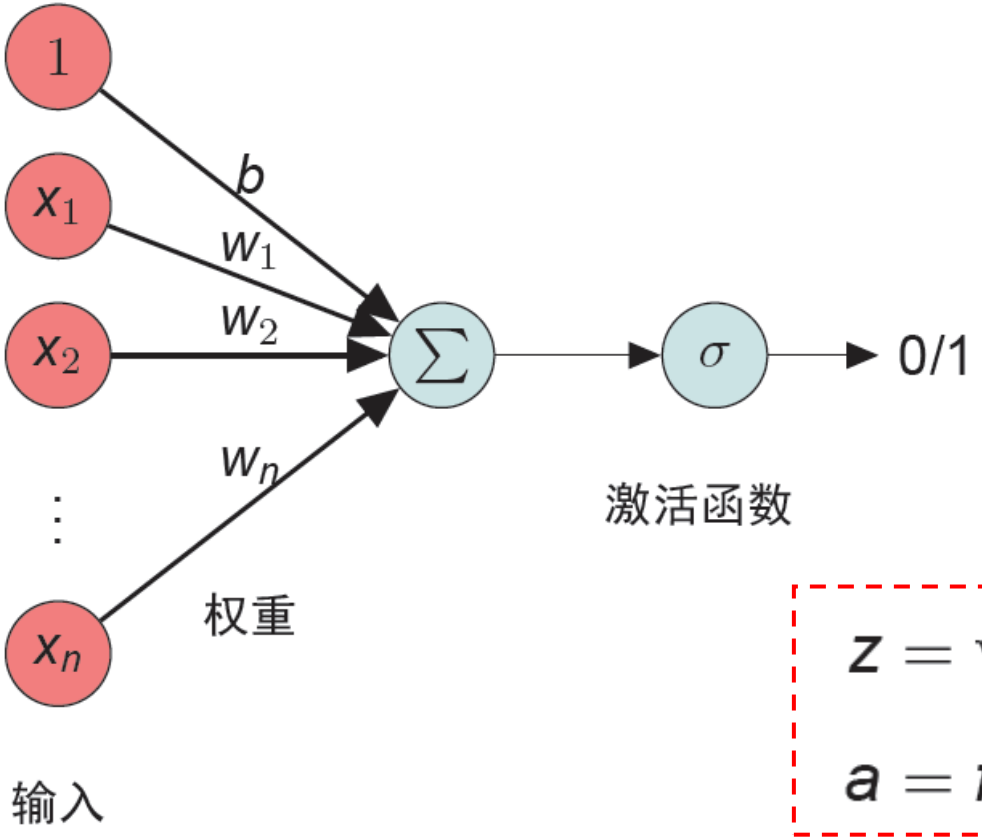
图像处理与识别

——Part 10 图像识别(II)

