Neural Models for Email Response Prediction

Tucker Leavitt
Stanford University
tuckerl@cs.stanford.edu

Abstract

In this project, we apply several neural models to the problem of predicting whether a user will respond to an email, given the text of the email. We extract email response information from the Enron email dataset, and train three types of neural models: a two-layer LSTM, a CNN with max-pooling over time, and a LSTM with an MLP attention layer. Our best model achieves an F1 score of 87.2, outperforming our baselines and existing machine learning frameworks for the task.

1 Introduction

Despite its age, email remains one of the most popular online activities. Email is the primary communication channel for talking across organizational boundaries, though in many ways it is outdated. Products like Google Inbox are starting to bring intelligent recommendations and better organization to email. The advances in natural language processing over the past four years have great potential for applications that improve email.

Towards creating a more intelligent email experience, this project seeks to create a neural network for the task of **predicting whether the recipient of an email will reply or forward it**. Machine learning techniques have been applied to the email reply prediction problem in the past, but there has been little work done on applying recent advances in deep learning and neural networks to the problem.

2 Related Work

2.1 Email Reply Prediction

Here we present a brief survey of the existing literature on email reply prediction. Most studies use manually curated email features and used simple, "conventional" machine learning models to perform classification.

In 2008, researchers at the University of Pennsylvania built a logistic regression classifier for the tasks of email reply prediction and email attachment prediction [3]. They used sparse, high-dimensional email feature vectors with bag-of-words and relational features. The study used a dataset of 3000 hand-labeled emails, and their model achieved an F1 score of 0.67 for reply prediction and 0.66 for attachment prediction.

In 2017, researchers from Microsoft and UMass Amherst used a logistic regression classifier trained on curated domain-specific features to predict whether a recipient will reply to an email and how long it will take to do so [10]. Their features included a bag-of-words text model, individual and pair-wise historical interactions, the time of day at which the email was sent, and the job titles of the senders and recipients, among other features. They report an AUROC of 0.72 for reply prediction and a classification accuracy of 42% for predicting response time from using an AdaBoost ensemble model.

We have found one paper that applies neural models to the task of email reply prediction. This paper was a 2015 CS224D project, by Louis Eugene and Isaac Caswell [4]. They trained an LSTM and a Convolutional Neural Net

using a unigram and bigram bag-of-words model to predict "email importance," using Gmail's "importance" tag as the ground-truth label. They found that the deep learning models did not significantly outperform a Random Forest baseline, and their loss curves indicate that their models were overfitting. They achieved classification accuracies of 80-90%. Unfortunately, the fact that they do not use ground-truth labels makes it difficult to compare their results with existing methodologies.

2.2 Neural Models for Document Classification

Many different approaches to document classification using neural networks have been proposed over the past five years. In this study, we applied two different models to the problem of email reply prediction: heirarchical attention networks [11] and convolutional neural networks (CNNs) [7].

2.2.1 Heirarchical Attention Networks

Yang et al. [11] proposed a hierarchical recurrent neural network (RNN) architecture for modeling documents. Their architecture is composed of two RNNs: one for learning sentence embeddings from word vectors, and another for learning document embeddings from sentence embeddings. They use an attention mechanism at both the sentence and document level, which enables the model to focus on important content when constructing a document representation.

