Project Neural Information Processing

Generalizing Optimization Surrogates based on Neural Networks

Jannik Wolff, Jonas Langhabel

28th of August 2016

Neural Information Processing Group

Institute of Software Engineering
and Theoretical Computer Science
Technical University of Berlin

Examiner: Prof. Dr. rer. nat. Klaus Obermayer

Supervisor: M.Sc. Raphael Holca-Lamarre

Abstract

Optimization constitutes a highly relevant but complex task in many industries. Special focus lies on efficiency, as obtaining samples from the objective functions is typically particularly expensive. We address this problem by using neural networks as surrogate models and estimates of uncertainty in the surrogates to orchestrate the optimization process. We evaluate multiple methods for uncertainty estimation and apply them to balance between exploration and exploitation. The approach is economical regarding the number of required samples. We further propose a method to build a meta-model over a class of similar optimization problems, which we show more than halves the number of samples required in new problems by generalizing from old ones.

Contents

List of Figures

1 Introduction

Optimization is crucial in aerospace industry. Particular focus lies not only on effectivity but also on efficiency. The latter refers to the amount of resources needed until a sufficient optimum has been found. Particularly costly is the process of retrieving performance measures for given parameters. A corresponding objective function that displays those dependencies can be constituted within a computational simulation [2]. Jeong et al. [5] provide an example regarding such simulations. They illustrate computational fluid dynamics, an essential field in aerodynamic optimization design. Even with supercomputers, large-scale calculation of many or all possible combinations of variables is not feasible [5]. As Forrester et al. [2] point out, this problem drastically intensifies with increasing dimensionality. Due to the diverse amount of relevant dimensions in aerospace industry, the 'curse of dimensionality' becomes highly relevant. Adkins et al. [1] provide several examples on possible dimensions in aerospace industry. Concerning the design of optimal propellers for maximum efficiency, relevant dimensions rank from fluid density to geometrical features and the amount of blades. Further possible dimensions might be coefficients from material science such as the elastic modulus.

In this report, we want to mimic the human capability of using the memory of past experience and applying them to novel problems. In everyday life, we have to face a variety of similar problems. For example, when being faced with a new traffic situation, a driver automatically generalizes from previous similar experiences. Applied to the optimization problems in aerospace industry, we assume that not all of those problems are independent of each other. For example, two propellers might have to be optimized in two different fluids, respectively. On the one hand, parameters of the fluid such as density and viscosity might vary. On the other hand, parameters in other dimensions might behave similarly, making both problems dependent on each other. The optimum of a second propeller's design might be detected more efficiently if knowledge from previously solved propeller optimization problems in different fluids is used. We refer to this process as *generalization*.

In this report, we use neural networks (NN) to build a *surrogate model* over multiple function optimization problems. We assume those to be minimization problems regarding the objective function. Specifically, we assume that calls to the objective function are expensive, but lack any observational noise. We use Google's TensorFlow framework to configure a NN that is capable of nonlinear functional regression. The selection process of samples is governed by an uncertainty based approach,

which considers the exploration and exploitation dilemma. First, we evaluate five methods of obtaining the NN's certainty: Two distance based methods, Kriging, MC-Dropout and ensembles of NNs. The latter yields the most promising results despite requiring vast computational costs. Then, we implement the two-stage approach to exploration and exploitation as proposed by Forrester [2] and test it on 1D and 2D problems. Finally, we generalize from 1D optimization problems over a second dimension using the Six-Hump-Camel function as the objective function. We construct a 2D meta-model, which we call a *meta-surrogate*, and show that we can lower the amount of necessary samples to find minima of novel 1D problems using knowledge of previously solved 1D problems.

2 Background

2.1 Surrogate based Optimization

In this report, we use approaches as laid out by Forrester et al. [2] to use surrogates in the context of optimization problems in aerospace industry. For further reading, we refer to Queipo et al. [8] who provide extensive background to surrogates used in this industry. Simpson et al. [11] portray the development of meta-model techniques such as surrogates from 1989 until 2008.

We assume that the objective function returns a performance measure for given parameters such as height or width of a propeller. Calls to the objective function are associated with prohibitive (computational) costs. Consequently, the amount of calls is limited by given resources as Forrester et al. [2] point out. They also elaborate that calls to the objective function do not necessarily have to return valid values due to calculation of performance measures as drag, lift or stress being sophisticated. In this report, however, we assume that the objective function always returns a valid output. We also assume that this function is smooth and continuous.

Queipo et al. [8] elaborate on the necessity of meta-models for coping with high-dimensionality, especially in aerospace industry. As many dimensions result in an enormous search space, meta-models such as surrogates are capable to significantly lower computational costs.

The following intuition might illustrate the characteristics of meta-models: In everyday life, humans face a variety of different decisions. For example, when turning right on congested roads in cities, humans internally build a meta-model assessing different scenarios. They instinctually recognize and evaluate positions of other cars, cyclists and pedestrians. They might also consider the weather and the estimated braking distance based on their past experiences. Furthermore, they regard traffic rules and anticipate other people's behavior. They assess their uncertainty regarding their evaluations. If they are not certain enough, they might want to gather helpful new information. If they have reached a sufficient level of certainty and accuracy in their estimations, they finally decide on an action.

This basic example mimics in its essence engineering problems in aerospace optimization. Based on a variety of different parameters, a surrogate can be constructed. However, the selection of parameters and the construction and optimization of the

surrogate are not trivial, but rather very complex. Once the meta-model is sufficiently established, it constitutes the basis of (design) decisions. Another crucial aspect displayed in the analogy above regards the decision on gathering new information. As we want to sample as little as possible in aerospace optimization, a sampling process governed by estimates of uncertainty might improve the efficiency.

Forrester et al. [2] portray a variety of sampling strategies, which are all motivated by the exploitation and exploration dilemma. A basic example of pure exploitation is an approach minimizing the predictor. This method recommends to sample at the surrogate's minimum as this could imply a low value of the objective function. However, pure exploitation might be unpractical as some regions of the search space might not have been explored properly. An example of pure exploration is sampling at the uncertainty's peaks. This approach, however, disregards any exploitation of regions that have already been explored. Therefore, Forrester et al. also propose balanced approaches named the two-stage and one-stage approach. In this work, we use the two-stage approach.

Forrester et al. refer to the *probability of improvement* which displays the probability that a new sample constitutes the new optimum. Within the two-stage approach, this probability is formulated as follows:

$$P[I(x)] = \frac{1}{s \cdot \sqrt{2\pi}} \int_{-\infty}^{0} \exp\left(\frac{-(I - \widehat{y}(\mathbf{x}))^2}{2 \cdot s^2}\right) dI \tag{2.1}$$

$I(x)$ is defined as $I(x) = y_{min} - y(x)$, meaning that $I(x)$ only yields positive values if the surrogate's function value is below the current minimum of all samples. As the integral is evaluated from $-\infty$ to 0, only values below the minimum are considered as an improvement. Note that we face a minimization problem and thus only samples with *lower* function values are of interest. *Figure 2.1* further illustrates *equation 2.1*, as only the part of the Gaussian distribution below the threshold of y_{min} is colored. *Equation 2.1* shows that this distribution is centered around the surrogate's estimate $\widehat{y}(\mathbf{x})$. The variance s^2 refers to the uncertainty regarding the surrogate's output.

Following upon *equation 2.1* Forrester et al. introduce *equation 2.2* which depicts the expected improvement for a new sample at $\mathbf{x}$:

$$E[I(\mathbf{x})] = \begin{cases} (y_{min} - \widehat{y}(\mathbf{x})) \cdot \Phi\left(\frac{y_{min} - \widehat{y}(\mathbf{x})}{s(\mathbf{x})}\right) + s \cdot \phi\left(\frac{y_{min} - \widehat{y}(\mathbf{x})}{s(\mathbf{x})}\right) & , s > 0 \\ 0 & , s = 0 \end{cases} \tag{2.2}$$

$\Phi(\cdot)$ refers to the normal cumulative distribution function while $\phi(\cdot)$ refers to the respective probability density function. *Equation 2.2* can be graphically interpreted considering *figure 2.1*: Basically, $E[I(x)]$ constitutes the mean of the integrated area of the normal distribution below the threshold. Therefore, if for example the uncertainty $s(\mathbf{x})$ is extraordinary high and the surrogate's value $\widehat{y}(\mathbf{x})$ is not too far above the minimum, the expected improvement might have its peak exactly at this

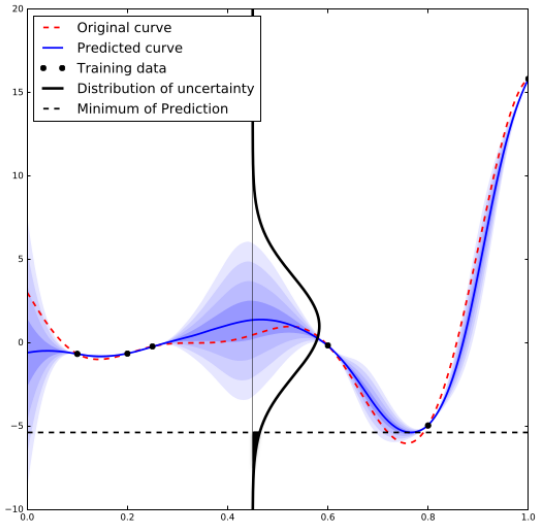


Figure 2.1: Normal distribution over the possible values of $\widehat{y}(x)$ portraying the probability of improvement.

point $\mathbf{x}$. If, however, the uncertainty $s(\mathbf{x})$ equals 0, there is no expected improvement as we have already sampled at this point.

2.2 Regression

Optimizing multidimensional functions can be a nonlinear regression problem. In contrast to classification the task is not to divide the data into discrete classes but to predict continuous function values. Ordinary Least Squares or Ridge Regression are classical machine learning algorithms that fit the data with linear functions. Using the kernel trick they broaden to nonlinear problems as well, such as Kernel Ridge Regression and Support Vector Regression. Noteworthy and more advanced algorithms are Kriging, which uses Gaussian processes to interpolate between mostly noise free data and Radial Basis Function Networks, a type of NN having one hidden layer with radial basis activation functions.

General Multilayer Perceptron (MLP) NNs can also be used for the task of nonlinear regression. However, they are most popular for their achievements in logistic regression tasks. NNs date back for at least 50 years but did not gain much attention until they outperformed the kernelized methods mentioned above about 15 years ago [9]. In 2011 they achieved the milestone of beating humans in the task of traffic sign recognition, and since then it is hard to escape their force of attraction. The massively parallel computing power of today's GPUs paved their way to win challenges in other areas, too. Popular assignments include time series prediction, source separation, as well as speech and pattern recognition.

Nonlinear regression with NNs can be considered a basic and solved problem. MLPs are general function approximators [4] and this statement even holds for nonsmooth and piecewise continuous functions [10]. However, this does not reduce the challenge to the designer who wants to adhere to Occam's principle and implement them: It is hard to keep the network as minimal as possible. If one sticks to the typical NN activation functions rectifier and sigmoid, the networks need to be surprisingly large. The required setup strongly depends on the type of data given. We will investigate this further in chapter 3.

3 Model

This chapter summarizes our experiences in choosing the NN parameters such that it achieves a good regression curve (or surface) for our purposes. We assume a noise free experimental set-up where the response variable y to an input x is determined by directly sampling the underlying function, i.e. $y = f(x)$. Thus, the target is not to optimize the bias-variance tradeoff such that the regression curve generalizes well. Instead, our objective is to fit the curve exactly on the training samples and likewise to achieve zero bias.

