COMP0137 Machine Vision Note

L.C.

April 2025

Contents

Introduction

Cheat sheet for the COMP0137 Machine Vision exam. Expected background knowledge:

- Probability (Bayes rule, conditional probability, expectation, random variables, basic combinatorics).
- Linear Algebra (up to and including singular value decomposition).
- Calculus (including partial derivatives).

Terminology

- **m** is the size of the training set.
- **n** is (usually) the number of dimensions in the input.
- $\hat{y}_i$ is the predicted output for input x_i.
- S is the training set.
- $\mathcal{E}$ is the expected error of a learning algorithm. $\mathcal{E}_{\text{emp}}(S, f)$ is the empirical error of an estimator f on training set S. When target (labels) are categorical, the error is measured as **error rate** - the proportion of cases where the prediction is wrong.
- MSE stands for mean square error.
- **i.i.d.** stands for Independent and Identically Distributed
- $\mathcal{I}[x]$ is the **indicator function**, $\mathcal{I}[x \in R_n] = 1$ if $x \in R_n$, and 0 otherwise.
- $\mathcal{H}$ is the hypothesis space

1 Introduction to machine vision

1.1 Discriminative vs. generative models

There are two (high-level) types of models:

- **Discriminative:** Build probability model of the world and make parameters depend on the data.
- **Generative:** Build a probability model of the data and make the parameters depend on the world. Use this model to infer probability of world.

2 Introduction to probability

2.1 Marginalization

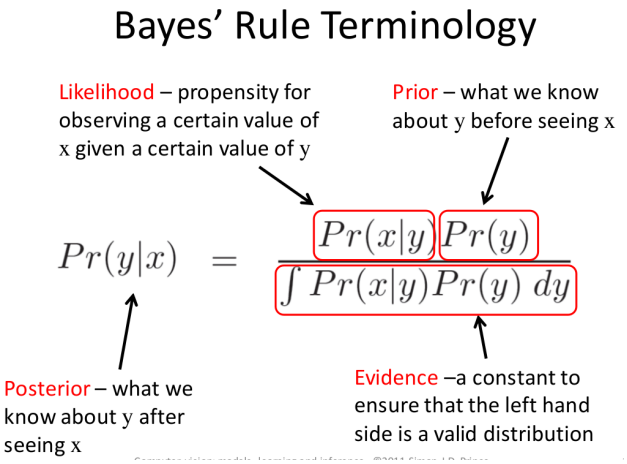
We can get the probability distribution of any variable in a joint distribution $Pr(x, y)$ (probability of x and y) by integration (for continuous) or summing (for discrete) over the other variable

- $Pr(x) = \int Pr(x, y)dy = \sum_y Pr(x, y)$
- $Pr(y) = \int Pr(x, y)dx = \sum_x Pr(x, y)$

  In higher dimension:
- $Pr(x, y) = \sum_w \int Pr(w, x, y, z)dz$

2.2 Bayes rule

Conditional probability: $Pr(x,y) = Pr(x|y)Pr(y) = Pr(y|x)Pr(x)$
In multidimensional case: $Pr(w,x,y,z) = Pr(w,x,y|z)Pr(z) = Pr(w,x|y,z)Pr(y|z)Pr(z)$



2.3 Expectation: common cases

$$\mathrm{E}[f[x]] = \int f[x]Pr(x)\,dx$$

Function $f[\bullet]$	Expectation
x	mean, μ_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

3 Models, learning and inference

3.1 Best distributions for different data types

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	$\mathbf{x} = [x_1, x_2, \ldots, x_K]^T$ $x_k \in [0, 1], \sum_{k=1}^{K} x_k = 1$	Dirichlet
bivariate, continuous, x_1 unbounded, x_2 bounded below	$\mathbf{x} = [x_1, x_2]$ $x_1 \in \mathbb{R}$ $x_2 \in \mathbb{R}^+$	normal-scaled inverse gamma
multivariate vector $\mathbf{x}$ and matrix $\mathbf{X}$ $\mathbf{x}$ unbounded, $\mathbf{X}$ square, positive definite	$\mathbf{x} \in \mathbb{R}^K$ $\mathbf{X} \in \mathbb{R}^{K \times K}$ $\mathbf{z}^T \mathbf{X} \mathbf{z} > 0 \quad \forall\, \mathbf{z} \in \mathbb{R}^K$	normal inverse Wishart

3.2 Conjugate distributions

3.2.1 Conjugate Relation

When we take the product of a distribution and its conjugate, the result has the same form as the conjugate.

- Beta is conjugate to Bernoulli
- Dirichlet is conjugate to categorical
- Normal Inverse Gamma (NIG) is conjugate to Univariate Normal
- Normal Inverse WiShart is conjugate to Multivariate Normal

3.2.2 Proof

Example: Proof that Beta $\times$ Bernoulli = constant $\times$ Beta

$$
\begin{aligned}
Pr(x|\lambda) &= Bern_x[\lambda] \\
Pr(\lambda) &= Beta_\lambda[\alpha\beta] \\
Bern_x[\lambda]Beta_\lambda[\alpha\beta] &= [\lambda^x(1-\lambda)^{1-x}] \times [\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\lambda^{\alpha-1}(1-\lambda)^{\beta-1}] \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\lambda^{x+\alpha-1}(1-\lambda)^{1-x+\beta-1} \\
Beta_\lambda[x+\alpha, 1-x+\beta] &= \frac{\Gamma(x+\alpha+1-x+\beta)}{\Gamma(x+\alpha)\Gamma(1-x+\beta)}\lambda^{x+\alpha-1}(1-\lambda)^{1-x+\beta-1} \qquad \textbf{(1)} \\
\textbf{(1)} &\Rightarrow [\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}] \times [\frac{\Gamma(x+\alpha)\Gamma(1-x+\beta)}{\Gamma(x+\alpha+1-x+\beta)}Beta_\lambda[x+\alpha, 1-x+\beta] \\
&= k(x,\alpha,\beta)Beta_\lambda[\alpha,\beta] \quad \text{Where } k(x,\alpha,\beta) \text{ is a constant.}
\end{aligned}
$$

3.2.3 Importance of Conjugate Relation

We can use conjugate to learn parameters by:

1. Choose prior that is conjugate to likelihood
2. Posterior must have same form as conjugate prior distribution
3. Posterior must be a distribution which implies that evidence must equal constant k from conjugate relation

We can use conjugate relation to marginalise over parameters:

$$Pr(y^*|x^*, X, Y) = \int Pr(y^*|x^*, \theta)Pr(\theta|X, Y)d\theta)$$

1. Choose $Pr(\theta|X, Y)d\theta)$ such that it is conjugate to other term
2. The integral then becomes $\int$constant $\times$ distribution = constant $\times$ **1** = constant

4 Fitting probability models

4.1 Maximum likelihood (ML)

Find the parameters which maximize the likelihood of seeing the observed data x, i.e. maximize $\Pr(x \mid \theta)$.

4.2 Maximum a posteriori (MAP)

Find the parameters which maximize the posterior, i.e. maximize $\Pr(\theta \mid x)$.

4.3 Bayesian approach

Instead of picking one set of parameters, model the distribution over many/all possible parameters using Bayes rule. That is, instead of just picking on θ that maximizes $\Pr(\theta \mid x)$, we compute $\Pr(\theta \mid x)$ over multiple possible θ's.

5 Normal distribution

5.1 Multivariate normal

Spherical or diagonal covariance matrix implies independence between variables, i.e. $\Pr(x_1, x_2) = \Pr(x_1)\Pr(x_2)$. Note that any covariance matrix can be decomposed into a rotational matrix R and a diagonal matrix Σ_{diag}. That is, $\Sigma_{full} = R^T \Sigma_{diag} R$.

Given a normal distribution, $\Pr(x) = Norm_x[\mu, \Sigma]$, we can generate a new distribution by transforming every point using $y = Ax + b$. The new distribution is then $\Pr(y) = Norm_y[A\mu + b, A^T \Sigma A]$. This trick can be used to generate new normal distributions.

6 Learning and inference

We observe measured data x, and infer the state of the world w. If w is continuous, we call it **regression**. If w is discrete, we call it **classification**.

A **model** relates the visual data x to the world state w. Model specifies a family of relationships, where a particular relationship depends on parameters θ. A **learning algorithm** fits parameters θ to paired training examples x_i, w_i. An **inference algorithm** uses the model to return $\Pr(w \mid x)$ for new observed data x.

For any type of model, we need to learn some parameters θ based on our training set - the tuples (x_i, w_i). We can use Bayes rule again:

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

6.1 Types of models

1. **(Discriminative)** Model $\Pr(w \mid x)$, dependence of world state on data. Note that since, we modelled $\Pr(w \mid x)$ directly, to do inference we can just evaluate this function and find the solution that maximises it.

2. **(Generative)** Model $\Pr(w, x)$, joint distribution of world state and data. In classification scenarios, we can't easily concatenate continous vector x with discrete vector w. After learning is done, we would have an expression for $\Pr(w, x)$ we can then define the posterior for inference as:

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

3. **(Generative)** Model $\Pr(x \mid w)$, dependence of data on world state. After learning is done, we would have an expression for $\Pr(x \mid w)$, so we can then define the posterior for inference as:
$$\Pr(w \mid x) = \frac{\Pr(x \mid w)\Pr(w)}{\int \Pr(x \mid w)\Pr(w)\,dw}$$

6.2 Which should we use?

Inference in discriminative models easy as we directly model posterior $\Pr(w \mid x)$. Generative models require more complex inference process using Bayes' rule.

That said, the data is really generated by the world, not the other way around. This makes generative models better when some data is missing. Generative also allow the user to incorporate some prior knowledge into the model.

7 Modeling complex densities

Here we rely heavily on the concept of **hidden variables**. That is, we represent our r.v. x as marginalization of joint density with some variable h which we cannot observer directly, i.e.:

$$\Pr(x) = \int \Pr(x, h)\,dh$$
$$\Pr(x \mid \theta) = \int \Pr(x, h \mid \theta)\,dh$$

7.1 Expectation Maximization (EM) algorithm

This is the algorithm that specializes on fitting pdfs which are a marginalization of distributions of x with a hidden variable h. For simplicity, assume that h is a positive integer in interval $\{1, ..., K\}$. Note that we can then marginalize over h using:

$$\Pr(x \mid \theta) = \sum_{k=1}^{K} \Pr(x, h = k \mid \theta)$$

We can use this approach to represent a mixture of distributions - for example $\Pr(x, h = 1)$ would be the first normal distribution, $\Pr(x, h = 2)$ is the second one, and so on. When we marginalize of all possible values of k, we get the total probability of x according to all these distributions.

