Evaluating Natural Language Models on Technical Query Tagging

Stanford CS224N Custom Project

Kantapong Kotchum
Department of Mathematics
Stanford University
kkotchum@stanford.edu

John Dalloul
Department of Bioengineering
Stanford University
jdalloul@stanford.edu

Atindra Jha
Department of Computer Science
Stanford University
atj10@stanford.edu

Abstract

In this study, the relative performances of a non-parametric supervised learning method; a non-recurrent, dense neural architecture; a sequential neural model; and a Bidirectional Encoder Representations from Transformers (BERT) model were evaluated on the task of identifying the broader subject of a technical question. This comparison was done to investigate whether the task of identifying the suitable tags for a technical computer science question, and the general task of identifying the topic(s) of a technical piece of text, is one that is aided by temporal and sequential inference and the ability to derive context from the underlying patterns and structures of sentences. This study involved using the titles, bodies, and tags of 300 thousand Stack Overflow questions to train, fine-tune, and test the aforementioned models. The multi-layer bidirectional LSTM model generated the lowest performance metrics. Meanwhile, the best performance was achieved by the BERT model, with a recall of over 90% and F-1 score around 0.45 in both the validation and held-out test sets.

1 Key Information to include

- TA mentor: Angelica Sun
- External collaborators: No
- Sharing project: No

2 Introduction

Technical questions are often asked on online forums, and users assign tags to the questions as signposts for the topics of the queries. This project was aimed at assessing the performance and examining the underlying behaviors of various machine learning architectures on the task of automatically identifying tags for a given question. The specific goal was to assess whether such a task was one that would depend solely on capturing relationships between the input and the output using a feed-forward neural network, one that would benefit from learning sequential information from the input data that an LSTM could facilitate, or one augmented by long-distance dependencies and positional information allowed by a BERT model.

3 Related Work

Previous approaches on tackling multi-class and multi-label problems include techniques such as Bayesian classification [1], in which the model attempts to learn the probability distribution for words associated with a given tag. While this type of model scales well, it forces the assumption that words for a question are drawn independently of one another. Another investigation utilized Support Vector machines [2] which are robust against sparse inputs, a problem that frequently arises in representing a few categories in large feature spaces.

With the advent of deep learning infrastructures, several models have been proposed to classify texts. Many of these models utilize a recurrent neural network (RNN) architecture, and, in particular, LSTM (long-short term memory) cells. For instance, Sachan et al. [3] used a simple bidirectional LSTM and a combination of entropy minimization, adversarial, and virtual adversarial losses for both labeled and unlabeled data to achieve a high classification accuracy on several NLP benchmark datasets.

Google's AI Language paper which proposed the Bidirectional Encoder Representations from Transformers (BERT) model [4] introduced a powerful natural language processing paradigm based on the recently developed transformer architecture. The BERT model can be fine-tuned for a range of tasks, including question answering and language inference. One of the two architectures proposed in the paper, the BERT base model, consists of 12 transformer blocks, each of which contains 6 layers of multi-headed self-attention linear layers. This architecture consists of 110 million parameters and a total of 72 linear layers. This study aimed to take advantage of the wide applicability of pre-trained BERT models in automatically identifying tag labels for Stack Overflow questions.

4 Approach

Each question (data point) consists of a title and a body, both of which are represented by a sequence of words. In order to facilitate input into our models, we utilized word embeddings to represent each word in the questions. For the KNN, non-recurrent dense neural network, and the bidirectional LSTM model, we used 300-dimensional GloVe embeddings learned from Wikipedia 2014 and Gigaword 5 [5]. For the BERT model, we used pretrained uncased-BERT embeddings [4].

We implemented the following models:

- **K-Nearest Neighbors**: Each question was represented by the arithmetic mean of its word embedding vectors. For a given query, the k nearest data points from the training set by Euclidean distance were identified as the nearest neighbors, where k is a hyperparameter specifying the number of neighbors. From these neighbors, the majority label was selected as the prediction for the query question. We evaluated KNN models for $k \in \{1, 4, 16, 128\}$.
- **Non-Recurrent Dense Neural Architecture**: Each question was represented as a concatenation of its constituent word embeddings. This concatenation was flattened and fed to a dense neural network model with four hidden layers of sizes 2048, 1024, 512, and 256 neurons. The ReLU activation function was used after each hidden layer. The output layer comprised of 150 neurons, the output of which was passed into a Sigmoid activation function to obtain putative probabilities for each tag's relevance to the query.
- **Sequential RNN**: Each question was represented as a sequence of its constituent word embeddings. This series was fed into a three layer bidirectional LSTM, with each layer having a hidden unit size of 256. The hidden cell of the final LSTM layer was fed into a small, fully-connected neural network, with two hidden layers of size 512 and 256 and an output layer of size 150 neurons. The TanH activation function was used after each hidden layer. Again, the output was passed into a Sigmoid activation function to obtain putative probabilities for each tag's relevance to the query.
- **Bidirectional Encoder Representations from Transformers (BERT)**: Each question was represented as a concatenation of its constituent word embeddings. This concatenation was fed to a base BERT model (the architecture of which is described in "Related Work") with output size 150. Again, this output was passed into a Sigmoid activation function to obtain putative probabilities for each tag's relevance to the query.

