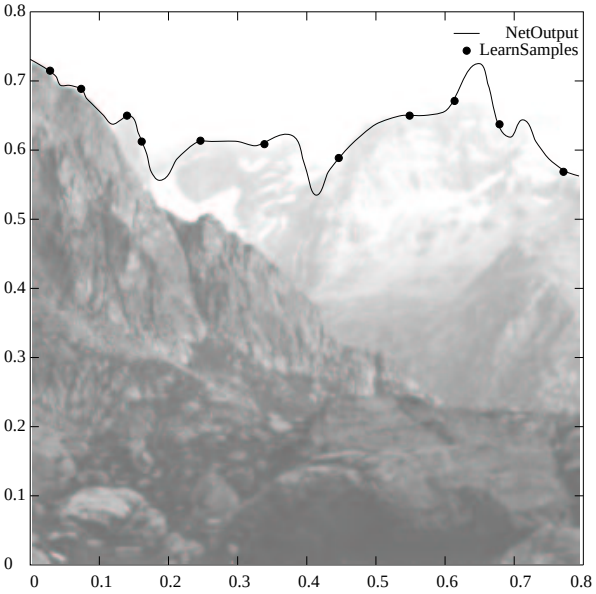
Locally weighted approximations:
yet another type of neural network

Sander Bosman

Intelligent Autonomous Systems group
Department of Computer Science
University of Amsterdam

July 1996



Abstract

This Master's thesis discusses locally weighted approximations, which can be viewed as a particular kind of neural network for approximating smooth functions. In such a network, the input space is divided into overlapping regions; each region is approximated by a local expert. Various problems are discussed: how many experts to use, what kind of regions to use, what is the effect of the local expert approximation, what is the effect of the number of learning samples and how to optimize the parameters. Experiments were done to compare some of these choices. Other experiments indicate the performance of locally weighted approximations as compared to feed forward neural networks and radial basis functions.

Acknowledgments

The thesis you are reading is the result of my research performed within the Intelligent Autonomous Systems Group at the University of Amsterdam. It would not have been possible to finish this without all the people who supervised, helped or just talked to me. I would like to thank them all, although I know that only a small number of them are listed here.

I got the honour to work with three supervisors during this period. I would like to thank my second supervisor, Prof. Frans Groen, who was very good at transferring his enthusiasm to me. Most of the time we had firm discussions about the subject; these were really nice and I have to admit that, looking back, he was right most of the time.

My third supervisor was Dr. Ir. Ben Kröse. I would like express my gratitude to him for stressing the importance of detail (which I sometimes tended to forget), and for his remarks on the numerous versions of my thesis; he always found time to read and comment on it.

Then, of course, I want to thank Dr. Huseyın Hakan Yakali for his never declining interest, all the long conversations, all that time he spend on looking at my results, giving me hints, coming up with interesting papers and just for being there every day.

I also want to thank the other staff: PhD students Joris van Dam, Anuj Dev and Stephan ten Hagen for for making useful remarks and providing me with interesting papers and programs; and Dr. Ir. Leo Dorst, who always had a refreshing and clarifying view on things during the student meetings.

Furthermore, I am thankful to all the students who were here the last year. In particular, Sirous Kavehercy, for leading the way towards graduation and warning me for all the pitfalls I could encounter; and Albert van Breemen, for those never ending discussions about the 'larger goal'. Naturally, I don't want to forget the guys of my indoor soccer team "*Dynamo Humm*", formed by a group of fellow teaching assistants: Wout van Albada. Mischa Bronstring, Robbert Heederik, Eddie Niese and Sidney van der Poll. In addition, I want to thank Drona Kandhai.

The final thank you is for my first supervisor, who is certainly not skipped here: Dr. Patrick van der Smagt, who brought me here in the first place. We started with general discussions which were marvelous, probably because we're always talking on the same wavelength. This ultimately led to the specific subject of this thesis; all the time he kept encouraging and showed interest, even making remarks on my thesis long after he had left to Germany. Thanks Patrick, I really learned a lot from you.

Sander Bosman
Amsterdam, July 1996

Contents

Chapter 1

Introduction

Since the early days of mankind, man has always tried to find structure in the world in which he lives. Not so much out of curiosity, but just because survival depends on it.

The human is trying to figure out how the world behaves, and how to make it behave in a way which is beneficial for him. This has to be done based on limited information: only the information gathered by the senses in a relatively short period of time. Therefore it is quite possible that theories are formed, which do explain the information gathered, but are not generally valid. In practice however, humans seem to do pretty well most of the time.

The same problem arises when trying to build artificial intelligent systems. Here, a computer is used instead of the human brain, but the same goal is pursued: trying to find structure in the world, based on limited (sensor) information, and using this information.

One specific form of structure discovery is finding the effect some *input values* have on some *output values*. Both input and output values can be sensor information, measurements, etc.; in all cases the output values are assumed to be dependent on the input values, so the output can be predicted by the input values only. When a (limited) number of *examples* or *samples* is available (input values with their corresponding outputs), a possible relation can be found by *function approximation* (or *regression*). This method tries to find a mathematical function which 'fits' the given samples; the resulting function is then expected to generalize: it can predict the outputs of the samples, as well as the outputs for input values it has never seen.

Function approximation is practiced in different fields: numerical analysis, statistics and recently by *neural networks*, which were inspired by the network structure of interconnected neurons in the brain. Especially neural networks have induced great interest, since they are capable of finding rather difficult relations in the real world. A classical example is NETtalk, a neural network which found out how to read aloud written English text [25].

However, even neural networks are not perfect. Several variants have been proposed, which create better approximations in some situations, are faster or do have some other nice properties. Still, little is known about what methods should be used in what situations.

This Master's thesis studies *locally weighted approximations*, which form a particular type of neural network. Such networks are based on local regression methods found in statistics; [14] reports that local regression was used by Danish actuaries as early as 1829, but the first published work appeared in 1883 [13].

Locally weighted approximations come in a large variety, the reason why relatively little is known about them. This text focuses on a small subset: various alternative forms are compared and the best form found is compared with other types of neural networks.

The structure of this thesis is as follows. Chapter 2 discusses the basics of function approximation. Chapter 3 introduces locally weighted approximations, showing various choices which have to be made to get a specific form. Then, chapter 4 compares some choices by experimenting with them and looking at the resulting approximations. The optimal choices found in this chapter are used in chapter 5, which compares the 'best' locally weighted approximation network with two other existing networks: feed forward neural networks and radial basis functions. Chapter 6 gives conclusions and lists unsolved questions to be answered by future research.

Chapter 2

Approximating smooth mappings

2.1 Function approximation

Consider a function $\mathbf{y} = \mathcal{D}(\mathbf{x})$, which maps an input vector $\mathbf{x}$ onto an output vector $\mathbf{y}$. The analytical form of this function is unknown; it can be an abstraction of some physical process, like tomorrows temperature ($\mathbf{y}$), given last year's temperatures ($\mathbf{x}$). We *do* have access to a set of *samples* obtained from this mapping, which is a set of $(\mathbf{x}, \mathcal{D}(\mathbf{x}))$ pairs[1]. Samples usually contain *noise*, which follows from the way they are generated (e.g. by non-perfect sensors).

Our goal is to restore the function $\mathcal{D}(\mathbf{x})$, given only the set of (noisy) samples. Of course it is not possible determine this uniquely, since the samples give only limited information about the function, therefore the best we can do is to make an *approximation* $\mathcal{N}(\mathbf{x})$. To do this, a $\mathcal{N}(\mathbf{x})$ has to be found which is a good approximation of the available samples; the function found can be expected to be a good approximation of $\mathcal{D}(\mathbf{x})$ on the *whole* input domain under certain assumptions.

The goodness of fit of $\mathcal{N}(\mathbf{x})$ on a set of samples is given by an error function. A commonly used measure is the squared error

$$\mathcal{E} = \frac{1}{2} \int_{\mathbf{x}} (\mathcal{D}(\mathbf{x}) - \mathcal{N}(\mathbf{x}))^2 \tag{2.1}$$

which measures the difference between desired and actual output of every possible input vector. However, since $\mathcal{D}(\mathbf{x})$ is not known for all $\mathbf{x}$, we have to make an estimate of $\mathcal{E}$ based on the set of available samples S only:

$$\mathcal{E} = \sum_{s \in S} \mathcal{E}^s = \sum_{s \in S} \frac{1}{2} (\mathcal{D}(\mathbf{x}) - \mathcal{N}(\mathbf{x}))^2 \tag{2.2}$$

This error function is referred to as the *summed squared error* (SSE). Dividing this by the number of examples $|S|$ results in the mean squared error (MSE).

How to find a good approximation $\mathcal{N}(\mathbf{x})$? Usually $\mathcal{N}(\mathbf{x})$ is chosen to be a *parametric function*, where the parameters determine the exact shape of the function. Then these parameters can be optimized, to minimize the error on the samples (MSE). In approximation

[1] In the context of machine learning, sometimes the term *example* is used instead of sample, and the term *sample* is used to indicate a set of examples

theory this is called *parameter estimation*; in neural network terminology this is called *learning*. A large number of methods exist to find the (sub)optimal parameters for some given parametric function.

2.2 Generalization

2.2.1 Remembering is not enough

An approximation is not useful if it can remember the samples, but performs poorly on the rest of the input space. We want $\mathcal{N}(\mathbf{x})$ to *generalize* over the samples, using the samples to create a plausible approximation of $\mathcal{D}(\mathbf{x})$ for the complete input domain.

In the general case, this is impossible. For example, take $\mathcal{D}(\mathbf{x})$ as the 'telephone book function', mapping a person's name and address to his (or her) telephone number. There is no problem in creating a list of names and addresses with the corresponding telephone numbers (a set of samples), but it is not possible to extract the telephone number of a person not in this list: the samples simply do not give any information about other people's telephone numbers.

To make approximation possible on the complete input domain, the function $\mathcal{D}(\mathbf{x})$ must be *redundant* in the sense that a limited set of samples contains information about the rest of the mapping. Fortunately, in the real world this is often the case [15].

The generalization performance of $\mathcal{N}(\mathbf{x})$ can not be measured by the error on the samples used to optimize $\mathcal{N}(\mathbf{x})$'s parameters. Therefore, the set of available samples is commonly split into two sets: a *learning set*, used for optimizing the parameters, and a *test set*, used to get an indication of the generalization. Note that the error on the test set is just an *estimate* of the generalization performance, since it still does not measure the error on the complete input domain.

2.2.2 Smoothness

A large class of functions found in practice are the so called *smooth* functions. A smooth function has the following characteristics:

- The function is continuous (or has a limited number of discontinuities);
- Input vectors close to each other in input space are mapped onto output vectors close to each other in output space. Closeness can be measured by, e.g., Euclidean distance.

The mapping $\mathcal{D}(\mathbf{x})$ can be viewed as a $(n + m)$ dimensional landscape, where n and m are the dimensions of input vector $\mathbf{x}$ and output vector $\mathbf{y}$ respectively. For example, a 2-dimensional input and 1-dimensional output mapping can be seen as a 3-dimensional landscape (surface). Using this analogy, a smooth mapping does not have sharp peaks and valleys, and the slopes do not change suddenly.

Of course, there is not a sharp distinction between smooth and non-smooth functions: smoothness is a matter of degree.

We will restrict ourselves to smooth mappings, since they frequently appear in real life and they are redundant, making it possible to construct reasonable approximations.

2.3 Approximating smooth functions

Approximating a smooth function from a given set of samples means creating a mapping with the following properties:

1. The error on the learning samples should be as small as possible, since these samples are used to optimize the unknown parameters of $\mathcal{N}(\mathbf{x})$.

2. The approximation should be as smooth as possible, since $\mathcal{D}(\mathbf{x})$ is assumed to be smooth.

These two properties are contradictory. A very smooth approximation can not approximate the learning samples properly[2]: it has a high *bias*. On the other hand, approximating the learning samples perfectly compromises smoothness, and is not needed because the samples are noisy anyway: the approximation has high *variance*. Having a small error on the learn set but bad smoothness (and therefore bad generalization) is called *overfitting* (figure 2.1).

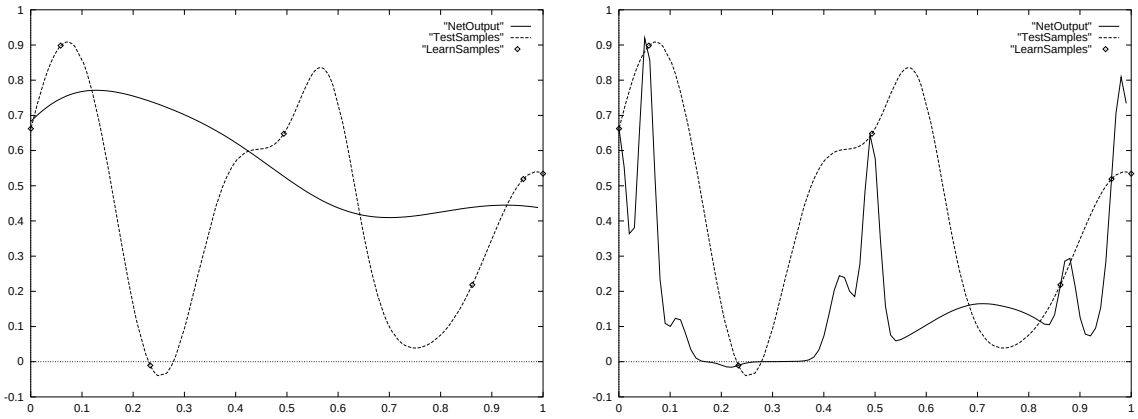


Figure 2.1: *Underfitting and overfitting on a smooth function.*

There is a tradeoff between having high bias and having high variance, which is commonly referred to as *the bias versus variance dilemma*. What the balance should be between these two aspects for one particular problem is not known beforehand. Only the error on the test set can give feedback about the effect of a particular choice (after optimization though).

2.3.1 Choosing the parametric model

Good performance of an approximation can be attained on both learning and test samples when the constraints of the previous section are satisfied:

- The parametric model is flexible enough to adapt to $\mathcal{D}(\mathbf{x})$: reaching a low bias is possible.

- The variance of the approximation is restricted.

[2]Unless, of course, $\mathcal{D}(\mathbf{x})$ is also extremely smooth.

The first constraint can only be satisfied by a good choice of parametric model. There are more possibilities of satisfying the smoothness constraint.

One thing that does usually *not* have influence on smoothness is the parameter estimation algorithm used. It only tries to minimize the error on the learning samples, and does not care about the approximation formed 'between' the learning samples. Therefore variance restriction has to be found in another area.

One possible way to restrict the variance is to limit the flexibility of the parametric model, so that the variance is limited no matter what parameters are found by the learning algorithm. The maximum variance possible for a parametric model depends on the number of parameters used, the *degree of freedom* of the model. Therefore the number of parameters should be kept as small as possible.

A good choice of parametric model is based on the a priori knowledge of $\mathcal{D}(\mathbf{x})$. More knowledge usually means that a more specific parametric model can be chosen having fewer parameters to adjust. For example, knowing that $\mathcal{D}(\mathbf{x})$ is linear dependent on $\mathbf{x}$ restricts $\mathcal{N}(\mathbf{x})$ to a linear model (which has few parameters to adjust).