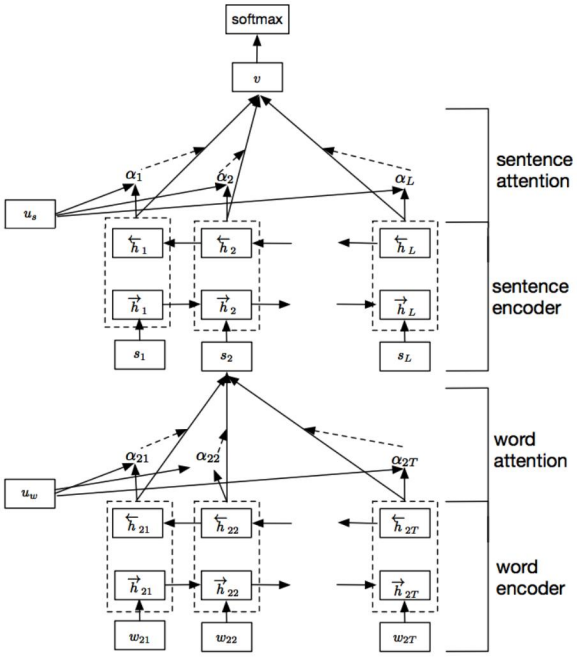


Figure 1: Overview of the heirarchical attention network, from [11].

They use a bi-directional GRU [2] as the underlying RNN. For each input word x_t, the RNN outputs a state vector h_t. Yang et al. proposed the following attention mechanism to construct a document representation on top of the state vectors:

$$u_t = \tanh(W_w h_t + b_w) \tag{1}$$

$$\alpha_t = \frac{\exp(u_t^T u)}{\sum_t \exp(u_t^T u)} \tag{2}$$

$$s = \sum_t \alpha_t h_t \tag{3}$$

That is, u_t is a hidden representation of the state h_t, and α_t is the normalized importance weight of the state h_t. s is the final document representation, computed as a weighted sum of the states h_t. The parameters W_w, b_w, and u are jointly learned during the training process. The vector s can be used as the input to a fully-connected softmax layer for classification, or as inputs to a higher-level RNN.

Yang et al. achieved state-of-the-art results on a variety of standard document classification datasets using this architecture.

2.2.2 Convolutional Neural Networks

Though traditionally used for computer vision, convolutional neural networks (CNNs) can also be effectively applied to natural language data. Kim [7] describes CNN architecture for sentence classification tasks, illustrated in figure 2.

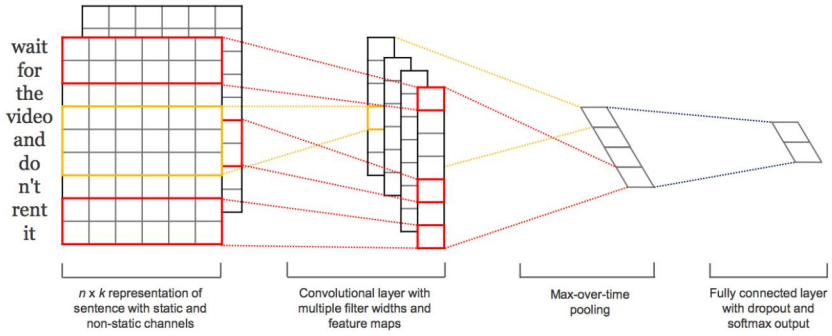


Figure 2: CNNs for sentence classification, using the max-pooling-over-time paradigm, from [7].

Sentences are represented by concatenating their constituent word vectors:

$$x_{1:n} = x_1 \,||\, x_2 \,||\, ... \,||\, x_n$$

for a sentence of length n, with $x_j \in \mathbb{R}^k$. The sentence vectors are convolved with a filter $w \in \mathbb{R}^h k$ to produce a new feature:

$$c_i = f(w \cdot x_{i:i+h-1} + b)$$

Where f is a nonlinear function such as the hyperbolic tangent. Note that the filter spans a window of h words, and we can apply this filter to each possible window of words to produce a set of features $(c_1, ..., c_m)$. Kim then applies a "max pooling over time" operation to produce a single feature $\hat{c} = \max(c_1, ..., c_m)$ for the given sentence; this extracts the "maximum activation" of the filter w over the entire sentence. The features $\{\hat{c}\}$ can be used as the input to a fully-connected softmax layer to produce the prediction. Typically, many filters of multiple different widths are applied in parallel. In addition, multiple "channels" of input vectors for each word may be used; Kim found that using one channel of static, pretrained word vectors and one channel that could be fine-tuned during backpropagation improves performance in most cases.