The EM algorithm specializes on fitting pdfs that rely on the concept described above. The optimization problem we're trying to solve looks like this:

$$\hat{\theta} = \text{argmax}_\theta \left[\sum_{i=1}^{I} \log \left(\sum_{k=1}^{K} \Pr(x_i, h = k \mid \theta) \right) \right]$$

Expanding the expression for log-likelihood of our θ, we get:

$$\begin{aligned}\text{log-likelihood}(\theta) &= \sum_{i=1}^{I} \log(\Pr(x_i \mid \theta)) \\ &= \sum_{i=1}^{I} \log\left(\sum_{k=1}^{K} \Pr(x_i, h=k \mid \theta)\right)\end{aligned}$$

Let $q(h)$ be the distribution on the hidden variable h. We can then define a lower bound in terms of this distribution and parameters θ:

$$\text{lower-bound}(q(h), \theta) = \sum_{i=1}^{I}\sum_{k=1}^{K} q(h=k) \log\left(\frac{\Pr(x_i, h=k \mid \theta)}{q(h=k))}\right)$$

$$\Downarrow$$

$$\text{log-likelihood}(\theta) \geq \text{lower-bound}(q(h), \theta)$$

The EM algorithm works by first maximizing the lower bound with respect to distribution $q(h)$ in the E-step, and then maximizing the lower bound with respect to parameters θ in the M-step. That is, at every time step t:

1. E-step:
$$\hat{(q)}(h) = \Pr(h \mid x_i, \theta_t) = \frac{\Pr(x_i \mid h, \theta_t)\Pr(h \mid \theta_t)}{\Pr(x_i)}$$

2. M-step:
$$\theta_{t+1} = \text{argmax}_\theta \left[\sum_{i=1}^{I}\sum_{k=1}^{K} q(h=k) \log\left(\Pr(x_i, h=k \mid \theta)\right)\right]$$

7.2 Mixture of Gaussians (MoG)

As the name suggests, we represent the probability of x as a combination of K Gaussians, represented via the hidden variable h:

$$\begin{aligned}\Pr(x \mid h, \theta) &= Norm_x\left[\mu_h, \Sigma_h\right] \\ \Pr(h \mid \theta) &= Cat_h\left[\lambda\right] \\ &\Downarrow \\ \Pr(x \mid \theta) &= \sum_{k=1}^{K} \Pr(x, h=k \mid \theta) \\ &= \sum_{k=1}^{K} \Pr(x \mid h=k, \theta)\Pr(h=k \mid \theta) \\ &= \sum_{k=1}^{K} \lambda_k Norm_x\left[\mu_k, \Sigma_k\right]\end{aligned}$$

Note that we can generate data using this MoG by first sampling the value of h from $\Pr(h)$, then sampling x from $\Pr(x \mid h)$. The variable h also has a clear interpretation - it tells us which Gaussian generation the data point. The E and M steps are outlined in the slides below. Note that the calculation in E-step needs to be repeated for every data point.

E-Step

$$
\begin{aligned}
&Pr(h_i = k|\mathbf{x}_i, \boldsymbol{\theta}^{[t]}) \\
&\quad = \frac{Pr(\mathbf{x}_i|h_i = k, \boldsymbol{\theta}^{[t]})Pr(h_i = k, \boldsymbol{\theta}^{[t]})}{\sum_{j=1}^{K} Pr(\mathbf{x}_i|h_i = j, \boldsymbol{\theta}^{[t]})Pr(h_i = j, \boldsymbol{\theta}^{[t]})} \\
&\quad = \frac{\lambda_k \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]}{\sum_{j=1}^{K} \lambda_j \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j]} \\
&\quad = r_{ik}
\end{aligned}
$$

M-Step

$$
\begin{aligned}
\hat{\boldsymbol{\theta}}^{[t+1]} &= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\sum_{i=1}^{I} \sum_{k=1}^{K} \hat{q}_i(\mathbf{h}_i = k) \log \left[Pr(\mathbf{x}_i, \mathbf{h}_i = k|\boldsymbol{\theta}) \right] \right] \\
&= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\sum_{i=1}^{I} \sum_{k=1}^{K} r_{ik} \log \left[\lambda_k \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k] \right] \right].
\end{aligned}
$$

Take derivative, equate to zero and solve (Lagrange multipliers for λ)

$$
\begin{aligned}
\lambda_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik}}{\sum_{j=1}^{K} \sum_{i=1}^{I} r_{ij}} \\
\boldsymbol{\mu}_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik} \mathbf{x}_i}{\sum_{i=1}^{I} r_{ik}} \\
\boldsymbol{\Sigma}_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k^{[t+1]})(\mathbf{x}_i - \boldsymbol{\mu}_k^{[t+1]})^T}{\sum_{i=1}^{I} r_{ik}}
\end{aligned}
$$

7.3 Student t-distributions as marginalization

A Student t-distribution can be parametrized as follows:

$$
\begin{aligned}
\Pr(x) &= Stud_x \left[\mu, \Sigma^2, \nu \right] \\
&= \frac{\Gamma \left[\frac{\nu+1}{2} \right]}{\sqrt{\nu \pi \Sigma^2} \Gamma \left[\frac{\nu}{2} \right]} \left(1 + \frac{(x-\mu)^2}{\nu \Sigma^2} \right)^{-\frac{\nu+1}{2}}
\end{aligned}
$$

We can express it as marginalization, similar to what we did with Mixture of Gaussians:

$$
\begin{aligned}
\Pr(x) &= Norm_x\left[\mu, \sigma/h\right] \\
\Pr(h) &= Gam_h\left[\frac{\nu}{2}, \frac{\nu}{2}\right] \\
&\Downarrow \\
\Pr(x) &= \int \Pr(x, h)\, dh \\
&= \int \Pr(x \mid h) \Pr(h)\, dh \\
&= \int Norm_x\left[\mu, \sigma/h\right] Gam_h\left[\frac{\nu}{2}, \frac{\nu}{2}\right]\, dh \\
&= Stud_x\left[\mu, \Sigma, \nu\right]
\end{aligned}
$$

We can think of this result as an infinite mixture of Gaussians with the same mean but different variances. Note that we can still sample from this distribution, and we know that the covariance is inversely proportional to h. The updates associated with E and M steps can be seen below.

E-Step

$$
\begin{aligned}
q_i(h_i) = Pr(h_i|\mathbf{x}_i, \boldsymbol{\theta}^{[t]}) &= \frac{Pr(\mathbf{x}_i|h_i, \boldsymbol{\theta}^{[t]})Pr(h_i)}{Pr(\mathbf{x}_i|\boldsymbol{\theta}^{[t]})} \\
&= \frac{\text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}, \boldsymbol{\Sigma}/h_i]\text{Gam}_{h_i}[\nu/2, \nu/2]}{Pr(\mathbf{x}_i)} \\
&= \text{Gam}_{h_i}\left[\frac{\nu+1}{2}, 0.5(\mathbf{x}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}) + \frac{\nu}{2}\right]
\end{aligned}
$$

Extract expectations

$$
\begin{aligned}
\text{E}[h_i] &= \frac{(\nu+D)}{\nu + (\mathbf{x}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu})} \\
\text{E}[\log[h_i]] &= \Psi\left[\frac{\nu+D}{2}\right] - \log\left[\frac{\nu + (\mathbf{x}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu})}{2}\right]
\end{aligned}
$$

M-Step

$$
\begin{aligned}
\hat{\boldsymbol{\theta}}^{[t+1]} &= \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{i=1}^{I} \int \hat{q}_i(h_i) \log\left[Pr(\mathbf{x}_i, h_i|\boldsymbol{\theta})\right] dh_i \\
&= \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{i=1}^{I} \int \hat{q}_i(h_i) \left(\log\left[Pr(\mathbf{x}_i|h_i, \boldsymbol{\theta})\right] + \log\left[Pr(h_i)\right]\right) dh_i \\
&= \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{i=1}^{I} \int Pr(h_i|\mathbf{x}_i, \boldsymbol{\theta}^{[t]}) \left(\log\left[Pr(\mathbf{x}_i|h_i, \boldsymbol{\theta})\right] + \log\left[Pr(h_i)\right]\right) dh_i \\
&= \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{i=1}^{I} E\left[\log\left[Pr(\mathbf{x}_i|h_i, \boldsymbol{\theta})\right]\right] + E\left[\log\left[Pr(h_i)\right]\right],
\end{aligned}
$$

Where…

$$
\begin{aligned}
E\left[\log\left[Pr(\mathbf{x}_i|h_i, \boldsymbol{\theta})\right]\right] &= -\frac{d\log 2\pi + \log|\boldsymbol{\Sigma}| + \text{E}[\log h_i] + (\mathbf{x}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu})\text{E}[h_i]}{2} \\
E\left[\log\left[Pr(h_i)\right]\right] &= \frac{\nu}{2}\log\left[\frac{\nu}{2}\right] - \log\Gamma\left[\frac{\nu}{2}\right] + \left(\frac{\nu}{2}-1\right)\text{E}[\log h_i] - \frac{\nu}{2}\text{E}[h_i].
\end{aligned}
$$

Updates

$$
\begin{aligned}
\boldsymbol{\mu}^{[t+1]} &= \frac{\sum_{i=1}^{I}\text{E}[h_i]\mathbf{x}_i}{\sum_{i=1}^{I}\text{E}[h_i]} \\
\boldsymbol{\Sigma}^{[t+1]} &= \frac{\sum_{i=1}^{I}\text{E}[h_i](\mathbf{x}_i-\boldsymbol{\mu}^{[t+1]})(\mathbf{x}_i-\boldsymbol{\mu}^{[t+1]})^T}{\sum_{i=1}^{I}\text{E}[h_i]}
\end{aligned}
$$

7.4 Factor analysis

Factor analysis is sort similar to applying PCA with a hidden component, then doing marginalization.

$$\begin{aligned}
\Pr(x|h) &= Norm_x\left[\mu + \Phi h, \Sigma\right] \\
\Pr(h) &= Norm_h\left[0, I\right] \\
&\Downarrow \\
\Pr(x) &= \int \Pr(x, h)\, dh \\
&= \int \Pr(x \mid h) \Pr(h)\, dh \\
&= \int Norm_x\left[\mu + \Phi h, \Sigma\right] Norm_h\left[0, I\right]\, dh \\
&= Norm_x\left[\mu, \Phi\Phi^T + \Sigma\right]
\end{aligned}$$

7.5 Combining models

We can combine the aforementioned models in different ways:

- Mixture of t-distribution = MoG + t-distributions
- Robust subspace = t-distributions + factor analysis
- Mixture of factor analysis = MoG + factor analysis
- Mixture of robust subspace = MoG + t-distributions + factor analysis

$$\text{Mixture of robust subspace } = \sum_{k=1}^{K} \lambda_k Stud_x[\mu_k, \Phi_k\Phi_k^T + \Sigma_k, \nu_k]$$

8 Regression

8.1 Linear regression

Let's augment our data x_i with an extra 1 in the end, so we get the bias term for "free". We will consider the discriminative model $\Pr(w_i|x_i, \theta)$. We can then express this posterior as a Normal distribution:

$$\Pr(w_i|x_i, \theta) = Norm_{w_i}\left[\phi^T x_i, \sigma^2\right]$$

We can vectorize this problem by combining w_i's into matrix w and x_i into matrix X:

$$\Pr(w|X, \theta) = Norm_w\left[X^T\phi, \sigma^2 I\right]$$

We then solve standard **maximum likelihood**:

$$\phi, \sigma^2 = \text{argmax}_{\phi,\sigma^2}\left[\Pr(w|X, \phi, \sigma^2)\right]$$

