

Comparing Task-Specific Ensembles to General Language Models in Distinguishing Elaborated and Compact Language Codes within CoCA-CoLA

Stanford CS224N Custom Project

Abraham Oshotse
Department of Computer Science
Stanford University
aoshotse@stanford.edu

Abstract

An interesting but heretofore un-operationalized concept in sociolinguistics is the notion of language code elaboration as articulated by Bernstein[1]. More elaborated code is used between individuals who have less assumptions of shared understandings, and so a measure of this concept has broad applicability in studying group behavior, dynamics, and cohesion. The goal of this study is to identify the most promising approach for developing such a measure. I compare a general language model (BERT) to an ensemble approach based on syntactic parsing, which was specifically constructed for it's suitability to the task. In out-of-sample testing on a combined corpus of CoCA and CoLA text blocks, both approaches struggle to successfully perform the task.

1 Introduction

Basil Bernstein first introduced to sociolinguistics the construct of *restricted* (or as we will refer to it, “compact”) and *elaborated* language codes. According to this framework, language and speech patterns between individuals can be categorized based on the extent to which the language used implies taken-for-granted information between the two speakers. Compact code is used between insiders who share assumptions and understandings of a topic, while elaborated code is used when this is not the case, resulting in language that is much more explicit about its referents. The object of this project is to attempt the development of a system of models that can determine the extent to which a phrase or piece of text can be identified as *elaborated* or *compact*. In order to get traction in defining the task computationally, we need concrete examples of elaborated verses compact code. Consider the following examples from Bernstein’s work[2]:

A group of children describe a cartoon strip as such (compact code):

“They’re playing football and he kicks it and it goes through there, it breaks the window and they’re looking at it, and he comes out and shouts at them because they’ve broken it, so they run away and then she looks out and she tells them off”

While another group describes the same strip this way (elaborated code):

“Three boys are playing football and one boy kicks the ball, and it goes through the window, the ball breaks the window, and the boys are looking at it, and a man comes out and shouts at them because they’ve broken the window, so they run away, and then that lady looks out of her window and she

tells the boys off”

To most who read the two sentences above, it becomes immediately clear how one is a “compact” form of the other. In the first sentence the children describe the cartoon as if the listener can see it along with them (shared understanding), and thus reference entities that are known between speaker and listener. In the latter sentence, the children are more explicit, making fewer assumptions of shared understandings beyond what is directly introduced by them into the conversation. A few heuristics stand out (citing the blogger aggslanguage, 2010[2]):

1. Syntax is more formally correct in the elaborated code, but looser in the compact code. There are, for example, more subordinate clauses in the elaborated code, and fewer unfinished sentences.
2. There are more logical connectives like “if” and “unless” in the elaborated code, whereas the compact code uses more words of simple coordination like “and” and “but”.
3. There is more originality in the elaborated code; there are more clichés in the compact code.
4. Reference is more explicit in the elaborated code, more implicit in the compact code: so the compact code uses a greater number of pronouns than the elaborated code

These heuristics suggest the applicability of several types of NLP tasks which ultimately comprise language-code identification: POS-tagging, constituency parsing, and dependency parsing, to name a few. Identifying subordinate clauses, logical connectives, co-reference, and the extent to which dependencies are explicit or implicit seem to be the building blocks in distinguishing elaborated vs compact code.

However, as has been demonstrated in recent NLP advances, large transformer-architecture based language models have shown tremendous ability to solve a wide array of NLP tasks and produce best-in-class performance. In particular, BERT[3] (and it’s many incarnations), has achieved remarkable success, obtaining state-of-the-art results on several GLUE[4] tasks. Therefore, the aim of this work is not only to develop a model that can successfully learn representations of compactness and elaboration, but to also establish a benchmark in comparing an ensemble approach of explicitly structure-based parser models to a large, pretrained transformer-based language model.

2 Approach

2.1 Overview

The BERT approach is end-to-end; it consists of encoding input sequences by finetuning the pretrained base-BERT on the specific classification task (i.e. by attaching a BERT encoder as a layer in a neural classifier). In this approach, with each pass through the data set the embeddings are updated at the same time as the classification parameters (although to a substantial extent the embedding representations have been pre-learned in masked language modeling and next sentence prediction pre-training). The trained model is then capable of producing a prediction by encoding an input and producing a classification.

