Slide05

Haykin Chapter 5: Radial-Basis Function Networks

CPSC 636-600

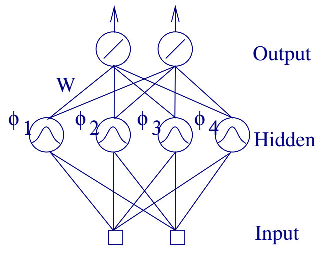
Instructor: Yoonsuck Choe

Spring 2012

Learning in MLP

- Supervised learning in multilayer perceptrons:
  - Recursive technique of *stochastic approximation*, e.g., backprop.
  - Design of nnet as a *curve-fitting (approximation) problem*, e.g., RBF.
- Curve-fitting:
  - Finding a surface in a multidimensional space that provides a best fit to the training data.
  - "Best fit" measured in a certain statistical sense.
  - RBF is an example: hidden neurons forming an arbitrary *basis* for the input patterns when they are expanded into the *hidden space*. These basis are called *radial basis functions*.

Radial-Basis Function Networks



Three layers:

- **Input**
- **Hidden**: *nonlinear* transformation from input to hidden space.
- **Output**: *linear* activation.

Principal motivation: *Cover's theorem*—pattern classifiction casted in high-dimensional space is more likely to be linearly separable than in low-dimensional space.

Cover's Theorem

Cover's theorem on the *separability of patterns*:

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

Basic idea: nonlinearly map points in the input space to a hidden space that has a higher dimension than the input space. Once the proper mapping is done, simple, quick algorithms can be used to find the separating hyperplane.

ϕ-Separability of Patterns

- N input patterns $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ in m_0-dimensional space.
- The inputs belong to either of two sets $\mathcal{X}_1$ and $\mathcal{X}_2$: they form a dichotomy.
- The dichotomy is *separable* wrt a **family of surfaces** if a surface exists *in the family* that separates the points in class $\mathcal{X}_1$ from $\mathcal{X}_2$.
- For each $\mathbf{x} \in \mathcal{X}$, define an m_1-vector $\{\phi_i(\mathbf{x}) | i = 1, 2, ..., m_1\}$:

$$\boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), ..., \phi_{m_1}(\mathbf{x})]^T$$

  that maps inputs in m_0-D space to the hidden space of m_1-D. $\phi_i(\mathbf{x})$ are called the *hidden functions*, and the space spanned by these functions is called the *hidden space* or *feature space*.
- A dichotomy is ϕ-*separable* if there exists an m_1-D vector $\mathbf{w}$ such that:

$$\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) > 0, \quad \mathbf{x} \in \mathcal{X}_1$$

$$\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) < 0, \quad \mathbf{x} \in \mathcal{X}_2$$

  with separating hyperplane $\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) = 0$.

Cover's Theorem Revisited

- Given a set $\mathcal{X}$ of N inputs picked from the input space independently, and suppose all the possible dichotomies of $\mathcal{X}$ are equiprobable.
- Let $P(N, m_1)$ denote the probability that a particular dichotomy picked at random is ϕ-separable, where the family of surfaces has m_1 degrees of freedom.
- In this case,

$$P(N, m_1) = \left(\frac{1}{2}\right)^{N-1} \sum_{m=0}^{m_1-1} \binom{N-1}{m}$$

  where

$$\binom{l}{m} = \frac{l(l-1)(l-2)...(l-m+1)}{m!}$$

Cover's Theorem: Interpretation

- Separability depends on: (1) particular dichotomy, and (2) the distribution of patterns in the input space.
- The derived $P(N, m_1)$ states that the probability of being ϕ-separable is equivalent to the *cumulative binomial distribution* corresponding to the probability that $(N-1)$ flips of a fair coin will result in $(m_1 - 1)$ or fewer heads.
- In sum, Cover's theorem has two basic ingredients:
  - Nonlinear mapping to hidden space with $\phi_i(\mathbf{x})$ $(i = 1, 2, .., m_1)$.
  - High dimensionality of hidden space compared to the input space $(m_1 > m_0)$.
- Corollary: A maximum of $2m_1$ patterns can be linearly separated by a hidden space of m_1-D.

Example: XOR (again!)

