Mail Classification using Deep Learning

Paper

Rico Meinl

January 2, 2019

Contents

1 Definition

1.1 Project Overview

Modern Enterprises have to deal with constant flows of information and customer enquiries from all different sources such as Email, Chat and Call.
Recently, automated solutions such as Chatbots have become a popular way to deal with the most frequently asked questions which saved a lot of resources and manual labor. These solutions have been able to take over many tasks from classification, to labelling and even fully-automated answering of the enquiry.
Most of these systems are considered somewhat intelligent as they draw their knowledge from a Knowledge Base with which they interact through a set of predefined rules. Through Pattern Matching using Regular Expressions, information such as the customers intent can be extracted and used for the task of classifying the request into a set of categories.
The problem is that it takes a long time to properly implement and refine these rules. They also have a fair amount of problems when dealing with unseen requests.
The goal is to develop a solution that can learn from a large corpus of data (e.g. email conversations) about how to handle customer requests that are often encountered. Such a solution consists of two parts:
The first part is the automated classification of incoming requests into categories. The second part then looks at the request, knowing the intent based on the category and tries to generate an answer based on the what it learned from the data it has seen.
At this point in time the first part is accomplished by the Pattern Matching approach which matches a request to an answer from the knowledge base. The goal of this work is to propose a Deep Learning approach for this first part, which will lay the foundation for further research and can be a first estimator for the feasibility of the whole 2-part project.
In order to evaluate our approach, a corpus of about 150k emails, which have been prelabelled and divided into 28 categories, was extracted from novominds product 'novomind iMAIL'.

1.2 Problem Statement

As classification we understand a method that divides a corpus of objects into different classes or categories.
"Classication [is the] process of determining the subject content of an item and assigning to it the appropriate classication number, code or notation from a classication scheme. Grouping together of like things according to some common quality or characteristic." [1]
Manually classifying emails into categories by defining rules (e.g. by using Regular Expressions) is a commonly used method as it allows for a high degree of control and usually achieves pretty decent results. Regardless, it is also a tedious task and has to be done basically from scratch for every new customer.

It allows for very little automation as new customers have their own individual requirements for category classification. Having an automated solution, that simply learns from past email conversations (which exist in all cases) would not only save a lot of manual creation but also enables more robustness as hard-coded rules are very limited in their ability to correctly classify unseen cases.
In the recent years there has been quite significant process in the field of Natural Language Processing which gives us an advanced toolbox for an automated solution. Because of the rise in computing power, Neural Networks and especially Sequence Models have become an established method to deal with such tasks, achieving human-level performance in even some of the most challenging ones (e.g. answering a set of questions about a text).
Recurrent Neural Networks, with Gated Recurrent Units (GRUs) and Long Short Term Memory Cells (LSTMs) are able to process sequences of words and understand semantic relations. In this paper I will go over the foundations of those models, slowly going up the hierarchy and eventually present a state-of-the-art model which we will use for the proposed task of classifying emails into categories.

1.3 Metrics

To effectively evaluate the results of our tested methods we need to define a quality metric to be able to compare different methods. The definition of the following quality metrics have their roots in the information retrieval. To effectively evaluate a categorization we need to know the ground truth.

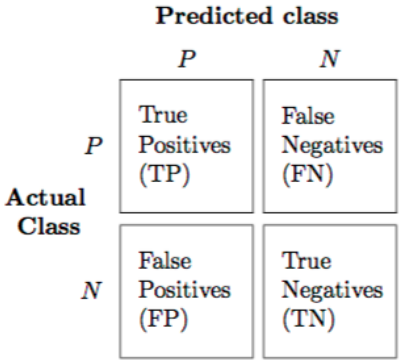

Figure 1: Confusion Matrix

The precision of the categorization is defined by the correctly classified positives divided by the total number of predicted positives.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

The recall is defined by the correctly classified positives divided by the total number of actual positives.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Precision and recall are often not significant enough to be used in practical applications. The F1 Score combines them and takes the harmonic average.

$$F1 = \frac{2 * P * R}{P + R} \tag{3}$$

There are two ways to evaluate our classification result over the whole set by taking the average. In micro averaging we take the sum of all metrics (TP, FP, FN, TN) over all categories and insert them into the equations defined above.
In this case we equally weigh each document disregarding the distribution over all categories. In the following I show an example calculation for the recall with micro averaging:

$$r^{\mu} = \frac{\sum_{j=1}^{k} a_j}{\sum_{j=1}^{k} (a_j + c_j)} \tag{4}$$

Macro averaging calculates the average over all results based on the metric for each category. Doing that we put equal weights to all categories to make sure a category with a small amounts of documents influences the result as much as a category with a lot of documents.

