Neuroengineering (I)
3. Radial Basis Function Networks

- **Scuola di Ingegneria Industriale e dell'Informazione**
  – Politecnico prior di Milano
- Prof. Pietro Cerveri

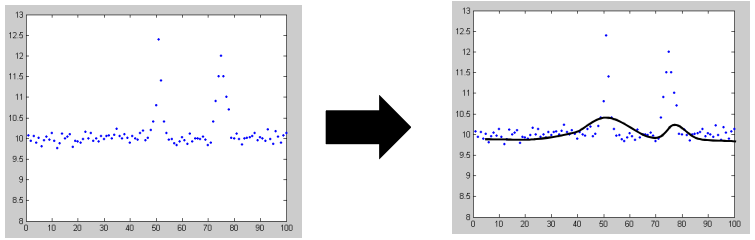
RBFN

function approximation, after learning the parameters I'm able to compute the function in any point, I have a continous version of my function

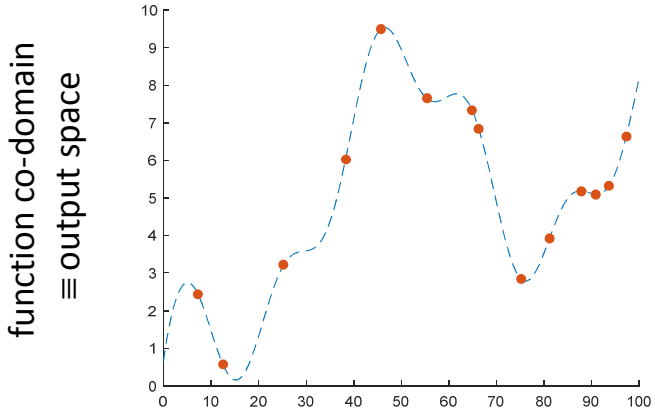
function modeling

classification

time series prediction



Reconstructing the function (continuous) from range data



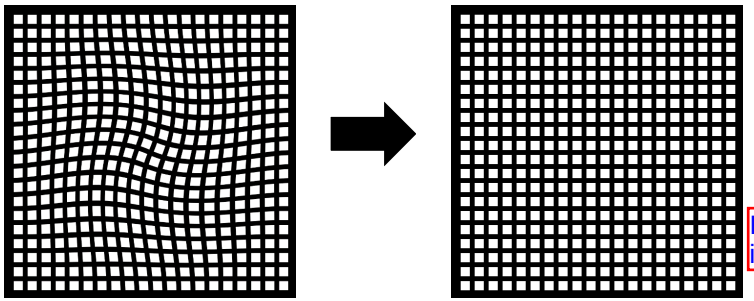
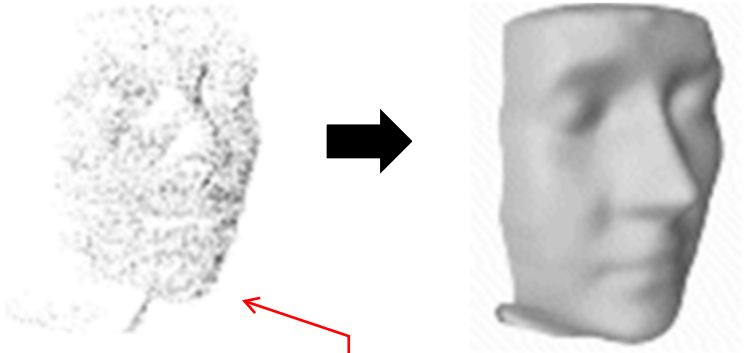


Image mapping from image deformation.



reconstruction of data

Applications in: clinical decision support systems, image/video 2D/3D processing, antenna data processing in communication systems, air pollutant forecasting, ground water level forecasting, exchange rates forecasting, anti-money laundering strategies, tourist appreciation prediction, …..

- Use of **Machine Learning Classifiers** and Sensor Data to Detect Neurological Deficit in Stroke Patients.

J Med Internet Res. 2017

- Prediction of Human intestinal absorption of compounds using **artificial intelligence techniques**.

Curr Drug Discov Technol. 2017

- Risk prediction for portal vein thrombosis in acute pancreatitis using **radial** basis **function**.

Ann Vasc Surg. 2017

- Biomimetic Hybrid Feedback Feedforward Neural-**Network** Learning Control.

IEEE Trans Neural Netw Learn Syst. 2017

- A **clinical decision support system** for prediction of pregnancy outcome in pregnant women with systemic lupus erythematosus.

Int J Med Inform. 2017

- Incorporating efficient **radial basis function** networks and significant amino acid pairs for predicting GTP binding sites in transport proteins.

BMC Bioinformatics. 2016

- Application of **Radial** Basis **Function Network** Tool for Correlation of CD4+ Count with Plasma Viral Load in HIV-Seropositive Individuals.

J Clin Diagn Res. 2016

- Lung Cancer Classification Employing Proposed Real Coded Genetic Algorithm Based **Radial Basis Function** Neural **Network** Classifier.

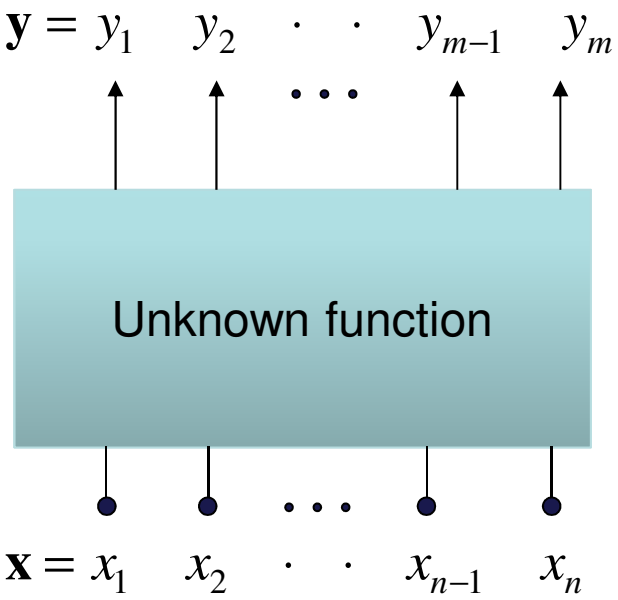
Comput Math Methods Med. 2016;

I have a multidimensional input space, and I'd like to map it into a codomain. I have the size of the input, and I'm moving to a space in m dimensions. I don't know the analityical function, I'm just able to measure it in specific points.I don't know the freq. distribution of the function (?). [28.00] noise??!?! LPfilter??!
I have couples of input/output quontities, I can build a sort of error function and exploit it as a classical way to learning.

Function approximation

$$\mathbf{y} = f(\mathbf{x}) \quad \mathbf{x} \in X \subset R^n \qquad \mathbf{y} \in Y \subset R^m$$

- Describing a continuous function
  - Finite set of data (measured)
    - » $(\mathbf{x},\mathbf{y})_i$ with i=1:k
  - Data are affected by noise
  - Data are not equally spaced in the input domain
  - The complexity of the function is not known a priori
    - » The available sampling can be only partially representative of the overall function
    - » different regions of the input space (function domain) can be affected by noise with different statistics
  - Approximation criterion (Least-square)
  - How to select f
  - How to measure the quality of the approximation



$$\mathrm{H}(f) = \sum_{i}^{k} \|f(\mathbf{x}_i) - \mathbf{y}_i\|^2$$

$\min \mathrm{H}$

You can ofcourse approximate a function with FNN. You have hidden layers with sigmoidal units and you get a combination of these. You have to assets both the weights of the hidden layer and the output layer, and you can use the backpropagation rule.

FFNN for function approximation

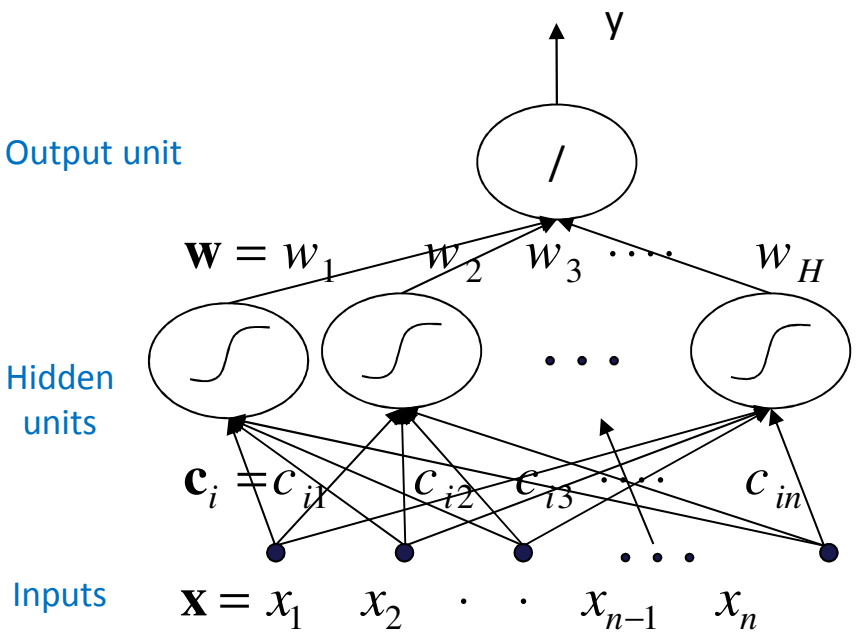
- Feed-forward neural networks with a single hidden layer of sigmoidal units are capable of approximating uniformly any continuous multi-variate function, to any desired degree of accuracy
  - Hornik et al., 1989, "Multilayer feedforward networks are universal approximators", Neural Networks, 2(5), 359-366.

Non-linear model in the weights

$$h_i(\mathbf{x}) = sigmoid\left(\sum_j^n c_{ij} x_j\right)$$

$$y = f(\mathbf{x}) = \sum_{i=1}^{H} w_i h_i(\mathbf{x})$$

$$\mathbf{x} \subset R^n \qquad y \subset R$$



I change the activation function using a radial basis function (it features radial symmetry, like gaussians). There by construction my output is the summations of the output of the neurons, and I have a depandance of the output just by the weigts of the output, I don't need to specify the weights of the hidden neurons, because **I can setup directly the construction of the activation function! This is a big advantage, this type of network can be trained with a closed form solution, we won't need anymore the backword propagations, we can find a direct approach and solve the equation like a linear system.**

Radial basis function network

- Feed-forward neural networks with a single hidden layer of sigmoidal units are capable of approximating uniformly any continuous multi-variate function, to any desired degree of accuracy
  - Hornik et al., 1989, "Multilayer feedforward networks are universal approximators", Neural Networks, 2(5), 359-366.
- Like feedforward NN with a single hidden layer of sigmoidal units, radial basis function (RBF) networks are universal approximators
  - Park and Sandberg, 1991, "Universal approximation using radial-basis-function networks", Neural Computation 3(2), 246-257).

