Movement Pruning: Adaptive Sparsity by Fine-Tuning

Author: @VictorSanh

Magnitude pruning is a widely used strategy for reducing model size in pure supervised learning; however, it is less effective in the transfer learning regime that has become standard for state-of-the-art natural language processing applications. We propose the use of movement pruning, a simple, deterministic first-order weight pruning method that is more adaptive to pretrained model fine-tuning. Experiments show that when pruning large pretrained language models, movement pruning shows significant improvements in high-sparsity regimes. When combined with distillation, the approach achieves minimal accuracy loss with down to only 3% of the model parameters:

Fine-pruning+Distillation (Teacher=BERT-base fine-tuned)	BERT base fine-tuned	Remaining Weights (%)	Magnitude Pruning	L0 Regularization	Movement Pruning	Soft Movem Prunii
SQuAD - Dev EM/F1	80.4/88.1	10% 3%	70.2/80.1 45.5/59.6	72.4/81.9 64.3/75.8	75.6/84.3 67.5/78.0	**76.6/8** **72.7/8**
MNLI - Dev acc/MM acc	84.5/84.9	10% 3%	78.3/79.3 69.4/70.6	78.7/79.7 76.0/76.2	80.1/80.4 76.5/77.4	**81.2/8** **79.5/8**
QQP - Dev acc/F1	91.4/88.4	10% 3%	79.8/65.0 72.4/57.8	88.1/82.8 87.0/81.9	89.7/86.2 86.1/81.5	**90.2/8** **89.1/8**

This page contains information on how to fine-prune pre-trained models such as `BERT` to obtain extremely sparse models with movement pruning. In contrast to magnitude pruning which selects weights that are far from 0, movement pruning retains weights that are moving away from 0.

For more information, we invite you to check out our paper. You can also have a look at this fun *Explain Like I'm Five* introductory slide deck.

Extreme sparsity and efficient storage

One promise of extreme pruning is to obtain extremely small models that can be easily sent (and stored) on edge devices. By setting weights to 0., we reduce the amount of information we need to store, and thus decreasing the memory size. We are able to obtain extremely sparse fine-pruned models with movement pruning: ~95% of the dense performance with ~5% of total remaining weights in the BERT encoder.

In this notebook, we showcase how we can leverage standard tools that exist out-of-the-box to efficiently store an extremely sparse question answering model (only 6% of total remaining weights in the encoder). We are able to reduce the memory size of the encoder **from the 340MB (the original dense BERT) to 11MB**, without any additional training of the model (every operation is performed *post fine-pruning*). It is sufficiently small to store it on a 91' floppy disk 📎!

While movement pruning does not directly optimize for memory footprint (but rather the number of non-null weights), we hypothetize that further memory compression ratios can be achieved with specific quantization aware trainings (see for instance Q8BERT, And the Bit Goes Down or Quant-Noise).

Fine-pruned models

As examples, we release two English PruneBERT checkpoints (models fine-pruned from a pre-trained `BERT` checkpoint), one on SQuAD and the other on MNLI.

- **`prunebert-base-uncased-6-finepruned-w-distil-squad`**
  Pre-trained `BERT-base-uncased` fine-pruned with soft movement pruning on SQuAD v1.1. We use an additional distillation signal from `BERT-base-uncased` finetuned on SQuAD. The encoder counts 6% of total non-null weights and reaches 83.8 F1 score. The model can be accessed with: `pruned_bert = BertForQuestionAnswering.from_pretrained("huggingface/prunebert-base-uncased-6-finepruned-w-distil-squad")`
- **`prunebert-base-uncased-6-finepruned-w-distil-mnli`**
  Pre-trained `BERT-base-uncased` fine-pruned with soft movement pruning on MNLI. We use an additional distillation signal from `BERT-base-uncased` finetuned on MNLI. The encoder counts 6% of total non-null weights and reaches 80.7 (matched) accuracy. The model can be accessed with: `pruned_bert = BertForSequenceClassification.from_pretrained("huggingface/prunebert-base-uncased-6-finepruned-w-distil-mnli")`

How to fine-prune?

Setup

The code relies on the 🤗 Transformers library. In addition to the dependencies listed in the `examples` folder, you should install a few additional dependencies listed in the `requirements.txt` file: `pip install -r requirements.txt`.

Note that we built our experiments on top of a stabilized version of the library (commit https://github.com/huggingface/transformers/commit/352d5472b0c1dec0f420d606d16747d851b4bda8): we do not guarantee that everything is still compatible with the latest version of the main branch.

Fine-pruning with movement pruning

Below, we detail how to reproduce the results reported in the paper. We use SQuAD as a running example. Commands (and scripts) can be easily adapted for other tasks.

