Basis Functions

Volker Tresp
Summer 2019

Nonlinear Mappings and Nonlinear Classifiers

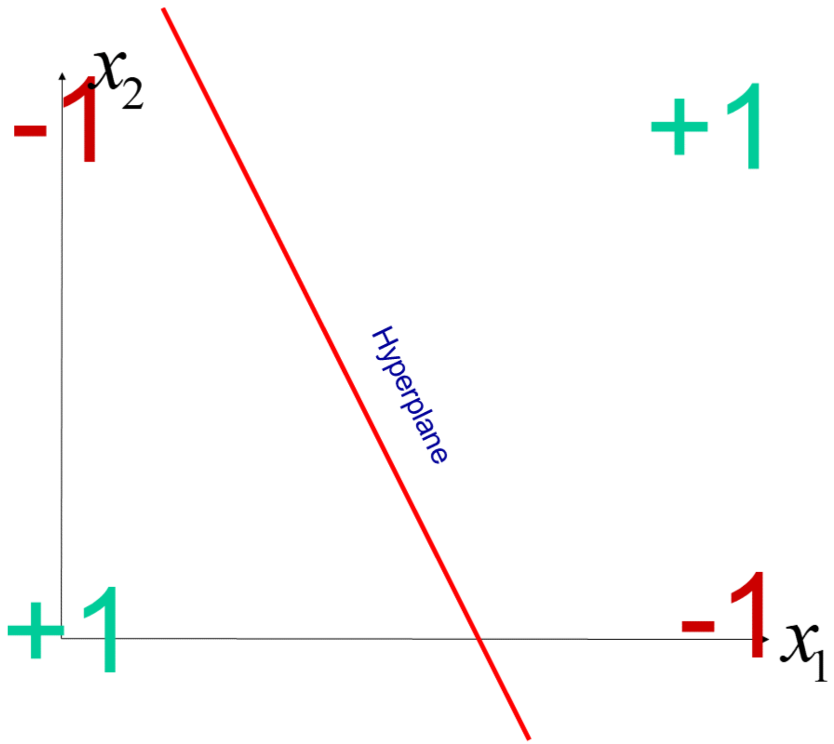
- Regression:
  - Linearity is often a good assumption when many inputs influence the output
  - Some natural laws are (approximately) linear $F = ma$
  - But in general, it is rather unlikely that a true function is linear
- Classification:
  - Linear classifiers also often work well when many inputs 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

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

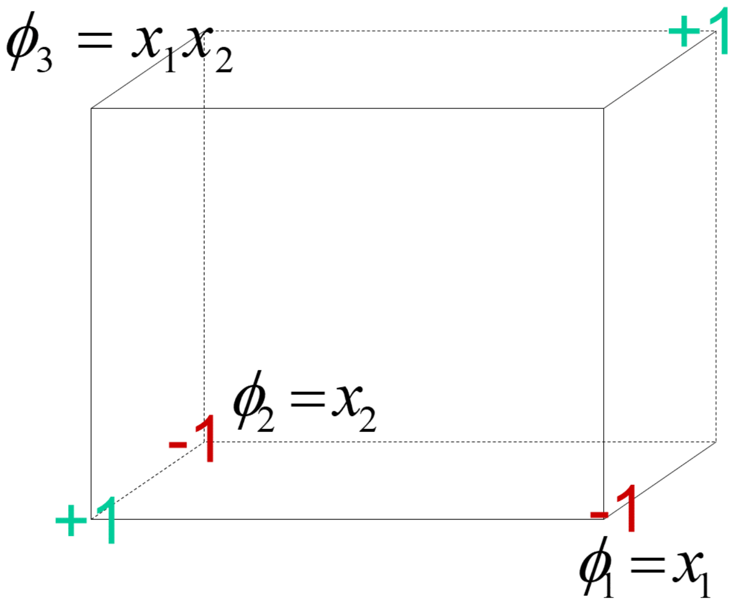
XOR-type problems are not Linearly Separable



Trick: Let's Add Basis Functions

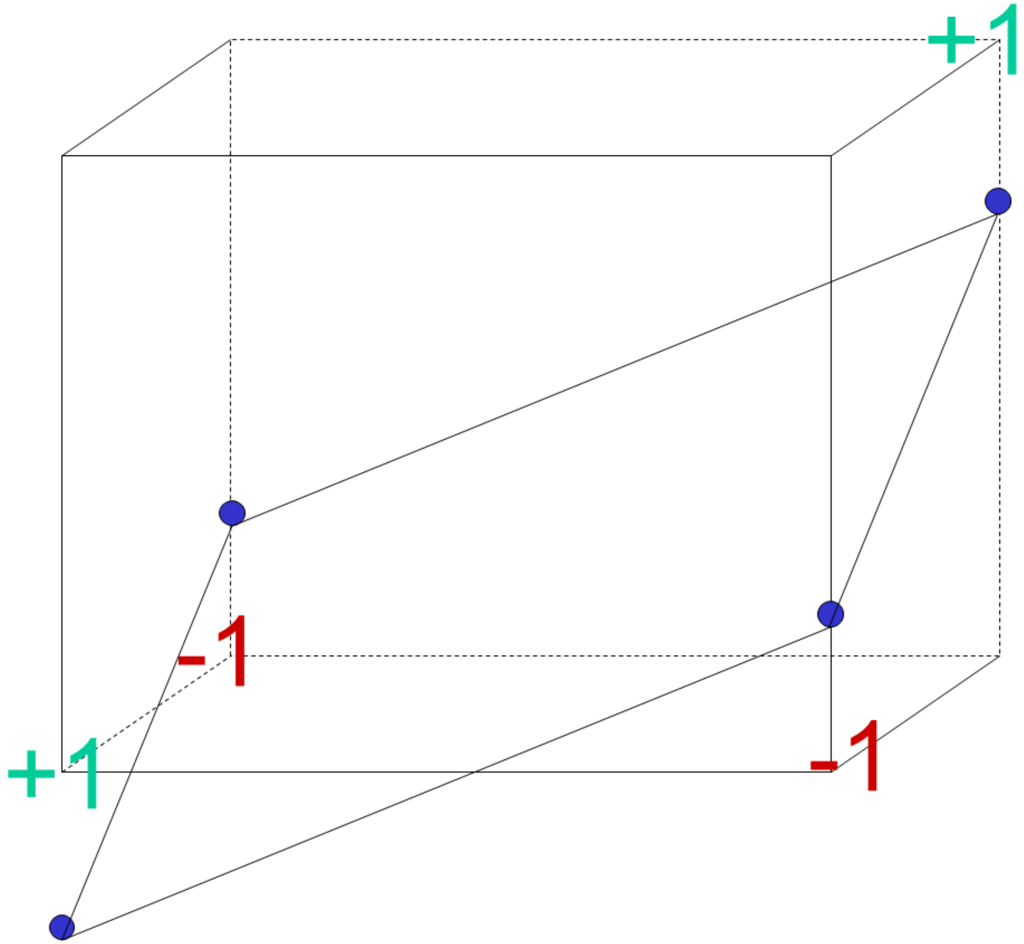
- Linear Model: input variables: x_1, x_2
- Let's consider the product $x_1 x_2$ as additional input
- The interaction term $x_1 x_2$ couples two inputs nonlinearly

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

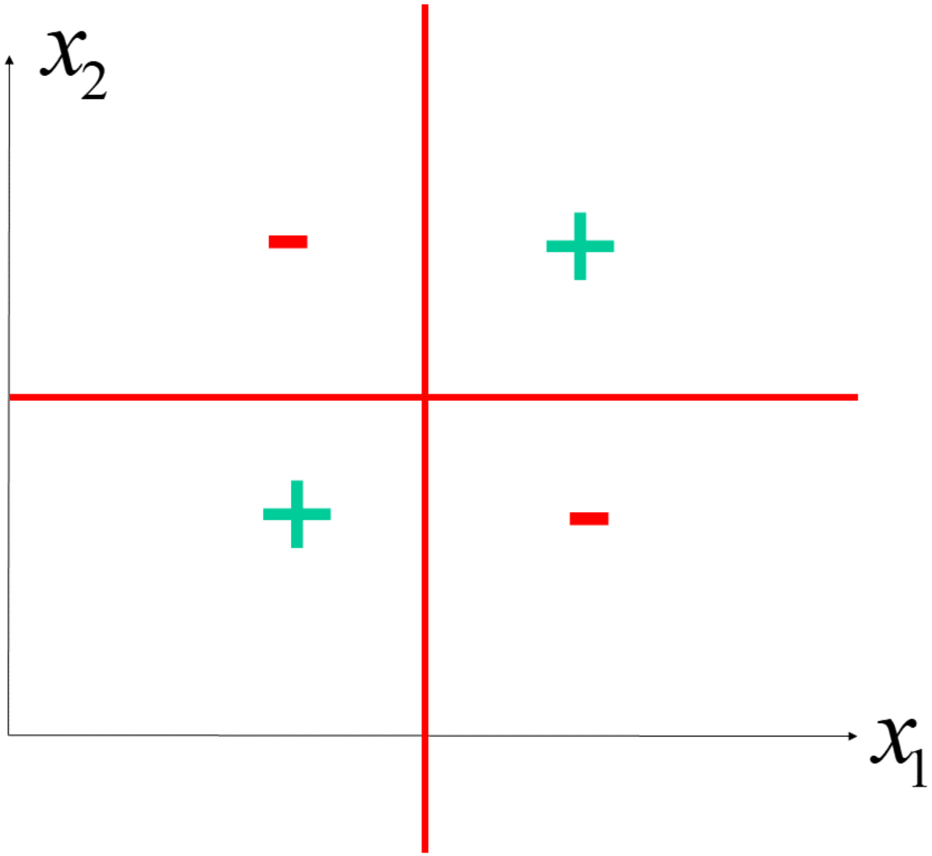


$$f(\mathbf{x}) = 1 - 2x_1 - 2x_2 + 4x_1x_2 = \phi_0(x) - 2\phi_1(x) - 2\phi_2(x) + 4\phi_3(x)$$

