Representing the Unification of Text Featurization using a Context-Free Grammar

Doruk Kilitçioğlu and Serdar Kadıoğlu
Artificial Intelligence Center of Excellence
Fidelity Investments, Boston, USA
{firstname.lastname@fmr.com}

Abstract

We propose a novel context-free grammar to represent text embeddings in conjunction with their various transformations. We show how this grammar can serve as a unification layer on top of different featurization techniques, and their hybridization thereof. The approach is embodied in an open-source library, called TEXTWISER, with a high-level user interface to serve researchers and practitioners. The goal of TEXTWISER is to enable rapid experimentation with various featurization methods and to serve as a building block within AI applications consuming unstructured data. We highlight several key benefits that are desirable especially in industrial settings where rapid experimentation, reusability, reproducibility, and time to market are of great interest. Finally, we showcase a deployed service powered by TEXTWISER as a proof-of-concept enterprise application.

Introduction

Unstructured text is a common source of input data for many modern machine learning applications from finance to healthcare and e-commerce. Given the vast amounts of unstructured data, applications turn to various text featurization techniques from Natural Language Processing (NLP). However, there exists no silver bullet, and it typically remains unknown which featurization technique, or which combinations, would provide the best performance for the downstream application at hand. Depending on the specific task, which might range from ranking to classification and similarity matching to personalization among many others, practitioners turn to experimentation in the pursuit of finding the best approach.

Consider, for example, a scenario in which we would like to find the pairwise similarities between the contents of a set of articles. A reasonable and commonly applied approach would be to select a distance measure and then compute distances among articles based on their text features. This leaves us with some algorithmic choices. On the one hand, we can start with relatively simple counting and frequency-based approach such as Term Frequency-Inverse Document Frequency (TF-IDF)(Jones 1972). On the other hand, we can employ drastically different techniques based on word embeddings such as Word2Vec (Mikolov et al. 2013), Doc2Vec (Le and Mikolov 2014), or even more sophisticated language models such as BERT (Devlin et al. 2018), or one of its many of the variants, e.g., RoBERTa (Liu et al. 2019), DistilBERT (Sanh et al. 2019), ALBERT (Lan et al. 2019) etc. When faced with this challenge, data scientists have to experiment with many techniques and select the best performing one while also considering other factors such as the inference time, simplicity, maintainability, availability of specific hardware, deployment constraints, and reproducibility. The task becomes even more involved when we consider not only the different featurization methods but their combination thereof. For instance, we could represent each article using a concatenation of two or more methods, or their transformation using dimensionality reduction or decomposition techniques such as non-negative matrix factorization (Lee and Seung 2001) and singular value decomposition (Golub and Reinsch 1971).

Recent developments in the open-source NLP community provided practitioners with a rich set of tools and libraries to combat this challenge. Popular examples include NLTK (Loper and Bird 2002), GENSIM (Řehůřek and Sojka 2010), and more recently SPACY (Honnibal and Montani 2017), ALLENNLP (Gardner et al. 2018), FLAIR (Akbik, Blythe, and Vollgraf 2018) and HUGGINGFACE TRANSFORMERS (Wolf et al. 2019). Thankfully, most of these libraries also offer pre-trained models to speed up application development. Still, the availability of various techniques in standalone, isolated libraries does not provide an immediate solution for our scenario to find the best content similarity approach. The task becomes working with many *different* tools and building a custom layer on top to find the best approach. When the scenario changes in a new use case, we have to re-do this work from scratch. Even worse, this is an effort that *every* practitioner has to repeat for *each* use case. Notice also that there is no guarantee of reproducibility between two different implementations and experimentations. Moreover, as one of the fastest moving research areas, with new NLP methods and tools being introduced on a rapid basis, data scientists and non-experts need to get familiar with new technology and extend their experimentation setup with another tool. This is exactly the problem we want to solve in this paper to improve applied AI innovation and deployment of AI systems.

