LM-inspector: End-to-end Inspection of Pre-trained Language Models

Felix Morger
University of Gothenburg
felix.morger@gu.se

Abstract

With the coming of of pre-trained language models, such as ELMo, BERT and XLNet, the NLP landscape has changed drastically. Due to their popularity but also computational size and complexity, many studies have focused on the inner workings and mechanisms of these models, especially BERT. While interesting findings have been made pertaining to their ability to encode certain linguistic phenomena, less focus has been made to analyze them in the context of down-stream tasks. This work presents a tool for end-to-end inspection of pre-trained language models. The tool lets a user define a configuration for querying evaluation data and subcomponents of the model in order to carry out localized interpretation. By looking at an example from word-sense disambiguation, we show how the framework and the method can be applied to make useful and localized interpretations, which facilitates transparent inspection in line with the vision of explainable AI.

1 Introduction

Pre-trained language models have in a few years become an ubiquitous part of many natural language processing systems. Models such as BERT (Devlin et al., 2019), are highly useful in that they bring semantically rich word representations and, unlike predecessors such as Word2Vec (Mikolov et al., 2013) or fastText (Bojanowski et al., 2017), are dependent on a given context of surrounding words. The ability to attain significant out-of-the-box performance boosts and to fine-tune for particular tasks, have made them not only very useful but also important in achieving new state-of-the-art results in many tasks such as sentiment classification, question answering and natural language inference (Devlin et al., 2019).

The sheer size and computational complexity of these models are both their curse and their blessing. The larger version of transformer-based XL-Net (Yang et al., 2019), for example, consists of 24 layers, 1024 hidden states, and 16 heads, and by some estimates can cost from 61 000 to 247 000 dollars to train as well as having the energy consumption of a car over its entire lifespan[1]. Besides environmental concerns, questions have been raised on what kind of linguistic information is stored in the models and how it is processed. So much so, that a subfield called BERTology (Rogers et al., 2020) has emerged. Within this field, interesting findings have been made, for example that pre-trained language models process linguistic information similar to that of a traditional NLP-pipeline (Peters et al., 2018) (Tenney et al., 2019) and that particular attention heads attend to linguistic notions of syntax and co-reference with high accuracy, such as direct objects to verbs, determiners to nouns and objects to prepositions (Clark et al., 2019).

While work in BERTology has given important insights into the encoded linguistic information and inner workings of pre-trained language models, in particular BERT, less focus has been given to interpretation of pre-trained models in the context of downstream tasks and domain-specific data. To address this, this work-in-progress presents a tool, LM-inspector, which showcases a framework for transparent end-to-end post-hoc interpretation of pre-trained language models, in particular BERT, in the context of task-specific classifiers.

2 Methodology

2.1 Framework

The goal of LM-inspector (**L**anguage **M**odel-inspector) is to enable *post-hoc* end-to-end inspection of a pre-trained language model in the context

[1]Follow this link (accessed 2020-09-08) for a more detailed discussion on the cost of training.

of a complete neural network architecture. Thus, three components are needed for inspection:

- A trained neural network model.
- A language model, such as BERT, embedded in the neural network model.
- A set of evaluation data of inputs and output labels.

LM-inspector makes it easy for different components and data points to be analyzed in isolation, by letting users define a **configuration** to query for specific subcomponents of interest. A configuration consists of a **scope**, which specifies the components of the models to be analyzed, e.g. layers and heads, a specified **filter** of inputs, for example by some word(s) or label(s), and an **input context**, which specifies the word representation(s) of the input to look at (most commonly the word representation used for the task at hand) and the window of words around that word.

Given a configuration, a user can choose an inspection method to apply. Currently, LM-inspector supports a basic feature attribution scoring method, which computes the *top-k* most attended to entities. These entities can be *words*, *word positions* or *words + positions*. Figure 1 illustrates an example of the top-k most attended to method (where $k = 5$ and the return entities are words) being applied to a configuration in the context of a neural network trained for word-sense disambiguation. It uses the DistilBert (Sanh et al., 2019) model and is trained on an annotated sense corpus where the input document contains an annotated ambiguous word and the output label is the sense of that word.