3.1 Configuration

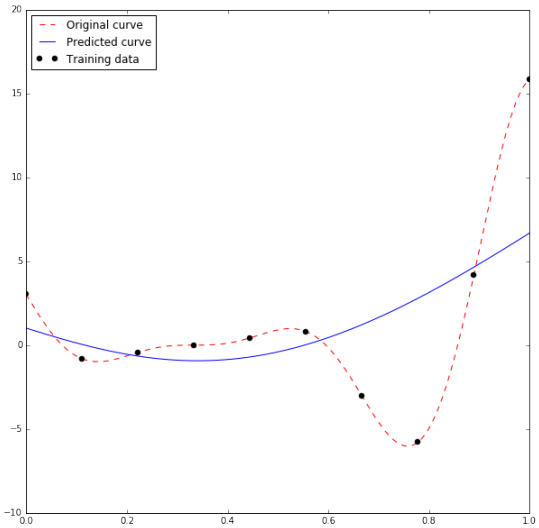
In this section, we investigate the most prominent parameters to gain an intuition of their characteristics. Our main criterion of performance is how quickly the data can be fit, under the above stated objective. Smoothness of the fit constitutes the second benchmark. We use the fully connected MLP architecture with both ReLU and sigmoid activation functions in two hidden layers. In this section, if not specified otherwise, we set the parameters as stated in *table 3.1*. The 1D simple function specified in the table is used for the benchmarks. An extension of the x value range to $[0, 2]$ results in massive difficulties concerning the fitting process.

Table 3.1: Regression testing configuration.

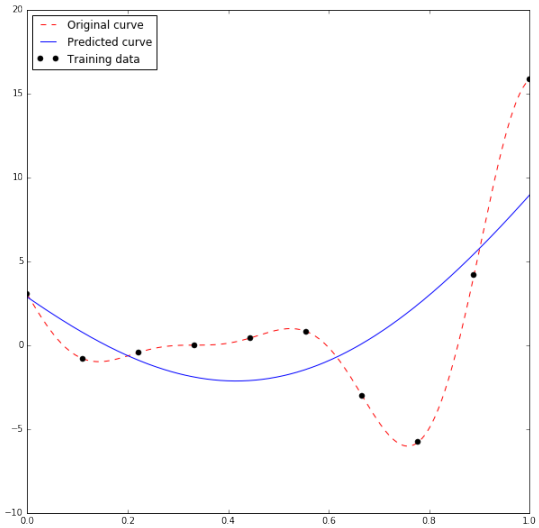
Object	**Property**
Function	$f(x) = (6x - 2)^2 \cdot sin(12x - 4)$
Range	$[0, 1]$
Samples	10 or 20 equidistant samples
Hidden Layers	2
Units per Layer	170
Optimizer	Adam
Initial Learning Rate	0.01
Training Epochs	2000
Regularization Factor	0.00005
Dropout (keep probability)	1.0

3.1.1 Hidden Layer Configuration

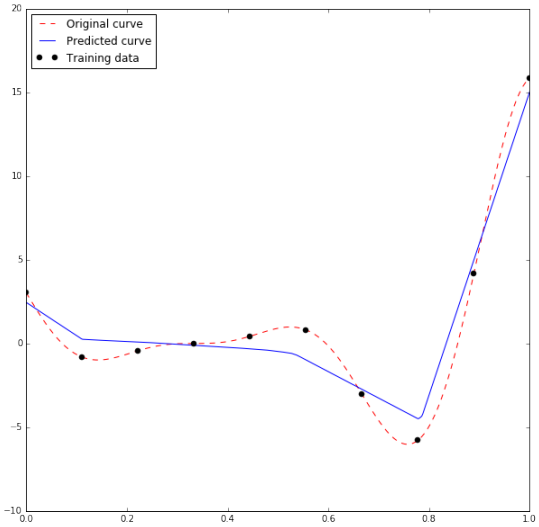
The number and size of the hidden layers determines the capabilities of the NN. Another important factor are the utilized activation functions and how they are combined.



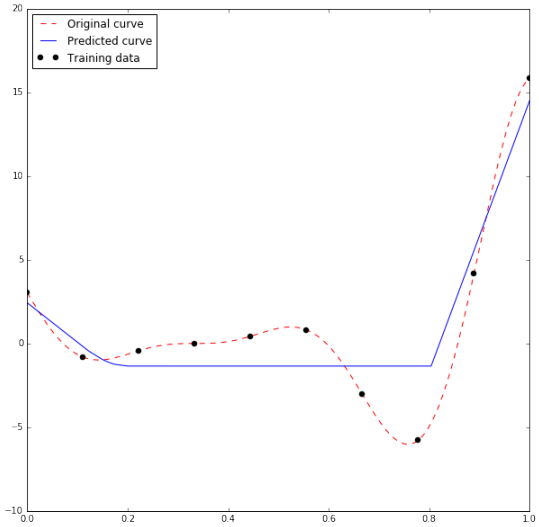
(a) Sigmoid activation, 170 hidden units.



(b) Sigmoid activation, 500 hidden units.



(c) ReLU activation, 170 hidden units.



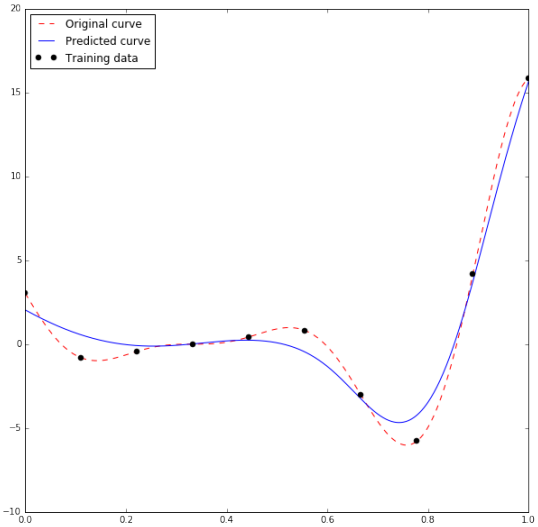
(d) ReLU activation, 500 hidden units.

Figure 3.1: The best fits with a one hidden layer network in different configurations.

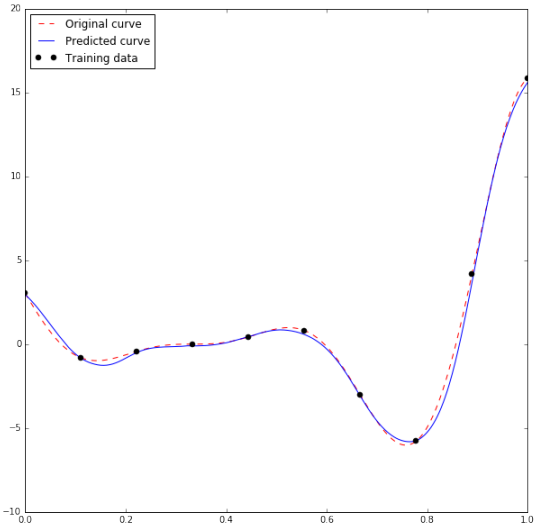
Number of Hidden Layers

We found one layer to be insufficient to fit the data, even with an exuberant number of hidden units. Surprisingly, the performance got worse when we used too many

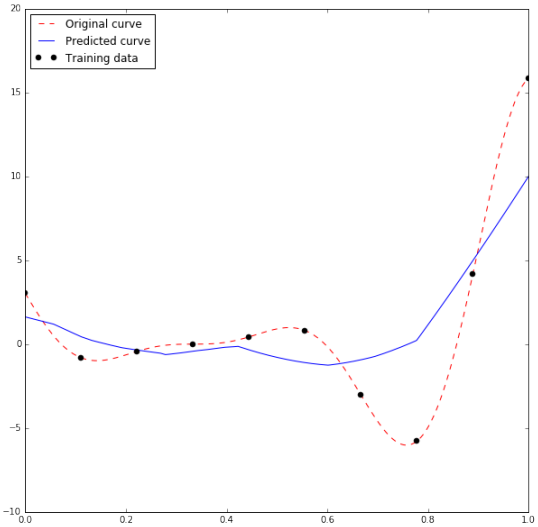
hidden units: The optimization was significantly more likely to get stuck in a local minimum. *Figure 3.1* not only illustrates that more hidden units may be inferior, but also the influence of the used activation function. In conclusion, the chief result is that one hidden layer is unsatisfactory to fit this function irrespective of the number of hidden units for the two given activation functions. *Figure 3.2* presents the outcome of combining ReLU and sigmoid activated layers in a two layer architecture. It provides evidence that a NN with two hidden layers has sufficient fitting powers.



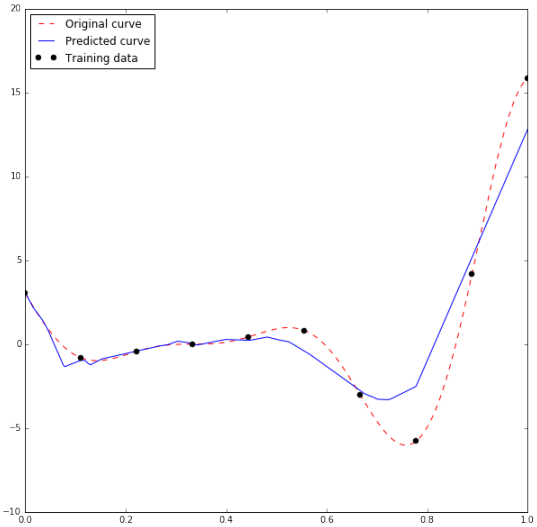
(a) 1st layer: Sigmoid, 2nd layer: Sigmoid



(b) 1st layer: ReLU 2nd layer: Sigmoid



(c) 1st layer: Sigmoid, 2nd layer: ReLU

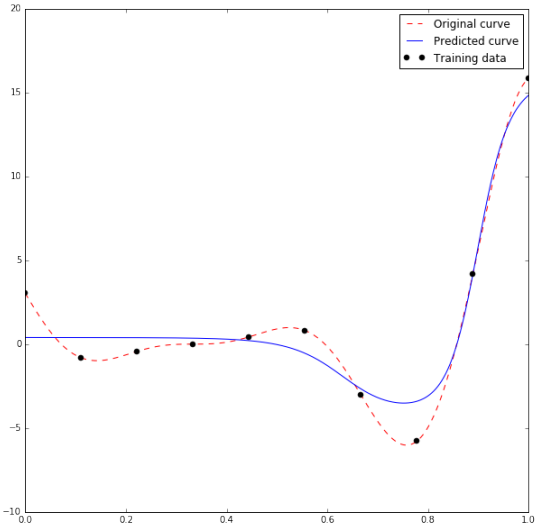


(d) 1st layer: ReLU, 2nd layer: ReLU

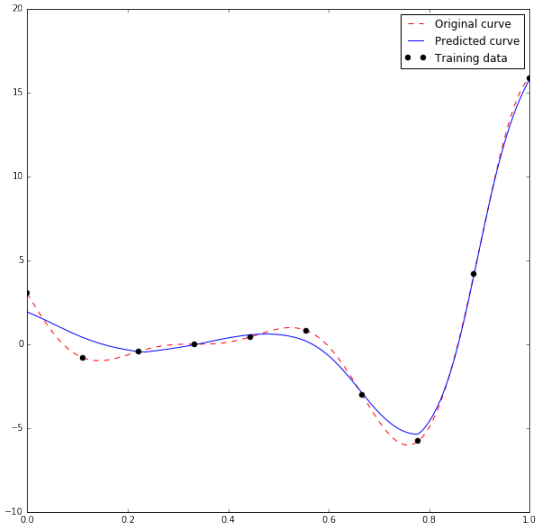
Figure 3.2: Different combinations of activation functions in a NN with two hidden layers.

