Radial-Basis Functions and Self Organization

Federico Baldassarre, Diego González Morín, Lucas Rodés Guirao
KTH Royal Institute of Technology

I. Introduction

In this lab we use Radial Basis Functions (RBF) to approximate some simple functions of one variable. Suppose we have the function $f : \mathbb{R} \to \mathbb{R}$. RBF introduces a hidden layer such that $\hat{f} : \mathbb{R} \to \mathbb{R}^n \to \mathbb{R}$, where n is the number of neurons in the hidden layer. The trick basically consists on mapping an input $x \in \mathbb{R}$ to a new space $\mathbb{R}^n$ using a set of functions $\{\phi_i\}_{i=1}^n$ and then back to $\mathbb{R}$.

The functions used are Gaussians with different means and, possibly, also different variances

$$\phi_i(x) = \frac{\exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i}\right)}{\sum_i \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i}\right)}.$$

Radial comes from the fact that the functions $\{\phi_i\}_{i=1}^n$ operate on distances rather than on the input points themselves. In this regard, the selection of $\{\mu_i\}_{i=1}^n$ is essential and is typically done using K-means or by simply selecting some points from the training set.

Given an input x the output of the network is

$$\hat{f}(x) = \sum_{i=1}^{n} w_i \phi_i(x).$$

Thus we can say that the units in the hidden layer work as *basis* in which the function $\hat{f}$ can be expressed.

The motivation behind this technique is the fact that in higher dimensional spaces, data is usually linearly separable.

Suppose we have a set of patterns $\{x_1, \ldots, x_N\}$ and their corresponding real function values $\{f_1, \ldots, f_N\}$. While training, the neural network minimizes the error measure

$$total\ error = \sum_{k=1}^{N} (\hat{f}_k - f_k)^2.$$

A. Computing the weight matrix

The weights of the network are found by solving the system in (1).

$$\begin{aligned}
\phi_1(x_1)w_1 + \phi_2(x_1)w_2 + \cdots + \phi_n(x_1)w_n &= f_1 \\
\phi_1(x_2)w_1 + \phi_2(x_2)w_2 + \cdots + \phi_n(x_2)w_n &= f_2 \\
&\vdots \\
\phi_1(x_k)w_1 + \phi_2(x_k)w_2 + \cdots + \phi_n(x_k)w_n &= f_k \\
&\vdots \\
\phi_1(x_N)w_1 + \phi_2(x_N)w_2 + \cdots + \phi_n(x_N)w_n &= f_N
\end{aligned} \tag{1}$$

Questions

- *What is the lower bound for the number of training examples, N?*
  From basic linear algebra, we need at least n equations in a system with n variables. Otherwise, the system is incompatible. Hence, we have that $N \geq n$.
- *What happens with the error if $N = n$? Why?*
  If this is the case, and the system is full-rank, we have that we can find a set of weights $\{w_i\}_{i=0}^n$ such that we perfectly reconstruct the target function, i.e. $\hat{f}_k = f_k$ for all the training samples. Thus we can decrease the error to zero.
- *Under what conditions, if any, does (1) have a solution in this case?*
  This happens if the rank per columns and the rank per rows is the same.
- *During training we use an error measure defined over the training examples. Is it good to use this measure when evaluating the performance of the network? Explain!*
  Although we can make the error zero, it does not mean that the network will perform well with unseen data. On the contrary, it might mean that we are overfitting the network and thus it is probable that it will have a poor generalization.

B. Least Squares

We write (1) as

$$\mathbf{\Phi w} = \mathbf{f},$$

where

$$\mathbf{\Phi} = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \dots & \phi_n(x_N) \end{pmatrix}$$

and

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}. \tag{2}$$

Now our error measure becomes $total\ error = ||\mathbf{\Phi w} - \mathbf{f}||^2$, which is minimized by solving $\mathbf{\Phi}^T\mathbf{\Phi w} = \mathbf{\Phi f}$, i.e.

$$\mathbf{w}_{\text{opt}} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{f}.$$

C. The Delta Rule

Sometimes, not all the sample patterns are accessible simultaneously. If the network operates on a continuous stream of data, the process of computing the weights changes from the previous case. We now define our error using the expectation

$$\xi = expected\ error = \mathbf{E}\Big[\frac{1}{2}(f(x) - \hat{f}(x))^2\Big].$$

However, since we cannot find an exact expression for ξ we use the instantaneous error as an estimate of ξ, i.e.

$$\xi \approx \hat{\xi} = \frac{1}{2}(f(x) - \hat{f}(x))^2 = \frac{1}{2}e^2.$$

Our goal here is to make $\hat{\xi} \to 0$ as fast as possible. To do so, we take a step $\Delta\mathbf{w}$ in the opposite direction of the gradient of the error surface, i.e.

$$\begin{aligned} \Delta\mathbf{w} &= -\eta\nabla_{\mathbf{w}}\hat{\xi} \\ &= -\eta\frac{1}{2}\nabla_{\mathbf{w}}(f(x_k) - \mathbf{\Phi}(x_k)\mathbf{w})^2 \\ &= \eta(f(x_k) - \mathbf{\Phi}(x_k)^T\mathbf{w})\mathbf{\Phi}(x_k) \\ &= \eta e\mathbf{\Phi}(x_k) \end{aligned}$$

where

$$\mathbf{\Phi}(x_k) = \begin{pmatrix} \phi_1(x_k) \\ \phi_2(x_k) \\ \vdots \\ \phi_n(x_k) \end{pmatrix}.$$

η is known as the learning rate constant and is essential in order to find a good solution. However its tunining is tricky since depending on its value we might never approach a good solution to the system. In particular, if η is too high, noise on the input data might lead to error increase in $\mathbf{w}$ instead. However, if η is too small we might never achieve the optimum (least squares solution). [1]

References

[1] Amitai Etzioni. Less is more: The moral virtue of policy minimalism. *Journal of Globalization Studies*, 2(1), 2011.