Classification of Trajectories - Extracting Invariants with a Neural Network

M. Kinder*, W. Brauer

Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2
Germany

Abstract

A neural classifier of planar trajectories is presented. There already exist a large variety of classifiers that are specialized on particular invariants contained in a trajectory classification task such as position-invariance, rotation-invariance, size-invariance, ... That is, there exist classifiers specialized on recognizing trajectories e.g. independently of their position. The neural classifier presented in this paper is not restricted to certain invariants in a task: The neural network itself extracts the invariants contained in a classification task by assessing only the trajectories. The trajectories need to be given as a set of points. No additional information must be available for training, which saves the designer from determining the needed invariants by himself. Besides its applicability to real-world problems, such a more general classifier is also cognitively plausible: In assessing trajectories for classification, human beings are able to find class specific features, no matter what kinds of invariants they are confronted with. Invariants are easily handled by ignoring unspecific features.

Keywords: invariant representation; task-dependant similarity measure; topological, distributed encoding; Wickelfeatures; tuple coding; delta rule; perceptron

1 Introduction

Similarity determination is usually considered to be the basis of every classification task. Whenever an object needs to be classified, the similarity between it and the prototypes of all classes has to be determined. Therefore usually a definition of similarity is necessary. An appropriate definition, however, depends on the actual classification task. The following two examples of trajectory classification tasks will illustrate this.

1. Example task: Three classes described by prototype functions

   class *sinus*: $f_1(\mathrm{x}) = \sin(\mathrm{x})$
   class *logarithm*: $f_2(\mathrm{x}) = \ln(\mathrm{x})$
   class *squareroot*: $f_3(\mathrm{x}) = \sqrt{x}$.

2. Example task: Three classes described by prototype functions

*This work is partly based on a diploma thesis at the Technische Universität München and was originally conducted at Kratzer Automatisierung GmbH, Carl-von-Linde-Str. 38, 8044 Unterschleißheim. It has further been funded by the German Ministry for Research and Technology (BMFT) under FKZ ITN9102B. The authors are solely responsible for the contents of this publication. They would like to thank Dr. H. Geiger and his team at Kratzer Automatisierung GmbH for many fruitful discussions as well as for providing neural network tools.

$f(x) = sin(b_i x); \;\; b_i \in \mathcal{R}; \;\; i = 1, 2, 3.$

We obviously need different similarity measures for these example tasks, since in the first task, any function $f(x) = a * sin(b * x) + c$ is likely to belong to the class *sinus*, whereas in the second task the frequency b is the important feature for class distinction.

However, we do not wish to develop a specific classifier for each trajectory classification task. Instead, the classifier we describe in this paper is able to solve a broad range of tasks.

Besides this, there is another difficulty with similarity measures: The more complicated a task is, the more difficult it is, even for experts, to realize what the criteria for class distinction are. Which are the relevant features that set each class apart from all others? The frequency, position, size, shape... ? Thus, even an algorithm defining only the similarity for a *specific* task often is not easy to develop.

In order to avoid such difficulties, we use a neural network that learns the mapping of each trajectory to the class it belongs to. In this way the data (the current trajectories used for training the network) itself define the similarity measures. It is important that the network only extracts the features necessary for class distinction, ignoring specific features of the training examples which are not necessary for class description. This asks for the extraction of relevant invariants by the network by learning.

2 An Encoding that Enables a Network to Extract Invariants

We want to perform trajectory classification with a very simple network. A simple network however presupposes a complex encoding, which means that complexity is partly shifted to encoding. A motivation for such a procedure is of theoretical nature: A complex network only makes sense if the network structure somehow mirrors the structure of the task. Although, it is not clear how the extraction of task-specific invariants by a network should look like. But a transformation of the trajectories into a massively redundant representation is possible. Having generated such a representation, a simple network might be able to ignore a part of this representation and, as a consequence, produce invariants. After analyzing which parts of the representation seem to be informative, we will be able to perform the necessary operations in a network. This transformation from trajectories into a redundant representation will be referred to as encoding.

2.1 Network Architecture

The network we use is a fully connected feed-forward perceptron (Rosenblatt, 1957) without hidden layers (Figure 1). The unit's sigmoid squashing function is roughly approximated by three straight lines (Figure 2).

