DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

Neural Sparse Grids for High-Dimensional Data

Zhen Zhang

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

Neural Sparse Grids for High-Dimensional Data

Neuronale Dünne Gitter für hochdimensionale Daten

Author:	Zhen Zhang
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Dr. Felix Dietrich
Submission Date:	May 15th, 2021

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

Munich, May 15th, 2021 Zhen Zhang

Acknowledgments

I would like to thank my advisor, Dr. Felix Dietrich, for his invaluable support, advice and feedback.

Abstract

Sparse Grids have been proven to be compelling tools for function approximation. They are based on an efficient space discretization scheme and are usually reliable and explainable through their mathematical foundations. Although Sparse Grids mitigate the curse of dimensionality, they still can hardly deal with high-dimensional data. Artificial Neural Networks, based on a linear combination of learned nonlinear basis functions, can easily deal with high-dimensional data but have drawbacks, such as being brittle and unreliable.

This thesis combines both approaches and proposes a new method called Neural Sparse Grids, in which a Sparse Grid layer is appended to an artificial neural network. The main focus is to show how this new approach can be applied in supervised learning, including regression and classification. We demostrate the approach on a swissroll dataset, a two moons variant dataset, a stellar spectra dataset and the MNIST dataset and see greatly improved performance compared to traditional machine learning with sparse grids. Furthermore, there are also discussions about the results and the problems we encounter.

Contents

1 Introduction

Nowadays, machine learning tools are becoming more critical due to the large amount of data available. Many applications are being used in not only business but also scientific scenarios. They can be used to extract useful information from data and make predictions. Artificial neural network and Sparse grids are two of these tools.

Artificial neural networks have achieved great success in the past years, especially in computer vision and natural language process fields. However, they sometimes can be unreliable and provide incorrect predictions. Moreover, they cannot easily be interpreted. Technically, they represent functions by a linear combination of nonlinear basis functions and adjust parameters according to the data until they work as expected. A good thing about them is that they can tackle high dimensional data.

Sparse Grids are a space discretization technique for function approximation. They are usually reliable and explainable through their foundations in mathematics. Although Sparse Grids mitigate the curse of dimensionality compared to the full grid approach, they still suffer from the problem. They still discretize the full ambient space and cannot deal with low-dimensional, nonlinear structures embedded in very high dimensions. For example, a regular sparse grids of level 5 in 10 dimension requires 13441 parameters. For higher dimension, they need more even if the actual data has lower intrinsic dimension.

Therefore, we combine these two approaches to function approximation: Sparse Grids and Artificial Neural Networks. We use Artificial Neural Networks to reduce the dimension and Sparse Grids to do the specific task afterwards. The new approach is called Neural Sparse Grids.

Ch. 2 theoretical background provides, as the name indicates, an introduction to the concepts needed for Neural Sparse Grids, including Sparse Grids as well as Artificial Neural Networks. It explains the hierarchical basis for grid based function approximation, which is followed by regular sparse grids. Then, the boundary treatment is also discussed there. In addition, a basic artificial neural network architecture, multilayer perceptron, is introduced, along with how an ANN can be trained. It follows a brief introduction of the convolutional neural network that will be used in the next chapter.

Then we open the stage for neural sparse grids in Ch. 3. Starting with the introduction to model architecture, we discuss how to construct such a model and the possible methods to learn its parameters. Moreover, it also considers the problems of regression and shows how neural sparse grids can deal with them. We will show an artificial example as well as an example from the real world in astronomy, using stellar spectra to estimate its atmospheric parameters. Furthermore, neural sparse grids are also applied to solve classification problems. Again, we will deal with an artificial data set as well as a real-world data set, handwritten digital image data set, MNIST. There, the convolutional neural networks is integrated into neural sparse grids. Last but not least, there is a discussion about the results and observations from the experiments, including training, latent space and comparisons.

2 Theoretical Background

In this chapter, the necessary background of Neural Sparse Grids is introduced. On the one hand, it provides an overview of Sparse Grids and associated concepts of Sparse Grids, such as grid based function approximation and hierarchical basis. On the other hand, Artificial Neural Networks are introduced briefly, including classic architectures like multiplayer perceptron and convolutional neural network.

2.1 Sparse Grids

A function can be interpolated in a piecewise way in its domain. In a grid-based approach, we can approximate it as a sum of weighted basis function. It is, in fact, a linear combination of basis functions whose number does not increase with the number of data points. Introduced by [4], sparse grids discretize the domain with some grid points and corresponding basis functions. This section reviewed basic concepts of sparse grids, including hierarchical basis, regular sparse grids construction and boundary treatment.

2.1.1 Intepolation on a Full Grid

Suppose the function we want to approximate f is on the domain $\Omega \in [0,1]^d$. This can be easily achieved by scaling the original domain. In addition, we restrict this function f to 0 on the boundary, which is released by boundary treatment later.

Firstly, to construct such interpolation of function f, we discretize Ω with mesh width 2^{-l} in each dimension, l being the discretization level. Obviously, large l brings a higher "resolution" to Ω while leads to more grid points. Next, we define some basis function $\varphi_i(\vec{x})$ centered at these grid points. A very commonly used one is the piecewise d-linear basis function. Then, we obtain the interpolant $u(\vec{x})$ through

$$f(\vec{x}) \approx u(\vec{x}) := \sum_i \alpha_i \varphi_i(\vec{x}). \tag{2.1}$$

In fact, there are a lot of choices for basis function other than piecewise d-linear basis function, like d-polynomial, Mexiacan hat and B-spine. But we will not discuss them in this thesis.

2.1.2 Hierarchical Basis

Sparse Grids are based on a hierarchical decomposition of target spaces, i.e., we discretize the target space in different levels and finally combine them linearly to get our interpolant u. For simplicity, we first only consider one-dimensional case that will later be extended to d-dimensional case.

One dimensional hierarchical decomposition

First we define the standard hat function in Eq. 2.2

$$\varphi(x) = \max(1 - |x|, 0), \tag{2.2}$$

which will be the standard choice in this thesis. A given grid point is $x_{l,i} = \frac{i}{2^l}$, where l stands for level and $0 < i < 2^l$ is its index in this level The basis function centered at $x_{l,i}$ is thus,

$$\varphi_{l,i} := \varphi(2^{-l}x - i). \tag{2.3}$$

In many cases, not all the grid points in this level are used. So for a given discretization level l, we introduce the hierarchical index sets,

$$I_l := \{i \in \mathbb{N} : 1 \leq i \leq 2^l - 1, i \; odd\}, \tag{2.4}$$

which results in the hierarchical subspaces W_l,

$$W_l := span\{\varphi_{l,i}(x) : i \in I_l\}. \tag{2.5}$$

Fig. 2.1 shows all basis function in level from 1 to 3 where basis functions in the same

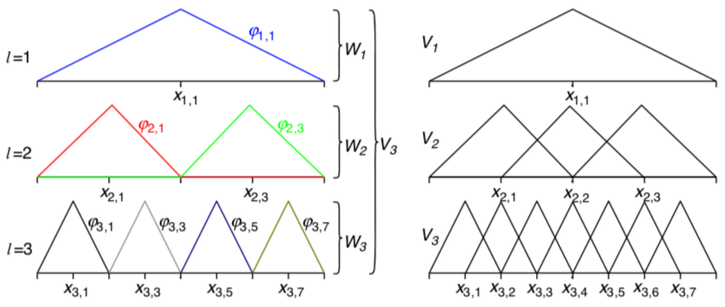


Figure 2.1: Hat basis functions centered at different grid points for $l \leq 3$ (left) and nodal point basis(right)

level has the same shape but the grid points do not overlap across all these levels.

Considering the maximum level n, we have the function space V_n as the sum of such subspaces,

$$V_n = \bigoplus_{l \leq n} W_l. \tag{2.6}$$

An interpolation function $u(x) \in V_n$ is thus,

$$u(x) = \sum_{l \leq n, i \in I_l} \alpha_{l,i} \varphi_{l,i}(x). \tag{2.7}$$

D-dimensional hierarchical decomposition

Now we consider the d-dimension case. Firstly, the basis functions can be easily extended to the d-dimensional case by multiplying the respective one-dimensional basis functions as is shown in Eq. 2.8

$$\varphi_{\vec{l},\vec{i}}(\vec{x}) := \prod_{j=1}^{d} \varphi_{l_j,i_j}(x_j), \tag{2.8}$$

where level and index become a vector in which l_j and i_j are level and index in dimension j.

Next, similar to Eq. 2.3 the index set is extended to

$$I_{\vec{l}} := \{\vec{i} : 1 \leq i_j \leq 2^{l_j} - 1, i_j\ odd, 1 \leq j \leq d\}, \tag{2.9}$$

along with the level $\vec{l}$ subspace becoming the span of all the basis functions as is shown in Eq. 2.10

$$W_{\vec{l}} := span\{\varphi_{\vec{l},\vec{i}}(\vec{x}) : \vec{i} \in I_l\}. \tag{2.10}$$

Same as 1d case Eq. 2.6, the functions space V_n up to level n is still sum of each level $\vec{l}$,

$$V_n = \bigoplus_{|l|_\infty \leq n} W_l. \tag{2.11}$$

It leads to an interpolant $u(\vec{x}) \in V_n$,

$$u(\vec{x}) = \sum_{|\vec{l}|_\infty \leq n, \vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \varphi_{\vec{l},\vec{i}}(\vec{x}). \tag{2.12}$$

2.1.3 Sparse Grids

As we can imagine, using these hierarchical basis to do function approximation is really expensive as we have so many grid points that may increase the complexity. But sparse grids omit some points of a full grid while still keeps the power of approximation.

When selecting certain subspaces $W_{\vec{l}}$, we omit subspaces whose level does not satisfy $|\vec{l}|_1 \leq n + d - 1$. So we get the following function space for sparse grid,

$$V_n^{(1)} := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}. \tag{2.13}$$

Fig. 2.2 shows an example of sparse grids of level 3 in 2d. Formally, the sparse grid interpolant $u(\vec{x}) \in V_n^{(1)}$ of level n is defined,

$$u(\vec{x}) = \sum_{|\vec{l}|_1 \leq n+d-1, \vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \varphi_{\vec{l},\vec{i}}(\vec{x}). \tag{2.14}$$

The sparse grids shown above is called regular sparse grids while there are also different

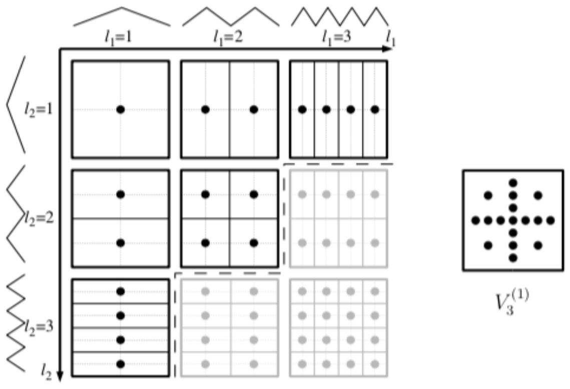


Figure 2.2: A level 3 regular sparse grids in 2d

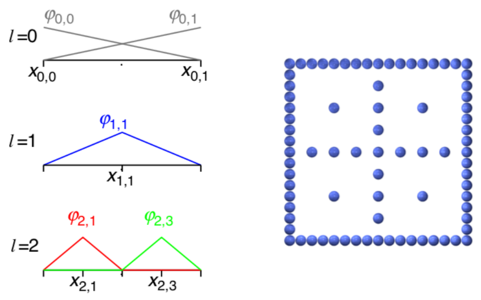
variants of it, such as sparse grid combination technique, $\sqrt{h}$ sparse grids and full grids. Each of them has other attributes, but we will not discuss them in this thesis.

2.1.4 Boundary Treatment

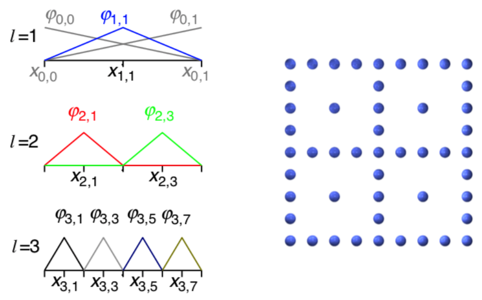
So far, we assumed that function values on the boundary are zeros. However, in reality, it is not often the case. To support non-zero boundary, we introduce level 0 with indices 0 and 1, i.e. $\varphi_{0,0}$ and $\varphi_{0,1}$. Using the same construction logic before, we get a set of subspaces $\tilde{W}_{\vec{l}}$ and a new sparse grids space,

$$V_n^{0B(1)} = \bigoplus_{|\vec{l}|_1 \leq n+d-1} \tilde{W}_{\vec{l}}. \tag{2.15}$$

The basis function and sparse grids are shown in Fig. 2.3a.



(a) Basis functions in 1d with level 0 functions on the boundary and 2d grids of level 3



(b) Basis functions in 1d with level 1 extended by extra functions on the boundary and 2d grids of level 3

Figure 2.3: Two methods of boundary treatment. Figures are from [20]

A straightforward intuition is that this routine results in too many grid points and will suffer from the curse of dimensionality. Therefore, we can extend level 1 by putting 2 basis functions in level 0 to it. It means we have $\varphi_{0,0}$ and $\varphi_{0,1}$ in level 1. This results in sparse grid space in Eq. 2.16. Fig. 2.3b shows an example of it.

$$V_n^{B(1)} = \bigoplus_{|\vec{l}|_1 \leq n+d-1} \hat{W}_{\vec{l}} \tag{2.16}$$

2.2 Artificial Neural Networks

Artificial neural networks [15] are composed of interconnected basic computing units called neurons. Just like in biology, a natural neuron is a cell that can be electrically stimulated. An artificial neuron receives input vectors, and each value has a different weight. It calculates a scalar value and sends the results to the next neuron. Once we have the network topology defined, we need to find the best weights to adapt to the data. The purpose of learning (or training) is to iteratively adjust the weights until the network works as expected.

2.2.1 Multilayer Perceptron

The Multilayer Perceptron(or MLP, multilayer neural network, dense neural network) is the basis of other advanced NN models. In this type of ANN, neurons are organized hierarchically where those in one layer are connected to all of(or part of) the others in the next layer. Fig. 2.4 shows an example of 3 layers.