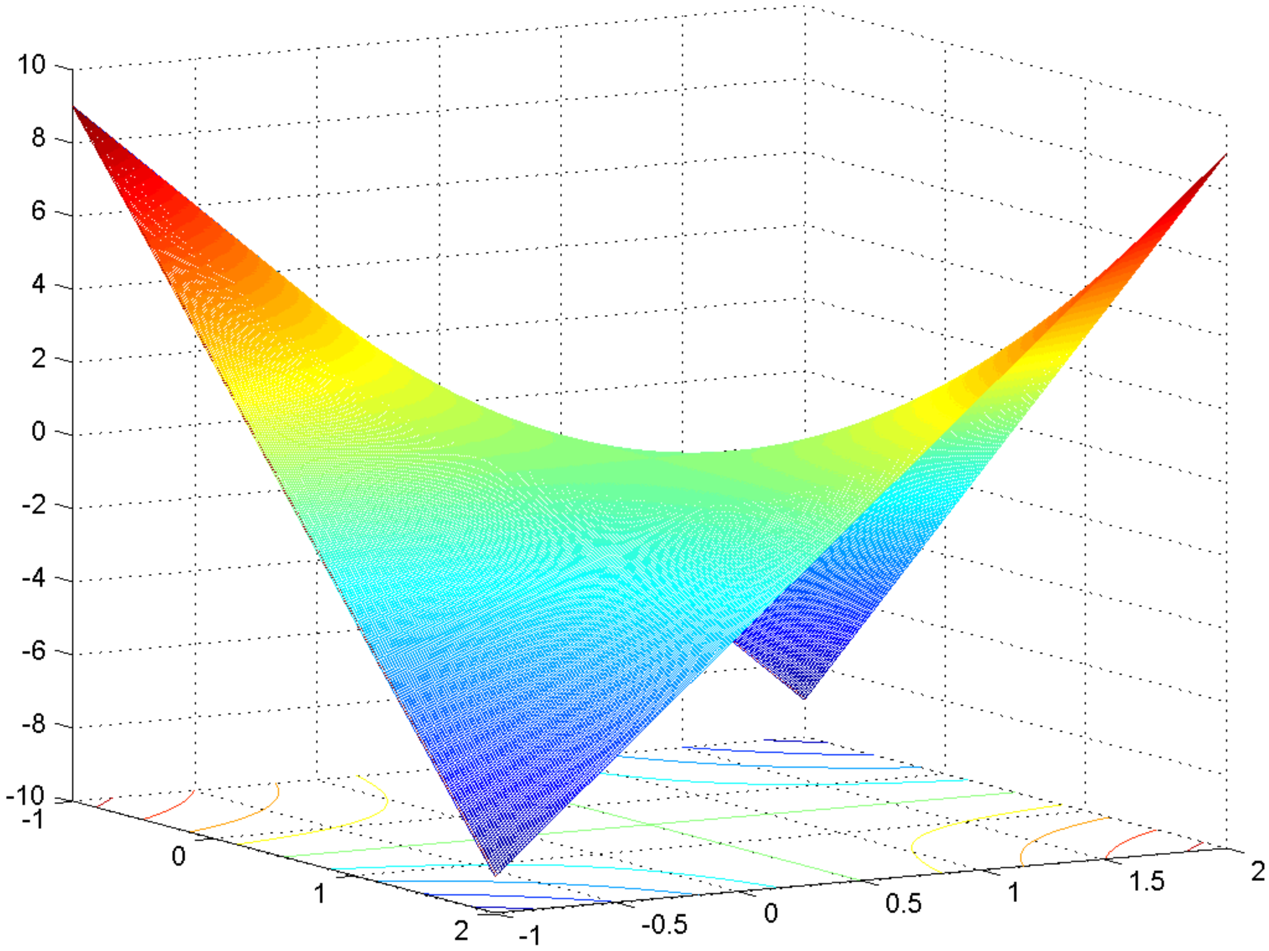
with $\phi_0(x) = 1, \phi_1(x) = x_1, \phi_2(x) = x_2, \phi_3(x) = x_1x_2$


+1
-1
+1
-1

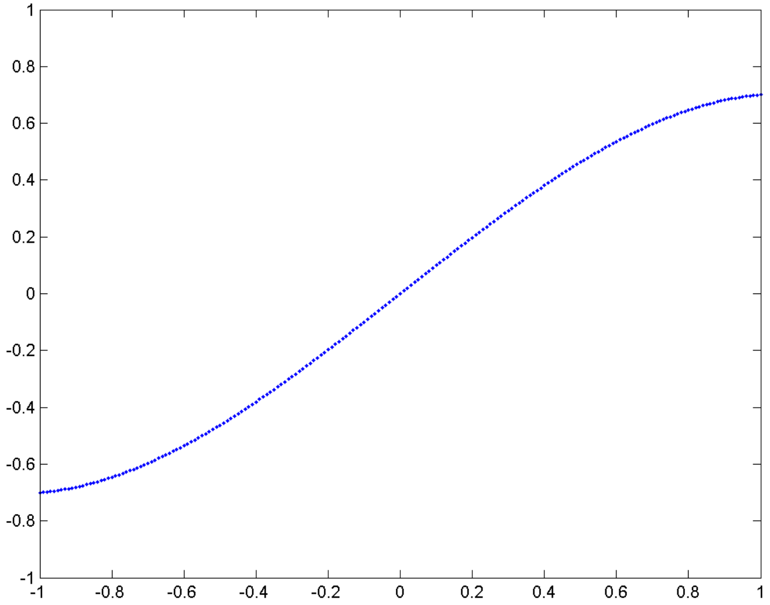
Separating Planes



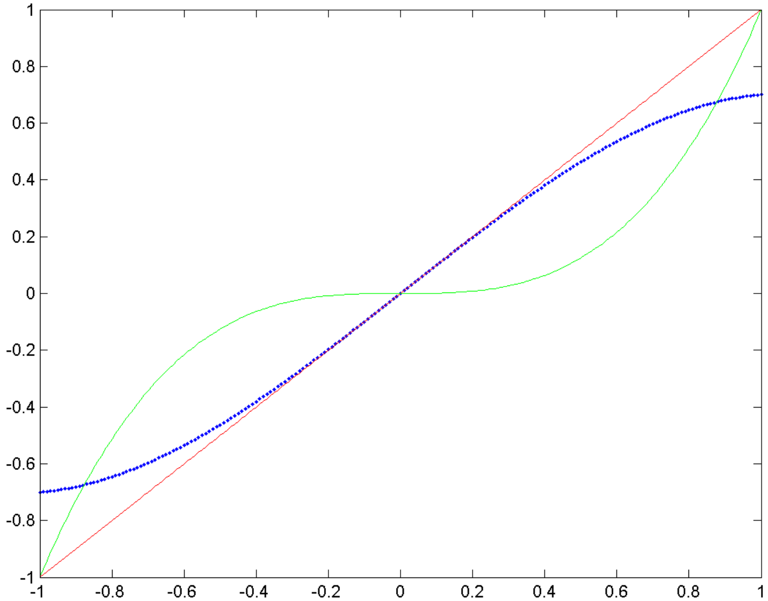
$$f(\mathbf{x}) = 1 - 2x_1 - 2x_2 + 4x_1x_2$$



