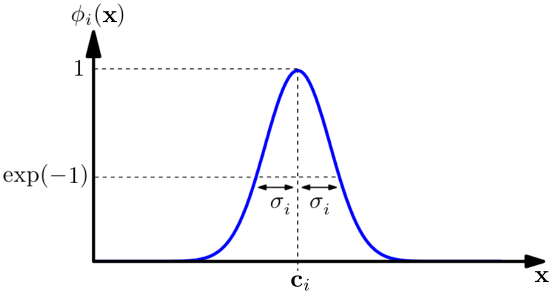
Radial Basis Function Networks

Olivier Sigaud

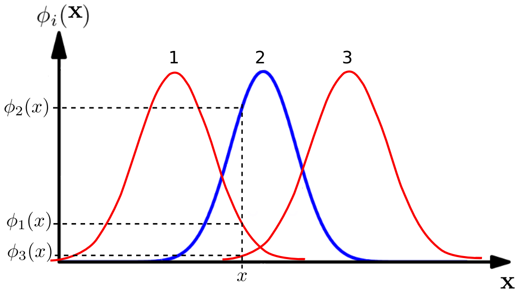
Sorbonne Université
http://people.isir.upmc.fr/sigaud

The Gaussian function



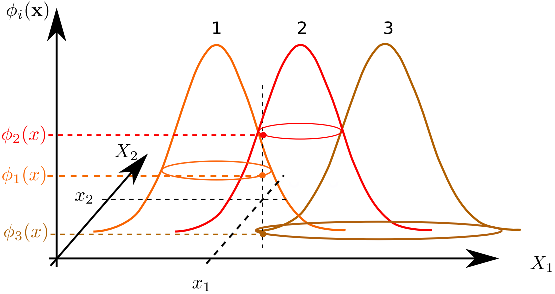
- Gaussian functions $\phi_i(\mathbf{x}) = \exp(-\frac{(\mathbf{x}-\mathbf{c}_i)^2}{\sigma_i^2})$
- Almost equal to zero everywhere except in a neighborhood of $\mathbf{c}_i$
- $\mathbf{c}_i$ is the "center" of the Gaussian
- The value of σ_i determines how large this neighborhood is.

Radial Basis Function Networks Projection (1D)



- The 1D input space (x) is projected into a 3D output space ($\phi_1(x)$, $\phi_2(x)$, $\phi_3(x)$).

Radial Basis Function Networks Projection (2D)



- When $\mathbf{x}$ is 2D, the output of each feature is still 1D
- Thus the output space $(\phi_1(x), \phi_2(x), \phi_3(x))$ is still 3D

Python code for a vector of Gaussians

```
class Gaussians:
    def __init__(self, nb_features):
        self.nb_features = nb_features
        self.centers = np.linspace(0.0, 1.0, self.nb_features)
        width_constant = 0.1 / self.nb_features
        self.sigma = np.ones(self.nb_features, ) * width_constant
```

The phi_output(x) python function

```python
def phi_output(self, x):
    """
    Get the output of the Gaussian features for a given input x of size N
    The output is a vertical vector. If one wants a standard vector,
    one needs to transpose it and take the first element of the result.

    :param x: A single or vector of dependent variables of size N
    :returns: A vector of feature outputs of size nb_features
    """
    if not hasattr(x, "__len__"):
        x = np.array([x])
    # number of dimensions in x (=N)
    dim_x = np.shape(x)[0]
    # Repeat vectors to vectorize output calculation
    input_mat = np.array([x, ] * self.nb_features)
    centers_mat = np.array([self.centers, ] * dim_x).transpose()
    widths_mat = np.array([self.sigma, ] * dim_x).transpose()
    phi = np.exp(-np.divide(np.square(input_mat - centers_mat), widths_mat))
    return phi
```

Perspective 1

- Project from the input space $\mathbf{x}$ into the feature space $\phi(\mathbf{x})$
- Then perform the standard linear least square calculation in this projected space.

$$\boldsymbol{\theta}^* = (\mathbf{G}^\intercal \mathbf{G})^{-1} \mathbf{G}^\intercal \mathbf{y}, \tag{1}$$

- where G is the Gram matrix obtained from the `phi_output(x)` python function
- This is the easiest approach to code in `python`.

