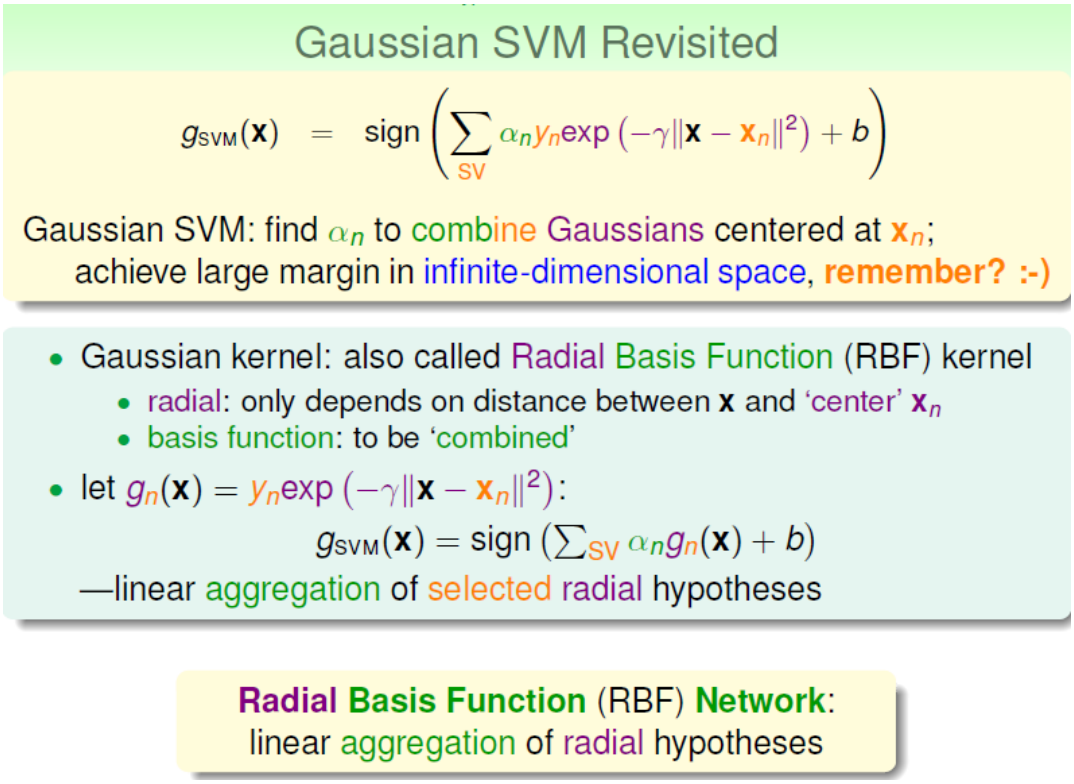
Radial Basis Function Network

一、RBF Network Hypothesis

1. 回顾一下kernel SVM：我们在无限维度的空间中找到large margin然后用α_n来结合这些以x_n为中心的高斯函数

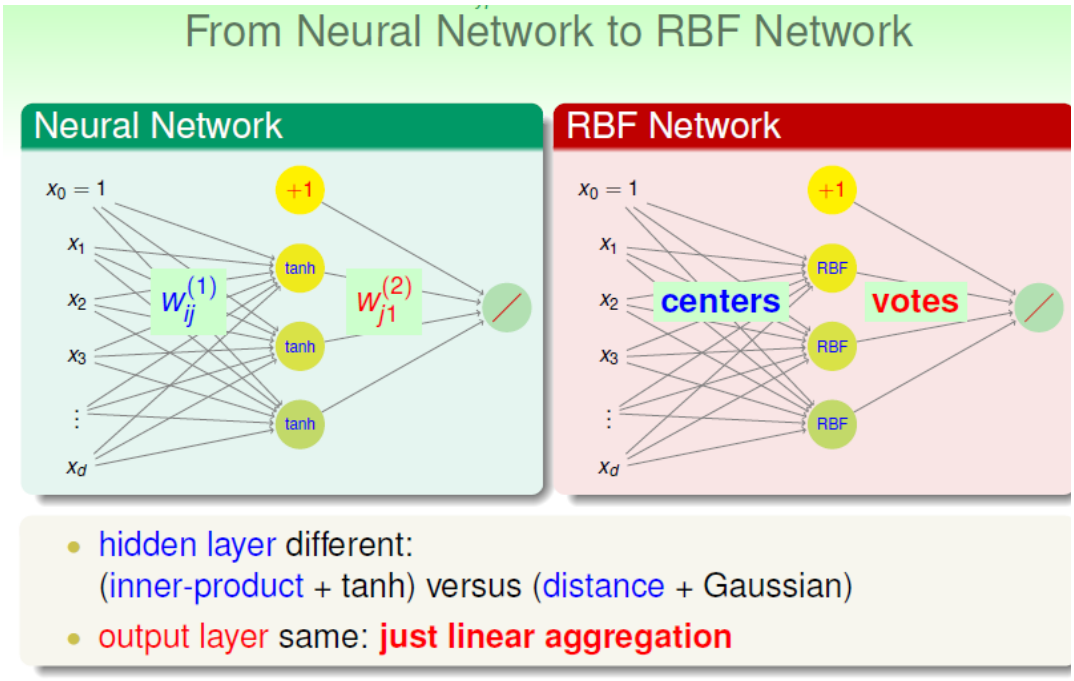


高斯核是径向基的一种，所谓径向基包含两个部分：

- radial: 径向就是半径，其实就是一个欧氏距离的度量
