Bachelor thesis
Computer Science


x

Radboud University

# Training the CIA model using back propagation

*Author:*
Stijn Voss
stijn.voss@student.ru.nl
s4150511

*First supervisor/assessor:*
MSc. Maya Sappelli
m.sappelli@cs.ru.nl

*Second assessor:*
prof. dr. T.M. (Tom) Heskes
t.heskes@science.ru.nl

February 20, 2015

**Abstract**

The SWELL project aims to improve the well being of knowledge workers. For this purpose it might be useful to identify the context a knowledge worker is working on at a given time. For example this could be used to provide the knowledge worker with an overview of his or her day.

The CIA model is able to recognize and identify the context in which a knowledge worker is working using low level computer events. The CIA model is build without the need for any examples, instead it uses the documents of the knowledge worker to build the model. But in some cases examples are available. In these cases it could be possible to increase the accuracy of the model using supervised learning techniques.

In this thesis a method based on backpropagation is explored that could be used to improve the accuracy of the model.

# Contents

# Chapter 1

# Introduction

The SWELL project aims to improve the well-being of knowledge workers by using intelligent tools. Knowledge workers are very vulnerable to stress (Michie, 2002). To optimize knowledge workers well-being it is useful to know on which project and in what context a knowledge worker spends his or her time. For example, an overview of the time spent on his or her working day can provide feedback to the knowledge worker with which the worker can improve his or her productivity.

Additionally the context could be used as input in other intelligent tools of the SWELL project. For example notifications of emails could be supressed untill the knowledge worker stops working on a specific task. Or documents that are relevant to the current task could automaticly be shown.

When a knowledge worker has to provide information manually about their time spent on different projects this can be very time-consuming. Additionally, manually providing information is more prone to errors. For instance small tasks like responding to an email may be ignored. Thats why Maya Sappelli (n.d.) proposed a method that is able to recognize the context in which someone is working by using computer low level interaction data, which in this case consisted of: mouse clicks, active windows and keystrokes.

The method devised by Sapelli recognizes the context of work by utilization of the CIA model. This model works a lot like a neural network but with a few special properties:

- The model contains recurrent connections
- The model uses a grossberg activation function that calculates the new activation by using the neuron inputs and the current activation of the neuron.
- During classification in most neural networks the label of certain input

is determined by using the label corresponding to the output node with the highest activation. In the CIA model the output node with the highest relative increase is used.

The current network is unsupervised, which means that it does not need event samples to function. Instead it uses the information of documents on the computer of the user to identify the project. This is called transductive transfer learning. An advantage of this approach is that no event samples or feedback are required to use the network.

The current model has an accuracy of 64.85%, meaning that 64.85% of the events where identified correctly. An accuracy of 64.85% is high enough to be useful in practice. We only need rough estimates for our overview and each event only takes a limited amount of time. There is, however, still room for improvement.

In some cases we might have feedback available. For example, when a knowledge worker manually adjusts a project label for a given period of time. When feedback is available it might be possible and useful to get a higher accuracy by applying supervised learning.

In this thesis a method based on backpropagation that is able to train the CIA model is proposed. Backpropagation cannot be applied to recurrent networks directly. Therefore we will use a method called 'unfolding' that enables us to apply back propagation. We will also introduce some small adjustments to the back propagation algorithm to make sure it is possible to use the Grossberg activation function.

In the second chapter the used techniques of neural networks, backpropagation and the CIA model will be explained. In the third chapter the used data set is explained, the adjustments to the back propagation algorithm and the application of the different techniques described in chapter 2. The fourth chapter contains the research results, which will finally be discussed in chapter five.

Our research question is: Can the accuracy of the CIA model be improved using back propagation?

# Chapter 2

# Preliminaries

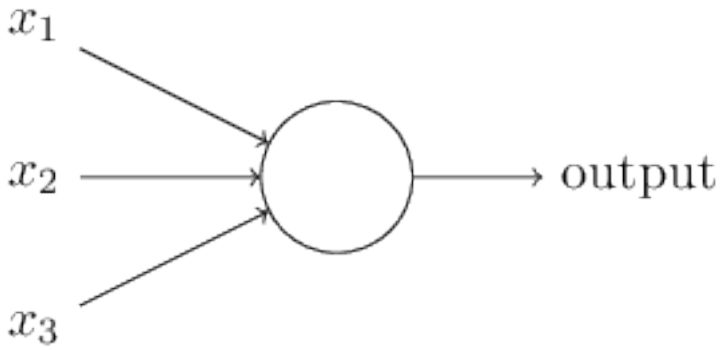
The current method (Maya Sappelli, n.d.) is based on the IA model (McClelland & Rumelhart, 1981) which is a recurrent network. In this thesis a training method for the CIA model based on the backpropagation method is proposed. Backpropagation can only be used for feedforward neural networks. In order to apply back propagation to the CIA model will be unfolded into a feedforward neural network.

These methods will be explained in more detail in this chapter. Firstly it will be explained what a feedforward neural network is, and how it can be trained using the back propagation method. Secondly the details of the CIA model will be described. Once the differences are clear, the unfolding method which is required to apply back propagation to the CIA model will be explained.

## 2.1 Feed forward neural network

A feedforward neural network can be trained using samples to estimate a function. In our case, a function that maps low level computer events to a project label. The basic building block of a feed forward neural network is a neuron.

Figure 2.1: A neuron, Adapted from "Neural networks and deep learning" by M. A. Nielsen, 2015




A neuron has multiple inputs and one output. Each input has a weight. An activation function determines based on the inputs and their weights what the output of an neuron will be. Usually this is a function that uses the sum of all activations of the connected neurons multiplied by the connection weight. A very common function is the sigmoid function: $S(t) = \frac{1}{1+e^{-}t}$. This function has an S shape, meaning that the largest part of the function domain will either have a value relatively close to the function maximum or to the function minimum and the values in between only occur on a very small part of the domain. In this way it is simulated that a neuron fires (close to the maximum value) or does not fire (close to minimum value). Often, a threshold is used to either delay or bring forward the firing of the neuron.

