DEEP BELIEF NETWORK ON CONTEXT OF HANDWRITTEN WORDS

Bachelorproject

Jonne Engelberts, s2228319, jonneengelberts@gmail.com

Abstract: In this paper feature vectors from images of words were used to test whether these feature vectors could be used to cluster semantically similar words in a Deep Belief Network through unsupervised learning. The feature vectors used in this paper came from the handwriting recognition system Monk. First the dimensionality of the feature vectors were reduced with a Restricted Boltzmann Machine. With a reduction of 99% the size of the vectors went from 4356 features to 50 features, without a big loss in Nearest Neighbor finding words with the same label: 75% correct for the 4356 features and 73% correct for the 50 features. To test the ability of the Deep Belief Network clustering semantically similar words a basic artificial language was created. Every sentence in this language had three words per sentence and belonged to one of the two types of context: people or food. The words in the language were first represented by simple Softmax units, which did need a label of the word, but were able to cluster the two contexts perfectly. Secondly the words were represented by a feature vector from an image of that word, this resulted in a pretty decent clustering. Finally the words were represented by ten feature vectors from different images of that word, but this did not results in any clustering in the network. Using feature vectors instead of Softmax units had a slightly worse performance, but when using ten feature vectors per word the network was unable to cluster the two contexts.

1. Introduction

1.1. Handwriting recognition

Off-line handwriting recognition is the conversion of handwritten text on paper to a digital form of text. This has many advantages like the preservation of old documents, the availability to many people and it makes it easier to search these documents. Unfortunately handwriting recognition is a difficult process. Even when considering only the Latin script, there are many differences in handwriting. Every writer has a different script and even individual characters can be written different, depending on surrounding letters and its position in a word. Other problems occur when characters overlap or are written connected, then it becomes extremely difficult to correctly segment these words into characters [1].

A solution is proposed in the paper "Handwritten-Word Spotting Using Biologically Inspired Features" by van der Zant, Schomaker and Haak [2]. The system described in the paper is called Monk. It has the goal of converting historical manuscripts into digital form. These historical manuscripts were often written with characters connected. The Monk system uses whole-word recognition, because words are easier isolated than connected characters. For recognizing whole words instead of single characters, most machine learning algorithms need hundreds of labeled examples of a word for training [2]. This takes a lot of effort from humans and has to be repeated for every new script. In the Monk system a word is labeled only once, then a list of possible words with the same label is generated through data-mining. Through a web browser users can quickly select correct words from the list, which can then be used for further training. Both Hidden Markov Models and Support Vector Machines already performed well in suggesting labels for unlabeled words, when presented with sufficient training data [2]. In the paper a new method called Standard Model Features (SMV) based on the visual cortex [3] is adapted to work with the Monk system. Nearest Neighbor is performed on the resulting feature vectors, with 89% accuracy when using Hamming distance as the distance metric. This method works well on infrequent occurring words and few labeled data [2].

1.2. Semantic clustering

It would be useful to not only search these historical manuscripts on keywords, but also on semantic groups like names of people or places (“Brussels” and “Europe”). In a paper by Dahl, Adams and Larochelle [4] a neural network called a Restricted Boltzmann Machine [5] was used to cluster semantically similar words. This network uses unsupervised learning, which is

very useful considering the lack of labeled data in the Monk system. In the paper the network was trained on n-grams of consecutive words from corpora, the best results were found using trigrams. Similar words (like "hotel" and "restaurant") were clustered nearby in the hidden feature vector space, because during training these words were often found within the same context. Even though the words from a corpus weren't labeled on context, it was very clear what the actual label of each word was. In the Monk system this was not the case, there were only feature vectors extracted from images of words, while the actual labels were unknown.