Kim found that CNN models perform comparably to many state-of-the-art models on a variety of document classification tasks, which is remarkable given the CNN's relative simplicity.

3 Approach

Inspired by [11] and [7], we implemented three different neural models for the email reply prediction task:

- a two-layer Long Short Term Memory (LSTM) RNN, referred to as **Vanilla LSTM**,
- a CNN with max-pooling over time (from [7]), referred to as **CNN**, and
- a two-layer LSTM with a MLP attention layer (inspired by [11]), referred to as **LSTM with Attention**.

3.1 Vanilla LSTM

The LSTM is a highly-popular recurrent neural network, originally described in 1997 [5]. At timestep t, it computes the output state h_t and cell state c_t from the input x_t and previous outpus h_{t-1}, c_{t-1} using a series of learned 'gates.' The gates can be interpreted as allowing the network to retain or forget information from previous time steps, and to expose part but not all of the current state.

LSTMs are often unreasonably good at modeling sequence data [6], and using one seemed like a natural choice for a first-attempt at a neural email reply predictor.

LSTMs can be stacked so that the output h_t of one is the input x_t of another. In this project we found that a two layer LSTM outperformed a single layer LSTM, but we found it difficult to train a three layer LSTM.

During training, an LSTM is typically "unrolled" so that the states for multiple inputs can be computed jointly in batches. This requires the input sequences $x_t^{(i)}$ to be padded to have equal length.

To perform prediction with the Vanilla LSTM, we fed the output of the second layer LSTM at the final timestep to a fully connected softmax layer that computed class probabilities.

3.2 CNN

We adopt the methodologies of Kim [7] as described in section 2.2.2. We opted to use two input channels: one with static word vectors, and the other with word vectors that we fine tuned during training.

3.3 LSTM with Attention

We implement a two-layer LSTM with an attention mechanism similar to Yang et al. [11] as described in section 2.2.1. Our model differs from theirs in two primary ways:
1. we only model documents on the word level; there is no sentence encoder, only a word encoder.
2. we use one directional LSTMs instead of bi-directional LSTMs

Given more time, it would have been interesting to explore how a full Heirarchical Attention Network would perform on this classification task; see Section 9 for more details.

4 Dataset

The dataset is derived from the Enron email dataset (`https://www.cs.cmu.edu/~enron/`), which contains over 1.5 million emails sent by 152 users at Enron. The dataset was originally made public by the Federal Energy Regulatory Commission during its investigation of the Enron bankruptcy scandal.

The Enron email dataset is not explicitly categorized into "no action taken" and "action taken" classes. However, "reply" and "forward" email chains appear as a single email example, and this can be used to categorize emails as "no action taken" and "action taken." We wrote a parse script that uses regular expression searches to detect whether an email contains a reply or forward chain. If so, the email extracts only the portion of the email after the "reply" or "forward" tag and labels the email as "action taken". Otherwise, the script takes the entire email text and labels it as "no action taken". The script also extracts several pieces of metadata about the emails, which may be used as additional features in the future. If the string parsing or tokenization of an example fails, the example is discarded.

(a) Class distribution

No Action	Action Taken	
	92,604 examples	
153,148 examples	Reply Sent	Forwarded
	18,721	73883

Total: 245,752 examples

(b) Distribution of email lengths, for 32,000 emails in the training dataset.

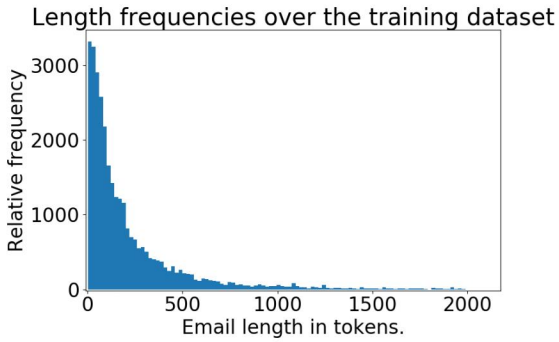


Figure 3: Dataset statistics

In the current project, only body text was used as a feature. Table Figure 3a shows the total number of examples and the class distribution of the dataset. This data was split into training, development, and testing sets in an 80%-10%-10% ratio.

Figure Figure 3b shows the distribution of the lengths of a subset of 32000 emails from the training dataset. In this case, the length of an email is the number of tokens in its body. Notice that the distribution is long-tailed. Half of the emails have a length of less than 119, and three quarters have a length less than 271. However, 512 email, or 1.6% of the subset, had a length of greater than 2000. This long-tailed behavior makes it more difficult to train sequence models, as the models need to be able to handle both very short and very long emails.

5 Experiments

We trained our models to minimize the cross entropy loss on the training dataset. We evaluated perdiodically on the development set during training. We report the classification scores on the testing dataset in the results section (6). Loss curves and other details can be found in the Supplementary Materials.

5.1 Preprocessing

The vocabulary for the model was taken to be the 10,000 most common tokens in the training dataset. We used pretrained GloVe embeddings [9] as word vectors; the GloVe vectors had dimension 50. We featurized words as a pair of word vectors: the first vector is the GloVe embedding for the case-normalized word, and the second vector is a randomly-initialized embedding for the word's casing. Word embeddings were fine tuned via back-propagation during training.

We tokenized the emails using the Python NLTK tokenization library [1]. Numbers were replaced with a special 'NUM' token, and words not in the vocabulary were replaced with an 'UNK' token. Sequences were padded with 'NIL' tokens to have length 300, and emails with lengths longer than 300 were truncated. This facilitated 'unrolling' the LSTM and allowed for efficient convolution computation in the CNN. The NIL token has a zero word vector and were masked out of the loss computation.

5.2 Evaluation methodology

To evaluate model performance, We compute the classification accuracy and multiclass precision, recall, and F1 metrics on a development dataset. The multiclass metrics are computed by evaluating the one-vs-many score for each class, and taking a weighted average of the scores. For example, the $F1$ score is computed as:

$$F1 = \sum_c w_c F1_c \tag{4}$$

Table 1: Model Performance Statistics, reported as percentages.

Model	Accuracy	Precision	F1 Score
Logistic Regression	58.7	66.5	62.4
Random Forest	80.5	80.9	80.7
Vanilla LSTM	82.9	83.4	83.0
CNN	82.8	85.4	83.2
LSTM w/ Attention	**87.2**	**87.7**	**87.3**
Yang et al. [10]	-	-	72.0
Dredze et al. [3]	-	73.0	67.0

where $F1_c$ is the $F1$ score for distinguishing class c from the other classes, and w_c is proportional to the number of instances of class c in the dataset.

5.3 Baseline models

We implemented a simple logistic regression classifier and a random forest classifier as a baseline to compare with our neural models against. We used a "bag-of-vectors" featurizer: for a given email, we used The random forest classifier was trained using 10 trees and by considering feature split sets of size 7. The logistic regression classifier used L2 regularization with a regularization strength of 1.

5.4 Model parameters

The Vanilla LSTM had a first hidden dimension of 100 and a second hidden dimension of 80. The CNN had filters of widths 3, 4, 5, 7, and 10, and had 100 instances of each filter. The the LSTM with Attention had a first hidden dimension of 50 and a second hidden dimension of 50. The context vector u for computing attention had dimension 100.

5.5 Hyper parameters

Training was run for 15 data epochs, using batches of 32 examples. The learning rate was initialized to 0.01 and linearly annealed to 0.001 during training. We used the Adam variant of stochastic gradient descent [8] to train the model via back-propagation. We used the default Adam parameters ($\beta_1 = 0.9$, $\beta_2 = 0.99$). We trained the Vanilla LSTM and LSTM with Attention for 55 hours on a CPU, and the CNN for 46 hours on a CPU [1]

The LSTMs were regularized by applying dropout to the input and state vectors. That is, at timestep t, the input x_t is replaced with $x'_t = m_1 \circ x_t$, and the previous state h_{t-1} is replaced with $h'_{t-1} = m_2 \circ h_{t-1}$, where m_1 and m_2 are elementwise boolean masks. We set the keep probability to 0.6 for both m_1 and m_2 during training, and 1 during validation. The CNN was regularized by applying dropout before the fully connected output layer. The keep probability was set to 0.5 during training and 1 during validation.

6 Results

Table 1 shows the accuracy, precision, and F1 scores[2] on the testing dataset, computed according to equation 4 .

The LSTM with Attention achieved the highest performance by all three metrics with an F1 score of 87.3. The CNN and Vanilla LSTM performed similarly, with F1 scores of 83.2 and 83.0. Note that the Random Forest baseline also performed very well, with an F1 score within 7 points of the LSTM with Attention.

[1] We ran into issues when trying to configure our Tensorflow graph to run on a GPU.

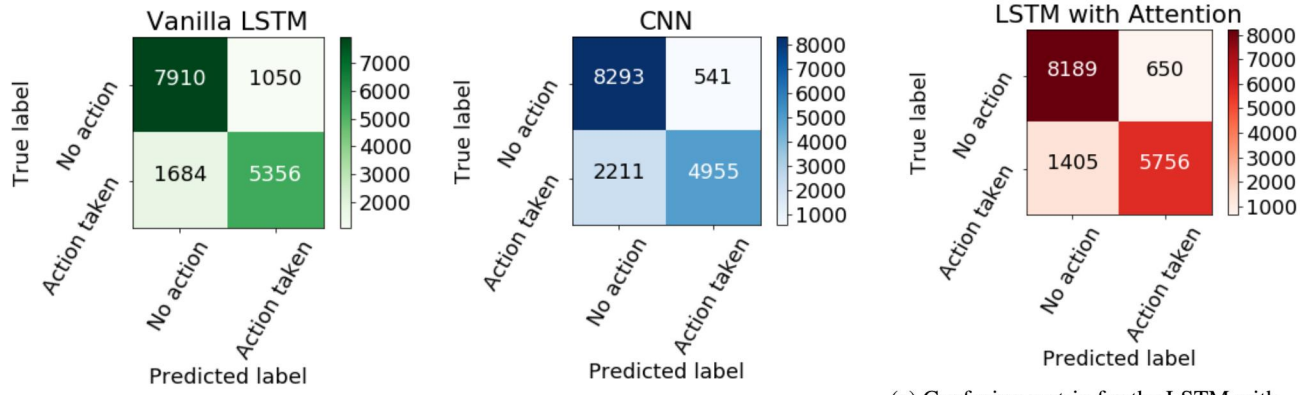
[2] the recall is not included since it is identically equal to the accuracy under 4

These scores are a significant improvement over previously reported scores for email reply prediction tasks: namely, the 0.72 AUROC score reported in [10] and the 67.0 F1 score reported in [3].

7 Analysis

7.1 Confusion Matrices

Figure Section 7.1 shows the confusion matrices for the Vanilla LSTM, the CNN, and the LSTM with Attention models. The confusion matrices show classification performance on a subset of 16,000 samples from the testing dataset. [3]



(a) Confusion matrix for the Vanilla LSTM.

(b) Confusion matrix for the CNN.

(c) Confusion matrix for the LSTM with Attention.

Figure 4: Confusion matrices for the three neural models.

Note that all three modes (and particularly the CNN) seem to over-predict the "no response" class.

7.2 Attention Visualizations

It is possible to visualize the attention weights of the LSTM on individual sentences, which can give insights into the words that the model is focusing on to make predictions.