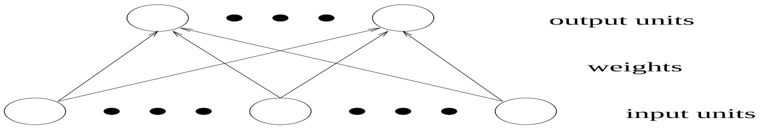

Figure 1: The network's topology: a perceptron without hidden layers

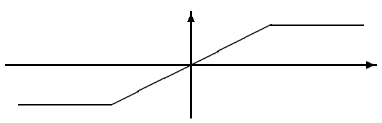

Figure 2: Units' squashing function

All networks are trained with the $\triangle$-rule only (Widrow & Hoff, 1960).

2.2 Encoding

There are two preconditions concerning the trajectories. First, the trajectories we want to classify need to be given by a sequence of points (x,y). Second, all points need to be available at once, which means that we are not confronted with a temporal series of data points. Please note that a simple means of generating these points of a trajectory is to use a function $f : y = f(x)$. This provides no loss of generality, since there is no difference in the classification task if there originally was a function defining the points of a trajectory or not. In this paper, we will sometimes use the wording function instead of trajectory.

There is no practicable straight-forward way to encode a trajectory in the input units of a network. Considering the set of points as one input pattern, the network will hardly be able to realize the similarity between a slightly modified trajectory and the class it belongs to. For example, consider a trajectory which is slightly stretched along the x-axis. y-values corresponding to subsequent x-values will differ more and more from the original ones, so that the network won't be able to realize the similarity. Another difficulty with this straight-forward solution is that the number of points in similar trajectories might often be different.

The following more complex encoding scheme can handle these problems. Figure 3 gives a brief overview of all steps to be performed, starting with sets of points and resulting in an activation pattern of the input units. Each step will be explained and discussed in detail below.

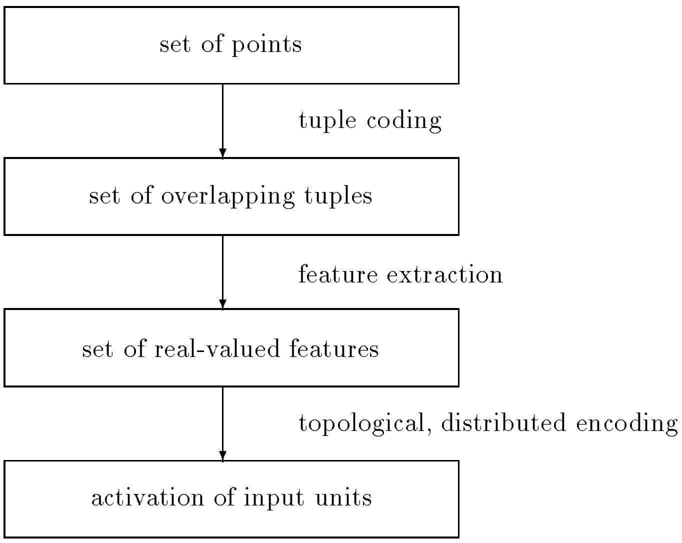

Figure 3: Overview of Encoding

2.2.1 Tuple Coding of an Ordered Set of Points

We derive tuples in analogy to Wickelgren's method. He proposed a set of tuples as a position-invariant encoding of words as an alternative to the well-known representation of words as a sequence of letters (Wickelgren, 1969). A word is represented as an unordered set of overlapping tuples of three consecutive letters. These tuples are called Wickelfeatures.

$$\text{TRAJECTORY} \Longleftrightarrow \{ \text{TRA, RAJ, AJE, JEC, ECT, CTO, TOR, ORY} \}$$

Please note that Wickelfeatures are not a unique representation of words. Different words might consist of the same Wickelfeatures. Even though, it is worthwhile to use Wickelfeatures as a representations of the input data of neural networks. If e.g. there is one letter missing in a misspelled word, the resultant set of Wickelfeatures it consists of is still similar to the one of the correctly spelled word. For more details and experimental results see (Eder, 1990). A similarity measurement based on the usual representation of words may yield a poor result if comparison is done position-wise. Characterizing tuple representations at a more abstract level, all the information is stored as a (unordered!) set of local informations in the sense of the neighborhood of each letter, being relative position-invariant compared to the normal representation.

