Appendices.

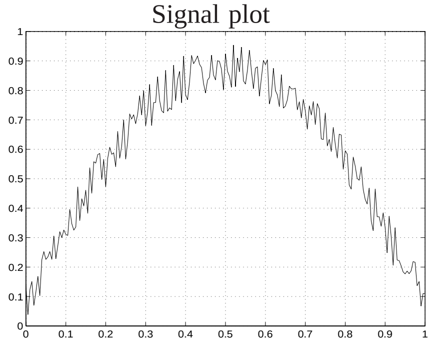
Appendix A: KRS–experiments.

The following figures represent different training cases, which has been performed to show the results from sample reordering algorithm. Every case is described by three figures. The first shows the original training signal; the second shows KRS plots by different example reorderings. The third plot represents either a snapshot from the Interact visualizer, or the generalization curves when testing. Some of the plots also appear in various places in the thesis. The systematization made here aims to give a quick overview of more made experiments and an easy way for their comparison and to allow a easy reference to various experiments, when the pictures are not really necessary to be plotted within the text. One is refered to table 5–1 for more numerical data.

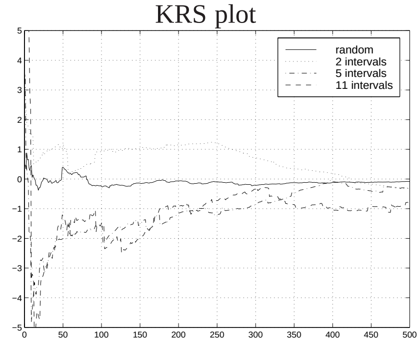
A.1 Signal No 1.

Sinewave with noise (see also figure 5–2).
It shows, how even the addition of noise on the input signal does not remove the potential internal cancelation. However, dividing the signal into intervals removes the tendence to paralysis.

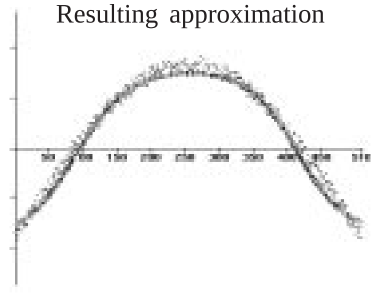
Signal plot



KRS plot

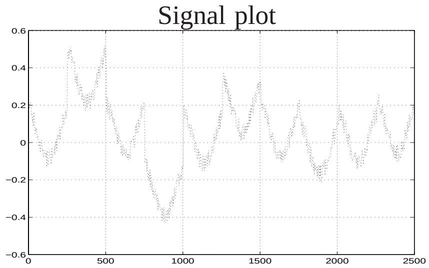


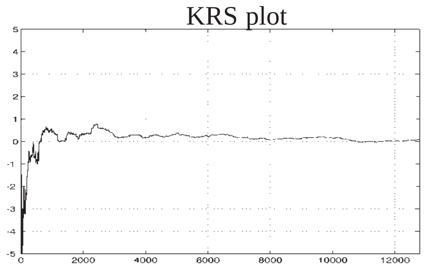
Resulting approximation

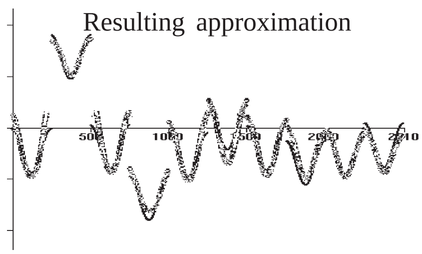


A.2 Signal No 2.

Complex goniometrical signal (see also figure 4–7).
The envelope of the signal is learned very fast, but then the internal cancelation prohibits any further progress.

