Applying Semantic and Syntactic Information to Contextual Embeddings

Jarem Saunders
COSI MS3, Year 1
jaremsaunders@brandeis.edu

Liam Lewis
CLMS, Year 1
wilewis@brandeis.edu

Peter Williams
CLMS, Year 1
peterwilliams@brandeis.edu

Abstract

This project was conducted to examine the possibility of using text-level representations of linguistic features to incorporate explicit semantic and syntactic features into the embeddings of fine-tuned transformer models. This approach is proposed as a relatively low-resource supplement to improve the performance of pre-trained LLMs. Representations using part-of-speech and semantic role tagging were generated and used to fine-tune DistilBERT models for two different NLP tasks: closed-book question answering and natural language inference classification. Results using this text-based strategy to supplement word embeddings were ultimately found not to improve model performance at this level, showing a neutral to slightly negative effect as more features were included with the fine-tuning text. [1]

1 Project Idea

The main goal of our project is to determine if including syntactic and semantic information in the inputs to a model could improve performance on some task that depends on the semantics of the language used in the task. Many of the recent improvements in models stem from changing something about a model architecture, such as the advent of transformers with self-attention (Vaswani, 2017), or from massively increasing the size of a model and the amount of data the model was trained on (e.g. OpenAI's improvements on versions of GPT). At some point, it isn't feasible to continue increasing the size of a model or the amount of data necessary for training, and so researchers are starting to consider other ways to improve model performance, such as ensuring higher quality of data, or including additional information.

For example, Lee et al. (2021) proposed a method of embedding explicit linguistic information into the word embeddings the transformer model would use. With their methods, they saw significant improvement in the performance of the transformer on readability assessment. It's important to note that readability assessment is a task that is traditionally heavily dependent on the use of linguistic features for categorization, seen in the metrics that are used such as Flesch-Kincaid Grade Level, which uses syllable counting measures to estimate the grade level of a text.

Related research supports the idea that embedding explicit linguistic information in the inputs to a model can improve the performance of that model on a given task (Saunders, 2023), although it seems likely to be limited to certain tasks, and not broadly applicable to all possible NLP tasks. Given this possibility and the desire for more explainability from machine learning, it seems wise to explore novel feature sets that include handcrafted linguistic features for various tasks.

Another method of encoding linguistic information that we have explored is the use of holographic reduced representations to encode word-level linguistic information, such as semantic role or part of speech, in such a way that the resulting sentence-level embedding is entirely compositional, due to the binding and bundling operations. These are one possible method of exploring vector transformations that may lead to improvements in NLP tasks.

Our project was inspired by these methods to experiment with encoding semantic and syntactic information in pursuit of improved performance on two tasks, natural language inference classification and closed-book question answering. These two tasks require some level of semantic knowledge, which makes them promising candidates for this experiment. We are additionally interested in whether these linguistic features can be combined with text data using a fundamental operation like string concatenation to easily integrate into existing pipelines for fine-tuning LLMs with text sequence data.

[1] https://github.com/peterjwms/semantic-embeddings-135

2 Implementation Method

For the purposes of this project, we chose to use the simplest method of encoding linguistic information that we could think of so that this could apply as a sort of baseline to compare other methods against. Given more time and compute power, we would have experimented with some of the methods described above (primarily the hyperdimensional embeddings or other vector transformations), but this project still serves as a baseline and exploration of the necessary steps for future endeavours in encoding explicit linguistic information in models.

The important differences in our model training originate in the preprocessing steps. Our plan was to generate two full datasets, one for each task. Each dataset would have four splits, with varying levels of linguistic information including. The first is the basic dataset with no encoded information. The second and third would have just semantic or syntactic information encoded, respectively, and the final partition would have both syntactic and semantic information encoded.