主讲：张磊

Multilayer Perceptron

- Feed-forward Neural Network----MLP

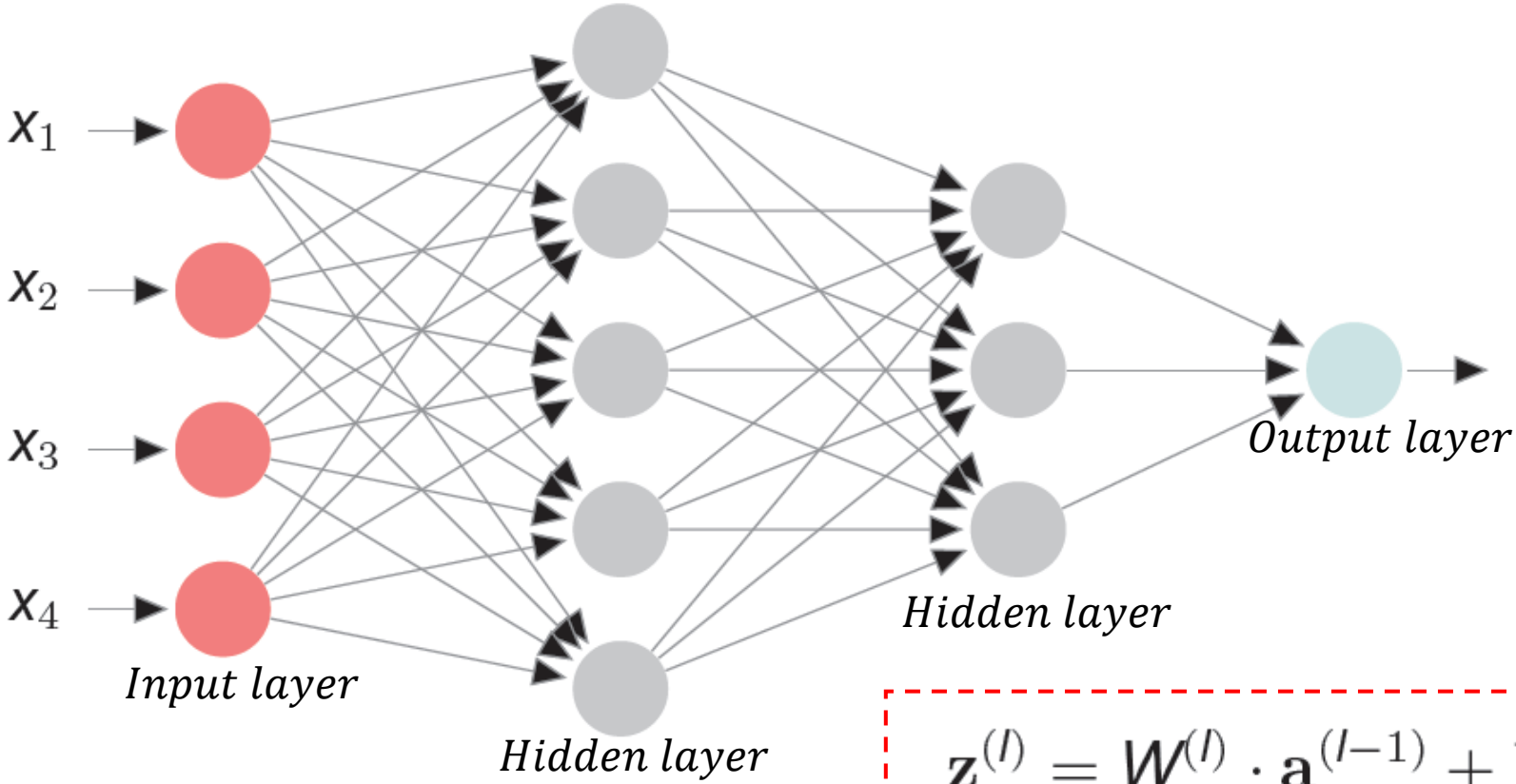


$$z = \mathbf{w}^{\top}\mathbf{x} + b$$

$$a = f(z)$$

Multilayer Perceptron

- Feed-forward Neural Network----MLP

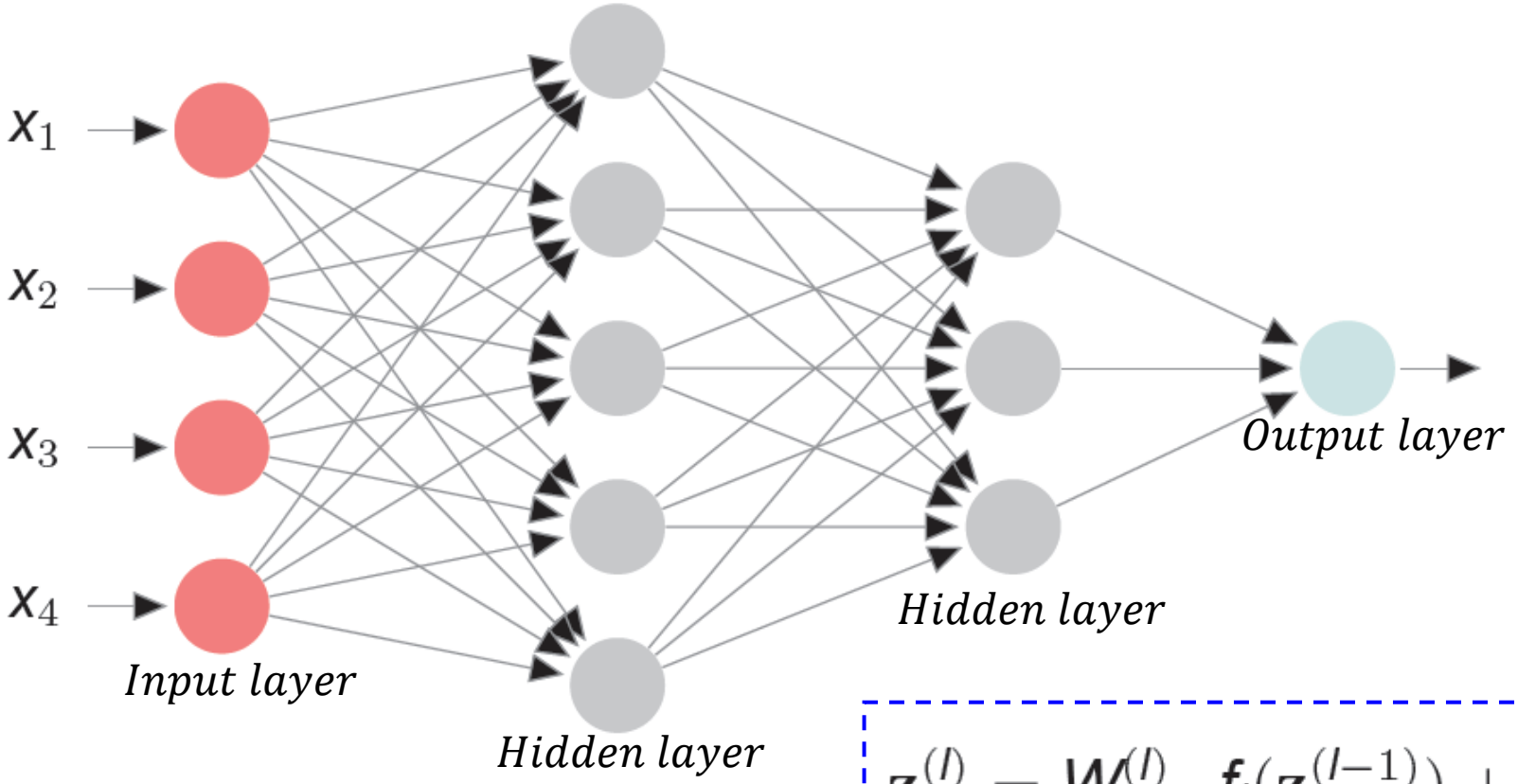


$$\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)})$$

Multilayer Perceptron

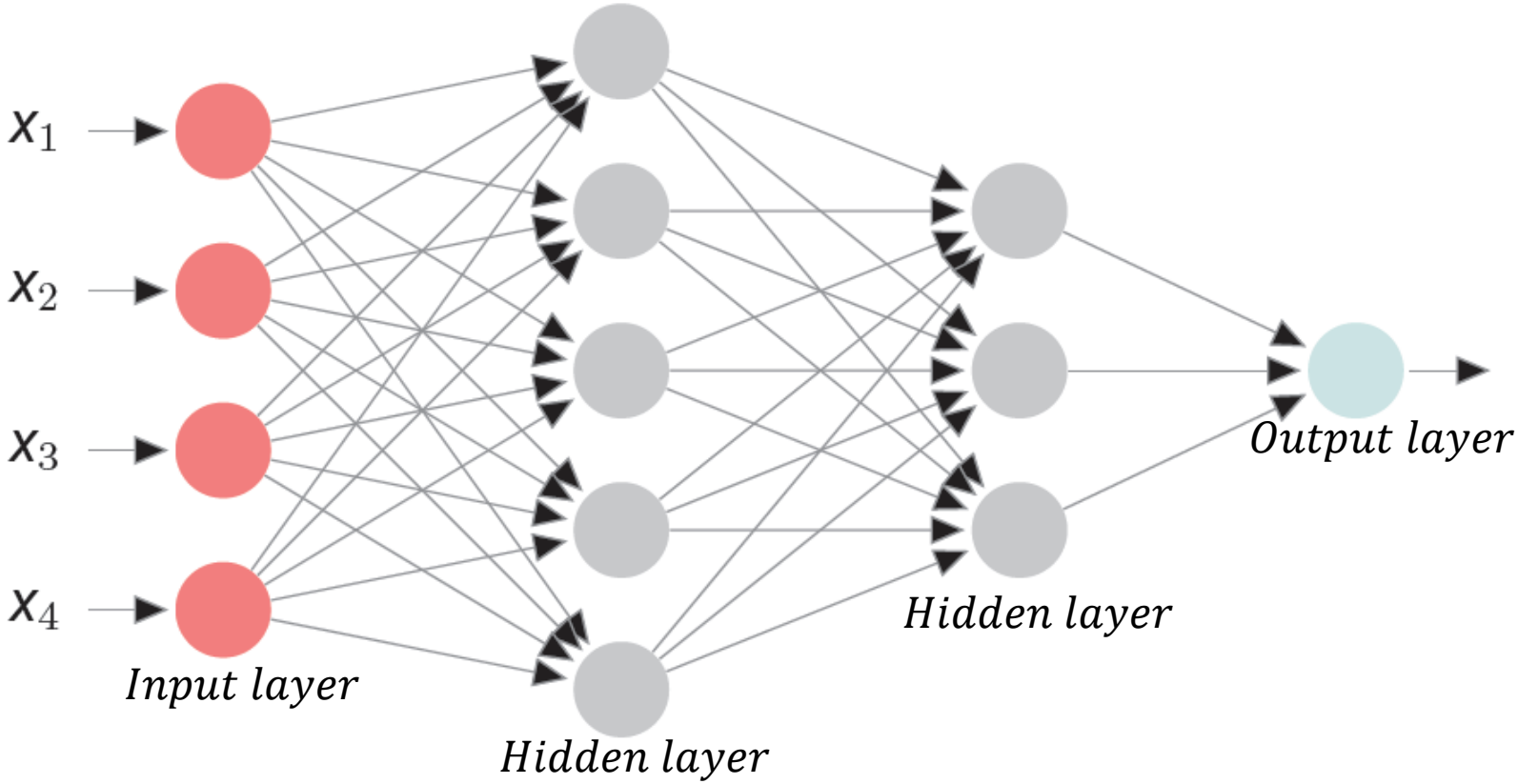
- Feed-forward Neural Network----MLP



$$\mathbf{z}^{(l)} = W^{(l)} \cdot f_l(\mathbf{z}^{(l-1)}) + \mathbf{b}^{(l)}$$

Multilayer Perceptron

- Feed-forward Neural Network----MLP



$$\mathbf{x} = \mathbf{a}^{(0)} \rightarrow \mathbf{z}^{(1)} \rightarrow \mathbf{a}^{(1)} \rightarrow \mathbf{z}^{(2)} \rightarrow \cdots \rightarrow \mathbf{a}^{(L-1)} \rightarrow \mathbf{z}^{(L)} \rightarrow \mathbf{a}^{(L)} = y$$

Multilayer Perceptron

- **Activation Function (Transfer function)**
  - Sigmoid (S-function)
    - Logistic function σ(x)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

    - Tanh function

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\tanh(x) = 2\sigma(2x) - 1$$

Multilayer Perceptron

- **Activation Function (Transfer function)**
  - Rectifier function

$$\mathbf{rectifier}(x) = \max(0, x)$$

rectifier 函数被认为有生物上的解释性。神经科学家发现神经元具有单侧抑制、宽兴奋边界、稀疏激活性等特性。

采用 rectifier 函数的单元也叫作**修正线性单元** *Rectifier linear unit (ReLU)*

  - Softplus function

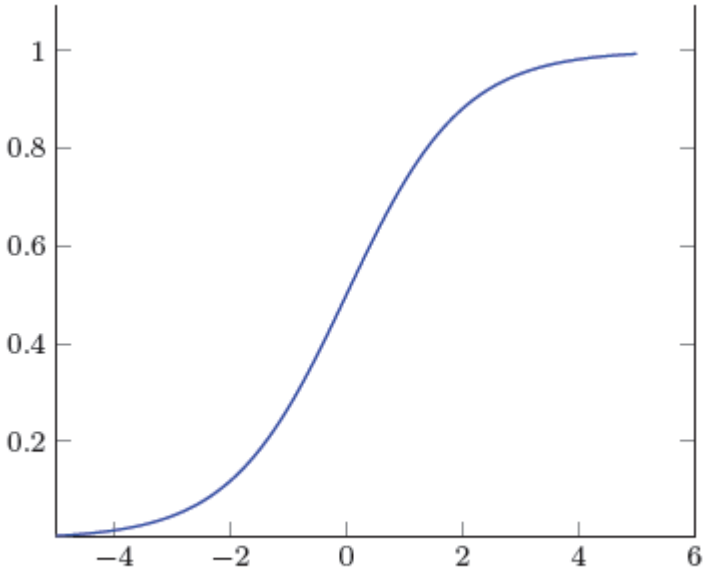
Softplus is a smooth version of Rectifier function.

$$\mathbf{softplus}(x) = \log(1 + e^x)$$

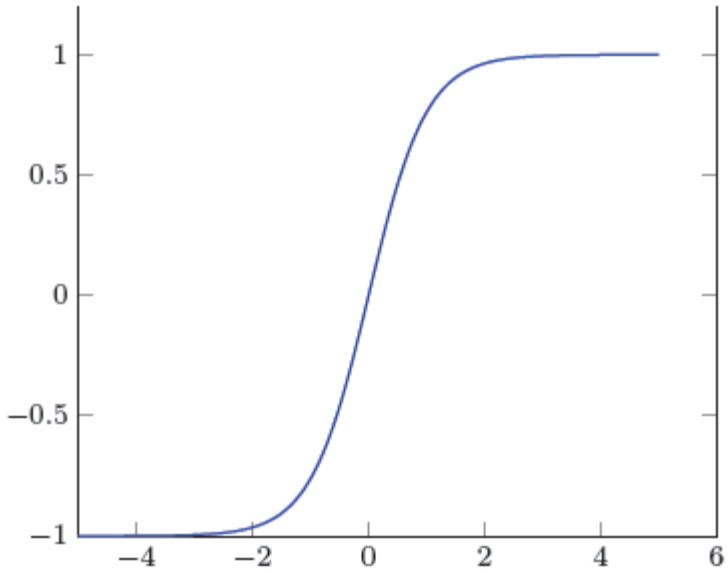
softplus 虽然也有具有单侧抑制、宽兴奋边界的特性 *what is its derivative? Logistic?*

但缺乏稀疏激活性

Multilayer Perceptron

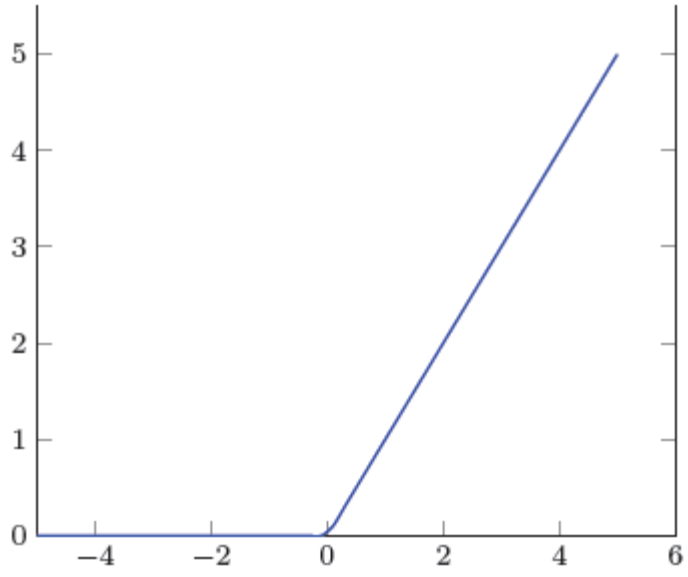


(a) logistic 函数

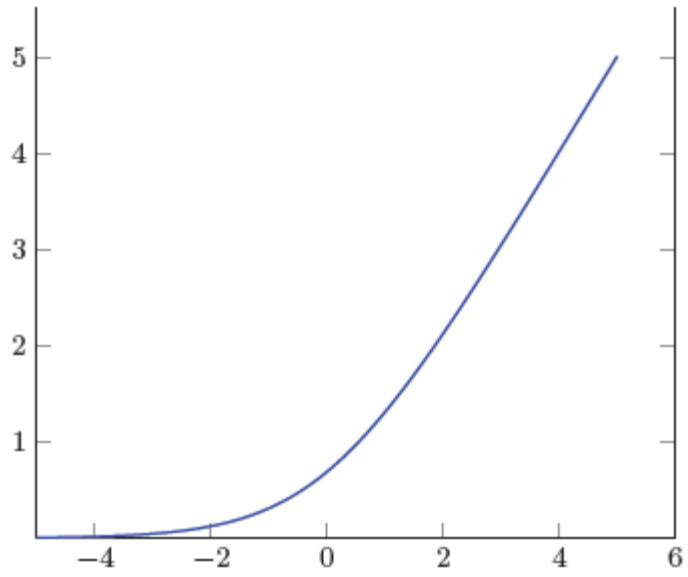


(b) tanh 函数

Multilayer Perceptron



(c) rectifier 函数



(d) softplus 函数

Multilayer Perceptron

- MLP Model

Given a group of samples $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right), 1 \leq i \leq N$, the network output is defined as $f(\mathbf{x}|\boldsymbol{W}, \mathbf{b})$, the objective function:

$$J(\boldsymbol{W}, \mathbf{b}) = \sum_{i=1}^{N} L(y^{(i)}, f(\mathbf{x}^{(i)}|\boldsymbol{W}, \mathbf{b})) + \frac{1}{2}\lambda\|\boldsymbol{W}\|_F^2,$$

$$= \sum_{i=1}^{N} J(\boldsymbol{W}, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}) + \frac{1}{2}\lambda\|\boldsymbol{W}\|_F^2,$$

这里，W 和 $\mathbf{b}$ 包含了每一层的权重矩阵和偏置向量

$$\|\boldsymbol{W}\|_F^2 = \sum_{l=1}^{L} \sum_{j=1}^{n^{l+1}} \sum_{j=}^{n^l} W_{ij}^{(l)}$$

Multilayer Perceptron

- Optimization

Analysis: our objective is to minimize $J(W, \mathbf{b}; \mathbf{x}, y)$, gradient descent (梯度下降) is often used.

