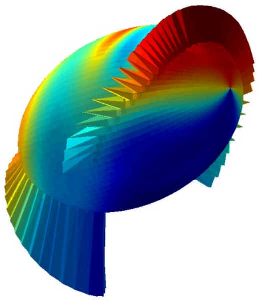
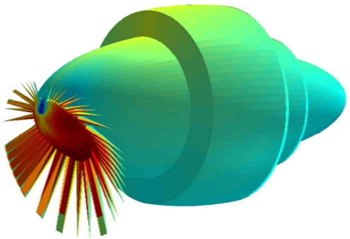
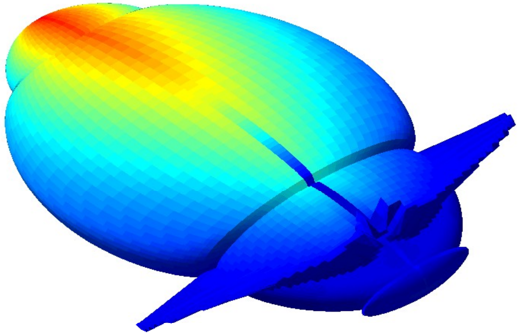
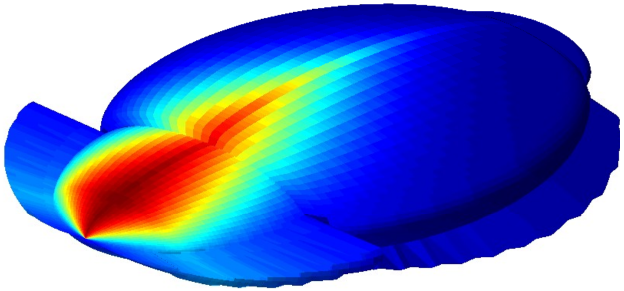




Lecture 9
Approximation





The Approximation Problem

Input values $\bar{x}^q \in X \subset \Re^{n+1} \qquad q = 1, \ldots, M$

$$\bar{x}^q = \begin{bmatrix} x_0^q & x_1^q & x_2^q & \ldots & x_n^q \end{bmatrix}^T$$

Output values $\bar{y}^q \in Y \subset \Re^{l+1} \qquad q = 1, \ldots, M$

$$\bar{y}^q = \begin{bmatrix} y_1^q & y_2^q & \ldots & y_l^q \end{bmatrix}^T$$

Training Set with ***real-valued*** inputs and outputs

Fit a model: $\hat{\bar{y}} = f\left(\bar{x}; \bar{w}\right)$

$\bar{w}$ are the parameters of the model

to minimize the loss function

$$J(\bar{w}) = \sum_k J^q(\bar{w}) = \sum_k e\left(\hat{\bar{y}}^q, \bar{y}^q; \bar{w}\right)$$