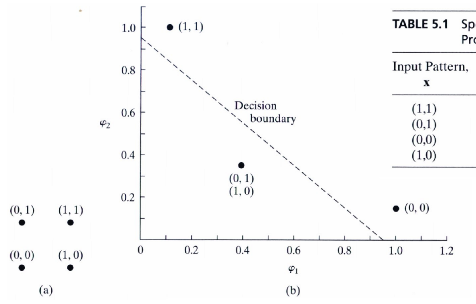


TABLE 5.1 Specification of the Hidden Functions for the XOR Problem of Example 5.1

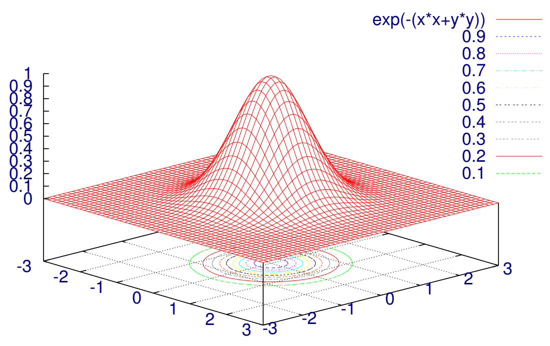
Input Pattern, $\mathbf{x}$	First Hidden Function, $\varphi_1(\mathbf{x})$	Second Hidden Function, $\varphi_2(\mathbf{x})$
(1,1)	1	0.1353
(0,1)	0.3678	0.3678
(0,0)	0.1353	1
(1,0)	0.3678	0.3678

- With Gaussian hidden functions, the inputs become linearly separable in the hidden space:

$$\phi_1(\mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{t}_1\|^2), \quad \mathbf{t}_1 = [1, 1]^T$$

$$\phi_2(\mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{t}_2\|^2), \quad \mathbf{t}_2 = [0, 0]^T$$

2D Gaussian

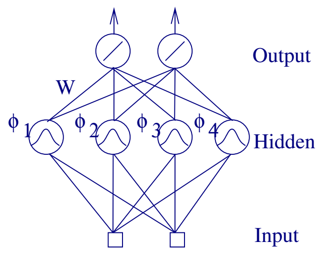


Basically, $\mathbf{t}_i$ determines the center of the 2D Gaussian,

$$\phi_i(\mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{t}_i\|^2)$$

and points $\mathbf{x}$ that are equidistance from the center have the same ϕ value.

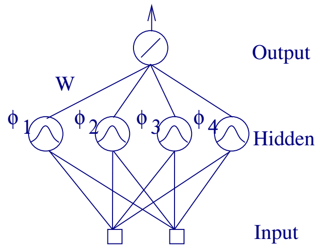
RBF Learning: Overview



The RBF learning problem boils down to two tasks:

1. How to determine the parameters associated with the radial-basis functions in the hidden layer $\phi_i(\mathbf{x})$ (e.g., the center of the Gaussians).
2. How to train the hidden-to-output weights?: This part is relatively easy.

RBF as an Interpolation Problem



- m_0-D input to 1-D output mapping $s : \mathbb{R}^{m_0} \rightarrow \mathbb{R}^1$.
- The map s can be thought of as a *hypersurface* $\Gamma \subset \mathbb{R}^{m_0+1}$.
  - **Training**: fit hypersurface Γ to the training data points.
  - **Generalization**: interpolate between data points, along the reconstructed surface Γ.
- Given $\{\mathbf{x}_i \in \mathbb{R}^{m_0} | i = 1, 2, ..., N\}$ and N labels $\{d_i \in \mathbb{R}^1 | i = 1, 2, ..., N\}$, find $F : \mathbb{R}^N \rightarrow \mathbb{R}^1$ such that

$$F(\mathbf{x}_i) = d_i \text{ forall } i.$$

RBF and Interpolation

- Interpolation is formulated as

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

  where $\{\phi(\|\mathbf{x} - \mathbf{x}_i\|) | i = 1, 2, ..., N\}$ is a set of N arbitrary (nonlinear) functions known as *radial-basis functions*.
- The *known data points* $\mathbf{x}_i \in \mathbb{R}^{m_0}, i = 1, 2, ..., N$ are treated as the *centers* of the RBFs. (Note that in this case, all input data need to be memorized, as in instance-based learning, but this is not a necessary requirement.)

RBF and Interpolation (cont'd)

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

- So, we have N inputs and N hidden units, and one output unit. Expressing everything (all N input-output pairs) in matrix form:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix},$$

where $\phi_{ji} = \phi(\| \underbrace{\mathbf{x}_j}_{Input} - \underbrace{\mathbf{x}_i}_{Center} \|^2)$. We can abbreviate the above as:

$$\boldsymbol{\phi}\mathbf{w} = \mathbf{d}.$$

RBF and Interpolation (cont'd)

- From $\boldsymbol{\phi}\mathbf{w} = \mathbf{d}$, we can find an explicit solution $\mathbf{w}$:

$$\mathbf{w} = \boldsymbol{\phi}^{-1}\mathbf{d},$$

assuming $\boldsymbol{\phi}$ is nonsingular.

(**Note:** in general, the number of hidden units is much less than the number of inputs, so we don't always have $\boldsymbol{\phi}$ as a square matrix! We'll see how to handle this, later.)

