Basis Functions

Volker Tresp

Summer 2018

Nonlinear Mappings and Nonlinear Classifiers

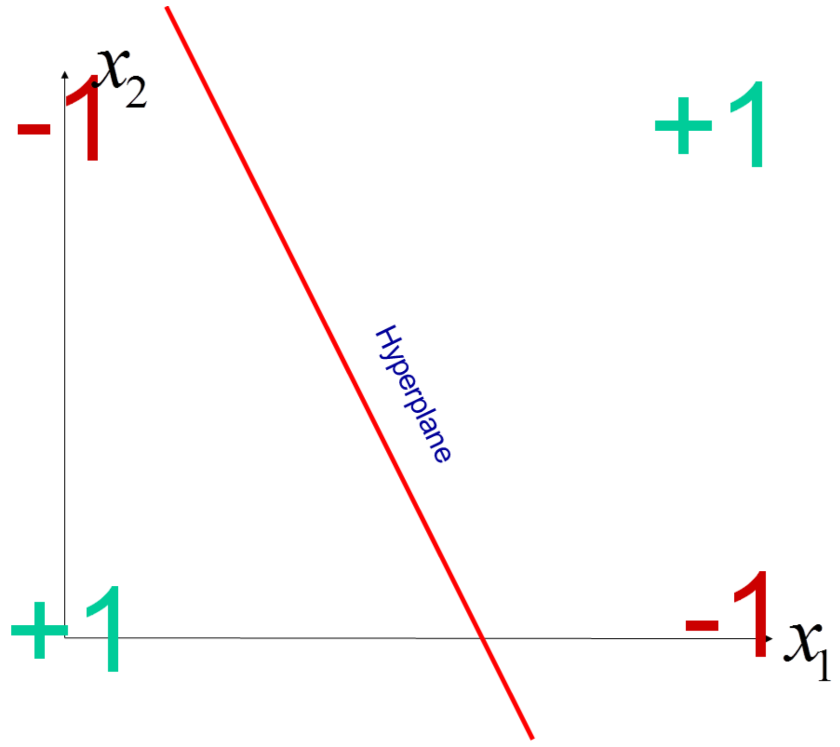
- Regression:
  - Linearity is often a good assumption when many inputs influence the output
  - Some natural laws are (approximately) linear $F = ma$
  - But in general, it is rather unlikely that a true function is linear
- Classification:
  - Linear classifiers also often work well when many inputs influence the output
  - But also for classifiers, it is often not reasonable to assume that the classification boundaries are linear hyperplanes

Trick

- We simply transform the input into a high-dimensional space where the regression/classification might again be linear!
- Other view: let's define appropriate features (feature engineering)
- Other view: let's define appropriate basis functions
- Challenge: XOR-type problem with patterns

$$\begin{array}{cccc} 0 & 0 & \rightarrow & +1 \\ 1 & 0 & \rightarrow & -1 \\ 0 & 1 & \rightarrow & -1 \\ 1 & 1 & \rightarrow & +1 \end{array}$$

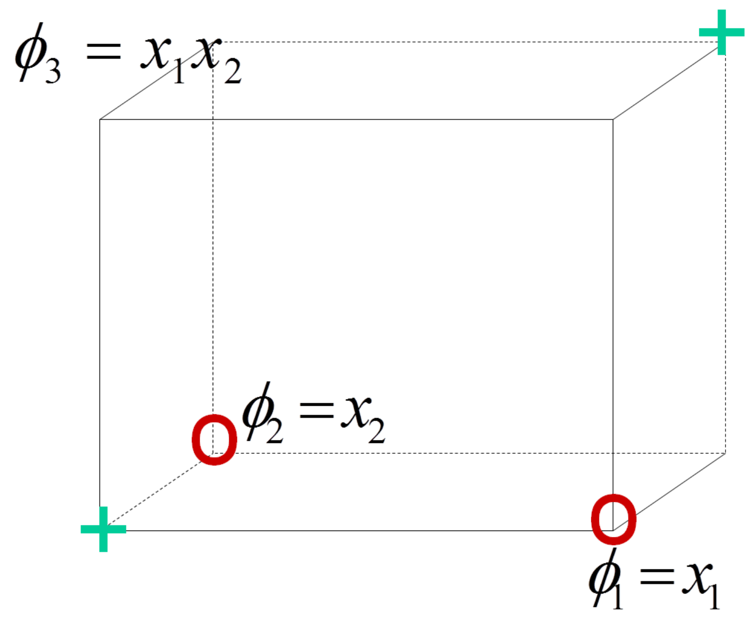
XOR-type problems are not Linearly Separable



Trick: Let's Add Basis Functions

- Linear Model: input variables: x_1, x_2
- Let's consider the product $x_1 x_2$ as additional input
- The interaction term $x_1 x_2$ couples two inputs nonlinearly

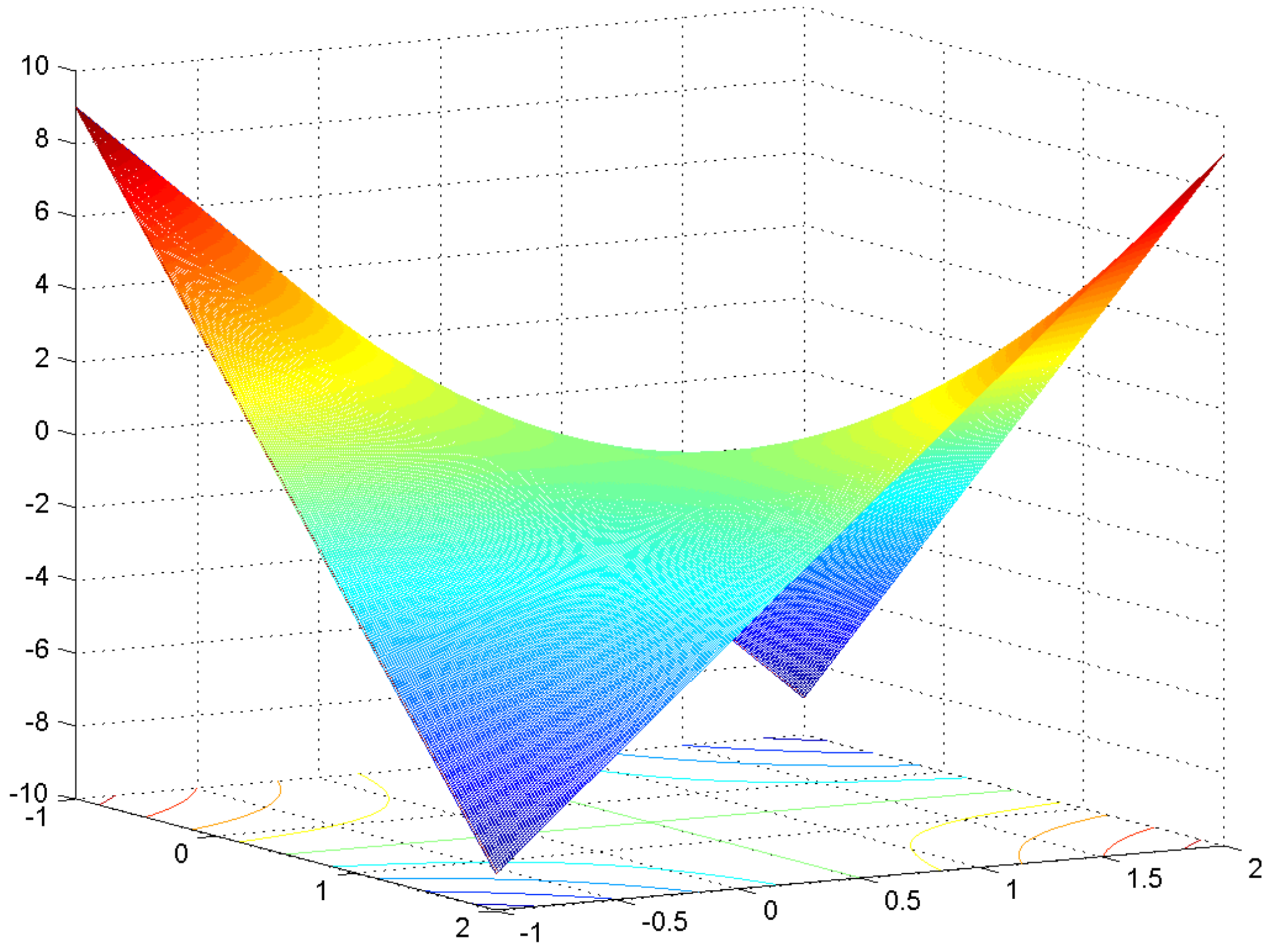
With a Third Input $z_3 = x_1 x_2$ the XOR Becomes Linearly Separable



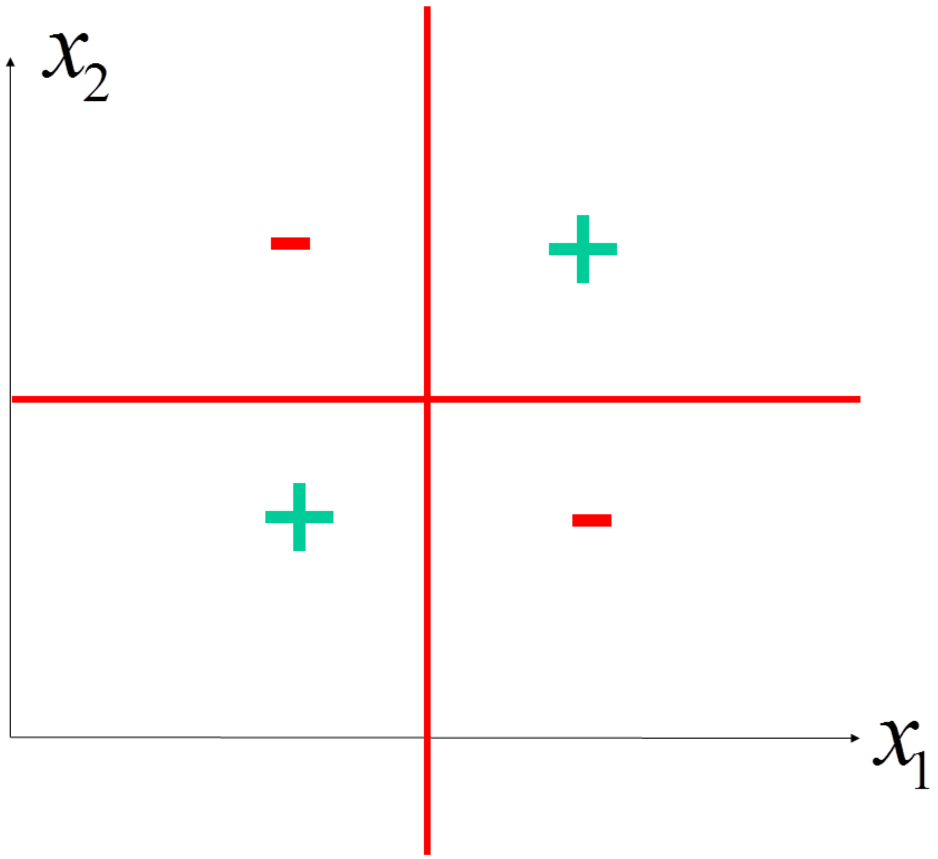
$$f(\mathbf{x}) = 1 - 2x_1 - 2x_2 + 4x_1x_2 = \phi_0(x) - 2\phi_1(x) - 2\phi_2(x) + 3\phi_3(x)$$