A feed forward neural network consists of at least one layer of these neurons. The output of each neuron in a layer is connected to an input of a neuron in the next layer. The input of the first layer is the input of the whole network. The output of the last layer is the output of the whole network. A one layer neural network is only capable of representing very simple problems and usually uses another training method and activation function, which will not be discussed in more detail in this thesis.

A network consisting of more than one layer contains at least one hidden

layer, which is a layer where the output is hidden and only used inside the network. When we want to predict or classify a certain input we use it as input for the first layer of neurons. The output for this layer is the input for the next layer and so forth until we reach the last layer. The output of the last layer is the output of the whole network(meaning the prediction or classification).

We will now introduce some basic math (Nielsen, 2015) to describe a neural network which will become handy when we explain back propagation.

$w^l_{jk}$ denotes the weight for the connection from the kth neuron in the $(l-1)^{th}$ layer.
$b^l_j$ denotes the bias for the $j^{th}$ node in the $l^{th}$ layer.
$a^l_j$ denotes the activation for the $j^{th}$ node in the $l^{th}$ layer
$z^l_j$ denotes the input of the activation function for the $j^{th}$ node in the $l^{th}$ layer
$\sigma$ denotes the activation function

When we want to get a complete vector of a variable for example the input of one layer $l$, we just remove the $j$ parameter: $z^l$.

### 2.1.1 Learning using backpropagation

Learning a function by training a neural network is usually done by changing the weights of the network. To do so the back propagation (Rummelhart, 1986) method is the most common method for training feed forward network. In order to train a neural network, an initial datas et is needed. This data set, also called training set, consists of samples. Each sample is defined by an input vector $\bar{x}$ that gives an input for each input node of the network and an output vector $\bar{y}$ that gives an expected output for each output node of the network. It is assumed that the data set is representive for the problem as a whole.

For each sample the output can be predicted using the current weights in the network. Based on the expected output of the sample an error- or cost function can be defined. The main property of such a function is that the closer the actual output is to the expected output, the lower the output of the cost function. The training problem can then be defined as a minimization problem of the cost function for the complete training set.

Minimizing the cost function is done by using the gradient descent method. The gradient descent method says that in order to minimize function F from a point $a$ we should move to point $a - F'(a)$. In terms of our neural network this means that when we want to minimize the cost function C by altering

weight $w^l_{jk}$ we should update $w^l_{jk} = w^l_{jk} - \frac{\partial C}{\partial w^l_{jk}}$ In other words we should update a weight in the opposite direction to the contribution it had to the cost function.

It's very common to add a learning parameter $\xi$ to the gradient descent formula. $w^l_{jk} = w^l_{jk} - \xi \cdot \frac{\partial C}{\partial w^l_{jk}}$ . A higher learning rate means that it will learn faster but it is easier to miss the minimum of the error function. A lower learning rate will cause the network to learn slower but will be more likely to find the minimum.

In order to calculate $\frac{\partial C}{\partial w^l_{jk}}$ we will first introduce an extra variable that is propagated back over the network.

$$\delta^l_j = \frac{\partial C}{\partial z^l_j}$$

Intuitively, one could see this as the fraction of the error the input of the neuron j in layer l is responsible for.

The error of the output layer can be calculated using:

$$\delta^L = \Delta_a C \odot \sigma'(z^L)$$

Here $\Delta_a C$ means the vector with the $\frac{\partial C}{\partial a^L_j}$for each output node. This is the derivative of the cost function.

With $\odot$ we mean the hadamard product or element-wise multiplication. Here we take two equals sized matrixes and get a new matrix with the same size. Each element value of this new matrix is the product of the two values of original matrix values corresponding to the same element. In this case the error of each output node is multiplied with the delta of its activation.

The error of the previous layers can be calculated using:

$$\delta^l = ((w^{l+1})^T \partial^{l+1}) \odot \sigma'(z^l)$$

As you can see the error of the previous layers depends on the error of the next layer. Therefore the error is propagated back in the network when calculating the new weights. This is why the method is called the back propagation algorithm. Based on the errors it is easy to calculate the change of bias, namely:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

The change of the weight can be calculated using:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$

## 2.2 CIA model

The method used by Maya Sappelli (n.d.) is an adapted version of the IA model McClelland and Rumelhart (1981) especially developed for recognizing context. It works like a feedforward network but connections can be cyclic and the connection of each layer could be connected to multiple other layers. The network is build using the documents found on the users computer including the weights. Each node has a minimun activation and a maximum activation, these values are user-defined parameters. In the CIA model the nodes are divided into multiple layers.

Each layer represents a type of real world objects.

- **Input layer** This layer has a node for each possible event block.
- **Context information layer** This layer contains nodes for different types of context information: terms or topics, entities, (file) location and date/time information.
- **Document layer** Contains nodes for all documents used by the user including e-mails and webpages.
- **Context identification layer** These are the nodes that represent a label that identifies the context.

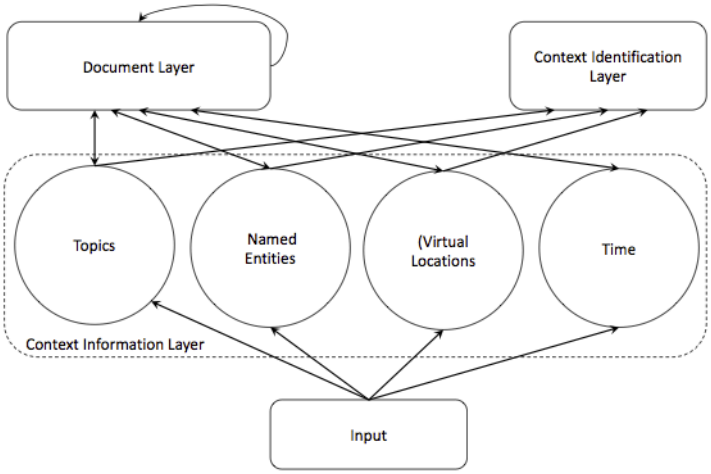
Figure 2.2. shows these different layers and how the are connected.