Activation Functions

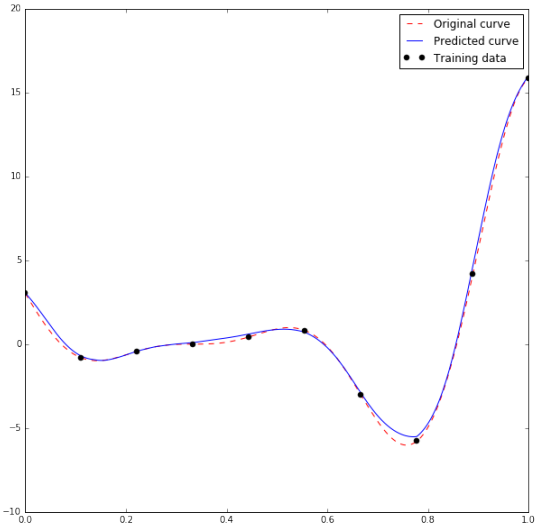
Most prominent is the finding that ReLUs give a curve with sharp inflexions, while the sigmoid activation fits more smoothly. The rectifier activations also appear to extend the latitude more than the sigmoid ones. Reason for that may be the different output ranges: $[0, \infty)$ for the rectifier in contrast to $(0, 1)$ for the sigmoid function. Positioning ReLUs in the second hidden layer required us to lower the learning rate to 0.0001. The number of training epochs was kept constant and each hidden layer consisted of 170 units.



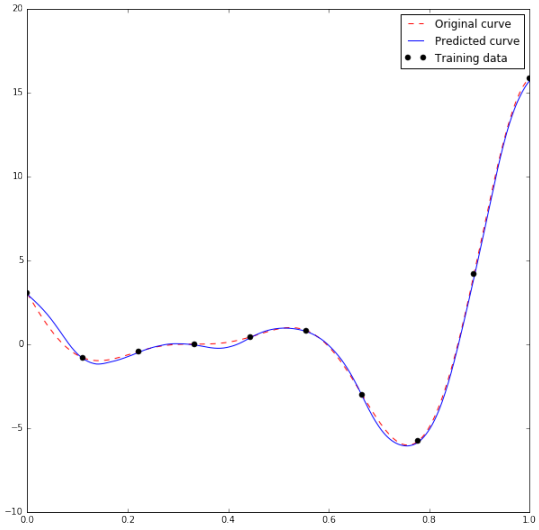
(a) 8 units



(b) 32 units



(c) 128 units



(d) 256 units

Figure 3.3: Different number of hidden units per layer.

Number of Hidden Units

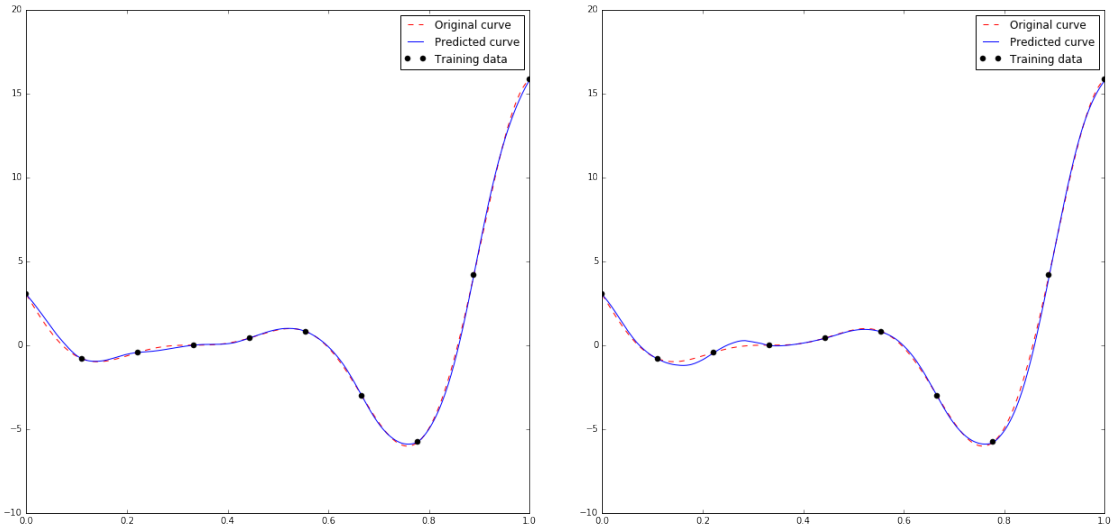
On the network configuration from above we evaluate the needed number of units per hidden layer. The results in *figure 3.3* lead us to the conclusion that we need at least 128 hidden units for an acceptable fit, ideally more. Almost optimal fits can also be obtained with between 8 and 32 units, if trained for about five times longer. However, as we require all data points to be precisely hit, the suggested setup seems to be more appropriate for regression with noisy data. For more complex functions we utilize up to 256 units. We mentioned before that a larger number of hidden units does not automatically increase the performance: Using about 500 or more units per layer for example leads to overflow errors in the cost computations.

3.1.2 Optimizer

We utilize two prominent optimization algorithms: Batch gradient descent and the Adam optimizer. Batch gradient descent computes the gradient for a minibatch or the whole dataset, followed by a single weight update. It requires learning rate annealing schedules for practicable performances. Adam is a state-of-the-art gradient based optimizer combining the advantages of AdaGrad and RMSProp, outperforming both algorithms [7]. Using moment estimations of the gradients, the learning rate is automatically adapted. *Figure 3.4* shows how batch gradient descent and the Adam optimizer perform on our setting, which can be considered simple in an optimizational point of view. The initial learning rate is set to 0.01 in both cases. For batch gradient descent, the decay rate equals 0.99 while the decay frequency is set to 100. The results are very similar. As expected, the differences in learning speed are marginal in this benchmark. However, we found Adam to give slightly smoother prediction curves. Finding the ideal learning rate decay and frequency for the batch gradient descent optimizer turned out to be a lengthy procedure and depended on the test function as well as the amount of given data. When applied to the function with range $[0, 2]$ both optimizers got frequently stuck in local optima (see *figure 3.6*).

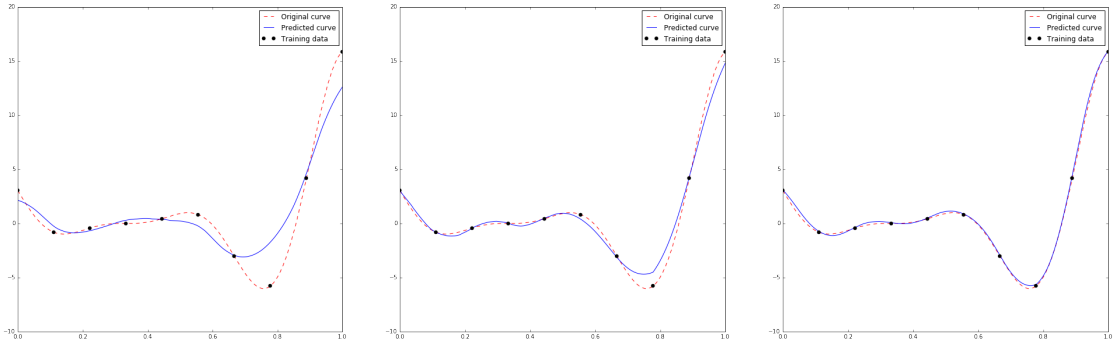
3.1.3 Learning Rate

Subsequently we analyze the influence of the initial learning rate parameter on the Adam optimizer. *Figure 3.5* shows the learned fit after 2000 training epochs for three different learning rates. Convergence was only achieved for rates of 0.005 to 0.02 within 2000 epochs. Higher learning rates quickly led to coarse results up to a straight line. Rates exceeding 0.3 resulted in overflows. The improvement from 1000 to 2000 epochs in training error constitutes only 3%, from 400 to 2000 epochs about 30%.



(a) Batch Gradient Descent (b) Adam

Figure 3.4: Comparison of the achieved fit by two popular optimization algorithms.



(a) Learning rate of 0.0001. (b) Learning rate of 0.001. (c) Learning rate of 0.01.

Figure 3.5: Comparison of different learning rates after 2000 epochs.

3.1.4 Training Duration

From *figure 3.7* we learn that 2000 to 5000 training epochs are needed to obtain a good fit (see *section 3.2*). More training makes the regression curve overly complex by oscillating between the samples.

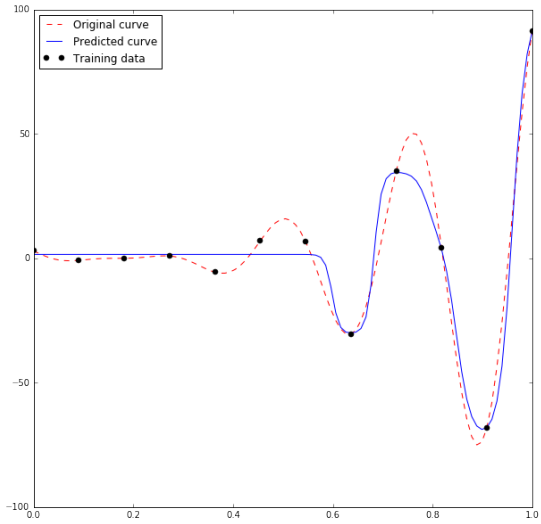
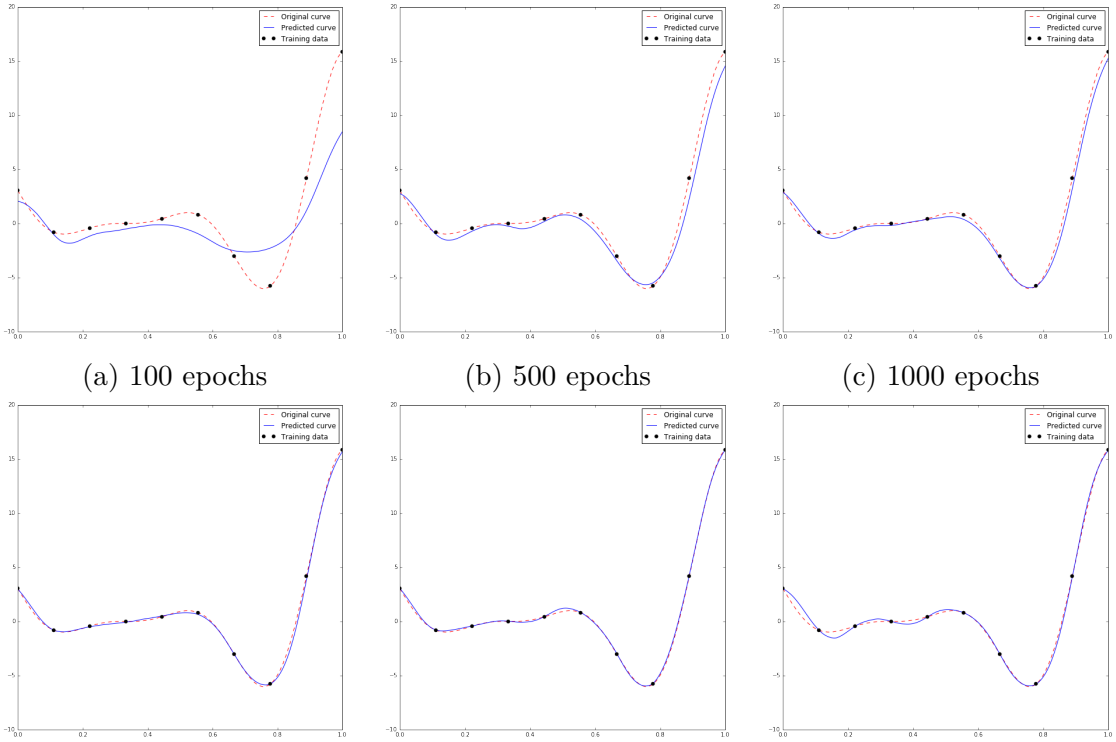


Figure 3.6: The optimization being stuck in a local optimum.