The dataset used for the natural language inference task was General Language Understanding Benchmark's (GLUE) Multi-Genre Natural Language Inference Corpus (MNLI) (Wang et al., 2019). Closed-book question answering data was obtained from the TruthfulQA dataset (Lin et al., 2022). Said dataset had both a generative and a multiple choice version of the training data, the latter of which was used for this project as a model with semantic knowledge may find correlation between the meaning of the correct answer and the question.

The specific methods employed for extracting linguistic information for each data instance are explained in more depth in the next section. In essence, each token is tagged with part of speech using spaCy (Honnibal et al., 2020), and labeled with semantic roles using VerbNet (Gung, 2024). Each piece is then encoded as part of the string representation of that token by appending it to the end of the token. For each partition of the dataset, only the corresponding information is appended. A full example for the token "Paul" would appear as the following string:

"Paul_PROPN_A0_Agent_Sayer"

After tagging the datasets, we chose to fine-tune DistilBERT (Sanh et al., 2019) separately on the two tasks, inference and QA, on each of the four partitions of their respective datasets, resulting in 8 models total. Each model was trained and evaluated using the HuggingFace transformers and evaluate libraries.

3 Pipeline/Example Walkthrough

3.1 Datasets & Tasks

Each task we examine with this feature embedding strategy has a dataset with 2 strings and a classification task based on identifying the relationship between the two input texts. Before adding our augmented feature representation, we must first extract the relevant text strings for each dataset and their associated labels.

3.1.1 MNLI

Within the MNLI dataset, each training instance includes a single statement described as the "premise" and a second statement described as the "hypothesis," as well as a label indicating whether the hypothesis is entailed by the premise, is unrelated to the premise, contradicts the premise, represented as an integer [0,1,2], respectively. The premise may be a statement or question, while the hypothesis must be a declarative statement, but all are full grammatical clauses, and most are complete sentences.

During processing and feature augmentation, the text for each premise and hypothesis was extracted to pass on to the pre-trained model to create our textual representation of the embeddings.

The full dataset included $\sim$400,000 instances, but due to the very long run times of our data processing strategy, only the validation set of 10,000 instances was used to train our models.

3.1.2 Closed-book QA

The closed-book QA dataset consists of $\sim$800 different questions, each of which has multiple possible answers, only one of which is labeled as correct. The classification task in this case is to identify a correct question/answer pair given an input question and set of possible answers. All the questions and answers are phrased as completed sentences of English. The number of possible answers available for a given question can vary, so the classification is represented as a classification task a number of output classes equal to the maximum number of possible answers, with padding done as necessary to accommodate for the uneven number of possible output choices.

3.2 Data Processing & Feature Encoding

For each dataset, we take the extracted text from each training item and tag the text to identify part of speech and semantic role information using pre-trained models. Using this information, we create 3 new encodings for each string, in which each word in a string is concatenated with a part of speech tag, a semantic role label, or both. Along with the original, unmodified strings, this produced a total of 4 different datasets for each task, or 8 data sets in total.

3.2.1 Verbnet

The Verbnet tags for each word are generated using the SemParse library by James Gung. Using the API of the demo version of the parser, we pass each input sentence of the data sets to the parser to get a representation of the thematic roles of the sentence in the Verbnet format. The parser provides several levels of detail, including theta roles such as agent/theme, argument positioning like A0/A1, and annotated labels for the specific nature of the roles.

For this project, we incorporate all of this information into the embedding schema. For example, the sentence

John gave Mary a cookie.

has three roles, each of which can be given 3 different types of label by the parser, shown in Figure 1.

Verbnet also allows a combination of all of these features represented as a single string, which we use as the basis for our augmented feature representation of semantic features. For each word, the -role, argument, and descriptive label are concatenated to the string using the delimiter "_".

3.2.2 spaCy

For part of speech tagging, we use the standard sentence tagging function in the spaCy python library. Each word of the sentence is tokenized and assigned a POS tag by spaCy, which we concatenate with the input string to create our augmented features. When processing both semantic and syntactic features for a sentence, we first append the POS tag, then the set of semantic labels, using the same delimiter "_".

spaCy is also used to guarantee alignment of tokens between the two parsers. Since parsers may disagree on how to tokenize words with contraction ot possesives, we use spaCy tokenization to split the sentence before feeding the tokens to the Verbnet parser.