$e(\hat{\bar{y}}, \bar{y}; w)$ is an error function

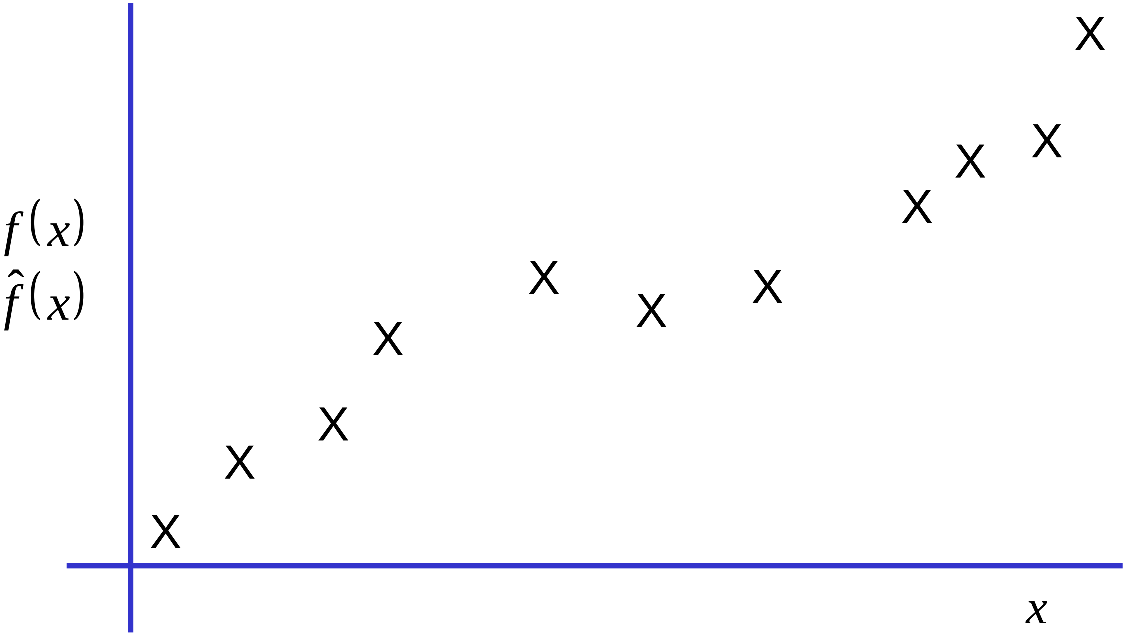
by adapting the parameters $\bar{w}$

Approximation = curve-fitting / regression

Approximation = curve fitting using a finite data set

interpolation → generalization
extrapolation → generalization

X = available (sample) data points



The Problem with Approximation

A finite number of points are given in real space

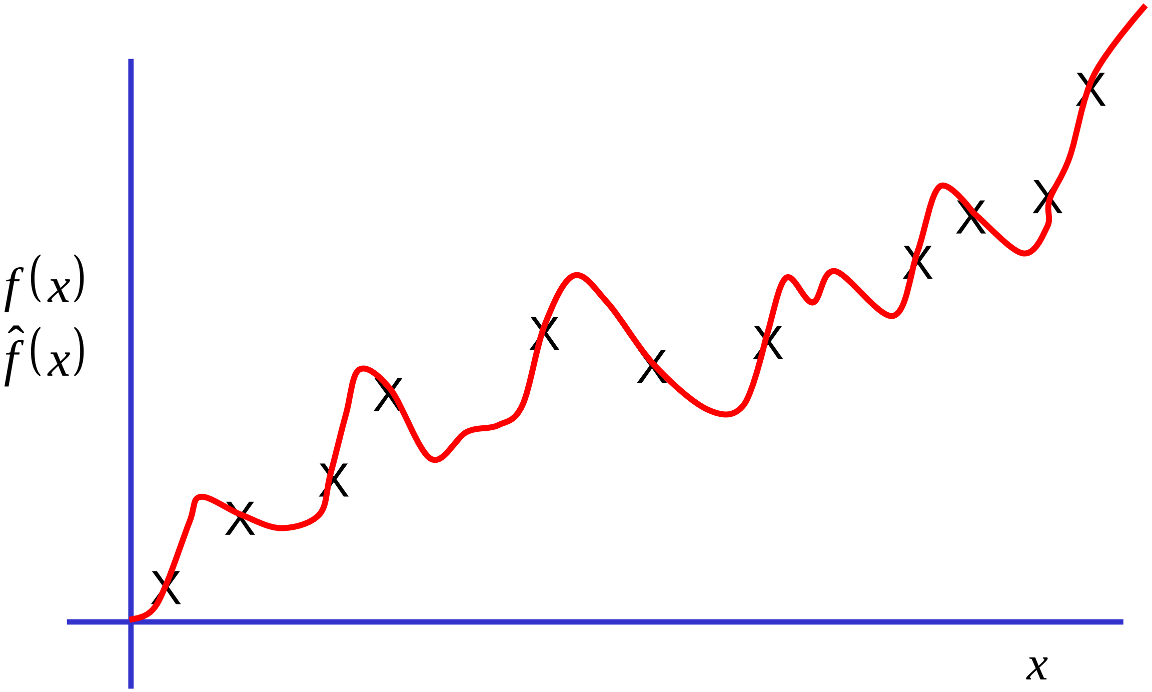
⇒ An infinite number of functions are compatible with it.

How should we choose the best?

Question to ask: Which function is most likely to be correct for unknown data?

Approximation = curve fitting using a finite data set

interpolation → generalization
extrapolation → generalization



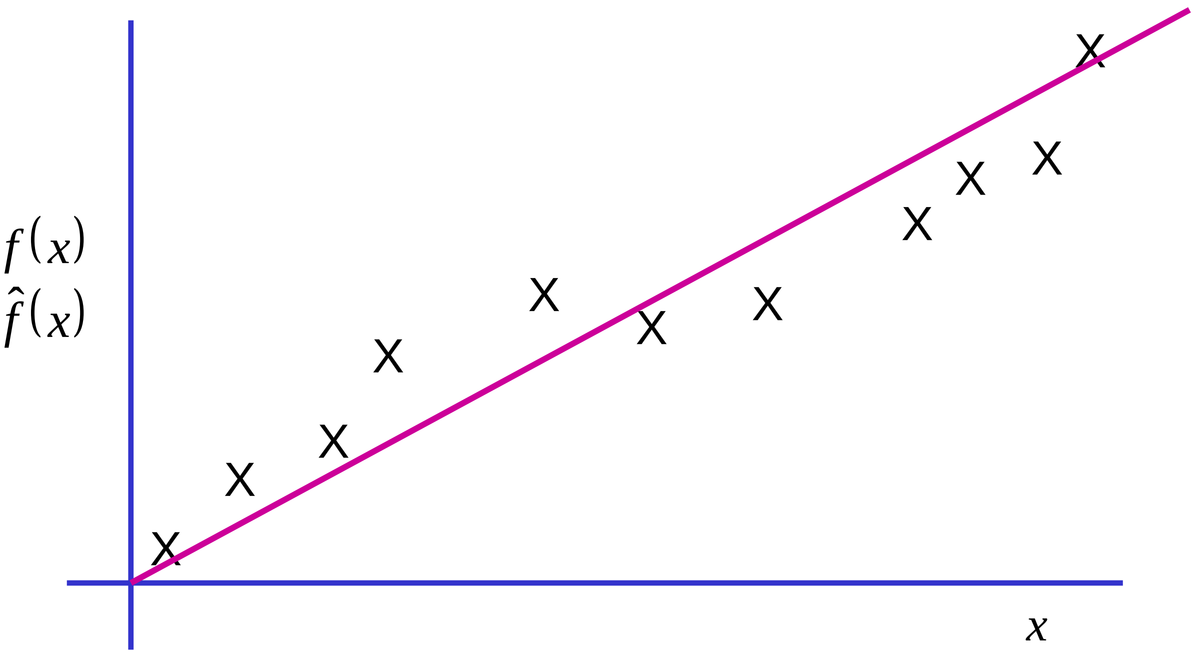
X = available (sample) data points

Is this a good approximation?

