- We already did a fine-tunning

- The models used depended on the language and taks availability of HuggingFace

🤗 Transformers: State-of-the-art Natural Language Processing

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond,
Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault,
Rémi Louf, Morgan Funtowicz[†], Jamie Brew

HuggingFace Inc., Brooklyn, USA
[†] NAVER LABS Europe, Grenoble, France

`{first-name}@huggingface.co`

arXiv:1910.03771v2 [cs.CL] 14 Oct 2019

Abstract

Recent advances in modern Natural Language Processing (NLP) research have been dominated by the combination of Transfer Learning methods with large-scale language models, in particular based on the Transformer architecture. With them came a paradigm shift in NLP with the starting point for training a model on a downstream task moving from a blank specific model to a general-purpose pretrained architecture. Still, creating these general-purpose models remains an expensive and time-consuming process restricting the use of these methods to a small sub-set of the wider NLP community. In this paper, we present Huggingface's `Transformers` library, a library for state-of-the-art NLP, making these developments available to the community by gathering state-of-the-art general-purpose pretrained models under a unified API together with an ecosystem of libraries, examples, tutorials and scripts targeting many downstream NLP tasks. Huggingface's `Transformers` library features carefully crafted model implementations and high-performance pretrained weights for two main deep learning frameworks, PyTorch and TensorFlow, while supporting all the necessary tools to analyze, evaluate and use these models in downstream tasks such as text/token classification, questions answering and language generation among others. The library has gained significant organic traction and adoption among both the researcher and practitioner communities. We are committed at Hugging Face to pursue the efforts to develop this toolkit with the ambition of creating the standard library for building NLP systems.

1 Introduction

In the past 18 months, advances on many Natural Language Processing (NLP) tasks have been dominated by deep learning models and, more specifically, the use of Transfer Learning methods (Ruder et al., 2019) in which a deep neural network language model is pretrained on a web-scale unlabelled text dataset with a general-purpose training objective before being fine-tuned on various downstream tasks. Following noticeable improvements using Long Short-Term Memory (LSTM) architectures (Howard and Ruder, 2018; Peters et al., 2018), a series of works combining Transfer Learning methods with large-scale Transformer architectures (Vaswani et al., 2017) has repeatedly advanced the state-of-the-art on NLP tasks ranging from text classification (Yang et al., 2019), language understanding (Liu et al., 2019; Wang et al., 2018, 2019), machine translation (Lample and Conneau, 2019), and zero-short language generation (Radford et al., 2019) up to co-reference resolution (Joshi et al., 2019) and commonsense inference (Bosselut et al., 2019).

While this approach has shown impressive improvements on benchmarks and evaluation metrics, the exponential increase in the size of the pretraining datasets as well as the model sizes (Liu et al., 2019; Shoeybi et al., 2019) has made it both difficult and costly for researchers and practitioners with limited computational resources to benefit from these models. For instance, RoBERTa (Liu et al., 2019) was trained on 160 GB of text using 1024 32GB V100. On Amazon-Web-Services cloud computing (AWS), such a pretraining would cost approximately 100K USD.

Contrary to this trend, the booming research in Machine Learning in general and Natural Language Processing in particular is arguably explained significantly by a strong focus on knowledge sharing and large-scale community efforts resulting in the development of standard libraries, an increased availability of published research code and strong incentives to share state-of-the-art pretrained models. The combination of these factors has lead researchers to reproduce previous results more easily, investigate current approaches and test hypotheses without having to redevelop them first, and focus their efforts on formulating and testing new hypotheses.

To bring Transfer Learning methods and large-scale pretrained Transformers back into the realm of these best practices, the authors (and the community of contributors) have developed `Transformers`, a library for state-of-the art Natural Language Processing with Transfer Learning models. `Transformers` addresses several key challenges:

Sharing is caring `Transformers` gathers, in a single place, state-of-the art architectures for both Natural Language Understanding (NLU) and Natural Language Generation (NLG) with model code and a diversity of pretrained weights. This allows a form of training-computation-cost-sharing so that low-resource users can reuse pretrained models without having to train them from scratch. These models are accessed through a simple and unified API that follows a classic NLP pipeline: setting up configuration, processing data with a tokenizer and encoder, and using a model either for training (adaptation in particular) or inference. The model implementations provided in the library follow the original computation graphs and are tested to ensure they match the original author implementations' performances on various benchmarks.