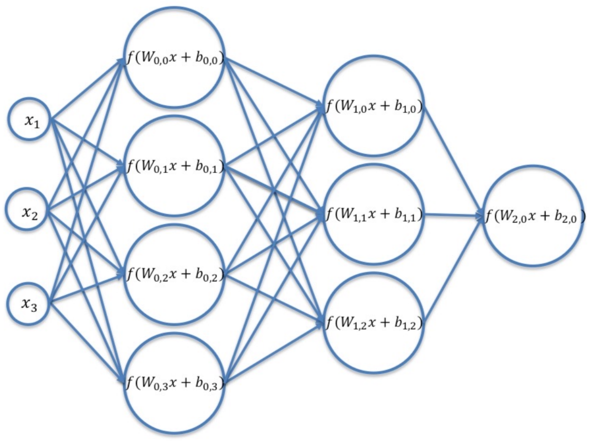


Figure 2.4: An example of MLP with layers$(4, 3, 1)$

Formally, let $\vec{x}^{(N-1)}$ be the input of N^{th} layer, $W^{(N)}$ be a matrix of weights, $\vec{b}^{(N)}$ be the bias. The output of this layer is thus

$$\vec{x}^{(N)} = f^{(N)}(W^{(N)}\vec{x}^{(N-1)} + \vec{b}^{(N)}). \tag{2.17}$$

where $f^{(N)}$ is an activation function. Therefore, we can write the model as

$$f(\vec{x}) = W^{(N)}f^{(N)}(W^{(N-1)}f^{(N-1)}(...f^{(0)}(W^{(0)}\vec{x}^{(0)} + \vec{b}^{(0)})...)) + \vec{b}^{(N)}. \tag{2.18}$$

Activation functions are necessary as they provide nonlinearity to our model. Otherwise, it will just be a linear transformation. Traditional activation functions are $tanh$ and $sigmoid$. However, the gradient of both nonlinearities saturates very much, especially in cases of deep layers. Recently, Rectified Linear Unit(ReLU)[2] is more often used as its gradient is either 0 or 1.

Besides, for classification tasks, a softmax is typically the last layer in ANN as it normalizes C outputs to real values in range $(0, 1)$ and ensures the sum of them equals 1. Thus we can take these normalized values as probabilities of each class. The probability of class j is calculated by

$$softmax_j(\vec{y}) = \frac{e^{y_j}}{\sum_{k=1}^{C} e^{y_k}}. \tag{2.19}$$

2.2.2 Learning

Learning is the process that we adjust parameters including weights and bias according to the given training data set in order to get outputs as close as possible to actual values. So firstly, we have to introduce a loss function for the task to measure how "bad" the current parameters are. Thus our goal becomes minimizing such a function. Usually for a classification task of M classes, the loss function is Cross-Entropy loss as is shown in Eq. 2.20.

$$L_{cross\_entropy} = \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{M} y_{i,c}\log p_{i,c} \tag{2.20}$$

where $y_{i,c}$ represents i^{th} observation being class c or not in a one-hot vector way. $p_{i,c}$ means the predicted probability for i^{th} observation being class c. For regression tasks, the most commonly used loss function is Mean Square Error(MSE) in Eq. 2.21.

$$L_{mse} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{2.21}$$

where $\hat{y}_i$ represents the predicted value for i^{th} observation while y_i is the ground truth.

Now we can reformulate the problem. Assuming we have an ANN $\mathcal{N}_{\vec{\theta}}(\vec{x})$ parameterized by $\vec{\theta}$ and we defined a loss function L, our goal is

$$\vec{\theta}^{*} = \mathrm{argmin}_{\vec{\theta}} L(\mathcal{N}_{\vec{\theta}}(\vec{x}), \vec{y}). \tag{2.22}$$

Since it is not a convex problem, a standard solution is Stochastic Gradient Descent(SGD). It has the following form,

$$\vec{\theta}^{(k+1)} = \vec{\theta}^{(k)} - \epsilon_k \frac{\partial L}{\partial \vec{\theta}^{(k)}}, \tag{2.23}$$

where $\vec{\theta}^{(k)}$ is the parameters after iteration k and ϵ_k is called learning rate. The basic idea is to initialize parameters $\vec{\theta} := \vec{\theta}^{(0)}$ by certain approach and do the iteration in Eq. 2.23 until convergence. In each iteration, the parameters move a step to the direction where the gradient drops. The step is the ϵ in Eq. 2.23. It is usually the key to get a good results after training.

Nowadays, some algorithms work better than pure SGD, such as Adagrad and ADAM [11]. However, they are all based on SGD that cannot be guaranteed to get a global minimum.

2.2.3 Convolutional Neural Networks

Convolutional neural networks(CNNs) are a sort of neural network focusing mainly on grid structure data such as image and time series. A CNN can directly use image pixels as input and extract useful features automatically. Compared to the traditional methods, it avoids complicated feature extraction and is more generic. A complete CNN usually includes convolutional layers, pooling layers and fully connected layers.

Convolution

Convolutional layers operate on data arrays. It uses kernels to do convolutional operations on the input data, which is mathematically defined as

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n]K[i-m, j-n], \tag{2.24}$$

where I is the input data of a convolutional layer with size $m \times n$, K is a parameterized kernel, and s is the output of this layer. Similar to MLP, there's also an activation function after convolutional layers. Usually, the first convolutional layer takes the original data, for example, an image, and produces feature maps. Subsequent convolutional layers operate on them.

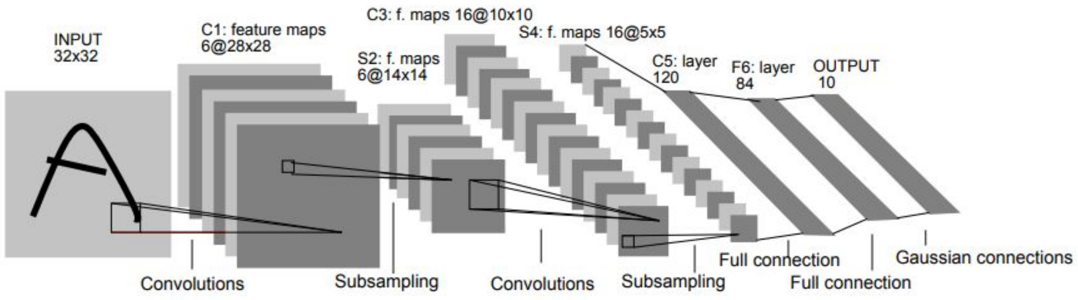


Figure 2.5: Overview of LeNet-5. Figure is from [16].

Pooling

A pooling layer reduces the size of features maps as there are usually too many values. It produces one value in the feature maps representing the neighbourhoods. Depending on how the representation is generated, the pooling function differs. A common choice is max pooling that returns the max value of a certain neighbourhood, keeping the most apparent feature. In addition, average pooling is also a commonly used choice.

Fully connected layers

Fully connected layers as a whole is actually an MLP. It takes charge of further processing the features extracted by the previous convolutional layers or pooling layers for classification or regression.

Fig. 2.5 shows a classic CNN [16], where we can see a convolutional layer takes an image as input and outputs feature maps that are later subsampled(in LeNet-5 it is average pooling). After another combination of convolutional layers and pooling is the fully connected layers that produce 10 outputs finally. In the past decades, many CNNs are proposed to deal with tasks on image data and get pretty good results such as AlexNet [13], ResNet, GVV, etc.

3 Neural Sparse Grids

In this chapter, we propose a new method, Neural Sparse Grids(NSG), where we have a Sparse Grids(SG) layer after Artificial Neural Networks(ANN). In Sec. 3.2, the basic concepts of NSG will be introduced, including the model structure and how to train it possibly. Afterwards, in Sec. 3.3 we will show how it deals with regression problems by applying it on an artificial and a real data set. Next is Sec. 3.4, where there is a discussion about how NSG deals with classification tasks, and we will show an example of image data. Finally, we finish this chapter by analyzing the results from Sec. 3.3 and 3.4.

3.1 Motivation

Artificial neural networks have achieved great success in the past years. It is applied in many fields such as computer vision, natural language processing and even scientific applications [21]. Since ANN-based models cannot easily be interpreted, they are still a black box for us. What's worse is that it sometimes produces adversarial examples. These incorrect predictions are even often with unknown biases [10].

A good method in scientific computing based on efficient space discretization schemes is Spare Grids, as we have introduced before. However, sparse grids and their variants cannot deal with high-dimensional data because the number of grid points needed increases much as the dimension increases. On the contrary, the ANN can easily tackle.

Therefore, we try to combine these two methods, Sparse Grids and Artificial Neural Networks, to take advantage of both of them.

3.2 Neural Sparse Grids

3.2.1 Standard Model Structure

In the following, we introduce the structure of Neural Sparse Grids(NSG). As the name Neural Sparse Grids indicates, it consists of 2 parts, Artificial Neural Network(ANN) and Sparse Grids(SG), where original input data is first reduced to lower-dimensional data by ANN, and the Sparse Grids subsequently approximates the function (see Fig. 3.1).

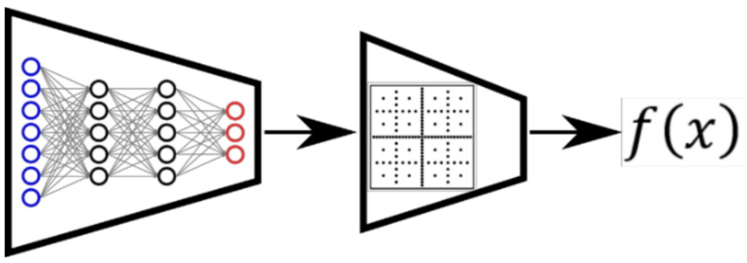


Figure 3.1: Concept of Neural Sparse Grids where a Sparse Grids layer is appended to Artificial Neural Networks

Formally, assuming we have data $\vec{x} \in \mathbf{X} \subset \Omega := \mathbb{R}^D$ and the domain of SG input is $\omega := [0,1]^d$ (it can also be $[-1,1]^d$, depending on the setup of SG). We have a component consisting of some ANN layers. As a whole, we denote these ANN layers as a function that outputs a vector, $\mathcal{N}_{\vec{\theta}} : \Omega \to \mathbb{R}^d$ with $\vec{\theta}$ being the parameters. Since ANN may outputs values in $\mathbb{R}$, a rescaling function is needed to restrict them in ω. It is denoted as $\rho : \mathbb{R}^d \to \omega$. After the scaling function is a SG layer with o sets of coefficients $\vec{u}_{\vec{c}} : \omega \to \mathbb{R}^o$ with $\vec{c}$ being coefficients. For simplicity we assume $o = 1$. If $o > 1$, we can set up other sets of SG coefficients. The above setting results in the following formula,

$$f(\vec{x}) \approx u(\rho(\mathcal{N}(\vec{x}))). \tag{3.1}$$

As for the rescaling function, we can choose $sigmoid$ function $\sigma = \frac{1}{1+\mathrm{e}^{-x}}$ for $\omega = [0,1]^d$ and $tanh$ function $tanh = \frac{\mathrm{e}^{2x}-1}{\mathrm{e}^{2x}+1}$ for $\omega = [-1,1]^d$. On the one hand they can map data from $\mathbb{R}$ to ω, which is exactly what we want. On the other hand, both functions are independent of the input. To simplify the formula, we integrate the rescaling function to ANN. The Eq. 3.1 becomes

$$f(\vec{x}) \approx u(\mathcal{N}(\vec{x})). \tag{3.2}$$

The vectors produced by $\vec{h} = \mathcal{N}(\vec{x})$ can be called embeddings as they are what we think the representations of input vectors.

Therefore, the whole process can be viewed as the ANN reduces the dimension of input data, and SG approximates the function afterwards. Theoretically, We can freely choose the structure of ANN and the configuration of SG only if the dimension matches. But we need to carefully select SG configurations because the number of grid points increases much as the dimension and level grow.

In NSG, we can also call the coefficients $\vec{c}$ parameters to align with the convention of ANN since it is also something we learn from the data.

3.2.2 Training

To train a model, it is necessary to know the forward pass. Assuming we have a defined ANN $\mathcal{N}(\vec{x})$ with a scaling function integrated to it. ANN produces d-dimensional embeddings $\vec{h}$, i.e. $\vec{h} = \mathcal{N}(\vec{x})$. In addition, we also have a defined SG with G grid points. No matter how we define these grid points, j^{th} grid point can be represented by $\vec{L_j}$ and $\vec{I_j}$ where $\vec{L_j}$ is a vector consisting of level of each dimension and $\vec{I_j}$ represent index of each dimension. And we use the standard hat function φ for SG. Thus, Eq. 3.2 is extended to Eq. 3.3. Fig. 3.2 shows an example of $d = 2$.

$$f(\vec{x}) \approx u(\mathcal{N}(\vec{x})) = u(\vec{h}) = \sum_{j}^{G} c_j \varphi_j(\vec{h}) = \sum_{j}^{G} c_j \prod_{p=1}^{d} \varphi(2^{L_{j,p}} h_p - I_{j,p}) \tag{3.3}$$

It can also be written in a matrix form. To start with, we first consider $o = 1$. Let $\vec{x}_i =$

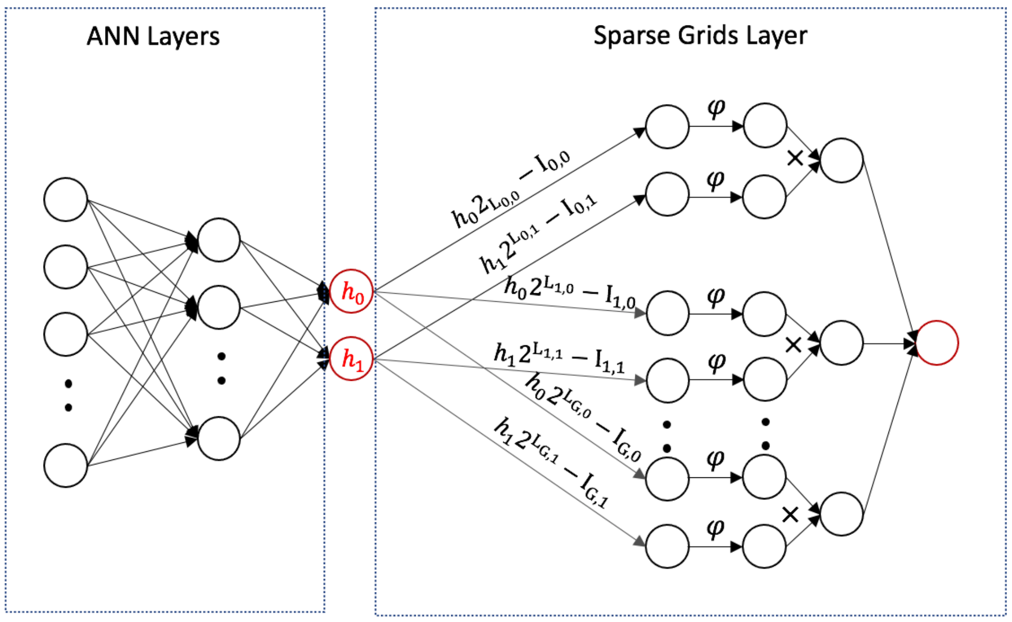

ANN Layers
Sparse Grids Layer
h_0
h_1
$h_0 2_{L_{0,0}} - I_{0,0}$
$h_1 2^{L_{0,1}} - I_{0,1}$
$h_0 2^{L_{1,0}} - I_{1,0}$
$h_1 2^{L_{1,1}} - I_{1,1}$
$h_0 2^{L_{G,0}} - I_{G,0}$
$h_1 2^{L_{G,1}} - I_{G,1}$
φ
$\times$

Figure 3.2: Forward pass of NSG

$X_i \in X \in \mathbb{R}^{N \times D}$ and its corresponding actual values $y_i \in Y \in \mathbb{R}^N$. And we know the ANN produces the embeddings $H \in [0, 1]^{N \times d}$. We construct a matrix $K \in \mathbb{R}^{N \times G}$, whose elements are defined as $K_{m,n} = \varphi_m(H_n)$ where H_n is the n^{th} point in H and φ_m is the m^{th} basis function of SG. The coefficients of SG are denoted as $C \in \mathbb{R}^G$. Therefore the matrix form is

$$KC \approx Y. \tag{3.4}$$

For $o > 1$, we have $Y \in \mathbb{R}^{N\times 0}$ and $C \in \mathbb{R}^{G\times o}$ while the rests remain the same.