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

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

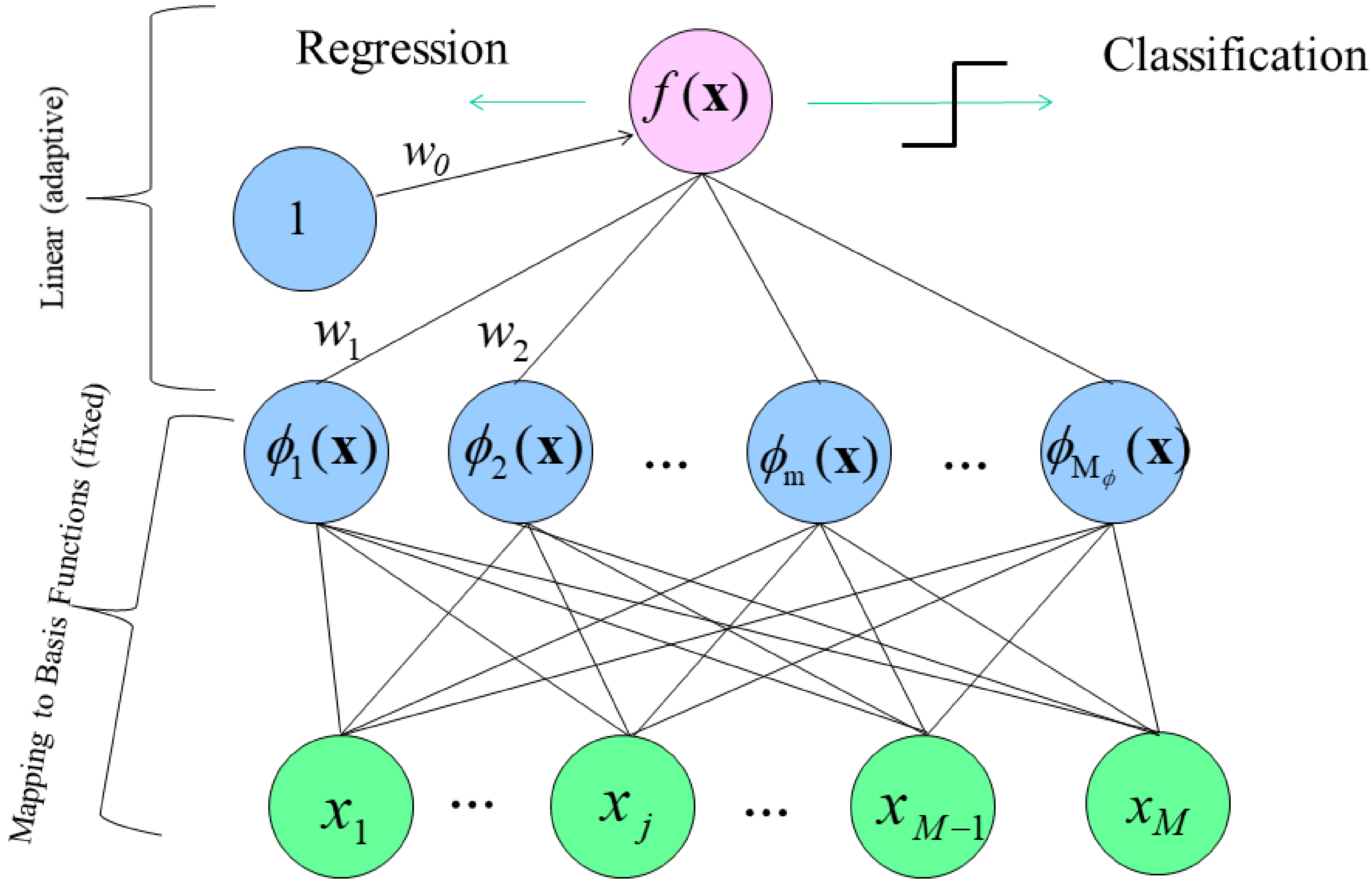
- Basis functions $\{\phi_m(\mathbf{x})\}_{0=1}^{M_\phi}$

- 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

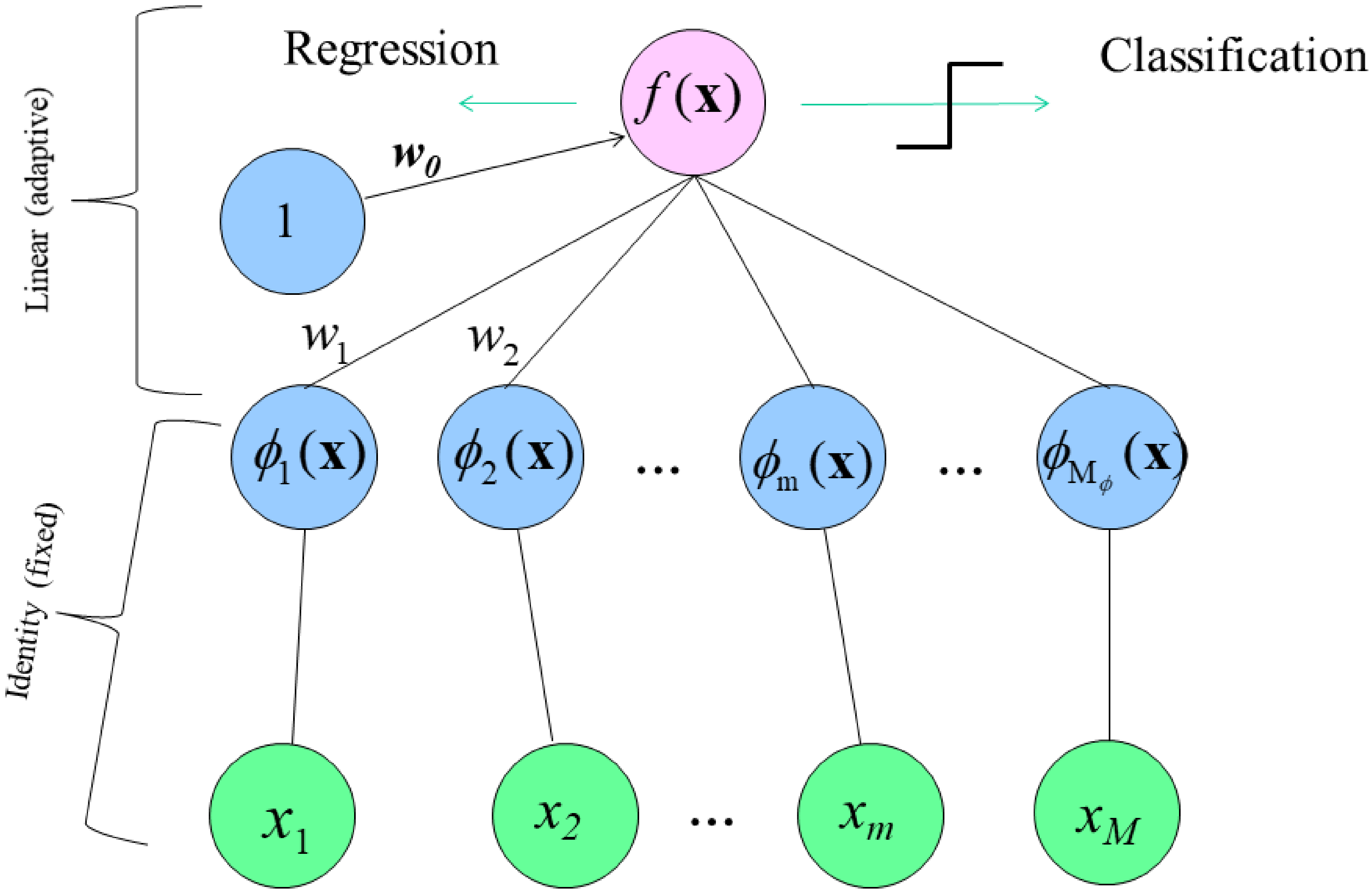


Linear Model Written as Basis Functions

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

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

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

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

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:

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

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

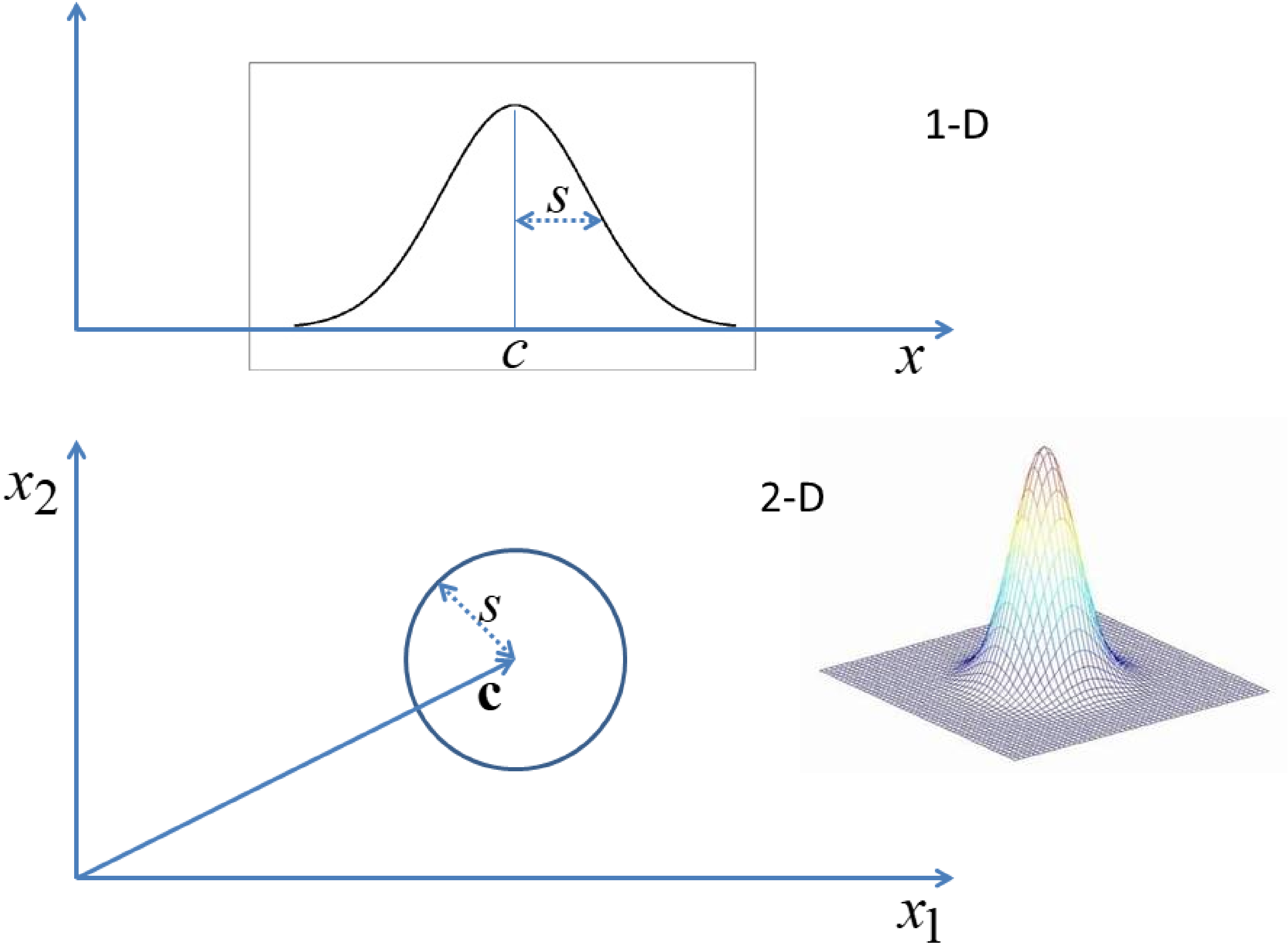
- If we include too few basis functions or unsuitable basis functions, we might not be able to model the true dependency

- 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)

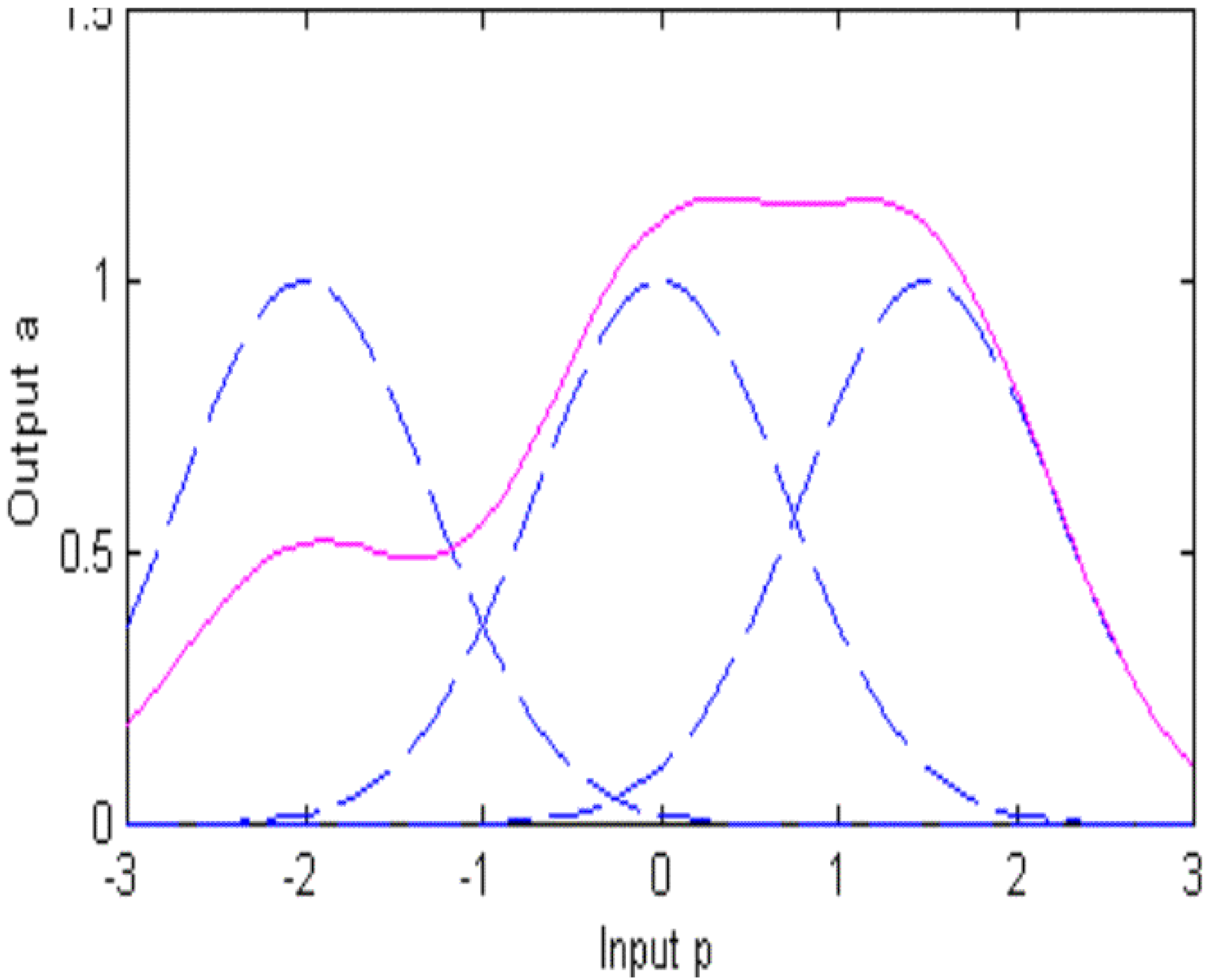
Radial Basis Function (RBF)

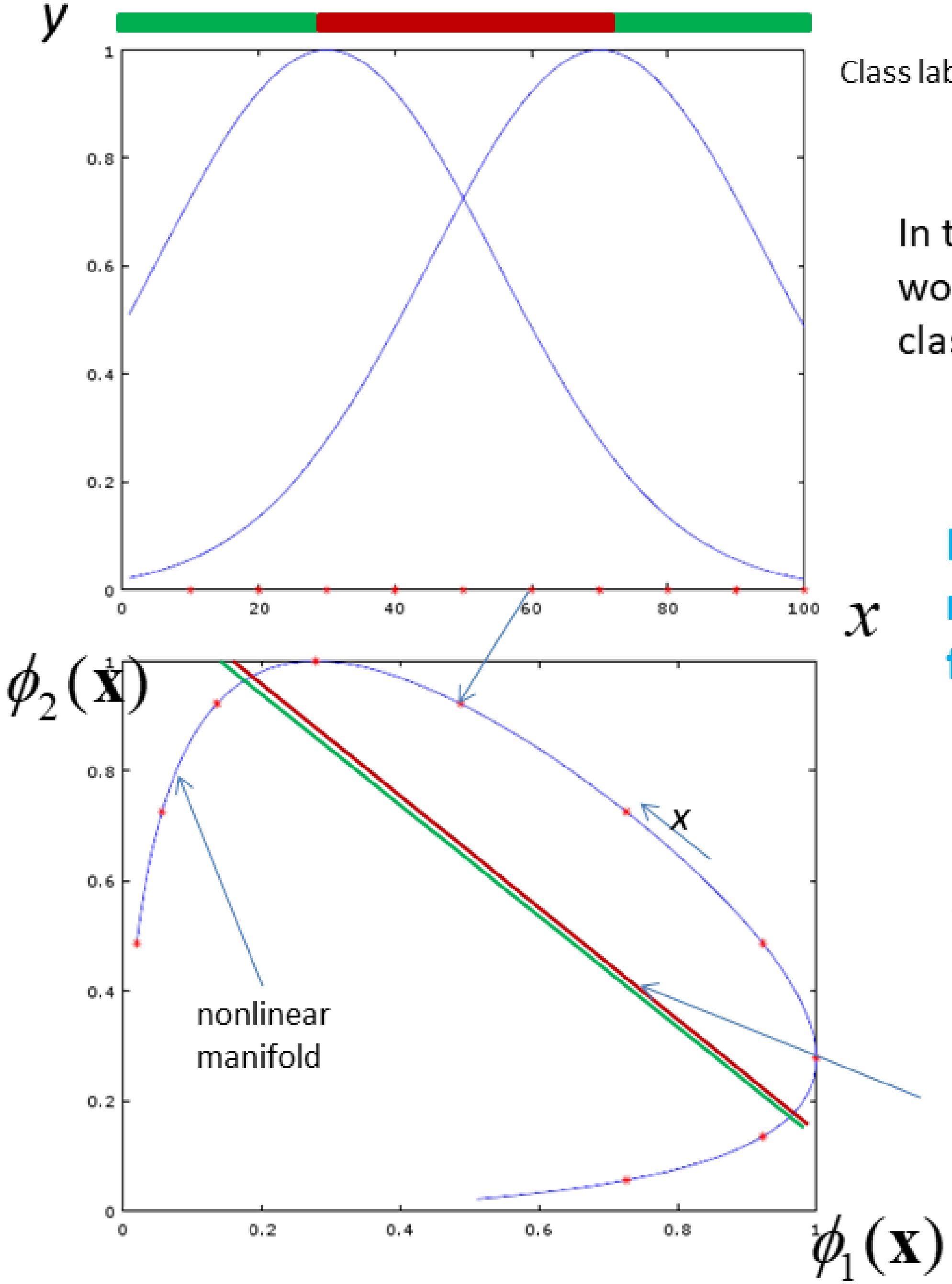
- We already have learned about polynomial basis functions
- Another class are radial basis functions (RBF). Typical representatives are Gaussian basis functions

