Enhancing Word Embeddings With Semantic Data In An NLP QA System

Anonymous Author (singular, although I use we in the paper)

Abstract

This paper explores ways to enhance a neural QA system (Chen et al. (2017)) by introducing additional semantic data. As word embedding techniques have become the foundation of neural NLP systems, it is worth asking *how much* information do these dense representations actually capture. We explore approaches to ensemble semantic information from other external sources to improve the performance of the model at the task at hand, without necessarily changing the architecture of the model, and analyze the challenges and results.

1 Introduction

Word embedding techniques like word2vec (Mikolov et al. (2013)) and GloVe (Pennington et al. (2014)) have become commonplace in neural NLP systems due to their ability to encode word meaning densely into a fixed-width vector; this allows us to build efficient neural models, without having to worry about the large dimensionality of the traditional bag-of-words sparse feature vector approach. Usually, word vectors are derived from their context, and their representation carries some meaning. For example, words that appear in the same context frequently will have vector representations that are dot-product-wise close to each other. Thus word embeddings model some information about word relationships that make them useful as a building block for NLP systems.

It is unclear, however, whether such embeddings 'embed' within their representations any useful meta-information - Named Entity Recognition (NER), Part of Speech (PoS), etc. Modern parsers can easily extract this data from text, and this data is still very valuable in more baseline feature-based approaches. There are some signs that large pre-trained models have some internal representation of these concepts (Tenney et al. (2019)).

This paper sets out to answer a couple of questions - can pre-trained word embeddings benefit from parsed data augmentation, or are they already saturated? If so, do they add to the general robustness of the system?

2 Setup

In order to explore whether a model's performance on a task can be improved by injecting some of semantic data into its pre-processing step, we first have to have a model and a task. The task is extractive question answering (Fisch et al. (2019)), by using an already implemented RNN attention-based neural model (Chen et al. (2017)) - Dr QA. The base code can be found here - https://github.com/gregdurrett/nlp-qa-finalproj and the updated code is attached with the paper. Extractive question answering involves identifying a sequence of words as an answer within a provided passage, based on a correlated question. The baseline implementation uses GloVe (Pennington et al. (2014)) pre-trained word embeddings, followed by an aligned attention layer that attends the question throughout the passage, LSTM passage and query encoding layer and question encoding layer. Finally, the question encoded hidden states undergo a softmax linear reduction to compute question attention weights, and then the hidden states of the passage get softmax reduced in a sliding-window style against the attention weights to compute the log likelihoods of a passage word being the start or end of the answer.

The baseline implementation achieves an ok performance on the main Squad dataset - 'EM': 48.74, 'F1': 61.43. A BERT-based solution has already achieved a state of the art performance on this task (Devlin et al. (2018)), so it seems reasonable that trying to incorporate additional higher-level semantic information into the pipeline should lead to an

improvement to the baseline performance. Since there are multiple datasets to test against, we would also expect the robustness of the model to increase in the cross-domain and adversarial cases. Question answering naturally revolves around named entities in some domains, so focusing on embedding NER data makes some intuitive sense, as the model can use that information to attend to tokens that are likely more important to the task at hand.

We propose a couple of approaches that we implement and analyze. The first one is a naive NER tag interpolation; the concept is simple - we can parse out NER tags using a parser, and then annotate the input (passage and questions) with the tags, thus making them regular tokens, and part of the vocabulary. This implementation is nice because it separates the interfaces - the model is a black box, and only the input and output have to be modified, but at the same time there is not much control of how the model integrates the tokens.

The second approach is a deep NER tag embedding - the NER tags are to be treated as a separate pair of inputs, and undergo the tokenization and sparse to dense embedding process. After that, they can be integrated into their associated word embeddings. For this step, we tried a couple of approaches; one is to use a bilinear transformation $v_w W_e v_t^T$ where v_w is the vector of word embeddings, W_e is a general linear transformation from the tag embedding space to the word embedding space and v_t is the vector of tag embeddings. Another approach is to concatenate v_w and v_t, then use a linear transformation to reduce them to the original embedding size. The goal is to have a learnable transformation that can embed the tag data into the word embedding space, while ideally preserving the subspace, or preserving desirable characteristics of it. There may be other approaches that would overall keep the embedding subspaces different (concatenation) or some other PCA-preserving transformation.

This seems like a good time to mention transformers, as is modern NLP paper tradition. Obviously we won't go into details as to how they work, or where they make sense in this whole thing, partly because we are unsure ourselves, but it's good luck to hopefully just mention it and link to the ref (Vaswani et al. (2017)).

3 Implementation

We already covered the base model at a high-level; the model is implemented using the pytorch library in python and the implementation details can be found with the linked code. Model hyperparameters are also included at the end of the paper. To implement the NER parsing we used the spaCy library (https://spacy.io/), which provides good off-the-shelf parsing and an easy to use python library. We introduce the NER data parsing early on in our pipeline, at the data loading step. For ease of repeated runs, we cache NER parse results to reuse them if available using the diskcache library.

For the naive NER implementation, the tokens are interpolated directly into the passages and questions in the dataset. Some updates were needed to handle length properly (since sentence lengths change vs. the original), and the gold positions were updated to adjust for the new tokens. The answer has the tags removed at the end to be able to properly compare with the gold answer for the accuracy. Example transformed input:

> [CONTEXT] super bowl 50 CARDINAL was an american NORP football game to determine the champion of the national football league (nfl) for the DATE 2015 DATE season DATE . the american NORP football conference (afc) champion denver broncos defeated the national football conference (nfc) champion carolina panthers 24–10 CARDINAL [truncated]
>
> [QUESTION] where did super bowl 50 CARDINAL take place ?

For the deep NER implementation, the tokens are indexed and embedded in a lower-dimensional dense vector. The vector is then sub-embedded in the word embeddings according to the implementation described earlier.

4 Results and Analysis

All models were trained in the Google Colab environment, using a variety of modern GPUs (nVidia T4 or equivalent) and a relatively performant underlying VM.

The parsing step definitely added overhead to the training process, and it also adds overhead to the application of the model. On SQuAd, the baseline model can be trained in 10 minutes, NER parsing

model	SQuAd	SQuAd Adversarial	bioasq	newsqa
baseline	48.74/61.43	38.39/48.34	11.44/19.41	19.66/31.0
naive ner	35.78/50.52	26.52/39.22	10.24/18.78	14.41/26.72
deep ner	48.75/61.19	34.86/46.45	6.98/14.85	18.04/29.87

Table 1: EM/F1 scores for the various model tests

adds another 10 minutes. A full parser run on the training set adds another 20 minutes (only used for the PoS pre-processing). To note, the spaCy parser cycle was not optimized; it could benefit from batching and GPU usage.

Quantitatively, we use the exact matches (EM) and F1 score to rate the performance of the models. We evaluated the model on the various cross-domain datasets, and list them in Table 1. Holistically, our implementation did not beat the benchmark, but we did find some encouraging results.

The naive NER augmentation performed worse than the baseline. In some sense, this is due to the fact that the NER tags become a 'first-class' citizen, as the model does not distinguish them from other words, and thus may weigh in more heavily in parts in which they shouldn't. When trying the deep NER augmentation, using an embedding size for the NER that was equal to the word embedding size (300 dimensions) also yielded a similar performance on the SQuad dataset - 'EM': 21.06, 'F1': 30.59 vs 35.78/50.52 for naive NER. In some sense, having the NER play a first-class role in the embedding impacts performance. Empirically, when analyzing some of the answers, it is obvious the model prefers words or sentences that have an NER tag associated, to a fault - i.e. "Aristotle", "Algeria", "twelve", "Nova Scotia", etc. In the opposite direction, the baseline model tends to error by producing seemingly 'out-of-context' sentences - "receive a large", "join a". In some sense, the NER focus doesn't improve the overall accuracy, but solves some of the problematic behaviors of the baseline model, since the NER answers tend to be terse. Of course, a language model may be able to improve the sentence likelihood problem, and a factuality constraint could work even better (Goyal and Durrett (2020)).

The deep NER augmentation results were run using the bilinear sub-embedding approach, with NER embeddings being set to a lower-dimensional space - 30 vs 300. The deep NER augmentation yields no gains over the baseline. In fact, on the bioasq dataset it performs worse. That gives us a hint that NER augmentation does change the behavior of the model, but doesn't give us a net positive gain, especially in domains where NERs are less useful (like the bio dataset). Even though the performance is similar, the answers the models give are different - for example, on the adversarial SQuAd dataset, out of 1787 answers, 44 are shared between the deep NER model and the baseline, even though they each get around 30% exact matches, so 10 times that number. Furthermore, the naive NER implementation shared 601 exact answers as the deep NER, which further shows that the deep NER model makes use of the NER data, albeit producing somewhat orthogonal errors.

Some more examples (see attached files for more):

baseline:

"qid": "57378c9b1c456719005744a7", "answer": "fully stated"
"qid": "57379ed81c456719005744d6", "answer": "these tandem effects result ultimately"
"qid": "5737a9afc3c5551400e51f64", "answer": "systems from ordered to more random conditions as entropy increases"
"qid": "5737a9afc3c5551400e51f64-high-conf-turk0", "answer": "systems from ordered to more random conditions as entropy increases . the 7th"
"qid": "5737aafd1c456719005744fc", "answer": "mug or hyl) is that mass that accelerates at 1 m0̆0b7s2̆2122 when subjected to a"

"qid": "57378c9b1c456719005744a7", "answer": "lorentz 's law"
"qid": "57379ed81c456719005744d6", "answer": "physical direction"
"qid": "5737a9afc3c5551400e51f64", "answer": "transfer of heat"
"qid": "5737a9afc3c5551400e51f64-high-conf-turk0", "answer": "open system heat exchange"
"qid": "5737aafd1c456719005744fc", "answer": "kilopond"

5 Other Work Future Work

We've showed evidence leading to the support of our hypothesis - even though NER augmentation of word embeddings didn't produce an improvement, it produced an orthogonally competitive model to our baseline. This gives some hints to the possibility of ensembling the two systems to produce an overall stronger learner.

After completing the data gathering, we found some work that is in the same vein by Pittaras and Karkaletsis (2019). The model application seems similar, but the augmentation approaches diverge. Still, the authors found some success in augmenting pretrained word2vec embeddings in their example.

For future work, it would be great to see an improvement from this type of embedding augmentation - could be different tags or parsing systems. Like noted before, experimenting with the type of deep augmentation may be worthwhile. The code submitted does include PoS tag embedding, but alas the time has run out on being able to finish and include that result. The next goal would be to check if fine-tuned BERT models produce embeddings that already include higher-level semantics. If large models like BERT can automatically learn an appropriate 'representation' for some of the additional semantic or syntactic data, then that is an easier approach than the manual/architectural integration described here.

6 Supplementary Materials

Model parameters (shared across all models):

```
{'add_tags': 'ner_deep',
 'batch_size': 64,
 'bidirectional': True,
 'dropout': 0.0,
 'embedding_dim': 300,
 'embedding_path': 'glove/glove.6B.300d.txt',
 'epochs': 10,
 'hidden_dim': 256,
 'learning_rate': 0.001,
 'max_context_length': 384,
 'max_question_length': 64,
 'model': 'baseline',
 'rnn_cell_type': 'lstm',
 'shuffle_examples': False,
 'use_gpu': True,
 'vocab_size': 50000,
 'weight_decay': 0.0}
```

References

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Adam Fisch, Alon Talmor, Robin Jia, Minjoon Seo, Eunsol Choi, and Danqi Chen. 2019. MRQA 2019 shared task: Evaluating generalization in reading comprehension. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 1–13, Hong Kong, China. Association for Computational Linguistics.

Tanya Goyal and Greg Durrett. 2020. Evaluating factuality in generation with dependency-level entailment. *CoRR*, abs/2010.05478.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

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)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Nikiforos Pittaras and Vangelis Karkaletsis. 2019. A study of semantic augmentation of word embeddings for extractive summarization. pages 63–72.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. *CoRR*, abs/1905.05950.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.