- Nonsingularity of $\boldsymbol{\phi}$ is guaranteed by **Micchelli's theorem**:

  Let $\{\mathbf{x}_i\}_{i=1}^{N}$ be a set of distinct points in $\mathbb{R}^{m_0}$. Then the N-by-N interpolation matrix $\boldsymbol{\phi}$, whose ji-th element is $\phi_{ji} = \phi(\|\mathbf{x}_j - \mathbf{x}_i\|)$, is nonsingular.

RBF and Interpolation (cont'd)

- When $m_0 < N$ (m_0: number of hidden units; N: number of inputs), we can find w that minimizes

$$\mathcal{E}(\mathbf{w}) = \sum_{i=1}^{N} (F(\mathbf{x}_i) - d_i)^2,$$

where $F(\mathbf{x}) = \sum_{k=1}^{m_0} w_k \phi_k(\mathbf{x})$.

- The solution involves the pseudo inverse of $\boldsymbol{\phi}$:

$$\mathbf{w} = \underbrace{\left(\boldsymbol{\phi}^T\boldsymbol{\phi}\right)^{-1}\boldsymbol{\phi}^T}_{\text{pseudo inverse}} \mathbf{d}.$$

Note: $\boldsymbol{\phi}$ is an $N \times m_0$ rectangular matrix.

- In this case, how to determine the centers of the $\phi_k(\cdot)$ functions becomes an issue.

Typical RBFs

For some $c > 0$, $\sigma > 0$, and $r \in \mathbb{R}$.

- Multiquadrics (non-local):

$$\phi(r) = (r^2 + c^2)^{1/2}$$

- Inverse multiquadrics (local):

$$\phi(r) = \frac{1}{(r^2 + c^2)^{1/2}}$$

- Gaussian functions (local):

$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$$

Other Perspectives on RBF Learning

- RBF learning is formularized as

$$F(\mathbf{x}) = \sum_{k=1}^{m_0} w_k \phi_k(\mathbf{x}).$$

- This kind of expression was given without much rationale, other than intuitive appeal.
- However, there's one way to derive the above formalism based on an interesting theoretical point-of-view, which we will see next.

Supervised Learning as Ill-Posed Hypersurface Reconstruction Problem

- The exact interpolation approach has limitations:
  - Poor generalization: data points being more numerous than the degree of freedom of the underlying process can lead to overfitting.
- How to overcome this issue?
  - Approach the problem from a perspective that learning is a *hypersurface reconstruction problem* given a *sparse set of data points*.
  - Contrast between *direct problem* (in many cases **well-posed**) vs. *inverse problem* (in many cases **ill-posed**).

Well-Posed Problems in Reconstructing Functional Mapping

Given an **unknown** mapping from domain $\mathcal{X}$ to range $\mathcal{Y}$, we want to reconstruct the mapping f. This mapping is **well-posed** if all the following conditions are satisfied:

- **Existence**: For all $\mathbf{x} \in \mathcal{X}$, there exist an output $y \in \mathcal{Y}$ such that $y = f(\mathbf{x})$.
- **Uniqueness**: For all $\mathbf{x}, \mathbf{t} \in \mathcal{X}$, $f(\mathbf{x}) = f(\mathbf{t})$ iff $\mathbf{x} = \mathbf{t}$.
- **Continuity**: The mapping is continuous.
  For any $\epsilon > 0$ there exists $\delta = \delta(\epsilon)$ such that $\rho_x(\mathbf{x}, \mathbf{t}) < \delta \rightarrow \rho_y(f(\mathbf{x}), f(\mathbf{t})) < \epsilon$, where $\rho(\cdot, \cdot)$ is the distance measure.

If any of these conditions are violated, the problem is called an **ill-posed** problem.

Ill-Posed Problems and Solutions

- Direct (causal) mapping are generally well-posed (e.g., 3D object to 2D projection).
- On the other hand, inverse problems are ill-posed (e.g., reconstructing 3D structure from 2D projections).
- For ill-posed problems, solutions are not unique (can in many cases they can be *infinite*): We need *prior knowledge* (or some kind of *preference*) to narrow down the range of solutions (this is called **regularization**).

Treating supervised learning as an ill-posed problem, and using certain prior knowledge, we can derive the RBF formalism.

Regularization Theory: Overview

- The main idea behind **regularization** is to *stabilize* the solution by means of **prior information**.
- This is done by including a **functional** (a function that maps from a function to a scalar) *in the cost function*, so that the functional can also be minimized. Only a small number of candidate solutions will minimize this functional.
- These functional terms are called the **regularization term**.
- Typically, the functionals measure the **smoothness** of the function.

