

1 Pruning

The basic idea behind model pruning is simple. Deep learning models which have been trained to convergence typically have a large number of weights very close to zero contributing very little weight to model inference. The smaller the weight, the more likely it is that it can be taken to zero without significantly affecting model performance. However, there are many decisions that need to be made for this to work in practice.

- **Structure:** Unstructured pruning approaches remove weights on a case-by-case basis. Structured pruning approaches remove weights in groups—e.g. removing entire channels at a time. Structured pruning typically has better runtime performance characteristics (it's a dense computation on fewer channels) but also has a heavier impact on model accuracy (it's less selective).
- **Scoring:** A measure must be chosen which decides based on a number that which weights are important and which weights are not.
- **When to prune:** (iterative pruning) which is applied in between training epochs. (one-shot pruning) which is applied all-at-once after model training is complete.

1.1 Practical issues about difficulties of pruning with PyTorch

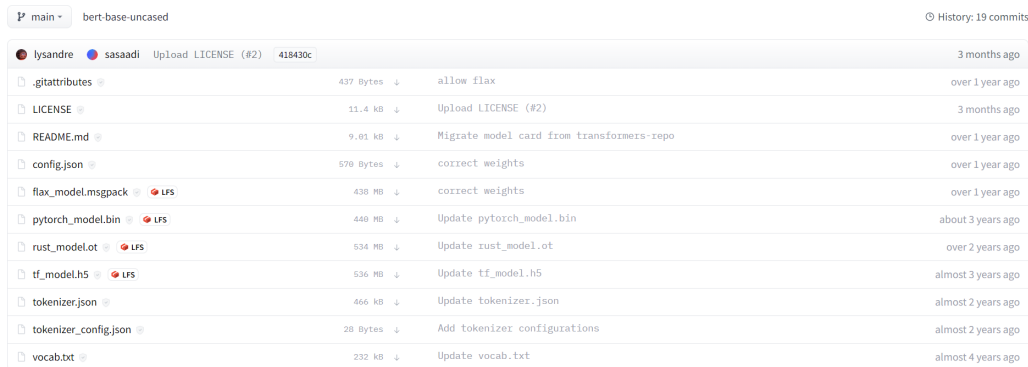
Unfortunately, model pruning in PyTorch does not currently improve model inference times. This unfortunate fact stems from the fact that model pruning does not improve inference performance or reduce model size if it is used with dense tensors. A dense tensor filled with zeroes is not any faster to compute, nor is it any smaller when written to disk. In order for that to happen, it needs to be converted to a sparse tensor. PyTorch does not have robust support for sparse tensors and the `torch.nn.sparse` library is currently in development phase.

1.2 Additional resources about pruning NNs

- <https://leimao.github.io/article/Neural-Networks-Pruning/>
- <https://machinelearningmastery.com/sparse-matrices-for-machine-learning/#:~:text=A%20sparse%20matrix%20is%20a,of%20its%20coefficients%20are%20zero>

2 TextPruner

TextPruner is an open-source model pruning toolkit designed for pre-trained language models, targeting fast and easy model compression. There are structured post-training pruning methods, including vocabulary pruning and transformer pruning, discussed in details below. **It can be only used to prune transformer based architectures; both encoder and encoder-decoder based architecture. To prune any transformer based architectures locally the directory containing the model must be arranged in such a way so as to replicate the Files and versions section of a model in huggingface model hub, as shown below in 1**



bert-base-uncased		History: 19 commits
lysandre	sasaadi	Upload LICENSE (#2) 418430c 3 months ago
.gitattributes		allow flax 437 Bytes over 1 year ago
LICENSE		Upload LICENSE (#2) 11.4 kB 3 months ago
README.md		Migrate model card from transformers-repo 9.01 kB over 1 year ago
config.json		correct weights 570 Bytes over 1 year ago
flax_model.msgpack	LFS	correct weights 438 MB over 1 year ago
pytorch_model.bin	LFS	Update pytorch_model.bin 440 MB about 3 years ago
rust_model.ot	LFS	Update rust_model.ot 534 MB over 2 years ago
tf_model.h5	LFS	Update tf_model.h5 536 MB almost 3 years ago
tokenizer.json		Update tokenizer.json 466 kB almost 2 years ago
tokenizer_config.json		Add tokenizer configurations 28 Bytes almost 2 years ago
vocab.txt		Update vocab.txt 232 kB almost 4 years ago

Figure 1: how the files for bert-base-uncased are arranged in huggingface model hub

As seen from the figure above the trained models, configuration file of the models and the required files to load the tokenizer used to train the model are placed in the same directory. So if we have a custom BERT model with a dense layer on top for multiclass classification like the one being trained in this repository, the trained files for the model and its tokenizer are placed in the directory as shown below in 2.



saved_model
config.json
pytorch_model.bin
special_tokens_map.json
tokenizer_config.json
vocab.txt

Figure 2: how the files for a custom BERT based classifier must be arranged

The train.py script in the repo does the training in that manner so that the models are saved as like the figure shown above. **The model class must also be written in a way so as to replicate the functionalities of model classes in huggingface, if not TextPruner will not be able to prune.** The BertClassifier class inside the bert_classifier.py script has been implemented keeping those things in mind.

2.1 How TextPruner prunes transformers

Each transformer contains a fully connected feed-forward network (FFN) following Multi Head Attention(MHA). The size of a transformer can be reduced by removing the attention heads or removing the intermediate neurons in the FFN layer. Attention heads and FFN neurons according to their proxy importance scores and then removed iteratively. A commonly used importance score is the sensitivity of the loss with respect to the values of the neurons. A lower importance score means the loss is less sensitive to the neurons. Therefore, the neurons are pruned in the order of increasing scores.