1.3. In this paper

The aim of this paper was to use the feature vectors from images of handwritten words to model the same kind of semantic knowledge as described in the previous section. For extra power deep learning with a Deep Belief Network [5] was used. This is a combination of multiple Restricted Boltzmann Machines where each layer can model more complicated and higher-order correlations in the data [5]. There are two main difficulties when using feature vectors from handwritten words instead of words from a corpus. The first is that words from a corpus can be modeled as so-called Softmax units. A Softmax unit consists of K binary units, where K is equal to the size of the vocabulary and only the unit with the same index as the word in the vocabulary is set to 1. A vector of features from an image is a much more complex representation of a word. The second difficulty is that words with the same label are often written slightly different (sometimes very different) in handwriting. So even though two words are the same, they are represented by two different feature vectors. These difficulties made training a network on feature vectors instead of words a real challenge. To reduce the complexity a Restricted Boltzmann Machine was used to reduce the size of the feature vectors before deep learning. A very basic artificial language model was created to find out whether it is possible at all for a Deep Belief Network to model semantic knowledge. First the network was trained by representing words from the language as Softmax units, to find out whether the basic artificial language and the settings of the Deep Belief Network were capable. Then the network was trained by representing words from the artificial language as feature vectors from the Monk system. Once with one feature vector per word and once with ten different feature vectors per word.

2. Deep Belief Network

A Deep Belief Network [5] is a multilayered neural network. It is created by stacking Restricted Boltzmann Machines [5]. The multilayered network can be trained layer by layer, which allows fast learning for many layers. After training a layer, the whole training set is given to the layer as input. The output of that layer is then used as input for training the next layer.

2.1. Restricted Boltzmann Machine

A normal Boltzmann Machine is a neural network with symmetric connections between binary, stochastic units [6]. In this network a unit is either on or off and whether the unit is on or off is based on a probability derived from the connections with other units. All units are connected and the connections between units are symmetric. In a Restricted Boltzmann Machine [5] the units are divided in two parts: the visible units and the hidden units. There are no connections between any two visible units and no connections between any two hidden units.

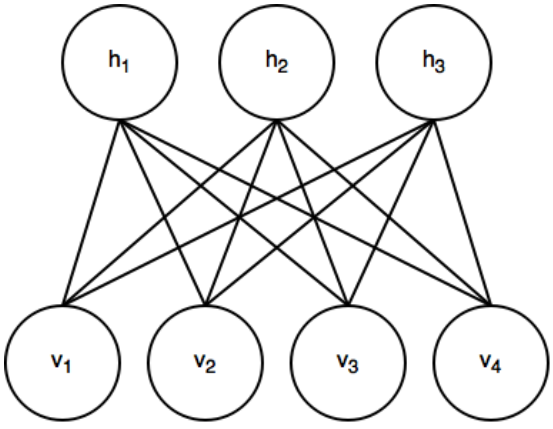

Figure 1: Diagram of a Restricted Boltzmann Machine.

The probability to turn on a hidden unit j is the sigmoid function of its activation, where the activation is the bias of hidden unit j plus the sum of all visible units i multiplied with the corresponding weight:

$$p(h_j=1) = \frac{1}{1+\exp(-act_j)} \quad (1.1)$$

$$act_j = \sum_i (w_{ij} v_i + b_j) \quad (1.2)$$

Where h_j is hidden unit j, v_i is the state of visible unit i, w_{ij} is the weight between visible unit i and hidden unit j, and b_i is the bias of hidden unit j. The probability to turn on visible unit i is almost the same, since the connections are symmetric:

$$p(v_i=1) = \frac{1}{1+\exp(-act_i)} \quad (2.1)$$

$$act_i = \sum_j (w_{ij} h_j + b_i) \quad (2.2)$$

In these formula v_i is visible unit i, h_j is the state of hidden unit j, w_{ij} is the weight between visible unit i and hidden unit j, and b_i is the bias of visible unit i.

The units in this network are binary, but the data used in this project are probabilities between 0 and 1. These probabilities are used directly as the states of the visible units. The hidden units still use binary states, to create an information bottleneck to regularize the data, which is important for learning [7]. To increase the learning speed the probabilities of both the visible states and the hidden states are used for learning the weights and biases. The code that was used for training Restricted Boltzmann Machines is an adaption of the Matlab code used in [8] from the website of G. E. Hinton [9]. It was edited to allow training data that doesn't fit in RAM, by loading partial data from a Matfile.

