Introduction

Sentiment analysis is a specific task within Natural Language Understanding (NLU), among other tasks such as question classification, language modeling, question answering, machine translation, etc. In the context of the Yelp review dataset, each review's corresponding star rating value is interpreted as a proxy of the review writer's sentiment towards the object of review.

From an abstract perspective, sentiment analysis has two main components: general to all NLU applications and specific to sentiment analysis. The general component is simply the understanding of thought and intent; the specific component is what differentiates a sentiment analysis model's capabilities from a model trained to perform well in other tasks. Does the sentiment analysis task require a different general understanding of thought and intent than other tasks?

Current sentiment analysis using deep learning approaches often rely on an initial conversion of words to vector representations in some latent semantic space (Medhat et al., 2014), for example using "word2vec" and "bag-of-words" (Giatsoglou et al., 2017) or "BERT" (Xu et al., 2019). Crucially, these word-to-vector mappings (WVMs) are the *implementation* of a model's general understanding of thought and intent. How important is the particular choice of implementation of this word-to-vector mapping to the salience of a model's sentiment analysis abilities? In order to differentiate task-general WVMs from sentiment analysis specific WVMs, we explore 3 different approaches for these word to vector mappings:

1. For general WVMs, we leverage the pre-trained BERT WVM capabilities.
2. For sentiment analysis specific WVMs, we construct end-to-end trained embeddings with and without a smaller RNN/Transformer network and CNN derived embeddings.

These approaches are explored in the context of the Yelp review dataset. The Yelp review dataset consists of a set of reviews and each reviews' corresponding star rating, in the set {1, 2, 3, 4, 5}. The objective is to take an input review and predict its star rating. In order to measure the predictive performance of a model on this task, we decided to use the Mean Absolute Error (MAE) of the model's outputs and the actual star ratings given, with lower values of this measure signifying better performance. Some alternative performance measures include classification accuracy and Mean Square Error (MSE), but MAE additionally describes *how bad* a classification is compared to simple classification accuracy and is more immediately understandable than MSE since MSE depends on the square of the errors. One drawback of MAE is that it isn't salient to whether the model is consistently over-rating or under-rating reviews, only that it's incorrectly rating reviews. Each word to vector mapping approach was constructed and evaluated using MAE and subsequently compared.

Understanding the degree to which sentiment analysis tasks need distinct structures to exploit compared to other NLU tasks will improve the understanding of sentiment itself and improve the approaches for sentiment analysis tasks.

Literature Survey/Related Work

Many previous authors have leveraged pre-trained, task-general WVMs such as BERT or word2vec for sentiment analysis tasks (Xu et al., 2019; Giatsoglou et al., 2017; Kim, 2014)). These models have generally performed well. Furthermore, other authors have argued that because most sentiment analysis tasks use much shorter sequences of text than other tasks e.g. document classification, pre-trained WVMs are able to fill in the missing contextual gaps better than WVMs that are solely trained on the sentiment analysis dataset (Santos & Gatti, 2014; Johnson & Zhang, 2015).

However, training WVMs end-to-end could result in sentiment analysis specific mappings that enable better performance than pre-trained WVMs because they could exploit any relevant structure in text data specific to sentiment analysis that isn't captured by task-general WVMs. Consider the differences between the structure necessary for machine translation and the structure necessary for sentiment analysis. The machine learning task objective is to understand the denotations and connotations of a sequence in one language and reproduce the same denotations and connotations in a different language while the sentiment analysis objective is to understand the *connotative intent* of a sequence.

For example, if something bad happens at a restaurant and someone remarks "This restaurant is just amazing!", the connotative intent of the remark is irony to express the exact opposite of what is denotatively and connotatively on the page. A machine translation model translating this phrase to e.g. Mandarin Chinese could output something such as "这家餐厅真是太棒了!", an exact one-to-one denotative translation of the phrase in English, but the ironic connotative intent is still present to the audience. In this sense, machine translation models may not have to worry about learning any structure or meaning related to irony. However, sentiment analysis models need to understand this connotative intent in order to output that this particular phrase is a negative reflection of the restaurant rather than a positive reflection. Thus, sentiment analysis models need to exploit a different structure or meaning of language that other tasks don't necessarily require, implemented in models using WVMs.

Previous work has been done to apply pre-trained task-general WVMs to sentiment analysis (Lyu et al., 2020). A popular pre-trained task-general WVM is BERT (Devlin et al., 2018), a transformer neural network architecture. We leverage BERT and dense layers as a baseline for our analysis.

The first end-to-end WVM approach taken is to use an architecture similar to BERT, but train it end-to-end with the Yelp dataset.

