DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Neural Network Hyperparameter Optimization with Sparse Grids

Maximilian Michallik

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Neural Network Hyperparameter Optimization with Sparse Grids

Parameteroptimierung von neuronalen Netzen mit dünnen Gittern

Author:	Maximilian Michallik
Supervisor:	Dr. Felix Dietrich
Advisor:	Dr. Michael Obersteiner, Dr. Felix Dietrich
Submission Date:	

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Maximilian Michallik

Acknowledgments

Abstract

In recent years, machine learning has gained much importance due to the increasing amount of available data. The models that are performing very different tasks have a thing in common. They have parameters that are fixed before being trained on the data. The right choice of those hyperparameters can have a huge impact on the performance which is why they have to be optimized. Different techniques like grid search, random search, and bayesian optimization tackle this problem.
In this thesis, a new approach called adaptive sparse grid search for hyperparameter optimization is introduced. This new technique allows to adapt to the hyperparameter space and the model which leads to less training and evaluation runs compared to normal grid search while still finding the optimal model configuration for the best model results.
We compare the new approach to the other three techniques mentioned regarding execution time and resulting model performance using different machine learning tasks. The results show that adaptive sparse grid search is very efficient with a model performance similar to bayesian optimization and grid search.

Zusammenfassung

Contents

1 Introduction

2 Theoretical Background

Machine Learning [1], [2] is a rapidly evolving field of artificial intelligence. There are different types of algorithms that are used for specific tasks involving supervised learning where the algorithm maps inputs to the given labels, unsupervised learning where the labels to the input are not available, and semi-supervised learning which combines labeled and unlabeled data. Additionally, there is reinforcement learning where the model learns by observing the environment [3]. Specific tasks are e.g. classification where input has to be assigned to specific classes, regression where input has to be assigned to a continuous value (both supervised) and clustering (unsupervised) where the goal is to group the input.

There are many different algorithms that accomplish these goals, for example support vector machines [4], the tsetlin machine [5], and decision trees [6]. One very important class of algorithms is *artificial neural networks* 2.1. After the introduction to neural networks, the hyperparameter optimization is presented with different techniques to improve machine learning models. In the following, sparse grids are presented which will be needed as foundation to hyperparameter optimization of neural networks with sparse grids.

2.1 Introduction to Neural Networks

Neural networks [7], [8] are very powerful for using various tasks. They are very versatile and they exist in very different variations, ranging from a very small size up to very large networks for more complex tasks.

The smallest part of a neural network is the *perceptron*. A network consisting only of one perceptron can be seen in Figure 2.1.

The output y is computed with

$$u = \sum_{i=1}^{n} w_i \cdot x_i - \theta, y = g(u). \tag{2.1}$$

The network has n inputs x_i and weights w_i. θ is the activation threshold (also called bias), g is the activation function, and u is the activation potential [8].

This basis building block can then be used to build a more complex architecture with multiple layers. All neural networks have an input layer consisting of $n \in \mathbb{N}$

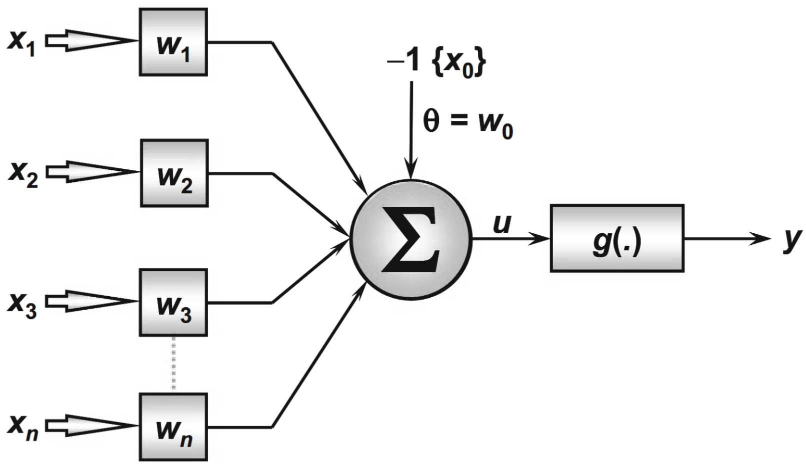

Figure 2.1: Neural network consisting of only one perceptron. Taken from [8].

input neurons and an output layer with $m \in \mathbb{N}$ output values. Between them, there can be multiple hidden neural layers. In deep neural networks, this number of layers is very high as the name suggests. Each neuron of each layer again has weights of the corresponding input and bias. The concrete values for them are important for the behavior of the model and determines the performance. These values are learned during the training phase of the model. There are two stages (forward and backward stage) during the training phase as it can be seen in Figure 2.2.

The figure shows a schematic neural network with two hidden layers and n_1 neurons in the first layer and n_2 ones in the second layer. The straight line is the forward stage where an input $x \in \mathbb{R}^n$ is put into the network and the output is computed by computing the corresponding output of each neuron and feeding it into the next layer according to the arrows. This output is then taken to update the weights of all neurons. In the simple case depicted in Figure 2.1 with only one perceptron, the weights are updated with

$$w_{current} = w_{previous} + \eta \cdot (d^{(k)} - y) \cdot x^{(k)} \tag{2.2}$$

where $w = [\theta \; w_1 \; ...w_n]^T$ is the vector with all weights and the bias, $x = [-1 \; x_1^{(k)} \; ...x_n^{(k)}]^T$ is the k$^{\text{th}}$ training sample, d^k the desired label, y the output of the perceptron and η the learning rate. The choice of η is fixed before training and usually $0 < \eta < 1$. For the update of the weights of networks with multiple layers, refer to [8].