with $\phi_0(x) = 1, \phi_1(x) = x_1, \phi_2(x) = x_2, \phi_3(x) = x_1x_2$

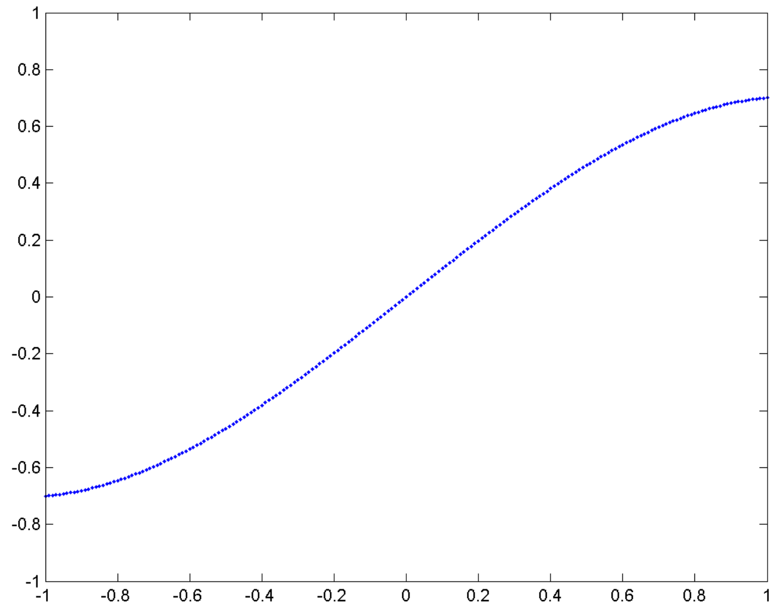
$$f(\mathbf{x}) = 1 - 2x_1 - 2x_2 + 4x_1x_2$$



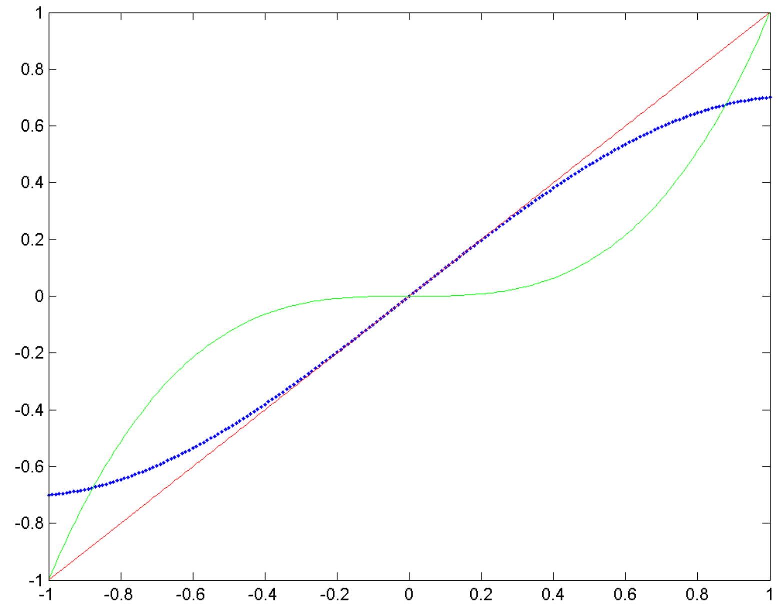
Separating Planes



A Nonlinear Function



$f(x) = x - 0.3x^3$



Basis functions $\phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2, \phi_3(x) = x^3$ und $\mathbf{w} = (0, 1, 0, -0.3)$

Basic Idea

- The simple idea: in addition to the original inputs, we add inputs that are calculated as deterministic functions of the existing inputs, and treat them as additional inputs

- Example: Polynomial Basis Functions

$$\{1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1^2, x_2^2, x_3^2\}$$

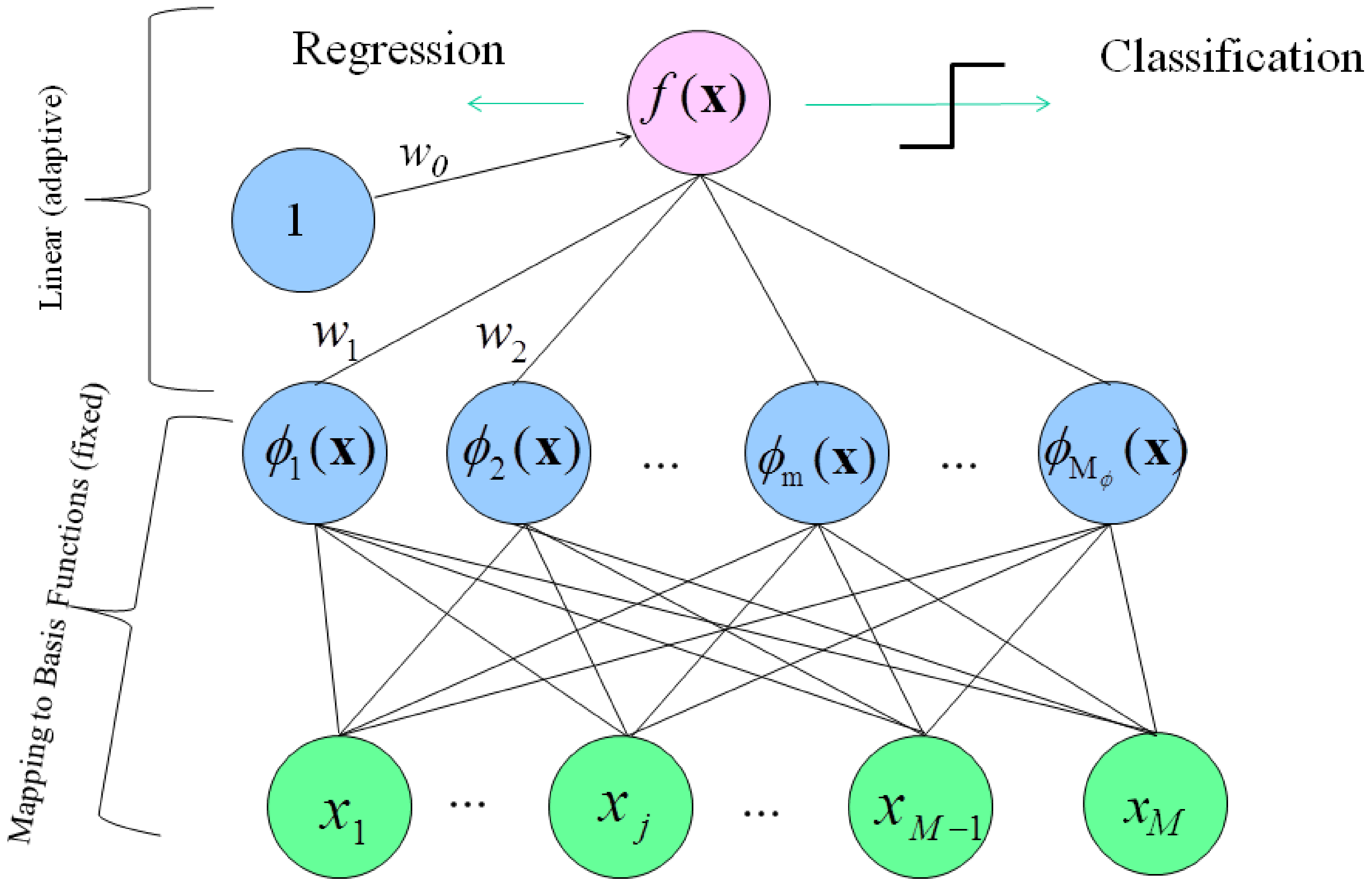
- Basis functions $\{\phi_m(\mathbf{x})\}_{0=1}^{M_\phi}$

- In the example:

$$\phi_0(\mathbf{x}) = 1 \quad \phi_1(\mathbf{x}) = x_1 \quad \phi_5(\mathbf{x}) = x_1x_3 \quad \ldots$$

- Independent of the choice of basis functions, the regression parameters are calculated using the well-known equations for linear regression