Taking the log of the likelihood, then taking derivatives with respect to each parameter and setting to zero, we obtain:

$$\phi = (XX^T)^{-1}Xw$$
$$\sigma^2 = \frac{\left\|w - X^T\phi\right\|^2}{I}$$

We can also try the **Bayesian approach** to the same problem. Our likelihood stays the same, but we introduce a new prior:

$$\textbf{Likelihood:} \quad \Pr(w|X,\theta) = Norm_w\left[X^T\phi, \sigma^2 I\right]$$
$$\textbf{Prior:} \quad \Pr(\phi) = Norm_\phi[0, \psi^2 I]$$

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

8.2 Non-linear regression

For non-linear regression, we simply pass our vector x_i through some non-linear feature map f to get $z_i = f(x_i)$. We then do linear regression on z_i, same as above. Potential functions we could use for every dimension of z_i (plus the bias term!) are shown below:

$$\textbf{Radial basis functions:} \quad z_{ij} = \exp\left[\frac{-(x_{ij} - \alpha_j)^2}{\lambda}\right]$$
$$\textbf{Arctan:} \quad z_{ij} = \arctan\left[\lambda x_{ij} - \alpha_j\right]$$

8.3 Gaussian process regression

Gaussian process regression is basically Bayesian regression that utilizes the kernel trick. Potential kernels are the linear kernel (standard inner product), the degree-p polynomial $k(v,t) = (\langle v,t\rangle + 1)^p$, and Gaussian kernel (RBF):

$$k(v,t) = \exp\left[\frac{-0.5\left\|v-t\right\|^2}{\lambda^2}\right]$$

Note that the Gaussian kernel is basically equivalent to having an infinite number of radial basis functions at every point in space, and thus it has an infinite-dimensional feature map. The parameter λ in the Gaussian kernel determines the "spread" - if λ is too large, the PDF will be too smooth; if λ is too small, the PDF will be disjoint. That said, the maximum likelihood solution for λ achieves a PDF that is neither too smooth nor disconnected.

The variance σ^2 in the Bayesian approach needs to be fitted using maximum likelihood after we marginalize over all ϕ. To solve for it, we need to use non-linear optimization.

8.4 Sparse linear regression

Sometimes, not every dimension of input x is informative, so we want to encourage sparsity by setting some elements of ϕ to be zero. We encourage this via a special prior on ϕ - the product of t-distributions. Let $\phi \in \mathbb{R}^D$ be our parameters. We will evaluate each element ϕ_d against a separate t-distribution, so we'll need D distributions in total. Each of these distributions will be parametrized by (the same) $Stud[0, 1, \nu]$.

We can't compute the parameters without doing non-linear optimization. We can approximate the solution with a normal distribution and a Gamma distribution. Once we compute the final solution, we throw away rows of X with high values h_i.

Note that this doesn't work for the non-linear case (with feature maps), since we need one hidden variable per dimension of input. We therefore move to the dual domain, solving the **dual linear regression** problem.

In dual linear regression, we represent our parameter ϕ as a linear combination of columns of X. To specify the weights in this linear combination, we use ψ to get:

$$\phi = X\psi$$

We then solve like we would normally solve in linear regression:

$$\Pr(w|X, \theta) = Norm_w\left[X^T X\psi, \sigma^2 I\right]$$

$$\psi = (X^T X)^{-1} w$$
$$\sigma^2 = \frac{\left\|w - X^T X\psi\right\|^2}{I}$$

The Bayesian case for dual linear regression is similar to the non-Bayesian case, and can also be kernelized. We can now extend this idea to add sparsity to dual linear regression (see next section).

8.5 Relevance vector machine

In the dual linear regression framework, we can introduce sparsity by placing a prior on the weights ψ that encourages many of them to be zero. Specifically, we use a product of Gaussian priors with individual variances: $p(\psi) = \prod_{i=1}^{n} \mathcal{N}(\psi_i|0, \alpha_i^{-1})$, where α_i are precision parameters. By optimizing over α_i (e.g., through type-II maximum likelihood or evidence approximation), many α_i will tend to infinity, effectively setting the corresponding ψ_i to zero. This results in a sparse model where only a subset of the data points (the "relevance vectors") contribute to the prediction. Thus, after computing the solution, we throw away examples x_i with high values h_i, where h_i corresponds to the precision α_i, as these data points do not significantly affect the model.

9 Classification

9.1 Iterative non-linear optimization

Unfortunately we can't always find a closed form solution for x that minimizes some $f(x)$, so we need to **iterative non-linear optimization**. To make sure that we find the global minimum, we should try to find a convex cost function (which only has one minimum). There are many types of iterative non-linear optimization that we can use:

- **Line search.** We fix some line s. We then perform extensive search along that line by adjusting the scalar λ to find $\lambda^* = \text{argmin}_\lambda f\left[\theta_t + \lambda s\right]$. We then set $\theta_{t+1} = \theta_t + \lambda^* s$.
- **Gradient descent.** Finding the steepest downhill gradient and taking a step in that direction. If we can't compute gradient exactly, can use finite differences approximation:
$$\frac{\partial f}{\partial \theta_j} = \frac{f[\theta + a e_j] - f[\theta]}{a}$$
- **Newton's method:** Newton's method exploits the fact that we can use the second derivative to determine how big of a step we need to take. We derive by taking the first two terms of the Taylor expansion of $f(\theta)$, then take the derivative and rearrange to obtain:
$$\theta_{t+1} = \theta_t + \lambda \left(\frac{\partial^2 f}{\partial \theta^2}\right)^{-1} \frac{\partial f}{\partial \theta}$$
The second inverse of the function is called the *Hessian* matrix. If Hessian is positive definite, then the function is convex. It is expensive to compute using finite differences.

9.2 Logistic regression

As always, we add 1 to the end of our input vectors to get the bias term for free. We then define the posterior for logistic regression using:

$$\Pr(w|\phi, x) = Bern_w\left[\frac{1}{1 + exp[-\phi^T x]}\right]$$

We express likelihood as:

$$\begin{aligned}\Pr(w|X, \phi) &= \prod_{i=1}^{I} \lambda^{w_i}(1-\lambda)^{1-w_i} \\ &= \prod_{i=1}^{I} \left(\frac{1}{1 + exp[-\phi^T x]}\right)^{w_i} \left(\frac{exp[-\phi^T x]}{1 + exp[-\phi^T x]}\right)^{1-w_i}\end{aligned}$$

Taking the logarithm and finding the derivative with respect to ϕ, we can compute the following expression:

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

There is no closed form solution, so we apply Newton's method. To do that, we also need to compute the second derivative:

$$\frac{\partial^2 L}{\partial \phi^2} = -\sum_{i=1}^{I} sig[a_i](1 - sig[a_i])x_i x_i^T$$

Let L denote our likelihood our function. Since we want to maximize L, we can minimize $-L$ using some form of gradient descent. The fact the second gradient above is always negative means that the second gradient of $-L$ is always positive, and thus the function is convex.

9.3 Bayesian logistic regression

We introduce a prior distribution over the parameters, typically a Gaussian prior: $\Pr(\boldsymbol{\phi}) = \mathcal{N}_{\boldsymbol{\phi}}(\mathbf{0}, \sigma_p^2\mathbf{I})$. Since this Gaussian prior is not conjugate to the Bernoulli likelihood used in logistic regression, the resulting posterior distribution $\Pr(\boldsymbol{\phi} \mid \mathbf{X}, \mathbf{w})$ does not have a simple closed-form expression. Exact Bayesian inference is intractable.

We can approximate the Bayesian approach. First, we find the Maximum A Posteriori (MAP) estimate for $\boldsymbol{\phi}$ by optimizing the log-posterior, which includes the log-likelihood and the log-prior:

$$\begin{aligned}
\hat{\boldsymbol{\phi}}_{\text{MAP}} &= \text{argmax}_{\boldsymbol{\phi}} \log \Pr(\boldsymbol{\phi} \mid \mathbf{X}, \mathbf{w}) \\
&= \text{argmax}_{\boldsymbol{\phi}} \left[\log \Pr(\mathbf{w} \mid \mathbf{X}, \boldsymbol{\phi}) + \log \Pr(\boldsymbol{\phi})\right] \\
&= \text{argmax}_{\boldsymbol{\phi}} \left[\sum_{i=1}^{I} \log \Pr(w_i \mid \boldsymbol{x}_i, \boldsymbol{\phi}) + \log \Pr(\boldsymbol{\phi})\right]
\end{aligned}$$

This optimization problem can be solved using methods like Newton's method. Once we find the MAP estimate $\hat{\boldsymbol{\phi}}_{\text{MAP}}$, we can approximate the true posterior $\Pr(\boldsymbol{\phi} \mid \mathbf{X}, \mathbf{w})$ using a Gaussian distribution centered at the MAP estimate. This is known as the Laplace approximation. The approximate posterior $q(\boldsymbol{\phi})$ is given by:

$$q(\boldsymbol{\phi}) = \Pr(\boldsymbol{\phi} \mid \mathbf{X}, \mathbf{w}) \approx \mathcal{N}_{\boldsymbol{\phi}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where the mean is the MAP estimate $\boldsymbol{\mu} = \hat{\boldsymbol{\phi}}_{\text{MAP}}$, and the covariance matrix $\boldsymbol{\Sigma}$ is the inverse of the negative Hessian of the log-posterior evaluated at the MAP estimate:

$$\boldsymbol{\Sigma} = \left(-\nabla^2_{\boldsymbol{\phi}} \log \Pr(\boldsymbol{\phi} \mid \mathbf{X}, \mathbf{w})\right)^{-1} \Big|_{\boldsymbol{\phi}=\hat{\boldsymbol{\phi}}_{\text{MAP}}}$$

Note that the Hessian includes terms from both the log-likelihood and the log-prior.

With this approximate posterior $q(\boldsymbol{\phi})$, we can make predictions for a new input $\boldsymbol{x}^*$. The predictive distribution for the new label w^* is obtained by marginalizing over $\boldsymbol{\phi}$:

$$\begin{aligned}
\Pr(w^* = 1 \mid \boldsymbol{x}^*, \mathbf{X}, \mathbf{w}) &\approx \int \Pr(w^* = 1 \mid \boldsymbol{x}^*, \boldsymbol{\phi}) q(\boldsymbol{\phi})\, d\boldsymbol{\phi} \\
&= \int \sigma(\boldsymbol{\phi}^T \boldsymbol{x}^*) \mathcal{N}_{\boldsymbol{\phi}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})\, d\boldsymbol{\phi}
\end{aligned}$$

Evaluating this integral is still challenging. We can approximate it further. Define the random variable $a = \boldsymbol{\phi}^T \boldsymbol{x}^*$. Since $\boldsymbol{\phi}$ is approximately Gaussian, a is also approximately Gaussian:

$$\Pr(a) \approx \mathcal{N}_a(\mu_a, \sigma_a^2) \quad \text{where} \quad \mu_a = \boldsymbol{\mu}^T \boldsymbol{x}^*, \;\; \sigma_a^2 = (\boldsymbol{x}^*)^T \boldsymbol{\Sigma} \boldsymbol{x}^*$$

The predictive probability can then be approximated by integrating the sigmoid function against this Gaussian distribution for a:

$$\Pr(w^* = 1 \mid \boldsymbol{x}^*, \mathbf{X}, \mathbf{w}) \approx \int \sigma(a) \mathcal{N}_a(\mu_a, \sigma_a^2) \, da$$

This integral can be approximated using the probit function or a common sigmoid approximation:

$$\Pr(w^* = 1 \mid \boldsymbol{x}^*, \mathbf{X}, \mathbf{w}) \approx \sigma\left(\frac{\mu_a}{\sqrt{1 + \pi\sigma_a^2/8}}\right) = \frac{1}{1 + \exp\left[-\mu_a / \sqrt{1 + \pi\sigma_a^2/8}\right]}$$

This allows for relatively fast computation of the approximate Bayesian predictive probability. Alternatively, the 1D integral over a can sometimes be evaluated more accurately using numerical quadrature.