$$r^{M} = \frac{\sum_{j=1}^{k} r_j}{k} \tag{5}$$

2 Analysis

Novomind was founded in 1999 with the objective to develop intelligent commerce and customer service software solutions. The product suite can be divided into Ecommerce and Customer Service solutions where novomind iSHOP, novomind iMARKET and novomind iPIM belong to the former and novomind iAGENT to the latter.
Novomind iAGENT offers a module for mail management in 'novomind iAGENT Mail' which has the purpose of supporting support center teams when processing incoming customer enquiries.
It can be categorized as a product for email response management and works with a data-driven rule-based system to categorize emails. All the components of novomind iAGENT are integrated as modules and share the same knowledge base. The following process illustrates an iAGENT Mail specific workflow:

1. A customer sends an email to the account of a company. (e.g. info@examplecompany.com)

2. After receiving the email it is possible to immediately send a confirmation of receipt to the customer.

3. The incoming emails are now classified using rules, which detect the customer and categorize the email. The email is assigned with a Ticket ID which enables tracking and future identification.

4. Based on the use case the email can be answered fully-automatic (fully-automatic response) but most of the time it is routed internally to a service agent.

5. The service agent can now generate an answer based on predefined text modules recommended by the system, create an answer from scratch or re-categorize the email to forward it to another agent.

6. All generated information as well as the Ticket ID are saved in a database and added to the existing customer data.

The data set described in the next section was extracted from 'novomind iAGENT MAIL'. The goal of this work is to improve the workflow described above by using a Deep Learning approach for the classification of categories in step No. 3 and eventually automate the steps No. 4 and 5 to a degree where the agent only has to check upon the most unusual requests.

2.1 Data Exploration

The data being used consists of emails from customers which have been classified into 28 different categories. Some more statistics about the dataset can be found below.

- Total amount of emails: 155,138
- Average Amount of emails per Category: 5,541
- Number of Categories: 28
- Average length of emails: 1089 Letters and 114 Words
- Embedding length per email: 769 dims

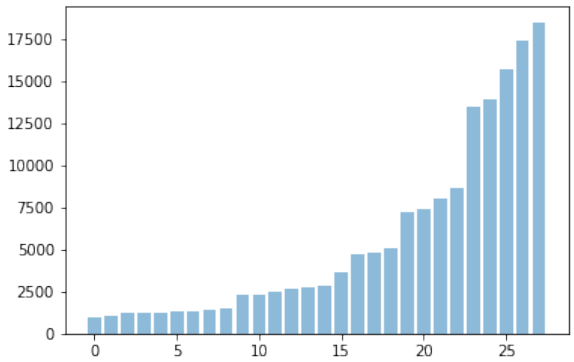

Figure 2: Distribution over classes

Some examples for categories are:

Change of address
Order
Data Privacy
Frustration
Catalogue
Customer Account
Delivery
Complaints
Returns
Paypal
Payments
...

Table 1: Category Examples

The emails all have the following structure:

TO: example@examplecompany.com
FROM: example@examplecustomer.com
REPLYTO: example@examplecustomer.com
SUBJECT: Example Subject
— Message Body —
(in case of a reply)
> *Beginning of an example reply*
>
>
>

When we compute the average length and standard deviation for our the amount of words in our emails after the undertaken preprocessing steps described in 3.1 on page 14 we get the following results:

AVG	STD
103,43	123,16

Table 2: Average and Standard Deviation of sentence length

2.2 Algorithms and Techniques

In order to process the data set described in the last section we are going to use different types of Sequence models. These types of Neural Networks are very strong with processing and creating sequences.
Common use cases are text classification, translation and creation. They can also be applied to music or image sequences.
Its main advantage over a vanilla neural network is that it uses one set of shared weights to recursively process sequences of inputs which enables it to deal with sequential input data very well. Artificial Neural Networks (ANNs) do not share features learned across different positions of text.
On top of that our sequences have a quite large deviation in length (see figure 2 on page 7) and Artificial Neural Networks are not able to process different lengths in different examples.
Another very significant characteristic of RNNs that can be observed is that it does not only use information from the corresponding input x_t but also from all the previous ones, which is why many people consider it having a kind of internal memory.
As there are a lot of interdependencies in between the words in each of our emails that contain the intent of the customer, we are going to use a RNN architecture. As I introduce the architectures I will use the following notation:

$x^{<t>}$: t-th element in the input sequence
$x^{(i)<t>}$: t-th element in the input sequence of training examples i
T_x : length of the input sequence
T_y : length of the output sequence
$T_x^{(i)}$: input sequence length for training example i
$y^{<t>}$: t-th element in the output sequence
$y^{(i)<t>}$: t-th element in the output sequence of training examples i

The graphic below shows the standard Recurrent Neural Network architecture, which I unrolled here to make it easier to understand the time dependent inputs x^t which represent a word in a sentence for examples, and which are fed into the shared weights W_{ax}.