3.3 Model Fine-Tuning

For both datasets, we used the unique sets of embeddings created from the dataset to fine-tune 4 different versions of a DistilBERT Classifier Model. Specifically, the QA models fine-tuned DistilBERT to perform multiclass sequence classification using the HuggingFace transformers package for streamlining pytorch projects. The MNLI models fine-tuned on DistilBERT to perform classification of each sentence pair as a contradiction, entailment, or unrelated.

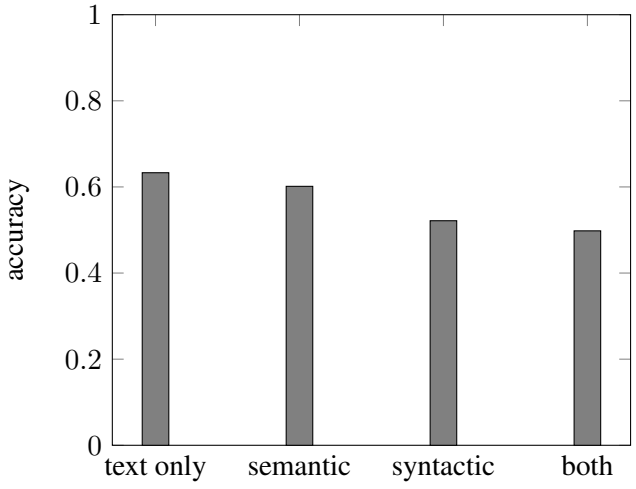
Each model trained with the same hyperparameters for ease and speed for this iteration of the project. Future iterations should experiment with hyperparameter tuning. Each model trained for three epochs with a learning rate of 0.00002, weight decay of 0.01, and batch size of 64.

4 Experiments and Results

4.1 MNLI Task Results

The results for the MNLI model generally showed that the more complex the string representations became, the worse the embedding system performed on the task. No augmented featured embedding was able to outperform the raw text for each metric, and, in fact, the more information was concatenated onto the strings before fine-tuning, the less accurate the final models were. This can be seen in Figure 2, which shows the final development accuracy for each of the models trained.

Fig. 2: Final Development Accuracy for MNLI Models



This same pattern of more complex feature representations leading to worse outcomes is mirrored in Table 1, showing that raw text outperforms our

Text	θ -role	Argument	Label
John	agent	A0	giver
Mary	recipient	A2	entity given to
a theme	A1	thing given	

Figure 1: Example Verbnet parser tags

augmented embeddings on all metrics. It can also be seen that the more features are included, the worse the model does.

Performance for all models was overall unimpressive, with all models falling between 49-63% accuracy. None of the models trained met the 76.9% benchmark for accuracy from the original paper, though it is important to remember that this project used only 10,000 instances for training/validation, while the original dataset was early 400,000 instnces in size.

4.2 QA Task Results

As seen in Table 2 these models performed poorly across the board: only one model exceeded a 0.25 accuracy. This is still better than random chance, as each question in the dataset had at least 4, but usually more options to choose from. Additionally, the GPT-3 models featured in the paper which introduced the dataset only achieved accuracies in the 30s. Despite DistillBERT seemingly not being a great model for this QA task, we do see that the addition of semantic and syntactic information to the embeddings had a negligible effect at best for model performance.

It is worth noting that three of the models achieved the exact same accuracy, which may indicate deeper underlying issues with the training process employed.