### 2.2.1 Building the CIA network

Firstly, all the documents of a user are gathered. For each document a node is created in the document layer. Additionally, the context information from each document is extracted. The document location creates a location node and using the Stanford entity recognizer (Finkel, Grenager, & Manning, 2005) the entities are extracted. Topics are determined using the Latent Dirichlet Allocation (McCallum, 2002) . All the connections between a document node and the context information nodes are bidirectional. However the weight of the connection varies depending on the direction of the connection.

The label identification nodes are connected to the context information layer. To decide which labels one could use to identify context and how it should

Figure 2.2: The CIA model layers. Adapted from Maya Sappelli, W. K., Suzan Verberne. A network-based model for working with context. n.d.




be connected to the context identification layer some form of user input is required. One proposed method is to use documents organised in folders where each folder name is a label and the documents are used to create the connections between the context information layer.

### 2.2.2 Activating the network

The event blocks are loaded into the network. The algorithm determines which context information nodes are related to which event nodes. To determine the labels of the events we start by the first event. We activate the corresponding input node, meaning we set the activation to 1.0. The activation is calculated multiple times for each node. The number of interations is a parameter of the algorithm. During the iterations the input node is kept at an activation of 1.0. Once the calculation is done the identification node that has the highest increase in activation compares to its baseline is selected. The network works in such a way that the activation of the previous event influences the activation of the next event block. This is done by activating the next event node while keeping the rest of the activations the same. The old activation then slowely fades away using the decay parameter.

### 2.2.3 Grossberg's activation function

The activation function used in the network is the Grossberg's activation function. It uses the sum of the positive inputs (excitatory) and the sum of the negative inputs (inhibitory). Grossberg's activation function uses the current activation a, the inhibitory input: in, the excitatory input: ex as

input. And uses a couple of parameters. The max activation value: max, the lowest possible activation: min, the speed in which a node loses its activation if no input is received: decay and the rest value when a node is not activated: rest. The activation is then calculated using:

$$MAX(min, MIN(max, a + (max - a) \cdot ex + (a - min) \cdot in - decay(a - rest)))$$

We can calculate the excitatory and inhibitory using the following formula's. When we want to calculate the excitatory and inhibitory input of node $a_j^l$ we sum as follows:

$$ex = \sum_{k \in K} \text{if } a_k^{l-1} > 0 : \alpha \cdot w_{jk}^a \cdot a_k^{l-1} \text{ else} : 0$$
$$in = \sum_{k \in K} \text{if } a_k^{l-1} < 0 : \alpha \cdot w_{jk}^a \cdot a_k^{l-1} \text{ else} : 0$$

Note that with MIN and MAX we mean the functions MIN and MAX that select the lowest and highest arguments respectively. And with lowercase min and max we mean the parameters of the network. $\alpha$ and $\gamma$ are user defined parameters.

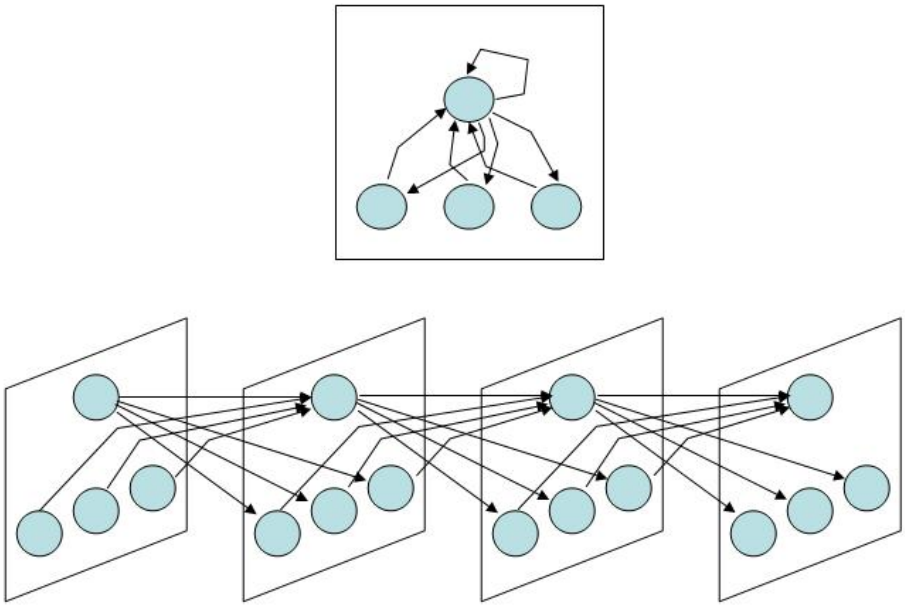
## 2.3 Unfold network to time

Normally the backward propagation algorithm is designed to train feed forward networks. However the CIA model is not a feed forward neural network, at each iteration the activation of all the layers are determined and the network contains cyclic links. The network therefore has to be unfolded to a feedforward neural network (Rogers et al., 2004). This is referred to as the unfolded network. To achieve this a property, that it runs for a limited number of iterations, of the CIA model is used.

In this unfolded network each layer contains all the nodes of the original network. Each layer represents one iteration. This is why the activation of the nodes in the input layer is the same as the activation of the CIA model before the first iteration. The activations of the nodes in the second layer correspond with activation of the CIA network after the first iteration. The connection are mapped in a very simple way. If the CIA model contains a connection from node a to b then and only then there will be a connection in the unfolded network from node a in layer l to b in layer l + 1. Figure 2.3 illustrates the working of this unfolding mechanism.