Tikonov's Regularization Theory

- **Task**: Given input $\mathbf{x}_i \in \mathbb{R}^{m0}$ and target $d_i \in \mathbb{R}^1$, find $F(\mathbf{x})$.
- Minimize the sum of two terms:
  1. **Standard error term:**

$$\mathcal{E}_s(F) = \frac{1}{2}\sum_{i=1}^{N}(d_i - y_i)^2 = \frac{1}{2}\sum_{i=1}^{N}(d_i - F(\mathbf{x}_i))^2.$$

  2. **Regularization term:**

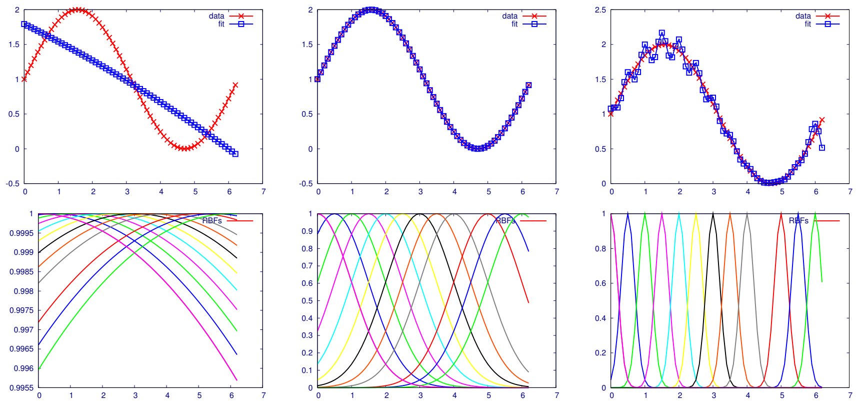
$$\mathcal{E}_c(F) = \frac{1}{2}\|\mathbf{D}F\|^2,$$

  where $\mathbf{D}$ is a *linear differential operator*, and $\|\cdot\|$ the *norm of the function space*.

- Putting these together, we want to **minimize** (w/ *regularization param.* λ)

$$\mathcal{E}(F) = \mathcal{E}_s(F) + \lambda\mathcal{E}_c(F) = \frac{1}{2}\sum_{i=1}^{N}(d_i - F(\mathbf{x}_i))^2 + \frac{1}{2}\lambda\|\mathbf{D}F\|^2.$$

Error Term vs. Regularization Term



	Error	Regularization
Left	Bad fit	Extremely smooth
Middle	Good fit	Smooth
Right	Over fit	Jagged

Try this demo: `http://lcn.epfl.ch/tutorial/english/rbf/html/`.

Solution that Minimizes $\mathcal{E}(F)$

- **Problem**: minimize

$$\mathcal{E}(F) = \mathcal{E}_s(F) + \lambda\mathcal{E}_c(F) = \frac{1}{2}\sum_{i=1}^{N}(d_i - F(\mathbf{x}_i))^2 + \frac{1}{2}\lambda\|\mathbf{D}F\|^2.$$

- **Solution**: $F_\lambda(\mathbf{x})$ that satisfies the *Euler-Lagrange equation* (below) minimizes $\mathcal{E}(F)$.

$$\widetilde{\mathbf{D}}\mathbf{D}F_\lambda(\mathbf{x}) - \frac{1}{\lambda}\left[d_i - F(\mathbf{x}_i)\right]\delta(\mathbf{x} - \mathbf{x}_i) = 0,$$

where $\widetilde{\mathbf{D}}$ is the *adjoint operator* of $\mathbf{D}$ and $\delta(\cdot)$ is the *Dirac delta function*.

Solution that Minimizes $\mathcal{E}(F)$ (cont'd)

- The solution to the *Euler-Lagrange equation* can be formulated in terms of the *Green's function* that satisfies:

$$\widetilde{\mathbf{D}}\mathbf{D}G(\mathbf{x},\mathbf{x}') = \delta(\mathbf{x},\mathbf{x}').$$

Note: the form of $G(\cdot,\cdot)$ depends on the particular choice of $\mathbf{D}$.

- Finally, the desired function $F_\lambda(\mathbf{x})$ that minimizes $\mathcal{E}(F)$ is:

$$F_\lambda(\mathbf{x}) = \frac{1}{\lambda}\sum_{i=1}^{N}[d_i - F(\mathbf{x}_i)]\,G(\mathbf{x},\mathbf{x}_i).$$

Solution that Minimizes $\mathcal{E}(F)$ (cont'd)

- Letting

$$w_i = \frac{1}{\lambda}[d_i - F(\mathbf{x}_i)], i = 1, 2, \ldots N$$

we can recast $F_\lambda(\mathbf{x})$ as

$$F_\lambda(\mathbf{x}) = \sum_{i=1}^{N} w_i G(\mathbf{x},\mathbf{x}_i).$$

- Plugging in input $\mathbf{x}_j$, we get

$$F_\lambda(\mathbf{x}_j) = \sum_{i=1}^{N} w_i G(\mathbf{x}_j,\mathbf{x}_i).$$

- Note the similarity to the RBF:

$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(\|\mathbf{x} - \mathbf{x}_i\|)$$