Why not?

Approximation = curve fitting using a finite data set

interpolation → generalization
extrapolation → generalization



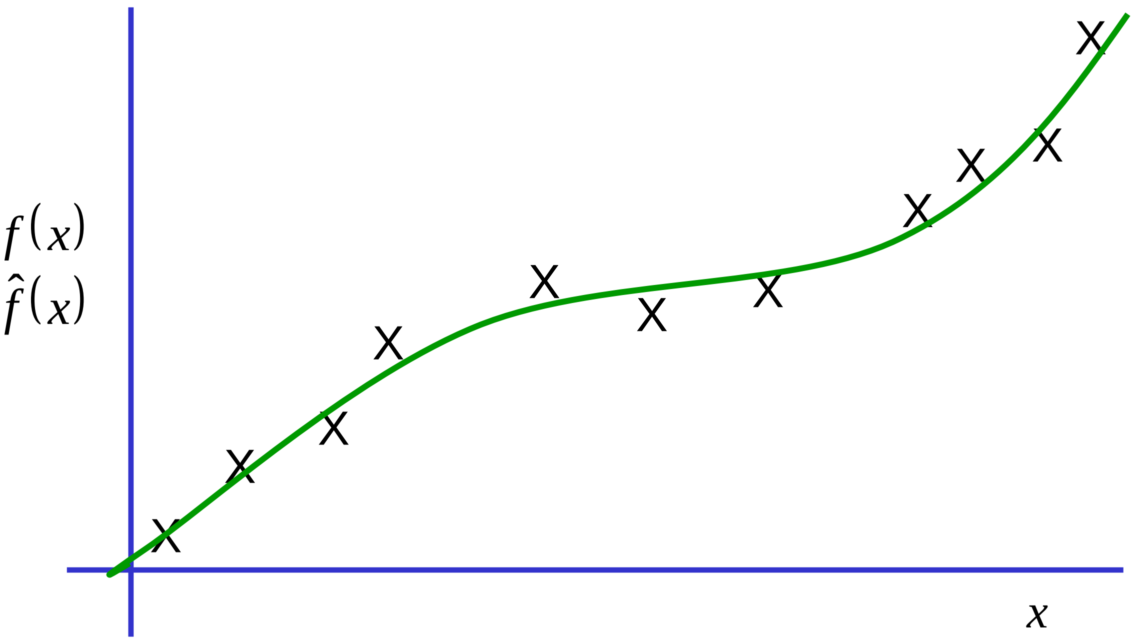
x = available (sample) data points

What about this one?

Better?

Approximation = curve fitting using a finite data set

interpolation → generalization
extrapolation → generalization



x = available (sample) data points

And this one?

Probably the best!

The Problem with Approximation

A finite number of points are given in real space

⇒ An infinite number of functions are compatible with it.

How should we choose the best?

Question to ask: Which function is most likely to be correct for unknown data?

Answer:

"One should not increase, beyond what is necessary, the number of entities required to explain anything" William of Ockham → Occam's Razor

"Everything should be made as simple as possible, but no simpler "

Attributed to Albert Einstein

The simplest adequate model is likelier to be correct.

Why are feedforward networks approximators?

Consider a 1 hidden-layer net with *n* inputs and one output.
The network can be written as:

$$\hat{y}_i = f_i\left[\sum_{j=0}^{m} w_{ij} f_j\left(\sum_{k=0}^{n} w_{jk} x_k\right)\right]$$ where *i* is the only output neuron

Assume $f_i(u) = u$ (linear output) $\Rightarrow \hat{y}_i = \sum_j w_{ij} f_j\left(\sum_k w_{jk} x_k\right)$

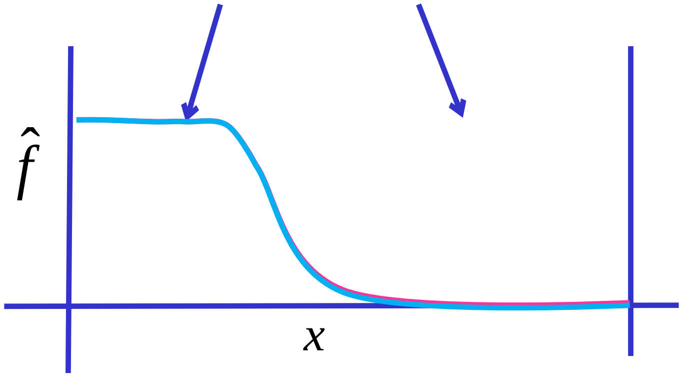
Thus $f_j(u)$ serve as ***basis functions***

i.e. a weighted sum of several nonlinear functions, f_j, defined over the input space