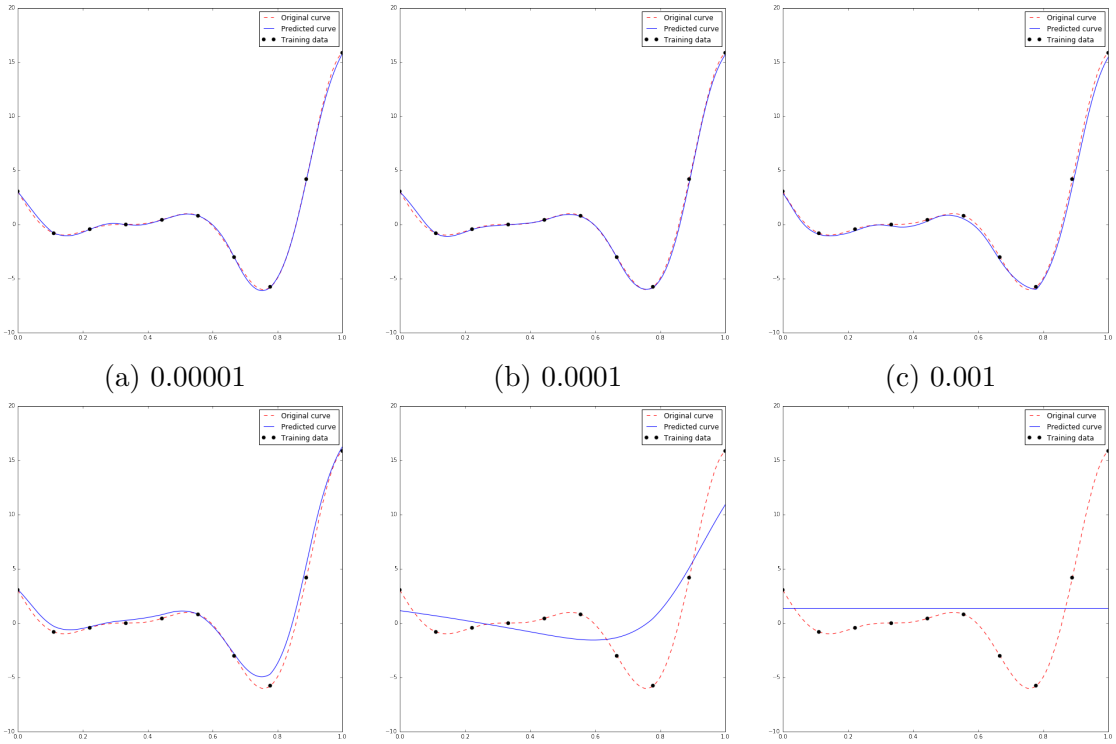
(a) 100 epochs (b) 500 epochs (c) 1000 epochs

(d) 2000 epochs (e) 5000 epochs (f) 10000 epochs

Figure 3.7: The fit after an increasing number of training epochs.

3.1.5 Regularization

We implement L2-Norm weight decay as a regularization method. *Figure 3.8* shows the results for different constant factors of the weight decay penalty term in the loss function.



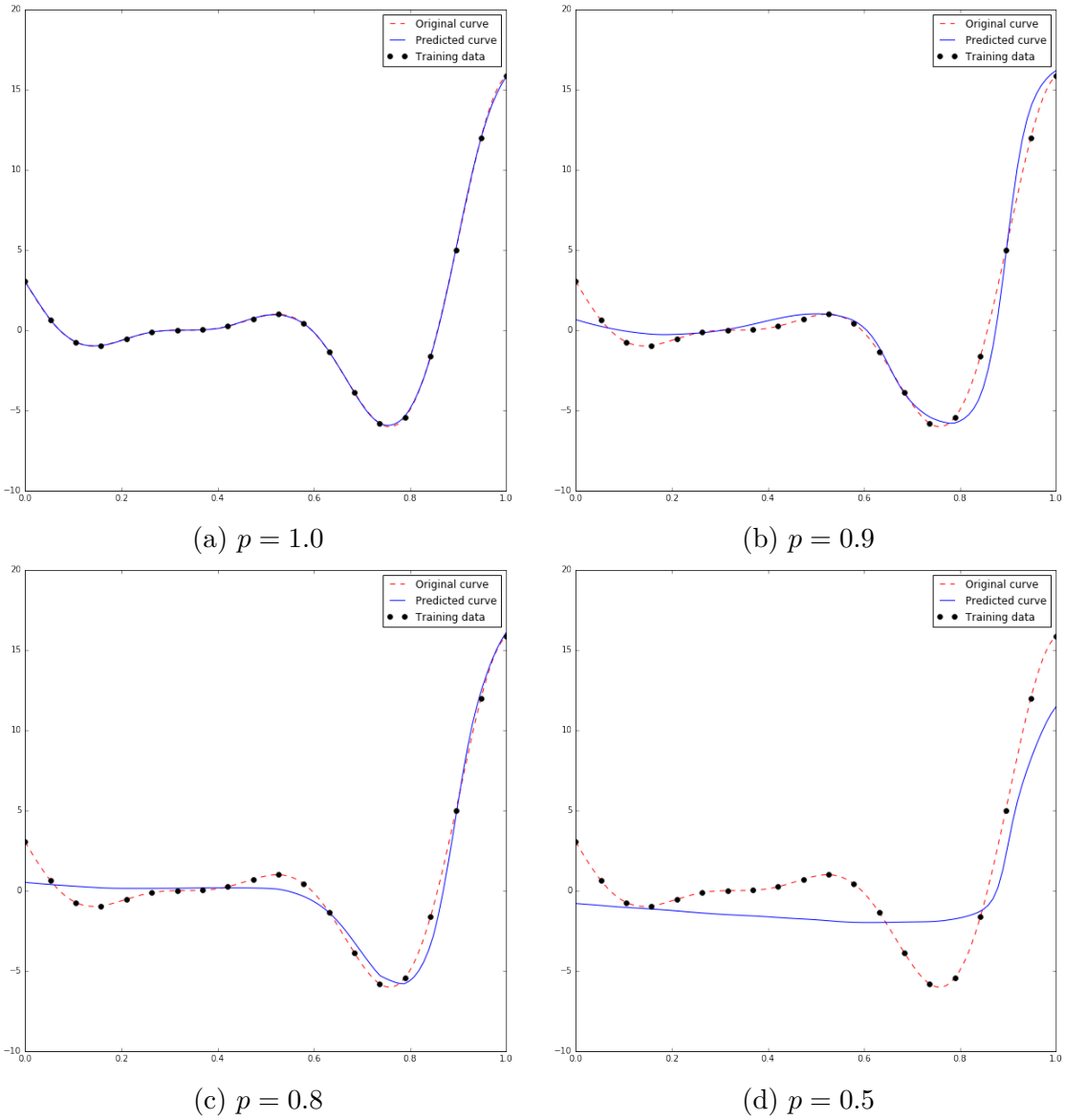
(a) 0.00001 (b) 0.0001 (c) 0.001

(d) 0.01 (e) 0.1 (f) 1.0

Figure 3.8: Comparison of different regularization factors.

3.1.6 Dropout

Srivastava et al. [12] introduced *dropout* as a method mitigating the NN's susceptibility to overfitting. When (sufficiently complex) NNs are trained long enough they start memorizing the training data. In dropout, a preset percentage of units is dropped stochastically in each iteration during training time. We use p to depict the keep probability and likewise $(1-p)$ for the dropout rate. Intuitively, dropping units encourages the NN to find alternative ways of comprehending the data. This makes the network more robust and allows better generalization. *Figure 3.9* shows the effect of dropout for different dropout rates compared to the case of no dropped units. A further decrease of the keep probability to for example 0.2 or 0.1 results in a fitting curve that gets close to linear.



(a) $p = 1.0$ (b) $p = 0.9$

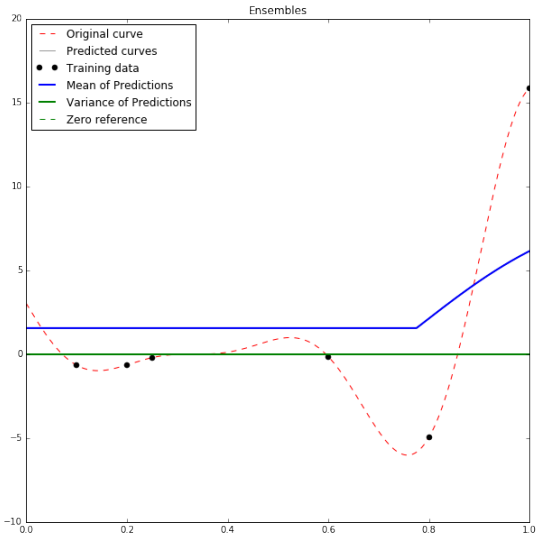
(c) $p = 0.8$ (d) $p = 0.5$

Figure 3.9: Dropout with different keep probabilities p.

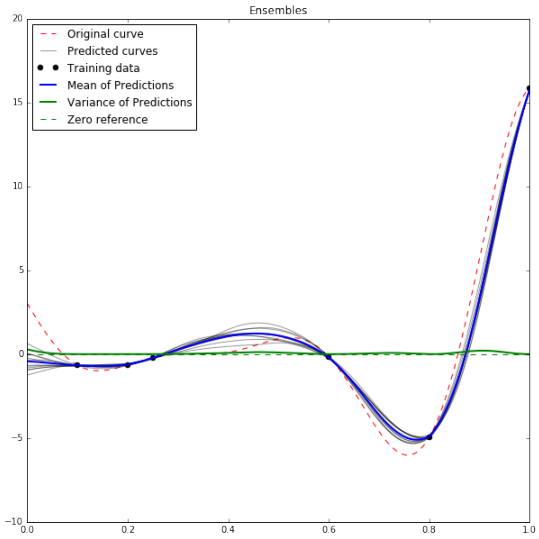
3.1.7 Weight and Bias Initialization

Weights and biases are typically initialized by stochastically drawn samples from a truncated normal distribution. The stochasticity is essential as networks initialized with the same constant value are not trainable. However, we can change the oscillation behavior of the regression curve between samples by scaling the underlying truncated normal distribution. This is especially useful for influencing the magnitude of the uncertainty values obtained from ensembles of NN (see *section 4.1.2*). *Figure 3.10* illustrates the effect of different random initializations on ten separately

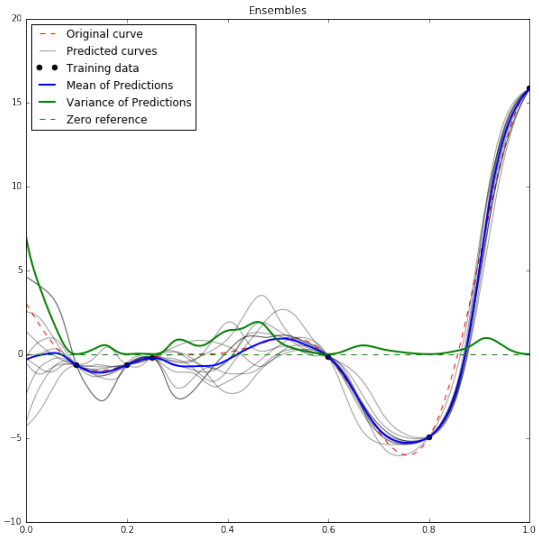
trained NNs with equal parameters but distinct scaling factors. The weights should be initialized sufficiently far away from 0, otherwise the NN loses fitting power and is attracted by local optima. The shape of the initial regression curve depends on the weight initialization and becomes smoother with smaller weight values. A plane corresponds to weight values of 0 and is not possible.