Network of Basis Functions

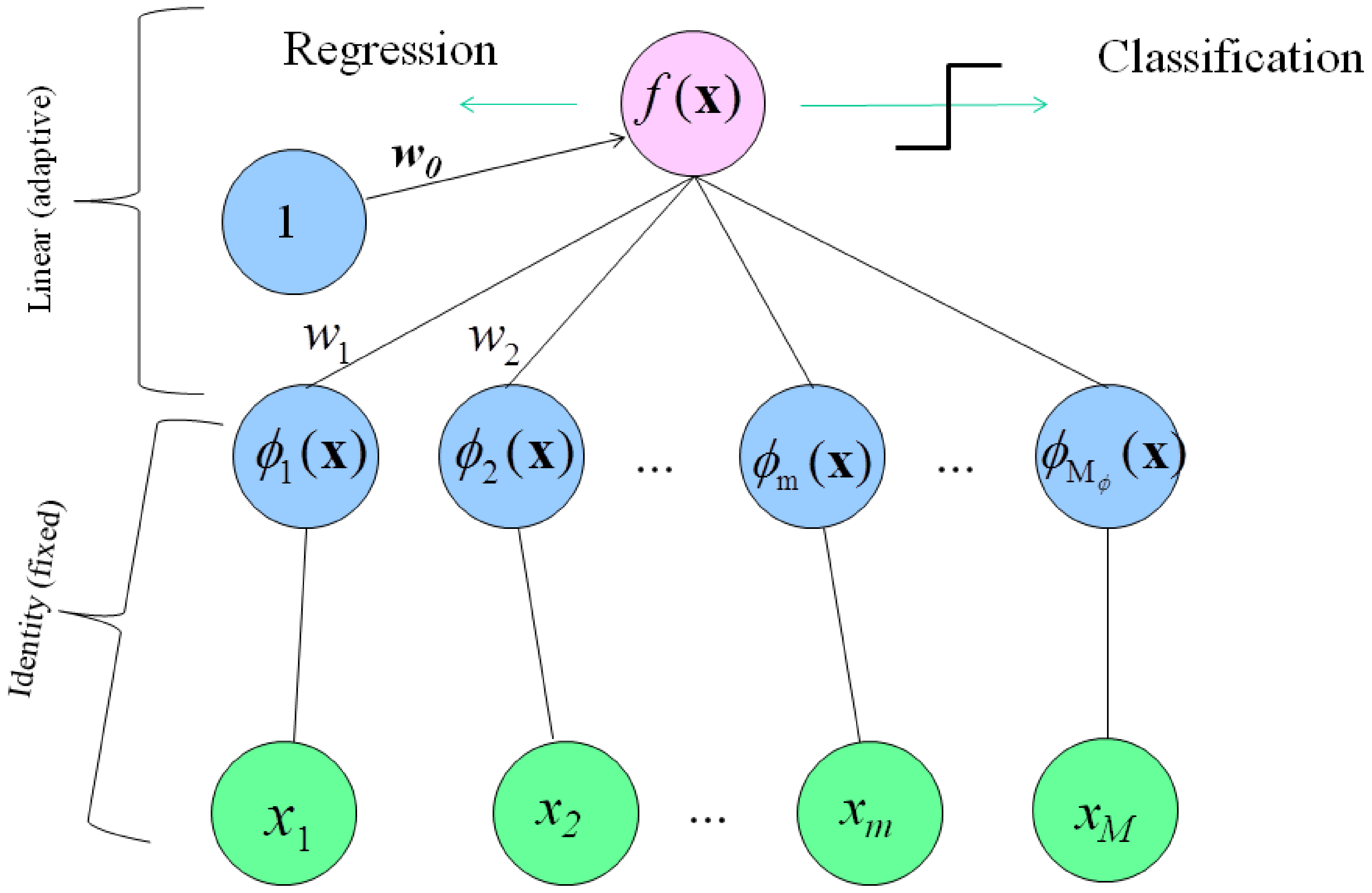


Linear Model Written as Basis Functions

- We can also write a linear model as a sum of basis functions with

$$\phi_0(\mathbf{x}) = 1, \quad \phi_1(\mathbf{x}) = x_1, \quad \dots \quad \phi_M(\mathbf{x}) = x_M$$

Network of Linear Basis Functions



Review: Penalized LS for Linear Regression

- Multiple Linear Regression:

$$f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{j=1}^{M} w_j x_j = \mathbf{x}^T \mathbf{w}$$

- Regularized cost function

$$\text{cost}^{pen}(\mathbf{w}) = \sum_{i=1}^{N} (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2 + \lambda \sum_{j=0}^{M} w_j^2$$

- The penalized LS-Solution gives

$$\hat{\mathbf{w}}_{pen} = \left(\mathbf{X}^T\mathbf{X} + \lambda I\right)^{-1} \mathbf{X}^T \mathbf{y} \quad \text{with} \quad \mathbf{X} = \begin{pmatrix} x_{1,0} & \dots & x_{1,M} \\ \dots & \dots & \dots \\ x_{N,0} & \dots & x_{N,M} \end{pmatrix}$$

Regression with Basis Functions

- Model with basis functions:

$$f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{m=1}^{M_\phi} w_m \phi_m(\mathbf{x})$$

- Regularized cost function with weights as free parameters

$$\text{cost}^{pen}(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - \sum_{m=0}^{M_\phi} w_m \phi_m(\mathbf{x}_i) \right)^2 + \lambda \sum_{m=0}^{M_\Phi} w_m^2$$

- The penalized least-squares solution

$$\hat{\mathbf{w}}_{pen} = \left(\mathbf{\Phi}^T \mathbf{\Phi} + \lambda I \right)^{-1} \mathbf{\Phi}^T \mathbf{y}$$

with

$$\mathbf{\Phi} = \left(\begin{array}{cccc} 1 & \phi_1(\mathbf{x}_1) & \dots & \phi_{M_\phi}(\mathbf{x}_1) \\ \dots & \dots & \dots & \dots \\ 1 & \phi_1(\mathbf{x}_N) & \dots & \phi_{M_\phi}(\mathbf{x}_N) \end{array} \right)$$

Nonlinear Models for Regression and Classification

- Regression:

$$f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{m=1}^{M_\phi} w_m \phi_m(\mathbf{x})$$

  As discussed, the weights can be calculated via penalized LS

- Classification:

$$\hat{y} = \text{sign}(f_{\mathbf{w}}(\mathbf{x})) = \text{sign}\left(w_0 + \sum_{m=1}^{M_\phi} w_m \phi_m(\mathbf{x})\right)$$

  The Perceptron learning rules can be applied, or some other learning rules for linear classifiers, if we replace $1, x_{i,1}, x_{i,2}, \ldots$ with $1, \phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), \ldots$

Which Basis Functions?

- The challenge is to find problem specific basis functions which are able to effectively model the true mapping, resp. that make the classes linearly separable; in other words we assume that the true dependency $f(\mathbf{x})$ can be modelled by at least one of the functions $f_{\mathbf{w}}(\mathbf{x})$ that can be represented by a linear combination of the basis functions, i.e., by one function in the function class under consideration

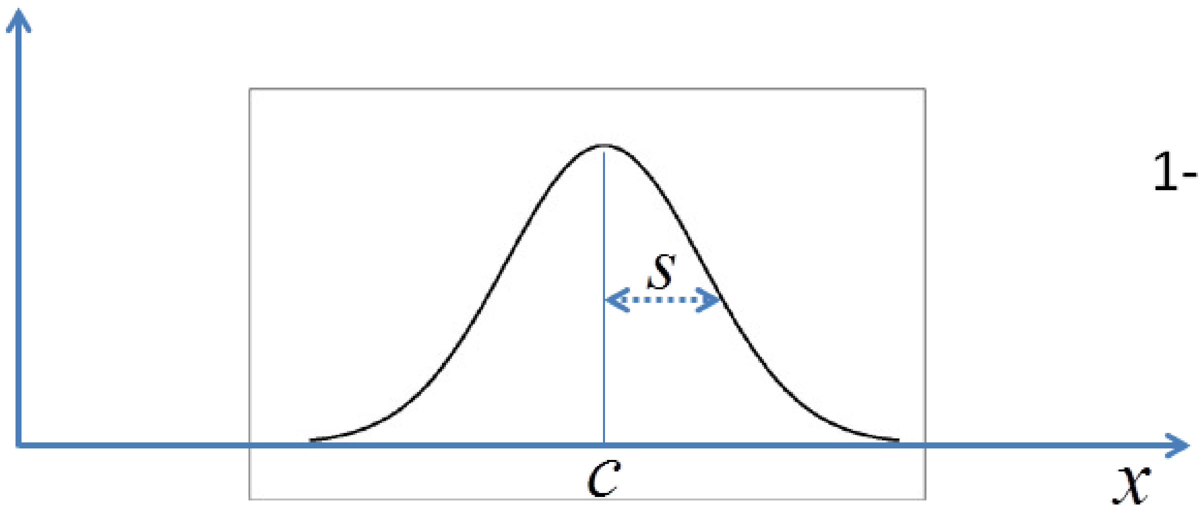
- If we include too few basis functions or unsuitable basis functions, we might not be able to model the true dependency

- If we include too many basis functions, we need many data points to fit all the unknown parameters (This sound very plausible, although we will see in the lecture on kernels that it is possible to work with an infinite number of basis functions)

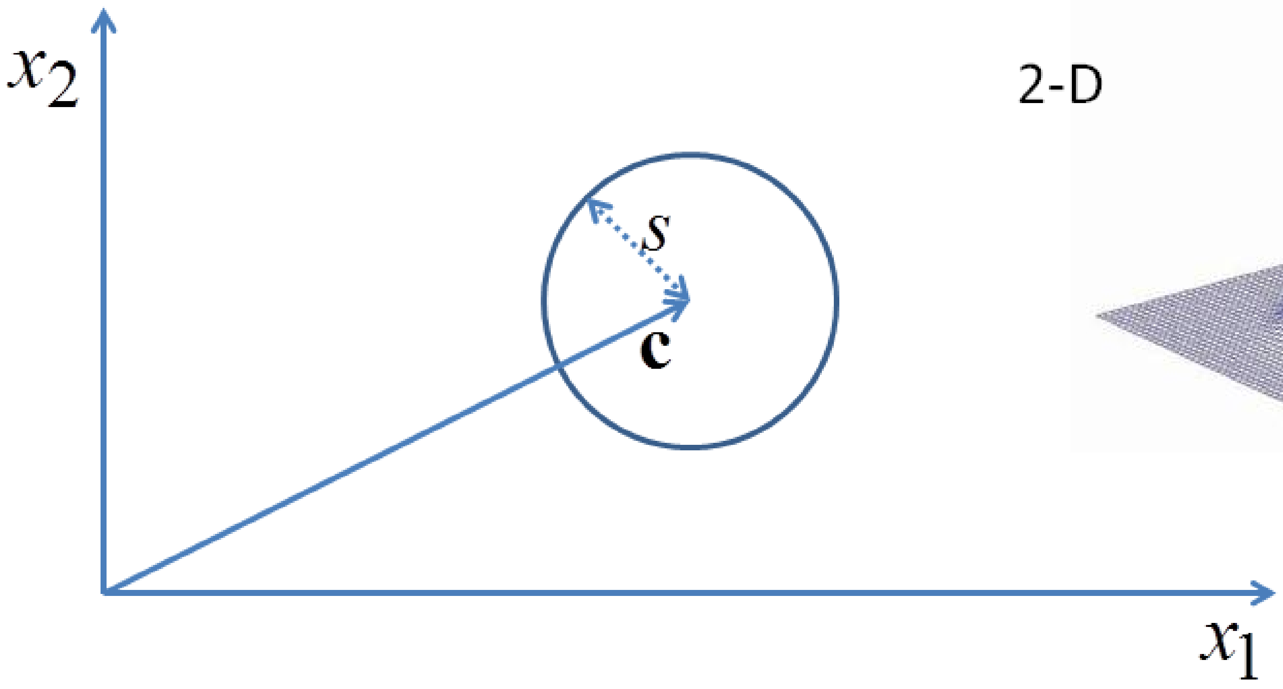
Radial Basis Function (RBF)

- We already have learned about polynomial basis functions
- Another class are radial basis functions (RBF). Typical representatives are Gaussian basis functions

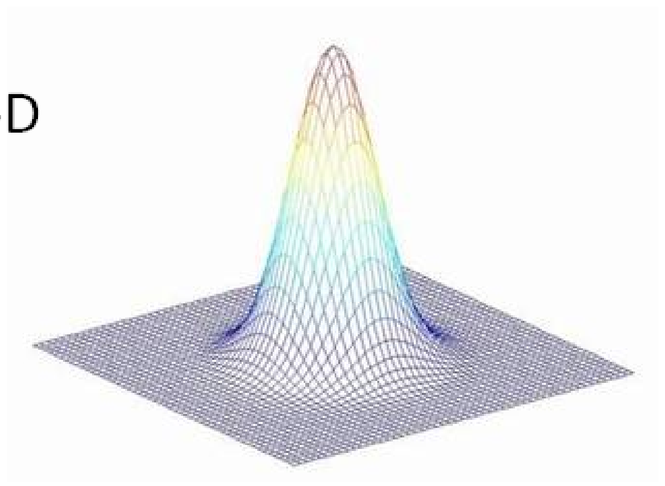
$$\phi_j(\mathbf{x}) = \exp\left(-\frac{1}{2s^2}\|\mathbf{x} - \mathbf{c}_j\|^2\right)$$