9.4 Non-linear logistic regression

Non-linear regression is simple - we simply pass our data through some feature map, $z = f(x)$, then perform linear logistic regression as usual. Some potential functions we could use as feature maps are the heaviside step function, $arctan$, or radial basis functions. Note that if our feature map has some parameters, we need to compute the gradient with respect to these parameters too.

9.5 Gaussian process classification (kernelization) via dual logistic regression

Standard logistic regression finds a linear separator $\boldsymbol{\phi}^T \boldsymbol{x} = 0$. By working with the dual formulation (expressing the solution in terms of data points) and applying the kernel trick, we replace inner products $\boldsymbol{x}_i^T \boldsymbol{x}_j$ with a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This implicitly maps the data to a high-dimensional (possibly infinite) feature space, allowing for non-linear decision boundaries, leading to Kernel Logistic Regression.

Gaussian Process Classification (GPC) offers a related but distinct Bayesian approach. Instead of placing a prior on the parameters $\boldsymbol{\phi}$, GPC places a Gaussian process prior directly on the latent function $a(\boldsymbol{x})$ which determines the classification probability via $\sigma(a(\boldsymbol{x}))$. The GP prior is defined by a mean function (often zero) and a covariance function (a kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$), which encodes assumptions about the function's smoothness and correlations.

$$a(\cdot) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$$

$$\Pr(w = 1 \mid \boldsymbol{x}, a(\cdot)) = \sigma(a(\boldsymbol{x}))$$

Like Bayesian logistic regression, the posterior distribution over the latent function $a(\cdot)$ given data $(\mathbf{X}, \mathbf{w})$ is intractable due to the non-conjugate sigmoid likelihood. Approximations like the Laplace approximation (similar to the one used in Bayesian LR) or Expectation Propagation (EP) are required for inference and prediction. Both Kernel LR and GPC use kernels to achieve non-linearity, but GPC is inherently Bayesian and non-parametric, providing a full posterior distribution over functions and naturally incorporating uncertainty.

9.6 Relevance vector classification

Relevance Vector Classification (RVC) is the classification counterpart to the Relevance Vector Machine (RVM) for regression. It is a Bayesian sparse kernel method that aims to achieve sparsity similar to Support Vector Machines (SVMs) while providing probabilistic outputs. RVC uses a model of the same form as Kernel Logistic Regression or SVMs:

$$a(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b$$

$$\Pr(w = 1 \mid \boldsymbol{x}, \boldsymbol{\alpha}, b) = \sigma(a(\boldsymbol{x}))$$

However, instead of using margin maximization (SVM) or a simple prior (Kernel LR), RVC places a specific type of prior on the weights α_i: an Automatic Relevance Determination (ARD) prior. Typically, this is a zero-mean Gaussian prior with an individual precision (inverse variance) parameter γ_i for each weight α_i:

$$\Pr(\alpha_i \mid \gamma_i) = \mathcal{N}(\alpha_i \mid 0, \gamma_i^{-1})$$

Hyperpriors are placed on the γ_i parameters (often Gamma distributions). During the optimization process (typically type-II maximum likelihood, maximizing the marginal likelihood integrated over $\boldsymbol{\alpha}$), many of the precision parameters γ_i are driven to infinity. This forces the corresponding weights α_i to be exactly zero, resulting in a sparse model. The data points $\boldsymbol{x}_i$ corresponding to the non-zero weights α_i are called "Relevance Vectors". Unlike SVMs, RVC directly yields probabilistic predictions and often achieves greater sparsity (fewer relevance vectors than support vectors). Inference typically requires iterative methods like EM or variational Bayes approximations.

9.7 Incremental fitting and boosting

In incremental fitting, we start by some "simple" activation and incrementally increase its representation power. We re-express our activation as:

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

So we start with $K = 1$ and we fit ϕ_0, ϕ_1, ξ_1. In the second step, we set $K = 2$ and fit ϕ_0, ϕ_2, ξ_2 while leaving ϕ_1, ξ_1 unchanged, and so on. Note that every step we have to refit

the bias term. Interestingly, we can observe the derivative for this type of regression,

$$\frac{\partial L}{\partial \theta} = -\sum_{i=1}^{I}(w_i - sig[a_i])\frac{\partial a_i}{\partial \theta}$$

to see that points with correct labels don't contribute that much to our search gradient.

Boosting is essentially incremental fitting with some weak classifiers, e.g. $f(x) = Heaviside(\alpha_k^T x)$. For functions like Heaviside, we can't take the derivative so we just do exhaustive search.

9.8 Branching logistic regression and trees

In branching logistic regression, you simply have a special function in your activation that returns one logistic function if some threshold is passed and another if the threshold is not passed. For example, if $g[\cdot,\cdot]$ is our gating function that returns value between 0 and 1, we can express branches via:

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

The trees are formed by simply "stacking" multiple gating function on top of each other.

9.9 Multi-class regression

In multiclass regression, we need to pick some distribution over w, and make the parameters of this distribution a function of x. For example:

$$\Pr(w|x) = Cat_w[\lambda[x]]$$

We can use the softmax function to convert real activations a_k into numbers between zero and one that all sum up to 1:

$$\lambda_k = softmax_k[a_{1\ldots K}]\frac{exp[a_k]}{\sum_{j=1}^{K} exp[a_j]}$$

Where each activation is $a_k = \phi_k^T x$. We solve multiclass logistic regression using non-linear optimization over ϕ_k's.

9.10 Random classification tree

Random classification tree is basically a binary tree where at each level we have different gating functions. The gating function is chosen randomly at every split, and the threshold parameters t are chosen to maximize log probabilities (for example using the Heaviside step function).

A **fern** is an example of a tree where functions at a level are all the same, and thresholds on each level can be the same or different. This type of tree is very efficient to implement.

A **forest** is a collection of trees that averages the results of individual trees to get a more robust answer. Similar to Bayesian approach since it averages models with different parameters.

9.11 Non-probabilistic classifiers

Non-probabilistic classifiers include multilayer perceptrons, AdaBoost and support vector machines. They are primarily used for historical reasons.

Remember that **probabilistic** classifiers don't have any serious disadvantages. They can also be easily extended to multiclass case, they naturally produce an estimate of uncertainty and can be easily related to each other.

10 Graphical models (GMs)

10.1 Directed GMs

Directed graphical models are known as **Bayesian networks**. Undirected graphical models are known as **Markov networks**. If there is no route between two variables and they share no ancestors, the variables are independent.

A **Markov blanket** of a variable x_i contains its parents, its children and the parents of its children. Given its Markov blanket, variable x_i is independent of all other variables.

Let $PA(n)$ denote the set of parent nodes of node n. In **directed** graphical models, we then express the probability as:

$$\Pr(x_{1...N}) = \prod_{n=1}^{N} \Pr(x_n \mid x_{PA(n)})$$

Note that re-expressing the model this way lets us get rid of redundancy in the distribution by exploiting conditional independence. Where variables to not depend on each other, we don't need to store their combinations in our distribution.

10.2 Undirected GMs

In undirected graphical models, instead of probabilities we consider non-negative "potential" functions, which represent the potential of a particular clique. Let C be the total number of unique maximal cliques. Let ϕ_c denote the potential function of some clique S_c. We can then express the probability distribution of the undirected model as:

$$\Pr(x_{1...N}) = \frac{1}{Z} \prod_{c=1}^{C} \phi_c(S_c)$$

We can also express this as a Gibbs distribution if we set $\psi_c(S_c) = -\log(\phi(S_c))$ (which can be positive or negative):

$$\Pr(x_{1...N}) = \frac{1}{Z} exp\left(-\sum c = 1^C \psi_c(S_c)\right)$$

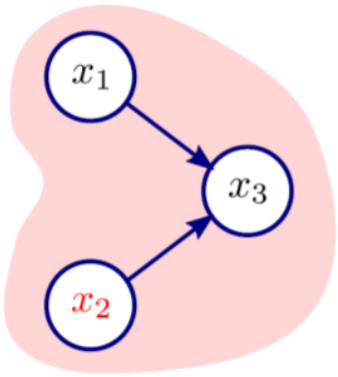
Z is the partition function, a.k.a. normalization constant. It is computed using:

$$Z = \sum_{x_1} \sum_{x_2} ... \sum_{x_N} \prod_{c+1}^{C} \phi_c\left[S_c\right] \qquad \text{where each } S_c \subseteq x_{1...N}$$

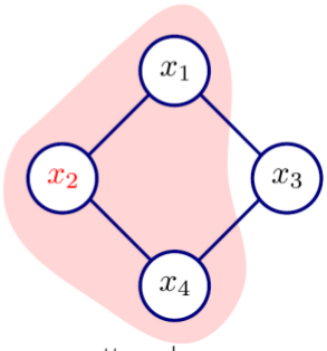
For large systems, computing Z is computationally intractable.

10.3 Directed vs undirected

Note that there are some relationships that can only be represented by directed or undirected models, and there are some that can't be represented using either of those. For example, below, the left example can only be represented using directed models, while the right example can only be represented using undirected models.



$x_2 \perp\!\!\!\perp x_1$



$x_1 \perp\!\!\!\perp x_4 | x_2, x_3$

$x_2 \perp\!\!\!\perp x_3 | x_1, x_4$

TL;DR Learning and sampling in directed models is easy. Learning and sampling in undirected models is hard.

10.4 Sampling the posterior of GMs

Note that computing the whole distribution, finding the MAP solution and even computing marginals is very expensive for big graphical models. One approach we could use is to compute the marginal posterior distribution for each variable w_n:

$$\Pr(w_n \mid x_{1\ldots N}) = \int \int \Pr(w_{1\ldots N} \mid x_{1\ldots N}) dw_{1\ldots n-1} dw_{n+1\ldots N}$$

Then for each individual w_n, we would find the maximum marginal solution:

$$\hat{w}_n = \text{argmax}_{w_n} \Pr(w_n \mid x_{1\ldots N})$$

Since we're computing each $\hat{w}_n$ independently, the final solution can still have probability 0, but it can still work. To speed up calculations, we consider generating samples from posterior $\Pr(w_{1\ldots N} \mid x_{1\ldots N})$. We then can use these samples to approximate the whole distribution - for example, we could compute empirical max-marginals to find an approximate solution.

Directed case: This case is easy - we start by sampling the parents, moving down to leaf nodes. We only sample a node after we have sample its parents. For each node, we condition the sample on the values we sampled from the parents. This is called **ancestral sampling**.