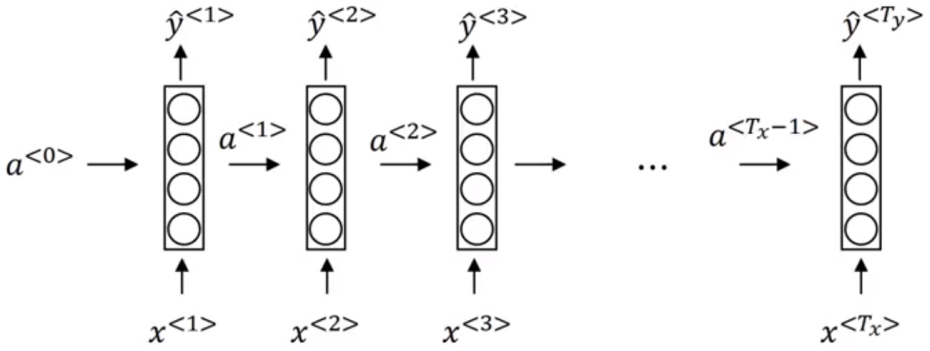

Figure 3: Recurrent Neural Network

In order to compute the forward computation we use the following formulas:

$$\begin{aligned} a^{<t>} &= g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \\ \hat{y}^{<t>} &= g(W_{ya}a^{<t>} + b_y) \end{aligned} \tag{6}$$

For Recurrent Neural Networks we compute the so called backpropagation through time where we start by computing the loss function as follows. A more detailed explanation can be found in paper [2].

$$\begin{aligned} L^{<t>}(\hat{y}^{<t>}, y^{<t>}) &= -y^{<t>} log\hat{y}^{<t>} - (1- <^{<t>})log(1 - \hat{y}^{<t>}) \\ L(\hat{y}, y) &= \sum_{t-1}^{T_y} L^{<t>}(\hat{y}^{<t>}, <^{<t>}) \end{aligned} \tag{7}$$

There are many RNN architectures but we will start with the many-to-one architecture where we receive a sequence of words as an input and output a predicted value ŷ which represents the different categories in our example.

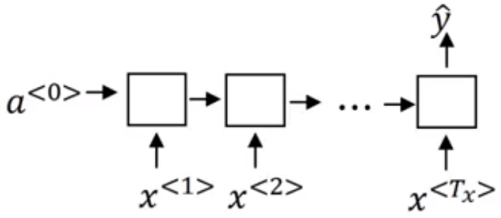

Figure 4: Many-to-one architecture

With these standard RNNs we have an architecture that is able to deal with dependencies within sentence structures.
Though a common problem arises when we are dealing with long inputs as RNNs are not very good at handling long-term dependencies. This goes back to the problem of vanishing and exploding gradients which I will not further evaluate in this work but some good explanations can be found in paper [3].
In a nutshell, to handle exploding gradients a common method used is to apply clipping gradients. To encounter vanishing gradients there are two different types of RNN cells, namely the Gated Recurrent Unit (GRU) [4] and Long Short Term Memory (LSTM) [5] cells which are much more robust to this problem, and which I will further delve into.
To get a better understanding of the difference between the architectures I want to start with the illustration of a RNN cell.

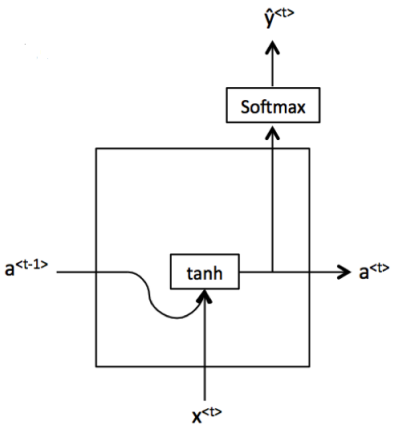

Figure 5: Recurrent Neural Network Cell

$$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a) \tag{8}$$

For the GRU cell we have an additional relevance gate which learns the relevance of the previous time step c_{t-1} for c_t and an update gate which decides when to update c_t.