5 Experiments

5.1 Data

The dataset used in this project was the "StackSample: 10% of Stack Overflow QA", provided on Kaggle [6] (link: `https://www.kaggle.com/stackoverflow/stacksample`). This dataset consisted of 1,264,214 Stack Overflow questions in total with associated tags. For each question, the title, body, and tags were used. The distribution of tags in the dataset was highly skewed, with 87% (1,098,644) of the questions in the dataset labeled with one or more of the 150 most frequently used tags (out of 37,034 tags total). Therefore, the dataset was filtered to only select the questions that were labeled with the top 150 tags. Additionally, due to computational restraints, a random subset of 300 thousand questions was chosen. Finally, this subset was processed by removing HTML tags and accent marks, expanding grammatical abbreviations, and converting all characters to the lowercase. The dataset was then split into train-val-test sets using the ratio 80:10:10.

5.2 Evaluation method

The multi-label classification task of predicting the subset of 150 tags appropriate for each question required training the three parametric models (dense neural network, bidirectional LSTM, and BERT) to minimize binary cross-entropy loss (BCE loss).

$$D_{BCE}\left(y,\hat{y}\right)=-\frac{1}{N}\sum_{i=1}^{N}\left[y_i log\left(\hat{y}_i\right)+\left(1-y_i\right)log\left(1-\hat{y}_i\right)\right]$$

Here, the loss is computed for each data point. N is the number of classes in the output tensor (150), y_i is the i-th entry in the ground truth label, and $\hat{y}_i$ is the i-th entry in the probability vector.

In addition to calculating the BCE loss from the probabilities generated by the models, a prediction tensor was derived using the five highest probability tags. Following is an example of the methodology used to derive the prediction tensor, albeit from a probabilities tensor scaled-down to 8 dimensions for the sake of visualization.

$$\begin{bmatrix}0.1\\0.2\\0.04\\0.5\\0.06\\0.01\\0.02\\0.07\end{bmatrix} \longrightarrow \begin{bmatrix}1\\1\\0\\1\\1\\0\\0\\1\end{bmatrix}$$

Left: Probability Vector $\hat{y}$, Right: Prediction Vector

This prediction tensor was compared to the true label to calculate the confusion matrix – true positive (TP), true negative (TN), false positive (FP), and false negative (FN) rates – for each tag. Precision, recall, and F1 scores for each tag were computed from these statistics.

$$\text{Precision}=\frac{\text{TP}}{TP+FP} \quad \text{Recall}=\frac{\text{TP}}{TP+FN} \quad \text{F1}=\frac{2\cdot\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$$

Recall indicates the proportion of true tags that were predicted by the model. Precision, meanwhile, indicates the proportion of true tags among those predicted by the model. Recall was prioritized as our objective was to find the model that was best able to suggest the most relevant tags from the ground truth. The F1 score, which is a harmonic mean of precision and recall, was also included to take into account the precision of the model.

Averaging these scores across the 150 tags gave an approximation of model performance on these three metrics alongside the BCE loss.

5.3 Experimental details

Since less than 10% of questions in our initial dataset contained more than 300 words in the body, each question body was clipped to the first 300 words. Furthermore, the longest question title in the dataset consisted of 35 words. The input for each question was a concatenation of the title and the body; hence, the maximum input size for a question was 335 words. Questions with concatenated title and body length less than 335 were padded with the "0" string at the end such that all question inputs were equally of length 335.

Four K-Nearest Neighbors models, each with a different number of neighbors ($k = 1, 4, 16, 128$) were used for initial experimentation. The first two parametric models that were tested were a feed-forward neural network ("Dense NN (2048)," Figure 1(b)) and a bidirectional LSTM model ("LSTM," Figure 1(c)), both of whose architectures are largely described in the "Approach" section. The Adam optimizer with a learning rate of 0.001 and no weight decay was used in training both models for 20 epochs. Batch sizes of 512 and 1024 were used for the two models respectively. Relatively large minibatches were used to decrease training time.