On the other hand, rather than being end-to-end, the ensemble approach is discrete. It is broken down into three distinct tasks: POS-tagging, syntactic parsing, and neural classification (where syntactic parsing itself is two distinct tasks: dependency-parsing and constituency parsing). The models learn weights by training on their own sub-tasks, independent of the overall task, and independent of each other (except for the dependency and constituency parsers, which are jointly trained according to a compound loss). In order to produce a prediction, output from each stage feeds into the other: tags from the POS-tagger are supplied to the syntactic parser along with the text input; and then POS-tags and parsed sentence structure is sent to the neural classifier.

For each task the models are trained by receiving an input sequence consisting of two text blocks, each of which has an elaboration “score”, according to how elaborated the language of the text is. The models are trained with the objective of correctly classifying the more elaborated text block. An illustration of the schema for each approach is given below.

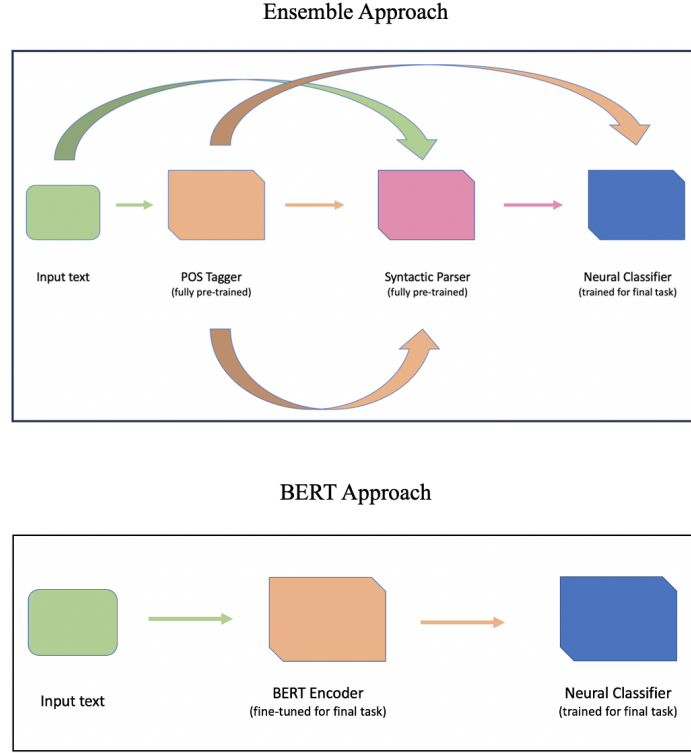


Figure 1: Simplified illustration of ensemble and BERT approach.

2.2 BERT / End-to-End Model Approach

For this task the BERT implementation from Huggingface (BertForSequenceClassification) is utilized. This implementation includes a word embedding layer, with position and token-type embeddings, 11 layers, and a final dense output layer that maps 768 inputs (the length of the embedding) to 1 output for the classification.

The training data consists of 18,653 examples of text block pairs (210 unique training text blocks and 75 unique test text blocks) with elaboration scores for each unique block (such that each training example has a “left” and “right” score), and a label of 1 if the right block is more elaborated but 0 otherwise. Each text block is tokenized and encoded with “[CLS]” prepended to the start of the sequence and “[SEP]” appended to the end. This is done separately for the 18,653 left and 18,653 right text blocks. Each block within the two sets is then padded, with attention masks applied. After this, the two sets are concatenated element-wise (with the CLS token from the right text block dropped) so that each sequence within the concatenated set is comprised of two separate (but concatenated) text blocks, beginning with a single [CLS] token and including a [SEP] token for each constituent block. As a result, each sequence within the concatenated set contains one CLS token and two SEP tokens. This is done so that the text blocks are encoded separately while still constituting a single example.

The training sample is then split into training and validation (9 : 1 ratio), and training commences with batch sizes of 4 (the largest allowed by our system’s memory constraints). The training was conducted with Adam-W optimization since regular Adam optimization has been shown to generalize less well [5]. We retained the fine-tuning parameters recommended by Devlin et al. [3]: learning rate of 0.05, an epsilon value of $1e^{-4}$, but we opt for much longer epochs than they suggested, since our task is so different from the original pre-training (55 epochs vs their suggestion of 4). With the processing of each batch, we apply gradient clipping with a clipping-value of 1. For the task, the appropriate loss function is binary cross entropy. Due to time constraints we were unable to conduct a thorough hyperparameter search.