2.3.2 Regularization

Sometimes the inherent variance restriction of a parametric model is not enough or not appropriate. In such situations it is possible to use *regularization*. Regularization penalizes approximations (settings of parameters) having high variance, by adding a *regularization term* to the error used in learning ([27], [12]):

$$\mathcal{E} = \sum_{s \in S} \frac{1}{2} \left(\mathcal{D}(\mathbf{x}) - \mathcal{N}(\mathbf{x}) \right)^2 + \lambda P\left(\mathcal{N} \right) \tag{2.3}$$

The first term is the error on the learn samples (the bias). The second term penalizes the variance of the approximation, where λ determines the balance between the two ($\lambda > 0$).

Regularization is a good and general way to achieve smooth approximations, but also has some disadvantages. The regularization term can be rather complex and computational expensive to calculate. Furthermore, choosing an appropriate regularization term is not trivial and depends on the a priori knowledge of $\mathcal{D}(\mathbf{x})$.

2.3.3 Effect of number of samples

Figure 2.2 displays the effect of the number of learn samples on learn- and test errors, obtained by optimizing the parameters of some parametric model (the same model is used every time):

Few learning samples. The learn error will be low, since the parametric model is flexible enough to approximate at least a small number of points well. The resulting approximation will usually not generalize very well: the information given by the learn samples is too small to get a reasonable overall reconstruction of $\mathcal{D}(\mathbf{x})$. So the test error will be high.

Large number of learning samples. Using a lot of learning samples results in a higher learn error, since the parametric model is not flexible enough to fit all the samples. On the other hand, the learning set gives a good indication of $\mathcal{D}(\mathbf{x})$ and therefore the approximation is a good generalization.

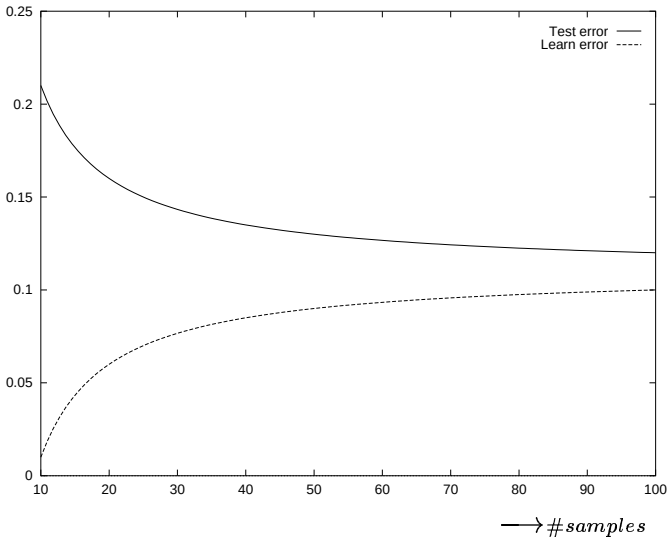


Figure 2.2: *Error on learning set and test set versus the number of samples.*

As could be expected, it is advantageous to have as many learning samples as possible. However, in practice this number is always limited due to time or space limitations, or simply because it is not possible to get more samples.

The actual number of samples needed is determined by the flexibility of the parametric function chosen. An also very important factor is the *dimensionality* of $\mathcal{D}(\mathbf{x})$, which is the sum of its input and output dimensions. Exponentially increasing numbers of samples are needed to densely populate Euclidean spaces of increasing dimensions (spanned by the input and output vectors), commonly referred to as the *curse of dimensionality*. Therefore high-dimensional mappings are very hard to approximate, unless there is a priori knowledge which simplifies the problem, e.g., the samples lie on a low-dimensional manifold.

2.4 Existing methodology

2.4.1 Traditional global approximations

The principle approach used in the past is fitting a global parametric model, usually having few parameters, to the sample points. These include constant, linear and polynomial approximations. For example, in the linear case $\mathcal{N}(\mathbf{x})$ is defined as:

$$\mathcal{N}(\mathbf{x}) = \mathbf{w}_0 + \sum_{i=1}^{N} \mathbf{w}_i \mathbf{x}_i \tag{2.4}$$

This approach has limited flexibility, since the parametric model chosen has to be close to the real model $\mathcal{D}(\mathbf{x})$. Therefore it is often used when there is some indication of what the function $\mathcal{D}(\mathbf{x})$ looks like. On the other hand, if the model chosen is appropriate, few learning samples are needed to estimate the small number of parameters involved.

2.4.2 Localized models

Local parametric modeling tries to alleviate the problems with the traditional global approximations by splitting the input space into smaller, sometimes overlapping, regions. Each region is approximated by a, usually simple, parametric function. Using an increasing number of regions decreases the bias of the approximation, while the variance increases. The number and positions of regions are important control knobs influencing the bias/variance tradeoff, and good settings can result in good approximations.

A simple localized model is the *locally constant* model or *lookup table*, where the input space is split into disjunct regions, and each region is approximated by a constant (local average). Figure 2.3 shows a resulting approximation.

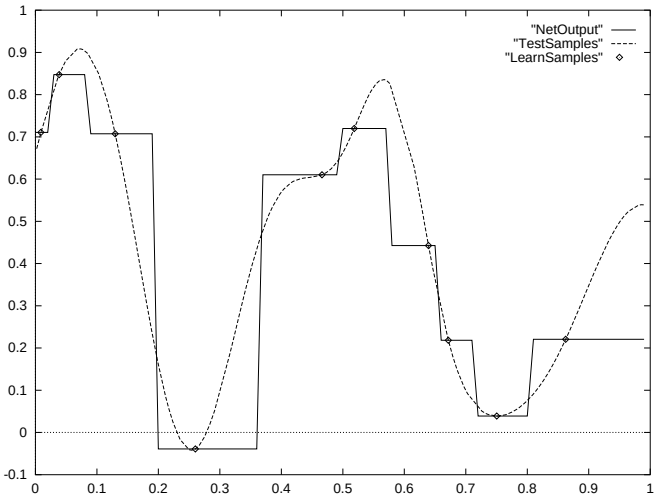


Figure 2.3: *Approximation using a lookup table.*

Other examples of localized approximation methods are splines [3], recursive partitioning [17], smoothers [2][5] and Radial Basis functions (which will be described in section 3.4).

Although localized approximations work well in low dimensional (say ≤ 2) settings, problems arise when higher dimensions are involved. Since the number of regions needed exponentially increases with the number of dimensions, and each region needs a couple of samples to base the local approximation on, the number of samples needed in high dimensions becomes huge.

For example, if we use local linear models and want to split each input dimension into 3 regions, the minimum number of samples needed in a n-dimensional situation equals $3^n(n+1)$: there are 3^n regions, each region containing $(n+1)$ samples. This function is exponential indeed.

2.4.3 Projection pursuit

Difficulty of approximating high dimensional function has motivated the use of *low dimensional expansions*, where the high dimensional input is reduced to a low dimensional intermediate result, which is used as input for a simple parametric approximator. For example, the *projection pursuit* method takes linear projections of the input vector as intermediate values [9][10]:

$$\mathcal{N}(\mathbf{x}) = \sum_{i=1}^{N} \mathcal{F}_i \left(\mathbf{w}_0 + \sum_{j=1}^{n} \mathbf{w}_j \mathbf{x}_j \right) \tag{2.5}$$

Here N *basis functions* $\mathcal{F}_i$ are used, which are applied to linear projections specified by $\mathbf{w}$. Basis functions are like building blocks for a function; the total output of the function is a combination of the basis functions' outputs (summation in this case).

It can be shown that each smooth function can be approximated well by the projection pursuit method, provided that N is large enough [6]. But even for small N the performance of these approximations is pretty good [7]. The big disadvantage lies in the difficulty of optimizing the parameters, resulting in non-optimal parameters and long optimization times.

2.5 Feed forward neural networks

2.5.1 Neurons

Artificial neural networks were initially devised as simplified models of the brain. A brain consists of a huge number of interconnected *neurons* (about 10^{11} for a human adult), cooperating to perform specific tasks. A biological neuron (figure 2.4) receives electric pulses on its incoming *dendrites*, and sends outgoing pulses through its *axon*. When and what pulses are sent depends on the incoming pulses and the importance of each dendrite.

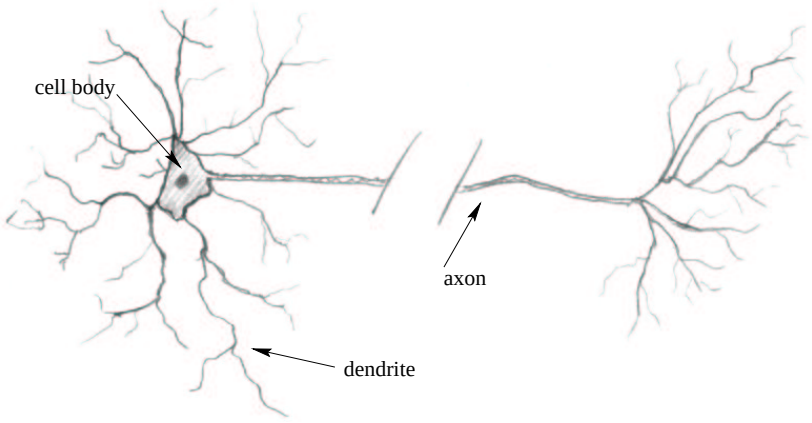


Figure 2.4: *A biological neuron.*

The *artificial neuron* (figure 2.5) is a simplification of its biological counterpart, introduced in 1943 [19]. Instead of pulses, real valued inputs and output are used, and the output is a simple calculation based on the inputs:

$$y = \mathcal{F} \left(\mathbf{w}_0 + \sum_{i=1}^{n} \mathbf{w}_i \mathbf{x}_i \right) \tag{2.6}$$

The inputs are first weighted by the *weights* $\mathbf{w}_i$, after which they are summed (also an *offset* $\mathbf{w}_0$ is added). This summation is supplied to the *activation function* $\mathcal{F}$, which produces the neurons output y. The weights are parameters of the neuron and influence the computation performed by it.

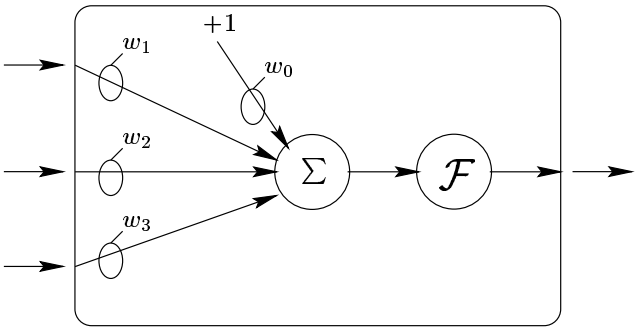


Figure 2.5: *An artificial neuron with 3 inputs.*

2.5.2 The 'standard' feed forward neural network

A set of neurons can be connected to form a network. One popular form of network is the *feed forward network* (figure 2.6), where neurons are organized into layers, and the output of each layer is used as input for the next layer.

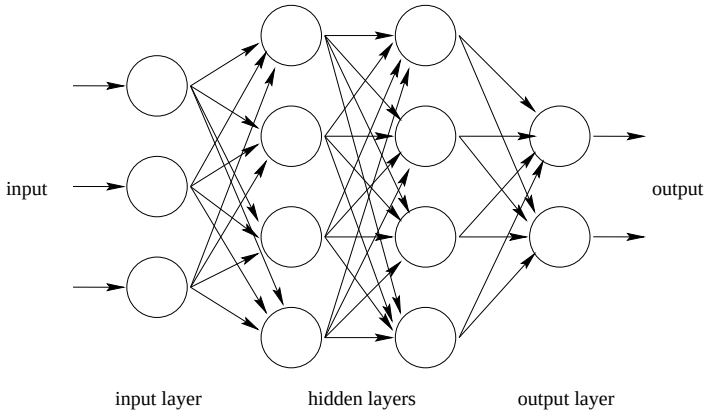


Figure 2.6: *A feed forward neural network, with three inputs, two outputs and two hidden layers.*

When an input is presented to the first layer, each layer of neurons can calculate its output based on the activations of the previous one, until an output is present at the output layer. This computation is called the *propagation* of the input. Note that the input layer only serves as provider of the input; it does not compute anything.

So the feed forward network is just a mapping between input and output, like we have seen before. It is nothing more than a parametric function, where the weights are the parameters. The exact form of the function is determined by the number of layers, the number of neurons in each layer, the activation functions used and the weights. Only the weights can be optimized (*learned* in neural network terminology) to minimize the error on the learning samples; the other settings have to be chosen before learning.

One specific network consisting of 3 layers is often used as universal function approximator:

- One input layer.
- One hidden layer with h units. The activation function is nonlinear, such as the commonly used *sigmoid* (figure 2.7):

$$\mathcal{F}(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

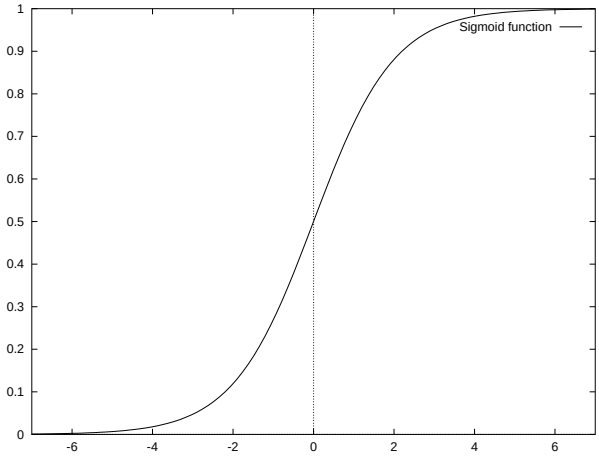


Figure 2.7: *Sigmoid activation function.*

- An output layer using a linear activation function (identity function).

Such a network implements the following function:

$$\mathcal{N}(\mathbf{x}) = \mathbf{w}_{2,0,0} + \sum_{i=1}^{h} \mathbf{w}_{2,0,i} \mathcal{F} \left(\mathbf{w}_{1,i,0} + \sum_{j=1}^{n} \mathbf{w}_{1,i,j} \mathbf{x}_j \right) \tag{2.8}$$

In this equation $\mathbf{w}_{a,b,c}$ indicates the c-th weight of the b-th neuron in the a-th layer (counting from 0). Notice the strong similarity with projection pursuit (equation 2.5). In fact, equation 2.8 can be written as equation 2.5.

It has been proven that these networks can approximate any function arbitrarily well, provided the number of hidden units h is large enough and that optimal parameters (weights) can be found [16]. In practice however, learning the weights is usually very hard and time consuming. Also there is considerable chance of reaching a non-optimal parameter setting (a *local minimum).*

2.6 Discussion

The approximation methods discussed all have their advantages and disadvantages. Which one to choose depends on the specific situation: the number of samples available, dimensionality, a priori knowledge and time available for optimizing the parameters.

