A neural network for dollar recognition in money changing machines

Citation for published version (APA):

Document status and date:
Published: 01/01/1991

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
A NEURAL NETWORK FOR DOLLAR RECOGNITION IN MONEY CHANGING MACHINES

Ir. A.A.F. Bloemen
Ir. R.J. de Lange
Drs. J.E.O. Meuwissen

June 1991
A NEURAL NETWORK FOR
DOLLAR RECOGNITION
IN MONEY CHANGING MACHINES

Ir. A.A.F. Bloemen
Ir. R.J. de Lange
Drs. J.E.O. Meuwissen

June 1991
CONTENTS

0. PREFACE 1

1. PROBLEM DESCRIPTION AND RESULTS 2
   1.1. Introduction 2
   1.2. Problem Description 2
   1.3. Results 4

2. NEURAL NETWORKS 5
   2.1. Introduction 5
   2.2. A General Description 5
   2.3. A Specific Network for Dollar Recognition 7
   2.4. The Mathematical Model 8
       2.4.1. Introduction 8
       2.4.2. A Two-Layer Network 8
       2.4.3. A Multi-Layer Network 10

3. THE ALGORITHM 14

4. CONCLUSIONS AND REMARKS 16

Appendix A: NEURAL NETWORK TRAINING, RESULTS AND REMARKS 17

Appendix B: AVERAGE METHOD 20
0. PREFACE

This report treats a problem that was presented to us by Dr. W.G. Eschmann (Universität Kaiserslautern) during the course “Cases in Mathematical Modelling” in Eindhoven, August 1990. This problem deals with the recognition of dollar notes in money changing machines on the basis of light transmission- and reflection data that are acquired in the machine. We were asked to develop an algorithm that was able to do this.

In this report we describe the development of a neural network that is able to recognize dollar notes in a very short time (< 5 seconds) and with a very high reliability (> 99.9%). In the first chapter we will give a detailed description of the problem and the results. Technical and mathematical details about the neural network concept can be found in Chapter 2. In Chapter 3 we give some more information about the program that is able to recognize dollar notes. The last chapter gives some conclusions and remarks. We end this report with two appendices. Appendix A deals with the training and results of our neural network, whereas Appendix B deals with an other solution method we tried to solve the problem with: the Average Method.

We would like to thank J. Brands and S. van Eijndhoven for their useful contributions and M. Hooijkaas for coming up with the idea to try to solve the problem with a neural network. 1

---

1A poster presentation of this project has won the first prize in a competition organized by the Eindhoven University, april 1991.
1. PROBLEM DESCRIPTION AND RESULTS

1.1. Introduction

The last decade all kinds of electronic devices have been developed to facilitate transfer of payments. One may think of cash dispensers, electronic banking and money changing machines. This last device has our special interest.

In Germany in the neighbourhood of Kaiserslautern there are some villages that are especially built for men who serve the American army and for their families. Everything in these villages is American. One buys American food, one reads American newspapers and one pays with US dollars. In such villages there are some money changing machines. These machines are meant to recognize the note that is put inside it and then to return the right amount of money in other notes and coins.

There are seven types of dollar notes: $1, $2, $5, $10, $20, $50 and $100. The manufacturer of these machines wondered if it would be possible to design an algorithm for a money changing machine that would be able to recognize dollar notes on the basis of data that are acquired in the machine from light transmission and reflection. This might not be as easy as it seems at first sight. First of all the different US dollar notes are much alike in contradistinction to e.g. Dutch banknotes which are clearly distinguishable by their colour and size. Moreover the banknotes of one value are not identically because they suffer from every day use. For example, they may have been washed, teared or crumpled.

1.2. Problem Description

First we will describe what happens when a dollar note is put into a money changing machine. Secondly we mention which data are available and finally we discuss the required performance we have to develop.

When put into the machine the dollar note is guided to four lamps which scan the dollar note. These four lamps emit light of three different frequencies. Two lamps emit Infra-Red light ($IR$), one lamp emits Red light ($R$) and one lamp emits Green ($G$) light. Special detectors beneath and above the note measure the amount of light that is transmitted and reflected respectively while the note is moving. In a picture it looks as follows:
When the note moves 40 or 80 measurements are taken. Thus we end up with integer data in the collection \( \{0, \ldots, 255\} \) which are gathered into so called tracks in the following way:

- Track 1: Transm1 IR
- Track 2: Transm2 IR
- Track 3: Transm3 IR
- Track 4: Transm4 IR
- Track 5: Transm1 R & Transm3 R (2 \( \times \) 40 measurements)
- Track 6: Transm2 G & Transm4 G
- Track 7: Refl1 R & Refl3 R
- Track 8: Refl2 IR & Refl2 G
- Track 9: Refl4 IR & Refl4 G

For each type of note (i.e. $1, $2 etc.) the manufacturer of the machine presented the measurements of 10 new banknotes and 10 used banknotes all in any of the four orientations, because it is possible to put a note into the machine in four different positions. So we have 7 (different types of dollar notes) \( \times \) 9 (number of tracks) \( \times \) 80 (number of measurements per track) \( \times \) 20 (10 new and 10 old notes for every type of dollar note) \( \times \) 4 (number of orientations of a note) = 403200 measurements.
We were asked to develop an algorithm that can determine what type of dollar note is put into the machine on the basis of the 720 measurements measured on 9 tracks. This algorithm must be developed with the help of the 403200 test-measurements and must have a reliability of at least 99.9%. This means that only one of thousand real notes may be recognized wrongly or may be rejected. So at first we do not consider falsifications.