Solution that Minimizes $\mathcal{E}(F)$ (cont'd)

We can use a matrix notation:

$$\begin{aligned}
\mathbf{F}_\lambda &= [F_\lambda(\mathbf{x}_1), F_\lambda(\mathbf{x}_1), \ldots F_\lambda(\mathbf{x}_N)]^T \\
\mathbf{d} &= [d_1, d_2, \ldots, d_N]^T \\
\mathbf{G} &= \begin{bmatrix} G(\mathbf{x}_1,\mathbf{x}_1) & G(\mathbf{x}_1,\mathbf{x}_2) & \cdots & G(\mathbf{x}_1,\mathbf{x}_N) \\ G(\mathbf{x}_2,\mathbf{x}_1) & G(\mathbf{x}_2,\mathbf{x}_2) & \cdots & G(\mathbf{x}_2,\mathbf{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ G(\mathbf{x}_N,\mathbf{x}_1) & G(\mathbf{x}_N,\mathbf{x}_2) & \cdots & G(\mathbf{x}_N,\mathbf{x}_N) \end{bmatrix} \\
\mathbf{w} &= [w_1, w_2, \ldots, w_N]^T.
\end{aligned}$$

Then we can rewrite the formula in the previous page as $\mathbf{w} = \frac{1}{\lambda}(\mathbf{d} - \mathbf{F}_\lambda)$, and $\mathbf{F}_\lambda = \mathbf{G}\mathbf{w}$.

Solution that Minimizes $\mathcal{E}(F)$ (cont'd)

- Combining

$$\mathbf{w} = \frac{1}{\lambda}(\mathbf{d} - \mathbf{F}_\lambda)$$
$$\mathbf{F}_\lambda = \mathbf{G}\mathbf{w}$$

we can eliminate $\mathbf{F}_\lambda$ to get

$$(\mathbf{G} + \lambda\mathbf{I})\mathbf{w} = \mathbf{d}.$$

- From this, we can get the weights:

$$\mathbf{w} = (\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{d}$$

if $\mathbf{G} + \lambda\mathbf{I}$ is invertible (it needs to be positive definite, which can be ensured by a large λ).

Note: $G(\mathbf{x}_i,\mathbf{x}_j) = G(\mathbf{x}_j,\mathbf{x}_i)$, thus $\mathbf{G}^T = \mathbf{G}$.

Solution that Minimizes $\mathcal{E}(F)$ (cont'd)

Steps to follow:

1. Determine $\mathbf{D}$.
2. Find the Green's function matrix $\mathbf{G}$ associated with $\mathbf{D}$.
3. Obtain the weights by

$$\mathbf{w} = (\mathbf{G} + \lambda \mathbf{I})^{-1}\mathbf{d}$$

For certain $\mathbf{D}$s, the corresponding Green's functions are Gaussian, inverse multiquadrics, etc. Thus, the whole approach is similar to RBF. (More on this later.)

Regularization Theory and RBF

- In conclusion, the regularization problem's solution is given by the expansion:

$$F_\lambda(\mathbf{x}) = \sum_{i=1}^{N} w_i G(\mathbf{x}, \mathbf{x}_i),$$

where $G(\cdot, \cdot)$ is the Green's function for the self-adjoint operator $\widetilde{\mathbf{D}}\mathbf{D}$.

- When the stabilizer $\mathbf{D}$ has certain properties, the resulting Green's function $G(\cdot, \cdot)$ become *radial basis functions*:
  - $\mathbf{D}$ is *translationally invariant*:

$$G(\mathbf{x}, \mathbf{x}_i) = G(\mathbf{x} - \mathbf{x}_i).$$

  - $\mathbf{D}$ is *translationally and rotationally invariant*:

$$G(\mathbf{x}, \mathbf{x}_i) = G(\|\mathbf{x} - \mathbf{x}_i\|),$$

which is a *RBF*!

Translationally and Rotationally Invariant $\mathbf{D}$

- One example of a Green's function that corresponds to a translationally and rotationally invariant $\mathbf{D}$ is the **multivariate Gaussian** function:

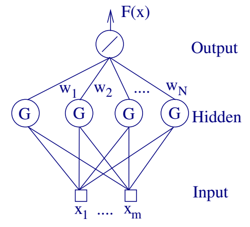
$$G(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\frac{1}{2\sigma_i^2}\|\mathbf{x} - \mathbf{x}_i\|^2\right).$$

- With this, the regularized solution becomes

$$F_\lambda(\mathbf{x}) = \sum_{i=1}^{N} w_i \exp\left(-\frac{1}{2\sigma_i^2}\|\mathbf{x} - \mathbf{x}_i\|^2\right).$$

This function is known to be a *universal approximator*.

