Large Scale Taxonomy Classification using BiLSTM with Self-Attention

Hang Gao
University of Maryland Baltimore County
Baltimore, Maryland
hanggao1@umbc.edu

Tim Oates
University of Maryland Baltimore County
Baltimore, Maryland
oates@cs.umbc.edu

ABSTRACT

In this paper we present a deep learning model for the task of large scale taxonomy classification, where the model is expected to predict the corresponding category ID path given a product title. The proposed approach relies on a Bidirectional Long Short Term Memory Network (BiLSTM) to capture the context information for each word, followed by a multi-head attention model to aggregate useful information from these words as the final representation of the product title. Our model adopts an end-to-end architecture that does not rely on any hand-craft features, and is regulated by various techniques.

KEYWORDS

taxonomy classification, BiLSTM, attention

ACM Reference Format:
Hang Gao and Tim Oates. 2018. Large Scale Taxonomy Classification using BiLSTM with Self-Attention. In *Proceedings of ACM SIGIR Workshop on eCommerce (SIGIR 2018 eCom Data Challenge)*. ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The cataloging of product listing through taxonomy categorization is a popular research area for many e-commerce marketplace. The task is challenging due to various reasons, for example, the lack of read data from actual commercial product catalogs, the noisy nature of product labels and the typical unbalanced data distribution.

In this paper, we present an end-to-end neural network based system for taxonomy classification. The proposed approach employs a BiLSTM network augmented with a multi-head self attention mechanism, producing a feature representation used for classification. We also regulate the system with various regulation techniques in order to obtain better generalization.

2 OVERVIEW

Our approach consists of two main steps: (1) the **sampling step**, where we over-sample instances of rare categories with augmentation; (2) the **network step**, which includes three parts: a recurrent neural network to generate word representation with context information; a self-attention network to generate a distribution over the enriched word representations and a classifier that performs classification.

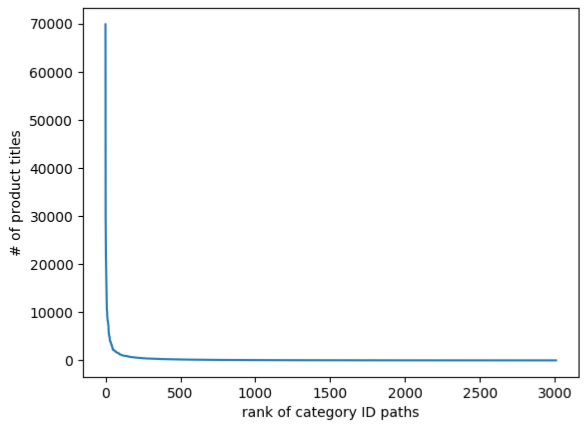


Figure 1: # of product titles w.r.t. the rank of their corresponding category ID path. The ranks are generated as the indices of those category ID paths sorted by the # of product titles.

Task Definition. The task is to predict the category ID path given a product title, as shown in Table 1, which includes examples taken from the data challenge description page. Evaluation of a system in this task is measured by weighted-precision, recall, F1 score with complete matching.

The task adopts the data released by Rakuten, which includes 1M product listings in tsv format, split into train/test set with ratio 80%/20%. The train set includes 3008 category ID paths and is highly imbalanced. In Figure 1, we show a comparison of the number of product titles among these category ID paths.

2.1 Sampling

As mentioned above, since the data is highly imbalanced, it is often important to over-sample instances of rare classes to force a model to pay more attention to them, instead of overwhelmed by those frequent ones. A common strategy of over-sampling is replication, but in our system, we adopts a different version where we augment the replicated samples to prevent our system from simply remembering them in order to get better generalization.

Given an sample, we first randomly replicate it by d times, where d is randomly picked from the set $\{1, 2, 3, ..., D\}$; we then concatenate these replicas together and split them into a bag of words,

Table 1: Examples of product titles and their corresponding category ID paths.