Linear model in the weights

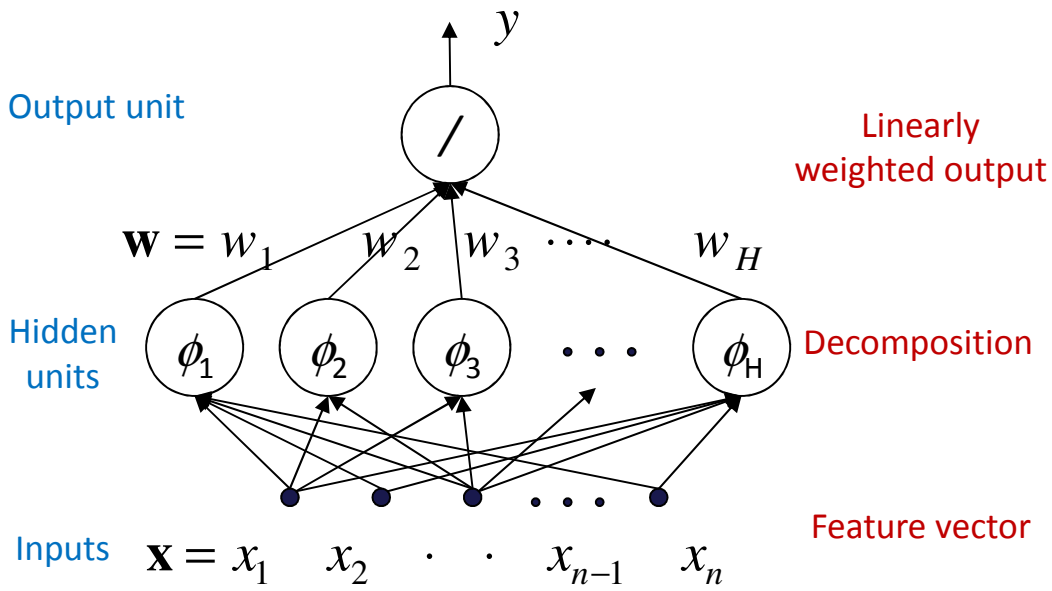
weights

$$y = f(\mathbf{x}) = \sum_{i=1}^{H} w_i \phi_i(\mathbf{x})$$

Basis radial functions

$$\mathbf{x} \subset R^n \qquad y \subset R$$

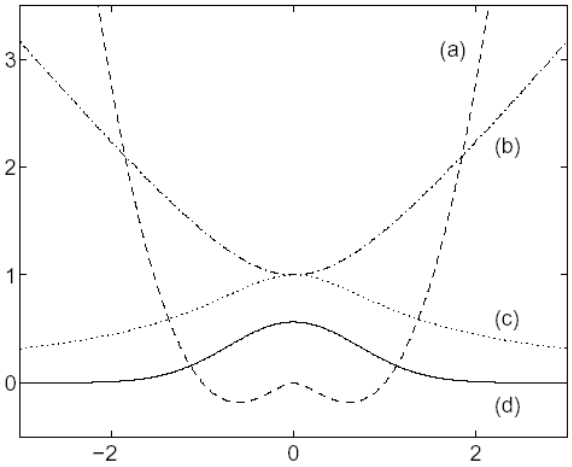
Output linear combination of gaussians



The weights simply modulate the amplitude of my function

Basic advatange of radial simmetry: I have a split of the support of the function, the region in the domain on which the function is active. If I'm using agaussian function I can expect maximal contribution close to the centrum, which is decreasing as I move away from the centrum. Also I can say that the external points are very much less useful, so I can chose a finite part of the gaussian. So the gaussian affects points that are in its subdomain. It's an advantage, you can place your gaussian in 2D, and you can change the STD, and the sensitivity of the function and it affects the support of the support. Increasing the STD I'm increasing the support. In the opposite I'm making the function more selective.

Radial symmetry



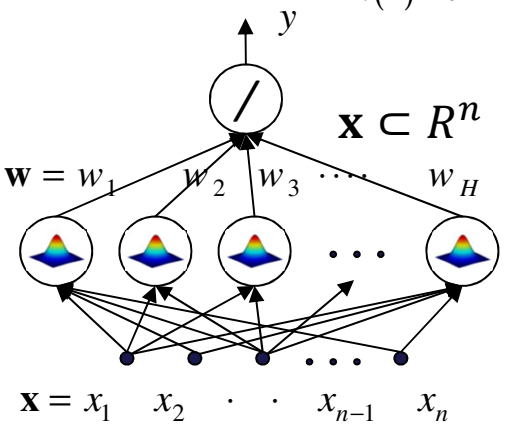
$$\phi_i(\mathbf{x}) = \phi\left(\left\|\mathbf{x} - \mathbf{x}_i\right\|\right)$$

a) Thin plate spline $h(r) = r^2 \log(r)$

b) Multi-quadrics $h(r) = \left(c^2 + r^2\right)^{\beta} \quad 0 < \beta < 1$

c) Inverse multi-quadrics $h(r) = \left(c^2 + r^2\right)^{-\alpha} \quad 0 < \alpha$

d) Gaussian $h(r) = e^{-r^2}$

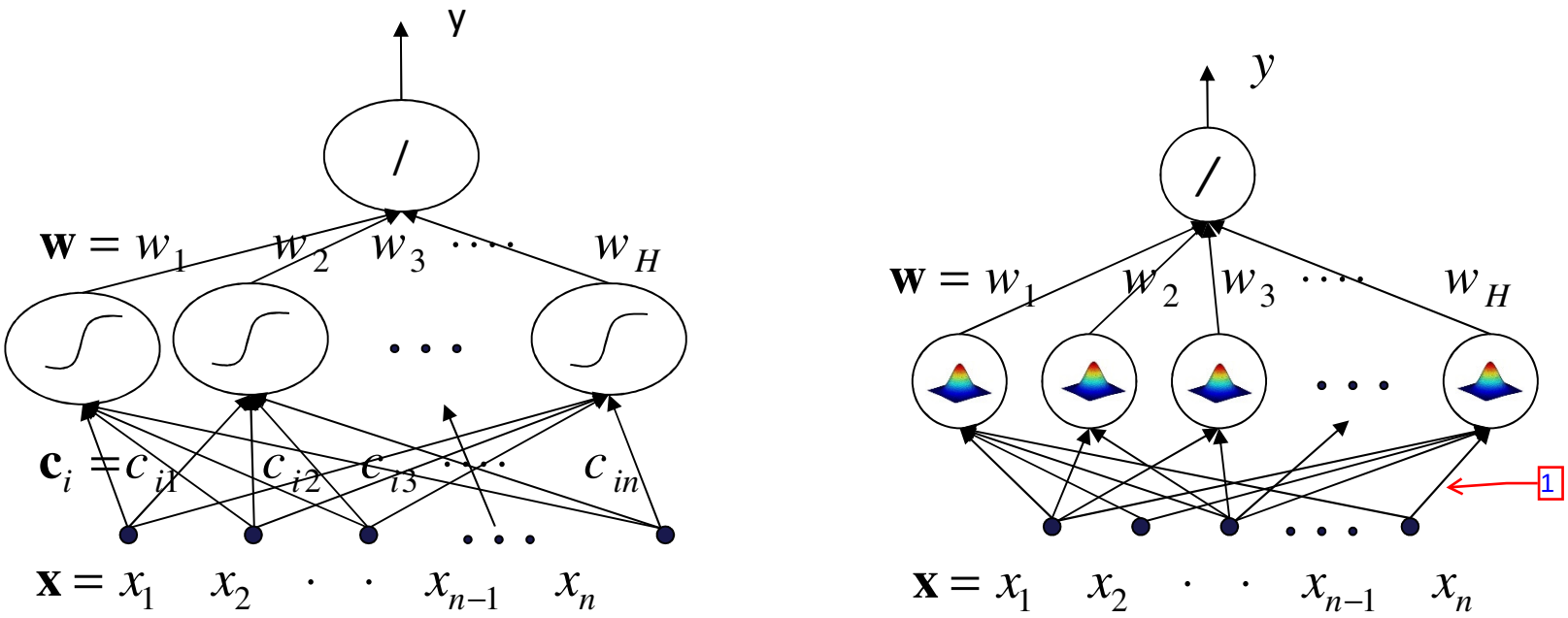


Four parameters for a radial function

- Shape ϕ
- Center $\mathbf{x}_i$
- Distance measure $r = \left\|\mathbf{x} - \mathbf{x}_i\right\|$ → A hidden neuron is more sensitive to data points near its center
- Spread (spatial support) → This sensitivity may be tuned by adjusting the spread, where a larger spread implies less sensitivity.

If I have a knowledge about the center and STD of the gaussian I can avoid to consider weights in (1)

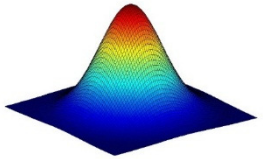
Look at the difference

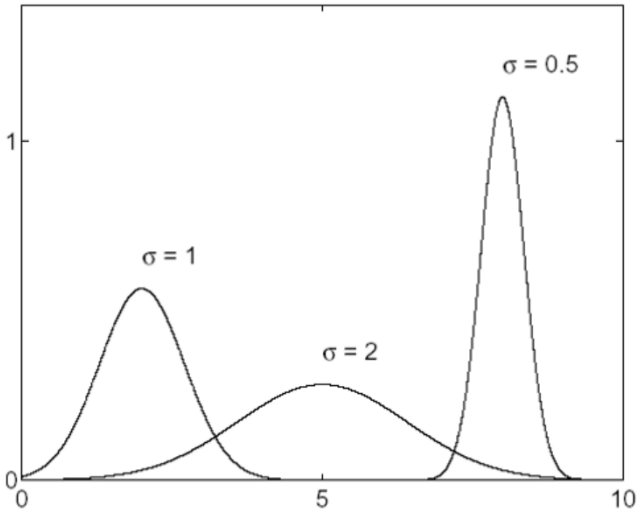


Setting the shape of the radial functions allows avoiding the computation of weights $\mathbf{c}_i$

Typical formal model to describe a gaussian function where changing the shape the integral is still 1. Actually we use a definition a little bit different. Here if I'm moving the std towards zero I obtain a pulse (the dirac function), we'd like that when we are chosing or modifing the STD we'd like to maintain the basic amplitude of the gaussian, because the amplitude is determined by the weights!!