Then the update of network weights (each layer)

$$W^{(l)} = W^{(l)} - \alpha \frac{\partial J(W, \mathbf{b})}{\partial W^{(l)}},$$

$$= W^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial W^{(l)}} \right) - \lambda W$$

Multilayer Perceptron

- Optimization

Analysis: our objective is to minimize $J(W, \mathbf{b}; \mathbf{x}, y)$, gradient descent (梯度下降) is often used.

Then the update of network bias (hidden and output layer)

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha \frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial \mathbf{b}^{(l)}},$$

$$= \mathbf{b}^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial \mathbf{b}^{(l)}} \right)$$

Multilayer Perceptron

- How to calculate $\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W^{(l)}}$?

链式求导法则

$$\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W_{ij}^{(l)}} = \left(\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}}\right)^{\top} \frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}$$

其中，$\delta^{(l)} = \frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}}$ 为误差项，表示目标函数关于第l层神经元 $\mathbf{z}^{(l)}$ 的偏导数，也反映第l层神经元对于目标函数的敏感性

Multilayer Perceptron

- How to calculate $\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W^{(l)}}$?

According to $\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$. There is,

$$\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}} = \frac{\partial (W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})}{\partial W_{ij}^{(l)}} = \begin{bmatrix} 0 \\ \vdots \\ a_j^{(l-1)} \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{第 } i \text{ 行}$$

So, $\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)} \Rightarrow \frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^{\top}$

Multilayer Perceptron

- How to calculate $\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{b}^{(l)}}$?

Similarly, $\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$

Finally, we have

$$\begin{cases} \frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^{\top} \\ \frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{b}^{(l)}} = \delta^{(l)} \end{cases}$$

Multilayer Perceptron

- How to calculate $\delta^{(l)}$ (第l层误差项)?

已知 $\delta^{(l)} = \dfrac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}}$

Then,

$$\delta^{(l)} \triangleq \frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}}$$

$$= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l+1)}}$$

$$= \mathbf{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^\top \cdot \delta^{(l+1)}$$

$$= f_l'(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^\top \delta^{(l+1)}),$$

其中 $\odot$ 是向量的点积运算符

可以看出第l层误差项可以由第$l+1$层误差项计算得到

Multilayer Perceptron

Back-propagation Algorithm

Input: Training set $(\mathbf{x}^{(i)}, y^{(i)})$, $i = 1, \dots, N$, maximum epoch T.

Output: $W, \mathbf{b}$

1. Random initialization of $W, \mathbf{b}$;
2. while 1
3. for $i = 1, \dots, N$ do
4. Feedforward calculating $\mathbf{z}^{(l)}$ and $\mathbf{a}^{(l)}, l = 1, \dots, L$; 状态值 激活值
5. Backward calculating the error term $\delta^{(l)}$;
6. Calculating the derivative of each layer

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^{\top}$$

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$

7. Parameter update

$$W^{(l)} = W^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial W^{(l)}} \right) - \lambda W;$$

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial \mathbf{b}^{(l)}} \right);$$

8. end
9. end

Face Recognition and Detection

The “Margaret Hildar Thatcher Illusion”, by Peter Thompson

Face Recognition and Detection

Recognition problems

- What is it?
  - Object and scene recognition
- Who is it?
  - Identity recognition
- Where is it?
  - Object detection
- What are they doing?
  - Activities
- All of these are **classification** problems
  - Choose one class from a list of possible candidates

What is recognition?

- A different taxonomy from [Csurka *et al.* 2006]:
- Recognition
  - Where is *this* particular object?
- Categorization
  - What *kind* of object(s) is(are) present?
- Content-based image retrieval
  - Find me something that looks similar
- Detection
  - Locate *all* instances of a given class

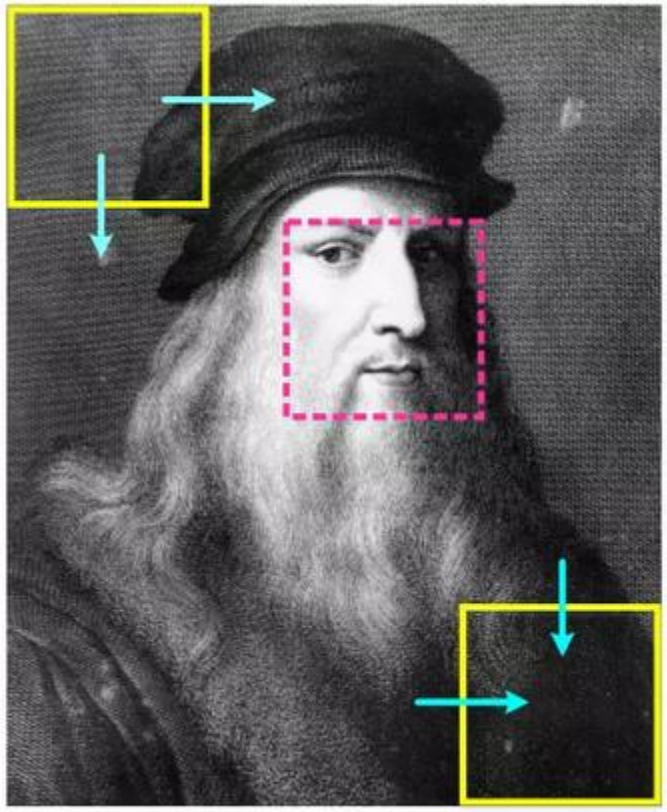
Readings

- C. Bishop, “Neural Networks for Pattern Recognition”, Oxford University Press, 1998, Chapter 1.
- Turk, M. and Pentland, A. *Eigenfaces for recognition*. Journal of Cognitive Neuroscience, 1991
- Viola, P. A. and Jones, M. J. (2004). Robust real-time face detection. *IJCV*, 57(2), 137 - 154.

Face detection

- How to tell if a face is present?

Face detection



Face detection

- Different face size? How to make the window size adapt to different face size?

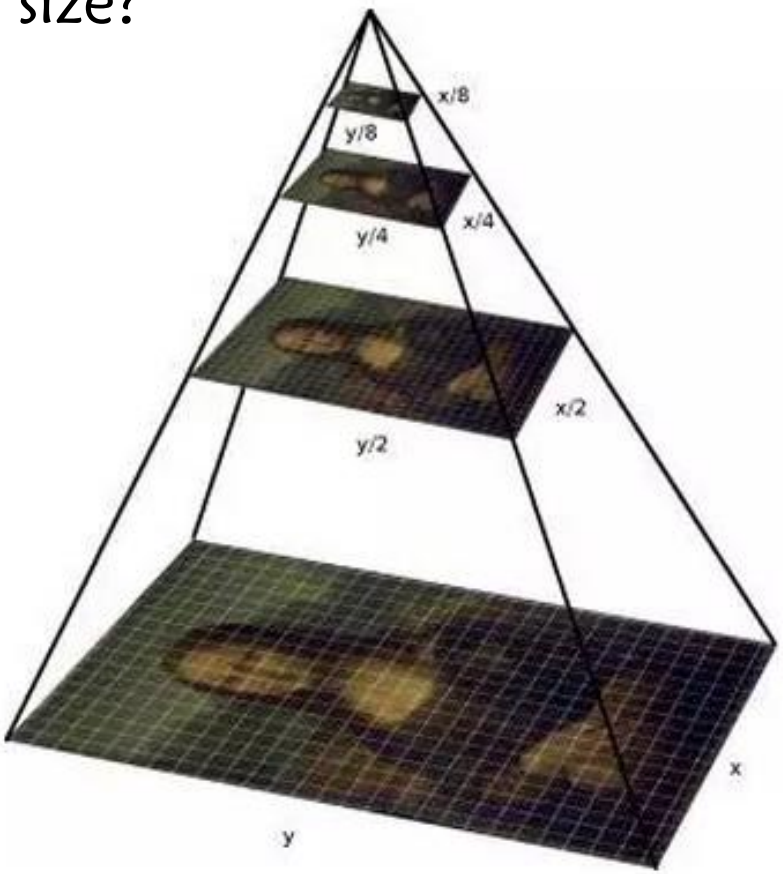
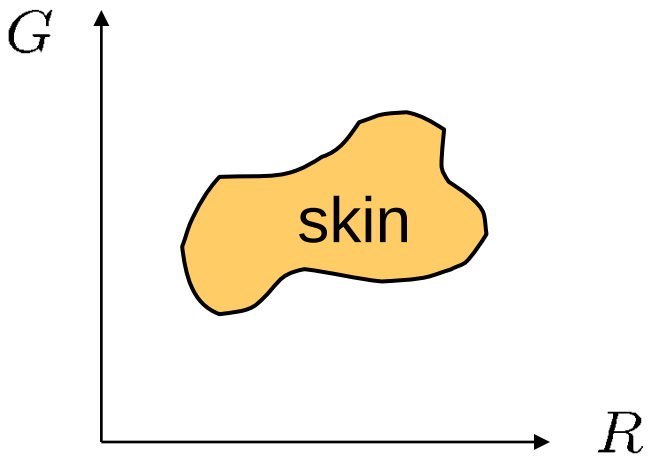


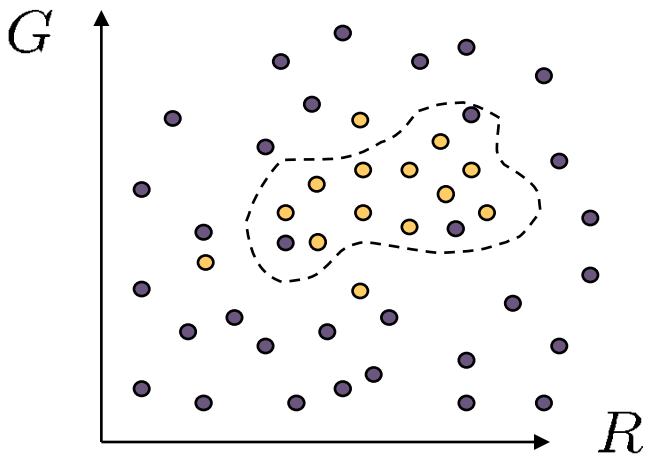
Image Pyramid

Skin detection



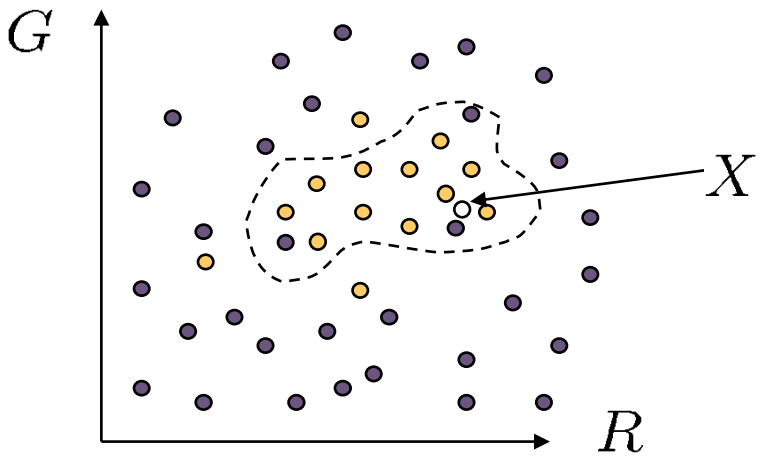
- Skin pixels have a distinctive range of colors
  - Corresponds to region(s) in RGB color space
- Skin classifier
  - A pixel X = (R,G,B) is skin if it is in the skin (color) region
  - How to find this region?

