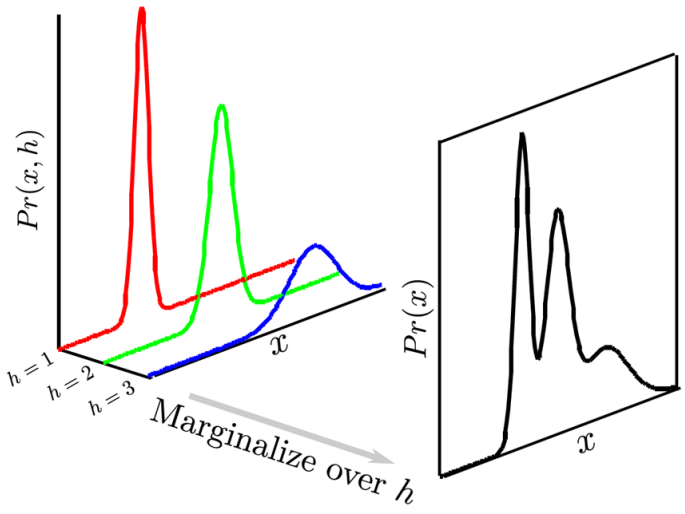



Figure 2.11: Mixture of Gaussian as Marginalization Visual Illustration.

**MoG as a Marginalization**

To simplify the optimization, MoG can be interpreted as a marginalization over latent variables. Introduce a discrete latent variable $h \in \{1, \ldots, K\}$ that indicates the component from which $x$ was drawn. The joint distribution can be expressed as:

$$\Pr(x, h \mid \boldsymbol{\theta}) = \Pr(x \mid h, \boldsymbol{\theta}) \Pr(h \mid \boldsymbol{\theta}),$$

where:

$$\Pr(x \mid h = k, \boldsymbol{\theta}) = \mathcal{N}(x \mid \mu_k, \Sigma_k), \quad \Pr(h = k \mid \boldsymbol{\theta}) = \pi_k.$$

By marginalizing out $h$, we recover the density $\Pr(x)$:

$$\Pr(x \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \Pr(x \mid h = k, \boldsymbol{\theta}) \Pr(h = k \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k).$$

This interpretation allows us to incorporate latent variable $h$ into the model, which:

- Provides a method for generating data from MoG:
  1. Sample $h \sim \text{Cat}(\boldsymbol{\pi})$.
  2. Sample $x \sim \mathcal{N}(\mu_h, \Sigma_h)$.
- Gives a clear interpretation of $h$ as the component membership for each data point $x$.

**Expectation-Maximization for MoG**

The EM algorithm is used to learn the parameters $\boldsymbol{\theta} = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^K$ from training data $\{x_1, \ldots, x_I\}$. The two steps are:

**E-Step:** Compute the responsibilities, $r_{ik}$, which represent the probability that data point $x_i$ belongs to component $k$:

$$r_{ik} = \Pr(h_i = k \mid x_i, \boldsymbol{\theta}^{[t]}) = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}.$$

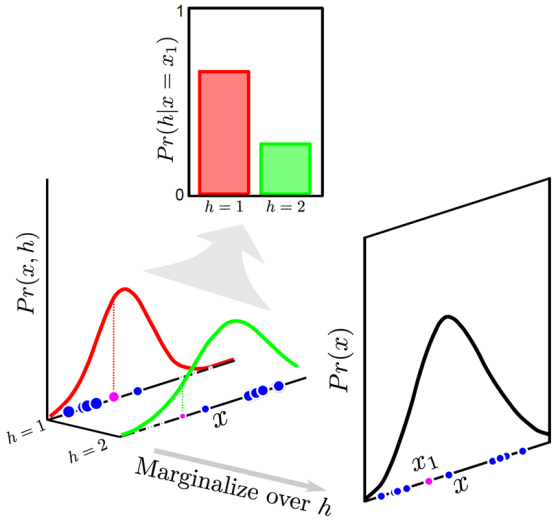



Figure 2.12: E-Step Illustration.

**M-Step:** Update the parameters to maximize the expected log-likelihood using the responsibilities:

$$\pi_k^{[t+1]} = \frac{1}{I}\sum_{i=1}^I r_{ik},$$

$$\mu_k^{[t+1]} = \frac{\sum_{i=1}^I r_{ik} x_i}{\sum_{i=1}^I r_{ik}},$$

$$\Sigma_k^{[t+1]} = \frac{\sum_{i=1}^I r_{ik}(x_i - \mu_k^{[t+1]})(x_i - \mu_k^{[t+1]})^\top}{\sum_{i=1}^I r_{ik}}.$$

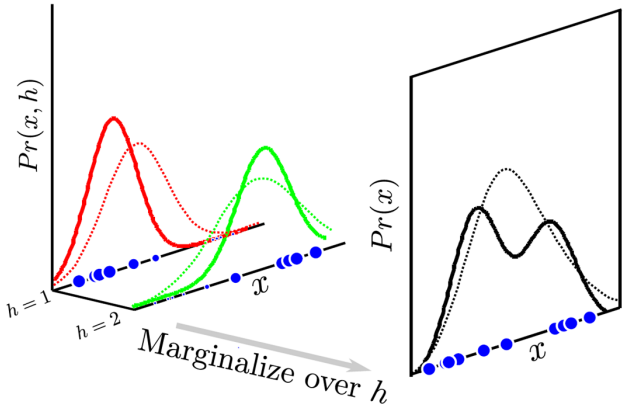



Figure 2.13: M-Step Illustration.

These steps are repeated until convergence, ensuring that the log-likelihood increases at each iteration. EM is particularly suited for MoG because it simplifies the estimation process by alternating between assigning data points to components (E-Step) and updating parameters (M-Step).

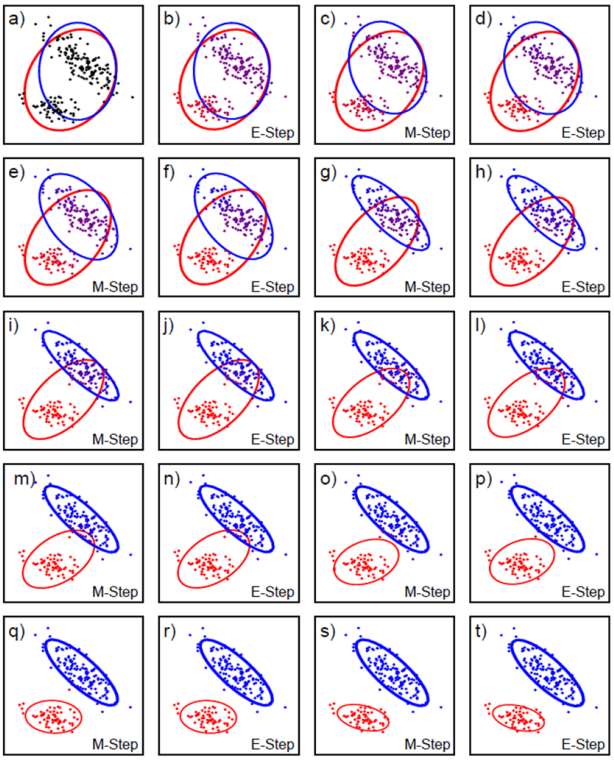



Figure 2.14: Iteration Example of MoG.

### 2.5.3 Student t-Distributions

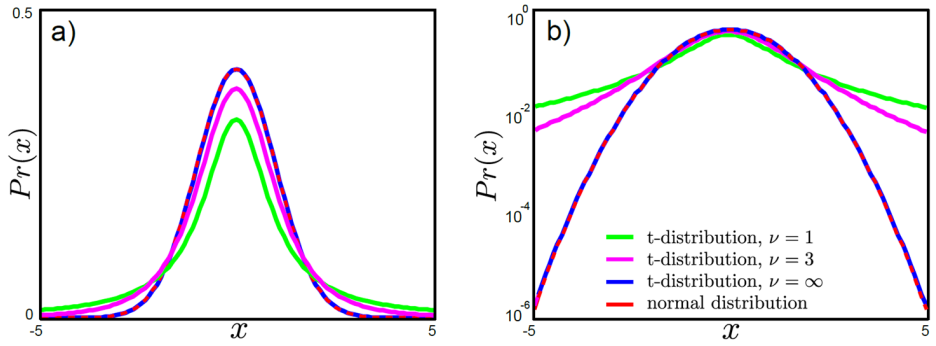



Figure 2.15: Student t-distribution.

**Remark.** *The Student t-distribution is a robust alternative to the Gaussian distribution, designed to handle outliers better due to its heavier tails. Unlike the Gaussian distribution, which can be significantly affected by extreme data points, the t-distribution reduces their impact on the model.*

**Definition 2.5.3.** The univariate Student t-distribution is defined as:

$$\Pr(x) = \text{Stud}_x[\mu, \sigma^2, \nu] = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi\sigma^2}\,\Gamma\left(\frac{\nu}{2}\right)}\left(1 + \frac{(x-\mu)^2}{\nu\sigma^2}\right)^{-\frac{\nu+1}{2}},$$

where:

- $\mu$: Mean of the distribution.
- $\sigma^2$: Scale parameter (variance).
- $\nu$: Degrees of freedom, controlling the heaviness of the tails.

The multivariate Student t-distribution generalizes this to higher dimensions:

$$\Pr(\mathbf{x}) = \text{Stud}_{\mathbf{x}}[\mu, \Sigma, \nu] = \frac{\Gamma\left(\frac{\nu+D}{2}\right)}{(\nu\pi)^{D/2}|\Sigma|^{1/2}\,\Gamma\left(\frac{\nu}{2}\right)}\left(1 + \frac{(\mathbf{x}-\mu)^\top\Sigma^{-1}(\mathbf{x}-\mu)}{\nu}\right)^{-\frac{\nu+D}{2}},$$

where $D$ is the dimensionality of $\mathbf{x}$.

### t-Distribution as a Marginalization

The t-distribution can be expressed as a marginalization of a Gaussian distribution and a Gamma-distributed latent variable $h$:

$$\Pr(x \mid h) = \text{Norm}_x[\mu, \Sigma/h], \quad \Pr(h) = \text{Gam}_h\left[\frac{\nu}{2}, \frac{\nu}{2}\right].$$

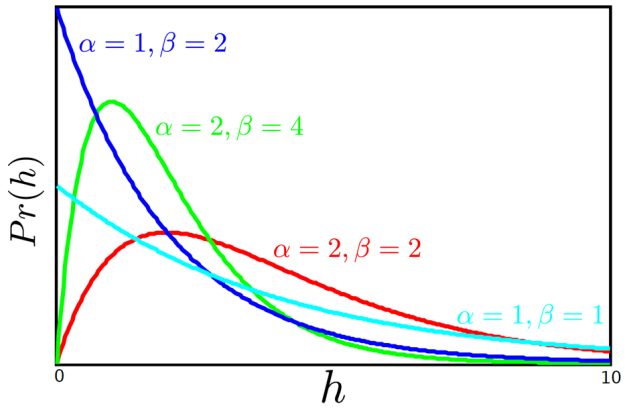



Figure 2.16: Gamma Distribution Illustration.

By integrating out $h$, we recover the t-distribution:

$$\Pr(x) = \int \Pr(x \mid h) \Pr(h)\, dh = \text{Stud}_x[\mu, \Sigma, \nu].$$

Key insights:

- This formulation provides a method to sample from the t-distribution by first sampling $h \sim$ Gamma and then sampling $x \sim$ Normal.
- The latent variable $h$ has a clear interpretation: it acts as a weight, inversely affecting the variance for each data point, allowing the t-distribution to adapt to outliers.
- The t-distribution can be viewed as an infinite mixture of Gaussians with the same mean but different variances.

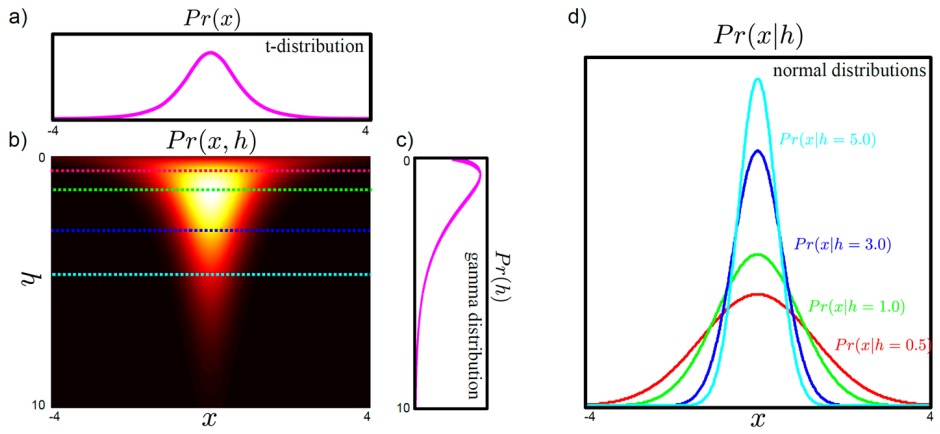



Figure 2.17: Student t-distribujtion Visual Illustration.

### EM Algorithm for t-Distributions

The EM algorithm is used to estimate the parameters $\boldsymbol{\theta} = \{\mu, \Sigma, \nu\}$ of the t-distribution from data $\{x_1, \ldots, x_I\}$. The steps are:

**E-Step:** Compute the posterior distribution over the latent variable $h_i$ for each data point:

$$q_i(h_i) = \Pr(h_i \mid x_i, \boldsymbol{\theta}^{[t]}) = \text{Gam}_h \left[ \frac{\nu + D}{2}, \frac{\nu + (\mathbf{x}_i - \mu)^\top \Sigma^{-1} (\mathbf{x}_i - \mu)}{2} \right].$$

Extract expectations:

$$\mathbb{E}[h_i] = \frac{\nu + D}{\nu + (\mathbf{x}_i - \mu)^\top \Sigma^{-1}(\mathbf{x}_i - \mu)},$$
$$\mathbb{E}[\log h_i] = \psi\left(\frac{\nu + D}{2}\right) - \log\left(\frac{\nu + (\mathbf{x}_i - \mu)^\top \Sigma^{-1}(\mathbf{x}_i - \mu)}{2}\right),$$

where $\psi$ is the digamma function.

**M-Step:** Update the parameters using the expectations from the E-Step:

$$\mu^{[t+1]} = \frac{\sum_{i=1}^{I} \mathbb{E}[h_i]\mathbf{x}_i}{\sum_{i=1}^{I} \mathbb{E}[h_i]},$$
$$\Sigma^{[t+1]} = \frac{\sum_{i=1}^{I} \mathbb{E}[h_i](\mathbf{x}_i - \mu^{[t+1]})(\mathbf{x}_i - \mu^{[t+1]})^\top}{\sum_{i=1}^{I} \mathbb{E}[h_i]}.$$

The degrees of freedom $\nu$ cannot be updated in closed form, but can be optimized using grid search or other numerical methods.
The EM algorithm iterates between these steps until convergence, ensuring a robust fit to data, particularly in the presence of outliers.

### 2.5.4 Factor Analysis

**Definition 2.5.4. Factor Analysis** is a statistical method used to model the covariance structure of high-dimensional data. It provides a compromise between:

- Using a full covariance matrix (rich model but with many parameters).
- Using a diagonal covariance matrix (simpler but no modeling of correlations).

The model assumes that the data $\mathbf{x}$ is generated as:

$$\Pr(\mathbf{x}) = \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}, \boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \boldsymbol{\Sigma}],$$

where:

- $\boldsymbol{\Phi}$: Factor loading matrix, capturing full covariance in a lower-dimensional subspace.
- $\boldsymbol{\Sigma}$: Diagonal covariance matrix, accounting for residual variances.
- $\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \boldsymbol{\Sigma}$: Full covariance structure, combining subspace and diagonal components.

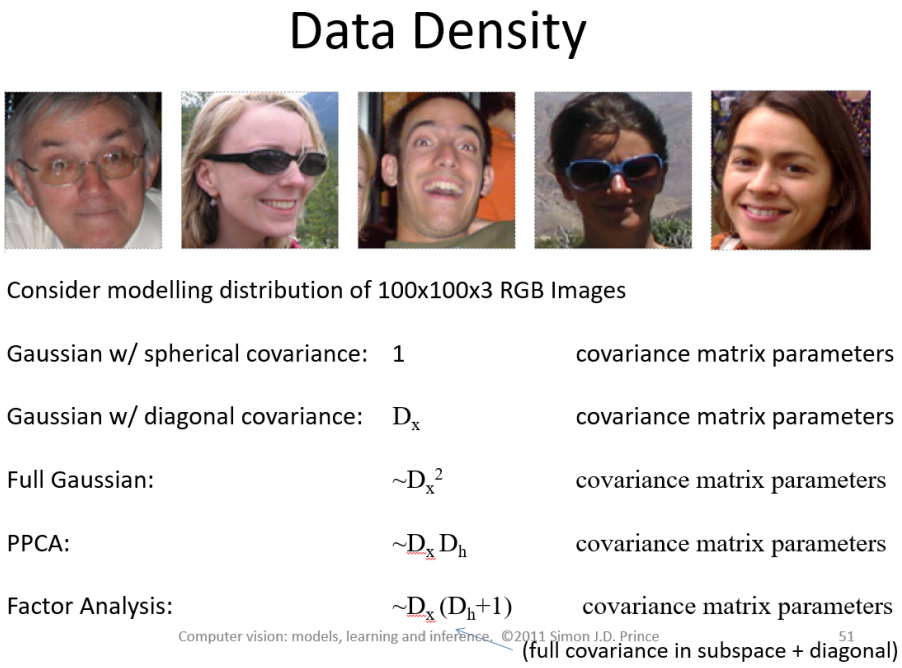



Figure 2.18: Data Density of Different Models.

### Sampling from a Factor Analyzer

Factor Analysis involves hidden latent variables $\mathbf{h}$ that explain the observed data. The generative process is:

$$\begin{aligned}\Pr(\mathbf{x} \mid \mathbf{h}) &= \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}, \boldsymbol{\Sigma}], \\ \Pr(\mathbf{h}) &= \text{Norm}_{\mathbf{h}}[\mathbf{0}, \mathbf{I}].\end{aligned}$$

This setup provides a method for sampling:

1. Sample $\mathbf{h} \sim \text{Norm}[\mathbf{0}, \mathbf{I}]$.
2. Compute $\mathbf{x} = \boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h} + \epsilon$, where $\epsilon \sim \text{Norm}[\mathbf{0}, \boldsymbol{\Sigma}]$.

### EM Algorithm for Factor Analysis

The EM algorithm is used to estimate the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Phi}, \boldsymbol{\Sigma}\}$.

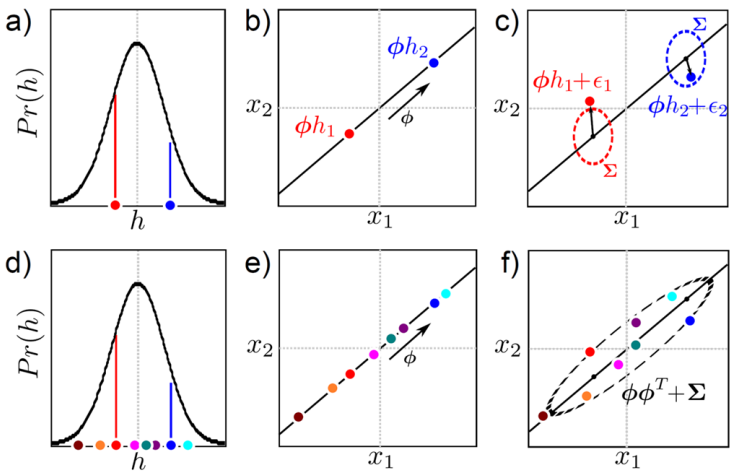



Figure 2.19: Sampling from Factor Analyzer.

**E-Step:** Compute the posterior over the latent variables $\mathbf{h}_i$ for each data point:

$$q_i(\mathbf{h}_i) = \Pr(\mathbf{h}_i \mid \mathbf{x}_i, \boldsymbol{\theta}^{[t]}) = \text{Norm}_{\mathbf{h}_i}[(\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}+\mathbf{I})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}), (\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}+\mathbf{I})^{-1}].$$

From this posterior, compute the expectations:

$$\begin{aligned}\mathbb{E}[\mathbf{h}_i] &= (\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}+\mathbf{I})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}),\\ \mathbb{E}[\mathbf{h}_i\mathbf{h}_i^\top] &= (\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}+\mathbf{I})^{-1} + \mathbb{E}[\mathbf{h}_i]\mathbb{E}[\mathbf{h}_i]^\top.\end{aligned}$$

**M-Step:** Optimize the parameters $\boldsymbol{\theta}$ with respect to the expected log-likelihood:

$$\mathbb{E}[\log \Pr(\mathbf{x}_i, \mathbf{h}_i \mid \boldsymbol{\theta})] = -\frac{1}{2}\sum_{i=1}^{I}\left(d\log(2\pi) + \log|\boldsymbol{\Sigma}| + (\mathbf{x}_i-\boldsymbol{\mu}-\boldsymbol{\Phi}\mathbb{E}[\mathbf{h}_i])^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}-\boldsymbol{\Phi}\mathbb{E}[\mathbf{h}_i])\right).$$

Update equations are:

$$\begin{aligned}\boldsymbol{\mu}^{[t+1]} &= \frac{1}{I}\sum_{i=1}^{I}\mathbf{x}_i,\\ \boldsymbol{\Phi}^{[t+1]} &= \left(\sum_{i=1}^{I}(\mathbf{x}_i-\boldsymbol{\mu})\mathbb{E}[\mathbf{h}_i]^\top\right)\left(\sum_{i=1}^{I}\mathbb{E}[\mathbf{h}_i\mathbf{h}_i^\top]\right)^{-1},\\ \boldsymbol{\Sigma}^{[t+1]} &= \frac{1}{I}\sum_{i=1}^{I}\text{diag}\left[(\mathbf{x}_i-\boldsymbol{\mu})(\mathbf{x}_i-\boldsymbol{\mu})^\top - \boldsymbol{\Phi}\mathbb{E}[\mathbf{h}_i](\mathbf{x}_i-\boldsymbol{\mu})^\top\right].\end{aligned}$$

These steps are iterated until the parameters converge, effectively modeling both shared structure (via $\mathbf{\Phi}$) and residual variance (via $\mathbf{\Sigma}$).

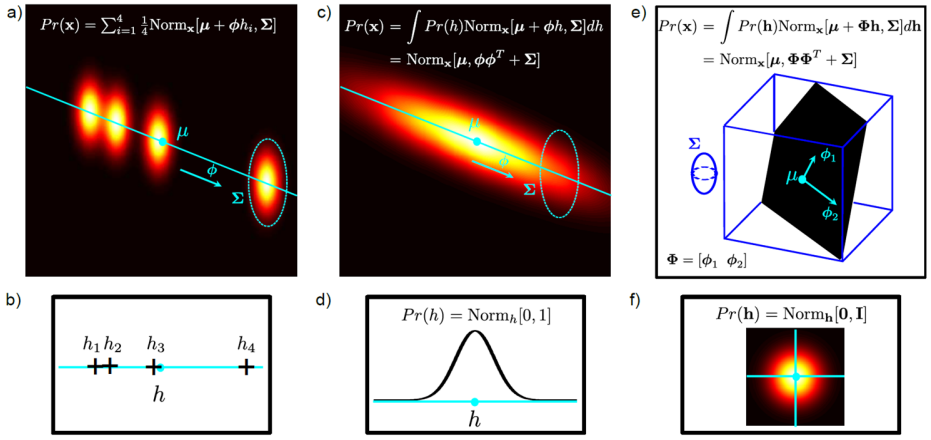



Figure 2.20: Factor Analysis vs. Mixture of Gaussian.

**Comparison with Other Models**

- Factor Analysis models full covariance in a subspace $\mathbf{\Phi}$ while modeling residual variance with a diagonal matrix $\mathbf{\Sigma}$.

- Compared to MoG, it is not suitable for modeling multimodal distributions but is more compact for high-dimensional data.

### 2.5.5 EM Algorithm in Detail (Extension)

The Expectation-Maximization (EM) algorithm is a powerful iterative method for optimizing log-likelihood functions in models with latent variables. It alternates between two steps:

- **E-Step (Expectation):** Compute the posterior distribution over the latent variables given the observed data and current parameters.

- **M-Step (Maximization):** Maximize the expected complete-data log-likelihood with respect to the model parameters.

### Problem Formulation

The EM algorithm optimizes a cost function of the form:

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{I} \log \int \Pr(x_i, h_i \mid \theta)\, dh_i,$$

where $x_i$ are observed variables, $h_i$ are latent variables, and $\theta$ represents the model parameters. The algorithm defines a lower bound on the log-likelihood and iteratively increases it:

$$\mathcal{B}(\{q_i(h_i)\}, \theta) = \sum_{i=1}^{I} \int q_i(h_i) \log \frac{\Pr(x_i, h_i \mid \theta)}{q_i(h_i)}\, dh_i.$$

### E-Step

In the E-step, we maximize the bound $\mathcal{B}$ with respect to the distributions $q_i(h_i)$, holding the parameters $\theta$ fixed. This leads to:

$$q_i(h_i) = \Pr(h_i \mid x_i, \theta^{[t]}).$$

This update minimizes the Kullback-Leibler (KL) divergence between $q_i(h_i)$ and the true posterior $\Pr(h_i \mid x_i, \theta)$. The KL divergence ensures that:

$$q_i(h_i) \log \frac{\Pr(h_i \mid x_i, \theta)}{q_i(h_i)} = 0.$$

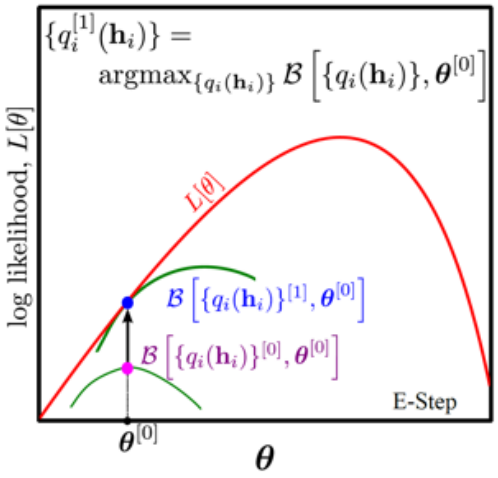



Figure 2.21: E-Step.

### M-Step

In the M-step, we maximize the bound $\mathcal{B}$ with respect to the parameters $\theta$, holding $q_i(h_i)$ fixed. The objective becomes:

$$\theta^{[t+1]} = \arg\max_{\theta} \sum_{i=1}^{I} \int q_i(h_i) \log \Pr(x_i, h_i \mid \theta)\, dh_i.$$

The parameters are updated to maximize the expected complete-data log-likelihood, where the expectation is taken with respect to $q_i(h_i)$ from the E-step.

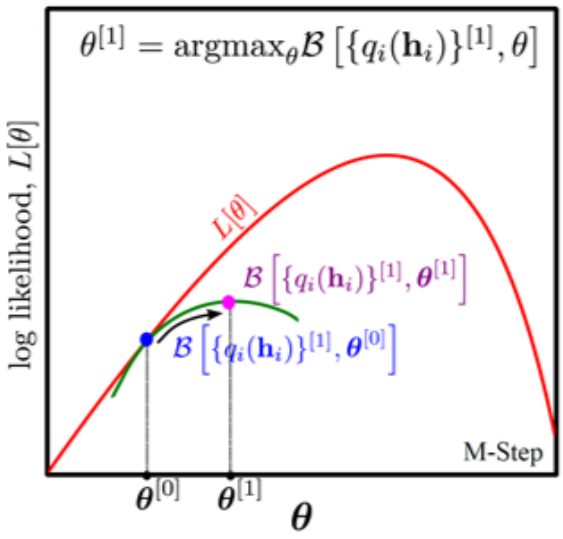



Figure 2.22: M-Step.

### Key Insights

- The lower bound $\mathcal{B}$ ensures that each iteration of the EM algorithm improves or maintains the log-likelihood of the observed data.
- The iterative process alternates between refining the posterior over latent variables and optimizing the model parameters, converging to a local optimum.
- Jensen's inequality provides the theoretical foundation for the lower bound:

  $$\log \mathbb{E}[y] \leq \mathbb{E}[\log y],$$

  ensuring $\mathcal{B} \leq \log \Pr(x \mid \theta)$.

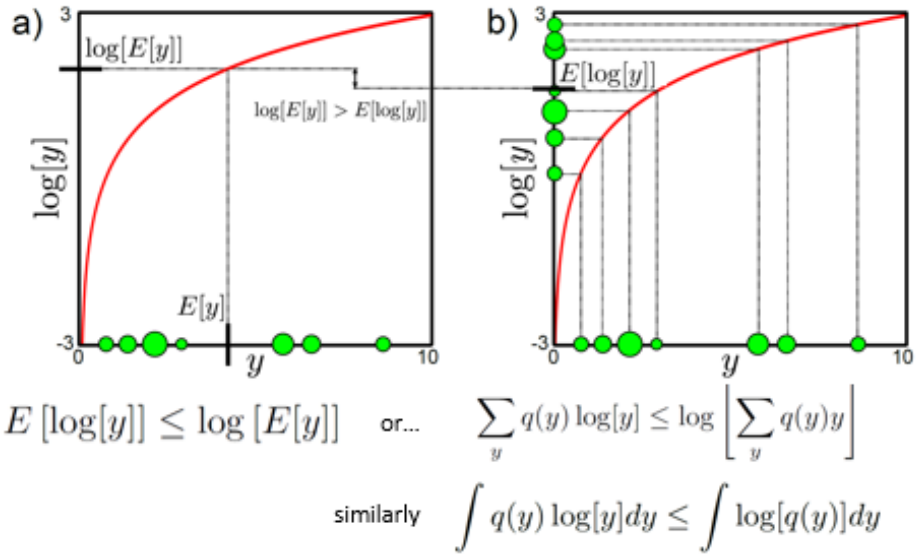



Figure 2.23: Jensen's Inequality in EM.

**Detailed E-Step and M-Step Derivations**

**E-Step:** The distribution $q_i(h_i)$ is chosen to be:

$$q_i(h_i) = \Pr(h_i \mid x_i, \theta^{[t]}) = \frac{\Pr(x_i, h_i \mid \theta^{[t]})}{\Pr(x_i \mid \theta^{[t]})}.$$

**M-Step:** The parameter update involves:

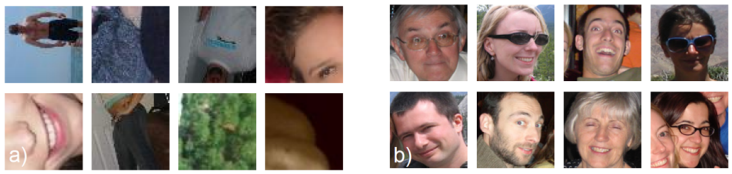
$$\theta^{[t+1]} = \arg\max_{\theta} \sum_{i=1}^{I} \int \Pr(h_i \mid x_i, \theta^{[t]}) \log \Pr(x_i, h_i \mid \theta)\, dh_i.$$

**Convergence and Properties**

- The EM algorithm guarantees that the log-likelihood $\log \Pr(x \mid \theta)$ does not decrease at each iteration.
- The process converges to a stationary point, which may be a local maximum.
- Missing parts of the argument include showing the validity of the lower bound and correctness of the E-step and M-step updates.

## 2.6 Applications of Complex Densities

- **Face Detection:** Classify face and non-face images using MoG or $t$-distributions.

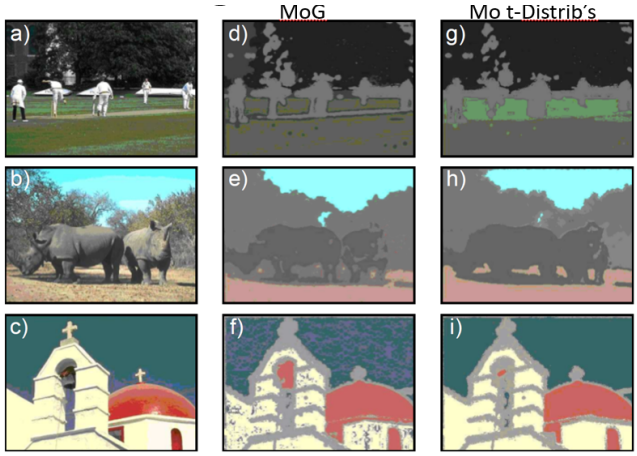



| | Color | Grayscale | Equalized |
|---|---|---|---|
| Single Gaussian | 76% | 79% | 80% |
| Mixture of 10 Gaussians | 81% | 85% | 89% |

(not a very realistic model – face detection really done using discriminative methods)

Figure 2.24: Example1: Face Detection.

- **Segmentation:** Use mixture models to partition images into meaningful regions.

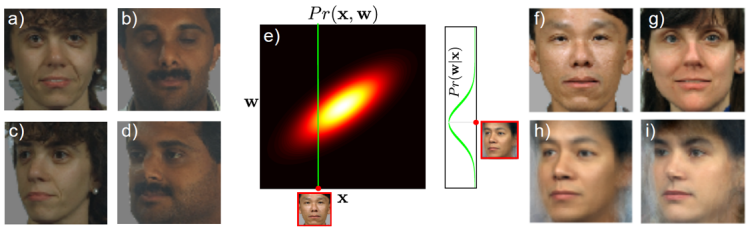



$$Pr(w_n) = \text{Cat}_{w_n}[\boldsymbol{\lambda}]$$
$$Pr(\mathbf{x}_i|w_i = k) = \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]$$

Sfikas *et al.* (2007) ©IEEE 2007

Figure 2.25: Example2: Segmentation.

- **Pose Regression:** Learn a factor analysis model for predicting pose from image data.




| Training data | Learning | Inference |
|---|---|---|
| Frontal faces x<br>Non-frontal faces w | Fit factor analysis model to joint distribution of x,w | Given new x, infer w<br>(cond. dist of normal) |

Figure 2.26: Example3: Pose Regression.

# Chapter 3

# Regression

## 3.1 Linear Regression

### 3.1.1 Model Formulation

**Definition 3.1.1.** Linear regression assumes that each dimension of the world $w$ is predicted separately, concentrating on modeling a univariate world state $w$. The model defines a normal distribution over $w$ given $x$:

$$\Pr(w \mid x, \theta) = \mathcal{N}(w \mid \phi_0 + \phi^\top x, \sigma^2),$$

where:

- $\phi_0 + \phi^\top x$: The mean is a linear function of the data $x$.
- $\sigma^2$: The variance is constant across all data points.

### 3.1.2 Compact Notation

**Remark.** *To make the notation simpler:*

- *Attach a constant* $1$ *to the start of each data vector* $x$*:*

$$x_i \leftarrow [1 \; x_i^\top]^\top.$$

- *Include the offset as the first element of the parameter vector* $\phi$*:*

$$\phi \leftarrow [\phi_0 \; \phi^\top]^\top.$$

*This reformulates the model as:*

$$\Pr(w \mid x, \theta) = \mathcal{N}(w \mid \phi^\top x, \sigma^2).$$

### 3.1.3 Combining Equations for the Dataset

**Remark.** *Considering the entire dataset with design matrix* $X = [x_1, x_2, \ldots, x_N]^\top$ *and target vector* $w = [w_1, w_2, \ldots, w_N]^\top$*, the joint probability becomes:*

$$\Pr(w \mid X, \theta) = \mathcal{N}(w \mid X^\top \phi, \sigma^2 I).$$

### 3.1.4 Learning Parameters with Maximum Likelihood

**Remark.** *The parameters* $\phi$ *and* $\sigma^2$ *are learned by maximizing the log-likelihood:*

$$\hat{\theta} =_{\phi,\sigma^2} \log \Pr(w \mid X, \theta).$$

*Substituting the likelihood expression:*

$$\hat{\phi}, \hat{\sigma}^2 =_{\phi,\sigma^2} \left[ -\frac{N}{2} \log(2\pi) - \frac{N}{2} \log \sigma^2 - \frac{1}{2\sigma^2} (w - X^\top \phi)^\top (w - X^\top \phi) \right].$$

*Taking derivatives, setting them to zero, and rearranging:*

- *The parameter estimate:*

$$\hat{\phi} = (X^\top X)^{-1} X^\top w.$$

- *The variance estimate:*

$$\hat{\sigma}^2 = \frac{(w - X^\top \hat{\phi})^\top (w - X^\top \hat{\phi})}{N}.$$

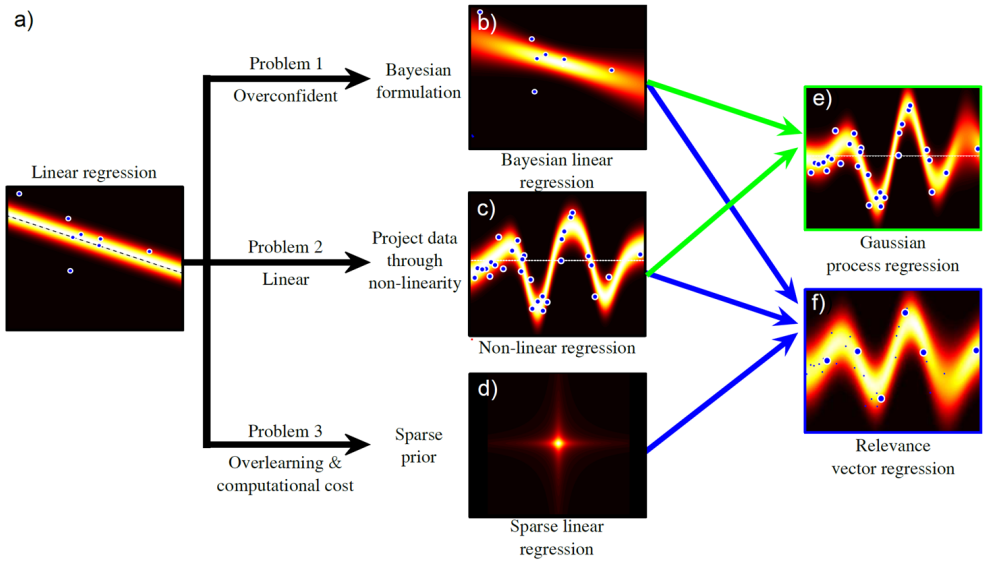



Figure 3.1: Regression Models.

## 3.2 Bayesian Regression

### 3.2.1 Posterior Distribution

**Definition 3.2.1.** Bayesian regression combines prior knowledge about the parameters with observed data using Bayes' rule. The likelihood is:

$$\Pr(w \mid X, \theta) = \mathcal{N}(w \mid X^\top \phi, \sigma^2 I),$$

and the prior over the parameters $\phi$ is:

$$\Pr(\phi) = \mathcal{N}(\phi \mid 0, \sigma_p^2 I).$$

By Bayes' rule, the posterior distribution is:

$$\Pr(\phi \mid X, w) = \frac{\Pr(w \mid X, \phi)\Pr(\phi)}{\Pr(w \mid X)}.$$

Substituting the prior and likelihood, the posterior distribution becomes:

$$\Pr(\phi \mid X, w) = \mathcal{N}\left(\phi \mid A^{-1}\frac{1}{\sigma^2}Xw, A^{-1}\right),$$

where:

$$A = \frac{1}{\sigma^2}XX^\top + \frac{1}{\sigma_p^2}I.$$

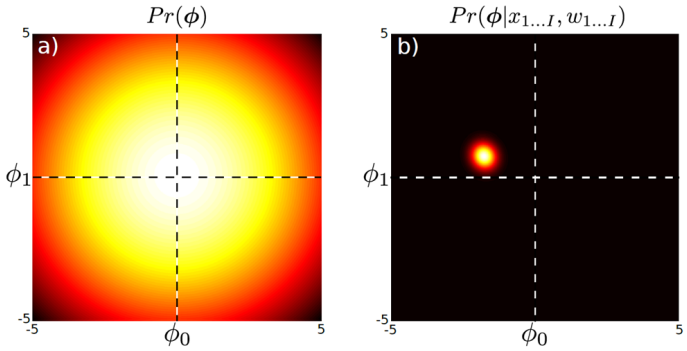



Figure 3.2: Posterior Distribution Over Parameters.

### 3.2.2 Predictive Distribution

**Remark.** *For a new data point $x_*$, the predictive distribution of $w_*$ is obtained by marginalizing over the posterior:*

$$\Pr(w_* \mid x_*, X, w) = \int \Pr(w_* \mid x_*, \phi) \Pr(\phi \mid X, w)\, d\phi.$$

*Substituting the posterior distribution:*

$$\Pr(w_* \mid x_*, X, w) = \mathcal{N}\left(w_* \mid \frac{1}{\sigma^2} x_*^\top A^{-1} X w, x_*^\top A^{-1} x_* + \sigma^2\right).$$

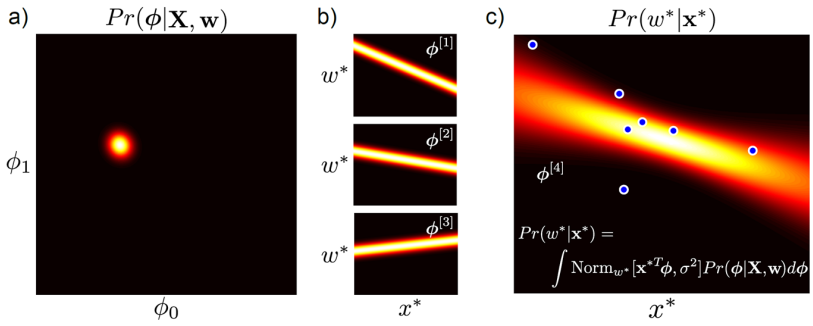



Figure 3.3: Bayesian Regression Inference.

### 3.2.3 Practical Issues

**Remark.** *In high-dimensional spaces, directly inverting $A$ can be computationally prohibitive. Using the matrix inversion lemma:*

$$A^{-1} = \frac{1}{\sigma_p^2} I - \frac{\sigma_p^2}{\sigma^2} \left(X^\top X + \frac{\sigma^2}{\sigma_p^2} I\right)^{-1} X^\top X,$$

*the inversion complexity is reduced. This reformulation ensures the inverses involve $(l \times l)$ matrices rather than $(d \times d)$ matrices, where $l$ is the number of data points.*

### 3.2.4 Fitting Variance

**Remark.** *The variance $\sigma^2$ can be estimated by optimizing the marginal likelihood:*

$$\Pr(w \mid X, \sigma^2) = \int \Pr(w \mid X, \phi, \sigma^2) \Pr(\phi)\, d\phi.$$

*Substituting the likelihood and prior, the marginal likelihood becomes:*

$$\Pr(w \mid X, \sigma^2) = \mathcal{N}\left(w \mid 0, \sigma_p^2 X^\top X + \sigma^2 I\right).$$

## 3.3 Non-Linear Regression

### 3.3.1 Model Formulation

**Definition 3.3.1.** Non-linear regression extends linear regression by transforming the input $x$ into a higher-dimensional feature space $z$, allowing the model to capture non-linear relationships:

$$\Pr(w_i \mid x_i, \theta) = \mathcal{N}(w_i \mid \phi^\top z_i, \sigma^2),$$

where:

- $z_i = f(x_i)$: The transformed feature vector derived from $x_i$ using a set of non-linear basis functions $f$.
- $\phi^\top z_i$: The mean of the Gaussian distribution is a linear combination of the transformed features.
- $\sigma^2$: The variance is assumed constant across all data points.

### 3.3.2 Examples of Basis Functions

**Remark.** *Different choices of basis functions allow the model to capture various types of non-linearities in the data. Some common examples include:*

- ***Polynomial Basis:***

  $$z_i = [1, x_i, x_i^2, \ldots, x_i^K]^\top.$$

  *This creates a feature space where the relationship between $x_i$ and $w_i$ can be modeled as a polynomial.*

- ***Radial Basis Functions (RBFs):***

  $$z_i = \left[\exp\left(-\frac{(x_i - c_1)^2}{\lambda}\right), \exp\left(-\frac{(x_i - c_2)^2}{\lambda}\right), \ldots\right]^\top,$$

  *where:*

  - *$c_k$: Centers of the RBFs.*
  - *$\lambda$: Controls the width (or spread) of the RBF.*

- ***Arc Tan Functions:***

  $$z_i = [\arctan(\lambda(x_i - \alpha_1)), \arctan(\lambda(x_i - \alpha_2)), \ldots]^\top,$$

  *where:*

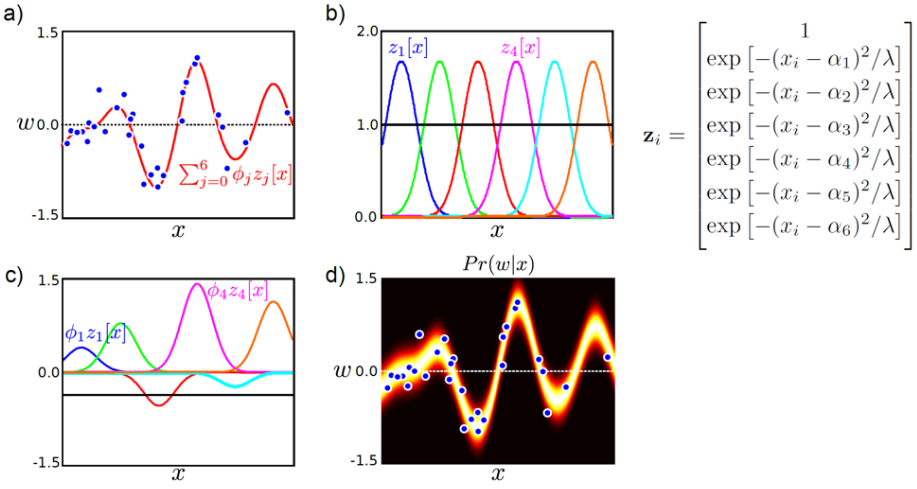



Figure 3.4: Radial Basis Functions.

– $\alpha_k$: *Parameters shifting the inputs for each basis function.*

– $\lambda$: *Controls the non-linearity scaling.*

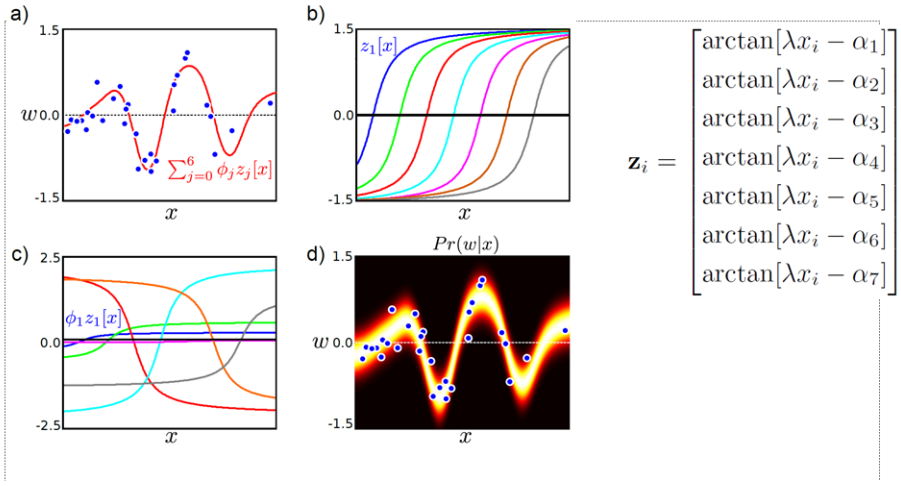



Figure 3.5: Arc Tan Functions.

### 3.3.3 Maximum Likelihood Estimation

**Remark.** *Similar to linear regression, the parameters $\phi$ and $\sigma^2$ are estimated by maximizing the likelihood:*

$$\hat{\phi} = (ZZ^\top)^{-1}Zw,$$

*where $Z$ is the design matrix containing the transformed features $z_i$.*

*The variance is estimated as:*

$$\hat{\sigma}^2 = \frac{(w - Z^\top\phi)^\top(w - Z^\top\phi)}{I},$$

*where $I$ is the number of data points.*

## 3.4 Kernel Regression and Gaussian Processes

### 3.4.1 The Kernel Trick

**Definition 3.4.1.** The kernel trick enables the use of high-dimensional (or infinite-dimensional) feature spaces without explicitly computing the transformed feature vectors $z_i$. Instead, it defines a kernel function that computes the dot product $z_i^\top z_j$ directly:

$$K(x_i, x_j) = z_i^\top z_j.$$

This allows regression models to operate implicitly in the feature space without constructing $z$.

### 3.4.2 Example Kernels

**Remark.** *Commonly used kernel functions include:*

- ***Linear Kernel:***

  $$K(x_i, x_j) = x_i^\top x_j.$$

  *Equivalent to linear regression.*

- ***Polynomial Kernel of Degree $p$:***

  $$K(x_i, x_j) = \left(x_i^\top x_j + 1\right)^p.$$

  *This kernel captures polynomial relationships between inputs.*

- ***Radial Basis Function (RBF) or Gaussian Kernel:***

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{\lambda^2}\right).$$

*This kernel corresponds to infinite-dimensional feature spaces and is widely used for its flexibility.*

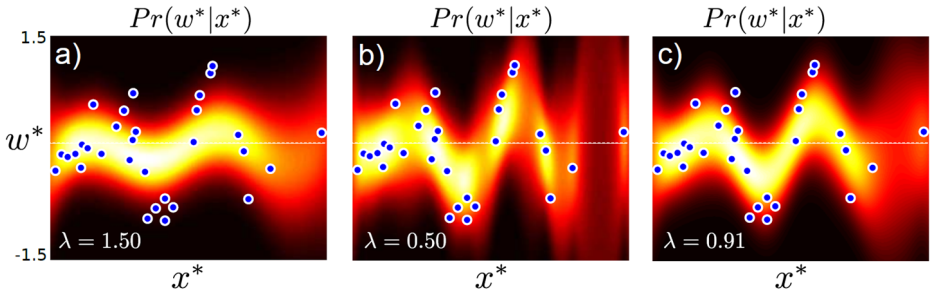



**Figure 8.9** Gaussian process regression using an RBF kernel a) When the length scale parameter $\lambda$ is large, the function is too smooth. b) For small values of the length parameter the model does not successfully interpolate between the examples. c) The regression using the maximum likelihood length scale parameter is neither too smooth nor disjointed.

Figure 3.6: RBF Kernel Fits.

### 3.4.3 Gaussian Processes

**Definition 3.4.2.** Gaussian processes (GPs) generalize Bayesian regression by placing a prior over functions $w(x)$:

$$w(x) \sim \mathcal{GP}(m(x), k(x, x')),$$

where:

- $m(x)$: Mean function, often assumed to be zero.
- $k(x, x')$: Kernel function, defining the covariance between any two points $x$ and $x'$.

The kernel function $k(x, x')$ specifies the smoothness and generalization properties of the GP.

### 3.4.4 Gaussian Process Regression

**Remark.** *In Gaussian process regression, the predictive distribution for a new data point $x_*$ is:*

$$\Pr(w_* \mid x_*, X, w) = \mathcal{N}\left(\mu_*, \sigma_*^2\right),$$

*where:*

$$\mu_* = K(x_*, X)[K(X, X) + \sigma^2 I]^{-1} w,$$
$$\sigma_*^2 = K(x_*, x_*) - K(x_*, X)[K(X, X) + \sigma^2 I]^{-1} K(X, x_*).$$

*Here:*

- $K(X, X)$*: Covariance matrix for the training points.*
- $K(x_*, X)$*: Covariances between the test point* $x_*$ *and training points.*
- $K(x_*, x_*)$*: Variance at* $x_*$*.*

### 3.4.5 Fitting Variance

**Remark.** *The model variance* $\sigma^2$ *is estimated by maximizing the marginal likelihood:*

$$\Pr(w \mid X, \sigma^2) = \mathcal{N}\left(w \mid 0, \sigma_p^2 K(X, X) + \sigma^2 I\right).$$

*Non-linear optimization techniques are used to optimize* $\sigma^2$ *and other kernel hyperparameters.*

## 3.5 Sparse Linear Regression

### 3.5.1 Model Formulation

**Definition 3.5.1.** Sparse linear regression encourages sparsity in the parameter vector $\phi$ by applying a prior that penalizes non-zero coefficients. The prior is modeled as a product of independent $t$-distributions:

$$\Pr(\phi) = \prod_{d=1}^{D} \text{Student-}t(\phi_d \mid 0, 1, \nu),$$

where:

- $\nu$: Degrees of freedom, controlling the sparsity level.
- $t(\phi_d)$: $t$-distribution encourages sparsity by penalizing large values of $\phi_d$.

### 3.5.2 Posterior Distribution

**Remark.** *Using Bayes' rule, the posterior distribution of* $\phi$ *is given by:*

$$\Pr(\phi \mid X, w, \sigma^2) \propto \Pr(w \mid X, \phi, \sigma^2) \Pr(\phi),$$

*where:*

- $\Pr(w \mid X, \phi, \sigma^2)$*: Likelihood from linear regression.*
- $\Pr(\phi)$*: Sparsity-inducing prior.*

*However, the $t$-distribution prior is not conjugate to the Gaussian likelihood, so the posterior cannot be computed in closed form.*

### 3.5.3 Reformulation with Hidden Variables

**Remark.** *To make progress, the $t$-distribution prior is expressed as a marginal of a joint distribution:*

$$\Pr(\phi) = \int \prod_{d=1}^{D} \mathcal{N}(\phi_d \mid 0, 1/h_d) \mathit{Gamma}(h_d \mid \nu/2, \nu/2)\, dh_d,$$

*where:*

- $h_d$*: Hidden variable controlling the precision (inverse variance) of each parameter $\phi_d$.*
- $\mathit{Gamma}(h_d \mid \nu/2, \nu/2)$*: Gamma distribution acting as a prior over $h_d$.*

*This reformulation allows the posterior to be expressed in terms of $\phi$ and the hidden variables $h_d$.*

### 3.5.4 Approximation and Training

**Remark.** *The marginal likelihood becomes:*

$$\Pr(w \mid X, \sigma^2) \propto \int \mathcal{N}(w \mid X^\top \phi, \sigma^2) \mathcal{N}(\phi \mid 0, H^{-1}) \prod_{d=1}^{D} \mathit{Gamma}(h_d \mid \nu/2, \nu/2)\, d\phi\, dH,$$

*where $H = \mathit{diag}(h_1, \ldots, h_D)$. While this integral cannot be computed exactly, it can be approximated by maximizing with respect to $H$:*

$$\Pr(w \mid X, \sigma^2) \approx \max_H \mathcal{N}(w \mid 0, X^\top H^{-1} X + \sigma^2 I) \prod_{d=1}^{D} \mathit{Gamma}(h_d \mid \nu/2, \nu/2).$$

### 3.5.5 Model Fitting

**Remark.** *To fit the model, the following updates are alternated:*

- ***Update Hidden Variables $h_d$:***

  $$h_d^{new} = \frac{1 - h_d \Sigma_{dd} + \nu}{\mu_d^2 + \nu},$$

  *where $\mu_d$ and $\Sigma_{dd}$ are the mean and diagonal of the covariance for $\phi_d$.*

- ***Update Variance $\sigma^2$:***

$$(\sigma^2)^{new} = \frac{1}{D - \sum_d (1 - h_d \Sigma_{dd})} (w - X^\top \mu)^\top (w - X^\top \mu),$$

*where:*

$$\mu = \frac{1}{\sigma^2} A^{-1} X w, \quad \Sigma = A^{-1}, \quad A = \frac{1}{\sigma^2} X X^\top + H.$$

### 3.5.6 Sparsity Effect

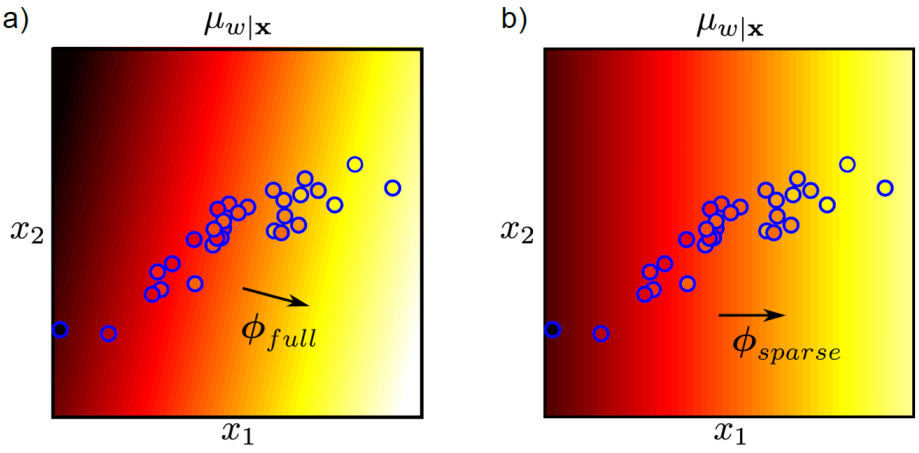



Figure 3.7: Sparse Linear Regression Sparsity Effect.

**Remark.** *After fitting, some hidden variables $h_d$ become very large, resulting in their corresponding parameters $\phi_d$ being effectively zero. This sparsifies the model by eliminating uninformative dimensions in the data.*

## 3.6 Dual Linear Regression

### 3.6.1 Dual Regression

**Definition 3.6.1.** In dual linear regression, the parameter vector $\phi$ is represented as a weighted sum of the training data points:

$$\phi = X\psi,$$

where:

- $\psi$: Dual variables, one for each training example.

- $X$: Design matrix containing all training data points.

This formulation allows the regression problem to be expressed in terms of dot products between data points, $X^\top X$.

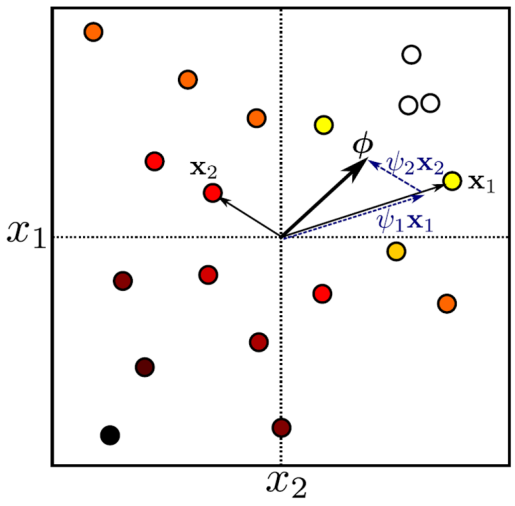



Figure 3.8: Dual Linear Regression Visual.

### 3.6.2 Dual Formulation

**Remark.** *Starting from the original linear regression model:*

$$\Pr(w \mid X, \theta) = \mathcal{N}(w \mid X^\top \phi, \sigma^2 I),$$

*substituting $\phi = X\psi$, the dual form becomes:*

$$\Pr(w \mid X, \theta) = \mathcal{N}(w \mid X^\top X\psi, \sigma^2 I).$$

*The dual representation simplifies the problem to operations involving $X^\top X$ rather than explicit manipulation of $\phi$.*

### 3.6.3 Maximum Likelihood Estimation

**Remark.** *The parameters $\psi$ and $\sigma^2$ are estimated by maximizing the likelihood:*

$$\hat{\psi}, \hat{\sigma}^2 = _{\psi,\sigma^2} \left[ -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log\sigma^2 - \frac{1}{2\sigma^2}(w - X^\top X\psi)^\top (w - X^\top X\psi) \right].$$

*Solving this gives:*

$$\hat{\psi} = (X^\top X)^{-1} X^\top w,$$

$$\hat{\sigma}^2 = \frac{(w - X^\top X\hat{\psi})^\top (w - X^\top X\hat{\psi})}{N}.$$

### 3.6.4 Bayesian Formulation

**Remark.** *In the Bayesian setting, a prior is placed over the dual variables $\psi$:*

$$\Pr(\psi) = \mathcal{N}(\psi \mid 0, \sigma_p^2 I).$$

*Combining this with the likelihood, the posterior over $\psi$ becomes:*

$$\Pr(\psi \mid X, w, \sigma^2) = \mathcal{N}\left(\psi \mid A^{-1}\frac{1}{\sigma^2}X^\top w, A^{-1}\right),$$

*where:*

$$A = \frac{1}{\sigma^2}X^\top X + \frac{1}{\sigma_p^2}I.$$

### 3.6.5 Predictive Distribution

**Remark.** *For a new data point $x_*$, the predictive distribution is:*

$$\Pr(w_* \mid x_*, X, w) = \int \Pr(w_* \mid x_*, \psi) \Pr(\psi \mid X, w)\, d\psi.$$

*This evaluates to:*

$$\Pr(w_* \mid x_*, X, w) = \mathcal{N}\left(w_* \mid \frac{1}{\sigma^2}x_*^\top A^{-1} X^\top w, x_*^\top A^{-1} x_* + \sigma^2\right).$$

### 3.6.6 Key Observations

**Remark.** *Both the maximum likelihood and Bayesian formulations depend only on dot products $X^\top X$. This makes dual linear regression particularly amenable to the **kernel trick**, enabling implicit use of high-dimensional feature spaces without explicitly computing $\phi$.*

## 3.7 Relevance Vector Machines (RVM)

### 3.7.1 Overview

**Definition 3.7.1.** Relevance Vector Machines (RVMs) combine the principles of:

- **Dual representation:** Parameters are represented using weights associated with training data.
- **Sparsity:** Most weights are driven to zero during training, resulting in a model that depends on a small subset of training points (relevance vectors).

This allows RVMs to produce compact, efficient models while maintaining predictive accuracy.

### 3.7.2 Model Formulation

**Remark.** *The RVM assumes the following probabilistic model:*

$$\Pr(w \mid X, \theta) = \mathcal{N}(w \mid X^\top X\psi, \sigma^2 I),$$

*where:*

- $\psi$*: Dual variables, one for each training example.*
- $X$*: Design matrix containing all training examples.*

*The prior over $\psi$ is given by a product of independent $t$-distributions:*

$$\Pr(\psi) = \prod_{i=1}^{I} \textit{Student-}t(\psi_i \mid 0, 1, \nu).$$

### 3.7.3 Learning Parameters

**Remark.** *To learn the parameters, the marginal likelihood is maximized by alternately updating:*

- ***Hidden variables $h_i$:****

  $$h_i^{\textit{new}} = \frac{1 - h_i\Sigma_{ii} + \nu}{\mu_i^2 + \nu},$$

  *where:*

  - $\mu_i$*: Mean of the posterior over $\psi$.*
  - $\Sigma_{ii}$*: Diagonal entry of the posterior covariance matrix.*

- ***Variance $\sigma^2$:***

$$(\sigma^2)^{new} = \frac{1}{I - \sum_i (1 - h_i \Sigma_{ii})} (w - X^\top \mu)^\top (w - X^\top \mu),$$

*where:*

$$\mu = \frac{1}{\sigma^2} A^{-1} X w, \quad \Sigma = A^{-1}, \quad A = \frac{1}{\sigma^2} X^\top X + H.$$

### 3.7.4 Relevance Vectors

**Remark.** *After training, most hidden variables $h_i$ become very large, forcing the corresponding weights $\psi_i$ to zero. The remaining non-zero weights correspond to a sparse subset of the training data, known as **relevance vectors**. This sparsity leads to efficient predictions.*

### 3.7.5 Kernelization

**Remark.** *Similar to dual regression, RVMs depend only on dot products $X^\top X$. This makes RVMs amenable to the **kernel trick**, enabling operations in high-dimensional feature spaces without explicit computation.*

## 3.8 Applications of Regression

- **Body Pose Estimation:** Predict a 55-dimensional pose vector from a 100-dimensional silhouette vector.
- **Dimensionality Reduction:** Use clustering and histograms to reduce complex feature spaces into manageable representations.
- **Shape Context:** Represent shapes as histograms over clustered feature vectors.

# Chapter 4

# Classification

## 4.1 Logistic Regression

### 4.1.1 Model Formulation

**Definition 4.1.1.** Logistic regression models the posterior probability $\Pr(w \mid x)$ for binary classification ($w \in \{0, 1\}$), assuming a Bernoulli distribution over the output:

$$\Pr(w \mid \phi, x) = \text{Bern}_w[\sigma(a)],$$

where:

- $a = \phi_0 + \phi^\top x$ is the linear activation function,
- $\sigma(a) = \frac{1}{1+e^{-a}}$ is the sigmoid function, mapping real numbers $a \in \mathbb{R}$ to probabilities $\sigma(a) \in [0, 1]$.