It is possible to calculate the delta weight of the now unfolded network using the back propagation network as one would do for a feedforward neural network. All the connections of the CIA model will be mapped to multiple connections in the unfolded network for each iteration. Because the back-propagation method is applied to the unfolded network, now it is possible to find delta weight values for each of the connections. However, in the CIA model there is only one connection, so to use all these delta values, they will have to be combined in some way. Rogers et al. (2004) has determined that a good way to do this is to simply sum the delta values. Therefore all the delta weight values of the connections in the CIA model are summed into one number. Now there is a delta weight value for each connection of the CIA model, which can be used in the gradient descent step.

Figure 2.3: Unfolding a recurrent network adapted from "Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises" by James L. McClelland, 2014.



# Chapter 3

# Method

## 3.1 Data set and training

The data set used to train and test the network is the SWELL-KW data set (Koldijk, 2014). It contains the data of 25 participants, each participating for roughly 3 hours. The participants performed typical knowledge work: writing reports, makings presentations, reading and performing research during an experiment. The data set contains three different working conditions: email interruptions, time pressure and neutral conditions

The data set contains different types of features: computer interaction, facial expressions, body postures and physiology.In this thesis only the computer interaction data is used. The data consists of the feautres: mouse position, words typed, window titles and application names. This data is collected using the keylogger ulog and IEhistory.

The same data set is used by (Maya Sappelli, n.d.) for training and testing the CIA model. There were 8 different topics the partcipants could be working on. The events were labeled with these topics using amazon mechanical turk. Some events could not be identified. These received the 9th label: 'unidentified'.

The unidentified events will be discarded in the training or test set since it is impossible to calculate error or determine accuracy based on these events. For all the events that have a label other than 'unidentified' the events that belong to each label are determined. For each label 70% of the events are being put into the training set and 30% are being put into the test set.

Since the network uses information of previous events to determine the activation of the next event, all the events (including the ones labelled 'unidenti-

fied') will have to be activated in the same order to train or test the network. During training the error is only propagated back if the activated event is part of the training set (and thus isn't labelled as unidentified). During the test phase the accuracy is calculated as follows:

$$\frac{\text{\# events that are part of the test set and correctly identified}}{\text{\# events that are part of the test set}}$$

The accuracy of the test set on the network will be determined twice: once before the network has been trained and once after the training has taken place. Both are determined by running the original cia model.

During training for each epoch, track is also being kept of the average error and accuracy on the training set. The accuracy is calculated in the same way as the accuracy was calculated on the test set:

$$\frac{\text{\# events that are part of the train set and correctly identified}}{\text{\# events that are part of the train set}}$$

## 3.2 Activation function

One property of the backpropagation algorithm is that we need a derivative of the activation function to calculate the deltaweight values. In case of the grossberg activation network we have a function with 3 arguments:

$$\begin{aligned}\alpha(a, in, ex) \quad = \quad & MAX(min, MIN(max, a + (max - a) * ex + (a - min) * \\ & in - decay(a - rest)))\end{aligned}$$

Where **a** is the old activation, **in** is the inhibitory input and **ex** the excitatory input. Our network has two types of input connections, namely inhibitory and excitatory input connections. Two derivatives are therefore needed. First:

$$\frac{\partial \alpha}{\partial in} = a - min$$

$$\frac{\partial \alpha}{\partial ex} = max - a$$

Where **a** is the current activation or in the case of the unfolded network the activation of the same node in the previous layer and **min** and **max** are parameters of the network.

## 3.3 Cost function

The cost function is difficult to define since the node with the highest increase in activation is selected instead of the node with the highest absolute activation. First the Mean Sqaured Error(MSE) is used. It is assumed that the ideal situation would be that the activation of an output node that is not the output node belonging to the label of this event is equal to the parameter *min*. And the ideal activation of the output node that corresponds with the label of this event is assumed to be equals to the parameter *max*.

If we have a vector $a$ which contains the activation of each output node for event x and have a vector y that gives the expected output for each output node(thus max for the expected node and min for all the other output nodes). Then the error for event x can be written as:

$$E_x = \frac{1}{2} \cdot \|y - a\|^2$$

The avarage error of the complete training set then looks like:

$$E = \frac{1}{|N|} * \sum_{n \in N} \sum_{x \in X} E_x^n$$

Where N are the training samples and thus $|N|$ the number of training events. X denotes the set of all output nodes and $E_x^n$ denotes the error of output node x in training set n.

The derivative of the error is:

$$\frac{\partial E}{\partial a} = a - y$$

So the delta error of the identification node that belongs to the event label will be its activation minus the *max* parameter. And the rest of the identification nodes have a delta error of their activation minus the *min* parameter. Note that the last layer of the unfolded network contains more nodes than just the identification nodes. However the only nodes that have a target activation are the identification nodes. The error for all the nodes in the last layer that are not identification nodes will be 0, since it doesn't matter what the activations are as long as the identification nodes have the right value. Since the error is 0 in the non identification nodes, the derivative will also be 0.

## 3.4 Feeding forward and propagating back

When we predict a label for an event the corresponding input node is activated, meaning we set the activation to 1.0. During all the iterations to

calculate the output of the network the activation is kept at 1.0, so the decay of this node is ignored. The number of iterations is a parameter. In this case it is set to 10. The activation step is exactly the same as used by (Maya Sappelli, n.d.)

Since the Grossberg activation function uses two types of input, inhibitory and excitatory, track is also being kept of both types separately during the feedforward process. The input is then calculated using the following formulas:

$$ex = \sum_{k \in K} \text{if } a_k^{l-1} > 0 : \alpha \cdot w_{jk}^a \cdot a_k^{l-1} \text{ else} : 0$$
$$in = \sum_{k \in K} \text{if } a_k^{l-1} < 0 : \gamma \cdot w_{jk}^a \cdot a_k^{l-1} \text{ else} : 0$$

More information about activation can be found in section 2.2.3

Since there are two types of inputs there are also two types of errors when propagating back. Namely, the error that is created via the inhibitory inputs and the error that is created via the excitatory input. When calculating the error of a node there are two possible cases. If the node has a positive activation, we sum the excitatory error of the output connected nodes into a number. If the node has a negative activation we sum the inhibitatory errors of the output connected nodes into a number. This can be explained in the following way: if a node has a positive activation it will only contribute to the excitatory error of the connected nodes. Additionaly if a node has a negative activation it will only contribute to the inhibitory error of the connected nodes.