Gaussian function



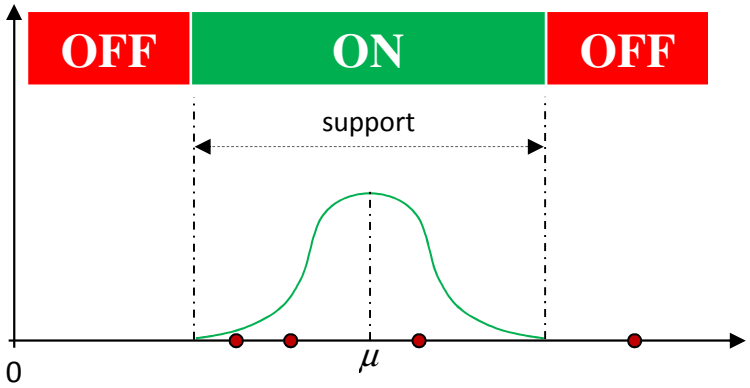


$$\phi(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Integral is 1

mean value μ (center) and standard deviation σ

σ is a measure of how spread the curve is

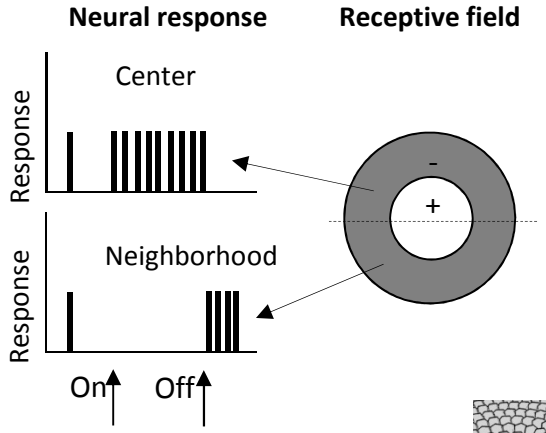


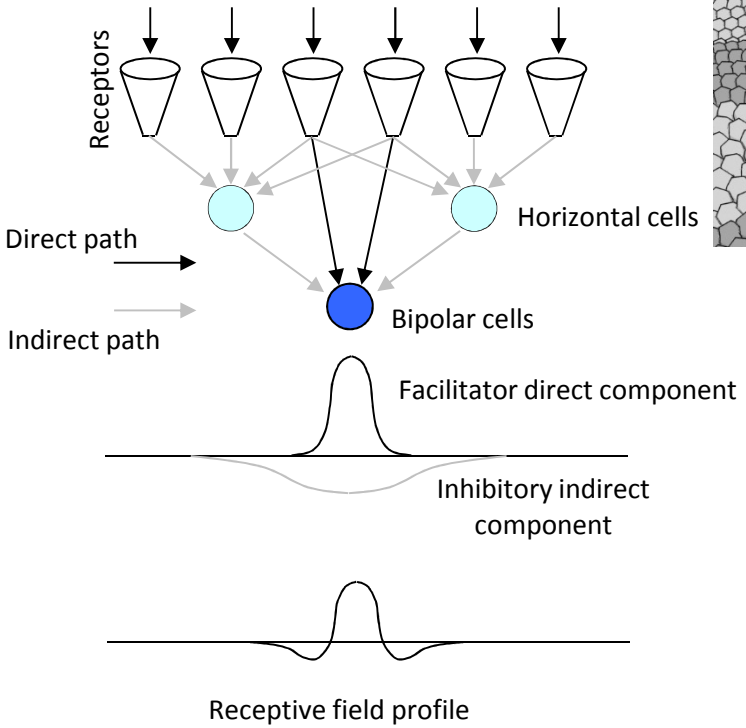
- Function support: infinite (theoretically)
  - Finite practically and tunable (σ)
  - On-center, off surround
- Biological example
  - ganglion cells (retina) encode alternatively large or small (localized) receptive fields
  - cochlear stereocilia cells (inner ear) have locally tuned frequency responses

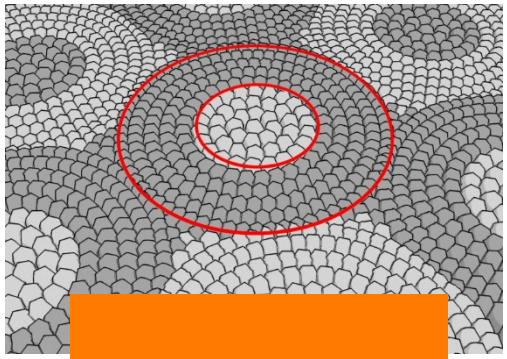
There's a biological corrispondence, in the brain there are cells that are spatially selective, example in the retina (sensitivity to spatial dimension) or ear (sensitivity to frequencies). Indeed the support can be a space or a frequency domain etc.

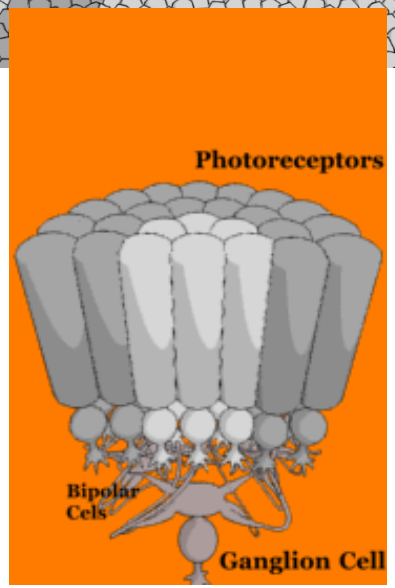
examples of biological. there's biological coherence

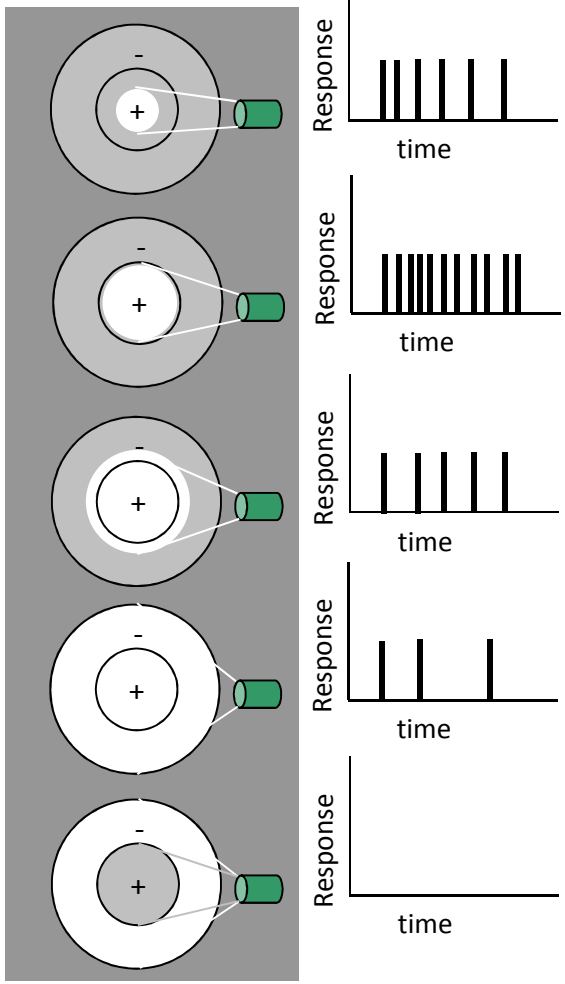
Ganglion cells







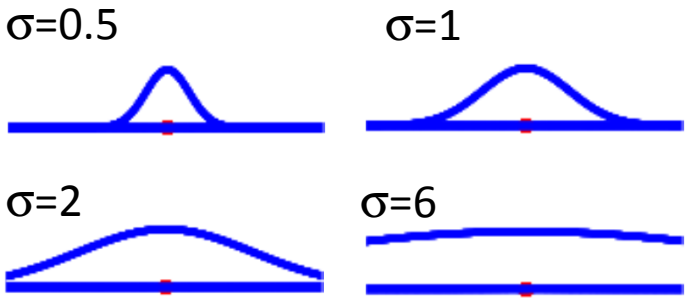




On-Center, Off-Surround

I remove the factor that enable us to integrate to 1, I'm changing the integration, but this allows me to have a constant amplitude to one.
If I have a filter I can apply it by convolving it. This would be equivalent in the freq. domain to a product. When Im dealing with function reconstruction with gaussian, since the reconstraction can be seen as a gaussian that moves in my domain and I multiply the function I can sort of seeing it as a convolution.

Gaussian Filter



$$\phi(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Amplitude no longer depends on σ
Max A = 1

Fourier transform

$$F(\nu) = e^{-\pi\sigma^2\nu^2}$$

The transform of a gaussian is a gaussian!

Low-pass Gaussian filter
Cut-off frequency

$$\nu_{cut-off} \rightarrow \frac{1}{\sigma}$$

Filtering in the spatial domain: convolution with the Gaussian kernel

If I use infinite number of gaussians I'll be in this condition

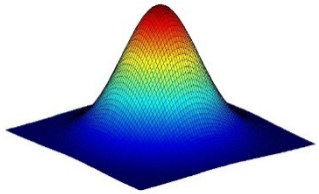
$$y = f(x) = w * \phi(x - \mu; \sigma) \quad \Longleftrightarrow \quad Y(\nu) = W(\nu)F(\nu)$$

w is not determining the amplitude

*this should be true only if we are using gaussians distributed in the input space at even incremenets, with identical STD

I'm assuming sampling my function with a specific sampling rate.

Gaussian filter

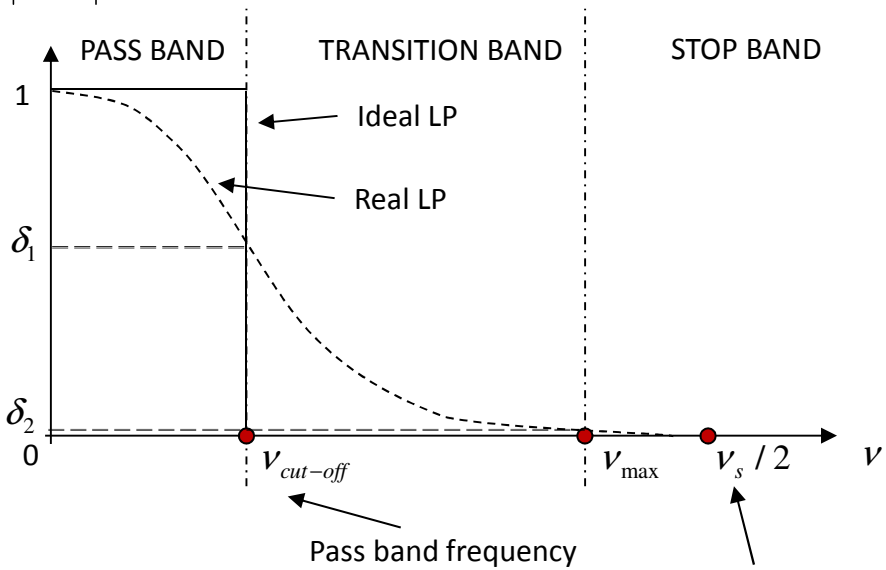


$$\phi(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$F(\nu) = e^{-\pi\sigma^2\nu^2}$$