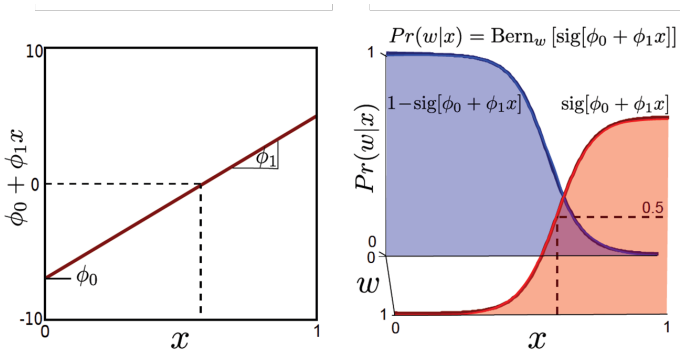



Figure 4.1: Logistic Regression Parameters. $\phi_0$,$\phi_1$

**Remark.** *The full probabilistic model is expressed as:*

$$\Pr(w \mid \phi, x) = (\sigma(a))^w (1 - \sigma(a))^{1-w},$$

*where $\sigma(a)$ is the probability of $w = 1$, and $1 - \sigma(a)$ is the probability of $w = 0$.*

### 4.1.2 Likelihood and Log-Likelihood

**Definition 4.1.2.** Given a dataset $\{x_i, w_i\}_{i=1}^I$, the likelihood function is:

$$\Pr(w \mid X, \phi) = \prod_{i=1}^{I} \sigma(a_i)^{w_i} (1 - \sigma(a_i))^{1-w_i}, \quad a_i = \phi^\top x_i.$$

The log-likelihood simplifies to:

$$\mathcal{L}(\phi) = \sum_{i=1}^{I} \Big[ w_i \log \sigma(a_i) + (1 - w_i) \log(1 - \sigma(a_i)) \Big].$$

**Remark.** *The derivative of the log-likelihood is:*

$$\frac{\partial \mathcal{L}(\phi)}{\partial \phi} = \sum_{i=1}^{I} \Big[ \sigma(a_i) - w_i \Big] x_i,$$

*where $\sigma(a_i)$ represents the model's predicted probability for the $i^{th}$ sample. There is no closed form solution for this, hence we use iterative non-linear optimization. This derivative forms the basis for gradient-based optimization.*

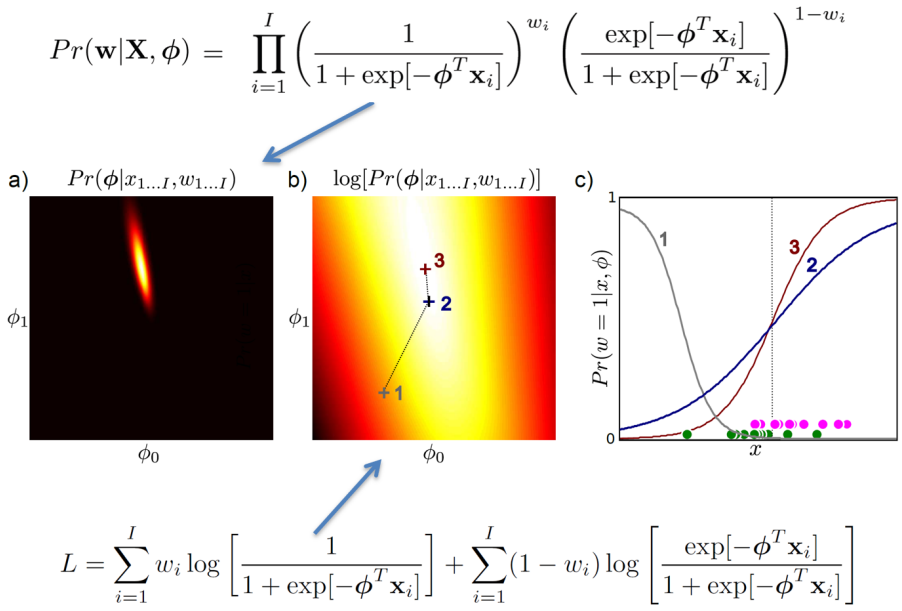



Figure 4.2: Logistic Regression Optimization.

### 4.1.3 Optimization Techniques

- **Gradient Descent:** Updates $\phi$ iteratively:

$$\phi^{[t+1]} = \phi^{[t]} - \eta\nabla\mathcal{L}(\phi^{[t]}),$$

where $\eta$ is the learning rate. Although simple, gradient descent can suffer from inefficiencies, such as zigzagging in narrow valleys.

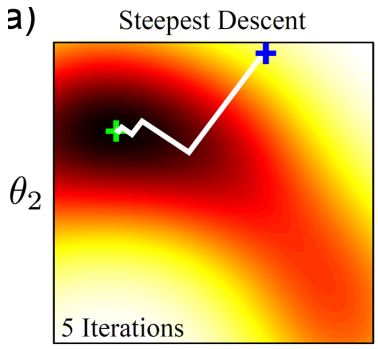



Figure 4.3: Gradient Descent: wConsider standing on a hillsize. Find the steepest direction downhill, and walk in that direction for some distance (line search).

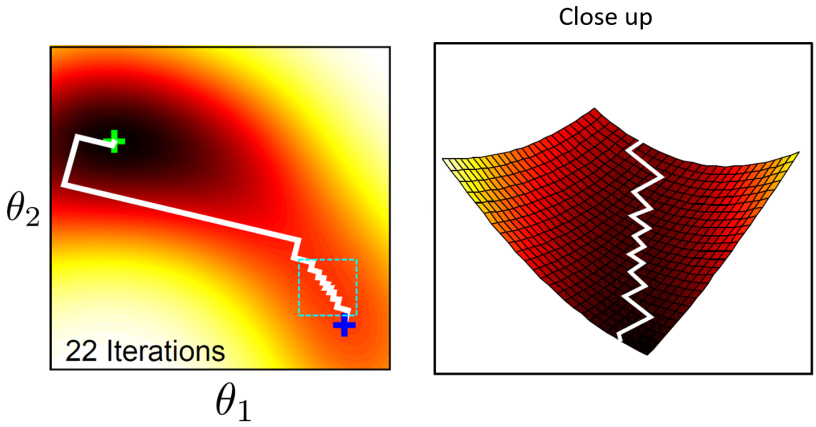



Figure 4.4: Steepest Descent Problems.

- **Line Search:** Adaptively determines the step size $\lambda$ along the gradient direction to minimize the objective:

$$\lambda = \arg\min_{\lambda} f(\phi^{[t]} + \lambda \nabla \mathcal{L}(\phi^{[t]})).$$

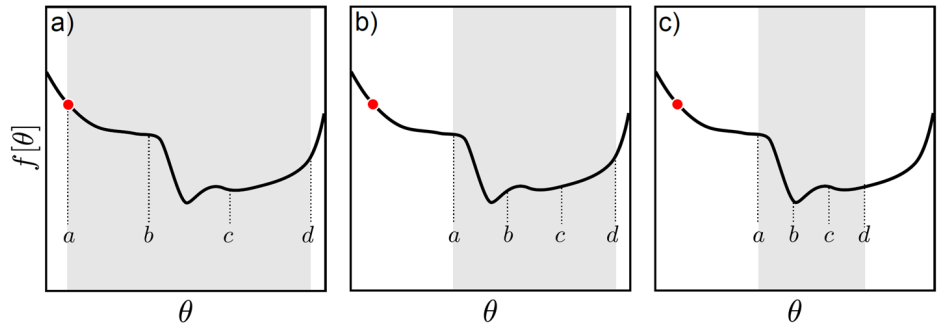



Figure 4.5: Linear Search: Gradually narrow down range.

- **Newton's Method:** Approximates the function locally as quadratic using a second-order Taylor expansion:

$$f(\phi) \approx f(\phi^{[t]}) + (\phi - \phi^{[t]})^\top \nabla f(\phi^{[t]}) + \frac{1}{2}(\phi - \phi^{[t]})^\top H(\phi - \phi^{[t]}),$$

where $H$ is the Hessian matrix:

$$H = \frac{\partial^2 \mathcal{L}(\phi)}{\partial \phi^2}.$$

Newton's update step is:

$$\phi^{[t+1]} = \phi^{[t]} - H^{-1} \nabla \mathcal{L}(\phi^{[t]}).$$

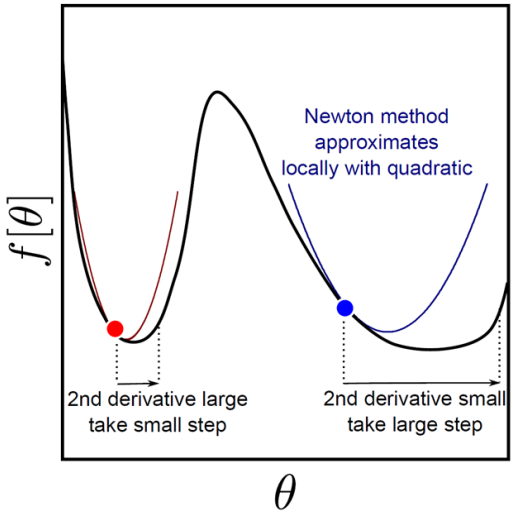



Figure 4.6: Newton's Method for second derivative.

- **Finite Differences:** For numerical approximation of gradients:

$$\frac{\partial f}{\partial \phi_j} \approx \frac{f(\phi + \epsilon e_j) - f(\phi)}{\epsilon},$$

where $e_j$ is the unit vector in the $j^{\text{th}}$ direction and $\epsilon$ is a small step size.

### 4.1.4 Second Derivatives and Newton's Method

**Remark.** *Second derivatives, represented by the Hessian matrix, adjust step sizes in different directions:*

- *Large curvature (high second derivative) implies small steps.*
- *Small curvature (low second derivative) implies large steps.*

*Newton's method leverages this to efficiently move towards the optimum. However, it is computationally expensive due to the need to compute and invert the Hessian matrix.*

### 4.1.5 Decision Boundary

**Definition 4.1.3.** The decision boundary for classification is defined by $\Pr(w = 1 \mid x) = 0.5$, which corresponds to:

$$\phi_0 + \phi^\top x = 0.$$

Data points on either side of this hyperplane are classified into their respective classes. In feature-transformed spaces, this boundary can become non-linear.

## 4.2 Bayesian Logistic Regression

### 4.2.1 Likelihood and Prior

**Definition 4.2.1.** Bayesian logistic regression introduces a probabilistic framework by incorporating a prior over the parameters $\phi$. The likelihood of the data is given by:

$$\Pr(w \mid X, \phi) = \prod_{i=1}^{I} \sigma(a_i)^{w_i} (1 - \sigma(a_i))^{1-w_i}, \quad a_i = \phi^\top x_i,$$

where $\sigma(a)$ is the sigmoid function.

The prior on $\phi$ is typically chosen as a multivariate normal distribution:

$$\Pr(\phi) = \mathcal{N}(\phi \mid 0, \sigma_p^2 I),$$

where $\sigma_p^2$ is the prior variance, controlling the regularization strength.

### 4.2.2 Posterior Distribution

**Definition 4.2.2.** The posterior distribution over the parameters is computed using Bayes' rule:

$$\Pr(\phi \mid X, w) = \frac{\Pr(w \mid X, \phi)\Pr(\phi)}{\Pr(w \mid X)}.$$

Since the likelihood and prior are not conjugate, the posterior does not have a closed-form solution and must be approximated.

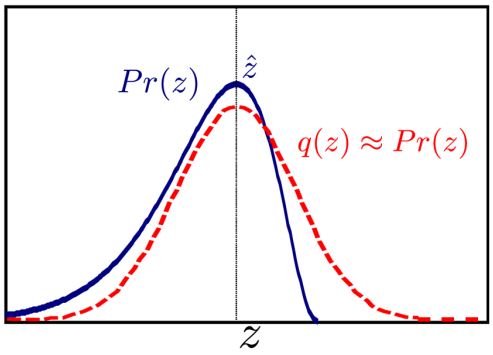



Figure 4.7: Laplace Approximation for Posterior.

### 4.2.3 Laplace Approximation

**Definition 4.2.3.** The Laplace approximation is used to approximate the posterior distribution with a Gaussian. This involves:

- Finding the Maximum a Posteriori (MAP) estimate $\hat{\phi}$ by maximizing the log-posterior and set posterior mean to MAP:

$$\mathcal{L}(\phi) = \sum_{i=1}^{I} \Big[ w_i \log \sigma(a_i) + (1 - w_i) \log(1 - \sigma(a_i)) \Big] - \frac{\phi^\top \phi}{2\sigma_p^2}.$$

- Approximating the posterior as:

$$\Pr(\phi \mid X, w) \approx \mathcal{N}(\phi \mid \hat{\phi}, \Sigma),$$

where the covariance matrix:

$$\Sigma = -\Big( \frac{\partial^2 \mathcal{L}(\phi)}{\partial \phi^2} \Big|_{\phi=\hat{\phi}} \Big)^{-1}.$$

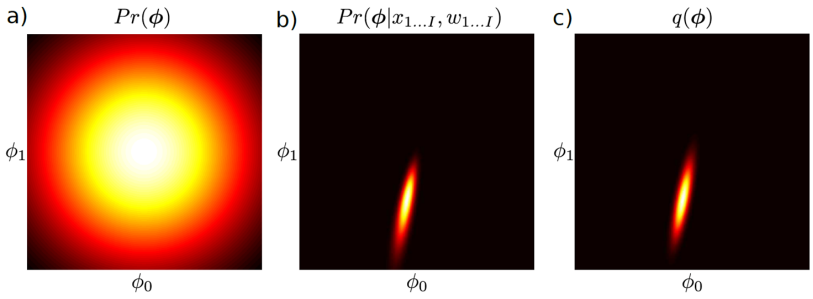



Figure 4.8: Laplace Approximation compared to actual posterior in visual.

### 4.2.4 Inference

**Definition 4.2.4.** For a new data point $x^*$, the predictive distribution is obtained by marginalizing over the posterior:

$$\Pr(w^* \mid x^*, X, w) = \int \Pr(w^* \mid x^*, \phi) \Pr(\phi \mid X, w) d\phi.$$

Using the Laplace approximation, this integral can be simplified. The predictive probability can be re-expressed in terms of the activation $a = \phi^\top x^*$:

$$\Pr(w^* \mid x^*, X, w) \approx \int \sigma(a) \Pr(a) da,$$

where:

$$a \sim \mathcal{N}(a \mid \mu_a, \sigma_a^2),$$

with:

$$\mu_a = \hat{\phi}^\top x^*, \quad \sigma_a^2 = x^{*\top} \Sigma x^*.$$

### 4.2.5 Integral Approximation

**Remark.** *The integral for the predictive probability can be approximated numerically or analytically. A common approximation is:*

$$\int \sigma(a) \mathcal{N}(a \mid \mu_a, \sigma_a^2) da \approx \frac{1}{1 + \exp\left(-\frac{\mu_a}{\sqrt{1 + \pi\sigma_a^2/8}}\right)}.$$

*This approximation provides a closed-form expression for the predictive probability.*

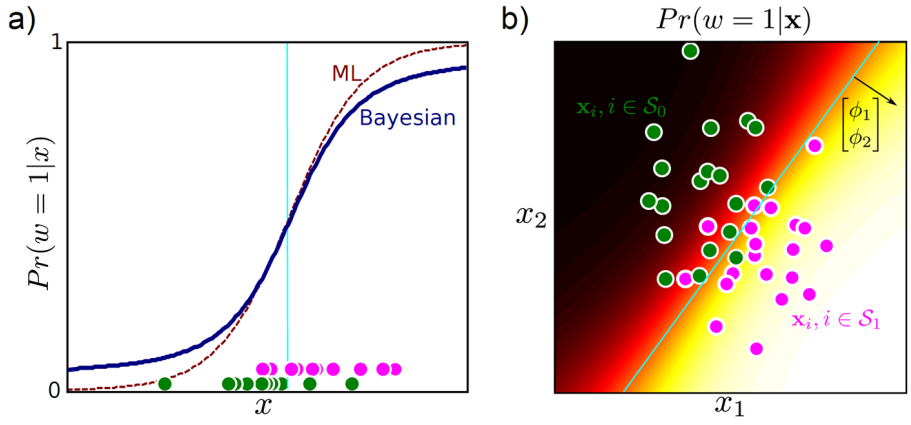



Figure 4.9: Bayesian Solution vs. Maximum Likelihood.

### 4.2.6 Bayesian Solution and Comparison to ML

**Remark.** *Bayesian logistic regression differs from maximum likelihood (ML) in that it incorporates prior knowledge, leading to:*

- *A smoother decision boundary due to regularization from the prior.*
- *Reduced overfitting in low-data regimes.*

*The Bayesian solution can be visualized as providing a more robust posterior distribution compared to point estimates in ML.*

## 4.3 Non-Linear Logistic Regression

### 4.3.1 Feature Transformation

**Definition 4.3.1.** Non-linear logistic regression extends logistic regression by applying a non-linear transformation $\mathbf{z} = f(\mathbf{x})$ to the input features, allowing for non-linear decision boundaries in the input space. The model is formulated as:

$$\Pr(w = 1 \mid \mathbf{x}, \phi) = \text{Bern}_w[\sigma(\phi^\top \mathbf{z})],$$

where:

$$\mathbf{z} = f(\mathbf{x}),$$

is the transformed feature vector, and $\sigma(\cdot)$ is the sigmoid function.

**Remark.** *Common non-linear transformations include:*

- ***Heaviside step function:***
$$z_k = \textit{Heaviside}[\alpha_k^\top \mathbf{x}],$$
*where $\alpha_k$ are parameters defining the step function.*
- ***Arc-tangent function:***
$$z_k = \arctan(\alpha_k^\top \mathbf{x}),$$
*providing smooth transitions.*
- ***Radial Basis Function (RBF):***
$$z_k = \exp\left(-\frac{\|\mathbf{x} - \alpha_k\|^2}{\lambda_0}\right),$$
*which measures the similarity of $\mathbf{x}$ to a center $\alpha_k$.*

### 4.3.2 Model Formulation

**Definition 4.3.2.** After applying the non-linear transformation, the logistic regression model becomes:
$$\Pr(w = 1 \mid \mathbf{x}, \phi) = \sigma(\phi^\top f(\mathbf{x})).$$
The weights $\phi$ are learned using maximum likelihood estimation, similar to linear logistic regression, but now include optimization of the transformation parameters $\alpha$.

### 4.3.3 Optimization

**Remark.** *The log-likelihood for the transformed features is given by:*
$$\mathcal{L}(\phi, \alpha) = \sum_{i=1}^{I} \Big[ w_i \log \sigma(a_i) + (1 - w_i) \log(1 - \sigma(a_i)) \Big],$$
*where:*
$$a_i = \phi^\top f(\mathbf{x}_i).$$
*The gradient and Hessian of the log-likelihood with respect to $\phi$ are:*
$$\frac{\partial \mathcal{L}}{\partial \phi} = \sum_{i=1}^{I} \Big[ \sigma(a_i) - w_i \Big] \frac{\partial a_i}{\partial \phi},$$
$$\frac{\partial^2 \mathcal{L}}{\partial \phi^2} = -\sum_{i=1}^{I} \Big[ \sigma(a_i)(1 - \sigma(a_i)) \Big] \frac{\partial a_i}{\partial \phi} \frac{\partial a_i^\top}{\partial \phi}.$$
*Similarly, derivatives with respect to the transformation parameters $\alpha$ are computed and jointly optimized with $\phi$.*

### 4.3.4 Kernel Logistic Regression

**Definition 4.3.3.** Kernel logistic regression uses the kernel trick to implicitly compute in high-dimensional feature spaces without explicitly transforming $\mathbf{x}$. The kernel function is:

$$K(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i)^\top f(\mathbf{x}_j).$$

The model becomes:

$$\Pr(w = 1 \mid \mathbf{x}, \phi) = \sigma\left(\sum_{i=1}^{I} \phi_i K(\mathbf{x}_i, \mathbf{x})\right),$$

where $\phi_i$ are the weights associated with the kernel function.

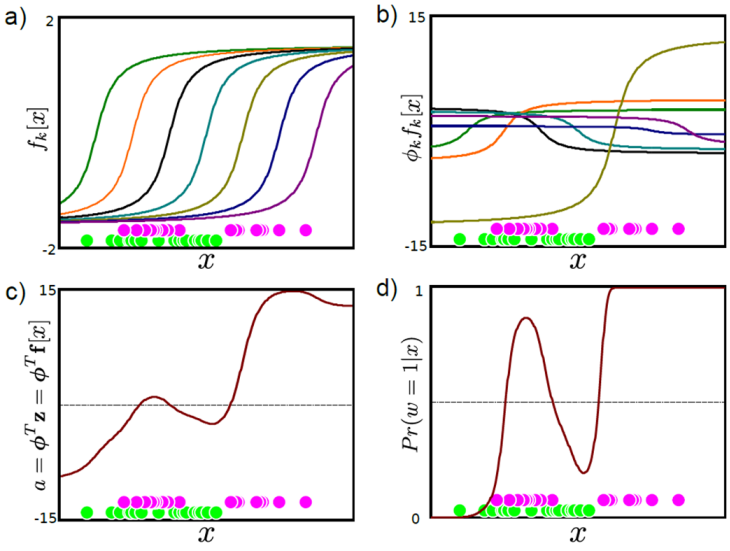



Figure 4.10: Non-linear Logistic Regression in 1D.

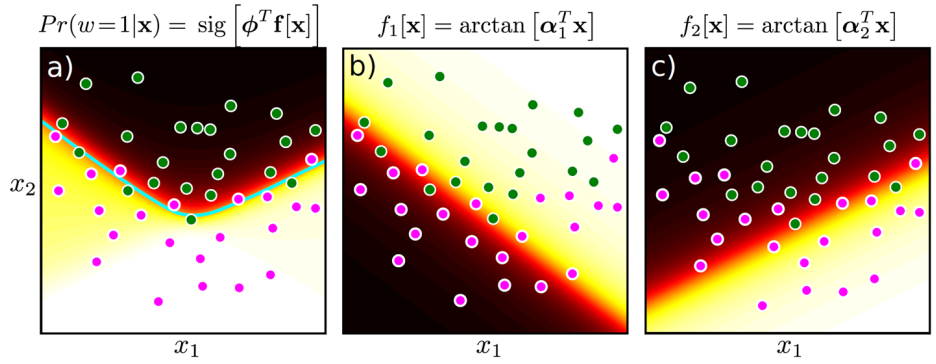



Figure 4.11: Non-linear Logistic Regresion in 2D.

### 4.3.5 1D and 2D Examples

**Remark.** *In one-dimensional examples, transformed features can generate complex decision boundaries by combining basis functions. For two-dimensional examples, transformations like* $\arctan(\alpha_k^\top \mathbf{x})$ *can introduce curved decision boundaries to separate non-linearly separable data.*

## 4.4 Kernelization and Gaussian Process Classification

### 4.4.1 Kernel Logistic Regression

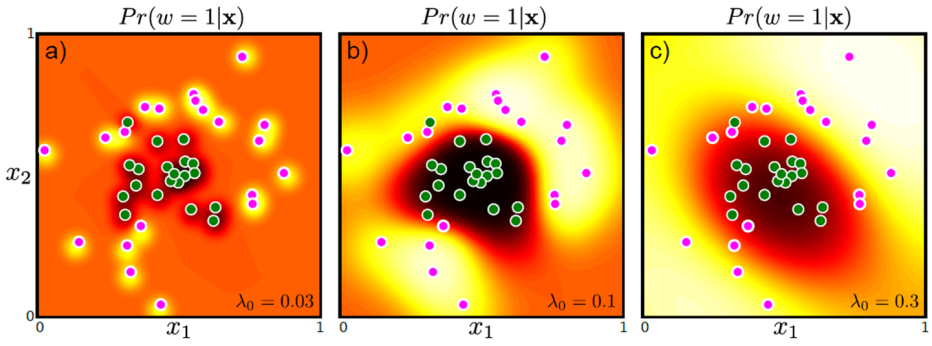



Figure 4.12: Kernel Logistic Regression with different $\lambda_0$.

**Definition 4.4.1.** Kernel logistic regression allows for efficient computation in high-dimensional or infinite-dimensional feature spaces by utilizing a kernel function:

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\lambda_0^2}\right),$$

where $\lambda_0$ is the kernel bandwidth parameter.

**Remark.** *The dual representation of logistic regression expresses the parameters as a linear combination of training points:*

$$\phi = X\psi,$$

*where $X$ is the design matrix, and $\psi$ represents the dual parameters.*

*The likelihood function in terms of the dual parameters $\psi$ is:*

$$\Pr(w \mid X, \psi) = \prod_{i=1}^{I} \text{Bern}_{w_i}\left[\sigma\left(\psi^\top X^\top x_i\right)\right].$$

**Definition 4.4.2** (Kernel Trick)**.** The kernel trick modifies the computations to depend only on inner products in the feature space:

$$\Pr(w \mid X, \psi) = \prod_{i=1}^{I} \text{Bern}_{w_i}\left[\sigma\left(\psi^\top K(x_i, x_j)\right)\right],$$

where $K(x_i, x_j)$ is the kernel matrix.

The derivatives of the log-likelihood with respect to the dual parameters are:

$$\frac{\partial L}{\partial \psi} = -\sum_{i=1}^{I} \left[\sigma(a_i) - w_i\right] K(x_i, x_j),$$

and the second derivatives are:

$$\frac{\partial^2 L}{\partial \psi^2} = -\sum_{i=1}^{I} \sigma(a_i)\left[1 - \sigma(a_i)\right] K(x_i, x_j) K(x_i, x_j)^\top.$$

### 4.4.2 Gaussian Process Classification

**Definition 4.4.3** (Gaussian Process Classification)**.** In Gaussian process classification, the prior over functions is defined as a Gaussian process:

$$f(x) \sim \mathcal{GP}(0, K(x, x')),$$

where $K(x, x')$ is the kernel function representing covariance between points.

The posterior is obtained by updating the prior with observed data using Bayes' rule:

$$\Pr(f \mid X, w) \propto \Pr(w \mid f) \Pr(f).$$

**Remark.** *The predictive probability for Gaussian process classification is:*

$$\Pr(w = 1 \mid x) = \int \sigma(f(x)) \Pr(f \mid X, w) df.$$

*Since the posterior* $\Pr(f \mid X, w)$ *is non-Gaussian, approximation methods such as:*

- ***Laplace approximation****: Approximates the posterior as a Gaussian around the mode.*
- ***Variational inference****: Minimizes the divergence between the true posterior and an approximate distribution.*

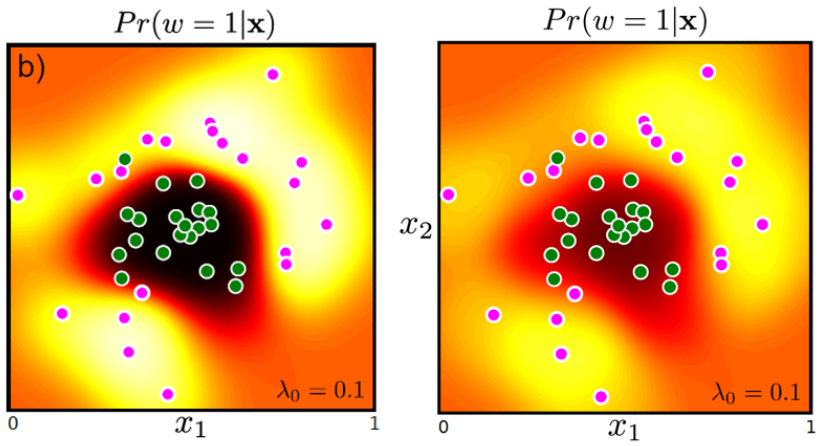



Figure 4.13: ML vs. Bayesian. (Bayesian case is known as Gaussian process classification.)

**Remark.** *Gaussian process classification provides a Bayesian framework for classification, yielding uncertainty estimates for predictions. The comparison between Maximum Likelihood (ML) and Bayesian methods highlights the additional flexibility and interpretability provided by the Bayesian approach.*

## 4.5 Incremental Fitting, Boosting, and Trees

### 4.5.1 Incremental Fitting

**Definition 4.5.1.** Incremental fitting is a technique where classifiers are added one at a time to iteratively improve the accuracy of the model. The decision function at stage $K$ is:

$$a_i = \phi_0 + \sum_{k=1}^{K} \phi_k f[x_i, \xi_k],$$

where:

- $\phi_0$ is the bias term.
- $\phi_k$ are the coefficients.
- $f[x_i, \xi_k]$ represents the feature transformation parameterized by $\xi_k$.
- $\xi_k$ is arc tan function or radial basis function.

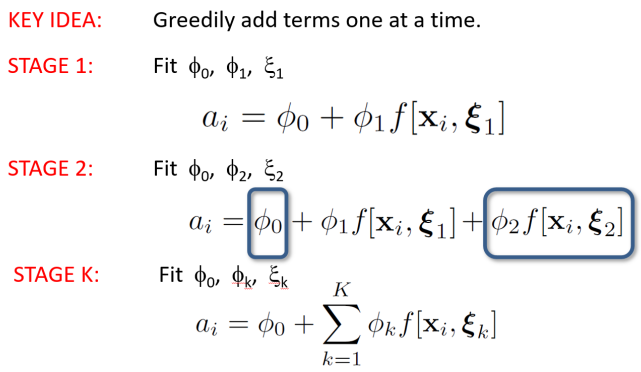



Figure 4.14: Incremental Fitting Stages.

**Theorem 4.5.2** (Convergence of Incremental Fitting)**.** *Under suitable conditions on the choice of $f[x, \xi]$, the incremental fitting procedure converges to a solution that minimizes the loss function.*

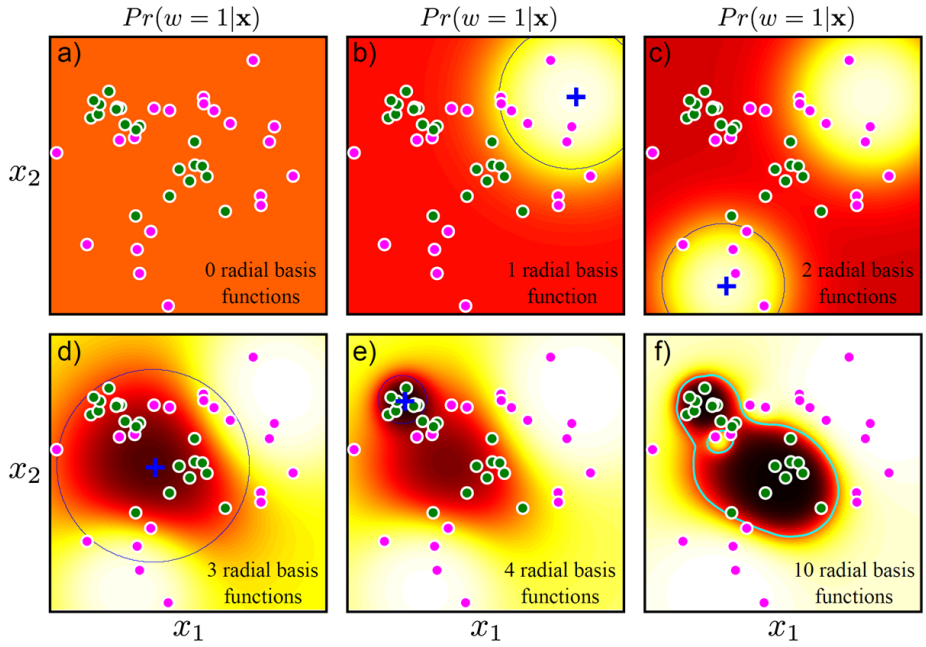



Figure 4.15: Incremental Fitting Visual with different RBF.

**Example 4.5.3.** Consider fitting with radial basis functions (RBF):

$$f[x, \xi] = \exp\left(-\frac{\|x - \alpha_k\|^2}{2\lambda_0^2}\right),$$

where $\alpha_k$ and $\lambda_0$ are parameters. At each stage, a new RBF term is added to better fit the data, as shown in Figure **??**.

### 4.5.2 Boosting

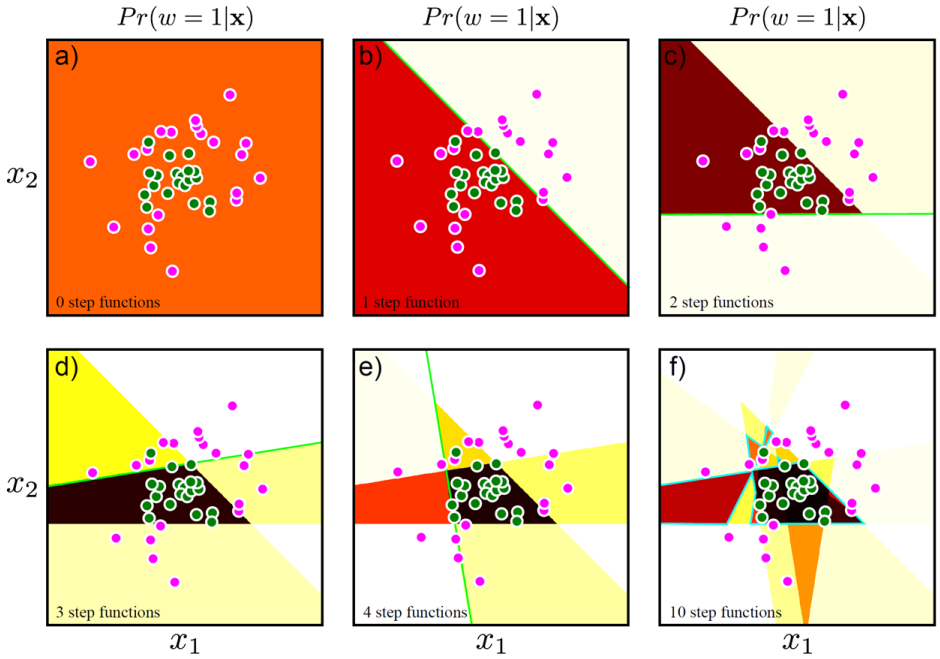



Figure 4.16: Boosting with different step functions.

**Definition 4.5.4.** Boosting is an ensemble learning technique that combines multiple weak classifiers into a strong classifier. It optimizes an additive model iteratively:

$$a_i = \phi_0 + \sum_{k=1}^{K} \phi_k \text{Heaviside}[\alpha_k^\top x],$$

where Heaviside$[\cdot]$ is a weak classifier.

**Remark.** *The key idea in boosting is to assign higher weights to misclassified points, forcing subsequent classifiers to focus on the hard examples.*

**Example 4.5.5.** In AdaBoost, the weights $w_i$ are updated iteratively based on the errors:

$$w_i^{(t+1)} = w_i^{(t)} \exp\left(-y_i h_t(x_i)\right),$$

where $h_t(x_i)$ is the $t$-th weak classifier, and $y_i$ is the true label.

### 4.5.3 Branching Logistic Regression

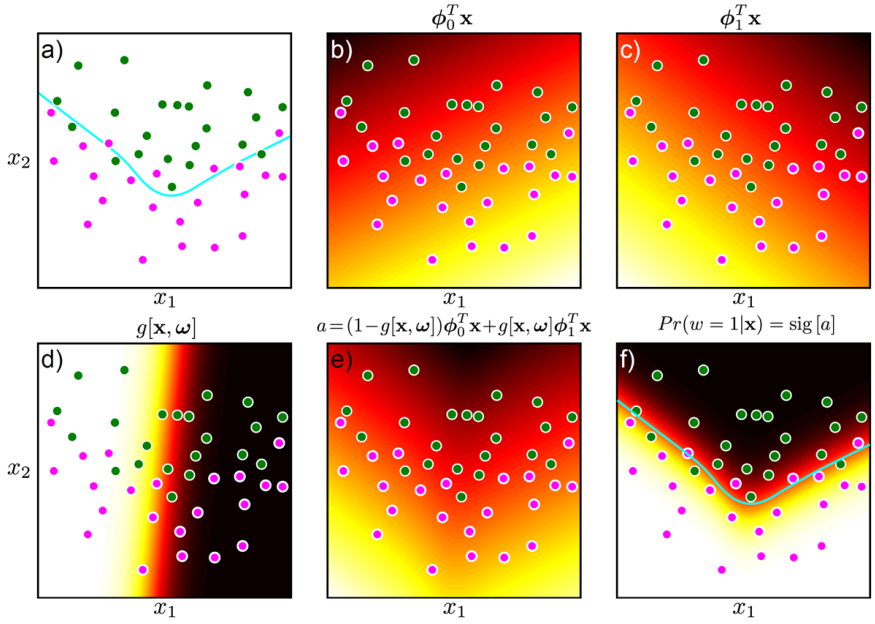



Figure 4.17: Branching Logistic Regression.

**Definition 4.5.6.** Branching logistic regression introduces a gating function $g[x, \omega]$ to switch between two logistic regression models:

$$a_i = (1 - g[x_i, \omega])\phi_0^\top x_i + g[x_i, \omega]\phi_1^\top x_i,$$

where $g[\cdot, \cdot]$ returns a value between 0 and 1.

**Remark.** *The gating function $g[x, \omega]$ allows the model to choose between two classifiers based on the input data. If $g[x, \omega] = 0$, $\phi_0^\top x_i$ is used; if $g[x, \omega] = 1$, $\phi_1^\top x_i$ is used.*

**Example 4.5.7.** Let $g[x, \omega] = \sigma(\omega^\top x)$, where $\sigma(\cdot)$ is the sigmoid function. This defines a smooth transition between the two logistic regression models.

## 4.6 Multi-Class Logistic Regression

### 4.6.1 Model Formulation

**Definition 4.6.1.** Multi-class logistic regression is a generalization of binary logistic regression for $K$ classes. The probability of a class $w = k$ given an input $x$ is modeled as:

$$\Pr(w = k \mid x) = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)},$$

where $a_k = \phi_k^\top x$ and $\phi_k$ is the parameter vector for class $k$. The function mapping the activations $\{a_k\}$ to probabilities is known as the *softmax function*.

### 4.6.2 Softmax Function

**Remark.** *The softmax function ensures that:*

- *Each probability* $\Pr(w = k \mid x)$ *lies between* $0$ *and* $1$.
- *The probabilities sum to* $1$*, i.e.,* $\sum_{k=1}^{K} \Pr(w = k \mid x) = 1$.

*The parameters* $\phi_k$ *are learned for each class* $k$*, and* $a_k = \phi_k^\top x$ *is the activation for class* $k$.

### 4.6.3 Visualization of Class Activations

**Example 4.6.2.** Consider a three-class problem where the activations are linear functions of $x$:

$$a_k = \phi_k^\top x \quad \text{for } k = 1, 2, 3.$$

The softmax function transforms these activations into probabilities, as illustrated in the graph below:

- Panel (a): Linear activations for each class.
- Panel (b): Corresponding class probabilities as functions of $x$.

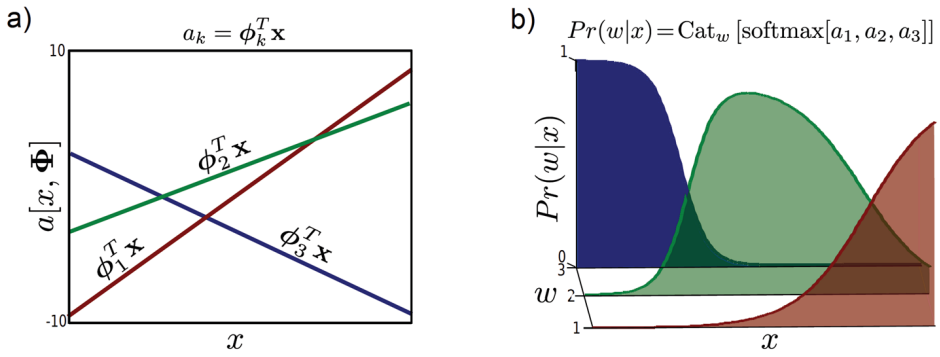



Figure 4.18: Multiclass Logistic Regression Class Activations.

### 4.6.4 Parameter Learning

**Theorem 4.6.3** (Maximum Likelihood Estimation). *Parameters $\{\phi_k\}_{k=1}^K$ are learned by maximizing the log-likelihood:*

$$L = \sum_{i=1}^{I} \log \Pr(w_i \mid x_i) = \sum_{i=1}^{I} \log \frac{\exp(a_{w_i})}{\sum_{j=1}^{K} \exp(a_j)}.$$

*This is equivalent to minimizing the cross-entropy loss.*

**Remark.** *The gradients and Hessians for optimization are given by:*

$$\frac{\partial L}{\partial \phi_k} = \sum_{i=1}^{I} x_i \big(y_{ik} - \delta[w_i - k]\big),$$

$$\frac{\partial^2 L}{\partial \phi_j \partial \phi_k} = \sum_{i=1}^{I} x_i x_i^\top \big(y_{ik}\delta[k-j] - y_{ij}y_{ik}\big),$$

*where:*

- $y_{ik} = \Pr(w_i = k \mid x_i)$ *is the predicted probability.*
- $\delta[w_i - k]$ *is an indicator function that equals* 1 *if* $w_i = k$ *and* 0 *otherwise.*

### 4.6.5 Optimization Challenges

**Remark.**

- *The optimization problem is non-linear and lacks a closed-form solution.*
- *Gradient-based optimization methods such as gradient descent or quasi-Newton methods (e.g., L-BFGS) are commonly used.*

## 4.7 Random Classification Trees

### 4.7.1 Binary Tree Splitting

**Definition 4.7.1.** A **random classification tree** is a decision tree constructed by:

- Randomly choosing a function $q(x)$ at each split.
- Selecting a threshold $\tau$ to maximize the log-likelihood of the split.

**Theorem 4.7.2** (Log-Likelihood Maximization)**.** *The objective function for a single split at node $t$ is the log-likelihood:*

$$L = \sum_{i=1}^{I} \Big[ (1 - \textit{heaviside}[q(x_i) - \tau]) \log \Pr(w_i \mid x_i; \lambda^{[l]}) + \textit{heaviside}[q(x_i) - \tau] \log \Pr(w_i \mid x_i; \lambda^{[r]}) \Big],$$

*where:*

- *$\lambda^{[l]}$ and $\lambda^{[r]}$ are the class probabilities for the left and right child nodes, respectively.*
- *The Heaviside function heaviside$[q(x) - \tau]$ determines the split direction.*

For a given threshold $\tau$, the class probabilities can be computed in closed form as:

$$\lambda_k^{[l]} = \frac{\sum_{i=1}^{I} \delta[w_i = k](1 - \text{heaviside}[q(x_i) - \tau])}{\sum_{i=1}^{I} (1 - \text{heaviside}[q(x_i) - \tau])},$$

$$\lambda_k^{[r]} = \frac{\sum_{i=1}^{I} \delta[w_i = k]\text{heaviside}[q(x_i) - \tau]}{\sum_{i=1}^{I} \text{heaviside}[q(x_i) - \tau]}.$$

### 4.7.2 Related Models

**Example 4.7.3** (Fern)**.** A **fern** is a simplified random classification tree where:

- All functions $q(x)$ at a given level are the same.
- Thresholds at each level may vary or remain constant.

Ferns are computationally efficient and commonly used in lightweight classification tasks.

**Example 4.7.4** (Random Forest)**.** A **random forest** is an ensemble of random classification trees, combining their predictions to improve robustness. Predictions are averaged, akin to Bayesian model averaging:

$$\Pr(w \mid x) = \frac{1}{T} \sum_{t=1}^{T} \Pr_t(w \mid x),$$

where $T$ is the number of trees in the forest.

## 4.8 Non-Probabilistic Classifiers

### 4.8.1 Overview

**Remark.** *Non-probabilistic classifiers are widely used in machine learning, including methods like neural networks, AdaBoost, and support vector machines. Historically, these methods gained popularity due to their simplicity and performance. However, probabilistic approaches offer several advantages:*

- *They produce uncertainty estimates naturally.*
- *They are easily extensible to multi-class cases.*
- *They are mathematically consistent and related to one another.*

### 4.8.2 Examples of Non-Probabilistic Classifiers

#### Multi-Layer Perceptron (Neural Networks)

**Definition 4.8.1.** A multi-layer perceptron is a non-linear logistic regression model that uses sigmoid functions for activation. The learning process is known as *backpropagation*. The hidden layers represent the transformed variable $z$.

#### AdaBoost

**Definition 4.8.2.** AdaBoost is a boosting algorithm that combines weak classifiers to form a strong ensemble. It is closely related to LogitBoost and achieves comparable performance.

#### Support Vector Machines (SVMs)

**Remark.** *Support vector machines (SVMs) are similar to relevance vector classification but differ in key aspects:*

- *The objective function is convex.*
- *SVMs do not provide uncertainty estimates.*
- *They are not easily extensible to multi-class classification.*
- *SVMs produce less sparse solutions.*
- *They impose stricter restrictions on kernel functions.*

### 4.8.3 Comparison to Probabilistic Approaches

**Remark.** *Probabilistic classifiers, such as logistic regression or Bayesian methods, are preferred in scenarios where uncertainty estimation and interpretability are crucial. Unlike non-probabilistic methods, they align naturally with statistical theory and extend seamlessly to more complex tasks.*

## 4.9 Applications of Classification

- **Gender Classification:** Achieved 87.5% accuracy using 300 arc tan basis functions.

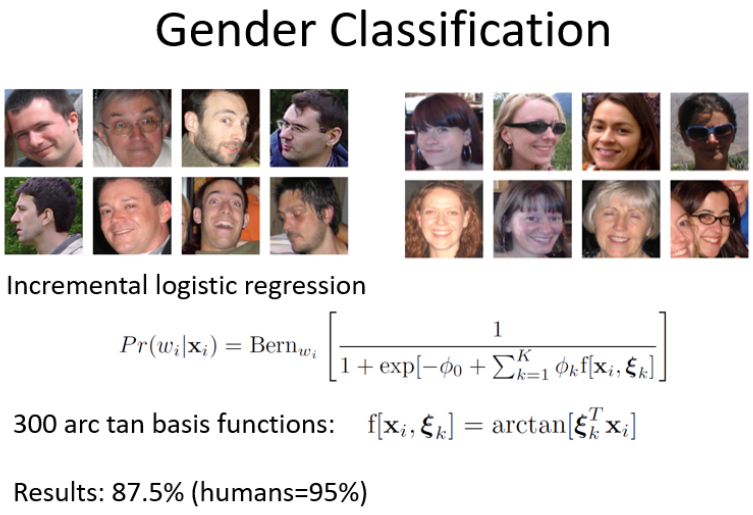



Figure 4.19:

- **Pedestrian Detection:** Used boosted classifiers with Haar wavelet features.

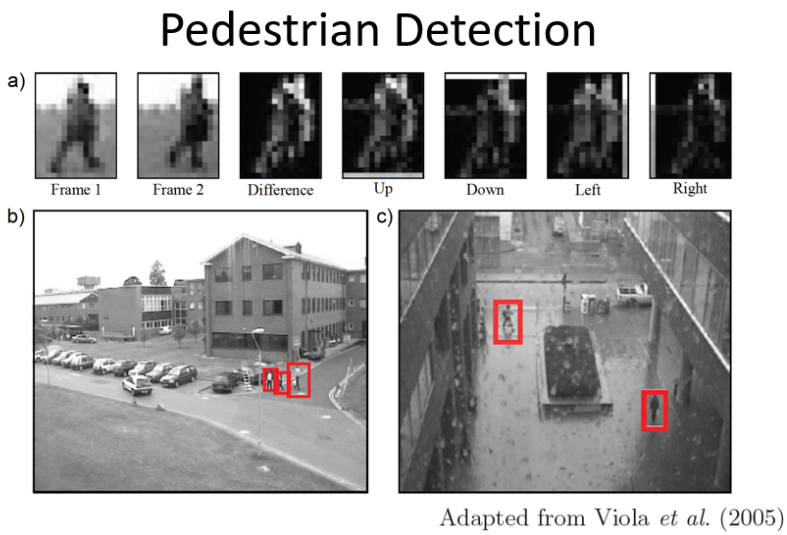



Figure 4.20:

- **Semantic Segmentation:** Classified image pixels into semantic categories.

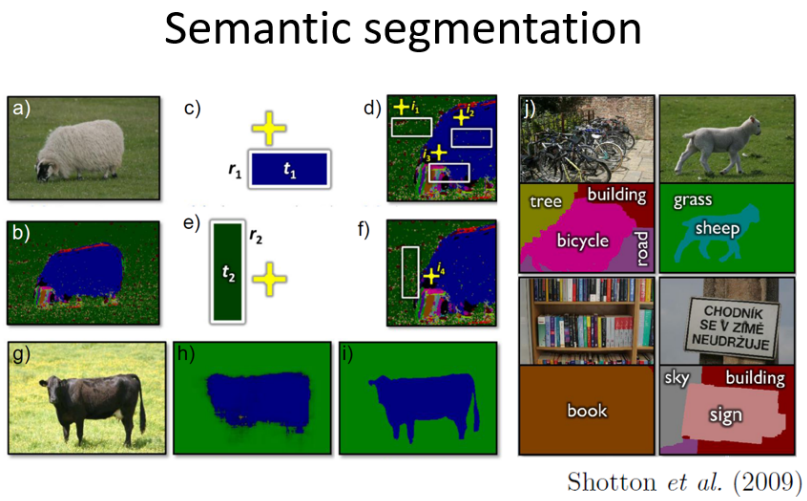



Figure 4.21:

- **Body Pose Estimation:** Predicted body pose vectors from silhouettes.

- Recovering Body Post

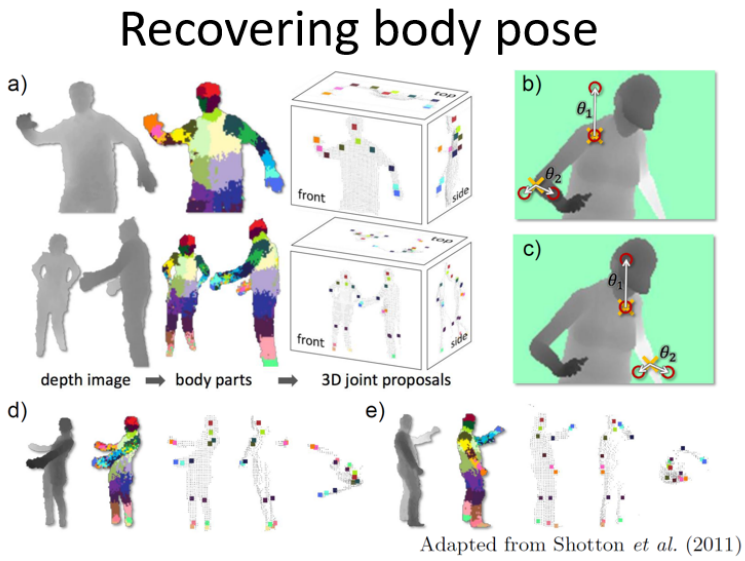



Figure 4.22:

- **Recovering Surface Layout**

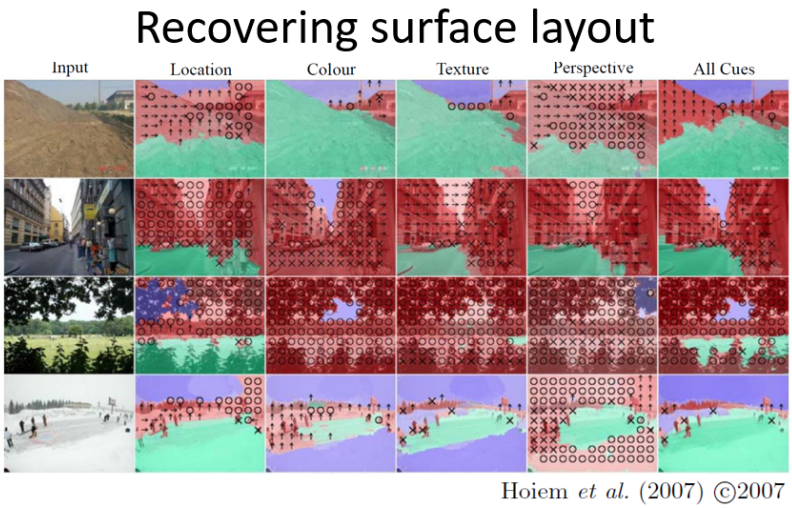



Figure 4.23:

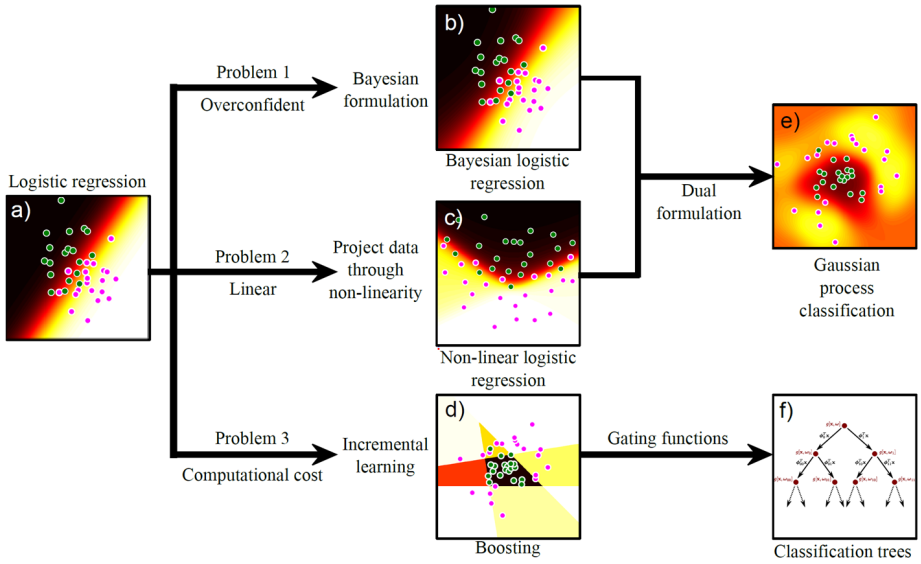



Figure 4.24: Classification Models Development.

# Chapter 5

# Graphical Models for Chains and Trees

## 5.1 Graphical Models: Definitions

**Definition 5.1.1** (Graphical Model)**.** A **graphical model** is a probabilistic framework that uses a graph to represent relationships between random variables. It facilitates:

- **Factorization:** Decomposing a joint distribution into a product of simpler distributions.
- **Conditional Independence:** Representing independence relationships efficiently.

**Definition 5.1.2** (Conditional Independence)**.** A variable $x_1$ is said to be conditionally independent of $x_3$ given $x_2$ if $x_1$ and $x_3$ are independent for fixed $x_2$. This is expressed as:

$$Pr(x_1|x_2, x_3) = Pr(x_1|x_2), \quad Pr(x_3|x_2, x_1) = Pr(x_3|x_2).$$

**Remark** (Factorization of Joint Density)**.** *When conditional independence holds, the joint density factorizes in the following way:*

$$Pr(x_1, x_2, x_3) = Pr(x_3|x_2, x_1)Pr(x_2|x_1)Pr(x_1),$$

*which simplifies to:*

$$Pr(x_1, x_2, x_3) = Pr(x_3|x_2)Pr(x_2|x_1)Pr(x_1).$$

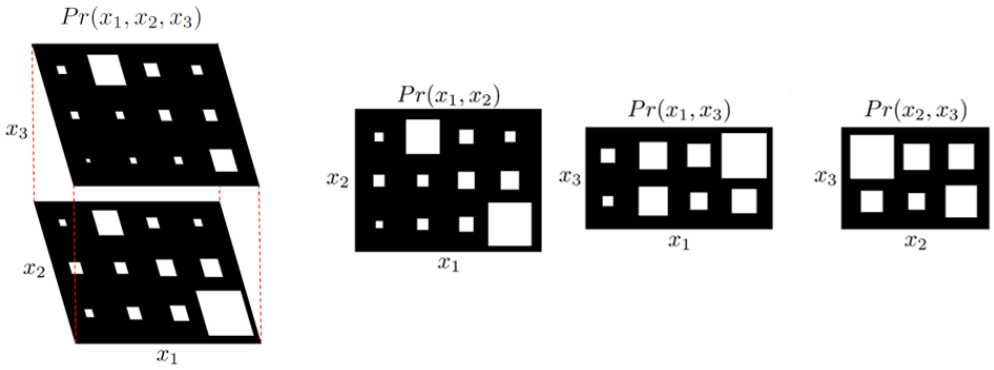



Figure 5.1: **Marginal Distribution Illustration** (big square means high probability). The three marginal distributions show that no pair of variables is independent.

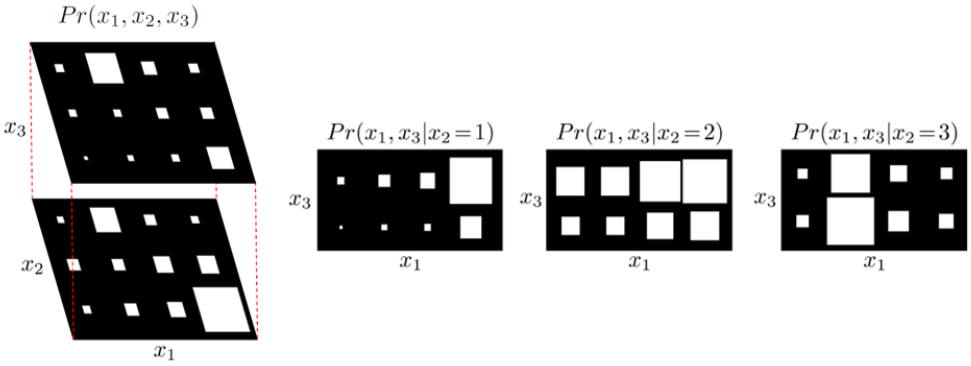



Figure 5.2: **Conditional Independence Illustration**. We can see that x1 is independent of x3 given x2.

**Remark.** *Graphical models are categorized into:*

- ***Directed Graphical Models (Bayesian Networks):*** *Encode conditional dependencies using directed edges.*

- ***Undirected Graphical Models (Markov Networks):*** *Use undirected edges to represent symmetric relationships.*

*Both models provide compact representations of high-dimensional joint distributions.*

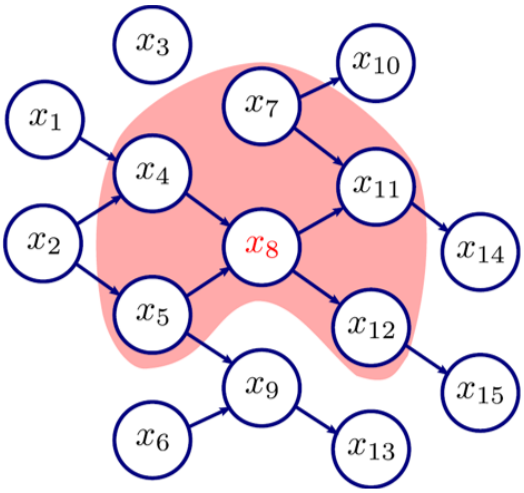
## 5.2 Directed Graphical Models (Bayesian Networks)

### 5.2.1 Definition and Factorization

**Definition 5.2.1.** A **directed graphical model** represents probability distribution that factorizes a product of conditional probability distributions. It represents a joint distribution as:

$$\Pr(\mathbf{x}) = \prod_{n=1}^{N} \Pr(x_n \mid \text{pa}[n]),$$

where pa$[n]$ denotes the parents of node $n$.




$$\begin{aligned} Pr(x_1 \dots x_{15}) \;=\; & Pr(x_1)Pr(x_2)Pr(x_3)Pr(x_4|x_1,x_2)Pr(x_5|x_2)Pr(x_6) \\ & Pr(x_7)Pr(x_8|x_4,x_5)Pr(x_9|x_5,x_6)Pr(x_{10}|x_7)Pr(x_{11}|x_7,x_8) \\ & Pr(x_{12}|x_8)Pr(x_{13}|x_9)Pr(x_{14}|x_{11})Pr(x_{15}|x_{12}). \end{aligned}$$

Figure 5.3: **Directed Graphical Model Example**.

**Remark.** *If there's no edge between two nodes and they share no ancestors, they are independent. A node is conditionally independent of all others, given its **Markov Blanket** (in Figure 5.3, the markov blanket of $x_8$ is the pink area: parents, children, and parents of children, basically all we need to predict $x_8$.)*

*The general rule is that the variables in set $A$ are conditionally independent of those in set $B$ given set $C$ if all edges from $A$ to $B$ are blocked. An edge is blocked at a node if (i) this node is in $C$ and the arrows meet head to tail or tail to tail or (ii) neither this node nor any of its descendants are in $C$ and the arrows meet head to head.*

### 5.2.2 Conditional Independence in Directed Models

**Theorem 5.2.2.** *Conditional independence relationships can be inferred from the structure of the graph:*

- *Two variables are conditionally independent given their Markov Blanket.*
- *The Markov Blanket of $x_n$ includes:*
  - *The parents of $x_n$.*
  - *The children of $x_n$.*
  - *The parents of the children of $x_n$.*

**Remark** (Redundancy)**.** *Conditional independence can be though of as redundancy in the full distribution. Redundancy can be very significant with larger models.*

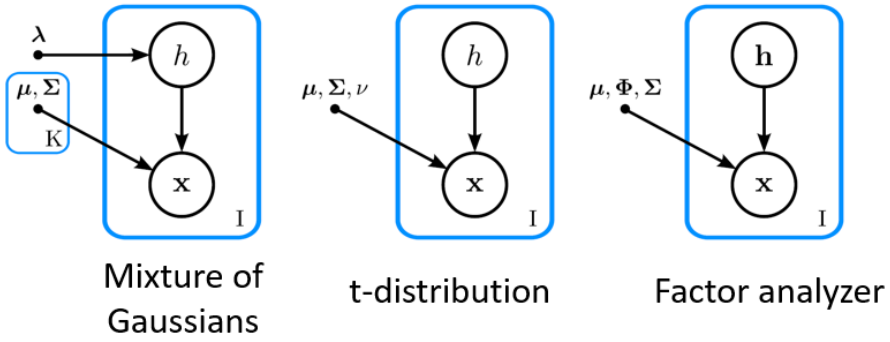



Figure 5.4: Directed Graphical Model Example.

### 5.2.3 Sampling in Directed Models

**Remark.** *Sampling from directed models uses **ancestral sampling**:*

1. *Sample variables with no parents (roots) from their marginal distributions.*
2. *Sample each subsequent variable given its parents, proceeding in topological order.*

*This method ensures samples are drawn consistent with the dependencies encoded in the graph.*

## 5.3 Undirected Graphical Models (Markov Networks)

### 5.3.1 Definition and Factorization

**Definition 5.3.1.** An **undirected graphical model** represents a joint distribution as:

$$\Pr(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \phi_C(\mathbf{x}_C),$$

where:

- $\mathcal{C}$: Set of all cliques in the graph. Product over $C$ functions.
- $\phi_C$: Potential function for clique $C$, returning a non-negative number.
- $Z$: Partition function, ensuring normalization constant.

For large systems, computing $Z$ is often intractable.

**Remark** (Gibbs Distribution)**.** *The joint distribution can also be written as a **Gibbs distribution**:*

$$\Pr(\mathbf{x}) = \frac{1}{Z} \exp\left( - \sum_{c=1}^{C} \psi_c(\mathbf{x}_C) \right),$$

*where:*

- $\psi_c(\mathbf{x}_C) = -\log(\phi_c(\mathbf{x}_C))$*: Cost function, which can be positive or negative.*
- $\phi_c$*: The original potential function.*

**Example 5.3.2** (Cliques)**.** To simplify notation and computation, the distribution can be expressed as a product over **cliques**:

$$\Pr(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \phi_C(\mathbf{x}_C),$$

where:

- A **clique** is a subset of variables $\mathbf{x}_C$ that are fully connected in the graph.
- Each $\phi_C(\mathbf{x}_C)$ represents the interaction within a clique.