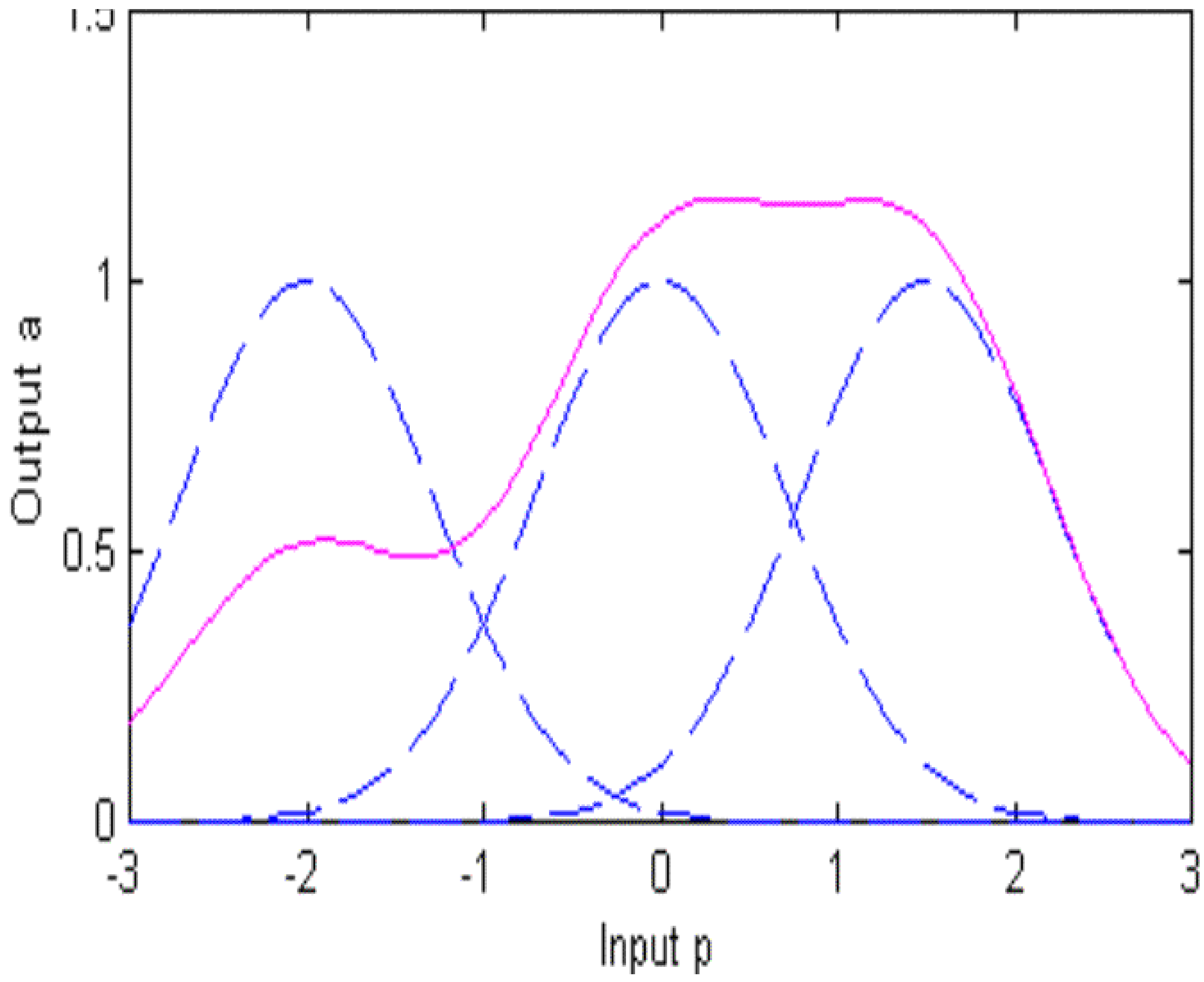
1-D

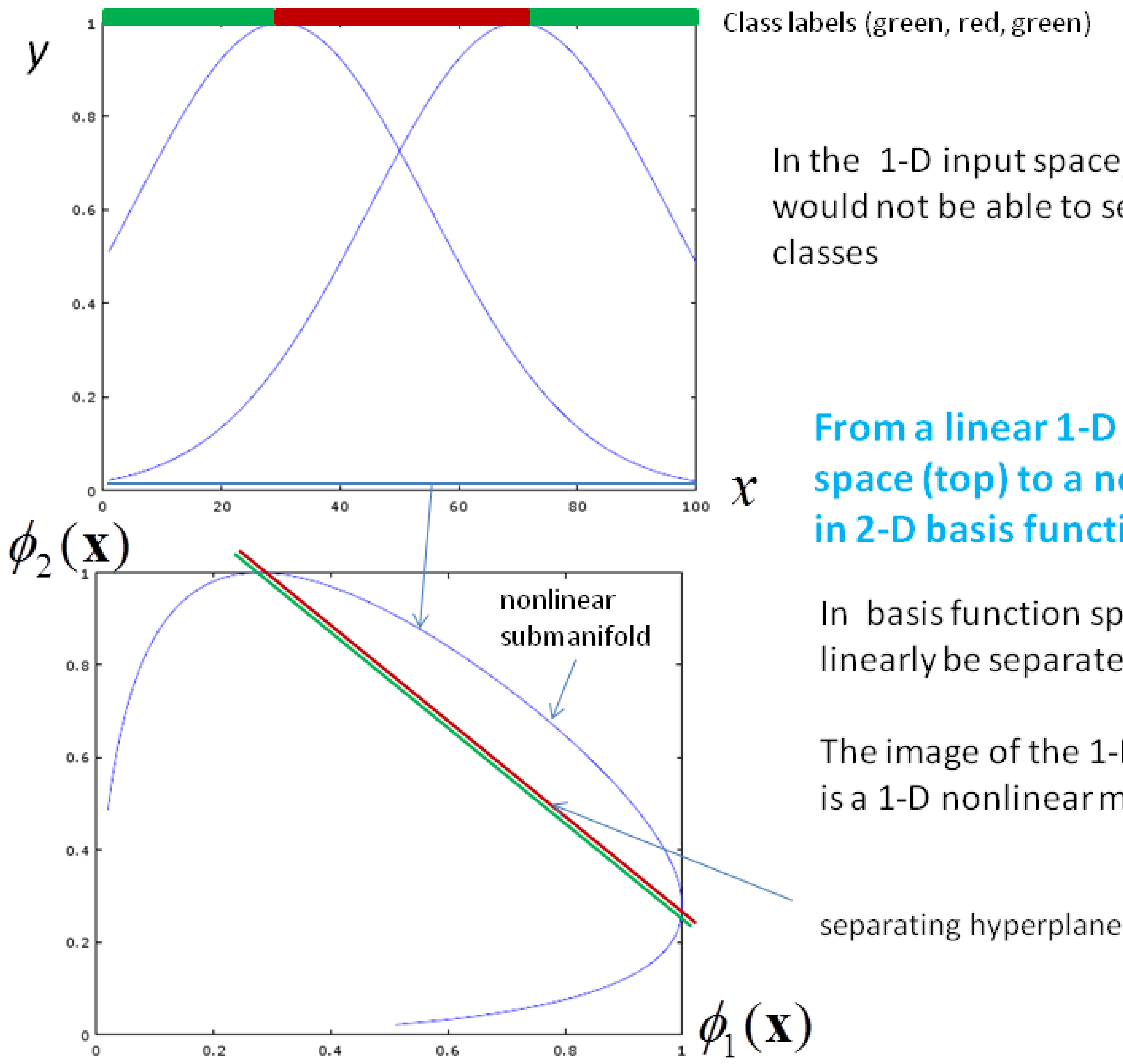


2-D



Three RBFs (blue) form $f(\mathrm{x})$ (pink)





In the 1-D input space, a linear classifier would not be able to separate the two classes

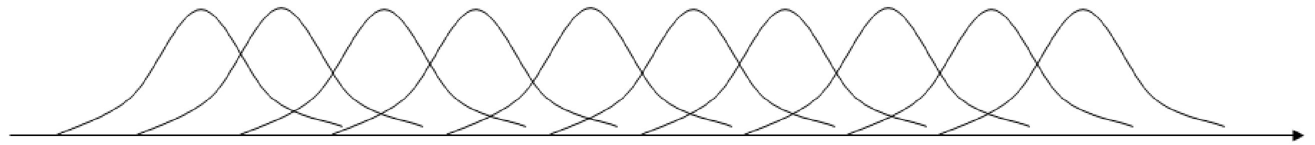
From a linear 1-D manifold in input space (top) to a nonlinear 1-D manifold in 2-D basis function space (bottom)

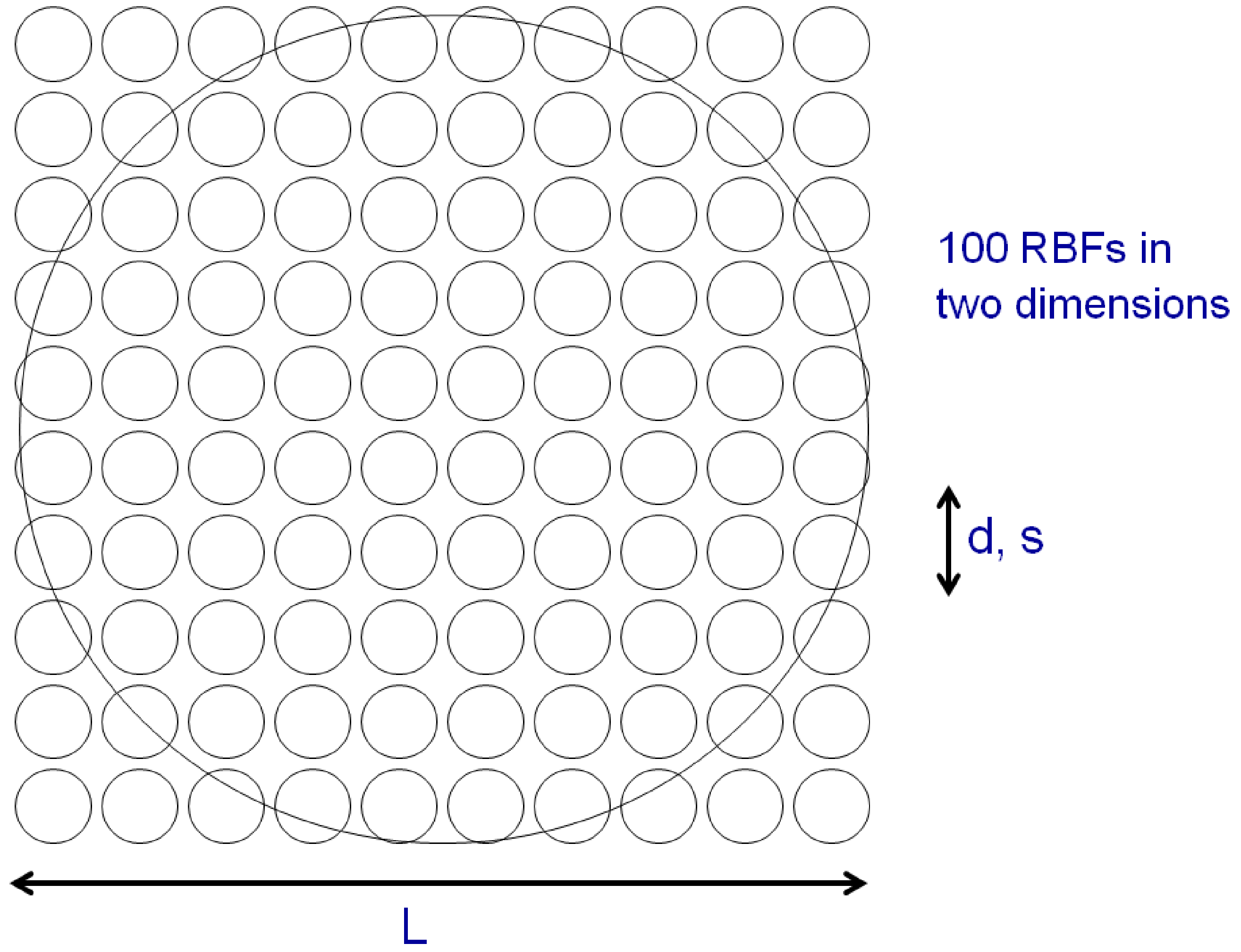
In basis function space, classes can linearly be separated!

