An Introduction to Function Approximation for Machine Learning

Volker Tresp
Winter 2023-2024
Problem Setting

• In an actual application, the data scientist needs to decide which model to use (linear Perceptron, fixed basis functions, neural networks, kernels, . . . ?)

• How well is one doing in solving the actual problem with the data actually available; in the lecture on model selection, we learn about some empirical methods for analysing some of these issues

• But what can theory tell us about these issues? Why are, e.g., deep neural networks so successful?

• In this lecture we will be a bit informal since formal treatments require advanced mathematical frameworks and are beyond the scope of this lecture
Target Function Class

• Let $\mathcal{F}$ be the set of target functions

• What characterizes the functions that mother nature generates for a particular problem, e.g., image classification, and how can I characterize them?

• In a theoretical analysis one characterizes this class in some way, hopefully limiting the class to the one actually occurring in practice (neither larger nor smaller)

• Often one defines the target class by some degree of smoothness of the functions; another target class of functions are composable functions (see lecture on deep learning)
Target Function Class (cont’d)

- The modern view is that the target function class assumes a tiny space in the space of all functions and e.g., deep learning models, work so well because they match this class reasonably well.

- Some claim that machine learning is impossible if the target function class is not restricted (no-free-lunch theorem).
Model Function Class

• What characterizes the model function class $\mathcal{M}$

• To simplify matter (mostly for notational simplicity), we assume a model function class can be described as

$$\mathcal{M} = \{f_w(\cdot)\}w$$

i.e. functions which only vary in their parameters (but this is not essential)
Distance between Functions

- Consider the true function $f(\cdot)$ and an model $f_w(\cdot)$. We define

$$
\|f - f_w(\cdot)\|_B^2 = \frac{1}{V_B} \int_B (f(x) - f_w(x))^2 dx
$$

Here $V_B$ is the volume of the unit ball $B$ in $M$ dimensions

- This is simply the average squared Euclidean distance, applied to two functions
Weighted Distance

- The average squared Euclidean distance between the two functions is (weighted by $P(x)$)

$$\|f - f_w(\cdot)\|_{P(x)}^2 = \int (f(x) - f_w(x))^2 P(x) dx$$

- $P(x)$ is the probability distribution of the input data

- In some cases the input data only occupy a small subspace (manifold) of the unit ball; some learning approaches are able to explore this
Distance between Functions

- We define $\epsilon_B$ to be the minimum Euclidean distance for the "most difficult" function out of the function class

$$\epsilon_B = \min_{w} \max_{f} \| f - f_w \|_B$$

$$\epsilon_{P(x)} = \min_{w} \max_{f} \| f - f_w \|_{P(x)}$$

$f \in \mathcal{F}$, $f_w \in \mathcal{M}$
Statistical Machine Learning

- Statistical machine learning analyses the distance between the expected distance between a model function, where the parameters were estimated based on some training data, and a given $f$.

- This is not the issue in approximation theory, and will be discussed in a later lecture.
Analysis of Dimensionality

- Consider input space dimension $M$
- If in one dimensions, we need $M^{(one\text{-}dim)}_{\phi}$ RBFs (e.g., $M^{(one\text{-}dim)}_{\phi} = 10$), and we want to maintain the same complexity in higher dimensions, then we need

$$M_{\phi} = \left( M^{(one\text{-}dim)}_{\phi} \right)^M$$

RBFs in $M$ dimensions
10 RBFs in one dimension

100 RBFs in two dimensions

$d, s$

$L$
Analysis of Dimensionality (cont’d)

• We get

\[ M_{\phi}^{(\text{one-dim})} = \mathcal{O} \left( \frac{1}{\epsilon B} \right) \frac{1}{1/m} \]

• Here, \( m \) is a characterization of the smoothness of the target class: \( m \) can be the set of all functions with continuous partial derivatives of orders up to \( m \) (derivatives of higher order can be discontinuous)

• This result can, e.g., be found in: “Why and When Can Deep-but Not Shallow-networks Avoid the Curse of Dimensionality: A Review” Tomaso Poggio et al., International Journal of Automation and Computing, 2017, Equation 5.
Analysis of Dimensionality (cont’d)

• We can write this as

\[ M_{\phi}^{(\text{one-dim})} = \mathcal{O}\left(\text{accuracy}^{\text{roughness}}\right) \]

where we have defined \( \text{accuracy} = 1/\epsilon_B \) and \( \text{roughness} = 1/m \)
Analysis of Dimensionality: Main Result