**Remark.**
- ***To visualize graphical model from factorization:*** *Sketch one node per random variable. For every clique, sketch connection from every node to every other*
- ***To extract factorization from graphical model*** *Add one term to factorization per maximal clique (fully connected subset of nodes where it is not possible to add another node and remain fully connected)*

### 5.3.2 Sampling in Undirected Models

**Remark.** *Sampling from undirected models is typically done using* ***Markov Chain Monte Carlo (MCMC)*** *methods, such as Gibbs Sampling:*

- *Fix all variables except one and sample from its conditional distribution.*
- *Repeat for all variables, iteratively updating each variable in turn.*

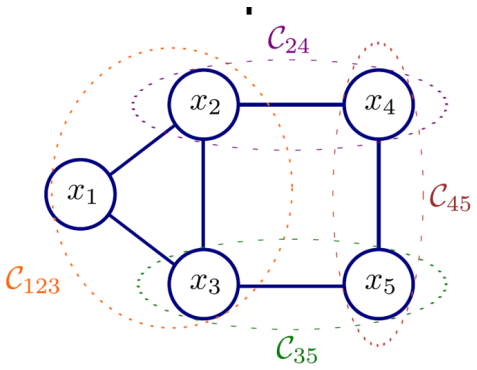



Figure 5.5: Undirected Graphical Model Example.

**Example 5.3.3.** Consider a joint probability distribution $P(x_1, x_2, x_3, x_4, x_5)$ represented by an undirected graphical model. The graph contains the following cliques:

- $C_{123}$: A clique containing $x_1, x_2, x_3$,
- $C_{24}$: A clique containing $x_2, x_4$,
- $C_{35}$: A clique containing $x_3, x_5$,
- $C_{45}$: A clique containing $x_4, x_5$.

Using these cliques, the joint distribution can be factorized as:

$$P(x_1, x_2, x_3, x_4, x_5) = \frac{1}{Z}\phi_{123}(x_1, x_2, x_3)\phi_{24}(x_2, x_4)\phi_{35}(x_3, x_5)\phi_{45}(x_4, x_5),$$

where $Z$ is the partition function ensuring normalization, and $\phi_C$ represents the potential function for clique $C$.

Alternatively, the factorization can be made less general by introducing overlapping factors:

$$P(x_1, x_2, x_3, x_4, x_5) = \frac{1}{Z}(\phi_1(x_1, x_2)\phi_2(x_2, x_3)\phi_3(x_1, x_3))(\phi_4(x_2, x_4)\phi_5(x_3, x_5)\phi_6(x_4, x_5)),$$

where smaller factors are used to represent overlapping subsets of variables. However, this alternative is less general as it imposes additional independence assumptions.

## 5.4 Comparison of Directed and Undirected Models

### 5.4.1 Independence Relations

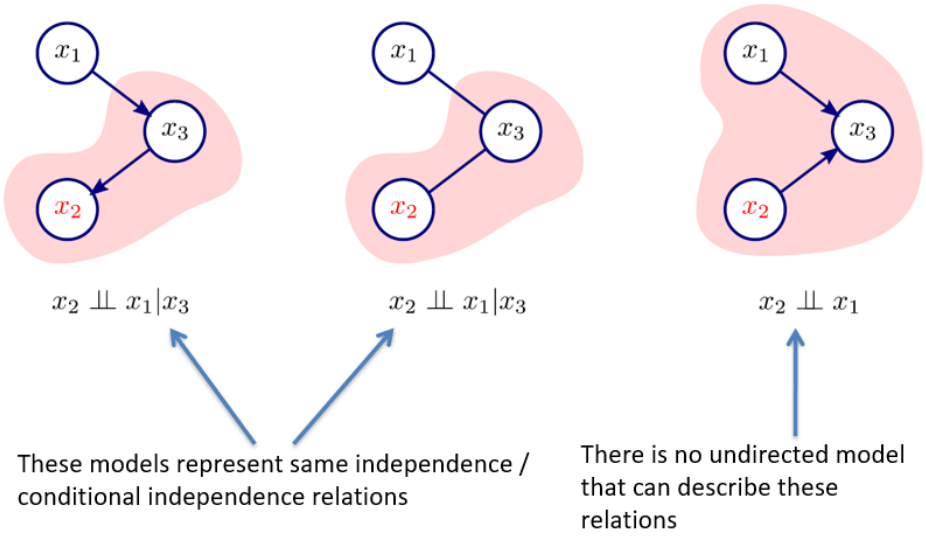



Figure 5.6: Comparing directed and undirected models: Independence Relations.

**Definition 5.4.1.** Directed and undirected graphical models represent independence or conditional independence relations differently:

- Directed models are represented by directed acyclic graphs (DAGs), capturing causal relationships.
- Undirected models use undirected graphs to depict symmetric relationships among variables.

**Remark.** ***Example of Conditional Independence:***

- *Both directed and undirected models can represent some independence relations, such as:*

$$x_2 \perp x_1 \mid x_3.$$

- *There are cases where directed or undirected models fail to describe certain independence relations:*
  - *For undirected models, some conditional independence relations cannot be expressed.*
  - *For directed models, the same applies to certain structures, such as loops.*

### 5.4.2 Structural Comparison

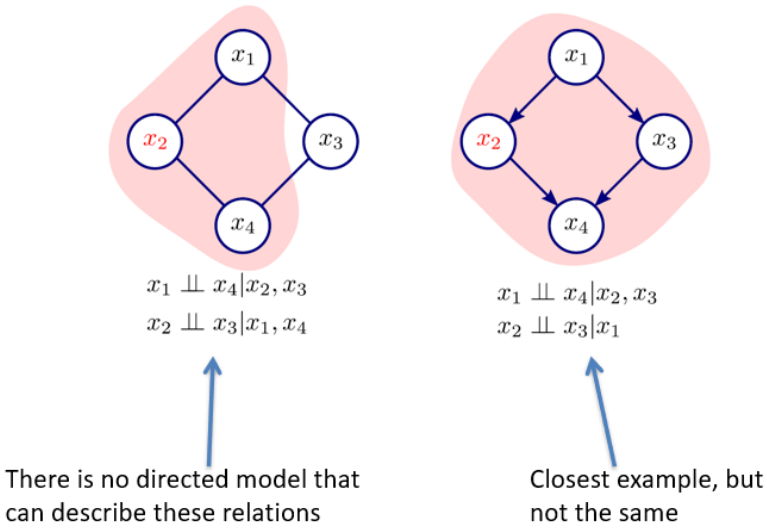



Figure 5.7: Comparing directed and undirected models: Structural Comparison

**Example 5.4.2.** Consider the relations:

$$x_1 \perp x_4 \mid x_2, x_3 \quad \text{and} \quad x_2 \perp x_3 \mid x_1, x_4.$$

- These relations are well-represented in undirected models.
- Closest directed models fail to express the exact relations.

## 5.5 Graphical Models in Computer Vision

### 5.5.1 Applications

**Remark.** *Graphical models are widely used in computer vision tasks:*

- ***Hidden Markov Models (Chain Model):*** *Sequence modeling, such as interpreting sign language.*
- ***Tree Models:*** *Generative modeling of structured data.*
- ***Grid Model (Markov Random Field)****: Semantic Segmentation.*
- ***Kalman Filter (Chain Model)****: Tracking Contours.*

### 5.5.2 Generative Modeling

**Definition 5.5.1.** Graphical models in computer vision often involve generative models, defined as:

$$\Pr(\mathbf{x} \mid \mathbf{w}),$$

where:

- $\mathbf{x}$ represents the data (e.g., body points).
- $\mathbf{w}$ represents latent variables (e.g., pose parameters).

**Example 5.5.2. Human Body Parsing:**

- Nodes represent body joints.
- Edges capture spatial relationships between joints.
- The goal is to estimate $\Pr(\mathbf{x} \mid \mathbf{w})$ for pose detection.

## 5.6 Inference and Learning in Graphical Models

Graphical models offer a structured approach to representing complex probabilistic systems. This section focuses on inference techniques for solving probabilistic queries, such as finding MAP estimates and marginal posterior distributions, and learning strategies for parameter estimation in both directed and undirected models.

### 5.6.1 Inference in Models with Many Unknowns

**Remark.** *In graphical models, the goal is often to compute the posterior distribution:*

$$\Pr(w_{1\ldots N} \mid \mathbf{x}_{1\ldots N}),$$

*which is intractable for large systems due to the exponential growth in the number of states. Instead, practical approaches include:*

- ***MAP Estimation:*** *Finding the most probable configuration of variables.*
- ***Marginal Posterior Distributions:*** *Summarizing uncertainty for each variable individually.*
- ***Maximum Marginals:*** *Maximizing the marginal probability for each variable.*
- ***Sampling:*** *Generating samples from the posterior distribution to approximate solutions.*

### 5.6.2 MAP Estimation

**Definition 5.6.1.** The **Maximum a Posteriori (MAP)** estimate identifies the configuration of variables that maximizes the posterior probability:

$$\hat{w}_{1\ldots N} = \arg\max_{w_{1\ldots N}} \Pr(w_{1\ldots N} \mid \mathbf{x}_{1\ldots N}).$$

**Theorem 5.6.2.** *By Bayes' theorem, the MAP estimate can be reformulated as:*

$$\hat{w}_{1\ldots N} = \arg\max_{w_{1\ldots N}} \Pr(\mathbf{x}_{1\ldots N} \mid w_{1\ldots N}) \Pr(w_{1\ldots N}),$$

*where:*

- $\Pr(\mathbf{x}_{1\ldots N} \mid w_{1\ldots N})$ *is the likelihood.*
- $\Pr(w_{1\ldots N})$ *is the prior distribution.*

**Remark.** *Although MAP estimation simplifies the problem, searching through all possible configurations remains computationally challenging.*

### 5.6.3 Marginal Posterior Distributions

**Definition 5.6.3.** The **marginal posterior distribution** for a variable $w_n$ is computed by summing or integrating over all other variables:

$$\Pr(w_n \mid \mathbf{x}_{1\ldots N}) = \int \Pr(w_{1\ldots N} \mid \mathbf{x}_{1\ldots N}) dw_1 \ldots dw_{n-1} dw_{n+1} \ldots dw_N.$$

**Remark.** *Marginalization exploits the conditional independence structure in graphical models to reduce computation.*

**Example 5.6.4.** Consider a model with three variables $w_1$, $w_2$, and $w_3$. The marginal posterior for $w_1$ is:

$$\Pr(w_1 \mid \mathbf{x}) = \int\int \Pr(w_1, w_2, w_3 \mid \mathbf{x}) dw_2 dw_3.$$

### 5.6.4 Maximum Marginals

**Definition 5.6.5. Maximum marginals** refer to finding the most probable value for each variable independently:

$$\hat{w}_n = \arg\max_{w_n} \Pr(w_n \mid \mathbf{x}_{1\ldots N}).$$

**Remark.** *Maximum marginals may lead to configurations where the joint probability is zero, as the marginal modes may not co-occur.*

### 5.6.5 Sampling from the Posterior

**Remark.** *Sampling provides an efficient way to approximate posterior distributions when exact computation is infeasible:*

- *Use generated samples to estimate probabilities.*
- *Compute summary statistics such as means, modes, or variances.*
- *Techniques include ancestral sampling for directed models and MCMC for undirected models.*

### 5.6.6 Directed Models: Ancestral Sampling

**Definition 5.6.6. Ancestral sampling** generates samples by traversing a directed graph according to its topological order:

$$\Pr(\mathbf{x}_{1\ldots N}) = \prod_{n=1}^{N} \Pr(x_n \mid \text{pa}(n)),$$

where $\text{pa}(n)$ denotes the parents of $x_n$.

**Example 5.6.7.** In a directed graph:

1. Sample $x_1$ from $\Pr(x_1)$.
2. Sample $x_2$ from $\Pr(x_2 \mid x_1)$.
3. Sample $x_4$ from $\Pr(x_4 \mid x_1, x_2)$.

### 5.6.7 Undirected Models: MCMC Methods

**Definition 5.6.8. Markov Chain Monte Carlo (MCMC)** methods, such as Gibbs sampling, are used to sample from undirected models. A Markov chain is constructed where each sample depends only on the previous one.

**Remark.** *Gibbs sampling updates one variable at a time by conditioning on all others:*

$$\Pr(x_n \mid \mathbf{x}_{-n}),$$

*where $\mathbf{x}_{-n}$ represents all variables except $x_n$.*

**Example 5.6.9.** For a model with three variables $x_1, x_2, x_3$, Gibbs sampling alternates between:

$$x_1 \sim \Pr(x_1 \mid x_2, x_3), \quad x_2 \sim \Pr(x_2 \mid x_1, x_3), \quad x_3 \sim \Pr(x_3 \mid x_1, x_2).$$

### 5.6.8 Learning in Directed Models

**Definition 5.6.10.** Learning in directed models involves maximizing the likelihood of the observed data:

$$\hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{I} \prod_{n=1}^{N} \Pr(x_{i,n} \mid \text{pa}(n), \theta),$$

or equivalently, the log-likelihood:

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{I} \sum_{n=1}^{N} \log \Pr(x_{i,n} \mid \text{pa}(n), \theta).$$

**Remark.** *Parameter estimation often employs optimization techniques like gradient ascent or Expectation-Maximization (EM).*

### 5.6.9 Learning in Undirected Models

**Remark.** *Undirected graphical models can be expressed using the Gibbs distribution:*

$$\Pr(\mathbf{x}) = \frac{1}{Z} \exp\left( -\sum_{c=1}^{C} \psi_c[\mathbf{x}, \theta] \right),$$

*where:*

- $Z$*: Partition function ensuring normalization, given by* $Z = \sum_{\mathbf{x}} \exp\left( -\sum_{c=1}^{C} \psi_c[\mathbf{x}, \theta] \right)$.
- $\psi_c$*: Cost functions associated with cliques* $c$*, parameterized by* $\theta$.

**Definition 5.6.11** (Maximum Likelihood in Undirected Models)**.** The maximum likelihood estimate $\hat{\theta}$ is obtained by maximizing:

$$\mathcal{L}(\theta) = \sum_{i=1}^{I} \sum_{c=1}^{C} \psi_c[\mathbf{x}_i, \theta] - I \log Z(\theta).$$

The derivative of the log-likelihood is:

$$\frac{\partial \mathcal{L}}{\partial \theta} = -I \frac{\partial \log Z(\theta)}{\partial \theta} - \sum_{i=1}^{I} \sum_{c=1}^{C} \frac{\partial \psi_c[\mathbf{x}_i, \theta]}{\partial \theta}.$$

**Remark** (Challenges in Computing the Partition Function)**.** *Computing* $Z(\theta)$ *is intractable for large systems since it involves summing over all possible states. This makes exact inference and learning challenging.*

### 5.6.10 Contrastive Divergence

**Theorem 5.6.12** (Approximation of the Gradient)**.** *The gradient of the partition function can be approximated as:*

$$\frac{\partial \log Z(\theta)}{\partial \theta} = \sum_{\mathbf{x}} \Pr(\mathbf{x}) \frac{\partial \log f[\mathbf{x}, \theta]}{\partial \theta} \approx \frac{1}{J} \sum_{j=1}^{J} \frac{\partial \log f[\mathbf{x}_j^*, \theta]}{\partial \theta},$$

*where $\mathbf{x}_j^*$ are samples generated via Gibbs sampling or MCMC.*

**Example 5.6.13** (Practical Use of Contrastive Divergence)**.** In practice, a single iteration of MCMC suffices for approximating the gradient during training, enabling faster convergence.

### 5.6.11 Conclusions

**Remark.** *Graphical models characterize joint distributions using:*

- *Graphical structure representing dependencies.*
- *Conditional independence relations.*
- *Factorizations over cliques or variables.*

*There are two types of graphical models:*

- ***Directed models:** Learning is easier due to factorization, and sampling uses ancestral methods.*
- ***Undirected models:** Learning is harder due to the partition function, and sampling often requires MCMC methods like Gibbs sampling.*

## 5.7 Chain and Tree Models

**Remark.** *Chain and tree models simplify inference for large systems by leveraging sparsity in relationships between variables. These models are applied to problems where:*

- *Measurements $x_n$ are associated with world states $w_n$.*
- *The goal is to infer $w_n$ given $x_n$.*

*Sparse models reduce the parameter count by describing only subsets of relations, enabling efficient computation. For instance:*

- ***Chain models** focus on sequential dependencies, capturing the relationship between consecutive states.*
- ***Tree models** generalize the structure to allow branching relationships without loops, further optimizing representation.*

### 5.7.1 Key Assumptions

- World states $\{w_n\}$ are discrete.
- Observed variables $x_n$ depend only on their corresponding world states $w_n$.
- $x_n$ is conditionally independent of all other variables given $w_n$.

### 5.7.2 Chain Model: Definition and Applications

**Definition 5.7.1.** A **chain model** assumes that each world state depends only on the preceding state:

$$\Pr(\mathbf{w}) = \Pr(w_1) \prod_{n=2}^{N} \Pr(w_n \mid w_{n-1}).$$

Observed data $x_n$ depends solely on $w_n$:

$$\Pr(\mathbf{x} \mid \mathbf{w}) = \prod_{n=1}^{N} \Pr(x_n \mid w_n).$$

**Remark.** *Chain models are particularly useful in:*

- ***Gesture tracking:*** *Observations (e.g., image features) are modeled as dependent on discrete states (e.g., hand positions or gestures). Transition probabilities encode temporal dynamics.*
- ***Hidden Markov Models (HMM):*** *State transitions* $\Pr(w_n \mid w_{n-1})$ *and measurement models* $\Pr(x_n \mid w_n)$ *jointly capture the process.*

*For example, in sign language interpretation:*

- *The observation* $x_n$ *follows a Gaussian distribution conditioned on the state:*

$$\Pr(x_n \mid w_n = k) = \mathcal{N}(x_n; \mu_k, \Sigma_k).$$

- *The world state transitions are modeled as categorical:*

$$\Pr(w_n \mid w_{n-1} = k) = \mathit{Cat}(\lambda_k).$$

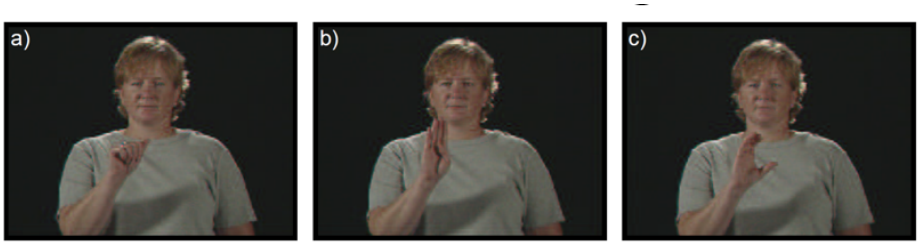



**Figure 10.1** Interpreting sign language. We observe a sequence of images of a person using sign language. In each frame we extract a vector $\mathbf{x}$ describing the shape and position of the hands. The goal is to infer the sign $w_n$ that is present. Unfortunately, the visual data in a single frame may be ambiguous. We improve matters by describing probabilistic connections between adjacent states $w_n$ and $w_{n-1}$; we impose knowledge about the likely sequence of signs and this helps disambiguate any individual frame. Frames from Purdue RVL-SLLL ASL database (Wilbur & Kak 2006).

Figure 5.8: Gesture Tracking using Chain Models.

### 5.7.3 Tree Model: Definition and Applications

**Definition 5.7.2.** A **tree model** organizes relationships between world states as a tree structure. Each node represents a state, and edges encode conditional dependencies:

$$\Pr(\mathbf{w}) = \prod_{n=1}^{N} \Pr(w_n \mid \text{pa}[n]),$$

where $\text{pa}[n]$ denotes the parent of node $n$.

**Remark.** *Tree models are widely used in:*

- ***Body parsing:*** *Nodes correspond to body parts (e.g., limbs, torso), and edges represent spatial constraints, such as joint connections.*
- ***Human pose estimation:*** *Tree structures efficiently model the dependencies among body parts while avoiding cycles.*

### 5.7.4 Directed vs. Undirected Models for Chains

**Remark.** *Chain models can be represented using both directed and undirected frameworks:*

- ***Directed:*** *Sequential dependencies are explicitly defined:*

$$\Pr(x_{1:N}, w_{1:N}) = \prod_{n=1}^{N} \Pr(x_n \mid w_n) \prod_{n=2}^{N} \Pr(w_n \mid w_{n-1}).$$

- ***Undirected:*** *Compatibility functions encode the relationships:*

$$\Pr(x_{1:N}, w_{1:N}) = \frac{1}{Z} \prod_{n=1}^{N} \phi(x_n, w_n) \prod_{n=2}^{N} \zeta(w_n, w_{n-1}),$$

*where $Z$ ensures normalization.*

### 5.7.5 Equivalence of Chain Models

**Remark.** *Directed and undirected chain models are equivalent under appropriate normalizations:*

$$\Pr(x_n \mid w_n) = \frac{1}{z_n}\phi(x_n, w_n), \quad \Pr(w_n \mid w_{n-1}) = \frac{1}{z'_n}\zeta(w_n, w_{n-1}).$$

## 5.8 MAP Inference in Chains

### 5.8.1 Definition

Maximum a Posteriori (MAP) inference aims to find the most probable configuration of world states $\mathbf{w}^*$ given observations $\mathbf{x}$:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \Pr(\mathbf{w} \mid \mathbf{x}).$$

For chain models, the problem can be reformulated using Bayes' rule and the Markov property, which simplifies the joint probability into unary and pairwise components:

$$\Pr(\mathbf{w}, \mathbf{x}) = \prod_{n=1}^{N} \Pr(x_n \mid w_n) \prod_{n=2}^{N} \Pr(w_n \mid w_{n-1}).$$

The logarithmic transformation results in:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left[ -\sum_{n=1}^{N} \log \Pr(x_n \mid w_n) - \sum_{n=2}^{N} \log \Pr(w_n \mid w_{n-1}) \right].$$

### 5.8.2 Dynamic Programming

Dynamic programming provides an efficient method to solve the MAP inference problem in chain models. The key idea is to decompose the minimization problem into subproblems that can be solved iteratively.

**Definition 5.8.1.** Let the unary cost $U_n(w_n)$ and pairwise cost $P(w_n, w_{n-1})$ be defined as:

$$U_n(w_n) = -\log \Pr(x_n \mid w_n), \quad P(w_n, w_{n-1}) = -\log \Pr(w_n \mid w_{n-1}).$$

The total cost function becomes:

$$\text{Cost}(\mathbf{w}) = \sum_{n=1}^{N} U_n(w_n) + \sum_{n=2}^{N} P(w_n, w_{n-1}).$$

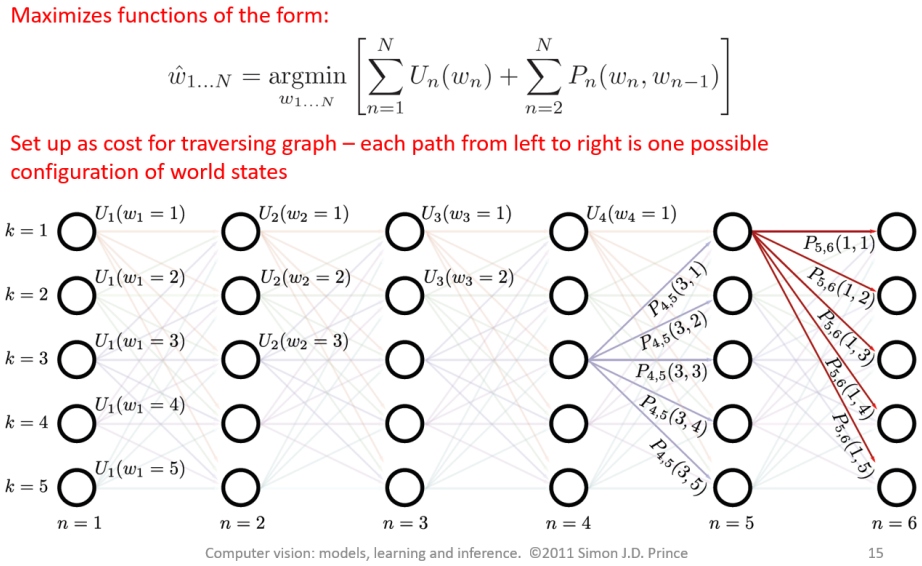



Figure 5.9: Dynamics Programming Illustrtaion.

**Remark.** *The dynamic programming algorithm involves:*

1. ***Forward Pass:*** *Compute the minimum cost $S_{n,k}$ to reach each node in the chain:*

$$S_{n,k} = U_n(w_n = k) + \min_{l} \left[ S_{n-1,l} + P(w_n = k, w_{n-1} = l) \right].$$

2. ***Backtrace:*** *Recover the optimal path by tracing the states with the minimum cost at each step.*

### 5.8.3 Worked Example

- **Initial Costs and Graph Structure.** The graph represents the chain model, where each node corresponds to a possible state $k$ at a given position $n$, and edges between nodes encode

pairwise costs. The unary costs $U_n(w_n)$ are shown at each node, while the pairwise costs $P(w_n, w_{n-1})$ are displayed on the edges connecting the nodes. At $n = 1$, the cost for each state is initialized to its unary cost, $S_{1,k} = U_1(w_1 = k)$.

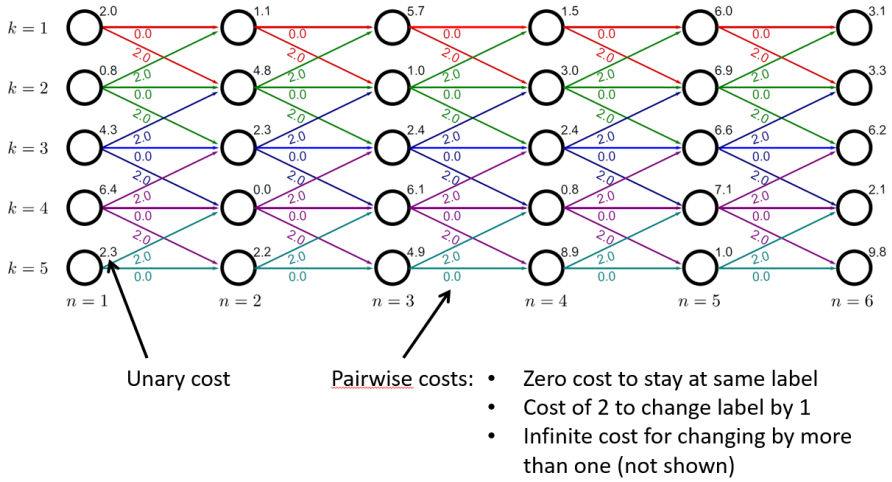
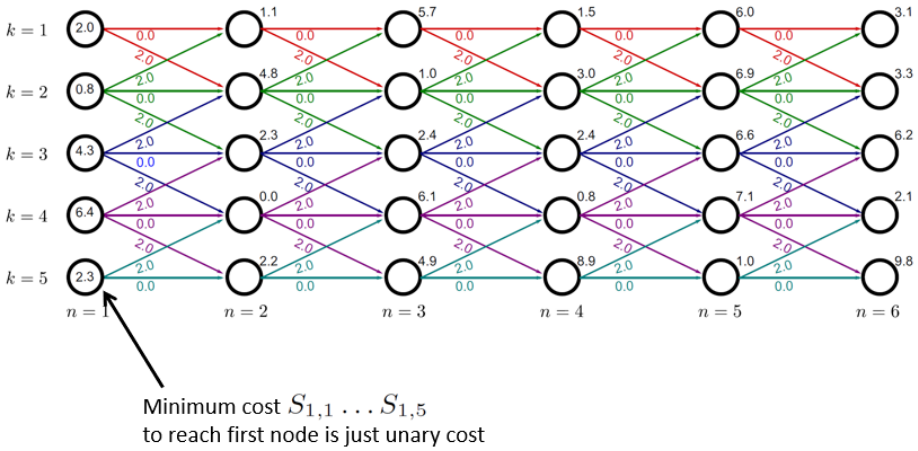



Figure 5.10: Worked Example Step 1: Unary Cost and Pairwise Cost.




$$S_{1,k} = U_1(w_1 = k)$$

Figure 5.11: Worked Example Step 1: Minimum Cost

- **General Rule for Cost Updates.** The general rule for updating the cost at any node is summarized as:

$$S_{n,k} = U_n(w_n = k) + \min_l \left[ S_{n-1,l} + P(w_n = k, w_{n-1} = l) \right].$$

This formula encapsulates the dynamic programming approach and is applied uniformly to all nodes in the graph.

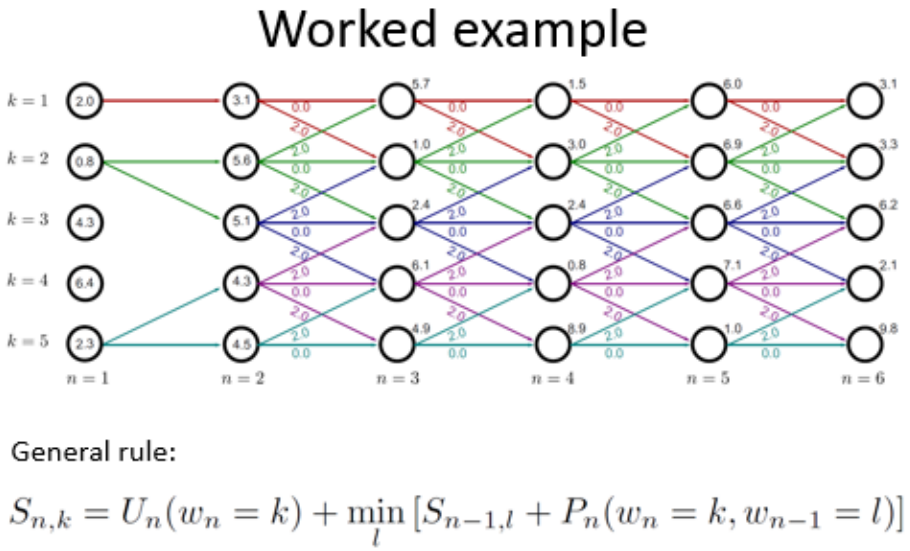



Figure 5.12: Worked Example Step 1: General Rule for Cost Updates. .

**Minimum Cost Computation for Node $n = 2$.** At $n = 2$, the minimum cost $S_{2,k}$ to reach each state $k$ is computed by considering all possible transitions from $n = 1$:

$$S_{2,k} = U_2(w_2 = k) + \min_l \left[ S_{1,l} + P(w_2 = k, w_1 = l) \right].$$

For example, to compute $S_{2,1}$, the algorithm evaluates all routes from $w_1 = l$ to $w_2 = 1$, adds the unary cost $U_2(w_2 = 1)$, and selects the route with the minimum total cost. This process is repeated for all states $k$ at $n = 2$.

$$S_{n,k} = U_n(w_n = k) + \min_l \left[ S_{n-1,l} + P(w_n = k, w_{n-1} = l) \right].$$

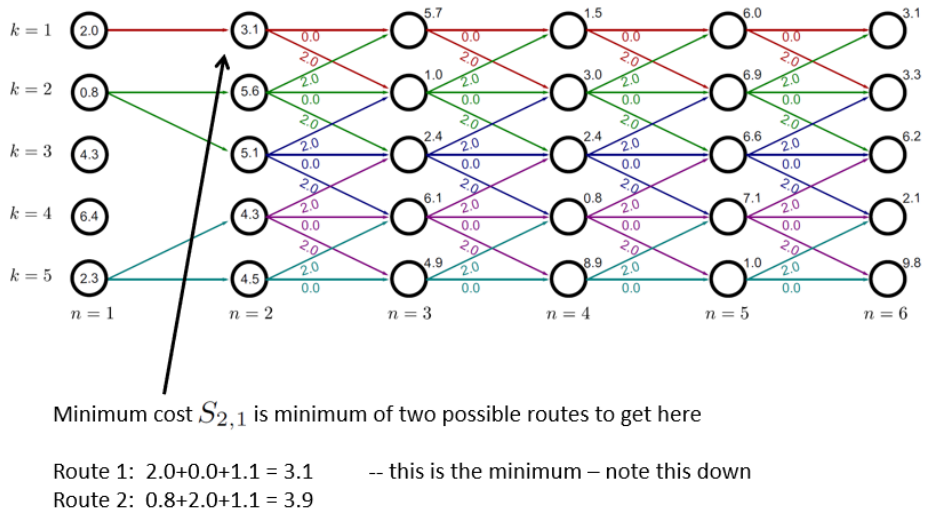



Figure 5.13: Worked Example Step 2: Minimum Cost Computation for Node at $n = 2$ .

**Cost Propagation to Nodes $n = 3$ to $n = 6$.** The computation continues iteratively across subsequent columns. For each node at $n = 3$, $n = 4$, and beyond, the minimum cost $S_{n,k}$ is computed by evaluating all possible transitions from $n - 1$ and selecting the one with the smallest total cost:

$$S_{n,k} = U_n(w_n = k) + \min_l \left[ S_{n-1,l} + P(w_n = k, w_{n-1} = l) \right].$$

Each computed cost is stored at the corresponding node, and this process ensures that the total cost to reach any state at the final column $n = N$ is minimized.

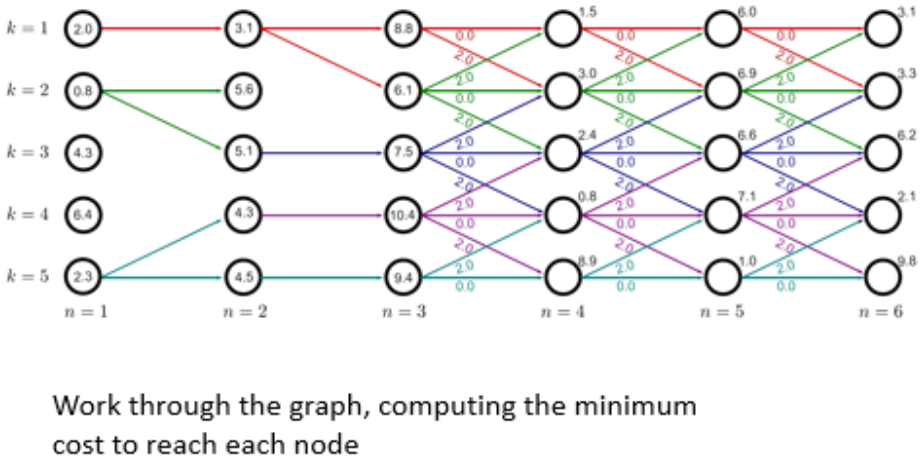



Figure 5.14: Worked Example Step 3: Cost Propagation to Nodes $n = 3$ to $n = 6$.

**Backtrace to Find the Optimal Path.** Once all costs are computed for the final column $n = N$, the algorithm identifies the state $k$ with the smallest total cost $S_{N,k}$. The backtrace step begins by recording this state and tracing backward through the graph, selecting the state at $n - 1$ that contributed to the minimum cost at $n$. This process is repeated until the initial column $n = 1$ is reached, reconstructing the optimal path $\mathbf{w}^*$.

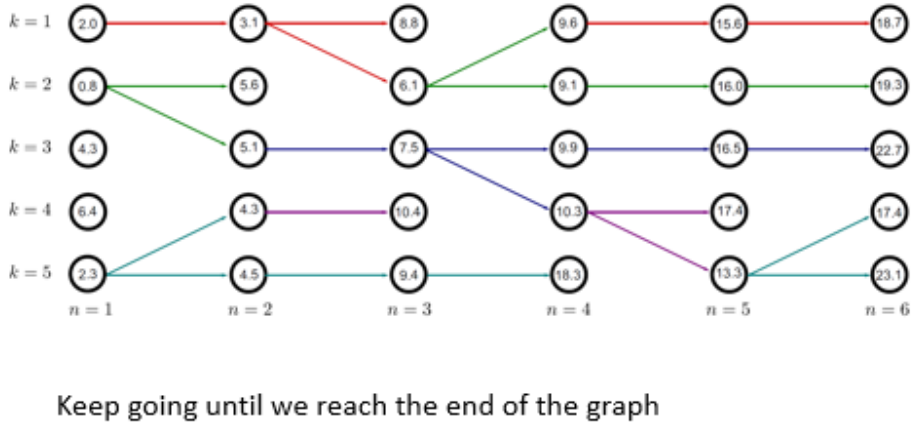



Figure 5.15: Worked Example Step 4: Backtrace to Find the Optimal Path.

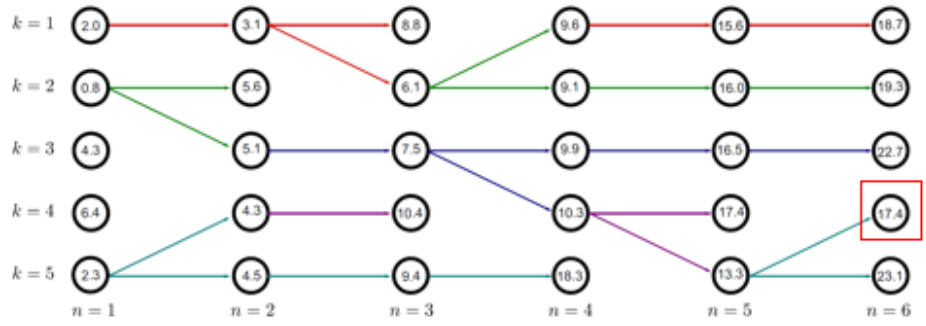



Figure 5.16: Worked Example Step 4: Backtrace to Find the Optimal Path.

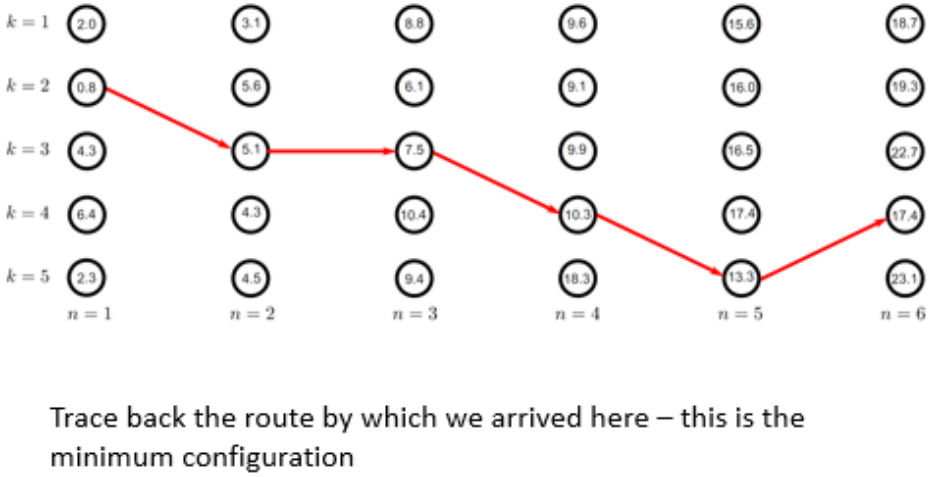



Figure 5.17: Worked Example Step 4: Backtrace to Find the Optimal Path

Consider a chain of $N = 6$ nodes, where each node $n$ can take one of $K = 5$ states. The MAP inference process for this chain is illustrated as follows:

- **Graph Representation:** Each node corresponds to a state at a particular position $n$, and edges between nodes represent transitions with pairwise costs. Each edge connects states from one column to the next, and the weights of these edges are derived from the pairwise costs $P(w_n, w_{n-1})$.

- **Unary Costs:** For each node, unary costs $U_n(w_n)$ represent the cost associated with observing $x_n$ given a particular state $w_n$.
- **Forward Pass:** Starting from the first column, the minimum cost $S_{n,k}$ to reach each node in subsequent columns is computed iteratively:
- **Backtrace:** Once the minimum cost for all nodes in the last column is computed, the optimal path is determined by tracing back through the states that contributed to the minimum cost at each step.
- **Visualization:** The computation of costs and the optimal path can be visualized using a graph, where:
  - Nodes represent states, labeled with their corresponding costs.
  - Edges show the transitions between states, labeled with their pairwise costs.

For example, in a graph with $N = 6$ and $K = 5$, the first node $S_{1,k}$ at $n = 1$ starts with its unary cost $U_1(w_1 = k)$. The subsequent costs are computed by considering all possible paths leading to each node and selecting the one with the minimum cost. This process continues until the last column is reached, and the optimal sequence of states is reconstructed by tracing back through the graph.

This worked example demonstrates the efficiency of dynamic programming for MAP inference in chain models by minimizing the computational complexity compared to exhaustive enumeration.

## 5.9 MAP Inference in Trees

### 5.9.1 Belief Propagation

**Definition 5.9.1.** Belief propagation is an algorithm for performing Maximum a Posteriori (MAP) inference in tree-structured models. It operates by propagating messages through the tree graph:

- **Forward pass**: Messages are computed from leaf nodes to the root to aggregate intermediate beliefs.
- **Backward pass**: Messages propagate from the root back to the leaf nodes to identify the optimal MAP configuration.

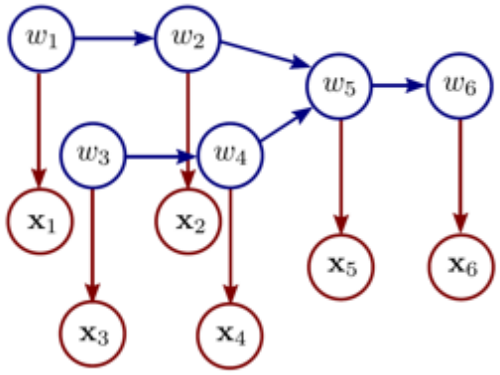
The computation relies on recursive combination of messages at each node and uses conditional independence properties to reduce computational complexity.

### 5.9.2 Worked Example

In this section, we analyze the MAP inference process in a tree-structured graph using belief propagation. Figures are referenced sequentially to explain the computations.

**Step 1: Initial Tree Structure.** Figure 1 illustrates a tree model with six world states $\{w_1, w_2, \ldots, w_6\}$ and corresponding observed data $\{x_1, x_2, \ldots, x_6\}$. The joint probability is expressed as:

$$\Pr(w_1, \ldots, w_6) = \Pr(w_1)\Pr(w_3 \mid w_1)\Pr(w_2 \mid w_1)\Pr(w_4 \mid w_3)\Pr(w_5 \mid w_2, w_4)\Pr(w_6 \mid w_5).$$



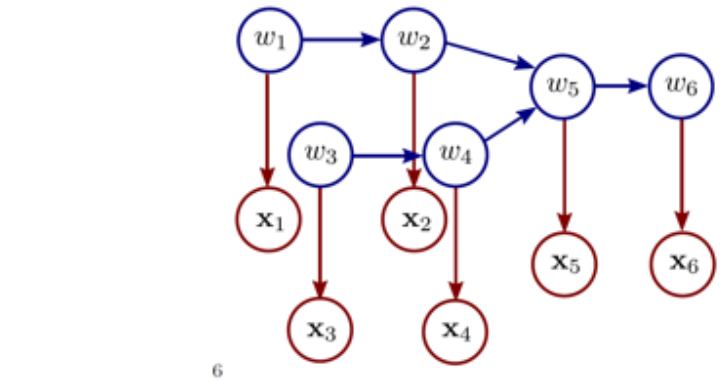

$$Pr(w_{1\ldots6}) = Pr(w_1)Pr(w_3)Pr(w_2|w_1)Pr(w_4|w_3)Pr(w_5|w_2, w_4)Pr(w_6|w_5)$$

Figure 5.18:

**Step 2: Recursive Formulation.** In Figure 2, the MAP inference problem is reformulated as a minimization of a sum of unary and pairwise costs:

$$\mathbf{w}^* = \arg\min_{w_1,\ldots,w_6} \sum_{n=1}^{6} U_n(w_n) + \sum_{\text{edges } (i,j)} P_{i,j}(w_i, w_j),$$

where $U_n(w_n) = -\log \Pr(x_n \mid w_n)$ and $P_{i,j}(w_i, w_j) = -\log \Pr(w_i \mid w_j)$.




$$\hat{w}_{1\ldots6} = \underset{w_{1\ldots6}}{\operatorname{argmax}} \sum_{n=1}^{6} \log[Pr(\mathbf{x}_n|w_n)] + \log[Pr(w_{1\ldots N})]$$

$$\hat{w}_{1\ldots6} = \underset{w_{1\ldots6}}{\operatorname{argmin}} \sum_{n=1}^{6} U_n(w_n) + P_2(w_2, w_1) + P_4(w_4, w_3) + P_6(w_6, w_5) + T_5(w_5, w_2, w_4)$$

Figure 5.19:

**Step 3: Message Passing and Cost Table Construction.** Figure 3 shows the cost tables $T_5(w_5, w_2, w_4)$ for node $w_5$ that aggregate messages from its child nodes. Each table entry is calculated by combining unary costs $U(w_5)$ and pairwise costs with $w_2$ and $w_4$:

$$T_5(w_5 = k, w_2 = l, w_4 = m) = U_5(w_5 = k) + P_{5,2}(w_5 = k, w_2 = l) + P_{5,4}(w_5 = k, w_4 = m).$$

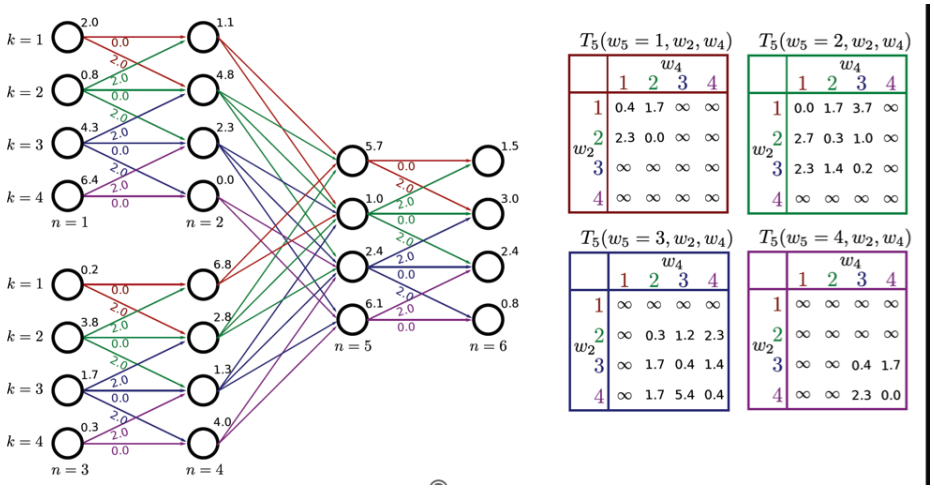



Figure 5.20:

**Step 4: Forward Pass to Compute Intermediate Costs.** In Figure 4, the algorithm proceeds recursively through the tree. Variables $w_1, w_2, w_3, w_4$ are computed first, as in the chain model:

$$S_{n,k} = U_n(w_n = k) + \min_l \left[ S_{n-1,l} + P_{n,n-1}(w_n = k, w_{n-1} = l) \right].$$

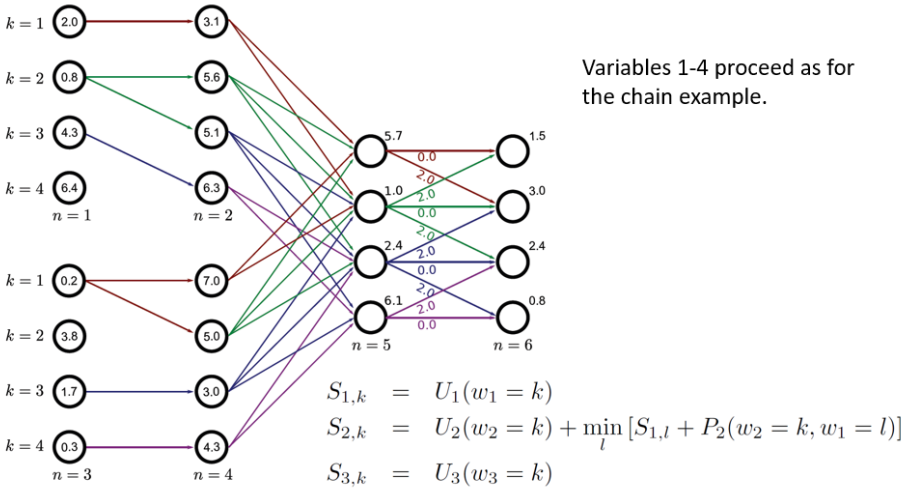
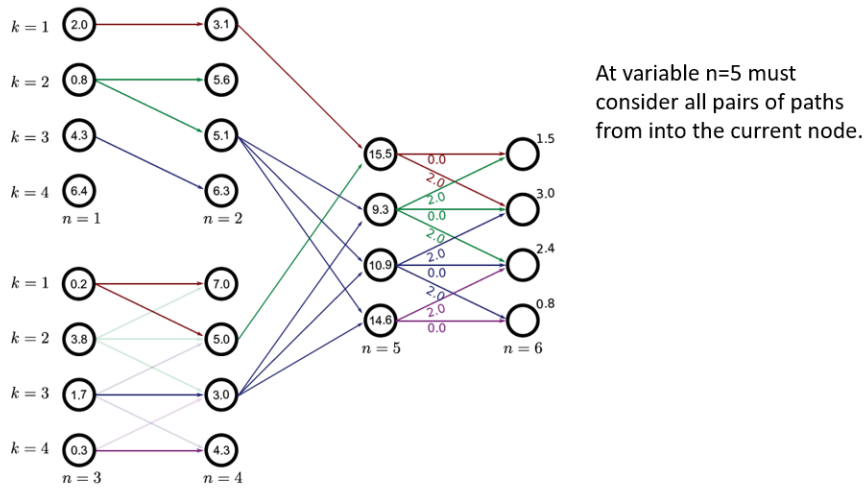



Figure 5.21:

**Step 5: Transition to Node $w_5$.** At $w_5$, shown in Figure 5, the algorithm considers all pairwise combinations of $w_2$ and $w_4$ to calculate:

$$S_{5,k} = U_5(w_5 = k) + \min_{l,m} \left[ S_{2,l} + S_{4,m} + T_5(w_5 = k, w_2 = l, w_4 = m) \right].$$




$$S_{5,k} = U_5(w_5 = k) + \min_{l,m} \left[ S_{2,l} + S_{4,m} + T_5(w_5 = k, w_2 = l, w_4 = m) \right]$$

Figure 5.22:

**Step 6: Completion at $w_6$ and Backtracking.** Figure 6 shows the final step, where $w_6$ is computed normally using:

$$S_{6,k} = U_6(w_6 = k) + \min_{l} \left[ S_{5,l} + P_{6,5}(w_6 = k, w_5 = l) \right].$$

The optimal configuration is reconstructed by tracing back from $w_6$ to $w_1$, splitting at junction nodes like $w_5$.

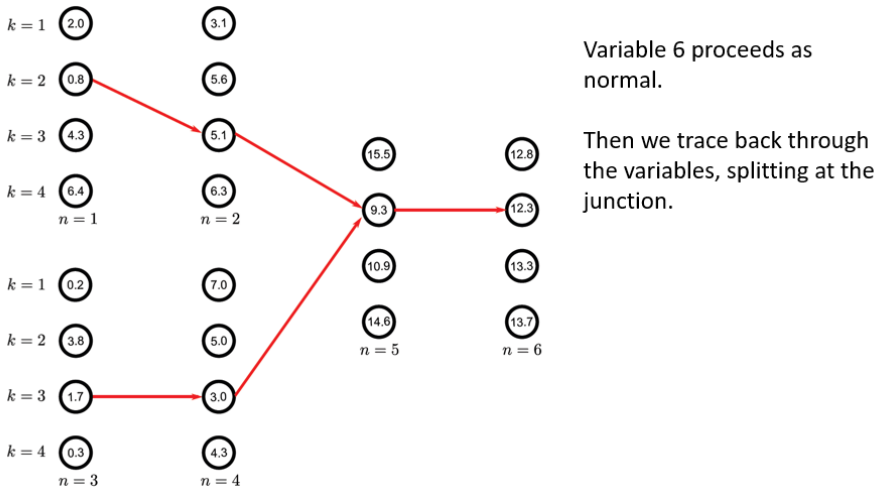



Figure 5.23:

## 5.10 Marginal Posterior Inference

### 5.10.1 Marginal Posterior Computation

To compute the marginal posterior for a single node, we begin by calculating the marginal distribution $\Pr(w_N \mid x_{1:N})$. Using Bayes' rule, this is expressed as:

$$\Pr(w_N \mid x_{1:N}) \propto \Pr(w_N, x_{1:N}),$$

where $\Pr(w_N, x_{1:N})$ represents the joint probability over all states and observations. The joint probability can be expanded as:

$$\Pr(w_N, x_{1:N}) = \sum_{w_1} \cdots \sum_{w_{N-1}} \Pr(w_1) \prod_{n=2}^{N} \Pr(w_n \mid w_{n-1}) \prod_{n=1}^{N} \Pr(x_n \mid w_n).$$

Directly summing over all $K^N$ possible states is computationally infeasible due to the exponential growth in the number of terms. To address this, we reorder terms and progressively move summations to the right:

$$\Pr(w_N \mid x_{1:N}) \propto \Pr(x_N \mid w_N) \sum_{w_{N-1}} \Pr(w_N \mid w_{N-1}) \sum_{w_{N-2}} \cdots \sum_{w_1} \Pr(w_2 \mid w_1) \Pr(w_1) \Pr(x_1 \mid w_1).$$

We define intermediate recursive functions to simplify the computation:

$$\begin{aligned} f_1[w_1] &= \Pr(x_1 \mid w_1)\Pr(w_1), \\ f_2[w_2] &= \Pr(x_2 \mid w_2)\sum_{w_1}\Pr(w_2 \mid w_1)f_1[w_1], \\ f_n[w_n] &= \Pr(x_n \mid w_n)\sum_{w_{n-1}}\Pr(w_n \mid w_{n-1})f_{n-1}[w_{n-1}]. \end{aligned}$$

The recursive relation allows efficient computation of $f_N[w_N]$. To normalize the result:

$$\Pr(w_N \mid x_{1:N}) = \frac{f_N[w_N]}{\sum_{w_N} f_N[w_N]}.$$

### 5.10.2 Forward-Backward Algorithm

To compute the marginal posteriors for all nodes efficiently, the forward-backward algorithm is employed. This method avoids redundancy by sharing intermediate computations.

**Forward Recursion:** Define the forward message $f_n[w_n]$ recursively:

$$f_n[w_n] = \Pr(x_n \mid w_n)\sum_{w_{n-1}}\Pr(w_n \mid w_{n-1})f_{n-1}[w_{n-1}].$$

The forward pass accumulates information from the beginning of the chain up to node $n$.

**Backward Recursion:** Similarly, define the backward message $b_{n-1}[w_{n-1}]$:

$$b_{n-1}[w_{n-1}] = \sum_{w_n}\Pr(x_{n+1} \mid w_n)\Pr(w_n \mid w_{n-1})b_n[w_n].$$

The backward pass accumulates information from the end of the chain back to node $n$.

**Combining Messages:** The marginal posterior for any node $n$ is computed as the product of forward and backward messages:

$$\Pr(w_n \mid x_{1:N}) \propto f_n[w_n]b_n[w_n],$$

where $f_n[w_n]$ incorporates evidence from preceding states, and $b_n[w_n]$ accounts for evidence from subsequent states.

### 5.10.3 Efficient Computation of Marginals

The forward-backward algorithm reduces the computational complexity to $O((N-1)K)$, a significant improvement over the brute force approach of $O(K^N)$.

### 5.10.4 Sum-Product Algorithm

The forward-backward algorithm is a special case of the sum-product algorithm, which generalizes to tree-structured graphs. This algorithm propagates messages between connected nodes and computes marginal posteriors as the product of incoming messages:

- **Forward Pass:** Messages are propagated from leaves to the root.
- **Backward Pass:** Messages are propagated back from the root to the leaves.

### 5.10.5 Factor Graphs

**Definition 5.10.1.** A **factor graph** represents the relationships between variables in a probabilistic model. It consists of:

- Variable nodes: Represent variables $w_n$ or observations $x_n$.
- Factor nodes: Represent functions $\Pr(w_{n+1} \mid w_n)$ or $\Pr(x_n \mid w_n)$.

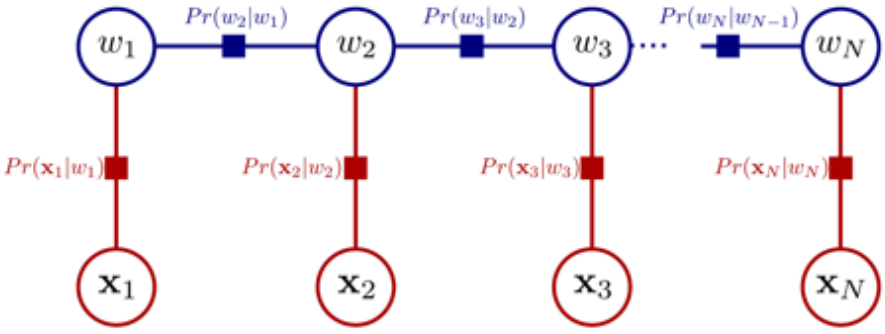



Figure 5.24: Factor Graph Definition.

### 5.10.6 Sum-Product Algorithm

**Definition 5.10.2.** The **sum-product algorithm** computes marginal distributions by passing messages between nodes in a factor graph. It consists of:

- **Forward pass:** Distributes evidence through the graph.
- **Backward pass:** Collates evidence from the graph.
- Both phases involve passing messages between nodes.. The forward phase can proceed in any order as long as the outgoing messages are not sent until all incoming ones received•Backward phase proceeds in reverse order to forward

**Remark.** *Three types of messages are exchanged in the sum-product algorithm:*

- ***Type 1:*** *From unobserved variables to functions. Take product of incoming messages.*

$$m_{z_p \to g_q} = \prod_{r \in Ne[p] \setminus q} m_{g_r \to z_p}.$$

- ***Type 2:*** *From observed variables to functions. Conveys certain belief that observed values are true.*

$$m_{z_p \to g_q} = \delta[z_p^*].$$

- ***Type 3:*** *From functions to variables. Takes beliefs from all incoming variables except recipient and uses function g to a belief about recipient*

$$m_{g_q \to z_p} = \sum_{z_r \in Ne[p] \setminus q} g_q[Ne[p]] \prod_{r \in Ne[p] \setminus q} m_{z_r \to g_q}.$$

### 5.10.7 Forward Pass

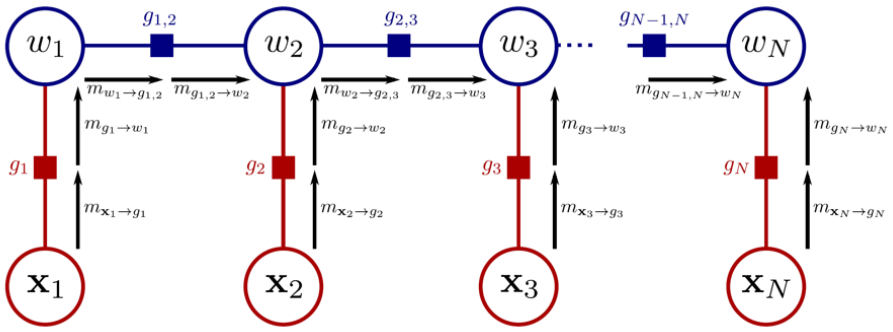



Figure 5.25: Sum Product: Forward Pass

**Example 5.10.3** (Message Passing in the Forward Pass).

- **From $x_1$ to $g_1$:** By Rule 2:

$$m_{x_1 \to g_1} = \delta[x_1^*].$$

- **From $g_1$ to $w_1$:** By Rule 3:

$$m_{g_1 \to w_1} = \int \Pr(x_1 \mid w_1)\delta[x_1^*]dx_1 = \Pr(x_1 = x_1^* \mid w_1).$$

- **From $w_1$ to $g_{1,2}$:** By Rule 1:

$$m_{w_1 \to g_{1,2}} = \Pr(x_1 = x_1^* \mid w_1).$$

- **From $g_{1,2}$ to $w_2$:** By Rule 3:

$$m_{g_{1,2} \to w_2} = \sum_{w_1} \Pr(w_2 \mid w_1) \Pr(x_1 = x_1^* \mid w_1).$$

- **Message from $x_2$ to $g_2$:**

$$m_{x_2 \to g_2} = \delta[x_2^*]$$

The observation for $x_2$ is conveyed directly.

- **Message from $g_2$ to $w_2$:**

$$m_{g_2 \to w_2} = \Pr(x_2 = x_2^* \mid w_2)$$

The probability of $x_2^*$ is passed to $w_2$.

- **Message from $w_2$ to $g_{2,3}$:**

$$m_{w_2 \to g_{2,3}} = \Pr(x_2 = x_2^* \mid w_2) \sum_{w_1} \Pr(w_2 \mid w_1) \Pr(x_1 = x_1^* \mid w_1)$$

This combines both the current and past messages.

- **Message from $g_{2,3}$ to $w_3$:**

$$m_{g_{2,3} \to w_3} = \sum_{w_2} \Pr(w_3 \mid w_2) \Pr(x_2 = x_2^* \mid w_2) \sum_{w_1} \Pr(w_2 \mid w_1) \Pr(x_1 = x_1^* \mid w_1)$$

This accumulates evidence from $x_1$ and $x_2$ and propagates it to $w_3$.

- **General Forward Message:** The forward recursion generalizes as:

$$m_{g_{n,n+1} \to w_{n+1}} = \sum_{w_n} \Pr(w_{n+1} \mid w_n) \Pr(x_n = x_n^* \mid w_n) m_{g_{n-1,n} \to w_n}.$$

This allows for systematic computation of messages across the chain.

### 5.10.8 Backward Pass

**Example 5.10.4** (Message Passing in the Backward Pass)**.**

- **From $x_N$ to $g_N$:** By Rule 2:

$$m_{x_N \to g_N} = \delta[x_N^*].$$

- **From $g_N$ to $w_N$:** By Rule 3:

$$m_{g_N \to w_N} = \int \Pr(x_N \mid w_N)\delta[x_N^*]dx_N = \Pr(x_N = x_N^* \mid w_N).$$

- **From $w_N$ to $g_{N-1,N}$:** By Rule 1:

$$m_{w_N \to g_{N-1,N}} = \Pr(x_N = x_N^* \mid w_N).$$

- **From $g_{N-1,N}$ to $w_{N-1}$:** By Rule 3:

$$m_{g_{N-1,N} \to w_{N-1}} = \sum_{w_N} \Pr(w_N \mid w_{N-1}) \Pr(x_N = x_N^* \mid w_N).$$

### 5.10.9 Collating Evidence

**Theorem 5.10.5.** *The marginal posterior distribution for any variable $w_n$ is computed as the product of all messages at the node:*

$$\Pr(w_n \mid \mathbf{x}) \propto \prod_{m \in \mathcal{N}[n]} m_{m \to w_n}$$

*Proof.* Using the factorization of the joint probability:

$$\Pr(w_n \mid \mathbf{x}) \propto \Pr(w_n, w_{<n}, w_{>n}, \mathbf{x}) = \Pr(w_n, w_{<n}, \mathbf{x}_{<n}) \Pr(w_{>n}, \mathbf{x}_{>n} \mid w_n).$$

Messages from the forward pass provide $\Pr(w_n, w_{<n}, \mathbf{x}_{<n})$, and messages from the backward pass provide $\Pr(w_{>n}, \mathbf{x}_{>n} \mid w_n)$. Combining these gives the marginal posterior. □

### 5.10.10 Marginal Posterior Inference for Trees

**Definition 5.10.6.** The marginal posterior inference for tree-structured graphs utilizes the sum-product algorithm to compute the marginal probabilities for all nodes. Unlike chains, the tree structure involves nodes with multiple parents or children, requiring bidirectional message passing.

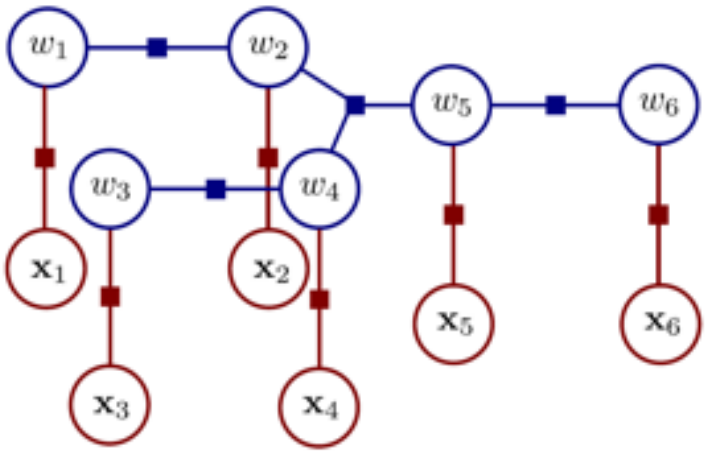



Figure 5.26: Marginal Posterior Inference for Trees.