The image of the 1-D input data manifold is a 1-D nonlinear manifold

Optimal Basis Functions

- So far all seems to be too simple
- Here is the catch: in some cases, the number of “sensible” basis functions increases exponentially with the number of inputs
- If d is a critical lower length scale of interest and inputs are constraint in a ball of diameter L, then one would need on the order of $(L/d)^M$ RBFs in M dimensions
- We get a similar exponential increase for polynomial basis functions; the number of polynomial basis functions of a given degree increases quickly with the number of dimensions (x^2); (x^2, y^2, xy); $(x^2, y^2, z^2, xy, xz, yz), \dots$
- *The most important challenge: How can I get a small number of relevant basis functions*, i.e., a small number of basis functions that define a function class that contains the true function (true dependency) $f(\mathbf{x})$?


10 RBFs in one dimension


100 RBFs in two dimensions
d, s
L

Forward Selection: Stepwise Increase of Model Class Complexity

- Start with a linear model
- Then we stepwise add basis functions; at each step add the basis function whose addition decreases the training cost the most (greedy approach)
- Examples: Polynomklassifikatoren (OCR, J. Schürmann, AEG)
  - Pixel-based image features (e.g., of hand written digits)
  - Dimensional reduction via PCA (see later lecture)
  - Start with a linear classifier and add polynomials that significantly increase performance
  - Apply a linear classifier

Backward Selection: Stepwise Decrease of Model Class Complexity (Model Pruning)

- Start with a model class which is too complex and then incrementally decrease complexity
- First start with many basis functions
- Then we stepwise remove basis functions; at each step remove the basis function whose removal increases the training cost the least (greedy approach)
- A stepwise procedure is not optimal. The problem of finding the best subset of K basis functions is NP-hard

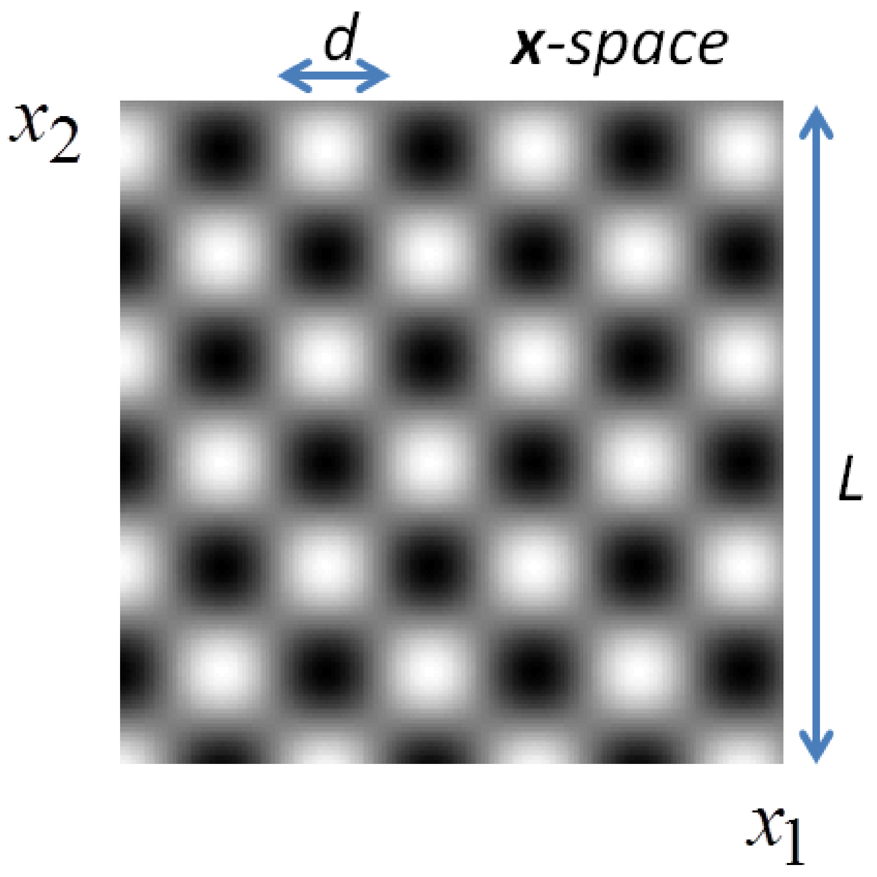
Analysis of Dimensionality

- Consider input space dimension M. All data points are normalized to zero mean and lie in a ball of diameter L
- Let us consider that d is a minimum length scale of interest, e.g., $d = 0.1$. ($\nu = L/2d$ would be an upper frequency of interest); thus y might change significantly (e.g., from 1 to -1) if I move a distance d in some direction
- Thus L/d is a complexity measure: the larger L/d, the more complex the function; in the following figure; $(L/d)^M$ corresponds to the number of maxima and mimima of the underlying function
- Then to be able to fit a nonlinear function, one would need on the order of

$$N >> (L/d)^M$$

data points

2-D Checker Board Function



M=2
d = 0.15
L/d = 6.6

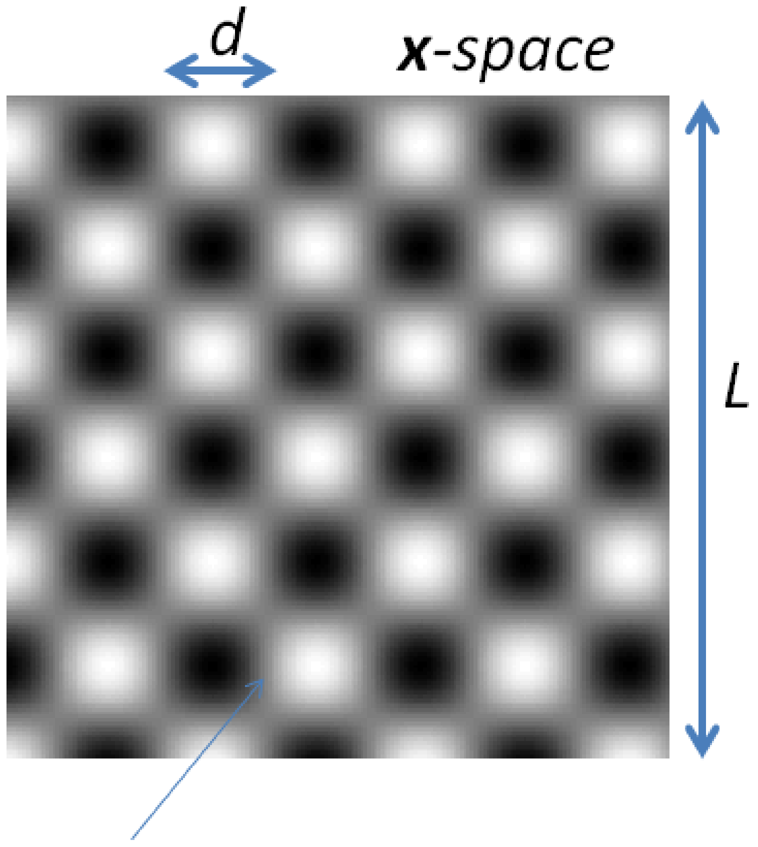
Thus
$N \gg (6.6)^2 = 62$

Here *M=2* is quite small and with 62 RBF basis functions we might get a good fit

HH: Curse of Dimensionality

- Here M is high, and L/d is high (HH)
- Data points are given in the M-dimensional space, where M is large
- L/d is also large
- With $N >> (L/d)^M$ would indicate that we need exponentially many training data points to learn about the functions
- This is the famous "Curse of Dimensionality"

20-D Checker Board Function: "Curse of Dimensionality"



2-D slice through a 20-D input space

M=20

d = 0.15

L/d = 6.6

Thus

$N \gg (6.6)^{20} = 2.5\text{x}10^{16}$

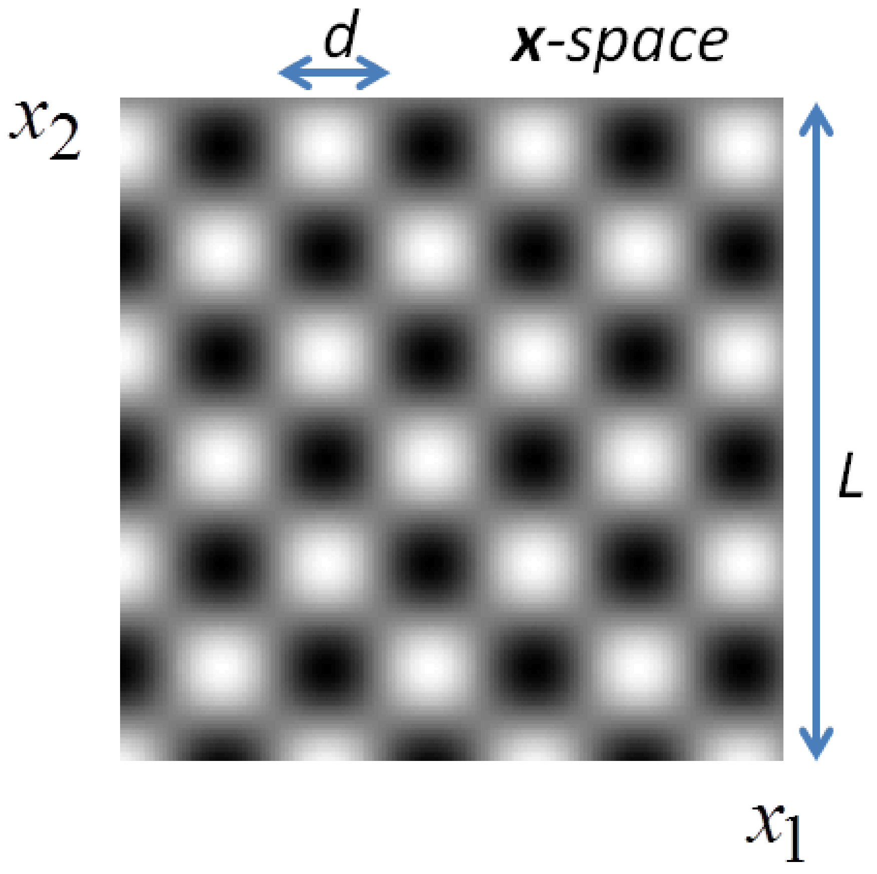
The required number of basis function is huge

