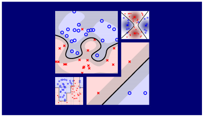
Machine Learning Techniques
(機器學習技法)



Lecture 14: Radial Basis Function Network

Hsuan-Tien Lin (林軒田)

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Roadmap

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 13: Deep Learning

pre-training with **denoising autoencoder** (**non-linear PCA**) and fine-tuning with **backprop** for NNet with **many layers**

Lecture 14: Radial Basis Function Network

- RBF Network Hypothesis
- RBF Network Learning
- k-Means Algorithm
- k-Means and RBF Network in Action

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM:

achieve large margin in infinite-dimensional space, **remember? :-)**

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$;
achieve large margin in infinite-dimensional space, **remember? :-)**

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$;
achieve large margin in infinite-dimensional space, **remember? :-)**

- Gaussian kernel: also called Radial Basis Function (RBF) kernel

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$; achieve large margin in infinite-dimensional space, **remember? :-)**

- Gaussian kernel: also called Radial Basis Function (RBF) kernel
  - radial: only depends on distance between $\mathbf{x}$ and 'center' $\mathbf{x}_n$

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$; achieve large margin in infinite-dimensional space, **remember? :-)**

- Gaussian kernel: also called Radial Basis Function (RBF) kernel
  - radial: only depends on distance between $\mathbf{x}$ and 'center' $\mathbf{x}_n$
  - basis function: to be 'combined'

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$; achieve large margin in infinite-dimensional space, **remember? :-)**

- Gaussian kernel: also called Radial Basis Function (RBF) kernel
  - radial: only depends on distance between $\mathbf{x}$ and 'center' $\mathbf{x}_n$
  - basis function: to be 'combined'
- let $g_n(\mathbf{x}) = y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right)$:

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n \qquad + b\right)$$

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$;
achieve large margin in infinite-dimensional space, **remember? :-)**

- Gaussian kernel: also called Radial Basis Function (RBF) kernel
  - radial: only depends on distance between $\mathbf{x}$ and 'center' $\mathbf{x}_n$
  - basis function: to be 'combined'
- let $g_n(\mathbf{x}) = y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right)$:

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n g_n(\mathbf{x}) + b\right)$$

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$;
achieve large margin in infinite-dimensional space, **remember? :-)**

- Gaussian kernel: also called Radial Basis Function (RBF) kernel
  - radial: only depends on distance between $\mathbf{x}$ and 'center' $\mathbf{x}_n$
  - basis function: to be 'combined'
- let $g_n(\mathbf{x}) = y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right)$:

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n g_n(\mathbf{x}) + b\right)$$

—linear aggregation of selected radial hypotheses

Gaussian SVM Revisited

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$; achieve large margin in infinite-dimensional space, **remember? :-)**

- Gaussian kernel: also called Radial Basis Function (RBF) kernel
  - radial: only depends on distance between $\mathbf{x}$ and 'center' $\mathbf{x}_n$
  - basis function: to be 'combined'
- let $g_n(\mathbf{x}) = y_n \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2\right)$:

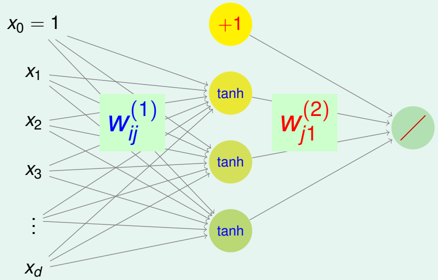
$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n g_n(\mathbf{x}) + b\right)$$

—linear aggregation of selected radial hypotheses

Radial Basis Function (RBF) **Network**: linear aggregation of radial hypotheses

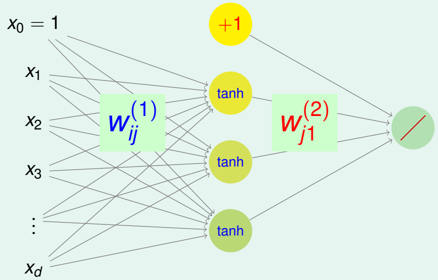
From Neural Network to RBF Network

Neural Network

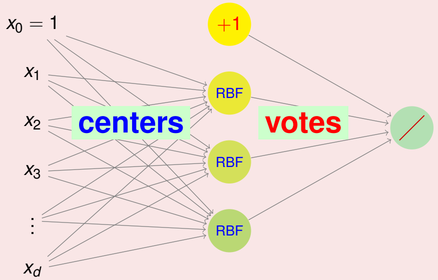


From Neural Network to RBF Network

Neural Network

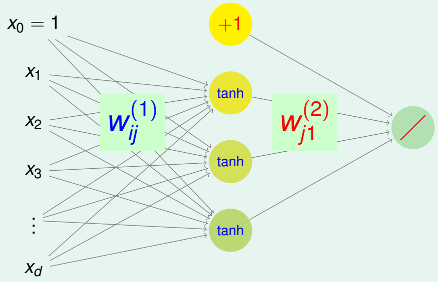


RBF Network

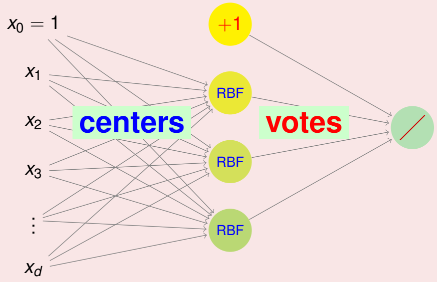


From Neural Network to RBF Network

Neural Network



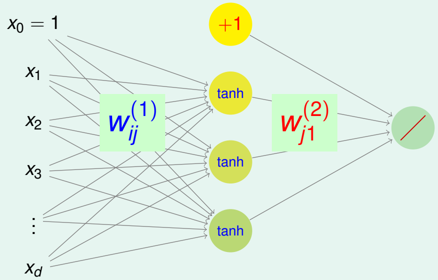
RBF Network



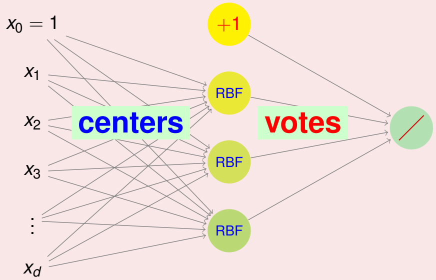
- output layer same: **just linear aggregation**