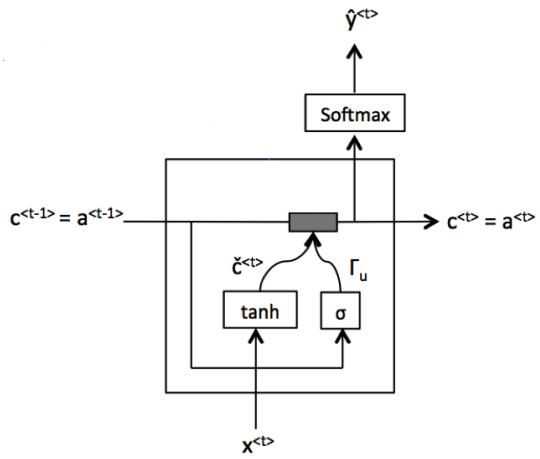

Figure 6: Gated Recurrent Unit Cell

$$
\begin{aligned}
\tilde{c}^{<t>} &= tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c) \\
\Gamma_u &= \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u) \\
\Gamma_r &= \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r) \\
c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) + c^{<t-1>} \\
a^{<t>} &= c^{<t>}
\end{aligned}
\tag{9}
$$

Here we compute a candidate for replacing c by multiplying the input from our last cell c_{t-1} by our input x_t and use the update gate to decide whether to update c or to keep it.
This allows us to detect dependencies even over long sequences of text. The update gate is represented by a sigmoid function so it will most close to 0 or 1 most of the time. With the LSTM cell we also have an update gate and on top of that a forget gate and an output gate.

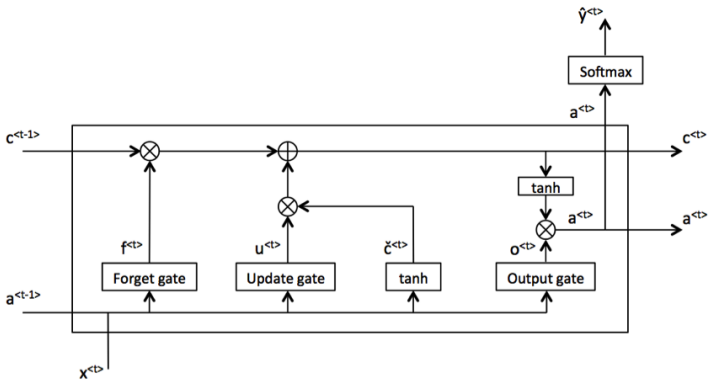

Figure 7: Long Short Term Memory Cell

$$
\begin{aligned}
\tilde{c}^{<t>} &= tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \\
\Gamma_u &= \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \\
\Gamma_f &= \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \\
\Gamma_o &= \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \\
c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \\
a^{<t>} &= \Gamma_o * tanh\, c^{<t>}
\end{aligned}
\tag{10}
$$

The forget gate determines whether the output from our previous cell will be relevant. The update gate has a similar behavior as the one explained in the section about GRUs as it is used to determine whether to use the newly generated candidate of c or to keep the previous one.
The output gate is used to generate the activation value for our cell which will be propagated to the next cell.
One more advancement to our standard RNN architecture is the birectional RNN [6] which is basically an acyclic graph and takes in information not only from earlier in the sequence but also from later on.

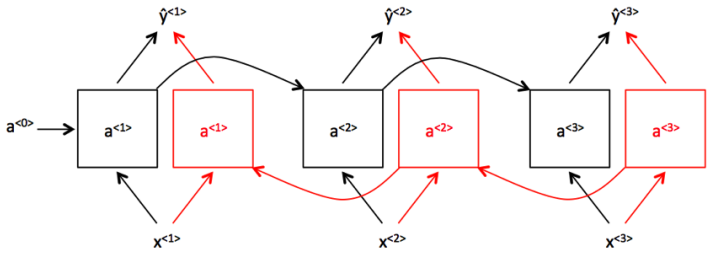

Figure 8: Bidirectional Recurrent Neural Network

Lastly I want to go over the attention model which has had quite a bit of success in recent NLP applications and builds on top of everything covered so far.
Previously we have seen the many-to-one RNN architecture which we use for our first attempts to classify the corpus of emails.
Another architecutre being used here is the many-to-many architecture which is very practical for text translation as it consists of an encoder and a decoder network.

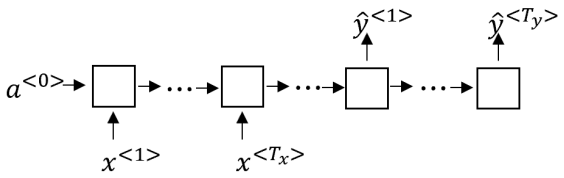

Figure 9: Many-to-many architecture

The attention model works quite similar in a way that it is not waiting for the whole input sequence before translating but starting doing it on the go, looking at one part at a time.

Such a mechanism can also be found in image recognition applications. The attention model consists of two RNNs. A bidirectional RNN to compute a set of features for each of the input words and a second RNN to generate the translation.
The input to the second RNN is governed by attention weights α which denote how much attention should be paid to each piece of the original sentence. These parameters tell us how much the context depends on the features we are getting.