These are interesting features for any task where position-invariance without loss of information is important, as e.g. for our task of trajectory classification. The transfer from words to sequences of points is straight-forward:

Figure 4: Tuple representation of a trajectory analogue to Wickelfeatures: One trajectory, approximated by a polygon, is represented as a set of overlapping tuples of line segments

This representation of trajectories will be referred to as tuple representation. One segment of a trajectory corresponds to one letter in a word. Instead of grouping three letters into a Wickelfeature, we group three segments into a tuple.
But how can we derive these segments from real world trajectories? The segmentation of a word into letters is predefined, whereas a segmentation of trajectories is not. There are two aspects with this segmentation that need to be considered:

1. properties of any approximation of a functions by a polygon:
   Too many segments are fairly sensible to noise, a too small number of segments does not represent details of the trajectory.

2. properties of the tuple representation:
   Too many segments lead to highly local tuples. Due to ignoring the order of these tuples in the tuple representation, much information gets lost. A too small number of segments on the other hand entails a more or less position-dependant representation, ruining the advantages of tuple representation.

We perform segmentation in the following way:
The first point of a trajectory is also the first point of the approximating polygon. A tangency to the trajectory is drawn in mind through this point. The distance in y-direction between the tangency and the trajectory exceeds a certain value *max* at the next polygon point and so forth. *max* is unique for all trajectories of a classification task. The trajectories we used for experiments consisted of 5 to 12 tuples, to give an idea of what segmentation is like.

Now assume that the set of tuples, representing a trajectory, has been generated. How can this representation of a trajectory be mapped onto the activity of input units? If we mapped one set at a time onto input activities, we would unavoidably specify an order of the tuples – which would ruin all the advantages of a tuple representation. Consequently we rather map one tuple at a time onto input activities and classify tuples instead of trajectories. Therefore we define the class of each tuple as the class of the set it belongs to. This classification of tuples however does not yet solve the original task of classifying trajectories with a neural network. That is, after the tuple classification has been accomplished by the network, we need to derive a classification result concerning trajectories. One possibility is to compute the average of the classification results of all tuples in a set. There still is another major feature of classifying tuples instead of trajectories by the neural network, the comparability of tuples. The network can easily compare tuples, since corresponding features of tuples will be encoded - at different time steps - by identical units.

2.2.2 Feature Extraction from Tuples

Having chosen one tuple as the only input dimension, we now have to solve the problem of mapping one tuple onto activities of units. In order to achieve a representation of tuples which does not necessarily depend on the exact value of the points describing a tuple, we again cannot use the straight-forward solution of mapping these points onto activities of input units. Instead, we choose a massively redundant description of tuples, incorporating absolute and relative information describing the geometric shape of tuples. This redundancy is the reason that the proposed method of trajectory classification is not task specific but appropriate for a large variety of tasks. In each single task, the network learns to choose parts of the representation that provide the most class-specific characteristics for an appropriate similarity measurement. In detail, the geometry of a tuple is redundantly defined by both a description of its three segments and different relations among these segments:

- redundant encoding of each segment $\overline{(x_i, y_i)\ (x_{i+1}, y_{i+1})}$
  Instead of just mapping the values of x_i, y_i, x_{i+1} and y_{i+1} onto the activity of four units, we generate redundant features, also suggested by (Waschulzik, 1990). The following features are used to characterize one segment: x_i, y_i, $\Delta x_i := x_{i+1} - x_i$, $\Delta y_i := y_{i+1} - y_i$, $l_i := \sqrt{\Delta x_i^2 + \Delta y_i^2}$, $\varphi_i := arctan(\Delta y_i / \Delta x_i)$. We did not put much effort into the selection of these features yet. Further investigations may lead to better features that improve the classification abilities of the network.

- pairwise relations between two segments:
  Much information about the geometric shape of a tuple can be extracted by comparing the segments of a tuple pairwise, i.e. by comparing segment 1 with segment 2, segment 1 with segment 3 and segment 2 with segment 3. Comparison of two segments is achieved by applying the operations *difference*, *sum* and *quotient* pairwise to all dimensions $x_i, y_i, \Delta x_i, \Delta y_i, l_i, \varphi_i$ mentioned above. Again, these pairwise relations were determined intuitively.

Summarizing, each tuple $\{\ (x_i, y_i), \ldots, (x_{i+3}, y_{i+3})\ \}$ is encoded as a sequence of real values:
First, the redundant features describing one segment after another are given, then the values describing all operations performed between corresponding dimensions of two segments, one pairwise combination of segments after another.