- basis function：就是combine这些东西在一起，可以想象成是一种aggregation model

RBF Network: linear aggregation of radial hypothesis

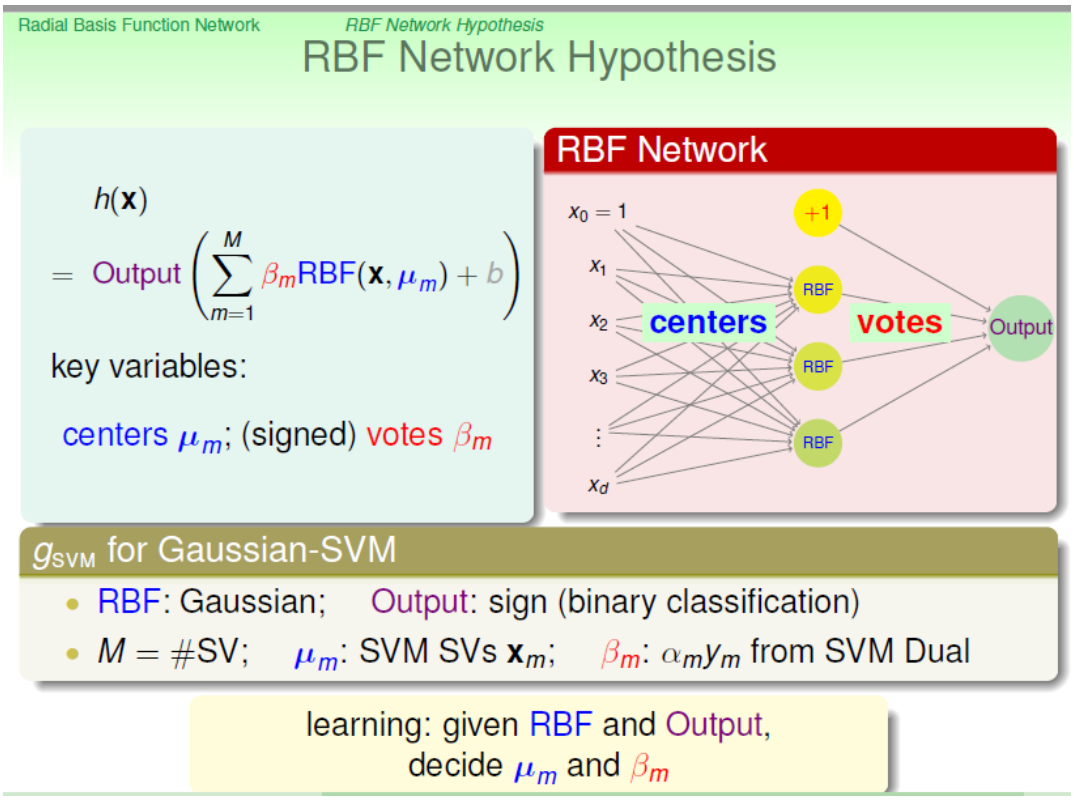
2. 对比一下NN 和 RBF Network:



hidden layer不一样，神经网络是tanh作non-linear的内积层，RBF网络是用这些输入作为center的aggregation vote

3. 我们设计一下RBF的一般结构如下:

投票的权重：β_m，RBF center：μ_m



回顾一下我们在SVM中做的事情，我们是直接把$x_1, x_2, \ldots$当作了center，然后把他们的输出y_1, y_2作为了votes。一般的RBF网络不同，我们需要学习center 和 votes：

我们首先理解一下RBF Network做的事情，其实就是学一些距离之间的相似关系，就类似于kernel Z域内积的相似性，RBF是X域和中心的相似性（距离度量）他们都受限于Mercer's Condition

二、RBF Network Learning

1. 看一下Full RBF Network model它就是把所有的数据都拿来当作中心。其实这个是一个偷懒的方式而且有的时候效果并不是特别好。

Full RBF Network

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \mu_m)\right)$$

- full RBF Network: $M = N$ and each $\mu_m = \mathbf{x}_m$
- physical meaning: each $\mathbf{x}_m$ **influences** similar $\mathbf{x}$ by β_m
- e.g. uniform influence with $\beta_m = 1 \cdot y_m$ for binary classification

$$g_{\text{uniform}}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{N} y_m \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_m\|^2\right)\right)$$

—**aggregate** each example's opinion subject to similarity

full RBF Network: **lazy** way to decide μ_m

它其实类似于K-nearest neighbor算法：selection而不是aggregate，因为高斯函数的波动比较大，往往最近的那个就dominate所有的结果了。这个时候aggregate就变成了selection。

Nearest Neighbor

$$g_{\text{uniform}}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{N} y_m \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_m\|^2\right)\right)$$

- $\exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_m\|^2\right)$: maximum when $\mathbf{x}$ **closest to** $\mathbf{x}_m$
  —maximum one often dominates the $\sum_{m=1}^{N}$ term
- take y_m of **maximum** $\exp(\ldots)$ instead of voting of all y_m
  —**selection** instead of **aggregation**
- physical meaning:

  $$g_{\text{nbor}}(\mathbf{x}) = y_m \text{ such that } \mathbf{x} \textbf{ closest to } \mathbf{x}_m$$

  —called **nearest neighbor** model
- can **uniformly aggregate** k neighbors also: k **nearest neighbor**

k nearest neighbor:
also **lazy** but **very intuitive**

2. Full RBF Network里面的插值操作：我们需要对这个模型作MSE的最小化：

$$h(x) = \left(\sum_{m=1}^{N} \beta_m RBF(x, x_m)\right) \tag{1}$$

首先我们定义一个matrix：

$$z_n = [RBF(x_n, x_1), RBF(x_n, x_2), \ldots, RBF(x_n, x_N)] \tag{2}$$

类似于一个transformation，那么根据之前学过的linear regression的知识，我们直到如果$Z^T Z$是可逆的，那么就有

$$\beta = (Z^T Z)^{-1} Zy \tag{3}$$

下面我们证明x_n不同的情况下Z可逆：

因为Gaussian Kernel Matrix是对称的，所以说行列对应相等。

于是我们可以继续化简：

$$\beta = Z^{-1} y \tag{4}$$

So Easy!

Interpolation by Full RBF Network

full RBF Network for squared error regression:

$$h(\mathbf{x}) = \cancel{\text{Output}}\left(\sum_{m=1}^{N} \beta_m \text{RBF}(\mathbf{x}, \mathbf{x}_m)\right)$$

- just linear regression on RBF-transformed data

$$\mathbf{z}_n = [\text{RBF}(\mathbf{x}_n, \mathbf{x}_1), \text{RBF}(\mathbf{x}_n, \mathbf{x}_2), \ldots, \text{RBF}(\mathbf{x}_n, \mathbf{x}_N)]$$