The perceptron and its training is the basic building block for most neural networks. Based on this, the concrete architecture can still be adjusted. One first thing is to

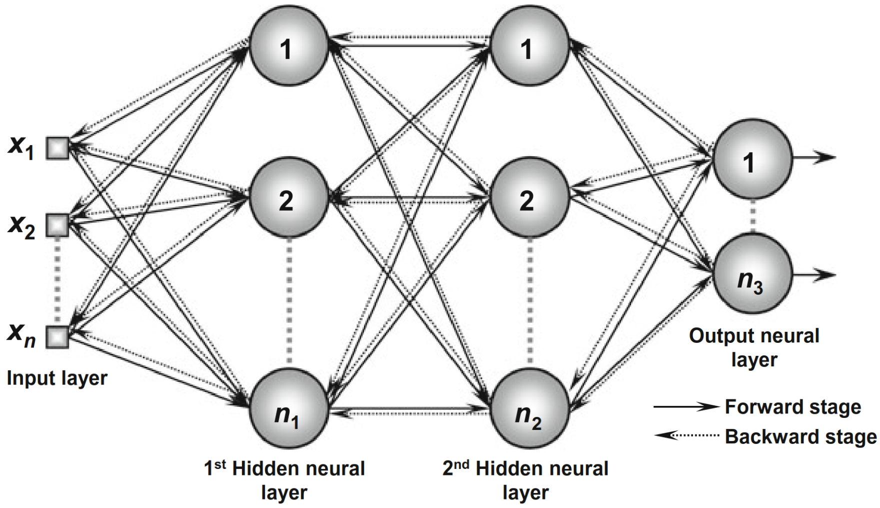

Figure 2.2: Neural network consisting of two hidden layers. Taken from [8].

increase the number of layers or neurons per layer. But also the choice of connections between layers can improve the performance of the network. Other things that can be done is to introduce connections from higher layers to lower layers which makes it a recurrent neural network. They are especially suited for sequential or time-varying patterns [9]. There is also a specific architecture for grid structured data like images. For them, convolutional neural networks are used to extract features [10]–[12]. For further readings on architecture choices, refer to [13].

In all cases, the weights of the network are updated automatically and it is impossible to understand the concrete decision making of the model. The weights are called parameters of the network. Besides them there are the hyperparameters that have to be fixed before training. They are design decisions how the network should behave. For some of them, experience can show which choices lead to better performances of the model but in all cases, they can be optimized which will be discussed in the following section 2.2. Some of the hyperparameters are

- Epochs: Number of times the training data is fed into the network and the weights are updated
- Learning rate: η of Equation 2.2 defining how fast the model should learn
- Optimizer: Optimizer used to update the weights of the network
- Loss function: Concrete loss metric how the label and the output are compared in Equation 2.2

- Batch size: Number of data samples processed in a batch
- Number of layers of the network
- Number of neurons in each layer

All these parameters can drastically influence the model performance. Some of them are independent and some of them depend each other. For example the number of epochs and learning rate are dependent because with higher number of epochs, the learning rate can be decreased as longer time to learn the data is available.

2.2 Hyperparameter Optimization

Most machine learning models have parameters that have to be defined before the learning phase. They are called hyperparameters and strongly influence the behavior of the model. One example is the number of epochs of the learning phase of a neural network. There are different techniques for the optimization of hyperparameters and they all define the machine learning model as a black box function f with the hyperparameters as input and the resulting performance as output. The overall goal is to find a configuration λ_{min} from $\Lambda = \Lambda_1 \times \Lambda_2 \times ... \times \Lambda_N$ that minimizes the function f with N hyperparameters with

$$\lambda_{min} = \arg\min_{\lambda \in \Lambda} f(\lambda). \tag{2.3}$$

In our case, the function f is a machine learning algorithm that is trained on a training set and evaluated on a test set. With this, the minimization of e.g. the loss of the model optimizes the decisions it is making which leads to better prediction results. Note that one function evaluation of f is usually very expensive as the training of a machine learning model with many parameters and weights takes much time. The data set consists of $\{(x_i, y_i) | x_i \in X, y_i \in Y, 0 \leq i \leq m\}$ with m being the number of data samples. The x_i is the input data to the model and the goal is that

$$\forall i : M(x_i) = y_i. \tag{2.4}$$

where M is the model. In the context of supervised learning, the whole data set is split into a training set which is used to optimize the model and a testing set to evaluate the performance on new, unseen data [14].

All in all, the goal is get evaluation scores on the testing data set which can be achieved with Equation 2.3. [15]–[17]

In the following, different techniques for the optimization are presented and discussed with their advantages and disadvantages.

2.2.1 Grid Search

The idea of the first approach for the optimization is to discretize the domains of each hyperparameter and evaluate each combination. This suffers from the curse of the dimensionality as it scales exponentially with the number of hyperparameters. For d parameters and n values per hyperparameter, n^d different configurations are possible which all have to be evaluated.

One advantage of this method is that it is easy to implement and very simple. Also, the whole search space is explored evenly.

On the other hand, the curse of the dimensionality makes it very slow if the function evaluations are very expensive which is the case for most machine learning algorithms. Another drawback is that each hyperparameter only takes n different values. The comparison to random search can be seen in Figure 2.3.

2.2.2 Random Search

The next technique [18] is similar to the grid search because the idea is also to evaluate different hyperparameter configurations. In contrast to the previous one, random search generates for each run and for each parameter exactly one random value from an interval which has to be specified. For this approach, a budget b has to be given. This parameter determines the number of different combinations that are evaluated. A direct comparison of grid search and random search can be seen in figure 2.3.

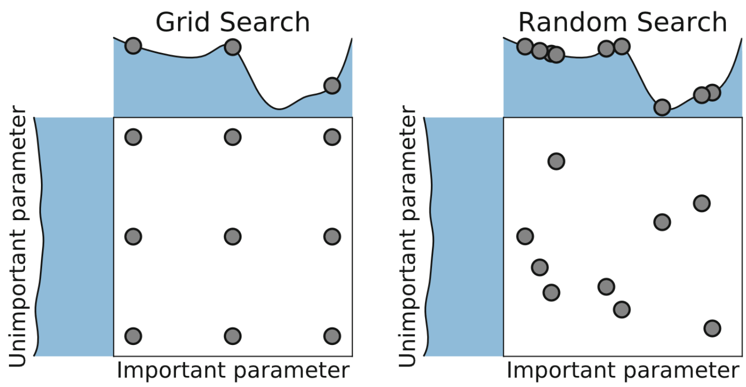

Figure 2.3: Comparison of grid search (left) and random search (right) in the two dimensional case. For both techniques, 9 different combinations are evaluated. In the left case, only 3 distinct values for each hyperparameter are set whereas there are 9 different values for each parameter in the random search. Taken from [15].