2.2. Contrastive Divergence

The weights and biases of the network are learned using an algorithm called Contrastive Divergence [7]. Every network configuration (states, weights and biases) has an energy. A configuration between visible units and hidden units with a low energy has a high probability of occurring together [7]. When training a network, the goal is to give the right configurations between visible units and hidden units a low energy, while giving other configurations a high energy.

$$\Delta w_{ij} = \varepsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model})$$

The $\langle v_i h_j \rangle$ is the correlation between unit i and unit j, which is a measure of how often these units are on together and is calculated by simply multiplying unit i with unit j. The $\langle v_i h_j \rangle_{data}$ is the configuration after setting the states of visible units i equal to a training set and calculating the states of hidden units j with formula 1. The $\langle v_i h_j \rangle_{model}$ is an estimation of a configuration in a local energy minimum. Ideally we want the data configuration to be in an energy minimum, not the model configuration. So the model is trained by increasing the weights between units that are on together in the data configuration, while decreasing the weights that are on together in the model configuration. This way the data configurations between visible and hidden units has a higher chance of occurring, so the hidden layer gets better at representing the visible layer. The ε is the learning rate, to make the learning smoother.

The model configuration can be obtained by alternating Gibbs sampling. First the states of the hidden units are calculated with formula 1, then the states of the visible units are calculated with formula 2. This process is repeated over and over again for a long time. Ultimately this process will end up with a configuration in a local minimum, but this takes too much time. To estimate the direction of the minimum only one cycle is performed, after calculating the hidden units from the training set.

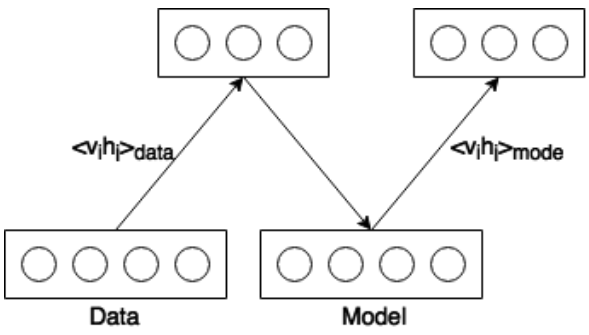

Figure 2: Diagram of Contrastive Divergence.

3. Dimension reduction

The first part of this project was to reduce the dimensionality of the feature vectors from the Monk system. The feature vectors had a dimensionality of 4356 features per vector. The dimensionality was reduced to 50 features per vector, which was a reduction of almost 99%. To do this a Restricted Boltzmann Machine [5] was used. This network can reduce high-dimensional data to low-dimensional data through unsupervised learning [8]. The unsupervised learning means that there are no labels needed for training. This works well with the lack of labeled data in the Monk system. The 4356 features were used as activation to the 4356 visible units of the network, while the activation of the 50 hidden units were interpreted as the low-dimensional feature vector. Nearest Neighbor can be applied to these low-dimensional feature vectors to find other feature vectors with similar features. Labeled word zones from the Monk system were used to test the performance of Nearest Neighbor on both the original feature vectors and the low-dimensional feature vectors.

3.1. Data set

The data used in this project came from the journal of Charlotte Perkins Gilman. This journal was already used in the Monk system [10], which gave a lot of preprocessed data to work with. The Monk system first divides all pages into lines. These lines are then split up into word zones through word mining [11]. Some word zones in the data do not correspond perfectly with whole words, they either correspond with only a part of the word or span multiple words.

Most word zones were converted to feature vectors by the system. These feature vectors had a size of 4356 features, with a value between 0 and 1. There were 385620 word zones with a corresponding feature vector, which were used as training data. These word zones did have labels, but these labels came from machine learning and weren't always correct. Fortunately no labels are needed for training a Restricted Boltzmann Machine.