From Neural Network to RBF Network

Neural Network



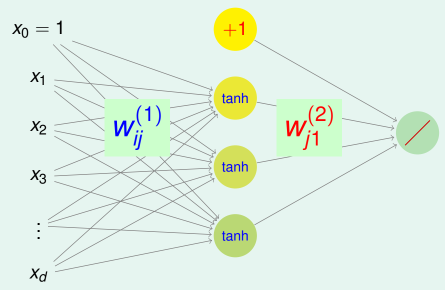
RBF Network



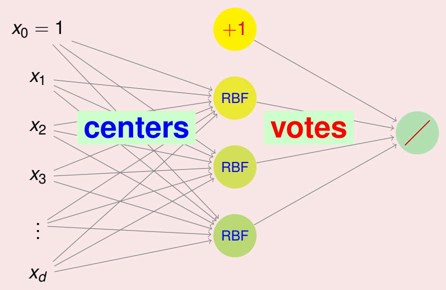
- hidden layer different:
  (inner-product + tanh) versus (distance + Gaussian)
- output layer same: **just linear aggregation**

From Neural Network to RBF Network

Neural Network



RBF Network



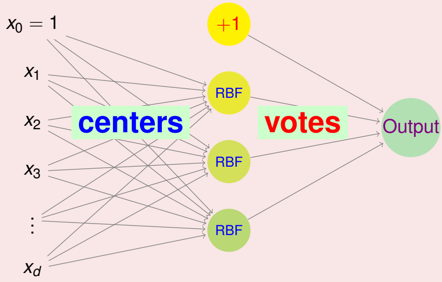
- hidden layer different: (inner-product + tanh) versus (distance + Gaussian)
- output layer same: **just linear aggregation**

RBF Network: historically **a type of NNet**

RBF Network Hypothesis

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_m) + b\right)$$

RBF Network



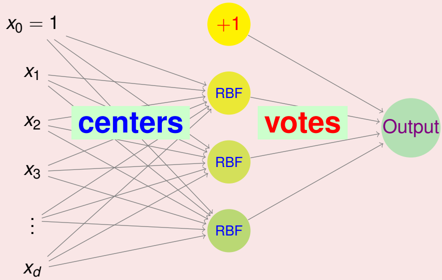
RBF Network Hypothesis

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_m) + b\right)$$

key variables:

centers $\boldsymbol{\mu}_m$; (signed) votes β_m

RBF Network



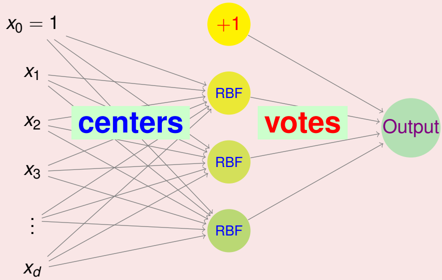
RBF Network Hypothesis

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_m) + b\right)$$

key variables:

centers $\boldsymbol{\mu}_m$; (signed) votes β_m

RBF Network



g_{SVM} for Gaussian-SVM

- RBF: Gaussian; Output: sign (binary classification)

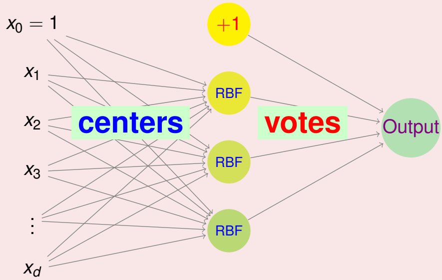
RBF Network Hypothesis

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_m) + b\right)$$

key variables:

centers $\boldsymbol{\mu}_m$; (signed) votes β_m

RBF Network



g_{SVM} for Gaussian-SVM

- RBF: Gaussian; Output: sign (binary classification)
- $M = \#\text{SV}$; $\boldsymbol{\mu}_m$: SVM SVs $\mathbf{x}_m$; β_m: $\alpha_m y_m$ from SVM Dual

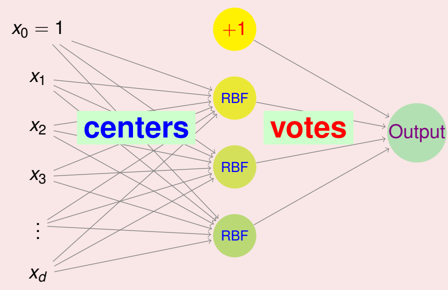
RBF Network Hypothesis

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_m) + b\right)$$

key variables:

centers $\boldsymbol{\mu}_m$; (signed) votes β_m

RBF Network



g_{SVM} for Gaussian-SVM

- RBF: Gaussian; Output: sign (binary classification)
- $M = \#\text{SV}$; $\boldsymbol{\mu}_m$: SVM SVs $\mathbf{x}_m$; β_m: $\alpha_m y_m$ from SVM Dual

learning: given RBF and Output,
decide $\boldsymbol{\mu}_m$ and β_m

RBF and Similarity

kernel: similarity via $\mathcal{Z}$-space inner product
—**governed by Mercer's condition, remember? :-)**

$\text{Poly}(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^2$

$\text{Gaussian}(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$

RBF and Similarity

kernel: similarity via $\mathcal{Z}$-space inner product
—**governed by Mercer's condition, remember? :-)**

$\text{Poly}(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T\mathbf{x}')^2$

$\text{Gaussian}(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$

$\text{Truncated}(\mathbf{x}, \mathbf{x}') = [\![\|\mathbf{x} - \mathbf{x}'\| \leq 1]\!] (1 - \|\mathbf{x} - \mathbf{x}'\|)^2$

RBF: similarity via $\mathcal{X}$-space distance
—often **monotonically non-increasing** to distance

RBF and Similarity

general similarity function between $\mathbf{x}$ and $\mathbf{x}'$:

$$\text{Neuron}(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \mathbf{x}^T \mathbf{x}' + 1)$$
$$\text{DNASim}(\mathbf{x}, \mathbf{x}') = \text{EditDistance}(\mathbf{x}, \mathbf{x}')$$

kernel: similarity via $\mathcal{Z}$-space inner product
—**governed by Mercer's condition, remember? :-)**

$$\text{Poly}(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^2$$

$$\text{Gaussian}(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

$$\text{Truncated}(\mathbf{x}, \mathbf{x}') = [\![\|\mathbf{x} - \mathbf{x}'\| \le 1]\!] (1 - \|\mathbf{x} - \mathbf{x}'\|)^2$$

