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

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>+1</td>
</tr>
<tr>
<td>1 0</td>
<td>−1</td>
</tr>
<tr>
<td>0 1</td>
<td>−1</td>
</tr>
<tr>
<td>1 1</td>
<td>+1</td>
</tr>
</tbody>
</table>
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 $x_1x_2$ as additional input
- The interaction term $x_1x_2$ couples two inputs non-linearly
With a Third Input \( z_3 = x_1 x_2 \) the XOR Becomes Linearly Separable

\[
\phi_3 = x_1 x_2
\]

\[
\phi_2 = x_2
\]

\[
\phi_1 = x_1
\]

\[
f(x) = 1 - 2x_1 - 2x_2 + 4x_1 x_2 = \phi_0(x) - 2\phi_1(x) - 2\phi_2(x) + 3\phi_3(x)
\]

with \( \phi_0(x) = 1, \phi_1(x) = x_1, \phi_2(x) = x_2, \phi_3(x) = x_1 x_2 \)
\[ f(x) = 1 - 2x_1 - 2x_2 + 4x_1x_2 \]
Separating Planes

\[ x_1 \]

\[ x_2 \]
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 \( w = (0, 1, 0, -0.3) \)
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 \( \{\phi_m(x)\}_{0=1}^{M_\phi} \)

• In the example:

\[
\phi_0(x) = 1 \quad \phi_1(x) = x_1 \quad \phi_5(x) = x_1x_3 \quad ...
\]

• Independent of the choice of basis functions, the regression parameters are calculated using the well-known equations for linear regression.
Network of Basis Functions

Regression

$\phi_1(x) \quad \phi_2(x) \quad \ldots \quad \phi_m(x) \quad \ldots \quad \phi_M(x)$

Mapping to Basis Functions (fixed)

$x_1 \quad \ldots \quad x_j \quad \ldots \quad x_{M-1} \quad x_M$

Linear (adaptive)

$w_0 \quad w_1 \quad w_2$

Classification
Linear Model Written as Basis Functions

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

\[ \phi_0(x) = 1, \quad \phi_1(x) = x_1, \quad \ldots \quad \phi_M(x) = x_M \]
Network of Linear Basis Functions

Regression

Classification

Identity (fixed)

Linear (adaptive)
Review: Penalized LS for Linear Regression

- Multiple Linear Regression:

\[ f_w(x) = w_0 + \sum_{j=1}^{M} w_j x_j = x^T w \]

- Regularized cost function

\[ \text{cost}^{pen}(w) = \sum_{i=1}^{N} (y_i - f_w(x_i))^2 + \lambda \sum_{j=0}^{M} w_j^2 \]

- The penalized LS-Solution gives

\[ \hat{w}_{pen} = \left( X^T X + \lambda I \right)^{-1} X^T y \quad \text{with} \quad X = \begin{pmatrix} x_{1,0} & \cdots & x_{1,M} \\ \vdots & \ddots & \vdots \\ x_{N,0} & \cdots & x_{N,M} \end{pmatrix} \]
Regression with Basis Functions

- Model with basis functions:

\[ f_w(x) = w_0 + \sum_{m=1}^{M_\phi} w_m \phi_m(x) \]

- Regularized cost function with weights as free parameters

\[
\text{cost}^{\text{pen}}(w) = \sum_{i=1}^{N} \left( y_i - \sum_{m=0}^{M_\phi} w_m \phi_m(x_i) \right)^2 + \lambda \sum_{m=0}^{M_\phi} w_m^2
\]

- The penalized least-squares solution

\[
\hat{w}_{\text{pen}} = \left( \Phi^T \Phi + \lambda I \right)^{-1} \Phi^T y
\]
with

\[
\Phi = \begin{pmatrix}
1 & \phi_1(x_1) & \ldots & \phi_{M\phi}(x_1) \\
\vdots & \vdots & \ddots & \vdots \\
1 & \phi_1(x_N) & \ldots & \phi_{M\phi}(x_N)
\end{pmatrix}
\]
Nonlinear Models for Regression and Classification

- Regression:

\[ f_w(x) = w_0 + \sum_{m=1}^{M_\phi} w_m \phi_m(x) \]

As discussed, the weights can be calculated via penalized LS

- Classification:

\[ \hat{y} = \text{sign}(f_w(x)) = \text{sign} \left( w_0 + \sum_{m=1}^{M_\phi} w_m \phi_m(x) \right) \]

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 \( 1, \phi_1(x_i), \phi_2(x_i), \ldots \)
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 $f(x)$ can be modelled by at least one of the functions $f_w(x)$ 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.

\[ \phi_j(x) = \exp \left( -\frac{1}{2s^2} \| x - c_j \|^2 \right) \]
Three RBFs (blue) form $f(x)$ (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 ($x^2$); ($x^2, y^2, xy$); ($x^2, y^2, z^2, xy, xz, yz$), ...