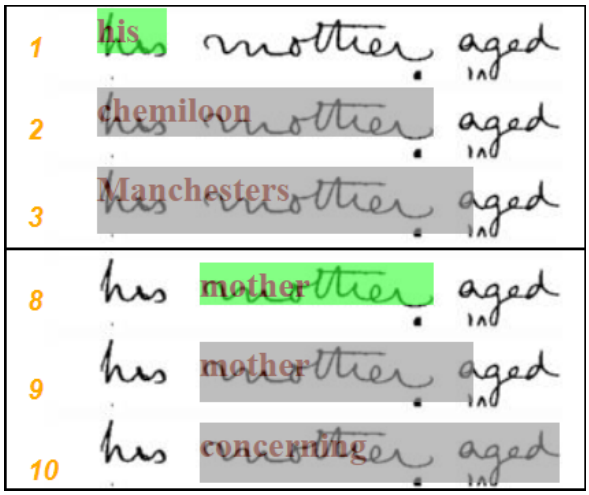

Figure 3: Some word zones of page 6 line 9 of the Gilman journal from the Monk system.

In the Monk system humans can create or confirm labels of word zones. From the 385620 word zones with feature vectors 6823 also had labels that were created or confirmed by humans. In this project the aim was to find other word zones with the same label. Since this is not possible for word zones that occur only once, these word zones were excluded. That left 6032 word zones that occurred more than once, which were used as test data.

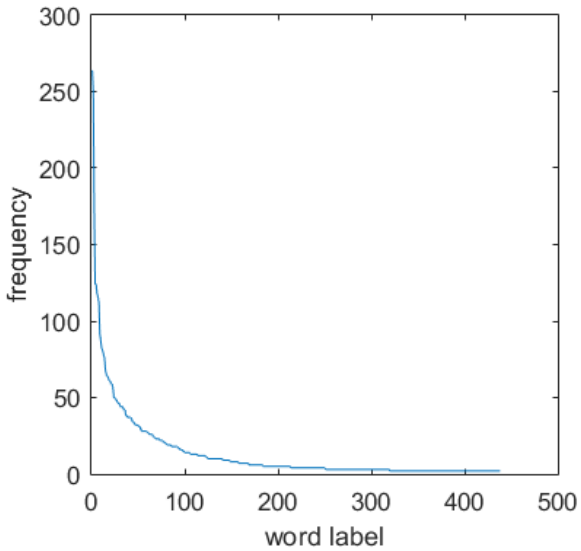

Figure 4: Plot of the frequency of a word label from the 6032 word zones that were used as test data.

3.2. Methods

The settings of the Restricted Boltzmann Machine mostly followed the guide to training Restricted Boltzmann Machines by Hinton [7]. The initial values of the weights were random values from a zero-mean Gaussian distribution

with a 0.01 standard deviation. Both the visible and the hidden bias were set to 0. To prevent overfitting a weight-cost of 0.0001 was used as recommended in the training guide. When using Matlab matrix versus matrix multiplications are very fast. To use this to make the program run much faster mini-batches with a size of 10 were used. This means that the weights were only updates after 10 cases. The changes in weight were added together and divided by ten before applied to the weights. A momentum of 0.9 was used, to increase the speed of learning and making the learning more stable by damping oscillations [7]. Because the weight changes are very large in the beginning of the learning process, it is better to use a more conservative momentum for the beginning, 0.5 in this case.

For the learning rate the training guide recommended a value that made the weight updates about 1000 times smaller than the weights itself. The value of 0.01 was just about right for this data. The guide also states that it pays off to decrease the learning rate at the end, to fine-tune the weights and biases. In this project the Restricted Boltzmann Machine was trained for 50 epochs, the first 40 with a learning rate of 0.01 and the last 10 with a learning rate of 0.001. The first 10 epochs used a conservative momentum of 0.5.

The network was trained with the feature vectors of the training data described in the previous section. After training the network the feature vectors of the test data were run through the network. Nearest Neighbor search was performed on both the high-dimensional feature vectors (4356 features) and the resulting low-dimensional feature vectors (50 features), to determine whether the low-dimensional feature vectors were still a good representation of the data. For each word zone in the test data the best Nearest Neighbor was selected, then the labels of these two word zones were compared. The results reported in table 1 are the percentages in which the target word zone and the best Nearest Neighbor had the same label. The test data only contained feature vectors of word zones that occurred at least two times in the test data, so it was always possible for Nearest Neighbor to find a word zone with the same label. Nearest Neighbor search was performed using four different metrics: Euclidean distance, Manhattan distance, Pearson correlation and Cosine similarity.