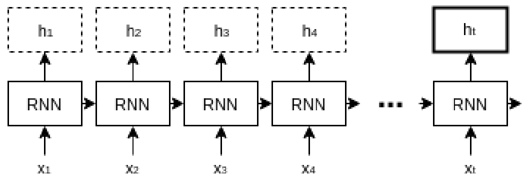
Product Title	Category ID Path
Replacement Viewsonic VG710 LCD Monitor 48Watt AC Adapter 12V 4A	3292>114>1231
Ka-Bar Desert MULE Serrated Folding Knife	4238>321>753>3121
5.11 TACTICAL 74280 Taclite TDU Pants, R/M, Dark Navy	4015>3285>1443>20
Skechers 4lb S Grip Jogging Weight set of 2- Black	2075>945>2183>3863

followed by random shuffling; next we pick a subset of the bag of words and generate a sequence based on them; finally we append the sequence to the end of the original sample to get the augmented one. This sampling strategy aims at enforcing the model to be robust to the noise generated by reordering or repeating pieces of a sample itself.

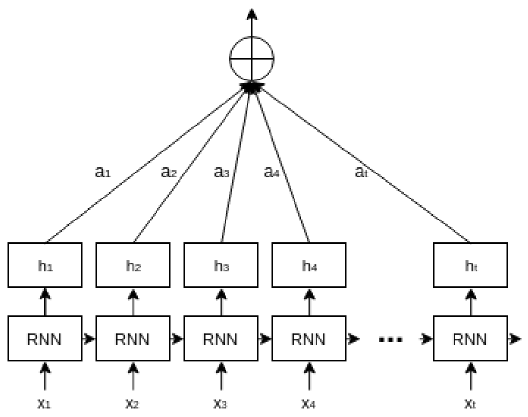
2.2 Recurrent Neural Network

We model product titles using recurrent neural networks (RNN). RNNs processes their input in a sequential way, with each time step sharing the same operation (commonly achieved by sharing weights). In addition, for a rnn, the output of each time step is fed back to itself as a part of the input at next time step. In this way, a rnn is powerful at handling inputs of variable length.

However, RNNs are known to be hard to train [9], due to the gradient exploding/vanishing problems [2, 4]. A key idea to overcome these problems is to construct constant error flow (CEF) for each RNN neuron. Inspired by it, more sophisticated variants of vanilla RNNs like Long Short Term Memory (LSTM) network [5] and Gated Recurrent Unit (GRU) network [3] are proposed, which allow better gradient flow to learn the long-term dependencies.



(a) Regular RNNs



(b) RNNs with attention mechanism

Figure 2: Comparison between a regular RNN and a RNN with attention.

2.3 Self-Attention Mechanism

Instead of directly using the final hidden state h_t of a rnn on a product title as its final representation r, we use a self-attention mechanism [1], in order to amplify the contribution of important words. When using a attention mechanism, we compute r as a convex combination of all hidden states h_i, $i \in [1, t]$, with weights a_i, indicating the importance of their corresponding h_i. Formally, $r = \sum_{i=0}^{t} a_i h_i$, where $\sum_i a_i = 1$ and $a_i >= 0$. Figure 2 illustrates the difference between regular RNNs and RNNs with attention mechanism.

3 MODEL DESCRIPTION

We use a multi-layer word-level BiLSTM to capture context information for each word of a product title and a multi-head attention model to aggregate useful information from the learned word representations generated by the BiLSTM. We present the architecture of the proposed model in Figure 3.

Embedding Layer. The input to the network is a product title, treated as a sequence of words. We use an embedding layer to project the words w_1, w_2, w_3, ..., w_t to a low dimensional dense vector v_i^r, where r is the dimension of embedding space and t is the number of words in a product title. It is often popular to pre-train word embeddings with algorithms like Word2Vec [8] and Glove [10], but we simply randomly initialize them with other parameters in our model.