LH: Blessing of Dimensionality

- Data points are given in the M-dimensional space, where M is small
- L/d is large
- With a sufficient amount of training data, $N >> (L/d)^M$, the data can explore the relevant input space
- If I map the inputs space to a basis function space of dimension M_ϕ, the data points still lie on an M-dimensional submanifold in the basis function space. So the estimate $N >> (L/d)^M$ might still be valid
- With $M_\phi = N$ basis functions, I can get a perfect fit/separation of the training data in basis function space (regression/classification). (Basis functions must be linearly independent).
- This is what I would call the "Blessing of Dimensionality"
- In basis function space, with $N \approx M_\phi$ effectively $L_\phi/d_\phi \approx 1$

- Still: not all basis function would lead to linear models that perform well on test data; a common choice are RBFs, with optimized length scale s
- With $M_\phi \approx N$, we should definitely apply regularization! A perfect fit on the training data is not our ultimate goal; we are interested in generalization performance

2-D Checker Board Function



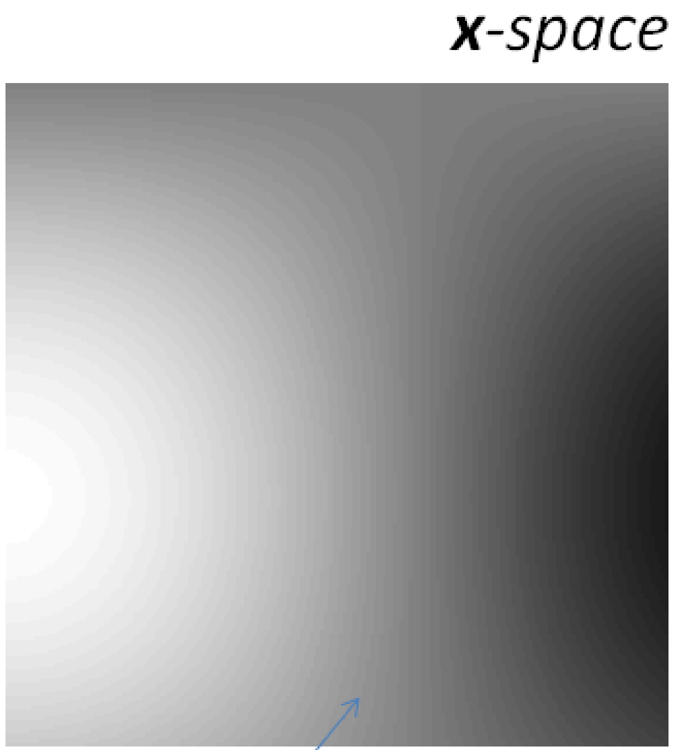
M=2
d = 0.15
L/*d* = 6.6

Thus
$N >> (6.6)^2 = 62$

Here *M=2* is quite small and with 62 RBF basis functions we might get a good fit

HL: No Curse of Dimensionality

- Data points are given in the M-dimensional space, where M is large
- L/d is small
- Then $N >> (L/d)^M$) can still be acceptable
- But: methods that directly depend on the concept of a neighborhood might not work well, since random data points tend to become equidistant in high dimensions. Example: nearest-neighbor classifiers; for these type of model, we again have a "curse of dimensionality"
- Trivially, the fourth situation, small M and small L/d is quite easy to model (LL)



2-D slice through a 20-D input space

$M=20$

$d = 0.8$

$L/d = 1.25$

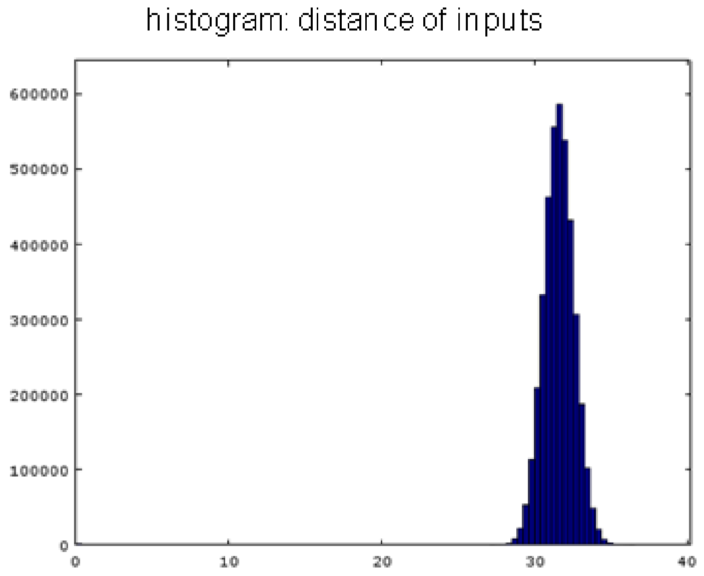
Thus

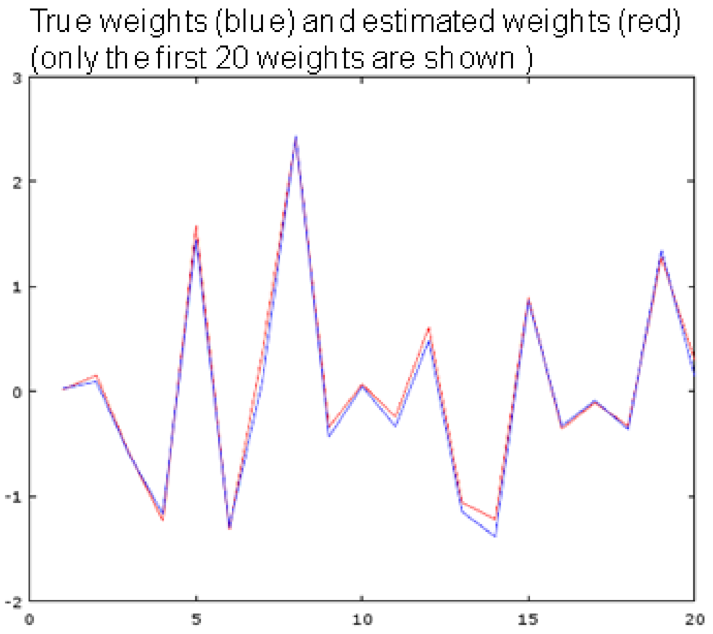
$N \gg (1.25)^{20} = 87$

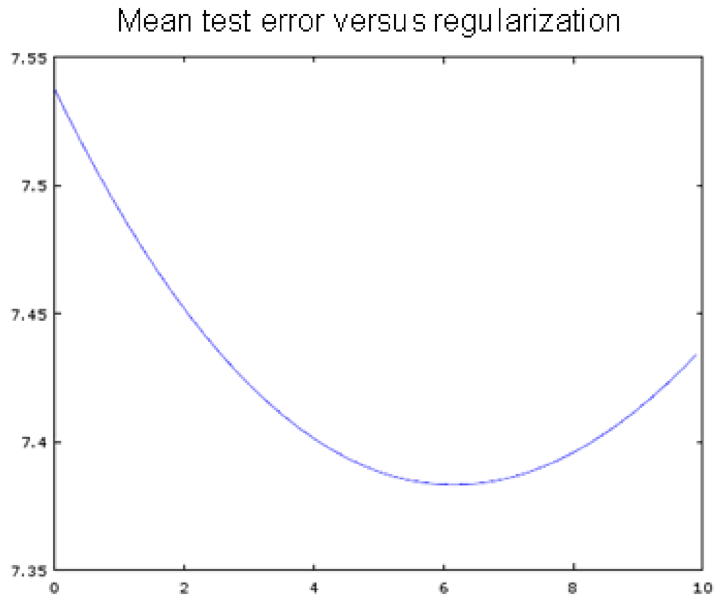
Here *M=20* is medium size and with 87 RBF basis functions we might get a good fit; L/d is quite small

HL: Illustration

- With $M = 500$ input dimensions, we generated $N = 1000$ random data points $\{\mathbf{x}_i\}_{i=1}^{1000}$
- "Curse of dimensionality": near equidistance between data points (see next figure): distance-based methods, such as nearest-neighbor classifiers, might not work well
- No "Curse of dimensionality" if supervised learning is used and the function has low complexity, $L/d \approx 1$:
- A linear regression model with $N = 1000$ training data points gives excellent results


histogram: distance of inputs
600000
500000
400000
300000
200000
100000
0
0
10
20
30
40


True weights (blue) and estimated weights (red)
(only the first 20 weights are shown)
3
2
1
0
-1
-2
0
5
10
15
20


Mean test error versus regularization
7.55
7.5
7.45
7.4
7.35
0
2
4
6
8
10

Data Represented in Some Feature Space

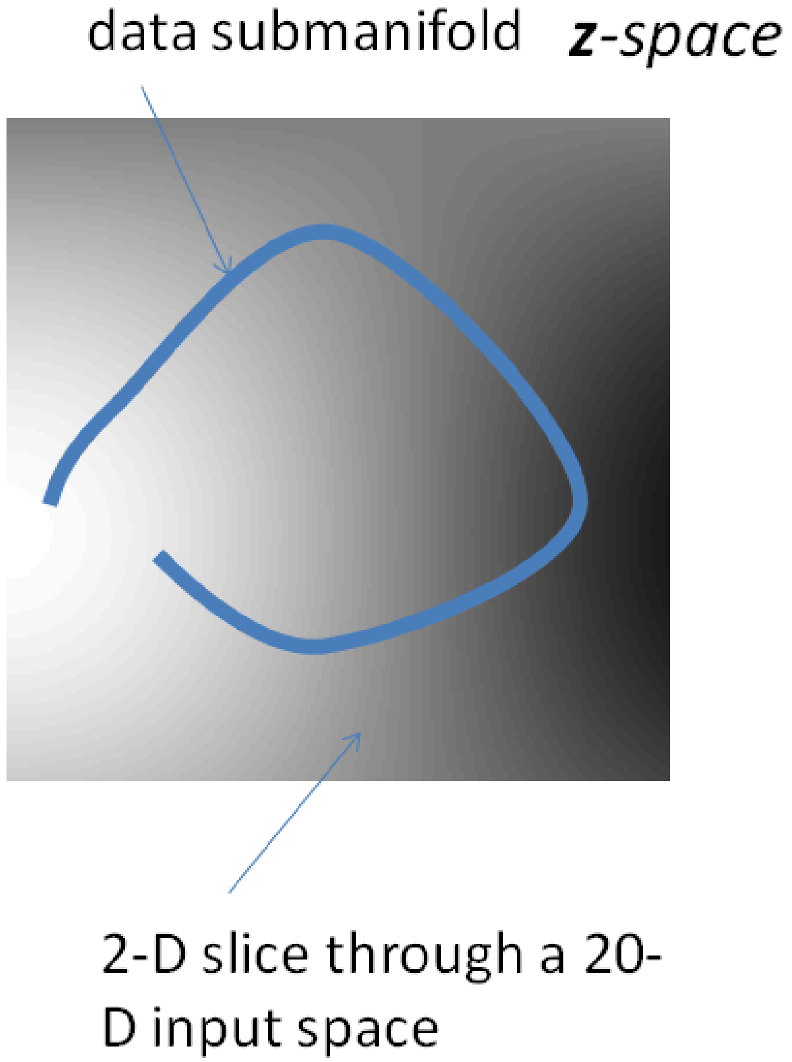
- M is small and L/d is large but we cannot measure $\mathbf{x} \in \mathbb{R}^M$
- Instead, we measure data in feature space $\mathbf{z} \in \mathbb{R}^{M_z}$ either supplied by nature or an application expert (feature engineering)
- Features map

$$\mathbf{x} \to (z_1(\mathbf{x}), z_2(\mathbf{x}), \ldots, z_{M_z}(\mathbf{x}))^T$$