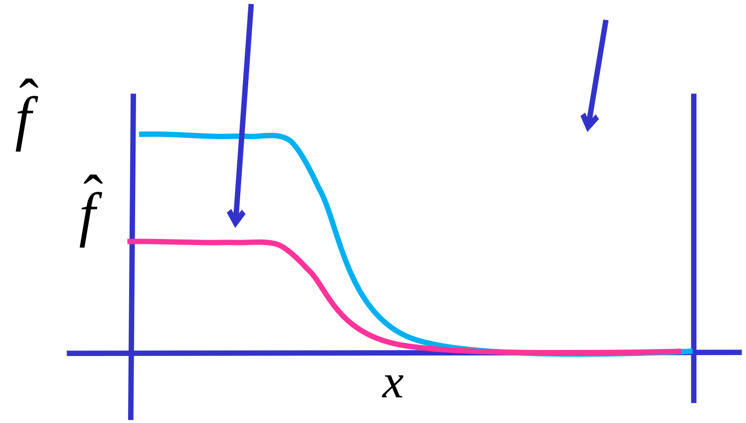
Recall the sinusoidal basis $f(x) = \sum_i a_i \cos(w_i x + \theta_i)$ = Fourier series approximation

Let $n = 1$, $m = 2$

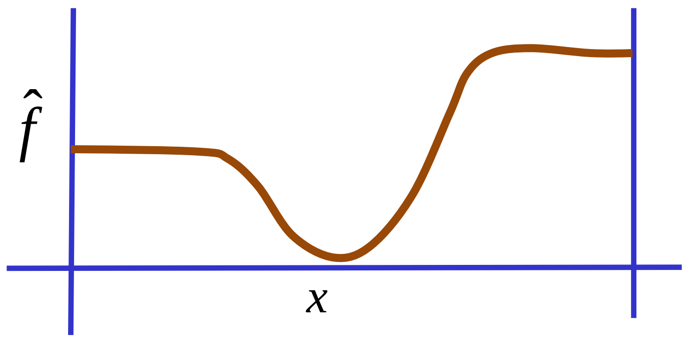
$f_1\left(w_{11}^h x + w_{10}^h\right)$ $\quad f_2\left(w_{21}^h x + w_{20}^h\right)$



$w_{11}^o f_1\left(w_{11}^h x + w_{10}^h\right)$ $\quad w_{12}^o f_2\left(w_{21}^h x + w_{20}^h\right)$



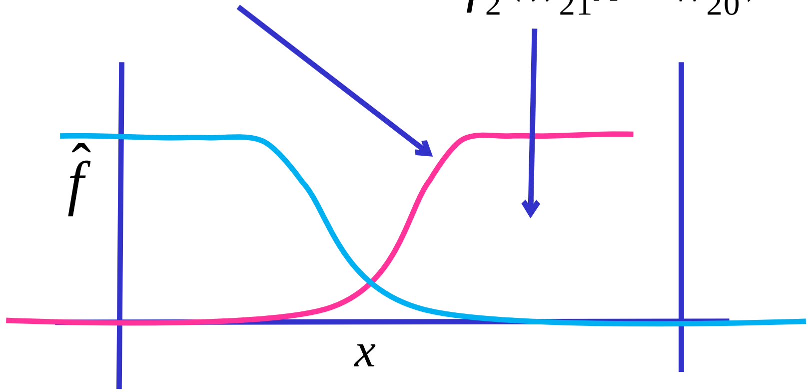
$$w_{11}^o f_1\left(w_{11}^h x + w_{10}^h\right) + w_{12}^o f_2\left(w_{21}^h x + w_{20}^h\right)$$



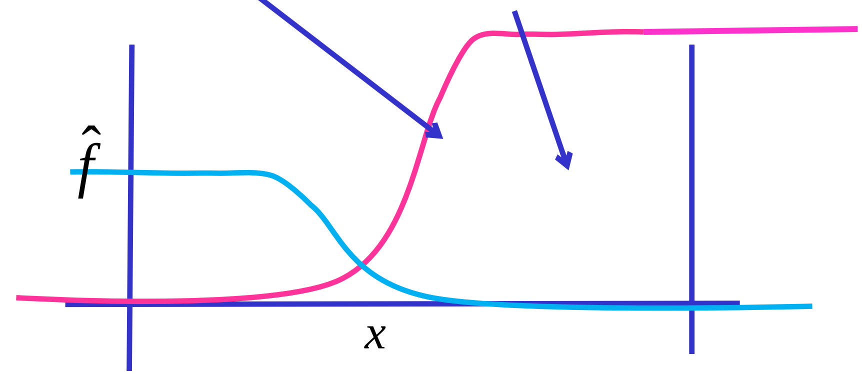
By adding several scaled and possibility reversed sigmoids, we can form pretty much any function. <u>This remains true even if $n > 1$</u>

Let $n = 1$, $m = 2$

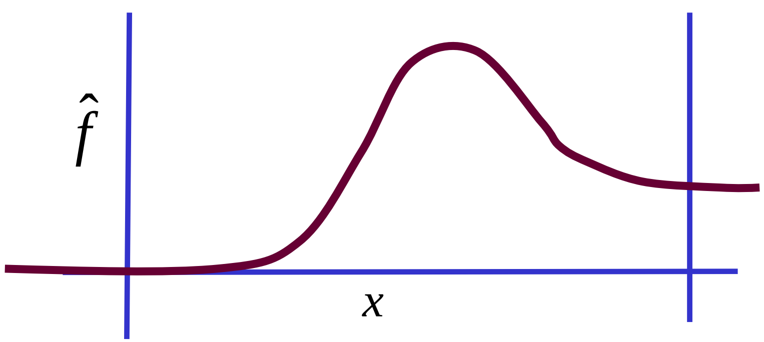
$$f_1\left(w_{11}^h x + w_{10}^h\right) \qquad f_2\left(w_{21}^h x + w_{20}^h\right)$$



