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:	

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 Related Work

2.1 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

$$\lambda_{min} = \arg\min_{\lambda \in \Lambda} f(\lambda). \tag{2.1}$$

In our case, the function f is a machine learning algorithm that is trained on a training set and evaluated on a testing 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

$$\forall i : M(x_i) = y_i. \tag{2.2}$$

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 [1].

All in all, the goal is get evaluation scores on the testing data set which can be achieved with Equation 2.1. [2]–[4]

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

2.1.1 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. This is the case for most machine learning algorithms because the training phase of the model takes much time. Another drawback is that each hyperparameter only takes n different values.

2.1.2 Random Search

The next technique [5] 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 2.1.

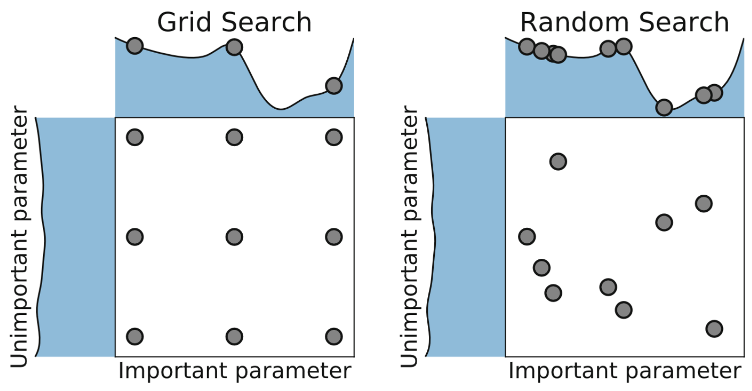

Figure 2.1: Comparison of grid search (left) and random search (right) in the two dimensional case. For both techniques, 9 different combinations are evaluated. In the left case, only 3 distinct values for each hyperparameter are set whereas there are 9 different values for each parameter in the random search. Taken from [2].

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.

2.1.3 Bayesian Optimization

Another possible technique for finding the best hyperparameters of machine learning models is called bayesian optimization (BO) [6]. 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 [3] which is only suitable if the number of hyperparameters is not too high [7]. 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 [8]. Another possible technique for the surrogate model is using random forests [9]. 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 [10] but the most used one is the *expected improvement* (EI) which is calculated with

$$E[I(\lambda)] = E[max(f_{min} - y), 0]. \tag{2.3}$$

If the model prediction y with configuration λ follows a normal distribution [2], it leads to

$$E[max(f_{min} - y), 0] = (f_{min} - \mu(\lambda))\Phi(\frac{f_{min} - \mu(\lambda)}{\sigma}) + \sigma\phi(\frac{f_{min} - \mu(\lambda)}{\sigma}) \tag{2.4}$$

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 2.2 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 $\lambda^+$ 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 $\lambda_{best}$: Best-performing $\lambda$ 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.

2.1.4 Other Techniques

There are also other techniques for finding the best hyperparameters. Multi-fidelity optimization [2] 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