Normalized spectrum amplitude of the filter

$|F(\nu)|$



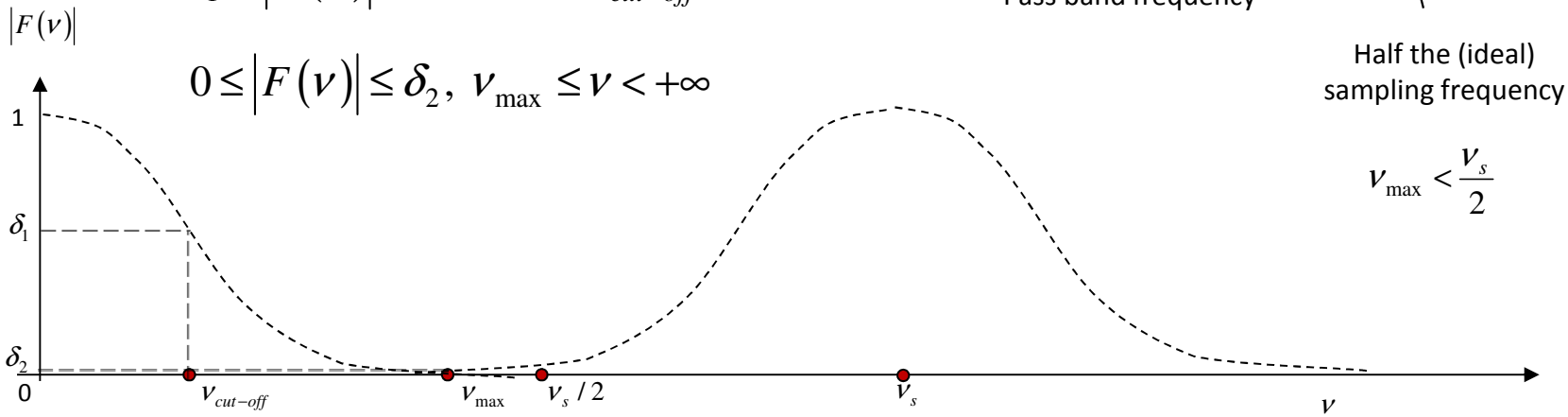
δ_1 max attenuation in the Pass band

δ_2 min decay in the Stop Band (finite support approximation)

$$\delta_1 \le |F(\nu)| \le 1, \ 0 \le \nu \le \nu_{cut-off}$$

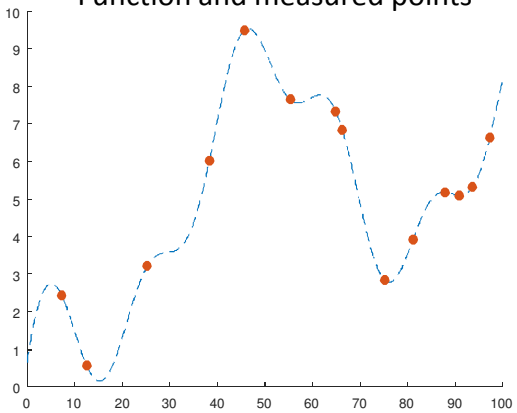
$$0 \le |F(\nu)| \le \delta_2, \ \nu_{max} \le \nu < +\infty$$

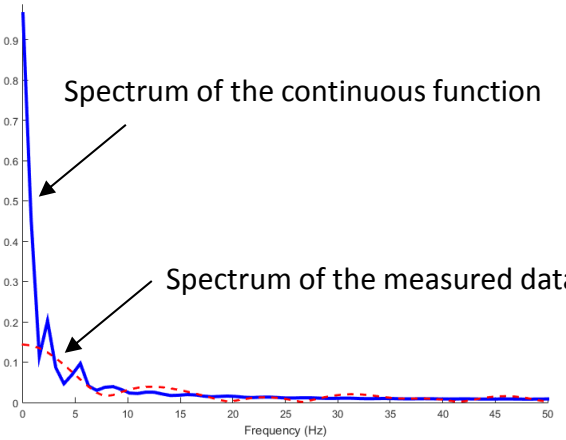
$$\nu_{max} < \frac{\nu_s}{2}$$



Frequency analysis

Function and measured points



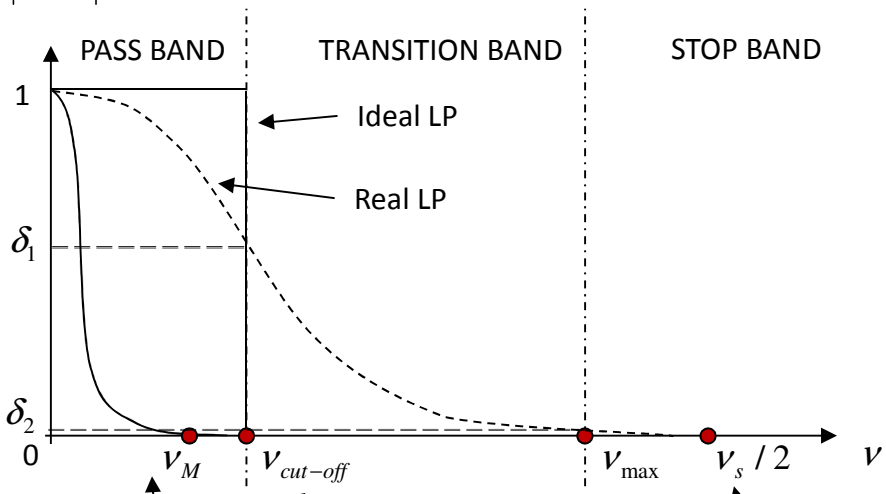


Spectrum of the continuous function

Spectrum of the measured data

Normalized spectrum amplitude of the filter

$|F(\nu)|$



Maximum frequency content of the function

Pass band frequency

Half the (ideal) sampling frequency

it should respect the nyquist theorem. This is more critical in a non uniform sampling rate

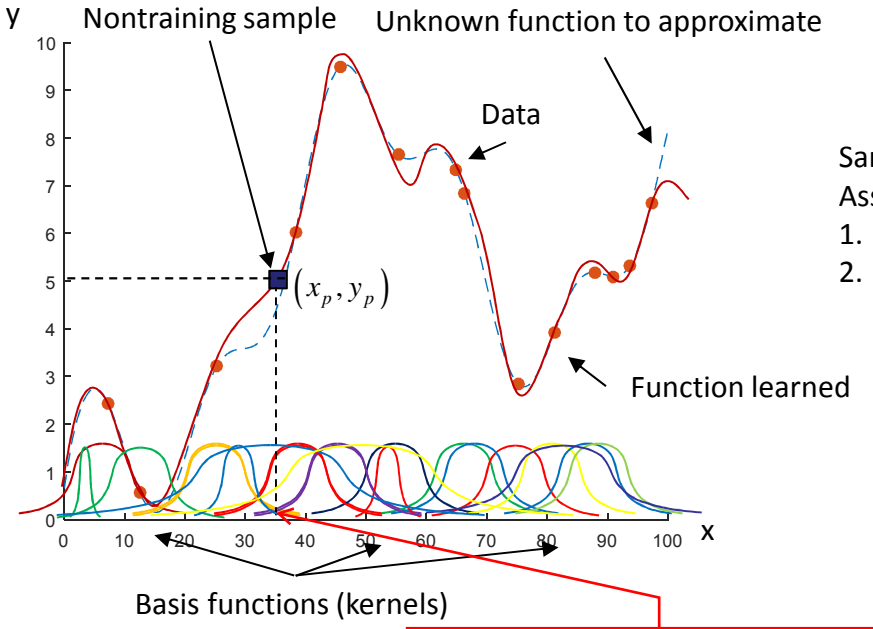
$$\nu_{\max} < \frac{\nu_s}{2}$$

In my ideal world I'ld to use a gaussian to reconstruct a function by convolution. (A finite number of gaussians). I should do a preprocessing of the data btw.

It can be done both as an approximaton and interpolator.

Function reconstruction (1-D)

$x \subset R \qquad y \subset R$ $\qquad$ K is the number of training data



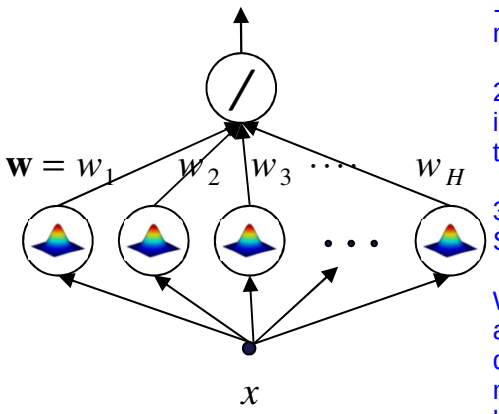
Sampled data can be regarded as training patterns

Assumptions

1. Gaussians are located at specific points in the input domain
2. Shape to be set

$$y_p = f(x_p) = w_2\phi_2(x_p) + w_3\phi_3(x_p) + w_4\phi_4(x_p)$$

$$y = f(x) = \sum_{i=1}^{H} w_i \phi_i(x)$$



H is the number of Gaussians

1. just one layer of neurons

2. gaussians are put in a specifc poin in the domain

3. I have can set the STD

We must have an automatic method to deploy gaussian in my domain. E.g.: in high freq function I'd use short gaussians! (large for low freq).

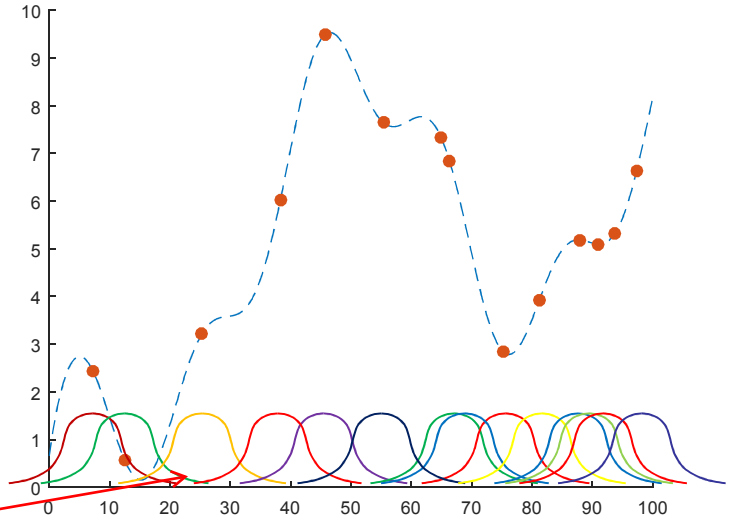
How to set the gaussians in my domain.
1. I could put one gaussian for each point
2. or a fixed number of gaussians, evenly spaced

same formulation for gaussian in both condition.

Domain sampling

Gaussians at the data points

Fixed variance



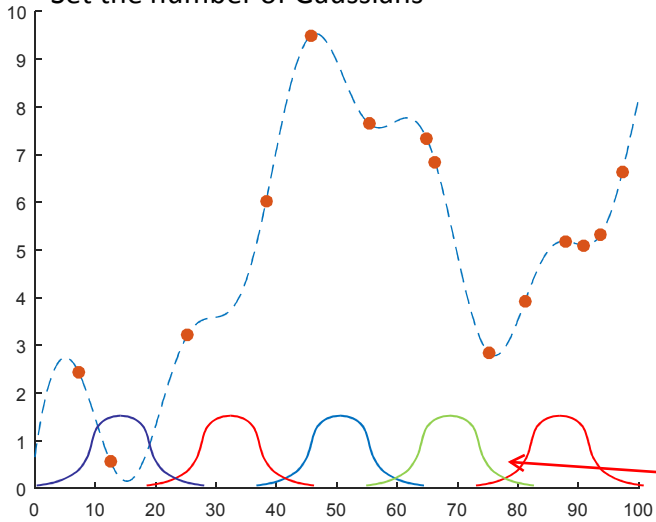
as many gaussians as the input data.

$$\mu_k = x_k$$

Evenly-spaced Gaussians

Fixed variance

Set the number of Gaussians



This is not specific for a dataset size so it could be used for different functions, so it's more flexible

$$\mu_k = \mu_{k-1} + \Delta$$

$$y = w_1 e^{-\frac{(x-\mu_1)^2}{\sigma^2}} + w_2 e^{-\frac{(x-\mu_2)^2}{\sigma^2}} + \ldots.$$

The STD changes the shape on the frequency domain, you can reconstruct content up to certain values.

