Making a Manageable Email Experience with Deep Learning
Final Report

Louis Eugene
Department of Management Science and Engineering
Stanford University
leugene@stanford.edu

Isaac Caswell
Department of Computer Science
Stanford University
icaswell@stanford.edu

Abstract

This paper uses Deep Learning algorithms in the context of Natural Language Processing (NLP) and more specifically applies them to the task of prioritizing e-mails based on their importance and whether they require an answer. We base our results on different datasets constructed from our own e-mails and from the ENRON dataset. We use the Bag of unigrams and bigrams model with SVM and Random Forests as a baseline and then develop an LSTM network and a Convolutional Neural Network as a way to improve our results. The CNN performs yields the highest accuracy and F1 for all datasets except those labeled by Google's Importance label, in which case they are outperformed by Random forests, leading the authors to suspect a basis upon which Google assigns the 'important' label.

1 Introduction

Every minute, about 168 millions emails are sent in the world (for an amusing diagram see figure 1). But what was once our a true ally is now damaging to productivity and a source of stress: Email overload is suffered by 87% of employees, with 53% of them saying they cannot cope with the volume of email they have to deal with [1]. Emails arrive in a continuous stream and people have a hard time prioritizing them, having to go through 5 to 10 emails before finding one that requires their attention. Indeed, not all emails are equal [2]:

- 39% of email is information only (read only)
- 29% copied in unnecessarily (cc, reply-to-all)
- 17% irrelevant or untargeted (inc. spam)
- 15% action required

Within each category emails also have different priority, with some that require immediate action and some that can be dealt with later.

The goal of this project is manage email overload and give back precious minutes for people to spend more meaningful time. Our solution is mainly based on deep-learning algorithms to sort, prioritize and display email content efficiently. We identified 3 important stages relevant to e-mail management:

1. **Reception**:
   i. give an importance ranking to every incoming e-mail so that people can read important emails first and disregard the rest
   ii. categorize emails into the categories from the bulleted list above

Figure 1: A pictorial illustration o f the overload of technological input

iii. provide a summary of emails

2. **Action**: pin out in a dashboard the most important actions that are required in an efficient way, displaying relevant information such as deadlines mentioned therein. Integrate content directly where relevant, for instance into a calendar or company organizational platform
3. **Archiving**: save the information contained in emails so that it is easy to access and search when necessary

For the purpose of this project, we will focus only on Reception, and more specifically on ranking and categorizing emails.

2 Background/Related work

The concept of email overload is not a new one, and the literature provides a panoply of previous approaches to this task, largely focused on email classification. Several approaches even try simple neural networks. However, to the knowledge of the authors, no more advanced deep techniques (using for instance LSTM and CNN) have been put forward.

Because there is no standard of how to evaluate email ranking systems, and because results vary widely depending on the dataset (which one observes even among our various datasets), no effort is made in this section to report numerical findings of previous methods; we focus instead on the tasks that have been approached, and which algorithms each employed.

Classification by user folder has been approached with the use of SVM and Wide-margin Winnow [3], by activities with custom algorithms called SimSubset and SimContent [7], by topics [8] and later semantic non-parametric K-Means++ clustering [9]. These methods tend to make use of varying levels of custom features, including ones related to presence of dates/times, salutation, header fields and the presence of questions in the text.

Some people have tried to prioritize emails by categorizing the ones requiring an action. Corston-Oliver et al applied SVMs on a hand-collected dataset of 15k emails, [10]. Similarly, Dredze has worked on predicting if an email needs an answer [11], using logistic regression with hand-crafted features.

One other way to solve email overload is around providing a summary of those emails [4] or more generally from any text [6].

Finally, some people have stepped back of the field of machine learning to focus on the core functionalities that people use with emails [13] and provided innovative ways to solve email overload, such as executive assistant crowdsourcing [12].

3 Problem statement and Approach

i. **Prioritization/Ranking task**
Our first goal is to prioritize e-mails by order of importance. The difficulty of the task is due to the subjectivity of what it means to be important. Moreover, there is no labeled dataset giving importance of e-mails. Therefore, our first significant challenge was to construct a meaningful dataset. We describe our data in more detail in section 5.