Undirected case: This case is much harder - we don't have clear parent/child relationships. We have to use the **Markov chain Monte Carlo** (MCMC) method. We stochastically generate a chain of samples where each sample depends on the previous one. One example of MCMC sampling is **Gibbs sampling**.

10.5 Gibbs sampling

1. To generate the initial vector, we start by sampling each of the N dimensions of the distribution in any order (usually the prior, if avaiable).

2. At step t, we pick the dimension with index $n = (t mod N) + 1$.

3. We fix the other $N - 1$ dimensions, and then update the value of x_n by sampling it from $\Pr(x_n \mid x_{1\ldots N \setminus n})$

4. We repeat steps 2 and 3 until we have enough samples.

Note that Gibbs sampling requires a burn-in (or warm-up) period, to make sure that the first samples are good. We also want to skip every $Q > 1$ samples to make sure adjacent samples are not correlated.

10.6 Learning in undirected models and contrastive divergence

Learning the parameters θ of an undirected graphical model (a Markov network) typically proceeds by maximum-likelihood estimation. Given data $\mathcal{D} = \{ x^{(d)} \}_{d=1}^{D}$, the log-likelihood is

$$\mathcal{L}(\theta) = \sum_{d=1}^{D} \log p(x^{(d)};\, \theta) = \sum_{d=1}^{D} \Big[-E(x^{(d)}; \theta) - \log Z(\theta) \Big],$$

where $E(x; \theta) = \sum_c \psi_c(S_c; \theta)$ is the energy and

$$Z(\theta) = \sum_{x} \exp\big(-E(x; \theta)\big)$$

is the partition function.

Taking the gradient gives

$$\frac{\partial \mathcal{L}}{\partial \theta} = -\sum_{d=1}^{D} \frac{\partial E(x^{(d)}; \theta)}{\partial \theta} \; + \; \sum_{d=1}^{D} \mathbb{E}_{p(x;\theta)} \left[\frac{\partial E(x; \theta)}{\partial \theta} \right].$$

The first "data" term is easy to compute, but the second "model" term requires expectations under $p(x; \theta)$, which in turn requires summing (or sampling) over all configurations—usually intractable for large models.

Contrastive Divergence (CD). To avoid full MCMC convergence at every gradient step, G. E. Hinton proposed *Contrastive Divergence* (CD-k). The algorithm for one mini-batch of data is:

1. **Positive phase:** Initialize each Markov chain at a data point $x^{(d)}$.

2. **Gibbs updates:** Run k full Gibbs-sampling steps to obtain a "reconstruction" $\tilde{x}^{(d)}$.

3. **Gradient estimate:**

$$\Delta\theta \;=\; -\frac{1}{|\mathcal{B}|}\sum_{d\in\mathcal{B}}\Big[\nabla_\theta E(x^{(d)};\theta) \;-\; \nabla_\theta E\big(\tilde{x}^{(d)};\theta\big)\Big],$$

   where $\mathcal{B}$ is the current mini-batch.

4. **Parameter update:** $\theta \leftarrow \theta + \eta\,\Delta\theta$.

Here the "negative phase" uses samples $\tilde{x}$ that are only a few steps away from the data, making each update fast while still pushing the model distribution to match the data distribution. Empirically, even $k = 1$ (CD-1) works well for many energy-based models such as Restricted Boltzmann Machines.

Remarks.

- As $k \to \infty$, CD-k approaches true maximum-likelihood learning (i.e. the exact gradient).
- Choosing k trades off bias (small k introduces approximation error) against computation per update.
- Variants like Persistent CD (PCD) maintain a set of "persistent" chains across mini-batches to reduce mixing bias.

11 Models for chains and trees

If the data set is big, i.e. N is large, the model relating together the world state w_i and the observation x_i will have a large number of parameters, which is often intractable. We can build sparse models that only describe a subset of relationships between variables.

11.1 Chain models

In a chain model, a world variable w_i is only connected to the next variables and the previous variable, e.g. $w_1 \to w_2 \to w_3$. We will assume that world states w_i are discrete, and that observed data for that world state, x_i is conditionally independent of all other data given the state w_i.

11.2 Directed vs undirected chain

Directed chain:

$$Pr(x_{1\ldots N}, w_{1\ldots N}) = \left(\prod_{n=1}^{N}\Pr(x_n \mid w_n)\right)\left(\prod_{n=1}^{N}\Pr(x_n \mid w_n)\right)$$

12 Preprocessing techniques

The purpose of preprocessing is to either "clean-up" the images, reducing unwanted variation in light, scale, deformation etc. Alternatively, it could also be used to reduce the size of the data, e.g. by reducing dimensionality.

Definition of preprocessing: deterministic transformation of pixels p to create data vector x.

12.1 Per-pixel techniques

Normalization involves improving the contrast and brightness of the image by bringing them to "standard" values. This involves bringing the mean and variance of pixel intensities to some standard (predefined) values.

Histogram equalization achieves a similar (but not identical) result by plotting a histogram of frequency of intensity values, then adjusting it to become a straight line.

Convolution involves applying a filter to every pixel based on the specified kernel. This can achieve various effects:

1. Gaussian kernel of different sizes can achieve blurring.
2. Gradient filters can be used to determine how quickly the intensity/colour changes - could be used for edge detection, or specifically, only-vertical or only-horizontal edge detection.
3. Gabor filters are orientation-sensitive filters that will give a strong response when the image has some texture in the same orientation as the filter. Useful for fingerprint enhancement and character recognition.
4. Haar features look at specific rectangles on the map (colour-coded as white or black), take the sum of intensities in each rectangle and then compute the difference between the two. The resultant difference can be used to classify a part of the image.
5. Local binary patterns (look up description online).

12.2 Textons

Textons basically aim to assign every pixel to some integer $i \in \{1...K\}$ representing the texture type. To compute textons, we apply a bank of filters to lots of images, then perform clustering in filter space to find "texture clusters". For a new pixel, we then classify it by filtering the region around it and assigning it to the nearest cluster in filter space.

12.3 Canny edge detector

Canny edge detector works in several stages:

1. Compute the horizontal and vertical gradient images h and v.
2. Combine h and v using $\sqrt{h^2 + v^2}$.

3. Perform non-maximal suppression to get make lines thinner.

4. Perform hysteresis thresholding to make edges clear.

12.4 Harris corner detector

Harris corner detector detects "windows" on the image such that walking away from that window changes the intensities a lot. This helps identify "good" corners. Example: If your window is on a straight horizontal, you can walk right and left without any change in intensities. But if your window is centred on a cross, moving in any direction will change intensities considerably.

12.5 SIFT descriptors

SIFT descriptors are amazing. They are invariant to scale, translation, rotation, illumination, view point. SIFT descriptors are derived in multiple steps:

1. First, we build the scale space by blurring the original image, halving its size, and storing the result. We repeat the same process multiple times to obtain multiple scales.

2. We compute difference of Gaussians (DoG) by finding difference between layers of the scale space.

3. We find key points by detecting maxima and minima of multiple DoG layers.

4. We remove "bad" keypoints. First we throw away keypoints that have magnitude in DoGs lower than some threshold. Then we compute the gradient on each keypoint - if both horizontal and vertical gradients are high, we have a corner and hence a good keypoint, so we keep it.

5. We assign orientation to each keypoint by computing the gradient in the region around it, building a histogram, normalizing it and then choosing the peak. All calculations are performed relative to this orientation to achieve rotation-independence.

6. Finally we compute SIFT descriptor by computing image gradients for 16x16 region around the keypoint. We then pool gradients into histograms, concatenate them and normalize them.

12.6 HoG descriptors

Histogram of Gradients (HoG) descriptor computes gradients for some fine grid in the image. Then, for every local region, it builds a histogram of these gradients and stores resultant count vector as a feature vector.

12.7 Bag of words descriptor

Bag of words descriptors are used for tasks like object classification. First, we need to produce a "dictionary" of k examples. The way we compute it is by first computing features of the image, e.g. SIFT features that gives 128-element vectors for each feature. We then perform k-means clustering on all of these features, and choose cluster centroids as the k "words" in our dictionary.

To classify a new input, we compute features in its region, assign label to each feature using the dictionary, and then compute histogram over all labels picking the most frequent one.

12.8 Shape context descriptor

Useful for character recognition. To generate the descriptor, we are given some "correct" contour for the object. We sample equidistant points from that contour, and generate the shape context descriptor for each point. This will give us some histogram - we record that histogram for each point.

To classify a new input, we detect the contour and sample points from that contour. We then compute the same histogram as above for each point, and match it to the point with the closest histogram from the training points.

12.9 Dimensionality reduction

Dimensionality reduction can be handled using a simple linear approximation. We compute some direction vector ϕ that agrees in direction with the most data points in our training set. We then represent each data point using that vector and some multiplier h_i:

$$x_i \approx \phi h_i$$

We can solve this for ϕ and $h_{i...I}$ as a least-squares problem. Since the solution is not unique, we need to constrain $\|\phi\| = 1$ using Lagrange multipliers. We get the problem:

$$\begin{aligned} E &= \sum_{i=1}^{I} \|x_i - \phi h_i\| + \lambda(\|\phi\|^2 - 1) \\ &= \sum_{i=1}^{I} \left[\|x_i\|^2 - 2h_i\langle\phi, x_i\rangle + h_i^2\right] + \lambda(\|\phi\|^2 - 1) \end{aligned}$$

You take derivative with respect to ϕ and h_i, equate resulting expression to zero and re-arrange, which should give you ϕ (after you solve for λ I guess). You then find the hidden value for each i using: $h_i = \langle\hat{\phi}, x_i\rangle$.

Alternatively, you can solve for ϕ very quickly by finding the first eigenvector of scatter matrix XX^T. After that solve for h_i as above. Note that X is a **flat** matrix:

$$X = [x_1\, x_2 \ldots x_I] \qquad (X \in \mathbb{M}_{D\times I})$$

12.9.1 Principal Component Analysis (PCA)

PCA is a more accurate approach to dimensionality reduction. Instead of using just one basis vector ϕ, we can use multiple vectors represented as a matrix Φ. To find such bases, we compute the eigenvalues of scatter matrix XX^T again. We take the first K eigenvectors (associated with K largest eigenvalues) and use them as our basis vectors. To find the h_i for each data point:

$$h_i = \Phi^T x_i$$

The columns of Φ (the basis vectors) are called *principal components*, and entries of $H = [h_1\, h_2 \ldots h_I]$ are called *loadings*.

Note that computing $XX^T \in \mathbb{M}_{I\times I}$ might be expensive. If $D < I$, we might want to compute the eigenvectors of $X^TX \in \mathbb{M}_{D\times D}$ instead. This is called **dual PCA**. We reparametrize our principal components as weighted sums of data:

$$\Phi = X\Psi$$

Where Ψ are the first K eigenvectors of X^TX. We compute h_i's same as before.

12.10 k-means algorithm

You intialize means to some random vectors from the set. You then assign labels based on these means. Then you recompute the means based on the labels. Repeat until convergence.