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


1-D
s
c
x
2-D
x2
s
c
x1

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





Class labels (green, red, green)

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

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

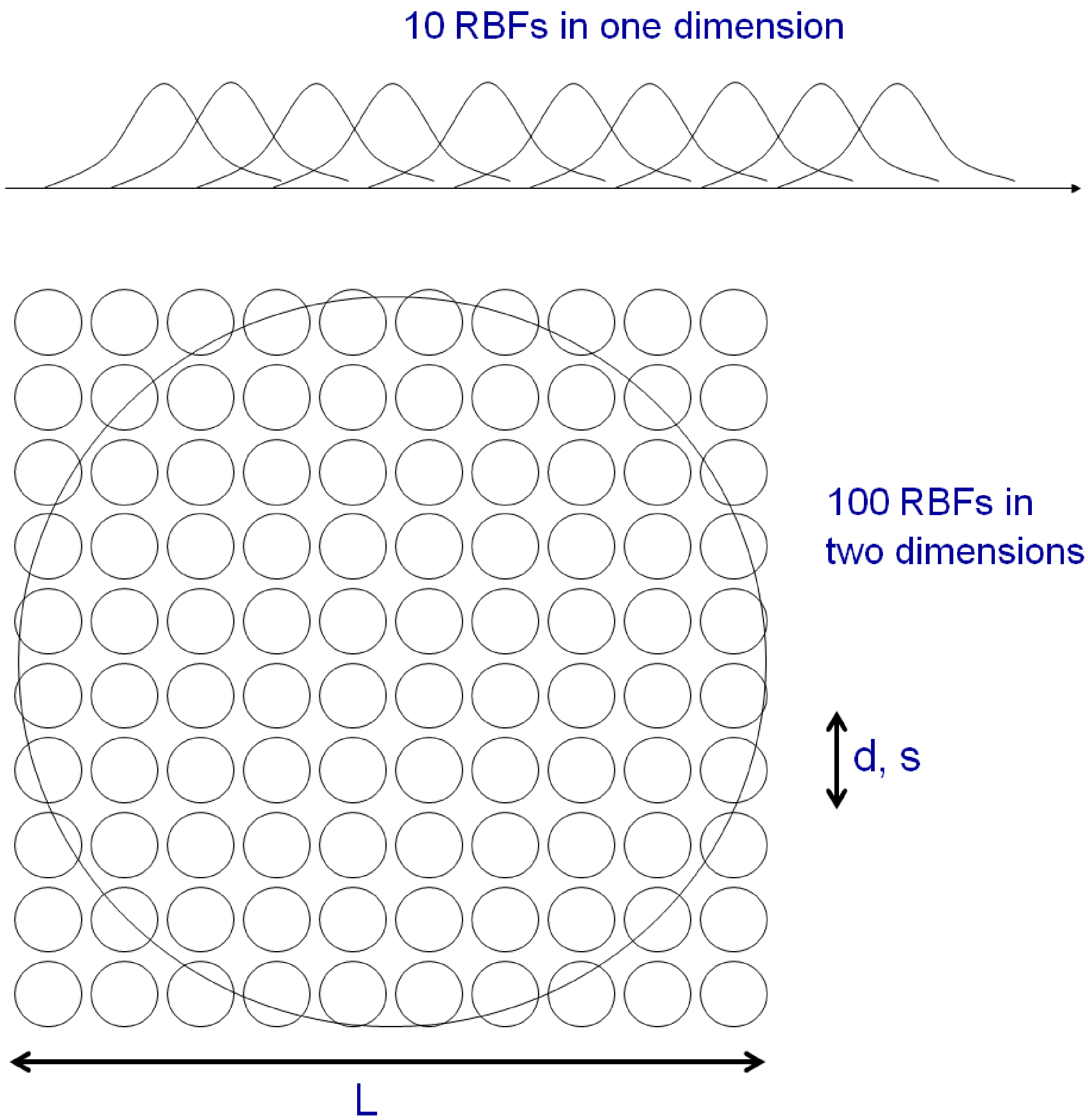
In basis function space, classes can linearly be separated!

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

separating hyperplane

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); $(x^2, y^2, z^2, xy, xz, yz), \ldots$
- *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

Example: Ad Placements

- The decision of which ad to place for which user in which context is defined as a classification or regression problem in a high-dimensional feature space
- Here the features are often handcrafted and new features are continuously added and removed, optimizing the prediction in the long run
- Speed in training and recall and a small memory trace are important criteria

Just Function Fitting? Informal Complexity Analysis

Best Approximation Error

- We are concerned with the properties of the **true function** $f(\mathbf{x})$, a **model class** $\{f_{\mathbf{w}}(\mathbf{x})\}_{\mathbf{w}}$, and the **best approximating function** out off this class $f_{\mathbf{w}_{opt}}(\mathbf{x})$
- **Average Approximation Error:** What is the approximation error between the true function and the best model out of the model class, **averaged over a unit ball**

$$\epsilon_{AAE}(f, f_{\mathbf{w}_{opt}}) = \int_B (f(\mathbf{x}) - f_{\mathbf{w}_{opt}}(\mathbf{x}))^2 d\mathbf{x}$$

- **Expected Approximation Error**: What is the smallest approximation error between the true function and the best model out of the model class, **weighted by the input data distribution** $P(\mathbf{x})$; important special case: data is on a manifold

$$\epsilon_{EAE}(f, f_{\mathbf{w}_{opt}}) = \int (f(\mathbf{x}) - f_{\mathbf{w}_{opt}}(\mathbf{x}))^2 P(\mathbf{x}) d\mathbf{x}$$

ϵ_{AAE} is (up to a constant) the same as ϵ_{AAE}, if the input data is distributed uniformly in the unit ball; the ϵ_{EAE} is related to the square of the *Bias* in statistics

Data Complexity and Computational Complexity

- **Statistics and Generalization (S&G):** How many training data points N are requires such that for the parameters $\hat{\mathbf{w}}$ that minimize the cost function, we have that $\epsilon_{EAE}(f_{\hat{\mathbf{w}}}, f_{\mathbf{w}_{opt}})$ is small; the last term is related to the Model *Variance* in statistics; in general, we assume that N must be on the order of the number of free parameters

- **Computational Complexity (CC):** What is the computational cost for training (finding $\hat{\mathbf{w}}$)? Are there problems with local optima?

Analysis of Dimensionality

- Consider input space dimension M. We consider a ball of a unit ball of diameter 1 (w.l.o.g.)