- optimal β? $\beta = (Z^T Z)^{-1} Z^T \mathbf{y}$, if $Z^T Z$ invertible, **remember? :-)**
- size of Z? N (examples) by N (centers)
  —symmetric square matrix
- theoretical fact: if $\mathbf{x}_n$ all different, Z with Gaussian RBF **invertible**

optimal β with invertible Z: $\beta = Z^{-1}\mathbf{y}$

3. Regularized Full RBF Network

我们看一下之前的结果：

Regularized Full RBF Network

full Gaussian RBF Network for regression: $\beta = Z^{-1}\mathbf{y}$

$$g_{\text{RBF}}(\mathbf{x}_1) = \beta^T \mathbf{z}_1 = \mathbf{y}^T Z^{-1} (\text{first column of } Z) = \mathbf{y}^T \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix}^T = y_1$$

—$g_{\text{RBF}}(\mathbf{x}_n) = y_n$, i.e. $E_{\text{in}}(g_{\text{RBF}}) = 0$, **yeah!! :-)**

- called **exact interpolation** for **function approximation**
- but **overfitting for learning? :-(**
- how about **regularization**? e.g. **ridge** regression for β instead
  —optimal $\beta = (Z^T Z + \lambda I)^{-1} Z^T \mathbf{y}$
- seen Z? $Z = [\text{Gaussian}(\mathbf{x}_n, \mathbf{x}_m)]$ = Gaussian kernel matrix K

这个完美的插值做的太好了！这样似乎很容易过拟合。。。为了防止过拟合，我们加入正则项，类比之前的脊回归：

$$\beta = \left(Z^T Z + \lambda I\right)^{-1} Z^T y \tag{5}$$

我们注意到其实Z就是高斯核矩阵，对比一下各种空间里面的正则化机制：

effect of regularization in different spaces:

kernel ridge regression: $\beta = (\mathrm{K} + \lambda \mathrm{I})^{-1}\mathbf{y}$;
regularized full RBFNet: $\beta = (\mathrm{Z}^T\mathrm{Z} + \lambda \mathrm{I})^{-1}\mathrm{Z}^T\mathbf{y}$

4. 我们还可以通过减少中心来减少模型复杂度，类似于SVM中只考虑SV的作用：

Fewer Centers as Regularization

recall:

$$g_{\mathrm{SVM}}(\mathbf{x}) = \mathrm{sign}\left(\sum_{\mathrm{SV}} \alpha_m y_m \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_m\|^2\right) + b\right)$$

—only '$\ll N$' SVs needed in 'network'

- next: $M \ll N$ instead of $M = N$
- effect: **regularization**
  by constraining **number of centers** and **voting weights**
- physical meaning of centers μ_m: **prototypes**

remaining question:
how to extract **prototypes**?

这里我们就需要考虑如何减少这些centers（原型）接下来的算法就可以解决这个问题。

三、K-Means Algorithm

1. 我们考虑这样一件事情，如果两个输入的x足够接近那么实际上我们就不需要把他们两个都作为center，基于距离的运算让我们可以根据距离把数据划分成几个cluster从而作为prototype

   因此我们需要作两件事情：

   - 对于X进行分类
   - 为每一个类找到一个中心

Good Prototypes: Clustering Problem

if $\mathbf{x}_1 \approx \mathbf{x}_2$,
$\Longrightarrow$ **no need** both RBF$(\mathbf{x}, \mathbf{x}_1)$ & RBF$(\mathbf{x}, \mathbf{x}_2)$ in RBFNet,
$\Longrightarrow$ **cluster** $\mathbf{x}_1$ and $\mathbf{x}_2$ by **one prototype** $\mu \approx \mathbf{x}_1 \approx \mathbf{x}_2$

- **clustering** with **prototype**:
  - **partition** $\{\mathbf{x}_n\}$ to disjoint sets $S_1, S_2, \cdots, S_M$
  - **choose** μ_m for each S_m

  —hope: $\mathbf{x}_1, \mathbf{x}_2$ both $\in S_m \Leftrightarrow \mu_m \approx \mathbf{x}_1 \approx x_2$
- cluster error with squared error measure:

$$E_{\text{in}}(S_1, \cdots, S_M; \mu_1, \cdots, \mu_M) = \frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \mu_m\|^2$$

goal: with $S_1, \cdots, S_M$ being a partition of $\{\mathbf{x}_n\}$,

$$\min_{\{S_1, \cdots, S_M; \mu_1, \cdots, \mu_M\}} E_{\text{in}}(S_1, \cdots, S_M; \mu_1, \cdots, \mu_M)$$