- We say that the data lie in an M-dimensional data submanifold in $\mathbb{R}^{M_z}$
- Of particular interest is when M is small, L/d is large, and M_z is large; LH in F
- In the spirit of the discussion in the lecture on the Perceptron: The map from $\mathbf{x}$ to y might be low-dimensional and explainable, but we can only measure $\mathbf{z}$

LH in F: Data in Feature Space

- M is small and L/d is large but we collect data in some high-dimensional feature space M_z
- In a way, features are like basis functions, but supplied by nature or an application expert (feature engineering)
- If $M_z > (L/d)^M$ linear regression/classification in feature space might solve the problem, or a small number of wide RBFs might solve the problem
- The definition of suitable features for documents, images, gene sequences, ... is a very active research area: feature engineering
- (One point of Deep Learning is that it does not require feature engineering!)



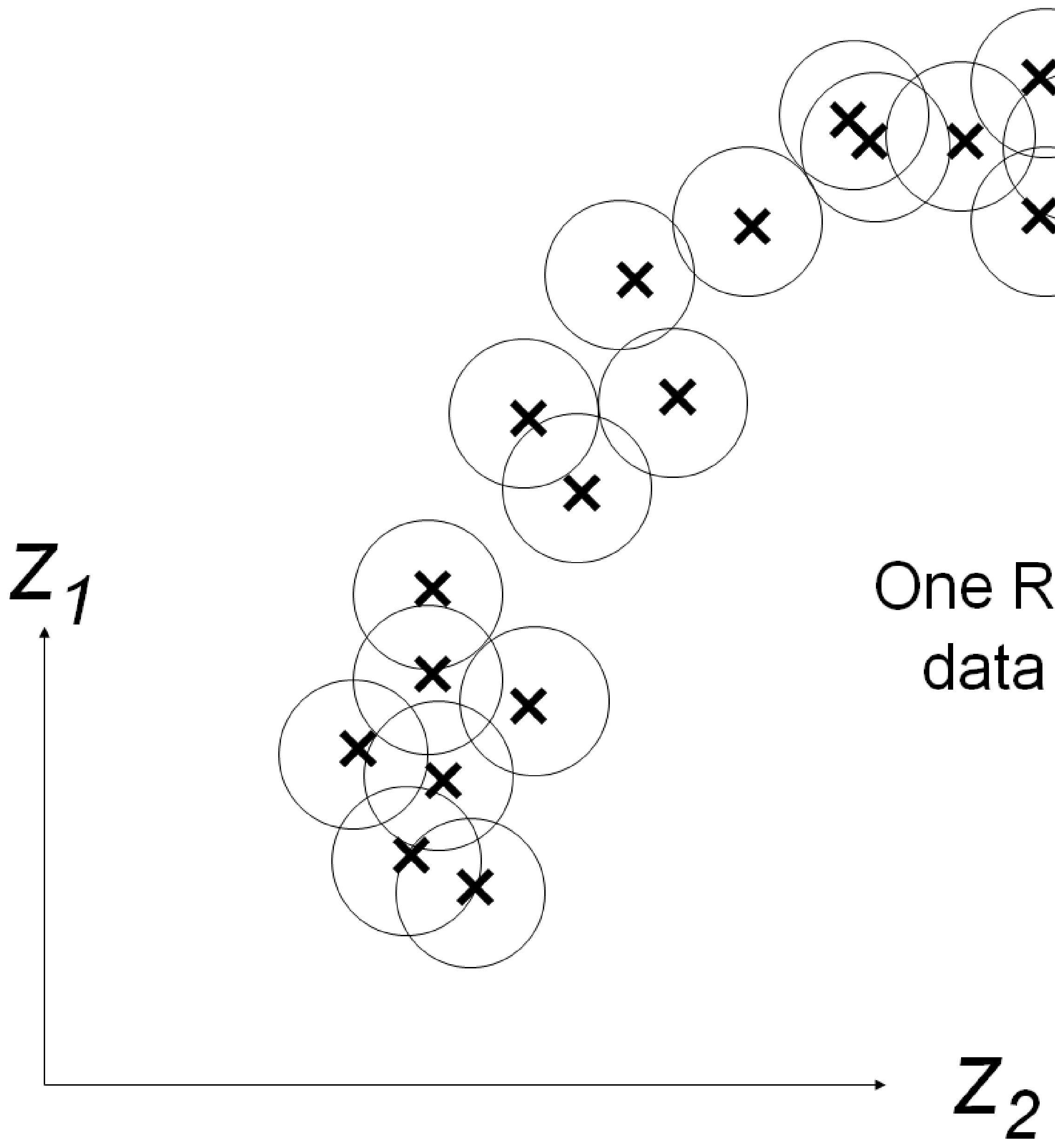
$M_z = 20$
$M = 1$
$L/d = 3$

Thus
$N > (3)^1 = 3$

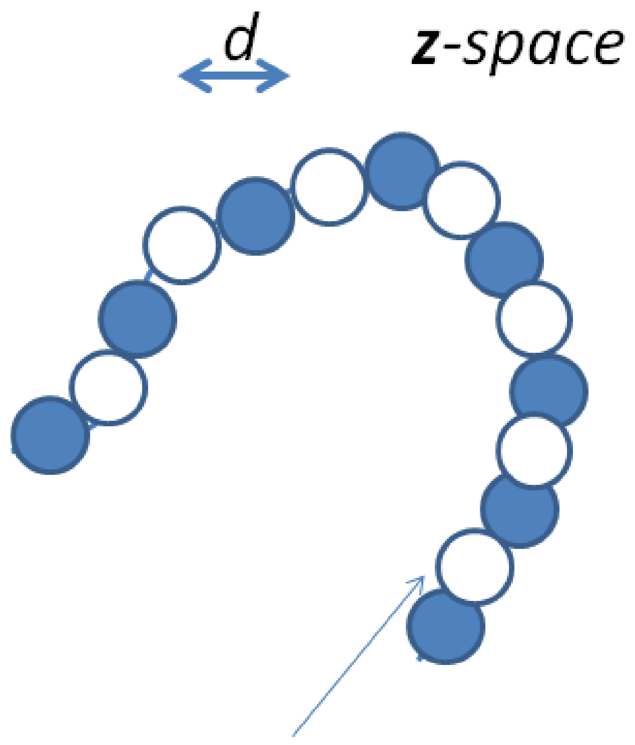
Here *with 3* RBF basis functions we might get a good fit

LH in F: Putting Resources on the Data Submanifold (“Kernels”)

- Sometimes a feature space M_z is large, but still $M_z << (L/d)^M$
- In this situation, it might work to put the RBFs on the data submanifold, where we assume that the manifold can be represented by the training data points
- Use one RBF per data point. The centers of the RBFs are simply the data points themselves and the widths are determined via some heuristics (or via cross validation, see later lecture)