In this figure, a two dimensional setting is depicted. For both techniques, 9 different combinations are evaluated. In the case of grid search, only 3 distinct values are taken for each hyperparameter while there are 9 different ones in the random search. In this example, the better result is found in random search as more distinct values are taken for the important parameter. Note that this is not always the case.

Compared to the normal grid search, this is one advantage. For each hyperparameter, b (budget) different values are taken into consideration which is much more compared to the grid search with the same overall number of combinations. Additionally, this technique is also easy to implement and relatively simple.

One disadvantage is that it is also very expensive if the budget is high because of the long training times of machine learning models.

2.2.3 Bayesian Optimization

Another possible technique for finding the best hyperparameters of machine learning models is called bayesian optimization (BO) [19]. This is an iterative approach for optimizing the expensive black box function by modeling it based on observations. A so-called *surrogate model* $\hat{f}$ is made with the help of the *archive* A which contains observed function evaluations. This surrogate model is created by regression and the technique which is most often used is the Gaussian process [16] which is only suitable if the number of hyperparameters is not too high [20]. The problem of this technique arises when some hyperparameters are categorical or integer-valued which is the reason why extra approximations can lead to worse results and special treatment is needed [21]. Another possible technique for the surrogate model is using random forests [22]. All in all, this function estimates the machine learning model depending on the hyperparameter configuration and also the prediction uncertainty $\sigma(\lambda)$. A second function called *acquisition function* $u(\lambda)$ is built based on the prediction distribution. This u is responsible for the trade-off between exploitation and exploration. This means that configurations that lead to better model performances are exploited and values where no much information is gathered are explored. There are many numerous different possibilities for this function [23] but the most used one is the *expected improvement* (EI) which is calculated with

$$E[I(\lambda)] = E[max(f_{min} - y), 0]. \tag{2.5}$$

If the model prediction y with configuration λ follows a normal distribution [15], it leads to

$$E[max(f_{min} - y), 0] = (f_{min} - \mu(\lambda))\Phi(\frac{f_{min} - \mu(\lambda)}{\sigma}) + \sigma\phi(\frac{f_{min} - \mu(\lambda)}{\sigma}) \tag{2.6}$$

with ϕ and Φ being the standard normal density and standard normal distribution and f_{min} the best result so far.

In each iteration, a new candidate configuration λ^+ is generated by optimizing the acquisition function u. This u is much cheaper to evaluate than the f which includes learning of an expensive neural network which makes the optimization much easier.

The exact steps are presented in Algorithm 1 and Figure 2.4 shows schematic iteration steps.

Algorithm 1 Bayesian Optimization

Generate initial $\lambda^{(1)}, ..., \lambda^{(k)}$
Initialize archive $A^{[0]} \leftarrow ((\lambda^{(1)}, f(\lambda^{(1)})), ..., (\lambda^{(k)}, f(\lambda^{(k)})))$
$t \leftarrow 1$
while Stopping criterion not met **do**
  Fit surrogate model $(f(\lambda), \sigma(\lambda))$ on $A^{[t-1]}$
  Build acquisition function $u(\lambda)$ from $(\hat{f}(\lambda), \sigma(\lambda))$
  Obtain proposal λ^+ by optimizing $u : \lambda^+ \in arg \max_{\lambda \in \Lambda} u(\lambda)$
  Evaluate $f(\lambda^+)$
  Obtain $A^[t]$ by augmenting $A^{[t-1]}$ with $(\lambda^{(+)}, f(\lambda^{(+)}))$
  $t \leftarrow t + 1$
end while
return λ_{best}: Best-performing λ from archive or according to surrogates prediction

First, k initial hyperparameter configurations are sampled and evaluated. This set is the starting archive $A^{[0]}$. After that, the loop is executed as long as the stopping criterion is not met. This can be for example a budget, meaning a maximum number of function evaluations. The first step of the loop is to fit the surrogate model on the current archive. Then the acquisition function is made and optimized to get the next configuration λ^+. This point is evaluated and added to the archive. The overall result of the algorithm is the λ which is the hyperparameter configuration for the machine learning model with the overall best result.

2.2.4 Other Techniques

There are also other techniques for finding the best hyperparameters. Multi-fidelity optimization [15] aims to probe the learning of model on a task with reduced complexity such as a subset of the data or less epochs for training the model for discovering the best configurations. For example, the learning curve can be predicted so that early stopping can be done if the prediction is not as good as the best model so far. There are also bandit-based selection methods that do not predict the learning curve but