3.3. Results

	4356 features	50 features
Euclidean distance	39.24 %	72.33 %
Manhattan distance	74.62 %	73.02 %
Pearson correlation	61.70 %	71.44 %
Cosine correlation	62.85 %	70.97 %

Table 1: Percentage of Nearest Neighbors having the same labels as the target word for both high and low dimensional feature vectors.

The Nearest Neighbor search on the high-dimensional feature vectors with Manhattan distance performed the best, with 75% of the best Nearest Neighbors having the same label. The same Nearest Neighbor search on the low-dimensional feature vectors performed only slightly worse, with 73%. When using other metrics for calculating the Nearest Neighbor the low-dimensional feature vectors performed better than the high-dimensional ones. Even though the best performance comes from the high-dimensional feature vectors, the low-dimensional feature vectors are a good representation of the data and are only 1% of the size.

4. Semantic clustering

After reducing the dimensionality of the feature vectors, the low-dimensional feature vectors were used in a very basic artificial language. A sentence from this language was always three words long. Every sentence belonged to one of the two types of contexts in this language: people or food. The Deep Belief Network was trained on trigrams of words from these sentences, to find out whether this resulted in semantic knowledge in the network. Three methods of representing the words in the language were used. The first method was representing the words with Softmax units. The second method was representing the words with a feature vector

from an image of that word. The third method was representing the words with feature vectors from ten different images of these words. These methods are further explained in section 4.2.

4.1. Data set

To test whether a Deep Belief Network is capable of modeling semantic knowledge by training on feature vectors instead of Softmax units a very basic artificial language was constructed. In this language only words that occurred at least ten times in the test set from section 3 were used. This way each word had at least ten different feature vectors that could be used. In this language the size of the vocabulary was 22 words and every sentence was exactly three words long. There were two different kinds of sentences: sentences about meeting people and sentences about eating food. Half of the vocabulary was only used in the sentences about meeting people and the other half only in the sentences about eating food. An example of a sentence about meeting people is: "write Carrie again". An example of a sentence about eating food is: "her breakfast with". Not all sentences are complete or correct, but that was not a problem as the actual meaning of the words was irrelevant. The important part was that some words only occur together with others and every word had at least ten feature vectors. Figure 5 shows all possible sentences. A random sentence can be created by following the lines from left to right, resulting in a sentence of three words from one of the two contexts.

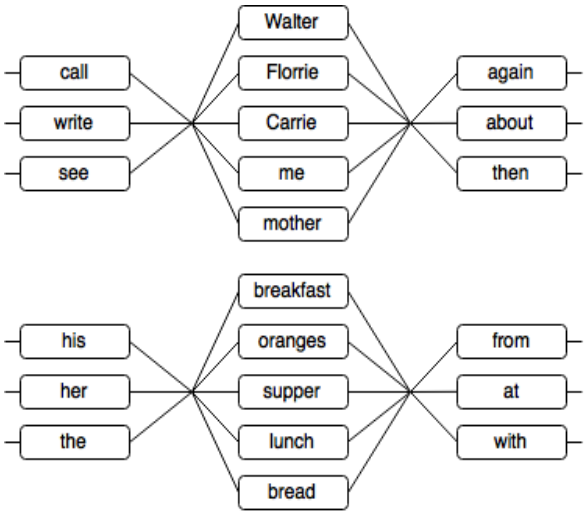

Figure 5: Basic artificial language model.

4.2. Methods

This paper used a Deep Belief Network [5] with three hidden layers, to increase the power of the network as each layer of the network is able to model more complicated correlations in the data. When using the low-dimensional feature vectors as input to the visible layer, the input consisted of three feature vectors with 50 features each, so the network had a visible layer of 150 units. There was no more need for dimension reduction, so the first, second and third hidden layer also had 150 units each.