Since ANN is part of NSG, it is unavoidable to use a Stochastic Gradient Descent(SGD) to optimize the parameters $\vec{\theta}$ of ANN. The coefficients $\vec{c}$ of SG can also be optimized along with them. But remember that the coefficients of SG are usually produced by solving a linear system. We can also choose to get $\vec{c}$ by solving a linear system using the embeddings $\vec{h}$ and the actual values Y.

Pure stochastic gradient descent

As is discussed above, a straightforward training method is optimizing both $\vec{\theta}$ and $\vec{c}$ using SGD or its variants.

Again, we first assume $o = 1$ for simplicity. The other cases are similar to this. The forward pass is what we have described above in Eq. 3.3. In order to do backward-propagation, we have to know the partial derivative w.r.t the parameters. Assuming the loss function is $L(\hat{y}, y)$, the partial derivative w.r.t. one parameter of SG c_j is

$$\frac{\partial L}{\partial c_j} = \frac{\partial L}{\partial u} \cdot \frac{\partial u}{\partial c_j} = \frac{\partial L}{\partial u} \cdot \prod_{p=1}^{d} \varphi(2^{L_{j,p}} h_p - I_{j,p}). \tag{3.5}$$

To get partial derivative w.r.t. parameters of ANN, we need to know $\frac{\partial u}{h_p}$ shown in Eq. 3.6, where h_p is the p^{th} dimension of $\vec{h}$

$$\frac{\partial L}{\partial h_p} = \frac{\partial L}{\partial u} \cdot \frac{\partial u}{\partial h_p} = \frac{\partial L}{\partial u} \cdot \sum_{j}^{G} c_j \prod_{k\neq p}^{d} \varphi(2^{l_{j_k}} h_k - i_{j_k}) \frac{\partial \varphi(2^{L_{j,p}} h_p - I_{j,p})}{\partial h_p} \tag{3.6}$$

By applying chain rule we can get partial derivative w.r.t. any parameter of ANN.

However, Eq. 3.5 and Eq. 3.6 indicate that for regular sparse grids the derivatives are 0 if points are on the boundary. If it appears, optimization will stop.

Get $\vec{c}$ by solving a linear system

Before we continue with a new training method, we first review how SG is solved. In fact, finding the best coefficients for SG is a least square problem

$$u = \operatorname*{argmin}_{u\in V} \left(\frac{1}{G} \sum_{i=1}^{G} (y_i - u(\vec{x_i}))^2 \right) \tag{3.7}$$

To avoid overfitting, we usually employ Tikhonov-style regularization [7] to enforce a certain smoothness. Since the direct sparse grid discretization scheme is employed, we can use the Euclidean norm of the vector surplusses.

$$u = \underset{u \in V}{\operatorname{argmin}} \left(\frac{1}{G} \sum_{i=1}^{G} (y_i - u(\vec{x_i}))^2 + \lambda \|\vec{C}\|_2^2 \right) \tag{3.8}$$

In matrix form, the problem becomes

$$C^* = \underset{C}{\operatorname{argmin}} \|KC - Y\|_2^2 + \lambda \|C\|_2^2 \tag{3.9}$$

Back to our NSG training, if the embeddings produced by $\mathcal{N}$ are available, we will be able to compute the kernel K and get the best coefficients by solving Eq. 3.9.

Fortunately, Eq 3.9 has a closed form solution. For $N \geq G$, the solution is,

$$C^* = (K^T K + \lambda I)^{-1} K^T Y \tag{3.10}$$

For $N < G$, the solution is,

$$C^* = K^T (K K^T + \lambda I)^{-1} Y \tag{3.11}$$

Thus, the forward pass can be described as Alg. 1 shows.

Algorithm 1: Optimizing $\vec{\theta}$ by SGD while computing $\vec{c}$ by solving a linear system

```
Result: Optimized $\vec{\theta}^*$ and $\vec{c}^*$
initialization;
k = 0 ;
while Not converged do
    $H^{(k)} = \mathcal{N}_{\theta^{(k-1)}}(X)$ ;
    Construct Matrix $K^{(k)}$ by $H^{(k)}$ ;
    Calculate $C^{(k)}$ by solving Eq. 3.9 ;
    $\hat{Y}^{(k)} = K^{(k)} C^{(k)}$ ;
    Update $\theta^{(k)}$ by SGD ;
    k = k + 1 ;
end
```

We can also merge the steps in forward pass and get the following equation for $N < G$

$$\hat{Y} = K K^T (K K^T + \lambda I)^{-1} Y. \tag{3.12}$$

Get $\vec{c}$ by solving a linear system iteratively

In fact, we can also solve the problem Eq. 3.9 by solving a linear system iteratively. There are many algorithms for these iterative methods, such as LSMR [6]. We can reformulate such a method as a function $g_{\lambda,t}(C, K, Y)$ where λ is the regularization term, and t is an integer representing the number of iterations for solving this linear system in each call of g. This function takes the old coefficients C, current kernel matrix K and actual values Y as input while returns new coefficients. It results in the following forward pass in t^{th} iteration,

$$\hat{Y} = K^{(t)} g(C^{(n-1)}, K^{(t)}, Y) \tag{3.13}$$

Knowing that function g is differentiable, we can do back propagation in each iteration. Alg. 2 shows the steps during a complete iteration.

Algorithm 2: Optimizing $\vec{\theta}$ by SGD while computing $\vec{c}$ by iteratively solving a linear system

```
Result: Optimized $\vec{\theta}^*$ and $\vec{c}^*$
initialization;
k = 0 ;
while Not converged do
    $H^{(k)} = \mathcal{N}_{\theta^{(k-1)}}(X)$ ;
    Construct Matrix $K^{(k)}$ ;
    $C^{(k)} = g(C^{(k-1)}, K^{(t)}, Y)$
    $\hat{Y} = K^{(t)} C^{(k)}$ ;
    Update $\theta^{(k)}$ by SGD ;
    k = k + 1 ;
end
```

3.2.3 Integration with Other Architectures

We can integrate this Sparse Grids layer into other architectures. On the one hand, like what we have proposed before, it can be appended to a classic ANN architecture such as MLP, CNN and GNN. The Sparse Grids layer can act as a final regressor or classifier on the embeddings in the latent space. On the other hand, we can also put it inside other architectures, and it acts as a component of that structure like attention mechanism.

3.3 Regression

Having considered Neural Sparse Grids structure, it is straightforward to see how Neural Sparse Grids can solve regression problems. What we aim at is to approximate an underlying function everywhere as good as possible.

In the following, we firstly introduce the regression problem formally; secondly, we discuss how Neural Sparse Grids can work on an artificial dataset, swissroll, where different training methods are applied. Finally, we show the results of a real problem in astrophysics, stellar atmospheric parameter estimation using spectra data from SDSS [1].

3.3.1 Neural Sparse Grids Based Regression

Suppose we have a set of m observations,

$$S = \{(\vec{x_i}, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^m \tag{3.14}$$

Assuming they have been directly sampled from a smooth, continuous multivariate function $g : \Omega \to \mathbb{R}^d$, with certain unknown noise. The object is to construct a function u, which approximates f everywhere as well as possible.

$$f = \text{argmin}_f \frac{1}{m} \sum_{i=1}^{m} (y_i - u(\vec{x_i}))^2 \tag{3.15}$$

Specifically, for a Neural Sparse Grids,

$$f = u_{\vec{\theta},\vec{c}} = \text{argmin}_{\vec{\theta},\vec{c}} \frac{1}{m} \sum_{i=1}^{m} (y_i - u_{\vec{c}}(\mathcal{N}_{\vec{\theta}}(\vec{x_i})))^2 \tag{3.16}$$

where, $\mathcal{N}$ is an ANN with a scaling function appended to it, and u is a Sparse Grids. They are parameterized by $\vec{\theta}$ and $\vec{c}$ respectively. The goal is to find the best $\vec{\theta}^*$ and $\vec{c}^*$. Remember in SG it is actually a least square problem that is convex, but in NSG it is not.

Thus, for this optimization problem, we can use a Stochastic Gradient Descent(SGD) based method with Mean Square Error Loss(MSE Loss) to optimize both $\vec{\theta}$ and $\vec{c}$ although we cannot ensure to get a global minimum. Also, learning ANN parameters $\vec{\theta}$ by SGD while calculating SG parameters $\vec{c}$ by solving a linear system(either directly or iteratively) as introduced in Sec. 3.2.2 also works. Because, in this task, we know exactly what after Sparse Grids layer is.

To measure the performance, we can use Mean Square Error(MSE) or Mean Absolute Error(MAE) as is shown in the following,

$$\epsilon_{MSE} = \sum_{i=1}^{N} (\hat{y_i} - y_i)^2 \tag{3.17}$$

$$\epsilon_{MAE} = \sum_{i=1}^{N} |\hat{y_i} - y_i| \tag{3.18}$$

where we have predicted values $\{\hat{y_i}\}_{i=1}^N$ and actual values $\{y_i\}_{i=1}^N$.

3.3.2 Regression on Swiss Roll

As a demonstration, we start with an artificial dataset embedded in a high dimensional ambient space, swissroll. An artificial dataset helps analyze the results and the performance better as we know how it is generated. We use all three training methods introduced in the previous sections.

Swiss roll dataset

Specifically, a swissroll dataset is sampled in the following steps[8]: first, we sample $t \sim U(\frac{3\pi}{2}, \frac{9\pi}{2}), h \sim U(0,3)$. A p-dimensional point x is then generated by

$$x_1 = t\cos(t) + \delta_1, x_2 = h + \delta_2, x_3 = t\sin(t) + \delta_3, x_i = \delta_i, i \geq 4, \delta_1, ..., \delta_4 \sim N(0, \tau^2)$$

Corresponding response is

$$y = \sin(5\pi t) + h^2 + \epsilon_i, \epsilon \sim N(0, \sigma^2)$$

Here we choose dimension three because it is easier for visualization and further analysis. Fig. 3.3 shows the swissroll data and its projection in a 2d space.

In our experiment, 8000 points are generated with noise equal to 0.1. The whole data set is divided into training(60%), validation(20%) and test(20%) sets. Models are evaluated finally on the test that we do not touch during the training period.

In addition, before modelling the data, we normalize them to ensure they are in $[0, 1]$ in each dimension. For data in dimension d, the following formula is applied,

$$x^d_{norm} = \frac{x^d - x^d_{min}}{x^d_{max} - x^d_{min}} \tag{3.19}$$

Finally, we have the data prepare,

$$S = \{(\vec{x_i}, y_i) \in [0,1]^3 \times \mathbb{R}\}_{i=1}^{8000} \tag{3.20}$$

Model overview and configurations

Now that the data is prepared, we start to configure the Neural Sparse Grids model. Firstly, We use an ANN with layers $(3, 25, 25, 2)$ as we have input dimension 3, and we know from the generation process that the data is in a $2d$ manifold. The experiment has shown such an ANN is able to encode the data to a $2d$ space. Then, a $sigmoid$ function is appended to the ANN, acting as a rescaling function.

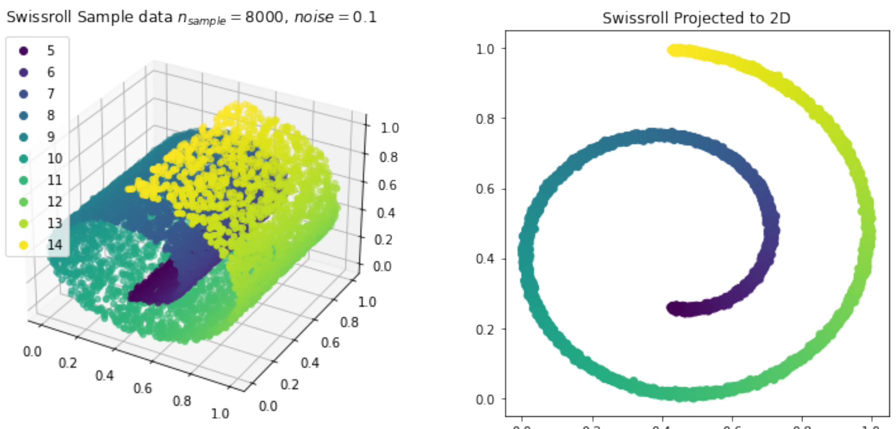


Figure 3.3: The left figure shows Swissroll dataset generated by noise equal to 0.1, containing 8000 data points where the color represents the response of each point. The right figure shows their projections to 2D space.

Next, a level 6 regular Sparse Grids layer getting 2 dimensional is defined after ANN. Since it is a regression task, the output dimension of Sparse Grids is 1. Fig. 3.4 shows the model structure.