13 Pinhole camera model

Adding in extrinsic parameters

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & 0 \\ 0 & \phi_y & \delta_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}$$

Or for short:

$$\lambda\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{\Lambda} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{\Omega} & \boldsymbol{\tau} \\ \mathbf{0}^T & 1 \end{bmatrix} \tilde{\mathbf{w}}$$

Or even shorter:

$$\lambda\tilde{\mathbf{x}} = \mathbf{\Lambda} \begin{bmatrix} \mathbf{\Omega} & \boldsymbol{\tau} \end{bmatrix} \tilde{\mathbf{w}}$$

14 Models for transformation

There are several types of 1-to-1 transforms:

- **Eucledian** transform: Just rotation and translation, no scaling.

- **Similarity** transform: Rotation, translation, and scaling. The scaling is achieved by adding the Z element to the transaltion vector (thus bringing the image closer or farther away).
- **Affine** transform: Rotation, translation, scaling and skewing. Skewing is achieved by changing the magnitude of columns in the rotation matrix in a non-symmetric way.
- **Homography**: A 3×3 matrix obtained by combining the intrinsics matrix and the (homogeneous 2D) extrinsics matrix.

To **learn** a homography, we take some known points on a flat plane (e.g. a simple rectangle) and try to transform them onto some points in the image (e.g. book lying on a table) to figure out the transformation parameters. When doing **inference**, we perform the reverse process - we take points from the image and transform them into our known plane.

If 2 cameras are capturing the same scene (with some reference object on it), we can find a homography from each camera to the scene plane. This also implies that a transformation between 2 cameras is a homography. When the camera is rotated using rotation matrix Ω with no translation, one can show that the final homography is $\Phi = \Lambda\Omega\Lambda^{-1}$.

14.1 Learning

To account for noise during learning, we use the following model (replace $hom[...]$ with $euc[...]$ or other types of transformation):

$$\Pr(x|w) = Norm_x[hom[w, \Phi], \sigma^2 I]$$

A maximum likelihood approach then gives us a list squares problem:

$$\hat{\theta} = \text{argmin}_\theta \sum_{i=1}^{I} \|x_i - trans[w_i, \theta]\|$$

- **Learning Euclidean transforms:** Represent the problem in least-squares form (as above). Solve for translation first to obtain $\tau = \mu_x - \Omega\mu_y$. The solution Ω will now minimze the problem $|B - \Omega A|$, and can be computed by finding SVD $BA^T = ULV^T$ and then setting $\Omega = VU^T$.
- **Learning Similarity transforms:** Learn Ω using the same approach as Eucledian trasforms. Then solve for ρ and τ using:

$$\rho = \frac{\sum_{i=1}^{I}(x_i - \mu_x)^T\Omega(w_i - \mu_w)}{\sum_{i=1}^{I}(w_i - \mu_w)^T(w_i - \mu_w)} \qquad \tau = \frac{\sum_{i=1}^{I}(x_i - \rho\Omega w_i)}{I}$$

- **Learning affine transforms:** Affine transforms are linear, so we can simply write out $\Phi w_i + \tau = x_i$ as a linear system $Ax = b$, then solve for x.

- **Learning homography parameters:** Let Φ be our homography. Consider the following mapping:

$$\lambda \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

  Note that we can express λ as $\lambda = \phi_{31}u_i + \phi_{32}v_i + \phi_{33}$. Plugging this value for lambda back into equation and rearranging, we can obtain a linear system $A\bar{\phi} = 0$ where $x = (\phi_{11}, \phi_{12}..., \phi_{32}, \phi_{33})$. We set constraint $\left\|\bar{\phi}\right\| = 1$ to avoid the obvious zero solution. $\bar{\phi}$ can then be computed by doing SVD $A = ULV^T$ and setting $\bar{\phi}$ to be the last column of V.

  The approach above gives the Direct Linear Transform (DLT) solution, which minimizes algebraic error but might not be very good geometrically. Thus we use $\bar{\phi}$ as initial parameters to perform non-linear optimization.

14.2 Inference

To do inference, we simply find points in w in world space using the transform parameters Φ that we learnt. In the absense of noise, this amounts to simply inverting the transform:

$$\lambda' \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix}^{-1} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

14.3 Calibration

We can compute exterior orientation and intrinsics if we are given some points in world space w, and corresponding points on the image plane, x. One approach (not very efficient) is to fix some random Λ and optimize for extrinsics. Then, we fix the intrinsics estimates and optimize for Λ. Rinse, repeat. After convergence, use non-linear optimization.

14.4 Computing extrinsics parameters (exterior orientation)

If the points are not all planar, we can use **method 1**: computing proper extrinsics (full rotation matrix and a translation vector). Recall that for every pair $x = (u_i, v_i), w = (x_i, y_i, z_i)$ we'll have the following relationship (if we premultiply each side by Λ^{-1}):

$$\lambda' \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} & \tau_x \\ \phi_{21} & \phi_{22} & \phi_{23} & \tau_y \\ \phi_{31} & \phi_{32} & \phi_{33} & \tau_z \end{bmatrix}^{-1} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

As usual, we can find the expression for $lambda'$, then substitute it back in. This will let us rearrange terms to get a linear system $Az = 0$, where z holds all of our unknowns. We constrain $\|z\| = 1$, and the solution is then obtained by doing $A = ULV^T$ and then setting z to last column of V.

Note that once we rebuild the rotation matrix Ω from z, it might not be valid - rotation matrices must be orthogonal which we do not enforce in our solution. To make it orthogonal we compute $\Omega = ULV^T$ and then set $\hat{\Omega} = UV^T$. We then re-scale τ using ratios between the old rotation matrix and the new one:

$$\tau' = \sum_{m=1}^{3}\sum_{n=1}^{3}\frac{\hat{\Omega}_{mn}}{\Omega_{mn}}\tau$$

If the points are planar, we can use **method 2**: In this case, we're basically estimating a homography between two planes MINUS the intrinsics. So we start by estimating the homography Φ using the previously described techniques. We then factor out the intrinsics from this homography to get $\Phi' = \Lambda^{-1}\Phi$. Note that the relationship we now have is:

$$\begin{bmatrix}\phi'_{11} & \phi'_{12} & \phi'_{13}\\ \phi'_{21} & \phi'_{22} & \phi'_{23}\\ \phi'_{31} & \phi'_{32} & \phi'_{33}\end{bmatrix} = \lambda' \begin{bmatrix}\omega_{11} & \omega_{12} & \tau_x\\ \omega_{21} & \omega_{22} & \tau_y\\ \omega_{31} & \omega_{32} & \tau_z\end{bmatrix}$$

Where the matrix on the right represents our extrinsics (with incomplete rotation matrix, obviously). We rebuild our rotation matrix by computing SVD of the first two columns of Φ' and then multiplying by pseudo-identity:

$$\begin{bmatrix}\phi'_{11} & \phi'_{12}\\ \phi'_{21} & \phi'_{22}\\ \phi'_{31} & \phi'_{32}\end{bmatrix} = ULV^T$$

$$\begin{bmatrix}\omega_{11} & \omega_{12}\\ \omega_{21} & \omega_{22}\\ \omega_{31} & \omega_{32}\end{bmatrix} = U \begin{bmatrix}1 & 0\\ 0 & 1\\ 0 & 0\end{bmatrix} V^T$$

We find the last column of Ω by taking the cross-product of the first two columns. If $det(\Omega)$ becomes negative, we multiply the last column by -1 to make it positive. We then compute the scaling parameters for the pixels using:

$$\lambda' = \frac{\sum_{m=1}^{3}\sum_{n=1}^{2}\phi'_{mn}/\omega_{mn}}{6}$$

Finally, we compute τ using:

$$\tau = [\phi'_{13}, \phi'_{23}, \phi'_{33}]/\lambda'$$

14.5 Intrinsics calibration

When points are not planar. Note that the equation for projecting points in camera space to the image plane using Λ is linear in the elements of Λ. This means we can simply build a linear system and solve for its elements to find a least squares solution.

If the points are planar, we do pretty much the same thing

14.6 Reconstruction

When intrinsics and extrinsics of the camera are known, we can infer the position of some point w in world space given its projections x_j into J images. To do that, we write out the classic equation for perspective projection, eliminate λ_i, rewrite everything to build a linear system $Az = b$ where z is our point, then solve for z.

14.7 Robust estimation using RANSAC or PEaRL

We can use **RANSAC** algorithm for robust estimation. We start by choosing some small subset of data, then compute the parameters using it. We count the number of inliers for the model that uses these parameters. We repeat this process some fixed number of times. In the end, we use the inliers from the best-fit model to re-estimate the parameters.

If there are multiple planes with different orientation in the picture, we can use the PEaRL algorithm (propose, estimate, and re-learn). PEaRL introduces multiple labels l_i into the problem, where each label has a unique homography Φ_i. We then set a prior on the labes as a Markov random field that encourages adjacent pixels to have the same label. Inference is done using a variation of the alpha expansion algorithm.

15 Multiple cameras

In the problem of structure from motion, we want have the projection of some I 3D points onto J images. Using this information, we want to estimate the position of points in real world, and the parameters of the pinhole camera.

Epipole is the intersection point(s) on the image plane of each camera if were to draw a line from one optical centre to another. **Epipolar plane** is the plane defined by a 3D point and the optical centres of cameras. **Epipolar line** is the image in one camera of a ray through the optical centre and image point in the other camera.

15.1 Essential and fundamental matrices

The geometric relationship between two cameras is captured by the essential matrix. Let the first camera be at origin, and extrinsics of the second camera be Ω and τ. Essential matrix is defined as:

$$E = \tau_{cross}\Omega \qquad \text{where} \quad \tau_{cross} = \begin{bmatrix} 0 & -\tau_z & \tau_y \\ \tau_z & 0 & -\tau_x \\ -\tau_y & \tau_x & 0 \end{bmatrix}$$

Let x_1 be the projection of some world point w in the first camera and x_2 be the projection of the same point in the second camera. The relationship with the essential matrix is then $x_2^T E x_1 = 0$. E has rank of 2, 5 degrees of freedom and non-linear constraints between elements.

Epipolar lines are then:

$$\begin{aligned} l_1 &= x_2^T E \\ l_2 &= x_1^T E^T \end{aligned}$$

Note that all epipolar lines l_2 have to go through the epipole e_1:

$$x_2^T E e_1 = 0 \quad \text{for all } x_2^T$$

This only holds if e_1 is in the null space of matrix E. We find the null space by computing SVD $E = ULV^T$, and taking the last column of V. If we want to find the epipole e_2, we compute SVD of $E^T = VLU^T$ (achieved by transposing), and take the last column of U.

We can recover the rotation and translation from the essential matrix E using SVD:

$$\begin{aligned} E &= ULV^T \\ W &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &\Downarrow \\ \tau_{cross} &= ULWU^T \\ \Omega &= UW^{-1}V^T \end{aligned}$$

Note that we can obtain 4 different solutions by changing the sign of τ and replacing W by W^{-1} (which can flip the optical centre of either camera to the opposite side of the image plane).

Fundamental matrix is related to essential matrix but it considers matrices that can have different (non-normalized) intrinsic parameters. The relationship is:

$$E = \Lambda_2^T F \Lambda_1$$

Estimation of fundamental matrix: We can manually mark the same 3D point w on both image planes to obtain x_1 and x_2. Note that if epipolar lines l_1 and l_2 don't go through x_1 and x_2 respectively, the fundamental matrix is incorrect. We can iteratively minimize the square distances from points x_i to lines l_i to find the best solution.