As already pointed out in the previous sections, approximators can roughly be divided into two classes: local and global approximators. The general characteristics of these two classes which can be found in the literature (e.g., [18]) are listed below:

Global approximations:

- A moderate number of samples is needed;
- Scales pretty well to higher dimensions;
- The flexibility of the approximation depends on a priori settings, e.g., the number of hidden units in a feed forward neural network;
- Parameter optimization (learning) can take a lot of time, and reaching a good optimum is hard to obtain. This is caused by a lot of interference between the parameters.

Local approximations:

- Many samples are needed;
- Scaling to higher dimensions can be a problem;
- There is a high flexibility, which can be regulated by the number, positions and size of the regions. Even during learning regions can be added and deleted (*incremental learning*);
- Learning can be fast, since most parameters have only local influence (for the region), and there is not much interference.

Local approximations have attracted a great deal of interest recently, mostly because of the high flexibility needed for general function approximation. Flexibility can be advantageous, but in order to get a good approximator, a large number of choices has to be made about the parameters, number of regions, region type, etc.

The next chapter will study some choices, resulting in *locally weighted approximations*. Chapter 5 will compare that method with feed forward neural networks (a more global global approximation method); it also gives our findings about the local/global approximation properties.

Chapter 3

Locally weighted approximations

3.1 Using a set of local experts

In local approximations, the input space is divided into regions. Each region has a *local expert* assigned to it, which tries to approximate that region by means of a local parametric model.

We can expect that in a region, provided it is small enough, the destination mapping $\mathcal{D}(\mathbf{x})$ will have small variance, i.e., it does not fluctuate very much within that region (due to the smoothness criterion). Therefore each expert's parametric model can be kept simple (not containing too many parameters), such as constant or linear.

The local method certainly has benefits, since the local experts are simple and the parameters can be optimized easily. This does not come for free: at one end problems disappear, at the other new problems are introduced.

Let us assume that each local expert i creates some approximation $\mathcal{N}_i(\mathbf{x})$ for its region. Now there are two basic questions:

- How are these regions defined?
- Present an input vector $\mathbf{x}$ to the network. How is the output of the network $\mathcal{N}(\mathbf{x})$ determined, given the predictions (output) of the individual experts $\mathcal{N}_i(\mathbf{x})$?

If each expert is assigned a position $\mathbf{p}$ in the input space, one approach would be to select the expert closest to the presented input vector $\mathbf{x}$. The output of this winning expert is then used as the network output. No other experts are consulted.

The disadvantage of this approach is that the regions are non overlapping, making it quite possible that the approximation $\mathcal{N}(\mathbf{x})$ will be discontinuous at the region boundaries (see figure 3.1). Therefore it is not very well suited for approximating smooth functions.

3.2 Locally weighted approximations

So we would like to get rid of the discontinuities introduced by sharp region boundaries. This can be achieved by by making the regions overlapping, creating transition zones between the experts. Within a transition zone, the prediction of neighbouring experts will be mixed to generate the output, ensuring that no discontinuities will occur. This method is known as

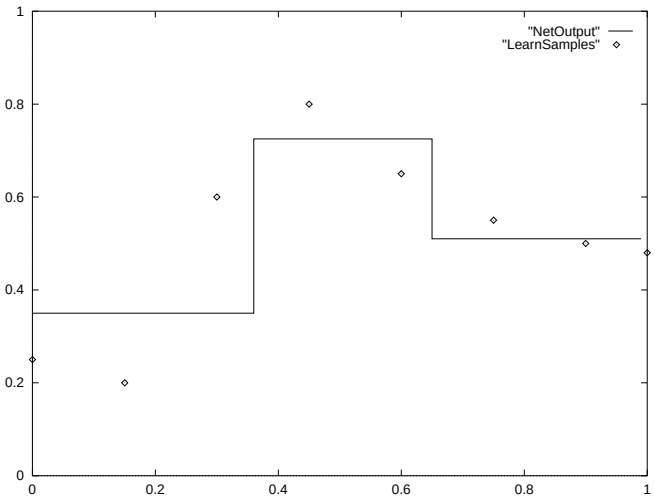


Figure 3.1: *Approximation with sharp boundaries for* 3 *experts. Each expert region* i *is defined as all the points closest to expert* i*; each expert approximates its region with a constant.*

a *locally weighted approximation.* We will also use the general term 'network' for it, since locally weighted approximators can be regarded (and drawn) as neural networks.

3.2.1 Creating smooth transitions between expert regions

How to create smooth transition zones? Let g_i be the *activation* of expert i, which lies in the interval [0..1]. It is an indication of how relevant the output of this expert is to the network output, given some input vector $\mathbf{x}$. The closer $\mathbf{x}$ is to the *position* $\mathbf{p}_i$ of the expert, the higher g_i.

Also define a *weighting function* $\mathcal{W}_i(\mathbf{x})$, which has its maximum at $\mathbf{p_i}$, and reaches zero rapidly when $\mathbf{x}$ gets further away from $\mathbf{p}$. For example, the Gaussian kernel can be used (see figure 3.2):

$$\mathcal{W}_i(\mathbf{x}) = e^{-\left(\frac{||\mathbf{p}_i - \mathbf{x}||^2}{\sigma_i^2}\right)} \tag{3.1}$$

Now, we define the activation g_i of expert i as being the weight $\mathcal{W}_i(\mathbf{x})$, normalized by the weights of the other experts (N is the number of experts):

$$g_i = \frac{\mathcal{W}_i(\mathbf{x})}{\sum_{i=1}^{N} \mathcal{W}_i(\mathbf{x})} \tag{3.2}$$

For every input vector $\mathbf{x}$, the sum of all the g_i's will equal 1.

As a final step, let the output of the network be the mix of all the individual expert outputs, weighted by their activation:

$$\mathcal{N}(\mathbf{x}) = \sum_{i=1}^{N} g_i(\mathbf{x}) \mathcal{N}_i(\mathbf{x}) \tag{3.3}$$

All this mumbo jumbo results in smooth transitions between expert regions, as can be seen in figure 3.3 (a 1-dimensional case).

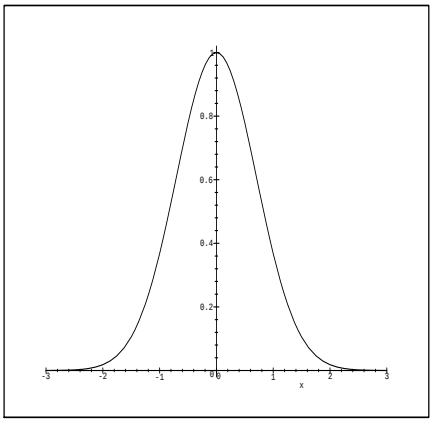
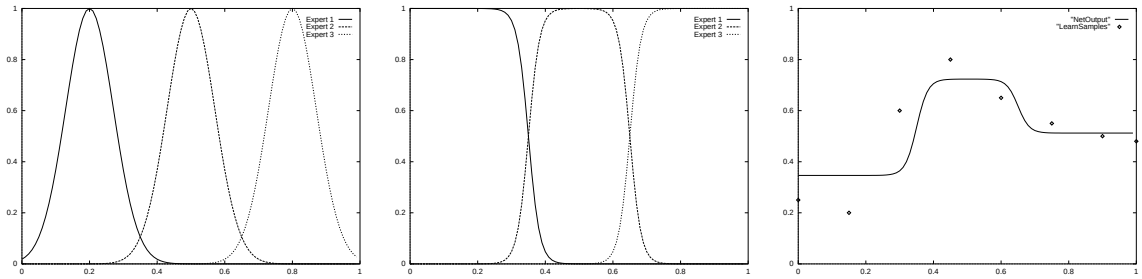


Figure 3.2: *The one-dimensional Gauss kernel, where $p = 0$ and $\sigma = 1$*



Gaussian kernels | Activations g_i | Network output $\mathcal{N}(\mathbf{x})$

Figure 3.3: *Smooth transitions between expert regions, for 3 experts. Each experts' region is defined by its activation g_i; each kernel approximates its region with a constant.*

Close to the center $\mathbf{p}_i$ of an expert i, only that expert is active[1]: the output of the network is equal to that expert's output. At transition zones however, two or more experts are activated a little, and the output of the network is a weighted average of the output of those experts. Discontinuities are avoided.

3.3 Choosing structure and parameters

The global structure of the locally weighted learning network outlined in the previous section has to be filled in with specific choices which will influence the quality of the final approximation:

- How many experts should be used?
- What are the locations $\mathbf{p}_i$ of the experts?
- The weighting function chosen influences the approximation. What weighting function $\mathcal{W}_i(\mathbf{x})$ to use? How to get good parameters for it?

[1] Actually, the other experts also have a very small activation, but we can ignore their influence.

- What local parametric model $\mathcal{N}_i(\mathbf{x})$ to use for each expert? How to find its parameters?

What we are aiming at is to find an approximation which has a small error on the learning samples (small bias), and is as smooth as possible (small variance), which improves the chance of obtaining a good generalization and avoid overfitting. This puts restrictions on the locally weighted approximation network:

- The approximation within a region has to be smooth, making sure that no overfitting will occur within a region;
- The approximation at the transition zones has to be smooth;
- The number of experts should be kept small, because a large number of experts means a lot of parameters, slower learning and a larger chance of high variance.

Finding good settings is very difficult. The remainder of this chapter will go into the available choices. Chapter 5 will compare some methods by showing experimental results.

3.3.1 Positions and number of experts

The placement and number of experts is obviously important for the final approximation obtained. Both parameters determine the size of expert regions. Having larger regions is advantageous, because less computational resources are needed. On the other hand, a large region is only possible when the expert can approximate that region properly: this occurs at places where $\mathcal{D}(\mathbf{x})$ has a smooth curvature. At more complex parts there need to be more experts.

Furthermore, region placement can be dictated by the number of samples available in that region: the expert needs a minimum number of samples to base its approximation on.

CMAC networks [1] place experts at fixed intervals, completely circumventing the problem.

The approach used by Moody and Darken [20] uses self-organization [23] to find the expert positions, following the input distribution of the samples. This reduces the chance of having too little samples available for an expert, but does not place more experts at 'difficult' parts of $\mathcal{D}(\mathbf{x})$.

Schaal and Atkeson [24] place experts in an incremental way, inserting an expert at places where there are samples but no experts nearby. The exact value of 'nearby' has to be set beforehand.

Other ways of positioning are placing one expert at each sample (usually resulting in too many experts), placing experts randomly, and optimizing the positions to decrease the error on the learning samples.

All these solutions seem to be non-optimal, since they either do not use the distribution of the samples, or do not put more experts at places where the destination function is not well approximated by one single expert.

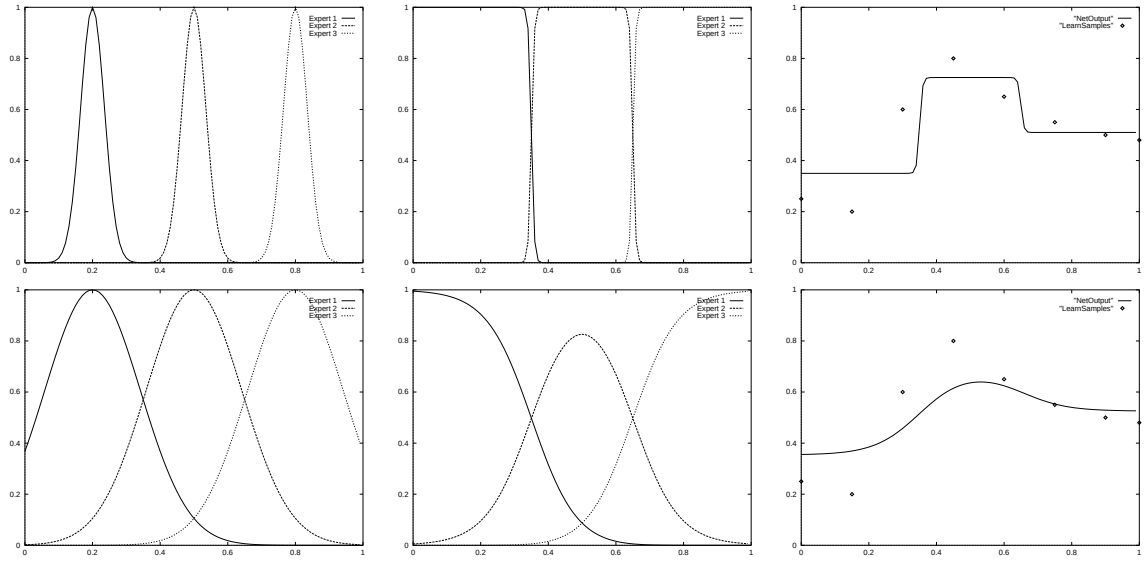
3.3.2 Shape of the weighting function

The weighting functions determine the places and width of the transition zones. Here again is a tradeoff: large transition zones create large interference between experts (since more experts are responsible for the output at some $\mathbf{x}$), but also very smooth transitions. Small transition zones result in exactly the opposite effect.

An often used weighting function is the Gaussian kernel:

$$\mathcal{W}_i(\mathbf{x}) = e^{-\left(\frac{||\mathbf{p}_i - \mathbf{x}||^2}{\sigma_i^2}\right)} \tag{3.4}$$

Where σ_i determines the width of the kernel. Figure 3.4 shows the effect of σ_i on the transition zones and resulting approximation.



Gaussian kernels | Activations g_i | Network output $\mathcal{N}(\mathbf{x})$

Figure 3.4: *Influence of σ_i on transition zones, drawn for 3 experts. The top plots are drawn for $\sigma_i = 0.05$, the bottom ones for $\sigma_i = 0.2$.*

For higher input dimensions there are more options. The 'width' of a kernel can be equal in all directions, or be of different width into each principal direction. It is even possible to rotate the kernel, resulting in a rotated ellipsoid.

Setting the parameters can be done in different ways. The CMAC approach uses uniform σ_i values, which have to be set beforehand. Schaal and Atkeson learn the widths by a gradient method, optimizing the MSE. Moody and Darken use a heuristic, so that σ_i is proportional to the distance to the nearest expert.

3.3.3 Local model

The local models $\mathcal{N}_i(\mathbf{x})$ are ultimately responsible for approximating $\mathcal{D}(\mathbf{x})$. Simple models like locally constant are not very flexible, and a lot of experts will be needed for a good

approximation. Models like high-order polynomials or feed forward neural networks can have high variance, making them less suited because they can cause overfitting.

Moody and Darken as well as the CMAC use locally constant models. Schaal and Atkeson apply local linear models. Using higher order polynomials as local model is discussed in [5].

3.4 Radial Basis Functions

This section is a deviation from the previous sections, describing another widely used class of local approximators: the so called Radial Basis Functions (RBF). Since they are very similar to locally weighted approximations, we will discuss them here.

A RBF is a weighted sum of N Gaussian kernels:

$$\mathcal{N}(\mathbf{x}) = \sum_{i=1}^{N} c_i \mathcal{W}_i(\mathbf{x}) = \sum_{i=1}^{N} c_i e^{-\left(\frac{||\mathbf{p}_i - \mathbf{x}||^2}{\sigma_i^2}\right)} \tag{3.5}$$

This is essentially the same as equation 3.3, if we take the local model $\mathcal{N}_i(\mathbf{x})$ to be constant (c_i), and remove the normalization of the weights. Note that the Gaussian kernels now play another role: they do not specify the overlap regions, but now together determine all of the approximation (by adding all the Gaussians).