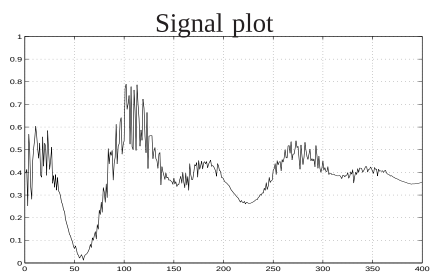


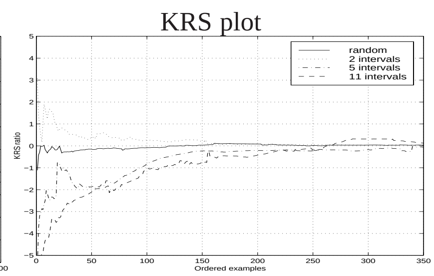


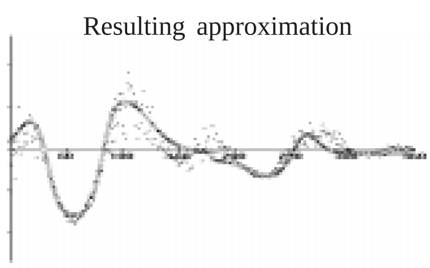


A.3 Signal No 3.

Not appearing elsewhere in this thesis.
An arbitrary signal with local symmetry and additional noise has learning problems unless the input space is divided in small enough intervals.



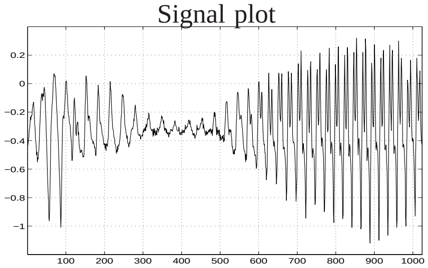




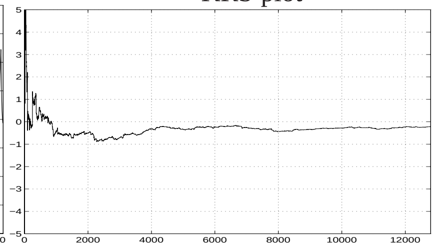
A.4 Signal No 4.

Power generator signal (see also figure 5–5a).
Approximation for the signalpart corresponding to the first 400 timesteps is limited to the training of the envelope.

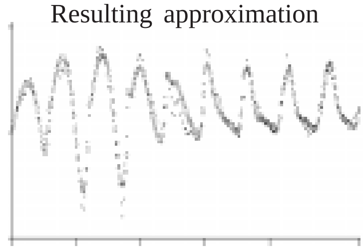
Signal plot



KRS plot



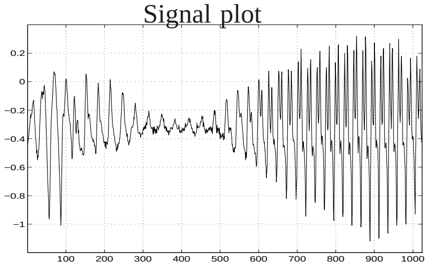
Resulting approximation



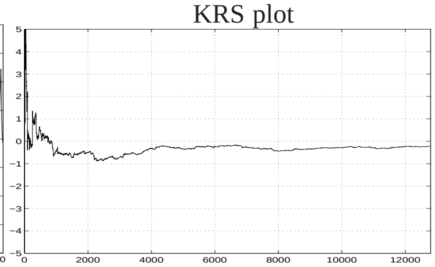
A.5 Signal No 5.

Power generator signal (see also figure 5–5a).
Approximation for the signalpart corresponding to the first 512 timesteps will not even learn the envelope for longer time fragments.

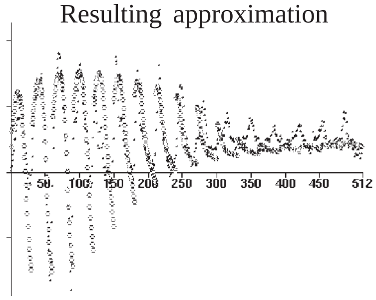
Signal plot



KRS plot

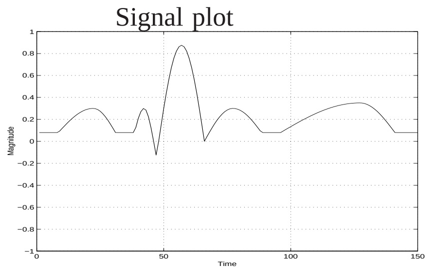


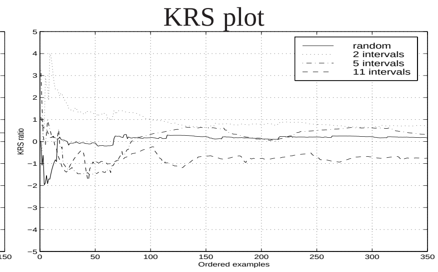
Resulting approximation

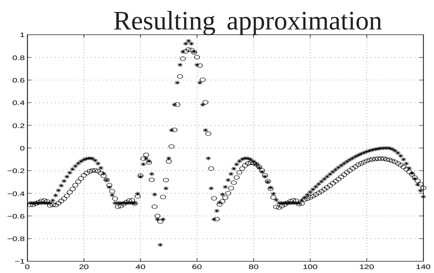


A.6 Signal No 6.