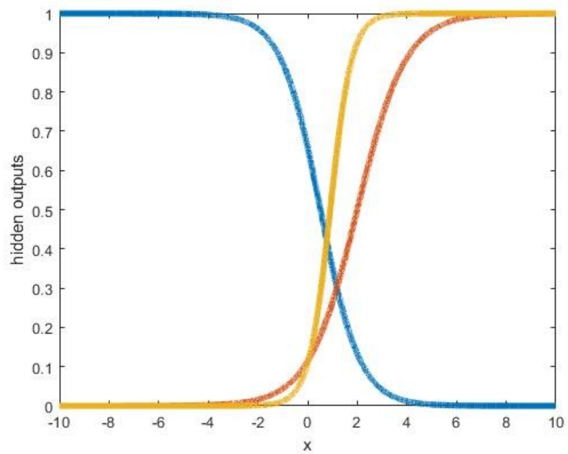
$$w_{11}^o f_1\left(w_{11}^h x + w_{10}^h\right) \qquad w_{12}^o f_2\left(w_{21}^h x + w_{20}^h\right)$$

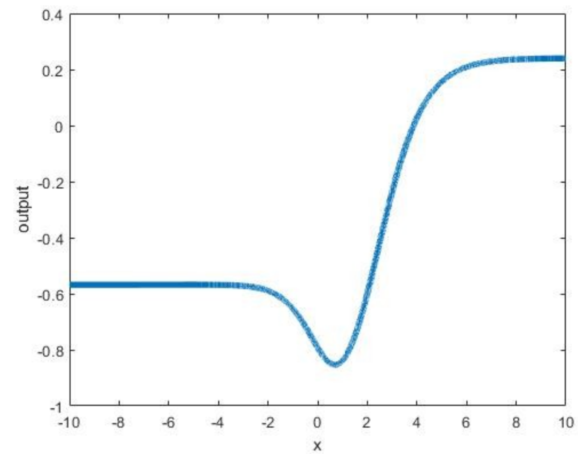


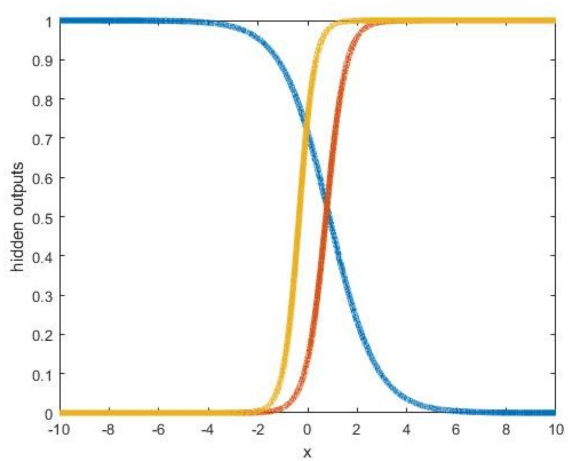
$$w_{11}^o f_1\left(w_{11}^h x + w_{10}^h\right) - w_{12}^o f_2\left(w_{21}^h x + w_{20}^h\right)$$