3.5 Conclusion

This chapter provided the general framework of locally weighted approximations. Still, choices have to be made about the specific settings to use; this will be done in the next chapter, which investigates and compares various choices.

Chapter 5 will compare the specific locally weighted approximation of the next chapter with two other methods: Radial Basis Functions and feed forward neural networks.

Chapter 4

Experiments: Locally weighted polynomials

4.1 Introduction

The previous chapter introduced locally weighted approximations where a number of local experts cooperate to produce a function approximation. However, it only discussed the general structure of such networks, but does not say anything about the details involved.

To get a specific locally weighted approximation method, we have to choose some particular settings:

- Each expert makes a local approximation using a local model. What local model to choose?
- The weighting function defines the regions of the experts and the overlap between neighbouring experts; but what is a good weighting function?
- How many experts have to be used?
- How are the parameters going to be optimized?

This chapter tries to answer these questions. To this end, we have performed a number of experiments, in which various settings are tested and compared.

In all these experiments, a network is trained on a 2-dimensional test function described in section 4.3. Each experiment uses the same parameter optimization method (discussed in section 4.2.4); the other three settings are varied: four different local models, two weighting functions and a varying number of experts.

The outline of this chapter is as follows. The next section first describes the general settings of the network, the local models and weighting functions used in the experiments. After that, the experimental results are discussed. Section 4.4.1 studies the result of varying the number of experts. Section 4.4.2 shows the results of using different local models. Finally, section 4.4.3 reports the effect of the two different weighting functions.

The result of the experimental comparison is a guideline of what specific choices should be made for the locally weighted approximation method; we will use this knowledge in the next chapter, where the optimal network found will be compared with existing methods.

4.2 General settings

This section discusses the general settings for the locally weighted network, which will be used in all the experiments. It also introduces two weighting functions and four local models, to be compared in the experiments.

4.2.1 Network structure

The output of locally weighted approximation network was given in section 3.2.1:

$$\mathcal{N}(\mathbf{x}) = \sum_{i=1}^{N} g_i \mathcal{N}_i(\mathbf{x}) \tag{4.1}$$

where $\mathcal{N}_i(\mathbf{x})$ is the local approximation performed by expert i (section 3.3.3), N is the number of experts and $g_i(\mathbf{x})$ is the activation of expert i:

$$g_i = \frac{\mathcal{W}_i(\mathbf{x})}{\sum_{i=1}^{N} \mathcal{W}_i(\mathbf{x})} \tag{4.2}$$

where $\mathcal{W}_i(\mathbf{x})$ is the weighting function (section 3.3.2).

Substituting $g_i(\mathbf{x})$ in equation 4.1 yields:

$$\mathcal{N}(\mathbf{x}) = \frac{\sum_{i=1}^{N} \mathcal{N}_i(\mathbf{x}) \mathcal{W}_i(\mathbf{x})}{\sum_{i=1}^{N} \mathcal{W}_i(\mathbf{x})} \tag{4.3}$$

This is the network structure which will be used. Still, the local model $\mathcal{N}(\mathbf{x})$ and weighting function $\mathcal{W}_i(\mathbf{x})$ have to be chosen.

4.2.2 Weighting function

As we wrote earlier, the Gaussian kernel is often used as weighting function for expert i:

$$\mathcal{W}_i(\mathbf{x}) = e^{-\left(\frac{||\mathbf{p}_i - \mathbf{x}||^2}{\sigma_i^2}\right)} \tag{4.4}$$

The parameters of this function are the position $\mathbf{p}_i$ and width σ_i. It uses the Euclidean distance metric $||\mathbf{p}_i - \mathbf{x}||$ as distance between position of the kernel $\mathbf{p}_i$ and input vector $\mathbf{x}$.

Since σ_i partly determines the overlap between neighbouring experts, it is an important parameter. Having a single parameter determining the width of the kernel in *every* direction can be restrictive for multi-dimensional inputs. Therefore, changing the weighting function so that it is possible to set different widths for different directions might result in better approximations. This can be achieved by using a weighted Euclidean distance[1] instead of the unweighted Euclidean distance used in equation 4.4:

$$\mathcal{W}_i(\mathbf{x}) = e^{-\left(\sqrt{(\mathbf{p}_i - \mathbf{x})^T \mathbf{M}_i^T \mathbf{M}_i (\mathbf{p}_i - \mathbf{x})}\right)^2)} \tag{4.5}$$

[1]This distance is also known in another context as the Mahalanobis distance: the distance between point $\mathbf{x}$ and a distribution with mean $\mathbf{p}_i$ and covariance matrix $\mathbf{M}_i^T \mathbf{M}_i$ [28].

$\mathbf{M}_i$ is a matrix of size $n \times n$, where n is the number of input dimensions. The elements of $\mathbf{M}_i$ determine the shape of the kernel.

The shapes of the unweighted and weighted distance functions are plotted in figure 4.1; we will use the names W1 and W2 to denote these two functions.

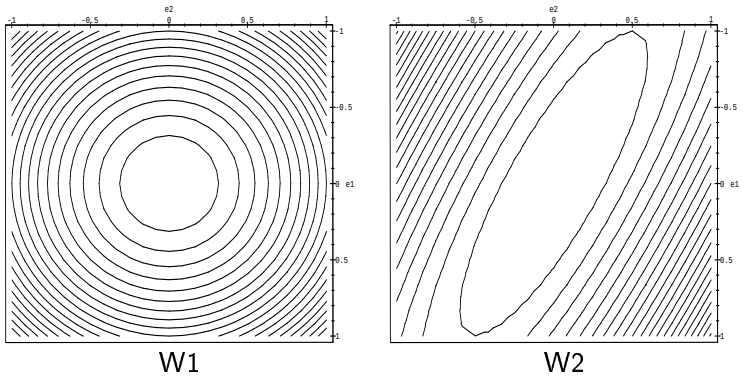


Figure 4.1: *The unweighted Euclidean distance (W1) and weighted Euclidean distance (W2).*

It is not directly clear which weighting function will give better approximations. Weighting W2 could be the best choice, since it is more flexible than W1. On the other hand, this flexibility could create overfitting. Furthermore, it is not known whether the parameter optimization method will find good weightings. Therefore, it is interesting to compare the two weighting functions, which is done in section 4.4.3.

4.2.3 Local model

As discussed in section 3.3.3, there is a tradeoff in choosing the local model. On one hand it should be flexible enough to create a good local approximation; on the other it should not be too flexible, overfitting the data.

Since the choice of local model appears to be important, we have performed experiments which compare different local models, to be introduced here.

A common choice of local model are low-order polynomials (up to order 3) [2][5]. Lower order polynomials are less flexible than higher order ones; therefore polynomials of differing order seem a good choice for our experiments.

However, there are some drawbacks. The number of parameters needed for a 2nd or 3rd order polynomial increases very quickly with the number of input dimensions: $O(N^2)$ parameters are needed for a 2nd order polynomials and N input dimensions. To estimate many parameters, many samples are needed, which we want to avoid. Furthermore, the sensitivity of noise in the samples greatly increases with increasing polynomial order.

Therefore we choose not to use the original polynomials, but a modified version based on a linear projection $\mathcal{Q}_i(\mathbf{x})$ of the input vector $\mathbf{x}$:

$$\mathcal{Q}_i(\mathbf{x}) = \sum_j \mathbf{w}_{i,j}\mathbf{x}_j \tag{4.6}$$

This projection is dependent on the parameters $\mathbf{w}_{i,j}$, which determine the direction of the projection. Using $\mathcal{Q}_i(\mathbf{x})$, we define 4 different functions:

$$\mathcal{N}_i(\mathbf{x}) = \begin{cases} c_0 & \text{(Poly0)} \\ c_0 + c_1\mathcal{Q}_i(\mathbf{x}) & \text{(Poly1)} \\ c_0 + c_1\mathcal{Q}_i(\mathbf{x}) + c_2(\mathcal{Q}_i(\mathbf{x}))^2 & \text{(Poly2)} \\ c_0 + c_1\mathcal{Q}_i(\mathbf{x}) + c_2(\mathcal{Q}_i(\mathbf{x}))^2 + c_3(\mathcal{Q}_i(\mathbf{x}))^3 & \text{(Poly3)} \end{cases} \tag{4.7}$$

This method is essentially a form of projection pursuit, described in section 2.4.3.

Using the projection greatly reduces the number of parameters: Poly2 has only $(N + 3)$ of them.

Note that Poly0 and Poly1 are just constant and linear approximations, respectively. The possible variance is largest for Poly3.

Choosing which of the four polynomials[2] will give the best approximation when used as local model is not trivial. The results of the experimental comparison are described in section 4.4.2.

4.2.4 Learning method

The parameters of the network are optimized to minimize the MSE on the learning samples. These parameters are:

- The expert (weighting function) positions $\mathbf{p}_i$;
- The widths of the weighting functions, coded in the matrices $\mathbf{M}_i$;
- The parameters of the local (polynomial) model: $\mathbf{w}_{i,j}$ and c_0, c_1, c_2, c_3.

All these parameters are updated simultaneously, using the conjugate gradient method [26] (CG) with Powell restarts [21]. This method is a general parameter estimation algorithm, and uses the first order derivatives of the parameters to the MSE. The algorithm uses a maximum of 5 line search iterations for each CG iteration. Each time, CG was run for 500 iterations (after more than 500 iterations the errors did not decrease significantly anymore).

We also tried other learning methods: the standard backpropagation [22] and Alopex [29]. These algorithms were found to be inferior to CG, mostly because of slower convergence.

4.2.5 Initial parameter settings

The conjugate gradient algorithm is not guaranteed to find a global minimum of the MSE, i.e., the parameters found after optimization need not be optimal. It is therefore essential to begin learning with good initial settings of the parameters, letting CG continue from there. The initial settings for our experiments are:

[2] We will use the term 'polynomial' to indicate the functions Poly0 to Poly3, although those functions are defined as polynomials on the *projection* $\mathcal{Q}_i(\mathbf{x})$ and not directly on $\mathbf{x}$ itself.

- The *positions* $\mathbf{p}_i$ are initialized using self-organization (section 3.3.1), so that the positions of the experts follow the input distribution of the learning samples. This should result in a 'fair' distribution, in which every expert has an equal share of learning samples in its region. Note that this is only an initial setting; the positions can move during learning.

- The widths of the weighting functions $\mathbf{M}_i$ are initially set according to the distance to their closest neighbouring expert:

  $$\sigma_i = 0.8\sqrt{d_{i,j}}$$

  where $d_{i,j}$ is the distance of expert i to its closest neighbouring expert j. The choice of constant 0.8 was based on visual indications only; using this constant seemed to give reasonable overlap of kernels (overlapping, but not too wide). The minimal value of σ_i is taken to be 0.05 (also based on visual indications), to prevent creation of infinitesimal small regions. The matrix $\mathbf{M}_i$ is set so that the width equals σ_i in every input direction.

  A similar method is used by Moody and Darken in [20].

- Local model parameters $w_{i,j}$ of the projections $\mathcal{W}_i(\mathbf{x})$ are initialized with 1.0. The polynomial parameters c_1 to c_3 are set to 0.0. Parameter c_0 is initialized with the average of all the learning samples, weighted by the activation of the expert:

  $$c_{0,i} = \frac{\sum_{s=1}^{|S|} g_i(\mathbf{x}^s)\,\mathcal{E}^s}{|S|} \tag{4.8}$$

  Here, $|S|$ denotes the number of samples, $\mathbf{x}^s$ is the input vector of sample s, $g_i(\mathbf{x}^s)$ is the activation of expert i (equation 3.2) and $\mathcal{E}^s$ is the squared error for sample s (equation 2.2).

  This initialization of c_0 is not essential, but speeds up the learning process.

4.2.6 The number of experts

The number of experts needed is important, but not easy to estimate before learning; it is like the problem of finding the number of hidden units in a feed forward neural network. Using too few experts will result in a bad approximation (high bias); using too many experts can cause overfitting.

Section 4.4.1 discusses the results of experiments where the number of experts is varied.

4.3 The approximated function

For the experiments we used a 2-dimensional function to be approximated, taken from [24] (figure 4.2):

$$\mathcal{D}(\mathbf{x}) = \max \begin{cases} e^{-10\mathbf{x}_1^2} \\ e^{-50\mathbf{x}_2^2} \\ 1.25\, e^{-5(\mathbf{x}_1^2+\mathbf{x}_2^2)} \end{cases} \tag{4.9}$$

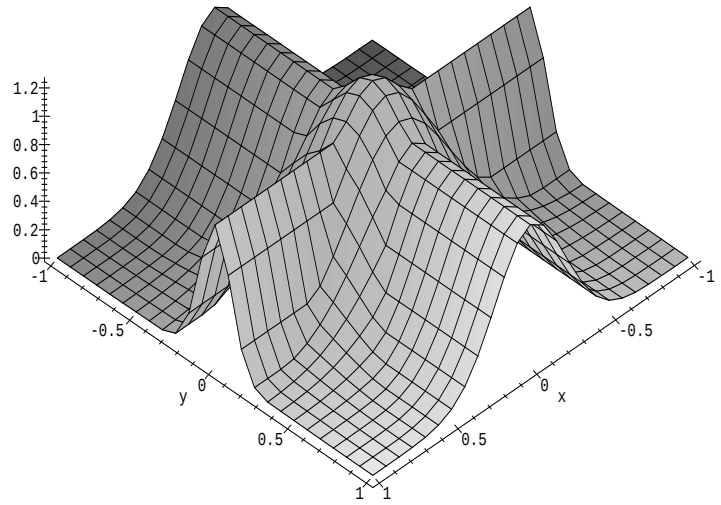


Figure 4.2: *Function Dest1.*

For the experiments, learning samples and test samples are generated from this function, in the form $(\mathbf{x}, \mathcal{D}(\mathbf{x}))$; the input vectors $\mathbf{x}$ are taken from a uniform random distribution. No noise was added to the output.

The test set used in the experiments always contains 2000 samples, and is used to determine the test error of an approximation[3]. The learn set contains either 100 or 1000 samples, depending on the experiment. Both sets do not have samples in common.

We will refer to this function as Dest1.