Single QRS signal (see also section 5.3.2).
The signal with some global but foremost local symmetry becomes (better) trainable when the number of intervals is raised.

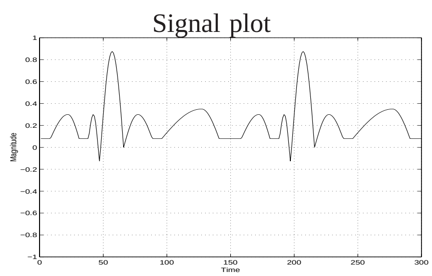


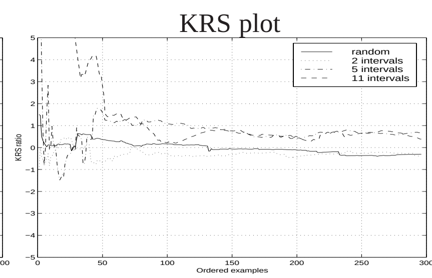


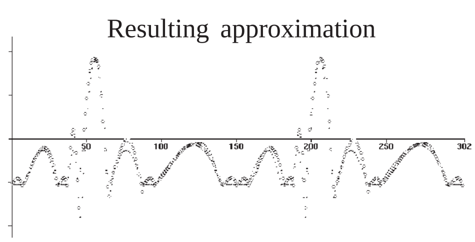


A.7 Signal No 7.

Double QRS signal (see also section 5.3.2).
For the larger signal length the global symmetry is less apparent and training is already performed for the smaller interval numbers.



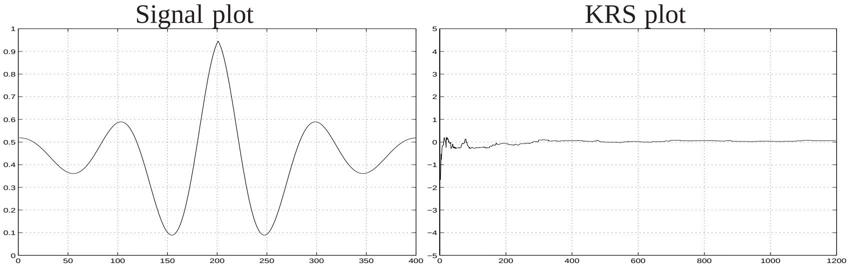




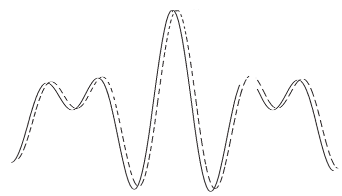
A.8 Signal No 8.

Symmetrical signal with second–order problem (see also figure 5–1a).
A complex signal with high global symmetry will have already have problems at the onset of learning.

Signal plot KRS plot



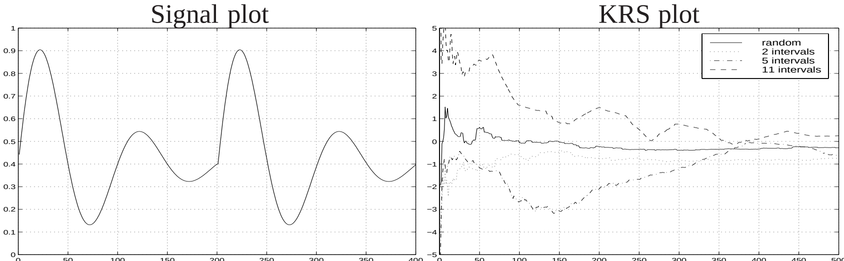
Resulting approximation



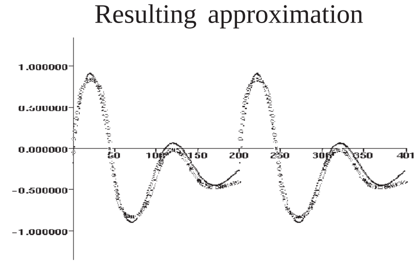
A.9 Signal No 9.