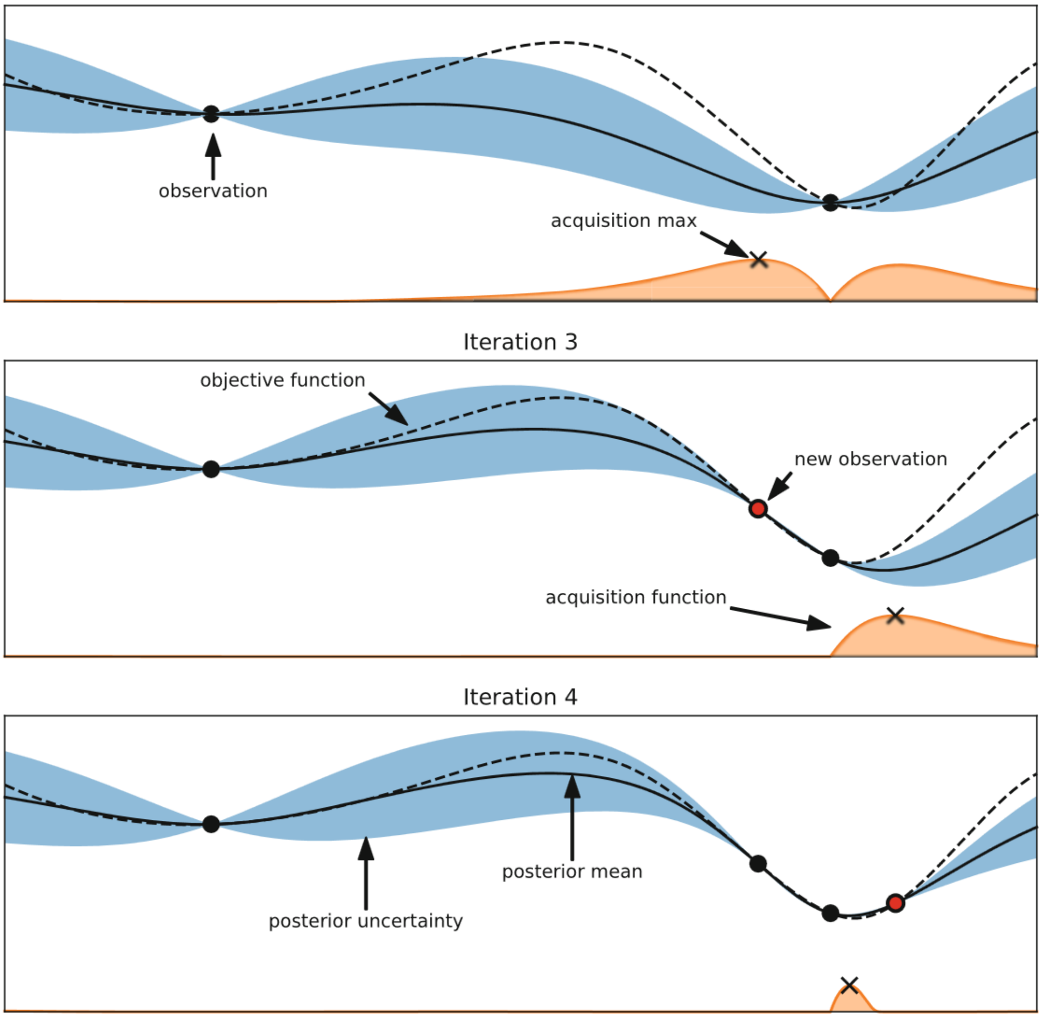

Figure 2.4: Schematic iteration steps of the bayesian optimization. The maximum of the acquisition function determines the next function evaluation (red dot in the middle). The goal is to find the minimum of the dashed line. The blue band is the uncertainty of the function. Taken from [15].

compare the different combinations on a small number of epochs and only performs the best ones. This can be done iteratively like it is done in *successive halving* for hyperparameter optimization [24]. The algorithm is very simple. It starts to evaluate all different combinations with very small budget. The best half of the candidates are then evaluated in the next iteration with double budget and so on until only one combination is left. In [25], a similar algorithm is presented. The authors use a model of the objective function (neural network depending on configurations) to find candidate hyperparameters. Those are then trained on a smaller number of epochs and the best ones then evaluated with higher budget. Also neural networks can be used for the optimization which was done by the authors in [26]. Also, covariance matrix adaptation evolution strategy was implemented as an alternative to bayesian optimization in [27].

2.3 Sparse Grids

Sparse grids are a useful tool to mitigate the *curse of the dimensionality* by reducing the number of grid points. In the following, this technique is presented after the general numerical approximation of functions.

2.3.1 Numerical Approximation of Functions

[28] Let $f : \Omega \to R$ be a function defined on the unit interval $\Omega = [0,1]^d$ in d dimensions. For simplicity, we first set $d = 1$. Now this function can be represented on a grid of level $l \in \mathbb{N}_0$ with $2^l + 1$ grid points which are

$$x_{l,i} = i * h_l,\ i = 0, ..., 2^l, \tag{2.7}$$

with i being the index and $h_l = 2^{-l}$ being the distance between the grid points. Each of them gets a basis function defined by

$$\varphi_{l,i} : [0,1] \to \mathbb{R}. \tag{2.8}$$

There are different possibilities for the basis functions which will be presented later. For the simplicity, we present a simple example being the hat function defined by

$$\varphi_{l,i}(x) = \max(1 - |\frac{x}{h_l} - i|, 0). \tag{2.9}$$

All in all, the space of functions that can be presented exactly by a linear combination is called the *nodal space V_l* with the assumption that the basis functions form a basis:

$$V_l = \text{span}\{\varphi_{l,i} | i = 0, ..., 2^l\}. \tag{2.10}$$

Every function $f : [0,1] \to \mathbb{R}$ can be interpolated by a the interpolant u defined by

$$f_l = \sum_{i=0}^{2^l} \alpha_{l,i}\varphi_{l,i}, \forall i = 0, ..., 2^l : f_l(x_{l,i}) = f(x_{l,i}) \tag{2.11}$$

for constants $\alpha_{l,i} \in \mathbb{R}$. An example can be seen in Figure 2.5.

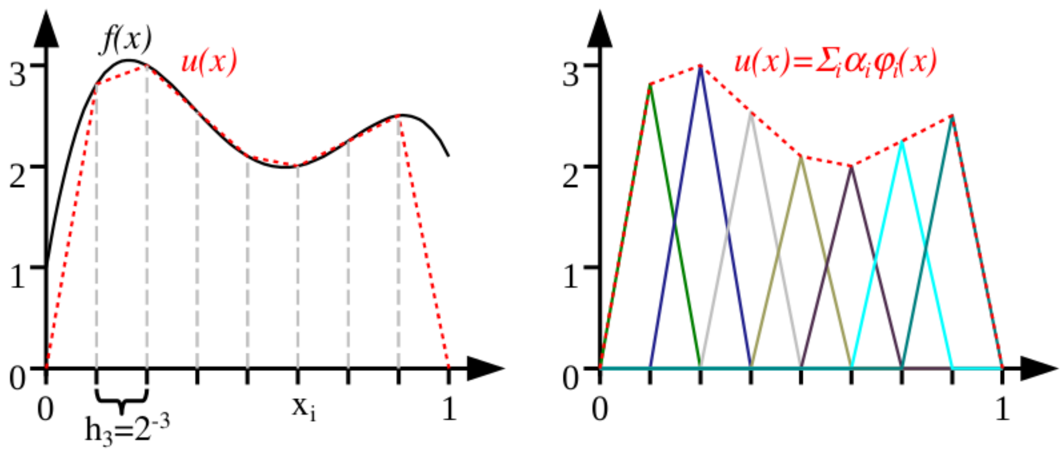

Figure 2.5: Interpolation of the function f (black line) by its interpolant u (red, dashed) in the nodal basis. Level of the grid is 3 and hat functions are used. Taken from [29].