With this goal in mind, we built TEXTWISER[1], an open-source Python library to provide a unified framework for text featurization based on a rich set of methods taking advantage of pre-trained models. In its simplest form, it is a wrapper around other state-of-the-art NLP libraries such as Flair (Akbik, Blythe, and Vollgraf 2018) and gensim (Řehůřek and Sojka 2010), but its overall contributions go beyond that:

- **Grammar of Embeddings:** We introduce a context-free grammar to represent the unification of text featurization. This allows designing embeddings from well-defined components systematically and can even form arbitrarily complex ones.
- **Rich Set of Embeddings:** A wide range of embeddings and transformations to choose from.
- **Fine-Tuning:** The implementation supports PyTorch backend natively, and as such, it retains the ability to fine-tune featurization for downstream tasks, a highly-desirable feature for niche performance. When the resulting fine-tunable embeddings are used subsequent training steps, the featurization can be optimized further for specific applications.
- **Parameter Optimization:** The library is interoperable with the standard scikit-learn pipelines for hyper-parameter tuning and rapid experimentation. All underlying model parameters are exposed to the user.
- **GPU Native:** The library is built with modern architecture and GPUs in mind. If it detects available hardware, the relevant models are automatically placed on the GPU.
- **Reproducibility:** Equipped with the formalism of a well-defined grammar, featurization methods are transferable using human-readable schema throughout the application life-cycle from the training phase to model deployment and inference.

In the remainder of this paper, we showcase the high-level design of TEXTWISER with several examples and provide details of the functionality. Finally, we demonstrate an proof-of-concept application powered by TEXTWISER and deployed as a service to facilitate the modeling and deployment of AI systems.

High-Level Usage Examples

Let us start by presenting a simple usage example to make the idea behind TEXTWISER more concrete.

```
# Unstructured Text Data
documents = ["Some document",
             "More text"]

# Example Embedding - I: Counting based
model = TextWiser(Embedding.TfIdf())

# Example Embedding - II: Doc vector
model = TextWiser(Embedding.Doc2Vec())

# Text Featurization
features = model.fit_transform(documents)
```

[1]https://github.com/fidelity/textwiser

In this example, given a set of text documents, TEXTWISER provides a high-level interface to access embeddings in a unified manner. The first example is the simple counting/frequency-based approach (Jones 1972), and the second example is the document embeddings from (Le and Mikolov 2014). Notice how both embeddings share a common training and transformation step. Under the hood, each approach performs the necessary training encapsulated in the *fit_transform* method. This method should be familiar to those with a general background in data science and the well-known Scikit-learn library (Pedregosa et al. 2011). The training operation might be built-in, delegated to another library, or come from a pre-trained model. While power users might want to explore some of these configurable parameters, it is not required for the out-of-box usage pattern showcased in this example.

Transformation Example

In practice, it is common to apply transformation and dimensionality reduction techniques on top of text featurization. This helps create compact representations to be consumed by machine learning models. Similarly, word vector embeddings need pooling operation to create document-level features. TEXTWISER captures the semantics of these operations as a chain of one or more transformation steps, as demonstrated in the next example:

```
# Embeddings and Transformations
tf_idf = Embedding.TfIdf()
doc2vec = Embedding.Doc2Vec()
lda = Transformation.LDA(n_components=30)
svd = Transformation.SVD(n_components=10)

# Example - I: Single Transformation
model = TextWiser(tf_idf, lda)

# Example - II: Chain of Transformations
model = TextWiser(doc2vec, [lda, svd])
```

The first case applies Latent Dirichlet Allocation (Blei, Ng, and Jordan 2003), a useful technique for topic modeling, on top of the TF-IDF features, while the second case transforms the representation further using singular-value decomposition. Notice how different embeddings and transformations are interchangeable with each other leading to various combinations.

A Context-Free Grammar of Embeddings

As shown in the usage examples, the main philosophy behind TEXTWISER can be captured neatly with the following formula:

$$\text{TEXTWISER} = \text{EMBEDDING} + \text{TRANSFORMATION(S)}$$

The formula itself raises an important compatibility question: which combinations of embeddings and transformations lead to *valid* featurization techniques? For instance, we cannot apply a maximum pooling transformation unless the embedding space creates word vectors. Our work's unique contribution is a context-free grammar (Chomsky 1956) that

Algorithm 1 Text Embedding Unification Language of TEXTWISER Interface

⟨*start*⟩ ::= ⟨*embed_like*⟩ | ⟨*merge*⟩

⟨*embed_like*⟩ ::= ⟨*embed_option*⟩
| [⟨*embed_option*⟩ , dict]