- Let us consider that d is a minimum length scale of interest, e.g., $d = 0.15$, thus the true $f(\mathbf{x})$ might change significantly (e.g., from 1 to -1) if I move a distance d in some direction

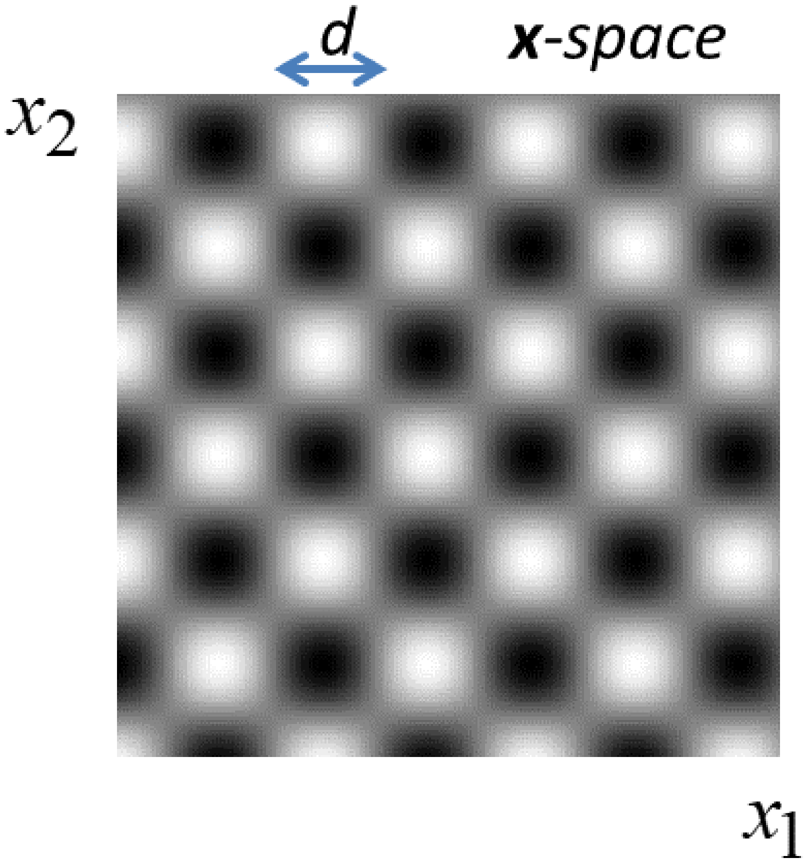
- Then the bandwidth $\nu = 1/(2d)$ would be an upper frequency of interest; example: with $d = 0.15$, $\nu = 3.3$

- Thus ν is a complexity measure: the larger ν, the more complex the function; in the following figure $(1/(2\nu))^M$ corresponds to the number of maxima and minima of the underlying true function

- ϵ_{AAE}: If $f_{\mathbf{w}}(\mathbf{x})$ represents fixed basis functions, where only the weights are learned, then M_ϕ (the number of basis functions) should be on the order of (if one cannot exploit some form of regularities)

$$\mathcal{O}((2\nu)^M)$$

2-D Checker Board Function



M=2

d = 0.15

nu = 3.3

Thus

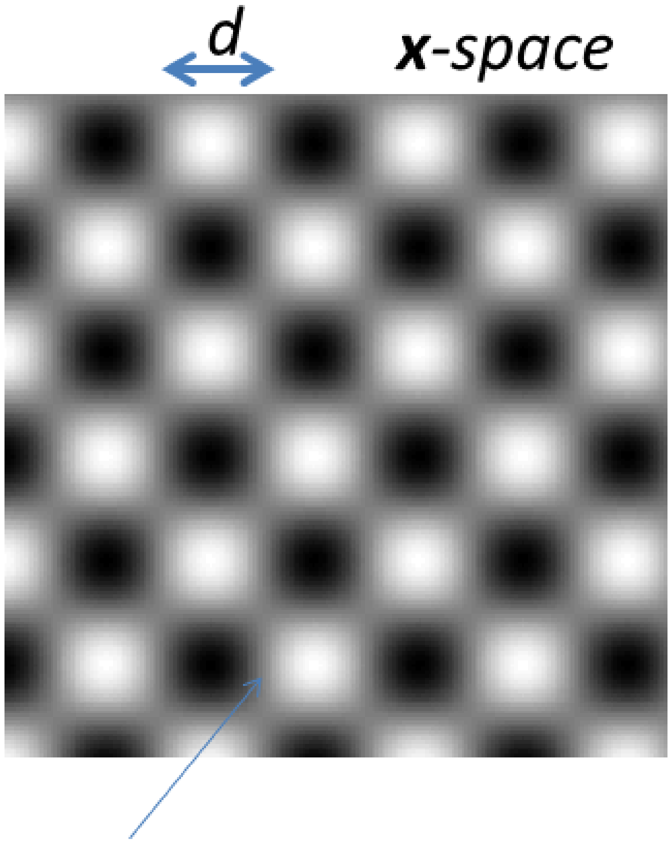
N >>(2x3.3)^2 = 62

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

Case I: Curse of Dimensionality

- Here M is large, and ν is large
- ϵ_{AAE}: M_ϕ on the order of $\mathcal{O}((2\nu)^M)$ would indicate that we need exponentially many basis functions
- This is the famous "Curse of Dimensionality"
- (The curse of dimensionality can also be related to the fact that you need $\mathcal{O}((2\nu)^M)$ data points for a nearest neighborhood approach to make sense or the fact that randomly generated data points tend to be equidistant in high dimensions)
- S&G: To be able to learn all the parameters, one would need the same order of data points $N \approx M_\phi$
- CC: The computational cost is on the order of M_ϕ^3

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



2-D slice through a 20-D input space

M=20
d = 0.15
nu = 3.3

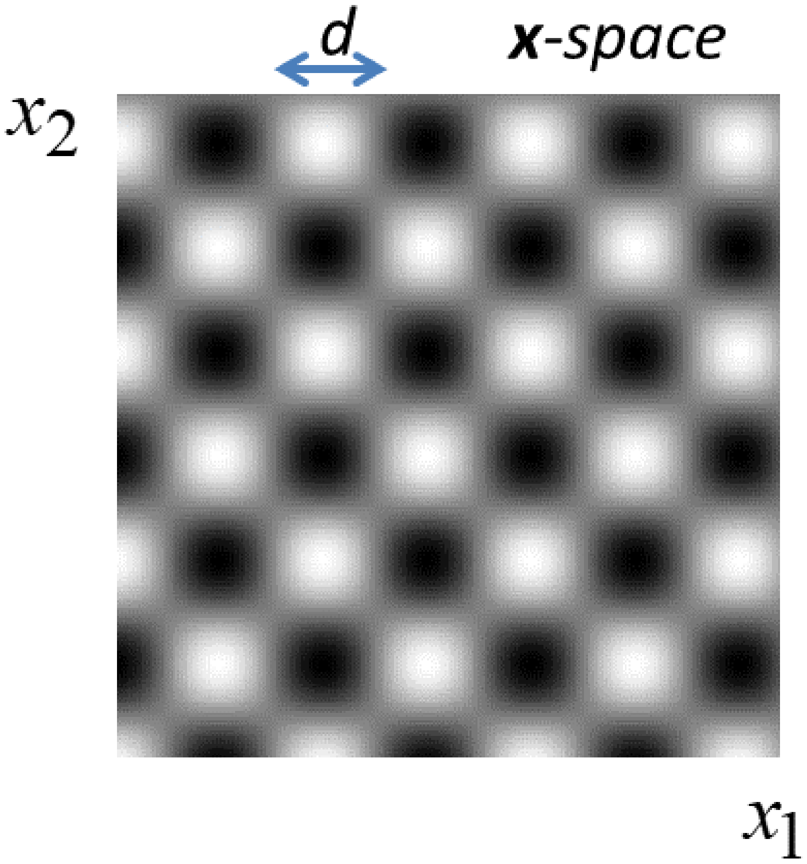
Thus
N >>$(2x3.3)^{20}$ =
$2.5x10^{16}$

The required number of basis function is huge

Case II: Blessing of Dimensionality

- M is small but ν is large
- ϵ_{AAE}: With M_ϕ on the order of $\mathcal{O}((2\nu)^M)$ would indicate that we many "many" (but not a huge number of) basis functions (recall, that M is small)
- This is what I would call the "Blessing of Dimensionality", since a nonlinear classification problem can be solved by a transformation into a high-dimensional space where it becomes linearly separable
- S&G: The number of required training data is $N \approx M_\phi$
- (Consider also that we only need on the order than $(2\nu)^M$ data points (with M small) for a nearest neighborhood approach to make sense)
- CC: The computational cost is on the order of M_ϕ^3