The second end-to-end WVM approach taken is to use convolutional neural networks (CNNs) to more explicitly exploit local structure, such as for text classification (Johnson & Zhang, 2015). For example, CNNs have been used on top of pre-trained WVMs (Kim, 2014) and used to perform character level analysis on Tweets (Santos & Gatti, 2014). CNNs have also been

seen as an approach in addition to recurrent neural networks (RNNs) that explicitly exploit the sequential structure of text data (Wang, 2015). In this last case, the authors concluded that CNNs and RNNs "provide complementary information for text classification tasks", which could help with the distinct structure necessary for sentiment analysis tasks.

Previous work has been done on incorporating CNNs into WVMs (Johnson & Zhang, 2015) and on using CNNs to model sentences (Kalchbrenner et al., 2014). In this paper, we first replicate these two approaches separately, then combine them into a single model.

Background

In this background section, we start with a discussion of word embeddings and word to vector mappings (WVMs). Then, we briefly cover common network layer architectures used in NLU: recurrent neural networks (RNNs), transformers, and convolutional neural networks (CNNs). Finally, we summarize the important points of Dynamic CNNs as designed by Kalchbrenner et al.

Text comes in the form of a sequence of words, digitally represented by a sequence of characters. The text is tokenized into distinct tokens, such as individual words or punctuation. Each unique token is given a particular numerical value, a categorical mapping of words to numbers in order to discretize and numericize the data. However, even though we have a representation of these words, this representation doesn't necessarily capture the impact, importance, or significance of these words. As a result, these words are typically sent through a word embedding layer, where the vector representation of each word is learned. This embedding can be viewed as a mapping of individual words into a latent word space, where each word is represented by a vector in the latent space, hence the name "word to vector mapping (WVM)".

Some popular network layer architectures used in NLU include RNNs, transformers, and CNNs. RNNs explicitly exploit the sequential structure of text. The particular RNN used is known as a long-short term memory (LSTM) model, which specializes in retaining long distance contextual information because the Yelp reviews are fairly lengthy. Transformers leverage RNNs to further exploit the sequential structure of text, first mapping and encoding data into some space where other layers can be applied, then decoding the data into the output space. CNNs can also be applied to data sequences, but they exploit localized structure within the data instead of any sequential structure like RNNs do.

The particular CNN design used is called the Dynamic CNN (DCNN) by Kalchbrenner et al. Vanilla CNNs typically have a "pooling layer" that combines multiple local values into a single value. The important addition that this DCNN has over vanilla CNNs is that DCNNs don't pool over local values, they pool over all global values in an effort to mitigate the restriction that CNNs only exploit local structure. This way, the DCNN can pay attention to salient values that may appear in different locations within different examples of texts.

Methods/Approach

In order to determine the sensitivity of a model's capacity for sentiment analysis on its WVM implementation, we used BERT and dense layer architecture as a baseline and compared it to several different models.

1. Pre-trained BERT + dense layers
2. Vanilla PyTorch word embeddings trained end-to-end + dense layers
3. Vanilla PyTorch word embeddings trained end-to-end + LSTM + dense layers
4. Vanilla PyTorch embeddings trained end-to-end + CNN + dense layers
5. Vanilla PyTorch embeddings trained end-to-end + DCNN + dense layers
6. Vanilla PyTorch embeddings trained end-to-end + CNN + DCNN + dense layers
7. Vanilla Pytorch Embeddings trained end to end + Small Transformer Encoder + Linear layers

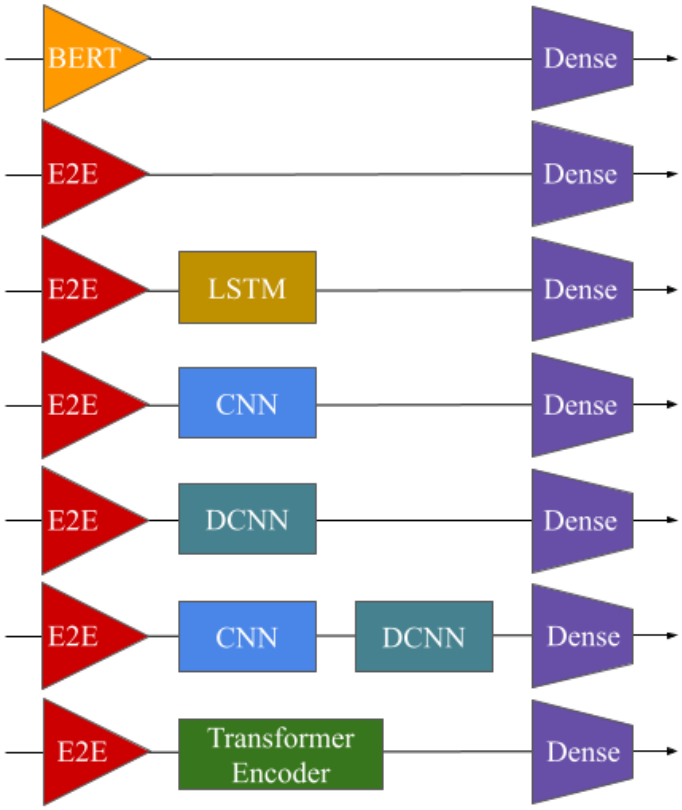

Figure 1: A graphical representation of the model architectures in this paper.

We considered a few variations when creating the BERT + dense model. Most of the small experiments trained on 5000 samples and were validated on 1000 samples.

- We originally were angling to use the vanilla BERT model, but I ran into GPU memory issues when running it on Azure VMs. Retrospectively, this likely could have been resolved by decreasing the batch size or the max token length, but this inspired a search for smaller models that were faster to train.
- Two other BERT variants were experimented on subsequently: Albert (Lan et al., 2020) and DistilBERT (Sanh et al., 2019). Albert is the smallest model at 11M parameters, but its training on the experimental setup took the longest, nearly twice as long as DistilBERT's at 417 seconds. DistilBERT has 65M parameters, but trained the fastest by far at 214 seconds. Vanilla BERT was the largest at 109M parameters, taking 375 seconds to train.
- We settled on using DistilBERT for its training speed, allowing for more iterations and flexibility. We also chose to train it in two different ways: one was to treat the output as a continuous prediction, minimizing MAE and doing the appropriate rounding at test time; the second was to treat this as a classification problem, minimizing cross-entropy. For the continuous model, rounding was elected to be performed only at test time and not part of the forward pass because it allowed for faster training since rounding interfered with the gradients.
- We experimented with a different number of linear layers on top of the BERT transformer, noticing that its effect had a near negligible effect on training time. However, we ran out of student credits to try different numbers of linear layers on training on the entire dataset, so this was not fully explored.

Several other neural network architectures involving CNNs were considered.

- Vanilla PyTorch embeddings trained end-to-end + many CNNs + dense layers
    - This model ended up not being able to fit any training sets that well and it also generalized poorly, most likely due to the fact that even though it contains many CNNs, the model still doesn't have the global contextual capabilities that is necessary.
- Vanilla PyTorch embeddings trained end-to-end + DCNN + LSTM + dense layers
    - This model ended up not learning anything at all, most likely due to the fact that the DCNN already captures much of the sequential structure that the LSTM aims to capture. It was surprising that having two layers that exploit the same structure in sequence lead to worse learning.
- Vanilla PyTorch embeddings trained end-to-end + CNN + DCNN + LSTM + dense layers
    - A similar case to the one previous.

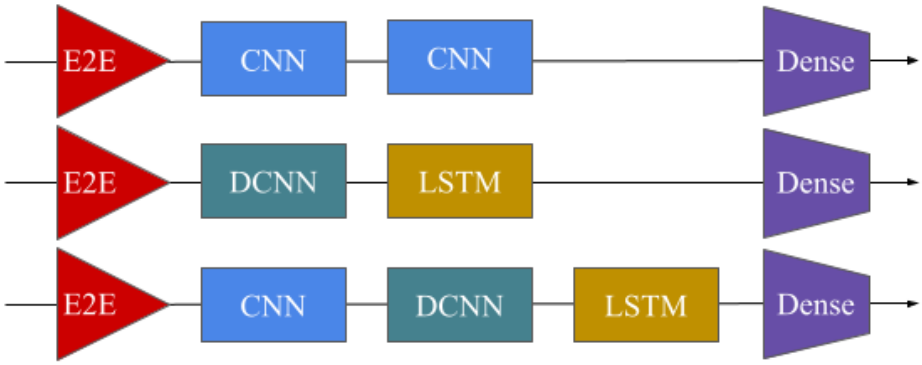

Figure 2: A graphical representation of other neural network architectures involving CNNs that were considered but ultimately not used.

Since the two architectures involving both a DCNN and an LSTM didn't perform well, another architecture that could be considered is to run the word embedding outputs separately through both of these layers, then concatenate their results.

- Vanilla PyTorch embeddings trained end-to-end + Transformer + dense layers
    - On a smaller training set than the full set, it was able to produce very low training losses. However, this tended to be the result of overfitting, as the model gave losses of ~1 on validation sets.

Results

An interesting observation that was made was that in the long run, models with simple dictionary embeddings passed through a few linear layers seemed to perform just as well, or sometimes better than models with a small transformer network. Their overall accuracy was, in most cases, less than that achieved by models that used BERT and related pretrained language models. However, BERT based models tended to have longer training times due to the much larger number of parameters. It is possible that task specific dictionary embeddings trained end to end with a few linear layers provide a "quick and dirty" alternative to the use of large pretrained language models, especially in scenarios where compute and memory are limited.

Name	Model Size (MB)	Test Set 5 MAE	Test Set 5 Accuracy	Test Set 8 MAE	Test Set 8 Accuracy
Bert + Dense (Continuous)	249	0.662	0.226	0.467	0.556
Bert + Dense (Class)	249	0.706	0.378	0.418	0.672
Embedding + Dense	59	0.256	0.796	1.166	0.446

Embedding + CNN + Dense	10	0.972	0.290	1.020	0.442
Embedding + DCNN + Dense	10	0.754	0.384	0.670	0.544
Embedding + CNN + DCNN + Dense	10	0.534	0.518	0.512	0.602

Conclusion/Lesson Learned

As seen in the table of MAEs, standard dictionary embeddings, trained end to end with a CNN and Linear layers seems to exhibit performance comparable to that of the models that use BERT. This would imply that training task-specific dictionary embeddings in this way is a powerful tool that can drastically reduce training times and memory usage. It is possible that, in situations where computing power is limited and the task is one that requires a numeric prediction or classification, word embedding may work better or at least comparably to large language models.

While this task specific embedding works well, it is unclear as to whether the embeddings would be suitable for use in other downstream tasks. On the other hand, BERT has proved to be very effective in such use cases. In conclusion, the solution architecture (BERT or task specific embedding) depends on compute power and memory available and need for use in downstream tasks.

Team Contributions

Albert Tang - 33%. Albert worked on the BERT + dense layers approach, including figuring out how to use the BERT package and testing out variants of BERT e.g. DistilBert, Albert, BERT, etc. Albert also worked on the final submission script and contributed his research to the final paper.

Ram Kripa - 33%. Ram worked on the end-to-end embedding + transformer + dense layers approach, including figuring out the best architectures to use for this dataset. Ram also generated a lot of the initial direction that we had on the project and contributed his results to the final paper.

Brian Yu - 33%. Brian worked on the CNN and DCNN + dense layers approach, including researching CNN NLP architectures and implementing them. Brian also wrote a lot of the final paper.

References

Devlin, Chang, Lee, & Toutanova. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://arxiv.org/pdf/1810.04805.pdf

Giatsoglou, Vozalis, & Diamantaras. (2017). Sentiment analysis leveraging emotions and word embeddings. https://www.sciencedirect.com/science/article/abs/pii/S095741741630584X

Johnson, & Zhang. (2015). Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. https://arxiv.org/pdf/1412.1058v1.pdf

Johnson, & Zhang. (2015). Semi-supervised Convolutional Neural Networks for Text Categorization via Region Embedding. https://arxiv.org/pdf/1504.01255.pdf

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. https://arxiv.org/pdf/1404.2188.pdf

Kim. (2014). Convolutional Neural Networks for Sentence Classification. https://arxiv.org/pdf/1408.5882.pdf

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2020). ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *ICLR 2020*, *1909*(11942). https://arxiv.org/abs/1909.11942

Lyu, Foster, & Graham. (2020). Improving Document-Level Sentiment Analysis with User and Product Context. https://www.aclweb.org/anthology/2020.coling-main.590.pdf

Medhat, Hassan, & Korashy. (2014). Sentiment analysis algorithms and applications: A survey. https://reader.elsevier.com/reader/sd/pii/S2090447914000550?token=46BE5F1F435F883C200E7674B354D5BBBE6CBEBC1B60A912BC0B7721C9B8D63E09519752A10ACBF8B743F758340C80A1&originRegion=us-east-1&originCreation=20210509231254

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS 2019*. https://arxiv.org/abs/1910.01108

Santos, & Gatti. (2014). Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. https://www.aclweb.org/anthology/C14-1008.pdf

Wang. (2015). Semantic Clustering and Convolutional Neural Network for Short Text Categorization. https://www.aclweb.org/anthology/P15-2058.pdf

Xu, Liu, Shu, & Yu. (2019). BERT Post-Training for Review Reading Comprehension and Aspect-based Sentiment Analysis. https://arxiv.org/pdf/1904.02232.pdf