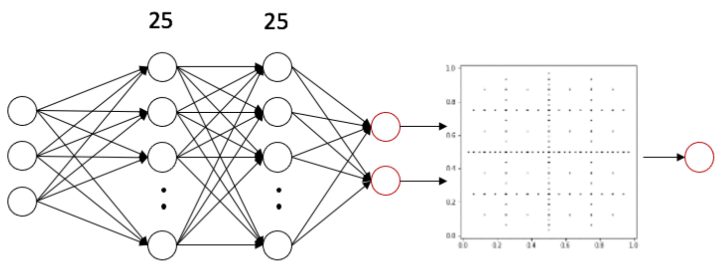
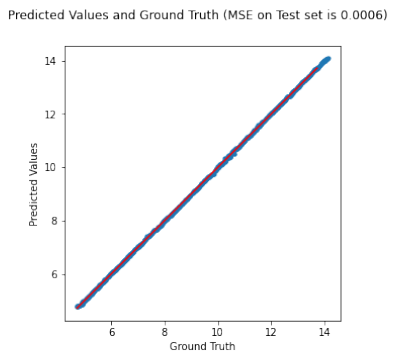


Figure 3.4: Neural Sparse Grids model for swiss roll dataset. The ANN with layers being(3, 25, 25, 2) reduces the dimension to 2 while the level 6 SG acts as a regressor

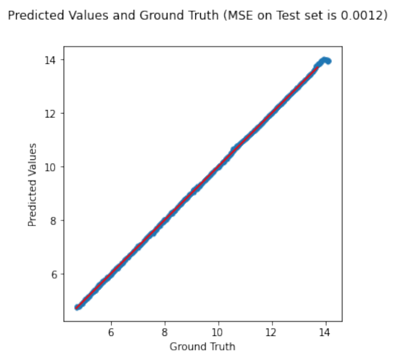
In the following, we will train the model in 3 methods and measure them by Mean Square Error(MSE).

Trained by SGD

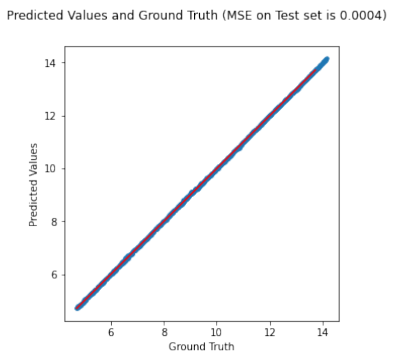
Firstly, we train the NSG by a pure SGD method. An important thing of this method is the learning rate. Tab. 3.1 shows the MSE on test set by using different learning rate.



(a) Trained by SGD



(b) Trained by LS



(c) Trained by ILS

Figure 3.5: Predicted Values and Actual Values

These models are trained for 200 epochs. As it is often the case that a small learning rate ”moves” too slowly and requires many updates and a large learning rate causes drastic updates leading to divergent behavior, we need a learning rate of a proper value. In this case, we choose 0.01 as it has the best result.

LR	MSE	LR	MSE
10^{-5}	0.6041	5×10^{-5}	0.9450
10^{-4}	0.1530	5×10^{-4}	0.0089
10^{-3}	0.0056	5×10^{-3}	0.0021
10^{-2}	0.0006	5×10^{-2}	0.0040
10^{-1}	7.38	5×10^{-1}	7.04

Table 3.1: MSE by different training rate

We take the model trained by learning rate 0.01 as an example. Fig. 3.5a shows that the predicted value is very close to the actual value. These points almost overlap with the line $y = x$, meaning the model perfectly fits the data. In fact, the MSE on the test set is only 6×10^{-4}.

Despite low MSE, if we look at the embedding in $2d$ latent space as Fig. 3.7 shows, we will find that the shape of embeddings is very irregular, even with some very sharp turning. Moreover, it takes only a very small proportion of the latent space and is centered around the centroid. It leads to a problem that the number of activated grids points is way less than the total number of grids points. It is not a coincidence. If we train it several times, we will always get small embeddings, although they may have different shapes.

Fig. 3.6 shows the Sparse Grids function in the latent space. When the embeddings are located in $[0.5, 0.55]^2$, the function looks pretty nice and makes sense. However, if we look at the whole latent space, the part other than $[0.5, 0.55]^2$ looks chaotic. If we got

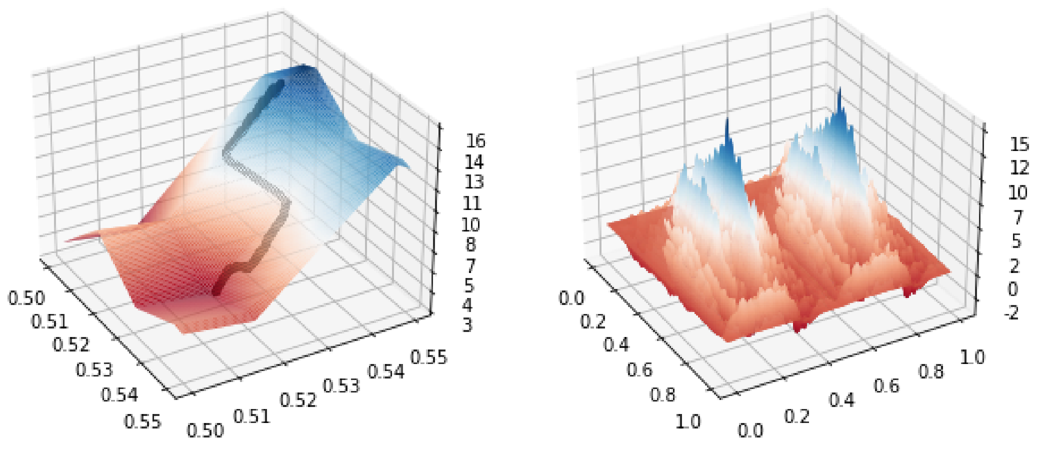


Figure 3.6: Sparse Grids in the latent space. x, y axis means the coordinate in latent space and the height represent the response values. The left figure takes the part $[0.5, 0.55]^2$ where the embeddings are located at while the right figure shows Sparse Grids in the whole latent space. In the left figure, the embeddings are marked in black.

embeddings from $[0,1]^2 \setminus [0.5, 0.55]^2$, the estimated response might may meaningless. It is even possible to get values less than zero, which in this dataset do not appear.

Besides, it is also meaningful to see how Sparse Grids level or the number of grid points affects the result. For Sparse Grids, typically, the performance increases as the level or the number of grid points increases. However, the experiments show it is not necessarily the case for NSG. The first row of Tab. 3.2 shows the MSE does not change much as the level increases. Sparse Grids of level 8 even perform worse while level 6 produces the best result.

	level 4	level 5	level 6	level 7	level 8
SGD	0.0013	0.0024	0.0006	0.0012	0.0039
LS	0.0088	0.0039	0.0012	0.0021	0.0052
ILS	0.0045	0.0025	0.0004	0.0009	0.0033

Table 3.2: MSE of models with different sparse grids levels trained by different methods

Train $\vec{c}$ by solving a linear system

As is discussed in the previous section, we can also learn the parameters of Sparse Grids by solving a linear system while learning the parameters of ANN by Stochastic Gradient

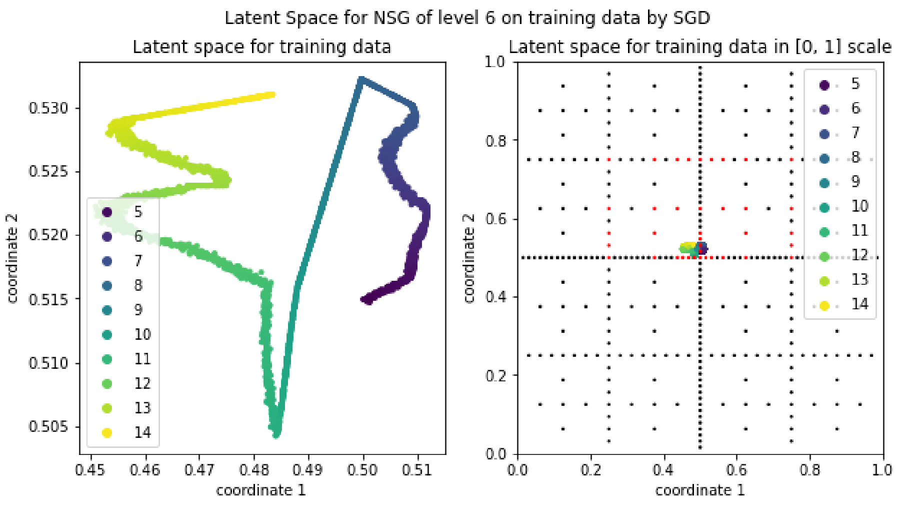


Figure 3.7: Embeddings in latent Space trained by SGD

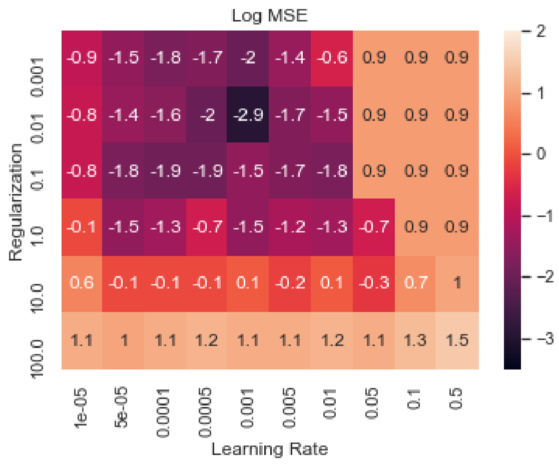
Descent. We call this method LS in this thesis. Thus, we have to find a good learning rate for ANN as well as a suitable regularization term for Sparse Grids. In the experiment, we test the combination of candidates of both learning rate and a regularization term.

Fig. 3.8a shows the choices and their corresponding $\log_{10}$ MSE on test set where the darker color indicates a lower MSE. It is pretty obvious that a learning rate of 10^{-3} and regularization if 10^{-2} are the best among them. Other good choices are centered around it, while those that are more away from it leads to worse results.

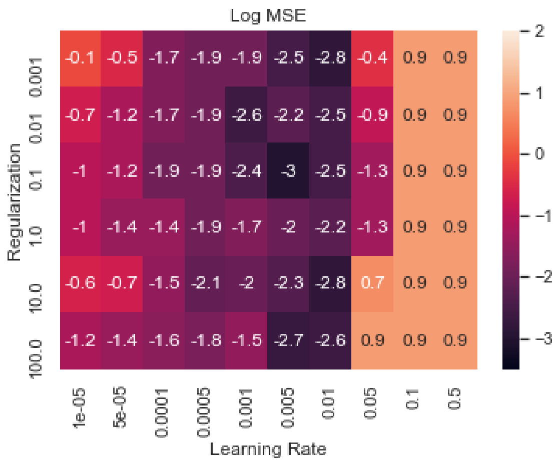
Using these two selected values, we trained a Neural Sparse Grids model, which results in 1.2×10^{-4} MSE. Fig. 3.5b shows that predicted values are very close to actual values.

We are also interested in the embeddings. Fig. 3.9 shows the embeddings of training data in latent space. Similar to the pure SGD method, we again get a very small group of points centered around the centroid, and over half of the grid points are not activated. They do not contribute to the final results, whatever their coefficients are. Nevertheless, in the meanwhile, the model does the regression task very well.

We also did the same experiment for different Sparse Grids levels. Fig. 3.5b provides the test results of them, where the small levels and the large levels do not perform very well compared to level 6. It is pretty different from pure Sparse Grids. The following section discusses this phenomenon in detail.



(a) Trained by LS



(b) Trained by ILS

Figure 3.8: $\log_{10}$ MSE by different training rate and regularization

Train $\vec{c}$ by solving a linear system iteratively

The last training method we will apply is learning parameters of Sparse Grids by solving a linear system iteratively while learning the ANN by SGD. We call this method ILS. In order to train an NSG in such a way, we need to determine a learning rate and a regularization term.

In Fig. 3.8b are the choices of learning rate and regularization term as well as their results on test set where the best choice is learning rate 5^{-3} and regularization term 10^{-1}. However, unlike solving a linear system directly, the change of regularization term does not seem to have much effect on the result with a learning rate being fixed.

Nevertheless, we choose the pair with the best result and get MSE 4×10^{-4} on the test set. Fig. 3.5c shows that the predicted values are almost the same as the actual values. Not surprisingly, the embeddings in latent space is again still very small as Fig. 3.10 shows.

Similar to the previous two training methods, higher level and lower level of Sparse grids do not perform as well as a middle level of Sparse Grids, as the Fig 3.2 shows. The best level, in this case, seems to be 6.

Summary of swissroll example

In this section, we have shown how NSG can be applied to the swissroll example. Three training methods are considered, and the results show that all of them are able to achieve good results. We will compare these training methods in detail later. In the meanwhile,

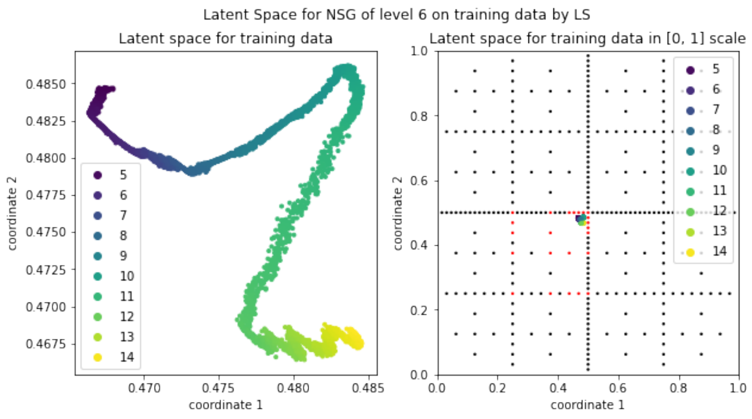


Figure 3.9: Embeddings in latent Space trained by LS

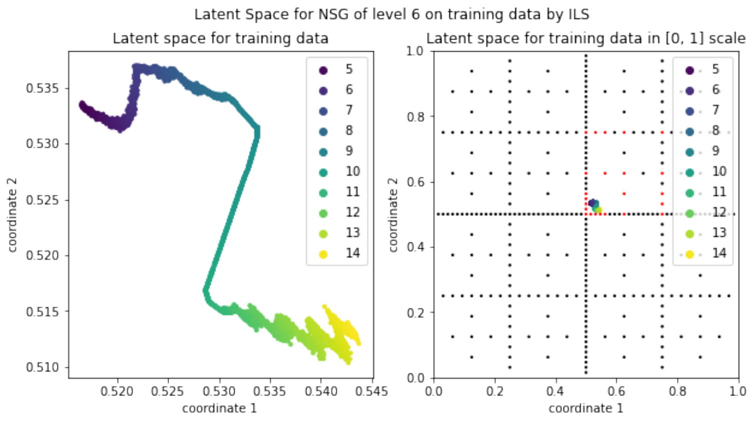


Figure 3.10: Embeddings in latent Space trained by ILS

there is the phenomenon of small embeddings, although the final results are good. Another point we have noted is that the increase of SG level does not always improve the performance. All these things will be discussed again in the later section.