The following command fine-prunes a pre-trained `BERT-base` on SQuAD using movement pruning towards 15% of remaining weights (85% sparsity). Note that we freeze all the embeddings modules (from their pre-trained value) and only prune the Fully Connected layers in the encoder (12 layers of Transformer Block).

```
SERIALIZATION_DIR=<OUTPUT_DIR>
SQUAD_DATA=<SQUAD_DATA>

python examples/movement-pruning/masked_run_squad.py \
    --output_dir $SERIALIZATION_DIR \
    --data_dir $SQUAD_DATA \
    --train_file train-v1.1.json \
    --predict_file dev-v1.1.json \
    --do_train --do_eval --do_lower_case \
    --model_type masked_bert \
    --model_name_or_path bert-base-uncased \
    --per_gpu_train_batch_size 16 \
    --warmup_steps 5400 \
    --num_train_epochs 10 \
    --learning_rate 3e-5 --mask_scores_learning_rate 1e-2 \
    --initial_threshold 1 --final_threshold 0.15 \
    --initial_warmup 1 --final_warmup 2 \
    --pruning_method topK --mask_init constant --mask_scale 0.
```

Fine-pruning with other methods

We can also explore other fine-pruning methods by changing the `pruning_method` parameter:

Soft movement pruning

```
python examples/movement-pruning/masked_run_squad.py \
    --output_dir $SERIALIZATION_DIR \
    --data_dir $SQUAD_DATA \
    --train_file train-v1.1.json \
    --predict_file dev-v1.1.json \
    --do_train --do_eval --do_lower_case \
    --model_type masked_bert \
    --model_name_or_path bert-base-uncased \
    --per_gpu_train_batch_size 16 \
    --warmup_steps 5400 \
    --num_train_epochs 10 \
    --learning_rate 3e-5 --mask_scores_learning_rate 1e-2 \
    --initial_threshold 0 --final_threshold 0.1 \
    --initial_warmup 1 --final_warmup 2 \
    --pruning_method sigmoied_threshold --mask_init constant --mask_scale 0. \
    --regularization l1 --final_lambda 400.
```

L0 regularization

```
python examples/movement-pruning/masked_run_squad.py \
    --output_dir $SERIALIZATION_DIR \
    --data_dir $SQUAD_DATA \
    --train_file train-v1.1.json \
    --predict_file dev-v1.1.json \
    --do_train --do_eval --do_lower_case \
    --model_type masked_bert \
    --model_name_or_path bert-base-uncased \
    --per_gpu_train_batch_size 16 \
    --warmup_steps 5400 \
    --num_train_epochs 10 \
    --learning_rate 3e-5 --mask_scores_learning_rate 1e-1 \
    --initial_threshold 1. --final_threshold 1. \
    --initial_warmup 1 --final_warmup 1 \
    --pruning_method l0 --mask_init constant --mask_scale 2.197 \
    --regularization l0 --final_lambda 125.
```

Iterative Magnitude Pruning

```
python examples/movement-pruning/masked_run_squad.py \
    --output_dir ./dbg \
    --data_dir examples/distillation/data/squad_data \
    --train_file train-v1.1.json \
    --predict_file dev-v1.1.json \
    --do_train --do_eval --do_lower_case \
    --model_type masked_bert \
    --model_name_or_path bert-base-uncased \
    --per_gpu_train_batch_size 16 \
    --warmup_steps 5400 \
    --num_train_epochs 10 \
    --learning_rate 3e-5 \
    --initial_threshold 1 --final_threshold 0.15 \
    --initial_warmup 1 --final_warmup 2 \
    --pruning_method magnitude
```

After fine-pruning

Counting parameters

Regularization based pruning methods (soft movement pruning and L0 regularization) rely on the penalty to induce sparsity. The multiplicative coefficient controls the sparsity level. To obtain the effective sparsity level in the encoder, we simply count the number of activated (non-null) weights:

```
python examples/movement-pruning/counts_parameters.py \
    --pruning_method sigmoied_threshold \
    --threshold 0.1 \
    --serialization_dir $SERIALIZATION_DIR
```

Pruning once for all

Once the model has been fine-pruned, the pruned weights can be set to 0. once for all (reducing the amount of information to store). In our running experiments, we can convert a `MaskedBertForQuestionAnswering` (a BERT model augmented to enable on-the-fly pruning capabilities) to a standard `BertForQuestionAnswering` :

```
python examples/movement-pruning/bertarize.py \
    --pruning_method sigmoied_threshold \
    --threshold 0.1 \
    --model_name_or_path $SERIALIZATION_DIR
```

Hyper-parameters

For reproducibility purposes, we share the detailed results presented in the paper. These tables exhaustively describe the individual hyper-parameters used for each data point.

Inference speed

Early experiments show that even though models fine-pruned with (soft) movement pruning are extremely sparse, they do not benefit from significant improvement in terms of inference speed when using the standard PyTorch inference. We are currently benchmarking and exploring inference setups specifically for sparse architectures. In particular, hardware manufacturers are announcing devices that will speedup inference for sparse networks considerably.

Citation

If you find this resource useful, please consider citing the following paper:

```
@article{sanh2020movement,
    title={Movement Pruning: Adaptive Sparsity by Fine-Tuning},
    author={Victor Sanh and Thomas Wolf and Alexander M. Rush},
    year={2020},
    eprint={2005.07683},
    archivePrefix={arXiv},
    primaryClass={cs.CL}
}
```