Unit 6.2 Kernel Methods and RBF Networks

K-Means Clustering

Kiran Bagale

Neural Networks and Pattern Recognition

August, 2025

K-Means Clustering

Definition

K-means clustering is a **fundamental unsupervised learning algorithm** that partitions a dataset into K clusters by minimizing within-cluster sum of squared distances.

- Partitions data into natural groupings
- Minimizes specified cost function
- Uses unlabeled data (unsupervised)
- Finds optimal cluster centers (centroids)

Key Insight

Clustering minimizes similarity measure between observations within each cluster

Objective Function

K-means Cost Function

The algorithm minimizes:

$$J = \sum_{i=1}^{N} \sum_{j=1}^{K} r_{ij} \|x_i - \mu_j\|^2 \tag{1}$$

Where:

- N = number of data points
- K = number of clusters
- x_i = i-th data point
- μ_j = centroid of cluster j
- r_{ij} = binary indicator (1 if $x_i \in$ cluster j, 0 otherwise)
- $\| \cdot \|$ = Euclidean norm

Algorithm Steps

Algorithm 1 K-Means Algorithm

1: **Initialize** K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$ randomly
2: **repeat**
3: **Assignment Step:** For each x_i:
4: $r_{ij} = 1$ if $j = \arg\min_k \|x_i - \mu_k\|^2$, else $r_{ij} = 0$
5: **Update Step:** For each cluster j:
6: $\mu_j = \frac{\sum_{i=1}^{N} r_{ij} x_i}{\sum_{i=1}^{N} r_{ij}}$
7: **until** Convergence (centroids no longer change significantly)

Mathematical Intuition

Voronoi Tessellation

- Creates Voronoi cells
- Each region contains points closest to one centroid
- Optimal partitioning

EM Perspective

- **E-step:** Assign points to clusters (expectation of hidden variables)
- **M-step:** Update centroids (maximize likelihood)

Convergence Properties

- Cost function $J \geq 0$
- Each iteration: J decreases or stays same
- Finite possible assignments
- Guaranteed convergence

RBF Network Architecture Drawback

Three-Layer Structure

1. **Input Layer:** Receives input vectors
2. **Hidden Layer:** Contains RBF units (Gaussian)
3. **Output Layer:** Linear combination of hidden units

Gaussian RBF Units

Each hidden unit computes:

$$\phi_j(x) = \exp\left(-\frac{\|x - c_j\|^2}{2\sigma_j^2}\right) \tag{2}$$

where c_j = center, σ_j = width parameter

Parameter Computation Challenge

Key Issue in RBF Design

How to compute parameters $\{c_j, \sigma_j\}$ of Gaussian units using **unlabeled data** in an **unsupervised manner**?

Clustering Solution

Clustering provides a natural framework for solving this parameter estimation problem through unsupervised learning.

Requirements:

- No labeled training data available
- Must determine optimal centers and widths
- Minimize computational complexity
- Ensure good generalization

K-Means Solution for RBF Networks

Center Determination

1. Apply K-means to input data
2. Use resulting centroids as RBF centers: $c_j = \mu_j$

Width Parameter Estimation

After obtaining centers:

$$\sigma_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} \|x_i - c_j\| \tag{3}$$

where C_j is the set of points assigned to cluster j

- Automatic parameter selection
- Data-driven approach
- Computationally efficient

Drawbacks of RBF Networks (1/2)

1. Curse of Dimensionality

Number of RBF units required grows **exponentially** with input dimensionality

2. Center Selection Sensitivity

Poor initialization leads to:

- Overfitting (too many centers)
- Underfitting (too few centers)
- Local optima in clustering

3. Uniform Width Assumption

Traditional approaches assume uniform widths, missing natural data density variations

Drawbacks of RBF Networks (2/2)

4. Limited Expressiveness

RBF networks with fixed centers struggle with:

- Complex decision boundaries
- Highly non-linear relationships
- Irregular cluster shapes

5. Scalability Issues

- Computational complexity increases with RBF units
- Memory requirements grow quadratically
- Training time prohibitive for large datasets

Cover's Theorem

Theorem Statement

A complex pattern classification problem cast in **high-dimensional space nonlinearly** is more likely to be **linearly separable** than in a low-dimensional space.