Easy-access and high-performance `Transformers` was designed with two main goals in mind: (i) be as easy and fast to use as possible and (ii) provide state-of-the-art models with performances as close as possible to the originally reported results. To ensure a low entry barrier, the number of user-facing abstractions to learn was strongly limited and reduced to just three standard classes: configuration, models and tokenizers, which all can be initialized in a simple and unified way by using a common 'from_pretrained()' instantiation method.

Interpretability and diversity There is a growing field of study, sometimes referred as *BERTology* from BERT (Devlin et al., 2018), concerned with investigating the inner working of large-scale pretrained models and trying to build a science on top of these empirical results. Some examples include Tenney et al. (2019), Michel et al. (2019), Clark et al. (2019b). `Transformers` aims at facilitating and increasing the scope of these studies by (i) giving easy access to the inner representations of these models, notably the hidden states, the attention weights or heads importance as defined in Michel et al. (2019) and (ii) providing different models in a unified API to prevent overfitting to a specific architecture (and set of pretrained weights). Moreover, the unified front-end of the library makes it easy to compare the performances of several architectures on a common language understanding benchmark. `Transformers` notably includes pre-processors and fine-tuning scripts for *GLUE* (Wang et al., 2018), *SuperGLUE* (Wang et al. (2019)) and *SQuAD1.1* (Rajpurkar et al., 2016).

Pushing best practices forward `Transformers` seeks a balance between sticking to the original authors' code-base for reliability and providing clear and readable implementations featuring best practices in training deep neural networks so that researchers can seamlessly use the code-base to explore new hypothesis derived from these models. To accommodate a large community of practitioners and researchers, the library is deeply compatible with (and actually makes compatible) two major deep learning frameworks: PyTorch (Paszke et al., 2017) and TensorFlow (from release 2.0) (Abadi et al., 2015).

From research to production Another essential question is how to make these advances in research available to a wider audience, especially in the industry. `Transformers` also takes steps towards a smoother transition from research to production. The provided models support *TorchScript*, a way to create serializable and optimizable models from PyTorch code, and features production code and integration with the *TensorFlow Extended* framework.

2 Community

The development of the `Transformers` originally steamed from open-sourcing internals tools used at HuggingFace but has seen a huge growth in scope over its ten months of existence as reflected by the successive changes of name of the library: from `pytorch-pretrained-bert` to `pytorch-transformers` to, finally, `Transformers`.

A fast-growing and active community of researchers and practitioners has gathered around `Transformers`. The library has quickly become used both in research and in the industry: at the moment, more than 200 research papers report using the library[1]. `Transformers` is also included either as a dependency or with a wrapper in several popular NLP frameworks such as `Spacy` (Honnibal and Montani, 2017), `AllenNLP` (Gardner et al., 2018) or `Flair` (Akbik et al., 2018).

`Transformers` is an ongoing effort maintained by the team of engineers and research scientists at Hugging Face[2], with support from a vibrant community of more than 120 external contributors. We are committed to the twin efforts of developing the library and fostering positive interaction among its community members, with the ambition of creating the standard library for modern deep learning NLP.

`Transformers` is released under the Apache 2.0 license and is available through *pip* or from source on GitHub[3]. Detailed documentation along with on-boarding tutorials are available on Hugging Face's website[4].

3 Library design

`Transformers` has been designed around a unified frontend for all the models: parameters and configurations, tokenization, and model inference. These steps reflect the recurring questions that arise when building an NLP pipeline: defining the model architecture, processing the text data and finally, training the model and performing inference in production. In the following section, we'll give an overview of the three base components of the library: configuration, model and tokenization classes. All of the components are compatible with PyTorch and TensorFlow (starting 2.0). For complete details, we refer the reader to the documentation available on `https://huggingface.co/transformers/`.

3.1 Core components

All the models follow the same philosophy of abstraction enabling a unified API in the library.

Configuration - A configuration class instance (usually inheriting from a base class 'PretrainedConfig') stores the model and tokenizer parameters (such as the vocabulary size, the hidden dimensions, dropout rate, etc.). This configuration object can be saved and loaded for reproducibility or simply modified for architecture search.

[1] `http://search.arxiv.org:8081/?query=huggingface&qid=1565055415921multi_nCnN_-1835167213&byDate=1`

[2] `https://huggingface.co`

[3] `https://github.com/huggingface/transformers`

[4] `https://huggingface.co/transformers/`

The configuration defines the architecture of the model but also architecture optimizations like the heads to prune. Configurations are agnostic to the deep learning framework used.