RBF: similarity via $\mathcal{X}$-space distance
—often **monotonically non-increasing** to distance

RBF and Similarity

general similarity function between $\mathbf{x}$ and $\mathbf{x}'$:

$$\text{Neuron}(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \mathbf{x}^T \mathbf{x}' + 1)$$
$$\text{DNASim}(\mathbf{x}, \mathbf{x}') = \text{EditDistance}(\mathbf{x}, \mathbf{x}')$$

kernel: similarity via $\mathcal{Z}$-space inner product
—**governed by Mercer's condition, remember? :-)**

$\text{Poly}(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^2$

$\text{Gaussian}(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$

$\text{Truncated}(\mathbf{x}, \mathbf{x}') = [\![\|\mathbf{x} - \mathbf{x}'\| \le 1]\!] (1 - \|\mathbf{x} - \mathbf{x}'\|)^2$

RBF: similarity via $\mathcal{X}$-space distance
—often **monotonically non-increasing** to distance

RBF Network: distance similarity-to-centers as **feature transform**

Fun Time

Which of the following is not a radial basis function?

1. $\phi(\mathbf{x}, \boldsymbol{\mu}) = \exp(-\gamma\|\mathbf{x} - \boldsymbol{\mu}\|^2)$
2. $\phi(\mathbf{x}, \boldsymbol{\mu}) = -\sqrt{\mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\boldsymbol{\mu} + \boldsymbol{\mu}^T\boldsymbol{\mu}}$
3. $\phi(\mathbf{x}, \boldsymbol{\mu}) = [\![\mathbf{x} = \boldsymbol{\mu}]\!]$
4. $\phi(\mathbf{x}, \boldsymbol{\mu}) = \mathbf{x}^T\mathbf{x} + \boldsymbol{\mu}^T\boldsymbol{\mu}$

Fun Time

Which of the following is not a radial basis function?

1. $\phi(\mathbf{x}, \boldsymbol{\mu}) = \exp(-\gamma\|\mathbf{x} - \boldsymbol{\mu}\|^2)$
2. $\phi(\mathbf{x}, \boldsymbol{\mu}) = -\sqrt{\mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\boldsymbol{\mu} + \boldsymbol{\mu}^T\boldsymbol{\mu}}$
3. $\phi(\mathbf{x}, \boldsymbol{\mu}) = [\![\mathbf{x} = \boldsymbol{\mu}]\!]$
4. $\phi(\mathbf{x}, \boldsymbol{\mu}) = \mathbf{x}^T\mathbf{x} + \boldsymbol{\mu}^T\boldsymbol{\mu}$

Reference Answer: (4)

Note that (3) is an extreme case of (1) (Gaussian) with $\gamma \to \infty$, and (2) contains an $\|\mathbf{x} - \boldsymbol{\mu}\|^2$ somewhere :-).

Full RBF Network

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

- full RBF Network: $M = N$ and each $\boldsymbol{\mu}_m = \mathbf{x}_m$

Full RBF Network

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

- full RBF Network: $M = N$ and each $\boldsymbol{\mu}_m = \mathbf{x}_m$
- physical meaning: each $\mathbf{x}_m$ **influences** similar $\mathbf{x}$ by β_m

Full RBF Network

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

- full RBF Network: $M = N$ and each $\boldsymbol{\mu}_m = \mathbf{x}_m$
- 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

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

Full RBF Network

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

- full RBF Network: $M = N$ and each $\boldsymbol{\mu}_m = \mathbf{x}_m$
- 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

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

- full RBF Network: $M = N$ and each $\boldsymbol{\mu}_m = \mathbf{x}_m$
- 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 $\boldsymbol{\mu}_m$

Nearest Neighbor

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

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 $\mathbf{x}_m$**

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

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

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**

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:

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

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

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:

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

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

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(...) 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**

Interpolation by Full RBF Network

full RBF Network for squared error regression:

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

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

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

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 $\boldsymbol{\beta}$? $\boldsymbol{\beta} = (\mathrm{Z}^T\mathrm{Z})^{-1}\mathrm{Z}^T\mathbf{y}$, if $\mathrm{Z}^T\mathrm{Z}$ invertible, **remember? :-)**

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 $\boldsymbol{\beta}$? $\boldsymbol{\beta} = (\mathrm{Z}^T\mathrm{Z})^{-1}\mathrm{Z}^T\mathbf{y}$, if $\mathrm{Z}^T\mathrm{Z}$ invertible, **remember? :-)**
- size of Z?

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 $\boldsymbol{\beta}$? $\boldsymbol{\beta} = (\mathrm{Z}^T\mathrm{Z})^{-1}\mathrm{Z}^T\mathbf{y}$, if $\mathrm{Z}^T\mathrm{Z}$ invertible, **remember? :-)**
- size of Z? N (examples) by N (centers)
  —symmetric square matrix

Interpolation by Full RBF Network

full RBF Network for squared error regression:

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

- 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 $\boldsymbol{\beta}$? $\boldsymbol{\beta} = (\mathrm{Z}^T\mathrm{Z})^{-1}\mathrm{Z}^T\mathbf{y}$, if $\mathrm{Z}^T\mathrm{Z}$ 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**

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 $\boldsymbol{\beta}$? $\boldsymbol{\beta} = (\mathrm{Z}^T\mathrm{Z})^{-1}\mathrm{Z}^T\mathbf{y}$, if $\mathrm{Z}^T\mathrm{Z}$ 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 $\boldsymbol{\beta}$ with invertible Z: $\boldsymbol{\beta} = \quad \mathbf{y}$

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 $\boldsymbol{\beta}$? $\boldsymbol{\beta} = (\mathrm{Z}^T\mathrm{Z})^{-1}\mathrm{Z}^T\mathbf{y}$, if $\mathrm{Z}^T\mathrm{Z}$ 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 $\boldsymbol{\beta}$ with invertible Z: $\boldsymbol{\beta} = \mathrm{Z}^{-1}\mathbf{y}$