3.3.3 Regression on Stellar Spectra

Having applied NSG on an artificial dataset, we now consider a real-world data set in astronomy.

Stellar atmostpheric parameters estimation

Here we briefly introduce stellar atmospheric parameter estimation. In astronomy, it is important and necessary to derive stellar parameters from its spectral data. There are a lot of astronomical observation instruments developed. The Sloan Digital Sky Survey(SDSS) is one of them whose sub-project Sloan Extension for Galactic Understanding and Exploration (SEGUE) provides a large amount of optical stellar spectra. These parameters usually include T_{eff}, $\log g$ and $[Fe/H]$. The term "parameters" here is an astronomical concept instead of parameters in the machine learning field.

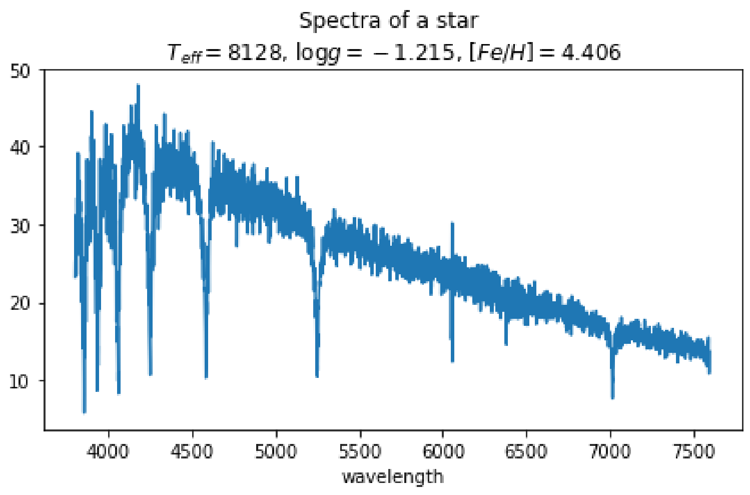


Figure 3.11: An example of stellar spectra data

A variety of machine-learning methods are applied to do this regression task, such as nearest neighbor (NN), artificial neural networks, and support vector machines(SVM). Some feature extraction-based methods are also used, such as wavelet transforms(WTs) and principle-component analysis(PCA). Since it is a very high-dimensional problem over 3000 dimensions, Sparse Grids cannot directly be applied. However, Neural Sparse Grids can do it as it has ANN to reduce the dimensionality. We only consider one of these three parameters for simplicity, which is $[Fe/H]$.

Data preparation

We sampled a subset containing 50000 of the original dataset. Reducing the data size helps reduce the complexity of training a model. And the results of a subset are very similar to the full dataset, as is stated in this paper [17]. Fig. 3.11 is an example from it that has atmospheric parameters $T_{eff} = 8128$, $\log g = 4.406$ and $[Fe/H] = -1.215$. The input data is also normalized by l_2 normalization. For each spectra $x = (x_1, ..., x_l)^T$, the normalized result $\bar{x}$ is thus $\bar{x} = \frac{x}{\|x\|_2} = \frac{x}{\sqrt{\sum_{i=1}^{l} x_i^2}}$. It results in the following expression of data set

$$S = \{(\vec{x_i}, y_i) \in [0,1]^{3800} \times \mathbb{R}, \|\vec{x_i}\|_2 = 1\}_{i=1}^{50000} \tag{3.21}$$

where $\vec{x_i}$ is the normalized spectra and y_i is $[Fe/H]$.

Whats's more, to develop and measure a model, we also need to divide this set into a training, validation, and test set, with ratios being 70%, 15%, and 15% respectively.

Model structure and measurement

Next, we introduce the model structure and some settings. As is shown in Fig. 3.12, we use an ANN with layers $(3800, 1000, 40, 3)$ and choose to use 3-dimensional latent space as input of Sparse Grids because, in some papers, this spectra data is reduced to 3 dimensions[18]. And we choose regular Sparse Grids of level 6.

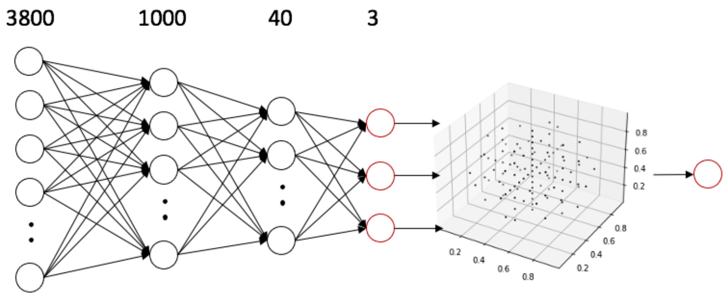


Figure 3.12: Neural Sparse Grids model for Stellar Parameter Estimation

Since it is a regression problem, we can choose any of the three training methods we introduced above. Here we train the Neural Sparse grids model by ADAM.

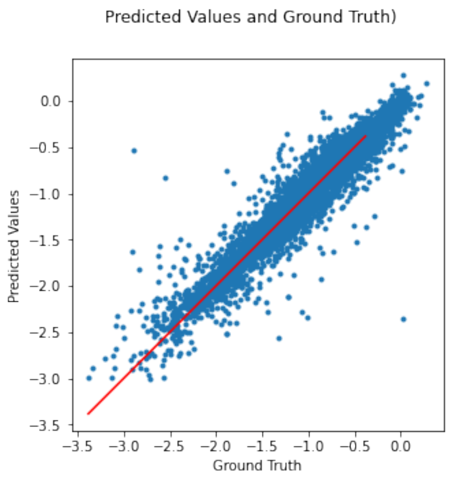
To measure the performance, as some papers use Mean Absolute Error(MAE, Eq. 3.18) we choose both MSE and MAE.

Results and discussion

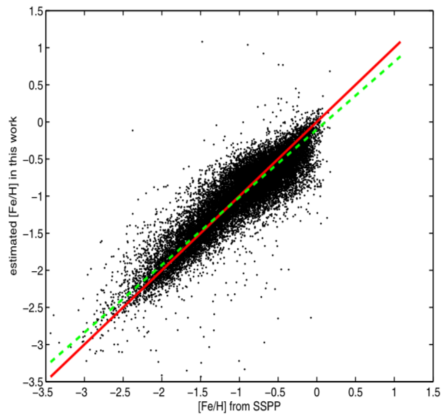
Similar to the swissroll example, we firstly try different learning rates and choose the best one. Tab. 3.3 shows the evaluation of models trained by different options. From the table we know neither a large learning rate nor a small learning rate brings a good result. In this case, 5×10^{-4} is the best in both measurements.

learning rate	MAE	MSE
5×10^{-5}	0.1108	0.0289
1×10^{-4}	0.1248	0.0307
5×10^{-4}	0.1104	0.0269
1×10^{-3}	0.1147	0.0288
5×10^{-3}	0.1219	0.0321

Table 3.3: Performance of NSG trained by different learning rate



(a) NSG results



(b) LI[17] method results. This figure is from the corresponding paper.

Figure 3.13: Predicted values VS actual values

The MAE and MSE of the model on test set is 0.1104 and 0.0269 respectively. Fig. 3.13a shows the predicted values and actual values of test set where points on the red line means predicted value equals actual value exactly.

In the previous section, we got embeddings with a very small diameter. This phenomenon also appears in this real dataset. As we can see from Fig. 3.14 that the embeddings in 3d latent space are a very small cluster around the centroid. The diameter is only about 0.08 in $[0, 1]$.

In the swissroll case, we found that the increase of dimension does not always help to improve the performance. In this case, it is similar to the test results of different levels, as is shown in Tab. 3.4. For benchmark purpose, Tab. 3.5 lists the results from different models where σ^* is the standard deviation of the difference used in paper[3]. We can see that Neural Sparse Grids performs well among these models but still worse than the best among them. Another thing is that these models are tested on the same data release(DR),

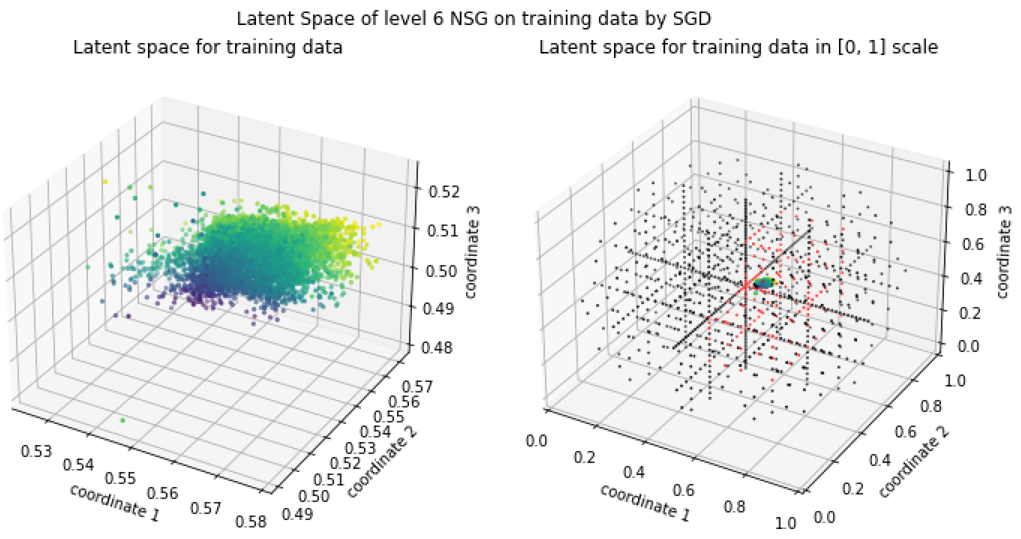


Figure 3.14: Embeddings in latent space. Grid points of Sparse Grids are shown on the right figure while the red ones are activated.

level	4	5	6	7
MSE	0.0329	0.0294	0.0269	0.0301

Table 3.4: MSE of different Sparse Grids levels

so it is hard to make a conclusion here.

Evaluation	FNN[17]	LI[17]	RF[17]	NSG	Autoencoder[23]	GPR[3]	KR[3]
MAE	0.179565	0.1820	0.204248	0.1104	0.1770	-	-
σ^*[3]	0.34	-	-	0.1715	-	0.24	0.046

Table 3.5: Comparisons of regression methods. Some values are missing because some papers do not provide them.

3.4 Classification

This section discusses classification problems with both artificial and real data sets. We take it as one type of predictive modeling that tries to assign a "correct" class label $k \in K$ to points in a specific feature space.

3.4.1 Neural Sparse Grids Based Classification

Formally, suppose we have a set of data points with its corresponding labels.

$$S := \{(\vec{x}_i, y_i) \in \Omega \times K\}_{i=1}^{m} \tag{3.22}$$

where $\vec{x}_i$ are data points in feature space Ω and $y_i \in K$ are their corresponding class label. Usually, we have such a data set available, either obtained by the data collection or manually labeled. The task is to predict a class label k that is unknown to us for any given point $\vec{x}$.

Sparse grids can deal with classification problems either by doing it as a regression problem(encoding the class labels as a one-hot vector) or by density estimation [19]. ANN usually tackles this task by using a softmax function after the output in order to produce probabilities and ensure the sum of probabilities of all classes equal to 1. For Neural Sparse Grids models, we have three options:

- use the ANN style that we add a softmax function after Sparse Grids to get probabilities;
- take it as a regression problem that we force the sparse grids layer to directly output a one-hot vector to get the predicted class label;
- let sparse grids do density estimation and then predict the target class label.

In this thesis, we only investigate the first option in an ANN style and train it by SGD based optimizer. The problem thus becomes,

$$f(\vec{x}) \approx softmax(u(\mathcal{N}(\vec{x}))) \tag{3.23}$$

As for the performance measurement, there are many evaluation metrics available for the classification task, such as accuracy, precision, recall, and F1 score, etc. Which metric to use depends on the specific scenario. For example, accuracy is good for well-balanced and not skewed cases, while recall is often used to capture as many positives as possible.

We can calculate by the following formula,

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.24}$$

where TP, FP, TN and FN stand for true positive, false positive, true negative and false negative respectively.

3.4.2 Classification on a Two Moons Variant

In the following, we introduce how Neural Sparse Grids can be used for classification problems by an artificial data set. We expand the original data set to a higher dimension to fit our topic better.

Two moons and its variant

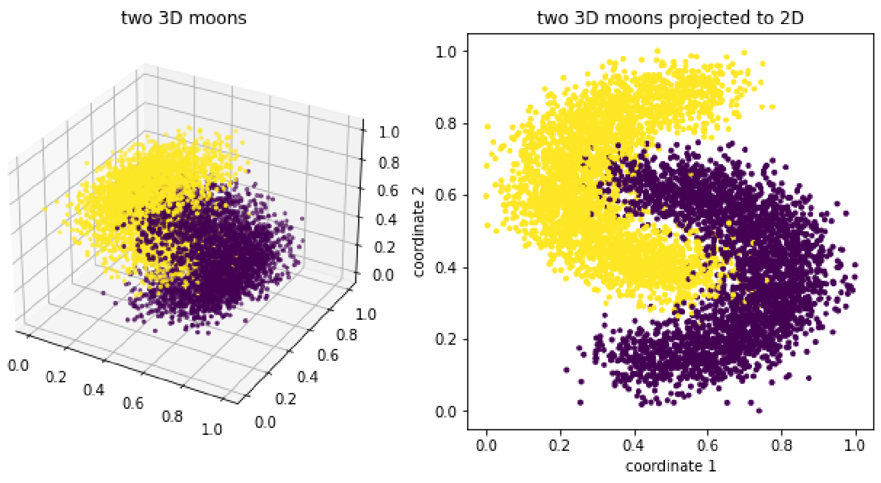


Figure 3.15: Sampled two moons in 3D. The left figure is points of the extended two moons. The left figure is their projection to 2D, which is almost the same as the traditional artificial dataset two moons.

The two moon data set is a prevalent artificial data set for classification algorithms, consisting of two interleaving half-circles. Since our topic is for high-dimensional data., we expand this 2d data set to a higher dimension using the following routine. An advantage of using such a variant is that we know it is in a 2-dimensional manifold.