After observing the results from the aforementioned two parametric models, the architecture of the LSTM model was altered in an attempt to offset its weak performance with the strength of the feed-forward neural network. The hidden cell of the final LSTM layer, which was initially being fed to a 512-neuron hidden layer, was instead passed to the 2048-neuron hidden layer of the described dense neural network. This modified LSTM ("Large LSTM," Figure 1(d)) was trained in a similar fashion to the unmodified one. Additionally, we tested the feed-forward architecture that the final hidden cell of the original LSTM was initially connected with as a separate model ("Small Dense NN," Figure 1(a)). The results that were observed have been discussed extensively later on in the paper.

Finally, a pretrained self-attentive model (BERT) was fine-tuned for the specific task, with the BCE loss being optimized using the BertAdam optimizer with a learning rate of 2×10^{-5}. A smaller learning rate was chosen for this model because its weights were not randomly initialized; rather, they were already optimized using a large, separate corpus of text. Minibatches of size 16 were used during training due to computational limitations. The model was initially set to be trained for 20 epochs; however, the BCE loss had converged by the 8^{th} epoch so the model was stopped early. Each epoch took approximately 2 hours to train due to the complexity of the model and the small batch size used.

5.4 Results

The results for the non-parametric K-Nearest Neighbors models can be seen in Table 1. It is evident that the model with $k = 1$ outperformed the others by a significant margin, achieving an F1 score of 0.2066 and a recall of 20.95% on the held-out test set.

# Neighbors	Val			Test		
	Loss	Recall	F1	Loss	Recall	F1
1	1.8750	**0.2115**	**0.2096**	1.8840	**0.2095**	**0.2066**
4	1.1398	0.0784	0.1391	1.1421	0.0745	0.1325
16	**1.1348**	0.0525	0.0980	**1.1328**	0.0509	0.0952
128	1.1644	0.0130	0.0255	1.1616	0.0116	0.0228

Table 1: K-Nearest Neighbor Results on Validation and Test Sets

The validation and test results for our three primary models can be observed in Table 3. The artificial neural network with 2048 neurons in the first hidden layer performed surprisingly well, yielding a recall of 67.32% and an F1 score of 0.3509 on the validation set. This model ultimately performed equally well on the test set, yielding a recall of 65.48%. These metrics were rather low for the 3-layer bidirectional LSTM connected to an artificial neural network with a 512-neuron first hidden layer. The recall was 24.45% and the F1 score was 0.1308 on the test set.

Upon observing the significantly worse performance of the LSTM as compared to the Dense NN (2048), we decided to alter and experiment with the former. As mentioned previously in "Experimental

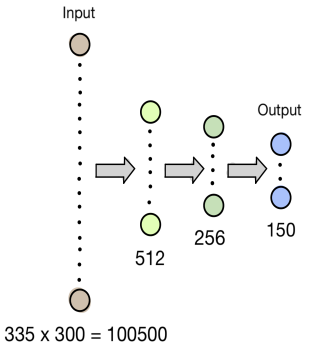
Methods," we replaced the feed-forward neural network connected to the final hidden cell of this LSTM with the Dense NN (2048) to form the Large LSTM. Additionally, we tested the Small Dense NN that the the final hidden cell of the LSTM was initially connected with as a separate model. The results in Table 2 indicate that the independent Small Dense NN had performance nearly identical to that of the original LSTM architecture.

Model	Val		
	Loss	Recall	F1
Small Dense NN	0.0555	0.2764	0.1441
Large LSTM	**0.0487**	**0.4224**	**0.2202**

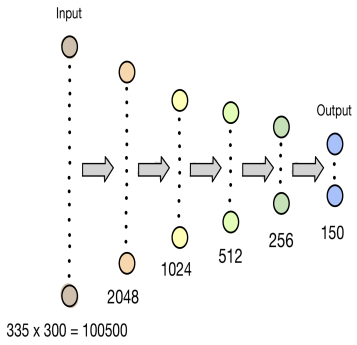
Table 2: Dense and LSTM Models

Model	Val			Test		
	Loss	Recall	F1	Loss	Recall	F1
Dense NN (2048)	0.0402	0.6732	0.3509	0.04082	0.6548	0.3478
LSTM	0.0556	0.2764	0.1441	0.05784	0.2445	0.1308
BERT	**0.0173**	**0.9201**	**0.4772**	**0.01788**	**0.9032**	**0.4697**