Mathematical Framework

Given input patterns $\{x_i\} \in \mathbb{R}^m$, map to higher-dimensional space $\mathbb{R}^M$ (where $M \gg m$) using nonlinear transformations $\phi(x)$.

Key Insight: Higher dimensionality through nonlinear mapping increases probability of linear separability

K-Means and Cover's Theorem

Connection Framework

The K-means algorithm fits within Cover's theorem because:

1. **Nonlinear Mapping:** Each centroid c_j defines transformation:

$$\phi_j(x) = \exp\left(-\frac{\|x - c_j\|^2}{2\sigma^2}\right) \tag{4}$$

2. **High-Dimensional Embedding:** $x \to \{\phi_1(x), \phi_2(x), \ldots, \phi_K(x)\}$ maps input to K-dimensional space
3. **Linear Separability:** Patterns become linearly separable in RBF feature space
4. **Probabilistic Interpretation:** K-means provides probabilistic partitioning

Theoretical Justification

Dimensionality Transformation

$$x \in \mathbb{R}^m \rightarrow \{\phi_1(x), \phi_2(x), \ldots, \phi_K(x)\} \in \mathbb{R}^K \tag{5}$$

When $K \gg m$:

- Cover's theorem suggests complex classification problems become more tractable
- Nonlinearly separable patterns in $\mathbb{R}^m$ may become linearly separable in $\mathbb{R}^K$
- K-means clustering provides the nonlinear basis functions

Result

K-means enables the practical implementation of Cover's theorem in RBF networks

Conclusion

Key Takeaways

- K-means provides **principled approach** to unsupervised RBF parameter learning
- Addresses challenge of computing Gaussian unit parameters from unlabeled data
- Despite RBF drawbacks (dimensionality, center selection), K-means offers elegant solution
- Algorithm fits within **Cover's theorem framework** through nonlinear mapping

Synergy

The combination demonstrates how **unsupervised learning techniques** can solve **supervised learning problems**, embodying Cover's theorem in practical ML applications.

K-means + RBF = Powerful Framework for Pattern Classification

6.3 Recursive Least-Squares (RLS) Algorithm

Weight Vector Estimation in RBF Networks

- The K-means algorithm performs computation in a recursive manner
- We want to compute the weight vector in RBF networks recursively too
- Traditional least squares requires matrix inversion, which is computationally expensive for large K
- Solution: Recursive Least-Squares (RLS) algorithm

The Normal Equation: We start with the normal equation:

$$\mathbf{R}(n)\hat{\mathbf{w}}(n) = \mathbf{r}(n), \quad n = 1, 2, \ldots \tag{6}$$

Where we have three key components:

1. Correlation function $\mathbf{R}(n)$ (K×K matrix)
2. Cross-correlation vector $\mathbf{r}(n)$ (K×1 vector)
3. Weight vector $\hat{\mathbf{w}}(n)$ (K×1 vector) - **unknown**

Component 1: Correlation Function

The K-by-K correlation function of hidden-unit outputs:

$$\mathbf{R}(n) = \sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\phi}^T(\mathbf{x}_i) \tag{7}$$

Where:

$$\boldsymbol{\phi}(\mathbf{x}_i) = [\varphi(\mathbf{x}_i, \boldsymbol{\mu}_1), \varphi(\mathbf{x}_i, \boldsymbol{\mu}_2), \ldots, \varphi(\mathbf{x}_i, \boldsymbol{\mu}_K)]^T \tag{8}$$

And each basis function is:

$$\varphi(\mathbf{x}_i, \boldsymbol{\mu}_j) = \exp\left(-\frac{1}{2\sigma_j^2}\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2\right) \tag{9}$$

Component 2: Cross-Correlation Vector

The K-by-1 cross-correlation vector between desired response and hidden-unit outputs:

$$\mathbf{r}(n) = \sum_{i=1}^{n} \phi(\mathbf{x}_i) d(i) \tag{10}$$

This represents the correlation between:

- The desired output $d(i)$
- The hidden layer activations $\phi(\mathbf{x}_i)$

RLS Algorithm Derivation - Step 1

We reformulate the cross-correlation vector recursively:

$$\mathbf{r}(n) = \sum_{i=1}^{n-1} \phi(\mathbf{x}_i) d(i) + \phi(\mathbf{x}_n) d(n) \tag{11}$$

$$= \mathbf{r}(n-1) + \phi(\mathbf{x}_n) d(n) \tag{12}$$

$$= \mathbf{R}(n-1)\hat{\mathbf{w}}(n-1) + \phi(\mathbf{x}_n) d(n) \tag{13}$$

Key insight: We can express the new cross-correlation in terms of the previous one!

RLS Algorithm Derivation - Step 2

Similarly, the correlation function updates as:

$$\mathbf{R}(n) = \mathbf{R}(n-1) + \boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n) \tag{14}$$

We introduce the prior estimation error:

$$\alpha(n) = d(n) - \boldsymbol{\phi}^T(n)\mathbf{w}(n-1) \tag{15}$$

$$= d(n) - \mathbf{w}^T(n-1)\boldsymbol{\phi}(n) \tag{16}$$

This is called "prior" because it uses the **old** weight estimate $\hat{\mathbf{w}}(n-1)$.

RLS Algorithm Derivation - Step 3

After algebraic manipulation, we get:

$$\mathbf{r}(n) = \mathbf{R}(n)\hat{\mathbf{w}}(n-1) + \boldsymbol{\phi}(n)\alpha(n) \tag{17}$$

Substituting into the normal equation:

$$\mathbf{R}(n)\hat{\mathbf{w}}(n) = \mathbf{R}(n)\hat{\mathbf{w}}(n-1) + \boldsymbol{\phi}(n)\alpha(n) \tag{18}$$

Therefore:

$$\boxed{\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{R}^{-1}(n)\boldsymbol{\phi}(n)\alpha(n)} \tag{19}$$

The Challenge: Computing $\mathbf{R}^{-1}(n)$

- We need $\mathbf{R}^{-1}(n)$ for the weight update
- Direct matrix inversion is computationally expensive ($O(K^3)$)
- Solution: Use the Matrix Inversion Lemma!

For matrices of the form:

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{D}\mathbf{C}^T \tag{20}$$

The Matrix Inversion Lemma gives:

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{B}\mathbf{C}(\mathbf{D} + \mathbf{C}^T\mathbf{B}\mathbf{C})^{-1}\mathbf{C}^T\mathbf{B} \tag{21}$$

Applying the Matrix Inversion Lemma

From $\mathbf{R}(n) = \mathbf{R}(n-1) + \boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)$, we identify:

$$\mathbf{A} = \mathbf{R}(n) \tag{22}$$
$$\mathbf{B}^{-1} = \mathbf{R}(n-1) \tag{23}$$
$$\mathbf{C} = \boldsymbol{\phi}(n) \tag{24}$$
$$\mathbf{D} = 1 \tag{25}$$

This gives us:

$$\boxed{\mathbf{R}^{-1}(n) = \mathbf{R}^{-1}(n-1) - \frac{\mathbf{R}^{-1}(n-1)\boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)\mathbf{R}^{-1}(n-1)}{1 + \boldsymbol{\phi}^T(n)\mathbf{R}^{-1}(n-1)\boldsymbol{\phi}(n)}} \tag{26}$$

Simplified Notation

Let's define:

$$\mathbf{P}(n) = \mathbf{R}^{-1}(n) \tag{27}$$

Then:

$$\boxed{\mathbf{P}(n) = \mathbf{P}(n-1) - \frac{\mathbf{P}(n-1)\boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)\mathbf{P}(n-1)}{\mathbf{P}(n-1)\boldsymbol{\phi}(n)\boldsymbol{\phi}^T(n)\mathbf{P}^T(n-1)}} \tag{28}$$

Note: The denominator is a scalar (quadratic form).