Moreover it is important to develop a fast algorithm. For a client it may not last too long before he gets back his money from the money changing machine.

1.3. Results

We have gone two basically different ways to develop an algorithm that would be able to recognize dollar notes. One is based on taking averages over a large number of data, whereas the other is based on the concept of neural networks.

The latter proved to be considerably better. Not only is the neural network algorithm able to recognize a higher percentage of dollar notes, also the time needed to do so is much smaller. From our calculations \(^1\) the neural network algorithm needs approximately 5 seconds to recognize a dollar note. The percentage of well-recognized notes is 100.00%. The corresponding numbers for the algorithm based on averages are respectively 1 minute and 90%.

From these figures it may be clear that we used the neural network idea to develop an algorithm to recognize dollar notes. With this report we supply a discette with an algorithm to recognize dollar notes based on the concept of neural networks. The main part of the algorithm consists of a function

\[
\varphi : \{0, 1, \ldots, 255\}_{9 \times 65} \rightarrow [0, 1]^7.
\]

This function maps every set of 9x65 data (only 65 of the 80 measurements per track appeared to be relevant) from an input dollar note to a 7-tuple \((\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6, \varphi_7)\). In \(\varphi_i\) we find the probability for an input note to be a $1, $2, $5, $10, $20, $50, $100 dollar note respectively. We say the dollar note is of type \(m\) if

\[
\varphi_m = \max_{1 \leq i \leq 7} \varphi_i \quad \text{and} \quad \varphi_m \geq 0.555.
\]

If none of the \(\varphi_i\)'s is greater than or equal to 0.555 we reject the note as being a falsification, although this does not need to be the case. It is also possible that the algorithm cannot determine on basis of the data what type of dollar note it is dealing with (for example because the dollar note lost its characteristics in consequence of every day use). \(^1\)

\(^1\)Calculations were done on a IBM AT computer with coprocessor
2. NEURAL NETWORKS

2.1. Introduction

In this chapter we will explain the algorithm we developed via the construction of a neural network. Some remarks on the other approach to solve the posed problem (taking averages) can be found in Appendix B.

In this chapter we will first explain the principles of a neural network. Secondly we will describe the specific network we have constructed for this specific problem and finally we will give some mathematical backgrounds of neural networks.

2.2. A General Description

A so called multi-layer feed-forward neural network in general consists of a number of layers \( l_1, l_2, \ldots, l_d \) say. These layers \( l_n, n = 1, 2 \ldots d \), are built up from so called units \( u_1(n), u_2(n), \ldots, u_m(n) \) which we can represent as nodes. The layers are interconnected by vertices \( v_{ij}(n - 1, n) \) that connect node \( u_i(n - 1) \) with node \( u_j(n) \). Every vertex \( v_{ij}(n - 1, n) \) is given a certain weight \( w_{ij}(n - 1, n) \), where \( k \) denotes a certain discrete time step, which will be explained later. So we can consider a neural network to be a time dependent special weighted graph.

More specific a unit \( u_j(n) \) looks like follows:

\[
\begin{align*}
\text{net}_j(n) &= \sum_{i} w_{ij}(n - 1, n) o_i(n) \\
\text{a}_j(n) &= F\left(\text{net}_j(n), \text{a}_{j-1}(n)\right) \\
\text{f}(n) &= f\left(\text{a}_j(n)\right)
\end{align*}
\]

where
\[ w_{ij}^k(n-1, n) = \text{the weight assigned to the edge } u_{ij}(n-1, n) \text{ in step } k \]

\[ \text{net}_j^k(n) = \text{the netto input in step } k, \text{ a weighted sum of outputs } o_j^k(n-1) \]
\text{of units } u_i(n-1), \text{ for which } v_{ij}(n-1, n) \text{ exists} \]

\[ a_j^k(n) = \text{the state of unit } u_j(n), \text{ in every step } k \text{ a function } F \]
\text{of the previous state } a_j^{k-1}(n) \text{ and the netto input } \text{net}_j^k(n) \]

\[ o_j^k(n) = \text{the output of unit } u_j(n) \text{ in step } k, \text{ a function } f \text{ of the state } a_j^k(n). \]

\( f \text{ en } F \) are free. Often the identity, a threshold function or a scale function are chosen.

The first and last layer are special. The first layer is called the input layer, whereas the last layer is called the output layer. Consequently the units in these two layers are called input and output units respectively.

All other nodes are called hidden units, since their connection with the initial input and final output stays hidden.

There are as many input units \( u_1(1), u_2(1), \ldots, u_m(1) \) as there are measurements from a pattern, because only then we are able to represent the complete pattern and no information is lost. Without changing the characteristics of the pattern we can achieve that the measurements are in the interval \([0,1]\) by some simple scaling. So the states of the \( m_1 \) input units are in the interval \([0,1]\).

The state of the hidden units and the output units is the outcome of a fixed non-linear function \( F \) applied to the weighted sum:

\[ a_j^k(n) = F \left( \sum_{j=1}^{m_{m-1}} w_{ij}^k(n-1, n) \cdot o_j(n-1) \right) \quad 1 \leq i \leq k, \quad 1 < n \leq d \]

and has values in \([0,1]\).