One RBF per data point



1-D manifold in a 20-D feature space

$M=1$

$M_f=20$

$L/d = 15$

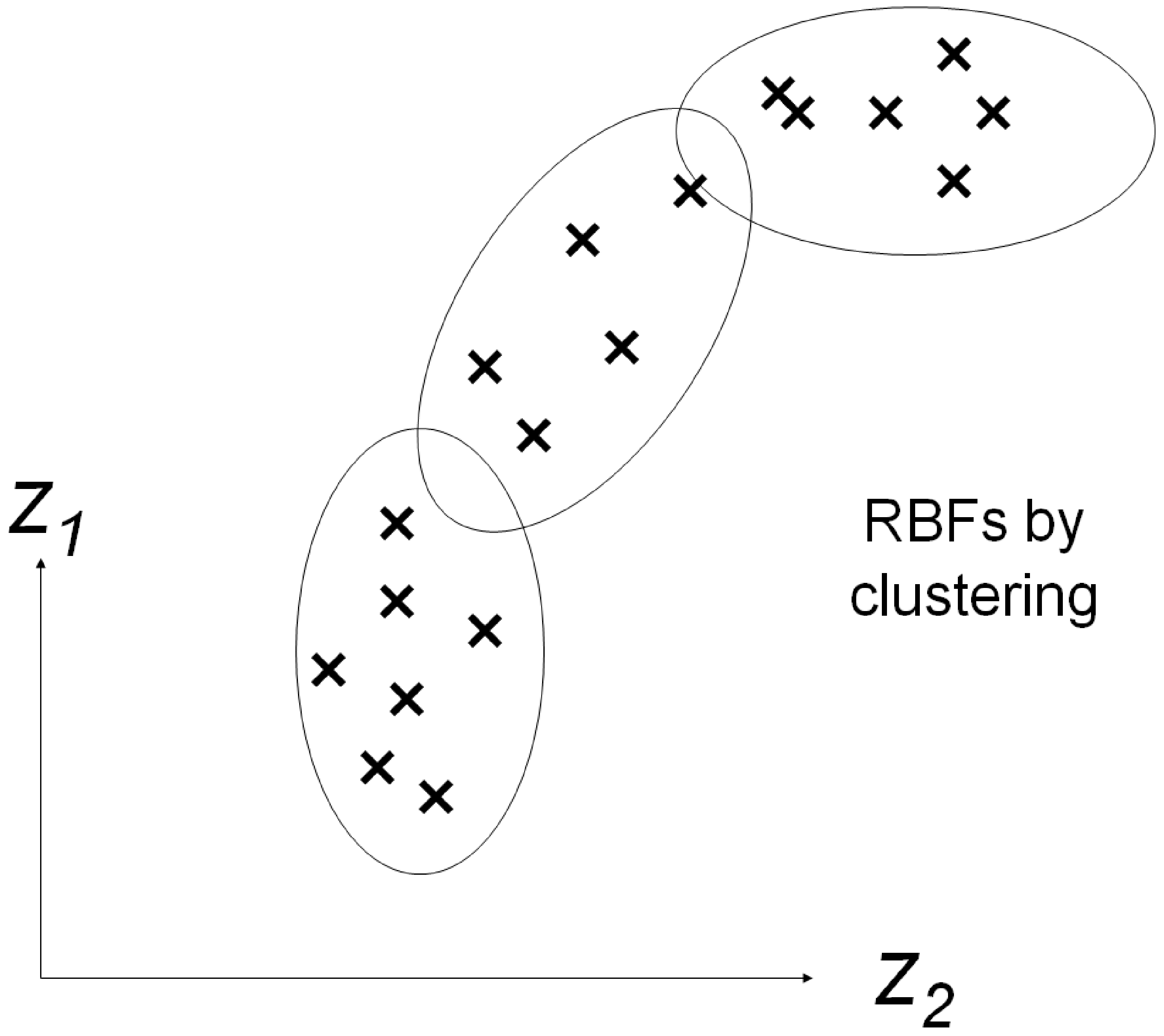
Thus

$N \gg (15)^1 = 15$

I might ge a good fit with 15 basis functions

LH in F: Clustering to Finding the Data Submanifold

- Sometimes, data are clustered in M-space, and also in M_z-space
- It might be sensible to first group (cluster) data in feature space and to then use the cluster centers as positions for the Gaussian basis functions; the widths of the Gaussian basis functions might be derived from the sample covariance of the data in the cluster
- Even simpler: we assign a unique output label to each cluster



RBFs by clustering

Data Represented in a Noisy Feature Space

- Finally, we assume that the features are noisy. Instead of

$$\mathbf{z} = \text{featureMap}(\mathbf{x})$$

  we might have

$$\mathbf{z} = \text{featureMap}(\mathbf{x} + \vec{\delta}) + \vec{\epsilon}$$

  where $\vec{\delta}$ and $\vec{\epsilon}$ are noise vectors

- Now data do not lie on the M-dimensional manifold anymore
- We call this situation: LH in noisy F

LH in noisy F: Mapping to the Submanifold

- One might attempt to find the original data submanifold and then apply basis functions

$$\mathbf{z} \rightarrow \hat{\mathbf{x}} \rightarrow \phi(\hat{\mathbf{x}}) \rightarrow y$$

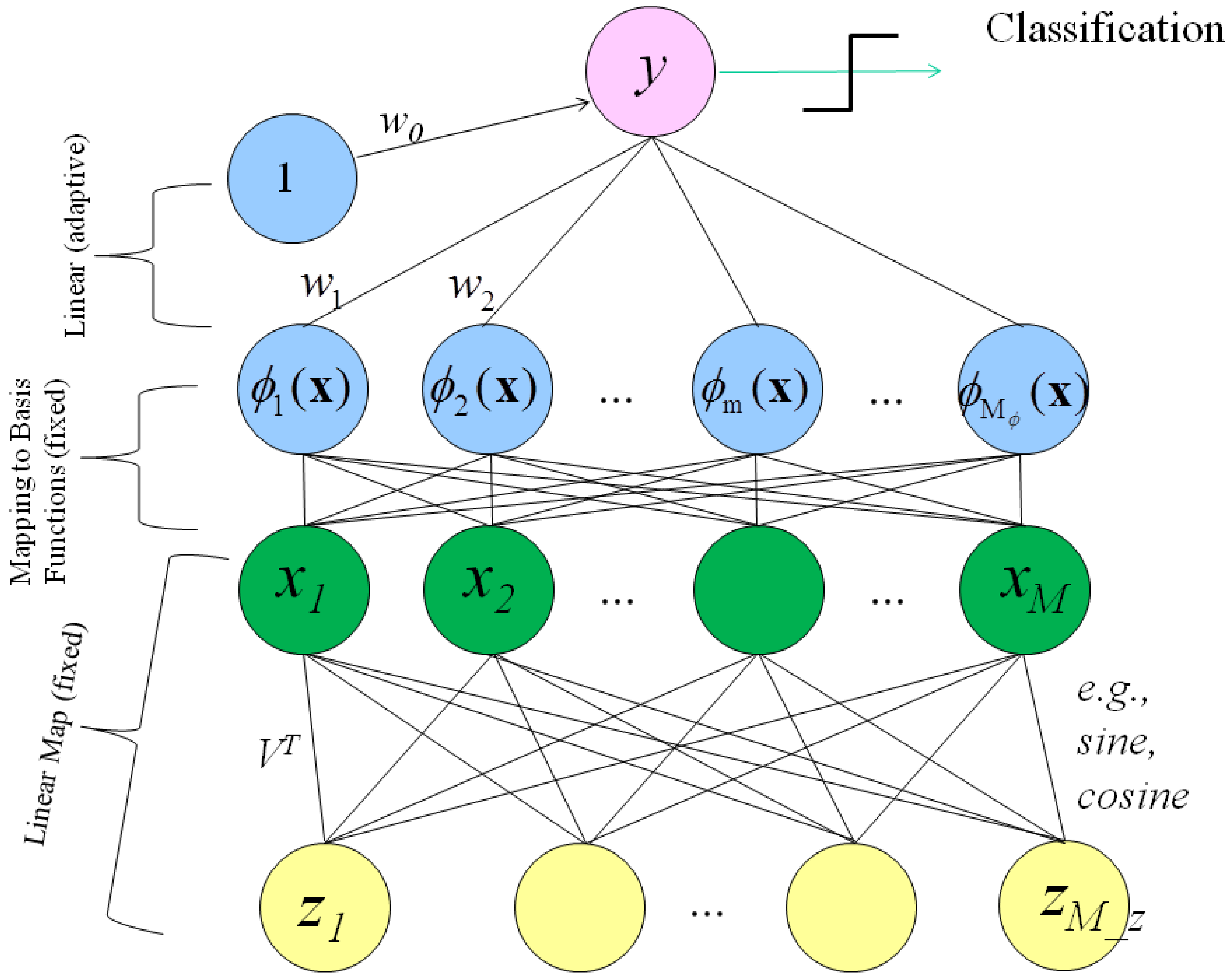
- Most of the times, the mapping $\mathbf{z} \rightarrow \hat{\mathbf{x}}$ is performed by a linear transformation (PCA, later lecture); deep neural autoencoders can also perform nonlinear transformations (lecture on deep learning)

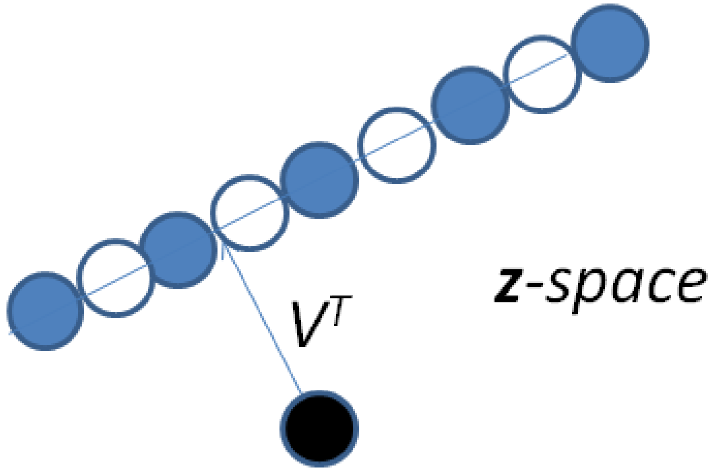
Example: Image Classification

- Typically we treat an image as an input feature vector to a supervised system; we treat the pixels as features!
- M_z is quite large, e.g., $M_z = 256 \times 256 = 65536$!
- Thus $z_{256(k-1)+l} = greyValue(k, l)$
- One can attempt to find the submanifold via a linear transformation

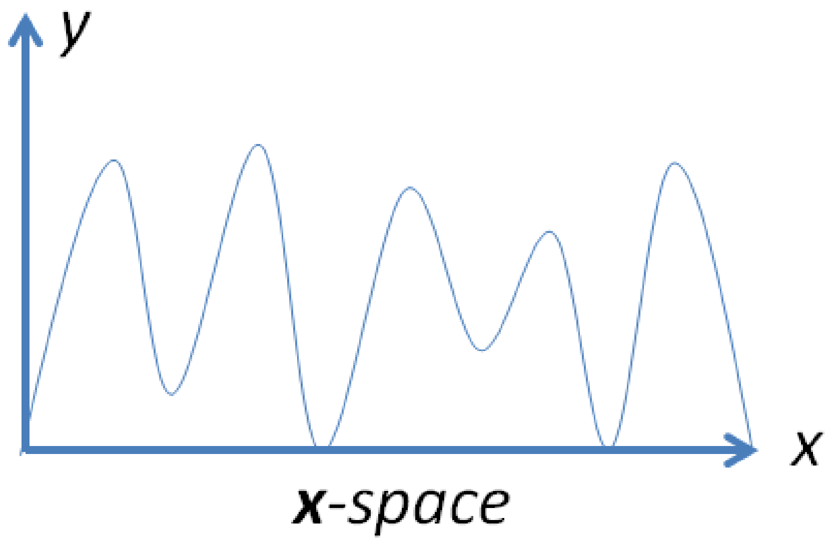
$$\mathbf{x} = \mathbf{V}^T \mathbf{z}$$

- Alternatively, we simply take M low dimensional frequency components (2-D cosine transform)
- Then we apply a basis function classifier (see next slide)


Classification
y
1
w_0
w_1
w_2
Linear (adaptive)
$\phi_1(\mathbf{x})$
$\phi_2(\mathbf{x})$
…
$\phi_m(\mathbf{x})$
…
$\phi_{M_\phi}(\mathbf{x})$
Mapping to Basis Functions (fixed)
x_1
x_2
…
…
x_M
Linear Map (fixed)
V^T
e.g., sine, cosine
z_1
…
z_{M_z}



Dimension reduction: 1-D manifold in a 20-D feature space



$M=1$

$M_f=20$

$L/d = 9$

Thus

$N \gg (9)^1 = 9$

The input is mapped to a 1-D projection, one might get a good fit with 9 or more basis functions

A Function as a Vector

- Note that we have treated a 2-D function (an image) as a vector in a vector space
- This view on functions was introduced and formalized by David Hilbert (Hilbert space)

Conclusions

- Basis functions perform a nonlinear transformation from input space to basis function space
- For a good model fit, for RBF basis functions, one needs $M_\phi >> (L/d)^M$, $N >> (L/d)^M$, thus either M should be small or $L/d \approx 1$
- If one selects $M_\phi \approx N$, one requires on the order of N^3 computations to learn the parameters
- A practical bound is $(L/d)^M << 100000$, for systems with basis functions to be computationally feasible