⟨*embed_option*⟩ ::= ⟨*bow*⟩ | ⟨*doc2vec*⟩ | ⟨*tfidf*⟩ | ⟨*use*⟩

⟨*merge*⟩ ::= { ⟨*transform*⟩ : [⟨*start*⟩ , ⟨*transform_list*⟩] }
| { ⟨*transform*⟩ : [⟨*word_like*⟩ , ⟨*pool_trfm_list*⟩] }
| { ⟨*concatenation*⟩ : [⟨*concat_list*⟩] }

⟨*transform_list*⟩ ::= ⟨*transform_like*⟩
| ⟨*transform_like*⟩ , ⟨*transform_list*⟩

⟨*transform_like*⟩ ::= ⟨*transform_option*⟩
| [⟨*transform_option*⟩ , dict]

⟨*transform_option*⟩ ::= ⟨*lda*⟩ | ⟨*nmf*⟩ | ⟨*svd*⟩ | ⟨*umap*⟩

⟨*word_like*⟩ ::= ⟨*word*⟩ | [⟨*word*⟩ , dict]
| word_option | [word_option , dict]

⟨*word_option*⟩ ::= (see Table 1 word embeddings†)

⟨*pool_trfm_list*⟩ ::= ⟨*pool_like*⟩
| ⟨*pool_like*⟩ , ⟨*transform_list*⟩
| ⟨*transform_list*⟩ , ⟨*pool_like*⟩
| ⟨*transform_list*⟩ , ⟨*pool_like*⟩ , ⟨*transform_list*⟩

⟨*pool_like*⟩ ::= ⟨*pool*⟩ | [⟨*pool*⟩ , dict]

⟨*concat_list*⟩ ::= ⟨*start*⟩ | ⟨*start*⟩ , ⟨*concat_list*⟩

⟨*foo-bar*⟩ ::= '`foo-bar`' # Omitting trivial terminals

defines the language of valid featurization techniques systematically. The idea is to treat each instantiation of embedding and transformation combinations as a *sentence* subject to language membership against a grammar.

More formally, Algorithm 1 presents the specification of our context-free grammar. We omit trivial string conversions from non-terminal to terminal symbols, but the complete language can be found in our repository. We also use the short-hand *dict* to mean a valid dictionary with key-value pairs for brevity. This dictionary passes hyper-parameter arguments to underlying implementations. To the best of our knowledge, the proposed grammar here can accommodate most, if not all, NLP featurization techniques that exist in the state-of-the-art.

TEXTWISER internalizes this grammar through a particular embedding, called *compound* embedding. The compound embedding operates on grammar-based specifications given as a JSON schema. In fact, there exists a compound embedding corresponding to *any* TEXTWISER instantiation including our previous usage examples (since the grammar defines the entire language).

At the heart of our grammar specification lies two main production rules, as part of the $\langle merge \rangle$ non-terminal symbol in Algorithm 1:

Transform Operation: This production rule defines a list of operations, the first of which should be an Embedding (or a valid compound embedding), while the rest should be Transformation(s). This rule is a direct translation of our formula. Internally, embeddings have access to raw text, turn them into vector representations, and then Transformations operate on these vectors. In PyTorch (Paszke et al. 2019) terminology, this is equivalent to using *nn.Sequential*.

Concatenation Operation: This operator defines a concatenation of multiple embeddings (or, as before, valid compound embeddings). The concatenation can be done both at word and sentence level. In PyTorch terminology, this is equivalent to using *torch.cat*.

Grammar-based Compound Embedding

In theory, grammar representation can form arbitrarily complex combinations of embeddings and transformations. In a later section, we discuss how to exploit this property to design non-trivial combinations within a process similar to neural architecture search (Stanley and Miikkulainen 2002), or, more generally, instance-specific algorithm configuration (Kadioglu et al. 2010). In practice, the compound embedding consumes a JSON schema as follows:

```
# Grammar-based Schema
schema = {"transform": [{"concat": [
    {"transform": ["word2vec", "pool"]},
    {"transform": ["elmo",
        ["pool",
         {"pool_option": "mean"}]]},
    {"transform": ["tfidf", "nmf"]}]
  }, "svd"]}

# Compound Embedding
model=TextWiser(
    Embedding.Compound(schema))
```

This example takes advantage of three different embeddings: i) Word2Vec embedding with the default max-pooling at the document level, ELMo (Peters et al. 2018) embedding with mean pooling, and TF-IDF embedding with an NMF transformation. These three embeddings are concatenated, and then the resulting vector is decomposed using SVD.