**Remark.** *The algorithm proceeds in two stages:*

- ***Upward Pass (Leaf to Root):*** *Messages are propagated from the leaves of the tree toward the root, accumulating evidence.*
- ***Downward Pass (Root to Leaves):*** *Messages are propagated back from the root to the leaves, distributing combined beliefs to all nodes.*

**Example 5.10.7** (Tree Model)**.** Consider the tree structure shown in the figure. Each $x_i$ represents observed variables, while $w_i$ are latent states connected by edges representing dependencies.

- **Upward Message Passing:** For a node $w_3$ connected to $x_1$ and $w_1$, the message from $w_3$ to $w_1$ is:
$$m_{w_3 \to w_1} = \Pr(x_1 \mid w_3) \sum_{w_3} \Pr(w_3 \mid w_1).$$
- **Downward Message Passing:** Starting from the root node $w_2$, a downward message is passed to $w_4$:
$$m_{w_2 \to w_4} = \sum_{w_2} m_{w_1 \to w_2} \Pr(w_4 \mid w_2).$$

**Remark.** *The result is a set of marginal probabilities for each latent state $w_i$, computed as:*

$$\Pr(w_i \mid \mathbf{x}) \propto \prod_{m \in Ne[i]} m_{m \to w_i},$$

*where $Ne[i]$ are the neighbors of node $w_i$.*

## 5.11 Models with Loops

### 5.11.1 Definition

**Remark.** *Models with loops are graphical models where the underlying structure contains cycles. These are prevalent in many practical scenarios, such as:*

- ***Grid-based graphs:*** *Common in computer vision tasks like segmentation and stereo vision.*
- ***Dense graphical models:*** *Arise in problems with highly interdependent variables.*

*Unlike tree models, loops pose additional challenges for inference, making dynamic programming infeasible.*

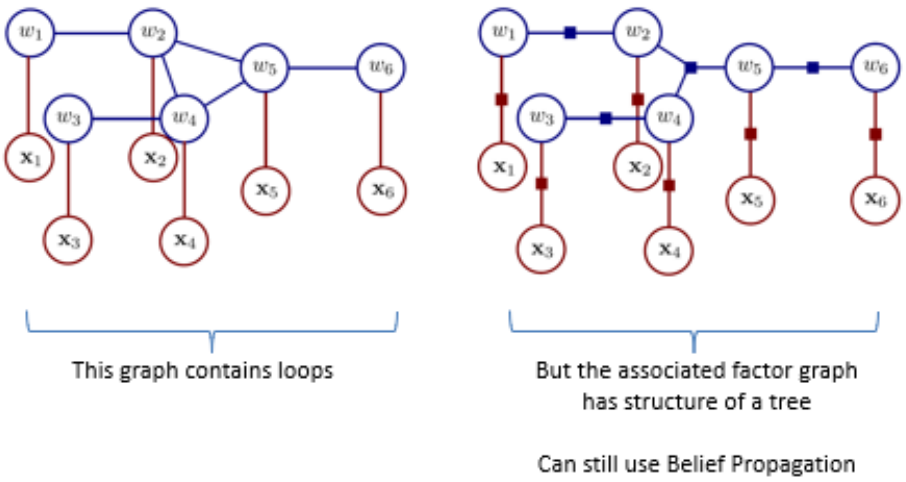
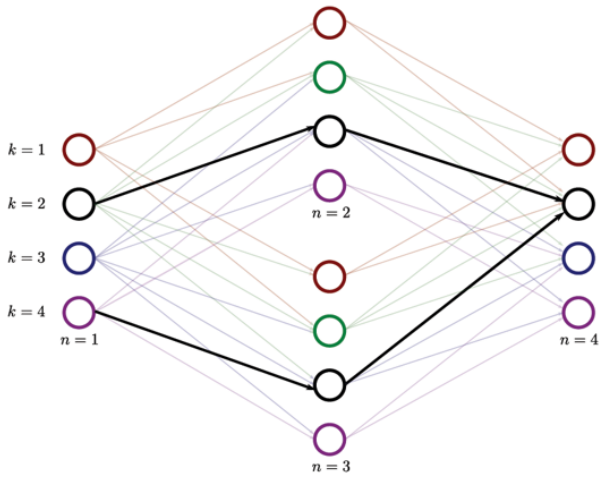



Figure 5.27: Tree Structured Graphs with Loops.

### 5.11.2 Challenges of Loops

**Theorem 5.11.1.** *Dynamic programming cannot be directly applied to models with loops because paths traced back from the final node do not converge to a unique solution, or do not guarantee to converge. This arises due to overlapping subproblems introduced by the loops.*




$$
\begin{aligned}
S_{1,k} &= U_1(w_1 = k) \\
S_{2,k} &= U_2(w_2 = k) + \min_l \left[ S_1(w_1 = l) + P_2(w_2 = k, w_1 = l) \right] \\
S_{3,k} &= U_3(w_3 = k) + \min_l \left[ S_1(w_1 = l) + P_2(w_3 = k, w_1 = l) \right]
\end{aligned}
$$

Figure 5.28: Dynamic Programming Issue with Model with Loops.

**Example 5.11.2.** In the graph below, the paths traced from different final nodes overlap, creating ambiguity:

$$
S_{4,k} \neq U_4(w_4 = k) + \min_{l,m} \left[ S_2(w_2 = l) + S_3(w_3 = m) + T(w_4 = k, w_2 = l, w_3 = m) \right]
$$

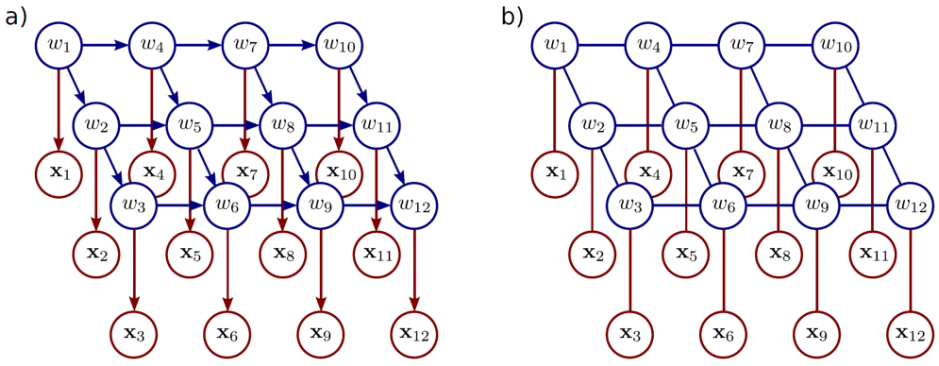



Figure 5.29: Grid-based Graphs.

### 5.11.3 Grid-Based Graphs in Vision

**Remark.** *Grid-based graphs often represent pixels or regions in images, with edges encoding spatial relationships. Common applications include:*

- ***Image segmentation:*** *Assigning labels (e.g., foreground or background) to image pixels.*
- ***Stereo vision:*** *Estimating disparity maps from stereo images.*

*In these cases, loopy belief propagation is a popular inference method due to its efficiency and simplicity.*

### 5.11.4 Approaches to Inference in Grid-Based Models

- **Pruning the graph:** Iteratively remove edges to transform the graph into a tree structure. While this simplifies inference, it may lose critical relationships.
- **Combining variables:** Merge variables into compound nodes, forming a tree structure with higher-dimensional nodes. This method is computationally expensive and impractical for large grids.
- **Loopy belief propagation:** Apply belief propagation to the loopy graph. While not guaranteed to converge, it often performs well in practice.
- **Sampling approaches:** Approximate marginal distributions using Monte Carlo methods. These are computationally feasible for directed models but less so for undirected models.
- **Other techniques:**
  - Tree-reweighted message passing.
  - Graph cuts, which are often used for energy minimization in vision applications.

### 5.11.5 Loopy Belief Propagation

**Definition 5.11.3.** Loopy belief propagation is a generalization of belief propagation for graphs with cycles. It iteratively updates messages between nodes:

$$m_{x\to g} = \prod_{f\in \text{ne}(x)\setminus g} m_{f\to x},$$
$$m_{g\to x} = \sum_{z} \big[g(z) \prod_{y\in \text{ne}(g)\setminus x} m_{y\to g}\big].$$

## 5.12 Applications in Vision

- **Gesture Tracking:** Chain models infer temporal states in sign language sequences.
- **Body Parsing:** Tree models estimate configurations of human body parts.
- **Stereo Vision:** Tree models represent pixel disparities in depth estimation.
- **Segmentation:** Grid-based extensions to chain models assign labels to image regions.

# Chapter 6

# Models for Geometry (Single Camera)

## 6.1 Pinhole Camera Model

**Remark.** *The pinhole camera model is a mathematical abstraction that represents the relationship between points in 3D space and their projections on a 2D image plane (where the sensor is). It is widely used in computer vision for tasks such as calibration, depth estimation, and 3D reconstruction.*

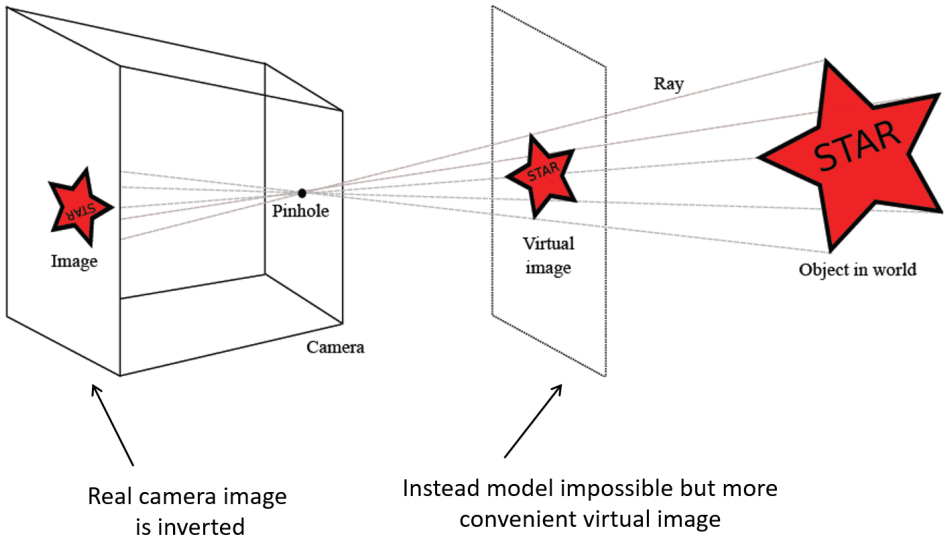



Figure 6.1: Pinhole Camera Illustration.

**Definition 6.1.1** (Terminology)**.** In the pinhole camera model, a point $\mathbf{w} = (u, v, w)^\top$ (object in

real world) in the 3D world maps to a point $\mathbf{x} = (x, y)^\top$ in the 2D image plane using a set of intrinsic and extrinsic parameters:

$$\mathbf{x} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]\mathbf{w},$$

where:

- $\mathbf{K}$: Intrinsic matrix (camera-specific parameters like focal length and skew).
- $\mathbf{R}$: Rotation matrix (orientation of the camera).
- $\mathbf{t}$: Translation vector (position of the camera in the world frame).

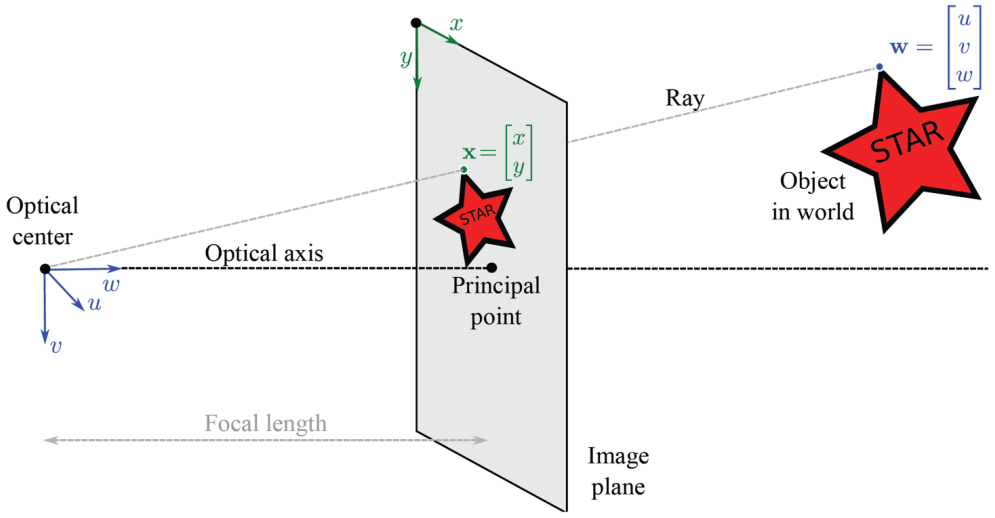



Figure 6.2: Pinhole Camera Terminology.

**Definition 6.1.2** (Normalized Camera)**.**

## 6.2 Intrinsic Parameters

### 6.2.1 Focal Length

**Remark.** *The focal length models both the effect of the distance to the focal plane, and the density of the receptors. It determines how the distance to the focal plane and the density of image receptors affect projection. In practical scenarios:*

- $f_x$ *and* $f_y$ *may differ due to non-uniform receptor spacing.*

- *These are encoded in the intrinsic matrix:*

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

*where $s$ accounts for skew.*

- *When focal length gets shorter, the image also gets smaller.*

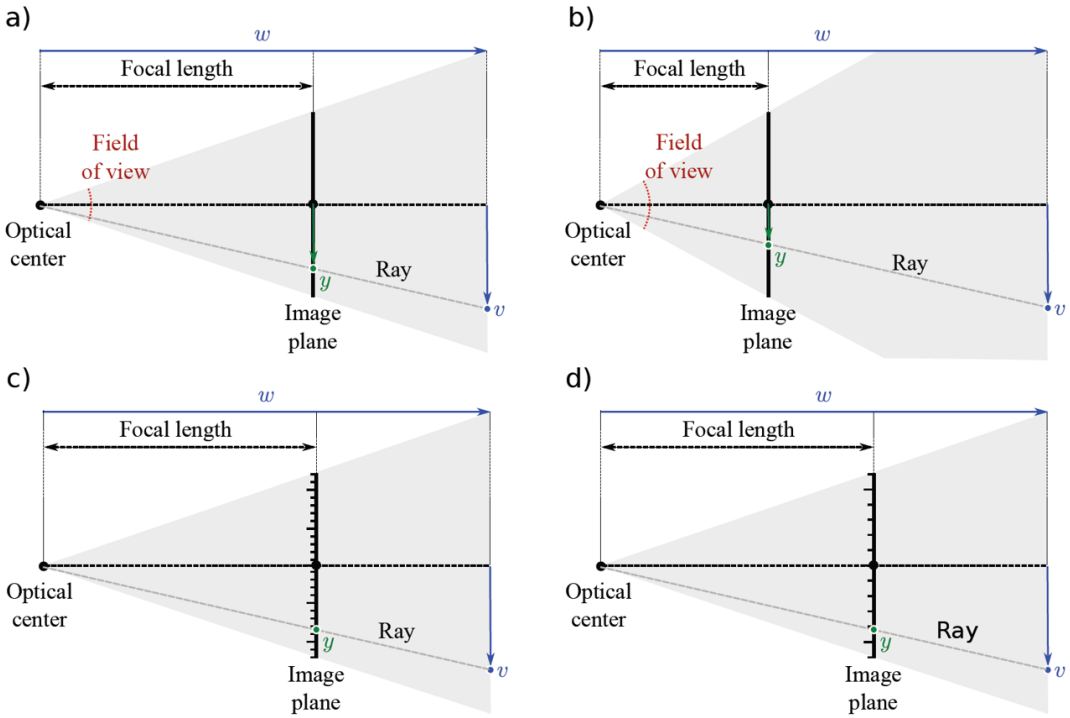



Figure 6.3: Focal Length Parameters.

### 6.2.2 Offset Parameters

**Remark.** *The offset parameters $(\delta_x, \delta_y)$ adjust for the assumption that the principal ray strikes the center of the image plane:*

$$x = \frac{\phi_x u}{w} + \delta_x, \quad y = \frac{\phi_y v}{w} + \delta_y.$$

*These parameters correct for any misalignment between the center of the image plane and the optical axis.*

### 6.2.3 Skew Parameter

**Remark.** *The skew parameter $\gamma$ accounts for the image plane not being exactly perpendicular to the principal ray:*

$$x = \frac{\phi_x u + \gamma v}{w} + \delta_x, \quad y = \frac{\phi_y v}{w} + \delta_y.$$

*This ensures proper alignment of the axes in the image plane.*

## 6.3 Extrinsic Parameters

### 6.3.1 Position and Orientation of the Camera

**Remark.** *A 3D transformation is used to express the world coordinates $(u, v, w)$ in the camera frame:*

$$\mathbf{w}' = \mathbf{\Omega}\mathbf{w} + \tau,$$

*where:*

- $\mathbf{\Omega}$ *is the rotation matrix (orientation).*
- $\tau$ *is the translation vector (position).*

*This transformation maps the point in the world coordinate system to the camera coordinate system.*

## 6.4 Complete Pinhole Camera Model

**Remark.** *Combining the intrinsic and extrinsic parameters gives the complete pinhole camera model:*

$$x = \frac{\phi_x(\omega_{11}u + \omega_{12}v + \omega_{13}w + \tau_x) + \gamma(\omega_{21}u + \omega_{22}v + \omega_{23}w + \tau_y)}{\omega_{31}u + \omega_{32}v + \omega_{33}w + \tau_z} + \delta_x,$$

$$y = \frac{\phi_y(\omega_{21}u + \omega_{22}v + \omega_{23}w + \tau_y)}{\omega_{31}u + \omega_{32}v + \omega_{33}w + \tau_z} + \delta_y.$$

*For simplicity, this is written as:*

$$\mathbf{x} = \mathit{pinhole}[\mathbf{w}, \mathbf{\Lambda}, \mathbf{\Omega}, \tau],$$

*where $\mathbf{\Lambda}$ is the intrinsic matrix, $\mathbf{\Omega}$ is the rotation matrix, and $\tau$ is the translation vector.*

### 6.4.1 Noise in Localization

**Remark.** *To model uncertainty in feature localization, noise is added:*

$$\Pr(\mathbf{x} \mid \mathbf{w}, \mathbf{\Lambda}, \mathbf{\Omega}, \tau) = \mathit{Norm}_{\mathbf{x}}[\mathit{pinhole}(\mathbf{w}, \mathbf{\Lambda}, \mathbf{\Omega}, \tau), \sigma^2].$$

## 6.5 Radial Distortion

### 6.5.1 Correcting Radial Distortion

**Remark.** *Radial distortion introduces nonlinear effects in image projection, which are corrected using the following equations:*

$$x' = x(1 + \beta_1 r^2 + \beta_2 r^4), \quad y' = y(1 + \beta_1 r^2 + \beta_2 r^4),$$

*where $r^2 = x^2 + y^2$. This corrects the bending of straight lines in images due to lens distortion.*

## 6.6 Homogeneous Coordinates

### 6.6.1 Definition

**Definition 6.6.1.** Homogeneous coordinates represent points in $n$-dimensional space as $(n + 1)$-dimensional vectors. For example, a 2D point $[x, y]$ can be written in homogeneous form as:

$$\widetilde{\mathbf{x}} = \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix}.$$

Here, $\lambda$ is a scaling factor. This representation facilitates linear transformations such as translation, scaling, and projection, which are otherwise nonlinear in Cartesian coordinates.

To convert back to Cartesian coordinates, divide by the last element ($\lambda$):

$$x = \frac{\widetilde{x}}{\widetilde{z}}, \quad y = \frac{\widetilde{y}}{\widetilde{z}}.$$

This conversion is essential for interpreting homogeneous coordinates in standard Euclidean space.

### 6.6.2 Geometric Interpretation

**Theorem 6.6.2.** *A point in homogeneous coordinates $\widetilde{\mathbf{x}}$ represents all points along a line through the origin in $(n+1)$-dimensional space. For a 2D point $[x, y]$, its homogeneous representation projects onto a reference plane $z = 1$, maintaining proportionality:*

$$\widetilde{\mathbf{x}} = \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix}, \quad \textit{projects to} \quad \begin{bmatrix} x \\ y \end{bmatrix}.$$

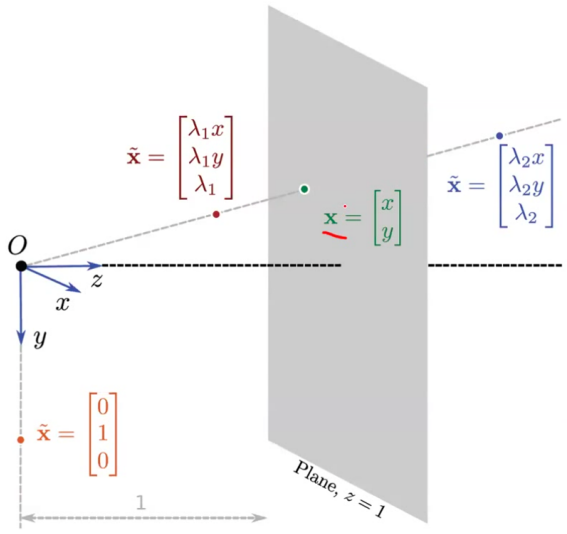



Figure 6.4: Geometric Interpretation of Homogeneous Coordinates.

### 6.6.3 Pinhole Camera in Homogeneous Coordinates

**Definition 6.6.3.** The pinhole camera model describes the relationship between 3D world coordinates and 2D image coordinates. The original camera model is :

$$x = \frac{\phi_x u + \gamma v}{w} + \delta_x,$$

$$y = \frac{\phi_y v}{w} + \delta_y.$$

In homogeneous form, the mapping is linear:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & 0 \\ 0 & \phi_y & \delta_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}.$$

Writing out these three equations:

$$\lambda x = \phi_x u + \gamma v + \delta_x w,$$

$$\lambda y = \phi_y v + \delta_y w,$$

$$\lambda = w.$$

**Remark.** *The matrix decomposition above includes:*

- *Intrinsic parameters ($\phi_x, \phi_y, \gamma, \delta_x, \delta_y$), which account for focal length, skew, and principal point.*
- *Extrinsic parameters ($\mathbf{R}, \mathbf{t}$), representing the camera's orientation and position in the world.*

### 6.6.4 Incorporating Extrinsic Parameters

**Theorem 6.6.4.** *Including rotation and translation (extrinsic parameters), the camera model can be extended as:*

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & 0 \\ 0 & \phi_y & \delta_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & t_x \\ \omega_{21} & \omega_{22} & \omega_{23} & t_y \\ \omega_{31} & \omega_{32} & \omega_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}.$$

*The extrinsic parameters can also be expressed in a compact form:*

$$\lambda \widetilde{\mathbf{x}} = \begin{bmatrix} \mathbf{\Lambda} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{\Omega} & \tau \\ 0^T & 1 \end{bmatrix} \widetilde{\mathbf{w}},$$

*or even more concisely:*

$$\lambda \widetilde{\mathbf{x}} = \mathbf{\Lambda} \begin{bmatrix} \mathbf{\Omega} & \tau \end{bmatrix} \widetilde{\mathbf{w}},$$

*where:*

- $\mathbf{\Lambda}$ *is the intrinsic parameter matrix.*
- $\mathbf{\Omega}$ *is the rotation matrix.*
- $\tau$ *is the translation vector.*

## 6.7 Geometric Problems in Camera Models

### 6.7.1 Problem 1: Exterior Orientation

**Definition 6.7.1.** Exterior orientation involves estimating the extrinsic parameters $(\mathbf{R}, \mathbf{t})$ of the camera. The maximum likelihood is:

$$\hat{\Omega}, \hat{\tau} = \arg\max_{\Omega, \tau} \sum_{i=1}^{I} \log \left[ \Pr(\mathbf{x}_i \mid \mathbf{w}_i, \Lambda, \Omega, \tau) \right]$$

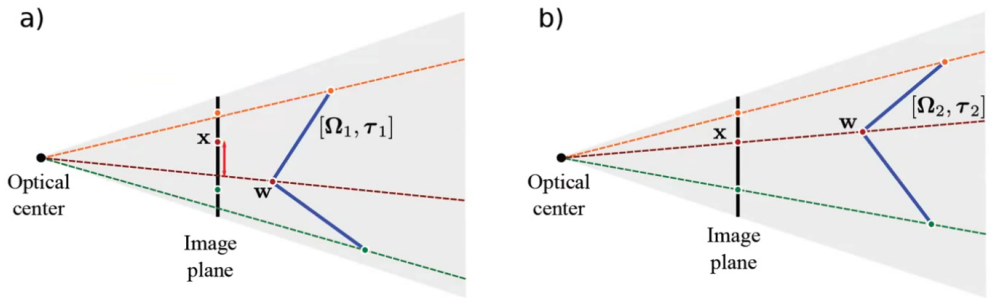



Figure 6.5: Learning extrinsic parameters (Exterior Orientation).

**Theorem 6.7.2** (Homogeneous Coordinates Solution - Exterior Orientation)**.** *1.* ***Start with the camera equation in homogeneous coordinates:***

$$\lambda_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & 0 \\ 0 & \phi_y & \delta_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & t_x \\ \omega_{21} & \omega_{22} & \omega_{23} & t_y \\ \omega_{31} & \omega_{32} & \omega_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ 1 \end{bmatrix}.$$

*2.* ***Pre-multiply both sides by the inverse of the camera calibration matrix:***

$$\lambda_i \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & t_x \\ \omega_{21} & \omega_{22} & \omega_{23} & t_y \\ \omega_{31} & \omega_{32} & \omega_{33} & t_z \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ 1 \end{bmatrix}.$$

*3.* ***Solve for*** $\lambda_i$***:*** *From the third row of the equation, express* $\lambda_i$ *as:*

$$\lambda_i = \omega_{31} u_i + \omega_{32} v_i + \omega_{33} w_i + t_z.$$

*4.* ***Substitute*** $\lambda_i$ ***back into the first two rows:***

$$\begin{bmatrix} (\omega_{31} u_i + \omega_{32} v_i + \omega_{33} w_i + t_z) x_i' \\ (\omega_{31} u_i + \omega_{32} v_i + \omega_{33} w_i + t_z) y_i' \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & t_x \\ \omega_{21} & \omega_{22} & \omega_{23} & t_y \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ 1 \end{bmatrix}.$$

*5.* ***Formulate the system of linear equations:*** *Each point* $(x_i', y_i')$ *contributes two equations. For*

*$N$ points, the system of equations is:*

$$\begin{bmatrix} u_1 & v_1 & w_1 & 1 & 0 & 0 & 0 & 0 & -x'_1u_1 & -x'_1v_1 & -x'_1w_1 & -x'_1 \\ 0 & 0 & 0 & 0 & u_1 & v_1 & w_1 & 1 & -y'_1u_1 & -y'_1v_1 & -y'_1w_1 & -y'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_N & v_N & w_N & 1 & 0 & 0 & 0 & 0 & -x'_Nu_N & -x'_Nv_N & -x'_Nw_N & -x'_N \\ 0 & 0 & 0 & 0 & u_N & v_N & w_N & 1 & -y'_Nu_N & -y'_Nv_N & -y'_Nw_N & -y'_N \end{bmatrix} \begin{bmatrix} \omega_{11} \\ \omega_{12} \\ \omega_{13} \\ t_x \\ \omega_{21} \\ \omega_{22} \\ \omega_{23} \\ t_y \\ \omega_{31} \\ \omega_{32} \\ \omega_{33} \\ t_z \end{bmatrix} = 0.$$

6. ***Solve using Singular Value Decomposition (SVD):*** *The system $\mathbf{Ab} = 0$ represents a minimum direction problem. To solve, minimize $\|\mathbf{Ab}\|^2$ subject to $\|\mathbf{b}\| = 1$. Compute the SVD:*

$$\mathbf{A} = \mathbf{ULV}^T,$$

*and set $\mathbf{b}$ to the last column of $\mathbf{V}$.*

7. ***Extract the parameters $\mathbf{\Omega}$ and $\tau$:***

   - *The rotation matrix $\mathbf{\Omega}$ can be extracted using:*

   $$\mathbf{\Omega} = \mathbf{UV}^T.$$

   - *The translation vector $\tau$ is scaled using the ratio between the rotation matrix before and after rescaling:*

   $$\tau' = \frac{\sum_{m=1}^{3}\sum_{n=1}^{3}\hat{\Omega}_{mn}\Omega_{mn}}{\sum_{m=1}^{3}\sum_{n=1}^{3}\Omega_{mn}^2}\tau.$$

8. ***Refine the estimates:*** *Use the extracted $\mathbf{\Omega}$ and $\tau$ as the starting point for a nonlinear optimization process to improve the solution.*

### 6.7.2 Problem 2: Camera Calibration

**Definition 6.7.3.** Calibration involves estimating intrinsic parameters $\mathbf{K}$. The maximum likelihood is:

$$\hat{\Lambda} = \arg\max_{\Lambda}\left[\max_{\Omega,\tau}\sum_{i=1}^{I}\log \Pr(\mathbf{x}_i \mid \mathbf{w}_i, \Lambda, \Omega, \tau)\right]$$

where you need to solve the extrinsic first.

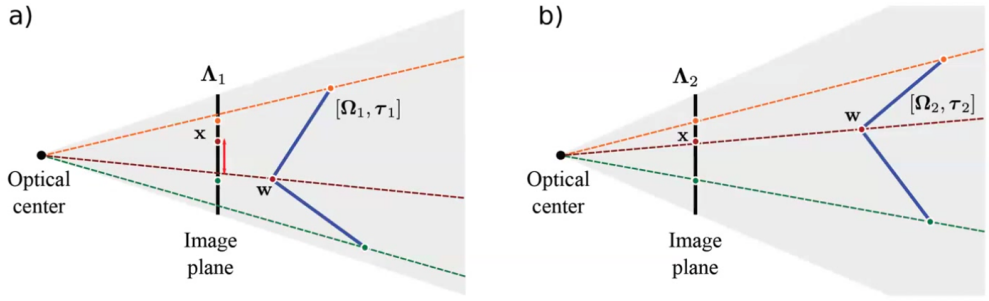



Figure 6.6: Learning intrinsic parameters (Calibration).

**Theorem 6.7.4** (Homogeneous Coordinates Solution - Camera Calibration)**.** *The calibration problem can be formulated as a maximum likelihood problem, where both intrinsic and extrinsic parameters are optimized. This section details the solution process:*

**Maximum Likelihood Formulation** *The likelihood maximization is expressed as:*

$$\hat{\Lambda} = \arg\max_{\Lambda} \left[ \max_{\Omega,\tau} \sum_{i=1}^{I} \log \Pr(\mathbf{x}_i \mid \mathbf{w}_i, \Lambda, \Omega, \tau) \right],$$

*where:*

- $\mathbf{x}_i$*: Observed 2D image points.*
- $\mathbf{w}_i$*: Known 3D world points.*
- $\Lambda$*: Intrinsic parameters of the camera.*
- $\Omega$*,* $\tau$*: Extrinsic parameters (rotation and translation).*

**Intrinsic Parameter Optimization** *Substituting the pinhole camera model:*

$$\mathbf{x}_i = \textit{pinhole}(\mathbf{w}_i, \Lambda, \Omega, \tau),$$

*and assuming Gaussian noise in image point observations:*

$$\Pr(\mathbf{x}_i \mid \mathbf{w}_i, \Lambda, \Omega, \tau) = \textit{Norm}_x \left( \textit{pinhole}(\mathbf{w}_i, \Lambda, \Omega, \tau), \sigma^2 \mathbf{I} \right),$$

*the maximum likelihood estimate becomes a least squares problem:*

$$\hat{\Lambda} = \arg\min_{\Lambda} \sum_{i=1}^{I} \left\| \mathbf{x}_i - \textit{pinhole}(\mathbf{w}_i, \Lambda, \Omega, \tau) \right\|^2.$$

**Linearization via Homogeneous Coordinates** *The pinhole function is linearized with respect to the intrinsic parameters $\Lambda$ using homogeneous coordinates. This leads to the following relation:*

$$\mathbf{A}_i \mathbf{h} = \mathbf{x}_i,$$

*where:*

- $\mathbf{A}_i$*: Camera projection matrix derived from known world coordinates* $\mathbf{w}_i$*.*
- $\mathbf{h}$*: Vector of intrinsic parameters, represented as:*

$$\mathbf{h} = [f_x, \gamma, c_x, f_y, c_y, \delta_x, \delta_y]^T.$$

**Global System Assembly** *Collecting all equations for $I$ observed points, the system can be written as:*

$$\mathbf{H}\mathbf{h} = \mathbf{x},$$

*where:*

$$\mathbf{H} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \vdots \\ \mathbf{A}_I \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_I \end{bmatrix}.$$

**Least Squares Solution** *The intrinsic parameters $\mathbf{h}$ are estimated by solving the least squares problem:*

$$\hat{\mathbf{h}} = \arg\min_{\mathbf{h}} \|\mathbf{H}\mathbf{h} - \mathbf{x}\|^2.$$

*The closed-form solution is given by:*

$$\hat{\mathbf{h}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{x}.$$

**Full Camera Model with Extrinsic Parameters** *For completeness, the camera model integrates both intrinsic and extrinsic parameters:*

$$\mathbf{x} = \Lambda \begin{bmatrix} \Omega & \tau \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{w},$$

*where:*

- $\Omega$*: 3x3 rotation matrix encoding camera orientation.*
- $\tau$*: 3x1 translation vector encoding camera position.*
- $\Lambda$*: Intrinsic parameter matrix, including focal lengths, principal points, and skew.*

**Iterative Refinement** *In practice, calibration involves alternating optimization of extrinsic $(\Omega, \tau)$ and intrinsic $(\Lambda)$ parameters:*

$$\hat{\Omega},$$

### 6.7.3 Problem 3: 3D Reconstruction

**Remark.** *Reconstruction estimates 3D points* $\mathbf{w}$ *from multiple 2D projections. The maximum likelihood is:*

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \left( \sum_{j=1}^{J} \log \Pr(\mathbf{x}_j \mid \mathbf{w}, \Lambda_j, \Omega_j, \tau_j) \right)$$

*This requires:*

- *At least two cameras (stereo vision).*
- *Optimization to minimize reprojection error.*

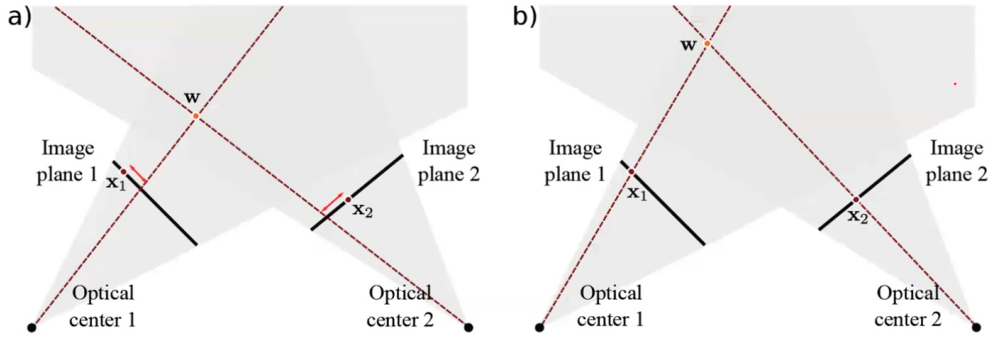



Figure 6.7: Inferring 3D points (Triangulation/Reconstruction).

**Theorem 6.7.5** (Homogeneous Coordinates Solution - 3D Reconstruction)**.** *To reconstruct a 3D point* $\mathbf{w} = [u, v, w]^T$ *in the world coordinate system using multiple camera views, the following steps are performed:*

1. ***Camera Projection in Homogeneous Coordinates:*** *The $j$-th pinhole camera projects a 3D point* $\mathbf{w}$ *in the world frame to a 2D image point* $\mathbf{x}_j = [x_j, y_j]^T$ *as:*

$$\lambda_j \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & 0 \\ 0 & \phi_y & \delta_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}.$$

2. ***Intrinsic Matrix Inversion:*** *Pre-multiply both sides by the inverse of the intrinsic matrix $\Lambda^{-1}$ to simplify the equations and normalize the image coordinates:*

$$\lambda_j \begin{bmatrix} x'_j \\ y'_j \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}.$$

3. ***Solving for Scale $\lambda_j$:*** *The third row of the equation above gives:*

$$\lambda_j = \omega_{31}u + \omega_{32}v + \omega_{33}w + \tau_z.$$

4. ***Substitution and Linear Equations:*** *Substitute $\lambda_j$ back into the first two rows to derive two equations for each camera:*

$$(\omega_{11}u + \omega_{12}v + \omega_{13}w + \tau_x) - x'_j(\omega_{31}u + \omega_{32}v + \omega_{33}w + \tau_z) = 0,$$
$$(\omega_{21}u + \omega_{22}v + \omega_{23}w + \tau_y) - y'_j(\omega_{31}u + \omega_{32}v + \omega_{33}w + \tau_z) = 0.$$

5. ***System of Linear Equations:*** *Using multiple cameras ($J > 1$), construct a linear system of equations:*

$$\mathbf{A}\mathbf{w} = \mathbf{b},$$

*where $\mathbf{A}$ encodes camera parameters and $\mathbf{b}$ encodes observed image coordinates.*

6. ***Non-Linear Optimization:*** *After obtaining an initial estimate of $\mathbf{w}$, refine it by minimizing a reprojection error:*

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{j=1}^{J} \|\mathbf{x}_j - \textit{proj}(\mathbf{w}; \Lambda_j, \Omega_j, \tau_j)\|^2,$$

*where proj$(\cdot)$ is the projection function.*

*The method allows for accurate reconstruction of 3D points provided sufficient views are available.*

## 6.8 Applications

### 6.8.1 Depth Estimation

**Remark.** *Depth estimation reconstructs the 3D structure of a scene using:*

- *Stereo vision.*
- *Structured light systems.*

*Example: Sparse stereo reconstruction computes depth at sparse matching points.*

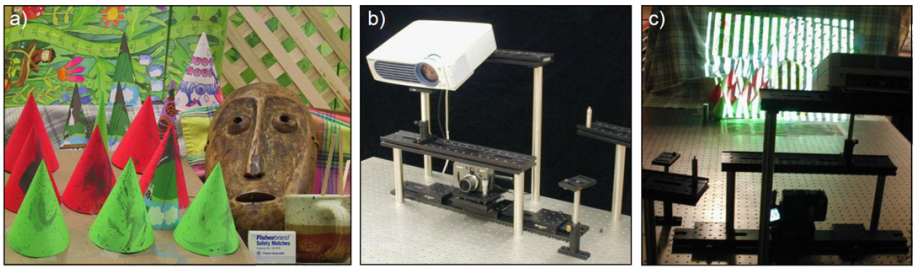



**Figure 13.12** Depth maps from structured light. a) A three-dimensional scene that we wish to capture. b) The capture hardware consists of a projector and a camera, which both view the scene from different positions. c) The projector is used to illuminate the scene and the camera records the pattern of illumination from its viewpoint. The resulting images contain information that can be used to compute a 3D reconstruction. Adapted from Scharstein & Szeliski (2003). ©2003 IEEE.

Figure 6.8: Depth from Structued Light.

### 6.8.2 Shape from Silhouette

**Remark.** *Reconstruct 3D shapes by combining multiple silhouettes of an object observed from different viewpoints.*

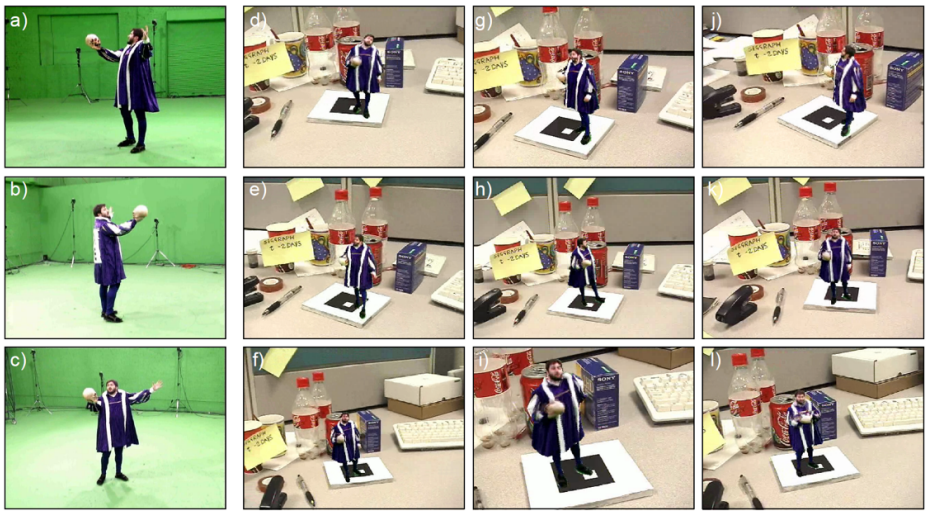



Figure 6.9: Shape from Silhouette.

## 6.9 Models for Transformations

### 6.9.1 Euclidean Transformations

**Definition 6.9.1.** Euclidean transformations preserve angles and lengths, involving rotation and translation:

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t},$$

where:

- $\mathbf{R}$: Rotation matrix.
- $\mathbf{t}$: Translation vector.

**Remark.** *In homogeneous coordinates, Euclidean transformations can be expressed compactly as:*

$$\mathbf{x}' = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \mathbf{x},$$

*where* $\mathbf{x}$ *and* $\mathbf{x}'$ *are augmented into homogeneous coordinates by appending a 1 as the last element.*

**Example 6.9.2** (Fronto-parallel Plane Transformation)**.** Consider a fronto-parallel plane at a known distance $D$ from the camera. Using homogeneous coordinates, the imaging equations can be writ-

ten as:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & \tau_x \\ 0 & \phi_y & \delta_y & \tau_y \\ 0 & 0 & 1/D & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 0 \\ 1 \end{bmatrix}.$$

Here:

- The $3D$ rotation matrix is reduced to $2D$ for points in the plane.
- $D$: Represents the distance of the plane from the camera.
- Points on the plane satisfy $w = 0$.

Simplifying and rearranging, the above equation becomes:

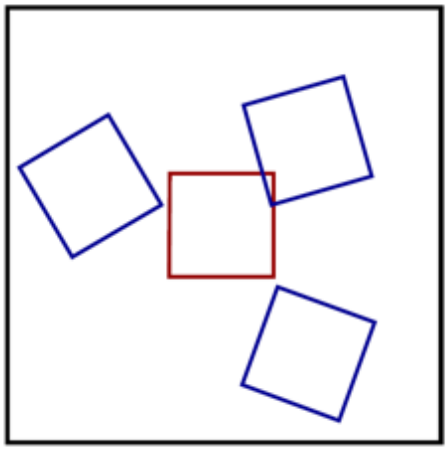
$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & \tau_x \\ 0 & \phi_y & \delta_y & \tau_y \\ 0 & 0 & 1/D & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 0 \\ 1 \end{bmatrix}.$$

By pre-multiplying with the inverse of the intrinsic matrix, we can simplify to (Homogeneous version):

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \tau_x \\ \omega_{21} & \omega_{22} & \tau_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

or Cartesion verision:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} \\ \omega_{21} & \omega_{22} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix}.$$



Homogeneous:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \tau_x \\ \omega_{21} & \omega_{22} & \tau_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Cartesian:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} \\ \omega_{21} & \omega_{22} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix}$$

For short:

$$\mathbf{x}' = \mathbf{euc}[\mathbf{w}, \boldsymbol{\Omega}, \boldsymbol{\tau}]$$

Figure 6.10: Euclidean Transformation Summary.

### 6.9.2 Similarity Transformations

**Definition 6.9.3.** Similarity transformations extend Euclidean transformations by adding a uniform scaling factor $s$:

$$\mathbf{x}' = s\mathbf{R}\mathbf{x} + \mathbf{t}.$$

**Remark.** *In homogeneous coordinates, this can be written as:*

$$\mathbf{x}' = s \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \mathbf{x}.$$

**Example 6.9.4** (Fronto-parallel Plane Transformation with Similarity Scaling)**.** Consider viewing a fronto-parallel plane at an unknown distance $D$. Using the same logic as Euclidean transformations, the imaging equations in homogeneous coordinates become:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & \tau_x \\ 0 & \phi_y & \delta_y & \tau_y \\ 0 & 0 & 1/D & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 0 \\ 1 \end{bmatrix}.$$

Pre-multiplying by the inverse of the intrinsic matrix gives:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \tau_x \\ \omega_{21} & \omega_{22} & \tau_y \\ 0 & 0 & 1/D \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

Simplifying, multiply by $\rho = 1/D$ to introduce uniform scaling:

$$\rho\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \rho\omega_{11} & \rho\omega_{12} & \rho\tau_x \\ \rho\omega_{21} & \rho\omega_{22} & \rho\tau_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

By incorporating constants ($\tau_x \leftarrow \rho\tau_x$, $\tau_y \leftarrow \rho\tau_y$, $\lambda \leftarrow \rho\lambda$), we write:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \rho\omega_{11} & \rho\omega_{12} & \tau_x \\ \rho\omega_{21} & \rho\omega_{22} & \tau_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

**Remark.** *The final form of the similarity can be expressed as (homogeneous version):*

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \rho\omega_{11} & \rho\omega_{12} & \tau_x \\ \rho\omega_{21} & \rho\omega_{22} & \tau_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

*In Cartesian coordinates, this simplifies to:*

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \rho\omega_{11} & \rho\omega_{12} \\ \rho\omega_{21} & \rho\omega_{22} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix}.$$

*For short, this is written as:*

$$\mathbf{x}' = sim[\mathbf{w}, \mathbf{\Omega}, \boldsymbol{\tau}, \rho].$$

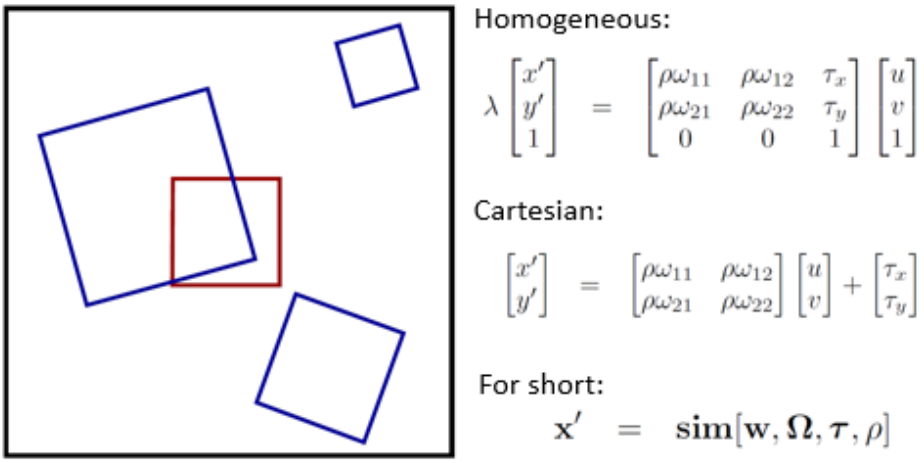



Figure 6.11: Similarity Transformation Summary.

### 6.9.3 Affine Transformations

**Definition 6.9.5.** Affine transformations generalize similarity transformations by allowing non-uniform scaling and shearing:

$$\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{t},$$

where $\mathbf{A}$ is a $2 \times 2$ matrix that encodes scaling, rotation, and shearing, and $\mathbf{t}$ is the translation vector.

**Remark.** *In homogeneous coordinates, affine transformations can be expressed as:*

$$\mathbf{x}' = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \mathbf{x}.$$

**Example 6.9.6** (Affine Transformation of a Plane)**.** When transforming a plane, the equations in homogeneous coordinates are given by:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \tau_x \\ \phi_{21} & \phi_{22} & \tau_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

In Cartesian coordinates, this simplifies to:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix}.$$

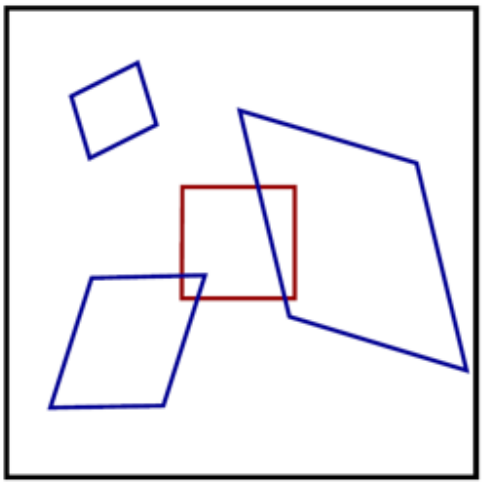



Figure 6.12: Affine Transformation Summary.

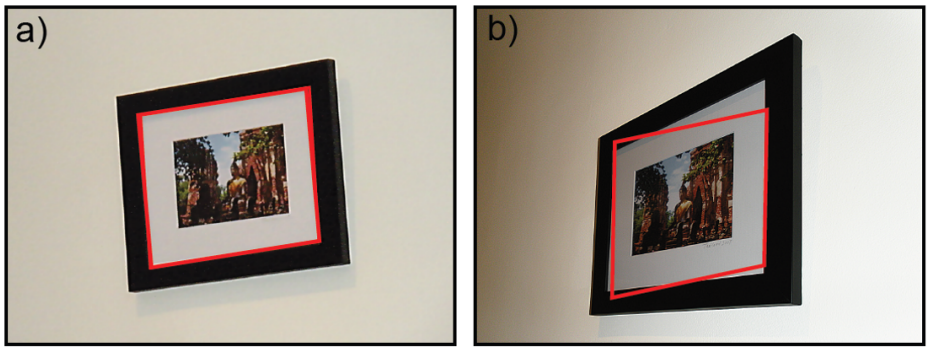



Figure 6.13: Use Cases of Affine Transformation.

**Remark** (Final Form: Homogeneous and Cartesian Representations)**.** *The affine transformation can be represented as:*

- ***Homogeneous Representation:***

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \tau_x \\ \phi_{21} & \phi_{22} & \tau_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

- ***Cartesian Representation:***

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix}.$$

### 6.9.4 Projective/Collinearity/Homographies Transformation

**Definition 6.9.7.** A homography is a projective transformation that maps points between planes in 3D space. It is the most general 2D transformation:

$$\mathbf{x}' \sim \mathbf{H}\mathbf{x},$$

where $\mathbf{H}$ is a $3 \times 3$ matrix and $\sim$ denotes equivalence up to scale.

**Remark.** *Homographies are used to model transformations where depth variation is comparable to the camera distance, making affine models insufficient.*

**Example 6.9.8** (Homography Equation)**.** Starting from the basic projection equation:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ 0 \\ 1 \end{bmatrix}.$$

Combining the matrices, this reduces to:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

**Remark** (Final Form: Homogeneous and Cartesian Representations)**.** *The homography transformation can be expressed as:*

- ***Homogeneous Representation:***

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

- ***Cartesian Representation:***

$$x = \frac{\phi_{11}u + \phi_{12}v + \phi_{13}}{\phi_{31}u + \phi_{32}v + \phi_{33}}, \quad y = \frac{\phi_{21}u + \phi_{22}v + \phi_{23}}{\phi_{31}u + \phi_{32}v + \phi_{33}}.$$

**Remark** (Application and Notation)**.** *For short, the homography can be denoted as:*

$$\mathbf{x}' = hom[\mathbf{w}, \mathbf{\Phi}].$$

*Homographies are critical in computer vision tasks like image rectification, perspective correction, and plane alignment.*

## 6.10 Learning Transformation Models

### 6.10.1 Maximum Likelihood Estimation

**Definition 6.10.1.** Learning transformation parameters involves estimating the transformation $\mathbf{T}$ that minimizes the reprojection error. For a set of corresponding points $\{\mathbf{x}_i, \mathbf{x}_i'\}$, this can be formulated as:

$$\mathcal{L}(\mathbf{T}) = \sum_{i=1}^{N} \|\mathbf{x}_i' - \mathbf{T}\mathbf{x}_i\|^2.$$

**Remark.** *This optimization problem becomes a least-squares problem for specific transformations, including Euclidean, similarity, and affine models.*

### 6.10.2 Learning Euclidean Parameters

**Definition 6.10.2.** For Euclidean transformations, the parameters consist of a rotation matrix $\mathbf{\Omega}$ and a translation vector $\tau$. The problem is:

$$\hat{\mathbf{\Omega}}, \hat{\tau} = \arg\min_{\mathbf{\Omega},\tau} \sum_{i=1}^{I} \|\mathbf{x}_i - \mathbf{\Omega}\mathbf{w}_i - \tau\|^2,$$

where $\mathbf{x}_i$ are the image points and $\mathbf{w}_i$ are the corresponding world points.

- *Solve for $\tau$ using:*

  $$\hat{\tau} = \mu_x - \hat{\mathbf{\Omega}}\mu_w,$$

  *where $\mu_x$ and $\mu_w$ are the centroids of the image and world points.*

- *Solve for $\mathbf{\Omega}$ using the Procrustes problem:*

  $$\hat{\mathbf{\Omega}} = \mathbf{V}\mathbf{U}^\top,$$

  *where $\mathbf{U}$ and $\mathbf{V}$ are obtained via the Singular Value Decomposition (SVD) of the covariance matrix $\mathbf{B}$.*

### 6.10.3 Learning Similarity Parameters

**Definition 6.10.3.** The similarity transformation includes an additional scaling factor $\rho$. The optimization problem is:

$$\hat{\mathbf{\Omega}}, \hat{\tau}, \hat{\rho} = \arg\min_{\mathbf{\Omega},\tau,\rho} \sum_{i=1}^{I} \|\mathbf{x}_i - \rho\mathbf{\Omega}\mathbf{w}_i - \tau\|^2.$$

- *Solve for $\tau$ and $\mathbf{\Omega}$ as in the Euclidean case.*

- *Solve for $\rho$ using:*

  $$\hat{\rho} = \frac{\sum_{i=1}^{I} (\mathbf{x}_i - \tau)^\top \hat{\mathbf{\Omega}} (\mathbf{w}_i - \mu_w)}{\sum_{i=1}^{I} \|\mathbf{w}_i - \mu_w\|^2}.$$

### 6.10.4 Learning Affine Parameters

**Definition 6.10.4.** For affine transformations, the transformation is linear:

$$\mathbf{x}_i = \mathbf{\Phi}\mathbf{w}_i + \tau,$$

where $\mathbf{\Phi}$ is a $2 \times 2$ matrix. The optimization becomes:

$$\hat{\mathbf{\Phi}}, \hat{\tau} = \arg\min_{\mathbf{\Phi},\tau} \sum_{i=1}^{I} \|\mathbf{x}_i - \mathbf{\Phi}\mathbf{w}_i - \tau\|^2.$$

**Remark.** *The problem reduces to solving:*

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}} \|\mathbf{A}\mathbf{b} - \mathbf{x}\|^2,$$

*where* $\mathbf{b}$ *is the concatenated vector of* $\mathbf{\Phi}$ *and* $\tau$*, and* $\mathbf{A}$ *encodes the input data.*

### 6.10.5 Learning Homography Parameters

**Definition 6.10.5.** Homography is a projective transformation represented as:

$$\lambda \mathbf{x}' = \mathbf{\Phi}\mathbf{x}.$$

The optimization problem is nonlinear and involves converting points to homogeneous coordinates.

**Remark.** *The key steps are:*

- *Write the constraints in the form:*
$$\mathbf{x}_i' \times (\mathbf{\Phi}\mathbf{x}_i) = 0,$$
*which leads to a set of linear equations.*
- *Solve using Singular Value Decomposition (SVD) on the matrix* $\mathbf{A}$ *formed by stacking these equations.*
- *Refine the solution using nonlinear optimization to minimize the reprojection error:*

$$\mathcal{L}(\mathbf{\Phi}) = \sum_{i=1}^{I} \|\mathbf{x}_i' - \mathbf{\Phi}\mathbf{x}_i\|^2.$$

## 6.11 Inference in Transformation Models

### 6.11.1 Mapping Image Points to World Points

**Definition 6.11.1.** Inference involves determining the corresponding world point $\mathbf{w}$ from a given image point $\mathbf{x}$, given the transformation model $\mathbf{T}$:

$$\mathbf{w} = \arg\max_{\mathbf{w}} \log \Pr(\mathbf{x} \mid \mathbf{w}, \theta).$$

For a deterministic transformation model, this reduces to minimizing the reprojection error:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \|\mathbf{x} - \text{trans}(\mathbf{w}, \theta)\|^2,$$

where $\text{trans}(\mathbf{w}, \theta)$ represents the forward transformation.

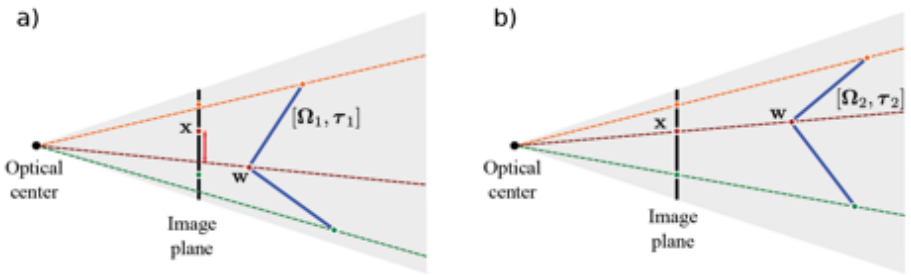



Figure 6.14: Exterior Orientation (3D).

### 6.11.2 Homogeneous Coordinates Inference

**Remark.** *In the absence of noise, the relation between image points* $\mathbf{x}$ *and world points* $\mathbf{w}$ *can be expressed as:*

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

*Here,* $\lambda$ *is a scalar representing the scale factor.*

### 6.11.3 Point Recovery through Matrix Inversion

**Remark.** *To solve for the corresponding world point* $\mathbf{w} = [u, v, 1]^\top$ *in homogeneous coordinates, we invert the relation:*

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

*This requires computing the inverse of the transformation matrix* $\mathbf{\Phi}$.

## 6.12 Transformation Models for Exterior Orientation

### 6.12.1 Problem Formulation

**Definition 6.12.1.** Exterior orientation refers to estimating the position and orientation of the camera in the world coordinate system. The parameters $\Omega$ (rotation matrix) and $\tau$ (translation vector) describe the transformation from the world to the camera coordinate system. The optimization problem is:

$$\hat{\Omega}, \hat{\tau} = \arg\max_{\Omega,\tau} \sum_{i=1}^{I} \log \Pr(\mathbf{x}_i \mid \mathbf{w}_i, \Lambda, \Omega, \tau),$$

where $\Lambda$ represents the intrinsic parameters of the camera, and $\Pr(\mathbf{x}_i \mid \mathbf{w}_i, \Lambda, \Omega, \tau)$ models the likelihood of observing $\mathbf{x}_i$ given the world point $\mathbf{w}_i$ and camera parameters.

### 6.12.2 Homogeneous Coordinates

**Remark.** *The camera equation in homogeneous coordinates can be written as:*

$$\lambda_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ 1 \end{bmatrix}.$$

*Here:*

- $\lambda_i$ *is a scalar ensuring the homogeneous coordinates are consistent.*
- *The intrinsic matrix* $\Lambda$ *maps world coordinates to normalized camera coordinates.*
- *The extrinsic parameters* $\Omega$ *and* $\tau$ *handle rotation and translation.*

### 6.12.3 Decoupling Intrinsic and Extrinsic Parameters

**Remark.** *Pre-multiplying both sides of the camera equation by* $\Lambda^{-1}$ *isolates the extrinsic parameters:*

$$\lambda_i \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ w_i \\ 1 \end{bmatrix}.$$

*The transformed coordinates* $\mathbf{x}_i'$ *represent points in normalized camera space.*

### 6.12.4 Linear System of Equations

**Remark.** *From the third row of the matrix equation:*

$$\lambda_i = \omega_{31}u_i + \omega_{32}v_i + \omega_{33}w_i + \tau_z.$$

*Substituting $\lambda_i$ into the first two rows yields a linear system for each matched point:*

$$\begin{bmatrix} u_1 & v_1 & w_1 & 1 & 0 & 0 & 0 & 0 & -u_1x_1' & -v_1x_1' \\ -w_1x_1' & -x_1' & & & & & & & & \\ 0 & 0 & 0 & 0 & u_1 & v_1 & w_1 & 1 & -u_1y_1' & -v_1y_1' \\ -w_1y_1' & -y_1' & & & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & & & & & & & & \end{bmatrix} \begin{bmatrix} \omega_{11} \\ \omega_{12} \\ \omega_{13} \\ \tau_x \\ \omega_{21} \\ \omega_{22} \\ \omega_{23} \\ \tau_y \\ \omega_{31} \\ \omega_{32} \\ \omega_{33} \\ \tau_z \end{bmatrix} = 0.$$

*This results in two equations per point, forming a system of equations for all matched points.*

### 6.12.5 Solving the System

**Theorem 6.12.2** (Exterior Orientation Solution)**.** *The linear system can be written in the form:*

$$\mathbf{A}\mathbf{b} = 0,$$

*where $\|\mathbf{b}\| = 1$. To solve:*

1. *Compute the Singular Value Decomposition (SVD) of $\mathbf{A}$:*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top.$$

2. *Set $\mathbf{b}$ to the last column of $\mathbf{V}$, corresponding to the smallest singular value.*

### 6.12.6 Extracting Rotation and Translation

**Remark.** *Extract $\Omega$ (rotation matrix) and $\tau$ (translation vector) from $\mathbf{b}$. Steps:*

1. *The first three columns of $\Omega$ are estimated using:*

$$\Omega = \mathbf{U}\mathbf{V}^\top,$$

   *ensuring orthogonality.*

2. *The last column of* $\Omega$ *is computed as the cross-product of the first two columns.*

3. *Ensure* $\det(\Omega) = 1$ *by flipping the sign of the last column if necessary.*

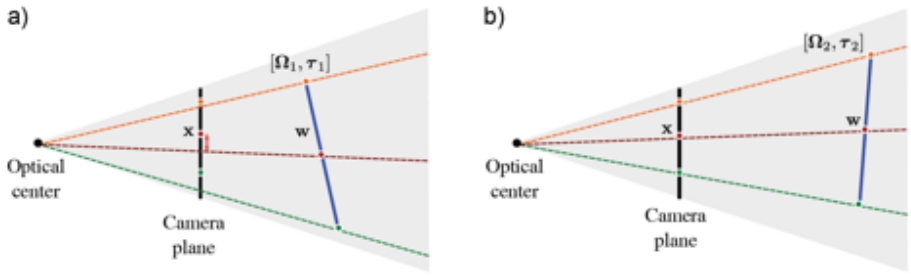
*Compute the scaling factor* $\lambda'$*:*

$$\lambda' = \frac{1}{6} \sum_{m=1}^{3} \sum_{n=1}^{3} \frac{\phi'_{mn}}{\omega_{mn}}.$$

*Finally, the translation vector is:*

$$\tau = \frac{1}{\lambda'} \begin{bmatrix} \phi'_{13} \\ \phi'_{23} \\ \phi'_{33} \end{bmatrix}.$$

### 6.12.7 Homogeneous Camera Equation




$$\begin{aligned} \hat{\Omega}, \hat{\tau} &= \underset{\Omega,\tau}{\operatorname{argmax}} \sum_{i=1}^{I} \log\left[Pr(\mathbf{x}_i|\mathbf{w}_i, \mathbf{\Lambda}, \mathbf{\Omega}, \boldsymbol{\tau})\right] \\ &= \underset{\Omega,\tau}{\operatorname{argmax}} \sum_{i=1}^{I} \log\left[\mathrm{Norm}_{\mathbf{x}_i}[\mathbf{pinhole}[\mathbf{w}_i, \mathbf{\Lambda}, \mathbf{\Omega}, \boldsymbol{\tau}], \sigma^2\mathbf{I}]\right] \end{aligned}$$

Figure 6.15: Exterior Orientation (Homography).