Tokenizers - A Tokenizer class (inheriting from a base class ‘PreTrainedTokenizer‘) is available for each model. This class stores the vocabulary token-to-index map for the corresponding model and handles the encoding and decoding of input sequences according to the model’s tokenization-specific process (ex. Byte-Pair-Encoding, SentencePiece, etc.). Tokenizers are easily modifiable to add user-selected tokens, special tokens (like classification or separation tokens) or resize the vocabulary.

Furthermore, Tokenizers implement additional useful features for the users, by offering values to be used with a model; these range from token type indices in the case of sequence classification to maximum length sequence truncating taking into account the added model-specific special tokens (most pretrained Transformers models have a maximum sequence length they can handle, defined during their pretraining step).

Tokenizers can be instantiated from existing configurations available through `Transformers` originating from the pretrained models or created more generally by the user from user-specifications.

Model - All models follow the same hierarchy of abstraction: a base class implements the model’s computation graph from encoding (projection on the embedding matrix) through the series of self-attention layers and up to the last layer hidden states. The base class is specific to each model and closely follows the original implementation, allowing users to dissect the inner workings of each individual architecture.

Additional wrapper classes are built on top of the base class, adding a specific head on top of the base model hidden states. Examples of these heads are language modeling or sequence classification heads. These classes follow similar naming pattern: `XXXForSequenceClassification` or `XXXForMaskedLM` where `XXX` is the name of the model and can be used for adaptation (fine-tuning) or pre-training.

All models are available both in PyTorch and TensorFlow (starting 2.0) and offer deep interoperability between both frameworks. For instance, a model trained in one of frameworks can be saved on drive for the standard library serialization practice and then be reloaded from the saved files in the other framework seamlessly, making it particularly easy to switch from one framework to the other one along the model life-time (training, serving, etc.).

Auto classes - In many cases, the architecture to use can be automatically guessed from the shortcut name of the pretrained weights (e.g. ‘bert-base-cased‘). A set of `Auto` classes provides a unified API that enable very fast switching between different models/configs/tokenizers. There are a total of four high-level abstractions referenced as `Auto` classes: `AutoConfig`, `AutoTokenizer`, `AutoModel` (for PyTorch) and `TFAutoModel` (for TensorFlow). These classes automatically instantiate the right configuration, tokenizer or model class instance from the name of the pretrained checkpoints.

3.2 Training

Optimizer - The library provides a few optimization utilities as subclasses of PyTorch ‘torch.optim.Optimizer‘ which can be used when training the models. The additional optimizer currently available is the Adam optimizer (Kingma and Ba, 2014) with an additional weight decay fix, also known as ‘AdamW‘ (Loshchilov and Hutter, 2017).

Scheduler - Additional learning rate schedulers are also provided as subclasses of PyTorch ‘torch.optim.lr_scheduler.LambdaLR‘, offering various schedules used for transfer learning and transformers models with customizable options including warmup schedules which are relevant when training with Adam.

4 Experimenting with `Transformers`

In this section, we present some of the major tools and examples provided in the library to experiment on a range of downstream Natural Language Understanding and Natural Language Generation tasks.

4.1 Language understanding benchmarks

The language models provided in Transformers are pretrained with a general purpose training objective, usually a variant of language modeling like *standard (sometime called causal) language modeling* as used for instance in Radford et al. (2019) or *masked language modeling* as introduced in Devlin et al. (2018). A pretrained model is often evaluated using wide-range language understanding benchmarks. Transformers includes several tools and scripts to evaluate models on *GLUE* (Wang et al. (2018)) and *SuperGLUE* (Wang et al. (2019)). These two benchmarks gather a variety of datasets to evaluate natural language understanding systems. Details of the datasets can be found in the Appendix on page 7.

Regarding the machine comprehension tasks, the library feature evaluations on *SQuAD1.1* (Rajpurkar et al. (2016)) and *SQuAD2.0* (Rajpurkar et al. (2018)).

Others currently-supported benchmarks include *SWAG* (Zellers et al. (2018)), *RACE* (Lai et al. (2017)) and *ARC* (Clark et al. (2018)).