2-D Checker Board Function



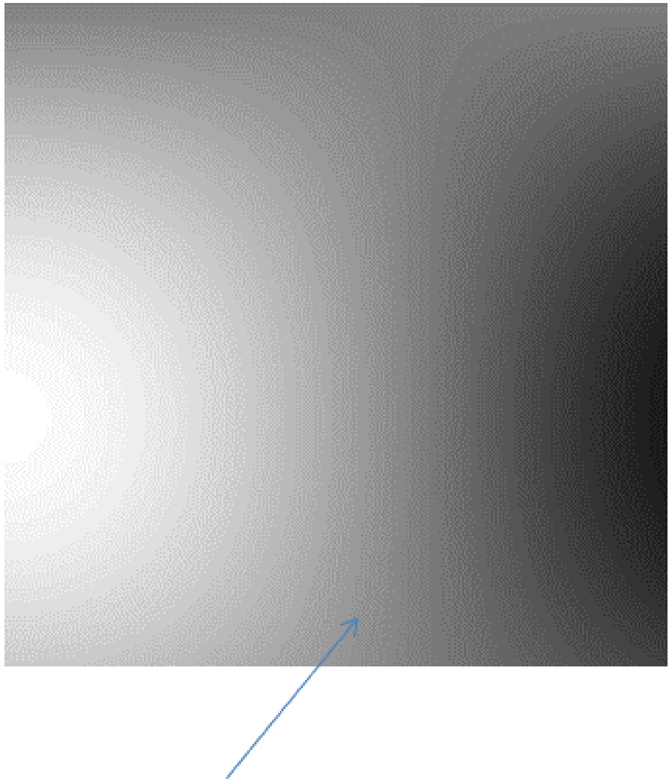
M=2
d = 0.15
nu = 3.3

Thus
N >>(2x3.3)^2 *= 62*

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

Case III: Smooth

- Here, M is large and ν is small; this is the case when the system has a voting behavior: each input itself has a (small) contribution to the output
- ϵ_{AAE}: Then $M_\phi \approx (2\nu)^M$ and $N \approx M_\phi$ might not be too large
- A special case would be a linear function, where $M_\phi = M + 1$
- (Interestingly, a nearest neighborhood approach would still have problems with the high M; this shows the advantage of training discriminately! Neighborhood methods sometimes learn the distance metrics to approach this issue (learning of Mahalanobis distance)
- Trivially, the fourth situation (Case IV: simple), small M and small ν is quite easy to model



2-D slice through a 20-D input space

M=20
d = 0.8
nu = 1.25

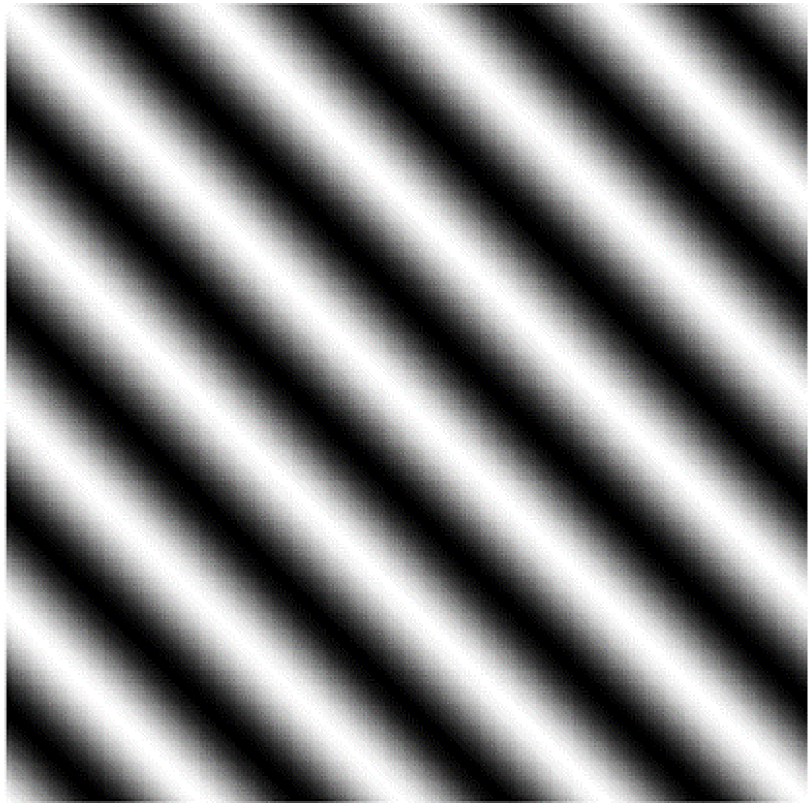
Thus
$N \gg (2\times0.62)^{20} = 87$

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

Case Ia: Sparse Basis: No Curse of Dimensionality with a Neural Network

- ϵ_{AAE}: We have Case I (curse), **but only H basis functions have nonzero weights**; e.g., high complexity might only be in some regions in input space. The number of hidden units is otherwise independent of M!
- Neural networks with H hidden units can adaptively find the sparse basis (with back-propagation)
- But note that to learn the correct basis functions, I need the training data targets $\mathbf{y}$
- S&G: For a neural network, the number of required data points is on the order of $\mathcal{O}(H \times M)$ (essentially the parameters in the V matrix (see following lecture))
- CC: Due to the nonlinear characteristics of the optimization problem, computational cost can be high (backpropaagtion) and local optima might become a problem