The end-to-end model consists of a fully-connected dense layer which takes as its input the tokenized sequence of concatenated text blocks and applies pre-trained BERT-base encoding to each batch. The encoding produces a 768 item vector which is passed to the first hidden layer that outputs a 350-dimensional vector, and then dropout ($p=0.5$) and ReLU are applied. The result is passed to a second hidden layer which outputs a 100-dimensional vector, and once again dropout ($p=0.25$) and ReLU are applied. That result is then passed to a third hidden layer which produces a 25-dimensional vector, and lastly is passed to the final layer which produces a logit value for each example in the batch. With the processing of each batch the optimizer updates parameter gradients and takes a step in the loss-space towards the direction of lower cost. Since the optimizer includes parameters of the BERT model as well as the fully connected layer, these are updated in tandem.

2.3 Grammatical Structure / Ensemble Model Approach

The ensemble model follows the method of Mrini et al[6] for syntactic parsing, coupled with a fully connected deep layer. In their paper, Mrini et al. introduce and apply a new form of self-attention where attention heads represent labels. They refer to this as the “Label Attention Layer”. Their approach achieves best-in-class performance on syntactic parsing tasks. We apply their fully pre-trained parser which includes a POS-tagger.

Similar to the end-to-end approach, the training data consists of 18,653 examples of text block pairs (285 unique text blocks) with elaboration scores for each unique block (such that each training example has a “left” and “right” score), and a label of 1 if the right block is more elaborated but 0 otherwise. The POS-tagger is applied to the 285 unique text-blocks, and then the tags and text-blocks are fed to the syntactic parser (the dependency parser and constituency parser use distinct sets of POS tags). The parser outputs three separate results for each text block: dependency head information (which maps a token in the sentence to a head in the dependency structure), dependency label information (the POS-tags of each token in the sentence, except the *root* token), and the constituency tree-structure.

These outputs are preprocessed in order to create a uniform encoding structure to feed to the deep layer. A structured dataset is produced as such: for each class of parser outputs, the total unique (dependency and constituency) tags in the corpus constitute features. The total amount of features for the dataset is $2N_{DL} + N_{CL}$ where N_{DL} is the total number of unique dependency-labels and N_{CL} is the total number of unique constituency labels observed in the corpus. In our corpus $N_{DL} = 44$ and $N_{CL} = 63$, making the final output encoding a 151-feature dataset.

The values of this structured dataset are determined as follows. For each of the 63 constituency-tag features, for a given example text block, the integer value for the constituency-tree layer where the tag appears in the text block populates the dataset in the corresponding constituency-tag column and example text row. For the first set of dependency-tag features, dependency tag values are similarly populated based on the constituency-tree layer where the tag belongs. For the second set of dependency-tag features, the dependency-head value is what populates the entry instead.

For the final classification layer, training examples consist of paired text blocks where the classification objective is to correctly identify the more elaborated block. The input for each block in the pair is the element-wise difference in the 151-feature encoding from the parser output, between the two text blocks. Hyperparameters were determined by random search along several dimensions: depth of network, number of nodes per layer, activation function for inner layers, regularization factor, dropout rate, learning rate, and gradient clipping value. The hyperparameter values of the model that had highest validation-set performance (out of 25^1) are as follows. The classification model consisted of three hidden layers of 5, 83, and 8 activation units, respectively. For each layer, L1 regularization (factor value = $1.44e^{-4}$), dropout ($p=0.1$), and a swish activation function are applied. We apply Adam optimization [7] with a learning rate of 0.005 and gradient clipping (with clip-norm value of 0.1).

¹Given that we could not adequately search the hyperparameter space for the BERT model (due to time constraints), we placed limits on how thoroughly the space was searched for the ensemble model.

3 Experiments

3.1 Data

The data used for training, validation, and testing consist of a roughly 7 : 3 ratio of CoCA-CoLA:

1. Random excerpts from the “Spoken” genre of The Corpus of Contemporary American English[8] (CoCA)
2. Random excerpts from The Corpus of Linguistic Acceptability[9] (CoLA)

To obtain a data set suited for the task, it was necessary to construct my own training corpus. Blocks of text were sampled from each of the two corpora (where a block of text is one or more sentences of variable length, usually no more than 5 sentences with modal value being 1) and edited (if need be) to meet a minimum level of grammatical correctness and readability². For each block sampled, an *elaborated* version was produced along with a *compact* version. Usually, if the original text block was fairly elaborated, a compact version was produced (and vice versa). Sometimes, though, the entire text block was overhauled to produce both versions and on occasion (when possible), multiple versions of the text block were produced across a range of elaboration levels.