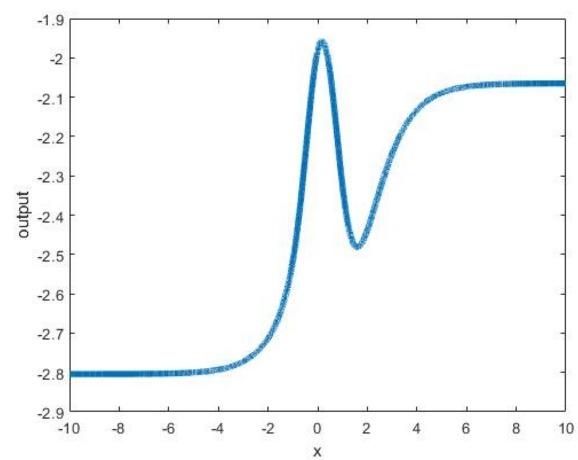


Two Real Examples with 3 Hidden Neurons

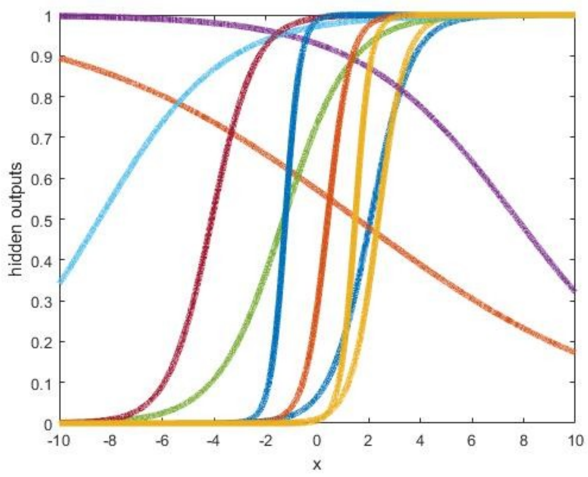


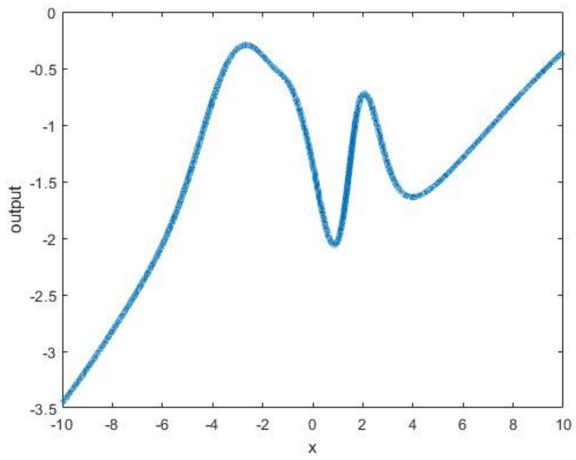




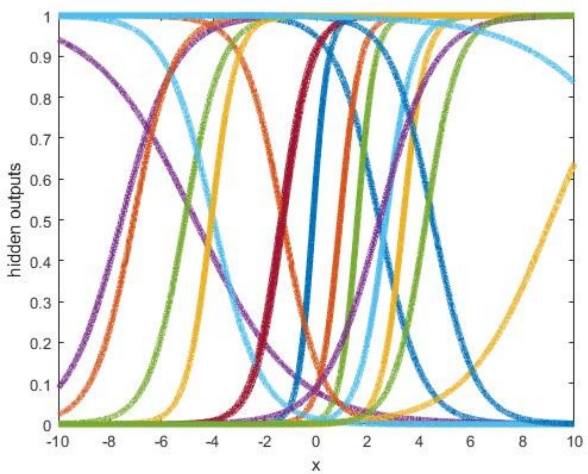


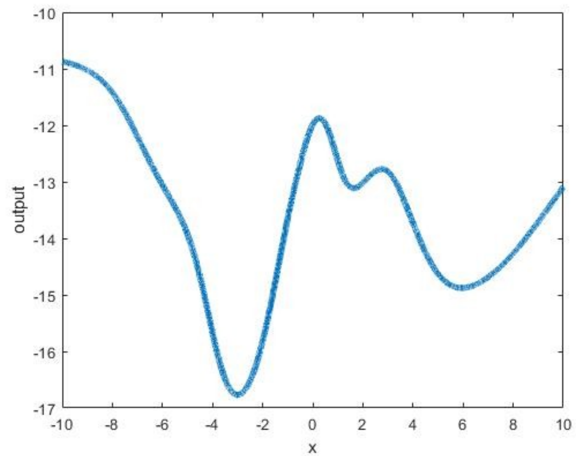
10 Hidden





20 Hidden





The Universal Approximation Theorem

(Cybenko; Funahashi; Hornik, Stinchcombe & White)

Let $f(.)$ be a non-constant, bounded and monotonically increasing continuous function.

Let $I_n \equiv [0,1]^n$

Let $C(I_n)$ be the space of continuous functions on I_n

Then, given any $\Phi \in C(I_n)$ and $\varepsilon > 0$

$\exists$ integer m and real constants a_j, b_j, w_{jk}

$$j = 1, \ldots, m$$

$$k = 1, \ldots, n$$

such that we can find:

$$\boxed{\hat{\Phi}(x_1, \ldots, x_n) = \sum_{j=1}^{m} a_j f\left(\sum_{k=1}^{n} w_{jk} x_k + b_j\right)}$$

and $$\boxed{\left|\hat{\Phi}(x_1, \ldots, x_n) - \Phi(x_1, \ldots, x_n)\right| < \varepsilon \;\; \forall \, x_1, \ldots, x_n \in I_n}$$

i.e. any cont. function on I_n can be approximated to arbitrary precision using <u>one</u> hidden layer of sigmoid neurons

Applications for Approximation

Modeling:

Given data points from an input-output system

$$\{(x^k, y^k)\}$$

determine $F : x^k \rightarrow y^k$

e.g.

F(heart rate, blood pressure) = autonomic nervous system function

F(externally monitored plant variables) = internal variables.