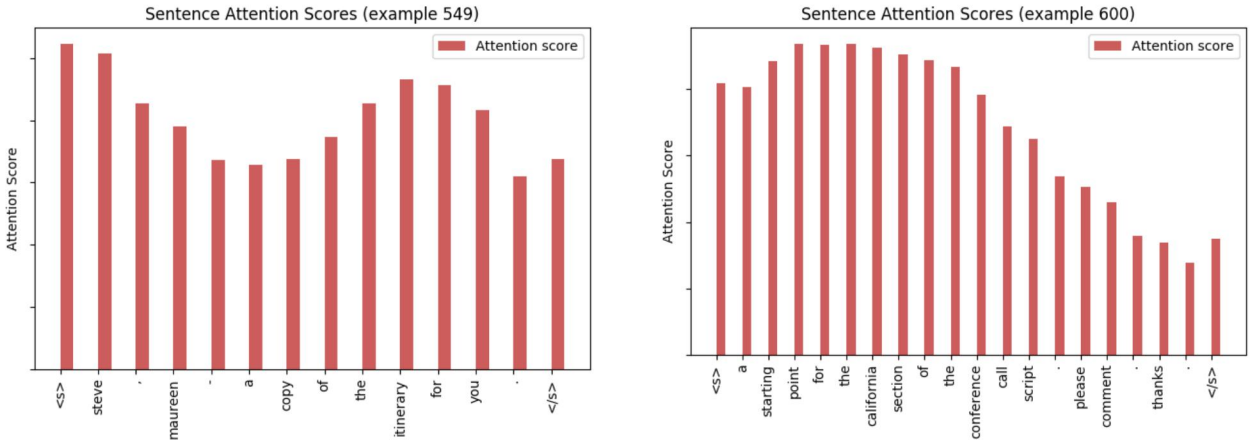
The model will often direct its attention to words that determine the meaning of the email, such as 'itenerary' in Figure 5a. However, the model's attention often seems to drop over time, as in Figure 5b. Here, the model misses the important phrase "please comment" and misclassifies the example. For more examples from the dataset and qualitative performance evaluations, see the Supplementary Materials.

8 Conclusion

We were able to build three neural models that significantly outperformed existing machine learning methods on the email reply prediction task. Of the models we tested, we found that a two-layer LSTM with an MLP attention layer performed best. The attention layer seemed to improve performance significantly over the Vanilla LSTM.

The neural models we built performed only marginally better than the bag-of-vectors Random Forest baseline. This indicates that much of the information in the email body relevant to predicting email response is captured by the body's set of word vectors.

[3] each model was tested on a different subset of the testing dataset when computing the confusion matrices.



(a) True label: No action.
Predicted label: No action

(b) True label: Action taken.
Predicted label: No action.

Figure 5: Relative attention distributions of an examples from the development set.

For a simple binary classification problem, the email reply prediction problem has proven to be quite difficult. This may be in part due to the large amount of variance in the type, length, and content of the emails. Additionally, whether or not a person responds to an email depends on several factors external to the email text itself: the subject line, the recipient's relationship with the sender, the other people involved in the message, and the time of day, week, or year that the email is sent, among other things. Furthermore, email response behavior may vary significantly from person to person, so incorporating information about a particular user's habits and behaviors may help boost performance.

We suspect that the models trained in this paper have overfit to the training dataset. More work and fine-tuning would be needed to extend these models outside of the Enron dataset.

9 Discussion and Future Work

Currently, both 'forwarded' and 'replied' emails are grouped under the same label; a natural next step would be to split them and repeat the training process. Given a richer dataset, it might also be possible to introduce classes like "not opened" or "archived."

One approach that may have improved training time and performance is to organize the training batches to contain input sequences of similar length. This could have reduced variance of the gradients between batches and may have helped the model train better.

Additional dataset preprocessing and filtering may also improve performance; for instance, the model could be composed with a spam filter to remove some additional emails; these email often contain many UNK tokens, making them more difficult to handle correctly (see the Supplementary Materials for examples.)

In figure 5b, the model attention decreases over time; this was not an uncommon behavior. Implementing a full Hierarchical Attention Network as described in [11] may improve performance. In particular, using a bidirectional RNN instead of a uni-directional RNN may help the model pay more attention to words that occur later in the email.

It would be interesting to extend this model to a dataset of personal email communications, instead of work emails. The email content and response behaviors may be markedly different.