• Overall, the total number of basis function is then

\[ M_\phi = \mathcal{O}(\text{accuracy}^M \times \text{roughness}) \]

• Note, that, for a fixed desired accuracy (e.g., \text{accuracy} = 10), the number of basis functions increases exponentially with \( M \times \text{roughness} \)

• Sometimes it is more instructive to look at the logarithm

\[ \log M_\phi = \mathcal{O}(M \times \text{roughness} \times \log(\text{accuracy})) \]
Case I: Curse of Dimensionality

- $\mathcal{F}$: dimensionality $M$ is large, and roughness is large
- $\mathcal{M}$: Considering that $(M \times \text{roughness})$ is in the exponent, $M_\phi$ is unrealistically large
- This is the famous Bellman’s “Curse of Dimensionality”
20-Dimensional Checker Board Function: "Curse of Dimensionality"

$M$ is high ($M=20$), roughness is large

The required number of basis function is huge

2-D slice through a 20-Dimensional input space
Case II: Blessing of Dimensionality

- $\mathcal{F}$: dimensionality $M$ is small but roughness is large
- In this case $(M \times \text{roughness})$ might be acceptable
- $\mathcal{M}$: This is what I would call the “Blessing of Dimensionality”: a complex nonlinear classification problem (large roughness) can be solved by a transformation of the low-dimensional input space ($M$) into a high-dimensional space ($M_\phi$) where the problem might even become linearly separable
Here $M=2$ (roughness is large) and with less than 100 RBF basis functions we might get a good fit.
Case III: Smooth Target Function in High Dimensions

- $\mathcal{F}$: dimensionality $M$ is large and roughness is small (the target function is smooth)

- A special case would be when the target functions are linear functions; then where $M_\phi = M + 1$; The target function exhibits a voting behavior: each input itself has a (small) contribution to the output

- $\mathcal{F}$: if the target functions can well be approximated by linear functions, the input dimension can be quite high ($M > 10000$)
Case IV (Simple): Smooth Target Function in Low Dimensions

- $F$: dimensionality $M$ is small and roughness is small (the target function is smooth)
- $M$: Only a small number $M_\phi$ of smooth basis functions are required
$\textit{x-space}$

$M$ is large ($M=20$) and roughness is small

Here $M=20$ is medium size and with less than 100 RBF basis functions we might get a good fit

2-D slice through a 20-D input space
Revisiting Case I

• Fortunately, even Case I is not as hopeless as it first appears, since, in reality, classes are more restricted

• Ia: $\mathcal{F}$: The target functions have high-frequency components, but only locally, and a sparse solution is feasible

• Ib: The input data points are restricted to a low-dimensional manifold (reflected in $P(x)$)

• Ic: $\mathcal{F}$: The target functions are composable (discussed in the lecture on deep learning)
Case Ia: Sparse Basis: No Curse of Dimensionality with a Neural Network

- $\mathcal{F}$: both $M$ and roughness are large, so the required $M_\phi$ is large, but only $H << M_\phi$ basis functions have nonzero weights; e.g., high complexity might only be present in a restricted region in input space

- $\mathcal{M}$: With a neural network model, the number of hidden units with nonzero weights (i.e., $H$) might even be independent of $M$!

- As a model class, classical neural networks with $H$ hidden units can adaptively find the “perfect” sparse basis during training (with backpropagation)
**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

- So far we did not assume any particular input data distribution: \( P(x) \) might be a uniform distribution within the unit ball.

- But sometimes \( P(x) \) is restricted to a subspace of small dimension \( M_h << M \); in the nonlinear case, the subspace is called a manifold (data is often on a manifold, when model accuracy is very high (like in OCR).

- \( \mathcal{M} \): we might only need on the order of \( accuracy^{M_h \times roughness} \) (instead of \( accuracy^{M \times roughness} \)) basis functions to cover the relevant region in input space.

- Some model classes, like neural networks / deep neural networks, model data on a low-dimensional manifold quite effectively.