or, for a dynamical system

$$\dot{x} = F(x)$$

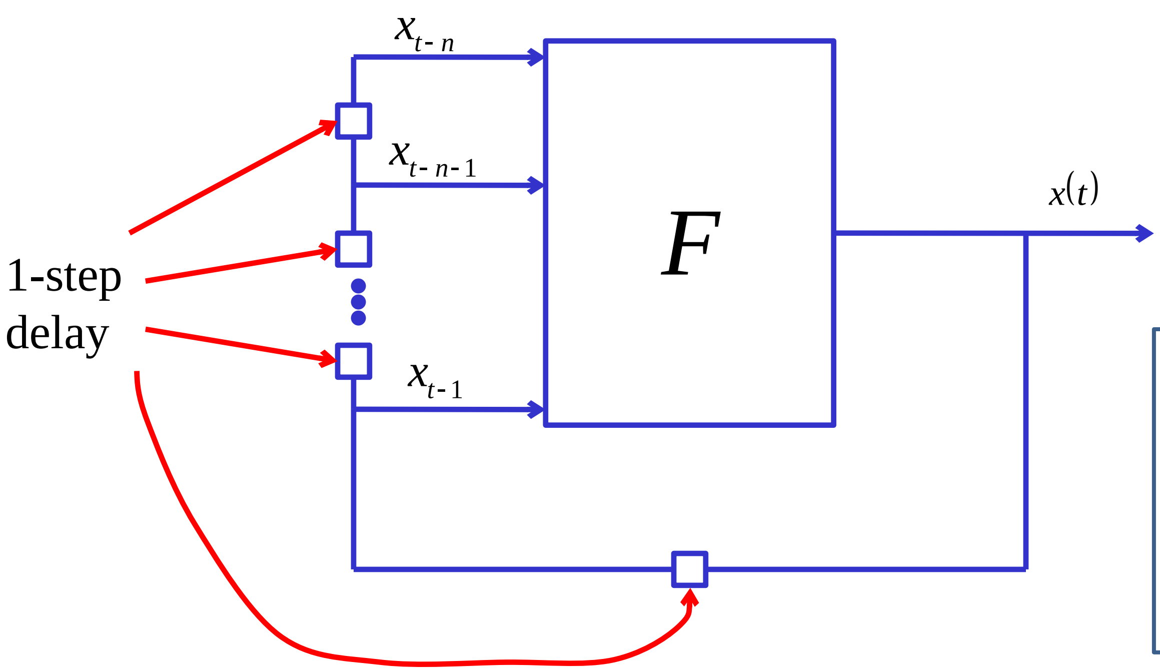
use data points $\{(x^k, \dot{x}^k)\}$ to find $\hat{F}(x) = \dot{x}$

Prediction:

Given a time series x_t

hypothesize $x_t = F(x_{t-1}, \ldots, x_{t-m}; \mathbf{w})$

Then use approximation to find F. This works also when the system has external inputs



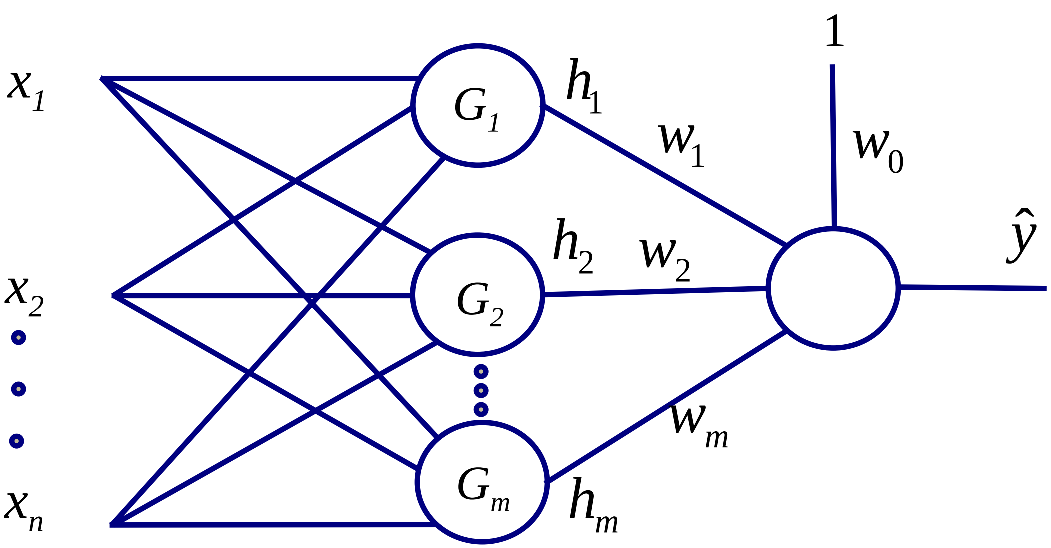
Compare with a linear autoregressive model:

$$x_t = \sum_{k=1}^{n} \alpha_i x_{t-k}$$

$$x_t = F(x_{t-1}, \ldots, x_{t-m}; \mathbf{w}) = \sum_{j=1}^{m} w_{ij}^{o} f\left(\sum_{k=1}^{n} w_{jk}^{h} x_{t-k} + w_{j0}^{h}\right)$$

Radial Basis Function Networks

Radial Basis Function (RBF) Networks



$$\|z\| = \sqrt{\sum_k z_k^2}$$

$$\boxed{h_j = G_j\left(\|\bar{x} - \bar{C}_j\|^2\right)}$$

$\bar{x} = \begin{bmatrix} x_1 & x_2 & . & . & . & x_n \end{bmatrix}^T$ n-dim

$\bar{w} = \begin{bmatrix} w_0 & w_1 & . & . & . & w_m \end{bmatrix}^T$ $(m+1)$-dim