Skin detection



- **Learn** the skin region from examples
  - Manually label skin/non pixels in one or more "training images"
  - Plot the training data in RGB space
    - skin pixels shown in orange, non-skin pixels shown in gray
    - some skin pixels may be outside the region, non-skin pixels inside.

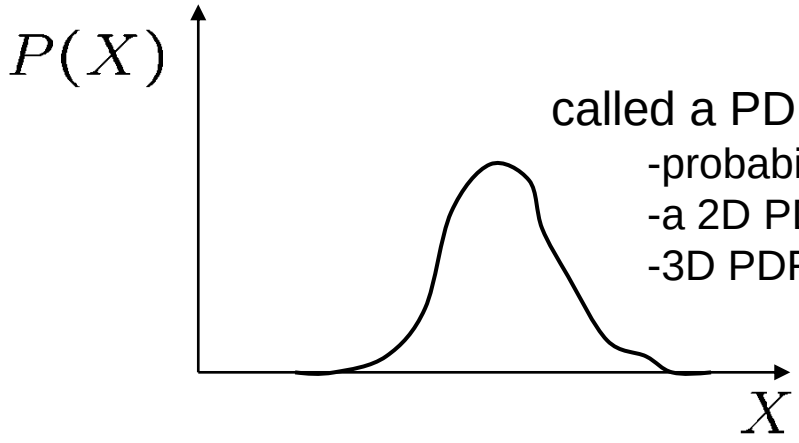
Skin classifier



- Given X = (R,G,B): how to determine if it is skin or not?
  - Nearest neighbor
    - find labeled pixel closest to X
  - Find plane/curve that separates the two classes
    - popular approach: Support Vector Machines (SVM)
  - Data modeling
    - fit a probability density/distribution model to each class

Probability

- X is a random variable
- P(X) is the probability that X achieves a certain value



called a PDF

-probability distribution/density function
-a 2D PDF is a surface
-3D PDF is a volume

$$0 \leq P(X) \leq 1$$

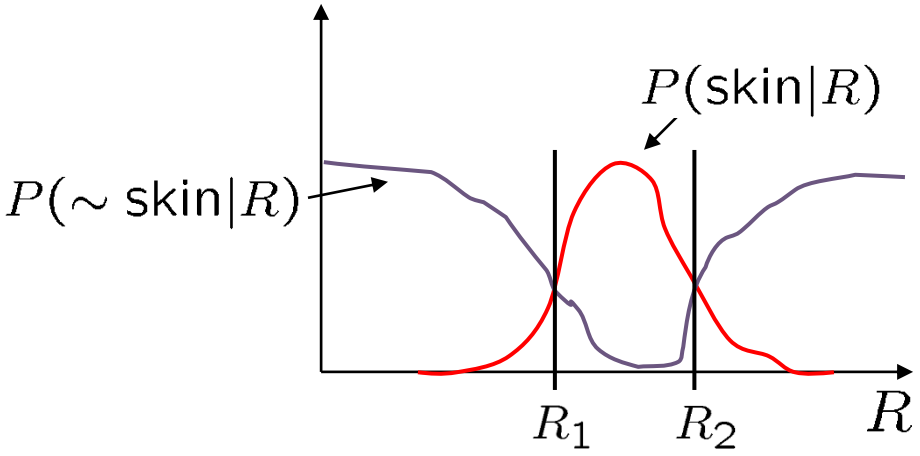
$$\int_{-\infty}^{\infty} P(X)dX = 1$$

continuous X

$$\sum P(X) = 1$$

discrete X

Probabilistic skin classification

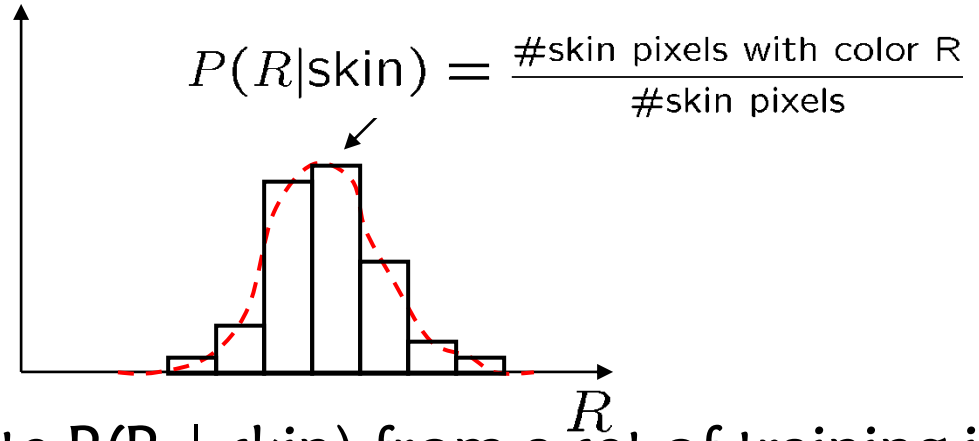


- Model PDF / uncertainty
  - Each pixel has a probability of being skin or not skin

$$P(\sim \text{skin}|R) = 1 - P(\text{skin}|R)$$

- Skin classifier
  - Given X = (R,G,B): how to determine if it is skin or not?
  - Choose interpretation of highest probability
- Where do we get $P(\text{skin}|R)$ and $P(\sim \text{skin}|R)$?

Learning conditional PDF's



- We can calculate P(R | skin) from a set of training images
- But this isn't quite what we want
  - Why not? How to determine if a pixel is skin?
  - We want P(skin | R) not P(R | skin)
  - How can we get it?

Bayesian rule

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

- In terms of our problem:

what we measure (**likelihood**)

domain knowledge (**prior**)

$$P(\text{skin}|R) = \frac{P(R|\text{skin})\ P(\text{skin})}{P(R)}$$

what we want (**posterior**)

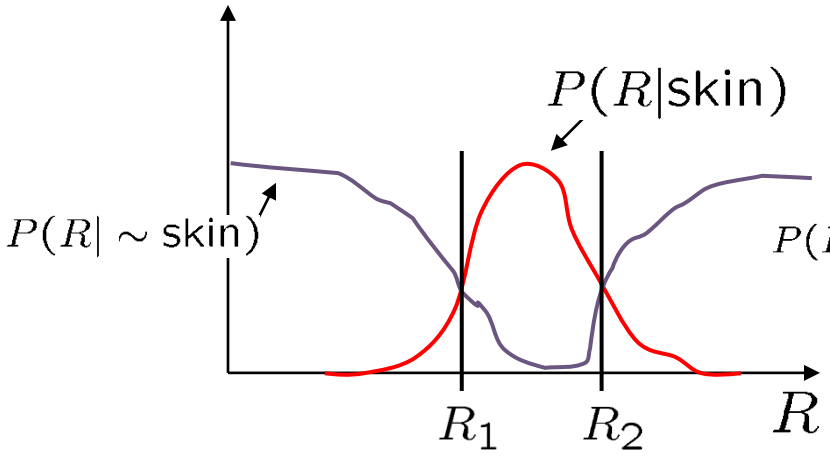
normalization term

$$P(R) = P(R|\text{skin})P(\text{skin}) + P(R|\sim\text{skin})P(\sim\text{skin})$$

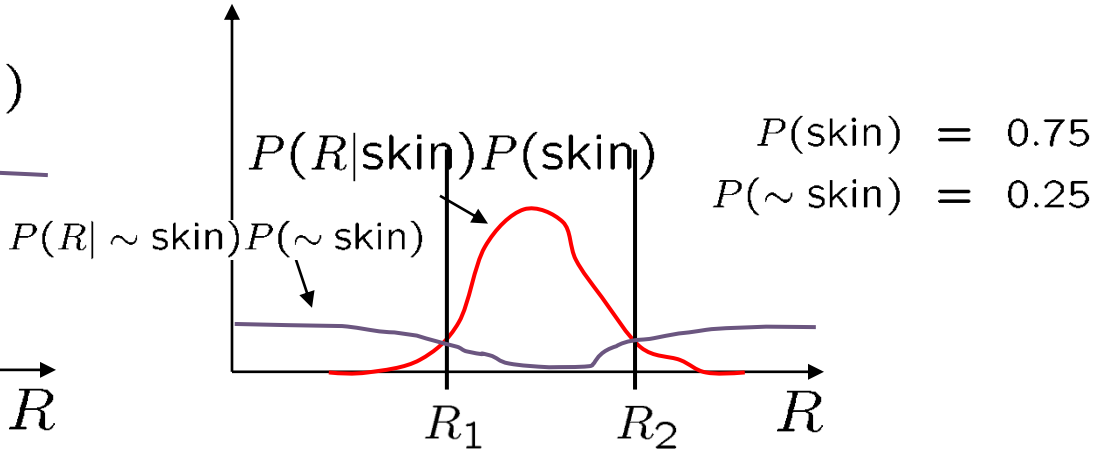
What can we use for the prior P(skin)?

- Domain knowledge:
  - P(skin) may be larger if we know the image contains a person
  - For a portrait, P(skin) may be higher for pixels in the center
- Learn the prior from the training set. How?
  - P(skin) is proportion of skin pixels in training set

Bayesian estimation



likelihood



posterior (unnormalized)

- Bayesian estimation
  - Goal is to choose the label (skin or ~skin) that maximizes the posterior ↔ minimizes probability of misclassification
    - this is called **Maximum A Posteriori (MAP) estimation**

Skin detection results

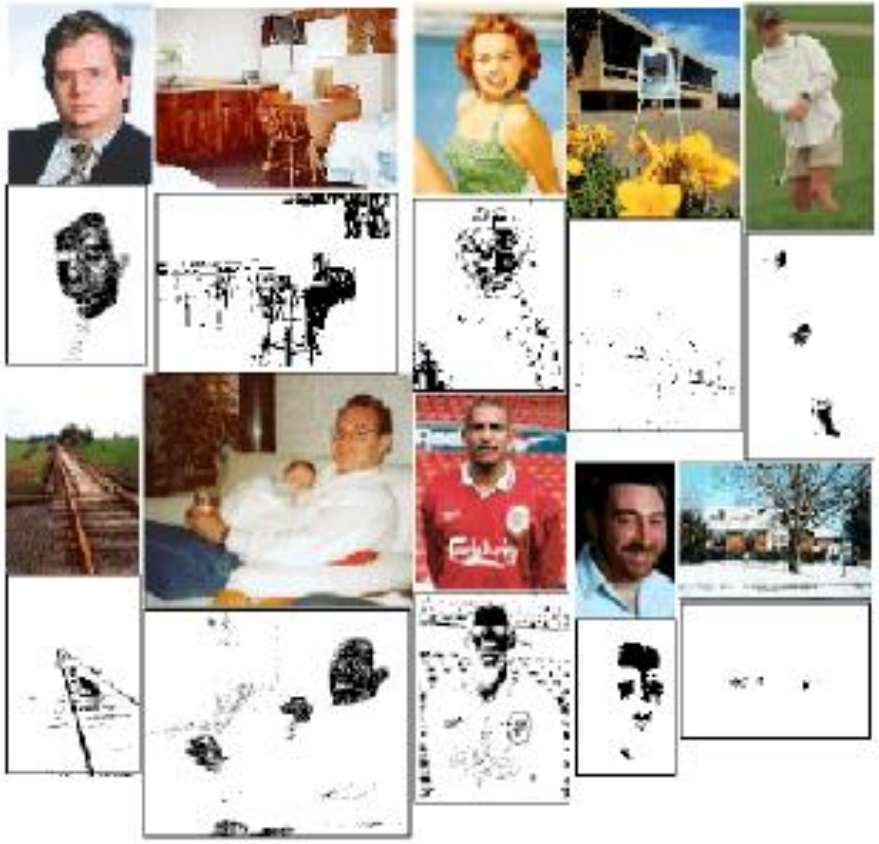
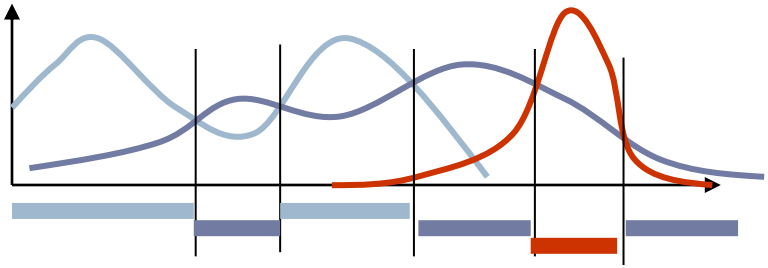