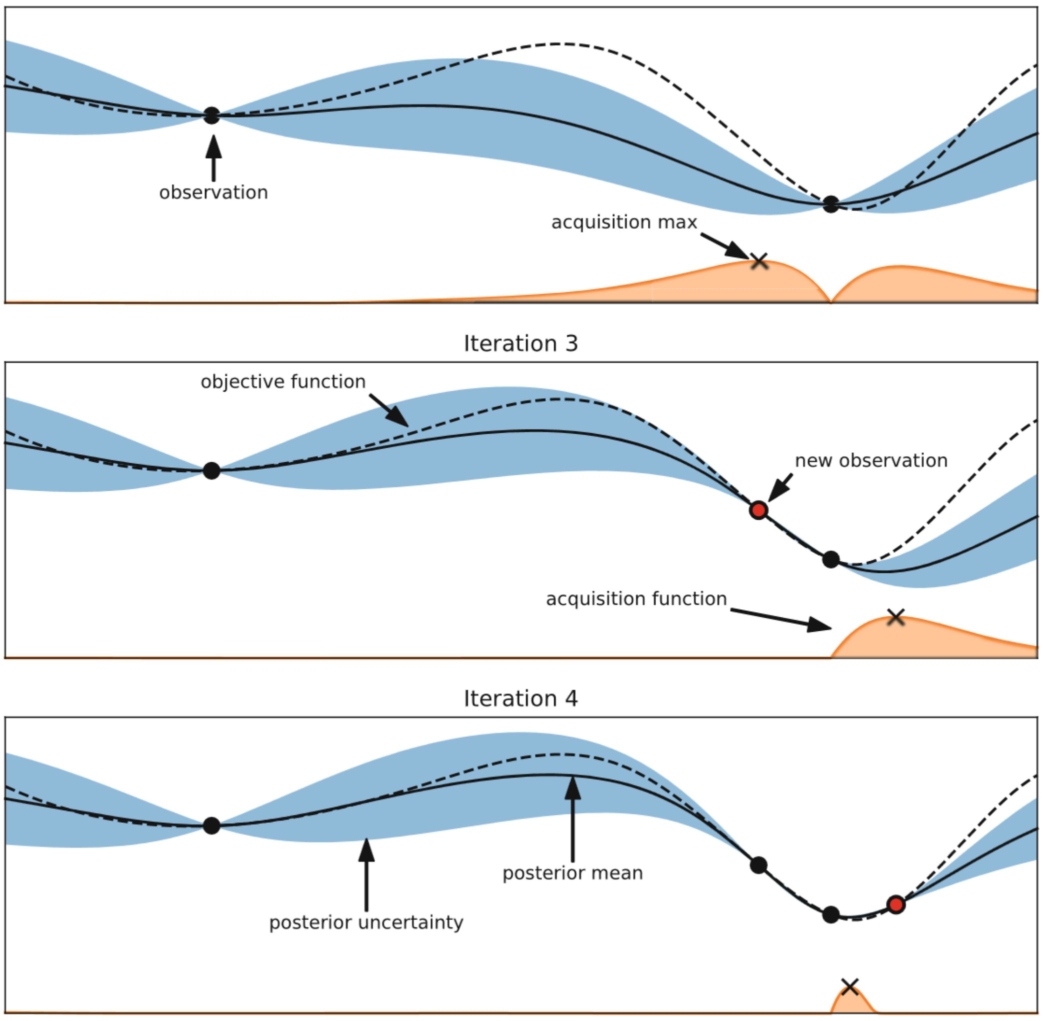

Figure 2.2: Schematic iteration steps of the bayesian optimization. The maximum of the acquisition function determines the next function evaluation (red dot in the middle). The goal is to find the minimum of the dashed line. The blue band is the uncertainty of the function. Taken from [2].

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 [11]. 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 [12], 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 [13]. Also, covariance matrix adaptation evolution strategy was implemented as an alternative to bayesian optimization in [14].

2.2 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.

2.2.1 Numerical Approximation of Functions

[15] 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

$$x_{l,i} = i * h_l, \ i = 0, ..., 2^l, \tag{2.5}$$

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

$$\varphi_{l,i} : [0,1] \to \mathbb{R}. \tag{2.6}$$

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

$$\varphi_{l,i}(x) = \max(1 - |\frac{x}{h_l} - i|, 0). \tag{2.7}$$

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:

$$V_l = \operatorname{span}\{\varphi_{l,i} | i = 0, ..., 2^l\}. \tag{2.8}$$

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

$$f_l = \sum_{i=0}^{2^l} \alpha_{l,i} \varphi_{l,i}, \forall i = 0, ..., 2^l : f_l(x_{l,i}) = f(x_{l,i}) \tag{2.9}$$

for constants $\alpha_{l,i} \in \mathbb{R}$. An example can be seen in Figure 2.3.

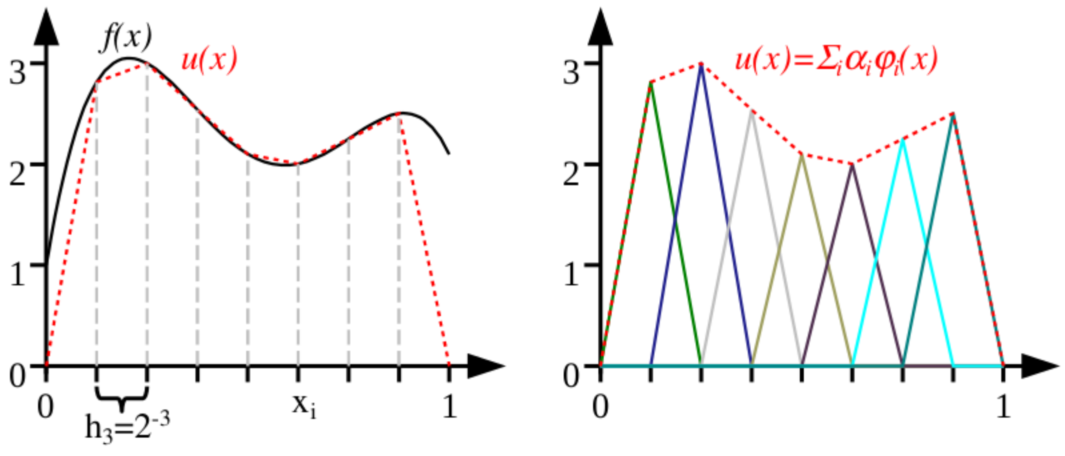

Figure 2.3: Interpolation of the function f (black line) by its interpolant u (red, dashed) in the nodal basis. Level of the grid is 3 and hat functions are used. Taken from [16].