For each text block an elaboration score was chosen based roughly on heuristics according to Bernstein – Elaborated Restricted Code (1971)[2]. The highest possible elaboration score is 100, where > 50 indicates an elaborated block of text and ≤ 50 indicates a compact block. Points were deducted when the main subject or object being referenced in the text block is not explicitly defined within the context (text block), and this is conditional on the “coverage” of entities in the text (i.e. the proportion of entities in the text with explicit reference, where explicit reference means noun or proper noun usage as opposed to pronoun usage, and where noun phrases are usually considered more elaborated than proper nouns - depending on the particularities of the text). Moderate point deductions occurred for excessive use of simple coordination (“and”, “but”), and minor point additions/deductions occurred for syntactic correctness/incorrectness. To some extent these points were also allocated according to an intuitive “feel” of what constitutes compactness. This is so that a model can be successfully trained to represent and identify that feeling. An actual example excerpt³ from CoCA is below:

"And we're going to drill down here in the next couple of months and determine exactly what those are and how we're going to go about implementing these."

The above text block is considered compact and given an elaboration score of 29. More elaborated versions are created with scores reflecting the higher levels of explicitness of the speech:

Version 1: "And we're going to drill *in the valley* in the next couple of months and determine exactly what those are and how we're going to go about implementing these."

Version 2: "And we're going to drill *in the valley* in the next couple of months and determine exactly what *the issue is* and how we're going to go about implementing these."

Version 3: "And we're going to drill *in the valley* in the next couple of months and determine exactly what *the issue is* and how we're going to go about implementing *the solution*."

Version 1 is still compact, but the score rises to 45 since there is some elaboration of one of the previous pronouns (“here”). For the same reason, Version 2 rises to 60 and Version 3, with no un-elaborated pronouns, rises to 70 (although for other reasons the score is still somewhat depressed). Versions 2 and 3 are considered elaborated.

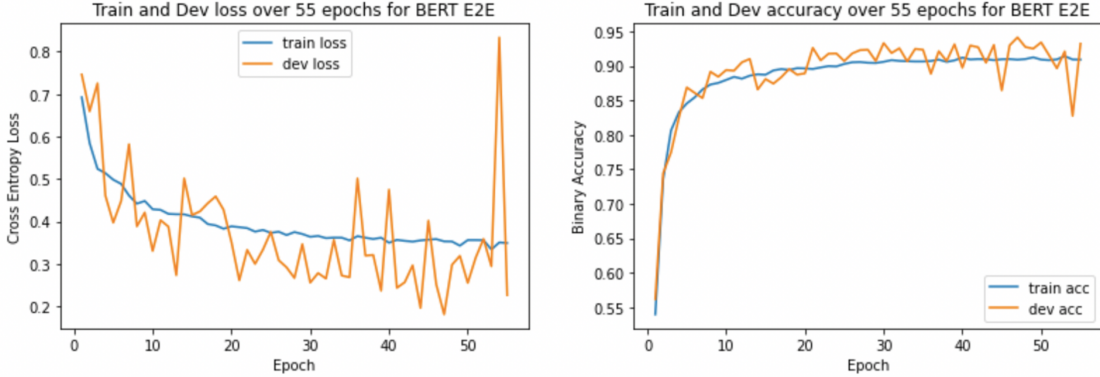
²This was usually necessary for the COCA text since transcribed speech is often badly formatted. The editing process was done carefully so that the unique structure of the text block was not badly compromised - a minimum amount of editing necessary for the maximum amount of clarity.

³A rather cheeky example observed in CoLA: “The belief that syntactic theory reveals the inner structure of sentences emboldened the already much too cocky professor.” The thesis of this study is that a syntactic language model should outperform a general one!

3.2 Details and Results

3.2.1 BERT End-to-End Results

After 55 fine-tuning epochs the BERT model exhibits the training loss shown in Figure 2.