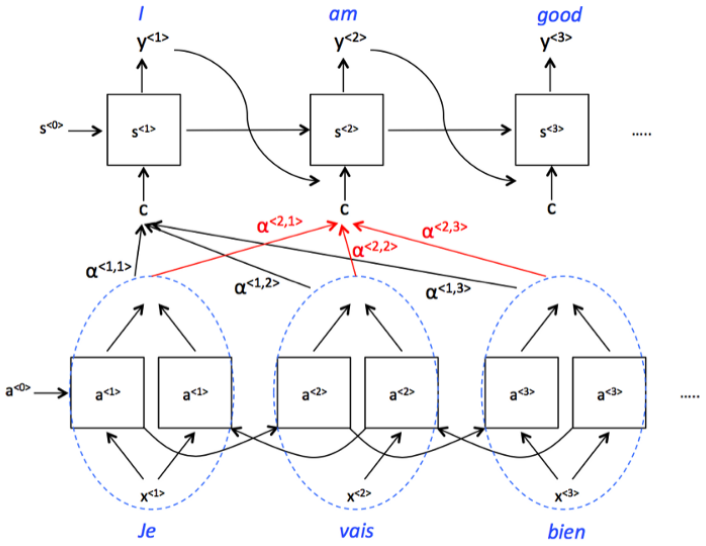

Figure 10: Attention Model

The c context is the weighted sum of the attention weights.

$$c^{<1>} = \sum_t a^{<1,t>} a^{<t>} \tag{11}$$

Then $\alpha^{<t,t>}$ is the amount of attention that $y^{<t>}$ should pay to $a^{<t>}$.

$$a^{<t,t>} = \frac{\exp(e^{<t,t>})}{\sum_{t=1}^{T_x} exp(e^{<t,t>})} \tag{12}$$

In order to compute the parameters $\alpha^{<t,t>}$ we train a small neural network where the inputs are represented by the network state in the previous time step $s^{<t-1>}$ and the features from the time step t $a^{<t>}$.

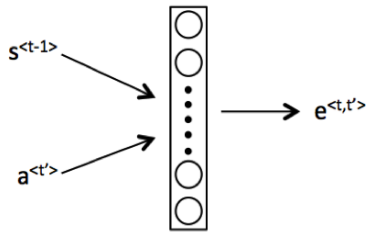

Figure 11: Small network for attention weights

Further details about this model can be found in paper [7]. This serves as the foundation for BERT (Bidirectional Encoder Representations from Transformers) which is a recent paper [8] published by researchers at Google AI Language.
BERT makes use of Transformer [9], an attention mechanism that learns contextual relations between words in a text. It is based on the encoder-decoder principle described above, where the encoder reads text input and the decoder produces a prediction for the task.
As BERTs goal is to generate a language model only the encoder mechanism is necessary. It is then possible to add a final classification layer, which is further described in section 3.2 on page 15. BERTs key technical innovation is applying the bidirectional training of Transformer to language modelling.
The use of bidirectional training gives a deeper sense of language context than previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The detailed workings of Transformer are described in the paper [9] by Google, but an illustration of the model can be found below.

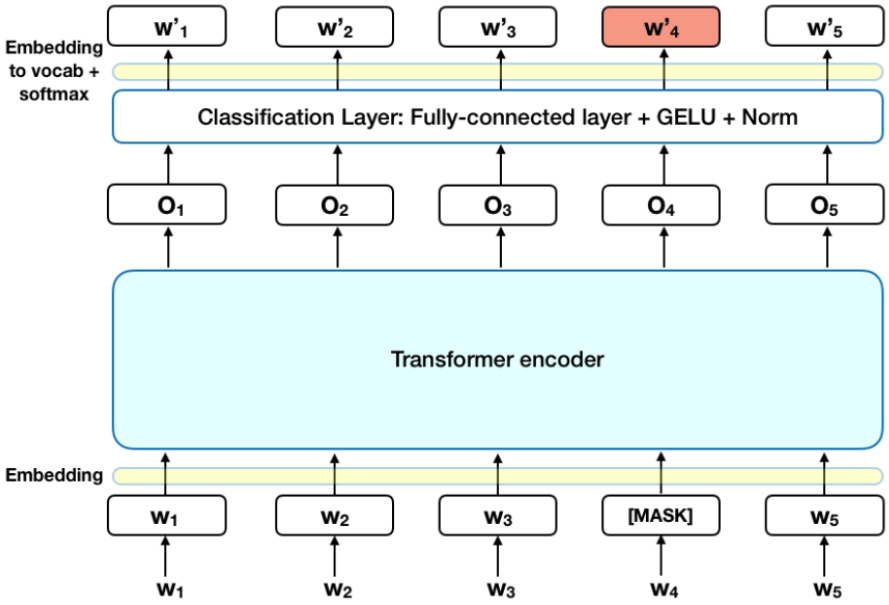

Figure 12: BERT

3 Methodology

3.1 Data Preprocessing

Before we create our training and test set we delete the Reply section (as described in section 2.1 on page 6) from all emails as they do not influence the classification decision. We also remove all of the header but the Subject part.
As a last step we make sure our preprocessed emails only contain words and no other characters. We then end up with a total dataset of 155,138 labelled emails.
We split this corpus in a train and test set with a ratio of 0.05 for the test size, using the Sklearn library [1].