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

$$W_l = \text{span}\{\varphi_{l,i}(x) | i \in I_l^h\}. \tag{2.10}$$

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

$$V_l = \bigoplus_{i \leq l} W_i. \tag{2.11}$$

An example can be seen in Figure 2.4.

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

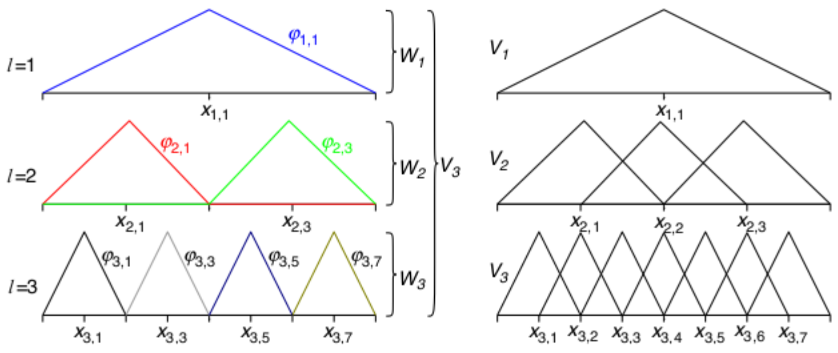

Figure 2.4: Hierarchical subspaces up to level 3 on the left. On the right, nodal spaces up to level 3. The combination of W_1 up to W_3 is the same space as V_3. Taken from [16].

interpolated by its interpolant u by

$$u = \sum_{i \in I_l^h} \alpha_{l,i} \varphi_{l,i}, \forall i = 0, ..., 2^l : u(x_{l,i}) = f(x_{l,i}). \tag{2.12}$$

An example can be seen in Figure 2.5.

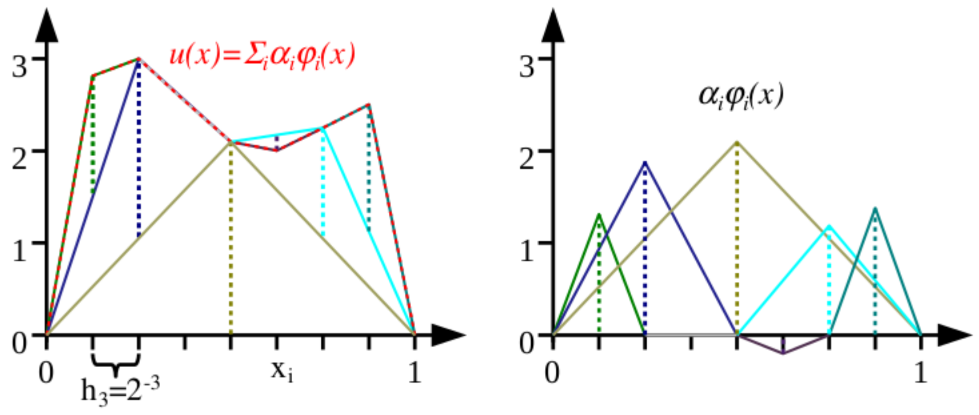

Figure 2.5: Interpolation of the function f (black line) by its interpolant u (red, dashed) in the hierarchical basis. Level of the grid is 3 and hat functions are used. Taken from [16].

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

$$I_{\vec{l}} = \{\vec{i} | 1 \leq i_j \leq 2^{l_j} - 1, i_j \text{odd}, 1 \leq j \leq d\} \tag{2.13}$$

and the subspaces

$$W_{\vec{l}} = \text{span}\{\varphi_{\vec{l},\vec{i}}(\vec{x}) | \vec{i} \in I_{\vec{l}}\} \tag{2.14}$$

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

$$V_n = \bigoplus_{|\vec{l}|_\infty \leq n} W_l \tag{2.15}$$

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

$$u = \sum_{|\vec{l}|_\infty \leq n, \vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \varphi_{\vec{l},\vec{i}}, \forall \vec{i} \in I_{\vec{l}} : u(x_{\vec{l},\vec{i}}) = f(x_{\vec{l},\vec{i}}). \tag{2.16}$$

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 2.6. It is constructed by the tensor product of two 1d hat functions.