4.4 Experimental results

This section discusses the results of experiments, where locally weighted polynomial networks are trained on a learning set derived from the Dest1 function. The experiments use the general settings given in the previous sections; the following variable settings are compared:

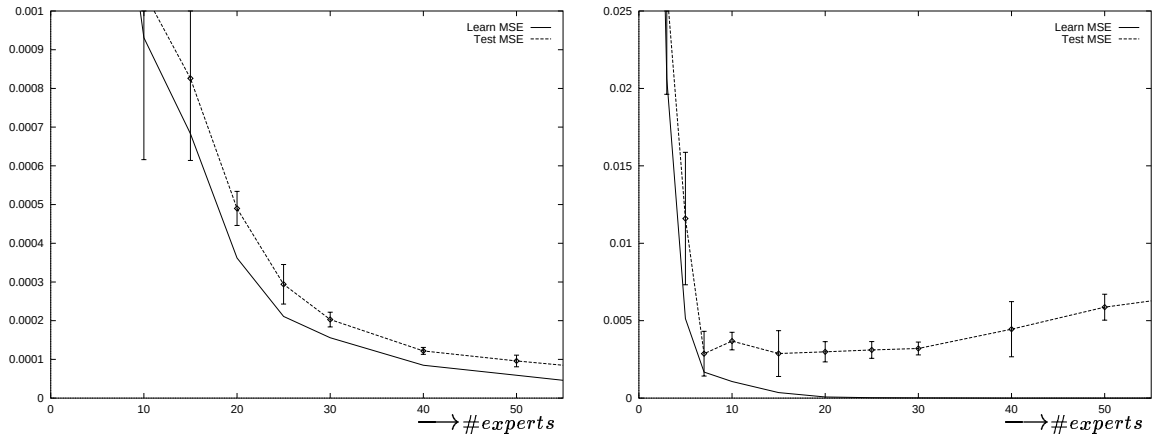
- Number of experts: varied from 1 to 50;
- Number of learning samples: 100 (which is rather small) and 1000 (which gives a considerably better indication of function Dest1);
- Local models: Poly0, Poly1, Poly2 and Poly3;
- Distance functions: W1 and W2.

Each combination determines a particular locally weighted network structure. For all those combinations, the parameters of the network were trained 5 times; the resulting errors discussed in the rest of this chapter are the averages of these 5 runs.

[3]We could have used the real integral error from equation 2.1 instead, but did not do it to keep the simulation program simple(r).

4.4.1 Effect of number of experts

Figure 4.3 shows the learn MSE and test MSE of approximations performed with varying number of experts. The weighting function W1 and local model Poly0 were used, but other settings of local model and weighting function give similar results. Some errors together with their standard deviations are listed in table 4.4.



1000 learn samples 100 learn samples

Figure 4.3: *Effect of number of experts on learn MSE and test MSE. The horizontal axis indicates the number of experts; note the different scales of the vertical axes.*

#experts	Learn MSE	Test MSE
5	$3.9 \cdot 10^{-3} \pm 2 \cdot 10^{-4}$	$4.0 \cdot 10^{-3} \pm 2 \cdot 10^{-4}$
50	$5.9 \cdot 10^{-5} \pm 8 \cdot 10^{-6}$	$9.6 \cdot 10^{-5} \pm 1 \cdot 10^{-6}$
100	$9.4 \cdot 10^{-6} \pm 1 \cdot 10^{-6}$	$6.2 \cdot 10^{-5} \pm 4 \cdot 10^{-5}$

Figure 4.4: *MSE's and standard deviations for* 1000 *samples, weighting function W1 and local model Poly0.*

As expected, the number of experts has a high impact on both learn error and test error. A network having few experts has small expressive power (high bias), and can not approximate the learn samples very well. Using more experts decreases both learn error and test error.

The learn error always decreases when the number of experts increases; this happens faster for the small learning set than for the large learning set, because the small learning set is easier to approximate.

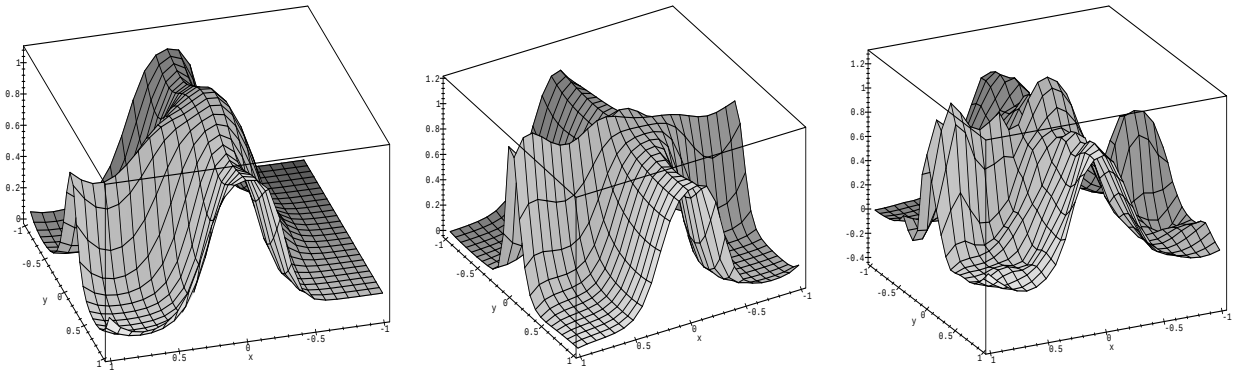
The test error initially decreases, but goes up again for the small learning set: an indication of overfitting. The same happens for the large learning set, but for a much higher number of experts. The optimal number of experts (for which the test error is minimal) is about 8 for the small set and about 90 for the large set.

Obviously, the network trained on the large learning set makes a better approximation than the one trained on the small set (for the optimal number of experts), since the small set gives less information about the underlying function Dest1.

Note that the overfitting does not increase very fast with increasing number of experts, because assigning a large number of experts implicates having small regions; if one expert

overfits the data, this is localized only to its small region and the effect on the total approximation is limited.

Figure 4.5 plots some approximations of the small learning set, showing underfit, good fit and overfit. The left plot shows an approximation where half of the function is 'missing', because there is no expert in that region. The middle plot is the optimal plot, which is not perfect but has good generalization. The right plot shows overfitting: the learning samples are approximated very well, but makes the surface bumpy and the generalization poor.



5 experts 8 experts 50 experts

Figure 4.5: *Approximations of function* Dest1 *using* 100 *learning samples, showing underfit, good fit and overfit, respectively.*

4.4.2 Effect of local model

This section discusses the effect of varying the local model on the performance of the approximation. Figures 4.6 and 4.7 show the results for the large and small learning set, with the four local models Poly0, Poly1, Poly2 and Poly3. Weighting function W1 was used.

As can be seen, there is generally little difference between the local models, at least for a rather large number of experts. In those situations, the regions of the experts are small and the variance of Dest1 within a region is also small: it can be approximated equally well by all the local models. Furthermore, the influence of the overlap parameters (kernel widths $\mathbf{M}_i$) increases with increasing number of experts, simply because there are more transition zones. The overlap parameters are then responsible for a large part of the flexibility of the approximation, so that a good approximation can be obtained relatively independent of the local model chosen.

The amount of overfitting on the small learning set is about the same for each local model. Only Poly2 shows an unusual high test error for 50 experts in one of the 5 trials. This can be understood by figure 4.10, where the faulty approximation is plotted; one of the experts is clearly overfitting with a very deep parabola.

In the case where few experts are employed, there is noticeable difference between the local models. Table 4.11 shows the errors for 3 kernels; these errors are not visible in figures

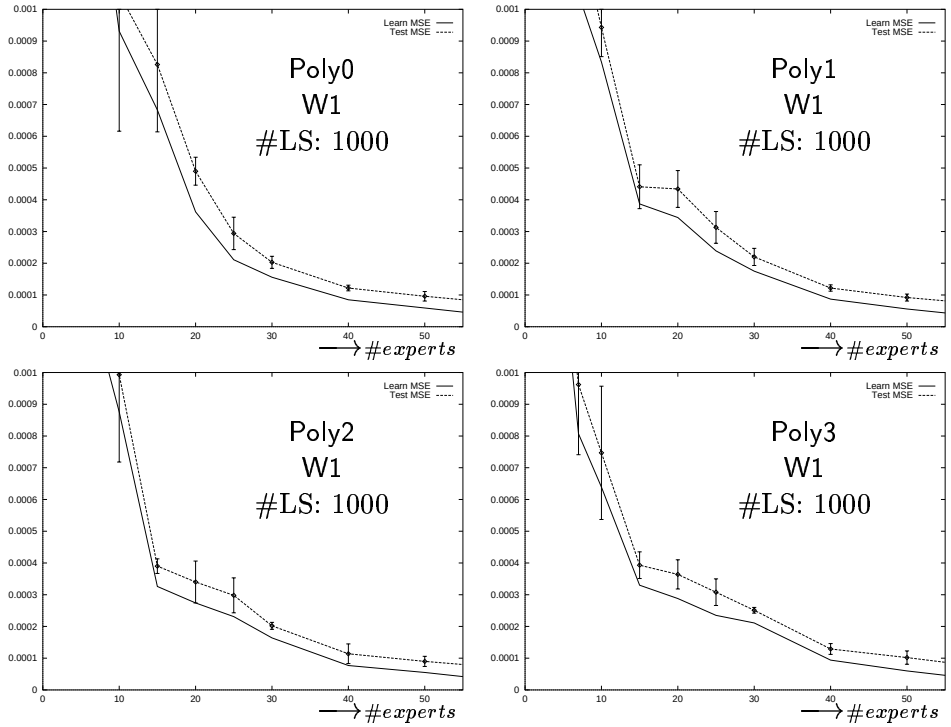


Figure 4.6: *MSE for varying number of experts and local models, using* 1000 *learning samples (LS) and weighting function W1.*

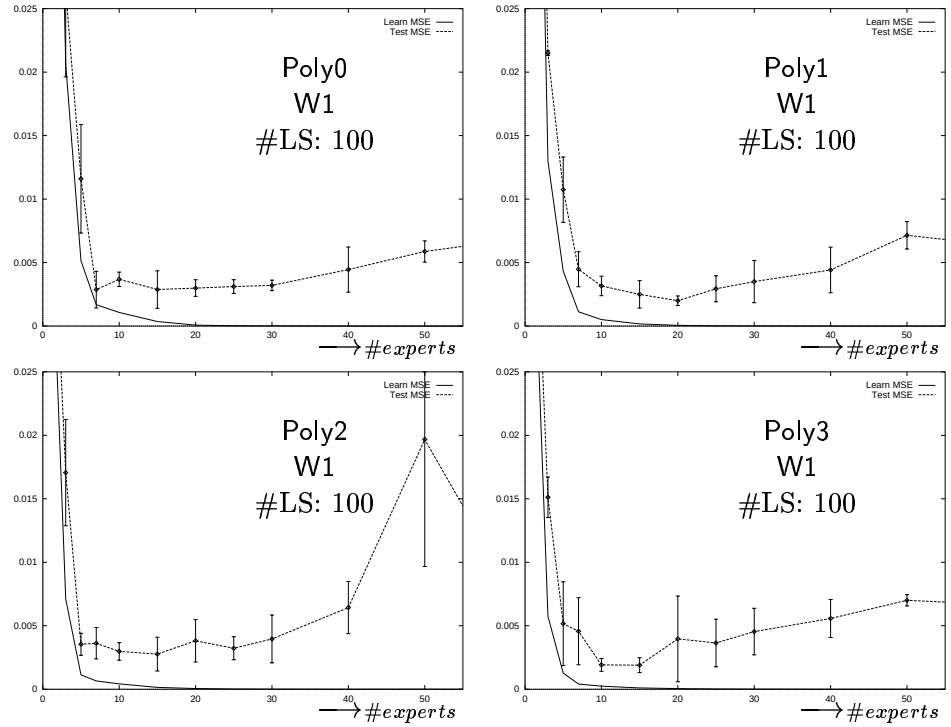


Figure 4.7: *MSE for varying number of experts and local models, using* 100 *learning samples (LS) and weighting function W1.*

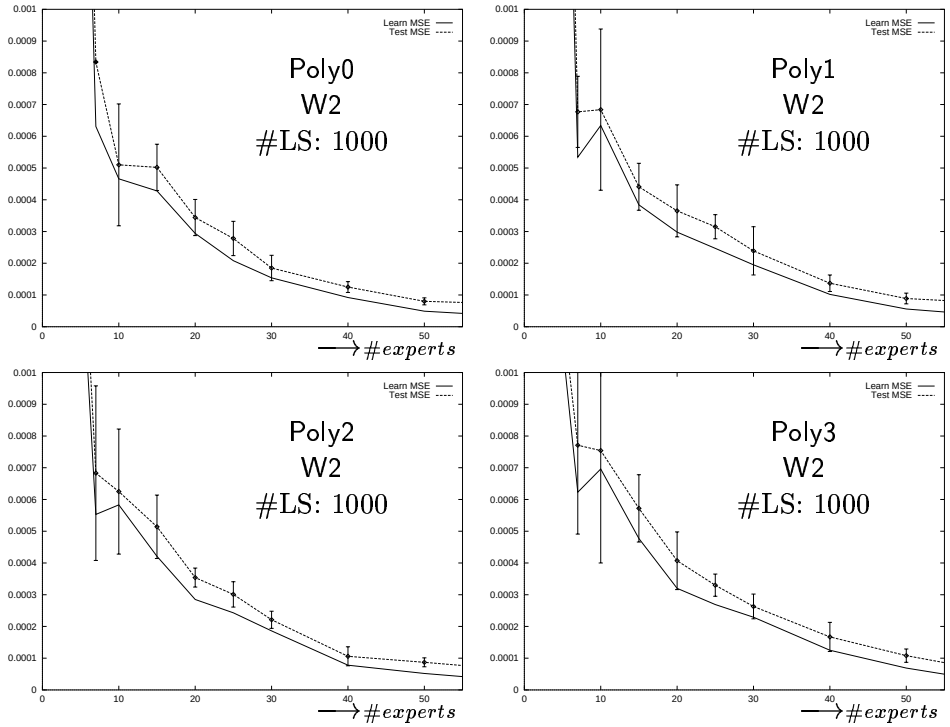


Figure 4.8: *MSE for varying number of experts and local models, using* 1000 *learning samples (LS) and weighting function W2.*

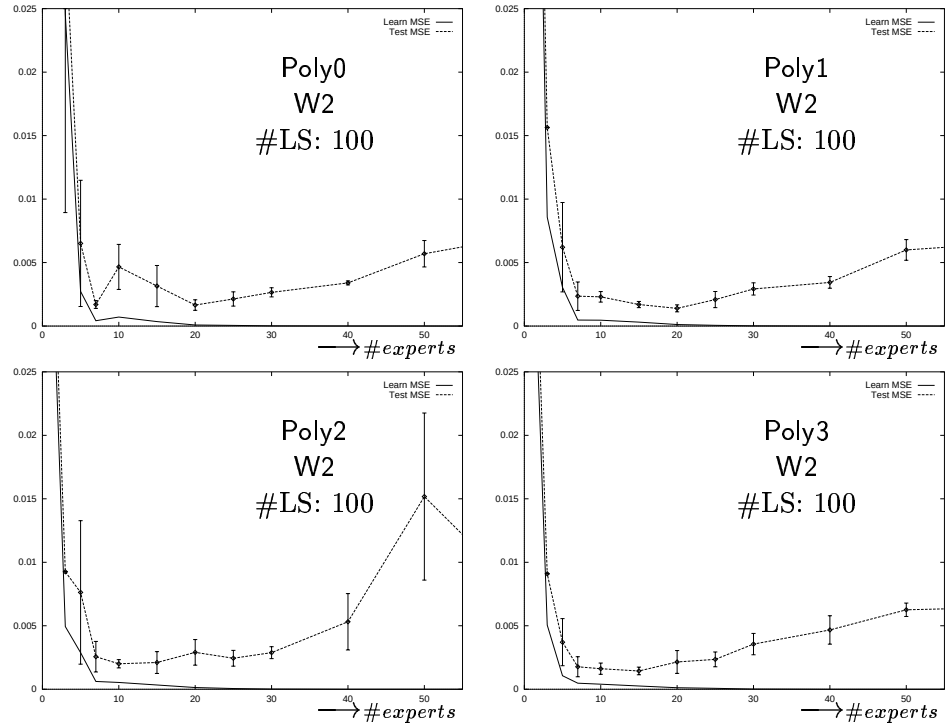


Figure 4.9: *MSE for varying number of experts and local models, using* 100 *learning samples (LS) and weighting function W2.*

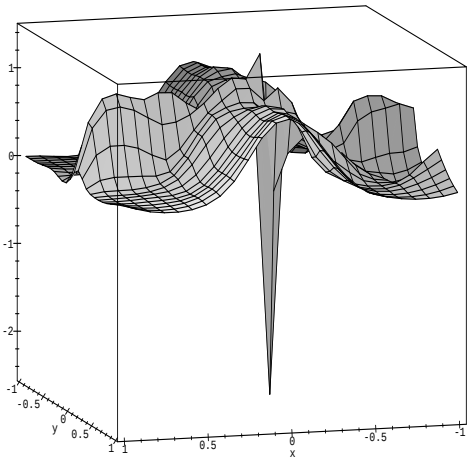


Figure 4.10: *A very high overfit by one of the experts, using Poly2 and W1 for* 50 *experts.*

4.6 and 4.7. The reason for the difference is that with few experts the regions are large and the local models have to handle the major part of the approximation, favouring the more flexible models Poly2 and Poly3.