Rich Set of Embeddings and Transformations

Table 1 lists the available Embeddings in TEXTWISER. The *Pre-Trained* column indicates whether there exists a pre-trained model that can be leveraged, and the *Fine-Tuning* column shows whether the Embedding can be fine-tuned for downstream tasks. We elaborate on fine-tuning in the next section. Similarly, Table 2 lists the available Transformations. The *Grad* column shows whether a Transformation can propagate gradients back, and the *Fine-Tuning* column shows whether the Transformation itself is fine-tunable. Interestingly, note that a Transformation such as pooling can

Embeddings	Pre-Trained	Fine-Tuning
Bag of Words (BoW)	✗	✗
TF-IDF	✗	✗
Doc2Vec	✗	✗
Universal Sentence Encoder	✓	✗
Word2Vec†	✓	✓
Character†	✗	✓
BytePair†	✓	✓
ELMo†	✓	✗
Flair†	✓	✗
BERT†	✓	✓
OpenAI GPT†	✓	✓
OpenAI GPT-2†	✓	✓
TransformerXL†	✓	✓
XLNet†	✓	✓
XLM†	✓	✓
RoBERTa†	✓	✓
DistilBERT†	✓	✓
CTRL†	✓	✓
ALBERT†	✓	✓
T5†	✓	✓
XLM-RoBERTa†	✓	✓
BART†	✓	✓
ELECTRA†	✓	✓
DialoGPT†	✓	✓
Longformer†	✓	✓

Table 1: Available Embeddings in TEXTWISER with the existence of pre-trained models and the ability for fine-tuning. The symbol † denotes word embeddings.

propagate gradients back without being fine-tunable as it does not have any parameters to tune.

Under the hood, the library integrates several other state-of-the-art NLP libraries to make Embeddings and Transformations available to the end-user in a seamless fashion. The libraries integrated in TEXTWISER include, but are not limited to, gensim (Řehůřek and Sojka 2010), AllenNLP (Gardner et al. 2018), Scikit-learn (Pedregosa et al. 2011), Flair (Akbik, Blythe, and Vollgraf 2018) and the HuggingFace Transformers (Wolf et al. 2019) library. We are indebted to the contributions of these powerful tools to the community.

Overall, this yields to more than 25 embeddings (with over 100 pre-trained models), accessible to the end-user with a single parameter change along with many complex combinations. We hope that this will serve as an accelerator to innovative AI applications using unstructured text.

Fine Tuning Downstream Applications

Text featurization can be treated as a separate step that provides input to machine learning applications. For example, we can featurize a set of articles and then build a personalization model to map this representation to historical engagement data based on user responses. We can then use the trained model to make personalized recommendations for new users when selecting the best content to present. Notice that, in this approach, there are two disjoint learning steps; the learning of the featurization purely from unstructured data and the learning of the response behavior from the structured engagement data.

Transformations	Grad	Fine-Tuning
LDA: Latent Dirichlet Allocation	✗	✗
NMF: Non-negative Matrix Factorization	✗	✗
Pooling Word Vectors (first, last, min, max, mean)	✓	✗
SVD: Singular Value Decomposition	✓	✓
UMAP: Uniform Manifold Approximation and Projection	✗	✗

Table 2: Available Transformations in TEXTWISER with the ability to propagate gradients and fine-tuning.

Performance can be improved by combining these two learning steps in joint optimization. This is shown to work well by Transfer Learning approaches, which enables starting from pre-trained language models and then fine-tuning them on specific applications (Ruder et al. 2019). Some of the libraries TEXTWISER relies on, such as the HuggingFace Transformers (Wolf et al. 2019), provide their pre-trained models as PyTorch models that can be further trained or fine-tuned. Along the same lines, TEXTWISER retains this ability to fine-tune embeddings and even enables fine-tuning some pre-trained embeddings that cannot be fine-tuned within the original library, e.g., word embeddings in Flair (Akbik, Blythe, and Vollgraf 2018)). When resulting TEXTWISER features are used in subsequent training loops in downstream tasks, the back-propagation algorithm can update the featurization layers for these embeddings.