Suppose we have a set of traditional two moons points. For a point $\vec{x} = (x_1, x_2)^T$, we expand an additional dimension $z \sim U(0,1)$ and keep the label unchanged. Fig. 3.15 shows the expanded data set.

We sample 8000 points from this routine with noise being 0.1. As there are 4000 points in each class, this is a pretty balanced set. Before we start modelling, we normalize each dimension to $[0,1]$ and divide it into training(70%), validation(15%) and test set(15%). Now we have the data set,

$$S := \{(\vec{x_i}, y_i) \in [0,1]^3 \times \{0,1\}\}_{i=1}^{8000} \tag{3.25}$$

Model overview and settings

Fig. 3.16 shows the model structure. Firstly, we have an ANN with layers(3, 15, 15, 2) because such an ANN is able to produce suitable embeddings in 2d. An other reason is that we know the data set is expanded from 2 dimensions. Then, we have a 2d regular sparse grids of level 5 that outputs two values. Lastly, as a classification task, we do $softmax$ on these two values and get two probabilities for each class.

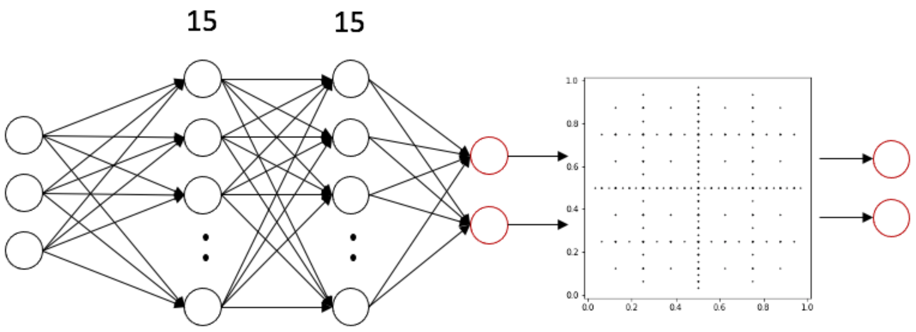


Figure 3.16: Neural Sparse Grids model for two moons variant classification

Having considered the balanced number of points in each class, we choose to use accuracy to evaluate the performance of this task. We use SGD to train the model.

Results and discussion

Tab. 3.6 shows the different training rate and their test results. Here we choose 5×10^{-3} as it makes the best accuracy 97%.

Learning rate	Accuracy	Learning rate	Accuracy
10^{-5}	0.8830	5×10^{-5}	0.9300
10^{-4}	0.9650	5×10^{-4}	0.9685
10^{-3}	0.9680	5×10^{-3}	0.9700
10^{-2}	0.9590	5×10^{-2}	0.9585
10^{-1}	0.8960	5×10^{-1}	0.4990

Table 3.6: Accuracy by different learning rate

Since we are interested in the embeddings, we again look into the latent space. The embeddings in latent space are shown in Fig. 3.17. Unlike the swissroll example, the embeddings are much larger and more grid points are activated. However, if we look at the embeddings in $[0, 1]^2$ scale, it will still be small.

Fig. 3.18 shows the probability of embeddings being class 0 predicted by our model. If we only look at the left figure whose domain is $[0.4, 0.55]^2$, we will find the Sparse Grids produce very sensible results. The test points in black fit the shape pretty well. However, in other regions, the results are not in order, which may lead to a potential problem of robustness. We also compared the different levels of Sparse Grids as Tab. 3.7 shows. Simi-

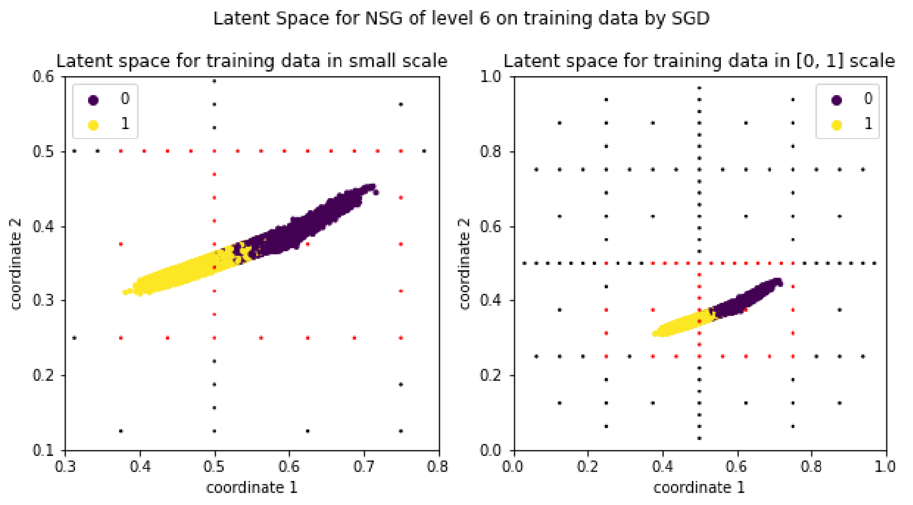


Figure 3.17: Embeddings in latent space. Dark dots are embeddings of class 0 while light ones are those of class 1. The left figures is in a small visualization scale.

lar to the previous examples, a mid-sized level of Sparse Grids leads to the best accuracy. Here the best levels are 5 and 6.

Level	3	4	5	6	7	8
Accuracy	0.9675	0.9660	0.9700	0.9700	0.9680	0.9670

Table 3.7: Accuracy of different Sparse Grids levels on two moons variate. Level 5 and 6 produces the best accuracy.

3.4.3 Classification on MNIST

This section discusses how an NSG deals with classification on the real-world data set MNIST. Usually, Sparse Grids are not good at processing image data due to the dimension. But for an NSG, if we use CNN as the first component that produces feature maps after pooling layer or convolutional layer, it will be able to do specific image-based tasks. We will construct such an NSG and see how it performs.

MNIST data set

We consider the handwritten digit dataset MNIST, which consists of 60000 grey-scaled images of handwritten digits for training and 10000 for the test. Each image is stored in a 28×28 matrix with values ranging from 0 to 255. Fig. 3.19 is some images from this data

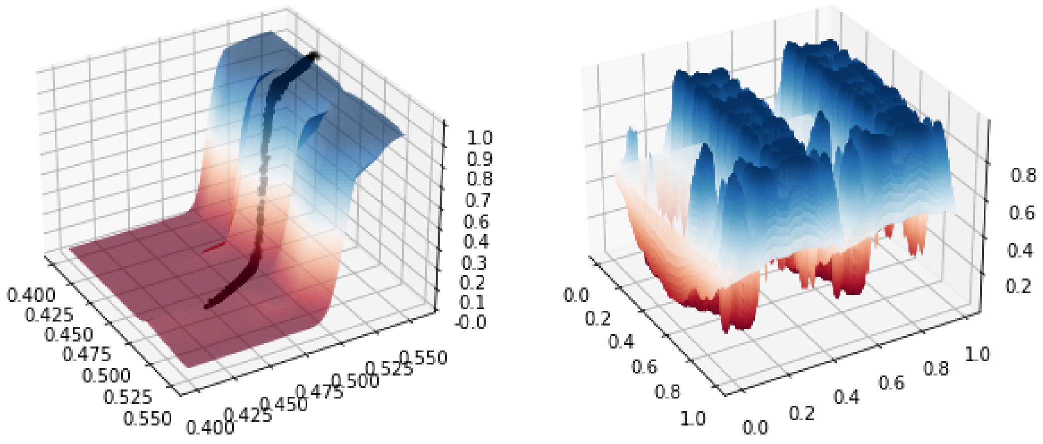


Figure 3.18: Probability of class 0 over the latent space. x, y axis means the coordinates in latent space and the height represents the probability of a certain embedding being class 0. The left figure takes the part $[0.4, 0.55]^2$ where the embeddings of training data are located while the right figure shows that over the whole latent space. In the left figure, the embeddings of test data are marked in black

set. It is a pretty classical classification task that has been considered in many papers.

Same as the previous examples, we normalize the data and ensure the values in each dimension are in the range $[0, 1]$ using Eq. 3.19. If we flatten the 2-d array data to 1-d(but we keep the 2d form as it is useful for CNN), we will get the data set S,

$$S := \{(\vec{x_i}, y_i) \in [0,1]^{784} \times K\}_{i=1}^{60000} \tag{3.26}$$

where $\vec{x_i}$ is a vectorized and normalized image and $y_i \in K = \{0, 1, ..., 9\}$ represents the label of image $\vec{x_i}$.

Model overview and settings

Since Convolutional Neural Network(CNN)[16] gives outstanding performance in image classification, we use a classic CNN structure as ANN part of NSG and append it with a Sparse Grids layer. As we have introduced in Ch. 2, the last component of a CNN is fully connected layers. We can therefore append a Sparse Grids layer after it. Fig. 3.20 shows the overview of such an NSG.

We choose to use ResNet18[9] as the ANN part of our model. On the one hand, it is specially designed to deal with image-based problems. On the other hand, this architecture does not have so many parameters(about 0.3 Million) as the other classic CNN models. But we have to adapt the structure to Sparse Grids:

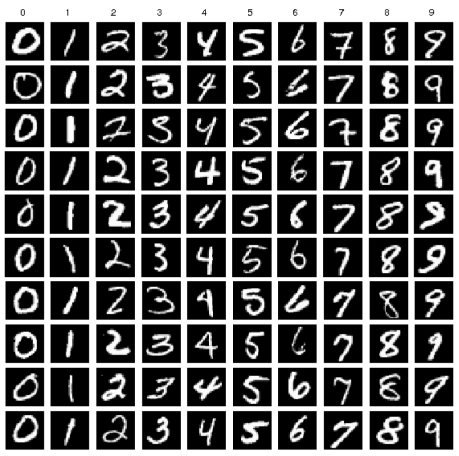


Figure 3.19: Example images in MNIST

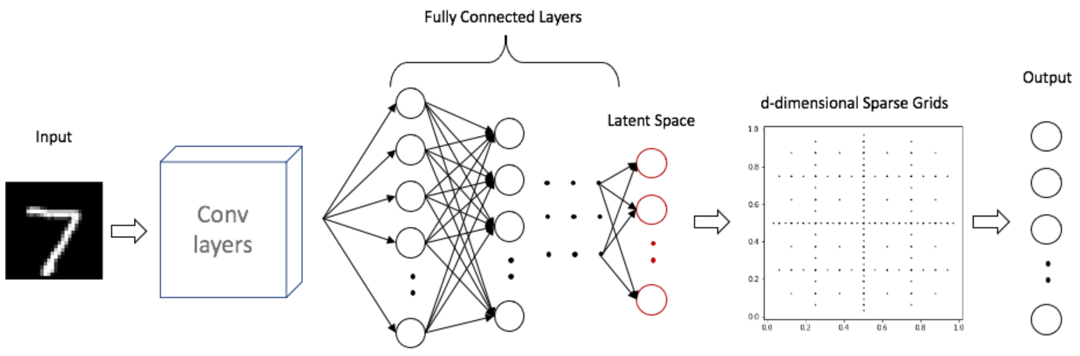


Figure 3.20: Overview of NSG integrated with CNN

- change the input channel to 1 as MNIST is a gray-scaled data set
- change the number of its output to our desired latent dimension, which in MNIST case is 8. In paper [14], MNIST is represented in a 6-dimensional space and get good results.
- we append a $sigmoid$ function after the CNN output as a re-scaling function to ensure the later Sparse Grids gets input values in $[0, 1]$

And we use regular Sparse Grids of level 4 because the task of Sparse Grids for the classification problems is to find the boundaries between different classes, which should not require a high level of discretization.

Since there are 10 classes, the Sparse Grids outputs 10 values, and we use $softmax$ to calculate the probability of each class.

We expect that the CNN can generate very good embeddings of images in latent space so that Sparse Grids can successfully do the classification. Here we use ADAM[11] to train the model as it performs very well.

Loss stuck and boundary treatment

However, in practice, the cross-entropy loss is sometimes stuck at 2.3. In fact, being 2.3 means the probability for each class is 10%, which equals a random guess. When it happens, the model actually learns nothing. In experiments, when it is stuck, the Sparse Grids outputs a very small number of logits that are almost zeros, meaning we get $\vec{c} = \vec{0}$.

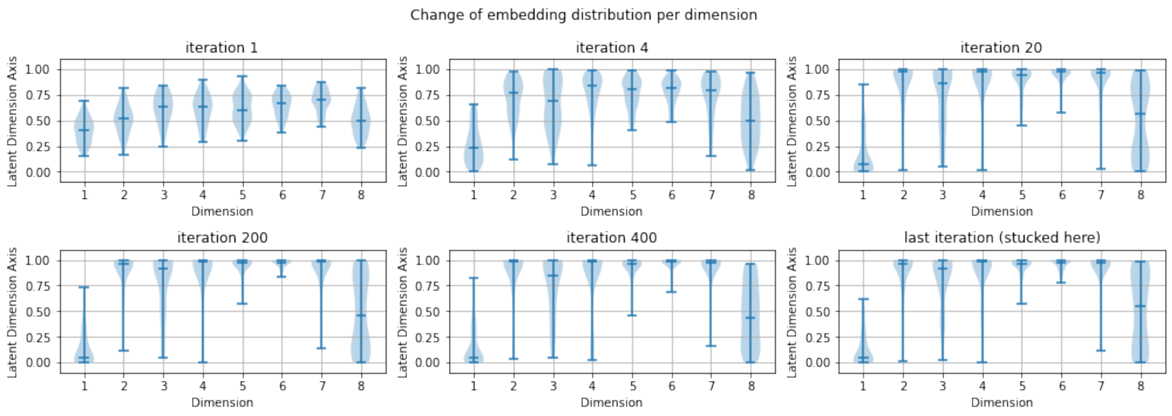


Figure 3.21: Change of dimension-wise embedding distribution in latent space

Fig. 3.21 shows how the dimension-wise distribution of embeddings change as is being optimized. We can see that it starts with "gaussian-like" distributions in the first iteration. Then, the embeddings are "pushed" to the corner as we can see most embeddings stay near 0 or 1 in most dimensions. However, our regular sparse grids do not treat the boundary especially. We have discussed in Sec. 3.2.2, the model may stop training when embeddings are on the boundary.