In pattern recognition we want to choose between a number of alternatives on the basis of a pattern we have measured. The output layer consists of as many units \( m_d \) as there are alternatives. The final state \( a_j(d) \) of an output unit \( u_j(d) \) gives the probability of being the corresponding alternative.

In our example the input is a vector of measurements in \( \{0, \ldots, 255\}^{585} \) (for each note put into the machine we have \( 9 \times 65 \) relevant measurements). The output vector represents the choice between 7 dollar notes.

This could be done with an output vector in \([0,1]^7\) in which each element gives the likelihood of being the specific note.

So eventually we arrive at a function which associates with each state of the input units \( (a_1(1), \ldots, a_{m_1}(1)) \) a state of the output units \( (a_1(d), \ldots, a_{m_d}(d)) \), where the state of all output units are in \([0,1]\).
The method used for pattern recognition is "learning by example" or "supervised training". For each sample \( k \) of a trial set, \( k = 1, \ldots, K \), with well known input and output vectors (the training set) the weights \( w_{ij}^{k+1}(n-1,n) \) are computed such that for each input vector of the trial set the eventual output vector is "as close as possible" to the known output vector.

If we are able to find such weights \( w_{ij}^{k+1}(n-1,n) \) it is likely that patterns can be recognized which correspond to input vector which resemble the input vectors in the training set, i.e. each dollar note has certain characteristics in common with dollar notes in the training set.

After each introduction of a dollar note from the training set the weights are changed such that the error between the desired output and the computed output is minimized. This minimization is done by implementing a steepest descent algorithm for changing the weights. In this way we construct a set of weights \( w_{ij} = \lim_{k \to K} w_{ij}^{k} \). These weights are such that the characteristics of the several types of dollar notes are represented. In this case "lim" means taking all samples from the training set.

### 2.3. A Specific Network for Dollar Recognition

We have constructed a neural network as follows:

When examining the data it appears that not all data are equally relevant. For all notes the last 15 measurements are 255 on a track with 80 measurements. On a track with two times 40 measurements the 34th until 40th measurement and 74th until 80th measurement are all 255. So an input vector will not be in \( \{0,1,\ldots,255\}^{80} \) but in \( \{0,1,\ldots,255\}^{65} \) (probably these measurements are taken, when the dollar-note has already passed the lamps that emit light). We have treated the 9 tracks separately, so we take \( m_{1} = 65 \).

As shown in the next paragraph a network with a hidden layer is at least as good as a single-layered network. So we construct an extra layer with so called hidden units, i.e. \( d = 3 \). The output vector is in \( [0,1]^{7} \), so there are 7 output units and so \( m_{3} = 7 \).

The first idea was to construct a hidden layer with 28 units. The network was supposed to distinguish between the different notes and their orientation in the first stage and to distinguish between the different notes only in the second stage. We thought it would be difficult for the network to distinguish between different notes in one step when it is trained with notes that are different in type and orientation.

After some computation time it proved that it would be profitable to take a larger number of hidden units. Although the computation time will increase, because more calculation is to be done, the results are better. The network is able to recognize more dollar-notes and the error-function is lower. Increasing the number of hidden units even more however is not a good thing to do, because the computation time is growing fast. So we have to find a balance between better results and increasing computation.
time when increasing the number of hidden units. We have found this balance to be 36 hidden units.

The state of the unit is the netto input, so the function $F$ is chosen to be the identity and independent of the previous state.

For $f$ we take the logistic function that is:

$$f(x) = \frac{1}{1 + e^{-x}}.$$

So every netto input is scaled to a number between 0 and 1. With the constructed network the training could begin. As remarked before we have done the training for the 9 tracks separately.

2.4. The Mathematical Model

2.4.1. Introduction

We will consider two cases:

I  The case where $d = 2$, i.e. there are only two layers: the input- and outputlayer.

II  The case where $d = 3$ and where the output of a hidden unit is a differentiable non-decreasing function of a weighted linear combination of inputs. The results derived for $d = 3$ nevertheless are easily generalized for $d > 3$.

In both cases we will derive a rule for changing the weights during the training. This rule implements a steepest descent method on a surface in weight space whose height at any point in weight space is equal to the squares of the differences between the actual and desired output values summed over the output units and all pairs of input/output vectors. For convenience of notation in the following we omitted the superscript $k$, denoting the discrete time step. Everything can be thought of as happening in step $k$.

2.4.2. A Two-Layer Network

Let

$$E_p = \frac{1}{2} \sum_{j=1}^{m_2} \{t_{pj} - o_{pj}(2)\}^2$$  \hspace{1cm} (1)

be the error function, with

$t_{pj} = \text{the desired output at output unit } u_j(2) \text{ with pattern } p$. $E_p$ is a measure of the error of a pattern $p$. Then
\[ E = \sum_p E_p \]  

is the overall error. We want to show that changing the weights according to:

\[ \Delta_p w_{ij}(1, 2) = \eta (t_{pj} - o_{pj}(2)) i_{pi}(1) = \eta \delta_{pj}(2) i_{pi}(1) \]  

implements a steepest descent method, when

- \( t_{pj} \) is the desired output,
- \( o_{pj}(2) \) is the actual output,
- \( i_{pi}(1) \) is the input,
- \( \eta \) is the size of the step in the steepest descent direction,
- \( \Delta_p w_{ij}(1, 2) \) is the change in weight on edge \( v_{ij}(1, 2) \) following pattern \( p \).

(3) is called the delta rule for obvious reasons.