Regularized Full RBF Network

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

$g_{\text{RBF}}(\mathbf{x}_1) = \boldsymbol{\beta}^T \mathbf{z}_1 =$

Regularized Full RBF Network

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

$g_{\text{RBF}}(\mathbf{x}_1) = \beta^T \mathbf{z}_1 =$ (first column of Z)

Regularized Full RBF Network

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

$g_{\text{RBF}}(\mathbf{x}_1) = \boldsymbol{\beta}^T \mathbf{z}_1 = \mathbf{y}^T \mathrm{Z}^{-1}$(first column of Z)

Regularized Full RBF Network

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

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

Regularized Full RBF Network

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

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

Regularized Full RBF Network

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

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

Regularized Full RBF Network

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

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

Regularized Full RBF Network

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

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

—$g_{\text{RBF}}(\mathbf{x}_n) = \quad ,$

Regularized Full RBF Network

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

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

—$g_{\text{RBF}}(\mathbf{x}_n) = y_n$,

Regularized Full RBF Network

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

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

—$g_{\text{RBF}}(\mathbf{x}_n) = y_n$, i.e. $E_{\text{in}}(g_{\text{RBF}}) =$

Regularized Full RBF Network

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

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

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

Regularized Full RBF Network

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

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

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

- called **exact interpolation** for **function approximation**

Regularized Full RBF Network

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

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

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

- called **exact interpolation** for **function approximation**
- but **overfitting for learning? :-(**

Regularized Full RBF Network

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

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

—$g_{\text{RBF}}(\mathbf{x}_n) = y_n$, 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 $\boldsymbol{\beta}$ instead —optimal $\boldsymbol{\beta} = (\mathrm{Z}^T\mathrm{Z} + \lambda \mathrm{I})^{-1}\mathrm{Z}^T\mathbf{y}$

Regularized Full RBF Network

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

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

—$g_{\text{RBF}}(\mathbf{x}_n) = y_n$, 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 $\beta = (\mathrm{Z}^T\mathrm{Z} + \lambda \mathrm{I})^{-1}\mathrm{Z}^T\mathbf{y}$
- seen Z? $\mathrm{Z} = [\text{Gaussian}(\mathbf{x}_n, \mathbf{x}_m)] =$

Regularized Full RBF Network

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

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

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

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

Regularized Full RBF Network

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

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

—$g_{\text{RBF}}(\mathbf{x}_n) = y_n$, 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 $\boldsymbol{\beta}$ instead —optimal $\boldsymbol{\beta} = (\mathrm{Z}^T\mathrm{Z} + \lambda\mathrm{I})^{-1}\mathrm{Z}^T\mathbf{y}$
- seen Z? $\mathrm{Z} = [\text{Gaussian}(\mathbf{x}_n, \mathbf{x}_m)] =$ Gaussian kernel matrix K

effect of regularization in different spaces:

regularized full RBFNet: $\boldsymbol{\beta} = (\mathrm{Z}^T\mathrm{Z} + \lambda\mathrm{I})^{-1}\mathrm{Z}^T\mathbf{y}$

Regularized Full RBF Network

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

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

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

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

effect of regularization in different spaces:

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

Fewer Centers as Regularization

recall:

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

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

Fewer Centers as Regularization

recall:

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

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

- next: $M \ll N$ instead of $M = N$

Fewer Centers as Regularization

recall:

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

—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**

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**

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 $\boldsymbol{\mu}_m$: **prototypes**

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

Fun Time

If $\mathbf{x}_1 = \mathbf{x}_2$, what happens in the Z matrix of full Gaussian RBF network?

1. the first two rows of the matrix are the same
2. the first two columns of the matrix are different
3. the matrix is invertible
4. the sub-matrix at the intersection of the first two rows and the first two columns contains a constant of 0

Fun Time

If $\mathbf{x}_1 = \mathbf{x}_2$, what happens in the Z matrix of full Gaussian RBF network?

1. the first two rows of the matrix are the same
2. the first two columns of the matrix are different
3. the matrix is invertible
4. the sub-matrix at the intersection of the first two rows and the first two columns contains a constant of 0

Reference Answer: (1)

It is easy to see that the first two rows must be the same; so must the first two columns. The two same rows makes the matrix singular; the sub-matrix in (4) contains a constant of $1 = \exp(-0)$ instead of 0.

Good Prototypes: Clustering Problem

if $\mathbf{x}_1 \approx \mathbf{x}_2$,

Good Prototypes: Clustering Problem

if $\mathbf{x}_1 \approx \mathbf{x}_2$,

$\Longrightarrow$ **no need** both $\text{RBF}(\mathbf{x}, \mathbf{x}_1)$ & $\text{RBF}(\mathbf{x}, \mathbf{x}_2)$ in RBFNet,

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** $\mu \approx \mathbf{x}_1 \approx \mathbf{x}_2$

Good Prototypes: Clustering Problem

if $\mathbf{x}_1 \approx \mathbf{x}_2$,

$\Longrightarrow$ **no need** both $\text{RBF}(\mathbf{x}, \mathbf{x}_1)$ & $\text{RBF}(\mathbf{x}, \mathbf{x}_2)$ 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**:

Good Prototypes: Clustering Problem

if $\mathbf{x}_1 \approx \mathbf{x}_2$,
$\Longrightarrow$ **no need** both $\text{RBF}(\mathbf{x}, \mathbf{x}_1)$ & $\text{RBF}(\mathbf{x}, \mathbf{x}_2)$ 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 $S_1, S_2, \cdots, S_M$

Good Prototypes: Clustering Problem

if $\mathbf{x}_1 \approx \mathbf{x}_2$,

$\Longrightarrow$ **no need** both $\text{RBF}(\mathbf{x}, \mathbf{x}_1)$ & $\text{RBF}(\mathbf{x}, \mathbf{x}_2)$ 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

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 $\in S_m \Leftrightarrow \mu_m \approx \mathbf{x}_1 \approx x_2$

Good Prototypes: Clustering Problem

if $\mathbf{x}_1 \approx \mathbf{x}_2$,

$\Longrightarrow$ **no need** both $\text{RBF}(\mathbf{x}, \mathbf{x}_1)$ & $\text{RBF}(\mathbf{x}, \mathbf{x}_2)$ 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:

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

Good Prototypes: Clustering Problem

if $\mathbf{x}_1 \approx \mathbf{x}_2$,

$\Longrightarrow$ **no need** both $\mathrm{RBF}(\mathbf{x}, \mathbf{x}_1)$ & $\mathrm{RBF}(\mathbf{x}, \mathbf{x}_2)$ in RBFNet,

$\Longrightarrow$ **cluster** $\mathbf{x}_1$ and $\mathbf{x}_2$ by **one prototype** $\boldsymbol{\mu} \approx \mathbf{x}_1 \approx \mathbf{x}_2$

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

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

$$E_{\text{in}}(S_1, \cdots, S_M; \boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M) = \frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

$$\min_{\{S_1, \cdots, S_M; \boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M\}} E_{\text{in}}(S_1, \cdots, S_M; \boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M)$$

Partition Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

Partition Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N}\sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

- **hard to optimize**: joint combinatorial-numerical optimization

Partition Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

Partition Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M$ **fixed**, for each $\mathbf{x}_n$

Partition Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M$ **fixed**, for each $\mathbf{x}_n$

- $[\![\mathbf{x}_n \in S_m]\!]$: choose one and only one subset

Partition Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M$ **fixed**, for each $\mathbf{x}_n$

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

Partition Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M$ **fixed**, for each $\mathbf{x}_n$

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

optimal chosen subset $S_m =$ the one with $\|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$

Partition Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N}\sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!]\|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M$ **fixed**, for each $\mathbf{x}_n$

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

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

Partition Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M$ **fixed**, for each $\mathbf{x}_n$

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

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

for given $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_M$, each $\mathbf{x}_n$
'optimally partitioned' using its **closest** $\boldsymbol{\mu}_m$

Prototype Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

Prototype Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each $\boldsymbol{\mu}_m$

Prototype Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each $\boldsymbol{\mu}_m$

$$\nabla_{\boldsymbol{\mu}_m} E_{\text{in}} = -2 \sum_{n=1}^{N} [\![\mathbf{x}_n \in S_m]\!]$$

Prototype Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each $\boldsymbol{\mu}_m$

$$\nabla_{\boldsymbol{\mu}_m} E_{\text{in}} = -2 \sum_{n=1}^{N} [\![\mathbf{x}_n \in S_m]\!](\mathbf{x}_n - \boldsymbol{\mu}_m)$$

Prototype Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each $\boldsymbol{\mu}_m$

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

Prototype Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each $\boldsymbol{\mu}_m$

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

Prototype Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N}\sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each $\boldsymbol{\mu}_m$

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

optimal prototype $\boldsymbol{\mu}_m =$ of $\mathbf{x}_n$ within S_m

Prototype Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each $\boldsymbol{\mu}_m$

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

optimal prototype $\boldsymbol{\mu}_m =$ **average** of $\mathbf{x}_n$ within S_m

Prototype Optimization

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

$$\min_{\{S_1,\cdots,S_M;\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \boldsymbol{\mu}_m\|^2$$

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

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each $\boldsymbol{\mu}_m$

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

optimal prototype $\boldsymbol{\mu}_m$ = **average** of $\mathbf{x}_n$ within S_m

for given $S_1, \cdots, S_M$, each $\boldsymbol{\mu}_n$
'optimally computed' as **consensus** within S_m

k-Means Algorithm

use k **prototypes** instead of M historically

k-Means Algorithm

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

k-Means Algorithm

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

k-Means Algorithm

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 $\boldsymbol{\mu}_i$

until **converge**

k-Means Algorithm

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

k-Means Algorithm

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 $\boldsymbol{\mu}_i$
   2. optimize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$:
      each $\boldsymbol{\mu}_n$ 'optimally computed' as consensus within S_m

   until **converge**

k-Means Algorithm

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

k-Means Algorithm

1. initialize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$:
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 $\boldsymbol{\mu}_i$
   2. optimize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$:
      each $\boldsymbol{\mu}_n$ 'optimally computed' as consensus within S_m

   until **converge**

k-Means Algorithm

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

k-Means Algorithm

1. initialize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$: 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 $\boldsymbol{\mu}_i$
   2. optimize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$:
      each $\boldsymbol{\mu}_n$ 'optimally computed' as consensus within S_m

   until **converge**

k-Means Algorithm

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

k-Means Algorithm

1. initialize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$: 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 $\boldsymbol{\mu}_i$
   2. optimize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$:
      each $\boldsymbol{\mu}_n$ 'optimally computed' as consensus within S_m

   until **converge**

converge: no change of $S_1, S_2, \ldots, S_k$ anymore

k-Means Algorithm

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

k-Means Algorithm

1. initialize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$: 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 $\boldsymbol{\mu}_i$
   2. optimize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$:
      each $\boldsymbol{\mu}_n$ '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 Algorithm

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

k-Means Algorithm

1. initialize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$: 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 $\boldsymbol{\mu}_i$
   2. optimize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$:
      each $\boldsymbol{\mu}_n$ '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**

RBF Network Using *k*-Means

RBF Network Using *k*-Means

1. run *k*-Means with $k = M$ to get $\{\boldsymbol{\mu}_m\}$

RBF Network Using k-Means

RBF Network Using k-Means

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

$$\boldsymbol{\Phi}(\mathbf{x}) = [\text{RBF}(\mathbf{x}, \boldsymbol{\mu}_1), \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_M)]$$

RBF Network Using *k*-Means

RBF Network Using *k*-Means

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

$$\boldsymbol{\Phi}(\mathbf{x}) = [\mathrm{RBF}(\mathbf{x}, \boldsymbol{\mu}_1), \mathrm{RBF}(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, \mathrm{RBF}(\mathbf{x}, \boldsymbol{\mu}_M)]$$

3. run linear model on $\{(\boldsymbol{\Phi}(\mathbf{x}_n), y_n)\}$ to get $\boldsymbol{\beta}$

RBF Network Using k-Means

RBF Network Using k-Means

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

$$\boldsymbol{\Phi}(\mathbf{x}) = [\text{RBF}(\mathbf{x}, \boldsymbol{\mu}_1), \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_M)]$$