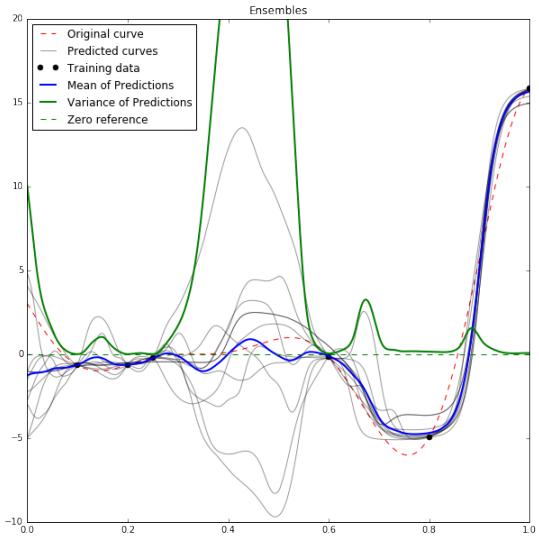
(a) Initialization with constant 0.1.



(b) Truncated normal with factor 0.5.



(c) Truncated normal with factor 1.5.



(d) Truncated normal with factor 2.0.

Figure 3.10: Comparison of different weight initialization factors.

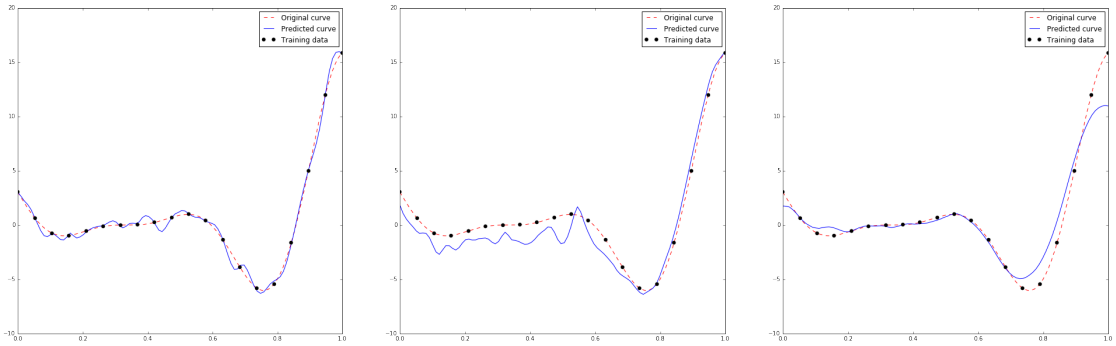
3.2 Over- and Underfitting

The goal is to obtain a regression curve that runs through all samples. This is commonly understood as *overfitting*, i.e. the data is being memorized. In this work however, we treat this as a *good fit* if furthermore the values in-between samples are filled in a smooth and approximately least distance manner. *Underfitting* is defined as a fit that fails to model the data sufficiently. We use the term in the same sense, however more strongly.

If the number of hidden units is chosen too large, the NN does not fulfill the smoothness constraint of good fits as can be seen in *figure 3.11a*. *Figure 3.11b* and 3.11c exemplify our results that smoothness cannot be recovered by using dropout and regularization.

We identified several factors that lead to over- or underfitting:

- How to overfit: Having an exuberant number of hidden units and scaling up the weight initializations.
- How to underfit: Having too few hidden units, setting the regularization factor too high, using dropout, prematurely stopping the training, scaling down the weight initializations and restricting to the use of sigmoid activations only.



(a) Using 2000 hidden units.

(b) Attempt of reducing overfitting using dropout.

(c) Attempt of reducing overfitting using regularization.

Figure 3.11: Analysis of an overfitting configuration.

3.3 Scaling of the Input Range

A rather surprising result is how strongly the input range affects the training success. *Table 3.2* presents the error after different training durations with the 1D benchmark function evaluated in the range $[0, 2]$. Before passing the samples to the NN, the x values were rescaled to different ranges.

Table 3.2: Training progress for differently scaled x value ranges.

Epochs	**Error**			
	$[0, 0.1]$	$[0, 1]$	$[0, 2]$	$[0, 4]$
0	633.73	623.77	700.47	196.65
6000	573.38	47.06	38.93	551.16
12000	543.81	5.90	3.64	530.42
18000	569.79	0.62	0.26	318.95
24000	397.53	0.068	0.037	72.32
30000	385.07	0.0087	0.0063	11.93

4 Results

This chapter presents the results of comparing different methods to obtain uncertainty, running optimizations, balancing exploitation and exploration and finally generalizing between optimization problems.

4.1 Uncertainty

We have implemented and evaluated several methods for obtaining uncertainty estimates. In this section, we will outline our results for each method and depict why the *ensembles of NN* method eventually yields the most promising results. We have evaluated all uncertainty methods in 1D due to visualization advantages. *Table 4.1* briefly covers the basic experimental setting.

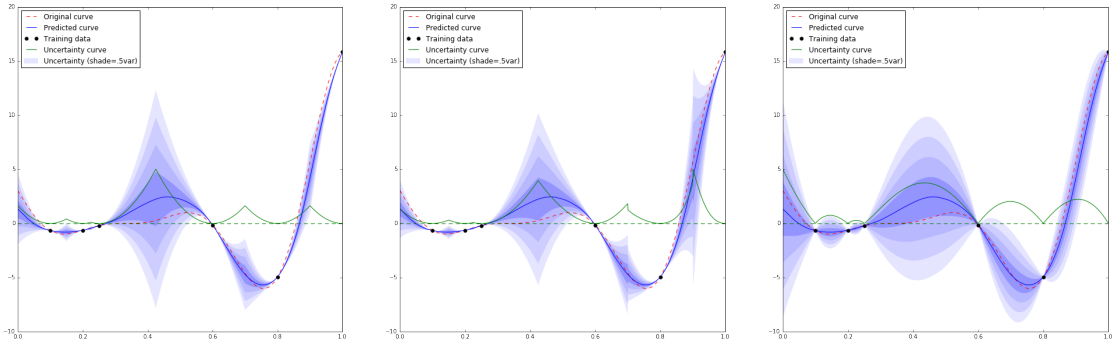
Table 4.1: Experimental setting.

Object	**Property**
Function	$f(x) = (6x-2)^2 \cdot sin(12x-4)$
Range	$[0, 1]$
Samples	$x_s \in \{0.1, 0.2, 0.25, 0.6, 0.8, 1.0\}$
Hidden Layers	2
Units per Layer	230
Initial Learning Rate	0.008
Regularization Factor	0
Epochs	5000

In all plots, the dashed red curve refers to the objective function, which we do not know in practical applications. The blue curve refers to the surrogate, which is the NN's prediction. The samples are displayed as black points while the uncertainty is pictured in two different ways: On the one hand, the green curve depicts the uncertainty at any given point x. On the other hand, the blue colored area also represents the uncertainty. Here, each shade corresponds to half a standard deviation. The uncertainty is normalized to a maximum value of $s(x) = 5$. However, this normalization only holds for this section for visualization purposes.

4.1.1 Computation of Uncertainty outside the Neural Network

In this subsection, we evaluate three different methods for uncertainty estimation that all share one common attribute: They are calculated outside the NN. Thus, the uncertainty estimates presented here do not rely on the surrogate model. We have evaluated two distance based measures, one based on the distance of two independent variables x_s and $x_{s'}$ and one based on the euclidean distance between two points $(x_s, f(x_s))$ and $(x_{s'}, f(x_{s'}))$. We will refer to the former method as *x-distance based* and to the latter as *euclidean distance based*. In addition, we have evaluated uncertainty estimation based on Kriging.



(a) x-distance based (b) Euclidean distance based (c) Kriging

Figure 4.1: Uncertainty estimates based on three different methods that operate independent of the NN.

X-Distance based Method

The x-distance based method is based on the fundamental assumption that uncertainty rises with increasing distance to samples. For example, when evaluating a point x in the immediate distance of a sample, the model's uncertainty would much likely be lower than when evaluating point x with great distance to any sample. We have ultimately implemented *equation 4.1* to formalize this behavior:

$$s(x) = (x - a)^2 \tag{4.1}$$

The variable a refers to the x value of the nearest available sample, x being the point that is currently evaluated. We square the resulting value in order to ensure that high distance results in extraordinary high uncertainty. *Figure 4.1a* visualizes the x-distance based method. The uncertainty peaks at $x = 0.425$ as this point constitutes the maximum distance of any point to any sample. The quadratic dependency of the uncertainty is clearly portrayed by the uncertainty curve.

Euclidean Distance based Method

The second method is based on the euclidean distance between the evaluated data point and its nearest sample[1]. In this section, each point is defined by two values: x and $f(x)$. The euclidean distance is the distance with respect to those values. We do not use any scaling. *Figure 4.1b* shows a visualization of the euclidean distance based method. The difference to the distance based method can be observed at around $x = 0.9$, where the euclidean distance based value is significantly higher. The discontinuities (especially visible at $x = 0.7$) result from the fact that the evaluated point's nearest neighbor switches at exactly this point.

Kriging

As an alternative method for calculating uncertainty we have investigated Kriging, which is a method for inter- and extrapolation. The estimates are modeled by a Gaussian process with cubic autocorrelation assumption. We use the root of the mean squared error as our uncertainty estimate. Note that we do not use the prediction calculated by Kriging, but only its uncertainty estimates. The prediction is still calculated by the NN. *Figure 4.1c* shows a visualization. Further note that the uncertainty distribution in between samples is much smoother compared to the distance based methods.