**Remark.** *The relationship between a 3D world point* $\mathbf{w}$ *and its 2D image projection* $\mathbf{x}$ *is described by the camera equation:*

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}.$$

**Definition 6.12.3** (Linearization of the Camera Equation)**.** By pre-multiplying the above equation with the inverse of the intrinsic matrix $\Lambda$, we can express the relationship in terms of normalized image coordinates:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}.$$

The scale factor $\lambda$ can be expressed using the third row of the matrix equation:

$$\lambda = \omega_{31}u + \omega_{32}v + \omega_{33}w + \tau_z.$$

Substituting this back into the first two equations yields a linear system for $\mathbf{\Omega}$ and $\tau$.

### 6.12.8 Solving for Camera Pose

**Remark.** *The resulting linear system is expressed as:*

$$\mathbf{Ab} = 0,$$

*where* $\mathbf{b}$ *contains the elements of* $\mathbf{\Omega}$ *and* $\tau$*. To solve this, the problem is formulated as a minimum direction problem:*

$$\min \|\mathbf{Ab}\|^2 \quad \textit{subject to} \quad \|\mathbf{b}\| = 1.$$

*Using Singular Value Decomposition (SVD):*

$$\mathbf{A} = \mathbf{U\Sigma V}^T,$$

*the solution* $\mathbf{b}$ *is set to the last column of* $\mathbf{V}$.

**Definition 6.12.4** (Refining the Rotation Matrix)**.** To ensure that $\mathbf{\Omega}$ is a valid rotation matrix (orthogonal with determinant 1), we refine it using:

$$\mathbf{\Omega} = \mathbf{UV}^T,$$

where $\mathbf{U}$ and $\mathbf{V}$ come from the SVD of the initial estimate of $\mathbf{\Omega}$.

**Remark.** *The translation vector* $\tau$ *is determined by incorporating the scaling factor* $\lambda'$*:*

$$\lambda' = \frac{1}{6} \sum_{m=1}^{3} \sum_{n=1}^{3} \frac{\phi'_{mn}}{\omega_{mn}},$$

*and:*

$$\tau = \frac{1}{\lambda'} \begin{bmatrix} \phi'_{13} \\ \phi'_{23} \\ \phi'_{33} \end{bmatrix}.$$

*These provide initial estimates for subsequent non-linear optimization.*

## 6.13 Transformation Models for Calibration

### 6.13.1 Overview of Calibration

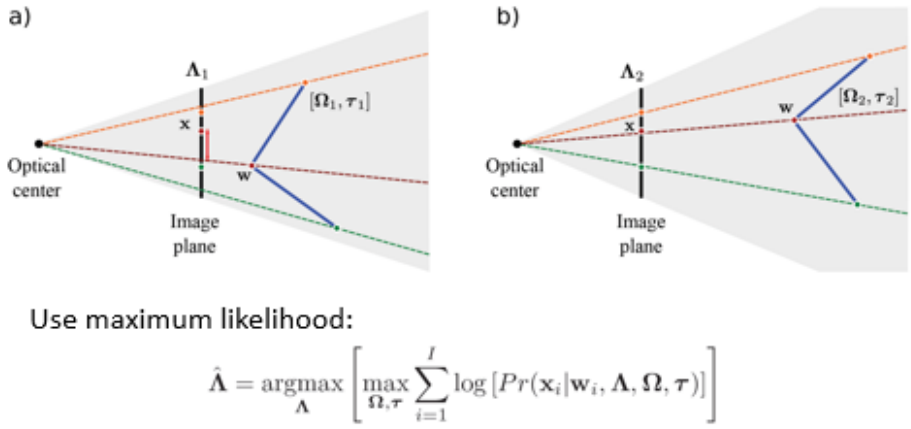



Figure 6.16: Calibration - Learning intrinsic parameters.

**Definition 6.13.1.** Calibration involves estimating the intrinsic parameters $\Lambda$ of the camera, such as focal lengths, principal point, and skew. The maximum likelihood estimation for calibration is given by:

$$\hat{\Lambda} = \arg\max_{\Lambda} \left[ \max_{\Omega,\tau} \sum_{i=1}^{I} \log \Pr(\mathbf{x}_i \mid \mathbf{w}_i, \Lambda, \Omega, \tau) \right].$$

The intrinsic parameters $\Lambda$ include:

$$\Lambda = \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix},$$

where $\phi_x, \phi_y$ are focal lengths, $\gamma$ is skew, and $(\delta_x, \delta_y)$ are the principal point coordinates.

### 6.13.2 Optimization Strategy

**Remark.** *Calibration typically requires alternating optimization:*

1. *Fix the intrinsic parameters $\Lambda$ and solve for the extrinsic parameters $\Omega$ (rotation) and $\tau$ (translation):*

$$\hat{\Omega}, \hat{\tau} = \arg\max_{\Omega,\tau} \sum_{i=1}^{I} \log \Pr(\mathbf{x}_i \mid \mathbf{w}_i, \Lambda, \Omega, \tau).$$

   *The extrinsic parameters are determined using the projection equations:*

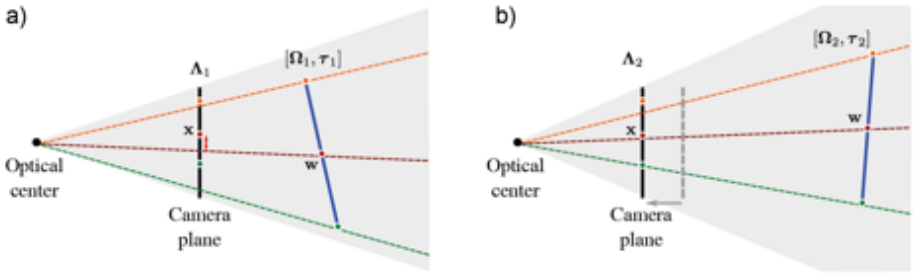
$$\mathbf{x}_i = \Lambda \left[\Omega \mathbf{w}_i + \tau\right].$$

2. *Fix the extrinsic parameters and optimize the intrinsic parameters:*

$$\hat{\Lambda} = \arg\max_{\Lambda} \sum_{i=1}^{I} \log \Pr(\mathbf{x}_i \mid \mathbf{w}_i, \Lambda, \Omega, \tau).$$

   *Substitute the extrinsic parameters back into the projection equation to isolate $\Lambda$.*

### 6.13.3 Homogeneous Camera Equation




$$\hat{\mathbf{\Lambda}} = \underset{\mathbf{\Lambda}}{\operatorname{argmax}} \left[ \max_{\mathbf{\Omega}_{1\ldots J}, \boldsymbol{\tau}_{1\ldots J}} \sum_{i=1}^{I} \sum_{j=1}^{J} \log \left[ Pr(\mathbf{x}_{ij} | \mathbf{w}_i, \mathbf{\Lambda}, \mathbf{\Omega}_j, \boldsymbol{\tau}_j) \right] \right]$$

Figure 6.17: Calibration (Homography).

**Remark.** *The camera projection model in homogeneous coordinates is:*

$$\lambda \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \Lambda \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix},$$

*where $(u, v, w)$ are world coordinates, and $(x_i, y_i)$ are image coordinates.*

### 6.13.4 Maximum Likelihood Estimation

**Remark.** *The intrinsic parameters $\Lambda$ are estimated using maximum likelihood:*

$$\hat{\Lambda} = \arg\max_{\Lambda} \sum_{i=1}^{I} \log \left[ \textit{pinhole}(\mathbf{w}_i, \Lambda, \Omega, \tau), \sigma_x^2 \right],$$

*where pinhole$(\cdot)$ is the projection function. In practice, this reduces to minimizing the reprojection error:*

$$\hat{\Lambda} = \arg\min_{\Lambda} \sum_{i=1}^{I} \|\mathbf{x}_i - \textit{pinhole}(\mathbf{w}_i, \Lambda, \Omega, \tau)\|^2.$$

### 6.13.5 Linear Least Squares for Intrinsic Parameters

**Theorem 6.13.2** (Linear Solution for Intrinsic Parameters)**.** *The pinhole projection model is linear with respect to the intrinsic parameters $\Lambda$. Using:*

$$\mathbf{x}_i = \textit{pinhole}(\mathbf{w}_i, \Lambda, \Omega, \tau),$$

*we write:*

$$A_i \mathbf{h} = \mathbf{x}_i,$$

*where $\mathbf{h} = [\phi_x, \phi_y, \gamma, \delta_x, \delta_y]^\top$, and $A_i$ is a matrix derived from the projection equation. The least-squares solution is:*

$$\hat{\mathbf{h}} = \arg\min_{\mathbf{h}} \sum_{i=1}^{I} \|A_i \mathbf{h} - \mathbf{x}_i\|^2.$$

*The solution is obtained via Singular Value Decomposition (SVD) of $A$:*

$$A = U \Sigma V^\top,$$

*and $\hat{\mathbf{h}}$ is given by the last column of $V$ corresponding to the smallest singular value.*

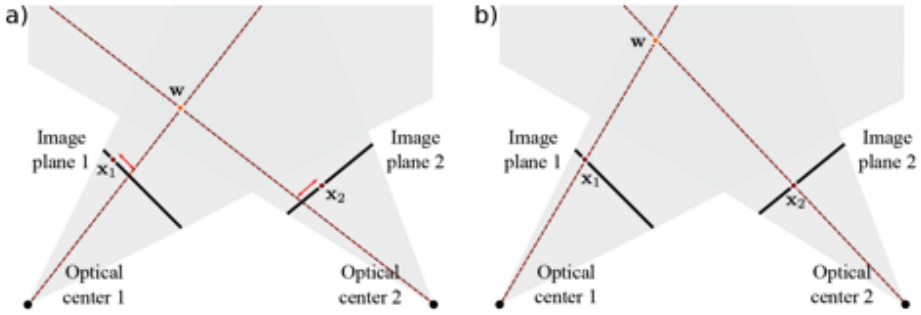
### 6.13.6 Coupling of Intrinsic and Extrinsic Parameters

**Remark.** *Once $\Lambda$ is calibrated, it is used to decouple extrinsic parameters $\Omega$ and $\tau$. This decoupling simplifies tasks such as:*

- ***3D Reconstruction:*** *Recovering 3D coordinates from 2D projections.*
- ***Exterior Orientation:*** *Estimating camera position and orientation relative to the scene.*

## 6.14 Transformation Model with 3D Reconstruction

### 6.14.1 Problem Definition




Use maximum likelihood:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \left( \sum_{j=1}^{J} \log[Pr(\mathbf{x}_j|\mathbf{w}, \mathbf{\Lambda}_j, \mathbf{\Omega}_j, \boldsymbol{\tau}_j)] \right)$$

Figure 6.18: Triangulation/Reconstruction - Inferring 3D points.

**Remark.** *Reconstruction involves inferring 3D world points* $\mathbf{w}$ *from corresponding image points* $\mathbf{x}_j$ *observed by multiple cameras. This can be achieved using a maximum likelihood approach:*

$$\hat{\mathbf{w}} =_{\mathbf{w}} \sum_{j=1}^{J} \log \Pr(\mathbf{x}_j \mid \mathbf{w}, \Lambda_j, \Omega_j, \tau_j),$$

*where* $J$ *denotes the number of cameras.*

### 6.14.2 Mathematical Framework

**Remark.** *The camera equations are expressed in homogeneous coordinates for the* $j$*th camera:*

$$\lambda_j \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{x_j} & \gamma_j & \delta_{x_j} & 0 \\ 0 & \phi_{y_j} & \delta_{y_j} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \omega_{11_j} & \omega_{12_j} & \omega_{13_j} & \tau_{x_j} \\ \omega_{21_j} & \omega_{22_j} & \omega_{23_j} & \tau_{y_j} \\ \omega_{31_j} & \omega_{32_j} & \omega_{33_j} & \tau_{z_j} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}.$$

### 6.14.3 Steps in Reconstruction

- ***Step 1: Pre-multiply with the inverse of the intrinsic matrix.***

$$\lambda_j \begin{bmatrix} x'_j \\ y'_j \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11_j} & \omega_{12_j} & \omega_{13_j} & \tau_{x_j} \\ \omega_{21_j} & \omega_{22_j} & \omega_{23_j} & \tau_{y_j} \\ \omega_{31_j} & \omega_{32_j} & \omega_{33_j} & \tau_{z_j} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}. \tag{6.1}$$

- ***Step 2: Solve for*** $\lambda_j$***.*** *Using the third equation:*

$$\lambda_j = \omega_{31_j} u + \omega_{32_j} v + \omega_{33_j} w + \tau_{z_j}.$$

*Substitute back into the first two equations:*

$$\lambda_j x'_j = \omega_{11_j} u + \omega_{12_j} v + \omega_{13_j} w + \tau_{x_j},$$
$$\lambda_j y'_j = \omega_{21_j} u + \omega_{22_j} v + \omega_{23_j} w + \tau_{y_j}.$$

- ***Step 3: Formulate a linear system.*** *Rearrange the equations into a linear form:*

$$\begin{bmatrix} \omega_{11_j} - x'_j\omega_{31_j} & \omega_{12_j} - x'_j\omega_{32_j} & \omega_{13_j} - x'_j\omega_{33_j} \\ \omega_{21_j} - y'_j\omega_{31_j} & \omega_{22_j} - y'_j\omega_{32_j} & \omega_{23_j} - y'_j\omega_{33_j} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} x'_j\tau_{z_j} - \tau_{x_j} \\ y'_j\tau_{z_j} - \tau_{y_j} \end{bmatrix}.$$

### 6.14.4 Transformation Between Plane and Image

**Remark.** *The transformation between the plane and the image is given by:*

$$\mathbf{T} = \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix}.$$

*Points in the reference frames are transformed as:*

$$\mathbf{w} = \mathbf{T}^{-1}\mathbf{x}, \quad \mathbf{w}' = \mathbf{\Omega}\mathbf{w} + \tau.$$

## 6.15 Properties of the Homography

### 6.15.1 Transformations Between Images

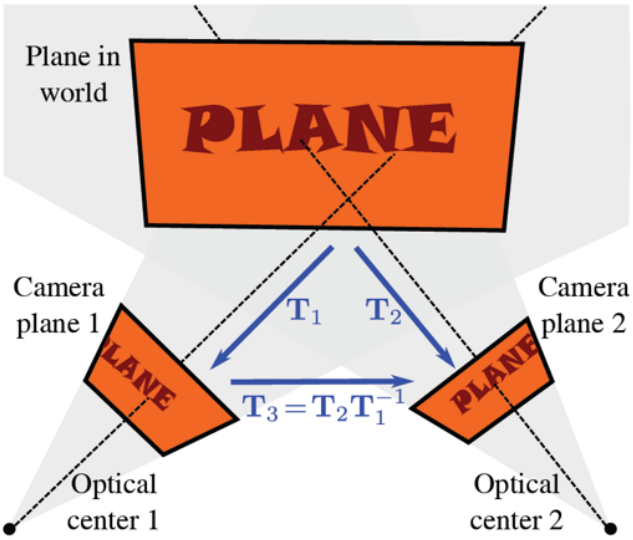



Figure 6.19:

**Remark.** *Homographies establish the relationship between transformations across multiple views:*

- *When two cameras view the same plane, each camera forms a homography with the plane:*

$$\mathbf{T}_1 = \Lambda_1\mathbf{R}_1, \quad \mathbf{T}_2 = \Lambda_2\mathbf{R}_2.$$

- *The transformation between the images formed by the two cameras is also a homography:*

$$\mathbf{T}_3 = \mathbf{T}_2\mathbf{T}_1^{-1}.$$

- *This indicates that images of the same planar surface from different viewpoints are related through a homography.*

*The diagram depicts two cameras viewing the same plane. Each camera generates a homography with the plane, and the transformation between the images is described by the composite homography* $\mathbf{T}_3$.

### 6.15.2 Linear Transformation of a Ray

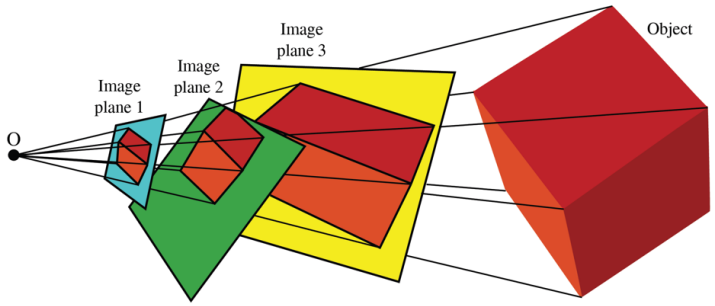



Figure 6.20:

**Remark.** *Homography can be interpreted as a linear transformation of a ray:*

$$\lambda \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}.$$

- *Images formed by all planes cutting the same ray bundle are related by homographies.*

- *This property enables mapping points from one image plane to another via the homography matrix.*

*The diagram illustrates how multiple planes intersect the same ray bundle, with their respective images being linked by homographies.*

### 6.15.3 Special Case: Camera Under Pure Rotation

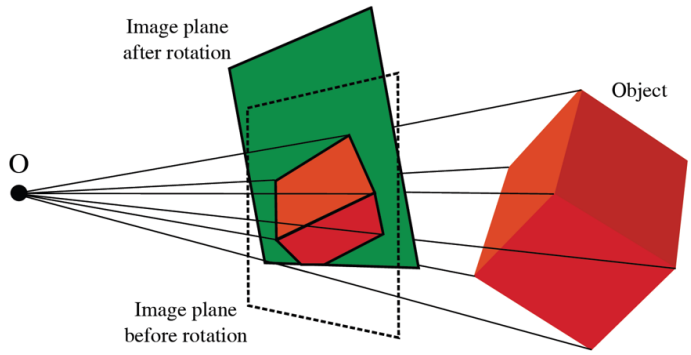



Figure 6.21:

**Remark.** *In the case of pure camera rotation:*

- *The image plane before and after rotation forms a homography:*

$$\mathbf{\Phi} = \Lambda \mathbf{\Omega}_2 \Lambda^{-1}.$$

- *Here, $\mathbf{\Omega}_2$ represents the rotation matrix, and $\Lambda$ corresponds to the intrinsic matrix of the camera.*

*The diagram represents a scenario where the camera rotates about its optical center, and the image planes before and after rotation are connected through the homography $\mathbf{\Phi}$.*

## 6.16 Robust Estimation of Transformations

### 6.16.1 RANSAC

**Remark.** *Robust estimation methods are critical for handling outliers in data, as standard least-squares estimation can be heavily influenced by such points. For instance, even a few outliers can lead to significant deviations in the estimated model. The RANSAC (Random Sampling and Consensus) algorithm offers a robust approach to estimating transformation models by iteratively refining the parameters.*

*The RANSAC algorithm operates as follows:*

1. ***Random Sampling:*** *Select a minimal subset of data points (e.g., two points for line fitting) to estimate the transformation model.*

2. ***Model Estimation:*** *Use the sampled subset to compute the model parameters.*

3. ***Inlier Evaluation:*** *Compute the number of inliers, i.e., points that fit the model within a specified tolerance.*
4. ***Iteration:*** *Repeat the process for a fixed number of iterations or until a stopping criterion is met.*
5. ***Refinement:*** *After selecting the model with the highest number of inliers, re-estimate the transformation parameters using all inliers.*

**Theorem 6.16.1** (RANSAC Transformation Estimation)**.** *Let $\mathcal{D}$ represent the set of data points, and $\mathcal{S} \subset \mathcal{D}$ denote a randomly sampled subset. The RANSAC objective is:*

$$\mathbf{T}^* = \arg\max_{\mathbf{T}} \sum_{i=1}^{|\mathcal{D}|} \mathbb{I}\left[\|\mathbf{x}_i - \mathbf{T}\mathbf{x}_i'\| \leq \epsilon\right],$$

*where $\mathbb{I}[\cdot]$ is the indicator function, $\epsilon$ is the inlier threshold, and $\mathbf{T}$ is the transformation model.*

### 6.16.2 Sequential RANSAC

**Remark.** *Sequential RANSAC applies the RANSAC algorithm iteratively to identify multiple transformation models in piecewise planar scenes. In each iteration:*

1. *A subset of data is assigned to a plane by identifying inliers using RANSAC.*
2. *The inliers are removed, and the process is repeated to find additional planes.*

*This approach can fail when the algorithm greedily identifies planes without considering spatial coherence.*

### 6.16.3 PEaRL: Propose, Estimate, and Re-Learn

**Remark.** *The PEaRL algorithm improves upon Sequential RANSAC by enforcing spatial coherence. This is achieved by associating a label $l$ with each point, indicating the plane to which it belongs. The model is defined as:*

$$\Pr(\mathbf{x}_i \mid \mathbf{y}_i, l_i = k) = \mathit{Norm}_{\mathbf{x}_i}\left[\mathit{hom}(\mathbf{y}_i, \Phi_k), \sigma^2\mathbf{I}\right],$$

*where $\mathit{hom}(\mathbf{y}_i, \Phi_k)$ represents the homography transformation for plane $k$.*

*To encourage spatial coherence, a Markov Random Field prior on the labels is used:*

$$\Pr(\mathbf{l}) \propto \exp\left(-\sum_{i,j\in\mathcal{N}} w_{ij}\delta[l_i \neq l_j]\right),$$

*where $\mathcal{N}$ denotes neighboring points and $w_{ij}$ is a weight indicating the likelihood of neighboring points belonging to the same plane.*

**Definition 6.16.2** (PEaRL Solution)**.** The solution involves:

- Proposing initial labels for each point using RANSAC.
- Refining the labels using the Markov Random Field prior with an optimization algorithm such as alpha expansion.
- Re-estimating the homography transformations for each plane iteratively until convergence.

# Chapter 7

# Temporal Models and Multiple Camera models

## 7.1 Multiple Cameras: Structure from Motion

### 7.1.1 Motivation

**Definition 7.1.1.** Structure from motion (SfM) aims to reconstruct the 3D structure of a scene and determine the camera parameters by analyzing a set of images. It involves:

- Estimating the **intrinsic matrix** (camera parameters like focal length, principal point).
- Estimating the **extrinsic matrix** (rotation and translation of the camera).
- Recovering the 3D positions of points projected into the images.

Given: Projections of $I$ 3D points into $J$ images.

## 7.2 Two View Geometry

### 7.2.1 Epipolar Lines and Epipoles.

**Definition 7.2.1.** The relationship between two images of the same scene is described using the geometry of the epipolar constraint:

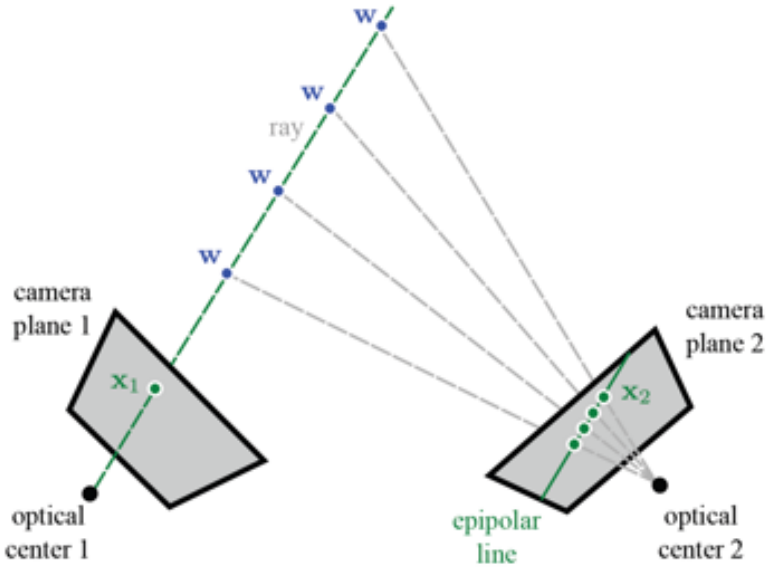



Figure 7.1: Epipolar Lines.

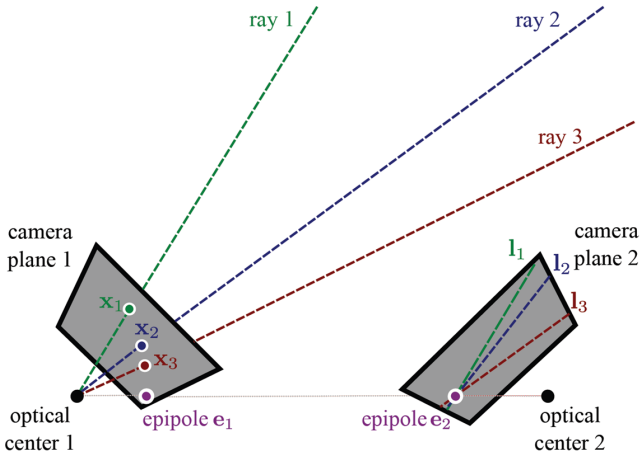



Figure 7.2: Epipoles.

- **Epipolar Lines**: For a point in one image, the corresponding point in the other image lies on a line, known as the epipolar line. This is because the 3D point projects to the second image as a ray.
- **Epipoles**: The points where the line joining the two camera centers intersects each image

plane. All epipolar lines pass through the epipoles.

**Remark.** *The relationship between two views is encoded by:*

- ***Essential Matrix* ($\mathbf{E}$)**: *Describes the relative rotation and translation between two cameras in normalized image coordinates. It is defined as:*

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R},$$

*where $[\mathbf{t}]_\times$ is the skew-symmetric matrix of translation and $\mathbf{R}$ is the rotation matrix.*

- ***Fundamental Matrix* ($\mathbf{F}$)**: *Generalizes the essential matrix to unnormalized image coordinates by incorporating the intrinsic camera parameters:*

$$\mathbf{F} = \mathbf{K}_2^{-\top} \mathbf{E} \mathbf{K}_1^{-1},$$

*where $\mathbf{K}_1$ and $\mathbf{K}_2$ are the intrinsic matrices of the two cameras.*

**Remark.** *The epipolar constraint is expressed as:*

$$\mathbf{x}_2^\top \mathbf{F} \mathbf{x}_1 = 0,$$

*where:*

- $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *are the homogeneous coordinates of corresponding points in the two images.*
- $\mathbf{F}$ *ensures that the corresponding point* $\mathbf{x}_2$ *lies on the epipolar line defined by* $\mathbf{x}_1$ *in the second image.*

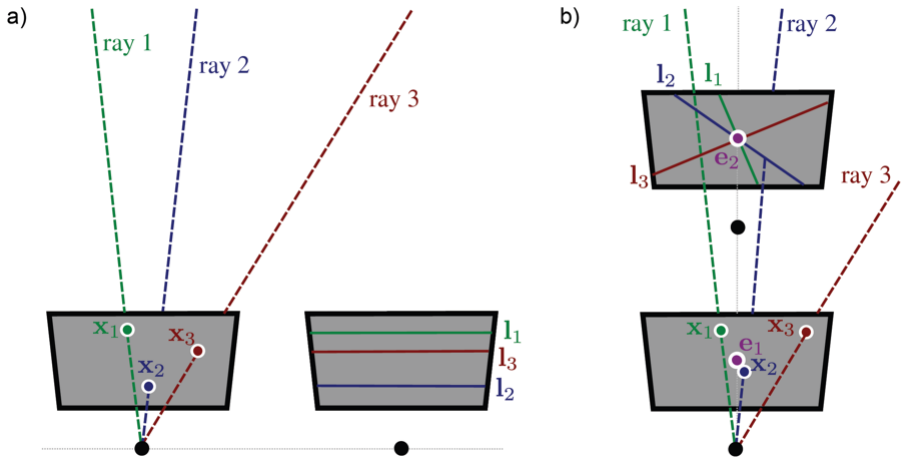



Figure 7.3: Special Configurations.

### 7.2.2 Special Configurations

**Remark.** *In certain configurations:*

- *If the camera centers are co-planar, the epipolar lines become parallel, and the epipoles may be at infinity.*
- *If the camera undergoes pure rotation, the epipoles coincide with the camera center.*

### 7.2.3 Significance of Epipolar Geometry

**Remark.** *Epipolar geometry simplifies correspondence search:*

- *Instead of searching in 2D, a corresponding point needs to be located along the epipolar line in the second image.*
- *This reduces the search space significantly and is key to stereo vision and 3D reconstruction.*

## 7.3 The Essential and Fundamental Matrices

### 7.3.1 Definition and Relationship

*The relationship between two views of the same scene is encoded by the **Essential Matrix** and the **Fundamental Matrix**:*

- ***Essential Matrix (**$\mathbf{E}$**)**: Captures the relative rotation $\mathbf{\Omega}$ and translation $\tau$ between two normalized cameras.*
- ***Fundamental Matrix (**$\mathbf{F}$**)**: Generalizes the essential matrix to unnormalized image coordinates. It relates pixel coordinates between two views and satisfies:*

$$\mathbf{x}_2^\top \mathbf{F} \mathbf{x}_1 = 0.$$

### 7.3.2 The Essential Matrix

*The essential matrix is defined as:*

$$\mathbf{E} = [\tau]_\times \mathbf{\Omega},$$

*where:*

- $[\tau]_\times$ *is the skew-symmetric matrix of the translation vector* $\tau$*:*

$$[\tau]_\times = \begin{bmatrix} 0 & -\tau_z & \tau_y \\ \tau_z & 0 & -\tau_x \\ -\tau_y & \tau_x & 0 \end{bmatrix}.$$

- $\mathbf{\Omega}$ *is the rotation matrix.*

**Essential Matrix Constraint:** *The essential matrix satisfies the bilinear constraint for corresponding points in two normalized images:*

$$\widetilde{\mathbf{x}}_2^\top \mathbf{E} \widetilde{\mathbf{x}}_1 = 0,$$

*where* $\widetilde{\mathbf{x}}_1, \widetilde{\mathbf{x}}_2$ *are homogeneous coordinates of the corresponding points.*

**Derivation:**

1. *For the first camera at the origin, the 3D point* $\mathbf{w}$ *projects as:*

$$\lambda_1 \widetilde{\mathbf{x}}_1 = \mathbf{w}.$$

2. *For the second camera, the projection is:*

$$\lambda_2 \widetilde{\mathbf{x}}_2 = \mathbf{\Omega}\mathbf{w} + \tau.$$

   *Substituting* $\mathbf{w}$ *from the first equation:*

$$\lambda_2 \widetilde{\mathbf{x}}_2 = \lambda_1 \mathbf{\Omega} \widetilde{\mathbf{x}}_1 + \tau.$$

3. *Taking the cross product with* $\tau$*:*

$$\lambda_2 \tau \times \widetilde{\mathbf{x}}_2 = \lambda_1 \tau \times (\mathbf{\Omega} \widetilde{\mathbf{x}}_1).$$

4. *Taking the inner product with* $\widetilde{\mathbf{x}}_2$*:*

$$\widetilde{\mathbf{x}}_2^\top (\tau \times (\mathbf{\Omega} \widetilde{\mathbf{x}}_1)) = 0.$$

   *Using the definition of the essential matrix:*

$$\mathbf{E} = \tau_\times \mathbf{\Omega},$$

   *we obtain:*

$$\widetilde{\mathbf{x}}_2^\top \mathbf{E} \widetilde{\mathbf{x}}_1 = 0.$$

**Properties of the Essential Matrix:**

- ***Rank Constraint:*** $\mathit{rank}(\mathbf{E}) = 2$.
- ***Determinant Constraint:*** $\mathit{det}(\mathbf{E}) = 0$.
- ***Non-Linear Constraints:*** *The elements of* $\mathbf{E}$ *satisfy:*

$$2\mathbf{E}\mathbf{E}^\top \mathbf{E} - \mathit{trace}(\mathbf{E}\mathbf{E}^\top)\mathbf{E} = 0.$$

### 7.3.3 Recovering Epipolar Lines and Epipoles

**Epipolar Lines:** *The epipolar line in the second image corresponding to a point $\widetilde{\mathbf{x}}_1$ in the first image is:*

$$\mathbf{l}_2 = \mathbf{E}\widetilde{\mathbf{x}}_1,$$

*and the epipolar line in the first image corresponding to $\widetilde{\mathbf{x}}_2$ is:*

$$\mathbf{l}_1 = \mathbf{E}^\top \widetilde{\mathbf{x}}_2.$$

**Epipoles:** *The epipoles $\mathbf{e}_1$ and $\mathbf{e}_2$ are the null spaces of $\mathbf{E}$ and $\mathbf{E}^\top$, respectively:*

$$\mathbf{E}\mathbf{e}_1 = 0, \quad \mathbf{E}^\top \mathbf{e}_2 = 0.$$

### 7.3.4 Decomposing the Essential Matrix

*To recover $\tau$ and $\mathbf{\Omega}$, compute the SVD:*

$$\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top,$$

*where:*

$$\tau_\times = \mathbf{U}\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\mathbf{U}^\top, \quad \mathbf{\Omega} = \mathbf{U}\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\mathbf{V}^\top.$$

### 7.3.5 Fundamental Matrix

*The fundamental matrix relates pixel coordinates in unnormalized images. It satisfies:*

$$\mathbf{x}_2^\top \mathbf{F}\mathbf{x}_1 = 0,$$

*where $\mathbf{x}_1$ and $\mathbf{x}_2$ are the homogeneous coordinates of corresponding points in two views. It is related to $\mathbf{E}$ as:*

$$\mathbf{F} = \mathbf{K}_2^{-\top}\mathbf{E}\mathbf{K}_1^{-1},$$

*where $\mathbf{K}_1$ and $\mathbf{K}_2$ are the intrinsic camera matrices.*

### 7.3.6 The Fundamental Matrix

**Definition 7.3.1.** The fundamental matrix $\mathbf{F}$ generalizes the essential matrix to unnormalized image coordinates. It encodes the relationship between corresponding points in two images taken by uncalibrated cameras. The bilinear constraint is given by:

$$\mathbf{x}_2^\top \mathbf{F}\mathbf{x}_1 = 0,$$

where $\mathbf{F} = \mathbf{\Lambda}_2^{-\top}\mathbf{E}\mathbf{\Lambda}_1^{-1}$, with $\mathbf{\Lambda}_1$ and $\mathbf{\Lambda}_2$ being the intrinsic calibration matrices of the two cameras.

### 7.3.7 Estimation of the Fundamental Matrix

**Remark.** *When the fundamental matrix is correctly estimated, the epipolar line induced by a point in one image should pass through the corresponding point in the other image. This suggests the following minimization criterion:*

$$\hat{\mathbf{F}} = \arg\min_{\mathbf{F}} \sum_{i=1}^{I} \left( dist[\mathbf{x}_1^i, \ell_1^i]^2 + dist[\mathbf{x}_2^i, \ell_2^i]^2 \right),$$

*where* $\ell_1^i = \mathbf{F}^\top \mathbf{x}_2^i$ *and* $\ell_2^i = \mathbf{F}\mathbf{x}_1^i$.

### 7.3.8 The Eight-Point Algorithm

- *Solve for the fundamental matrix* $\mathbf{F}$ *using homogeneous coordinates.*
- *Stack constraints from at least 8 pairs of points into a linear system* $\mathbf{A}\mathbf{f} = 0$*, where* $\mathbf{f}$ *contains the elements of* $\mathbf{F}$.
- *Solve the minimum direction problem using Singular Value Decomposition (SVD):*

  $$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top,$$

  *and set* $\mathbf{f}$ *to the last column of* $\mathbf{V}$.

### 7.3.9 Fitting Concerns

**Remark.** *The eight-point algorithm has several limitations:*

- *The solution does not ensure that* $\mathbf{F}$ *is of rank 2. To address this, set the smallest singular value of* $\mathbf{F}$ *to zero.*
- *It is sensitive to numerical issues related to data scaling. Pre-normalize the data to improve robustness.*
- *The points must be in general positions (not all planar) and must provide sufficient translation between views.*
- *Use the result to initialize non-linear optimization of the true criterion while ensuring non-linear constraints are satisfied.*
- *For robustness, consider the seven-point algorithm, especially in conjunction with RANSAC.*

### 7.3.10 Properties of the Fundamental Matrix

- *The fundamental matrix* $\mathbf{F}$ *is of rank 2.*
- *The epipolar constraint* $\mathbf{x}_2^\top \mathbf{F} \mathbf{x}_1 = 0$ *holds for all corresponding points.*
- *The epipoles* $\mathbf{e}_1$ *and* $\mathbf{e}_2$ *are the null spaces of* $\mathbf{F}$ *and* $\mathbf{F}^\top$*, respectively.*

## 7.4 Reconstruction Pipeline

*The two-view reconstruction pipeline is a systematic method for extracting 3D information from two images taken from slightly different viewpoints. It combines geometric estimation with mathematical optimization to recover camera motion and reconstruct 3D points.*

### 7.4.1 Pipeline Steps

*The reconstruction process can be described as follows:*

*1.* ***Feature Detection and Matching:*** *Let* $\mathbf{x}_i$ *and* $\mathbf{x}'_i$ *represent points in the two images. Features are detected and matched, yielding correspondences* $\{(\mathbf{x}_i, \mathbf{x}'_i)\}_{i=1}^N$.

*2.* ***Estimation of the Fundamental Matrix:*** *The fundamental matrix* $\mathbf{F}$ *is estimated by solving the relation:*

$$\mathbf{x}_i'^\top \mathbf{F} \mathbf{x}_i = 0,$$

*using robust algorithms such as RANSAC to handle outliers. This establishes the geometric relationship between the views.*

*3.* ***Extraction of the Essential Matrix:*** *For calibrated cameras, the essential matrix* $\mathbf{E}$ *is derived from* $\mathbf{F}$*:*

$$\mathbf{E} = \mathbf{\Lambda}'^\top \mathbf{F} \mathbf{\Lambda},$$

*where* $\mathbf{\Lambda}$ *and* $\mathbf{\Lambda}'$ *are the intrinsic parameter matrices of the two cameras.*

*4.* ***Camera Motion Recovery:*** *The essential matrix* $\mathbf{E}$ *is decomposed to recover the relative rotation* $\mathbf{\Omega}$ *and translation* $\tau$ *of the cameras:*

$$\mathbf{E} = [\tau]_\times \mathbf{\Omega},$$

*where* $[\tau]_\times$ *is the skew-symmetric matrix representation of* $\tau$*.*

*5.* ***Triangulation for 3D Reconstruction:*** *Given* $\mathbf{\Omega}$ *and* $\tau$*, the 3D position* $\mathbf{w}_i$ *of a point is reconstructed via triangulation. Using the projection matrices* $\mathbf{P}$ *and* $\mathbf{P}'$*, the following equations are solved:*

$$\lambda \mathbf{x}_i = \mathbf{P} \mathbf{w}_i, \quad \lambda' \mathbf{x}'_i = \mathbf{P}' \mathbf{w}_i,$$

*where* $\mathbf{P} = \mathbf{\Lambda}[\mathbf{I} \,|\, \mathbf{0}]$ *and* $\mathbf{P}' = \mathbf{\Lambda}'[\mathbf{\Omega} \,|\, \tau]$*.*

6. ***Non-linear Optimization:*** *To minimize reprojection error, the parameters $\{\mathbf{w}_i, \mathbf{\Omega}, \tau\}$ are refined by solving:*

$$\min_{\mathbf{w}_i, \mathbf{\Omega}, \tau} \sum_{i=1}^{N} \|\mathbf{x}_i - \pi(\mathbf{P}\mathbf{w}_i)\|^2 + \left\|\mathbf{x}'_i - \pi(\mathbf{P}'\mathbf{w}_i)\right\|^2,$$

*where $\pi$ denotes the projection operation.*

### 7.4.2 Dense Reconstruction and Rectification

*To compute a dense depth map, the disparity $d$ for each pixel is estimated by warping the images to align epipolar lines (rectification). The disparity map is refined using techniques like dynamic programming and graph cuts, ensuring accurate per-pixel depth estimates.*

### 7.4.3 Mathematical Formulation

*The overall optimization for reconstruction is expressed as:*

$$\hat{\mathbf{w}}, \hat{\mathbf{\Omega}}, \hat{\tau} = _{\mathbf{w}, \mathbf{\Omega}, \tau} \sum_{i=1}^{N} \log \Pr(\mathbf{x}_i, \mathbf{x}'_i \mid \mathbf{w}_i, \mathbf{\Lambda}, \mathbf{\Omega}, \tau),$$

*where the probability model incorporates the camera parameters, the 3D points, and the observation noise.*

## 7.5 Rectification

*Rectification is the process of transforming images so that corresponding points in two views lie on the same horizontal lines, aligning the epipolar lines to simplify correspondence search for stereo matching. This is crucial in scenarios such as dense reconstruction, where matching points across images requires consistent geometry.*

### 7.5.1 Epipolar Geometry and Pure Translation

*When the camera motion is a pure translation along the $u$-axis, the epipolar lines in the rectified image are horizontal and parallel. This simplifies the search for corresponding points, as they are constrained along the same row in both images.*

### 7.5.2 Planar Rectification Process

*To achieve rectification, planar homographies $\Phi_1$ and $\Phi_2$ are applied to images 1 and 2, respectively. The homography $\Phi_2$ is decomposed into three transformations:*

$$\Phi_2 = T_3 T_2 T_1,$$

*where:*

- $T_1$*: Translates the image origin to the center of the image:*

$$T_1 = \begin{bmatrix} 1 & 0 & -\delta_x \\ 0 & 1 & -\delta_y \\ 0 & 0 & 1 \end{bmatrix}.$$

- $T_2$*: Rotates the epipole to align with the horizontal axis:*

$$T_2 = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

- $T_3$*: Projects the epipole to infinity along the horizontal direction:*

$$T_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/e_x & 0 & 1 \end{bmatrix},$$

  *where $e_x$ is the $x$-coordinate of the epipole.*

### 7.5.3 Family of Homographies for Rectification

*A family of homographies can be applied to image 1 to achieve rectification. These are parameterized as:*

$$\Phi_1[\alpha] = (I + e_2 \alpha^\top)\Phi_2 M,$$

*where $\Phi_2 M$ represents the desired rectified transformation. The parameter $\alpha$ is chosen to minimize the reprojection error in a least squares sense:*

$$\alpha = \arg\min_{\alpha} \sum_{i=1}^{I} \|\mathit{hom}(\mathbf{x}_i, \Phi_1[\alpha]) - \mathit{hom}(\mathbf{x}_i, \Phi_2)\|^2.$$

### 7.5.4 Before and After Rectification

*Prior to rectification, the epipolar lines converge and are not aligned. After rectification, the epipolar lines become horizontal and parallel, ensuring that corresponding points in the two images are aligned along the same rows.*

*Rectification is a crucial preprocessing step in stereo vision tasks, enabling efficient and accurate matching of points across views by enforcing geometric constraints on epipolar lines.*

## 7.6 Multi-view Reconstruction

*Multi-view reconstruction involves estimating the 3D structure of a scene by using multiple views taken from different perspectives. This section covers the key steps and mathematical formulations.*

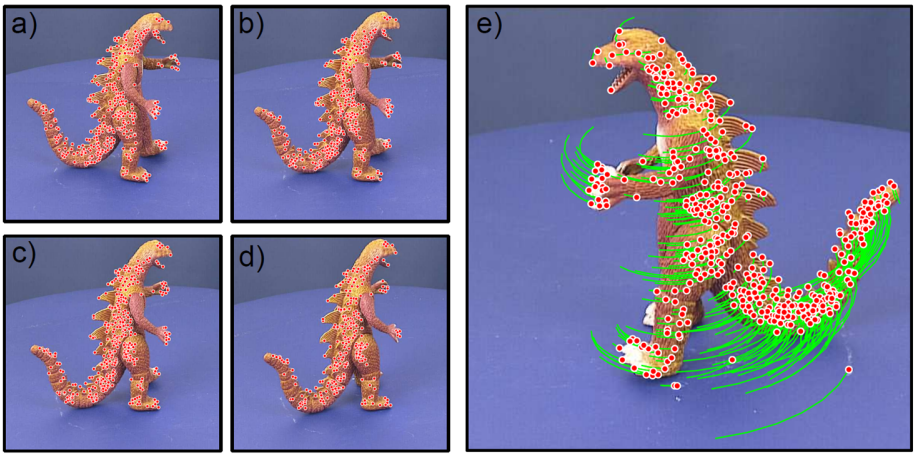



Figure 7.4: Multi-view Reconstruction Example.

### 7.6.1 Reconstruction from Video

*Given a sequence of images captured from the same camera, multi-view reconstruction can leverage temporal continuity. The steps are as follows:*

1. *Images are taken from the same calibrated camera, enabling intrinsic parameter optimization (auto-calibration).*
2. *Matching points across frames is facilitated by temporal tracking.*
3. *Not every 3D point is visible in all images due to occlusions or field-of-view limitations.*

4. *Constraints on matching can include the three-view equivalent of the fundamental matrix, termed the tri-focal tensor.*

5. *Simultaneous initialization of camera parameters is achieved through factorization methods.*

### 7.6.2 Bundle Adjustment

*Bundle adjustment is a refinement process to optimize the estimates of both structure (3D points) and motion (camera parameters) using non-linear optimization. The objective function is:*

$$\hat{\theta} =_{\theta} \sum_{i=1}^{I} \sum_{j=1}^{J} \log \Pr(\mathbf{x}_{ij} \mid \mathbf{w}_i, \Lambda, \Omega_j, \tau_j),$$

*which can be equivalently expressed as:*

$$\hat{\theta} =_{\theta} \sum_{i=1}^{I} \sum_{j=1}^{J} \|\mathbf{x}_{ij} - \textit{pinhole}[\mathbf{w}_i, \Lambda, \Omega_j, \tau_j]\|^2.$$

*This is a least squares problem:*

$$\hat{\theta} =_{\theta} \mathbf{z}^\top \mathbf{z},$$

*where:*

$$\mathbf{z} = \begin{bmatrix} \mathbf{x}_{11} - \textit{pinhole}[\mathbf{w}_1, \Lambda, \Omega_1, \tau_1] \\ \mathbf{x}_{12} - \textit{pinhole}[\mathbf{w}_1, \Lambda, \Omega_2, \tau_2] \\ \vdots \\ \mathbf{x}_{IJ} - \textit{pinhole}[\mathbf{w}_I, \Lambda, \Omega_J, \tau_J] \end{bmatrix}.$$

*Optimization of the least squares objective is typically performed using the Gauss-Newton method:*

$$\theta^{[t]} = \theta^{[t-1]} + \lambda \left(\mathbf{J}^\top \mathbf{J}\right)^{-1} \frac{\partial f}{\partial \theta},$$

*where:*

$$J_{mn} = \frac{\partial z_m}{\partial \theta_n},$$

*and* $\mathbf{J}^\top \mathbf{J}$ *is structured as:*

$$\mathbf{J}^\top \mathbf{J} = \begin{bmatrix} \mathbf{J}_{\mathbf{w}}^\top \mathbf{J}_{\mathbf{w}} & \mathbf{J}_{\mathbf{w}}^\top \mathbf{J}_{\Omega} & \mathbf{J}_{\mathbf{w}}^\top \mathbf{J}_{\tau} \\ \mathbf{J}_{\Omega}^\top \mathbf{J}_{\mathbf{w}} & \mathbf{J}_{\Omega}^\top \mathbf{J}_{\Omega} & \mathbf{J}_{\Omega}^\top \mathbf{J}_{\tau} \\ \mathbf{J}_{\tau}^\top \mathbf{J}_{\mathbf{w}} & \mathbf{J}_{\tau}^\top \mathbf{J}_{\Omega} & \mathbf{J}_{\tau}^\top \mathbf{J}_{\tau} \end{bmatrix}.$$

*Efficient inversion of this structure is critical for practical performance.*

### 7.6.3 Applications and Considerations

- *Multi-view reconstruction is widely used in applications such as 3D modeling, augmented reality, and robotics.*
- *Using video sequences allows for temporal continuity, making matching more robust.*
- *Factorization and bundle adjustment are key techniques for solving the optimization problem.*

### 7.6.4 Applications

- ***Multi-View Reconstruction:*** *Extends two-view methods to multiple images for better coverage and accuracy.*
- ***3D Model Creation:*** *Used in photogrammetry and augmented reality.*
- ***Image Rectification:*** *Essential for stereo vision and dense depth map computation.*

## 7.7 Temporal Models

### 7.7.1 Motivation

**Definition 7.7.1.** Temporal models aim to track the state of an object over time, given a sequence of observations. The goal is to compute the marginal posterior distribution over the hidden state vector $\mathbf{w}_t$ at time $t$:

$$P(\mathbf{w}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t),$$

where $\mathbf{x}_t$ represents the measurements at time $t$.

**Remark.** *Examples in temporal models:*

- *2D Position of tracked object in image*
- *3D Pose of tracked obeject in world*
- *Joint positions of tracked articulated model*

*Challenges in temporal modeling include:*

- *Clutter and data association issues.*
- *Incomplete information in a single frame to fully define the state.*
- *Complex relationships between states across frames. (e.g. joint angles)*

### 7.7.2 Temporal Model

**State Estimation**

**Definition 7.7.2.** State estimation relies on:

- A **measurement model**: $P(\mathbf{x}_t|\mathbf{w}_t)$, which relates a set of measurements $x_t$ that provide information about the state $w_t$ at time $t$. This is a generative model.
- A **temporal model**: $P(\mathbf{w}_t|\mathbf{w}_{t-1})$, which describes how the state evolves over time. This is a time series model.

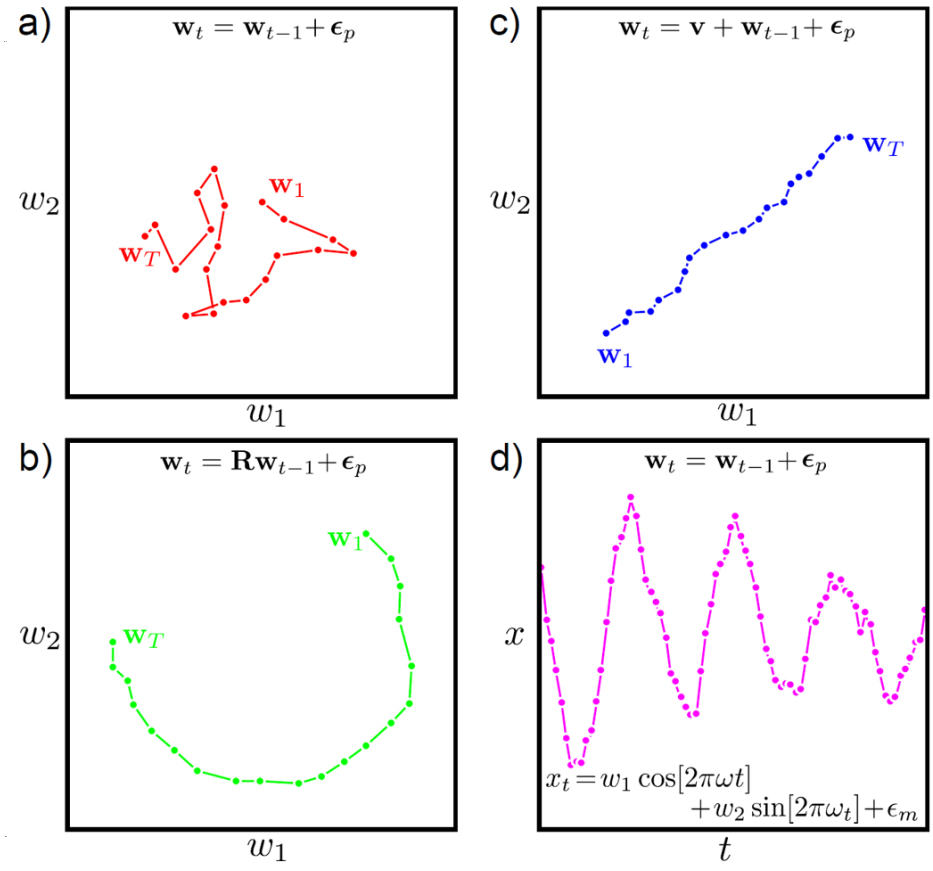



Figure 7.5: Temporal Model Illustration. **a** and **c** cover most cases for Kalman Filter. $a$ is behaving like a Random Walk / Brownian Motion. $b$ is linear temporal model with $R$ being the rotation matrix. $c$ is the constant velocity model as the $v$ is unchanged.

**Remark.** *The model assumes:*

- ***Markov property**: The state at time $t$ depends only on the state at $t-1$. It's conditionally independent of states at times 1, ..., $t-2$ given $t-1$.*
- ***Conditional independence of measurements**: Measurements $x_i$ at time $t$ depend only on the state at time $t$.*

**Recursive Estimation**

**Definition 7.7.3.** Recursive estimation alternates between:

1. **Temporal Evolution (Temporal Model)**:

$$P_r(w_t|\mathbf{x}_{1:t-1}) = \int P_r(w_t|w_{t-1})P_r(w_{t-1}|\mathbf{x}_{1:t-1})\, dw_{t-1}$$

   - **Prior at time** $t$: $P_r(w_t|\mathbf{x}_{1:t-1})$. Each time the prior is based on the Chapman-Kolmogorov equation.
   - **Temporal model**: $P_r(w_t|w_{t-1})$
   - **Posterior at time** $t-1$: $P_r(w_{t-1}|\mathbf{x}_{1:t-1})$

2. **Measurement Update (Measurement Model)**:

$$P(\mathbf{w}_t|\mathbf{x}_1, \ldots, \mathbf{x}_t) \propto P(\mathbf{x}_t|\mathbf{w}_t)P(\mathbf{w}_t|\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}).$$

### 7.7.3 Kalman Filter

**Theorem 7.7.4** (Kalman Filter)**.** *The Kalman filter is a special case of recursive estimation procedure. It assumes linear temporal models. If the posterior at time $t-1$ is Gaussian, the prior and posterior at time $t$ will also remain Gaussian:*

- ***State Evolution (Time Evolution Equation):***

$$\mathbf{w}_t = \mu_p + \mathbf{\Psi}\mathbf{w}_{t-1} + \epsilon_p, \quad \epsilon_p \sim \mathcal{N}(0, \Sigma_p),$$

  *where:*

  - *$\mu_p$ is the mean of the process noise,*
  - *$\mathbf{\Psi}$ is the state transition matrix,*
  - *$\Sigma_p$ is the covariance of the process noise.*
  - *$P(w_t|w_{t-1})$ also follows Gaussian.*

- ***Measurement Model:***

$$\mathbf{x}_t = \mu_m + \mathbf{\Phi}\mathbf{w}_t + \epsilon_m, \quad \epsilon_m \sim \mathcal{N}(0, \Sigma_m),$$

*where:*

  - $\mu_m$ *is the mean of the measurement noise,*
  - $\mathbf{\Phi}$ *relates the state to the measurement,*
  - $\Sigma_m$ *is the covariance of the measurement noise.*
  - $P(x_t|w_t)$ *also follows Gaussian.*

*Kalman filter equations are rules for updating the means and covariances of these Gaussians. In Kalman filter, the goal is to estimate* $w_t$ *(the true but hidden state that evolves over time) as accurately as possible based on the observed* $x_t$ *over time.*

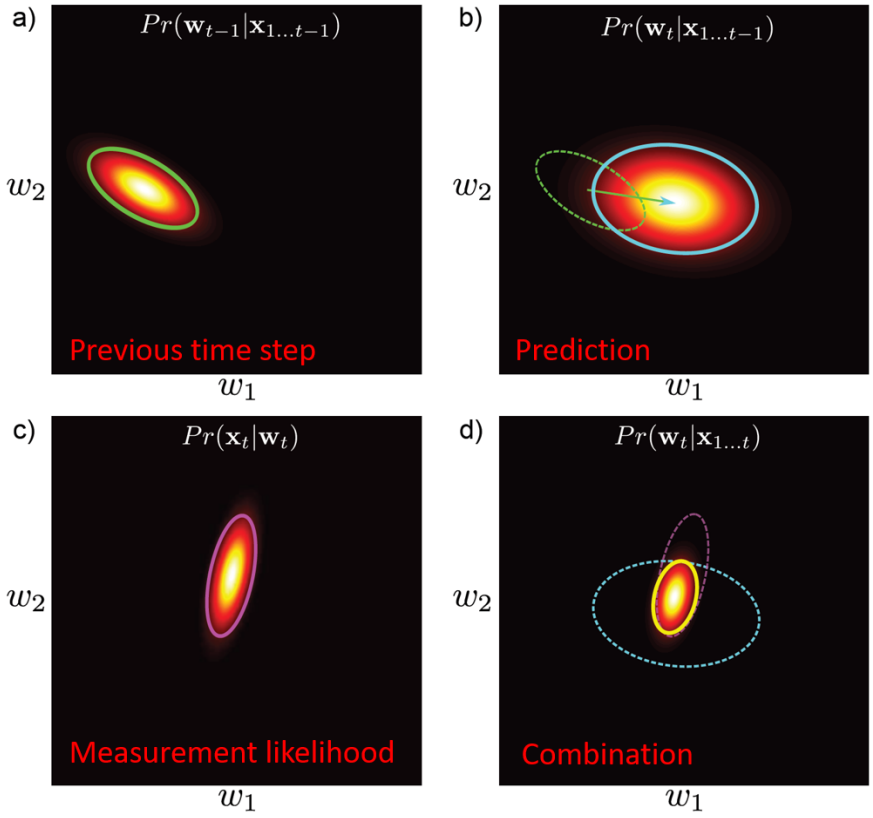



Figure 7.6: The Kalman Filter.

**Definition 7.7.5** (Temporal Evolution (Prediction))**.** The temporal evolution of the state in the Kalman filter can be expressed as:

$$P(\mathbf{w}_t \mid \mathbf{x}_{1:t-1}) = \int P(\mathbf{w}_t \mid \mathbf{w}_{t-1}) P(\mathbf{w}_{t-1} \mid \mathbf{x}_{1:t-1}) d\mathbf{w}_{t-1}.$$

Using the state transition equation and assuming Gaussian priors, this becomes:

$$P(\mathbf{w}_t \mid \mathbf{x}_{1:t-1}) = \mathcal{N}(\mathbf{w}_t \mid \mu_+, \Sigma_+),$$

where:

$$\mu_+ = \mathbf{\Psi}\mu_{t-1}, \quad \Sigma_+ = \mathbf{\Psi}\Sigma_{t-1}\mathbf{\Psi}^T + \Sigma_p.$$

**Definition 7.7.6** (Measurement Incorporation (Update))**.** The measurement incorporation step updates the belief of the state based on the new measurement:

$$P(\mathbf{w}_t \mid \mathbf{x}_{1:t}) = \frac{P(\mathbf{x}_t \mid \mathbf{w}_t) P(\mathbf{w}_t \mid \mathbf{x}_{1:t-1})}{P(\mathbf{x}_t \mid \mathbf{x}_{1:t-1})}.$$

Assuming Gaussian noise in the measurements, this simplifies to:

$$P(\mathbf{w}_t \mid \mathbf{x}_{1:t}) = \mathcal{N}(\mathbf{w}_t \mid \mu_t, \Sigma_t),$$

where:

$$\mu_t = \Sigma_t \left( \mathbf{\Phi}^T \Sigma_m^{-1} (\mathbf{x}_t - \mu_m) + \Sigma_+^{-1} \mu_+ \right),$$

$$\Sigma_t = \left( \mathbf{\Phi}^T \Sigma_m^{-1} \mathbf{\Phi} + \Sigma_+^{-1} \right)^{-1}.$$

**Theorem 7.7.7** (Kalman Gain)**.** *The Kalman gain matrix* $\mathbf{K}$ *is defined to simplify calculations and improve numerical stability:*

$$\mathbf{K} = \Sigma_+ \mathbf{\Phi}^T (\Sigma_m + \mathbf{\Phi}\Sigma_+ \mathbf{\Phi}^T)^{-1}.$$

**Mean Term**

*Using matrix inversion relations, the mean of the posterior distribution can be expressed as:*

$$\mu_t = \mu_+ + \mathbf{K}(\mathbf{x}_t - \mathbf{\Phi}\mu_+).$$

*This is derived from the relation:*

$$\mu_t = \mathbf{K}(\mathbf{x}_t - \mu_m) + (\mathbf{I} - \mathbf{K}\mathbf{\Phi})\mu_+.$$

**Covariance Term**

*Using similar matrix inversion relations, the covariance of the posterior distribution is given by:*

$$\Sigma_t = \Sigma_+ - \mathbf{K}\mathbf{\Phi}\Sigma_+.$$

*This can also be expressed as:*

$$\Sigma_t = (\mathbf{I} - \mathbf{K}\mathbf{\Phi})\Sigma_+.$$

- $\mu_t$*: Updated mean of the state.*
- $\Sigma_t$*: Updated covariance of the state.*
- $\mathbf{K}$*: Kalman gain, determining how much the measurement affects the state update.*

**Kalman Filter Summary**

- ***Time Evolution Equation:***

  $$P(\mathbf{w}_t \mid \mathbf{w}_{t-1}) = \mathcal{N}(\mathbf{w}_t \mid \mu_p + \mathbf{\Psi}\mathbf{w}_{t-1}, \Sigma_p).$$

  *This equation describes how the hidden state evolves from $t-1$to $t$, incorporating noise.*

- ***Measurement Equation:***

  $$P(\mathbf{x}_t \mid \mathbf{w}_t) = \mathcal{N}(\mathbf{x}_t \mid \mu_m + \mathbf{\Phi}\mathbf{w}_t, \Sigma_m).$$

  *This equation models how the hidden state is translated into noisy measurements.*

- ***Inference:***
  - ***State Prediction:***

    $$\mu_+ = \mu_p + \mathbf{\Psi}\mu_{t-1}.$$

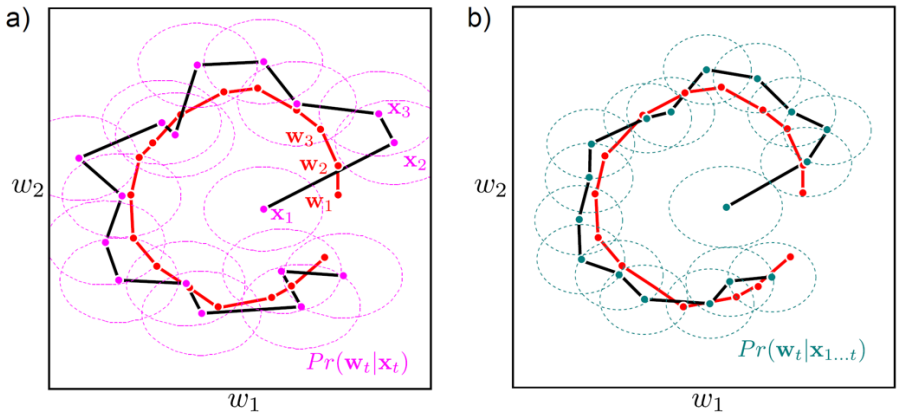
    *This step predicts the hidden state's mean based on its previous estimate and the state evolution model.*
  - ***Covariance Prediction:***

    $$\Sigma_+ = \Sigma_p + \mathbf{\Psi}\Sigma_{t-1}\mathbf{\Psi}^T.$$

    *This step predicts the uncertainty in the hidden state after evolution.*
  - ***State Update:***

    $$\mu_t = \mu_+ + \mathbf{K}(\mathbf{x}_t - \mu_m - \mathbf{\Phi}\mu_+).$$

    *This step updates the state estimate by incorporating new measurements, weighting them based on their uncertainty.*




$$Pr(\mathbf{w}_t|\mathbf{w}_{t-1}) = \text{Norm}_{\mathbf{w}_t}[\mathbf{w}_{t-1}, \sigma_p^2\mathbf{I}]$$
$$Pr(\mathbf{x}_t|\mathbf{w}_t) = \text{Norm}_{\mathbf{x}_t}[\mathbf{w}_t, \boldsymbol{\Sigma}_m]$$

Figure 7.7: Kalman Filter Example 1. Black dots represent true hidden states $w_t$ over time. Red dots represent the measurements $x_t$ at corresponding time steps.