Gain Vector

We also define the gain vector:

$$\mathbf{g}(n) = \mathbf{R}^{-1}(n)\phi(n) \tag{29}$$
$$= \mathbf{P}(n)\phi(n) \tag{30}$$

The gain vector represents how much correction is needed when updating from $\hat{\mathbf{w}}(n-1)$ to $\hat{\mathbf{w}}(n)$.

The weight update becomes:

$$\boxed{\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{g}(n)\alpha(n)} \tag{31}$$

Complete RLS Algorithm

RLS Algorithm Summary

1. **Prior estimation error:**

$$\alpha(n) = d(n) - \mathbf{w}^T(n-1)\phi(n) \tag{32}$$

2. **Gain vector:**

$$\mathbf{g}(n) = \mathbf{P}(n)\phi(n) \tag{33}$$

3. **Weight update:**

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{g}(n)\alpha(n) \tag{34}$$

4. **Inverse correlation matrix update:**

$$\mathbf{P}(n) = \mathbf{P}(n-1) - \frac{\mathbf{P}(n-1)\phi(n)\phi^T(n)\mathbf{P}(n-1)}{1 + \phi^T(n)\mathbf{P}(n-1)\phi(n)} \tag{35}$$

Statistical Interpretation

Consider the linear regression model:

$$d(n) = \mathbf{w}^T \phi(n) + \varepsilon(n) \tag{36}$$

Where $\varepsilon(n)$ is white noise with zero mean and variance σ_e^2.

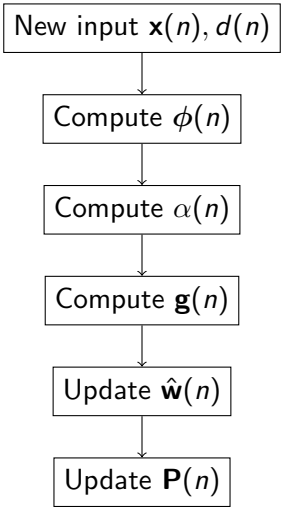
Then $\mathbf{P}(n)$ represents the state-error covariance matrix:

$$\mathbb{E}[(\mathbf{w} - \hat{\mathbf{w}}(n))(\mathbf{w} - \hat{\mathbf{w}}(n))^T] = \sigma_e^2 \mathbf{P}(n) \tag{37}$$

Benefits of RLS Algorithm

- Computational Efficiency: No matrix inversion required at each step
- Recursive Nature: Updates weights incrementally as new data arrives
- Memory Efficient: Only stores $\mathbf{P}(n)$ and $\hat{\mathbf{w}}(n)$
- Fast Convergence: Typically faster than gradient-based methods
- Optimal: Gives the least-squares solution at each time step

Algorithm Flow



Summary

- RLS provides a recursive way to solve the least-squares problem
- Uses the Matrix Inversion Lemma to avoid expensive matrix inversions
- Key components: prior estimation error, gain vector, and covariance matrix
- Computationally efficient and optimal for linear-in-parameters models
- Essential for real-time applications in RBF networks

6.4 Hybrid Learning Procedure for RBF Networks

Why Hybrid Learning?

The Challenge

RBF networks have two types of parameters that need to be learned:

1. **Hidden layer parameters**: Centers and widths of RBF functions
2. **Output layer parameters**: Linear weights

The Solution

Hybrid approach: Use different algorithms for different layers

- K-means algorithm for hidden layer (unsupervised)
- RLS algorithm for output layer (supervised)

The "K-means, RLS" Algorithm

K-means + RLS = Hybrid Learning

1. **Step 1**: Apply K-means to find RBF centers
2. **Step 2**: Calculate RBF widths based on center distances
3. **Step 3**: Use RLS to train output weights

Key Advantage

Computationally efficient because both K-means and RLS are individually efficient algorithms

Input Layer

Structure

- Size determined by input dimensionality
- If input vector $\mathbf{x}$ has dimension m_0
- Then input layer has m_0 nodes

Example

For a 2D pattern recognition problem:

- Input: $\mathbf{x} = [x_1, x_2]^T$
- Input layer size: $m_0 = 2$

Hidden Layer - Structure

Key Parameters

1. **Size (m_1)**: Number of RBF units = Number of clusters (K)
2. **Centers (μ_j)**: Determined by K-means clustering
3. **Widths (σ)**: Calculated from center distances

The K Parameter

- K = "degree of freedom" under designer's control
- Controls both performance and computational complexity
- Key to model selection problem

Hidden Layer - Center Calculation

K-means Algorithm

1. Apply K-means clustering to unlabeled input samples $\{\mathbf{x}_i\}$
2. Find K cluster centers $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$
3. Each center $\boldsymbol{\mu}_j$ becomes the center of RBF unit j

Gaussian RBF Function

For hidden unit j:

$$\varphi(\mathbf{x}_i) = \exp\left(-\frac{\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2}{2\sigma^2}\right)$$

Hidden Layer - Width Calculation

Width Formula

All Gaussian functions use the same width:

$$\sigma = \frac{d_{\max}}{\sqrt{2K}}$$

where:

- K = number of centers
- $d_{\max}$ = maximum distance between any two centers

Why This Formula?

- Prevents Gaussian units from being too peaked or too flat
- Based on the spread of centers found by K-means
- Ensures reasonable overlap between neighboring units

Output Layer

After Hidden Layer Training...

We have the hidden layer output vector:

$$\phi(\mathbf{x}_i) = \begin{bmatrix} \varphi(\mathbf{x}_i, \boldsymbol{\mu}_1) \\ \varphi(\mathbf{x}_i, \boldsymbol{\mu}_2) \\ \vdots \\ \varphi(\mathbf{x}_i, \boldsymbol{\mu}_K) \end{bmatrix}$$

RLS Training

- Training sample: $\{[\phi(\mathbf{x}_i), d_i]\}_{i=1}^{N}$
- d_i = desired response for input $\mathbf{x}_i$
- Use RLS algorithm to find optimal output weights

Complete Algorithm - Step by Step

Step 1: Prepare Data

- Collect input samples $\{\mathbf{x}_i\}_{i=1}^N$
- Collect desired outputs $\{d_i\}_{i=1}^N$

Step 2: Train Hidden Layer (Unsupervised)

1. Choose number of clusters K
2. Apply K-means to $\{\mathbf{x}_i\}$ to find centers $\{\boldsymbol{\mu}_j\}$
3. Calculate width: $\sigma = \frac{d_{\max}}{\sqrt{2K}}$

Complete Algorithm - Step by Step (cont.)

Step 3: Compute Hidden Layer Outputs

For each training sample $\mathbf{x}_i$, compute:

$$\phi(\mathbf{x}_i) = \begin{bmatrix} \exp\left(-\frac{\|\mathbf{x}_i - \boldsymbol{\mu}_1\|^2}{2\sigma^2}\right) \\ \vdots \\ \exp\left(-\frac{\|\mathbf{x}_i - \boldsymbol{\mu}_K\|^2}{2\sigma^2}\right) \end{bmatrix}$$

Step 4: Train Output Layer (Supervised)

- Use RLS algorithm with training pairs $\{[\phi(\mathbf{x}_i), d_i]\}$
- Find optimal linear weights for output layer

Advantages of Hybrid Learning

Computational Efficiency

- K-means is computationally efficient
- RLS is computationally efficient
- Combined approach inherits both efficiencies

Simplicity

- Separates unsupervised and supervised learning
- Each step uses well-established algorithms
- Easy to implement and debug

Flexibility

- Parameter K provides design flexibility
- Can handle various input dimensionalities

Limitations

Main Limitation

No overall optimality criterion

- Hidden and output layers trained separately
- No guarantee of global optimum for entire network
- System optimality not assured in statistical sense

Practical Implications

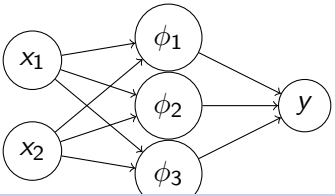
- May not achieve best possible performance
- Trade-off: efficiency vs. optimality
- Often "good enough" for many applications