Learning the weights

Risk of over-fitting or under-fitting in some regions of the domain

e.g: one dimension in input and one dimension in output

Training the RBF network (supervision)

Training set $T = \left\{ x^{(k)}, y^{(k)} \right\}_{k=1}^{K}$

Goal $y^{(k)} = f\left(x^{(k)} \right)$ for all *k*

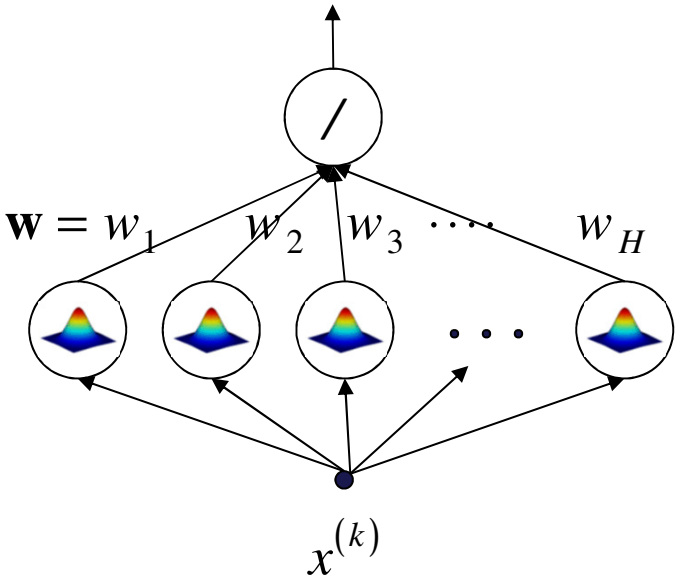
$$\min \; SSE = \sum_{k=1}^{K} \left[y^{(k)} - f\left(x^{(k)} \right) \right]^2$$

$$= \sum_{k=1}^{K} \left[y^{(k)} - \sum_{i=1}^{H} w_i \phi_i \left(x^{(k)} \right) \right]^2$$

The shape is known! The only variable unknown is the wi

$$\overline{y}^{(k)} = f\left(x^{(k)} \right) = \sum_{i=1}^{H} w_i \phi_i \left(x^{(k)} \right)$$

Predicted by the network



Linear combination of all the error Im committin by using the gaussian function to reconstruct the function. I'm just accumulating the output difference

This means for solve for weights, but that formulation could be done for the

Training the RBF network (supervision)

Learn the optimal weigth vector based on K training samples (not evenly spaced) assuming that the shape and the location of the *H* basis functions are known.

Simplest assumption

$H = K$ As much as Gaussian functions as training samples

$\{\mu_i\} = \{x^{(k)}\}$ Gaussian centers at the input training data

$\{\sigma_i\} = \sigma$ All the kernels share the same standard deviation

but we are going to see what happens….

Trying generalization

H It can be tailored according to data distribution (heuristic,..)

$\{\mu_i\}$ Gaussian centers: 1) uniformly distributed in the input domain; 2) set by cluster analysis

$\{\sigma_i\}$ Multiple values: multi resolution exploiting pre-processing (local frequency analysis)

The computation of the weights it's in closed form!! you don't have to iterate! Then we have NI in the weights so you can't have a linear realtion (?)

Aw = b
w = pinv(A)+b
psuedoinverse of the matrix

Learn the optimal weight vector

Training set $T = \left\{ x^{(k)}, y^{(k)} \right\}_{k=1}^{K}$

$$\overline{y}^{(k)} = f\left(x^{(k)}\right) = \sum_{i=1}^{H} w_i \phi_i\left(x^{(k)}\right)$$

Cost function to be minimized

$$E = \sum_{k=1}^{K} \left[y^{(k)} - \sum_{i=1}^{H} w_i \phi_i\left(x^{(k)}\right) \right]^2 \quad \Rightarrow \quad \frac{dE}{dw_j} = 0$$

$$0 = \frac{\partial E}{\partial w_j} = -2 \sum_{k=1}^{K} \left[y^{(k)} - f\left(x^{(k)}\right) \right] \frac{\partial f\left(x^{(k)}\right)}{\partial w_j}$$

NO

$$= -2 \sum_{k=1}^{K} \left[y^{(k)} - f\left(x^{(k)}\right) \right] \phi_j\left(x^{(k)}\right)$$

$$\sum_{k=1}^{K} \left[\phi_j\left(x^{(k)}\right) f\left(x^{(k)}\right) \right] = \sum_{k=1}^{K} \left[\phi_j\left(x^{(k)}\right) y^{(k)} \right]$$

$$\boldsymbol{\varphi}_j^T \mathbf{f}^* = \boldsymbol{\varphi}_j^T \mathbf{y}$$

$$\boldsymbol{\varphi}_j = \left[\phi_j\left(x^{(1)}\right) \quad \phi_j\left(x^{(2)}\right) \quad \cdot \quad \cdot \quad \phi_j\left(x^{(K)}\right) \right]^T$$

$$\mathbf{y} = \left[y^{(1)} \quad y^{(2)} \quad \cdot \quad \cdot \quad y^{(K)} \right]^T$$

$$\mathbf{f}^* = \left[f\left(x^{(1)}\right) \quad f\left(x^{(2)}\right) \quad \cdot \quad \cdot \quad f\left(x^{(K)}\right) \right]^T$$

Learn the optimal weight vector

Training set $T = \left\{ x^{(k)}, y^{(k)} \right\}_{k=1}^{K}$

$$\overline{y}^{(k)} = f\left(x^{(k)}\right) = \sum_{i=1}^{H} w_i \phi_i\left(x^{(k)}\right)$$

Cost function to be minimized

$$E = \sum_{k=1}^{K} \left[y^{(k)} - \sum_{i=1}^{H} w_i \phi_i\left(x^{(k)}\right) \right]^2 \quad \Rightarrow \quad \frac{dE}{dw_j} = 0$$

For all *j (span all the Gaussians)*

$$\boldsymbol{\varphi}_1^T \mathbf{f}^* = \boldsymbol{\varphi}_1^T \mathbf{y}$$
$$\boldsymbol{\varphi}_2^T \mathbf{f}^* = \boldsymbol{\varphi}_2^T \mathbf{y}$$
$$\dots$$
$$\boldsymbol{\varphi}_H^T \mathbf{f}^* = \boldsymbol{\varphi}_H^T \mathbf{y}$$

NO

$$\boldsymbol{\Phi} = \left[\boldsymbol{\varphi}_1 \quad \boldsymbol{\varphi}_2 \quad \cdot \quad \cdot \quad \boldsymbol{\varphi}_H \right] \Rightarrow \boldsymbol{\Phi}^T \mathbf{f}^* = \boldsymbol{\Phi}^T \mathbf{y}$$

$$\mathbf{f}^* = \begin{bmatrix} \sum_{i=1}^{H} w_i \phi_i\left(x^{(1)}\right) \\ \sum_{i=1}^{H} w_i \phi_i\left(x^{(2)}\right) \\ \cdot \\ \cdot \\ \sum_{i=1}^{H} w_i \phi_i\left(x^{(K)}\right) \end{bmatrix} = \begin{bmatrix} \phi_1\left(x^{(1)}\right) & \phi_2\left(x^{(1)}\right) & \cdot & \cdot & \phi_H\left(x^{(1)}\right) \\ \phi_1\left(x^{(2)}\right) & \phi_2\left(x^{(2)}\right) & \cdot & \cdot & \phi_H\left(x^{(2)}\right) \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ \phi_1\left(x^{(K)}\right) & \phi_2\left(x^{(K)}\right) & & & \phi_H\left(x^{(K)}\right) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \cdot \\ \cdot \\ w_H \end{bmatrix}$$

$$\boldsymbol{\Phi}^T \boldsymbol{\Phi} \mathbf{w} = \boldsymbol{\Phi}^T \mathbf{y}$$

I have a tool for computin the weights! :)

Learn the optimal weight vector

Training set $T = \left\{ x^{(k)}, y^{(k)} \right\}_{k=1}^{K}$

$$\overline{y}^{(k)} = f\left(x^{(k)}\right) = \sum_{i=1}^{H} w_i \phi_i\left(x^{(k)}\right)$$

Cost function to be minimized

$$E = \sum_{k=1}^{K} \left[y^{(k)} - \sum_{i=1}^{H} w_i \phi_i\left(x^{(k)}\right) \right]^2 \Rightarrow \frac{dE}{dw_j} = 0$$

Training error

$$\mathbf{w}^* = \left(\mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathbf{y} = \mathbf{A}^{-1} \mathbf{\Phi}^T \mathbf{y}$$

$$\mathbf{e} = \mathbf{y} - \mathbf{f}^* = \mathbf{y} - \mathbf{\Phi} \mathbf{w}^*$$

-..... RECONSTRUCTION FILTER ..... ????????????

In the end you have to set a proper number of gaussians and the best STD.. if you have too many gaussians you could be overfitting

The threshold is a sort of bias of the function, here I'm not learning the threshold but only the weights for the gaussian, this means that the weights are sufficient to learn also the bias of the continous content of the function. The average level of the function doesn't matter. As I increase the number of points (frequency) it can learn better. If I use a too small STD I may have antialiasing. I must be able to set the proper combination of STD and number of gaussian, I can reduce over and over the need for a regular sampling, and obtain an adeguate function approximation. Maybe I could get the low freq content of my funciton if I'm using like a low number of neurons with an high STD. I get like te average trend of my function. I can also take another layer that uses the difference between this "Medium" value and the real function and takes into the account the high frequency content.. And I can decrease the STD layer by layer.And I also double the number of gaussians. This is what is called a multiple resolution reconstruction.

I could generalize my network including in the estimations also the centers (instead of uniform or in datapoints). I could put them to minimize the error. This way the center becomes variables, also the STD could be used. But this complicates the estimation, I can no longer exploit linear solution, but I must go in NL, derivating. In practice this can become very computational throublesome and time consuming. The error function is becoming very complex and there's the problem of local minima.