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```

In order to deal with the words as input we have to find a proper representation. In earlier days it was common to define a vocabulary corpus and represent each word as a one hot encoded vector, meaning the index of the word is 1 and every other vector index 0.
A much more efficient representation can be achieved by using word embeddings. In a nutshell, we train a small network to encode each word within a specified dimension and make sure that words which are often used in the same context are close to each other. This simply means that if we use a dimension reduction technique such as PCA or T-SNE and plot the word vectors in a lower dimension we can observe that for example all animals are close to each other.

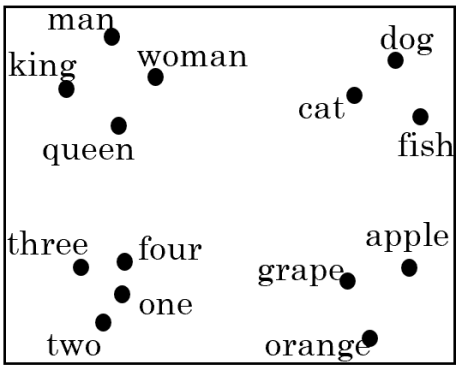

Figure 13: Embeddings Visualization

As this is quite a common approach there are pretrained embedding matrices which can be openly accessed. In our case we first use a german word2vec [10] embedding and later use BERT [8] as our embedding to represent the words.

3.2 Implementation

In order to implement the models I am going to use the Keras wrapper [11]. Keras makes it really easy to create quite complex models with very little code. The skeleton of the code is pretty straight forward.
The model is initialised, the Embedding is added to encode the input sequence and then we try out different cell types from Simple RNNs, GRUs to LSTMs.

```
model = Sequential()
model.add(Embedding(vocab_size, weights=[embedding_matrix], trainable=False))
```

We use a Dense Layer with a Softmax and the output size of the number of categories. For the optimizer we use Adam and as we have multi-class decision we use a categorical cross-entropy loss.

```
model.add(Dense(n_categories, activation=softmax))
model.compile(optimizer=adam, loss=categorical_crossentropy, metrics=[accuracy])
```

We use an additional validation split of 0.25, a batch size of 2048 and train for 75 epochs using Early stopping.

```
model.fit(X_train, y_train, validation_split=0.15, batch_size=2048,
epochs=75, shuffle=True)
```

To evaluate the efficiency of the model we use a weighted average of the micro and macro average of the F1 score. Starting off with the bidirectional RNN cell as our baseline we get a pretty fast convergence but the results are not pleasing as expected.
The GRU cell takes a bit longer to train but should be much better at handling the long-term dependencies the RNN cell is known to struggle with. Lastly the LSTM cell reaches a weighted average of 0.52.
In order to capture the complexity of the emails we try adding a second layer for each of our three cells using a dropout of 0.5 to prevent overfitting. The results can be found in the table below.

	RNN	GRU	LSTM
1-layer	0.38	0.52	0.52
2-layer + dropout	0.39	0.53	0.53

Table 3: Weighted Averages

One key characteristics of RNNs is that they do not get much better accuracies by simply stacking layers on top of each other. By analyzing other factors we find that the word embedding plays a huge role in the performance of our model.
With BERT we have access to a state of the art language model which was pretrained on

a huge vocabulary corpus. In computer vision it is a common method to use pretrained weights from a huge dataset like ImageNet [12].
BERT can be seen as the equivalent in Natural Language Processing. We try running BERT with the same parameters for batch size and epochs and with a simple dense layer with 512 neurons and the dense layer with our number of categories as the number of neurons and a softmax on top to output our categories. We also use Batch Normalization before first dense layer and Adam as our optimizer.

```
x = BatchNormalization()(input_tensor)
x = Dense(512)(x)
x = Dense(len(categories))(x) #28 categories
x = Activation(softmax)(x)
```

With this simple architecture on top of BERT we already reach a test accuracy of 88% and a weighted average of 0.88.
Taking a look at the errors our model makes we can conclude that the model needs to be more complex to be able to understand the fine borders between our categories. We therefore add another three dense layers, as defined below, in between using a dropout of 0.5 to prevent overfitting and the PReLU activation function.
During training we can observe the error rate oscillating which we can prevent using batch normalization after each output from the dense layer.

```
x = Dense(512)(x)
x = BatchNormalization()(x)
x = Activation(PReLU())(x)
x = Dropout(0.5)(x)
```

4 Results

4.1 Model Evaluation and Validation

The model takes a little over a minute to train on our training set of 147780 samples and achieves an accuracy of 93.7% and a weighted average of 0.94. The following figures show that we get our model to converge.