Then we want to propagate the number back over the node to get the excitatory and inhibitory error of the node. The excitatory error of the node is dertermined by multiplying the number with the delta activation with respect to the excitatory input. The inhibitory error of the node is dertermined by multiplying the number with the delta activation with respect to the inhibitory input. In formula form:

$$\partial e_j^l = \frac{\partial \alpha}{\partial z e^l} \cdot e_j^l$$
$$\partial i_j^l = \frac{\partial \alpha}{\partial z i^l} \cdot e_j^l$$

Here

$$
\begin{aligned}
&\text{if } a_j^l \geq 0: \\
&e_j^l = \sum_{n \in J} w_{jn}^l \cdot \delta e_n^{l+1} \\
&\text{otherwise:} \\
&e_j^l = \sum_{n \in J} w_{jn}^l \cdot \delta i_n^{l+1}
\end{aligned}
$$

$\delta e_j^l$ describes the excitatory error of node l in layer j, $\delta i_j^l$ describes the inhibitory error of node l in layer j. $e_j^l$ is a temporary variable that we use. It describes the error after the node j in layer l. $zi_j^l$ describes the inhibitory input of node j in layer l. And $ze_j^l$ describes excitatory input of node j in layer l

Once the errors for all nodes have been calculated, the delta weights of all the connections can be calculated. This is done by multiplying the error of the next node with the activation of the previous node. If a connection starts at a node with a positive activation the excitatory error is used.

In a feedforward neural network the weights of connections from nodes that are not activated are not altered. Since the delta weights are calculated using $\frac{\delta C}{\delta w_{jk}^l} = a_k^{l-1} \delta_j^l$
and the activation would be 0. In the case of the CIA model the minimum activation is set to -0.2. Therefore the weights of the connections connected to nodes that are not activated are still altered. It was found that the network performed better when the connections that start with a node with a negative activation are not altered. Therefore in our case these deltaweights are set to 0.0.

The delta weights for each connection are derived from the errors as follows:

$$
\begin{aligned}
&\text{if } a_k^{l-1} < 0: \\
&\frac{\delta C}{\delta w_{jk}^l} = 0 \\
&\text{else:} \\
&\frac{\delta C}{\delta w_{jk}^l} = a_k^{l-1} \cdot \delta e_j^l
\end{aligned}
$$

Since we are using an unfolded network the deltaweight of the connections in multiple layers representing one connection in the orginal layer are summed to find the deltaweight for the original network (as described in section 2.3).

To prevent local-minima problems all the delta weights of all training events during one training epoch are summed and the deltaweights are substracted

from the connections once the epoch is done. This method is called "batching". In the unfolded network the same connections between the different layers always have the same weight. When adjusting the weights of the unfolded network using gradient descent the same delta weights are used for each connection used in the CIA model.

## 3.5 Experiment

The complete process follows the following steps:

```
for pp in participants:
        build_ciamodel(pp)
        split_train_and_test_set(pp)
        determine_accuracy_using_test_set(pp)
        reset_ciamodel(pp)
        build_unfolded_network(pp)
        for epoch in epochs:
                calc_delta_weight(epoch)
                update_weights_ciamodel(learning_rate)
                update_weights_unfolded_network(learning_rate)
        determine_accuracy_using_test_set(pp)
```

### 3.5.1 Parameters

We used the same parameters values as the values used in the original CIA model paper (Maya Sappelli, n.d.) namely:

| | | | |
|---|---|---|---|
| **min** | -0.2 | **alpha** | 0.1 |
| **max** | 1.0 | **gamma** | 0.1 |
| **rest** | -0.1 | **decay** | 0.1 |

The CIA model supports a few of topic extraction algorithms from which LDA gets the best results. However LDA (Latent Dirichlet Allocation) is not deterministic. We prefer that the results are deterministic so we can meassure the differences in performances without uncertainty. For this purpose, the latent semantic analysis (Deerwester, Dumais, Landauer, Furnas, & Harshman, 1990) is used to retrieve the topics.

# Chapter 4

# Results

We have optimized the network by trying different learning rates and epochs numbers and use the performance as an indicator. We found that training the network with a learning rate of 0.00075 and 10 epochs for each participant was a good configuration. We also used batching, which means we summed all the deltaweights of all training events during one training cycle and after the epoch substracted it from the network weights. We described the batching process in chapter 3. Based on this configuration the behaviour on the training set and the test set will be shown in this chapter. In the last section some of the results of other configurations are also shown.

## 4.1 Behaviour on training set

First, the behaviour on the training set is shown. Table 4.1 shows the accuracy on the training set and the change of the error for each participant. It was found that in almost all cases the error decreased slightly except for participant 25. The accuracy on the training set for participant 3, 9, 10, 14, 20, 22 and 24 went down while the error also went down. All other participants showed a decrease of the error while showing an improvement of the performance.

In figure 4.1 the error and performance of participant 1 for each training epoch is plotted. Participant 1 is an example of an participant where the error went down and the performance went up. We see that the error went down very smoothly and the performance is went up in the opposite direction.

In figure 4.2 the error and performance of participant 14 for each training epoch is plotted. Participant 14 is an example of an participant where the error went down while the performance also went down. We see that the error went down in a straight line but the decrease of the performance also slowed down.