Perspective 2

- Minimize the squared residual error:

$$\epsilon(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} (y^{(i)} - f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))^2.$$

- To get a local minimum over $\boldsymbol{\theta}$ of the function $\epsilon(\boldsymbol{\theta})$, we need to solve $\nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}) = 0$.
- To compute the gradient, we use $\nabla(g^2) = 2g\nabla g$.
- Therefore, we have

$$\nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - f_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)}\right) \right) \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}).$$

- Since $f_{\boldsymbol{\theta}}(\mathbf{x}) = \phi(\mathbf{x})^\mathsf{T} \boldsymbol{\theta}$, we have $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = \phi(\mathbf{x}^{(i)})$ and we get

$$\nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \phi(\mathbf{x}^{(i)})^\mathsf{T} \boldsymbol{\theta} \right) \phi(\mathbf{x}^{(i)})$$

Perspective 2 (continued)

- To make the gradient $\nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}) = 0$, we get:

$$\frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \phi(\mathbf{x}^{(i)})^{\mathsf{T}} \boldsymbol{\theta} \right) \phi(\mathbf{x}^{(i)}) = 0$$

$$\frac{1}{N} \sum_{i=1}^{N} \left(\phi(\mathbf{x}^{(i)}) y^{(i)} - \phi(\mathbf{x}^{(i)}) \phi(\mathbf{x}^{(i)})^{\mathsf{T}} \boldsymbol{\theta} \right) = 0$$

$$\left(\sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) \phi(\mathbf{x}^{(i)})^{\mathsf{T}} \right) \boldsymbol{\theta} = \sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) y^{(i)}.$$

- By setting $\mathbf{A} = \left(\sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) \phi(\mathbf{x}^{(i)})^{\mathsf{T}} \right)$ and $\mathbf{b} = \sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) y^{(i)}$,
- We get $\mathbf{A}\boldsymbol{\theta} = \mathbf{b}$.
- Matrix $\mathbf{A}$ is not necessarily invertible, and the general solution is obtained as $\boldsymbol{\theta} = A^{\sharp} b$, by using either the "pseudo-inverse" $A^{\sharp}$ (`np.linalg.pinv(A)`) or using `theta = np.linalg.solve(A,b)`.

The RBFN python class

```
class RBFN(Gaussians):
    def __init__(self, nb_features):
        super().__init__(nb_features)
        self.theta = np.random.random(self.nb_features)
        self.a = np.zeros(shape=(self.nb_features, self.nb_features))
        self.a_inv = np.matrix(np.identity(self.nb_features))
        self.b = np.zeros(self.nb_features)
```

The `f()` python function

```
def f(self, x, theta=None):
    """
    Get the FA output for a given input vector

    :param x: A vector of dependent variables of size N
    :param theta: A vector of coefficients to apply to the features.
    :If left blank the method will default to using the trained thetas in self.theta.

    :returns: A vector of function approximator outputs with size nb_features
    """
    if not hasattr(theta, "__len__"):
        theta = self.theta
    value = np.dot(self.phi_output(x).transpose(), theta.transpose())
    return value
```

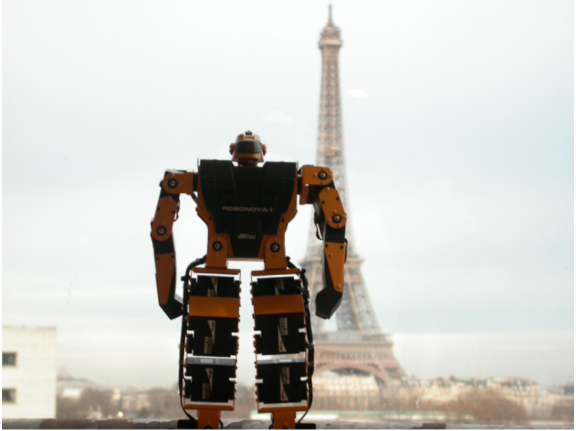
The `feature()` python function

```python
def feature(self, x, idx):
    """
     Get the output of the idx^th feature for a given input vector
     This is function f() considering only one feature
     Used mainly for plotting the features

     :param x: A vector of dependent variables of size N
     :param idx: index of the feature

     :returns: the value of the feature for x
     """
    phi = self.phi_output(x)
    return phi[idx] * self.theta[idx]
```

Any question?



Send mail to: `Olivier.Sigaud@upmc.fr`

ISIR
INSTITUT DES SYSTÈMES INTELLIGENTS ET DE ROBOTIQUE