Figure 25.3. The figure shows a variety of images together with the output of the skin detector of Jones and Rehg applied to the image. Pixels marked black are skin pixels, and white are background. Notice that this process is relatively effective, and could certainly be used to focus attention on, say, faces and hands. *Figure from "Statistical color models with application to skin detection," M.J. Jones and J. Rehg, Proc. Computer Vision and Pattern Recognition, 1999 © 1999, IEEE*

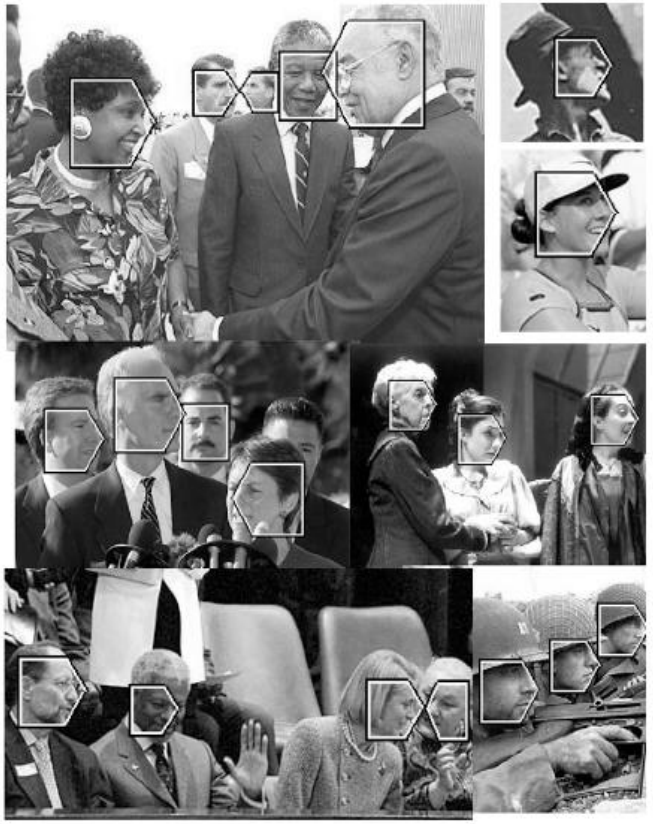
General classification

- This same procedure applies in more general circumstances
  - More than two classes
  - More than one dimension



Example: face detection

- Here, X is an image region
  - dimension = # pixels
  - each face can be thought of as a point in a high dimensional space

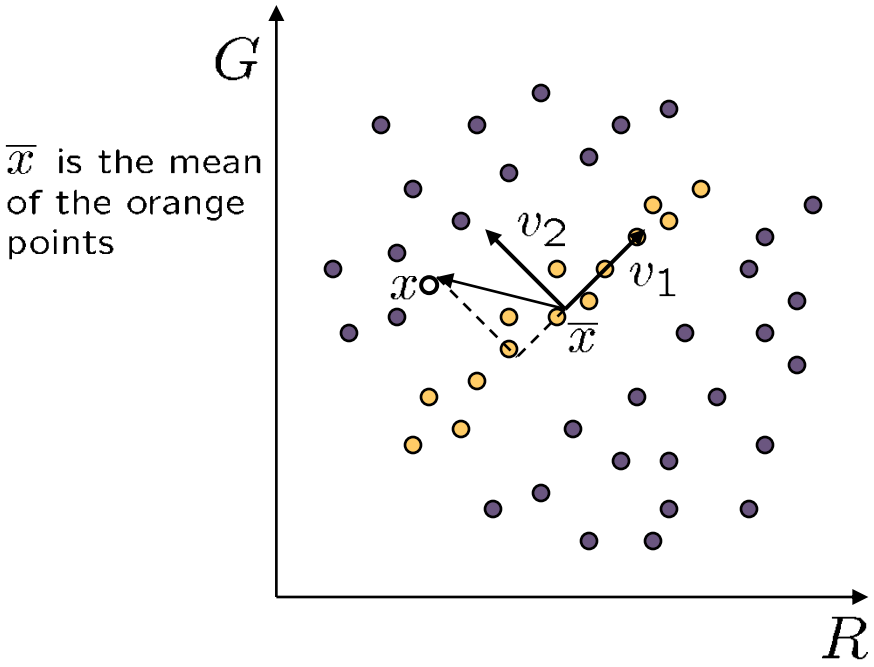


H. Schneiderman, T. Kanade. "A Statistical Method for 3D Object Detection Applied to Faces and Cars". CVPR 2000

Eigenfaces for recognition

Matthew Turk and Alex Pentland, *J. Cognitive Neuroscience*, 1991

Linear subspaces



convert **x** into $\mathbf{v_1}$, $\mathbf{v_2}$ coordinates

$$\mathbf{x} \rightarrow ((\mathbf{x} - \overline{x}) \cdot \mathbf{v_1}, (\mathbf{x} - \overline{x}) \cdot \mathbf{v_2})$$

What does the $\mathbf{v_2}$ coordinate measure?

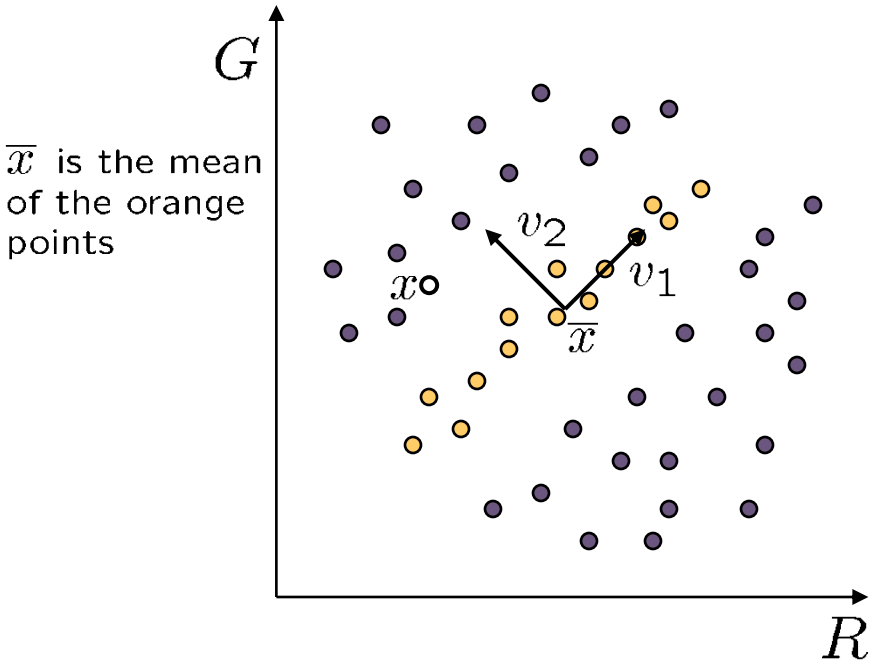
- distance to line
- use it for classification—near 0 for orange pts

What does the $\mathbf{v_1}$ coordinate measure?

- position along line
- use it to specify which orange point it is

- Classification can be expensive:
  - Big search prob (e.g., nearest neighbors) or store large PDF's
- Suppose the data points are arranged as above
  - Idea—fit a line, classifier measures distance to line

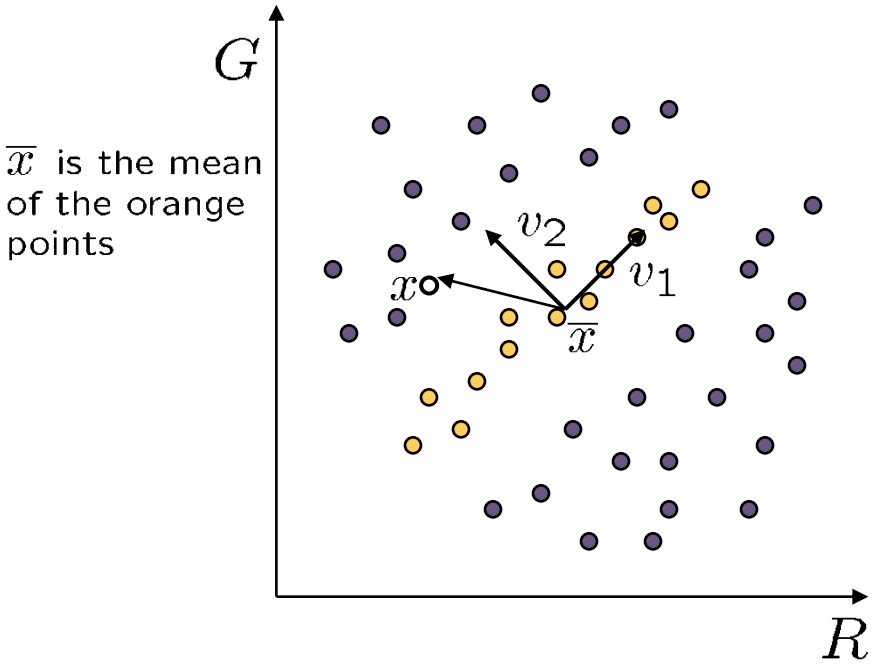
Dimensionality reduction



Dimensionality reduction

- We can represent the orange points with *only* their $\mathbf{v}_1$ coordinates (since $\mathbf{v}_2$ coordinates are all essentially 0)
- This makes it much cheaper to store and compare points
- A bigger deal for higher dimensional problems

Linear subspaces



Consider the variation along direction **v** among all of the orange points:

$$var(\mathbf{v}) = \sum_{\text{orange point } \mathbf{x}} \|(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}} \cdot \mathbf{v}\|^2$$

What unit vector **v** minimizes *var*?

$$\mathbf{v}_2 = min_{\mathbf{v}} \{var(\mathbf{v})\}$$

What unit vector **v** maximizes *var*?