In principle, whether an instantiation is fine-tuneable depends on the specific Embedding and Transformation. Any pre-trained word2vec embedding and any pre-trained transformer-based embedding is fine-tunable. Likewise, any featurization that ends with an SVD transformation becomes fine-tunable as the SVD transformation itself is fine-tunable. This is true even if the embedding used before the SVD transformation is not inherently fine-tunable. For example, the TF-IDF + SVD combination becomes fine-tunable despite the static count-based vectors. Conversely, the UMAP (McInnes, Healy, and Melville 2018) transformation optimizes a particular objective function outside the underlying computation graph, which invalidates fine-tuning.

Hyper-Parameter Optimization

To find the best hyper-parameters for a given task, random search and grid search are two popular techniques together with Bayesian-based optimization techniques (Bergstra, Yamins, and Cox 2013). TEXTWISER is designed with rapid prototyping and interoperability in mind. Not only all underlying model parameters are exposed to the user, but integration with the Scikit-learn Pipelines is also supported. TEX-

TWISER instantiations fully cooperate with model selection methods such as *RandomizedSearch* and *GridSearch*.

We can even go one step further and treat all text featurization techniques as hyper-parameters. For example, in a classification scenario, we can consider multiple instantiations, each with multiple parameter distributions, chained with a set of classifier models, each with their model parameters. All of these can interoperate within a Scikit-learn pipeline to optimize a selected metric, e.g., F1 score, over the training and test set. We provide a detailed hyper-parameter tuning notebook example in the repository.

Automatically Building Embeddings From Components

In the previous two sections, we described how to tune hyper-parameters of different instantiations with and without fine-tuning. Notice that there is still a cognitive step involved: a model designer must specify which instantiations to tune for. Given our grammar specification that *implicitly* defines all possible instantiations, it is possible to alleviate the design step from humans by automatically building embeddings from components. A naïve approach would be to brute-force all possible combinations and perform language membership with a parser, e.g., the CYK parser (Cocke and Schwartz 1970), to eliminate invalid strings and generate candidate designs. Taking this a step further, we can employ a reasoning based approach, such as grammar constraints (Sellmann 2006; Kadioglu and Sellmann 2008) to generate *only* the valid embeddings within a finite length of strings. Constraint-based reasoning help incorporate other preferences too; e.g., we can restrict the cardinality of an SVD decomposition in the entire string representation to a fixed value, or even a range, and seek at least one word embedding in the compound. The intersection of NLP and Algorithm Configuration is one of our active research directions.

Reproducibility

One of the major hurdles faced in ML research and productionization is reproducibility. Specifically, in industrial settings, the ability to re-trace the exact decision-making process is not only desirable, e.g., for debugging purposes, but also in part required by law for trusted decision making.

The formalism provided by our unification grammar, which is captured by a simple JSON schema in the compound embedding, serves as the *contract* behind the specific featurization technique. Granted the same library version and the seed, the grammar representation guarantees identical instantiations given the same schema. This is especially beneficial as the platforms used by data scientists during the model development stage typically differs from the platforms used by ML engineers during model deployment and scoring. With the grammar-based approach, the deployment of a specific text featurization component is reduced down to sharing a human-readable schema between model development and deployment. Moreover, if we do not want to create the featurization from scratch, TEXTWISER models support the standard persistence protocols such as *pickle* and *torch.save* to store the resulting model rather than the schema.

EaSe: Embeddings-as-a-Service

Let us finally share a proof-of-concept application powered by TEXTWISER. This application's motivating scenario is as follows: even with a unified text featurization library, similar efforts are re-invented in machine learning projects. A considerable amount of data scientist hours and shared computing resources (especially GPUs) are spent utilizing text data, often leading to different results. According to NVIDIA's benchmark, a V100 GPU making inference on a BERT base model can process 766 sequences per second[2]. Sequences with over 100M tokens is considered typical in text datasets which amounts to 36 hours of GPU time spent generating embeddings for a single featurization task.

Moreover, there is a proliferation of pre-processing pipelines, embedding models, and pooling methods, resulting in different representations across different use cases with no quantified benefit. This complexity only increases when fine-tuning and data-specific models are introduced. Our proof-of-concept, EASE: EMBEDDINGS-AS-A-SERVICE, abstracts these steps away from data scientists by hosting pre-computed embeddings for various datasets and language models. EASE serves feature vectors built via TEXTWISER to users on-demand with a REST API and removes repetitive steps from the AI modeling pipeline and offers benefits such as:

- Reduced entry barrier to using textual data
- Ease of sharing language models across teams
- Transparency and archivability of data processing steps
- Immediate baseline performance from readily available embeddings

An additional benefit is increased security and privacy. When embeddings are already available for consumption, fewer access requests are needed for computing resources and raw data stored across multiple systems. There exist similar efforts in the public domain, such as bert-as-a-service[3], albeit specific to only one approach compared to what TEXTWISER can offer through EaSe.