2.2.3 Topological, Distributed Encoding

The last stage of encoding involves the mapping of the sequence of real numbers describing the shape of tuples onto the activity of input units. Here we only describe the mapping of one real number onto the activity of input units. All other numbers will be mapped in the same way. Several features are required:

- a real number should be represented in a distributed way, so that there are several adaptable weights per real number

- the mapping of 'similar' real numbers should entail a similar unit's activation distribution (i.e. topological mapping)

- the mapping should be continuous

- sufficient resolution must be provided

These requirements are met by a mapping similar to the way acoustic information is encoded in the cochlea of mammals (Geiger, 1990). To encode one real number, we use several units ordered along a one-dimensional axis, each of them having a Gaussian-like curve as the activation function (i.e. as 'receptive field'). These activation functions are a special case of the well-known Radial Basis Functions (Poggio & Girosi, 1990). Neighboring activation functions overlap. A real number x is encoded by activating the units according to their activation function (Figure 5). For details see (Eldracher, 1992).

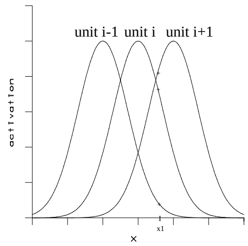

Figure 5: Topological, distributed encoding: x-axis: real number, y-axis: activations of three units according to their 'receptive fields' (Gaussian-like curves)

3 Experiments and Results

Unfortunately there is no way to check the usefulness of all steps performed in encoding separately. The only step we can check separately is the topological, distributed encoding. Experiments concerning this step and results are presented in (Eldracher, 1990). Several data sets were classified, each consisting of a set of real values, using topological, distributed encoding and the perceptron described earlier. That is, all steps we need to perform here for trajectory classification in order to determine the sequence of real numbers were eliminated in that task. The relevant results concerning the topological distributed encoding are:

- 10 units seem to be appropriate to encode one number.
- There exists a theoretical optimal value for the standard deviation of the Gaussian-like curves used as activation functions with respect to resolution (Kinder, 1990), which is 0.7 times the distance of nearest neighbor units. Experiments however show that resolution is not really the most important criterion for a good classification. Any standard deviation that entails the activation of about 5 or 6 units by encoding of one real number seems to be sufficient. Please note that this emphasizes the robustness of this encoding against parameter settings.

The following experiments report on the classification of trajectories. As stated earlier, the main requirement for a method to classify trajectories is task independence. The network itself is expected to discover task specific dimensions that should be invariant in the representation being developed while training. In order to test the capabilities of the presented method, we conducted several experiments with different expectations and will report on two of them:

1. parameter-invariant classification of four prototype functions
   - training data: classification of the four prototypes

sinus:	$f_1(x) = sin(x);$
parabola:	$f_2(x) = (x-1.0)^2 - 1.0;$
logarithm:	$f_3(x) = -ln(x);$
square root:	$f_4(x) = \sqrt{x} - 1.0;$
	$x \in [0.0, 7.0].$

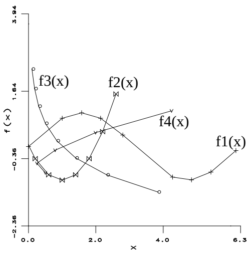

Figure 6: polygones of the prototypes

Only one function was chosen as the prototype of a class. If the network really did not specialize more than necessary on the training examples while training, we would expect the representation of these classes to be parameter-invariant.

- test data: To test the representation, we used functions with varying parameters. Since space is limited here, we will only report on details concerning the class *sinus*. The other classes were treated accordingly and showed similar results. The following parameters were taken in test functions:

  $f_{11}(x) = 1.5 * sin(x);$
  $f_{12}(x) = 2.0 * sin(x);$
  $f_{13}(x) = sin(1.5 * x);$
  $f_{14}(x) = sin(2.0 * x);$
  $f_{15}(x) = sin(x) + 1.0;$

- results: After 63 iterations on the training data, the network has learned to classify each training tuple correctly. Figure 7 shows the test result for the tuples derived from all test functions for the class *sinus*. All tuples are ordered along the x-axis. The y-axis shows the activity of output units, i.e. class units. The higher the activation of a class unit, the higher the estimate that the tuple belongs to this class. The activation of the class unit *sinus* is plotted as a square, that of the class unit *parabola* as a plus. (Lines between squares and pluses are only drawn for a better presentation.) Summarizing figure 7, there are 49 test tuples. 32 of them are classified absolutely correctly. 46 out of 49 are correct, if the associated class is defined as the class unit being maximally activated. In this case, only 3 tuples are classified incorrectly. The class units *square root* and *logarithm* are never activated by *sinus* test functions.