2. 下面我们来最优化这个问题。

Partition Optimization

with $S_1, \cdots, S_M$ being a partition of $\{\mathbf{x}_n\}$,

$$\min_{\{S_1, \cdots, S_M; \mu_1, \cdots, \mu_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \mu_m\|^2$$

- **hard to optimize**: joint combinatorial-numerical optimization
- **two sets** of variables: will optimize alternatingly

这是一个两组变量的优化问题，突然想到多年以前中学时代的调整大法（固定一些，调整一些）这人里我们轮流调整。

- 固定中心，找到和中心最近的点归入那一类

if $\mu_1, \cdots, \mu_M$ **fixed**, for each $\mathbf{x}_n$

- $[\![\mathbf{x}_n \in S_m]\!]$: choose one and only one subset
- $\|\mathbf{x}_n - \mu_m\|^2$: distance to each prototype

optimal chosen subset $S_m =$ the one with **minimum** $\|\mathbf{x}_n - \mu_m\|^2$

for given $\mu_1, \cdots, \mu_M$, each $\mathbf{x}_n$
'optimally partitioned' using its **closest** μ_m

- 固定分类，让center是这个类的平均值

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each μ_m

$$\nabla_{\mu_m} E_{\text{in}} = -2 \sum_{n=1}^{N} [\![\mathbf{x}_n \in S_m]\!] (\mathbf{x}_n - \mu_m) = -2 \left(\left(\sum_{\mathbf{x}_n \in S_m} \mathbf{x}_n \right) - |S_m| \mu_m \right)$$

optimal prototype $\mu_m =$ **average** of $\mathbf{x}_n$ within S_m

for given $S_1, \cdots, S_M$, each μ_n
'optimally computed' as **consensus** within S_m

3. 总结一下K-means算法的流程：

   搞清楚两件事情：有头有尾

   - 头：随机初始化centers（从x里面挑）
   - 尾：当每个类别里面的元素基本上不动了就停止（给足够的训练量）

k-Means Algorithm

use k **prototypes** instead of M historically
(different from k nearest neighbor, though)

k-Means Algorithm

1. initialize $\mu_1, \mu_2, \ldots, \mu_k$: say, as k randomly chosen $\mathbf{x}_n$
2. **alternating optimization** of E_{in}: repeatedly
   1. optimize $S_1, S_2, \ldots, S_k$:
      each $\mathbf{x}_n$ 'optimally partitioned' using its closest μ_i
   2. optimize $\mu_1, \mu_2, \ldots, \mu_k$:
      each μ_n 'optimally computed' as consensus within S_m

   until **converge**

converge: no change of $S_1, S_2, \ldots, S_k$ anymore
—guaranteed as E_{in} **decreases** during alternating minimization

k-Means: the most popular **clustering**
algorithm through **alternating minimization**

4. 把K-Means算法用到RBF网络里面：

RBF Network Using k-Means

RBF Network Using k-Means

1. run k-Means with $k = M$ to get $\{\mu_m\}$
2. construct transform $\mathbf{\Phi}(\mathbf{x})$ from RBF (say, Gaussian) at μ_m

$$\mathbf{\Phi}(\mathbf{x}) = [\text{RBF}(\mathbf{x}, \mu_1), \text{RBF}(\mathbf{x}, \mu_2), \ldots, \text{RBF}(\mathbf{x}, \mu_M)]$$

3. run linear model on $\{(\mathbf{\Phi}(\mathbf{x}_n), y_n)\}$ to get β
4. return $g_{\text{RBFNET}}(\mathbf{x}) = \text{LinearHypothesis}(\beta, \mathbf{\Phi}(\mathbf{x}))$

- using unsupervised learning (k-Means) to assist **feature transform**—like **autoencoder**
- parameters: M (prototypes), RBF (such as γ of Gaussian)

RBF Network: a simple (**old-fashioned**) model

不过RBF Network。。。有点过时了，不过学习他相当于对于我们之前的其他内容也做了一个总结。