As we introduced in Ch. 2, we can consider introducing level 0 hierarchical basis to capture the boundary, but it leads to much more grid points. Another choice used often is to have extended level 1 that captures the boundary. It has the sparse grid space $V_n^{B(1)}$. Both approaches introduce many new grid points to our model, which is not what we want. We can also reserve some margin to the boundary to get a certain degree of boundary treatment without adding new grid points. For example, to reserve a margin of width 0.1, we can do it by $\bar{h} = 0.8h + 0.1$

Experiments show that doing the boundary treatment can solve the "loss stuck" issue. But it is essential to know how the different boundary treatment affects the accuracy. Tab.

Boundary Treatment	Reserve Margin	$V_n^{B(1)}$
Accuracy	99.18%	99.04%

Table 3.8: Accuracy of different boundary treatment

3.8 shows the accuracy of different boundaries for sparse grids level 3 and dimension 7. We reduced both level and dimension because sparse grids $V_n^{B(1)}$ brings too many grid points. From the table, we can conclude that adding grid points to treat boundary does not necessarily increase the performance. However, reserving margin seems a good choice.

Results

Tab. 3.9 shows how different sparse grids level affects the accuracy. With other configurations unchanged, the increase of sparse grids level does not seem to increase the accuracy. It may be because that the boundary for different classes in latent space is pretty explicit. Fig. 3.22 shows an example of a 2d hidden space where the separation boundary is very clear, so increasing the level does not necessarily improve the accuracy.

Sparse Grids Level	3	4	5	6	7
Accuracy	99.15%	99.16%	98.91%	98.87%	99.01%

Table 3.9: Accuracy of different Sparse Grids levels

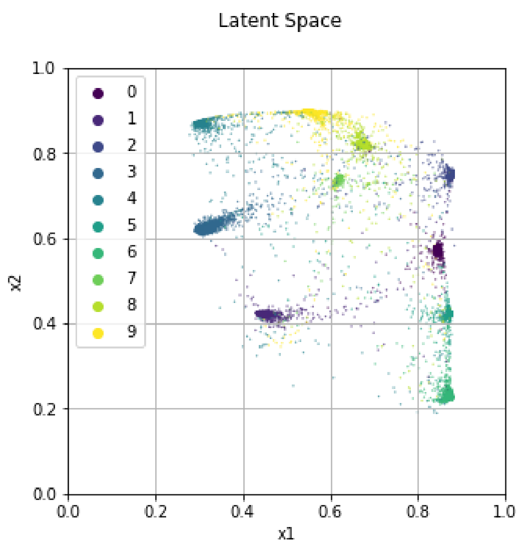


Figure 3.22: Final embeddings in 2d latent space of MNIST

In addition, the results of different models are listed in Tab. 3.10. Our approach obviously perform better than traditional Sparse Grids. What we can also know is that a CNN-SG model is comparable with ResNet18, although a ResNet18 is slightly better. Besides, it can be concluded that a CNN based model performs better than a dense network based model like the general cases in deep learning.

Model	B/M CNN[5]	ResNet18	SG[22]	CNN-SG	CNN-SG(2d)	MLP-SG(2d)
Acc	99.84%	99.21%	82.51%	99.16%	98.27%	97.13%

Table 3.10: Accuracy on MNIST of different models

3.5 Result Analysis

This section discusses the results from the previous sections. Firstly, we look into different training methods and compare them. Secondly, the change of embeddings in latent space is shown. Lastly, comparisons between NSG and ANN are presented.

3.5.1 Training

In the following, we discuss some observations in training. On the one hand, we analyze loss burst appeared during training. On the other hand, we also show how the results and training process differ from the three training methods.

Loss burst

When we optimize the parameters, we might encounter the problem of loss burst. It is a phenomenon that the loss increases much(usually over 10 times) within several epochs and then drops to a normal value. Fig. 3.23 shows an example where we can see that after epoch around 160 we got a loss burst that is more than fluctuation. It increases the loss over 10 times and normally goes afterward. Despite this loss burst, the other things look very normal. In fact, it appears no matter which training method we choose.

Usually, when a pure ANN is being trained, a large learning rate leads to a large fluctuation or smooth(compared to this loss burst)increase of loss. This type of loss peak is rare. In order to know if it results from our training methods, we train an NSG with different methods and learning rates multiple times. The results are in Tab. 3.11. We can conclude from the results that a lower learning rate can reduce the probability of occurrence regardless of how we train it, although we can not exclude the other factors that influence it. Tab. 3.11 tells us that the burst rate increase as the learning rate increase regardless of training

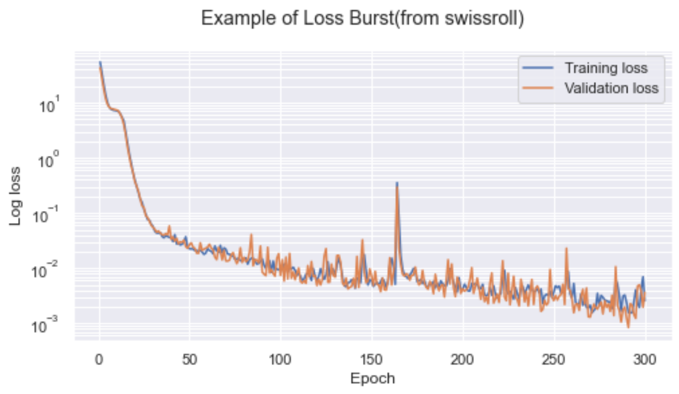


Figure 3.23: An example of loss burst

methods. In this case, the threshold might be 10^{-3} and 5^{-3}. The former brings no burst, while the latter leads to a very high burst rate.

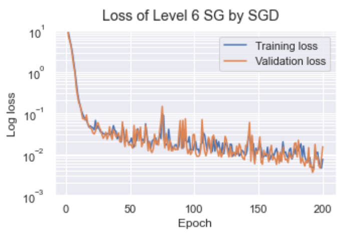
Train	LR	Burst %	Train	LR	Burst %	Train	LR	Burst %
SGD	1×10^{-3}	0%	LS	1×10^{-3}	0%	ILS	1×10^{-3}	0%
SGD	2×10^{-3}	30%	LS	2×10^{-3}	30%	ILS	2×10^{-3}	30%
SGD	3×10^{-3}	60%	LS	3×10^{-3}	40%	ILS	3×10^{-3}	50%
SGD	4×10^{-3}	90%	LS	4×10^{-3}	80%	ILS	4×10^{-3}	80%
SGD	5×10^{-3}	100%	LS	5×10^{-3}	100%	ILS	5×10^{-3}	90%

Table 3.11: Loss burst rate by different learning rate. Statistics are collected by swissroll example and we run each experiment for 10 times.

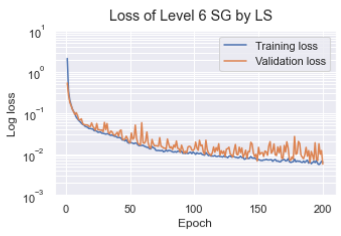
Training methods

Next, we look at these three training methods. Fig. 3.24 shows how loss changes during optimization using these three methods. Firstly, we can see all of them converges and lead to a good result. Secondly, if we compare the curve between training loss and validation loss, we will know that in SGD they fit each other very well, while in LS and ILS they fit each other worse. And ILS has more fluctuations than LS. This actually means LS and ILS overfit the data because, in each iteration, they use the actual values to compute the coefficients of Sparse Grids. Even adding large regularization terms cannot stop the validation loss from departing from training loss. However, if we only compare the first several epochs, we will see that SGD does not converge at fast as the other two methods.

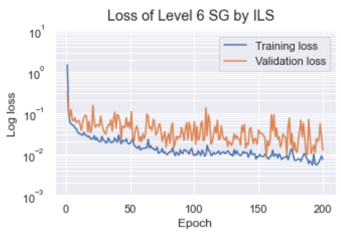
Yet, what we care about most is usually the performance. Tab. 3.12 shows MSE by those training methods that shows these methods are comparable on swissroll example, but SGD produces good results on stellar parameter estimation problem.



(a) Loss change by SGD



(b) Loss change by LS



(c) Loss change by ILS

Figure 3.24: Loss changes by different training methods

MES by different trainig methods		
Train	Swissroll	Stellar Parameter
SGD	0.0006	0.0269
LS	0.0012	0.0416
ILS	0.0004	0.0410

Table 3.12: MSE by 3 training methods on swissroll data set and stelar parameter estimation

3.5.2 Analysis of Latent Space

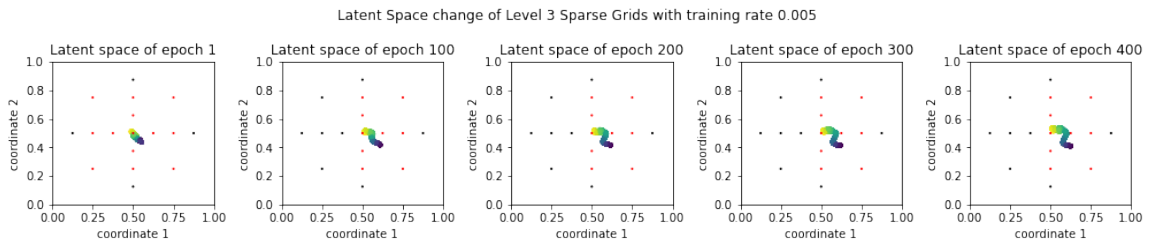
Investigating latent space is meaningful as we know how the embeddings look and how they change. Besides, a good representation of data helps to get good performance. In the following, we discuss how the embeddings change during optimization and how sparse grids level affects the embeddings.

Embeddings change with epoch

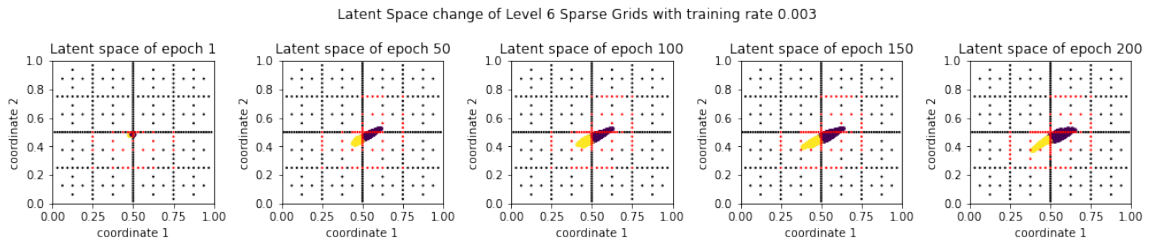
In the following, we focus on the change of embeddings with epoch. To start with, we take the two moons variant as an example. Fig. 3.28 records the embeddings, telling us how they in latent space change as the optimization goes on. Fig. 3.25a shows the records of swiss roll, where we also observe the similar phenomenon. We can conclude that the embeddings start with a small cluster of points around the centroid and then grow up to a certain size when it converges. From this point, it stops growing up even though a large number of grid points are wasted.

This phenomenon appears in all FNN based NSG models. Fig. 3.26 shows how the projected embeddings change for stellar parameters estimation. Similar behavior is observed, although it is projected from 3d to 2d.

However, in CNN-based NSG models, it is totally different. For simplicity, we reduce the



(a) Embeddings change of swissroll dataset.



(b) Embeddings change of two moons variant dataset

Figure 3.25: Change of embeddings with epochs on artificial dataset

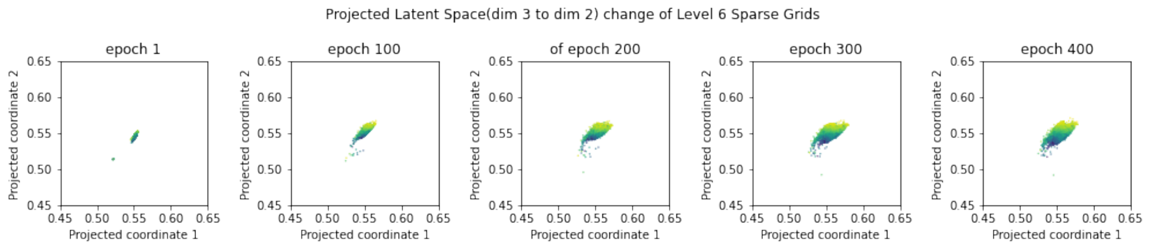
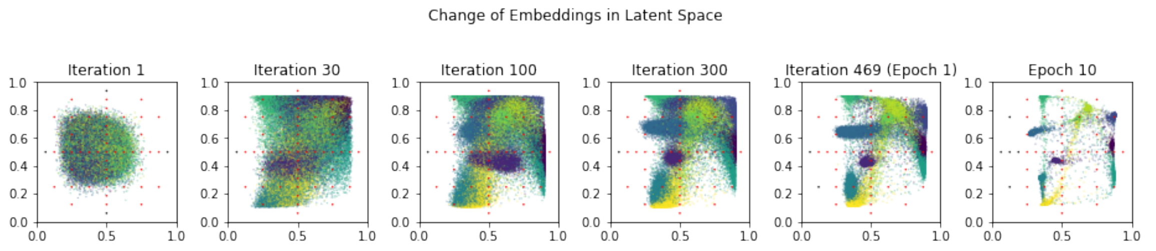


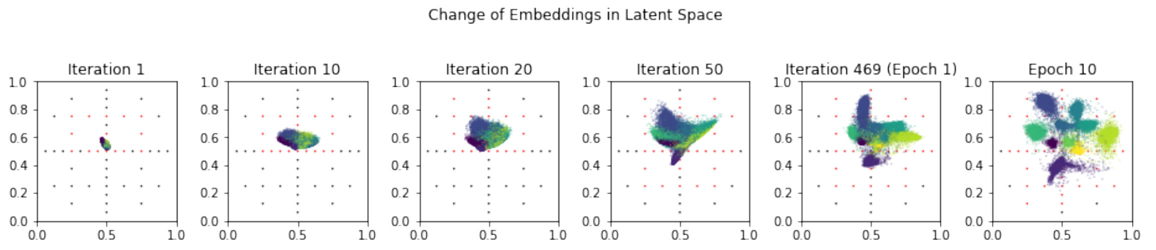
Figure 3.26: Change of embeddings with epochs of stellar parameter estimation

dimension of hidden space to 2 and reserve margin for boundary treatment. Although it does not sound like a good idea to have two dimensions for MNIST, it produces a test accuracy of 98.2%. Fig. 3.27a shows the change of MNIST embeddings with iterations. We can see that right after initialization, the embeddings are quickly pushed to the boundary and corner. The points that belong to the same class keep gathering together until convergence. Therefore, during the whole training process, the embeddings spread out almost all over the hidden space.

For comparison, similar things are done on an MLP based NSG with a 2d hidden layer. Fig. 3.27b shows the records. It still behaves similarly to what we have discussed above, although there are 10 classes.