Conclusion

The utilization of unstructured text data and sophisticated featurization methods are among the most significant advances in unlocking better performance in real-world applications. Given the evolving research landscape and the plethora of tools, we run the risk of building pipelines and solutions that are not only repetitive but may become obsolete in a short time. This increases the development time, maintenance cost and the room for error.

[2]NVIDIA Data Center Deep Learning Product Performance: https://developer.nvidia.com/deep-learning-performance-training-inference

[3]bert-as-service, Han Xiao, 2018: https://github.com/hanxiao/bert-as-service

In this paper, we presented TEXTWISER that can alleviate this problem by providing access to state-of-the-art text featurization methods in a unified fashion. Our novel research contribution introduces a context-free grammar to capture the representation of (complex) embeddings. This formalism also opens the door for building embeddings automatically from components as a future research direction.

We welcome the feedback from the AI community. Hopefully, TEXTWISER can serve as an accelerator in both academia and the industry for the development of innovative AI applications that enjoy the benefits of ever-increasing amounts of unstructured data.

Acknowledgements

We would like to thank Justin Rackliffe and Nurtekin Savaş for their support in open source software and Daniel Choate, Nicholas Cilfone, Pranab Mohanty and Siddharth Narayanan for their help with the EaSe service.

References

Akbik, A.; Blythe, D.; and Vollgraf, R. 2018. Contextual String Embeddings for Sequence Labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, 1638–1649.

Bergstra, J.; Yamins, D.; and Cox, D. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, 115–123.

Blei, D. M.; Ng, A. Y.; and Jordan, M. I. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3(Jan): 993–1022.

Chomsky, N. 1956. Three models for the description of language. *IRE Transactions on information theory* 2(3): 113–124.

Cocke, W. J.; and Schwartz, J. T. 1970. *Programming languages and their compilers*. Courant Institute of Mathematical Sciences.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* .

Gardner, M.; Grus, J.; Neumann, M.; Tafjord, O.; Dasigi, P.; Liu, N.; Peters, M.; Schmitz, M.; and Zettlemoyer, L. 2018. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640* .

Golub, G. H.; and Reinsch, C. 1971. Singular value decomposition and least squares solutions. In *Linear Algebra*, 134–151. Springer.

Honnibal, M.; and Montani, I. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Jones, K. S. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* .

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC - Instance-Specific Algorithm Configuration. In Coelho, H.; Studer, R.; and Wooldridge, M. J., eds., *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 751–756. IOS Press. doi:10.3233/978-1-60750-606-5-751. URL https://doi.org/10.3233/978-1-60750-606-5-751.

Kadioglu, S.; and Sellmann, M. 2008. Efficient Context-Free Grammar Constraints. In Fox, D.; and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 310–316. AAAI Press. URL http://www.aaai.org/Library/AAAI/2008/aaai08-049.php.

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* .

Le, Q.; and Mikolov, T. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*, 1188–1196.

Lee, D. D.; and Seung, H. S. 2001. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, 556–562.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* .

Loper, E.; and Bird, S. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, 63–70. USA: Association for Computational Linguistics. doi:10.3115/1118108.1118117. URL https://doi.org/10.3115/1118108.1118117.

McInnes, L.; Healy, J.; and Melville, J. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* .

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, 8026–8037.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12: 2825–2830.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365* .

Řehůřek, R.; and Sojka, P. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45–50. Valletta, Malta: ELRA. http://is.muni.cz/publication/884893/en.

Ruder, S.; Peters, M. E.; Swayamdipta, S.; and Wolf, T. 2019. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, 15–18.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* .

Sellmann, M. 2006. The theory of grammar constraints. In *International Conference on Principles and Practice of Constraint Programming*, 530–544. Springer.

Stanley, K. O.; and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10(2): 99–127.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Davison, J.; Shleifer, S.; von Platen, P.; Ma, C.; Jernite, Y.; Plu, J.; Xu, C.; Scao, T. L.; Gugger, S.; Drame, M.; Lhoest, Q.; and Rush, A. M. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* abs/1910.03771.