$$\mathbf{v}_1 = max_{\mathbf{v}} \{var(\mathbf{v})\}$$

$$\begin{aligned} var(\mathbf{v}) &= \sum_{\mathbf{x}} \|(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}} \cdot \mathbf{v}\| \\ &= \sum_{\mathbf{x}} \mathbf{v}^{\mathrm{T}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}} \mathbf{v} \\ &= \mathbf{v}^{\mathrm{T}} \left[\sum_{\mathbf{x}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}} \right] \mathbf{v} \\ &= \mathbf{v}^{\mathrm{T}} \mathbf{A} \mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathrm{T}} \end{aligned}$$

Solution: $\mathbf{v}_1$ is eigenvector of **A** with *largest* eigenvalue
$\mathbf{v}_2$ is eigenvector of **A** with *smallest* eigenvalue

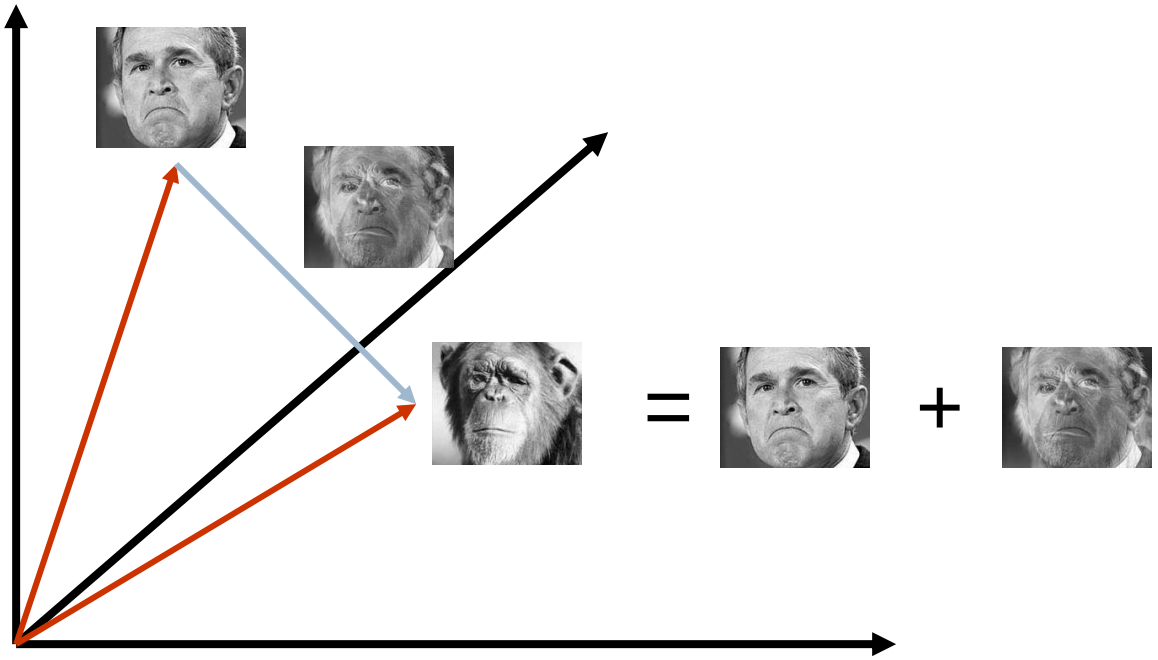
Principal component analysis

- Suppose each data point is N-dimensional
  - Same procedure applies:

$$
\begin{aligned}
var(\mathbf{v}) &= \sum_{\mathbf{x}} \|(\mathbf{x}-\bar{\mathbf{x}})^{\mathrm{T}} \cdot \mathbf{v}\| \\
&= \mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}} (\mathbf{x}-\bar{\mathbf{x}})(\mathbf{x}-\bar{\mathbf{x}})^{\mathrm{T}}
\end{aligned}
$$

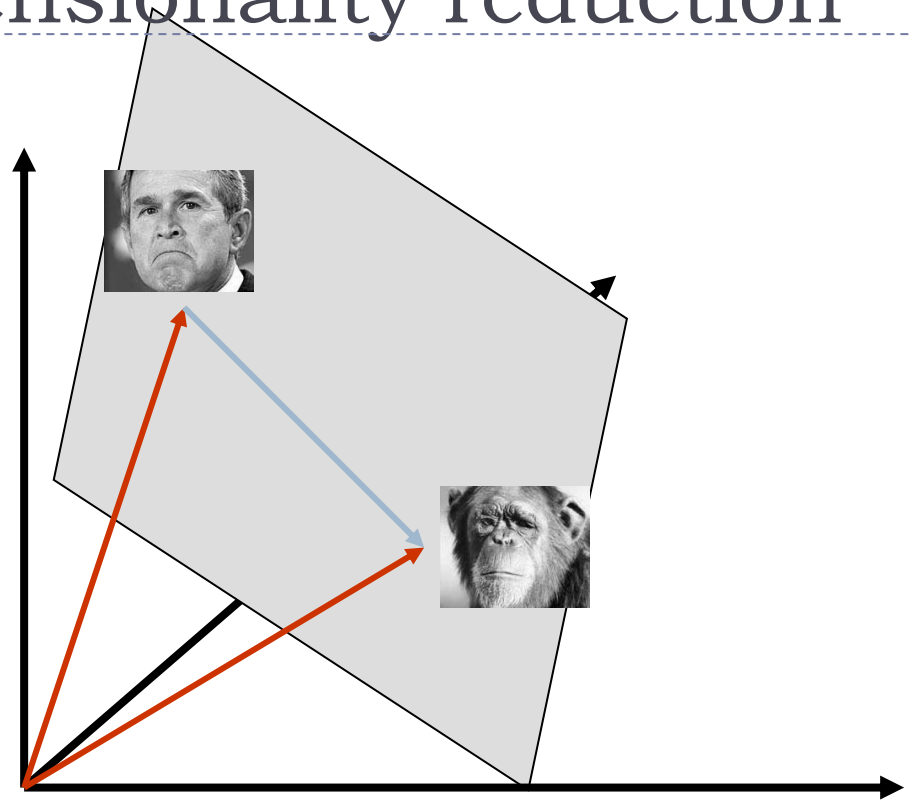
  - The eigenvectors of **A** define a new coordinate system
    - eigenvector with largest eigenvalue captures the most variation among training vectors **x**
    - eigenvector with smallest eigenvalue has least variation
  - We can compress the data using the top few eigenvectors
    - corresponds to choosing a "linear subspace"
      - represent points on a line, plane, or "hyper-plane"
    - these eigenvectors are known as the ***principal components***

The space of faces



- An image is a point in a high dimensional space
  - An N x M image is a point in R^{NM}
  - We can define vectors in this space as we did in the 2D case

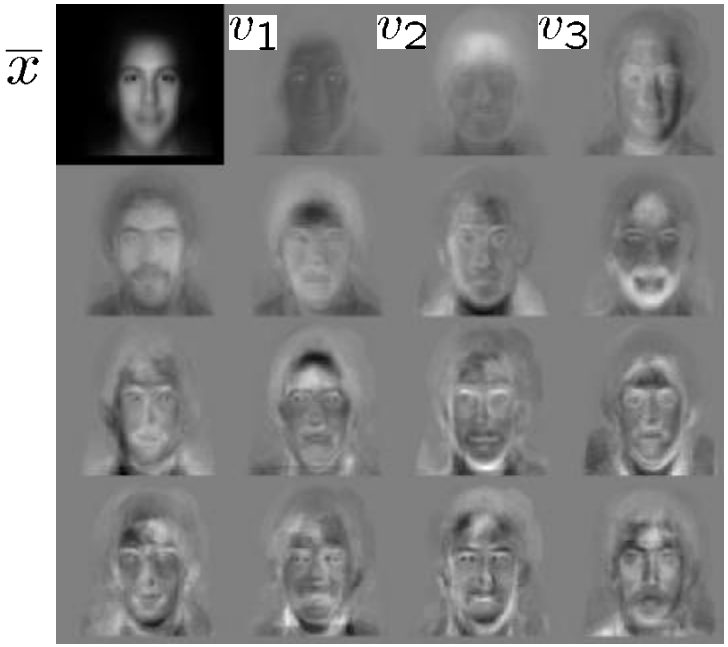
Dimensionality reduction



- The set of faces is a “subspace” of the set of images
  - We can find the best subspace using PCA
  - This is like fitting a “hyper-plane” to the set of faces
    - spanned by vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$
    - any face $\mathbf{x} \approx \bar{\mathbf{x}} + a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_k\mathbf{v}_k$

Eigenfaces

- PCA extracts the eigenvectors of **A**
  - Gives a set of vectors v_1, v_2, v_3, ...
  - Each vector is a direction in face space
    - what do these look like?

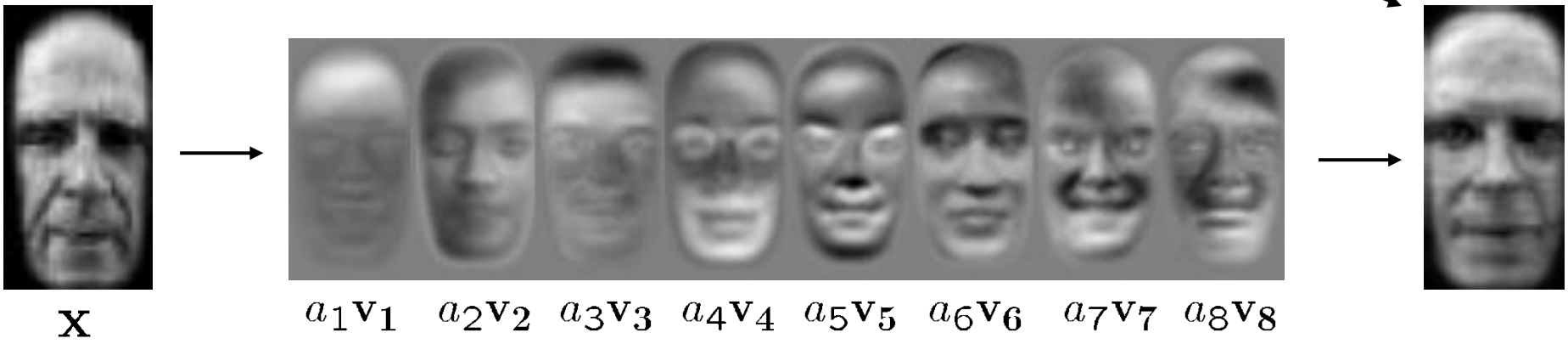


Projecting onto the eigenfaces

- The eigenfaces $\mathbf{v}_1, \ldots, \mathbf{v}_K$ span the space of faces
  - A face is converted to eigenface coordinates by

$$\mathbf{x} \rightarrow (\underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v}_1}_{a_1}, \underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v}_2}_{a_2}, \ldots, \underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v}_K}_{a_K})$$

$$\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_K\mathbf{v}_K$$



$\mathbf{x}$ $\quad a_1\mathbf{v}_1 \quad a_2\mathbf{v}_2 \quad a_3\mathbf{v}_3 \quad a_4\mathbf{v}_4 \quad a_5\mathbf{v}_5 \quad a_6\mathbf{v}_6 \quad a_7\mathbf{v}_7 \quad a_8\mathbf{v}_8$