(a) CNN based NSG. Embeddings spread out over the space during training and points belonging to the same class become closer with each other.



(b) MLP based NSG. Diameter of embeddings increases during training until convergence.

Figure 3.27: Change of embeddings with epochs in 2d latent space of MNIST

Embeddings change with Sparse Grids level

Having considered the changes of embeddings during optimization, we now look at how sparse grids level affects it. Fig. 3.28 is the final embeddings in latent space of NSG with different SG levels. From the figure, we can conclude that the shape of embeddings decreases as the SG level increases. In level 8, the diameter of embeddings is so small that we can hardly find it in the latent space.

A good explanation is that a higher level of SG has a higher resolution in the space. Remember, the embeddings of an MLP-based NSG grow up during optimization until convergence, as we have shown before. For an SG with higher resolution, it converges when the shape of embeddings is still small because, in this small area, sparse grids are already able to achieve a good result.

Tab. 3.13 shows the number of activated grid points of different SG level. We can find that the number of activated points increases as the level increases. But the ratio of activated points and the total points decreases rapidly. The level 8 SG even only has 6% points activated while it still gets 96.2% accuracy.

However, CNN-based NSG is different. Since we have 8 dimensions in CNN-based example, we use violin plot to show how the embeddings distribute per dimension in Fig. 3.29. It shows the dimension-wise embedding distribution of level 3, 4, 5, 7 Sparse Grids.

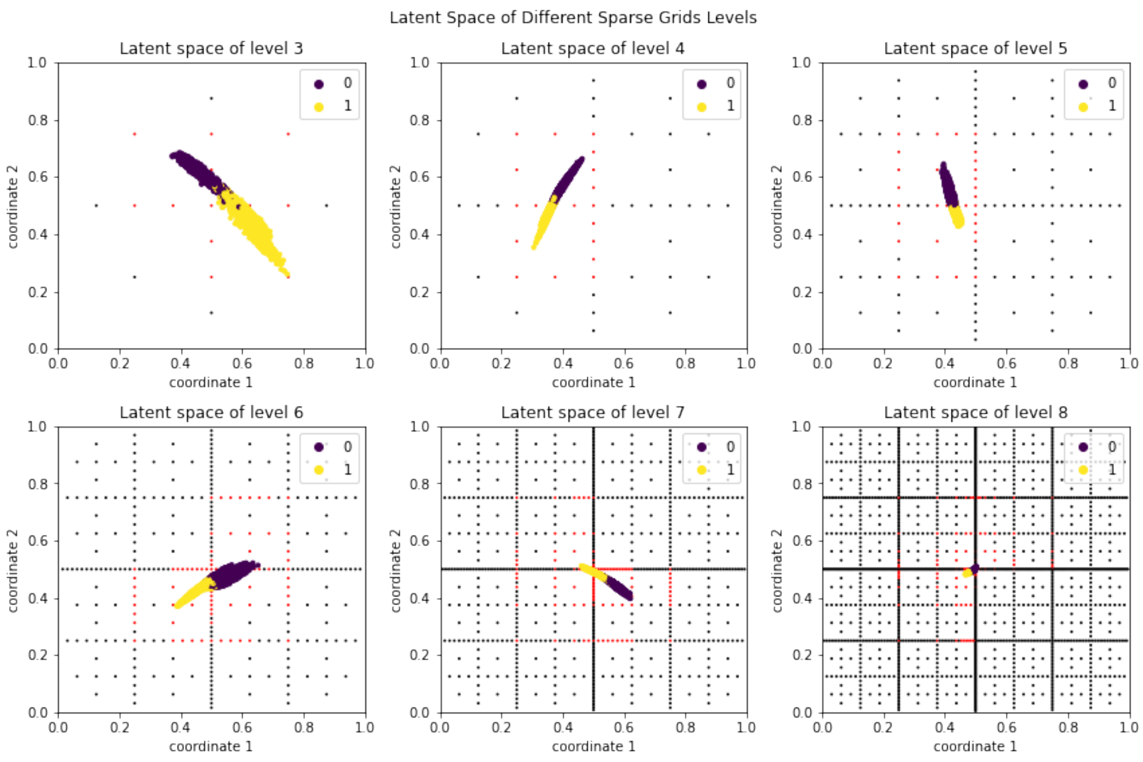
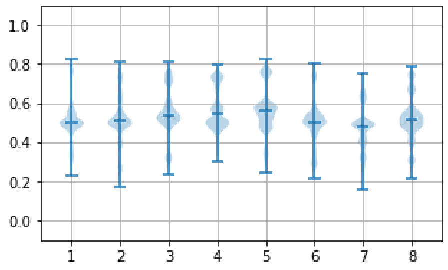


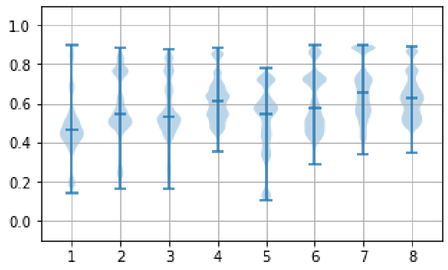
Figure 3.28: Embeddings produced by NSG with different SG levels on two moons variant. The diameter of embeddings decrease dramatically as SG level increases.

We can see almost all of these SG levels have embeddings spreading over the space. But there is a minor difference between them that lower levels, like 3, do not cover the axis as much as the higher levels do. For level 7, we can see the embeddings almost cover $[0.1, 0.9]$ in all dimensions, meaning it is spreading out the whole space as we have restricted the rescaling function producing values in $[0.1, 0.9]$.

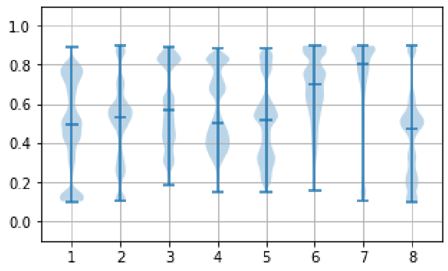
Since it is different from the MLP-based NSG, we also calculate the activated grid points in this case. Tab. 3.14 lists the number of activated grid points for different levels. We can find that the number of activated grid points increases as the level increases. But there seems no obvious regular pattern of how the percentage changes. We can know from the table that a large proportion (over 50%)of grid points is activated no matter what the level is. However, the accuracy of them seems the same.



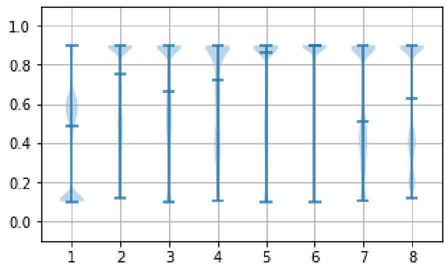
(a) Level 3 SG



(b) Level 4 SG



(c) Level 5 SG



(d) Level 7 SG

Figure 3.29: Dimension-wise embedding distribution change with SG level. Here, X-axis represents each dimension while y-axis represent the domain.

Levels	# Grid points	# Activated grid points	Ratio	Accuracy
3	17	12	0.70	0.9565
4	49	19	0.38	0.9580
5	129	31	0.24	0.9575
6	321	80	0.25	0.9645
7	769	105	0.13	0.9620
8	1793	108	0.06	0.9620

Table 3.13: Activated grid points in the two moons variant

Levels	# Grid points	# Activated grid points	Ratio	Accuracy
3	161	157	0.975	0.9912
4	1121	848	0.756	0.9907
5	6401	5251	0.820	0.9907
6	31745	19525	0.615	0.9905
7	141569	91479	0.646	0.9915

Table 3.14: Activated grid points in the MNIST example

3.5.3 Comparison with Artificial Neural Networks

To compare the performance between ANN and NSG, we replace the Sparse Grids in NSG with a layer of ANN that has the same number of parameters.

Suppose we have a Sparse Grids in d-dimension that has p outputs and the number of grid points of Sparse Grids is m. Assume the ANN layer for replacement with bias has n neurons, d inputs and p outputs. As they have the same number of parameters, we have the Eq. 3.27:

$$mp = (d+1)n + (n+1)p \tag{3.27}$$

So this ANN layer should have $\frac{mp-p}{d+p+1}$ neurons. We replace the Sparse Grids layer with such an ANN layer and train models again and get the results in Tab. 3.15 where the row of ANN is the result of the model with Sparse Grids layer replaced by an ANN layer. We can see that both models produce similar results. Although NSG seems to perform slightly better, we cannot make a deterministic conclusion that it is superior. Nevertheless, NSG does not perform worse in these 4 cases.

Besides, it is also meaningful to compare a layer of sparse grids and a layer of ANN on the same embeddings. Thus, we construct models with a replacement ANN layer as is mentioned above to produce embeddings that are used for comparison; then, we replace the last ANN layer with a Sparse Grids layer; finally, we train the Sparse Grids layer by SGD(or solve a linear system for regression problem).

The results are shown in Tab. 3.15 where the NSG constructed on the existing embeddings

	Regression(MSE)		Classification(Accuracy)	
	Swissroll	Stellar Parameter	Two Moons Variate	MNIST
ANN	0.0012	0.0315	0.9690	0.9914
NSG	0.0006	0.0269	0.9700	0.9918
NSG*	0.0009	0.0373	0.9624	0.9932

Table 3.15: Performance of ANN and NSG on different data set

is denoted as NSG*. Compared with the other two models, it does not perform significantly differently.

4 Conclusion and Future Work

This thesis introduces a new model Neural Sparse Grids(NSG), as well as how it is applied to solve regression and classification problems. In Ch. 2 we briefly reviewed Artificial Neural Networks(ANN) as well as Sparse Grids(SG), which are the basic concepts in NSG. Then, NSG is introduced in Sec. 3.2 including its mathematical form, three methods for learning parameters. After that, in Sec. 3.3 we have seen how this new methodology is applied in regression problem on an artificial data set swiss roll as well as a real-world data set stellar spectra from SDSS in detail. Similarly, its application on classification is also discussed in Sec. 3.4 by showing a two moon variant example as well as a real-world image data set MNIST on which convolutional neural network(CNN) is integrated into our model. Last but not least, in Sec. 3.2 there is a discussion about this new model, including training, embeddings in latent space as well as comparisons with pure ANN methods.

Looking through the entire thesis, we have the main contributions being

- to propose this new model structure and three possible methods to learn the parameters.
- to show its results of applications on regression and classification problems by using both artificial data and real-world data ,as well as how sparse grids level can affect the result.
- the integration with CNN in order to solve image-based problems.
- to show how the embeddings change with epoch and how they change with sparse grid levels.

As for applications, we have found the sparse grids layer comparable with an ANN layer that has the same number of parameters in the cases we have investigated. However, as the data set we used is not complicated, so far, we cannot draw the conclusion that NSG is superior to ANN. At least we provided another possibility to solve machine learning problems.

Since it is a newly proposed methodology, there are some points that can be improved. Having considered the contributions of this work, we believe the following work can be done in the future.

This thesis uses an ANN-style way to do classification by adding a softmax function after the output. But SG can do classification in a density estimation way, which can also be applied in NSG. In addition, remember the inefficient use of grid points in our applications, especially for regression tasks, we can improve the embeddings in latent space. For example, ideas from variational autoencoder(VAE)[12] may help get ”nicer” embeddings, resulting perhaps in better performance as more grid points can be used. Moreover, integration with other classic ANN architectures can also be considered, such as graph neural network(GNN) and transformer.

Bibliography

[1] Kevork N Abazajian, Jennifer K Adelman-McCarthy, Marcel A Agüeros, Sahar S Allam, Carlos Allende Prieto, Deokkeun An, Kurt SJ Anderson, Scott F Anderson, James Annis, Neta A Bahcall, et al. The seventh data release of the sloan digital sky survey. *The Astrophysical Journal Supplement Series*, 182(2):543, 2009.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[3] Yude Bu and Jingchang Pan. Stellar atmospheric parameter estimation using gaussian process regression. *Monthly Notices of the Royal Astronomical Society*, 447(1):256–265, 2015.

[4] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta numerica*, 13(1):147–269, 2004.

[5] Adam Byerly, Tatiana Kalganova, and Ian Dear. A branching and merging convolutional network with homogeneous filter capsules. *arXiv preprint arXiv:2001.09136*, 2020.

[6] David Chin-Lung Fong and Michael Saunders. Lsmr: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33(5):2950–2971, 2011.

[7] Gene H Golub, Per Christian Hansen, and Dianne P O'Leary. Tikhonov regularization and total least squares. *SIAM journal on matrix analysis and applications*, 21(1):185–194, 1999.

[8] Rajarshi Guhaniyogi and David B Dunson. Compressed gaussian process for manifold regression. *The Journal of Machine Learning Research*, 17(1):2472–2497, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Douglas Heaven. Why deep-learning ais are so easy to fool. *Nature*, 574(7777):163–166, 2019.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint*

arXiv:1312.6114, 2013.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[14] Martin HC Law and Anil K Jain. Incremental nonlinear dimensionality reduction by manifold learning. *IEEE transactions on pattern analysis and machine intelligence*, 28(3):377–391, 2006.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] Xiangru Li, QM Jonathan Wu, Ali Luo, Yongheng Zhao, Yu Lu, Fang Zuo, Tan Yang, and Yongjun Wang. Sdss/segue spectral feature analysis for stellar atmospheric parameter estimation. *The Astrophysical Journal*, 790(2):105, 2014.

[18] James McQueen, Marina Meila, and Dominique Joncas. Nearly isometric embedding by relaxation. *Advances in Neural Information Processing Systems*, 29:2631–2639, 2016.

[19] Benjamin Peherstorfer, Fabian Franzelin, Dirk Pflüger, and Hans-Joachim Bungartz. Classification with probability density estimation on sparse grids. In *Sparse Grids and Applications-Munich 2012*, pages 255–270. Springer, 2014.

[20] Dirk Michael Pflüger. *Spatially adaptive sparse grids for high-dimensional problems*. PhD thesis, Technische Universität München, 2010.

[21] Peter Sadowski and Pierre Baldi. Deep learning in the natural sciences: applications to physics. In *Braverman Readings in Machine Learning. Key Ideas from Inception to Current State*, pages 269–297. Springer, 2018.

[22] Samuel Weber. Exploiting the data hierarchy with geometry aware sparse grids for image classification. 2019.

[23] Tan Yang and Xiangru Li. An autoencoder of stellar spectra and its application in automatically estimating atmospheric parameters. *Monthly Notices of the Royal Astronomical Society*, 452(1):158–168, 2015.