On the left side, the function f (black line) can be seen with a grid of level 3. On the right side, the interpolant u as a linear combination of the basis functions (hat functions centered on the grid points) can be seen. This approach is the nodal basis. The second possibility is called hierarchical basis and the index set is $I_l^h = \{i \in \mathbb{N} | 1 \leq i \leq i \leq s^l - 1, i\text{odd}\}$. The hierarchical subspaces are then

$$W_l = \text{span}\{\varphi_{l,i}(x) | i \in I_l^h\}. \tag{2.12}$$

The same nodal space V_l can be obtained with the hierarchical subspaces with

$$V_l = \bigoplus_{i \leq l} W_i. \tag{2.13}$$

An example can be seen in Figure 2.6.

On the left, you can see the hierarchical subspaces up to level 3. All in all, combined they span the same space as V_3. In the hierarchical case, a function f can also be

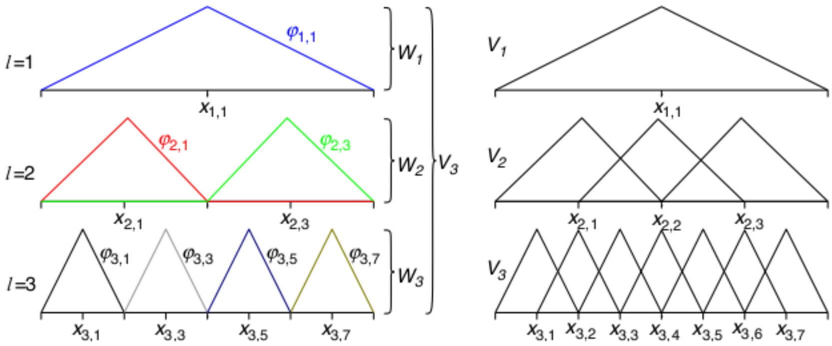

Figure 2.6: Hierarchical subspaces up to level 3 on the left. On the right, nodal spaces up to level 3. The combination of W_1 up to W_3 is the same space as V_3. Taken from [29].

interpolated by its interpolant u by

$$u = \sum_{i \in I_l^h} \alpha_{l,i}\varphi_{l,i}, \forall i = 0, ..., 2^l : u(x_{l,i}) = f(x_{l,i}). \tag{2.14}$$

An example can be seen in Figure 2.7.

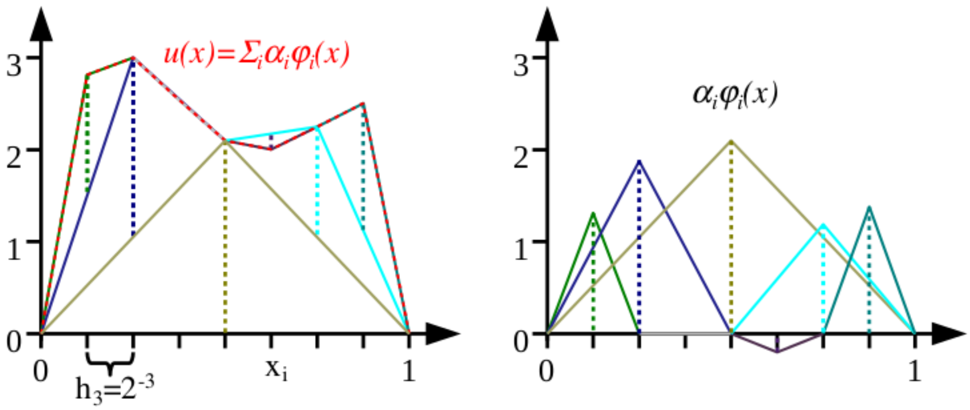

Figure 2.7: Interpolation of the function f (black line) by its interpolant u (red, dashed) in the hierarchical basis. Level of the grid is 3 and hat functions are used. Taken from [29].

To get into higher dimensions $d > 1$, we use the tensor product. The domain is now $\Omega = [0,1]^d$ and the level is defined by the level per dimension meaning

$\vec{l} = (l_1, ..., l_d) \in \mathbb{N}_0^d$. The index set is then

$$I_{\vec{l}} = \{\vec{i} | 1 \leq i_j \leq 2^{l_j} - 1, i_j \text{odd}, 1 \leq j \leq d\} \tag{2.15}$$

and the subspaces

$$W_{\vec{l}} = \text{span}\{\varphi_{\vec{l},\vec{i}}(\vec{x}) | \vec{i} \in I_{\vec{l}}\} \tag{2.16}$$

with the basis functions $\varphi_{\vec{l},\vec{i}} = \prod_{j=1}^{d} \varphi_{l_j,i_j}(x_j)$ which are constructed with the tensor product. The function space V_n is constructed by

$$V_n = \bigoplus_{|\vec{l}|_\infty \leq n} W_l \tag{2.17}$$

with $|\vec{l}| = \max_{1 \leq i \leq d} |d_i|$. Again, a function can be interpolated by its interpolant u with

$$u = \sum_{|\vec{l}|_\infty \leq n, \vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \varphi_{\vec{l},\vec{i}}, \forall \vec{i} \in I_{\vec{l}} : u(x_{\vec{l},\vec{i}}) = f(x_{\vec{l},\vec{i}}). \tag{2.18}$$

The resulting regular grid has then $(2^n - 1)^d$ basis points. An example of a basis function in two dimensions can be seen in Figure 2.8. It is constructed by the tensor product of two 1d hat functions.