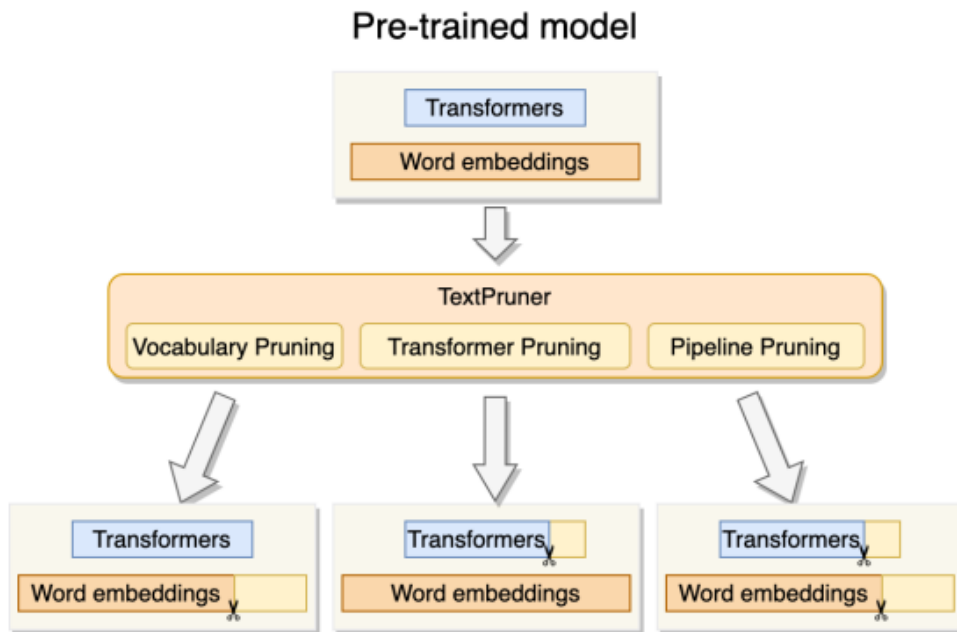


Figure 3: Three pruning modes of Transformers

As shown above in [3](#), TextPruner has three pruning modes.

- **Transformer Pruning:** Not all attention heads are equally important in the transformers, and some of the attention heads can be pruned without performance loss. TextPruner reads the examples and computes the importance scores of attention heads and the feed-forward networks' neurons. TextPruner reads the examples and computes the importance scores of attention heads and the feed-forward networks' neurons.
- **Vocabulary Pruning:** The pre-trained models have a large vocabulary, but some tokens in the vocabulary rarely appear in the downstream tasks. These tokens can be removed to reduce the model size and accelerate the training speed of the tasks. In this mode, TextPruner reads and tokenizes an input corpus. TextPruner goes through the vocabulary and checks if the token in the vocabulary has appeared in the text file. If not, the token will be removed from both the model's embedding matrix and the tokenizer's vocabulary.
- **Pipeline Pruning:** Combines both vocabulary and transformer pruning.

2.2 Results on performing transformer pruning on a trained BERT classifier in the repo

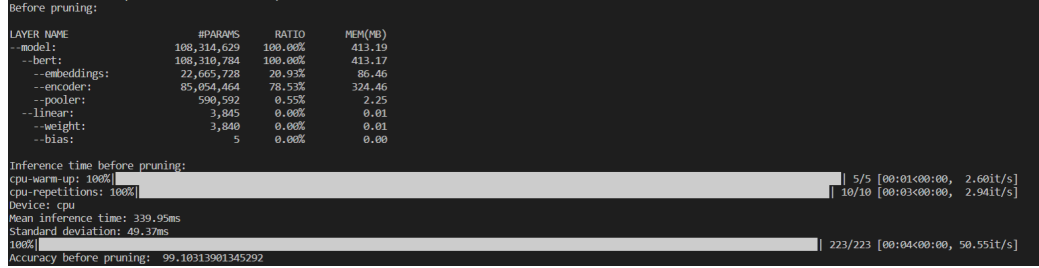


Figure 4: Before pruning the model size, inference time and accuracy

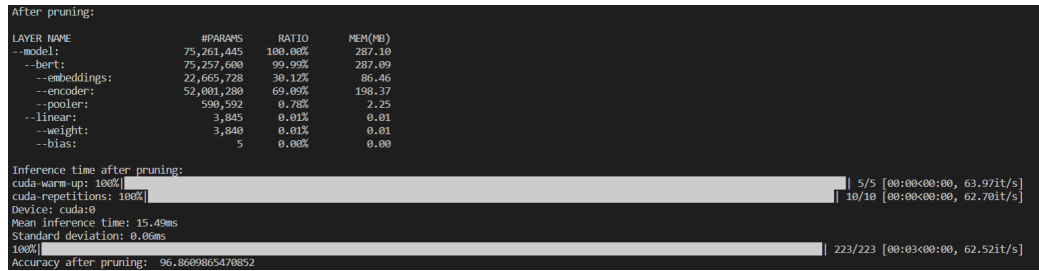


Figure 5: After pruning the model size, inference time and accuracy

As seen from 4 and 5 although accuracy is reduced a little, the model size gets smaller and inference time reduces greatly. The hyperparameter for transformer pruning can be tweaked further to get a better balance of accuracy, model size and inference time.