Recognition with eigenfaces

- Algorithm
  1. Process the image database (set of images with labels)
     - Run PCA—compute eigenfaces
     - Calculate the K coefficients for each image
  2. Given a new image (to be recognized) **x**, calculate K coefficients

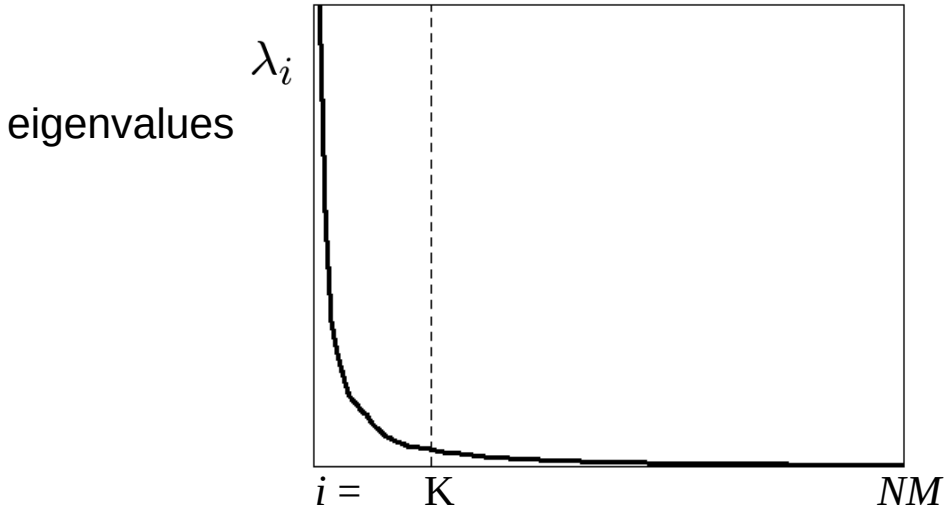
$$\mathbf{x} \rightarrow (a_1, a_2, \ldots, a_K)$$

  3. Detect if x is a face

$$\|\mathbf{x} - (\bar{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_K\mathbf{v_K})\| < \text{threshold}$$

  4. If it is a face, who is it?
     - Find closest labeled face in database
       - nearest-neighbor in K-dimensional space

Choosing the dimension K



- How many eigenfaces to use?
- Look at the decay of the eigenvalues
  - the eigenvalue tells you the amount of variance “in the direction” of that eigenface
  - ignore eigenfaces with low variance

Eigenface algorithm

- Mathematics:

Given a training set of N faces $\mathbf{X} \in \Re^{D \times N}$, D is the dimension.

Construct covariance matrix

$$\mathbf{C} = (\mathbf{X} - \bar{\mathbf{X}})(\mathbf{X} - \bar{\mathbf{X}})^{\mathrm{T}} \in \Re^{D \times D}$$

If *D* is too large, the Eigen-decomposition of C will be very difficult.

For example, if a face is 256x256, the dimension D=65536.

How to solve this problem?

If N is smaller than D, the covariance matrix can be computed as

$$\mathbf{C}' = (\mathbf{X} - \bar{\mathbf{X}})^{\mathrm{T}}(\mathbf{X} - \bar{\mathbf{X}}) \in \Re^{N \times N}$$

Equivalent?

Eigenface algorithm

- Mathematics:

Let $\mathbf{\Phi} = \mathbf{X} - \overline{\mathbf{X}}$,

$$\mathbf{C} = \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}$$

and

$$\mathbf{C}' = \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}$$

There is

$$\begin{gathered}\mathbf{C}'\mathbf{p}_i = \lambda\mathbf{p}_i \\ \rightarrow \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{p}_i = \lambda\mathbf{p}_i \\ \rightarrow \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{p}_i = \lambda\mathbf{\Phi}\mathbf{p}_i \\ \rightarrow \mathbf{C}\mathbf{\Phi}\mathbf{p}_i = \lambda\mathbf{\Phi}\mathbf{p}_i \\ \rightarrow \mathbf{C}\mathbf{v}_i = \lambda\mathbf{v}_i\end{gathered}$$

where$\mathbf{v}_i = \mathbf{\Phi}\mathbf{p}_i$

That is, when the eigen-vector $\mathbf{p}_i$ is solved, a $\mathbf{\Phi}$ should be multiplied with $\mathbf{\Phi} = \mathbf{X} - \overline{\mathbf{X}}$

Bayesian Face Recognition

- Baback Moghaddam, Tony Jebara and Alex Pentland
- *Pattern Recognition*
- 33(11), 1771-1782, November 2000

Bayesian Face Recognition

Intrapersonal Ω_{I}

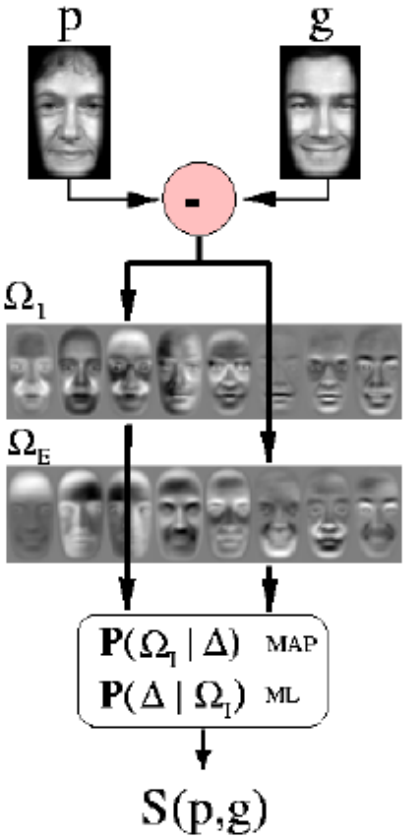
Extrapersonal Ω_{E}

$$\Omega_{\mathrm{I}} \equiv \{\Delta = x_i - x_j : L(x_i) = L(x_j)\}$$

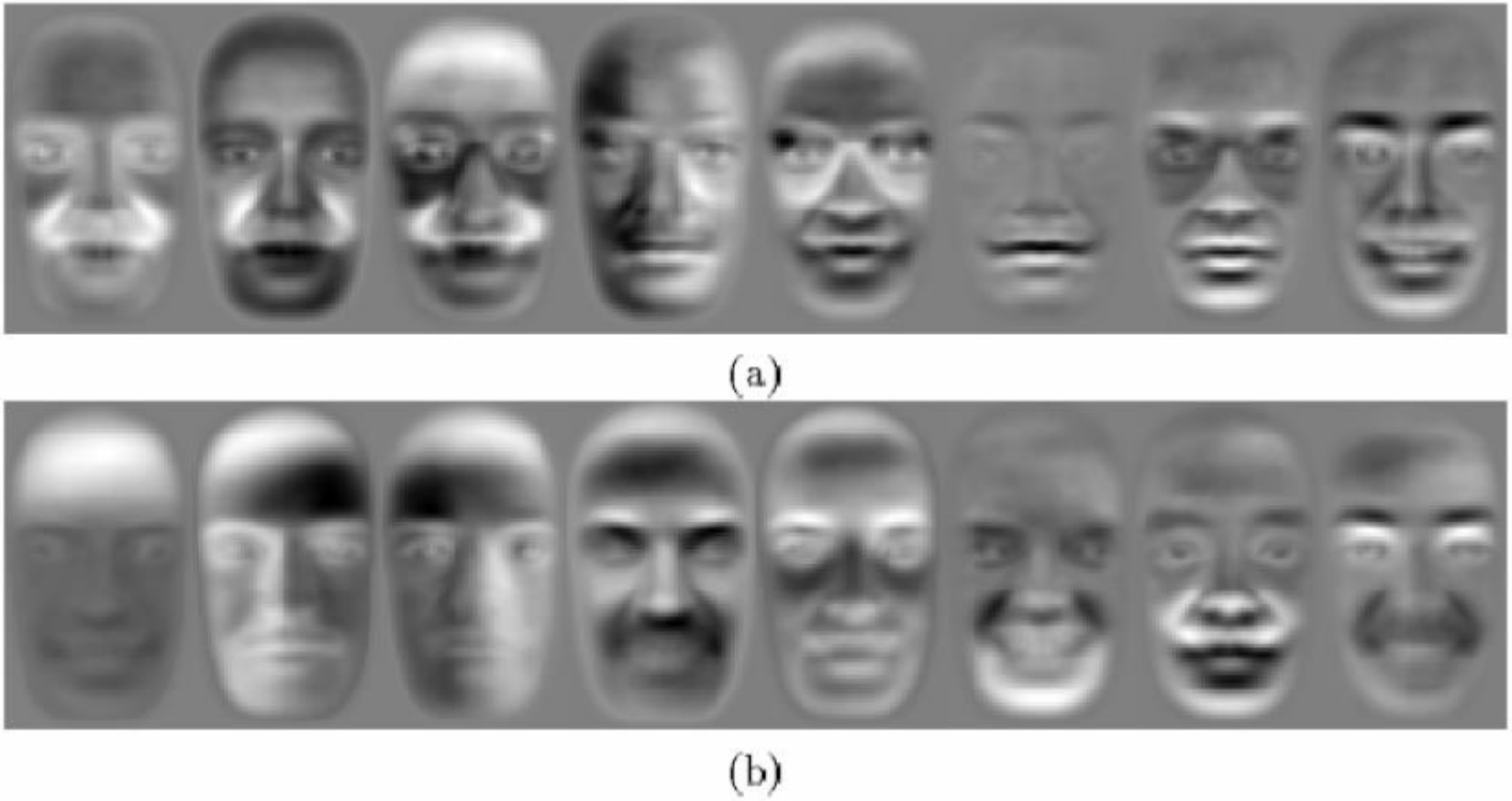
$$\Omega_{\mathrm{E}} \equiv \{\Delta = x_i - x_j : L(x_i) \neq L(x_j)\}$$

$$S = \frac{P(\Delta \mid \Omega_{\mathrm{I}})P(\Omega_{\mathrm{I}})}{P(\Delta \mid \Omega_{\mathrm{I}})P(\Omega_{\mathrm{I}}) + P(\Delta \mid \Omega_{\mathrm{E}})P(\Omega_{\mathrm{E}})}$$

$P(\Delta \mid \Omega) \longrightarrow$ [Moghaddam ICCV'95]