If we can show that \( -\frac{\partial E_p}{\partial o_{pj}(2)} = \delta_{pj} i_{pi}(1) \) it follows that the delta rule does what we want it to do because \( \delta_{pj} i_{pi}(1) \) is proportional to \( \Delta_p w_{ij}(1, 2) \) in the delta rule.

From the chain rule we know:

\[ \frac{\partial E_p}{\partial w_{ij}(1, 2)} = \frac{\partial E_p}{\partial o_{pj}(2)} \frac{\partial o_{pj}(2)}{\partial w_{ij}(1, 2)}. \]  

From equation (1) we derive that

\[ \frac{\partial E_p}{\partial o_{pj}(2)} = -(t_{pj} - o_{pj}(2)) = -\delta_{pj}(2). \]  

Since we have linear units, that is

\[ o_{pj}(2) = \sum_{i=1}^{m_1} w_{ij}(1, 2) i_{pi}(1) \]  

it easily follows that

\[ \frac{\partial o_{pj}(2)}{\partial w_{ij}(1, 2)} = i_{pi}(1). \]  

So substituting (5) and (7) into (4) yields:
\(-\frac{\partial E_p}{\partial w_{ij}(1,2)} = \epsilon_{pj}(2) i_{pi}(1)\)

(8)
as desired. Observing that

\[\frac{\partial E}{\partial w_{ij}(1,2)} = \sum_p \frac{\partial E_p}{\partial w_{ij}(1,2)}\]

(9)finally gives the result that the delta rule implements a gradient descent in \(E\) after one complete cycle of representations.

2.4.3. A Multi-Layer Network

Now we have

\[o_{pj}(n) = f(net_{pj}(n))\]

(10)with

\[net_{pj}(n) = \sum_{i=1}^{m_{n-1}} w_{ij}(n-1,n) o_{pi}(n-1) .\]

(11)(Remark that the output of unit \(u_i(n-1)\) is the input for unit \(u_j(n)\) if \(v_{ij}(n-1,n)\) is an edge.)

Again we must show that changing the weights according to the generalized delta rule implements a steepest descent method. That is, we must show that

\[\Delta_p w_{ij}(n-1,n) \sim \frac{\partial E_p}{\partial w_{ij}(n-1,n)} .\]

(12)

In a similar way as before we derive with the chain rule that

\[\frac{\partial E_p}{\partial w_{ij}(n-1,n)} = \frac{\partial E_p}{\partial net_{pj}(n)} \frac{\partial net_{pj}(n)}{\partial w_{ij}(n-1,n)} .\]

(13)From (11) we see that

\[\frac{\partial net_{pj}(n)}{\partial w_{ij}(n-1,n)} = o_{pi}(n-1) .\]

(14)

If we define

\[\epsilon_{pj}(n) = -\frac{\partial E_p}{\partial net_{pj}(n)}\]

(15)
we have a definition that is consistent with (5) since \( o_{pj}(n) = net_{pj}(n) \) when unit \( u_j(n) \) is linear.

Substituting (14) and (15) into (13) we have

\[
\frac{-\partial E}{\partial w_{ij}(n-1,n)} = \delta_{pj}(n) o_{pi}(n-1).
\] (16)

So again we implement a steepest descent rule.

There is a simple recursive way to compute the \( \delta_{pj}(n) \)'s. We have (15) and applying the chain rule again we obtain:

\[
\delta_{pj}(n) = -\frac{\partial E}{\partial o_{pj}(n)} \frac{\partial o_{pj}(n)}{\partial net_{pj}(n)}.
\] (17)

Since

\[
\frac{\partial o_{pj}(n)}{\partial net_{pj}(n)} = f'(net_{pj}(n))
\] (18)

and

\[
\frac{\partial E}{\partial o_{pj}(d)} = -(t_{pj} - o_{pj}(d))
\] (19)

for an output unit, we get

\[
\delta_{pj}(d) = (t_{pj} - o_{pj}(d)) f'(net_{pj}(d))
\] (20)

for any output unit \( u_j(d) \). For any hidden unit \( u_j(n), \ 1 < n < d \), we use the chain rule to write

\[
\sum_{t=1}^{m_n+1} \frac{\partial E}{\partial net_{pt}(n+1)} \frac{\partial net_{pt}(n+1)}{\partial o_{pj}(n)} = \\
\sum_{t=1}^{m_n+1} \frac{\partial E}{\partial net_{pt}(n+1)} \frac{\partial}{\partial o_{pj}(n)} \sum_{i=1}^{m_n+1} w_{it}(n+1) o_{pi}(n) = \\
\sum_{t=1}^{m_n+1} \frac{\partial E}{\partial net_{pt}(n+1)} w_{jt} = -\sum_{t=1}^{m_n+1} \hat{c}_{pl}(n+1) w_{tj}(n+1).
\] (21)

So for any hidden unit we can write
\[
\delta_{pj}(n) = f'(\text{net}_{pj}(n)) \sum_{t=1}^{m+1} \delta_{pt}(n+1) w_{tj}(n+1).
\] (22)

In our case the implementation of a neural network and the applied training we do with it comes down to the following for each track separately:

We try to map a vector \( x, x \in \{0, 1, \ldots, 255\}^{65} \) via a function \( g \) to a vector \( y, y \in [0, 1]^7 \). So \( g : \{0, 1, \ldots, 255\}^{65} \rightarrow [0, 1]^7 \). We can consider \( g \) to be:

\[
g(x) = f_2 \circ B \circ f_1 \circ A \circ s(x)
\]

where

\[
s : \{0, 1, \ldots, 255\}^{65} \rightarrow [0, 1]^{65}, \text{ the first scaling}
\]