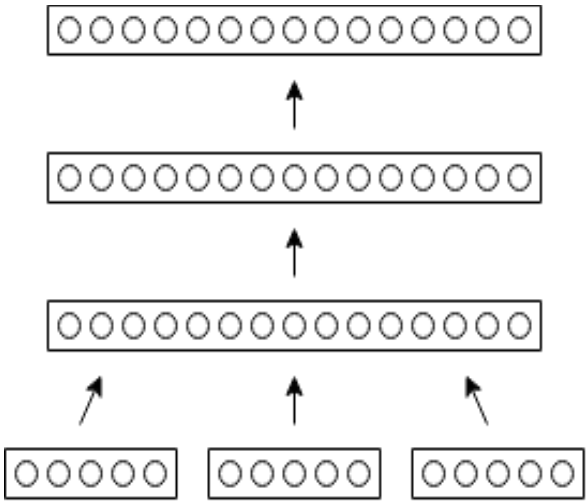

Figure 6: Deep Belief Network with three hidden layers. The visible layer gets three concatenated feature vectors or Softmax units as input.

In the case of the Softmax units, the visible layer consisted of 66 units. The hidden layers all kept a size of 150 units per layer, to make the comparison between Softmax units and feature vectors as fair as possible given the different sizes of the input.

Between every layer the weights and biases were trained using Restricted Boltzmann Machines [5]. The settings for all three Restricted Boltzmann Machines were the same and very similar to the settings used during the dimension reduction. The initial values of the weights were random values from a zero-mean Gaussian distribution with a 0.01 standard deviation. Both the visible and the hidden bias were set to 0. To prevent overfitting a weight-cost of 0.0001 was used. The size of the mini-batches was 10 cases per batch. A momentum of 0.9 (0.5 for the first 10 epochs) was used. The best learning rate for this data was 0.1 for the first layer. The next layers used a learning rate of 0.01. Every epoch

the network was trained on the whole training set. The training was done in 100 epochs for every layer. To finetune the weights at the end of training each layer the learning rate was divided by ten, for the last 10 epochs.

Three methods were used to represent the word trigrams as input to the network. In the first method a single word was represented as a Softmax unit, which is a set of binary units where only one unit is set to 1. The size of this set was equal to the size of the vocabulary, which was 22 in this language. The index of a word in the language determined which unit was switched on. For every sentence the Softmax units of the three words in that sentence were concatenated together and given as input to the visible layer. Since a Softmax unit consisted of 22 binary units, the input to the network was 66 binary units for every sentence. The paper by Dahl, Adams and Larochelle [4] already showed that a Restricted Boltzmann Machine with Softmax units as input is capable of clustering semantically similar words. They even used a real language which is obviously more complicated and has a larger vocabulary. The first method using Softmax units was merely used to show whether the clustering also works using the artificial language and a Deep Belief Network.

In the second method feature vectors were used instead of Softmax units. These vectors were a collection of features from images of words, which came from the Monk system. The feature vectors were first reduced in size in section 3, to only 50 features per vector. Every word in the language was represented as a single feature vector from an image of that word, after dimension reduction in section 3. For every sentence the three feature vectors of the words in that sentence were concatenated together and given as input to the visible layer. Since a single feature vector consisted of 50 features, a sentence had a size of 150 features and the visible layer a size of 150 visible units. Contrary to the first method, unlabeled data could be used. In the first method the label of a word was needed to known which binary unit to switch on. In this method images of the words were used, without the network knowing the actual label of the words, only the features extracted from the image.

The third method also used feature vectors to represent word trigrams as input to the network. In the second method a word was always represented by the same feature vector. However in handwriting a word is usually written different every time, which results in different feature vectors for the same word. To find out whether the Deep Belief Network could handle this, the third method used ten feature vectors per word. For every word in the language ten feature vectors from images of that word were selected from the training data of section 3. For every sentence three random feature vectors of the words in that sentence were selected.

In method 1 and 2 there were only 90 possible different input configurations, 45 (3*5*3) per context. But in method 3 there were 90,000 possible different input configurations, since every word could have ten times as many different vectors, so 45,000 (30*50*30) per context. To keep the settings of training the Deep Belief Network the same for all three methods for every method 100,000 random sentences were selected. Every epoch all 100,000 sentences were used exactly once to train the network.