Asymmetrical signal with second–order problem (see also figure 5–1b).
A complex signal with little global symmetry will encounter local symmetries later in the training process.

Signal plot KRS plot



Resulting approximation



Appendix B: The Random walk.

There are numerous intuitions about the outcome of myriad experiments with events of a random nature, which happen to be wrong. The random walk theory reveals this misjudgment and is a basis for more advanced theories. For the analysis, made in this thesis, several properties of a random walk are interesting. First, it is of use to know how often the successive cumulative gains are becoming zero. This quantification is related to getting a notion of how fast network parameters can degrade to the zero point during adaptation. Second, it is of interest how the random walk depends on the length of the step. This gives inside to the problem why the initial symmetrical phase, when the network parameters has small values, is the most often encountered degradation problem. The third interesting question is what the spread and distribution of the endpoints by many random walks is, which will help to understand how many experiments, which because of the reasons explained in the thesis are governed nearly by the lows of the random processes, have the chance to escape the stationary areas, and how this chance is related to the number of training examples in a cancelation set.

B.1 The random walk in one direction.

A ideal coin–tossing game is a well accepted way to describe the random walk problem. The outcomes of individual tosses are represented geometrically on a rectangular coordinate system with horizontal t–axis and vertical y–axis. Every point on this coordinate system has as an abscis the number p of the current trial and as an ordinate the partial sum of the previous coin tossings s_p. All the partial sums draw a path $(s_1, s_2, \ldots, s_p)$. Every path is the outcome of a random walk experiment. Correspondingly the statistical characteristics of the multitude of paths can be quantified.

The probability, that at epoch n the path S has reached the point r, is denoted by $p_{n,r}$.

$$p_{n,r} = P\{S_n = r\} = \binom{n}{\frac{n+r}{2}} 2^{-n} \tag{B–1}$$

To the investigations, made in this thesis, the case is interesting, when the point r is the zero point. In the theory of random walks, this case is known as a *return to the origin.*

A return to the origin occurs at epoch k, if $S_k = 0$. Here k is necessarily even, and for $k = 2\nu$ the probability of a return to the origin equals $p_{2\nu,0}$. Because of its frequent occurrence this probability it will be denoted by $u_{2\nu}$.

$$u_{2\nu} = \binom{2\nu}{\nu} 2^{-2\nu} \tag{B–2}$$

The probability, that the first return to the origin occurs at epoch $2n$ is given by the following equation:

$$f_{2n} = \frac{1}{2n - 1} u_{2n} \tag{B–3}$$

Another quantifiable characteristic for a set of paths is the spread of the endpoints. It is defined by the probability, that the maximum of a path of length n leading to point $A = (n, k)$ and having a maximum $\geq r$ with $k \leq r$ is denoted by $p_{n,2r-k} = P\{S_n = 2r - k\}$.

B.2 Generalizing the random walk.

All the conclusions made in the previous section concern the one–dimensional random walk by which the step length equals to unity. They can be illustrated with the outcome of a coin–tossing game. Here generalizations for multi–dimensional random walks as well as for random walks with unequal step length will be made.

In a two–dimensional random walk it can be imagined, that a particle moves in unit steps in the four directions, parallel to the $t-$ and $y-$ axes. For a particle which starts from the beginning of the coordinate system, there are four possible positions which have real–valued coordinates. Similarly, in three dimensions every point has six neighbors. The random walk is then specified by the corresponding four or six probabilities. The generalizations will be made for a symmetric random walk, where all four or six directions are equally probable. The probability of a return to the origin is:

$$u_{2\nu} = \frac{1}{4^{2n}} \sum_{k=0}^{n} \frac{(2n)!}{k!k!(n-k)!(n-k)!} = \frac{1}{4^{2n}} \binom{2n}{n} \sum_{k=0}^{n} \binom{n}{k}^2 \qquad \text{(B–4)}$$

Equation (B–4) can be generalized for a higher dimensional case.

By the generalized one–dimensional random walk the restriction, that the particle moves in unit steps is avoided. In this case at each step the particle shall have the probability p_k to move from any point x to $x + k$. where the integer k can be zero, positive or negative. This generalization is also known as *sequential sampling*.