The model is slow to train, and so the training time was limited to 55 epochs. The model achieved up to 90% accuracy in validation sets. This initially raised strong concern that the model is over-fitting, particularly because the limited training data are constrained in how much they can vary. Despite the numerous examples, the amount of unique text blocks is rather small and so at this point the model appears to have fit the data rather easily. This is especially possible given that the train and validation sets exhibit some cross contamination of text blocks⁴. When evaluating the performance of the BERT classification model on a test data set where there is no contamination between any text block in training/validation or test, there is a steep drop-off in classification ability.

The results for performance on the test sample are evaluated using raw prediction accuracy, Matthew’s correlation coefficient, and F1. They are shown below:

Table 1: BERT model performance, out-of-sample.

Model	Test Accuracy	Matthews Corr. Coef.	F1 Score
BERT base (9.35k examples)	52.3%	0.0497	0.46
BERT base (18.7k examples)	52.9%	0.0606	0.48

Results suggest that at this stage our general language model does a poor job in the out of sample classification task. This strongly supports the inclination that our sample of unique text blocks must be greatly expanded for the model to learn representations that generalize well out of sample.

3.2.2 Parser Ensemble Results

After 100 training epochs the parser ensemble exhibits the training loss shown in Figure 3.

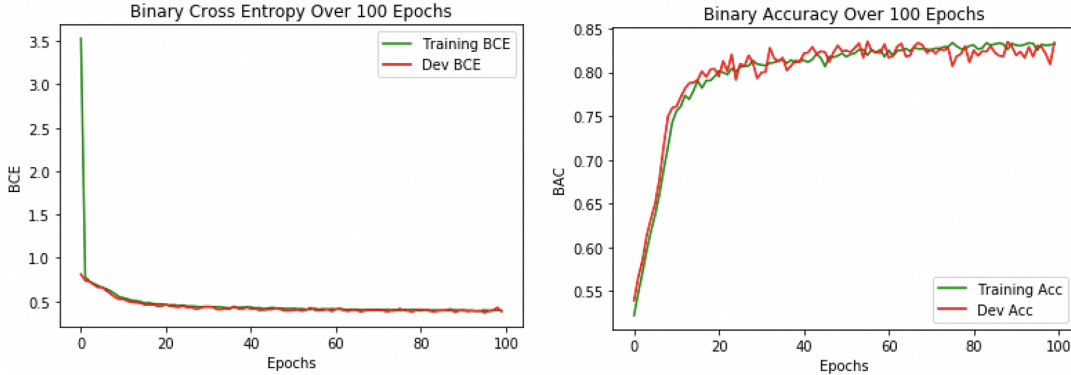
The loss drops rather quickly, roughly plateauing at around epoch 50. Accuracy seems to follow a similar pattern. By epoch 100 the validation loss is 0.381 and validation accuracy = 83.38%.

The results for performance on the test sample are shown in Table 2:

Table 2: Parser model performance, out-of-sample.

Model	Test Accuracy	Matthews Corr. Coef.	F1 Score
Parser ensemble (9.35k examples)	55.89%	0.1236	0.49
Parser ensemble (18.7k examples)	59.42%	0.1236	0.60

⁴No example is actually shared between the two (where an example is a unique pair of text blocks), but text blocks are shared between the two sets.



Results demonstrate a clear advantage for the ensemble model, especially when the training examples are doubled. However, the ensemble model performs only moderately better than chance. Further, given the trajectory of loss in the BERT model compared to the parser, it is potentially the case that the lift in performance between the two is all due to the difference in training time.

3.2.3 Parser Ablation Results

The parser ablation study is conducted by considering three separate models that each represent the parser ensemble with a different component removed. The first model removes the POS tagging, using only constituency-tree and dependency-head information for the sentence encodings. For a given block of text, POS info is removed by collapsing all dependency and constituency tags within a level of the constituency tree into one, so that the information retained for each text block is the number of tokens at each level in the constituency tree, as well as the number of constituency levels. This is done for the set of features that map both dependency POS tags and constituency POS tags to constituency tree levels. Additionally, the dependency head associated with each token in the text-block is preserved since it does not contain POS information. This results in an embedding of 119 features.