General learning with RBF

Training set $T = \left\{ x^{(k)}, y^{(k)} \right\}_{k=1}^{K}$

no?

$$\overline{y}^{(k)} = f\left(x^{(k)}\right) = \sum_{i=1}^{H} w_i \phi_i\left(x^{(k)}\right)$$

- Weight w_i
- Center μ_i
- Width σ_i
- Number of ϕ_i

$$E = \sum_{k=1}^{K} \left[y^{(k)} - \sum_{i=1}^{H} w_i \phi_i\left(x^{(k)}\right) \right]^2 = \sum_{k=1}^{K} \left[y^{(k)} - \sum_{i=1}^{H} w_i e^{-\frac{\left(x^{(k)} - \mu_i\right)^2}{2\sigma_i^2}} \right]^2$$

$$\frac{dE}{dw_j} =$$

$$\frac{dE}{d\mu_j} = 0$$

$$\frac{dE}{d\sigma_j} = 0$$

It can be done iteratively in bundle but convergence is an issue

Two-stage training

1. Determining kernel functions
2. Learning weights

We could make additional assumption: computing the centers by analizing the distrubition of the domain in the datapoints. Clustering. The result is to assign to each classes just one single gaussian. I'm just dealing with the domain (and not codomain) so I don't really know the freq. of the function in these points. So I'm losing dynamic of the function, but reducing the number of gaussians.

Analysis of the domain of the input data distribution

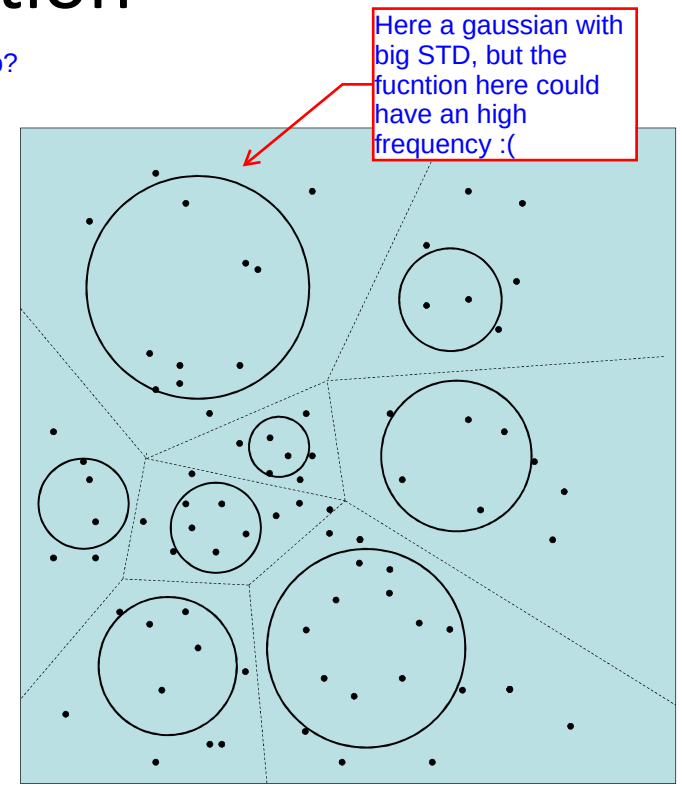
$$z = f(x, y) = \sum_{i=1}^{M} w_i \phi_i(x, y)$$

Automatic clustering (e.g. k-means)

$\{\mu_k\}$ Center of the cluster

$\{\sigma_k\}$ Amplitude of the cluster

no?



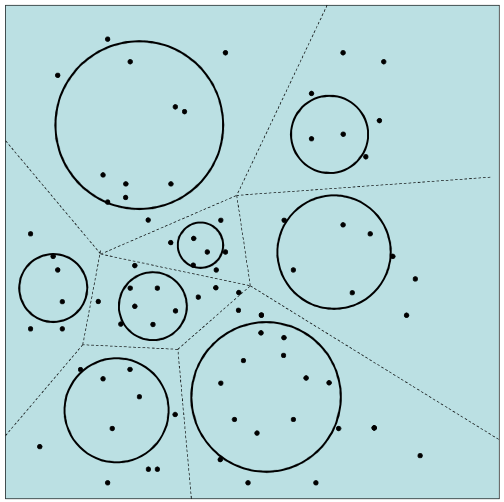
Main issues

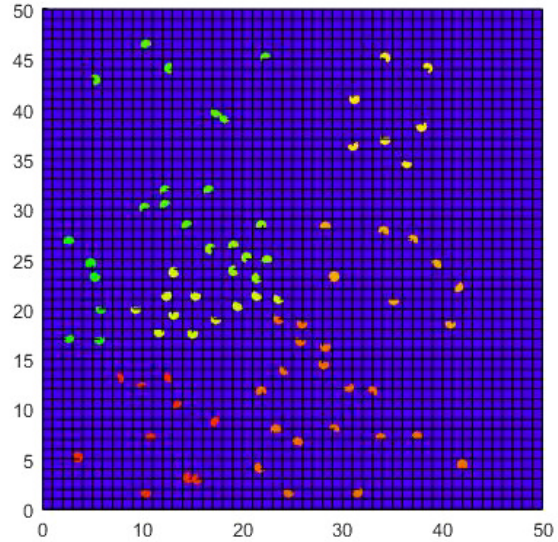
- Frequency selectivity is not generally guaranteed (it depends on local sampling)
- Into large cluster (higher σ/lower frequency) regions high frequencies are not properly captured
- Into small cluster (smaller σ/greater frequency) regions small frequencies are lost (noise effect maybe relevant)

no?

Analysis of both domain (input) and co-domain (output)

In his example we see that the choose of gaussians is not feasable because there's an high frequency. And maybe we should use more than one gaussian, or maybe completely discard that class





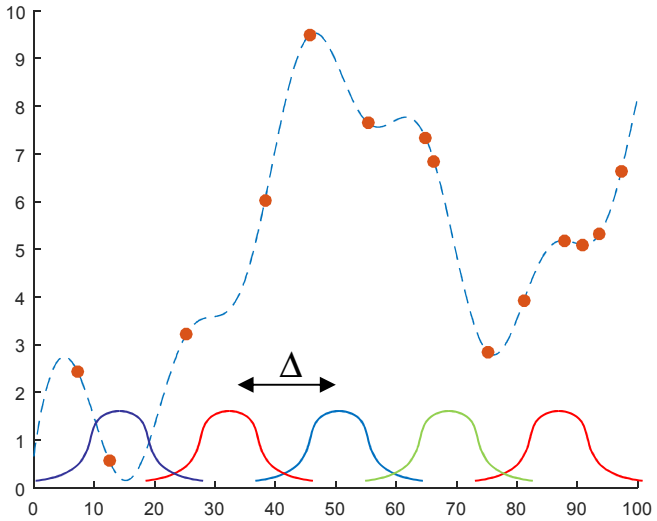
$\{\sigma_k\}$ Frequency analysis in each cluster

Main issues

- Requiring more complex computation
- The dynamics of the codomain can be in disagreement with kernels selected according to the domain data distribution
- Heuristics for the selection of the amplitude of the kernels

We have seen that It can be seen as a convolutional filter that allows me to reconstruct a function like a convulutional product. Here my assumption is that I'm moving the kernel by fixed step. Evenly spaced.

Uniform Gaussian sampling of the input domain



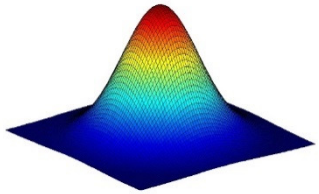
$$y = w_1 e^{-\frac{(x-\mu_1)^2}{\sigma^2}} + w_2 e^{-\frac{(x-\mu_2)^2}{\sigma^2}} + \ldots.$$

$$\mu_k = \mu_{k-1} + \Delta$$

- Evenly-spaced Gaussians with identical variance
- How to set the spacing Δ and the standard deviation σ?

I'm considering the gaussian filter, the spectrum of the filter, I'm considering the classical values to define stop and pass band. You'd like at max -3(?) dB on passband, thus the cutoff is d1 (sqrt2/2). But also a minimum of 40dB in the stop band, thus d2 =0.01 (I guess if it's normalized).
ES. matlab:
1. se ha una funzione lenta e mette sigma molto piccolo dopo un po' appare l'aliasing, mentre per sigma molto alte va abbastanza bene.
2. Con una funzione a freq. elevate con sigma elevata perdo il contenuto di freq alta e me la smussa. ricostruisco una LF version of the function
The noise will be a problem though. Plus the example were assuming uniform sampling. There's a problem if it's not uniform. The non uniform sampling is breaking everything. It can be difficult to obtain a nice reconstruction. There's a way of overcoming this issue by allowing layer reconstruction. Reconstruction as a sum of many layer of gaussians. The fist with a low freq and it gets bigger and bigger.

Cut-off



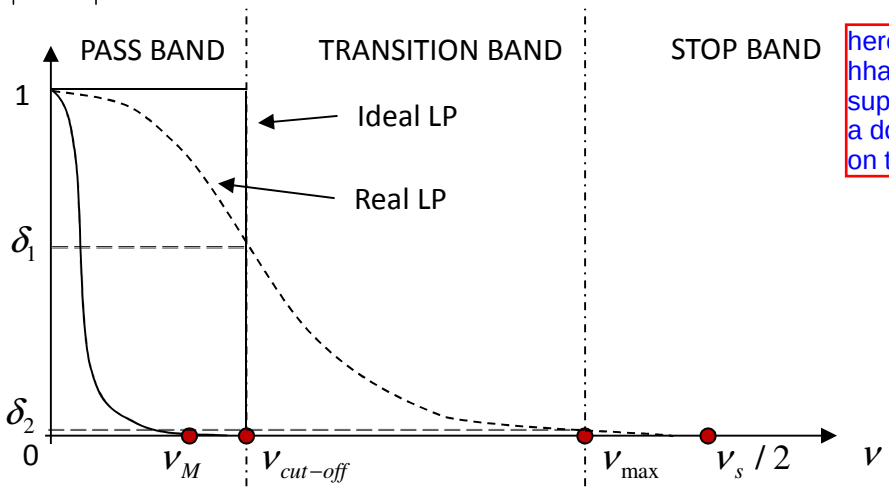
$$\phi(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$F(\nu) = e^{-\pi\sigma^2\nu^2}$$

Normalized spectrum amplitude of the filter

$|F(\nu)|$

NO?



here I have a replica hhat will be superimposing, doing a double contribution on the stopband

$\delta_1 = \frac{\sqrt{2}}{2}$ 3dB attenuation (max) in the Pass band

$\delta_2 = 0.01$ 40dB decay (min) in the Stop Band

I have to satisfy these

$$e^{-\pi^2\sigma^2\nu_{cut-off}^2} = \delta_1$$

$$e^{-\pi^2\sigma^2\nu_{max}^2} = \delta_2 / 2$$

$$\nu_{max} = \frac{0.7327}{\sigma}$$

$$\nu_{cut-off} = 0.2558\,\nu_{max}$$

I get my conditions knowning what happens with a gaussian

$$\begin{cases} \frac{\nu_M}{0.2558} < \nu_{max} < \frac{\nu_s}{2} \\ \sigma_{min} = \frac{1.465}{\nu_s} = 1.465\Delta \le \sigma \le \frac{0.1874}{\nu_M} = \sigma_{max} \\ \nu_s = \frac{1}{\Delta} \end{cases}$$

Δ Distance between two Gaussians in the frequency domain

Using this I'm probably reconstructing correctly with just one layer! The issue is that if I have nice sampling of the data it works well, but I'd like to be able to reconsturct function with a nonuniform sampling. [24.00] [XXX] also I don't know exaclty the freq content of the function

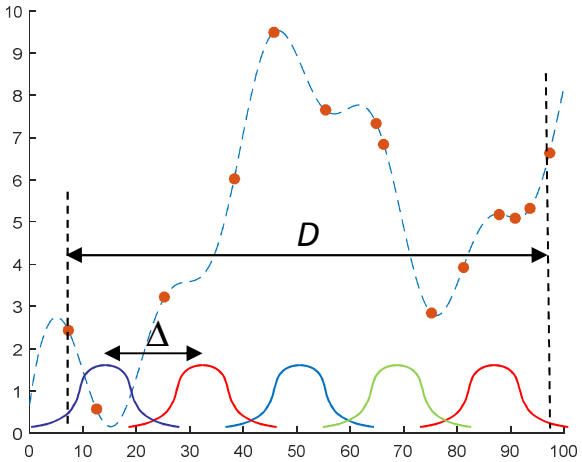
Uniform sampling

$$\sigma_{\min} = \frac{1.465}{\nu_s} = 1.465\Delta \le \sigma \le \frac{0.1874}{\nu_M} = \sigma_{\max}$$

- *D* size of the input dataset domain
- *N* number of input data

Question: do we know the exact value of ν_M ?