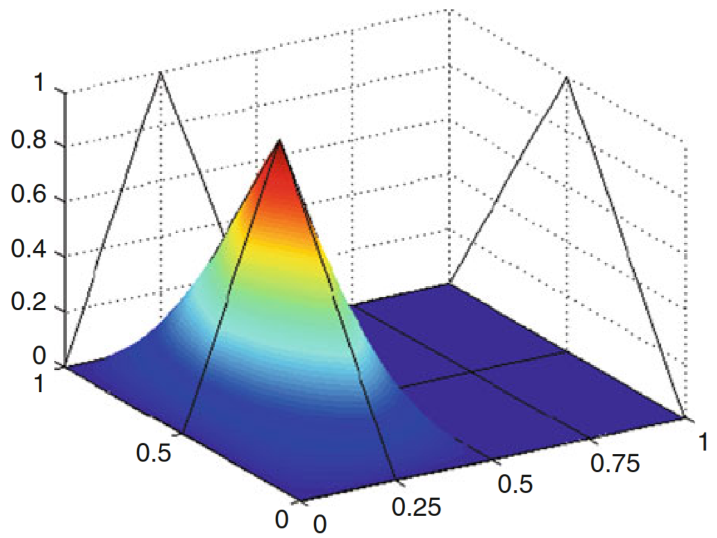

Figure 2.8: Example of a basis function in two dimensions. It is constructed with the tensor product of two 1d hat functions. Taken from [30].

In the higher dimensional case, the grid can also be constructed hierarchically. The proof that the hierarchical splitting given by

$$V_{\vec{l}} = \bigoplus_{\vec{m}=0}^{\vec{l}} W_{\vec{m}} \tag{2.19}$$

with $W_{\vec{l}} = \text{span}\{\varphi_{\vec{l},\vec{i}} | \vec{i} \in I_{\vec{l}}\}$, $I_{\vec{l}} = I_{l_1} \times ... \times I_{l_d}$ holds for the basis with hat functions can be found in [28].

2.3.2 Adaptive Sparse Grids

The problem of regular grids is the *curse of the dimensionality* because of the high number of grid points in higher dimensions. This is tackled by sparse grids [31], [32] by reducing this number. The first technique to achieve this is by just leaving out subspaces. The resulting sparse function space is given by

$$V_n^1 = \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}} \subset V_n. \tag{2.20}$$

An example with $n = 3$ can be seen in Figure 2.9.

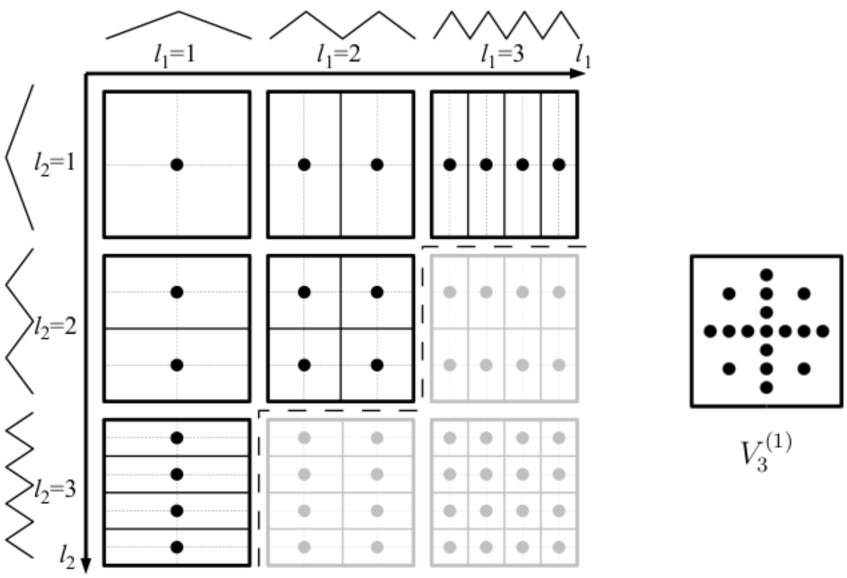

Figure 2.9: Two dimensional example of a sparse grid with $n = 3$. Left, the subspaces $W_{\vec{l}}$ can be seen and on the right is the resulting sparse grid. Taken from [30].

An interpolant u_n of a function f is then constructed by

$$u_l = \sum_{|\vec{l}|_1 \leq l+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \varphi_{\vec{l},\vec{i}} \alpha_{\vec{l},\vec{i}} \tag{2.21}$$

where the $\alpha_{\vec{l},\vec{i}}$ are the coefficients of the basis functions [33].

A second approach for sparse grids exists. The so-called *combination technique* [34] combines anisotropic full grids to get the same subspace as the conventional sparse grid approach. This has the advantage that we can use normal full grid operations on each subspace which will then be combined. This implies the possibility of parallelization. The combined solution can be computed with

$$u_l^c = \sum_{\vec{l} \in I} u_{\vec{l}} c_{\vec{l}} \tag{2.22}$$

where $\vec{l}$ is the level vector of the full grid solution $u_{\vec{l}}$, $c_{\vec{l}}$ is a scalar factor, and I is the set of included level vectors. For a standard sparse grid, this evaluates to

$$u_l^c = \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{q} \sum_{\vec{l} \in I_{l,q}} u_{\vec{l}} \tag{2.23}$$

with $I_{l,q} = \{\vec{l} \in \mathbb{N}_0^d | ||\vec{l}||_1 = l + d - 1 - q\}$ [35]. An example of the 2-dimensional combination technique can be seen on the left side of Figure 2.10.

With the normal combination technique, this grid is still symmetric and focuses on a low global error. Especially in optimization or data driven problems where the points are not distributed equally in the domain, special regions are of interest. In the case of optimization which is our focus, the errors around the extrema have to be interpolated more exactly than other regions. This is the reason why we use *refinement*. In the case of dimension-adaptive refinement [36], more grid points are added in the dimensions of higher relevance.