Local model	Learn MSE	Test MSE
Poly0	$2.1 \cdot 10^{-2} \pm 1 \cdot 10^{-3}$	$2.1 \cdot 10^{-2} \pm 1 \cdot 10^{-3}$
Poly1	$1.8 \cdot 10^{-2} \pm 5 \cdot 10^{-4}$	$1.9 \cdot 10^{-2} \pm 1 \cdot 10^{-3}$
Poly2	$1.1 \cdot 10^{-2} \pm 1 \cdot 10^{-3}$	$1.2 \cdot 10^{-2} \pm 1 \cdot 10^{-3}$
Poly3	$9.5 \cdot 10^{-3} \pm 7 \cdot 10^{-4}$	$0.1 \cdot 10^{-2} \pm 3 \cdot 10^{-4}$

Figure 4.11: *MSE's and the standard deviations for* 3 *experts,* 1000 *samples and weighting function W1. There is a significant difference in both learn and test errors.*

4.4.3 Effect of distance function

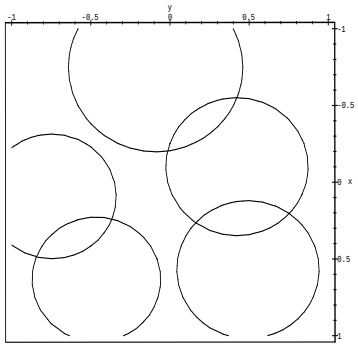
Section 4.2.2 pointed out that weighting function W2 allows different widths of the kernel in different directions (as opposed to W1) and also allow rotation. This would suggest that using W2 results in better approximations.

Figures 4.8 and 4.9 show the results of experiments similar to the ones performed in the previous section, except that weighting function W2 was used instead of W1. Comparing figures 4.6/4.7 and 4.8/4.9, we can conclude that there is generally little difference in using the two weighting functions. Only for a small number of experts, weighting function W2 performs significant better (like with the local models); see table 4.12.

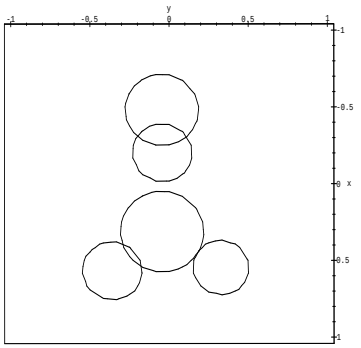
Weighting function	Learn MSE	Test MSE
W1	$3.9 \cdot 10^{-3} \pm 2 \cdot 10^{-4}$	$4.0 \cdot 10^{-3} \pm 2 \cdot 10^{-4}$
W2	$2.3 \cdot 10^{-3} \pm 1 \cdot 10^{-4}$	$2.5 \cdot 10^{-3} \pm 1 \cdot 10^{-4}$

Figure 4.12: *MSE's and standard deviations for* 5 *experts,* 1000 *samples and local model Poly0.*

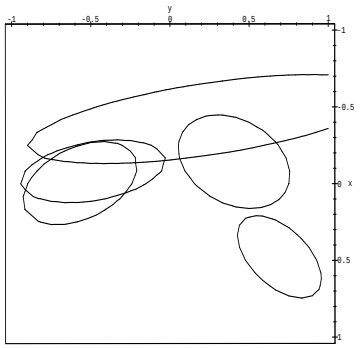
In order to explain these results, we have to take a closer look at the positions and shapes of the weighting functions after learning. Figure 4.13 shows this information after 5 experts were trained using either the W1 or W2 weighting function. In both cases, the initial positions and shapes of the kernels were the same.



Initial configuration, before learning.



Final configuration, using W1.



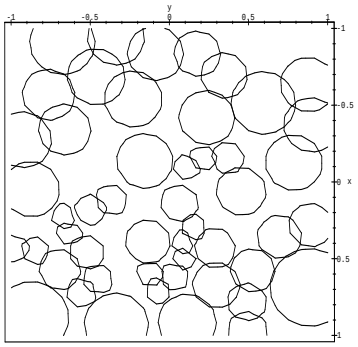
Final configuration, using W2.

Figure 4.13: *Configuration of weighting functions before and after learning, using* 5 *experts and* 100 *learning samples. The plots show the contours* $\mathcal{W}_i(\mathbf{x}) = 0.5$.

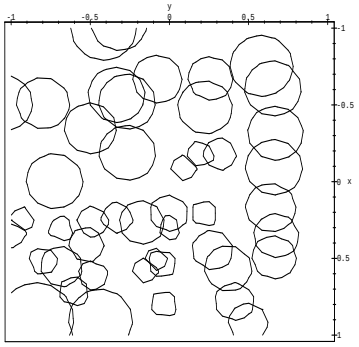
As can be seen, the final configurations of the two networks show little similarity. Apparently, for each network a different optimization route was followed, which does not even result in the same kernel positions. Although the configuration in W2 looks strange, it actually gives a better approximation than the network using W1 (a test error of 0.010 versus 0.013).

For a large number of experts, the situation is completely different, as can be seen in figure 4.14. The configurations resulting from training W1 and W2 are similar; the positions are almost the same, as are the shapes of the kernels. Even in the network using W2, all kernels are circular. This means that the learning algorithm did not use the weighting and rotation possibilities provided by weighting function W2.

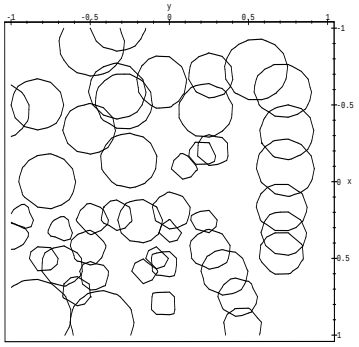
Further study indicated that the derivatives of the local model parameters to the MSE were relatively much higher than the derivatives of the weighting parameters to the error. Since the conjugate gradient algorithm makes changes in the parameters proportional to the derivatives, this means that the local model is updated much faster than the weighting functions. Furthermore, the parameters of the weighting functions are only changed very slightly during learning, even when the learning was continued after 500 iterations. This can



Initial configuration, before learning.



Final configuration, using W1.



Final configuration, using W2.

Figure 4.14: *Configuration of weighting functions before and after learning, using* 50 *experts and* 100 *learning samples. The plots show the contours* $\mathcal{W}_i(\mathbf{x}) = 0.5$.

also be seen from figure 4.14. Why exactly the parameters of the weighting functions are changed so little remains unclear.

This explains the small difference between the use of W1 and W2 for a large number of kernels: since the kernels of W2 are almost not scaled or rotated, these kernels become equivalent to kernel type W1.

4.5 Conclusion

This chapter discussed various settings which have to be made for the locally weighted approximation model. In every situation, estimating the number of experts needed is crucial. If too few experts are used, both learning and test error will be large, caused by the small flexibility of the network. If too *many* experts are employed, the learning error will be very small, but overfitting causes a non-optimal test error: a result of the network being *too* flexible. So the number of experts has to be chosen somewhere in between, where the minimal test error occurs. The exact position of this minimum depends on the number of samples used, and on the destination function $\mathcal{D}(\mathbf{x})$. The optimal number of experts is hard, if not impossible, to predict. However, since the problem is essentially the same as finding the optimal number of hidden units in a feed forward neural network, existing heuristic prediction methods might be applicable [31].

The effect of local model and weighting function depends on the number of experts. When rather many experts are used, different local models and different distance functions do not affect the resulting approximation very much, although more flexible local models like Poly2 sometimes cause strange (but local) overfitting. This is not much difference, because large part of the flexibility of the network is determined by the many expert positions and weighting function widths. Therefore, the choice can be based on computational efficiency: taking local model Poly0 and distance function W1 is the most efficient choice. Note that the optimal number of experts found always resided in the 'rather large number of experts' range.

There is noticeable, but small, difference between the local models and weighting functions when few experts are employed. The local models then have to approximate relatively large regions, and the flexibility of the models is important; local model Poly2 or Poly3 should be used. The extra parameters of the weighting function W2 make the network more flexible than function W1, making W2 a better choice.

We have to remark that that these results are obtained by experimenting with one destination function only (Dest1); therefore, they need not be valid for every conceivable destination function.

Chapter 5

Experiments: comparison with other methods

5.1 Introduction

In this chapter the locally weighted polynomials (LWP) approach will be compared with two other widely used methods: feed forward neural networks (FFNN) described in section 2.5 and radial basis functions (RBF), described in section 3.4. The goal is to get an idea of the relative performances, by letting the algorithms approximate 3 different functions:

- The 1 dimensional Moments function, which will be described later, in section 5.2;
- The 2 dimensional function Dest1, introduced in section 4.3;
- A 4 dimensional function Dest2, to be introduced in section 5.4.

For all three methods, finding the optimal number of units (experts, hidden units, Gaussian kernels) is a problem. Therefore, all the experiments are done for a varying number of units, after which the optimal number can be located.

The previous chapter discussed the effect of different local models and weighting functions, and concluded that using local model Poly0 and weighting function W1 is an efficient choice which gives similar performance compared to the other local models/weighting functions. Therefore, we will use this type of locally weighted approximation network in this chapter for all the experiments.

The settings of the 3 approximation methods which will be used in the experiments are summarized below:

- LWP: Locally weighted polynomial approximation, using weighting function W1 and local model Poly0. Note that such networks are equivalent to normalized radial basis function networks;
- FFNN: Feed forward neural network with one hidden layer, sigmoid activation functions in the hidden layer and linear activation functions in the output layer;

- RBF: Radial basis functions, where the widths and positions of the Gaussian kernels are optimized. The initialization procedure of the kernel widths and positions is the same as the one used by LWP (described in section 4.2.5).

The parameters of all three methods are optimized using the conjugate gradient algorithm; LWP and RBF networks were optimized for 500 iterations, FFNN for 1500 iterations.

Section 5.2 introduces the Moments test function and reports the experimental results of approximating it. Section 5.3 shows comparing experiments on the Dest1 function used in the previous chapter. The outcome of the comparison on the 4-dimensional function is given in section 5.4. Finally, section 5.5 concludes this chapter.

5.2 Comparison on the Moments function

The Moments function is given as[1]:

$$\begin{aligned} \mathcal{D}(x) \quad = \quad & \mathcal{G}(x, 0.074725, 0.131047, 0.908775) + \\ & \mathcal{G}(x, 0.255331, 0.104940, -0.534662) + \\ & \mathcal{G}(x, 0.392394, 0.178218, 0.644876) + \\ & \mathcal{G}(x, 0.580345, 0.079290, 0.605652) + \\ & \mathcal{G}(x, 0.987371, 0.131897, 0.539336) \end{aligned} \tag{5.1}$$

where $\mathcal{G}$ is a Gaussian kernel:

$$\mathcal{G}(x, p, \sigma, \alpha) = \alpha\, e^{-\frac{(x-p)^2}{\sigma^2}}$$

Using this function, 30 learning samples were chosen, as well as 200 test samples. They were drawn from a uniform input distribution; the samples and function itself are plotted in figure 5.1.

The three methods were tested on the Moments function. Results are plotted in figure 5.3. As can be seen, all three methods are sensitive to the number of units used. Using too few units results in poor approximations; using too many units sometimes results in some overfitting.

It is remarkable that increasing the number of units does not necessarily result in more overfitting, which we would expect, especially for the RBF networks. The reason can be revealed by taking a closer look at the RBF case.

Overfitting can occur, when narrow Gaussian kernels with a high 'top' are positioned in input space where no learning samples are nearby. This does not affect the learn error very much, but does create a high peak in the approximation. An example of this behaviour can be found in figure 4.10.

Now, for a large number of kernels, the initialization procedure (self organization) moves the kernels very close together and most of the kernel widths are initialized to the minimum

[1] This formula looks quite bizarre. Actually, this is because it is the result of an RBF approximation using 5 kernels. We did some earlier experiments with this function, and kept using it ever since...

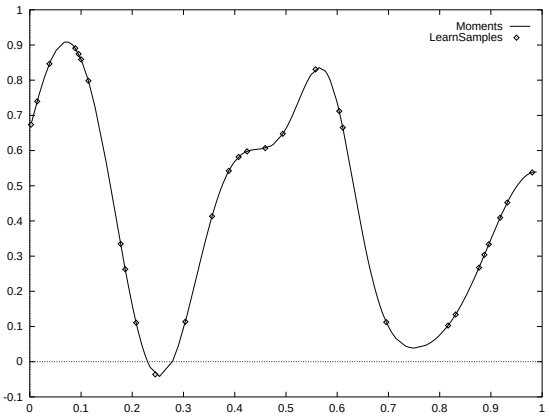


Figure 5.1: *The Moments test function. The dots indicate the* 30 *learning samples generated.*

width (as described in section 4.2.5). This minimum is still wide enough to prevent the occurrence of high peaks in the approximation, so at least for the initial parameters (before learning) major overfitting is avoided. During learning, the same happens as was described in section 4.4.3: the parameters of the local models are changed rather fast, but the weighting function parameters (widths) show very little and slow change. Therefore, the kernels do not shrink substantially, and major overfitting is not introduced during learning. However, the kernels are not wide enough to prevent overfitting completely.

How small kernels have to be to create peaks in the approximation depends on the number of learning samples and their positions in input space. Obviously, if more learning samples are present, the average distance between them is smaller; then a Gaussian kernel also has to be smaller to 'squeeze' between learning samples. Therefore, more learning samples means less chance of overfitting.

Furthermore, the self organization makes that the kernels move towards learning samples in input space. This also reduces the chance of overfitting, because a kernel first has to be moved away from learning samples by the learning algorithm before the situation described above can occur (no learning samples nearby).