Simple Example

Example

Problem: Function approximation with 2D input

- Input: $\mathbf{x} = [x_1, x_2]^T$
- Choose $K = 3$ clusters
- Apply K-means $\rightarrow$ find 3 centers
- Calculate σ from center distances
- Compute 3 Gaussian outputs for each input
- Use RLS to find weights for final output



Summary

Key Points

- **Hybrid approach**: K-means + RLS
- **Two-stage training**: Unsupervised then supervised
- **Efficient**: Both algorithms are computationally efficient
- **Practical**: Good trade-off between performance and complexity

Remember

- Choose K carefully (model selection)
- K-means finds RBF centers automatically
- Width calculated from center spread
- RLS provides optimal linear output weights

Kernel Regression and RBF Networks

- **RBF Networks**: Based on interpolation
- **Kernel Regression**: Based on density estimation
- Both approaches solve the same problem but from different perspectives

The Problem

Given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, find a function $f(\mathbf{x})$ that predicts y for new inputs $\mathbf{x}$.

The Nonlinear Regression Model

Basic Model

$$y_i = f(\mathbf{x}_i) + \varepsilon(i), \quad i = 1, 2, \ldots, N$$

Where:

- y_i: observed output
- $f(\mathbf{x}_i)$: unknown regression function we want to find
- $\varepsilon(i)$: additive white noise (zero mean, variance σ^2)

Key Insight

The unknown function $f(\mathbf{x})$ equals the **conditional mean** of y given $\mathbf{x}$:

$$f(\mathbf{x}) = E[y|\mathbf{x}]$$

The Mathematical Foundation

Conditional Mean Formula

$$f(\mathbf{x}) = E[y|\mathbf{x}] = \int_{-\infty}^{\infty} y\, p_{Y|X}(y|\mathbf{x})\, dy$$

Using probability theory:

$$p_{Y|X}(y|\mathbf{x}) = \frac{p_{X,Y}(\mathbf{x}, y)}{p_X(\mathbf{x})}$$

This leads to:

$$f(\mathbf{x}) = \frac{\int_{-\infty}^{\infty} y\, p_{X,Y}(\mathbf{x}, y)\, dy}{p_X(\mathbf{x})}$$

Problem

We don't know the probability densities $p_{X,Y}(\mathbf{x}, y)$ and $p_X(\mathbf{x})$!

Estimating Unknown Densities

Solution: Use Training Data

Estimate the densities using the Parzen-Rosenblatt method with kernel functions.

For marginal density $p_X(\mathbf{x})$:

$$\hat{p}_X(\mathbf{x}) = \frac{1}{Nh^{m_0}} \sum_{i=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

For joint density $p_{X,Y}(\mathbf{x}, y)$:

$$\hat{p}_{X,Y}(\mathbf{x}, y) = \frac{1}{Nh^{m_0+1}} \sum_{i=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) k\left(\frac{y - y_i}{h}\right)$$

Where:

- $k(\cdot)$: kernel function
- h: bandwidth (smoothing parameter)
- m_0: dimension of input vector $\mathbf{x}$

Properties of Parzen-Rosenblatt Estimator

Consistency Property

The estimator is **consistent** (asymptotically unbiased) if:

- $\lim_{N\to\infty} h(N) = 0$
- $\lim_{N\to\infty} E[\hat{p}_X(\mathbf{x})] = p_X(\mathbf{x})$

Intuition

- **Bandwidth** h: Controls smoothness
- **Large** h: Smoother estimate, more bias
- **Small** h: Less smooth, more variance
- As $N \to \infty$ and $h \to 0$ properly, we get the true density

Deriving the Kernel Regression Estimator

Substituting our density estimates into the regression formula:

$$F(\mathbf{x}) = \hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{N} y_i k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\sum_{j=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_j}{h}\right)}$$

Key Observation

This is a **weighted average** of the training outputs $\{y_i\}_{i=1}^{N}$!