One of the first difficulties is simply to define what it means to be important. For instance, one might imagine a whole list of things that a person might want so sort by:

(a) does the message require action?
(b) did the user reply to the incoming message?
(c) if yes, after how long?
(d) if yes, what is the length of the answer?
(e) did the user delete the email just after reading it?
(f) how long is the reply thread? (log thread length seems like a reasonable feature)
(g) does the email contain named entities? (an e-mail mentioning a place and a date or people is often more important)
(h) does the body contain keywords such as "important or "urgent"
(i) has gmail labeled this email as important?

To simplify the task at hand, we focus particularly on three categories from the above, namely (a), (b) and (i). The last of these may be viewed more as a measure of how well our predictions agree with those of Google, but also have meaning in their own right.

Once the rankings of emails is determined, we can use our categorization to mark emails with specific tags, such as "action required" for an e-mail that needs immediate response.

ii. **Summarization task**
The goal is to give a short one line summary of an email. We can base our method on *Improvement of Email Summarization Using Statistical Based Method* [4], which deals specifically with emails, or any other research or on text summarization, of which there are multitudes. (This is not the subject of this paper and will receive no further treatment, but is included for completeness, as this paper outlines a potential product.)

4 Technical Approach and Models

4.1 Features

Most of our work is based on the text content of an e-mail. We therefore present several approaches to represent a sequence of sentences as a convenient input for our different models.

A first baseline is to use Bag Of Words with unigrams and bigrams. The way it works is just counting occurrence of words in a text and returning an array where the ith index is the number of time word i appears in the text (or sequences of n words i appears in the text). This is used for our baselines (SVM and Random Forest), and is fairly representative of the basic approach in the industry.

For our deep approaches, we use dense word vectors in a high dimensional space. Word vectors can be fixed or updated at training time with the model; the latter is the approach we take. Their initialization could be random or pre-trained (such as with Word2Vec, or GloVe), both of which we test (see results).

Depending of the model we use, there are different ways to combine words vectors to represent a sentence as an input. The first naïve way is to average the vectors of every word in the sentence (not very effective). The most effective thing approach so far has been to keep a list containing the indices (and therefore complete word vectors) of the different words. This way we keep all the

information as well as the order of words, which is disregarded in the BOW approach. The specifics of this approach are model-specific; for instance, the CNN represents each sentence as a vector of the concatenated word vectors of the words in the sentence.

4.2 Algorithms

We compared a standard approach from the industry (as a baseline) with a variety of deep approaches:

1. *Baseline 1*
   Random Forest on unigram and bigram features, with 100 estimators, splitting on the Gini criterion. Implemented via sklearn.
2. *Baseline 2*
   Support Vector Machine on unigram and bigram features, using Gaussian kernel, with the penalty variable (C) at 100.0.
3. *(Algorithm 0: Standard Recurrent Neural Network)*
   A vanilla recurrent neural network. We used this mainly as a comparison for LSTM-RNN.
4. *Algorithm 1: Recurrent Long Short Term Memory (LSTM) network*
   LSTM are a special case of recurrent Neural Networks using gates. Those gates allow the model to incorporate long term dependencies and solve the problem of vanishing and exploding gradients commonly seen in RNNs. Our implementation of LSTM is based on [18] and [19].
5. *Algorithm 2: Convolutional Neural Network (CNN)*
   Our implementation of CNN is based on [20]. A sentence is represented as the concatenation of the different word vectors corresponding to the words of the sentence, padding the sentence with 0s as necessary in order to have a fixed sized input. The word vectors used are from Google word2vec.

5 Experiments and Results

1. **Data** Some innovation had to go into getting good data for this task. Good, labeled email datasets are hard to find, largely because of privacy concerns. One of the largest publicly available email dataset is the Enron Email Dataset[15], which contains about 600k emails from about 150 employees of Enron that were made public during the investigation of the company. The data, however, are noisy and unlabeled, and even EnronSent, [14] a subset specifically vetted for machine learning purposes, has no labels or metadata. Furthermore, some other publicly available datasets are too anonymized to be used; for instance, the RADAR dataset [16] has replaced all words with random strings of letters, making deep approaches relying on prelearned word representations unusable.
   As a result, we found one dataset, and made four more:
   (a) **Parakweet Intent dataset** [17]: A hand-labeled subset of the Enron dataset labeled for whether an email requires action.
   (b) **Isaac's Stanford Email Datasets** There are four versions of this dataset, each of which is drawn from 26k emails sent by Isaac Caswell from 2010 to 2015. There are two types of labels: whether an email was responded to, and whether Gmail marked the email as important. The former serves as a proxy for importance of an email; the second as a measure of how well our dataset agrees with Google's ranking. There are two versions of each of these, one of which contains only one email from each thread, and the other of which contains all emails sent. These four are referred to as *isaac-rep-1*, *isaac-imp-1*, *isaac-rep-all*, *isaac-imp-all*

   The Isaac datasets also have a more practical advantage, in that it is a personalized dataset, and has a direct application to our goal of a personalized message ranking system.
2. **Results**

5.1 Evaluation Metrics

Several metrics are applicable to our problem, and depending on the specifics of the product and the use-case, one may be preferable to another. In one use case, especially if the importance-ranking email client is effective enough and many emails are presented at once, a user may spend the most time looking at the emails that our system presents to them to see, and less time on unflagged emails. In such a scenario, recall is the most important; it's OK if they see a few emails that are not necessary, but is potentially very bad if an email they need to see is not represented. In different use cases and applications, however, the precision is more important. If the product is primarily focused on flagging unimportant emails, the precision is more important. As a compromise between these possibilities, we evaluate based on F1-score, as is standard in classification paradigms.

5.2 Analysis and Experimentation

We ran a battery of tests to determine the optimal configuration of our models. Following is an enumeration of different approaches we tried, with plots as appropriate.

(a) **Number of Epochs** We plotted the dev and training accuracy over several models. The result tended to be that the dev accuracy did not increase much beyond around ten iterations, while the training curve continued to overfit after that (See fig 2). With certain models, especially LSTM, dev accuracy tended to decrease after around ten epochs, demonstrating overfitting (See results: Figure 4).

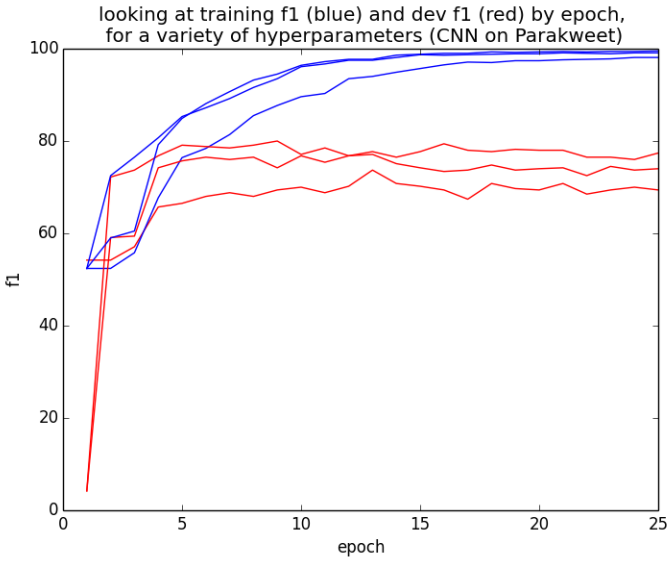


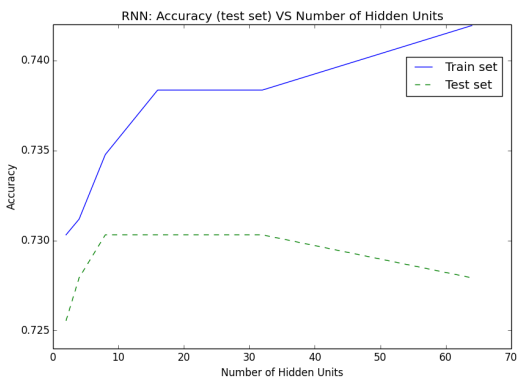
Figure 2: Dev accuracy stops increasing after O(5) epochs.

(b) **Dropout** Noticing the great overfitting of the training data (reference the graph on epochs), we experimented with and with dropout for a variety of models. Unfortunately, dropout tended to gain us at most 1% F1.