In figure 4.3 the error and performance of participant 25 for each training epoch is plotted. Participant 25 is the only participant where the error went up. We see that the performance jumps back and forth while the error slowly increases.

| **pp** | **first accuracy** | **last accuracy** | **error diff** |
|---|---|---|---|
| 1 | 0.630 | 0.760 | -0.00134 |
| 2 | 0.377 | 0.410 | -0.00012 |
| *3* | *0.577* | *0.440* | *-0.00030* |
| 4 | 0.207 | 0.241 | -0.00114 |
| 5 | 0.280 | 0.360 | -0.00005 |
| 6 | 0.227 | 0.302 | -0.00012 |
| 7 | 0.439 | 0.485 | -0.00038 |
| 8 | 0.464 | 0.470 | -0.00017 |
| *9* | *0.358* | *0.317* | *-0.00020* |
| *10* | *0.515* | *0.491* | *-0.00022* |
| 11 | 0.239 | 0.310 | -0.00037 |
| 12 | 0.308 | 0.321 | -0.00012 |
| 13 | 0.425 | 0.467 | -0.00056 |
| *14* | *0.560* | *0.432* | *-0.00021* |
| 15 | 0.311 | 0.419 | -0.00028 |
| 16 | 0.429 | 0.476 | -0.00042 |
| 17 | 0.504 | 0.556 | -0.00017 |
| 18 | 0.184 | 0.194 | -0.00011 |
| 19 | 0.000 | 0.027 | -0.00040 |
| *20* | *0.657* | *0.624* | *-0.00020* |
| 21 | 0.624 | 0.682 | -0.00027 |
| *22* | *0.578* | *0.374* | *-0.00005* |
| 23 | 0.482 | 0.491 | -0.00017 |
| *24* | *0.557* | *0.542* | *-0.00016* |
| 25 | 0.624 | 0.624 | 0.00003 |

Table 4.1: Showing the accuracy of the first training epoch, accuracy of the last training epoch and the difference in error for each participant. Accuracy is obtained by diving the number of training event correctly identified divided by the number of training events.

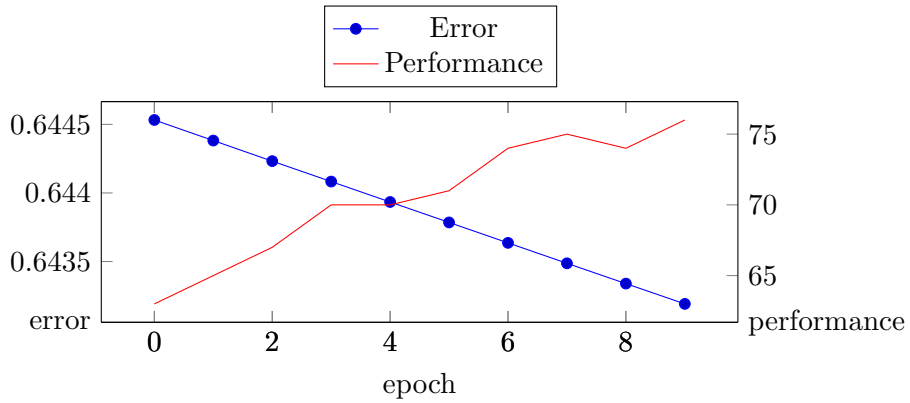



Figure 4.1: Error and performance of participant 1 during training. Performance is defined as the number of training events correctly identified.

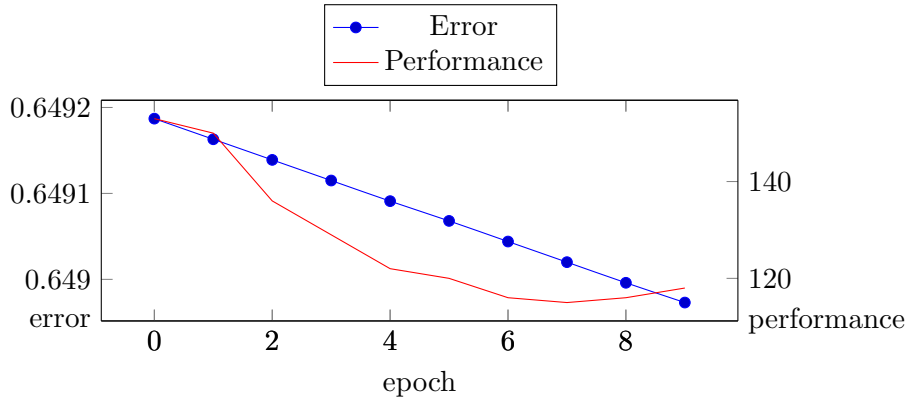



Figure 4.2: Error and performance of participant 14 during training. Performance is defined as the number of training events correctly identified.

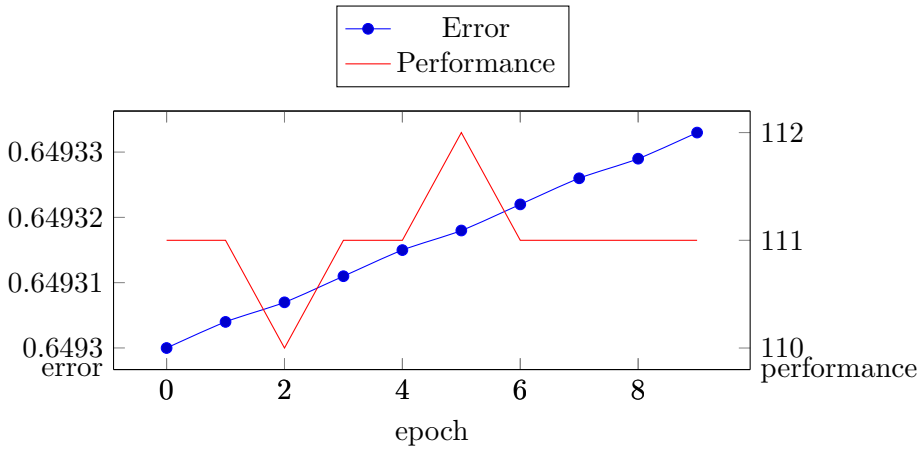



Figure 4.3: Error and performance of participant 25 during training. Performance is defined as the number of training events correctly identified.