5 Error Analysis & Limitations

5.1 Adding Noise to the Data

In the MNLI case, the most likely cause for the decrease in performance for more detailed tags was the fact that our concatenation process created some very long strings that were not at all like actual English text data. While the extra information we encoded was meaningful and possibly helpful to the tasks, it did likely make the actual string data more noisy, as each addition made it less and less like the text that the pre-trained model was built on. Increasing the amount of training examples could help the model better adapt to these new sequence types, as could simplifying the actual strings we are concatenating to create the embeddings. For the MNLI set, this is very feasible, as we trained on less than 3% of available data for this project; for the QA task this would require compiling additional corpora. A more plausible solution would be to move from this textual approach of encoding information to a system that combines the features in a higher-dimensional vector space.

5.2 Time Restriction of Verbnet API

One of the greatest hurdles to this analysis was the long processing time for tagging Verbnet featues for the whole corpus using the API. Because the tagger uses API calls to a Docker image, compute resources were strained during pre-processing, and processing times could take over an hour to process 10,000 instances, which bottle-necked the progress of the research in many ways. Integrating a more efficient lookup for the Verbnet parse could greatly improve the ability to examine the usefulness of semantic role represenatations in feature embeddings.

Conclusion

This project did not produce groundbreaking models for either of the tasks considered. While some routes for improving the models for each task generally were noted, the goal of this project was to determine whether incorporating semantic and/or syntactic information into word embeddings would produce better models for these tasks. In that context, further work on the MNLI inference classification task with these expanded embeddings is not recommended. The QA task had slightly more promising results, but future researchers should consider if the projected increase in accuracy is worth the overhead of adding the tags in preprocessing.

While the this showed little promise for the text-based approach to incorporating semantic and syntactic features, methods that more directly manipulate the feature encoding space and allow the optimizer to adjust weights for the raw text and augmented features separately may be worth consider-

Feature Embedding	F1	Precision	Recall	Accuracy
text only	0.6319	0.6330	0.6330	0.6330
semantic tag	0.5987	0.6013	0.6014	0.6014
syntactic tags	0.5200	0.5246	0.5215	0.5215
semantic and syntactic tags	0.4968	0.5044	0.4976	0.4980

Table 1: Full feature metric scores for models on the MNLI task

Feature Embedding	F1	Precision	Recall	Accuracy
text only	0.1543	0.1199	0.2195	0.2195
semantic tag	0.0790	0.0482	0.2195	0.2195
syntactic tags	0.0955	0.2209	0.2195	0.2195
semantic and syntactic tags	0.1699	0.1465	0.2683	0.2683

Table 2: Full feature metric scores for models on the closed-book QA task

ing. The potential impact on model explainability, as well as the potential for improvement in particular tasks makes this a field worth further exploring.

6 Group Work Division

Our group split up our work fairly evenly. Whenever possible, we tried to work together so that we could have multiple sets of eyes on code both for debugging and ensuring that it was actually doing what we said we were trying to do. We felt that this also helped us each use our strengths while also involving the others so that they could learn and improve in that area. For example, Jarem focused on the semantic tagging process at first because he had the most recent experience with Java, but when there were issues with applying the tags and integrating with spaCy in Python, Liam and Peter were able to help troubleshoot and ensure that processing could run smoothly. Similarly, Peter focused more on setting up the training pipelines using HuggingFace, and then Liam took over to finish building the preprocessing pipeline for the TruthfulQA training. Outside of the coding process, the division of labor was split evenly for the remaining tasks, including project definition and exploration, data collection, presentation preparation, and writing the final report.

References

James Gung. 2024. jgung/verbnet-parser. Original-date: 2019-04-05T03:05:01Z.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Bruce W Lee, Yoo Sung Jang, and Jason Hyung-Jong Lee. 2021. Pushing on text readability assessment: A transformer meets handcrafted linguistic features. *arXiv preprint arXiv:2109.12258*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring How Models Mimic Human Falsehoods. *arXiv preprint*. ArXiv:2109.07958 [cs].

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Jarem Saunders. 2023. Improving automated prediction of english lexical blends through the use of observable linguistic features. In *Proceedings of the 20th SIGMORPHON workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 93–97.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding.