Large Scale Multi-label Text Classification with Semantic Word Vectors

Mark J. Berger
Department of Computer Science
Stanford University
Stanford, CA 94305
mjberger@stanford.edu

Abstract

Multi-label text classification has been applied to a multitude of tasks, including document indexing, tag suggestion, and sentiment classification. However, many of these methods disregard word order, opting to use bag-of-words models or TF-IDF weighting to create document vectors. With the advent of powerful semantic embeddings, such as word2vec and GloVe, we explore how word embeddings and word order can be used to improve multi-label learning. Specifically, we explore how both a convolutional neural network (CNN) and a recurrent network with a gated recurrent unit (GRU) can independently be used with pre-trained word2vec embeddings to solve a large scale multi-label text classification problem. On a data set of over two million documents and 1000 potential labels, we demonstrate that both a CNN and a GRU provide substantial improvement over a Binary Relevance model with a bag-of-words representation.

1 Introduction

In recent years, neural network models have achieved remarkable performance in a variety of NLP tasks, including part-of-speech tagging (Collobert et al. 2011), language modeling (Mikolov et al. 2010, 2011), and sentiment classification (Socher et al. 2013). In addition to more powerful computer architecture and more sophisticated training algorithms, pretrained semantic word embeddings have empowered these models to achieve state of the art performance. However, to the best of our knowledge, semantic word vectors have not been used in the field of multi-label text classification. Rather, multi-label learning algorithms use a bag-of-words representation or TF-IDF weighting to model text, which disregards word order and can significantly limit the size of the vocabulary used during training. In an attempt to overcome these shortcomings, we explore the use of word embeddings in both a convolutional neural network (CNN), and a recurrent neural network model with a gated recurrent unit (GRU), to label documents with a significantly large label set. Despite a lack of extensive hyper-parameter tuning, we demonstrate that both a CNN model and a GRU model exceed the performance of the popular Binary Relevance method, when using a bag-of-words model.

2 Related Work

2.1 Problem Overview

First, we briefly describe the problem of multi-label learning and some standard approaches that are used to solve multi-label classification problems. Broadly, multi-label classification is the task of assigning a set of labels from a fixed vocabulary to an instance of data. For multi-label text classification, this often involves labeling a piece of text with a set of tags. Since each document has an indeterminate number of labels, the task is significantly harder than multiclass classification.

Where as multiclass classification has $|C|$ potential choices, a multi-label classification problem has $2^{|L|}$ possible outputs. Additionally, due to the high dimensionality of the data, labels are often correlated, and labels are likely to be unbalanced (Gibaja and Ventura, 2015). Therefore, an ideal algorithm would utilize correlated labels to make predictions, while attempting to avoid simply learning label frequencies.

Algorithms to address multi-label learning can be roughly categorized into three different types: multi-label classification, label ranking, and multi-label ranking (Gibaja and Ventura, 2015). Multi-label classification algorithms take an input X and produce the bipartitioned set Y and $\bar{Y}$, where Y is the set of relevant labels, and $\bar{Y}$ is the complement of Y. In contrast, label ranking does not determine a partition, but rather orders the labels from relevant to irrelevant. Specifically, given an input X, a label ranking algorithm produces an array of length $|L|$, where a lower index indicates a higher relevance. Multi-label ranking is seen as a combination of the two, because it produces a ranking of the labels, as well as a bipartition. Namely, the algorithm aims to rank labels in Y higher than all the labels in $\bar{Y}$. A threshold function is then learned, or manually set, in order to determine how many labels in the sorted array are pertinent. We now present two applicable algorithms, one which is a multi-label classification algorithm, and the other, a multi-label ranking algorithms.

2.2 Binary Relevance

In practice, the Binary Relevance (BR) method is a popular means of addressing the multi-label learning problem. In this method, the problem is transformed into a set of $|L|$ binary classification problems, where each problem corresponds to a specific label. A separate classifier is trained for each label, and the given document is evaluated with every classifier to determine which sets of labels apply to the text. The final outputs are then merged into a vector to produce a single output. By training $|L|$ classifiers, the method is naturally parallelizable, and scales linearly, often leading to a low computational complexity (Read et al. 2011). Additionally, labels can be added and removed from the set without having to retrain the entire set of classifiers. However, transforming the problem also creates a set of disadvantages (Zhou et al. 2010). First, Binary Relevance assumes that labels are independent, which is often a false assumption. By discarding label correlation, BR often fails to predict different combinations or labels, leading to decreased performance. Data set imbalance can also have a larger effect on the classifiers as well, because negative examples will tend to outnumber positive examples. Finally, BR may train an unnecessary number of classifiers. Certain labels may only appear with other labels, which makes training a separate classifier inefficient. Similarly, some labels may be incredibly infrequent, and allocating an equal number of parameters for frequent and infrequent labels can be costly. It is with these disadvantages in mind, that we explore the use of neural networks in multi-label classification.

2.3 Feedfoward Networks

However, this is not the first instance in which neural networks have been used for the task at hand. Feedforward neural networks have a well established history in the field of multi-label learning, and were first introduced by Zhang et al. (2006), with their Backpropagation for Multi-Label Learning algorithm (BP-MLL). BP-MLL consists of a conventional feedfoward neural network which takes in a fixed input of dimensionality d and produces a ranked output of size $|L|$. The researchers then train a function to predict how many of the top labels should be assigned to the input. The network itself is trained using backpropagation, but Zhang et al. introduced a novel error function, which was designed to account for correlation between pairs of labels:

$$E = \sum_{i=1}^{m} \frac{1}{|Y_i||\bar{Y}_i|} \sum_{(k,l)\in Y_i \times \bar{Y}_i} \exp(-(c_k^i - c_l^i))$$

where Y_i is the set of labels associated with example i, $\bar{Y}_i$ is the complement of Y_i, and c_k^i is the output for label k on instance i. Therefore, if the difference between a correct label and an incorrect label is large, the error will be smaller. With this new cost function, the BP-MLL model achieved state of the art results on both functional genomics and text classification data sets.

With the advent of more powerful training and regularization methods, Nam et al. (2014) decided to revisit the feedfoward network introduced by Zhang et al. (2006). They found by applying Adagrad (Duchi et al. 2011), dropout (Hinton et al. 2012), and a rectified linear unit, they were able to achieve state of the art results on multiple academic text-classification data sets. Additionally, they found that their model achieved better results when using a standard cross-entropy error function instead of the error function from Zhang et al. (2006).

However, both of these methods failed to account for the complete nature of textual information. Since feedfoward networks only accept an input of size d, document vectors were constructed by using bag-of-words representations, or filtering the vocabulary by document frequency. Both models do not preserve word order, and filtering by document frequency can significantly limit the size of the vocabulary. Specifically, Zhang et al. limited their vocabulary to the top two percent of words with the highest document frequency. We hope to account for both of these shortcomings by preserving word order and a significantly larger percent of the vocabulary with the use of semantic word embeddings. How we intend to do this is described in detail within the next two sections.

3 Convolutional Neural Network

For our experiments we explore the use of a convolutional neural network (CNN) model, which has had a dramatic impact on the computer vision community. We use a slight variation of the CNN architecture presented by Kim (2014), and we briefly describe it here. In order to represent a given document, we construct an n by k matrix, where n is the number of words and k is the dimension of the word vectors. For each row in the matrix, row i contains the word vector for the ith word in the document. In order to conduct batch training, we zero-pad or shorten the documents to a fixed length of size n.

After constructing our matrix, we conduct a convolutional operation over each document's respective matrix. Specifically, we apply a weight matrix (known as a filter) $W \in \mathbb{R}^{hk}$ to a window of size h by k, where h is the length of an n-gram, to produce a scalar c_i. Specifically, this is calculated by:

$$c_i = f(Wx_{i:i+h-1} + b^{(c)})$$

where b is the bias term and f is a chosen non-linearity. During convolution, we apply this filter over all possible $N - h + 1$ windows to produce the set of features $\{c_1, c_2, ..., c_{N-h+1}\}$. We then apply a max-over-time pooling operation (Collobert et al. 2011), to the set of $N - h + 1$ features to produce the feature $\hat{c}$.

$$\hat{c} = \max(c_1, c_2, ..., c_{N-h+1})$$

Therefore, we extract a single feature from the set of features produced by the filter. We repeat this same process for multiple feature maps of size h by k, as well as for filters with different sizes of h. We then concatenate all of these features into a single vector, creating the vector c', and supply c' to our fully connected layer. In order to prevent feature adaption, we apply the popular dropout method with a probability of p to c'.

We then add a fully connected output layer of size $|L|$ with a sigmoid activation function, which produces a probability for each of our potential labels.

$$\hat{y} = \sigma(Uc' + b^{(o)})$$

During training, these probabilities are used to compute the error, while during testing, we round each of these labels to 0 or 1 depending upon a set threshold. Unlike Kim (2014), we keep our word embeddings static throughout our experiments.

4 Gated Recurrent Neural Network

We now briefly describe the recurrent network with a gated recurrent unit model (GRU) introduced by Cho, et al. (2014). Similar to a recurrent network, a GRU uses a set of weights to produce a

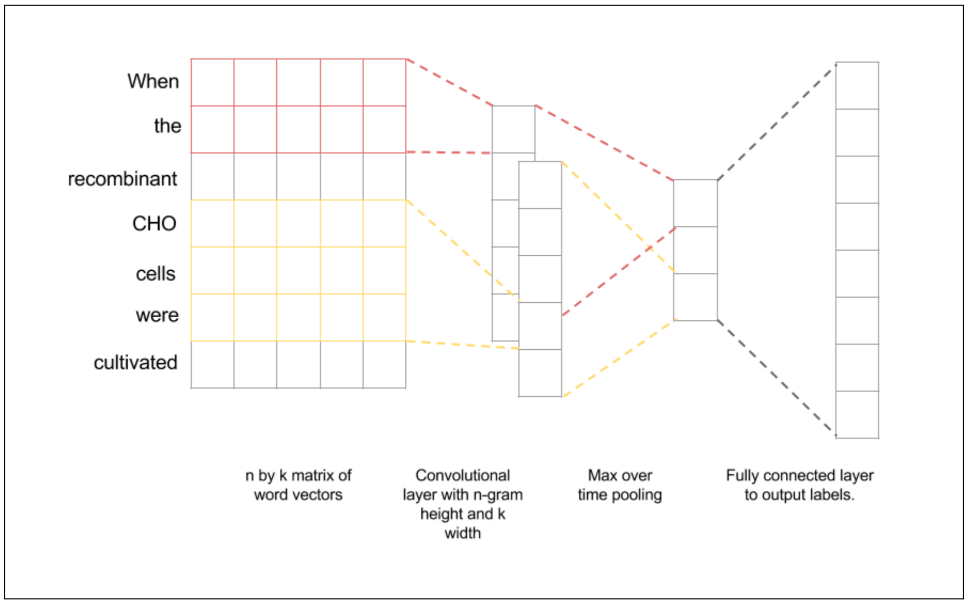
When
the
recombinant
CHO
cells
were
cultivated
n by k matrix of word vectors
Convolutional layer with n-gram height and k width
Max over time pooling
Fully connected layer to output labels.

Figure 1: Our adaptation of the model presented by Kim, 2014 for multi-label classification. Inputs are zero-padded to have a token length of n, and each word vector is k dimensional.

hidden vector h. Then over a series of time-steps, it uses the same set of weights to combine the input at time step t with the previous hidden vector, h_{t-1}. However, unlike a standard RNN, a GRU adds two ”gates”, an update gate, z, and a reset gate, r:

$$z = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

$$r = \sigma(W_r x_t + U_r h_{h-1} + b_r)$$

We then use the reset gate to control how much of the previous state we should carry over to the new candidate state, which we define as:

$$\widetilde{h} = \text{tanh}(W x_t + r \odot U h_{t-1} + b)$$

Then finally, we use the update gate to calculate a weighted average between the previous hidden state, and the new candidate state:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \widetilde{h}$$

With these gates, the model is then able to learn when it should forget, or retrain certain information about the sequence. For our model, we apply the GRU to our documents, with the word embeddings as input at time step t, which ultimately produces a final hidden output layer h_f. We then supply this vector to a fully connected layer, with an output size of $|L|$ to produce the final output $\hat{y}$, where the element at index i represents the probability of label i being associated with the given document.

$$\hat{y} = \sigma(W^{(o)} h_t + b^{(o)})$$

Similar to the CNN, we use $\hat{y}$ to calculate the cost of our models during training. During testing, we round the probabilities to 0 or 1 via a threshold t in order to indicate whether label i should be applied to the document.

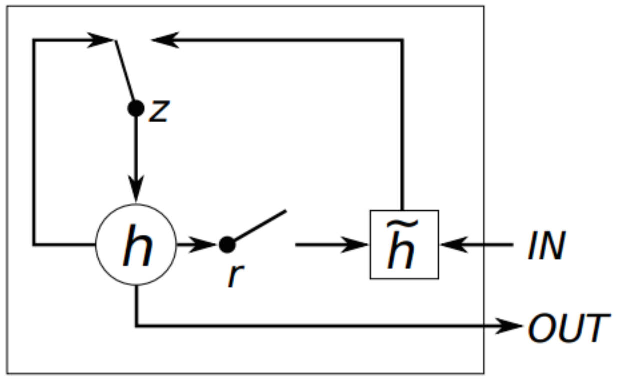

Figure 2: A recurrent neural network with a gated recurrent unit (Chung et al. 2014)

5 Data Set and Experimental Setup

5.1 Data Set

To explore this problem, we use data from the BioASQ Challenge for Large-Scale Biomedical Semantic Indexing from 2014 (Tsatsaronis et al. 2014). Specifically, we use the second version of the data set, which contains roughly 4.5 million biomedical abstracts from PubMed, a publicly available database maintained by the United States National Library of Medicine and the National Institute of Health. Each abstract is assigned a variety of terms from the Medical Subject Headings (MeSH) thesaurus, a hierarchical vocabulary of roughly 27,000 terms. Due to the limited architecture available to the investigator, we restrict the scope of our study to the 1000 most frequent labels. Furthermore, because all of the articles in the test set are from 2014, we removed all documents that were published before the year 2000. This is done because the MeSH thesaurus changes every year, and some articles in the training data span back to the 1940s. After filtering for publication year, we were left with roughly two million documents in our data set. Since no validation set is given, we randomly selected 100,000 articles in the training data to be used as a validation set. For the test set, we use the documents from the first test batch from the 2014 competition. Finally, for our word vectors, we choose to use the 200-dimensional word2vec embeddings provided by the competition organizers, which were trained on the PubMed database.

5.2 Metrics

We compare our models using the micro-F1 measure, which is defined as the harmonic mean of the micro-precision and the micro-recall of all of the labels:

$$P_\mu = \frac{T_P}{T_P + F_P}$$
$$R_\mu = \frac{T_P}{T_P + F_N}$$
$$F1_\mu = \frac{P_\mu \cdot R_\mu}{P_\mu + R_\mu}$$

Where T_P is the number of true positive labels, F_P is the number of false positive labels, and F_N is the number of false negative labels.

5.3 Models

As a baseline, we use the Binary Relevance method with bag-of-words features. We train a set of 1000 linear classifiers, one for each label, with the popular scikit-learn library (Pedregosa et al.

Training Set Size	2033549
Vocab Set Size	3274079
Label Set Size	1000
Median Number Tokens	220
Median Number Labels	8

Table 1: Statistics about our filtered BioASQ data set

		Validation Set			Test Set	
Model	Precision	Recall	F1	Precision	Recall	F1
Baseline	$0.721 \pm .001$	$0.260 \pm .001$	$0.383 \pm .001$	$0.711 \pm .001$	$0.294 \pm .001$	$0.416 \pm .001$
CNN	$0.798 \pm .001$	$0.329 \pm .001$	$0.466 \pm .001$	$0.802 \pm .001$	$0.354 \pm .001$	$0.491 \pm .001$
GRU	1.0	$0.447 \pm .001$	$0.617 \pm .001$	1.0	$0.471 \pm .001$	$0.640 \pm .001$

Table 2: Micro F1 results from our experiments

2011) using hinge loss, L2 regularization, and an annealing learning rate, with an initial learning rate of 1. For feature selection, we trained on three different sets of n-grams: unigrams, unigrams and bigrams, and unigrams through trigrams, and selected the best set after evaluating them on the validation set. Our best model was a set of linear classifiers trained solely on unigrams.

For the CNN, we train five different filters, each corresponding to a unigram through 5-gram filter height, where each filter produces 100 feature maps. Each filter is initialized using LeCun uniform initialization (LeCun et al. 2012) and we use tanh as our non-linearity, with a dropout rate of 0.5. In order to conduct batch training and conduct max over time pooling, we zero-pad or truncate the input data to 300 tokens. We trained our model using Adadelta (Zeiler 2012), with $\alpha = 1.0$ and $\rho = .95$, and a batch size of 64. Our model was built using the theano library (Bergstra, James, et al. 2010) and we used a manually set threshold of 0.5 for predicting labels.

For the GRU, we use a hidden size of 200, and we initialize the weight matrices W with Glorot uniform initialization (Glorot et al. 2010), while we initialize the U matrices with the normal distribution $N(0, .01)$ (Pascanu et al. 2012). To train our model, we used RmsProp (Tieleman, Hinton 2012) with $\alpha = .001$ and $\rho = .95$, and we soft clip the gradient norm to 10. Similarly, we also use a batch size of 64 and zero-pad the beginning of documents if necessary. For this model, we used the open source library Keras[1] and manually set the prediction threshold to 0.5.

6 Results and Analysis

On both the validation and the test sets, the CNN and the GRU provided a significant increase in micro-F1 score. Specifically, on the held out test set, the CNN and GRU provide an 18% and 53% increase, respectfully. While dropout increases our performance on the CNN model, we saw little to no difference with the GRU, and therefore list the results of a GRU without any regularization. We attribute the lack of improvement to the relatively small size of the hidden layer and the large number of diverse training examples. While we attempted to experiment with the size of the hidden layer with the GRU, we were unable to do so given the time span of the project. The model took roughly three days to converge on a high performance GPU from Amazon Web Services, and we experienced memory limitations when attempting to increase the size of the hidden layer. In contrast, the CNN model was able to converge in less than a day after training for many more epochs. However, with the GRU's 30% higher micro-F1 measure, it is unlikely that a shorter training time would warrant using a CNN instead of a GRU.

7 Conclusion & Future Work

Overall, we demonstrate that both a CNN and a GRU using semantic word embeddings significantly outperform the Binary Relevance method with bag-of-words features on a large scale multi-label

[1]https://github.com/fchollet/keras

classification problem. While we produced positive results, a significant amount of work is needed in order to scale the model to run at the scale of the PubMed database. Namely, we encountered multiple memory limitations which would make it difficult to use the same models to make predictions from a set of roughly 27,000 labels on a single machine. Additionally, using a sigmoid function on the output layer of our models prevented us from using a rectified linear unit as our activation function, which has been shown to significantly improve neural network models (Nam et al. 2014). Therefore, in order to take advantage of this activation function, we hope to explore multi-label ranking methods in the future, as was done by Zhang et al. (2006) and Nam et al. (2014) with feedfoward networks. Finally, using a model to learn an optimal threshold for rounding the probabilities and backpropagating into our word vectors may also lead to increased performance.

References

Bergstra, James, et al. ”Theano: a CPU and GPU math expression compiler.” Proceedings of the Python for scientific computing conference (SciPy). Vol. 4. 2010.

Chung, Junyoung, et al. ”Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.” arXiv preprint arXiv:1412.3555 (2014).

Collobert, Ronan, et al. ”Natural language processing (almost) from scratch.” The Journal of Machine Learning Research 12 (2011): 2493-2537.

Duchi, John, Elad Hazan, and Yoram Singer. ”Adaptive subgradient methods for online learning and stochastic optimization.” The Journal of Machine Learning Research 12 (2011): 2121-2159.

Gibaja, Eva, and Sebastin Ventura. ”A Tutorial on Multilabel Learning.” ACM Computing Surveys (CSUR) 47.3 (2015): 52.

Glorot, Xavier, and Yoshua Bengio. ”Understanding the difficulty of training deep feedforward neural networks.” International conference on artificial intelligence and statistics. 2010.

Hinton, Geoffrey E., et al. ”Improving neural networks by preventing co-adaptation of feature detectors.” arXiv preprint arXiv:1207.0580 (2012).

Kim, Yoon. ”Convolutional neural networks for sentence classification.” arXiv preprint arXiv:1408.5882 (2014).

LeCun, Yann A., et al. ”Efficient backprop.” Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.

Mikolov, Tomas, et al. ”Recurrent neural network based language model.” INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010. 2010.

Mikolov, Tomas, et al. ”Extensions of recurrent neural network language model.” Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on. IEEE, 2011.

Mikolov, Tomas, et al. ”Efficient estimation of word representations in vector space.” arXiv preprint arXiv:1301.3781 (2013).

Nam, Jinseok, et al. ”Large-scale Multi-label Text ClassificationRevisiting Neural Networks.” Machine Learning and Knowledge Discovery in Databases. Springer Berlin Heidelberg, 2014. 437-452.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. ”On the difficulty of training recurrent neural networks.” arXiv preprint arXiv:1211.5063 (2012).

Pedregosa, Fabian, et al. ”Scikit-learn: Machine learning in Python.” The Journal of Machine Learning Research 12 (2011): 2825-2830.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. ”Glove: Global vectors for word representation.” Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014) 12 (2014).

Read, Jesse. ”Advances in Multi-label Classification” 2011. Retrieved from http://users.ics.aalto.fi/jesse/talks/ Charla-Malaga.pdf.

Socher, Richard, et al. ”Recursive deep models for semantic compositionality over a sentiment treebank.” Proceedings of the conference on empirical methods in natural language processing (EMNLP). Vol. 1631. 2013.

T Tieleman and G Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Tsatsaronis, George, et al. ”An overview of the BIOASQ large-scale biomedical semantic indexing and question answering competition.” BMC bioinformatics 16.1 (2015): 138.

Tsoumakas, Grigorios, Ioannis Katakis, and Ioannis Vlahavas. ”Mining multi-label data.” Data mining and knowledge discovery handbook. Springer US, 2010. 667-685.

Zeiler, Matthew D. ”ADADELTA: an adaptive learning rate method.” arXiv preprint arXiv:1212.5701 (2012).

Zhou, Tianyi, Dacheng Tao, and Xindong Wu. ”Compressed labeling on distilled labelsets for multi-label learning.” Machine Learning 88.1-2 (2012): 69-126.