- ***Covariance Update:***

$$\Sigma_t = (\mathbf{I} - \mathbf{K}\boldsymbol{\Phi})\Sigma_+.$$

*This step updates the uncertainty in the hidden state based on the new measurement.*

- ***Kalman Gain:***

$$\mathbf{K} = \Sigma_+\boldsymbol{\Phi}^T(\Sigma_m + \boldsymbol{\Phi}\Sigma_+\boldsymbol{\Phi}^T)^{-1}.$$

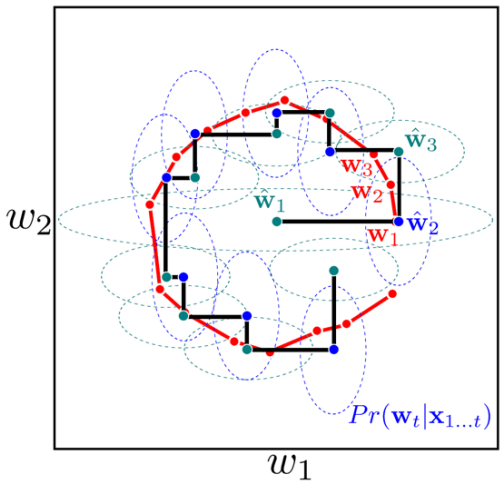
*The Kalman gain balances the trust between the prediction (state evolution) and the new observation (measurement). Larger measurement noise $\Sigma_m$ leads to less reliance on $x_t$.*

### 7.7.4 Kalman Smoothing

*Kalman smoothing is used when we want to estimate the state of a system not only based on past and present measurements but also by incorporating future measurements. Unlike standard Kalman filtering, which is causal and works sequentially, smoothing uses both forward and backward passes to refine state estimates.*

#### Types of Kalman Smoothing

- ***Fixed Lag Smoother:*** *This is an online scheme where the optimal estimate for the state at time $t - \tau$ is calculated using measurements up to time $t$. Here, $\tau$ is the time lag. The goal is to*




Alternates:

$$Pr(x_t|\mathbf{w}_t) = \text{Norm}_{x_t}\left[\begin{bmatrix}1 & 0\end{bmatrix}\mathbf{w}_t, \sigma_m^2\right]$$
$$Pr(x_t|\mathbf{w}_t) = \text{Norm}_{x_t}\left[\begin{bmatrix}0 & 1\end{bmatrix}\mathbf{w}_t, \sigma_m^2\right]$$

Figure 7.8: Kalman Filter Example 2

*compute:*

$$P(\mathbf{w}_{t-\tau} \mid \mathbf{x}_1, \ldots, \mathbf{x}_t).$$

- ***Fixed Interval Smoother:*** *In this scheme, we have a fixed interval of measurements and estimate the optimal state based on all the measurements in that interval. The objective is to calculate:*

$$P(\mathbf{w}_t \mid \mathbf{x}_1, \ldots, \mathbf{x}_T),$$

*where $T$ is the total length of the interval.*

### Fixed-Lag Kalman Smoothing

- ***Visualization:***

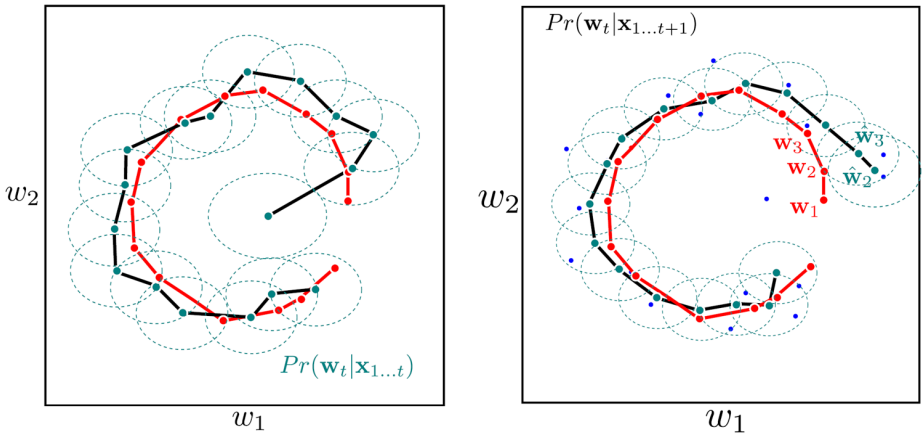



Figure 7.9: Fixed-lag Kalman Smoothing.

*The left panel of the figure illustrates standard Kalman filtering, where the posterior distribution $P(\mathbf{w}_t \mid \mathbf{x}_{1:t})$ is updated sequentially. The right panel shows fixed-lag smoothing, where a lag of $\tau = 1$ is used. The posterior $P(\mathbf{w}_t \mid \mathbf{x}_{1:t+1})$ incorporates measurements one step ahead ($\mathbf{x}_{t+1}$).*

**Fixed Interval Smoothing**

*The backward pass in fixed interval smoothing refines the state estimates using the following set of recursive equations:*

$$\mu_{t|T} = \mu_{t|t} + C_t(\mu_{t+1|T} - \mu_{t+1|t}),$$
$$\Sigma_{t|T} = \Sigma_{t|t} + C_t(\Sigma_{t+1|T} - \Sigma_{t+1|t}),$$

*where:*

$$C_t = \Sigma_{t|t}\boldsymbol{\Psi}^T(\Sigma_{t+1|t})^{-1}.$$

**Explanation:** *- $\mu_{t|T}$: Smoothed mean estimate at time $t$ based on all measurements. - $\Sigma_{t|T}$: Smoothed covariance estimate at time $t$. - $\mu_{t+1|T}$ and $\Sigma_{t+1|T}$: Smoothed estimates at the next time step $t+1$. - $\mu_{t+1|t}$ and $\Sigma_{t+1|t}$: Filtered (forward-pass) estimates at time $t+1$. - $C_t$: The smoothing gain, which determines the contribution of the next time step to the current state estimate.*

**Equivalence:** *This process is mathematically equivalent to belief propagation or the forward-backward algorithm used in probabilistic graphical models.*

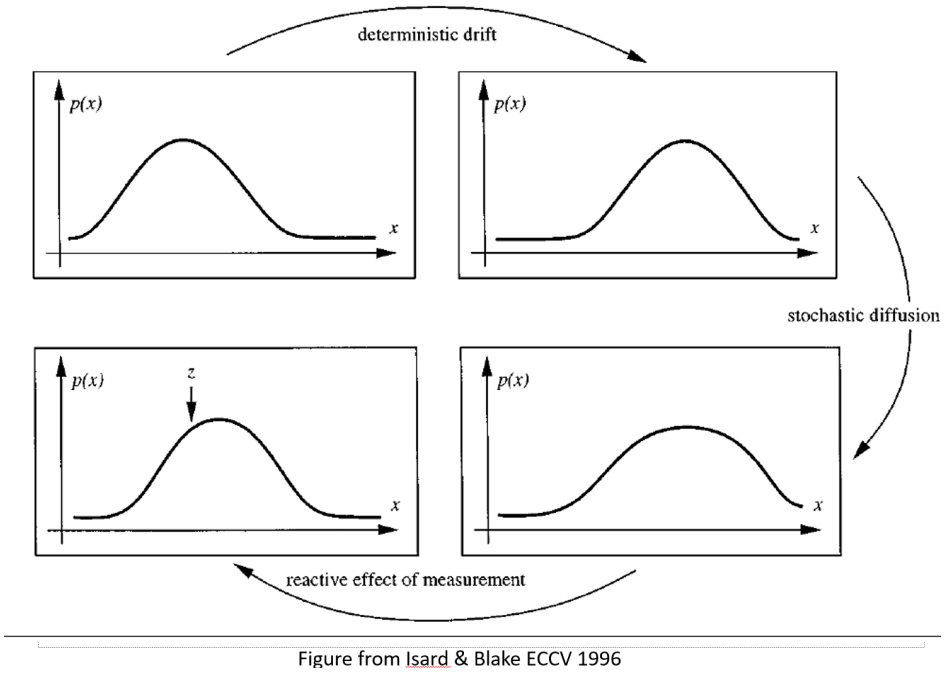



Figure 7.10: Fixed Interval Smoothing.

### 7.7.5 Extended Kalman Filter

*The Extended Kalman Filter (EKF) extends the standard Kalman filter to handle **non-linear measurement** and **temporal equations**. It approximates non-linear models by linearizing them around the current estimate using a first-order Taylor expansion.*

### Problems with the Standard Kalman Filter

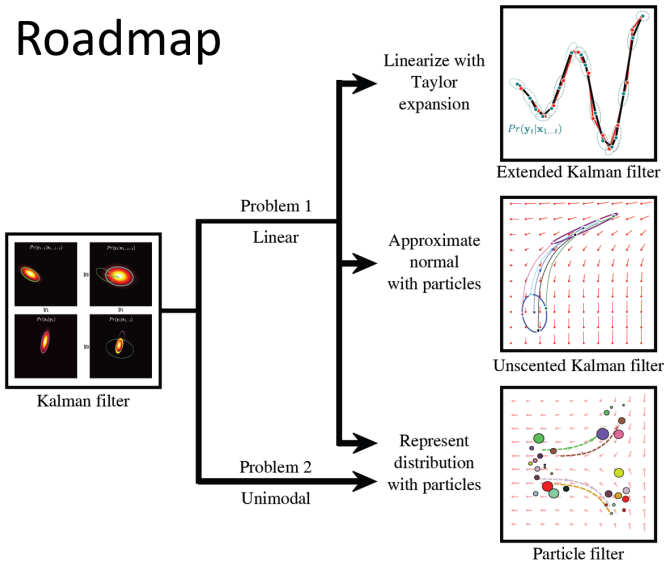



Figure 7.11: Problems with Kalman Filter.

- *The Kalman filter requires* ***linear temporal and measurement equations****, which limits its application to linear systems.*
- *It assumes the posterior distribution is Gaussian, making it unsuitable for genuinely multi-modal distributions.*

### Non-Linear Models in EKF

*The EKF allows non-linear state evolution and measurement equations:*

$$\mathbf{w}_t = \mathbf{f}(\mathbf{w}_{t-1}, \epsilon_p),$$

$$\mathbf{x}_t = \mathbf{g}(\mathbf{w}_t, \epsilon_m),$$

*where:*

- $\mathbf{f}(\cdot)$*: Non-linear state transition function.*
- $\mathbf{g}(\cdot)$*: Non-linear measurement function.*
- $\epsilon_p$*: Process noise.*

- $\epsilon_m$: *Measurement noise.*

*The key idea is to approximate* $\mathbf{f}$ *and* $\mathbf{g}$ *as locally linear by using the first-order Taylor expansion.*

**Jacobians for Linearization**

*The Jacobian matrices of partial derivatives are used to approximate non-linear functions:*

$$\mathbf{\Psi} = \left.\frac{\partial \mathbf{f}(\mathbf{w}_{t-1}, \epsilon_p)}{\partial \mathbf{w}_{t-1}}\right|_{\mu_{t-1},0}, \quad \mathbf{\Upsilon}_p = \left.\frac{\partial \mathbf{f}(\mathbf{w}_{t-1}, \epsilon_p)}{\partial \epsilon_p}\right|_{\mu_{t-1},0},$$

$$\mathbf{\Phi} = \left.\frac{\partial \mathbf{g}(\mathbf{w}_t, \epsilon_m)}{\partial \mathbf{w}_t}\right|_{\mu_+,0}, \quad \mathbf{\Upsilon}_m = \left.\frac{\partial \mathbf{g}(\mathbf{w}_t, \epsilon_m)}{\partial \epsilon_m}\right|_{\mu_+,0}.$$

**EKF Equations**

*The EKF equations, incorporating the Jacobians, are as follows:*

**Prediction Step:**

$$\mu_+ = \mathbf{f}(\mu_{t-1}, 0),$$

$$\Sigma_+ = \mathbf{\Psi}\Sigma_{t-1}\mathbf{\Psi}^T + \mathbf{\Upsilon}_p\Sigma_p\mathbf{\Upsilon}_p^T.$$

**Update Step:**

$$\mu_t = \mu_+ + \mathbf{K}(\mathbf{x}_t - \mathbf{g}(\mu_+, 0)),$$

$$\Sigma_t = (\mathbf{I} - \mathbf{K}\mathbf{\Phi})\Sigma_+,$$

*where the Kalman gain is:*

$$\mathbf{K} = \Sigma_+\mathbf{\Phi}^T\left(\mathbf{\Upsilon}_m\Sigma_m\mathbf{\Upsilon}_m^T + \mathbf{\Phi}\Sigma_+\mathbf{\Phi}^T\right)^{-1}.$$

### Illustration of EKF

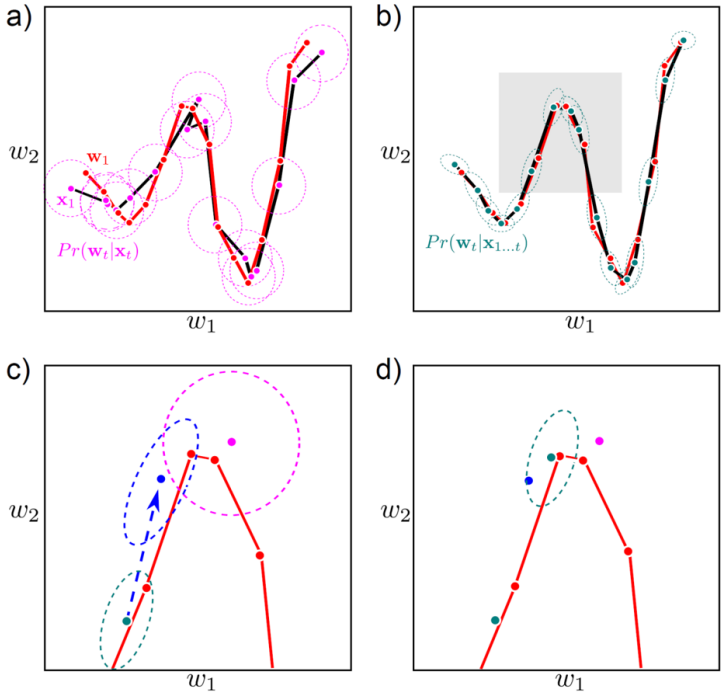



Figure 7.12: Extended Kalman Filter Illustration.

- *In the example shown, the state evolution is non-linear:*

$$\mathbf{f}(\mathbf{w}, \epsilon_p) = \begin{bmatrix} w_1 \\ w_1 \sin(w_1) \end{bmatrix} + \epsilon_p,$$

*with the Jacobian:*

$$\mathbf{\Psi} = \begin{bmatrix} 1 & 0 \\ \cos(w_1) + w_1 \cos(w_1) & 0 \end{bmatrix}.$$

- *The EKF approximates the non-linear posterior by linearizing the state and measurement models. The magenta and teal curves in the plots represent the linearized predictions and measurements.*

**Problems with EKF**

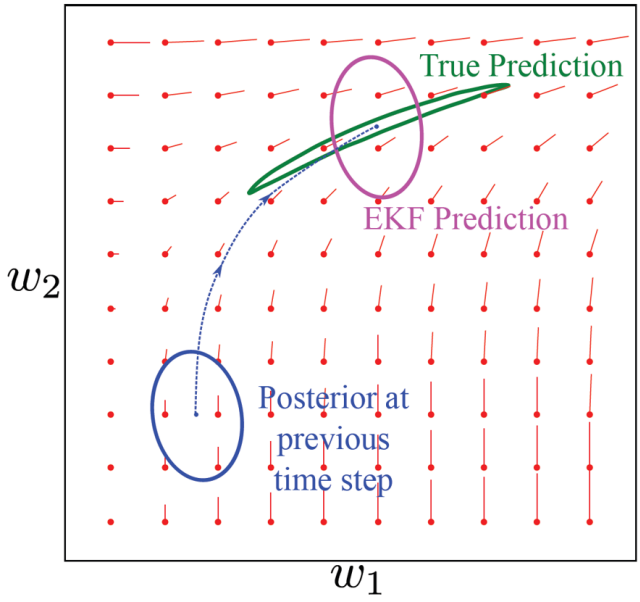



Figure 7.13: EKF Issues in Visual.

- *EKF predictions can deviate from the true prediction because linearization introduces errors, particularly for highly non-linear models.*
- *The figure illustrates how the EKF's posterior may fail to align perfectly with the true posterior due to this approximation.*

### 7.7.6 Unscented Kalman Filter

**Definition 7.7.8.** The Unscented Kalman Filter (UKF) approximates distributions using a sum of weighted particles. It handles non-linear systems by propagating particles through non-linear dynamics and observation models.

The state evolution and observation models are defined as:

$$\mathbf{w}_t = f(\mathbf{w}_{t-1}) + \boldsymbol{\epsilon}_p, \quad \mathbf{x}_t = g(\mathbf{w}_t) + \boldsymbol{\epsilon}_m,$$

where $f(\cdot)$ is the dynamic model, $g(\cdot)$ is the observation model, and $\boldsymbol{\epsilon}_p, \boldsymbol{\epsilon}_m$ are noise terms.

### Particle Approximation

**Theorem 7.7.9.** *The prior distribution $P(\mathbf{w}_{t-1} \mid \mathbf{x}_{1:t-1})$ can be approximated as:*

$$P(\mathbf{w}_{t-1} \mid \mathbf{x}_{1:t-1}) \approx \sum_{j=0}^{2D_w} a_j \delta(\mathbf{w}_{t-1} - \hat{\mathbf{w}}^{[j]}),$$

*where:*

$$\mu_{t-1} = \sum_{j=0}^{2D_w} a_j \hat{\mathbf{w}}^{[j]}, \quad \Sigma_{t-1} = \sum_{j=0}^{2D_w} a_j (\hat{\mathbf{w}}^{[j]} - \mu_{t-1})(\hat{\mathbf{w}}^{[j]} - \mu_{t-1})^\top.$$

### Sigma Point Scheme

- *The particles $\hat{\mathbf{w}}^{[j]}$ are chosen as:*

$$\hat{\mathbf{w}}^{[0]} = \mu_{t-1}, \quad \hat{\mathbf{w}}^{[j]} = \mu_{t-1} + \sqrt{\frac{D_w}{1-a_0}}\mathbf{e}_j, \quad \hat{\mathbf{w}}^{[D_w+j]} = \mu_{t-1} - \sqrt{\frac{D_w}{1-a_0}}\mathbf{e}_j.$$

- *The weights are given by:*

$$a_j = \frac{1-a_0}{2D_w}.$$

### Reconstitution of Mean and Covariance

**Theorem 7.7.10.** *After propagating particles, the mean and covariance are computed as:*

$$\mu_+ = \sum_{j=0}^{2D_w} a_j \hat{\mathbf{w}}_+^{[j]}, \quad \Sigma_+ = \sum_{j=0}^{2D_w} a_j (\hat{\mathbf{w}}_+^{[j]} - \mu_+)(\hat{\mathbf{w}}_+^{[j]} - \mu_+)^\top + \Sigma_p.$$

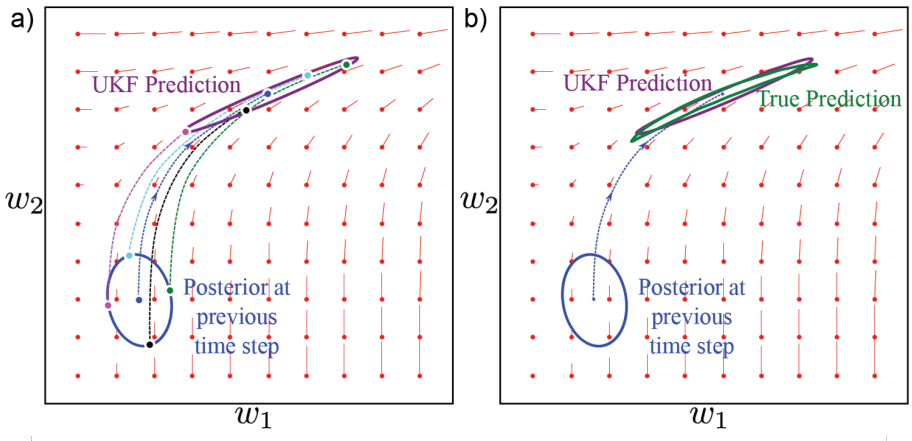



Figure 7.14: Unscented Kalman Filter Illustration.

### Measurement Update

**Remark.** *The predicted distribution is updated using the measurement equation:*

$$P(\mathbf{w}_t \mid \mathbf{x}_{1:t}) \approx \sum_{j=0}^{2D_w} a_j \delta(\mathbf{w}_t - \hat{\mathbf{w}}^{[j]}).$$

**Theorem 7.7.11.** *The updated mean and covariance are:*

$$\mu_t = \mu_+ + \mathbf{K}(\mathbf{x}_t - \mu_x), \quad \Sigma_t = \Sigma_+ - \mathbf{K}\Sigma_x\mathbf{K}^\top,$$

*where:*

$$\mathbf{K} = \left( \sum_{j=0}^{2D_w} a_j (\hat{\mathbf{w}}^{[j]} - \mu_+)(\hat{\mathbf{x}}^{[j]} - \mu_x)^\top \right) \Sigma_x^{-1}.$$

### Optimization Techniques

**Remark.** *The UKF optimization is performed using least squares methods, such as the Gauss-Newton method:*

$$\boldsymbol{\theta}^{[t]} = \boldsymbol{\theta}^{[t-1]} + \lambda(\mathbf{J}^\top\mathbf{J})^{-1}\frac{\partial f}{\partial \boldsymbol{\theta}},$$

*where* $\mathbf{J}$ *is the Jacobian matrix of the error terms.*

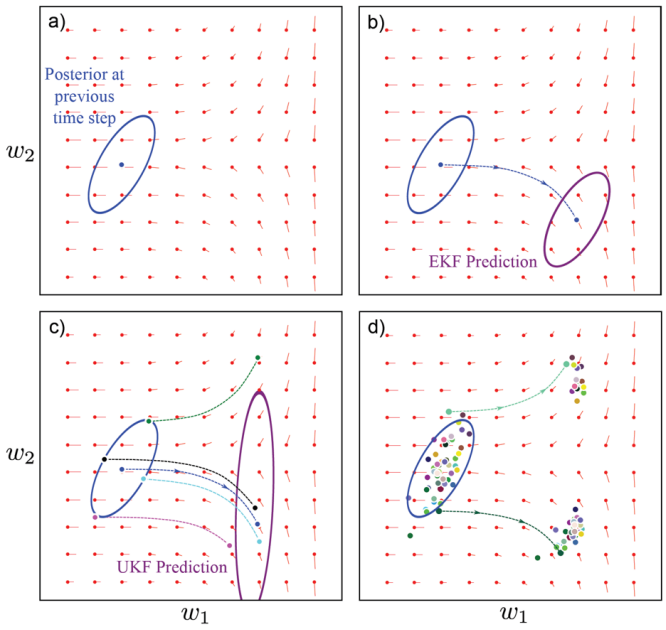



Figure 7.15: Problems with UKF: The UKF is advantageous in handling non-linear systems, but its performance depends on the selection of sigma points and weighting factors. It provides better estimates compared to the Extended Kalman Filter (EKF) in cases with significant non-linearities.

### 7.7.7 Particle Filters

**Definition 7.7.12.** Particle filters represent the posterior distribution as a weighted set of particles:

$$P(\mathbf{w}_t|\mathbf{x}_1,\ldots,\mathbf{x}_t) \approx \sum_{j=1}^{J} a_j \delta(\mathbf{w}_t - \widehat{\mathbf{w}}_t^{[j]}),$$

where $\delta$ is the Dirac delta function, $a_j$ are the weights, and $\widehat{\mathbf{w}}_t^{[j]}$ are the particle locations.

#### Advantages and Disadvantages

- ***Advantages:***
  - *Can model multi-modal and non-Gaussian distributions effectively.*
  - *Avoids the need for data association.*
- ***Disadvantages:***
  - *Computationally expensive due to the large number of particles required.*

### Condensation Algorithm

*The condensation algorithm consists of the following stages:*

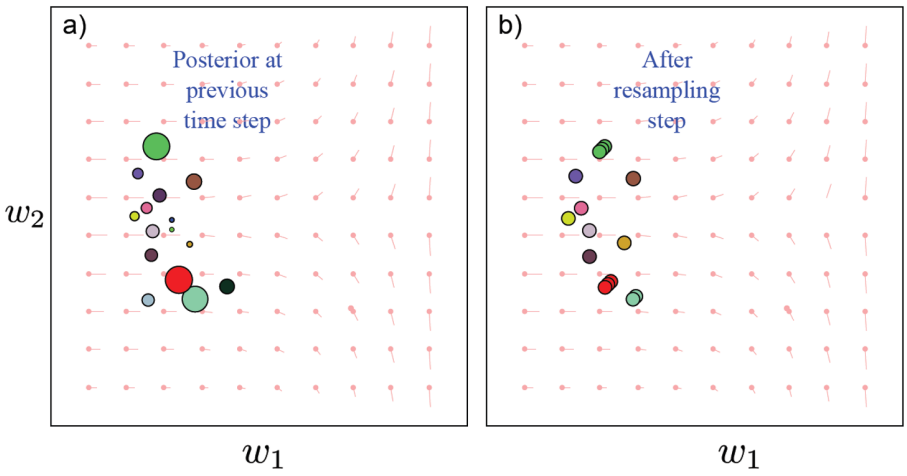



Figure 7.16: Condensation Algorithm - Step1

1. ***Resampling:*** *Particles are resampled based on their weights. Particles with higher weights are duplicated more frequently, while those with lower weights are removed. This process results in a set of unweighted particles.*

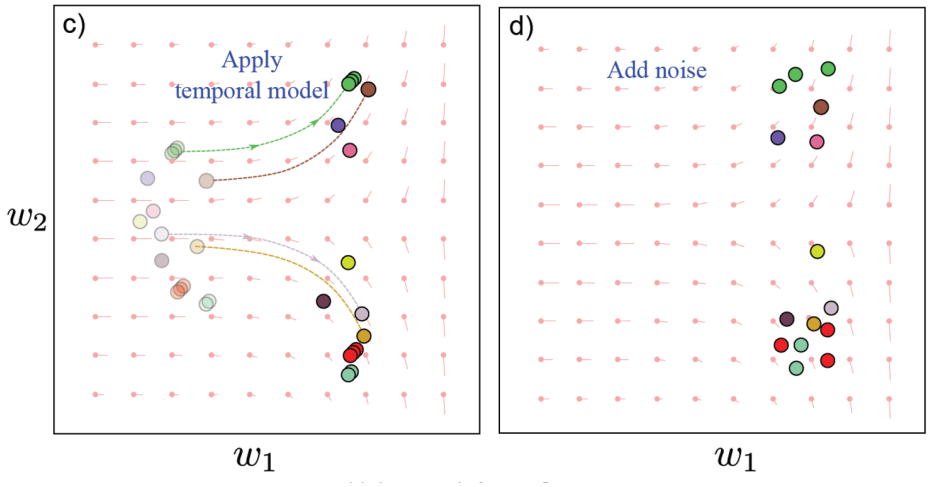



Figure 7.17: Condensation Algorithm - Step2

2. ***Propagation:*** *The unweighted particles are propagated forward through the temporal model:*

$$\mathbf{w}_t = f(\mathbf{w}_{t-1}) + \epsilon_p,$$

*where $f$ is the temporal model, and $\epsilon_p$ is the process noise. Noise is added to account for uncertainty in the model.*

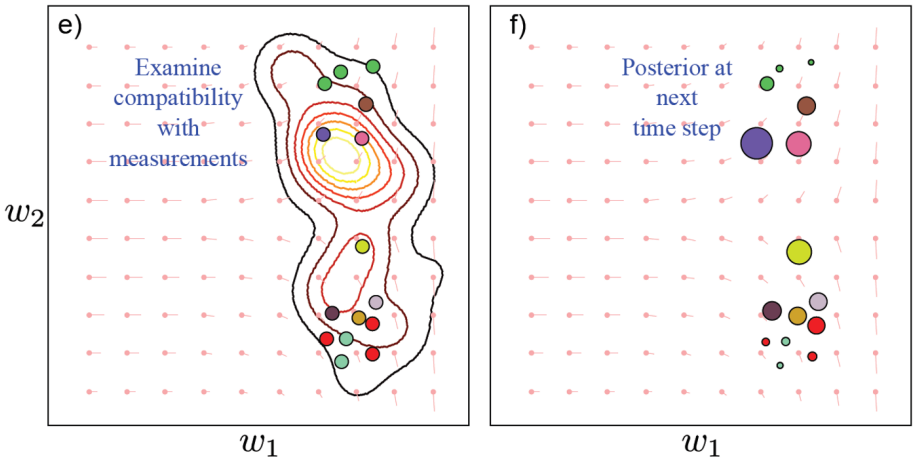



Figure 7.18: Condensation Algorithm - Step3

3. ***Measurement Update:*** *Each particle is weighted according to the measurement likelihood:*

$$w_t^{[j]} \propto P(\mathbf{x}_t|\mathbf{w}_t^{[j]}).$$

*This step evaluates how compatible each particle is with the observed measurements, leading to an updated posterior distribution.*

### Applications and Visualization

*Particle filters can represent complex distributions, as illustrated in the accompanying figures:*

- *In the first stage, particles are resampled to concentrate around regions of high likelihood.*
- *During propagation, particles follow possible trajectories influenced by noise.*
- *Measurement weighting ensures particles are concentrated in regions consistent with the observed data, forming the posterior at the next time step.*

**Comparison with Kalman Filters**

*In tracking applications, particle filters outperform Kalman filters when dealing with multi-modal distributions. For example:*

- *The Kalman filter assumes a unimodal Gaussian posterior, which may fail in cases with multiple plausible hypotheses.*
- *The particle filter, by contrast, can maintain multiple hypotheses simultaneously, as shown by the spread of particles in the posterior.*

### 7.7.8 Applications in Computer Vision

- ***Tracking Pedestrians:*** *Temporal models, such as Kalman filters and particle filters, are used to predict and update the positions of pedestrians over time. By incorporating motion models and measurement updates, these techniques handle uncertainties due to occlusions, sudden changes in direction, or interactions with other pedestrians.*
- ***Tracking Contour in Clutter:*** *Temporal models are used to adaptively track object boundaries in cluttered environments. For example, the Unscented Kalman Filter helps propagate smooth contour dynamics, while particle filters can represent multiple hypotheses, making them robust to occlusions and background clutter.*
- ***Simultaneous Localization and Mapping:*** *In SLAM, temporal models such as Extended Kalman Filters and particle filters allow robots to iteratively refine their estimates of both position and the surrounding environment. These models handle sensor noise and dynamic changes in the map, enabling robust navigation in real-world settings.*

# Chapter 8

# Shape Models and Fully Connected Neural Network Models

## 8.1 Models for Shape

### 8.1.1 What is Shape?

**Definition 8.1.1** (Shape)**.** Shape is all the geometrical information that remains when location, scale, and rotational effects are filtered out from an object.

### 8.1.2 Representing Shape

- ***Algebraic Modeling:*** *Represent simple shapes like lines or conics algebraically.*
- ***Landmark Points:*** *Discrete samples from an underlying contour. These can be:*
  - *Ordered (continuous contour),*
  - *Ordered with wrapping (closed contour),*
  - *More complex collections (closed and open contours).*

### 8.1.3 Snakes (Active Contours)

**Definition 8.1.2** (Snake Model)**.** A snake is represented by $N$ 2D landmark points. Given $x$ is the image we observed and $W$ is the landmark points, the model maximizes the posterior:

$$P(W|x) \propto P(x|W)P(W),$$

where:

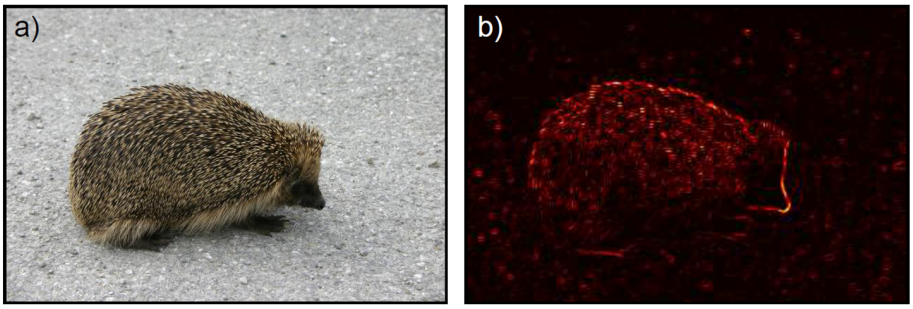



Figure 8.1: Edge-based Snake Likelihood. Bright regions represent higher likelihood of edges.

- $P(x|W)$ is observting an image $x$ given landmark points $W$. It encourages the contour to lie on image borders (likelihood)

- $P(W)$ as prior encourages smoothness and equal spacing (prior)

**Remark.** *Initialize contour and let it evolve until it grabs onto an object across the image, hence called snake or active contour. convergence depens on the initialization. Inference involves non-linear optimization as there is no closed-form solution.*

### Edge-based Likelihood

*The likelihood of observing an image* $\mathbf{x}$ *given the contour parameterized by* $\mathbf{W}$ *is defined as:*

$$P(\mathbf{x}|\mathbf{W}) \propto \prod_{n=1}^{N} \exp(\textit{sobel}[\mathbf{x}, \mathbf{w}_n]),$$

*where:*

- $\textit{sobel}[\mathbf{x}, \mathbf{w}_n]$*: A function (e.g., the Sobel operator) that computes the gradient magnitude at the contour points* $\mathbf{w}_n$*.*

- $\mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N\}$*: The set of contour points.*

*This likelihood has the correct property of assigning high probabilities to points near edges. However, it is not smooth in regions far from the edges, making it unsuitable for optimization.*

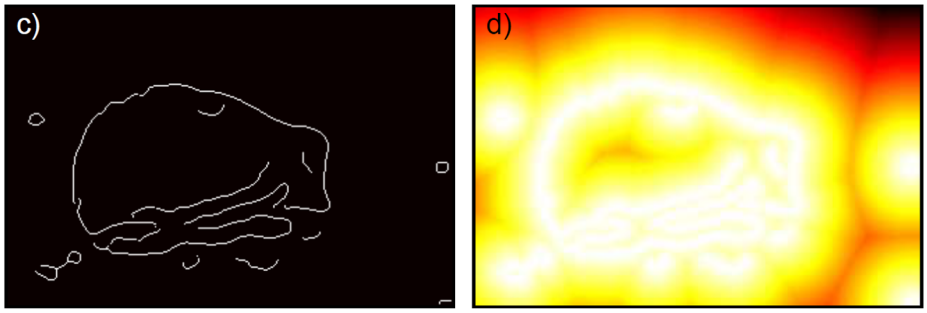



Figure 8.2: Distance-based Snake Likelihood. Bright regions represent high likelihood.

**Distance-based Likelihood**

*To improve the likelihood function's smoothness, a distance transform of the image can be used. The updated likelihood is:*

$$P(\mathbf{x}|\mathbf{W}) \propto \prod_{n=1}^{N} \exp\left(-\left(dist[\mathbf{x}, \mathbf{w}_n]\right)^2\right),$$

*where:*

- $dist[\mathbf{x}, \mathbf{w}_n]$*: The distance from a point* $\mathbf{w}_n$ *on the contour to the nearest edge in the image (e.g., computed using the Canny edge detector).*

*This distance-based likelihood varies smoothly with distance from the image edges, making it better suited for optimization.*

### 8.1.4 Prior for Snake Models

*The prior* $P(\mathbf{W})$ *in snake models is designed to encourage smoothness in the contour. It incorporates terms that penalize uneven spacing between contour points and high curvature. The prior is defined as:*

$$P(\mathbf{W}) \propto \prod_{n=1}^{N} \exp\left[\alpha\, space[\mathbf{W}, n] + \beta\, curve[\mathbf{W}, n]\right],$$

*where:*

- $space[\mathbf{W}, n]$*: A term encouraging equal spacing between adjacent points.*
- $curve[\mathbf{W}, n]$*: A term encouraging low curvature of the contour.*
- $\alpha, \beta$*: Weights controlling the relative importance of spacing and curvature penalties.*

**Encouraging Equal Spacing** *The term space*$[\mathbf{W}, n]$ *ensures that the distances between adjacent contour points are approximately equal. It is defined as:*

$$\textit{space}[\mathbf{W}, n] = -\left(\sqrt{\frac{\sum_{n=1}^{N}(\mathbf{w}_n - \mathbf{w}_{n-1})^\top(\mathbf{w}_n - \mathbf{w}_{n-1})}{N}} - \sqrt{(\mathbf{w}_n - \mathbf{w}_{n-1})^\top(\mathbf{w}_n - \mathbf{w}_{n-1})}\right)^2.$$

**Encouraging Low Curvature** *The term curve*$[\mathbf{W}, n]$ *penalizes large deviations in curvature between consecutive contour points. It is defined as:*

$$\textit{curve}[\mathbf{W}, n] = -(\mathbf{w}_{n-1} - 2\mathbf{w}_n + \mathbf{w}_{n+1})^\top(\mathbf{w}_{n-1} - 2\mathbf{w}_n + \mathbf{w}_{n+1}).$$

### 8.1.5 Inference in Snake Models

*To estimate the optimal contour* $\hat{\mathbf{W}}$*, we maximize the posterior probability:*

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{x}),$$

*which can be rewritten using Bayes' theorem:*

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} \left[\log P(\mathbf{x}|\mathbf{W}) + \log P(\mathbf{W})\right],$$

*where:*

- $P(\mathbf{x}|\mathbf{W})$*: The likelihood, encouraging the contour to align with the edges in the image.*
- $P(\mathbf{W})$*: A prior on the contour, which encourages smoothness or regularization of the contour.*

***Challenges in Optimization**:*

- ***No closed-form solution:** The optimization problem must be solved numerically.*
- ***Non-linear optimization:** Methods such as gradient descent are required to find the maximum posterior probability.*
- ***Dimensionality:** The number of unknowns is* $2N$*, where* $N$ *is the number of contour points, as each point has both* $x$*- and* $y$*-coordinates.*

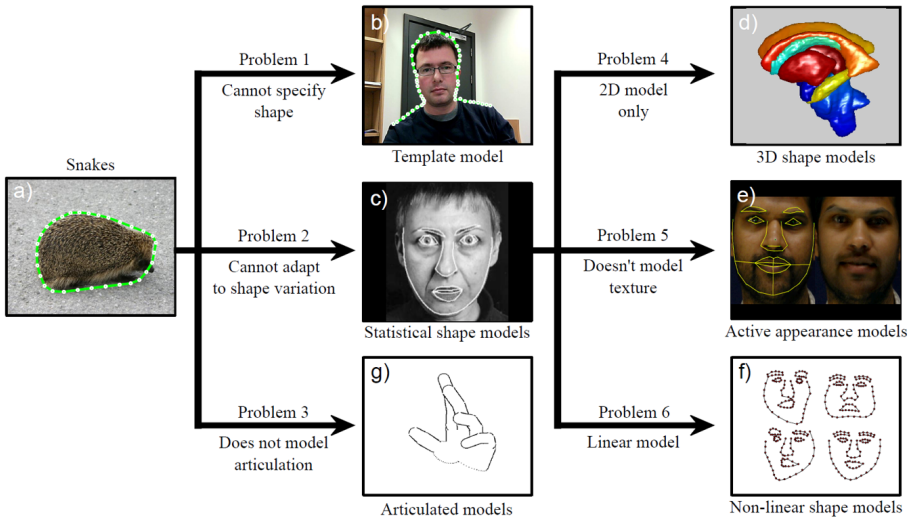



Figure 8.3: Relationships between models.

### 8.1.6 Template Models

**Definition 8.1.3** (Template Shape Model)**.** Template shape model is where the shape is defined by **known** landmark points $W$ transformed into the image space using parameters $\theta$, which is what we need to find. Inference maximizes:

$$\mathcal{L}(\theta) = \text{distance transform}(\theta).$$

The likelihood of the image $\mathbf{x}$ given the contour $\mathbf{W}$ and transformation parameters $\Psi$ is based on the distance transform and is defined as:

$$P(\mathbf{x}|\mathbf{W}, \Psi) \propto \prod_{n=1}^{N} \exp\left(-\left(\text{dist}[\mathbf{x}, \text{trans}[\mathbf{w}_n, \Psi]]\right)^2\right),$$

where:

- $\text{dist}[\mathbf{x}, \text{trans}[\mathbf{w}_n, \Psi]]$: The distance from the transformed point $\text{trans}[\mathbf{w}_n, \Psi]$ to the nearest edge in the image $\mathbf{x}$,
- $\text{trans}[\mathbf{w}_n, \Psi]$: A transformation applied to the contour point $\mathbf{w}_n$ based on parameters $\Psi$.
- No prior on parameters, but could have one.

No prior on parameters (but could have one).

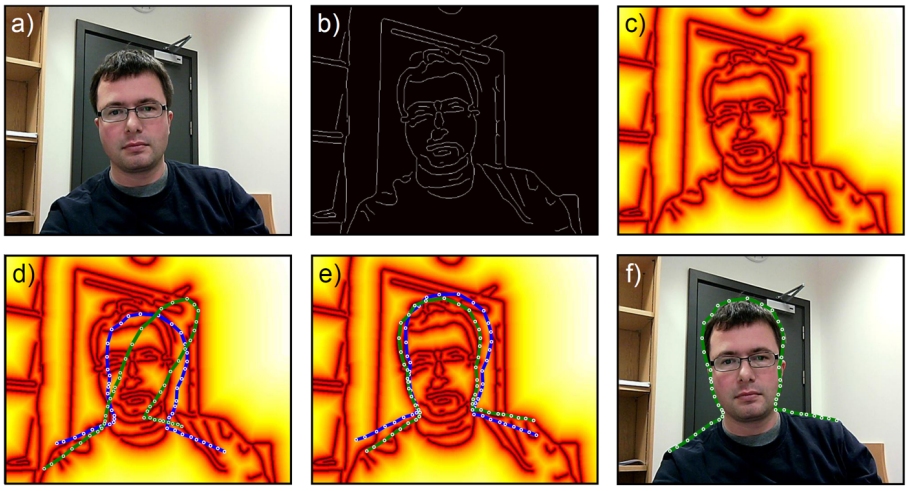



Figure 8.4: Shape Template Model. (d) is the initialized template. (e) is a different initialization with the green fits the shape well.

**Theorem 8.1.4** (Maximum Likelihood Inference for Template Model)**.** *To estimate the transformation parameters $\Psi$ that best explain the observed data $\mathbf{x}$, we use the Maximum Likelihood (ML) approach:*

$$\hat{\Psi} =_{\Psi} L =_{\Psi} \log P(\mathbf{x}|\mathbf{W}, \Psi),$$

*where:*

$$\hat{\Psi} =_{\Psi} -\sum_{n=1}^{N} \big(dist[\mathbf{x}, trans(\mathbf{w}_n, \Psi)]\big)^2.$$

- *The optimization does not have a closed-form solution.*
- *Non-linear optimization techniques are required.*

*The gradient of the log-likelihood can be computed using the chain rule:*

$$\frac{\partial L}{\partial \Psi} = -\sum_{n=1}^{N}\sum_{j=1}^{2} \frac{\partial \big(dist[\mathbf{x}, \mathbf{w}'_n]\big)^2}{\partial \mathbf{w}'_{jn}} \frac{\partial \mathbf{w}'_{jn}}{\partial \Psi},$$

*where* $\mathbf{w}'_n = trans(\mathbf{w}_n, \Psi)$.

**Remark** (Iterative Closest Points (ICP) Alternative)**.** *An alternative to maximum likelihood optimization is the Iterative Closest Points (ICP) algorithm. The steps are as follows:*

1. *Identify the nearest edge point to each landmark point.*

2. *Compute the transformation in closed form for these correspondences.*
3. *Repeat until convergence.*

*This method is particularly useful in cases where closed-form transformations can be iteratively refined for accuracy.*

### 8.1.7 Statistical Shape Models

*Statistical shape models, also referred to as:*

- ***Point distribution models**, or*
- ***Active shape models** (as they adapt to the image),*

*are used to represent and analyze variability in shapes. The key components of this model include the likelihood, prior, and learning mechanisms.*

#### Likelihood

*The likelihood function models the relationship between the observed data $\mathbf{x}_i$ and the underlying shape parameters $\mathbf{w}_i$. It is based on a distance transform:*

$$P(\mathbf{x}_i|\mathbf{w}_i) \propto \prod_{n=1}^{N} \exp\left(-\left(dist[\mathbf{x}_i, trans[\mathbf{w}_{in}, \Psi_i]]\right)^2\right),$$

*where:*

- *$dist[\mathbf{x}_i, trans[\mathbf{w}_{in}, \Psi_i]]$: The distance from the transformed point $\mathbf{w}_{in}$ to the nearest edge in the image,*
- *$trans[\mathbf{w}_{in}, \Psi_i]$: A transformation applied to the contour points based on parameters $\Psi_i$,*
- *$N$: The number of points on the contour.*

#### Prior

*The prior encodes the statistical distribution of the shape parameters $\mathbf{w}_i$:*

$$P(\mathbf{w}_i) = \mathcal{N}(\mu, \Sigma),$$

*where:*

- *$\mu$: The mean shape,*
- *$\Sigma$: The covariance matrix encoding the variability of the shape.*

### Learning

*To learn the statistical model, we need to estimate $\mu$ and $\Sigma$. However, the given examples are typically provided in a transformed space:*

$$\mathbf{w}'_{in} = \textit{trans}[\mathbf{w}_{in}, \Psi_i].$$

*The learning process involves:*

1. *Computing the inverse transformation to obtain the original points:*

$$\mathbf{w}_{in} = \textit{trans}[\mathbf{w}'_{in}, \Psi_i^{-1}],$$

*where $\Psi_i^{-1}$ is the inverse of the transformation parameters.*

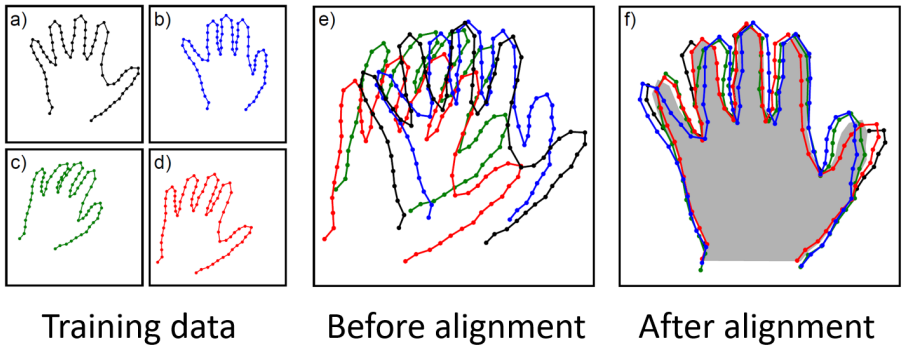



Figure 8.5: Generalized Procrustes Analysis. By picking one mean, we can do the alignment of the landmarks.

2. *Using **generalized Procrustes analysis** to align the points and learn the parameters $\mu$ and $\Sigma$:*

   - *Update transformations $\Psi_i^{-1}$ to map the landmark points to the current mean:*

$$\hat{\Psi}_i^{-1} = \arg\min_{\Psi_i^{-1}} \sum_{n=1}^{N} \left| \textit{trans}[\mathbf{w}'_{in}, \Psi_i^{-1}] - \mu_n \right|^2.$$

   - *Update the mean $\mu$ to be the average of the transformed values:*

$$\hat{\mu} = \arg\min_{\mu} \sum_{n=1}^{N} \left| \textit{trans}[\mathbf{w}'_{in}, \Psi_i^{-1}] - \mu_n \right|^2.$$

*This iterative process alternates between aligning the transformations and updating the mean, after which the covariance parameters $\Sigma$ are learned.*

### Inference

*To infer the optimal shape parameters $\hat{\mathbf{w}}$, we maximize the posterior probability:*

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \max_{\Psi} \left[ \sum_{n=1}^{N} - (\textit{dist}[\mathbf{x}_i, \textit{trans}[\mathbf{w}_{in}, \Psi]])^2 + \log P(\mathbf{w}_i) \right].$$

*Key points about the inference process:*

- *There is no closed-form solution, requiring non-linear optimization methods.*
- *The Iterative Closest Point (ICP) approach can be used for optimization.*
- *The large number of parameters can make the process inefficient, which is why subspace models are often preferred.*

## Subspace Shape Models

**Definition 8.1.5.** The subspace shape model generates data using the following equation:

$$\mathbf{w}_i = \boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}_i + \boldsymbol{\epsilon}_i,$$

where:

- $\boldsymbol{\mu}$ is the mean shape.
- $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_K]$ contains $K$ basis functions in its columns.
- $\boldsymbol{\epsilon}_i$ is normal noise with covariance $\sigma^2\mathbf{I}$.

**Remark.** *The model can also be written in an expanded form as:*

$$\mathbf{w}_i = \boldsymbol{\mu} + \sum_{k=1}^{K} \boldsymbol{\phi}_k h_{ik} + \boldsymbol{\epsilon}_i.$$

### Approximating with Subspace

**Theorem 8.1.6.** *A vector $\mathbf{w}_i$ can be approximated as a weighted sum of the basis functions:*

$$\mathbf{w}_i \approx \boldsymbol{\mu} + \sum_{k=1}^{K} \boldsymbol{\phi}_k h_{ik}.$$

*This approximation works surprisingly well, even with a small number of basis functions $K$.*

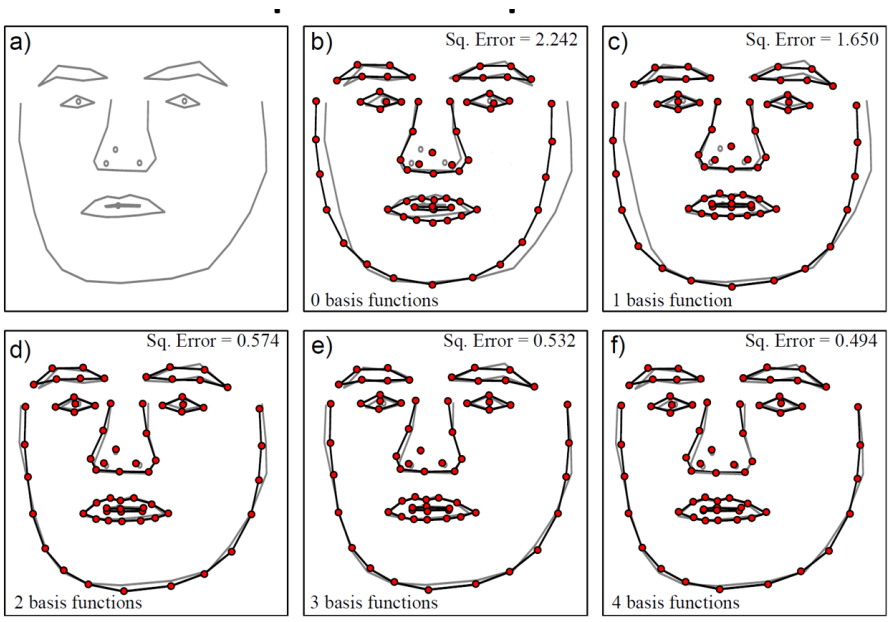



Figure 8.6: Subspace Shape model.

### 8.1.8 Probabilistic PCA

**Definition 8.1.7.** Probabilistic PCA (PPCA) is an extension of Principal Component Analysis (PCA) where the data is assumed to be generated from a probabilistic model. The generative model for PPCA is defined as:

$$\mathbf{w}_i = \boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}_i + \boldsymbol{\epsilon}_i,$$

where:

- $\boldsymbol{\mu}$ is the mean shape,
- $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_K]$ contains $K$ basis functions,
- $\boldsymbol{\epsilon}_i$ is Gaussian noise with covariance $\sigma^2\mathbf{I}$.

#### Probabilistic Framework

*The probabilistic version of the generative model assumes:*

$$P(\mathbf{w}_i|\mathbf{h}_i, \boldsymbol{\mu}, \boldsymbol{\Phi}, \sigma^2) = Norm_{\mathbf{w}_i}[\boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}_i, \sigma^2\mathbf{I}],$$

*and introduces a prior on the latent variables:*

$$P(\mathbf{h}_i) = Norm_{\mathbf{h}_i}[0, \mathbf{I}].$$

*The marginal density of the observed data is obtained by integrating over the latent variables:*

$$P(\mathbf{w}_i) = \int P(\mathbf{w}_i|\mathbf{h}_i)P(\mathbf{h}_i)\, d\mathbf{h}_i,$$

*which evaluates to:*

$$P(\mathbf{w}_i) = Norm_{\mathbf{w}_i}[\boldsymbol{\mu}, \boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma^2\mathbf{I}].$$

**Learning Parameters**

*To estimate the parameters $\boldsymbol{\mu}$, $\boldsymbol{\Phi}$, and $\sigma^2$, the following steps are taken:*

- *Compute the mean:*

$$\boldsymbol{\mu} = \frac{1}{I}\sum_{i=1}^{I}\mathbf{w}_i.$$

- *Center the data:*

$$\mathbf{W} = [\mathbf{w}_1 - \boldsymbol{\mu}, \mathbf{w}_2 - \boldsymbol{\mu}, \ldots, \mathbf{w}_I - \boldsymbol{\mu}].$$

- *Perform eigen-decomposition:*

$$\mathbf{W}\mathbf{W}^\top = \mathbf{U}\mathbf{L}^2\mathbf{U}^\top.$$

- *Choose parameters:*

$$\sigma^2 = \frac{1}{D-K}\sum_{j=K+1}^{D} L_{jj}^2, \quad \boldsymbol{\Phi} = \mathbf{U}_K(\mathbf{L}_K^2 - \hat{\sigma}^2\mathbf{I})^{1/2}.$$

**Properties of Basis Functions**

**Remark.** *The basis functions in $\boldsymbol{\Phi}$ exhibit the following properties:*

- *They are orthogonal.*
- *They are ordered by their contribution to the variance in the data.*

**Learned Hand Model**

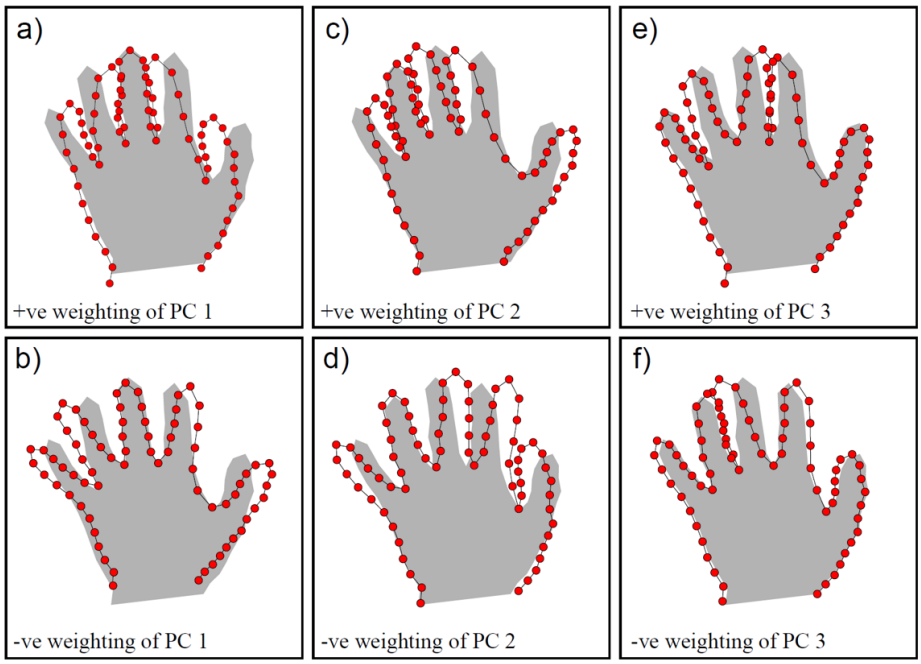



Figure 8.7: Learned Hand Model.

*This model visualizes variations in hand shape by manipulating the weights of principal components (PCs). Each PC captures a specific aspect of hand shape variability. For example:*

- *PC 1 might control overall size,*
- *PC 2 could adjust finger curvature.*

### Learned Spine Model

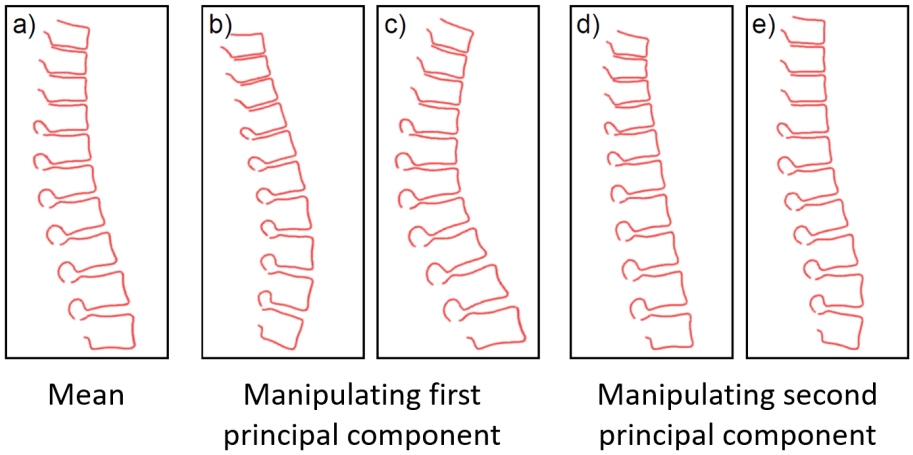



Figure 8.8: Learned Spine Model.

*Similarly, a spine model represents mean shape and variations:*

- *The first principal component might alter overall curvature,*
- *The second component could control local adjustments in vertebrae orientation.*

### Inference

*The goal of inference is to fit the model to new data by finding latent variables* $\mathbf{h}$ *and transformation parameters* $\boldsymbol{\Psi}$ *that maximize the likelihood:*

$$\hat{\mathbf{h}}, \hat{\boldsymbol{\Psi}} = \arg\max_{\mathbf{h},\boldsymbol{\Psi}} \left[ -\sum_{n=1}^{N} \left( dist[\mathbf{x}_n, trans[\boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}_n, \boldsymbol{\Psi}]] \right)^2 / \sigma^2 + \log P(\mathbf{h}) \right].$$

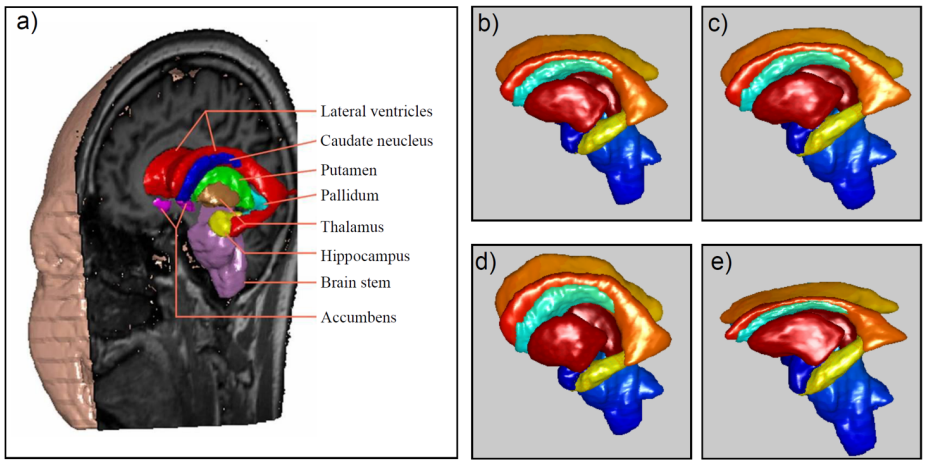
- *Weight updates:*

$$\mathbf{h} = \arg\max_{\mathbf{h}} \left[ -\sum_{n=1}^{N} dist^2/\sigma^2 - \log \|\mathbf{h}\|^2 \right].$$

- *Transformation updates:*

$$\boldsymbol{\Psi} = \textit{closed-form solution}.$$

### 3D Shape Models




Babalola *et al.* (2008)

Figure 8.9: 3D Shape Model Example.

### 8.1.9 Models for Shape and Appearance

*Statistical models for shape and appearance combine geometric information with image intensities to represent objects in a flexible manner. These models typically follow a generative approach, described as follows:*

**Definition 8.1.8.** The model involves three stages of generation:

- Draw a hidden variable $\mathbf{h}$ from a prior:

$$\Pr(\mathbf{h}_i) = \text{Norm}_{\mathbf{h}_i}[0, \mathbf{I}]$$

- Generate landmark points $\mathbf{w}_i$ using a subspace model:

$$\Pr(\mathbf{w}_i|\mathbf{h}_i) = \text{Norm}_{\mathbf{w}_i}[\mu_{\mathbf{w}} + \Phi_{\mathbf{w}}\mathbf{h}_i, \sigma^2_{\mathbf{w}}\mathbf{I}]$$

  Here, $\mu_{\mathbf{w}}$ represents the mean shape, and $\Phi_{\mathbf{w}}$ represents the basis functions for shape.

- Generate image intensities $\mathbf{x}_i$ by warping the shape and adding noise:

$$\Pr(\mathbf{x}_i|\mathbf{w}_i, \mathbf{h}_i) = \text{Norm}_{\mathbf{x}_i}[\text{warp}(\mu_{\mathbf{x}} + \Phi_{\mathbf{x}}\mathbf{h}_i, \mathbf{w}_i, \Psi), \sigma^2_{\mathbf{x}}\mathbf{I}]$$

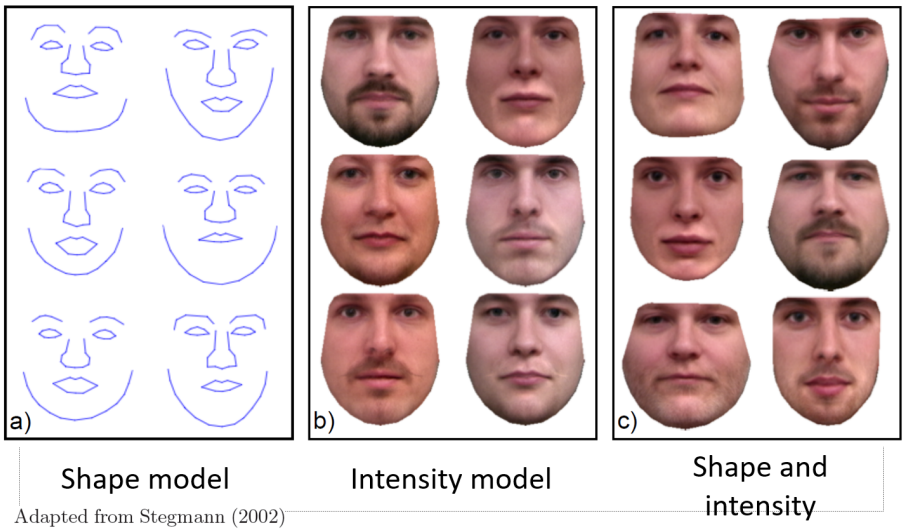
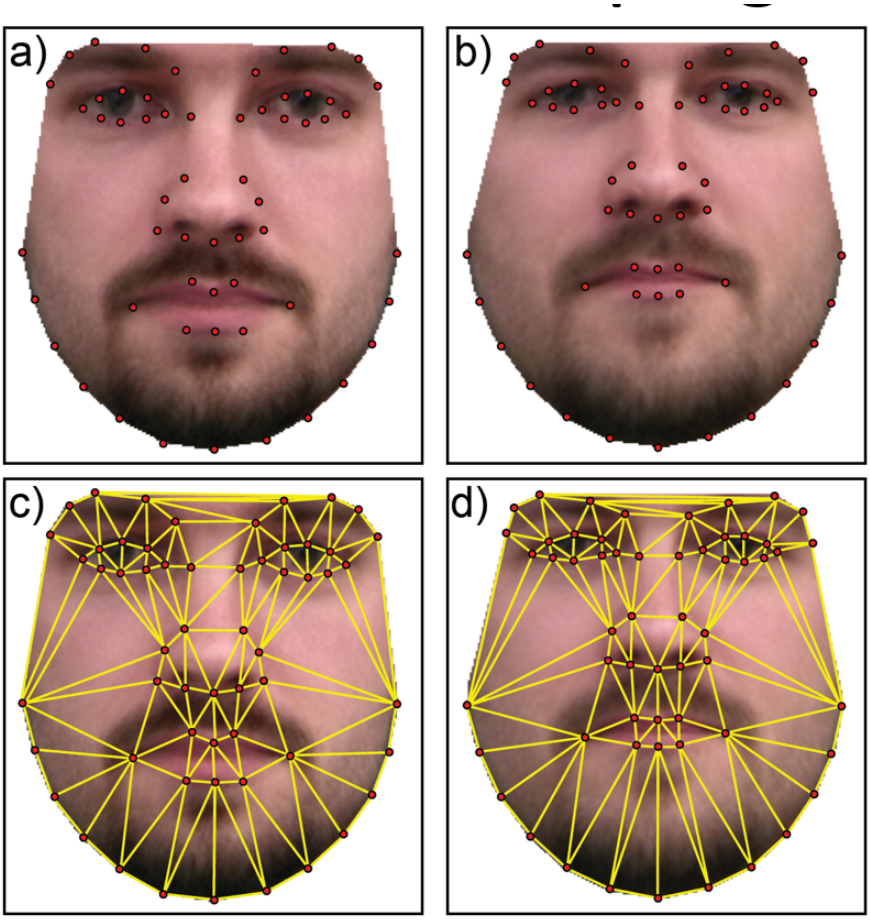



Figure 8.10: Shape and Appearance Model Example.