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

RBF Network Using k-Means

RBF Network Using k-Means

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

$$\boldsymbol{\Phi}(\mathbf{x}) = [\text{RBF}(\mathbf{x}, \boldsymbol{\mu}_1), \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_M)]$$

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

- using unsupervised learning (k-Means) to assist **feature transform**

RBF Network Using k-Means

RBF Network Using k-Means

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

$$\boldsymbol{\Phi}(\mathbf{x}) = [\mathrm{RBF}(\mathbf{x}, \boldsymbol{\mu}_1), \mathrm{RBF}(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, \mathrm{RBF}(\mathbf{x}, \boldsymbol{\mu}_M)]$$

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

- using unsupervised learning (k-Means) to assist **feature transform**—like **autoencoder**

RBF Network Using k-Means

RBF Network Using k-Means

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

$$\boldsymbol{\Phi}(\mathbf{x}) = [\mathrm{RBF}(\mathbf{x}, \boldsymbol{\mu}_1), \mathrm{RBF}(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, \mathrm{RBF}(\mathbf{x}, \boldsymbol{\mu}_M)]$$

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

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

RBF Network Using *k*-Means

RBF Network Using *k*-Means

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

$$\boldsymbol{\Phi}(\mathbf{x}) = [\text{RBF}(\mathbf{x}, \boldsymbol{\mu}_1), \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_2), \ldots, \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_M)]$$

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

- 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

Fun Time

For k-Means, consider examples $\mathbf{x}_n \in \mathbb{R}^2$ such that all $x_{n,1}$ and $x_{n,2}$ are non-zero. When fixing two prototypes $\boldsymbol{\mu}_1 = [1, 1]$ and $\boldsymbol{\mu}_2 = [-1, 1]$, which of the following set is the optimal S_1?

1. $\{\mathbf{x}_n \colon x_{n,1} > 0\}$
2. $\{\mathbf{x}_n \colon x_{n,1} < 0\}$
3. $\{\mathbf{x}_n \colon x_{n,2} > 0\}$
4. $\{\mathbf{x}_n \colon x_{n,2} < 0\}$

Fun Time

For k-Means, consider examples $\mathbf{x}_n \in \mathbb{R}^2$ such that all $x_{n,1}$ and $x_{n,2}$ are non-zero. When fixing two prototypes $\boldsymbol{\mu}_1 = [1, 1]$ and $\boldsymbol{\mu}_2 = [-1, 1]$, which of the following set is the optimal S_1?

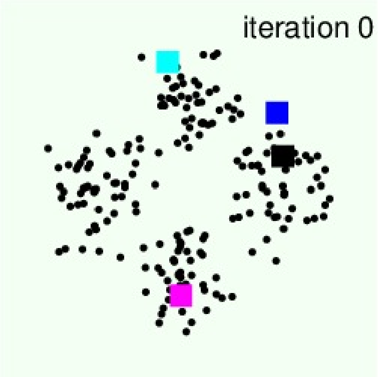
1. $\{\mathbf{x}_n : x_{n,1} > 0\}$
2. $\{\mathbf{x}_n : x_{n,1} < 0\}$
3. $\{\mathbf{x}_n : x_{n,2} > 0\}$
4. $\{\mathbf{x}_n : x_{n,2} < 0\}$

Reference Answer: 1

Note that S_1 contains examples that are closer to $\boldsymbol{\mu}_1$ than $\boldsymbol{\mu}_2$.