Regularization Networks



- The regularized solution

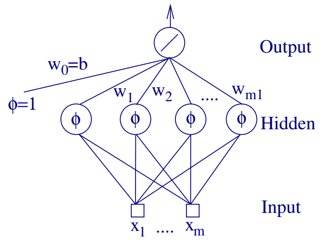
$$F_\lambda(\mathbf{x}) = \sum_{i=1}^{N} w_i \exp\left(-\frac{1}{2\sigma_i^2}\|\mathbf{x} - \mathbf{x}_i\|^2\right)$$

can be represented as a network.
  - Hidden unit i computes $G(\mathbf{x}, \mathbf{x}_i)$ (one hidden unit per input pattern).
  - Output unit is *linear* weighted sum of hidden unit activation.
- **Problem**: we need N hidden units for N input patterns, which can be excessive for large input sets.

Desirable Properties of Regularization Networks

- The regularization network is a **universal approximator**, that can approximate any multivariate continuous function arbitrarily well, given sufficiently large number of hidden units.
- The approximation shows the **best approximation** property (best coefficients will be found).
- The solution is **optimal**: it will minimize the cost functional $\mathcal{E}(\mathcal{F})$.

Generalized Radial-Basis Function Networks



- Regularization networks can be computationally demanding when N is huge. **Generalized RBF** overcomes this problem.

$$F^*(\mathbf{x}) = \sum_{i=1}^{m_1} w_i \phi_i(\mathbf{x}),$$

where $m_1 < N$, and $\phi_i(\mathbf{x}) = G(\|\mathbf{x} - \mathbf{t}_i\|)$, so that

$$F^*(\mathbf{x}) = \sum_{i=1}^{m_1} w_i G(\|\mathbf{x} - \mathbf{t}_i\|).$$

Generalized Radial-Basis Function Networks (cont'd)

- To find the weights w_i, we minimize

$$\mathcal{E}(F^*) = \sum_{i=1}^{N} \left(d_i - \sum_{i=1}^{m_1} w_i G(\|\mathbf{x} - \mathbf{t}_i\|) \right)^2 + \lambda \|\mathbf{D}F^*\|^2$$

- Minimization of $\mathcal{E}(F^*)$ yields

$$(\mathbf{G}^T\mathbf{G} + \lambda \mathbf{G}_0)\mathbf{w} = \mathbf{G}^T\mathbf{d}, \text{ where}$$

$$\mathbf{G}_0 = \begin{bmatrix} G(\mathbf{t}_1, \mathbf{t}_1) & G(\mathbf{t}_1, \mathbf{t}_2) & \cdots G(\mathbf{t}_1, \mathbf{t}_{m_1}) & \\ G(\mathbf{t}_2, \mathbf{t}_1) & G(\mathbf{t}_2, \mathbf{t}_2) & \cdots G(\mathbf{t}_2, \mathbf{t}_{m_1}) & \\ \vdots & \vdots & \vdots & \vdots \\ G(\mathbf{t}_{m_1}, \mathbf{t}_1) & G(\mathbf{t}_{m_1}, \mathbf{t}_2) & \cdots G(\mathbf{t}_{m_1}, \mathbf{t}_{m_1} & \end{bmatrix}.$$

- As λ approaches 0, we get $\mathbf{G}^T\mathbf{G}\mathbf{w} = \mathbf{G}^T\mathbf{d}$, so,

$$\mathbf{w} = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T\mathbf{d} = \mathbf{G}^+\mathbf{d}.$$

Weighted Norm

- When individual input lines in the input vector $\mathbf{x}$ are from different classes, a **weighted norm** can be used.

$$\|\mathbf{x}\|_C^2 = (\mathbf{C}\mathbf{x})^T(\mathbf{C}\mathbf{x}) = \mathbf{x}^T\mathbf{C}^T\mathbf{C}\mathbf{x}$$

- The approximation function can be rewritten as

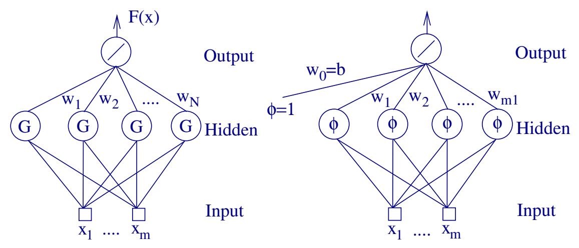
$$F^*(\mathbf{x}) = \sum_{i=1}^{m_1} w_i G(\|\mathbf{x} - \mathbf{t}_i\|_C).$$

- For Gaussian RBF, it can be interpreted as

$$\begin{aligned} G(\|\mathbf{x} - \mathbf{t}_i\|_C) &= \exp\left(-(\mathbf{x} - \mathbf{t}_i)^T\mathbf{C}^T\mathbf{C}(\mathbf{x} - \mathbf{t}_i)\right) \\ &= \exp\left(-\tfrac{1}{2}(\mathbf{x} - \mathbf{t}_i)^T\mathbf{\Sigma}^{-1}(\mathbf{x} - \mathbf{t}_i)\right), \end{aligned}$$

where $\mathbf{t}_i$ represents the mean vector and $\mathbf{\Sigma}$ the covariance matrix of a **multivariate Gaussian** function.

Regularization Networks vs. Generalized RBF



- Hidden layer in GRBF is much smaller: $m_1 < N$.
- In GRBF, (1) the weights, (2) the RBF centers $\mathbf{t}_i$, and (3) the norm weighting matrix are all unknown parameters to be determined.
- In regularization networks, RBF centers are known (same as all the inputs), and only the weights need to be determined.

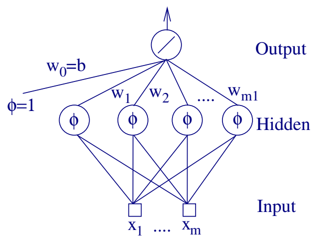
Estimating the Parameters

- Weights w_i: already discussed (more next).
- Regularization parameter λ:
  - Minimize averaged squared error.
  - Use generalized cross-validation.
- RBF centers:
  - Randomly select fixed centers.
  - Self-organized selection.
  - Supervised selection.

Estimating the Regularization Parameter λ

- Minimize **average squared error**: For a fixed λ, for all N inputs, calculate the squared error betwen the *true function value* and the estimated RBF network output using the λ. Find the optimal λ that minimizes this error. Problem: This requires knowledge of the true function values.
- **Generalized cross-validation**: Use leave-one-out cross validation. With a fixed λ, for all N inputs, find the difference between the target value (from the training set) and the predicted value from the leave-one-out-trained network. This approach depend only on the training set.

RBF Learning



Basic idea is to learn in two *different time scales*:

- Nonlinear, slow learning of the RBF parameters (center, variance).
- Linear, fast learning of the hidden-to-output weights.

RBF Learning (1/3): Random Centers

- Use m_1 hidden units:

$$G(\|\mathbf{x} - \mathbf{t}_i\|) = \exp\left(-\frac{m_1}{d_{\max}^2}\|\mathbf{x} - \mathbf{t}_i\|^2\right),$$

where $\mathbf{t}_i (i = 1, 2, ..., m_1)$ are picked by random from the available inputs $\mathbf{x}_j (j = 1, 2, ..., N)$.

- Note that the standard deviation (width) of the RBF is fixed to:

$$\sigma = \frac{d_{\max}}{\sqrt{2m_1}},$$

where $d_{\max}$ is the max distance between the chosen centers $\mathbf{t}_i$. This gives a width that is not too peaked nor too flat.

- The linear weights are learned using the pseudoinverse:

$$\mathbf{w} = \mathbf{G}^+\mathbf{d} = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T\mathbf{d},$$

where the matrix $\mathbf{G} = \{g_{ji}\}$, $g_{ji} = G(\|\mathbf{x}_j - \mathbf{t}_i\|^2)$.

Finding $\mathbf{G}^+$ with Singular Value Decomposition

If for a real $N \times M$ matrix $\mathbf{G}$, there exists orthogonal matrices

$$\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_N]$$

$$\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_M], \text{ such that}$$

$$\mathbf{U}^T\mathbf{G}\mathbf{V} = \mathrm{diag}(\sigma_1, \sigma_2, ..., \sigma_K) = \mathbf{\Sigma}, \quad K = \min(M, N),$$

then $\mathbf{U}$ is called the *left singular matrix*, $\mathbf{V}$ the *right singular matrix*, and $\sigma_1, \sigma_2, ..., \sigma_K$ the *singular values* of the matrix $\mathbf{G}$.

Once these are known, we can obtain $\mathbf{G}^+$ as

$$\mathbf{G}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T$$

where $\mathbf{\Sigma}^+ = \mathrm{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, ... \frac{1}{\sigma_K}\right)$. There are efficient algorithms for singular value decomposition that can be used for this.

Finding $\mathbf{G}^+$ with Singular Value Decomposition (cont'd)

Using these properties

$$\mathbf{U}^{-1} = \mathbf{U}^T$$
$$\mathbf{V}^{-1} = \mathbf{V}^T$$
$$\mathbf{\Sigma}\mathbf{\Sigma}^+ = \mathbf{I}$$

we can verify that $\mathbf{G}\mathbf{G}^+ = \mathbf{I}$:

$$\begin{aligned}
\mathbf{U}^T\mathbf{G}\mathbf{V} &= \mathbf{\Sigma} \\
\mathbf{U}\mathbf{U}^T\mathbf{G}\mathbf{V}\mathbf{V}^T &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \\
\mathbf{G} &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \\
\mathbf{G}\mathbf{G}^+ &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T \\
&= \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^+\mathbf{U}^T \\
&= \mathbf{U}\mathbf{U}^T \\
&= \mathbf{I}.
\end{aligned}$$

RBF Learning (2/3): Self-Organized Centers

The random-center approach is only effective with large input sets. To overcome this, we can take a hybrid approach: (1) self-organized learning of centers, and (2) supervised learning of linear weights.

Clustering for RBF center learning (similar to Self-Organizing Maps):

1. **Initialization**: Randomly choose distinct $\mathbf{t}_k(0)$s.
2. **Sampling**: Draw a random input vector $\mathbf{x} \in \mathcal{X}$.
3. **Similarity matching**: Find *best-matching* center vector $\mathbf{t}_{k(\mathbf{x})}$:

$$k(\mathbf{x}) = \arg\min_k \|\mathbf{x}(n) - \mathbf{t}_k(n)\|$$

4. **Updating**: Update center vectors

$$\mathbf{t}_k(n+1) = \begin{cases} \mathbf{t}_k(n) + \eta[\mathbf{x}(n) - \mathbf{t}_k(n)], & \text{if } k = k(\mathbf{x}) \\ \mathbf{t}_k(n), & \text{otherwise} \end{cases}$$

5. **Continuation**: increment n and repeat from step 2.

RBF Learning (3/3): Supervised Selection of Centers

Use *error correction learning* to adjust all RBF parameters to minimize the error cost function:

$$\mathcal{E} = \frac{1}{2}\sum_{j=1}^{N} e_j^2,$$

$$e_j = d_j - F^*(\mathbf{x}_j) = d_j - \sum_{i=1}^{M} w_i G(\|\mathbf{x}_j - \mathbf{t}_i\|_{C_i}).$$

- **Linear weights** (output layer): $w_i(n+1) = w_i(n) - \eta_1 \frac{\partial \mathcal{E}(n)}{\partial w_i(n)}$.
- **Position of centers** (hidden layer): $\mathbf{t}_i(n+1) = \mathbf{t}_i(n) - \eta_2 \frac{\partial \mathcal{E}(n)}{\partial \mathbf{t}_i(n)}$
- **Spread of centers** (hidden layer):

$$\mathbf{\Sigma}_i^{-1}(n+1) = \mathbf{\Sigma}_i^{-1}(n) - \eta_3 \frac{\partial \mathcal{E}(n)}{\partial \mathbf{\Sigma}_i^{-1}(n)}$$

RBF Learning (3/3): Supervised Selection of Centers (cont'd)

- Linear weights

$$\frac{\partial \mathcal{E}(n)}{\partial w_i(n)} = \sum_{j=1}^{N} e_j(n) G(\|\mathbf{x}_j - \mathbf{t}_i(n)\|_{C_i}).$$

- Position of centers

$$\frac{\partial \mathcal{E}(n)}{\partial \mathbf{t}_i(n)} = 2w_i(n)\sum_{j=1}^{N} e_j(n) G'\left(\|\mathbf{x}_j - \mathbf{t}_i(n)\|_{C_i}\right)\mathbf{\Sigma}_i^{-1}(n)\left[\mathbf{x}_j - \mathbf{t}_i(n)\right]$$

- Spread of centers

$$\frac{\partial \mathcal{E}(n)}{\partial \mathbf{\Sigma}_i^{-1}(n)} = -w_i(n)\sum_{j=1}^{N} e_j(n) G'\left(\|\mathbf{x}_j - \mathbf{t}_i(n)\|_{C_i}\right)\mathbf{Q}_{ij}(n),$$

$$\mathbf{Q}_{ji}(n) = [\mathbf{x}_j - \mathbf{t}_i(n)][\mathbf{x}_j - \mathbf{t}_i(n)]^T$$

Comparison of RBF and MLP

- RBF has a single hidden layer, while MLP can have many.
- In MLP, hidden and output neurons have the same underlying function. In RBF, they are specialized into distinct functions.
- In RBF, the output layer is linear, but in MLP, all neurons are nonlinear.
- The hidden neurons in RBF calculate the Euclidean norm of the input vector and the center, while in MLP the inner product of the input vector and the weight vector is calculated.
- MLPs construct global approximations to nonlinear input–output mapping. RBF uses exponentially decaying localized nonlinearities (e.g. Gaussians) to construct local approximations to nonlinear input–output mappings.

Summary

- RBF network is unusual due to its two different unit types: RBF hidden layer, and linear output layer.
- RBF is derived in a principled manner, starting from Tikohonov's regularization theory, unlike MLP.
- In RBF, the smoothing term becomes important, with different $\mathbf{D}$ giving rise to different Green's function $G(\cdot,\cdot)$.
- Generalized RBF lifts the requirement of N hidden units for N input patterns, greatly reducing the computational complexity.
- Proper estimation of the regularization parameter λ is needed.