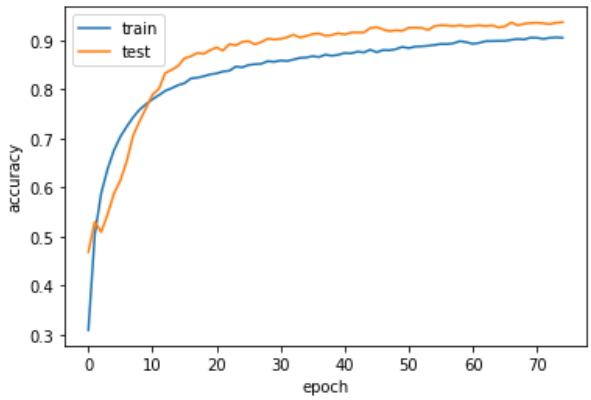

Figure 14: Model Accuracy

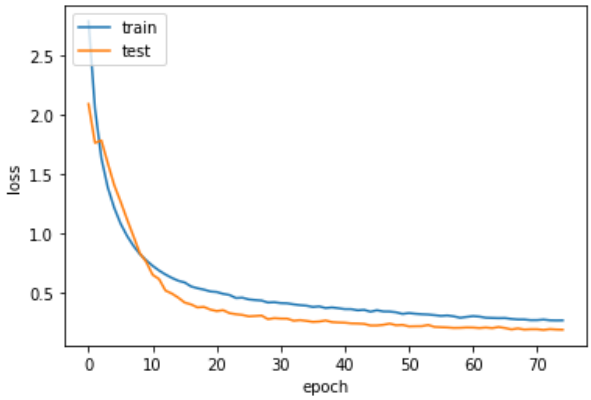

Figure 15: Model Loss

Looking at the classification report over 7778 test samples the results are quite solid as we get good equally good results for Precision, Recall and F1-score.

	Precision	Recall	F1-Score
Micro Avg.	0.94	0.94	0.94
Macro Avg.	0.90	0.89	0.90
Weighted Avg.	0.94	0.94	0.94

Table 4: Classification Report

5 Conclusion

5.1 Free-Form Visualization

In the following confusion matrix we get a good feeling for where the model did a good job classifying the categories and where it had problems. Throughout the set the results are quite solid, though we can observe that the model did mix up the categories in some cases.