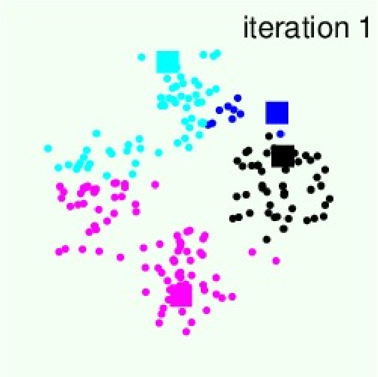
Beauty of *k*-Means

$k = 4$



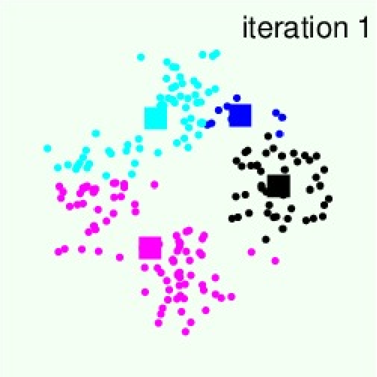
Beauty of *k*-Means

$k = 4$



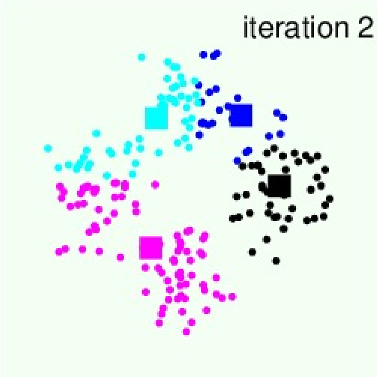
Beauty of *k*-Means

$k = 4$



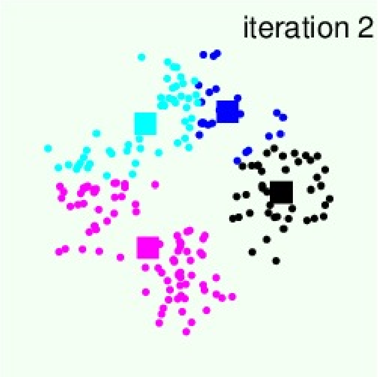
Beauty of k-Means

$k = 4$



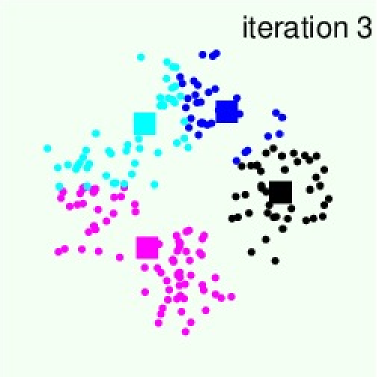
Beauty of k-Means

$k = 4$



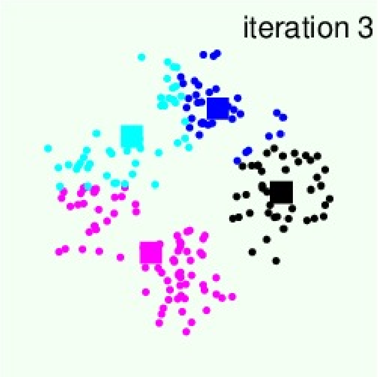
Beauty of *k*-Means

$k = 4$



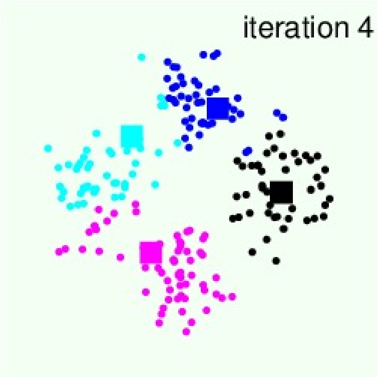
Beauty of *k*-Means

$k = 4$



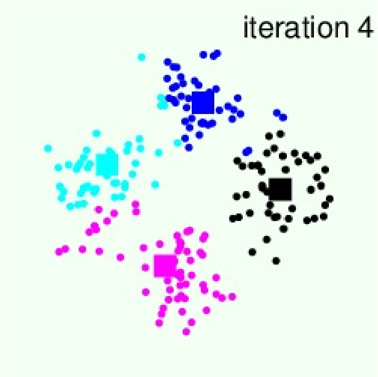
Beauty of k-Means

$$k = 4$$



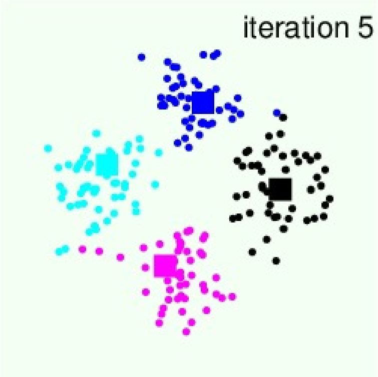
Beauty of k-Means

$k = 4$



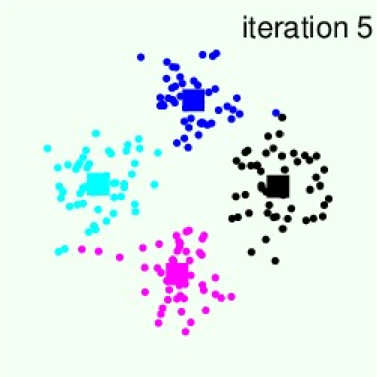
Beauty of *k*-Means

$k = 4$



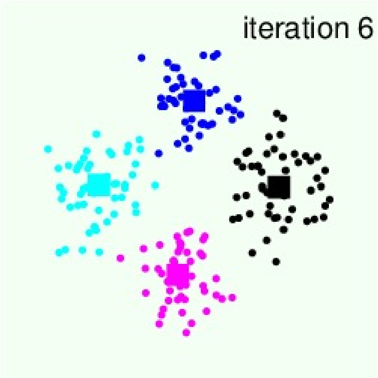
Beauty of k-Means

$k = 4$



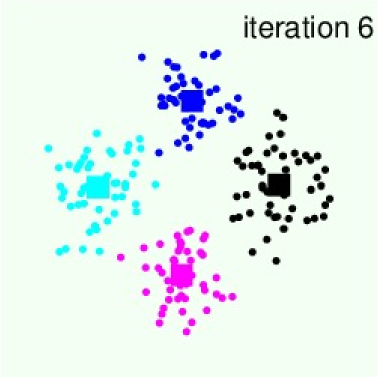
Beauty of k-Means

$k = 4$



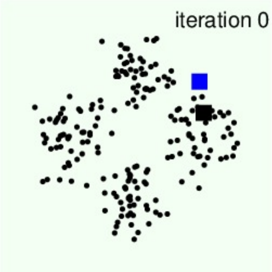
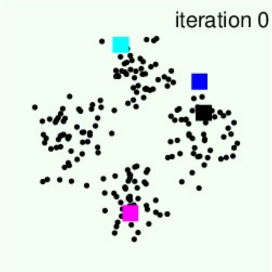
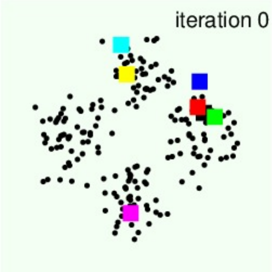
Beauty of *k*-Means

$k = 4$

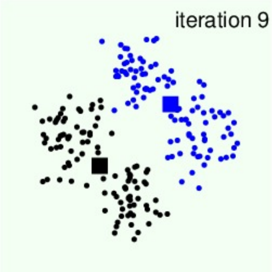
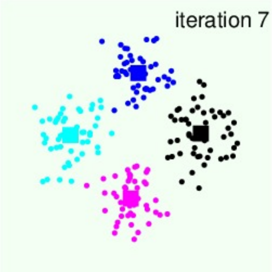
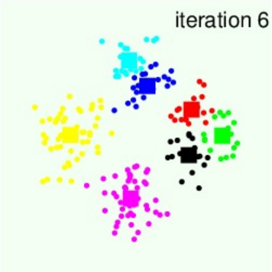


usually works well
with **proper *k* and initialization**

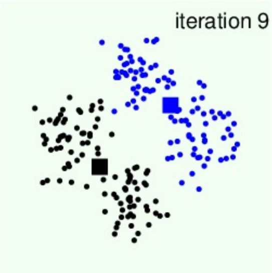
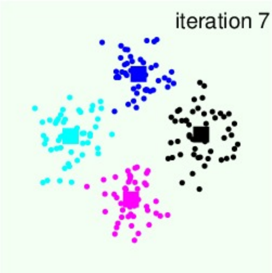
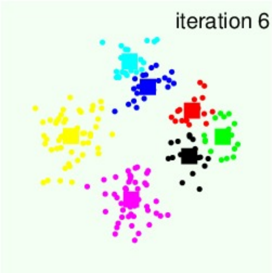
Difficulty of *k*-Means