In contrast to the previously mentioned refinement concentrating on whole dimensions, the *spatially adaptive refinement* directly adds grid points where the discretization error is still high. An example of the spatially adaptive combination technique presented by [35] can be seen in figure 2.11. In this example, the basis points of the component grids are no longer equidistant because refinement was already made.

All in all, Table 2.1 shows the comparison of full grids, sparse grids, and the combination technique in terms of number of points and interpolation accuracy [29].

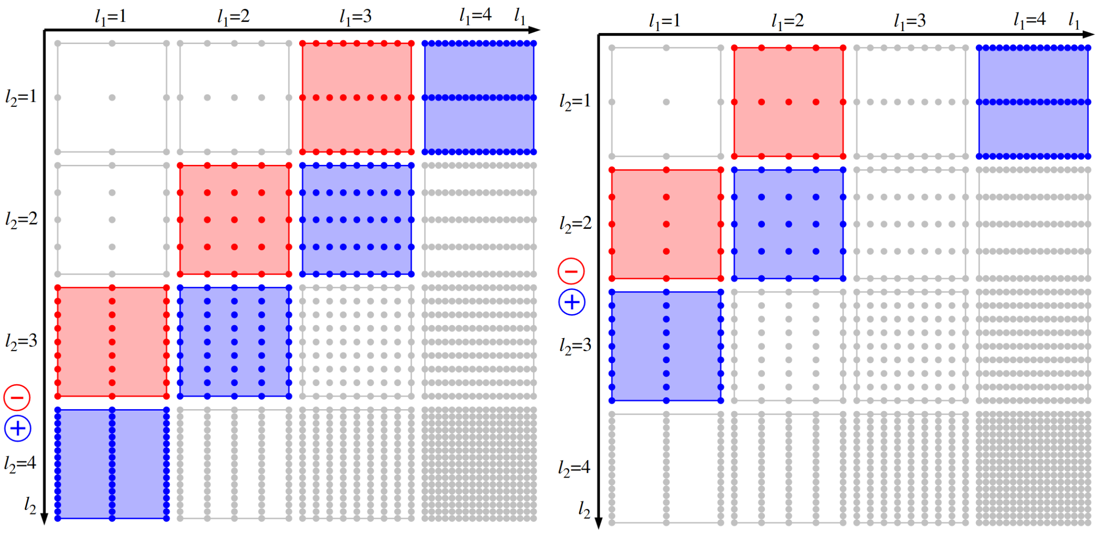

Figure 2.10: Example of the 2-dimensional combination technique. Here the blue regular grids are added and the red ones are subtracted. On the left, the normal combination technique can be seen and on the right is an dimension-adaptive version. Taken from [29].

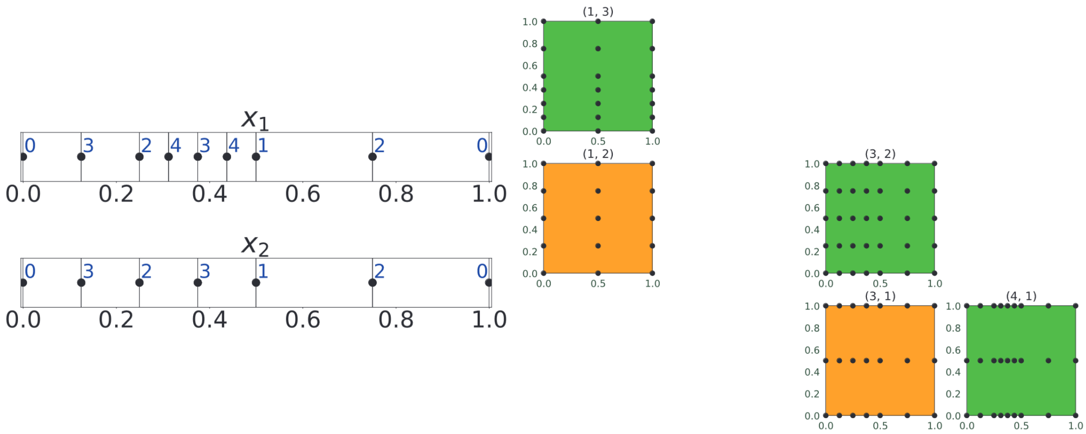

Figure 2.11: Example of the spatially adaptive combination technique in two dimensions. Taken from [35].

Grid	Number grid points	Accuracy
Full grid	$\mathcal{O}(h_n^2)$	$\mathcal{O}(2^{nd})$
Sparse grid	$\mathcal{O}(h_n^{-1}(\log h_n^{-1})^{d-1})$	$\mathcal{O}(h_n^2(\log h_n^{-1})^{d-1})$
Combination technique	$\mathcal{O}(d(\log h_n^{-1})^{d-1}) \times \mathcal{O}(h_n^{-1})$	$\mathcal{O}(h_n^2(\log h_n^{-1})^{d-1})$

Table 2.1: Comparison of sparse grids, full grids, and the combination technique in terms of number of grid points and the accuracy.

2.3.3 Basis Functions for Sparse Grids

So far, we only considered the simple case of the hat function on the support points. Besides them, there are other possibilities, for example piecewise d-polynomial, wavelet, and B-spline basis functions. For the first two cases, refer to [29], [32], [37] for further readings. In this thesis, we will concentrate on the B-spline basis for the sparse grids as the hat function is not continuously differentiable [28]. This is the reason why we can not compute globally continuous gradients which is a problem for the optimization. The general cardinal B-spline with degree $p \in \mathbb{N}_0$ is defined by

$$b^p(x) = \begin{cases} \int_0^1 b^{p-1}(x-y)\mathrm{d}y & p \geq 1 \\ \chi_{[0,1[}(x) & p = 0 \end{cases} \tag{2.24}$$

with $\chi_{[0,1[}$ being the characteristic function of the half-open unit interval [38]. The b^p as defined above has the following 8 properties:

1. compactly supported on $[0, p+1]$
2. symmetric and $0 \leq b^p \leq 1$
3. weighted combination of b^{p-1} and $-b^{p-1}$
4. piecewise polynomial of degree p
5. $\frac{d}{dx}b^p$ is the difference of b^{p-1} and $-b^{p-1}$
6. has unit integral
7. is the convolution of b^{p-1} and b^0
8. hat function and gaussian function are special cases

This is the case for uniform B-splines. For adaptive grids, the distances between basis points are not always uniform. This is the reason why we need also non-uniform

B-splines. Let $m, p \in \mathbb{N}_0$ and $\xi = (\xi_0, ..., \xi_{m+p})$ be an increasing sequence of real numbers called *knot sequence*. For $k = 0, ..., m-1$, the non-uniform B-spline is defined by

$$b^p_{k,\xi}(x) = \begin{cases} \frac{x-\xi_k}{\xi_{k+p}-\xi_k} b^{p-1}_{k,\xi}(x) + \frac{\xi_{k+p+1}-x}{\xi_{k+p+1}-\xi_{k+1}} b^{p-1}_{k+1,\xi}(x) & p \geq 1 \\ \chi_{[\xi_k, \xi_{k+1}[}(x) & p = 0 \end{cases} \tag{2.25}$$

This definition and the proof that the hierarchical splitting also holds for using the B-splines for restricted functions can be found in [28].

3 Related Work

4 Hyperparameter optimization with sparse grids

4.1 Methodology

4.1.1 Adaptive Grid Search with Sparse Grids

4.1.2 Implementation

4.2 Results

5 Conclusion and Outlook

List of Figures

List of Tables

Bibliography

[1] H. Wang, Z. Lei, X. Zhang, B. Zhou, and J. Peng, "Machine learning basics," *Deep learning*, pp. 98–164, 2016.

[2] B. Mahesh, "Machine learning algorithms-a review," *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020.

[3] T. O. Ayodele, "Types of machine learning algorithms," *New advances in machine learning*, vol. 3, pp. 19–48, 2010.

[4] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[5] O.-C. Granmo, "The tsetlin machine–a game theoretic bandit driven approach to optimal pattern recognition with propositional logic," *arXiv preprint arXiv:1804.01508*, 2018.

[6] L. Rokach and O. Maimon, "Decision trees," *Data mining and knowledge discovery handbook*, pp. 165–192, 2005.

[7] C. M. Bishop, "Neural networks and their applications," *Review of scientific instruments*, vol. 65, no. 6, pp. 1803–1832, 1994.

[8] I. N. Da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni, S. F. dos Reis Alves, I. N. da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, *Artificial neural network architectures and training processes*. Springer, 2017.

[9] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, pp. 64–67, 2001.

[10] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[11] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[12] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[13] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[14] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," *Machine learning techniques for multimedia: case studies on organization and retrieval*, pp. 21–49, 2008.

[15] M. Feurer and F. Hutter, "Hyperparameter optimization," *Automated machine learning: Methods, systems, challenges*, pp. 3–33, 2019.

[16] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, e1484, 2021.

[17] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[18] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[19] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[20] R. Andonie, "Hyperparameter optimization in learning systems," *Journal of Membrane Computing*, vol. 1, no. 4, pp. 279–291, 2019.

[21] E. C. Garrido-Merchán and D. Hernández-Lobato, "Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes," *Neurocomputing*, vol. 380, pp. 20–35, 2020.

[22] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, Springer, 2011, pp. 507–523.

[23] J. Wilson, F. Hutter, and M. Deisenroth, "Maximizing acquisition functions for bayesian optimization," *Advances in neural information processing systems*, vol. 31, 2018.

[24] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Artificial intelligence and statistics*, PMLR, 2016, pp. 240–248.

[25] G. I. Diaz, A. Fokoue-Nkoutche, G. Nannicini, and H. Samulowitz, "An effective algorithm for hyperparameter optimization of neural networks," *IBM Journal of Research and Development*, vol. 61, no. 4/5, 9:1–9:11, 2017. DOI: 10.1147/JRD.2017.2709578.

[26] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, "Neural networks designing neural networks: Multi-objective hyper-parameter optimization," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2016, pp. 1–8.

[27] I. Loshchilov and F. Hutter, "Cma-es for hyperparameter optimization of deep neural networks," *arXiv preprint arXiv:1604.07269*, 2016.

[28] J. Valentin, "B-splines for sparse grids : Algorithms and application to higher-dimensional optimization," 2019. DOI: http://dx.doi.org/10.18419/opus-10504.

[29] D. M. Pflüger, "Spatially adaptive sparse grids for high-dimensional problems," Ph.D. dissertation, Technische Universität München, 2010.

[30] J. Garcke, "Sparse grids in a nutshell," in *Sparse grids and applications*, Springer, 2013, pp. 57–80.

[31] C. Zenger and W. Hackbusch, "Sparse grids," in *Proceedings of the Research Workshop of the Israel Science Foundation on Multiscale Phenomenon, Modelling and Computation*, 1991, p. 86.

[32] H.-J. Bungartz and M. Griebel, "Sparse grids," *Acta numerica*, vol. 13, pp. 147–269, 2004.

[33] M. Obersteiner and H.-J. Bungartz, "A spatially adaptive sparse grid combination technique for numerical quadrature," in *Sparse Grids and Applications-Munich 2018*, Springer, 2022, pp. 161–185.

[34] M. Griebel, M. Schneider, and C. Zenger, "A combination technique for the solution of sparse grid problems," 1990.

[35] M. Obersteiner and H.-J. Bungartz, "A generalized spatially adaptive sparse grid combination technique with dimension-wise refinement," *SIAM Journal on Scientific Computing*, vol. 43, no. 4, A2381–A2403, 2021.

[36] M. Hegland, "Adaptive sparse grids," *Anziam Journal*, vol. 44, pp. C335–C353, 2002.

[37] H.-J. Bungartz, "Finite elements of higher order on sparse grids," Ph.D. dissertation, Technische Universität München, 1998.

[38] K. Höllig and J. Hörner, *Approximation and modeling with B-splines*. SIAM, 2013.