- Other approaches perform a preprocessing step (clustering, PCA, ICA, …) to find the manifold (dimensionality reduction), and then apply any model class suitable for low-dimensional data.
• Although the input space might be high dimensional, the data lives in a subspace
• The dimension of the subspace is $M_h$, here 1

Input data distribution lives in a subspace

In case that the columns of $V$ are orthonormal, there is a simple geometric interpretation
More general, the data lives in a manifold.
Why Nature Generates Data on Manifolds

- We encountered this in the lecture on basis functions
- Assume that nature generates data in some low-dimensional space; nature then transforms this data to a high-dimensional space by some nonlinear transformation
- This data then become the input data; then the input data might be on a manifold, as discussed in the lecture on basis functions!
- See lecture on manifold learning
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.
Data provided by nature is on a 2D manifold

ML
e.g., a neural network

Visible features accessible to machine learning: data is on a 2D manifold

Low dimensional (noisy) features in some hidden space of nature

Output
**Adversarial problem**
- Training data provided by nature is on a 2D manifold
- Test data is on a 3D manifold

```
Output

ML
  e.g., a neural network

Visible features accessible to machine learning: data is on a 2D manifold

Some test data are on a 3D manifold

Low dimensional (noisy) features in some hidden space of nature
```
An MLP with a Bottleneck Layer

Data at each layer is on a 2D manifold

"bottleneck"

Visible features accessible to machine learning (might not be on a manifold)
Manifold: Adversarial Examples

- But there is a danger: if we consider test data outside of the manifold, then performance might degrade quickly

- So although, $\epsilon_{P(x)}$ might be small, $\epsilon_B$ could be large!

- A common issue is: even on a test set (generated from the available data) the performance is excellent, but if I apply my model to new data collected independently, performance is much worse (even if $f(x)$ did not change)

- This might explain the bad performance of DNNs on adversarial examples

- Sometimes this problem is also called covariate shift (covariates are the inputs)
Conclusions

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

• To avoid the Curse of Dimensionality and if one uses fixed basis functions, \((M \times roughness)\) should not be very large

• Neural networks are effective when the basis is sparse (Ia (sparse basis)) or when data is on a manifold (Ib (data on a low-dimensional manifold))

• The next table evaluates linear models, distance-based methods (like nearest neighbor methods), models with fixed basis functions, neural networks, deep neural networks, and kernel approaches
<table>
<thead>
<tr>
<th>Target \ Model</th>
<th>Lin</th>
<th>Neighb.</th>
<th>fixed BF</th>
<th>Neural Nets</th>
<th>Deep NNs</th>
<th>Kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>I (curse)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>II (blessing)</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>III (smooth)</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>IV (simple)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Ia (sparse basis)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ib (manifold)</td>
<td>-</td>
<td>- (+dr)</td>
<td>- (+dr)</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Ic (compos.)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

- (+dr) stands for possibly good results with suitable 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
Appendix: Entropies

- Assume $n$ discretization steps for each of the $M$ input dimensions, e.g., $x_j \in 0, 1, 2, ..., n - 1$

- With $K$ discretization steps for the function, e.g., $f \in 0, 1, 2, ..., K - 1$, we can realize $K^{(n^M)}$ functions, with entropy (number of required bits) (each function has the same probability for being generated)

$$\text{Entropy}_\mathcal{F} = \log_2 K^{(n^M)} = n^M \log_2 K$$

- For each possible input, we simply need $\log_2 K$ bits and there are $n^M$ possible inputs

- Interesting: It is not the accuracy of the representation (i.e., $K$) that “kills” us, it is the dimensionality $M$ reflected in the number of possible inputs (i.e., $n^M$)

- For a model class of fixed basis functions,

$$\text{Entropy}_\mathcal{M} = \log_2 K^{(M_\phi)} = M_\phi \log_2 K$$

if we represent each weight with $\log_2 K$ bits
Appendix: VC-dimension

• For systems with fixed basis functions and binary classification, $\dim_{VC} = M_P = M_\phi$ is the VC-dimension (proportional to our entropy) of the model class

• Note that the VC-dimension is a property of the model class $\mathcal{M}$ and not of the function class $\mathcal{F}$

• If we have $N = M_\phi = \dim_{VC}$ data points, the design matrix $\Phi^T \Phi$ is a square matrix and might be invertible; in that case, no matter what the assignment of training labels $y$, we perfectly fit the classification labels (e.g., with regression)

• VC-theory states that one needs at least $\dim_{VC}$ data points for a valid generalization; this makes sense, since, without regularization, there are an infinite number of solutions when $M_\phi < N$

• Formally, $\dim_{VC}$ is defined as the cardinality of the largest set of points that the model class can shatter (i.e., perfectly model for any assignments of targets)