The weights depend on how "close" the input $\mathbf{x}$ is to each training point $\mathbf{x}_i$.

Interpretation 1: Nadaraya-Watson Estimator

Normalized Weighting Function

$$W_{N,i}(\mathbf{x}) = \frac{k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}{\sum_{j=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_j}{h}\right)}, \quad i = 1, 2, \ldots, N$$

Properties:

- $\sum_{i=1}^{N} W_{N,i}(\mathbf{x}) = 1$ (weights sum to 1)
- $F(\mathbf{x}) = \sum_{i=1}^{N} W_{N,i}(\mathbf{x}) y_i$ (weighted average)

Named After

Nadaraya (1964) and Watson (1964) independently proposed this approach.

Interpretation 2: Normalized RBF Network

Assume **spherical symmetry** of the kernel:

$$k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) = k\left(\frac{\|\mathbf{x}-\mathbf{x}_i\|}{h}\right)$$

Normalized Radial Basis Function

$$\psi_N(\mathbf{x}, \mathbf{x}_i) = \frac{k\left(\frac{\|\mathbf{x}-\mathbf{x}_i\|}{h}\right)}{\sum_{j=1}^{N} k\left(\frac{\|\mathbf{x}-\mathbf{x}_j\|}{h}\right)}, \quad i = 1, 2, \ldots, N$$

Setting $y_i = w_i$ (weights), the regression function becomes:

$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i \psi_N(\mathbf{x}, \mathbf{x}_i)$$

This is exactly a **normalized RBF network**!

Properties of Normalized RBF

- $\sum_{i=1}^{N} \psi_N(\mathbf{x}, \mathbf{x}_i) = 1$ for all $\mathbf{x}$
- $0 \leq \psi_N(\mathbf{x}, \mathbf{x}_i) \leq 1$ for all $\mathbf{x}$ and $\mathbf{x}_i$

Probabilistic Interpretation

$\psi_N(\mathbf{x}, \mathbf{x}_i)$ can be interpreted as the **probability** of an event described by input vector $\mathbf{x}$, conditional on $\mathbf{x}_i$.

Key Difference

- **Regular RBF**: Basis functions can have any value
- **Normalized RBF**: Basis functions are probabilities (sum to 1)

Multivariate Gaussian Kernel

General Form

$$k(\mathbf{x}) = \frac{1}{(2\pi)^{m_0/2}} \exp\left(-\frac{\|\mathbf{x}\|^2}{2}\right)$$

Centered on data point $\mathbf{x}_i$ with bandwidth σ:

$$k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) = \frac{1}{(2\pi\sigma^2)^{m_0/2}} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma^2}\right)$$

Where h plays the role of smoothing parameter σ.

Nadaraya-Watson with Gaussian Kernels

Regression Estimator

$$F(\mathbf{x}) = \frac{\sum_{i=1}^{N} y_i \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_j\|^2}{2\sigma^2}\right)}$$

Normalized RBF Network

$$F(\mathbf{x}) = \frac{\sum_{i=1}^{N} w_i \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_j\|^2}{2\sigma^2}\right)}$$

The denominator is the Parzen-Rosenblatt density estimator: sum of N multivariate Gaussian distributions centered on the data points.

Key Takeaways

1. **Two viewpoints, same result**:
   - Interpolation $\rightarrow$ RBF Networks
   - Density estimation $\rightarrow$ Kernel Regression
2. **Kernel regression** estimates unknown probability densities using training data
3. **Nadaraya-Watson estimator** gives weighted averages of training outputs
4. **Normalized RBF networks** emerge naturally from kernel regression with spherically symmetric kernels
5. **Gaussian kernels** are widely used in practice
6. Centers of RBF functions typically coincide with training data points

Practical Implications

Advantages of Normalized RBF

- Probabilistic interpretation
- Weights always sum to 1
- More stable numerically
- Better generalization properties

Design Considerations

- **Bandwidth selection**: Critical for performance
- **Kernel choice**: Gaussian is most common
- **Center placement**: Usually at training points
- **Computational cost**: $O(N)$ evaluations per prediction

Thank You!