4.2 Language model fine-tuning

Fine-tuning a language model on a downstream text corpus usually leads to significant gains for tasks on this corpus, in particular when the domain is different (domain adaptation). It also significantly reduces the amount of training data required for fine-tuning on a target task in the target domain. Transformers provides simple scripts to fine-tune models on custom text datasets with the option to add or remove tokens from the vocabulary and several other adaptability features.

4.3 Ecosystem

Write with Transformer Because Natural Language Processing does not have to be serious and boring, the generative capacities of auto-regressive language models available in Transformers are showcased in an intuitive and playful manner. Built by the authors on top of Transformers, *Write with Transformer*[5] is an interactive interface that leverages the generative capabilities of pretrained architectures like GPT, GPT2 and XLNet to suggest text like an auto-completion plugin. Generating samples is also often used to qualitatively (and subjectively) evaluate the generation quality of language models (Radford et al., 2019). Given the impact of the decoding algorithm (top-K sampling, beam-search, etc.) on generation quality (Holtzman et al., 2019), *Write with Transformer* offers various options to dynamically tweak the decoding algorithm and investigate the resulting samples from the model.

Conversational AI HuggingFace has been using Transfer Learning with Transformer-based models for end-to-end Natural language understanding and text generation in its conversational agent, *Talking Dog*. The company also demonstrated in fall 2018 that this approach can be used to reach state-of-the-art performances on academic benchmarks, topping by a significant margin the automatic metrics leaderboard of the *Conversational Intelligence Challenge 2* held at the Thirty-second Annual Conference on Neural Information Processing Systems (NIPS 2018). The approach used to reach these performances is described in Wolf et al. (2019); Golovanov et al. (2019) and the code and pretrained models, based on the Transformers library, are available online[6].

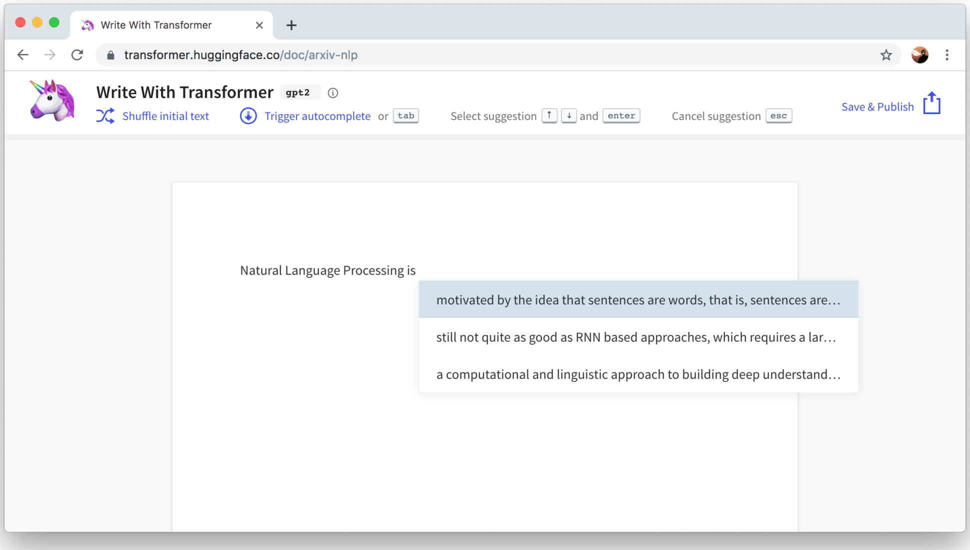
Using in production To facilitate the transition from research to production, all the models in the library are compatible with *TorchScript*, an intermediate representation of a PyTorch model that can then be run either in Python in a more efficient way, or in a high-performance environment such as C++[7]. Fine-tuned models can thus be exported to production-friendly environment.

Optimizing large machine learning models for production is an ongoing effort in the community and there are many current engineering efforts towards that goal. The distillation of large models (e.g. *DistilBERT* (Sanh et al., 2019)) is one of the most promising directions. It

[5] https://transformer.huggingface.co

[6] https://github.com/huggingface/transfer-learning-conv-ai

[7] https://pytorch.org/tutorials/beginner/Intro_to_TorchScript_tutorial.html


Write With Transformer
transformer.huggingface.co/doc/arxiv-nlp
Write With Transformer gpt2
Shuffle initial text
Trigger autocomplete or tab
Select suggestion ↑ ↓ and enter
Cancel suggestion esc
Save & Publish

Natural Language Processing is

motivated by the idea that sentences are words, that is, sentences are…

still not quite as good as RNN based approaches, which requires a lar…

a computational and linguistic approach to building deep understand…

Figure 1: Write With Transformer

lets users of `Transformers` run more efficient versions of the models, even with strong latency constraints and on inexpensive CPU servers. We also convert Transformers models to `Core ML` weights that are suitable to be embbeded inside a mobile application, to enable on-the-edge machine learning. Code is also made available[8].

Community Many libraries in NLP and Machine Learning have been created on top of `Transformers` or have integrated `Transformers` as a package dependency or through wrappers. At the time of writing, the authors have been mostly aware of `FastBert`[9], `FARM`[10], `flair` (Akbik et al., 2018, 2019), `AllenNLP` (Gardner et al., 2018) and `PyText`[11] but there are likely more interesting developments to be found, from research and internal projects to production packages.

5 Architectures

Here is a list of architectures for which reference implementations and pretrained weights are currently provided in `Transformers`. These models fall into two main categories: generative models (GPT, GPT-2, Transformer-XL, XLNet, XLM) and models for language understanding (Bert, DistilBert, RoBERTa, XLM).

- BERT (Devlin et al. (2018)) is a bi-directional Transformer-based encoder pretrained with a linear combination of *masked language modeling* and *next sentence prediction* objectives.
- RoBERTa (Liu et al. (2019)) is a replication study of BERT which showed that carefully tuning hyper-parameters and training data size lead to significantly improved results on language understanding.
- DistilBERT (Sanh et al. (2019)) is a smaller, faster, cheaper and lighter version BERT pretrained with knowledge distillation.
- GPT (Radford et al. (2018)) and GPT2 (Radford et al. (2019)) are two large auto-regressive language models pretrained with *language modeling*. GPT2 showcased

[8] `https://github.com/huggingface/swift-coreml-transformers`

[9] `https://github.com/kaushaltrivedi/fast-bert`

[10] `https://github.com/deepset-ai/FARM`

[11] `https://github.com/facebookresearch/pytext`

zero-shot task transfer capabilities on various tasks such as machine translation or reading comprehension.

- Transformer-XL (Dai et al. (2019)) introduces architectural modifications enabling Transformers to learn dependency beyond a fixed length without disrupting temporal coherence via segment-level recurrence and relative positional encoding schemes.
- XLNet (Yang et al. (2019)) builds upon Transformer-XL and proposes an auto-regressive pretraining scheme combining BERT's bi-directional context flow with auto-regressive language modeling by maximizing the expected likelihood over permutations of the word sequence.
- XLM (Lample and Conneau (2019)) shows the effectiveness of pretrained representations for cross-lingual language modeling (both on monolingual data and parallel data) and cross-lingual language understanding.

We systematically release the model with the corresponding pretraining heads (language modeling, *next sentence prediction* for BERT) for adaptation using the pretraining objectives. Some models fine-tuned on downstream tasks such as *SQuAD1.1* are also available. Overall, more than 30 pretrained weights are provided through the library including more than 10 models pretrained in languages other than English. Some of these non-English pretrained models are multi-lingual models (with two of them being trained on more than 100 languages) [12].

6 Related work

The design of `Transformers` was inspired by earlier libraries on transformers and Natural Language Processing. More precisely, organizing the modules around three main components (configuration, tokenizers and models) was inspired by the design of the `tensor2tensor` library (Vaswani et al., 2018) and the original code repository of Bert (Devlin et al., 2018) from Google Research while concept of providing easy caching for pretrained models steamed from features of the `AllenNLP` library (Gardner et al., 2018) open-sourced by the Allen Institute for Artificial Intelligence (AI2).

Works related to the `Transformers` library can be generally organized along three directions, at the intersection of which stands the present library. The first direction includes Natural Language Processing libraries such as `AllenNLP`[13] (Gardner et al., 2018), `SpaCy`[14] (Honnibal and Montani, 2017), `flair`[15] (Akbik et al., 2018, 2019) or `PyText`[16]. These libraries precede `Transformers` and target somewhat different use-cases, for instance those with a particular focus on research for `AllenNLP` or a strong attention to production constrains (in particular with a carefully tuned balance between speed and performance) for `SpaCy`. The previously mentioned libraries have now been provided with integrations for `Transformers`, through a direct package dependency for `AllenNLP`, `flair` or `PyText` or through a wrapper called `spacy-transformers`[17] for `SpaCy`.