The optimal number of units for approximating the Moments function is given in table 5.2, together with the errors. Note that the optimal numbers are not very representative, because of the large standard deviations. For example, a LWP network with 40 experts has almost the same test error as a similar network with 20 experts; the same is true for the other two networks.

Algorithm	Number of units	Learn MSE	Test MSE
FFNN	70	$1.3 \cdot 10^{-4} \pm 4 \cdot 10^{-5}$	$4.0 \cdot 10^{-4} \pm 7 \cdot 10^{-5}$
LWP	40	$0.0 \cdot 10^{-6} \pm 0 \cdot 10^{-7}$	$7.0 \cdot 10^{-5} \pm 5 \cdot 10^{-5}$
RBF	10	$5.9 \cdot 10^{-5} \pm 5 \cdot 10^{-5}$	$8.0 \cdot 10^{-5} \pm 5 \cdot 10^{-5}$

Figure 5.2: *MSE's and standard deviations for the optimal number of units.*

It can be concluded that both the LWP and RBF networks have about equal performance with optimal number of units; the FFNN network has a minimal test error which is one order of magnitude higher.

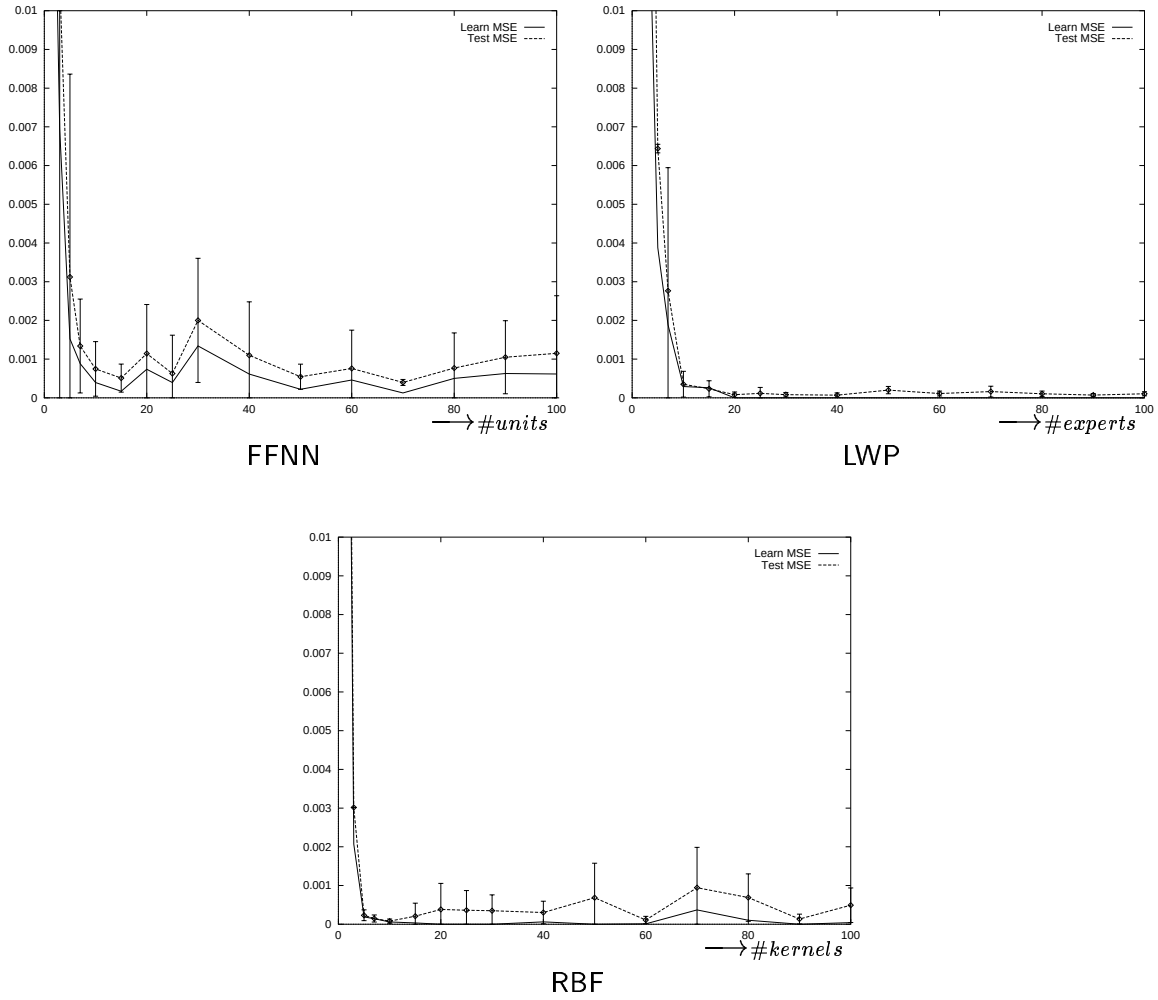


Figure 5.3: *Approximation MSE's of* Moments *function. The horizontal axis indicates the number of units used. Errorbars for the test MSE are standard deviations over* 5 *trials; some were removed to clarify the plots.*

5.3 Comparison on the Dest1 function

The two dimensional Dest1 function as described in section 4.3 was also used to compare the three different methods. Again, two learn sets were generated, with 100 and 1000 samples.

Learn and test errors on the *large* learning set are given in figure 5.6. The results are similar to the ones in the previous section: LWP performs best, followed by RBF which creates a little more overfitting. Again, very high overfitting does not occur because the Gaussian kernels are initialized and remain wide enough to prevent creation of large peaks in the approximation. The FFNN networks show relatively large learn and test errors.

Algorithm	Number of units	Learn MSE	Test MSE
FFNN	30	$9.1 \cdot 10^{-4} \pm 2 \cdot 10^{-4}$	$9.9 \cdot 10^{-4} \pm 2 \cdot 10^{-4}$
LWP	80	$1.6 \cdot 10^{-5} \pm 5 \cdot 10^{-6}$	$4.1 \cdot 10^{-5} \pm 4 \cdot 10^{-6}$
RBF	80	$2.3 \cdot 10^{-5} \pm 4 \cdot 10^{-6}$	$6.4 \cdot 10^{-5} \pm 1 \cdot 10^{-5}$

Figure 5.4: *Minimal test errors on target function Dest1 with* 1000 *learning samples.*

The approximation of the *small* learning set is a good test case to find out the generalization performance, when little information is available about the destination function. Figure 5.7 shows that there is indeed a large difference with the previous results. Again, the learn error of the FFNN network is generally larger than that of the other two methods, but generalization now is significantly better; also, this network never overfits the data.

The LWP network shows some overfitting when the number of experts becomes larger. Overfitting really becomes apparent with the RBF approximations: the test error becomes proportional to the number of kernels, because the kernels get small enough to move between the few learning samples.

To understand why overfitting for LWP networks is much lower than that for RBF networks, consider the following. In RBF networks, the network output is a weighted summation of Gaussian kernels. In LWP networks, the network output can be seen as a weighted summation of *normalized* Gaussian kernels (or the *activations* g_i, as defined in equation 3.2); see equation 3.3 and remember that the local model $\mathcal{N}_i(\mathbf{x})$ is a constant c_i. Because normalized kernels are much wider and less 'peaky' than simple Gaussian kernels (figure 3.4), it is less likely to create overfitting with it, in the way described in the previous section.

Algorithm	Number of units	Learn MSE	Test MSE
FFNN	80	$7.8 \cdot 10^{-4} \pm 9 \cdot 10^{-5}$	$2.3 \cdot 10^{-3} \pm 2 \cdot 10^{-3}$
LWP	7	$1.7 \cdot 10^{-3} \pm 8 \cdot 10^{-4}$	$2.8 \cdot 10^{-3} \pm 1 \cdot 10^{-3}$
RBF	20	$8.1 \cdot 10^{-5} \pm 4 \cdot 10^{-5}$	$5.7 \cdot 10^{-3} \pm 3 \cdot 10^{-4}$

Figure 5.5: *Minimal test errors on target function Dest1 with* 100 *learning samples.*

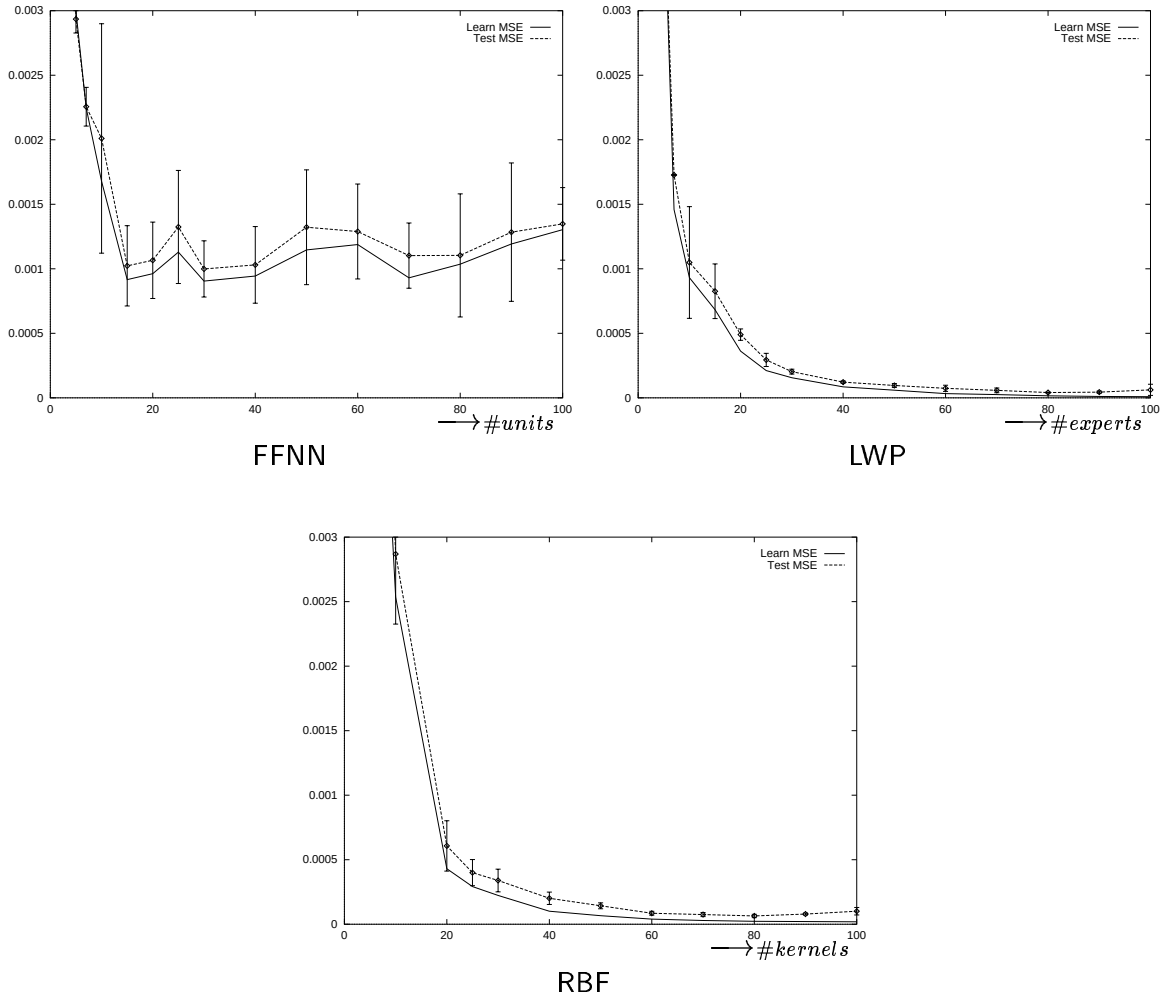


Figure 5.6: *Approximation MSE's of* Dest1 *function, using* 1000 *samples. The horizontal axis indicates the number of units used. Errorbars for the test MSE are standard deviations over* 5 *trials; some were removed to clarify the plots.*

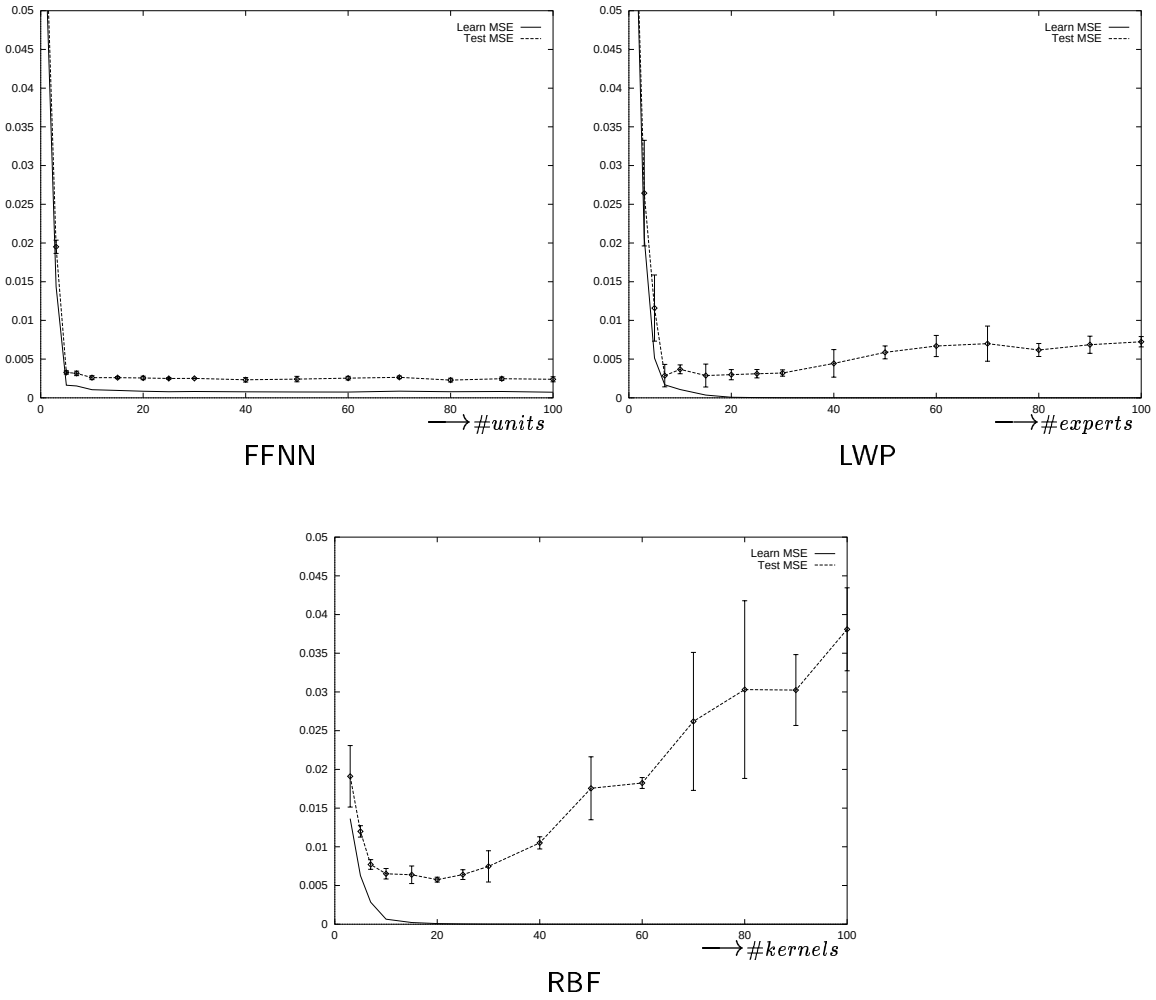


Figure 5.7: *Approximation MSE's of Dest1 function, using* 100 *samples. The horizontal axis indicates the number of units used. Errorbars for the test MSE are standard deviations over* 5 *trials; some were removed to clarify the plots.*

5.4 Comparison on the Dest2 function

The 2 dimensional Dest1 function was extended to a 4-dimensional version:

$$\mathcal{D}(\mathbf{x}) = \max \begin{cases} e^{-10\mathbf{x}_1^2} \\ e^{-50\mathbf{x}_2^2} \\ e^{-20\mathbf{x}_3^2} \\ e^{-30\mathbf{x}_4^2} \\ 1.25\, e^{-2(\mathbf{x}_1^2+\mathbf{x}_2^2+\mathbf{x}_3^2+\mathbf{x}_4^2)} \end{cases} \tag{5.2}$$

The learn set was composed by drawing 1000 samples from this function, in the same way as before. For 4 dimensions, 1000 samples is not a lot (to get the same number of samples per dimension as for Dest1 with 1000 samples, we would have to use 10^6 learn samples).

Approximation errors for the Dest2 function with the FFNN, LWP and RBF methods are shown in figure 5.9 and table 5.8.

Algorithm	Number of units	Learn MSE	Test MSE
FFNN	100	$6.0 \cdot 10^{-3} \pm 2 \cdot 10^{-4}$	$9.9 \cdot 10^{-3} \pm 6 \cdot 10^{-4}$
LWP	50	$4.4 \cdot 10^{-3} \pm 3 \cdot 10^{-4}$	$1.5 \cdot 10^{-2} \pm 5 \cdot 10^{-4}$
RBF	50	$1.1 \cdot 10^{-2} \pm 9 \cdot 10^{-4}$	$3.2 \cdot 10^{-2} \pm 1 \cdot 10^{-3}$

Figure 5.8: *Minimal test errors on target function Dest2.*

The results are comparable with those given in figure 5.7 (also approximating a small learning set): FFNN performs best and does not suffer from overfitting. Both LWP and RBF clearly have an optimal number of units after which test error increases. This optimal number is higher than the one found in table 5.5, because now more experts/kernels are needed to populate the 4 dimensional input space.

Another indication of the need for more experts/kernels is the learn error: to reach some specific learn error, more units are needed in the 4D space than in the 2D space.

5.5 Conclusion

Three approximation methods were compared in this chapter: FFNN, LWP and RBF. The results are very dependent on the number of units used, as well as on the number of learning samples. In fact, the absolute number of learning samples is not relevant, but the number of learning samples *per dimension* ξ is, which we define as:

$$\xi = \sqrt[d]{|S|} \tag{5.3}$$

where d denotes the number of input dimensions, and $|S|$ denotes the number of learning samples.

The value ξ is an indication of the number of learning samples per volume unit. For example, if 10 learning samples are used for one input dimension, we should have 10^2 learning samples for two input dimensions, in order go get the same amount of information in both cases.

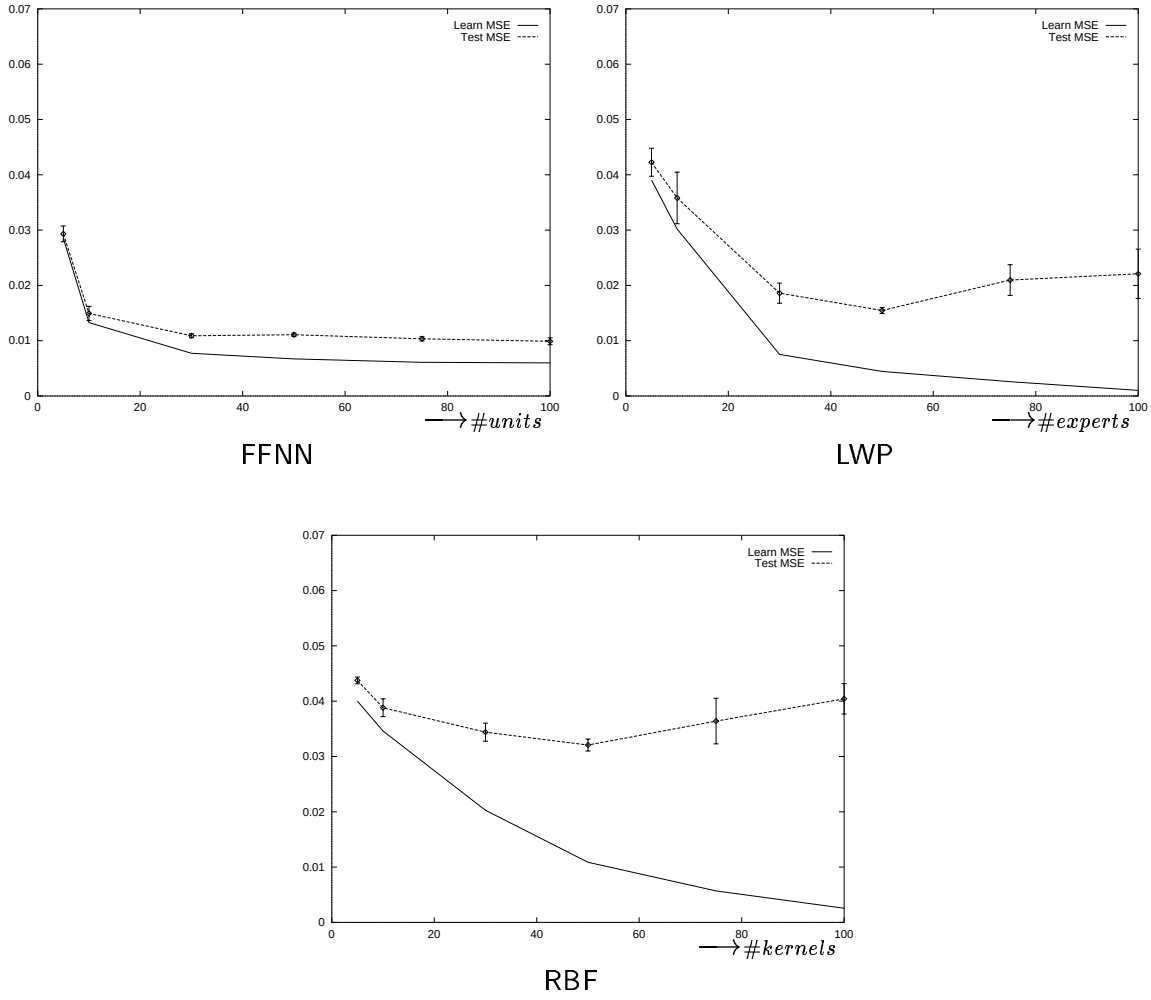


Figure 5.9: *Approximation MSE's for* 4*D* Dest2 *function, using* 1000 *learning samples.*

The number of learning samples per dimension ξ for the Moments function is 30; for the Dest1 function using 1000 learning samples, ξ equals $\sqrt{1000} \approx 33$. The experiments show that for this relatively high number of learning samples, the LWP and RBF methods create very good approximations and do not suffer much from overfitting. The FFNN networks perform worse: both learn and test error are higher.

On the other hand, the number of samples per dimension available for the other two test cases is much smaller: $\xi = \sqrt{100} = 10$ for the Dest1 function using 100 samples, $\xi = \sqrt[4]{1000} \approx 6$ for the 4 dimensional Dest2 function. The approximations formed by the FFNN networks were superior to the other two in these cases, and overfitting did not occur. Radial basis functions created relatively high overfitting, caused by the high peaks appearing in the approximations. The LWP networks performed somewhere in between.

It can be concluded that the three methods have quite different properties, due to the different ways of approximation. Note that the number of parameters used is about the same for each method: RBF and LWP have $N(d+2)$, where N is the number of units and d the number of input dimensions; FFNN networks have 1 more.

LWP networks are to be preferred when relatively many learning samples are available, FFNN networks in the other case. Radial basis functions easily suffer from overfitting, which make them not a very good choice for approximating smooth functions.

Chapter 6

Conclusion

6.1 Concluding...

This thesis discussed locally weighted approximations, which are applicable for approximating smooth functions based on a limited number of learning samples. They create these approximations by splitting the input space into overlapping regions and allocating a local expert to each region. The output of the network is a mix of the output of the individual experts; this is done in a way which ensures continuous approximations.

Different kinds of locally weighted networks were compared, varying the number of experts, type of weighting function and local model. A surprising result followed: the type of weighting function and local model tested do not influence the approximation performance, at least when the number of experts chosen is rather high. The other result, which was foreseen, is that the approximation performance is highly dependent on the number of learning samples and on the number of experts used. The number of learning samples should be as large as possible; finding the optimal number of experts is a problem, and can generally only be figured out by experimenting.

The locally weighted networks were compared with two other existing methods: radial basis functions and feed forward neural networks. Experiments performed on a few test functions indicated that locally weighted networks perform best when the number of learning samples (per dimension) is rather high; thereby outperforming the other two methods. On the other hand, when few learning samples are available, feed forward neural networks show better generalization. The number of learning samples forming the border between 'few' and 'many' depends on the problem (the destination function to be approximated); therefore it is generally hard to tell beforehand which of the two methods will perform best.

Radial basis functions created high overfitting earlier than the other two methods (i.e., already with a relatively large number of learning samples); the amount of overfitting is highly dependent on the number of Gaussian kernels used. Therefore this type of network seems less suitable for approximating smooth functions.

6.2 Future work

Many questions remain unanswered or were introduced. The approximation learned by a locally weighted network was found to be dependent on the initial widths of the weighting

functions; a heuristic method was used during the experiments, but what is the optimal initialization method? Furthermore, during learning with many experts, the weightings did not change very much. Why this happens is still not clear.

In all the experiments, the number of experts had to be set beforehand. It would be much nicer to have an incremental way of learning, where new experts are added during learning at places where they are needed. This is certainly not a trivial change, because it is not clear what the optimal place is to insert a new expert. However, for related approximation methods a lot of methods already exist, such as [4] [8] [11] [12] [17] [24] [30] [32].

In addition to developing particular methods, more has to be known about the theoretical background of function approximation. For example, why do methods like feed forward neural networks often generalize better than other (e.g., local) methods? More theoretical research might provide a better understanding of this.

Bibliography

[1] J.S. Albus. A new approach to manipulator control: The Cerebellar Model Articulation Controller (CMAC). *Transactions of the ASME, Journal of Dynamics Systems Measurement and Control*, 97:220–227, 1975.

[2] C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, August 1995.

[3] C. de Boor. *A practical guide to splines*. Springer-Verlag, New York, 1978.

[4] S. Chen, F.N. Cowan, and P.M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[5] William S. Cleveland and Clive Loader. Smoothing by local regression: principles and methods. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1995.

[6] P. Diaconis and M. Shahshahani. On nonlinear functions of linear combinations. *J.Sci.Statist.Comp.*, 5:175–191, 1984.

[7] D.L. Donoho and I. Johnstone. Projection-based approximation and a duality with kernel methods. *Annals of Statistics*, 17:58–106, 1989.

[8] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems II*, pages 524–532. Morgan Kaufman, 1990.

[9] Jerome H. Friedman. Multivariate adaptive regression splines. *The annals of statistics*, 19(1):1–141, 1991.

[10] J.H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76(376):817–823, December 1981.

[11] Bernd Fritzke. Growing cell structures — a self-organizing network for unsupervised and supervised learning. Technical Report TR-93-026, International Computer Science Institute, Berkely, CA, May 1993.

[12] Federico Girosi, Michael Jones, and Tomaso Poggio. Priors, stabilizers and basis functions: from regularization to radial, tensor and additive splines. Memo 1430, Center for Biological and Computational Learning, MIT, June 1993.

[13] J.P. Gram. Über Entwicklung reeller Functionen in Reihen mittelst der Methode der kleinsten Quadrate. *J. Math.*, 94:41–73, 1883.

[14] J.M. Hoem. The reticent trio: some little-known early discoveries in life insurance mathematics by l.h.f. oppermann, t.n. thiele and j.p. gram. *Inter. Stat. Rev.*, 51:213–221, 1983.

[15] J.H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R. Thagard. *Induction: processes of inference, learning and discovery.* Computational Models of Cognition and Perception. M.I.T. press, Cambridge (MA), 1986.

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

[17] A. Jansen, P. van der Smagt, and F. Groen. Nested networks for robot control. In A. F. Murray, editor, *Applications of Neural Networks*, pages 221–239. Kluwer Academic Publishers, Dordrecht, the Netherlands, 1995.

[18] Steve Lawrence, Ah Chung Tsoi, and Andrew D. Back. Function approximation with neural networks and local methods: bias, variance and smoothness. In *Australian Conference on Neural Networks*, 1996. To appear.

[19] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[20] John Moody and Christian Darken. Learning with localized receptive fields. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 133–143, San Mateo, 1988. Morgan Kaufman.

[21] M.J.D. Powell. Restart procedures for the conjugate gradient method. *Mathematical programming*, 12:241–254, 1977.

[22] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[23] D.E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.

[24] Stefan Schaal and Christopher C. Atkeson. From isolation to cooperation: an alternative view of a system of experts. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8.* MIT Press, 1996. In press.

[25] T. J. Sejnowski and C. R. Rosenberg. NETtalk: A parallel network that learns to read aloud. Technical Report JHU/EECS-86/1, John Hopkins University Department of Electrical Engineering and Computer Science, 1986.

[26] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, School of Computer Science, Carnegie Mellon University, March 1994.

[27] A.N. Tikhonov and V.Ya. Arsenin. *Solutions of ill-posed problems.* Winston, Washington, 1977. Translation from the Russian "Metody resheniya nekor rektnykh zadach".

[28] Julius T. Tou and Rafael C. Gonzalez. *Pattern recognition principles*, volume 7 of *Applied mathematics and computation.* Addison-Wesley, Reading (MA), 1974.

[29] Unnikrishnan and Kootala P. Venugopal. Learning in connectionist networks using the alopex algorithm. *Proceedings of IJCNN*, pages 1926–1931, 1992.

[30] P. van der Smagt and F. Groen. Approximation with neural networks: Between local and global approximation. In *Proceedings of the 1995 International Conference on Neural Networks*, pages II:1060–II:1064, 1995. (Invited paper).

[31] V. Vyšniauskas, F. C. A. Groen, and B. J. A. Kröse. The optimal number of learning samples and hidden units in function approximation with a feedforward network. Technical Report CS-93-15, Dept. of Comp. Sys, Univ. of Amsterdam, Nov. 1993.

[32] V. Vyšniauskas, F.C.A. Groen, and B.J.A. Kröse. Orthogonal incremental learning of a feedforward network. In Fogelman-Soulie and Gallinari, editors, *Proceedings of the 1995 International Conference on Neural Networks*, 1995.