A: a 36 \times 65 matrix with \( a_{ij} = w_{ij}(1, 2) \), the weights assigned to the edges between the input layer and the hidden layer.

\[
f_1 : \mathbb{R}^{36} \rightarrow [0, 1]^{36} \text{ the first logistic function}
\]

B: a 7 \times 36 matrix with \( b_{ij} = w_{ij}(2, 3) \), the weights assigned to the edges between the hidden layer and the output layer.

\[
f_2 : \mathbb{R}^7 \rightarrow [0, 1]^7 \text{ the second logistic function.}
\]

It is easy to see that adding a hidden layer offers a wider variety of functions \( g \). In the case where \( d = 2 \) (only an input- and outputlayer)

\[
g(x) = f_3 \circ C \circ s(x)
\]

where

\[
s(x) : \{0, 1, \ldots, 255\}^{65} \rightarrow [0, 1]^{65}, \text{ the first scaling}
\]

C: a 7 \times 65 matrix, with \( C_{ij} = w_{ij}(1, 2) \), the weights assigned to the edges between the input- and outputlayer.

\[
f_3 : \mathbb{R}^{65} \rightarrow [0, 1]^7 \text{ the logistic function.}
\]

In our network with hidden units we can achieve this by taking all weights \( a_{ij} = 0 \)
for \( j = 8, \ldots, 36 \), \( a_{ij} = c_{ij} \) for \( j = 1, \ldots, 7 \) and \( B = I \) the identity-matrix.

Although we apply an extra function \( f_2 \) on top of it, this will not change the order of the elements, what is important when we want to make a decision to what dollar type a specific note belongs.

In practice however this is not what will happen. The network will be trained so that

\[
B \circ f_1 \circ A \cong C.
\]
3. THE ALGORITHM

The algorithm for recognizing the dollar notes is a program called "check". The program needs the files "weights.trx" for \( x = 1, 2, \ldots, 9 \). The program expects input from the notes in the files "dollar.spx" for \( x = 1, 2, \ldots, 9 \). It is possible to see the neural network as a large function which calculates a value for each of the possible answers \{1, 2, \ldots, 7\}, these answers are in the interval \([0, 1]\). The note with the highest value (if this value is at least 0.555) is the number of the recognized note. Below are printed for each kind of dollar an example of these function values for which that note is recognized.

The program will print as output:

".... dollar" (with on the dots the value of the recognized note)

or

"not recognized".

On the next page are examples of possible function values:
<table>
<thead>
<tr>
<th>note</th>
<th>functionvalue</th>
<th>note</th>
<th>functionvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.80</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>2</td>
<td>0.03</td>
<td>2</td>
<td>8.93</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>3</td>
<td>0.06</td>
</tr>
<tr>
<td>4</td>
<td>0.02</td>
<td>4</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>0.56</td>
<td>5</td>
<td>0.20</td>
</tr>
<tr>
<td>6</td>
<td>0.16</td>
<td>6</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>0.21</td>
<td>7</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td>note 1 output: “1 dollar”</td>
<td>note 2 output: “2 dollar”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>note</th>
<th>functionvalue</th>
<th>note</th>
<th>functionvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>1</td>
<td>0.13</td>
</tr>
<tr>
<td>2</td>
<td>0.02</td>
<td>2</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>8.08</td>
<td>3</td>
<td>1.11</td>
</tr>
<tr>
<td>4</td>
<td>0.20</td>
<td>4</td>
<td>8.11</td>
</tr>
<tr>
<td>5</td>
<td>0.06</td>
<td>5</td>
<td>0.04</td>
</tr>
<tr>
<td>6</td>
<td>0.99</td>
<td>6</td>
<td>0.35</td>
</tr>
<tr>
<td>7</td>
<td>1.26</td>
<td>7</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>note 3 output: “5 dollar”</td>
<td>note 4 output: “10 dollar”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>note</th>
<th>functionvalue</th>
<th>note</th>
<th>functionvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.02</td>
<td>1</td>
<td>0.32</td>
</tr>
<tr>
<td>2</td>
<td>0.03</td>
<td>2</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>0.15</td>
<td>3</td>
<td>0.11</td>
</tr>
<tr>
<td>4</td>
<td>0.16</td>
<td>4</td>
<td>0.72</td>
</tr>
<tr>
<td>5</td>
<td>7.95</td>
<td>5</td>
<td>0.16</td>
</tr>
<tr>
<td>6</td>
<td>0.36</td>
<td>6</td>
<td>8.92</td>
</tr>
<tr>
<td>7</td>
<td>0.78</td>
<td>7</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>note 5 output: “20 dollar”</td>
<td>note 6 output: “50 dollar”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>note</th>
<th>functionvalue</th>
<th>note</th>
<th>functionvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.02</td>
<td>1</td>
<td>0.04</td>
</tr>
<tr>
<td>2</td>
<td>0.04</td>
<td>2</td>
<td>0.50</td>
</tr>
<tr>
<td>3</td>
<td>0.01</td>
<td>3</td>
<td>0.01</td>
</tr>
<tr>
<td>4</td>
<td>0.02</td>
<td>4</td>
<td>0.02</td>
</tr>
<tr>
<td>5</td>
<td>8.86</td>
<td>5</td>
<td>8.86</td>
</tr>
<tr>
<td></td>
<td>note 7 output: “100 dollar”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. CONCLUSIONS AND REMARKS

At the end of this report we will briefly summarise the results and place some remarks.