Alternatively, we can use the **8 point algorithm**. For some known points x_1 and x_2, we can solve $x_2^T F x_1 = 0$. If we write it out in full:

$$\begin{bmatrix} x_{i2} & y_{i2} & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_{i1} \\ y_{i1} \\ 1 \end{bmatrix} = 0$$

Note we can rewrite this as a linear system $Af = 0$, where $f = (f_{11}, f_{12}, ...f_{33})$. To solve it, we need at least 8 pairs of points marked on each camera plane - we just use the least squares solution with a constraint $|b| = 1$. The solution f is then the last column of V from $A = ULV^T$. This solution can be used to start non-linear optimization.

Problems with 8 point algorithm: Does not always give rank 2, need to set last singular value to 0. The 8 points used must not be planar. Fails if there's not sufficient translation between the views. Can suffer from numerical issues, need to scale the data first.

There also exists a 7-point algorithm useful if fitting is done iteratively via RANSAC.

Two view reconstruction:

1. Find features using corner detection algorithm. Match features on each image using a greedy algorithm.
2. Fit fundamental matrix F using a robust algorithm like RANSAC. Find matching points that agree with F.
3. Extract E from F, then extract ω, τ from E.
4. Finally perform non-linear optimization over points and camera extrinsics.

15.2 Rectification

Rectification achieves the effect of making all epipolar lines horizontal and aligned with each other (they appear at the same level on both image planes). For planar rectification, we need to apply some transformation matrices Φ_1 and Φ_2 to image 1 and 2 respectively. We start by deriving $\Phi_2 = T_3T_2T_1$, where T_1 movew the image to the center of the camera, T_2 rotates the epipole in the horizontal direction, and T_3 moves the epipole to infinity:

$$T_1 = \begin{bmatrix} 1 & 0 & -\delta_x \\ 0 & 1 & -\delta_y \\ 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} cos(-\theta) & -sin(-\theta) & 0 \\ sin(-\theta) & cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/e_x & 0 & 1 \end{bmatrix}$$

There are multiple solutions that can now give us Φ_1 based on Φ_2. These solutions can be parametrized as:

$$\Phi_1(\alpha) = (I + e_2\alpha^T)\Phi_2 M$$

Where M comes from $F = SM$, decomposition of F into a skew-symmetric matrix S and the required homography M. One way to choose these parameters is to choose α such that it minimises the distance between pairs of points after $\Phi_1(\alpha)$ and Φ_2 are applied to each image (in least squares sense).

Planar rectification only works when the epipole lies **outside** of the image. If the epipole is inside the image, we need to perform polar rectification, which distorts the image much more but aligns the epipolar lines as required.

Once the images are rectified, we can perform **dense reconstruction** to estimate a dense depth map (which tells us disparity at each pixel). For this to be possible, the correct match for each point must lie in the same horizontal line on both points, which is exactly whta rectification achieves.

15.3 Bundle adjustment

Bundle adjustment is the process of refining initial estimates of structure and motion using non-linear optimization. The problem is related to likelihood maximization:

$$\hat{\theta} = \text{argmax}_\theta \sum_{i=1}^{I}\sum_{j=1}^{J} \log[\Pr(x_{ij}|w_i, \Lambda, \Omega_j, \tau_j)]$$

The problem has a least squares form of:

$$\hat{\theta} = \text{argmin}_\theta z^T z \qquad z = \begin{bmatrix} x_{11} - pinhole(w_1, \Lambda, \Omega_1, \tau_1) \\ x_{12} - pinhole(w_1, \Lambda, \Omega_2, \tau_2) \\ \vdots \\ x_{IJ} - pinhole(w_I, \Lambda, \Omega_J, \tau_J) \end{bmatrix}$$

The problem of this form is suitable to optimisation by techniques such as the Gauss-Newton method, with Jacobian matrix $J_{mn} = \frac{\partial z_m}{\partial \theta_n}$. When we need to invert $J^T J$, we can do so efficiently by exploiting the structure of the matrix.

16 Models for shapes

We can represent shapes using lines of form $ax + by + c = 0$, or conic:

$$x^T \begin{bmatrix} \alpha & \beta & \gamma \\ \beta & \delta & \epsilon \\ \gamma & \epsilon & \zeta \end{bmatrix} x = 0$$

We can also represent shapes as landmarks - these can be thought of as discrete samples from the underlying contour. They can be ordered (single continuous contour), ordered with wrapping (closed contour), or a collection of both closed and open contours.

Let $W = [w_1, ..., w_N]$ represent the N 2D landmark points we'll use to represent contours. We will define a likelihood function $\Pr(x|W)$ that will encourage landmarks to stick to the desired object/contour. We will also define the prior $\Pr(W)$ that will encourage contours to be smooth.

16.1 Snakes

We can detect the borders between images by doing Canny edge detection, but this alone would not be enough to define a likelihood because the resulting image is flat in most places. To solve this, we derive a distance image from the Canny borders, which varies smoothly and hence is suitable for likelihood. The likelihood function is then:

$$\Pr(x \mid w) \propto \prod_{n=1}^{N} exp(-(dist[x, w_n])^2)$$

We encourage smoothness using the following prior:

$$\Pr(W) \propto \prod_{n=1}^{N} exp[\alpha \cdot space[w, n] + \beta \cdot curve[w, n]$$

Where $space[w, n]$ encourages equal spacing and $curve[w, n]$ encourages low curvature:

$$space[w, n] = -\left(\frac{\sum_{n=1}^{N} \|w_n - w_{n-1}\|}{N} - \|w_n - w_{n-1}\| \right)^2 \quad curve[w, n] = -\|w_{n-1} + w_{n+1} + 2w_n\|$$

We will then need to solve a problem with $2N$ unknown using non-linear optimization: $\hat{W} = \text{argmax}_W \Pr(x|W)\Pr(W)$. With each iteration, the landmarks will "crawl" towards the contour, which is why this method is called **snakes**.

16.2 Template models

In template models, we assume we know some rigid template (contour) defined with points $W = \{w_n\}_{n=1}^N$. Our aim is to simply figure out the transform $trans(w, \Psi)$ that maps this template onto the image. We define the likelihood based on the distance of transform from desired position:

$$\Pr(x|W, \Psi) \propto \prod_{n=1}^{N} exp\left[-(dist(x, trans[w_n, \Psi]))^2\right]$$

Note that we don't have any prior parameters, but we could add that if needed. We then simply maximize the log-likelihood:

$$\hat{\Psi} = \text{argmax}_\Psi \sum_{n=1}^{N} \left[-(dist(x, trans[w_n, \Psi]))^2\right]$$

Since there is no closed form solution, we need to perform non-linear optimization (to speed up the process, we can compute the derivative using chain rule). Alternatively, we could use the **Iterative Closest Point** algorithm: For each point w_n, we find the closest point x (using Canny edges, for example) then compute a closed form transform, repeating the process until convergence.

16.3 Statistical shape models

Also known as *point distribution models* or *active shape models* (since they adapt to the image). We use likelihood similar to the template case:

$$\Pr(x_i|w_i) \propto \prod_{n=1}^{N} exp\left[-(dist(x_i, trans[w_{in}, \Psi_i]))^2\right]$$

Note that the i index in w_{in} denotes the current "example" - we are usually given a lot of examples of the shape (e.g. contours of hands of different sizes). Each example contains the same number of landmarks which are generally in the same relative position. Our aim is to find parameters such that as many examples as possible "fit well". We define the prior as:

$$\Pr(w_i) = Norm_{w_i}[\mu, \Sigma]$$

Note that elements μ_n of the mean μ represent means for each landmark. The **learning** is performed as follows. The training examples we are given, w'_{in}, are usually transformed in some way, i.e. $w'_{in} = trans[w_{in}, \Psi_i]$. Before we can learn the normal distribution for the prior, we need transform all examples into some common reference frame by working out the inverse transformation Ψ_i^-:

$$w_{in} = trans[w_{in}, \Psi_i^-]$$

This is usually done using generalized Procrustes analysis. We repeat the following two steps: First, we update all transformations to move the examples as close to the current mean as possible. Then we recompute the mean based on the new position of examples. Mathematically:

$$\textbf{Update transformations:} \quad \hat{\Psi}_i^- = \text{argmin}_{\Psi_i^-} \sum_{i=1}^{N} \left\| trans[w'_{in}, \Psi_i^-] - \mu_n \right\|^2$$

$$\textbf{Update means:} \quad \hat{\mu} = \text{argmin}_{\mu} \sum_{n=1}^{N} \left\| trans[w'_{in}, \Psi_i^-] - \mu_n \right\|^2$$

We perform MAP inference as follows:

$$\hat{w} = \text{argmax}_w \left[\max_{\Psi} \sum_{n=1}^{N} -(dist[x_i, trans[w_n, \Psi]])^2 + \log[Norm_w[\mu, \Sigma]] \right]$$

There is no closed form solution so we have to use non-linear optimization or ICP algorithm. Note there are a lot of parameters, and often some of them are not useful. To improve efficiency of our system, we can use a **subspace shape model**. We sample some small number K of shapes from the examples, and then represent new data as a combination of mean and these samples:

$$w_i = \mu + \sum_{k=1}^{K} \phi_k h_{ik} + \epsilon_i$$

Where ϵ_i is Gaussian noise with covariance $\sigma^2 I$. Probabilistic version:

$$\Pr(w_i | h_i, \mu, \Phi, \sigma^2) = Norm_{x_i}[\mu + \Phi h_i, \sigma^2 I]$$

If we set the prior to $\Pr(h_i) = Norm_{h_i}[0, I]$ and treat prior $\Pr(w_i)$ as marginalization over hidden variable h_i, we get:

$$\begin{aligned} \Pr(w_i) &= \int \Pr(w_i | h_i) \Pr(h_i) \, dh_i \\ &= \int Norm_{w_i}[\mu + \Phi h_i, \sigma^2 I] Norm_{h_i}[0, I] dh_i \\ &= Norm_{w_i}[\mu, \Phi\Phi^T + \sigma^2 I] \end{aligned}$$

We can learn parameters μ, σ^2 and ϕ directly from data. Let $w_i = [w_{i1}, w_{i2}, ..., w_{iN}]$, and the whole set of examples be $\{w_i\}_{i=1}^I$. First, we calculate the mean using:

$$\mu = \frac{\sum_{i=1}^{I} w_i}{I}$$

Then, we zero-centre the data by subtracting the mean from every example. We combine this zero centred data into matrix $W = [w_1 - \mu, ..., w_I - \mu]$. Then we compute eigendecomposition:

$$WW^T = UL^2U^T$$

And finally compute parameters $\hat{\sigma}^2$ and $\hat{\Phi}$:

$$\hat{\sigma}^2 = \frac{1}{D-K} \sum_{j=K+1}^{D} L_{jj}^2$$
$$\hat{\Phi} = U_K (L_K^2 - \sigma^2 I)^{1/2}$$

16.4 Subspace Shape Models

Having estimated the subspace basis Φ and noise variance σ^2, we can perform inference on a new shape instance w. We seek the low-dimensional code h that best explains w under the probabilistic model:

$$\Pr(w \mid h, \mu, \Phi, \sigma^2) = \mathcal{N}\big(w;\, \mu + \Phi h,\; \sigma^2 I\big), \quad \Pr(h) = \mathcal{N}(h; 0, I).$$

The posterior over h is Gaussian,

$$\Pr(h \mid w) = \frac{\Pr(w \mid h)\Pr(h)}{\Pr(w)} = \mathcal{N}\big(h;\, \hat{h},\, \Sigma_h\big),$$
$$\hat{h} = \big(I + \tfrac{1}{\sigma^2}\,\Phi^T\Phi\big)^{-1} \tfrac{1}{\sigma^2}\,\Phi^T(w - \mu), \qquad \Sigma_h = \big(I + \tfrac{1}{\sigma^2}\,\Phi^T\Phi\big)^{-1}.$$

For fitting to image data we embed this inference inside the MAP criterion:

$$\hat{w},\, \hat{h},\, \hat{\Psi} \;=\; \arg\max_{w,h,\Psi} \sum_{n=1}^{N} \big[-\big\|\mathrm{dist}\big(x,\, \mathrm{trans}[w_n, \Psi]\big)\big\|^2\big] \;+\; \log \Pr(w \mid h) \;+\; \log \Pr(h).$$

Optimization alternates between (1) estimating the transformation Ψ and shape code h given current w, and (2) updating $w = \mu + \Phi h$. This yields a fast shape fitting algorithm.