Removing constituency information involves removing all features associated with the constituency-tree - namely, the levels of the tree and the association of tags with levels. The constituency tags are retained as indicator dummies for each text block, and dependency tags are counted this way as well, which preserves POS information but not constituency tree-level information. Finally, the dependency heads are retained to preserve dependency parsing information. Therefore, like the unablated model, the constituency-ablated model also has 151 features. The last ablated model includes POS information and constituency information but excludes dependency parsing information. This is accomplished by dropping dependency head features but retaining everything else. The dependency-ablated model has 107 features.

For all three ablated versions of the model, the results for performance on the test sample are shown below (after training on the full 18,653 size data set):

Table 3: Out of sample performance for ablated models (18.7k training examples)

Model	Test Accuracy	Matthews Corr. Coef.	F1 Score
Parser ablation 1 (Parts of Speech tags removed)	52.76%	0.0550	0.58
Parser ablation 2 (Constituency structure removed)	64.25%	0.2849	0.65
Parser ablation 3 (Dependency head info removed)	54.74%	0.0947	0.58

In line with expectations, removing dependency head information and POS information degrades model performance. However, counter to expectations, eliminating constituency information appears to improve model performance by roughly 8%. These findings were robust to repetition (over 20 runs, best performing constituency-ablated models outperformed non-ablated models). Future work will explore why this occurred. Current results suggest that POS tags are the most important component,

as model performance falls furthest when this is removed.

3.2.4 Combining BERT and Syntactic Parsing

The performance of the model when POS tags are removed imply that a lot of information about language code is imbued in the text’s terms. This is most obvious in the importance of pronouns as a classification feature. The model we can best apply to exploiting the information in terms is perhaps a word embedding model, which the BERT embeddings include. I accomplish this by creating an ensemble of the BERT model and the parser. This is done by encoding input sentences both ways and then passing a concatenated vector of encodings into a neural classifier (with the exact same architecture of the one trained with the BERT model). The BERT encodings and neural weights update with each step of the optimizer but the parser encodings do not. This is done on the full model and the constituency-ablated model, with results shown in Table 4.

Table 4: Out of sample performance for combined models (18.7k training examples)

Model	Test Accuracy	Matthews Corr. Coef.
BERT and Constituency-ablated Parser	49.70%	0.0
BERT and Full Parser	54.20%	0.3214

Combining the models appears to especially degrade performance, particularly in the case of the ablated parser. This is potentially due to the the limited training epochs and data set size.

4 Future work

There is still a lot of future work cut out. First, the corpora of text blocks needs to be greatly expanded. Next, the architecture of the BERT approach must be refined. Currently, we encode each text block in a sequence ([CLS], [SEP], [SEP]), that was pre-trained on largely different tasks such as next sentence prediction. This may negatively affect the suitability of the text block encodings. Using two separate BERT encodings (of the form [CLS], [SEP], each), and then feeding the classifier concatenated vectors of these may improve the classification performance. Next, a custom loss function will implemented. The custom loss that is currently being considered is a nonlinear function which weighs loss due to miss-classification by some factor of the amount by which the miss-classification was off by:

$$L_i = \begin{cases} 0, & \text{if } \hat{y} = y \\ f(|m|), & \text{if } \hat{y} \neq y \end{cases} \quad (1)$$

Where $\hat{y}$ is the model prediction, y is the ground-truth value, and m is the difference in elaboration value between the elaborated and compact text in the training example. Finally, the training and validation text distributions will change. The validation set will be a set of text-block pairs that share a much smaller overlap in text-blocks. Model selection based on performance on this set must depend on the model’s ability to generalize to out-of-sample data. Our relatively tight overlap in text blocks between train and validation may have resulted in suboptimal model selection.

References

- [1] Basil Bernstein. *Class, Codes and Control: Volume 1 – Theoretical Studies Towards A Sociology Of Language*. Routledge Kegan Paul Ltd, 1971.
- [2] aggslanguage. Bernstein – elaborated restricted code (1971).
- [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [4] Julian Michael Felix Hill Omer Levy Alex Wang, Amanpreet Singh and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

- [5] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.
- [6] Khalil Mrini, Franck Deroncourt, Trung Bui, Walter Chang, and Ndapa Nakashole. Rethinking self-attention: An interpretable self-attentive encoder-decoder parser. *arXiv preprint arXiv:1911.03875*, 2019.
- [7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
- [8] Mark Davies. The corpus of contemporary american english. *www.english-corpora.org/coca/*, 2008.
- [9] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.