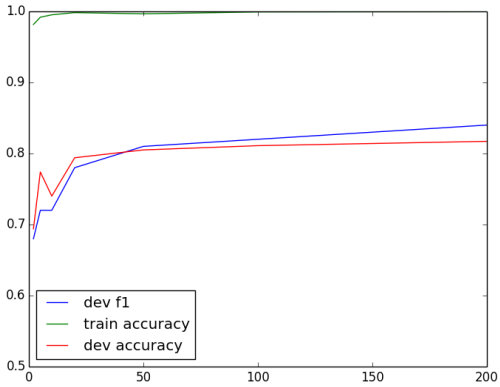
(c) **Hidden Dimension** Increasing the hidden dimensionality of our more advanced models always improved f1 over the range we tested, with a locally linear trend of 1%F1 per 50 hidden dimensions, from hidden dimensionality of 50 until 200 (see fig. 3). Because the runtime also increased dramatically for higher hidden dimensions, we tended run tests with lower hidden dimension.

It is interesting to note that in the more basic model (the vanilla recurrent neural network), the benefit of extra hidden dimensions decreases quickly, and leads to overfitting already at 30 hidden dimensions. (Figure 3)

(d) **Initialization of word vectors** We experimented with using Google Word2vec, random static and random nonstatic initializations to initialize word vectors for the CNN model.



(a) Vanilla RNN



(b) LSTM

Figure 3: [Parakweet data] Increasing hidden dimension on LSTM continues to increase F1 score, whereas it quickly overfits with vanilla RNN.

While word2vec converged much faster to start with (.76 vs .66 accuracy after the first epoch, for instance), after five epochs there was no discernible difference between the initializations.

(e) **Filter window sizes for CNN** Of the different filter window sizes for our convolutional nets we experimented with, smaller windows tended to work better, with sizes of [1,2,3] getting 84% maximum accuracy on Parakweet, compared with [3,4,5], which got only 81%. This could easily be a function of the fact that the dataset is not particularly large. The smaller window sizes are also much more efficient to run.

An important thing to note, however, is that the 84% is reached after around ten epochs, after which overfitting starts to happen, and accuracy drops to closer to 80%. This trend is especially clear in Figure 4.

(f) **clipping long sentences for efficiency** As an optimization, we filtered out all sentences longer than a certain length, often set at 50-100. Since actual sentences are rarely this long, this is in a sense a primitive form of data cleaning of anomalies that occur in the email data.

5.3 Performance

Our deep approaches outperformed the standard NLP approaches, represented by our baseline, with relative ease, tending to achieve 81-91% F1 depending on the dataset. These numbers are also exciting because we use no specialized features, e.g. header features and keywords. Interestingly, the dataset which we performed the best on was isaac-reply-allthreads, whose labels are whether an email has been replied to. (See fig 4)

A curious thing, however, is that for the isaac datasets labeled with Google's importance label, random forests outperform the CNN. This is especially curious because CNN wins handily with the exact same emails, labeled for whether they got a reply instead of Google importance ranking. It is hard to understand why this might be so, though it is possible that Google's internal importance ranking system uses techniques similar to random forests. It is possible then that these results mean something more to the effect of 'how similar is this approach to gmail's ranking system?' rather than 'How well do we predict importance on these data?'

To get a graphical understand by epoch, figure 4 demonstrates the performance of LSTM on our various datasets.

Table 1: Comparative results of different classifiers on different datasets

	Parakweet	Isaac-rep-1	Isaac-imp-1	Isaac-rep-all	Isaac-imp-all
RF	0.772	0.887	**0.826**	0.891	**0.904**
SVM	0.778	0.876	0.799	0.882	0.871
LSTM	0.809	0.854	0.801	0.900	0.879
CNN	**0.811**	**0.913**	0.817	**0.914**	0.900

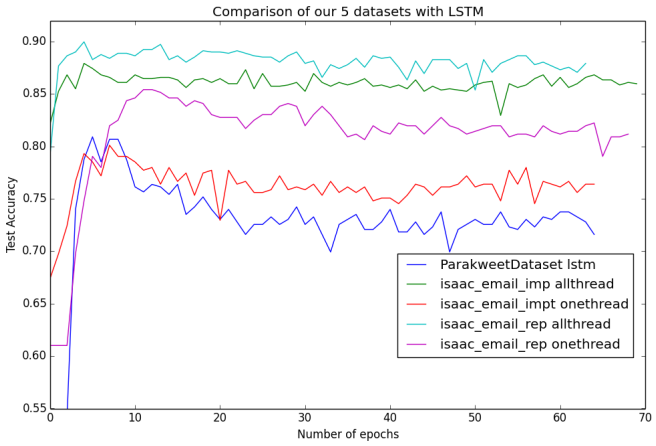


Figure 4: F1 of LSTM for our five datasets as a function of eopch. Note the spke around 10 epochs, before overitting hits