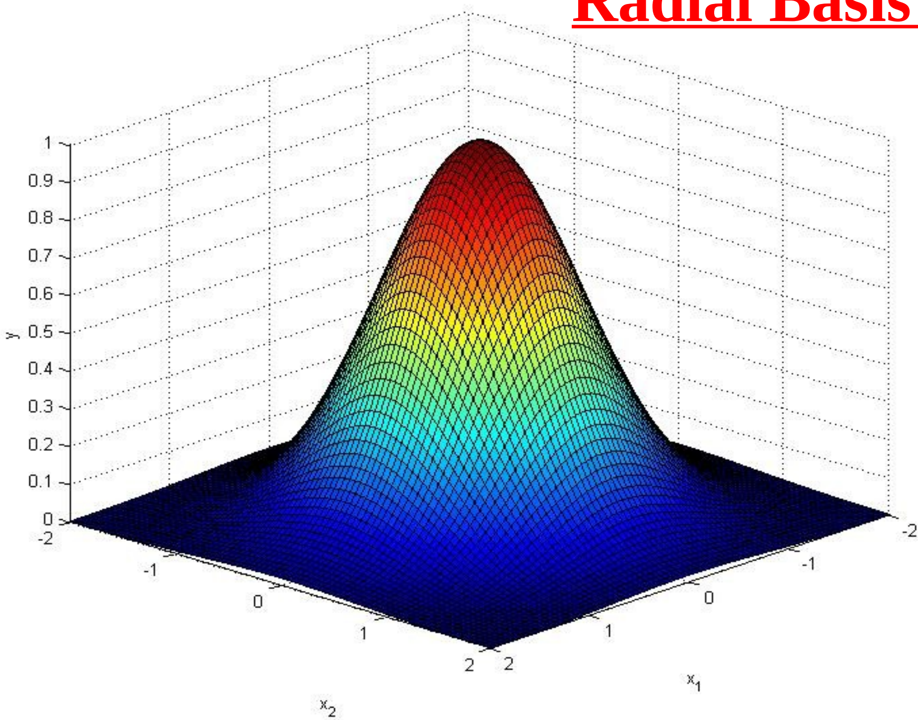
where $\boxed{G_j(u) = e^{\frac{-u}{2\sigma_j^2}}}$ Gaussian

$\bar{h} = \begin{bmatrix} h_0 & h_1 & . & . & . & h_m \end{bmatrix}^T$ $(m+1)$-dim

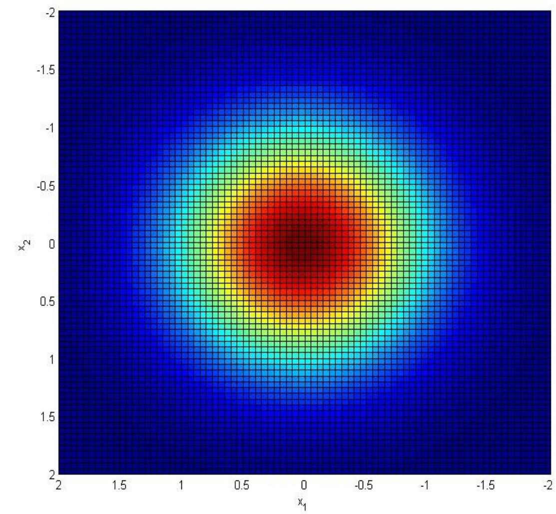
and $\boxed{\hat{y} = \bar{w}^T \bar{h}}$ linear

$\bar{C} = \begin{bmatrix} C_1 & C_2 & . & . & . & C_n \end{bmatrix}$ n-dim

Radial Basis Functions



since σ_j is a scalar, the contours on $x_1 - x_2$ are circular (equal variance)



A more general form would be to use $h_j = e^{-\left|(\bar{x}-\bar{C}_j)^T \Sigma^{-1} (\bar{x}-\bar{C}_j)\right|}$

where Σ is a symmetric $n \times n$ matrix with non-negative diagonal values

One can use $\Sigma^{-1} = K^T K$ where K is any real $n \times n$ matrix – weighting matrix.

Training RBF Networks

Option 1:

- Choose fixed $\overline{C}_j, \sigma_j, \forall_j$
- Train $\overline{w}$

Given training set

$$X = \{\overline{x}^1, \overline{x}^2, \ldots, \overline{x}^N\}$$

$$Y = \{y^1, y^2, \ldots, y^N\}$$

define

$$H = \{\overline{h}^1, \overline{h}^2, \ldots, \overline{h}^N\}$$

where $\overline{h}^q = \left[1 \quad h_1(\overline{x}^q) \quad . \quad . \quad . \quad h_m(\overline{x}^q)\right]^T$

Now training $\overline{w}$ is finding weights

for the linear estimate $y = \overline{w}^T \overline{h}$

using the training set $H \rightarrow Y$

$\boxed{\text{Use LMS}}$

(or use other linear estimation methods)

Note: σ_j s could be identical or different.

Option 2:

- Adapt centers $\overline{C}_j$ through self-organization
- Fix $\sigma_j \quad \forall_j$
- Obtain *H* using the $\overline{C}_j, \sigma_j$
- Train $\overline{w}$ using LMS etc.

we'll do this later

Notes: a) $\overline{C}_j$'s can be obtained by <u>k-means clustering</u> or some other clustering method

b) σ_j can be identical for each *j* or chosen heuristically

Option 3:

Treat $\sigma_j, \overline{C}_{jk}, w_j$ as parameters in an optimization problem and train them all by gradient descent or some other method.

Full Supervised Learning