We basically have tried two methods to solve the problem of dollar recognition. With both methods we have tried to reach a reliability percentage of 99.9%. The concept of neural networks proved to be considerably better than the concept of taking average patterns. With the former we could reach a percentage of 100.00% whereas the latter only could reach 90%.

The time needed to recognize a dollar note with the neural network algorithm is approximately 5 seconds which is acceptable. A client is willing to wait this time.

The training of a neural network comes down to implementing a steepest descent method. Although we cannot prove that we will always find a global optimum, in practice this seems to work very well.

There is a large variety in constructing a neural network. The various types differ in computational effort and reliability. One should always try to find a balance between these two.

We did not consider larger neural networks, because we could reach our goal with only one layer of hidden units. Nevertheless it might be worth the effort to try to construct multi-layered networks to solve the problem. The problem is that one looses sight of what really happens in the network and that the computational effort becomes greater.
A. NEURAL NETWORK: TRAINING, RESULTS AND REMARKS

A.1. Training of the neural network

The training of the network existed in passing all the notes of a trial set in a random order through the network and changing the weights after each note as described. These weights get a random chosen start value on which the earlier described changing method does not depend. After all the notes of the trial set were passed the total error was calculated and the learning rate could be changed. Then the cycle was repeated, until the total error did not change significantly any more. The time needed for the learning process for one track was about 24 hours.

A.2. Numerical results

Before we will give results we will say something about the network. The network reads the information, does some calculation and gives each of the function values a value between 0 and 1. This will be done for each track so that the total network gets 9 results which hopefully will be the same. We have made a test which checked the amount of notes recognized well on each track. If it is possible to recognize on each track almost all the notes then it is possible to make a test which recognizes all the notes by combining the results from all the tracks. Therefore it is necessary that there are no notes which are not recognized well on most of the tracks. This last remark gives no problem with the notes we have. Another check was the value of the error function on each track. The value of the error function was minimized during the learning. The smaller the value of the error function the better the notes are recognized. This means that the "recognized" note gets a value near to 1 and the other notes gets a value near to 0. For the training we used 280 notes. So by the test we had 280 notes which were used by the training and 280 notes which were new for the network. The 280 notes which are new for the network are important for the test. Because with these notes it is possible to see what the network does with notes which are different from the notes used by the training. We have data of 7 kinds of dollar notes. From each kind of dollar note we have 20 examples of each orientation, 10 new notes and 10 old notes. For the training we used half of the notes, so 5 old and 5 new examples of each orientation of each kind of a dollar. For a hidden layer of length 36 we have the following results:
The tracks 8 and 9 are trained longer to get a higher amount of well recognized notes. This has resulted in a small value of the error function but still 7 or 8 misses.

Finally we have trained the network with all the 560 notes. This is done to get weights which are based on all the notes. This has led to the following results:

From these results you can see that almost all the tracks (except track 1, 8 and 9) can recognize all the notes. So the network works very well. Now we can construct a test to combine the 9 results to find the right note. For this test there are a few options.

- Take the note which is recognized on most tracks.
- Take the note which has the highest value if you sum the answers of all the tracks.
- A combination of these 2.

In both options you can check for fake notes. By the first option you can check if the note is recognized on at least 5 tracks. By the second option you can check if the note is recognized on at least 5 tracks. By the second option you can check if the value of the recognized is at least 5. Another check is to look at each track at the difference between the highest value and the second highest value. If this difference is too small you do not count the result of this track. All these checks can be made as sharp as you want. The sharper the check the more good recognized notes will not be accepted and the less fake notes will be accepted. We check with the second option if the value
of the recognized note is at least 5.

A.3. Remarks

The learning process which we used may not be optimal. By using a better strategy for the value of the learning rate the time can decrease to about 6 hours. Also the criterium for stopping the training can be more properly chosen. If you train a network too long on one trial set, you learn the network not only to recognize the notes but also to recognize the noise on the data of the trial set. This can be prevented as follows: use a trial set and a "control" set. After each training run you check the "control" set. During the first runs the error in the trial set and the error in the "control" set will decrease. But if you train too long the error in the trial set will decrease but the error in the "control" set will increase. At this time you train the network to recognize the noise of the trial set. So the optimal point to stop the training is when the error in the "control" set starts to increase. The recognizing of dollar notes on the basis of only one track works already pretty well. So it is possible to change the check so that you use only a few tracks in stead of nine tracks. In this way the time needed for the check decreases and the memory needed to store all the weights also decreases, but you may lose some of the accuracy.
B. AVERAGES METHOD

In this appendix we describe an algorithm to determine the type of a dollar note given the measured pattern of the note. This algorithm is based on average patterns and determines that type of dollar note for which the average pattern is closest to the measured pattern.

In the first part we describe what average patterns are used, in the second part we describe how the type of a dollar note is determined, in the third part some numerical results are stated and in the last part conclusions are given.

B.1. Average patterns

As stated in the introduction, measurements are available of 10 new notes and 10 old notes for each type of note. Per note, measurements are available for each of its four possible orientations. A set of measurements of a certain note in a certain orientation is divided in 9 tracks, each track containing 80 measurements. The measurements of each of the first four tracks correspond to one kind of light test, while the measurements of each of the last five tracks correspond to two kinds of light tests, each kind 40 measurements. So, in total we have 14 different light tests.