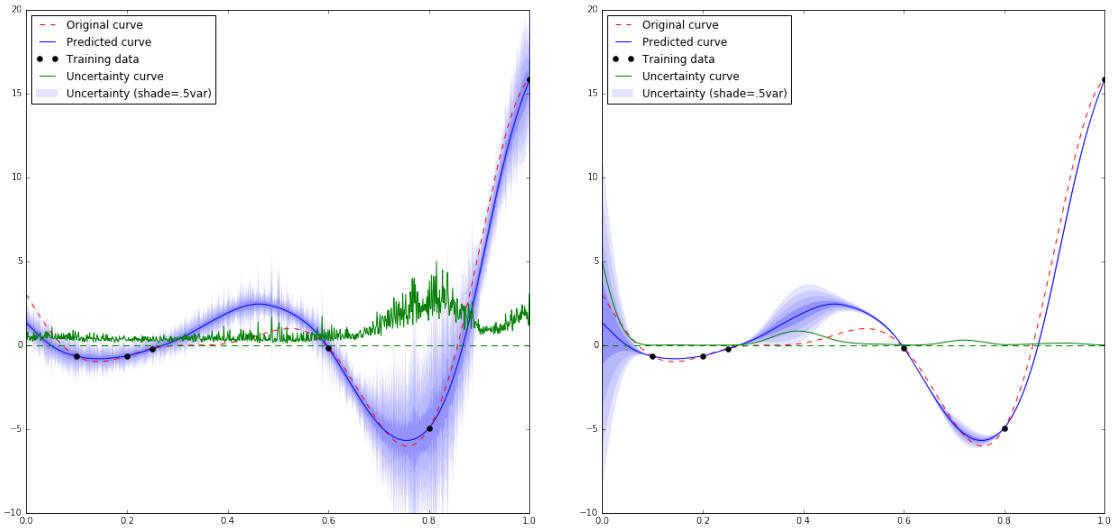
4.1.2 Computation of Uncertainty based on the Neural Network

We have also implemented two uncertainty estimation methods that are based on the NN. Thus, those methods possess an internal dependency to the surrogate.

MC Dropout

First, we implemented uncertainty estimation based on Monte-Carlo Dropout as proposed by Gal and Ghahramani [3]. Its basic idea is the approximation of a deep Gaussian process by applying dropout in test instead of training time. Dropout is normally used during training time to prevent overfitting (see *section 3.1.6* for further details). If dropout is applied multiple times during test time, the NN returns several different (but similar) results. As proposed, we take the mean of those results as our predictive value and its variance as an uncertainty estimate. *Figure 4.2a* visualizes the MC Dropout based method. Note that the obtained uncertainty distribution fundamentally differs from any previous method.

[1] Note that the nearest sample is calculated upon x-distances (not euclidean distance). This is supposed to maintain a proper sequence of the independent variables.



(a) MC Dropout (b) Ensembles of NN

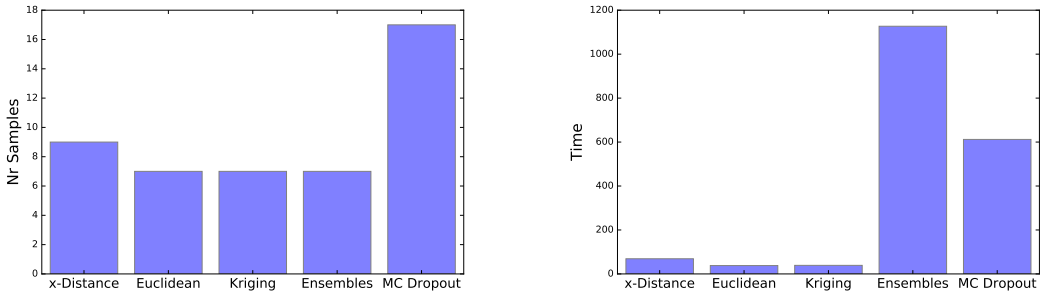
Figure 4.2: Uncertainty estimates based on two different methods that built on the NN.

Ensembles of Neural Networks

As a second method that calculates uncertainty within the neural network we have implemented *ensembles of NN* as proposed by Jiang [6]. In this method, several NN with different random weight initializations but otherwise identical are applied to the same problem. After each network has returned its prediction, the mean over all predictions constitutes the final prediction. The variance depicts the uncertainty estimate. *Figure 4.2b* shows a visualization of the ensemble based method.

4.1.3 Performance Indicators

As our chief performance criterion we evaluated the required number of samples for exploitation and exploration when utilizing the different uncertainty generating methods. *Figure 4.3* shows the results when they are applied to the function defined in *table 3.1*: The euclidean distance based method, Kriging and ensembles of NN all share the first place. Second is the x-distance based method while MC Dropout is left behind. We also recorded the execution times: The euclidean distance based method and Kriging win. Ensembles of NN takes 30 times longer. MC Dropout is 4.4 times faster than ensembles when measuring the time per sample instead of total time.



(a) Benchmark of samples needed (b) Benchmark of execution time needed

Figure 4.3: Benchmark of different uncertainty methods for optimization.

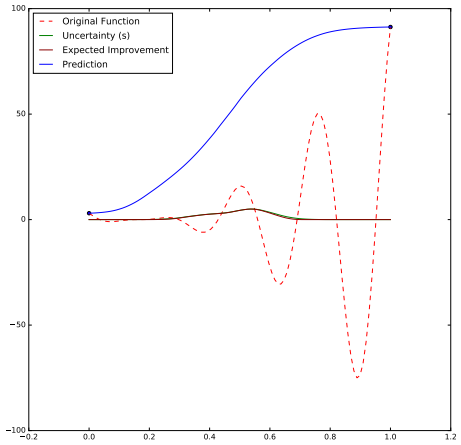
4.2 Exploration and Exploitation

We have implemented exploration and exploitation in 1D and 2D using the two-stage approach as presented in *chapter 2*. This section focuses on the results we have obtained testing our implementation in different scenarios. In every scenario, we use the ensembles method to estimate uncertainty. In 1D, we place two initial samples at the respective end of the interval. In 2D, we place four initial samples at the respective corners of the 2D grid. In all plots, the dashed red curve refers to the objective function which we normally do not know. The blue curve corresponds to the prediction of the network while the green curve portrays the uncertainty. The dark red curve depicts the expected improvement. Uncertainty and expected improvement are scaled so that each reaches its maximum at $f(x) = 5$.

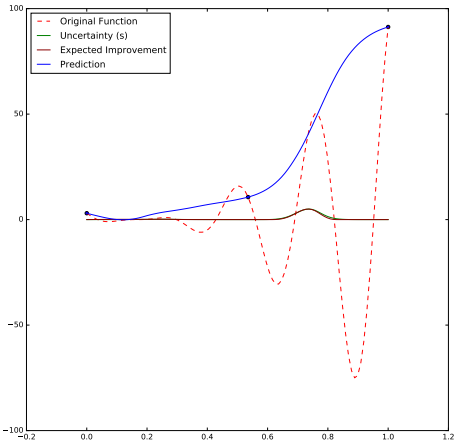
4.2.1 1D

Table 4.2: Experimental setting.

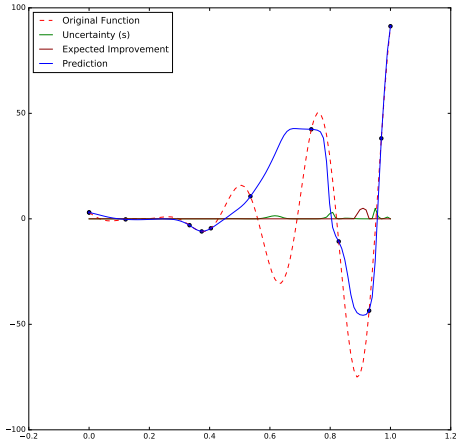
Object	**Property**
Function	$f(x) = (6x-2)^2 \cdot sin(12x-4); x \in [0,2]$
Range	$[0,2]$ (scaled to $[0,1]$)
Hidden Layers	2
Units per Layer	170
Initial Learning Rate	0.01
Regularization Factor	0.00005
Training Epochs Prediction	30000
Training Epochs Uncertainty Estimation	30000
Number of Ensembles per Estimation	10
Optimizer	Adam



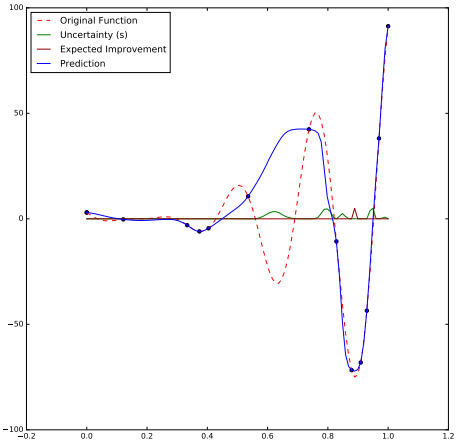
(a) 2 samples (squared calibration of s)



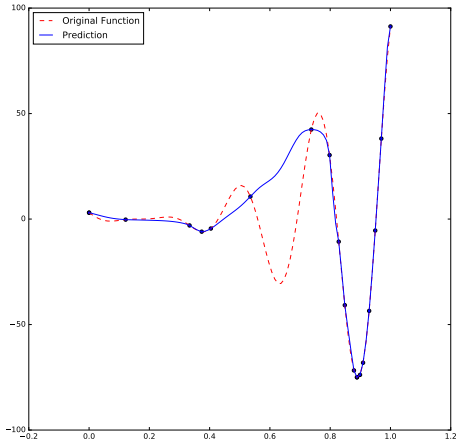
(b) 3 samples (squared calibration of s)



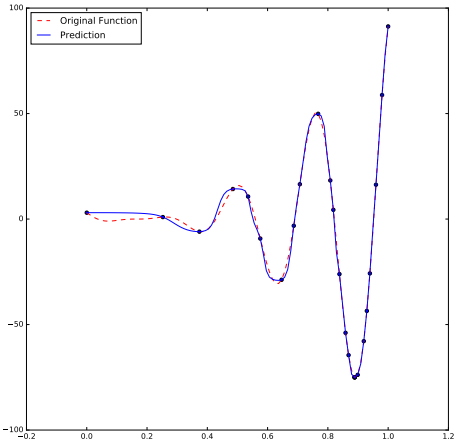
(c) 12 samples (squared calibration of s)



(d) 14 samples (squared calibration of s)



(e) Final state after 19 samples (squared calibration of s)



(f) Final state after 25 samples (**cubic** calibration of s)

Figure 4.4: Intermediate and final optimization results in 1D.

We present two different scenarios in 1D, both scenarios differ with regard to the normalization of the uncertainty s. As we do not use the proposed methods by Forrester et al. to compute uncertainty, we use s as a hyperparameter to balance exploration and exploitation in *equation 2.2*. In all scenarios, we use the function that is depicted in *table 4.2*. This function comprises both high and low amplitude oscillations, and might therefore be difficult to approximate. Note that the x-range is scaled to $[0, 1]$ to satisfy specifications of the neural network that have been laid out in *chapter 3*. We do not scale back to the original space as the calculation of the minimum is easily computable and readable in the transformed space. *Table 4.2* provides an overview regarding the setup.

We have implemented the following termination criterion consisting of two elements:

(1) After each iteration, the minimal $f(x)$ over all samples is retrieved. If this value does not significantly change over the course of two iterations, the first component of the termination criterion is fulfilled.

(2) If two consecutive samples do not differ significantly with regard to their x value, then the second component is fulfilled.

The termination criterion is based on the assumption that the network converges if two conditions hold: On the one hand, it does not provide significantly better solutions with more iterations (condition (1)), and, on the other hand, it has stopped exploring (condition (2)).

Figure 4.4 illustrates the exploration and exploitation process. Note that *figure 4.4f* is based on a different calibration of s in contrast to the other plots.