**Remark.** *The process involves aligning image intensities and shapes to a standard template. For instance:*

- *A shape model represents geometric features using landmark points.*
- *An intensity model generates pixel intensities conditioned on the aligned shape.*
- *The combined shape and intensity model encapsulates both geometry and appearance, enabling robust object representation.*

**Warping Images**




Adapted from Stegmann (2002)

Figure 8.11: Warping Images Illustration.

*To align image intensities with a template, the following steps are performed:*

- *Points in the image are triangulated using Delaunay triangulation.*
- *Each triangular region is warped to the corresponding region in the template using an affine transformation.*
- *The resulting warped image aligns with the statistical model's standard shape.*

**Learning Parameters**

*The goal is to learn parameters* $\{\mu_{\mathbf{w}}, \Phi_{\mathbf{w}}, \sigma^2_{\mathbf{w}}, \mu_{\mathbf{x}}, \Phi_{\mathbf{x}}, \Psi, \sigma^2_{\mathbf{x}}\}$.

1. ***Input:*** *Transformed landmark points* $\mathbf{w}$ *and warped images* $\mathbf{x}$.

2. ***Solution:***

   - *Use Procrustes analysis to reverse-transform the landmark points.*
   - *Align the observed images with the template shape.*
   - *Estimate the model parameters from the aligned data.*

**Inference**

*To fit the model to observed data:*

$$\Pr(\mathbf{x}|\mathbf{h}) = \textit{Norm}_{\mathbf{x}}[\textit{warp}(\mu_{\mathbf{x}} + \Phi_{\mathbf{x}}\mathbf{h}, \mu_{\mathbf{w}} + \Phi_{\mathbf{w}}\mathbf{h}, \Psi), \sigma^2_{\mathbf{x}}\mathbf{I}]$$

*The maximum likelihood estimate is obtained by minimizing:*

$$\hat{\mathbf{h}}, \hat{\Psi} =_{\mathbf{h},\Psi} \log \Pr(\mathbf{x}|\mathbf{h}, \Psi)$$

*This optimization can be solved using least squares or iterative methods like the Gauss-Newton method, where the Jacobian matrix* $\mathbf{J}$ *represents derivatives of the function with respect to parameters.*

### 8.1.10 Non-linear Models

*The shape and appearance models discussed previously are grounded in normal distribution assumptions. However, for more complex shapes, such assumptions might be inadequate. These cases demand more sophisticated models such as mixtures of Probabilistic Principal Component Analysis (PPCA) models or non-linear subspace models. In this section, we explore the Gaussian Process Latent Variable Model (GPLVM) as an extension of PPCA into the non-linear domain.*

### PPCA as Regression

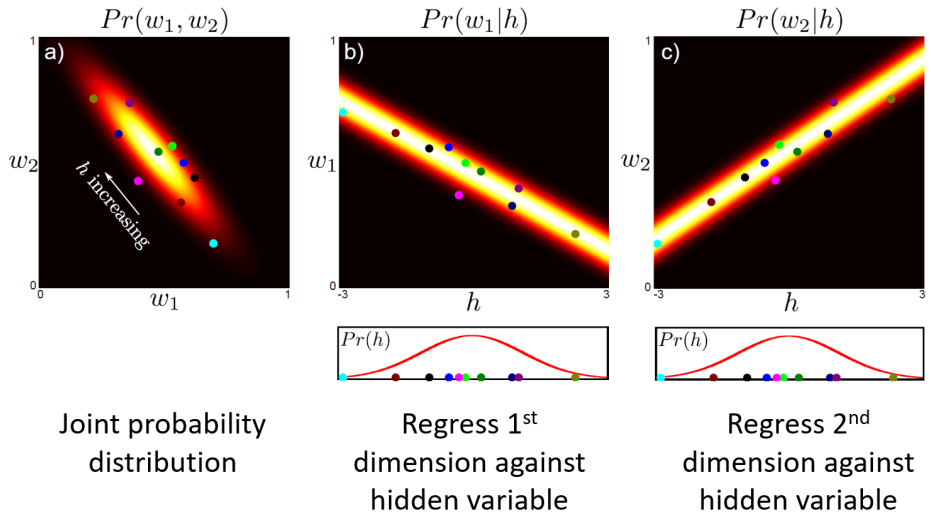



Figure 8.12: PPCA as regression.

*The PPCA model can be reframed as a regression problem. Specifically, the conditional probability* $P(\mathbf{w}|\mathbf{h}, \boldsymbol{\mu}, \boldsymbol{\Phi}, \sigma^2)$ *predicts* $\mathbf{w}$ *for a given latent variable* $\mathbf{h}$*:*

$$P(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Phi}, \sigma^2) = \int P(\mathbf{w}|\mathbf{h}, \boldsymbol{\mu}, \boldsymbol{\Phi}, \sigma^2) P(\mathbf{h}) d\mathbf{h}.$$

*This equation can be interpreted as a linear regression model for each dimension of* $\mathbf{w}$*, where:*

$$P(w_d|\mathbf{h}, \mu_d, \boldsymbol{\phi}_d, \sigma^2) = \mathcal{N}(w_d; \mu_d + \boldsymbol{\phi}_d^\top \mathbf{h}, \sigma^2).$$

### Gaussian Process Latent Variable Model (GPLVM)

*The GPLVM extends the linear regression framework of PPCA by replacing it with a non-linear regression model, specifically Gaussian Process regression. This transformation introduces several implications:*

- *Parameters* $\boldsymbol{\mu}$ *and* $\boldsymbol{\Phi}$ *can be marginalized over, allowing for flexibility in representation.*
- *The latent variable* $\mathbf{h}$*, however, cannot be marginalized over, requiring explicit optimization.*

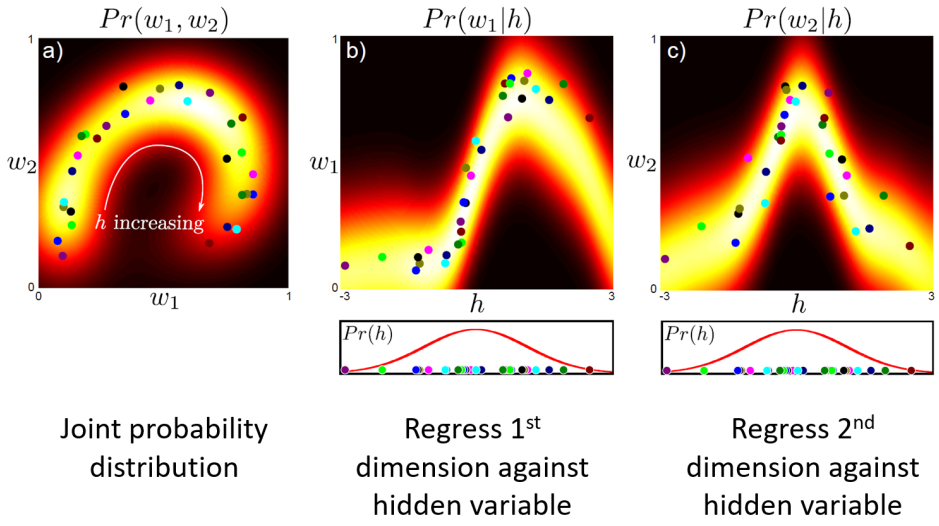



Figure 8.13: GPLVM as Regression.

### Learning the GPLVM

*To train a GPLVM, the marginal likelihood of the observed data is optimized with respect to the variance parameter $\sigma^2$. The learning objective is formulated as:*

$$\hat{\sigma}^2 = \arg\max_{\sigma^2} P(\mathbf{w}|\mathbf{X}, \sigma^2),$$

*where the marginalization is performed over the Gaussian Process parameters $\mathbf{\Phi}$.*

*In contrast to PPCA, the GPLVM also requires optimizing the latent variables $\mathbf{H}$:*

$$\hat{\mathbf{H}}, \hat{\sigma}^2 = \arg\max_{\mathbf{H}, \sigma^2} P(\mathbf{W}, \mathbf{H}|\mathbf{X}, \sigma^2).$$

*This optimization is typically performed using non-linear techniques.*

### Inference in GPLVM

*Inference in GPLVM involves predicting new values of $\mathbf{w}^*$ given a latent variable $\mathbf{h}^*$. The conditional probability is given by:*

$$P(\mathbf{w}^*|\mathbf{h}^*, \mathbf{H}, \mathbf{W}) = \mathcal{N}\left(\mathbf{w}^*; \mathbf{K}(\mathbf{h}^*, \mathbf{H})\mathbf{K}(\mathbf{H}, \mathbf{H})^{-1}\mathbf{W}, \sigma^2\right),$$

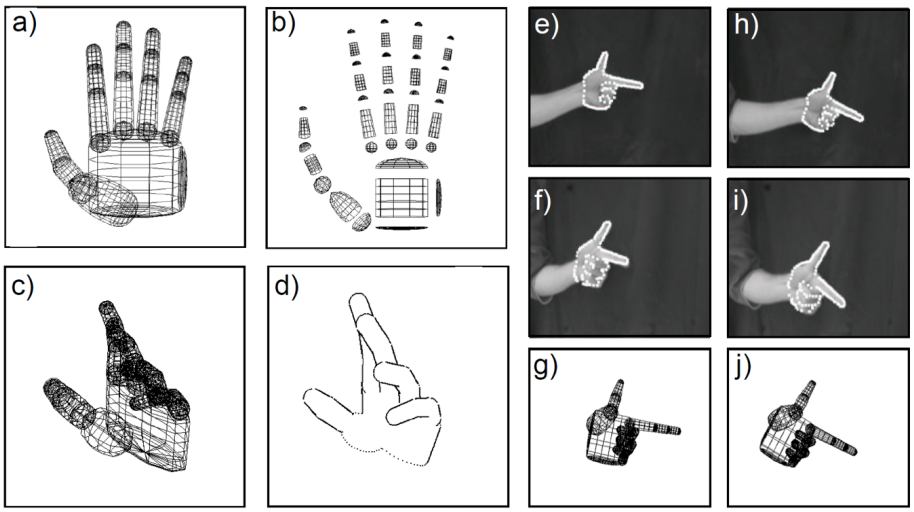
*where $\mathbf{K}$ represents the covariance matrix from the Gaussian Process.*

*To compute the density of* $\mathbf{w}^*$*, marginalization over the latent variable* $\mathbf{h}^*$ *is required:*

$$P(\mathbf{w}^*) = \int P(\mathbf{w}^*|\mathbf{h}^*)P(\mathbf{h}^*)d\mathbf{h}^*.$$

*This integral cannot be solved in closed form and often requires numerical approximation.*

### 8.1.11 Articulated Models




Adapted from Stenger *et al.* (2001a). ©2001 IEEE.

Figure 8.14: Articulated Model Example.

*Articulated models describe objects as kinematic chains, where transformations are applied sequentially. For example, the transformation of the foot is relative to the lower leg, which is itself relative to the upper leg. A root transformation,* $\mathbf{T}_{root}$*, describes the object's position relative to the camera.*

*Each part of the object can be modeled using a quadric, represented mathematically as:*

$$\mathbf{x}^\top \mathbf{\Psi} \mathbf{x} = 0,$$

*where:*

$$\mathbf{\Psi} = \begin{bmatrix} \psi_1 & \psi_2 & \psi_3 & \psi_4 \\ \psi_2 & \psi_5 & \psi_6 & \psi_7 \\ \psi_3 & \psi_6 & \psi_8 & \psi_9 \\ \psi_4 & \psi_7 & \psi_9 & \psi_{10} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

*This defines shapes such as spheres, ellipsoids, and cylinders. For truncated cylinders, the quadric is clipped using planes:*

$$\mathbf{n}^\top \mathbf{x} + d = 0,$$

*where* $\mathbf{n}$ *is the plane normal and* $d$ *is the offset.*

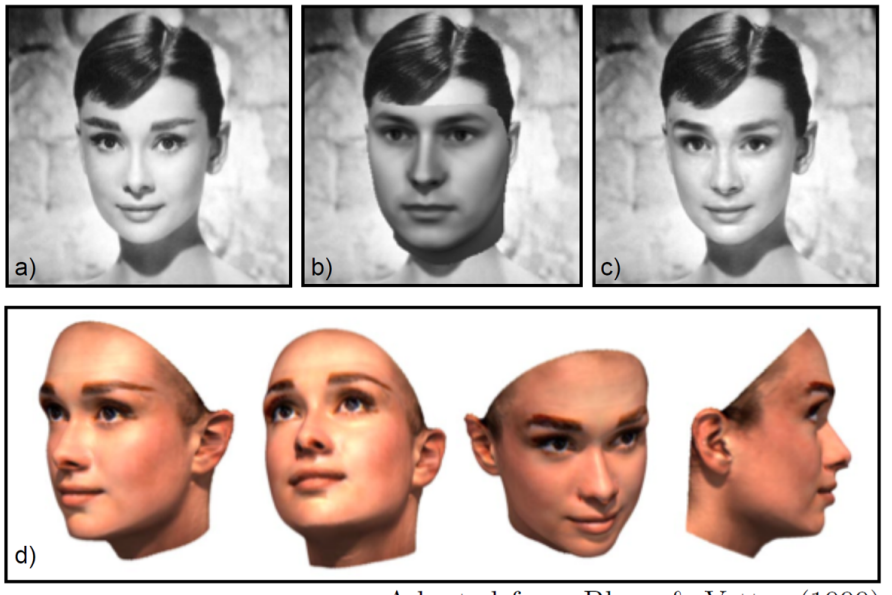
*The projection of the quadric onto the image plane forms a conic:*

$$\mathbf{u}^\top \mathbf{Q}\mathbf{u} = 0, \quad \mathbf{u} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

*Articulated models use these mathematical representations to capture the hierarchical relationships between parts and their transformations.*

## Applications

- *3D Morphable Models*




Adapted from Blanz & Vetter (1999).

Figure 8.15: 3D Morphable Models.

- *3D body modeling.*
- *Medical imaging.*

## Conclusions

- *Shape models provide varying levels of prior knowledge:*
  - *Contour Smoothness (Snakes)*
  - *Known Shape but not Position (Template)*
  - *Known shape class (Statistical Models)*
  - *Structure of parts (Articulated Models)*
- *These models relate to subspace and temporal models.*

## 8.2 Shallow Neural Networks

**Definition 8.2.1** (Shallow Neural Networks)**.** A *shallow neural network* is a neural network consisting of a single hidden layer, where the hidden layer is composed of a small number of units (neurons). The network maps an input $x$ to an output $y$ using parameterized functions and activation functions. For a shallow neural network, the relationship between input $x$, parameters $\phi$, and the output $y$ is defined as:

$$y = f(x, \phi) = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3,$$

where each $h_i$ is a hidden unit defined as:

$$h_i = a(\theta_{i0} + \theta_{i1} x),$$

and $a[z]$ is the activation function, one example is ReLU:

$$a[z] = \text{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0, \\ z & \text{if } z \geq 0. \end{cases}$$

**Example 8.2.2.** For an example with a single input $x$, the shallow network can be explicitly written as:

$$y = \phi_0 + \phi_1 a(\theta_{10} + \theta_{11} x) + \phi_2 a(\theta_{20} + \theta_{21} x) + \phi_3 a(\theta_{30} + \theta_{31} x).$$

Here:

- $h_1 = a(\theta_{10} + \theta_{11} x)$,
- $h_2 = a(\theta_{20} + \theta_{21} x)$,
- $h_3 = a(\theta_{30} + \theta_{31} x)$,

with the network output:

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3.$$

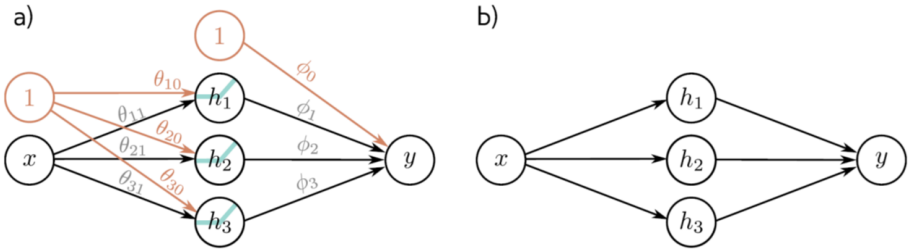



Figure 8.16: Graphical representation of a shallow neural network. (a) Detailed structure with weights labeled. (b) Simplified representation focusing on connections.

**Theorem 8.2.3.** *The network can be represented graphically in two forms:*

- ***Detailed representation:*** *Each input $x$ and bias term $1$ is connected to the hidden units $h_1, h_2, h_3$, which are then connected to the output $y$. The parameters $\theta_{ij}$ and $\phi_i$ are labeled along the edges of the graph.*

- ***Simplified representation:*** *The connections are abstracted to focus on the functional relationships between $x, h_i$, and $y$.*

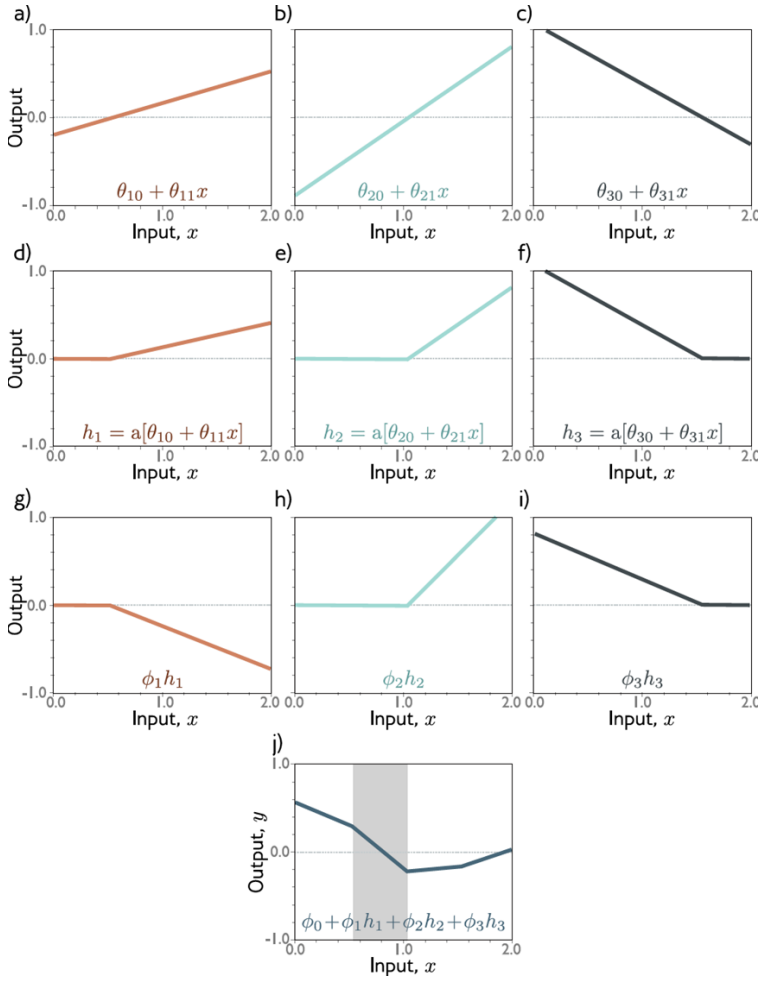



Figure 8.17: Visualization of the computation steps in a shallow neural network. Each subfigure shows the contribution of one component to the final output $y$.

*The computation for $h_1, h_2$, and $h_3$ at each layer, and the final output $y$, is illustrated below:*

- ***First Layer:*** *Compute $\theta_{i0} + \theta_{i1}x$ for $i = 1, 2, 3$ and apply ReLU activation $a[z]$.*

- ***Second Layer:*** *Use $h_i$ values from the hidden layer and compute the final output:*

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3.$$

### 8.2.1 Context Models

**Definition 8.2.4** (Context Model)**.** The goal is to estimate $P(w|x)$, where $x$ represents input data (e.g., raw pixels or preprocessed features). The input $x$ can come from:

- Raw data/pixels,
- Preprocessed filters (e.g., Gabor filters),
- Higher-level feature factories (e.g., Stacked Trees, Object Bank).

### 8.2.2 Filter Banks and Feature Factory

**Definition 8.2.5** (Filter Bank)**.** A method to transform input data $x$ into a higher-dimensional representation $z = f(x)$. Examples include:

- **Gabor Filters:** Applied for texture analysis and edge detection.

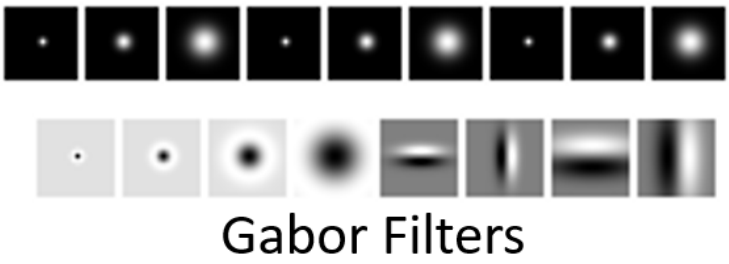



Figure 8.18:

- **Bag-of-Words:** Clusters textons to describe local image patches.

**Definition 8.2.6** (Feature Factory)**.** The concept of a feature factory shifts away from hand-crafted features ($x$) toward generating a "family" of features automatically. This approach is particularly useful in applications requiring dense and semantic feature extraction.

***Example: Semantic Texton Forests*** *Semantic Texton Forests, as introduced by Shotton et al., employ decision trees to classify image pixels into semantic categories (e.g., road, building, bicycle) based on local image patches. Each tree node evaluates conditions on pixel intensities or color channels to refine classification.*

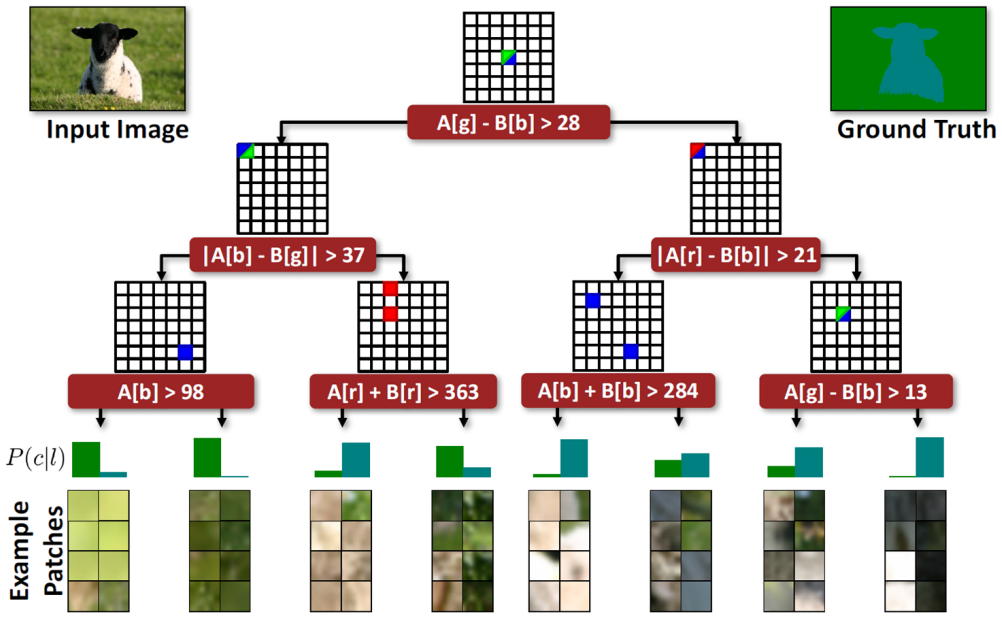



Figure 8.19: Visualization of Semantic Texton Forests: Decision trees refining semantic classification of image patches.

- *Input image patches are passed through a decision tree structure.*
- *Each node in the tree performs a comparison (e.g., $A[b] - B[b] > 28$), refining classification at deeper levels.*
- *Leaf nodes correspond to specific example patches and probability distributions $P(c|x)$ over semantic classes.*

### 8.2.3 Context Model: Second Generation Features

*Second-generation features focus on leveraging more structured and distributed representations of data for complex tasks like object detection and classification.*

#### Stacked Trees

*Stacked trees are hierarchical decision tree structures used to refine feature representation through successive layers. Each layer processes the input using tree-based decisions and outputs probabilities that serve as input for the next layer. This hierarchical approach allows for a progressively refined representation of the input.*

- ***Process:***

  - *Input image patches are processed through a decision tree where each node evaluates conditions (e.g., pixel intensity differences or sums).*

  - *Leaf nodes store probability distributions $P(c|x)$ over semantic categories based on example patches.*

  - *Outputs from one level are used as inputs for subsequent levels, forming a stacked structure.*

- ***Applications:***

  - *Semantic segmentation, where stacked trees are used to segment images into meaningful regions (e.g., road, building).*

  - *Object detection, leveraging hierarchical refinement to identify objects in images with higher accuracy.*

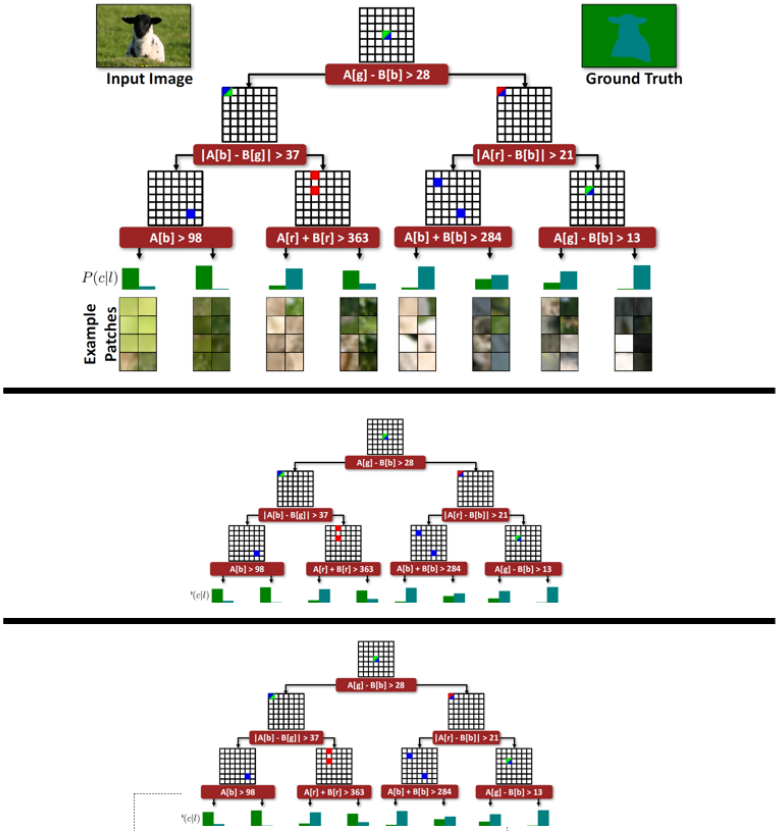



Figure 8.20: Stacked Trees: Hierarchical decision tree layers used to refine feature representations progressively.

### Object Bank

*The Object Bank model represents input images using high-level semantic features derived from a bank of object detectors. The representation is distributed and decouples the training sets for object categories, allowing flexibility in feature learning.*

- ***Features:***
  - *Describes images using a distributed representation $x$ where each feature corresponds to the confidence of a specific object detector (e.g., church, building, car).*
  - *Provides flexibility in capturing semantic content of images.*
- ***Advantages:***

  - *Enables decoupling of training sets, allowing different sets of object detectors to be trained independently.*
  - *Scales well to large datasets and a variety of object categories.*

**Non-Distributed vs. Distributed Representations**

*Distributed representations divide the feature space into combinatorially rich sub-partitions, offering greater flexibility and expressiveness compared to non-distributed methods like clustering.*

- ***Non-Distributed Representations:***
  - *Use clustering to divide the feature space into distinct regions.*
  - *The number of distinct regions is linear with respect to the number of parameters, limiting expressiveness.*
- ***Distributed Representations:***
  - *Utilize multi-clustering, creating non-mutually exclusive sub-partitions to increase the number of distinguishable configurations.*
  - *Enables combinatorial feature generation, which allows for richer and more expressive models.*

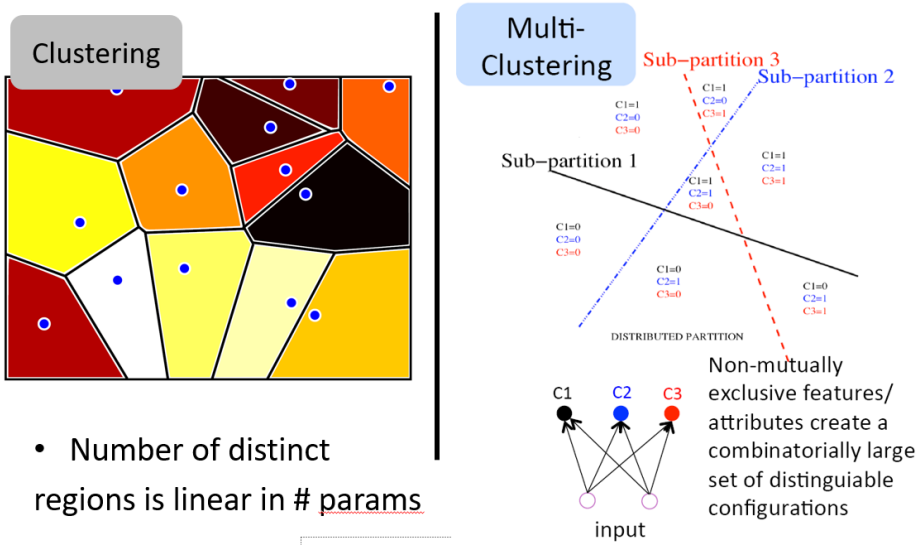



Figure 8.21: Non-Distributed vs. Distributed Representations: Distributed representations allow for richer and more flexible feature spaces.

### 8.2.4 Feed-forward Networks

**Definition 8.2.7** (Feed-forward Network)**.** A feed-forward network is a type of neural network where information flows in one direction from the input layer through the hidden layers to the output layer. There are no cycles or loops in this architecture. The network consists of:

- **Input layer:** Accepts the input data $x \in \mathbb{R}^d$, where $d$ is the dimensionality of the input.
- **Hidden layers:** Perform transformations on the input data via weights, biases, and activation functions. These layers extract features and map them to a latent space.
- **Output layer:** Maps the final latent representation to the target output, which can be a class probability distribution or a regression value.

### 8.2.5 Type of Layers

*Let $x \in \mathbb{R}^d$ denote the input, $W^{(l)}$ the weight matrix of layer $l$, and $b^{(l)}$ the bias vector of layer $l$. Each layer applies the following transformations:*

#### Affine Layer (Linear Transformation)

**Definition 8.2.8** (Affine Layer)**.** An affine layer performs a linear transformation of the input:

$$z^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)},$$

where:

- $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ is the weight matrix for layer $l$,
- $b^{(l)} \in \mathbb{R}^{d_l}$ is the bias vector,
- $h^{(l-1)} \in \mathbb{R}^{d_{l-1}}$ is the output from the previous layer (or the input $x$ for the first layer).

#### Activation Functions

**Definition 8.2.9** (Activation Functions)**.** Non-linear activation functions are applied element-wise to the output of the affine layer:

$$h^{(l)} = f(z^{(l)}),$$

where $f$ is the activation function. Common activation functions include:

- **ReLU (Rectified Linear Unit):**

  $$f(x) = \max(0, x).$$

  ReLU is widely used due to its simplicity and efficiency in training deep networks.

- **ELU (Exponential Linear Unit):**

$$f(x) = \begin{cases} x, & \text{if } x > 0, \\ \alpha(e^x - 1), & \text{if } x \leq 0, \end{cases}$$

  where $\alpha > 0$ controls the slope for negative values.

- **Sigmoid:**

$$f(x) = \frac{1}{1 + e^{-x}},$$

  which squashes the input to the range $[0, 1]$.

- **Softmax:** Converts logits into probabilities:

$$f(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}, \quad i = 1, \ldots, d.$$

**Output Layer**

**Definition 8.2.10** (Output Layer)**.** The output layer aggregates the features learned by the hidden layers to produce the final result. Depending on the task:

- For **classification tasks:** Use a softmax activation function to convert the logits into probabilities over classes.
- For **regression tasks:** Use a linear activation function $f(x) = x$.

### 8.2.6 Network Flow and Layer Connections

*A feed-forward network can be expressed as the composition of layers:*

$$h^{(L)} = f^{(L)}(W^{(L)} h^{(L-1)} + b^{(L)}),$$

*where $L$ is the total number of layers. The final output $y$ is:*

$$y = f_{output}(h^{(L)}),$$

*where $f_{output}$ is the activation function applied in the output layer.*

*Each layer in a feed-forward network sequentially applies:*

1. ***Linear transformation:*** $z^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)}$,
2. ***Activation:*** $h^{(l)} = f(z^{(l)})$.

*By stacking multiple layers, the network can approximate highly complex functions, enabling it to solve tasks like classification and regression.*

### 8.2.7 Training Using Backpropagation

**Definition 8.2.11** (Backpropagation)**.** Backpropagation is an algorithm used to compute the gradient of the loss function $\mathcal{L}$ with respect to the network parameters $\theta$. It uses the chain rule to efficiently propagate gradients through the network layers. For a composition of functions $z = f(y)$ and $y = g(x)$, the chain rule is:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}.$$

#### Gradient Computation: Single Path Chain Rule

*Given $z = f(y)$, $y = g(x)$, the gradient is propagated back as follows:*

- *Compute $\Delta z = \frac{\partial z}{\partial y} \Delta y$,*
- *Compute $\Delta y = \frac{\partial y}{\partial x} \Delta x$,*
- *Combine: $\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$,*
- *Gradient: $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$.*

*This process ensures the gradient flows efficiently from output back to input.*

#### Gradient Computation: Multiple Paths Chain Rule

*For cases where the output depends on multiple intermediate nodes ($y_1, y_2, \ldots$), the chain rule generalizes:*

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x}.$$

1. ***Forward propagation:*** *Compute the values of $z$, $y_1$, $y_2$, and $x$ layer by layer.*
2. ***Backward propagation:*** *Compute gradients at each node:*

$$\Delta z = \frac{\partial z}{\partial y_1} \Delta y_1 + \frac{\partial z}{\partial y_2} \Delta y_2,$$

*where $\Delta y_1$ and $\Delta y_2$ are obtained from their respective children nodes.*

**Theorem 8.2.12** (Complexity of Backpropagation)**.** *If computing the loss for one example involves $O(n)$ operations, then backpropagation also requires $O(n)$ operations for gradient computation. This makes the algorithm efficient for training deep networks.*

**Summary of Backpropagation Steps**

- *Forward pass:*
  - *Compute activations and outputs for each layer in sequence.*
- *Backward pass:*
  - *Compute gradients for each layer starting from the output.*
  - *Use the chain rule to propagate gradients back to earlier layers.*

**Remark.** *Forward propagation computes node activations, while backward propagation computes gradients using child nodes.*

### 8.2.8 Auto-Encoders

**Definition 8.2.13** (Auto-Encoder)**.** An **Auto-Encoder** is a type of neural network designed to learn efficient, compressed representations of data. It achieves this by mapping input data to a latent representation and reconstructing the input from this representation. The network consists of two main parts:

- **Encoder:** Maps the input $x \in \mathbb{R}^d$ to a latent representation $z \in \mathbb{R}^m$, where $m \ll d$. The encoder is typically represented as:

  $$z = f_{\text{enc}}(x; \theta_{\text{enc}})$$

  where $f_{\text{enc}}$ is a neural network parameterized by $\theta_{\text{enc}}$.

- **Decoder:** Reconstructs the input $\hat{x} \in \mathbb{R}^d$ from the latent representation $z$. The decoder is typically represented as:

  $$\hat{x} = f_{\text{dec}}(z; \theta_{\text{dec}})$$

  where $f_{\text{dec}}$ is a neural network parameterized by $\theta_{\text{dec}}$.

**Remark.** *The objective of the auto-encoder is to minimize the reconstruction error, often measured as:*

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

*or cross-entropy loss for binary data. This ensures that the latent representation $z$ captures the most relevant features of the input.*

**Definition 8.2.14** (Bottleneck Architecture)**.** A bottleneck architecture is a key characteristic of auto-encoders, where the latent representation $z$ is constrained to have significantly fewer dimensions than the input $x$. This enforces the network to learn meaningful, compressed features.

### Network Structure

*The auto-encoder architecture typically consists of:*

- *Multiple **Sigmoid Layers**, which apply the activation function:*

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- *A **Bottleneck Layer**, which reduces the dimensionality to $m$.*
- *Symmetry in the encoder and decoder structure.*

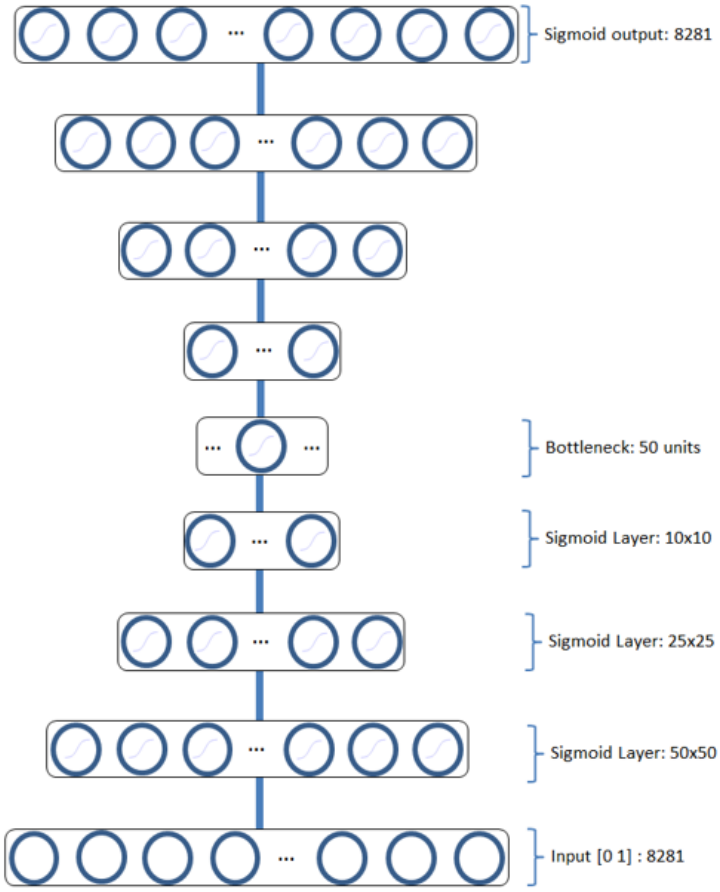



Figure 8.22: Illustration of an auto-encoder architecture, including sigmoid layers and a bottleneck with 50 units.

**Applications**

*Auto-encoders have a wide range of applications:*

- ***Dimensionality Reduction:*** *Similar to PCA but can capture non-linear relationships.*
- ***Anomaly Detection:*** *Reconstruction errors highlight anomalies.*
- ***Denoising:*** *Removing noise from input data by training with noisy inputs.*
- ***Pre-training:*** *Initializing deep networks using features learned by the auto-encoder.*

## 8.3 Multi-Layer Perceptrons (MLPs) and Iterative Training

### 8.3.1 Terminology and Architecture

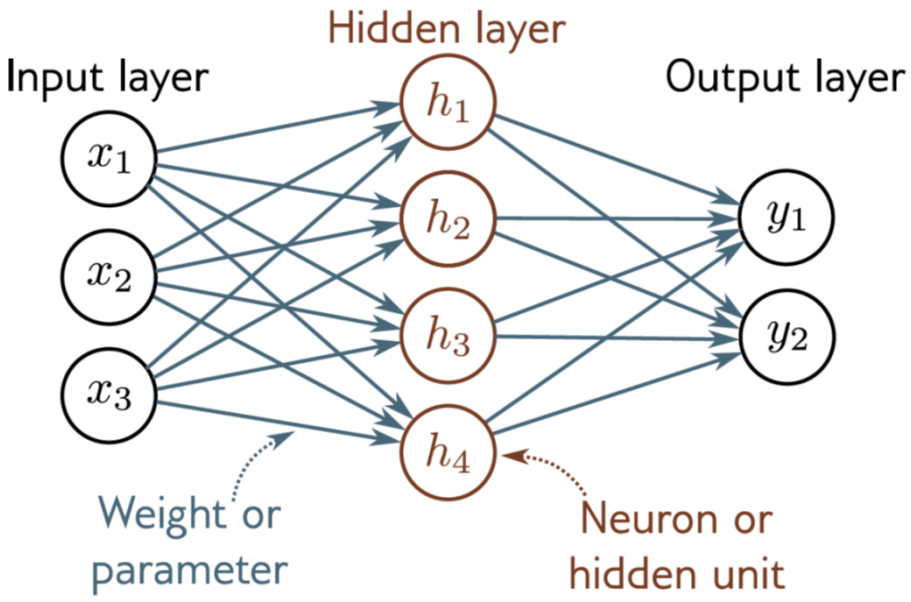



Figure 8.23: Terminology in MLPs: The figure illustrates the structure of a multi-layer perceptron with input, hidden, and output layers.

***Explanation:*** *This figure breaks down key terminologies used in MLPs:*

- *Each neuron computes a pre-activation, which is the weighted sum of inputs.*
- *The activation is the result of applying a non-linear function (e.g., ReLU) to the pre-activation.*

- *The connections between layers are represented by weights or parameters, which are learned during training.*

### 8.3.2 Shallow Network Examples

#### Single Input Example

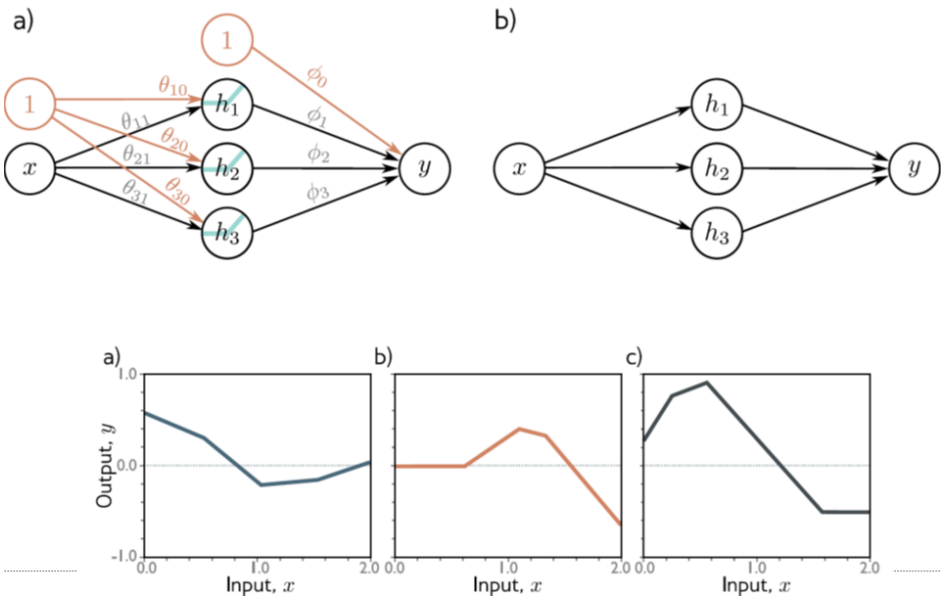
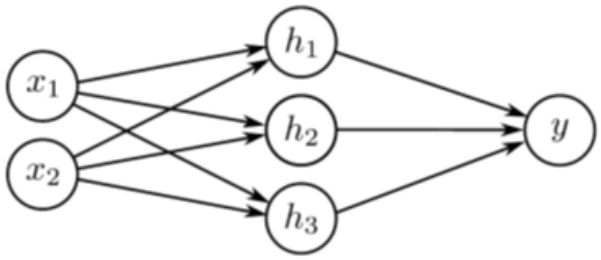



Figure 8.24: Shallow network with one input: (a) shows the linear transformation of input $x$. (b) applies ReLU activation. (c) computes the weighted sum of activations to produce the output $y$.

***Explanation:***

- *In (a), the input $x$ is transformed using weights $\theta_{ij}$ and biases $\theta_{i0}$.*
- *In (b), the ReLU activation $a(z) = \max(0, z)$ filters negative values.*
- *In (c), the final output $y$ is computed as $y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$, where $h_i$ are the activated hidden layer values.*

**Two Input Example**




$$
\begin{aligned}
h_1 &= \mathrm{a}[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2] \\
h_2 &= \mathrm{a}[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2] \\
h_3 &= \mathrm{a}[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]
\end{aligned}
$$

$$
y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3
$$

Figure 8.25: Shallow network with two inputs: (a)-(c) show intermediate activations $h_1, h_2, h_3$ based on inputs $x_1$ and $x_2$. The final output $y$ is a weighted sum of these activations.

***Explanation:***

- *Intermediate activations are computed as $h_i = a(\theta_{i0} + \theta_{i1}x_1 + \theta_{i2}x_2)$, where $a$ is the activation function.*
- *The output $y$ is obtained by combining the activations as $y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$.*
- *This demonstrates how a shallow network combines multiple inputs linearly in the transformed space.*

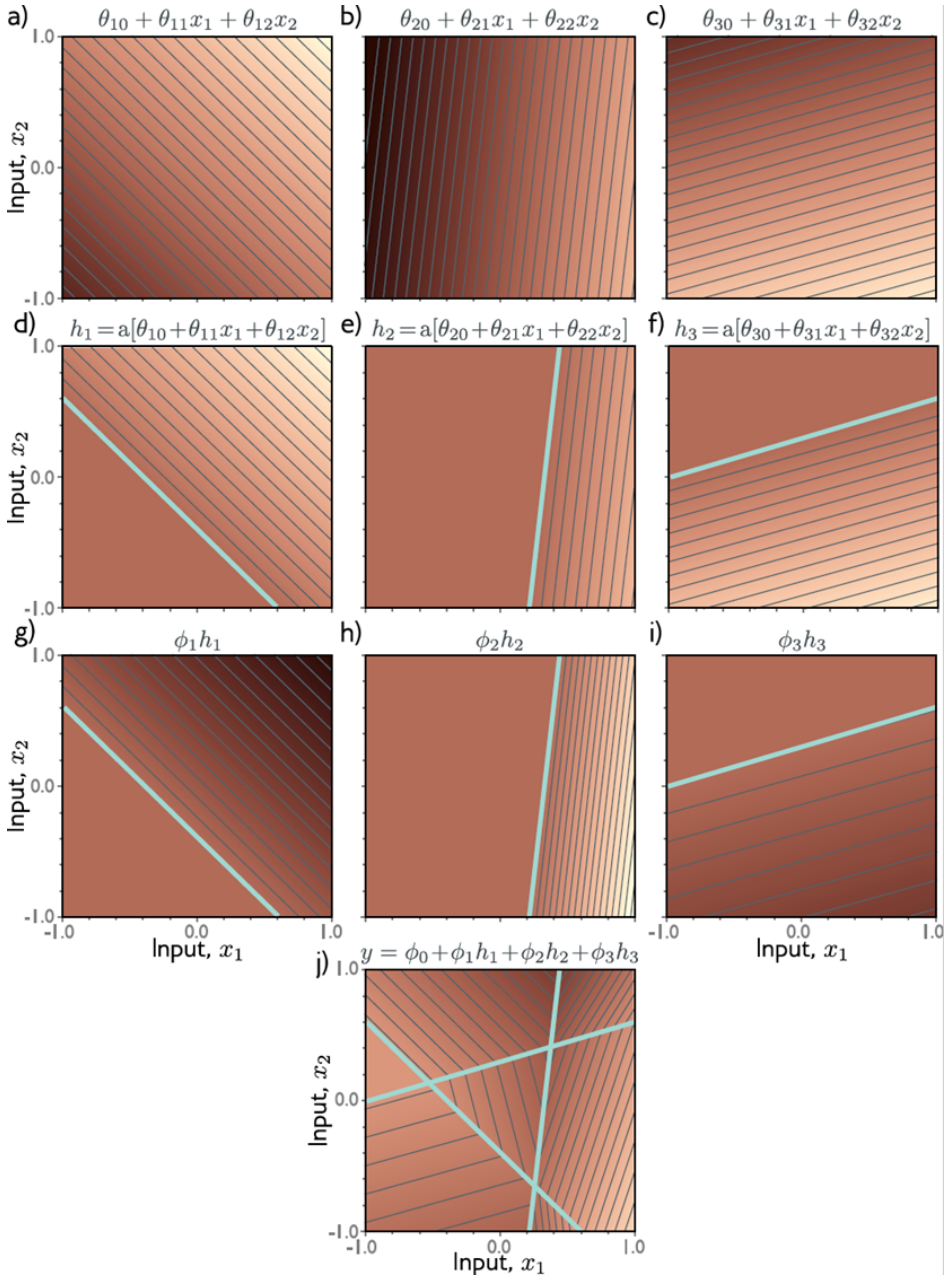



Figure 8.26: Decision surfaces for a shallow network with two inputs: (a)-(i) show how individual activations $h_1$, $h_2$, $h_3$ contribute to linear decision boundaries, culminating in a combined boundary in (j).

***Explanation:***

- *(a)-(i) illustrate the linear boundaries formed by each activation $h_i$ in the input space.*
- *(j) combines these boundaries to create a piecewise linear decision surface that segments the input space.*

### 8.3.3 Deeper Network Examples

**Single Input Example**

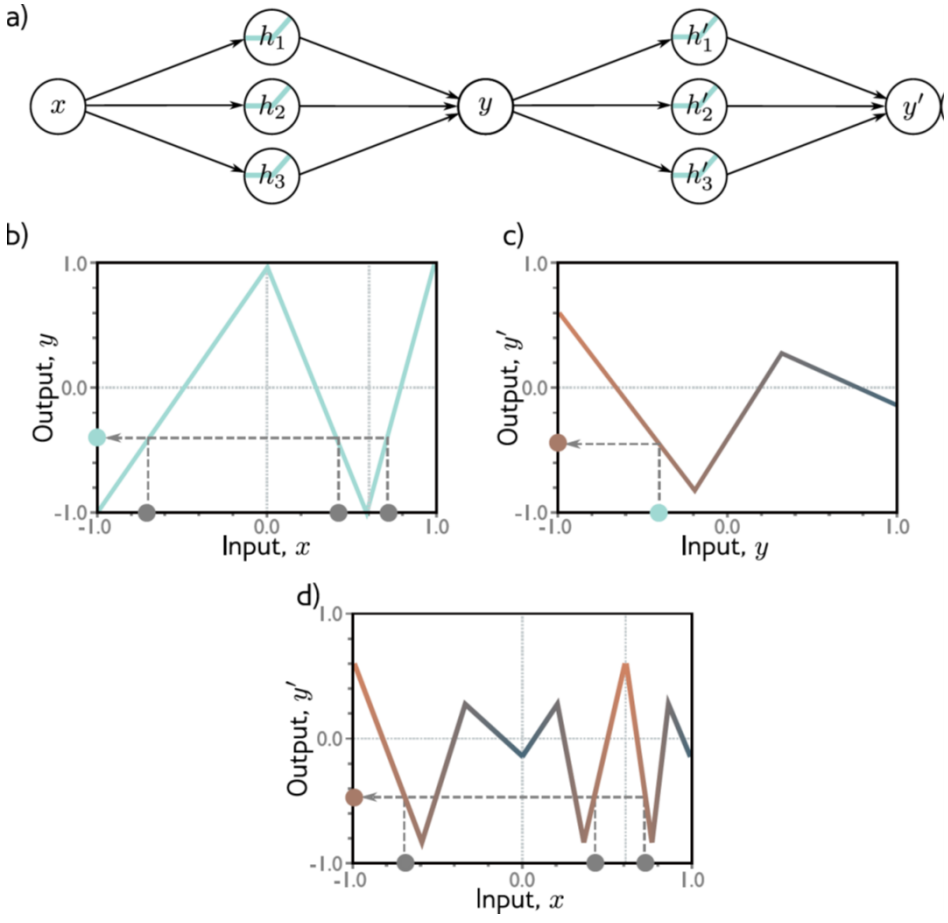



Figure 8.27: Deeper network with one input: (a) depicts a deeper structure. (b)-(d) demonstrate the increasing complexity of the learned features and their influence on the output $y$.

***Explanation:***

- *Deeper architectures stack multiple hidden layers, allowing for hierarchical feature extraction.*
- *The network transforms the input through successive layers, capturing complex patterns in the data.*
- *The final output reflects these combined transformations, enabling the model to learn non-linear mappings.*

**Two Input Example**

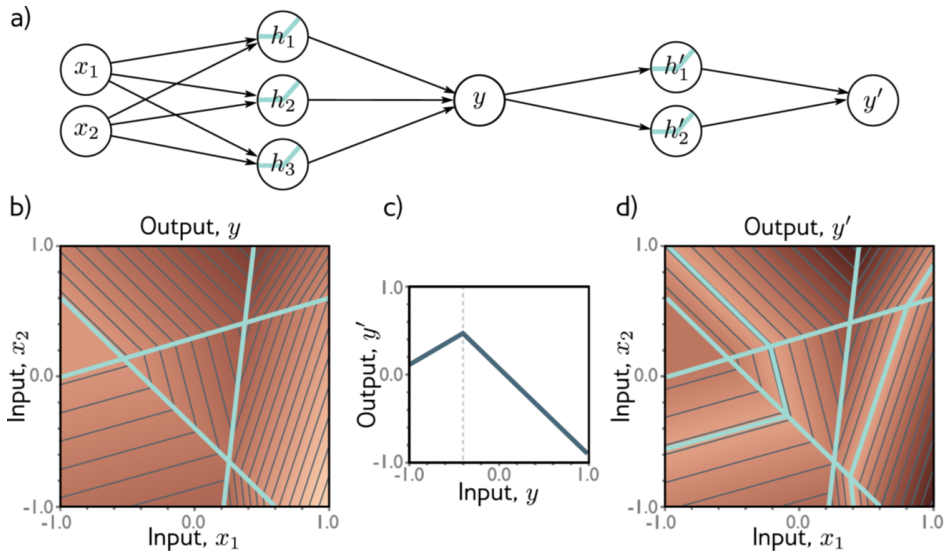



Figure 8.28: Deeper network with two inputs: (a) represents the architecture. (b)-(d) illustrate the progressive refinement of decision boundaries through the layers.

***Explanation:***

- *(a) shows how inputs $x_1$ and $x_2$ are processed through deeper layers to form complex activations.*
- *(b)-(d) depict the evolution of decision boundaries, with deeper layers enabling the network to separate data points in highly non-linear regions of the input space.*
- *The final decision boundaries reflect the network's ability to model complex input-output relationships.*

### 8.3.4 Weight Training

*Weight training in neural networks involves adjusting the parameters (weights) to minimize a loss function. The following slides demonstrate key concepts and examples.*

**Weight Training Workflow**

- ***Step 1:*** *Compute the derivatives of the loss function $\mathcal{L}$ relative to each parameter:*

$$\frac{\partial \mathcal{L}}{\partial \phi} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \phi_0} \\ \frac{\partial \mathcal{L}}{\partial \phi_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \phi_N} \end{bmatrix}$$

  *This gradient indicates the uphill direction of the loss function.*

- ***Step 2:*** *Update parameters using:*

$$\phi \leftarrow \phi - \alpha \frac{\partial \mathcal{L}}{\partial \phi},$$

  *where $\alpha$ is the learning rate.*

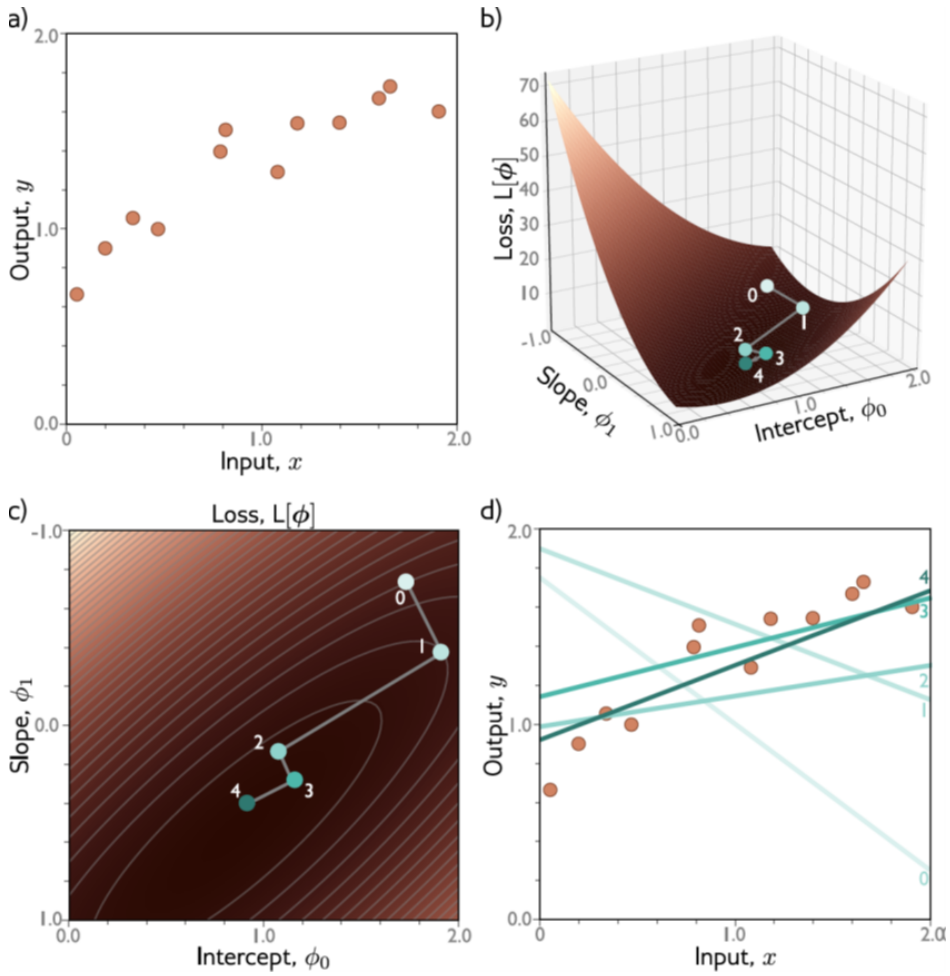



Figure 8.29: Weight training workflow with parameter updates.

**Linear Regression Example**

***Loss Function:***

$$\mathcal{L}[\phi] = \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \big(f(x_i, \phi) - y_i\big)^2,$$

*where* $f(x, \phi) = \phi_0 + \phi_1 x$.

***Gradients:***

$$\frac{\partial \mathcal{L}}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}.$$

**Gabor Model Example**

*In this example, the model function is:*

$$f(x,\phi) = \sin[\phi_0 + 0.06 \cdot \phi_1 x] \cdot \exp\left(-\frac{(\phi_0 + 0.06 \cdot \phi_1 x)^2}{8.0}\right).$$

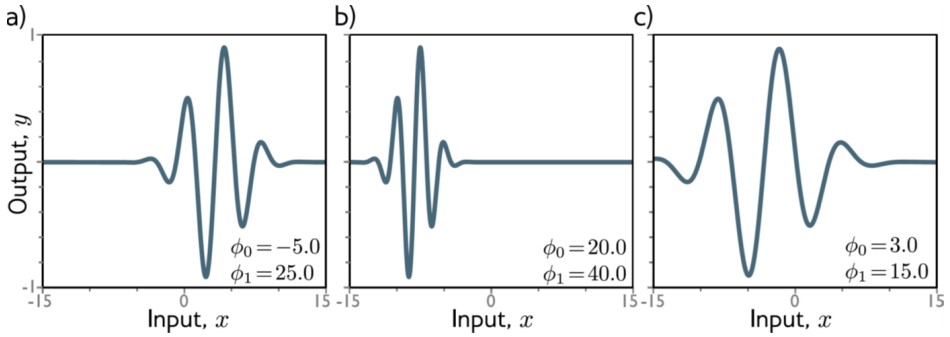



Figure 8.30: abor model training example with input-output data points and specific instance function.

### 8.3.5 Loss Function across different parameters

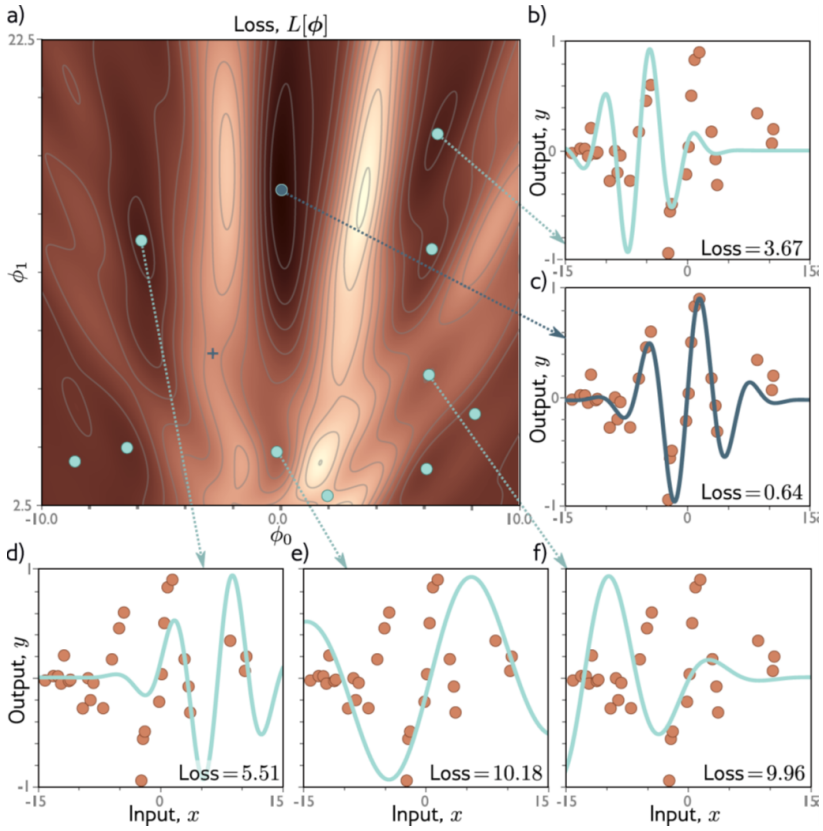



Figure 8.31: Visualization of the loss function $L[\phi]$ across different parameter values.