We define \( n_{d,i,o,t} \), \( n_{d,i,o,t} \in \{0,1,\ldots,255\}^{80} \), as the vector of 80 measurements corresponding to track \( t \) of the \( i^{th} \) note of type \( d \) in orientation \( o \). Here, \( i \in \{1,2,\ldots,20\} \), \( i \in \{1,2,\ldots,10\} \) corresponds to a new note, \( i \in \{11,12,\ldots,20\} \) corresponds to an old note.

For each type of note \( d \) and each orientation \( o \) and each track \( t \), \( t \in \{1,2,\ldots,9\} \), we define \( \bar{n}_{d,o,t} \) as the average pattern taken over the patterns of the 10 new notes of type \( d \), orientation \( o \) and track \( t \):

\[
\bar{n}_{d,o,t}(j) := \frac{1}{10} \sum_{i=1}^{10} n_{d,i,o,t}(j) \quad j \in \{1,2,\ldots,80\}
\]

Obviously \( \bar{n}_{d,o,t} \in [0,255]^{80} \).

The way these averages are used to determine the type of a dollar note is described in the next part.

B.2. Type determination

Note that we have calculated average patterns of the new notes. We use these average patterns to determine the type of new notes by comparing, track by track, the pattern with the 28 average patterns (7 types of notes, 4 orientations). If our algorithm does not succeed to recognize new notes as required, we can not expect that the analogous algorithm for old notes recognizes old notes as required, so we initially restrict ourselves
to the new notes. Mark that if the “new note” and “old note” algorithm recognize new and old notes as required, respectively, we have not yet an algorithm that recognizes a note which is not known to be new or old.

In words we describe the algorithm to determine the type of a new dollar note as follows. Firstly we determine for each pattern corresponding to one of the 14 tests, the type of dollar and orientation for which the average pattern has minimal distance to the concerning pattern. Each of the 14 tests yields a type of dollar as result. Secondly we determine the type \( d^* \) which is the most occurring result among the 14 results. This type \( d^* \) is our prediction of the type of this new dollar note if \( d^* \) is the only most occurring result.

More specific, we describe the algorithm as follows.

The type of a new dollar note with a pattern represented by \( n_t, n_t \in \{0, 1, \ldots, 255\}^{80}, t \in \{1, 2, \ldots, 9\} \), where \( n_t \) denotes the 80 measurements corresponding to track \( t \), is determined in two steps:

1. \( \bullet \) for \( t \in \{1, 2, 3, 4\} \):
   
   (a) calculate
   \[
   \Delta_{d, o, t} := \sum_{j=1}^{80} \left| n_t(j) - \bar{n}_{d, o, t}(j) \right|^\alpha
   \]
   for all \( d \) and for all \( o \) (28 combinations)
   
   (b) determine \( d_t \) so that
   \[
   \Delta_{d_{t, o, t}} = \min \{ \Delta_{d, o, t} \mid d \in \{1, 2, \ldots, 7\}, o \in \{1, 2, \ldots, 4\} \}
   \]

   \( \bullet \) for \( t \in \{5, 6, 7, 8, 9\} \):
   
   (a) calculate
   \[
   \Delta_{d, o, tf} := \sum_{j=1}^{80} \left| n_t(j) - \bar{n}_{d, o, t}(j) \right|^\alpha
   \]
   for all \( d \) and for all \( o \)
   
   (b) determine \( d_{tf} \) so that
   \[
   \Delta_{d_{tf, o, t}} = \min \{ \Delta_{d, o, tf} \mid d \in \{1, 2, \ldots, 7\}, o \in \{1, 2, \ldots, 4\} \}
   \]
   
   (c) calculate
   \[
   \Delta_{d, o, tl} := \sum_{j=41}^{80} \left| n_t(j) - \bar{n}_{d, o, t}(j) \right|^\alpha
   \]
   for all \( d \) and for all \( o \)
   
   (d) determine \( d_{tl} \) so that
   \[
   \Delta_{d_{tl, o, t}} = \min \{ \Delta_{d, o, tl} \mid d \in \{1, 2, \ldots, 7\}, o \in \{1, 2, \ldots, 4\} \}
   \]

   The distance between two patterns is determined by \( \alpha, \alpha > 0 \).

2. Determine \( d^* \), where

\[
| d^* | = \max \{|d| \mid d \in \{1, 2, \ldots, 7\} \}
\]

with

\[
|d| = |\{t \mid t \in \{1, 2, 3, 4\} \land d_t = d\}| +
\]

\[
|\{t \mid t \in \{5, 6, 7, 8, 9\} \land d_{tf} = d\}| +
\]

\[
|\{t \mid t \in \{5, 6, 7, 8, 9\} \land d_{tl} = d\}| +
\]
B.3. Numerical results

Firstly we state results of the first step of the algorithm for each test separately. Secondly we calculate the percentage of the notes that should be recognized well, based on these results and assuming that the 14 tests are independent. Finally we state the results of the second step of the algorithm and compare the performance with the calculated performance based on the assumption that the tests are independent, and give some conclusions.

B.3.1. Results per test

For all 14 tests separately the percentage of the 280 new notes (actually 70 new notes, each in 4 orientations) which type was recognized well according to the test is listed in the following table. Between brackets the percentage of new notes which type and orientation was recognized well, is listed. These percentages are listed for two values of $\alpha$: $\alpha = 1$ and $\alpha = 2$.