Answer: coarse estimation maybe so that setting ν_s (and therefore Δ) from data can be inconsistent

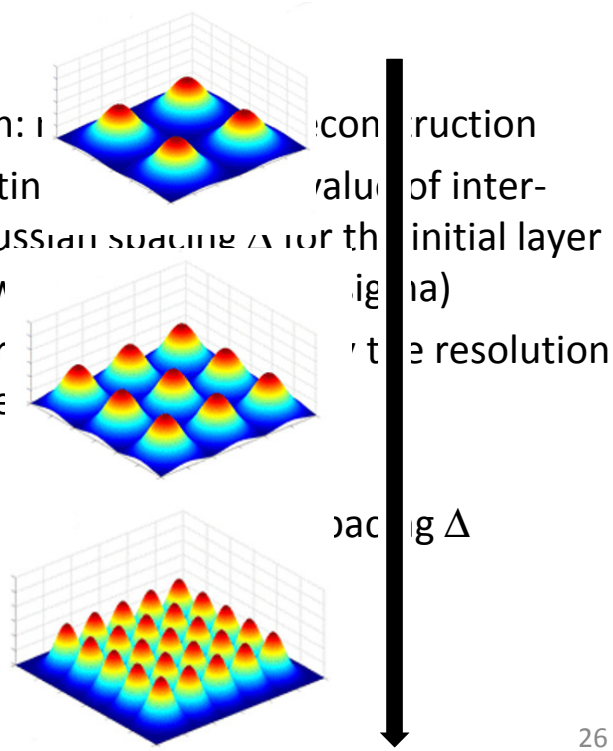


- the larger is σ, the smoother will be the reconstruction and the more the noise cleaned.
- the smaller is σ, the more the reconstruction will be close to the finest details of $f(x)$.

layer reconstruction
[[NEW SLIDE]]

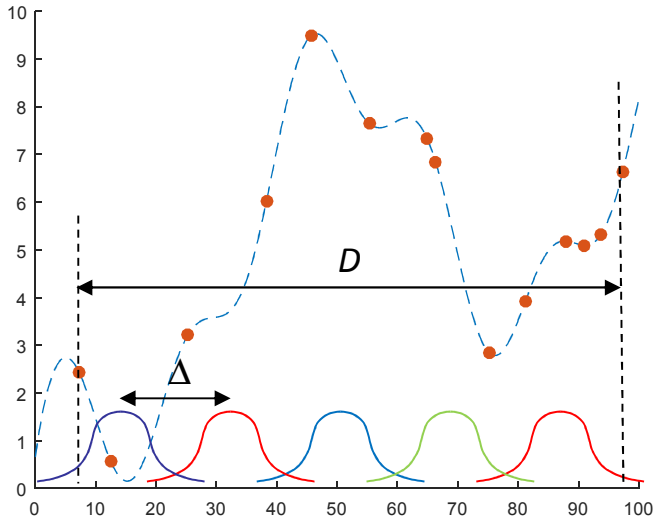
Solution: r con ruction

1. settin alu of inter-
   Gaussian spacing Δ for th initial layer
   (lov ig a)
2. incr t e resolution
   laye
   –
   – pac g Δ



Uniform sampling and multiple layers: heuristic initialization of the spacing Δ

$$\sigma_{\min} = \frac{1.465}{\nu_s} = 1.465\Delta \le \sigma \le \frac{0.1874}{\nu_M} = \sigma_{\max}$$



- *D* size of the input dataset domain
- *N* number of input data
- Δ (Gaussian spacing)?
  - e.g.: a fraction of *D*
  - starting with a high spacing means to reconstruct first low frequency version of the target function
- σ=1.465*Δ
- $M = D/\sigma$ number of Gaussian kernels
- *c* is the center of the input domain
- Gaussian centers $\mu_i = c + \left(i - \frac{(M-1)}{2} \right) \Delta$

Weight learning

Again closed form-solution $\mathbf{w}^* = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{y} = \mathbf{A}^{-1}\mathbf{\Phi}^T\mathbf{y}$

$$s_i = \frac{\sum_{r=1}^{N} y_r e^{-\left(\frac{(x_r-\mu_i)^2}{2\sigma^2}\right)}}{\sum_{r=1}^{N} e^{-\left(\frac{(x_r-\mu_i)^2}{2\sigma^2}\right)}}$$

per ogni punto del dataset

$$w_i = s_i\Delta$$

$$y = f(x) = \sum_{i=1}^{M} w_i e^{-\left(\frac{(x-\mu_i)^2}{2\sigma^2}\right)}$$

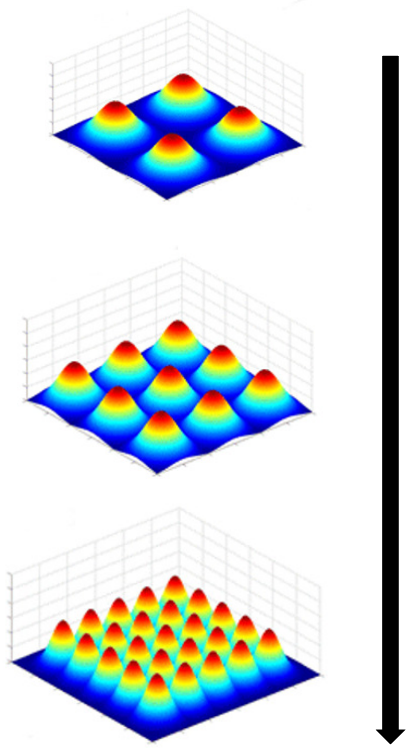
N: number of training data

M: number of Gaussians

Single layer RBF drawbacks

- How to select the best variance?
- It impacts on the number of Gaussians
  - Overfitting in the regions of low frequency content
  - Number of data can be insufficient to compute the weights

- Solution
  - Multi-layer approximation
  - Each layer has its own variance and inter-Gaussian spacing
  - Possibly eliminating the Gaussian functions that produce a prediction error under a predefined threshold



""hieratical"" reconstruction. Layer by layer I use [xxxx]. "F1" first functiion reconstructed, with veery low freq. But I still have my points of the function. I can compute E1, the error committin using this first layer. E1 = {yk} - F1. Then i consider the E1 the new function I'm dealing with, I'm computing the weights using this function! and I got F2. F2 is the reconstruction of the error function in the previous layer. E2 = E1 - F2. The final reconstructed function is F=F1+F2+F3 + ... + Fk. If I stop at K level I'm losing Ek. What's the point where I can stop the process? I could monitor the norm of the error (threshold).

Hierarchical approximation (multi-layer)

A. Start at the lowest frequency for layer k=1

$$\sigma_1 = 1.465\Delta_1 \qquad M_1 = D / \sigma_1$$

These are obtained from the simple case

B. Perform the learning of the weights of layer 1

$$s_{i,1} = \frac{\sum_{r=1}^{N} y_r e^{-\left(\frac{(x_r-\mu_{i,1})^2}{2\sigma_1^2}\right)}}{\sum_{r=1}^{N} e^{-\left(\frac{(x_r-\mu_{i,1})^2}{2\sigma_1^2}\right)}} \qquad w_{i,1} = s_{i,1}\Delta_1$$

C. Compute the predicted function values using the layer 1

$$f_1(x) = \sum_{i=1}^{M_1} w_{i,1} e^{-\left(\frac{(x-\mu_{i,1})^2}{2\sigma_1^2}\right)}$$

D. Compute the error signal for each x_r

$$err_{1,r} = y_r - f_1(x_r)$$

raddoppia il numero

tutte piu' vicine

tutte più piccole

E. Add a new layer k=k+1 of RB (layer 2)

$$\sigma_2 = \sigma_1 / 2 \quad M_2 = 2M_1 \quad \Delta_2 = \Delta_1 / 2$$

F. Perform the learning of the weights of layer 2

$$s_{i,2} = \frac{\sum_{r=1}^{N} err_{1,r} e^{-\left(\frac{(x_r-\mu_{i,2})^2}{2\sigma_2^2}\right)}}{\sum_{r=1}^{N} e^{-\left(\frac{(x_r-\mu_{i,2})^2}{2\sigma_2^2}\right)}} \qquad w_{i,2} = s_{i,2}\Delta_2$$

la funzione e' err ora... prima era y

Hierarchical approximation (multi-layer)

m2?

G. Compute the predicted function values using the layer 2

$$f_2(x) = \sum_{i=1}^{M_1} w_{i,2} e^{-\left(\frac{(x-\mu_{i,1})^2}{2\sigma_1^2}\right)}$$

H. Compute the error signal for each x_r

$$err_{2,r} = \{err_{1,r} - f_2(x_r)\}$$

I. Add a new layer k=k+1 of RB

$$\sigma_k = \sigma_{k-1} / 2 \quad M_k = 2M_{k-1} \quad \Delta_k = \Delta_{k-1} / 2$$

J. continue …..

I can set a stop criteria like with a threshold of the error.

How to stop?