In the paper by Dahl, Adams and Larochelle [4] to test the network only one Softmax unit was set to a word, while all binary units in the first and third Softmax unit were set to 0. This way the resulting feature vector from the hidden layer of the network is based only on one word. Semantically similar words were located close to each other in the hidden feature vector space. To test the Deep Belief Network in this paper only the middle words from the artificial language were used. These words were either about people or about food. For every word the corresponding Softmax unit or feature vector was selected. Method 3 had ten feature vectors per word, in this case a random one was selected. These units were given as input to the network, while the units of the first and last word were all set to 0. This resulted in a feature vector at the last hidden layer. Nearest Neighbor was performed on these feature vectors to find out which words were located close to each other. The metric used for Nearest Neighbor was the Manhattan distance, because this metric performed the best in section 3.

Walter	Florrie	Carrie	me	mother	breakfast	oranges	supper	lunch	bread
Florrie	Walter	me	mother	me	bread	supper	oranges	bread	lunch
Carrie	Carrie	Walter	Carrie	Carrie	supper	bread	bread	oranges	breakfast
me	me	Florrie	Florrie	Florrie	oranges	oranges	breakfast	breakfast	oranges
mother	mother	mother	Walter	Walter	lunch	breakfast	oranges	supper	supper
oranges	breakfast	oranges	supper	oranges	me	Carrie	me	Walter	Walter
bread	bread	supper	breakfast	supper	mother	mother	mother	mother	me
supper	supper	breakfast	oranges	breakfast	Carrie	me	Carrie	Florrie	mother
breakfast	oranges	bread	bread	bread	Florrie	Walter	Walter	me	Florrie
lunch	lunch	lunch	lunch	lunch	Walter	Florrie	Florrie	Carrie	Carrie

Figure 7: Clustering from method 1: using Softmax units.

Walter	Florrie	Carrie	me	mother	breakfast	oranges	supper	lunch	bread
Florrie	Walter	me	mother	me	bread	supper	oranges	bread	lunch
Carrie	Carrie	Walter	Carrie	supper	supper	bread	mother	oranges	breakfast
me	me	Florrie	Walter	Carrie	orange	mother	breakfast	breakfast	oranges
mother	mother	mother	Florrie	oranges	lunch	breakfast	bread	supper	supper
oranges	breakfast	oranges	supper	Walter	me	lunch	me	Walter	Walter
bread	bread	supper	breakfast	Florrie	mother	Carrie	lunch	mother	me
supper	supper	breakfast	oranges	breakfast	Carrie	me	Carrie	Florrie	mother
breakfast	oranges	bread	bread	bread	Florrie	Walter	Walter	me	Florrie
lunch	lunch	lunch	lunch	lunch	Walter	Florrie	Florrie	Carrie	Carrie

Figure 8: Clustering from method 2: using one feature vector per word.

Walter	Florrie	Carrie	me	mother	breakfast	oranges	supper	lunch	bread
Florrie	Walter	me	Carrie	supper	me	supper	mother	bread	lunch
lunch	Carrie	breakfast	breakfast	oranges	Carrie	mother	oranges	Walter	breakfast
Carrie	breakfast	Florrie	mother	me	bread	me	me	Florrie	Walter
bread	me	Walter	bread	breakfast	Florrie	bread	breakfast	breakfast	me
breakfast	bread	mother	supper	Carrie	mother	Carrie	bread	Carrie	Florrie
me	lunch	bread	Florrie	bread	Walter	lunch	Carrie	oranges	Carrie
oranges	mother	supper	Walter	Florrie	supper	breakfast	Florrie	supper	mother
mother	supper	lunch	oranges	lunch	lunch	Walter	lunch	mother	supper
supper	oranges	oranges	lunch	Walter	oranges	Florrie	Walter	me	oranges

Figure 9: Clustering from method 3: using ten feature vectors per word.