<table>
<thead>
<tr>
<th>test</th>
<th>$\alpha = 1$</th>
<th>$\alpha = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>69.93 (55.00)</td>
<td>72.68 (58.57)</td>
</tr>
<tr>
<td>2</td>
<td>71.79 (60.71)</td>
<td>75.00 (67.50)</td>
</tr>
<tr>
<td>3</td>
<td>71.43 (63.57)</td>
<td>73.21 (65.71)</td>
</tr>
<tr>
<td>4</td>
<td>68.21 (57.86)</td>
<td>71.79 (63.93)</td>
</tr>
<tr>
<td>5</td>
<td>75.00 (21.07)</td>
<td>83.57 (25.00)</td>
</tr>
<tr>
<td>6</td>
<td>73.21 (61.79)</td>
<td>81.43 (68.57)</td>
</tr>
<tr>
<td>7</td>
<td>67.50 (24.64)</td>
<td>72.86 (26.07)</td>
</tr>
<tr>
<td>8</td>
<td>82.50 (67.86)</td>
<td>77.86 (68.93)</td>
</tr>
<tr>
<td>9</td>
<td>66.43 (14.29)</td>
<td>66.43 (14.29)</td>
</tr>
<tr>
<td>10</td>
<td>81.79 (71.43)</td>
<td>82.86 (73.21)</td>
</tr>
<tr>
<td>11</td>
<td>62.14 (12.86)</td>
<td>62.50 (13.57)</td>
</tr>
<tr>
<td>12</td>
<td>85.71 (71.07)</td>
<td>86.07 (74.29)</td>
</tr>
<tr>
<td>13</td>
<td>64.29 (14.29)</td>
<td>63.21 (16.43)</td>
</tr>
<tr>
<td>14</td>
<td>76.43 (66.79)</td>
<td>85.00 (75.36)</td>
</tr>
</tbody>
</table>

Table 1: Recognition percentages per test

Obviously, the percentage of good type recognition is always not less than than the percentage of good type/orientation recognition. Note, however, that the percentage of good type/orientation recognition is very small in comparison to the percentage of good type recognition for the odd numbered tests starting from 5. We can not explain this behaviour.

Note that $\alpha = 2$ yields better results than $\alpha = 1$ in almost all tests. The average percentage of good type recognition for $\alpha = 1$ and $\alpha = 2$ is 72.53 and 75.33 respectively.
B.3.2. Theoretical recognition percentage

So, each of the 14 tests with $\alpha = 2$ recognizes approximately 75% of the new notes. If we assume that the tests are independent, i.e. the probability that a new note is recognized by a test does not depend on the result of another test, then we can calculate the percentage of the new notes that should be recognized well. If our prediction of the type of a new dollar note is the most occurring result among the 14 test results, than the percentage of new notes that should be recognized well is approximately 99.99. This meets our requirement to recognize at least 99.9% of the new notes, so our approach looks very promising if our assumption is correct. The correctness of our assumption is tested in the next part, where the results of the second part of the algorithm are stated.

B.3.3. Results of the total algorithm

In the following table, the percentages of well, not, and falsely recognized new notes are listed for $\alpha = 1$ and $\alpha = 2$, for the 7 different types of dollar notes separately as well as the averages over all types. We say that a note is not recognized if there are two or more results which occur equally often.

<table>
<thead>
<tr>
<th>type</th>
<th>$\alpha = 1$</th>
<th>$\alpha = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>good</td>
<td>not</td>
</tr>
<tr>
<td>1</td>
<td>95.00</td>
<td>5.00</td>
</tr>
<tr>
<td>2</td>
<td>85.00</td>
<td>12.50</td>
</tr>
<tr>
<td>3</td>
<td>97.50</td>
<td>2.50</td>
</tr>
<tr>
<td>4</td>
<td>100.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>80.00</td>
<td>7.50</td>
</tr>
<tr>
<td>6</td>
<td>90.00</td>
<td>5.00</td>
</tr>
<tr>
<td>7</td>
<td>40.00</td>
<td>40.00</td>
</tr>
<tr>
<td>ave</td>
<td>83.93</td>
<td>10.36</td>
</tr>
</tbody>
</table>

Table 2: Recognition percentages of the total algorithm

The most important result is that the percentage of new notes that is recognized well, is only 90.00 for $\alpha = 2$. So, the tests are not independent. The percentage does not nearly meet the required recognition percentage, so from this point of view we have to reject this algorithm. We have to reject this algorithm from another point of view also, because it can not determine the type of a new note within reasonable time.
B.4. Conclusions

1. We have described a type recognition algorithm for new notes that

   • determines for each pattern of a new note corresponding to one of the 14
tests, the type of dollar and orientation for which the average pattern (cor-
responding to that test) has minimal distance in some way to the pattern

   • yields as prediction of the type of a new note the most occurring test result
among the 14 test results.

2. The average percentage of type recognition per test is approximately 75 for some
distance.

3. The percentage of type recognition of the total algorithm is 90.00, which does
not nearly meet the required percentage of 99.9.

4. This averages algorithm does not work for new notes, so the analogous algorithm
for old notes will probably not work either.

5. If both algorithms (for new notes, for old notes) would work, we still had to
construct an overall algorithm that recognizes a note which is not known to be
new or old.

6. The algorithm would not nearly meet the restriction on the time available to
determine the type of a note.