BiLSTM Layer. A LSTM takes a sequence of vectors as input and produces an annotation for each time step h_1, h_2, ..., h_t. A BiLSTM performs similar operations, but in both forward and backward directions. Although there are various ways to combine the forward and backward annotation $h_{f,i}$ and $h_{b,i}$ for a BiLSTM, we simply concatenate them together, i.e., $h_i = h_{f,i} || h_{b,i}$, where $||$ denotes the concatenation operation. Note that $h_i \in R^{2L}$, where L is the size of the BiLSTM hidden layer.

Multi-Head Attention. Similar to the attention mechanism mentioned above, a multi-head attention model also aims at aggregating useful information from word features, but allows multiple convex combinations for attention on different words. In our model, we adopt an attention model with the following transitions:

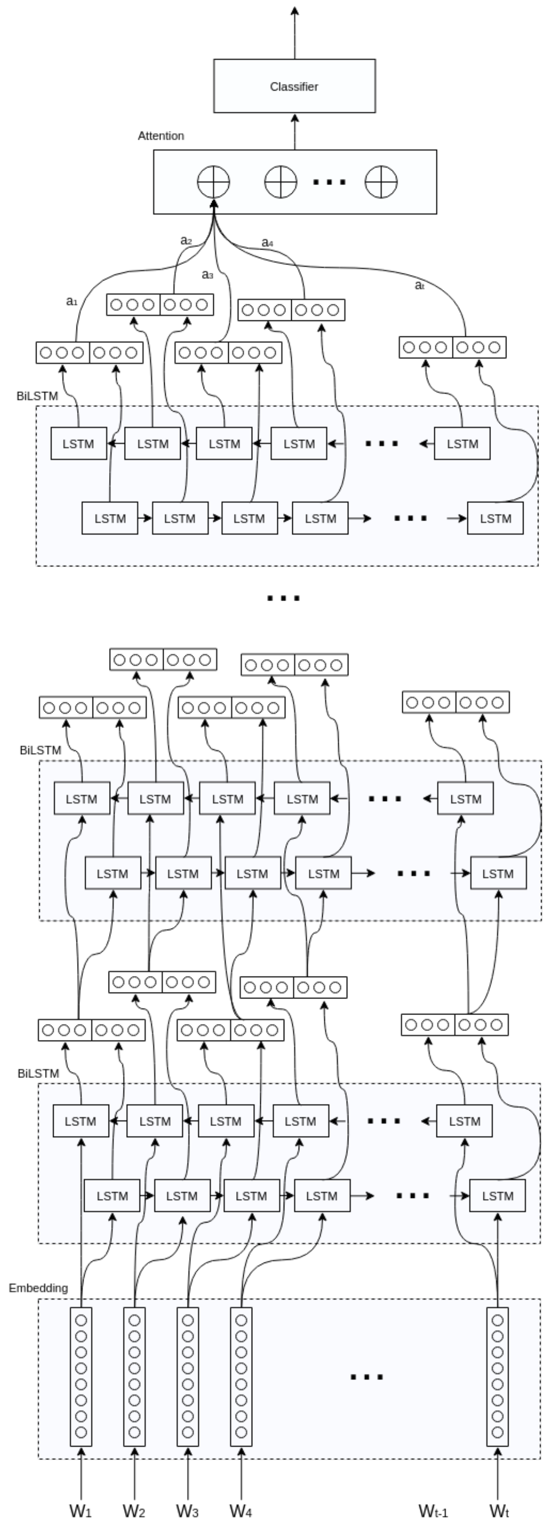


Figure 3: The architecture of our proposed model.

$$d_i^{(j)} = w_2^{(j)} \tanh(W_1^{(j)} h_i + b_1^{(j)}) + b^{(j)} \tag{1}$$

$$a_i^{(j)} = \frac{exp(d_i^{(j)})}{\sum_l exp(d_l^{(j)})} \tag{2}$$

$$m^{(j)} = \sum_i a_i^{(j)} h_i \tag{3}$$

$$r = m^{(1)} || m^{(2)} || ... || m^{(K)} \tag{4}$$

where h_i is the representation of word w_i, $a_i^{(j)}$ is its corresponding attention weight for the jth head, $m^{(j)}$ is the aggregation result for the jth head and r is the aggregation result for all the heads. $||$ denotes the concatenation operator and $W_1^{(j)} \in R^{2Lx2L}$, $b_1^{(j)} \in R^{2L}$, $w_2^{(j)} \in R^{2L}$, $b^{(j)} \in R$ are the weight parameters, with L denotes the size of the BiLSTM hidden layer and K indicates the number of heads.

Classifier. We use a single layer MLP followed by a softmax activation as the classifier in our model.

3.1 Regulation

In this paper, we also adopt different regulation techniques to improve the model's generalization capability. In specific, we use L_2 regularization, embedding dropout, DropConnect [7] and dropout [12].

Embedding dropout. We employ embedding dropout by randomly dropping out dimensions of word embeddings with the rest dimensions scaled by $1/1 - \rho$, where ρ denotes the dropout probability. This is equivalent to adding random bernoulli noise to the word embeddings.

DropConnect. Preventing overfitting within recurrent neural network has been a popular research area that draws a lot of attention. Many of the proposed methods focus on the hidden state vector h_i, aiming at introducing a dropout operation between time steps or on the update to the memory state c_i. [7] instead proposes an approach called "DropConnect" that randomly throws away connections of hidden neurons to themselves, i.e., the hidden to hidden weight matrices. In our approach, we adopt this technique on both forward and backward LSTMs.

Dropout. Dropout is widely used as a regulation technique for deep neural networks. We employ the technique between BiLSTM layers and before self-attention model. When applying dropout between BiLSTM layers, we scale the non-dropped dimensions by $1/1 - \rho$, similar to embedding dropout, while when used before self-attention model, a regular version is adopted, i.e., dimensions are scaled by $(1 - \rho)$ at evaluation time.

3.2 Optimization

SGD remains one of the most popular optimization techniques for training deep learning models in various areas, such as computer vision, natural language processing and deep reinforcement learning. As a variant of SGD, Non-monotonically Triggered ASGD [7] (NT-ASGD), may further improve the training process as it provides certain advantages such as its asymptotic second-order convergence [6, 11]. We adopt NT-ASGD and SGD as the optimization algorithms.

4 EXPERIMENTS

4.1 Experiment Setup

Training. We use the combination of SGD and NT-ASGD [7] as the optmization algorithm. Initially we start training the model by SGD algorithm with logging interval set as one epoch. After 5 non-monotone interval, NT-ASGD is triggered and employed for the rest epochs. We set the mini-batch size to be 32, the word embedding size to be 300, the hidden size of BiLSTM to be 400, the number of layers of BiLSTMs to be 2, the number of heads to be 3, the embedding dropout rate to be 0.4, the DropConnect rate to be 0.5, the dropout rate between BiLSTM layers to be 0.25 and the dropout rate before self-attention model to be 0.3. The initial learning rate is set to be 0.5 and the weight decay rate to be 1.2e-6.

Result. We list the current evaluation result on test data in Table 2, along with systems with relative close performance. Our system currently rank at 15 with weighted precision, recall and F1 to be 0.78, 0.77 and 0.77.

4.2 Accuracy Analysis

In order to analyze the strength and weakness of our system, we perform an analysis on accuracy with respect to $log_2(n) + 1$ for each category ID path, where n is their corresponding number of product titles in the train set. We show the result in Figure 4.

Generally speaking, our model performs better on frequent category ID paths than rare ones. For most frequent category ID paths, the model can achieve almost 100% accuracy, but varies when it comes to rare ones. It is as expected since deep learning models are well known to often be data hungry. The more data they are fed, the better performance they can achieve.

Another observation is that the overall accuracy for train set is at least above 0.80 as the accuracy of only some less frequent category ID paths is below that threshold. Compared to the performance of our model on test data, this suggests that our model is overfitting the train set, indicating the necessity of better regulation techniques.

5 EXTENSION

After stage2, we further improve our model by adopting a set of pre-processing steps and Glove vectors [10] for word embedding

Table 2: Evaluation result on the test data released

Stage	Team	Precision	Recall	F1
stage1	Tyche	0.8536	0.7655	0.7976
stage1	Topsig	0.8009	0.8042	0.7967
stage1	VanGuard	0.7950	0.7915	0.7871
stage1	Waterloo	0.7819	0.7853	0.7767
stage1	**CorUmBc**	**0.7822**	**0.7722**	**0.7702**
Stage	Team	Precision	Recall	F1
stage2	Topsig	0.7921	0.8014	0.7941
stage2	VanGuard	0.7899	0.7917	0.7884
stage2	HSJX-ITEC-YU	0.7809	0.7821	0.7790
stage2	Waterloo	0.7802	0.7857	0.7781
stage2	**CorUmBc**	**0.7745**	**0.7712**	**0.7690**

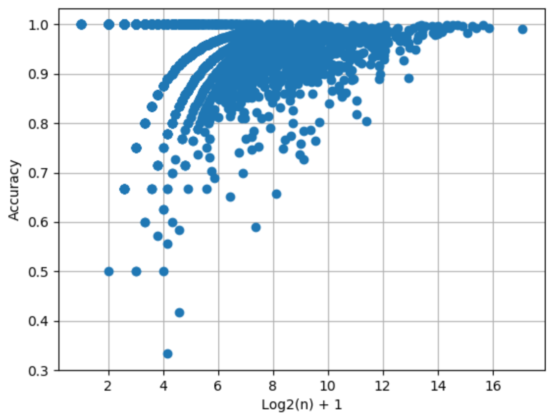


Figure 4: Accuracy w.r.t. $log_2(n) + 1$ where n is the number of product listings for a category ID path.

initialization. These pre-processing steps include: (1) lower case the product titles; (2) remove all non-ascii characters; (3) remove all punctuation; (4) remove all digits; (5) stem all words with NLTK WordNet Lemmatizer; (6) remove all rare words with document frequency less than 3.

We randomly split train data into train/valid/test sets according to the ratio 0.8/0.1/0.1. In order to compare the impact of the new set of pre-processing steps and Glove vectors, we perform two different runs: one with the same setting adopted in section 4 and the other with all the newly added extensions. Except that the number of layers of BiLSTM is set to be 3 and the hidden size is set to be 350, we use exactly the same training setting as in section 4. We show the results in Table 3 and these results indicate that these extension steps may further improve the performance of our model.

6 FUTURE WORK

We aim at further improving the performance of our model in the following directions.

Word Dropout. We find a large portion of words occur only once or twice in the train set, which may cause the model to unexpectedly rely on them as they show strong discrimination power when it comes to classification when the model has enough capacity. A possible way to reduce the impact is to randomly dropping out rare words before feeding a product title to the network.

Table 3: Evaluation result on the test set sampled from train data

Set	Model	Precision	Recall	F1
valid	CorUmBc + extension	**0.8054**	**0.8112**	**0.8036**
valid	CorUmBc	0.7751	0.7755	0.7699
test	CorUmBc + extension	**0.7888**	**0.7876**	**0.7801**
test	CorUmBc	0.7570	0.7504	0.7448

Dynamic Class Re-Weighting. One technique to overcome the problem that the network simply ignores rare category ID paths is to re-weight classes in order to force the model to pay more attention to rare ones. Thus changing class weights dynamically during the training procedure seems promising.

Ensemble of Models. Ensembling through bagging or boosting has proven to benefit many systems, thus we seek to improve model robustness by adopting this technique in the future.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.

[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[4] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[5] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[6] Stephan Mandt, Matthew D Hoffman, and David M Blei. 2017. Stochastic gradient descent as approximate bayesian inference. *arXiv preprint arXiv:1704.04289* (2017).

[7] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing LSTM language models. *arXiv preprint arXiv:1708.02182* (2017).

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[9] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. 1310–1318.

[10] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[11] Boris T Polyak and Anatoli B Juditsky. 1992. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization* 30, 4 (1992), 838–855.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.