## 4.2 Performance on test set

In table 4.2 the accuracy of the test set for each participant before and after training is shown. We see that the accuracy decreased for participants 2, 8 and 14. The performance did not change for the participants 4, 11, 16 and 19. For all other participants the accuracy improved..

| **pp** | **train set size** | **test set size** | **before** | **after** |
|---|---|---|---|---|
| 1 | 100 | 43 | 0.488 | 0.581 |
| 2 | 122 | 52 | 0.288 | 0.269 |
| 3 | 168 | 73 | 0.685 | 0.767 |
| 4 | 58 | 23 | 0.261 | 0.261 |
| 5 | 214 | 93 | 0.344 | 0.355 |
| 6 | 172 | 72 | 0.361 | 0.431 |
| 7 | 66 | 29 | 0.690 | 0.724 |
| 8 | 151 | 65 | 0.554 | 0.538 |
| 9 | 123 | 53 | 0.434 | 0.528 |
| 10 | 171 | 70 | 0.614 | 0.643 |
| 11 | 71 | 31 | 0.581 | 0.581 |
| 12 | 78 | 32 | 0.438 | 0.469 |
| 13 | 120 | 50 | 0.600 | 0.660 |
| 14 | 273 | 117 | 0.607 | 0.538 |
| 15 | 167 | 72 | 0.361 | 0.431 |
| 16 | 84 | 36 | 0.639 | 0.639 |
| 17 | 135 | 57 | 0.632 | 0.649 |
| 18 | 196 | 84 | 0.464 | 0.524 |
| 19 | 37 | 15 | 0.600 | 0.600 |
| 20 | 181 | 76 | 0.500 | 0.539 |
| 21 | 85 | 34 | 0.676 | 0.676 |
| 22 | 211 | 91 | 0.692 | 0.703 |
| 23 | 112 | 48 | 0.479 | 0.500 |
| 24 | 201 | 83 | 0.470 | 0.518 |
| 25 | 178 | 75 | 0.280 | 0.293 |
| **Total** | **3474** | **1474** | | |
| **Average** | | | **.510** | **0.537** |

Table 4.2: Accuracy per participant on the test set before and after training. Accuracy is determined by dividing number of events correctly identified on test set by the number of events in the test set. Averages are obtained by dividing the sum of all accuracies by the number of participants.

## 4.3 Other configurations

Some additional experiments with different learning rates and different numbers of epochs were also run. In the following table 4.3 the different configurations and their performance on the test and training set are shown.

The results show that with a higher learning rate the error decreased faster. However, at the same time the performance on the training set decreased and the accuracy of the test set decreased faster. We also see that with batching enabled the performance on the training set is higher.

Additionally we see that a dividing the learning rate by 2 and increasing the number of epochs with a factor 2 increased the error while improving the error on the test set slightly.

| $\epsilon$ | $\eta$ | **Batching** | **Accuracy** | **Per. diff** | **Error diff** |
|---|---|---|---|---|---|
| 0.0025 | 10 | no | 0.434 | - 88 | -0.000836 |
| 0.001 | 10 | no | 0.498 | - 62 | -0.000449 |
| 0.001 | 10 | yes | 0.501 | - 36 | -0.000433 |
| 0.0005 | 20 | yes | 0.501 | - 34 | -0.000533 |

Table 4.3: The performance of different configurations. $\epsilon$ denotes the learning rate, $\eta$ denotes the number of training epochs, batching denotes if we batched all the delta weight values(as described in chapter 3). Accuracy is obtained by dividing the total number of test events correctly identified after training by the total number of test events. The performance difference denotes the total number of training events correct after the training minus the total number of training events correct before the training. The error difference shows the average error after training minus the average error before training.

# Chapter 5

# Discussion

Training the CIA network with backpropagation improves the accuracy of the context detection slightly. There is an increase from an average accuracy of 51% to 53.7%. Since optimizing a neural network is very time intensive, we have trained all the different networks for each participant with the same configuration. Table 4.2 shows that there are differences in performance gain among te different participants. Since for each participant there is a separately network with a different number of nodes and connections. It is possible that higher performance gains could be achieved when the training configuration is optimized separately for each participant.

The used learning rate and number of epochs might seem low. This is mainly caused by the relatively low differences in activation of the nodes. Only a relatively small activation difference is required to change the identification label. Also when comparing these values with other neural networks it should be considered that the unfolding process sums the delta weight values of all iterations. In our case this makes the learning rate effectively 10 times higher. .

Perhaps the most notable fact of the results is that in many cases the error goes down while the actual performance on the training set also goes down. Table 4.3 indicates that a higher learning rate will also lead to even more performance loss on the training set while decreasing the error. This strongly suggests that the error function isn't performing well. One possible cause might be that we assume that the activation of an identification node should be equal to either the minimum or maximum parameter. However, in a normal feedforward neural network this is not a problem, because the previous input does not influence the result of the next input.

# Chapter 6

# Conclusions

In this thesis a method was proposed that is able to improve the accuracy of the CIA model using back propagation. Since the CIA model is a recurrent network the unfolding technique has been applied. Also the thesis describes a method that is able to handle the two types of inputs for the Grossberg activation function.

The research question was: Can the accuracy of the CIA model be improved using back propagation? We were able to improve the accuracy of the network just a litte from 51% to 53.7% on the SWELL-KW data set. The results strongly suggest that the proposed error function isn't performing well. Since in most cases the error is decreasing steady, while at the same time in many cases the accuracy on the training set is decreasing. We therefore believe that with the right error function the accuracy is very likely to show more improvement. Future research is required to find a better error function that is capable of improving the performance on the training set.

The SWELL-KW data set consists of 25 participants. For each participant a seperate CIA model is build. But in our approach we use the same learning rate and number of training epochs for each participant. It is likely that optimizing the learning rate and number of training epochs for each participant separately will lead to better accuracies. Future reasearch would be required to verify if the accuracy increases when optimizing the networks separately.

# References


Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., & Harshman, R. A. (1990). Indexing by latent semantic analysis. *JASIS*, *41*(6), 391–407.

Finkel, J. R., Grenager, T., & Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics* (pp. 363–370).