Bayesian Face Recognition

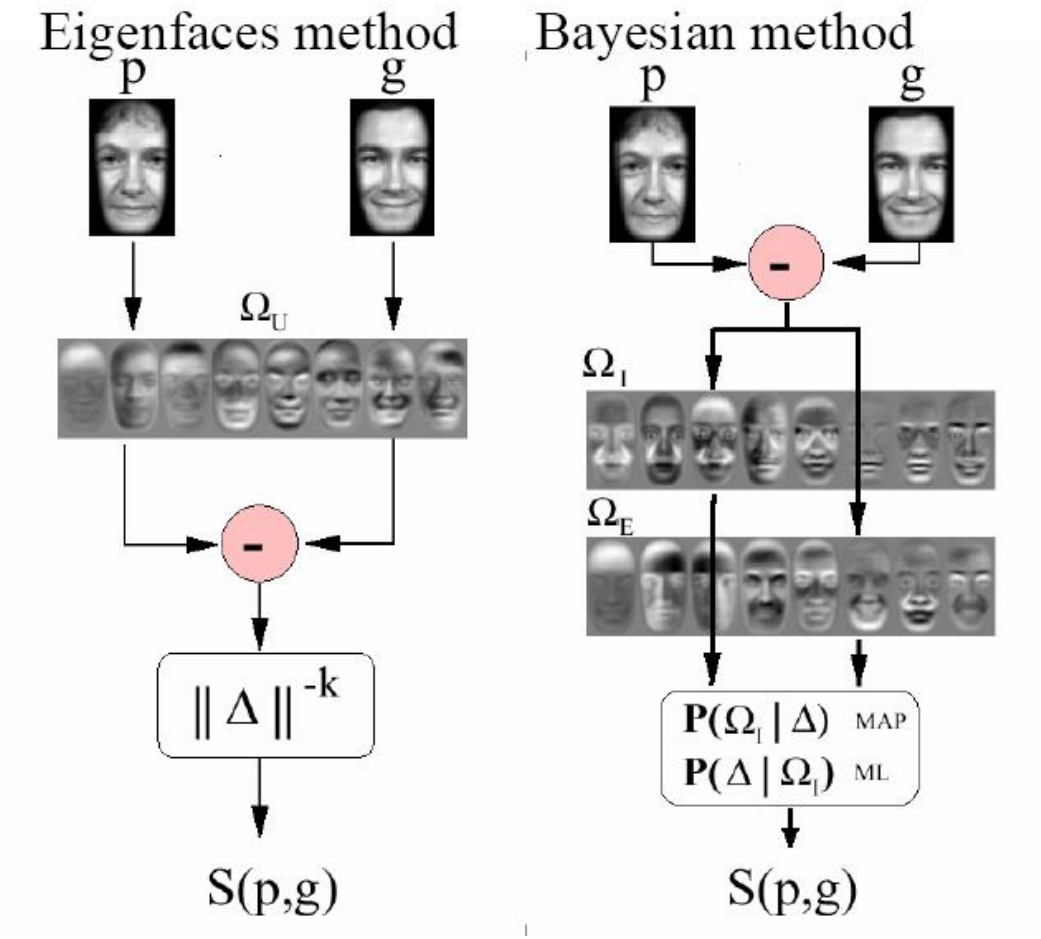


(a)

(b)

Figure 6: "Dual" Eigenfaces: (a) Intrapersonal, (b) Extrapersonal

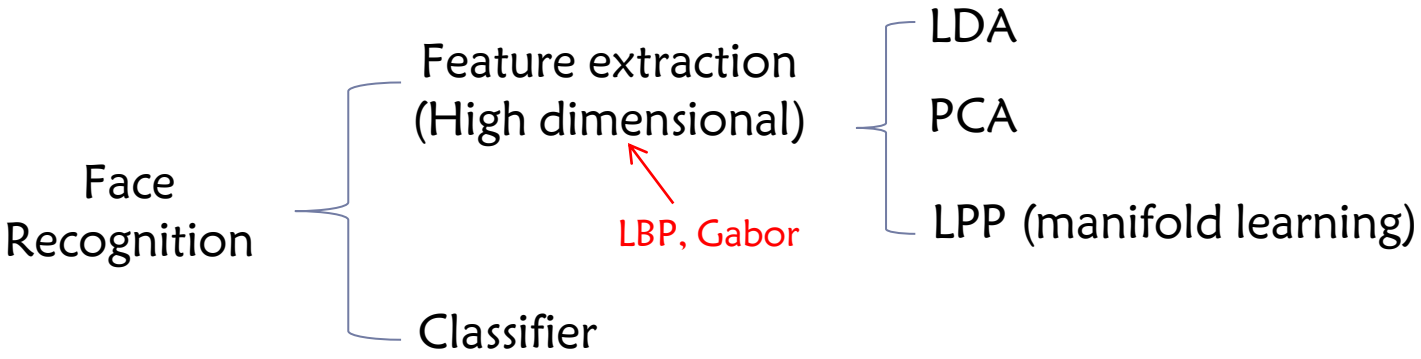
Bayesian Face Recognition



Face Recognition

- General routes

Narrowly,



Face Recognition

- Face images

(a) Session 1

(b) Session 2

Unconstrained faces

Face Recognition

- Labeled Face in the Wild
  - Face recognition in unconstrained condition
  - Data sampling from Internet
  - Verification task

13,233 face images from 5749 persons

Standard protocol:

6000 pairs with 10-folds.

300 similar pairs and 300 Dissimilar pairs per fold.

Face Recognition

Metric Learning

Metric learning aims at learning a distance metric **M**, such that the distance between similar samples is smaller than that between dissimilar samples.

- Large margin nearest neighbor (LMNN) classification

Let $\{\vec{x}_i, y_i\}_{i=1}^n$ denote a training set, a binary matrix $y_{ij} \in \{0,1\}$ is used to indicate whether or not the labels y_i and y_j match.

Goal: learn a linear transformation L.

The square distance is computed as

$$\mathcal{D}(\vec{x}_i, \vec{x}_j) = \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2$$

Face Recognition

- Large margin nearest neighbor (LMNN) classification

Define a binary matrix $\eta_{ij} \in \{0,1\}$, to indicate whether input $\vec{x}_i$ is a neighbor of input $\vec{x}_j$.

$$\varepsilon(\mathbf{L}) = \sum_{ij} \eta_{ij} \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 + c \sum_{ijl} \eta_{ij}(1 - y_{il}) \left[1 + \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 - \|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|^2\right]_+$$

where $[z]_+ = \max(z, 0)$ denotes the hinge loss.

第一项惩罚输入与其近邻之间较大的距离。

第二项惩罚输入与其非近邻之间较小的距离。

Face Recognition

- Large margin nearest neighbor (LMNN) classification

$$\mathcal{D}(\vec{x}_i, \vec{x}_j) = \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2$$

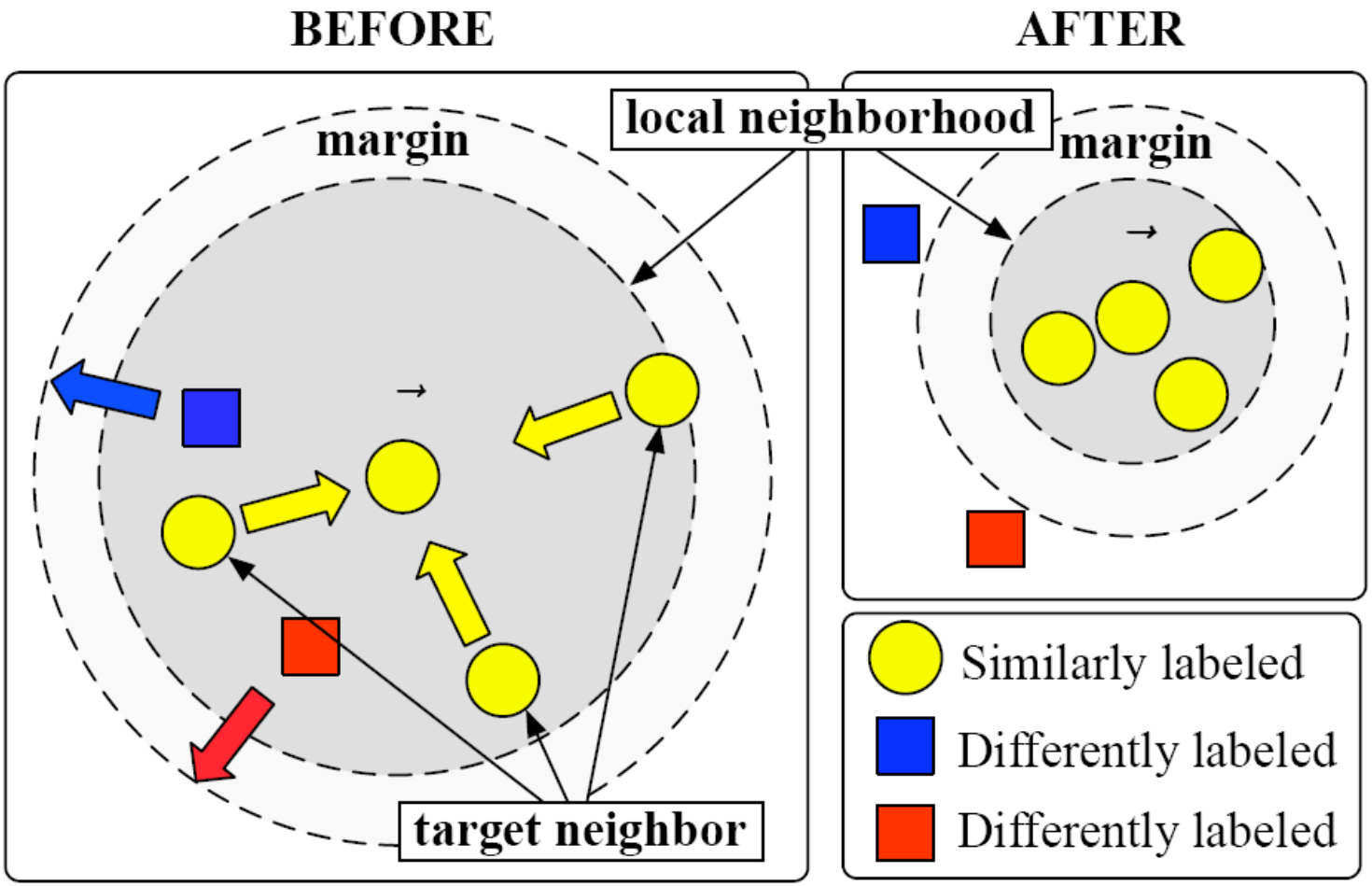
$$\mathcal{D}(\vec{x}_i, \vec{x}_j) = (\vec{x}_i - \vec{x}_j)^\top \mathbf{M}(\vec{x}_i - \vec{x}_j)$$

where $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$

Mahalanobis distance metric

Face Recognition

- Large margin nearest neighbor (LMNN) classification



Face Recognition

- Large margin nearest neighbor (LMNN) classification

By introducing slack variable ξ_{ij}, the model is formulated as

Minimize $\sum_{ij}\eta_{ij}(\vec{x}_i-\vec{x}_j)^\top\mathbf{M}(\vec{x}_i-\vec{x}_j)+c\sum_{ij}\eta_{ij}(1-y_{il})\xi_{ijl}$ **subject to:**

(1) $(\vec{x}_i-\vec{x}_l)^\top\mathbf{M}(\vec{x}_i-\vec{x}_l)-(\vec{x}_i-\vec{x}_j)^\top\mathbf{M}(\vec{x}_i-\vec{x}_j)\ \geq\ 1-\xi_{ijl}$

(2) $\xi_{ijl}\geq 0$

(3) $\mathbf{M}\succeq 0$.

Semidefinite program 半正定规划问题求解

$$\varepsilon(\mathbf{L})=\sum_{ij}\eta_{ij}\|\mathbf{L}(\vec{x}_i-\vec{x}_j)\|^2+c\sum_{ijl}\eta_{ij}(1-y_{il})\left[1+\|\mathbf{L}(\vec{x}_i-\vec{x}_j)\|^2-\|\mathbf{L}(\vec{x}_i-\vec{x}_l)\|^2\right]_+$$