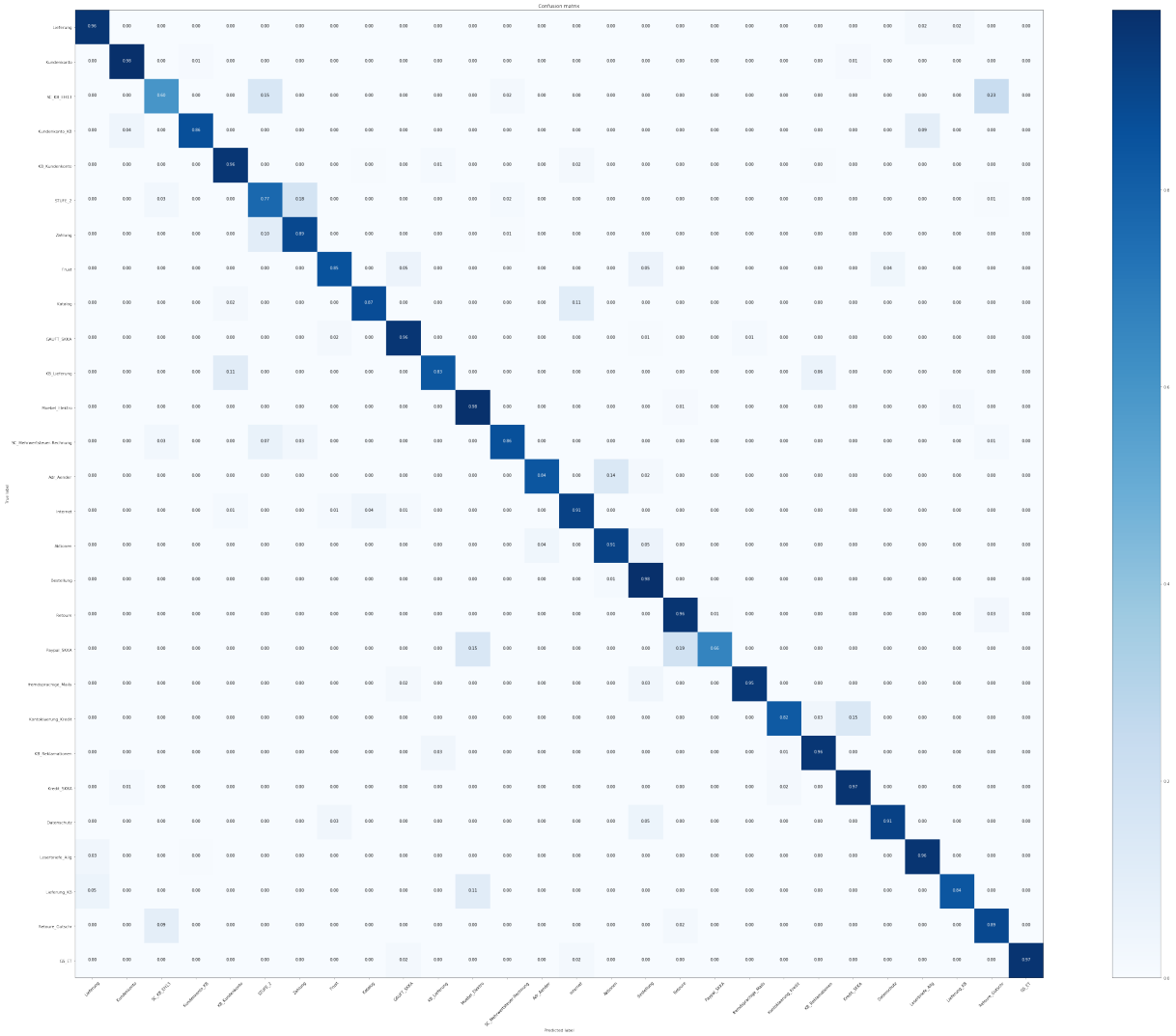

Figure 16: Confusion Matrix

5.2 Improvement

In this paper I described a Deep Learning approach to solve the problem defined in the introductory section 1.2 on page 2. The goal of this work was to present a proof that the concept of using a Deep Learning approach to solve the task of automatically classfifying emails into categories is not only feasible but gives us similar or even better results than a rule based method.
Training such a classifier is a challenging task and as we observed, the different kinds of RNN architectures with the word2vec embeddings were not able to get the desired results. Using the pre-trained model of BERT eventually enabled us to get solid classification scores. This gives us a hint that the data being used is one huge factor that determines the outcome of our model. In fact we believe that this is most likely the single most important factor that we will need to focus on for future development.
When taking a close look at the categories, one can observe that they are not well distinguishable by human classsification. For the trained service center employees it might seem quite intuitive but a third person will have a hard time making sense of it.
One example for this is the separation of "Kundenkonto_KB" and "KB_Kundenkonto", or "Lieferung" and "KB_Lieferung".
Optimizing these interdependencies was not the objective of this paper though, and initially we were aiming for a robust and easily trainable method that requires little to no manual intervention. Before taking any of the preprocessing steps described in section 3.1 on page 14, we removed all categories with less than 1000 emails where one was able to observe quite a drop to the next smallest number per category.
We believe that this step is justifiable through the argument that skewed datasets can make the training process vastly more difficult. The conclusion we draw here is that our current method can be greatly improved by defining a set of new upper level categories which can combine similar categories.
Another, possibly more simple way to achieve a comparable result would be to perform an unsupervised classification on the dataset beforehand and then using these new categories as our high-level classification targets. This can be done with several clustering algorithms. In both cases one could train a smaller classifier for more fine grain separation in each subcategory.
Another possibility for great improvement would be to drastically increase the size of our dataset. Currently we are using an email corpus that was extracted from one month of conversations. We believe that, in order to get a robust model one needs at least data from one year as there can be huge differences in the types of requests coming in during summer and christmas time for instance. Training a model on a dataset with emails from a couple years back though, might not be increasingly more helpful and could even make the performance worse because of changes in customer behaviour, product lines or even changes in category labelling.
The reasoning for why we only argue about improvements within the data domain is that we already get more than decent classifcation results regardless of the mentioned flaws our current dataset has. This gives us hope that our proposed method can work even for small to medium sized companies having a smaller email corpus to train on.

References

[1] Stella. Keenan. *Concise dictionary of library and information science /*. Bowker-Saur,, London ;, c1996.

[2] Paul Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78:1550 – 1560, 11 1990.

[3] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[4] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[5] Sepp Hochreiter and Jrgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[6] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, Nov 1997.

[7] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[11] François Chollet et al. Keras. `https://keras.io`, 2015.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.