$k = 2$	$k = 4$	$k = 7$
		

Difficulty of *k*-Means

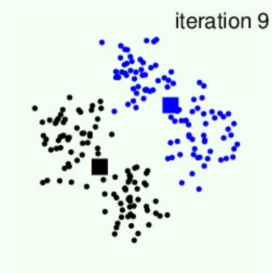
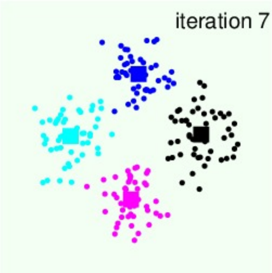
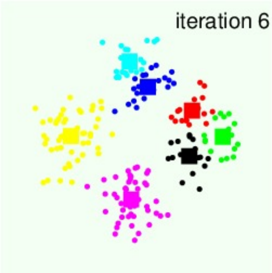
$k = 2$	$k = 4$	$k = 7$
		

Difficulty of *k*-Means

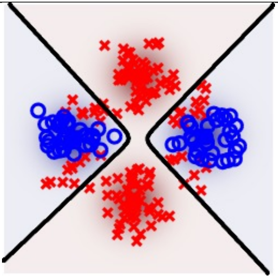
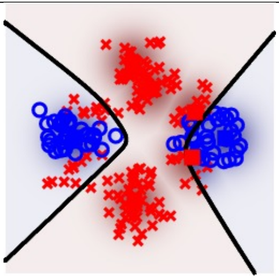
$k = 2$	$k = 4$	$k = 7$
		

'sensitive' to *k* and initialization

RBF Network Using *k*-Means

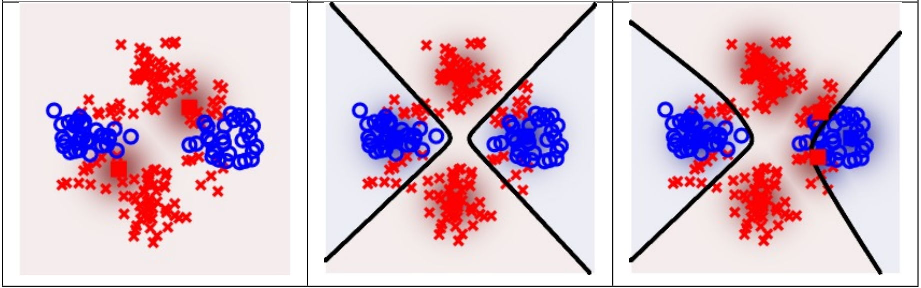
$k = 2$	$k = 4$	$k = 7$
		

RBF Network Using k-Means

$k = 2$	$k = 4$	$k = 7$
		

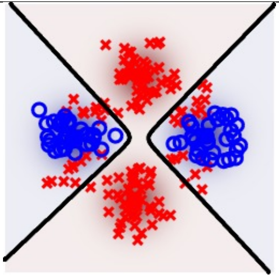
RBF Network Using k-Means

$k = 2$	$k = 4$	$k = 7$

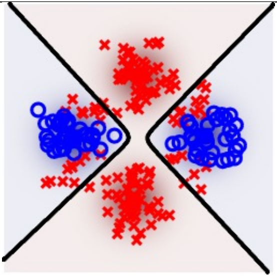
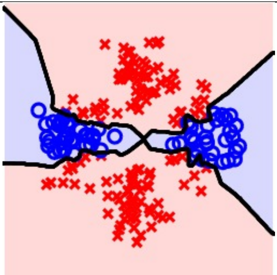


reasonable performance
with **proper centers**

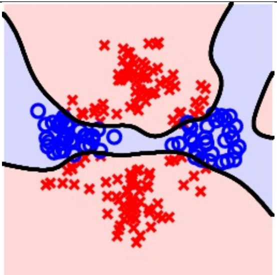
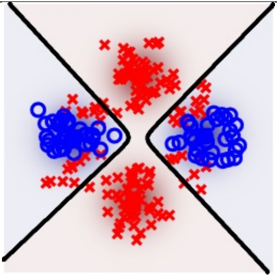
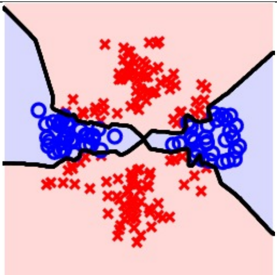
Full RBF Network

$k = N$ $\lambda = 0.001$	$k = 4$	nearest neighbor
		

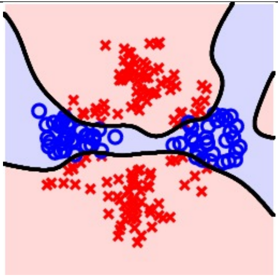
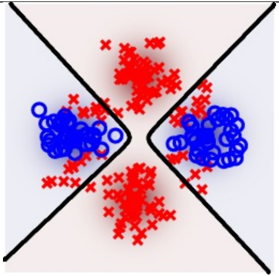
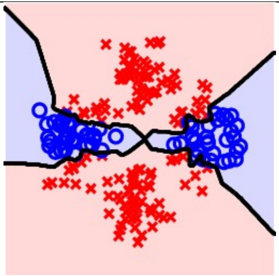
Full RBF Network

$k = N$ $\lambda = 0.001$	$k = 4$	nearest neighbor
		

Full RBF Network

$k = N$ $\lambda = 0.001$	$k = 4$	nearest neighbor
		

Full RBF Network

$k = N$ $\lambda = 0.001$	$k = 4$	nearest neighbor
		

full RBF Network: generally less useful

Fun Time

When coupled with ridge linear regression, which of the following RBF Network is ‘most regularized’?

1. small M and small λ
2. small M and large λ
3. large M and small λ
4. large M and large λ

Fun Time

When coupled with ridge linear regression, which of the following RBF Network is 'most regularized'?

1. small M and small λ
2. small M and large λ
3. large M and small λ
4. large M and large λ

Reference Answer: ②

small M: fewer weights and more regularized;
large λ: shorter β more and more regularized.

Summary

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 14: Radial Basis Function Network

- RBF Network Hypothesis
  prototypes instead of neurons as transform
- RBF Network Learning
  linear aggregation of prototype 'hypotheses'
- *k*-Means Algorithm
  clustering with alternating optimization
- *k*-Means and RBF Network in Action
  proper choice of # prototypes important

- **next: extracting features from abstract data**