16.5 Active Appearance Models

Active Appearance Models (AAMs) extend statistical shape models by also modeling texture (pixel intensities) within the shape region. Let

$$g_i(\,x\,)$$

be the vectorized image appearance sampled over a common reference mesh (e.g. via warping to the mean shape). We compute the mean appearance $\bar{g}$ and appearance basis A by PCA:

$$G = \big[g_1 - \bar{g},\; \ldots,\; g_I - \bar{g}\big], \quad G\,G^T = U_g\,\Lambda_g^2\,U_g^T, \quad A = U_{g,K}\,(\Lambda_{g,K}^2)^{1/2}.$$

The combined shape-appearance model is

$$\begin{pmatrix} w \\ g \end{pmatrix} = \begin{pmatrix} \mu_w \\ \bar{g} \end{pmatrix} + \begin{pmatrix} \Phi_w & 0 \\ 0 & A \end{pmatrix} \begin{pmatrix} h_w \\ h_g \end{pmatrix} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I).$$

Fitting to a new image requires maximizing the appearance likelihood under warping parameters Ψ:

$$\hat{h},\, \hat{\Psi} = \arg\max_{h,\Psi} \Big\| g\big(x;\, \Psi\big) - \big[\bar{g} + A\,h_g\big] \Big\|^2 \;+\; \|h\|^2,$$

where $g(x; \Psi)$ denotes the warped image texture. Efficient algorithms use the "project-out" and "inverse compositional" methods for real-time performance.

16.6 Mixtures of Shape Models

Real-world object classes often exhibit multimodal shape variations (e.g. open vs. closed hand). We can model this by a mixture of M subspace models:

$$\Pr(w) = \sum_{m=1}^{M} \pi_m \mathcal{N}(w;\, \mu^{(m)},\, \Phi^{(m)}{\Phi^{(m)}}^T + \sigma_m^2 I),$$

with mixture weights π_m. Learning proceeds by EM:

- E-step: compute responsibilities $\gamma_{i,m} = \frac{\pi_m \Pr(w_i|m)}{\sum_j \pi_j \Pr(w_i|j)}$.
- M-step: update $\pi_m, \mu^{(m)}, \Phi^{(m)}, \sigma_m^2$ using weighted PCA on each cluster.

At inference, one selects the mode m with highest posterior and fits that subspace model.

16.7 Deep Statistical Shape Priors

Recent work leverages deep generative models to learn nonlinear shape manifolds. Let $z \in \mathbb{R}^d$ be a latent code. A decoder network $f_\theta(z)$ outputs landmarks w. Training minimizes

$$\min_\theta \sum_{i=1}^{I} \|w_i - f_\theta(z_i)\|^2 \quad + \quad \lambda \|z_i\|^2,$$

optionally with a VAE or GAN formulation to regularize z. Fitting to images uses gradient-based optimization in the latent space:

$$\hat{z},\, \hat{\Psi} = \arg\min_{z,\Psi} \sum_{n=1}^{N} \left\| \text{dist}(x, \text{trans}[f_\theta(z)_n, \Psi]) \right\|^2 \;+\; \lambda \|z\|^2.$$

Such nonlinear priors capture complex shape variations beyond the linear subspace.

16.8 Summary

We have surveyed three families of landmark-based shape models:

- **Active Contours (Snakes)**: purely image-driven, smoothness prior via curvature and spacing.
- **Template & ICP Models**: rigid or affine alignment of a known template; fast but no learning.
- **Statistical Shape Models**: learn shape subspaces (PDMs), mixtures, or deep priors; combine with image likelihood for robust fitting.

Each paradigm balances flexibility, prior knowledge, and computational cost. Modern systems often integrate these approaches, using learned shape priors embedded in deep networks for end-to-end contour detection and segmentation.

17 Temporal models

Used to model evolving systems with some state w that change over time. Measurements x_t come from (unknown) generative model $\Pr(x_t \mid w_{1...T})$. They tell us about w_t. The expected way for state to evolve is defined in time series: $\Pr(w_t \mid w_{1...t-1}, w_{t+1...T})$.

Assumptions: Only the immediate past matters (Markov), and measurement x_t only depends on w_t.

Temporal models are updated in two steps. First, we compute the prior for step t by considering the temporal model and the posterior of previous step **(Step 1, temporal evolution)**:

$$\Pr(w_t \mid x_{1...t-1}) = \int \Pr(w_t \mid w_{t-1}) \Pr(w_{t-1} \mid x_{1...t-1}) dw_{t-1}$$

Then, we update our predictions based on the measurement x_t and our prior to obtain the posterior for step t **(Step 2, measurement update)**:

$$\Pr(w_t \mid x_{1...t}) = \frac{\Pr(x_t|w_t) \Pr(w_t|x_{1...t-1})}{\int \Pr(x_t|w_t) \Pr(w_t|x_{1...t-1}) dw_t}$$

17.1 Kalman filter

Kalman filter is a model specifically chosen such that: if the posterior at time $t-1$ is Gaussian, prior and posterior and time t will also be Gaussian. Kalman filter equations are the rules for updating means and covariances of these Gaussians. The update equations are then:

$$\begin{aligned} \textbf{Time evolution:} \quad & w_t = \mu_p + \Psi w_{t-1} + \epsilon_p \\ \textbf{Measurement:} \quad & x_t = \mu_m + \Phi w_{t-1} + \epsilon_m \end{aligned}$$

Where ϵ_p, ϵ_m is Gaussian additive noise, Ψ is the state transition matrix and Φ related state and measurement. We can express them in terms of probabilities as:

$$\begin{aligned} \textbf{Time evolution (Prior):} \quad & \Pr(w_t|w_{t-1}) = Norm_{w_t}[\mu_p + \Psi w_{t-1}, \Sigma_p] \\ \textbf{Measurement (Posterior):} \quad & \Pr(x_t|w_t) = Norm_{x_t}[\mu_m + \Phi w_{t-1}, \Sigma_m] \end{aligned}$$

Where Σ_p, Σ_m, again, define additive Gaussian noise. The inference is performed as follows:

$$\begin{aligned} \textbf{State prediction:} \quad & \mu_+ = \mu_p + \Psi \mu_{t-1} \\ \textbf{Covariance prediction:} \quad & \Sigma_+ = \Sigma_p + \Psi \Sigma_{t-1} \Psi^T \\ \textbf{State update:} \quad & \mu_t = \mu_+ + K(x_t - \mu_m - \Phi \mu_+) \\ \textbf{Covariance update:} \quad & \Sigma_t = (I - K\Phi)\Sigma_+ \\ & K = \Sigma_+ \Phi^T (\sigma_m + \Phi \Sigma \Phi^T)^{-1} \end{aligned}$$

Problems: Kalman filter requires linear temporal and measurement equations. It also represents posterior as a normal distribution, which will not work when the true posterior is multimodal.

17.1.1 Smoothing

Fixed lag smoother: Perform prediction of w_t based on $x_{1...t+\tau}$, where τ is the lag. That is, we look into the "future" after t to make predictions about w_t. Obviously we can only do this once we reach step $t + \tau$.

Fixed interval smoother: Record the whole interval $x_{1...T}$, then predict w_t using $\Pr(w_t|x_{1...T})$.

17.1.2 Extended Kalman filter

Extended Kalman filter allows the usage of non-linear temporal and measurement equations of form $x_t = f(w_{t-1}, \epsilon_p)$ and $w_t = g(w_t, \epsilon_m)$. The trick is take Taylor expansion of each of the functions and treat them as locally linear. To do that, we compute the Jacobian matrix (matrix of the derivative) for each function with respect to both of its parameters (getting 4 Jacobians in total). In the Taylor expansion, we evaluate the derivative around $\mu_{t-1}, 0$ and for f and around $\mu_+, 0$ for g.

17.1.3 Unscented Kalman filter

UKF approximates the distribution as a sum of weighted particles with correct mean and variance. In normal KF and EKF, we only approximate the movement of one point, whereas in UKF we approximate using multiple particles (including the original point from EKF). The more particles we use, the more accurate will our approximation be.

UKF is the compromise between the speed of KF and the accuracy of particle filters.

17.2 Particle filters

Particle filters work better for non-linear scenarios than KF variants. They can represent multimodal non-Gaussian densities, they don't need data association but they are expensive. We represent probability distribution as a set of weighted particles $\hat{w}_{t-1}^{1...J}$:

$$\Pr(w_{t-1}|x_{1...t-1}) = \sum_{j=1}^{J} a_j \delta(w_{t-1-\hat{w}_{t-1}^{[j]}})$$

We use condensation to handle the logic. In step 1, we resample the particles based on their weights to obtain new unweighted particles. In step 2, we pass all of the particles through the temporal model. In step 3, we estimate the weight of each particle based on how compatible it is with the measurement. Then we repeat from step 1.

18 Neural nets

In **feed forward** neural networks, vectors of inputs are passed through a set of logistic regressors to produce vectors of outputs. The more layers in the neural net, the "separating polygon/shape" it can define, thus increasing its capacity through depth.

Each layer must usually have some activation function, such as ReLU, ELU, softmax, affine transform, cross entropy softmax, etc. The networks are trained using backpropogation - we compute the gradient of loss for each example with respect to its parameters and then propagate it backwards using chain rule. We perform stochastic gradient descent using:

$$\theta^{(t)} \leftarrow \theta^{t-1} - \epsilon_t \frac{\partial L(z_t, \theta)}{\partial \theta}$$

The important parts are to pick a good initialization, vary learning rate as training progresses (e.g. using ADAM optimizer) and to split data into mini-batches. We can use a validation set to make sure we don't overfit and to optimize hyperparameters. It is important to do sensible data optimization and to address class imbalance.

Overfitting can also be avoided by using dropout techniques during training (can also be done during training). Vanishing gradients can be tackled using batch normalization techniques.