• 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) $f(x)$?
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 handwritten 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
Just Function Fitting? Informal Complexity Analysis
Analysis of Dimensionality

- Consider input space dimension $M$. All data points are normalized to zero mean and lie in a 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 $y$ 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; with $d = 0.15$, $\nu = 3.3$

- Thus $\nu$ is a complexity measure: the larger $\nu$, the more complex the function; in the following figure $(1/(2\nu))^M$ 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

  $$N \gg (2\nu)^M$$

  data points (if one cannot exploit some form of regularities); one approximately needs the same number of basis functions $M_\phi \approx N$
2-D Checker Board Function

\[ M = 2 \]
\[ d = 0.15 \]
\[ nu = 3.3 \]

Thus
\[ N \gg (2 \times 3.3)^2 = 62 \]

Here \( M = 2 \) is quite small and with 62 RBF basis functions we might get a good fit
I. Large $M$ and large $\nu$ requires huge $M_\phi$: Curse of Dimensionality

- Here $M$ is large, and $\nu$ is large

- With $N >> (2\nu)^M$, $M_\phi \approx N$ would indicate that we need exponentially many training data points and exponentially many basis functions to learn about the functions

- This is the famous “Curse of Dimensionality”

- (The curse of dimensionality can also be related to the fact that you need more than $(2\nu)^M$ 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)
20-D Checker Board Function: “Curse of Dimensionality”

\[ M = 20 \]
\[ d = 0.15 \]
\[ nu = 3.3 \]

Thus
\[ N \gg (2 \times 3.3)^{20} = 2.5 \times 10^{16} \]

The required number of basis function is huge
II. Small $M$ and large $\nu$ requires large $M_\phi$: Blessing of Dimensionality

- $M$ is small but $\nu$ is large
- With a sufficient amount of training data, $N \gg (2\nu)^M$, the data can explore the relevant input space
- If I map the inputs space to a basis function space of dimension $M_\phi$, the data points still lie on an $M$-dimensional manifold in the basis function space. So the estimate $N \gg (2\nu)^M$ might still be valid
- With $M_\phi \approx N$ 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”, since a nonlinear classification problem can be solved by a transformation into a high-dimensional space where it becomes linearly separable.
• 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

• (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)
2-D Checker Board Function

\[ M = 2 \]
\[ d = 0.15 \]
\[ nu = 3.3 \]

Thus
\[ N \gg (2 \times 3.3)^2 = 62 \]

Here \( M = 2 \) is quite small and with 62 RBF basis functions we might get a good fit.
III. Large $M$ and small $\nu$ requires small $N$ and small $M_\phi$: No Curse of Dimensionality

- Here, $M$ is large and $\nu$ is small
- Then $N >> (2\nu)^M$ can still be acceptable
- (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 (IV), small $M$ and small $\mu$ is quite easy to model, although neural networks sometimes have a hard time
$M=20$
$d = 0.8$
$nu = 1.25$

Thus
$N >\left(2\times 0.62\right)^{20} = 87$

Here $M=20$ is medium size and with 87 RBF basis functions we might get a good fit; nu is quite small
Ia: Large $M$ and Large $\nu$, but only in a projection in a subspace: No Curse of Dimensionality with a Neural Network

- In case, $M$ and large $\nu$, but the function $f(x)$ only depends on some projected subspace $z = V^T x$ (encoder) where $z \in \mathbb{R}^{M_z}$ is low dimensional

- Then with the encoder $z = W^T x$, which gives

$$f_w(x) = w_0 + \sum_{m=1}^{M_\phi} w_m \phi_m(V^T x)$$

- Thus if we have a smart algorithm, which finds us $V$ the complexity is $(2\nu)^{M_z}$ where $M_z >> M$

- This is the key to the success of neural networks

- Note that a naive basis function approach would need $(2\nu)^M$ basis functions, which can be prohibitive!

- This can be generalized to a nonlinear encoder $z \leftarrow x$
$M = 20$
$M_z = 1$

Thus, we can choose $N > 8$

Although the input space might mean high dimensional, only a few dimensions matter
In Ia we did not assume any particular input data distribution: \( x \) might be uniformly distributed.

In Ia a neural network solution would require information on the targets’s \( y \).

Here we assume that the data is distributed in a subspace and that this subspace can be found by a preprocessing step, without looking at the targets \( y \): The \( V \) ins \( z = V^T x \) can be found without looking at the targets!

Principal Component Analysis and Independent Component Analysis are two popular methods for finding linear subspaces.

Autoencoders can be used to find (nonlinear) manifolds.

Sounds a bit paradox: First we do a dimension reduction from \( M \) to \( M_z \) and then a dimension enlargement from \( M_z \) to \( M_\phi \).
• Example: We have seen before that feature transformations (basis functions) can generate data that live in a manifold

• Example: Consider $K$ images of faces where the images have $256 \times 256 = 2^{16}$ pixels; the linear combinations of those faces live in a $K$-dimensional subspace
Thus, we can choose $N > 8$

Although the input space might be high dimensional, the data lives in a subspace I can find the dimensions that matter, in a preprocessing step without looking at $\mathbf{y}$
Illustration

- With $M = 500$ input dimensions, we generated $N = 1000$ random data points $\{x_i\}_{i=1}^{1000}$

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

Mean test error versus regularization.
Conclusions

- Basis functions perform a nonlinear transformation from input space to basis function space.

- For a good model fit, one needs $N \gg (2\mu)^M$, $M\phi \approx N$, thus either $M$ should be small (II.) or $\mu \approx 1$ (III.)

- With $M$ and $\nu$ large, we experience the curse of dimensionality; if, actually, data is distributed in a subspace (or nonlinear manifold), one can attempt to find that one in a preprocessing step (PCA, autoencoder, ...) (Ib); if the function is only complex in a projected subspace (or nonlinear manifold), neural networks might be effective (Ia).

- The next table evaluates distance-based methods (like nearest neighbor methods), models with fixed basis functions, and neural networks.
<table>
<thead>
<tr>
<th></th>
<th>Distance Based</th>
<th>fixed BF</th>
<th>Neural Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>II</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>III</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>IV</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>la</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>lb</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>