4.3. Results

The results can be seen in figure 7, 8 and 9. In the first row (in a darker shade) are the target words, the second row contains the nearest neighbor of the target word, the third row the second nearest neighbor of the target word and so on. The blue words are the words about people, while the red words are about food. The results from method 1 can be seen in figure 7. In this method Softmax units were used. The figure confirms that this was not a problem, the words about people were always closer to each other, while words about food were closer to other words about food. This showed that a Deep Belief Network was able to cluster semantically similar words using a very basic artificial language, when using Softmax units. In figure 8 the results from method 2 can be seen. In this method the Softmax units were replaced by feature vectors from images of words. The figure shows that the clustering went pretty well, the first nearest neighbor always had the same context as the target word. However in some cases the second nearest neighbor already had a different context than the target word. Method 3 used ten different feature vectors instead of just one. In figure 9 the results of this method can be seen. For this method there was no clustering at all. The Deep Belief Network showed great difficulty when a word could be represented by ten different feature vectors.

5. Discussion

The aim of this paper was to cluster semantically similar words, using feature vectors from unlabeled images, by training on trigrams. If this could be applied to handwritten documents, people would be able to search semantically similar words in a document where the words aren't even recognized yet. Using Softmax units resulted in perfect clustering, but Softmax units need labeled data. The most interesting handwritten documents are those without much labeled data. This was the reason for using feature vectors from the images of words. The results showed that deep learning struggled on semantic clustering, even when using a very basic language. When using feature vectors instead of Softmax units the clustering was still pretty good, even though a feature vector is a much more complicated representation of a word. The biggest problem appeared to be the different feature vectors per word. This was not very surprising, since section 3 showed that only 73% of the nearest neighbors were actually the same word. Still no apparent clustering at all was a bit disappointing.

The current method of using trigrams of feature vectors in a Deep Belief Network is not good enough to be used with real data yet. In this paper a very basic artificial language was used, while historical manuscripts from the Monk are written in real languages, which are obviously much more complicated. When using ten different feature vectors per word the network wasn't capable of clustering the two contexts any more. In an actual text there are many more contexts and every word has a different feature vector. Before using a Deep Belief Network to cluster semantically similar words in a real handwritten text, much improvement is needed.

A part that could be improved upon is the size of the hidden layers and the number of hidden layers. Perhaps the 150 units per hidden layer were not capable of containing enough information required for the task. In the paper by G. E. Hinton, S. Osindero, M. Welling and Y. W. Teh [5] the hidden layers used 500 hidden units, for recognizing a single character. Combined with more layers would give the network the ability of containing more information and computing more complicated correlations in the data. Another possible way to improve results is to reduce the complexity of the feature vectors even more. Reducing the size of a feature vector even more had drastic effects on the results. Perhaps a larger feature vector, but encouraging sparsity works better. In the training guide by Hinton [7] sparse hidden units are explained, where units are only rarely active. By using the binary states of these sparse hidden units, the resulting feature vector will be more similar to Softmax units. Even though much more data is lost, maybe the network is able to perform better on a less complex representation.

6. References

[1] Yi Lu and M. Shridhar, "Character segmentation in handwritten words - an overview", 1995

[2] T. van der Zant, L. Schomaker and K. Haak, "Handwritten-Word Spotting Using Biologically Inspired Features", 2008

[3] T. Serre et al., "Robust Object Recognition with Cortex-Like Mechanism", 2007

[4] G. E. Dahl, R. P. Adams and H. Larochelle, "Training Restricted Boltzmann Machines on Word Observations", 2012

[5] G. E. Hinton, S. Osindero, M. Welling and Y. W. Teh, "A fast learning algorithm for deep belief nets", 2006

[6] D. H. Ackley, G. E. Hinton and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines", 1985

[7] G. E. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines", 2010

[8] G. E. Hinton, R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks", 2006

[9] G. E. Hinton, "http://www.cs.toronto .edu/~hinton/MatlabForSciencePaper.html"

[10] L. Schomaker, "http://www.monkweb.nl/"

[11] L. Schomaker, "Word Mining in a Sparsely Labeled Handwritten Collection", 2008