B.3 Interpretation.

The theory of random walks is not (or at least not directly) applicable to neural networks. On each presentation, the weights will be adapted. In other words, each state of the neural network represents a different history and therefore a permuted input set will lead to a different state. Nevertheless, the characteristics of a non–learning neural network seem comparable. This can be interpreted in the following way. When a neural network is still in a state of infancy, it makes no difference what this state is: by providing a canceling input stream all that has been learned can be unlearned. It is only when a neural network has matured and “knows” what to do, that the different streams have a reduced input.

It is clear, that a neural random walk requires an enhanced theoretical model. The reason why this has not been attempted here, is largely that we intended to eliminate the occurrence of a longest ruin by construction rather than by analysis. Despite that, such a model will still be a welcome addition to the theory of neural design.

Appendix C: Software.

As illustration to the provided algorithms, we supply here the basic routine that allows for the reordering schemes as discussed in this thesis.

C.1 The Permutation procedure.

NR_PATTERNS: the overall number of patterns
UINT_MAX: largest unsigned integer

```
void            Permutation (int nr, int ptype)
/*______________________________________________________________________*/
/* AIM:         select interval and permute the elements within         */
/*                                                                       */
/* INPUT:       int     nr      window count                            */
/*              int     ptype   permutation style:  INDEXSWAPPED        */
/*                                                  BOTHSWAPPED         */
/*                                                  VALUESSWAPPED       */
/*                                                  PERMWithOFFSET      */
/*______________________________________________________________________*/
{               int     i;                  /* example index within window */
                int     val;                    /* example to be exchanged */
                int     j;                  /* support variable in exchange */
                int     k;                          /* exchangable example */
                int     size;                     /* # patterns per window */
                int     memint[NR_PATTERNS];               /* example set */
                int     pl[NR_PATTERNS];              /* presentation set */
                int     offset;                  /* index of window start */
                int     ii;                               /* window index */
                time_t  t;                                  /* clock time */
 unsigned       int     seed;                             /* random value */

/* Initialize for random value generation ................................ */
                time (&t);
                seed = t%UINT_MAX;
                srand48 (seed);

/* Initialize the presentation array in increasing index order .......... */
/*      size: the number of patterns to be permuted .......................*/
                if (nr > 1) size = NR_PATTERNS / nr;
                else        size = NR_PATTERNS;

                for (i = 0; i < NR_PATTERNS; i++)  pl[i] = i;


/* Perform the different   permutations ................................. */
```

```
/* PERMWithOFFSET delivers  offset..offset+size   or
 offset..NR_PATTERNS-1; 0..offset+size MOD NR_PATTERNS .    */
        if (ptype == PERMWithOFFSET)
        {     for (ii = 0; ii < nr; ii++)
              {         offset = (int) floor(NR_PATTERNS *drand48());
                    for (i = 0; i < size; i++)
                    {   val     = (int) floor(((size-i)*drand48()));
                        val     = (offset+val)% NR_PATTERNS;
                        k       = (offset+i)  % NR_PATTERNS;

                      /* swap pl[offset+i] and pl[offset+val] ............ */
                        j = pl[k]; pl[k] = pl[val]; pl[val] = j;
                    }
              }
        }

        else if ((ptype == BOTHSWAPPED) ||
                    (ptype == VALUESWAPPED))
        {       for (k = 0; k < nr; k++)
                {     for (i = 0; i < size; i++)
                      {   val = k * size + (int) floor(((size-i)*drand48()));

                        /* swap pl[N-i] pl[val] ............................   */
                          j  = pl[(k+1)*size-i-1];
                         pl[(k+1)*size-i-1] = pl[val];
                         pl[val] = j;
                      }
                }
        }

        for (i = 0; i < nr; i++) memint[i] = i;

        if ((ptype == BOTHSWAPPED) ||
           (ptype == INDEXSWAPPED))
        {   for (i = 0; i < nr; i++)
            {       val = (int) floor(((nr-i)*drand48()));

                   /* swap m[N-i] m[val] ................................. */
                    j  = memint[nr-i-1];
                    memint[nr-i-1] = memint[val];
                    memint[val]    = j;
             }
        }

}
```

Index of Terms.

M

N

O

P

Q

R

S

T

U

W