The second direction concerns lower-level deep-learning frameworks like PyTorch (Paszke et al., 2017) and TensorFlow (Abadi et al., 2015) which have both been extended with model sharing capabilities or hubs, respectively called `TensorFlow Hub`[18] and `PyTorch Hub`[19]. These hubs are more general and while they offer ways to share models they differ from the present library in several ways. In particular, they provide neither a unified API across models nor standardized ways to access the internals of the models. Targeting a more general machine-learning community, these Hubs lack the NLP-specific user-facing features provided by `Transformers` like tokenizers, dedicated processing scripts for common

[12] `https://huggingface.co/transformers/multilingual.html`

[13] `https://allennlp.org/`

[14] `https://spacy.io//`

[15] `https://github.com/zalandoresearch/flair`

[16] `https://github.com/facebookresearch/pytext`

[17] `https://github.com/explosion/spacy-transformers`

[18] `https://github.com/tensorflow/hub`

[19] `https://pytorch.org/hub`

downstream tasks and sensible default hyper-parameters for high performance on a range of language understanding and generation tasks.

The last direction is related to machine learning research frameworks that are specifically used to test, develop and train architectures like Transformers. Typical examples are the `tensor2tensor`[20] library (Vaswani et al., 2018), `fairseq`[21] (Ott et al., 2019) and `Megatron-LM`[22]. These libraries are usually not provided with the user-facing features that allow easy download, caching, fine-tuning of the models as well as seamless transition to production.

7 Conclusion

We have presented the design and the main components of `Transformers`, a library for state-of-the-art NLP. Its capabilities, performances and unified API make it easy for both practitioners and researchers to access various large-scale language models, build and experiment on top of them and use them in downstream task with state-of-the-art performance. The library has gained significant organic traction since its original release and has become widely adopted among researchers and practitioners, fostering an active community of contributors and an ecosystem of libraries building on top of the provided tools. We are committed to supporting this community and making recent developments in transfer learning for NLP both accessible and easy to use while maintaining high standards of software engineering and machine learning engineering.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Agirre, E., M‘arquez, L., and Wicentowski, R., editors (2007). *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*. Association for Computational Linguistics, Prague, Czech Republic.

Akbik, A., Bergmann, T., and Vollgraf, R. (2019). Pooled contextualized embeddings for named entity recognition. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, page 724–728.

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649.

Bar Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., and Szpektor, I. (2006). The second PASCAL recognising textual entailment challenge.

Bentivogli, L., Dagan, I., Dang, H. T., Giampiccolo, D., and Magnini, B. (2009). The fifth PASCAL recognizing textual entailment challenge.

Bosselut, A., Rashkin, H., Sap, M., Malaviya, C., Çelikyilmaz, A., and Choi, Y. (2019). Comet: Commonsense transformers for automatic knowledge graph construction. In *ACL*.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. (2019a). BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of NAACL-HLT 2019*.

[20] `https://github.com/tensorflow/tensor2tensor`

[21] `https://github.com/pytorch/fairseq`

[22] `https://github.com/NVIDIA/Megatron-LM`

Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. (2019b). What does bert look at? an analysis of bert's attention. In *BlackBoxNLP@ACL*.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. (2018). Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457.

Dagan, I., Glickman, O., and Magnini, B. (2006). The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pages 177–190. Springer.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL*.

De Marneffe, M.-C., Simons, M., and Tonhauser, J. (2019). The CommitmentBank: Investigating projection in naturally occurring discourse. To appear in proceedings of Sinn und Bedeutung 23. Data can be found at https://github.com/mcdm/CommitmentBank/.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Dolan, W. B. and Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*.

Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M., and Zettlemoyer, L. S. (2018). Allennlp: A deep semantic natural language processing platform. In *ACL - NLP OSS workshop*.

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, B. (2007). The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics.

Golovanov, S., Kurbanov, R., Nikolenko, S., Truskovskyi, K., Tselousov, A., and Wolf, T. (2019). Large-scale transfer learning for natural language generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6053–6058.

Holtzman, A., Buys, J., Forbes, M., and Choi, Y. (2019). The curious case of neural text degeneration. *ArXiv*, abs/1904.09751.

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

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. In *ACL*.

Iyer, S., Dandekar, N., and Csernai, K. (2017). First quora dataset release: Question pairs.

Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., and Levy, O. (2019). Spanbert: Improving pre-training by representing and predicting spans. *arXiv preprint arXiv:1907.10529*.