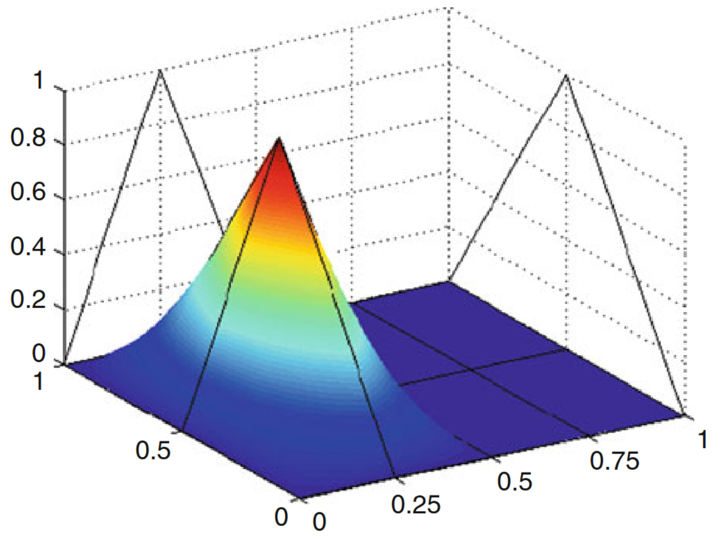

Figure 2.6: Example of a basis function in two dimensions. It is constructed with the tensor product of two 1d hat functions. Taken from [17].

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

$$V_{\vec{l}} = \bigoplus_{\vec{m}=0}^{\vec{l}} W_{\vec{m}} \tag{2.17}$$

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 [15].

2.2.2 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 [18] by reducing this number. The first technique to achieve this is by just leaving out subspaces. The resulting sparse function space is given by

$$V_n^1 = \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}} \subset V_n. \tag{2.18}$$

An example with $n = 3$ can be seen in Figure 2.7.

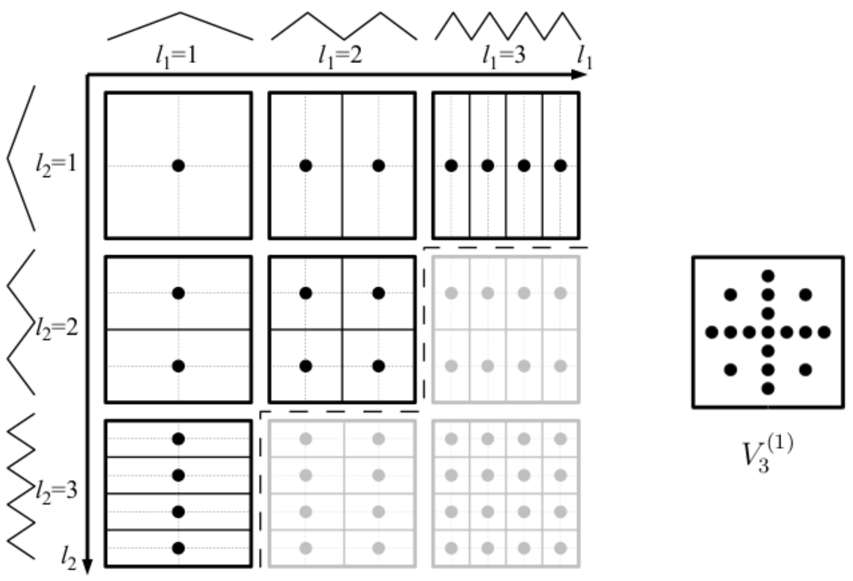

Figure 2.7: Two dimensional example of a sparse grid with $n = 3$. Left, the subspaces $W_{\vec{l}}$ can be seen and on the right is the resulting sparse grid. Taken from [17].

An interpolant u_n of a function f is then constructed by

$$u_l = \sum_{|\vec{l}|_1 \leq l+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \varphi_{\vec{l},\vec{i}} \alpha_{\vec{l},\vec{i}} \tag{2.19}$$

where the $\alpha_{\vec{l},\vec{i}}$ are the coefficients of the basis functions [19].

3 Hyperparameter optimization with sparse grids

3.1 Methodology

3.1.1 Adaptive Grid Search with Sparse Grids

3.1.2 Implementation

3.2 Results

4 Conclusion and Outlook

List of Figures

List of Tables

Bibliography

[1] 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.

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

[3] 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.

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

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

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

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

[8] 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.

[9] 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.

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

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

[12] 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.

[13] 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.

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

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

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

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

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

[19] 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.