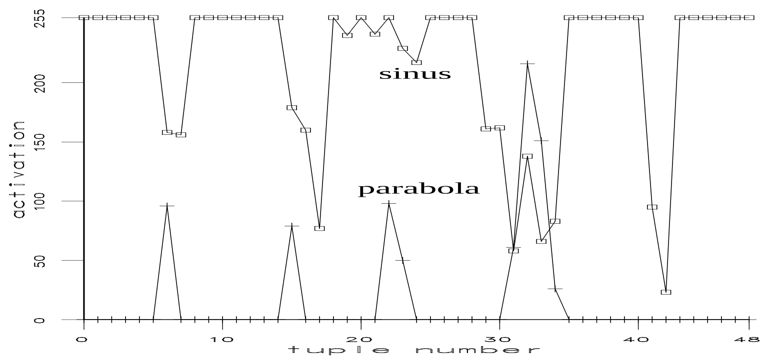

Figure 7: Classification results of the test functions of the class *sinus*: the five test functions consist of altogether 49 tuples. In this diagram, these tuples are ordered along the x-axis. Corresponding activations of class units are shown on the y-axis.

2. classification of different sinus-functions

   - training data:

     $f_i(x) = sin(b_i x), \; b_i \in \{1, 3, 5, 7\}.$

     Since there is only one training example for each class, there are several possible classification criteria. We assumed that the most important classification criteria might be features directly related to the frequencies.

$prototypefunctions$	$sin(x)$	$sin(3x)$	$sin(5x)$	$sin(7x)$
$testfunctions$				
$2*sin(x)$	4	6		
$4*sin(x)$	3	12		
$5*sin(x)$	3	14		
$2*sin(3x)$		2	4	8
$3*sin(3x)$			14	3
$4*sin(3x)$			13	8
$2*sin(5x)$			1	8
$3*sin(5x)$			5	3
$2*sin(7x)$				

Table 1: For each test function the number of tuples associated to each prototype function is shown. Note that the classification criterion cannot be the frequency.

- test data: functions with different amplitudes

  $f_{1j}(x) = a_{1j} * sin(x),\ a_{1j} \in \{2,4,5\}.$
  $f_{2j}(x) = a_{2j} * sin(3x),\ a_{2j} \in \{2,3,4\}.$
  $f_{3j}(x) = a_{3j} * sin(5x),\ a_{3j} \in \{2,3\}.$
  $f_{4j}(x) = 2 * sin(7x).$

- results:

  At a first glance (Table 1) the test functions were associated to *any* class - the representation is neither amplitude-invariant nor does it take the frequency into account. Test function $5 * sin(x)$ for example associates mostly to the class $sin(3x)$, test function $3 * sin(3x)$ mostly to the class $sin(5x)$. The question is: 'What else might be the classification creterion?'. Figure 8 gives the answer.

  In the upper row of figure 8, the 'shapes' of the prototype functions are compared to their most similar test function. It turns out that the similarity measure and thus the representation in the weights is size-invariant: Scaling of test functions leads to shapes similar to the shapes of the prototypes they associate to. The scaling factors used to get the bottom row of figure 8 were chosen seperately for each test function (they depend on the test functions chosen), whereas the scaling procedure itselt (i.e. the relation of the scaling factors in x and y direction) is the same for all scalings performed. The high degree of similarity reslulting from scaling emphasizes that the similarity measure is size-independent. Such a size-invariant and shape-specific simularity measure also gives an explanation to another question: Why does the test function $2 * sin(7x)$ not associate to any prototype? There is no prototype with a similar shape.

  This result of a shape-specific representation is very promising also for classifying real-world trajectories where no mathematical description is available: Regardless of whether there exists a mathematical formula and/or parameter values defining a trajectory or not, it is the shape which is used by the network as the classification criterion. Please note that such a criterion is cognitive, since it is mainly the shape which humans can take into consideration when looking at a trajectory. Also note that we do not want a neural classifier to explicitly depend on a mathematical description and distinct parameter settings, since for such tasks, classical parameter comparison methods appear to be more effective than neural networks.