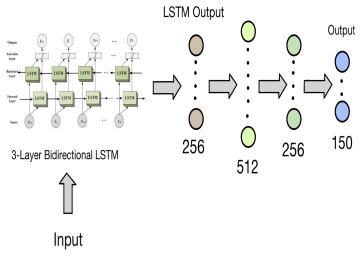
Table 3: Three Best Models on Validation and Test Sets



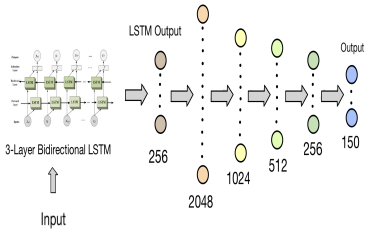
(a) Small Dense NN



(b) Dense NN (2048)



(c) LSTM

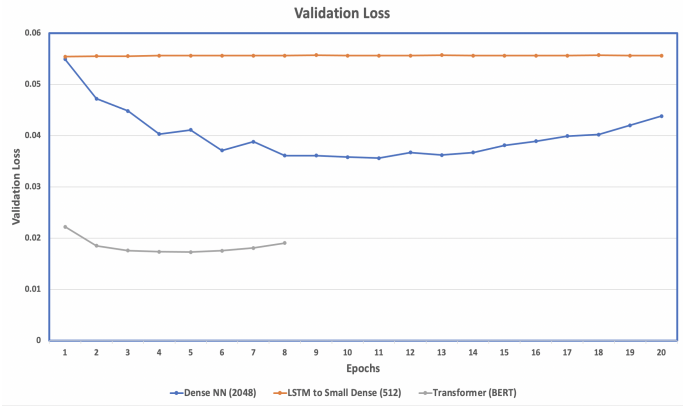


(d) Large LSTM

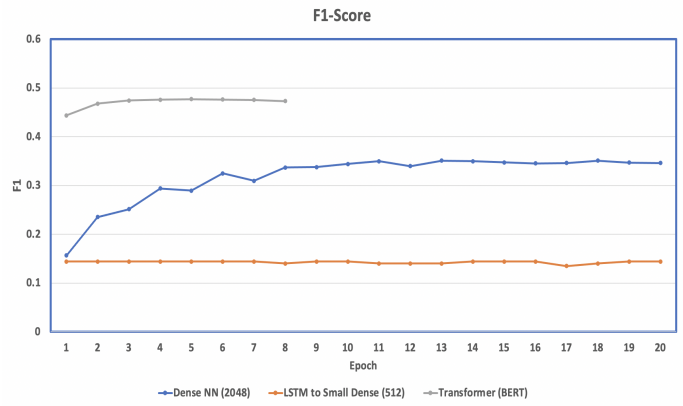
Figure 1: Model Architectures for Dense and LSTM Networks

The best performance was achieved by the pretrained self-attentive model (BERT), which reached a BCE loss less than half that of the next best model. There was a dramatic improvement in recall and a

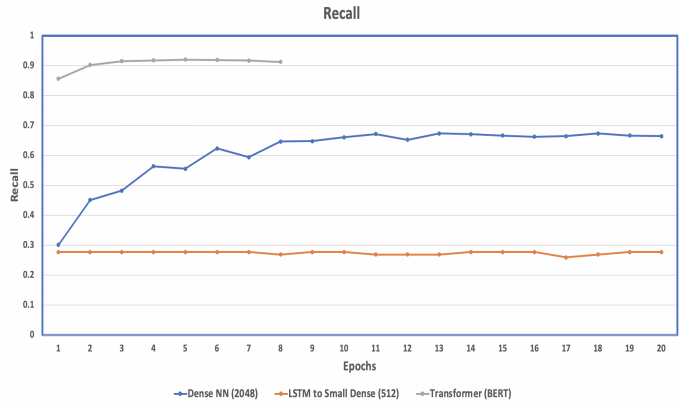
simultaneous jump in the F1 score, 90.32% and 0.4697 respectively on the test set. The performance metrics of this model across validation and testing can be observed in Table 3.



(a) Validation Loss on Validation Set



(b) F1 Score on Validation Set



(c) Recall on Validation Set

Figure 2: Evaluation Metrics Across Model Training

6 Analysis

The K-Nearest Neighbors model with the highest recall only selects one neighbor, and as the number of neighbors increases the recall decreases. Since the KNN algorithm generates a prediction per tag by taking the majority among the nearest neighbors, from the low recall values for higher number of neighbors we can infer that the distribution of most tags across data points is quite sparse. As more neighbors are taken into account, it becomes less likely that a given tag will be predicted, decreasing the true positive rate and increasing the false negative rate; both of these effects decrease recall as suggested by the KNN model results.

The LSTM performed significantly worse than the Dense NN (2048) model. Moreover, independently training the Small Dense NN, i.e. the feed-forward neural network that was attached to the final hidden cell of the LSTM, yielded results identical to the entire LSTM architecture. This behaviour suggests that predicting the correct tags for the questions in our dataset may not be a sequential classification problem.

Furthermore, training the Large LSTM, i.e. the Dense NN (2048) attached to the final hidden cell of the LSTM, gave results that represented a marked improvement over the initial LSTM, yet still fell short of the Dense NN (2048) model standing alone. In the case of the previously mentioned Small Dense NN, prepending the LSTM neither improved nor deteriorated the metrics. With the Dense NN (2048), the inclusion of the LSTM failed to reach the scores of the independent Dense NN (2048). It can be argued that the condensed, 256-dimensional hidden unit representation produced by the LSTM and fed into the Dense NN (2048) is not as sufficient as the entire 335 * 300 = 100500 dimensional input being passed directly into the Dense NN (2048). LSTM networks' general inability to keep relevant information active as the input becomes longer, as well as the fewer features in the described LSTM's last hidden unit, make it a relatively lower quality input for the feed-forward neural network.

In addition, it can be inferred that either of the following could hold true:

1. Keeping track of arbitrary dependencies (short- or long-term) in the input was not pertinent to this task. This would imply that the intra-attention capability of the BERT model would not contribute to performance enhancement.
2. The dependencies that the LSTM was able to capture were not long enough for an uptick in performance and that we needed a multi-headed self-attentive transformer model (BERT) to capture even longer-term dependencies and improve performance. If this is the case, it can be deduced that the BERT model's self-attention capability would improve performance on this task.

The remarkably better performance of the BERT model in relation to the other two models has been illustrated in the graphs in Figure 2. However, this does not help us conclude whether its ability to capture long-term dependencies is the reason why it outperformed Dense NN (2048) and the LSTM model.

On one hand, it could be inferred that self-attention does help, given this model's greater ability to capture dependencies than the LSTM. A transformer relies entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs. The self-attention mechanism relates different positions of a single sequence to compute a representation for it. BERT follows the same core principle - attention, which understands the contextual relationship between different words. While our bidirectional LSTM model conceives each input sequentially (left to right **and** right to left), BERT is non-directional and reads the whole sentence as the input instead of sequential ordering. This characteristic allows this model to learn the contextual information of a word with respect to all other words in the sentence. This ability to capture dependencies and underlying patterns significantly better than the LSTM may have boosted the recall and F1 score for this model.

On the other hand, the sheer depth of the base BERT model (12 transformer blocks, each with 6 multi-headed self-attention layers, i.e. 72 linear layers across the model) may have boosted its performance. While the 5-layer feed forward dense neural network is deep, the BERT model is deeper by an order of magnitude, which may have drastically improved its evaluation metrics. Its extraordinary ability to capture context and dependencies may have been irrelevant to this specific task.

7 Conclusion and Future Work

This study showed that a BERT model was best able to capture important long-term dependencies from input questions and output relevant question tags. Surprisingly, a fully-connected neural network bested a bidirectional LSTM model, suggesting that the task does not benefit from sequential inputs, and that the limited memory and condensed features of the LSTM model yield poor inputs to be fed into a feed-forward neural network.

Despite the superior performance of the BERT model, it remains to be seen whether this result is due to the self-attention built into the BERT architecture or merely the depth of the model. To distinguish between these two possibilities, future work would need to implement dense neural networks with depth comparable to the base BERT model's 72 layers. Evaluating the performance of such deep neural networks against that of the BERT model may help ascertain the reason behind the BERT model's success.

References

[1] A. K. McCallum. Multi-label text classification with a mixture model trained by em. In *AAAI 99 Workshop on Text Learning*, 1999.

[2] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. 1998.

[3] Ruslan Salakhutdinov Devendra Singh Sachan, Manzil Zaheer. Revisiting lstm networks for semi-supervised text classification via mixed objective function. In *AAAI 2019*, 2019.

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

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove 300-dimensional word vectors trained on wikipedia 6b. In *GloVe*, 2014.

[6] StackOverflow. Stacksample: 10% of stack overflow qa. In *Kaggle*, 2021.