***Explanation:***

*The figure illustrates the relationship between the parameters $\phi_0$ and $\phi_1$, the corresponding loss function $L[\phi]$, and the model's performance on the data:*

- ***(a) Contour Plot:***
  *The contour plot depicts the loss landscape over the parameters $\phi_0$ and $\phi_1$. The darker regions represent lower loss values, indicating better parameter configurations. The points on the contour correspond to specific parameter settings visualized in subplots (b-f).*

- ***(b-f) Model Predictions:***
  *Each subplot shows the model's output $y$ (blue curve) compared to the true data points (orange dots), demonstrating the effects of different parameter settings:*

- ***(b): Loss*** $= 3.67$
  *The model exhibits overfitting, capturing noise in the data rather than the underlying trend.*
- ***(c): Loss*** $= 0.64$
  *Near-optimal parameters result in a good fit to the data, with minimal deviation between predictions and true values.*
- ***(d): Loss*** $= 5.51$
  *Underfitting behavior is evident, as the model fails to capture key patterns in the data.*
- ***(e): Loss*** $= 10.18$
  *The parameters lead to a poor fit, with significant deviation from the true data, resulting in high loss.*
- ***(f): Loss*** $= 9.96$
  *The model exhibits a mix of overfitting in some regions and underfitting in others, leading to suboptimal performance.*

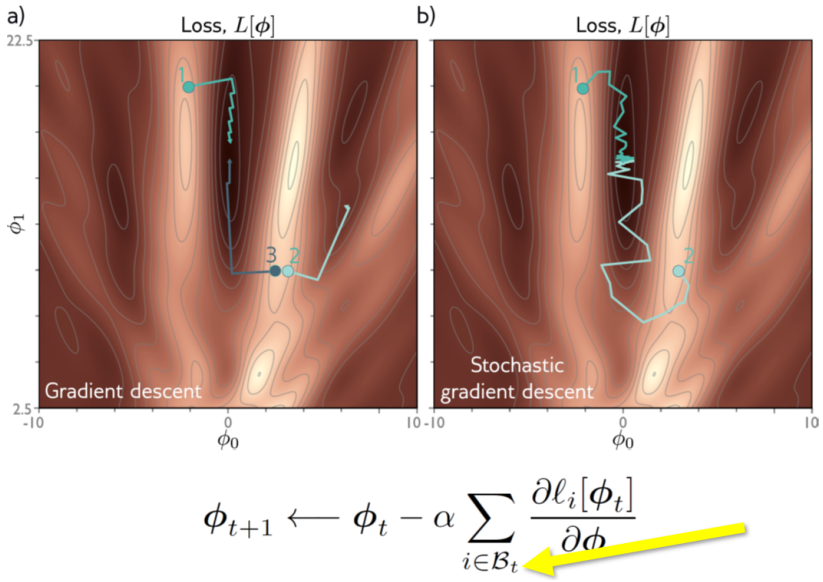



Figure 8.32: Comparison of Gradient Descent and Stochastic Gradient Descent methods over the loss landscape.

*Gradient-based optimization methods are illustrated in this figure:*

- ***Panel (a): Gradient Descent (GD)***
  *In this approach, the parameters $\phi_t$ are updated using the gradient of the loss function $L[\phi]$ with*

*respect to all data points. The update rule is:*

$$\phi_{t+1} \leftarrow \phi_t - \alpha \nabla L[\phi_t],$$

*where $\alpha$ is the learning rate. The trajectory shown follows a smooth path toward the global minimum due to the aggregated gradients computed over the entire dataset.*

- ***Panel (b): Stochastic Gradient Descent (SGD)***
  *In stochastic gradient descent, the parameters are updated using gradients calculated for a random subset (or mini-batch) of the data. The update rule is:*

$$\phi_{t+1} \leftarrow \phi_t - \alpha \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi},$$

  *where $\mathcal{B}_t$ represents the mini-batch of data points at iteration $t$. This method introduces noise into the updates, causing the trajectory to fluctuate as it approaches the minimum.*

- ***Comparison***
  *- Gradient Descent (a) shows a smooth and consistent path but requires the computation of gradients over the entire dataset, making it computationally expensive for large datasets. - Stochastic Gradient Descent (b) converges faster in practice as it uses only a subset of data per iteration, but the trajectory exhibits more variability due to the stochasticity in mini-batch selection.*

### 8.3.6 Improving Gradient Descent

- ***Stochastic Gradient Descent (SGD):*** *Updates parameters using a subset (mini-batch) of the data to reduce computation.*

- ***Momentum:*** *Adds a fraction of the previous update to the current one:*

$$v_t = \gamma v_{t-1} + \eta \nabla \mathcal{L}(\theta), \quad \theta \leftarrow \theta - v_t,$$

  *where $\gamma$ is the momentum coefficient.*

- ***Normalized Gradients:*** *Ensures gradients have unit norm to stabilize updates:*

$$\nabla \mathcal{L} \leftarrow \frac{\nabla \mathcal{L}}{\|\nabla \mathcal{L}\|}.$$

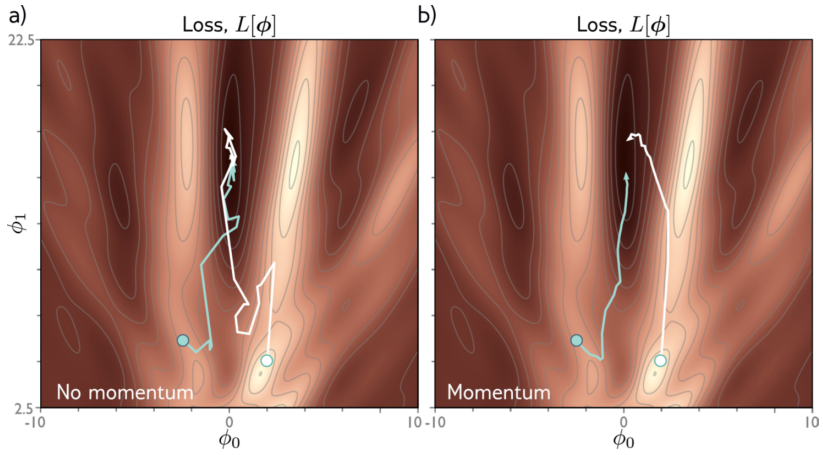



Figure 8.33: Comparison of optimization paths with and without momentum.

***Explanation:***

*This figure compares optimization trajectories for gradient descent with and without momentum in a loss landscape defined by parameters $\phi_0$ and $\phi_1$.*

- ***(a) No Momentum:***
  *The trajectory demonstrates erratic movement across the loss landscape, particularly in regions with high curvature. The optimizer frequently oscillates between directions, leading to inefficient convergence and a longer time to reach the minimum.*
- ***(b) With Momentum:***
  *The addition of momentum smooths the optimization path by incorporating velocity from previous updates. This results in a more direct trajectory toward the minimum, reducing oscillations and accelerating convergence, especially in curved regions.*

***Key Observation:*** *Momentum improves the efficiency of gradient descent by dampening oscillations and guiding the optimizer along more stable paths through the parameter space.*

# Chapter 9

# Bias vs Variance, Regularization, and CNNs

## 9.1 Measuring Network Performance

### 9.1.1 Network Performance and Test Errors

**Definition 9.1.1** (Training and Test Error)**.**

- **Training Error:** The error calculated on the same dataset used to train the model.
- **Test Error:** The error calculated on a separate dataset to evaluate generalization ability.

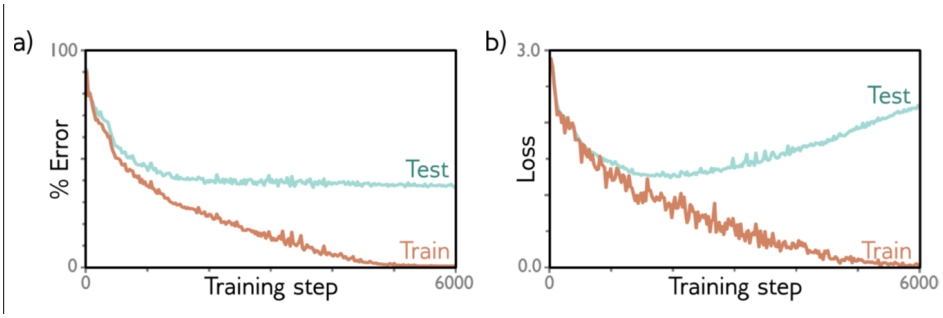



Figure 9.1: Experiment testing separate MNIST-1D datasets: I=4000 Training examples vs 1000 Testing examples. Network has 40D input, 2 layers of 100x hidden units, 10D output passed through softmax to return y is in {0..9}.

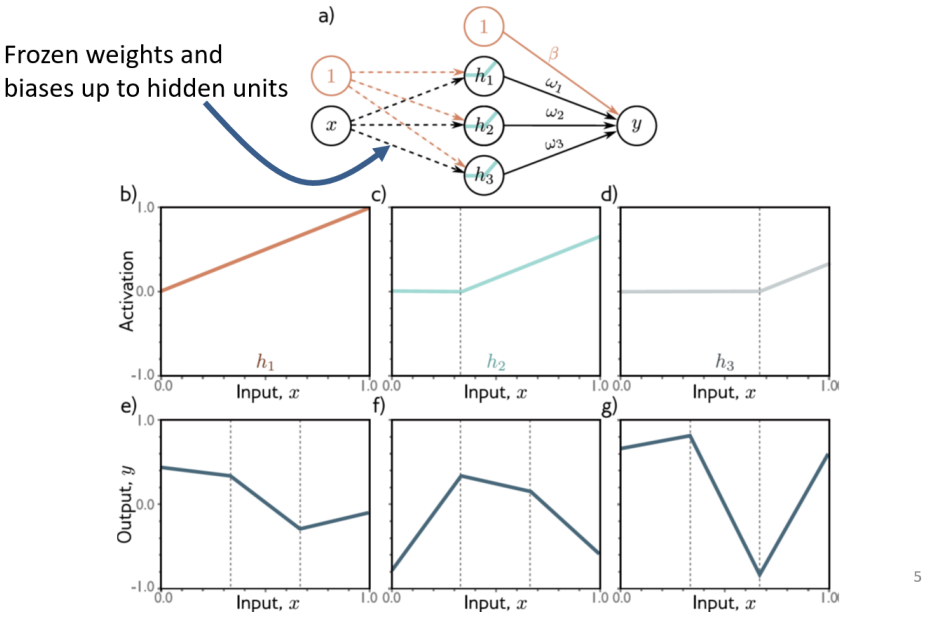



Figure 9.2: Test Errors: Linear Regression Example with Least Square Loss.

### 9.1.2 Reasons of Increasing Test Errors

- ***Data Mismatch:*** *Test data may originate from a different distribution (e.g., synthetic rendering vs. real-world data).*
- ***Input Gaps:*** *The test dataset may leave gaps in the spectrum of possible inputs, failing to represent the full domain.*
- ***Bias VS Variance Tradeoff:***
  - ***Noise:*** *Identical inputs can lead to different outputs due to randomness in the data.*
  - ***Bias:*** *When the true underlying function is more complex than the model can represent.*
  - ***Variance:*** *Sensitivity to fluctuations in the training dataset.*

### 9.1.3 Linear Regression: Test Error Analysis with Least Square Loss

**Definition 9.1.2** (Expected Loss)**.** Given the true function $f(x)$ and noise $\sigma^2$, the expected loss for linear regression is:

$$\mathbb{E}[\text{Loss}] = \mathbb{E}[(y - \hat{y})^2],$$

accounting for both training and test data uncertainty.

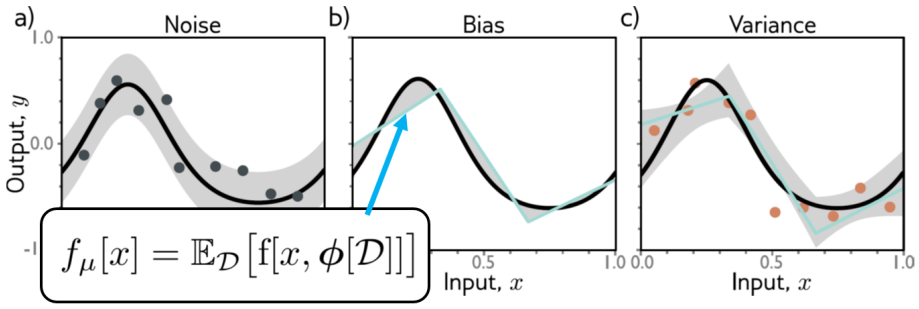



Figure 9.3: Comparsion of Noise, Bias, and Variance. **Noise** means two identical x's could give different outputs. **Bias** means the underlying function is more complex than our model can handle. **Variance** means the model is only as good as the available training samples.

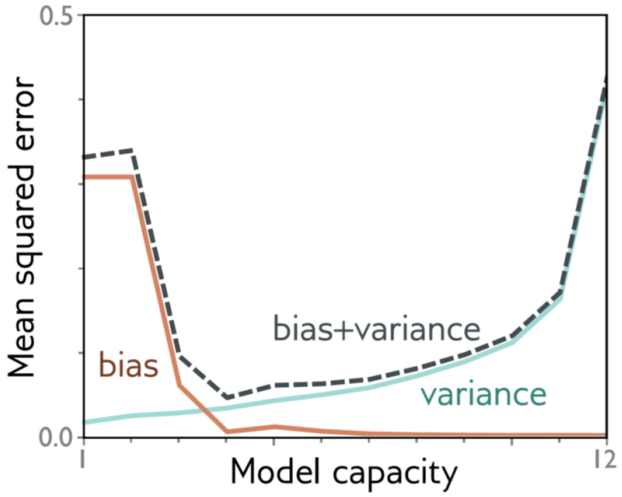



Figure 9.4: Variance vs Bias "Sweet Spot"

**Theorem 9.1.3** (Bias-Variance Tradeoff)**.** *In a toy problem where we have privileged access to the true function $f[x]$ and noise $\sigma^2$, the expected loss can be expressed as:*

$$\mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_y[L[x]]\right] = \mathbb{E}_{\mathcal{D}}\left[\left(f[x, \phi[\mathcal{D}]] - f_\mu[x]\right)^2\right] + \left(f_\mu[x] - \mu[x]\right)^2 + \sigma^2.$$

- *The first term, $\mathbb{E}_{\mathcal{D}}\left[\left(f[x, \phi[\mathcal{D}]] - f_\mu[x]\right)^2\right]$, represents the **variance**, capturing the sensitivity of the model to fluctuations in the training dataset $\mathcal{D}$.*

- *The second term, $\big(f_\mu[x] - \mu[x]\big)^2$, represents the **bias**, indicating how far the average prediction deviates from the true function $\mu[x]$.*

- *The third term, $\sigma^2$, accounts for the noise inherent in the data.*

*Here:*

$$f_\mu[x] = \mathbb{E}_{\mathcal{D}}[f[x, \phi[\mathcal{D}]]], \quad \mu[x] = \mathbb{E}_y[y[x]].$$

### 9.1.4 Reducing Bias and Variance

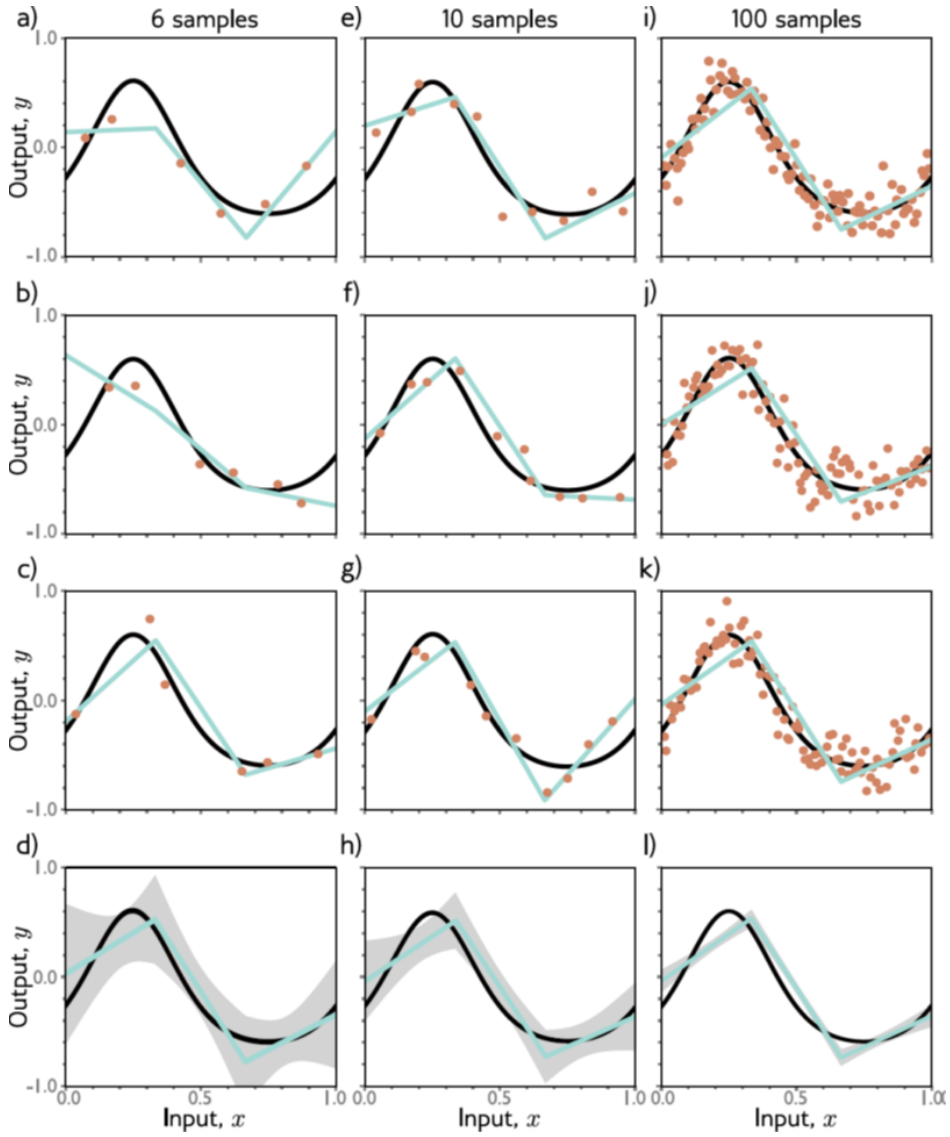



Figure 9.5: **Reducing Variance**: Comparison of how training sample size affects the variance. The shaded area represents the variance.

- ***Reducing Variance:***
  - *Increase training data size.*

– *Use regularization techniques (e.g., L2 regularization).*

– *Reduce model complexity.*

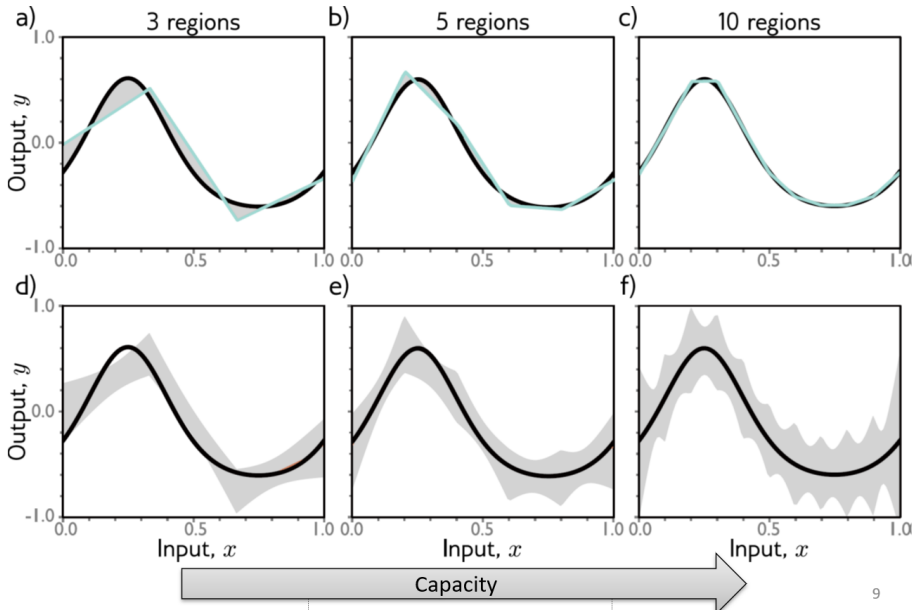



Figure 9.6: **Reducing Bias**: increase the number of hidden units/nodes, or add more layers, will reduce bias.

- ***Reducing Bias:***

  – *Increase model capacity (e.g., number of hidden units or layers), which impacts the trade-off between bias and variance:*

    * *Low capacity: High bias, low variance.*
    * *High capacity: Low bias, high variance*

  – *Use a more complex model architecture.*

### 9.1.5 Overfitting

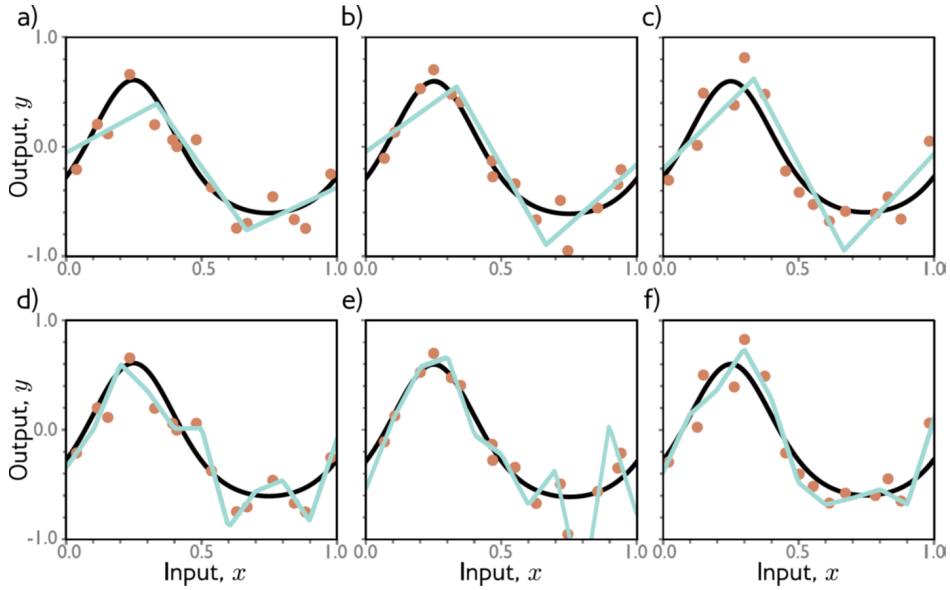



Figure 9.7: An overcomplicated model could leads to overfitting (regularization).

### 9.1.6 Double Descent Phenomenon

**Remark.** *Training error can exhibit a* ***double descent*** *phenomenon:*

- *Initial overfitting is followed by improved performance as model capacity increases beyond a certain point.*

## 9.2 Convolutional Neural Network

### 9.2.1 Images and Fully-connected Nets

*Convolutional Neural Networks (CNNs) address the inefficiency of fully-connected layers for image data by leveraging local spatial correlations. For an input image of size* $224 \times 224$*, processing with a fully-connected network would require* $224 \times 224 \times n_{hidden}$ *parameters, where* $n_{hidden}$ *is the number of hidden units. Nearby pixels are often statistically related, allowing CNNs to reduce parameters using localized operations.*

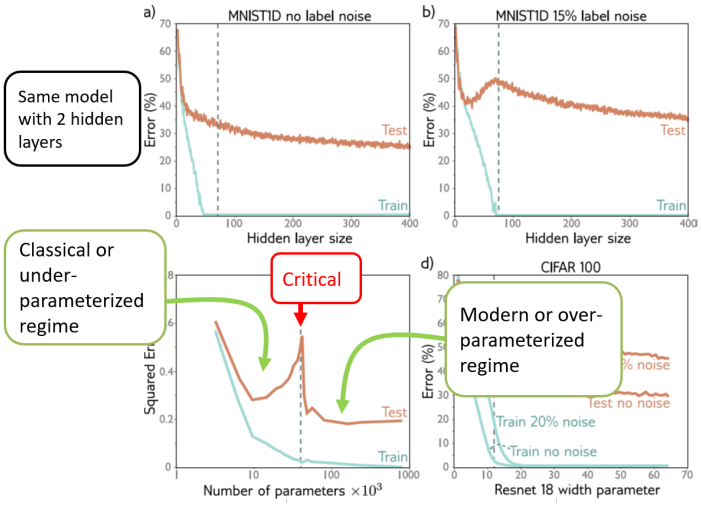



Figure 9.8: Double Descent Phenomenon.

### 9.2.2 Invariance and Equivariance

**Invariance:** *Let $f(x)$ be a function representing a classifier. If $t(x)$ is a transformed input (e.g., a translation), invariance implies $f(x) = f(t(x))$.*

**Equivariance:** *For a transformation $t$ and function $g$, equivariance implies $g(t(x)) = t(g(x))$. For example, if $t$ translates the input by 20 pixels, the feature map is equivalently translated.*

### 9.2.3 1D Convolution

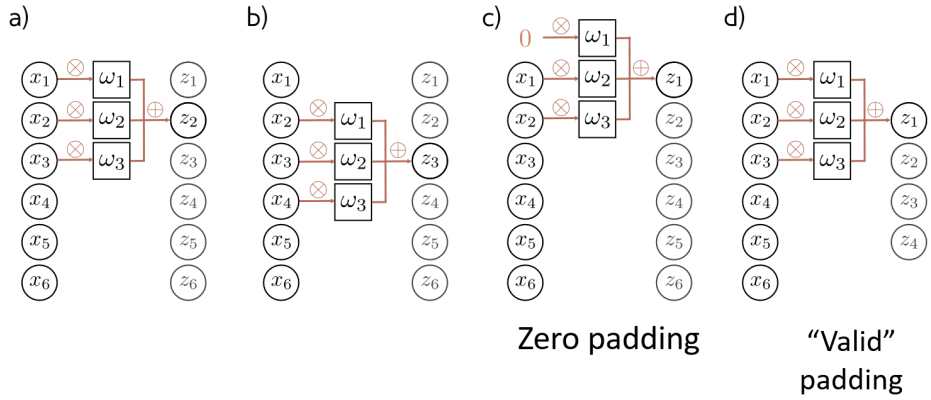



Figure 9.9: 1-D Convolution.

*Given an input $x[n]$ and a filter $h[k]$ of size $K$, the convolution operation is defined as:*

$$z_i = \sum_{k=0}^{K-1} w_k x_{i+k},$$

*where $z_i$ represents the output at position $i$ after applying the filter.*

**Zero Padding:** *To preserve the input size, zero padding adds $p$ zeros to both ends of the input sequence. The output length $L_{out}$ after applying the convolution is given by:*

$$L_{out} = L_{in} + 2p - K + 1.$$

***Example:***

- *For $p = 1$, the input sequence becomes $[0, x_1, x_2, \ldots, x_n, 0]$.*
- *The filter then slides over this padded sequence, producing an output of the same length as the original input.*

**"Valid" Padding:** *In contrast to zero padding, valid padding does not add any extra elements to the input sequence, resulting in an output length:*

$$L_{out} = L_{in} - K + 1.$$

*This minimizes the output size by excluding regions where the filter would exceed the bounds of the input.*

**Stride:** *The stride $s$ determines the step size by which the filter moves across the input. The output length $L_{out}$ for stride $s$ is given by:*

$$L_{out} = \left\lfloor \frac{L_{in} + 2p - K}{s} + 1 \right\rfloor.$$

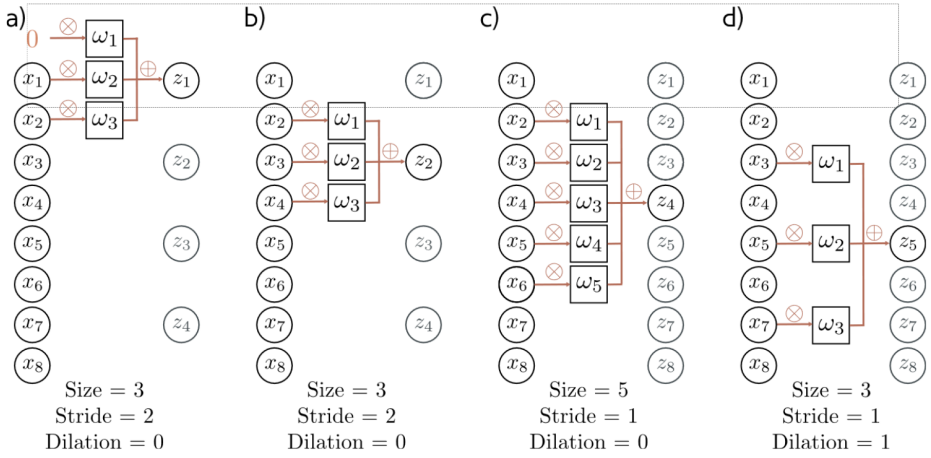



Figure 9.10: Stride, Kernel Size, Dilation.

***Example:***

- *For $s = 2$, the filter skips every alternate position, effectively halving the spatial resolution of the output.*

**Kernel Size and Dilation:** *The kernel size $K$ defines the window of input covered by the filter. Dilation $d$ modifies the spacing between elements within the kernel, effectively enlarging the receptive field without increasing the number of parameters. The effective kernel size becomes:*

$$K_{effective} = K + (K - 1)(d - 1).$$

***Example:***

- *For $K = 3$ and $d = 2$, the filter spans 5 input positions: $x[i], x[i + 2], x[i + 4]$.*

**Illustrations:** *The following operations demonstrate key concepts:*

*[label=()]Simple convolution with no padding, stride = 1, and kernel size = 3. Zero padding to preserve input size. Stride = 2 reduces the spatial resolution of the output. Dilation expands the receptive field without increasing the kernel size.*

### 9.2.4 Fully-Connected vs Convolutional Layers

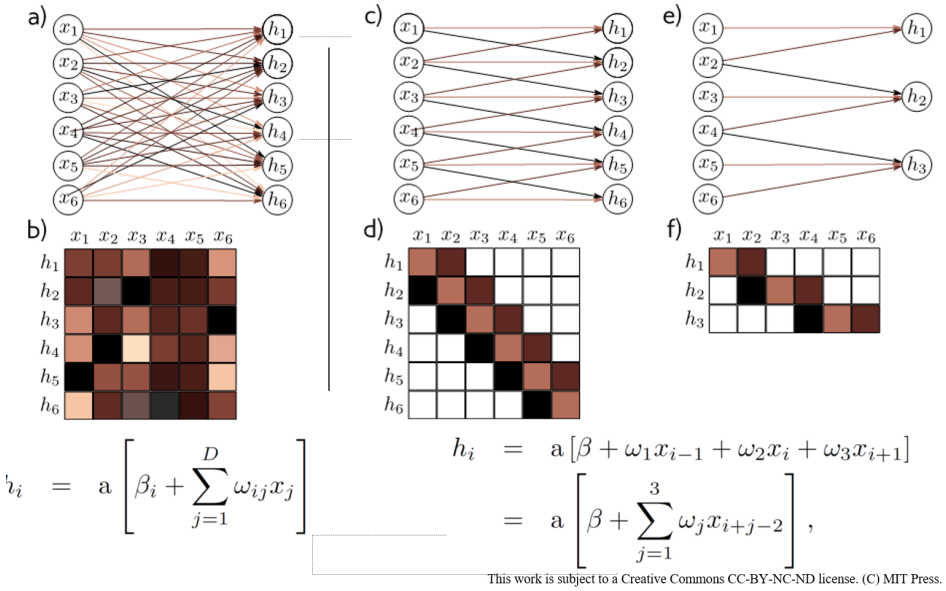




Figure 9.11: Fully Connected vs. Convolutional Layers Illustration.

**Definition 9.2.1** (Fully-Connected Layers)**.** A fully-connected layer connects every input to every output using distinct weights. For a layer with $L_{\text{in}}$ input units and $L_{\text{out}}$ output units, the number of parameters is given by:

$$\text{Parameters} = L_{\text{in}} \times L_{\text{out}}.$$

**Definition 9.2.2** (Convolutional Layers)**.** Convolutional layers utilize sparse connectivity and shared weights. For a convolutional layer with $C_i$ input channels, $C_o$ output channels, and a kernel of size $K \times K$, the number of parameters is:

$$\text{Parameters} = K^2 \times C_i \times C_o.$$

**Theorem 9.2.3** (Parameter Efficiency of Convolutional Layers)**.** *Convolutional layers require fewer parameters than fully-connected layers for the same input size, making them more efficient for high-dimensional data such as images.*

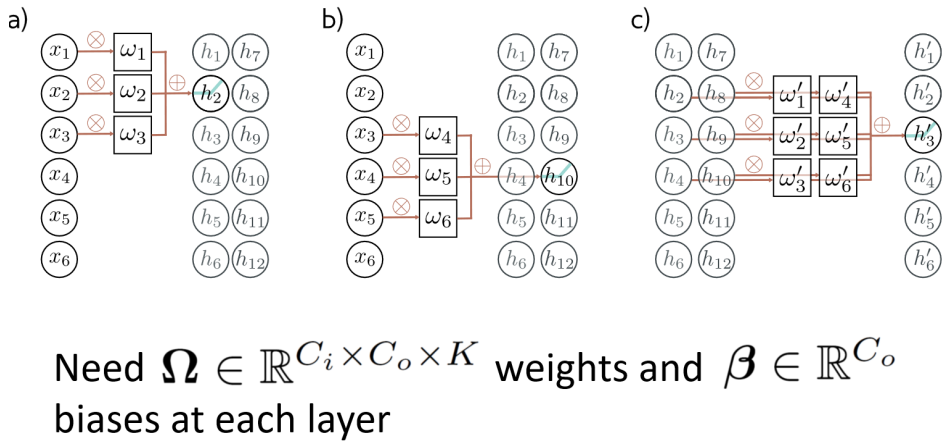



Figure 9.12: Multiple Channels.

**Remark.** *Fully-connected layers are computationally expensive because the parameters grow quadratically with the input size, whereas convolutional layers grow linearly with the kernel size and number of channels.*

**Example 9.2.4** (Comparison of Parameter Counts)**.** Consider a 1D input with 6 elements processed by a fully-connected layer with 6 hidden units. The parameter count is $6 \times 6 = 36$. In contrast, for a convolutional layer with a kernel size of 3, stride 1, and 1 filter ($C_i = C_o = 1$), the parameter count is $3 \times 1 \times 1 = 3$.

### 9.2.5 Receptive Field

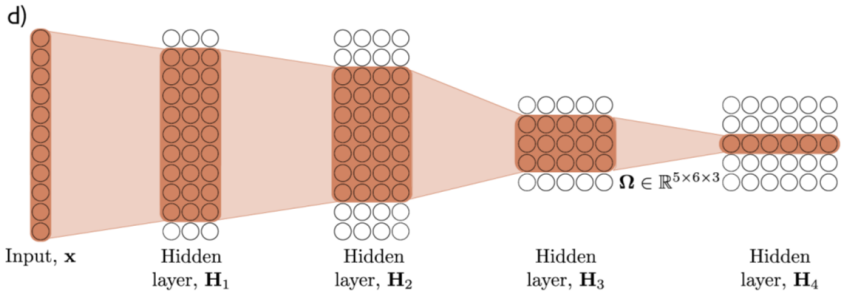



Figure 9.13: Receptive Field Example.

**Definition 9.2.5** (Receptive Field)**.** The receptive field of a neuron in a CNN is the region of the input that influences its output. The size of the receptive field grows with the depth of the network.

**Theorem 9.2.6** (Receptive Field Growth)**.** *For a convolutional layer $l$ with kernel size $K$, stride $s$, and dilation $d$, the receptive field $R_l$ is given recursively as:*

$$R_l = s \cdot R_{l-1} + d \cdot (K - 1),$$

*where $R_{l-1}$ is the receptive field of the previous layer.*

**Example 9.2.7** (Receptive Field Calculation)**.**

- **Single Layer:** For a layer with $K = 3$, $s = 1$, and $d = 1$, the receptive field is $3$.
- **Two Layers:** Stacking two layers with the same parameters expands the receptive field to $5$ due to overlapping kernel operations.
- **Increased Stride:** If $s = 2$, the receptive field grows faster. For $K = 3$, $d = 2$, and $s = 2$, the effective receptive field becomes $5$.

**Remark.** *Larger receptive fields in deeper layers allow CNNs to aggregate local patterns into global representations, such as shapes or objects.*

### 9.2.6 Case Study: Inductive Bias in 1D Convolutional Networks

**Definition 9.2.8** (Inductive Bias)**.** Inductive bias refers to the assumptions a model makes about the data. Convolutional networks have a strong inductive bias due to locality and weight sharing, enabling them to generalize better for structured data like images.

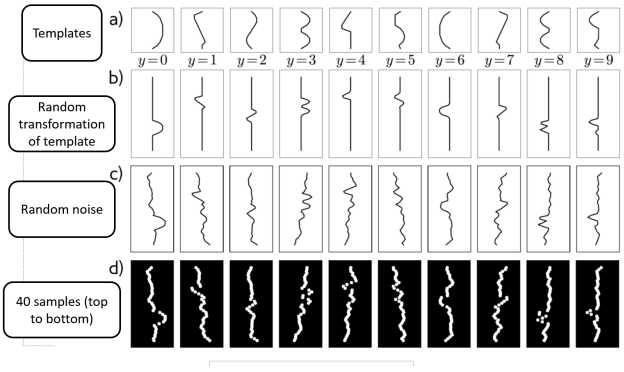



Figure 9.14: 1-D Example: MNIST1D

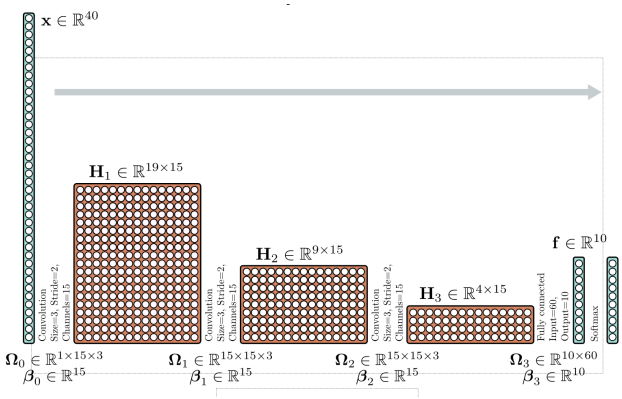



Figure 9.15: 1-D Example: MNIST1D Model Architecture.

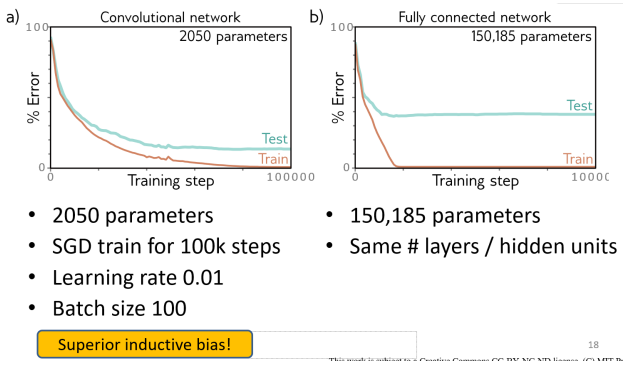



Figure 9.16: 1-D Example: MNIST1D CNN vs. Fully Connected.

**Example 9.2.9** (1D Convolutional Network vs Fully-Connected Network)**.** A 1D convolutional network with 2,050 parameters trained on MNIST1D achieves better generalization than a fully-connected network with 150,185 parameters.

- **Convolutional Network:** Optimized using SGD for 100,000 steps, learning rate $0.01$, batch size $100$.

- **Fully-Connected Network:** Despite the same number of layers and hidden units, it overfits due to the lack of spatial inductive bias. Convolutional networks exploit spatial structure, achieving lower test error and better generalization compared to fully-connected networks.

### 9.2.7 2D Convolutional Layer

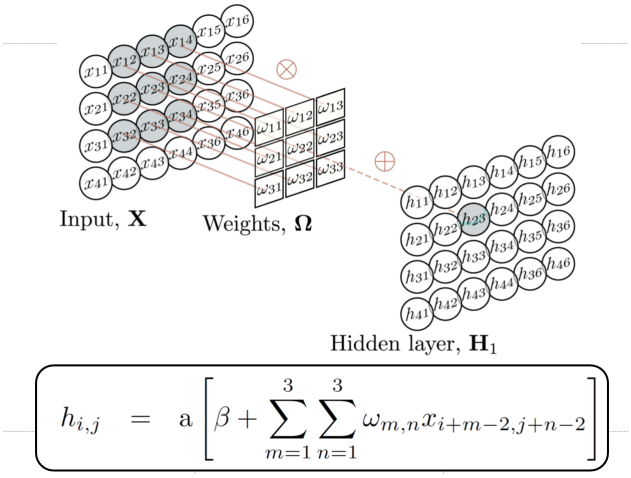



Figure 9.17: 2D Convolutional Layer.

*The 2D convolutional layer operates on 2D inputs, such as images. For an input* $X \in \mathbb{R}^{H\times W\times C_i}$ *and a filter* $W \in \mathbb{R}^{K\times K\times C_i\times C_o}$*, the output* $Y \in \mathbb{R}^{H_{out}\times W_{out}\times C_o}$ *is computed as:*

$$Y[i,j,c_o] = \sum_{c_i=1}^{C_i}\sum_{k=0}^{K-1}\sum_{l=0}^{K-1} W[k,l,c_i,c_o]\cdot X[i+k,j+l,c_i].$$

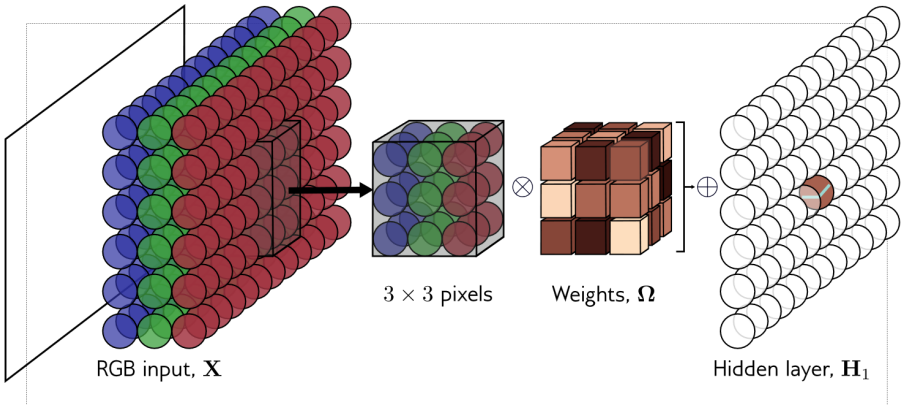



Figure 9.18: 2D Convolution of Image.

**Definition 9.2.10** (Parameters in 2D Convolution)**.** For a kernel size $K \times K$, $C_i$ input channels, and $C_o$ output channels, the number of parameters is:

$$\text{Parameters} = K \times K \times C_i \times C_o + C_o \text{ (biases)}.$$

### 1x1 Convolution

- *A $1 \times 1$ convolution applies weights of size $1 \times 1 \times C_i \times C_o$ to each pixel independently across channels.*
- *Total parameters: $C_i \times C_o$ weights plus $C_o$ biases.*
- *Useful for merging CNN blocks or reducing the number of channels in feature maps.*

### Downscaling an Image

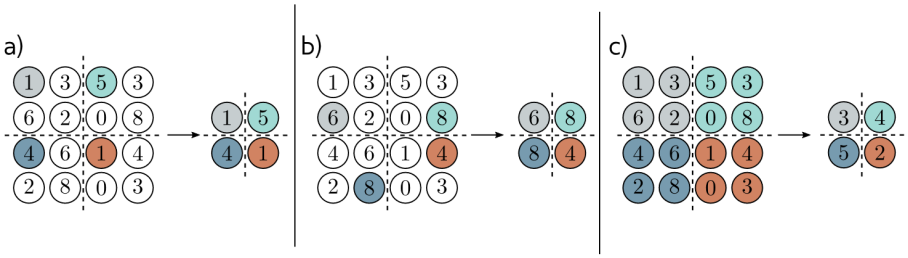



Figure 9.19: Downscaling an Image.

*Downscaling reduces the spatial dimensions of the input, typically by a factor of 2. The methods include:*

1. *Sampling every other pixel using a stride of 2.*
2. *Max-pooling: Retains the maximum value in a given pooling window.*
3. *Average pooling: Computes the average value in a pooling window.*

### Upscaling an Image

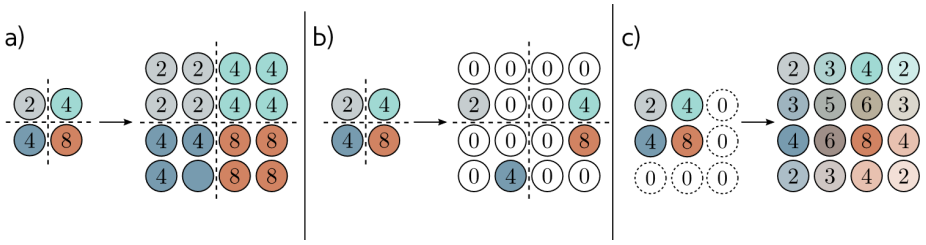



Figure 9.20: Upscaling an Image.

*Upscaling increases the spatial dimensions of the input, often by a factor of 2. The methods include:*

1. ***Duplicate:*** *Each pixel is replicated spatially to increase dimensions.*
2. ***Max Unpooling:*** *Restores spatial information using indices from prior max-pooling layers.*
3. ***Bilinear Interpolation:*** *Generates smoother transitions by interpolating between neighboring pixel values.*
4. ***Transposed Convolution:*** *Also known as deconvolution, it reverses the effect of standard convolution to upsample the input.*

### Transposed Convolution

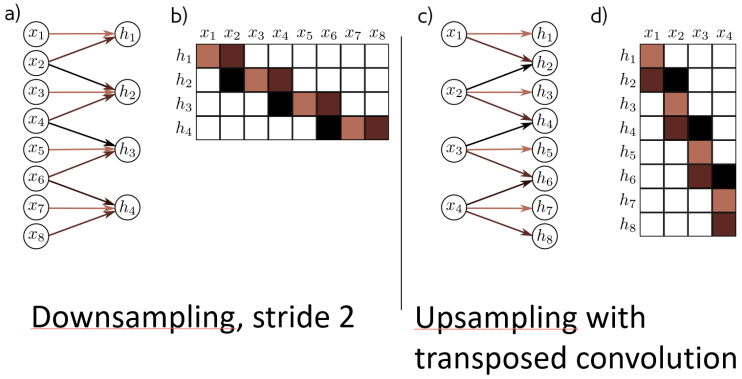



Figure 9.21: Transposed Convolution.

- *Transposed convolution performs upsampling while learning patterns during training.*

- *The spatial dimensions of the output are given by:*

$$\text{Output Shape} = (H_{in} - 1) \times s + K - 2p,$$

  *where $s$ is the stride, $K$ is the kernel size, and $p$ is the padding.*

- *Useful in tasks like image generation and semantic segmentation.*

**Remark.** *Transposed convolutions are particularly valuable in reconstructing high-resolution outputs from lower-resolution feature maps.*

### Changing the Number of Channels

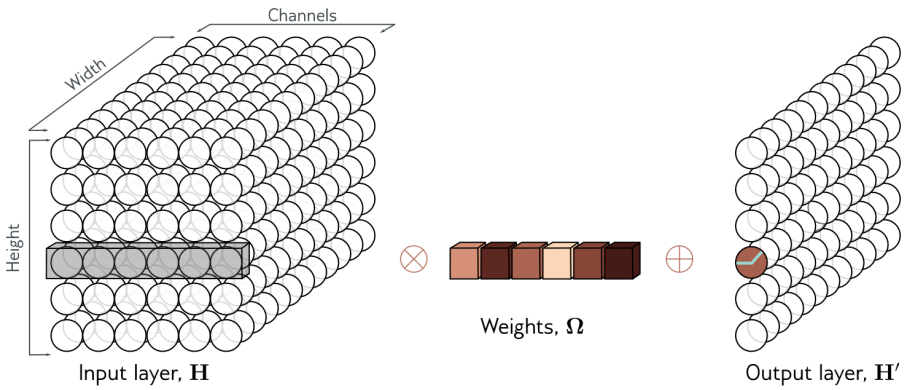



Figure 9.22: Changing the Number of Channels.

*To adjust the number of channels in a feature map, a $1 \times 1$ convolution is applied. This operation:*

- *Has $1 \times 1 \times C_i \times C_o$ weights and $C_o$ biases.*

- *Is particularly useful when merging CNN blocks or altering feature map dimensions without modifying spatial resolution.*

**Example 9.2.11** (Channel Adjustment with $1 \times 1$ Convolution)**.** Consider an input with $C_i = 64$ channels and $C_o = 32$. A $1 \times 1$ convolution reduces the output channels to 32, preserving the spatial dimensions of the feature map.

### 9.2.8 Applications

#### ImageNet Classification

**AlexNet:** *Utilized $K = 11, 5, 3$ kernels in successive layers with stride and padding for downscaling. Its architecture included:*

$$\begin{aligned}&\textit{Conv1: } 96 \textit{ filters, } K = 11, s = 4, p = 2,\\ &\textit{Conv2: } 256 \textit{ filters, } K = 5, s = 1, p = 2,\\ &\textit{Conv3–5: } 384, 384, 256 \textit{ filters, } K = 3, s = 1, p = 1.\end{aligned}$$

*Training employed SGD with momentum (0.9), dropout, and L2 regularization, achieving a top-5 error of 16.4%.*

**VGGNet:** *Increased depth to 19 layers using smaller kernels ($K = 3$) with $p = 1$, achieving a top-5 error of 6.8%.*

### 9.2.9 Key Takeaways

- ***Convolutional vs Fully-Connected Layers:***
  - ***Sparser Connections:*** *Convolutional layers utilize sparse connections, reducing the number of parameters compared to fully-connected layers.*
  - ***Shared Weights:*** *The same filter is applied across the entire input, ensuring translational equivariance and reducing computational cost.*
  - ***Parallel Processing Paths:*** *Convolutional layers enable efficient computation by leveraging parallel processing for feature extraction.*
- ***CNNs Outperform MLPs on Images:***
  - *Demonstrated through live comparisons, CNNs achieve significantly better performance than Multi-Layer Perceptrons (MLPs) on image tasks.*
- ***Data Augmentation is Crucial:***
  - *CNNs effectively handle **translational equivariance**, ensuring robustness to shifts in input data.*
  - *Achieving **rotational equivariance** is more challenging and often requires additional architectural modifications or preprocessing.*
- ***Architectural Choices Matter:***
  - *The design of CNN architectures, including the number of layers, kernel sizes, and activation functions, significantly impacts their performance.*

## 9.3 Examples: Regularization, Learned Embeddings, 3D Scan Completion

### 9.3.1 Regularization: Ladder Networks

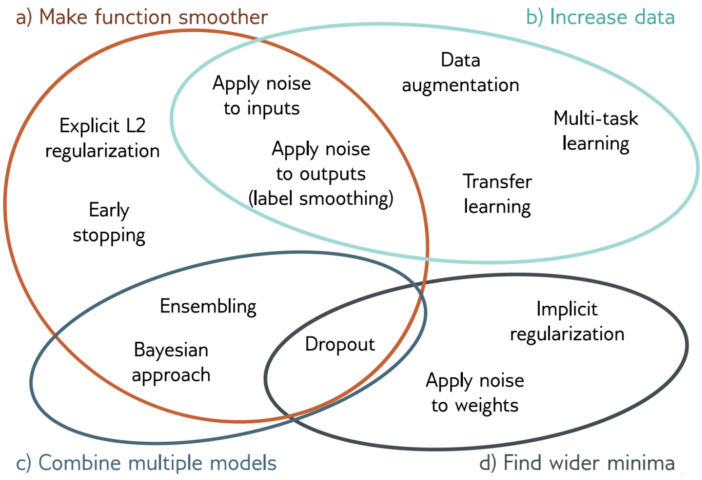



Figure 9.23: Overview of Regularization.

**Definition 9.3.1** (Regularization)**.** Regularization refers to techniques that improve generalization by preventing overfitting. In neural networks, regularization is critical for high-dimensional datasets where over-parameterization can lead to memorization of training data.

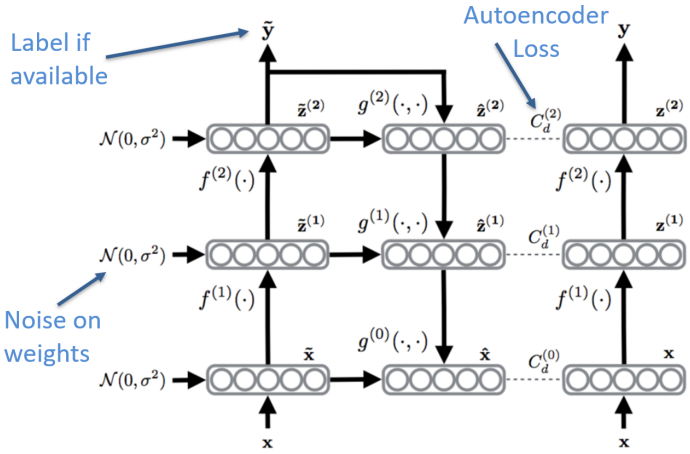



Figure 9.24: Ladder Network: Example Semi-Supervised Network.

**Ladder Networks (Rasmus et al., 2015):** *Ladder networks integrate supervised and unsupervised learning by combining:*

1. ***Supervised Loss:*** *Applied to labeled data to minimize classification error:*

$$\mathcal{L}_{sup} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_i^{(c)} \log(\hat{y}_i^{(c)}),$$

   *where $y_i^{(c)}$ is the true label and $\hat{y}_i^{(c)}$ is the predicted probability for class $c$.*

2. ***Unsupervised Loss:*** *Enforces consistency between corrupted ($z_{corr}$) and clean ($z_{clean}$) latent representations through an autoencoder reconstruction objective:*

$$\mathcal{L}_{unsup} = \sum_{l} \|z_{corr}^{(l)} - z_{clean}^{(l)}\|^2.$$

*The total loss combines both objectives:*

$$\mathcal{L} = \mathcal{L}_{sup} + \lambda \mathcal{L}_{unsup},$$

*where $\lambda$ balances the contributions of supervised and unsupervised components.*

**Key Properties:**

- ***Noise Injection:*** *Adds noise to the input and intermediate layers during training to improve robustness.*

- ***Consistency Loss:*** *Forces the model to learn robust representations by aligning noisy and clean representations.*

- ***Feature Hierarchies:*** *Ladder networks enforce reconstruction at every layer, encouraging useful feature hierarchies for downstream tasks.*

—

### 9.3.2 Learned Embeddings: Box Embeddings

**Definition 9.3.2** (Box Embeddings)**.** Box embeddings represent data points as $d$-dimensional hyperrectangles (boxes) in a latent space. The relationship between two boxes (e.g., overlap or containment) encodes semantic relationships between data points.

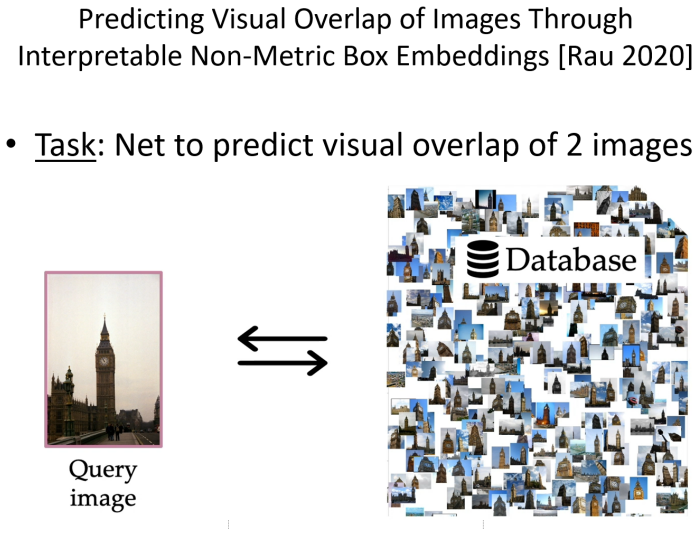



Figure 9.25: pREDI

**Predicting Visual Overlap (Rau et al., 2020):** *Box embeddings are used to predict the normalized surface overlap (NSO) between two images, defined as:*

$$NSO(x, y) = \frac{Area(Overlap(x, y))}{Area(Union(x, y))}.$$

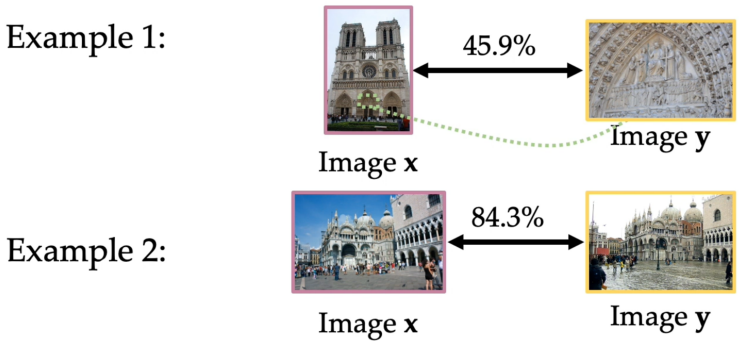



Figure 9.27: NSO Examples (Averging both NSO(x,y) and NSO(y,x) tells us something?)

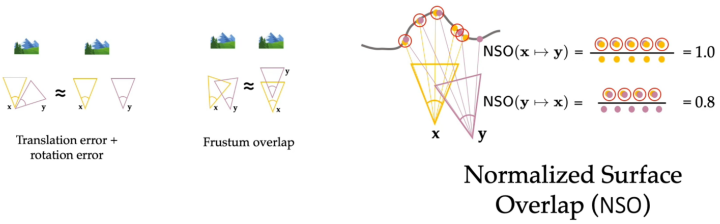



Figure 9.26: Normalized Surface Overlap.

*The NSO metric assumes access to depth information, making it sensitive to perspective changes. To handle asymmetry, the NSO is averaged:*

$$NSO_{avg}(x, y) = \frac{NSO(x, y) + NSO(y, x)}{2}.$$

**Model Architecture:**

- ***Input:*** *Image pairs.*
- ***Structure:*** *Five residual convolutional layers followed by average pooling and two fully connected layers.*
- ***Output:*** *32-dimensional box embeddings.*

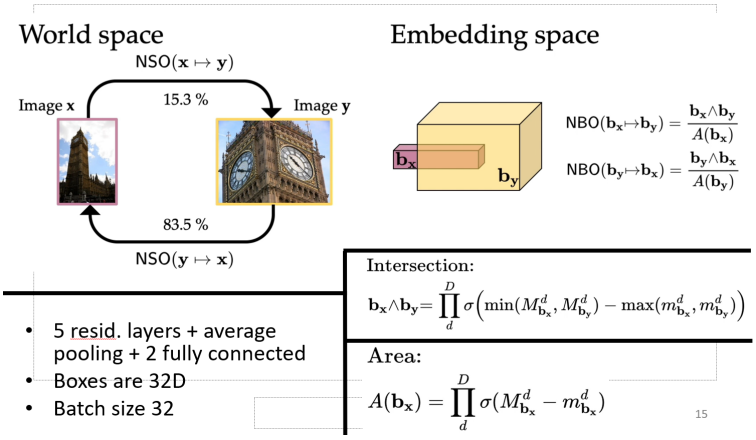



Figure 9.28: NSO Model Architecture.

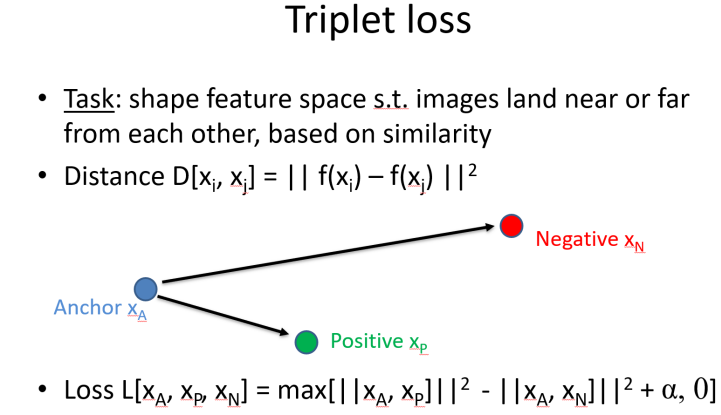



Figure 9.29: Triplet Loss.

**Training Objective:** *The embeddings are optimized using a triplet loss to preserve relationships between anchor ($x_A$), positive ($x_P$), and negative ($x_N$) samples:*

$$\mathcal{L} = \max\left(0, \|x_A - x_P\|^2 - \|x_A - x_N\|^2 + \alpha\right),$$

*where $\alpha$ is the margin ensuring separation between positive and negative samples.*

**Remark.** *Box embeddings provide an interpretable way to model pairwise relationships, leveraging geometric overlap to measure similarity.*

—

### 9.3.3 3D Scan Completion: Heightfields

**Definition 9.3.3** (Heightfields)**.** Heightfields represent 3D geometry as 2D top-down views, where each pixel encodes height information. These representations simplify scene reconstruction by reducing the dimensionality of 3D data.

**Depth Prediction:** *Depth prediction involves estimating the depth map* $D \in \mathbb{R}^{H \times W}$ *from a single RGB image* $I \in \mathbb{R}^{H \times W \times 3}$*. The model minimizes the disparity loss between the predicted and ground truth depth:*

$$\mathcal{L}_{depth} = \|D_{pred} - D_{gt}\|^2.$$

**Heightfields for Scene Reconstruction (Watson, 2023):** *Heightfields aggregate depth maps from multiple viewpoints to create dense 3D reconstructions. The workflow includes:*

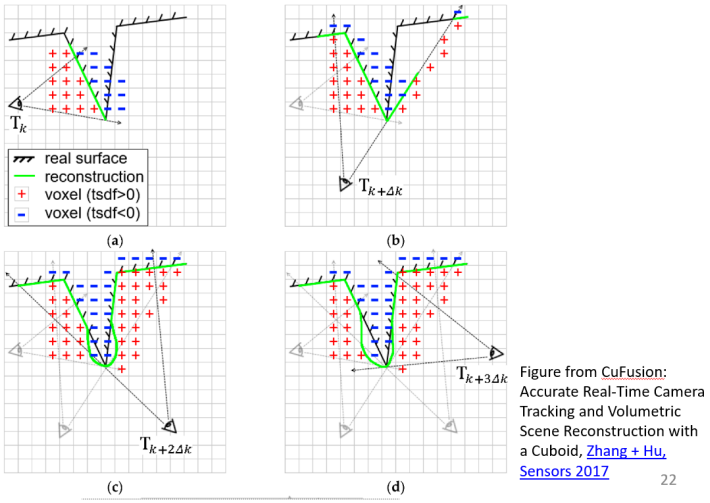



Figure 9.30: Depth Fusion: Given multiple depth-maps with camera poses, fuse into 3D voxels using Truncated Signed Distance Function

1. ***Depth Map Fusion:*** *Combines depth maps using the Truncated Signed Distance Function (TSDF), defined as:*

$$TSDF(p) = \begin{cases} d/p_{trunc} & \text{if } |d| \leq p_{trunc}, \\ 1 & \text{otherwise}, \end{cases}$$

*where* $d$ *is the signed distance of point* $p$ *from the surface and* $p_{trunc}$ *is a truncation threshold.*

2. ***Top-Down Aggregation:*** *Projects the 3D TSDF into 2D heightfields for efficient processing.*
3. ***Output:*** *A dense 3D reconstruction suitable for AR/VR applications.*

**Remark.** *Heightfields simplify 3D reconstruction while maintaining geometric accuracy, making them a practical choice for real-time applications.*

### 9.3.4 Final Key Points

- ***Neural Networks and Regularization:***
  - *Neural networks (NNs) have a high number of parameters, which increases the risk of overfitting.*
  - *Due to the presence of multiple local minima, regularization techniques are essential to guide optimization and improve generalization.*
- ***Data Source Selection:***
  - *Carefully select the upstream data source to ensure data quality and relevance to the task.*
- ***Posterior Probability and Loss Functions:***
  - *Clearly define the posterior probability or the loss function that you aim to optimize.*
  - *Ensure that the chosen objective aligns with the task requirements and constraints.*
- ***Debugging and Iteration:***
  - *Practical experience will teach efficient ways to debug and iterate during model development.*
  - *Emphasize learning through experimentation and testing different approaches.*
- ***Model Simplicity:***
  - *Always begin with the simplest model that is sufficient for your task.*
  - *Complexity can be added incrementally as required, but starting simple helps in understanding and debugging.*