  The classification result of the previous expamle also shows that much care has to be taken when choosing the training data. If a classification result does not match the expectations, the training data might specify the classes insufficiently. The addition of other trajectories to

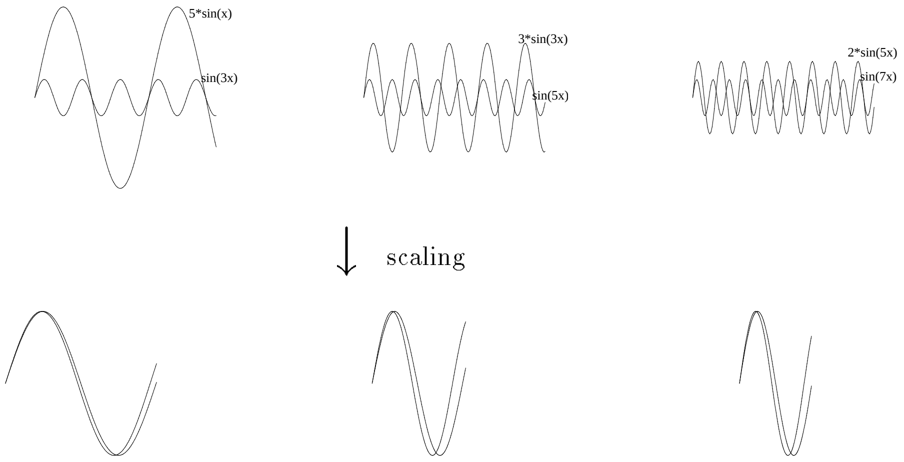

Figure 8: Upper row: the shapes of the prototypes compared to their most similar test functions. Recall that only one period of each function was shown to the network - several periods are drawn here only for a better comparison. Lower row: results of scaling of the test functions of the upper row.

the training data might entail the development of different classification criteria.

4 Summary and Conclusion

We introduced a neural trajectory classifier that is not only able to represent and classify trajectories e.g. in a size- or parameter-invariant way, but in addition is able to extract the relevant invariants, given the example data for each class. To achieve these properties, a massively redundant representation of the data was provided in the network's input units. This coding can be subdivided into three sequentially combined parts:

- *tuple coding* to represent local features of a trajectory independently of global trajectory features.
- *feature extraction* to generate a set of real numbers describing the tuples in a massively redundant manner,
- *topological, distributed encoding* to represent real numbers cognitively adequate.

Up to now we only use a simple perceptron without hidden layers to solve our task of extracting relevant invariants and representing them. Ongoing research comprises the application of this method to real-world trajectories generated by a robot's movement. A heuristics must probably be developed to define more task-independently at which points a trajectory should be devided into tuples. Furthermore, part of the encoding should be accomplished by the network. Thereby, we might be able to dispense with the uneconomic method of first generating and then ignoring parts of the redundant representation.

References

Eder, K. (1990). Satzerkennung mit konnektionistischen Methoden. Master's thesis, Technische Universität München, Institut für Informatik.

Eldracher, M. (1990). Klassifikation grosser Datenmengen anhand dynamisch extrahierter relevanter Merkmale mit neuronalen Netzen. Master's thesis, Technische Universität München, Institut für Informatik.

Eldracher, M. (1992). Classification of non-linear-separable real-world-problems using δ-rule, perceptrons, and topologically distributed encoding. In Berghel, H., Deaton, E., Hedrick, G., Roach, D., & Wainwright, R. (Eds.), *Applied Computing: Technological Challenges of the 1990's. Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pp. 1098–1104. acm Press.

Geiger, H. (1990). Storing and Processing Information in Connectionist Systems. In Eckmiller, R. (Ed.), *Advanced Neural Computers*, pp. 271–277. Elsevier Science Publishers B.V., Amsterdam.

Kinder, M. (1990). Repräsentation mehrdimensionaler, funktionaler Abhängigkeiten in neuronalen Netzen. Master's thesis, Technische Universität München, Institut für Informatik.

Poggio, T., & Girosi, F. (1990). Networks for Approximation and Learning. In *Proceedings of the IEEE*, Vol. 78, pp. 1481–1497.

Rosenblatt, F. (1957). The Perceptron – A Percieving and Recognition Automation. Tech. rep. 85-460-1, Cornell Aeronautical Laboratory, Ithaca New York.

Waschulzik, T. (1990) personal communications.

Wickelgren, W. (1969). Context-sensitive coding, associative memory, and serial order in (speech) behavior. *Psychological Review*, *76*, 1–15.

Widrow, G., & Hoff, M. (1960). Adaptive switching circuits. In *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4*, pp. 96–104.