First scenario: *Figure 4.4a* shows the initial state with two initial samples at 0 and 1. Due to the low amount of samples, the prediction differs tremendously from the objective function. Both, the expected improvement and the uncertainty have their peak at around $x = 0.5$ which roughly constitutes the mean of the two initial samples. As only little information is available, exploration at high uncertainty seems like a reasonable decision. *Figure 4.4b* portrays the same scenario with one additional sample. Basically, the NN is still uncertain regarding the properties of the objective function. Thus, further exploration is suggested. *Figure 4.4c* displays the scenario after twelve samples. After having explored sufficiently, the expected improvement now suggests to exploit at the surrogate's minimum. *Figure 4.4d* displays the scenario after fourteen samples. This is the first time the real minimum is approximately hit. However, as the NN does not obtain any information regarding the objective function, the network is at this point not sufficiently sure to claim the

discovery of the desired minimum. Finally, *figure 4.4e* shows the state after convergence. The accuracy[2] of the found minimum equals 100%, meaning that the NN has found the true minimum of the objective function.

Second scenario: The first scenario shows that the network is able to find the minimum of the given function with high certainty. However, its surrogate still differs significantly from the objective function. Therefore, we want to use the hyperparameter s to increase exploration. We hope that this will result in a more proper surrogate with respect to the objective function. *Figure 4.4f* displays the final state after 25 samples with *cubic* calibration of s. Besides achieving an accuracy (with regard to the found minimum) of 100%, it has achieved a surrogate that is visually very similar to the objective function. This claim is supported by the test errors of the two scenarios. While the scenario with squared calibration yields a final error of 145.7, cubic calibration leads to an error of 2.4. We compute the test error using 500 equidistantly spaced true function values.

In summary, this subsection shows that our approach to exploration and exploitation works successfully on a difficult function in 1D.

4.2.2 2D

Table 4.3: Experimental setting.

Object	**Property**
Function	$f(x) = (4-2.1x_1^2+\frac{x_1^4}{3})\cdot x_1^2+x_1x_2+(-4+4x_2^2)\cdot x_2^2$
Range	$[0, 1]$ in each dimension
Hidden Layers	2
Units per Layer	170
Initial Learning Rate	0.01
Regularization Factor	0.00005
Training Epochs Prediction	2000
Training Epochs Uncertainty Estimation	2000
Number of Ensembles per Estimation	10
Optimizer	Adam

This subsection focuses on our multidimensional implementation for the case of 2D. As an objective function we use the Six-Hump-Camel function as displayed in *table 4.3*. For each dimension, we consider a range of $[0, 1]$. Initially, we sample at the four grid corners. *Table 4.3* provides an overview regarding the NN setup. It is very

[2]The accuracy describes the relative distance of the found minimum to the real (global) minimum. If the accuracy equals for example 0%, then the found minima is the global maximum of the objective function. If the accuracy equals 50%, the found minima is half way between the objective function's global minimum and maximum. Thus, this method requires knowledge of the objective function. Here, this knowledge available as the underlying functions are given.

similar to the 1D case. Note that if was possible to reduce the number of training epochs to 2000 regarding the calculation of both uncertainty and prediction.

Figure 4.8 displays the state after 23 samples. The graph located at the top left displays the objective function while the top right portrays the surrogate. On the lower left the uncertainty is visualized while on the lower right the expected improvement is depicted. Uncertainty and expected improvement are calculated based on the surrogate that is displayed in the upper right. Therefore, they constitute the basis for the next iteration. In this case, the next iteration equals the final iteration as the NN would once again sample at the minimum value.

4.3 Generalization

As briefly outlined in the introduction we generalize between optimization problems by considering a related class of n-D optimization problems that bear a shared $(n+1)$-th dimension. We do so by constructing and maintaining a $(n+1)$-D surrogate model and optimizing n-D *slices* while profiting from previously optimized slices. For benchmarking, we set $n = 1$ and interpret optimization problems to be slices along the x_1 axis of the Six-Hump-Camel function. The x_2 axis represents the newly introduced dimension, i.e. the generalization axis. *Figure 4.5* gives an example with slice $x_2 = 0.86$ highlighted in blue.

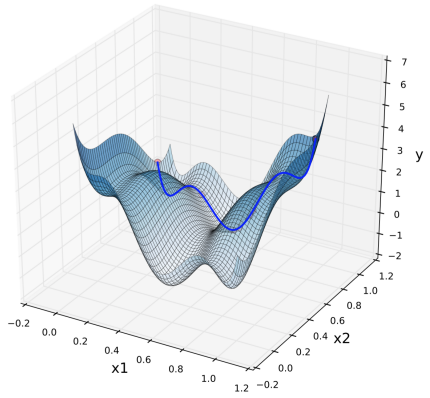


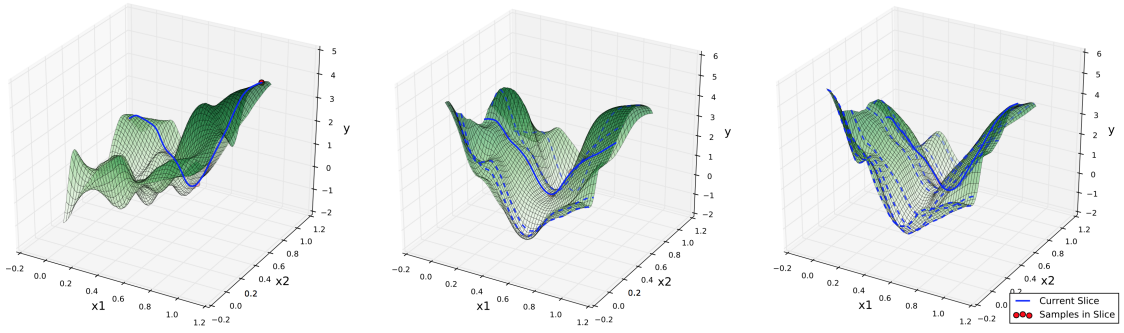
Figure 4.5: A slice visualized in the original Six-Hump-Camel function.

The surrogate model expands over the entire Six-Hump-Camel function. Then, we simulate real life usage by optimizing one slice after the other while updating the overall surrogate model. At the same time, we continue sampling. Prediction and expected improvement are evaluated for the current slice only and advise the next sampling location. *Figure 4.6* shows different states of the surrogate considering various amounts of slices that have already been optimized. Because the surrogate becomes increasingly accurate, more prior knowledge is transferred to new slices.

Thus, the sampling process within those new slices is usually more efficient. This reduces the number of samples per slice needed. We have repeated the subsequent optimization of ten randomly chosen slices five times. *Figure 4.7* shows the amount of necessary samples over the slices that have already been optimized. We see that with prior knowledge of two evaluated slices the demand for new samples has almost halved from 14.4 to 7.8. As the number of optimized slices further increases the sampling demand does not reduce as significantly as before. However, a slightly decreasing trend to a mean of 6.25 is clearly visible. This constitutes a reduction by 57% compared to optimization without generalization.

The time needed to train the NN depends on the number of samples taken. Running ensembles of NN on the Six-Hump-Camel takes about 275s for ten ensembles and fife samples on a 2.6GHz Intel Core i5 with 8GB 1600MHz DDR3 memory. The same setup takes already about 600s for 30 samples.

Note that our implementation has a memory leak depending on the number of previously trained NN instances. Moreover, the performance starts to decrease even before the memory is exhausted, independent on the current number of training samples.



(a) 1 slice (b) 5 slices (c) 10 slices

Figure 4.6: The surrogate model after different numbers of slices optimized.

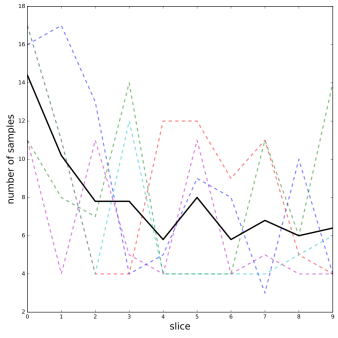


Figure 4.7: Number of samples needed per slice plotted against number of already optimized slices.

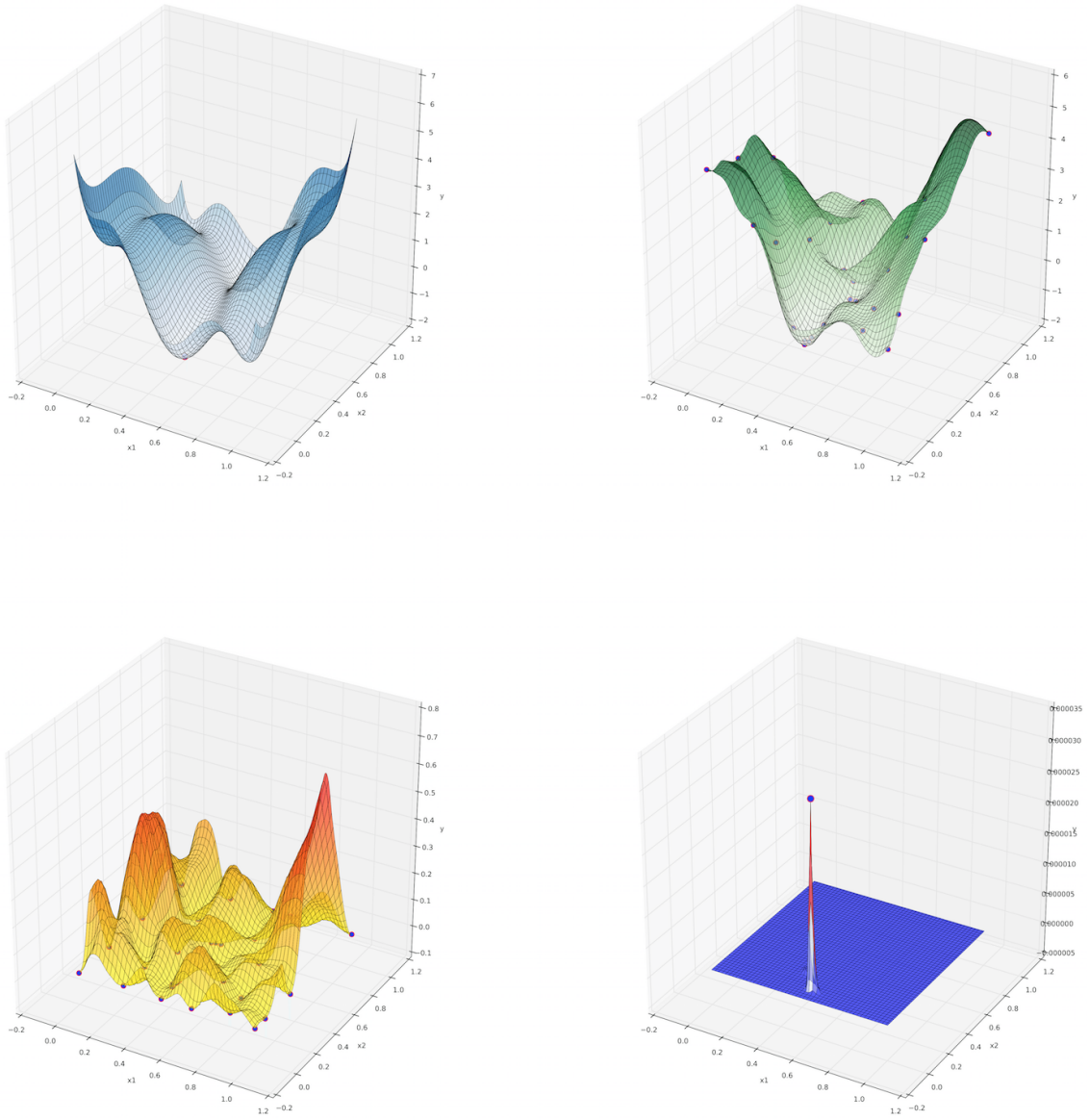


Figure 4.8: 2D optimization after 23 samples.