x-*space*

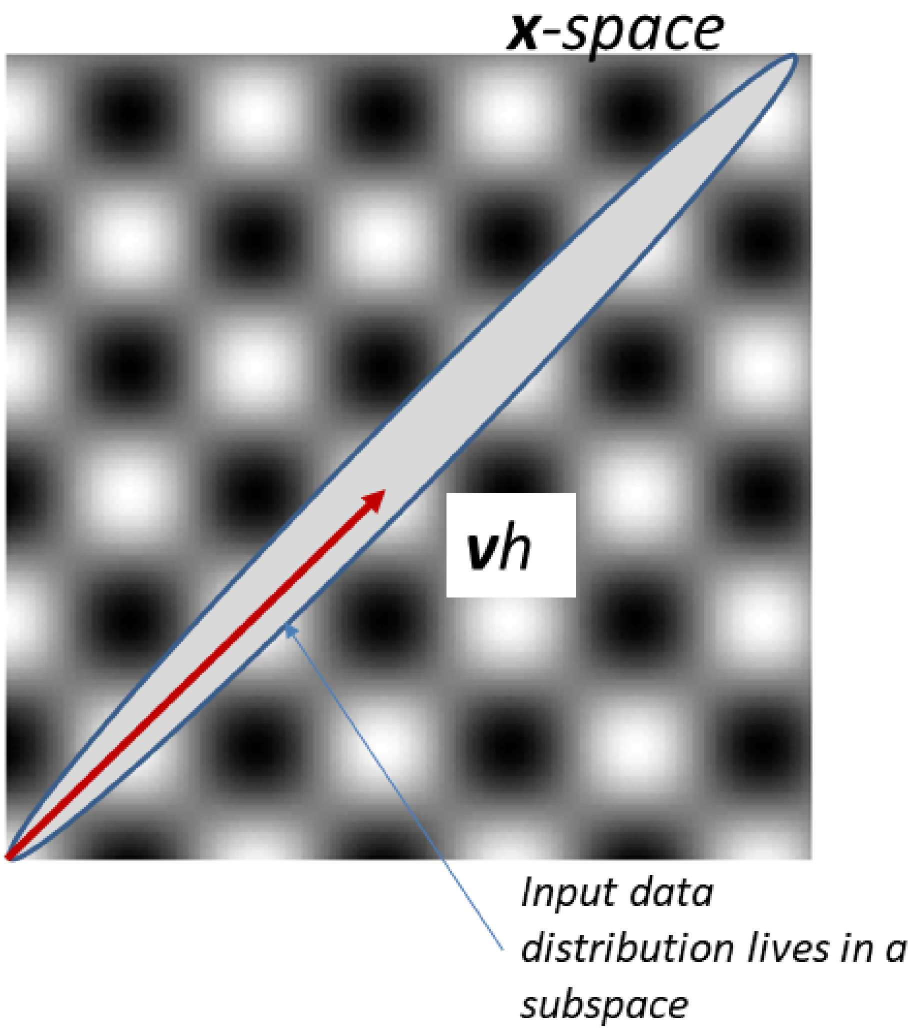


H=16 hidden units in a neural network might be sufficient

Although the input space might be high dimensional, complexity is limited

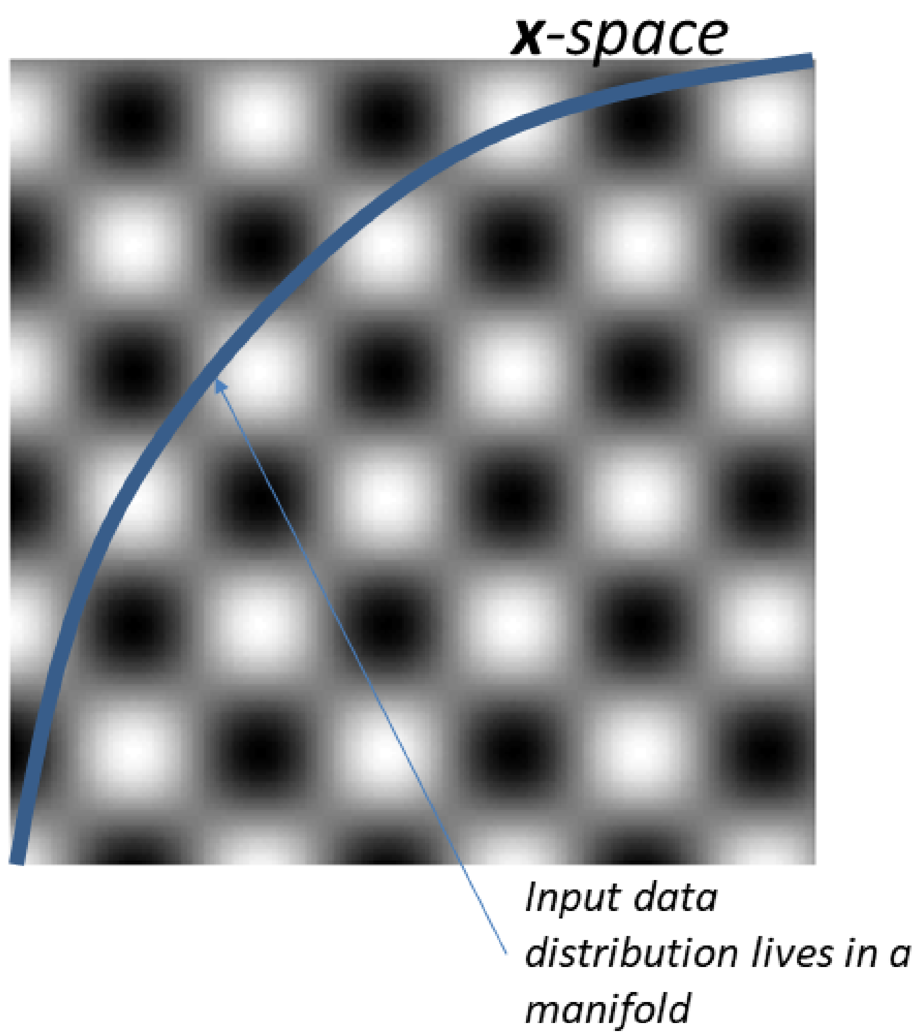
Case Ib: Manifold

- ϵ_{AAE}: So far we did not assume any particular input data distribution: $\mathbf{x}$ might be uniformly distributed
- Sometimes $\mathbf{x}$ is restricted to a subspace (in the nonlinear case: manifold) (see lecture on manifolds)
- ϵ_{EAE}/S&G: If the data is on an M_h-dimensional manifold, one only need $\mathcal{O}((2\nu)^{M_h})$ basis functions and data points and the performance for test data on the manifold might be excellent
- ϵ_{AAE}: but if we consider test data outside of the manifold (as in a unit sphere) performance degrades: ϵ_{AAE} could be large!



- Although the input space might me high dimensional, the data lives in a subspace
- The dimension of the subspace is M_h, here 1
- I can find the dimensions that matter, in a preprocessing step without using ***y***

In case that the columns of V are orthonormal, there is a simple geometric interpretation

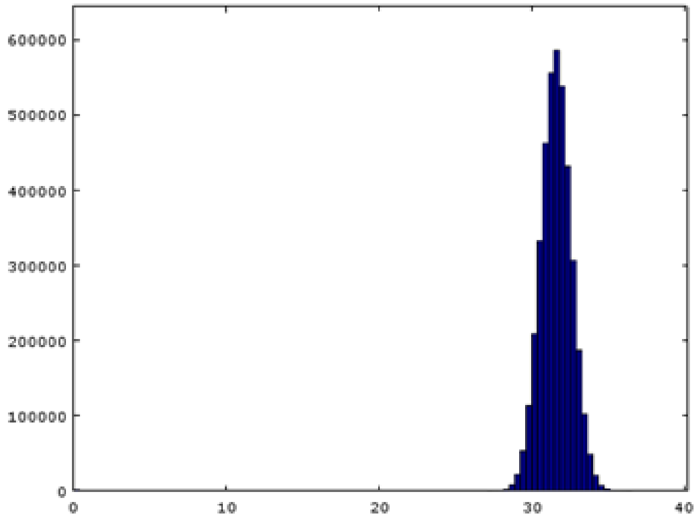


More general, the data lives in a manifold

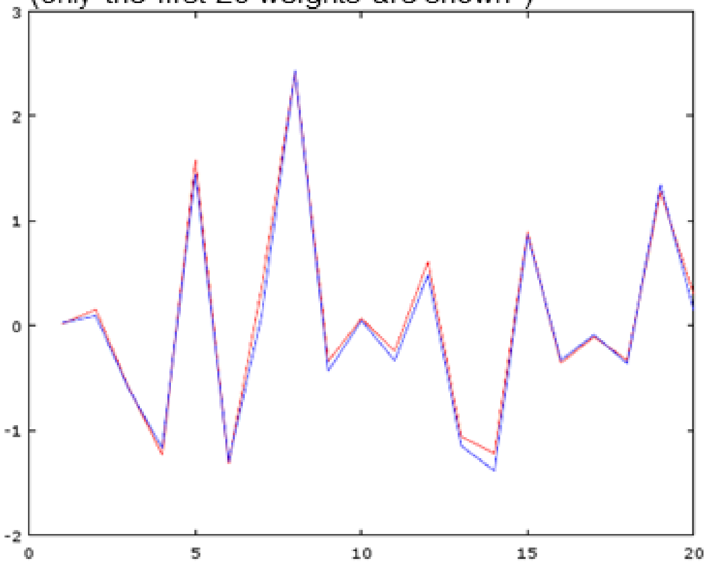
Illustration

- 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, $\nu \approx 1$:
- A linear regression model with $N = 1000$ training data points gives excellent results

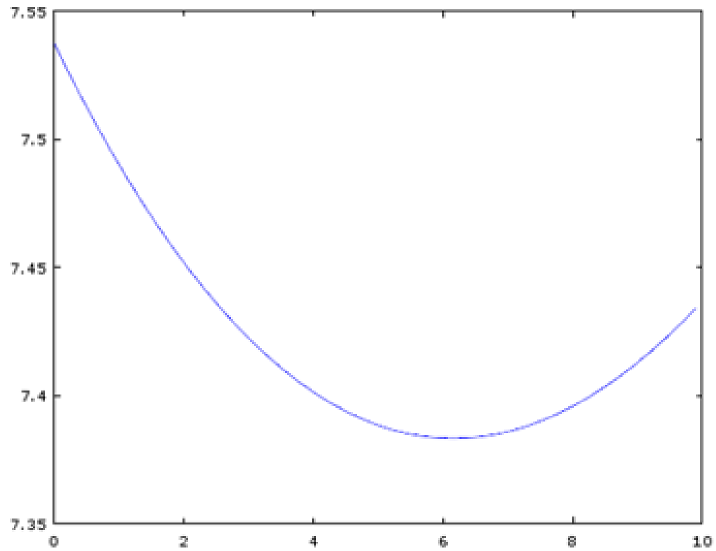
histogram: distance of inputs



True weights (blue) and estimated weights (red)
(only the first 20 weights are shown)



Mean test error versus regularization



Functions with Unlimited Bandwidth

- So far, we assumed that the true function $f(\mathbf{x})$ had a limited bandwidth ν; we concluded that if the model class would be able to approximate such a function, then ϵ_{AAE} would be sufficiently small; of course, in general ν is unknown
- Another approach is to assume that $f(\mathbf{x})$ has unlimited bandwidth, and one analyses how the approximation errors ϵ_{AAE} scales with the number of model parameters
- In the previous equations, we substitute $\nu \rightarrow 1/\epsilon_{AAE}$
- Then the number of required basis functions M_ϕ is on the order of

$$\mathcal{O}\left(\frac{1}{\epsilon_{AAE}}^{M}\right)$$

or, equivalently, $\mathcal{O}\left({\epsilon_{AAE}}^{-M}\right)$ (this expression is often found in literature)

Conclusions

- Basis functions perform a nonlinear transformation from input space to basis function space
- For a good model fit, one needs $\mathcal{O}((2\nu)^M)$ fixed basis functions and training data points; thus either M should be small (Case II (blessing)) or $\nu \approx 1$ (Case III) (smooth) or (Case IV) (simple)
- With M and ν large, we experience the curse of dimensionality; if, actually, data is distributed in a subspace (or nonlinear manifold) with dimension M_h, then one only needs $\mathcal{O}((2\nu)^{M_h})$ fixed basis functions and training data points
- Neural networks are effective when the basis is sparse (Ia (sparse basis))
- The next table evaluates distance-based methods (like nearest neighbor methods), models with fixed basis functions, neural networks, and deep neural networks

Case	Neighb. Methods	fixed BF	Neural Nets	Deep NNs	Kernels
I (curse)	-	-	-	-	-
II (blessing)	+	+	+	+	+
III (smooth)	-	+	+	+	+
IV (simple)	+	+	+	+	+
Ia (sparse basis)	-	-	+	+	-
Ib (manifold)	+ (dr)	+ (dr)	+	+	+
Ic (compos.)	-	-	-	+	-

(dr) stands for dimensionality reduction by a preprocessing step;

Case Ic are compositional functions, introduced in the lecture on deep neural networks

Kernels are introduced in a later lecture