6 Conclusion

Based on our experiments, LSTMs and CNNs outperformed the baselines by a relatively high margin the baseline on the Parakweet dataset, which was our main point of reference. Our CNN seems to be our best working algorithm, with a F1 score of 84.0%, which is + 6.2% higher that the previous state of the art F1 on the Parakweet dataset: 77.8% (with LibSVM on feature vectors that combine n-grams and proprietary ActivityRelation signal). The main drawback of our CNN is its training time (up to multiple days, in comparison with a couple of minutes for our baseline SVM and tens of minutes for our LSTM).

Moving forward, we will implement a final algorithm: parsing sentences and using Recursive Neural networks. We will compare this algorithm to our LSTM and CNN. Then we will see if combining those algorithms together outperforms the best of them; for instance, we are looking into ensembles which weight predictions by the sigmoid of the confidences of each algorithm.

We also intend to find and make more diverse datasets to run our algorithms on, to test their generalizibility. We can can also extend our current dataset with labels wuch as"has the message been deleted or archived", and if the message has been replied to, "how long is it?" and "replied after how long?". Furthermore, combining datasets with diverse labeling schemes and running one single algorithm on them may lead to a more holistic measure of importance.

Our final step will be to combine those algorithms with a web client made by another Stanford student and see in practice if their important/non important labels make sense on a day to day basis.

References

[1] T. Jackson, E. Russell. (2015) Four email problems that even titans of tech havent resolved. Availible online at http://eprints.kingston.ac.uk/30433/1/four-email-problems-that-even-titans-of-tech-havent-resolved-37389

[2] T. Jackson. What to do with Employees that are too busy to manage their Email? Availible online at]http://www.proni.gov.uk/what_to_do_with_employees_that_are_too_busy_to_manage_their_email__tom_jackson.pdf

[3] R. Bekkerman. (2004). Automatic categorization of email into folders: Benchmark experiments on Enron and SRI corpora.

[4] M. I. Hashem (2014). Improvement of Email Summarization Using Statistical Based Method.

[5] S. Kiritchenko & S. Matwin, (2011, November). Email classification with co-training. In Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research (pp. 301-312). IBM Corp..

[6] X. Zhang & Y. LeCun. (2015). Text Understanding from Scratch. arXiv preprint arXiv:1502.01710.

[7] M. Dredze, T. Lau and N. Kushmerick. Automatically Classifying Emails into Activities

[8] G. Cselle, K. Albrecht and R. Wattenhofer. BuzzTrack: Topic Detection and Tracking in Email

[9] G. Soni and C.I. Ezeife.An Automatic Email Management Approach Using Data Mining

[10] Integration of Email and Task Lists S. Corston-Oliver, E. Ringger, M. Gamon and R. Campbell.

[11] M. Dredze, J. Blitzer, and F. Pereira. Reply Expectation Prediction for Email Management.

[12] N. Kokkalis, T. Köhn, C. Pfeiffer, D. Chornyi, M.S. Bernstein and S.R. Klemmer. Emailvalet: Managing Email Overload through Private, Accountable Crowdsourcing

[13] A.M. Szóstek. 'Dealing with My Emails': Latent user needs in email management. Needs benchmark by Agnieszka Matysiak Szóstek

[14] Styler, Will (2011). The EnronSent Corpus. Technical Report 01-2011, University of Colorado at Boulder Institute of Cognitive Science, Boulder, CO.

[15] B. Klimmt, Y. Yang. Introducing the Enron corpus. CEAS conference, 2004.

[16] P.N. Bennett & J.G Carbonell. Detecting Action-Items in E-mail. SIGIR 2005 Beyond Bag of Words Workshop. http://www.cs.cmu.edu/ pbennett/action-item-datset.html. 2005.

[17] EmailIntentDataSet from Parakweet Labs, Inc. https://github.com/ParakweetLabs/EmailIntentDataSet/wiki

[18] S. Hochreiter & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[19] F.A. Gers, J. Schmidhuber & F. Cummins (2000). Learning to forget: Continual prediction with LSTM. Neural computation, 12(10), 2451-2471

[20] Y. Kim. Convolutional Neural Networks for Sentence Classification. New York University