5 Discussion

In this chapter we summarize our major conclusions, highlight some caveats and point out possible future work.

5.1 Regression

In *section 2.2* we have outlined that NNs (theoretically) achieve excellent fits with regard to nonlinear regression problems. However, this needs to be regarded with care, especially under time and computational constraints. Furthermore, we have outlined a number of vital parameters. Those parameters depend on the given function and can ideally be found using cross-validation. Practically, this is impossible in a surrogate optimization setting, as the underlying function is unknown and only a limited amount of samples are attainable. Additionally, cross-validation needs even in its simplest form extreme amounts of computational resources, and every round should be run repeatedly to increase its robustness with regard to initialization noise. Consequently we chose our final configuration based on the gained experience. If confronted with a new optimization task, we recommend gaining an intuition for the required configuration in related problems. For some functions, the parameter finding process is simple (e.g. the Six-Hump-Camel). For others, it is rather tedious, e.g. the 1D function as defined in *table 4.2*. In fact, the same NN configuration that needs 30000 epochs for the 1D function can fit Six-Hump-Camel in 2000 epochs. We assume that the former function is more difficult as its amplitude of the oscillation changes strongly from a y-spread of about 5 to about 170. On the contrary the Six-Hump-Camel resembles a slightly bent surface instead of a sinusoidal.

We use the Adam optimizer for two reasons: Firstly, it results in a slightly smoother fit which is desirable for us. Secondly and chiefly, the learning rate is the only parameter that we have to tweak. Gradient descent, however, has to be parameterized concerning the weight decay and frequency.

We assume a scenario with noise free data. That means we expect the prediction curve to precisely run through all data points and furthermore to be smooth in between points. That is why we decide against using dropout and regularization[1]. We still recommend using regularization in real life applications as those are always

[1]Note that the chosen regularization factor is small enough to not let us observe any change.

prone to noise. Depending on the type of data, dropout or regularization might perform better: When working with NNs there is no bypass around error and trial experiments.

In *section 3.1.1* we have looked into combinations of differently activated layers. We have chosen ReLUs for the first layer to achieve the necessary latitude when fitting the data. For the second layer we have chosen sigmoid activations, as those smoothen the fit. Alternatively, more complex activations such as polynomial ($\psi(a) = a^k,\ k \in \mathbb{N}_+$) or exponential functions ($\psi(a) = e^a$) can be utilized. The radial basis function network is a popular representative of specialized NNs which performs more efficient on the task of curve fitting, yet it fails to extrapolate or interpolate between far spaced samples: The prediction has its default at 0, thus it converges towards this value far away from any sample. Another approach to ensembles of NN is running different architectures in juxtaposition and unifying them in a combining neuron. Custom tailored solutions such as ensembles with polynomial activations (that include sinusoidal or ReLU elements) could prove to be strong in fitting certain classes of functions, but also increase the model complexity.

We close the discussion about regression with NNs by highlighting a rather unusual discovery. Typically, NNs thrive when the underlying patterns are too difficult to learn for other machine learning algorithms. They then require exceptional amounts of data for training. We have analyzed quite the opposite scenario: Fairly simple functions with extremely little data. However, this might be exactly the aspired perfect match, as samples are expected to be very costly in surrogate optimization.

5.2 Uncertainty

To analyze the peculiarities of the different uncertainty computing methods, we first need to define the uncertainty estimate's specification:

- The uncertainty should be 0, where we have samples.
- The uncertainty should be unequal 0 everywhere else.
- The further away from a sample, the higher the uncertainty should be.
- Furthermore, we want it to be defined everywhere.

We group the methods into whether they have been computed using the NN or are only relying on the samples. Former group naturally has a higher correlation with the NN and its surrogate than latter. But this bonus in information ships with a computational price tag. MC Dropout is the only method that does not adhere to the specification.

We have ultimately decided to utilize the ensembles of NN method as its results contain more information than the distance based methods and Kriging: The resulting

uncertainty does not only depend on the distance between the samples but also on the curvature of the surrogate. Furthermore, weighted combination of different methods constitutes an alternative approach that we have only briefly considered so far. For example a distance based method could be inoculated with the surrogate's curvature information inherent in MC Dropout. This could simulate the results of the ensembles method.

Before we go into details of each method we will outline a general problem: In all cases we compute the uncertainty on all points in a sufficiently dense grid. If we have a very highly dimensional optimization problem, it becomes unfeasible to evaluate every grid point, as the grid size grows exponentially. Thus, we could employ stochastical optimization methods such as simulated annealing or tabu search to find local and global optima.

5.2.1 Distance based Methods

The distance based methods are dependent on the scaling of each dimension. Therefore, one major drawback of those methods can be seen in the possible necessity of proper scaling. The peakiness of the resulting curves might be too dictatorial when it comes to balancing exploration and exploitation. The peak has an unproportionally high chance of being elected as best new sampling location compared to the locations surrounding it. However, distance based methods are easy to calculate and do not require any knowledge other than the sample's positions.

5.2.2 Kriging

Kriging impressed with efficient calculation and constitutes our second favorite candidate for calculating uncertainty. Contrary to the peaky estimates of the two distance based methods its uncertainty estimates between samples are distributed much more evenly. Thus, those estimates could allow more flexibility in later exploitation and exploration.

5.2.3 MC Dropout

MC Dropout estimates the uncertainty inherent in the NN's fit and impresses with low computational costs as only the prediction needs to be run repeatedly, and not the training. However, dropout during test time might result in misleading uncertainty estimates with regard to the setting introduced in this report: Those uncertainty estimates increase proportional to the damage by the dropped units. Thus, data regions with robust fit are assigned low uncertainty. In opposition are regions where the prediction is based on a scarce number of neurons: Dropped units

will create more damage leading to higher uncertainty. This seems to be mostly the case in regions with higher curvature of the function. This does not have to indicate any need for exploration, therefore high uncertainty estimates in those regions (and low estimates in respective other regions) might be misleading. For example, those regions could have already been explored sufficiently while regions with less curvature lack exploration. *Figure 4.2a* might illustrate uncertainty estimates of MC Dropout that lack usefulness in the context of exploitation and exploration.

5.2.4 Ensembles

The benchmarks described in *section 4.1.3* show ensembles uncertainty estimates performing best along with euclidean distance based uncertainty and Kriging. However, as it involves retraining the NN repetitively it is also the most expensive method in our arsenal. In this method, uncertainty increases with rising distance from samples and equals 0 at sampled points. In addition, the surrogate's curvature is incorporated: The stronger e.g. high latitude movements, the less the ensembles fluctuate. *Figure 3.10* illustrates this behavior.

5.3 Generalization

The underlying pipeline is modular, so the methods computing uncertainty, the surrogate model or the expected improvement can simply be substituted. The defaults are ensembles of NN for uncertainty, a nonlinear regression NN generating the surrogate and expected improvement that is evaluated using the two-stage approach. In *section 4.3* we have exemplified that using this setup, we can reduce the number of samples significantly by a percentage of 57% in the context of generalization. However, this result is by far not yet optimal for at least the following reasons:

- The scaling of the uncertainty balances exploration versus exploitation and furthermore puts the magnitude of the uncertainty and regression curve in proportion. We have not yet examined the scaling parameter exhaustively. Besides, substituting the two-stage approach with an alternative method might make especially the proportion matching issue redundant. A promising candidate would be the one-stage approach as outlined by Forrester et al. [2] which takes a structurally different approach to adaptive sampling.
- The implemented convergence criterion awaits a three times repeated sampling in the same location. These two surplus samples could potentially be spared. Incorporating knowledge gained from the expected improvement curve such as maximum amplitude or number of peaks would be one approach to do so.

The performance bottleneck of the program is the uncertainty computation using ensembles of NN. However, for several reasons this constitutes not a massive obstacle: First, the execution of one NN can be parallelized and run on specialized hardware, e.g. CUDA cards. Second, the simulation of ensembles of NN can be run fully concurrently. Third, obtaining the samples can be assumed to be more expensive than running the optimization process. In addition, we can always decide to replace it with one of the other presented methods. The final choice in practical application should depend on experiments with real life benchmarks similar to the field of action.

6 Conclusion

We have achieved a reduction in the number of required samples for optimization tasks by generalizing over a meta-model that inherits information from previously solved similar problems. Specifically, we have first successfully configured a NN for nonlinear regression. We then evaluated five methods of obtaining the NN's uncertainty, including two distance based methods, Kriging, MC Dropout and ensembles of NN. The last-mentioned method yields the most promising results as its uncertainty estimates are especially suitable for the chosen exploration exploitation balancing approach: We implemented the two-stage approach and used it in 1D and 2D optimization tasks. We showed that our implementation reaches an excellent accuracy with respect to the found minimum. Also, we have shown that the hyperparameter s balances exploration and exploitation and can be chosen based on the user's goal. Such goal might be finding the global minimum or constructing a surrogate maximally similar to the objective function. Finally, we have used the NN to generalize over different 1D optimization problems. Using a continuously adjusted meta-model we show that similar optimization problems can be solved faster using previously gained experience. Next to our main goal, we have shown that MC Dropout is not suitable as an uncertainty estimator for nonlinear regression with zero observational noise, if the estimates are to be used for exploitation and exploration.

One downside we have found is that the NN's configuration seems to be problem-specific, at least to some degree. Thus, proper configuration depends on the user's expertise and requires additional time. A good fit is still theoretically always possible as NNs generally are universal function approximators.

As another downside we have found that uncertainty estimation using the ensembles method requires extensive computational resources. To cope with this restriction, the Kriging based method for uncertainty estimation might be a good alternative. It is already used by us in a 1D case and can be scaled to higher dimensions. Therefore, we encourage future research to either test Kriging in higher dimensional cases, to significantly lower the computational costs of the ensembles method or to combine several methods.

Furthermore, future research might focus on increasing the number of dimensions beyond two. Ultimately, high dimensionality will be closer to the complex problems that applicants face. Scaling to higher dimensionality exponentially increases the

search space. Therefore, efficient stochastic or heuristic optimization techniques might be helpful to lower computational costs.

Bibliography

[1] Adkins, C. N. and Liebeck, R. H. (1994). Design of Optimum Propellers. *Journal of Propulsion and Power*, 10(5).

[2] Forrester, A. I. and Keane, A. J. (2009). Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45:50–79.

[3] Gal, Y. and Ghahramani, Z. (2015). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. stat.ML.

[4] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366.

[5] Jeong, S., Murayama, M., and Yamamoto, K. (2005). Efficient Optimization Design Method Using Kriging Model. *Journal of Aircraft*, 42(2).

[6] Jiang, Y. (2007). Uncertainty in the Output of Artificial Neural Networks. Orlando, Florida, USA.

[7] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[8] Queipo, N., Haftka, R., Shyy, W., Goel, T., Vaidyanathan, R., and Tucker, P. (2005). Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41:1–28.

[9] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[10] Selmic, R. R. and Lewis, F. L. (2002). Neural-network approximation of piecewise continuous functions: application to friction compensation. *IEEE Transactions on Neural Networks*, 13(3):745–751.

[11] Simpson, T., Toropov, V., Balabanov, V., and Viana, F. (2008). Design and Analysis of Computer Experiments in Multidisciplinary Design Optimization: A Review of How Far We Have Come - or Not. Victoria, British Columbia, Canada.

[12] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.