Khashabi, D., Chaturvedi, S., Roth, M., Upadhyay, S., and Roth, D. (2018). Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *ICLR*.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. H. (2017). Race: Large-scale reading comprehension dataset from examinations. In *EMNLP*.

Lample, G. and Conneau, A. (2019). Cross-lingual language model pretraining. In *NeurIPS*.

Levesque, H. J., Davis, E., and Morgenstern, L. (2011). The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, page 47.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M. S., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. S., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Loshchilov, I. and Hutter, F. (2017). Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*.

Michel, P., Levy, O., and Neubig, G. (2019). Are sixteen heads really better than one? In *NeurIPS*.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. (2019). fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

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*.

Pilehvar, M. T. and Camacho-Collados, J. (2019). WiC: The word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of NAACL-HLT*.

Poliak, A., Haldar, A., Rudinger, R., Hu, J. E., Pavlick, E., White, A. S., and Van Durme, B. (2018). Collecting diverse natural language inference problems for sentence representation evaluation. In *Proceedings of EMNLP*.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for squad. In *ACL*.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*.

Roemmele, M., Bejan, C. A., and Gordon, A. S. (2011). Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

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*, pages 15–18.

Rudinger, R., Naradowsky, J., Leonard, B., and Van Durme, B. (2018). Gender bias in coreference resolution. In *Proceedings of NAACL-HLT*.

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*.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pages 1631–1642.

Tenney, I., Das, D., and Pavlick, E. (2019). Bert rediscovers the classical nlp pipeline. In *ACL*.

Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A. N., Gouws, S., Jones, L., Kaiser, L., Kalchbrenner, N., Parmar, N., Sepassi, R., Shazeer, N., and Uszkoreit, J. (2018). Tensor2tensor for neural machine translation. *CoRR*, abs/1803.07416.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NIPS*.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). Superglue: A stickier benchmark for general-purpose language understanding systems. *ArXiv*, abs/1905.00537.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Warstadt, A., Singh, A., and Bowman, S. R. (2018). Neural network acceptability judgments. *arXiv preprint 1805.12471*.

Williams, A., Nangia, N., and Bowman, S. R. (2018). A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*.

Wolf, T., Sanh, V., Chaumond, J., and Delangue, C. (2019). Transfertransfo: A transfer learning approach for neural network based conversational agents. In *NeurIPS ConvAI Wokshop*.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *ArXiv*, abs/1906.08237.

Zellers, R., Bisk, Y., Schwartz, R., and Choi, Y. (2018). Swag: A large-scale adversarial dataset for grounded commonsense inference. In *EMNLP*.

Zhang, S., Liu, X., Liu, J., Gao, J., Duh, K., and Durme, B. V. (2018). ReCoRD: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint 1810.12885*.

A GLUE and SuperGLUE

The datasets in GLUE are: CoLA (Warstadt et al. (2018)), Stanford Sentiment Treebank (SST) (Socher et al. (2013)), Microsoft Research Paragraph Corpus (MRPC) Dolan and Brockett (2005), Semantic Textual Similarity Benchmark (STS) Agirre et al. (2007), Quora Question Pairs (QQP) Iyer et al. (2017), Multi-Genre NLI (MNLI) Williams et al. (2018), Question NLI (QNLI) Rajpurkar et al. (2016), Recognizing Textual Entailment (RTE) Dagan et al. (2006); Bar Haim et al. (2006); Giampiccolo et al. (2007); Bentivogli et al. (2009) and Winograd NLI (WNLI) Levesque et al. (2011).

The datasets in SuperGLUE are: Boolean Questions (BoolQ) Clark et al. (2019a), CommitmentBank (CB) De Marneffe et al. (2019), Choice of Plausible Alternatives (COPA) Roemmele et al. (2011), Multi-Sentence Reading Comprehension (MultiRC) Khashabi et al. (2018), Reading Comprehension with Commonsense Reasoning Dataset (ReCoRD) Zhang et al. (2018), Word-in-Context (WiC) Pilehvar and Camacho-Collados (2019), Winograd Schema Challenge (WSC) Rudinger et al. (2018), Diverse Natural Language Inference Collection (DNC) Poliak et al. (2018), Recognizing Textual Entailment (RTE) Dagan et al. (2006); Bar Haim et al. (2006); Giampiccolo et al. (2007); Bentivogli et al. (2009) and Winograd NLI (WNLI) Levesque et al. (2011)