The configuration for this example consists of:

scope $layers = [0, 1, 3, 5], heads = [0, 2, 4, 6, 8]$

filter $labels = [case\#2]$, where case#2 is the specific WordNet (Fellbaum, 1998) sense of *case*, denoting *a special set of circumstances*, such as in the sentences "in that event, the first possibility is excluded" or "it may rain in which case the picnic will be canceled".

input context the word representation of the ambiguous word *case* in the input text.

For our particular data set and configuration filter, the configuration returns 10 samples, which the classifier scored with 0.69 accuracy. Apart from the special token <UNK> as well as non-lexical words and delimiters such as *the*, *of* and *,* we see words like *held* attended to in the middle layers l_1 and l_3 while *cases* and *reaches* get more prominence in the last layer l_5.

2.2 Implementation

LM-inspector is implemented in Python (Van Rossum and Drake, 2009) and uses PyTorch (Paszke et al., 2017) and the transformers library (Wolf et al., 2019). Visualization is done with the JavaScript library D3.js (Bostock et al., 2011). LM-inspector is, on the one hand, an API to make the kind of inspection described above, and, on the other hand, a visualization tool to aid in that inspection, which will be accessible with Jupyter Notebook (Kluyver et al., 2016).

3 Discussion

This work-in-progress project presents a framework for end-to-end inspection of pre-trained language models. By providing a framework for querying specific pre-trained language model components as well as evaluation data and applying an inspection method of *top-k* most attended to words, we have shown how the tool can be used to carry out localized interpretation. As such, this tool facilitates some desirable properties of machine learning interpretability, namely *transparency* and *decomposability* (Lipton, 2018), and puts it more in line with the goals of explainable AI in increasing accountability of AI systems. With that said, in practical terms, the tool can be used both on the developer side to better understand the inner workings of a classifier as well as on the end-user side as a part of a product solution.

However, general linguistic inquiry can also benefit from a framework like the one suggested in this work. One the one hand, it could be used to find evidence of linguistic phenomena relating to particular word inputs or output labels. For example a developer of a WSD-classifier could use the *top-k* most attended to entities method described in this work to see which words of a particular sense are most attended to. On the other hand, it could be used to discover previously unknown phenomena, for example a designer of a word sense annotated corpus could look at word senses with high error rates to determine if new senses need to be added.

Although this work has focused on a specific type of interpretation method, namely feature at-

Figure 1: The top 5 most attended words across all components, i.e. layers and heads, defined in **scope**.

tribution, in the end that should preferably not be the be-all end-all method to interpret pre-trained language models. In the best of all worlds, this tool should be used in conjunction with other interpretation techniques, such as probing tasks or adversarial data sets, in order to make a broad and holistic diagnosis of the inner workings of pre-trained language models.

Remaining work for the tool consists of extending the set of available interpretation methods, with a focus on attribution scores and integrating the tool into Jupyter Notebook. Although the focus is at the moment primarily on Transformer-based architectures, such as BERT, the aim is to support non-transformer based architectures as well. Another important aim is to carry out case studies of different neural network architectures and downstream tasks as well as pre-trained language models to assess the feasibility of the tool in a real world application.

Complementary visualization of inspection results will remain an important feature since it increases the *transparency* of the system. With visualization such as in Figure 1, users can more easily identify relevant patterns of the models (in this case words that are highly attended to). For this, a lot of inspiration can be drawn from related work in computer vision, where visualization as a means for explainable AI is quite evolved, such as heat mapping of salient pixels (Bach et al., 2015) or activation maximation of specific neural network components (Olah et al., 2018).

References

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

M. Bostock, V. Ogievetsky, and J. Heer. 2011. D data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics.

J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press.

Zachary C Lipton. 2018. The mythos of model interpretability. *Queue*, 16(3):31–57.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. 2018. The building blocks of interpretability. *Distill*, 3(3):e10.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Matthew Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. 2018. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Brussels, Belgium. Association for Computational Linguistics.

Anna Rogers, O. Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *ArXiv*, abs/2002.12327.

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.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. In *ACL*.

Guido Van Rossum and Fred L. Drake. 2009. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763.