- At layer *k* the error signal for the *r*-th training data is

$$\{err_{k,r}\} = \{y_r - f_k(x_r)\} \quad \text{with} \quad f_k(x) = \sum_{i=1}^{M_1} w_{i,k} e^{-\left(\frac{(x-\mu_{i,k})^2}{2\sigma_k^2}\right)}$$

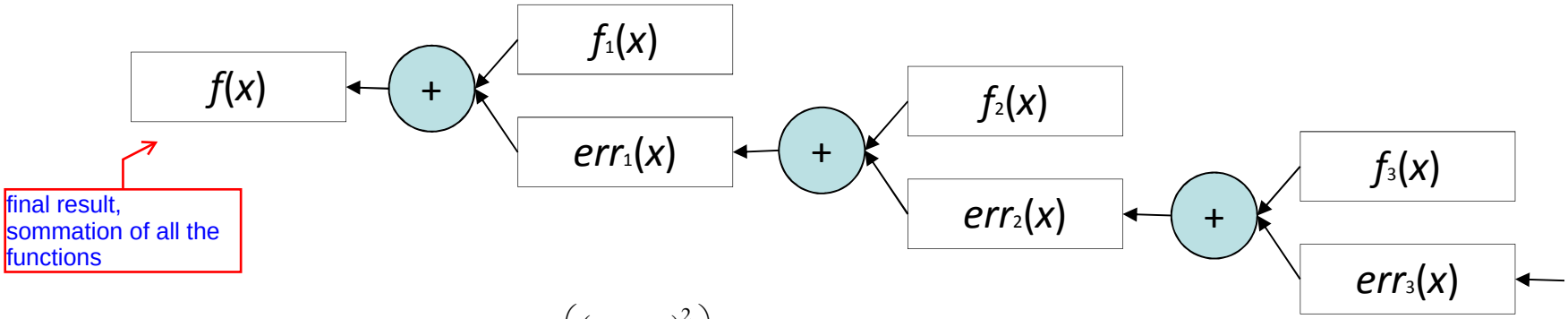
$$err_k = \frac{\sqrt{\sum_{r=1}^{N} err_{k,r}^2}}{N} < \overline{err}$$

*non dovrebbe essere err(k-1), r? Alternativamente se fa l'errore su yr allra dovrebbe considerare f() come somma di tutte le funzioncine.

A-posteriori stop criterion based on a predefined threshold

Hierarchical approximation (multi-layer)

Iterative processing of the residual reconstruction error



$$f_k(x) = \sum_{i=1}^{M} s_{i,k}\Delta_k e^{-\left(\frac{(x-\mu_{i,k})^2}{2\sigma_k^2}\right)}$$

$$f(x) = \sum_{k=1}^{L} f_k(x)$$

$$\sigma_k = \frac{\sigma_{k-1}}{2} \quad M_k = 2M_{k-1} \quad \Delta_k = \Delta_{k-1}/2$$

$$err_k = \{err_{k-1} - f_k(x)\}$$

I'm using an high number of gaussians, some part could have a very low freq, so if I'm putting a lot of gaussians at high freq they probably are useless in that region, the previous layer has probably already reconstructed the function, so there it could be overfitting. Before computing the weights I can evaluate what's the real contribution of the gaussians on the reconstruction! If the gaussian is providing me with a very little increment to the reconstruction (monitored by a specifc error threshold) you can remove the gaussian :). Thus I won't need to compute the weight for that gaussian. This can be done layer by layer.

How to remove undue Gaussians?

- At layer *k* the error signal for the *r*-th training data is

$$\{err_{k,r}\} = \{y_r - f_k(x_r)\} \quad \text{with} \quad f_k(x) = \sum_{i=1}^{M_k} w_{i,k} e^{-\left(\frac{(x-\mu_{i,k})^2}{\sigma_k^2}\right)}$$

per me è sbagliata (nell'ottica di valutare tutti così però può funzionare...) praticamente guarda quanto incide la gaussiana su tutti i punti in media

We can compute actually the error contribute of each *i*-th Gaussian function for all the training data

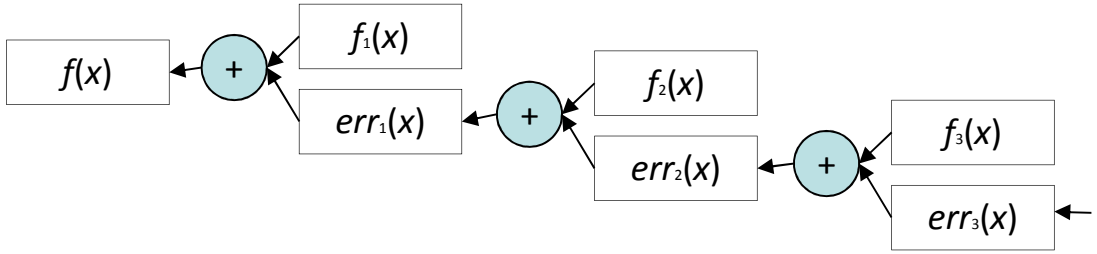
$$err_k^{(i)} = \frac{\sqrt{\sum_{r=1}^{N} \left(y_r - w_{i,k} e^{-\left(\frac{(x_r-\mu_{i,k})^2}{\sigma_k^2}\right)} \right)^2}}{N}$$

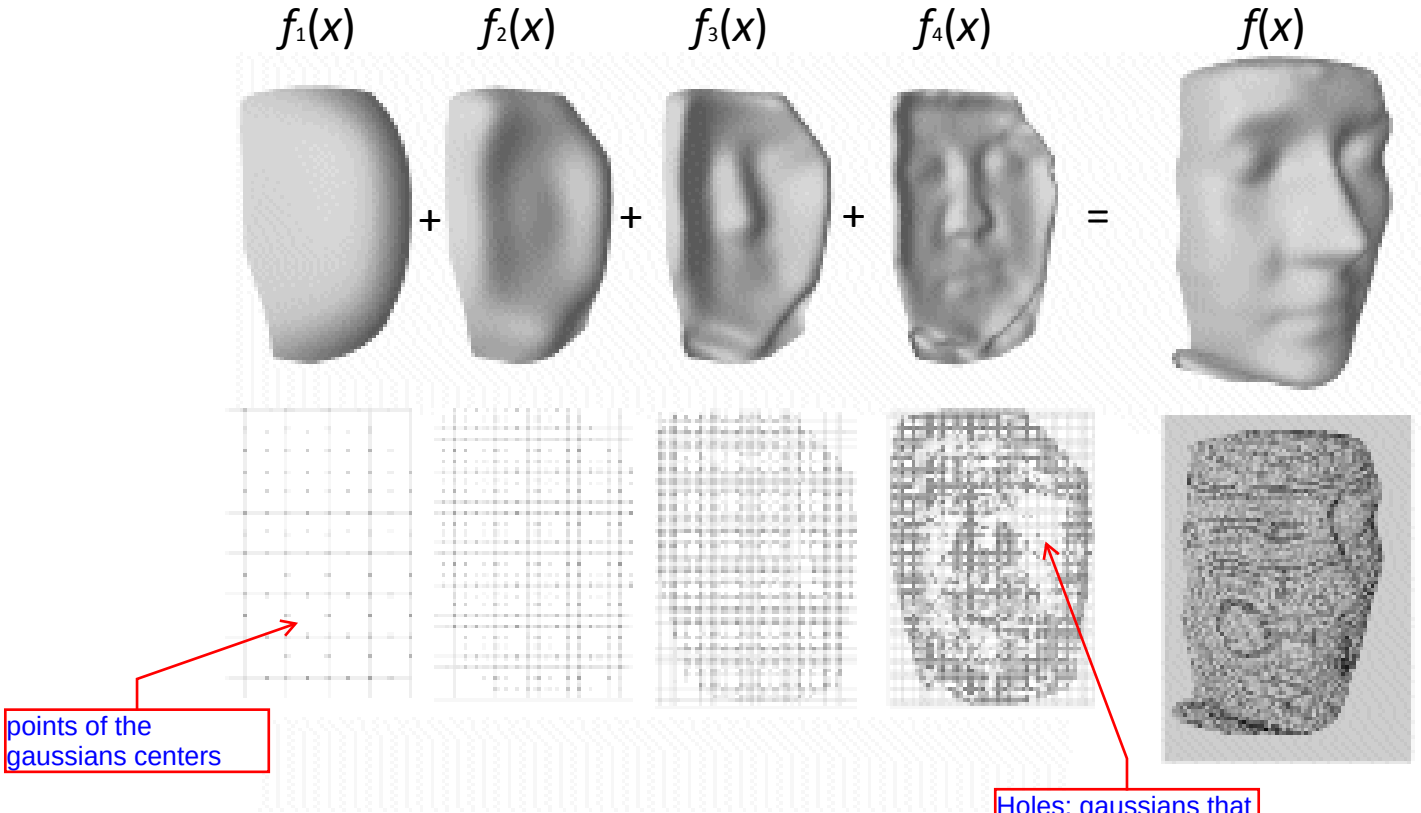
IF

$$err_k^{(i)} < \overline{err}$$

the contribute of the Gaussian i-th on the overall reconstruction is negligible

Hierarchical approximation (multi-layer)





points of the gaussians centers

Holes: gaussians that have been removed. They would have contributed little to the reconstruction.

This allows me to avoid a preprocessing analysis, I'm just performing a bruteforce approach. Also I can remove useless gaussians which are under a threshold.

Comparison with multilayer NN

Architecture

- Basic RBF networks have one *single* hidden layer
- FFNN networks may have *more* hidden layers

Neuron Model

- In RBF the neuron model of the hidden neurons is *different* from the one of the output nodes
- Typically in FFNN hidden and output neurons share a *common neuron model*
- The hidden layer of RBF is *non-linear*, the output layer of RBF is *linear*
- Hidden and output layers of FFNN are usually *non-linear*

Activation functions

- The argument of activation function of each hidden neuron in a RBF NN computes the *Euclidean distance* between input vector and the center of that unit
- The argument of the activation function of each hidden neuron in a FFNN computes the *inner product* of input vector and the synaptic weight vector of that neuron

Approximation

- RBF NN using Gaussian functions construct *local* approximations to non-linear I/O mapping.
- FF NN construct *global* approximations to non-linear I/O mapping

Resuming remarks (1)

- Gaussian location: *data driven* and *error driven*
  - Data driven: a predefined number of Gaussians is distributed in the input space according to the local density of the data points (clustering)
    - more points are sampled in the regions where the function is more difficult to be reconstructed, because it is more rapidly varying or, equivalently, its bandwidth is larger; which is not always the case
  - Error driven: to decouple the dependence of the density of the Gaussians from the density of the input data
    - insert incrementally one Gaussian after the other based on residual reconstruction error
    - computational intensive and optimality is not ensured
    - regular grid of Gaussian units but the Gaussian step (function of σ) determines the maximum frequency content
      - HRBF avoids undue Gaussians

Resuming remarks (2)

- Standard deviation of the Gaussians
  - a too large value of σ produces a low-pass filtered reconstruction; on the other side, σ cannot be decreased ad libitum as a too small value will lead to a spiky reconstruction
  - Local error driven (bounded) hierarchical solution
- Weights of the RBF network
  - Closed-form solution (linear equation system)

Final comments

- In the smooth regions of the function lot of Gaussians are automatically removed
- The reconstructed function is continuous
  - It can be arbitrarily resampled
- The network is parameterized
  - Basis functions: Gaussians
  - Number of layers
  - Variance set (one variance for each layer)
  - Position set
- Hierarchical RBF decomposition is in relation with the theory of **deep learning** in the human brain