Koldijk, S. M. V. S. N. M. . K. W., S. (2014). The swell knowledge work dataset for stress and user modeling research. *Proceedings of the 16th ACM International Conference on Multimodal Interaction*.

Maya Sappelli, W. K., Suzan Verberne. (n.d.). *A network-based model for working with context.*

McCallum, A. K. (2002). *Mallet: A machine learning for language toolkit.* (http://mallet.cs.umass.edu)

McClelland, J. L., & Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: I. an account of basic findings. *Psychological review*, *88*(5), 375.

Michie, S. (2002). Causes and management of stress at work. *Occupational and Environmental Medicine*, *59*(1), 67–72.

Nielsen, M. A. (2015). *Neural networks and deep learning.* Determination Press.

Rogers, T. T., Lambon Ralph, M. A., Garrard, P., Bozeat, S., McClelland, J. L., Hodges, J. R., & Patterson, K. (2004). Structure and deterioration of semantic memory: a neuropsychological and computational investigation. *Psychological review*, *111*(1), 205.

Rummelhart, D. (1986). Learning representations by back-propagating errors. *Nature*, *323*(9), 533–536.

# Appendix A

# Appendix

| **pp** | **train set size** | **First performance** | **Last performance** | **First error** | **Last error** |
|---|---|---|---|---|---|
| 1 | 100 | 63 | 76 | 0.644532 | 0.643191 |
| 2 | 122 | 46 | 50 | 0.649225 | 0.649102 |
| 3 | 168 | 97 | 74 | 0.648566 | 0.648270 |
| 4 | 58 | 12 | 14 | 0.644988 | 0.643847 |
| 5 | 214 | 60 | 77 | 0.649724 | 0.649642 |
| 6 | 172 | 39 | 52 | 0.649727 | 0.649606 |
| 7 | 66 | 29 | 32 | 0.646295 | 0.645916 |
| 8 | 151 | 70 | 71 | 0.648357 | 0.648182 |
| 9 | 123 | 44 | 39 | 0.649174 | 0.648975 |
| 10 | 171 | 88 | 84 | 0.648751 | 0.648530 |
| 11 | 71 | 17 | 22 | 0.648196 | 0.647825 |
| 12 | 78 | 24 | 25 | 0.649689 | 0.649565 |
| 13 | 120 | 51 | 56 | 0.647172 | 0.646612 |
| 14 | 273 | 153 | 118 | 0.649187 | 0.648973 |
| 15 | 167 | 52 | 70 | 0.648898 | 0.648615 |
| 16 | 84 | 36 | 40 | 0.646080 | 0.645659 |
| 17 | 135 | 68 | 75 | 0.648250 | 0.648082 |
| 18 | 196 | 36 | 38 | 0.650440 | 0.650329 |
| 19 | 37 | 0 | 1 | 0.650693 | 0.650290 |
| 20 | 181 | 119 | 113 | 0.649080 | 0.648881 |
| 21 | 85 | 53 | 58 | 0.646855 | 0.646589 |
| 22 | 211 | 122 | 79 | 0.648654 | 0.648571 |
| 23 | 112 | 54 | 55 | 0.647905 | 0.647735 |
| 24 | 201 | 112 | 109 | 0.649679 | 0.649516 |
| 25 | 178 | 111 | 111 | 0.649300 | 0.649333 |
| **total** | **3474** | **1556** | **1539** | | |
| **Average** | | | | **0.648377** | **0.648073** |

Table A.1: For each participant the train set size, the correctly identified events during the first epoch and the last epoch. And the average error of the first epoch and the last epoch.

| **pp** | **train set size** | **test set size** | **before** | **after** |
|---|---|---|---|---|
| 1 | 100 | 43 | 21 | 25 |
| 2 | 122 | 52 | 15 | 14 |
| 3 | 168 | 73 | 50 | 56 |
| 4 | 58 | 23 | 6 | 6 |
| 5 | 214 | 93 | 32 | 33 |
| 6 | 172 | 72 | 26 | 31 |
| 7 | 66 | 29 | 20 | 21 |
| 8 | 151 | 65 | 36 | 35 |
| 9 | 123 | 53 | 23 | 28 |
| 10 | 171 | 70 | 43 | 45 |
| 11 | 71 | 31 | 18 | 18 |
| 12 | 78 | 32 | 14 | 15 |
| 13 | 120 | 50 | 30 | 33 |
| 14 | 273 | 117 | 71 | 63 |
| 15 | 167 | 72 | 26 | 31 |
| 16 | 84 | 36 | 23 | 23 |
| 17 | 135 | 57 | 36 | 37 |
| 18 | 196 | 84 | 39 | 44 |
| 19 | 37 | 15 | 9 | 9 |
| 20 | 181 | 76 | 38 | 41 |
| 21 | 85 | 34 | 23 | 23 |
| 22 | 211 | 91 | 63 | 64 |
| 23 | 112 | 48 | 23 | 24 |
| 24 | 201 | 83 | 39 | 43 |
| 25 | 178 | 75 | 21 | 22 |
| **Total** | **3474** | **1474** | **745** | **784** |

Table A.2: For each participant the train and test set sizes and the number of events of the test set correctly identified before and after the training.

| $\epsilon$ | $\eta$ | **B** | **tea** | **trpf** | **trpl** | **tref** | **trel** |
|---|---|---|---|---|---|---|---|
| 0.0025 | 10 | no | 676 | 1583 | 1495 | .64833456 | .64749828 |
| 0.001 | 10 | no | 775 | 1574 | 1512 | .64835992 | .64791072 |
| 0.001 | 10 | yes | 780 | 1556 | 1520 | .64837668 | .64794416 |
| 0.0005 | 20 | yes | 780 | 1556 | 1522 | .64837668 | .64784348 |

Table A.3: Performance of different configurations. **Tea** denotes the correct number of events after training. **trpf** and **tref** denotes the number of events of the training set correctly identified and the average error during first epoch. **trpl** and **trel** denotes the number of events of the training set correctlye identified and the average error during last epoch.